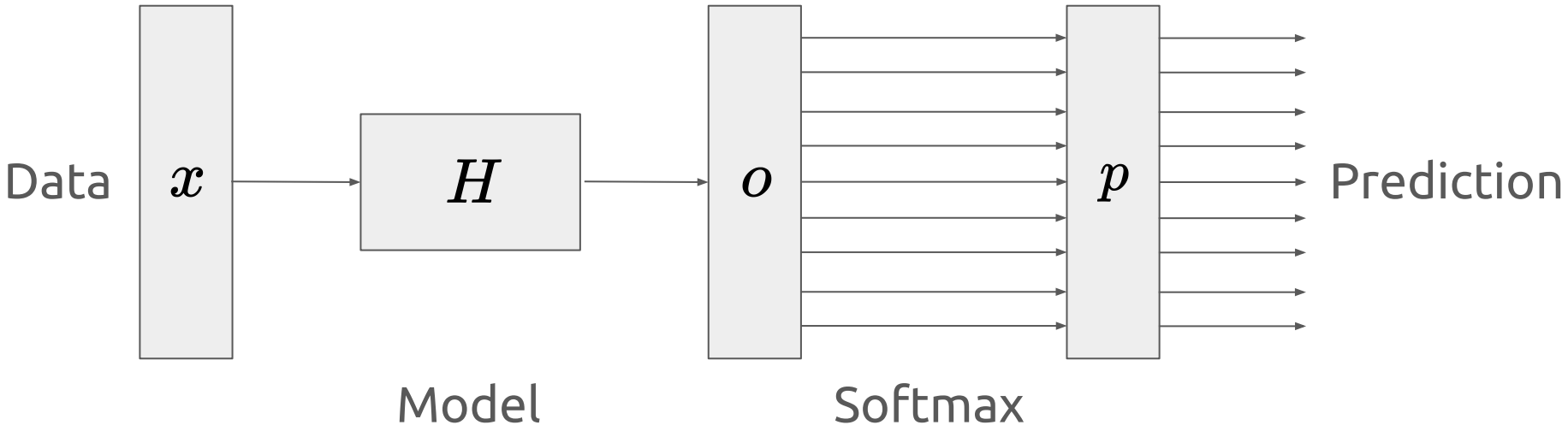
Adaptive Sampled Softmax with Kernel Based Sampling

Guy Blanc, Steffen Rendle
Google, Mountain View, USA

Proceedings of the 35th International Conference on Machine Learning
Stockholm, Sweden, PMLR 80, 2018

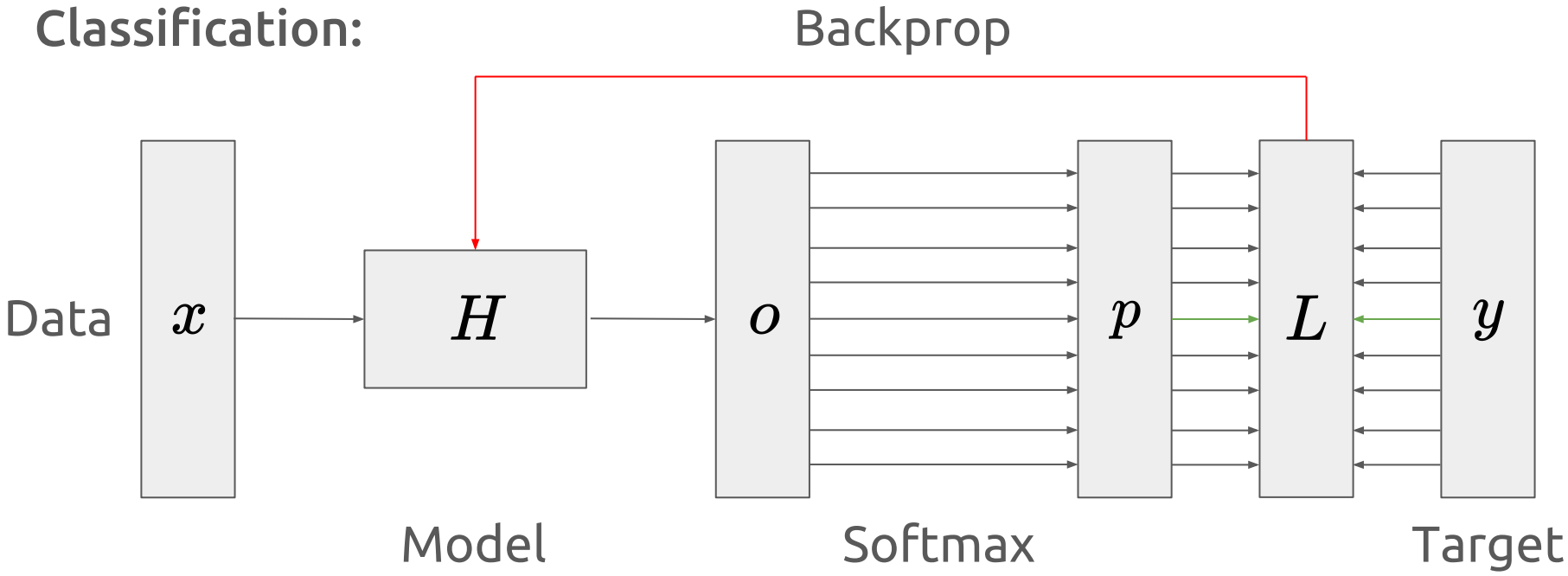
Full Softmax

Classification:



Full Softmax

Classification:

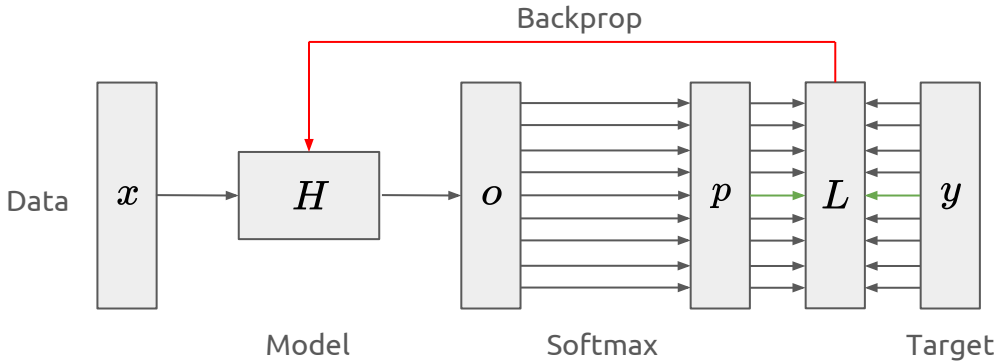


Full Softmax

Softmax: $p_i := \frac{\exp(o_i)}{\sum_{j=1}^{n} \exp(o_j)}$

Loss: $L(\boldsymbol{y}, \boldsymbol{p}) := -\sum_{i=1}^{n} y_i \log p_i = \log \sum_{i=1}^{n} \exp(o_i) - \sum_{i=1}^{n} y_i \, o_i$

Backprop: $\frac{\partial L(\boldsymbol{p}, \boldsymbol{y})}{\partial o_i} = p_i - y_i$



Full Softmax

Problem: loss function depends on all classes

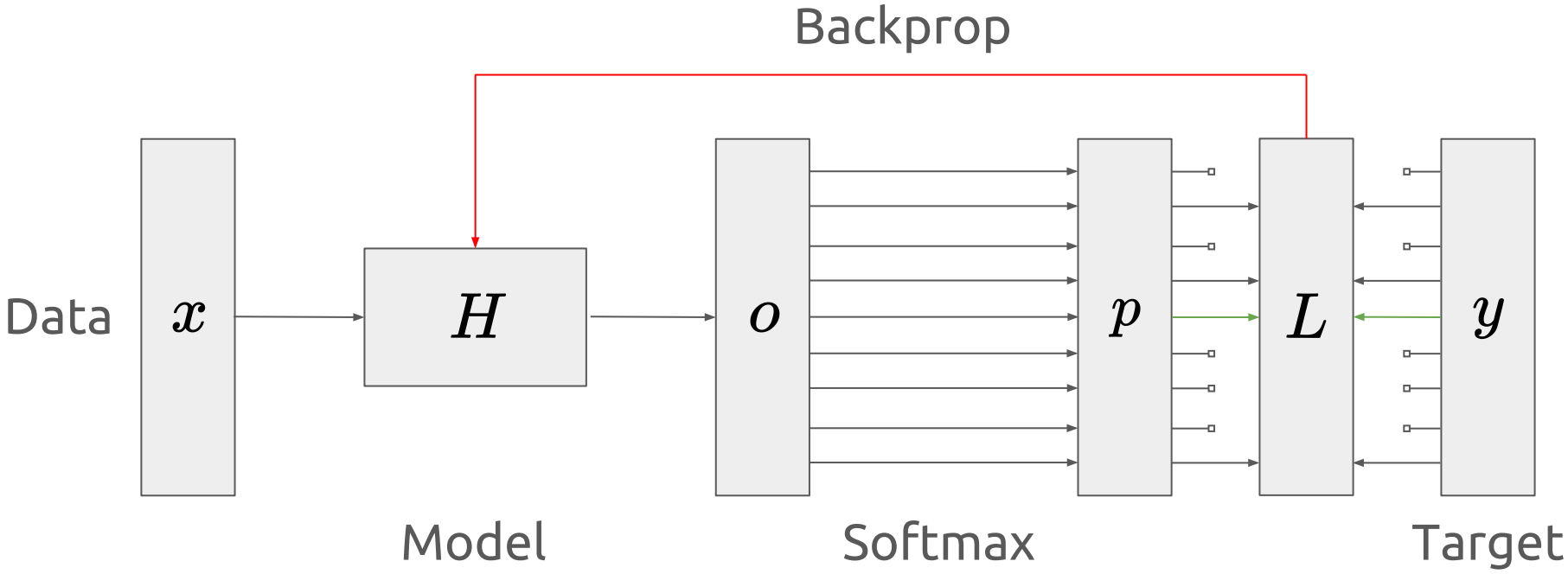
Many classes ⇒ very slow evaluation ⇒ very slow learning

$$L(\boldsymbol{y}, \boldsymbol{p}) := -\sum_{i=1}^{n} y_i \log p_i = \log \sum_{i=1}^{n} \exp(o_i) - \sum_{i=1}^{n} y_i \, o_i$$

Examples:

- Language modelling (very big vocabularies)
- Recommender systems (a lot of items to recommend from)

Sampled Softmax



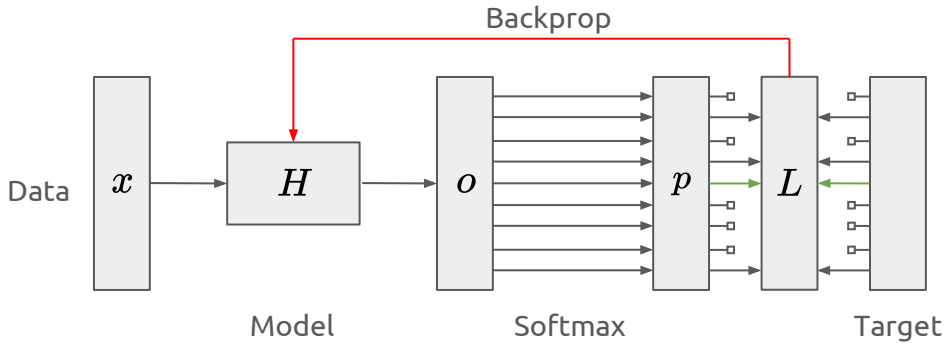
Sampled Softmax

Sample: $\boldsymbol{s} \in \{1, \ldots, n\}^{m+1}$

$$o'_i := \begin{cases} o_{s_i} - \ln(m\, q_{s_i}) & \text{if } y_{s_i} = 0 \\ o_{s_i} - \ln(1) = o_{s_i} & \text{else} \end{cases}$$

Softmax: $$p'_i := \frac{\exp(o'_i)}{\sum_{j=1}^{m+1} \exp(o'_j)}, \quad y'_i := y_{s_i}$$

Backprop: $$\frac{\partial L(\boldsymbol{p}', \boldsymbol{y}')}{\partial o_i} = \sum_{j=1}^{m+1} I(s_j = i)(p'_j - y'_j) = \sum_{j=1}^{m+1} I(s_j = i) p'_j - y_i$$



Sampling distribution

How to sample the negative classes?
The choice of sampling distribution affects the bias and how big should the sample size m be to mitigate this bias.

Ideally, we want the sampled softmax to converge to the same value as softmax…
At least, with infinitely small step size and infinitely many steps.
This is guaranteed if the estimator is unbiased:

$$E\left[\frac{\partial L(\boldsymbol{p}', \boldsymbol{y}')}{\partial o_i}\right] \stackrel{?}{=} \frac{\partial L(\boldsymbol{p}, \boldsymbol{y})}{\partial o_i} \Leftrightarrow E\left[\sum_{j=1}^{m+1} I(s_j = i) p'_j\right] \stackrel{?}{=} p_i$$

Sampling distribution

This is guaranteed if the estimator is unbiased:

$$E\left[\frac{\partial L(\boldsymbol{p}', \boldsymbol{y}')}{\partial o_i}\right] \stackrel{?}{=} \frac{\partial L(\boldsymbol{p}, \boldsymbol{y})}{\partial o_i} \Leftrightarrow E\left[\sum_{j=1}^{m+1} I(s_j = i) p'_j\right] \stackrel{?}{=} p_i$$

It was shown that sampling proportional to the softmax distribution is an unbiased estimator: $q_i = p_i \propto \exp(o_i)$

In fact, it is the only one:

Theorem 2.1. *The gradient of sample softmax is an unbiased estimator of the full softmax gradient iff* $q_i = p_i \propto \exp(o_i)$.

Sampling distribution

Three properties of a good sampling distribution:

1. ***Example dependent:*** every input x has its own sampling distribution
2. ***Model structure dependent:*** the sampling depends on the functional structure of the model.
3. ***Model parameter dependent:*** the sampling distribution changes while the model is learned.

Kernel Based Sampling

Now we focus on the models where the logits $\boldsymbol{o}$ are defined as dot-products between an input embedding $\boldsymbol{h} \in \mathbb{R}^d$ and class embeddings $\boldsymbol{w}_i \in \mathbb{R}^d$:

$$\boldsymbol{o} = \boldsymbol{W}^T \boldsymbol{h}$$

A lot of models satisfy this property.

$\mathcal{O}(nd)$ to compute the full softmax.

Kernel Based Sampling

We consider sampling distributions proportional to some kernel:

$$K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^+$$

That is, we assume there is a mapping $\boldsymbol{\phi} : \mathbb{R}^d \to \mathbb{R}^D$ such that

$$K(\boldsymbol{a}, \boldsymbol{b}) = \langle \boldsymbol{\phi}(\boldsymbol{a}), \boldsymbol{\phi}(\boldsymbol{b}) \rangle$$

Thus, the sampling distribution is written as:

$$q_i = \frac{K(\boldsymbol{h}, \boldsymbol{w}_i)}{\sum_{j=1}^{n} K(\boldsymbol{h}, \boldsymbol{w}_j)} = \frac{K(\boldsymbol{h}, \boldsymbol{w}_i)}{\left\langle \boldsymbol{\phi}(\boldsymbol{h}), \underbrace{\sum_{j=1}^{n} \boldsymbol{\phi}(\boldsymbol{w}_j)}_{=:\boldsymbol{z} \in \mathbb{R}^D} \right\rangle} \quad \longleftarrow \textit{key property}$$

Divide and Conquer

The key property allows us to compute the probability of one class efficiently.
But how do we sample the classes?..

We can first sample a subset of classes!

$$C \subseteq \{1, \dots, n\}$$

$$C' \cup C'' = C$$

$$\boldsymbol{z}(C) := \sum_{j \in C} \boldsymbol{\phi}(\boldsymbol{w}_j)$$

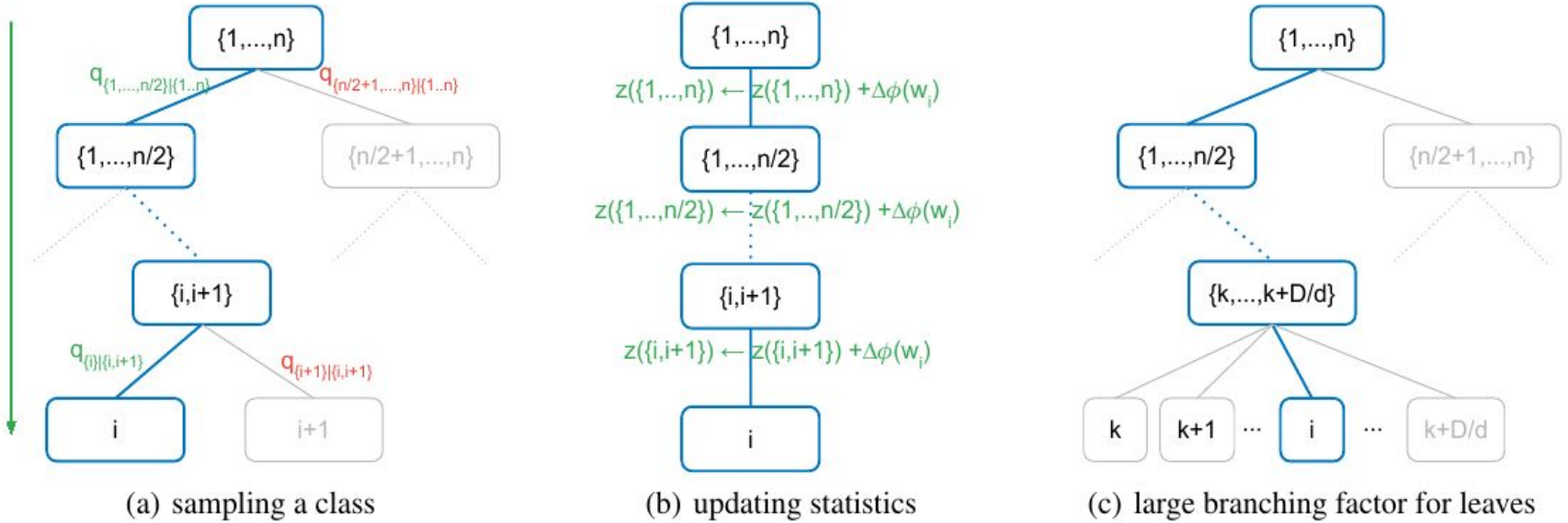
$$q_{C'|C} := \sum_{j \in C'} \frac{K(\boldsymbol{h}, \boldsymbol{w}_j)}{\sum_{l \in C} K(\boldsymbol{h}, \boldsymbol{w}_l)}$$

$$= \frac{\langle \phi(\boldsymbol{h}), \sum_{j \in C'} \phi(\boldsymbol{w}_j) \rangle}{\langle \phi(\boldsymbol{h}), \sum_{l \in C} \phi(\boldsymbol{w}_l) \rangle} = \frac{\langle \phi(\boldsymbol{h}), \boldsymbol{z}(C') \rangle}{\langle \phi(\boldsymbol{h}), \boldsymbol{z}(C) \rangle}$$

Divide and Conquer

Do the same thing recursively until an individual class is sampled.

This results in $\mathcal{O}(D \log_2 n)$ run time.



(a) sampling a class (b) updating statistics (c) large branching factor for leaves

Quadratic Kernel

One obvious choice for a kernel is a quadratic function:

$$K(\boldsymbol{h}, \boldsymbol{w}_i) = \alpha \langle \boldsymbol{h}, \boldsymbol{w}_i \rangle^2 + 1$$

It has the following feature representation:

$$\phi(\boldsymbol{a}) = \left[\sqrt{\alpha}\,\mathrm{vec}(\boldsymbol{a} \otimes \boldsymbol{a}), 1\right] \text{ with } D = O(d^2)$$

It is a poor approximation for negative logits, so we work with *absolute softmax* instead.

$$p_i = \frac{\exp(|o_i|)}{\sum_{j=1}^{n} \exp(|o_j|)}$$

Quadratic Kernel

One obvious choice for a kernel is a quadratic function:

$$K(\boldsymbol{h}, \boldsymbol{w}_i) = \alpha \langle \boldsymbol{h}, \boldsymbol{w}_i \rangle^2 + 1$$

It has the following feature representation:

$$\phi(\boldsymbol{a}) = \left[\sqrt{\alpha}\,\mathrm{vec}(\boldsymbol{a} \otimes \boldsymbol{a}), 1\right] \text{ with } D = O(d^2)$$

It is a poor approximation for negative logits, so we work with *absolute softmax* instead.

$$p_i = \frac{\exp(|o_i|)}{\sum_{j=1}^{n} \exp(|o_j|)}$$

Experimental Setup

- **Penn Tree Bank**
  Language modelling, 1M training words, vocabulary size 10K
  “Medium regularized LSTM” *(Zaremba et al., 2014)* with 200 units per layer
  Reporting perplexity loss
- **YouTube**
  Two recommendation datasets, with 10K videos (113M examples) and 100K videos (187M examples).
  A DNN is trained on user features and three previous videos.
  Cross-entropy loss is reported.

Experimental Setup

These three sampling distributions were tested:

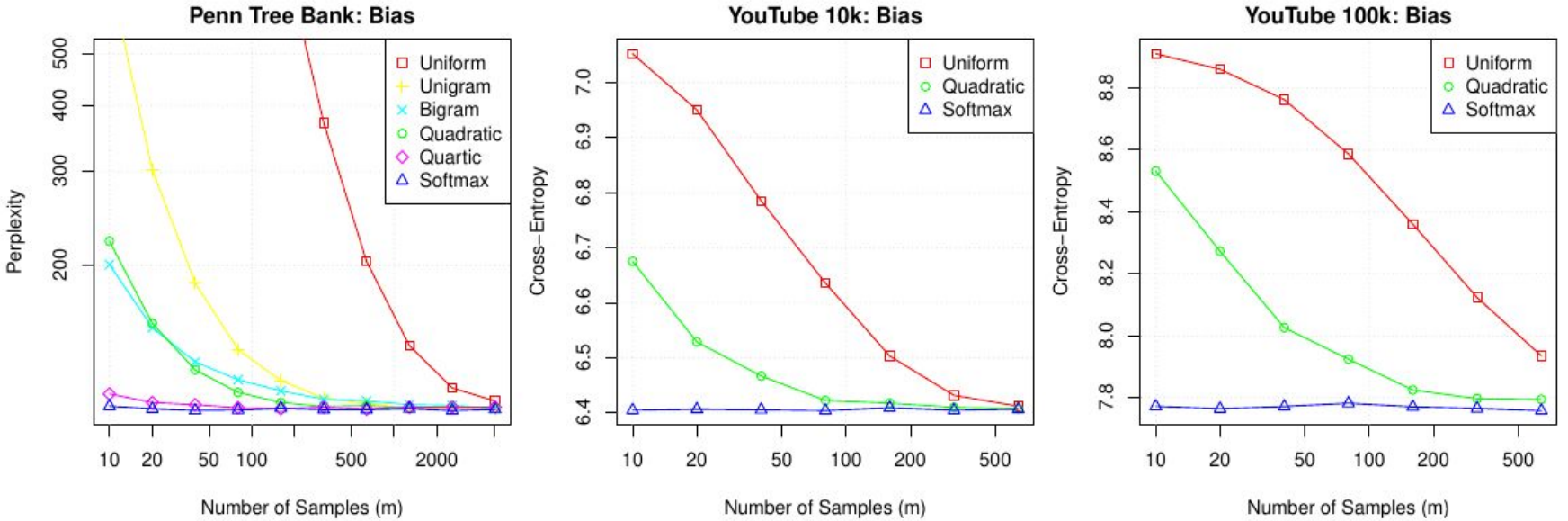
1. Uniform distribution, $q_i \propto 1$, where every class is sampled with the same probability. This provides a convenient baseline.

2. Softmax distribution, $q_i \propto \exp(o_i)$, which is the ideal sampling distribution as shown in Theorem 2.1, but is very expensive to sample from.

3. Quadratic distribution, $q_i \propto 100(o_i)^2 + 1$, as proposed in Section 3.3

Bias of Sampling

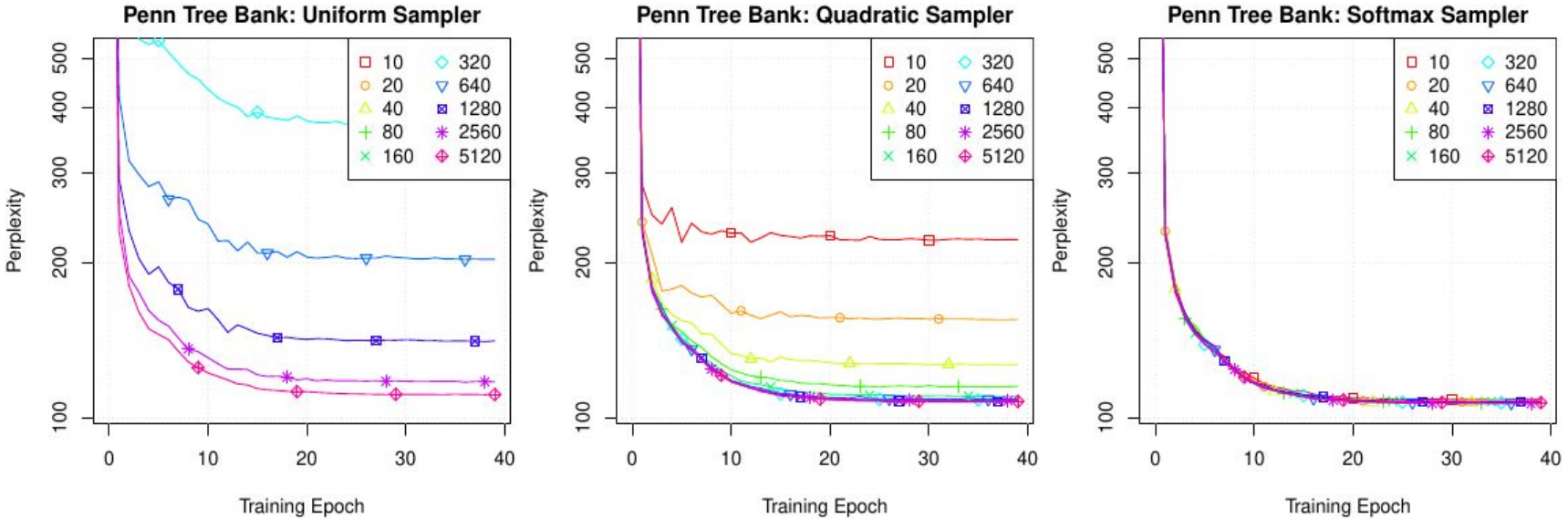
The bias decreases with the growth of sample size



But with different rates for different distributions

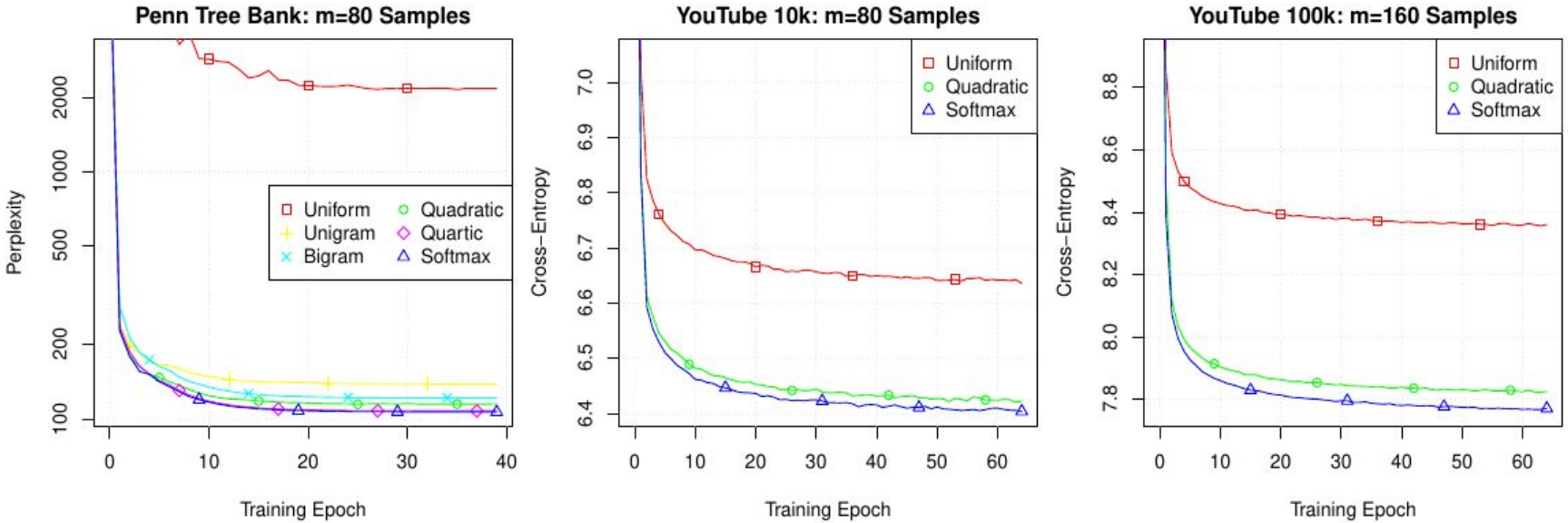
Convergence Speed *vs* Sample Size

The sample size influences the bias but not the convergence speed



Convergence Speed *vs* Sampling Distribution

The convergence speed is the same for different distributions



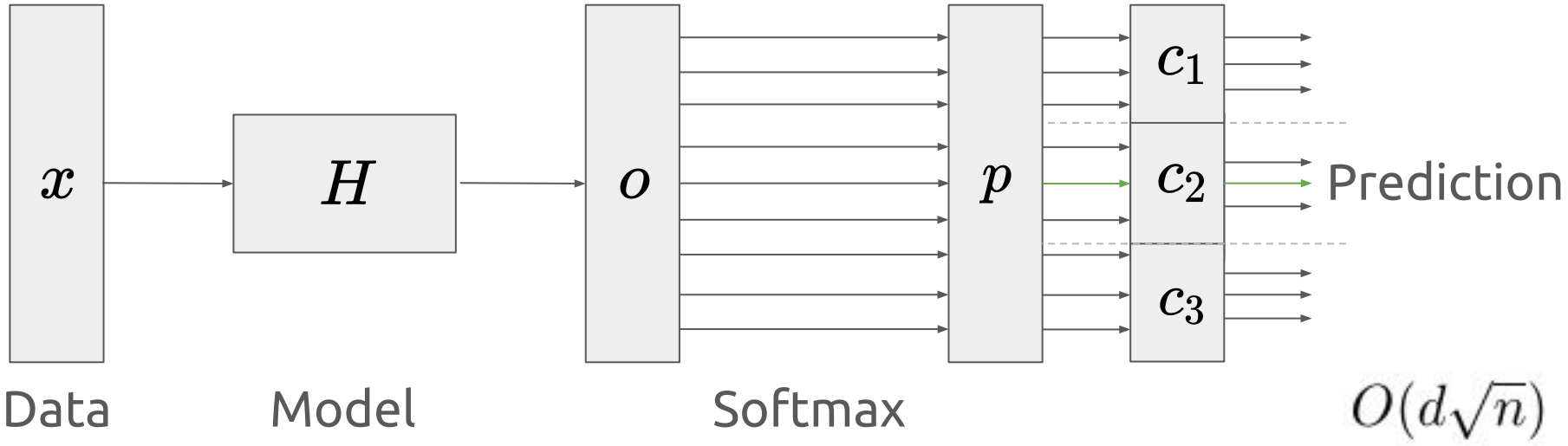
Related work

Sampled Softmax

- Learning the sampling distribution as a mixture of n-grams in a second model *(Bengio & Sénécal, 2008)*.
- Two-pass sampling: take a large sample of classes, e.g. 100K, then choose a smaller subset of classes, e.g. 1K, based on computed logits *(Bai et al., 2007)*. The logits can be computed faster using GPUs.
- ...

Related work

Hierarchical Softmax *(Goodman, 2001)*



Related work

Hierarchical Softmax and Variations

- Make a tree instead of the 2-level structure, get $O(d \log n)$ time for free* *(Morin & Bengio, 2005)*.
- How to build the tree? Class similarity *(Le et al., 2011)*, frequency binning *(Mikolov et al., 2011)*, optimize the speed of model *(Grave et al., 2017)*.

Cons:

- Worse at convergence.
- Slows down inference.

Related work

AdaptiveLogSoftmaxWithLoss

CLASS `torch.nn.AdaptiveLogSoftmaxWithLoss(`*`in_features`*`, `*`n_classes`*`, `*`cutoffs`*`, `*`div_value=4.0`*`, `*`head_bias=False`*`)` [SOURCE]

Efficient softmax approximation as described in Efficient softmax approximation for GPUs by Edouard Grave, Armand Joulin, Moustapha Cissé, David Grangier, and Hervé Jégou.

Adaptive softmax is an approximate strategy for training models with large output spaces. It is most effective when the label distribution is highly imbalanced, for example in natural language modelling, where the word frequency distribution approximately follows the Zipf's law.

Adaptive softmax partitions the labels into several clusters, according to their frequency. These clusters may contain different number of targets each. Additionally, clusters containing less frequent labels assign lower dimensional embeddings to those labels, which speeds up the computation. For each minibatch, only clusters for which at least one target is present are evaluated.

Conclusion

- Full softmax can be slow.
- Sampled softmax is an approximation that reduces computation complexity.
- Sampling distribution is very important: it affects how good the approximation is and how big the sample size should be.
- A good distribution is model and example dependent.
- Kernel based sampling is cool :)

- Hierarchical softmax could be applied, but it has its +/-