TEMPORAL KERNEL NEURAL NETWORK LANGUAGE MODEL

YongZhe Shi, Wei-Qiang Zhang, Meng Cai and Jia Liu

Tsinghua National Laboratory for Information Science and Technology,
Department of Electronic Engineering, Tsinghua University, Beijing 100084, China
{shiyz09, caimeng06}@gmail.com, {wqzhang, liuj}@tsinghua.edu.cn

ABSTRACT

Using neural networks to estimate the probabilities of word sequences has shown significant promise for statistical language modeling. Typical modeling methods include multi-layer neural networks, log-bilinear networks and recurrent neural networks, etc. In this paper, we propose the temporal kernel neural network language model, a variant of models mentioned above. This model explicitly captures long-term dependencies of words with exponential kernel, where the memory of history is decayed exponentially. Additionally, several sentences with variable lengths as a mini-batch are efficiently implemented for speeding up. Experimental results show that the proposed model is very competitive to the recurrent neural network language model and obtains the lower perplexity of 111.6 (more than 10% reduction) than the state-of-the-art results reported in the standard Penn Treebank Corpus. We further apply this model to Wall Street Journal speech recognition task, and observe significant improvements in word error rate.

Index Terms— language modeling, temporal kernel neural network, speech recognition

1. INTRODUCTION

Statistical language models (LMs) estimate the probability of a word occurring in a given context, which plays an important role in many practical applications such as automatic speech recognition and machine translation. Having been used for several decades, n-gram models are still the cornerstone of modern language modeling for their simplicities and efficiencies. Standard n-gram back-off LMs rely on a discrete representation of the vocabulary, where each word is associated with a discrete index and the morphological, syntactic and semantic relationships which structure the lexicon are completely ignored. N-gram models rely on Markovian assumption. Despite this simplification, the maximum likelihood estimate (MLE) remains unreliable and tends to underestimate the probability of very rare n-grams which are hardly observed even in huge corpora. One of the most successful alternatives to date is to use neural networks to estimate the probabilities of word sequences, which has shown significant promise for statistical language modeling. In this approach, distributed word representations and the associated probability estimates are jointly computed in a multi-layer or recurrent neural network architecture, where distributionally similar words are represented as neighbors in a continuous space. The predicted probability of the next word is turned into a smooth function of the word representations, which alleviates the sparsity issue to some extent and leads to better generalization for unseen n-grams.

Typical modeling methods include multi-layer neural networks [1], log-bilinear neural networks [2] and recurrent neural networks [3, 4, 5], etc. The traditional multi-layer neural network LMs (MLPLMs) and log-bilinear neural network LMs (LBLMs) are still based on Markovian assumption and they can do nothing about the words beyond the context range. In contrast, recurrent neural network (RNN) is beyond Markovian assumption and is very suitable for long series modeling, where the hidden states compactly cluster the entire history. In practice, recurrent neural network language models (RNNLMs) have achieved record-breaking perplexities on many tasks since they were proposed [6, 7]. However, according to [7], the improvements obtained with RNNLMs come from better representation of short context information, while not from learning cache information. Actually, if recurrent neural network is trained by stochastic gradient descent (SGD), the error signal propagated through the recurrent connections converges to zero or explodes fast in most cases [8]. Thus, it is hard to train a RNN to represent long-term patterns that span over the entire sentence.

In this paper, the temporal kernel neural network language model (TKNNLM) is proposed to explicitly capture long term dependencies of words with exponential kernel, just like [9]. In this approach, the error signal is directly propagated to every word in the entire history and the context information is forgotten with exponential decay. Experimental results show that our proposed TKNNLM can easily capture the long-term dependencies and obtains the lower perplexity of 111.6 than

This work was supported by National Natural Science Foundation of China under Grant No. 90920302, No. 61005019 and No. 61273268, and in part by Beijing Natural Science Foundation Program and Scientific Research Key Program of Beijing Municipal Commission of Education under Grant No. KZ201110005005

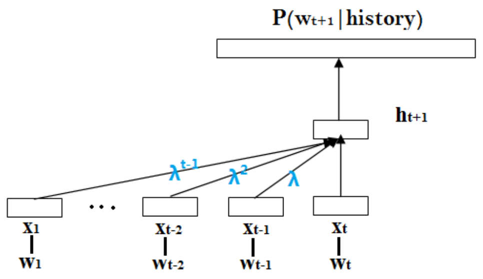

Fig. 1. *Temporal kernel neural network language model*

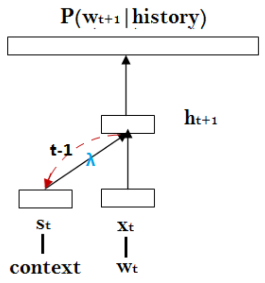

Fig. 2. *The recurrent connection in temporal kernel neural network language model*

the state-of-the-art results reported on the standard Penn Treebank Corpus.

This paper is organized as follows: In section 2, the temporal kernel neural network language model is described. The training algorithm based on mini-batch is presented in section 3. Detailed experiments and performance evaluations are presented in section 4 and 5. Finally section 6 gives the conclusion and the future work.

2. MODEL DESCRIPTION

In this section, we formally define the TKNNLM as shown in Fig. 1, and we consider the words as indices in a finite vocabulary with V words. A word w will either refer to the word or the index in the vocabulary, and it can be represented by a 1-of-V coding vector $\mathbf{v} \in \mathbb{R}^V$, where all elements are null except the w-th.

The first layer builds a continuous representation of the history by mapping each word into its real-valued representation. The mapping is defined by $\mathbf{R}^T\mathbf{v}$, where $\mathbf{R} \in \mathbb{R}^{V\times D}$ is a projection matrix and D is the dimension of the continuous projection space. The TKNNLM is a neural network that operates in time. Let $\mathbf{x}_t = \mathbf{R}^T\mathbf{v}_t$, where $\mathbf{x}_t$ denotes the word vector at time t.

The second layer transforms the entire input history with exponential kernel, where each unit of the non-linear layer is an efficient leaky integrator, which makes it easier to notice long-term dependencies. The dimension of the second layer is denoted as H and the second layer are computed as follows:

$$\mathbf{h}_{t+1} = \mathrm{f}(\mathbf{s}_{t+1} + \mathbf{b}_h) \tag{1}$$

and

$$\mathbf{s}_{t+1} = \mathbf{x}_t + \lambda * \mathbf{x}_{t-1} + \lambda^{\mathbf{2}} * \mathbf{x}_{t-2} + ... \tag{2}$$

where each element of the vector $\lambda \in \mathbb{R}^H$ is a decay factor for the specific dimension and "$*$" means the element-wise multiplication of vectors. The contribution of the history is decayed exponentially with time, if each element of λ is constrained between -1.0 and 1.0. In Eq.(1), $\mathbf{b}_h \in \mathbb{R}^H$ is a bias for the hidden layer and $\mathrm{f}(\mathbf{z})$ is a non-linear activation function:

$$\mathrm{f}(\mathbf{z}) = \tanh(\mathbf{z}) = \frac{e^{\mathbf{z}} - e^{-\mathbf{z}}}{e^{\mathbf{z}} + e^{-\mathbf{z}}} \tag{3}$$

Finally, the predicted probability $\mathrm{P}(\mathbf{w}_{t+1}|\mathbf{history})$ is computed in the output layer.

$$\mathrm{P}(\mathbf{w}_{t+1}|\mathbf{history}) = \mathrm{g}(\mathbf{h}_{t+1}\mathbf{W}_{ho} + \mathbf{b}_o) \tag{4}$$

where $\mathbf{W_{ho}} \in \mathbb{R}^{H\times V}$ is a prediction matrix, $\mathbf{b_o} \in \mathbb{R}^V$ is a bias and $\mathrm{g}(\mathbf{z})$ is the softmax function:

$$\mathrm{g}(\mathbf{z}_m) = \frac{e^{\mathbf{z}_m}}{\sum_k e^{\mathbf{z}_k}} \tag{5}$$

which ensures that the output is a probability distribution. Actually, according to Eq.(2), $\mathbf{s}_{t+1}$ can be easily rewritten as a recursive form:

$$\begin{aligned} \mathbf{s}_{t+1} &= \lambda * \mathbf{s_t} + \mathbf{x_t} \\ \mathbf{s}_0 &= \mathbf{0} \end{aligned} \tag{6}$$

The implicit recurrent connection of TKNNLM is shown in Fig. 2 and the structure of TKNNLM is quite similar to that of RNNLM. By using the recurrent connection, information can cycle inside the network for arbitrarily long time. Note that the recurrent connection in TKNNLM is before rather than after the non-linear activation, where an exponential kernel is introduced and each element of decay vector is trained separately. As usual, the dimension D and H can be different in traditional MLPLMs, so do $\mathbf{R}$ and $\mathbf{W_{ho}}$. In practice, the word vectors in these two matrices are distributed similarly in the high-dimensional space, where similar words are distributed together. These two matrices are constrained to be identical in LBLM and the parameters are reduced by half. Therefore, to compress the size of the model and reduce the parameters, the projection matrix is assumed to be a linear transformation of the prediction matrix. That is to say $\mathbf{R} = \mathbf{W}_{ho}^T\mathbf{W}_{ih}$, where $H = D$ and $\mathbf{W}_{ih} \in \mathbb{R}^{D\times D}$. Additionally, to constrain the range of the λ vector between -1.0 and 1.0, another vector $\lambda' \in \mathbb{R}^D$ is introduced where $\lambda = \tanh(\lambda')$, and λ' are trained.

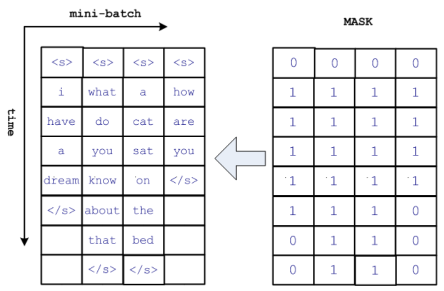

Fig. 3. *Several sentences with variable lengths are taken as a mini-batch in the training for speeding up. Mask is used to indicate whether the error of the word needs to be summed.*

3. TRAINING BASED ON STOCHASTIC GRADIENT DESCENT

At each training step, the error vector is computed according to the cross entropy criterion. The TKNNLM can easily be unfolded to a feedforward neural network as the RNNLM and the gradient of the error can be efficiently computed by back-propagation through time algorithm (BPTT). Compared with the RNNLM, TKNNLM is a shallow network, which is more easily trained by SGD. In this approach, a sentence is taken as a unit for processing, where the parameters are updated after one forward and backward pass of the entire sentence. To speed up, several sentences with variable lengths are taken as a mini-batch in the training, where matrix-vector operations are replaced by matrix-matrix operations. The matrix-matrix operations have been efficiently optimized in BLAS Library of MKL or CUDA. At the same time, a binary mask matrix is generated according to the mini-batch as shown in Fig. 3 where only the element corresponding to a word will be set as 1. The error signal is accumulated only if the mask of that word is 1, so does the gradient. If a sentence is longer than a predefined threshold $Tmax$, it will be truncated. Additionally, to improve the convergence, the sentences in the corpus are shuffled at the beginning of each iteration.

4. PERPLEXITY EVALUATION

One of the most widely used data sets for evaluating performance of statistical language models is the Penn Treebank portion of the Wall Street Journal Corpus (denoted as PTB corpus). The proposed TKNNLM is evaluated on the PTB corpus which is preprocessed by lowercasing words, removing punctuation and replacing numbers with the “N” symbol. Sections 00-20 (930K words) are used as training sets, sections 21-22 as validation sets (74K words), and sections 23-24 as test sets (82K words). The vocabulary size is 10K, including a special token for unknown words. The same setup has been previously used by many researchers, which allows us to compare directly the performance of different techniques in language modeling. First of all, according to [6], the perplexity of state-of-the-art LMs reported in this setup, without consideration of the complexity, are given in Table 1, where the RNNLM with 400 hidden units is the best model. Then, a TKNNLM with 400 hidden units is trained by BPTT to compare the performance. The learning rate is set as $lr = initlr/(1.0 + lrmult \times count)$, where initial learning rate ($initlr$) is set as 1.0, the learning rate decay factor ($lrmult$) is set as 4×10^{-7} and $count$ denotes the number of words processed. When the perplexity of the validation decreases very slowly or increases, $initlr$ is halved. To speed up, all the TKNNLMs are trained with a mini-batch of 5 sentences. On average, the weights are updated once every 100 words. The baseline 5-gram with modified Kneser-Ney smoothing (KN5) is trained by MITLM toolkit [10]. Experimental results in Table 1 show that the proposed TKNNLM is very competitive to the RNNLM, and obtains the lower perplexity of 111.6 (more than 10% reduction compared with that of RNNLM). The improvement is significant, even compared to the interpolated with KN5. In the following, different RNNLMs are trained using rnnlm toolkit [11] for further comparisons. The truncated BPTT is used for training the RNNLMs with bptt=5, bptt-block=10 and min-improvement=1.001. To compare fairly, the full vocabulary is used for the output without a class layer. Detailed perplexity comparisons with different size of hidden layers on the test set of PTB Corpus are shown in Fig.4. The dashed curves denote the RNNLMs (blue) and the interpolated with KN5 (red). The solid curves denote the TKNNLMs (blue) and the interpolated with KN5 (red). Note that the perplexity of the RNNLM with 400 hidden units is a little higher than 124.7 in Table 1. Set bptt-block=1 and min-improvement=1.0005, the performance of RNNLMs can be further improved with a longer training time. Obvious improvements are observed in Fig.4. The number of parameters in TKNNLMs is $D \times V + D \times D + D + D + V$, including $\mathbf{W}_{ho}$, $\mathbf{W}_{ih}$, λ', $\mathbf{b}_h$ and $\mathbf{b}_o$. A TKNNLM with 100 hidden units needs around 1.01 million parameters for 10K vocabulary, which is half of that of a RNNLM with the same size of hidden layer. Perplexity comparisons with different number of parameters are shown in Fig.5. The proposed TKNNLM is more compact and efficient.

Table 1. The perplexity of state-of-the-art LMs reported on the test set of Penn Treebank Corpus[6].

Model	Perplexity	
	individual	+KN5
5-gram with Kneser-Ney smoothing	141.2	-
Log-bilinear LM	144.5	115.2
Feedforward Neural Network LM	140.2	116.7
Recurrent Neural Network LM	124.7	105.7
Temporal Kernel Neural Network LM	111.6	100.7

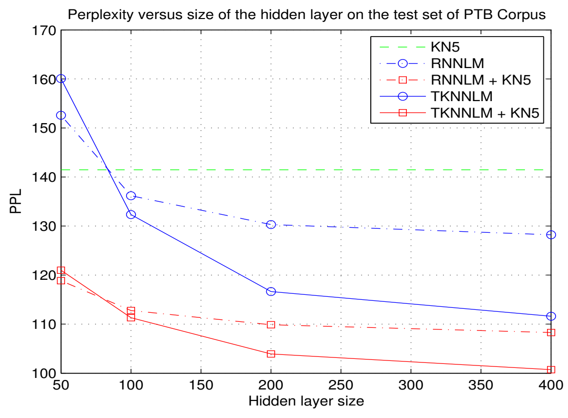

Fig. 4. *Perplexity comparisons with different size of the hidden layer on the test set of Penn Treebank Corpus. The PPL of KN5 is 141.46*

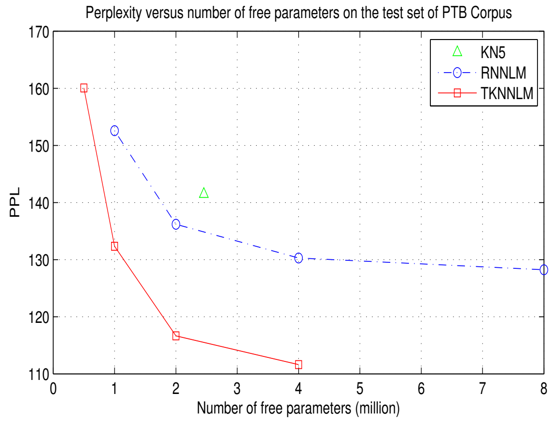

Fig. 5. *Perplexity comparisons with different number of free parameters on the test set of Penn Treebank Corpus. KN5 contains 2.45 million free parameters in total.*

5. WSJ ASR EXPERIMENT

In this section, the TKNNLM is applied to rescore the N-best in Wall Street Journal speech recognition task. To make the experimental results reproducible, publicly available models, data and tool, including the acoustic model, dictionary, training text for LMs and the decoder, are used as possible as we can. In the following experiments, the acoustic model is a speaker-independent crossword triphones with 8000 tied states and 32 gaussian mixtures per state trained on WSJ0, WSJ1 and TIMIT, which is publicly available [12]. The LMs' training data consists of 37M tokens from WSJ0 corpus, from which 1% is used as heldout data (0.37M words). The vocabulary is limited to 20K words used by HDecode [13]. A backoff trigram is smoothed according to the modified Kneser-Ney smoothing for decoding, where the pronouncing dictionary is from the CMU pronouncing dictionary [14]. In the experiment, 100-best hypotheses are generated from DARPA WSJ'92 (eval92) and WSJ'93 (eval93) data sets to be re-scored by different LMs. The interpolation weights are tuned on the eval92 set (333 sentences), and eval93 set used for evaluations consists of 213 sentences. As it is very time-consuming to train RNNLM or TKNNLM on large data, we randomly selected 10% tokens (3.7M) to train a RNNLM and a TKNNLM for comparisons, where 400 hidden units were used and a class layer with 300 classes was used for speeding up the training of RNNLM. Detailed results are shown in Table 2, where 'H400' denotes 400 hidden units, 'S0.1' means the 10% randomly selected data, and 'C300' means a class layer with 300 classes. Even though only 10% data is used, the improvement of the model interpolated with KN5 is obvious. The mixture of KN5 and TKNNLM obtains the best WER 10.3% in this setup.

Table 2. 100-best rescoring of eval92 and eval93 set in WSJ speech recognition task.

Model	Perplexity		Word Error Rate	
	heldout	eval92	eval92	eval93
One-best	-	-	9.4%	12.6%
KN5	80.05	111.50	8.4%	11.1%
RNNLMC300-H400-S0.1	121.00	153.25	9.5%	12.6%
+KN5	72.38	98.67	**7.7%**	10.6%
TKNNLM-H400-S0.1	104.32	130.58	9.0%	12.0%
+KN5	69.07	93.18	**7.7%**	**10.3%**

6. CONCLUSION AND FUTURE WORK

In this paper, a temporal kernel neural network langauge model is proposed to directly capture the long-term dependencies of words, where the memory of history is decayed exponentially. Experimental results show that the TKNNLM obtains a lower perplexity than the state-of-the-art results reported in PTB corpus, and the improvement in WSJ speech recognition task is significant, even though only 10% data is used. TKNNLM is very compact and efficient for language modeling. Additionally, in WSJ speech recognition, more training data of the TKNNLM is expected for better performance. A TKNNLM trained on large data needs to be compared with other state-of-the-art LMs in the future, especially the recurrent neural network with hash-based maximum entropy language model (RNNME)[7]. TKNNLM can also be easily extended with a maximum entropy model just like RNNME and better performance is expected. Last but not least, how to train the TKNNLM efficiently is a very important topic, which need to be further investigated. Although SGD is simple and efficient for large scale data, the hyper-parameters such as $initlr$, $lrmult$ need to be tuned carefully. Smarter optimization methods need to be explored.

7. REFERENCES

[1] Yoshua Bengio, Rejean Ducharme, Pascal Vincent, and Christian Jauvin, "A neural probabilistic language model," *Journal of Machine Learning Research (JMLR)*, pp. 1137–1155, 2003.

[2] Andriy Mnih and Geoffrey Hinton, "Tree new graphical models for statistical langauge modelling," in *Proc. of ICML*, 2007, pp. 641–648.

[3] Mikolov Tomas, Karafit Martin, Burget Lukas, Hanza Gernocky Jan, and Khudanpur Sanjeev, "Recurrent neural network based language model," in *Proc. of InterSpeech*, 2010, vol. 2010, pp. 1045–1048.

[4] Mikolov Tomas, Kombrink Stefan, Burget Lukas, Hanza Gernocky Jan, and Khudanpur Sanjeev, "Extensions of recurrent neural network language model," in *Proc. of ICASSP*, 2011.

[5] Martin Sundermeyer, Ralf Schlter, and Hermann Ney, "LSTM neural networks for language modeling," in *Proc. of InterSpeech*, 2012.

[6] Mikolov Tomas, Deoras Anoop, Kombrink Stefan, Burget Lukas, and Hanza Gernocky Jan, "Empirical evaluation and combination of advanced language modeling techniques," in *Proc. of InterSpeech*, 2011.

[7] Tomas Mikolov, *Statistical Language Models Based on Neural Netowrks*, Ph.D. thesis, Brno University of Technology (BUT), 2012, [Online] `http://www.fit.vutbr.cz/~imikolov/rnnlm/thesis.pdf`.

[8] Yoshua Bengio, Patrice Simard, and Paolo Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, mar 1994.

[9] Ilya Sutskever and Geoffrey E. Hinton, "Temporal-kernel recurrent neural networks," *Neural Networks*, vol. 23, pp. 239–243, 2010.

[10] Bo-June(Paul)Hsu, "MIT Langauge Modeling Tookit," [Available] `http://code.google.com/p/mitlm`.

[11] Mikolov Tomas, Deoras Anoop, Kombrink Stefan, Burget Lukas, and Hanza Gernocky Jan, "Rnnlm - recurrent neural network language modeling toolkit," in *Proc. of ASRU*, 2011, [Available] `http://www.fit.vutbr.cz/~imikolov/rnnlm/`.

[12] Keith Vertanen, "Baseline WSJ acoustic models for HTK and Sphinx: Training recipes and recognition experiments," Tech. Rep., 2006, [Available] `http://keithv.com/software/htk/us`.

[13] Steve Young, Gunnar Evermann, Mark Gales, Thomas Hain, Dan Kershaw, Xunying Liu, Gareth Moore, Julian Odell, Dave Ollason, Dan Povey, Valtcho Valtchev, and Phil Woodland, *The HTK book, version 3.4.1*, 2009.

[14] "The CMU Pronouncing Dictionary Release 0.7a," 2007, [Available] `http://www.speech.cs.cmu.edu/cgi-bin/cmudict`.