CSE 412: Machine Learning Lab

Lab Activity 8: Understanding Backpropagation Algorithm

Dr Muhammad Abul Hasan*

May 20, 2024

1 Objective

In this lab activity, you will learn:

- To be familiar with the inside mechanics of Backpropagation algorithm.
- To be able to connect the relationship between parameter update and cost function.
- To be able to solve real-life problems using neural networks.

2 Background

In the previous lab manual, we assumed that we had a black box of the proper weights for each neural network. In this lab manual, we will figure out how to get our neural network to "learn" the appropriate weights. However, as we need to start somewhere, let us just initialize each of the weights with random values as an initial guess. we will come back and revisit this random initialization step later on in this lab manual.

Given our randomly initialized weights connecting each of the neurons, we can now feed in input data and calculate the outputs of our neural network. This is called forward propagation. Given that we chose our weights at random, our output will probably not be very good with respect to our expected output for the dataset.

So, how can we achieve the expected output from here?

First, let us define what a "good" output looks like. Namely, we will develop a cost function that penalizes outputs proportional to the error made (that is, difference between expected value and the output).

Next, we need to figure out a way to change the weights of the neural network so that the overall error reduces. 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.

3 Pre-requisites

In order to fully grasp the concepts, you should be familiar with the following:

- Partial derivatives

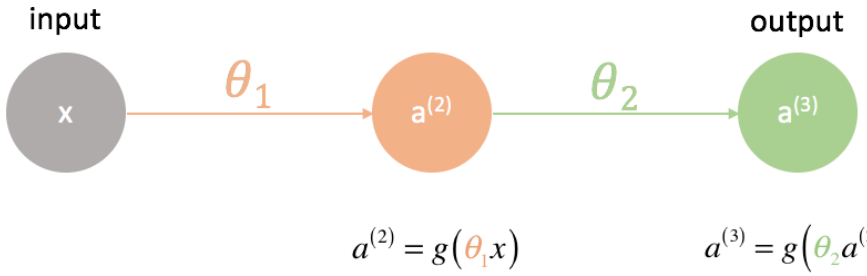
*Associate Professor, Deptartment of CSE, GUB ⁘ ✉ muhammad.hasan@cse.green.edu.bd

- Gradient descent (discussed in lab manual 2)
- Matrix multiplication

4 Toy Exercise

To figure out how to use gradient descent in training a neural network, let us start with the simplest neural network which has:

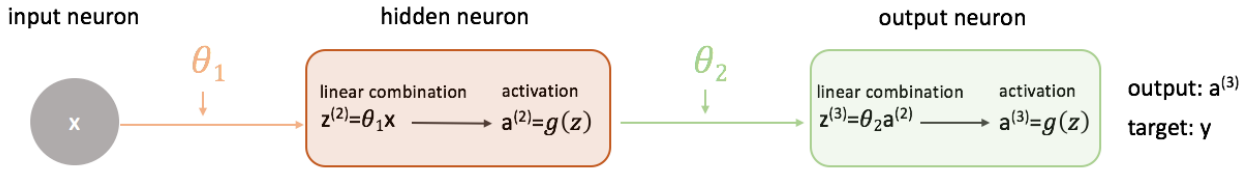
- one input neuron
- one hidden layer neuron
- and one output neuron



To show a more complete picture of what is going on, each neuron is expanded to show:

1. the *linear combination* of inputs and weights, and
2. the *activation* of this linear combination.

It is easy to see that the *forward propagation* step is simply a series of functions where the output of one node feeds as the input to the next node.



4.1 Defining Cost Function in a Neural Network:

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

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

There is a number of options that we could use as cost functions, but for this neural network, squared error will work just fine. We intend (the objective) to evaluate our model's output with respect to the target output in an attempt to minimize the difference between the two.

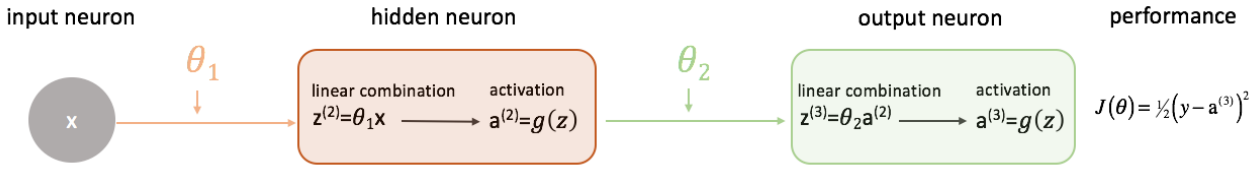
4.2 Relating the weights to the cost function

In order to minimize the difference between our neural network's output and the target output, we need to know how the model performance changes with respect to each parameter in our model. In other words, we 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 us look at $\frac{\partial J(\theta)}{\partial \theta_2}$ first. Keep the following figure in mind as we progress.



Let us take a moment to examine how we could express the relationship between $J(\theta)$ and θ_2. Carefully look at the diagram above, how θ_2 is an input to $z^{(3)}$, which is an input to $\mathrm{a}^{(3)}$, which is an input to $J(\theta)$. When we are trying to compute a derivative of this sort, we can use the chain rule to solve.

As a reminder, the chain rule states:

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

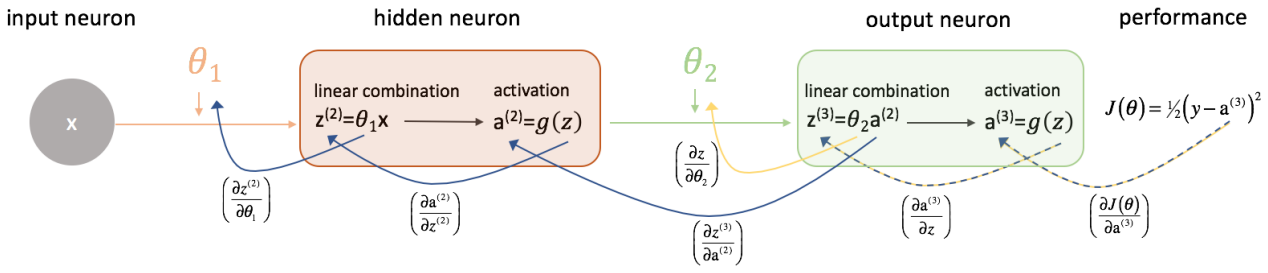
Let us apply the chain rule to solve for $\frac{\partial J(\theta)}{\partial \theta_2}$

$$\frac{\partial J(\theta)}{\partial \theta_2} = \left(\frac{\partial J(\theta)}{\partial \mathrm{a}^{(3)}}\right)\left(\frac{\partial \mathrm{a}^{(3)}}{\partial z}\right)\left(\frac{\partial z}{\partial \theta_2}\right) \tag{2}$$

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

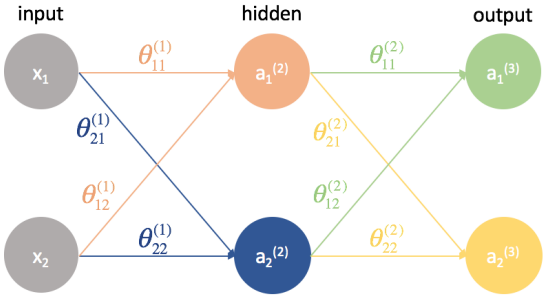
$$\frac{\partial J(\theta)}{\partial \theta_1} = \left(\frac{\partial J(\theta)}{\partial \mathrm{a}^{(3)}}\right)\left(\frac{\partial \mathrm{a}^{(3)}}{\partial z^{(3)}}\right)\left(\frac{\partial z^{(3)}}{\partial \mathrm{a}^{(2)}}\right)\left(\frac{\partial \mathrm{a}^{(2)}}{\partial z^{(2)}}\right)\left(\frac{\partial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

For better understanding, the following diagram is use to visualize these chains. Please take your time to have a grasp on it before proceeding.

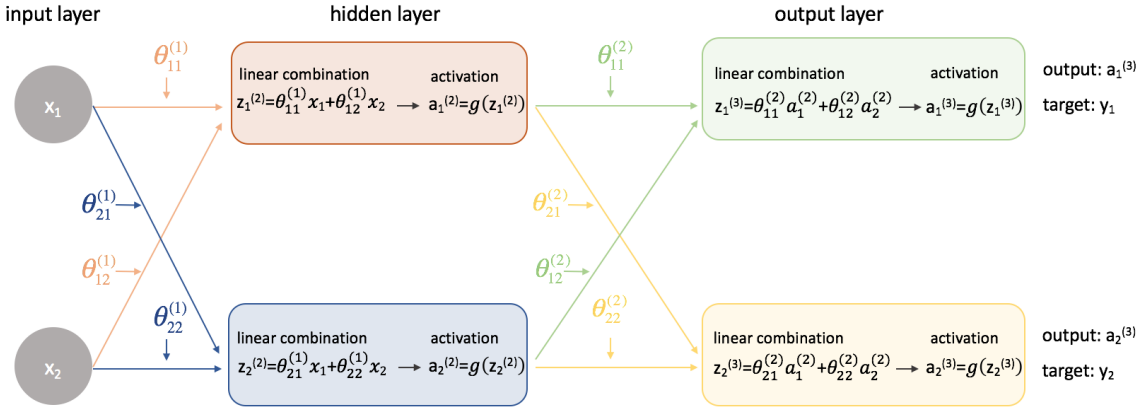


4.3 Adding Complexity

Now, let us try this same approach on a slightly more complicated example. Now, we will look at a neural network with two neurons in our input layer, two neurons in one hidden layer, and two neurons in our output layer. For now, we will disregard the bias neurons that are missing from the input and hidden layers.



Let us expand this network to expose all of the math that is going on.



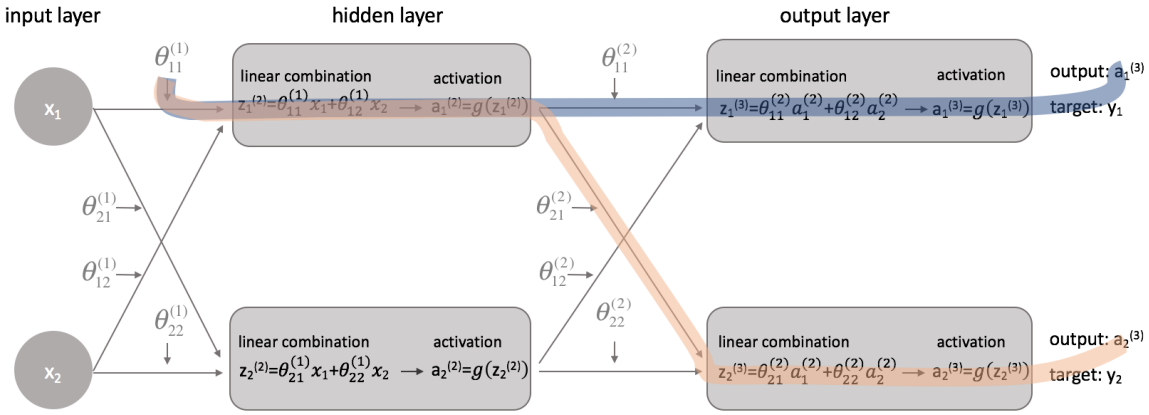
Clearly, things got a little more complicated. We will go through the process for finding one of the partial derivatives of the cost function with respect to one of the parameter. It it is understood clearly, the rest of the calculations can be done effortlessly.

Initially, we will need to revisit our cost function now that we are dealing with a neural network with more than one output. Let us now use the mean squared error as our cost function.

$$J\,(theta) = \frac{1}{2m}\sum_{i}\left(y_i - \mathrm{a}_i^{(2)}\right)^2 \tag{4}$$

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

Now that we have corrected our cost function, we can look at how changing a parameter affects the cost function. Specifically, Let us calculate $\frac{\partial J(\theta)}{\partial \theta_{11}^{(1)}}$ in this example. Looking at the diagram, $\theta_{11}^{(1)}$ affects the output for both $a_1^{(3)}$ and $a_2^{(3)}$. Because our cost function is a summation of individual costs for each output, we can calculate the derivative chain for each path and simply add them together.



The derivative chain for the blue path is:

$$\left(\frac{\partial J(\theta)}{\partial a_1^{(3)}}\right)\left(\frac{\partial a_1^{(3)}}{\partial z_1^{(3)}}\right)\left(\frac{\partial z_1^{(3)}}{\partial a_1^{(2)}}\right)\left(\frac{\partial a_1^{(2)}}{\partial z_1^{(2)}}\right)\left(\frac{\partial z_1^{(2)}}{\partial \theta_{11}^{(1)}}\right) \tag{5}$$

The derivative chain for the orange path is:

$$\left(\frac{\partial J(\theta)}{\partial a_2^{(3)}}\right)\left(\frac{\partial a_2^{(3)}}{\partial z_2^{(3)}}\right)\left(\frac{\partial z_2^{(3)}}{\partial a_1^{(2)}}\right)\left(\frac{\partial a_1^{(2)}}{\partial z_1^{(2)}}\right)\left(\frac{\partial z_1^{(2)}}{\partial \theta_{11}^{(1)}}\right) \tag{6}$$

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

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

The remainder of the partial derivatives is provided 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.

4.4 Layer 2 Parameters

$$\frac{\partial J(\theta)}{\partial \theta_{11}^{(2)}} = \left(\frac{\partial J(\theta)}{\partial a_1^{(3)}}\right)\left(\frac{\partial a_1^{(3)}}{\partial z_1^{(3)}}\right)\left(\frac{\partial z_1^{(3)}}{\partial \theta_{11}^{(2)}}\right) \tag{7}$$

$$\frac{\partial J(\theta)}{\partial \theta_{12}^{(2)}} = \left(\frac{\partial J(\theta)}{\partial a_1^{(3)}}\right)\left(\frac{\partial a_1^{(3)}}{\partial z_1^{(3)}}\right)\left(\frac{\partial z_1^{(3)}}{\partial \theta_{12}^{(2)}}\right) \tag{8}$$

$$\frac{\partial J(\theta)}{\partial \theta_{21}^{(2)}} = \left(\frac{\partial J(\theta)}{\partial a_2^{(3)}}\right)\left(\frac{\partial a_2^{(3)}}{\partial z_2^{(3)}}\right)\left(\frac{\partial z_2^{(3)}}{\partial \theta_{21}^{(2)}}\right) \tag{9}$$

$$\frac{\partial J(\theta)}{\partial \theta_{22}^{(2)}} = \left(\frac{\partial J(\theta)}{\partial a_2^{(3)}}\right)\left(\frac{\partial a_2^{(3)}}{\partial z_2^{(3)}}\right)\left(\frac{\partial z_2^{(3)}}{\partial \theta_{22}^{(2)}}\right) \tag{10}$$

4.5 Layer 1 Parameters

$$\frac{\partial J(\theta)}{\partial \theta_{11}^{(1)}} = \left(\frac{\partial J(\theta)}{\partial \mathrm{a}_1^{(3)}}\right)\left(\frac{\partial \mathrm{a}_1^{(3)}}{\partial z_1^{(3)}}\right)\left(\frac{\partial z_1^{(3)}}{\partial \mathrm{a}_1^{(2)}}\right)\left(\frac{\partial \mathrm{a}_1^{(2)}}{\partial z_1^{(2)}}\right)\left(\frac{\partial z_1^{(2)}}{\partial \theta_{11}^{(1)}}\right) + \left(\frac{\partial J(\theta)}{\partial \mathrm{a}_2^{(3)}}\right)\left(\frac{\partial \mathrm{a}_2^{(3)}}{\partial z_2^{(3)}}\right)\left(\frac{\partial z_2^{(3)}}{\partial \mathrm{a}_1^{(2)}}\right)\left(\frac{\partial \mathrm{a}_1^{(2)}}{\partial z_1^{(2)}}\right)\left(\frac{\partial z_1^{(2)}}{\partial \theta_{11}^{(1)}}\right) \tag{11}$$

$$\frac{\partial J(\theta)}{\partial \theta_{12}^{(1)}} = \left(\frac{\partial J(\theta)}{\partial \mathrm{a}_1^{(3)}}\right)\left(\frac{\partial \mathrm{a}_1^{(3)}}{\partial z_1^{(3)}}\right)\left(\frac{\partial z_1^{(3)}}{\partial \mathrm{a}_1^{(2)}}\right)\left(\frac{\partial \mathrm{a}_1^{(2)}}{\partial z_1^{(2)}}\right)\left(\frac{\partial z_1^{(2)}}{\partial \theta_{12}^{(1)}}\right) + \left(\frac{\partial J(\theta)}{\partial \mathrm{a}_2^{(3)}}\right)\left(\frac{\partial \mathrm{a}_2^{(3)}}{\partial z_2^{(3)}}\right)\left(\frac{\partial z_2^{(3)}}{\partial \mathrm{a}_1^{(2)}}\right)\left(\frac{\partial \mathrm{a}_1^{(2)}}{\partial z_1^{(2)}}\right)\left(\frac{\partial z_1^{(2)}}{\partial \theta_{12}^{(1)}}\right) \tag{12}$$

$$\frac{\partial J(\theta)}{\partial \theta_{21}^{(1)}} = \left(\frac{\partial J(\theta)}{\partial \mathrm{a}_1^{(3)}}\right)\left(\frac{\partial \mathrm{a}_1^{(3)}}{\partial z_1^{(3)}}\right)\left(\frac{\partial z_1^{(3)}}{\partial \mathrm{a}_2^{(2)}}\right)\left(\frac{\partial \mathrm{a}_2^{(2)}}{\partial z_2^{(2)}}\right)\left(\frac{\partial z_2^{(2)}}{\partial \theta_{21}^{(1)}}\right) + \left(\frac{\partial J(\theta)}{\partial \mathrm{a}_2^{(3)}}\right)\left(\frac{\partial \mathrm{a}_2^{(3)}}{\partial z_2^{(3)}}\right)\left(\frac{\partial z_2^{(3)}}{\partial \mathrm{a}_2^{(2)}}\right)\left(\frac{\partial \mathrm{a}_2^{(2)}}{\partial z_2^{(2)}}\right)\left(\frac{\partial z_2^{(2)}}{\partial \theta_{21}^{(1)}}\right) \tag{13}$$

$$\frac{\partial J(\theta)}{\partial \theta_{22}^{(1)}} = \left(\frac{\partial J(\theta)}{\partial \mathrm{a}_1^{(3)}}\right)\left(\frac{\partial \mathrm{a}_1^{(3)}}{\partial z_1^{(3)}}\right)\left(\frac{\partial z_1^{(3)}}{\partial \mathrm{a}_2^{(2)}}\right)\left(\frac{\partial \mathrm{a}_2^{(2)}}{\partial z_2^{(2)}}\right)\left(\frac{\partial z_2^{(2)}}{\partial \theta_{22}^{(1)}}\right) + \left(\frac{\partial J(\theta)}{\partial \mathrm{a}_2^{(3)}}\right)\left(\frac{\partial \mathrm{a}_2^{(3)}}{\partial z_2^{(3)}}\right)\left(\frac{\partial z_2^{(3)}}{\partial \mathrm{a}_2^{(2)}}\right)\left(\frac{\partial \mathrm{a}_2^{(2)}}{\partial z_2^{(2)}}\right)\left(\frac{\partial z_2^{(2)}}{\partial \theta_{22}^{(1)}}\right) \tag{14}$$

That is a lot of computations. 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 are smart about how we approach this problem, we can dramatically reduce the computational cost of training. Further, it would really be a painful task if we had to manually calculate the derivative chains for every parameter. Let us look at what we have done so far and see if we can generalize a method to these crazy computations.

5 Backpropagation

Backpropagation is simply a method for calculating the partial derivative of the cost function with respect to all of the parameters. The actual optimization of parameters (training) is done by gradient descent or another more advanced optimization technique.

Generally, we established that you can calculate the partial derivatives for layer l by combining δ terms of the next layer forward with the activations of the current layer.

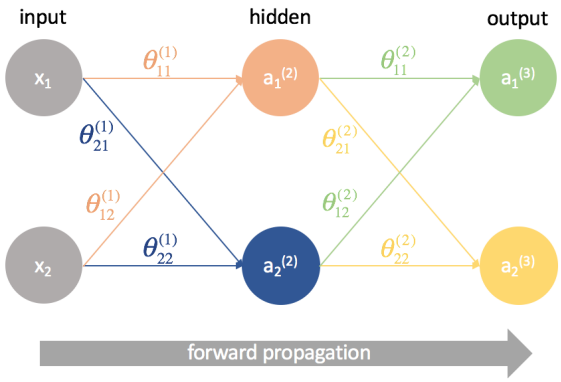
$$\frac{\partial J(\theta)}{\partial \theta_{ij}^{(l)}} = \left(\delta^{(l+1)}\right)^T a^{(l)} \tag{15}$$

In the Appendix A, we showed 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, δ, 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. Check it yourself if necessary.

5.1 Backpropagation visualized

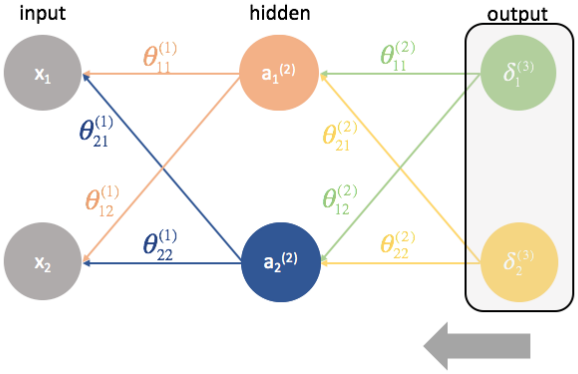
Before defining the formal method for backpropagation, I'd like to provide a visualization of the process.

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



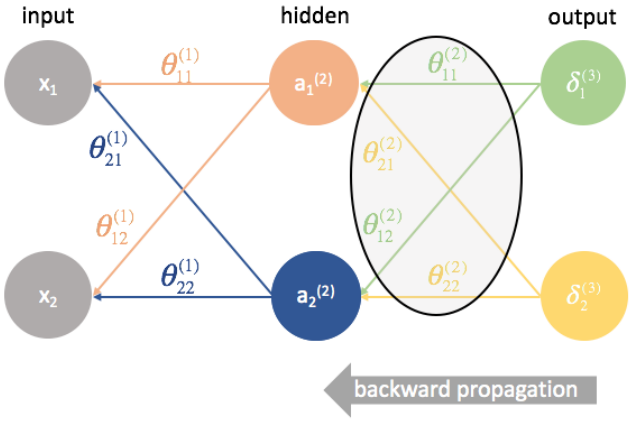
Next, we compute the $\delta^{(3)}$ terms for the last layer in the network. Remember, these δ terms consist of all of the partial derivatives that will be used again in calculating parameters for layers further back. In practice, we typically refer to δ as the "error" term.

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



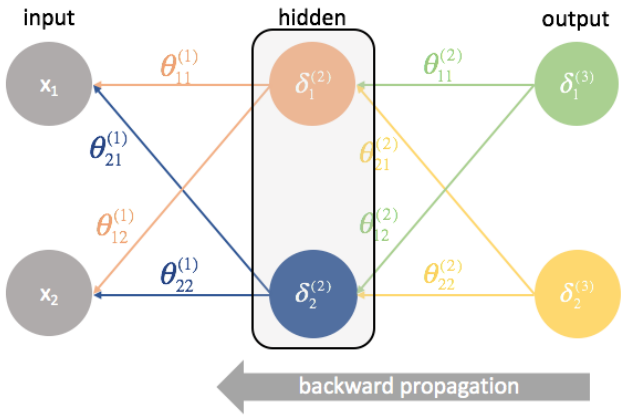
$\Theta^{(2)}$ is the matrix of parameters that connects layer 2 to 3. We multiply the error from the third layer by the inputs in the second layer to calculate our partial derivatives for this set of parameters.

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



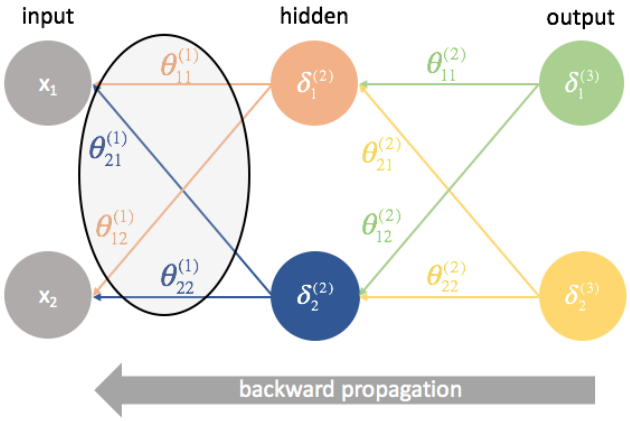
Next, we "send back" the "error" terms in the exact same manner that we "send forward" the inputs to a neural network. The only difference is that time, we are starting from the back and we are feeding an error term, layer by layer, backwards through the network. Hence the name: backpropagation. The act of "sending back our error" is accomplished via the expression $\delta^{(3)}\Theta^{(2)}$.

$$\delta^{(2)} = \delta^{(3)}\Theta^{(2)} f'\left(a^{(2)}\right)$$



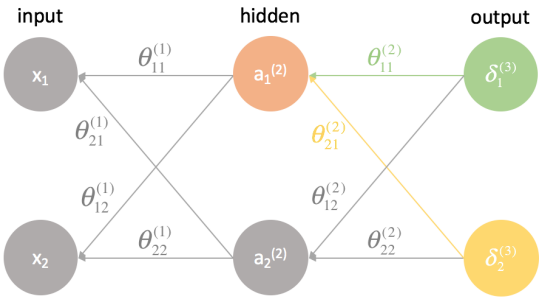
$\Theta^{(1)}$ is the matrix of parameters that connects layer 1 to 2. We multiply the error from the second layer by the inputs in the first layer to calculate our partial derivatives for this set of parameters.

$$\frac{\partial J(\theta)}{\partial \theta_{ij}^{(1)}} = \left(\delta^{(2)}\right)^T x$$



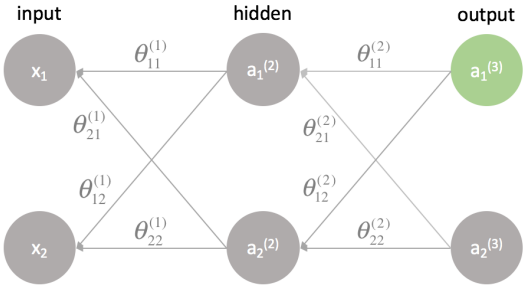
For every layer except for the last, the "error" term is a linear combination of parameters connecting to the next layer (moving forward through the network) and the "error" terms of that next layer. This is true for all of the hidden layers, since we don't compute an "error" term for the inputs.

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



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

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



5.2 Formalized Backpropagation Algorithm

Here, I'll present a practical method for implementing backpropagation through a network of layers $l = 1, 2, \ldots, L$.

1. Perform forward propagation.
2. Compute the δ term for the output layer.

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

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

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

4. Go back one layer.

$$l = l - 1$$

5. Compute the δ term for the current hidden layer.

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

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

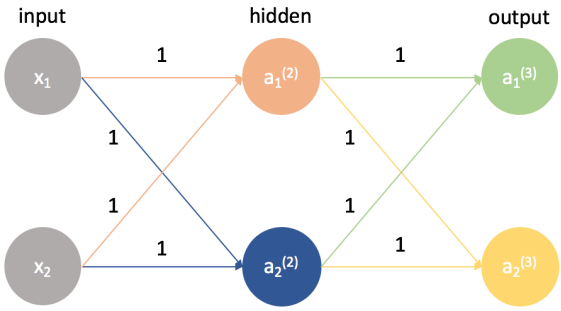
$$\frac{\partial J(\theta)}{\partial \theta_{ij}^{(l)}} = \left(\delta^{(l+1)}\right)^T a^{(l)}$$

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

5.3 Revisiting the weights initialization

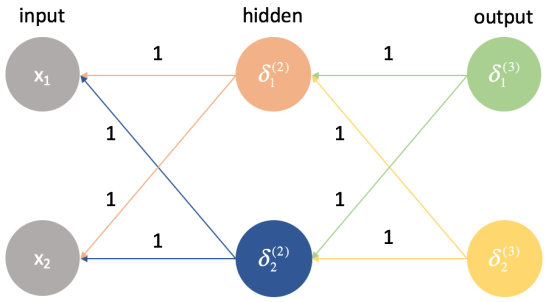
When we started, I proposed that we just randomly initialize our weights in order to have a place to start. This allowed us to perform forward propagation, compare the outputs to the expected values, and compute the cost of our model.

It is actually very important that we initialize our weights to random values 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.



$$a_1^{(2)} = x_1 + x_2 = a_2^{(2)}$$

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



$$\delta_1^{(2)} = \left(\delta_1^{(3)} + \delta_2^{(3)}\right) f'\left(a^{(2)}\right) = \delta_2^{(2)}$$

Further, because we calculate the partial derivatives in any given layer by combining δ values and activations, all of the partial derivatives in any given layer would be identical. As such, the weights would update symmetrically in gradient descent and multiple neurons in any layer would be useless. This obviously would not be a very helpful neural network.

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 will assign each parameter to a random value in $[-\varepsilon, \varepsilon]$ where ε is some value close to zero.

5.4 Putting it all together

After we have 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, η.

$$\Delta\theta_i = -\eta \frac{\partial J(\theta)}{\partial \theta_i}$$

we will use this formula to update each of the weights, recompute forward propagation with the new weights, backpropagate the error, and calculate the next weight update. This process continues until we have 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.

6 Training a Neural Network using Backpropagation

Let's see how we can implement Backpropagation in Python in a step-by-step manner. First of all, we need to import all the necessary libraries.

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

We will be working with a very simple dataset today i.e the iris dataset. We will load the dataset using `load_iris()` function, which is part of the scikit-learn library. The dataset consists of three main classes. We will divide them into target variables and features.

```
# Loading dataset
data = load_iris()
# Dividing the dataset into target variable and features
X=data.data
y=data.target
```

Now we will split the dataset into training and test sets. We will use the function `train_test_split()`. The function takes three parameters: the features, target, and size of the test set.

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                            test_size=20, random_state=4)
y_train = y_train.reshape((y_train.shape[0], 1))
y_test = y_test.reshape((y_test.shape[0], 1))
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape
```

Output:

```
(130, 4)
(20, 4)
(130, 1)
(20, 1)
```

Now in the next step, we have to start initializing the hyperparameters. We will input the learning rate, iterations, input size, number of hidden layers, and number of output layers.

```
learning_rate = 0.1
iterations = 5000
N = y_train.size

# Input features
input_size = 4

# Hidden layers
hidden_size = 2

# Output layer
output_size = 3

results = pd.DataFrame(columns=["mse", "accuracy"])
```

Randomly initialize the weights.

```
np.random.seed(10)

# Hidden layer
W1 = np.random.normal(scale=0.5, size=(input_size, hidden_size))

# Output layer
W2 = np.random.normal(scale=0.5, size=(hidden_size , output_size))
```

Now we will create helper functions such as mean squared error, accuracy and sigmoid.

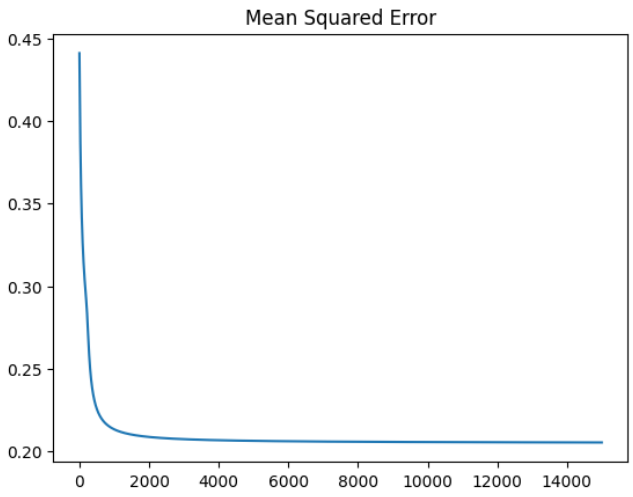
```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def mean_squared_error(y_pred, y_true):
    return ((y_pred - y_true)**2).sum() / (2*y_pred.size)

def accuracy(y_pred, y_true):
    acc = y_pred.argmax(axis=1) == y_true.argmax(axis=1)
    return acc.mean()
```

Next is building the Backpropogation model. We will create a for loop for a given number of iterations and will update the weights in each iteration. The model will go through three phases feedforward propagation, the error calculation phase, and the backpropagation phase.

```
for itr in range(iterations):

    # Implementing feedforward propagation on hidden layer
    Z1 = np.dot(X_train, W1)
    A1 = sigmoid(Z1)

    # Implementing feed forward propagation on output layer
    Z2 = np.dot(A1, W2)
    A2 = sigmoid(Z2)


    # Calculating the error
    mse = mean_squared_error(A2, y_train)
    acc = accuracy(A2, y_train)

    results = results.append({"mse":mse, "accuracy":acc}, ignore_index=True)



    # Backpropagation phase
    E1 = A2 - y_train
    dW1 = E1 * A2 * (1 - A2)

    E2 = np.dot(dW1, W2.T)
    dW2 = E2 * A1 * (1 - A1)


    # Updating the weights
    W2_update = np.dot(A1.T, dW1) / N
    W1_update = np.dot(X_train.T, dW2) / N

    W2 = W2 - learning_rate * W2_update
    W1 = W1 - learning_rate * W1_update
```

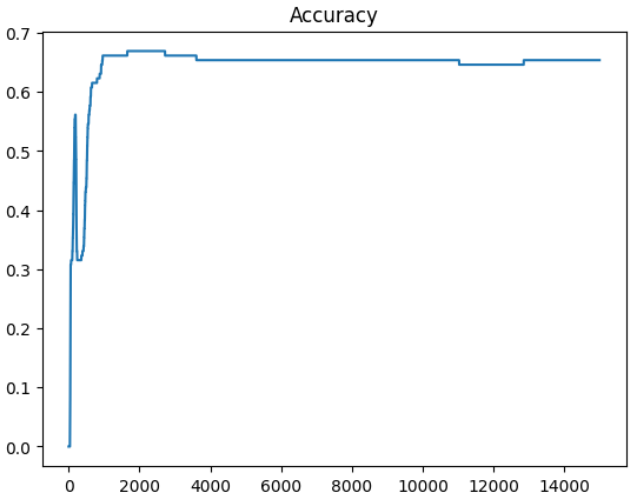
Now we will plot the mean squared error using pandas plot() function.

```
results.mse.plot(title="Mean Squared Error")
```



And accuracy using the pandas plot() function.

```
results.accuracy.plot(title="Accuracy")
```



Now we will calculate the accuracy of the model.

```
Z1 = np.dot(X_test, W1)
A1 = sigmoid(Z1)

Z2 = np.dot(A1, W2)
A2 = sigmoid(Z2)

acc = accuracy(A2, y_test)
print("Accuracy: {}".format(acc))
```

Output:

```
Accuracy: 0.85
```

7 Lab Exercises

Please code yourself and write a report based on your findings:

1. Experiment with a different number of hidden layer neurons and report your results.
2. Experiment with different learning rates and report your results.
3. Experiment with a different number of iterations and report your results.
4. Modify the above code and Experiment with MNIST dataset (google yourself). Report your optimal Neural Network and analyze the experimental results.

A Appendix: Generalizing Method

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

A.1 Layer 2 Parameters

Let us analyze the following expressions; You are strongly encouraged to solve the partial derivatives yourself as we go along to convince yourself of the logic.

$$\frac{\partial J(\theta)}{\partial \theta_{11}^{(2)}} = \left(\frac{\partial J(\theta)}{\partial \mathrm{a}_1^{(3)}}\right)\left(\frac{\partial \mathrm{a}_1^{(3)}}{\partial z_1^{(3)}}\right)\left(\frac{\partial z_1^{(3)}}{\partial \theta_{11}^{(2)}}\right) \tag{16}$$

$$\frac{\partial J(\theta)}{\partial \theta_{12}^{(2)}} = \left(\frac{\partial J(\theta)}{\partial \mathrm{a}_1^{(3)}}\right)\left(\frac{\partial \mathrm{a}_1^{(3)}}{\partial z_1^{(3)}}\right)\left(\frac{\partial z_1^{(3)}}{\partial \theta_{12}^{(2)}}\right) \tag{17}$$

$$\frac{\partial J(\theta)}{\partial \theta_{21}^{(2)}} = \left(\frac{\partial J(\theta)}{\partial \mathrm{a}_2^{(3)}}\right)\left(\frac{\partial \mathrm{a}_2^{(3)}}{\partial z_2^{(3)}}\right)\left(\frac{\partial z_2^{(3)}}{\partial \theta_{21}^{(2)}}\right) \tag{18}$$

$$\frac{\partial J(\theta)}{\partial \theta_{22}^{(2)}} = \left(\frac{\partial J(\theta)}{\partial \mathrm{a}_2^{(3)}}\right)\left(\frac{\partial \mathrm{a}_2^{(3)}}{\partial z_2^{(3)}}\right)\left(\frac{\partial z_2^{(3)}}{\partial \theta_{22}^{(2)}}\right) \tag{19}$$

Here, 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. It is not immediately evident why this would be helpful, but we will see as we go backward another layer why this is useful. In the literature, this expression is often referred to as the "error" term which we use to "send back error from the output throughout the network". We will see why this is the case soon. Each neuron in the network will have a corresponding δ term that we will solve for.

$$\delta_i^{(3)} = \frac{1}{m}\left(y_i - a_i^{(3)}\right) f'\left(a^{(3)}\right) \tag{20}$$

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.

Note: the first two partial derivative terms seem concerned with the first output neuron (neuron 1 in layer 3) while the last two partial derivative terms seem concerned with the second output neuron (neuron 2 in layer 3). This is evident in the $\frac{\partial J(\theta)}{\partial \mathrm{a}_i^{(3)}}$ term. Let us use this knowledge to rewrite the partial derivatives using the δ expression we defined above.

$$\frac{\partial J(\theta)}{\partial \theta_{11}^{(2)}} = \delta_1^{(3)}\left(\frac{\partial z_1^{(3)}}{\partial \theta_{11}^{(2)}}\right) \tag{21}$$

$$\frac{\partial J(\theta)}{\partial \theta_{12}^{(2)}} = \delta_1^{(3)}\left(\frac{\partial z_1^{(3)}}{\partial \theta_{12}^{(2)}}\right) \tag{22}$$

$$\frac{\partial J(\theta)}{\partial \theta_{21}^{(2)}} = \delta_2^{(3)}\left(\frac{\partial z_2^{(3)}}{\partial \theta_{21}^{(2)}}\right) \tag{23}$$

$$\frac{\partial J(\theta)}{\partial \theta_{22}^{(2)}} = \delta_2^{(3)}\left(\frac{\partial z_2^{(3)}}{\partial \theta_{22}^{(2)}}\right) \tag{24}$$

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

$$\frac{\partial J(\theta)}{\partial \theta_{11}^{(2)}} = \delta_1^{(3)} \mathrm{a}_1^{(2)} \tag{25}$$

$$\frac{\partial J(\theta)}{\partial \theta_{12}^{(2)}} = \delta_1^{(3)} \mathrm{a}_2^{(2)} \tag{26}$$

$$\frac{\partial J(\theta)}{\partial \theta_{21}^{(2)}} = \delta_2^{(3)} \mathrm{a}_1^{(2)} \tag{27}$$

$$\frac{\partial J(\theta)}{\partial \theta_{22}^{(2)}} = \delta_2^{(3)} \mathrm{a}_2^{(2)} \tag{28}$$

It appears that we are combining the "error" terms with activations from the previous layer to calculate each partial derivative. It is also interesting to note that the indices j and k for θ_{jk} match the combined indices of $\delta_j^{(3)}$ and $\mathrm{a}_k^{(2)}$.

$$\frac{\partial J(\theta)}{\partial \theta_{jk}^{(2)}} = \delta_j^{(3)} \mathrm{a}_k^{(2)} \tag{29}$$

Let us 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.

$$\delta^{(3)} = \begin{bmatrix} y_1 - a_1^{(3)} \\ y_2 - a_2^{(3)} \\ \cdots \\ y_j - a_j^{(3)} \end{bmatrix} f'(a^{(3)}) \tag{30}$$

$\mathrm{a}^{(2)}$ 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 is the vector of inputs to the output layer.

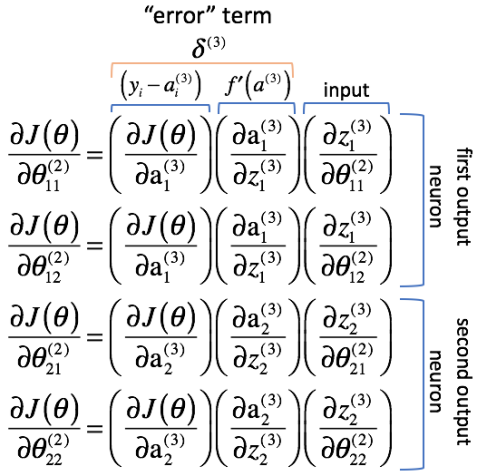
$$a^{(2)} = \begin{bmatrix} a_1^{(2)} & a_2^{(2)} & \cdots & a_k^{(2)} \end{bmatrix} \tag{31}$$

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

$$\frac{\partial J(\theta)}{\partial \theta_{ij}^{(2)}} = \begin{bmatrix} \delta_1^{(3)} \\ \delta_2^{(3)} \end{bmatrix} \begin{bmatrix} a_1^{(2)} & a_2^{(2)} \end{bmatrix} = \begin{bmatrix} \delta_1^{(3)} a_1^{(2)} & \delta_1^{(3)} a_2^{(2)} \\ \delta_2^{(3)} a_1^{(2)} & \delta_2^{(3)} a_2^{(2)} \end{bmatrix} \tag{32}$$

To summarize, see the graphic below.

Layer 2



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

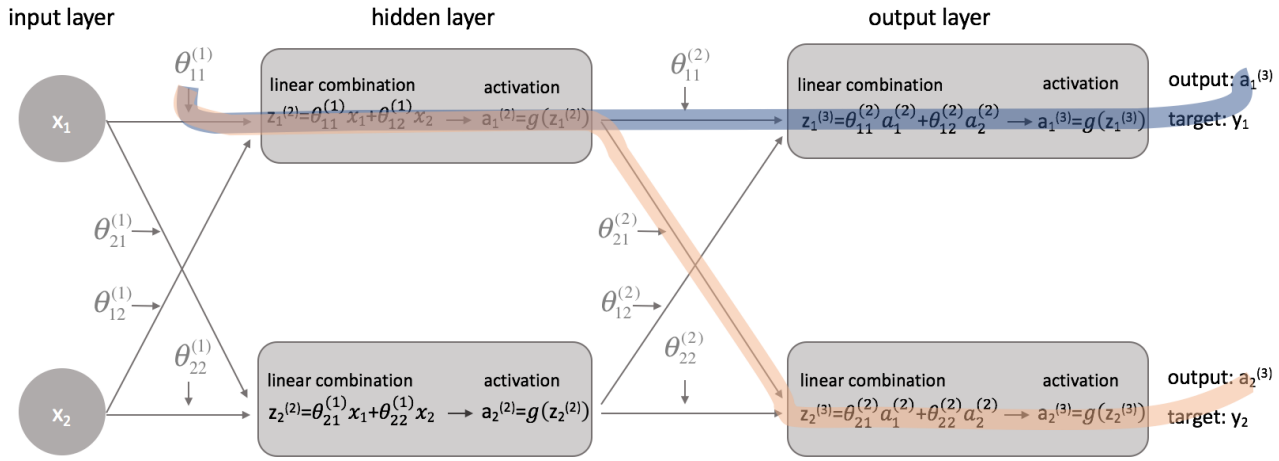
$$\frac{\partial J(\theta)}{\partial \theta_{ij}^{(2)}} = \begin{bmatrix} \delta_1^{(3)} \\ \delta_2^{(3)} \end{bmatrix} \begin{bmatrix} a_1^{(2)} & a_2^{(2)} \end{bmatrix} = \begin{bmatrix} \delta_1^{(3)} a_1^{(2)} & \delta_1^{(3)} a_2^{(2)} \\ \delta_2^{(3)} a_1^{(2)} & \delta_2^{(3)} a_2^{(2)} \end{bmatrix}$$

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 are concerned with the difference between the expected and actual outputs. The δ term on the right has the full derivative expression.

A.2 Layer 1 Parameters

Now let us take a look and see what is 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.



This results in a partial derivative of the cost function with respect to a parameter now becoming a summation of different chains. Specifically, we will have a derivative chain for every δ we calculated in the next layer forward. Remember, we started at the end of the network and are working our way backward through the network. Thus, the next layer forward represents the δ values that we previously calculated.

Layer 1

$$\frac{\partial J(\theta)}{\partial \theta_{11}^{(1)}} = \underbrace{\left(\frac{\partial J(\theta)}{\partial a_1^{(3)}}\right)}_{\delta_1^{(3)}}\left(\frac{\partial a_1^{(3)}}{\partial z_1^{(3)}}\right)\underbrace{\left(\frac{\partial z_1^{(3)}}{\partial a_1^{(2)}}\right)}_{\theta^{(2)}}\underbrace{\left(\frac{\partial a_1^{(2)}}{\partial z_1^{(2)}}\right)}_{f'(a^{(2)})}\underbrace{\left(\frac{\partial z_1^{(2)}}{\partial \theta_{11}^{(1)}}\right)}_{\text{input}} + \underbrace{\left(\frac{\partial J(\theta)}{\partial a_2^{(3)}}\right)}_{\delta_2^{(3)}}\left(\frac{\partial a_2^{(3)}}{\partial z_2^{(3)}}\right)\underbrace{\left(\frac{\partial z_2^{(3)}}{\partial a_1^{(2)}}\right)}_{\theta^{(2)}}\underbrace{\left(\frac{\partial a_1^{(2)}}{\partial z_1^{(2)}}\right)}_{f'(a^{(2)})}\underbrace{\left(\frac{\partial z_1^{(2)}}{\partial \theta_{11}^{(1)}}\right)}_{\text{input}}$$

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

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

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

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

As we did for layer 2, let us 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.

we will also redefine δ for all layers excluding the output layer to include this combination of weighted errors.

$$\delta_j^{(l)} = f'\left(a^{(l)}\right)\sum_{i=1}^{n}\delta_i^{(l+1)}\theta_{ij}^{(l)} \tag{33}$$

We could write this more succinctly using matrix expressions.

$$\delta^{(l)} = \delta^{(l+1)}\Theta^{(l)}f'\left(a^{(l)}\right) \tag{34}$$

Note: It is standard notation to denote vectors with lowercase letters and matrices as uppercase letters. Thus, θ represents a vector while Θ 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 us 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 $\delta_j^{(3)}$.

$$\frac{\partial J(\theta)}{\partial \theta_{11}^{(1)}} = \delta_1^{(3)}\left(\frac{\partial z_1^{(3)}}{\partial a_1^{(2)}}\right)\left(\frac{\partial a_1^{(2)}}{\partial z_1^{(2)}}\right)\left(\frac{\partial z_1^{(2)}}{\partial \theta_{11}^{(1)}}\right) + \delta_2^{(3)}\left(\frac{\partial z_2^{(3)}}{\partial a_1^{(2)}}\right)\left(\frac{\partial a_1^{(2)}}{\partial z_1^{(2)}}\right)\left(\frac{\partial z_1^{(2)}}{\partial \theta_{11}^{(1)}}\right) \tag{35}$$

Further, we established the the third columns of each derivative chain was a parameter that acted to weight each of the respective δ terms. We also established that the fourth column was the derivative of the activation function.

$$\frac{\partial J(\theta)}{\partial \theta_{11}^{(1)}} = \delta_1^{(3)}\theta_{11}^{(2)}f'\left(a^{(2)}\right)\left(\frac{\partial z_1^{(2)}}{\partial \theta_{11}^{(1)}}\right) + \delta_2^{(3)}\theta_{21}^{(2)}f'\left(a^{(2)}\right)\left(\frac{\partial z_1^{(2)}}{\partial \theta_{11}^{(1)}}\right) \tag{36}$$

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

$$\frac{\partial J(\theta)}{\partial \theta_{11}^{(1)}} = \left(\frac{\partial z_1^{(2)}}{\partial \theta_{11}^{(1)}}\right)\left(\delta_1^{(3)}\theta_{11}^{(2)} f'\left(a^{(2)}\right) + \delta_2^{(3)}\theta_{21}^{(2)} f'\left(a^{(2)}\right)\right) \tag{37}$$

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

$$\frac{\partial J(\theta)}{\partial \theta_{11}^{(1)}} = \left(\frac{\partial z_1^{(2)}}{\partial \theta_{11}^{(1)}}\right)\left(\delta_1^{(2)}\right) \tag{38}$$

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

$$\frac{\partial J(\theta)}{\partial \theta_{11}^{(1)}} = \delta_1^{(2)} x_1 \tag{39}$$

Again let us 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 layer (layer 2). We can calculate $\delta^{(2)}$ as the weighted combination of errors from layer 3, multiplied by the derivative of the activation function used in layer 2.

$$\delta^{(2)} = \begin{bmatrix}\delta_1^{(3)} & \delta_2^{(3)}\end{bmatrix}\begin{bmatrix}\theta_{11}^{(2)} & \theta_{12}^{(2)} \\ \theta_{21}^{(2)} & \theta_{22}^{(2)}\end{bmatrix} f'(a^{(2)}) = \begin{bmatrix}\delta_1^{(2)} & \delta_2^{(2)}\end{bmatrix} \tag{40}$$

x is a vector of input values.

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

$$\frac{\partial J(\theta)}{\partial \theta_{ij}^{(1)}} = \begin{bmatrix}\delta_1^{(2)} \\ \delta_2^{(2)}\end{bmatrix}\begin{bmatrix}x_1 & x_2\end{bmatrix} = \begin{bmatrix}\delta_1^{(2)} x_1 & \delta_1^{(2)} x_2 \\ \delta_2^{(2)} x_1 & \delta_2^{(2)} x_2\end{bmatrix} \tag{41}$$

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 have 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.

My Notes Date:_____