Applied Machine Learning - Lecture Notes Unit #2

Gal Lavee
May 17, 2018

In this unit we will discuss word representations and algorithmic approaches to derive them from data.

1 Applications of Word Representations

- Information Retrieval
  - Similarity between queries and documents
  - Similarity between documents
- Natural Language Processing
  - Named Entity Recogniton
  - Chunking of Documents
  - Part of Speech Recognition
  - Word-sense disambiguation
- Semantic/Syntactic similarity between words
  - Synonyms/Antonyms
  - Paraphrasing

2 Classical Approaches

- One hot encoding
- Hand-designed features
  - Capitalization
  - Parts of Speech

3 Generalization Ability

Consider the following sentences that may appear in our training corpus:

1. The cat is walking in the bedroom.
2. A dog was running in a room.

If we could achieve a similar representation to words that have similar function: (dog,cat) , (the,a) , (bedroom,room), (is,was) ,(running,walking), then our model would be able to generalize to sentences that did not appear in the training corpus, but have similar structure:

1. A cat was running in the bedroom.
2. The dog is walking in a bedroom.

4 Statistical Models of Language

A statistical model of language defines a probability distribution over text. So that a particular string of words:

$$\mathbf{w}_i^j = \langle w_i, w_{i+1}, \ldots, w_{j-1}, w_j \rangle$$

where each w_i takes on a discrete value contained in the 'vocabulary' set V. Let the word sequence $\mathbf{w}_1^T$ represent an entire corpus of text available (of length T). A statistical language model chooses a particular parametrization for the quantity:

$$P(\mathbf{w}_1^T) = \prod_{t=1}^{T} P(w_t \mid \mathbf{w}_1^{t-1})$$

The *unigram model* makes an assumption that the appearance of each token is independent of other tokens, that is $P(w_t \mid \mathbf{w}_1^{t-1}) = P(w_t)$, and further that the position of the word in the sequence has no effect on its probability, i.e $P(w_i = w) = P(w_j = w)$. This means that this mode is parameterized by the number of values each w_t can take on, which is the size of our vocabulary set, denoted $|V|$.

The parameters for each token $v \in V$ can be estimated from data using empirical frequencies:

$$P(w_t = v) = \frac{f(v)}{f(\cdot)}$$

where $f(v)$, denotes the number of times the token v appeared in the training corpus. And $f(\cdot)$ denotes the total words in the training corpus.

The *n-gram model* considers groups of tokens of size n as its basic unit. That is, $P(w_t \mid \mathbf{w}_1^{t-1})$ $= P(w_t \mid \mathbf{w}_{t-n+1}^{t-1})$. The number of parameters for the language model is thus $|V|^n$.

These parameters can also be estimated from data as follows, in a bi-gram (n=2) model the appearance of v_1 followed by v_2 is given probability:

$$P(w_t = v_1, w_{t+1} = v_2) = \frac{f(v_1, v_2)}{f(v_1, \cdot)}$$

where $f(v_1, v_2)$, denotes the number of times the token v_1 appeared followed by v_2 in the training corpus and $f(v_1, \cdot)$ denotes the number of times v_1 was followed by any other token

These local approaches are simple and easy to implement on large amount of data and as a rule work better the more data is available. However, they suffer from several weaknesses:

1. There is no notion of similarity between words
2. Data sparsity becomes a problem for infrequent token strings as n becomes large
3. In many domains the amount of training data is small
4. "long" term (more than 1 or two words) context is ignored

5 Distributed Representation

One way to achieve this type of generalization is to use a continuous representation for words. That is, each word will be represented by a real-valued vector in some continuous space. If we learn a smooth probabillity distibution over this space, then a small change in the input (perhaps swapping out similar words), would lead to a small change in the probability.

In a distributed representation/ word embedding model each word in the vocabulary is associated with a vector in m dimensions, where m is significantly smaller than the size of the vocabulary. This forces all words to have some similarity to each other. These vectors, sometimes called word embeddings are free parameters and are learned by maximizing the likelihood of the data under the model.

6 Distributional Representation

In distributional representation we define a notion of discrete context. Each word in the corpus can appear in any number of contexts. Contexts may be documents in a document corpus, nearby words, or semantic tags (e.g. wikipedia categories).

The words in the vocabulary can then be arranged in a matrix who's rows are the word representations, and who's columns are the contexts. The entries in the matrix are the counts (weighted or normalized) of how many times a word in our corpus appeared in a particular context.

We can use the rows in the matrix as our word representations and achieve a model where words that appear in similar contexts are similar. However, when we have a large number of contexts this may result in a word representation that is large and possibly sparse. Thus, further dimensionality reduction is performed on these distributions to make them compact and continuous.

- Latent Semantic Indexing(LSI) - is a method that performs linear dimensionality reduction on the matrix using document contexts and a counting method called TFIDF(term frequency - inverse document frequency)
- Latent Dirichlet Allocation (LDA)- is a method that also uses document contexts and straightforward counting, but yields an intermediary (small) set of parameters called "topics", which can be interpreted as distributions over words in the corpus. Words, in turn, are described as a (sparse) distribution over these latent "topics".
- Neural Network Language Model (NNLM)- contexts are words in the 'neighborhood' of the target word. Design decision determine whether the neighborhood consists of a window to the right or left and what the size of the window is.

Thus, these representation are both distributional (similar contexts yield similar representations) and distributed (words are represented using continuous vectors)

7 Maximizing The Likelihood

Each of the models above defines a probability distribution over sequences of words give the parameters. We often have a large corpus of text, which can be divided into N sequences (either naturally or arbitrarily). An additional assumption of independence allows us to define a function of the parameters known as the likelihood.

$$L(\theta) = \prod_{i=1}^{N} P(\mathbf{w}^{(i)} \mid \theta)$$

where $\mathbf{w}^{(i)}$ denotes the i-th sequence of words in our training corpus.

Of course, the exact value of the likelihood depends on the words in the data, how the model is defined and the choice of the parameters. However, since the model and the data are fixed at the time of learning, for our purposes we can consider the likelihood as function of θ the model parameters.

The principle of maximum likelihood, as is implied, says that the value of θ that maximizes the above expression is useful for the application of the model to data.

This is very related to loss minimization, which we can see by taking first taking the logarithm to obtain the log-likelihood:

$$\ell(\theta) = \sum_{i=1}^{N} \log P(\mathbf{w}^{(i)} \mid \theta)$$

negating this quantity gives the negative log likelihood and turns our maximization into a minimization:

$$\mathbf{NLL}(\theta) = \sum_{i=1}^{N} -\log P(\mathbf{w}^{(i)} \mid \theta)$$

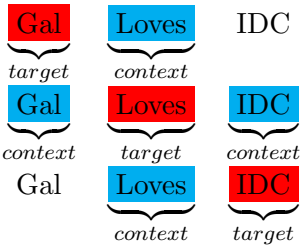
So we can treat $-\log P(\mathbf{w}^{(i)} \mid \theta)$ as the additive loss component for each example i

8 Neural Net Language model

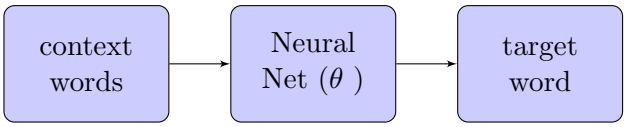
In the neural net language model, we model each word using the context words in its neighborhood. Consider the sentence:

Gal Loves IDC

can yield three separate targets and contexts as follows (using a symmetric neighborhood of size 2):



The specific probability of the target word w_t given the context words w_{c_1}, w_{c_2} is given by a neural network as follows:



Further the probability of an entire string of words is the product of each of the input context pairs.

$$
\begin{aligned}
P(\text{“Gal Loves IDC”}) = & \\
& P(w_t = \text{“Gal”} \mid w_{c_1} = \text{“”}, w_{c_2} = \text{“Loves”}) \cdot \\
& P(w_t = \text{“Loves”} \mid w_{c_1} = \text{“Gal”}, w_{c_2} = \text{“IDC”}) \cdot \\
& P(w_t = \text{“IDC”} \mid w_{c_1} = \text{“Loves”}, w_{c_2} = \text{“”})
\end{aligned}
$$

When we take the logarithm of this expression, we get a sum of the (log of the) terms, which means that we can optimize each target/context separately.

Thus a way to learn the parameters of the neural net is to consider all the target/context pairs in the data, time computing a gradient for each and summing the gradients to obtain the gradient direction that optimizes the likelihood. Of course, a faster variant is to use stochastic gradient descent or mini-batch gradient descent.

9 The Softmax Activation Function

In order to obtain a probability distribution over the words in the vocabulary, the NNLM needs to apply a so-called softmax function, which takes d real numbers as $x_1 \dots x_d$ input and outputs and outputs d real numbers on the d-simplex $y_1 \dots y_d$. That is for all i, $y_i \geq 0$ and $\sum y_i = 1$. Each y_i is given by the following expression:

$$y_i = \frac{\exp(a_i \cdot x_i)}{\sum_j \exp(a_j \cdot x_j)}$$

where a_i are the parameters our neural network. The difficulty with this activation function is that each y_i (and its gradient w.r.t. the parameters) depends on *all* d parameters. In the case where $d = |V|$, computing this sum can be quite prohibitive, further each example requires updating all parameters

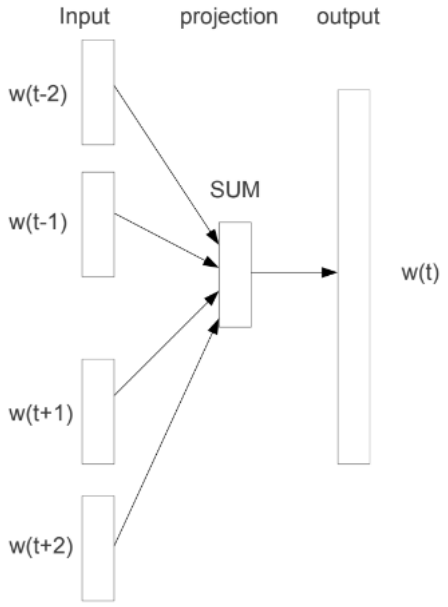
10 word2vec

word2vec is a name given coollectively to the word embedding approach described in two papers by Mikolov et al. [1, 2]. It is collectively used to describe a number of ideas that we will discuss below.

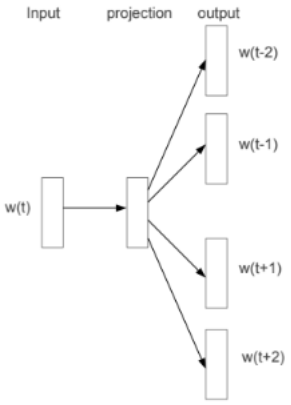
10.1 Continuous Bag Of Words vs Skip-Gram Model

In the Continuous Bag of Words (CBOW) we try to preidct the target word from the context, similar to the NNLM model.

In the skip-gram model we flip the equation and try to predict the context words from the target.



Continuous bag of words



Skip-Gram

10.2 Context and Target Parameters

word2vec uses a double representation for each word, once as a target and once as a context. That is, for every word in our vocabulary, we keep track of two d-dimensional vectors. We will use the notation w_j to denote the *value* of the j-th word in the vocabulary (e.g. 'Gal'), and $\mathbf{u}_j \in \mathbb{R}^d$ to denote the target word representation of this word. Similarly, and $\mathbf{v}_j \in \mathbb{R}^d$ will denote the context representation.

10.3 Doing away with hidden layers (the softmax loss)

CBOW:

$$\log P(\mathbf{w}^{(i)} \mid \theta) = \sum_{(t,c)\in\mathbf{w}^{(i)}} \log P(w_t \mid \mathbf{w}_c)$$

where $(t, c) \in \mathbf{w}^{(i)}$ denote all the target/context pairs in the sequence. Further $\mathbf{w}_c = w_1, w_2, \ldots, w_k$ is the set of context words. Defining $\bar{\mathbf{v}}_c = \frac{1}{k}\sum_{w_c\in\mathbf{w}_c} \mathbf{v}_c$, that is the average of all context word representations.

The CBOW model then defines the summed quantity using the softmax as follows:

$$\log P(w_t \mid \mathbf{w}_c) = \log\left(\frac{\exp\left(\bar{\mathbf{v}}_c^\top \mathbf{u}_t\right)}{\sum_{j=1}^{|V|}\exp\left(\bar{\mathbf{v}}_c^\top \mathbf{u}_j\right)}\right) = \bar{\mathbf{v}}_c^\top \mathbf{u}_t - \log\left(\sum_{j=1}^{|V|}\exp\left(\bar{\mathbf{v}}_c^\top u_j\right)\right)$$

Skip-Gram: The Skip-Gram model uses the assumption that each context word is predicted independently of the others and thus:

$$\log P(\mathbf{w}^{(i)} \mid \theta) = \sum_{(t,c)\in\mathbf{w}^{(i)}} \log P(\mathbf{w}_c|w_t) = \sum_{(t,c)\in\mathbf{w}^{(i)}} \sum_{w_c\in\mathbf{w}_c} \log P(w_c|w_t)$$

and we apply the softmax to evaluate the terms in the sum as follows:

$$\log P(w_c|w_t) = \log\left(\frac{\exp\left(\mathbf{v}_c^\top \mathbf{u}_t\right)}{\sum_{j=1}^{|V|}\exp\left(\mathbf{v}_j^\top \mathbf{u}_t\right)}\right) = \mathbf{v}_c^\top \mathbf{u}_t - \log\left(\sum_{j=1}^{|V|}\exp\left(\mathbf{v}_j^\top \mathbf{u}_t\right)\right)$$

In either case, there are no longer any hidden layers, the only parameters are word vectors which interact via inner product between appropriate context and target vectors.

10.4 The negative sampling loss

As defined above the (log) probabillity of every target/context pair depends on (target) parameters for all words in the vocabulary. This leads to undesirable computational properties (discussed later). An alternative is the Negative Sampling loss where we model the probabillity of a word to appear given the parameters as the product of *binary* variables representing all possible words as follows:

$$P(w = w_t \mid \theta) = P(w_t|\theta) \cdot \prod_{w_{t'}\neq w_t} 1 - P(w_{t'}|\theta)$$

taking the log this becomes:

$$\log P(w = w_t \mid \theta) = \log P(w_t|\theta) + \sum_{w_{t'}\neq w_t} \log\left(1 - P(w_{t'}|\theta)\right)$$

However, now each $P(w|\theta) \in [0, 1]$ denotes the probability of the word w appearing. Translating this quantity into the word2vec parametrization:

CBOW:

$$\log P(w_t|\theta) = \log P(w_t|\mathbf{w}_c) = \log\sigma\left(\bar{\mathbf{v}}_c^\top \mathbf{u}_t\right)$$

skip-gram:

$$\log P(w_t|\theta) = \log P(\mathbf{w}_c|w_t) = \sum_{w_c\in\mathbf{w}_c} \log\sigma\left(\mathbf{v}_c^\top \mathbf{u}_t\right)$$

where $\sigma(z) = \frac{1}{1+\exp(-z)}$, denotes the sigmoid function.
Now each term in our sum depends on only a few of our model parameters. Unfortunately, the sum for *non-target* words is large and hard to compute.

The solution is to approximate the sum with sampling. That is, we sample K words from some noise distribution, $P_n(w)$. A typical choice is the unigram distribution. Using this idea called *negative sampling*, because we are sampling unseen or "negative" data, for each target and context we sample K negative words, $w_{n_1}, \ldots, w_{n_K}$, and our computation becomes:

CBOW:

$$\log P(\mathbf{w}^{(i)} \mid \theta) = \sum_{(t,c)\in\mathbf{w}^{(i)}} \log P(w_t \mid \mathbf{w}_c) = \sum_{(t,c)\in\mathbf{w}^{(i)}} \left[\log \sigma\left(\bar{\mathbf{v}}_c^\top \mathbf{u}_t\right) + \sum_{j=1}^{K} \log\left(1 - \sigma\left(\bar{\mathbf{v}}_c^\top \mathbf{u}_{n_j}\right)\right) \right]$$

Skip-gram:

$$\log P(\mathbf{w}^{(i)} \mid \theta) = \sum_{(t,c)\in\mathbf{w}^{(i)}} \log P(\mathbf{w}_c \mid w_t) = \sum_{(t,c)\in\mathbf{w}^{(i)}} \sum_{w_c\in\mathbf{w}_c} \left[\log \sigma\left(\mathbf{v}_c^\top \mathbf{u}_t\right) + \sum_{j=1}^{K} \log\left(1 - \sigma\left(\mathbf{v}_j^\top \mathbf{u}_t\right)\right) \right]$$

11 Learning The Parameters from data

We seek a configuration of the parameter set θ that will maximize the (log) likelihood of the data under our model. More formally, letting V denote the set of words in our corpus and d denote the dimensionality of representation, the parameters of our model are given by

$$\theta = \left\{ \{\mathbf{v}_j\}_{j=1}^{|V|}, \{\mathbf{u}_j\}_{j=1}^{|V|} \right\}$$

letting $\mathcal{D} = \left\{\mathbf{w}^{(i)}\right\}_{i=1}^{N}$ denote the dataset of N word sequences our log- likelihood (in the case of the skip-gram negative sampling model is given by):

$$\ell(\theta) = \sum_{\mathbf{w}^{(i)}\in\mathcal{D}} \sum_{(t,c)\in\mathbf{w}^{(i)}} \sum_{w_c\in\mathbf{w}_c} \left[\log \sigma\left(\mathbf{v}_c^\top \mathbf{u}_t\right) + \sum_{j=1}^{K} \log\left(1 - \sigma\left(\mathbf{v}_{n_j}^\top \mathbf{u}_t\right)\right) \right]$$

with a slight tweak of our thinking we can consider our dataset to be a dataset of target/context pairs instead of documents, leading to the simplified expression:

$$\ell(\theta) = \sum_{(t,c)\in\mathcal{D}} \sum_{w_c\in\mathbf{w}_c} \left[\log \sigma\left(\mathbf{v}_c^\top \mathbf{u}_t\right) + \sum_{j=1}^{K} \log\left(1 - \sigma\left(\mathbf{v}_{n_j}^\top \mathbf{u}_t\right)\right) \right]$$

our generic recipe for Gradient Descent is given by:

$$\theta_{new} = \theta_{old} - \eta \cdot \nabla_\theta \ell(\theta_{old})$$

However, this will minimize a quantity we want to maximize, so instead lets apply gradient ascent:

$$\theta_{new} = \theta_{old} + \eta \cdot \nabla_\theta \ell(\theta_{old})$$

so in order to apply this recipe we will have to determine the value of the gradient for each of the parameters. Using the fact that the gradient operator is linear we consider the gradient for a particular choice of single target/context word pair (w_t, w_c)

$$\underbrace{\ell(\theta)}_{\textbf{w.r.t. } (w_t,w_c)} = f(w_t, w_c) = \log \sigma\left(\mathbf{v}_c^\top \mathbf{u}_t\right) + \sum_{j=1}^{K} \log\left(1 - \sigma\left(\mathbf{v}_{n_j}^\top \mathbf{u}_t\right)\right)$$

first let us consider the gradient of this expression with respect to the target representation of the target word, $\mathbf{u}_t$,using the fact that $\frac{d}{dx}\log\sigma(x) = \frac{1}{\sigma(x)}\frac{d}{dx}\sigma(x) = \frac{1}{\sigma(x)}\sigma(x)(1-\sigma(x)) = (1-\sigma(x))$

$$\nabla_{\mathbf{u}_t} f(w_t, w_c) = \left(1 - \sigma\left(v_c^\top \mathbf{u}_t\right)\right) \mathbf{v}_c - \sum_{j=1}^{K} \left(\sigma\left(\mathbf{v}_{n_j}^\top \mathbf{u}_t\right)\right) \mathbf{v}_{n_j}$$

now let us consider the gradient with respect to the context representation of the context word, v_c

$$\nabla_{\mathbf{v}_c} f(w_t, w_c) = \left(1 - \sigma\left(\mathbf{v}_c^\top \mathbf{u}_t\right)\right) \mathbf{u}_t$$

Finally, let us consider the gradient for the context representation of one of the sampled negative words, denoted $\mathbf{v}_{n_j}$:

$$\nabla_{\mathbf{v}_{n_j}} f(w_t, w_c) = -\left(\sigma\left(\mathbf{v}_{n_j}^\top \mathbf{u}_t\right)\right) \mathbf{u}_t$$

what happens when the target word is sampled as a negative?

Since each context has multiple words our gradients will be the sum of the gradients for the relevant contexts:

$$\nabla_\theta \sum_{w_c \in \mathbf{w}_c} f(w_t, w_c) = \sum_{w_c \in \mathbf{w}_c} \nabla_\theta f(w_t, w_c)$$

If we were using stochastic gradient descent (ascent) we would be done, simply select a target and context from the dataset and use the updates. However, a good thing to do is in fact called *mini-batch* gradient descent. That is rather than compute the gradient on the entire dataset (batch) or a single example we sample a subset of target/context pairs, denoted $\mathcal{D}_{mb}$ and sum their gradients before doing the updates.

The pseudo code for mini-batch gradient descent (ascent) for the skip-gram model with negative sampling loss is given in Algorithm 1

11.1 Additional Considerations for implementation

1. normalizing vectors - we are only interested in the direction of the embedding vectors not their magnitude. Unfortunately, there is nothing in the algorithm that prohibits growth in magnitude. As such, a simple postprocessing normalizing the vectors to unit norm after the gradient update is helpful to keep the sizes of the vectors in check.

2. normalizing gradients - as with vectors, we are mostly interested in the direction of the gradients (the size of the step is determined by the hyper-parameter η). Because the gradients grow as a function of the mini-batch size, it is a good idea to normalize the gradient magnitude by dividing by `MINIBATCHSIZE`

3. annealing the learning rate - in order to achieve convergence in stochastic (and mini-batch) gradient descent one technique often applies is decreasing the learning rate by a fixed proportion (e.g. $\frac{1}{2}$) after some number of iterations have elapsed. The number of these iterations can be specified as an algorithm hyperparameter.

4. context window size - Symmetric context windows are usually applied in word2vec where the number of words on each side of the target word, C, is specified as a hyperparameter. A variant that adds some robustness to this design decision, is to use hyperparameter C as a maximum window size. That is, when sampling a target word from the datasets, sample a number uniformly over the range $1 - C$, and use that number as the context window size.

Algorithm 1 Gradient Ascent for learning skip-gram model with negative sampling loss

Input:

$\mathcal{D}$ -Training data of target/context pairs $(w_t, \mathbf{w}_c)$ pairs

η - the learning rate

$P_n(w)$ - the noise distribution

V - the vocabulary set

`NUMITER` - number of iterations to run

`MINIBATCHSIZE` - number of examples in each mini-batch

Output: setting of $\theta = \left\{ \{\mathbf{v}_j\}_{j=1}^{|V|}, \{\mathbf{u}_j\}_{j=1}^{|V|} \right\}$ that maximizes $\ell(\theta)$

initialize params θ randomly

for `NUMITER` iterations **do**

\\Compute mini-batch gradients

for $j = 1, \ldots, |V|$ **do**

$\mathbf{g}_j^{(T)} \leftarrow \mathbf{0}$

$\mathbf{g}_j^{(C)} \leftarrow \mathbf{0}$

end for

for $i = 1, \ldots,$ `MINIBATCHSIZE` **do**

Sample a word context pair $(w_t, \mathbf{w}_c)$ from dataset

for $w_c \in \mathbf{w}_c$ **do**

sample K "negative words", $w_{n_1}, \ldots, w_{n_K}$ from noise distribution $P_n(w)$

$\mathbf{g}_t^{(T)} \leftarrow \mathbf{g}_t^{(T)} + \left(1 - \sigma\left(\mathbf{v}_c^\top \mathbf{u}_t\right)\right) \mathbf{v}_c$

$\mathbf{g}_c^{(C)} \leftarrow \mathbf{g}_c^{(C)} + \left(1 - \sigma\left(\mathbf{v}_c^\top \mathbf{u}_t\right)\right) \mathbf{u}_t$

for $k = 1, \ldots, K$ **do**

$\mathbf{g}_t^{(T)} \leftarrow \mathbf{g}_t^{(T)} - \sigma\left(\mathbf{v}_{n_k}^\top \mathbf{u}_t\right) \mathbf{v}_{n_k}$

$\mathbf{g}_{n_k}^{(C)} \leftarrow \mathbf{g}_{n_k}^{(C)} - \sigma\left(\mathbf{v}_{n_k}^\top \mathbf{u}_t\right) \mathbf{u}_t$

end for

end for

end for

\\Perform parameter updates

for $j = 1, \ldots, |V|$ **do**

$\mathbf{u}_j \leftarrow \mathbf{u}_j + \eta \cdot \mathbf{g}_j^{(T)}$

$\mathbf{v}_j \leftarrow \mathbf{v}_j + \eta \cdot \mathbf{g}_j^{(C)}$

end for

end for

5. Selecting the noise distribution - Instead of using the Unigram distribution, $U(w)$ which gives very little probability to rare words, the word2vec authors claim that using $P_n(w) \propto U(w)^{3/4}$ works even better as it gives slightly more probability to the tail words in the vocabulary

References

[1] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[2] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.