Neural Networks

Deep Learning

Brad Quinton, Scott Chin

Overview

- Today we will cover Neural Networks, which are the basis of almost all of Deep Learning today!
- This lecture will cover the basics of NNs and will also try to build an intuition as to why they are so effective for certain problems…
- Like last time, we'll use another case study to motivate our work

Raptors Case Study #2

- After winning “intern of the year” for your work building an AI to select draft picks, you have been offered a full-time position with the Raptors filling the newly created role of **Director of AI**

- Wow, pretty good since you don’t even really understand Basketball. Good thing you took that Deep Learning course, though!

- You quickly accept the position and move to a very expensive but very tiny condo in downtown Toronto

Case Study: First day as Direction of AI

- Coaches: *“Great to have you back. We are really getting great value out of the Draft pick selection AI. But now I’d like to help the players on the court”*

- You: *“Uh, … ya, great.”* (Again wondering about your lack of basketball knowledge)

- Coaches: *“Ya. We need to improve our offense. I’d like you to build an AI that will tell the players if they should shoot or pass when they have the ball”*

Case Study: Understanding the Problem

- Wow. That seems *a lot* harder that selecting Draft picks…. But luckily, now that you are the Director of AI, so you can ask for a bit of help!

- After talking to some of the players and coaching staff you realize the core of the issue:

  Once a player has the ball on offense, they can either try to make a basket immediately or pass to another player (presumably in a better position). If they make the basket, great, but it they miss, the team could lose possession of the ball.

Case Study: Understanding the Problem

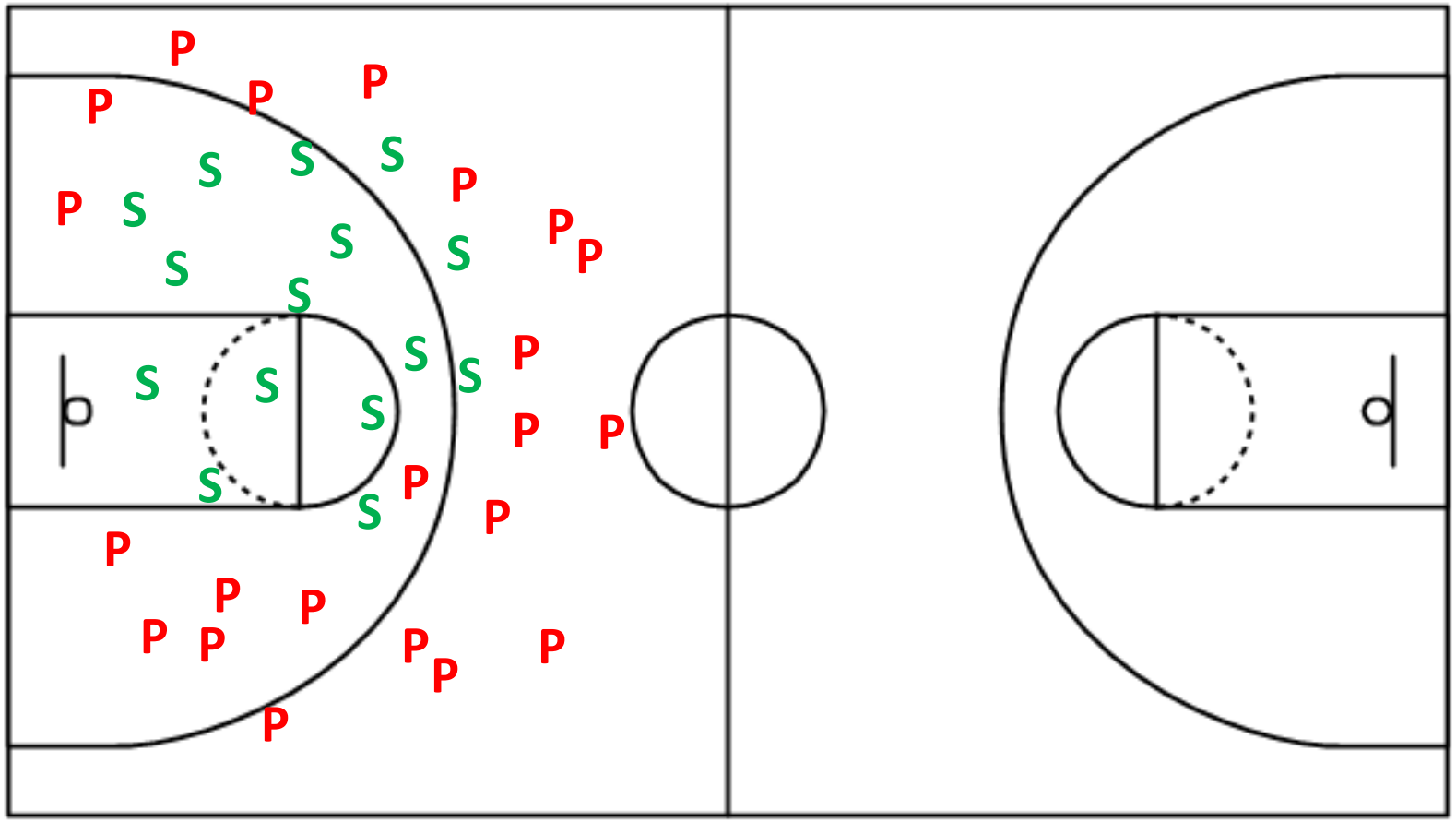
"There are a lot of factors that go into the decision to shoot or pass: current score, time left in the game, opponent, but the biggest factor is really ***position on the court*** *…"* says one of the assistant coaches.

"We actually have a lot of data on this. We carefully analyze every game. So looking backwards we do actually know what the right decision was."

- This sounds promising. Maybe we can figure this out…

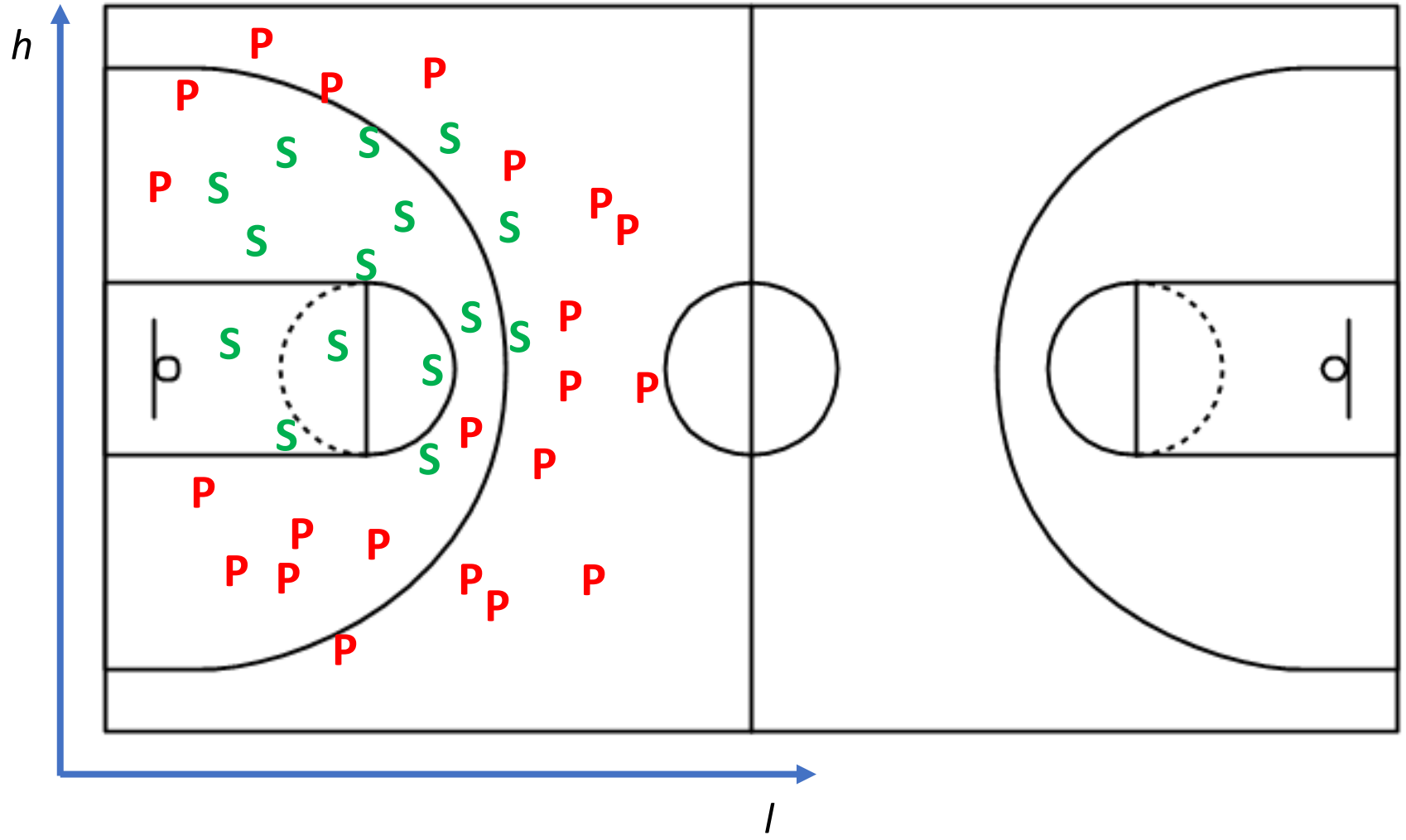
Brad Quinton, Scott Chin

Case Study: Take a Look at the Data



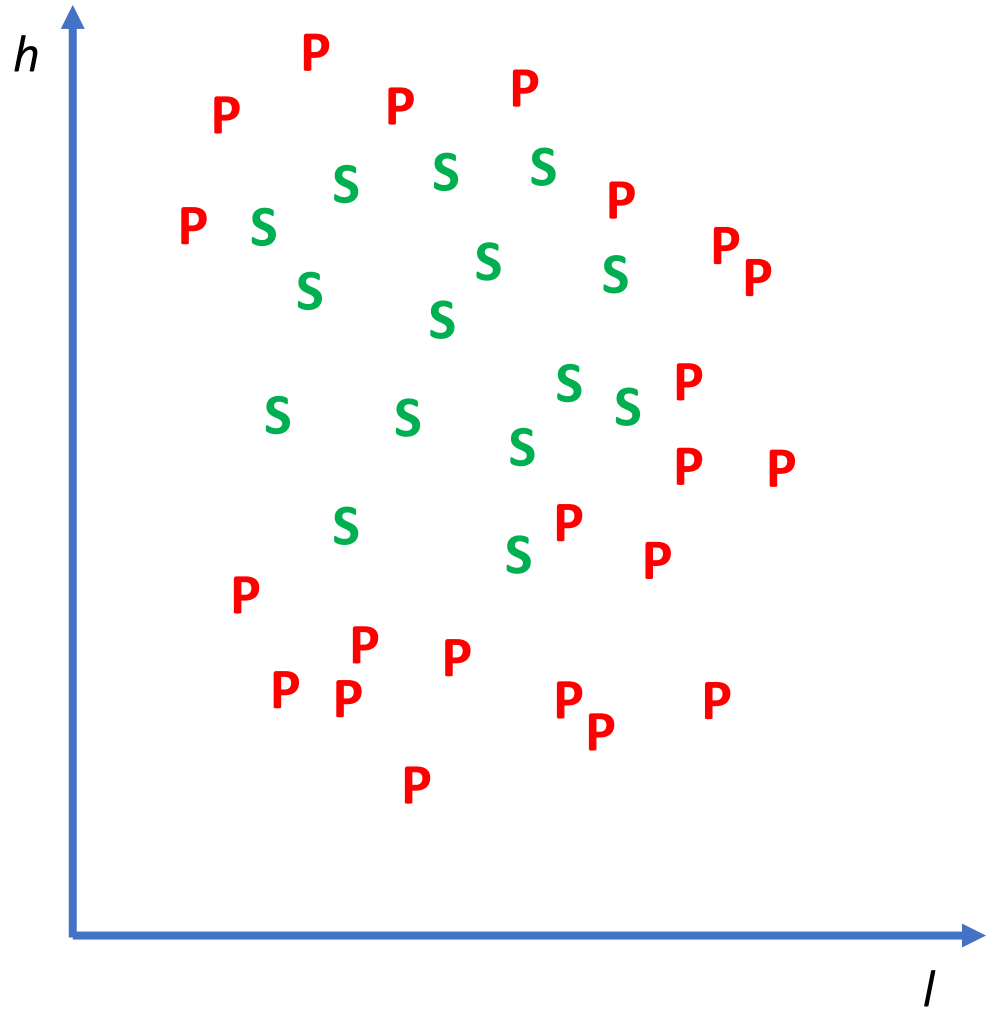
Brad Quinton, Scott Chin

Case Study: Take a Look at the Data



Brad Quinton, Scott Chin

Case Study: Take a Look at the Data



- In some ways this looks pretty similar to what we saw when we were looking to build our Draft Pick selection AI
- When we look at two of the (many) inputs features, there is certainly a pattern…

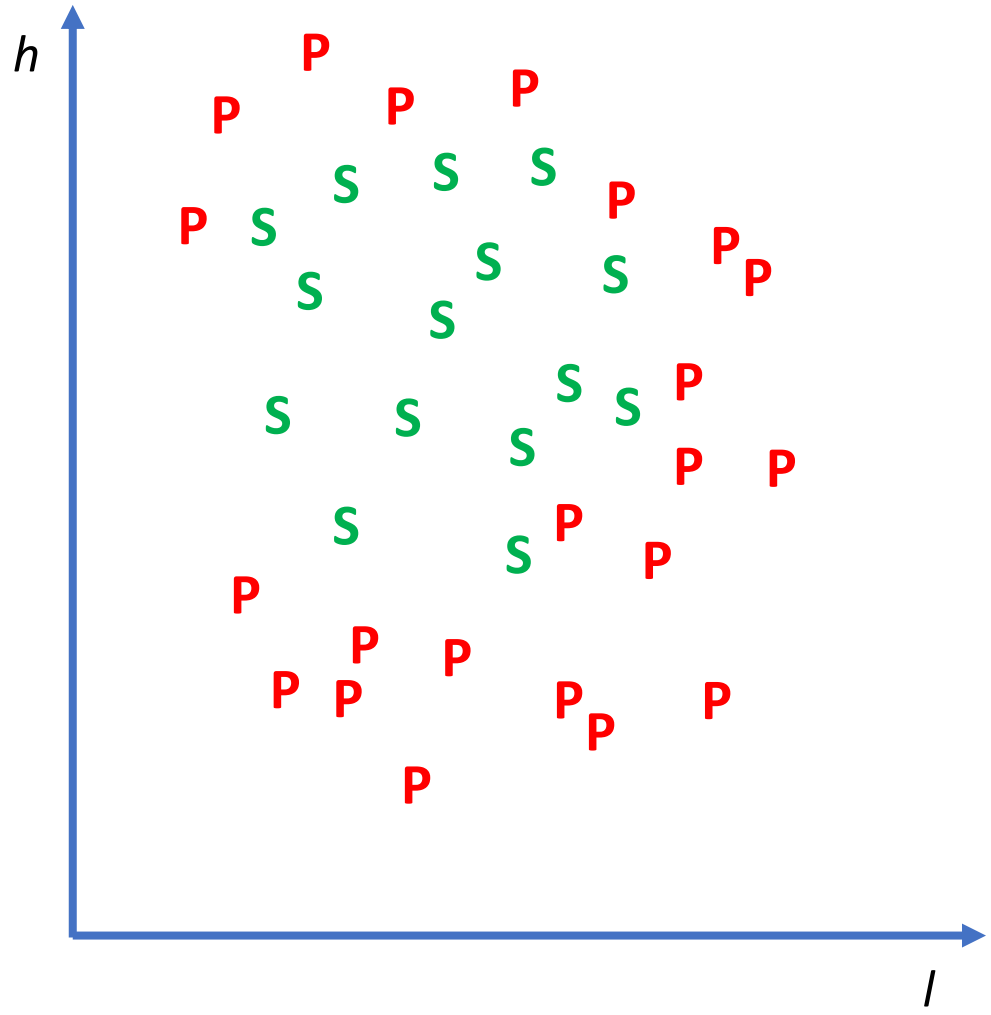
Case Study: Selecting an Approach

- Clearly there are some major technical challenges like communicating with the players in real time, but let's focus other data processing: can we make an accurate prediction?

- Logistic Regression based ML did the trick last time, maybe it can work again…

Brad Quinton, Scott Chin

Case Study: Linear Fit?



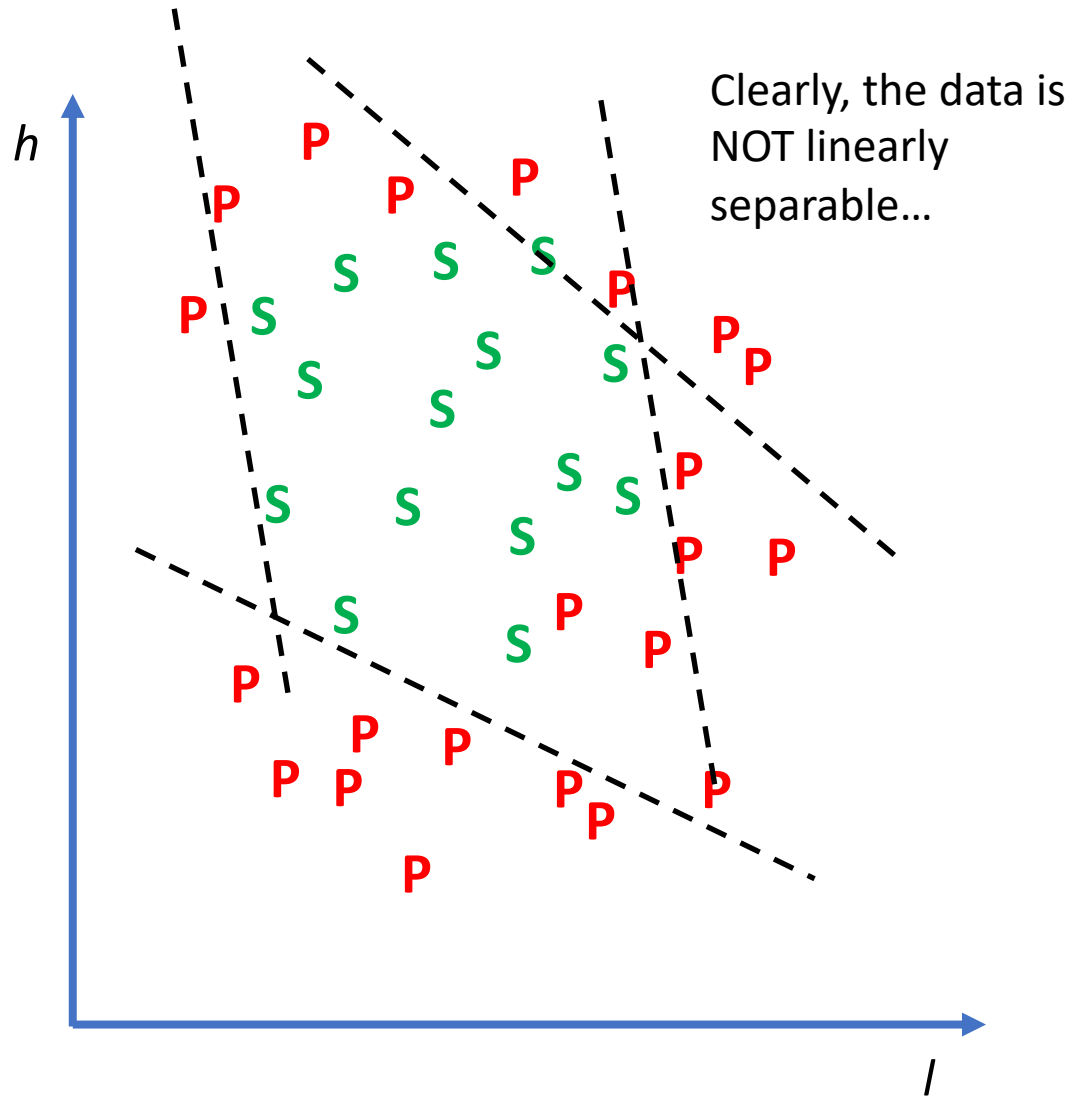
- Remember our logistic regression assumption:

$$a = \sigma(w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b)$$

- In this case we could try:

$$a = \sigma(w_1 l + w_2 h + b)$$

Case Study: Linear Fit?



- Remember our logistic regression assumption:

$$a = \sigma(w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b)$$

- In this case we could try:

$$a = \sigma(w_1 l + w_2 h + b)$$

The Problem With Logistic Regression

- The problem with Logistic Regression is the **assumption** about the linear relationship

- It is possible to make a *new assumption* about the relationship:

$$a = \sigma(w_1 l + w_2 h + \boldsymbol{w_3 l^2} + b)\text{ , or}$$

$$a = \sigma(w_1 l + w_2 h + \boldsymbol{w_3}(\boldsymbol{l} + \boldsymbol{h})^{\boldsymbol{2}} + b)\text{, or even}$$

$$a = \sigma(w_1 l + w_2 h + \boldsymbol{w_3 cos}(\boldsymbol{l} + \boldsymbol{h}) + b)$$

The Problem With Logistic Regression

- The problem with Logistic Regression is the **assumption** about the linear relationship

- It is possible to make a *new assumption* about the relationship:

$$a = \sigma(w_1 l + w_2 h + \boldsymbol{w_3 l^2} + b)\text{ , or}$$

$$a = \sigma(w_1 l + w_2 h + \boldsymbol{w_3}(\boldsymbol{l} + \boldsymbol{h})^{\boldsymbol{2}} + b)\text{, or even}$$

$$a = \sigma(w_1 l + w_2 h + \boldsymbol{w_3 cos}(\boldsymbol{l} + \boldsymbol{h}) + b)$$

We can continue to add various terms to our logistic regression equation… and gradient descent will work.

The Problem With Logistic Regression

- However, how would we know which is the right relationship?

- Imagine a problem with 10, 20, 100 input features! (in our case study, *current score, time on clock*, *opponent*, *current score, …*, *etc.*) they each may have some different non-linear relationship to each other and the final hypothesis!

- If we are going to keep our new job and deliver some results *quickly*, we need away to escape the burden of specifying the specific relationship between each of the the inputs and the hypothesis…

Neural Networks?

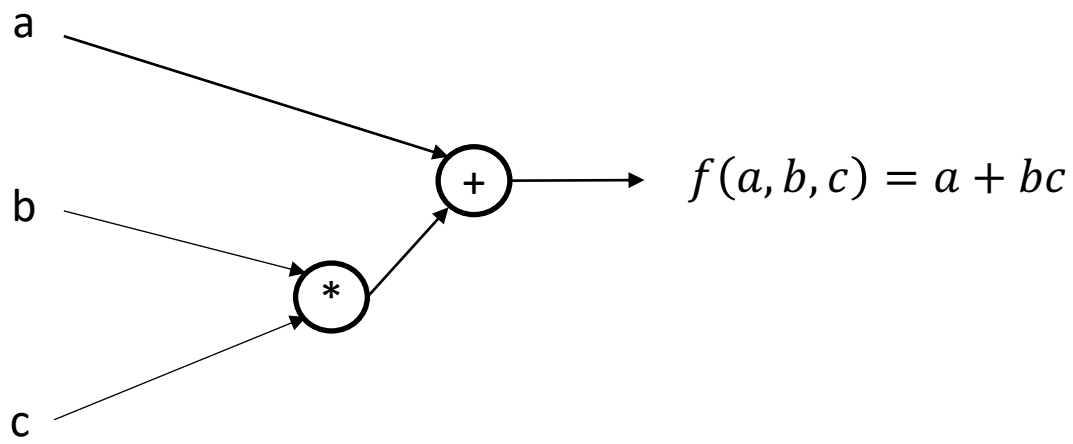
- Luckily for us (and our career prospects as Director of AI!), Neural Networks offer a potential solution

- With Neural Networks we can *learn* very complex non-linear relationships rather than having to specify them

Computation Graph

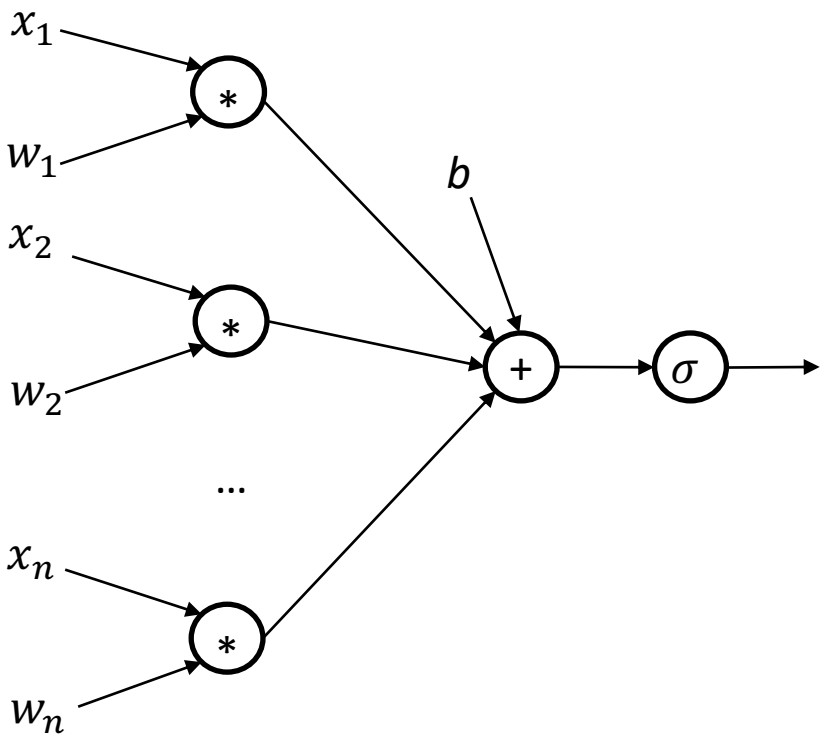
- Although they sound very biological, (Artificial) Neural Network are simply a type of computation graph inspired by an idealized view of a real neuron

- Computation graphs are a (often convenient) way to specify a computation relationship between inputs and outputs



Logistic Regression as a Computation Graph

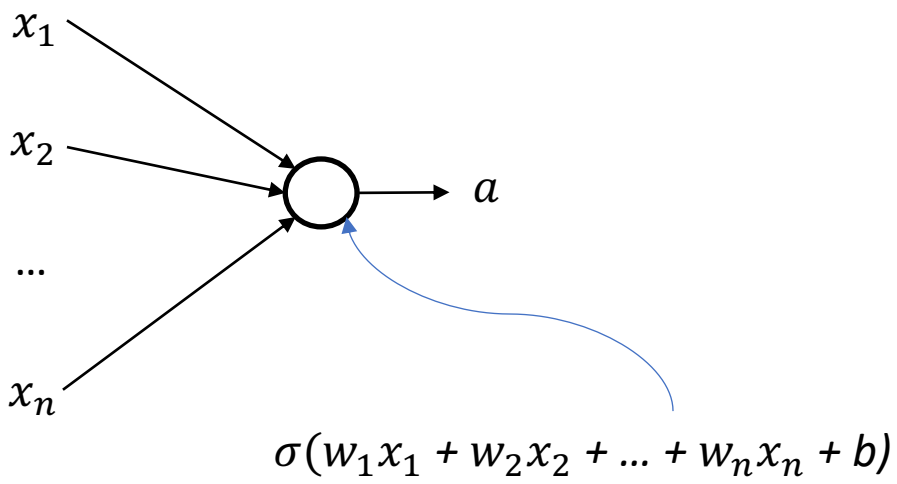
- Let's look at Logistic Regression as a computation graph



$$a = \sigma(w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b)$$

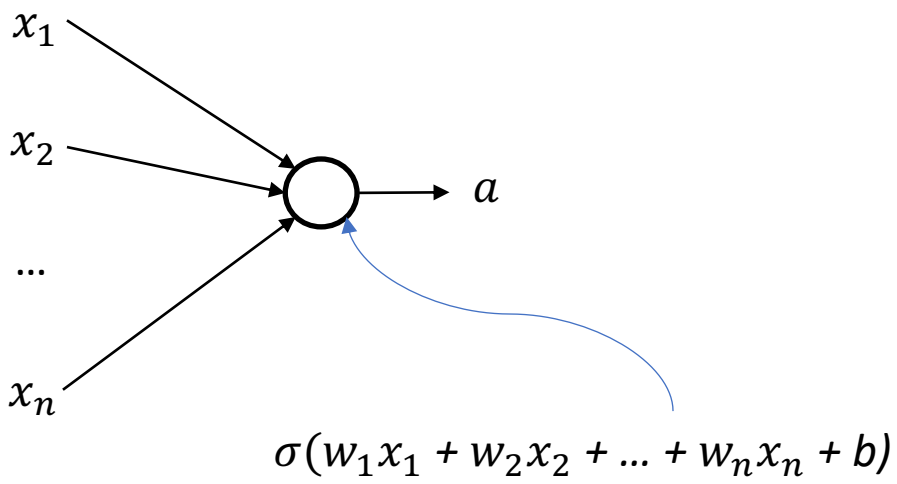
Logistic Regression as a Computation Graph

- w_{1-n}, b are constants associated with the specific node, so as a shorthand we can simply draw the inputs connecting directly to the node:



Logistic Regression as a Computation Graph

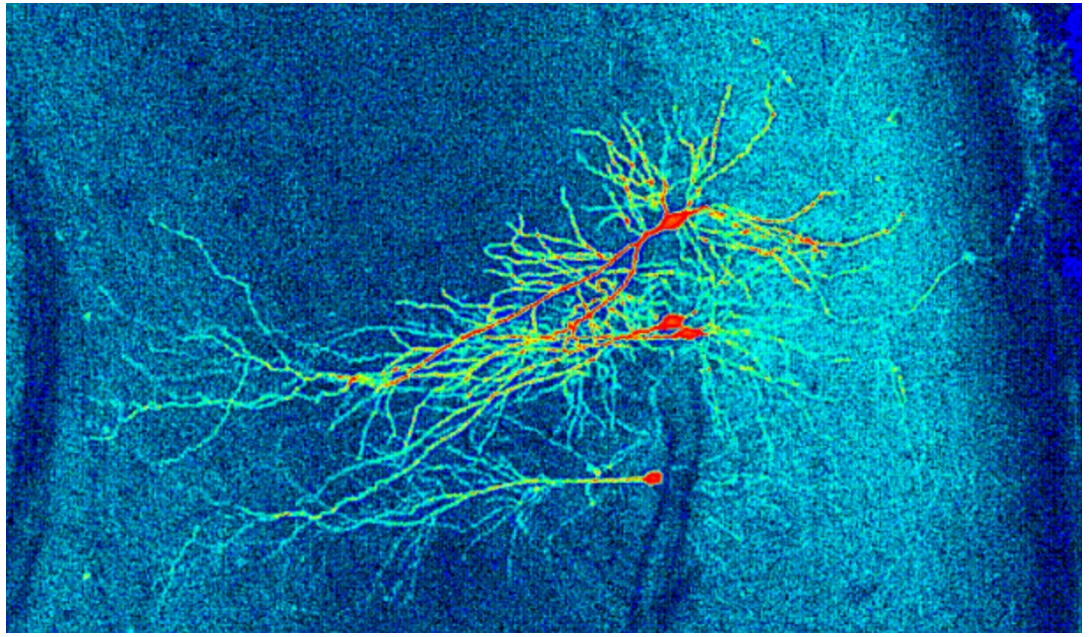
- w_{1-n}, b are constants associated with the specific node, so as a shorthand we can simply draw the inputs connecting directly to the node:



In fact, this is actually a Neural Network, just a very small one!

Putting Logistic Regression Units Together

- If we are going to continue using biological inspiration we need to consider more than one neuron. After all, even a Roundworm has 302 neurons! (People, by the way have > 100 Billion…)

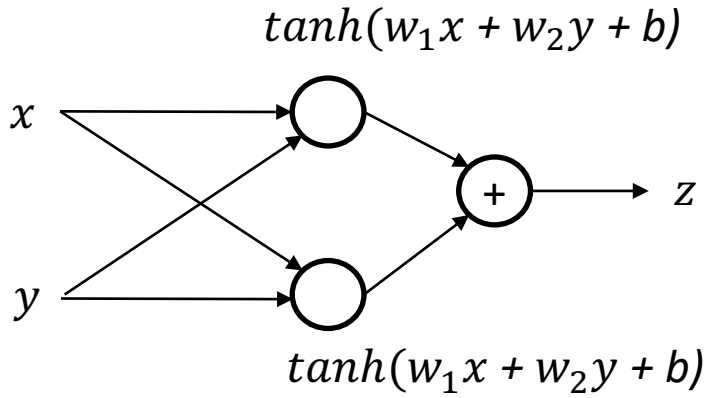


http://www.soton.ac.uk/~jchad/BIG/Neuronal_Morphology/multiple.neurons.html

Brad Quinton, Scott Chin

Putting Logistic Regression Units Together

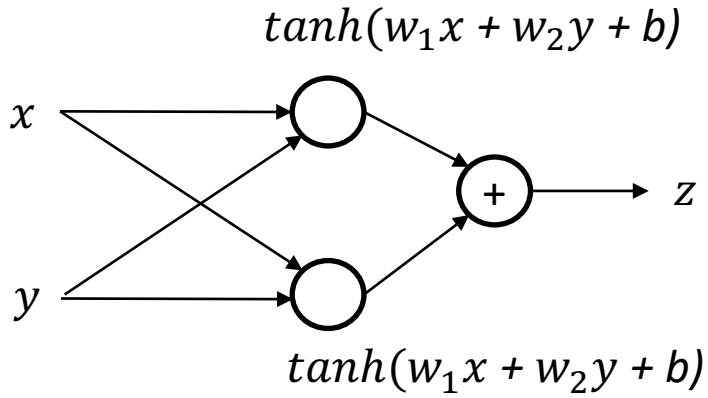
- Let's consider the simplest possible case (two logistic regression units together):



  - What have new abilities have we gained?

Putting Logistic Regression Units Together

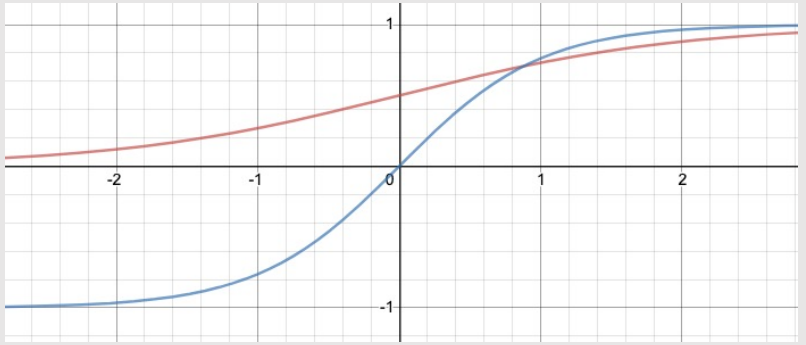
- Let's consider the simplest possible case (two logistic regression units together):



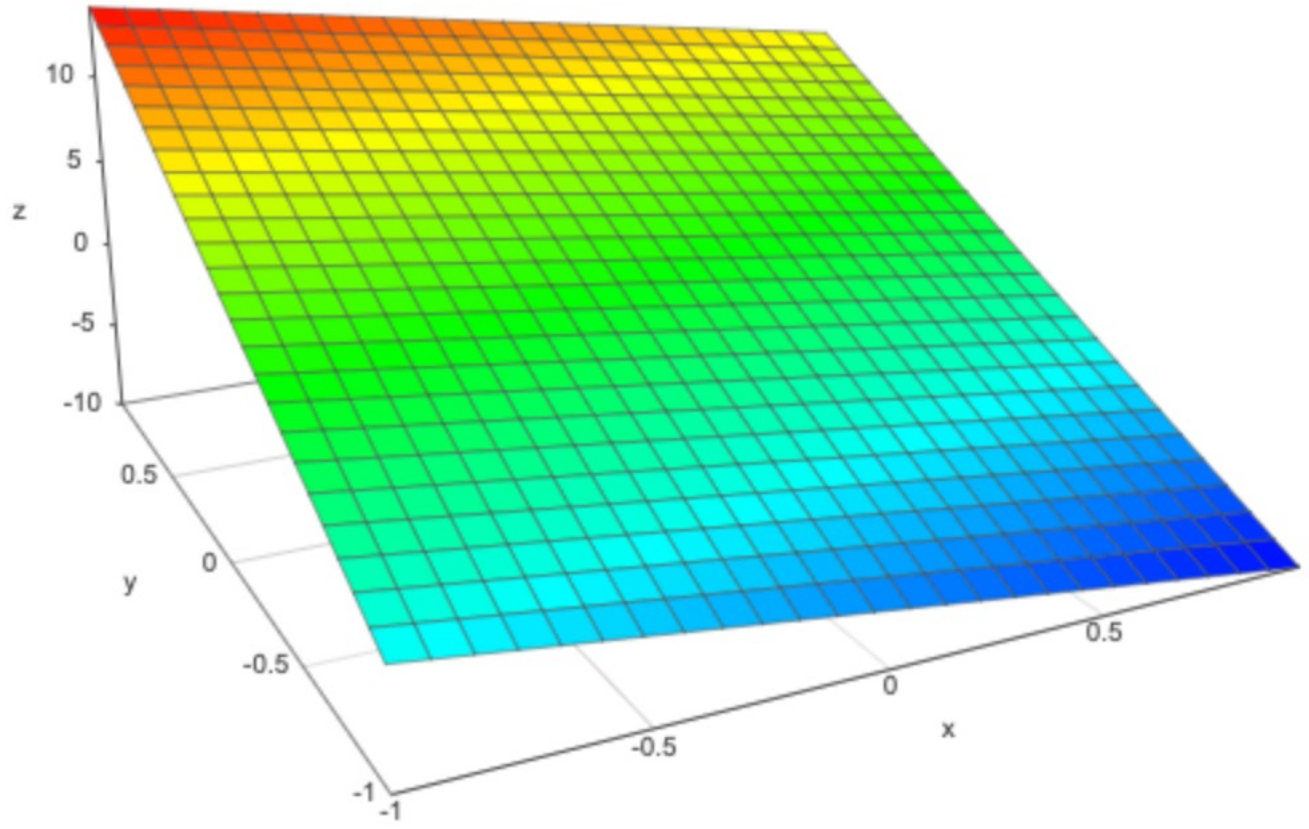
- What have new abilities have we gained?

Side note: tanh vs. sigmoid**:**

For NNs we tend to use tanh in the middle layers. Both functions are similar. For now, we can consider them to serve the same purpose.



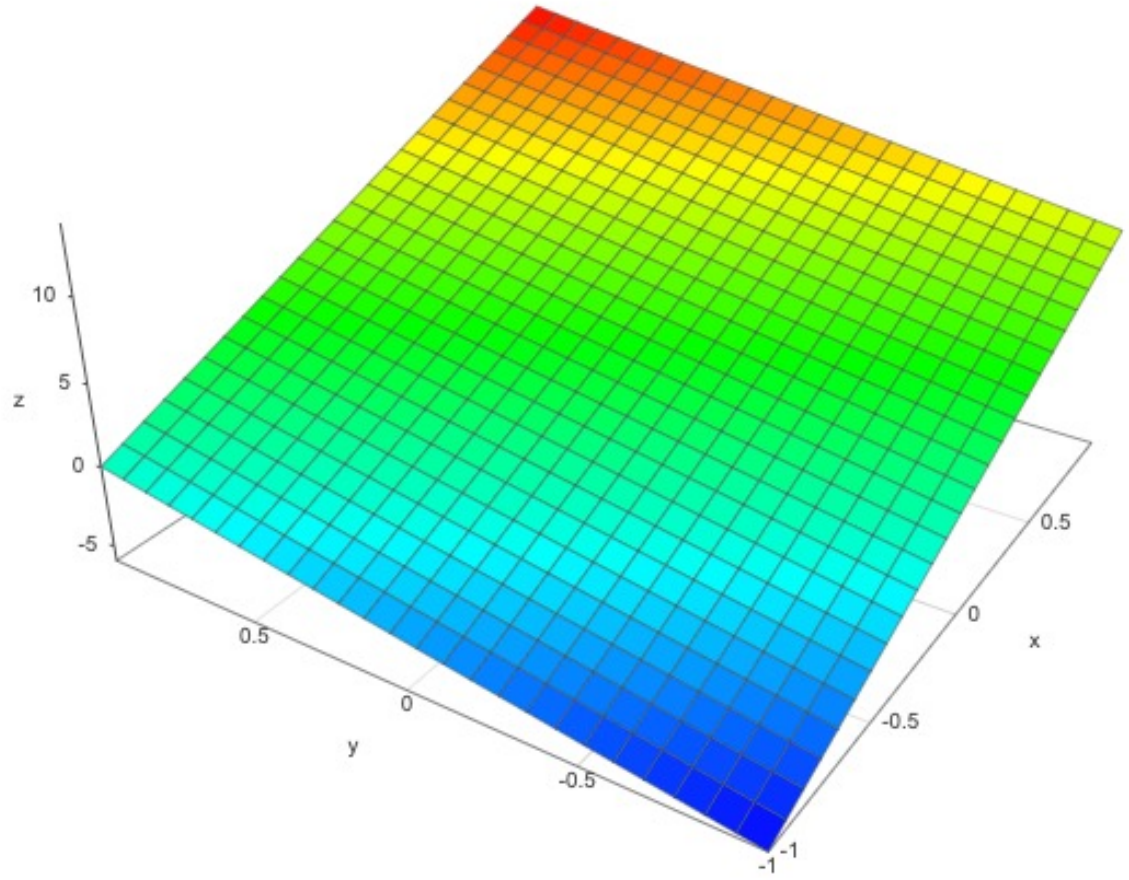
Weighted Sums



- Consider:

$$z = 9x - 3y + 2$$

Weighted Sums



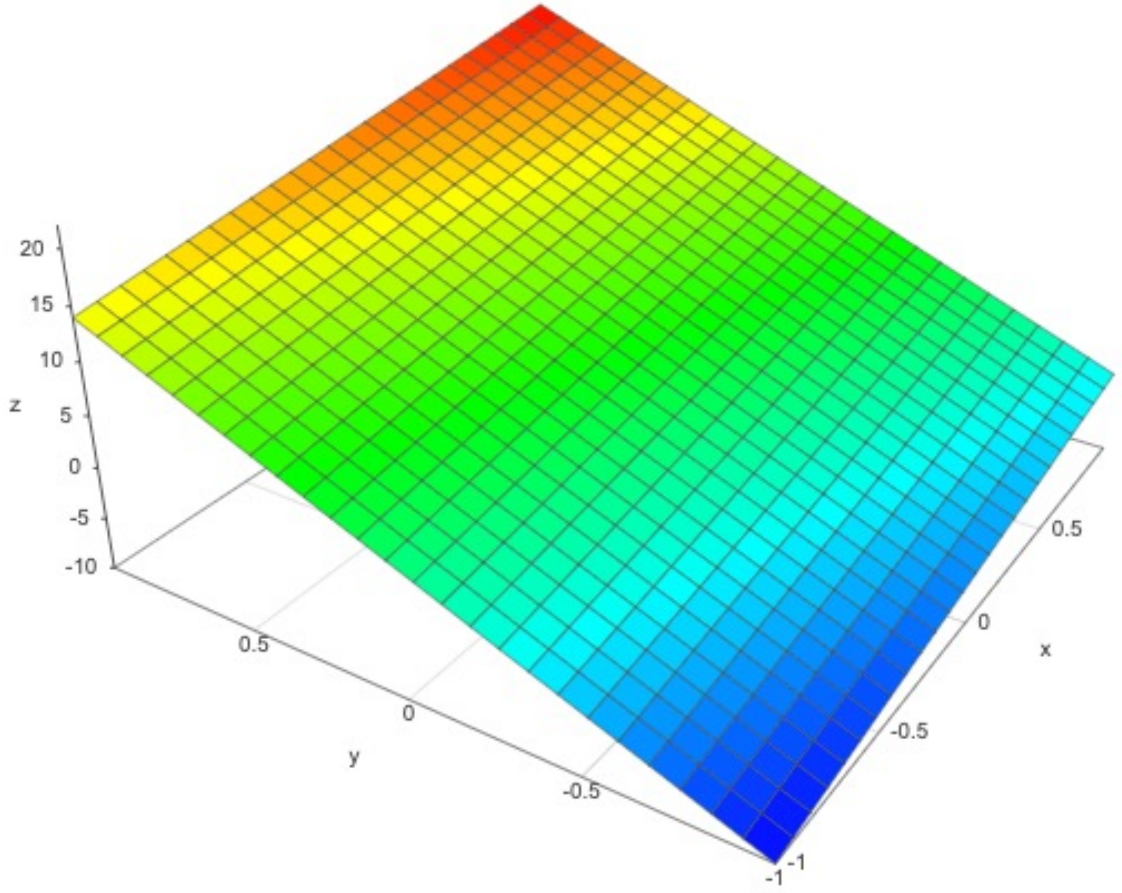
- Consider:

$$z = 9x - 3y + 2$$

- Or:

$$z = 3x + 7y + 4$$

Weighted Sums



- Consider:

$$z = 9x - 3y + 2$$

- Or:

$$z = 3x + 7y + 4$$

- Or, even:

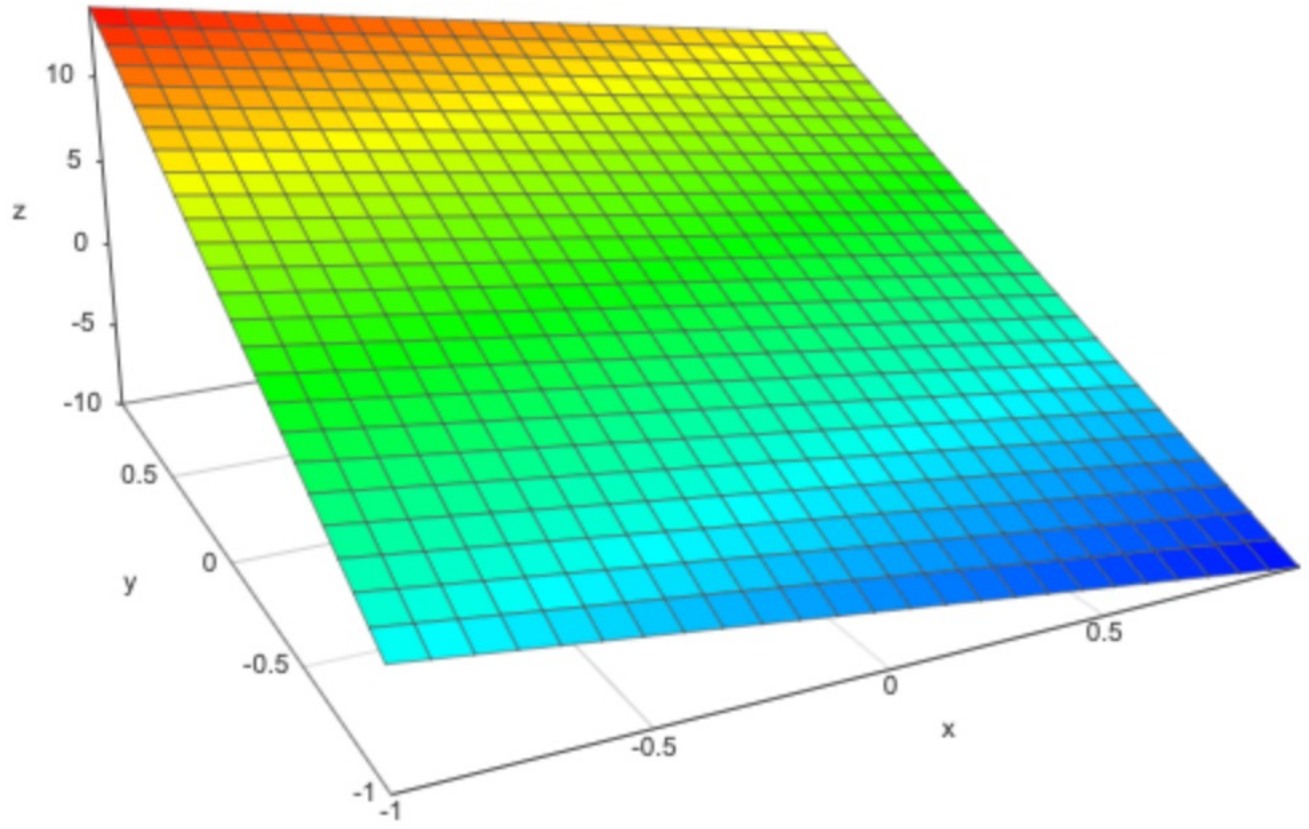
$$z = (9x - 3y + 2) + (3x + 7y + 4)$$

Importance of the Activation Function

- At first glance it seems that connecting multiple Linear Regression units does NOT add much/any new flexibility

- However, we are missing something. Remember the *activation* function we added to our Logistic Regression

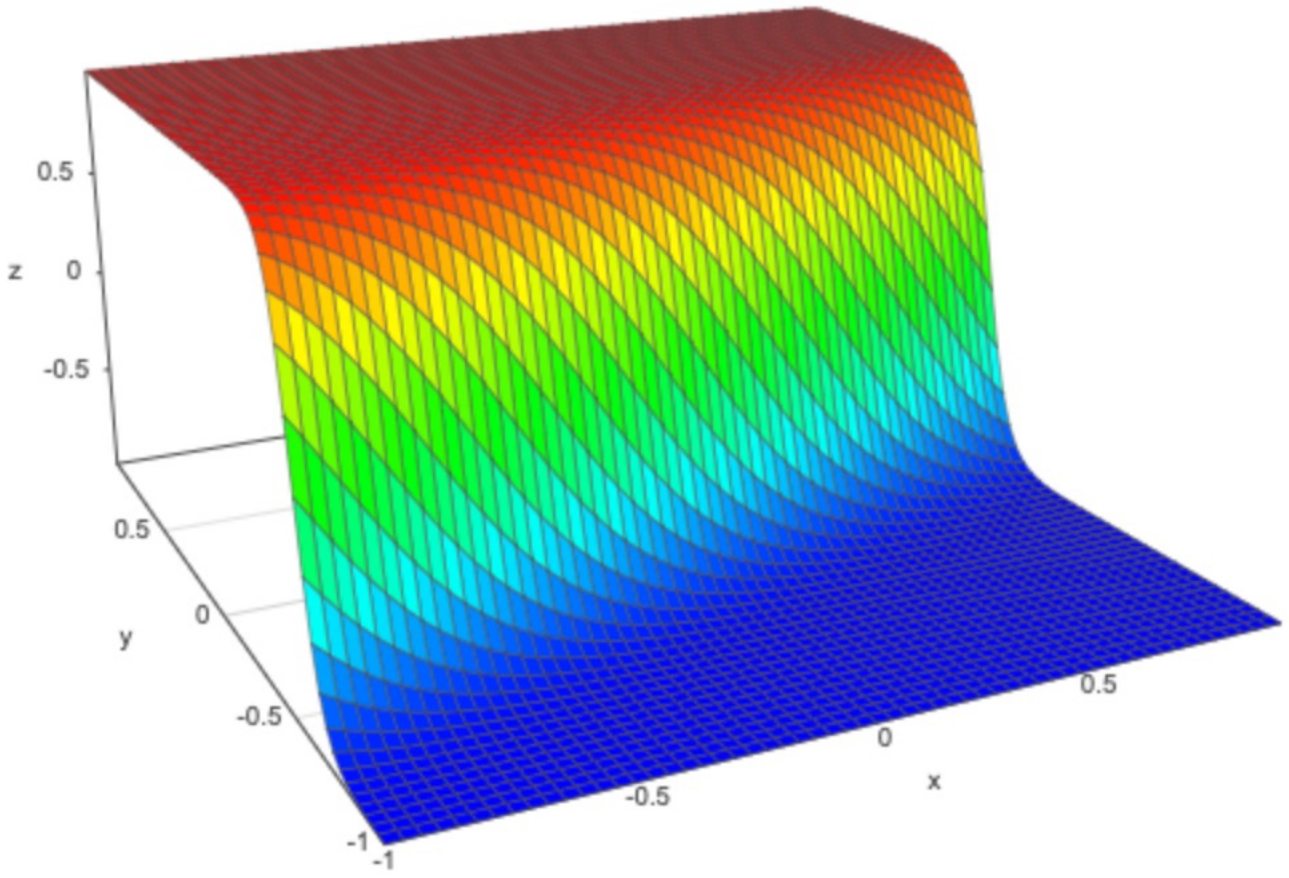
Activation Function



- Consider:

$$z = 9x - 3y + 2$$

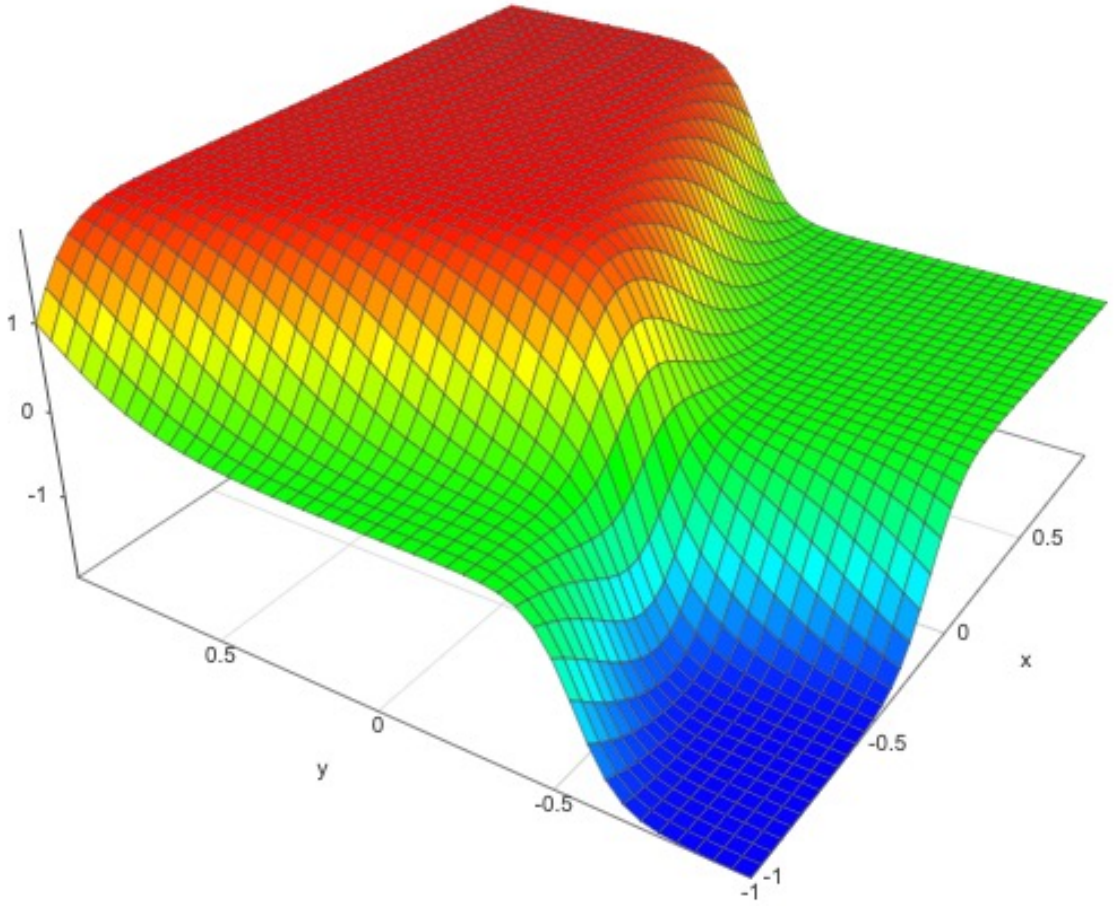
Activation Function



- Consider:

$$z = \tanh(9x - 3y + 2)$$

Activation Function



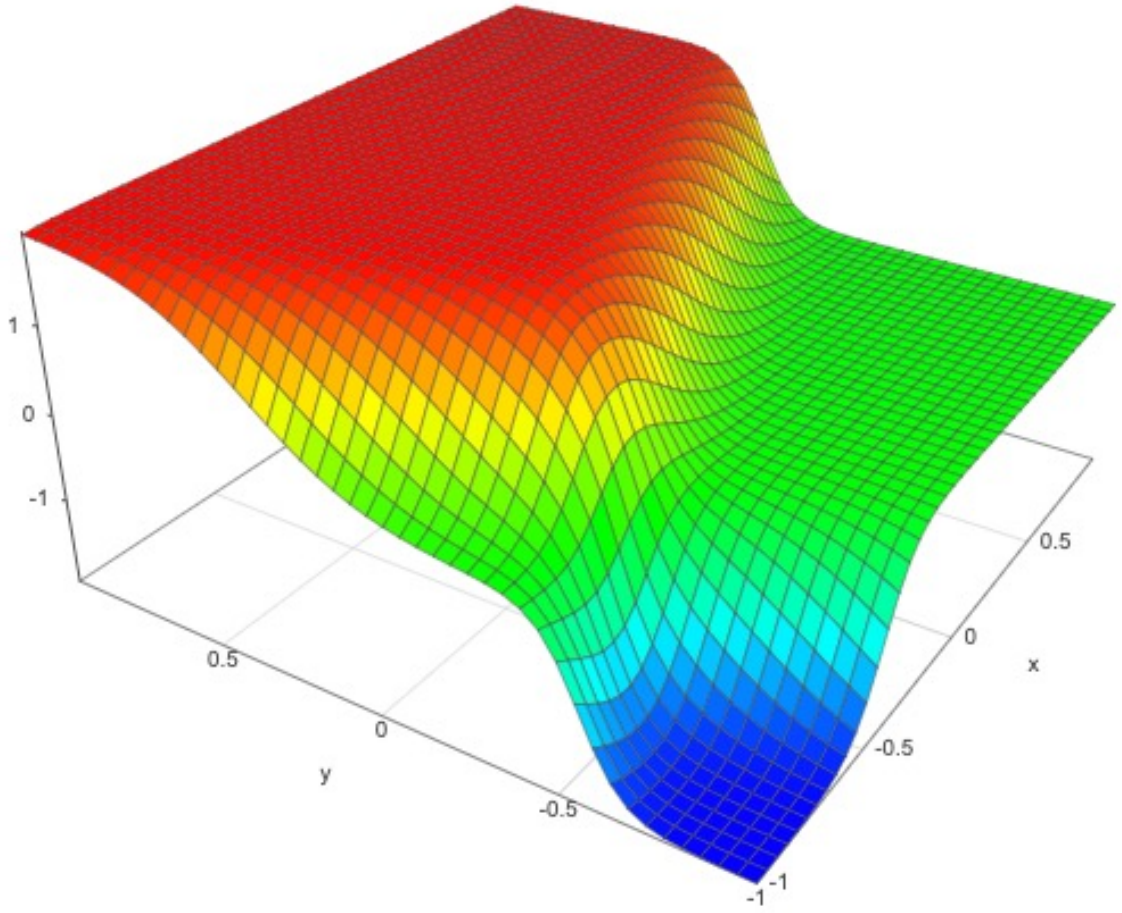
- Consider:

$$z = \tanh(9x - 3y + 2)$$

- Now consider:

$$z = \tanh(9x - 3y + 2) + \tanh(3x + 7y + \mathbf{4})$$

Activation Function



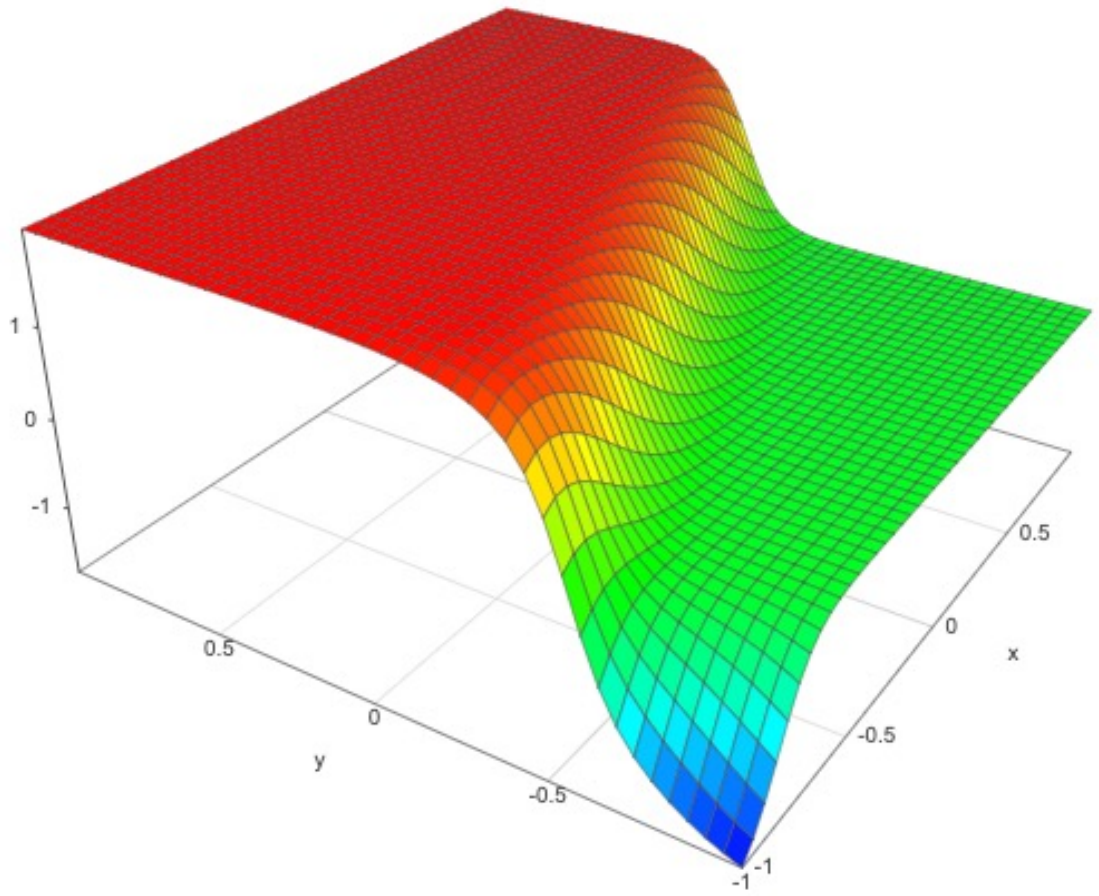
- Consider:

$$z = \tanh(9x - 3y + 2)$$

- Now consider:

$$z = \tanh(9x - 3y + 2) + \tanh(3x + 7y + \mathbf{6})$$

Activation Function



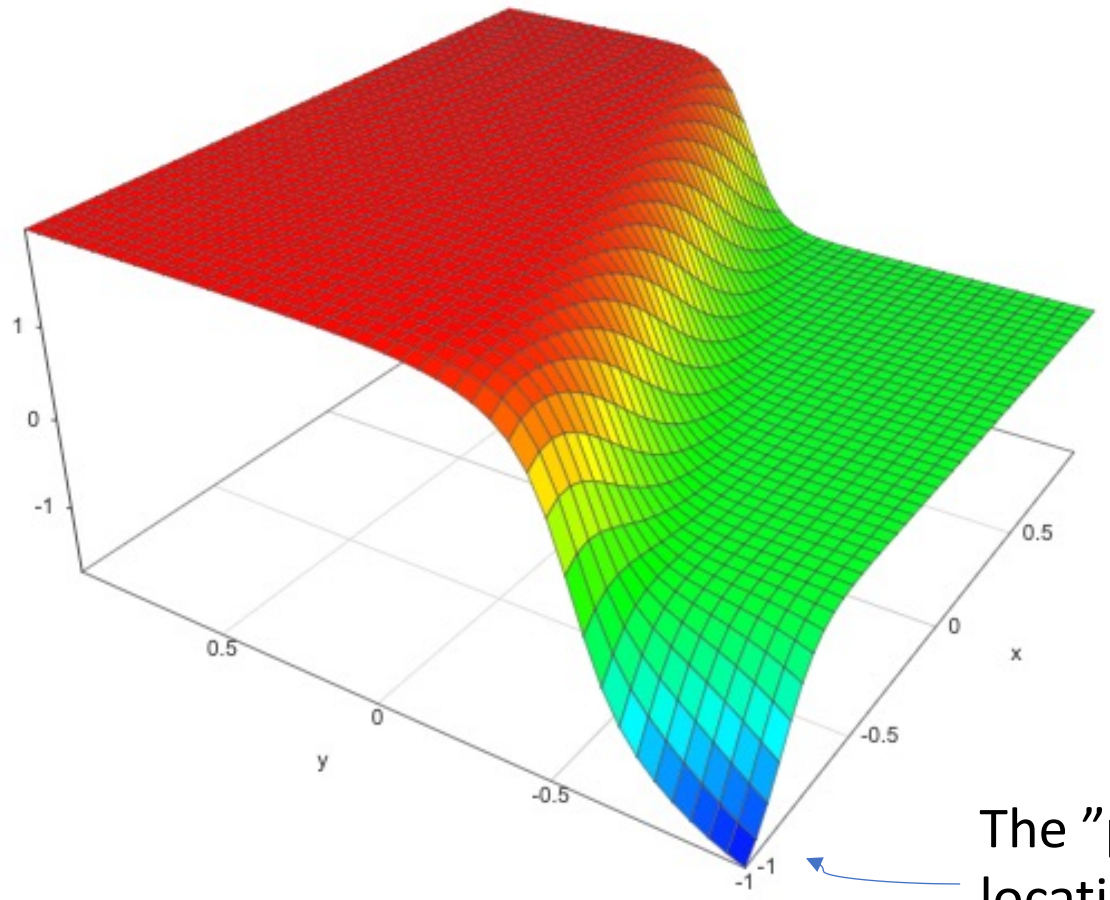
- Consider:

$$z = \tanh(9x - 3y + 2)$$

- Now consider:

$$z = \tanh(9x - 3y + 2) + \tanh(3x + 7y + \mathbf{9})$$

Activation Function



- Consider:

$$z = \tanh(9x - 3y + 2)$$

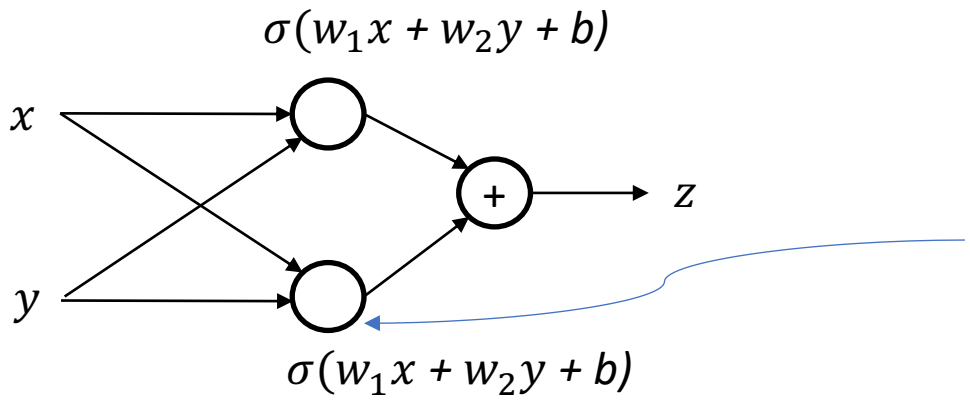
- Now consider:

$$z = \tanh(9x - 3y + 2) + \tanh(3x + 7y + \mathbf{9})$$

The ”parameter” directly effects the location of the decision boundary.

Importance of the Activation Function

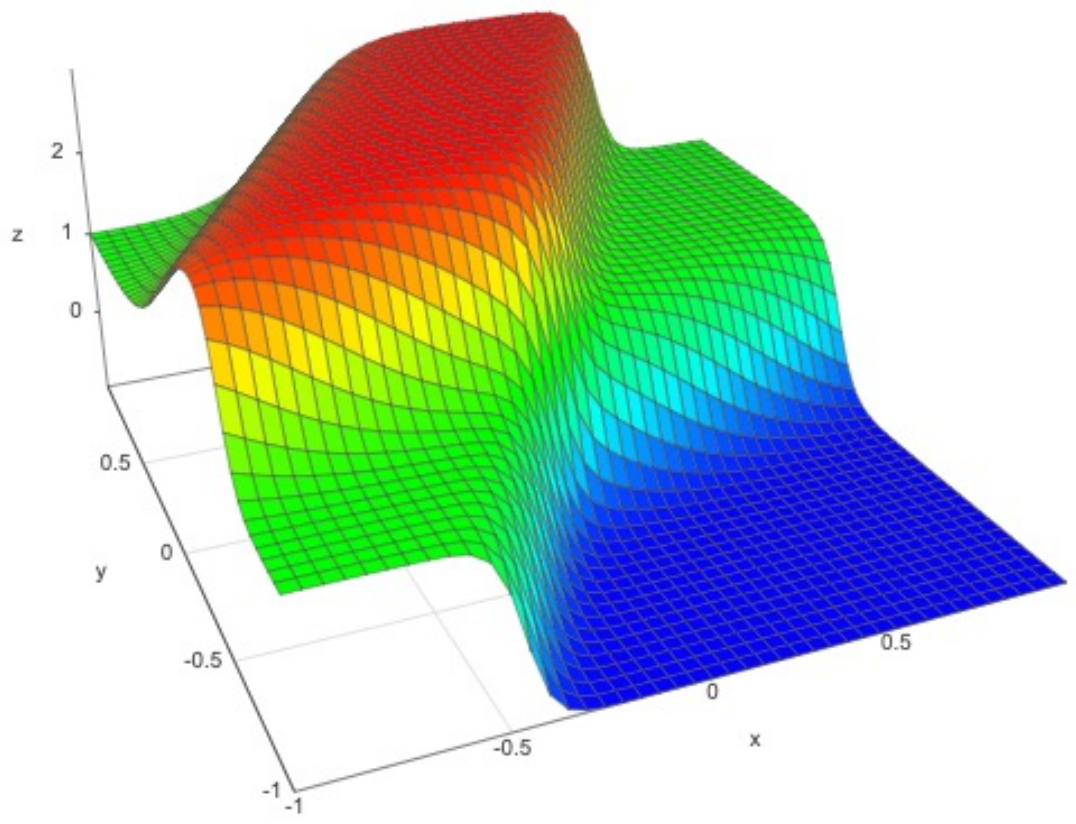
- The non-linear activation function is *the key* to allowing combinations of logistic regression units to produce complex functions:



Each "neuron" (Logistic Regression Unit) produces a non-linear output, even though it is creating a linear decision boundary.

- Without a non-linear activation function, all combinations of logistic regression would continue to be **linear**!

Complex Decision Boundaries

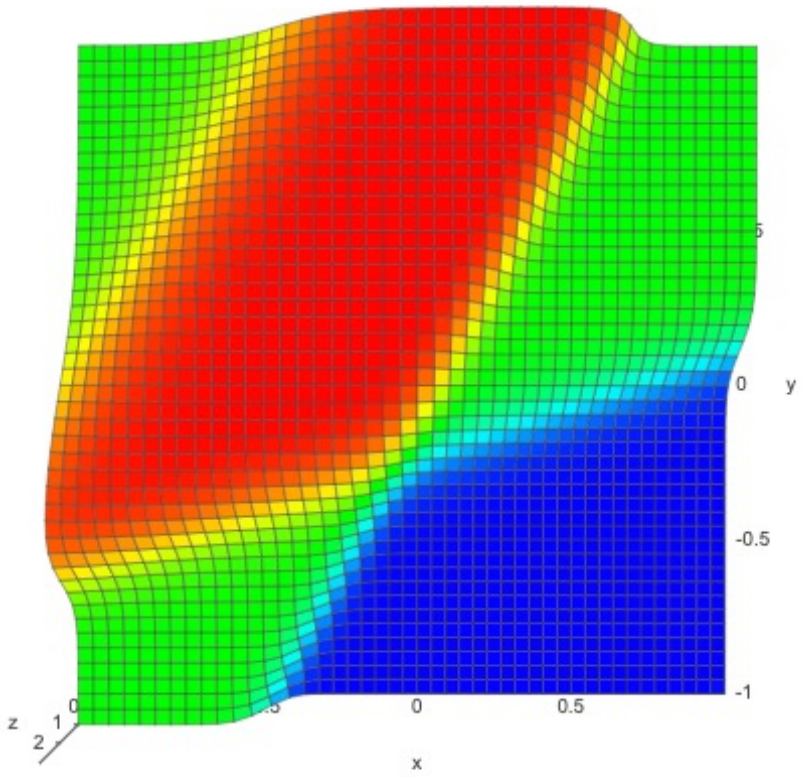


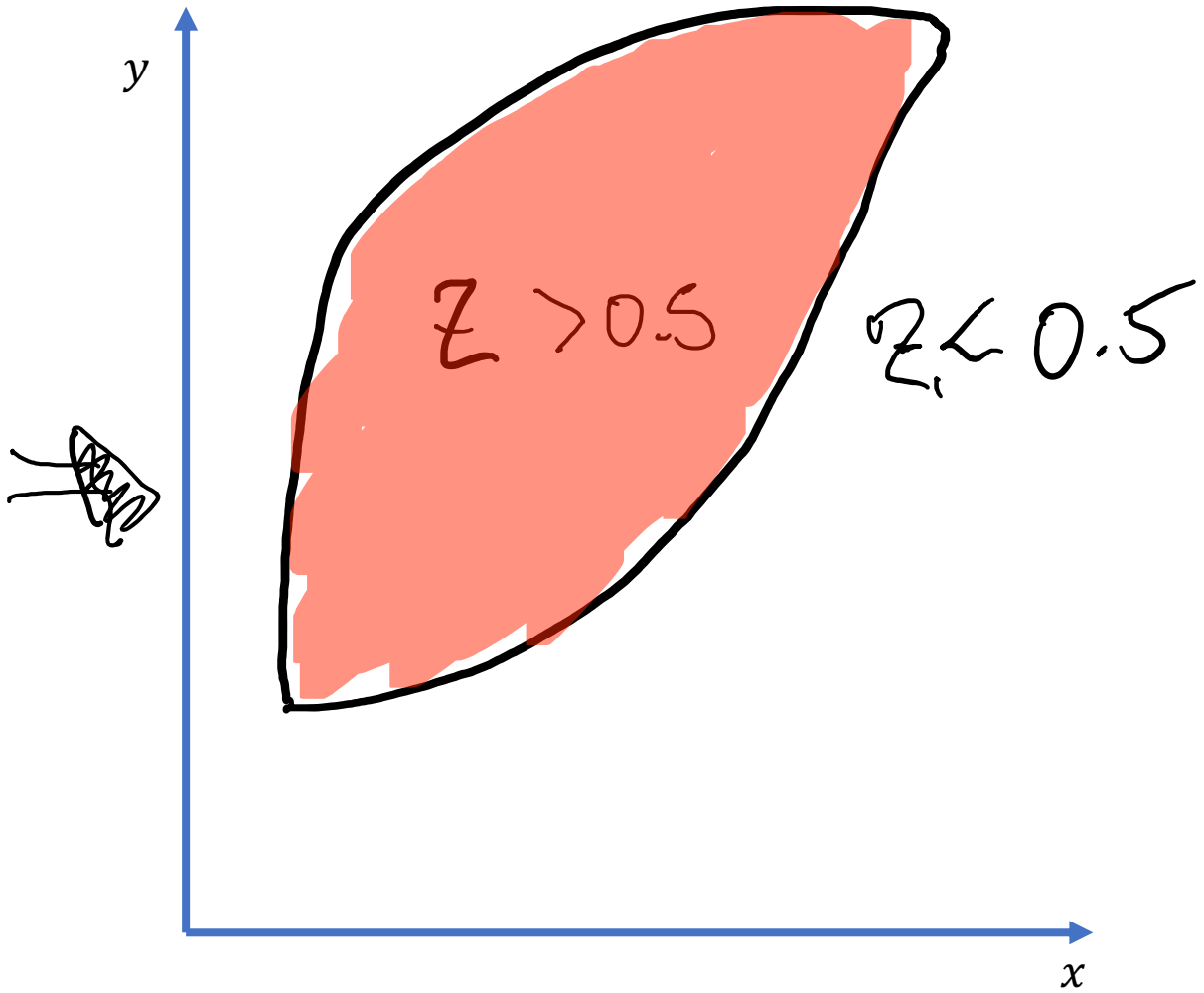
- With only three Logistic Regression Units, it is already possible to “group” points:

$$
\begin{aligned}
&\mathrm{Z} \\
&= \tanh(-3y + 5x + 5) \\
&+ \tanh(9y - 3x + 2) \\
&+ \tanh(8y - 15x + 1)
\end{aligned}
$$

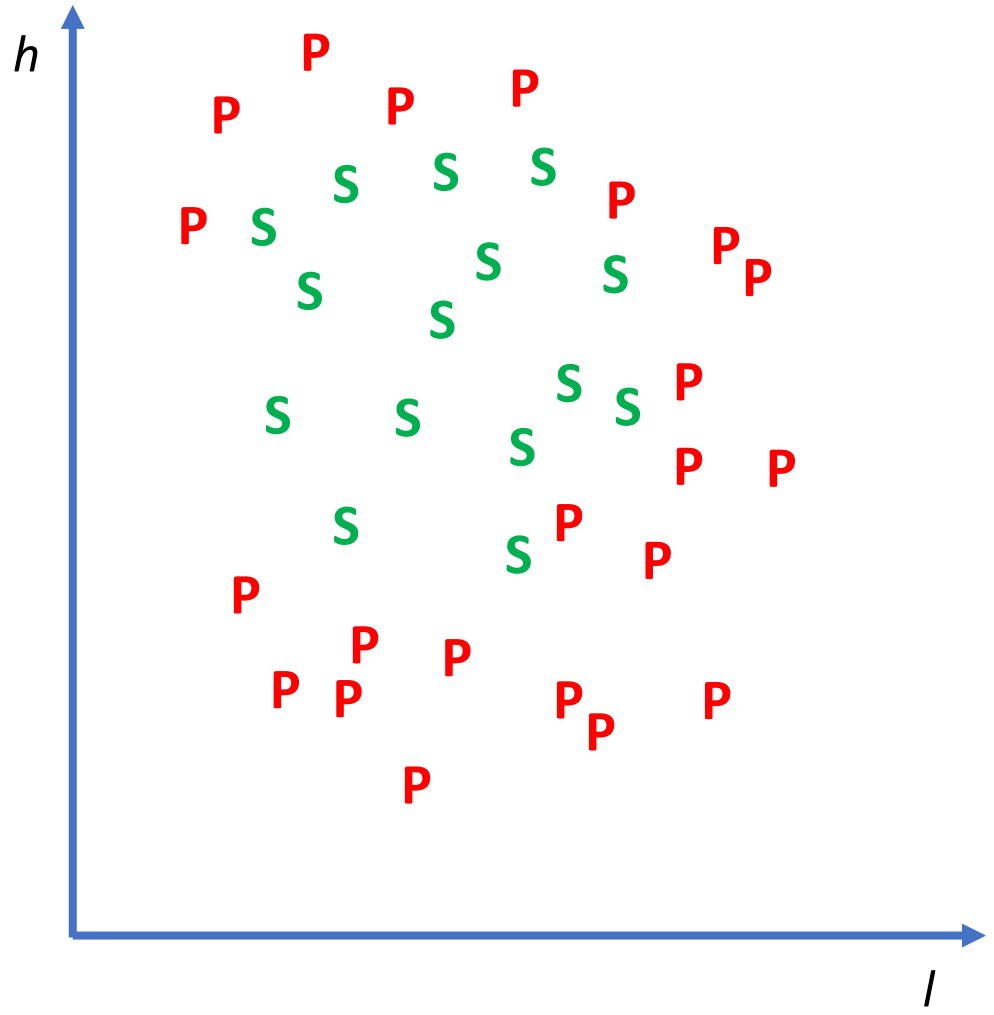
Brad Quinton, Scott Chin

Complex Decision Boundaries



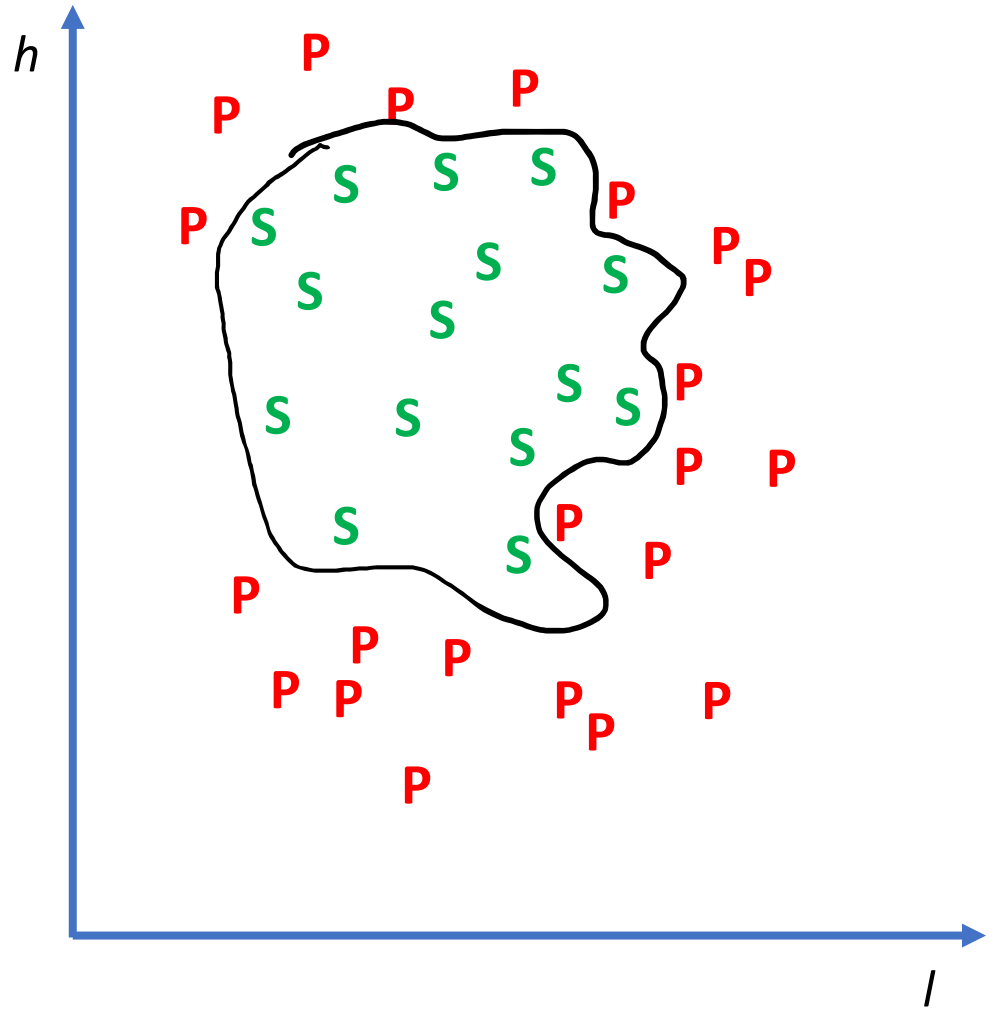


Thinking About the Case Study…



- It looks like Neural Networks will have much better chance fitting the data than logistic regression…

Thinking About the Case Study…



- It looks like Neural Networks will have much better chance fitting the data than Logistic Regression…

Neural Networks

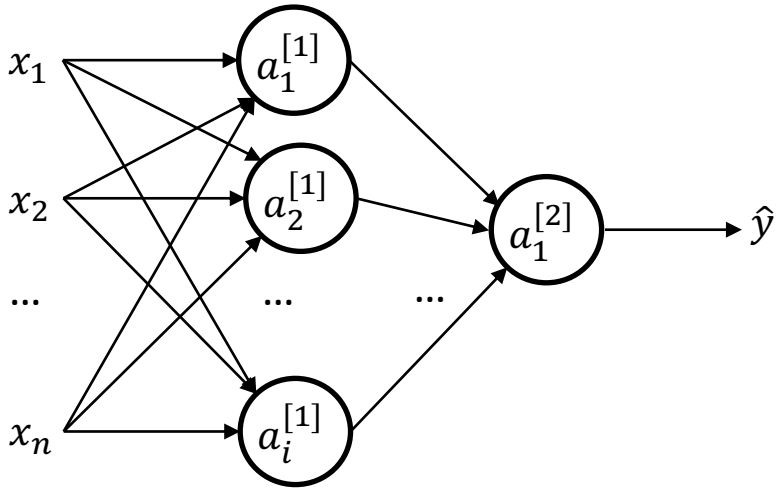
- The power of Neural Networks is that **connecting** even a small number of units with simple behaviours enables the *approximation* of very **complex functions** *(this is an 'emergent' behaviour)*

- And, as we will see later in Deep Learning: connecting a *large* number of simple units enables *extremely complex* functions

- However, the truly remarkable thing about NNs, is that in spite of the potential for complexity, they can be trained in a straight-forward and efficient manner, lets see how….

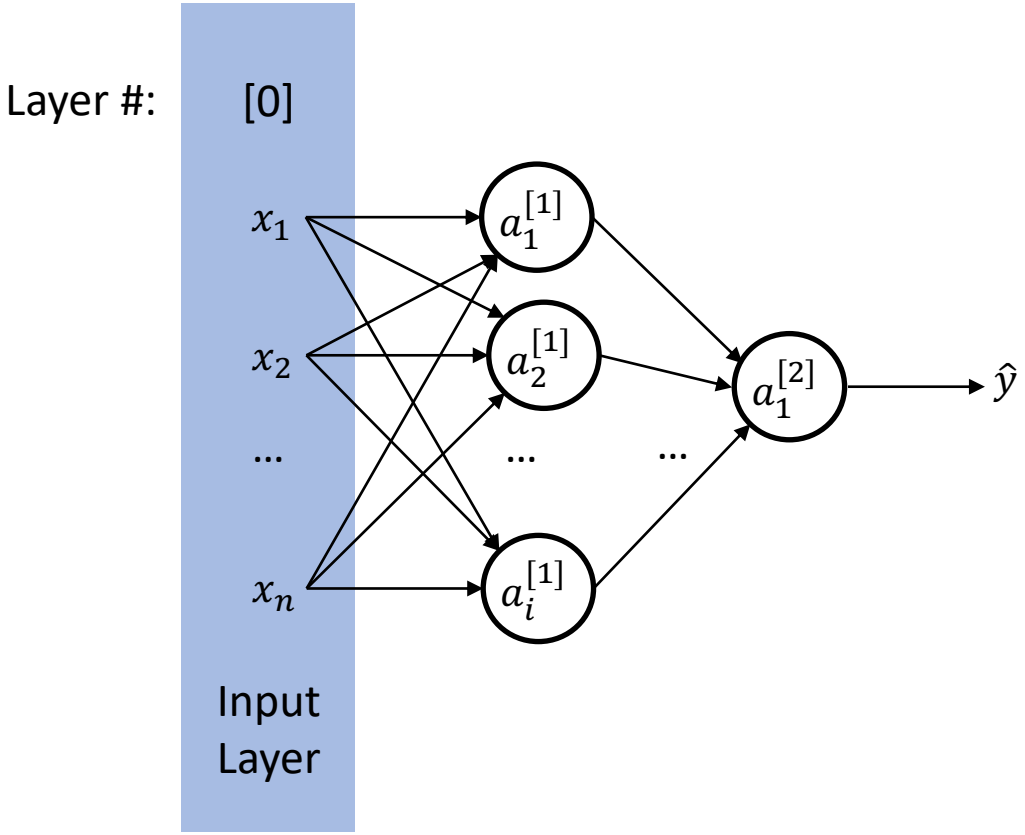
Formalizing Neural Network Construction

- Let’s consider a 2-layer Network:



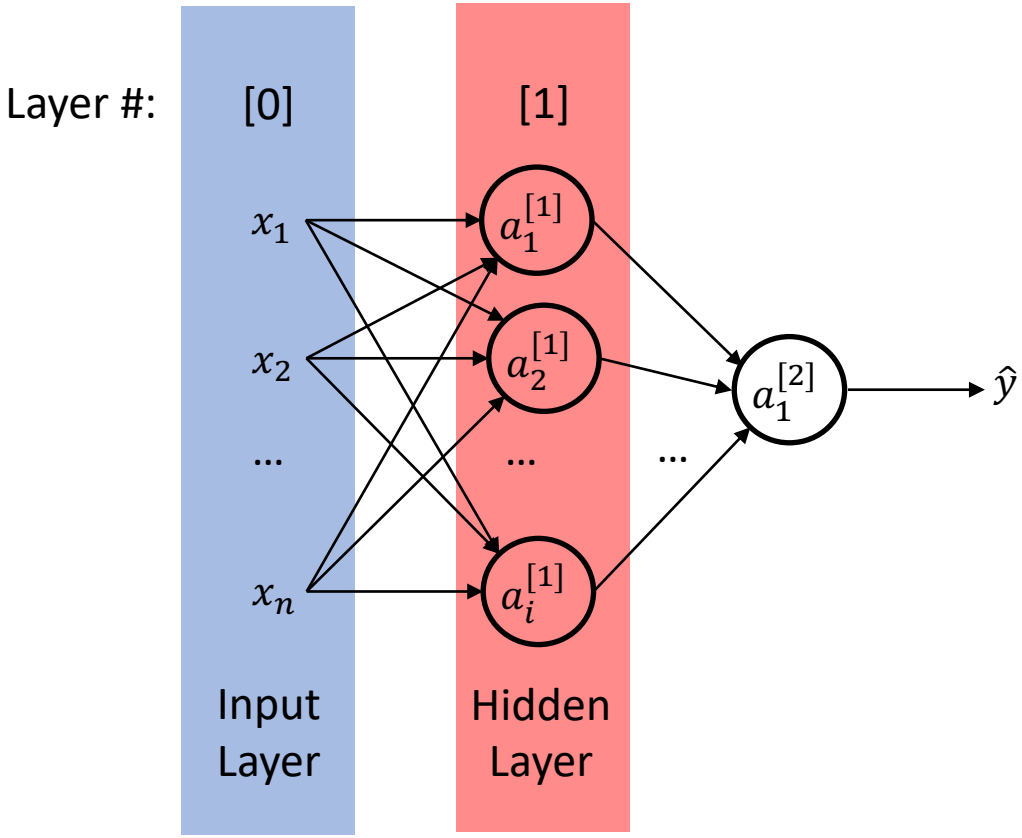
Formalizing Neural Network Construction

- Let's consider a 2-layer Network:



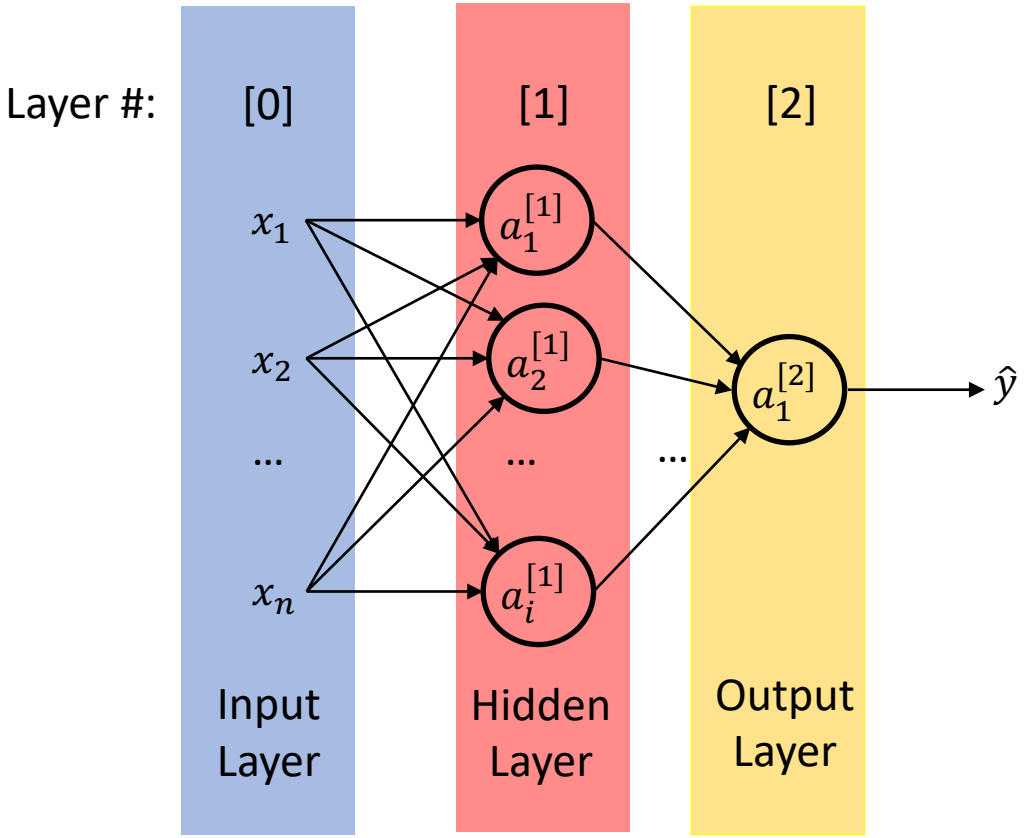
Formalizing Neural Network Construction

- Let's consider a 2-layer Network:



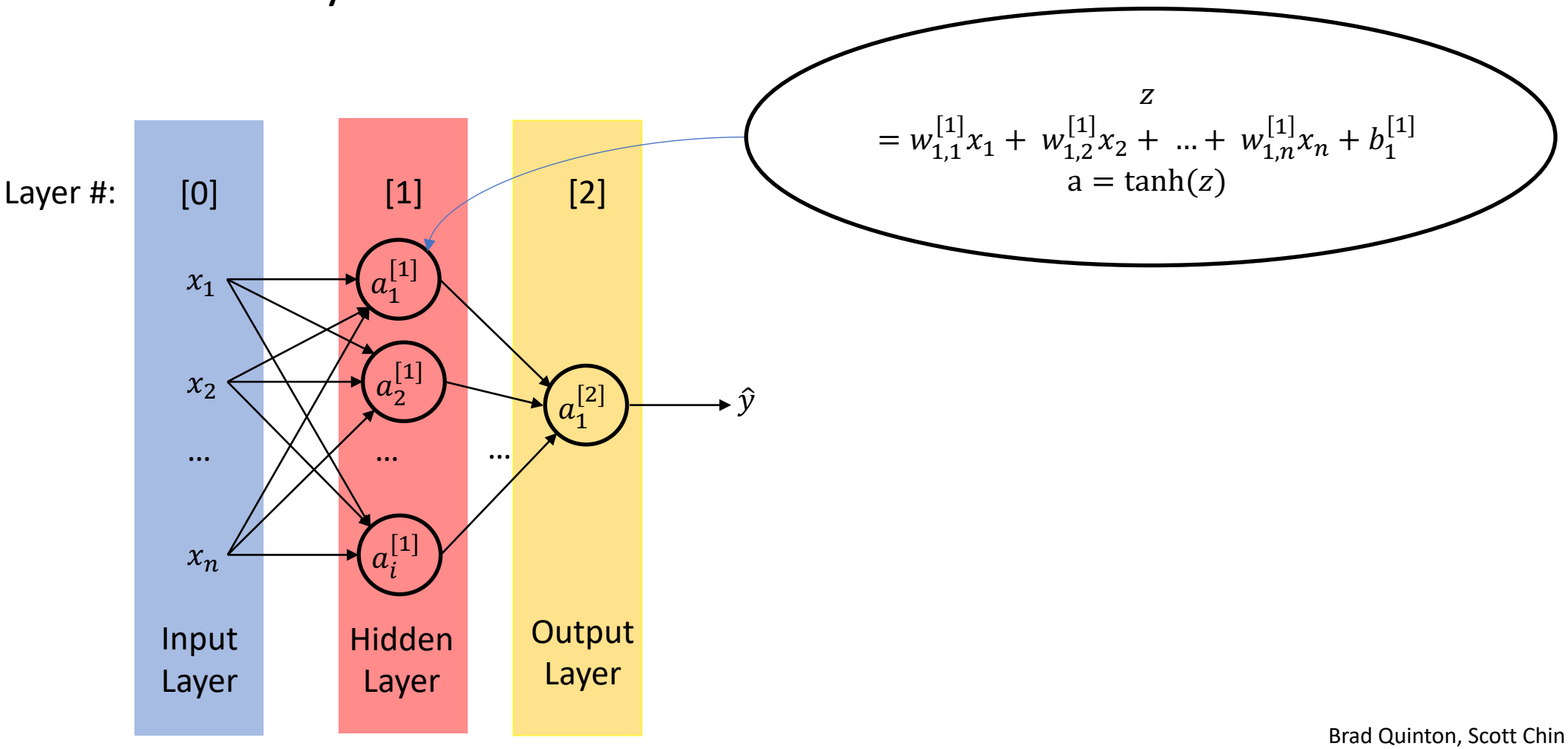
Formalizing Neural Network Construction

- Let's consider a 2-layer Network:



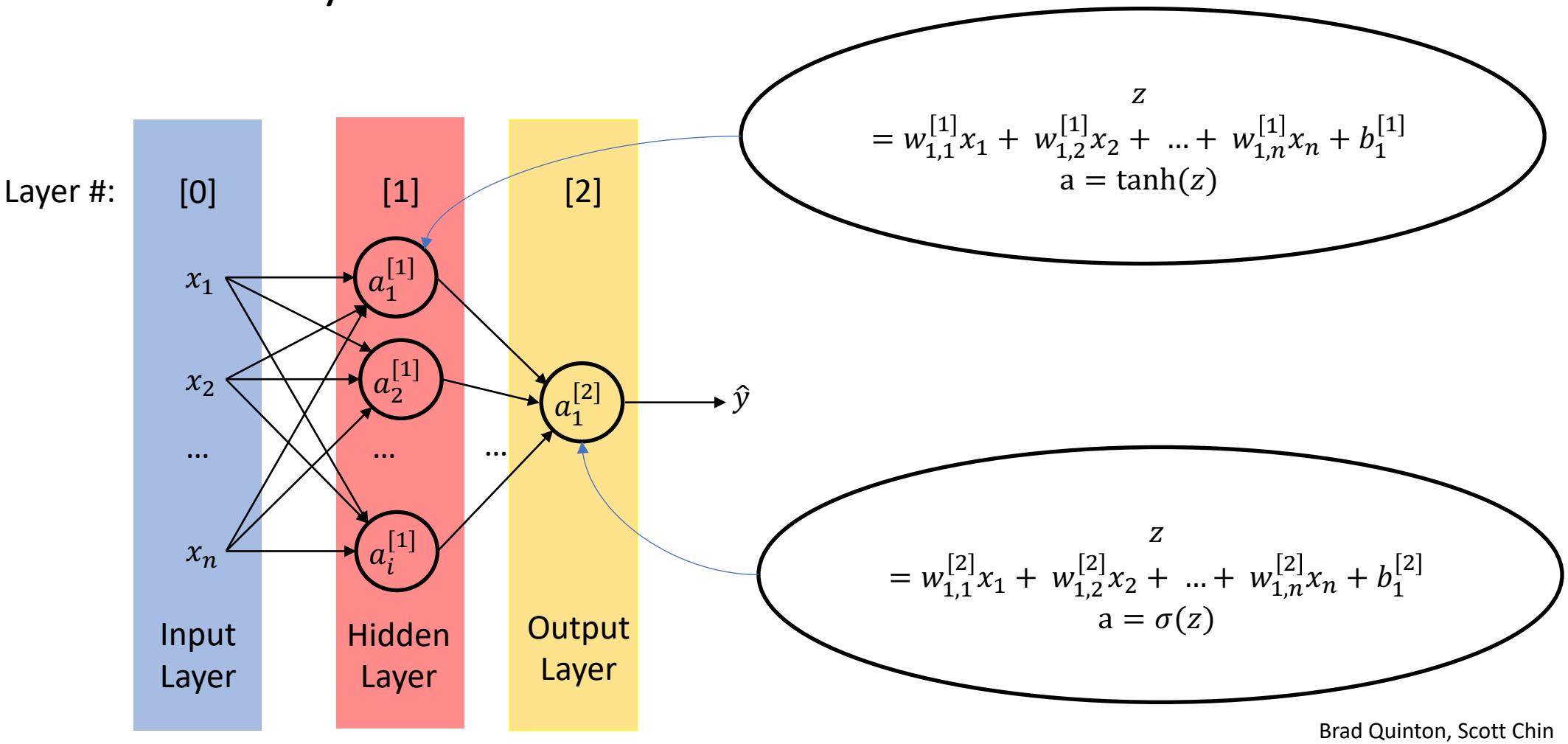
Formalizing Neural Network Construction

- Let's consider a 2-layer Network:



Formalizing Neural Network Construction

- Let's consider a 2-layer Network:



How do we Find the Parameters?

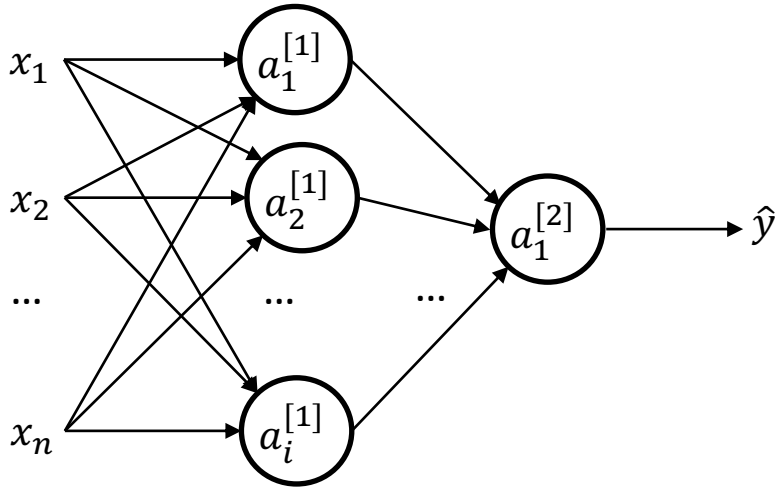
- Luckily, we can leverage our success using gradient descent for Logistic Regression to learn the parameter of our Neural Network

- Consider the output, $\hat{y}$, for binary classification, the same cost function we developed for Logistic Regression continues to be appropriate:

$$J(\hat{y}, y) = -\frac{1}{m}\left(\sum_{i=1}^{m} y^{i}\log\left(\hat{y}^{(i)}\right) + \sum_{i=1}^{m}\left(1 - y^{(i)}\right)\log(1 - \hat{y}^{(i)})\right)$$

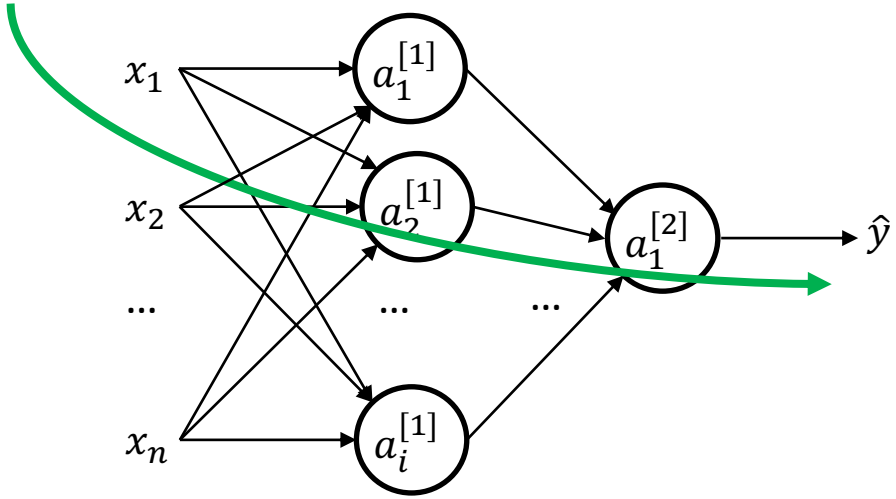
Because there are now many activations, a, in the NN, we traditionally use $\hat{y}$ to denote the activation of the output layer of the NN.

Backpropagation: Go Forwards, then Backwards…



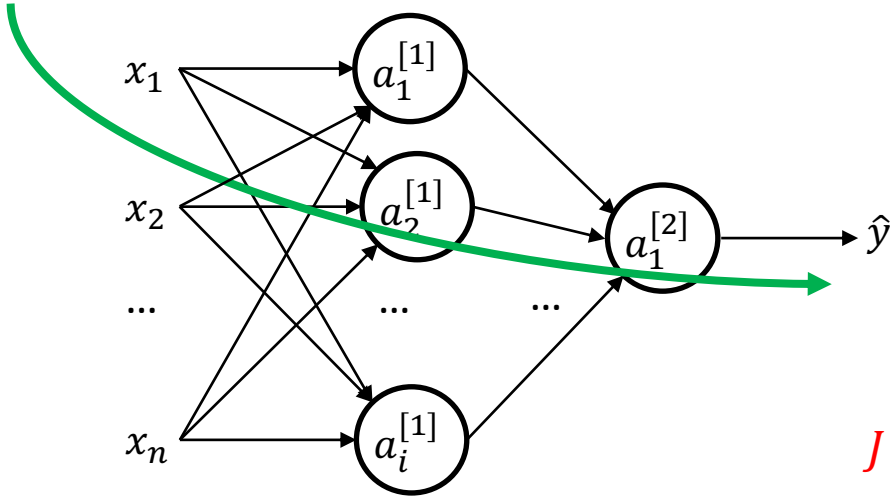
Backpropagation: Go Forwards, then Backwards…

Step 1: Calculate $\hat{y}$ using computation graph.



Backpropagation: Go Forwards, then Backwards…

Step 1: Calculate $\hat{y}$ using computation graph.

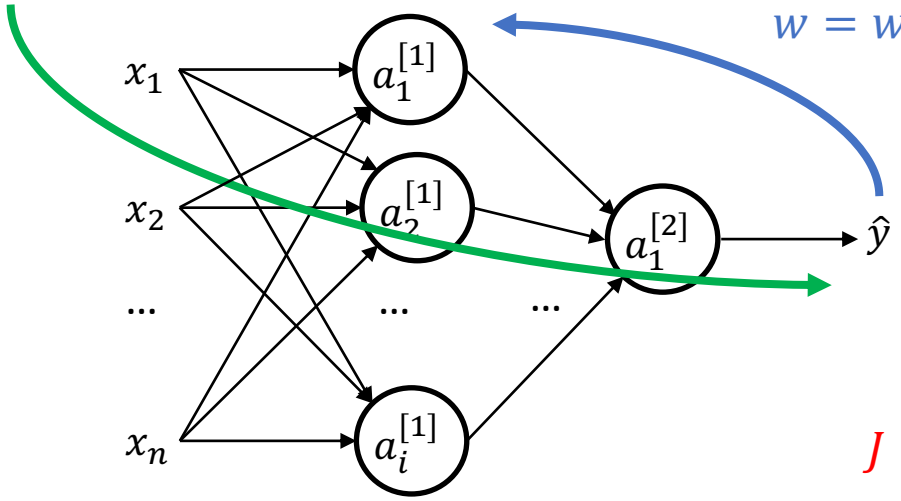


Step 2: Determine the loss.

$$J = -\frac{1}{m}\left(\sum_{i=1}^{m} y^i \log\left(\hat{y}^{(i)}\right) + \sum_{i=1}^{m}\left(1 - y^{(i)}\right)\log\left(1 - \hat{y}^{(i)}\right)\right)$$

Backpropagation: Go Forwards, then Backwards…

Step 1: Calculate $\hat{y}$ using computation graph.



Step 3: Update *each* parameter (Using the partial derivative of cost).

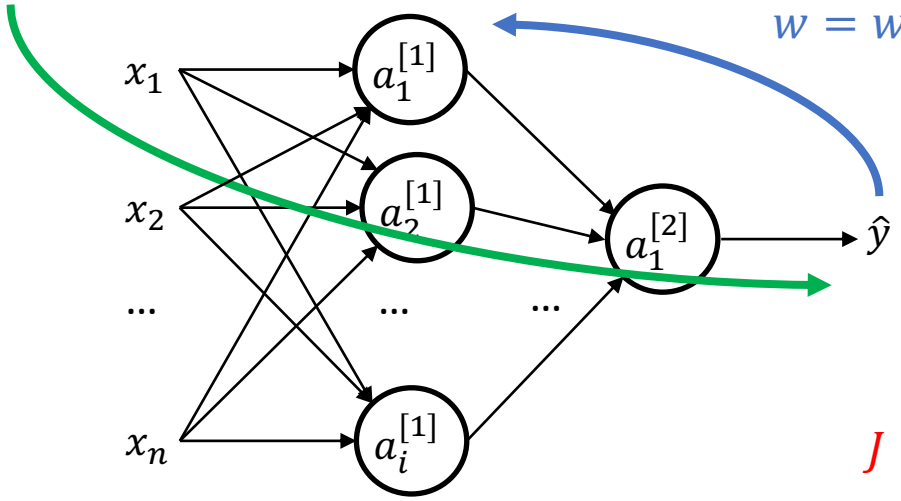
$$w = w - \alpha \frac{\partial J}{\partial w}; b = b - \alpha \frac{\partial J}{\partial b}$$

Step 2: Determine the loss.

$$J = -\frac{1}{m}\left(\sum_{i=1}^{m} y^i \log\left(\hat{y}^{(i)}\right) + \sum_{i=1}^{m}\left(1 - y^{(i)}\right)\log\left(1 - \hat{y}^{(i)}\right)\right)$$

Backpropagation: Go Forwards, then Backwards…

Step 1: Calculate $\hat{y}$ using computation graph.



Step 3: Update *each* parameter (Using the partial derivative of cost).

$$w = w - \alpha \frac{\partial J}{\partial w}; b = b - \alpha \frac{\partial J}{\partial b}$$

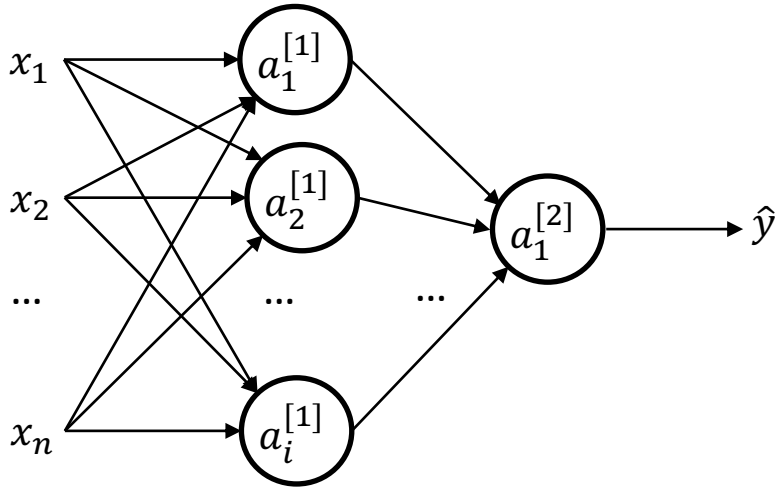
Step 2: Determine the loss.

$$J = -\frac{1}{m}\left(\sum_{i=1}^{m} y^i \log\left(\hat{y}^{(i)}\right) + \sum_{i=1}^{m}\left(1 - y^{(i)}\right)\log\left(1 - \hat{y}^{(i)}\right)\right)$$

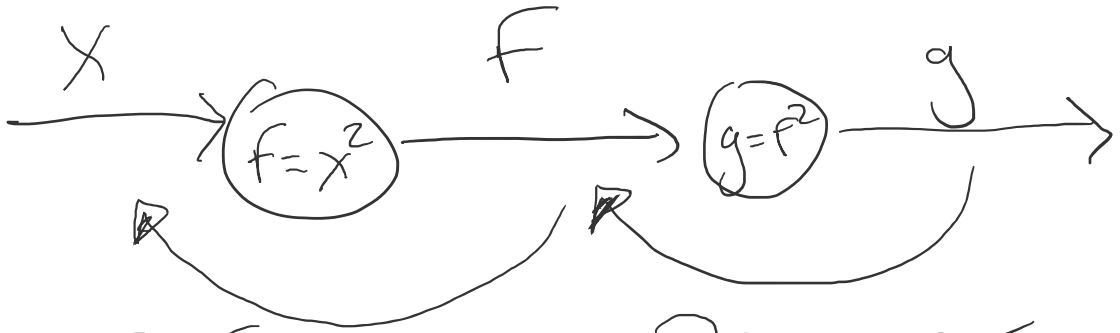
Step 4: Repeat until J < target.

Brad Quinton, Scott Chin

What about the Partial Derivatives?



graph:



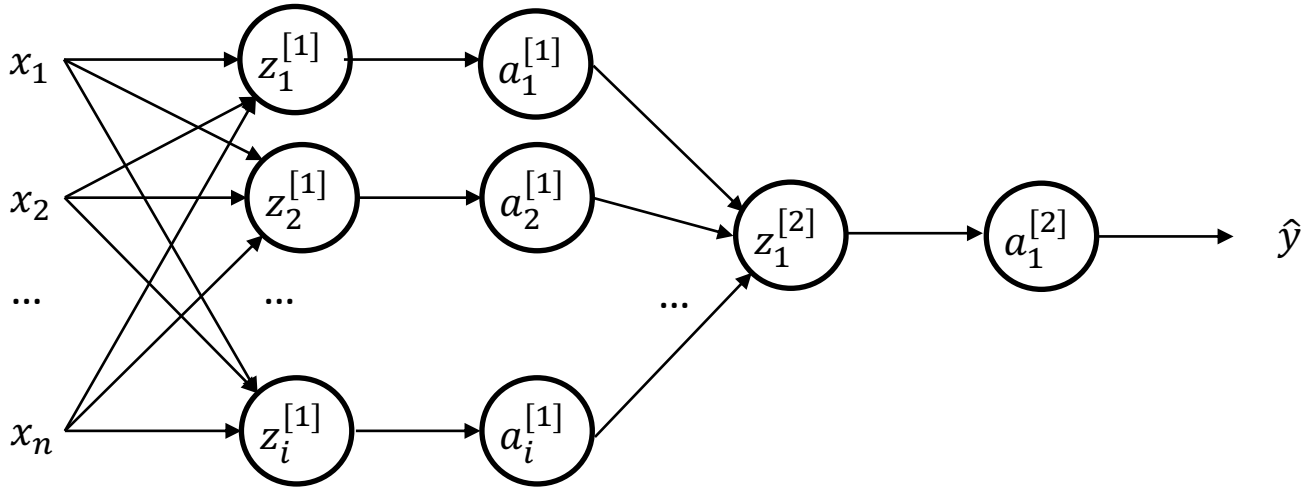
how does g change w.r.t. x?

$$\frac{\partial f}{\partial x} = 2x \qquad \frac{\partial g}{\partial f} = 2f \Rightarrow \frac{\partial f}{\partial x}\cdot\frac{\partial g}{\partial f} = \frac{\partial g}{\partial x} = 4xf = 4x^3$$

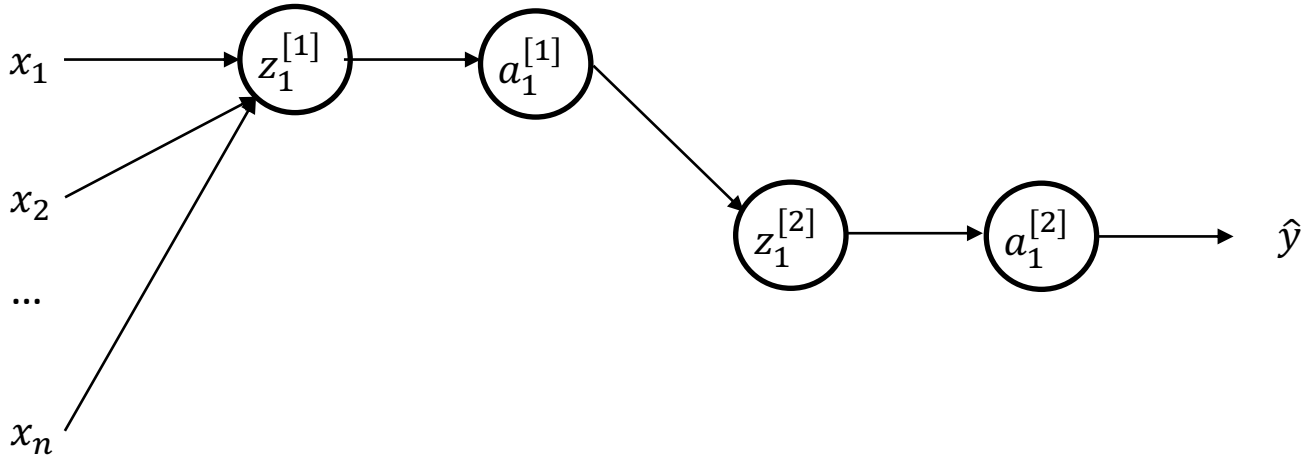
$$g = f^2 = (x^2)^2 = x^4 \Rightarrow \frac{\partial g}{\partial x} = 4x^3$$ Same!

we can use the computation graph to determine partial derivative "piece-wise"!

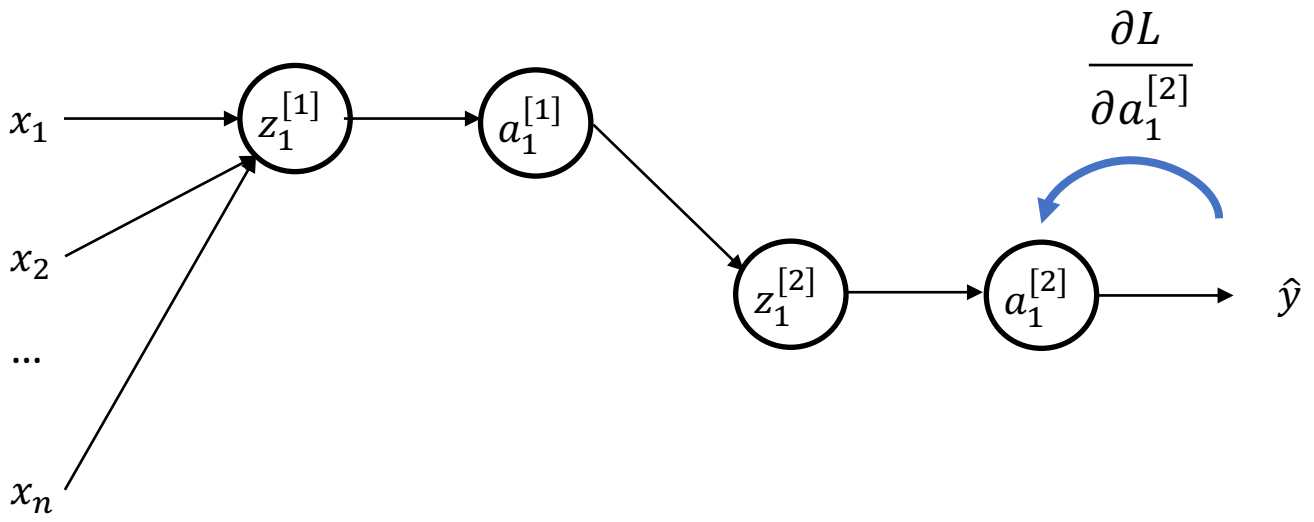
What about the Partial Derivatives?



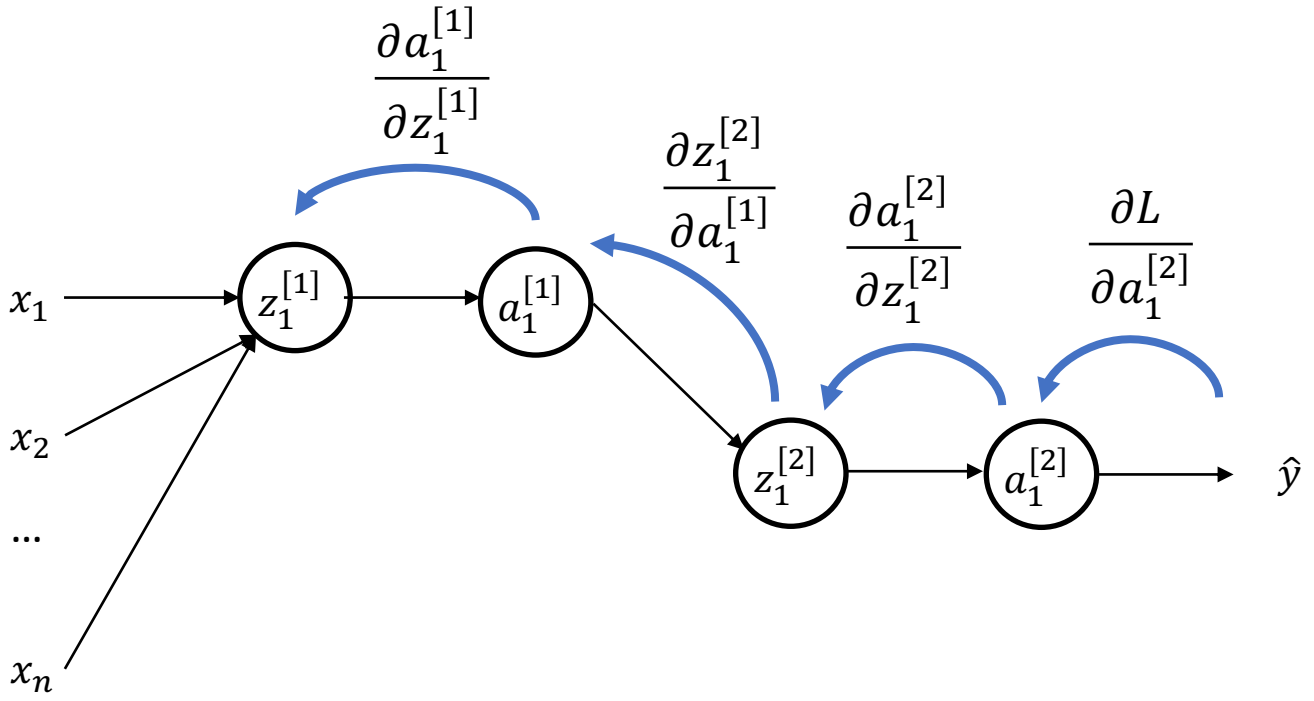
What about the Partial Derivatives?



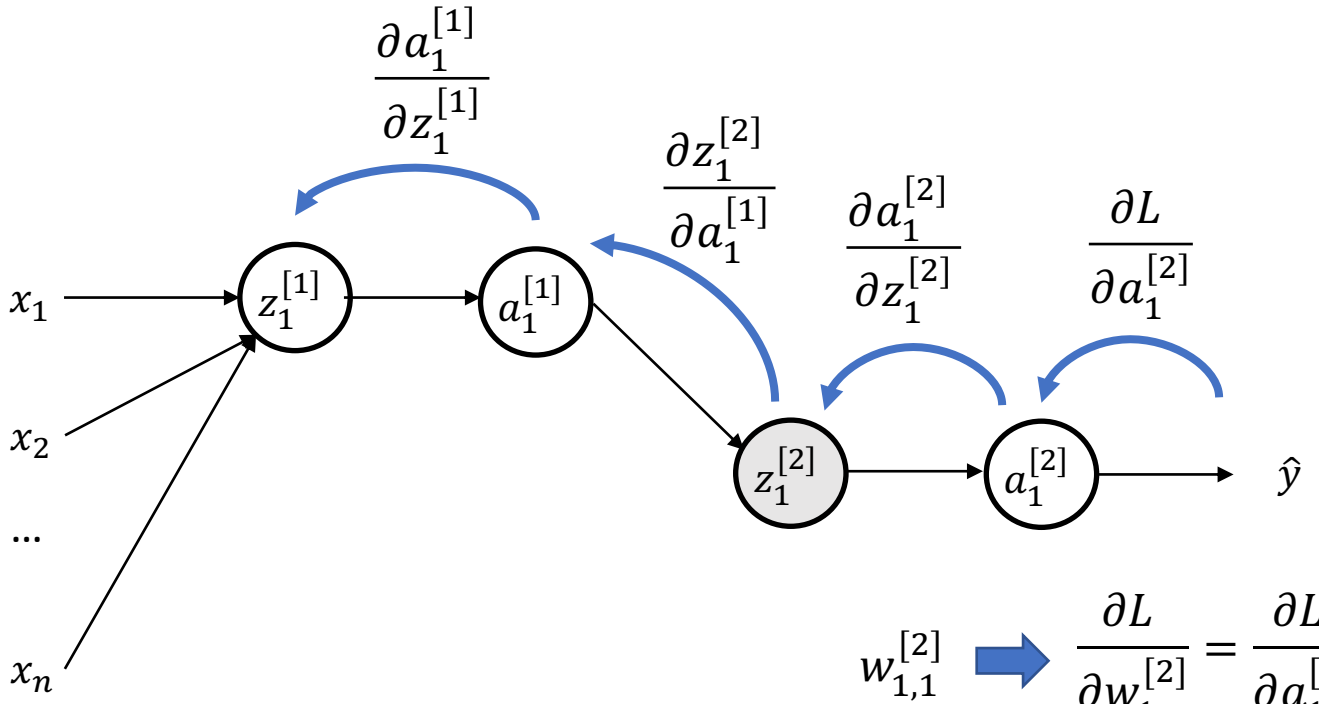
What about the Partial Derivatives?



What about the Partial Derivatives?



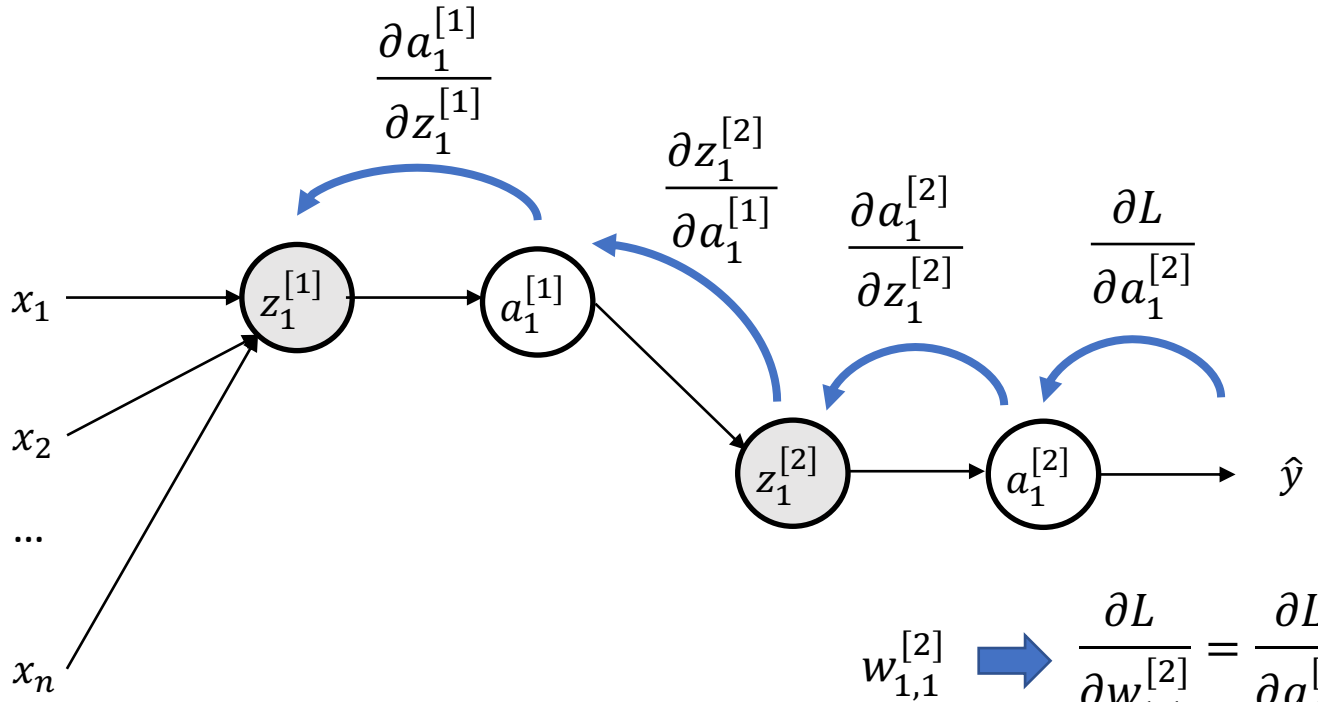
What about the Partial Derivatives?



$$w_{1,1}^{[2]} \Rightarrow \frac{\partial L}{\partial w_1^{[2]}} = \frac{\partial L}{\partial a_1^{[2]}} \cdot \frac{\partial a_1^{[2]}}{\partial z_1^{[2]}} \cdot \frac{\partial z_1^{[2]}}{\partial w_{1,1}^{[2]}}$$

$$b^{[2]} \Rightarrow \frac{\partial L}{\partial b^{[2]}} = \frac{\partial L}{\partial a_1^{[2]}} \cdot \frac{\partial a_1^{[2]}}{\partial z_1^{[2]}} \cdot \frac{\partial z_1^{[2]}}{\partial b^{[2]}}$$

What about the Partial Derivatives?



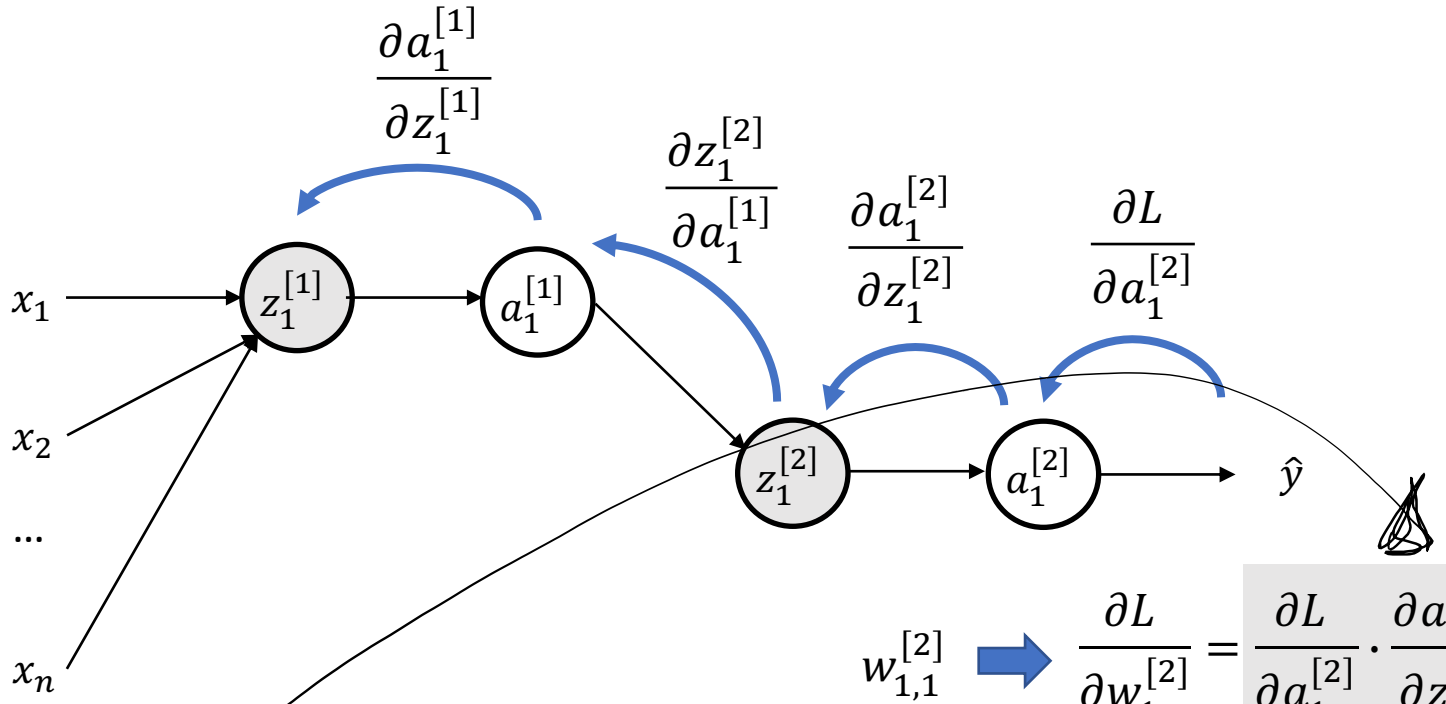
$$w_{1,1}^{[2]} \Rightarrow \frac{\partial L}{\partial w_{1,1}^{[2]}} = \frac{\partial L}{\partial a_1^{[2]}} \cdot \frac{\partial a_1^{[2]}}{\partial z_1^{[2]}} \cdot \frac{\partial z_1^{[2]}}{\partial w_{1,1}^{[2]}}$$

$$w_1^{[1]} \Rightarrow \frac{\partial L}{\partial w_{1,1}^{[1]}} = \frac{\partial L}{\partial a_1^{[2]}} \cdot \frac{\partial a_1^{[2]}}{\partial z_1^{[2]}} \cdot \frac{\partial z_1^{[2]}}{\partial a_1^{[1]}} \cdot \frac{\partial a_1^{[1]}}{\partial z_1^{[1]}} \cdot \frac{\partial z_1^{[1]}}{\partial w_{1,1}^{[1]}}$$

$$b^{[2]} \Rightarrow \frac{\partial L}{\partial b^{[2]}} = \frac{\partial L}{\partial a_1^{[2]}} \cdot \frac{\partial a_1^{[2]}}{\partial z_1^{[2]}} \cdot \frac{\partial z_1^{[2]}}{\partial b^{[2]}}$$

$$b^{[1]} \Rightarrow \frac{\partial L}{\partial b^{[1]}} = \frac{\partial L}{\partial a_1^{[2]}} \cdot \frac{\partial a_1^{[2]}}{\partial z_1^{[2]}} \cdot \frac{\partial z_1^{[2]}}{\partial a_1^{[1]}} \cdot \frac{\partial a_1^{[1]}}{\partial z_1^{[1]}} \cdot \frac{\partial z_1^{[1]}}{\partial b^{[1]}}$$

What about the Partial Derivatives?



$$w_{1,1}^{[1]} \Rightarrow \frac{\partial L}{\partial w_1^{[1]}} = \frac{\partial L}{\partial a_1^{[2]}} \cdot \frac{\partial a_1^{[2]}}{\partial z_1^{[2]}} \cdot \frac{\partial z_1^{[2]}}{\partial a_1^{[1]}} \cdot \frac{\partial a_1^{[1]}}{\partial z_1^{[1]}} \cdot \frac{\partial z_1^{[1]}}{\partial w_1^{[1]}}$$

$$b^{[1]} \Rightarrow \frac{\partial L}{\partial b^{[1]}} = \frac{\partial L}{\partial a_1^{[2]}} \cdot \frac{\partial a_1^{[2]}}{\partial z_1^{[2]}} \cdot \frac{\partial z_1^{[2]}}{\partial a_1^{[1]}} \cdot \frac{\partial a_1^{[1]}}{\partial z_1^{[1]}} \cdot \frac{\partial z_1^{[1]}}{\partial b^{[1]}}$$

$$w_{1,1}^{[2]} \Rightarrow \frac{\partial L}{\partial w_1^{[2]}} = \frac{\partial L}{\partial a_1^{[2]}} \cdot \frac{\partial a_1^{[2]}}{\partial z_1^{[2]}} \cdot \frac{\partial z_1^{[2]}}{\partial w_{1,1}^{[2]}}$$

$$b^{[2]} \Rightarrow \frac{\partial L}{\partial b^{[2]}} = \frac{\partial L}{\partial a_1^{[2]}} \cdot \frac{\partial a_1^{[2]}}{\partial z_1^{[2]}} \cdot \frac{\partial z_1^{[2]}}{\partial b^{[2]}}$$

Notice how the same derivatives are re-used "across" and "back" through the NN… Brad Quinton, Scott Chin

Remembering Logistic Regression

From Logistic Regression:

$$\frac{\partial L}{\partial a} = -\frac{y}{a} + \frac{1-y}{1-a} = \frac{a-y}{a(1-a)}$$

$$\frac{\partial a}{\partial z} = \sigma(z)(1-\sigma(z)) = a(1-a)$$

$$\frac{\partial z}{\partial w_1} = x_1, \frac{\partial z}{\partial w_2} = x_2, \dots, \frac{\partial z}{\partial b} = 1$$

Last layer in the NN:

$$\frac{\partial L}{\partial a_1^{[2]}} = \frac{\hat{y}-y}{\hat{y}(1-\hat{y})}$$

$$\frac{\partial a_1^{[2]}}{\partial z_1^{[2]}} = \hat{y}(1-\hat{y})$$

$$\frac{\partial z_1^{[2]}}{\partial w_{1,1}^{[2]}} = x_1^{[2]}, \dots, \frac{\partial z_1^{[2]}}{\partial b^{[2]}} = 1$$

- But what about crossing Neural Network layers?:

$$\frac{\partial z_1^{[2]}}{\partial a_1^{[1]}} = ?, \frac{\partial a_1^{[1]}}{\partial z_1^{[1]}} = ?$$

Let's Figure it Out…

- What's the equation for $z_1^{[2]}$?:

$$z_1^{[2]} = w_{1,1}^{[2]} a_1^{[1]} + w_{1,2}^{[2]} a_2^{[1]} + \ldots + w_{1,1}^{[2]} a_1^{[1]} + b_1^{[2]} \quad \Rightarrow \quad \frac{\partial z_1^{[2]}}{\partial a_1^{[1]}} = w_{1,1}^{[2]}$$

- And, $a_1^{[1]}$?:

$$a_1^{[1]} = \tanh(z_1^{[1]}) \quad \Rightarrow \quad \frac{\partial a_1^{[1]}}{\partial z_1^{[1]}} = 1 - \tanh^2(z_1^{[1]})$$

Pulling it all Together

$$\frac{\partial L}{\partial w_{1,1}^{[2]}} = \frac{\partial L}{\partial a_1^{[2]}} \cdot \frac{\partial a_1^{[2]}}{\partial z_1^{[2]}} \cdot \frac{\partial z_1^{[2]}}{\partial w_{1,1}^{[2]}} = \frac{\hat{y}-y}{\hat{y}(1-\hat{y})} \cdot \hat{y}(1-\hat{y}) \cdot x_1^{[2]} = (\hat{y}-y) \cdot x_1^{[2]}$$

$$\frac{\partial L}{\partial b^{[2]}} = \frac{\partial L}{\partial a_1^{[2]}} \cdot \frac{\partial a_1^{[2]}}{\partial z_1^{[2]}} \cdot \frac{\partial z_1^{[2]}}{\partial b^{[2]}} = \frac{\hat{y}-y}{\hat{y}(1-\hat{y})} \cdot \hat{y}(1-\hat{y}) = (\hat{y}-y)$$

$$\frac{\partial L}{\partial w_{1,1}^{[1]}} = \frac{\partial L}{\partial a_1^{[2]}} \cdot \frac{\partial a_1^{[2]}}{\partial z_1^{[2]}} \cdot \frac{\partial z_1^{[2]}}{\partial a_1^{[1]}} \cdot \frac{\partial a_1^{[1]}}{\partial z_1^{[1]}} \cdot \frac{\partial z_1^{[1]}}{\partial w_{1,1}^{[1]}} = \frac{\hat{y}-y}{\hat{y}(1-\hat{y})} \cdot \hat{y}(1-\hat{y}) \cdot w_{1,1}^{[2]} \cdot (1-\tanh^2(z_1^{[1]})) \cdot x_1 = w_{1,1}^{[2]} \cdot (\hat{y}-y) \cdot (1-\tanh^2(z_1^{[1]})) \cdot x_1$$

$$\frac{\partial L}{\partial b^{[1]}} = \frac{\partial L}{\partial a_1^{[2]}} \cdot \frac{\partial a_1^{[2]}}{\partial z_1^{[2]}} \cdot \frac{\partial z_1^{[2]}}{\partial a_1^{[1]}} \cdot \frac{\partial a_1^{[1]}}{\partial z_1^{[1]}} \cdot \frac{\partial z_1^{[1]}}{\partial w_1^{[1]}} = \frac{\hat{y}-y}{\hat{y}(1-\hat{y})} \cdot \hat{y}(1-\hat{y}) \cdot w_{1,1}^{[2]} \cdot (1-\tanh^2(z_1^{[1]})) = w_{1,1}^{[2]} \cdot (\hat{y}-y) \cdot (1-\tanh^2(z_1^{[1]}))$$

$$\frac{\partial L}{\partial z_1^{[2]}} = (\hat{y}-y)$$

$$\frac{\partial L}{\partial z_1^{[1]}} = w_{1,1}^{[2]} \cdot (\hat{y}-y) \cdot (1-\tanh^2(z_1^{[1]}))$$

Brad Quinton, Scott Chin

Final Equations

(1) $$\frac{\partial L}{\partial z_1^{[2]}} = (\hat{y} - y)$$

(2) $$\frac{\partial L}{\partial w_{1,1}^{[2]}} = \frac{\partial L}{\partial z_1^{[2]}} \cdot x_1^{[2]} = \frac{\partial L}{\partial z_1^{[2]}} \cdot a_1^{[1]}$$

(3) $$\frac{\partial L}{\partial b^{[2]}} = \frac{\partial L}{\partial z_1^{[2]}}$$

(4) $\frac{\partial L}{\partial z_1^{[1]}} = w_1^{[2]} \cdot \frac{\partial L}{\partial z_1^{[2]}} \cdot \mathrm{g}'(z_1^{[1]})$, where $g(z_1^{[1]})$ is the activation function for $a_1^{[1]}$

(5) $$\frac{\partial L}{\partial w_{1,1}^{[1]}} = \frac{\partial L}{\partial z_1^{[1]}} \cdot x_1$$

(6) $$\frac{\partial L}{\partial b^{[1]}} = \frac{\partial L}{\partial z_1^{[1]}}$$

Brad Quinton, Scott Chin

Wait, what does it all mean?

- It is easy to get lost in the backpropagation derivation, but it is important not lose sight of what we are doing:

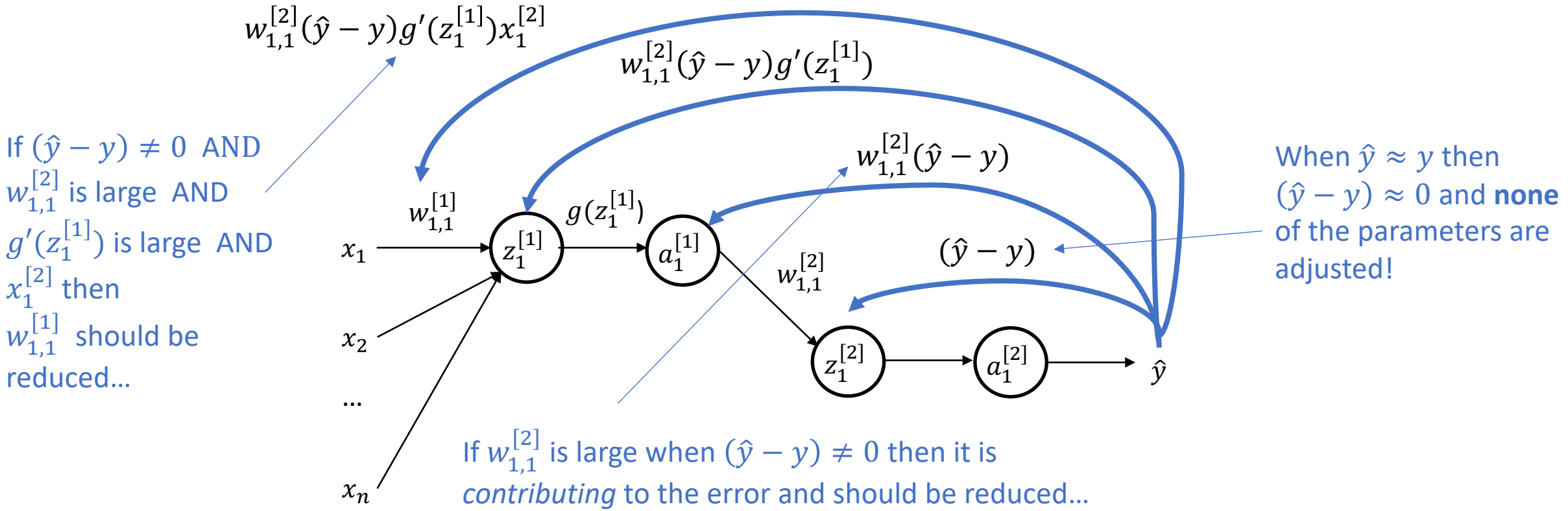
1. Combined a set of Logistic Regression units using a computation graph representation.

2. Applying a cost function to the output of the computation graph.

3. Deriving partial differential equations w.r.t. the cost function to understand “where our cost is going” with respect to each parameter.

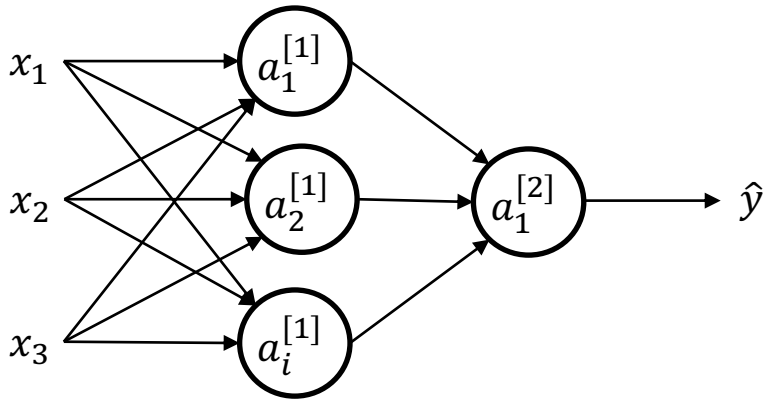
“Backpropagating” the Error

- One interpretation of this is that we are propagating the error and attributing it to each node (and then each parameter).



There is no “Magic” in the Network

- It is important to realize that the graphical representation of the network is simply a computational and notational convenience



$$\equiv \quad \hat{y} = \sigma($$
$$w_1^{[2]} tanh\left(w_{1,1}^{[1]} x_1^{[1]} + w_{1,2}^{[1]} x_2^{[1]} + w_{1,3}^{[1]} x_3^{[1]} + b_1^{[1]}\right) +$$
$$w_2^{[2]} tanh\left(w_{2,1}^{[1]} x_1^{[1]} + w_{2,2}^{[1]} x_2^{[1]} + w_{2,3}^{[1]} x_3^{[1]} + b_2^{[1]}\right) +$$
$$w_3^{[2]} tanh\left(w_{3,1}^{[1]} x_1^{[1]} + w_{3,2}^{[1]} x_2^{[1]} + w_{3,3}^{[1]} x_3^{[1]} + b_3^{[1]}\right) +$$
$$b_1^{[2]})$$

- We can equally well for each view assign a cost function and then iteratively search for the “best” parameters…

Why Can’t We Simply Solve for Parameters?

- We know the cost function, and it is ”differentiable” so why can’t we just set $J'(w, b) = 0$ and solve for w, b?

- Unfortunately, the problem is under constrained (we don’t have enough equations!)

- There are a large number of parameters, $\{w, b\}$ and our data only represents specific “points” in space, not a complete relationship….

- All we can only evaluate function if we pick a set of $\{w, b\}$

So, Where are we in Case Study?

- So, it looks like Neural Networks have potential, but we still have a ways to go:
  - What configuration of Neural Network do we use? How many hidden layers/units?
  - How do we implement backpropagation it rationally/efficiently?
  - How do we know if we are succeeding?
- We will cover all this (and more) next lecture!

Key Take-Aways

- An (Artificial) Neural Network is a computation graph loosely inspired by biological neurons, where each ”Neuron” is essentially a logistic regression unit
- By combining “Neurons”, we can create very complex functions
- The “shape” of the resulting function is defined by the parameters of each Logistic Regression unit working together
- An algorithm called backpropagation uses gradient descent to update the parameters efficiently