# *DYOA: We Be Optimizing*

*QEA*

*Fall 2018*

## *What is this about?*

This assignment will give you the tools you need to understand the core mathematical concepts of training a neural network. First, we will be introducing some new concepts as well as expanding upon familiar ones, including

- Neural network terminology
- Multivariable chain rule
- Sigmoid Functions
- Iterative Gradient Descent

## *Learning Goals*

- Learn the basic definitions and properties of neural networks
- Understand the mathematical concepts required to implement iterative gradient descent
- Understand the significance of iterative gradient descent in neural network optimization

## *Terminology of Neural Networks [1 hr]*

According to this MIT article, neural networks are a "means of doing machine learning, in which a computer learns to perform some task by analyzing training examples." These examples are usually hand-labeled in advance (think back to Faces, where the training data set contained a labeled set of photos, so that there was a name that went with every image).

There are many different types of neural networks, with varying levels of complexity. In order to eventually implement a convolutional neural network, it is also important to understand some of the vocabulary that will be commonly used in any online or printed reference.

Spend some time Googling definitions to the following key terms. For each term, write the definition and comment on its relationship to neural networks.

1. Layers
2. Nodes
3. Loss Functions
4. Training Set
5. Weights
6. Biases
7. "Feed-forward"
8. Training
9. Activation function
10. Create a concept map containing these key terms. Draw connections between the different key terms and make links to topics previously covered in QEA and elsewhere.

## *Multivariable Chain Rule [30 minutes]*

Recall the chain rule from your high school Calculus class. The definition likely looked something like this:

If $F(x) = f(g(x))$, then

$$F'(x) = f'(g(x))g'(x) \tag{1}$$

This version of the chain rule works wonderfully for functions of one variable. However, when we begin to tackle functions of more than one variable, a more robust version is required. This "more robust version" is the multivariable chain rule, and it will play an important role when we optimize our neural network.

### *Chain Rule for One Independent Variable*

Suppose $x = g(t)$ and $y = h(t)$ are differentiable functions of $t$, and $z = f(x, y)$, where $z$ is a differentiable function of $x$ and $y$. Then $z = f(x(t), y(t))$ is a differentiable function and[1],

$$\frac{dz}{dt} = \frac{\delta z}{\delta x} \cdot \frac{dx}{dt} + \frac{\delta z}{\delta y} \cdot \frac{dy}{dt} \tag{2}$$

[1] If you are interested in the proof of the multivariable chain rule, check out this resource.

Here is an example, taken from Paul's Online Notes: Suppose $z = xe^{xy}$, $x = t^2$, and $y = t^{-1}$. Then, solving for each of the elements of $\frac{dz}{dt}$ gives

$$\frac{\delta z}{\delta x} = e^{xy} + xye^{xy} \tag{3}$$

$$\frac{\delta z}{\delta y} = x^2 e^{xy} \tag{4}$$

$$\frac{dx}{dt} = 2t \tag{5}$$

$$\frac{dy}{dt} = -t^{-2} \tag{6}$$

Note that equations (3) and (4) are partial derivatives, which you have encountered in Faces Night 4.

Plugging the result into equation (2), we get

$$\frac{dz}{dt} = 2t(e^{xy} + xye^{xy}) - t^{-2}x^2e^{xy} \tag{7}$$

This is the derivative in terms of $x$ and $y$. This derivative is not extremely messy, so it is a good idea to plug back in for $x$ and $y$, so that the final answer is in terms of $t$. Note that this is not always the case, and that depending on the complexity of the derivative, the final answer may be left in terms of $x$ and $y$. Plugging in for $x$ and $y$ gives,

$$\frac{dz}{dt} = 2t(e^t + te^t) - t^{-2}t^4e^t \tag{8}$$

which is just,

$$\frac{dz}{dt} = 2te^t + t^2e^t \tag{9}$$

For each of the following problems, find $\frac{dz}{dt}$.

11. $z(x,y) = x^2y - y^2$, where $x(t) = t^2$, $y(t) = 2t$

12. $z(x,y) = x^2 - 3xy + 2y^2$, where $x(t) = 3\sin 2t$, $y(t) = 4\cos 2t$

This multivariable chain rule can also be extended to situations where $x$ and $y$ are themselves multivariable functions of $u$ and $v$.

*Chain Rule for Two Independent Variables*

Let $x = g(u,v)$ and $y = h(u,v)$ be differentiable functions of $u$, and $v$, and $z = f(x,y)$ be a differentiable function of $x$ and $y$. Then $z = f(g(u,v), h(u,v))$ is a differentiable of $u$ and $v$ given by,

$$\frac{\delta z}{\delta u} = \frac{\delta z}{\delta x} \cdot \frac{\delta x}{\delta u} + \frac{\delta z}{\delta y} \cdot \frac{\delta y}{\delta u} \tag{10}$$

$$\frac{\delta z}{\delta v} = \frac{\delta z}{\delta x} \cdot \frac{\delta x}{\delta v} + \frac{\delta z}{\delta y} \cdot \frac{\delta y}{\delta v} \tag{11}$$

13. Using the formula above and the information presented in the previous section, find $\frac{\delta z}{\delta u}$ and $\frac{\delta z}{\delta v}$ for $z = e^{2r} sin(3\theta)$, where $r = st - t^2$, $\theta = \sqrt{s^2 + t^2}$.

According to Wikipedia, sigmoid functions are mathematical functions that have a characteristic "s"-shaped curve. It is very common to see sigmoid functions used as activation functions in neural networks.

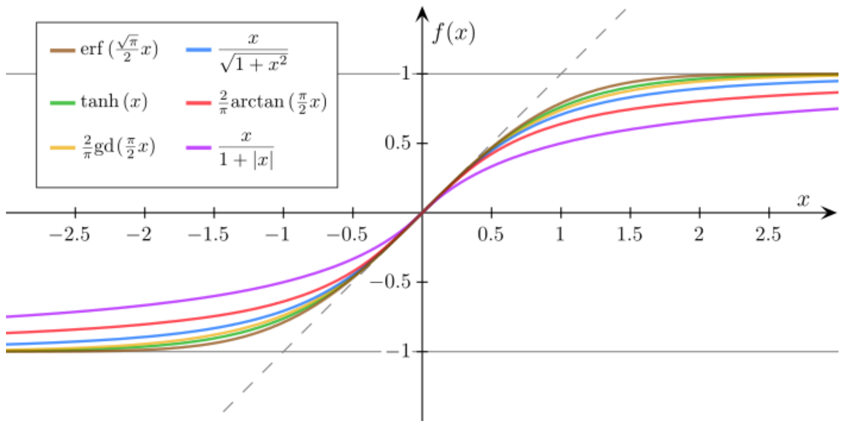

Figure 1: Graph of various sigmoid functions. source.

You can clearly see the "s"-shaped curve present in the sigmoid functions in figure 1.

One of the most common sigmoid functions is the logistic function, given by the equation,

$$s(x) = \frac{1}{1+e^{-x}} \tag{12}$$

14. Hmm... Equation (12) looks quite similar to some formulas we spent a lot of time with in other units and other classes. Where else have we seen cases of this logistic function before? List the relevant concepts and the corresponding courses and/or units.

15. What is the domain and range of (12)?

Why are sigmoids a popular choice for neural networks? Well, it turns out that the derivative of (12) is,

$$s'(x) = s(x) \cdot [1 - s(x)] \tag{13}$$

where $s(x)$ is the sigmoid function[2]. As you can see, the derivative is made up of simple multiplication and subtraction operations. Furthermore, notice that if you have the value of the sigmoid at a point, finding its derivative at that point requires only basic arithmetic operations.

[2] If you are interested in seeing the proof, see here.

For the following exercises it is recommended that you use Python's matplotlib library, which is very similar to Matlab.

16. Plot Equation (12) for $-5 \leq x \leq 5$.

17. Plot Equation (13) over the same values of $x$. Comment on any properties of the derivative that you notice.

## *Iterative Gradient Descent [2 hrs]*

We've seen gradient descent a couple of times before. It was first presented during Faces Night 4, and then again as the opposite of gradient ascent back in NEATO Night 3. Recall from Faces, that the gradient of a function $f(x, y)$ is,

$$\bigtriangledown f = \begin{bmatrix} \frac{\delta f}{\delta x} \\ \frac{\delta f}{\delta y} \end{bmatrix} \tag{14}$$

With respect to neural networks, the gradient is used to minimize the loss function that in turn optimizes the network. We will be using gradient descent to optimize a linear least squares model between 2 variables. The following section is based on Paul Ruvolo's Github.

Given a random point cloud such as the one seen in Figure 2, our goal is to find a line of best fit that best represents the data.

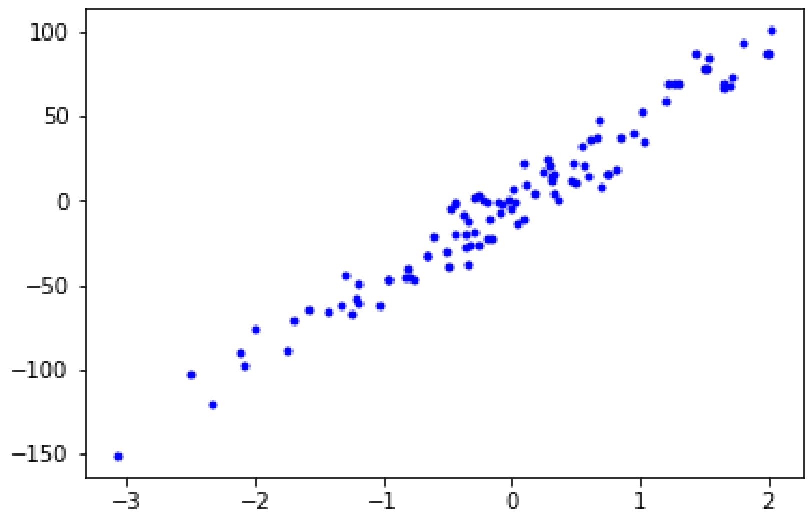

Figure 2: Graph of a randomly generated point cloud

18. Using scikit-learn's `make_regression` function, generate a point cloud with 200 samples, 1 feature, and 10% noise. Set the `coef` argument to True. This function allows us to generate a data set where the actual parameters of the line of best fit are known.

    Given a linear relationship between 2 variables,

$$y = wx + b + \epsilon$$

where $w$ is the slope of a line describing the relationship between $x$ and $y$, $b$ is the y-intercept, and $\epsilon$ is the error term.

The goal of our optimization then is to minimize the $\epsilon$ value and find a value for $w$ that most closely relates $x$ and $y$. In order to do this, we must define an error function that takes in pairs of $(x, y)$ terms and returns an error based on how closely related the $(x, y)$ pair is to the line of fit. To find how closely related a line is to the entire data set, we can sum all the squared differences between the proposed line's output and each point's $y$ coordinate[3]. This gives the equation,

[3] The differences are squared to ensure that they are positive

$$e(w, b) = \frac{1}{N} \sum_{i=1}^{n} ((wx_i + b) - y_i)^2 \tag{15}$$

where $wx_i + b$ is the output of the proposed line of the $i$th data point in the set, $y_i$ is the actual output of the data set at the $i$th point, and $N$ is the total number of points in the data set.

19. Write a function called `error` that takes in parameters `w, b, x, y` and returns the error.

20. Plot this function over a range of $w$ and $b$ values using the data set we generated in the previous exercise as $x$ and $y$ values. (Hint: use `from mpl_toolkits.mplot3d import Axes3D`, then `plot_surface` or `contour`)

In order to find the minimum of the error function $e(w, b)$, we can use gradient descent. We do this by making an initial guess, $w_0$, and incrementally adjusting our guess in the direction of greatest descent until we reach the minimum.

21. Take the partial derivatives of $e(w, b)$ with respect to $w$ and $b$ respectively.

22. Write a function that calculates the gradient of $e(b, w)$ at a point $(x, y)$ given some $(b, w)$. The function should take inputs of `b, w, x, y`.

Another important consideration is the fact that we need to decide how "big of a step" we want to take in the direction of the gradient at any each point. The size of this step is called the **learning rate**, and it is an additional coefficient that scales the gradient.

23. Conceptually, what will happen if the learning rate is too large? Too small? How might these cases affect the efficiency of the network?

Let's simplify things a bit. Now, let's assume that $b = 0$. So now we only need to deal with optimizing $w$. All we need to do is make an initial guess at the $w_0$, calculate the gradient multiplied by some learning rate at that point, move forward, and recalculate. This is the basic idea of how we will train our network. This process can be defined by the following equation,

$$w_{t+1} = w_t - \alpha \frac{de(w_t)}{dw} \tag{16}$$

where $\alpha$ is the learning rate, $w_t$ is the current slope, $\frac{de(w_t)}{dw}$ is the gradient at $w_t$, and $w_{t+1}$ is the new value of $w$.

24. Looking back at (15), calculate the new equation for the gradient with $b = 0$.

25. Write a function that calculates the gradient of the equation that you derived in the previous question. It should take arguments of `w, x, y`. Choose an initial value of $w_0$ and plug it into the function. What do you get?

26. Verify that your function works by using the concept of numerical differentiation (Hint: Choose a small $\Delta w$ value, find the slope between the error at $w_0$ and $(w_0 + \Delta w)$, then compare it to the value you got from the previous question. Does your answer make sense?

27. Using Equation (16), write a function that performs gradient descent for some number of iterations. The function should have parameters of `w, x, y, learningRate, iters` and should return an array of all the errors, as well as the final value of $w$.

28. Plot this function for a range of $\alpha$ values.