Lecture 13
Back-propagation

02 March 2016

Taylor B. Arnold
Yale Statistics
STAT 365/665

Yale

Notes:

- Problem set 4 is due this Friday
- Problem set 5 is due a week from Monday (for those of you with a midterm crunch this week); I will post the questions by tomorrow morning

Neural network review

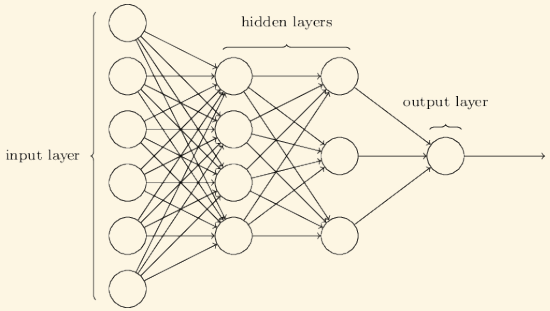
Last time we established the idea of a sigmoid neuron, which takes a vector of numeric variables x and emits a value as follows:

$$\sigma(x \cdot w + b) = \frac{1}{1 + e^{-(x \cdot w + b)}}$$

It is entirely defined by a vector of weights w and bias term b, and functions exactly like logistic regression.

Neural network review, cont.

These single neurons can be strung together to construct a neural network. The input variables are written as special neurons on the left-hand side of the diagram:



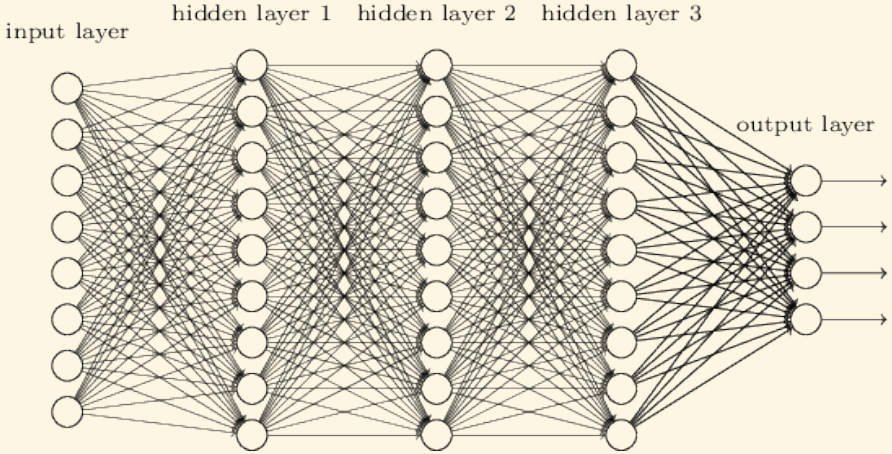
Stochastic gradient descent

We started talking about how to learn neural networks via a variant of gradient descent, called stochastic gradient descent. The only detail left to figure out is exactly how calculate the gradient of the cost function in an efficient way.

Idea behind back-propagation

Before starting down the path of explaining the math behind back-propagation, I want to explain what the big idea behind the method is and why it makes sense as a general approach.

Idea behind back-propagation, cont.



Some notation, cont.

We need a way of referring unambiguously to the weights and biases in a multi-layer neural network. To start define

$$w^l_{j,k}$$

As the weight of node k in layer $l-1$ as applied by node j in layer l.

Some notation, cont.

Now let:

$$b_j^l$$

Be the bias of node j of layer l.

Some notation, cont.

We now define two additional quantities. The weighted input emitted from node j in layer l:

$$z_j^l$$

The term input refers to the fact that this in the input to the activation function.

Some notation, cont.

And the activation

$$a_j^l$$

Derived from applying the function σ (the sigmoid or logit function for us so far) to the the weighted input z_j^l.

Some notation, cont.

That's a lot to take in all at once. Here is the cheat-sheet version that shows how all of these quantities fit together:

$$\begin{aligned} a_j^l &= \sigma(z_j^l) \\ &= \sigma(\sum_k w_{jk}^l a_k^{l-1} + b_j^l) \end{aligned}$$

Some notation, cont.

That's a lot to take in all at once. Here is the cheat-sheet version that shows how all of these quantities fit together:

$$\begin{aligned} a_j^l &= \sigma(z_j^l) \\ &= \sigma(\sum_k w_{jk}^l a_k^{l-1} + b_j^l) \end{aligned}$$

It will, also, be beneficial to define one more quantity:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l}$$

Called the error functions. These error functions will help in keeping the number of computations we need to make as small as possible.

Cost function again

In what follows we assume that we are dealing with one individual data point. So the cost is simply the cost of that one training point (which we called C_i in Monday's notes).

If the neural network has L total layers, notice that in general we will have the cost being a function only of the activations from layer L of the neural network:

$$C(w, b) = f(a_1^L, a_2^L, \ldots)$$

In our case, this can be explicitly written as:

$$C(w, b) = \sum_k (y_k - a_k^L)^2$$

Note: this sum is over the dimension of the output, not the number of samples! If we only have a univariate output, this will just be a single value.

Feed-forward step

We want to calculate how much the cost changes with respect to the errors on the outer layer. This ends up being a fairly straightforward application of the chain rule:

$$
\begin{aligned}
\delta_j^L &= \frac{\partial C}{\partial z_j^L} \\
&= \sum_k \frac{\partial C}{\partial a_k^L} \cdot \frac{\partial a_k^L}{\partial z_j^L} \\
&= \frac{\partial C}{\partial a_j^L} \cdot \frac{\partial a_j^L}{\partial z_j^L} \\
&= \frac{\partial C}{\partial a_j^L} \cdot \sigma'(z_j^L)
\end{aligned}
$$

Feed-forward step, cont.

We can explicitly calculate the first partial derivative given the cost function. Here for example it is given as:

$$\begin{aligned}\frac{\partial C}{\partial a_j^L} &= \frac{\partial}{\partial a_j^L} \sum_k (y_k - a_k^L)^2 \\ &= 2 \cdot (a_j^L - y_j)\end{aligned}$$

The function σ' is also easily determined by differentiation of the sigmoid function:

$$\sigma'(z) = \sigma(z) \cdot \sigma(1 - z)$$

Back-propagation step

We now want to relate the errors δ_j^l to the errors in δ_j^{l+1}. This allows us to then work backwards from the feed-forward step. Notice that:

$$\begin{aligned}\delta_j^l &= \frac{\partial C}{\partial z_j^l} \\ &= \sum_k \frac{\partial C}{\partial z_k^{l+1}} \cdot \frac{\partial z_k^{l+1}}{\partial z_j^l} \\ &= \sum_k \delta_k^{l+1} \cdot \frac{\partial z_k^{l+1}}{\partial z_j^l}\end{aligned}$$

Back-propagation step

We now want to relate the errors δ_j^l to the errors in δ_j^{l+1}. This allows us to then work backwards from the feed-forward step. Notice that:

$$\begin{aligned}\delta_j^l &= \frac{\partial C}{\partial z_j^l} \\ &= \sum_k \frac{\partial C}{\partial z_k^{l+1}} \cdot \frac{\partial z_k^{l+1}}{\partial z_j^l} \\ &= \sum_k \delta_k^{l+1} \cdot \frac{\partial z_k^{l+1}}{\partial z_j^l}\end{aligned}$$

To calculate the remaining derivatives, notice that we have a formula relating one set of weighted inputs to the next set:

$$\begin{aligned}z_k^{l+1} &= \sum_i w_{ki}^{l+1} a_i^l + b_k^{l+1} \\ &= \sum_i w_{ki}^{l+1} \sigma(z_i^l) + b_k^{l+1}\end{aligned}$$

Back-propagation step, cont.

Using this relationship, we can now take partial derivatives

$$\begin{aligned}\frac{\partial z_k^{l+1}}{\partial z_j^l} &= \frac{\partial}{\partial z_j^l}\left(\sum_i w_{ki}^{l+1}\sigma(z_j^l) + b_k^{l+1}\right) \\ &= w_{kj}^{l+1} \cdot \sigma'(z_j^l)\end{aligned}$$

Which when plugged into our original equations yields:

$$\delta_j^l = \sum_k \delta_k^{l+1} \cdot w_{kj}^{l+1} \cdot \sigma'(z_j^l)$$

Relating errors to weights and bias

We now just need to relate the errors δ_j^l to the gradient with respect to the weights and biases directly. We have, from the linearity of the bias term, a very simple relationship between δ_j^l with the gradient of the bias terms:

$$\begin{aligned}\delta_j^l &= \frac{\partial C}{\partial z_j^l} \\ &= \frac{\partial C}{\partial b_j^l} \cdot \frac{\partial b_j^l}{\partial z_j^l} \\ &= \frac{\partial C}{\partial b_j^l}\end{aligned}$$

Relating errors to weights and bias

For the weights, a very similar

$$\begin{aligned}\delta_j^l &= \frac{\partial C}{\partial z_j^l} \\ &= \frac{\partial C}{\partial w_{jk}^l} \cdot \frac{\partial w_{jk}^l}{\partial z_j^l} \\ &= \frac{\partial C}{\partial w_{jk}^l} \cdot \frac{1}{a_k^{l-1}}\end{aligned}$$

Which can be re-written as:

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l$$

Putting these all together

We have now four equations that describe the mechanics of back-propagation. I will write them here a bit more compactly for reference:

$$\begin{aligned}
\delta_j^L &= \nabla_{a^L} C \circ \sigma'(z^L) \\
\delta_j^l &= ((w^{l+1})^T \delta^{l+1}) \circ \sigma'(z^l) \\
\frac{\partial C}{\partial b_j^l} &= \delta_j^l \\
\frac{\partial C}{\partial w_{jk}^l} &= a_k^{l-1} \delta_j^l
\end{aligned}$$

Where dropping an index indicates that I am doing math on the entire vector or matrix of values. The symbol $\circ$ (Hadamard product) refers to element-wise multiplication; a common operation in programming but less-so in abstract mathematics.

The back-propagation algorithm

The back-propagation algorithm as a whole is then just:

1. Select an element i from the current minibatch and calculate the weighted inputs z and activations a for every layer using a forward pass through the network
2. Now, use these values to calculate the errors δ for each layer, starting at the last hidden layer and working backwards, using back-propagation
3. Calculate $\nabla_w C_i$ and $\nabla_b C_i$ using the last two equations
4. Repeat for all samples in the minibatch, to get:

$$\frac{\sum_{i \in M} \nabla_w C_i}{|M|} \approx \nabla_w C \qquad \frac{\sum_{i \in M} \nabla_b C_i}{|M|} \approx \nabla_b C$$

5. Update the weights and biases by (the estimates of) $-\eta \cdot \nabla_w C$ and $-\eta \cdot \nabla_b C$, respectively
6. Repeat over every mini-batch, and for the desired number of epochs