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).

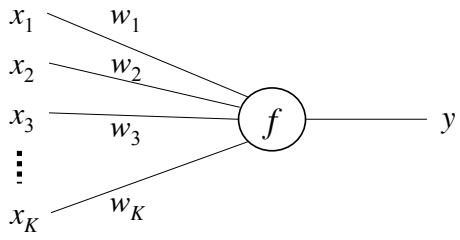

Figure 5: An artificial neuron

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

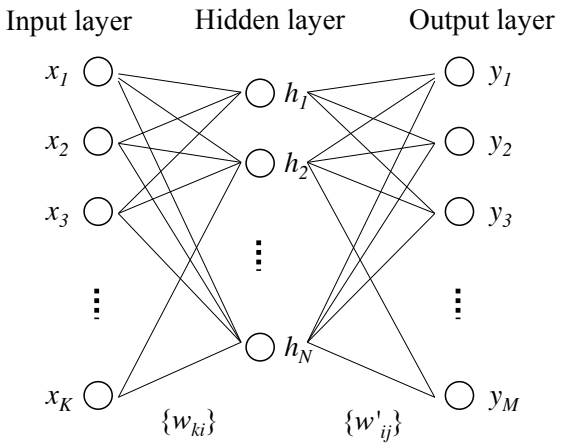

Figure 6: A multi-layer neural network with one hidden layer

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`.

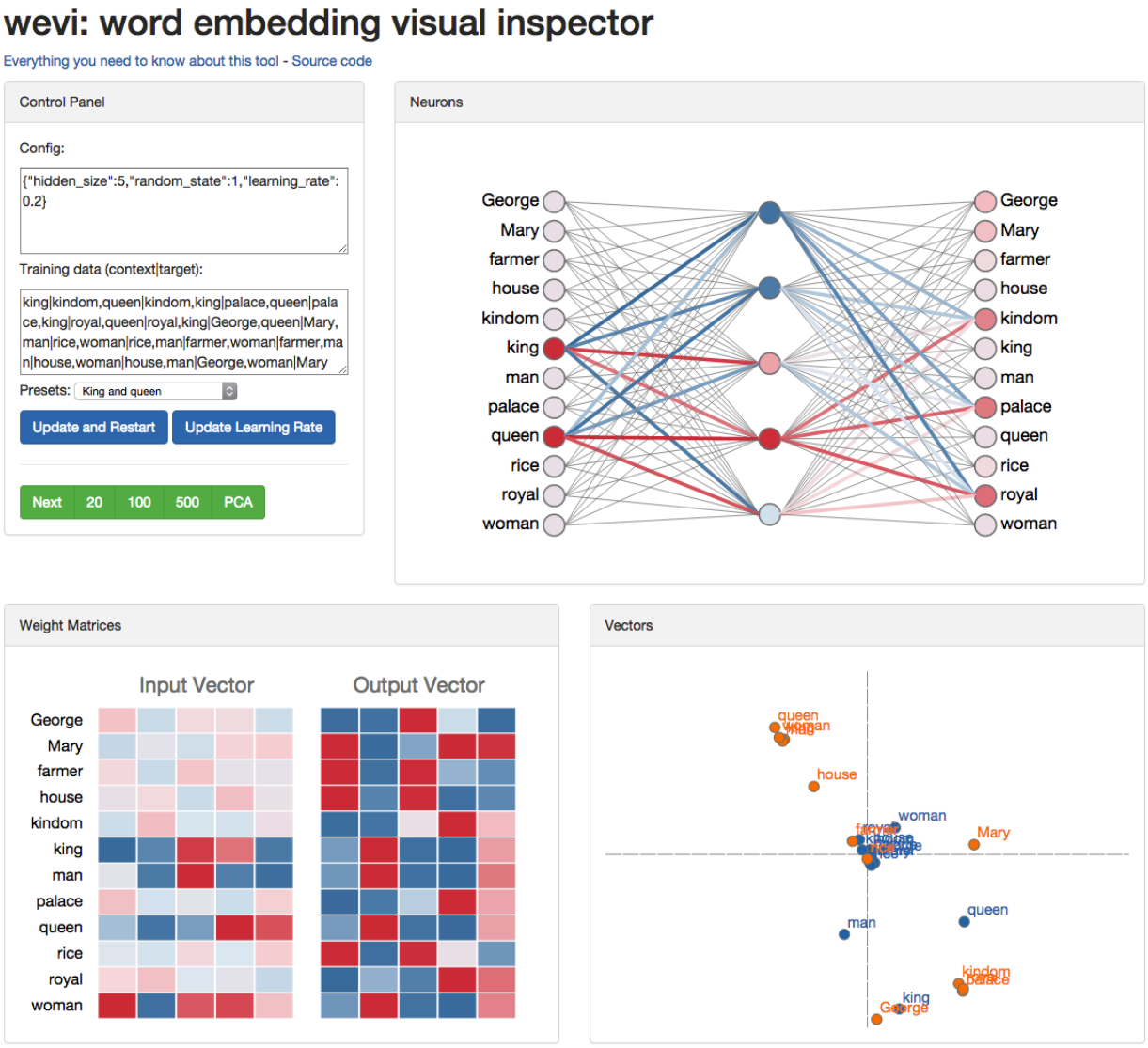

Figure 7: wevi screenshot (`http://bit.ly/wevi-online`)