Week 5: Neural Networks

Instructor: Sergey Levine

1 Neural Networks Summary

In the previous lecture, we saw how we can construct neural networks by extending logistic regression. Neural networks consist of multiple layers of computation. At each layer, we have a vector values called *activations*, which we'll denote $h^{(\ell)}$ for the activations at layer ℓ. By convention, we'll say that $h^{(0)} = \mathbf{x}$ (the input) and for the last layer L, we have $h^{(L)}$ be the output, which could be the probability of a label (we'll discuss regression and other types of outputs later). We can express the activations in the neural net recursively as

$$h^{(\ell)} = \sigma(\mathbf{W}^{(\ell)} h^{(\ell-1)} + \mathbf{b}^{(\ell)}),$$

where $\sigma(z)$ is a nonlinearity. Last lecture, we saw a common choice of nonlinearity, the sigmoid (or logistic function):

$$\sigma(z) = \frac{1}{1 + \exp(-z)}.$$

In a neural network, we must choose the number of layers L and the size of each layer (the size of $h^{(0)}$ is the number of input attributes and the size of $h^{(L)}$ is the number of outputs – more on that later). If we have a network with three layers of weights (corresponding to two hidden layers), we can explicitly write it as:

$$h^{(3)} = \sigma(\mathbf{W}^{(3)} \sigma(\mathbf{W}^{(2)} \sigma(\mathbf{W}^{(1)} h^{(0)} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}) + \mathbf{b}^{(3)}).$$

Note that there is a little bit of confusion in terminology: this network has three "layers" of weights, given by $\mathbf{W}^{(1)}$, $\mathbf{W}^{(2)}$, and $\mathbf{W}^{(3)}$, but two "layers" of hidden activations $h^{(1)}$ and $h^{(2)}$, since $h^{(0)}$ is simply $\mathbf{x}$ (and therefore not "hidden"), and $h^{(3)}$ is the output. Traditionally, "hidden layer" in a neural network refers to a layer of hidden activations (or "neurons"), but more recent terminology typically refers to the weights as layers. They are off by one, so don't get confused!

Now, let's check our understanding:

Question. What is the data?

Answer. Just like in logistic regression, we have an input vector $\mathbf{x}$, which can be either continuous real-valued, binary, or categorical. Just like with logistic regression, categorical values are often converted into "one-hot" encodings: if some feature takes on M values, we might add M entries to $\mathbf{x}$, where all but one is zero. This avoids the need to impose an ordering on that variable. The output y for now is categorical (just like in logistic regression), though we'll discuss how we can have real-valued y as well later.

Question. What defines (parameterizes) the hypothesis?

Answer. To define a neural network, we have to first choose the number and size of the layers. This is a design decision (so technically, number and size of layers are hyperparameters). We choose these the same way we choose the features $h(\mathbf{x})$ to use for logistic regression, the amount of regularization, etc: we use our intuition and, when in doubt, validate against the validation set. Once we've figured out the number of layers and their size, the neural network is fully defined by the weights $\theta = \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \ldots, \mathbf{W}^{(L)}, \mathbf{b}^{(L)}\}$. It is precisely these weights that our learning algorithm needs to optimize.

Question. What is the objective?

Answer. Neural networks are conditional (or "discriminative") models. Just like logistic regression and linear regression, neural networks optimize the conditional log-likelihood, given by

$$\mathcal{L}(\theta) = \sum_{i=1}^{N} \log p(y^i | \mathbf{x}^i, \theta)$$

In the case of a binary label y and a three-layer network (two hidden layers), this is given by

$$\mathcal{L}(\theta) = \sum_{i=1}^{N} \begin{cases} \text{if } y^i = 0 : 1 - \sigma(\mathbf{W}^{(3)}\sigma(\mathbf{W}^{(2)}\sigma(\mathbf{W}^{(1)}\mathbf{x}^i + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}) + \mathbf{b}^{(3)}) \\ \text{if } y^i = 1 : \sigma(\mathbf{W}^{(3)}\sigma(\mathbf{W}^{(2)}\sigma(\mathbf{W}^{(1)}\mathbf{x}^i + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}) + \mathbf{b}^{(3)}) \end{cases}$$

Question. What is the algorithm?

Answer. Just like with logistic regression, we'll use gradient ascent to optimize the neural network parameters θ. However, we have many more parameters now, and computing the gradient becomes a lot more difficult. Furthermore, unlike with logistic regression, the neural network objective is not convex, because the weights $\mathbf{W}^{(\ell)}$ and $\mathbf{b}^{(\ell)}$ have complex nonlinear effects on the output. That means that gradient ascent can get stuck in local optima when optimizing neural networks, and we cannot in general guarantee a globally optimal solution. In practice, especially for smaller networks, it's often not hard to find "good

enough" solutions with gradient ascent that are not globally optimal but still perform well. For very large and deep networks, we often have to think about more sophisticated optimization algorithms (more on this later). To summarize, the gradient ascent algorithm simply consists of repeatedly applying the following operation:

$$\theta^{(j+1)} \leftarrow \theta^{(j)} + \alpha \nabla \mathcal{L}(\theta^{(j)}).$$

Note that $\nabla \mathcal{L}(\theta^{(j)})$ here is a huge vector that consists of the concatenation of the gradient with respect to each weight matrix and bias vector. Assume that each weight matrix $\mathbf{W}^{(\ell)}$ has M_ℓ rows and $M_{\ell-1}$ columns, then $\nabla \mathcal{L}(\theta)$ can be written as:

$$\nabla \mathcal{L}(\theta) = \begin{bmatrix} \frac{d\mathcal{L}}{d\mathbf{W}^{(1)}_{1,1}} \\ \frac{d\mathcal{L}}{d\mathbf{W}^{(1)}_{2,1}} \\ \cdots \\ \frac{d\mathcal{L}}{d\mathbf{W}^{(1)}_{M_1,1}} \\ \frac{d\mathcal{L}}{d\mathbf{W}^{(1)}_{1,2}} \\ \cdots \\ \frac{d\mathcal{L}}{d\mathbf{W}^{(1)}_{M_1,2}} \\ \cdots \\ \frac{d\mathcal{L}}{d\mathbf{W}^{(1)}_{M_1,M_0}} \\ \frac{d\mathcal{L}}{d\mathbf{b}^{(1)}_{1}} \\ \cdots \\ \frac{d\mathcal{L}}{d\mathbf{b}^{(1)}_{M_1}} \\ \frac{d\mathcal{L}}{d\mathbf{W}^{(2)}_{1,1}} \\ \cdots \\ \frac{d\mathcal{L}}{d\mathbf{W}^{(2)}_{M_2,1}} \\ \cdots \\ \frac{d\mathcal{L}}{d\mathbf{W}^{(2)}_{M_2,M_1}} \\ \frac{d\mathcal{L}}{d\mathbf{b}^{(2)}_{1}} \\ \cdots \\ \frac{d\mathcal{L}}{d\mathbf{b}^{(2)}_{M_2}} \\ \cdots \\ \frac{d\mathcal{L}}{d\mathbf{W}^{(L)}_{M_L,M_{L-1}}} \\ \cdots \\ \frac{d\mathcal{L}}{d\mathbf{b}^{(L)}_{M_L}} \end{bmatrix}$$

In practice, it can often be more convenient when implementing gradient ascent for neural networks to simply compute the gradient with respect to each matrix $\mathbf{W}^{(\ell)}$ and vector $\mathbf{b}^{(\ell)}$ and increment them individually according to the gradient

ascent rule (which has exactly the same effect). For example, in an object-oriented framework, each matrix and vector can be its own object that "knows" how to compute its own gradient and apply gradient ascent, or else concatenate its gradient to the huge full gradient vector. In the next section, we'll discuss how we can compute the gradient with respect to each weight matrix $\mathbf{W}^{(\ell)}$.

2 Backpropagation: base case

To evaluate the output of a neural network, we recursively evaluate each layer as following:

$$h^{(\ell)} = \sigma(\mathbf{W}^{(\ell)} h^{(\ell-1)} + \mathbf{b}^{(\ell)}).$$

We will also introduce $z^{(\ell)} = \mathbf{W}^{(\ell)} h^{(\ell-1)} + \mathbf{b}^{(\ell)}$, such that $h^{(\ell)} = \sigma(z^{(\ell)})$. Sometimes, you'll see h referred to as "post-synaptic" and z as "pre-synaptic."

This recursive evaluation of a neural network is referred to as "forward propagation," because we are "propagating" the activations from the input "forward" to the output. To compute derivatives, we use the chain rule to differentiate each layer of the neural network with respect to the objective. This algorithm proceeds from the end of the neural network back down to the first layer, and is therefore referred to as "backward propagation" or "backpropagation."

First, let's revisit the chain rule. If we have the composition of two functions $f(g(x))$, and we want $\frac{df}{dx}$, we can evaluate it as:

$$\frac{df}{dx} = \frac{df}{dg}\frac{dg}{dx}.$$

For example, if $f(y) = ay$, and $g(x) = bx$, then $\frac{df}{dx} = ab$. The same idea applies in the multivariate case. Let's say that we have a vector $\mathbf{x}$, and $f(\mathbf{y}) = \mathbf{A}\mathbf{y}$, and $g(\mathbf{x}) = \mathbf{B}\mathbf{x}$. Then

$$\frac{df}{d\mathbf{x}} = \frac{df}{dg}\frac{dg}{d\mathbf{x}} = \mathbf{A}\mathbf{B} \Rightarrow \left(\frac{df}{d\mathbf{x}}\right)_{i,j} = \sum_k \mathbf{A}_{i,k}\mathbf{B}_{k,j}.$$

Next, let's see how we can compute the gradient for a single-layer neural network, which simply corresponds to logistic regression. Although the output of this network is 1D in the binary case, we'll still do the math for the multivariate case, assuming there are M_1 outputs (it just happens that $M_1 = 1$). This will make things more convenient later. The likelihood is given by

$$\mathcal{L}(\theta) = \sum_{i=1}^{N} \log p(y^i | \mathbf{x}^i, \mathbf{W}^{(1)}, \mathbf{b}^{(1)}).$$

In the binary case, we simply have $\log p(y|\mathbf{x}, \mathbf{W}^{(1)}, \mathbf{b}^{(1)}) = \log(h^{(1)})$ if $y = 1$ and $\log p(y|\mathbf{x}, \mathbf{W}^{(1)}, \mathbf{b}^{(1)}) = \log(1 - h^{(1)})$ otherwise. If we assume for now that there is only one datapoint (we'll see what happens with multiple datapoints later), we just have

$$\frac{d\mathcal{L}}{dh^{(1)}} = \begin{cases} y^1 = 0 : \frac{-1}{1-h^{(1)}} \\ y^1 = 1 : \frac{1}{h^{(1)}} \end{cases}$$

We know that $\mathcal{L}(h^{(1)}) = \mathcal{L}(\sigma(z^{(1)})) = \mathcal{L}(\sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}))$, and therefore we can use the chain rule to get

$$\frac{d\mathcal{L}}{dz^{(1)}} = \frac{d\mathcal{L}}{dh^{(1)}}\frac{dh^{(1)}}{dz^{(1)}}$$

We already know that $\frac{d\mathcal{L}}{dh^{(1)}}$ is simply 1 or -1. We know that

$$h_i^{(1)} = \sigma(z_i^{(1)}) = \frac{1}{1+\exp(-z_i^{(1)})},$$

and therefore

$$\frac{d\mathcal{L}}{dz_i^{(1)}} = \frac{d\mathcal{L}}{dh_i^{(1)}}\frac{d}{dz_i^{(1)}}\frac{1}{1+\exp(-z_i^{(1)})} = \frac{d\mathcal{L}}{dh_i^{(1)}}\frac{\exp(-z_i^{(1)})}{(1+\exp(-z_i^{(1)}))^2} = \frac{d\mathcal{L}}{dh_i^{(1)}}\sigma(z_i^{(1)})(1-\sigma(z_i^{(1)}))$$

To see why this is true, note that

$$(1-\sigma(z)) = \frac{1+\exp(-z)-1}{1+\exp(-z)} = \frac{\exp(-z)}{1+\exp(-z)}.$$

Note that $\frac{d\mathcal{L}}{dh_i^{(1)}}\frac{dh_i^{(1)}}{dz_j^{(1)}} = 0$ if $i \neq j$, so we only need to pointwise multiply each entry in $\frac{d\mathcal{L}}{dh_i^{(1)}}$ by $\sigma(z_i^{(1)})(1-\sigma(z_i^{(1)}))$. Now we just need to evaluate

$$\frac{d\mathcal{L}}{d\mathbf{W}^{(1)}} = \frac{d\mathcal{L}}{dz^{(1)}}\frac{dz^{(1)}}{d\mathbf{W}^{(1)}}$$
$$\frac{d\mathcal{L}}{d\mathbf{b}^{(1)}} = \frac{d\mathcal{L}}{dz^{(1)}}\frac{dz^{(1)}}{d\mathbf{b}^{(1)}}.$$

We know that $z^{(1)} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$. Let's start with the bias, we simply have

$$\frac{d\mathcal{L}}{d\mathbf{b}_i^{(1)}} = \sum_{j=1}^{M_1}\frac{d\mathcal{L}}{dz_j^{(1)}}\frac{dz_j^{(1)}}{d\mathbf{b}_i^{(1)}} = \frac{d\mathcal{L}}{dz_i^{(1)}},$$

since $\mathbf{b}_i^{(1)}$ only affects $z_i^{(1)}$, and not any other $z_j^{(1)}$ where $j \neq i$. Evaluating the derivative with respect to the weights matrix $\mathbf{W}^{(1)}$ is a bit more complex. We have

$$z_i^{(1)} = \sum_{j=1}^{M_0}\mathbf{W}_{i,j}^{(1)}\mathbf{x}_j + \mathbf{b}_i^{(1)},$$

and therefore

$$\frac{d\mathcal{L}}{d\mathbf{W}_{i,j}^{(1)}} = \sum_{k=1}^{M_1}\frac{d\mathcal{L}}{dz_k^{(1)}}\frac{dz_k^{(1)}}{d\mathbf{W}_{i,j}^{(1)}} = \frac{d\mathcal{L}}{dz_i^{(1)}}\mathbf{x}_j,$$

since we can see in the sum that $z_i^{(1)}$ only depends on $\mathbf{W}_{i,j}^{(1)}$ from $j = 1$ to $j = M_1$. We can also express this in matrix notation as

$$\frac{d\mathcal{L}}{d\mathbf{W}^{(1)}} = \frac{d\mathcal{L}}{dz^{(1)}}\mathbf{x}^T.$$

3 Backpropagation: recursive case intro

Now, let's say that we have a multilayer neural network. The last layer simply looks like logistic regression, except that now instead of $\mathbf{x}$, we have the activations in the second-to-last layer $h^{(L-1)}$. We therefore have:

$$\frac{d\mathcal{L}}{d\mathbf{b}^{(L)}} = \frac{d\mathcal{L}}{dz^{(L)}}$$
$$\frac{d\mathcal{L}}{d\mathbf{W}^{(L)}} = \frac{d\mathcal{L}}{dz^{(L)}}(h^{(L-1)})^T.$$

But how can we get the derivatives with respect to the weights in the preceding layers? Well, we note that the previous layers only affect $\mathcal{L}$ via $h^{(L-1)}$, so we simply need to know $\frac{d\mathcal{L}}{dh^{(L-1)}}$. Since we know that

$$z^{(L)} = \mathbf{W}^{(L)}h^{(L-1)} + \mathbf{b}^{(L)}$$

and therefore

$$z_i^{(L)} = \sum_{j=1}^{M_{L-1}} \mathbf{W}_{i,j}^{(L)} h_j^{(L-1)} + \mathbf{b}_i^{(L)},$$

we can derive

$$\frac{d\mathcal{L}}{dh_j^{(L-1)}} = \sum_{i=1}^{M_L} \frac{d\mathcal{L}}{dz_i^{(L)}} \frac{dz_i^{(L)}}{dh_j^{(L-1)}} = \sum_{i=1}^{M_L} \frac{d\mathcal{L}}{dz_i^{(L)}} \mathbf{W}_{i,j}^{(L)} = \sum_{i=1}^{M_L} \left((\mathbf{W}^{(L)})^T\right)_{j,i} \frac{d\mathcal{L}}{dz_i^{(L)}}$$

in matrix notation, this simply becomes

$$\frac{d\mathcal{L}}{dh^{(L-1)}} = (\mathbf{W}^{(L)})^T \frac{d\mathcal{L}}{dz^{(L)}}.$$

Now, we can simply proceed recursively, and use $\frac{d\mathcal{L}}{dh^{(L-1)}}$ in place of $\frac{d\mathcal{L}}{dh^{(L)}}$ to compute the derivatives with respect to $\mathbf{W}^{(L-1)}$ and $\mathbf{b}^{(L-1)}$.