BlackOut: Speeding up RNNLMs with Very Large Vocabularies

Shihao Ji (Intel Labs), S.V.N. Vishwanathan (UCSC)
Nadathur Satish, Michael Anderson, Pradeep Dubey (Intel Labs)

May 2, 2016
Prevalence of Large Softmax Output Layers

Typical vocabulary size $\approx 1$M words (classes) in NLP applications

Neural Machine Translation (Kyunghyun Cho, etc.)

AlexNet (Alex Krizhevsky, etc.)
Up to 22K classes for ImageNet
Case Study: RNNLM

Objective: Given previous word sequence (history), predict next word

\[ p(w_t|w_{t-1}, \ldots, w_0) \]

\[ y_t = f(W_{out}s_t) \]

\[ s_t = \sigma(W_{in}^T x_t + W_r s_{t-1}) \]

\[ \Omega = \{ W_{in}^{V \times h}, W_r^{h \times h}, W_{out}^{V \times h} \} \]

Fig. 1 The standard RNNLM architecture.

Tall-Skinny SGEMM (e.g., V=1M, h=1K, b=128)

- Efficiency is low due to low arithmetic intensity and high BW requirements
- Matrix transpose incurs additional high overheads
- Latest Intel MKL has significant improvements to this case.
Vocabulary size = **20K** (Google’s One Billion Words LM benchmark)

<table>
<thead>
<tr>
<th></th>
<th>Throughput (words/sec)</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Published Result [1]</td>
<td>Our Result (w/o approx.)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GPU¹</td>
<td>CPU²</td>
<td>CPU³</td>
</tr>
<tr>
<td>RNN 512</td>
<td>9.9k</td>
<td>0.37k</td>
<td>12.6k</td>
</tr>
</tbody>
</table>

¹ Nvidia Geforce GTX Titan  
² Intel Xeon E5-2670 2.6GHz  
³ Intel Xeon Haswell E5-2697 v3

Strategies to Speed up Softmax

- **Hierarchical softmax** (Morin & Bengio, 2005, Mnih & Hinton, 2008)
  - Instead of a flat output layer, a hierarchical binary tree is used to encode $p_\theta(w_i|s)$

- **Sampling-based approximations** (IS, NCE, LSH, BlackOut)
  - Select at random or heuristically a small subset of the output layer

- **Self normalization** (Devlin et al., 2014)
  - Regularize the cross-entropy loss by explicitly encouraging the partition function to be as close to 1.0 as possible

- **Exact gradient on limited loss functions** (Vincent et al., 2015)
  - Algorithmic approach to efficiently compute the exact loss and gradient; only applies to squared error and spherical softmax, but not standard softmax
<table>
<thead>
<tr>
<th></th>
<th>Traditional ML</th>
<th>Blackout Training</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Softmax</strong></td>
<td>$p_\theta(w_i</td>
<td>s) = \frac{\exp((\theta_i, s))}{\sum_{j=1}^{V} \exp((\theta_j, s))} = p_i \quad \forall i \in {1, \ldots, V}$</td>
</tr>
<tr>
<td><strong>Objective function</strong></td>
<td>$J_{ml}^s(\theta) = \log p_\theta(w_i</td>
<td>s)$</td>
</tr>
<tr>
<td><strong>Gradient</strong></td>
<td>$\frac{\partial J_{ml}^s(\theta)}{\partial u_i} = 1 - p_i$</td>
<td>$\frac{\partial J_{disc}^s(\theta)}{\partial u_i} = 1 - \left( K + 1 - \sum_{j \in S_K} \frac{1}{1 - \tilde{p}_j} \right) \tilde{p}_i$ for $j \in S_K$</td>
</tr>
<tr>
<td></td>
<td>$\frac{\partial J_{ml}^s(\theta)}{\partial u_j} = -p_j$</td>
<td>$\frac{\partial J_{disc}^s(\theta)}{\partial u_j} = - \left( K + 1 - \sum_{k \in S_K \setminus {j}} \frac{1}{1 - \tilde{p}_k} \right) \tilde{p}_j$, for $j \in S_K$</td>
</tr>
</tbody>
</table>
Connection to Importance Sampling

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

Differences to previous IS-based approximations (e.g., Bengio & Senecal, 2003; 2008; Jean et al., 2015)

- Proposal density function \( Q_\alpha(w) \propto p^\alpha_{uni}(w), \quad \alpha \in [0, 1] \)
  - Uniform distribution (\( \alpha = 0 \), large bias)
  - Unigram distribution (\( \alpha = 1 \), high variance)

- ML training vs. Discriminative training
Connection to Noise Contrastive Estimate (NCE)

Convert density estimation to learning by comparison, e.g., estimating the parameters of a binary classifier that distinguish samples from the data distribution $p_\theta(w|s)$ from samples generated by a noise distribution $p_n(w|s)$ (Gutmann & Hyvarinen, 2012).

Typically, unigram is used for $p_n(w|s)$. Here we propose to use

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

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

Assume noise samples are $K$ times more frequent than data samples, the posterior of $w_i$ being generated from data dist. is

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

$$= \frac{q_i \exp(\langle \theta_i, s \rangle)}{q_i \exp(\langle \theta_i, s \rangle) + \sum_{j \in S_K} q_j \exp(\langle \theta_j, s \rangle)}$$

Advantages:
- the expensive partition function of $p_\theta(w|s)$ is cancelled out
- log-sum-exp trick can still be used for numerical stability
Comparison to Dropout

<table>
<thead>
<tr>
<th></th>
<th>Dropout</th>
<th>Blackout</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>input, hidden layers</td>
<td>(softmax) output layer</td>
</tr>
<tr>
<td>Training</td>
<td>retain a node with a fixed probability ( p )</td>
<td>sample ( K ) nodes from ( Q(w) ), each selected node is weighted by ( 1/Q(w) ) for a weighted softmax with discriminative training</td>
</tr>
<tr>
<td>Test</td>
<td>full network participates with scaled-down weights ( pW )</td>
<td>full network participates with the trained weights ( W )</td>
</tr>
<tr>
<td>Benefits</td>
<td>- speed up training, but not test</td>
<td>- speed up training, but not test</td>
</tr>
<tr>
<td></td>
<td>- model averaging, avoid overfitting (with some theoretical justifications)</td>
<td>- avoid overfitting empirically (need more theoretical justifications)</td>
</tr>
</tbody>
</table>
Experiments on Small Dataset

Fig. 2 Sample efficiency and regularization effect
Experiments on Small Dataset (cont’d)

Fig. 3 Rate of Convergence of NCE and BlackOut

(a) K=10

(b) K=50
Experiments on 1-Billion Word Benchmark

Fig. 4 Impact of $\alpha$ $Q_\alpha(w) \propto p_{\text{unl}}(w)$, $\alpha \in [0, 1]$  

Fig. 5 Traditional ML training vs. Discriminative training
Comparison to State-of-The-Arts

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.

<table>
<thead>
<tr>
<th>Model</th>
<th>#Params [millions]</th>
<th>Test Perplexity</th>
<th>Time to Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Published¹</td>
<td>BlackOut</td>
</tr>
<tr>
<td>KN 5-gram</td>
<td>1,748</td>
<td>66.95</td>
<td>45m</td>
</tr>
<tr>
<td>RNN-128 + KN 5-gram</td>
<td>1,764</td>
<td>60.8</td>
<td>59.0</td>
</tr>
<tr>
<td>RNN-256 + KN 5-gram</td>
<td>1,781</td>
<td>57.3</td>
<td>55.1</td>
</tr>
<tr>
<td>RNN-512 + KN 5-gram</td>
<td>1,814</td>
<td>53.2</td>
<td>51.5</td>
</tr>
<tr>
<td>RNN-1024 + KN 5-gram</td>
<td>1,880</td>
<td>48.9</td>
<td>47.6</td>
</tr>
<tr>
<td>RNN-2048 + KN 5-gram</td>
<td>2,014</td>
<td>45.2</td>
<td>43.9</td>
</tr>
<tr>
<td>RNN-4096 + KN 5-gram</td>
<td>2,289</td>
<td><strong>42.4</strong></td>
<td><strong>42.0</strong></td>
</tr>
</tbody>
</table>

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

GPU: Nvidia GTX Titan, CPU: 28-core Intel Xeon Haswell

Table 2: Performance on the one billion word benchmark with a vocabulary of 1,000,000 words.

<table>
<thead>
<tr>
<th>Model</th>
<th>Perplexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Le et al. (2015) 32 machines 60 hours</td>
<td><strong>68.8</strong></td>
</tr>
<tr>
<td>Our Results 1 machine 175 hours</td>
<td><strong>68.3</strong></td>
</tr>
</tbody>
</table>
Conclusion

- BlackOut is a sampling-based approximation to speed up large softmax output layers of any networks;
- We established its connection to importance sampling, NCE and the analogy to Dropout; Blackout is complementary to Dropout to train DNNs;
- The application of BlackOut to RNNLM demonstrated its stability, sample efficiency and rate of convergence on the 1-billion word benchmark;
- An optimized CPU code matched or outperformed the best known performance on GPUs and CPU clusters.

source code: https://github.com/IntelLabs/rnnlm