# word2vec

Kristian Wichmann

May 26, 2017

## 1 A simple neural network

Consider a feed forward neural network with one hidden layer, as shown in figure 1. The input layer consists of a row[1] vector $x$ of dimension $D_x$, i.e. $x \in \mathbb{R}^{1 \times D_x}$. The hidden layer has $h$ neurons with a sigmoid activation function:

$$h = \sigma(xW^{(1)} + b^{(1)}) = \sigma(z^{(1)}), \quad z^{(1)} = xW^{(1)} + b^{(1)} \tag{1.1}$$

So $h \in \mathbb{R}^{1 \times H}, W^{(1)} \in \mathbb{R}^{D_x \times H}, b^{(1)} \in \mathbb{R}^{1 \times H}$. The output layer has $D_y$ softmax neurons:

$$\hat{y} = s(hW^{(2)} + b^{(2)}) = s(z^{(2)}), \quad z^{(2)} = hW^{(2)} + b^{(2)} \tag{1.2}$$

Similarly $\hat{y} \in \mathbb{R}^{1 \times D_y}, W^{(2)} \in \mathbb{R}^{H \times D_y}, b^{(2)} \in \mathbb{R}^{1 \times D_y}$.

[1]Note that this is different from the column vector convention usually used.

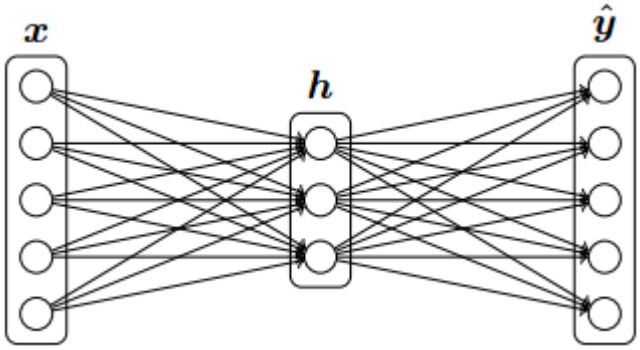


Figure 1: The neural network.

## 1.1 Error function

Assuming a labelled dataset $x$ with the correct label $c$ encoded as a one-hot vector $t \in \mathbb{R}^{1\times D_y}$, so $t_i = \delta_{ic}$. We will use the cross-entropy error function:

$$J(x) = -\sum_{i=1} t_i \log \hat{y}_i \tag{1.3}$$

Since $t$ is one-hot encoded, only the correct label $c$ will contribute to the sum, so:

$$J(x) = -\log \hat{y}_c \tag{1.4}$$

This does not mean that the other components of $\hat{y}$ will not matter, since the softmax indirectly depends on all components.

### 1.1.1 Derivative with respect to input

We now wish to compute the derivative of $J(x)$ with respect to $x$. In shorthand, the chain rule gives us:

$$\frac{\partial J}{\partial x} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial h} \frac{\partial h}{\partial z^{(1)}} \frac{\partial z^{(1)}}{\partial x} \tag{1.5}$$

Writing out indices explicitly:

$$\frac{\partial J}{\partial x_m} = \sum_{i=1}^{D_y} \sum_{j=1}^{D_y} \sum_{k=1}^{H} \sum_{l=1}^{H} \frac{\partial J}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial z_j^{(2)}} \frac{\partial z_j^{(2)}}{\partial h_k} \frac{\partial h_k}{\partial z_l^{(1)}} \frac{\partial z_l^{(1)}}{\partial x_m} \tag{1.6}$$

Let's compute these partial derivatives one by one:

$$\frac{\partial J}{\partial \hat{y}_i} = -\frac{\partial}{\partial \hat{y}_i} \log \hat{y}_c = -\frac{\delta_{ic}}{\hat{y}_c} \tag{1.7}$$

The second is a standard result for the softmax function:

$$\frac{\partial \hat{y}_i}{\partial z_j^{(2)}} = s_i(z^{(2)})(\delta_{ij} - s_j(z^{(2)}) = \hat{y}_i(\delta_{ij} - \hat{y}_j) \tag{1.8}$$

Let's pause for a moment and combine the two:

$$\frac{\partial J}{\partial z_j^{(2)}} = \sum_{i=1}^{D_y} \frac{\partial J}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial z_j^{(2)}} = -\sum_{i=1}^{D_y} \frac{\delta_{ic}}{\hat{y}_c} \hat{y}_i(\delta_{ij} - \hat{y}_j) \tag{1.9}$$

The Kronecker delta removes the sum and we get:

$$-\frac{1}{\hat{y}_c} \hat{y}_c(\delta_{jc} - \hat{y}_j) = -(\delta_{jc} - \hat{y}_j) = \hat{y}_j - \delta_{jc} \tag{1.10}$$

But $\delta_{jc} = t_j$, and we get that the derivative is simply equal to the *error* in the output layer:

$$\frac{\partial J}{\partial z_j^{(2)}} = \delta_j^{(2)} \tag{1.11}$$

Here, $\delta_j^{(2)} = \hat{y}_j - \delta_{jc}$. Going back to the original derivation, the third derivative is:

$$\frac{\partial z_j^{(2)}}{\partial h_k} = W_{kj}^{(2)} \tag{1.12}$$

The fourth uses a standard result for the sigmoid:

$$\frac{\partial h_k}{\partial z_l^{(1)}} = \delta_{kl}\sigma(z_l^{(1)})(1 - \sigma(z_l^{(1)})) = \delta_{kl}h_l(1 - h_l) \tag{1.13}$$

And finally:

$$\frac{\partial z_l^{(1)}}{\partial x_m} = W_{ml}^{(1)} \tag{1.14}$$

Inserting, letting the delta functions cancel, and renaming indices, this becomes:

$$\frac{\partial J}{\partial x_i} = \sum_{j=1}^{H}\sum_{k=1}^{D_y} W_{ij}^{(1)}h_j(1 - h_j)W_{jk}^{(2)}\delta_k^{(2)} \tag{1.15}$$

We may rephrase this in terms of the backpropagated errors in the hidden layer:

$$\delta_j^{(1)} = \sum_{k=1}^{D_y} h_j(1 - h_j)W_{jk}^{(2)}\delta_k^{(2)} \tag{1.16}$$

Now equation 1.15 can be rewritten as:

$$\frac{\partial J}{\partial x_i} = \sum_{j=1}^{H}\sum_{k=1}^{D_y} W_{ij}^{(1)}h_j(1 - h_j)W_{jk}^{(2)}\delta_k^{(2)} = \sum_{j=1}^{H} W_{ij}^{(1)}\delta_j^{(1)} \tag{1.17}$$

# 2 The word2vec algorithm

## 2.1 The situation

Imagine a large corpus of consecutive tokens (words) of length $T$. Each token comes from a vocabulary of size $W$. We wish to encode the tokens as vectors. To do this, we look at the words surrounding a given word in the corpus probabilistically. If $c$ is the center word, we can consider the

probability of a word $j$ places away being $o$. We should really denote this probability $p_j(o|c)$, but since we're considering skipgrams, we will not worry about the value of $j$ as long as it is within the specified distance. So we will simply call it $p(o|c)$.

## 2.2 Softmax and word vectors

If we have each word represented by vectors, an appropriate model for the probability could be a softmax over the entire vocabulary size:

$$p(o|c) = \frac{\exp(u_o^t v_c)}{\sum_{w=1}^{W} \exp(u_w^t v_c)} \tag{2.1}$$

Note that each word $w$ has two vectors: $u_w$ and $v_w$. $u_w$ is for $w$ as an 'outside' word, and $v_w$ is for $w$ as a 'center' word.

## 2.3 Error function and derivatives

An appropriate error function for the softmax is the cross-entropy error. The situation is entirely analogous to the propagation from the hidden layer to the output layer in the neural network described in last section. I.e. it is described by equations 1.9 and 1.11 with the following replacements:

- $j \mapsto o$
- $z_j^{(2)} \mapsto z_o = u_0^t v_c$
- $i \mapsto w$
- $\hat{y}_i \mapsto p(o|c)$

So, the derivative with respect to $v_c$ is:

$$\frac{\partial J}{\partial v_c} = \sum_{o=1}^{W} \frac{\partial J}{\partial z_0} \frac{\partial z_0}{\partial v_c} = \sum_{o=1}^{W} \delta_o^{(2)} u_o \tag{2.2}$$

Remember, that the error is a scalar, while $u_o$ is a vector:

$$u_0 = \begin{pmatrix} u_{o1} \\ \vdots \\ u_{oW} \end{pmatrix} \tag{2.3}$$