UC Irvine
UC Irvine Previously Published Works

Title
TrueHappiness: Neuromorphic Emotion Recognition on TrueNorth

Permalink
https://escholarship.org/uc/item/5mv970n5

Authors
Diehl, Peter U
Pedroni, Bruno U
Cassidy, Andrew
et al.

Publication Date
2016

Peer reviewed

eScholarship.org Powered by the California Digital Library
University of California

arXiv:1601.04183v1 [q-bio.NC] 16 Jan 2016

TrueHappiness: Neuromorphic Emotion Recognition on TrueNorth

Peter U. Diehl*[1], Bruno U. Pedroni†[1], Andrew Cassidy‡, Paul Merolla‡, Emre Neftci†§ and Guido Zarrella¶
*Institute of Neuroinformatics
ETH Zurich and University Zurich, Switzerland
Email: peter.u.diehl@gmail.com
†Institute for Neural Computation, UC San Diego, La Jolla, USA
Email: bpedroni@eng.ucsd.edu
‡IBM Research Almaden, San Jose, CA, USA
§Department of Cognitive Sciences, UC Irvine, Irvine, USA
¶The MITRE Corporation, Bedford, MA, USA
[1] Peter U. Diehl and Bruno U. Pedroni have contributed equally to this work

***Abstract*—We present an approach to constructing a neuromorphic device that responds to language input by producing neuron spikes in proportion to the strength of the appropriate positive or negative emotional response. Specifically, we perform a fine-grained sentiment analysis task with implementations on two different systems: one using conventional spiking neural network (SNN) simulators and the other one using IBM's Neurosynaptic System TrueNorth. Input words are projected into a high-dimensional semantic space and processed through a fully-connected neural network (FCNN) containing rectified linear units trained via backpropagation. After training, this FCNN is converted to a SNN by substituting the ReLUs with integrate-and-fire neurons. We show that there is practically no performance loss due to conversion to a spiking network on a sentiment analysis test set, i.e. correlations between predictions and human annotations differ by less than 0.02 comparing the original DNN and its spiking equivalent. Additionally, we show that the SNN generated with this technique can be mapped to existing neuromorphic hardware – in our case, the TrueNorth chip. Mapping to the chip involves 4-bit synaptic weight discretization and adjustment of the neuron thresholds. The resulting end-to-end system can take a user input, i.e. a word in a vocabulary of over 300,000 words, and estimate its sentiment on TrueNorth with a power consumption of approximately 50 μW.**

1. Introduction

The rise of the internet and more powerful computers has enabled an unprecedented ability to interpret and understand natural language. Deep neural networks (DNN) can be trained on massive datasets to perform a wide variety of natural language understanding and generation tasks [1], [2], [3]. One drawback of DNNs is that they usually require power hungry hardware, such as GPUs, posing a problem for mobile devices (e.g. smartphones), which present very stringent power constraints. A common solution is to outsource the computation to the cloud by sending the data to a data center, processing it there and then sending the results back to the mobile device. This works well as long as the amount of data to be processed is limited and as long as there is a reliable connection between the data center and the mobile device. If either one of these conditions is not met, the system will not operate adequately and the user is left without the desired functionality.

A possible solution for the problem of high power consumption is to use neuromorphic hardware to perform the processing [4], [5], [6], [7]. These brain-inspired systems work on an extremely low power budget: for example, the IBM's Neurosynaptic System TrueNorth can simulate 1 million neurons using less than 100 mW [8]. Mapping DNNs to neuromorphic hardware would make viable pattern recognition systems which present simultaneously low power and high performance [9], [10]. As a practical example, a TrueNorth chip analyzing a stream of language content could run on an iPhone battery nonstop for a week.

Here we present a first example of a NLP system that is based on spiking neural networks and is also implemented on neuromorphic hardware. Specifically, the system performs fine-grained sentiment analysis, i.e. evaluating how positive/negative a word or phrase is on a 1 to 9 scale. We start by training a DNN with rectified linear units (ReLU) on a dataset labeled with crowd-sourced happiness ratings [11]. After training, we substitute the ReLUs with integrate-and-fire neurons and adjust neuron thresholds and scale the weights appropriately [12]. A comparison between the original DNN and the converted SNN on the sentiment analysis test set shows that the drop in correlation between prediction and target due to the conversion to a spiking network is less than 0.02 for any of the tested setups. Using the SNN, we provide code for a real-time interactive demo, where the user can query any word in a >300,000 word vocabulary and compute the associated sentiment estimate.

In addition to the SNN that is simulated on a traditional computer, we present an implementation of the same fine-grained sentiment analysis task on TrueNorth. For the construction of this network we start from the generic SNN and proceed by mapping the synaptic weights between neurons using a quantization strategy with resulting weight precision of 4 bits.

We call this approach "train-then-constrain", comparing it to the "constrain-then-train" used in [13]. Using the "linear reset" mode, TrueNorth neurons can be configured to produce spiking rates similar to ReLUs. The final network uses 3 cores, consuming less than 0.1% of a TrueNorth chip to process language with an estimated power consumption of less than 50 μW.

The result is an important first step in the creation of a new generation of spiking cognition systems: a neuromorphic device which receives words as input and outputs a variable number of spikes in proportion to the strength of expected emotional response.

2. Methods

The workflow is depicted in figure 2.1. We begin by learning distributed word representations using the word2vec method, which models word meaning via a training process that predicts word co-occurrences in an unlabeled text corpus. We then convert the words in our training set into vectors, which we use as input features in the next processing stage where a neural network learns to predict the sentiment score associated with the input word. Training is done using traditional backpropagation.

After training is completed, the learned weights are used to construct a spiking neural network that performs the same task[1].

Finally, this spiking network is mapped onto TrueNorth. This is a "train-then-constrain" methodology, where an unconstrained network is learned using full precision weights, values, and neuron types, and then converted to a hardware compatible network. This contrasts with a "constrain-then-train" approach that accounts for the architectural constraints directly in the learning algorithm [13].

2.1. Data and Pretraining

The dataset used here was constructed from data first presented in [11]. It contains 4500 words with corresponding crowdsourced sentiment labels, i.e. a number between 1 and 9 (inclusive).

Our goal is to learn a fine-grained sentiment predictor that generalizes to words and phrases that are not in the training set. As such we need a large dictionary of words encoded with a feature representation that captures latent attributes of word meaning and usage. For this purpose we pre-train a word2vec [2] model which projects words into a high-dimensional space where the location of the

1. The code for running the python based demo of TrueHappiness is available under https://github.com/peter-u-diehl/truehappiness.

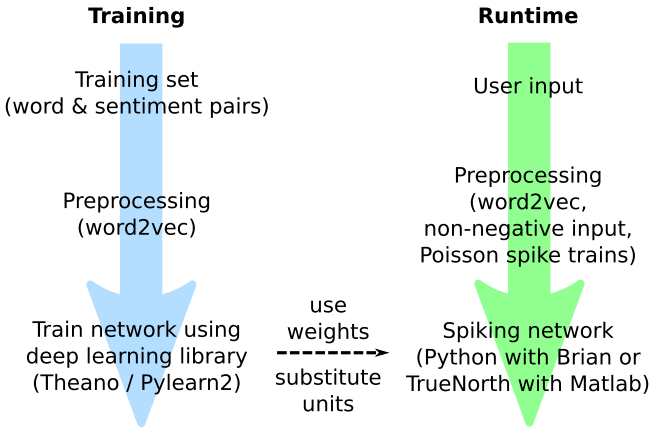


Figure 1. Conversion of the fully-connected neural network to a spiking neural network and a TrueNorth compatible network. The necessary steps are 1) training of the network using a traditional deep-learning library 2) substitution of the ReLU units with integrate-and-fire neurons 3) making inputs non-negative by either doubling number of input units or applying the exponential function 4) converting the real-numbered inputs to Poisson spike-trains. Additionally, for the TrueNorth network the weights are discretized to 4-bit precision and unit thresholds are adapted.

words corresponds to their meaning. Word similarities can be computed using the cosine distance between word vectors, e.g. "good" is close to "great" in this space but far away from "terrorist". It is also possible to use vector arithmetic to compute analogies and create features for novel or ambiguous concepts. For example, in our model the vector calculated from wordvector('happy') - wordvector('laughter') + wordvector('sad') is nearby wordvector('anguish'). A downstream model trained to recognize the sentiment of 'happy', 'laughter', and 'sad' could thus infer without explicit training that 'anguish' is a strongly negative term.

Our word vector representations are learned from a large corpus, in particular the text of the English Wikipedia. After some preprocessing of the 3.4 billion tokens in the corpus, like substitution of uncommon words with an "unknown" token, we use word2vec's skipgram algorithm with negative sampling [2] to learn a vector for the most common 324,263 words. Each word is thus represented as a vector of 64 dimensions, a number selected for convenience in mapping to the TrueNorth architecture (explained in section 2.4).

These word vectors form the basis for the inputs to a fully-connected feedforward neural network which we train to predict fine-grained sentiment scores. The network contains a hidden layer of 64 rectified linear units without biases and a single output unit connected by linear weights. The weights were trained via stochastic gradient descent using 4000 labeled examples. Hyperparameters were tuned to optimize performance on a held-out evaluation set of 500 examples.

2.2. Converting to a Spiking Neural Network

The conversion of the original FCNN to a spiking network has to address the following issues. First, it needs to substitute

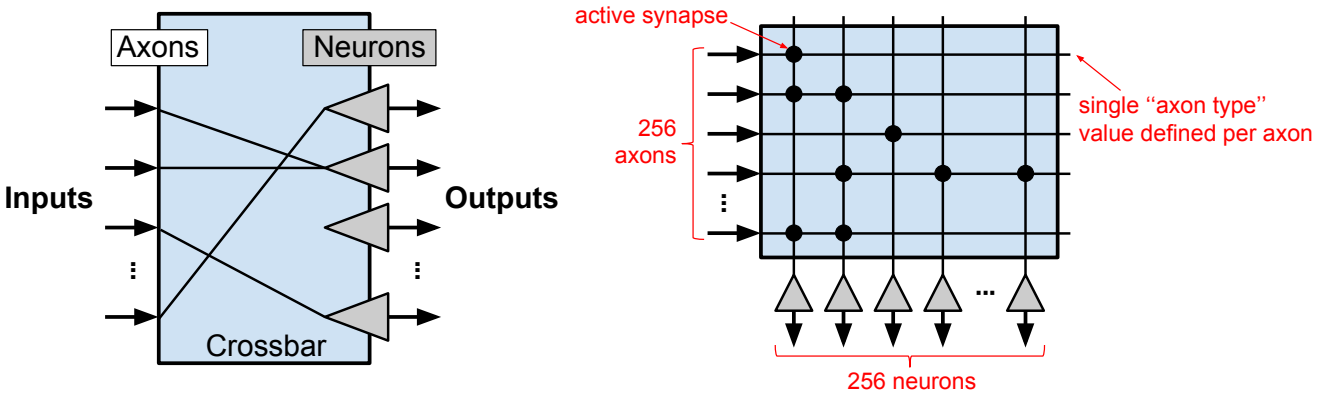


Figure 2. High-level abstraction of a TrueNorth core: the axons can be seen as the inputs, while the neurons integrate weights and produce output spikes. The synaptic connection between neurons is realized inside each core, with a weight value associated to each connection (dependent on the programmed axon type). Each core is formed by 256 axons and 256 neurons, totaling 2^{16} configurable synapses per core [8].

the 32-bit precision information transmission with single bit precision. Second, the information is not transmitted at every time step but instead only when enough information is acquired by the processing unit (i.e. when the neuron's membrane potential crosses its threshold).

These differences in information transmission imply in a range of changes to the original architecture. The first issue is easily addressed by only sending "spikes", i.e. the single bit information packet. This discretization will lead to high error if it is not compensated for. Two common ways of dealing with this discretization is to either use many units that send the same information to increase precision (population coding), or to use more than one time step to transmit the information (rate coding). In both cases, there needs to be a mechanism to determine when transmission of the next input pattern begins.

The second issue, in theory, can be easily addressed by setting the spiking threshold as small as the smallest incoming weight. However, in practice the amount of information which is integrated before a spike is transmitted should be taken into account. In other words, the relation between the "spiking threshold" of the units and the magnitude of the incoming weights must be considered to reduce the loss of information when many spikes arrive at the neuron during one timestep [12]. Here we decided to substitute the ReLUs with integrate-and-fire neurons since both units' response is linearly proportional to the input signal and is zero if the sum of the input is negative, see Figure 4 (left). For a better representation of the input we chose to accumulate inputs over time. Specifically, we use spike rate code with a Poisson spike-train, where at each time step the probability of an input spike is proportional to the strength of the input. One caveat of the system involves negative inputs, since there is no direct way of representing them using only an "on" signal. We present two approaches to address this problem. The first consists of doubling the number of input neurons, copying the weights from the trained network and then multiplying them by -1. If the number of input neurons is an issue, which is commonly the case in neuromorphic systems, it would be more useful to transform the input such that it is non-negative. For our TrueNorth implementation, we chose to follow a second strategy, where we trained and tested the network using the exponential transform of the original input, thereby assuring that all input values are non-negative.

2.3. TrueNorth

The IBM TrueNorth is a very low power digital neurosynaptic processor that uses brain-inspired processing and topology [8], [14]. The chip consists of 1 million programmable spiking neurons and 256 million configurable synapses, uniformly distributed throughout a 64 x 64 core architecture. Each core is composed of 256 axons (inputs) and 256 neurons (outputs), connected via a 256 x 256 crossbar of configurable synapses (see Figure 2). The system operates in 1 ms timesteps ("ticks"), during which membrane potential processing and spike event routing occur asynchronously inside the chip. Spikes generated by a neuron can target any single axon on the chip, with each neuron presenting over 20 individually programmable features (e.g. threshold, leak, and reset).

The following three sequentially processed equations define the dynamics of the membrane potential $V_j(t)$ for neuron j at time t (for the full equation see [15]:

$$V_j(t) = V_j(t-1) + \Sigma_{i=0}^{255}\ A_i(t)\ w_{i,j}\ s_j^{G_i} \qquad (1)$$

$$V_j(t) = V_j(t) + \lambda_j \qquad (2)$$

$$\text{if } (V_j(t) \geq \alpha_j),\ \text{Spike and set } V_j(t) = V_j(t) - \alpha_j \qquad (3)$$

The first equation represents the synaptic integration of all active axons at time t. The term $A_i(t)$ is the binary-valued input spike on the i-th axon at time t; $w_{i,j}$ is the binary-valued synaptic connection between axon i and neuron j; and $s_j^{G_i}$ is the synaptic weight between axon i and neuron j. Each neuron presents four individually configurable 9-bit

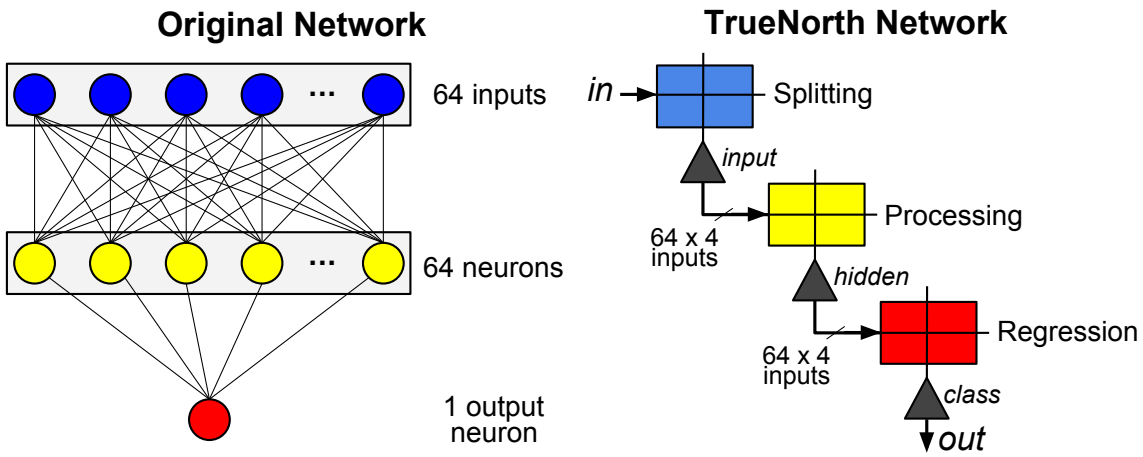
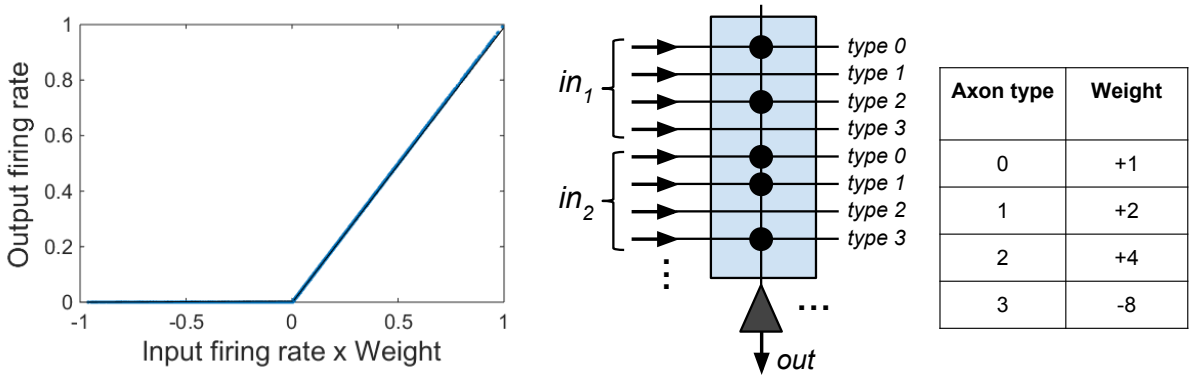


Figure 3. Structure of the fully-connected neural network and the corresponding implementation on TrueNorth using 3 out of the 4096 cores available on one chip.



Axon type	Weight
0	+1
1	+2
2	+4
3	-8

Figure 4. Rectified linear unit (ReLU) implemented using TrueNorth neurons (left). Example of a TrueNorth crossbar with multi-axon strategy per input for better weight representation per synaptic connection (right).

signed integer weights, however the last term in equation 1 is dependent on the axon type, making each connection's weight not independent. In this manner, an axon can be configured to be one of four types, and this defines which of the four weights (in each neuron) will be integrated into each neuron case the axon is active.

The second equation simply integrates the configurable leak value for the neuron. Lastly, the third equation compares the updated membrane potential (after weight integration) with the threshold. One of the neuron properties particular to TrueNorth is the "linear reset mode". In this mode, if $V_j(t)$ is equal to or surpasses the threshold ($\alpha_j(t)$), the neuron spikes and its membrane potential is subtracted by the threshold value.

Given the high-level view of the TrueNorth system, the following subsection will cover the implementation details of the neural network mapped onto TrueNorth, including the architectural and algorithmic adaptations demanded when using digital spiking neurons for inference [16].

2.4. Implementation Details

The architecture of the original artificial neural network to be mapped onto the TrueNorth system can be seen in Figure 3 (left). The entire network was mapped using 3 TrueNorth cores – one per layer. A high-level architecture of the TrueNorth network is shown in Figure 3 (right). The first layer consists of 64 inputs: the 64-dimensional word2vec representation of the input word under analysis. The next layer is comprised of 64 hidden units, all connected to a single output regression neuron.

The TrueNorth implementation of the FCNN uses the linear reset mode to closely match the ReLUs of the original network and 4-axon inputs to obtain better weight representation. Figure 4 (left) shows how a linear reset TrueNorth neuron configured with threshold $\alpha = 100$ and $\lambda = 0$ behaves similarly to a ReLU: there is a linear relationship between "normalized product of input firing rate and connection weight" and the "normalized output firing rate".

The 64 inputs in the second and third layers (cores) make use of all of the 256 axons on their respective cores, while

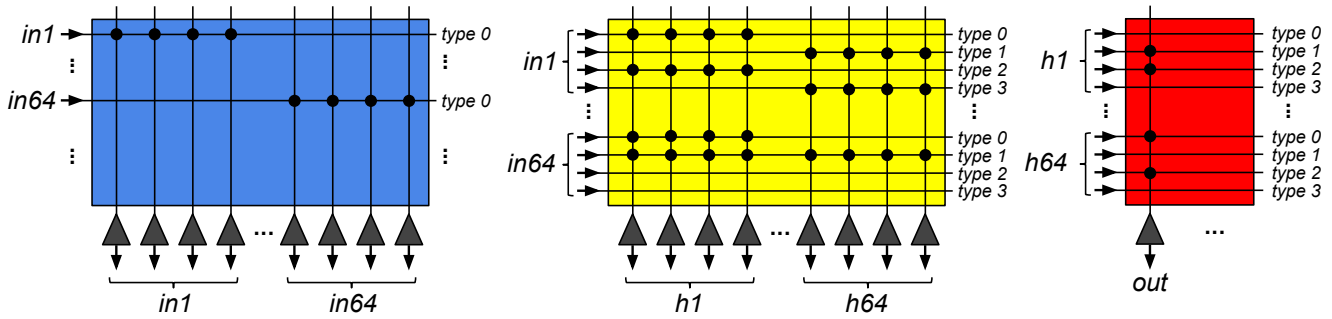


Figure 5. Example configuration of the crossbars on TrueNorth cores for each network layer. The first core is used for replicating the inputs for better weight representation (left). The second core processes spikes between the input layer and the hidden layer (center). The third core processes spikes between the hidden layer and the output neuron, where the number of output spikes represents the winner class (right).

the first core is used simply for splitting (replicating $4\times$) the inputs. The reason for using all 256 axons on one core for 64 inputs is that the weight precision on TrueNorth is limited due to the axon type defining the weight used in each connection. A strategy for obtaining a better weight representation in TrueNorth is to use multiple axons for each input. In this manner, the total effect of an active input will be a linear combination of all the axons it is connected to, resulting in multiple available weights per connection. Figure 4 (right) visualizes this strategy, where 4 axons are used per input, resulting in 2^4 different possible weight values available per input-output connection. In the example, the weight between in_1 and out equals +5, while the weight between in_2 and out equals -5. The main drawback is the need to use "splitter" cores for replicating a single input to multiple axons.

Figure 5 (left) shows how the neurons and axons are used for each layer. The first core is used to create replicas of the 64 inputs, which is obtained by setting the neurons with threshold $\alpha = 1$ and making the weight of all connections also equal to one. In this manner, any incoming spike event will trigger 4 replicas of this event. The second core, shown in Figure 5 (center), is used to process the input spikes by integrating the mapped weights and generating hidden layer spike events whenever the membrane potential crosses its spiking threshold. The final core is depicted in Figure 5 (right) and operates in the same manner as the second core, however now the spiking activity of the output neuron is sent off-chip. The regression task, therefore, produces a result based on the number of spikes which were output during the experiment time window T.

The data, code for the Python implementation, and code for the real-time TrueNorth demo will be made publicly available but the latter will require access to a TrueNorth chip or to the Compass simulator (for which currently an agreement with IBM is required).

3. Results

After training the fully-connected network on 4000 word vector and sentiment pairs, we test the sentiment prediction performance on the remaining 500 word-sentiment pairs of the dataset. We measure the Pearson correlation between the human sentiment annotations and our networks' predictions. Table 1 shows the performance for the original DNN and for the converted SNN using a range of different setups. The original DNN and the DNN trained with exponentiated inputs both achieve a fairly high correlation above 0.6. When discretizing the weights of the networks, the correlation goes down to 0.499 and 0.408 for the original and the exponential inputs, respectively. The SNNs derived from these four different networks show comparable performances. Notably, as integration time increases, the performance of the SNNs approaches the performance of their 32-bit counterpart for every single one of the four setups.

Configuration	Correlation
Original DNN	**0.637**
SNN, 0.01s integration	0.291
SNN, 0.10s integration	0.377
SNN, 1.00s integration	**0.631**
Original DNN - exp	**0.661**
SNN, 0.01s integration - exp	0.279
SNN, 0.10s integration - exp	0.544
SNN, 1.00s integration - exp	**0.651**
Original DNN - 4 bit	**0.499**
SNN, 0.01s integration - 4 bit	0.183
SNN, 0.10s integration - 4 bit	0.293
SNN, 1.00s integration - 4 bit	**0.486**
Original DNN - exp, 4 bit	**0.408**
SNN, 0.01s integration - exp, 4 bit	0.310
SNN, 0.10s integration - exp, 4 bit	0.380
SNN, 1.00s integration - exp, 4 bit	**0.392**

TABLE 1. RESULTS FOR SENTIMENT TEST SET. THE TABLE SHOWS THE CORRELATION BETWEEN THE PREDICTED SENTIMENT AND THE CORRECT SENTIMENT LABELS FOR THE ORIGINAL MODEL AND THE SNN. WITH INCREASING INTEGRATION TIME THE PERFORMANCE OF THE SNN APPROACHES THE PERFORMANCE OF THE ORIGINAL RATE-BASED NETWORK.

TrueHappiness, the sentiment prediction system adapted for TrueNorth, uses an interactive GUI where the user can type any word in the >300,000 word vocabulary. The input word is then converted to a constant length vector using

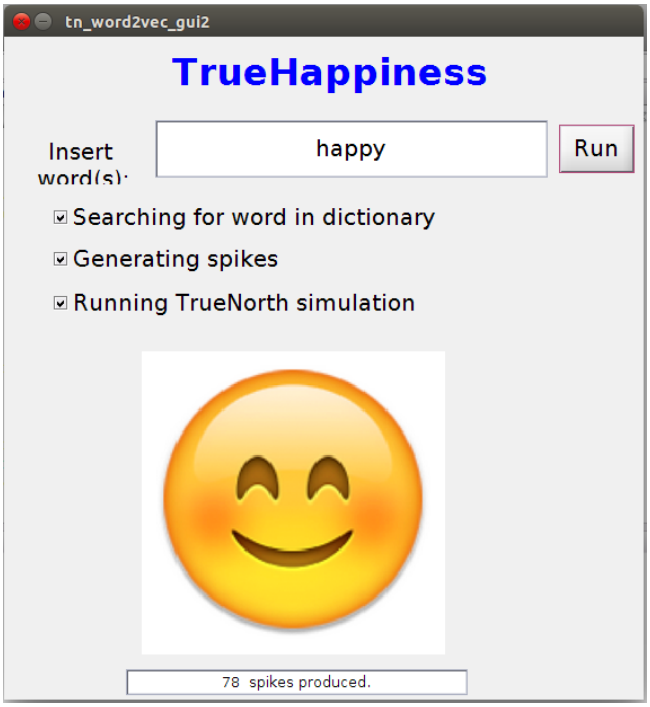


Figure 6. The interface for the real-time interactive demo. The user can type words contained in the >300,000 word vocabulary and the system will predict its sentiment using TrueNorth.

word2vec, which is then converted to Poisson spike trains and transmitted to the TrueNorth chip. On TrueNorth, the SNN processes the input and sends the output spikes back to the computer where the sentiment is categorized into classes based on the number of spikes, and the class and the number of output spikes are displayed in the GUI (see Figure 6).

4. Discussion

We showed a first example of a low power NLP system by leveraging established machine learning techniques and mapping the resulting neural networks to the TrueNorth chip. This framework for converting fully-connected networks trained with backpropagation to a TrueNorth compatible network is more generally applicable. In particular, the framework offers high flexibility because of its modularity. A neural network designer can optimize the performance of the network by improving optimization methods, loss functions or other recently developed methods and, as long as the weights and connectivity are in the required format, the rest of the processing pipeline does not change. Similarly, the rest of the pipeline is oblivious to which library is used for training the NN, which means the NN designer has the freedom to choose libraries such as Pylearn2/Theano [17], Torch or Caffe [18]. Moreover, none of the steps in the framework (besides the word2vec preprocessing) depend on the task to be solved. For example, a similar approach has been used for question classification [19]. The inputs might just as well be pixel intensities or an audio signal, without needing to change the framework.

One important issue that is not tackled here is the connectivity of the NN and it is assumed that the NN designer takes care of the connectivity constraints on TrueNorth, i.e. every neuron can only connect to one other axon (i.e. 256 neurons) and every neuron can only receive up to 256 inputs. While this might sound like a harsh restriction, many high performance NNs, like convolutional neural networks [20], use local connectivity by design and are therefore easier to adapt to TrueNorth. Another restriction is that learning is done offline, i.e. the system can not adapt its parameters to new incoming data. While this is probably acceptable for many practically relevant systems, many other systems might need to adjust to a specific user or a new environment. In such cases it would be necessary to use neuromorphic hardware that brings efficient online learning [5], [21], [22] in combination with online learning algorithms [23], [24], [25], [26], [27]. However, so far it remains a challenge to achieve performances comparable to conversion methods using such online learning approaches.

The mapping from the FCNN to the spiking network does lead to a performance gap between the original network and its spiking counterpart. However, this gap is very small for longer presentations of the inputs and makes it possible for the system designer to trade-off accuracy for resources depending on the application. For example in an application where there is very high variance in the inputs, it is likely not useful to push the limits of accuracy for a given input but instead decrease latency, thus increasing throughput. Another way to minimize the performance gap, would be to duplicate the inputs and thereby use multiple Poisson spike-trains as inputs per actual input. Since those two methods are complementary, they can also be combined.

A more important source of performance loss is the weight discretization to 4 bits which is used here for the mapping to TrueNorth. Using the naive approach of discretizing or rounding the weights to the nearest of the 16 values results in a reduction in Pearson correlation of 0.138. While this is quite a significant reduction, there are many ways to improve upon this method. One possibility is to use a layer-wise training method where after the first rounding step the resulting network is trained again (using standard backpropagation) but with fixed weights for the first layer. After this re-training, the weights are rounded again and the re-training could then be repeated as often as there are remaining layers. In this way, the network will learn to use the discretized weights in the earlier layers in a better way and potentially compensate the discretization loss. Other ways would be to use stochastic rounding [28] or more sophisticated training methods such as the dual-copy rounding [29], where the weights are discretized in the forward pass of the backpropagation algorithm but full resolution weights are used for backpropagating the error signal. This results in weights that show only minimal loss due to discretization. If resources are not a major issues at run time, it would also be possible to use a probabilistic rounding of the weights and use multiple instances of the

network. For example if a weight has a value of 0.7 and we want to discretize is to 0 or 1, we could flip ten biased coins, using every coin for one of ten different networks. A possible outcome of those coin flips would be that seven of the ten networks are using a weight of 1 and the remaining three networks use a weight of 0. By averaging over the results of these networks, the result will most likely improve, similar to other committee methods in machine learning [30], but only requiring to train the original network once. "Constrain-then-train" approaches [13] successfully employ strategies such as these for maximizing runtime system accuracy.

In order to deal with negative inputs, we introduced two methods for representing these data using spike trains. The first one consists of duplicating every input neuron which potentially needs to represent a negative value. This leads to a very small performance gap between the original and the spiking network, i.e. the correlation only decreased by 0.006. The other presented approach is to exponentiate the inputs to ensure that they are non-negative. Interestingly, for the sentiment analysis task this even improved the performance of the original network as well as the performance of the spiking network, even though the performance gap slightly increased to 0.01. After discretizing the weights to 4 bits to be able to map them on TrueNorth, the performance gap increases slightly for both methods: to 0.013 for the input duplication and to 0.016 for the exponentiation. However, in the case of the exponentiation, the drop in correlation due to weight discretization is with 0.253 much bigger than the performance gap for spiking networks, rendering the performance gap due to a spiking representation relatively insignificant. A likely explanation for this increased loss is that the exponentiation reduces the absolute difference between some words and increases the difference for other words. This would require more finely tuned weights to correctly process the small differences, but this exact fine tuning will not be represented in the 4-bit weights. It remains an open question whether the above mentioned methods for reducing the discretization loss might help to avoid the negative effect of the exponentiation.

Despite the mentioned challenges, this work represents an important first step towards practically relevant and extremely low power NLP systems and other neuromorphic pattern recognition systems. The main advantage of our method lies in its modularity and flexibility since progress in deep learning research will directly impact the performance of neuromorphic recognition systems designed with our framework.

Acknowledgments

We thank the organizers and the participants of Telluride Neuromorphic Cognition Engineering Workshop 2015 for the unique environment that enabled the presented work. Especially the natural language processing group, Rodrigo Alvarez, and John Arthur for many fruitful discussions.

Funding: PUD: SNF Grant 200021-143337 "Adaptive Relational Networks." BUP: The Office of Naval Research (ONR MURI 14-13-1-0205) and CNPQ Brazil (CsF 201174/2012-0) EN: the Office of Naval Research (ONR MURI 14-13-1-0205)

References

[1] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *The Journal of Machine Learning Research*, vol. 12, pp. 2493–2537, 2011.

[2] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[3] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[4] G. Indiveri, E. Chicca, and R. Douglas, "A VLSI array of low-power spiking neurons and bistable synapses with spike–timing dependent plasticity," *IEEE Transactions on Neural Networks*, vol. 17, no. 1, pp. 211–221, Jan 2006. [Online]. Available: http://ncs.ethz.ch/pubs/pdf/Indiveri_etal06.pdf

[5] M. Khan, D. Lester, L. Plana, A. Rast, X. Jin, E. Painkras, and S. Furber, "Spinnaker: mapping neural networks onto a massively-parallel chip multiprocessor," in *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*. IEEE, 2008, pp. 2849–2856.

[6] J. Schemmel, D. Brüderle, A. Grübl, M. Hock, K. Meier, and S. Millner, "A wafer-scale neuromorphic hardware system for large-scale neural modeling," in *International Symposium on Circuits and Systems, ISCAS 2010*. IEEE, 2010, pp. 1947–1950.

[7] G. Indiveri, B. Linares-Barranco, T. Hamilton, A. van Schaik, R. Etienne-Cummings, T. Delbruck, S.-C. Liu, P. Dudek, P. Häfliger, S. Renaud, J. Schemmel, G. Cauwenberghs, J. Arthur, K. Hynna, F. Folowosele, S. Saighi, T. Serrano-Gotarredona, J. Wijekoon, Y. Wang, and K. Boahen, "Neuromorphic silicon neuron circuits," *Frontiers in Neuroscience*, vol. 5, pp. 1–23, 2011. [Online]. Available: http://www.frontiersin.org/Neuromorphic_Engineering/10.3389/fnins.2011.00073/abstract

[8] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[9] S. K. Esser, A. Andreopoulos, R. Appuswamy, P. Datta, D. Barch, A. Amir, J. Arthur, A. Cassidy, M. Flickner, P. Merolla *et al.*, "Cognitive computing systems: Algorithms and applications for networks of neurosynaptic cores," in *Neural Networks (IJCNN), The 2013 International Joint Conference on*. IEEE, 2013, pp. 1–10.

[10] D. Martí, M. Rigotti, M. Seok, and S. Fusi, "Energy-efficient neuromorphic classifiers," *arXiv preprint arXiv:1507.00235*, 2015.

[11] P. S. Dodds, K. D. Harris, I. M. Kloumann, C. A. Bliss, and C. M. Danforth, "Temporal patterns of happiness and information in a global social network: Hedonometrics and twitter," *PloS one*, vol. 6, no. 12, p. e26752, 2011.

[12] P. U. Diehl, D. Neil, J. Binas, M. Cook, S.-C. Liu, and M. Pfeiffer, "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *International Joint Conference on Neural Networks (IJCNN),*. IEEE, 2015, pp. 1–8.

[13] S. K. Esser, R. Appuswamy, P. Merolla, J. Arthur, and D. Modha, "Backpropagation for energy-efficient neuromorphic computing," *Neural Information Processing Systems*, 2015.

[14] P. Merolla, J. Arthur, F. Akopyan, N. Imam, R. Manohar, and D. Modha, "A digital neurosynaptic core using embedded crossbar memory with 45pj per spike in 45nm," in *Custom Integrated Circuits Conference (CICC), 2011 IEEE*, Sept. 2011, pp. 1–4.

[15] A. S. Cassidy, P. Merolla, J. V. Arthur, S. K. Esser, B. Jackson, R. Alvarez-icaza, P. Datta, J. Sawada, T. M. Wong, V. Feldman, A. Amir, D. B. dayan Rubin, E. Mcquinn, W. P. Risk, and D. S. Modha, "Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores," in *in International Joint Conference on Neural Networks (IJCNN). IEEE*, 2013.

[16] A. Amir, P. Datta, W. P. Risk, A. S. Cassidy, J. Kusnitz, S. K. Esser, A. Andreopoulos, T. M. Wong, M. Flickner, R. Alvarez-Icaza *et al.*, "Cognitive computing programming paradigm: a corelet language for composing networks of neurosynaptic cores," in *Neural Networks (IJCNN), The 2013 International Joint Conference on*. IEEE, 2013, pp. 1–10.

[17] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: a CPU and GPU math expression compiler," in *Proceedings of the Python for Scientific Computing Conference (SciPy)*, vol. 4, 2010.

[18] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[19] P. U. Diehl, G. Zarrella, A. Cassidy, B. Pedroni, and E. Neftci, "Spike-based recurrent neural networks," *arXiv*, 2016.

[20] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[21] P. U. Diehl and M. Cook, "Efficient implementation of stdp rules on spinnaker neuromorphic hardware," in *Neural Networks (IJCNN), 2014 International Joint Conference on*. IEEE, 2014, pp. 4288–4295.

[22] F. Galluppi, X. Lagorce, E. Stromatias, M. Pfeiffer, L. A. Plana, S. B. Furber, and R. B. Benosman, "A framework for plasticity implementation on the spinnaker neural architecture," *Frontiers in neuroscience*, vol. 8, 2014.

[23] M. Beyeler, N. D. Dutt, and J. L. Krichmar, "Categorization and decision-making in a neurobiologically plausible spiking network using a stdp-like learning rule," *Neural Networks*, vol. 48, pp. 109–124, 2013.

[24] T. Masquelier, R. Guyonneau, and S. J. Thorpe, "Competitive stdp-based spike pattern learning," *Neural Computation*, vol. 21, pp. 1259–1276, 2009.

[25] D. Kappel, B. Nessler, and W. Maass, "Stdp installs in winner-take-all circuits an online approximation to hidden markov model learning," *PLoS computational biology*, vol. 10, no. 3, p. e1003511, 2014.

[26] E. Neftci, S. Das, B. Pedroni, K. Kreutz-Delgado, and G. Cauwenberghs, "Restricted boltzmann machines and continuous-time contrastive divergence in spiking neuromorphic systems," May 2013.

[27] P. U. Diehl and M. Cook, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," *Frontiers in Computational Neuroscience*, vol. 9, p. 99, 2015.

[28] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," *arXiv preprint arXiv:1502.02551*, 2015.

[29] E. Stromatias, D. Neil, M. Pfeiffer, F. Galluppi, S. B. Furber, and S.-C. Liu, "Robustness of spiking deep belief networks to noise and reduced bit precision of neuro-inspired hardware platforms," *Frontiers in neuroscience*, vol. 9, 2015.

[30] D. C. Cireşan, U. Meier, L. M. Gambardella, and J. Schmidhuber, "Convolutional neural network committees for handwritten character classification," in *Document Analysis and Recognition (ICDAR), 2011 International Conference on*. IEEE, 2011, pp. 1135–1139.