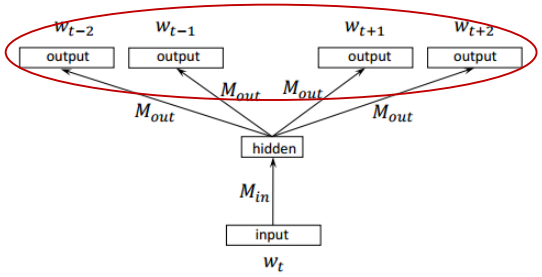
BlackOut: Speeding up RNNLMs with Very Large Vocabularies

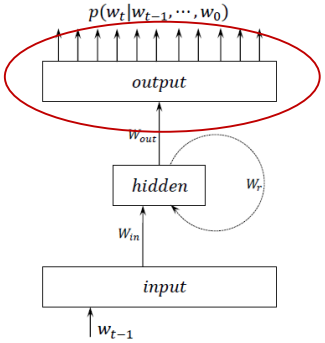
Shihao Ji (Intel Labs), S.V.N. Vishwanathan (UCSC)
Nadathur Satish, Michael Anderson, Pradeep Dubey (Intel Labs)

May 2, 2016

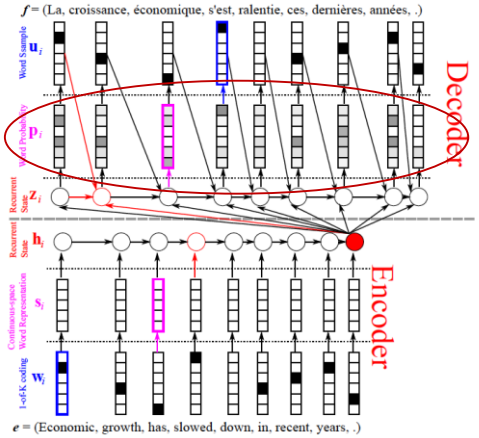
Prevalence of Large Softmax Output Layers



Word2Vec

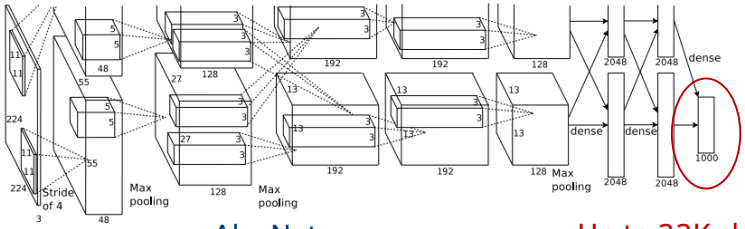


RNNLM



Neural Machine Translation (Kyunghyun Cho, etc.)

Typical vocabulary size ≈ 1M words (classes) in NLP applications



AlexNet (Alex Krizhevsky, etc)

Up to 22K classes for ImageNet

Case Study: RNNLM

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

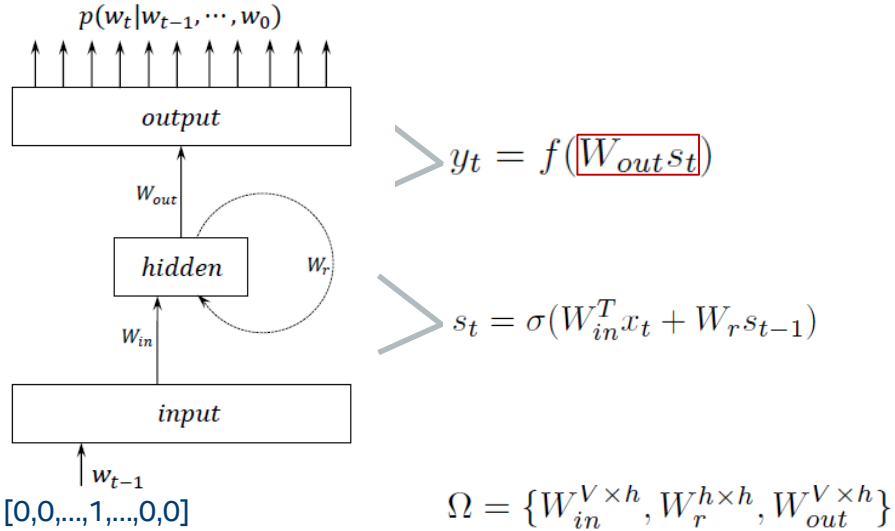
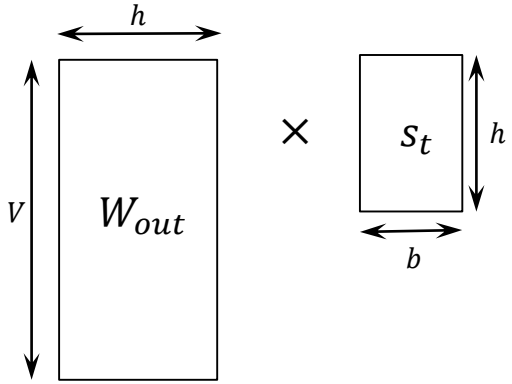


Fig. 1 The standard RNNLM architecture.

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.

System Optimization

Vocabulary size= **20K** (Google's One Billion Words LM benchmark)

	Throughput (words/sec)		
	Published Result [1]		Our Result (w/o approx.)
	GPU[1]	CPU[2]	CPU[3]
RNN 512	9.9k	0.37k	12.6k

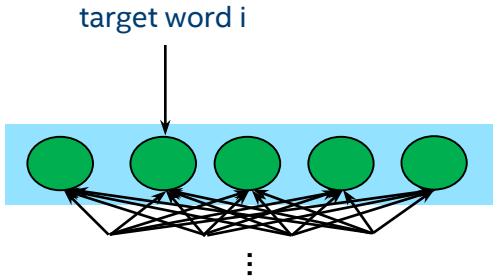
[1] Nvidia Geforce GTX Titan
[2] Intel Xeon E5-2670 2.6GHz
[3] Intel Xeon Haswell E5-2697 v3

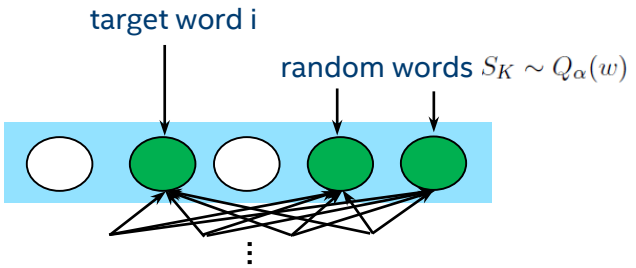
[1] X. Chen, Y. Wang, X. Liu, M. Gales, and P. Woodland. *Efficient gpu-based training of recurrent neural network language models using spliced sentence bunch*. In INTERSPEECH, 2014.

Strategies to Speed up Softmax

$$f(W_{out}s_t) \iff p_\theta\left(w_i|s\right) = \frac{\exp\left(\langle\theta_i, s\rangle\right)}{\sum_{j=1}^{V}\exp\left(\langle\theta_j, s\rangle\right)}$$

- 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 efficient compute the exact loss and gradient; only applies to squared error and spherical softmax, but not standard softmax





	Traditional ML	Blackout Training
Softmax	$p_\theta(w_i\|s) = \frac{\exp(\langle\theta_i, s\rangle)}{\sum_{j=1}^{V}\exp(\langle\theta_j, s\rangle)} \doteq p_i \quad \forall i \in \{1,\cdots,V\}$	$\tilde{p}_\theta(w_i\|s) = \frac{q_i \exp(\langle\theta_i, s\rangle)}{\sum_{j\in\{t\}\cup S_K} q_j \exp(\langle\theta_j, s\rangle)} \doteq \tilde{p}_i \quad \forall i \in \{t\}\cup S_K$ where $Q_\alpha(w) \propto p_{uni}^{\alpha}(w), \quad \alpha \in [0,1] \qquad q_j := \frac{1}{Q(w_j)}$
Objective function	$J_{ml}^{s}(\theta) = \log p_\theta(w_i\|s)$	$J_{disc}^{s}(\theta) = \log \tilde{p}_\theta(w_i\|s) + \sum_{j\in S_K} \log(1-\tilde{p}_\theta(w_j\|s))$
Gradient	$\frac{\partial J_{ml}^{s}(\theta)}{\partial u_i} = 1 - p_i$ $\frac{\partial J_{ml}^{s}(\theta)}{\partial u_j} = -p_j$	$\frac{\partial J_{disc}^{s}(\theta)}{\partial u_i} = 1 - \left(K+1-\sum_{j\in S_K}\frac{1}{1-\tilde{p}_j}\right)\tilde{p}_i$ $\frac{\partial J_{disc}^{s}(\theta)}{\partial u_j} = -\left(K+1-\sum_{k\in S_K\setminus\{j\}}\frac{1}{1-\tilde{p}_k}\right)\tilde{p}_j, \quad \text{for } j \in S_K$

Connection to Importance Sampling

$$\frac{\partial}{\partial\theta}\log\tilde{p}_\theta(w_i|s) = \frac{\partial}{\partial\theta}\langle\theta_i, s\rangle - \frac{1}{\sum_{k\in\{i\}\cup S_K} q_k \exp(\langle\theta_k, s\rangle)} \sum_{j\in\{i\}\cup S_K} q_j \exp(\langle\theta_j, s\rangle)\frac{\partial}{\partial\theta}\langle\theta_j, s\rangle$$

$$= \frac{\partial}{\partial\theta}\langle\theta_i, s\rangle - \mathbb{E}_{\tilde{p}_\theta(w|s)}\left[\frac{\partial}{\partial\theta}\langle\theta_w, s\rangle\right].$$

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

- Proposal density function $Q_\alpha(w) \propto p_{uni}^\alpha(w), \quad \alpha \in [0, 1]$
  - 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

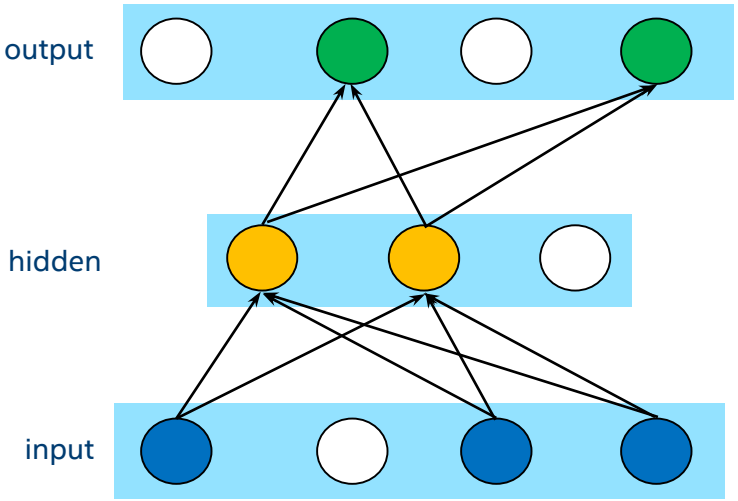
$$p_\theta(D = 1|w_i, s) = \frac{p_\theta(w_i|s)}{p_\theta(w_i|s) + K p_n(w_i|s)}$$

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

Advantages:

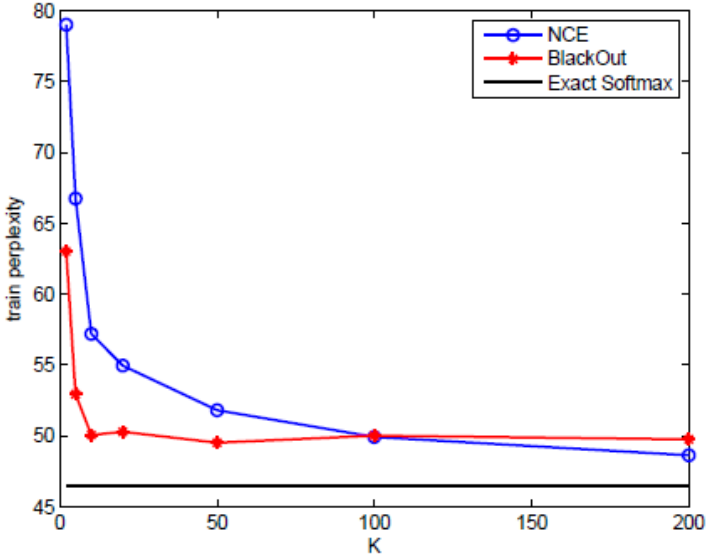
- the expensive partition function of $p_\theta(w|s)$ is cancelled out
- log-sum-exp trick can still be used for numerical stability

Comparison to Dropout

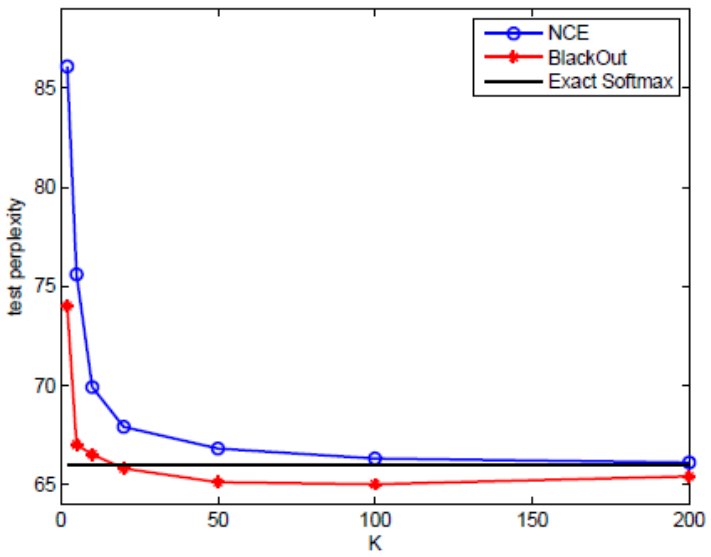


	Dropout	Blackout
Application	input, hidden layers	(softmax) output layer
Training	retain a node with a fixed probability p	sample K nodes from Q(w), each selected node is weighted by 1/Q(w) for a weighted softmax with discriminative training
Test	full network participates with scaled-down weights pW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

Experiments on Small Dataset



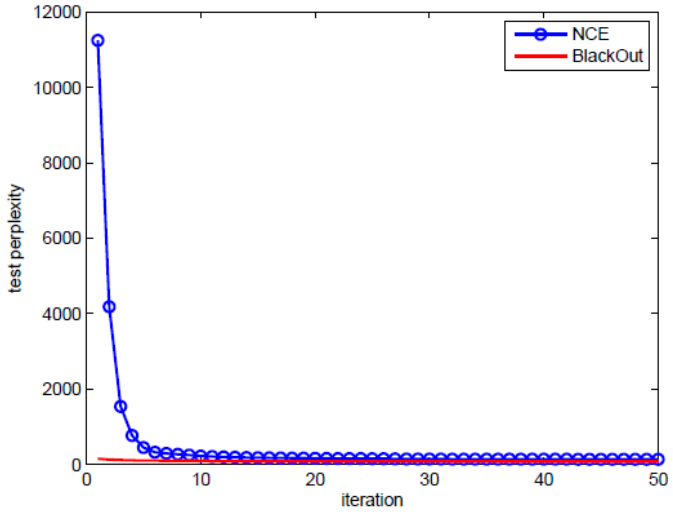
(a) On training data



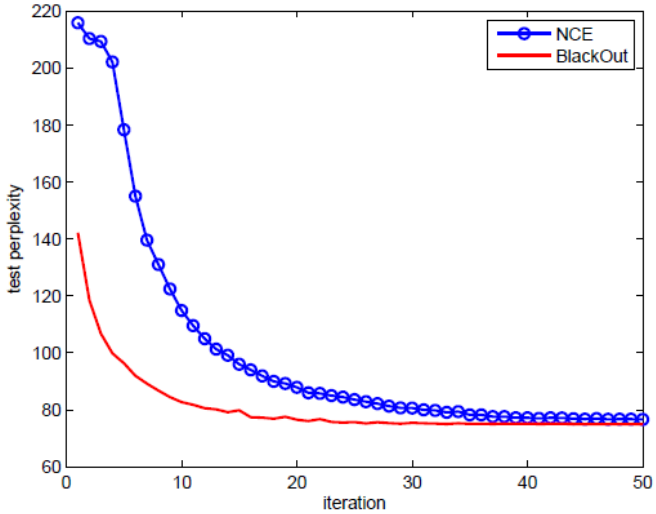
(b) On test data

Fig. 2 Sample efficiency and regularization effect

Experiments on Small Dataset (cont'd)



(a) K=10



(b) K=50

Fig. 3 Rate of Convergence of NCE and BlackOut

Experiments on 1-Billion Word Benchmark

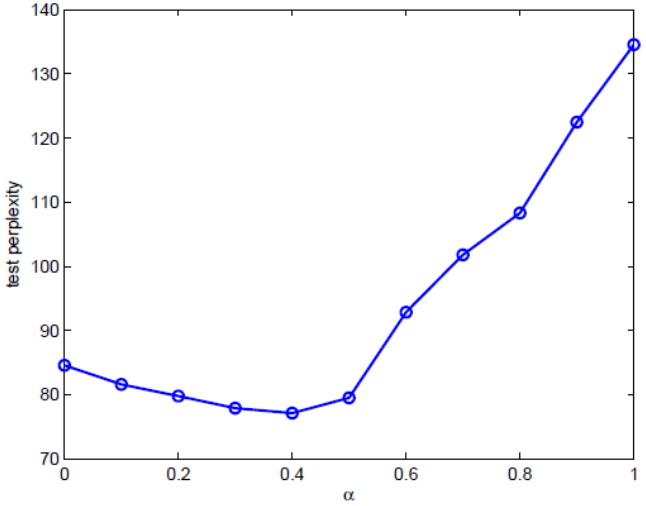


Fig. 4 Impact of α $Q_\alpha(w) \propto p_{uni}^{\alpha}(w), \quad \alpha \in [0,1]$

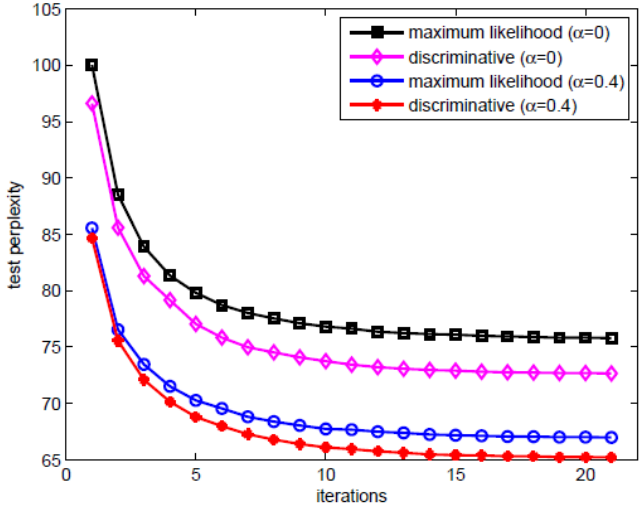


Fig. 5 Traditional ML training vs. Discriminative training

Comparison to State-of-The-Arts

Compared to GPU performance

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).

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

Compared to CPU cluster performance

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

	Model	Perplexity
Le et al. (2015) 32 machines 60 hours	LSTM (512 units)	**68.8**
Our Results 1 machine 175 hours	RNN (1 layer, 2048 sigmoid units)	**68.3**

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


intel®
experience
what's inside™