UNIVERSITY OF CALIFORNIA SAN DIEGO

Neural-Symbolic Methods For Neural Architecture Design

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy

in

Computer Science

by

Hui Shi

Committee in charge:

Professor Jishen Zhao, Chair
Professor Sicun Gao
Professor Ndapa Nakashole
Professor Nadia Polikarpova
Professor Zhuowen Tu

2023

The Dissertation of Hui Shi is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2023

DEDICATION

To my family, for their unconditional love and support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my advisor, Professor Jishen Zhao, for her unwavering guidance and support throughout my graduate studies. Her passion for groundbreaking works, ability to connect various field of studies, and belief in my potential have been instrumental in shaping my research journey. My sincere gratitude extends to Professor Sicun Gao for his willingness to engage in thought-provoking discussions, insightful guidance on numerous challenging problems, and the encouragement to delve beyond superficial observations and investigate the deeper complexities in the neural network behaviours. His emphasis on critical thinking has instilled in me a passion for unveiling the provable reasons to explain the observation. I'm also grateful to other committee members, Professor Nadia Polikarpova, Professor Ndapa Nakashole, and Professor Zhuowen Tu for their expertise and feedback during the preparation of this thesis.

I'm indebted to my collaborators, Yang Zhang, Yuandong Tian, Xinyun Chen, Yupeng Gu, Yitong Zhou, Hao Wu, Kaizhi Qian, Bo Zhao, Hengyu Zhao, Professor Shiyu Chang, and Professor Mark Hasegawa-Johnson, for their contributions to our research projects. Their dedication and expertise were invaluable assets to this research project, and have significantly enriched my understanding of the machine learning research. I would also like to thank my peers and colleagues in our lab and Computer Science and Engineering department for their stimulating discussions. The collaborative environment is always a source of inspiration and motivation.

I am immensely grateful to Professor Pin-chao Liao for being the catalyst that sparked my interest in machine learning. He graciously provided me with invaluable research opportunities to apply the machine learning techniques and contribute to meaningful projects. His years of encouragement and support bolstered my confidence and propelling me forward in my pursuit of doctoral study.

Finally, I would like to thank my family and friends for their love and encouragement. Their belief in me has been my guiding light throughout this journey.

Chapter 3, in full, is a reprint of the material as it appears in "Learning bounded context-free-grammar via LSTM and the transformer: Difference and the explanations.", Shi, Hui, Sicun Gao, Yuandong Tian, Xinyun Chen, and Jishen Zhao. In Proceedings of the AAAI Conference on Artificial Intelligence, vol. 36, no. 8, pp. 8267-8276. 2022. The dissertation author was the primary investigator and author of this paper.

Chapter 5, in full, is a reprint of the material as it appears in "Continuous cnn for nonuniform time series." Shi, Hui, Yang Zhang, Hao Wu, Shiyu Chang, Kaizhi Qian, Mark Hasegawa-Johnson, and Jishen Zhao. , In ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 3550-3554. IEEE, 2021. The dissertation author was the primary investigator and author of this paper.

Chapter 4, in full, and Chapter 7, in part, are a reprint of the material as it appears in "Everyone's Preference Changes Differently: Weighted Multi-Interest Retrieval Model." Shi, Hui, Yupeng Gu, Yitong Zhou, Bo Zhao, Sicun Gao, and Jishen Zhao. In International Conference on Machine Learning. PMLR, 2023. The dissertation author was the primary investigator and author of this paper.

Chapter 6, in full, Chapter 7, in part, and Chapter 8, in part, are a reprint of the material as it appears in "Deep symbolic superoptimization without human knowledge." Shi, Hui, Yang Zhang, Xinyun Chen, Yuandong Tian, and Jishen Zhao. In International Conference on Learning Representations. 2019. The dissertation author was the primary investigator and author of this paper.

VITA

2014	Bachelor of Science, Tsinghua University
2015	Master of Science, University of Illinois at Urbana-Champaign
2018–2023	Graduate Student Researcher, University of California San Diego
2023	Doctor of Philosophy, University of California San Diego

PUBLICATIONS

"Deep symbolic superoptimization without human knowledge." Shi, Hui, Yang Zhang, Xinyun Chen, Yuandong Tian, and Jishen Zhao. In International Conference on Learning Representations. 2019.

"Learning bounded context-free-grammar via LSTM and the transformer: Difference and the explanations.", Shi, Hui, Sicun Gao, Yuandong Tian, Xinyun Chen, and Jishen Zhao. In Proceedings of the AAAI Conference on Artificial Intelligence, vol. 36, no. 8, pp. 8267-8276. 2022.

"Everyone's Preference Changes Differently: Weighted Multi-Interest Retrieval Model." Shi, Hui, Yupeng Gu, Yitong Zhou, Bo Zhao, Sicun Gao, and Jishen Zhao. In International Conference on Machine Learning. PMLR, 2023

"Continuous cnn for nonuniform time series." Shi, Hui, Yang Zhang, Hao Wu, Shiyu Chang, Kaizhi Qian, Mark Hasegawa-Johnson, and Jishen Zhao. , In ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 3550-3554. IEEE, 2021.

ABSTRACT OF THE DISSERTATION

Neural-Symbolic Methods For Neural Architecture Design

by

Hui Shi

Doctor of Philosophy in Computer Science

University of California San Diego, 2023

Professor Jishen Zhao, Chair

Modern neural architectures present strong performance on various tasks, but their abilities are compromised when faced with the tasks requiring robust reasoning or involving abstract concepts. This dissertation starts with a study of the empirical representation power of two popular sequential neural networks, Long Short-Term Memory (LSTM) and Transformer network, in learning context-free grammar. The experiment results exhibit that the performance of both models is affected by the types of supervision and model regularization setting. Specifically, the LSTM and the Transformer network struggle to capture the grammatical structures, particularly the grammars with deeper recursions, when solely trained on input sequences in the generative language model setting. The performance gap between settings suggests two improvements when

the input data structure is available. One approach is to supervise the model with intermediate states, while another is to regularize the models' hidden states to enforce the its decomposition to store the state and to simulate the stack, mimicking a pushdown automaton, on separate segment.

Complementing the study on sequential neural networks with prototypical grammars, several real-world applications are presented to showcase how the controlling the model complexity and intermediate state can enhance the performance. For the item retrieval problem in recommendation system, another transformer-based sequential model is proposed to capture the multiple interests of a user. Unlike the existing multi-interest frameworks, the proposed model is unbiased towards the site-wide popular items at the cost of incorporating a non-differentiable clustering method inside the neural network. The non-differentiable challenge are solved by two-stage training method, borrowing the idea from K-means clustering algorithm. In time series prediction tasks, a continuous convolutional neural network (CCNN) is proposed to handle non-uniformly sampled time series without preset interpolation kernels. CCNN can directly apply to the auto-regression tasks, and also works with temporal point process front-end to predict the time interval to the future event in the event-based time series. Lastly, a progressive optimization system, HISS , is designed to simplify the expressions in the intermediate representation of Halide compiler. HISS is built end-to-end without any human knowledge of the mathematical equivalences. Multiple training techniques, existing symbolic transformations of the expressions, and the evolving neural network modules, contribute to the HISS 's performance that exceeds conventional searching method and human-heuristic methods. These three applications illustrate the benefits and techniques of applying the aforementioned philosophy, which suggests designing and supervising the neural networks to reflect the desired properties of neural networks.

In conclusion, by presenting an empirical study method for evaluating the representation power of neural networks, which can be extended to broader areas, and the demonstrating how to design and overcome the challenges associated with symbolic properties in various real-world applications, this dissertation contributes to the advancement of the neural symbolic systems.

Chapter 1

Introduction

In recent decades, the artificial intelligence (AI) has undergone remarkable breakthroughs.These advancements emerge from a confluence of refined machine learning algorithms, especially the proliferation of neural network, Moore's Law's exponential growth of computation power, and the ever-expanding data source. This had led to the rapid development of powerful AI models that continuously expand its capability boundary and can be deployed on various edge devices. As a result, from recommending contents to the users' taste to aiding automobile driving, AI has permeated various aspects in our daily life.

In the earlier years, a focus on the machine learning was developing expert systems that could mimic decision-making process of human experts. For example, Bayesian networks is consist of nodes with physical meanings and edges between nodes to denote the causal relationship. The network is specified by both the network structure, *i.e.* the graph topology, and the probabilistic models, *i.e.* the probability distribution of value of each node given the value of its parents nodes. Both the topology and probabilistic model can be learned from data or defined by human experts, and vice versa, the learned topology and probabilities can be interpret and examined by human seamlessly. Therefore, the Bayesian networks are widely used in identifying and measuring the causal relationship between many factors [4, 97]. Similar examples include logistic regression model, decision trees, Markov decision process, and etc. In more complicated systems, it was also a paradigm to build the interpretable probabilistic models

with human knowledge, and learn the distribution parameters from data. Canonical examples are hidden Markov models for part-of-speech tagging in natural language processing (NLP) [89] and Markov random field for image segmentation in computer vision (CV) [14].

Despite the merit of interpretability, the applications of expert systems are narrow owing to tremendous design effort and lack of generalization. On the other branch, with the advancement in neural network and chip manufacturing, the era of deep learning began. In 2008, the one of the first handwriting recognition neural network was proposed [54], and three years later, the neural network recognizer exceeded the human performance [34]. While the success in handwriting recognition was known to a specialized group of people, the accuracy of AlexNet [88] of recognizing thousands of objects surprised general public in 2012, and ever since then, the Convolutional Neural Networks (CNN) have been continually improving in accuracy and efficiency [59, 68, 119, 133, 141]. Meanwhile, in 2016 and 2017, AlphaGo won two of the worldwide top human players in Go, the traditional game that was believed to be one of the most difficult game for computers to defeat human. The victory of AlphaGo announced a new era of AI, where the AI learned from the past playing experience to master the one of the most complicated and strategic game and beat the best human player. The AlphaGo was retired after its pinnacle era, and just one year later, its successor AlphaZero achieved a new record of 100 to 0 victory against AlphaGo. Moreover, the AlphaZero was developed fully by self-teaching. In the same year, BERT [41] was released and greatly elevated the AI capability to a near human level in natural language processing. Sharing the same basic blocks with BERT but wider and deeper, ChatGPT was launched in 2022 and is available to general users. Its ability of interaction with human and perform various tasks makes people realize that general artificial intelligence is close to reality for the first time.

While neural networks were inspired by the human nervous system, their behavior remains largely enigmatic to humans. For instance, convolutional neural networks (CNN) was designed with the intuition that the initial convolutional layers should capture the low-level vision features, such as edges and textures, and the later layers should combine the low-level

features to high-level features, *e.g.* the entire shape of object or the structure of a scene. The remarkable progress of convolutional neural networks in object recognition, even surpassing human performance, seemed demonstrating the success of the design, and was initially met with great enthusiasm. However, it was soon discovered that these networks often relied on unexpected features, such as textures, rather than the more obvious shape and topology that humans use for classification [51]. This was less considered a problem until the neural network attacking became the evidently feasible. Attackers can effectively alter the CNNs' prediction by adding low energy noises to images, which almost won't change human judgement [158, 170]. While not yet a reality, this drawback could expose AI-aided systems to the risk of silent corruption, going unperceptive by humans. Not only in the perception, AI also generates contents that not following the human's common sense. One of the widely used AI production tool, AI Generated Content (AIGC), could generate high-quality artistic or realistic images, usually fails to correctly generate hand postures and structures [2]. Instinctively, it's interpreted that the intricate structure and versatility of human hands, enabling them to form a vast array of postures, make them more challenging to learn than body postures. However, due to the opaque natural of neural networks reasoning process, it's hard to ascertain if the neural network output adhere to certain property, for examples, each hand should have five fingers with well-defined constraints of the movement. Moreover, it also hinders efficient improvement of the model behaviour in specific cases.

In additional to their vulnerability to attacks, neural networks face challenges of efficient reasoning. A famous example is the hardship of neural networks to distinguish even and odd integers based on numerical value instead of its string form. Although fundamentally the neural networks are a graph of mathematical operations, they often exhibit shortcomings in mathematical reasoning. Evidences also illustrates that neural network are yet unable to count the object or to compare the attributes of objects with human-level accuracy, such as shapes and materials, even in narrowly defined tasks [74]. The deficiency also manifest in document processing, in which the structures over the textual and image elements are critical and certain extend of calculation and reasoning are essential. For instance, identifying the subtotal of certain

category on a scanned receipt requires matching the amount with entry names and category, and then filter by category and compute the total amount [105, 127, 134]. Also in the area of computer vision, while human can recognize the objects correctly regardless of the size and direction of the object, the neural networks usually being sensitive to transformations and scaling [29, 36]. In the NLP domain, the powerful models like ChatGPT also have hardship to precisely extract facts from input and conduct multi-step reasoning [24, 162].

The uninterpretable behavior, vulnerability, and difficulty in reasoning are symptoms of the same issue: neural networks lack symbolic thinking. To alleviate these, the researchers are pursuing two primary directions of exploration: deep learning and rule-based systems. The former emphasizes on improving training method, increasing the training data, and enlarge the model size, while the later insists on including rule-based logic inside the neural networks or vice versa. Since 2018, a lively debate among renowned AI researchers on the two directions has unfolded on social media. On on side, championed by Yann LeCun, a group of researchers claim that artificial general intelligence (AGI) will not be realized without deep neural networks, and possibly deep neural networks could be a standalone solution. In the other camp, Gary Marcus and other researchers disagree as the low interpretability and the *de facto* limitations of deep learning models, and advocate for rule-based systems. The debate has diminishes but never faded. While the newer and more powerful deep neural network models comes to life, it reinforces both arguments for its growing ability and the persistence of shortcomings in symbolic reasoning.

Despite the debate, the researchers assent to neural symbolic AI direction, which emphasizes the combination of symbolic system or attributes and the deep learning. The approach attempts to marry the strength of symbolic system in logic and reasoning and the advantage of neural networks in generalization. Although conceptually straightforward, the design of neural symbolic AI requires careful consideration on a case-by-case basis. Given the ubiquity of reasoning in machine learning tasks, neural symbolic AI intersects with a broad spectrum of subareas of machine learning, including natural language processing and computer vision, and fields beyond machine learning community, such as computer architecture, programming

languages, and optimization.

This dissertation delves into the intricacies of reasoning hardships of neural networks in a few scenarios, and introduces the common design paradigms through successful specialized neural symbolic systems. This chapter starts with the observations of failure cases of neural networks in various scenarios, where the data distribution is beyond the capacity of the neural networks (Section 1.1.1) or the neural networks struggles to achieve their theoretical capacity (Section 1.1.3 and Section 1.1.2). Then Chapter 2 briefly introduces the aspects in neural symbolic AI, including the common approaches and components, the representation power of neural networks, and specialized neural architecture design for structured input data. In Chapter 3, the formal analysis of the representation power of sequential models are presented, and concludes that without reinforcement the models behavior with supervision on the intermediate state with additional labels, the models could not achieve the theoretical power of simulating context-free languages. Not only for sequential models in learning context-free languages, the model regularization and intermediate supervision are powerful tools to improve the neural networks in other tasks. Chapter 4 demonstrates the conclusion via an item retrieval task in recommendation system: a joint multi-interest encoding module and a preference module is designed but trained with two stages to enhance the supervision on both modules. In Chapter 5, a non-uniform time series scenario fails the canonical CNN models and a continuous convolutional layer is proposed. This work illustrates the approach to incorporate the symbolic operation inside the neural networks. Chapter 6 presents a specialized neural symbolic system to perform expression simplification inside the compilers. The sophisticated system is consist of neural encoder, selector for decomposing the problem, and the decoder to explore the simplification possibilities, and has multifaceted interactions with string operation utilities, knowledge base, and symbolic executors. Lastly, the discussion on the training techniques and interactions with peripheral systems is continued in Chapter 7 and Chapter 8.

1.1 Challenges and Limitations of Neural Networks

Machine learning is dedicated to developing algorithms that can learn from data and make predictions or decisions without need to fully specify the process by human. The fundamental goal of algorithms is to discover effective and efficient parameterized models $f_\theta(X)$ that transforms input data X into the corresponding label or outputs y. The model parameters θ is typically obtained by minimizing the difference between predicted output $\hat{y}$ and the true observation y in the supervised learning, or by minimizing some heuristic metric in unsupervised learning. As indicated, the problem is duplex: 1) the parameterized model or hypothesis $f_\theta(\cdot)$ should be powerful enough to approximate the ground truth mapping from input to output, and preferably $f_\theta(\cdot)$ should consume reasonable computation power; 2) the data and training method could lead to set of parameters θ^* with which the approximation error of $|f_\theta(X) - y|$ is satisfying.

One of the compelling advantage of neural networks is their ability to represent complex mappings. For instance, convolutional neural networks can easily learn the kernels to extract various visual features solely from training data without human's knowledge. Moreover, as the neural layers stack, the representation power broadens. Despite the appealing aspect of end-to-end learning, the question of how effectively the neural network can capture the complex underlying relationship remains an open area of research. The neural networks' performance is contingent upon, similar to aforementioned, both the soundness of the neural architecture design and the effectiveness of model optimization approach.

In support of the proposition that neural network are not always capable to represent the desired relationship or could be trained to achieve the approximation, a few failure cases are presented below. Building upon the insights gained from the failure analysis, the subsequent chapters will detail the design of enhanced architecture tailored to fulfill the tasks.

1.1.1 Learn Non-uniform Time Series with Convolutional Neural Network

Time series, as a prevalent type of model input, encompasses a range of tasks, including classification, future value forecasting, next occurrence interval prediction. Convolutional neural network (CNN) and recurrent neural network (RNN) are two popular neural architectures to process time series data. Conventionally, RNNs are the dominant option for their flexibility of handling arbitrary long input sequence with small amount of parameters. After the success of WaveNet [110] on audio data, processing time series with CNN became popular on long series, as the computation of CNN can be better paralleled and thus has lower latency. However, the vanilla convolution operator and the variants of RNNs, including vanilla RNN, gated recurrent unit (GRU), and LSTM, all assume the input and the output time series are temporally uniformly sampled. In other word, the time interval between each two successive data points should be a constant. However, this assumption can be easily violated in the real world. Numerous time series dataset originate from discrete events rather than continuously monitored signals, or uniformly sampling from the signals is impossible. For instances, stock transactions [52], social media blogs [21], and health care records [73].

The non-uniform sample rate potentially introduces significant error into the discrete convolution, a building block of CNNs, on the sequence of observation. Figure 1.1 compares the output of convolution under uniform and non-uniform sampling scenario given the same signal and discrete convolution kernel. The signal is a sine function over time, and the demonstrative kernel C is $[0,1,0,-1,0]$. When the kernel is applied to the uniformly sampled observations (inputs), the outputs (prediction) still forms a sine function but with some non-zero phase shift. On the contrary, if the observations are non-uniformly sampled, the outputs are no longer on the ground truth sine wave, and the difference to the true values could be large.

To handle non-uniform time series by neural networks, besides the observation values, the timestamp of the observations are necessary. In principal, CNN and RNN can learn the

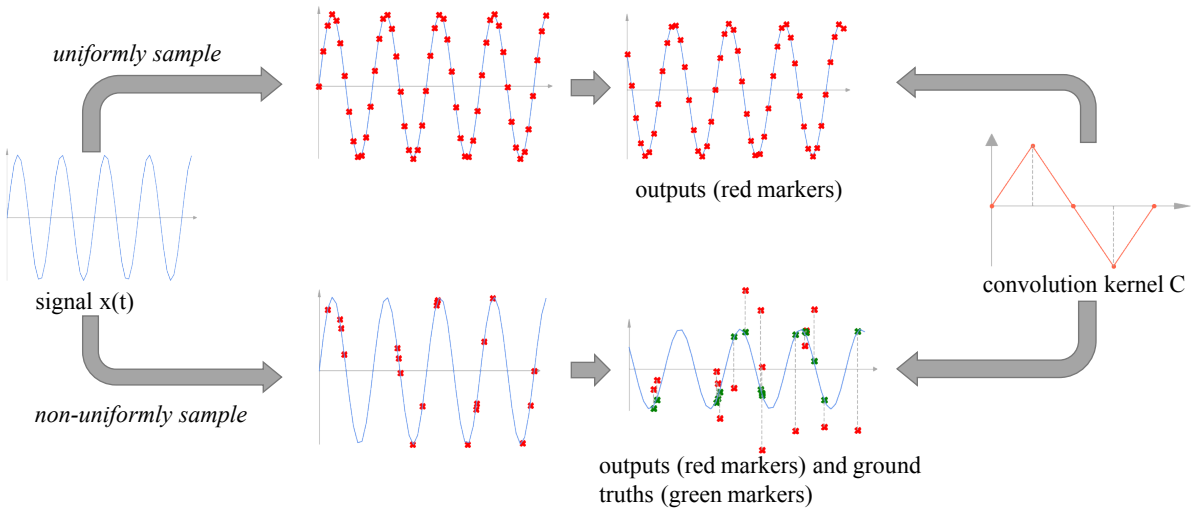


Figure 1.1. Illustration of discrete convolution on non-uniform sampled time series.

non-uniform time series with arbitrary temporal dynamics by feeding the timestamps with observation, given adequate kernel size in CNN, hidden units in RNN, and number of layers. However, the straightforward approach won't work well. A simple experiment can show how CNN and RNN under uniform sampling assumption are easily failed by nonuniform time series, and it's also hard for both of them to utilize the timestamps directly.

In this simple experiment, a series of observations $x(t_1)$, $x(t_2)$, ..., $x(t_n)$ is sampled from a sine function $x = sin(\frac{2\pi t}{T})$, where the period $T = 5$, and the target is to predict $x(t_{n+1})$. During sampling, $t_1, t_2, ..., t_{n+1}$ are monotonically increasing, and each interval $[t_i, t_{i+1}]$ is randomly sampled from a Poisson process with mean of 1. The input length n is 13, and the models are informed timestamp of the value to be predict t_{n+1}. A family of CNN approaches are compared: vanilla CNN, CNN-T (T stands for timestamps), and ICNN (I stands for interpolation). Figure 1.2 illustrated the architecture of CNN and CNNT. They both have two standard convolutional layer, the kernel lengths of which on the temporal dimension are the same. CNN only takes the observations as input, while CNNT takes observation and timestamps as 2D input. We set the kernel length on the temporal dimension to be 7, so that each kernel should be able to see a complete period of the sine wave, and by two convolution layers without padding, the final

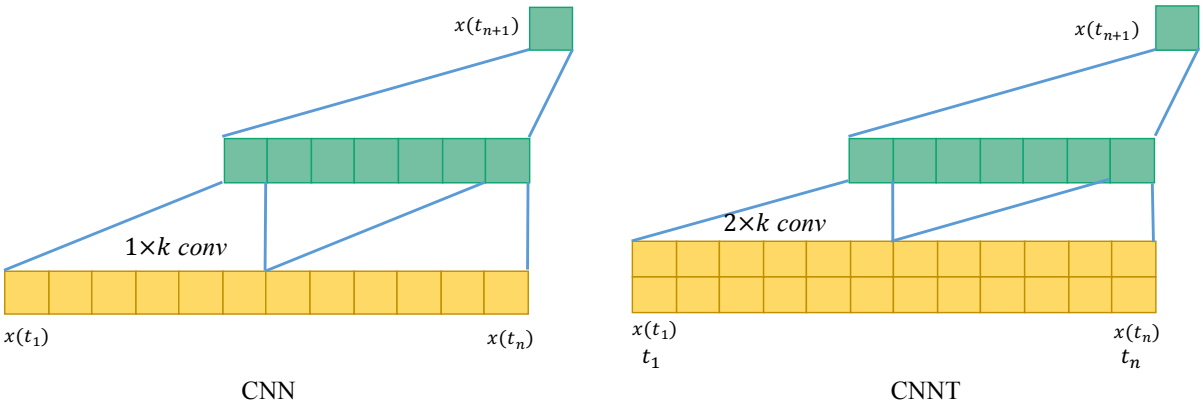


Figure 1.2. CNN and CNNT architecture for time series prediction.

output dimension on is 1. The final output is the $x(t_{n+1})$. Additionally, one common practice in handling non-uniform time series is to interpolate the observations to a uniform time series and then feed into the CNN. In the results below, the cubic interpolation method is used.

In the experiment, the mean squared error (MSE) for the models are: 0.46 for CNN, 0.20 for CNNT, and 0.007 for the ICNN. Apparently, vanilla CNN models can hardly handle the non-uniform time series. Nevertheless, in the Chapter 5, the evidences are shown that better neural architecture can be designed to exceed the performance of interpolation methods and to be applied to more universal scenarios.

1.1.2 Learn Significance of Embedding

Embedding vector, a representation of arbitrary input in the high-dimensional latent space, is a fundamental concept in deep learning realm. For example, the meaning of a word can be encoded into a vector space [108], and an image can be embedded into a vector [49]. In recommendation systems, a user can be represented by single vector and be compared with item embeddings to predict if the user would be interested in the item. Usually, the mode (length) of the embedding vector is less concerned as the direction of the vector. However, in a recent study, where the users are represented by multiple embeddings and the mode of the embeddings represents the significance of the embedding, it's observed that a monolithic model could hardly

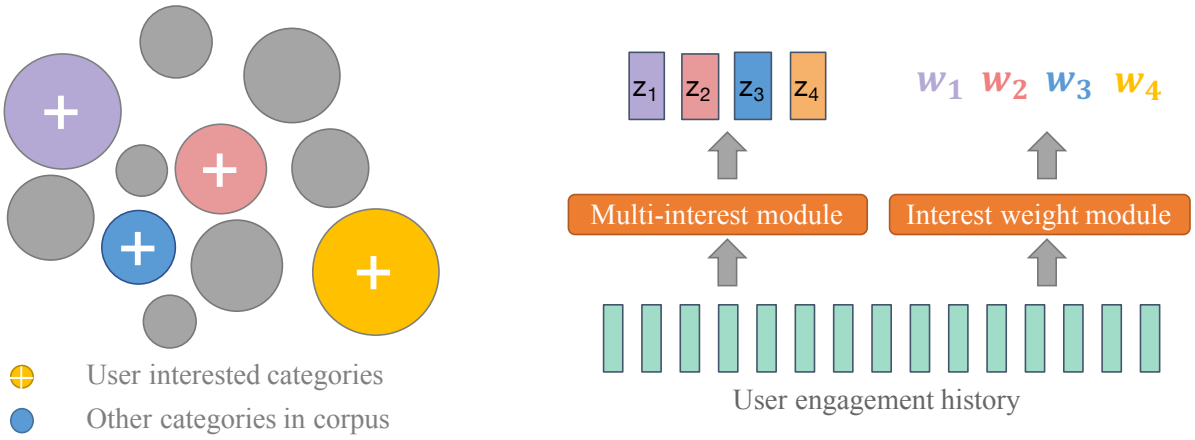


Figure 1.3. Multi-interest representation of user and weighted multiple interest neural network.

learn the embedding vectors and its significance at the same time.

Formally, the user-item interaction prediction, in the single user embedding case, is to compute the a score $y = \boldsymbol{z} \cdot \boldsymbol{p}$, where $p \in \mathbb{R}^d$ is an item embedding vector and $\boldsymbol{p} \in \mathbb{R}^d$ is the user embedding. When the user is embedded into multiple embeddings $[\boldsymbol{z}_1, ..., \boldsymbol{z}_\Lambda]$, the scoring function can be re-written as:

$$y = max_\lambda (\boldsymbol{z}_\lambda \cdot \boldsymbol{p}) \tag{1.1}$$

Furthermore, it's reasonable to assume the users has preferences over their interest, and the scoring function should consider not only the item-category affinity but also the user-category preference. Let the positive scalars $w_1, ..., w_\lambda$ indicate the strongness of user's interest in each category $\boldsymbol{z}_\lambda$. Correspondingly, the scoring function can be extended to consider weighted multiple interest:

$$y = max_\lambda (w_\lambda (\boldsymbol{z}_\lambda \cdot \boldsymbol{p})) \tag{1.2}$$

The rationale of the weighted multi-embedding is left in later chapters, and now the discussion focuses on learning the w_λ and $\boldsymbol{z}_\lambda$.

Despite the physical meaning differs, equation 1.2 can be rewritten as $y = max_\lambda ((w_\lambda \boldsymbol{z}_\lambda) \cdot$

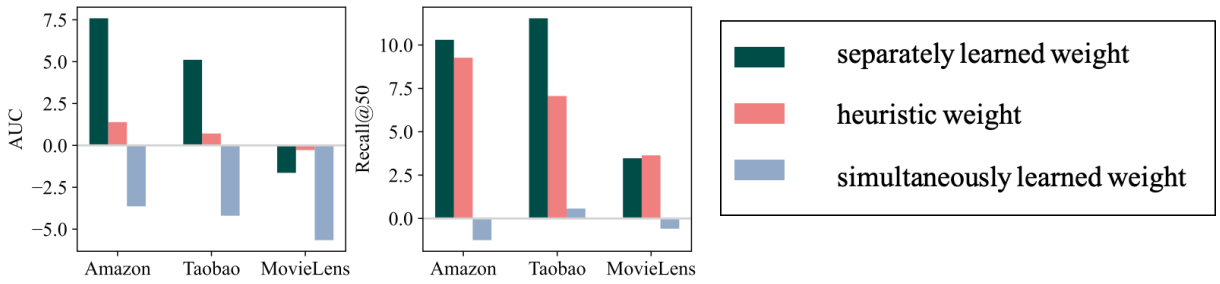


Figure 1.4. Performance comparison of explicitly model the embedding significance, implicitly modelling, and using heuristic methods.

$\boldsymbol{p}$). Mathematically, scaling the similarity score is equivalent to scaling the embedding vector $\boldsymbol{z}_\lambda$. Since the learned $\boldsymbol{z}_\lambda$ is commonly not normalized and can have arbitrary mode, it seems unnecessary to include w_λ in the scoring function when the $\boldsymbol{z}_\lambda$ is learned and not normalized. In the ablation study below, the intuition is challenged.

In the ablation study, the retrieval problem is formulated as Equation 1.1 for a monolithic model, and as Equation 1.2 for a modularized model. The input to both models is the user engagement history that is consist of the item embeddings $\boldsymbol{p}_1, ..., \boldsymbol{p}_l$ and timestamps $t_1, ..., t_l$ of user's engagement actions. A two-layer transformer network (detailed in Chapter 4) is used to compute $\boldsymbol{z}_1, ..., \boldsymbol{z}_\Lambda$. A standalone module will take the timestamps and learned $\boldsymbol{z}_\lambda$ to further predict w_λ. The performance comparing implicitly embedding significance (without the standalone weight module) and explicit learn the embedding weights are shown in Figure 1.4. Experimental results imply that simultaneously learning the embedding vectors with proper lengths are challenging.

1.1.3 Learn Context-Free Languages with Sequential Models

Long-Short Term Memory, as well as the transformers, are believed to be capable of simulating arbitrary Turing machine, and thus should be able to recognize many sequential patterns. However, LSTM and the transformer can be easily failed by a simple case named *parity*: deciding if there are odd number of 0s in a bit string. Despite similarity to the famous challenge that neural network could hardly classify odd number from even numbers, if the input

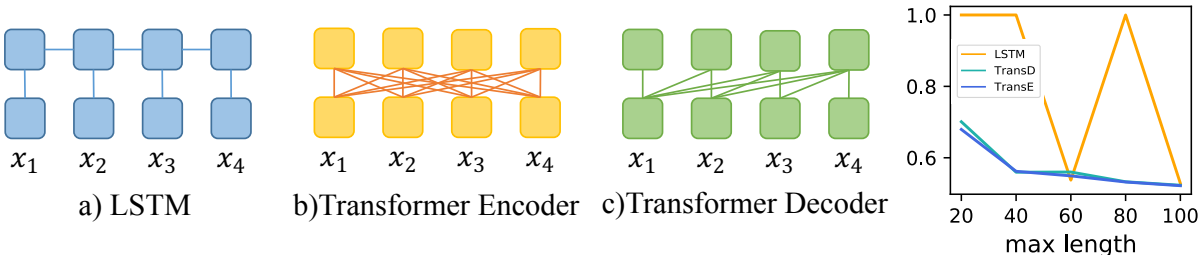


Figure 1.5. LSTM, Transformer encoder and decoder, and their classification accuracy on parity.

number format is numeric, the task of parity is actually simpler, a simple solution to simulate is to maintain an binary internal state, and whenever the next bit in the sequence is 0, flip the state. Then the final state is exactly the classification results. The solution seems straightforward and within sequence neural networks' capability. The LSTM and transformers are challenged with this classification task.

In the experiment, the task is modeled as a classification task. For the maximum sequence lengths of 20, 40, 60, 80, and 100, a non-duplicated set of sequences up to the length of maximum lengths are generated. The number of sequences is the minimum of all possible combinations or 100,000, and split 1:1 to training and testing set. Three model architectures are compared, shown in Figure 1.5(a-c). The models are trained with binary cross entropy loss. The results are compared in Figure 1.5(d). The results clearly show that 1) transformers barely can learn the pattern; 2) LSTM sometimes might converge to a solution that perfectly learn the state flipping, but the convergence to the optimal solution is not guaranteed.

1.2 Summary of Contributions

This dissertation contributes an empirical study of representation power of modern sequential neural networks, and a few neural symbolic applications.

1.2.1 Representation Power Analysis of Sequential Models

The study connects the modern sequential neural networks, Long Short-Term Memory (LSTM) network and Transformer networks, with context-free grammar, the expressiveness of

which is comparable to most of the programming languages and natural languages. The study first show that given proper model regularizer, *i.e.* forced decomposition of their latent space, and full supervision with state and stack status of the corresponding Pushdown automaton to the Context-Free language, two models perform similarly, and LSTM even outperforms the Transformer model in simulating the state machine and operating on the stack. Then ablation studies explain how the absence of regularizer and full supervision affects the performance difference of the models in reality: 1) LSTM hardly learns to factorize its latent space to encode both state and stack without force decomposition regularizer, while the Transformer experience marginally performance drop; 2) full supervision helps LSTM in language recognition but hinders the Transformers. Then, we connect the synthetic studies to a parsing task and show the results on the synthetic PDAs are consistent with real-world tasks. Lastly, the experiments conclude that the empirical performance difference of the two models comes mainly origin from LSTM's failure to decompose its latent space to simulate the stack.

1.2.2 Continuous CNN for Non-Uniform Time Series

This dissertation presents Continuous CNN (CCNN), a generalization to CNN for nonuniform time series, to serve as the backend of the neural-based temporal point process (TPP) model. CCNN estimates the implicit continuous signal on the fly performing continuous convolution on the continuous signal. As a result, CCNN is capable of capturing useful patterns in the implicit input signal under the nonuniform sampling rate. Unlike existing interpolation method, the neural estimation of the implicit continuous signal does not rely on a pre-set interpolation kernel, but rather learned in an end-to-end manner, so that the misspecification error can be avoided. As shown in Section 5.6, CCNN can achieve much better performance than the state-of-the-art systems on non-uniform time series data.

1.2.3 Personalized Multi-Interest User Embedding Model with Preference

The main contribution of the Chapter 4 can be summarized as follows: 1) proposes a methodology that utilizes the sequential engagement history to find multiple embedding for a user, each of which represents the user's interest in a certain category. These representations will be used in the retrieval task in recommendation systems, which we find successful in various industry-scale datasets; 2) in addition to the multi-facet vector representations of a user, MIP assigns weights to each embedding, which is automatically customized for each user interest, and improve the recall of candidate generation by retrieving more candidates from the most representative embedding; 3) MIP do not require any prior knowledge on the number of categories per user, nor does MIP assume it to be the same among all users. The number of meaningful categories can be trivially modified post training process.

1.2.4 Symbolic Expression Simplification without Human Knowledge

Chapter 6 proposes Human-Independent Symbolic Superoptimization (HISS), a reinforcement learning framework for symbolic superoptimization that is completely independent of human knowledge. Instead of using human-defined equivalence, HISS adopts a set of unsupervised techniques to maintain the tractability of action space. First, HISS introduces a tree-LSTM encoder-decoder architecture with attention to ensure that its exploration is confined within the set syntactically correct expressions. Second, the process of generating a simplified expression is broken into two stages. The first stage selects a sub-expression that can be simplified and the second stage simplifies the sub-expression. A set of evaluations is performed on artificially generated expressions as well as a publicly available code dataset, called the Halide dataset [26], and show that HISS can achieve competitive performance. The results also find out that HISS can automatically discover simplification rules that are not included in the human predefined rules in the Halide dataset.

Chapter 2

Background

2.1 Neural Symbolic Approaches

Neural symbolic systems represent a multifaceted field in the artificial intelligence striving for robustness in recognition and reasoning tasks. As its name implies, neural symbolic AI deftly blends the strength of neural networks with symbolic system,. The fusion results in a diverse array of applications ranging from neural program synthesis to robotics. The symbolic systems, meticulously designed by human experts and evolving over the years, exhibit remarkable efficiency and reliability. However, migration these systems to new domains often demands significant effort. This inflexibility highlights the inherent advantage of neural networks, which possess the ability to learn from data with minimal human knowledge or from system feedback. A prime example is the neural-guided searching. After years of research, generic searching methods have been developed to address program synthesis problems. Recently, researchers have discovered that utilizing neural network to identify the most promising direction within the search space offers significant benefits over following the human-defined heuristic searching order [77, 172]. Meanwhile, the incorporation of neural network and symbolic method varies owing to versatile symbolic nature of each application. Detailed below, there are standalone neural networks perform symbolic tasks without explicit symbolic components.

Despite that neural symbolic systems are usually different case by case, distilling the common system architecture and summarizing the techniques to support the system accordingly

remains crucial. In this dissertation, the paradigms proposed by Kautz [79] serve as a framework for understanding the neural symbolic approaches. These paradigms are explained as follows and visualized in Figure 2.1.

- **Symbolic neuro symbolic** is the most common workflow in the machine learning application. The input and output are in the symbolic domain, and a monolithic neural network is computing the output from the input. There is no explicit interactions with symbolic systems. Such works are widely seen in the applications where the input to output mapping rules are implicit and various, *e.g.* recognizing handwriting characters [34, 54], translating between natural languages [8, 101, 155], and segment the objects from images [7, 25, 121, 165]. The symbolic counterpart of symbolic neuro symbolic methods are usually takes huge human effort to build the rule base and can not easily scale or update. For instance, the machine translation system before deep learning era is usually consist of *translate*, a parameterized probabilistic model of word or phrase A mapping to word or phrase B, and *align*, another parameterized probabilistic model to adjust the word order in the target language. Human knowledge is required to build the two probabilistic models, and meanwhile the parallel dataset is used to learn the parameters in the probabilistic models [84].

- **Neuro:symbolic** $\rightarrow$ **neuro** is similar to the symbolic neuro symbolic approach in a way that output is directly generated by neural network given the input, but different in that symbolic neuro symbolic methods relies on human labeled (or human calibrated) parallel dataset, while the input-output of neuro:symbolic $\rightarrow$ neuro is verified or generated by an existing symbolic system. The emphases on this approach are 1) demonstration of capability of neural networks in simulating a complicated symbolic system; and 2) exploring the efficiency of neural network in solving symbolic problems comparing to the underlying symbolic system. A few examples can be found in mathematics applications [23, 91].

- In **neuro** $\mid$ **symbolic** approaches, the neural networks generates intermediate results that

can be accepted by a symbolic system, then the symbolic system will generate the final output. Contrary to the believe in neuro:symbolic $\rightarrow$ neuro, this approaches believes the neural networks are better in extracting the logic from the input than executing the logic by themselves. For instance, it's famous that neural networks could hardly distinguish odd numbers and even numbers, if they are in the numeric format instead of string format. In the similar applications, where the execution of the logic is challenging for the neural network, neuro | symbolic approaches are appropriate. Despite the odd and even number problem is famous, the common use cases of neuro | symbolic are query language applications, *e.g.*, find the number of certain types of object in visual question answering (VQA) [103], retrieve the information from tabular data [61]. Follow the definition in [61], when the direct output of the neural network is available in the dataset, the task is *strongly supervised*, *e.g.* the semantic parsing datasets that contains SQL query of each input question [167, 175] can train the neural networks in the supervised manner; when the direct output is absent in the dataset, which is more usually the case due to the labelling cost, the task is weakly supervised. In the weakly supervised tasks, the reinforcement learning, and some times efficient searching methods, must be leveraged to provide the signal to train the neural network, since the symbolic systems are usually non-differentiable.

- **Symbolic[neuro]** approach employs a neural network as an integral component within a symbolic system. A prominent application is AlphaGo, the AI Go player. AlphaGo compute the next action via a high-level Monte-Carlo tree search solver augmented by a neural network that evaluate the value of any possible future board state. Actually, symbolic[neuro] is the most widely used approach in the robotics industry, since the high level controls are still symbolic, while the lower level components are well developed neural networks, *e.g.* object recognition, instruction understanding.

- In contrast to symbolic[neuro] approach, **neuro[Symbolic]** envisions neural network

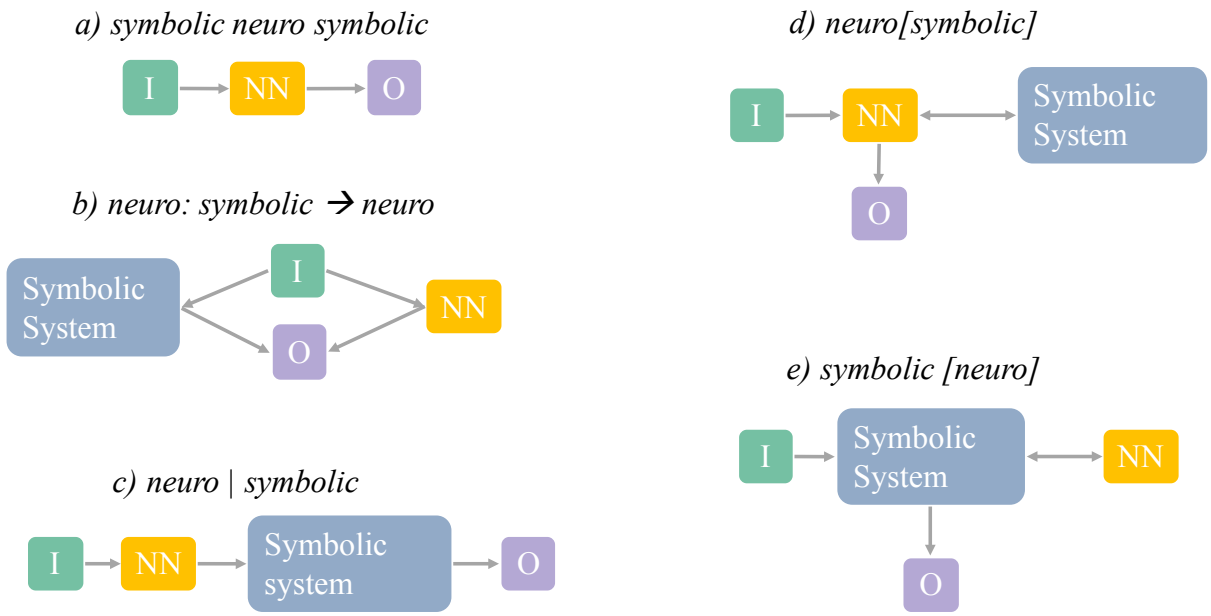


Figure 2.1. Neural symbolic AI approaches. **I** stands for input data, and **O** stands for output data. **NN** stands for neural networks.

taking control at the high-level. If there are some subroutines found to be particularly challenging for neural networks, neural network can invoke symbolic functions to obtain the results. Unlike the neuro | symbolic approach, neuro[symbolic] aims to establish multi-step interaction between the symbolic system and the neural network.

- **Neuro_{Symbolic}** approach, with relatively lower popularity, directly encodes the symbolic rule templates within the neural networks. The methods are usually limited to non-combinatorial problems.

2.2 Neural Symbolic AI Component

In Kautz's categorization, the approaches differ in **representation**, **logic**, and **interaction**. Understanding these three aspects of each paradigm helps researchers to select the most promising approaches according to the problem natures. Generally speaking, the data structure decides the type of neural networks. Then the representation power of neural networks, logical complexity of

the problem, and the desired neural network - symbolic system interaction lead to the appropriate neural symbolic approach, as summarized in Figure 2.2.

2.2.1 Representation

Representation is the process of transforming the input, usually in a human comprehensible form, to the latent vectors or matrices that the neural network could operate, and also the reverse process of generating the output from latent vectors. The ability of model to preserve the useful information and make necessary transformation from the input to the output is referred to as representation power of a neural network. Theoretically, the modern neural architectures are proven to have adequate representation power, if deep and wide enough. However, given the same number of layers and number of parameters, some neural networks are outperforming others, owing to more efficient neural network architecture.

When choosing a neural network for a certain task, the input and output format, or structure, is always a first consideration, and is also the first direction when improving an existing generic neural network. In the symbolic systems, *e.g.* compiler and solvers, the format of the input must follow the strict grammar to be parsible to intermediate form, *e.g.* intermediate representation (IR) or the abstract syntax tree (AST). On the contrary, the neural network accepts for generic input formats: feed-forward neural networks can accept any fixed length input and produce fixed-length output; CNN can accept almost arbitrary input size, and the output size will depends on the input size; RNN can accept arbitrary long sequence, and produce arbitrary long sequence independently to the input length. The Transformer network, originally was proposed in NLP tasks to as a sequence-to-sequence type model, can also handle image data as well.

2.2.2 Logic

Logic is the operation or the transformation space that defines the process to generate the output from the input, in most case, in a deterministic way. Roughly speaking, there are *strong-logic* tasks and *weak-logic* tasks. In the strong-logic tasks, the operations or the

transformations are usually well defined and the process has high-order. For example, visual question answering tasks that requires only simple skills but need to be combined correctly [74]; in program synthesis, the function defined on a domain specific language need to be inferred from the input and output status [131]. In the weak logic scenarios, the mapping from input to output can be hardly expressed in a set of clear rules. For instance, the natural language translation, if expressed in logic, is roughly consist of a bilingual vocabulary translation with numerous exceptions, and mapping of grammar rules between languages, *e.g.* some languages are subject-verb-object language while some others are subject-object-verb languages. Reading comprehension is another case of weak logic, since its questions can cover a large range of human common sense and logic.

2.2.3 Interaction

Except in symbolic neuro symbolic method, where the symbolic system is implicitly embedded in the neural network, the neural network in other neural symbolic systems need to interact with symbolic system. In neuro: symbolic $\rightarrow$ neuro approach, the interaction is through the dataset, thus the interaction could be synchronous or asynchronous. In the synchronous interaction, the input is presented to the neural network and the symbolic system, then the neural network receive instance supervision from the symbolic system's output. In the asynchronous interaction, all the input data is pre-processed by symbolic system. In the neuro|symbolic approach, the neural network produces entire symbolic output and feeds into symbolic system to directly generate the target output. On the contrary, neuro[symbolic] and symbolic[neuro] methods allow and encourage multiple rounds of interactions.

2.3 Representation Power of Neural Networks

The representation power of neural networks remains in core of deep learning research. In the 1980s, the confirmation of representation power of feed-forward neural network was established by Cybenko [37] and Hornik et al. [67]: a feed-forward neural network (FFN),

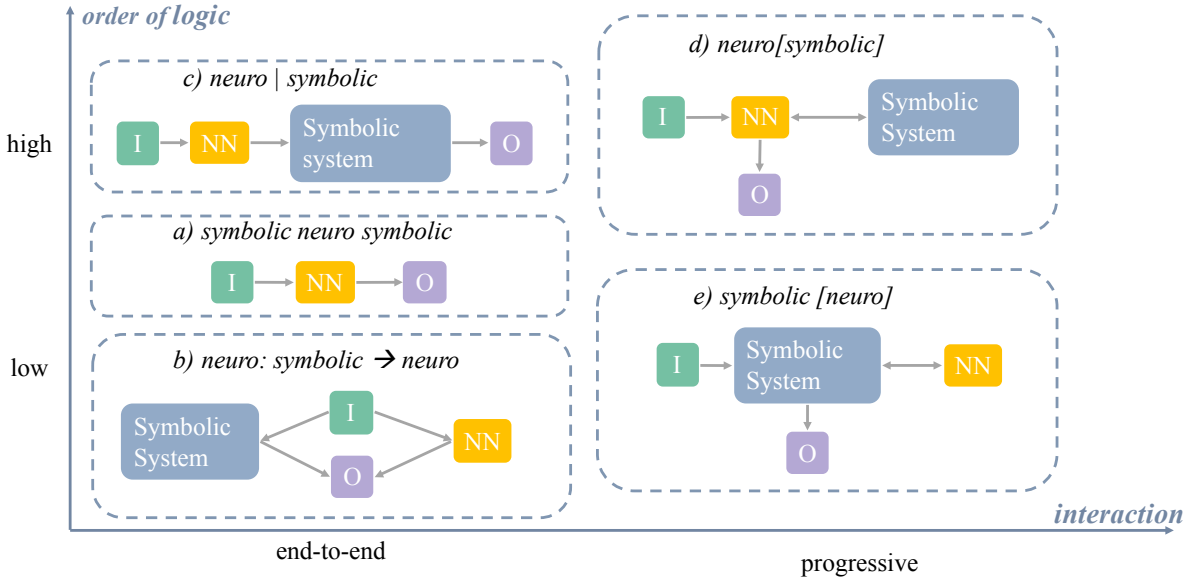


Figure 2.2. Decide the neural symbolic approach by logic and interaction.

which has one hidden layer that uses sigmoid as activation function and large enough, can approximate any bounded continuous function to arbitrary precision; with one more hidden layer, can approximate any function to arbitrary precision.

Though the simple feed-forward neural networks are believed to be universal approximators, the *large enough* requirement can hardly be fulfilled and tested in practice. Especially, FFNs are less computational efficient comparing to the CNN and RNN networks when the apparent structure exists in the input, *e.g.* spatial adjacency of pixels in an image, temporal order of observations of a signal, and the chronological order of the input sequences.

2.4 Input/Output Structure Representation in Neural Networks

Recognizing the inherent structure embedded with in input and output data, exemplified by Figure 2.3 and Figure 2.4, researchers have dedicated significant efforts to incorporate the structure into the neural networks, especially in sequential models. While the neural network should be able to automatically understand the input and output structure given sufficient training

and data, explicitly incorporating the knowledge of structure of data often proves beneficial. Embedding structural information into encoder networks enhances their ability to accurately comprehend the input, while constrain the decoder neural network with structural information facilitates efficient decoding of valid outputs.

In the **input** domain, the structures can be represented as additional fields within the features provided to a regular neural network or as explicitly structural definitions that shape the neural network topology. For instance, in natural language processing, the dependency parsing graph and part-of-speech tags of sentences illustrates the structure of the texts. This information can be directly fed as supplementary input features into the linear neural networks such as BERT model [33] and LSTM [9].

In the **output** domain, the the structure is typically employed to refine the generation space. For example, tree-structured decoders [33] have been proposed to generate the parse tree, which can be later translated to the target sequences. This approach effectively leverages structural information to guide the generation of syntax-correct samples. In specific domains, it's also common to transform the structural constraint as a template, and ask the decoder to define the placeholders in the template. Such examples are abundant in decoder of domain-specific applications, such as for SQL queries [160], operators on tabular data [61], and regular expressions [172].

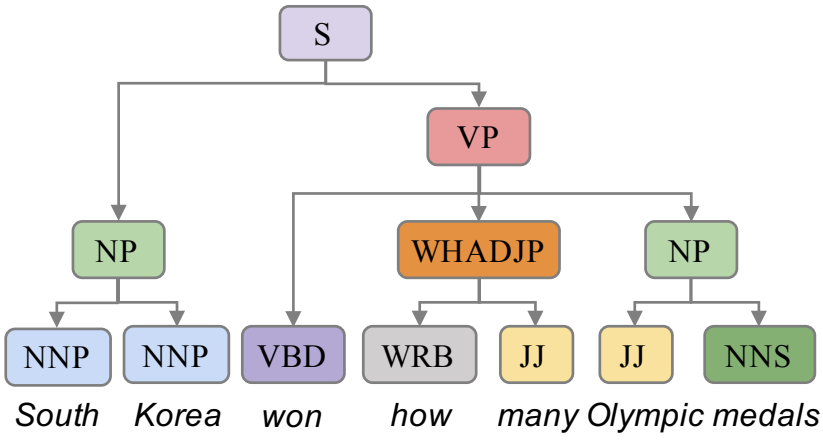


Figure 2.3. Grammar structure in natural language.

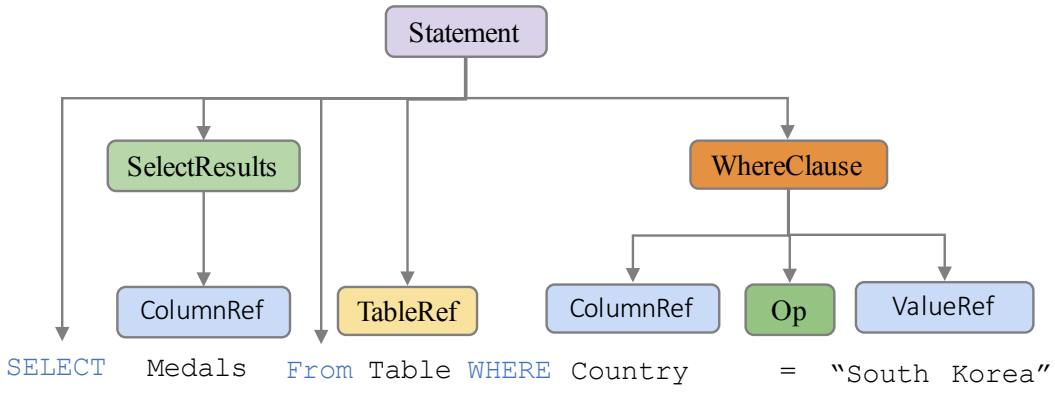


Figure 2.4. Tree structure of SQL query.

Chapter 3

Computation Power of Linear Models.

Long Short-Term Memory (LSTM) and Transformers are two popular neural architectures used for natural language processing tasks. Theoretical results show that both are Turing-complete and can represent any context-free language (CFL). In practice, it is often observed that Transformer models have better representation power than LSTM. But the reason is barely understood. We study such practical differences between LSTM and Transformer and propose an explanation based on their latent space decomposition patterns. To achieve this goal, we introduce an oracle training paradigm, which forces the decomposition of the latent representation of LSTM and the Transformer, and supervises with the transitions of the Pushdown Automaton (PDA) of the corresponding CFL. With the forced decomposition, we show that the performance upper bounds of LSTM and Transformer in learning CFL are close: both of them can simulate a stack and perform stack operation along with state transitions. However, the absence of forced decomposition leads to the failure of LSTM models to capture the stack and stack operations, while having a marginal impact on the Transformer model. Lastly, we connect the experiment on the prototypical PDA to a real-world parsing task to re-verify the conclusions.

3.1 Introduction

The LSTM network has achieved great success in various natural language processing (NLP) tasks [138, 150], and in recent years, the Transformer network keeps breaking the

record of state-of-the-art performances established by LSTM-based models in translation [145], question-answering [42], and so on [15, 40]. Besides exploring the capacity boundary of the Transformer network, there is an increasing interest in investigating the representation power of the Transformer network and explaining its advantage over the LSTM models theoretically and empirically.

Existing analysis has proven that both LSTM [132] and the Transformer network [115] are Turing-Complete. However, much empirical evidence shows both models are far from perfect in imitating even simple Turing machines [40, 76]. Several explanations of the performance gap between practice and theory are 1) some theoretical proofs relies on infinite precision assumption while the precision in practical computations is limited [85, 152]; 2) given fixed latent space dimension, according to the pigeonhole principle, as input sequence length goes towards infinity, there will be information that can not encode into the latent space or is forgotten [56].

Despite the soundness of the theoretical proofs and explanations, none of them can directly explain the phenomenons in practice: 1) given the same computation precision, Transformer has an advantage over LSTM model in many cases; 2) both Transformer and LSTM fails even when the input sequence is short and their latent space dimension is large.

In this work, we study the empirical representation power of the LSTM and the Transformer networks and investigate the origination of their difference. We compare the models via learning context-free languages. The reason to study CFLs other than regular languages or Turing machines is that the learning of CFLs provides most insights into NLP tasks where understanding the underlying hierarchy of the language is crucial. In the rest of the paper, we will first introduce an oracle training paradigm to predict the status of the PDA that accepts the CFL, and a regularizer to explicitly decompose the latent space of the LSTM and the Transformer such that the PDA state and the positions in the stack are encoded in distinct dimensions. Lastly, the experiment section exhibits the empirical results and leads us to the following conclusions:

- LSTM and Transformer have similar upper bound of empirical representation power in

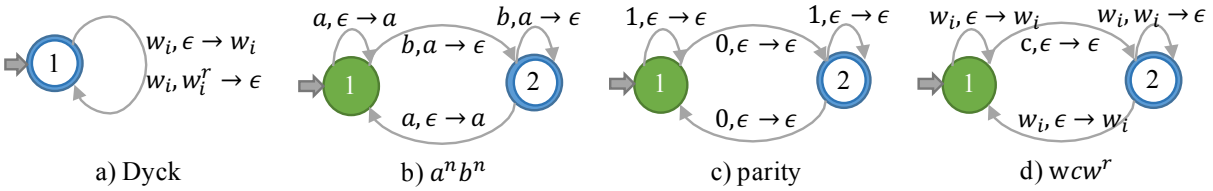


Figure 3.1. Graphical notations of PDAs for the CFLs.

simulating PDAs.

- LSTM fails to factorize its latent space to encode the state and multiple elements of the stack without explicit supervision, which is the pivot to its compromised performance in real-world tasks. Meanwhile, the Transformer is marginally affected by absence of explicit decomposition regularization.

- Language recognition is not a reliable task to compare the empirical capacity of LSTM and Transformer since the results is sensitive to the setting of PDAs, the hyperparameters of the models, and the training methods.

3.2 Preliminaries and Definitions

3.2.1 Context-Free Language

A context-free grammar $\mathscr{G}$ is defined by a collection of nonterminal symbols (*i.e.* variables), terminal symbols (*i.e.* the alphabet of $\mathscr{G}$), production rules and a start symbol E. The context-free language of $\mathscr{G}$ is the set of strings that can be derived from the start symbol by iteratively applying the production rules until there are no nonterminal symbols.

A Pushdown Automaton (PDA) is a state machine with a stack that reads an input symbol and the top element in a stack at each step and performs the state transition and some stack operations (push/pop). Formally, a PDA can be defined as a tuple of $< Q, \Sigma, S, \delta, q_0, I, F >$. Q is a finite set of states, and $q_0 \in Q$ is the initial state; Σ is the alphabet of the inputs, S is the stack symbols and $I \in S$ is the initial stack top symbol; $F \in Q$ is a set of accepting states. δ is a transition function $\delta(q \in Q, x \in (\Sigma \cup \{\varepsilon\}), s \in S) \to q', s'$, where ε denotes an empty symbol

and s denotes the top element in the stack S. The transition function implies the stack operation. For example, let $*$ be a wildcard for arbitrary state or symbol, $\delta(*, \varepsilon, *)$ represents transition that consumes no input symbols[1]; $\delta(*, *, \varepsilon) \rightarrow *, s'$ where $s' \neq \varepsilon$ is a stack push operation, and $\delta(*, *, s) \rightarrow *, \varepsilon$ where $s \neq \varepsilon$ is a stack pop operation.

A PDA can be equivalently expressed by some CFG and vice versa [31, 124]. In learning CFGs, the neural networks are expected to learn the equivalent PDAs instead of the production rules.

In this study, we are interested in bounded CFGs and PDAs where the recursion depth is finite and a stack with finite size should be adequate for process the CFLs. The reason is two-fold. First, we agree with the existing theoretical analysis that encoding an unbounded stack into finite space is the bottleneck for LSTM and Transformer, thus we focus on the investigation of their realistic representation power before they reach the theoretical upper bound (*e.g.* number of recursions $\rightarrow \infty$). Second, in natural languages and even programming language, the nesting of the production rules are very limited (*e.g.* < 100), so it's important to understand LSTM and Transformer's behavior under bounded CFGs.

Four canonical PDAs (shown in Figure 3.1) are introduced as follows:

Definition 3.2.1 (Dyck-(k, m))**.** The language of paired parentheses of k types up to m recursions. Dyck is the prototypical language to any context-free languages [87].

Definition 3.2.2 ($a^n b^n$)**.** The set of strings consisting of a and b, and every occurrence of n consecutive as is followed by exact n successive bs.

Definition 3.2.3 (Parity)**.** The binary strings that containing an even number of 0s. Parity can be expressed by a Deterministic Finite Machine (DFA). To formalize parity in PDA, we denote all parity transition function with no stack operation $\delta(*, *, \varepsilon) \rightarrow *, \varepsilon$.

Definition 3.2.4 ($(wcw^r)^n$-(k, m))**.** wcw^r-(k, m) generates strings that starts with a sub-string ω followed by a character c, then followed by the reverse of ω. The chars $w \in \omega$ comes from a

[1]For instance, the *reduce* operation in shift-reduce parsing, as shown in Table 3.2

vocabulary Ω ($c \notin \Omega$) of size k. The string ω contains no more than m chars. The $(wcw^r)^n$-(k, m) generates strings that contains a to n substrings from wcw^r-(k, m).

3.2.2 Long-short Term Memory Network

Given the input sequence embeddings $\mathbf{X} = \{x_t\}_{t=1}^n$ and initial hidden state and cell state $(h_0, c_0) \in \mathbb{R}^d$, LSTM [64] produces the latent representation of the sequences $\{h_t\}_{t=1}^n$, where $h_t \in \mathbb{R}^d$, as follows:

$$
\begin{aligned}
f_t &= \sigma(\mathbf{W}_f h_{t-1} + \mathbf{U}_f x_t + b_f) \\
i_t &= \sigma(\mathbf{W}_i h_{t-1} + \mathbf{U}_i x_t + b_i) \\
o_t &= \sigma(\mathbf{W}_o h_{t-1} + \mathbf{U}_o x_t + b_o) \\
\tilde{c}_t &= tanh(\mathbf{W}_c h_{t-1} + \mathbf{U}_c x_t + b_c) \\
c_t &= f_t \odot c_{t-1} + i_t \odot c_t \\
h_t &= o_t \odot tanh(c_t)
\end{aligned}
\tag{3.1}
$$

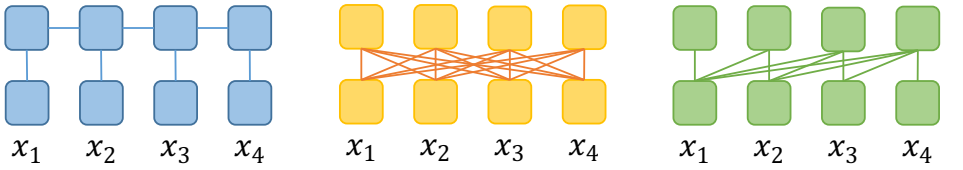
3.2.3 Transformer Network

The Transformer network [145] process the sequence via multi-head attention. Specifically, let the $att(Q,K,V)$ represent the scaled dot-product attention function over query $Q \in \mathbb{R}^{r_0 \times r_1}$, key $K \in \mathbb{R}^{r_0 \times r_1}$, and value $V \in \mathbb{R}^{r_0 \times d}$, defined as:

$$
att(Q,K,V) = softmax(\frac{QK^\top}{\sqrt{r_1}}V) \tag{3.2}
$$

Given sequence $\mathbf{X}$, the Transformer encodes the sequences via multi-head attention followed by a point-wise feed-forward network (denote as $FFN(\cdot)$).

$$
\begin{aligned}
h_t &= FFN([head_1; ...; head_k]); \\
head_i &= att(y\mathbf{W}_i^Q, y\mathbf{W}_i^K, y_t\mathbf{W}_i^V)
\end{aligned}
\tag{3.3}
$$



a) LSTM b)Transformer Encoder c)Transformer Decoder

Figure 3.2. The information flow in LSTM, transformer encoder and decoder.

To distinguish the order of input symbols, the y_t is produced by summing a positional encoding with input encoding:

$$\begin{aligned} p_{t,2j} &= sin(t/10000^{2j/r_1}) \\ p_{t,2j+1} &= cos(t/10000^{2j/r_1}) \\ y_t &= x_t + p_t \end{aligned} \tag{3.4}$$

We distinguish the *Transformer encoder* that allows attention between any pairs of input symbols and the *Transformer decoder* that allows multi-head attentions to compute only on past symbols. The comparison is illustrated in Figure 3.2.

3.3 Oracle Training

In this section, we introduce the oracle training method assuming complete PDA transition steps are exposed to the model to provide the densest supervision signal.

Formally, given symbol sequences $\{X^{(i)}\}_{i=1}^{n}$ accepted by a PDA, the *oracle* includes $\{X^{(i)}\}_{i=1}^{n}$, the states of DPA $\{S^{(i)}\}_{i=1}^{n}$ and the stack status $\{\mathscr{T}^{(i)}\}_{i=1}^{n}$ at each step while processing the symbol sequences. The oracle training forces the models to predict not only the next symbols as in the language model, but also the internal state and stack status. Hereinafter, for simplicity, we omit the superscript i that denotes i-th sample. Let $X = \{x_t\}_{t=1}^{\tau}$, $S = \{s_t\}_{t=1}^{\tau}$, and $\mathscr{T} = \{T_t\}_{t=1}^{\tau}$, and the $T_t[j]$ being the j-th item in the stack at step t.

Figure 3.3 shows the generic architecture for oracle training. Several multi-layer percep-

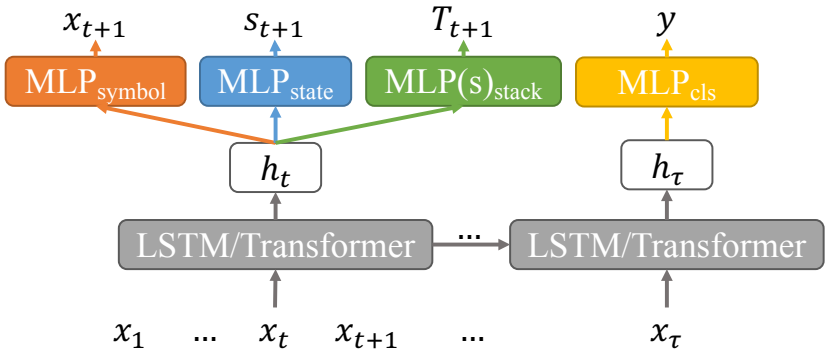


Figure 3.3. Oracle training paradigm. The hidden state, the output vector of a LSTM and transformer layer, is fed to independent prediction head to predict the pushdown automaton's state, stack, and the next symbol.

tron (MLP) networks are employed to independently predict the symbol, state, and stack from the latent representation h_t. For stack status, a non-full stack is padded with the empty token ε. Therefore, the models always predict a constant number of symbols for the stack and predict εs in the correct positions to indicate a non-full stack. We also include a language recognition task in which the model predicts if the sequences are accepted by the PDA. The language recognition task is widely used in arguing the inability of LSTM and Transformer in recognizing CFLs. Though from the PDA's perspective, the recognition is equivalent to the task of learning the transition, while in the experiment, we show both models benefit greatly from dense supervision in the oracle training, compared to the sparse supervision in the language recognition.

We also introduce two vital model configurations: **forced decomposition** (Figure 3.4) and **latent decomposition** (Figure 3.5). In forced decomposition, the latent representation vector h_t is split into $m+1$ segments, where m is the maximum stack size, to predict the PDA state and m elements in the stack separately. Since each position in the stack shares the same set of stack symbols, we let the stack predictors MLP_{stack} share parameters. In contrast, the latent decomposition uses whole h_t but individual MLPs for prediction, and the m stack predictors are independently trained. In both configurations, the next symbol is predicted based on complete h_t because the valid symbol for the next step depends on both the current state and stack.

Predictions of symbol, state, and stack are all trained with cross-entropy loss, and the

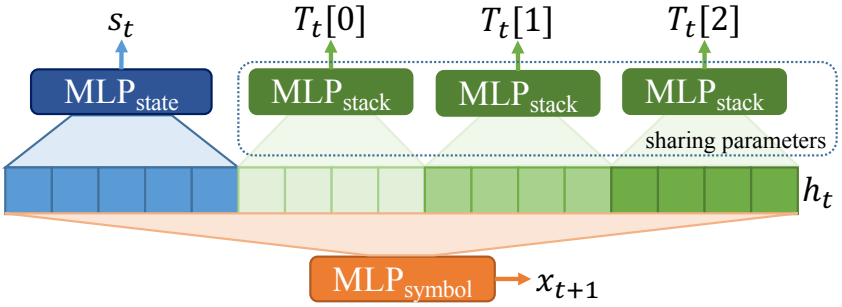


Figure 3.4. Forced decomposition of h_t.

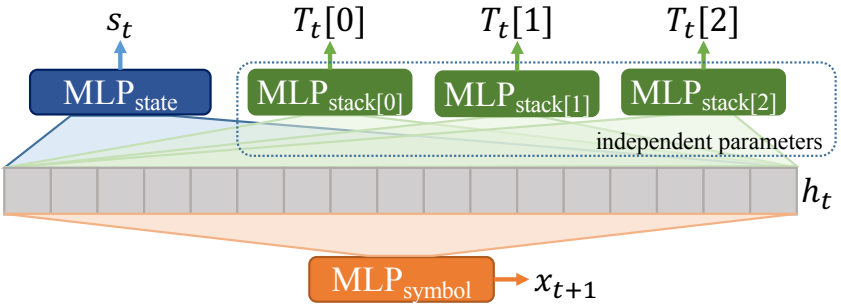


Figure 3.5. Latent decomposition of h_t (with stack size of 3).

three-part losses are summed [2] :

$$\mathscr{L}_{oracle} = \mathscr{L}_{symbol} + \mathscr{L}_{state} + \mathscr{L}_{stack} \tag{3.5}$$

3.4 Experiment on Canonical PDAs

In this section, we evaluate the representation power of LSTM and Transformer by simulating canonical PDAs.

3.4.1 Experiment Setup

Data generation. For PDAs introduced and their hyperparameters $\{k, m, n\}$ when applicable, we generate the 50k sequences for training and another 50k for testing except for wcw^r(n=2, m=2, k=*) which enumerate all accepted sequences. For each sequence, the

[2] An option is to assign weights to each term on the right hand side, our additional experiment in Section 3.4.5 shows that the choice of loss weights does not influence the main conclusion in the experiment.

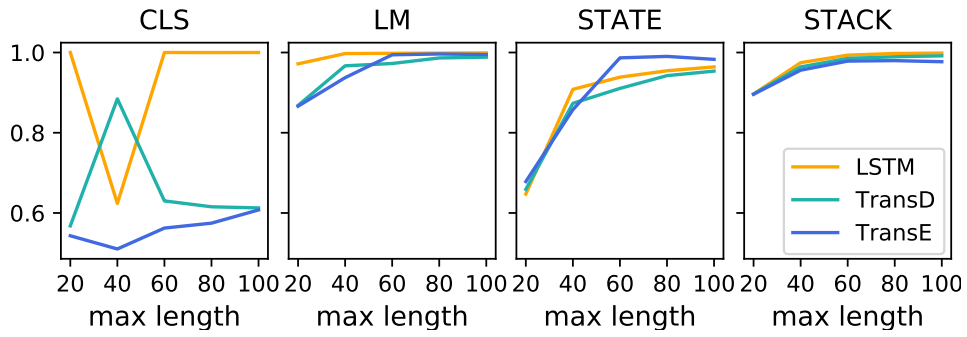


Figure 3.6. Performance on $a^n b^n$.

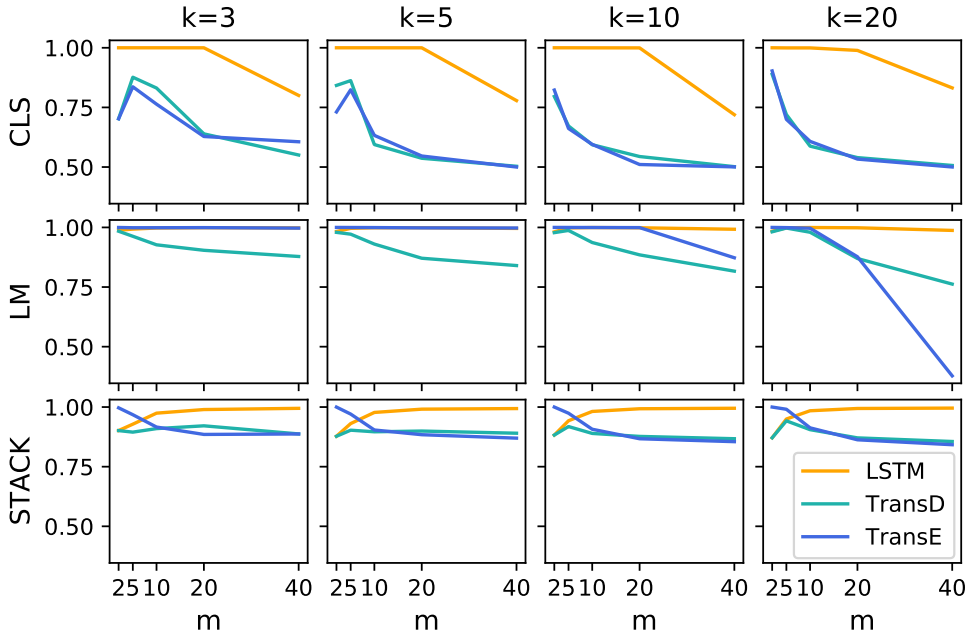


Figure 3.7. Performance on Dyck.

ground-truth PDA annotates the sequence and produces the state and stack labels for oracle training. Meanwhile, a corrupted sequence that is not accepted by the PDA is generated for the classification sub-task. For language modelling, we compute the valid symbols for each step t given sequence $x_1, \cdots, x_{t-1}$ and denote this validity over alphabet as *LM mask*.

Model configurations. For each PDA task, we set the hidden size in the LSTM model and the latent dimension in the Transformer model to be $\alpha * (|Q| + m * |S|)$, where α is the scale factor, $|Q|$ is the size of states, m is the maximum recursion (*i.e.* maximum stack size), and $|S|$ is

the size of stack symbols[3]. We always let $\alpha \geq 1$ such that the latent vector h_t will always have enough dimensions to encode both the state and stack. For the Transformer encoder and decoder, the number of attention heads is 8. For all models, unless specified otherwise, the number of layers is 1 and $\alpha = 1$. We set all MLP modules to be a two-layer feed-forward network with sigmoid as an activation function. The hidden dimension of the MLPs is twice of their input feature dimension. For both models, we use an embedding layer to encode input symbols to $\mathbb{R}^{2|\Sigma|}$. For the Transformer model, the filter size for the position-wise feed-forward network is 32, and we apply a dropout layer with a rate of 0.1.

Training. Models are trained with Adam optimizer with a learning rate of 0.001 on the AWS platform. The models are trained for 200 epochs or up to convergence. We introduce *two-phase training* for language recognition tasks, *i.e.* classify whether the sequences are accepted by the PDA. In phase 0, the models are initialized and solely trained by a classification task, while in phase 1, the models are retrained on classification tasks after being trained with oracle training.

Metrics. 1) Classification accuracy: portion of the sequences that are correctly accepted/rejected. 2) LM accuracy: percentage of predicted symbols that are valid according to the LM mask. 3) State accuracy: accuracy in predicting the current PDA state shown only the sequence of symbols. 4) Stack accuracy: accuracy in predicting current stack status, stack symbols (including empty symbol), and their position in the stack shown only the sequence of symbols.

3.4.2 Forced Factorization Results

Figure 3.6, 3.8, 3.7, 3.9 shows the LSTM and Transformer performance over multiple configuration of PDAs [4]. There are few observations from the results: 1) LSTM has higher accuracy in learning PDAs, as it generally achieves higher accuracy in both state and stack predictions,

[3]For PDAs introduced, we set $S = \Sigma \bigcup \{I, \varepsilon\}$

[4]These figs shows phase 0 classification accuracy

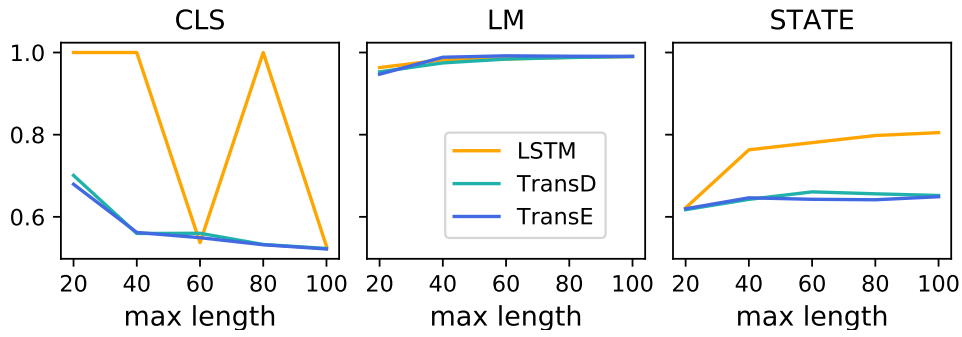


Figure 3.8. Performance on parity

especially shown in Figure 3.8, 3.9. The results shows that when CFGs are bounded, LSTM does not need external memory to simulate the stack. 2) For predicting the state, Transformer decoder behaves in a similar way to LSTM, as shown in state accuracy in Figure 3.6, 3.9. In $a^n b^n$and $(wcw^r)^n$, the Transformer encoder slightly outperforms LSTM and Transformer decoder. This indicates that though past information should be adequate to determine the current status of PDA, it's still benefits the neural models to foresee the future sequences. 3) Both language recognition task and language modeling solely should not be used to examine the capability of neural models in learning CFG, since they do not fully reflect the capability of models to learn the internal dynamics in state transition and stack operation (Figure 3.8), and reversely learning the precise PDA does not necessarily lead to perfection in language recognition and language modeling (Figure 3.7, 3.9).

3.4.3 Decomposition Hardship of LSTM: Forced v.s. Latent Factorization

Though LSTM and Transformer show comparable representation power under forced decomposition setting, the disadvantage of LSTM in the latent decomposition training is significant, as can be viewed from Figure 3.10, 3.11, meanwhile, Transformer models are roughly as good as they are trained with forced decomposition. The failure of LSTM in stack prediction is key to its disadvantage in many tasks [42].

Visualization Figure 3.13 shows the two-component t-SNE [144] results of the LSTM

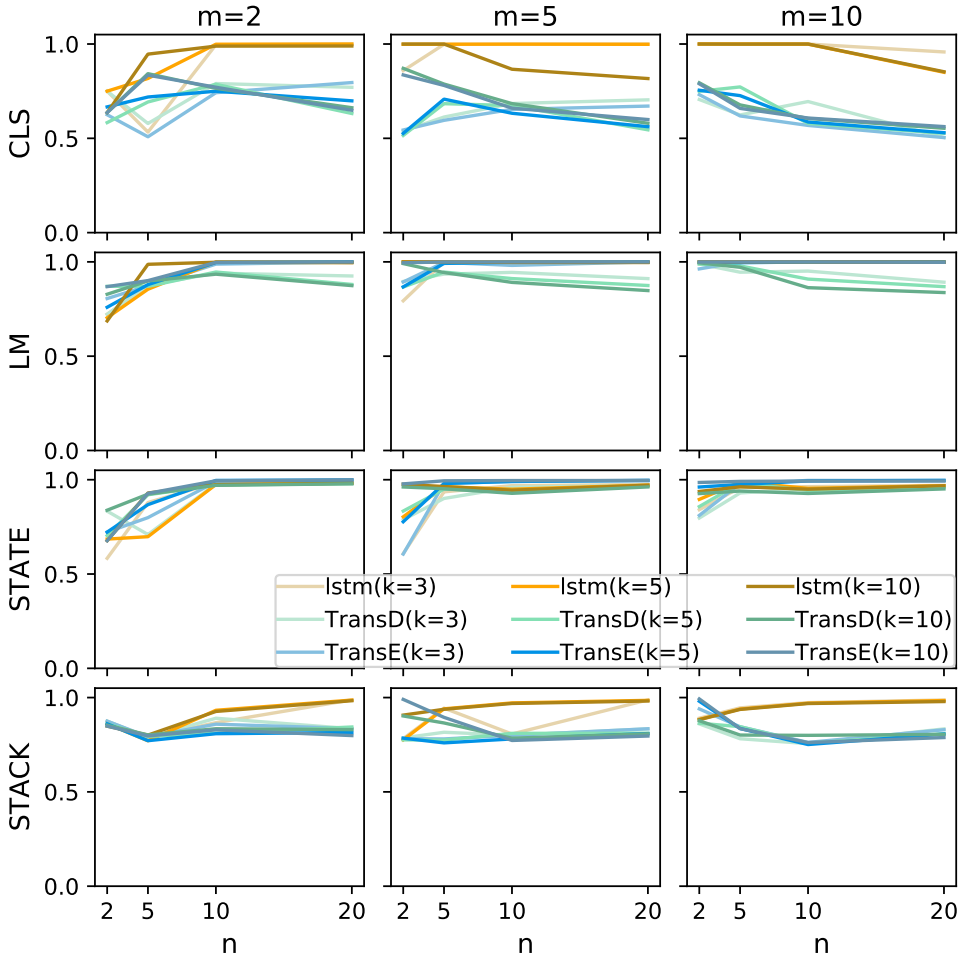


Figure 3.9. Performance on wcw^r.

hidden states h_t. The different colors represent distinct stack status in the oracle. As shown, the hidden states in forced decomposition tend to separately encode different stack status, while in the latent decomposition, there are much more clusters containing multiple colors, which causes the failure of stack predictor to correctly predict the stack status, and also prevents LSTM itself to learn the correct stack operations.

3.4.4 Two-Stage Training Improves Language Recognition Accuracy

We detail the four-tier reasons for our earlier conclusion that language recognition accuracy is unfair in judging the models' capability of learning PDA. Firstly, we explain the observations from Figure 3.7, 3.9 that perfection in state and stack prediction does not lead to

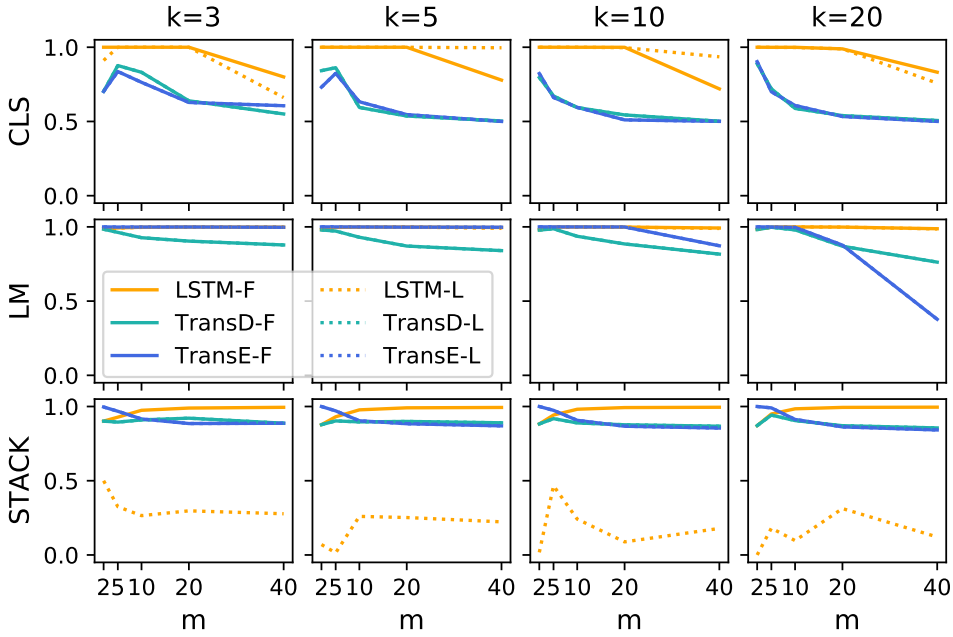


Figure 3.10. Performance on Dyck with forced (solid lines) and latent (dotted lines) decomposition.

accurate recognition: the language recognition requires a higher level of reasoning than just learning a set of PDA transition functions. A rejected sequence for PDA might due to the current symbol not accepted given the state and stack top, popping an empty stack, or exceeding the maximum recursion. Thus the classification decision boundary might be inseparable for an MLP without special design. Furthermore, for any models, the error message might occur at random steps, and the models have to preserve and pass the message to the last step for the classification prediction in the general training paradigm for language recognition. This is mainly why the classification accuracy of LSTM declines much abruptly than of the Transformers since Transformer can pass information between any pairs of inputs, thus the comparison of LSTM and Transformer will be sensitive to sequence lengths. Besides sequence length, the model size also influences the recognition accuracy differently. Comparing the classification accuracy of Figure 3.7 and Figure 3.19, the relative advantage of two models are reversed when models scale up. Lastly, Transformer models may recognize language in some manners that are dissimilar to PDAs. To prove this, we show the classification accuracy of phase 0 and

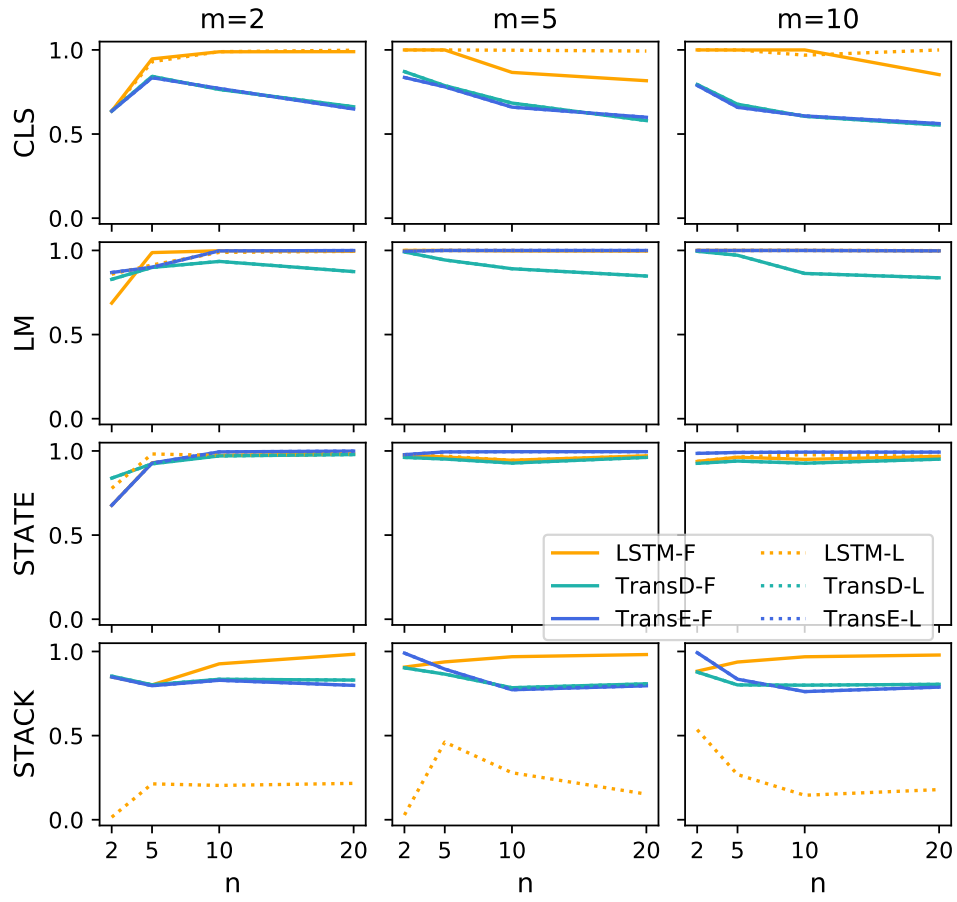


Figure 3.11. Performance on wcw^r with forced (solid lines) and latent (dotted lines) decomposition ($k = 10$).

phase 1 in Figure 3.14, 3.15, 3.16. LSTM generally improves after oracle training (phase 1), especially in Dyck (Figure 3.16), while the Transformer models suffers from oracle training (Figure 3.14-3.15).

3.4.5 Sensitivity to Loss Weights

Loss function: In Equation (3.5), the weights of $\mathscr{L}_{symbol}$, $\mathscr{L}_{state}$, $\mathscr{L}_{stack}$ are set to 1. To confirm the conclusion is general without dependent on the choice of weight values, we examine the model performance using the learnable loss weights [80], and the oracle training loss has the form:

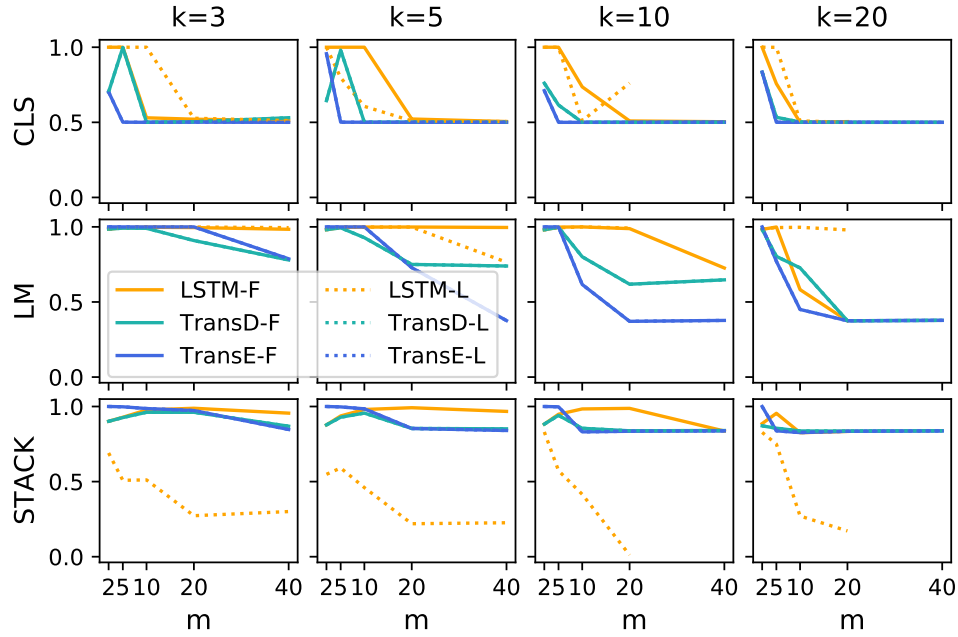


Figure 3.12. Four-layer models ($\alpha = 4$) on Dyck with forced (solid lines) and latent (dotted lines) decomposition.

$$\mathscr{L}_{oracle} = \frac{1}{2\sigma_1^2}\mathscr{L}_{symbol} + \frac{1}{2\sigma_2^2}\mathscr{L}_{state} + \frac{1}{2\sigma_3^2}\mathscr{L}_{stack} + log(\sigma_1\sigma_2\sigma_3) \tag{3.6}$$

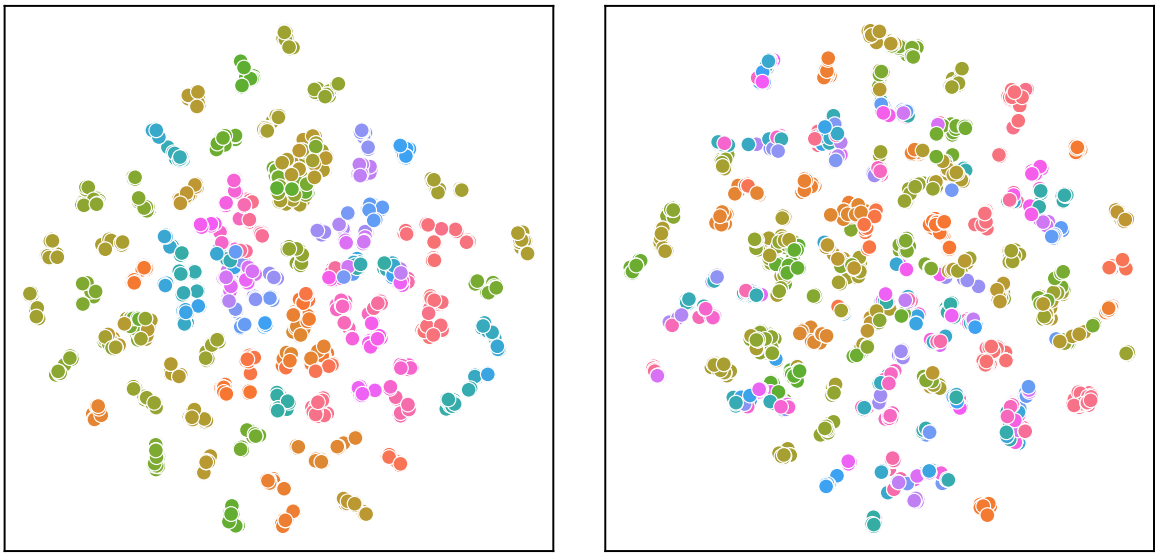
where $\sigma_1, \sigma_2, \sigma_3$ are trainable parameters.

Dataset and training: We used Dyck-(5,*) datasets ($m = 2, 5, 10, 20$). For each model on each dataset, the training is repeated five times.

Results: Figure 3.17 compares the model performance with fixed or learned weights and illustrates that neither the performance nor the hardship in factorization is sensitive to the choice of loss weights.

3.4.6 Deeper and Wider Models

Generally, a larger number of parameters brings higher representation power, so does a larger number of layers. We verified the conclusion that 1) LSTM and Transformer have similar representation power of learning CFG on larger models; and 2) forced factorization improves performance of LSTM.



a) Forced b) Latent

Figure 3.13. t-SNE analysis on LSTM hidden states trained on Dyck (k=3,m=5).

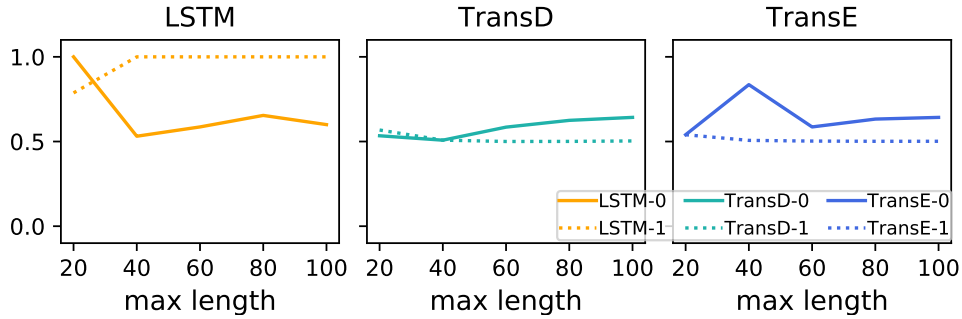


Figure 3.14. Two-phase classification accuracy on $a^n b^n$.

Figure 3.19, 3.12 show that additional layers and extra hidden size can not remedy the hardship in decomposition. Figure 3.18, 3.24 provide further evidence. From Figure 3.18, we notice that the difference between forced and latent factorization is alleviated in Transformer models as the number of layer increases and α grows[5]. However, the gap remains huge for the LSTM model. Besides, the decrease of classification and language model accuracy is generally observed when scaling up the models, which indicates the training difficulty introduced by the

[5]The ablation on factorization is not conducted for L4S1 and L1S4 for any PDA. Also, since the factorization makes most significant difference on learning stack and Parity does not require a stack, the comparison of factorization is not made on Parity.

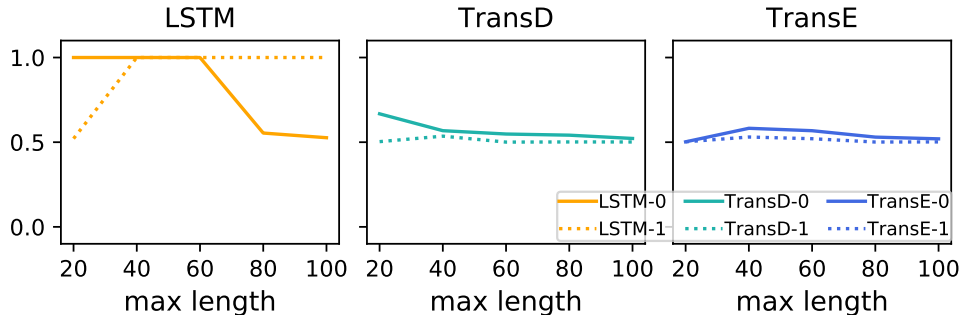


Figure 3.15. Two-phase classification accuracy on parity.

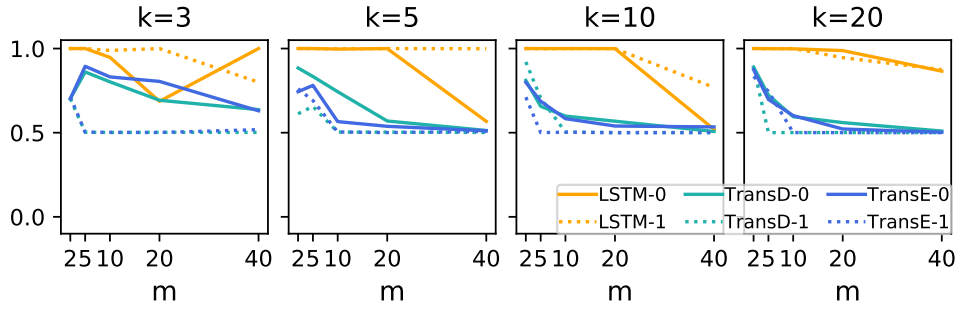


Figure 3.16. Two-phase classification accuracy on Dyck.

deeper and wider model overwhelms the benefit of increased model capacity.

3.5 Experiment on Shift-Reduce Parsing

Dataset. SCAN [90] is a semantic parsing dataset consisting of commands in natural language and sequences of actions to execute the commands. Tab. 3.1 shows the generation rules producing SCAN commands. The dataset contains 16728 training samples and 16728 test samples. In our experiment, instead of parsing to the sequence of actions, we parse the linguistic command according to its CFG production rules (Tab. 3.1) using shift-reduce parsing. Since there are production rules in the CFG that map a single nonterminal variable to another one, the corresponding PDA is nondeterministic. To facilitate the training, we insert a special token $<$*reduce*$>$ to make the process deterministic. Tab. 3.2 illustrates an example of the annotation of the sequence with stack status and the padding process to insert $<$*reduce*$>$ tokens for reduction operations that do not consume symbol from the input sequence. The equivalent PDA of SCAN's

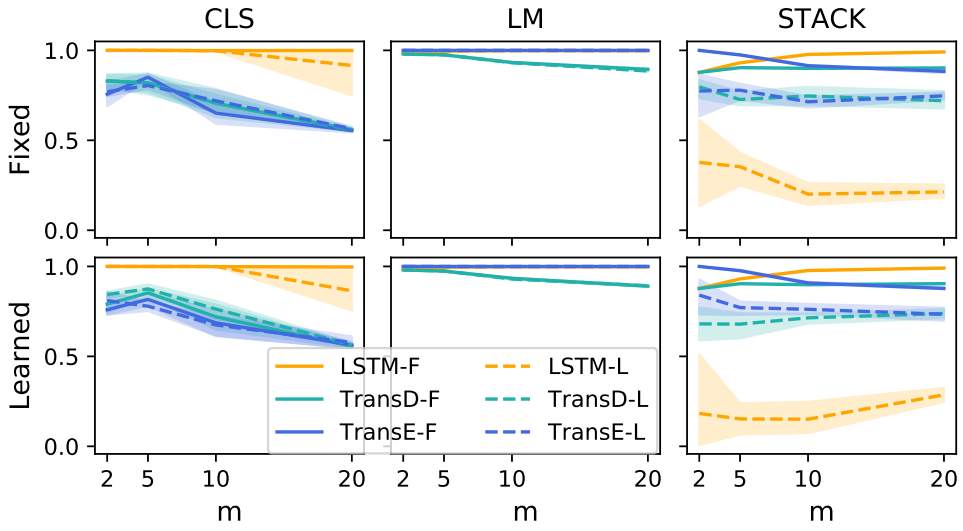


Figure 3.17. Model performance on Dyck-(5,*). First row shows results with Equation (3.5), and second row shows results with Equation (3.6). Lines indicates the average accuracy over 5 runs, and the shadows illustrate the range.

Table 3.1. Domain Specific Language for SCAN linguistic commands.

C	:=	CP S — CP V — S
CP	:=	S and — S after
S	:=	V — V twice — V thrice
V	:=	VP D— VP left — VP right — D
VP	:=	D opposite — D around
D	:=	U — U left — U right
U	:=	walk — look — run — jump — turn

CFG has only one state q_0. Alphabet Σ covers English words in the linguistic commands and the $<reduce>$ token, and the stack symbols are $\{C, CP, S, V, VP, D, U\}$.

Models and metrics. The model configurations are the same as in learning the canonical PDAs. We compute perplexity to evaluate language modeling and compute the accuracy of the stack prediction as the parsing accuracy.

Results. Tab. 3.3 sums up the results of LSTM and Transformer on SCAN dataset, which is consistent with the observations in Sec. 3.4: LSTM and Transformer decoder perform similarly in language modeling and parsing when decomposition is forced, and the parsing accuracy of LSTM suffers heavily from latent decomposition setting. The Transformer encoder

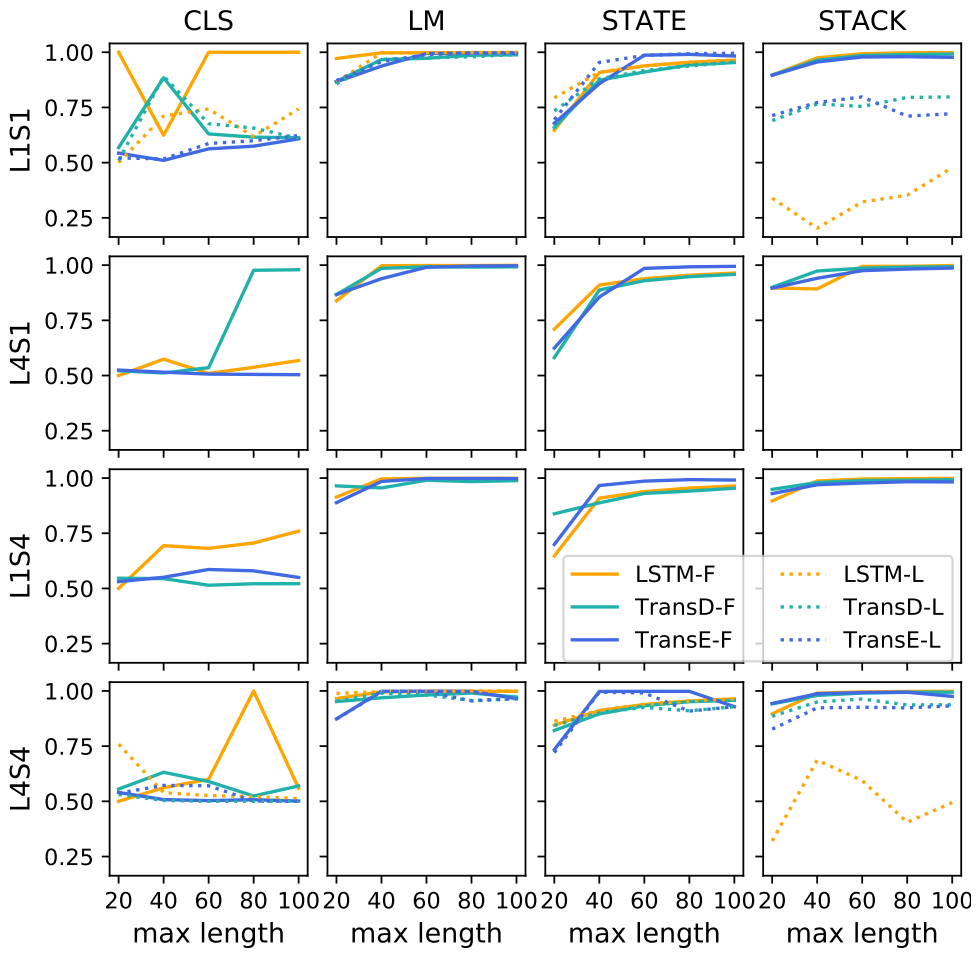


Figure 3.18. Performance on $a^n b^n$. Y-label shows number of layers and scaling factor α. For example, *L1S4* represents single-layer model with $\alpha = 4$.

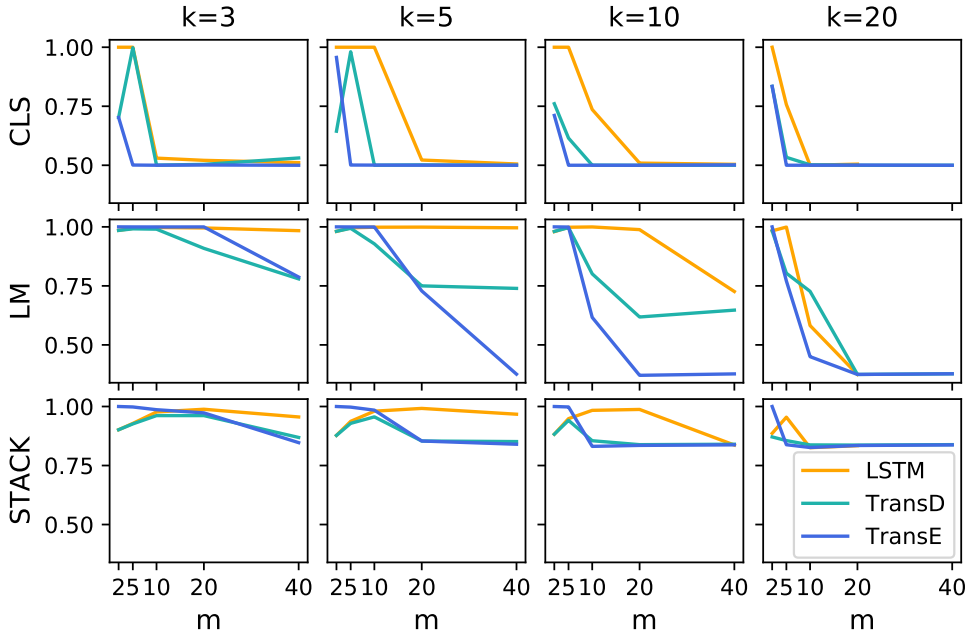


Figure 3.19. Performance of four-layer models on Dyck with $\alpha = 4$

can see the whole sequence at each step, so it achieves an almost lower bound of perplexity and has the highest parsing accuracy.

3.6 Related Works

There are many theoretical analyses on the representation power of LSTM and the Transformer and their comparisons that motivate this work to re-examine their representation power from an empirical aspect. Siegelmann and Sontag [132] firstly established the theory that given infinite precision and adequate number of hidden units RNNs are Turing-Complete, and Hölldobler et al. [65] has designed an one-unit vanilla RNN counter for recognizing counter languages (*e.g.* $a^n b^n$ and $a^n b^n c^n$) with finite range of n. Recently, Hewitt et al. [62] proposed the construction of RNN that performs stack operations as in PDA and encodes PDA stack within hidden states of RNN without external memory. Pérez et al. [115] proofs that with arbitrary precision, the Transformer network could simulate the single execution step for Turing machine, then by induction the Transformer is Turing-Complete. The majority of theoretical

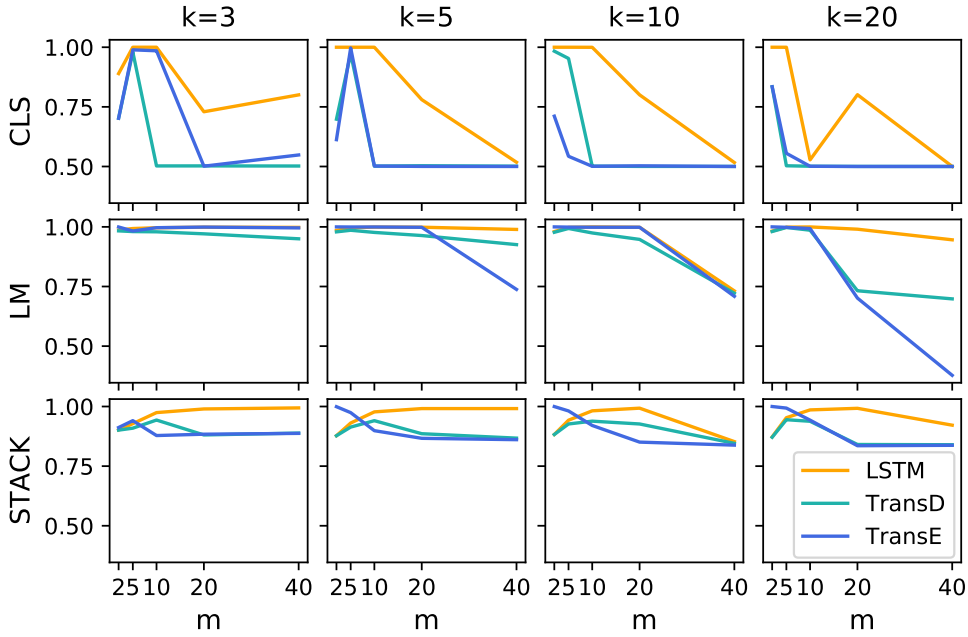


Figure 3.20. Performance on Dyck ($\alpha = 1$, four layers)

analysis emphasis that limited computation precision may break the proofs and compromise the performance in practice. Nevertheless, the models are not supervised with either step-by-step execution of the Turing Machine or the actual counters, which might be the crux to the failures of both models in reality.

This work also closely relates to previous attempts to connecting LSTM and transformer model with a specific type of languages, *e.g.* languages from Chomsky's hierarchy [32] and counter languages. 1) For regular languages (representable by DFAs), Michalenko et al. [107] shows the empirical ability of LSTM to represent DFAs and Rabusseau et al. [117] has proposed a construction method of RNNs from a weighted DFA. 2) For context-free languages (representable by PDAs), Sennhauser and Berwick [125] observed that CFGs are hardly learnable by LSTM models. on the other hand, Bhattamishra et al. [12] showed LSTM could learn CFGs with bounded recursion depth but the performance will be limited for infinite recursion. 3) For counter languages, the Transformer network [11] and LSTM Suzgun et al. [139] have been trained to predict the outputs of a dynamic counter. However, their results disagree on the capability of LSTM in representing DYCK languages. In our work, we focus on bounded CFG since the

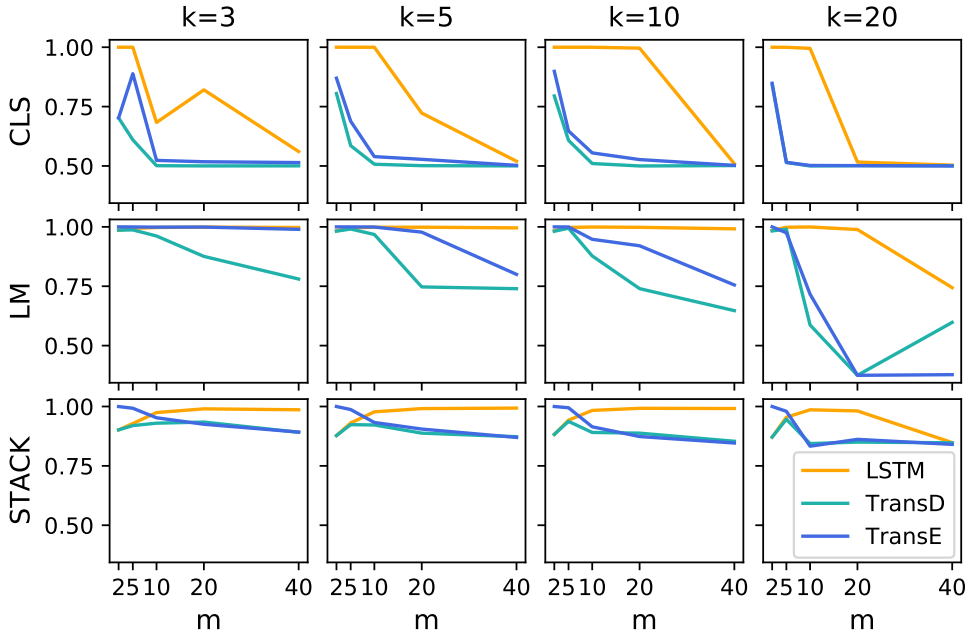


Figure 3.21. Performance on Dyck ($\alpha = 4$, single layer)

capacity of learning regular languages is widely agreed upon while there are disputes on the CFG level.

Finally, observing the defects of both LSTM and Transformer in learning CFG and algorithmic tasks, many works propose to use external memory to enhance the LSTM model [38, 76, 140], introduce recurrence in the Transformer network [40], and design specialized architectures [55, 57, 135, 136]. Though it's commonly believed that LSTM with finite memory, *i.e.* hidden states, can not handle CFGs which requires infinite stack spaces, we investigate the capacity of LSTM and transformer in bounded CFGs that requires finite-size stack and conclude that finite memory is not the bottleneck of LSTM capacity in learning CFGs.

3.7 Conclusion

We illustrate that only the state and stack prediction accuracy trained with dense supervision and explicit decomposition regularizer are the fair and stable metric to compare the empirical representation power of LSTM and the Transformer network. Then we conclude

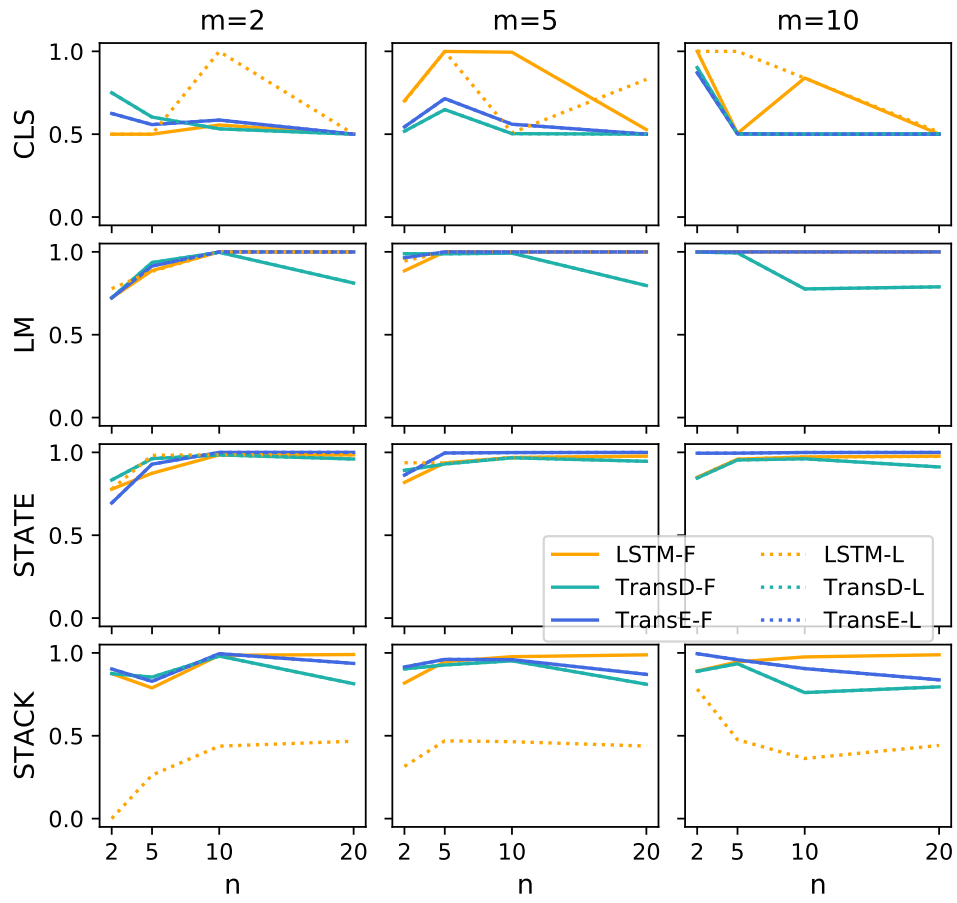


Figure 3.22. 4-layer model with $\alpha = 4$ on $(wcw^r)^n$, $k = 3$.

Table 3.2. Shift-reduce parsing of SCAN commands.

step	stack	unconsumed symbols	transition δ
0	[]	jump left and turn opposite left	–
1	[U]	left and turn opposite left	$\delta(left, \varepsilon) \to U$
2	[D]	and turn opposite left	$\delta(and, U) \to D$
3	[V]	and turn opposite left	$\delta(\varepsilon, D) \to V$
4	[S]	and turn opposite left	$\delta(\varepsilon, V) \to S$
5	[CP]	turn opposite left	$\delta(and, S) \to CP$
6	[CP, D]	opposite left	$\delta(turn, \varepsilon) \to U$
7	[CP, VP]	left	$\delta(opposite, D) \to VP$
8	[CP, V]		$\delta(left, VP) \to V$
9	[C]		$\delta(\varepsilon, [CP, V]) \to C$
Padded Sequence			
$[jump, left, and, \langle reduce \rangle, \langle reduce \rangle, turn, opposite, left, \langle reduce \rangle]$			

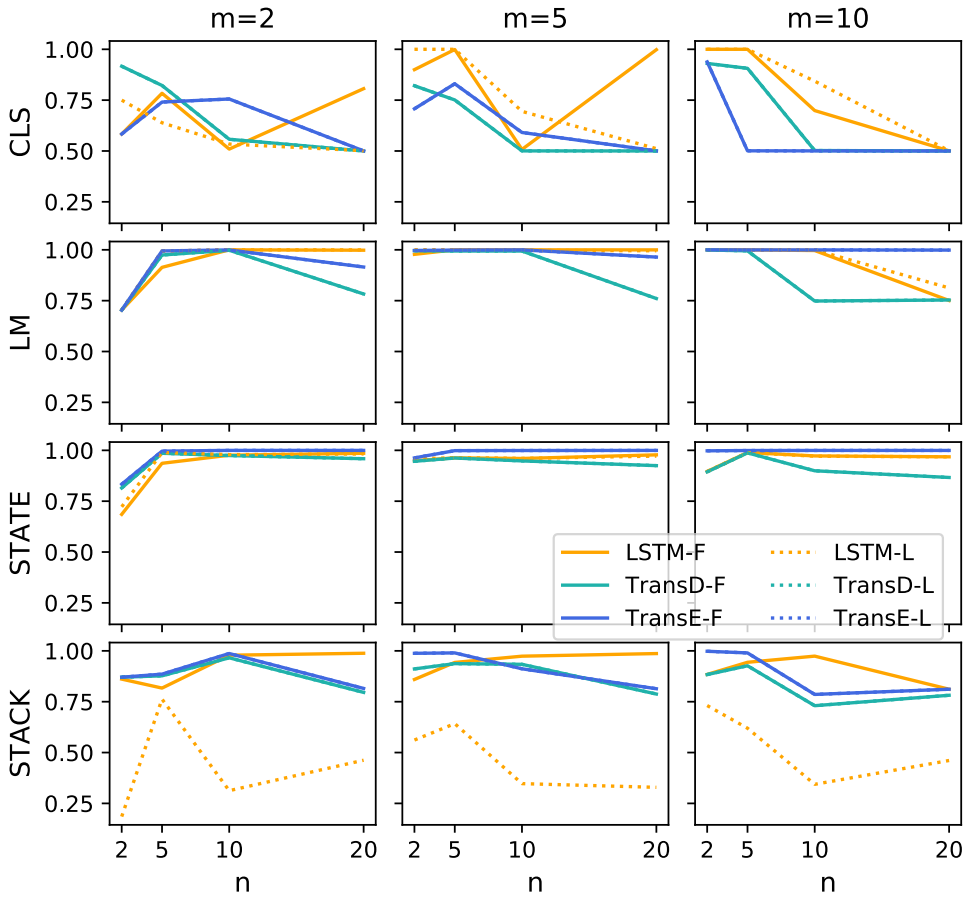


Figure 3.23. 4-layer model with $\alpha = 4$ on $(wcw^r)^n$, $k = 5$.

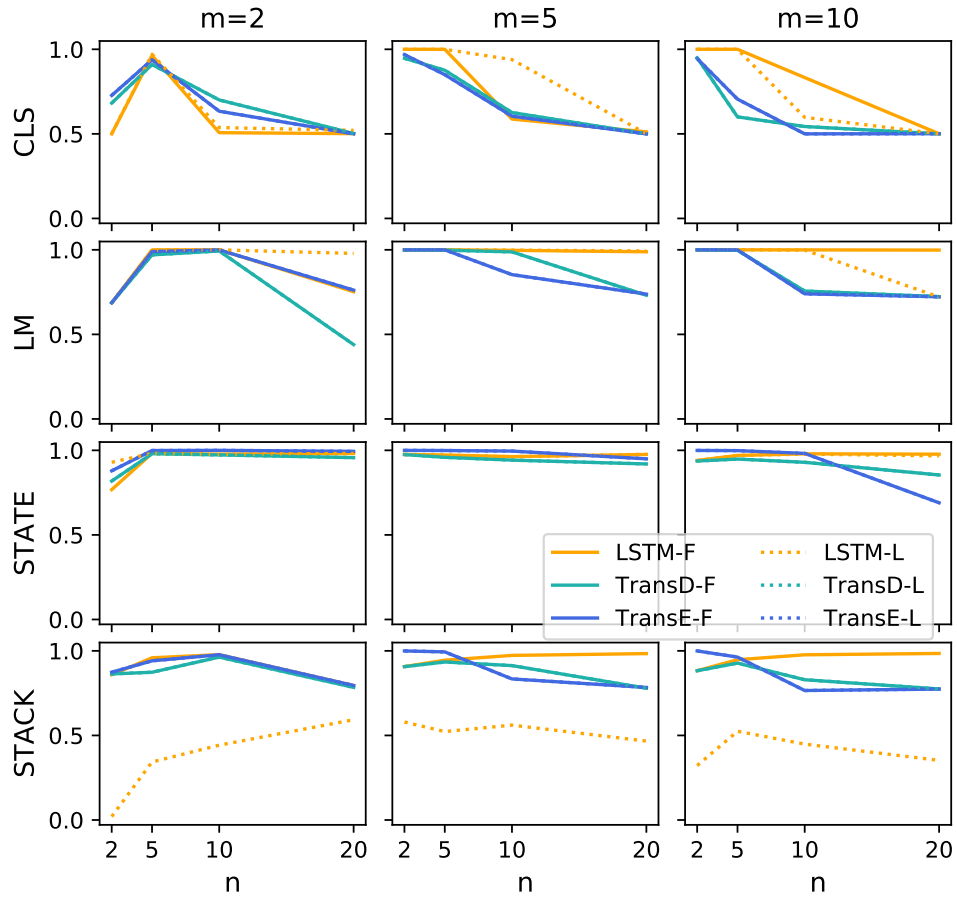


Figure 3.24. 4-layer model with $\alpha = 4$ on $(wcw^r)^n$, $k = 10$.

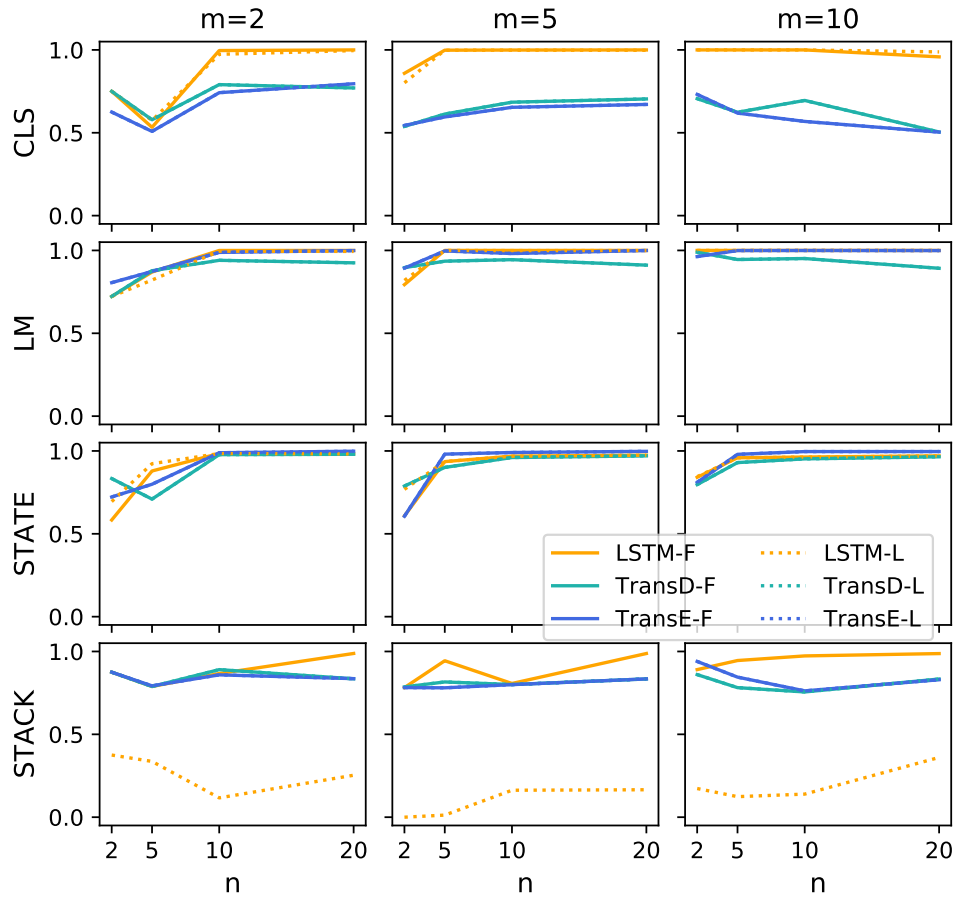


Figure 3.25. Performance on $(wcw^r)^n$, $k = 3$.

that both LSTM and Transformer network can simulate context-free languages with bounded recursion with a similar representation power, and unveiled the disadvantage of LSTM model in practice is from its inability to decompose the latent representation space.

3.8 Acknowledgement

This chapter, in full, is a reprint of the material as it appears in "Learning bounded context-free-grammar via LSTM and the transformer: Difference and the explanations.", Shi, Hui, Sicun Gao, Yuandong Tian, Xinyun Chen, and Jishen Zhao. In Proceedings of the AAAI Conference on Artificial Intelligence, vol. 36, no. 8, pp. 8267-8276. 2022. The dissertation

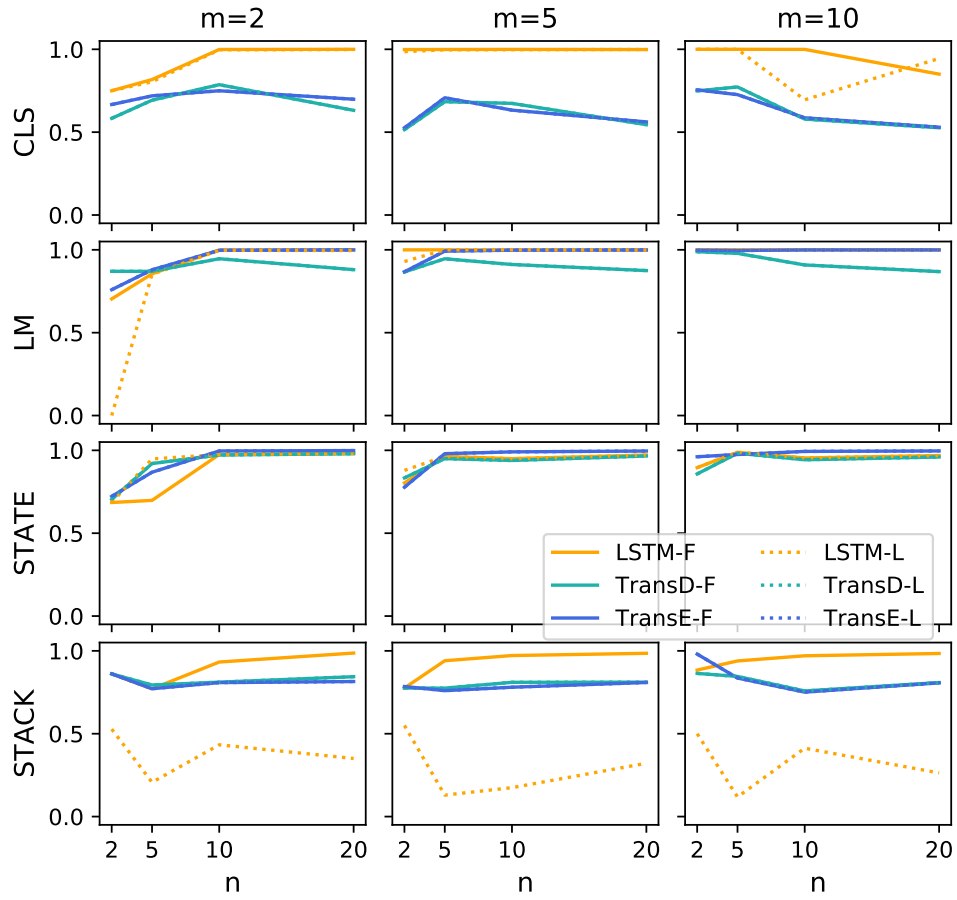


Figure 3.26. Performance on $(wcw^r)^n$, $k = 5$.

Table 3.3. Perplexity and parsing accuracy on SCAN. **E** denotes Transformer encoder and **D** denotes decoder.

Model	Perplexity	Accuracy
LSTM($\alpha = 1$, Forced)	2.705	91.30
LSTM($\alpha = 1$, Latent)	2.705	16.61
LSTM($\alpha = 4$, Forced)	2.706	91.30
LSTM($\alpha = 4$, Latent)	2.708	35.00
Transformer(D, $\alpha = 1$, Forced)	2.710	91.02
Transformer(D, $\alpha = 1$, Latent)	2.713	76.61
Transformer(D, $\alpha = 4$, Forced)	2.708	91.30
Transformer(D, $\alpha = 4$, Latent)	2.710	90.56
Transformer(E, $\alpha = 1$, Forced)	1.020	99.84
Transformer(E, $\alpha = 1$, Latent)	1.014	72.39
Transformer(E, $\alpha = 4$, Forced)	1.002	99.99
Transformer(E, $\alpha = 4$, Latent)	1.001	99.29

author was the primary investigator and author of this paper.

Chapter 4

Neural Architectures I: Weighted Multi-Interest User Representation

User embeddings (vectorized representations of a user) are essential in recommendation systems. Numerous approaches have been proposed to construct a representation for the user in order to find similar items for retrieval tasks, and they have been proven effective in industrial recommendation systems. Recently people have discovered the power of using multiple embeddings to represent a user, with the hope that each embedding represents the user's interest in a certain topic. With multi-interest representation, it's important to model the user's preference over the different topics and how the preference changes with time. However, existing approaches either fail to estimate the user's affinity to each interest or unreasonably assume every interest of every user fades at an equal rate with time, thus hurting the performance of candidate retrieval. In this paper, we propose the Multi-Interest Preference (MIP) model, an approach that not only produces multi-interest for users by using the user's sequential engagement more effectively but also automatically learns a set of weights to represent the preference over each embedding so that the candidates can be retrieved from each interest proportionally. Extensive experiments have been done on various industrial-scale datasets to demonstrate the effectiveness of our approach.

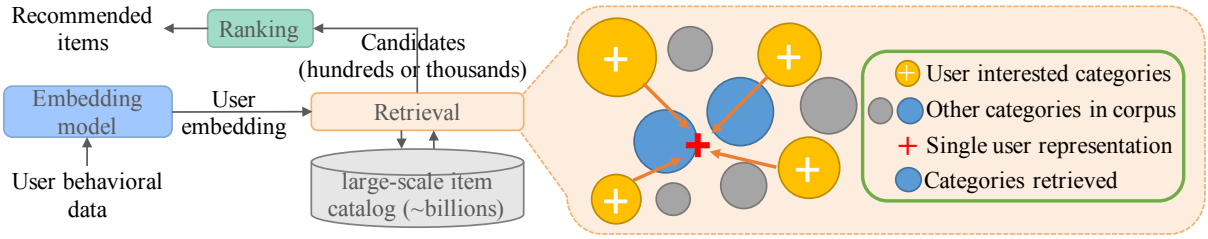


Figure 4.1. Mis-representation with single user embedding in the retrieve-then-rank framework.

4.1 Introduction

Today, the recommendation system is widely used in online platforms to help users discover relevant items and deliver a positive user experience. In industrial recommendation systems, there are usually billions of entries in the item catalog, which makes it impossible to calculate the similarity between a user and every item. The common approach is, illustrated in Figure 4.1, retrieving only hundreds or thousands of candidate items based on their similarity to the user embedding on an approximate level (*e.g.* inverted indexes, locality-sensitive hashing) without consuming too much computational power, and then sending the retrieved candidates to the more nuanced ranking models. Thus, finding effective user embedding is fundamental to the recommendation quality.

The user representations learned from the neural networks are proven to work well on large-scale online platforms, such as Google [30], YouTube [35], and Alibaba [148]. Mostly, the user embeddings are learned by aggregating the item embeddings from the user engagement history, via sequential models [63, 78, 116, 164]. These works usually rely on the sequential model, *e.g.* a Recurrent Neural Network (RNN) model or an attention mechanism, to produce a single embedding that summarizes the user's one or more interests from recent and former actions.

Recently researchers [47, 96, 111, 153] have discovered the importance of having multiple embeddings for an individual, especially in the retrieval phase, with the hope that they can capture a user's multiple interests. The intuition is quite clear: if multiple interests of a user are collapsed

into a single embedding, though this embedding could be similar to and can be decoded to all the true interests of the user, directly using the single collapsed embedding to retrieve the closest items might result in items that the user is not quite interested in, as illustrated in Figure 4.1.

Though, conventional sequential models like RNN or the Transformer network do not naturally produce multiple sequence-level embeddings as desired in the multi-interest user representation. Existing solutions fall into two directions: 1) **split-by-cluster** approaches first cluster the items in the user engagement history by category labels [92] or item embedding vectors [111] and then compute a representation embedding per cluster; 2) **split-by-attention** models adopt transformer-like architecture with two modifications. The query vectors in the attention are learnable vectors instead of the projections from the input and the results of each attention head are directly taken as multiple embeddings [20, 178]. The limitations of the two approaches are obvious: the split-by-cluster method works best with dense item feature [161]; and split-by-attention models bias towards the popular categories owing to its shared query vector among all the users and are inflexible to adjust the number of interests, which is fixed in the training phase as the number of attention heads.

Moreover, the existing multi-interest works ignore one important aspect: the weights of each embedding. In the retrieval stage, given the limited number of items to return, retrieving items from each embedding uniformly will cause a recall problem when the user clearly indicates a high affinity towards one or two categories. Some existing approaches, *e.g.* PinnerSage [111], use exponentially decayed weights to assign a higher score to interests that have more frequent and recent engagements. However, the methods still assume that in the same period, regardless of whether the interest is enduring or ephemeral, the level of interest decays equally for any user. Furthermore, these works also assume the number of embeddings to be fixed across all users. Not only is this hyperparameter costly to find, but also the assumption that all users have the same number of interests is questionable. Some dormant users can be well represented using one or two vectors, while others might have a far more diverse set of niche interests that requires tens of embeddings to represent.

In this paper, we propose Multi-Interest Preference (MIP) model that learns user embeddings on multiple interest dimensions with personalized and context-aware interest weights. The MIP model consists of a clustering-enhanced multi-head attention module to compute multiple interests and a feed-forward network to predict the weights for each embedding from the interest embedding as well as the temporal patterns of the interest. The clustering-enhanced attention overcomes the aforementioned shortcomings from two aspects: the query, key, and value vectors are projected from the user's engaged items, thus the output of the attention is personalized and minimized the bias toward globally popular categories; moreover, the clustering module can be applied before or after the multi-head attention, releasing the assumption that item features are pre-computed or the item-category labels are available. The main contribution of this paper and the experimental evidence can be summarized as follows:

- We propose a multi-interest user representation model that minimizes the bias towards popular categories and is applicable no matter if the item embeddings are pre-computed. MIP is successful in various industry-scale datasets (Section 4.4.1, 4.4.2); Section 4.4.6 reveals the bias resulting from the global query vector and the error resulting from a fixed number of clusters in the split-by-attention approaches, in comparison to MIP.
- In addition to the multi-facet vector representations of a user, MIP assigns weights to each embedding, which are automatically customized for each user interest. This approach improves the recall of candidate generation by retrieving more candidates from the most representative embedding. (Section 4.4.3).
- Although if the clustering algorithms require, MIP still asks for a number of clusters during the training phase, the number of clusters in MIP in the inference phase can be trivially increased or decreased without re-training of the model. And the experimental results (Section 4.4.7) show that re-configuring the number of clusters has an insignificant impact on the retrieval performance, thus allowing the system to trade off the storage and computation cost for better performance. Thus, MIP does not require prior knowledge of the number of interests of users during the model training phase.

Table 4.1. Comparison of MIP to existing recommendation models.

Name	User Embedding	Sequential Model	Additional Input	Preference Weight
GRU4Rec [63]	single	RNN	interaction session	N/A
TiSASRec [93]	single	time-aware self-attention	timestamps	N/A
BERT4Rec [137]	single	self-attention	–	N/A
MIND [92]	multiple	label-aware self-attention	category labels	✗
ComiRec [20]	multiple	global-query attention	–	✗
PinText2 [178]	multiple	shared global-query attention	–	✗
PinnerSage [111]	multiple	N/A	–	heuristic
MIP	multiple	time-aware self-attention	timestamps	learned

4.2 Related Work

This work relates to two important aspects of existing recommendation systems: sequential models and the multi-interest framework.

Sequential models. A basic consensus in the recommendation system is that user embeddings should be inferred from the user's historical behavior, and thus the sequential models have been at the heart of recommendation models. A typical and classical sequential model is the Markov Chain [60, 120]. While Markov Chain captures short-term patterns of engagement sequence well, it fails to make the recommendation that requires memorizing long sequences. With stronger representation power on long sequences, Recurrent Neural Networks (RNNs) have been adopted for learning user embedding from arbitrarily long sequences, *e.g.* GRU4Rec [63] and others [43, 159]. Besides the standard RNN models, specialized recurrent units are proposed to meet the special need of incorporating certain information, *e.g.* user demographic information [44], global context [156], interest drifts with time [28], and interaction session [63]. Recently, the success of the Transformer network [145] has brought revolution to sequential modeling tasks [115, 129] and has been soon adapted to the recommendation models, *e.g.* ComiRec [20], BERT4Rec [137], TiSASRec [93], SASRec [78], MIND [92], PinText2 [178], and also our MIP.

Multi-interest user representation. Representing users by multiple embeddings greatly improves the recommendation quality, but not every existing recommendation model can easily

extend to a multi-interest framework. Classical collaborative filtering and matrix factorization methods do not naturally produce multiple user embeddings, and so do RNNs and attention-based models. To discover multiple interests from user engagement history, heuristic methods [72, 168] and unsupervised learning methods like clustering [111, 146] and community mining [147, 166] have been adopted. Besides, researchers have made efforts to modify the existing neural networks to produce multiple results, for instance, the capsule network [20, 92, 122] and multi-head attention models [20, 93, 176]. However, they require an estimation of the number of interests of users as a hyperparameter and do not learn the weight of interests. Therefore, unlike MIP, they produce an equal number of clusters for every user and treat each interest with uniform importance.

Relationship to previous works. The motivation of MIP is to acquire weighted multiple user embeddings with standard self-attention but without explicit item-category labels. ComiRec and PinText2 use global-query attention to produce multiple embeddings, which introduces a bias toward frequent items or popular categories and the phenomenon is shown in Section 4.4.6. Furthermore, they also predefine a number of interests that is uniform for all the users. TiSASRec and BERT4Rec adopt self-attention but can not learn multiple embeddings. MIND relies on the category labels to produce multiple embeddings from self-attention and capsule networks. However the category labels are sometimes unavailable or vague in other applications, *e.g.* YouTube and Pinterest. PinnerSage produces multiple embeddings without category labels, but requires pre-computed item embeddings. The comparison are summarized in Table 4.1.

4.3 Methodology

In this section, we formulate the recommendation problem and the neural architecture to model the multiple user interests with preference weights in detail.

Table 4.2. Notations

Notations	Description
$\mathscr{I}, \mathscr{U}$	Item set and user set
$\mathscr{S}$	Abbreviation of user engagement history $\mathscr{S}^u$
l	Abbreviation of l_u, length of $\mathscr{S}$
d	The item embedding dimension
d_{model}	Projected key/query vector dimension
h	Attention head superscript
$\boldsymbol{p}$	An item embedding, $\boldsymbol{p} \in \mathbb{R}^d$
$\boldsymbol{p}_j, t_j$	Short form of the $\boldsymbol{p}^u_{t_j}$ and t^u_j
$W^h_q, \boldsymbol{b}^h_q$	Query projection weights and bias
$W^h_k, \boldsymbol{b}^h_k$	Key projection weights and bias
$\boldsymbol{M}$	Attention mask matrix
Λ	Maximum number of interests per user
$\mathscr{C}$	Cluster assignment, $\mathscr{C} \in \mathbb{R}^l$. $\mathscr{C}_{[i]} = \lambda$ if i-th item belongs to λ-th cluster
$\boldsymbol{L}_\lambda$	The set of indices of items in the λ-th cluster.
$\boldsymbol{z}_\lambda, Z$	User embedding vector(s)
$\mathbb{1}_{[cond]}$	1 if *cond* is true, 0 otherwise
$\boldsymbol{\tau}_{[m]}$	m-th digit in the vector $\boldsymbol{\tau}$
$[;]$	Vector concatenation operator

4.3.1 Problem Statement

Let $\mathscr{I}$ denote the collection of items and $\mathscr{U}$ denote the set of users. The interaction sequence of a user $u \in \mathscr{U}$ is represented as $\mathscr{S}^u$ with a list of item IDs $(v^u_{t_1}, v^u_{t_2}, ..., v^u_{t_{l_u}})$ and timestamps $(t^u_1, t^u_2, ..., t^u_{|l_u|})$. Each item $v^u_i \in \mathscr{I}$ is associated with an item embedding $\boldsymbol{p}^u_i \in \mathbb{R}^d$ and l_u is the length of the interaction sequence.

The objective is to learn a set of user embeddings $\boldsymbol{z}^u_\lambda \in \mathbb{R}^d$ and their weights w^u_λ ($\lambda = 1, ..., \Lambda$) for each user u. Since the user representation is learned only from the history of that user, hereinafter, we omit the superscript for u in both the input and output sides for simplicity. Notations are summarized in Table 4.2.

Item embedding $\boldsymbol{p}_j$ can be either represented by item metadata features or treated as model parameters and learned from the data. When items have dense metadata features (*e.g.* text embeddings), we will use them as the item embedding and focus on learning the user

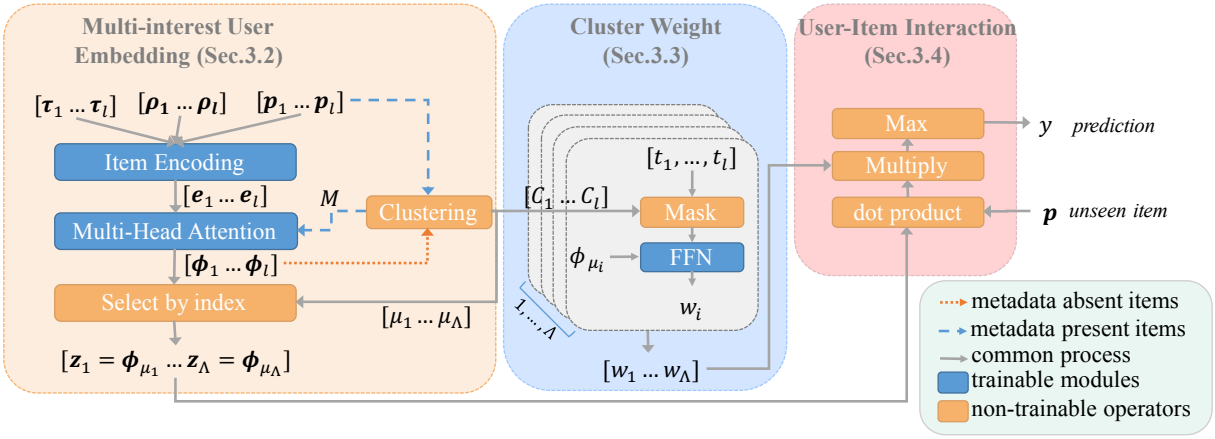


Figure 4.2. An overview of MIP architecture.

representation. When items are only represented by an ID and do not have other metadata, we will learn the item embedding table. MIP is able to handle both cases and eventually learn users' multi-interest embeddings and their weights.

4.3.2 Process Overview

Overall, the proposed model consist two parts to learn the multi-interest user representation and cluster weights. Figure 4.2 shows the model architecture. The input is the user engagement history containing item embeddings $[\boldsymbol{p}_1 ... \boldsymbol{p}_l]$), temporal encoding $[\boldsymbol{\tau}_1 ... \boldsymbol{\tau}_l]$, and positional encoding ($[\boldsymbol{\rho}_1 ... \boldsymbol{\rho}_l]$). The multi-interest user embedding module produces Λ embeddings, where Λ is decided by the clustering method or as a hyperparameter. With clustering and multi-interest representation, the cluster weight module will then estimate the cluster weights for each cluster. Finally, the multi-interest embeddings with corresponding weights are combined to predict the user's interest in an unseen item $\boldsymbol{p}$. The processes with item metadata absent/present are shown with different arrows. Respectively, the rest of this section will introduce the sequential module for learning a set of user embedding (Section 4.3.3), the module for cluster weight prediction (Section 4.3.4), and the final prediction modules that combines them together (Section 4.3.5). An unsupervised clustering step is required in both parts, which is utilized differently in metadata present and absent feature scenario.

4.3.3 Multi-Interest User Modeling

Item Representation.

For the j-th item in the user engagement history, we represent the item by $\boldsymbol{p}_j$ and it's either learned (when item metadata is absent) or copied from the item metadata feature (when item metadata is present). In addition to $\boldsymbol{p}_j$, the sequential order and relative timestamps of the interactions are represented by positional encoding and temporal encoding respectively as $\boldsymbol{\rho}$ and $\boldsymbol{\tau}$. Following Vaswani et al. [145], the odd digits $(2m+1)$ and even digits $(2m)$ on the encoding vectors are given by:

$$\begin{aligned}
\boldsymbol{\tau}_{[2m]}(t_j) &= sin\big(t_j/(\tau_{max})^{2m/m_t}\big) \\
\boldsymbol{\tau}_{[2m+1]}(t_j) &= cos\big(t_j/(\tau_{max})^{2m/m_t}\big) \\
\boldsymbol{\rho}_{[2m]}(j) &= sin\big(j/(\rho_{max})^{2m/m_p}\big) \\
\boldsymbol{\rho}_{[2m+1]}(j) &= cos\big(j/(\rho_{max})^{2m/m_p}\big)
\end{aligned} \tag{4.1}$$

The hyper-parameters are set as $\tau_{max} = \rho_{max} = 1 \times 10^4$, and $m_t = m_p = 1$. The timestamps unit is *day*. In the Section 4.4.5, other encodings forms are compared, and show that the design used in Equation 4.1 has a slight advantage.

Finally, item ID embedding is concatenated with the positional and temporal encodings to produce the final representation of an interaction:

$$\mathbf{e}_j := [\boldsymbol{p}_j; \boldsymbol{\tau}(t_j); \boldsymbol{\rho}(j)] \tag{4.2}$$

User Representation. Our user representation is built upon the multi-head self-attention module [145]. Using the interaction embedding as above, for each attention head h, we define the attention weight between interaction i and j as

$$a^h_{i,j} = \frac{\exp(s^h_{i,j})}{\sum_{k=1}^{l} \exp(s^h_{i,k})} \tag{4.3}$$

where $s^h_{i,j}$ is the dot product between the projected query of j and the projected key of i:

$$s^h_{i,j} = \Big((W^h_q \boldsymbol{e}_j + \boldsymbol{b}^h_q)^\top \cdot (W^h_k \boldsymbol{e}_i + \boldsymbol{b}^h_k)\Big) / \sqrt{d_{model}} \tag{4.4}$$

In order to build the user's multi-interest embedding, we need to cluster the items in the user sequence. When we use item metadata features as $\boldsymbol{p}$, we pre-compute the item clusters $L_1, \cdots, L_\Lambda$ where each L_λ is the set of item IDs that belong to λ-th cluster. Let $\mathscr{C}$ denote the mapping from item ID to cluster assignment, *i.e.* $\mathscr{C}_{[j]} = \lambda$ if $j \in L_\lambda$. Given the cluster information, we have the advantage of summarizing similar items into a single representation. Specifically, when the context vector only attends to items within the same cluster as the current item, we force that vector to contain only the information from that cluster. Naturally, a mask is introduced to enforce such constraint: let $M \in \{0,1\}^{l \times l}$ be the mask matrix where $M_{i,j} = \mathbb{1}_{[\mathscr{C}_{[i]} = \mathscr{C}_{[j]}]}$ (and 0 otherwise). Each attention head h produces the context vector at position j by aggregating the sequence as:

$$\boldsymbol{\phi}^h_j = \sum_{i=1}^{l} a^h_{i,j} M_{i,j} \boldsymbol{p}_i \tag{4.5}$$

To process the context vector from all attention heads, a dropout layer and a feed-forward network (FFN) are applied, and the output vector is computed as

$$\boldsymbol{\phi}_j = FFN(Dropout([\boldsymbol{\phi}^1_j; ...; \boldsymbol{\phi}^H_j])), \quad j = 1, \cdots, l \tag{4.6}$$

The $FFN()$ consists of two fully-connected linear layers with a hyperbolic tangent activation function after the first layer, *i.e.* $FFN(x) = W(tanh(W'x + b')) + b$.

When item metadata is absent and $\boldsymbol{p}$ need to be learned, the mask M will be an all-ones matrix, and output vectors are still computed according to Equation 4.5 and 4.6. Then the item clusters are computed from $\boldsymbol{\phi}$ instead of from $\boldsymbol{p}$. We still use $\mathscr{C}$ to represent such cluster assignment, and it will be used to define the user's multi-interest embedding as below.

So far, the multi-head attention module has produced l vectors $\boldsymbol{\phi}_1, ..., \boldsymbol{\phi}_l$, and each $\boldsymbol{\phi}_j$

uses $\boldsymbol{p}_j$ as the (unprojected) query. We will build the multi-interest user embedding by selecting the Λ context vectors that represent each cluster. Denote the position of the last item in each cluster λ as μ_λ (*i.e.* $\mu_\lambda = \text{argmax}_j(\mathscr{C}_{[j]} = \lambda)$), we will take context vector at that position to represent the λ-th user embedding. In sum, the multi-interest user embedding is

$$Z = [\mathbf{z}_1^\top; ...; \mathbf{z}_\Lambda^\top] = [\boldsymbol{\phi}_{\mu_1}^\top; ...; \boldsymbol{\phi}_{\mu_\Lambda}^\top] \in \mathbb{R}^{\Lambda \times d} \tag{4.7}$$

Each $\boldsymbol{z}_\lambda$ attends to only items that belong to the same cluster as the item on position μ_λ.

4.3.4 Cluster Weight Modeling

Besides the multi-interest user embedding, it's also likely that a user favors each interest unequally. As mentioned earlier, ranking these interests correctly can greatly benefit the candidate generation task given its limited budget. PinnerSage [111] uses an exponential-decay heuristic function to represent the weight for a cluster, following the assumption that more recent interactions should contribute more to the cluster weight. While we believe that the intuition is generally true, it would be best for the model to automatically learn the role of timestamps from the data. We also incorporate the user embeddings on that cluster's dimension ($\boldsymbol{z}_\lambda$) into cluster weight modeling, since certain categories of interest can have different impacts on the cluster weights as well. Therefore we design the cluster weights to be a function of the recency of the interaction and user embeddings. We model the weight of cluster λ as

$$w_\lambda = FFN([\boldsymbol{z}_\lambda;\ \mathbb{1}_{[C_{[1]}=\lambda]} \cdot \boldsymbol{\tau}_1; ...; \mathbb{1}_{[C_{[l]}=L_\lambda]} \cdot \boldsymbol{\tau}_l]) \tag{4.8}$$

The first part inside the FFN is the user's embedding on cluster λ. The second part inside the FFN serves as a mask that retains only the timestamps of relevant items that belong to cluster λ. We will discuss how to learn these weights in the next subsection.

4.3.5 User-Item Interaction Modeling

On the item level, intuitively, a user will engage with an item as long as the item matches *at least one* of his/her interests (not *all*). For example, a user who is interested in both running and home decor purchased a lawn mower, and this behavior will be explained by the user's embedding of the "home decor" cluster (*i.e.* $\boldsymbol{z}_{home}$) and has nothing to do with the embedding of the "running" cluster (*i.e.* $\boldsymbol{z}_{running}$). In other words, the similarity of $\boldsymbol{z}_{home}$ and $\boldsymbol{p}_{lawn\ mower}$ should be high, and the similarity of $\boldsymbol{z}_{running}$ and $\boldsymbol{p}_{lawn\ mower}$ should not even matter. Therefore, when measuring the user-item affinity, we should consider the one user embedding that is the most similar (*e.g.* highest cosine similarity) to the item.

On the cluster level, we need another factor to explain a user's behavior towards different clusters. This can no longer be represented simply by the semantic similarity in the embedding space any more. When the user purchases 20 items in the "home decor" cluster and 5 items in the "running" cluster, it does not indicate that the similarities between $\boldsymbol{z}_{home}$ and these 20 items are higher than the similarities between $\boldsymbol{z}_{running}$ and these 5 items (in fact, all of the similarities should be as high as possible). Therefore, we will multiply the user-item affinity by the cluster weight here to represent a user's intensity towards different clusters.

Considering the arguments above, we propose the likelihood that a user (represented by $Z = [\mathbf{z}_1^\top; ...; \mathbf{z}_\Lambda^\top]$) interacts with an item (represented by $\boldsymbol{p}$) as follow:

$$y = \max\{w_\lambda \cdot (\boldsymbol{z}_\lambda \cdot \boldsymbol{p})\}_{\lambda=1}^{\Lambda} \tag{4.9}$$

Given the set of items with positive label ($\{\mathscr{I}_+^u\}_{u\in\mathscr{U}}$) and negative label ($\{\mathscr{I}_-^u\}_{u\in\mathscr{U}}$), the negative log-likelihood (NLL) loss of our model can be written as:

$$\mathscr{L} = -\frac{\sum_{u\in\mathscr{U}} \left(\sum_{\boldsymbol{p}_i\in\mathscr{I}_+^u} \log(y_i^u) + \sum_{\boldsymbol{p}_i\in\mathscr{I}_-^u} \log(1-y_i^u) \right)}{\sum_{u\in\mathscr{U}} \left(|\mathscr{I}_+^u| + |\mathscr{I}_-^u| \right)} \tag{4.10}$$

Table 4.3. Dataset statistics.

	Amazon	MovieLens	Taobao
# Items	425,582	15,243	823,971
# Interactions	51M	20M	100M
# Training Seq	57,165	127,212	343,171
# Test Seq	5,000	5,000	10,000
# Validation Seq	5,000	5,000	10,000

4.4 Experiments

We conduct an exhaustive analysis to demonstrate the effectiveness of MIP on the data from Pinterest, one of the largest online content discovery platforms, and a few public datasets. We will divide our discussions to two categories: (1) learning item ID embeddings (Section 4.4.1) and (2) using item metadata features as is (Section 4.4.2). Finally, an ablation study is done on different components of the model (Section 4.4.3).

4.4.1 Learning Item ID Embeddings

We first evaluate MIP on learning from collaborative filtering datasets, where the item features are absent and will be learned from the user-item interactions.

Dataset. Three public datasets are used: Amazon-book [1] (hereinafter, Amazon), Taobao[2], and MovieLens[3]. We adopt a 10-core setting as previous works [93, 149] and filter out items that appear less than 10 times in the dataset. We then split each user's engagement history to non-overlapping sequences of length 100, and use the first 50 items to learn the user embeddings and the last 50 items as labels (as used in Cen et al. [20]). Any sequence shorter than this threshold are discarded. For each sequence, another 50 negative samples are uniformly sampled at random from the items that the user does not interact with. Our goal is to rank the positive items (that users have actually interacted with) higher than the negative items (random). The dataset statistics are listed in Table 4.3.

[1] https://jmcauley.ucsd.edu/data/amazon/

[2] https://tianchi.aliyun.com/dataset/dataDetail?dataId=649

[3] https://www.kaggle.com/grouplens/movielens-20m-dataset

Baseline and model configuration. We compare several open-sourced baseline models with MIP. For fair comparison, we set up the configurations as follow: (1) item and user embedding vectors have the same size ($d = 32$); (2) the number of attention heads is the same ($H = 8$) if the model includes a multi-head attention module; (3) the baseline models should have similar or more parameters than MIP. We let the hidden size in GRU4Rec [63] be 128, the key and query projected dimension (d_{model} in Equation 4.3) is labeled in place with the results, and if the model contains a position-wise FFN (Equation 4.6), it will be a two-layered fully-connected structure with a hidden size of 32 each. The BERT4Rec model [137] is originally proposed to predict the item directly as a classification task, so we take its last BERT output as the user embedding to compute the similarity between user and item, and train with the NLL loss. We disabled the session-parallel mini-batch in these models since the session information is absent. We also replace the text encoder in the PinText2 [178] with an item embedding layer since the inputs in our experiments are items instead of texts.

Training setup. All the models are trained for 100 epochs on a NVIDIA Tesla T4 GPU with an early stop strategy that stops the training when validation AUC does not improve for 20 epochs. The clustering method used in MIP is the Ward clustering algorithm Ward Jr [151].[4] We compare other clustering methods in Section 4.4.7.

Table 4.4. Performance on public datasets. *Params* excludes the parameters in the item embedding table. Recall and nDCG are measured at top-50 items. See Section 4.4.8 for latency measure details.

Model	Amazon			Taobao			MovieLens		
	AUC	recall	nDCG	AUC	recall	nDCG	AUC	recall	nDCG
GRU4Rec	0.682	0.635	0.678	0.816	0.745	0.794	0.961	0.903	0.934
BERT4Rec	0.681	0.632	0.678	0.815	0.745	0.794	0.960	0.901	0.928
BERT4Rec	0.721	0.665	0.698	0.815	0.745	0.794	0.960	0.902	0.941
PinText2	0.558	0.541	0.608	0.716	0.669	0.691	0.883	0.817	0.782
TiSASRec	0.721	0.667	0.704	0.815	0.744	0.794	0.960	0.902	0.928
ComiRec	0.717	0.674	0.704	0.709	0.656	0.698	**0.963**	0.907	**0.979**
MIP	**0.805**	**0.789**	**0.781**	**0.885**	**0.884**	**0.909**	0.930	**0.933**	0.954

[4]We adopted the scikit-learn implementation. https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html, with n_cluster=5, and other default arguments.

Table 4.5. Model hyperparemter setting and inferentce runtime latency on public datasets. See Section 4.4.8 for latency measure details.

Model	Parameters	Layers	d_{model}	Latency (ms)
GRU4Rec	66338	1	–	1.15
BERT4Rec	50242	1	64	38.34
BERT4Rec	55426	2	32	57.53
PinText2	69634	1	256	14.46
TiSASRec	67586	2	64	14.54
ComiRec	67586	1	256	14.61
MIP	49347	1	32	40.05

Results and analysis. The performance is summarized in Table 4.4. MIP has a better performance on Amazon and Taobao datasets and is trivially worse than GRU4Rec and ComiRec in AUC on MovieLens. Intuitively, the purchase behavior on e-commerce websites (Amazon, Taobao) can be largely explained by the user's interest in multiple categories or brands, while movie-watching is driven more by a movie's popularity and quality rather than the category. Since all models have very close performance, MIP is still a competitive approach in applications that do not support the strong multi-interest assumption.

4.4.2 Using Item Metadata Features

Dataset. The dataset contains user engagement history collected from Pinterest, an image-sharing and social media service that allows users to share and discover visual content (images and videos). The interactions between a user and an item (also referred to as a *pin*) are categorized into *impression* (pin is shown to the user), *clickthrough* (user clicks the pin), *re-pin* (user saves the pin into their board collection), and *hide* (user manually hides the pin). In total, there are 38 million interactions from 510 thousand users during three weeks of time. Each pin is represented as a 256-dimension feature extracted by the PinSage model [163].

User's engagement sequences are processed in a similar way as the public datasets, except that we enforce a one-day gap between the inputs and labels, because adjacent user engagements are usually very similar which makes the prediction task easy. Intuitively, we can

use clickthrough and re-pin as the positive label, and hide (which is less often) and impression (without click or re-pin) as the negative label, but since impressions are also recommended to the user at some point, they are likely to be relevant to the user as well, and thus correlate with the positive data. In order to alleviate this bias, we introduce the *random* negative data where pins are sampled from the whole set of pins. The entire negative dataset will consist of 50% *observed* negative data (hide and impression), and 50% *random* negative data.

Baselines and model configuration. We compare the multi-interest models PinnerSage and ComiRec, and the single-embedding model TiSASRec with the same setting as in Section 4.4.1.

Results and analysis. As shown in Table 4.6, MIP outperforms all the state-of-the-art multi-interest sequential models.

- PinnerSage shares the same clustering algorithm with MIP, but differs in that 1) each cluster embedding is represented by the medoid of all of its item embeddings; and 2) the cluster weights are heuristic-based (not learned from the model). Instead, MIP learns the cluster representations and weights collectively from data, and thus has a clear advantage over PinnerSage.
- TiSASRec has a similar attention module as MIP, except only using the single last attention output as the user embedding. The comparison confirms the necessity of multi-interest representation, as in ComiRec and MIP.
- Compared to ComiRec, MIP interestingly shows that self-attention has stronger representation power than attention with global query. In Section 4.4.6, we further use synthetic data to illustrate the fundamental difference between the two types of models.

4.4.3 Ablation Study on Interest Weights

This section focuses on the claim that the user's preference should be dynamically learned from the temporal pattern by demonstrating the effectiveness of the learned personalized cluster weights module of MIP.

Table 4.6. Performance on the Pinterest dataset.

	precision@20	recall@20	AUC	NLL
PinnerSage	0.740	0.296	0.815	1.033
TiSASRec	0.798	0.312	0.850	0.478
ComiRec	0.864	0.345	0.875	0.407
MIP	**0.882**	**0.353**	**0.893**	**0.377**

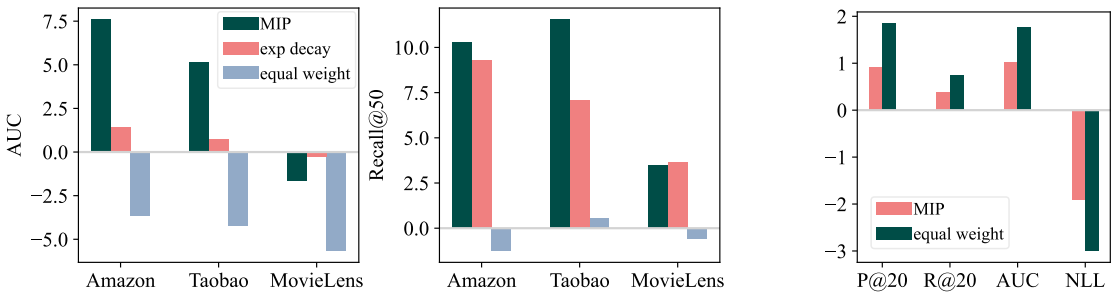
Table 4.7. MIP on MovieLens with different loss functions.

Loss Function	NLL		Triplet (α =0.2)		Triplet (α =0.5)		Triplet (α =0.8)	
Performance	AUC	R@50	AUC	R@50	AUC	R@50	AUC	R@50
	0.930	0.933	0.885	0.882	0.903	0.901	0.906	0.906

To validate the assumption that the preference trends (weights of multiple interests) change from user to user, we compare the MIP with two variants that disable the cluster weight module. The first one (referred to as *MIP (Equal Weight)*) constantly assigns 1 to the cluster weights, leading to an equally weighted multi-interest user representation. Another one (referred to as *MIP (Exp Decay)*) uses an exponential-decay heuristic weights given by $w_\lambda = \sum_{i:C_i \in \lambda} exp(-\varepsilon(t_{now} - t_i))$[111]. We let t_{now} be the last user engagement time t_{last_item}, since in practice it's unrealistic to update the weights in real-time. According to Pal et al. [111], we also set $\varepsilon = 0.01$, which balances well between emphasizing recently engaged items and accentuating frequently engaged categories. We let the number of clusters be $\Lambda = 5$. We calculate performance on these model variants and show the relative percentage difference from the best baseline model in Figure 4.3. Note that smaller NLL means better performance. In all experiments, the AUC and recall can benefit from learning cluster weights (MIP). The second best variant is usually having an equal weight for all clusters.

4.4.4 Ablation Study on Loss function.

We claim the NLL is adequate for MIP training, and compared the choices of loss function in this section. Beside the NLL loss function in Equation 4.10, triplet loss is also widely used in recommendation system and contrastive learning. Let the y^+ be the similarity prediction



(a) Public datasets (b) Pinterest dataset

Figure 4.3. Cluster weight variants performance (difference to the best baseline models, in the unit of 10^{-2})

between the user representation and a positive item, and y^- be the predicted similarity between user representation and a negative item. The triplet loss is given by:

$$\mathscr{L}_\alpha = \sum (y^+ - y^- + \alpha)_{\mu \in \mathscr{U}} \tag{4.11}$$

where α is a hyperparameter of the positive-negative margin. With triplet loss, we added a linear factor β (*learned*) in Equation 4.9, (*i.e.* $y = max\{\beta w_\lambda \cdot (\boldsymbol{z}_\lambda \cdot \boldsymbol{p})\}_{\lambda=1}^{\Lambda}$) to re-scale the similarity since y is unbounded and makes the choice of α to be hard. We let the MIP train on the MovieLens dataset to investigate the impact on the performance of loss function choice. The results in Table 4.7 illustrate that the triplet loss marginally underperforms the NLL loss we used.

4.4.5 Ablation Study on Positional and Temporal Encoding

In Equation 4.2, the sequential (positional) and temporal information are encoded and included in the self-attention module to produce the multi-interest representations. The motivation is that given items from the same category, the recent ones might better represent the user's current interest than the obsolete items. We verify 1) if the incorporation of positional and temporal encoding is critical to the performance; 2) how the encoding (Equation 4.1) method affects the performance.

Configuration. The MIP are configured on the two set of choices: the Equation 4.2 can

be configured alternatively:

- item embedding only: $\boldsymbol{e}_j = \boldsymbol{p}_j$.
- + positional: $\boldsymbol{e}_j = [\boldsymbol{p}_j; \boldsymbol{\rho}(j)]$, where $\boldsymbol{\rho}(j)$ is given by Equation 4.1.
- + temporal: $\boldsymbol{e}_j = [\boldsymbol{p}_j; \boldsymbol{\tau}(t_j)]$, where $\boldsymbol{\tau}(t_j)$ is given by Equation 4.1.
- + positional and temporal: Equation 4.2 and Equation 4.1

and there are several other choices of temporal encodings other than the sinusoidal form in the Equation 4.1:

- One-hot [176]: Create exponential buckets $[0,b)$, $[b,b^2)$, $\cdots$, $[b^{k-1},\infty)$ with base b, and encode the timestamp as an one-hot vector, *i.e.* $\tau_i = lookup(buckets(t))$.
- Two-hot [128]: Similarly create exponential boundaries $\{0,\ b,\ b^2,\ \cdots,\ b^{k-1},\ \infty\}$, and encode the timestamp as $\boldsymbol{\tau}_i = log_b(t) - i$ and $\boldsymbol{\tau}_{i+1} = i + 1 - log_b(t)$, where $b^i \leq t < b^{i+1}$.

Dataset and Results. The options to include positional and temporal information are evaluated on all the dataset (Table 4.8), and the encodings methods are compared exhaustively on Pinterest dataset (Table 4.9).

Analysis. Two conclusions can be made from Table 4.8 and Table 4.9: 1) including both temporal and positional information is a safe option, which has the best performance on Amazon and Pinterest and marginally (< 0.01) worse performance on Taobao and MovieLens; and 2) the model is insensitive to encoding methods.

4.4.6 Ablation Study on Variants of Attention Mechanism

To interpret the performance improvement of our models against other attention models that have been applied in the recommendation system, we further construct a synthetic dataset and visualize the internal attention and the user representations aggregated from different attentions.

Table 4.8. Ablation study on the positional and temporal encoding on public datasets. (in 10^{-2})

Dataset	Configuration	AUC	Recall
Amazon	item embedding	76.34	73.75
	+positional	76.19	74.28
	+temporal	78.59	76.94
	+both	**79.31**	**78.22**
Taobao	item embedding	82.06	81.75
	+positional	86.54	**86.22**
	+temporal	**86.72**	86.56
	+both	86.59	85.83
MovieLens	item embedding	**95.26**	**94.45**
	+positional	95.01	94.31
	+temporal	94.96	94.19
	+ both	94.61	94.12

Table 4.9. Influence of temporal and positional encoding in attention on the performance in MIP

Configuration	AUC	NLL
item embedding	0.8923	**0.377**
+ positional	0.8846	0.386
+ temporal (one-hot)	0.8850	0.388
+ temporal (two-hot)	0.8846	0.385
+ temporal (sinusoid)	0.8921	**0.377**
+ both (one-hot)	0.8861	0.382
+ both (two-hot)	0.8852	0.387
+ both (sinusoid)	**0.8926**	**0.377**

Synthetic dataset. Without loss of generality, we assume there are G global clusters in the corpus, representing different global categories, each of which is a d-dimensional Gaussian distribution. Each user is interested in up to k ($< G$) categories, referred to as user clusters. We generate the oracle user interest model by sampling no more than k clusters from G global clusters following a multinomial distribution. Then each of the items in the user engagement history is sampled in two steps: uniformly sample one cluster from user clusters, then sample from the d-dimensional Gaussian distribution. Note that in the synthetic model, the item-to-cluster affinity is measured in Euclidean distance, while in the recommendation model, the affinity is decided by cosine distance. To eliminate this discrepancy, we force the Gaussian distributions to center

Table 4.10. Variations of multi-head attention.

Model	Global query		Self-attention
	Non-shared	Shared	
Key vector $\boldsymbol{k}_j$	$(W_r^h \boldsymbol{p}_j + \boldsymbol{b}^h)$		
Query vector	$\boldsymbol{q}^h$	$\boldsymbol{q}$	$\boldsymbol{q}_i^h = W_q^h \boldsymbol{p}_i + \boldsymbol{b}_q^h$
$\boldsymbol{q}$ shared between sequences	Yes		No
$\boldsymbol{q}$ shared between att. heads	No	Yes	No
Dot-product	$e_j^h = \boldsymbol{q}^{h\top} \cdot \boldsymbol{k}_j^h$	$e_j^h = \boldsymbol{q}^\top \cdot \boldsymbol{k}_j^h$	$e_{i,j}^h = \boldsymbol{q}_i^{h\top} \cdot \boldsymbol{k}_j^h$

on the unit sphere, so that the rankings by cosine distance and Euclidean distance are consistent.

We use a 2D dataset ($d = 2$) for visualization purposes and another high-dimensional dataset for quantitative evaluation. For the 2D dataset, we set a relatively small $G = 8$ and $k = 4$ in order to have a clear boundary between clusters. Since there are only 162 distinct subsets[5] with $G = 8, k = 4$, we use 100 of them for training, 31 for validation, and the remaining 31 for testing. For the high-dimensional dataset, we set $G = 1024$ and $k = 8$, and let $d = 16, 32, 64, 128$. We generate 10000 users for training, 1000 for validation, and another 1000 for testing.

Attention models. We focus on comparing our attention model (i.e. *self-attention*), the attention model utilized in ComiRec (i.e. *Non-shared query*), and the one used in PinText2 (i.e. *Shared query*). The comparison of the attentions is in Table 4.10.

For simplicity, we remove the temporal and positional encoding from the computation of attentions, skip the Ward clustering step from MIP, and directly represent user as Equation 4.7. Also, the dropout layer is removed in order to eliminate randomness in visualization.

Metrics. We visualize the intermediate results and user representations learned from the 2D dataset for qualitative evaluation. For high-dimensional data, we evaluate the performance by AUC and normalized discounted cumulative gain (nDCG).

Qualitative results and interpretations. Figure 4.4 shows the learned user representations given the engagement history. There are three observations. 1) When $H = 1$, global query attention fails to capture all the user interests, while the self-attention model is free from

[5]Number of ways to select no more than 4 clusters from a pool of 8 clusters: $162 = \binom{8}{1} + \binom{8}{2} + \binom{8}{3} + \binom{8}{4}$

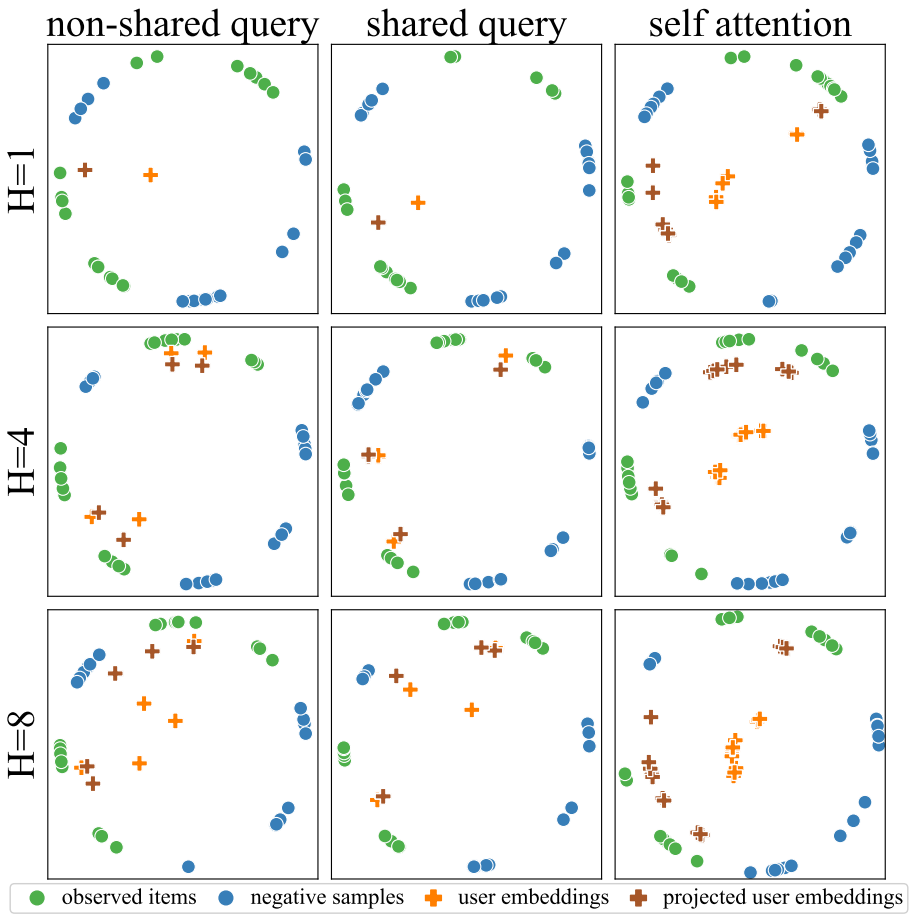


Figure 4.4. Learned user representations with different attention mechanisms. Non-shared and shared global query results might miss some user interest or are close to the negative categories, while self-attention results are comprehensive and accurate.

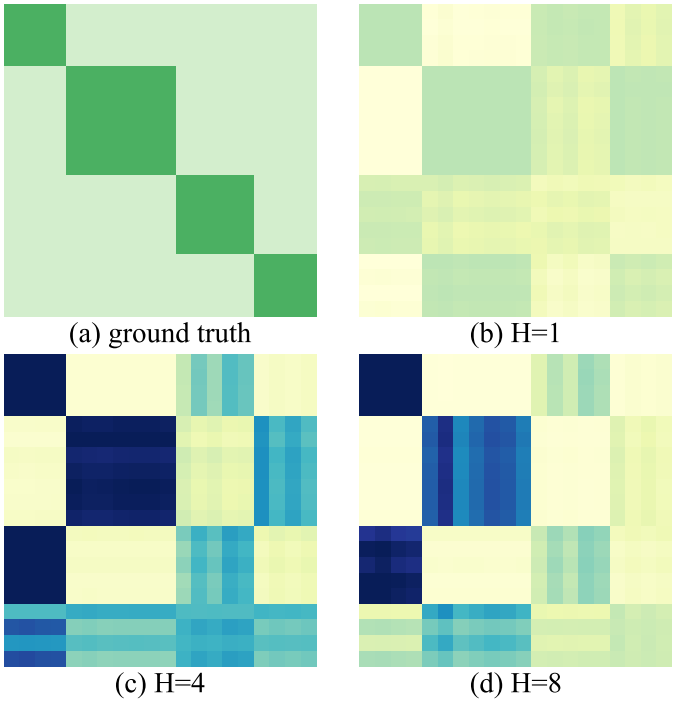


Figure 4.5. Learned attention scores in self-attention model. Darker color represents higher values. (a) the indicator function $\mathbb{1}_{[\mathscr{C}_i=\mathscr{C}_j]}$, (b-d) $a_{i,j}$. The input sequence is re-ordered for better visualization.

the limitation. 2) Viewing from the third row, the self-attention model is more accurate in learning cluster representations than global query models. The latter is systematically biased due to the global query as shown in global query models in Figure 4.6. 3) All the models learn super-clusters, depending on the bias in the dataset. For the example shown in Figure 4.4, the two adjacent clusters on the top side of the unit circle are often represented to be a super-cluster.

We also visualize the internal attention scores and self-attention models (Figure 4.5). Some attention heads show highly similar attention patterns because their queries are close to each other, which can be verified from Figure 4.6. Figure 4.5 compares the ground truth attention model with the learned attention. The learned attention shows clear boundaries between clusters in the heatmap. Note that the ground truth ignores the adjacency of clusters but the self-attention model considers the similarity between clusters, so Figure 4.5(a) is block diagonal

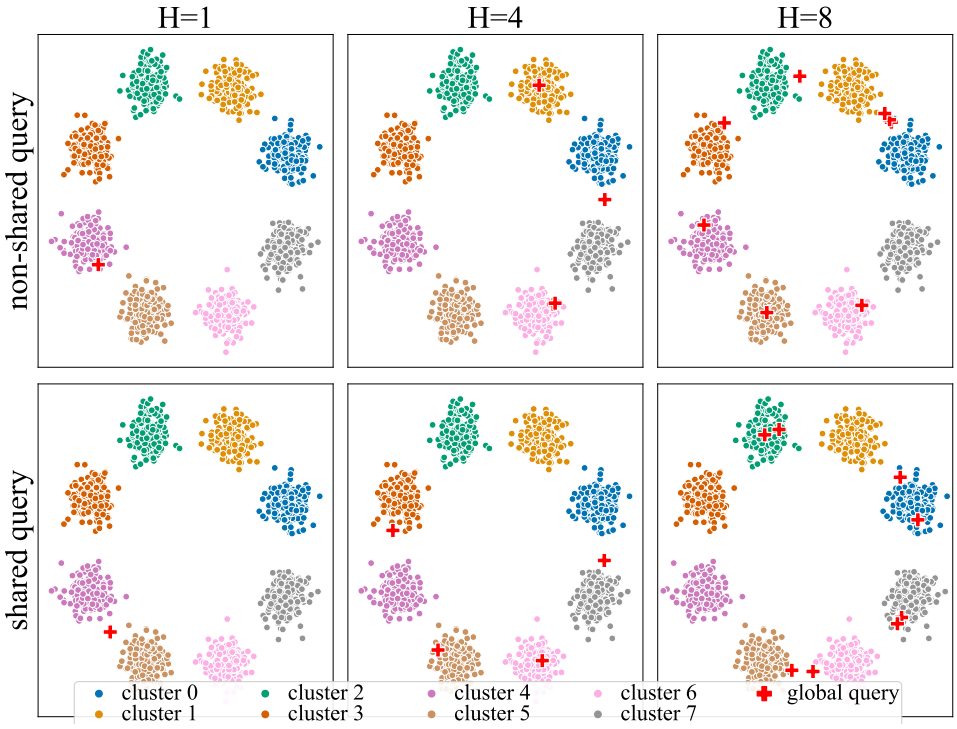


Figure 4.6. Learned global interests in global query models. The query vector is reversely projected and normalized.

while Figure 4.5(b-d) has dark blocks off the diagonal.

Quantitative results. Previous results show the intuitive comparison between global query models and the self-attention model, and the quantitative results further confirm the consistency of performance gain of self-attention. Experiments are repeated on the dataset for feature dimension $d = 16, 32, 64, 128$ and number of attention heads $H = 4, 8$. Figure 4.7 shows that the MIP model constantly and significantly outperforms global query models. As illustrated in the 2D dataset, the performance gain benefits from the personalized user representation, rather than matching to the globally popular clusters. Another observation from the result is that for global query models, $H = 4$ under-performs $H = 8$ models, as the number of attention heads decides the number of global clusters the model can learn; however, for self-attention model, $H = 4$ performs even better than $H = 8$. The explanation is that the self-attention model does not require a growing number of attention heads with respect to the number of global clusters, and $H = 4$ could be already enough for capturing user interest but easier than $H = 8$ to train.

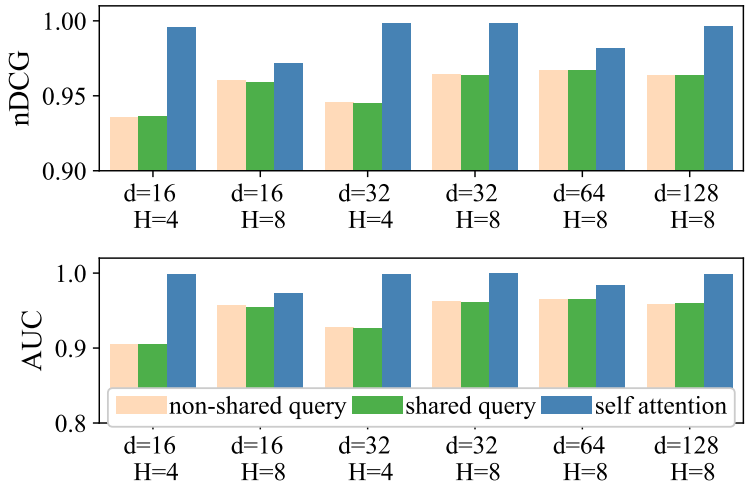


Figure 4.7. Performance comparison on high dimensional synthetic dataset. d denotes the feature dimension and H is the number of attention heads.

4.4.7 Ablation Study on Clustering Options

The Ward's algorithm is applied to MIP considering its success in PinnerSage[111], it's beneficial to explore the selection of the clustering algorithm and the number of clusters on the collaborative filtering dataset. To illustrate the impact, we evaluate MIP with a wide range of clustering algorithms.

Model Configuration and training: MIP models are configured with an attention module that takes both positional and temporal encoding. For unweighted MIP, no clustering method is applied to the encoded user engagement history $\{z_*\}$ (computed from Eq. 4.6) in the training stage. For weighted MIP, Ward's algorithm is applied to $\{z_*\}$ and the number of clusters is set to 5. To keep the MIP fully differentiable, the cluster embedding is the encoding of the last item in each cluster, instead of the medoid.

Inference: The choice of clustering in the inference phase is independent of its configuration during the training. We explore the inference options on the pre-trained models. Different types of clustering methods are compared:

- Ward: hierarchical clustering method that minimizes the sum of squared distances within

Table 4.11. Comparison of clustering options in AUC (in 10^{-2}). Note that the number of inference clusters is independent of training, i.e. changing the number of inference clusters does not require the re-training of the model. With the same number of clusters, the best performances are bold.

Clustering Method	Inference Clusters	Unweighted MIP			Weighted MIP		
		Amazon	Taobao	MovieLens	Amazon	Taobao	MovieLens
None	-	73.11	82.09	95.97	-	-	-
Ward	5	71.56	80.58	95.53	79.31	8**6.49**	94.61
	8	71.99	80.99	95.72	80.47	87.85	**95.25**
	10	72.16	81.20	**95.78**	80.84	88.42	**95.25**
K-Means	5	71.58	80.62	95.53	79.26	86.18	**94.86**
	8	71.95	81.03	95.71	80.66	88.02	95.17
	10	72.14	81.22	95.77	80.62	**88.61**	95.10
Spectral	5	**72.28**	**80.72**	**95.54**	78.99	85.84	94.46
	8	**72.37**	**81.08**	**95.73**	**80.79**	87.61	94.81
	10	**72.64**	**81.26**	95.78	**81.19**	88.40	95.07
BIRCH	5	71.98	80.63	95.52	**79.39**	86.29	94.61
	8	72.03	81.02	95.71	80.65	**88.03**	95.25
	10	72.44	81.21	95.78	80.91	88.53	**95.25**
DBSCAN	-	71.98	80.63	95.52	70.05	75.58	89.63

all clusters.

- K-Means: an iterative method also minimizes the sum of in-cluster summed squared distances.
- Spectral [130]: performs clustering on the projection of the normalized Laplacian computed from the affinity matrix.
- BIRCH [173]: another hierarchical method that clusters the points by building the Clustering Feature Tree.
- DBSCAN [48]: a density-based clustering method that does not require specifying the number of clusters.

The number of clusters is set to 5, 8, and 10 when required. Note that during training, the number of clusters is fixed to 5, however, after training, MIP can produce other numbers of embeddings

per user, which gives the system huge flexibility to trade-off between storage/computation cost and recommendation performance.

Result and analysis: There are two observations from Table 4.11. 1) the choice of clustering algorithm has a marginal impact on the performance. While PinnerSage reported that Ward's algorithm outperforms the K-Means, their result does not conflict with our observation here. Recall that for PinnerSage and our experiment on the Pinterest dataset, the clustering method is applied to the exogenous item embeddings, thus the clustering methods can be influenced by the non-flat geometry and outliers. However, with the collaborative filtering dataset, the clustering method is applied to the encodings produced by multi-head self-attention layers which average the embedding of the items and all other items (Eq. 4.3). The encodings after the multi-head self-attention should be smoothly distributed, and as a result, any clustering methods work almost equally well on that. 2) Selecting the number of clusters is a non-trivial trade-off. The motivation to decrease the number of clusters is the storage and computation cost which grow linearly as the number of clusters increases. For unweighted MIP, though the non-clustering (each item is a cluster) settings have the best AUC, decreasing the number of user embedding from 50 (non-clustering) to 10 is still acceptable. For weighted MIP, since it's impossible to learn the clustering weights without applying a clustering method, the trade-off can be more complicated: besides the storage concern, when the number of clusters increases the average information to learn the weights of each cluster decreases, and consequently may hurt the overall performance; on the other hand, 10-cluster settings are better than the 5-cluster settings for all the dataset.

4.4.8 Model Latency Comparison

Seeing the performance gain, another prominent question will be what is the time cost of the performance increase. In this section, we profiled the model latency on a desktop computer with a 12-core Intel i7-8700k CPU, and a single Nvidia GeForce RTX 2080 Ti GPU. The neural network training and inference are on the GPU with vanilla PyTorch framework (version

Table 4.12. Latency and performance comparison of the models. Training and inference latencies are measured in *ms*, and brackets show the standard deviations.

Latency/ Recall	GRU4Rec	BERT4Rec ($L = 1$)	BERT4Rec ($L = 2$)	PinText2	TiSASRec	ComiRec	MIP (total)	MIP (clustering)
Train	**218.57**	925.36	1058.34	555.79	452.94	479.59	998.62	5.86
(std)	(0.81)	(108.96)	(733.33)	(76.96)	(67.07)	(67.50)	(56.31)	(0.76)
Inference	**1.15**	38.34	57.53	14.46	14.54	14.61	40.05	5.93
(std)	(0.20)	(56.60)	(56.23)	(54.99)	(56.34)	(55.70)	(27.08)	(0.72)
R@50	63.50	63.15	66.52	54.13	66.67	67.36	**78.85**	-

1.12) without any further optimization on the computation. The clustering algorithm in MIP is performed by CPU with Python's scikit-learn package. We set the batch size to 1 and the dataset to Amazon, then measure and summarize the training and inference latency in Table 4.12.

There are a few observations from Table 4.12. First, compared to the neural network inference latency, the clustering step time cost is trivial. PinText2, ComiRec, and TiSASRec has similar training and inference latency, while the performance of them is worse than MIP. BERT4Rec has similar latency as MIP since our sequential model architecture is similar, while the BERT4Rec has worse performance. GRU4Rec has the least inference time. Notice that the standard deviations of the inference latency of PinText2, TiSASRec, and ComiRec are large. It indicates, though on average the three models are faster in inference, MIP inference latency is less possible to be very large while the other latency might be several times longer than average.

Conclusively, MIP, as well as other baselines compared, can all satisfy the latency requirement when applying online, even without further optimization on the computation and serving. MIP has a higher time cost compared to some of the baselines, but the performance increase is also appealing.

4.5 Conclusions

In this paper, we study the problem of multi-interest user embedding for recommendation systems. We follow the recent findings on representing users with multiple embeddings, which has been proven helpful over the single user representation. In addition, we illustrate that in

industrial recommendation systems, it is important to have a set of weights for these multiple embeddings for a more efficient candidate generation process due to its budget on the number of items returned. More specifically, we define the likelihood of an engagement based on the *closest* user embedding to the item embedding and update the weight for the corresponding cluster. Moreover, the case studies on multiple real-world datasets have demonstrated our advantage over state-of-the-art approaches. Finally, the ablation study on the cluster weight module demonstrated our intuition that simple heuristic does not work as well as personalized model-learned interest weights.

4.6 Acknowledgement

This chapter, in full, and Chapter 7, in part, are a reprint of the material as it appears in "Continuous cnn for nonuniform time series." Shi, Hui, Yang Zhang, Hao Wu, Shiyu Chang, Kaizhi Qian, Mark Hasegawa-Johnson, and Jishen Zhao. , In ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 3550-3554. IEEE, 2021. The dissertation author was the primary investigator and author of this paper.

Chapter 5

Neural Architectures II: Continuous CNN for Non-Uniform Time Series

Many real-world applications involve non-uniform time series, where the timestamps of the samples are non-uniform. In the traditional Bayesian framework, a classical model for nonuniform time series is the temporal point process (TPP). Recently, there have been increasing research attempts to infer and model the hyperparameters of TPP with deep neural networks, such as CNNs and RNNs. Despite its success, such a paradigm suffers from one disadvantage. Although the TPP frontend has good modeling of nonuniform time series, the deep learning backend does not, because both CNN and RNN implicitly assume that the input time series is uniform. In this paper, we propose the Continuous CNN (CCNN), which is specifically designed for nonuniform time series, and thus can serve as an improved deep learning backend for TPP. CCNN estimates the inherent continuous inputs by interpolation and performs continuous convolution on the continuous input. The interpolation and convolution kernels are learned in an end-to-end manner, and are able to learn useful patterns despite the nonuniform sampling rate. Results of several experiments verify that CCNN achieves a better performance on nonuniform data, and learns meaningful continuous kernels.

5.1 Introduction

In canonical time series prediction or processing problems, the time series data are usually assumed to be uniformly sampled, *i.e.* the time stamps are uniformly spaced. However, there are a significant number of applications that need to process event-based data, where the time stamps are randomly nonuniformly spaced, such as stock price [52], social media data [21] and health care data [73].

In the conventional Bayesian framework, the nonuniform time intervals can be elegantly modeled by temporal point processes (TPPs). A TPP is characterized by a conditional intensity function, $\lambda(t)$, parameterized based on some prior knowledge of the process. Therefore, learning of TPPs boils down to inferring the $\lambda(t)$.

Inspired by the strong representation power of deep learning models, there have been many recent research attempts that incorporate deep learning architectures to modeling or inferring $\lambda(t)$ for predicting the time of the upcoming event [46, 95, 106, 126, 143, 157]. A common paradigm is to introduce a deep neural network (DNN) as a backend, which produces a history embedding, given the time series, to predict the $\lambda(t)$ together with the TPP frontend. $\lambda(t)$ further determines the probability density function of the arrival time of the next event, $f^*(t)$, shown in Figure 5.1. The model parameters are trained to maximize the likelihood of the training event occurrence times under the TPP process.

Although the introduction of the DNN backend has been successful, and the TPP frontend has been continuously improving, there is one pitfall in this paradigm that remains unresolved. While the TPP frontend has good handling of the nonuniform time intervals of the output prediction, the neural network backend is not well-tailored for the nonuniform time series as the input, because both CNN and RNN rest on the assumption that both the input and output data are sampled uniformly.

Though RNN variations have been proposed to handle the time-sensitive series [106, 109, 177], the CNN structure for nonuniform time series has not yet been proposed. Two

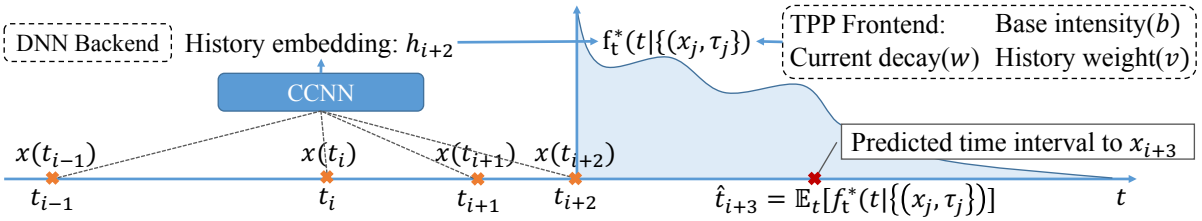


Figure 5.1. Predicting the time interval to the future event with CCNN backend and TPP frontend.

conventional CNN solutions to non-uniform time series are (1) directly concatenating the time stamps or time intervals to the input feature [171]; (2) interpolating the nonuniform time series to uniform by a pre-set kernel; then feeding the concatenated or transformed signal to a regular CNN. Notwithstanding, the former is ungrounded due to the violation of the uniform sampling assumption of CNN, while the latter can suffer from misspecification of interpolation kernel, and interpolation is not reasonable when the time series is categorical, thus the performance of both accommodations are compromised.

Motivated by these, we propose Continuous CNN (CCNN), a generalization to CNN for nonuniform time series, to serve as the backend of the neural-based TPP model. CCNN estimates the implicit continuous signal on the fly performing continuous convolution on the continuous signal. As a result, CCNN is capable of capturing useful patterns in the implicit input signal under the nonuniform sampling rate. Unlike existing interpolation method, the neural estimation of the implicit continuous signal does not rely on a pre-set interpolation kernel, but rather learned in an end-to-end manner, so that the misspecification error can be avoided. As shown in section 5.6, CCNN can achieve much better performance than the state-of-the-art systems on non-uniform time series data.

5.2 Related Works

The related research efforts mainly focus on two aspects: improving the efficiency and effectiveness of the TPP frontend, and adapting RNN backend for nonuniform event history.

Neural TPP frontend. Traditional TPP models make assumptions on the point generation process and the form of the intensity function $\lambda(t)$, *e.g.*, Poisson process, Hawkes process [58], and self-correcting process [69]. To avoid the modeling bias introduced by the pre-defined intensity function in the convention approaches, neural-based TPP methods define the intensity function in the most general form, and include as the conditional term an embedded sequence history, which is produced by the DNN backend [46, 106, 157]. Other efforts to improve this paradigm include training the model in a reinforcement learning framework [95], reducing the computation complexity [143], and directly estimating the conditional distribution of time intervals [126]. As mentioned in the introduction, they all rely on some DNN backends that are not suitable for processing nonuniform time series.

Time-sensitive RNN backend. There are some research efforts of adapting RNN for nonuniform series. In [19, 50, 114], continuous-time dynamical system methods are used to design RNN structures. Phased-LSTM [109] and Time-LSTM [177] introduce a time gate that passes the data at a certain frequency. Similar multi-resolution ideas can be found in Clockwork RNN [86] and DilatedRNN [22]. In neural-based TPP works, [106] utilizes a continuous-time LSTM, and [46] leverages a standard LSTM for embedding temporal sequence.

5.3 The CCNN Algorithm

The nonuniform time series prediction problem is formulated as follows. Given a nonuniform input sequence $x(t_1), x(t_2), \cdots, x(t_N) \in \mathscr{X}_{\text{in}}$, where the input time stamps $t_n \in \mathscr{T}_{\text{in}}$ can be distributed nonuniformly, our goal is to design a continuous convolutional layer that can produce output, $y(t_{out})$, for any *arbitrary* output time t_{out}.

The proposed CCNN solves the problem via two steps:

(1) interpolation to recover the continuous signal $\hat{x}(t)$;

(2) *continuous* convolution on $\hat{x}(t)$, producing the final output $y(t_{out})$.

Furthermore, rather than applying a preset interpolation, CCNN learns the interpolation

kernel and the convolution kernel in an end-to-end manner. The following two subsections elaborate on the two steps respectively. The channel dimension and input dimension are set to one for simplification.

5.3.1 Reconstruct Continuous Signal

CCNN reconstructs the underlying continuous input signal, $\hat{x}(t)$, by interpolating among nonuniform input samples.

$$\hat{x}(t) = \sum_{i=1}^{N} x(t_i) I(t - t_i; \mathscr{T}_{\text{in}}, \mathscr{X}_{\text{in}}) + \varepsilon(t; \mathscr{T}_{\text{in}}, \mathscr{X}_{\text{in}}) \tag{5.1}$$

where the first term is the interpolation term, and $I(\cdot)$ is the *interpolation kernel*; the second term is the *error correction* term. For the first term, a form analogous to the Parzen window approach [113] is used.

Considering the versatility of $I(\cdot)$, the interpolation algorithms representable by Equation (5.1) are vast. The error correction term, $\varepsilon(\cdot)$, are assumed to be determined by the input output time stamps and input values, hence its arguments include t, $\mathscr{T}_{\text{in}}$ and $\mathscr{X}_{\text{in}}$.

5.3.2 Continuous Convolution

Analogous to a standard CNN layer, after the continuous input is estimated by interpolation, the CCNN layer performs a continuous convolution to produce the final output.

$$y(t_{out}) = [\hat{x}(t) * C(t)]|_{t=t_{out}} + b \tag{5.2}$$

where $*$ denotes continuous convolution, $C(t)$ denotes the *convolution kernel*, and b denotes bias.

Unfortunately, without observing $\hat{x}(t)$, $I(\cdot)$, $\varepsilon(\cdot)$ and $C(\cdot)$ are not individually identifiable. To see this, we combine Equations. (5.1) and (5.2).

$$
\begin{aligned}
y(t_{out}) &= \sum_{i=1}^{N} x(t_i) \underbrace{[I(t-t_i;\mathscr{T}_{\text{in}},\mathscr{X}_{\text{in}}) * C(t)]}_{\text{collapsed kernel function}} + \underbrace{[\varepsilon(t;\mathscr{T}_{\text{in}},\mathscr{X}_{\text{in}}) * C(t) + b]}_{\text{collapsed bias function}} \big|_{t=t_{out}} \\
&= \sum_{i=1}^{N} x(t_i) K(t_{out}-t_i;\mathscr{T}_{\text{in}},\mathscr{X}_{\text{in}}) + \beta(t_{out};\mathscr{T}_{\text{in}},\mathscr{X}_{\text{in}})
\end{aligned}
\tag{5.3}
$$

where $K(t_{out};\mathscr{T}_{\text{in}},\mathscr{X}_{\text{in}})$ is the collapsed kernel function, representing the combined effect of interpolation and convolution; $\beta(t_{out};\mathscr{T}_{\text{in}},\mathscr{X}_{\text{in}})$ is the collapsed bias function, representing the combined effect of error correction and convolution. Equation (5.3) shows that learning the interpolation and convolution kernels and errors is now simplified into learning the collapsed kernel and bias functions. Once these two functions are learned, the final output can be readily computed using Equation (5.3). The next section will explain how CCNN is structured to learn these functions in an end-to-end manner.

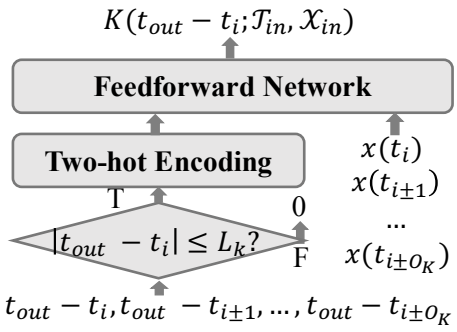
5.4 The CCNN Model

Following the discussion in Section 5.3, a CCNN layer is divided into three parts: the kernel network learning the collapsed kernel function, the bias network learning the collapsed bias function, and the main network producing the final output using Equation (5.3). The CCNN backend will be detailed in Sections 5.4.1-5.4.4, and the TPP frontend proposed by [46, 106] is introduced in Section 5.4.5.

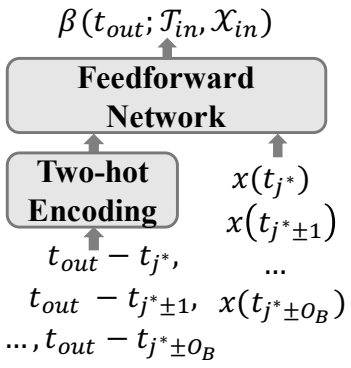
5.4.1 The Kernel Network

The basic idea of the kernel network is to represent the kernel function using a neural network, based on the fact that a neural network can represent any function given enough layers and nodes [66]. In order to regularize the complexity, a few assumptions on $K(\cdot)$ are introduced:

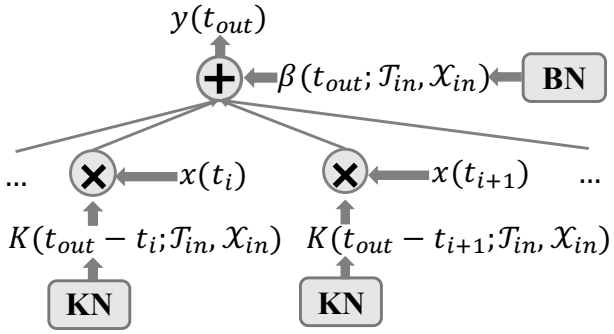
Stationarity and Finite Dependency: The dependency of $K(\cdot)$ on $\mathscr{T}_{\text{in}}$ is relative to the output time t_{out}, and is constrained to among the adjacent time stamps, i.e.



(a) Kernel network.



(b) Bias network.



(c) Compute $y(t)$ by collapsed interpolation and convolution.

Figure 5.2. CCNN structure.

$$K(t_{out} - t_i; \mathscr{T}_{\text{in}}, \mathscr{X}_{\text{in}}) = K(\{t_{out} - t_{i\pm k}, x(t_{i\pm k})\}_{k=0:O_K}) \tag{5.4}$$

where $\{t_{out} - t_{i\pm k}, x(t_{i\pm k})\}_{k=0:O_K}$ denotes the set of $t_{out} - t_{i\pm k}$ and $x(t_{i\pm k})$ where k runs from 0 to O_K, and O_K is the order of the kernel network.

Finite Kernel Length: The collapsed kernel function has finite length.

$$K(t_{out} - t_i; \mathscr{T}_{\text{in}}) = 0, \forall |t_{out} - t_i| > L_K \tag{5.5}$$

where L_K is the kernel length. This assumption implies the interpolation and the convolutional kernels both have finite length. While many interpolation kernels do have finite length (*e.g.* linear interpolation), others do not (*e.g.* sinc interpolation). Nevertheless, most infinite-length

interpolation kernels, including sinc interpolation, have tapering tails, and thus truncation on both sides still provides good approximations. Regarding the convolutional kernel, the finite length assumption naturally extends from the standard CNN.

Fig. 5.2(a) shows the kernel network structure, which is a feedforward network. According to Eq. (5.4), the inputs are $(\{t_{out} - t_{i\pm k}, x(t_{i\pm k})\}_{k=0:O_K})$. The output represents the kernel function, which is forced to be zero when $|t_{out} - t_i| > L_K$. To reduce learning difficulties, the time differences are fed into an optional two-hot encoding layer, which will be discussed later in details.

5.4.2 The Bias Network

For the bias network, a similar stationarity and finite dependency assumption is applied as follows.

$$\beta(t_{out}; \mathscr{T}_{\text{in}}, \mathscr{X}_{\text{in}}) = \beta(\{t_{out} - t_{j^*\pm k}, x(t_{j^*\pm k})\}_{k=0:O_B}), \text{ where, } t_{j^*} = \underset{t_j \in \mathscr{T}_{\text{in}}}{\operatorname{argmin}} |t_j - t_{out}| \tag{5.6}$$

t_{j^*} is the closest input time stamp to output time t_{out}, and O_B denotes the order of the bias network. The only difference from Eq. (5.4) is that the closest input time stamp, t_{j^*}, is chosen as a reference on which the time difference and the adjacent input time stamps are defined, because the major argument of the bias function is the output time itself, t_{out}, instead of the input-output time difference $t_{out} - t_i$. Fig. 5.2(b) shows the bias network, which is also a feedforward network.

5.4.3 Causal Setting

For causal tasks, current output should not depend on future input, and therefore the $t_{out} - t_{i\pm k}$ terms that are greater than 0, as well as the corresponding $x(t_{i\pm k})$, are removed from Eq. (5.4). Similarly, $t_{out} - t_{j^*\pm k}$ that are greater than 0, as well as the corresponding $x(t_{j^*\pm k})$, are removed from Eq. (5.6). Also, the condition bound in Eq. (5.5), *i.e.* $|t_{out} - t_i| > L_K$, is replaced

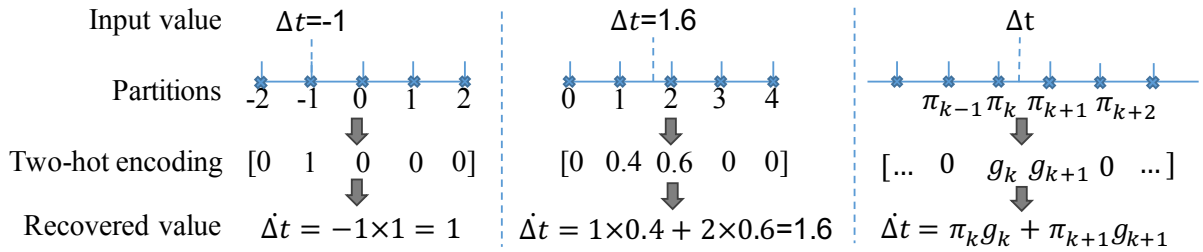


Figure 5.3. Two-hot encoding illustration. Two hot encoding transforms numeric value Δt to a vector in which at most two elements are non-zero. The two hot encoding does not lose information.

with $t_{out} - t_i > L_K$ or $t_{out} - t_i < 0$.

5.4.4 Two-Hot Encoding

The kernel and bias functions can be complicated functions of the input times, so model complexity and convergence can be serious concerns. Therefore, we introduce a two-hot encoding scheme based on [3] for transforming the input time intervals into vectors without losing information.

Denote the time interval value to be encoded as Δt. The two-hot encoding scheme partitions a range into $d-1$ intervals, whose edges are denoted as $\pi_1, \pi_2, \cdots, \pi_d$, and the interval width $\delta = \pi_k - \pi_{k-1}$ is a constant value. The two-hot encoding of Δt is a vector of length $d-1$, the elements of which are decided as: when Δt falls in an interval, the two elements corresponding to its two edges are lit. Formally, denote the encoded vector as $\boldsymbol{g}$, and suppose Δt falls in interval $[\pi_k, \pi_{k+1})$. Then

$$\boldsymbol{g}_k = \frac{\pi_{k+1} - \Delta t}{\delta}, \boldsymbol{g}_{k+1} = \frac{\Delta t - \pi_k}{\delta}; \boldsymbol{g}_l = 0, \forall l \notin \{k, k+1\} \tag{5.7}$$

where $\boldsymbol{g}_k$ denotes the k-th element of $\boldsymbol{g}$. Fig. 5.3 gives an intuitive visualization of the encoding process.

5.4.5 Temporal Point Processes Frontend

In a similar way to [46, 106], a TPP frontend is parameterized by $\lambda^*(t)$, which depicts the rate of the probability of the event occurrence. Formally

$$\lambda^*(t)dt = Pr\left(\text{Event } i \text{ happens in } [t, t+dt) | \bigcup_{j<i} \{x(t_j), t_j\}\right) \tag{5.8}$$

It can be shown that the probability density function (PDF) of an event happening at time t conditional on the history of the events $\bigcup_{j\leq i-1}\{x(t_j), t_j\}$ can be expressed as

$$f^*(t) = \lambda^*(t) \exp\left(-\int_{t_{i-1}}^{t} \lambda^*(\tau)d\tau\right). \tag{5.9}$$

Rather than applying some preset functional form for $\lambda^*(t)$ as in conventional TPPs, we propose to use a CCNN to model $\lambda^*(t)$ as follows. First, we pass the historical time series to a CCNN to learn a history embedding

$$\boldsymbol{h}_{i-1} = \text{CCNN}(\bigcup_{j\leq i-1} \{x(t_j), t_j\}, t_{i-1}), \tag{5.10}$$

where CCNN $(\mathscr{D}_{in}, t_{out})$ is just a functional abstraction of CCNN, where $\mathscr{D}_{in}$ denotes the input sequence, and t_{out} denotes the output time stamp. Then $\lambda^*(t)$ is obtained by combining the history information and the current time information as follows

$$\lambda^*(t) = \exp(\boldsymbol{v}\boldsymbol{h}_{i-1} + w(t - t_{i-1}) + b), \tag{5.11}$$

where $\boldsymbol{v}$, w and b are trainable parameters. Combining Eqs. (5.9) and (5.11), we can obtain a closed-form expression for $f^*(t)$

$$f^*(t) = \exp\Bigg(\boldsymbol{v}\boldsymbol{h}_{i-1} + w(t - t_{i-1}) + b + \frac{1}{w}\Big(\exp(\boldsymbol{v}\boldsymbol{h}_{j-1} + b) - \exp(\boldsymbol{v}\boldsymbol{h}_{j-1} + w(t - t_{j-1}) + b)\Big)\Bigg). \tag{5.12}$$

By maximizing this likelihood on the training data, we can estimate the conditional distribution of the time intervals. To obtain a point estimate of the time interval till the next event, we compute the expectation under Eq. (5.12) numerically.

5.4.6 Summary and Generalization

Fig. 5.2 illustrates the structure of a CCNN layer. To sum up, the kernel network and bias network learn the continuous kernel and bias as functions of t, $\mathscr{T}_{\text{in}}$ and $\mathscr{X}_{\text{in}}$. The main network applies these functions to produce the output according to Eq. (5.3). The hyperparameters include O_K (Eq. (5.4)), L_K (Eq. (5.5)), O_B (Eq.(5.6)) and δ (Eq. (5.7)).

It is worth highlighting that CCNN not only accommodates arbitrary input timestamps, it can also produce output at any output timestamps, by simply adjusting the value of t in Eqs. (5.3), (5.4) and (5.6). So a CCNN layer can accept input at a set of timestamps, and produces output at a different set of timestamps, which is very useful for resampling, interpolation, and continuous sequence prediction.

When the inputs $x(t_i)$ and output $y(t)$ need to be multidimensional, according to Eq. (5.3), $K(\cdot)$ and $\beta(\cdot)$ become vectors or matrices with matching dimensions. Therefore, we simply need to adapt the output of the kernel and the bias networks from scalars to vectors or vectorized matrices. Also, a multi-layer CCNN can be constructed by stacking individual CCNN layers, with the input timestamps of a layer matching the output timestamps of its previous layer.

5.5 Representation Power Analysis

In this section, we study how the proposed CCNN layer relates to and compares against two CNN baselines in terms of representation power. The first baseline is simply a regular convolutional layer, and the second baseline is a convolutional layer with input time intervals, Δt_i, appended to the input features, which we will call CNNT throughout.

5.5.1 Case 1: Output Timestamps Same as Input

This subsection intuitively explains why CCNN has a superior representation power to CNNT. Suppose the output timestamps are the same as input timestamps, i.e. $t \in \mathscr{T}_{\text{in}}$. Then, combining Equation (5.3) with Eqs. (5.4)-(5.6), we have

$$\begin{aligned} y(t_j) = &\sum_i x(t_i) K(t_j - t_i; \mathscr{T}_{\text{in}}, \mathscr{X}_{\text{in}}) \\ &+ \beta(0; \{t_j - t_{j\pm k}, x(t_{j\pm k})\}_{k=0:O_B}). \end{aligned} \tag{5.13}$$

In contrast, for a CNNT layer, if we separate the convolution on time interval from the rest

$$y(t_j) = \sum_i x(t_i) K_{j-i} + \left[\sum_i (t_j - t_{j-1}) K'_{j-i} + b \right]. \tag{5.14}$$

The second term of Equation (5.13) represents a feed-forward network on the input of $\{t_j - t_{j\pm k}, x(t_{j\pm k})\}_{k=0:O_B}$, whereas the second term of Equation (5.14) can be regarded as a one-layer feed-forward network on $\{t_j - t_{j-1}\}$, which is equivalent to a one-layer feedforward network on $\{t_j - t_{j\pm k}\}_{k=1:O_B}$. In other words, appending the time interval feature to the convolution layer is only a weak version of the CCNN bias network.

Yet a more fundamental disadvantage of CNNT lies in the first term, where the convolution kernel of CNNT, K_{j-i}, does not depend on the actual time difference, but the order in which the sample arrives. This makes CNNT very vulnerable to the sampling variations. CCNN, on the

other hand, has a more robust kernel function for nonuniform data.

5.5.2 Case 2: Uniform Timestamps

Both CNNT and CCNN have the same representation power as CNN under uniform sampling rate, and thus both are strict generalizations to CNN. For CNNT this is trivial because the second term of Equation (5.14) reduces to a constant. For CCNN, we have the following theorem.

Theorem 1. Let $\mathscr{F}$ be the set of all functions that can be represented by a CCNN layer, and $\mathscr{G}$ be the set of all functions that can be represented by a $1 \times W$ convolutional layer. Then $\mathscr{F} = \mathscr{G}$, if the following conditions hold:

1. the input and output timestamps are uniform and the same, i.e. $\Delta t_i = \Delta t, \forall i$.

2. The two-hot encoding interval boundaries are at multiples of Δt, i.e. $\pi_k = k\Delta t$.

3. The dimension of the two-hot vector of CCNN is no smaller than the CNN kernel length, i.e. $D \geq W$.

4. If the kernel network has hidden layers, the hidden node dimension is no smaller than W.

5. CCNN and CNN have the same receptive field size, i.e. $2L_K = W\Delta t$.

6. The kernel and bias networks do not depend on $\mathscr{X}_{\text{in}}$.

Proof to Theorem 1. We will only consider the 1D case. The generalization to multi-dimensional cases is straightforward. The regular $1 \times W$ CNN performs the following operation to generate the output sequence

$$y(\tau_k) = \sum_{i=k-(W-1)/2}^{k+(W-1)/2} x(t_i) K_{k-i} + b \tag{5.15}$$

where $\left\{K_{-(W-1)/2}, \cdots, K_{(W-1)/2}\right\}$ and b are trainable parameters. Here we implicitly assume that W is odd. We will prove the theorem by primarily utilizing the correspondence between Equations (5.13) and (5.15).

• $\mathscr{F} \subset \mathscr{G}$:

$\forall K(\{t_j - t_{j+k}\}_{k=\pm 0:\pm O_K})$ represented by the kernel network, and $\forall \beta(\{t_j - t_{j+k}\}_{k=1:O_B})$ represented by the bias network, and thereby $\forall f \in \mathscr{F}$.

By Equation (5.4) and Cond. 1, the arguments of the kernel function $K(\cdot)$, namely the input to the kernel network, can only be a set of consecutive multiples of Δt, i.e.

$$K(\{(w+k)\Delta t\}_{k=\pm 0:\pm O_K}) \tag{5.16}$$

where w is an integer.

Moreover, from Equation (5.5) and Cond. 5, $K(\cdot)$ is non-zero iff w lies in the interval $[-(W-1)/2, (W-1)/2]$.

Similarly, from Equation (5.13) and Cond. 1, the arguments of the bias function $\beta(\cdot)$ can only take one set of values:

$$\beta(\{k\Delta t\}_{k=\pm 1:\pm O_B}) \tag{5.17}$$

Then, Equation (5.15) can be made equivalent to Equation (5.13) by setting

$$\begin{aligned} K_w &= K(\{(w+k)\Delta t\}_{k=\pm 0:\pm O_K}) \\ b &= \beta(\{k\Delta t\}_{k=\pm 1:\pm O_B}) \end{aligned} \tag{5.18}$$

which means $f \in \mathscr{G}$. Here concludes the proof that any CCNN layer can be matched by a CNN layer.

• $\mathscr{G} \subset \mathscr{F}$

Here we would only prove the case where both the weight network and bias network of CCNN has only one layer, which is the most difficult case. If either network has more than one layers, the proof is easily generalizable by setting the bottom layers to identity, which is feasible

because of Cond. 4. Also, we only consider the case where the kernel network order and the bias network order are both ones, i.e.

$$O_K = 1, O_B = 1 \tag{5.19}$$

The proof can be generalized to larger orders by setting the additional weights to zero. In the special case defined above, the kernel network in Equation (5.16) is further simplified to $K(w\Delta t)$. The bias network in Equation (5.17) is further simplified to $\beta(0)$.

Further, notice that the $w\Delta t$, as the input to the kernel network, has to go through two-hot encoding. By Cond. 2 and 3, the two-hot encoding of $w\Delta t$ is a one-hot vector, where only the w-th dimension is activated and the rest is zero. Since the kernel network is a one-layer feedforward network, denote the weight connected to the w-th dimension of the two-hot vector as P_w, then we have

$$K(w\Delta t) = P_w \tag{5.20}$$

$\forall\{C_w\}$ and b that define a CNN layer, and thereby $\forall g \in \mathscr{G}$, let

$$P_w = \begin{cases} K_w & \text{if} -\frac{W-1}{2} \leq w \leq \frac{W-1}{2} \\ 0 & \text{otherwise} \end{cases} \tag{5.21}$$

Cond. 3 ensures that there are enough number of nontrivial P_ws to cover all the w in the case specified in line 1 of the equation above; and let

$$\beta(0) = b \tag{5.22}$$

which means the bias network learns a constant. Then the CCNN layer can be made equivalent to the CNN layer, i.e. $g \in \mathscr{F}$. Here concludes the proof that any CNN layer can be matched by a CCNN layer. □

Thm. 1 implies that in the uniform case where the increased model complexity of CCNN

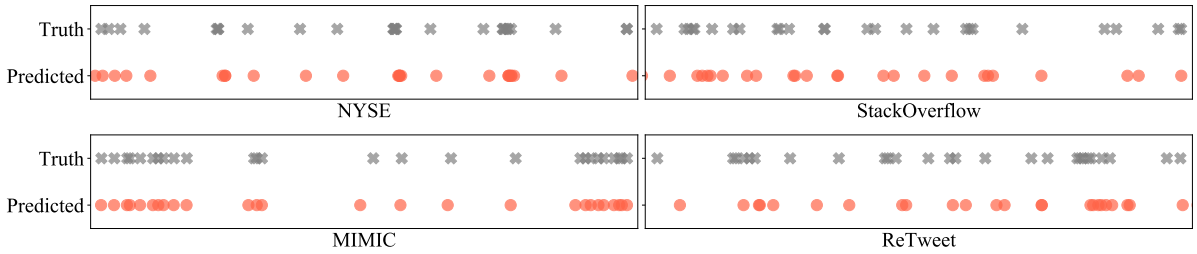


Figure 5.4. Predicted timeline compared to the ground truth timeline over consecutive events.

is not needed, it will not harm the performance either. Replacing CNN or CNNT with CCNN will not be worse off regardless of how the data are distributed in time.

As a final remark, readers may argue that the improved representation power of CCNN is merely a trivial result of increased model complexity, not because CCNN handles the time information more correctly. However, as will be shown in the next section, even with matching model complexity, CNN and CNNT are still unable to match the performance of CCNN. CCNN does not just increase the model complexity but increases the model complexity the right way.

5.6 Evaluation

In this section, CCNN is evaluated on a set of prediction tasks on both simulated and real-world data.

5.6.1 Predicting Time Intervals to Next Event with TPP

In this section, we evaluate CCNN on real-world data to to predict the time intervals to the next event.

Datasets and Configurations. Four time series datasets with nonuniform time intervals are introduced, *i.e.* NYSE, Stackoverflow [1], MIMIC [73] and Retweet [174].

- **Stackoverflow:** The dataset contains 26535 training samples, 3315 test samples, and 3315 validation samples. Each sequence represents the history awarding *badges* to a single user. The sampling timestamps are the time of awarding each badge, and the signal value is the

type of the badge (e.g., Guru, Nice Question, Great Answer, etc.). There are 22 types of badges in total.

- **Retweet:** The dataset contains 20000 training samples, 1000 test samples, and 1000 validation samples. The sequence is the history of a tweet being re-posted by other users. According to the number of followers of the users retweeted the post, the label of each retweet, in the whole retweet history, is one of the 3 the user groups.

- **MIMIC:** The dataset is the collection of clinical visit history. The training set contains 2925 sequences; the test and validation set contains 312 each. The diagnosis are filtered to preserve only top-10 common diseases as the label.

- **NYSE** The dataset is book order data collected from NYSE of high-frequency transaction in one day. The dataset contains 2 types of events (sell and buy) with 298710 training sequences, 33190 testing sequences, and 82900 validation sequences.

The input sequence is the time stamps and the one-hot encoded types of a series of events. The task is to predict the time interval until the next event of a specific type, given the previous events. As mentioned in Sec. 5.4.5, following the design in [46], the input sequence is assumed to be generated via an underlying marked TPP, where the time stamps follow a TPP, and the marker, *i.e.* the type of the event, is generated from a multinomial distribution conditioned on history. The output of the networks is a condition intensity function $\lambda^*(t)$ as in Eq. (5.11). As the model prediction, the expected duration is computed numerically from the estimated conditional distribution (Eq (5.12)). The evaluation metric is the MSE of the expected duration and the actual duration to the next event. All the models are trained with Adam optimizer [82] and negative log likelihood (NLL) loss.

Models. We benchmark with two baselines specialized for this type of tasks, Recurrent Marked Temporal Point Process (RMTPP) [46] and N-SM-MPP [106]. Among the approaches with the same TPP frontend modeling, N-SM-MPP is the current state-of-the-art deep learning

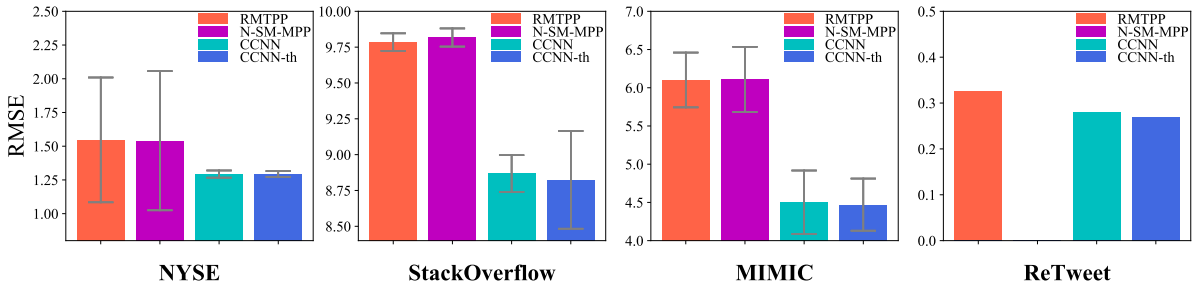


Figure 5.5. RMSE of the predicted time intervals to next event. Standard deviation is calculated among 5 train-test-validation splits (ReTweet dataset has only one split so no standard deviation is reported). N-SM-MPP did not report its RMSE on the ReTweet dataset.

method. For NYSE, StackOverflow and MIMIC, we directly compare to the results [106] reported, and re-implemented RMTPP to benchmark ReTweet.

The configurations for CCNN are as follows. The input sequence length to NYSE, Stackoverflow, and ReTweet dataset is 13, and the CCNN uses two 1×7 kernels with 16 filters for each. MIMIC contains only very short sequences, so CCNN uses two 1×2 kernels and predicts only based on past 3 events. The input one-hot encoded event types, $\{x(t_i)\}$, which are first passed through an embedding layer, which is a 1×1 convolutional layer with a channel dimension of 8, and the resulting embedded vectors are then fed into the networks. This task is a causal task, where current output should not depend on future input. Therefore, the CCNN configuration is adapted to the causal setting, as discussed in section 5.4.3.

Results and Analysis. Figs. 5.4 and Figure 5.5 show the estimated event interval (expectation of Eq. (5.9)) and the RMSEs. Figure 5.4 shows that the predicted timelines align well with the ground truth and result in smaller RMSE, though the ground truth shows extreme fluctuation on event intervals: events occur frequently in some period while merely anything occurs during other times. Fig .5.5 compares the RMSE with the baselines. CCNN algorithms outperformed two baselines in all datasets. There is a slight advantage of CCNN-th over CCNN, which verifies the effectiveness of two-hot encoding.

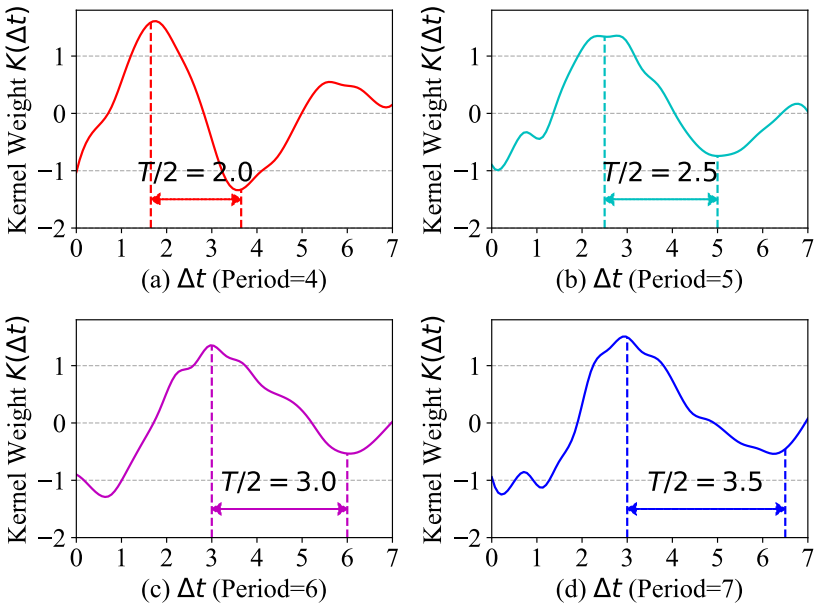


Figure 5.6. The learned continuous kernel function on the sine, as functions of $\Delta t = t - t_i$.

5.6.2 Easing the Interpolation Kernel Misspecification Error

To reveal the misspecification error introduced by interpolation methods with the pre-set kernel, we compare the performance of CCNN and other adaptions of CNN/RNN structures for nonuniform time series on the prediction task that predicts the next sample $x(t_{N+1})$, given the previous nonuniform samples $x(t_1) \cdots x(t_N)$.

Datasets The synthetic datasets are generated by unevenly sampling from three standard time series: Sine, Mackey-Glass(MG), and Lorenz, as listed in Table 5.1. The standard time series functions are introduced as $x(t)$. We set the parameters of the three sequences in the following way: (1) $T = 5$ in Sine; (2) $\beta = 0.2$, $\tau = 17$, $n = 10$, and $\gamma = 0.2$ in MG; (3) $\sigma = 10$,

Table 5.1. Synthetic continuous signals. $\dot{x}$ denotes dx/dt

Sine	Mackey-Glass (MG) [53]	Lorenz [100]
$x(t) = sin\left(\frac{2\pi t}{T}\right)$	$\dot{x}(t) = \beta \frac{x(t-\tau)}{1+x(t-\tau)^n} - \gamma x(t)$	$\dot{x}(t) = \sigma(y(t) - x(t))$ $\dot{y}(t) = -x(t)z(t) + rx(t) - y(t)$ $\dot{z}(t) = x(t)y(t) - bz(t)$

Table 5.2. Mean squared error of prediction on simulated data ($\times 10^{-2}$).

Alg.	Sine	MG	Lorenz
CNN	46.0 (8.22)	12.8 (3.92)	9.90 (3.33)
CNNT	20.2 (7.65)	3.50 (1.29)	5.97 (2.41)
CNNT-th	8.44 (4.58)	3.00 (1.21)	8.37 (3.24)
ICNN-L	1.13 (0.87)	0.97 (0.53)	5.81 (2.78)
ICNN-Q	0.75 (0.65)	0.83 (0.46)	5.08 (2.59)
ICNN-C	0.72 (0.83)	0.72 (0.42)	4.22 (2.27)
ICNN-P	20.5(6.43)	1.95(0.79)	8.50(3.32)
ICNN-S	17.2(5.57)	3.51(1.36)	8.20(3.31)
RNNT	36.1(12.9)	8.15(3.32)	13.4(3.95)
RNNT-th	19.5(6.48)	8.48(3.11)	13.9(4.36)
CCNN	0.88 (0.61)	2.46 (0.89)	3.93 (1.73)
CCNN-th	**0.42** (0.36)	**0.53** (0.97)	**3.25** (1.67)

$r = 28$, $b = 8/3$ in Lorenz. Sine signal is sampled by time intervals that follow the Poisson distribution with mean of 1. MG and Lorenz are two chaotic time series which do not have closed-form solutions to their delay differential equations. Therefore the Runge-Kutta method [13] is applied to obtain a discrete numerical solution, which is a uniform sequence with sampling interval Δt. A set of N-sample short uniform sequences are generated from the long sequence by a sliding window with window shift of one. The uniform sequences are subsampled into nonuniform sequences by choosing M ($M < N$) sample points uniformly at random. We set $M = 42$, $N = 14$; we set $\Delta t = 2$ in MG, and $\Delta t = 0.05$ in Lorenz.

Models. The following algorithms are compared:

• *CCNN:* The first layer is a CCNN layer takes both the time intervals and the signal sequence. Then the sequence is resampled onto a uniform time interval. CCNN is set to the causal setting as of Sec. 5.4.3. The time information is either two-hot encoded (CCNN-th) or not encoded (CCNN). CCNN has two layers. The first layer is a CCNN layer with output timestamps at $t_{n+1} - k$, $k = 1, \cdots, 13$. The bias network has two layers and 4 hidden channels. Its order O_B is 7. The kernel network has two layers and 4 hidden nodes. Its order O_K is 3. The kernel length $L_k = 3$. The output of CCNN nerwork has 72 channels. The second layer is a regular 1×7 convolutional layer. The total number of parameters is 1,273. CCNN-th has almost the

same structure as CCNN, except that the number of hidden channels is 36. The total number of parameters is 1,261.

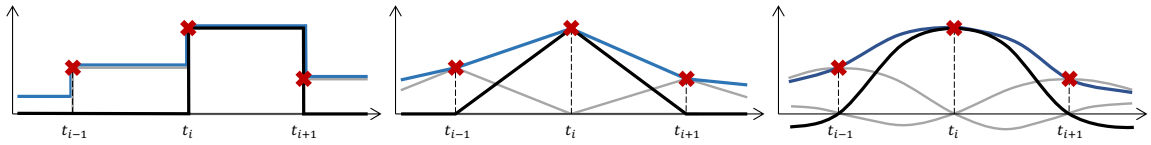
• *CNN:* data are directly fed into a regular CNN, with no special handling of nonuniform sampling. CNN has two 1×7 layers, and the number of hidden channels is 84. The total number of parameters is 1,261.

• *CNNT:* The time intervals are appended to the input data, which are fed to a regular CNN. The time information is either two-hot encoded (CNNT-th), or not encoded (CNNT). CNNT has two 1×7 layers, and the number of hidden channels is 60. The total number of parameters is 1,261. The sampling time intervals are appended as input features. The time interval appended to the last sample of the sequence is the difference between the prediction time and the time of the last sample. In CNNT-th, the two-hot encoding interval width $\delta = 0.5$, and the two-hot vector dimension is 14. The network has two 1×7 layers, and the number of hidden channels is 10. The total number of parameters is 1,261.

• *ICNN:* data are interpolated to be uniform before being fed to a regular CNN. Piecewise Constant (ICNN-P), linear (ICNN-L), quadratic (ICNN-Q), cubic spline (ICNN-C) and sinc (ICNN-S) interpolation algorithms are implemented. ICNN takes the interpolated signal at $t_{n+1} - k$, $k = 1, \cdots, 13$ as input. The network has two 1×7 layers, and the number of hidden channels is 84. The total number of parameters is 1,261.

• *RNNT:* the time intervals are appended to the input data, then are fed into a vanilla RNN. The time information is either two-hot encoded (RNNT-th), or not encoded (RNNT). RNNT and RNNT-th have two layers and the number of hidden channels is 32. The total number of parameters is 1153 for RNNT and 1633 for RNNT-th.

All the networks have two layers with ReLU activations in the hidden layers and no activations in the output layer. The hyperparameters are set such that all the architectures share the same number of layers, receptive field size, and number of parameters. For CNN, ICNN, and CNNT, the convolution kernel length of each layer is set to 7. For ICNN, the input signal is interpolated at timestamps $t_{N+1} - k$, $k = 1, \cdots, 13$ to form a uniform sequence before feeding



(a) Piecewise Constant **(b)** Linear **(c)** Sinc

Figure 5.7. Illustration of interpolation kernels. The red crosses denote the input data samples. The black line shows the interpolation kernel for $x(t_i)$; the gray lines show the kernels for the other two points. The blue line shows the interpolated result.

into two-layers regular CNN. For CCNN, the output time stamps of the first layer are $t_{N+1} - k$, $k = 1, \cdots, 13$. The kernel length $L_K = 3$. Since its input is uniform, the second layer of CCNN is a regular convolutional layer, with kernel length 7. These configurations ensure that all the neural networks have the same *expected* receptive field size of 13.

All the networks are trained with Adam optimizer [82] and mean squared error (MSE) loss. The training batch size is 20. The number of training steps is determined by validation. The validation set size is 10,000.

Interpolation Kernels

• *Piecewise Constant Interpolation:*

$$I(t - t_i; \mathscr{T}_{\text{in}}, \mathscr{X}_{\text{in}}) = \mathbb{1}[0 < t - t_i \leq t_{i+1} - t_i] \tag{5.23}$$

where $\mathbb{1}[\cdot]$ denotes the indicator function.

• *Linear Interpolation:*

$$I(t - t_i; \mathscr{T}_{\text{in}}, \mathscr{X}_{\text{in}}) = \begin{cases} \frac{t - t_{i-1}}{t_i - t_{i-1}} & \text{if } 0 < t - t_{i-1} < t_i - t_{i-1} \\ \frac{t_{i+1} - t}{t_{i+1} - t_i} & \text{if } 0 < t_{i+1} - t < t_{i+1} - t_i \\ 0 & \text{otherwise.} \end{cases} \tag{5.24}$$

• *Sinc Interpolation:*

$$I(t - t_i; \mathscr{T}_{\text{in}}, \mathscr{X}_{\text{in}}) = a \sin\left(\frac{\pi(t - t_i)}{a}\right) / (t - t_i). \tag{5.25}$$

These examples are illustrated in Fig 5.7. Notice that in Eqs. (5.23) and (5.24), the interpolation kernels depend not only on $t - t_i$, but also on adjacent input times, t_{i+1} and/or t_{i-1}, and hence all the input times $\mathscr{T}_{\text{in}}$ are put in the argument. In some nonlinear interpolations, the interpolation kernel is also affected by the input values $\mathscr{X}_{\text{in}}$.

Results and Analysis. Table 5.2 lists the MSEs. There are three observations. **First**, CCNN-th outperforms the other baselines in terms of prediction accuracy. Notice that the number of convolution channels of CCNN are significantly smaller than most of the other baselines, in order to match the number of parameters. Nevertheless, the advantage in properly dealing with the nonuniform sampling still offsets the reduction in channels in most tasks. **Second**, interpolation methods (ICNNs and CCNN) generally outperform the other baselines, particularly CNNT. This again shows that interpolation is more reasonable for dealing with nonuniform time series than simply appending the time intervals. Furthermore, preset interpolation algorithms (ICNNs) can rarely match CCNN that has the flexibility to learn its own interpolation kernel. **Third**, two-hot encoding usually improves performance. Again, there are fewer channels with two-hot encoding in order to match model complexity, but the advantage of two-hot encoding still stands out.

5.6.3 Kernel Analysis

To clearly interpret the representation power of CCNN on nonuniform time series, it's critical to visualize the learned collapsed continuous kernels $K(t_{out} - t_i; \mathscr{T}_{\text{in}}, \mathscr{X}_{\text{in}})$. We repeat the experiment that predicts the next sample $x(t_{N+1})$ as Sec. 5.6.2 with different Sine signals, $T = 4, 5, 6, 7$. The CCNN has the same configuration as Sec. 5.6.2 except that the number of filter is 1, so that the $K(t_{out} - t_i; \mathscr{T}_{in}, \mathscr{X}_{in})$ outputs a scalar that could be straightforward for visualization.

Figure 5.6 shows the learned continuous kernel functions on 4 input signals, which are all quite interpretable. Each kernel is a sine-like function with estimated period equaling the underlying signal period. The result also explains the advantage of CCNN in the previous

Table 5.3. Mean squared error of prediction on realworld data.

Alg.	DM1	DM2	DM3	DM4	DM5	DM6	DM7	DM8	DM9	DM10	DM11	DM12	DM13
CNNT-th	0.86	0.81	0.91	0.99	0.42	0.76	0.52	0.55	0.43	0.69	0.84	0.99	0.47
ICNN-L	0.52	0.27	0.70	1.03	0.06	0.30	0.02	0.24	**0.26**	0.44	0.51	0.84	0.13
ICNN-Q	0.61	0.28	0.69	**0.98**	0.06	0.30	0.06	0.23	0.29	0.45	0.57	0.84	0.13
ICNN-C	0.63	0.29	0.71	1.02	0.06	0.30	0.04	0.25	0.29	0.43	0.56	0.82	0.13
ICNN-P	0.71	0.40	0.74	0.99	0.09	0.37	**0.02**	0.28	0.33	0.45	0.60	0.94	0.17
ICNN-S	0.53	0.26	0.66	1.00	0.07	0.28	0.21	0.26	0.31	0.43	**0.49**	0.87	0.15
RNNT-th	0.79	0.27	0.70	1.06	0.05	0.38	0.42	**0.20**	0.33	0.48	0.57	1.01	0.11
CCNN-th	**0.49**	**0.23**	**0.65**	1.00	**0.05**	**0.27**	0.04	0.22	0.29	**0.41**	0.50	**0.80**	**0.12**

experiments.

5.6.4 Autoregression on Real-world Dataset with Missing Data

In order to test the advantage of CCNN in real-world scenarios with missing observations, 13 time-series datasets are randomly chosen from the Data Market[1], named DM1 through DM13. A brief description of these datasets is given below. Each dataset consists of a univariate uniform time series, which is split into training, test and validation sequences by a ratio of 6:2:2. Nonuniform subsequences are generated the same way as in MG with $N = 28$ and $M = 14$. All of the data are monthly data. The network configurations are the same as those in the simulated experiment. In particular, the receptive field size is set to seven, which means the prediction is based on an average of 14 months of historic data. This should lend adequate information for prediction.

DM1: Australia monthly production of cars and station wagons from July 1961 to August 1995.[2] The total length of the sequence is 414.

DM2: Monthly data on Clearwater River at Kamiah, Idaho from 1911 to 1965.[3] The total length of the sequence is 604.

DM3: Monthly data on James River at Buchanan, VA from 1911 to 1960.[4] The total

[1]https://datamarket.com

[2]https://datamarket.com/data/set/22lf/australia-monthly-production-of-cars-and-station-wagons-jul-1961-aug-1995#!ds=22lf&display=line

[3]https://datamarket.com/data/set/22zg/clearwater-river-at-kamiah-idaho-1911-1965#!ds=22zg&display=line

[4]https://datamarket.com/data/set/22y3/james-river-at-buchanan-va-1911-1960

length of the sequence is 604.

DM4: Average monthly precipitation from 1907 to 1972.[5] The total length of the sequence is 796.

DM5: Average monthly temperature from 1907 to 1972.[6] The total length of the sequence is 796.

DM6: Monthly data on Middle Boulder Creek at Nederland, CO from 1912 to 1960.[7] The total length of the sequence is 592.

DM7: Monthly electricity production in Australia (million kilowatt hours) from Jan 1956 to Aug 1995.[8] The total length of the sequence is 480.

DM8: Monthly flows of Chang Jiang (Yangtze River) at Han Kou, China from 1865 to 1979.[9] The total length of the sequence is 1372.

DM9: Monthly production of clay bricks (million units) from Jan 1956 to Aug 1995.[10] The total length of the sequence is 480.

DM10: Monthly rain in Coppermine (mm) from 1933 to 1976.[11] The total length of the sequence is 532.

DM11: Monthly riverflow (cms) in Pigeon River near Middle Falls, Ontario from 1924 to 1977.[12] The total length of the sequence is 640.

DM12: Monthly riverflow (cms) in Turtle River near Mine Centre, Ontario from 1921 to 1977.[13] The total length of the sequence is 676.

DM13: Monthly temperature in England (F) from 1723 to 1970.[14] The total length of

[5] https://datamarket.com/data/set/22w1/mean-monthly-precipitation-1907-1972

[6] https://datamarket.com/data/set/22o4/mean-monthly-temperature-1907-1972

[7] https://datamarket.com/data/set/22vt/middle-boulder-creek-at-nederland-co-1912-1960

[8] https://datamarket.com/data/set/22l0/monthly-electricity-production-in-australia-million-kilowatt-hours-jan-1956-aug-1995

[9] https://datamarket.com/data/set/22r8/monthly-flows-chang-jiang-at-han-kou-1865-1979

[10] https://datamarket.com/data/set/22lv/monthly-production-of-clay-bricks-million-units-jan-1956-aug-1995

[11] https://datamarket.com/data/set/22n8/monthly-rain-coppermine-mm-1933-1976

[12] https://datamarket.com/data/set/22mi/monthly-riverflow-in-cms-pigeon-river-near-middle-falls-ont-1924-1977

[13] https://datamarket.com/data/set/22mf/monthly-riverflow-in-cms-turtle-river-near-mine-centre-ont-1921-1977

[14] https://datamarket.com/data/set/22vp/monthly-temperature-in-england-f-1723-1970

the sequence is 2980.

Table 5.3 shows the mean squared prediction errors. CCNN-th maintains its lead on most datasets. Where it does not, different ICNNs alternately take the lead by small margins. Here are two comments. First, note that the kernel length in ICNN-C and ICNN-Q is much larger than L_K, so it falls beyond the representation power of CCNN. Nevertheless, this result shows that an interpolation kernel within the scope of Eq. (5.1) usually suffices to outperform the standard interpolation methods that fall beyond. Second, unlike in the simulated test, ICNN-L generally performs better than ICNN-C, which emphasizes the importance of choosing a suitable interpolation scheme for each task. CCNN, with its ability to choose its own kernels, avoids such trouble.

5.6.5 Speech Interpolation

Since CCNN is motivated by interpolation, it is insightful to see CCNN's performance in interpolation tasks. The speech interpolation task involves restoring the high-resolution speech signal from the downsampled signal.

The Dataset To mitigate the complexity in directly working on speech waveforms, the sampling rate of the high-resolution speech is set to 4 kHz, and that of the downsampled signals is 2 kHz. Three different downsampling schemes are tested. The first scheme, called *uniform filtered*, uniformly downsamples the speech to 2 kHz after passing it to an anti-aliasing filter. The second scheme, called *uniform unfiltered*, uniformly downsamples the signal without the anti-aliasing filter. The third scheme, called *nonuniform*, randomly preserves half of the speech samples, and thus the resulting signal has an average sampling rate of 2 kHz.

Our dataset consists of one hour of lab-recorded speech of one speaker reading structured composite sentences. We use 80% of the dataset as training, 10% as validation, and the rest of 10% as test. The high-resolution speech is chunked into 40-point sequences without overlap, and the corresponding downsampled speech into 20-point sequences.

Configurations Similar to the prediction experiment, the ICNN approaches interpolates

the low-resolution speech into high-resolution speech (4 kHz) before it is fed to a two-layer regular CNN. A similar practice has also been adopted in [81]. CCNN also has two layers. The first layer outputs at the uniform timestamps at the rate of 4 kHz, and the second layer is a regular convolutional layer. Detailed configurations are shown below. Again, the hyperparameters are set such that the two architectures have the same number of layers, receptive field size and model complexity. The activation functions for CCNN and ICNN are both sigmoid. The hyperparameters are detailed below.

CCNN-th: The timestamps are normalized such that the sampling interval of the original 4 kHz speech is 0.5. CCNN-th has two layers. The first layer is a CCNN layer that outputs the timestamps uniformly distributed at the rate of 4 kHz. The number of hidden channels is 16. The bias network has one layer, and the order $O_B = 28$. The bias kernel network has one layer, and the order $O_K = 7$. The kernel length $L_K = 10$. The two-hot encoding time interval $\delta = 0.5$. The second layer is a 1×28 convolutional layer.

ICNN-C: The CNN contains two 1×28 convolutional layers with 80 number of hidden filters.

Speech DNN & Speech DNN CP: Speech DNN converts both the high resolution and downsampled speech into amplitude spectra using FFT with 64ms window length and 48ms window shift. Speech DNN has 3 hidden layers, each of which has 1024 hidden nodes. Because our temporal resolution is half of that in [94], our hidden node number is a half of that in [94] too. The number of parameters in Speech DNN is around 2.3×10^6, which is much larger than that in CCNN and the ICNN baseline models (both have around 5×10^3 parameters).

Since sample-based speech interpolation methods have yet to achieve the state-of-the-art, we also include a spectrum-based benchmark from [94] called Speech DNN. Speech DNN only works on the uniform filtered case. It predicts the higher half of the amplitude spectrum of the high-resolution speech of that of the low-resolution speech. The flipped phase spectrum of the down-sampled speech is used as an estimate of the higher half of the phase spectrum of the high-resolution speech.

Table 5.4. Signal-to-Noise Ratio (dB) in Speech Interpolation. * Speech DNN does not work with non-filtered down-sampled signals and nonuniformly down-sampled signals.

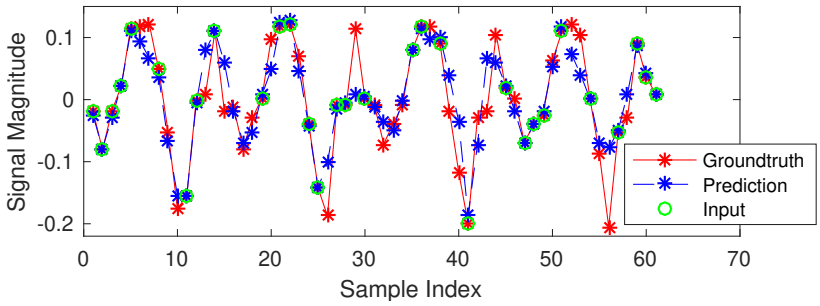
Alg.	Uniform		Non-uniform
	filtered	non-filtered	
Speech DNN	9.13	-*	-*
Speech DNN (CP)	**13.64**	-*	-*
ICNN-C	9.74	7.49	2.33
ICNN-Q	9.66	5.67	2.77
ICNN-L	9.72	5.81	3.16
ICNN-P	3.17	2.63	1.82
ICNN-S	9.62	5.90	2.83
CCNN-th	9.61	**7.80**	**6.58**

Since the phase spectrum estimate of Speech DNN can be inaccurate, we introduced another version, called Speech DNN with a cheated phase (CP), where the ground truth phase spectrum is applied. Note that this version is given too much oracle information to be fairly compared with. Nevertheless, we will include its results for reference.

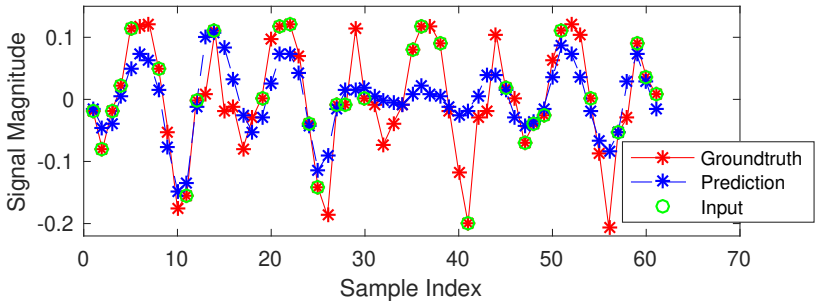
Results and Analysis. Tab. 5.4 shows the Signal-to-Noise Ratio (SNR) of the signals recovered from different input signals by different models. The Speech DNN with cheated phase yields the best SNR, because it uses the phase information from ground truth. However, the Speech DNN without cheated phase has similar performance to CNN and CCNN, even though it has much more weights, largely because of the inaccurate phase estimates.

As for the comparison between CCNN and ICNN, they have similar SNR under uniform sampling cases. This verifies that both architectures have similar representation power given uniform data. However, CCNN has much higher SNR than ICNN in nonuniform case. One important reason is that CCNN, by construction, is aware of whether neighboring samples are dense or sparse, and outputs robust interpolation kernels accordingly, despite the variation in sampling patterns; whereas CNN is unable to deal with various random sampling instances.

Figure 5.8 shows an example of the restored signal from the nonuniform samples by CCNN and CNN respectively. CCNN can restore some spikes (*e.g.* ones at 25 and 40) even without an input sample point in the spike, because CCNN can learn the continuous kernels and



(a) By CCNN



(b) By ICNN

Figure 5.8. Examples for CCNN and ICNN in restoring nonuniformly down-sampled speech. CCNN generates better approximation at especially at crests and troughs.

restore the original spikes. CNN model does poorly in restoring spikes even when there are input sample points in the spikes.

5.7 Conclusion

In this paper, we have introduced CCNN for nonuniform time series with two takeaways. First, interpolation before continuous convolution is shown to be a reasonable way for nonuniform time series. Second, learning task-specific kernels in a data-driven way significantly improves the performance. There are two promising directions. First, we have focused on 1D convolution, but this framework can be generalized to multi-dimensional nonuniform data. Second, while the computational complexity is similar for CCNN and CNN, the runtime of the former is much longer, because of the lack of parallelization. Fast implementation of CCNN is thus another

research direction.

5.8 Acknowledgement

This chapter, in full, is a reprint of the material as it appears in “Continuous cnn for nonuniform time series.“ Shi, Hui, Yang Zhang, Hao Wu, Shiyu Chang, Kaizhi Qian, Mark Hasegawa-Johnson, and Jishen Zhao. , In ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 3550-3554. IEEE, 2021. The dissertation author was the primary investigator and author of this paper.

Chapter 6

Neural Architectures III: Compiler Optimizer without Human Knowledge

Deep symbolic superoptimization refers to the task of applying deep learning methods to simplify symbolic expressions. Existing approaches either perform supervised training on human-constructed datasets that define equivalent expression pairs, or apply reinforcement learning with human-defined equivalent transformation actions. In short, almost all existing methods rely on human knowledge to define equivalence, which suffers from large labeling cost and learning bias. We thus propose HISS, a reinforcement learning framework for symbolic superoptimization that keeps humans outside the loop. HISS introduces a tree-LSTM encoder-decoder network with attention to ensure tractable learning. Our experiments show that HISS can discover more simplification rules than existing human-dependent methods, and can learn meaningful embeddings for symbolic expressions, which are indicative of equivalence.

6.1 Introduction

Superoptimization refers to the task of simplifying and optimizing over a set of machine instructions, or code [104, 123], which is a fundamental problem in computer science. As an important direction in superoptimization, symbolic expression simplification, or symbolic superoptimization, aims at transforming symbolic expression to a simpler form in an effective way, so as to avoid unnecessary computations. Symbolic superoptimization is an important component

in compilers, *e.g.* LLVM and Halide, and it also has a wide application in mathematical engines including Wolfram[1], Matlab, and Sympy.

Over recent years, applying deep learning methods to address symbolic superoptimization has attracted great attention. Despite their variety, existing algorithms can be roughly divided into two categories. The first category is supervised learning, *i.e.* to learn a mapping between the input expressions and the output simplified expressions from a large number of human-constructed expression pairs [6, 169]. Such methods rely heavily on a human-constructed dataset, which is time- and labor-consuming. What is worse, such systems are highly susceptible to bias, because it is generally very hard to define a minimum and comprehensive axiom set for training. It is highly possible that some forms of equivalence are not covered in the training set, and fail to be recognized by the model. In order to remove the dependence on human annotations, the second category of methods leverages reinforcement learning to autonomously discover simplifying equivalence [27]. However, to make the action space tractable, such systems still rely on a set of equivalent transformation actions defined by human beings, which again suffers from the labeling cost and learning bias.

In short, the existing neural symbolic superoptimization algorithms all require human input to define equivalences. It would have benefited from improved efficiency and better simplification if there were algorithms independent of human knowledge. In fact, symbolic superoptimization should have been a task that naturally keeps human outside the loop, because it directly operates on machine code, whose consumers and evaluators are machines, not humans.

Therefore, we propose Human-Independent Symbolic Superoptimization (HISS), a reinforcement learning framework for symbolic superoptimization that is completely independent of human knowledge. Instead of using human-defined equivalence, HISS adopts a set of unsupervised techniques to maintain the tractability of action space. First, HISS introduces a tree-LSTM encoder-decoder architecture with attention to ensure that its exploration is confined within the set syntactically correct expressions. Second, the process of generating a simplified expression is

[1] https://www.wolframalpha.com/

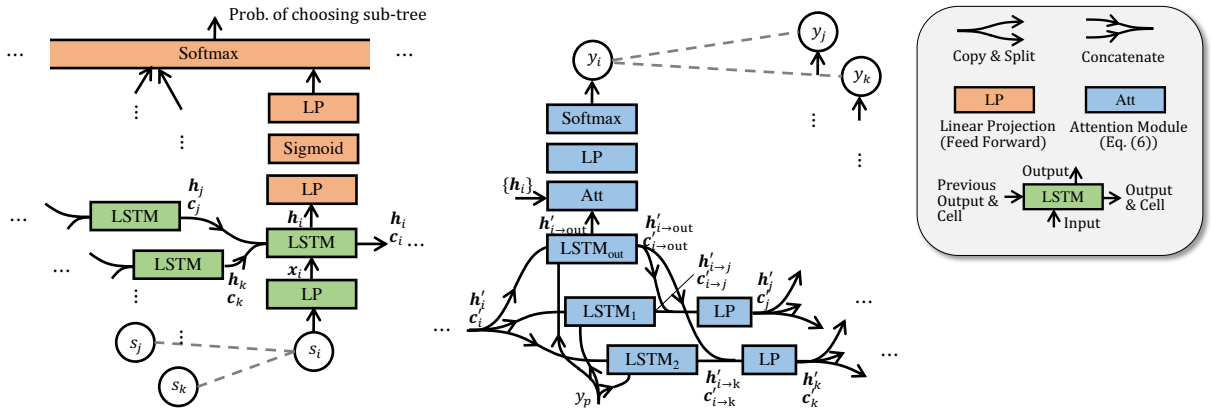
broken into two stages. The first stage selects a sub-expression that can be simplified and the second stage simplifies the sub-expression. We performed a set of evaluations on artificially generated expressions as well as a publicly available code dataset, called the Halide dataset [26], and show that HISS can achieve competitive performance. We also find out that HISS can automatically discover simplification rules that are not included in the human predefined rules in the Halide dataset.

6.2 Related Work

Superoptimization origins from 1987 with the first design of [104]. With the probabilistic testing to reduce the testing cost, the brute force searching is aided with a pruning strategy to avoid searching sub-spaces that contains pieces of code that have known shorter alternatives. Due to the explosive searching space for exhaustive searching, the capability of the first superoptimizer is limited to only very short programs. More than a decades later, [75] presented Denali, which splits the superoptimization problem into two phases to expand the capability to optimize longer programs. STOKE [123] follows the two phases but sacrifices the completeness for efficiency in the second phase.

Recent attempts to improve superoptimization are categorized into two fields: exploring transformation rules and accelerating trajectory searching. Searching the rules are similar to the problem of superoptimization on limited size program, but targeting more on the comprehensiveness of the rules. [16] exhaustively enumerates all possible expressions given the syntax and checks the equivalence of pairs of expressions by SMT solver. A similar method with an adaption of the SMT solver to reuse the previous result is proposed by [70]. On the other hand, deep neural networks are trained to guide the trajectory searching [18, 26].

Considering transformation rule discovery as a limited space superoptimization, the large action space and sparse reward are the main challenges for using neural networks. Special neural generator structures are proposed for decoding valid symbolic programs, which leverage the



(a) The tree encoder (green) and subtree selector (orange).

(b) The tree decoder.

Figure 6.1. The HISS architecture, illustrated on a three-node binary subtree, where i is the parent of j, k, and p is the parent of i.

syntax constraints to reduce the searching space as well as learn the reasoning of operations, and are gaining popularity in program synthesis [17, 112, 175], program translation [27, 45], and other code generation tasks [5, 98]. Among the symbolic expression decoders, the family of tree structure RNNs [5, 27, 45, 112] are more flexible than template-based predictors [98, 175].

6.3 The HISS Architecture

In this section, we will detail our proposed HISS architecture. We will first introduce a few notations. $\mathscr{T}$ denotes a tree; $\boldsymbol{a}$ denotes a vector, and $\boldsymbol{A}$ denotes a matrix. We introduce an LSTM$(\cdot)$ function that summarizes standard one-step LSTM operation as

$$[\boldsymbol{h}_t, \boldsymbol{c}_t] = \text{LSTM}(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}, \boldsymbol{c}_{t-1}), \tag{6.1}$$

where $\boldsymbol{h}_t$, $\boldsymbol{c}_t$ and $\boldsymbol{x}_t$ denote the output, cell and input at time t of a standard LSTM respectively.

6.3.1 Framework Overview

Our problem can be formulated as follows. Given a symbolic expression $\mathscr{T}_I$, represented in the *expression tree form*, our goal is to find a simplified expression $\mathscr{T}_O$, such that 1) the two expressions are equivalent, and 2) $\mathscr{T}_O$ contains a smaller number of nodes than $\mathscr{T}_I$.

It is important to write the symbolic expressions in their expression tree form, rather than strings because HISS will be operating on tree structures. An expression tree assigns a node for each operation or variable. Each non-terminal node represents an operation, and each terminal node, or leaf node, represents a variable or a constant. The arguments of operation are represented as the descendant subtrees of the corresponding node. Compared to string representation, tree representation naturally ensures any randomly generated expression in its form is syntactically correct. It also makes working with subexpressions easier: simply by working with subtrees.

HISS approaches the problem using the reinforcement learning framework, where the action of generating simplified expressions are divided into two consecutive actions. The first action is to pick a subexpression (or subtree) that can be simplified, and the second action generates the simplified expression for the selected subexpression.

Accordingly, HISS contains three modules. The first module is a **tree encoder**, which computes an embedding for each subtree (including the entire tree) of the input expression. The embeddings are useful for picking subtree for simplification as well as simplifying a subtree. The second module is a **subtree selector**, which selects a subtree for simplification. The third module is a **tree decoder** with an attention mechanism, which generates a simplified expression based on the input subtree embedding. The subsequent subsections will introduce each module respectively.

6.3.2 The Tree Encoder

The tree encoder generates embedding for every subtree of the input expression. We apply the N-ary Tree LSTM as proposed in [142], where N represents the maximum number of arguments that an operation has. It is important to note that although different operations have a different number of arguments, for structural uniformity, we assume that all operations have N arguments, with the excessive arguments being a NULL symbol.

The tree encoder consists of two layers. The first layer is called the embedding layer, which is a fully-connected layer that converts the one-hot representation of each input symbol to

an embedding. The second layer is the tree LSTM layer, which is almost the same as the regular LSTM, except that the cell information now flows from the children nodes to their parent node. Formally, denote $\boldsymbol{c}_i$, $\boldsymbol{h}_i$, and $\boldsymbol{x}_i$ as the cell, output and input of node i respectively. Then the tree LSTM encoder performs the following information

$$[\boldsymbol{h}_i, \boldsymbol{c}_i] = \text{LSTM}\left(\boldsymbol{x}_i, \bigcup_{j \in \mathbb{D}(i)} \boldsymbol{h}_j, \bigcup_{j \in \mathbb{D}(i)} \boldsymbol{c}_j\right), \tag{6.2}$$

where $\mathbb{D}(i)$ denotes the set of children of node i. Fig. 6.1(a) plots the architecture of the tree LSTM encoder (in green). Since each node fuses the information from its children, which again fuse the information from their own children, it is easy to see that the output $\boldsymbol{h}_i$ summarizes the information of the entire subtree led by node i, and thus can be regarded as an embedding for this subtree.

6.3.3 The Subtree Selector

The subtree selector performs the first action to select a subtree for simplification. It takes the output of the tree encoder, $\{\boldsymbol{h}_i\}$, as its input, and produces the probability with which each subtree is selected. It consists of two feed-forward layers followed by a softmax layer across all the nodes in the input tree. Figure 6.1(a) shows the architecture of the subtree selector (in orange).

6.3.4 The Tree Decoder

Once a subtree has been selected, and suppose the root node of the selected subtree is node i, the output of the encoder at node i, $\boldsymbol{h}_i$, is then fed into the tree decoder, which generates a simplified version of the subtree. The tree decoder can be regarded as the inverse process of the tree encoder: the latter fuses information from the children to the parents, whereas the former unrolls the information from parents down to the entire N-ary tree. When the parent node outputs a non-operation symbol, the expansion of this branch terminates and no child is further decoded.

The tree decoder adopts a novel LSTM architecture with attention, which, compared with the attention LSTM proposed by [27], is more parameter- and computationally-efficient. It consists of two layers. The first layer is a tree LSTM layer, and the second layer is the symbol generation layer with attention. Figure 6.1(b) illustrates the decoder structure. The decoder shares the same vocabulary with the encoder, and the embedding layer of the decoder shares the parameters with the embedding layer in the encoder.

Tree LSTM Layer. The tree LSTM in the decoder needs to accomplish two tasks. First, it needs to extract the information for generating the output for the current node. Second, it needs to split and pass on the information to its children. To better control the information flow, we introduce two tracks of LSTMs for the two different tasks. Formally, denote $[\boldsymbol{h}'_i, \boldsymbol{c}'_i]$ as the output and cell of node i, and assume $[j_1, \cdots, j_N]$ are children nodes of i. Also denote $\boldsymbol{y}_p$ as the decoder output for node p, which is the parent node of node i (If node i is already the root node of the selected subtree, then $\boldsymbol{y}_p$ becomes a special start token). Then the first LSTM track extracts the information that generates the current output:

$$[\boldsymbol{h}'_{i\to\text{out}}, \boldsymbol{c}'_{i\to\text{out}}] = \text{LSTM}_{\text{out}}(\boldsymbol{y}_p, \boldsymbol{h}'_i, \boldsymbol{c}'_i). \tag{6.3}$$

The second LSTM track splits and passes on the information to the children, *i.e.* $\forall n \in \{1, \cdots, N\}$

$$[\boldsymbol{h}'_{i\to j_n}, \boldsymbol{c}'_{i\to j_n}] = \text{LSTM}_n(\boldsymbol{y}_p, \boldsymbol{h}'_i, \boldsymbol{c}'_i). \tag{6.4}$$

Notice that we have appended a subscript to the LSTM$(\cdot)$ to emphasize that LSTM functions with different subscripts do not share parameters. Finally, the LSTM information for a specific children is derived by linearly projecting the output track and that specific children track:

$$\boldsymbol{h}'_{j_n} = \boldsymbol{W}_h[\boldsymbol{h}'_{j\to\text{out}}, \boldsymbol{h}'_{i\to j_n}] + \boldsymbol{b}_h, \quad \boldsymbol{c}'_{j_n} = \boldsymbol{W}_c[\boldsymbol{c}'_{j\to\text{out}}, \boldsymbol{c}'_{i\to j_n}] + \boldsymbol{b}_c. \tag{6.5}$$

We find that this linear projection is useful for adding additional dependencies between the parent

output and the descendants, so that the generated expression is more coherent.

Symbol Generation Layer with Attention. The symbol generation layer takes the output track produced by the previous tree LSTM layer, $\boldsymbol{h}'_{i\to\text{out}}$, as input, and outputs the probability distribution of generating different output symbols for the current node. It adopts an attention mechanism [8] to attend to the relevant part in the encoder, so that the input and output expressions have better correspondence. Formally, when generating the output for decoder node i, the attention weight on encoder node j is computed from $\boldsymbol{h}'_{i\to\text{out}}$ and $\boldsymbol{h}_j$ as follows:

$$\begin{aligned} e_i(j) &= \boldsymbol{v}^T \tanh(\boldsymbol{W}_d \boldsymbol{h}'_{i\to\text{out}} + \boldsymbol{W}_e \boldsymbol{h}_j + \boldsymbol{b}_a), \\ [a_i(1), \cdots, a_i(J)] &= \text{softmax}([e_i(1), \cdots, e_i(J)]), \end{aligned} \tag{6.6}$$

where J is the total number of input nodes at the encoder. Finally, the probability of symbol generation at node i, denoted as $\boldsymbol{p}_i$, is computed by passing into a linear projection layer $\boldsymbol{h}'_{i\to\text{out}}$ and an attention context vector $\boldsymbol{c}_i$, which is a linear combination of the encoder embeddings with the attention weights, *i.e.*

$$\boldsymbol{p}_i = \boldsymbol{W}_o[\boldsymbol{h}'_{i\to\text{out}}; \boldsymbol{c}_i] + \boldsymbol{b}_o, \quad \text{where } \boldsymbol{c}_i = \sum_{j=1}^{J} a_i(j)\boldsymbol{h}_j. \tag{6.7}$$

6.4 Learning with HISS

In this section, we will elaborate on the training and inference schemes of HISS. In particular, we will introduce several mechanisms to improve the exploration efficiency of HISS.

6.4.1 Training

We apply the standard REINFORCE framework [154] for training, where the reward function is given by

$$R(\mathscr{T}_I, \mathscr{T}_O) = \gamma^{\text{card}(\mathscr{T}_O)} \text{ if } \mathscr{T}_I \equiv \mathscr{T}_O, \quad -\beta\gamma^{\text{card}(\mathscr{T}_O)} \text{ otherwise}, \tag{6.8}$$

where '$\equiv$' denotes that the two expressions are equivalent; card$(\cdot)$ denotes the number of nodes in the tree expression, or the length of the expression. β is a hyperparameter that depicts the penalty of not producing an equivalent expression. This reward prioritizes equivalence, and given equivalence, favors shorter expressions. We applied a probabilistic testing scheme to determine equivalence as proposed in [104]. For each input, multiple outputs are decoded via beam search, on each of which a reward is evaluated. We introduce the following mechanisms to maintain the efficiency and stability of training.

Curriculum Learning. Since generating the simplified expression is divided into two actions, subtree selection and subtree simplification, directly learning both can lead to very inefficient exploration. Instead, we introduce a two-stage curriculum. The first stage trains only the encoder and decoder on very short expressions (maximum depth less than four). The subtree selector is not trained. Instead, we always feed the entire tree to the decoder for simplification. The second stage trains all the modules on longer expressions.

Subtree Embedding Similarity. In order to guide the encoder to learning meaning embeddings, we introduce an additional ℓ_2 loss to enforce that the equivalent expressions have similar encoder embeddings, *i.e.* similar $\boldsymbol{h}_i$s . Specifically, for each input expressions $\mathscr{T}_I$, we decode a set of generated expressions $\mathbb{S} = \{\mathscr{T}_O\}$ with beam search, and obtain their embeddings $\{\boldsymbol{h}(\mathscr{T}_O)\}$ by feeding them back into the encoder (here we add an argument to $\boldsymbol{h}$ to emphasize that the embedding is a function of input expression). Then the ℓ_2 loss is expressed as follows

$$L = \frac{1}{|\mathbb{S}|} \sum_{\mathscr{T}_O \in \mathbb{S}} \|\boldsymbol{h}(\mathscr{T}_O) - \boldsymbol{h}(\mathscr{T}_I)\|_2^2 \cdot (-1)^{\mathbb{1}[\mathscr{T}_I \neq \mathscr{T}_O]}, \tag{6.9}$$

where $\mathbb{1}[\cdot]$ denotes the indicator function, which equals one if the statement in its argument is true, and zero otherwise. Note that this ℓ_2 applies to the encoder only, and can be optimized by regular gradient descent methods. REINFORCE is not needed.

6.4.2 Inference

Similar to training, the inference is performed by decoding multiple outputs via beam search and finding the best result as the final output. In order to accelerate the inference process, we introduce an offline procedure. During the first stage of the curriculum training, *i.e.* training on very short expressions, all the simplified equivalence rules discovered are logged. During inference, if the subtree to be fed into the decoder has an exact match in the log, we will apply the logged simplified equivalence directly, rather than redoing the entire decoding process.

6.5 Experiments

We performed two experiments. The first experiment compares HISS with human-independent naive search algorithms. The second experiment compares HISS with existing human-dependent state-of-the-art systems on benchmark datasets. Throughout all the experiments, we use the same hyperparameter setting.

6.5.1 Hyperparameter Setting and Determination

The input to the network is one-hot encoded sequences where the vocabulary size is 50, then the input is encoded by a single fully connected layer with output size 32. The hidden units of LSTM are set to 64 for both encoder and decoder, as a common setting adopted in many previous works [99], and the number of layers is 1. The output size of the encoder is 64, and the output size of the decoder is equal to the vocabulary size (50). The subtree selector consists of two feed-forward layers with output sizes of 128 and 1 respectively. The model is trained with the ADAM optimizer with a learning rate of 1e-3. Rather than tuned on the validation set, this hyperparameter setting is determined by following the common setting in previous works ([99]) as well as referencing to the Halide vocabulary size. The same hyperparameter setting has been applied throughout this research project. Finetuning the hyperparameters is expected to have a minor effect on the performance as compared to refining our major algorithm design, *e.g.*

Table 6.1. The vocabulary of the symbols and operators that are used to construct the traverse equivalence dataset.

Type	Symbols
Operators	min, max, $\geq$, $\leq$, $<$, $>$, ==, !, $\neq$, select, +, -, *, /, &&, ||
Constants	0, 1, 2, True, False
Variables	v0, v1, ..., v14
Constant symbols	c0, c1, ... c13

introducing subtree selector and ℓ_2 embedding regularizer, which will be discussed in further details in the ablation studies in Section 6.5.9.

The penalty of not getting an equivalent expression, β (as in Equation (6.8)), is set to 0.1. This is motivated by our observation that in the Monte Carlo sampling results the ratio of equivalent expressions over nonequivalent expressions is roughly 10:1 on a randomly initialized model. Therefore, by matching β to this ratio we can balance the reward and penalty and achieve an average reward of around zero, at least for the initial iterations, which is shown to contribute to a more stable REINFORCE training. However, please be reminded that β is not a crucial parameter because the baseline removal process in REINFORCE would automatically balance the reward and penalty. A good choice of β would mostly only benefit the onset of the training when the baseline estimation is not yet accurate. The weight to embedding similarity loss term is set to 0.1.

6.5.2 Comparing with Human-Independent Methods

Since there are no existing human-independent methods specifically for symbolic super-optimization, we compare several search algorithms. Due to the search complexity, the evaluation cannot be performed on very long expressions. Therefore, this experiment is performed on the traverse equivalence dataset.

Traverse Equivalence Dataset As a complement to the Halide dataset, we build a dataset by traversing all the possible expressions with maximum depth four that consist of operations and symbols drawn from the Halide vocabulary in table 6.1. Among these expressions, we

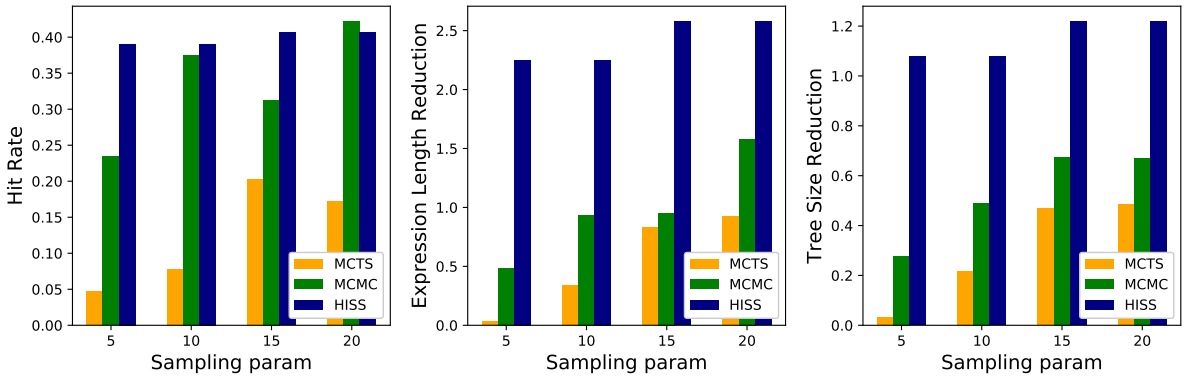


Figure 6.2. Comparison with human-independent methods in terms of hit rate (left), expression length reduction (middle), tree size reduction (right), on Traverse Equivalence dataset.

use the FingerPrint method [71] to test if they can be simplified, and discard those that cannot. From the remaining expressions, we sample 900 expressions as the training set, 300 as the validation set and 300 as the test set. The advantage of this dataset is that it is not built from human-predefined equivalence relation. However, the disadvantage of this dataset is that it does not contain expressions with a maximum depth greater than four, limited by the complexity of the FingerPrint method. Additional details of our Traverse Equivalence dataset can be found in Section 7.2.

Training Since HISS does not operate on very long expressions, it is only trained with stage-one in curriculum learning (the one with no subtree selection). Additional details regarding training can be found in Section 6.5.5.

Baselines Two baseline searching methods are compared: Monte Carlo Tree Search (MCTS) [10] and Markov Chain Monte Carlo (MCMC) [123]. MCTS decides the expression tree from root to leaves, choosing one symbol from Halide vocabulary for each node, and adopts Upper Confidence Bound [83] for balancing exploration and exploitation. Similar to [123], MCMC takes one of four transformations: 1) replace an operator by another random operator, and generate or discard operands if two operator takes a different number of operands. 2) replace a variable/constant with another random variable/constant. 3) replace a subexpression with a random single variable/constant. 4) replace a variable/constant with a random expression. The

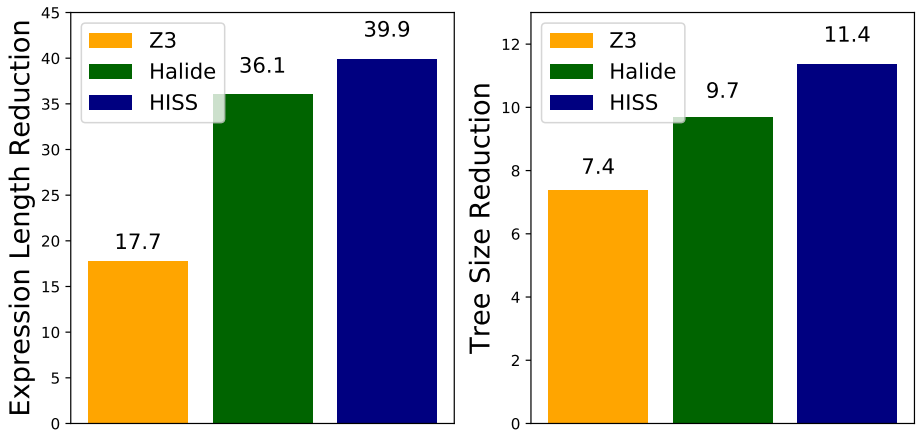


Figure 6.3. Comparison with human-dependent methods in terms of expression length reduction (left), and tree size reduction (right), on the Halide dataset.

probability distribution of taking the transformation is defined as the same as in [123].

Metrics. Three metrics are introduced: 1) *hit rate*, defined as percentage of expressions that the model successfully found an equivalence given the computation budget parameter; 2) *expression length reduction*, defined as reduction in the total number of tokens; and 3) *tree size reduction*, defined as reduction in the number of nodes in the expression tree.

Results. The performance comparison of three models is shown in figure 6.2. The sampling parameter in the horizontal axis refers to the beam size for HISS , the max trials budget for MCTS for each token decoded, and the sampling budget for MCMC. These quantities equivalently define the number of search attempts per token. As can be seen, HISS is significantly more powerful in finding the simpler equivalent than MCTS and MCMC. MCMC performs almost equally well as HISS in terms of Hit Rate, and both of them far outperform MCTS. However, both MCTS and HISS adopt top-down decoding in the huge decoding space, while MCMC starts with the input expression and applies local transformations, which makes it much easier to find an equivalence. Also, MCMC achieves much worse average length reduction and average tree size reduction than HISS.

Table 6.2. Example simplification rules learned by HISS (left) and their corresponding rules listed in Halide (right). If the learned rule has no corresponding preset rule in Halide, the row is left blank.

HISS		Halide	
Input	**Output**	**Input**	**Output**
(c0+v0) -(v0+c0)	0		
(c0/v0)*v0	c0	(y*x)/x	y
c0-c0	0	x-x	0
c0 $\leq$ c0	true	x $<$ x	false
c0 == c0	true	x == x	true
min(c0, v0) $\leq$ v0	true	x $<$ min(x, y)	false
min(c0 + v0, c1) $\leq$ (c0 + v0)	true		
min(c0, (min(v0, c1) + c2) + c1) $\leq$ ((c1 + c2 + c1)	true		
(c0 + v0) - c0	v0	(x + y) - x	y
max(c0, c0)	c0	max(x, x)	x
min(c0, v0 + c1) $\leq$ (c1 + v0)	true		
c0 $\leq$ max(v0, c0)	true	max(y, x) $<$ x	false
c0 / c0	1	x / x	1
min(c0 * v0, c1) $\leq$ (c0 * v0)	true		
c0 + (v0 - c0)	v0	x + (y - x)	y
min(min(c0, v0), c1 - c2) $\leq$ v0	true		
min(min(c0, v0), v1) $\leq$ v0	true		

6.5.3 Comparing with Human-Dependent Methods

In this section, we compare HISS with existing human-dependent state-of-the-art methods on the Halide dataset.

Halide Dataset The Halide dataset [26] is the benchmark dataset for symbolic expression simplification. It contains around 10,000 training sequences, 1,000 testing sequences, and 1,000 validation sequences, generated and split randomly. The number of words for each sequence ranges from 6 to 100, averaged at 58. The expressions generated contain many constants beyond $\{0, 1, 2, \text{True}, \text{False}\}$, and the constant is renamed to constant symbols shown in table 6.1, and the same constant value is renamed to the same constant symbol. There are at most 14 constant symbols and at most 15 variables in a single expression. The dataset is constructed by reversely applying the human-defined simplification rules, so the target expression in the dataset might

not be in the simplest form. Also because of HISS is an unsupervised method, only the longer expressions from each pair are used for training.

Training & Inference In this experiment, HISS is trained with both stages of curriculum learning. The first stage is trained on the moderately short expressions from the Traverse Equivalence dataset, whose configuration follows that in section 6.5.2. The second stage is trained on the Halide training set in an unsupervised manner. More details can be found in Section 6.5.5.

Since HISS only simplifies one subtree at a time, while the actual simplification usually requires sequentially simplifying multiple subtrees, we iteratively invoke the HISS procedure for both training and inference, as in [26]. Table 6.3 shows an example of the iterative process. The iterations terminate when 1) the number of iterations reaches 20; 2) the simplification output becomes a single node; or 3) the subtree selector assigns small scores (below 0.05) to all subtrees.

Baselines Two baselines are included: 1) *Halide* [118], which applies Halide predefined rules; 2) *Z3*, the simplification function in Z3, a high-performance theorem prover developed by [39], to perform transformations using the Halide predefined rules. It is worth mentioning that both baselines have access to the Halide predefined rules that are used to construct the dataset, which gives them an advantage over HISS.

Metrics Expression length reduction and tree size reduction are applied as the metrics.

Results Figure 6.3 shows the performance of HISS compared with the baselines. As can be seen, HISS outperforms both Halide and Z3 in both metrics. This result is quite non-trivial because the Halide dataset is built exclusively from the Halide's ruleset, to which both baseline algorithms have access. Therefore, this result implies that even for expressions that are specifically designed to be simplified by a set of predefined rules, they can still be further simplified by rules that are outside of the set.

6.5.4 Simplification Process Analysis

In this subsection, we will provide an in-depth analysis of the simplification process of HISS and Halide, which can explain why human-predefined rules can be insufficient, and how HISS can exceed the limit of human-predefined rules.

Example Simplification Rules To start with, table 6.2 lists some example simplification rules learned by HISS . For each example rule, the corresponding human-predefined rule in Halide is also listed if available. There are two observations. First, HISS is able to learn the most fundamental axiomatic identities such as the inverse relationship between plus and minus, and between multiplication and division, most of which can be matched with the human-predefined rules in Halide. Second and more importantly, HISS can also uncover some more complicated rules that have no matches in Halide, nor could be equivalently derived from any composition of the rules in Halide. A close inspection into these rules reveals that these rules, despite their complexity, are still very fundamental, and therefore the failure to include these rules is expected to impact the simplification performance. In addition to these fundamental rules, HISS is also able to find some involved but interesting rules, which are listed in table 6.4 and discussed in Section 6.5.7.

Example Simplification Traces. To illustrate how the completeness of the rules can impact on the simplification performance, table 6.3 compares the simplification traces of Halide and HISS for the following expression

$$(((((144 - (v0*72))/2)*2) + 4) \leq ((150 - (v0*72))/4)*4) \tag{6.10}$$

For Halide, each step represents the process of applying one Halide predefined rule to simplify the expression. For HISS, each step represents one simplification iteration. As each step, the subtree that is chosen for simplification for the next step is marked with box, unless the entire expression is chosen. As can be seen, HISS follows some reasonable steps to trim the constants

Table 6.3. Simplification process of HISS (upper) and Halide (lower) on Equation 6.10. The subtrees selected for simplification in the next iteration are marked in bold unless the entire tree is selected.

Step	HISS
0	(((((144 - (v0*72))/2)*2) + 4) $\leq$ **((150 - (v0*72))/4)*4)**
1	**(((((144 - (v0*72))/2)*2) + 4)** $\leq$ (150 - (v0*72)))
2	(((144 - (v0*72)) + 4) $\leq$ (150 - (v0*72))
3	((144 - (v0*72)) $\leq$ (146 - (v0*72)))
4	**(((v0*72) -(v0*72))** $\leq$ 2
5	0 $\leq$ 2
6	True
Step	**Halide**
0	(((((144 - (v0*72))/2)*2) + 4) $\leq$ **((150 - (v0*72))/4)*4)**
1	((((72 - (v0*36))*2) + 4) $\leq$ (((150 - (v0*72))/4)*4))
2	((((72 - (v0*36))*2) + 4) $\leq$ ((37.5 - (v0*18))*4))
3	!((37.5 - (v0*18))*4)) $<$ ((((72 - (v0*36))*2) + 4)

and eliminates v0, but Halide gets stuck after some trivial constant reduction. The reason for this failure is there are no such rules as $(((c0-(v0*c1))/c2)*c2 \mapsto c0-(v0*c1)$ or $x+y \mapsto y+x$ or $(x+y)*z \mapsto x*z+y*z$ in the Halide ruleset. Of course, one can fix this problem one time by appending the aforementioned rules to the ruleset, but this does not fundamentally solve the problem because one would never be able to exhaustively list all the possible rules needed for simplification in the ruleset.

Why Subtree Selector Matters. Table 6.3 also illustrates why the subtree selector is an integral part of HISS. In many reduction steps, only a subexpression is simplified (as boxed). Without the subtree selector, the decoder of HISS would have to process the entire expression only to simplify the subexpression, which makes it hard to effectively learn via reinforcement learning. For more analysis on the effect of the subtree selector, please refer to Section 6.5.9.

6.5.5 Training and Inference

As mentioned, the training for section 6.5.2 involves only the first stage in the curriculum learning, whereas the training for section 6.5.3 involves both stages in the curriculum learning. Below are the details regarding the training and inference schemes.

Training Iterations For the stage-one training in both experiments, we apply an identical setting, where the encoder-decoder model is trained for 10 epochs (90,000 steps). The model with the highest hit rate on the validation set is selected for evaluation in section 6.5.2 as well as for the stage-two training of the curriculum. The stage-two training takes two weeks to train the HISS for full simplification pipeline on RTX 2080 for 40 epochs (400k steps).

Beam Search The beam search algorithm is performed as follows. In the beginning, the top k choices of the root node with the highest probability are decoded. Then for each choice of root node value, the next step would be to decode the top k choices of each child node of the root node. Since the decoding processes of different child nodes of the same parent node are independent, we then perform the Cartesian product of all the k choices for each node at the current step, and preserve the top k combinations with the highest probability. Repeating this way, at step t, beam search decodes k highest probability tree up to depth t. Finally, top s ($s < k$) expressions are backtracked and used to estimate the expected reward of the model. The probability of beam search decoded expression is re-normalized according to [17]. In all the experiments, we set beam size $k = 20$ and $s = 20$.

6.5.6 Attention Visualization

To understand the attention mechanism, we visualize the attention to the input sequence shown in figure 6.4. We find that when decoding an operator, the attention tends to be flat (with a few exceptions in the right two figures), because it needs to understand the overall logic. This is different from machine translation or summarization, where the output attention of a single word is usually focused on several input tokens. On the other hand, when decoding a variable,

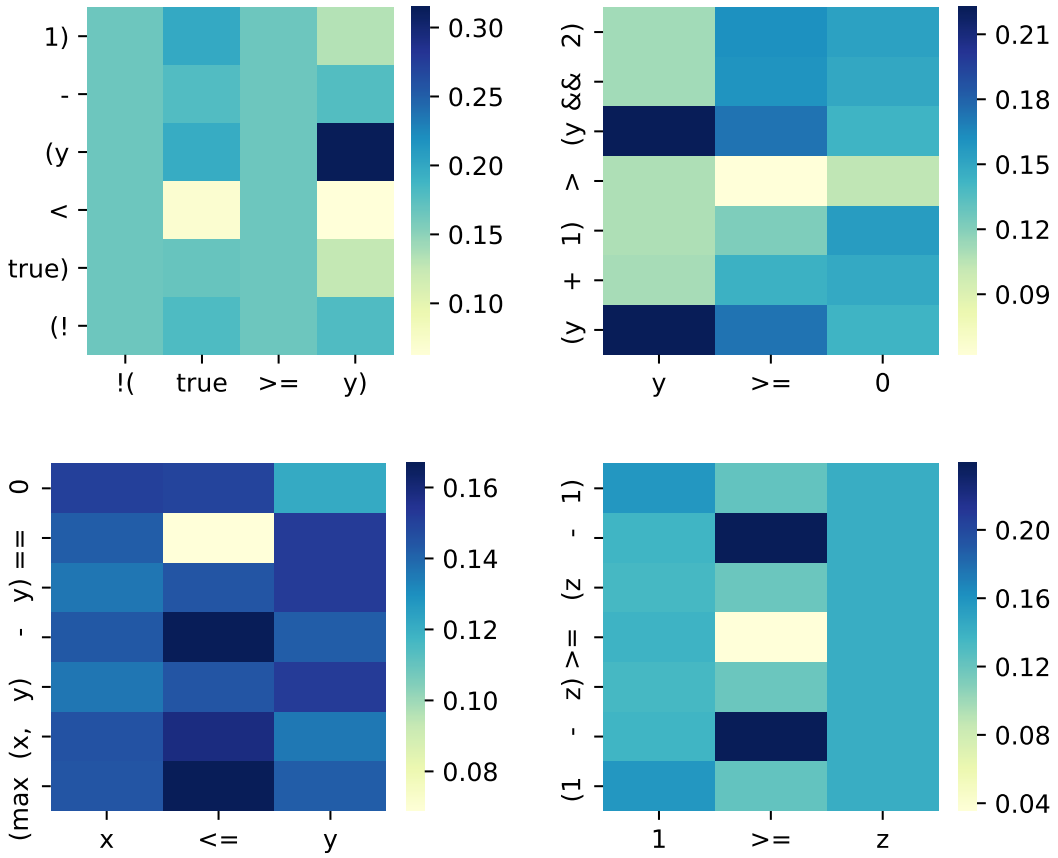


Figure 6.4. Attention weights on the input sequences for each token decoded. The x-axis shows the output sequence, and the y-axis shows the input sequence. Tokens are re-arranged in the natural order for better visualization.

the model attends sharply to the corresponding variable in the input.

6.5.7 Examples of Involved Simplification Rules

In order to further appreciate the ability of HISS in finding equivalence, in addition to the rules listed is table 6.2, we list some simplification rules discovered by HISS on randomly generated expressions that are more involved in table 6.4. In fact, it takes the authors quite a while to figure out the equivalence. These rules are hardly useful in practice because no humans will code in this way, but it is a vivid illustration of the advantage of HISS in finding powerful simplifications beyond human knowledge.

Table 6.4. Involved simplification rules discovered by HISS.

Input	Output
$(!\text{true}) < (y-1)$	$!(\text{true} \geq y)$
$(y-\text{true})\|\|(y \geq \text{true})$	true
$\max(z,1) \geq (\text{false}\|\|x)$	true
$(1-z) \geq (z-1)$	$1 \geq z$
$\min(x,\text{true})\&\&(z/2)$	$x\&\&z$
$1-(1<x)$	$1 \geq x$
$\text{select}(z,z,\text{true}) == \text{select}(z,\text{false},\text{true})$	$!z$
$(x*y)\&\&\text{true}$	$y\&\&x$
$(y+1) > (y\&\&2)$	$y \geq 0$

Table 6.5. Mean and standard deviation of the performance metrics among random initialization. The experiment setting is the same as in section 6.5.2.

	Sampling Parameter			
	5	10	15	20
Hit Rate	0.391 ± 0.013	0.391 ± 0.009	0.406 ± 0.014	0.406 ± 0.014
Expression Length Reduction	2.250 ± 0.172	2.250 ± 0.140	2.578 ± 0.242	2.578 ± 0.242
Tree Size Reduction	1.078 ± 0.077	1.078 ± 0.069	1.219 ± 0.114	1.219 ± 0.114

6.5.8 Robustness Against Random Initialization

To assess if the performance of HISS is robust to random model initialization, we perform the same experiment in section 6.5.2 eight times with different random initialization and compute the mean and standard deviation of the three metrics, which are shown in table 6.5. Compared to the absolute value of the mean, the standard deviation is very small, which shows that the random initialization has a minor influence on the performance of HISS.

6.5.9 Ablation Studies

In this section, we introduce a set of ablation studies that investigate the significance of the major components of HISS in terms of contribution to performance. The major components of interest are the subtree embedding similarity loss as in Equation (6.9), the subtree selector as introduced in section 6.3.3, and the tree-LSTM encoder-decoder architecture as in sections 6.3.2 and 6.3.4.

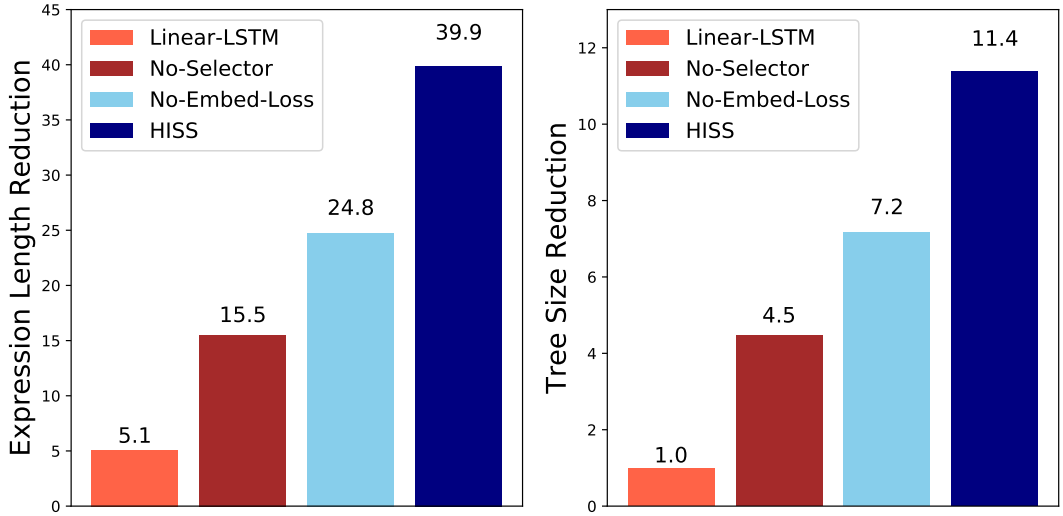


Figure 6.5. Performance of different variants of HISS on the Halide dataset.

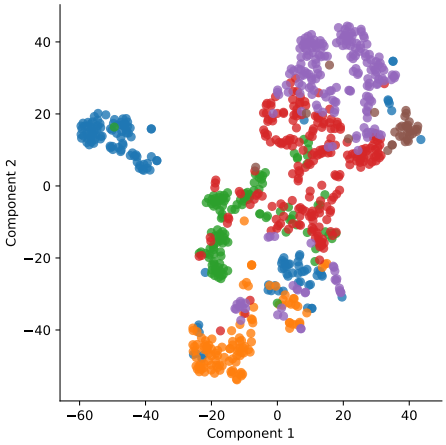
As an overview, figure 6.5 compares the performance between HISS and the following variants of HISS on the Halide dataset.

• *No-Embed-Loss:* The original HISS trained without the subtree embedding similarity loss.

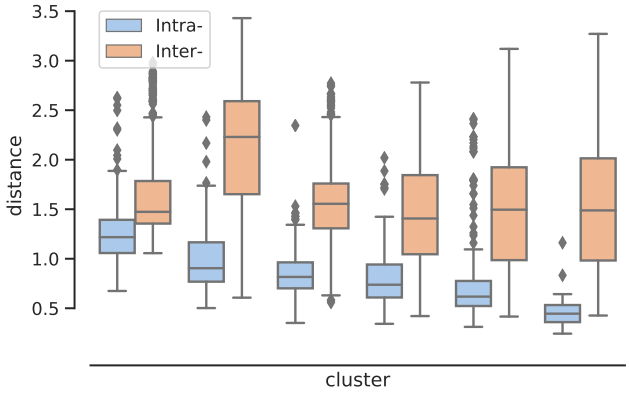
• *No-Selector:* The original HISS without the subtree selector.

• *Linear-LSTM:* The Tree-LSTM structure is removed and a simple linear-LSTM is used instead. The embedding loss as well as the subtree selector are no longer applicable and therefore also removed.

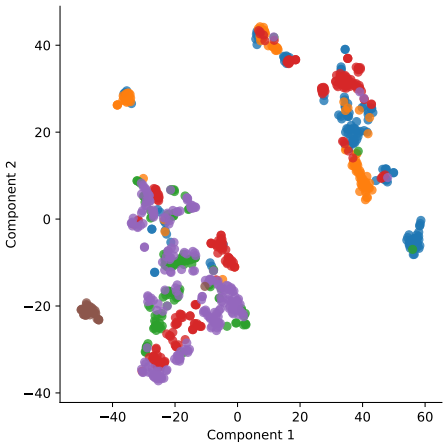
Other than the variations aforementioned, all the experiment settings are identical to the experiment in section 6.5.3. As can be seen in figure 6.5, without training with the embedding similarity loss or the subtree selector, the performance is significantly compromised. Furthermore, removing the tree-LSTM structure leads to almost a complete failure. The subsequent subsections further investigate why each of these modules has such a significant impact on the performance respectively.



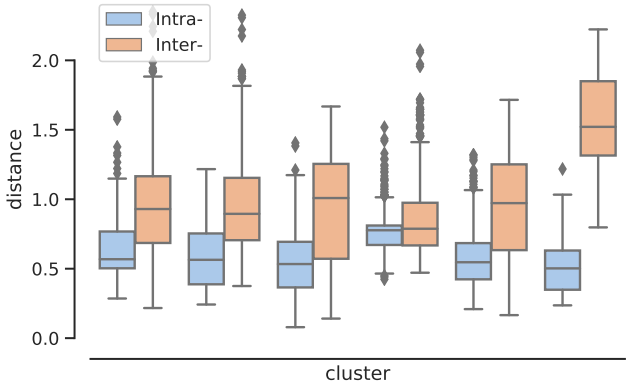
(a-1) Projected embedding vectors that acquired from the original HISS model.



(a-2) Box plots of intra-(blue) and inter-(orange) subset distance of the embeddings trained with ℓ_2 loss.



(b-1) Projected embedding vectors that acquired from the No-Embed-Loss HISS model.



(b-2) Box plots of intra-(blue) and inter-(orange) subset distance of the embeddings trained without ℓ_2 loss.

Figure 6.6. Evaluation of similarity of the embeddings of equivalent expressions.

6.5.10 Subtree Embedding Similarity

We would like to investigate the specific effects that introducing the embedding similarity loss brings.

The direct goal of the embedding similarity loss is to better cluster the embeddings whose corresponding expressions are equivalent. Therefore, we would like to first check if this direct goal is achieved. To evaluate this, in the experiment described in section 6.5.2, we select the six most-populated subsets of equivalent expressions in the test set and evaluate the similarity of

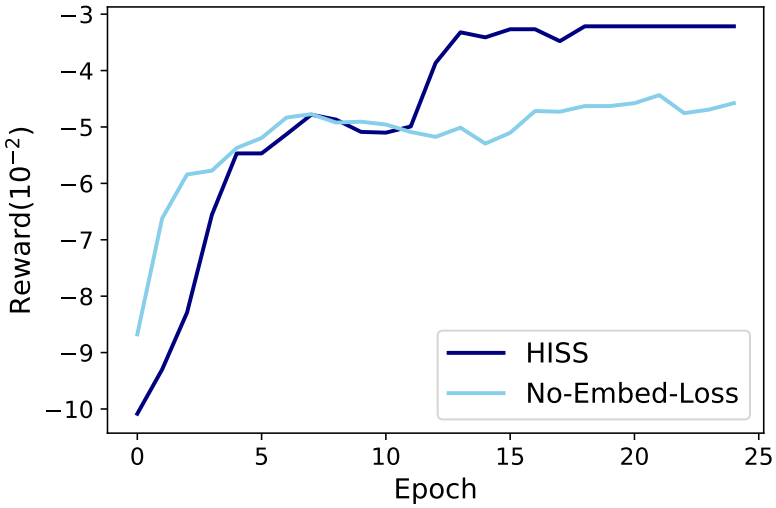


Figure 6.7. Expected reward of HISS with the embedding similarity loss (HISS) and without the embedding similarity loss (No-Embed-Loss)

the embeddings computed by HISS in two ways. First, the embeddings are further projected to two-dimensional space using t-SNE [102], which forms a scatter plot as in Figure 6.6(a-1). The points corresponding to equivalent expressions are shown in the same color. As can be seen, the embeddings equivalent expressions are highly clustered. Notice that this result is on the low-dimensional projection of the embedding. To better evaluate their similarity in the original space, we compare their inter- and intra-subset distances. The inter-/intra-subset distance of a subset is defined as the Euclidean distance between the centroid of the subset and the samples outside/within the subset. Figure 6.6(a-2) illustrates the box plot of these distances. Left plots ((a-1) and (b-1)) are scatter plots of embeddings projected onto two-dimensional space using t-SNE. Points corresponding to equivalent expressions are shown in the same color. The bars in the box represent 25%, 50% and 75% quartile values. The line intervals denote the 1.5 interquartile range (IQR) beyond the quartile values. The dots represent the extreme values. As shown, there is a significant difference between intra- and inter-subset distances. Except for the first subset, the quartile intervals are well separated.

Figure 6.6(b-1) and (b-2) show the same plots on the No-Embed-Loss model, *i.e.* the

HISS variant that is trained without the embedding similarity loss. As can be seen, the scatter points are apparently less well-clustered, and the difference between inter- and intra-subset distances, although still exists, is smaller. Therefore, we can draw two conclusions. First, even without the embedding similarity loss, HISS is still able to learn embeddings that are somewhat clustered according to equivalence, which shows the goal of finding equivalent expressions roughly aligns with the need to cluster the embedding based on the expression equivalence. Second, the embedding similarity loss can improve embedding clustering.

However, what we are more interested in is why an improved clustering of the embeddings would lead to a significant performance gain. To answer this question, figure 6.7 compares the average reward as a function of training epoch of HISS and its variant without the embedding similarity loss, which gives us some very interesting insights. Notice that despite their initial difference, which is due to the random initialization, both reward curves reach the same plateau at around epoch seven, where they both become stagnant for a while. However, with the help of embedding similarity loss, HISS is finally able to escape from the plateau and reach a higher reward level, whereas the one without the embedding similarity loss gets trapped in the plateau. This result suggests that the embedding similarity loss provides an extra training signal to address the convergence issue of REINFORCE.

6.5.11 Subtree Selector

We have already demonstrated the inner-workings of the subtree selector in section 6.5.4 and shown in figure 6.5 that subtree selector is indispensable for the good performance of HISS. Here we would like to intuitively explain why this is the case. Table 6.6 compares the simplification traces of HISS with and without the subtree selector on the following expression

$$((v1+v2)-7)\leq(((((\max(v1,16)+18)/8)*8)+(v1+v2))-27) \tag{6.11}$$

As can be seen, HISS with the subtree selector can first offset the '*8' and '/8' terms

Table 6.6. Simplification traces of HISS with and without the subtree selector on Equation (6.11). The subexpressions selected by the subtree selector are boxed, unless the entire expression is chosen.

Step	HISS with the Subtree Selector
0	((v1+v2)-7)≤(((((max(v1,16)+18)/8)*8)+(v1+v2))-27)
1	((v1+v2)-7)≤(((max(v1,16)+18)+(v1+v2))-27)
2	(v1+v2)≤(((max(v1,16)+18)+(v1+v2))-20)
Step	**HISS without the Subtree Selector**
0	((v1+v2)-7)≤(((((max(v1,16)+18)/8)*8)+(v1+v2))-27)
1	(v1+v2)≤(((((max(v1,16)+18)/8)*8)+(v1+v2))-20)

Table 6.7. Output sequences of Linear-LSTM trained on Traverse Equivalence dataset

Linear-LSTM with only equivalence reward	
Input	**Output**
max(v1 + v0, v2 + v0)	[',', 'c8', '2', '2', '1', 'v9', ',', 'c8', '2', '2']
min(min(v2, v0 + v1), v0 + v3)	[',', 'c9', 'c9', 'c9', 'c9', '2', '2', '1', ',', 'c9']
(v0 &&(v2||v1))||v1	[',', 'c8', '2', '2', '1', ',', 'c9', 'c9', 'c9', '2']
Linear-LSTM trained with equivalence and valid expression reward	
Input	**Output**
min(v2, v0 + v1) - v1	['v1']
v1||(v0||v1)	['c0']
v0 - min(v2, v0 + v1)	['2', '≥', 'v0']

in the subexpression ((max(v1,16)+18)/8)*8), before it applies the cancellation rule to merge the constants '-7' and '-27'. On the other hand, HISS without the subtree selector is unable to identify the reducible subexpression, and so it only applies the cancellation rule. This result shows that the reason why the subtree selector helps is that it can thoroughly check on each subexpression, and therefore contributes to a better simplification. Without the subtree selector, the algorithm is prone to overlook some small subexpressions that are reducible.

6.5.12 Tree LSTM v.s. Linear LSTM

We can see the conspicuous disadvantage of the Linear-LSTM model from figure 6.5. To illustrate the fundamental issue with the Linear-LSTM model, we sampled some output

of the model trained on the Traverse Equivalence dataset, listed in table 6.7. The model was trained with REINFORCE for 200k steps, with two different reward settings: 1) the equivalence reward as in Equation 6.8; 2) the equivalence reward plus a valid expression reward, which is an additional reward of 0.1 if the decoded sequence is syntactically correct.

As can be seen in table 6.7, when trained with only the equivalence reward, the linear decoder is unable to decode even a valid expression, not to mention generating an equivalent expression to the input. So the linear decoder could hardly converge and learn useful information from reward, which was almost always a negative constant. As a remedy, we can add a valid expression reward to guide the linear LSTM to generate valid expressions. However, as can be observed in table 6.7, under this additional reward, the model would overfit this reward by only generating short valid expressions, which are not equivalent to the input. This is because short expressions contain only one variable or constant have a higher generation probability than longer valid expressions. During the training, whenever the model happens to generate a single expression, it was rewarded a positive valid expression reward. So the behavior of generating only a single constant or variable is strengthened. Still, the probability that a random variable was equivalent to the input expression, is small, and therefore the model can hardly be guided by equivalence reward, but focuses only on generating short but valid expressions.

This experiment demonstrates the advantage of using the tree LSTM, which is guaranteed to generate syntactically correct, and which has a much higher probability of hitting an equivalent expression. Therefore, the tree LSTM can be much better guided by the equivalence reward.

6.6 Conclusions

We presented HISS as a symbolic expression simplification algorithm that is independent of human knowledge. We demonstrated that removing the dependence on humans is advantageous for this task because machines can autonomously figure out rewrite rules that humans fail to discover, and thus achieve comparably well simplification results. We also showed that

we are one step closer to finding an equivalence-preserving embedding for symbolic expressions. Although HISS has achieved promising results, there is still much room for improvement. Although HISS has adopted several techniques to reduce the complexity of the search space, learning simplification rules on very long expressions is still challenging, which calls for the exploration of more efficient reinforcement learning algorithms as a future research direction.

6.7 Acknowledgement

This chapter, in full, are a reprint of the material as it appears in "Deep symbolic superoptimization without human knowledge." Shi, Hui, Yang Zhang, Xinyun Chen, Yuandong Tian, and Jishen Zhao. In International Conference on Learning Representations. 2019. The dissertation author was the primary investigator and author of this paper.

Chapter 7

Training and Searching

Symbolic neural methods usually require heightened cautiousness in training and cooperation with specialized searching methods. In Section 2.1, several neural symbolic paradigms require direct interaction with symbolic systems. The interaction has two direct impacts: 1) the end-to-end supervised training method might be infeasible because many symbolic systems are non-differentiable, and either the intermediate labels are required to supervise the actions or the reinforcement learning is used; and 2) consequentially, the searching method is usually used to obtain the intermediate labels semi- or fully automatically, and to sample multiple actions in the reinforcement learning. In this chapter, the training methods are compared to illustrate the common techniques in handling different types of neural symbolic methods according to the label availability. Then the searching algorithm to aid the training of HISS is detailed.

7.1 Training Paradigms

The training of a neural symbolic AI is nontrivial. In the Chapter 3, two approaches to improve the models' performance on understanding the structured input are identified: 1) supervise the model with intermediate labels; 2) control the model complexity, *e.g.* by modularizing the neural networks and parameter sharing. Previous chapters focused on the design of neural architectures: CCNN learns the collapsed interpolation and convolution kernel from the non-uniform time series, and its output defines a parameterized temporal point process, which

later predicts the expected time interval to the next event; MIP learns the users multiple interest vectors and their preference weight by two neural modules and a clustering algorithm; HISS simplifies the mathematical expressions progressively with an encoder, a sub-expression selector, and a decoder that simplifies the sub-expression. The training techniques are equally crucial as the neural architecture design in fulfilling these tasks. Depending on the neural symbolic interaction approaches and label availability, the training can have the following conditions:

- When the label to the neural network output presents, the training is conducted in supervised learning manner.
- In **symbolic neuro symbolic** and **neuro:symbolic→neuro** approaches, usually the label is available, and the training is supervised learning.
- With **neuro|symbolic** approaches, where the label is the output of the symbolic system, it's a **weakly** supervised problem. If the symbolic system is differentiable, the neural network can be trained end-to-end by supervised learning. However, usually the symbolic system is non-differentiable. In simple tasks where the input action space to symbolic systems is small, directly training with reinforcement learning is feasible. Otherwise, the randomly sampling can not efficiently discover any plausible sample from the action space, and consequently the reinforcement learning progress would be protracted. A common alleviation is to divide the action space and to supervise with heuristic loss functions. For example, in the table question answering task, Herzig et al. [61] distinguishes two different types of answers, and proposed heuristic loss functions to supervise the selection of the table columns.
- With **neuro|symbolic** approaches, where the label is the output of the neural network and the input to the symbolic system, it's a **strongly** supervised problem, and its training is the same as in the symbolic neuro symbolic approaches.
- In **neuro[symbolic]** approach, the neural network calls the symbolic system with its

intermediate output, and receives the output of symbolic system for further computation. In some scenarios, the neural network's output to the symbolic system can be supervised by reinforcement learning, *e.g.* discrete actions. The training problem becomes a multi-step reinforcement learning. In other cases, the interaction between neural network and symbolic systems is hard to sample, *e.g.* in MIP the neural network invokes a clustering algorithm with its output vectors as input to the clustering algorithm, then one way to training is to substitutes the symbolic system with a differentiable alternative in the initial stage of the training.

- Similarly, in the **symbolic[neuro]** approach, the problem is multi-step reinforcement learning in general, owing to the interactive nature. The advantage comparing to the neuro[symbolic] approach is that since symbolic system invokes neural network, the intermediate steps are more interpretable, and designing reward function for the intermediate status can greatly alleviate the gradient sparsity problem in the multi-step reinforcement learning.

Meanwhile, when label of the actions are absent and the action space of the neural-symbolic interaction is searchable, there's always a design choice. One option is to search the action space and obtain a set of noisy labels and then to train with supervised learning. Another option is to randomly sampling from the action space, according to the probabilistic distribution that is defined by the neural network output, and train with reinforcement learning. There is also a hybrid approach that whenever the sampling finds a plausible action, the action is buffered and replayed in the future training process.

In the following of this section, the training methods of CCNN, MIP and HISS are compared and explained.

7.1.1 CCNN with Temporal Point Process: Differentiable Symbolic Front-End

CCNN exhibits two neural symbolic design: the auto-regression setting leverages a *symbolic neuro symbolic* approach, and the event interval prediction is *neural|symbolic*. Discussed above, the symbolic system is usually non-differentiable and thus the reinforcement learning is used. Fortunately, the generic temporal point process is defined fully differentiable. In Equation 5.12, the probability density function of historical embedding h_{i-1}, time passed $t-t_{i-1}$, and learnable parameters $\boldsymbol{v}, w, b$ is given by:

$$f^*(t) = \exp\Bigg(\boldsymbol{v}\boldsymbol{h}_{i-1} + w(t-t_{i-1}) + b + \frac{1}{w}\Big(\exp(\boldsymbol{v}\boldsymbol{h}_{j-1}+b) - \exp(\boldsymbol{v}\boldsymbol{h}_{j-1} + w(t-t_{j-1}) + b)\Big)\Bigg). \tag{7.1}$$

In inference, the interval to the next event is predicted by computing the expectation of $f^*(t)$, $\mathbb{E}_{t-t_{i-1}}(f^*(t-t_{i-t}))$ numerically. Since the expectation computation would block the gradients if CCNN is trained by mean square error loss of the computed expectation and the actual intervals, in the training, the objective is to maximizing the probability density of the true interval:

$$\mathscr{L}_{tpp} = -log(f^*(t_i)) \tag{7.2}$$

Then the gradients of the loss can be back propagated to the CCNN, and the training is end-to-end supervised learning. Figure 7.1 summarized the training and inference mode of CCNN with TPP front-end.

7.1.2 MIP with Clustering Algorithm: Two-Stage Training

The MIP architecture leverages a clustering algorithm to 1) select the multiple interest embeddings from the output of the self-attention module by cluster assignment and the temporal

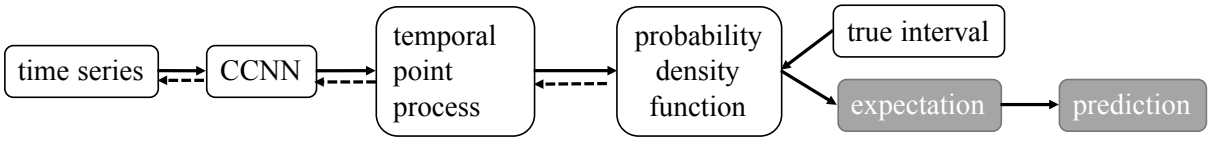


Figure 7.1. CCNN with temporal point process front-end. Dashed arrows show the gradient back propagation. Gray blocks shows the inference mode.

recentness; and 2) select the input to the cluster weight module. Despite the clustering algorithm is non-differentiable, it doesn't blocks the gradient back propagation since its output acts only as the index to select and the output of index selection is the output of neural networks. Therefore, the MIP can be trained end-to-end from scratch.

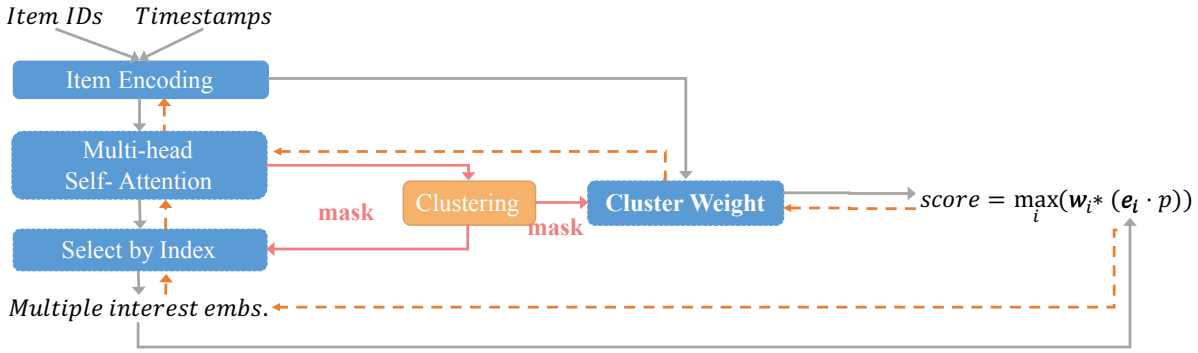
The training from scratch faces convergence difficulty with the meta data is absent in the dataset. The rationale of the MIP architecture is that the multi-interest user embedding modules (item encoding and multi-head self-attention modules) can learn the cluster representation, then the clustering algorithm selects the most recent cluster representations, finally the cluster weight is computed based on the cluster vector and the temporal pattern inside this cluster. Meanwhile, the clustering algorithm groups vectors based on the spatial affinity. Imagine at the early stage of the training, the item encoding are randomly initialized, and the multi-head self-attention outputs are also less meaningful. Consequentially, the clustering assignment is highly noisy and the clustering weight module might generate meaningless random weights to the interests. It's reasonable to assume that the cluster weight module may start to converge after the multi-interest user embedding reaches a plateau, otherwise it's input is always highly noisy. The different convergence speed causes the overall convergence hardship. Intuitively, when the loss encourages higher score for a positive item, it encourages either higher weights or higher similarity of the interest embedding and the target item embedding. The noisy weights can distract the training of interest embeddings.

Therefore, a two-stage training strategy is adopted to enhance the model performance. In the first stage, the cluster weights are fixed to be 1, and only the multi-interest user embedding module is trained. After a few epochs, when the item encoding tends to converge, the cluster

weight module is unfrozen and all parameters are trained until convergence.

Table 7.1 summarizes the model performance of two-stage training vs. joint training (end-to-end training from scratch), and it's clearly seen that this strategy is working really well in practice. This maneuver in the MIP training potentially can benefit the implementation of similar models.

(a) Training stage 1: cluster algorithm and cluster weight module is disabled.



(b) Training stage 2: full pipeline is trained.

Figure 7.2. Simplified MIP architecture and training strategy when meta data is absent. Dotted lines show the gradients flow.

7.1.3 HISS with Symbolic Equivalence Checker: Curriculum Training

The proposed HISS is trained fully without label, *i.e.*, given the input expression, the model knows neither the ultimate simplified equivalent form of the expression, nor the intermediate steps. Even at the early phase of the training, the model barely sees any equivalent expression

Table 7.1. Comparison of training strategy and the performance of MIP. The columns *Epoch* shows the training epochs when the best validation AUC is achieved.

MIP Model	Amazon			Taobao			MovieLens		
	Epoch	AUC	recall@50	Epoch	AUC	recall@50	Epoch	AUC	recall@50
First stage	22	0.731	0.667	4	0.821	0.749	34	**0.960**	0.901
Second stage	6	**0.806**	**0.789**	4	**0.885**	**0.884**	6	0.930	0.933
Joint training	18	0.576	0.570	28	0.803	0.802	14	0.924	**0.937**

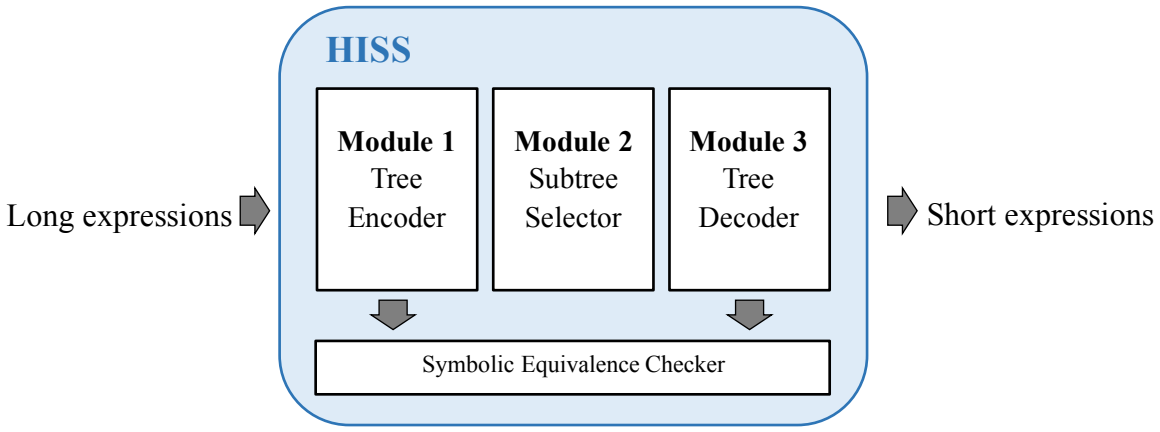


Figure 7.3. Overview of HISS framework.

pairs. Therefore, the training of the HISS is extremely hard and might take plenty machine hours to converge.

Similar to MIP, HISS has multiple cascaded component that one's output is another components' input, and training them simultaneously could be in vain. However, in the Halide dataset, the input expressions are long and unlabeled, it's unreasonable to either disable the subtree selector or the tree decoder: if the subtree selector is removed, it's hard for decoder to find a long equivalent form owing to the gigantic space, so it will only guess the short answers without knowing the mechanism and fails to generalize; if tree decoder is removed, to check if there's an equivalent shorter expression of the selected subtree, the searching space is vast as well. Consequently, an elementary set of training data is need to *pretrain* the HISS model.

The elementary dataset is created to train a simplified pipeline without subtree selector, therefore the input in the dataset should be short enough to be directly simplified by the tree decoder. Given the Halide IR grammar in Table 7.2, all the short expressions (tree size less than

4) are generated. Then the short expressions are examined by whether they have a shorter form equivalence, if so, both the expression and its simplest equivalence are paired and recorded in the elementary dataset. In process of searching the expression is detailed in the Section 7.2.

The result dataset is used to train the tree encoder and tree decoder end-to-end in a supervised manner. After the tree encoder and decoder are trained and their parameters stabilized, the subtree selector is activated and trained together on the Halide dataset. On the second phase, since the Halide dataset has long expressions with no intermediate label, the training is standard REINFORCE. Recall from Equation 6.8, the reward of the reinforcement learning is given by:

$$R(\mathscr{T}_I, \mathscr{T}_O) = \gamma^{\text{card}(\mathscr{T}_O)} \text{ if } \mathscr{T}_I \equiv \mathscr{T}_O, \quad -\beta\gamma^{\text{card}(\mathscr{T}_O)} \text{ otherwise,} \tag{7.3}$$

Note that to receive a positive reward, the decoder generated expression must be tested to be equivalent to the selected subtree. At the same time, the subtree selector need to pick up a redundant subtree, which also need to be small enough for decoder to simplify, and the decoder has to generate an equivalent one. It's totally possible that the subtree selector finds a qualified one, but the decoder fails to generate the equivalent, and consequently, both of them receives negative reward. To avoid this noise, the beam search is applied on the decoding phase and as long as one of the top-K results is tested equivalence, the positive reward is given.

Table 7.2. The Halide intermediate representation (IR) domain specific language (DSL). The operations of tensors are not included.

Unary Operators U	:=	!, -
Binary Operators B	:=	min, max, $\geq$, $\leq$, $<$, $>$, ==, $\neq$, +, -, *, /, &&, ||
Ternary Operators T	:=	select
Constants C	:=	0, 1, 2, True, False
Variables V	:=	v0, v1, ..., v14
Term S	:=	E | C | V
Expression E	:=	(U S) | (B S1 S2) | (T S1 S2 S3)

7.2 Searching.

Searching are leveraged in order to help with the cold start challenge: the model has never seen a equivalent pair of expression, so it hardly possible to generate an equivalence and receives a reward.

Table 6.1 shows the vocabulary of Halide syntax, which is used to construct the traverse equivalence dataset. As there are 14 binary operators, 1 unary operator, and 1 ternary operator, the total number of different expressions of depth two would be over 50k, and the number of different expressions of depth three is over 1.25e+14. However, the rule of thumb is that the simplification is most possible when there are some variables repeatedly appear in the expression, for example, $(x+y)-x \mapsto y$ while $(x+z)+y$ could not be simplified. Therefore, we constraint the enumeration of expression to only 16 operators, 5 constants, and the first three variables (v0, v1, v2). As a result, roughly 3 billion expressions are enumerated.

To check whether an expression could be simplified in this plethora of possible candidates, we adopted the FingerPrint method [71]. The idea is that equivalent expressions should produce the same output under the same set of input. Initially, n sets of random value assignment to all variables are generated, and the fingerprint of an expression is defined as the tuple of the corresponding n output given the assignment. According to the fingerprint, expressions with exactly n ($n = 4$ in our case) same results are grouped into a shard. It can be implied that any equivalent expressions must be in the same shard. Thus, the shards could be processed in parallel. It is highly possible that the initial assignment would lead to an extremely large shard. Therefore, when processing each shard, we repetitively compute new fingerprints until the shard breaks into small shards containing fewer than 5,000 expressions. Then, we use probabilistic testing on each pair of expressions from the same shard to determine whether the expression pair is equivalent. For each expression pair that is equivalent, the longer expression is labeled as reducible. After all the pairs are tested, all the expressions that are labeled as reducible are retained, and the rest are removed.

Further, it is obvious that if an expression could be simplified, the longer expression containing it could also be simplified. So if an expression contains a subexpression that could be simplified, then this expression does not represent the minimal sequence of this simplification and is removed from the dataset as well. After the FingerPrint method and the minimality checking, we obtain approximately 20,000 equivalent expression pairs, from which 1,500 samples are randomly sampled and further split into training, testing, and validation set with a ratio of 6:2:2.

7.3 Acknowledgement

This chapter, in part, is a reprint of material as it appears in "Deep symbolic superoptimization without human knowledge." Shi, Hui, Yang Zhang, Xinyun Chen, Yuandong Tian, and Jishen Zhao. In International Conference on Learning Representations. 2019 and "Everyone's Preference Changes Differently: Weighted Multi-Interest Retrieval Model." Shi, Hui, Yupeng Gu, Yitong Zhou, Bo Zhao, Sicun Gao, and Jishen Zhao. In International Conference on Machine Learning. PMLR, 2023. The dissertation author was the primary investigator and author of this paper.

Chapter 8

Integration

Chapter 7 shows that the interaction pattern between neural networks and symbolic system, together with whether the symbolic system is differentiable, determines the training method of the neural symbolic system. Similarly, the interaction pattern also directly translate to the integration paradigm of the neural neural network in the system.

8.1 Single-Interaction System

In the neural symbolic systems presented, most of them have a clear interface with the system: CCNN predicts the parameters to the temporal point process model, which further forecasts the time interval to the future event by computing the expectation. In MIP, the neural network interacts with the clustering method by feeding the hidden states and receives the cluster assignment, on the other hand the final output of MIP will be sent to efficient adjacency-based retriever to generate the candidate items.

8.2 Multi-Interaction System: HISS

In HISS , the entire pipeline requires multi-point interactions between neural network and symbolic systems. Figure 8.1 illustrate the four-tier simplification process. On the top-level a iteration controller decides the number of iterations needed. In each iteration, the direct layer is the expression transformation utilities that prepares the expression or sub-expressions in the

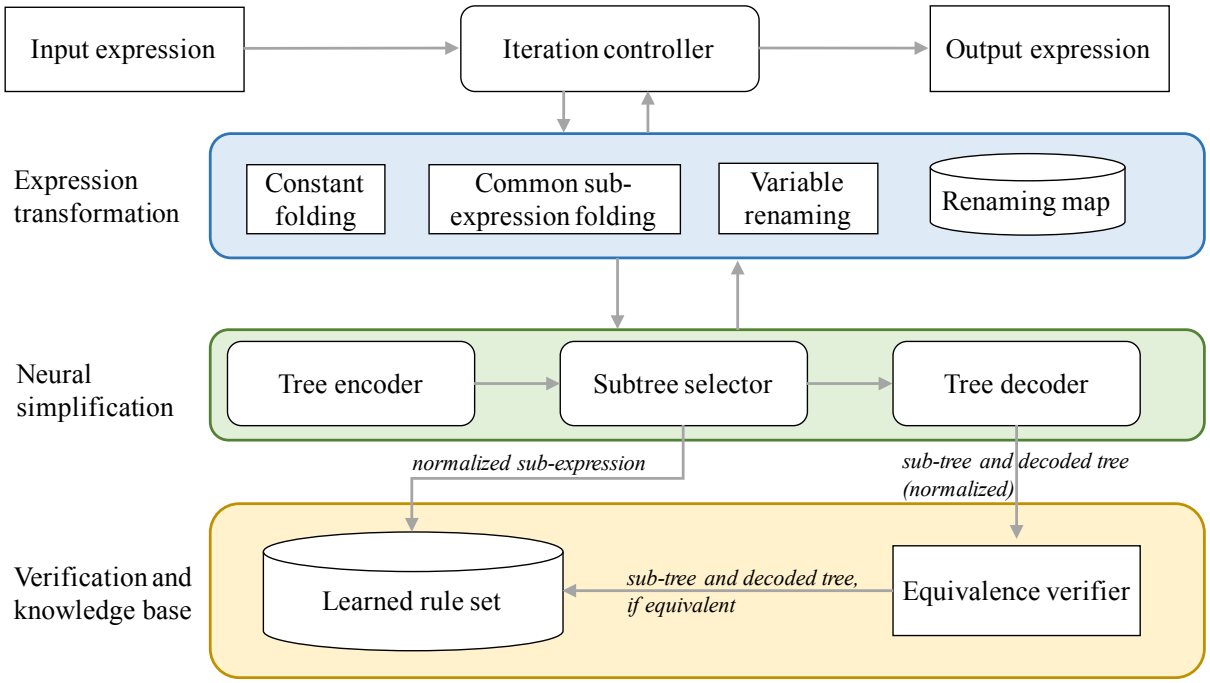


Figure 8.1. HISS system in expression simplification.

format that is easy for neural network to encode and decode and for learned rule set to directly apply. Below lies the neural network layer consist of tree encoder, sub-tree selector, and tree decoder. The most back-end is the learned rule set and the equivalence verifier.

8.2.1 Iteration Controller

In the categorization of Figure 2.1, HISS is a symbolic[neuro] method, where the high level control is the symbolic system, and the neural network is sub-routine in the entire system. On the top level, the following rules are controlling the termination of the process: 1) expression is reduced to a single variable or constant; 2) subtree selector assigns low scores to all subtrees; 3) maximum number of iteration reached. Inside the iterations, a few symbolic methods are applied interleaving with the tree encoder, sub-tree selector and the tree decoder.

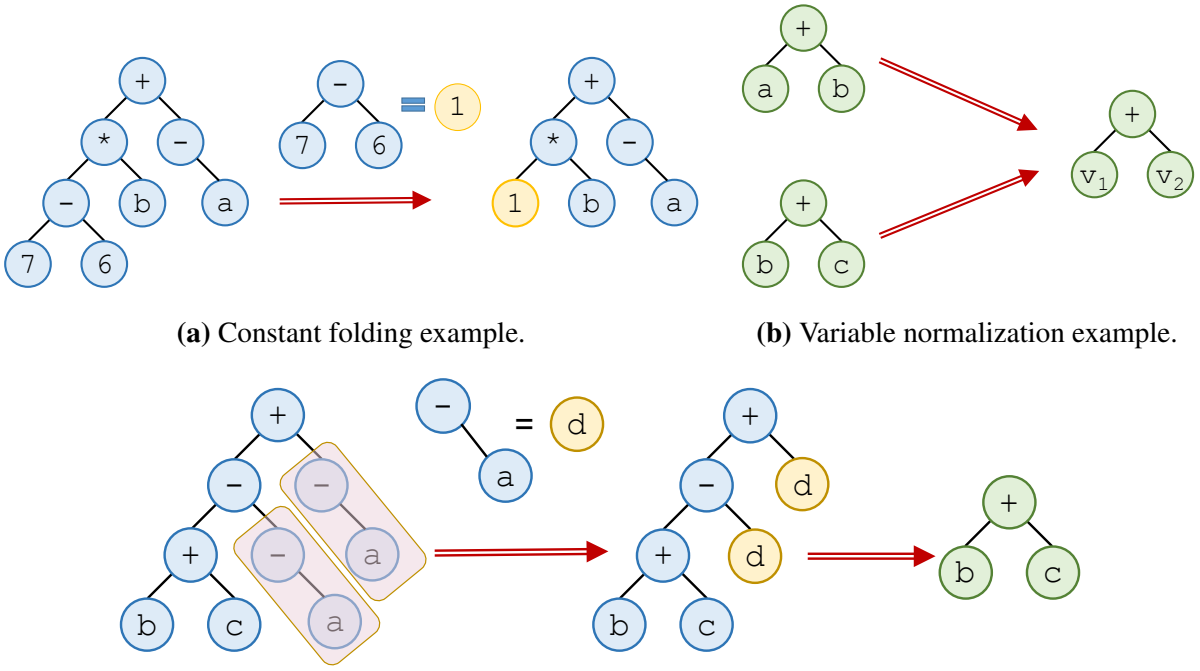
8.2.2 Expression Transformation Layer

Variable Normalization is the process of re-naming the variables and constants in each expression. The normalized names use the cardinal order of the occurrence of each variable

and constants as the subscript of the variable and constant identifier. A look-up table is built to recover the original variable names and constant values after the decoding phase. In each iteration, the variable normalization happens twice: on the entire tree before feeding into the tree encoder, and on the subtree selected. The rationale is that both tree encoder and decoder might be confused with various variable names, since the variable names are just meaningless symbols. Note that the special constants are not normalized, *i.e.*, 0, 1, True, and False, because these constants usually cause the subtree containing them being reducible.

Constant Folding Since the Halide dataset contains a large number of constant values, we apply a technique called constant folding to both stage-two training and inference of HISS as well as to all other baselines. Specifically, once the expression is rewritten by the neural network in the symbolic domain, it will be checked if all the leaf nodes in the subtree are constant. If so, the expression is executed and replaced by a new single node with the execution result. Constant folding is applied in both training and inference.

Common Sub-expression Folding is a technique that examines if there are multiple occurrence of the same sub-expression in the entire expression, if there are, replace the sub-expression with an identifier. When common sub-expression presents, the iteration attempts thrice: 1) simplify the expression after common sub-expression folding; 2) simplify the common sub-expression; 3) if the previous two attempts fails, simplify the original expression. For instance, in the example in Table 6.3, the sub-expression $(v0*72)$ in $(((((144-(v0*72))/2)*2)+4) \leq ((150-(v0*72))/4)*4)$ can be replaced with a single identifier to reduce the tree depth by 1, which is non-trivially easier for the tree encoder and tree decoder. In another case, for example, if the original expression is $((144-((((v0*72)/2)*2)+4)) \leq (150-((((v0*72)/2)*2))))$ and the common sub-expression $(((v0*72)/2)*2)$ can be simplified to $(v0*72)$, thus all the common sub-expression will be replace in the one run. In the worst case, the reducible part of the original expression can not be handled in the previous two cases, *e.g.* $((144-((((v0*72)/2)*2)+4)) \leq (150-((((v0*72)/2)*5))))$, if the common sub-expression folding is performed, the new expression $((144-((E0*2)+4)) \leq (150-(E0*5)))$, though



(a) Constant folding example.

(b) Variable normalization example.

(c) Common sub-expression elimination example.

Figure 8.2. Expression transformation techniques.

still reducible, but the $E0*2$ can not be reduced.

8.2.3 Verification and Knowledge Base

Knowledge base A knowledge base, *i.e.* set of transformation rules, is built along the training and online process. Owing to the searching cost of the decoder and the testing cost for verifying the equivalent rewrite rules, once a equivalent pair is found, it's archived in the rule set. The rule set serves as a shortcut for the tree decoder: the subtree selected sub-expression is first tried to be reduced by the rule set, if there's no existing rule, the tree decoder will explore the new rules.

Equivalence verifier The equivalence of two expressions is examined by probabilistic test as proposed in [104]. Each test has 1000 iterations in which the all variables in two expressions are instantiated with values. In which variable value assignment, the same variables in the two expression receives same value. An iteration is passed if the outcomes from two expressions are the same, considering the computation tolerance of $10^{-}7$ if both results have

float data type. A failed iteration ceases the test and report inequivalent result. The test starts with common values, *e.g.*, [-1, 1, 0, True, False], for variable assignment, and after the enumeration of common value combinations, the assigned values are sampled from normal distribution $\mathcal{N}(0, 1024)$ if the data type is non-Boolean, and sampled from [True, False] with fifty-fifty chance for Boolean variables.

8.2.4 Conclusion

In conclusion, the integration with the existing compiler functionalities is pivotal for HISS to handle complicated long expressions.

8.3 Acknowledgement

This chapter, in part, are a reprint of the material as it appears in "Deep symbolic superoptimization without human knowledge." Shi, Hui, Yang Zhang, Xinyun Chen, Yuandong Tian, and Jishen Zhao. In International Conference on Learning Representations. 2019. The dissertation author was the primary investigator and author of this paper.

Chapter 9

Conclusions

9.1 Summary

In summary, motivated by the failures that neural networks by themselves could not well generalize, reason, and handle the tasks with complicated input and output structure, and the long winded reasoning process, neural symbolic approaches have emerged as a promising avenue for addressing these shortcomings. In this dissertation, an empirical study confirms and analyzes the inadequacy of popular sequential models in learning the structure of context-free languages in the realistic setting, despite that performance of LSTM model can be greatly improved to the same level as the Transformer networks given additional regularization. Building upon the analysis, a few neural symbolic applications are presented to illustrate the common techniques in designing neural architectures, training the models, and integrating to the system, to reflect the symbolic nature of each application. With the applications, this dissertation showcases the diverse paradigms of neural symbolic approaches and demonstrates their effectiveness through these successful instantiations.

The paradigms exemplified in each applications are as follows:

- In CCNN under the auto-regression setting, the symbolic transformation of an interpolation and convolution is performed inherently by the neural network. Therefore, it demonstrates the **symbolic neuro symbolic** paradigm.

- When connecting CCNN with a parameterized temporal point process, the output of the neural network serves as the input to the exterior symbolic system. In the inference phase, the entire pipeline generates the time interval prediction by taking expectation from the probability density function jointly defined by the temporal point process and the neural network output, other than directly from neural network. Thus, the overall paradigm is a typical **neuro|symbolic** approach.

- The MIP architecture showcases a simple **neural[symbolic]** paradigm where the neural network generates the final outputs, the multiple user interest vectors, while in the middle of the neural computations, there are some interaction to a non-neuro process of a clustering algorithm. In this paradigm, it's crucial to ensure that neural modules before the non-neuro components could receive the gradients.

- The first stage in the curriculum training of HISS follows the **neuro:symbolic** $\rightarrow$ **neuro** paradigm. In the data preparation, the enumerator generates the expressions. Then, combined with fingerprint grouping strategy, the symbolic equivalent tester efficiently identifies the pairs of equivalent expressions. The expression pairs are used to pre-train the HISS .

- The full HISS functionality is a **symbolic[neuro]** approach. The high-level control is a symbolic program that dynamically activates distinct neural modules based on the specific scenarios of the simplification process.

Numerous theoretical and empirical evidences have revealed the inherent limitation in the representation power of neural networks owing to the architecture and the optimization methods. When the straightforward **symbolic neural symbolic** approaches reach their performance ceiling, introducing a symbolic system into the entire pipeline emerges as an efficient and effective way, besides scaling up neural networks and increasing the amount of training data. Simultaneously, specialized neural architectures may be necessary to reflect the symbolic nature while maintaining

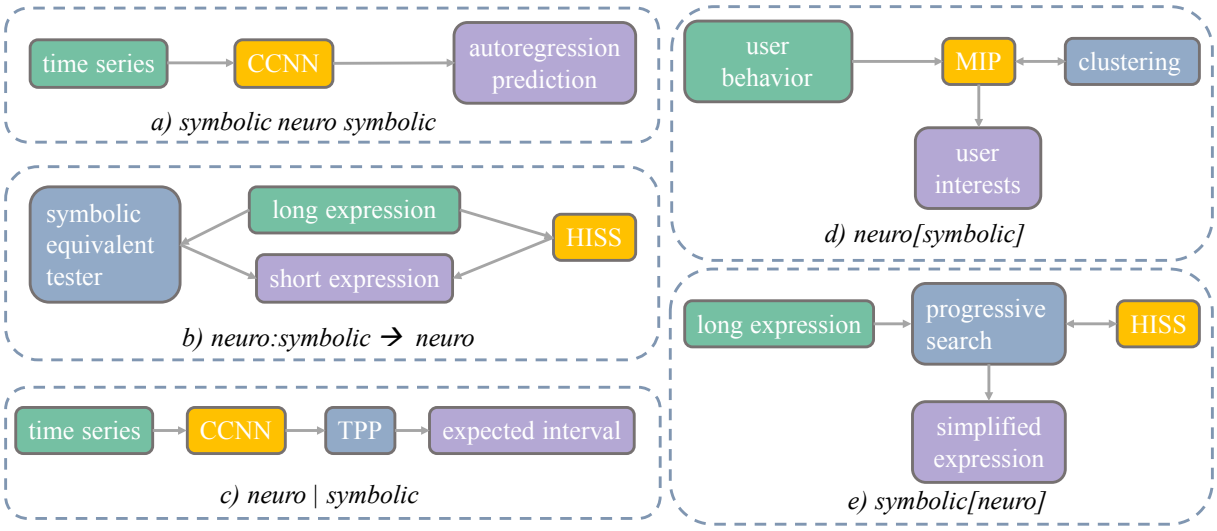


Figure 9.1. Neural symbolic AI paradigms instantiated in CCNN, MIP, and HISS models.

the generalization capabilities. With both symbolic system and neural network, the training of the neural network should be strategic. This dissertation also demonstrated training techniques when the symbolic system is non-differentiable, including sub-module pre-training (*i.e.* two-stage training) and the utilization of reinforcement learning methods.

9.2 Future Works

In this dissertation, CCNN delves into the fundamental problem of the time series analysis, demonstrating the versatility of its methodology and potential applications in diverse domains. The CCNN framework can be effectively applied to other tasks. For example, analyzing social media data to identify trending topics and the dissemination patterns across the internet, and detect abnormal activities in user-facing systems to enhance cybersecurity. The CCNN's capability to capture arbitrary dynamic temporal patterns in the time series makes it well-suited for those time sensitive applications.

Expanding on the success of MIP in multi-interest user representation and preference modeling, there are many future directions to work including improvement of the model architecture and integration with other platform-specific requirements. In the direction of improving

model architecture, current preference model, a simple feed-forward network, can be replaced with nuanced sequential models to increase the performance. As preference is predicted for a specific future time, *e.g.* assuming the user will visit the system in the next week, and the preference could be sensitive to the future time window, the preference scores shall be refreshed every a period of time. To reduce this computation cost, the preference model could potentially leverage a generic temporal point process to isolate the prediction of TPP parameters with the prediction based on future time. The parameter-prediction is an one-time computation by the relatively costly neural network, unless the user's multi-interest embeddings are updated. Then with TPP, the preference prediction in future time can be refreshed periodically at trivial computation cost. In the application side, the preference scores are beneficial for a variety purpose. The scores suggest portions of each categories when the system recommend a batch of items to the user. In addition, the score can be post-engineered to combine with promotional weights for items or categories.

HISS overcomes several prevalent challenges in code understanding and generation, and it addresses the challenges of a combinatorial optimization problem. Its ability to efficiently understand the code and decompose the code optimization problem to solve progressively holds immense potential for general computation optimizations. A promising future work in its direction lies in extending HISS 's capacities to optimize general tensor graph computations, which is fundamental to many machine learning training and serving workloads. HISS also offers promising application in database query optimization. Crafting optimal query requires thorough understanding of query execution model, and therefore is challenging to inexperienced users. HISS , requiring no human supervision and expert knowledge, could learn to transform the queries to reduce the computation and storage cost and alleviate the problem of inefficient queries.

Within the broader landscape of neural symbolic system, a preeminent direction is harnessing the power of large language models (LLMs) to establish the paradigm of LLM-enabled neural symbolic system. Language models are standalone neural system and themselves

follows **symbolic neuro symbolic** paradigm, though their performance in symbolic reasoning are still compromised despite the gigantic model size and data size. However, the success of LLM provokes widespread interest to enhance the LLM with symbolic tools. LLMs have undoubtedly emerged as the most captivating and promising area of research owing to their advantages for the neural symbolic applications. Pre-trained LLMs have broad prior knowledge in natural languages, common sense in understanding human-readable documents, programming ability in a wide spectrum of languages, and zero-shot learning capability in new tasks. Moreover, despite the accuracy of LLM in reasoning tasks might yet reach human-level accuracy, its improved performance greatly reduces the searching effort in the multi-action tasks. In this framework, the LLMs predict intermediate actions and receive result or feedback from a symbolic component, *e.g.* a calculator, a compiler, a database, and iterate based on the output of symbolic component, employing the **neuro[symbolic]** paradigm.

In addition to LLM-powered neuro[symbolic] approaches, the complementary paradigm of symbolic[neuro] is poised to make significant impact in real-world applications. Not only in the specialized domains, like the Go competition and drug discovery, but also in broader areas such as autonomous vehicles, the advancement steadily propels the vehicles to achieve the full automation. The nature of symbolic[neuro] allows the system to operate with transparency and accountability. As we progress towards the future of full automatic driving, ensuring the reliability and safety of such complex systems becomes paramount. More efforts will be needed in verification of the behavior of the entire system, so the reliability can be fully evaluated and supervised by human and the government.

The two direction of neuro[symbolic] and symbolic[neuro] are not mutually exclusive but rather complementary pathways towards achieving comprehensive artificial intelligence. While the LLM-enabled neuro[symbolic] AIs excel in applications involving understanding and interaction with human user in general natural languages, specialized symbolic[neuro] approaches demonstrate remarkable efficiency and adaptability in addressing specific domain-centric tasks. When developing user-facing end-to-end AI solutions, a strategic approach involves

adopting neuro[symbolic] AIs as front-end to triage user needs and direct them to appropriate domain-specific modules. These domain-specific modules can employ their own symbolic[neuro] AIs for optimal performance within their respective areas of expertise. The synergistic fusion of two approaches offers a promising strategy for developing robust, versatile, and efficient AI solutions.

Bibliography

[1] Stackoverflow. https://archive.org/details/stackexchange.

[2] Why ai image generators can't get hands right. https://petapixel.com/2023/03/02/why-ai-image-generators-cant-get-hands-right/, 2023. Accessed: 2023-09-05.

[3] Andrew Adams, Jongmin Baek, and Myers Abraham Davis. Fast high-dimensional filtering using the permutohedral lattice. In *Computer Graphics Forum*, volume 29, pages 753–762. Wiley Online Library, 2010.

[4] Pedro Aguilera Aguilera, Antonio Fernández, Rosa Fernández, Rafael Rumí, and Antonio Salmerón. Bayesian networks in environmental modelling. *Environmental Modelling & Software*, 26(12):1376–1388, 2011.

[5] David Alvarez-Melis and Tommi S Jaakkola. Tree-structured decoding with doubly-recurrent neural networks. 2016.

[6] Forough Arabshahi, Sameer Singh, and Animashree Anandkumar. Combining symbolic expressions and black-box function evaluations in neural programs. *arXiv preprint arXiv:1801.04342*, 2018.

[7] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

[8] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[9] Manoj Kumar Balwant. Bidirectional lstm based on pos tags and cnn architecture for fake news detection. In *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pages 1–6, 2019. doi: 10.1109/ICCCNT45670.2019.8944460.

[10] Dimitri P Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena scientific Belmont, MA, 1995.

[11] Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. On the ability of self-attention networks to recognize counter languages. *arXiv preprint arXiv:2009.11264*, 2020.

[12] Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. On the practical ability of recurrent neural networks to recognize hierarchical languages. *arXiv preprint arXiv:2011.03965*, 2020.

[13] Przemyslaw Bogacki and Lawrence F Shampine. A 3 (2) pair of runge-kutta formulas. *Applied Mathematics Letters*, 2(4):321–325, 1989.

[14] Charles A Bouman and Michael Shapiro. A multiscale random field model for bayesian image segmentation. *IEEE Transactions on image processing*, 3(2):162–177, 1994.

[15] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[16] Sebastian Buchwald. Optgen: A generator for local optimizations. In *International Conference on Compiler Construction*, pages 171–189. Springer, 2015.

[17] Rudy Bunel, Matthew Hausknecht, Jacob Devlin, Rishabh Singh, and Pushmeet Kohli. Leveraging grammar and reinforcement learning for neural program synthesis. *arXiv preprint arXiv:1805.04276*, 2018.

[18] Cheng-Hao Cai, Yanyan Xu, Dengfeng Ke, and Kaile Su. Learning of human-like algebraic reasoning using deep feedforward neural networks. *Biologically inspired cognitive architectures*, 25:43–50, 2018.

[19] Gert Cauwenberghs. An analog VLSI recurrent neural network learning a continuous-time trajectory. *IEEE Transactions on Neural Networks*, 7(2):346–361, 1996.

[20] Yukuo Cen, Jianwei Zhang, Xu Zou, Chang Zhou, Hongxia Yang, and Jie Tang. Controllable multi-interest framework for recommendation. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2942–2951, 2020.

[21] Shiyu Chang, Yang Zhang, Jiliang Tang, Dawei Yin, Yi Chang, Mark A Hasegawa-Johnson, and Thomas S Huang. Positive-unlabeled learning in streaming networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 755–764. ACM, 2016.

[22] Shiyu Chang, Yang Zhang, Wei Han, Mo Yu, Xiaoxiao Guo, Wei Tan, Xiaodong Cui, Michael Witbrock, Mark A Hasegawa-Johnson, and Thomas S Huang. Dilated recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 76–86,

2017.

[23] François Charton, Amaury Hayat, and Guillaume Lample. Learning advanced mathematical computations from examples. *arXiv preprint arXiv:2006.06462*, 2020.

[24] Angelica Chen, Jason Phang, Alicia Parrish, Vishakh Padmakumar, Chen Zhao, Samuel R Bowman, and Kyunghyun Cho. Two failures of self-consistency in the multi-step reasoning of llms. *arXiv preprint arXiv:2305.14279*, 2023.

[25] Liang-Chieh Chen, Yi Yang, Jiang Wang, Wei Xu, and Alan L Yuille. Attention to scale: Scale-aware semantic image segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3640–3649, 2016.

[26] Xinyun Chen and Yuandong Tian. Learning to progressively plan. *arXiv preprint arXiv:1810.00337*, 2018.

[27] Xinyun Chen, Chang Liu, and Dawn Song. Tree-to-tree neural networks for program translation. In *Advances in Neural Information Processing Systems*, pages 2547–2557, 2018.

[28] Xu Chen, Yongfeng Zhang, and Zheng Qin. Dynamic explainable recommendation based on neural attentive models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 53–60, 2019.

[29] Gong Cheng, Junwei Han, Peicheng Zhou, and Dong Xu. Learning rotation-invariant and fisher discriminative convolutional neural networks for object detection. *IEEE Transactions on Image Processing*, 28(1):265–278, 2018.

[30] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*, pages 7–10, 2016.

[31] N Chomsky. Context-free grammars and pushdown storage. vol. 65 of quarterly progress report, 1962.

[32] Noam Chomsky. Three models for the description of language. *IRE Transactions on information theory*, 2(3):113–124, 1956.

[33] Chinmay Choudhary and Colm O'riordan. Multilingual end-to-end dependency parsing with linguistic typology knowledge. In *Proceedings of the 5th Workshop on Research in Computational Linguistic Typology and Multilingual NLP*, pages 12–21, 2023.

[34] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks

for image classification. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3642–3649. IEEE, 2012.

[35] Paul Covington, Jay Adkan, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pages 191–198, 2016.

[36] Eric Crawford and Joelle Pineau. Spatially invariant unsupervised object detection with convolutional neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3412–3420, 2019.

[37] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[38] Sreerupa Das, C Lee Giles, and Guo-Zheng Sun. Learning context-free grammars: Capabilities and limitations of a recurrent neural network with an external stack memory. In *Proceedings of The Fourteenth Annual Conference of Cognitive Science Society. Indiana University*, page 14. Citeseer, 1992.

[39] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[40] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

[41] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[42] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[43] Robin Devooght and Hugues Bersini. Long and short-term recommendations with recurrent neural networks. In *Proceedings of the 25th Conference on User Modeling, Adaptation and Personalization*, pages 13–21, 2017.

[44] Tim Donkers, Benedikt Loepp, and Jürgen Ziegler. Sequential user-based recurrent neural network recommendations. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 152–160, 2017.

[45] Mehdi Drissi, Olivia Watkins, Aditya Khant, Vivaswat Ojha, Pedro Sandoval, Rakia Segev, Eric Weiner, and Robert Keller. Program language translation using a grammar-driven

tree-to-tree model. *arXiv preprint arXiv:1807.01784*, 2018.

[46] Nan Du, Hanjun Dai, Rakshit Trivedi, Utkarsh Upadhyay, Manuel Gomez-Rodriguez, and Le Song. Recurrent marked temporal point processes: Embedding event history to vector. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1555–1564. ACM, 2016.

[47] Alessandro Epasto and Bryan Perozzi. Is a single embedding enough? learning node representations that capture multiple social contexts. In *The World Wide Web Conference*, pages 394–404, 2019.

[48] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996.

[49] Andrea Frome, Greg S Corrado, Jon Shlens, Samy Bengio, Jeff Dean, Marc'Aurelio Ranzato, and Tomas Mikolov. Devise: A deep visual-semantic embedding model. *Advances in neural information processing systems*, 26, 2013.

[50] Ken-ichi Funahashi and Yuichi Nakamura. Approximation of dynamical systems by continuous time recurrent neural networks. *Neural Networks*, 6(6):801–806, 1993.

[51] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A Wichmann, and Wieland Brendel. Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness. *arXiv preprint arXiv:1811.12231*, 2018.

[52] Ramazan Gençay, Michel Dacorogna, Ulrich A Muller, Olivier Pictet, and Richard Olsen. *An Introduction to High-frequency Finance*. Academic press, 2001.

[53] Leon Glass and Michael C Mackey. Pathological conditions resulting from instabilities in physiological control systems. *Annals of the New York Academy of Sciences*, 316(1): 214–235, 1979.

[54] Alex Graves and Jürgen Schmidhuber. Offline handwriting recognition with multidimensional recurrent neural networks. *Advances in neural information processing systems*, 21, 2008.

[55] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[56] Michael Hahn. Theoretical limitations of self-attention in neural sequence models. *Transactions of the Association for Computational Linguistics*, 8:156–171, 2020.

[57] Yiding Hao, William Merrill, Dana Angluin, Robert Frank, Noah Amsel, Andrew Benz,

and Simon Mendelsohn. Context-free transductions with neural stacks. *arXiv preprint arXiv:1809.02836*, 2018.

[58] Alan G Hawkes. Spectra of some self-exciting and mutually exciting point processes. *Biometrika*, 58(1):83–90, 1971.

[59] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[60] Ruining He and Julian McAuley. Fusing similarity models with markov chains for sparse sequential recommendation. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 191–200. IEEE, 2016.

[61] Jonathan Herzig, Paweł Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. Tapas: Weakly supervised table parsing via pre-training. *arXiv preprint arXiv:2004.02349*, 2020.

[62] John Hewitt, Michael Hahn, Surya Ganguli, Percy Liang, and Christopher D Manning. Rnns can generate bounded hierarchical languages with optimal memory. *arXiv preprint arXiv:2010.07515*, 2020.

[63] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*, 2015.

[64] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[65] Steffen Hölldobler, Yvonne Kalinke, and Helko Lehmann. Designing a counter: Another case study of dynamics and activation landscapes in recurrent networks. In *Annual Conference on Artificial Intelligence*, pages 313–324. Springer, 1997.

[66] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

[67] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[68] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[69] Valerie Isham and Mark Westcott. A self-correcting point process. *Stochastic processes*

and their applications, 8(3):335–347, 1979.

[70] Abhinav Jangda and Greta Yorsh. Unbounded superoptimization. In *Proceedings of the 2017 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, pages 78–88. ACM, 2017.

[71] Zhihao Jia, Oded Padon, James Thomas, Todd Warszawski, Matei Zaharia, and Alex Aiken. Taso: optimizing deep learning computation with automatic generation of graph substitutions. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 47–62. ACM, 2019.

[72] Hao Jiang, Wenjie Wang, Yinwei Wei, Zan Gao, Yinglong Wang, and Liqiang Nie. What aspect do you like: Multi-scale time-aware user interest modeling for micro-video recommendation. In *Proceedings of the 28th ACM International Conference on Multimedia*, pages 3487–3495, 2020.

[73] Alistair EW Johnson, Tom J Pollard, Lu Shen, H Lehman Li-wei, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3:160035, 2016.

[74] Justin Johnson, Bharath Hariharan, Laurens Van Der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2901–2910, 2017.

[75] Rajeev Joshi, Greg Nelson, and Keith Randall. *Denali: a goal-directed superoptimizer*, volume 37. ACM, 2002.

[76] Armand Joulin and Tomas Mikolov. Inferring algorithmic patterns with stack-augmented recurrent nets. *arXiv preprint arXiv:1503.01007*, 2015.

[77] Ashwin Kalyan, Abhishek Mohta, Oleksandr Polozov, Dhruv Batra, Prateek Jain, and Sumit Gulwani. Neural-guided deductive search for real-time program synthesis from examples. *arXiv preprint arXiv:1804.01186*, 2018.

[78] Wang-Cheng Kang and Julian McAuley. Self-attentive sequential recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 197–206. IEEE, 2018.

[79] Henry Kautz. The third ai summer: Aaai robert s. engelmore memorial lecture. *AI Magazine*, 43(1):105–125, 2022.

[80] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to

weigh losses for scene geometry and semantics. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7482–7491, 2018.

[81] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. Accurate image super-resolution using very deep convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1646–1654, 2016.

[82] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[83] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[84] Philipp Koehn, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 127–133, 2003.

[85] Samuel A Korsky and Robert C Berwick. On the computational power of rnns. *arXiv preprint arXiv:1906.06349*, 2019.

[86] Jan Koutnik, Klaus Greff, Faustino Gomez, and Juergen Schmidhuber. A clockwork RNN. In *International Conference on Machine Learning*, pages 1863–1871, 2014.

[87] Dexter C Kozen. The chomsky—schützenberger theorem. In *Automata and Computability*, pages 198–200. Springer, 1997.

[88] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[89] Julian Kupiec. Robust part-of-speech tagging using a hidden markov model. *Computer speech & language*, 6(3):225–242, 1992.

[90] Brenden Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International Conference on Machine Learning*, pages 2873–2882. PMLR, 2018.

[91] Guillaume Lample and François Charton. Deep learning for symbolic mathematics. *arXiv preprint arXiv:1912.01412*, 2019.

[92] Chao Li, Zhiyuan Liu, Mengmeng Wu, Yuchi Xu, Huan Zhao, Pipei Huang, Guoliang Kang, Qiwei Chen, Wei Li, and Dik Lun Lee. Multi-interest network with dynamic routing for recommendation at tmall. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 2615–2623, 2019.

[93] Jiacheng Li, Yujie Wang, and Julian McAuley. Time interval aware self-attention for sequential recommendation. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, pages 322–330, 2020.

[94] Kehuang Li and Chin-Hui Lee. A deep neural network approach to speech bandwidth expansion. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 4395–4399. IEEE, 2015.

[95] Shuang Li, Shuai Xiao, Shixiang Zhu, Nan Du, Yao Xie, and Le Song. Learning temporal point processes via reinforcement learning. In *Advances in neural information processing systems*, pages 10781–10791, 2018.

[96] Yu Li, Liu Lu, and Li Xuefeng. A hybrid collaborative filtering method for multiple-interests and multiple-content recommendation in e-commerce. *Expert systems with applications*, 28(1):67–77, 2005.

[97] Pin-Chao Liao, Hui Shi, Yusung Su, and Xintong Luo. Development of data-driven influence model to relate the workplace environment to human error. *Journal of construction engineering and management*, 144(3):04018003, 2018.

[98] Wang Ling, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiskỳ, Andrew Senior, Fumin Wang, and Phil Blunsom. Latent predictor networks for code generation. *arXiv preprint arXiv:1603.06744*, 2016.

[99] Hongbin Liu and Ickjai Lee. End-to-end trajectory transportation mode classification using bi-lstm recurrent neural network. In *2017 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*, pages 1–5. IEEE, 2017.

[100] Edward N Lorenz. Deterministic nonperiodic flow. *Journal of the Atmospheric Sciences*, 20(2):130–141, 1963.

[101] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[102] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[103] Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B Tenenbaum, and Jiajun Wu. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. *arXiv preprint arXiv:1904.12584*, 2019.

[104] Henry Massalin. Superoptimizer: a look at the smallest program. In *ACM SIGARCH Computer Architecture News*, volume 15, pages 122–126. IEEE Computer Society Press, 1987.

[105] Minesh Mathew, Ruben Tito, Dimosthenis Karatzas, R. Manmatha, and C. V. Jawahar. Document visual question answering challenge 2020, 2020.

[106] Hongyuan Mei and Jason M Eisner. The neural hawkes process: A neurally self-modulating multivariate point process. In *Advances in Neural Information Processing Systems*, pages 6754–6764, 2017.

[107] Joshua J Michalenko, Ameesh Shah, Abhinav Verma, Richard G Baraniuk, Swarat Chaudhuri, and Ankit B Patel. Representing formal languages: A comparison between finite automata and recurrent neural networks. *arXiv preprint arXiv:1902.10297*, 2019.

[108] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[109] Daniel Neil, Michael Pfeiffer, and Shih-Chii Liu. Phased LSTM: Accelerating recurrent network training for long or event-based sequences. In *Advances in Neural Information Processing Systems*, pages 3882–3890, 2016.

[110] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[111] Aditya Pal, Chantat Eksombatchai, Yitong Zhou, Bo Zhao, Charles Rosenberg, and Jure Leskovec. Pinnersage: Multi-modal user embedding framework for recommendations at pinterest. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2311–2320, 2020.

[112] Emilio Parisotto, Abdel-rahman Mohamed, Rishabh Singh, Lihong Li, Dengyong Zhou, and Pushmeet Kohli. Neuro-symbolic program synthesis. *arXiv preprint arXiv:1611.01855*, 2016.

[113] Emanuel Parzen. On estimation of a probability density function and mode. *The Annals of Mathematical Statistics*, 33(3):1065–1076, 1962.

[114] Barak A Pearlmutter. Learning state space trajectories in recurrent neural networks. *Learning*, 1(2), 2008.

[115] Jorge Pérez, Javier Marinković, and Pablo Barceló. On the turing completeness of modern neural network architectures. *arXiv preprint arXiv:1901.03429*, 2019.

[116] Massimo Quadrana, Alexandros Karatzoglou, Balázs Hidasi, and Paolo Cremonesi. Personalizing session-based recommendations with hierarchical recurrent neural networks. In *proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 130–137, 2017.

[117] Guillaume Rabusseau, Tianyu Li, and Doina Precup. Connecting weighted automata and recurrent neural networks through spectral learning. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1630–1639. PMLR, 2019.

[118] Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. *Acm Sigplan Notices*, 48(6): 519–530, 2013.

[119] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[120] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 811–820, 2010.

[121] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

[122] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. *arXiv preprint arXiv:1710.09829*, 2017.

[123] Eric Schkufza, Rahul Sharma, and Alex Aiken. Stochastic superoptimization. In *ACM SIGPLAN Notices*, volume 48, pages 305–316. ACM, 2013.

[124] Marcel Paul Schützenberger. On context-free languages and push-down automata. *Information and control*, 6(3):246–264, 1963.

[125] Luzi Sennhauser and Robert C Berwick. Evaluating the ability of lstms to learn context-free grammars. *arXiv preprint arXiv:1811.02611*, 2018.

[126] Oleksandr Shchur, Marin Biloš, and Stephan Günnemann. Intensity-free learning of temporal point processes. *arXiv preprint arXiv:1909.12127*, 2019.

[127] Hui Shi, Yusheng Xie, Luis Goncalves, Sicun Gao, and Jishen Zhao. Wikidt: Visual-based table recognition and question answering dataset.

[128] Hui Shi, Yang Zhang, Hao Wu, Shiyu Chang, Kaizhi Qian, Mark Hasegawa-Johnson, and Jishen Zhao. Continuous cnn for nonuniform time series. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3550–3554. IEEE, 2021.

[129] Hui Shi, Sicun Gao, Yuandong Tian, Xinyun Chen, and Jishen Zhao. Learning bounded context-free-grammar via lstm and the transformer: Difference and the explanations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8267–8276, 2022.

[130] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905, 2000.

[131] Richard Shin, Neel Kant, Kavi Gupta, Christopher Bender, Brandon Trabucco, Rishabh Singh, and Dawn Song. Synthetic datasets for neural program synthesis. *arXiv preprint arXiv:1912.12345*, 2019.

[132] Hava T Siegelmann and Eduardo D Sontag. On the computational power of neural nets. *Journal of computer and system sciences*, 50(1):132–150, 1995.

[133] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[134] Amanpreet Singh, Vivek Natarajan, Meet Shah, Yu Jiang, Xinlei Chen, Dhruv Batra, Devi Parikh, and Marcus Rohrbach. Towards vqa models that can read. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[135] Dr Stogin, Ankur Mali, Dr Giles, et al. Provably stable interpretable encodings of context free grammars in rnns with a differentiable stack. *arXiv preprint arXiv:2006.03651*, 2020.

[136] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. End-to-end memory networks. *arXiv preprint arXiv:1503.08895*, 2015.

[137] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pages 1441–1450, 2019.

[138] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *arXiv preprint arXiv:1409.3215*, 2014.

[139] Mirac Suzgun, Sebastian Gehrmann, Yonatan Belinkov, and Stuart M Shieber. Lstm networks can perform dynamic counting. *arXiv preprint arXiv:1906.03648*, 2019.

[140] Mirac Suzgun, Sebastian Gehrmann, Yonatan Belinkov, and Stuart M Shieber. Memory-augmented recurrent neural networks can learn generalized dyck languages. *arXiv preprint arXiv:1911.03329*, 2019.

[141] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir

Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[142] Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

[143] Ali Caner Türkmen, Yuyang Wang, and Alexander J Smola. Fastpoint: Scalable deep point processes. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2019.

[144] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[145] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[146] Herman Wandabwa, M Asif Naeem, Farhaan Mirza, Russel Pears, and Andy Nguyen. Multi-interest user profiling in short text microblogs. In *International Conference on Design Science Research in Information Systems and Technology*, pages 154–168. Springer, 2020.

[147] Fang Wang. Multi-interest communities and community-based recommendation. 2007.

[148] Jizhe Wang, Pipei Huang, Huan Zhao, Zhibo Zhang, Binqiang Zhao, and Dik Lun Lee. Billion-scale commodity embedding for e-commerce recommendation in alibaba. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 839–848, 2018.

[149] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. Neural graph collaborative filtering. In *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*, pages 165–174, 2019.

[150] Yequan Wang, Minlie Huang, Xiaoyan Zhu, and Li Zhao. Attention-based lstm for aspect-level sentiment classification. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pages 606–615, 2016.

[151] Joe H Ward Jr. Hierarchical grouping to optimize an objective function. *Journal of the American statistical association*, 58(301):236–244, 1963.

[152] Gail Weiss, Yoav Goldberg, and Eran Yahav. On the practical computational power of finite precision rnns for language recognition. *arXiv preprint arXiv:1805.04908*, 2018.

[153] Jason Weston, Ron J Weiss, and Hector Yee. Nonlinear latent factorization by embedding multiple user interests. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 65–68, 2013.

[154] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[155] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[156] Bin Xia, Yun Li, Qianmu Li, and Tao Li. Attention-based recurrent neural network for location recommendation. In *2017 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*, pages 1–6. IEEE, 2017.

[157] Shuai Xiao, Junchi Yan, Xiaokang Yang, Hongyuan Zha, and Stephen M Chu. Modeling the intensity function of point process via recurrent neural networks. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[158] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Yuyin Zhou, Lingxi Xie, and Alan Yuille. Adversarial examples for semantic segmentation and object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 1369–1378, 2017.

[159] Chengfeng Xu, Pengpeng Zhao, Yanchi Liu, Jiajie Xu, Victor S Sheng S. Sheng, Zhiming Cui, Xiaofang Zhou, and Hui Xiong. Recurrent convolutional neural network for sequential recommendation. In *The World Wide Web Conference*, pages 3398–3404, 2019.

[160] Xiaojun Xu, Chang Liu, and Dawn Song. Sqlnet: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436*, 2017.

[161] Gui-Rong Xue, Chenxi Lin, Qiang Yang, WenSi Xi, Hua-Jun Zeng, Yong Yu, and Zheng Chen. Scalable collaborative filtering using cluster-based smoothing. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 114–121, 2005.

[162] Xi Ye and Greg Durrett. The unreliability of explanations in few-shot prompting for textual reasoning. *Advances in neural information processing systems*, 35:30378–30392, 2022.

[163] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 974–983, 2018.

[164] Jiaxuan You, Yichen Wang, Aditya Pal, Pong Eksombatchai, Chuck Rosenburg, and Jure Leskovec. Hierarchical temporal convolutional networks for dynamic recommender systems. In *The world wide web conference*, pages 2236–2246, 2019.

[165] Changqian Yu, Jingbo Wang, Chao Peng, Changxin Gao, Gang Yu, and Nong Sang. Bisenet: Bilateral segmentation network for real-time semantic segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 325–341, 2018.

[166] Li Yu. Using ontology to enhance collaborative recommendation based on community. In *2008 The Ninth International Conference on Web-Age Information Management*, pages 45–49. IEEE, 2008.

[167] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887*, 2018.

[168] Wu Yue and Chen Xiang. A multi-interests model of recommendation system based on customer life cycle. In *2012 Fifth International Conference on Intelligent Computation Technology and Automation*, pages 22–25. IEEE, 2012.

[169] Wojciech Zaremba and Ilya Sutskever. Learning to execute. *arXiv preprint arXiv:1410.4615*, 2014.

[170] Haichao Zhang and Jianyu Wang. Towards adversarially robust object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[171] Junbo Zhang, Yu Zheng, and Dekang Qi. Deep spatio-temporal residual networks for citywide crowd flows prediction. In *AAAI*, pages 1655–1661, 2017.

[172] Lisa Zhang, Gregory Rosenblatt, Ethan Fetaya, Renjie Liao, William Byrd, Matthew Might, Raquel Urtasun, and Richard Zemel. Neural guided constraint logic programming for program synthesis. *Advances in Neural Information Processing Systems*, 31, 2018.

[173] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: an efficient data clustering method for very large databases. *ACM sigmod record*, 25(2):103–114, 1996.

[174] Qingyuan Zhao, Murat A Erdogdu, Hera Y He, Anand Rajaraman, and Jure Leskovec. Seismic: A self-exciting point process model for predicting tweet popularity. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1513–1522. ACM, 2015.

[175] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries

from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*, 2017.

[176] Chang Zhou, Jinze Bai, Junshuai Song, Xiaofei Liu, Zhengchao Zhao, Xiusi Chen, and Jun Gao. Atrank: An attention-based user behavior modeling framework for recommendation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[177] Yu Zhu, Hao Li, Yikang Liao, Beidou Wang, Ziyu Guan, Haifeng Liu, and Deng Cai. What to do next: Modeling user behaviors by Time-LSTM. In *Twenty-Sixth International Joint Conference on Artificial Intelligence (IJICAI)*, pages 3602–3608, 2017.

[178] Jinfeng Zhuang, Jennifer Zhao, Anant Subramanian, Srinivas, Yun Lin, Balaji Krishnapuram, and Roelof van Zwol. Pintext 2: Attentive bag of annotations embedding. In *Proceedings of DLP-KDD 2020*, 2020.