A NEURAL NETWORK APPROACH FOR MIXING LANGUAGE MODELS

Youssef Oualil, Dietrich Klakow

Spoken Language Systems (LSV)
Collaborative Research Center on Information Density and Linguistic Encoding
Saarland University, Saarbrücken, Germany
{firstname.lastname}@lsv.uni-saarland.de

ABSTRACT

The performance of Neural Network (NN)-based language models is steadily improving due to the emergence of new architectures, which are able to learn different natural language characteristics. This paper presents a novel framework, which shows that a significant improvement can be achieved by combining different existing heterogeneous models in a single architecture. This is done through 1) a feature layer, which separately learns different NN-based models and 2) a mixture layer, which merges the resulting model features. In doing so, this architecture benefits from the learning capabilities of each model with no noticeable increase in the number of model parameters or the training time. Extensive experiments conducted on the Penn Treebank (PTB) and the Large Text Compression Benchmark (LTCB) corpus showed a significant reduction of the perplexity when compared to state-of-the-art feedforward as well as recurrent neural network architectures.

***Index Terms*—** Neural networks, mixture models, language modeling

1. INTRODUCTION

For many language technology applications such as speech recognition [1] and machine translation [2], a high quality Language Model (LM) is considered to be a key component to success. Traditionally, LMs aim to predict probable sequences of predefined linguistic units, which are typically words. These predictions are guided by the semantic and syntactic properties that are encoded by the LM.

The recent advances in neural network-based approaches for language modeling led to a significant improvement over the standard N-gram models [3, 4]. This is mainly due to the continuous word representations they provide, which typically overcome the exponential growth of parameters that N-gram models require. The NN-based LMs were first introduced by Bengio et al. [5], who proposed a Feedforward Neural Network (FNN) model as an alternative to N-grams. Although FNNs were shown to perform very well for different tasks [6, 7], their fixed context (word history) size constraint was a limiting factor for their performance. In order to overcome this constraint, Mikolov et al. [8, 9] proposed a Recurrent Neural Network (RNN), which allows context information to cycle in the network. Investigating the inherent shortcomings of RNNs led to the Long-Short Term Memory (LSTM)-based LMs [10], which explicitly control the longevity of context information in the network. This chain of novel NN-based LMs continued with more complex and advanced models such as Convolutional Neural Networks (CNN) [11] and autoencoders [12], to name a few.

This research was funded by the German Research Foundation (DFG) as part of SFB 1102.

LMs performance has been shown to significantly improve using model combination. This is typically done by either 1) designing deep networks with different architectures at the different layers, as it was done in [11], which combines LSTM, CNN and a highway network, or by 2) combining different models at the output layer, as it is done in the maximum entropy RNN model [13], which uses direct N-gram connections to the output layer, or using the classical linear interpolation [14]. While the former category, requires a careful selection of the architectures to combine for a well-suited feature design, and can be difficult/slow to train, the second category knows a significant increase in the number of parameters when combining multiple models.

Motivated by the work in [13], we have recently proposed a Sequential Recurrent Neural Network (SRNN) [15], which combines FFN information and RNN. In this paper, we continue along this line of work by proposing a generalized framework to combine different heterogeneous NN-based architectures in a single mixture model. More particularly, the proposed architecture uses 1) a hidden feature layer to, separately, learn each of the models to be combined, and 2) a hidden mixture layer, which combines the resulting model features. Moreover, this architecture uses a single word embedding matrix, which is learned from all models, and a single output layer. This framework is, in principle, able to combine different NN-based LMs (e.g., FNN, RNN, LSTM, etc.) with no direct constraints on the number of models to combine or their configurations.

We proceed as follows. Section 2 presents an overview of the basic NN-based LMs. Section 3 introduces the proposed neural mixture model. Then, Section 4 evaluates the proposed network in comparison to different state-of-the-art language models for perplexity on the PTB and the LTCB corpus. Finally, we conclude in Section 5.

2. NEURAL NETWORK LANGUAGE MODELS

The goal of a language model is to estimate the probability distribution $p(w_1^T)$ of word sequences $w_1^T = w_1, \cdots, w_T$. Using the chain rule, this distribution can be expressed as

$$p(w_1^T) = \prod_{t=1}^{T} p(w_t | w_1^{t-1}) \tag{1}$$

Let U be a word embedding matrix and let W be the hidden-to-output weights. NN-based LMs (NNLMs), that consider word embeddings as input, approximate each of the terms involved in this product in a bottom-up evaluation of the network according to

$$H^t = \mathcal{M}(\mathcal{P}, \mathcal{R}^{t-1}, U) \tag{2}$$

$$O^t = g\left(H^t \cdot W\right) \tag{3}$$

where $\mathcal{M}$ represents a particular NN-based model, which can be a deep architecture, $\mathcal{P}$ denotes its parameters and $\mathcal{R}^{t-1}$ denotes its recurrent information at time t. $g(\cdot)$ is the softmax function.

The rest of this section briefly introduces $\mathcal{M}$, $\mathcal{P}$ and $\mathcal{R}^{t-1}$ for the basic architectures, namely FNN, RNN and LSTM, which were investigated and evaluated as different components in the proposed mixture model. The proposed architecture, however, is general and can include all NNLMs that consider world embeddings as input.

2.1. Feedforward Neural Networks

Similarly to N-gram models, FNN uses the Markov assumption of order $N-1$ to approximate (1). That is, the current word depends only on the last $N-1$ words. Subsequently, $\mathcal{M}$ is given by

$$E^{t-i} = X^{t-i} \cdot U \ , \qquad i = N-1, \cdots, 1 \tag{4}$$

$$H^t = f\left(\sum_{i=1}^{N-1} E^{t-i} \cdot V^i\right) \tag{5}$$

X^{t-i} is a one-hot encoding of the word w_{t-i}. Thus, E^{t-i} is the continuous representation of the word w_{t-i}. $f(\cdot)$ is an activation function. Hence, for an FNN model $\mathcal{M}$, $\mathcal{P} = \{V^i\}_{i=1}^{N-1}$ and $\mathcal{R}^{t-1} = \emptyset$.

2.2. Recurrent Neural Networks

RNN attempts to capture the complete history in a context vector h_t, which represents the state of the network and evolves in time. Therefore, RNN approximates each term in (1) as $p(w_t|w_1^T) \approx p(w_t|h_t)$. As a result, $\mathcal{M}$ for an RNN is given by

$$H^t = f\left(X^{t-1} \cdot U + H^{t-1} \cdot V\right) \tag{6}$$

Thus, for an RNN model $\mathcal{M}$, $\mathcal{P} = V$ and $\mathcal{R}^{t-1} = H^{t-1}$.

2.3. Long-Short Term Memory Networks

In order to alleviate the rapidly changing context issue in standard RNNs and control the longevity of the dependencies modeling in the network, the LSTM architecture [10] introduces an internal memory state C^t, which explicitly controls the amount of information, to forget or to add to the network, before estimating the current hidden state. Formally, an LSTM model $\mathcal{M}$ is given by

$$E^{t-1} = X^{t-1} \cdot U \tag{7}$$

$$\{i, f, o\}^t = \sigma\left(V_w^{i,f,o} \cdot E^{t-1} + V_h^{i,f,o} \cdot H^{t-1}\right) \tag{8}$$

$$\tilde{C}^t = f\left(V_w^c \cdot E^{t-1} + V_h^c \cdot H^{t-1}\right) \tag{9}$$

$$C^t = f^t \odot C^{t-1} + i^t \odot \tilde{C}^t \tag{10}$$

$$H^t = o^t \odot f\left(C^t\right) \tag{11}$$

where $\odot$ is the element-wise product, $\tilde{C}^t$ is the memory candidate, whereas i^t, f^t and o^t are the input, forget and output gates of the network, respectively. Hence, for an LSTM model $\mathcal{M}$, $\mathcal{R}^t = \{H^t, C^t\}$ and $\mathcal{P} = \{V_w^{i,f,o,c}, V_h^{i,f,o,c}\}$.

3. NEURAL NETWORK MIXTURE MODELS

On the contrary to a large number of research directions on improving or designing (new) particular neural architectures for language modeling, the work presented in this paper is an attempt to design a general architecture, which is able to combine different types of existing heterogeneous models rather than investigating new ones.

3.1. Model Combination for Language Modeling

The work presented in this paper is motivated by recent research showing that model combination can lead to a significant improvement in LM performance [14]. This is typically done by either 1) designing deep networks with different architectures at the different layers, as it was done in [11]. This category of model combination, however, requires a careful selection of the architectures to combine for a well-suited feature design, as it can be difficult/slow to train, whereas the second category 2) combines different models at the output layer, as it is done in the maximum entropy RNN model [13] or using the classical linear interpolation [14]. This category typically leads to a significant increase in the number of parameters when combining multiple models.

In a first attempt to circumvent these problems, we have recently proposed an SRNN model [15], which combines FFN information and RNN through additional sequential connections at the hidden layer. Although SRNN was successful and did not noticeably suffer from the aforementioned problems, it was solely designed to combine RNN and FNN and is, therefore, not well-suited for other architectures. This paper continues along this line of work by proposing a general architecture to combine different heterogeneous neural models with no direct constraints on the number or type of models.

3.2. Neural Network Mixture Models

This section introduces the mathematical formulation of the proposed mixture model. Let $\{\mathcal{M}_m\}_{m=1}^M$ be a set of M models to combine, and let $\{\mathcal{P}_m, \mathcal{R}_m^t\}_{m=1}^M$ be their corresponding model parameters and recurrent information at time t, respectively. For the basic NNLMs, namely FNN, RNN and LSTM, $\mathcal{M}_m$, $\mathcal{P}_m$ and $\mathcal{R}_m^t$ were introduced in Section 2.

Let U be the shared word embedding matrix, which is learned during training from all models in the mixture. The mixture model is given by the following steps (see illustration in Fig. 1):

1) Feature layer: update each model and calculate its features

$$H_m^t = \mathcal{M}_m(\mathcal{P}_m, \mathcal{R}_m^{t-1}, U), \quad m = 1, \cdots, M \tag{12}$$

2) Mixture layer: combine the different features

$$H_{mixture}^t = f_{mixture}\left(\sum_{m=1}^{M} H_m^t \cdot S_m\right) \tag{13}$$

3) Output layer: calculate the output using a softmax function

$$O^t = g\left(H_{mixture}^t \cdot W\right) \tag{14}$$

$f_{mixture}$ is a non-linear mixing function, whereas S_m, $m = 1, \cdots, M$ are the mixture weights (matrices).

Although the experiments conducted in this work mainly include FNN, RNN and LSTM, the set of possible model selection for $\mathcal{M}_m$ is not restricted to these but includes all NN-based models that take word embeddings as input.

The proposed mixture model uses a single word embedding matrix and a single output layer with predefined and fixed sizes. The latter are independent of the sizes of the mixture models. In doing so, this model does not suffer from the significant parameter growth when increasing the number of models in the mixture. We can also see that this architecture does not impose any direct constraints on the number of models to combine, their size or their type. Hence, we can combine, for instance, models of the same type but with different sizes/configurations, as we can combine heterogeneous models such as recurrent and non-recurrent models, in a single mixture.

Moreover, the mixture models can also be deep architectures with multiple hidden layers.

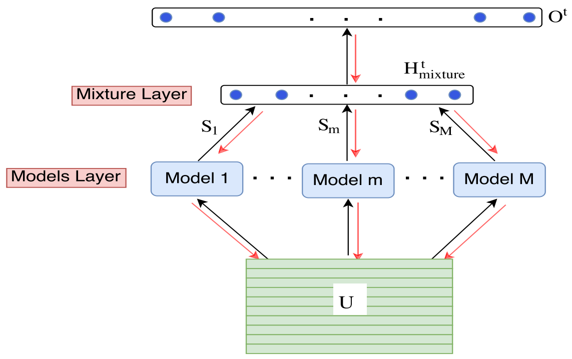

Fig. 1: Neural Mixture Model (NMM) architecture. Red (back) arrows show the error propagation during training.

3.3. Training of Neural Mixture Models

NMM training follows the standard back-propagation algorithm used to train neural architectures. More particularly, the error at the output layer is propagated to all models in the mixture. At this stage, each model receives a network error, updates its parameters, and propagates its error to the shared word embedding (input) layer. We should also mention here that recurrent models can be "unfolded" in time, independently of the other models in the mixture, as it is done for standard networks. Once each model is updated, the continuous word representations are then updated as well while taking into account the individual network errors emerging from the different models in the mixture (see illustration in Fig. 1).

The joint training of the mixture models is expected to lead to a "complementarity" effect. We mean by "complementarity" that the mixture models perform poorly when evaluated separately but lead to a much better performance when tested jointly. This is typically a result of the models learning and modeling, eventually, different features. Moreover, the joint learning is also expected to lead to a richer and more expressive word embeddings.

3.4. Model Dropout

In order to 1) enforce models co-training and 2) avoid network overfitting when the number of models in the mixture is large. We use a model dropout technique, which is inspired by the standard dropout regularization [16] that is widely used to train neural networks. The idea here is to have "models" replace "neurons" in the standard dropout. Therefore, for each training example, a model is to be dropped with a probability p_d. Then, only models that are selected contribute to the mixture and have their parameters and mixing weights S_m updated. Similarly to standard dropout, model dropout is applied only to non-recurrent models in the mixture.

4. EXPERIMENTS AND RESULTS

4.1. Experimental Setup

We evaluated the proposed architecture on two different benchmark tasks. The first set of experiments was conducted on the Penn Treebank (PTB) corpus using the standard division, e.g. [9, 17]; sections 0-20 are used for training while sections 21-22 and 23-24 are used for validation and testing. The vocabulary was limited to the 10k most frequent words while the remaining words were all mapped to the token <unk>. In order to evaluate how the proposed approach scales to large corpora, we run a set of experiments on the Large Text Compression Benchmark (LTCB) [18]. This corpus is based on the enwik9 dataset which contains the first 10^9 bytes of enwiki-20060303-pages-articles.xml. We adopted the same training-test-validation data split and pre-processing from [17]. The vocabulary was limited to the 80k most frequent words. Details about the sizes of these two corpora and the percentage of Out-Of-Vocabulary (OOV) words that were mapped to <unk> can be found in Table 1.

Table 1: *Corpus size in number of words and <unk> rate.*

	Train		Dev		Test	
Corpus	#W	<unk>	#W	<unk>	#W	<unk>
PTB	930K	6.52%	82K	6.47%	74K	7.45%
LTCB	133M	1.43%	7.8M	2.15%	7.9M	2.30%

The results reported below compare the proposed Neural Mixture Model (NMM) approach to the baseline NNLMs. In particular, we compare our model to the FNN-based LM [5], the full RNN [9] (without classes) as well as RNN with maximum entropy (RNNME) [13]. We also report results for the LSTM architecture [10], and the recently proposed SRNN model [15].

Although the proposed approach was not designed for a particular mixture of models, we only report results for different combinations of FNN, RNN and LSTM, which are considered to be the baseline NNLMs. For clarity, an NMM result is presented as $F^{N_1,\cdots,N_f}_{S_1,\cdots,S_f} + R_{S_1,\cdots,S_r} + L_{S_1,\cdots,S_l}$, where f is the number of FNNs in the mixture, $S_m, m = 1,\cdots,f$ are their corresponding hidden layer sizes (that are fed to the mixture) and $N_m, m = 1,\cdots,f$ are their fixed history sizes. The same notation holds for RNN and LSTM, where r and l are the number of RNNs and LSTMs in the mixture, respectively, and $N_r, N_l = 1$. The number of models in the mixture is given by $f + r + l$. Moreover, the notation $F^{N_b-N_e}_{S_f}$ means that this model combines $N_e - N_b + 1$ consecutive FNN models with respective history sizes $N_b, N_b+1,\cdots, N_e$, with all models having the same hidden layer size S_f.

4.2. PTB Experiments

For the PTB experiments, all models have a hidden layer size of 400, with FFNN and SRNN using the Rectified Linear Unit (ReLu) i.e., $f(x) = max(0, x)$ as activation function and having 2 hidden layers. ReLu is also used as activation function for the mixture layer in NMMs, which use a single hidden layer. The embedding size is 100 for SRNN and NMMs, whereas it is set to 400 for RNN and 200 for FNN and LSTM. The training is performed using the stochastic gradient descent algorithm with a mini-batch size of 200. the learning rate is initialized to 0.4, the momentum is set to 0.9, the weight decay is fixed at 4x10^{-5}, the model dropout is set to 0.4 and the training is done in epochs. The weights initialization follows the normalized initialization proposed in [19]. Similarly to [8], the learning rate is halved when no significant improvement in the log-likelihood of the validation data is observed. The BPTT was set to 5 time steps for all recurrent models. In the tables below, the results are reported in terms of perplexity (PPL), Number of model Parameters (NoP) and the Parameter Growth (PG) for NMM, which is defined as the relative increase in the number of parameters of NMM w.r.t. the baseline model in the table. In order to demonstrate the power of the joint training, we also report the perplexity PPL and NoP of the Linearly Interpolated (LI) models in the mixture after training them separately. In this case, each model learns its own word embedding and output layer.

Table 2: *LMs performance on the PTB test set.*

model	PPL	NoP	PG	PPL(LI)	NoP(LI)
FNN (N=5)	114	6.49M	—	—	—
$F_{200}^{2,3}$	117	5.27M	-18.80%	120.0	6.10M
F_{200}^{2-5}	110	5.61M	-13.56%	112.0	12.28M
LSTM	105	6.97M	—	—	—
$L_{100} + F_{200}^{2}$	**102**	5.25M	-24.68%	114	5.12M
$L_{100} + R_{100}$	**102**	5.18M	-25.68%	118	4.09M
RNN	117	8.17M	—	—	—
$R_{100} + F_{200}^{2}$	109	5.18M	-36.60%	119	5.05M
$R_{100} + F_{200}^{2-6}$	105	5.86M	-28.27%	108	17.41M
RNNME	117	10G	—	—	—
WD-SRNN	104	6.33M	—	—	—
WI-SRNN	104	5.33M	—	—	—

The PTB results reported in Table 2 show clearly that combining different small-size models with a reduced word embedding size results in a better perplexity performance compared to the baseline models, with a significant decrease in the NoP required by the mixture. More particularly, we can see that adding a single FNN model to a small size LSTM or RNN is sufficient to outperform the baseline models while reducing the number of parameters by 24% and 36%, respectively. The same conclusion can be drawn when combining an RNN with an LSTM. We can also see that adding more FNN models to each of these mixtures leads to additional improvements while keeping the number of parameters significantly small. Table 2 also shows that training the small size models (in the mixture) separately, and then linearly interpolating them, results in a slightly worse performance compared to the mixture model with a noticeable increase in the NoP. This conclusion emphasizes the importance of the joint training. Moreover, we can also see that mixing RNN and FNNs leads to a comparable performance to SRNN, which was particularly designed to enhance RNN with FNN information. The proposed approach, however, does not particularity encode the individual characteristics of the models in the mixture, which reflects its ability to include different types of NNLMs. We can also conclude that combining FNN with recurrent models leads to a more significant improvement when compared to mixtures of FNNs. This conclusion shows, similarly to other work e.g. [15, 13], that recurrent models can be further improved using N-gram/feedforward information, given that they model different linguistic features.

Fig. 2 is an extension of Table 2, which shows the change in the perplexity and NoP of different NMMs when iteratively adding more FFN models to the mixture. This figure confirms that combining heterogeneous models (combining LSTM or RNN with FNNs) achieves a better performance compared to combining only FNN models. We can also conclude from this figure that the improvement becomes very slow after adding 4 FNN models to each mixture.

4.3. LTCB Experiments

The LTCB experiments use the same PTB setup with minor changes. The results shown in Table 3 follow the same experimental setup used in [15]. More precisely, these results were obtained without usage of momentum, model dropout or weight decay whereas the

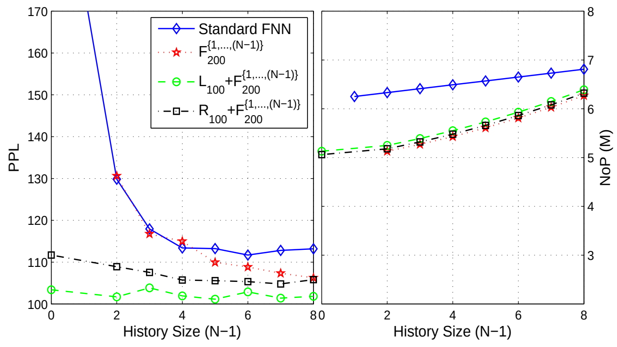

Fig. 2: Perplexity vs parameter growth of different mixture models while iteratively adding more FNN models to the mixture.

mini-batch size was set to 400. The FNN architecture contains 2 hidden layers of size 600 whereas RNN, LSTM, SRNN and NMM have a single hidden layer of size 600.

Table 3: *LMs Perplexity on the LTCB test set.*

model	PPL	NoP	PG
FNN[4*200]-600-600-80k	110	64.92M	—
F_{600}^{2-4}	102	66.24M	2.03%
F_{100}^{2-7}	92	64.98M	0.09%
RNN[600]-600-80k	85	96.44M	—
$R_{200} + F_{400}^{4}$	84	64.80M	-32.81%
$R_{200} + F_{600}^{2-4}$	77	66.40M	-31.15%
LSTM[600]-600-80k	66	66.00M	—
$L_{400} + R_{200}$	64	65.44M	-1.51%
$L_{300} + F_{300}^{4}$	64	65.28M	-1.75%
$L_{600} + F_{600}^{4}$	**58**	67.21M	1.16%
WI-SRNN[4*200]-600-80k	77	64.56M	—
WD-SRNN[4*200]-600-80k	72	80.56M	—

The LTCB results shown in Table 3 generally confirm the PTB conclusions. In particular, we can see that combining recurrent models, with half (third for RNN) their original size, with a single FNN model leads to a comparable performance to the baseline models. Moreover, increasing the mixture models size (for LSTM) or increasing the number of FNNs (for RNN) improves the performance further with no noticeable increase in the NoP. Similarly to the PTB, we can also see that NMM achieves the same performance as the WI-SRNN model with a NoP reduction of 31% compared to the original RNN model. The FFN mixture results show a more significant improvement when combining multiple small-size (100) models compared to mixing few large models (600). This conclusion shows that the strength of mixture models lies in their ability to combine the learning capabilities of different models, even with small sizes.

5. CONCLUSION AND FUTURE WORK

We have presented a neural mixture model which is able to combine heterogeneous NN-based LMs in a single architecture. Experiments on PTB and LTCB corpora have shown that this architecture substantially outperforms many state-of-the-art neural systems, due to its ability to combine learning capabilities of different architectures. Further gains could be made using a more advanced model selection or feature combination at the mixing layer instead of the simple model weighting. These will be investigated in future work.

6. REFERENCES

[1] Slava M. Katz, "Estimation of probabilities from sparse data for the language model component of a speech recognizer," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 35, no. 3, pp. 400–401, Mar. 1987.

[2] Peter F. Brown, John Cocke, Stephen A. Della Pietra, Vincent J. Della Pietra, Fredrick Jelinek, John D. Lafferty, Robert L. Mercer, and Paul S. Roossin, "A statistical approach to machine translation," *Computational Linguistics*, vol. 16, no. 2, pp. 79–85, Jun. 1990.

[3] Ronald Rosenfeld, "Two decades of statistical language modeling: where do we go from here?," *Proceedings of the IEEE*, vol. 88, no. 8, pp. 1270–1278, Aug. 2000.

[4] Reinhard Kneser and Hermann Ney, "Improved backing-off for m-gram language modeling," in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Detroit, Michigan, USA, May 1995, pp. 181–184.

[5] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin, "A neural probabilistic language model," *Journal of Machine Learning Research*, vol. 3, pp. 1137–1155, Mar. 2003.

[6] Holger Schwenk and Jean-Luc Gauvain, "Training neural network language models on very large corpora," in *Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Oct. 2005, pp. 201–208.

[7] Joshua Goodman, "A bit of progress in language modeling, extended version," Tech. Rep. MSR-TR-2001-72, Microsoft Research, 2001.

[8] Tomas Mikolov, Martin Karafiát, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur, "Recurrent neural network based language model," in *11th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, Makuhari, Chiba, Japan, Sep. 2010, pp. 1045–1048.

[9] Tomas Mikolov, Stefan Kombrink, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur, "Extensions of recurrent neural network language model," in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Prague, Czech Republic, May 2011, pp. 5528–5531.

[10] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney, "LSTM neural networks for language modeling," in *13th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, Portland, Oregon, USA, Sep. 2012, pp. 194–197.

[11] Kim Yoon, Jernite Yacine, Sontag David, and Rush Alexander M., "Character-aware neural language models," in *30th AAAI Conference on Artificial Intelligence*, 2016.

[12] Sarath Chandar A P, Stanislas Lauly, Hugo Larochelle, Mitesh Khapra, Balaraman Ravindran, Vikas C Raykar, and Amrita Saha, "An autoencoder approach to learning bilingual word representations," in *Advances in Neural Information Processing Systems 27*, pp. 1853–1861. 2014.

[13] Tomas Mikolov, Anoop Deoras, Daniel Povey, Lukás Burget, and Jan Cernocký, "Strategies for training large scale neural network language models," in *IEEE Workshop on Automatic Speech Recognition & Understanding (ASRU)*, Waikoloa, Hawaii, USA, Dec. 11-15, 2011, pp. 196–201.

[14] Tomas Mikolov, Anoop Deoras, Stefan Kombrink, Lukás Burget, and Jan Cernocký, "Empirical evaluation and combination of advanced language modeling techniques," in *12th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, Florence, Italy, Aug. 27-31, 2011, pp. 605–608.

[15] Youssef Oualil, Clayton Greenberg, Mittul Singh, and Dietrich Klakow, "Sequential recurrent neural network for language modeling," in *17th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, San Francisco, California, USA, Sep. 2016.

[16] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting.," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[17] Shiliang Zhang, Hui Jiang, Mingbin Xu, Junfeng Hou, and Li-Rong Dai, "The fixed-size ordinally-forgetting encoding method for neural network language models," in *53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing ACL*, July 2015, vol. 2, pp. 495–500.

[18] Matt Mahoney, "Large text compression benchmark," 2011.

[19] Xavier Glorot and Yoshua Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, Chia Laguna Resort, Sardinia, Italy, May 2010, pp. 249–256.