Approximating the Softmax

Problem:

We observed that mitigating the complexity of computing the final softmax layer has been one of the main challenges in devising better word embedding models, a commonality with machine translation (MT)

Origin softmax 的分母计算量太大.

Some Notations:

- T training words $w1, w2, w3, \cdots, wT$
- vocabulary V whose size is$|V|$
- a context c of n words context words.
- input embedding v_w(the eponymous word embedding in the Embedding Layer) with d dimensions 单词经过神经网络映射的 embedding
- output embedding v'_w (the representation of the word in the weight matrix of the softmax layer) 输入到 softmax 中的 embedding.
- objective function with regard to our model parameters θ.

Origin Softmax:

Computing the softmax is expensive as the inner product between h and the output embedding of every wordwi in the vocabulary V needs to be computed as part of the sum in the denominator in order to obtain the normalized probability of the target word w given its context c.

$$p(w|c) = \frac{\exp(h^\top v'_w)}{\sum_{w_i \in V} \exp(h^\top v'_{w_i})}$$

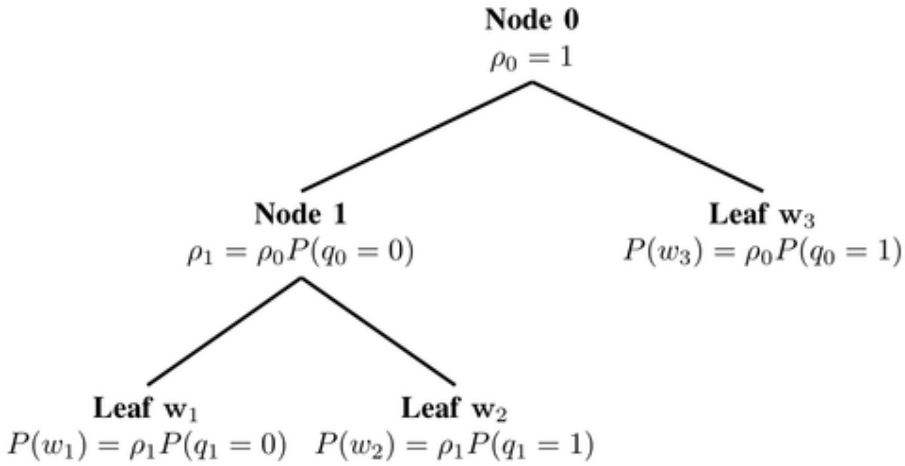
to generate word Embedding for word wi.

Approaches:

These approaches can be grouped into **softmax-based** and **sampling-based** approaches. Softmax-based approaches are methods that keep the softmax layer intact, but modify its architecture to improve its efficiency. Sampling-based approaches on the other hand completely do away with the softmax layer and instead optimise some other loss function that approximates the softmax.

Softmax Based

Hierarchical Softmax



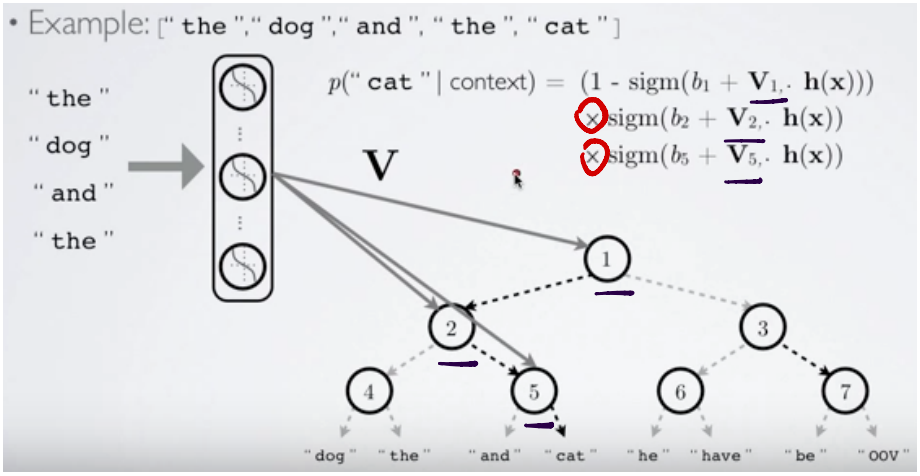
we only need to follow the path to the leaf node of that word, without having to consider any of the other nodes.

Note that this probability is already normalized, as the probabilities of all leaves in a binary tree sum to 1 and thus form a probability distribution. At each subsequent node, the probability mass is then split among its children, until it eventually ends up at the leaf nodes

In contrast to the regular softmax, we thus no longer have output embeddings v′w for every word w -- instead, we have embeddings v′nfor every node n.

$$p(\text{right}|n, c) = \sigma(h^\top v'_n)$$

instead of computing the dot product between h and the output word embedding v′w, we compute the dot product between h and the embedding v′wof each node in the tree.



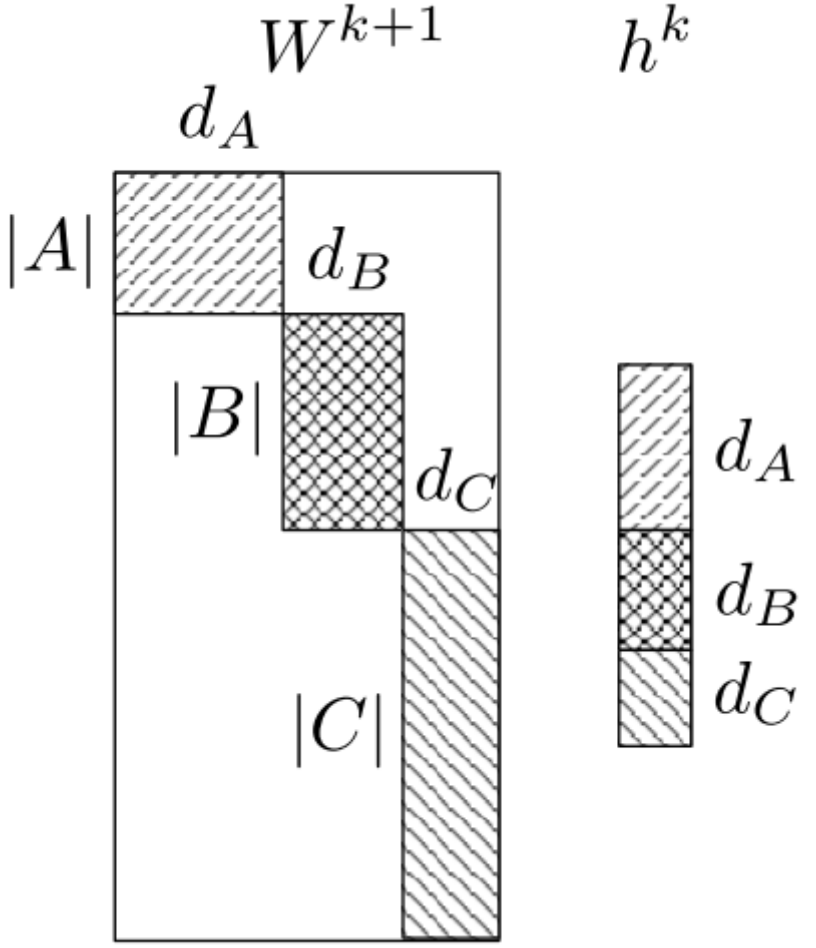
During testing, when we need to find the most likely prediction, we still need to calculate the probability of all words.

the structure of the tree

- Morin and Bengio use the synsets in WordNet as clusters for the tree
- Mnih and Hinton (2008) learn the tree structure with a clustering algorithm that recursively partitions the words in two clusters and allows them to achieve the same performance as the regular softmax at a fraction of the computation.
- by taking into account frequencies, we can reduce the average number of bits per word in the corpus from 13.3 to 9.16 in this case, which amounts to a speed-up of 31%. A **Huffman tree**, which is used by Mikolov et al. (2013) [6] for their hierarchical softmax, generates such a coding by **assigning fewer bits to more common symbols**. 频率越高，分配越短的路径

Differentiated Softmax

D-Softmax is based on the intuition that not all words require the same number of parameters: Many occurrences of frequent words allow us to fit many parameters to them, while extremely rare words might only allow to fit a few.



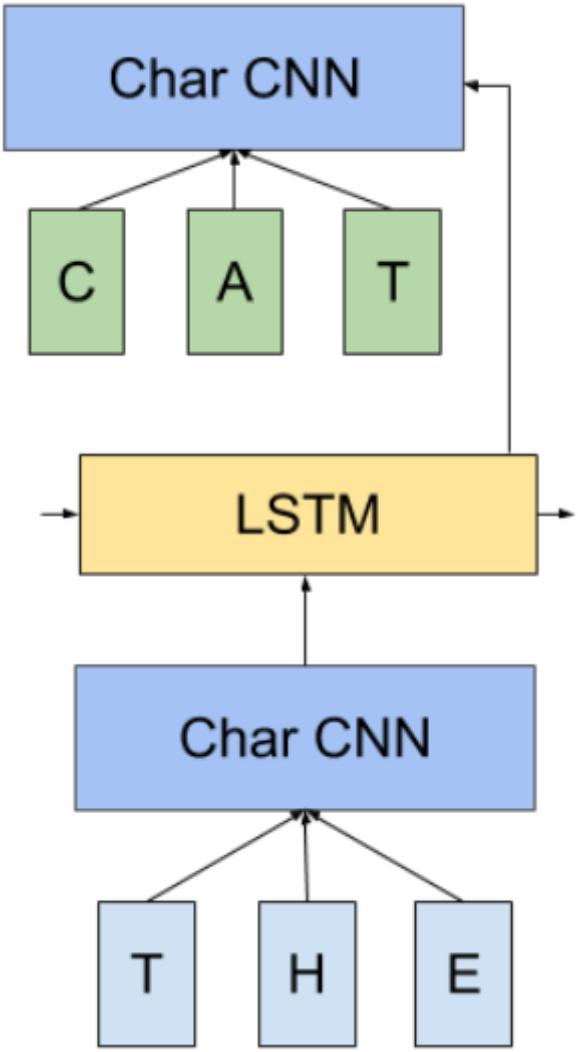
Words in blocks A have more parameters than in B,C...; The number of blocks and their embedding sizes are hyperparameters that can be tuned.

In contrast to H-Softmax, this speed-up persists during testing.

It assigns fewer parameters to rare words, D-Softmax does a worse job at modelling them.

CNN-Softmax

produce input word embeddings w_v via a character-level CNN. Jozefowicz et al. (2016) in turn suggest to do the same thing for the output word embeddings v'_w via a character-level CNN.



we now only need to keep track of the parameters of the CNN. During testing, the output word embeddings v'_w can be pre-computed, so that there is no loss in performance. It's difficult to differentiate between similarly spelled words with different meanings. To mitigate this, the authors add a correction factor that is learned per word, which significantly reduces the performance gap between regular and CNN-softmax.

Sampling-based → training time.

Sampling-based approaches are only useful at training time -- during inference, the full softmax still needs to be computed to obtain a normalised probability.

$$J_\theta = -\log \frac{\exp(h^\top v'_w)}{\sum_{w_i \in V} \exp(h^\top v'_{w_i})}$$

we replace the dot product $h^\top v'_w$ with $-\mathcal{E}(w)$, we can now compute the gradient ∇ of J_θ w.r.t. our model's parameters θ:

$$\nabla_\theta J_\theta = \nabla_\theta \mathcal{E}(w) - \sum_{w_i \in V} P(w_i) \nabla_\theta \mathcal{E}(w_i)$$

期望.

Bengio and Senécal (2003) note that the gradient essentially has two parts: a positive reinforcement for the target word w (the first term in the above equation) and a negative reinforcement for all other words wi, which is weighted by their probability (the second term). As we can see, this negative reinforcement is just the expectation $\mathbb{E}_{w_i \sim P}$ of the gradient of $\mathcal{E}$ for all words wi in V.

$$\sum_{w_i \in V} P(w_i) \nabla_\theta \mathcal{E}(w_i) = \mathbb{E}_{w_i \sim P}[\nabla_\theta \mathcal{E}(w_i)]$$

The crux of most sampling-based approach now is to approximate this negative reinforcement in some way to make it easier to compute, since we don't want to sum over the probabilities for all words in V.

methods

We can approximate the expected value $\mathbb{E}$ of any probability distribution using the Monte Carlo method.

Monte Carlo method

By taking the mean of random samples of the probability distribution. If we knew the network's distribution, i.e. P(w), we could thus directly sample mm words $w_1, \cdots, w_m$ from it and approximate the above expectation with:

$$\mathbb{E}_{w_i \sim P}[\nabla_\theta \mathcal{E}(w_i)] \approx \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \mathcal{E}(w_i)$$ ⇒从真实分布 P(w)中采样

However, in order to sample from the probability distribution P, we need to compute P, which is just what we wanted to avoid in the first place. We therefore have find some other distribution Q(we call this the proposal distribution), from which it is cheap to sample and which can be used as the basis of Monte-Carlo sampling. Preferably, Q should also be similar to P, since we want our approximated expectation to be as accurate as possible.

近似可代替分布 P 的 Q.

Importance Sampling

A straightforward choice in the case of language modelling is to simply use the unigram distribution of the training set for Q. This is essentially what classical Importance Sampling (IS) does: It uses Monte-Carlo sampling to approximate a target distribution P via a proposal distribution Q. However, this still requires computing P(w) for every word w that is sampled. To avoid this, Bengio and Senécal (2003) use **a biased estimator** that was first proposed by Liu (2001) [13].

因为 $\mathbb{E}_{w_i \sim P}[\nabla_\theta \mathcal{E}(w_i)] = \sum_{w_i} P(w_i) \nabla_\theta \mathcal{E}(w_i)$

w_i 采样出的样本.

Our biased estimator that approximates the expectation thus looks like the following:

$$\mathbb{E}_{w_i \sim P}[\nabla_\theta \mathcal{E}(w_i)] \approx \frac{1}{R} \sum_{i=1}^{m} r(w_i) \nabla_\theta \mathcal{E}(w_i)$$

weight $\nabla_\theta \mathcal{E}(w_i)$ with a factor (利用Q)

$r(w) = \frac{\exp(-\mathcal{E}(w))}{Q(w)}$ ⇒ $\frac{1}{R} r(w_i)$

$R = \sum_{j=1}^{m} r(w_j)$

Adaptive Importance Sampling

和IS不一样的地方.

they propose an n-gram distribution that is adapted during training to follow the target distribution PPmore closely. To this end, they interpolate a bigram distribution and a unigram distribution according to some mixture function

Target Sampling

In order to make the method more suitable for processing on a GPU with limited memory, they limit the number of target words that need to be sampled from. They do this by partitioning the training set and including only a fixed number of sample words in every partition, which form a subset V′of the vocabulary.

Noise Contrastive Estimation

a more stable sampling method than Importance Sampling (IS), as we have seen that IS poses the risk of having the proposal distribution Q diverge from the distribution PP that should be optimized. In contrast to the former, NCE does not try to estimate the probability of a word directly. Instead, **it uses an auxiliary loss that also optimises the goal of maximizing the probability of correct words.**

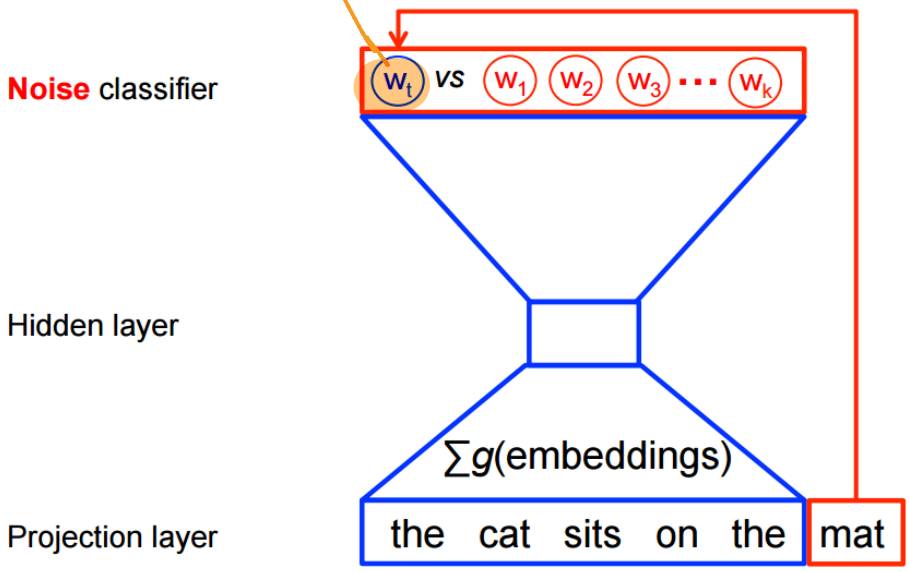
IS存在跟P不一致的风险.

最大化正确词的概率.

不估计 P(w|C)

二分类

We can thus reduce the problem of predicting the correct word to a binary classification task, where the model tries to distinguish positive, genuine data from noise samples, as can be seen in Figure below.



Similarity between NCE and IS

两者是非常相似的，NCE使用的是多个二分类达到多分类的功能，当IS使用softmax+CE，即采用多分类思想时，就能得到相似的效果；对于多分类问题，采用IS更合适，因为这个softmax+CE的loss能达到关联性的更新正负样本，而不是像NCE一样独立的更新。

NCE推导公式：

1) use **logistic regression** to minimize the negtive log-likelihood.

$$J(\theta) = -\sum_{w_i \in V}\left[\log P(y=1 \mid w_i, C_i) + k\, \mathbb{E}_{\tilde{w}_{ik}\sim Q}\left[\log P(y=0 \mid \tilde{w}_{ij}, C_i)\right]\right]$$

采样个数。　Q：a noise distribution.

2) $\mathbb{E}_{\tilde{w}_{ik}\sim Q}$ 仍然需要在整个V上计算，因此仍然应用 Monte Carlo，求平均：

$$J(\theta) = -\sum_{w_i \in V}\left[\log P(y=1 \mid w_i, C_i) + k \sum_{j=1}^{k} \frac{1}{k} \log P(y=0 \mid \tilde{w}_{ij}, Q)\right]$$

3) Two distributions P、Q → correct words / noise samples.

对context的w，来自真实分布　　对w的采样

$$P(y, w \mid C) = \frac{1}{k+1} P(w \mid C) + \frac{k}{k+1} Q(w)$$

$$\Downarrow$$

$$P_{train} = P(y=1 \mid w, C) = \frac{\frac{1}{k+1} P(w \mid C)}{P(y, w \mid C)}$$

$$= \frac{P(w \mid C)}{P(w \mid C) + kQ(w)}$$

$$P_{noise} = 1 - P_{train}$$

4) $p(w \mid C) = \dfrac{\exp(h^T v'_w)}{Z(C)}$　need to sum over V.

NCE的解决方法：keep Z(C) fixed at 1. ⇒

$$p(w \mid C) = \exp(h^T v'_w)$$

$$P(y=1 \mid w_i, C_i) = \frac{\exp(h^T v'_w)}{\exp(h^T v'_w) + kQ(w)}$$ ⇒ 代回NCE loss function：

$$J_\theta = -\sum_{w_i \in V}\left[\log(\cdots) + \sum_{j=1}^{k} \log(1 - \cdots)\right]$$

注：这里的Q被当成噪声分布，不像IS一样，要求Q与P近似。

> *They observe that as IS performs multi-class classification, it may be a better choice for language modelling, as the loss leads to tied updates between the data and noise samples rather than independent updates as with NCE.*

Negative Sampling (NEG)

Been popularised by Mikolov et al. (2013), can be seen as an approximation to NCE. NEG simplifies NCE and does away with this guarantee, as the objective of NEG is to learn high-quality word representations rather than achieving low perplexity on a test set, as is the goal in language modelling. The key difference to NCE is that NEG only approximates this probability by making it as easy to compute as possible. For this reason, it sets the most expensive term, kQ(w) to 1, can be transformed into the sigmoid function:

$$P(y=1|w,c) = \frac{1}{1+\exp(-h^\top v'_w)}$$

We have seen that NEG is only equivalent to NCE when $k = |V|$ and Q is uniform. In all other cases, NEG only approximates NCE, which means that it will not directly optimise the likelihood of correct words, which is key for language modelling. While NEG may thus be useful for learning word embeddings, its lack of asymptotic consistency guarantees makes it **inappropriate for language modelling**. 因为word embedding是language modeling的副产物，如果主要不是为了获得word embedding的话，用~~NEG~~更好。

Self-Normalisation

Is not a sampling-based approach. Recall that our loss function :

$$J_\theta = -\sum_i [\log \frac{\exp(h^\top v'_{w_i})}{Z(c)}]$$

If we are able to constrain our model so that it sets Z(c)=1 or similarly logZ(c)=0, then we can avoid computing the normalization in Z(c) altogether. Devlin et al. (2014) thus propose to add a squared error penalty term to the loss function that encourages the model to keep logZ(c) as close as possible to 0:

$$J_\theta = -\sum_i [h^\top v'_{w_i} + \log Z(c) - \alpha(\log(Z(c)) - 0)^2]$$

Devlin et al. (2014) then set the denominator of the softmax to 11 and only use the numerator for computing P(w|c) together with their penalty term:

$$J_\theta = -\sum_i [h^\top v'_{w_i} - \alpha \log^2 Z(c)]$$

They report that self-normalisation achieves a speed-up factor of about 1515, while only resulting in a small degradation of BLEU scores compared to a regular non-self-normalizing neural language model.

Infrequent Normalisation

It should even be sufficient to only normalise a fraction of the training examples and still obtain approximate self-normalising behaviour. They thus propose Infrequent Normalisation (IN), which down-samples the penalty term, making this a sampling-based approach.

$$J_\theta = -\sum_i h^\top v'_{w_i} + \frac{\alpha}{\gamma} \sum_{c_j \in C} \log^2 Z(c_j)$$

Other Approaches

We have thus not paid particular attention to $h^\top v'_w$, i.e. the dot-product between the penultimate layer representation h and output word embedding v'_w. .....

Which Approach to Choose?

APPROACH	SPEED-UP FACTOR	DURING TESTING?	PERFORMANCE (SMALL VOCAB)	DURING TRAINING?	PERFORMANCE (LARGE VOCAB)	PROPORTION OF PARAMETERS
Softmax	1x	-	very good	-	very poor	100%
Hierarchical Softmax	25x (50-100x)	-	very poor	X	very good	100%
Differentiated Softmax	2x	X	very good	X	very good	< 100%
CNN-Softmax	-	-	-	X	bad - good	30%
Importance Sampling	(19x)	-	-	X	-	100%
Adaptive Importance Sampling	(100x)	-	-	X	-	100%
Target Sampling	2x	-	good	X	bad	100%
Noise Contrastive Estimation	8x (45x)	-	very bad	X	very bad	100%
Negative Sampling	(50-100x)	-	-	X	-	100%
Self-Normalisation	(15x)	-	-	X	-	100%
Infrequent Normalisation	6x (10x)	-	very good	X	good	100%

Table 1: Comparison of approaches to approximate the softmax for language modelling.