Fast Softmax Sampling for Deep Neural Networks

CS 231N Final Project

Ifueko Igbinedion and Derek Phillips

{ifueko, djp42}@stanford.edu

Problem:

In neural network classification, the softmax function converts the outputs of the last layer of a neural network to probabilities for each class, and is used to compute the ubiquitous **Cross Entropy Loss**.

The function is shown here for (x, y^*); y^* is the index of the correct class, w is the weight matrix

$$-\phi(x)^T w_{y^*} + \log \sum_j \exp(\phi(x)^T w_j)$$

It has *O(n)* complexity, which is a computational bottleneck for problems with large output spaces.

Theoretical work done here at Stanford has shown that it is possible to reduce the complexity to *O(√n)* while maintaining comparable accuracy [1].

$$-\phi(x)^T w_{y^*} + \log \left[\sum_{j \in S} \exp(\phi(x)^T w_j) + \frac{n - |S|}{|T|} \sum_{j \in S} \exp(\phi(x)^T w_j) \right]$$

S - Indices of Top K classes (nearest neighbors to inputs to last linear layer)

T - Indices of random sampled classes

$\hat{Z}$ - The quantity in [brackets], the approximation.

n - The number of classes

Approach:

We use the Facebook AI Research similarity search (FAISS) implementation for the nearest neighbors search, as it is the fastest known algorithm [2].

We examine the performance with varying:

- Output space size
- Update frequency of FAISS data structure
- Number of clusters
  - And other hyperparameters in FAISS

We first implement a Convolutional Neural Network with a subset of the Flickr 100M dataset.

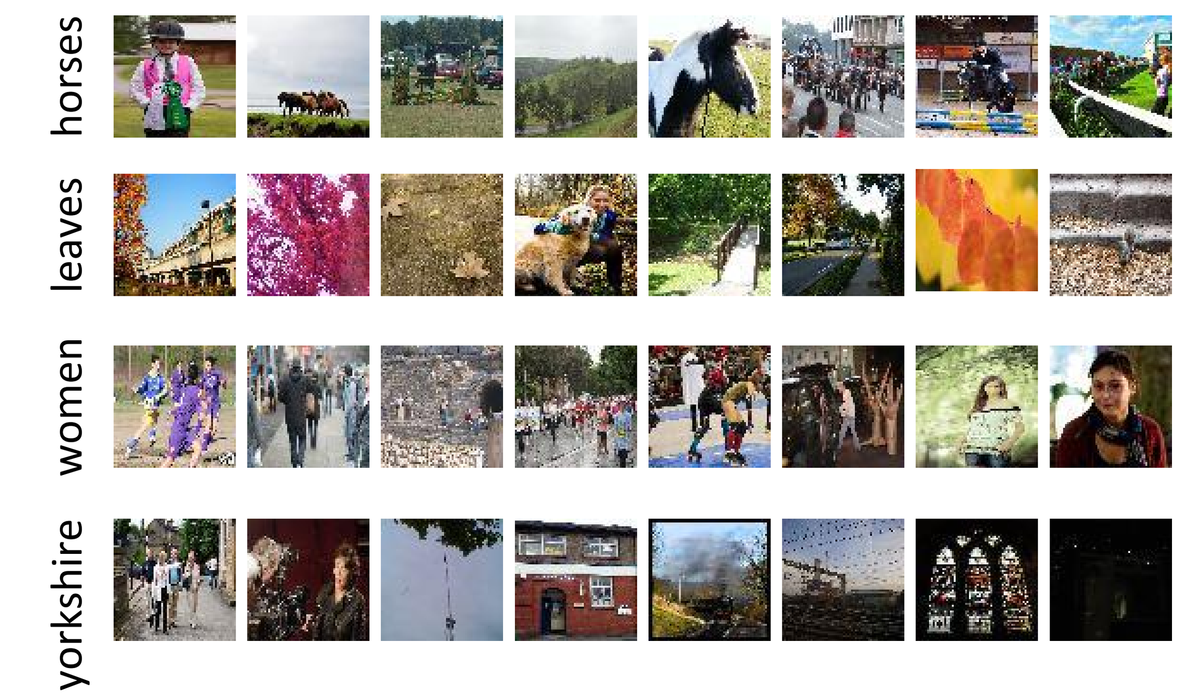
A major challenge is the computation of the gradients:

$$\nabla_{w_i} L = -\phi(x)\mathbf{1}_{i=y^*} + \mathbf{1}_{i \in S \cup T} \frac{\exp(\phi(x)^T w_i)}{\hat{Z}}$$

$$\nabla_{\phi(x)} L(x, y^*) = -w_{y^*} + \frac{\sum_{i \in S} e^{y_i} + \frac{n-|S|}{|T|} \sum_{i \in T} e^{y_i}}{\sum_{i \in S} e^{y_i} w_i + \frac{n-|S|}{|T|} \sum_{i \in T} e^{y_i} w_i}$$

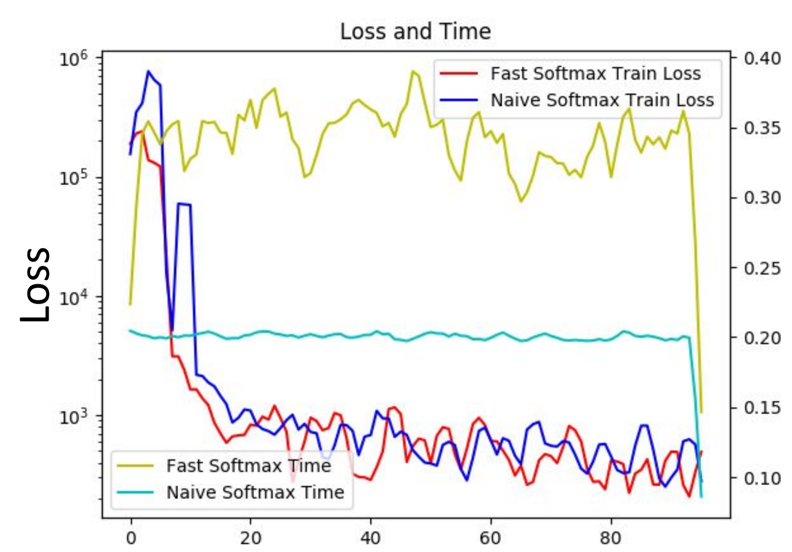
Dataset:

We use a sampled version of the Flickr 100M dataset.

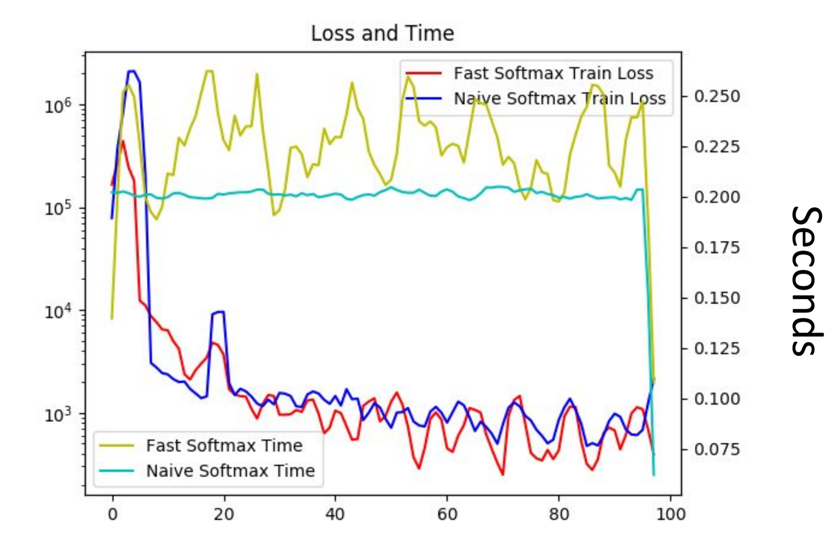


Results:

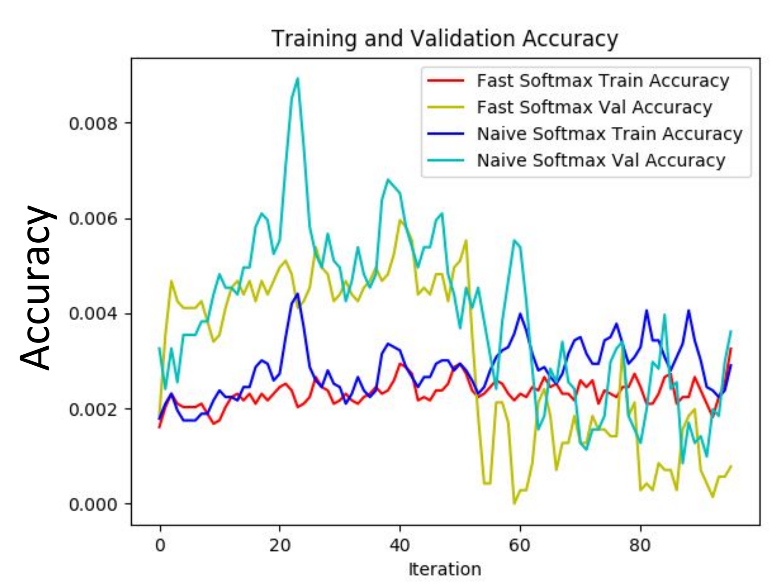
500 output classes

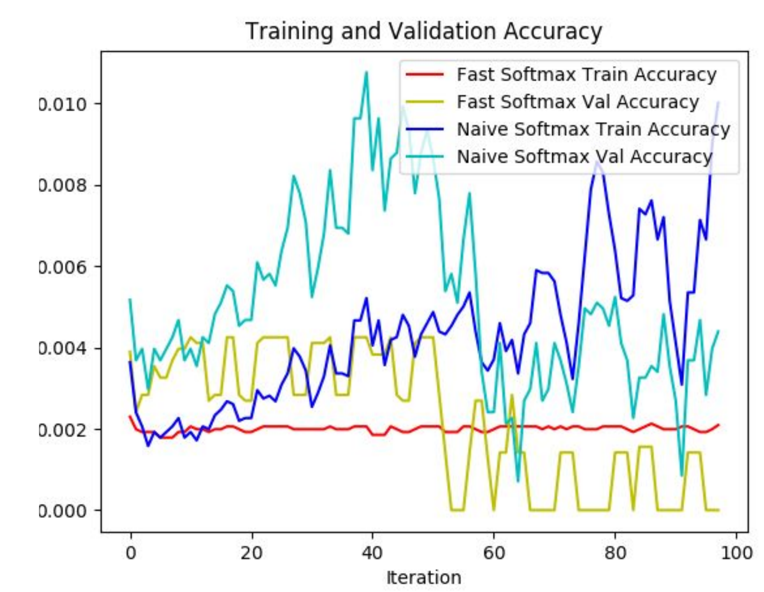


Update every 10 iterations, using 223 clusters, 277 random samples



Update every 10 iterations, using 23 clusters, 115 random samples





Conclusions:

- Some implementational challenges limit the effectiveness of the application of the theory:
  - Conversions between CPU and GPU based structures in the network frameworks
  - Lack of support for advanced indexing of PyTorch data structures
- Tradeoff is only beneficial for very large output spaces
  - RNNs with large vocabularies
- Training deep neural networks is very expensive
  - For just 500 output classes, it takes about 30 minutes to run 1 epoch with our setup

Next Steps:

- More tests with varied hyperparameters
  - Update Frequency
  - Number of Nearest Neighbors vs amount of Random Sampling (theoretical is 10√n vs 100√n)
- Improve overall performance
  - More epochs
  - Better hyperparameters for learning (i.e. learning rate)
- Larger output spaces
  - RNN for image captioning

References:

[1] S. Mussmann, D. Levy, and S. Ermon. Fast amortized in- ference and learning in log-linear models with randomly per- turbed nearest neighbor search. Unpublished.

[2] J. Johnson, M. Douze, H. Jégou. Billion Scale Similarity Search with GPUs. arXiv:1702.08734

[3] E. Grave, A. Joulin, M. Cisse´, D. Grangier, and H. Je´gou. Efficient softmax approximation for gpus. *CoRR*, abs/1609.04309, 2016.

[4] A. Joulin, L. van der Maaten, A. Jabri, and N. Vasilache. Learning visual features from large weakly supervised data. *CoRR*, abs/1511.02251, 2015.

[5] B. Thomee, D. A. Shamma, G. Friedland, B. Elizalde, K. Ni, D. Poland, D. Borth, and L.-J. Li. Yfcc100m: The new data in multimedia research. *Communications of the ACM*, 59(2), 2016.