1 Forward propagation

Recall that given input $x$, we define $a^{[0]} = x$. Then for layer $\ell = 1, 2, \ldots, N$, where $N$ is the number of layers of the network, we have

1. $z^{[\ell]} = W^{[\ell]}a^{[\ell-1]} + b^{[\ell]}$
2. $a^{[\ell]} = g^{[\ell]}(z^{[\ell]})$

In these notes we assume the nonlinearities $g^{[\ell]}$ are the same for all layers besides layer $N$. This is because in the output layer we may be doing regression [hence we might use $g(x) = x$] or binary classification [$g(x) = \text{sigmoid}(x)$] or multiclass classification [$g(x) = \text{softmax}(x)$]. Hence we distinguish $g^{[N]}$ from $g$, and assume $g$ is used for all layers besides layer $N$.

Finally, given the output of the network $a^{[N]}$, which we will more simply denote as $\hat{y}$, we measure the loss $J(W, b) = \mathcal{L}(a^{[N]}, y) = \mathcal{L}(\hat{y}, y)$. For example, for real-valued regression we might use the squared loss

$$\mathcal{L}(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$$

and for binary classification using logistic regression we use

$$\mathcal{L}(\hat{y}, y) = -(y \log \hat{y} + (1 - y) \log(1 - \hat{y}))$$

or negative log-likelihood. Finally, for softmax regression over $k$ classes, we use the cross entropy loss

$$\mathcal{L}(\hat{y}, y) = -\sum_{j=1}^{k} 1\{y = j\}\log \hat{y}_j$$

which is simply negative log-likelihood extended to the multiclass setting. Note that $\hat{y}$ is a $k$-dimensional vector in this case. If we use $y$ to instead denote the $k$-dimensional vector of zeros with a single 1 at the $l$th position, where the true label is $l$, we can also express the cross entropy loss as

$$\mathcal{L}(\hat{y}, y) = -\sum_{j=1}^{k} y_j \log \hat{y}_j$$
2 Backpropagation

Let’s define one more piece of notation that’ll be useful for backpropagation.\(^1\)

We will define  
\[
\delta^{[\ell]} = \nabla_{z^{[\ell]}} \mathcal{L}(\hat{y}, y)
\]

We can then define a three-step “recipe” for computing the gradients with respect to every  
\[W^{[\ell]}, b^{[\ell]}\]  
as follows:

1. For output layer  \(N\), we have  
   \[
   \delta^{[N]} = \nabla_{z^{[N]}} \mathcal{L}(\hat{y}, y)
   \]

   Sometimes we may want to compute  \(\nabla_{z^{[N]}} \mathcal{L}(\hat{y}, y)\) directly (e.g. if  \(g^{[N]}\)  
is the softmax function), whereas other times (e.g. when  \(g^{[N]}\) is the sigmoid function  \(\sigma\)) we can apply the chain rule:
   \[
   \nabla_{z^{[N]}} \mathcal{L}(\hat{y}, y) = \nabla_{\hat{y}} \mathcal{L}(\hat{y}, y) \circ (g^{[N]})'(z^{[N]})
   \]

   Note \((g^{[N]})'(z^{[N]})\) denotes the elementwise derivative w.r.t.  \(z^{[N]}\).

2. For  \(\ell = N-1, N-2, \ldots, 1\), we have  
   \[
   \delta^{[\ell]} = (W^{[\ell+1]\top} \delta^{[\ell+1]}) \circ g'(z^{[\ell]})
   \]

3. Finally, we can compute the gradients for layer  \(\ell\) as  
   \[
   \nabla_{W^{[\ell]}} J(W, b) = \delta^{[\ell] a^{[\ell-1]\top}}
   \]
   \[
   \nabla_{b^{[\ell]}} J(W, b) = \delta^{[\ell]}
   \]

where we use \(\circ\) to indicate the elementwise product. Note the above procedure is for a single training example.

You can try applying the above algorithm to logistic regression (\(N = 1\),  \(g^{[1]}\) is the sigmoid function  \(\sigma\)) to sanity check steps (1) and (3). Recall that  \(\sigma'(z) = \sigma(z) \circ (1 - \sigma(z))\) and  \(\sigma'(z^{[1]})\) is simply  \(a^{[1]}\). Note that for logistic regression, if  \(x\) is a column vector in  \(\mathbb{R}^{n \times 1}\), then  \(W^{[1]} \in \mathbb{R}^{1 \times n}\), and hence  \(\nabla_{W^{[1]}} J(W, b) \in \mathbb{R}^{1 \times n}\). Example code for two layers is also given at:

http://cs229.stanford.edu/notes/backprop.py

\(^1\)These notes are closely adapted from:
http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/
Scribe: Ziang Xie