How exactly does word2vec work?

David Meyer

dmm@{1-4-5.net,uoregon.edu,brocade.com,...}

July 31, 2016

1 Introduction

The word2vec model [4] and its applications have recently attracted a great deal of attention from the machine learning community. These *dense vector representations* of words learned by word2vec have remarkably been shown to carry semantic meanings and are useful in a wide range of use cases ranging from natural language processing to network flow data analysis.

Perhaps the most amazing property of these word embeddings is that somehow these vector encodings effectively capture the semantic meanings of the words. The question one might ask is how or why? The answer is that because the vectors adhere surprisingly well to our intuition. For instance, words that we know to be synonyms tend to have similar vectors in terms of cosine similarity and antonyms tend to have dissimilar vectors. Even more surprisingly, word vectors tend to obey the laws of analogy. For example, consider the analogy "Woman is to queen as man is to king". It turns out that

$$v_{\text{queen}} - v_{\text{woman}} + v_{\text{man}} \approx v_{\text{king}}$$

where $v_{\text{queen}}, v_{\text{woman}}, v_{\text{man}}$ and v_{king} are the word vectors for queen, woman, man, and king respectively. These observations strongly suggest that word vectors encode valuable semantic information about the words that they represent.

Note that there are two main word2vec models: Continuous Bag of Words (CBOW) and Skip-Gram. In the CBOW model, we predict a word given a context (a context can be something like a sentence). Skip-Gram is the opposite: predict the context given an input word. Each of these models is examined below.

This document contains my notes on the word2vec. NB: there are probably lots of mistakes in this...

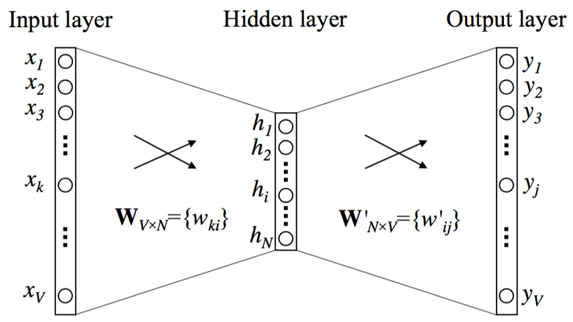

Figure 1: Simple CBOW Model

2 Continuous Bag-of-Words Model

The simplest version of the continuous bag-of-word model (CBOW) is a ”single context word” version [3]. This is shown in Figure 1. Here we assume that there is only one word considered per context, which means the model will predict one target word given one context word (which is similar to a bi-gram language model).

In the scenario depicted in Figure 1, V is the vocabulary size and the hyper-parameter N is the hidden layer size. The input vector $\mathbf{x} = \{x_1, x_2, \dots, x_V\}$ is one-hot encoded, that is, that some $x_k = 1$ and all other $x_{k'} = 0$ for $k \neq k'$.

The weights between between the input layer and the hidden layer can be represented by the $V \times N$ matrix $\mathbf{W}$. Each row of $\mathbf{W}$ is the N-dimensional vector representation $\mathbf{v}_w$ of the associated word w in the input layer. Give a context (here a single word), and assuming again $x_k = 1$ and $x_{k'} = 0$ for $k \neq k'$ (one-hot encoding), then

$$\mathbf{h} = \mathbf{x}^{\mathrm{T}}\mathbf{W} = \mathbf{W}_{(k,.)} := \mathbf{v}_{w_I} \tag{1}$$

This essentially copies the k-th row of $\mathbf{W}$ to $\mathbf{h}$ (this is due to the one-hot encoding of $\mathbf{x}$). Here $\mathbf{v}_{w_I}$ is the vector representation of the input word w_I. Note that this implies that the link (aka activation) function of the hidden units is linear ($g(x) = x$). Recall that

$$\mathbf{W}_{V\times N} = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1\mathrm{N}} \\ w_{21} & w_{22} & \dots & w_{2\mathrm{N}} \\ \vdots & \vdots & \ddots & \vdots \\ w_{\mathrm{V}1} & w_{\mathrm{V}2} & \dots & w_{\mathrm{VN}} \end{bmatrix} \tag{2}$$

and

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{\mathrm{V}} \end{bmatrix} \tag{3}$$

so that

$$\begin{aligned}
\mathbf{h} = \mathbf{x}^{\mathrm{T}}\mathbf{W} &= \begin{bmatrix} x_1 \; x_2 \; \dots \; x_{\mathrm{k}} \; \dots \; x_{\mathrm{V}} \end{bmatrix} \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1\mathrm{N}} \\ w_{21} & w_{22} & \dots & w_{2\mathrm{N}} \\ \vdots & \vdots & \ddots & \vdots \\ w_{\mathrm{k}1} & w_{\mathrm{k}2} & \dots & w_{\mathrm{kN}} \\ \vdots & \vdots & \ddots & \vdots \\ w_{\mathrm{V}1} & w_{\mathrm{V}2} & \dots & w_{\mathrm{VN}} \end{bmatrix} && (4) \\
&= \begin{bmatrix} x_k w_{k1} \; x_k w_{\mathrm{k}2} \; \dots \; x_k w_{\mathrm{kN}} \end{bmatrix} \qquad \# \text{ for } x_k = 1, x_{k'} = 0 \text{ for } k \neq k' && (5) \\
&= \begin{bmatrix} w_{k1} \; w_{\mathrm{k}2} \; \dots \; w_{\mathrm{kN}} \end{bmatrix} \qquad \# \; k\text{-th row of } \mathbf{W} && (6) \\
&= \mathbf{W}_{(k,.)} && (7) \\
&:= \mathbf{v}_{\mathrm{WI}} && (8)
\end{aligned}$$

From the hidden layer to the output layer there is a different weight matrix $\mathbf{W}' = \{w'_{i,j}\}$ which is a $N \times V$ matrix. Using this we can compute a score for each word in the vocabulary:

$$u_j = {\mathbf{v}'_{w_j}}^{\mathrm{T}} \cdot \mathbf{h} \tag{9}$$

where $\mathbf{v}'_{w_j}$ is the j-th column of the matrix $\mathbf{W}'$. Note that the score u_j is a measure of the match between the context and the next word [1] and is computed by taking the dot product between the predicted representation ($\mathbf{v}'_{w_j}$) and the representation of the candidate target word ($\mathbf{h} = \mathbf{v}_{wI}$).

Now you can use a softmax (log linear) classification model to obtain the posterior distribution of words (turns out to be multinomial distribution):

$$p(w_j|w_I) = y_j = \frac{exp(u_j)}{\sum\limits_{j'=1}^{V} exp(u_{j'})} \tag{10}$$

[1]Though almost all statistical language models predict the next word, it is also possible to model the distribution of the word preceding the context or surrounded by the context.

where y_j is the output of the j-th unit of the output layer. Substituting Equations 1 and 9 into Equation 10 we get

$$p(w_j|w_I) = \frac{exp\big(\mathbf{v'}_{wo}{}^{\mathrm{T}}\mathbf{v}_{w_I}\big)}{\sum\limits_{j'=1}^{V} exp\big(\mathbf{v'}_{w'_j}{}^{\mathrm{T}}\mathbf{v}_{wI}\big)} \tag{11}$$

Notes

- Both the input vector $\mathbf{x}$ and the output $\mathbf{y}$ are one-hot encoded
- v_w and v'_w are two representations of the input word w
- v_w comes from the rows of $\mathbf{W}$
- v'_w comes from the columns of $\mathbf{W}'$
- v_w is usually called the *input vector*
- v'_w is usually called the *output vector*

2.1 Updating Weights: hidden layer to output layer

The training objective (for one training sample) is to maximize Equation 11, the conditional probability of observing the actual output word w_O (denote its index in the output layer as j^*) given the input context word w_I (and with regard to the weights). That is,

$$\begin{align}
\max\ p(w_{\mathrm{O}}|w_{\mathrm{I}}) &= \max\ y_{j^*} \tag{12}\\
&= \max \log y_{j^*} \tag{13}\\
&= u_{j^*} - \log \sum_{j'=1}^{V} exp(u_{j'}) := -E \tag{14}
\end{align}$$

where $E = -\log p(w_{\mathrm{O}}|w_{\mathrm{I}})$ is our loss function (which we want to minimize), and j^* is the index of the actual output word (in the output layer). Note (again) that this loss function can be understood as a special case of the cross-entropy measurement between two probabilistic distributions.

The next step is to derive the update equation of the weights between hidden and output layers. Take the derivative of E with respect to j-th unit's net input u_j, we obtain

$$\frac{\partial E}{\partial u_j} = y_j - t_j := e_j \tag{15}$$

where $t_j = \mathbb{1}(j = j^*)$, the indicator function, i.e., t_j will be 1 when the j-th unit is the actual output word, otherwise $t_j = 0$. Interestingly this derivative is simply the prediction error e_j of the output layer.

The next step is to take the derivative on w'_{ij} to obtain the gradient on the hidden $\rightarrow$ output weights which (by the chain rule) is

$$\frac{\partial E}{\partial w'_{ij}} = \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial w'_{ij}} = e_j \cdot h_i \tag{16}$$

since (sketch of the proof)

$$\begin{aligned}
\frac{\partial E}{\partial u_j} &= \frac{\partial \left(u_{j^*} - \log \sum_{j'=1}^{V} exp(u_{j'}) \right)}{\partial u_{j^*}} && (17) \\
&= \frac{\partial u_{j^*}}{\partial u_j} - \frac{\partial \left(\log \sum_{j'=1}^{V} exp(u_{j'}) \right)}{\partial u_j} && (18) \\
&= t_j - \frac{exp(u_j)}{\sum_{j'=1}^{V} exp(u_{j'})} && (19) \\
&= t_j - y_j \qquad\qquad \text{\# by Equations 10 and 15} && (20)
\end{aligned}$$

Now, using stochastic gradient descent, we obtain the weight updating equation for hidden $\rightarrow$ output weights:

$$w'_{ij}{}^{(\text{new})} = w'_{ij}{}^{(\text{old})} - \eta \cdot e_j \cdot h_i \tag{21}$$

and/or

$$\mathbf{v}'_{w_j}{}^{(\text{new})} = \mathbf{v}'_{w_j}{}^{(\text{old})} - \eta \cdot e_j \cdot \mathbf{h} \tag{22}$$

where $\eta > 0$ is the learning rate (standard SGD), $e_j = y_j - t_j$ (Equation 15), h_i is the i-th unit in the hidden layer, and $\mathbf{v}'_{w_j}$ is the output vector for word w_j.

2.2 Updating Weights: Input to hidden layers

Now that we have the update equations for $\mathbf{W}'$, we an look at $\mathbf{W}$. Here we take the derivative for E on the output of the hidden layer:

$$\begin{aligned} \frac{\partial E}{\partial h_i} &= \sum_{j=1}^{V} \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial h_i} && (23) \\ &= \sum_{j=1}^{V} e_j \cdot w'_{ij} && (24) \\ &:= \mathrm{EH}_i && (25) \end{aligned}$$

where h_i is the output of the i-th unit in the hidden layer, u_j is defined in Equation 9 (the input of the j-th unit to the output layer), and $e_j = y_j - t_j$ is the prediction error of the j-th word in the output layer. EH is an N-dimensional vector is the sum of the output vectors of all words in the vocabulary, weighted by their prediction error e_j.

The next job is to take the derivative of E with respect to $\mathbf{W}$. So first recall that the hidden layer performs a linear computation on the values from the input layer, specifically

$$h_i = \sum_{k=1}^{V} x_k \cdot w_{ki} \qquad \text{\# see Equation 1} \tag{26}$$

Now we can use the chain rule to get the derivative of E with respect to $\mathbf{W}$, as follows (chain rule again):

$$\begin{aligned} \frac{\partial E}{\partial w_{ki}} &= \frac{\partial E}{\partial h_i} \cdot \frac{\partial h_i}{\partial w_{ki}} && (27) \\ &= \mathrm{EH}_i \cdot x_k && (28) \end{aligned}$$

and so (vector form)

$$\frac{\partial E}{\partial \mathbf{W}} = \mathrm{EH} \cdot \mathbf{x} \tag{29}$$

From this we obain an $V \times N$ matrix. Notice that since only one component of $\mathbf{x}$ is non-zero, only one row of $\frac{\partial E}{\partial \mathbf{W}}$ is non-zero, and the value of that row is EH (an $1 \times N$ dimensional vector).

Now we can write the update equation for $\mathbf{W}$ as

$$\mathbf{v}'_{w_I}{}^{(\text{new})} = \mathbf{v}'_{w_I}{}^{(\text{old})} - \eta \cdot \mathrm{EH} \tag{30}$$

Here $\mathbf{v_{w_I}}$ is a row of $\mathbf{W}$ (namely the input vector of the (only) context word), and because of the one-hot encoding it is the only row of $\mathbf{W}$ whose derivative is non-zero.

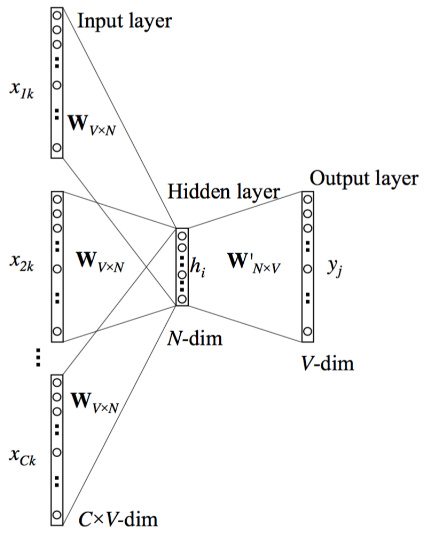

Figure 2: General CBOW Model

3 General Continuous Bag-of-Words Model

4 Skip-Gram Model

The Skip-Gram model was introduced in Mikolov [3] and is depicted in Figure 3. As you can see, Skip-Gram is the opposite of CBOW; in Skip-Gram we predict the context C given and input word, where is in CBOW we predict the word from C. Basically the training objective of the Skip-Gram model is to learn word vector representations that are good at predicting nearby words in the associated context(s).

For the Skip-Gram model we continue to use w_{w_i} to denote the input vector of the only word on the input layer, and as a result have the same definition of the hidden-layer outputs h as in Equation 1 (again this means h copies a row from the input $\rightarrow$ hidden weight matrix $\mathbf{W}$ associated with input word w_I). Recall that the definition of $\mathbf{h}$ was

$$\mathbf{h} = \mathbf{W}_{(k,.)} := v_{\mathrm{wI}} \tag{31}$$

Now, at the output layer, instead of outputting one multinomial distribution, we output C multinomial distributions. Each output is computed using the same hidden $\rightarrow$ output matrix as follows:

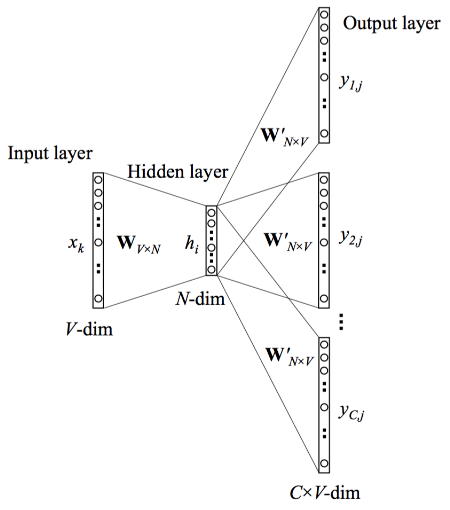

Figure 3: Skip-Gram Model

$$p(w_{\mathrm{c,j}} = w_{\mathrm{O,c}} | w_{\mathrm{I}}) = y_{\mathrm{c,j}} = \frac{exp(u_{\mathrm{c,j}})}{\sum\limits_{j'=1}^{V} exp(u_{j'})} \tag{32}$$

where $w_{\mathrm{c,j}}$ is the j-th word on the c-th panel of the output layer. $w_{O,c}$ is the actual c-th word in the output context. Note that w_{I} is the (only) input word. $y_{c,j}$ is the output of the j-th unit on the c-th panel of the output layer. Finally, $u_{\mathrm{c,j}}$ is the input of the j-th unit on the c-th panel of the output layer.

Said in words, this is the probability that our prediction of the j-th word on the c-th panel, w_{cj}, equals the actual c-th output word, w_{Oc}, conditioned on the input word w_{I}.

Now, because the output layer panels share the same weights, we have

$$u_{\mathrm{c,j}} = u_{\mathrm{c}} = \mathbf{v}'_{w_j}{}^{T} \cdot \mathbf{h}, \text{ for } c = 1, 2, \ldots, C \tag{33}$$

where $\mathbf{v}'_{w_j}$ is the output vector of the j-th word w_j of the vocabulary[2].

Given all of this, the parameter update equations are not so different from the one context word CBOW model. The loss function is changed to

[2] $\mathbf{v}'_{w_j}$ is again a column of the hidden $\rightarrow$ output weight matrix $\mathbf{W}'$

$$E = -\log p(w_{\mathrm{O},1}, w_{\mathrm{O},2}, \ldots, w_{0,\mathrm{C}} | w_{\mathrm{I}}) \tag{34}$$

$$= -\log \prod_{c=1}^{C} \frac{exp(u_{c,j_c^*})}{\sum_{j'=1}^{V} exp(u_{j'})} \tag{35}$$

$$= -\sum_{c=1}^{C} u_{c,j_c^*} + C \cdot \log \sum_{j'=1}^{V} exp(u_{j'}) \tag{36}$$

where j_c^* is the index of the actual c-th output context word in V [3].

Now, if we take the derivative of E with respect to the net input of every unit on panel of the output layer (i.e., $u_{c,j}$), we get

$$\frac{\partial E}{\partial u_{c,j}} = y_{c,j} - t_{c,j} := e_{c,j} \tag{37}$$

which again is the prediction error on the unit (same as in Equation 15). Now, let $\mathrm{EI} = \{\mathrm{EI}_1, \mathrm{EI}_2, \ldots, \mathrm{EI}_V\}$ (EI is a V-dimensional vector) as the sum of the prediction errors over all context words, that is,

$$\mathrm{EI}_j = \sum_{c=1}^{C} e_{c,j} \tag{38}$$

Now you can find the derivative of E with respect to $\mathbf{W}'$ as follows:

$$\frac{\partial E}{\partial w'_{ij}} = \sum_{c=1}^{C} \frac{\partial E}{\partial u_{cj}} \cdot \frac{\partial u_{cj}}{\partial w'_{ij}} = \mathrm{EI}_j \cdot h_i \tag{39}$$

Given all of this machinery we can now get the update equation for the hidden $\to$ output matrix $\mathbf{W}'$, which might look familiar by this time:

$${w'_{ij}}^{(\mathrm{new})} = {w'_{ij}}^{(\mathrm{old})} - \eta \cdot \mathrm{EI}_j \cdot h_i \tag{40}$$

or

$${\mathbf{v}'_{wj}}^{(\mathrm{new})} = {\mathbf{v}'_{wj}}^{(\mathrm{old})} - \eta \cdot \mathrm{EI}_j \cdot \mathbf{h} \qquad j = 1, 2, \ldots, V \tag{41}$$

[3]Recall that $\log AB = \log A + \log B$

4.1 Revisiting Learning In Neural Probabilisitic Language Models

We saw in previous sections that both CBOW and Skip-Gram Neural Probabilisitic Language Models (NPLMs) have well-defined posterior distributions and loss functions. First, the basic form of these posteriors is as follows (and roughly following the notation in [6]): In a neural language model, either CBOW or Skip-Gram, the conditional distribution corresponding to context c, $P^c(w)$, is defined to be

$$P^c_\theta = \frac{exp(s_\theta(w,c))}{\sum\limits_{w'} exp(s_\theta(w',c))} \qquad \text{\# see e.g., Equation 32} \tag{42}$$

where $s_\theta(w,c)$ is a *scoring* function (i.e., Equation 9 above) with parameters θ which quantifies the compatibility of word w with context c.[4]

Note that a NPLM represents each word in the vocabulary using a real-valued vector and defines the scoring function (s_θ or equivalently u_j) in terms of vectors of the context words and the next word. The important point here is that these vectors account for most of the parameters in neural language models, which in turn means that their memory requirements are linear in the vocabulary size. More generally, the time complexity of these models is $\mathcal{O}(|V| \cdot n)$, where $|V|$ is the size of the input vocabulary and n is the size of the input vectors $\mathbf{x} \in \mathbb{R}^n$. If $\mathbf{x}$ is one-hot encoded (such as in the CBOW or Skip-Gram models), then this complexity is $\mathcal{O}(|V|^2)$,

4.2 An Obvious Question

A question one might ask is *why*, given the reported superior accuracy of NPLMs, had they until recently been far less widely used than n-gram models? The answer is due to their notoriously long training times, which had been measured in weeks even for moderately-sized datasets. But then what is causing this expensive training? The answer is that training NPLMs is computationally expensive because they are **explicitly normalized**. For example, consider the denominator in Equation 42, which essentially requires that we consider *all* words in the vocabulary when computing the posterior distributions (or in the training context, the log-likelihood gradients, such as in Equation 34). This points to a problem with softmax classifiers in general, namely, that they are explicitly normalized.

Given these complexity considerations, [4] describes two alternative training objectives for the Skip-Gram model: Hierarchical softmax and Skip-Gram Negative Sampling (SGNS); we will focus on SGNS here. The rest of this section is orgamized as follows: Section 4.3 reviews

[4]$E = -s_\theta(w,c)$ is sometimes referred to as the *energy function* [1].

the basics of Parametric Density Estimation, and Section 4.4 describes Noise-Contrastive Estimation (NCE), where we consider the situation where the model probability density function is unnormalized (recall that the problem with softmax is that it is computationally expensive due to the requirement for explicit normalization). Finally, Section 4.6 looks at a modification of NCE called Negative Sampling.

4.3 Basics of Parametric Density Estimation

The basic set up for parametric density estimation is that we sample $X = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{T_d})$ from random vector $\mathbf{x} \in \mathbb{R}^n$. This is the observed data which follows an *unknown* probability distribution function (pdf) p_d. This data pdf (p_d) is modeled by a parameterized set of functions $\{p_m(.;\boldsymbol{\theta})\}_\theta$ where $\boldsymbol{\theta}$ is a parameter vector. It is generally assumed (but not required) that p_d comes from this family, that is, $p_d(.) = p_m(.;\boldsymbol{\theta})$ for some parameter $\boldsymbol{\theta}^*$.

With these definitions we can describe the Parametric Density Estimation (PDE) problem. In particular, the PDE problem is about finding $\boldsymbol{\theta}^*$ from the observed sample X. Note also that any estimate $\hat{\boldsymbol{\theta}}$ must yield a normalized pdf $p_m(.;\boldsymbol{\theta})$ which satisfies two properties:

$$\int p_m(\mathbf{u};\hat{\boldsymbol{\theta}})d\mathbf{u} = 1 \tag{43}$$

and

$$p_m(.;\hat{\boldsymbol{\theta}}) \geq 0 \tag{44}$$

These are the two constraints on the estimation.

Now, if both constraints hold for all $\boldsymbol{\theta}$ (and not only $\hat{\boldsymbol{\theta}}$), then we say that the model is *normalized*. If the constraint in Equation 44 holds but Equation 43 does not, we say that the model is *unnormalized*. The assumption, however, is that there is at least one value of the parameters for which an unnormalized model integrates to one (Equation 43), namely $\boldsymbol{\theta}^*$.

Next, denote an unnormalized model, parameterized by some $\boldsymbol{\alpha}$ as $p_m^0(.;\boldsymbol{\alpha})$. Then the partition function $Z(\boldsymbol{\alpha})$ is defined as

$$Z(\boldsymbol{\alpha}) = \int p_m^0(\mathbf{u};\boldsymbol{\alpha})d\mathbf{u} \tag{45}$$

$Z(\boldsymbol{\alpha})$ can be used to convert the unnormalized model $p_m^0(\mathbf{u};\boldsymbol{\alpha})$ into a normalized one: $p_m^0(\mathbf{u};\boldsymbol{\alpha})/Z(\boldsymbol{\alpha})$, which integrates to one for every value of $\boldsymbol{\alpha}$ (as required by Equation 43).

If we rewrite Equation 42 in terms of the partition function Z, can see that

$$Z(\boldsymbol{\theta}) = \sum_{w'} exp(s_\theta(w', h)) \tag{46}$$

$$P_\theta^h = \frac{exp(s_\theta(w, h))}{Z(\boldsymbol{\theta})} \tag{47}$$

Note that I changed the symbol we're using for the context c to h (also sometimes used for the context) to avoid name clashes below.

Unfortunately, the function $\boldsymbol{\alpha} \mapsto Z(\boldsymbol{\alpha})$ is defined by the integral in Equation 45 which, unless $p_m^0(.;\boldsymbol{\alpha})$ has a particularly convenient form, is likely intractable and/or doesn't have a nice closed form. In particular, the integral will not be amenable to analytic computation so a closed form for $Z(\boldsymbol{\alpha})$ can't be found. In addition, for low-dimensional problems, numerical methods can be used to approximate $Z(\boldsymbol{\alpha})$ to a very high accuracy (MCMC, Gibbs, or other sampling techniques), but for high-dimensional problems numeric methods are computationally expensive. Since we are considering the Skip-Gram model here, we are dealing with a PDE problem in high dimension where computation of the partition function is analytically intractable and/or computationally expensive.

4.4 Noise Contrastive Estimation

Noise Contrastive Estimation (NCE) was introduced in [2]. The basic idea is to consider Z (or alternatively $c = \ln 1/Z$) not as a function of $\boldsymbol{\alpha}$ but rather as an additional parameter to the model. Here the unnormalized model $p_m^0(.;\boldsymbol{\alpha})$ is extended with an additional normalizing parameter (c, note the change in meaning of c from context to the normalizing parameter) and then we estimate

$$\ln p_m(.;\boldsymbol{\alpha}) = \ln p_m^0(.;\boldsymbol{\alpha}) + c \tag{48}$$

with parameters $\boldsymbol{\theta} = (\boldsymbol{\alpha}, c)$. The estimate $\hat{\boldsymbol{\theta}} = (\hat{\boldsymbol{\alpha}}, \hat{c})$ is intended to make the unnormalized model $p_m^0(.;\hat{\boldsymbol{\alpha}})$ match the *shape* of p_d and $\hat{c}$ provides the proper *scaling* so that the constraints (Equations 43 and 44) hold. Note that separating estimation of shape and scale is not possible for maximum likelihood estimation (MLE) since the likelihood can be made arbitrarily large by setting the normalizing parameter c successively larger values.

The key observation underlying NCE is that density estimation is largely about characterizing properties of the observed data $X = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{T_d})$, and a convenient way to describe properties is to describe them relative to the properties of some reference data Y.

Now, assume that the reference (noise) data Y is an iid sample[5] $(\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{T_n})$ of a random variable $\mathbf{y} \in \mathbb{R}^n$ with probability distribution function (pdf) p_n, and let the (unknown) pdf of X be p_d. Then a relative description of the data X can be given by the ratio p_d/p_n of the two density functions. If the reference distribution p_n is known (which it is), we can recover p_d from the ratio p_d/p_n. That is, since we know the differences between X and Y and also the properties of Y, we can deduce the properties of X. Finally, following [2], we assume that the noise samples are k times more frequent than data samples so data points come from the mixture

$$\frac{1}{k+1} P_d^h(w) + \frac{k}{k+1} P_n(w) \tag{49}$$

NCE connects the problem of PDE to supervised learning, in particular to logistic regression, and provides a hint as to how the proposed estimator works: By discriminating, or comparing, between data and noise, NCE can learn properties of the data in the form of a statistical model. That is, the key idea behind noise-contrastive estimation is ?learning by comparison?.

So how does this supervised learning work and what exactly does it estimate? Consider first the following the notation (which with minor modifications largely follows [2]): Let $U = (X \cap Y) = (\mathbf{u}_1, \mathbf{x}_2, \ldots, \mathbf{u}_{T_d+T_n})$. NCE then converts the problem of density estimation to a binary classification problem as follows: For each $\mathbf{u}_t \in U$ assign a class label C_t such that $C_t = 1$ if $\mathbf{u}_t \in X$ and $C_t = 0$ if $\mathbf{u}_t \in Y$. Now we can use logistic regression to estimate the posterior probabilities since

$$\begin{aligned}
P(A) &= \sum_n P(A \cap B_n) && \text{\# by the \textit{Sum Rule}} \quad (50)\\
&= \sum_n P(A, B_n) && \text{\# alternate notation} \quad (51)\\
&= \sum_n P(A|B_n)P(B_n) && \text{\# by the \textit{Product Rule}} \quad (52)
\end{aligned}$$

so that the posterior distribution $P(\mathcal{C}_1|\mathbf{x})$ for two classes $\mathcal{C}_1$ and $\mathcal{C}_2$ given input vector $\mathbf{x}$ would look like

[5]Independent Identically Distributed

$$P(\mathcal{C}_1|\mathbf{x}) = \frac{P(\mathbf{x}|\mathcal{C}_1)P(\mathcal{C}_1)}{P(\mathbf{x}|\mathcal{C}_1)P(\mathcal{C}_1) + P(\mathbf{x}|\mathcal{C}_2)P(\mathcal{C}_2)} \tag{53}$$

Interestingly, the posterior distribution is related to logistic regression as follows: First recall that the posterior $P(\mathcal{C}_1|\mathbf{x})$ is

$$P(\mathcal{C}_1|\mathbf{x}) = \frac{P(\mathbf{x}|\mathcal{C}_1)P(\mathcal{C}_1)}{P(\mathbf{x}|\mathcal{C}_1)P(\mathcal{C}_1) + P(\mathbf{x}|\mathcal{C}_2)P(\mathcal{C}_2)} \tag{54}$$

Now, if we set

$$a = \ln \frac{P(\mathbf{x}|\mathcal{C}_1)P(\mathcal{C}_1)}{P(\mathbf{x}|\mathcal{C}_2)P(\mathcal{C}_2)} \tag{55}$$

we can see that

$$P(\mathcal{C}_1|\mathbf{x}) = \frac{1}{1+e^{-a}} = \sigma(a) \tag{56}$$

that is, the sigmoid function. This starts to give us a sense that the sigmoid function is related to the log of the ratio of likelihoods of $p(\mathbf{x}|\mathcal{C}_1)$ and $p(\mathbf{x}|\mathcal{C}_2)$, or in our context, p_d/p_n.

Now, since the pdf p_d of $\mathbf{x}$ is unknown, we cam model the class-conditional probability $p(.|C=1)$ with $p_m(.;\boldsymbol{\theta})$, and the class conditional probability densities are

$$p(\boldsymbol{u}|C=1;\boldsymbol{\theta}) = p_m(\boldsymbol{u};\boldsymbol{\theta}) \tag{57}$$
$$p(\boldsymbol{u}|C=0;\boldsymbol{\theta}) = p_n(\boldsymbol{u}) \tag{58}$$

So the prior probabilities are

$$p(C=1) = \frac{T_d}{T_d+T_n} \tag{59}$$
$$p(C=0) = \frac{T_n}{T_d+T_n} \tag{60}$$

and the posteriors are therefore

$$P(C=1|\mathbf{u};\boldsymbol{\theta}) = \frac{p_m(\mathbf{u};\boldsymbol{\theta})}{p_m(\mathbf{u};\boldsymbol{\theta}) + k \cdot p_n(\mathbf{u})} \tag{61}$$

$$P(C=0|\mathbf{u};\boldsymbol{\theta}) = \frac{k \cdot p_n(\mathbf{u})}{p_m(\mathbf{u};\boldsymbol{\theta}) + k \cdot p_n(\mathbf{u})} \tag{62}$$

where k is the ratio $P(C=0)/P(C=1) = T_n/T_d$ (remembering that noise samples $\mathbf{y}_i$ are k times more frequent that data samples $\mathbf{x}_i$).

Note that the class labels C_t are Bernoulli-distributed. Recall the details of the Bernoulli distribution: First, the random variable Y takes values $y_i \in \{0,1\}$. Then the Bernoulli distribution is a $Binomial(1,p)$ distribution, where $0 < p < 1$ and $P(Y=y) = p^y(1-p)^{1-y}$. The the probability that $Y_i = y_i$ for $i = 1, 2, \ldots, n$ is

$$P(Y) = \prod_{i=1}^{n} p^{y_i}(1-p)^{1-y_i} \tag{63}$$

and the log likelihood $\ell_n(p)$ is

$$\ell_n(p) = \sum_{i=1}^{n} \big[Y_i \log p + (1-Y_i)\log(1-p)\big] \tag{64}$$

Returning to NCE, the log-likelihood of the parameters θ is then

$$\ell(\theta) = \sum_{t=1}^{T_d+T_n} C_t \ln P(C_t=1|\mathbf{u}_t;\boldsymbol{\theta}) + (1-C_t)\ln P(C_t=0|\mathbf{u}_t;\boldsymbol{\theta}) \tag{65}$$

$$= \sum_{t=1}^{T_d} \ln\big[h(\mathbf{x}_t;\boldsymbol{\theta})\big] + \sum_{t=1}^{T_n} \ln\big[1 - h(\mathbf{y}_t;\boldsymbol{\theta})\big] \tag{66}$$

where

$$G(\mathbf{u};\theta) = \ln p_m(\mathbf{u};\theta) - \ln p_n(\mathbf{u}) \tag{67}$$

$$h(\mathbf{u};\theta) = \sigma(G(\mathbf{u};\theta)) \qquad \# \; \sigma(x) = 1/(1+e^{-x}) \tag{68}$$

Optimizing $\ell(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$ leads to an estimate $G(.;\boldsymbol{\theta})$ of the log ratio $\ln(p_d/p_n)$ (see Equation 55 for the derivation). That is, an approximate description of X relative to

Y is given by Equation 66. Interestingly, the sign inverted objective function, $-\ell(\boldsymbol{\theta})$, is also known as the cross entropy (or cross entropy error function).

So density estimation, which is an *unsupervised* learning task, can be estimated with a *supervised* learning technique, namely logistic regression. The important result here is that even *unnormalized* models can be estimated using the same principle.

Now, given an unnormalized statistical model $p_m^0(.;\boldsymbol{\alpha})$, the NCE technique adds an additional *normalization* parameter c to the model, and defines

$$\ln p_m(.;\boldsymbol{\alpha}) = \ln p_m^0(.;\boldsymbol{\alpha}) + c \tag{69}$$

where $\boldsymbol{\theta} = (\boldsymbol{\alpha}, c)$. The parameter c *scales* the unnormalized model so that the Equation 43 holds. After learning, $\hat{c}$ provides an estimate of $\ln 1/Z(\hat{\boldsymbol{\alpha}})$ (this is closely related to Equation 55).

4.5 NCE Cost Function

Let $X = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{T_d})$ consist of T_d independent observations of $\mathbf{x} \in \mathbb{R}^n$. Similarily, $Y = (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{x}_{T_n})$ is a artificially generated data set that consists of $T_n = kT_d$ independent observations of $\mathbf{y} \in \mathbb{R}^n$ with known distribution p_n. The cost function $J_T(\boldsymbol{\theta})$ is defined to be (look familiar?):

$$J_T(\boldsymbol{\theta}) = \frac{1}{T_d}\left\{ \sum_{t=1}^{T_d} \ln\left[h(x;\boldsymbol{\theta})\right] + \sum_{t=1}^{T_n} \ln\left[1 - h(\mathbf{y}_i;\boldsymbol{\theta})\right] \right\} \tag{70}$$

and the NCE estimator is defined to be the argument $\hat{\theta}$ which minimizes $-J_T(\boldsymbol{\theta})$ (or alternatively, maximizes $J_T(\boldsymbol{\theta})$) and $h(.;\boldsymbol{\theta})$ is the nonlinearity defined in Equation 68.

4.6 Skip-Gram Negative Sampling

Mikolov et al. [5] introduce Skip-Gram Negative Sampling in as an alternative to the hierarchical softmax method outlined there. Negative Sampling is a form NCE (see Section 4.4). The key assertion underlying NCE is that a good model should be able to differentiate data from noise by means of logistic regression. And while NCE can be showin to approximately maximize the log probability of the softmax, the authors point out that the Skip-Gram model is only concerned with learning high-quality vector representations, and such the authors were free to simplify NCE as long as the vector representations retain their quality. The negative sampling (NEG) objective is defined to be

$$\log \sigma({v'_{\mathrm{WO}}}^T v_{\mathrm{WI}}) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)} \big[\log \sigma(-{v'_{W_i}}^T v_{WI} \big] \tag{71}$$

which they claim is used to replace every occurrence of $\log P(w_O|w_I)$ in the Skip-Gram objective (though exactly how this work isn't discussed).

5 Acknowledgements

References

[1] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, March 2003.

[2] Michael U. Gutmann and Aapo Hyvärinen. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *J. Mach. Learn. Res.*, 13:307–361, February 2012.

[3] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. 01 2013.

[4] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. word2vec, 2014.

[5] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013.

[6] Andriy Mnih and Yee Whye Teh. A fast and simple algorithm for training neural probabilistic language models. 06 2012.