BLACKOUT: SPEEDING UP RECURRENT NEURAL NETWORK LANGUAGE MODELS WITH VERY LARGE VOCABULARIES

Shihao Ji
Parallel Computing Lab, Intel
shihao.ji@intel.com

S. V. N. Vishwanathan
Univ. of California, Santa Cruz
vishy@ucsc.edu

Nadathur Satish, Michael J. Anderson & Pradeep Dubey
Parallel Computing Lab, Intel
{nadathur.rajagopalan.satish,michael.j.anderson,pradeep.dubey}@intel.com

arXiv:1511.06909v7 [cs.LG] 31 Mar 2016

ABSTRACT

We propose *BlackOut*, an approximation algorithm to efficiently train massive recurrent neural network language models (RNNLMs) with million word vocabularies. BlackOut is motivated by using a discriminative loss, and we describe a weighted sampling strategy which significantly reduces computation while improving stability, sample efficiency, and rate of convergence. One way to understand BlackOut is to view it as an extension of the DropOut strategy to the output layer, wherein we use a discriminative training loss and a weighted sampling scheme. We also establish close connections between BlackOut, importance sampling, and noise contrastive estimation (NCE). Our experiments, on the recently released one billion word language modeling benchmark, demonstrate scalability and accuracy of BlackOut; we outperform the state-of-the art, and achieve the lowest perplexity scores on this dataset. Moreover, unlike other established methods which typically require GPUs or CPU clusters, we show that a carefully implemented version of BlackOut requires only 1-10 days on a single machine to train a RNNLM with a million word vocabulary and billions of parameters on one billion words. Although we describe BlackOut in the context of RNNLM training, it can be used to any networks with large softmax output layers.

1 INTRODUCTION

Statistical language models are a crucial component of speech recognition, machine translation and information retrieval systems. In order to handle the data sparsity problem associated with traditional n-gram language models (LMs), neural network language models (NNLMs) (Bengio et al., 2001) represent the history context in a continuous vector space that can be learned towards error rate reduction by sharing data among similar contexts. Instead of using fixed number of words to represent context, recurrent neural network language models (RNNLMs) (Mikolov et al., 2010) use a recurrent hidden layer to represent longer and variable length histories. RNNLMs significantly outperform traditional n-gram LMs, and are therefore becoming an increasingly popular choice for practitioners (Mikolov et al., 2010; Sundermeyer et al., 2013; Devlin et al., 2014).

Consider a standard RNNLM, depicted in Figure 1. The network has an input layer x, a hidden layer s (also called context layer or state) with a recurrent connection to itself, and an output layer y. Typically, at time step t the network is fed as input $x_t \in \mathbb{R}^V$, where V denotes the vocabulary size, and $s_{t-1} \in \mathbb{R}^h$, the previous state. It produces a hidden state $s_t \in \mathbb{R}^h$, where h is the size of the hidden layer, which in turn is transformed to the output $y_t \in \mathbb{R}^V$. Different layers are fully connected, with the weight matrices denoted by $\Omega = \{W_{in}^{V\times h}, W_r^{h\times h}, W_{out}^{V\times h}\}$.

For language modeling applications, the input x_t is a sparse vector of a 1-of-V (or one-hot) encoding with the element corresponding to the input word w_{t-1} being 1 and the rest of components of x_t set to 0; the state of the network s_t is a dense vector, summarizing the history context $\{w_{t-1}, \cdots, w_0\}$

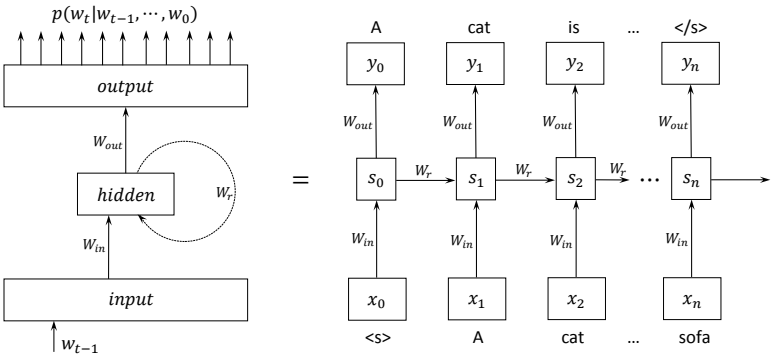

Figure 1: The network architecture of a standard RNNLM and its unrolled version for an example input sentence: <s> A cat is sitting on a sofa </s>.

preceding the output word w_t; and the output y_t is a dense vector, with the i-th element denoting the probability of the next word being w_i, that is, $p(w_i|w_{t-1}, \cdots, w_0)$, or more concisely, $p(w_i|s_t)$. The input to output transformation occurs via:

$$s_t = \sigma(W_{in}^T x_t + W_r s_{t-1}) \tag{1}$$
$$y_t = f(W_{out} s_t), \tag{2}$$

where $\sigma(v) = 1/(1 + \exp(-v))$ is the sigmoid activation function, and $f(\cdot)$ is the softmax function $f(u_i) := \exp(u_i)/\sum_{j=1}^{V} \exp(u_j)$.

One can immediately see that if x_t uses a 1-of-V encoding, then the computations in equation (1) are relatively inexpensive (typically h is of the order of a few thousand, and the computations are $\mathcal{O}(h^2)$), while the computations in equation (2) are expensive (typically V is of the order of a million, and the computations are $\mathcal{O}(Vh)$). Similarly, back propagating the gradients from the output layer to the hidden layer is expensive. Consequently, the training times for some of the largest models reported in literature are of the order of weeks (Mikolov et al., 2011; Williams et al., 2015).

In this paper, we ask the following question: Can we design an approximate training scheme for RNNLM which will improve on the state of the art models, while using significantly less computational resources? Towards this end, we propose *BlackOut* an approximation algorithm to efficiently train massive RNNLMs with million word vocabularies. BlackOut is motivated by using a discriminative loss, and we describe a weighted sampling strategy which significantly reduces computation while improving stability, sample efficiency, and rate of convergence. We also establish close connections between BlackOut, importance sampling, and noise contrastive estimation (NCE) (Gutmann & Hyvärinen, 2012; Mnih & Teh, 2012), and demonstrate that BlackOut mitigates some of the limitations of both previous methods. Our experiments, on the recently released one billion word language modeling benchmark (Chelba et al., 2014), demonstrate scalability and accuracy of BlackOut; we outperform the state-of-the art, achieving the lowest perplexity scores on this dataset. Moreover, unlike other established methods which typically require GPUs or CPU clusters, we show that a carefully implemented version of BlackOut requires only 1-10 days on a single CPU machine to train a RNNLM with a million word vocabulary and billions of parameters on one billion words.

One way to understand BlackOut is to view it as an extension of the DropOut strategy (Srivastava et al., 2014) to the output layer, wherein we use a discriminative training loss and a weighted sampling scheme. The connection to DropOut is mainly from the way they operate in model training and model evaluation. Similar to DropOut, in BlackOut training a subset of output layer is sampled and trained at each training batch and when evaluating, the full network participates. Also, like DropOut, a regularization technique, our experiments show that the models trained by BlackOut are less prone to overfitting. A primary difference between them is that DropOut is routinely used at input and/or hidden layers of deep neural networks, while BlackOut only operates at output layer. We chose the name BlackOut in light of the similarities between our method and DropOut, and the complementary they offer to train deep neural networks.

2 BLACKOUT: A SAMPLING-BASED APPROXIMATION

We will primarily focus on estimation of the matrix W_{out}. To simplify notation, in the sequel we will use θ to denote W_{out} and θ_j to denote the j-th row of W_{out}. Moreover, let $\langle \cdot, \cdot \rangle$ denote the dot product between two vectors. Given these notations, one can rewrite equation (2) as

$$p_\theta(w_i|s) = \frac{\exp(\langle \theta_i, s \rangle)}{\sum_{j=1}^{V} \exp(\langle \theta_j, s \rangle)} \quad \forall i \in \{1, \cdots, V\}. \tag{3}$$

RNNLMs with a softmax output layer are typically trained using cross-entropy as the loss function, which is equivalent to maximum likelihood (ML) estimation, that is, to find the model parameter θ which maximizes the log-likelihood of target word w_i, given a history context s:

$$J_{ml}^s(\theta) = \log p_\theta(w_i|s), \tag{4}$$

whose gradient is given by

$$\begin{aligned} \frac{\partial J_{ml}^s(\theta)}{\partial \theta} &= \frac{\partial}{\partial \theta} \langle \theta_i, s \rangle - \sum_{j=1}^{V} p_\theta(w_j|s) \frac{\partial}{\partial \theta} \langle \theta_j, s \rangle, \\ &= \frac{\partial}{\partial \theta} \langle \theta_i, s \rangle - \mathbb{E}_{p_\theta(w|s)} \left[\frac{\partial}{\partial \theta} \langle \theta_w, s \rangle \right]. \end{aligned} \tag{5}$$

The gradient of log-likelihood is expensive to evaluate because (1) the cost of computing $p_\theta(w_j|s)$ is $\mathcal{O}(Vh)$ and (2) the summation above takes time linear in the vocabulary size $\mathcal{O}(V)$.

To alleviate the computational bottleneck of computing the gradient (5), we propose to use the following *discriminative* objective function for training RNNLM:

$$J_{disc}^s(\theta) = \log \tilde{p}_\theta(w_i|s) + \sum_{j \in S_K} \log(1 - \tilde{p}_\theta(w_j|s)), \tag{6}$$

where S_K is a set of indices of K words drawn from the vocabulary, and $i \notin S_K$. Typically, K is a tiny fraction of V, and in our experiments we use $K \approx V/200$. To generate S_K we will sample K words from the vocabulary using an easy to sample distribution $Q(w)$, and set $q_j := \frac{1}{Q(w_j)}$ in order to compute

$$\tilde{p}_\theta(w_i|s) = \frac{q_i \exp(\langle \theta_i, s \rangle)}{q_i \exp(\langle \theta_i, s \rangle) + \sum_{j \in S_K} q_j \exp(\langle \theta_j, s \rangle)}. \tag{7}$$

Equation 6 is the cost function of a standard logistic regression classifier that discriminates one positive sample w_i from K negative samples $w_j, \forall j \in S_K$. The first term in (6) corresponds to the traditional maximum likelihood training, and the second term explicitly pushes down the probability of negative samples in addition to the implicit shrinkage enforced by the denominator of (7). In our experiments, we found the discriminative training (6) outperforms the maximum likelihood training (the first term of Eq. 6) in all the cases, with varying degree of accuracy improvement depending on K.

The weighted softmax function (7) can be considered as a stochastic version of the standard softmax (3) on a different base measure. While the standard softmax (3) uses a base measure which gives equal weights to all words, and has support over the entire vocabulary, the base measure used in (7) has support only on $K + 1$ words: the target word w_i and K samples from $Q(w)$. The noise portion of (7) has the motivation from the sampling scheme, and the q_i term for target word w_i is introduced mainly to balance the contributions from target word and noisy sample words.[1] Other justifications are discussed in Sec. 2.1 and Sec. 2.2, where we establish close connections between BlackOut, importance sampling, and noise contrastive estimation.

Due to the weighted sampling property of BlackOut, some words might be sampled multiple times according to the proposal distribution $Q(w)$, and thus their indices may appear multiple times in S_K. As w_i is the target word, which is assumed to be included in computing (7), we therefore set $i \notin S_K$ explicitly.

[1] It's shown empirically in our experiments that setting $q_i = 1$ in (7) hurts the accuracy significantly.

Substituting (7) into (6) and letting $u_j = \langle \theta_j, s \rangle$ and $\tilde{p}_j = \tilde{p}_\theta(w_j|s)$, we have

$$J^s_{disc}(\theta) \propto u_i - (K+1) \log \sum_{k \in \{i\} \cup S_K} q_k \exp(u_k) + \sum_{j \in S_K} \log \left(\sum_{k \in \{i\} \cup S_K} q_k \exp(u_k) - q_j \exp(u_j) \right). \tag{8}$$

Then taking derivatives with respect to $u_j, \forall j \in \{i\} \cup S_K$, yields

$$\frac{\partial J^s_{disc}(\theta)}{\partial u_i} = 1 - \left(K + 1 - \sum_{j \in S_K} \frac{1}{1 - \tilde{p}_j} \right) \tilde{p}_i \tag{9}$$

$$\frac{\partial J^s_{disc}(\theta)}{\partial u_j} = - \left(K + 1 - \sum_{k \in S_K \setminus \{j\}} \frac{1}{1 - \tilde{p}_k} \right) \tilde{p}_j, \quad \text{for } j \in S_K. \tag{10}$$

By the chain rule of derivatives, we can propagate the errors backward to previous layers and compute the gradients with respect to the full model parameters Ω. In contrast to Eq. 5, Eqs. 9 and 10 are much cheaper to evaluate as (1) the cost of computing $\tilde{p}_j$ is $\mathcal{O}(Kh)$ and (2) the summation takes $\mathcal{O}(K)$, hence roughly a V/K times of speed-up.

Next we turn our attention to the proposal distribution $Q(w)$. In the past, a uniform distribution or the unigram distribution have been advocated as promising candidates for sampling distributions (Bengio & Senécal, 2003; Jean et al., 2015; Bengio & Senécal, 2008; Mnih & Teh, 2012). As we will see in the experiments, neither one is suitable for a wide range of datasets, and we find that the power-raised unigram distribution of Mikolov et al. (2013) is very important in this context:

$$Q_\alpha(w) \propto p^\alpha_{uni}(w), \quad \alpha \in [0, 1]. \tag{11}$$

Note that $Q_\alpha(w)$ is a generalization of uniform distribution (when $\alpha = 0$) and unigram distribution (when $\alpha = 1$). The rationale behind our choice is that by tuning α, one can interpolate smoothly between sampling popular words, as advocated by the unigram distribution, and sampling all words equally. The best α is typically dataset and/or problem dependent; in our experiments, we use a holdout set to find the best value of α. It's worth noting that this sampling strategy has been used by Mikolov et al. (2013) in a similar context of word embedding, while here we explore its effect in the language modeling applications.

After BlackOut training, we evaluate the predictive performance of RNNLM by perplexity. To calculate perplexity, we explicitly normalize the output distribution by using the exact softmax function (3). This is similar to DropOut (Srivastava et al., 2014), wherein a subset of network is sampled and trained at each training batch and when evaluating, the full network participates.

2.1 Connection to Importance Sampling

BlackOut has a close connection to importance sampling (IS). To see this, differentiating the logarithm of Eq. 7 with respect to model parameter θ, we have

$$\begin{aligned} \frac{\partial}{\partial \theta} \log \tilde{p}_\theta(w_i|s) &= \frac{\partial}{\partial \theta} \langle \theta_i, s \rangle - \frac{1}{\sum_{k \in \{i\} \cup S_K} q_k \exp(\langle \theta_k, s \rangle)} \sum_{j \in \{i\} \cup S_K} q_j \exp(\langle \theta_j, s \rangle) \frac{\partial}{\partial \theta} \langle \theta_j, s \rangle \\ &= \frac{\partial}{\partial \theta} \langle \theta_i, s \rangle - \mathbb{E}_{\tilde{p}_\theta(w|s)} \left[\frac{\partial}{\partial \theta} \langle \theta_w, s \rangle \right]. \end{aligned} \tag{12}$$

In contrast with Eq. 5, it shows that the weighted softmax function (7) corresponds to an IS-based estimator of the standard softmax (3) with a proposal distribution $Q(w)$.

Importance sampling has been applied to NNLMs with large output layers in previous works (Bengio & Senécal, 2003; 2008; Jean et al., 2015). However, either uniform distribution or unigram distribution is used for sampling and all aforementioned works exploit the maximum likelihood learning of model parameter θ. By contrast, BlackOut uses a discriminative training (6) and a power-raised unigram distribution $Q_\alpha(w)$ for sampling; these two changes are important to mitigate some of limitations of IS-based approaches. While an IS-based approach with a uniform proposal distribution

is very stable for training, it suffers from large bias due to the apparent divergence of the uniform distribution from the true data distribution $p_\theta(w|s)$. On the other hand, a unigram-based IS estimate can make learning unstable due to the high variance (Bengio & Senécal, 2003; 2008). Using a power-raised unigram distribution $Q_\alpha(w)$ entails a better trade-off between bias and variance, and thus strikes a better balance between these two extremes. In addition, as we will see from the experiments, the discriminative training of BlackOut speeds up the rate of convergence over the traditional maximum likelihood learning.

2.2 Connection to Noise Contrastive Estimation

The basic idea of NCE is to transform the density estimation problem to the problem of learning by comparison, e.g., estimating the parameters of a binary classifier that distinguishes samples from the data distribution p_d from samples generated by a known noise distribution p_n (Gutmann & Hyvärinen, 2012). In the language modeling setting, the data distribution p_d will be the distribution $p_\theta(w|s)$ of interest, and the noise distribution p_n is often chosen from the ones that are easy to sample from and possibly close to the true data distribution (so that the classification problem isn't trivial). While Mnih & Teh (2012) uses a context-independent (unigram) noise distribution $p_n(w)$, BlackOut can be formulated into the NCE framework by considering a context-dependent noise distribution $p_n(w|s)$, estimated from K samples drawn from $Q(w)$, by

$$p_n(w_i|s) = \frac{1}{K} \sum_{j \in S_K} \frac{q_j}{q_i} p_\theta(w_j|s), \tag{13}$$

which is a probability distribution function under the expectation that K samples are drawn from $Q(w)$: $S_K \sim Q(w)$ since $\mathbb{E}_{S_K \sim Q(w)}(p_n(w_i|s)) = Q(w_i)$ and $\mathbb{E}_{S_K \sim Q(w)}(\sum_{i=1}^{V} p_n(w_i|s)) = 1$ (See the proof in Appendix A).

Similar to Gutmann & Hyvärinen (2012), noise samples are assumed K times more frequent than data samples so that data points are generated from a mixture of two distributions: $\frac{1}{K+1} p_\theta(w|s)$ and $\frac{K}{K+1} p_n(w|s)$. Then the conditional probability of sample w_i being generated from the data distribution is

$$p_\theta(D = 1|w_i, s) = \frac{p_\theta(w_i|s)}{p_\theta(w_i|s) + K p_n(w_i|s)}. \tag{14}$$

Inserting Eq. 13 into Eq. 14, we have

$$p_\theta(D = 1|w_i, s) = \frac{q_i \exp(\langle \theta_i, s \rangle)}{q_i \exp(\langle \theta_i, s \rangle) + \sum_{j \in S_K} q_j \exp(\langle \theta_j, s \rangle)}, \tag{15}$$

which is exactly the weighted softmax function defined in (7). Note that due to the noise distribution proposed in Eq. 13, the expensive denominator (or the partition function Z) of $p_\theta(w_j|s)$ is canceled out, while in Mnih & Teh (2012) the partition function Z is either treated as a free parameter to be learned or approximated by a constant. Mnih & Teh (2012) recommended to set $Z = 1.0$ in the NCE training. However, from our experiments, setting $Z = 1.0$ often leads to sub-optimal solutions[2] and different settings of Z sometimes incur numerical instability since the log-sum-exp trick[3] can not be used there to shift the scores of the output layer to a range that is amenable to the exponential function. BlackOut does not have this hyper-parameter to tune and the log-sum-exp trick still works for the weighted softmax function (7). Due to the discriminative training of NCE and BlackOut, they share the same objective function (6).

We shall emphasize that according to the theory of NCE, the K samples should be sampled from the noise distribution $p_n(w|s)$. But in order to calculate $p_n(w|s)$, we need the K samples drawn from $Q(w)$ beforehand. As an approximation, we use the same K samples drawn from $Q(w)$ as the K samples from $p_n(w|s)$, and only use the expression of $p_n(w|s)$ in (13) to evaluate the noise density value required by Eq. 14. This approximation is accurate since $\mathbb{E}_{S_K \sim Q(w)}(p_n(w_i|s)) = Q(w_i)$ as proved in Appendix A, and we find empirically that it performs much better (with improved stability) than using a unigram noise distribution as in Mnih & Teh (2012).

[2] Similarly, Chen et al. (2015) reported that setting $\ln(Z) = 9$ gave them the best results.

[3] https://en.wikipedia.org/wiki/LogSumExp

2.3 RELATED WORK

Many approaches have been proposed to address the difficulty of training deep neural networks with large output spaces. In general, they can be categorized into four categories:

- *Hierarchical softmax* (Morin & Bengio, 2005; Mnih & Hinton, 2008) uses a hierarchical binary tree representation of the output layer with the V words as its leaves. It allows exponentially faster computation of word probabilities and their gradients, but the predictive performance of the resulting model is heavily dependent on the tree used, which is often constructed heuristically. Moreover, by relaxing the constraint of a binary structure, Le et al. (2011) introduces a structured output layer with an arbitrary tree structure constructed from word clustering. All these methods speed up both the model training and evaluation considerably.
- *Sampling-based approximations* select at random or heuristically a small subset of the output layer and estimate gradient only from those samples. The use of importance sampling in Bengio & Senécal (2003; 2008); Jean et al. (2015), and the use of NCE (Gutmann & Hyvärinen, 2012) in Mnih & Teh (2012) all fall under this category, so does the more recent use of Locality Sensitive Hashing (LSH) techniques (Shrivastava & Li, 2014; Vijayanarasimhan et al., 2014) to select a subset of good samples. BlackOut, with close connections to importance sampling and NCE, also falls in this category. All these approaches only speed up the model training, while the model evaluation still remains computationally challenging.
- *Self normalization* (Devlin et al., 2014) extends the cross-entropy loss function by explicitly encouraging the partition function of softmax to be as close to 1.0 as possible. Initially, this approach only speeds up the model evaluation and more recently it's extended to facilitate the training as well with some theoretical guarantees (Andreas & Klein, 2014; Andreas et al., 2015).
- *Exact gradient on limited loss functions* (Vincent et al., 2015) introduces an algorithmic approach to efficiently compute the exact loss, gradient update for the output weights in $\mathcal{O}(h^2)$ per training example instead of $\mathcal{O}(Vh)$. Unfortunately, it only applies to a limited family of loss functions that includes squared error and spherical softmax, while the standard softmax isn't included.

As discussed in the introduction, BlackOut also shares some similarity to DropOut (Srivastava et al., 2014). While DropOut is often applied to input and/or hidden layers of deep neural networks to avoid feature co-adaptation and overfitting by uniform sampling, BlackOut applies to a softmax output layer, uses a weighted sampling, and employs a discriminative training loss. We chose the name *BlackOut* in light of the similarities between our method and DropOut, and the complementary they offer to train deep neural networks.

3 IMPLEMENTATION AND FURTHER SPEED-UP

We implemented BlackOut on a standard machine with a dual-socket 28-core Intel®Xeon®[4] Haswell CPU. To achieve high throughput, we train RNNLM with Back-Propagation Through Time (BPTT) (Rumelhart et al., 1988) with mini-batches (Chen et al., 2014). We use RMSProp (Hinton, 2012) for learning rate scheduling and gradient clipping (Bengio et al., 2013) to avoid the gradient explosion issue of recurrent networks. We use the latest Intel MKL library (version 11.3.0) for SGEMM calls, which has improved support for tall-skinny matrix-matrix multiplications, which consume about 80% of the run-time of RNNLMs.

It is expensive to access and update large models with billions of parameters. Fortunately, due to the 1-of-V encoding at input layer and the BlackOut sampling at output layer, the model update on W_{in} and W_{out} is sparse, i.e., only the model parameters corresponding to input/output words and the samples in S_K are updated at each training batch. However, subnet updates have to be done carefully due to the dependency within RMSProp updating procedure. We therefore propose an approximated RMSProp that enables an efficient subnet update and thus speeds up the algorithm even further. Details can be found in Appendix C.

[4] Intel and Xeon are trademarks of Intel Corporation in the U.S. and/or other countries.

4 EXPERIMENTS

In our experiments, we first compare BlackOut, NCE and exact softmax (without any approximation) using a small dataset. We then evaluate the performance of BlackOut on the recently released one billion word language modeling benchmark (Chelba et al., 2014) with a vocabulary size of up to one million. We compare the performance of BlackOut on a standard CPU machine versus the state-of-the-arts reported in the literature that are achieved on GPUs or on clusters of CPU nodes. Our implementation and scripts are open sourced at `https://github.com/IntelLabs/rnnlm`.

Corpus Models are trained and evaluated on two different corpora: a small dataset provided by the RNNLM Toolkit[5], and the recently released one billion word language modeling benchmark[6], which is perhaps the largest public dataset in language modeling. The small dataset has 10,000 training sentences, with 71,350 words in total and 3,720 unique words; and the test perplexity is evaluated on 1,000 test sentences. The one billion word benchmark was constructed from a monolingual/English corpora; after all necessary preprocessing including de-duplication, normalization and tokenization, 30,301,028 sentences (about 0.8 billion words) are randomly selected for training, 6,075 sentences are randomly selected for test and the remaining 300,613 sentences are reserved for future development and can be used as holdout set.

4.1 RESULTS ON SMALL DATASET

We evaluate BlackOut, NCE and exact softmax (without any approximation) on the small dataset described above. This small dataset is used so that we can train the standard RNNLM algorithm with exact softmax within a reasonable time frame and hence to provide a baseline of expected perplexity. There are many other techniques involved in the training, such as RMSProp for learning rate scheduling (Hinton, 2012), subnet update (Appendix C), and mini-batch splicing (Chen et al., 2014), etc., which can affect the perplexity significantly. For a fair comparison, we use the same tricks and settings for all the algorithms, and only evaluate the impact of the different approximations (or no approximation) on the softmax output layer. Moreover, there are a few hyper-parameters that have strong impact on the predictive performance, including α of the proposal distribution $Q_\alpha(w)$ for BlackOut and NCE, and additionally for NCE, the partition function Z. We pay an equal amount of effort to tune these hyper-parameters for BlackOut and NCE on the validation set as number of samples increases.

Figure 2 shows the perplexity reduction as a function of number of samples K under two different vocabulary settings: (a) a full vocabulary of 3,720 words, and (b) using the most frequent 2,065 words as vocabulary. The latter is a common approach used in practice to accelerate RNNLM computation by using RNNLM to predict only the most frequent words and handling the rest using an n-gram model (Schwenk & Gauvain, 2005). We will see similar vocabulary settings when we evaluate BlackOut on the large scale one billion word benchmark.

As can be seen, when the size of the samples increases, in general both BlackOut and NCE improve their prediction accuracy under the two vocabulary settings, and even with only 2 samples both algorithms still converge to reasonable solutions. BlackOut can utilize samples much more effectively than NCE as manifested by the significantly lower perplexities achieved by BlackOut, especially when number of samples is small; Given about 20-50 samples, BlackOut and NCE reach similar perplexities as the exact softmax, which is expensive to train as it requires to evaluate all the words in the vocabularies. When the vocabulary size is 2,065, BlackOut achieves even better perplexity than that of the exact softmax. This is possible since BlackOut does stochastic sampling at each training example and uses the full softmax output layer in prediction; this is similar to DropOut that is routinely used in input layer and/or hidden layers of deep neural networks (Srivastava et al., 2014). As in DropOut, BlackOut has the benefit of regularization and avoids feature co-adaption and is possibly less prone to overfitting. To verify this hypothesis, we evaluate the perplexities achieved on the training set for different algorithms and provide the results in Figure 5 at Appendix B. As can been seen, the exact softmax indeed overfits to the training set and reaches lower training perplexities than NCE and BlackOut.

[5] http://www.rnnlm.org/

[6] https://code.google.com/p/1-billion-word-language-modeling-benchmark/

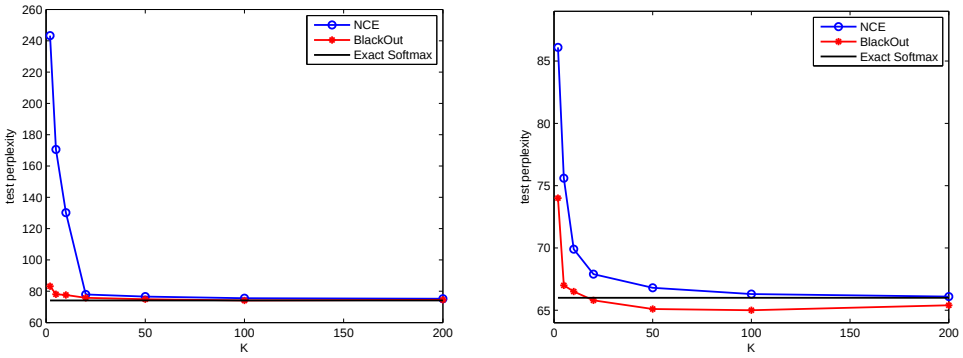

Figure 2: Test perplexity evolution as a function of number of samples K (a) with a full vocabulary of 3,720 words, and (b) with the most frequent 2,065 words in vocabulary. The experiments are executed on the RNNLMs with 16 hidden units.

Next we compare the convergence rates of BlackOut and NCE when training the RNNLMs with 16 hidden units for a full vocabulary of 3,720 words. Figures 3(a) and 3(b) plot the learning curves of BlackOut and NCE when 10 samples or 50 samples are used in training, respectively. The figure shows that BlackOut enjoys a much faster convergence rate than NCE, especially when number of samples is small (Figure 3(a)); but this advantage gets smaller when number of samples increases (Figure 3(b)). We also observed similar behavior when we evaluated BlackOut and NCE on the large scale one billion word benchmark.

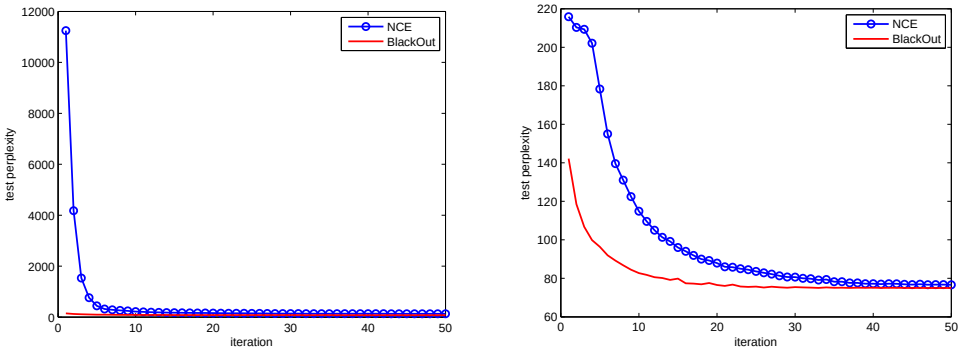

Figure 3: The learning curves of BlackOut and NCE when training the RNNLMs with 16 hidden units with (a) 10 samples, and (b) 50 samples.

4.2 Results On One Billion Word Benchmark

We follow the experiments from Williams et al. (2015) and Le et al. (2015) and compare the performance of BlackOut with the state-of-the-art results provided by them. While we evaluated BlackOut on a dual-socket 28-core Intel®Xeon®Haswell machine, Williams et al. (2015) implemented RNNLM with the NCE approximation on NVIDIA GTX Titan GPUs, and Le et al. (2015) executed an array of recurrent networks, including deep RNN and LSTM, without approximation on a CPU cluster. Besides the time-to-solution comparison, these published results enable us to cross-check the predictive performance of BlackOut with another implementation of NCE or with other competitive network architectures.

4.2.1 When vocabulary size is 64K

Following the experiments in Williams et al. (2015), we evaluate the performance of BlackOut on a vocabulary of 64K most frequent words. This is similar to the scenario in Figure 2(b) where the

most frequent words are kept in vocabulary and the rest rare words are mapped to a special <unk> token. We first study the importance of α of the proposal distribution $Q_\alpha(w)$ and the discriminative training (6) as proposed in BlackOut. As we discussed in Sec. 2, when $\alpha = 0$, the proposal distribution $Q_\alpha(w)$ degenerates to a uniform distribution over all the words in the vocabulary, and when $\alpha = 1$, we recover the unigram distribution. Thus, we evaluate the impact of α in the range of $[0, 1]$. Figure 4(a) shows the evolution of test perplexity as a function of α for the RNNLMs with 256 hidden units. As can be seen, α has a significant impact on the prediction accuracy. The commonly used uniform distribution (when $\alpha = 0$) and unigram distribution (when $\alpha = 1$) often yield sub-optimal solutions. For the dataset and experiment considered, $\alpha = 0.4$ gives the best perplexity (consistent on holdout set and test set). We therefore use $\alpha = 0.4$ in the experiments that follow. The number of samples used is 500, which is about 0.8% of the vocabulary size.

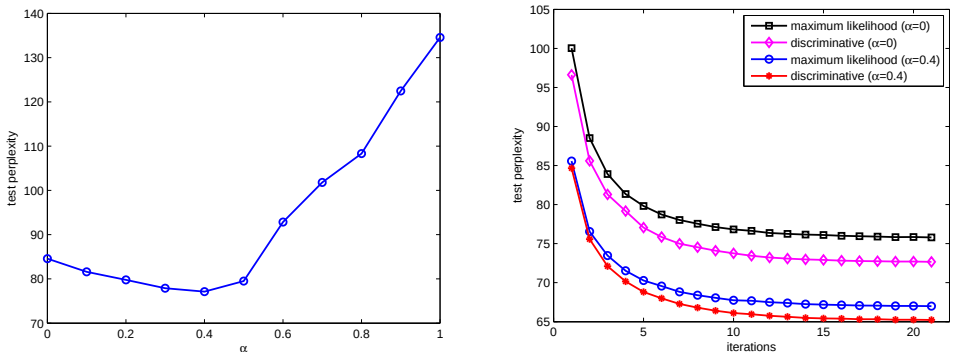

Figure 4: (a) The impact of α evaluated when 256 hidden units are used; (b) The learning curves of maximum likelihood and discriminative training when 512 hidden units are used.

Figure 4(b) demonstrates the impact of discriminative training (6) over the maximum likelihood training (the first term of Eq. 6) on the RNNLMs with 512 hidden units using two different α's. In general, we observe 1-3 points of perplexity reduction due to discriminative training over traditional maximum likelihood training.

Finally, we evaluate the scalability of BlackOut when number of hidden units increases. As the dataset is large, we observed that the performance of RNNLM depends on the size of the hidden layer: they perform better as the size of the hidden layer gets larger. As a truncated 64K word vocabulary is used, we interpolate the RNNLM scores with a full size 5-gram to fill in rare word probabilities (Schwenk & Gauvain, 2005; Park et al., 2010). We report the interpolated perplexities BlackOut achieved and compare them with the results from Williams et al. (2015) in Table 1. As can be seen, BlackOut reaches lower perplexities than those reported in Williams et al. (2015) within comparable time frames (often 10%-40% faster). We achieved a perplexity of 42.0 when the hidden layer size is 4096. To the best of our knowledge, this is the lowest perplexity reported on this benchmark.

Table 1: Performance on the one billion word benchmark by interpolating RNNLM on a 64K word vocabulary with a full-size KN 5-gram LM.

Model	#Params [millions]	Test Perplexity		Time to Solution	
		Published[1]	BlackOut	Published[1]	BlackOut
KN 5-gram	1,748	66.95		45m	
RNN-128 + KN 5-gram	1,764	60.8	59.0	6h	9h
RNN-256 + KN 5-gram	1,781	57.3	55.1	16h	14h
RNN-512 + KN 5-gram	1,814	53.2	51.5	1d2h	1d
RNN-1024 + KN 5-gram	1,880	48.9	47.6	2d2h	1d14h
RNN-2048 + KN 5-gram	2,014	45.2	43.9	4d7h	2d15h
RNN-4096 + KN 5-gram	2,289	**42.4**	**42.0**	14d5h	10d

[1] Data from Table 1 of Williams et al. (2015).

4.2.2 WHEN VOCABULARY SIZE IS 1M

In the final set of experiments, we evaluate the performance of BlackOut with a very large vocabulary of 1,000,000 words, and the results are provided in Table 2. This is the largest vocabulary used on this benchmark that we could find in existing literature. We consider the RNNLM with 1,024 hidden units (about 2 billion parameters) and 2,048 hidden units (about 4.1 billion parameters) and compare their test perplexities with the results from Le et al. (2015). We use 2,000 samples, 0.2% of the vocabulary size, for BlackOut training with $\alpha = 0.1$. Comparing to the experiments with the 64K word vocabulary, a much smaller α is used here since the sampling rate (0.2%) is much lower than that is used (0.8%) when the vocabulary size is 64K, and a smaller α strikes a better balance between sample coverage per training example and convergence rate. In contrast, NCE with the same setting converges very slowly (similar to Figure 3(a)) and couldn't reach a competitive perplexity within the time frame considered, and its results are not reported here.

As the standard RNN/LSTM algorithms (without approximation) are used in Le et al. (2015), a cluster of 32 CPU machines (at least 20 cores each) are used to train the models for about 60 hours. BlackOut enables us to train this large model using a single CPU machine for 175 hours. Since different model architectures are used in the experiments (deep RNN/LSTM vs. standard RNNLM), the direct comparison of test perplexity isn't very meaningful. However, this experiment demonstrates that even though our largest model is about 2-3 times larger than the models evaluated in Le et al. (2015), BlackOut, along with a few other optimization techniques, make this large scale learning problem still feasible on a single box machine without using GPUs or CPU clusters.

Table 2: Performance on the one billion word benchmark with a vocabulary of 1,000,000 words. Single model (RNN/LSTM-only) perplexities are reported; no interpolation is applied to any models.

	Model	Perplexity
Results from Le et al. (2015) 60 hours 32 machines	LSTM (512 units)	**68.8**
	IRNN (4 layers, 512 units)	69.4
	IRNN (1 layer, 1024 units + 512 linear units)	70.2
	RNN (4 layers, 512 tanh units)	71.8
	RNN (1 layer, 1024 tanh units + 512 linear units)	72.5
Our Results 175 hours, 1 machine	RNN (1 layer, 1024 sigmoid units)	78.4
	RNN (1 layer, 2048 sigmoid units)	**68.3**

Last, we collect all the state of the art results we are aware of on this benchmark and summarize them in Table 3. Since all the models are the interpolated ones, we interpolate our best RNN model[7] from Table 2 with the KN 5-gram model and achieve a perplexity score of 47.3. Again, different papers provide their best models trained with different architectures and vocabulary settings. Hence, an absolutely fair comparison isn't possible. Regardless of these discrepancies, our models, within different groups of vocabulary settings, are very competitive in terms of prediction accuracy and model size.

Table 3: Comparison with the state of the art results reported on the one billion word benchmark.

Model	#Params [billions]	Test Perplexity
RNN-1024 (full vocab) + MaxEnt[1]	20	51.3
RNN-2048 (full vocab) + KN 5-gram[2]	5.0	47.3
RNN-1024 (full vocab) + MaxEnt + 3 models[1]	42.9	43.8
RNN-4096 (64K vocab) + KN 5-gram[3]	2.3	42.4
RNN-4096 (64K vocab) + KN 5-gram[2]	2.3	42.0

[1]Data from Chelba et al. (2014); [2]Our results; [3]Data from Williams et al. (2015).

[7]To be consistent with the benchmark in Chelba et al. (2014), we retrained it with the full-size vocabulary of about 0.8M words.

5 CONCLUSION

We proposed *BlackOut*, a sampling-based approximation, to train RNNLMs with very large vocabularies (e.g., 1 million). We established its connections to importance sampling and noise contrastive estimation (NCE), and demonstrated its stability, sample efficiency and rate of convergence on the recently released one billion word language modeling benchmark. We achieved the lowest reported perplexity on this benchmark without using GPUs or CPU clusters.

As for future extensions, our plans include exploring other proposal distributions $Q(w)$, and theoretical properties of the generalization property and sample complexity bounds for BlackOut. We will also investigate a multi-machine distributed implementation.

ACKNOWLEDGMENTS

We would like to thank Oriol Vinyals, Andriy Mnih and the anonymous reviewers for their excellent comments and suggestions, which helped improve the quality of this paper.

REFERENCES

Andreas, Jacob and Klein, Dan. When and why are log-linear models self-normalizing? In *Proceedings of the Annual Meeting of the North American Chapter of the Association for Computational Linguistics*, 2014.

Andreas, Jacob, Rabinovich, Maxim, Klein, Dan, and Jordan, Michael I. On the accuracy of self-normalized log-linear models. In *NIPS*, 2015.

Bengio, Yoshua and Senécal, Jean-Sébastien. Quick training of probabilistic neural nets by importance sampling. In *AISTATS*, 2003.

Bengio, Yoshua and Senécal, Jean-Sébastien. Adaptive importance sampling to accelerate training of a neural probabilistic language model. In *IEEE Transactions on Neural Networks*, volume 19, pp. 713–722, 2008.

Bengio, Yoshua, Ducharme, Rjean, and Vincent, Pascal. A neural probabilistic language model. In *NIPS*, pp. 932–938, 2001.

Bengio, Yoshua, Boulanger-Lewandowski, Nicolas, and Pascanu, Razvan. Advances in optimizing recurrent networks. In *ICASSP*, pp. 8624–8628, 2013.

Chelba, Ciprian, Mikolov, Tomas, Schuster, Mike, Ge, Qi, Brants, Thorsten, Koehn, Phillipp, and Robinson, Tony. One billion word benchmark for measuring progress in statistical language modeling. In *INTERSPEECH*, pp. 2635–2639, 2014.

Chen, Xie, Wang, Yongqiang, Liu, Xunying, Gales, Mark JF, and Woodland, Philip C. Efficient gpu-based training of recurrent neural network language models using spliced sentence bunch. In *INTERSPEECH*, 2014.

Chen, Xie, Liu, Xunying, Gales, Mark JF, and Woodland, Philip C. Recurrent neural network language model training with noise contrastive estimation for speech recognition. In *ICASSP*, 2015.

Devlin, Jacob, Zbib, Rabih, Huang, Zhongqiang, Lamar, Thomas, Schwartz, Richard, and Makhoul, John. Fast and robust neural network joint models for statistical machine translation. In *ACL*, 2014.

Gutmann, Michael U. and Hyvärinen, Aapo. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *JMLR*, 13:307–361, 2012.

Hinton, Geoffrey. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

Jean, Sbastien, Cho, Kyunghyun, Memisevic, Roland, and Bengio, Yoshua. On using very large target vocabulary for neural machine translation. In *ACL*, 2015.

Le, Hai-Son, Oparin, Ilya, Allauzen, Alexandre, Gauvain, Jean-Luc, and Yvon, François. Structured output layer neural network language model. In *ICASSP*, pp. 5524–5527, 2011.

Le, Quoc V., Jaitly, Navdeep, and Hinton, Geoffrey. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arxiv:1504.00941*, 2015.

Mikolov, Tomas, Karafiát, Martin, Burget, Lukás, Cernocký, Jan, and Khudanpur, Sanjeev. Recurrent neural network based language model. In *INTERSPEECH*, pp. 1045–1048, 2010.

Mikolov, Tomas, Deoras, Anoop, Povey, Dan, Burget, Lukar, and Cernocky, Jan Honza. Strategies for training large scale neural network language models. IEEE Automatic Speech Recognition and Understanding Workshop, 2011.

Mikolov, Tomas, Sutskever, Ilya, Chen, Kai, Corrado, Greg, and Dean, Jeffrey. Distributed representations of words and phrases and their compositionality. In Burges, Chris, Bottou, Leon, Welling, Max, Ghahramani, Zoubin, and Weinberger, Kilian (eds.), *Advances in Neural Information Processing Systems 26*, 2013.

Mnih, Andriy and Hinton, Geoffrey E. A scalable hierarchical distributed language model. In *NIPS*, volume 21, pp. 1081–1088, 2008.

Mnih, Andriy and Teh, Yee Whye. A fast and simple algorithm for training neural probabilistic language models. In *Proceedings of the International Conference on Machine Learning*, 2012.

Morin, Frederic and Bengio, Yoshua. Hierarchical probabilistic neural network language model. In *Proceedings of the international workshop on artificial intelligence and statistics*, pp. 246–252. Citeseer, 2005.

Park, Junho, Liu, Xunying, Gales, Mark J. F., and Woodland, P. C. Improved neural network based language modelling and adaptation. In *Proc. ISCA Interspeech*, pp. 10411044, 2010.

Rumelhart, David E., Hinton, Geoffrey E., and Williams, Ronald J. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pp. 696–699. MIT Press, Cambridge, MA, USA, 1988.

Schwenk, Holger and Gauvain, Jean-Luc. Training neural network language models on very large corpora. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pp. 201–208, 2005.

Shrivastava, Anshumali and Li, Ping. Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). In *NIPS*, volume 27, pp. 2321–2329, 2014.

Srivastava, Nitish, Hinton, Geoffrey, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: A simple way to prevent neural networks from overfitting. *JMLR*, 15:1929–1958, 2014.

Sundermeyer, Martin, Oparin, Ilya, Gauvain, Jean-Luc, Freiberg, Ben, Schl
"uter, Ralf, and Ney, Hermann. Comparison of feedforward and recurrent neural network language models. In *ICASSP*, pp. 8430–8434, 2013.

Vijayanarasimhan, Sudheendra, Shlens, Jonathon, Monga, Rajat, and Yagnik, Jay. Deep networks with large output spaces. *arXiv preprint arxiv:1412.7479*, 2014.

Vincent, Pascal, de Brbisson, Alexandre, and Bouthillier, Xavier. Efficient exact gradient update for training deep networks with very large sparse targets. In *NIPS*, 2015.

Williams, Will, Prasad, Niranjani, Mrva, David, Ash, Tom, and Robinson, Tony. Scaling recurrent neural network language models. In *ICASSP*, 2015.

BlackOut: Speeding up Recurrent Neural Network Language Models with Very Large Vocabularies

(Supplementary Material)

A NOISE DISTRIBUTION $p_n(w_i|s)$

Theorem 1 *The noise distribution function $p_n(w_i|s)$ defined in Eq. 13 is a probability distribution function under the expectation that K samples in S_K are drawn from $Q(w)$ randomly, $S_K \sim Q(w)$, such that $\mathbb{E}_{S_K \sim Q(w)}(p_n(w_i|s)) = Q(w_i)$ and $\mathbb{E}_{S_K \sim Q(w)}(\sum_{i=1}^{V} p_n(w_i|s)) = 1$.*

Proof

$$
\begin{aligned}
\mathbb{E}_{S_K \sim Q(w)}(p_n(w_i|s)) &= \mathbb{E}_{S_K \sim Q(w)} \left(\frac{1}{K} \sum_{j \in S_K} \frac{q_j}{q_i} p_\theta(w_j|s) \right) \\
&= \frac{Q(w_i)}{K} \mathbb{E}_{S_K \sim Q(w)} \left(\sum_{j \in S_K} \frac{p_\theta(w_j|s)}{Q(w_j)} \right) \\
&= \frac{Q(w_i)}{K} \sum_{w_k, \forall k \in S_K} \left(\prod_{k \in S_K} Q(w_k) \cdot \sum_{j \in S_K} \frac{p_\theta(w_j|s)}{Q(w_j)} \right) \\
&= \frac{Q(w_i)}{K} K \\
&= Q(w_i)
\end{aligned}
$$

$$
\mathbb{E}_{S_K \sim Q(w)} \left(\sum_{i=1}^{V} p_n(w_i|s) \right) = \sum_{i=1}^{V} \mathbb{E}_{S_K \sim Q(w)} \left(p_n(w_i|s) \right) = \sum_{i=1}^{V} \left(Q(w_i) \right) = 1
$$

■

B PERPLEXITIES ON TRAINING SET

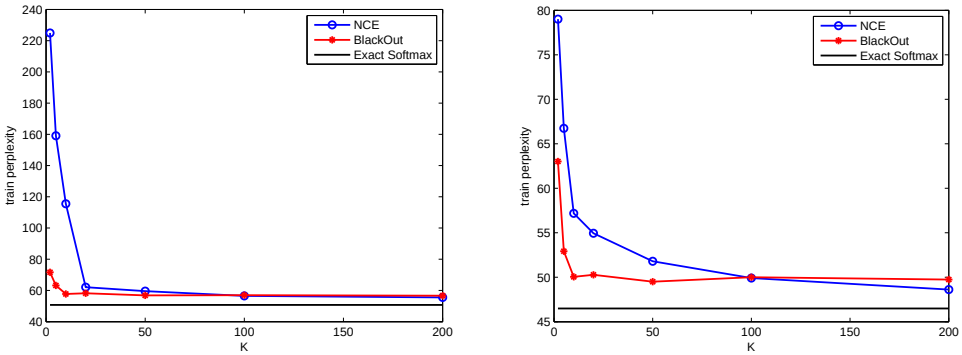

Figure 5: Training perplexity evolution as a function of number of samples K (a) with a full vocabulary of 3,720 words, and (b) with the most frequent 2,065 words in vocabulary. The experiments are executed on the RNNLMs with 16 hidden units.

C SUBNET UPDATE WITH APPROXIMATED RMSPROP

RMSProp (Hinton, 2012) is an adaptive learning rate method that has found much success in practice. Instead of using a single learning rate to all the model parameters in Ω, RMSProp dedicates a learning rate for each model parameter and normalizes the gradient by an exponential moving average of the magnitude of the gradient:

$$v_t = \beta v_{t-1} + (1-\beta)(\nabla J)^2 \tag{16}$$

where $\beta \in (0,1)$ denotes the decay rate. The model update at time step t is then given by

$$\theta_t = \theta_{t-1} + \epsilon \frac{\nabla J(\theta_{t-1})}{\sqrt{v_t + \lambda}} \tag{17}$$

where ϵ is the learning rate and λ is a damping factor, e.g., $\lambda = 10^{-6}$. While RMSProp is one of the most effective learning rate scheduling techniques, it requires a large amount of memory to store per-parameter v_t in addition to model parameter Ω and their gradients.

It is expensive to access and update large models with billions of parameters. Fortunately, due to the 1-of-V encoding at input layer and the BlackOut sampling at output layer, the model update on W_{in} and W_{out} is sparse, e.g., only the model parameters corresponding to input/output words and the samples in S_K are to be updated.[8] For Eq. 16, however, even a model parameter is not involved in the current training, its v_t value still needs to be updated by $v_t = \beta v_{t-1}$ since its $(\nabla J)^2 = 0$. Ignoring this update has detrimental effect on the predictive performance; in our experiments, we observed $5-10$ point perplexity loss if we ignore this update completely.

We resort to an approximation to $v_t = \beta v_{t-1}$. Given $p_u(w)$ is the probability of a word w being selected for update, the number of time steps elapsed when it is successfully selected follows a geometric distribution with a success rate $p_u(w)$, whose mean value is $1/p_u(w)$. Assume that an input/output word is selected according to the unigram distribution $p_{uni}(w)$ and the samples in S_K are drawn from $Q_\alpha(w)$, Eq. 16 can be approximated by

$$v_t \approx \beta^{1/p_u} v_{t-n} + (1-\beta)(\nabla J)^2 \tag{18}$$

with

$$p_u(w) = \begin{cases} p_{uni}(w) \times B \times T & \text{for word } w \text{ at input layer} \\ p_{uni}(w) \times B \times T + Q_\alpha(w) \times K \times T & \text{for word } w \text{ at output layer,} \end{cases} \tag{19}$$

where B is the mini-batch size and T is the BPTT block size. Now we can only update the model parameters, typically a tiny fraction of Ω, that are *really* involved in the current training, and thus speed up the RNNLM training further.

[8] The parameter update on W_r is still dense, but its size is several orders of magnitude smaller than those of W_{in} and W_{out}.