What is Candidate Sampling

Say we have a multiclass or multi-label problem where each training example (x_i, T_i) consists of a context x_i a small (multi)set of target classes T_i out of a large universe L of possible classes. For example, the problem might be to predicting the next word (or the set of future words) in a sentence given the previous words.

We wish to learn a compatibility function $F(x,y)$ which says something about the compatibility of a class y with a context x. For example - the probability of the class given the context.

"Exhaustive" training methods such as softmax and logistic regression require us to compute $F(x,y)$ for every class $y \in L$ for every training example. When $|L|$ is very large, this can be prohibitively expensive.

"Candidate Sampling" training methods involve constructing a training task in which for each training example (x_i, T_i), we only need to evaluate $F(x,y)$ for a small set of candidate classes $C_i \subset L$. Typically, the set of candidates C_i is the union of the target classes with a randomly chosen sample of (other) classes $S_i \subset L$.

$$C_i = T_i \bigcup S_i$$

The random choice of S_i may or may not depend on x_i and/or T_i.

The training algorithm takes the form of a neural network, where the layer representing $F(x,y)$ is trained by back-propagation from a loss function.

Table of Candidate Sampling Algorithms

	Positive training classes associated with training example (x_i, T_i): $POS_i =$	Negative training classes associated with training example (x_i, T_i): $NEG_i =$	Input to Training Loss $G(x,y) =$	Training Loss	$F(x,y)$ gets trained to approximate:
Noise Contrastive Estimation (NCE)	T_i	S_i	$F(x,y) - log(Q(y\|x))$	Logistic	$log(P(y\|x))$
Negative Sampling	T_i	S_i	$F(x,y)$	Logistic	$log\left(\frac{P(y\|x)}{Q(y\|x)}\right)$
Sampled Logistic	T_i	$(S_i - T_i)$	$F(x,y) - log(Q(y\|x))$	Logistic	$logodds(y\|x) = log\left(\frac{P(y\|x)}{1-P(y\|x)}\right)$
Full Logistic	T_i	$(L - T_i)$	$F(x,y)$	Logistic	$log(odds(y\|x)) = log\left(\frac{P(y\|x)}{1-P(y\|x)}\right)$
Full Softmax	$T_i = \{t_i\}$	$(L - T_i)$	$F(x,y)$	Softmax	$log(P(y\|x)) + K(x)$
Sampled Softmax	$T_i = \{t_i\}$	$(S_i - T_i)$	$F(x,y) - log(Q(y\|x))$	Softmax	$log(P(y\|x)) + K(x)$

- $Q(y|x)$ is defined as the probability (or expected count) according to the sampling algorithm of the class y in the (multi-)set of sampled classes given the context x.
- $K(x)$ is an arbitrary function that does not depend on the candidate class. Since Softmax involves a normalization, addition of such a function does not affect the computed probabilities.
- $$logistic\ training\ loss = \sum_i \left(\sum_{y \in POS_i} log(1 + exp(-G(x_i, y)) + \sum_{y \in NEG_i} log(1 + exp(G(x_i, y)) \right)$$
- $$softmax\ training\ loss = \sum_i \left(-G(x_i, t_i) + log\left(\sum_{y \in POS_i \cup NEG_i} exp(G(x_i, y)) \right) \right)$$
- NCE and Negative Sampling generalize to the case where T_i is a multiset. In this case, $P(y|x)$ denotes the expected count of y in T_i. Similarly, NCE, Negative Sampling, and Sampled Logistic generalize to the case where S_i is a multiset. In this case $Q(y|x)$ denotes the expected count of y in S_i.

Sampled Softmax

(A faster way to train a softmax classifier)

Reference: http://arxiv.org/abs/1412.2007

Assume that we have a single-label problem. Each training example $(x_i, \{t_i\})$ consists of a context and one target class. We write $P(y|x)$ for the probability of that the one target class is y given that the context is x.

We would like to train a function $F(x,y)$ to produce softmax logits - that is, relative log probabilities of the class given the context:

$$F(x,\ y)\ \leftarrow\ log(P(y|x)) + K(x)$$

Where $K(x)$ is an arbitrary function that does not depend on y.

In full softmax training, for every training example $(x_i, \{t_i\})$, we would need to compute logits $F(x_i, y)$ for all classes in $y \in L$. This can get expensive if the universe of classes L is very large.

In "Sampled Softmax", for each training example $(x_i, \{t_i\})$, we pick a small set $S_i \subset L$ of "sampled" classes according to a chosen sampling function $Q(y|x)$. Each class $y \in L$ is included in S_i independently with probability $Q(y|x_i)$.

$$P(S_i = S|x_i) = \prod_{y \in S} Q(y|x_i) \prod_{y \in (L-S)} (1 - Q(y|x_i))$$

We create a set of candidates C_i containing the union of the target class and the sampled classes:

$$C_i = S_i \bigcup \{t_i\}$$

Our training task is to figure out, given this set C_i, which of the classes in C_i is the target class.

For each class $y \in C_i$, we want to compute the posterior probability that y is the target class given our knowledge of x_i and C_i. We call this $P(t_i = y|x, C_i)$

Applying Bayes' rule:

$$\begin{aligned} P(t_i = y|x_i, C_i) &= P(t_i = y, C_i|x_i) \,/\, P(C_i|x_i) \\ &= P(t_i = y|x_i)\, P(C_i|t_i = y, x_i) \,/\, P(C_i|x_i) \\ &= P(y|x_i)\, P(C_i|t_i = y,\ x_i) \,/\, P(C_i|x_i) \end{aligned}$$

Now to compute $P(C_i|t_i = y,\ x_i)$, we note that in order for this to happen, S_i may or may not contain y, must contain all other elements of C_i, and must not contain any classes not in C_i. So:

$$P(t_i = y|x_i,\ C_i) = P(y|x_i) \prod_{y' \in C_i - \{y\}} Q(y'|x_i) \prod_{y' \in (L - C_i)} (1 - Q(y'|x_i)) \ \ / P(C_i|x_i)$$

$$= \frac{P(y|x_i)}{Q(y|x_i)} \prod_{y' \in C_i} Q(y'|x_i) \prod_{y' \in (L - C_i)} (1 - Q(y'|x_i)) \ \ / P(C_i|x_i)$$

$$= \frac{P(y|x_i)}{Q(y|x_i)} \ \ / K(x_i, C_i)$$

where $K(x_i, C_i)$ is a function that does not depend on y. So:

$$log(P(t_i = y|x_i, C_i)) = log(P(y|x_i)) \ - \ log(Q(y|x_i)) \ + \ K'(x_i, C_i)$$

These are the relative logits that should feed into a softmax classifier predicting which of the candidates in C_i is the true one.

Since we are trying to train the function $F(x,y)$ to approximate $log(P(y|x))$, we take the layer in our network representing $F(x,y)$, subtract $log(Q(y|x))$, and pass the result to a softmax classifier predicting which candidate is the true one.

$$Training\ Softmax\ Input\ = F(x,y)\ - log(Q(y|x)$$

Backpropagating the gradients from that classifier trains F to give us what we want.

Noise Contrastive Estimation (NCE)

Reference: http://www.jmlr.org/proceedings/papers/v9/gutmann10a/gutmann10a.pdf

Each training example (x_i, T_i) consists of a context and a small multiset of target classes. In practice, T_x may always be a set or even a single class, but we use a multiset here for generality.

We use the following as a shorthand for the expected count of a class in the set of target classes for a context. In the case of sets with no duplicates, this is the probability of the class given the context:

$$P(y|x) := E(T(y) \mid x)$$

We would like to train a function $F(x,y)$ to approximate the log expected count of the class given the context, or in the case of a sets, the log probability of the class given the context.

$$F(x,y) \leftarrow log\,(P(y|x))$$

For each example (x_i, T_i), we pick a multiset of sampled classes S_i. In practice, it probably makes sense to pick a set, but we use a multiset here for generality. Our sampling algorithm may or may not depend on x_i but may not depend on T_i. We construct a multiset of candidates consisting of the sum of the target classes and the sampled classes.

$$C_i = T_i + S_i$$

Our training task is to distinguish the true candidates from the sampled candidates. We have one positive training meta-example for each element of T_i and one negative training meta-example for each element of S_i.

We introduce the shorthand $Q(y|x)$ to denote the expected count, according to our sampling algorithm, of a particular class in the set of sampled classes. If S never contains duplicates, then this is a probability.

$$Q(y|x) := E(S(y) \mid x))$$

$$logodds(y\ came\ from\ T\ vs\ S \mid x) = log\left(\frac{P(y|x)}{Q(y|x)}\right)$$
$$= log\,(P(y|x)) - log(Q(y|x))$$

The first term, $log\,(P(y|x))$, is what we would like to train $F(x,y)$ to estimate.

We have a layer in our model which represents $F(x,y)$. We add to it the second term, $-log(Q(y|x))$, which we compute analytically, and we pass the result to a logistic regression loss whose "label" indicates whether y came from T as opposed to S.

$$Logistic\ Regression\ Input = F(x,y) - log(Q(y|x))$$

The backpropagation signal trains $F(x,y)$ to approximate what we want it to.

Negative Sampling

Reference:
http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf

Negative sampling is a simplified variant of Noise Contrastive Estimation where we neglect to subtract off $log(Q(y|x))$ during training. As a result, $F(x,y)$ is trained to approximate $log\,(E(y|x)) - log(Q(y|x))$.

It is noteworthy that in Negative Sampling, we are optimizing $F(x,y)$ to approximate something that depends on the sampling distribution Q . This will make the results highly dependent on the choice of sampling distribution. This is not true for the other algorithms described here.

Sampled Logistic

Sampled Logistic is a variant on Noise Contrastive Estimation where we discard without replacement all sampled classes that happen to also be target classes. This requires T_i to be a set, as opposed to a multiset, though S_i may be a multiset. As a result we learn an estimator of the log-odds of a class as opposed to the log-probability of a class. The math changes from the NCE math as follows:

$$logodds(y\ came\ from\ T\ vs\ (S-T)\ |\ x) = log\left(\frac{P(y|x)}{Q(y|x)(1-P(y|x)}\right) = log\left(\frac{P(y|x)}{1-P(y|x)}\right) - log(Q(y|x))$$

The first term, $log\left(\frac{P(y|x)}{1-P(y|x)}\right)$, is what we would like to train $F(x,y)$ to estimate.

We have a layer in our model, which represents $F(x,y)$. We add to it the second term, $-log(Q(y|x))$, which we compute analytically, and we pass the result to a logistic regression loss predicting whether y came from T_i vs $(S_i - T_i)$.

$$Logistic\ Regression\ Input = F(x,y) - log(Q(y|x)$$

The backpropagation signal trains the $F(x,y)$ layer to approximate what we want it to.

$$F(x,y) \leftarrow log\left(\frac{P(y|x)}{1-P(y|x)}\right)$$

Context-Specific vs. Generic Sampling

In the methods discussed, the sampling algorithm is allowed to depend on the context. It is possible that for some models, context-specific sampling will be very useful, in that we can generate context-dependent hard negatives and provide a more useful training signal. The authors have to this point focused on generic sampling algorithms such as uniform sampling and unigram sampling, which do not make use of the context. The reason is described in the next section.

Batchwise Sampling

We have focused on models which use the same set S of sampled classes across a whole batch of training examples. This seems counterintuitive - shouldn't convergence be faster if we use different sampled classes for each training example? The reason for using the same sampled classes across a batch is computational.

In many of our models, $F(x, y)$ is computed as the dot product of a feature vector for the context (the top hidden layer of a neural network), and an embedding vector for the class. Computing the dot products of many feature vectors with many embedding vectors is a matrix multiplication, which is highly efficient on modern hardware, especially on GPUs. Batching like this often allows us to use hundreds or thousands of sampled classes without noticeable slowdown.

Another way to see it is that the overhead of fetching a class embedding across devices is greater than the time it takes to compute its dot products with hundreds or even thousands of feature vectors. So if we are going to use a sampled class with one context, it is virtually free to use it with all of the other contexts in the batch as well.