Slide Set 2:
Probabilistic Neural Language Models (PNLM)

Alvaro Soto

Computer Science Department (DCC), PUC

NPLM and Big Data

Bottleneck

- Previous NPLM solutions outperforms n-grams approaches.
- However, moving scores $s(w, h(w))$ to probs using a soft-max expression imposes a high computational complexity.
- Training NPLMs is computationally expensive because they are explicitly normalized, which leads to having to consider all words in the vocabulary when computing the log-likelihood gradients.
- As a consequence, models do not scale properly to large datasets, in particular, large vocabularies.

Possible solutions

- Importance sampling schemes can be used to reduce the problem (Bengio et al., 2003).
- Hierarchical softmax approaches also alleviate the problem.
- So far, one of the best solutions is provided by Noise-Contrastive approaches (Gutmann and Hyvarinen, JMLR, 2012). We will discuss this method here.

Noise-Contrastive Estimation (NCE)

- In cases where density estimation is performed using a family of probability distributions, scaling (probs must sum to 1) is not a problem (why?). Then a ML approach can be used to estimate relevant shape params.
- In case of an unnormalized hypothesis space, density estimation can be decomposed into 2 main tasks:
  - Estimating the shape of the density.
  - Scaling the density so probs integrate to 1.
- This is the case of the previous NPLM approach, where the neural model estimates the shape of the density and the partition function performs the scaling.

- NCE aims to learn the partition function as a model parameter. So it solves:

$$P_m(z;\theta) = \bar{P}_m(z;\alpha) + \mathcal{Z}, \quad \theta = \{\alpha, \mathcal{Z}\}$$

  $\bar{P}_m(z;\alpha)$ corresponds to an unnormalized model that matches the shape of $P_m(z;\theta)$ and $\mathcal{Z}$ (or $log\mathcal{Z}$) is the scaling parameter.
- A ML approach can't be used to estimate $\theta = \{\alpha, \mathcal{Z}\}$ because the likelihood can increase arbitrarily by setting $\mathcal{Z}$ to a suitable value.

Noise-Contrastive Estimation (NCE)

IDEA: Estimate the shape of the target density by learning to discriminate between samples from the data distribution and samples from a known noise distribution.

This idea is similar to the ranking approach used to train multiclass SVMs. As an example, Collobert & Weston (2008) replace the typical ML based estimation of NPLM models by a loss function that encourages to fulfill a min-margin condition.

In particular, they estimate a language model using a cost function based on a hinge loss that encourages to score correct n-grams higher than randomly chosen incorrect n-grams (at least by the min margin of 1):

$$\sum_{(w_t,h(w_t))\in D} \sum_{w_i \notin (w_t,h(w_t))} max\{0, 1 - f(w_t, h(w_t)) + f(w_i, h(w_t))\}$$

where $f(\cdot)$ is given by a neural model and $(w_t, h(w_t))$ represents the correct n-grams in training set D.

Noise-Contrastive Estimation (NCE)

IDEA: Estimate the shape of the target density by learning to discriminate between samples from the data distribution and samples from a known noise distribution.

In contrast to the margin loss, NCE learns the parameters θ of a data distribution $p_m(z;\theta)$, such that it is possible to identify if a given sample is coming from $p_m(z;\theta)$ or from a known noise distribution $p_n(z)$.

Using this idea we can estimate parameters θ through a ML approach, but in this case we use a "learning by comparison" scheme.

As a relevant advantage, in practice, the resulting distribution $p_m(z;\theta)$ models the data distribution and it does not need to be scaled.

Next, we will discuss first the general operation of the NCE technique. Afterwards, we will see how we can use NCE in the context of a NPLM application.

Noise-Contrastive Estimation (NCE)

Let z: random data sample.

$p_m(z;\theta)$: unknown data distribution of parameter θ.

$p_n(z)$: known auxiliary (noise) distribution.

$C \in [0,1]$: label indicating if z is from data ($C = 1$) or noise distributions ($C = 0$).

So we have:

$$p(z|C=1;\theta) = p_m(z;\theta)$$
$$p(z|C=0) = p_n(z)$$

If we assume that samples from p_n are K times more frequent than samples from p_m, then each sample z is coming from the mixture:

$$p(z;\theta) \sim \frac{1}{k+1} p_m(z;\theta) + \frac{k}{k+1} p_n(z)$$

Then the posterior distribution for C given z is given by:

$$p(C=1|z;\theta) = \frac{p(z|C=1;\theta)p(C=1)}{p(z)} = \frac{p_m(z;\theta)\frac{1}{k+1}}{\frac{1}{k+1}p_m(z;\theta) + \frac{k}{k+1}p_n(z)}$$

$$p(C=1|z;\theta) = \frac{p_m(z;\theta)}{p_m(z;\theta) + kp_n(z)}$$

Noise-Contrastive Estimation (NCE)

Assuming labels C_t are iid (Bernoulli), and we have T_d and T_n observations from p_m and p_n, respectively, the conditional likelihood over training set D is given by:

$$\ell(\theta) = \prod_{(C_t, z_t) \in D} P(C_t | z_t; \theta)$$

So the conditional log-likelihood $L(\theta)$ is given by:

$$L(\theta) = \sum_{z_t \in T_d} logP(C_t = 1 | z_t; \theta) + \sum_{z_t \in T_n} logP(C_t = 0 | z_t; \theta)$$

$$L(\theta) = \sum_{z_t \in T_d} logP(C_t = 1 | z_t; \theta) + \sum_{z_t \in T_n} log(1 - P(C_t = 1 | z_t; \theta)) \quad (1)$$

As a relevant observation, it is possible to write $p(C = 1 | z; \theta)$ as a logistic function of the log-ratio between P_m and P_n:

$$P(C = 1 | z; \theta) = \frac{P_m(z; \theta)}{P_m(z; \theta) + kP_n(z)} = \frac{1}{1 + k \frac{P_n(z)}{P_m(z; \theta)}}$$

$$P(C = 1 | z; \theta) = \frac{1}{1 + ke^{(-log \frac{P_m(z; \theta)}{P_n(z)})}} = \frac{1}{1 + ke^{-u}},$$

where u corresponds to the log-ratio between P_m and P_n:

$$u = log \frac{P_m(z; \theta)}{P_n(z)}$$

Noise-Contrastive Estimation (NCE)

- Maximization of Eq.(1) has an upper bound given by correct identification of data and noise samples.
- In this sense, the last expression shows that we can find a ML estimator of θ in Eq.(1) by solving a logistic regression problem that learns how to identify samples from the data and noise distributions.
- The approach solves then a density estimation problem, which is an unsupervised learning problem, via supervised learning ("learning by comparison")
- Noise distributions that are more similar to the true distribution poses a more difficult classification problem, then a more precise value of θ. Similar to a hard mining sampling scheme.
- In fact, one could choose a noise distribution by first estimating a preliminary model of the data, and then use this preliminary model as the noise distribution.
- As a wishful property, noise distribution should be easy to sample from and should provide analytical expressions to evaluate the log-pdf.
- There are some mild conditions to check consistency of the estimator see Gutmann and Hyvarinen JMLR, 2012. An important requirement is that P_n has same support than P_m.

NCE for a NPLM application (Mnih and Teh, 2012)

- Suppose we would like to learn the distribution of words $P_m^h(w)$ for some specific context h.
- We can use NCE by introducing a noise distribution $P_n(w)$.
- Mnih and Teh, ICML 2012, propose to define $P_n(w)$ as the unigram distribution over the training data ($P_n(w) \approx \frac{\#w}{\#TotalWords}$).
- $P_m(w)$ is given by the unnormalized version of the softmax function for the score that relates a word w with its context h plus the normalization parameter $\mathcal{Z}$, i.e., $P_m^h(w) = s_\alpha(w, h(w)) + \mathcal{Z}$.
- Let consider the likelihood of a specific context h and ignore for the moment that context functions share parameters, we have:

$$L^h(\theta) = \sum_{T_d} logP(C_t = 1|w;\theta) + \sum_{T_n} log(1 - P(C_t = 1|w;\theta))$$

where $\theta = \{\alpha, \mathcal{Z}\}$ and $P(C_t = 1|w;\theta) = \frac{P_m^h(w;\theta)}{P_m^h(w;\theta)+kP_n(w)}$

Solution Based on Stochastic Gradient Descent (SGD)

$$L^h(\theta) = \sum_{T_d} log \frac{P_m^h(w;\theta)}{P_m^h(w;\theta) + kP_n(w)} + \sum_{T_n} log(1 - \frac{P_m^h(w;\theta)}{P_m^h(w;\theta) + kP_n(w)})$$

$$J^h(\theta) = \frac{\partial}{\partial\theta} L(\theta) = \sum_{T_d} \frac{\partial}{\partial\theta} log \frac{P_m^h(w;\theta)}{P_m^h(w;\theta) + kP_n(w)} + \sum_{T_n} \frac{\partial}{\partial\theta} log(1 - \frac{P_m^h(w;\theta)}{P_m^h(w;\theta) + kP_n(w)})$$

Solution Based on Stochastic Gradient Descent (SGD)

$$J^h(\theta) = \frac{\partial}{\partial\theta} L(\theta) = \sum_{T_d} \frac{\partial}{\partial\theta} log \frac{P_m^h(w;\theta)}{P_m^h(w;\theta) + kP_n(w)} + \sum_{T_n} \frac{\partial}{\partial\theta} log(1 - \frac{P_m^h(w;\theta)}{P_m^h(w;\theta) + kP_n(w)})$$

$$\frac{\partial}{\partial\theta} log \frac{P_m^h(w;\theta)}{P_m^h(w;\theta) + kP_n(w)} = \frac{P_m^h(w;\theta) + kP_n(w)}{P_m^h(w;\theta)} \left\{ \frac{(P_m^h(w;\theta) + kP_n(w)) \frac{\partial}{\partial\theta} P_m^h(w;\theta) - P_m^h(w;\theta) \frac{\partial}{\partial\theta} (P_m^h(w;\theta))}{(P_m^h(w;\theta) + kP_n(w))^2} \right\}$$

$$= \frac{1}{P_m^h(w;\theta)} \left\{ \frac{kP_n(w) \frac{\partial}{\partial\theta} (P_m^h(w;\theta))}{P_m^h(w;\theta) + kP_n(w)} \right\} = \frac{kP_n(w)}{P_m^h(w;\theta) + kP_n(w)} \frac{\partial}{\partial\theta} log(P_m^h(w;\theta))$$

$$\frac{\partial}{\partial\theta} log(1 - \frac{P_m^h(w;\theta)}{P_m^h(w;\theta) + kP_n(w)}) = \frac{\partial}{\partial\theta} log \frac{kP_n(w;\theta)}{P_m^h(w;\theta) + kP_n(w)} = \frac{P_m^h(w;\theta) + kP_n(w)}{kP_n(w;\theta)} \left\{ \frac{-kP_n(w) \frac{\partial}{\partial\theta} (P_m^h(w;\theta))}{(P_m^h(w;\theta) + kP_n(w))^2} \right\}$$

$$= \frac{-\frac{\partial}{\partial\theta} (P_m^h(w;\theta))}{P_m^h(w;\theta) + kP_n(w)} = -\frac{P_m^h(w;\theta)}{P_m^h(w;\theta) + kP_n(w)} \frac{\partial}{\partial\theta} log(P_m^h(w;\theta))$$

$$J^h(\theta) = \sum_{T_d} \frac{kP_n(w)}{P_m^h(w;\theta) + kP_n(w)} \frac{\partial}{\partial\theta} log(P_m^h(w;\theta)) - \sum_{T_n} \frac{P_m^h(w;\theta)}{P_m^h(w;\theta) + kP_n(w)} \frac{\partial}{\partial\theta} log(P_m^h(w;\theta))$$

Solution Based on Stochastic Gradient Descent (SGD)

$$J^h(\theta) = \sum_{T_d} \frac{kP_n(w)}{P_m^h(w;\theta) + kP_n(w)} \frac{\partial}{\partial\theta} log(P_m^h(w;\theta)) - \sum_{T_n} \frac{P_m^h(w;\theta)}{P_m^h(w;\theta) + kP_n(w)} \frac{\partial}{\partial\theta} log(P_m^h(w;\theta))$$

Given a word w observed in context h, we generate k noise samples $x_1, \ldots, x_k$ and update the gradient using:

$$J^h(\theta) = \frac{kP_n(w)}{P_m^h(w;\theta) + kP_n(w)} \frac{\partial}{\partial\theta} log(P_m^h(w;\theta)) - \sum_{i=1}^{k} \frac{P_m^h(x_i;\theta)}{P_m^h(x_i;\theta) + kP_n(x_i)} \frac{\partial}{\partial\theta} log(P_m^h(x_i;\theta))$$

- But, distributions for different context h are not independent, they share parameters, so we can't optimize one $J^h(\theta)$ at a time.
- Instead, Mnih & Teh, 2012, define a global NCE objective by combining the per-context NCE objectives using the empirical context probabilities $P(h)$ as weights:

$$J(\theta) = \sum_h P(h) J^h(\theta).$$

NCE and NPLM: Normalization constants

The last trick solves the problem related to the estimation of the gradient, but at testing time we still need to evaluate $P_m^h(z;\theta)$ that depends on context h, so potentially, there is a different normalization parameter for each distribution.

$$P_m^h(z;\theta) = \bar{P}_m^h(z;\alpha) + log(\mathcal{Z}^h),$$

- One possibility is to use a hash table indexed by the context h to learn a normalization constant $log(\mathcal{Z}^h)$ for each context.
- However, estimating one parameter per context does not scale properly to large datasets.
- Fortunately, thanks to the magic of NCE, Mnih & Teh, 2012, report that fixing the normalizing constants to 1, i.e., $log(\mathcal{Z}^h) = 0$, instead of learning them, did not affect the performance of the resulting models.
- Using a LBM they consider the $score(w_t, h(w_t)) = r_{w_t}^T r'_{w_t}$. So the data distribution is given by $P_m^h(z;\theta) = e^{r_{w_t}^T r'_{w_t}}$.
- In this case the learning complexity of the softmax approximation is $O(k)$ instead of $O(|V|)$, why?, (hint: check the estimation of the gradient).
- Notice that NCE is very effective for speeding up training, but has no impact on testing time, why?.

Results from Mnih & Teh, 2012

- Model: LBM with 100D feature vectors and a 2-word context.
- Dataset: Penn Treebank – news stories from Wall Street Journal.
  - Training set: 930K words.
  - Validation set: 74K words.
  - Test set: 82K words.
  - Vocabulary: 10K words.
- Each parameter update using 100 mini-batches of context/word pairs.
- Models are evaluated based on their test set perplexity.

Results from Mnih & Teh, 2012

Training algorithm	Number of samples	Test PPL	Training time (h)
ML		163.5	21
NCE	1	192.5	1.5
NCE	5	172.6	1.5
NCE	25	163.1	1.5
NCE	100	159.1	1.5

- NCE training is 14 times faster than ML training in this setup.
- The number of samples has little effect on the training time because the cost of computing the predicted representation dominates the cost of the NCE-specific computations.
- In general, using 25 noise samples from the empirical unigram distribution provides good results.

Results from Mnih & Teh, 2012

Unigram vs Uniform noise distributions.

Number of samples	PPL using unigram noise	PPL using uniform noise
1	192.5	291.0
5	172.6	233.7
25	163.1	195.1
100	159.1	173.2

Results from Mnih & Teh, 2012

MSR Sentence Completion Challenge

- Large-scale application: MSR Sentence Completion Challenge
- Task: given a sentence with a missing word, find the correct completion from a list of candidate words.
  - Test set: 1,040 sentences from five Sherlock Holmes novels
  - Training data: 522 19th-century novels from Project Gutenberg (48M words)
- Five candidate completions per sentence.
- Random guessing achieves 20% accuracy.
- Candidate words chosen from relatively infrequent words using a maximum entropy model (so distractors are not too improbable). Human judges pick the best four candidates for each sentence so that all completions are grammatically correct but the correct answer is unambiguous.

Results from Mnih & Teh, 2012

Example from dataset

During two years I have had three _____ and one small job, and that is absolutely all that my profession has brought me.

a) cheers
b) jackets
c) crackers
d) fishes
e) consultations

Results from Mnih & Teh, 2012

- They use a LBM with two extensions:
  - Diagonal context matrices for better scalability w.r.t word representation dimensionality.
  - Separate representation tables for context words and the next word.
- Handling sentence boundaries:
  - Use a special “out-of-sentence” token for words in context positions outside of the sentence containing the word being predicted.
- Word representation dimensionality: 100, 200, or 300.
- Context size: 2-10.
- Training time (48M words, 80K vocabulary): 1-2 days on a single core.
  - Estimated MLE training time: 1-2 months.

Results from Mnih & Teh, 2012

Method	Context size	Latent dim	Test PPL	Percent correct
Chance	0			20.0
3-gram	2		130.8	36.0
4-gram	3		122.1	39.1
5-gram	4		121.5	38.7
6-gram	5		121.7	38.4
LSA	sentence	300		49
RNN	sentence	?	?	45
LBL	2	100	145.5	41.5
LBL	3	100	135.6	45.1
LBL	5	100	129.8	49.3
LBL	10	100	124.0	50.0
LBL	10	200	117.7	52.8
LBL	10	300	116.4	**54.7**
LBL	10×2	100	38.6	44.5

NPLMs and Word Feature Learning

Word feature learning

- As we discussed, the two previous NPLM approaches achieve 2 goals: i) Word vector embedding and ii) Word-context probabilistic modeling (language modeling).
- Actually, the original goal of the previous models was word-context modeling, but as a collateral effect the methods provide also a word vector representation.
- Learning a suitable vector representation for words is indeed a highly relevant goal. This task can be cast as feature learning, where the resulting features can be used (transfer) to feed several relevant NLP applications.

Word2vec

- As described in Mikolov et al., 2013, NPLM methods such as Bengio 2003 are highly inefficient to obtain vector representations. In short, they have too many parameters and are costly to train.
- As a consequence, these methods can hardly scale to large datasets with billions of words and millions of words in the vocabulary.
- If we do not care about language modelling, we can use simpler models to learn word vector representations: word2vec.
- Idea: simpler model can scale to bigger datasets. Algorithmic design should trade model's complexity for efficiency.

Continuous BoW and Skip-gram models, Mikolov et al., 2013

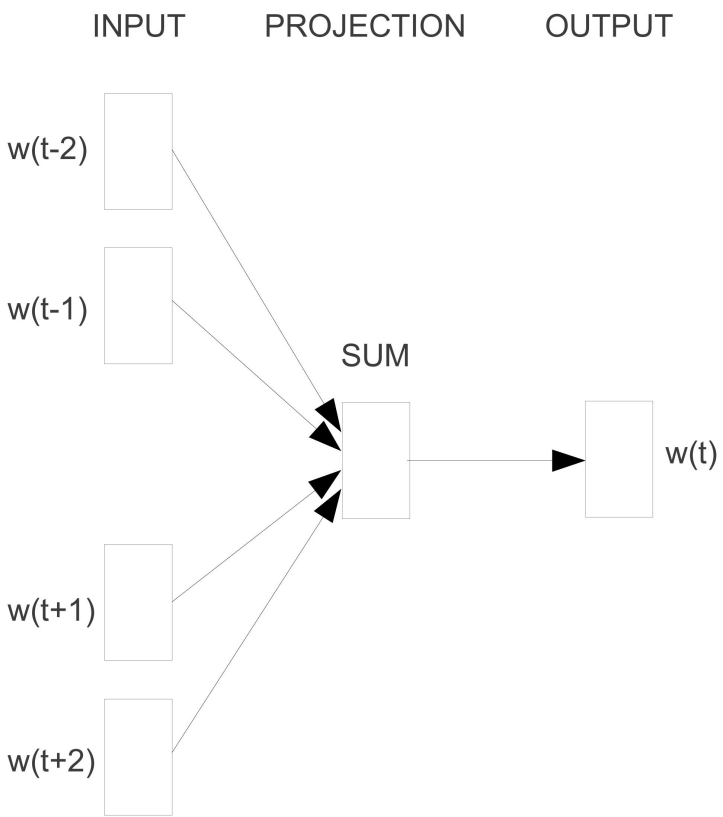
- Mikolov et al., ICLR-2013 and NIPS-2013, present two NPLM models that primarily focus on word embedding.
- These models explore simple network architectures that can be efficiently trained on large datasets.

With respect to Bengio et al. 2003, they propose two main simplifications:

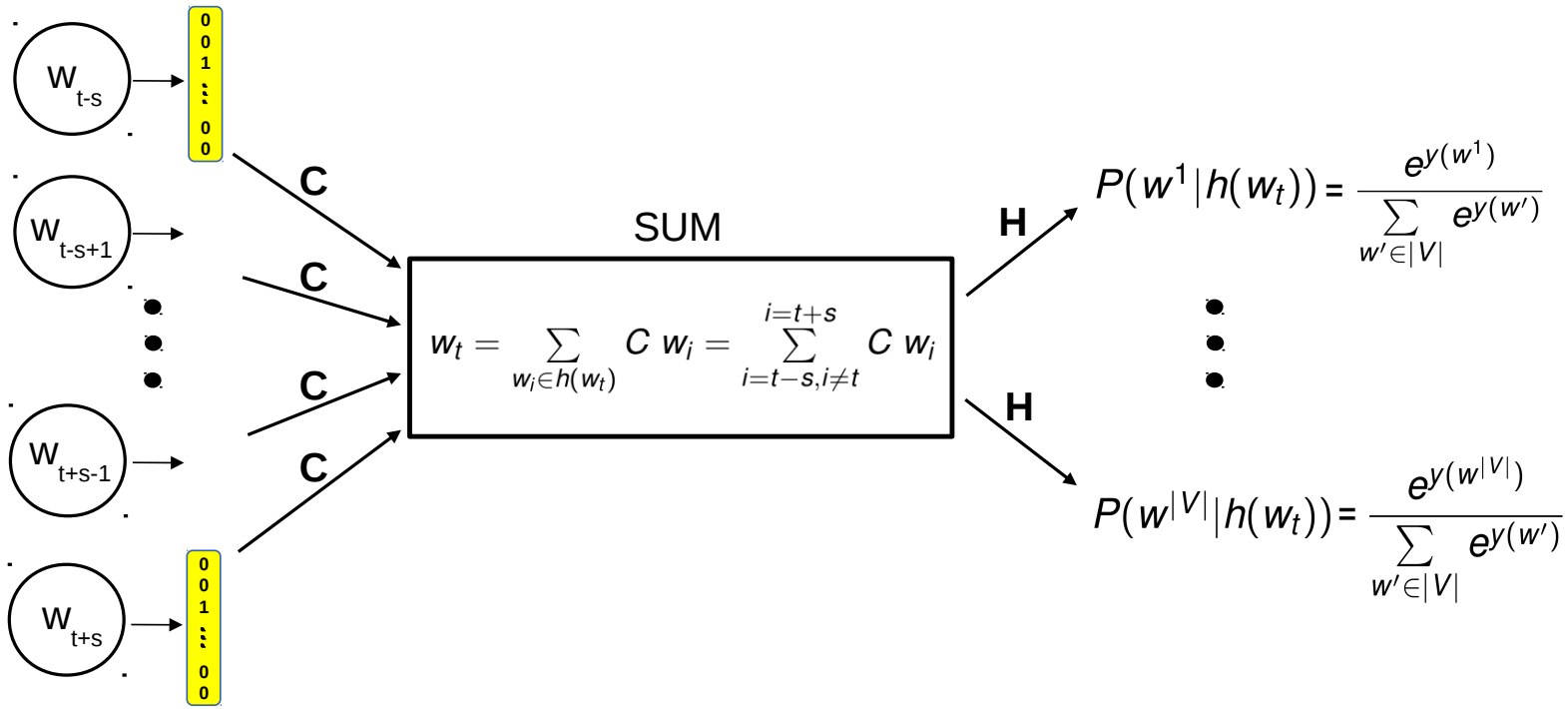
1. Remove the hidden layer and directly connect the low-dim word vector representation to the soft-max output units. To achieve this, they propose two scheme:
   - Continuous Bag-of-Words Model (CBoW).
   - Continuous Skip-gram Model (Skip-gram).
2. Replace the costly soft-max step by simpler schemes, such as, hierarchical soft-max (HSMax) or NCE.

Continuous Bag-of-Words Model (CBoW)

- CBoW sums the projected vector representations of the context words.
- Like BoWs, the resulting vector representation is an average vector, where word position is lost.
- Unlike BoWs, the projection layer produces continuous vector representations, so the name CBoW.
- The CBoW output vector w(t) is directly connected to the soft-max output nodes.



Continuous Bag-of-Words Model (CBoW)

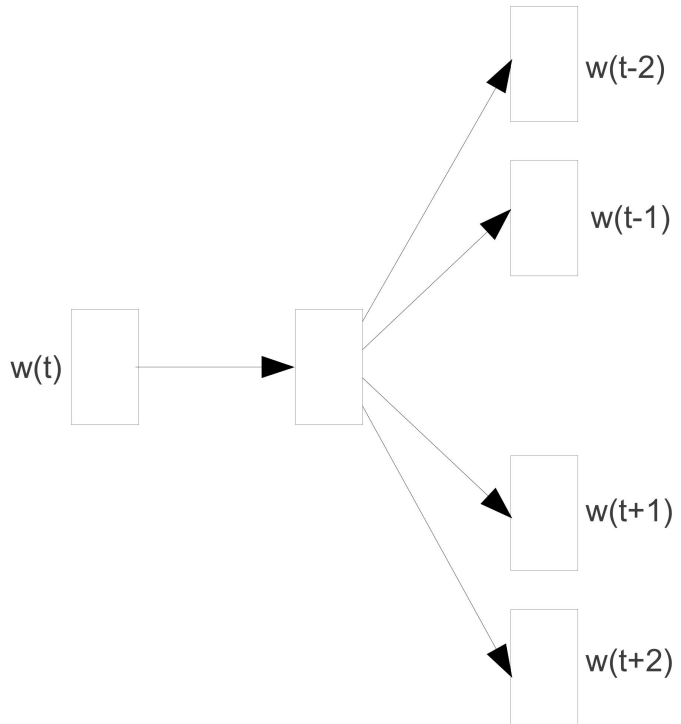


$$\mathbf{C} \in \Re^{|V| x m}, \mathbf{H} \in \Re^{m x |V|}$$

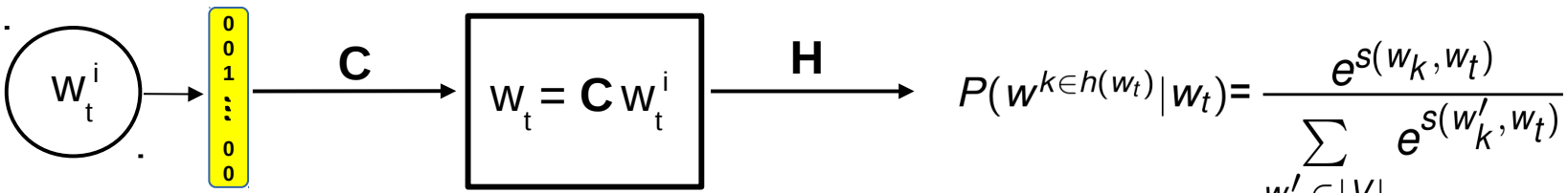
- The goal of the training step is to learn weight matrices **C** and **H**.
- In particular, we are interested on matrix **C** that has on its columns the $|V|$ word embedded vectors.
- Notice that there is not a hidden layer, but a direct bilinear connection **H** between the projection layer and softmax outputs.

Continuous Skip-gram Model

- Given a target word *w* (usually the center word), the Skip-gram model predicts possible context words within a certain range surrounding word *w*.
- More distance words are usually less related to the central one. The Skip-gram model takes account of this fact by sampling less from these words, so they have less relevance in the parameter estimation.



Continuous Skip-gram Model



where $s(w_k, w_t) = w_k^T w_t$. $\mathbf{C} \in \Re^{|V| x m}$, $\mathbf{H} \in \Re^{m x |V|}$

- The goal of the training step is to learn weight matrices **C** and **H**.
- Notice that there is not a hidden layer, but a direct linear connection **H** between the projection layer and softmax outputs.

Table: Number of parameters

Algorithm	Projection layer	Hidden layer	SoftMax Score	Total
Bengio et al. 2003	|V|*m	n*m*h + h*|V|	–	|V|*m + n*m*h + h*|V|
Log-Bilinear Model (Mnih & Hinton, 2007)	|V|*m	m*m	–	|V|*m + m*m
CBoW (Mikolov et al. 2013)	|V|*m	m *|V|	–	|V|*m + m*|V|
Skip-gram (Mikolov et al. 2013)	|V|*m	m *|V|	–	|V|*m + m *|V|

Table: Computational Complexity at Test Time

Algorithm	Projection layer	Hidden layer	SoftMax Score	Total
Bengio et al. 2003	n*m	n*m*h + h*|V|	|V|*m	n*m + n*m*h + |V|*(h+m)
Log-Bilinear Model (Mnih & Hinton, 2007)	n*m	n*m*m [(1)]	m*|V| [(2)]	n*m + n*m*m + m*|V|
CBoW (Mikolov et al. 2013)	n*m	–	m*|V|	n*m + m*|V|
Skip-gram (Mikolov et al. 2013)	m	–	m*n*|V|	m + m*n*|V|

where:

- n : number of context words.
- $|V|$: vocabulary size.
- m : dimensionality of word-context embedding.
- h : number of units in hidden layer.

By using NCE techniques complexity of Soft-Max step is proportional to k+1 instead of |V|, where k is the number of samples from noise distribution.

For simplicity, we omit the bias terms.

[(1)] To get $r'(w_t)$ for context words.

[(2)] To get $r(w_t)$ for each candidate word w_t.

CBoW and Skip-gram models: Learning

- CBoW parameter learning is similar to previous NPLM. For the experiments reported, they use stochastic gradient descent and backpropagation. Learning rate starts at 0.025 and decreases linearly, so that it approaches zero at the end of the last training epoch.
- Skip-gram parameter learning is more complex, we will discuss next this process in detail.
- Optimized code for computing the CBOW and skip-gram models has been published as word2vec project: `https://code.google.com/p/word2vec/`

Skip-gram model and NCE

Skip-gram model learns word embeddings that are useful to predict surrounding words in a sentence.

Idea

- NCE approximately maximizes the log probability of the softmax.
- But, this property is not important for the Skip-gram model, because it focuses only on learning high-quality vector representations.
- So it is possible to simplify the NCE formulation.
- In particular, Mikolov et al. (2013) introduces a variant to the NCE scheme named Negative Sampling (NS).
- The main difference between NCE and NS is that NCE needs both, samples and the numerical probabilities of the noise distribution, while NS needs only the samples.
- As NCE, NS maximizes $P(D = 1|w_t, h(w_t))$ for data pairs $(w_t, h(w_t))$ and $P(D = 0|w_t, h(w_t))$ for "negative" examples $(w_t, h(w_t))$ coming from the noise distribution.
- To obtain negative samples, they use the assumption that randomly selecting a context for a given word is likely to result in an unobserved $(w_t, h(w_t))$ pair.

Skip-gram model and NS

Let:

W : words vocabulary.
$v(w)$: $\in R^d$, vector representation of word $w \in W$.
$h(w)$: set of contextual words for word w.
$h'(w)$: set of randomly sampled, noisy context words for the word w.

Probability of observing word $c \in h(w)$ is given by:

$$P(D = 1|v(w), v(c)) = \frac{1}{1 + e^{-v(w)^T v(c)}}$$

Prob. that $c \notin h(w)$ is given by:

$$P(D = 0|v(w), v(c)) = 1 - P(D = 1|v(w), v(c))$$

Skip-gram model maximizes the following objective function:

$$J(\theta) = \sum_{(w_t, h(w_t)) \in T_d} \sum_{c \in h(w_t)} logP(D = 1|v(w), v(c)) + \sum_{(w_t, h'(w_t)) \in T_n} \sum_{c' \in h'(w_t)} logP(D = 0|v(w), v(c'))$$

- Context words includes R_t words to the left and right sides of the given word w_t, respectively. They use $R_t = 5$. Also, since more distant words are usually less related to the center word than those close to it, they give less weight to the distant words by sampling less from those words in the training examples.
- The set of noisy context words $h(w)$ are randomly sampled from a power of the unigram distribution. Specifically:

$$P(w) = \frac{unigram(w)^{3/4}}{Z}$$

Further implementation issues

- It is useful to sub-sample the frequent words (such as ‘the’, ‘is’, ‘a’, ...) during training. Also it is useful to eliminate very rare words.
- word2vec has two parameters for discarding some of the input words: words appearing less than min-count times are not considered as either words or contexts, frequent words (as defined by the sample parameter) are down-sampled.
- A subsampling scheme that depends on word frequency and work well in practice is given by:

$$P(w_i) = \sqrt{1 - \frac{s}{f(w_i)}}$$

where $f(w_i)$ is the frequency of word w_i and s is a chosen threshold, typically around 10^{-5}. This subsampling aggressively subsamples words whose frequency is greater than threshold s while preserving the ranking of the frequencies.

Results

Goal: Evaluate capacity to identify semantic and syntactic word relationships.

- Test set contains 8869 semantic and 10675 syntactic questions (Semantic-Syntactic dataset).
- Relations are divided in 5 types of semantic relations and 9 types of syntactic relations. Table shows two examples from each type of relations.
- Training set is given by Google News corpus, which contains 6B tokens.
- For training set, they restrict the vocabulary size to 1 million most frequent words.

Type of relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
Currency	Angola	kwanza	Iran	rial
City-in-state	Chicago	Illinois	Stockton	California
Man-Woman	brother	sister	grandson	granddaughter
Adjective to adverb	apparent	apparently	rapid	rapidly
Opposite	possibly	impossibly	ethical	unethical
Comparative	great	greater	tough	tougher
Superlative	easy	easiest	lucky	luckiest
Present Participle	think	thinking	read	reading
Nationality adjective	Switzerland	Swiss	Cambodia	Cambodian
Past tense	walking	walked	swimming	swam
Plural nouns	mouse	mice	dollar	dollars
Plural verbs	work	works	speak	speaks

Results

- For the evaluation they answer questions. Ex. “What is the word that is similar to small in the same sense as biggest is similar to big?”
- They answer each question by performing simple algebraic operations with the vector representation of words.
- Ex. To find a word that is similar to small in the same sense as biggest is similar to big, they compute vector X=vector(“biggest”)-vector(“big”) + vector(“small”).
- Then, they search in the vector space for the word closest to X measured by cosine distance, and use it as the answer to the question (we discard the input question words during this search).
- Question is assumed to be correctly answered only if the closest word to the vector computed using the above method is exactly the same as the correct word in the question
- Synonyms are thus counted as mistakes. This could be improved in the future using some more elaborated language information.

Results

- Table compares results on Semantic-Syntactic dataset for different publicly available algorithms to obtain word vector embedding.
- CBOW and Skip-gram model are trained on subset of the Google News data.
- CBoW training is about 1 day, Skip-gram is about 3 days.
- Note that in the semantic relationships, the skip-gram model substantially outperforms the CBoW model, any guess?

Model	Vector Dimensionality	Training words	Accuracy [%]		
			Semantic	Syntactic	Total
Collobert-Weston NNLM	50	660M	9.3	12.3	11.0
Turian NNLM	50	37M	1.4	2.6	2.1
Turian NNLM	200	37M	1.4	2.2	1.8
Mnih NNLM	50	37M	1.8	9.1	5.8
Mnih NNLM	100	37M	3.3	13.2	8.8
Mikolov RNNLM	80	320M	4.9	18.4	12.7
Mikolov RNNLM	640	320M	8.6	36.5	24.6
Huang NNLM	50	990M	13.3	11.6	12.3
Our NNLM	20	6B	12.9	26.4	20.3
Our NNLM	50	6B	27.9	55.8	43.2
Our NNLM	100	6B	34.2	**64.5**	50.8
CBOW	300	783M	15.5	53.1	36.1
Skip-gram	300	783M	**50.0**	55.9	**53.3**

Results

- Table compares models trained for 3 and 1 epochs on the same data.
- Accuracy is reported on the full Semantic-Syntactic dataset.
- Note that training a model on twice as much data using 1 epoch gives comparable, or better, results than iterating over the same data for 3 epochs and provides additional small speedup.

Model	Vector Dimensionality	Training words	Accuracy [%]			Training time [days]
			Semantic	Syntactic	Total	
3 epoch CBOW	300	783M	15.5	53.1	36.1	1
3 epoch Skip-gram	300	783M	50.0	55.9	53.3	3
1 epoch CBOW	300	783M	13.8	49.9	33.6	0.3
1 epoch CBOW	300	1.6B	16.1	52.6	36.1	0.6
1 epoch CBOW	600	783M	15.4	53.3	36.2	0.7
1 epoch Skip-gram	300	783M	45.6	52.2	49.2	1
1 epoch Skip-gram	300	1.6B	52.2	55.1	53.8	2
1 epoch Skip-gram	600	783M	56.7	54.5	55.5	2.5

Results

- They perform a big test using the 6B word corpus on a distributed framework called DistBelief.
- They use mini-batch asynchronous gradient descent and Adagrad algorithm to adaptively control the learning rate.
- They use 50 to 100 model replicas during the training.
- The number of CPU cores reported is an estimate (cluster is shared with other tasks).
- Note that training of NNLM with 1000-dimensional vectors would take too long to complete.

Model	Vector Dimensionality	Training words	Accuracy [%]			Training time [days x CPU cores]
			Semantic	Syntactic	Total	
NNLM	100	6B	34.2	64.5	50.8	14 x 180
CBOW	1000	6B	57.3	68.9	63.7	2 x 140
Skip-gram	1000	6B	66.1	65.1	65.6	2.5 x 125

Results

Microsoft Research Sentence Completion Challenge

- They train skip-gram using 640D vectors.
- Training data is given by 50M words provided for the challenge.

Architecture	Accuracy [%]
4-gram [32]	39
Average LSA similarity [32]	49
Log-bilinear model [24]	54.8
RNNLMs [19]	55.4
Skip-gram	48.0
Skip-gram + RNNLMs	**58.9**

Results: Additive Compositionality

- Previous examples show that the word2vect embeddings exhibit a linear structure that makes it possible to perform analogical reasoning using vector arithmetics.
- Element-wise vector additions provide Additive Compositionality of semantic concepts. Following table shows some examples:

Czech + currency	Vietnam + capital	German + airlines	Russian + river	French + actress
koruna	Hanoi	airline Lufthansa	Moscow	Juliette Binoche
Check crown	Ho Chi Minh City	carrier Lufthansa	Volga River	Vanessa Paradis
Polish zolty	Viet Nam	flag carrier Lufthansa	upriver	Charlotte Gainsbourg
CTK	Vietnamese	Lufthansa	Russia	Cecile De

Results: Additive Compositionality

- Figure shows 2D PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities.
- Note that during training, the algorithm did not receive any supervised information about what a capital city means.
- However, the model is able to automatically organize concepts and learn implicitly the relationships between them.
- Any intuition of why we obtain this behavior?.