Outline of the algorithm

Us
Department of Statistics
University of Oxford
and
Department of Statistics
University of Warwick

January 23, 2016

1 Introduction to the algorithm

1.1 Notation

- The data are denoted by $\{(x_i, y_i)\}_{i=1}^{n}$, where $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{Z}_{0+}$ or $y \in \mathbb{R}$, depending on problem at hand.
- **Layers** of the Neural Network run through: $l = 0, 1, \ldots, L$. Layer (0) is an *artificial* layer whose output are just the input data $x_i = x_i^{(0)}$. Layer (L) is an output layer,whose output are the fitted values/predicted class probabilities. Layers $l = 1, \ldots, L-1$ are called the hidden layers.
- $w_{i,j}^{(l)}$ - **weight** for the connection between the i^{th} unit in layer $(l+1)$ with the j^{th} unit in layer (l).
- $b_i^{(l)}$ - **bias** for the i^{th} unit in layer (l).
- $m^{(l)}$ - number of units in layer (l). In general, for our setup we will have $2m$ units in hidden layer, but the weights will be shared among unit pairs, numbered $2i$ and $2i+1$, for $i = 0, \ldots m-1$.

- $\sigma(\cdot)$, $s(\cdot)$, $c(\cdot)$ - generic, sine and cosine **activation functions** respectively.
- $x_i^{(l)} = \sigma(x_i^{(l-1)T} w_i^{(l-1)} + b_i^{(l)})$ - **activation** i.e. output of the i^{th} unit in the $(l)^{th}$ layer.
- $z_i^{(l)} = x_i^{(l-1)T} w_i^{(l-1)} + b_i^{(l)}$ - **score** of the i^{th} unit in the $(l)^{th}$ layer.
- J - **loss** function.

1.2 Algorithm

The following steps will be performed for each data point (x_i, y_i):
First, feed forward x_i's through our network to produce $z_{(l)}$ and $x_{(l)}$, for $l = 1, \ldots, L$. Then backpropagate the information from the loss function. To do this we need to know the quantities: $\frac{\partial J}{\partial w_{ij}^{(l)}}$ and $\frac{\partial J}{\partial b_i^{(l)}}$, for all i, j, l. We calculate them separately for the last layer and other layers. Consider the final layer (L) first.

$$\frac{\partial J}{\partial w_{ij}^{(L-1)}} = \frac{\partial J}{\partial z_i^{(L)}} \frac{\partial z_i^{(L)}}{\partial w_{ij}^{(L-1)}} = \frac{\partial J}{\partial z_i^{(L)}} (x^{(L-1)})_j := \delta_i^{(L)} (x^{(L-1)})_j, \tag{1}$$

so if we could calculate a vector $\delta^{(L)}$, whose i^{th} entry is $\frac{\partial J}{\partial z_i^{(L)}}$ (and we can), then we can concisely write:

$$\frac{\partial J}{\partial w^{(L-1)}} = \delta^{(L)} x^{(L-1)T}. \tag{2}$$

In python we would just have:

```python
import numpy as np

...

# Assumes we have access to an array/tuple of
# numpy arrays:'deltas' and activations. The latter
# is obtained from feed forward alg., and it will
# be explained how to derive the former.

w_grad = np.dot(deltas[-1], activations[-2].transpose())
```

There is no term $\frac{\partial J}{\partial b_i^{(L)}}$, as there is no bias in the final layer. Updating the hidden layers is only slightly more complicated (and it is complicated still a bit further due to the dual nature of our units) and update of the first hidden layer is again slightly different due to presence of a bias term. Thus, for $l = L-1, \ldots, 1$:

$$\begin{aligned} \delta_i^{(l)} :=& \frac{\partial J}{\partial z_i^{(l)}} = \sum_k \frac{\partial J}{\partial z_k^{(l+1)}} \left(\frac{\partial z_k^{(l+1)}}{\partial x_i^{(l)}} \frac{\partial x_i^{(l)}}{\partial z_i^{(l)}} + \frac{\partial z_k^{(l+1)}}{\partial x_{(i+m)}^{(l)}} \frac{\partial x_{(i+m)}^{(l)}}{\partial z_i^{(l)}} \right) = \\ =& \left(\sum_k \delta_k^{(l+1)} (w_{ki}^{(l)} + w_{k(i+m)}^{(l)}) \right) \sigma'(z_i^{(l)}), \end{aligned} \tag{3}$$

(notice that we assume each unit i, $i = 1, \ldots, m$ produces scores with indices i and $i+m$) which gives:

$$\frac{\partial J}{\partial w_{ij}^{(l-1)}} = \frac{\partial J}{\partial z_i^{(l)}} \frac{\partial z_i^{(l)}}{\partial w_{ij}^{(l-1)}} = \delta_i^{(l)} (x^{(l-1)})_j = \left(\sum_k \delta_k^{(l+1)} (w_{ki}^{(l)} + w_{k(i+m)}^{(l)}) \right) \sigma'(z_i^{(l)}) (x^{(l-1)})_j, \tag{4}$$

and this can be written concisely as:

$$\frac{\partial J}{\partial w^{(l-1)}} = \delta^{(l)} x^{(l-1)T} = \left(\left((w_1^{(l)T} + w_2^{(l)T}) \delta^{(l+1)} \right) \cdot \sigma'(z^{(l)}) \right) x^{(l-1)T}. \tag{5}$$

where '$\cdot$' is an element-wise multiplication and $w_1^{(l)T}$ and $w_2^{(l)T}$ are top and bottom halves respectively of $w^{(l)T}$. In python we would write:

```
midpoint = m
temp = np.dot(weight[l], deltas[l+1]) * \
    np.append(sigma_1_prime(activation[l]),
              sigma_2_prime(activation[l]))
deltas[l] = temp[:midpoint] + temp[midpoint:]
w_grad = np.dot(deltas[l], activations[l-1].transpose())
```

The bias terms are still not present until the first hidden layer. It is easily seen that:

$$\frac{\partial J}{\partial b^{(1)}} = \delta^{(2)} \tag{6}$$