



## DATA SCIENCE

# Neural networks: training with backpropagation.



JEREMY JORDAN

18 JUL 2017 • 23 MIN READ

In my first post on neural networks, I discussed a model representation for neural networks and how we can feed in inputs and calculate an output. We calculated this output, layer by layer, by combining the inputs from the previous layer with weights for each neuron-neuron connection. I mentioned that we'd talk about how to find the proper weights to connect neurons together in a future post - this is that post!

## Overview

In the previous post I had just assumed that we had magic prior knowledge of the proper weights for each neural network. In this post, we'll actually figure out how to get our neural network to "learn" the proper weights. However, for the sake of having somewhere to start, let's just initialize each of the weights with random values as an initial guess.

You've successfully subscribed to Jeremy Jordan!

Given our randomly initialized weights connecting each of the neurons, we



chose our weights at random, our output is probably not going to be very good with respect to our expected output for the dataset.

So where do we go from here?

Well, for starters, let's define what a "good" output looks like. Namely, we'll develop a **cost function** which penalizes outputs far from the expected value.

Next, we need to figure out a way to change the weights so that the cost function improves. Any given path from an input neuron to an output neuron is essentially just a composition of functions; as such, we can use partial derivatives and the chain rule to **define the relationship between any given weight and the cost function**. We can use this knowledge to then leverage gradient descent in updating each of the weights.

## Pre-requisites

When I was first learning backpropagation, many people tried to abstract away the underlying math (derivative chains) and I was never really able to grasp what the heck was going on until watching Professor Winston's lecture at MIT. I hope that this post gives a better understanding of backpropagation than simply "this is the step where we send the error backwards to update the weights".

In order to fully grasp the concepts discussed in this post, you should be familiar with the following:

You've successfully subscribed to Jeremy Jordan!

However, it is my hope that even a reader without prior knowledge of multivariate calculus can still follow the logic behind backpropagation.

If you're not familiar with calculus,  $\frac{\partial f(x)}{\partial x}$  will probably look pretty foreign. You can interpret this expression as "how does  $f(x)$  change as I change  $x$ ?" This will be useful because we can ask questions like "How does the cost function change when I change this parameter? Does it increase or decrease the cost function?" in search of the optimal parameters. If you'd like to brush up on multivariate calculus, check out [Khan Academy's lessons](#) on the subject.

## Gradient descent

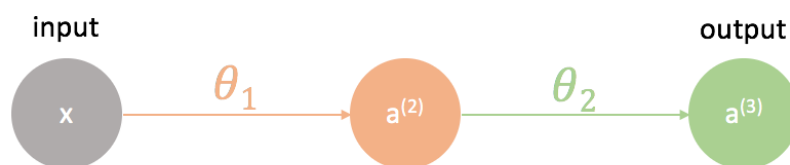
To prevent this post from getting too long, I've separated the topic of gradient descent into another post. If you're not familiar with the method, be sure to read about it [here](#) and understand it before continuing this post.

## Matrix multiplication

[Here](#) is a quick refresher from Khan Academy.

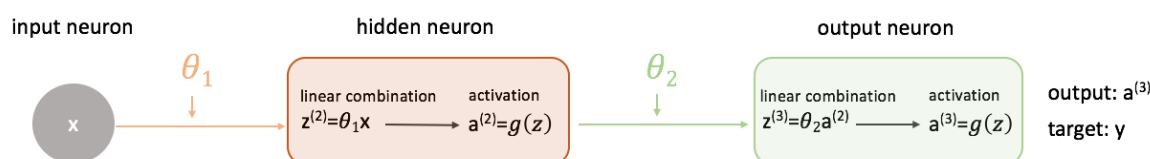
## Starting simple

To figure out how to use gradient descent in training a neural network, let's start with the simplest neural network: one input neuron, one hidden layer neuron, and one output neuron.



You've successfully subscribed to Jeremy Jordan!

To show a more complete picture of what's going on, I've expanded each neuron to show 1) the linear combination of inputs and weights and 2) the activation of this linear combination. It's easy to see that the forward propagation step is simply a series of functions where the output of one feeds as the input to the next.



## Defining "good" performance in a neural network

Let's define our cost function to simply be the squared error.

$$J(\theta) = \frac{1}{2} (y - a^{(3)})^2$$

There's a myriad of cost functions we could use, but for this neural network squared error will work just fine.

Remember, we want to evaluate our model's output with respect to the target output in an attempt to minimize the difference between the two.

You've successfully subscribed to Jeremy Jordan!



In order to minimize the difference between our neural network's output

need to define the relationship (read: partial derivative) between our cost function and each weight. We can then update these weights in an iterative process using gradient descent.

$$\frac{\partial J(\theta)}{\partial \theta_1} = ?$$

$$\frac{\partial J(\theta)}{\partial \theta_2} = ?$$

Let's look at  $\frac{\partial J(\theta)}{\partial \theta_2}$  first. Keep the following figure in mind as we progress.



Let's take a moment to examine how we could express the relationship between  $J(\theta)$  and  $\theta_2$ . Note how  $\theta_2$  is an input to  $z^{(3)}$ , which is an input to  $a^{(3)}$ , which is an input to  $J(\theta)$ . When we're trying to compute a derivative of this sort, we can use the chain rule to solve.

You've successfully subscribed to Jeremy Jordan!

$$\frac{\partial}{\partial z} p(q(z)) = \frac{\partial p}{\partial z} \frac{\partial q}{\partial z}$$

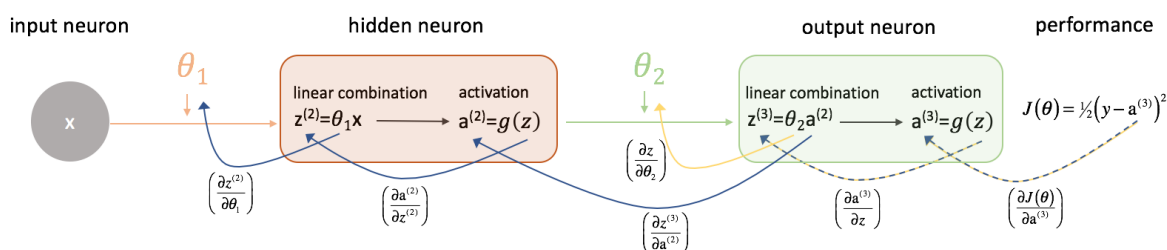
Let's apply the chain rule to solve for  $\frac{\partial J(\theta)}{\partial \theta_2}$ .

$$\frac{\partial J(\theta)}{\partial \theta_2} = \left( \frac{\partial J(\theta)}{\partial a^{(3)}} \right) \left( \frac{\partial a^{(3)}}{\partial z} \right) \left( \frac{\partial z}{\partial \theta_2} \right)$$

By similar logic, we can find  $\frac{\partial J(\theta)}{\partial \theta_1}$ .

$$\frac{\partial J(\theta)}{\partial \theta_1} = \left( \frac{\partial J(\theta)}{\partial a^{(3)}} \right) \left( \frac{\partial a^{(3)}}{\partial z^{(3)}} \right) \left( \frac{\partial z^{(3)}}{\partial a^{(2)}} \right) \left( \frac{\partial a^{(2)}}{\partial z^{(2)}} \right) \left( \frac{\partial z^{(2)}}{\partial \theta_1} \right)$$

For the sake of clarity, I've updated our neural network diagram to visualize these chains. Make sure you are comfortable with this process before proceeding.



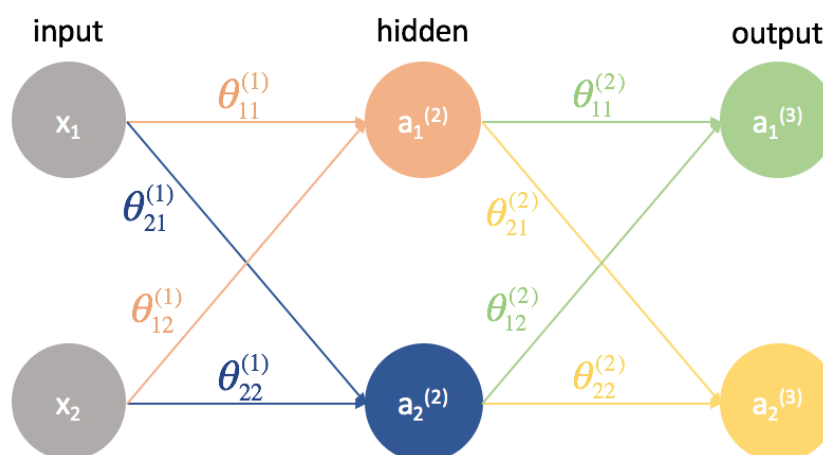
## Adding complexity

You've successfully subscribed to Jeremy Jordan!

two neurons in one hidden layer, and two neurons in our output layer. For now, we'll disregard the bias neurons that are missing from the input and

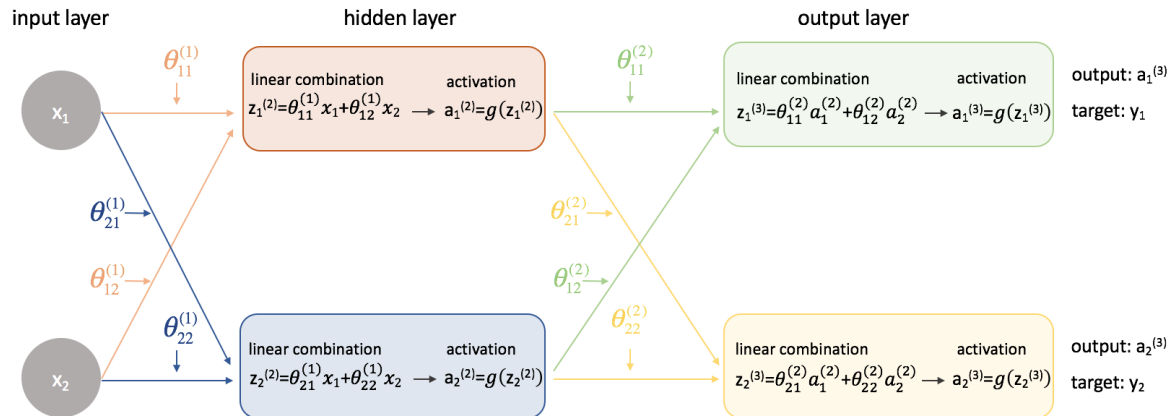


Let's take a second to go over the notation I'll be using so you can follow along with these diagrams. The **superscript** <sup>(1)</sup> denotes **what layer the object is in** and the **subscript** denotes **which neuron** we're referring to **in a given layer**. For example  $a_1^{(2)}$  is the activation of the first neuron in the second layer. For the **parameter values**  $\theta$ , I like to read them like a **mailing label** - the first value denotes what neuron the input is being sent **to** in the next layer, and the second value denotes **from** what neuron the information is being sent. For example,  $\theta_{21}^{(2)}$  is used to send input **to** the 2<sup>nd</sup> neuron, **from** the 1<sup>st</sup> neuron in layer 2. The superscript denoting the layer corresponds with where the input is coming from. This notation is consistent with the matrix representation we discussed in my post on neural networks representation.



You've successfully subscribed to Jeremy Jordan!

Let's expand this network to expose all of the math that's going on.



Yikes! Things got a little more complicated. I'll walk through the process for finding one of the partial derivatives of the cost function with respect to one of the parameter values; I'll leave the rest of the calculations as an exercise for the reader (and post the end results below).

Foremost, we'll need to revisit our cost function now that we're dealing with a neural network with more than one output. Let's now use the mean squared error as our cost function.

$$J(\theta) = \frac{1}{2m} \sum (y_i - a_i^{(2)})^2$$

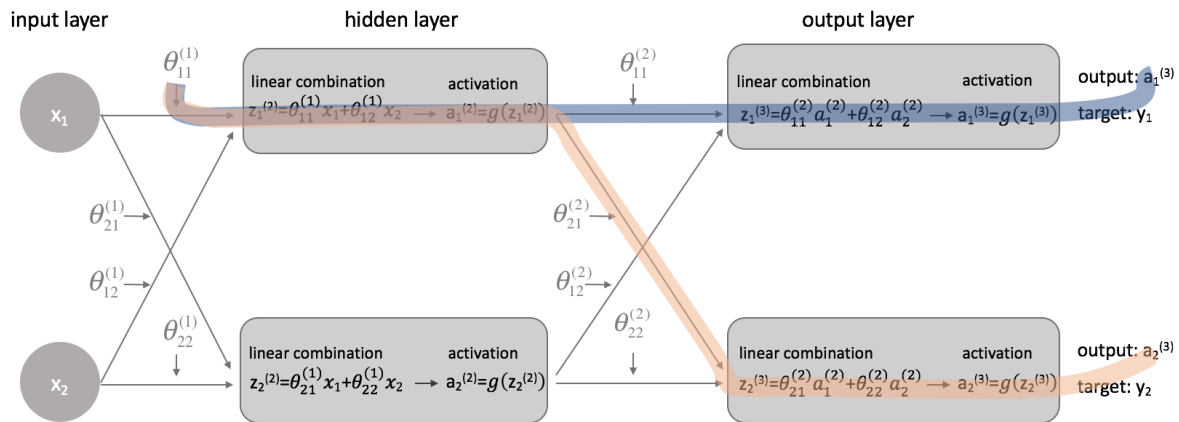
*Note: If you're training on multiple observations (which you always will in practice), we'll also need to perform a summation of the cost function over all training examples. For this cost function, it's common to normalize by  $\frac{1}{m}$  where  $m$  is the number of examples in your training*

You've successfully subscribed to Jeremy Jordan!

parameter affects the cost function. Specifically, I'll calculate  $\frac{\partial J(\theta)}{\partial \theta_{11}^{(1)}}$  in this



output, we can calculate the derivative chain for each path and simply add them together.



The derivative chain for the blue path is:

$$\left( \frac{\partial J(\theta)}{\partial a_1^{(3)}} \right) \left( \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} \right) \left( \frac{\partial z_1^{(3)}}{\partial a_1^{(2)}} \right) \left( \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \right) \left( \frac{\partial z_1^{(2)}}{\partial \theta_{11}^{(1)}} \right)$$

The derivative chain for the orange path is:

$$\left( \frac{\partial J(\theta)}{\partial a_2^{(3)}} \right) \left( \frac{\partial a_2^{(3)}}{\partial z_2^{(3)}} \right) \left( \frac{\partial z_2^{(3)}}{\partial a_1^{(2)}} \right) \left( \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \right) \left( \frac{\partial z_1^{(2)}}{\partial \theta_{11}^{(1)}} \right)$$

Combining these, we get the total expression for  $\frac{\partial J(\theta)}{\partial \theta_{11}^{(1)}}$ .

You've successfully subscribed to Jeremy Jordan!



$$\frac{\partial J(\theta)}{\partial \theta_{11}^{(1)}} = \left( \frac{\partial J(\theta)}{\partial a_1^{(3)}} \right) \left( \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} \right) \left( \frac{\partial z_1^{(3)}}{\partial a_1^{(2)}} \right) \left( \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \right) \left( \frac{\partial z_1^{(2)}}{\partial \theta_{11}^{(1)}} \right) + \left( \frac{\partial J(\theta)}{\partial a_2^{(3)}} \right) \left( \frac{\partial a_2^{(3)}}{\partial z_2^{(3)}} \right) \left( \frac{\partial z_2^{(3)}}{\partial a_1^{(2)}} \right) \left( \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \right) \left( \frac{\partial z_1^{(2)}}{\partial \theta_{11}^{(1)}} \right)$$

I've provided the remainder of the partial derivatives below. Remember, we need these partial derivatives because they describe how changing each parameter affects the cost function. Thus, we can use this knowledge to change all of the parameter values in a way that continues to decrease the cost function until we converge on some minimum value.

## Layer 2 Parameters

$$\frac{\partial J(\theta)}{\partial \theta_{11}^{(2)}} = \left( \frac{\partial J(\theta)}{\partial a_1^{(3)}} \right) \left( \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} \right) \left( \frac{\partial z_1^{(3)}}{\partial \theta_{11}^{(2)}} \right)$$

$$\frac{\partial J(\theta)}{\partial \theta_{12}^{(2)}} = \left( \frac{\partial J(\theta)}{\partial a_1^{(3)}} \right) \left( \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} \right) \left( \frac{\partial z_1^{(3)}}{\partial \theta_{12}^{(2)}} \right)$$

$$\frac{\partial J(\theta)}{\partial \theta_{21}^{(2)}} = \left( \frac{\partial J(\theta)}{\partial a_2^{(3)}} \right) \left( \frac{\partial a_2^{(3)}}{\partial z_2^{(3)}} \right) \left( \frac{\partial z_2^{(3)}}{\partial \theta_{21}^{(2)}} \right)$$

$$\frac{\partial J(\theta)}{\partial \theta_{22}^{(2)}} = \left( \frac{\partial J(\theta)}{\partial a_2^{(3)}} \right) \left( \frac{\partial a_2^{(3)}}{\partial z_2^{(3)}} \right) \left( \frac{\partial z_2^{(3)}}{\partial \theta_{22}^{(2)}} \right)$$

## Layer 1 Parameters

$$\frac{\partial J(\theta)}{\partial \theta_{11}^{(1)}} = \left( \frac{\partial J(\theta)}{\partial a_1^{(3)}} \right) \left( \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} \right) \left( \frac{\partial z_1^{(3)}}{\partial a_1^{(2)}} \right) \left( \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \right) \left( \frac{\partial z_1^{(2)}}{\partial \theta_{11}^{(1)}} \right) + \left( \frac{\partial J(\theta)}{\partial a_2^{(3)}} \right) \left( \frac{\partial a_2^{(3)}}{\partial z_2^{(3)}} \right) \left( \frac{\partial z_2^{(3)}}{\partial a_1^{(2)}} \right) \left( \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \right) \left( \frac{\partial z_1^{(2)}}{\partial \theta_{11}^{(1)}} \right)$$

$$\frac{\partial J(\theta)}{\partial \theta_{12}^{(1)}} = \left( \frac{\partial J(\theta)}{\partial a_1^{(3)}} \right) \left( \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} \right) \left( \frac{\partial z_1^{(3)}}{\partial a_1^{(2)}} \right) \left( \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \right) \left( \frac{\partial z_1^{(2)}}{\partial \theta_{12}^{(1)}} \right) + \left( \frac{\partial J(\theta)}{\partial a_2^{(3)}} \right) \left( \frac{\partial a_2^{(3)}}{\partial z_2^{(3)}} \right) \left( \frac{\partial z_2^{(3)}}{\partial a_1^{(2)}} \right) \left( \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \right) \left( \frac{\partial z_1^{(2)}}{\partial \theta_{12}^{(1)}} \right)$$

You've successfully subscribed to Jeremy Jordan!

$$\frac{\partial J(\theta)}{\partial \theta_{11}^{(1)}} = \left( \frac{\partial J(\theta)}{\partial a_1^{(3)}} \right) \left( \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} \right) \left( \frac{\partial z_1^{(3)}}{\partial a_1^{(2)}} \right) \left( \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \right) \left( \frac{\partial z_1^{(2)}}{\partial \theta_{11}^{(1)}} \right) + \left( \frac{\partial J(\theta)}{\partial a_2^{(3)}} \right) \left( \frac{\partial a_2^{(3)}}{\partial z_2^{(3)}} \right) \left( \frac{\partial z_2^{(3)}}{\partial a_1^{(2)}} \right) \left( \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \right) \left( \frac{\partial z_1^{(2)}}{\partial \theta_{11}^{(1)}} \right)$$



Woah there. We just went from a neural network with 2 parameters that needed 8 partial derivative terms in the previous example to a neural network with 8 parameters that needed 52 partial derivative terms. This is going to quickly get out of hand, especially considering many neural networks that are used in practice are much larger than these examples.

Fortunately, upon closer inspection many of these partial derivatives are repeated. If we're smart about how we approach this problem, we can dramatically reduce the computational cost of training. Further, it'd really be a pain in the ass if we had to manually calculate the derivative chains for every parameter. Let's look at what we've done so far and see if we can generalize a method to this madness.

## Generalizing a method

Let's examine the partial derivatives above and make a few observations. We'll start with looking at the partial derivatives with respect to the parameters for layer 2. Remember, the parameters in for layer 2 are combined with the activations in layer 2 to feed as inputs into layer 3.

### Layer 2 Parameters

Let's analyze the following expressions; I encourage you to solve the partial derivatives as we go along to convince yourself of my logic.

$$\frac{\partial J(\theta)}{\partial \theta_{11}^{(2)}} = \left( \frac{\partial J(\theta)}{\partial a_1^{(3)}} \right) \left( \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} \right) \left( \frac{\partial z_1^{(3)}}{\partial \theta_{11}^{(2)}} \right)$$

$$\frac{\partial J(\theta)}{\partial \theta_{21}^{(2)}} = \left( \frac{\partial J(\theta)}{\partial a_2^{(3)}} \right) \left( \frac{\partial a_2^{(3)}}{\partial z_2^{(3)}} \right) \left( \frac{\partial z_2^{(3)}}{\partial \theta_{21}^{(2)}} \right)$$

You've successfully subscribed to Jeremy Jordan!

$$\frac{\partial J(\theta)}{\partial \theta_{21}^{(2)}} = \left( \frac{\partial J(\theta)}{\partial a_2^{(3)}} \right) \left( \frac{\partial a_2^{(3)}}{\partial z_2^{(3)}} \right) \left( \frac{\partial z_2^{(3)}}{\partial \theta_{21}^{(2)}} \right)$$



Foremost, it seems as if the columns contain very similar values. For example, the **first column** contains partial derivatives of the cost function with respect to the neural network outputs. In practice, this is the difference between the expected output and the actual output (and then scaled by  $m$ ) for each of the output neurons.

The **second column** represents the derivative of the activation function used in the output layer. Note that for each layer, neurons will use the same activation function. A homogenous activation function within a layer is necessary in order to be able to leverage matrix operations in calculating the neural network output. Thus, the value for the second column will be the same for all four terms.

The first and second columns can be combined as  $\delta_i^{(3)}$  for the sake of convenience later on down the road. It is not immediately evident why this would be helpful, but you'll see as we go backwards another layer why this is useful. People will often refer to this expression as the "error" term which we use to "send back error from the output throughout the network". We'll see why this is the case soon. Each neuron in the network will have a corresponding  $\delta$  term that we'll solve for.

$$\delta_i^{(3)} = \frac{1}{m} (y_i - a_i^{(3)}) f' (a^{(3)})$$

The **third column** represents how the parameter of interest changes with respect to the weighted inputs for the current layer; when you calculate the derivative this corresponds with the activation from the previous layer.

You've successfully subscribed to Jeremy Jordan!

concerned with the first output neuron (neuron 1 in layer 3) while the last two partial derivative terms seem concerned with the second output



this knowledge to rewrite the partial derivatives using the  $\delta$  expression we defined above.

$$\frac{\partial J(\theta)}{\partial \theta_{11}^{(2)}} = \delta_1^{(3)} \left( \frac{\partial z_1^{(3)}}{\partial \theta_{11}^{(2)}} \right)$$

$$\frac{\partial J(\theta)}{\partial \theta_{12}^{(2)}} = \delta_1^{(3)} \left( \frac{\partial z_1^{(3)}}{\partial \theta_{12}^{(2)}} \right)$$

$$\frac{\partial J(\theta)}{\partial \theta_{21}^{(2)}} = \delta_2^{(3)} \left( \frac{\partial z_2^{(3)}}{\partial \theta_{21}^{(2)}} \right)$$

$$\frac{\partial J(\theta)}{\partial \theta_{22}^{(2)}} = \delta_2^{(3)} \left( \frac{\partial z_2^{(3)}}{\partial \theta_{22}^{(2)}} \right)$$

Next, let's go ahead and calculate the last partial derivative term. As we noted, this partial derivative ends up representing activations from the previous layer.

*Note: I find it helpful to use the expanded neural network graphic in the previous section when calculating the partial derivatives.*

$$\frac{\partial J(\theta)}{\partial \theta_{11}^{(2)}} = \delta_1^{(3)} a_1^{(2)}$$

$$\frac{\partial J(\theta)}{\partial \theta_{12}^{(2)}} = \delta_1^{(3)} a_2^{(2)}$$

$$\frac{\partial J(\theta)}{\partial \theta_{21}^{(2)}} = \delta_2^{(3)} a_1^{(2)}$$

$$\frac{\partial J(\theta)}{\partial \theta_{22}^{(2)}} = \delta_2^{(3)} a_2^{(2)}$$

You've successfully subscribed to Jeremy Jordan!

the previous layer to calculate each partial derivative. It's also interesting to note that the indices  $j$  and  $k$  for  $\theta_{jk}$  match the combined indices of  $\delta_j^{(3)}$



$$\frac{\partial J(\theta)}{\partial \theta_{jk}^{(2)}} = \delta_j^{(3)} a_k^{(2)}$$

Let's see if we can figure out a matrix operation to compute all of the partial derivatives in one expression.

$\delta^{(3)}$  is a vector of length  $j$  where  $j$  is equal to the number of output neurons.

$$\delta^{(3)} = \begin{bmatrix} y_1 - a_1^{(3)} \\ y_2 - a_2^{(3)} \\ \dots \\ y_j - a_j^{(3)} \end{bmatrix} f'(a^{(3)})$$

$a^{(2)}$  is a vector of length  $k$  where  $k$  is the number of neurons in the previous layer. The values of this vector represent activations of the previous layer calculated during forward propagation; in other words, it's the vector of inputs to the output layer.

$$a^{(2)} = \begin{bmatrix} a_1^{(2)} & a_2^{(2)} & \dots & a_k^{(2)} \end{bmatrix}$$

Multiplying these vectors together, we can calculate all of the partial derivative terms in one expression.

$$\frac{\partial J(\theta)}{\partial \theta_{:,j}^{(2)}} = \begin{bmatrix} \delta_1^{(3)} \\ \delta_j^{(3)} \end{bmatrix} \begin{bmatrix} a_1^{(2)} & a_2^{(2)} \end{bmatrix} = \begin{bmatrix} \delta_1^{(3)} a_1^{(2)} & \delta_1^{(3)} a_2^{(2)} \\ \delta_j^{(3)} a_1^{(2)} & \delta_j^{(3)} a_2^{(2)} \end{bmatrix}$$

You've successfully subscribed to Jeremy Jordan!



"error" term  
 $\delta^{(3)}$

$(y_i - a_i^{(3)})$     $f'(a_i^{(3)})$    input

Layer 2

$$\frac{\partial J(\theta)}{\partial \theta_{11}^{(2)}} = \left( \frac{\partial J(\theta)}{\partial a_1^{(3)}} \right) \left( \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} \right) \left( \frac{\partial z_1^{(3)}}{\partial \theta_{11}^{(2)}} \right)$$

first output neuron

$$\delta_i^{(3)} = \frac{1}{m} (y_i - a_i^{(3)}) f'(a_i^{(3)})$$

$$\frac{\partial J(\theta)}{\partial \theta_{ij}^{(2)}} = \begin{bmatrix} \delta_1^{(3)} \\ \delta_2^{(3)} \end{bmatrix} \begin{bmatrix} a_1^{(2)} & a_2^{(2)} \end{bmatrix} = \begin{bmatrix} \delta_1^{(3)} a_1^{(2)} & \delta_1^{(3)} a_2^{(2)} \\ \delta_2^{(3)} a_1^{(2)} & \delta_2^{(3)} a_2^{(2)} \end{bmatrix}$$

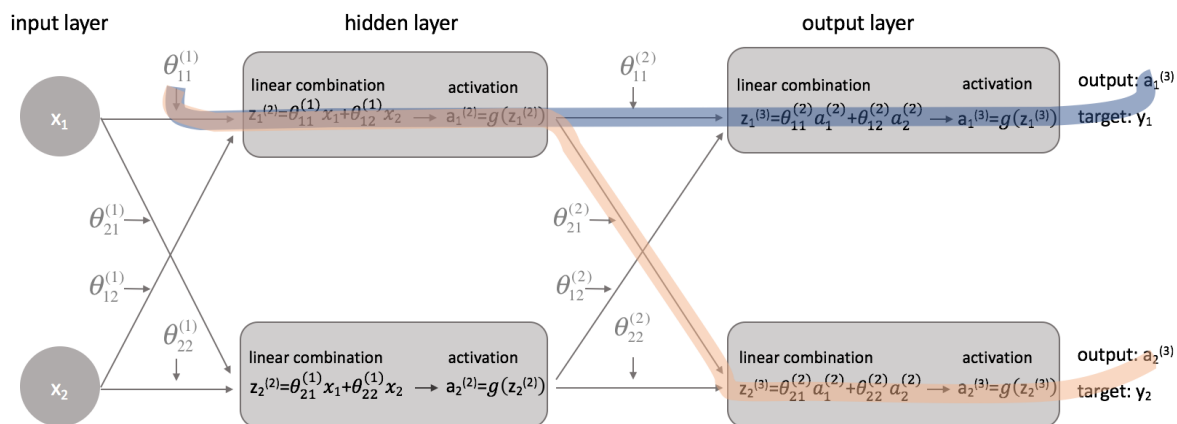
second output neuron

Note: Technically, the first column should also be scaled by  $m$  to be an accurate derivative. However, my focus was to point out that this is the column where we're concerned with the difference between the expected and actual outputs. The  $\delta$  term on the right has the full derivative expression.

## Layer 1 Parameters

Now let's take a look and see what's going on in layer 1.

Whereas the weights in layer 2 only directly affected one output, the weights in layer 1 affect all of the outputs. Recall the following graphic.



You've successfully subscribed to Jeremy Jordan!

This results in a partial derivative of the cost function with respect to a



we'll have a derivative chain for every  $\delta$  we calculated in the next layer forward. Remember, we started at the end of the network and are working our way backwards through the network. Thus, the next layer forward represents the  $\delta$  values that we previously calculated.

$\delta^{(2)}$                        $\delta^{(2)}$   
 "proportional error"      "proportional error"  
 $\delta^{(3)}$     $\theta^{(2)}$     $f'(a^{(2)})$    input    $\delta^{(3)}$     $\theta^{(2)}$     $f'(a^{(2)})$    input

Layer 1

$$\frac{\partial J(\theta)}{\partial \theta_{11}^{(1)}} = \left( \frac{\partial J(\theta)}{\partial a_1^{(3)}} \right) \left( \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} \right) \left( \frac{\partial z_1^{(3)}}{\partial a_1^{(2)}} \right) \left( \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \right) \left( \frac{\partial z_1^{(2)}}{\partial \theta_{11}^{(1)}} \right) + \left( \frac{\partial J(\theta)}{\partial a_2^{(3)}} \right) \left( \frac{\partial a_2^{(3)}}{\partial z_2^{(3)}} \right) \left( \frac{\partial z_2^{(3)}}{\partial a_1^{(2)}} \right) \left( \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \right) \left( \frac{\partial z_1^{(2)}}{\partial \theta_{11}^{(1)}} \right)$$

$$\frac{\partial J(\theta)}{\partial \theta_{12}^{(1)}} = \left( \frac{\partial J(\theta)}{\partial a_1^{(3)}} \right) \left( \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} \right) \left( \frac{\partial z_1^{(3)}}{\partial a_1^{(2)}} \right) \left( \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \right) \left( \frac{\partial z_1^{(2)}}{\partial \theta_{12}^{(1)}} \right) + \left( \frac{\partial J(\theta)}{\partial a_2^{(3)}} \right) \left( \frac{\partial a_2^{(3)}}{\partial z_2^{(3)}} \right) \left( \frac{\partial z_2^{(3)}}{\partial a_1^{(2)}} \right) \left( \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \right) \left( \frac{\partial z_1^{(2)}}{\partial \theta_{12}^{(1)}} \right)$$

$$\frac{\partial J(\theta)}{\partial \theta_{21}^{(1)}} = \left( \frac{\partial J(\theta)}{\partial a_1^{(3)}} \right) \left( \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} \right) \left( \frac{\partial z_1^{(3)}}{\partial a_2^{(2)}} \right) \left( \frac{\partial a_2^{(2)}}{\partial z_2^{(2)}} \right) \left( \frac{\partial z_2^{(2)}}{\partial \theta_{21}^{(1)}} \right) + \left( \frac{\partial J(\theta)}{\partial a_2^{(3)}} \right) \left( \frac{\partial a_2^{(3)}}{\partial z_2^{(3)}} \right) \left( \frac{\partial z_2^{(3)}}{\partial a_2^{(2)}} \right) \left( \frac{\partial a_2^{(2)}}{\partial z_2^{(2)}} \right) \left( \frac{\partial z_2^{(2)}}{\partial \theta_{21}^{(1)}} \right)$$

$$\frac{\partial J(\theta)}{\partial \theta_{22}^{(1)}} = \left( \frac{\partial J(\theta)}{\partial a_1^{(3)}} \right) \left( \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} \right) \left( \frac{\partial z_1^{(3)}}{\partial a_2^{(2)}} \right) \left( \frac{\partial a_2^{(2)}}{\partial z_2^{(2)}} \right) \left( \frac{\partial z_2^{(2)}}{\partial \theta_{22}^{(1)}} \right) + \left( \frac{\partial J(\theta)}{\partial a_2^{(3)}} \right) \left( \frac{\partial a_2^{(3)}}{\partial z_2^{(3)}} \right) \left( \frac{\partial z_2^{(3)}}{\partial a_2^{(2)}} \right) \left( \frac{\partial a_2^{(2)}}{\partial z_2^{(2)}} \right) \left( \frac{\partial z_2^{(2)}}{\partial \theta_{22}^{(1)}} \right)$$

derivative chain for  $\delta_1^{(3)}$                       derivative chain for  $\delta_2^{(3)}$

As we did for layer 2, let's make a few observations.

The **first two columns** (of each summed term) corresponds with a  $\delta_j^{(3)}$  calculated in the next layer forward (remember, we started at the end of the network and are working our way back).

The **third column** corresponds with some parameter that connects layer 2 to layer 3. If we considered  $\delta_j^{(3)}$  to be some "error" term for layer 3, and each derivative chain is now a summation of these errors, then this third column allows us to weight each respective error. Thus, the first three terms combined represent some measure of the proportional error.

You've successfully subscribed to Jeremy Jordan!





We could write this more succinctly using matrix expressions.

$$\delta^{(l)} = \delta^{(l+1)} \Theta^{(l)} f' (a^{(l)})$$

*Note: It's standard notation to denote vectors with lowercase letters and matrices as uppercase letters. Thus,  $\theta$  represents a vector while  $\Theta$  represents a matrix.*

The **fourth column** represents the derivative of the activation function used in the current layer. Remember that for each layer, neurons will use the same activation function.

Lastly, the **fifth column** represents various inputs from the previous layer. In this case, it is the actual inputs to the neural network.

Let's take a closer look at one of the terms,  $\frac{\partial J(\theta)}{\partial \theta_{11}^{(1)}}$ .

Foremost, we established that the first two columns of each derivative chains were previously calculated as  $\delta_j^{(3)}$ .

$$\frac{\partial J(\theta)}{\partial \theta_{11}^{(1)}} = \delta_1^{(3)} \left( \frac{\partial z_1^{(3)}}{\partial a_1^{(2)}} \right) \left( \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \right) \left( \frac{\partial z_1^{(2)}}{\partial \theta_{11}^{(1)}} \right) + \delta_2^{(3)} \left( \frac{\partial z_2^{(3)}}{\partial a_1^{(2)}} \right) \left( \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \right) \left( \frac{\partial z_1^{(2)}}{\partial \theta_{11}^{(1)}} \right)$$

Further, we established the the third columns of each derivative chain was

You've successfully subscribed to Jeremy Jordan!

function.



$$\frac{\partial J(\theta)}{\partial \theta_{11}^{(1)}} = \delta_1^{(3)} \theta_{11}^{(2)} f'(a^{(2)}) \left( \frac{\partial z_1^{(2)}}{\partial \theta_{11}^{(1)}} \right) + \delta_2^{(3)} \theta_{21}^{(2)} f'(a^{(2)}) \left( \frac{\partial z_1^{(2)}}{\partial \theta_{11}^{(1)}} \right)$$

Factoring out  $\left( \frac{\partial z_1^{(2)}}{\partial \theta_{11}^{(1)}} \right)$ ,

$$\frac{\partial J(\theta)}{\partial \theta_{11}^{(1)}} = \left( \frac{\partial z_1^{(2)}}{\partial \theta_{11}^{(1)}} \right) (\delta_1^{(3)} \theta_{11}^{(2)} f'(a^{(2)}) + \delta_2^{(3)} \theta_{21}^{(2)} f'(a^{(2)}))$$

we're left with our new definition of  $\delta_j^{(l)}$ . Let's go ahead and substitute that in.

$$\frac{\partial J(\theta)}{\partial \theta_{11}^{(1)}} = \left( \frac{\partial z_1^{(2)}}{\partial \theta_{11}^{(1)}} \right) (\delta_1^{(2)})$$

Lastly, we established the fifth column  $\left( \frac{\partial z_1^{(2)}}{\partial \theta_{11}^{(1)}} \right)$  corresponded with an input from the previous layer. In this case, the derivative calculates to be  $x_1$ .

$$\frac{\partial J(\theta)}{\partial \theta_{11}^{(1)}} = \delta_1^{(2)} x_1$$

Again let's figure out a matrix operation to compute all of the partial derivatives in one expression.

$\delta^{(2)}$  is a vector of length  $j$  where  $j$  is the number of neurons in the current

You've successfully subscribed to Jeremy Jordan!

layer 2.



$$\delta^{(2)} = \begin{bmatrix} \delta_1^{(2)} & \delta_2^{(2)} \end{bmatrix} f' (a^{(2)}) = \begin{bmatrix} \delta_1^{(2)} & \delta_2^{(2)} \end{bmatrix}$$

$x$  is a vector of input values.

Multiplying these vectors together, we can calculate all of the partial derivative terms in one expression.

$$\frac{\partial J(\theta)}{\partial \theta_{ij}^{(1)}} = \begin{bmatrix} \delta_1^{(2)} \\ \delta_2^{(2)} \end{bmatrix} \begin{bmatrix} x_1 & x_2 \end{bmatrix} = \begin{bmatrix} \delta_1^{(2)} x_1 & \delta_1^{(2)} x_2 \\ \delta_2^{(2)} x_1 & \delta_2^{(2)} x_2 \end{bmatrix}$$

If you've made it this far, congrats! We just calculated all of the partial derivatives necessary to use gradient descent and optimize our parameter values. In the next section, I'll introduce a way to visualize the process we've just developed in addition to presenting an end-to-end method for implementing backpropagation. If you understand everything up to this point, it should be smooth sailing from here on out.

## Backpropagation

In the last section, we developed a way to calculate all of the partial derivatives necessary for gradient descent (partial derivative of the cost function with respect to all model parameters) using matrix expressions. In calculating the partial derivatives, we started at the end of the network and, layer by layer, worked our way back to the beginning. We also developed a new term,  $\delta$ , which essentially serves to represent all of the partial derivative terms that we would need to reuse later and we progress, layer by layer, backwards through the network.

*Note: Backpropagation is simply a method for calculating the partial*

You've successfully subscribed to Jeremy Jordan!

*or another more advanced optimization technique.*

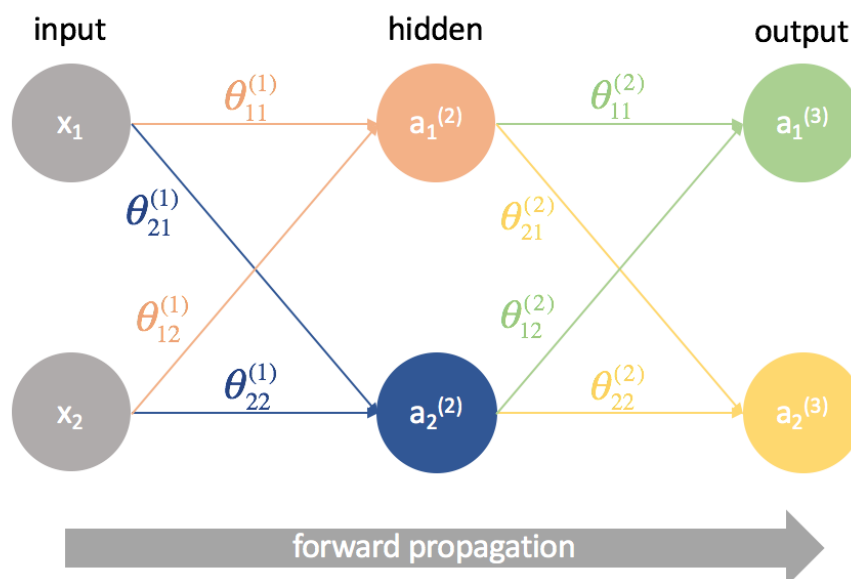
of the current layer.

$$\frac{\partial J(\theta)}{\partial \theta_{ij}^{(l)}} = (\delta^{(l+1)})^T a^{(l)}$$

## Backpropagation visualized

Before defining the formal method for backpropagation, I'd like to provide a visualization of the process.

First, we have to compute the output of a neural network via forward propagation.

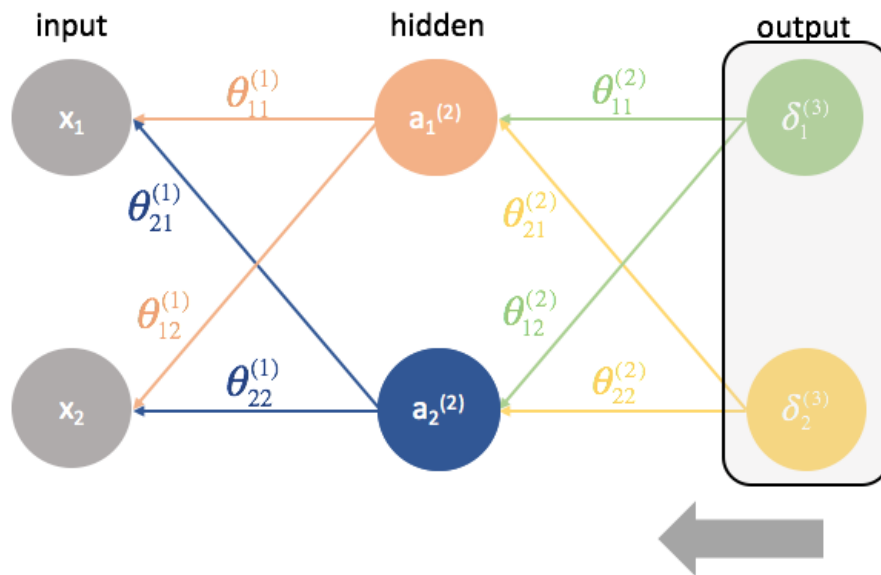


You've successfully subscribed to Jeremy Jordan!

Next, we compute the  $\delta^{(3)}$  terms for the last layer in the network.

be used again in calculating parameters for layers further back. In practice, we typically refer to  $\delta$  as the "error" term.

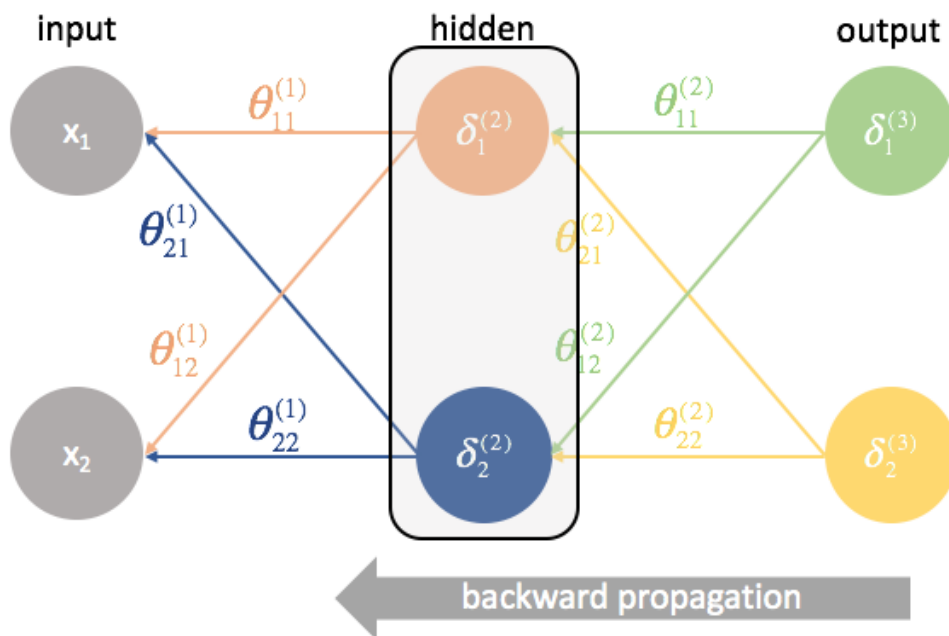
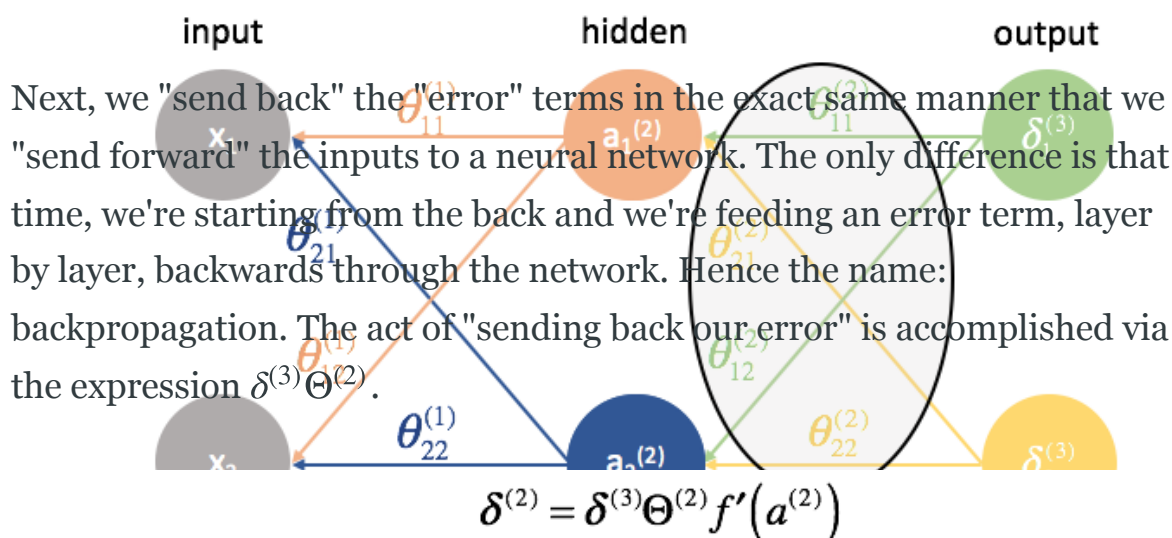
$$\delta^{(3)} = \frac{1}{m}(y - a^{(3)})f'(a^{(3)})$$



$\Theta^{(2)}$  is the matrix of parameters that connects layer 2 to 3. We multiply the error from the third layer by the inputs in the second layer to calculate our partial derivatives for this set of parameters.

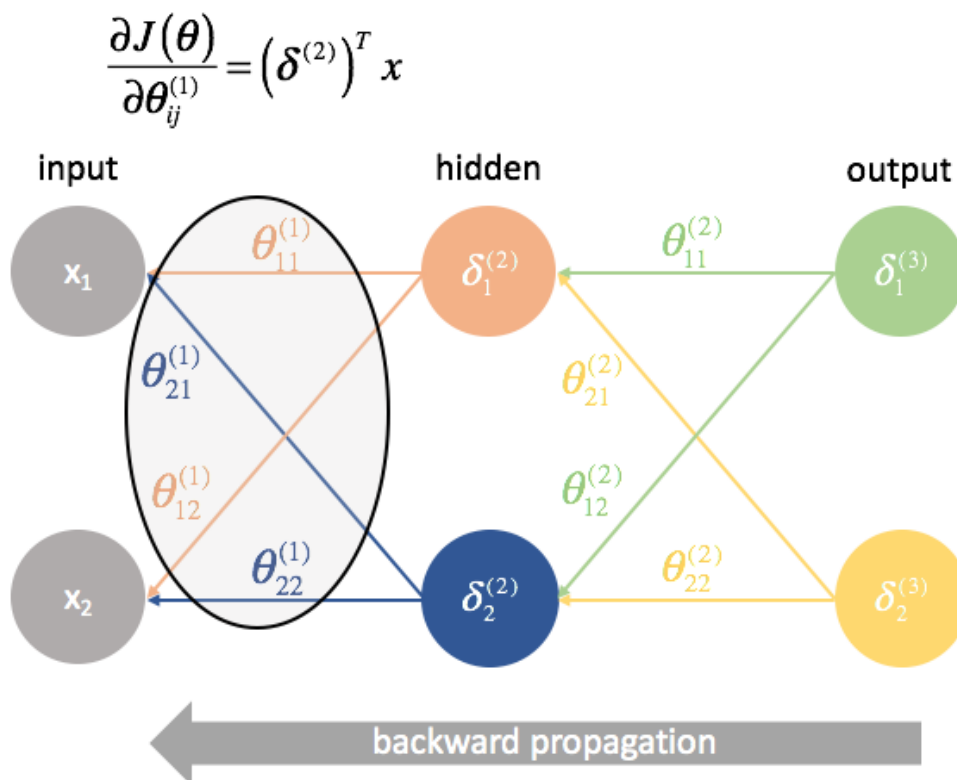
You've successfully subscribed to Jeremy Jordan!

$$\frac{\partial J(\theta)}{\partial \theta_{ij}^{(2)}} = (\delta^{(3)})^T a^{(2)}$$



You've successfully subscribed to Jeremy Jordan!

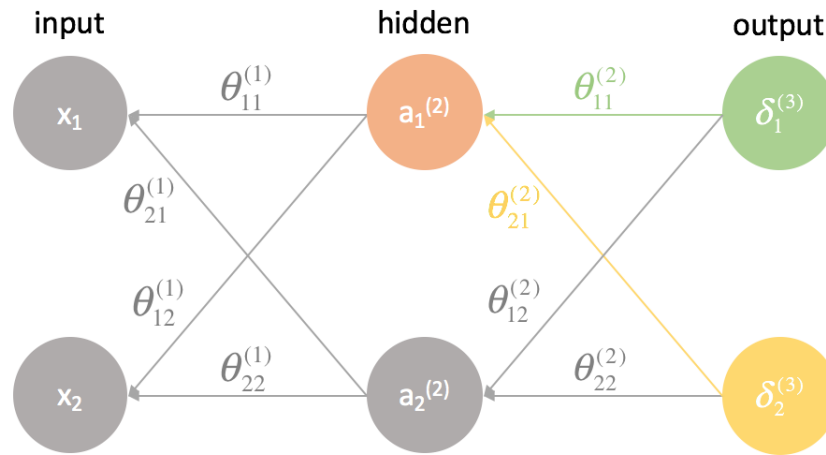
$\Theta^{(1)}$  is the matrix of parameters that connects layer 1 to 2. We multiply the error from the second layer by the inputs in the first layer to calculate our partial derivatives for this set of parameters.



For every layer except for the last, the "error" term is a linear combination of parameters connecting to the next layer (moving forward through the network) and the "error" terms of that next layer. This is true for all of the

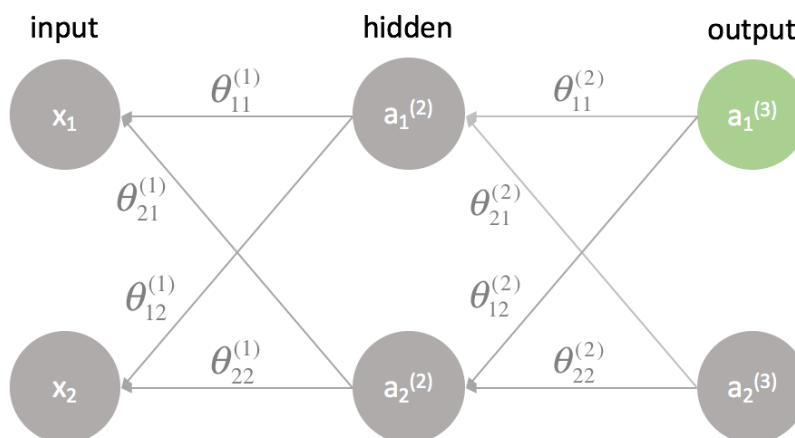
You've successfully subscribed to Jeremy Jordan!

$$\delta_1^{(2)} = \left( \delta_1^{(3)} \theta_{11}^{(2)} + \delta_2^{(3)} \theta_{21}^{(2)} \right) f'(a_1^{(2)})$$



The last layer is a special case because we calculate the  $\delta$  values by directly comparing each output neuron to its expected output.

$$\delta_1^{(3)} = \frac{1}{m} (y_1 - a_1^{(3)}) f'(a_1^{(3)})$$



You've successfully subscribed to Jeremy Jordan!

## A formalized method for implementing backpropagation





1. Perform forward propagation.
2. Compute the  $\delta$  term for the output layer.

$$\delta^{(L)} = \frac{1}{m} (y - a^{(L)}) f' (a^{(L)})$$

3. Compute the partial derivatives of the cost function with respect to all of the parameters that feed into the output layer,  $\Theta^{(L-1)}$ .

$$\frac{\partial J(\theta)}{\partial \theta_{ij}^{(L-1)}} = (\delta^{(L)})^T a^{(L-1)}$$

4. Go back one layer.  
 $l = l - 1$

5. Compute the  $\delta$  term for the current hidden layer.

$$\delta^{(l)} = \delta^{(l+1)} \Theta^{(l)} f' (a^{(l)})$$

6. Compute the partial derivatives of the cost function with respect to all of the parameters that feed into the current layer.

$$\frac{\partial J(\theta)}{\partial \theta_{ij}^{(l)}} = (\delta^{(l+1)})^T a^{(l)}$$

7. Repeat 4 through 6 until you reach the input layer.

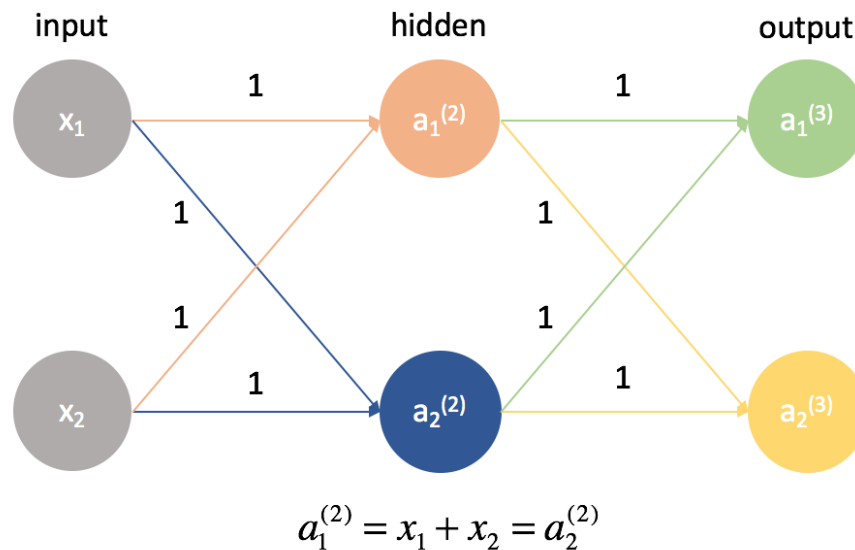
## Revisiting the weights initialization

When we started I proposed that we just randomly initialize our weights

You've successfully subscribed to Jeremy Jordan!

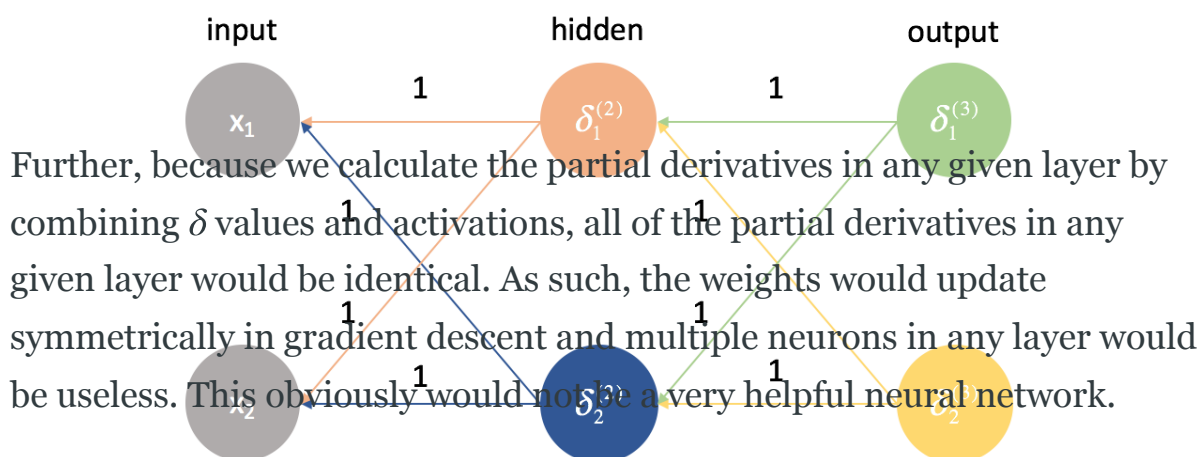
cost of our model.

so that we can break the symmetry in our model. If we had initialized all of our weights to be equal to be the same, every neuron in the next layer forward would be equal to the same linear combination of values.



By this same logic, the  $\delta$  values would also be the same for every neuron in a given layer.

You've successfully subscribed to Jeremy Jordan!



Randomly initializing the network's weights allows us to break this symmetry and update each weight individually according to its relationship with the cost function. Typically we'll assign each parameter to a random value in  $[-\epsilon, \epsilon]$  where  $\epsilon$  is some value close to zero.

## Putting it all together

After we've calculated all of the partial derivatives for the neural network parameters, we can use gradient descent to update the weights.

In general, we defined gradient descent as

$$\theta_i := \theta_i + \Delta\theta_i$$

where  $\Delta\theta_i$  is the "step" we take walking along the gradient, scaled by a learning rate,  $\eta$ .

31/06

You've successfully subscribed to Jeremy Jordan!

We'll use this formula to update each of the weights, recompute forward



the next weight update. This process continues until we've converged on an optimal value for our parameters. During each iteration we perform forward propagation to compute the outputs and backward propagation to compute the errors; one complete iteration is known as an epoch. It is common to report evaluation metrics after each epoch so that we can watch the evolution of our neural network as it trains.

## Further reading

- [The Matrix Calculus You Need For Deep Learning](#)
- [How the backpropagation algorithm works](#)

### 12a: Neural Nets



You've successfully subscribed to Jeremy Jordan!



## Lecture 10 - Neural Networks



- [Stanford cs231n: Backpropagation, Intuitions](#)
- [CS231n Winter 2016: Lecture 4: Backpropagation, Neural Networks 1](#)
- [Lectures on Deep Learning](#)
- [Yes you should understand backprop](#)

You've successfully subscribed to Jeremy Jordan!

- And in case you just gave up on backpropagation... [Deep Learning without Backpropagation](#)

Subscribe to Jeremy Jordan

Get the latest posts delivered right to your inbox

youremail@example.com

Subscribe

ALSO ON JEREMYJORDAN

<div>Organizing machine learning projects: ...</div> <div>4 years ago • 5 comments</div> <div>The goal of this document is to provide a common framework for ...</div>	<div>Common architectures in convolutional ...</div> <div>5 years ago • 4 comments</div> <div>In this post, I'll discuss commonly used architectures for ...</div>	<div>Effective testing for machine learning ...</div> <div>2 years ago • 6 comments</div> <div>Working as a core maintainer for PyTorch Lightning, I've grown a ...</div>	<div>Ar</div> <div>Ku</div> <div>3 y</div> <div>Thi</div> <div>an</div> <div>Ku</div>
--	--	---	---

12 Comments

JeremyJordan

Disqus' Privacy Policy

1 Login

Favorite 6

Tweet

Share

Sort by Best

Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Jithin J Kumar • 3 years ago

You've successfully subscribed to Jeremy Jordan!

Jeremy.!

1 ^ | v • Reply • Share ›

Jeremy Jordan

HOME

ABOUT

DATA SCIENCE

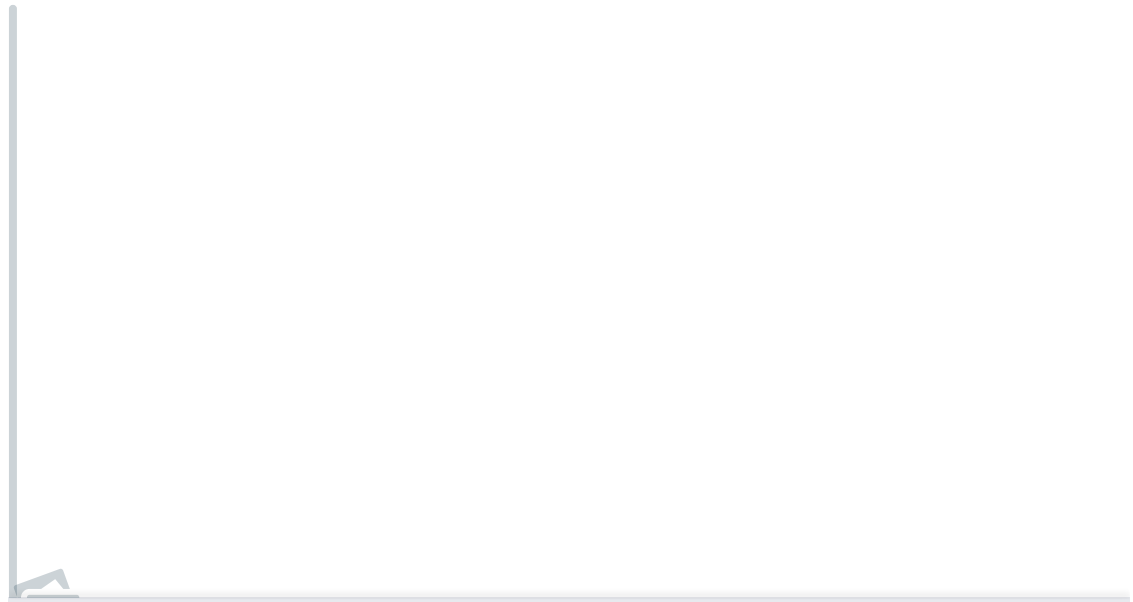
READING LIST

QUOTES

LIFE

FAQ

up: I'm denoting the bias neuron as a constant value of 1, which is multiplied by the bias values. (Each line represents a parameter)



see more

^ | v • Reply • Share ›



**Alexandr Klimov** • 7 months ago

Awesome explanation!

But I have one question: why we calculate partial derivatives of the 2th layer by not-updated-yet 2->3 conn. weights?

Why don't we calculate partial derivatives in such way:

1. calculate part. derivatives for the 3th layer's weights
2. update them
3. propagate calculation for the 2th layer's weight by already-updated 2->3 conn. weight

Why may this way above be wrong?

^ | v • Reply • Share ›



**Alexandr Klimov** → Alexandr Klimov • 7 months ago

It think I've found answer on my question: <https://stats.stackexchange.com/questions/474444/why-do-we-calculate-partial-derivatives-of-the-2th-layer-by-not-updated-yet-2-to-3-conn-weights>

^ | v • Reply • Share ›



**Ruvi Zhang** • a year ago

You've successfully subscribed to Jeremy Jordan!



**Karin** • 2 years ago

Hev.



derivate? And total derivate not a partial derivate?

Thanks

^ | v • Reply • Share ›



**Файль Гафаров** • 3 years ago

Great post! Many thanks! The mathematics used in neural networks and machine learning is rather simple. The most complex mathematics is in backpropagation algorithm. Yo can learn also in <https://learn-neural-networ...> But for doing machine learning it is not necessary to understand it mathematical basis. There are lot of ready tools for machine learning where all algorithms and mathematics is realized (Keras, Theano, Tensorflow, ...) But if someone is planning to develop new algorithms he needs to know mathematics of neural networks

^ | v • Reply • Share ›



**Jungyeon Baek** • 3 years ago

Thank you so much! it is definitely the clearest and simplest description of backpropagation I've seen!!

^ | v • Reply • Share ›



**Salman Shaukat** • 4 years ago

Thanks. This is simply the best explanation i could find.

^ | v • Reply • Share ›



**J Philius** • 4 years ago

Hi Jeremy for

You've successfully subscribed to Jeremy Jordan!



Jeremy Jordan

HOME

ABOUT

DATA SCIENCE

READING LIST

QUOTES

LIFE

FAV

**Jeremy Jordan** Mod → J Philius • 4 years ago

Thank you so much for catching this! I've updated the graphic to fix the typo :)

2 ^ | v • Reply • Share ›

**Saheli Jana** • 3 years ago

Nice Article !

Here is another one I liked, which focused mainly on the mathematical explanation and coding.

<http://www.adeveloperdiary....>

## MORE IN DATA SCIENCE

**A simple solution for monitoring ML systems.**

2 Jan 2021 – 10 min read

**Effective testing for machine learning systems.**

19 Aug 2020 – 9 min read

**An introduction to Kubernetes.**

26 Nov 2019 – 15 min read

[←](#) [→](#) [Home](#) [About](#) [Data Science](#) [Reading List](#) [Quotes](#) [Life](#) [Fav](#)

You've successfully subscribed to Jeremy Jordan!

## DATA SCIENCE

**Evaluating a machine learning model**<https://www.jeremyjordan.me/neural-networks-training/>


based on what we learn evaluating it. I'll answer questions like: How

 JEREMY JORDAN  
21 JUL 2017 • 10 MIN READ

DATA SCIENCE

Gradient descent.

Gradient descent is an optimization technique commonly used in training machine learning algorithms. Often when we're building a machine learning model, we'll develop a cost function which is capable of measuring how well our model is doing. This function will penalize any error our

 JEREMY JORDAN  
14 JUL 2017 • 9 MIN READ