CS 189: Introduction to Machine Learning - Discussion 11

This discussion assumes the knowledge of backpropagation, and builds upon the fundamental concepts we discussed in last section. Notations are similar to notations used last time.

The network is of 1, . . . ,L layers, that is, the input data passes through L groups of neurons to produce an output. Since we want to allow varying number of neurons in each group, we denote the number of neurons in each layer by $d^{(l)}$. Consider the l^{th} layer. It's input $x^{(l-1)}$ comes from the previous layer therefore it can be represented by a vector of length equal to the number of neurons in the layer $l-1$ i.e. $d^{(l-1)}$. As we discussed, back propagation uses chain rule to pass the gradient backwards. Let $\mathcal{L}$ be the final loss. For layer l, let $\delta^{(l)}$ be the gradient that this layer passes backwards, thus it is

$$\begin{aligned}\delta^{(l)} &= \frac{\partial \mathcal{L}}{\partial x^{(l-1)}} \\ &= \left(\frac{\partial x^{(l)}}{\partial x^{(l-1)}}\right) \delta^{(l+1)}\end{aligned}$$

1. **Initialization of weights for back propagation**

   Recall that back propagation is simply a clever method to solve for the gradient of the loss function so we can use it in a numerical optimization method such as gradient descent. Methods like these require the parameters to be initialized to some value. In the case of logistic regression, we were able to set the weights to be 0. Assume a fully connected 1-hidden layer network with K output nodes. For notation, let us assume that $x_j^{(l)} = g(S_j^{(l)})$, and $S_j^{(l)} = \sum_i^{d^{(l-1)}} w_{ij}^{(l)} x_i^{(l-1)}$, where g is the nonlinearity.

   (a) Imagine that we initialize the values of our weights to be some constant w. After performing the forward pass, what is the relation between the members of the set $\{x_j^{(1)} : j = 1, \ldots, d^{(1)}\}$ and $\{x_i^{(0)} : i = 1, \ldots, d^{(0)}\}$?

   Solution: Since all of the weights are equal, we have

   $$x_j^{(1)} = g\left(\sum_i^{d^{(0)}} w x_i^{(0)}\right) = g\left(w \sum_i^{d^{(0)}} x_i^{(0)}\right)$$

Note that this equation does not depend on j, thus, all the members of the set are equal. Also note that the $x_i^{(0)}$ are just the inputs.

(b) After the backwards pass of back propagation, what is the relation between the members of the set $\{\delta_j^{(1)} : j = 1, \ldots, d^{(0)}\}$ and $\{\delta_k^{(2)} : k = 1, \ldots, d^{(1)}\}$?

Solution:

$$x_j^{(1)} = g\left(\sum_i^{d^{(0)}} w x_i^{(0)}\right)$$

Again, since all of the weights are equal,

$$\begin{aligned}
\delta_i^{(1)} &= \frac{\partial \mathcal{L}}{\partial x_i^{(0)}} \\
&= \frac{\partial \mathcal{L}}{\partial x^{(1)}} \frac{\partial x^{(1)}}{\partial x_i^{(0)}} = \delta^{(2)} \frac{\partial x^{(1)}}{\partial x_i^{(0)}} \\
&= \delta^{(2)} g'\left(\sum_i^{d^{(0)}} w x_i^{(0)}\right) [w, w \ldots, w]^T \\
&= g'\left(\sum_i^{d^{(0)}} w x_i^{(0)}\right) \sum_k^{d^{(1)}} \delta_k^{(2)} w \qquad\qquad = g'\left(S_j^{(1)}\right) \sum_k^{d^{(1)}} \delta_k^{(2)} w
\end{aligned}$$

The sum term does not depend on i and is the same for all values of i in layer l. The $S_j^{(1)}$ doesn't depend on j and are equal for all values of j. The members of the set are equal.

(c) After the weights are updated and one iteration of gradient descent has been completed, what can we say about the weights?

Solution: Our gradient descent update looks like this:

$$\begin{aligned}
w_{ij}^{(l)} &= w_{ij}^{(l)} - \eta\left(g'\left(S_j^{(l)}\right) x_i^{(l-1)}\right) \delta_j^{(l+1)} \\
&= w - \eta\left(g'\left(S_j^{(l)}\right) x_i^{(l-1)}\right) \delta_j^{(l+1)}
\end{aligned}$$

One can see that all $w_{ij}^{(1)}$ will be the same for all values of j. Also, $w_{ij}^{(2)}$ will be the same for all values of i because The $S_j^{(1)}$ doesn't depend on j and are

equal for all values of j. This pattern will continue and will not break for as many iterations you do. This is because the $x_j^{(1)}$ will always be the same for all j, due to the "outgoing" symmetry in the weights in layer 1, and the $d_j^{(1)}$ will always be the same due to the "incoming" symmetry in the weights in layer 2.

(d) To solve this problem, we randomly initialize our weights. This is called symmetry breaking. Why are we able to set our weights to 0 for logistic regression?

Solution: Logistic regression can be viewed as a neural network with one output node and zero hidden layers. In logistic regression, the loss function is convex. Any starting point should lead us to the global optimum.

2. Modifying neural networks for fun and profit

   (a) How could we modify a neural network to perform regression instead of classification?

   Solution: Change the output function of the final layer to be a linear function rather than the normal non-linear function.

   Consider a neural network with the addition that the input layer is also fully connected to the output layer. This type of neural network is also called "skip-layer".

   (b) How many weights would this model require? (Let d_0 be the dimensionality of the input vector, and $d_1 \dots d_L$ be the number of nodes in the L following layers. Don't worry about the bias term. Also, you may want to try drawing out the NN.)

   Solution:

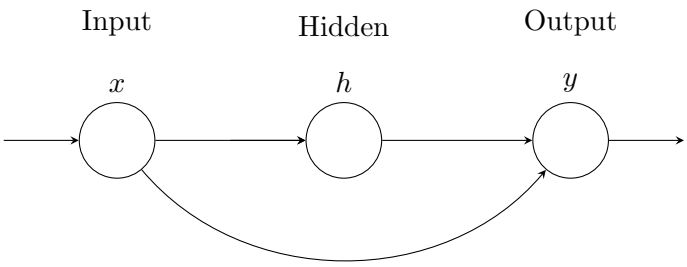
   $$\sum_{i=0}^{L-1} d_i d_{i+1} + d_0 d_L$$

   (c) What sort of problems may this sort of neural network introduce? How do we compensate for these problems?

   Solution: If skip layer has a lot of parameters, then we might have increased the number of weights (parameters). So now there may be a higher chance of overfitting the data. We could fix this by reducing the number of nodes at each layer and/or reducing the number of layers to accomodate for the increased connectivity.

   One potential advantage of the skip layers in the cases when network contains lot of layers. In this setting, higher layers tend to learn higher level concepts and normally output would connect just to these higher layers. The presence of skip layers ensures that output gets direct access to both low-level concepts (initial layers) and high level concepts (later layers). This can be a better modeling choice in many settings.

   (d) Consider the simplest skip-layer neural network pictured below. The weights are $w = [w_{xh}, w_{hy}, w_{xy}]^T$.



Given some non-linear function g, calculate $\nabla_w y$.

Solution: The output y is given by the function:

$$y = g(s_y) = g(w_{hy}h + w_{xy}x) = g(w_{hy}g(s_h) + w_{xy}x) = g(w_{hy}g(w_{xh}x) + w_{xy}x)$$

To calculate ∇y we need all the partial derivatives $\frac{\partial y}{\partial w_h}$, $\frac{\partial y}{\partial w_{hy}}$, $\frac{\partial y}{\partial w_{xy}}$. We'll start with the ones closest to the output.

$$\frac{\partial y}{\partial w_{hy}} = \frac{\partial y}{\partial s_y} \cdot \frac{\partial s_y}{\partial w_{hy}} = g'(s_y)h = \delta_y h$$

$$\frac{\partial y}{\partial w_{xy}} = \frac{\partial y}{\partial s_y} \cdot \frac{\partial s_y}{\partial w_{xy}} = g'(s_y)x = \delta_y x$$

$$\frac{\partial y}{\partial w_{xh}} = \frac{\partial y}{\partial s_y} \cdot \frac{\partial s_y}{\partial w_{xh}} = g'(s_y)\frac{\partial}{\partial w_{xh}}(w_{hy}h + w_{xy}x)$$

$$= g'(s_y)(\frac{\partial}{\partial s_h}(w_{hy}g(s_h))\frac{\partial s_h}{\partial w_{xh}} + \frac{\partial}{\partial w_{xh}}w_{xy}x)$$

$$= w_{hy}g'(s_y)g'(s_h)\frac{\partial s_h}{\partial w_{xh}} + 0$$

$$= w_{hy}g'(s_y)g'(s_h)\frac{\partial(w_{xh}x)}{\partial w_{xh}}$$

$$= w_{hy}g'(s_y)g'(s_h)x = w_{hy}\delta_y g'(s_h)x$$