word2vec Parameter Learning Explained

Xin Rong
ronxin@umich.edu

Abstract

The word2vec model and application by Mikolov et al. have attracted a great amount of attention in recent two years. The vector representations of words learned by word2vec models have been shown to carry semantic meanings and are useful in various NLP tasks. As an increasing number of researchers would like to experiment with word2vec or similar techniques, I notice that there lacks a material that comprehensively explains the parameter learning process of word embedding models in details, thus preventing researchers that are non-experts in neural networks from understanding the working mechanism of such models.

This note provides detailed derivations and explanations of the parameter update equations of the word2vec models, including the original continuous bag-of-word (CBOW) and skip-gram (SG) models, as well as advanced optimization techniques, including hierarchical softmax and negative sampling. Intuitive interpretations of the gradient equations are also provided alongside mathematical derivations.

In the appendix, a review on the basics of neuron networks and backpropagation is provided. I also created an interactive demo, wevi, to facilitate the intuitive understanding of the model.[1]

1 Continuous Bag-of-Word Model

1.1 One-word context

We start from the simplest version of the continuous bag-of-word model (CBOW) introduced in Mikolov et al. (2013a). 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 like a bigram model. For readers who are new to neural networks, it is recommended that one go through Appendix A for a quick review of the important concepts and terminologies before proceeding further.

Figure 1 shows the network model under the simplified context definition[2]. In our setting, the vocabulary size is V, and the hidden layer size is N. The units on adjacent

[1] An online interactive demo is available at: `http://bit.ly/wevi-online`.

[2] In Figures 1, 2, 3, and the rest of this note, $\mathbf{W}'$ is not the transpose of $\mathbf{W}$, but a different matrix instead.

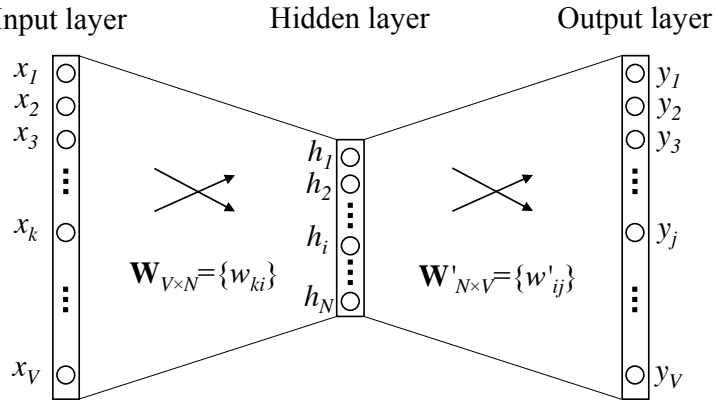

Figure 1: A simple CBOW model with only one word in the context

layers are fully connected. The input is a one-hot encoded vector, which means for a given input context word, only one out of V units, $\{x_1, \cdots, x_V\}$, will be 1, and all other units are 0.

The weights between the input layer and the hidden layer can be represented by a $V \times N$ matrix $\mathbf{W}$. Each row of $\mathbf{W}$ is the N-dimension vector representation $\mathbf{v}_w$ of the associated word of the input layer. Formally, row i of $\mathbf{W}$ is $\mathbf{v}_w^T$. Given a context (a word), assuming $x_k = 1$ and $x_{k'} = 0$ for $k' \neq k$, we have

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

which is essentially copying the k-th row of $\mathbf{W}$ to $\mathbf{h}$. $\mathbf{v}_{w_I}$ is the vector representation of the input word w_I. This implies that the link (activation) function of the hidden layer units is simply *linear* (i.e., directly passing its weighted sum of inputs to the next layer).

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

$$u_j = {\mathbf{v}'_{w_j}}^T \mathbf{h}, \tag{2}$$

where $\mathbf{v}'_{w_j}$ is the j-th column of the matrix $\mathbf{W}'$. Then we can use softmax, a log-linear classification model, to obtain the posterior distribution of words, which is a multinomial distribution.

$$p(w_j|w_I) = y_j = \frac{\exp(u_j)}{\sum_{j'=1}^{V} \exp(u_{j'})}, \tag{3}$$

where y_j is the output of the j-th unit in the output layer. Substituting (1) and (2) into

(3), we obtain

$$p(w_j|w_I) = \frac{\exp\left({\mathbf{v}'_{w_j}}^T \mathbf{v}_{w_I}\right)}{\sum_{j'=1}^{V} \exp\left({\mathbf{v}'_{w_{j'}}}^T \mathbf{v}_{w_I}\right)} \tag{4}$$

Note that $\mathbf{v}_w$ and $\mathbf{v}'_w$ are two representations of the word w. $\mathbf{v}_w$ comes from rows of $\mathbf{W}$, which is the input→hidden weight matrix, and $\mathbf{v}'_w$ comes from columns of $\mathbf{W}'$, which is the hidden→output matrix. In subsequent analysis, we call $\mathbf{v}_w$ as the "**input vector**", and $\mathbf{v}'_w$ as the "**output vector**" of the word w.

Update equation for hidden→output weights

Let us now derive the weight update equation for this model. Although the actual computation is impractical (explained below), we are doing the derivation to gain insights on this original model with no tricks applied. For a review of basics of backpropagation, see Appendix A.

The training objective (for one training sample) is to maximize (4), 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 with regard to the weights. We take the logarithm of the conditional probability and use it to define our loss function.

$$\log p(w_O|w_I) = \log y_{j^*} \tag{5}$$

$$= u_{j^*} - \log \sum_{j'=1}^{V} \exp(u_{j'}) := -E, \tag{6}$$

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

Let us now derive the update equation of the weights between hidden and output layers. Take the derivative of E with regard to j-th unit's net input u_j, we obtain

$$\frac{\partial E}{\partial u_j} = y_j - t_j := e_j \tag{7}$$

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

Next we take the derivative on w'_{ij} to obtain the gradient on the hidden→output weights.

$$\frac{\partial E}{\partial w'_{ij}} = \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial w'_{ij}} = e_j \cdot h_i \tag{8}$$

Therefore, using stochastic gradient descent, we obtain the weight updating equation for hidden→output weights:

$$w_{ij}'^{\text{(new)}} = w_{ij}'^{\text{(old)}} - \eta \cdot e_j \cdot h_i. \tag{9}$$

or

$$\mathbf{v}_{w_j}'^{\text{(new)}} = \mathbf{v}_{w_j}'^{\text{(old)}} - \eta \cdot e_j \cdot \mathbf{h} \qquad \text{for } j = 1, 2, \cdots, V. \tag{10}$$

where $\eta > 0$ is the learning rate, $e_j = y_j - t_j$, and h_i is the i-th unit in the hidden layer; $\mathbf{v}'_{w_j}$ is the output vector of w_j. Note that this update equation implies that we have to go through every possible word in the vocabulary, check its output probability y_j, and compare y_j with its expected output t_j (either 0 or 1). If $y_j > t_j$ ("overestimating"), then we subtract a proportion of the hidden vector $\mathbf{h}$ (i.e., $\mathbf{v}_{w_I}$) from $\mathbf{v}'_{w_j}$, thus making $\mathbf{v}'_{w_j}$ farther away from $\mathbf{v}_{w_I}$; if $y_j < t_j$ ("underestimating", which is true only if $t_j = 1$, i.e., $w_j = w_O$), we add some $\mathbf{h}$ to $\mathbf{v}'_{w_O}$, thus making $\mathbf{v}'_{w_O}$ closer[3] to $\mathbf{v}_{w_I}$. If y_j is very close to t_j, then according to the update equation, very little change will be made to the weights. Note, again, that $\mathbf{v}_w$ (input vector) and $\mathbf{v}'_w$ (output vector) are two different vector representations of the word w.

Update equation for input→hidden weights

Having obtained the update equations for $\mathbf{W}'$, we can now move on to $\mathbf{W}$. We take the derivative of E on the output of the hidden layer, obtaining

$$\frac{\partial E}{\partial h_i} = \sum_{j=1}^{V} \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial h_i} = \sum_{j=1}^{V} e_j \cdot w'_{ij} := \mathrm{EH}_i \tag{11}$$

where h_i is the output of the i-th unit of the hidden layer; u_j is defined in (2), the net input of the j-th unit in the output layer; and $e_j = y_j - t_j$ is the prediction error of the j-th word in the output layer. EH, an N-dim vector, is the sum of the output vectors of all words in the vocabulary, weighted by their prediction error.

Next we should take the derivative of E on $\mathbf{W}$. First, recall that the hidden layer performs a linear computation on the values from the input layer. Expanding the vector notation in (1) we get

$$h_i = \sum_{k=1}^{V} x_k \cdot w_{ki} \tag{12}$$

Now we can take the derivative of E with regard to each element of $\mathbf{W}$, obtaining

$$\frac{\partial E}{\partial w_{ki}} = \frac{\partial E}{\partial h_i} \cdot \frac{\partial h_i}{\partial w_{ki}} = \mathrm{EH}_i \cdot x_k \tag{13}$$

[3]Here when I say "closer" or "farther", I meant using the inner product instead of Euclidean as the distance measurement.

This is equivalent to the tensor product of $\mathbf{x}$ and EH, i.e.,

$$\frac{\partial E}{\partial \mathbf{W}} = \mathbf{x} \otimes \mathrm{EH} = \mathbf{x}\mathrm{EH}^T \tag{14}$$

from which we obtain a $V \times N$ matrix. 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^T, an N-dim vector. We obtain the update equation of $\mathbf{W}$ as

$$\mathbf{v}_{w_I}^{(\mathrm{new})} = \mathbf{v}_{w_I}^{(\mathrm{old})} - \eta \mathrm{EH}^T \tag{15}$$

where $\mathbf{v}_{w_I}$ is a row of $\mathbf{W}$, the "input vector" of the only context word, and is the only row of $\mathbf{W}$ whose derivative is non-zero. All the other rows of $\mathbf{W}$ will remain unchanged after this iteration, because their derivatives are zero.

Intuitively, since vector EH is the sum of output vectors of all words in vocabulary weighted by their prediction error $e_j = y_j - t_j$, we can understand (15) as adding a portion of every output vector in vocabulary to the input vector of the context word. If, in the output layer, the probability of a word w_j being the output word is overestimated ($y_j > t_j$), then the input vector of the context word w_I will tend to move farther away from the output vector of w_j; conversely if the probability of w_j being the output word is underestimated ($y_j < t_j$), then the input vector w_I will tend to move closer to the output vector of w_j; if the probability of w_j is fairly accurately predicted, then it will have little effect on the movement of the input vector of w_I. The movement of the input vector of w_I is determined by the prediction error of all vectors in the vocabulary; the larger the prediction error, the more significant effects a word will exert on the movement on the input vector of the context word.

As we iteratively update the model parameters by going through context-target word pairs generated from a training corpus, the effects on the vectors will accumulate. We can imagine that the output vector of a word w is "dragged" back-and-forth by the input vectors of w's co-occurring neighbors, as if there are physical strings between the vector of w and the vectors of its neighbors. Similarly, an input vector can also be considered as being dragged by many output vectors. This interpretation can remind us of gravity, or force-directed graph layout. The equilibrium length of each imaginary string is related to the strength of cooccurrence between the associated pair of words, as well as the learning rate. After many iterations, the relative positions of the input and output vectors will eventually stabilize.

1.2 Multi-word context

Figure 2 shows the CBOW model with a multi-word context setting. When computing the hidden layer output, instead of directly copying the input vector of the input context word, the CBOW model takes the average of the vectors of the input context words, and

use the product of the input→hidden weight matrix and the average vector as the output.

$$\mathbf{h} = \frac{1}{C}\mathbf{W}^T(\mathbf{x}_1 + \mathbf{x}_2 + \cdots + \mathbf{x}_C) \tag{16}$$

$$= \frac{1}{C}(\mathbf{v}_{w_1} + \mathbf{v}_{w_2} + \cdots + \mathbf{v}_{w_C})^T \tag{17}$$

where C is the number of words in the context, $w_1, \cdots, w_C$ are the words the in the context, and $\mathbf{v}_w$ is the input vector of a word w. The loss function is

$$E = = -\log p(w_O|w_{I,1}, \cdots, w_{I,C}) \tag{18}$$

$$= -u_{j^*} + \log \sum_{j'=1}^{V} \exp(u_{j'}) \tag{19}$$

$$= -\mathbf{v}'^{T}_{w_O} \cdot \mathbf{h} + \log \sum_{j'=1}^{V} \exp(\mathbf{v}'^{T}_{w_j} \cdot \mathbf{h}) \tag{20}$$

which is the same as (6), the objective of the one-word-context model, except that $\mathbf{h}$ is different, as defined in (17) instead of (1).

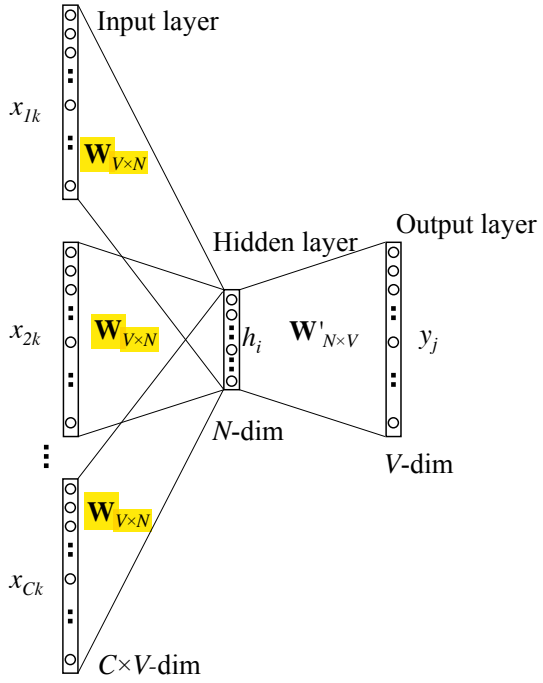

Figure 2: Continuous bag-of-word model

The update equation for the hidden→output weights stay the same as that for the one-word-context model (10). We copy it here:

$$\mathbf{v}'^{\,(\text{new})}_{w_j} = \mathbf{v}'^{\,(\text{old})}_{w_j} - \eta \cdot e_j \cdot \mathbf{h} \qquad \text{for } j = 1, 2, \cdots, V. \tag{21}$$

Note that we need to apply this to every element of the hidden→output weight matrix for each training instance.

The update equation for input→hidden weights is similar to (15), except that now we need to apply the following equation for every word $w_{I,c}$ in the context:

$$\mathbf{v}^{(\text{new})}_{w_{I,c}} = \mathbf{v}^{(\text{old})}_{w_{I,c}} - \frac{1}{C} \cdot \eta \cdot \text{EH}^T \qquad \text{for } c = 1, 2, \cdots, C. \tag{22}$$

where $\mathbf{v}_{w_{I,c}}$ is the input vector of the c-th word in the input context; η is a positive learning rate; and $\text{EH} = \frac{\partial E}{\partial h_i}$ is given by (11). The intuitive understanding of this update equation is the same as that for (15).

2 Skip-Gram Model

The skip-gram model is introduced in Mikolov et al. (2013a,b). Figure 3 shows the skip-gram model. It is the opposite of the CBOW model. The target word is now at the input layer, and the context words are on the output layer.

We still use $\mathbf{v}_{w_I}$ to denote the input vector of the only word on the input layer, and thus we have the same definition of the hidden-layer outputs $\mathbf{h}$ as in (1), which means $\mathbf{h}$ is simply copying (and transposing) a row of the input→hidden weight matrix, $\mathbf{W}$, associated with the input word w_I. We copy the definition of $\mathbf{h}$ below:

$$\mathbf{h} = \mathbf{W}^T_{(k,\cdot)} := \mathbf{v}^T_{w_I}, \tag{23}$$

On the output layer, instead of outputing one multinomial distribution, we are outputing C multinomial distributions. Each output is computed using the same hidden→output matrix:

$$p(w_{c,j} = w_{O,c} | w_I) = y_{c,j} = \frac{\exp(u_{c,j})}{\sum_{j'=1}^{V} \exp(u_{j'})} \tag{24}$$

where $w_{c,j}$ 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 words; 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; $u_{c,j}$ is the net input of the j-th unit on the c-th panel of the output layer. Because the output layer panels share the same weights, thus

$$u_{c,j} = u_j = \mathbf{v}'^{\,T}_{w_j} \cdot \mathbf{h}, \text{ for } c = 1, 2, \cdots, C \tag{25}$$

where $\mathbf{v}'_{w_j}$ is the output vector of the j-th word in the vocabulary, w_j, and also $\mathbf{v}'_{w_j}$ is taken from a column of the hidden→output weight matrix, $\mathbf{W}'$.

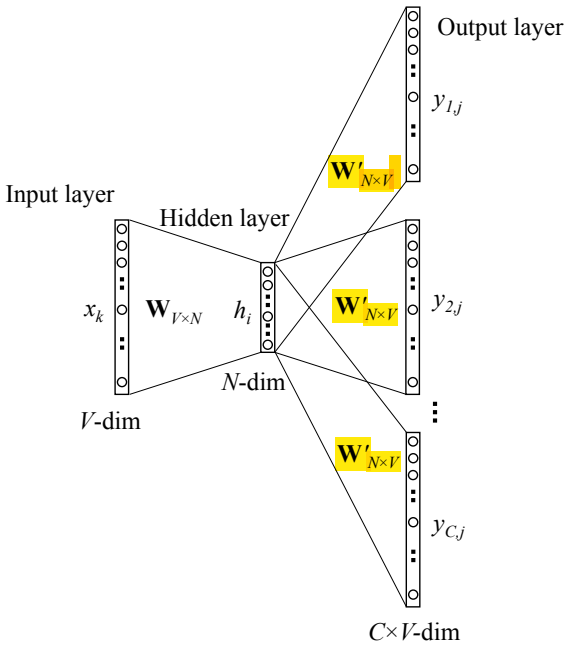

Figure 3: The skip-gram model.

The derivation of parameter update equations is not so different from the one-word-context model. The loss function is changed to

$$E = -\log p(w_{O,1}, w_{O,2}, \cdots, w_{O,C}|w_I) \tag{26}$$

$$= -\log \prod_{c=1}^{C} \frac{\exp(u_{c,j_c^*})}{\sum_{j'=1}^{V} \exp(u_{j'})} \tag{27}$$

$$= -\sum_{c=1}^{C} u_{j_c^*} + C \cdot \log \sum_{j'=1}^{V} \exp(u_{j'}) \tag{28}$$

where j_c^* is the index of the actual c-th output context word in the vocabulary.

We take the derivative of E with regard to the net input of every unit on every panel of the output layer, $u_{c,j}$ and obtain

$$\frac{\partial E}{\partial u_{c,j}} = y_{c,j} - t_{c,j} := e_{c,j} \tag{29}$$

which is the prediction error on the unit, the same as in (7). For notation simplicity, we define a V-dimensional vector $\mathrm{EI} = \{\mathrm{EI}_1, \cdots, \mathrm{EI}_V\}$ as the sum of prediction errors over all

context words:

$$\mathrm{EI}_j = \sum_{c=1}^{C} e_{c,j} \tag{30}$$

Next, we take the derivative of E with regard to the hidden→output matrix $\mathbf{W}'$, and obtain

$$\frac{\partial E}{\partial w'_{ij}} = \sum_{c=1}^{C} \frac{\partial E}{\partial u_{c,j}} \cdot \frac{\partial u_{c,j}}{\partial w'_{ij}} = \mathrm{EI}_j \cdot h_i \tag{31}$$

Thus we obtain the update equation for the hidden→output matrix $\mathbf{W}'$,

$$w'^{(\mathrm{new})}_{ij} = w'^{(\mathrm{old})}_{ij} - \eta \cdot \mathrm{EI}_j \cdot h_i \tag{32}$$

or

$$\mathbf{v}'^{(\mathrm{new})}_{w_j} = \mathbf{v}'^{(\mathrm{old})}_{w_j} - \eta \cdot \mathrm{EI}_j \cdot \mathbf{h} \qquad \text{for } j = 1, 2, \cdots, V. \tag{33}$$

The intuitive understanding of this update equation is the same as that for (10), except that the prediction error is summed across all context words in the output layer. Note that we need to apply this update equation for every element of the hidden→output matrix for each training instance.

The derivation of the update equation for the input→hidden matrix is identical to (11) to (15), except taking into account that the prediction error e_j is replaced with EI_j. We directly give the update equation:

$$\mathbf{v}^{(\mathrm{new})}_{w_I} = \mathbf{v}^{(\mathrm{old})}_{w_I} - \eta \cdot \mathrm{EH}^T \tag{34}$$

where EH is an N-dim vector, each component of which is defined as

$$\mathrm{EH}_i = \sum_{j=1}^{V} \mathrm{EI}_j \cdot w'_{ij}. \tag{35}$$

The intuitive understanding of (34) is the same as that for (15).

3 Optimizing Computational Efficiency

So far the models we have discussed ("bigram" model, CBOW and skip-gram) are both in their original forms, without any efficiency optimization tricks being applied.

For all these models, there exist two vector representations for each word in the vocabulary: the input vector $\mathbf{v}_w$, and the output vector $\mathbf{v}'_w$. Learning the input vectors is cheap; but learning the output vectors is very expensive. From the update equations (21) and (32), we can find that, in order to update $\mathbf{v}'_w$, for each training instance, we have to iterate through every word w_j in the vocabulary, compute their net input u_j, probability

prediction y_j (or $y_{c,j}$ for skip-gram), their prediction error e_j (or EI_j for skip-gram), and finally use their prediction error to update their output vector $\mathbf{v}'_j$.

Doing such computations for all words for every training instance is very expensive, making it impractical to scale up to large vocabularies or large training corpora. To solve this problem, an intuition is to limit the number of output vectors that must be updated per training instance. One elegant approach to achieving this is hierarchical softmax; another approach is through sampling, which will be discussed in the next section.

Both tricks optimize only the computation of the updates for output vectors. In our derivations, we care about three values: (1) E, the new objective function; (2) $\frac{\partial E}{\partial \mathbf{v}'_w}$, the new update equation for the output vectors; and (3) $\frac{\partial E}{\partial \mathbf{h}}$, the weighted sum of predictions errors to be backpropagated for updating input vectors.

3.1 Hierarchical Softmax

Hierarchical softmax is an efficient way of computing softmax (Morin and Bengio, 2005; Mnih and Hinton, 2009). The model uses a binary tree to represent all words in the vocabulary. The V words must be leaf units of the tree. It can be proved that there are $V-1$ inner units. For each leaf unit, there exists a unique path from the root to the unit; and this path is used to estimate the probability of the word represented by the leaf unit. See Figure 4 for an example tree.

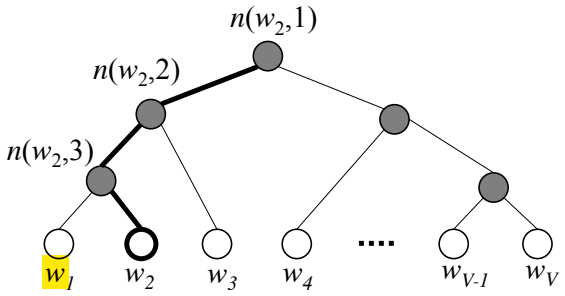

Figure 4: An example binary tree for the hierarchical softmax model. The white units are words in the vocabulary, and the dark units are inner units. An example path from root to w_2 is highlighted. In the example shown, the length of the path $L(w_2) = 4$. $n(w, j)$ means the j-th unit on the path from root to the word w.

In the hierarchical softmax model, there is no output vector representation for words. Instead, each of the $V-1$ inner units has an output vector $\mathbf{v}'_{n(w,j)}$. And the probability of a word being the output word is defined as

$$p(w = w_O) = \prod_{j=1}^{L(w)-1} \sigma\left(\llbracket n(w, j+1) = \mathrm{ch}(n(w, j)) \rrbracket \cdot {\mathbf{v}'_{n(w,j)}}^T \mathbf{h}\right) \tag{36}$$

where $\text{ch}(n)$ is the left child of unit n; $\mathbf{v}'_{n(w,j)}$ is the vector representation ("output vector") of the inner unit $n(w,j)$; $\mathbf{h}$ is the output value of the hidden layer (in the skip-gram model $\mathbf{h} = \mathbf{v}_{w_I}$; and in CBOW, $\mathbf{h} = \frac{1}{C}\sum_{c=1}^{C}\mathbf{v}_{w_c}$); $[\![x]\!]$ is a special function defined as

$$[\![x]\!] = \begin{cases} 1 & \text{if } x \text{ is true;} \\ -1 & \text{otherwise.} \end{cases} \tag{37}$$

Let us intuitively understand the equation by going through an example. Looking at Figure 4, suppose we want to compute the probability that w_2 being the output word. We define this probability as the probability of a random walk starting from the root ending at the leaf unit in question. At each inner unit (including the root unit), we need to assign the probabilities of going left and going right.[4] We define the probability of going left at an inner unit n to be

$$p(n, \text{left}) = \sigma\left({\mathbf{v}'_n}^T \cdot \mathbf{h}\right) \tag{38}$$

which is determined by both the vector representation of the inner unit, and the hidden layer's output value (which is then determined by the vector representation of the input word(s)). Apparently the probability of going right at unit n is

$$p(n, \text{right}) = 1 - \sigma\left({\mathbf{v}'_n}^T \cdot \mathbf{h}\right) = \sigma\left(-{\mathbf{v}'_n}^T \cdot \mathbf{h}\right) \tag{39}$$

Following the path from the root to w_2 in Figure 4, we can compute the probability of w_2 being the output word as

$$\begin{aligned} p(w_2 = w_O) &= p\left(n(w_2,1), \text{left}\right) \cdot p\left(n(w_2,2), \text{left}\right) \cdot p\left(n(w_2,3), \text{right}\right) && (40) \\ &= \sigma\left({\mathbf{v}'_{n(w_2,1)}}^T \mathbf{h}\right) \cdot \sigma\left({\mathbf{v}'_{n(w_2,2)}}^T \mathbf{h}\right) \cdot \sigma\left(-{\mathbf{v}'_{n(w_2,3)}}^T \mathbf{h}\right) && (41) \end{aligned}$$

which is exactly what is given by (36). It should not be hard to verify that

$$\sum_{i=1}^{V} p(w_i = w_O) = 1 \tag{42}$$

making the hierarchical softmax a well defined multinomial distribution among all words.

Now let us derive the parameter update equation for the vector representations of the inner units. For simplicity, we look at the one-word context model first. Extending the update equations to CBOW and skip-gram models is easy.

For the simplicity of notation, we define the following shortenizations without introducing ambiguity:

$$[\![\cdot]\!] := [\![n(w, j+1) = \text{ch}(n(w,j))]\!] \tag{43}$$

[4] While an inner unit of a binary tree may not always have both children, a binary Huffman tree's inner units always do. Although theoretically one can use many different types of trees for hierarchical softmax, word2vec uses a binary Huffman tree for fast training.

$$\mathbf{v}'_j := \mathbf{v}'_{n_{w,j}} \tag{44}$$

For a training instance, the error function is defined as

$$E = -\log p(w = w_O|w_I) = -\sum_{j=1}^{L(w)-1} \log \sigma([\![\cdot]\!]\mathbf{v}'^T_j \mathbf{h}) \tag{45}$$

We take the derivative of E with regard to $\mathbf{v}'_j\mathbf{h}$, obtaining

$$\begin{aligned} \frac{\partial E}{\partial \mathbf{v}'_j\mathbf{h}} &= \left(\sigma([\![\cdot]\!]\mathbf{v}'^T_j \mathbf{h}) - 1\right)[\![\cdot]\!] & (46)\\ &= \begin{cases} \sigma(\mathbf{v}'^T_j \mathbf{h}) - 1 & ([\![\cdot]\!] = 1) \\ \sigma(\mathbf{v}'^T_j \mathbf{h}) & ([\![\cdot]\!] = -1) \end{cases} & (47)\\ &= \sigma(\mathbf{v}'^T_j \mathbf{h}) - t_j & (48) \end{aligned}$$

where $t_j = 1$ if $[\![\cdot]\!] = 1$ and $t_j = 0$ otherwise.

Next we take the derivative of E with regard to the vector representation of the inner unit $n(w, j)$ and obtain

$$\frac{\partial E}{\partial \mathbf{v}'_j} = \frac{\partial E}{\partial \mathbf{v}'_j\mathbf{h}} \cdot \frac{\partial \mathbf{v}'_j\mathbf{h}}{\partial \mathbf{v}'_j} = \left(\sigma(\mathbf{v}'^T_j \mathbf{h}) - t_j\right) \cdot \mathbf{h} \tag{49}$$

which results in the following update equation:

$$\mathbf{v}'^{(\text{new})}_j = \mathbf{v}'^{(\text{old})}_j - \eta \left(\sigma(\mathbf{v}'^T_j \mathbf{h}) - t_j\right) \cdot \mathbf{h} \tag{50}$$

which should be applied to $j = 1, 2, \cdots, L(w) - 1$. We can understand $\sigma(\mathbf{v}'^T_j \mathbf{h}) - t_j$ as the prediction error for the inner unit $n(w, j)$. The "task" for each inner unit is to predict whether it should follow the left child or the right child in the random walk. $t_j = 1$ means the ground truth is to follow the left child; $t_j = 0$ means it should follow the right child. $\sigma(\mathbf{v}'^T_j \mathbf{h})$ is the prediction result. For a training instance, if the prediction of the inner unit is very close to the ground truth, then its vector representation $\mathbf{v}'_j$ will move very little; otherwise $\mathbf{v}'_j$ will move in an appropriate direction by moving (either closer or farther away[5] from $\mathbf{h}$) so as to reduce the prediction error for this instance. This update equation can be used for both CBOW and the skip-gram model. When used for the skip-gram model, we need to repeat this update procedure for each of the C words in the output context.

[5] Again, the distance measurement is inner product.

In order to backpropagate the error to learn input→hidden weights, we take the derivative of E with regard to the output of the hidden layer and obtain

$$\frac{\partial E}{\partial \mathbf{h}} = \sum_{j=1}^{L(w)-1} \frac{\partial E}{\partial \mathbf{v}'_j \mathbf{h}} \cdot \frac{\partial \mathbf{v}'_j \mathbf{h}}{\partial \mathbf{h}} \tag{51}$$

$$= \sum_{j=1}^{L(w)-1} \left(\sigma({\mathbf{v}'_j}^T \mathbf{h}) - t_j\right) \cdot \mathbf{v}'_j \tag{52}$$

$$:= \mathrm{EH} \tag{53}$$

which can be directly substituted into (22) to obtain the update equation for the input vectors of CBOW. For the skip-gram model, we need to calculate a EH value for each word in the skip-gram context, and plug the sum of the EH values into (34) to obtain the update equation for the input vector.

From the update equations, we can see that the computational complexity per training instance per context word is reduced from $O(V)$ to $O(\log(V))$, which is a big improvement in speed. We still have roughly the same number of parameters ($V-1$ vectors for inner-units compared to originally V output vectors for words).

3.2 Negative Sampling

The idea of negative sampling is more straightforward than hierarchical softmax: in order to deal with the difficulty of having too many output vectors that need to be updated per iteration, we only update a sample of them.

Apparently the output word (i.e., the ground truth, or positive sample) should be kept in our sample and gets updated, and we need to sample a few words as negative samples (hence "negative sampling"). A probabilistic distribution is needed for the sampling process, and it can be arbitrarily chosen. We call this distribution the noise distribution, and denote it as $P_n(w)$. One can determine a good distribution empirically.[6]

In word2vec, instead of using a form of negative sampling that produces a well-defined posterior multinomial distribution, the authors argue that the following simplified training objective is capable of producing high-quality word embeddings:[7]

$$E = -\log \sigma({\mathbf{v}'_{w_O}}^T \mathbf{h}) - \sum_{w_j \in \mathcal{W}_{\text{neg}}} \log \sigma(-{\mathbf{v}'_{w_j}}^T \mathbf{h}) \tag{54}$$

where w_O is the output word (i.e., the positive sample), and $\mathbf{v}'_{w_O}$ is its output vector; $\mathbf{h}$ is the output value of the hidden layer: $\mathbf{h} = \frac{1}{C}\sum_{c=1}^{C} \mathbf{v}_{w_c}$ in the CBOW model and $\mathbf{h} = \mathbf{v}_{w_I}$

[6] As described in (Mikolov et al., 2013b), word2vec uses a unigram distribution raised to the $\frac{3}{4}$th power for the best quality of results.

[7] Goldberg and Levy (2014) provide a theoretical analysis on the reason of using this objective function.

in the skip-gram model; $\mathcal{W}_{\text{neg}} = \{w_j | j = 1, \cdots, K\}$ is the set of words that are sampled based on $P_n(w)$, i.e., negative samples.

To obtain the update equations of the word vectors under negative sampling, we first take the derivative of E with regard to the net input of the output unit w_j:

$$\begin{aligned} \frac{\partial E}{\partial {\mathbf{v}'_{w_j}}^T \mathbf{h}} &= \begin{cases} \sigma({\mathbf{v}'_{w_j}}^T \mathbf{h}) - 1 & \text{if } w_j = w_O \\ \sigma({\mathbf{v}'_{w_j}}^T \mathbf{h}) & \text{if } w_j \in \mathcal{W}_{\text{neg}} \end{cases} && (55) \\ &= \sigma({\mathbf{v}'_{w_j}}^T \mathbf{h}) - t_j && (56) \end{aligned}$$

where t_j is the "label" of word w_j. $t = 1$ when w_j is a positive sample; $t = 0$ otherwise. Next we take the derivative of E with regard to the output vector of the word w_j,

$$\frac{\partial E}{\partial \mathbf{v}'_{w_j}} = \frac{\partial E}{\partial {\mathbf{v}'_{w_j}}^T \mathbf{h}} \cdot \frac{\partial {\mathbf{v}'_{w_j}}^T \mathbf{h}}{\partial \mathbf{v}'_{w_j}} = \left(\sigma({\mathbf{v}'_{w_j}}^T \mathbf{h}) - t_j\right) \mathbf{h} \tag{57}$$

which results in the following update equation for its output vector:

$$\mathbf{v}'^{\,(\text{new})}_{w_j} = \mathbf{v}'^{\,(\text{old})}_{w_j} - \eta \left(\sigma({\mathbf{v}'_{w_j}}^T \mathbf{h}) - t_j\right) \mathbf{h} \tag{58}$$

which only needs to be applied to $w_j \in \{w_O\} \cup \mathcal{W}_{\text{neg}}$ instead of every word in the vocabulary. This shows why we may save a significant amount of computational effort per iteration.

The intuitive understanding of the above update equation should be the same as that of (10). This equation can be used for both CBOW and the skip-gram model. For the skip-gram model, we apply this equation for one context word at a time.

To backpropagate the error to the hidden layer and thus update the input vectors of words, we need to take the derivative of E with regard to the hidden layer's output, obtaining

$$\begin{aligned} \frac{\partial E}{\partial \mathbf{h}} &= \sum_{w_j \in \{w_O\} \cup \mathcal{W}_{\text{neg}}} \frac{\partial E}{\partial {\mathbf{v}'_{w_j}}^T \mathbf{h}} \cdot \frac{\partial {\mathbf{v}'_{w_j}}^T \mathbf{h}}{\partial \mathbf{h}} && (59) \\ &= \sum_{w_j \in \{w_O\} \cup \mathcal{W}_{\text{neg}}} \left(\sigma({\mathbf{v}'_{w_j}}^T \mathbf{h}) - t_j\right) \mathbf{v}'_{w_j} := \text{EH} && (60) \end{aligned}$$

By plugging EH into (22) we obtain the update equation for the input vectors of the CBOW model. For the skip-gram model, we need to calculate a EH value for each word in the skip-gram context, and plug the sum of the EH values into (34) to obtain the update equation for the input vector.

Acknowledgement

The author would like to thank Eytan Adar, Qiaozhu Mei, Jian Tang, Dragomir Radev, Daniel Pressel, Thomas Dean, Sudeep Gandhe, Peter Lau, Luheng He, Tomas Mikolov, Hao Jiang, and Oded Shmueli for discussions on the topic and/or improving the writing of the note.

References

Goldberg, Y. and Levy, O. (2014). word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv:1402.3722 [cs, stat]*. arXiv: 1402.3722.

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

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119.

Mnih, A. and Hinton, G. E. (2009). A scalable hierarchical distributed language model. In Koller, D., Schuurmans, D., Bengio, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 21*, pages 1081–1088. Curran Associates, Inc.

Morin, F. and Bengio, Y. (2005). Hierarchical probabilistic neural network language model. In *AISTATS*, volume 5, pages 246–252. Citeseer.

A Back Propagation Basics

A.1 Learning Algorithms for a Single Unit

Figure 5 shows an artificial neuron (unit). $\{x_1, \cdots, x_K\}$ are input values; $\{w_1, \cdots, w_K\}$ are weights; y is a scalar output; and f is the link function (also called activation/decision/transfer function).

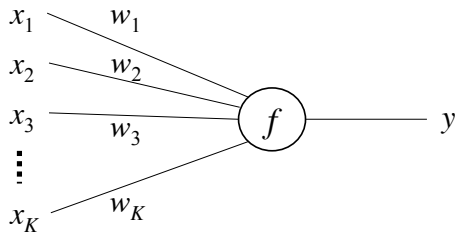

Figure 5: An artificial neuron

The unit works in the following way:

$$y = f(u), \tag{61}$$

where u is a scalar number, which is the net input (or "new input") of the neuron. u is defined as

$$u = \sum_{i=0}^{K} w_i x_i. \tag{62}$$

Using vector notation, we can write

$$u = \mathbf{w}^T \mathbf{x} \tag{63}$$

Note that here we ignore the bias term in u. To include a bias term, one can simply add an input dimension (e.g., x_0) that is constant 1.

Apparently, different link functions result in distinct behaviors of the neuron. We discuss two example choices of link functions here.

The first example choice of $f(u)$ is the **unit step function** (aka **Heaviside step function**):

$$f(u) = \begin{cases} 1 & \text{if } u > 0 \\ 0 & \text{otherwise} \end{cases} \tag{64}$$

A neuron with this link function is called a perceptron. The learning algorithm for a perceptron is the perceptron algorithm. Its update equation is defined as:

$$\mathbf{w}^{(\text{new})} = \mathbf{w}^{(\text{old})} - \eta \cdot (y - t) \cdot \mathbf{x} \tag{65}$$

where t is the label (gold standard) and η is the learning rate ($\eta > 0$). Note that a perceptron is a linear classifier, which means its description capacity can be very limited. If we want to fit more complex functions, we need to use a non-linear model.

The second example choice of $f(u)$ is the **logistic function** (a most common kind of **sigmoid function**), defined as

$$\sigma(u) = \frac{1}{1 + e^{-u}} \tag{66}$$

The logistic function has two primary good properties: (1) the output y is always between 0 and 1, and (2) unlike a unit step function, $\sigma(u)$ is smooth and differentiable, making the derivation of update equation very easy.

Note that $\sigma(u)$ also has the following two properties that can be very convenient and will be used in our subsequent derivations:

$$\sigma(-u) = 1 - \sigma(u) \tag{67}$$

$$\frac{d\sigma(u)}{du} = \sigma(u)\sigma(-u) \tag{68}$$

We use stochastic gradient descent as the learning algorithm of this model. In order to derive the update equation, we need to define the error function, i.e., the training objective. The following objective function seems to be convenient:

$$E = \frac{1}{2}(t - y)^2 \tag{69}$$

We take the derivative of E with regard to w_i,

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial u} \cdot \frac{\partial u}{\partial w_i} \tag{70}$$

$$= (y - t) \cdot y(1 - y) \cdot x_i \tag{71}$$

where $\frac{\partial y}{\partial u} = y(1 - y)$ because $y = f(u) = \sigma(u)$, and (67) and (68). Once we have the derivative, we can apply stochastic gradient descent:

$$\mathbf{w}^{(\text{new})} = \mathbf{w}^{(\text{old})} - \eta \cdot (y - t) \cdot y(1 - y) \cdot \mathbf{x}. \tag{72}$$

A.2 Back-propagation with Multi-Layer Network

Figure 6 shows a multi-layer neural network with an input layer $\{x_k\} = \{x_1, \cdots, x_K\}$, a hidden layer $\{h_i\} = \{h_1, \cdots, h_N\}$, and an output layer $\{y_j\} = \{y_1, \cdots, y_M\}$. For clarity we use k, i, j as the subscript for input, hidden, and output layer units respectively. We use u_i and u'_j to denote the net input of hidden layer units and output layer units respectively.

We want to derive the update equation for learning the weights w_{ki} between the input and hidden layers, and w'_{ij} between the hidden and output layers. We assume that all the

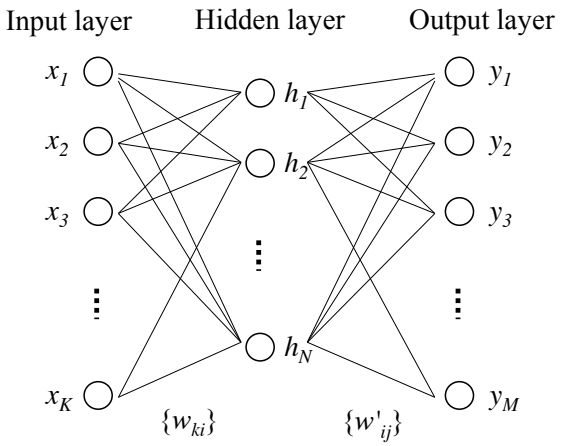

Figure 6: A multi-layer neural network with one hidden layer

computation units (i.e., units in the hidden layer and the output layer) use the logistic function $\sigma(u)$ as the link function. Therefore, for a unit h_i in the hidden layer, its output is defined as

$$h_i = \sigma(u_i) = \sigma\left(\sum_{k=1}^{K} w_{ki} x_k\right). \tag{73}$$

Similarly, for a unit y_j in the output layer, its output is defined as

$$y_j = \sigma(u'_j) = \sigma\left(\sum_{i=1}^{N} w'_{ij} h_i\right). \tag{74}$$

We use the squared sum error function given by

$$E(\mathbf{x}, \mathbf{t}, \mathbf{W}, \mathbf{W}') = \frac{1}{2}\sum_{j=1}^{M}(y_j - t_j)^2, \tag{75}$$

where $\mathbf{W} = \{w_{ki}\}$, a $K \times N$ weight matrix (input-hidden), and $\mathbf{W}' = \{w'_{ij}\}$, an $N \times M$ weight matrix (hidden-output). $\mathbf{t} = \{t_1, \cdots, t_M\}$, a M-dimension vector, which is the gold-standard labels of output.

To obtain the update equations for w_{ki} and w'_{ij}, we simply need to take the derivative of the error function E with regard to the weights respectively. To make the derivation straightforward, we do start computing the derivative for the right-most layer (i.e., the output layer), and then move left. For each layer, we split the computation into three steps, computing the derivative of the error with regard to the output, net input, and weight respectively. This process is shown below.

We start with the output layer. The first step is to compute the derivative of the error w.r.t. the output:

$$\frac{\partial E}{\partial y_j} = y_j - t_j. \tag{76}$$

The second step is to compute the derivative of the error with regard to the net input of the output layer. Note that when taking derivatives with regard to something, we need to keep everything else fixed. Also note that this value is very important because it will be reused multiple times in subsequent computations. We denote it as EI'_j for simplicity.

$$\frac{\partial E}{\partial u'_j} = \frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial u'_j} = (y_j - t_j) \cdot y_j(1 - y_j) := \mathrm{EI}'_j \tag{77}$$

The third step is to compute the derivative of the error with regard to the weight between the hidden layer and the output layer.

$$\frac{\partial E}{\partial w'_{ij}} = \frac{\partial E}{\partial u'_j} \cdot \frac{\partial u'_j}{\partial w'_{ij}} = \mathrm{EI}'_j \cdot h_i \tag{78}$$

So far, we have obtained the update equation for weights between the hidden layer and the output layer.

$$w'^{(\mathrm{new})}_{ij} = w'^{(\mathrm{old})}_{ij} - \eta \cdot \frac{\partial E}{\partial w'_{ij}} \tag{79}$$

$$= w'^{(\mathrm{old})}_{ij} - \eta \cdot \mathrm{EI}'_j \cdot h_i. \tag{80}$$

where $\eta > 0$ is the learning rate.

We can repeat the same three steps to obtain the update equation for weights of the previous layer, which is essentially the idea of back propagation.

We repeat the first step and compute the derivative of the error with regard to the output of the hidden layer. Note that the output of the hidden layer is related to all units in the output layer.

$$\frac{\partial E}{\partial h_i} = \sum_{j=1}^{M} \frac{\partial E}{\partial u'_j} \frac{\partial u'_j}{\partial h_i} = \sum_{j=1}^{M} \mathrm{EI}'_j \cdot w'_{ij}. \tag{81}$$

Then we repeat the second step above to compute the derivative of the error with regard to the net input of the hidden layer. This value is again very important, and we denote it as EI_i.

$$\frac{\partial E}{\partial u_i} = \frac{\partial E}{\partial h_i} \cdot \frac{\partial h_i}{\partial u_i} = \sum_{j=1}^{M} \mathrm{EI}'_j \cdot w'_{ij} \cdot h_i(1 - h_i) := \mathrm{EI}_i \tag{82}$$

Next we repeat the third step above to compute the derivative of the error with regard to the weights between the input layer and the hidden layer.

$$\frac{\partial E}{\partial w_{ki}} = \frac{\partial E}{\partial u_i} \cdot \frac{\partial u_i}{\partial w_{ki}} = \mathrm{EI}_i \cdot x_k, \tag{83}$$

Finally, we can obtain the update equation for weights between the input layer and the hidden layer.

$$w_{ki}^{(\text{new})} = w_{ki}^{(\text{old})} - \eta \cdot \text{EI}_i \cdot x_k. \tag{84}$$

From the above example, we can see that the intermediate results (EI'_j) when computing the derivatives for one layer can be reused for the previous layer. Imagine there were another layer prior to the input layer, then EI_i can also be reused to continue computing the chain of derivatives efficiently. Compare Equations (77) and (82), we may find that in (82), the factor $\sum_{j=1}^{M} \text{EI}'_j w'_{ij}$ is just like the "error" of the hidden layer unit h_i. We may interpret this term as the error "back-propagated" from the next layer, and this propagation may go back further if the network has more hidden layers.

B wevi: Word Embedding Visual Inspector

An interactive visual interface, wevi (word embedding visual inspector), is available online to demonstrate the working mechanism of the models described in this paper. See Figure 7 for a screenshot of wevi.

The demo allows the user to visually examine the movement of input vectors and output vectors as each training instance is consumed. The training process can be also run in batch mode (e.g., consuming 500 training instances in a row), which can reveal the emergence of patterns in the weight matrices and the corresponding word vectors. Principal component analysis (PCA) is employed to visualize the "high"-dimensional vectors in a 2D scatter plot. The demo supports both CBOW and skip-gram models.

After training the model, the user can manually *activate* one or multiple input-layer units, and inspect which hidden-layer units and output-layer units become active. The user can also customize training data, hidden layer size, and learning rate. Several preset training datasets are provided, which can generate different results that seem interesting, such as using a toy vocabulary to reproduce the famous word analogy: *king* - *queen* = *man* - *woman*.

It is hoped that by interacting with this demo one can quickly gain insights of the working mechanism of the model. The system is available at `http://bit.ly/wevi-online`. The source code is available at `http://github.com/ronxin/wevi`.

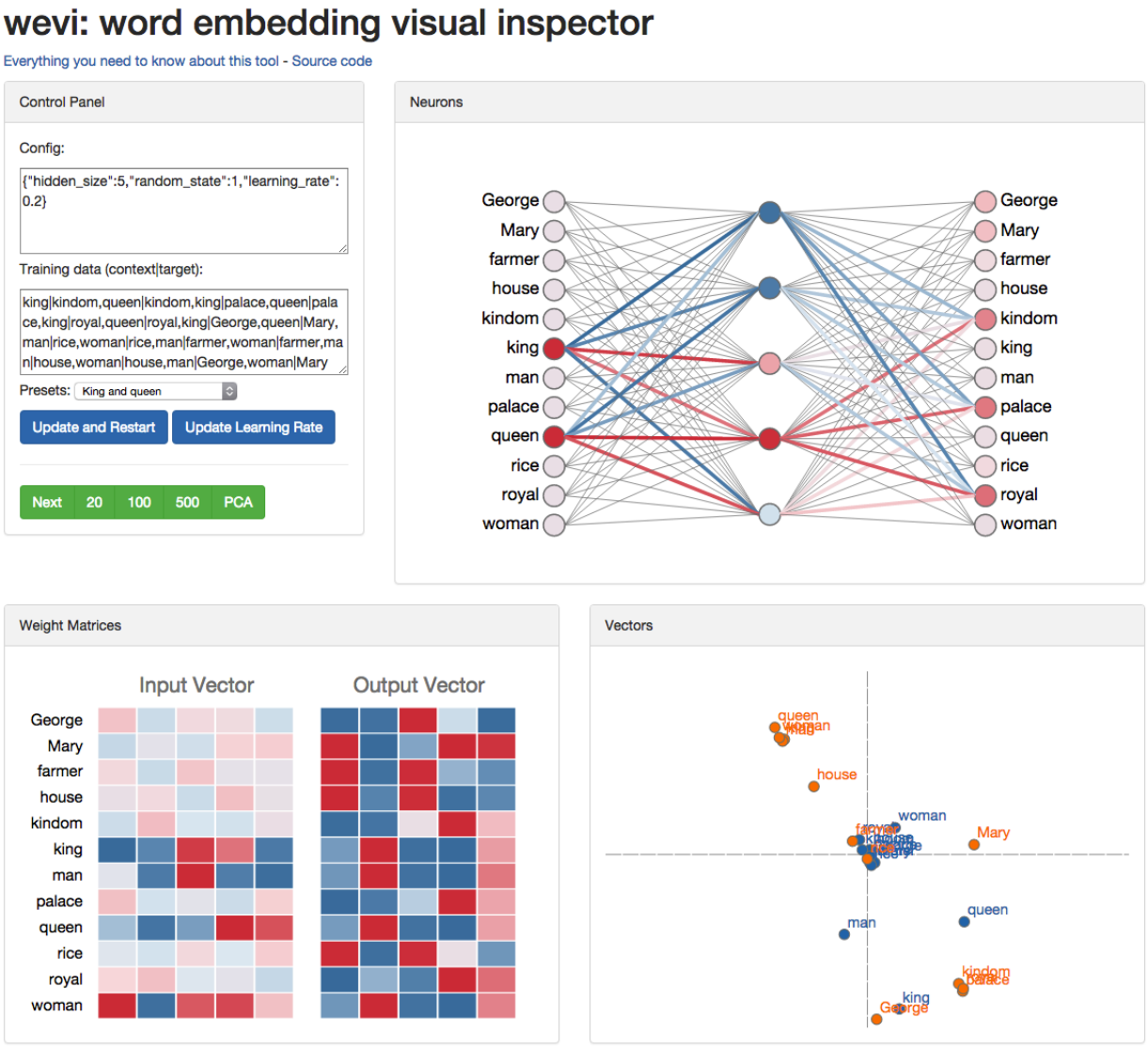

Figure 7: wevi screenshot (`http://bit.ly/wevi-online`)