UNIVERSITY OF CALIFORNIA, SAN DIEGO

# Comparison and Fine-grained Analysis of Sequence Encoders for Natural Language Processing

A Thesis submitted in partial satisfaction of the requirements
for the degree Master of Science

in

Computer Science

by

Thomas Anderson Keller

Committee in charge:

Professor Garrison W. Cottrell, Chair
Professor Kamalika Chaudhuri
Professor Ndapandula Nakashole

2017

The Thesis of Thomas Anderson Keller is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

Chair

University of California, San Diego

2017

DEDICATION

To my mom, who fed me and kept me going through the good and the bad. To my dad, who has always supported me no matter what crazy decisions I made. To my brother, who believed in me even when I didn't. To my friends, who continually told me I'm the dumbest of the group.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I am very grateful for Gary's support, guidance, and mentorship throughout this research. I'd also like to thank professor Nakashole and professor Chaudhuri for their review and assessment of this work as members of the committee. Finally, I'd like to thank the other master's students of the Cottrell lab for their insightful commentary and discussions.

ABSTRACT OF THE THESIS

**Comparison and Fine-grained Analysis of Sequence Encoders for Natural Language Processing**

by

Thomas Anderson Keller

Master of Science in Computer Science

University of California, San Diego, 2017

Professor Garrison Cottrell, Chair

Most machine learning algorithms require a fixed length input to be able to perform commonly desired tasks such as classification, clustering, and regression. For natural language processing, the inherently unbounded and recursive nature of the input poses a unique challenge when deriving such fixed length representations. Although today there is a general consensus on how to generate fixed length representations of individual words which preserve their meaning, the same cannot be said for sequences of words in sentences, paragraphs, or documents. In this work, we study the encoders commonly used to generate fixed length representations of natural language sequences,

and analyze their effectiveness across a variety of high and low level tasks including sentence classification and question answering. Additionally, we propose novel improvements to the existing Skip-Thought and End-to-End Memory Network architectures and study their performance on both the original and auxiliary tasks. Ultimately, we show that the setting in which the encoders are trained, and the corpus used for training, have a greater influence of the final learned representation than the underlying sequence encoders themselves.

# Chapter 1

# Introduction

As the field of machine learning has continued to advance, applications to natural language processing have also seen significant improvements. Performance benchmarks on tasks such as machine translation and speech recognition have been shattered by novel modeling techniques and increased model capacity. This growth has largely been driven by improved methods of training of deep neural networks, as well as increased computational resources.

At the heart of many of these models is a distributed representation of language. The first step of any machine learning algorithm is to convert the input from its original form to a representation the model is able to learn from. For traditional algorithms, these inputs are typically required to be real numbers such that weights can be applied to each as the model is trained. For typical analytics tasks, this is not an issue since features are already in a real-valued representation. However, for natural language, the conversion between text and numeric inputs is not always as straightforward. Early natural language representations considered words and n-grams uniquely and distinctly [51], meaning the words 'King', 'Queen', and 'Dog' were all equally distant in representational space. Modern approaches have shown model performance and sample efficiency can be greatly

improved by representing natural language elements as vectors of real numbers such that semantically similar elements will be closer together [41]. These vectors can be thought of as an embedding of language in a high dimensional space.

Although there is a general consensus on how distributed word representations can be learned effectively, learning representations for sequences of words, be they sentences, paragraphs, or documents, is still an active area of research. This work will focus on representations of sentences as compositions of word vectors, and their application to natural language processing. Specifically, we will focus on modifications of the skip-thought architecture, an application of encoder-decoder networks to unsupervised sentence representation learning, and attempt to improve the learned representations on a variety of tasks. Additionally, we examine how sentence representations are employed in question answering models, specifically Memory Networks, and study the impact of increasingly complex sentence encoders on overall model performance. Ultimately, through fine grained analysis of the learned sentence embeddings, we reassert the power and viability of simple bag-of-words embeddings, and show how multiplicative position encodings can boost their performance to be similar to that of recurrent embeddings for certain tasks.

# Chapter 2

# Background

This chapter contains a brief overview of the necessary background on neural networks and their applications to natural language processing as sequence encoders.

## 2.1 Feedforward Neural Networks

Feedforward Neural Networks are the basic building blocks of modern sophisticated deep-learning architectures. In essence, they are composed layers of units, where each maintains a weighted connection to every unit in the layer above it. Formally, a generic feedforward neural network with $N$ hidden layers can be described as in equation 2.1, where $x$ is the input vector, $(W_0, ..., W_N)$ are the weight matrices, $(b_0, ..., b_N)$ are the bias vectors, $\sigma$ is an element-wise non-linear activation function (typically the hyperbolic tangent function, logistic function, or rectified linear function), and $\hat{y}$ is the output.

$$
\begin{aligned}
h_0 &= \sigma(W_0 x + b_0) \\
h_1 &= \sigma(W_1 h_0 + b_1) \\
&\ldots \\
\hat{y} &= \sigma_{out}(W_N h_{N-1} + b_N)
\end{aligned}
\tag{2.1}
$$

The size of the output of the final layer, and the final activation function $\sigma_{out}$, are chosen to suit the desired output. For example, for multi-class classification, the softmax function, given by equation 2.2, can be used to represent a categorical distribution over the desired classes where the output layer size is equal to the number of classes. Similarly, for simple single-variable logistic regression, a single output unit can be used with a logistic activation function.

$$
softmax(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \forall j \in \{1,..,K\} \tag{2.2}
$$

The network is trained by optimizing the parameters, encompassed by the weight matrices and the bias vectors, to minimize a defined loss on the training set. Typical loss functions include cross-entropy for classification and mean squared error for regression. Optimization is achieved through gradient descent methods such as stochastic gradient descent. For deep networks, the backpropagation algorithm is used to compute the gradient for all layers according to the loss of the final output through the chain rule of derivatives. In this work, we employ feedforward networks, which we will frequently call multi-layer perceptrons (MLP), for classification and other prediction tasks using fixed-length sentence representations.

## 2.2 Recurrent Neural Networks

This work makes heavy use of recurrent neural networks (RNN) and their variants. The simple recurrent neural network [24], exhibits the most basic form of recurrence in a neural network architecture. At a high level, RNNs maintain a hidden state which is updated with each input, and used to predict the output for the given time-step. The use of a hidden state which persists across time-steps differentiates RNNs from feedforward neural networks on which they are based. Using this hidden state, RNNs can map input sequences to output sequences, and also maintain a state representing the accumulation of the input at each point in the sequence. Equations 2.3 describe the operation of a simple recurrent neural network where $x_t$ is the input for time $t$, $h_{t-1}$ is the hidden state of the network from the previous time-step, $(W_h, W_y, U_h)$ are weight matrices, and $(b_h, b_y)$ are bias vectors. Again, $\sigma$ and $\sigma_{out}$ are the hidden and output non-linearities.

$$\begin{aligned} h_t &= \sigma\left(W_h x_t + U_h h_{t-1} + b_h\right) \\ y_t &= \sigma_{out}\left(W_y h_t + b_y\right) \end{aligned} \tag{2.3}$$

Similar to feedforward networks, RNNs are trained via gradient descent with a modified version of the backpropagation algorithm called backpropagation through time (BPTT)[32]. In the forward pass of BPTT, the activations of the units are stored across time steps. The following backwards pass then uses these activations to recursively compute the required gradients. The loss is typically defined as the sum of losses for each output over all time-steps.

Recurrent Neural Networks can also be viewed as deep feed-forward networks with shared weights across time steps by unrolling the above equations across time $t$. An example of this 'unrolling' is presented in figure 2.1. Figure 2.2 describes the notation used for all figures in this section. In our equations we represent the input to

each nonlinearity as a sum of multiple distinct matrix-vector products, however, simple algebraic manipulation shows this to be equivalent to vector concatenation followed by a single matrix-vector product as is displayed in the figures.

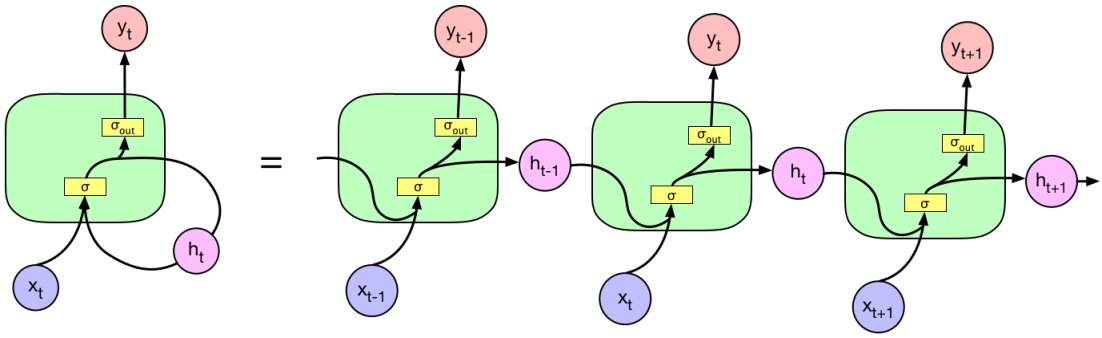


**Figure 2.1**: Example of a recurrent neural network unrolled in time.

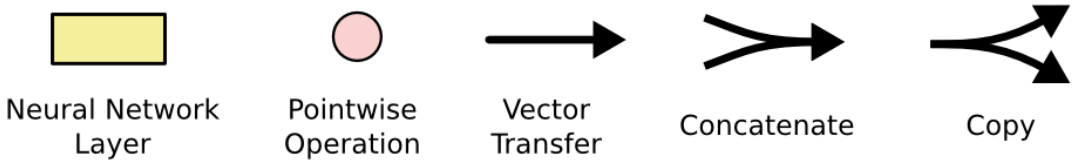


**Figure 2.2**: Notation used for all recurrent network figures in this section. Figure from [11].

Although the cycles in the graph of the RNN allow the network to theoretically keep information about past inputs for an unlimited number of steps, it has been shown that the basic BPTT algorithm is not sufficiently powerful to discover dependencies of the input spanning long time intervals. Bengio et al. [48] provide analysis showing the magnitude of the derivative of the system at time $t$ with respect to the state at time 0 decreases exponentially as $t$ increases. This is sometimes referred to as the vanishing gradient problem.

To help remedy vanishing gradients, and improve the RNNs ability to discover long-range dependencies, the long short-term memory (LSTM) unit was created [39]. The LSTM unit uses a series of gated connections to maintain a separate internal cell state

which it updates based on the current input and its main hidden state. This separate cell state allows information to be retained for an unlimited number of time steps, mitigating the problems of the simple RNN. The gates used to accomplish this include the forget gate $f_t$, which is used to remove information from the cell state, the input gate $i_t$, which is used to update the cell state with information from the hidden state, and the output gate $o_t$, which is used to control which portions of the cell state will be output as the next hidden state. Equations 2.4 describe the operations of the LSTM unit formally, and a corresponding graphic representation of an unrolled LSTM unit is presented in figure 2.3.

$$
\begin{aligned}
f_t &= sigmoid\left(W_f x_t + U_f h_{t-1} + b_f\right) \\
i_t &= sigmoid\left(W_i x_t + U_i h_{t-1} + b_i\right) \\
o_t &= sigmoid\left(W_o x_t + U_o h_{t-1} + b_o\right) \\
\widetilde{C}_t &= tanh\left(W_c x_t + U_c h_{t-1} + b_c\right) \\
C_t &= f_t \odot C_{t-1} + i_t \odot \widetilde{C}_t \\
h_t &= o_t \odot tanh\left(C_t\right)
\end{aligned}
\tag{2.4}
$$

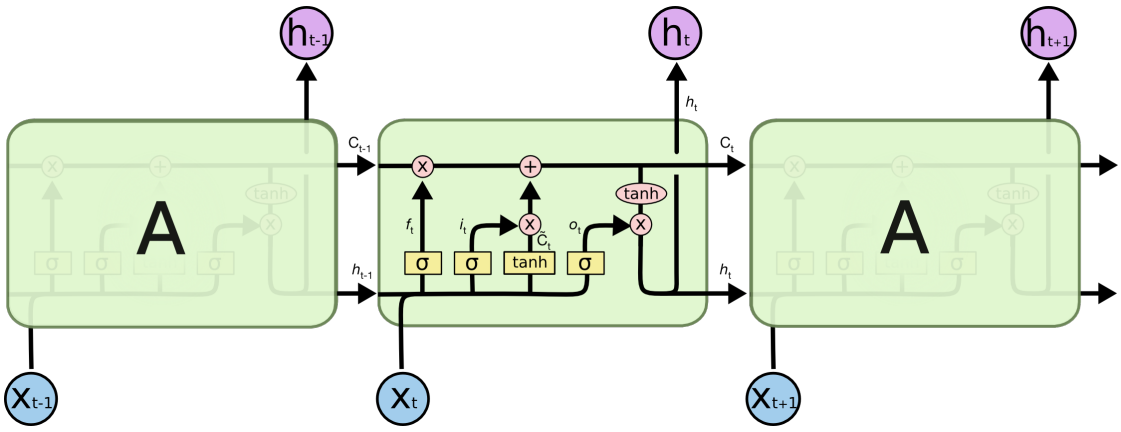


**Figure 2.3**: Unrolled LSTM unit. Figure from [11].

We note the LSTM cell has around four times the number of parameters as a basic

hidden unit, causing training times and sample complexity to be equivalently increased. An alternative to the LSTM unit, the gated recurrent unit (GRU) [28], was recently proposed to reduce the total number of parameters while maintaining an internal gated memory, and has been shown to achieve comparable performance on many tasks. To achieve this, the forget gate $f_t$ and input gate $i_t$ are combined into a single 'update gate' $z_t$, and the cell state $c_t$ is merged with the hidden state $h_t$. The equations for the GRU are given in equations 2.5 with a corresponding diagram in figure 2.4.

$$\begin{aligned} z_t &= sigmoid\left(W_z x_t + U_z h_{t-1} + b_z\right) \\ r_t &= sigmoid\left(W_r x_t + U_r h_{t-1} + b_r\right) \\ \widetilde{h_t} &= tanh\left(W_h x_t + U_h\left(r_t \odot h_{t-1}\right) + b_h\right) \\ h_t &= \left(1 - z_t\right) \odot h_{t-1} + z_t \odot \widetilde{h_t} \end{aligned} \tag{2.5}$$

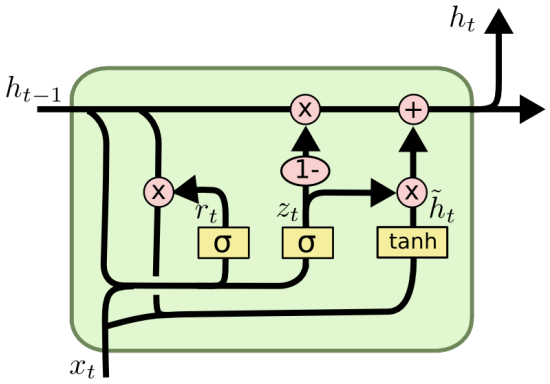


**Figure 2.4**: GRU cell. Figure from [11].

We mainly use GRU cells in this work unless stated otherwise. This work also makes use of bi-directional RNNs, which are simply a combination of two RNNs, one which processes the input in the standard order, and one which processes the input in reverse order. The hidden states of these two RNNs can then be summed or concatenated together to form a single set of hidden states.

## 2.3 Distributed Representation of Language

The idea of a distributed representation of language originates from computational linguistics and the early connectionist literature, with distributed representations of symbolic data [17]. A distributed representation typically refers to a dense real-valued vector representation of the inputs such that similarities between inputs can be represented by distances, or other metrics, in the vector space. For language, distributed representations have historically been used for information retrieval [18] and document classification, such as Latent Semantic Indexing (LSI) [40], with word representations being computed from word co-occurrence statistics within documents. Specifically, LSI uses a bag of words vector to represent each document (a vector with each element denoting the frequency of a word in the document), stacks these vectors into a term-document matrix $C$, and computes a low-rank approximation to this matrix: $C = TSD^T$. The low-rank approximation is typically computed via singular value decomposition (SVD), with the rows of $T$ representing individual words, and rows of $D$ representing individual documents.

More recently, neural networks have been used to learn distributed representations of language in the service of specific tasks and have been shown to achieve state of the art-performance in both supervised and unsupervised settings. Below, we describe how neural network models are constructed to learn word representations and how the idea can be expanded to learn representations of phrases, sentences, and documents.

### 2.3.1 Word Representations

The idea behind distributed word representations, often called word-vectors, is to learn a dense vector representation for each word in a vocabulary such that words with similar semantic meanings can express this through a comparison of their vector

representations, rather than being entirely distinct inputs to a model. Today, there is a general consensus on how word-vectors can be effectively learned from unlabeled text, with many sets of pre-trained word vectors available online [41] [23].

Relatively recently, the skip-gram model [41] was proposed as an unsupervised neural architecture to learn word embeddings from unstructured text. The idea is to learn dense vector representations of words which assist in predicting the words in a context window surrounding a given word. The belief that this will produce useful embeddings stems from the distributional hypothesis of language, which states that words that occur in similar contexts tend to have similar meanings [51].

The skip-gram model itself is surprisingly simple, consisting of a single layer neural network encoder (a matrix $W$ of shape (vocabulary size, embedding size)) and another single layer decoder $C$ with no non-linearity, as displayed in figure 2.5. The basic model is parameterized with a softmax over the output vocabulary, and trained to maximize the likelihood of (word, context) pairs in the dataset.

However, given the size of the vocabulary $|V|$ is typically very large, and the computational complexity of the softmax scales linearly with its size, this formulation is impractical. Instead, it has been shown that training a model to differentiate real data from noise through logistic regression, via a method known as Noise Contrastive Estimation (NCE) [5], approximately maximizes the same objective as the softmax while being significantly more computationally efficient. The accuracy of the approximation is controlled by the number of negative examples $k$ presented for each positive example, with the NCE objective matching the softmax in the limit $k = |V|$.

In the Skip-gram paper, Mikolov et al. propose a simplified variant of NCE, called Negative Sampling (NS), which makes use of the same idea (distinguishing real data from noise), but relaxes the constraint that it must approximate the objective of the softmax. The equations for the conditional probabilities of a (word $w$, context-word

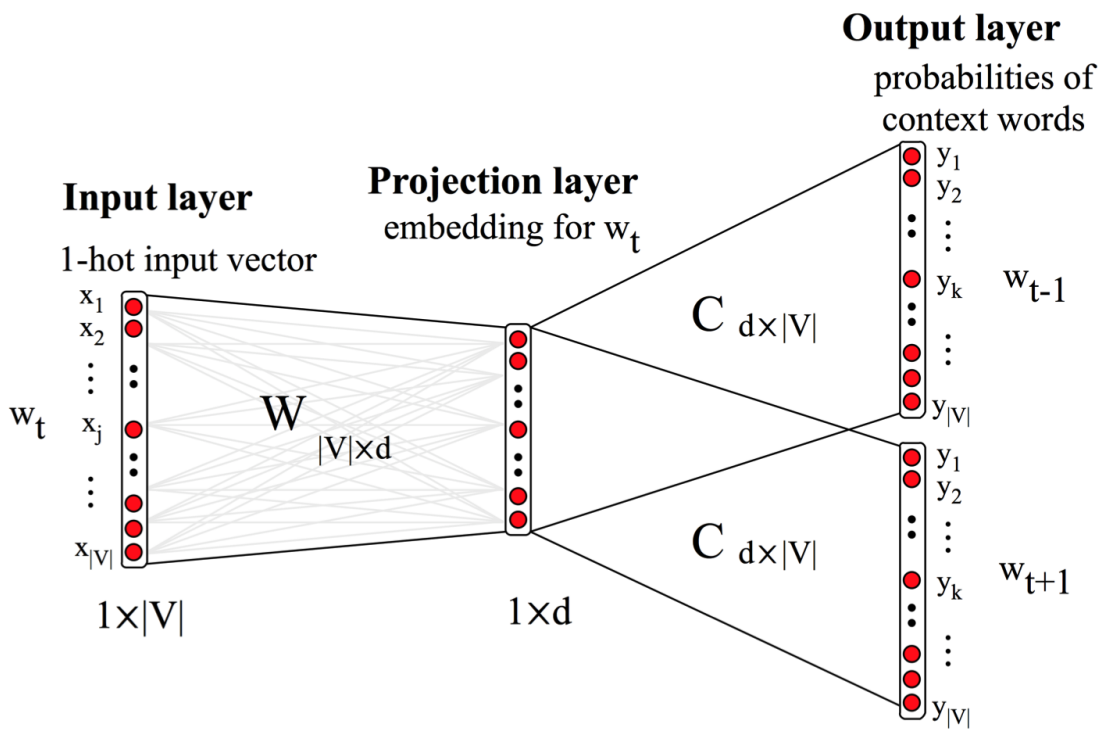


**Figure 2.5**: Skip-gram model for learning word representations. Where $w_{t-1}, w_t, w_{t+1}$ is a naturally occurring sequence of words, $|V|$ is the size of the vocabulary, and $d$ is the embedding size. The rows of W correspond to word-embeddings, and the columns of C correspond to context-word embeddings. Figure from [12].

$c$) example being from the data $(D = 1)$, or being from the noise distribution $(D = 0)$, are provided in equation 2.6, where $\vec{w}, \vec{c}$ are the word and context-word embeddings respectively, and $\theta$ denotes the parameters of the embedding matrices. The objective to maximize for each example is then given by equation 2.7, where the $k$ negative examples $\vec{c}_N$ are sampled from the unigram distribution of the dataset $P_d$.

$$\begin{aligned} P(D = 1|c, w; \theta) &= \frac{1}{1 + exp(-\vec{c} \cdot \vec{w})} = \sigma(\vec{c} \cdot \vec{w}) \\ P(D = 0|c, w; \theta) &= 1 - P(D = 1|c, w; \theta) = \sigma(-\vec{c} \cdot \vec{w}) \end{aligned} \tag{2.6}$$

$$\arg\max_{\theta} \log \sigma(\vec{c} \cdot \vec{w}) + k \cdot \mathbb{E}_{c_N \sim P_d} [\log \sigma(-\vec{c}_N \cdot \vec{w})] \tag{2.7}$$

### 2.3.2 Sentence Representations

Similar to word-vectors, sentence-vectors attempt to encode semantic information of natural language in fixed-length dense vectors. The goal is to produce sentence-vectors such that sentences with similar meanings express this through similar embeddings. However, due to the complexity and variability of natural language sequences, this task is significantly more complex than that of word representations.

In their most basic form, dense vector-representations of sentences can be formed by summing word embeddings of the words in the sentence. This is equivalent to encoding the sentence as a bag of words vector and taking the matrix-vector product with the word embedding matrix. Despite its simplicity, this type of encoder, called a bag of words (BoW) sentence encoder by Sukhbaatar et al. [36], has been shown to perform surprisingly well, and due to its low computational complexity it is still in use today. One of the main shortcomings of bag of words encoders is their lack of positional information of words within a sentence. To overcome this, numerous recent works have proposed applying a multiplicative mask to the sentence before summation [36] [6] [10] [30], allowing the position of words to have some impact on their final representation. This mask can either be learned during training, or predefined. We will examine the success of these position encodings at accomplishing the goal of preserving word-order information in this work.

Finally, as will be a main focus of this study, recurrent neural networks are used in a variety of forms today as sentence encoders. Their strengths come from their ability to encode arbitrary length sequences while maintaining word order and their ability to learn compositional features of their inputs. To create sentence embeddings, word vectors are fed as input to the RNN one step at a time, updating the internal hidden state of the network based on each input, until the end of sentence token is reached. Then, the final hidden state of the network can be projected, or taken directly, as the encoded sentence.

This approach has been used successfully to generate fixed-length representations of sequences in numerous works [19] [6].

## 2.4 Skip-Thought Vectors

The skip-thought architecture, originally proposed in [35], is an unsupervised architecture capable of training a generalizable sentence encoder. The idea stems directly from the distributional hypothesis behind the skip-gram architecture, and thus derives its name. However, unlike the skip-gram model, the skip-thought model operates at the sentence level, attempting to predict the previous and next sentences from a given sentence in the middle of a story. Given the increased difficulty of this task, more complex models, namely recurrent neural networks, must be used for the encoders and decoders. The full detailed model description is provided in Chapter 3.

After training, the sentence encoder can be used alone to extract 'skip-thought' vectors for a given sentence. Using these vectors, Kiros et al. were able to achieve performance competitive with state-of-the art models on a wide variety of tasks including semantic-relatedness, paraphrase detection, and general classification. The reason this achieved significant attention was due to the fact that most of the state-of-the-art models were trained in a supervised manner to generate sentence representations specifically for the tasks at hand, limiting their general applicability, while the skip-thought vectors were trained in an entirely unsupervised manner and generalized across many tasks.

## 2.5 Neural Network Language Models

Language models, defined as a joint probability distribution over sequences of words, have played an important role in many NLP tasks ranging from speech recognition

to machine translation [34]. Although the history of language modeling is diverse, somewhat recently neural networks have begun to play a larger role. Starting with [46], Bengio et al. demonstrated that distributed representation of words could be learned at the same time as the parameters of a language model and achieve good results with a feedforward neural network. In our work, we focus on recurrent neural network language models. Language model training of recurrent networks is performed by training the network to predict the next word in a sequence, given the previous words, and is the current state of the art in unsupervised modeling of natural language sentences [42]. Using these models as a form of unsupervised pre-training has been shown to improve performance on a range of tasks [33].

## 2.6 Variational Autoencoders

Variational autoencoders [13] are similar to the standard autoencoder except with a non-deterministic encoder function, and an imposed prior over the representations generated by this encoder. At a simple level, the encoder learns codes for the input not as single points, but as gaussian regions in latent space, forcing the codes to fill the space rather than mapping the input to a single deterministic point. The prior determines the shape of these regions, and is enforced by adding the KL divergence of the generated codes from the prior to the loss function. Empirically, this causes the network to be required to generate plausible outputs from all regions in the latent space encompassed by the prior, rather than the point-based coverage of standard autoencoders. Specifically, variational autoencoders rely on a reparameterization trick [14] to allow the model to remain differentiable while sampling from the hidden space. In this work, we take inspiration from Bowman et al. [37] who used a variational autoencoder for sentences, and showed that the network was able to learn to interpolate between sentences in the

latent space, while still maintaining coherence of the intermediate sentences.

## 2.7 Memory Networks

Memory Networks [21], created in 2015, can be seen as a form of recurrent neural network with an external addressable memory. Although their architecture is generally applicable to any input data, their applications have been focused on the text domain, and they have been shown to achieve competitive performance on the task of question answering based on supporting documents. Similar to more complex memory augmented neural networks (MANN) such as the Neural Turing Machine [2], and the Differentiable Neural Computer [3], Memory Networks learn to use their external memory to solve tasks which require a form of complex reasoning. Unlike MANN's, however, Memory Networks do not update old memories, and instead only store each new input as a distinct immutable memory. Each memory is stored as a fixed-length sentence vector, and the number of sentence vectors stored in memory can be unlimited. Due to this unbounded size, the original memory network uses a supervised label for supporting sentences during training, providing direct supervision for the memory-attention mechanism as to which memories are required to answer the query.

Successor to the original Memory Network, the End-to-End Memory Network [36] has a nearly identical form to its predecessor, but does away with the need for supervision of supporting sentences. This allows the network to be trained directly from question-answer-story triplets, and makes the model much more generalizable. The architecture and equations describing the operation the End-to-End Memory Network are provided in detail in Chapter 4. Given their unique architecture, Memory Networks make heavy use of sentence encoders and thus are featured as a main method of analysis in this work.

## 2.8 Organization of the Thesis

We organize this thesis as follows. First, we describe the methods used for training and evaluating skip-thought vectors, as well as our proposed architecture modifications. Second, we discuss the methods used for training and evaluation of memory networks and present our proposed sentence representation improvements in detail. Next, we present results and discussion for both the skip-thought and memory network modifications. This is followed by a summary of similar work specifically related to learning sentence encoders and the analysis of sentence embeddings. We additionally discuss work related to question answering with a focus on the sentence encoders used. Finally, we present concluding remarks and provide additional results in the appendix.

# Chapter 3

# Methods: Skip-Thought Vectors

Here we describe the methods and dataset used for training and evaluation of traditional skip-thought vectors. Additionally, we describe the suggested improvements proposed by this work and the architectures of these models in detail.

## 3.1 Dataset

The BookCorpus dataset, introduced in [50], is a collection of 11,038 novels, totaling over 74 million sentences, and spanning 16 different genres from Romance to Science Fiction. The books are free online, written by unpublished authors, and contain varied dialogue, interactions between characters, and general scene descriptions. Their breadth is intended to aid in the learning of generalizable sentence encoders not biased towards a particular domain; however, as demonstrated in [1], encoders trained on this dataset tend to perform better when evaluated on similar text and worse on dissimilar text such as reviews of consumer goods. As required by the distributional hypothesis underlying the skip-thought architecture, the dataset is composed entirely of contiguous text. This dataset was used for the original skip-thought model training, as well as for training our proposed architecture modifications.

The original skip-thought paper [35], does not explicitly mention the number of training iterations, or epochs, performed to produce the results given. It does mention that training was performed for approximately two weeks. Due to the size of this dataset, it appears that two weeks is roughly equivalent to a single epoch of training on a single modern GPU, thus we use this as a benchmark for our experiments.

## 3.2 Evaluation Metrics

For comparison with the original skip-thought work [35], we use seven of the eight original tasks to evaluate the performance of the trained skip-thought models: semantic-relatedness, paraphrase detection, and 5 standard classification benchmarks. Image-sentence ranking using the Microsoft COCO dataset was chosen to not be used for this study, given that it requires image features from a trained convolutional neural network for each image, and neither the network weights, nor the pre-processed feature representations were available online, so a valid comparison could not be easily guaranteed.

### 3.2.1 Semantic-relatedness

The dataset used for semantic relatedness is the SemEval 2014 Task 1 SICK dataset [29]. The dataset is composed of pairs of sentences with a score between 1 and 5 describing how semantically similar the sentences are as determined by an average of 10 human annotations. The dataset contains roughly 10,000 sentences, evenly split between training and testing. The metrics used to evaluate predictive performance on this dataset are Pearson's $r$, Spearman's $\rho$, and mean squared error (MSE). In our experiments we find all three metrics to be highly correlated, and thus only present MSE in our results.

To predict a score from a set of sentence vectors $u$ and $v$, we follow the methodology used in [35] and [26], where two sentence-pair features were created by encoding each sentence with the network. Given two sentence vectors $u$ and $v$, the pair features were computed as an elementwise product $u \cdot v$, and elementwise absolute difference $|u - v|$. These features were then concatenated together, and used as input to a logistic regression classifier trained over the SICK dataset. We use the original classifier implementation from [35] and thus refer to this work for an exact description of the training set creation procedure.

### 3.2.2 Paraphrase detection

The dataset used for paraphrase detection is the Microsoft Research Paraphrase Corpus (MSRP) [7]. The dataset is composed of 4076 sentence pairs for training, and 1725 pairs for testing, where the goal is to predict whether one sentence is a paraphrase of the other. Sentence-pair features are computed the same way as for the semantic-relatedness task as the elementwise product $u \cdot v$, and elementwise absolute difference $|u - v|$. Again these features are used to train a logistic regression classifier to predict the binary label with performance measured as accuracy and F1 score. In [35], a second set of experiments is performed where additional basic statistics between sentence pairs are added to the feature vector. These features are shown to improve performance, but given our goal is a direct comparison of our proposed architecture modifications, these additional features are irrelevant and unnecessary, so they were not included.

### 3.2.3 Classification benchmarks

The five classification benchmarks used for evaluation are: movie review sentiment (MR) [9], customer product reviews (CR) [31], subjectivity/objectivity classification

(SUBJ) [8], opinion polarity (MPQA) [20], and question-type classification (TREC) [43]. For all tasks, skip-thought vectors are computed and a logistic regression classifier is trained on top, and tuned with cross validation.

## 3.3 Original Skip-Thought Model

The heart of the skip-thought model is the encoder-decoder network first introduced in [27]. The basis of the model is to use a recurrent neural network to read an input sequence one token at a time, modifying the hidden state of the model at each step, until the end-of-sentence token is reached. At this point, the entire input sequence is encoded in the final hidden state of the network, and is transferred to a second 'decoder' RNN to attempt to reproduce the target sequence. An example of the basic encoder-decoder architecture from [27] is shown in Figure 3.1.

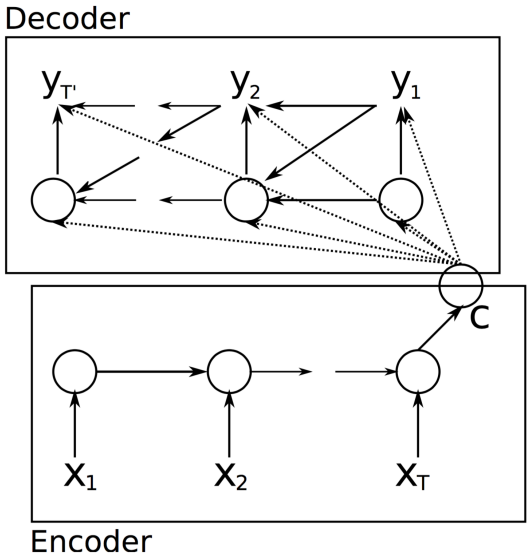


**Figure 3.1**: Basic encoder-decoder model where the decoder gets the encoded representation as an additional input at each step. Each circle represents a hidden state of the RNN, where C is the encoded representation of the entire input sentence. Figure from [27].

This model was popularized in [19], termed a sequence to sequence model for

machine translation, and has since been applied to a variety of challenges. As it relates to this work, the same model is used in the Skip-Thought architecture, with the only modification being that instead of a single decoder, the encoded representation is now fed to two separate decoders which attempt to decode the preceding and subsequent sentences in the text. A graphical example of the skip-thought model from [35] is shown in Figure 3.2.

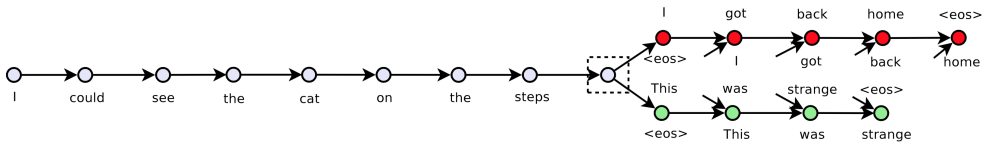


**Figure 3.2**: Skip-Thought model. Given a triplet of contiguous sentences, the middle sentence is encoded and the two decoders attempt to reconstruct the next and previous sentences. Figure from [35].

For the implementation, the regular RNN units are replaced with GRU units to provide a working memory. The equations which describe the encoder are the same as in equations 2.5 with $x_1^i, ..., x_N^i$ being the embeddings of the words in sentence $s^i$, where $N$ is the number of words in the sentence. Next, the final hidden state of the encoder, $h_N$, which we denote below as $h^i$ for sentence $s^i$, is used to bias the computation of each state of the decoder. The equations to compute each hidden state of the decoder for sentence $s^{i+1}$ are given in equations 3.1, where $x_{t-1}^{i+1}$ is the previous word in the sentence taken from the ground truth, a technique known as teacher forcing. For $t = 0$, we set the first input $x_{-1}^{i+1}$ to be a vector of all zeros, forcing the network to rely on the the encoders final

hidden state for reconstruction.

$$
\begin{aligned}
r_t &= \sigma(W_r^d x_{t-1}^{i+1} + U_r^d h_{t-1} + C_r h^i) \\
z_t &= \sigma(W_z^d x_{t-1}^{i+1} + U_z^d h_{t-1} + C_z h^i) \\
\bar{h}_t &= \tanh(W^d x_{t-1}^{i+1} + U^d (r_t \odot h_{t-1}^{i+1}) + C h^i) \\
h_t^{i+1} &= (1 - z_t) \odot h_{t-1}^{i+1} + z_t \odot \bar{h}_t
\end{aligned}
\tag{3.1}
$$

As can be seen, these are identical to the standard GRU equations, with the bias replaced by a projection of the encoder's final hidden state. The same equations are used to compute the hidden states for sentence $s^{i-1}$, with separate weight matrices. Given $h_t^{i+1}$ and the previous $t-1$ output words $w_{<t}^{i+1}$, the probability of the output word $w_t^{i+1}$, is parameterized by a softmax where $v_{w_t^{i+1}}$ is the word embedding for word $w_t^{i+1}$ in the vocabulary lookup table. This is formally written in equation 3.2.

$$
P(w_t^{i+1} | w_{<t}^{i+1}, h^i) \propto exp(v_{w_t^{i+1}} \cdot h_t^{i+1}) \tag{3.2}
$$

Finally, the objective function to be maximized over all training triplets can be expressed using these output probabilities as in equation 3.3.

$$
\sum_t \log P(w_t^{i+1} | w_{<t}^{i+1}, h^i) + \sum_t \log P(w_t^{i-1} | w_{<t}^{i-1}, h^i) \tag{3.3}
$$

## 3.4 Variational Skip-Thought

The basic idea behind variational auto encoders is to add gaussian noise to the representation produced by the auto-encoder, forcing the representational space generated to be continuous and more evenly distributed. The key is to do this in a differentiable manner. To achieve this, the reparameterization trick of [14] is used to sample from

a gaussian around a given point while maintaining differentiability. The trick consists of observing that sampling $z$ from $\mathcal{N}(\mu, \Sigma)$ is equivalent to sampling $\varepsilon \sim \mathcal{N}(0,1)$ and setting $z = \mu + \sigma\varepsilon$, which is differentiable w.r.t. $\mu$ and $\sigma$. We can then use networks to predict $\mu$ and $\sigma$ conditionally based on the input while maintaining differentiability.

For sequence auto encoders, this involves using the final representation of the input produced by the encoder to predict the mean and standard deviation of a gaussian from which the new encoded representation is sampled, and which the decoder decodes. A graphical depiction of a sequence variational autoencoder is shown in Figure 3.3.

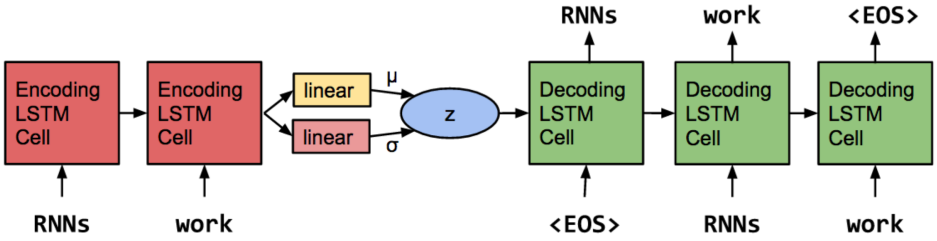


**Figure 3.3**: Variational autoencoder language model from which the variational skip-thought model is based. Figure from [37].

The equations which describe the variational skip-thought model are only a slight modification of the original equations. First, the encoder and decoder remain the same, with the only difference being a 'variational layer' added between. This layer takes as input the final state of the encoder, $h_N$, and maps it to the mean and the log of the standard deviation of the region of latent space for the input, as in equation 3.4.

$$\begin{aligned} \mu_z &= W_{h\mu} h_N + b_{hz} \\ \log \sigma_z &= W_{h\sigma} h_N + b_{h\sigma} \end{aligned} \tag{3.4}$$

Next, following the reparameterization trick, we sample an $\varepsilon$ from the standard

normal distribution and use it as follows to compute the sampled latent variable $z$.

$$z = \mu_z + e^{\log \sigma_z} * \varepsilon \tag{3.5}$$

This latent variable is then mapped through a standard single layer network to produce the new hidden state to be decoded by the decoder, $h^i$ in equation 3.3. This hidden state is used by both the forwards and backwards decoders.

$$h_{decoder} = h^i = tanh\left(W_{zh}z + b_{zh}\right) \tag{3.6}$$

Finally, the KL divergence between the posterior distribution of $z$ and the standard gaussian prior is added to the cost of the model, resulting in the new objective function in equation 3.7.

$$\sum_t \log P(w_t^{i+1}|w_{<t}^{i+1}, h^i) + \sum_t \log P(w_t^{i-1}|w_{<t}^{i-1}, h^i) - \alpha * KL\left(p(z|h^i)||\mathcal{N}(0,1)\right) \tag{3.7}$$

The $\alpha$ term in equation 3.7 is the weighting of the KL divergence term relative to the reconstruction error. Following [37] we use a sigmoid annealing schedule for the KL divergence term, starting at 0, and quickly increasing to 1 by batch 10,000. Through experimentation, we found reducing this weight by a factor of 10 improved the performance of the model, so our $\alpha$ term instead increased from 0 to 0.1.

## 3.5 Auto-Encoder Regularization

A second set of experiments were conducted based on sequence to sequence auto-encoders [4]. The idea is to regularize the learning of the skip-thought encoder by forcing it to generate a representation which is not just helpful for decoding the

surrounding sentences, but also for reproducing the input sentence. The hope is that such an additional constraint would force the model to include a more accurate representation of the input sequence and prevent overfitting.

The construction of such a model is achieved by simply adding a third output decoder branch to the model which is penalized to reproduce the input sequence. The input encoder, forwards, and backward decoders remain identical to the original model, with the only modification being an additional decoder being added. The equations which describe this decoder are the same as in equation 3.8, simply replacing sentence $i+1$ with sentence $i$.

The main difference lies in the objective function of the model which is formulated as follows:

$$\sum_t \log P(w_t^{i+1}|w_{<t}^{i+1},h_i) + \alpha \sum_t \log P(w_t^i|w_{<t}^i,h_i) + \sum_t \log P(w_t^{i-1}|w_{<t}^{i-1},h_i) \tag{3.8}$$

Where $\alpha$ is a parameter which can be dynamically modified to control the effect of the auto-encoder regularization during training. In our experiments, we tested the constant $\alpha = 1$ as well as $\alpha = 0.33$. We believe an additional interesting test would be to use a linearly decreasing value for alpha, as a function of training iterations, causing it to act as a form of pre-training. However, due to the extended training time of this model, this has not yet been explored. The results presented in Chapter 5 use $\alpha = 1$.

# Chapter 4

# Methods: End-to-End Memory Networks

## 4.1 Dataset

The dataset used for training and evaluation of memory networks in this study is the question-answering portion of the bAbI dataset. The bAbI dataset [22] was created in 2015 and was intended to be a set of 'toy tasks' to measure reading comprehension and understanding. There are 20 question-answer tasks in total, where each task tests a different reasoning capability of the network required to answer the posed question. These capabilities include chaining facts, simple induction, deduction, and many more. Each training example in the dataset is composed of a set of 'story' sentences, a single question sentence, and the resulting one-word answer which is derived from the provided story. Although the original paper includes some experiments with multi-word answers, all experiments in this paper will focus on tasks with single word answers. An example of Task 2: Two Supporting Facts, is provided below. Example (story, question, answer) triplets for all tasks are provided in the appendix.

- **Story**:

    1. Mary got the milk there.
    2. John moved to the bedroom.
    3. Sandra went back to the kitchen.
    4. Mary travelled to the hallway.

- **Query**: Where is the milk?
- **Answer**: hallway

The examples also include a label for 'supporting' story sentences, which can be used to provide supervision for which sentences are required to answer the question. In the above example, the supporting sentences would be 1 and 4. These labels are only used for the original memory network, and are not used for the end-to-end memory network or in any of our experiments.

The data is produced using a simulation of characters moving around and interacting with a simple world, and a basic grammar is used to produce human readable sentences from the interactions. Because of the synthetic nature of the dataset, the sentences are somewhat mechanical, as seen in the example above, and vocabulary size is limited (20 to 50 unique words for each task). The main versions of the bAbI dataset used in the literature include a small version, with 1000 training examples for each task (bAbI-1k), and a larger version with 10,000 training examples for each task (bAbI-10k). We explored the performance of our representations on both datasets and found the larger dataset is better suited for training the more complex sentence representations. Therefore, most results in this study use the 10k dataset unless stated otherwise. Specifically, the basic position encoder results presented in tables 6.1 and 6.2 use the bAbI 1k dataset, as

well as the skip-thought encoder experiment presented in table 6.9. The remainder of the other memory network experiments use the bAbI 10k dataset.

## 4.2 Evaluation Metrics

Models are evaluated based on the overall answer accuracy for each task separately. Models can be trained jointly over all tasks, or trained on each task independently, but accuracy is always separated by task. In our experiments, we observed limited improvement from joint training, so all memory network experiments are trained on single tasks independently. We leave joint evaluation of the proposed models to future work.

Tasks are considered to be solved, or 'passed', if they achieve an accuracy over 95%. As reported in the original work, as well as in subsequent work [10], certain tasks such as 3, 17 and 18 have a high degree of variability in model performance dependent on weight initialization for the model. This variability can cause the model to pass tasks on certain runs, while achieving less than 50% accuracy on other runs. To overcome this, training is repeated 10 times for each task, and the model with the highest validation accuracy is presented in the results.

## 4.3 End-to-End Memory Networks

The End-to-End Memory Network [36] depicted in Figure 4.1 will be the starting point for evaluation of our improved sentence representations on the bAbI [22] question answer tasks. Roughly, the model is composed of a query-input component, a memory-input component, a memory-output component, and a final response component. The memory input and output components can be repeated to allow the model to have multiple

distinct memory access operations, called memory hops, before producing a response. We will initially describe a single hop version of this model using the baseline bag of words encoder, and discuss multiple hop and alternate encoder expansions later.

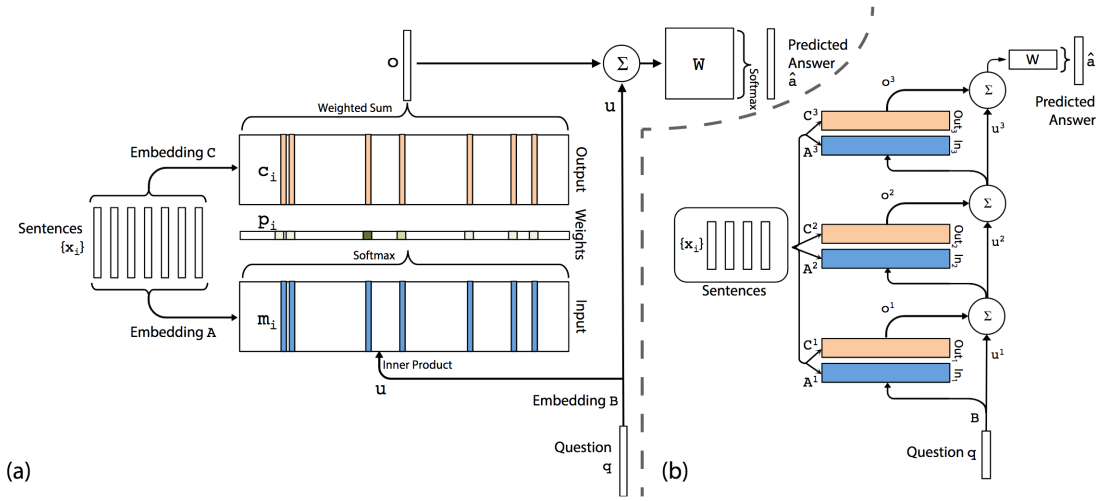


**Figure 4.1**: (a) A single layer version of the End-to-End Memory Network. (b) A three layer version of the model. Memory-input components are colored blue and memory-output components are colored orange. Figure from [36].

The operation of the model can be described in words as follow. Given a query sentence $q$, represented as a bag-of-words vector (i.e. having dimension equal to the vocabulary size, and each element of the vector denoting the frequency of the corresponding word in the sentence), the query-input component generates a fixed-length sentence representation of the query through matrix-vector multiplication with the embedding matrix $B$. The resulting dense vector representation of the query is stored as the initial internal state of the model $u^0 = Bq$.

Next, given a set of story sentences $x^1, ..., x^i$, again represented as bag-of-words vectors, the memory-input component is responsible for generating an input-encoding $m_i$ for each sentence $x^i$ using the embedding matrix $A^0$. This is again achieved through matrix-vector multiplication. The memory-input component then compares each memory vector $m_i$ with the internal state of the model $u^0$ through a dot product, and performs a

softmax over all memory locations to assign a probability value $p_i$ to each memory. This value can be interpreted as the relevance score between the current internal state of the model (initially the encoded query) and the respective memory.

The memory-output component then uses these probability values to return a weighted sum of the encoded memories $o^0 = \sum_i p_i c_i$, where $c_i$ is the output encoding of each memory sentence $x^i$, and is computed using the separate embedding matrix $C^0$. This weighted sum of memories is then added to the internal state of the model to produce the new internal state $u^1 = u^0 + o^0$.

From this definition, we see that the model can naturally be extended to have multiple sets of memory-input and memory-output components, called memory 'hops', by creating additional embedding matrices $A^1, C^1, ..., A^K, C^K$ for up to $K$ hops, and using the updated internal state of the model for querying the memory at each step. These hops provide the network with multiple distinct memory access operations, thereby allowing it to perform complex reasoning tasks which require multiple steps.

Finally, after the last memory hop is complete, the response component multiplies the final internal state of the model with a weight matrix $W$ of shape (embedding size, answer-vocabulary size), and performs a softmax over the answer-vocabulary dimension to predict the output word.

The operations for a single-hop model are written formally in equations 4.1, 4.2 and 4.3.

$$u^0 = Bq \tag{4.1}$$

$$
\begin{aligned}
m_i &= A^0 x^i \quad \forall i \\
p_i &= Softmax\left(u^{0^T} m_i\right) \\
c_i &= C^0 x^i \quad \forall i \\
o^0 &= \sum_i p_i c_i \\
u^1 &= o^0 + u^0
\end{aligned}
\tag{4.2}
$$

$$
\hat{a} = Softmax\left(W u^1\right) \tag{4.3}
$$

Again, these can easily be extended to multiple layers by repeating equations 4.2, replacing $u^0$ with $u^1$, $u^1$ with $u^2$, $A^0$ with $A^1$, and $C^0$ with $C^1$ before using equation 4.3 to predict the output. In the remainder of this work, we will frequently drop the superscript denoting which hop the embedding matrix belongs to (i.e. $A$ instead of $A^0$) since the same techniques can be applied for any number of hops. Finally, the loss of the model is defined as the cross entropy between the predicted output word $\hat{a}$ and the ground truth answer word.

It has been shown that sharing weights between layers improves the performance of the memory network. Two methods of weight sharing are presented in the original paper. The first is an adjacent sharing scheme where the output embedding for one layer is the input embedding for the layer above (i.e. $A^{k+1} = C^k$). The second is a layer-wise sharing where the input and output embeddings are the same across different layers (i.e. $A^1 = A^2 = ... = A^K$ and $C^1 = C^2 = ... = C^K$). In the original work the adjacent sharing scheme is shown to provide a greater improvement, and so it is used for all experiments in this work. The adjacent sharing scheme also applies for all sentence encoder modifications we use in this work. For example, the weights of all RNN encoders will be shared in the same manner as the embedding matrices. In this work, all experiments are performed with a memory network composed of three memory hops.

The baseline sentence representations used in [36] are constructed as follows. Given a question sentence $q = [q_0, q_1, ..., q_J]$ and a set of factual sentences to be stored in memory $x^i = [x_0^i, x_1^i, ..., x_J^i]$ for all facts $i \in \{0, .., \texttt{Memory\_size}\}$, where all sentences are padded with null symbols to length $J$, and words are represented as one-hot vectors, the sentence representations are defined as:

1. **Bag of Words (BoW) encoder:**

   Embed each word $j$ of each sentence $i$ with an embedding matrix that is trained end-to-end with the rest of the network. Matrix $A$ is used for the 'input' representations, $B$ for the question representation, and $C$ for the 'output' representations. The sentence representation in each case is then obtained by a sum of the word embeddings.

$$u^0 = \sum_j^J Bq_j \qquad m_i = \sum_j^J Ax_j^i \qquad c_i = \sum_j^J Cx_j^i$$

   We note this is identical to the formulation described above, where a bag of words vector (composed of word frequencies) is used for each sentence, and is then multiplied with the respective embedding matrix via matrix-vector multiplication.

2. **Position Encoding (PE):**

   In an attempt to preserve the word-order of the sentences, a position encoding matrix is created, where each column vector corresponds to the position index $j$ of a word in the sentence. These vectors are then multiplied element-wise with each corresponding embedded word of the sentence. This makes the position of words within a sentence affect their embedding. The modified embedded words are again summed to produce the sentence representation:

$$u^0 = \sum_j^J l_j \odot Bq_j \qquad m_i = \sum_j^J l_j \odot Ax_j^i \qquad c_i = \sum_j^J l_j \odot Cx_j^i$$

Where $j$ is again the index of the word in the sentence, and $i$ is the index of the sentence in the memory. The position encoding matrix is then constructed as:

$$l_{kj} = (1 - j/J) - (k/d)(1 - 2j/J)$$

Where $J$ is the number of words in the sentence, $j$ is the index of the current word in the sentence, $d$ is the dimension of the embedding, and $k$ is the index into the embedding dimension. An example of such a position-encoding matrix for a sentence size $J = 5$ and embedding size $d = 10$ is shown below:

$$\begin{bmatrix} 1.72 & 1.36 & 1.00 & 0.64 & 0.28 \\ 1.56 & 1.28 & 1.00 & 0.72 & 0.44 \\ 1.40 & 1.20 & 1.00 & 0.80 & 0.60 \\ 1.24 & 1.12 & 1.00 & 0.88 & 0.76 \\ 1.08 & 1.04 & 1.00 & 0.96 & 0.92 \\ 0.92 & 0.96 & 1.00 & 1.04 & 1.08 \\ 0.76 & 0.88 & 1.00 & 1.12 & 1.24 \\ 0.60 & 0.80 & 1.00 & 1.20 & 1.40 \\ 0.44 & 0.72 & 1.00 & 1.28 & 1.56 \\ 0.28 & 0.64 & 1.00 & 1.36 & 1.72 \end{bmatrix}$$

As can be seen, the middle word of the sentence will be unaffected by this encoding, while the outer words will have their embeddings modified in a symmetric manner. Alternatively, learned position encoding matrices have also been used, and have been shown to achieve similar performance [30].

3. **Temporal Encoding:**

   In addition to the above, in order to capture the relative order of *sentences* within a

story, additional 'time-words' are appended to the end of each sentence. The first sentence receiving 'time-word-1', the second 'time-word-2', and so on. These time words are treated as regular vocabulary words, and thus an embedding vector from the appropriate matrix ($A$ or $C$) is learned for each. This modifies the resulting sentence representation based on the location of the sentence within the story (and has shown to be crucial for achieving high accuracy in our experiments). Temporal encoding is included for all results presented in this work unless stated otherwise.

We have re-implemented the original End-to-End Memory Network model, along with the position encoder, in Tensorflow to use as a baseline. The performance of our model is comparable to that in the original paper. On average, our model achieves 1.1% higher accuracy on the bAbI-1k dataset, while achieving 2.4% lower accuracy on the bAbI-10k dataset. We attribute these differences to the high variability between runs as previously mentioned. The accuracy of our implementation compared with the published results is shown in Tables B.1 and B.2 in the Appendix. Models trained on the bAbI 1-k dataset were trained for 100 epochs, while models trained on the 10k dataset were trained for 20 epochs, following the original matlab implementation. We additionally follow the matlab implementation (found here: https://github.com/facebook/MemNN/tree/master/MemN2N-babi-matlab) for hyperparameter selection.

The sections below describe the modifications we made to this architecture, and specifically the sentence encoders used to compute the embeddings of questions and memory sentences.

## 4.4 GRU Encoder

In an attempt to improve the sentence representations ($u^0, m_i, \& \ c_i$) of the End-to-End Memory Network, and alleviate the need for position encoding, we suggest using a recurrent neural network to process each sentence, taking the final hidden state of the network as the sentence representation. At each step, the network takes in the embedded representation of the word, obtained with the original embedding matrices $B, A, \& \ C$. A visual representation of this architecture is in Figure 3. In our experiments, a GRU was chosen for the RNN. We additionally experimented with LSTM units but found no significant difference beyond increased training time. We find the full 10,000 examples to be necessary to train the RNN encoder, and thus compare with the original MemN2N also trained on 10,000 examples in the results section.

We keep all hyperparameters of the network the same as the original implementation for a fair comparison. These include the embedding dimension, number of memory hops, and memory size. The only modification made is to the optimizer used. Instead of stochastic gradient descent with the annealed learning rate described in [36], we use the Adam optimizer with a learning rate of 0.01 since this is shown to work well for RNN training.

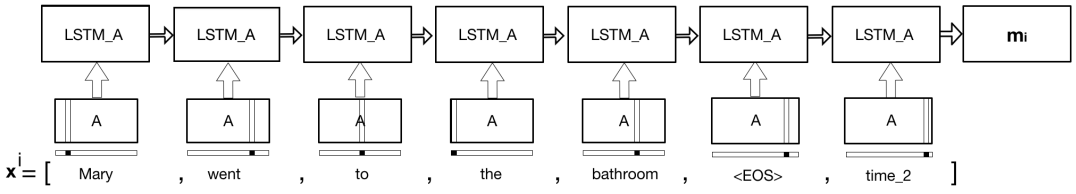


**Figure 4.2**: RNN sentence encoder used with the original End-to-End Memory network. Each word is embedded with the learned embedding matrix $B$, $A$, or $C$ for the query, input, and output representations respectively. The embedded words are then fed to the RNN, LSTM, or GRU cell one at a time, and the final hidden state of the network is taken as the sentence representation $u^0$, $m_i$, $c_i$ for the query, input memories, and output memories respectively.

## 4.5 Bi-Directional GRU Fusion Layer

Motivated by [10], we test the application of a fusion layer on top of the sentence representations. The idea is to use a bi-directional RNN to allow information to flow between sentences of the story – something which otherwise must be handled directly by the internal states and memory hops of the network. The hope is that such a fusion layer will help better represent the sentences in the larger context of the story as a whole, and potentially eliminate the need for a time-encoding. A graphic representation of the fusion layer is shown in figure 4.3, where the positional encoder is identical to that described above, with $w_j^i$ being the embedding of word $j$ of sentence $i$, and $f_i$ being the encoded representation of sentence $i$ (previously referred to as $m_i$ or $c_i$ for the input and output respectively).

As can be seen, the fusion layer implementation is relatively straight forward. To allow for a fair comparison with the standard memory network, we test the application of a bi-directional RNN fusion layer on-top of the postion-encoded bag of words representation of sentences, and compare with the standard end-to-end memory network both with and without time encoding. Given the sentences have a natural ordering, we provide the sentence embeddings in order, one at a time, as input to the fusion layer RNN (in our case a bi-directional GRU). The concatenated hidden states of the fusion layer, for each corresponding input, are taken as the new embedding for the sentence.

## 4.6 Skip-Thought Encoder

To further evaluate the skip-thought encodings, we decided to test the pre-trained Skip-thought encoder as the input-encoder for a memory network. To achieve this, we precomputed the skip-thought vectors for all sentences in the bAbI dataset, and use these embeddings directly, removing the standard input, output, and question encoders for the

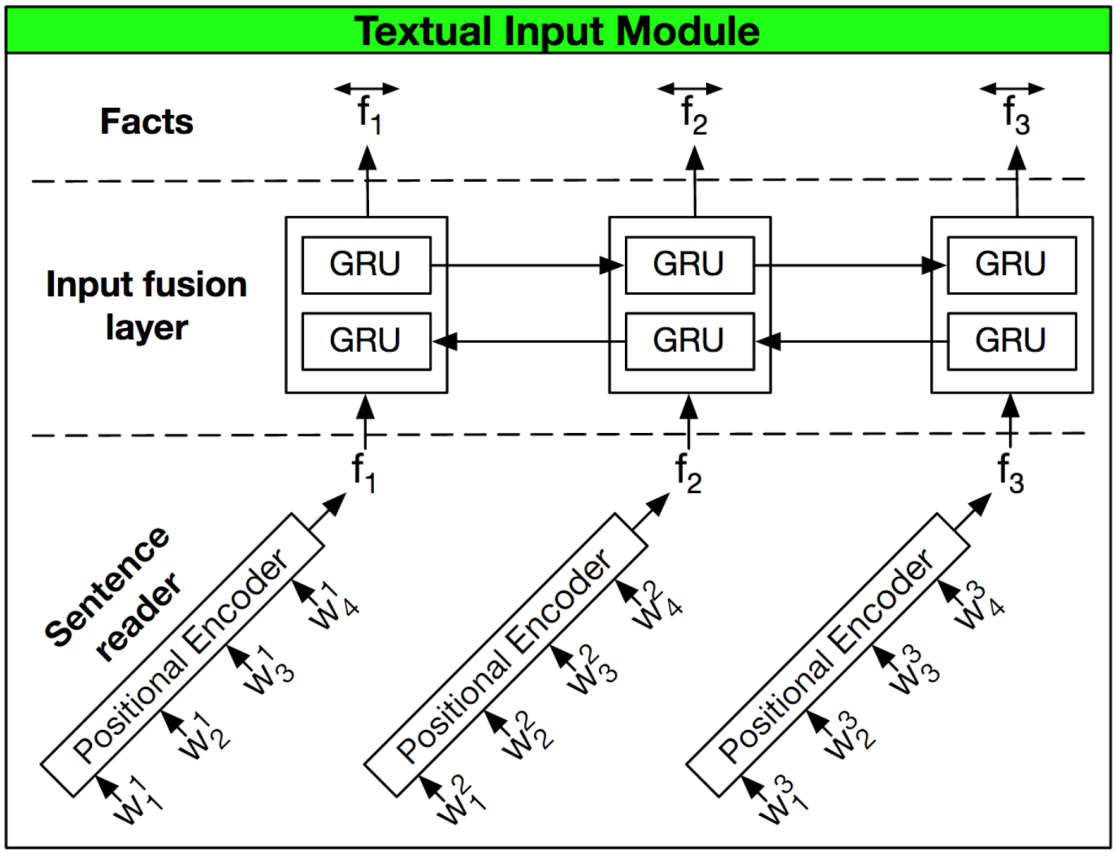


**Figure 4.3**: Fusion layer added to position-encoding allows information to flow between sentences. Figure from [10].

memory network. This removed a significant portion of the trainable parameters of the network, relying mainly on the pre-trained skip-thought encoder. We demonstrate the effect this has on performance of the network in the results section.

For implementation, we use the pre-trained skip-thought weights from the Tensorflow model repository (https://github.com/tensorflow/models/tree/master/skip_thoughts).

## 4.7 Convolutional Encoder

Encouraged by recent results showing the power of convolutional neural networks for sentence classification [45] language modeling [44], and machine translation

[25], we decide to try convolving over words in each sentence to generate the sentence representations. A visual representation of this architecture is shown in Figure 4.4.

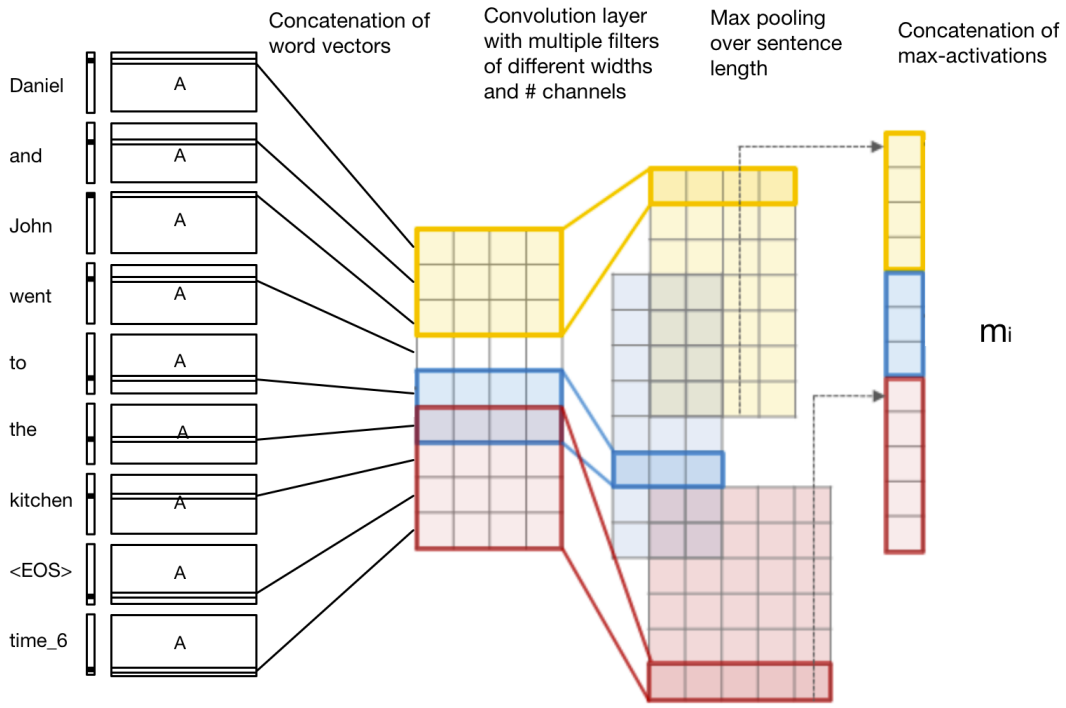


**Figure 4.4**: CNN sentence encoder used with the original End-to-End Memory network. In this example there are 3 filter widths of size 3, 2, and 4 with 4, 3, and 5 channels respectively from top to bottom. The maximum activations of of each channel over the length of the sentence are concatenated together to form the sentence representation. Again the word embedding matrices $B$, $A$, & $C$ are used to initially embed the words for the respective parts of the network. In sharing schemes, the convolutional filter weights are shared in the same manner as the word embeddings.

The basic idea is to stack the word vectors obtained from $B$, $A$, & $C$ into a matrix, and use convolutional filters of size $(N, d)$, where $d$ is the embedding size, and $N$ is the number of words the filter covers. We then max-pool over the sentence length to obtain an scalar feature value for each filter channel. The features from all filters are concatenated together and treated as the resulting sentence representation.

In our experiments, convolutional filter sizes were set as the complete range from 1 to $J$, where $J$ is the number of words in the sentence. We then set the number of

channels for each filter size such that the total number of features obtained at the end of pooling was less than, but close to, 100. For example, for task 1, with $J = 8$ and $d = 20$, we create the following filters [(1, 20), (2, 20), ... , (7, 20), (8, 20)] each with 12 channels, such that the final sentence representation size is $96 = 8 * 12$.

We note that the resulting embedding size of 100 is significantly larger than the size of 20 used for the BoW and RNN encoders, making direct comparison unfair. However, we empirically see little effect on performance from increasing the size of the BoW embedding size, and thus include the results regardless to show the potential for convolutional encoders.

## 4.8 Curriculum Learning

A common method for improving the performance of neural networks, and improving the reliability of optimization, is curriculum learning [47]. The basic idea is to provide simpler examples earlier in training, allowing the network to learn fundamental concepts more quickly, and then increasing the difficulty as the error decreases.

In the case of memory networks, we empirically see that the number of sentences in the memory was a significant contributing factor to how well the network performs (e.g. more sentences require more reasoning during each hop, and greater potential for the network to place large attention weights on the wrong memory sentence). Using this insight, we consider using the number of sentences in memory as a curriculum training strategy, slowly increasing the number of memory sentences during training until the maximum length is reached.

For our experiments, we used a curriculum of three sections, defined as the three quantiles of lengths in the dataset, ensuring that each part of the curriculum had an equivalent number of training examples. We performed curriculum training for the

first 32 epochs, increasing the length every 16 epochs until the maximum length was reached. The model was then trained as normal for the remaining 18 epochs. We perform curriculum training on the bAbI 10k dataset.

## 4.9 Order Testing

Inspired by [49], we decided to use auxiliary prediction tasks to evaluate the modifications of the basic bag of words encoder, such as position encoding, and determine their quantitative impact on the final sentence representation. Specifically, we perform the order-testing, and compare the ability of the bag of words encoder to predict the order of words in the original sentence, with and without the position encoding. We additionally perform order testing on the convolutional and recurrent sentence encoders.

To accomplish order testing, we follow the procedure and hyperparemters stated in [49] explicitly. That is, after training our sentence encoders on the general bAbI question-answer tasks, we freeze the weights of the encoder and embedding layers, and encode all sentences in the bAbI dataset (including question sentences). We additionally randomly extract two words from each sentence without replacement, encode the words with the trained word-embedding layer, and store a binary label representing if the first word comes before the second word or not. The original sentence vector and the two word vectors are all then concatenated into a single vector of length 3 times the embedding size, and fed to a two-layer MLP which is trained to predict the binary order label with a two unit softmax and cross entropy loss. The sentences are split into a training, a held-out validation, and a held-out testing set with the performance reported in the results being the test accuracy of the model with the highest validation accuracy. For all models we use the sentence encoder from the final hop of the network (in our case the third hop) to generate sentence embeddings.

# Chapter 5

# Results and Discussion: Skip-Thought Vectors

This chapter shows the performance of the original skip-thought architecture, as re-trained for this work, compared with the performance of our modifications of the architecture across all of the seven tasks mentioned in the methods section. We additionally include the published results from the uni-directional skip-thought model in [35] as a baseline. We note that none of the results presented here correspond directly to the training objective of the skip-thought model, and thus error may not decrease monotonically with training iterations, resulting in graphs which appear noisy. We address the implications of this in the discussion.

## 5.1 Semantic Similarity

Figure 5.1 shows the mean squared error of the skip-thought encoder variations on the SICK semantic similarity task. The dashed line shows the final MSE reported in the original published work. From the figure, we see the autoencoder variant of the

skip-thought model appears to take significantly longer to reach a comparable accuracy, while the original skip-thought model appears to have an error which decreases in an expected manner consistent with the published results. From these results we see no significant improvement at generating sentence embeddings which are more suited for semantic similarity evaluation for either the variational modification, or the autoencoder modification.

We note that it is possible, given the increased training time for the autoencoder network, that additional training iterations could allow it to reach an MSE equal to that or lower than the original network. However, given the already long training time of two weeks for the standard network, we deem this to be impractical. Additionally, given the advent of sentence encoders such as FastSent which are able to exploit the same signal of the skip-thought encoder with significantly less computational complexity required to train, we do not see significant advantages in pursing further training.

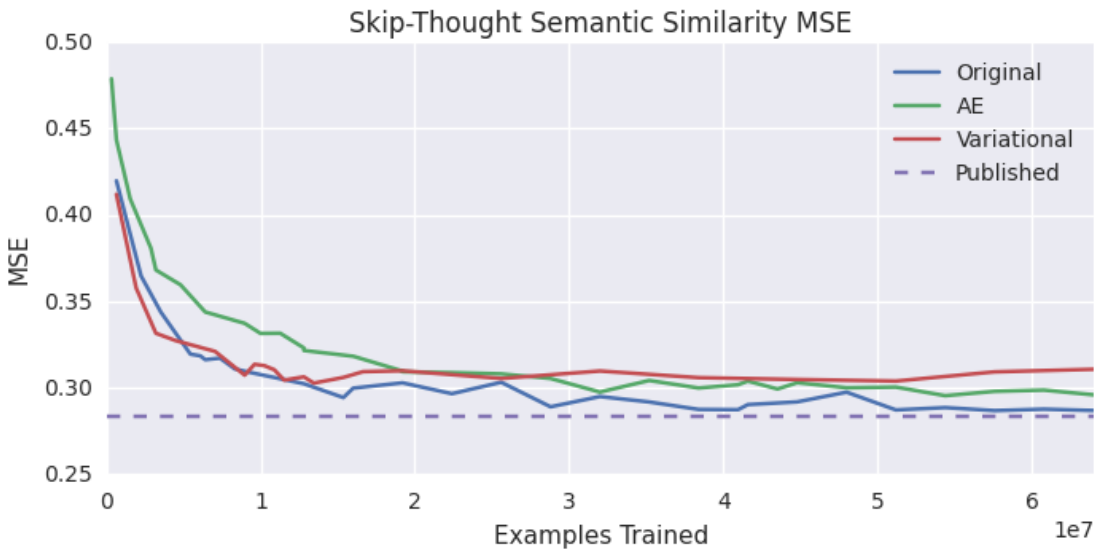


**Figure 5.1**: Mean Squared Error of the three skip-thought architecture variations on the SICK semantic similarity task at various points during training.

## 5.2 Question-type Classification

Figure 5.2 shows the accuracy of the skip-thought encoder variations on the TREC question-type classification task. Unfortunately, for this task, and all remaining tasks, the trained models for the first 1.2 million examples were lost during transfer between remote machines. Therefore, as can be seen, the initial increase in accuracy is missing from these plots. Regardless, we are still able to evaluate the accuracy of the model during the majority of training, and during when it should be converging, so we believe the results are still valuable.

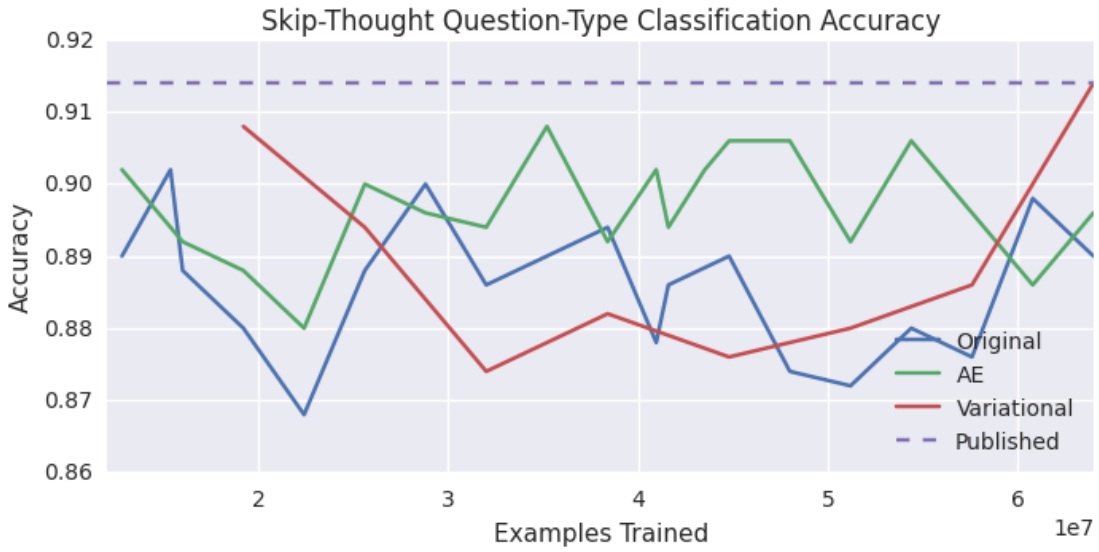


**Figure 5.2**: Accuracy of the three skip-thought architecture variations on the TREC question-type classification task at various points during training.

As can be seen, none of the models, including the original work, appear to surpass the published results. We additionally note that the performance on the TREC dataset appears to not be directly related to the performance of the network as a whole beyond the initial 1.2 million examples of training. We postulate that this is due to the domain mismatch mentioned in [1], where by training on text from fiction novels, the model is not able to significantly improve its representation to generalize to the slight variations

between question types.

Again we see no significant improvement from either architecture modification. We note that given the original architecture does not reach the published performance, it is likely that the skip-thought model has a significant degree of variability between training runs, so results should be considered with this in mind.

## 5.3 Paraphrase Detection

Figure 5.3 shows the accuracy of the skip-thought encoder variations at classifying paraphrased examples from the Microsoft Research Paraphrase Corpus. Here, we see that both the variational skip-thought model, and the original network, perform roughly equivalent to the published result. We again see that performance is also relatively constant while training the skip-thought encoder, suggesting that the skip-thought objective is not well aligned with the paraphrase detection task, and the basic properties of the GRU encoder are sufficient once initialized well. The autoencoder variation appears to perform slightly worse throughout training as well.

## 5.4 Classfication

Through all four of the remaining classification tasks, we see consistent relative performance from the networks. Looking at figures 5.4, 5.5, 5.6, and 5.7, we see that the original skip-thought architecture consistently outperforms our two modifications, with the variational modification coming in second, and appearing to closely follow the original networks performance in some cases.

We believe these results can be explained by a few factors. First, and we believe most importantly, the cost function of the variational skip-thought model is much closer

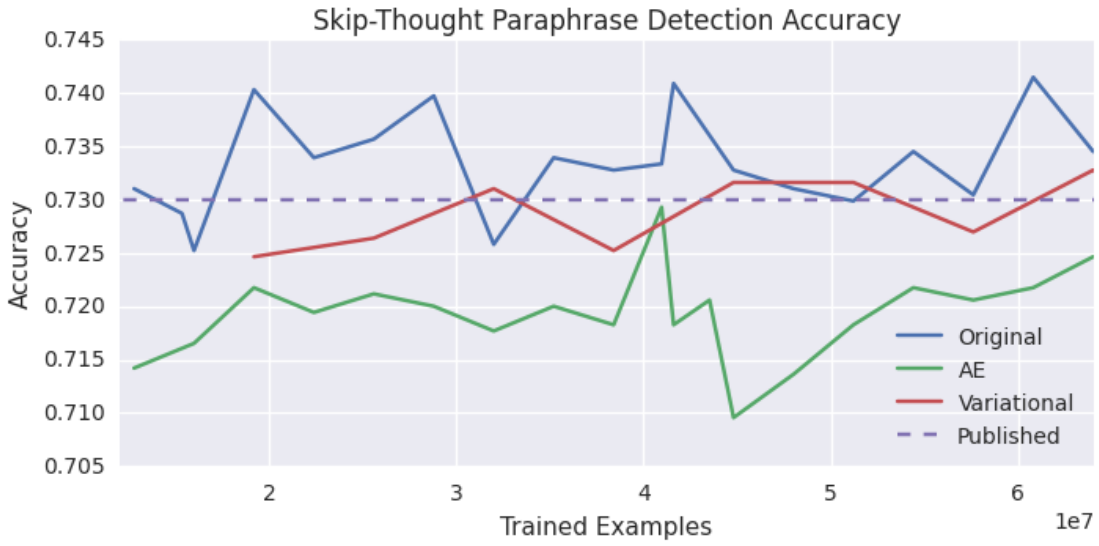


**Figure 5.3**: Accuracy of the three skip-thought architecture variations on the MSRP paraphrase detection task at various points during training.

to that of the original model than is the cost function of the autoencoder model. As noted in the methods, the weight on the KL divergence term of the cost was made very small relative to the reconstruction loss, so it is possible the network is learning to map inputs to a latent space which does not match the prior very well, and therefore does not take advantage of the variational architecture. We did, however, experiment with larger weights on the KL divergence term, and these performed significantly worse, making us believe that the additional variational layer is not well suited to the skip-thought objective. We additionally see the skip-thought autoencoder model performs the worst of the three, only reaching the published results for the opinion polarity classification test and the subjectivity/objectivity classification test.


**Figure 5.4**: Accuracy of the three skip-thought architecture variations on the CR customer review classification task at various points during training.

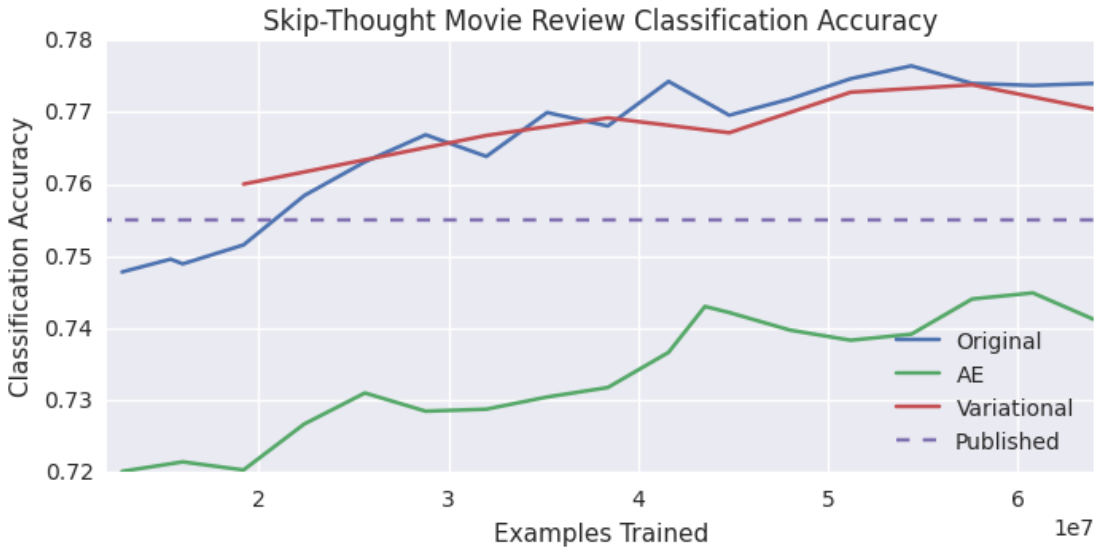


**Figure 5.5**: Accuracy of the three skip-thought architecture variations on the movie review sentiment classification task at various points during training.

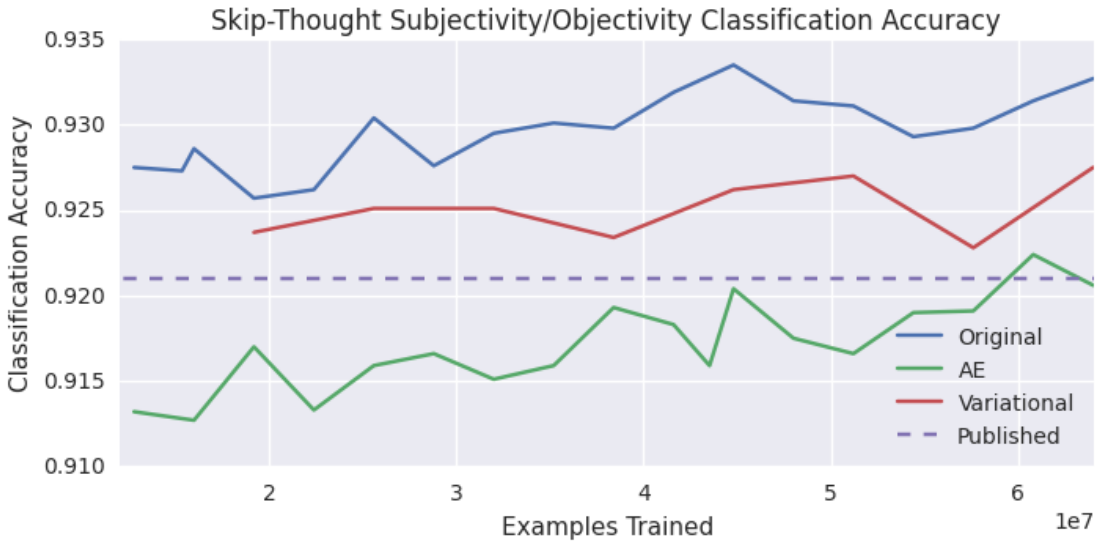


**Figure 5.6**: Accuracy of the three skip-thought architecture variations on the subjectivity/objectivity classification task at various points during training.

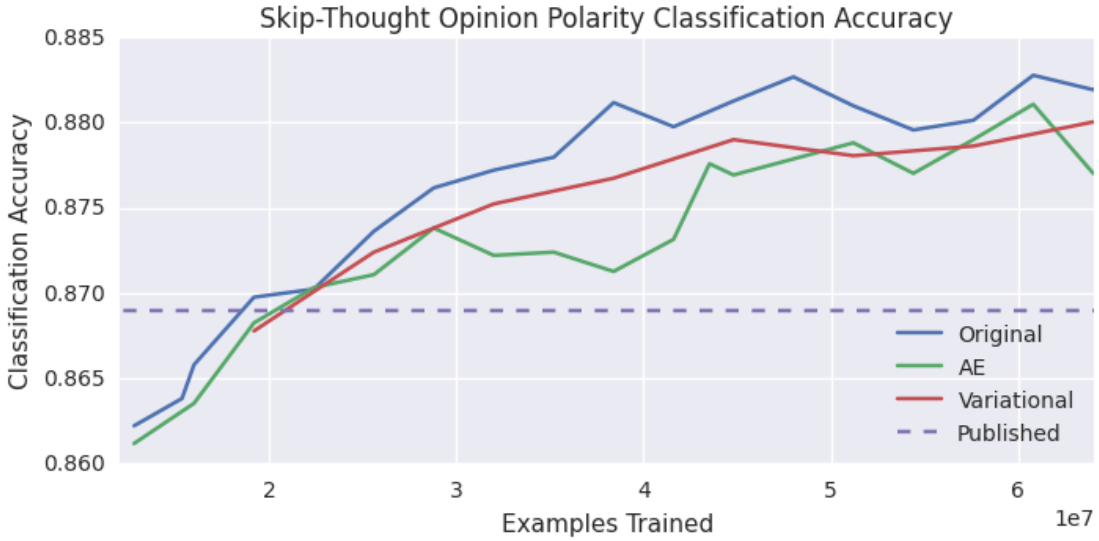


**Figure 5.7**: Accuracy of the three skip-thought architecture variations on the MPQA opinion polarity classification task at various points during training.

# Chapter 6

# Results and Discussion: End-to-End Memory Networks

This chapter shows the results of our end-to-end memory network experiments. The results of each experiment are presented as a table of accuracies of the model trained on each of the 20 bAbI tasks independently. Each table includes an appropriate baseline, and the third column shows the resulting improvement or decrement in performance due to the proposed modification. The mean accuracy over all tasks is shown as the last row of each table, and captures the overall performance of the model. The final section of this chapter contains a table comparing the mean accuracy of all modifications to facilitate evaluation of the proposed approaches.

## 6.1 Position Encoding

As mentioned in the methods section, position encoding is a common technique used to boost the performance of bag of words encoders when more complex encoders are not desired. The performance increase due to position encoding is somewhat significant

overall for the bAbI dataset, with a 7% increase in mean accuracy, but is most pronounced for tasks where word order is crucial for solving the task. These tasks include 3,4,15 and 18 most notably, where accuracy improvements reach as much as 50%. The improvement from position encoding on these tasks is shown in the original work, however we also reduplicated these results with our own implementation and include them in table 6.1 below.

**Table 6.1**: Comparison of question-answering accuracy for End-to-End Memory Network with regular bag of words encoding and position encoding trained on bAbI-1k.

| Task | Regular BoW | Position Enc. | Pos. Enc. - BoW |
|---|---|---|---|
| 1: Single Supporting Fact | 0.999 | 0.997 | -0.002 |
| 2: Two Supporting Facts | 0.820 | 0.852 | 0.032 |
| 3: Three Supporting Facts | 0.350 | 0.572 | 0.221 |
| 4: Two Argument Relations | 0.662 | 0.842 | 0.180 |
| 5: Three Argument Relations | 0.817 | 0.894 | 0.076 |
| 6: Yes/No Questions | 0.918 | 0.918 | 0.0 |
| 7: Counting | 0.778 | 0.830 | 0.052 |
| 8: Lists/Sets | 0.855 | 0.877 | 0.021 |
| 9: Simple Negation | 0.806 | 0.837 | 0.031 |
| 10: Indefinite Knowledge | 0.785 | 0.774 | -0.011 |
| 11: Basic Co-reference | 0.855 | 0.865 | 0.010 |
| 12: Conjunction | 1.0 | 0.996 | -0.004 |
| 13: Compound Co-reference | 0.892 | 0.891 | -0.001 |
| 14: Time Reasoning | 0.874 | 0.920 | 0.045 |
| 15: Basic Deduction | 0.530 | 1.0 | 0.469 |
| 16: Basic Induction | 0.454 | 0.425 | -0.028 |
| 17: Positional Reasoning | 0.5 | 0.504 | 0.004 |
| 18: Size Reasoning | 0.499 | 0.865 | 0.366 |
| 19: Path Finding | 0.100 | 0.099 | -0.001 |
| 20: Agent's Motivation | 1.0 | 1.0 | 0.0 |
| Mean Accuracy | 0.7249 | 0.7983 | 0.0734 |

We note that the above results in table 6.1 are from models trained on the bAbI-1k dataset, and thus do not correspond with our more complex encoder implementations which were trained on the bAbI-10k dataset. An example of the position encoder

performance trained on the bAbI-10k dataset is shown in table 6.5.

Additionally, following [49], we examined the impact of position encoding on the ability of a model to accurately predict the order of words contained in the sentence given the encoding. The results of order-testing with and without position encoding are presented in table 6.2.

**Table 6.2**: Comparison of order-test accuracy for regular bag of words sentence encoder and position encoder both trained on bAbI 1k dataset.

| Task | Regular BoW | Position Enc. | Pos. Enc. - BoW |
|---|---|---|---|
| 1: Single Supporting Fact | 1.0 | 1.0 | 0.0 |
| 2: Two Supporting Facts | 1.0 | 0.999 | -0.001 |
| 3: Three Supporting Facts | 0.998 | 0.998 | 0.0 |
| 4: Two Argument Relations | 0.761 | 0.816 | 0.056 |
| 5: Three Argument Relations | 0.962 | 0.989 | 0.028 |
| 6: Yes/No Questions | 0.996 | 0.999 | 0.003 |
| 7: Counting | 0.970 | 0.981 | 0.01 |
| 8: Lists/Sets | 0.999 | 1.0 | 0.001 |
| 9: Simple Negation | 0.988 | 0.989 | 0.001 |
| 10: Indefinite Knowledge | 0.958 | 0.951 | -0.007 |
| 11: Basic Co-reference | 1.0 | 0.999 | -0.001 |
| 12: Conjunction | 0.964 | 0.966 | 0.002 |
| 13: Compound Co-reference | 0.978 | 0.983 | 0.004 |
| 14: Time Reasoning | 0.861 | 0.959 | 0.098 |
| 15: Basic Deduction | 0.894 | 1.0 | 0.106 |
| 16: Basic Induction | 1.0 | 1.0 | 0.0 |
| 17: Positional Reasoning | 0.713 | 0.753 | 0.04 |
| 18: Size Reasoning | 0.781 | 0.805 | 0.024 |
| 19: Path Finding | 0.757 | 0.828 | 0.071 |
| 20: Agent's Motivation | 0.998 | 0.996 | -0.002 |
| Mean Accuracy | 0.929 | 0.951 | 0.022 |

The first observation is that even without position encoding, the order of words in the bAbI dataset is fairly consistent and predictable for most tasks, so an accuracy of 93% is possible without any position encoding. We believe this is due to the automated nature of sentence generation. However, we see that on the same tasks where position

encoding helps the performance of the main question-answering task the most, it also has a significant positive impact on order-testing accuracy (see Task 5: Three Argument Relations , Task 15: Basic Deduction, and Task 18: Size Reasoning).

Interestingly, we see that position encoding is not perfect, and fails to achieve 100% accuracy on the order tests for a significant number of tasks including 4, 17, 18, and 19. We see that these tasks also have relatively low accuracy on the main question answering task, and thus hypothesize that the lacking order-test performance is due to the networks inability to train successfully for its original task, resulting in poorly trained word vectors.

## 6.2 GRU Encoder

The performance of the GRU encoder is shown in Table 6.3 below. We compare the GRU encoder with our re-implementation of the original position encoder where both models were trained on the bAbI 10k dataset for valid comparison.

From table 6.3, we see that the GRU encoder fails to improve significantly on any of the tasks besides task 17, a task on which position encoding interestingly does not seem to help. This is surprising, but suggests the benefits of using a recurrent encoder are not fully taken advantage of by the memory network architecture. We also believe this implies that the representation of sentences may not be the performance bottleneck of the memory network architecture, an assumption which is backed up by the recent Recurrent Entity Networks [30] which are able to achieve greater than 95% accuracy on all tasks while still using the basic position encoder.

We additionally performed the same experiment as for the position encoding, testing the order-encoding capabilities of the GRU encoder. These results are presented in table 6.4 and compared with the order testing results of the position encoder trained

**Table 6.3**: Accuracy of GRU encoder MemN2N model trained on bAbI-10k compared with position encoding of the original model also trained on bAbI-10k.

| Task | Pos. Enc. | GRU Enc. | GRU - Pos. |
|---|---|---|---|
| 1: Single Supporting Fact | 1.0 | 0.997 | -0.003 |
| 2: Two Supporting Facts | 0.995 | 0.974 | -0.021 |
| 3: Three Supporting Facts | 0.841 | 0.693 | -0.148 |
| 4: Two Argument Relations | 1.0 | 0.998 | -0.002 |
| 5: Three Argument Relations | 0.997 | 0.976 | -0.021 |
| 6: Yes/No Questions | 0.998 | 0.867 | -0.131 |
| 7: Counting | 0.955 | 0.797 | -0.158 |
| 8: Lists/Sets | 0.965 | 0.988 | 0.023 |
| 9: Simple Negation | 0.984 | 0.944 | -0.04 |
| 10: Indefinite Knowledge | 0.968 | 0.738 | -0.23 |
| 11: Basic Co-reference | 0.906 | 0.892 | -0.014 |
| 12: Conjunction | 1.0 | 0.999 | -0.001 |
| 13: Compound Co-reference | 0.944 | 0.942 | -0.002 |
| 14: Time Reasoning | 0.998 | 0.992 | -0.006 |
| 15: Basic Deduction | 1.0 | 1.0 | 0.0 |
| 16: Basic Induction | 0.472 | 0.485 | 0.013 |
| 17: Positional Reasoning | 0.492 | 0.628 | 0.136 |
| 18: Size Reasoning | 0.901 | 0.909 | 0.008 |
| 19: Path Finding | 0.221 | 0.199 | -0.022 |
| 20: Agent's Motivation | 1.0 | 0.995 | -0.005 |
| Mean Accuracy | 0.882 | 0.851 | -0.031 |

on the same bAbI 10k dataset. As can be seen, the GRU encoder does perform better than the bag of words encoder without position encoding, but actually performs slightly worse than the position encoding alone. We believe this partially explains the lack of superior performance of the GRU encoder on the question answering tasks, as reported in table 6.3, but also concede that poor performance on the question answering tasks could equivalently be the cause of poor order testing performance.

**Table 6.4**: Comparison of order-test accuracy for position encoder and GRU encoder both trained on the bAbI 10k dataset.

| Task | Pos. Enc. | GRU Enc. | GRU - Pos. |
|---|---|---|---|
| 1: Single Supporting Fact | 1.0 | 1.0 | 0.0 |
| 2: Two Supporting Facts | 1.0 | 1.0 | 0.0 |
| 3: Three Supporting Facts | 0.998 | 0.998 | -0.0 |
| 4: Two Argument Relations | 0.912 | 0.864 | -0.047 |
| 5: Three Argument Relations | 1.0 | 0.992 | -0.008 |
| 6: Yes/No Questions | 1.0 | 1.0 | 0.0 |
| 7: Counting | 0.999 | 0.99 | -0.01 |
| 8: Lists/Sets | 1.0 | 1.0 | -0.0 |
| 9: Simple Negation | 0.991 | 0.99 | -0.001 |
| 10: Indefinite Knowledge | 0.958 | 0.965 | 0.007 |
| 11: Basic Co-reference | 1.0 | 1.0 | 0.0 |
| 12: Conjunction | 0.968 | 0.967 | -0.001 |
| 13: Compound Co-reference | 0.984 | 0.98 | -0.005 |
| 14: Time Reasoning | 1.0 | 0.902 | -0.098 |
| 15: Basic Deduction | 1.0 | 1.0 | 0.0 |
| 16: Basic Induction | 1.0 | 0.995 | -0.005 |
| 17: Positional Reasoning | 0.741 | 0.717 | -0.024 |
| 18: Size Reasoning | 0.85 | 0.763 | -0.087 |
| 19: Path Finding | 0.844 | 0.763 | -0.081 |
| 20: Agent's Motivation | 1.0 | 1.0 | 0.0 |
| Mean Accuracy | 0.962 | 0.944 | -0.018 |

## 6.3 Convolutional Encoder

Table 6.5 shows the question-answer accuracy of our End-to-End Memory Network with convolutional sentence representations trained on the bAbI 10k dataset compared with that of the position encoding network.

From the table, we see the simple position encoding outperforms the convolutional encoder by a significant margin. We find this surprising given the convolutional encoder is similarly using a weighted sum over words, and is using a larger embedding size as mentioned in the methods. However, due to the apparent proclivity of memory

**Table 6.5**: Accuracy of MemN2N with convolutional encoder trained on bAbI-10k compared with the original MemN2N with position encoding.

| Task | Pos. Enc. | Conv. Enc. | Conv. - Pos. |
|---|---|---|---|
| 1: Single Supporting Fact | 1.0 | 0.97 | -0.03 |
| 2: Two Supporting Facts | 0.995 | 0.749 | -0.246 |
| 3: Three Supporting Facts | 0.841 | 0.645 | -0.196 |
| 4: Two Argument Relations | 1.0 | 0.999 | -0.001 |
| 5: Three Argument Relations | 0.997 | 0.978 | -0.019 |
| 6: Yes/No Questions | 0.998 | 0.827 | -0.171 |
| 7: Counting | 0.955 | 0.863 | -0.092 |
| 8: Lists/Sets | 0.965 | 0.889 | -0.076 |
| 9: Simple Negation | 0.984 | 0.902 | -0.082 |
| 10: Indefinite Knowledge | 0.968 | 0.813 | -0.155 |
| 11: Basic Co-reference | 0.906 | 0.879 | -0.027 |
| 12: Conjunction | 1.0 | 0.973 | -0.027 |
| 13: Compound Co-reference | 0.944 | 0.938 | -0.006 |
| 14: Time Reasoning | 0.998 | 0.833 | -0.165 |
| 15: Basic Deduction | 1.0 | 0.999 | -0.001 |
| 16: Basic Induction | 0.472 | 0.476 | 0.004 |
| 17: Positional Reasoning | 0.492 | 0.684 | 0.192 |
| 18: Size Reasoning | 0.901 | 0.992 | 0.091 |
| 19: Path Finding | 0.221 | 0.216 | -0.005 |
| 20: Agent's Motivation | 1.0 | 1.0 | 0.0 |
| Mean Accuracy | 0.882 | 0.831 | -0.051 |

networks to settle in local minima, we hypothesize that the additional complexity of the convolutional encoder only exacerbates this issue. We believe specifically tasks 3, 7, 10 may be evidence of this.

In table 6.6 we additionally show the results of order testing on the convolutional encoder embeddings, and surprisingly find that it performs the best of the three encoders presented in this work for the End-to-End Memory Network. This is especially surprising given the relatively poor performance of the convolutional encoder on the question answer tasks. This leads us to believe, contrary to our initial assumption, that encoding the order of words accurately is not essential to solving the majority of the bAbI tasks, and for

those where position is necessary, the position encoding does well enough to not be a performance bottleneck. We hesitate to conclude the convolutional encoder is better able to encode position of words in general, given the embedding size of 100 compared with the position encoding embedding size of 20, but we believe this is something that should be studied further in future work.

**Table 6.6**: Comparison of order-test accuracy for position encoder and convolutional encoder both trained on the bAbI 10k dataset.

| Task | Pos. Enc. | Conv. Enc. | Conv. - Pos. |
|---|---|---|---|
| 1: Single Supporting Fact | 1.0 | 1.0 | 0.0 |
| 2: Two Supporting Facts | 1.0 | 1.0 | 0.0 |
| 3: Three Supporting Facts | 0.998 | 0.998 | -0.0 |
| 4: Two Argument Relations | 0.912 | 0.909 | -0.002 |
| 5: Three Argument Relations | 1.0 | 1.0 | 0.0 |
| 6: Yes/No Questions | 1.0 | 1.0 | 0.0 |
| 7: Counting | 0.999 | 1.0 | 0.001 |
| 8: Lists/Sets | 1.0 | 1.0 | 0.0 |
| 9: Simple Negation | 0.991 | 1.0 | 0.009 |
| 10: Indefinite Knowledge | 0.958 | 0.981 | 0.022 |
| 11: Basic Co-reference | 1.0 | 1.0 | 0.0 |
| 12: Conjunction | 0.968 | 1.0 | 0.032 |
| 13: Compound Co-reference | 0.984 | 1.0 | 0.016 |
| 14: Time Reasoning | 1.0 | 1.0 | -0.0 |
| 15: Basic Deduction | 1.0 | 1.0 | 0.0 |
| 16: Basic Induction | 1.0 | 1.0 | 0.0 |
| 17: Positional Reasoning | 0.741 | 0.848 | 0.108 |
| 18: Size Reasoning | 0.85 | 0.873 | 0.023 |
| 19: Path Finding | 0.844 | 0.886 | 0.042 |
| 20: Agent's Motivation | 1.0 | 1.0 | 0.0 |
| Mean Accuracy | 0.962 | 0.975 | 0.013 |

## 6.4 Bi-Directional GRU Fusion layer

Table 6.7 shows the results of attempting to use the fusion layer as a replacement for time encoding. The table compares the fusion layer, which uses no time encoding,

with the position encoder additionally without time encoding. By comparison with tables 6.5 and 6.3, which both use the time encoding, we see that the fusion layer does not entirely replace the need for a time encoding, as evidenced by its inferior performance. However, as evidenced by table 6.7 below, we see that the fusion layer is able to account for a significant fraction of the performance gains from including a time encoding. We believe this is simply due to the fact that, with the fusion layer, information is able to flow between sentences, so sentences which overrule other sentences, or make information from other sentences irrelevant in their context, are able to influence these representations in a manner useful to the overall network.

**Table 6.7**: Accuracy of the MemN2N without time encoding compared with the fusion layer modification, also without time encoding, both trained on the bAbI 10k dataset.

| Task | No Time Enc. | Fusion | Fusion - No Time |
|---|---|---|---|
| 1: Single Supporting Fact | 0.658 | 1.0 | 0.342 |
| 2: Two Supporting Facts | 0.51 | 0.902 | 0.392 |
| 3: Three Supporting Facts | 0.419 | 0.771 | 0.352 |
| 4: Two Argument Relations | 1.0 | 1.0 | 0.0 |
| 5: Three Argument Relations | 0.886 | 0.99 | 0.104 |
| 6: Yes/No Questions | 0.902 | 0.502 | -0.4 |
| 7: Counting | 0.929 | 0.971 | 0.042 |
| 8: Lists/Sets | 0.954 | 0.995 | 0.041 |
| 9: Simple Negation | 0.916 | 0.627 | -0.289 |
| 10: Indefinite Knowledge | 0.842 | 0.94 | 0.098 |
| 11: Basic Co-reference | 0.729 | 1.0 | 0.271 |
| 12: Conjunction | 0.55 | 1.0 | 0.45 |
| 13: Compound Co-reference | 0.597 | 1.0 | 0.403 |
| 14: Time Reasoning | 1.0 | 1.0 | 0.0 |
| 15: Basic Deduction | 1.0 | 1.0 | 0.0 |
| 16: Basic Induction | 0.466 | 0.479 | 0.013 |
| 17: Positional Reasoning | 0.5 | 0.51 | 0.01 |
| 18: Size Reasoning | 0.919 | 0.912 | -0.007 |
| 19: Path Finding | 0.275 | 0.116 | -0.159 |
| 20: Agent's Motivation | 1.0 | 1.0 | 0.0 |
| Mean Accuracy | 0.753 | 0.836 | 0.083 |

## 6.5 Curriculum Learning

Table 6.8 shows the results of curriculum training for the original position encoding network trained on the babi 10k dataset. We see the overall final performance results are slightly improved, but not significantly. Interestingly we see the greatest improvements on the tasks with the lowest performance for the original network, namely tasks 17 and 19. We believe this is because these are the most complex tasks in terms of reasoning ability requirements over the sentences, so allowing the network to learn simpler reasoning techniques initially appears to help. We had hoped that the curriculum

**Table 6.8**: Accuracy of MemN2N with position encoding trained on bAbI-10k with and without using a story length based curriculum.

| Task | Orig. | Curric. | Curric. - Orig. |
|---|---|---|---|
| 1: Single Supporting Fact | 1.0 | 1.0 | 0.0 |
| 2: Two Supporting Facts | 0.995 | 0.977 | -0.018 |
| 3: Three Supporting Facts | 0.841 | 0.839 | -0.002 |
| 4: Two Argument Relations | 1.0 | 1.0 | 0.0 |
| 5: Three Argument Relations | 0.997 | 0.993 | -0.004 |
| 6: Yes/No Questions | 0.998 | 0.999 | 0.001 |
| 7: Counting | 0.955 | 0.953 | -0.002 |
| 8: Lists/Sets | 0.965 | 0.971 | 0.006 |
| 9: Simple Negation | 0.984 | 0.98 | -0.004 |
| 10: Indefinite Knowledge | 0.968 | 0.977 | 0.009 |
| 11: Basic Co-reference | 0.906 | 0.902 | -0.004 |
| 12: Conjunction | 1.0 | 1.0 | 0.0 |
| 13: Compound Co-reference | 0.944 | 0.94 | -0.004 |
| 14: Time Reasoning | 0.998 | 0.995 | -0.003 |
| 15: Basic Deduction | 1.0 | 1.0 | 0.0 |
| 16: Basic Induction | 0.472 | 0.462 | -0.01 |
| 17: Positional Reasoning | 0.492 | 0.515 | 0.023 |
| 18: Size Reasoning | 0.901 | 0.894 | -0.007 |
| 19: Path Finding | 0.221 | 0.325 | 0.104 |
| 20: Agent's Motivation | 1.0 | 1.0 | 0.0 |
| Mean Accuracy | 0.882 | 0.886 | 0.004 |

would help the optimization procedure of the memory network to avoid suboptimal local minima, however from examining the 10 runs of training, it appears this was not what happened. Specifically, looking at task 2, we see nine of the ten random initialization resulted in a final accuracy of less than 0.4, with only one of ten achieving the reported 0.977. We believe this speaks to the inherent tendency of the network as a whole to settle in local minima, and explains some of the challenges encountered by the modifications in this work.

## 6.6 Skip-Thought Encoder

Table 6.9 shows the results of a memory network with skip-thought sentence encodings trained on the bAbI 1k dataset. After initial testing, it was seen that due to the limited number of trainable parameters, the bAbI 10k dataset provided no significant benefit. Looking at Table 6.9 , we see the skip-thought encoder performs significantly worse than the original position encoded bag of words on almost all tasks. We postulate this is again due to the limited nature of the bAbI tasks, and the significantly different training corpus used to train the skip-thought encoder. We do see, however, that performance is increased for task 18, the size reasoning task. This is surprising given how poorly the network performs on the majority of the other tasks, but interesting given task 18 seems to require the least amount of memory access, and the most 'common sense'. Looking at examples from the dataset, we see it is composed of question such as: 'Is the chocolate bigger than the suitcase' – 'No', 'Is the suitcase bigger than the box of chocolates' – 'Yes'. Almost all task 18 questions in the dataset seem answerable without reading the provided story, and sizes seem consistent with reality, potentially explaining why the skip-thought encoder was able to perform well here.

**Table 6.9**: Accuracy of Skip-Thought-MemN2N model trained on bAbI-1k.

| Task | Original MemN2N | ST-MemN2N | ST - Original |
|---|---|---|---|
| 1: Single Supporting Fact | 0.999 | 0.441 | -0.558 |
| 2: Two Supporting Facts | 0.784 | 0.182 | -0.602 |
| 3: Three Supporting Facts | 0.358 | 0.237 | -0.121 |
| 4: Two Argument Relations | 0.962 | 0.763 | -0.199 |
| 5: Three Argument Relations | 0.859 | 0.485 | -0.374 |
| 6: Yes/No Questions | 0.921 | 0.498 | -0.423 |
| 7: Counting | 0.784 | 0.613 | -0.171 |
| 8: Lists/Sets | 0.874 | 0.415 | -0.459 |
| 9: Simple Negation | 0.767 | 0.634 | -0.133 |
| 10: Indefinite Knowledge | 0.826 | 0.483 | -0.343 |
| 11: Basic Coreference | 0.957 | 0.445 | -0.512 |
| 12: Conjunction | 0.997 | 0.393 | -0.604 |
| 13: Compound Coreference | 0.901 | 0.514 | -0.387 |
| 14: Time Reasoning | 0.982 | 0.396 | -0.586 |
| 15: Basic Deduction | 1 | 0.544 | -0.456 |
| 16: Basic Induction | 0.479 | 0.481 | 0.002 |
| 17: Positional Reasoning | 0.499 | 0.495 | -0.004 |
| 18: Size Reasoning | 0.864 | 0.925 | 0.061 |
| 19: Path Finding | 0.126 | 0.083 | -0.043 |
| 20: Agent's Motivation | 1 | 0.889 | -0.111 |
| Mean Accuracy | 0.797 | 0.496 | -0.301 |

## 6.7 Overall Comparison

Table 6.10 shows the mean accuracy reported in each of the above experiments performed on the bAbI-10k dataset.

**Table 6.10**: Mean accuracy of all End-to-End Memory Network experiments trained on bAbI-10k. Rows show standard question answering accuracy and order-test accuracy when available. The highest accuracy in each row is bolded.

| Test Case | Pos. Enc. | GRU Enc. | Conv. Enc. | Fusion BoW | Curric. |
|---|---|---|---|---|---|
| QA | 0.882 | 0.851 | 0.831 | 0.836 | **0.886** |
| Order-test | 0.962 | 0.944 | **0.975** | - | - |

# Chapter 7

# Related Work

In this section we describe work specifically related to the analysis of sentence representations, the skip-thought architecture, and memory networks.

## 7.1 Sentence Representations

The study of sentence representations for natural language processing is a broad area of active research. Roughly, sentence encoders can be divided into two groups: those which are trained on structured sets of labeled examples to achieve a supervised training objective, and those which are trained on naturally occurring unlabeled text to maximize a task-agnostic objective. In this work, we examine both groups. We study supervised encoders as trained in the service of solving the memory network tasks, and unsupervised encoders as trained though the skip-thought architecture.

Directly related to this work, much research recently has focused on improving unsupervised representations of sentences, including improving the skip-thought architecture. In [15], the authors add additional supervised tasks to the skip-thought training objective, showing it improves performance on paraphrase detection, natural language inference, and semantic relatedness. Although this differs from the goal of

entirely unsupervised training, it shows how embeddings can be improved with additional information. Similarly, by adding unsupervised pretraining tasks to otherwise supervised models, specifically language model pretraining for sequence to sequence translation, Ramachandran et al. [33] were able to show significant acceleration of training, as well as improved generalization capability.

Other sentence encoders besides recurrent neural networks have been shown to perform well for classification tasks. These including convolutional neural network encoders at the word level [45], as well as at the character level [44], "A simple but tough-to-beat baseline for sentence embeddings" [38], and FastSent [16]. The FastSent encoder is of particular relevance to this work since it exploits the same signal as the skip-thought network, but in a more computationally efficient manner. This is achieved by mimicking the log-linear training strategy used for the skip-gram encoder. By replacing the RNN encoder of the skip-thought model with a bag of words encoder, and then trying to assign high probability to words in adjacent sentences through a softmax function, FastSent is able to avoid the computationally expensive recurrent decoders, yet still achieves high accuracy on many tasks with a fraction of the training time. These works are not directly explored in this work, but inspiration is drawn from the word-level convolutional encoder for our memory network experiments.

Recently, work has been done to explore the capabilities of sentence representations in a more systematic manner though auxiliary tasks [49]. These auxiliary tasks include predicting, from the sentence encoding, sequence length, word content, and word order of the encoded sentence. Our work on order testing of memory network encoders is directly inspired by this, and we believe the tools presented here to be valuable for providing insights into complex embedding procedures. Our work differs from [49] by analyzing convolutional encoders, as well as the position encoded bag of words, two models which were not studied in the original work.

As mentioned in the background, the work on variational auto encoders applied to sequence to sequence models [37] is the inspiration for our variational augmentation of the skip-thought architecture. Our work differs from previous works by applying it directly to the skip-thought architecture, rather trying to reconstruct the original input as a traditional autoencoder.

Finally, as the final version of this thesis was being prepared, novel work in [25] used an entirely convolutional neural network to achieve state-of-the-art results on machine translation, while simultaneously decreasing training time by an order of magnitude. This encourages future work with convolutional networks for natural language tasks, and motivates our convolutional sentence embeddings explored in the memory networks section.

## 7.2 Question Answering

Although the bAbI question-answer dataset, described in Section 5.1, was released relatively recently in 2015, a significant number of novel model architectures have been proposed specifically for the task. Some of these include Dynamic Memory Networks [6] and Recurrent Entity Networks [30].

Two versions of the Dynamic Memory Network have been published. The first modifies the structure of the original memory network by using a single recurrent neural network as an encoder applied over all sentences [6]. The encoding for each sentence is taken as the hidden state of the network at the end of each sentence. They also modify the attention mechanism by which the memories are retrieved. Instead of simply using a softmax over the dot products of the question with each memory, they learn a two-layer MLP with input being engineered features from the question, memory, and current model hidden state. Our work with the recurrent encoder differs from this by using separate

RNN encoder weights for each hop, and sharing them in an adjacent fashion as mentioned in [36]. Additionally, we attempt to use the original attention mechanism (softmax of dot products), believing it helps show the true representational capacity of the embeddings, without using engineered features. The second dynamic memory network [10] does away with the recurrent encoder, and returns to using a bag of words encoder with a learnable position encoding.

Recurrent Entity Networks [30], the most recent work in the space, are the first class of models to achieve greater than 95% accuracy on all 20 of the bAbI question answer tasks. The REN model works by having a bank of RNN's which share weights but have independent hidden states. Each network is fed the same input, but also fed a separate trainable key vector, and uses the combination of key and input to determine if information from the current sentence should be remembered by the given network (i.e. added to its hidden state). In the end, Henaff et al. show each RNN is able to track a specific entity in dataset, and store the necessary information in its hidden state. Although this network is shown to solve all of the bAbI tasks, its architecture seems engineered specifically for the bAbI tasks, making it difficult to generalize. Additionally, they continue to use the trained position encoding, as in [10], making the work less similar to this study.

Another class of models, Memory Augmented Neural Networks, such as the Neural Turing Machine [2] and Differentiable Neural Computer [3], have also been applied to the bAbI dataset with slightly underwhelming results. Instead of using specific sentence encoders, they learn to write arbitrary inputs to their memory, and read from it to solve the posed tasks. Although their architectures are much more general, they appear to be less appropriate for natural language comprehension and question answering than their specifically engineered counterparts.

# Chapter 8

# Conclusion

From the skip-thought results, we see that the autoencoder regularization and variational layer additions are unnecessary for learning a generalizable sequence encoder. In hindsight, we realize that, given the massive amount of training data in the BookCorpus dataset, it is unlikely that the network would overfit sufficiently to benefit from this type of strict regularization, resulting mainly in the slower training demonstrated in our results. Similarly, we believe the large number of training examples help the skip-thought encoder to have a fairly continuous representational space, and thus the variational layer addition provides no real benefits. We do believe, however, that there is significant work to be done in the space of unsupervised sentence representations including those with autoencoder and variational elements, but the benefits of these elements may be more apparent when less training data is present, or when the models are able to be trained and tuned more quickly.

From the End-to-End Memory Network results, we see the general trend that more complex sentence encoders do not necessarily improve the performance of the network as a whole. Although certain encoders, such as the convolutional encoder, are better able to encode the order of words within a sentence, this has little to no impact

on the final network question answering performance. This finding is backed up by the recent Recurrent Entity Network which is able to achieve greater than 95% accuracy on all tasks while still using a simple bag of words with position encoding. These results emphasize the importance of the setting in which supervised encoders are trained, and also back up the assumption that the sentence representations used in the End-to-End Memory Network are indeed not the performance bottleneck. Furthermore, through the application of the generally determined powerful skip-thought encoder to the memory network tasks, and the resulting poor performance, we see the importance of matching the training dataset of an encoder to the desired testing regime.

We additionally suggest future sentence encoder comparisons and analysis be performed with networks with more stable optimization routines. The fluctuation of memory network performance, leading to the requirement that training be performed with 10 separate random initializations, made comparison of network architecture modifications extremely challenging, and we believe may have hid some of the true performance differences. We believe improvements to the stability of memory network training could be a promising direction for future work.

Overall, we find the quality and generalizability of sentence representations depends on a a variety of factors beyond the encoder itself. Largely, the quality of representations depends on the dataset with which the encompassing algorithm is trained, and the task for which the encodings are being trained to solve. As noticed in prior work [1], we re-emphasize the importance of the contents of the training set, and how this relates to the intended use of the encoder, suggesting fiction novels may not be the most general of natural language, especially for question answering.

Finally, through significant comparisons and auxiliary task analysis, we find the position encoded bag of words to be a successful and strong baseline for models which desire low computational complexity while maintaining a representation of word order in

the encoded sentence. The position encoder's relatively satisfying performance across the board continue to make it a viable alternative when computational complexity is a concern.

# Appendix A

# Examples of bAbI Tasks

**Task 1: Single Supporting Fact**
Mary went to the bathroom.
John moved to the hallway.
Mary travelled to the office.
Where is Mary? A:office

**Task 2: Two Supporting Facts**
John is in the playground.
John picked up the football.
Bob went to the kitchen.
Where is the football? A:playground

**Task 3: Three Supporting Facts**
John picked up the apple.
John went to the office.
John went to the kitchen.
John dropped the apple.
Where was the apple before the kitchen? A:office

**Task 4: Two Argument Relations**
The office is north of the bedroom.
The bedroom is north of the bathroom.
The kitchen is west of the garden.
What is north of the bedroom? A: office
What is the bedroom north of? A: bathroom

**Task 5: Three Argument Relations**
Mary gave the cake to Fred.
Fred gave the cake to Bill.
Jeff was given the milk by Bill.
Who gave the cake to Fred? A: Mary
Who did Fred give the cake to? A: Bill

**Task 6: Yes/No Questions**
John moved to the playground.
Daniel went to the bathroom.
John went back to the hallway.
Is John in the playground? A:no
Is Daniel in the bathroom? A:yes

**Task 7: Counting**
Daniel picked up the football.
Daniel dropped the football.
Daniel got the milk.
Daniel took the apple.
How many objects is Daniel holding? A: two

**Task 8: Lists/Sets**
Daniel picks up the football.
Daniel drops the newspaper.
Daniel picks up the milk.
John took the apple.
What is Daniel holding? milk, football

**Task 9: Simple Negation**
Sandra travelled to the office.
Fred is no longer in the office.
Is Fred in the office? A:no
Is Sandra in the office? A:yes

**Task 10: Indefinite Knowledge**
John is either in the classroom or the playground.
Sandra is in the garden.
Is John in the classroom? A:maybe
Is John in the office? A:no

**Figure A.1**: Sample stories, questions and answers (presented in red) for tasks 1 through 10. Answers composed of multiple words are treated as individual vocabulary words (i.e. all possible multi-word combinations are represented as independent words in the output vocabulary.). Figure from [22].

**Task 11: Basic Coreference**
Daniel was in the kitchen.
Then he went to the studio.
Sandra was in the office.
Where is Daniel? A:studio

**Task 12: Conjunction**
Mary and Jeff went to the kitchen.
Then Jeff went to the park.
Where is Mary? A: kitchen
Where is Jeff? A: park

**Task 13: Compound Coreference**
Daniel and Sandra journeyed to the office.
Then they went to the garden.
Sandra and John travelled to the kitchen.
After that they moved to the hallway.
Where is Daniel? A: garden

**Task 14: Time Reasoning**
In the afternoon Julie went to the park.
Yesterday Julie was at school.
Julie went to the cinema this evening.
Where did Julie go after the park? A:cinema
Where was Julie before the park? A:school

**Task 15: Basic Deduction**
Sheep are afraid of wolves.
Cats are afraid of dogs.
Mice are afraid of cats.
Gertrude is a sheep.
What is Gertrude afraid of? A:wolves

**Task 16: Basic Induction**
Lily is a swan.
Lily is white.
Bernhard is green.
Greg is a swan.
What color is Greg? A:white

**Task 17: Positional Reasoning**
The triangle is to the right of the blue square.
The red square is on top of the blue square.
The red sphere is to the right of the blue square.
Is the red sphere to the right of the blue square? A:yes
Is the red square to the left of the triangle? A:yes

**Task 18: Size Reasoning**
The football fits in the suitcase.
The suitcase fits in the cupboard.
The box is smaller than the football.
Will the box fit in the suitcase? A:yes
Will the cupboard fit in the box? A:no

**Task 19: Path Finding**
The kitchen is north of the hallway.
The bathroom is west of the bedroom.
The den is east of the hallway.
The office is south of the bedroom.
How do you go from den to kitchen? A: west, north
How do you go from office to bathroom? A: north, west

**Task 20: Agent's Motivations**
John is hungry.
John goes to the kitchen.
John grabbed the apple there.
Daniel is hungry.
Where does Daniel go? A:kitchen
Why did John go to the kitchen? A:hungry

**Figure A.2**: Sample stories, questions and answers (presented in red) for tasks 11 through 20. Figure from [22].

# Appendix B

# Additional Memory Network Results

Below we present additional results including comparisons of our reimplementations of the End-to-End memory network with the original published results.

**Table B.1**: Accuracy of our baseline implementation of the End-to-End Memory network with position encoding compared with the original published result trained on bAbI-1k. We see that our implementation actually achieves 1% mean increase in accuracy. This is likely due to the large amount of variability between runs which is only partially mitigated by the multiple random initializations.

| Task | Original | Ours | Ours - Original |
|---|---|---|---|
| 1: Single Supporting Fact | 0.999 | 1 | 0.001 |
| 2: Two Supporting Facts | 0.784 | 0.83 | 0.046 |
| 3: Three Supporting Facts | 0.358 | 0.54 | 0.182 |
| 4: Two Argument Relations | 0.962 | 0.98 | 0.018 |
| 5: Three Argument Relations | 0.859 | 0.87 | 0.011 |
| 6: Yes/No Questions | 0.921 | 0.92 | -0.001 |
| 7: Counting | 0.784 | 0.84 | 0.056 |
| 8: Lists/Sets | 0.874 | 0.86 | -0.014 |
| 9: Simple Negation | 0.767 | 0.9 | 0.133 |
| 10: Indefinite Knowledge | 0.826 | 0.78 | -0.046 |
| 11: Basic Coreference | 0.957 | 0.84 | -0.117 |
| 12: Conjunction | 0.997 | 1 | 0.003 |
| 13: Compound Coreference | 0.901 | 0.9 | -0.001 |
| 14: Time Reasoning | 0.982 | 0.93 | -0.052 |
| 15: Basic Deduction | 1 | 1 | 0 |
| 16: Basic Induction | 0.479 | 0.44 | -0.039 |
| 17: Positional Reasoning | 0.499 | 0.52 | 0.021 |
| 18: Size Reasoning | 0.864 | 0.88 | 0.016 |
| 19: Path Finding | 0.126 | 0.13 | 0.004 |
| 20: Agent's Motivation | 1 | 1 | 0 |
| Mean Accuracy | 0.79695 | 0.808 | 0.01105 |

**Table B.2**: Accuracy of published MemN2N model with position encoding trained on bAbI-10k compared with our re-implementation trained on bAbI-10k. We see that our implementation has a mean accuracy of 2.4% lower than the published results. For this reason, we compare our sentence encoder modifications in this work with our re-implementation to ensure a consistent baseline.

| Task | Original 10k | Ours 10k | Ours - Original |
|---|---|---|---|
| 1: Single Supporting Fact | 1.0 | 1.0 | 0.0 |
| 2: Two Supporting Facts | 0.996 | 0.995 | -0.001 |
| 3: Three Supporting Facts | 0.874 | 0.841 | -0.033 |
| 4: Two Argument Relations | 1.0 | 1.0 | 0.0 |
| 5: Three Argument Relations | 0.992 | 0.997 | 0.005 |
| 6: Yes/No Questions | 0.998 | 0.998 | 0.0 |
| 7: Counting | 0.943 | 0.955 | 0.012 |
| 8: Lists/Sets | 0.976 | 0.965 | -0.011 |
| 9: Simple Negation | 0.987 | 0.984 | -0.003 |
| 10: Indefinite Knowledge | 0.983 | 0.968 | -0.015 |
| 11: Basic Co-reference | 1.0 | 0.906 | -0.094 |
| 12: Conjunction | 1.0 | 1.0 | 0.0 |
| 13: Compound Co-reference | 0.999 | 0.944 | -0.055 |
| 14: Time Reasoning | 0.998 | 0.998 | 0.0 |
| 15: Basic Deduction | 1.0 | 1.0 | 0.0 |
| 16: Basic Induction | 0.514 | 0.472 | -0.042 |
| 17: Positional Reasoning | 0.597 | 0.492 | -0.105 |
| 18: Size Reasoning | 0.926 | 0.901 | -0.025 |
| 19: Path Finding | 0.334 | 0.221 | -0.113 |
| 20: Agent's Motivation | 1.0 | 1.0 | 0.0 |
| Mean Accuracy | 0.906 | 0.882 | -0.024 |

# Bibliography


[1] Alec Radford, Rafal Jozefowicz, and Ilya Sutskever. Learning to Generate Reviews and Discovering Sentiment. *arXiv preprint arXiv:1704.01444*, 2017

[2] Alex Graves, Greg Wayne and Ivo Danihelka. Neural Turing Machines. *NIPS*, 2014

[3] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwi?ska, Sergio Gmez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, Adri Puigdomnech Badia, Karl Moritz Hermann, Yori Zwols, Georg Ostrovski, Adam Cain, Helen King, Christopher Summerfield, Phil Blunsom, Koray Kavukcuoglu and Demis Hassabis. Hybrid computing using a neural network with dynamic external memory. *Nature*, 2016

[4] Andrew M. Dai and Quoc V. Le. Semi-supervised sequence learning. *NIPS*, 2015.

[5] Andriy Mnih and Koray Kavukcuoglu. Learning word embeddings efficiently with noise-contrastive estimation. *NIPS*, 2013

[6] Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. Ask Me Anything: Dynamic Memory Networks for Natural Language Processing. *arXiv preprint arXiv:1506.07285*, 2015

[7] Bill Dolan, Chris Quirk, and Chris Brockett. Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources. *In Proceedings of the 20th international conference on Computational Linguistics*, 2004

[8] Bo Pang and Lillian Lee. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. *ACL*, 2004

[9] Bo Pang and Lillian Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. *ACL*, 2005

[10] Caiming Xiong, Stephen Merity, Richard Socher. Dynamic Memory Networks for Visual and Textual Question Answering. *CoRR*, 2016

[11] Chris Olah. ”Understanding LSTM Networks.” Blog post. Published August 2015, Accessed June 2017.

[12] Daniel Jurafsky and James H. Martin. ”Semantics with Dense Vectors”. Web. *Speech and Language Processing*, Nov. 2016.

[13] Danilo J. Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *ICML*, 2014

[14] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *ICLR*, 2015.

[15] Eleni Triantafillou, Jamie Ryan Kiros, Raquel Urtasun, Richard Zemel. Towards Generalizable Sentence Embeddings. *ACL*, 2016

[16] Felix Hill, Kyunghyun Cho, and Anna Korhonen. Learning Distributed Representations of Sentences from Unlabelled Data. *arXiv preprint arXiv:1602.03483*, 2016

[17] Geoffrey E. Hinton. Learning Distributed Representations of Concepts. *Proceedings of the Cognitive Science Society*, 1986

[18] Hinrich Schutze. Word Space. *NIPS*, 1993

[19] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *NIPS*, 2014

[20] Janyce Wiebe, Theresa Wilson, and Claire Cardie. Annotating expressions of opinions and emotions in language. *Language resources and evaluation*, 2005

[21] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory Networks. *CoRR*, 2014

[22] Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M. Rush, Bart van Merrinboer, Armand Joulin, and Tomas Mikolov. Towards AI-complete question answering: A set of prerequisite toy tasks. *arXiv preprint: 1502.05698*, 2015

[23] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. GloVe: Global Vectors for Word Representation. *EMNLP*, 2014

[24] Jeffrey L. Elman. Finding Structure in Time. *Cognitive Science*, 1990

[25] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats and Yann N. Dauphin. Convolutional Sequence to Sequence Learning. *arXiv preprint arXiv: 1705:03122*, 2017

[26] Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. *ACL*, 2015

[27] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, and Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. *EMNLP*, 2014

[28] Kyunghyun Cho, Bart van Merrinboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *SSST-8*, 2014

[29] Marco Marelli, Luisa Bentivogli, Marco Baroni, Raffaella Bernardi, Stefano Menini, and Roberto Zamparelli. Semeval-2014 task 1: Evaluation of compositional distributional semantic models on full sentences through semantic relatedness and textual entailment. *SemEval-2014*, 2014

[30] Mikael Henaff, Jason Weston, Arthur Szlam, Antoine Bordes and Yann LeCun. Tracking The World State with Recurrent Entity Networks. *ICLR*, 2017

[31] Minqing Hu and Bing Liu. Mining and summarizing customer reviews. *ACM SIGKDD*, 2004

[32] Paul J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 1990

[33] Prajit Ramachandran, Peter J. Liu, and Quoc V. Le. Unsupervised Pretraining for Sequence to Sequence Learning. *ICLR*, 2017

[34] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016

[35] Ryan Kiros, Yukun Zhu, Ruslan Salakhutdinov, Richard S. Zemel, Antonio Torralba, Raquel Urtasun, Sanja Fidler. Skip-Thought Vectors. *NIPS*, 2015

[36] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. End-to-end memory networks. *CoRR*, 2015

[37] Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Jozefowicz, and Samy Bengio. Generating Sentences From a Continuous Space. *arXiv preprint arXiv:1511.06349*, 2015

[38] Sanjeev Arora, Yingyu Liang, Tengyu Ma. A Simple But Tough-To-Beat Baseline For Sentence Embeddings. *ICLR*, 2017

[39] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 1997

[40] Susan T. Dumais. Latent Semantic Indexing (LSI) and TREC-2. *TREC*, 1994

[41] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. *NIPS*, 2013

[42] Tomas Mikolov, Stefan Kombrink, Lukas Burget, Jan Honza Cernocky, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. *ICASSP*, 2011

[43] Xin Li and Dan Roth. Learning question classifiers. *In Proceedings of the 19th international conference on Computational linguistics*, 2002

[44] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush. Character-Aware Neural Language Models. *CoRR*, 2015

[45] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014

[46] Yoshua Bengio, Rejean Ducharme, Pascal Vincent, and Christian Jauvin. A Neural Probabilistic Language Model. *JMLR*, 2003

[47] Yoshua Bengio, Jerome Louradour, Ronan Collobert, and Jason Weston. Curriculum Learning. *ICML*, 2009

[48] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 1994

[49] Yossi Adi, Einat Kermany, Yonatan Belinkov, Ofer Lavi, and Yoav Goldberg. Fine-grained Analysis of Sentence Embeddings Using Auxiliary Prediction Tasks. *ICLR*, 2017

[50] Yukun Zhu, Ryan Kiros, Richard S. Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. *arXiv preprint arXiv:1506.06724*, 2015

[51] Zellig S. Harris. Distributional Structure. *Word*, 1954