Improved Learning through Augmenting the Loss

Hakan Inan
inanh@stanford.edu

Khashayar Khosravi
khosravi@stanford.edu

Abstract

We present two improvements to the well-known Recurrent Neural Network Language Models(RNNLM). First, we use the word embedding matrix to project the RNN output onto the output space and already achieve a large reduction in the number of free parameters while still improving performance. Second, instead of merely minimizing the standard cross entropy loss between the prediction distribution and the "one-hot" target distribution, we minimize an additional loss term which takes into account the inherent metric similarity between the target word and other words. We show with experiments on the Penn Treebank Dataset that our proposed model (1) achieves significantly lower average word perplexity than previous models with the same network size and (2) achieves the new state of the art by using much fewer parameters than used in the previous best work.

1 Background & Introduction

Language Modeling, which is the task of determining the probability that a sequence of words occur in a sentence, is an important task in Natural Language Processing(NLP). Its scope includes a large variety of tasks such as speech recognition, machine translation, part of speech tagging, and spelling correction [1]. Old statistical approaches for addressing this problem were based on n-grams and their associated properties. Although these methods seemed to work reasonably well, they needed a larger value of n in order to capture a better model for the language, and they needed a number of parameters exponential in the size of the vocabulary [2]. Researchers have proposed methods to improve these models [1]; nonetheless, they are still difficult to implement in practice.

Recently, with the advances in neural network research and the increased availability of faster computers, neural networks have started to replace the old methods for capturing the language structure. Recurrent Neural Networks(RNN) were first proposed in [3] for addressing the language modelling task and attained a reasonably satisfactory performance. The follow-up work [4] improved the result by applying back-propagation in addition to the existing forward-propagation.

In [5], Zaremba et. al. used Long Short-Term Memory(LSTM) which were fist introduced in [6], as the building block of their recurrent neural network for language modelling. Furthermore, the authors showed that appropriat application of dropout may avoid overfitting and lead to a better and more generalizable model. The authors were further able to achieve the state of the art for language modeling on the Penn Dataset Treebank corpus[7]. In particular, by training a two-layer LSTM Recurrent Neural Network with 1500 hidden states and appropriate dropout, they could reduce the test perplexity to 78.4 for a single model.

However, Recurrent Neural Network Language Models need big models for proper training (2 RNN layers with 1500 hidden units in each in [5]). In addition, their performance rely on training two very large matrices, one for word embeddings and one for output projection, both of which essentially capture the same thing: projection from token space to word vector space and vice versa. In this paper we explain how imposing this structure may help us to achieve a much better accuracy even on smaller networks. We also propose a novel method for augmenting the loss function with ad additional term which utilizes the similarity between words captured in word embeddings.

In the remainder of the paper, we proceed as follows. In Section 2, we introduce our notation and give detailed explanation of the standard RNNLM. In Section 3, we introduce our method and explain in detail the proposed model mechanism. In Section 4, we validate our model by running experiments on the Penn Treebank corpus [7] and show that it achieves the new state of the art for this dataset. In Section 5, we discuss some of the future directions and we present our conclusions in Section 6.

The code to replicate the results shown in this paper can be found in `https://github.com/inanhkn/informed-rnnlm`

2 Problem Statement

In language modeling, the task is to build a model which can predict the next word in a sentence based on all previous words.

Let V denote our vocabulary and N be the number of samples (words) that we have. Let u_t and y_t be the $|V|$ dimensional one-hot input and target word tokens at time t, respectively. In language modeling, we typically have $y_t = u_{t+1}$, since at each time the target is the next word of the sentence.

Figure 1 shows the structure of a standard RNNLM. The blocks labeled f are the hidden units of the RNN, and they can be Gated Recurrent Units (GRU) [8, 9], LSTMs, or even stacked LSTMs. We shall denote the predictor part of the model (i.e. part which outputs $\hat{y}'s$) with $\mathcal{M}(L, f, U, b)$. The following equations describe the model mathematically:

$$x_t = Lu_t \qquad \text{(Embedding)} \tag{1}$$

$$[h_t, m_t] = f(x_t, m_{t-1}) \qquad \text{(Recurrent Neural Network)} \tag{2}$$

$$z_t = Uh_t + b \qquad \text{(Output Projection)} \tag{3}$$

$$\hat{y}_{t,i} = \frac{\exp\left(z_{t,i}\right)}{\sum_{j=1}^{|V|} \exp\left(z_{t,j}\right)} \qquad \text{(Softmax Layer)} \tag{4}$$

where m_t is the memory at time t which depends on the choice of hidden units for each layer, h_t is the final RNN output, and the network(function) f basically is the core of the structure which performs the predictive loop. For instance, in a simple one-layer RNN the following equation holds:

$$h_t = \sigma\left(W^{hh}h_{t-1} + W^{hx}x_t\right).$$

Therefore, by defining $m_t = h_t$ and $f(x, u) = \sigma(W^{hh}u + W^{hx}x)$, we arrive at the simple RNN model. After creating the model, a cost function is to be defined for the learning task. Most commonly, the *Cross Entropy* loss between the prediction distribution $\hat{y}_t$ and the target distribution y_t is used to capture the prediction error and is defined as

$$CE(y_t, \hat{y}_t) = \sum_{j=1}^{|V|} y_{t,j} \log(\hat{y}_{t,j}), \tag{5}$$

Since we have N samples, we minimize the sum of cross entropy loss over all samples:

$$\text{Loss} = \sum_{t=1}^{N} CE(y_t, \hat{y}_t).$$

3 Our Approach

In this section, we describe the two modifications we make on the classical RNNLM.

3.1 Projection onto the Output Token Space Through the Embedding Matrix

In language modeling, the targets and the inputs have the same number of dimensions($|V|$), and they could naturally be considered to live in the same metric space. Based on this idea, we make the

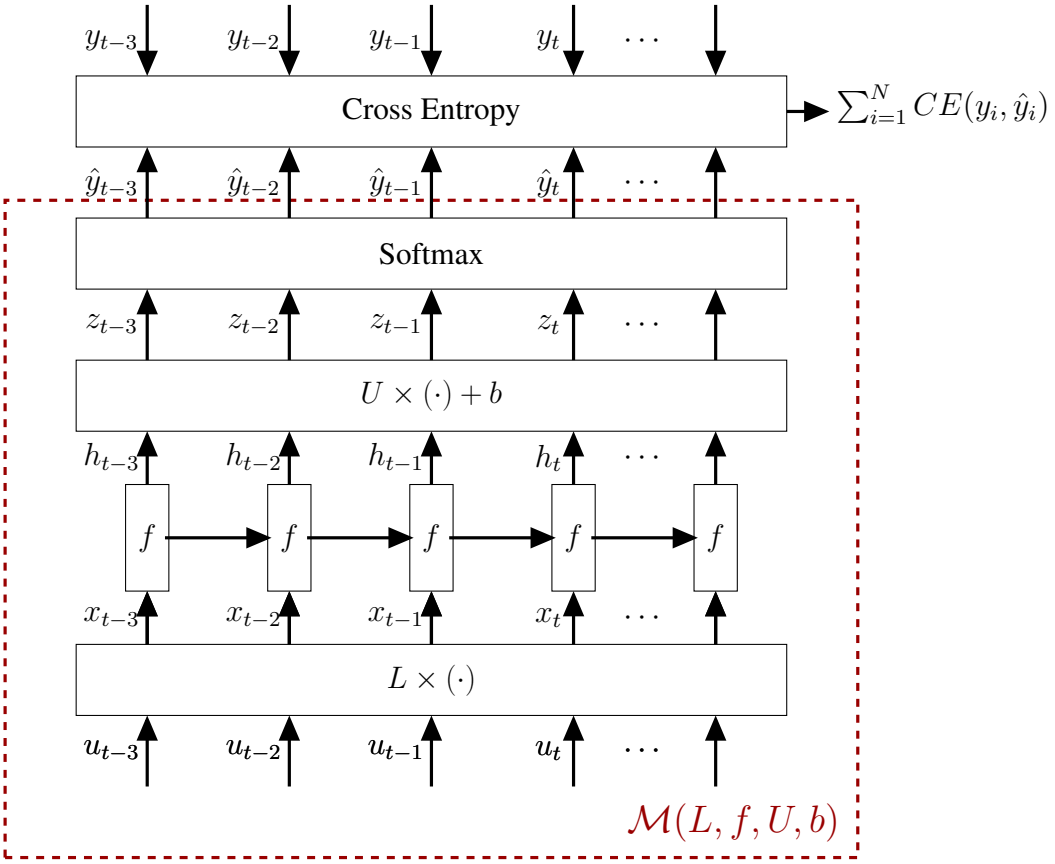

Figure 1: Typical RNNLM unrolled in time

following modifications: We first constrain h_t to be of the same dimension as x_t. We then modify the output projection to be as follows:

$$z_t = L^T h_t. \tag{6}$$

This achieves two things: First, it forces the RNN output to be in the same space as the input word vectors. This is because (6) essentially takes inner product of the RNN output with all the other word vectors, and the prediction distribution $\hat{y}_t$ will have its maximum weight at the index where the corresponding word vector has the maximum inner product with h_t (maximum vector similarity). Second, eliminating the U matrix significantly cuts down the number of variables in the model, especially for very large vocabularies.

3.2 Augmenting the Loss

In training multi-class classification models, the standard objective is to match the prediction distribution, $\hat{y}$ with the target distribution, y through minimizing the cross entropy loss. However, the target distribution is classically a "one-hot" distribution; that is, it is one at the entry which points to the true target and zero everywhere else. On the other hand, in most cases it might be desirable to have a target distribution which is more informed about the statistical relationships between the targets. In our example of language modeling , for instance, the word "terrific" in the sentence "this is a terrific method" could be replaced with the word "fantastic" without having a notable alteration in the sentence meaning. Motivated by this fact, we introduce a more informative target distribution,

$\tilde{y}_t$ at time t, as follows:

$$v_t = L^T(Ly_t), \tag{7}$$

$$\tilde{y}_{t,i} = \frac{\exp(v_{t,i})}{\sum_{j=1}^{|V|} \exp(v_{t,j})}. \tag{8}$$

Essentially, (7) looks up the word vector for the target word and takes the inner product of that word vector with all other word vectors, and (8) turns these inner products to a probability distribution by applying the softmax function. Note that the calculation of $\tilde{y}_t$ (equations (7) and (8)) is very similar in structure to that of the prediction distribution $\hat{y}_t$ (equations (6) and (4)). This similarity underlies the overarching idea of our work, which is to have the target and prediction at the word-vector level and then to project them onto the token space through utilizing the metric space captured by the word embeddings.

For computing the augmented loss at time t, we compute the KL-distance between the prediction distribution $\hat{y}_t$ and the informative target distribution $\tilde{y}_t$. The final loss is a linear combination of this loss and the standard loss:

$$\text{Loss} = CE(y_t, \hat{y}_t) + \alpha D_{KL}(\tilde{y}_t || \hat{y}_t). \tag{9}$$

α is a model hyper-parameter that controls the relative weight of the augmented loss in the overall loss. Note that in (9), theoretically speaking, we are not able to replace the second term (KL divergence) with cross-entropy since

$$D_{KL}(\tilde{y}_t || \hat{y}_t) = \sum_i \tilde{y}_{t,i} \log \frac{\tilde{y}_{t,i}}{\hat{y}_{t,i}} = \sum_i \tilde{y}_{t,i} \log \tilde{y}_{t,i} - \sum_i \tilde{y}_{t,i} \log \hat{y}_t = CE(y, \hat{y}_t) - H(\tilde{y}_t),$$

and $H(\tilde{y}_t)$ contains L, which is a variable we wish to optimize for. However, in practice we did not observe a significant disparity in performance when we replaced the KL-divergence term with cross-entropy. Our proposed model with the two improvements is depicted in Figure 2.

Before concluding this section, we would like to give a unified mechanistic account of our proposed changes to the RNNLM. By constraining the RNN output to be the same size as the input word vectors and by using the same matrix (embedding matrix, L) to project from token space to RNN state space and vice versa, we are forcing the final RNN output to live in the word vector space. At the same time, we project the target word token onto the same word vector space and compute similarities to all the other words in that space (the same as what we do with the RNN output) to yield a distributed target probability distribution which utilizes the information stored in the word vector space. When the loss calculated with this distribution is combined with the standard loss, model incurs relatively less loss by predicting words which are similar (in the language modeling scope) to the observed target word although not exactly the same, hence it receives improved supervision.

4 Experiments

4.1 Dataset

We evaluate our method on the Penn Treebank corpus [7], which has been widely used for benchmarking language models. The dataset contains 929k training words, 73k validation words, and 82k test words. The size of vocabulary for this dataset is $|V| = 10000$. This dataset is available online and can be downloaded here [1]

4.2 Results

For fair comparison of our model to the current state of the art, we constructed networks very similar in architecture to those published in [5], where the authors achieved the state of the art on the Penn Treebank corpus. Specifically, we trained 2-layer LSTMs with equal number of hidden units, and applied dropout to the inputs and to the output of each hidden layer as described in [5]. Again, as

[1] www.fit.vutbr.cz/ imikolov/rnnlm/simple-examples.tgz

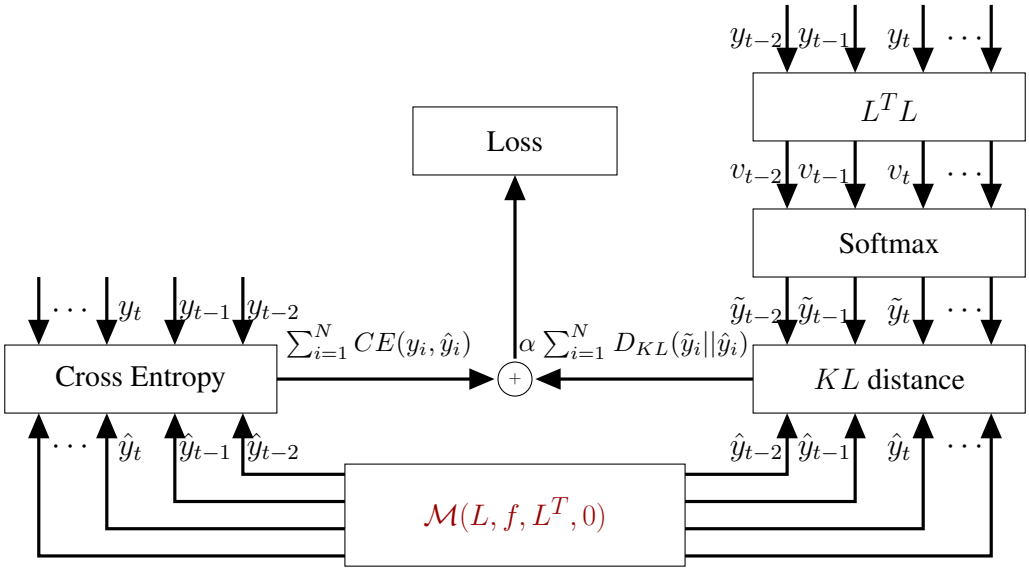

Figure 2: Proposed Structure

was done in [5], we set the input word vector dimension to be the same as the number of hidden units.

As a first experiment, we tested the improvement contributed by our two proposed modifications individually relative to baseline. Note that although the two proposed changes are not entirely independent (see discussion in Section 3.2), we could in principle still study them in isolation. For this, we constructed a baseline network with a 2-layer LSTM with 200 hidden units in each layer. We tuned the model parameters to achieve the best validation perplexity. We used Adam optimizer with the parameters recommended in [10], performed truncated backpropagation for 20 time steps, and chose to use keep probability of 0.5 for dropout. We then separately constructed a network where we modify the baseline network by using the embedding matrix in the output projection step (equation 6), and one where we augment the loss with our proposed loss term (equation 9) with $\alpha = 0.3$. Finally, we constructed a network with both our proposed modifications. The results are listed in Table 1.

In our second experiment, we compared our method with both proposed modifications to those presented in [5]. We trained 2-layer LSTM models with 200,300,650, and 750 hidden units. For the networks with 200 and 300 hidden units, we unrolled the network for 20 time steps, used batches of size 50, and used Adam optimizer. For the networks with 650 and 750 hidden units, we unrolled the network for 35 time steps, used batches of size 20, and used stochastic gradient descent as the optimizer. In stochastic gradient descent, we set the initial learning rate to 1 and used a rate decay of 0.83 starting from the 9th training epoch. In the small networks (200 and 300 hidden units), we also tried replacing the KL-divergence term in (9) with cross entropy and found that there are no notable differences between using the two (Table 2), hence we opted for cross-entropy for the large (650 and 750 hidden units) networks for the sake of runtime. The results of this experiment are listed in Table 2.

Overall, we see that using our proposed approach, with networks as small as 300 hidden units in each LSTM layer we attain perplexity close to that achieved with 650 hidden units in each LSTM layer previously (84.7 versus 82.7). Furthermore, with medium sized networks (650 and 750 hidden units), we beat the previous state of the art which was achieved by using significantly larger number of RNN units (1500 hidden units).

Due to our limited computational resources, we were not able to explore the large parameter regime; however based on our assessment of the results of incrementally growing the network size, we believe that further increase in network size will have only a modest decrease in the average word perplexity for the Penn Treebank corpus.

Model	Test Perplexity	Configuration
200 hidden units	104.5	batch size:50, optimizer:Adam dropout: 0.5
200 hidden units only L^T in output projection	97.6	batch size:50, optimizer:Adam dropout: 0.7
200 hidden units only augmented KL loss	98.2	batch size:50, optimizer:Adam dropout: 0.7, $\alpha = 0.3$
200 hidden units both proposed modifications	90	batch size:50, optimizer:Adam dropout: 0.7, $\alpha = 0.3$

Table 1: Comparison of proposed modifications to baseline in the Penn Treebank Corpus

Model	Test Perplexity	Configuration
200 hidden units [5]	114.5	batch size:20, optimizer:SGD dropout: None
200 hidden units Proposed-$KL(CE)$	90 (89.6)	batch size:50, optimizer:Adam dropout: 0.7, $\alpha = 0.3$
300 hidden units Proposed-$KL(CE)$	84.9 (84.7)	batch size:50, optimizer:Adam dropout: 0.6, $\alpha = 0.3$
650 hidden units [5]	82.7	batch size:20, optimizer:SGD dropout: 0.5
1500 hidden units [5]	78.4	batch size:20, optimizer:SGD dropout: 0.35
650 hidden units Proposed-CE	78	batch size:20, optimizer:SGD dropout: 0.5, $\alpha = 0.35$
750 hidden units Proposed-CE	**76.7**	batch size:20, optimizer:SGD dropout: 0.5, $\alpha = 0.15$

Table 2: Comparison of our method to previous best work in the Penn Treebank Corpus

5 Possible Extensions

We believe that the idea we introduce in this paper is versatile and could be deployed in various other settings where inputs and outputs live in the same space. Natural language processing field in general is naturally a very suitable host for such a scheme as often times one is interested in having words as the output. Machine translation for instance is an immediate candidate that we would like to use the method we introduce here.

6 Conclusions

We presented a novel loss scheme which utilizes the information already present in the word representations of a language model to improve learning. We showed that this scheme helps even smaller models to learn and generalize much better compared to the previously reported networks of the similar sizes, and that it pushes the state of the art in relatively medium sized models.

References

[1] Joshua T Goodman. A bit of progress in language modeling. *Computer Speech & Language*, 15(4):403–434, 2001.

[2] Tomáš Mikolov. Statistical language models based on neural networks. *Presentation at Google, Mountain View, 2nd April*, 2012.

[3] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *INTERSPEECH*, volume 2, page 3, 2010.

[4] Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Honza Černockỳ, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 5528–5531. IEEE, 2011.

[5] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

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

[7] Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

[8] KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, abs/1409.1259, 2014.

[9] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

[10] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.