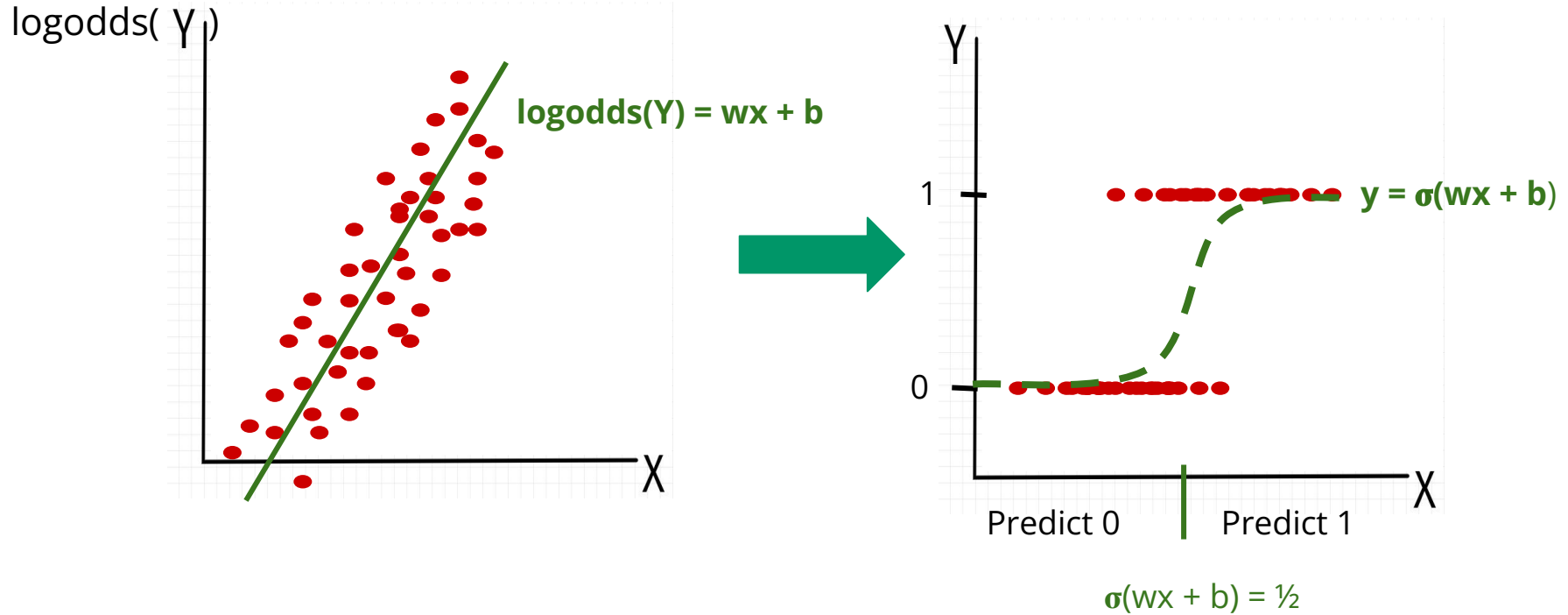

Neural Networks

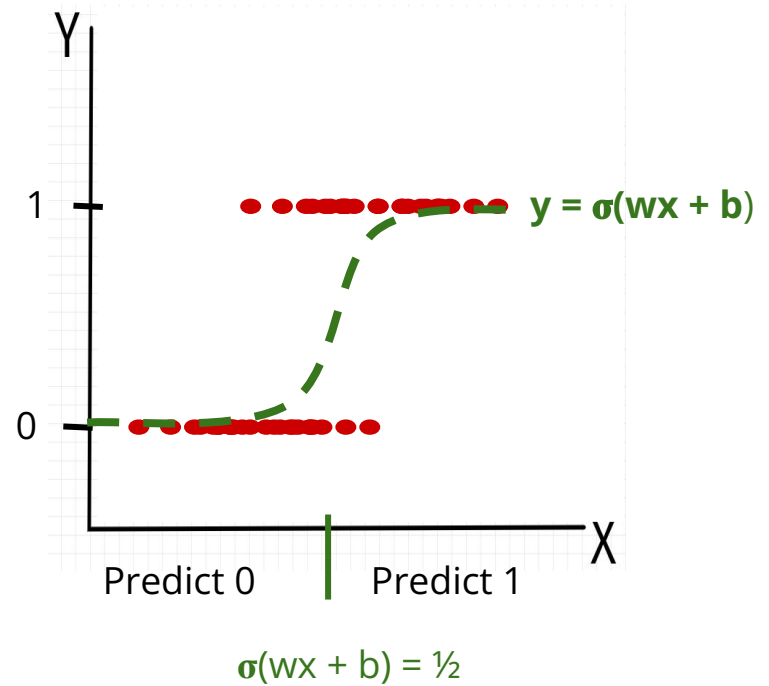
— Boston University CS 506 - Lance Galletti —

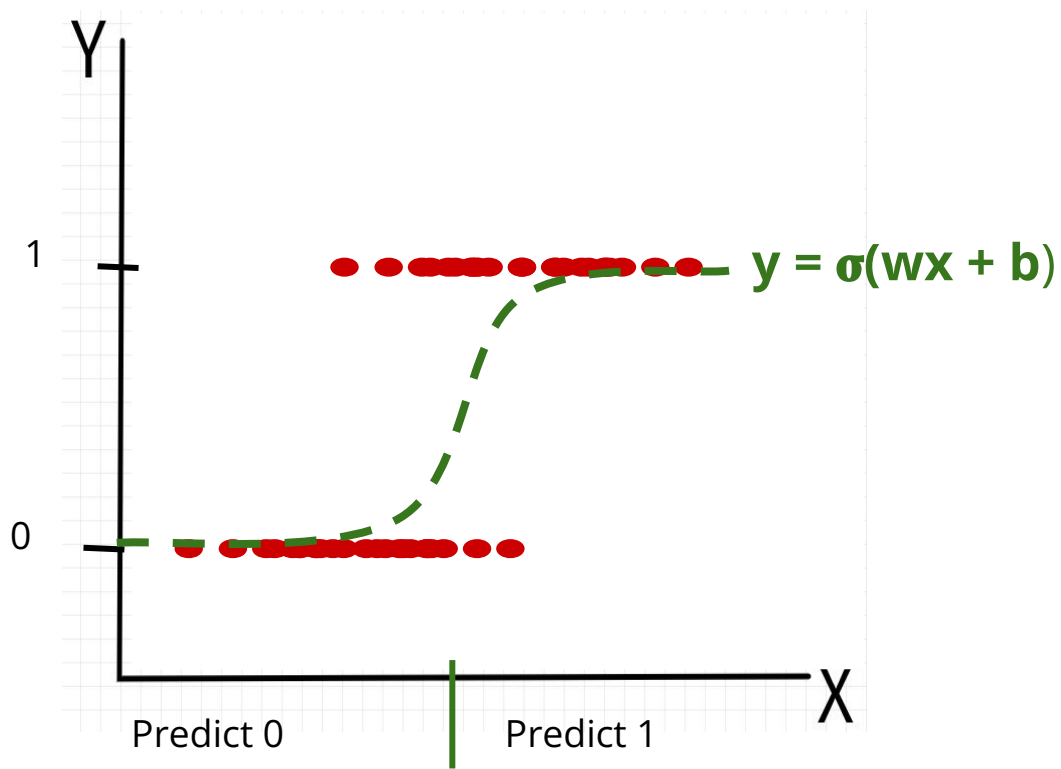
DECISION RULE:
IF $P(Y=1 | X) > \frac{1}{2}$ THEN 1 ELSE 0



DECISION RULE:

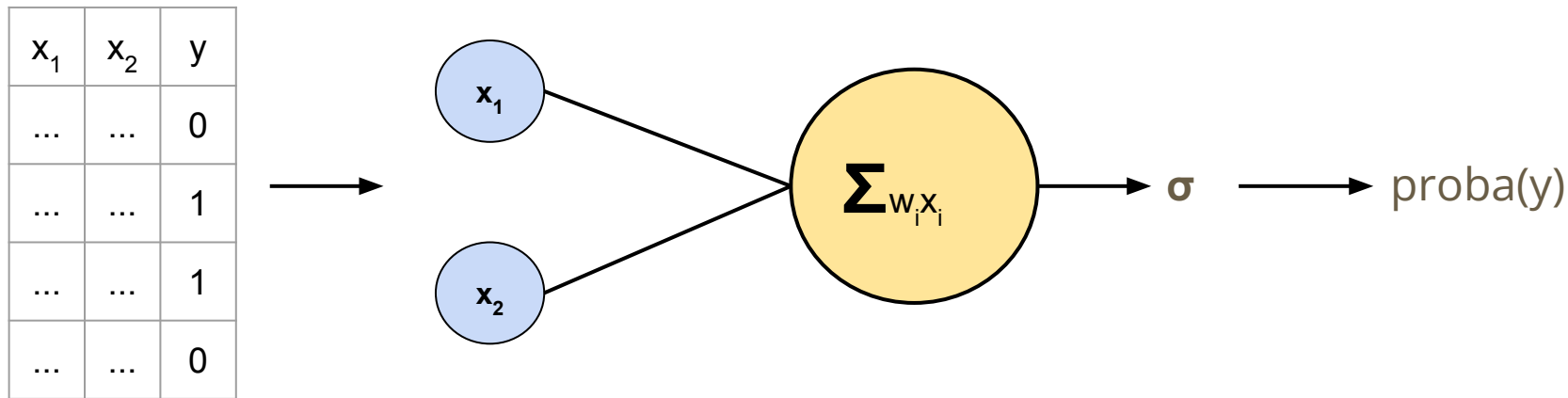
IF $P(Y=1 | X) > \frac{1}{2}$ THEN 1 ELSE 0



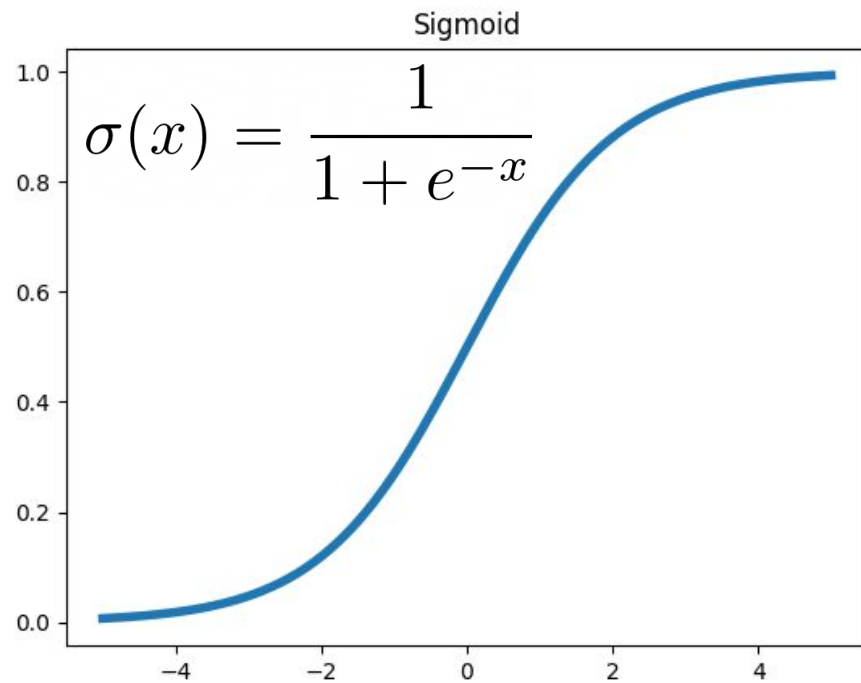


$$\sigma(wx + b) = \frac{1}{2}$$

Logistic Regression Revisited

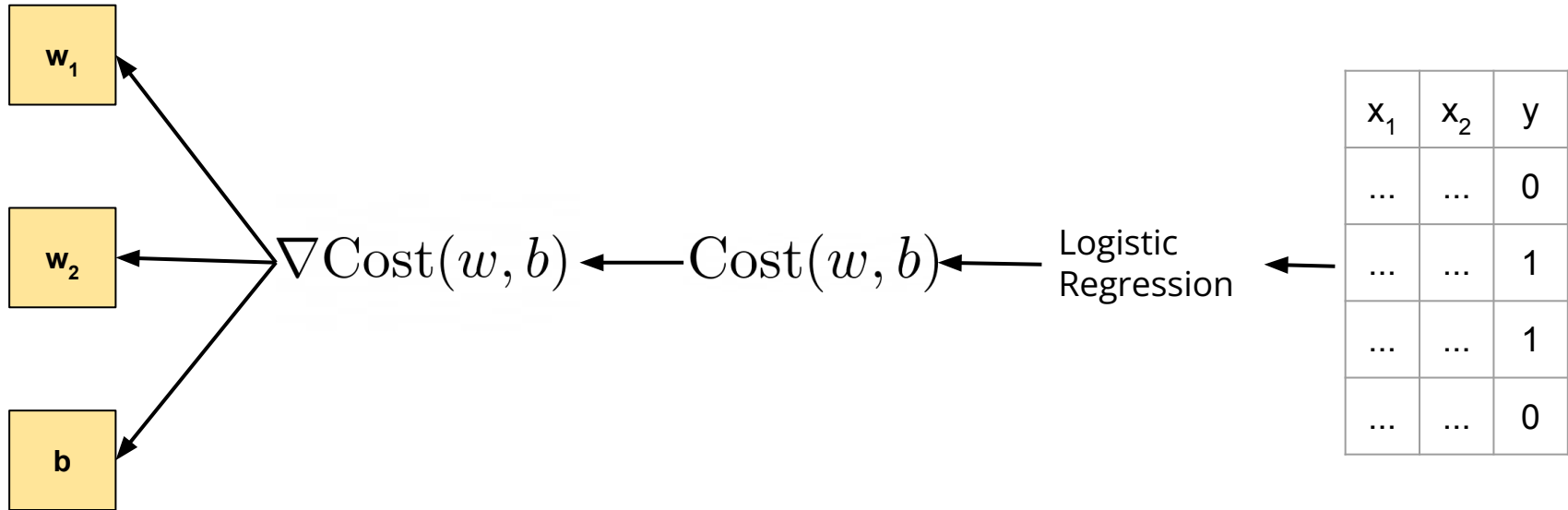


where
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

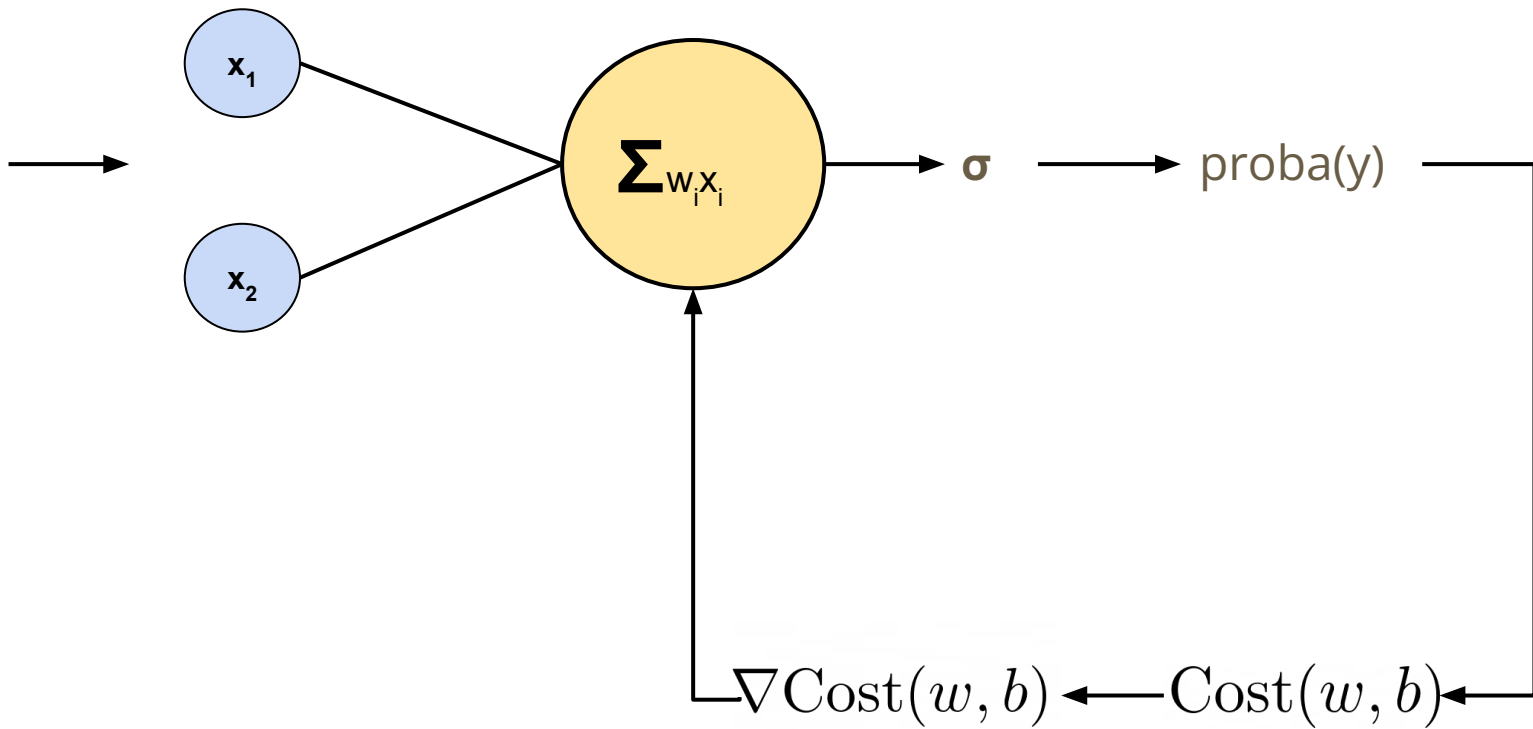


$$\begin{aligned}\max \prod_{i=1}^n P(y_i|x_i) &= \prod_i (\text{logit}^{-1}(w^T x_i + b))^{y_i} (1 - \text{logit}^{-1}(w^T x_i + b))^{1-y_i} \\&= \min -\frac{1}{n} \sum_{i=1}^n [y_i \log(\sigma(-w^T x_i + b)) + (1 - y_i) \log(1 - \sigma(-w^T x_i + b))] \\&= \min \text{Cost}(w, b)\end{aligned}$$

Gradient Descent



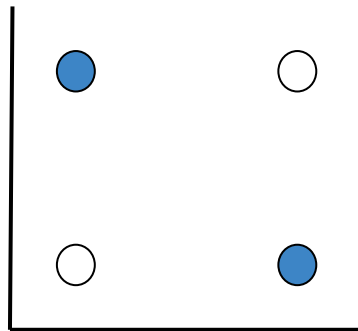
x_1	x_2	y
...	...	0
...	...	1
...	...	1
...	...	0



Logistic Regression Revisited

XOR function

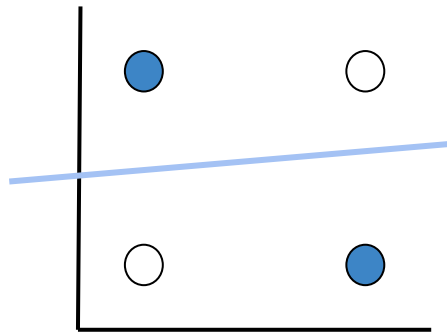
x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0



Logistic Regression Revisited

XOR function

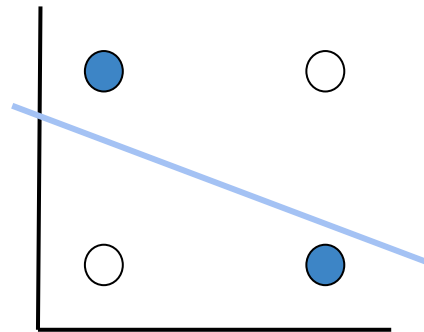
x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0



Logistic Regression Revisited

XOR function

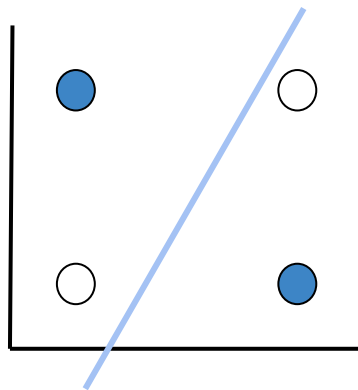
x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0



Logistic Regression Revisited

XOR function

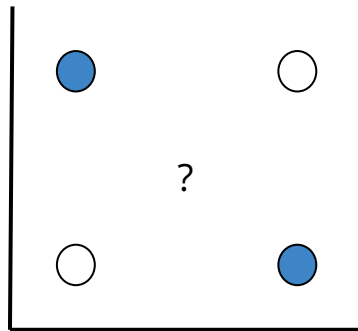
x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0



Logistic Regression Revisited

XOR function

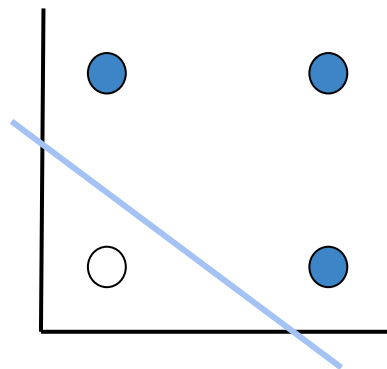
x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0



Logistic Regression Revisited

Recall, the **OR** function is linearly separable:

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	1



Logistic Regression Revisited

$$\begin{aligned}\mathbf{XOR}(x_1, x_2) &= \mathbf{OR}(\mathbf{AND}(x_1 = 0, x_2 = 1), \mathbf{AND}(x_1 = 1, x_2 = 0)) \\ &= (x_1 = 0 \wedge x_2 = 1) \vee (x_1 = 1 \wedge x_2 = 0)\end{aligned}$$

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0

Logistic Regression Revisited

$$\mathbf{XOR}(x_1, x_2) = \mathbf{OR}(\mathbf{AND}(x_1 = 0, x_2 = 1), \mathbf{AND}(x_1 = 1, x_2 = 0))$$

$$= (x_1 = 0 \wedge x_2 = 1) \vee (x_1 = 1 \wedge x_2 = 0)$$

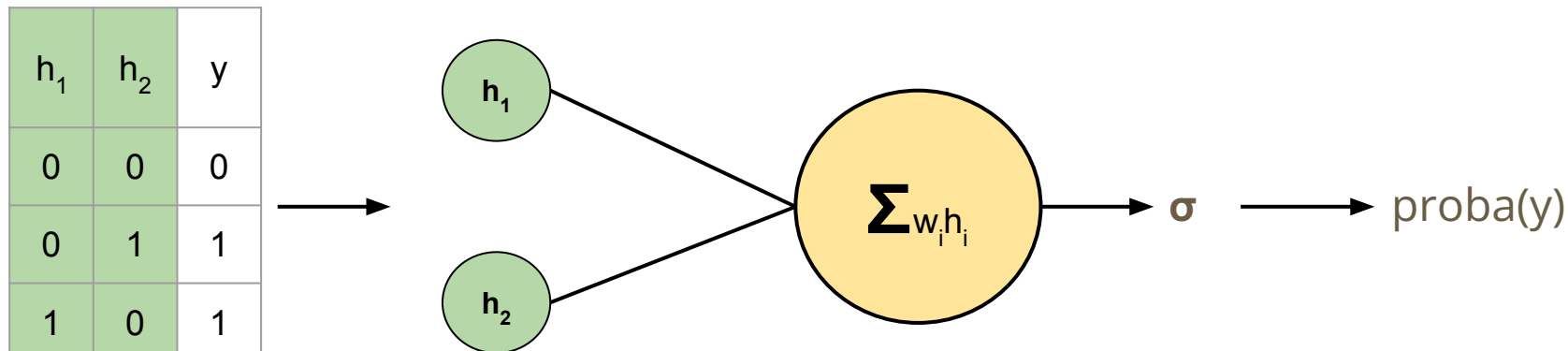
$$= h_1 \vee h_2$$

$$h_1 = \mathbf{AND}(x_1 = 0, x_2 = 1)$$

$$h_2 = \mathbf{AND}(x_1 = 1, x_2 = 0)$$

x_1	x_2	h_1	h_2	y
0	0	0	0	0
1	0	0	1	1
0	1	1	0	1
1	1	0	0	0

Logistic Regression Revisited

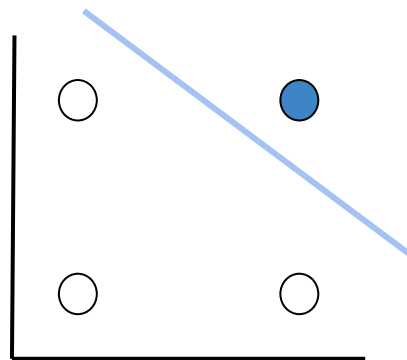


where
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Logistic Regression Revisited

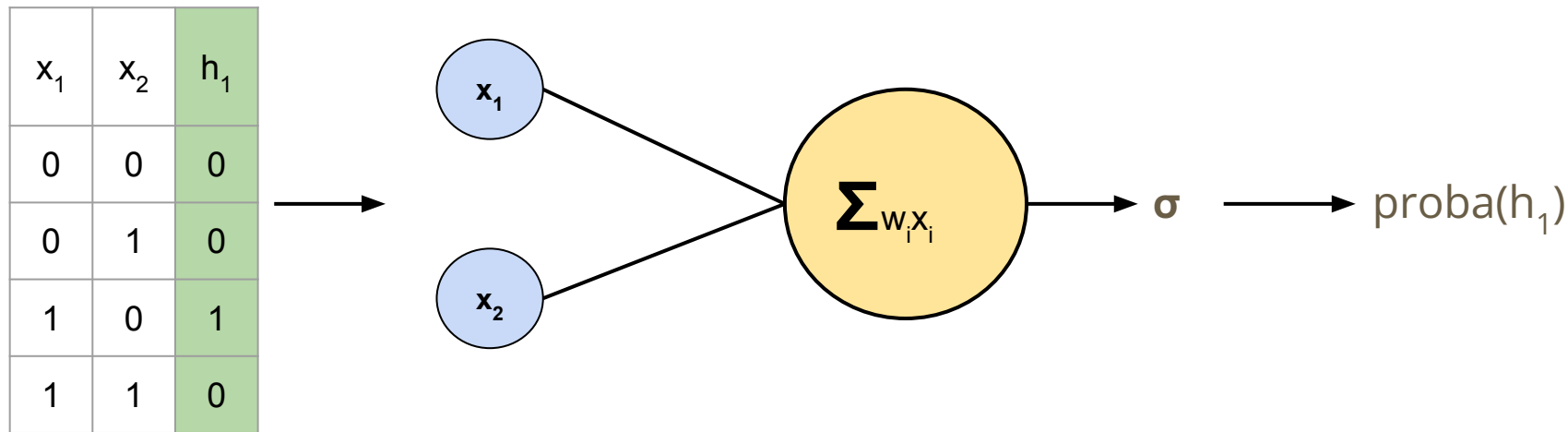
But, the **AND** function is also linearly separable:

x_1	x_2	y
0	0	0
1	0	0
0	1	0
1	1	1



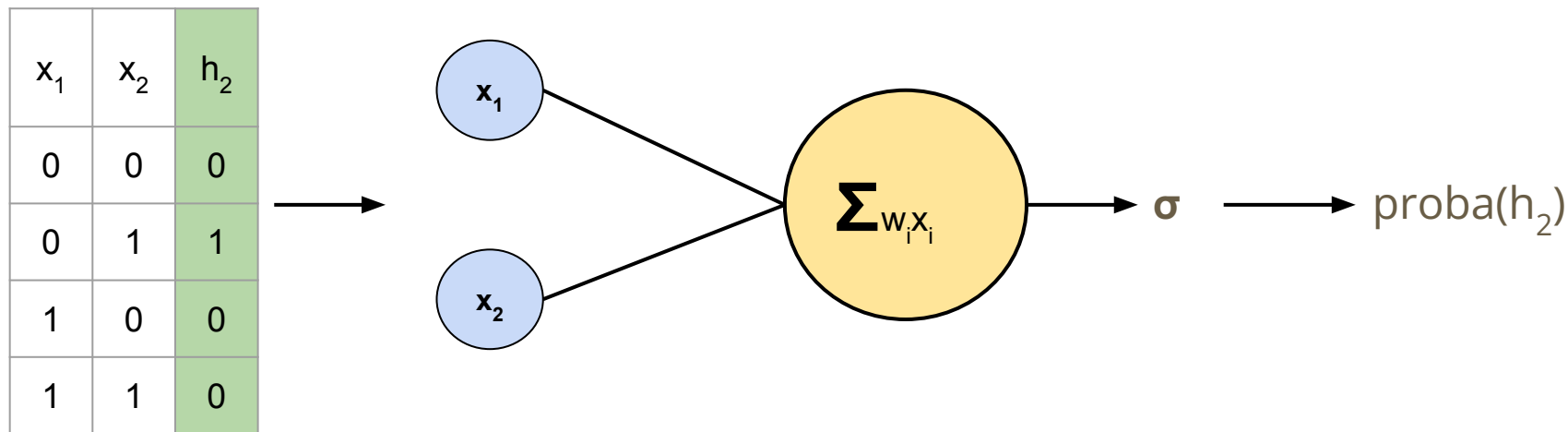
Logistic Regression Revisited

Since we can learn h_1 and h_2 automatically through logistic regression



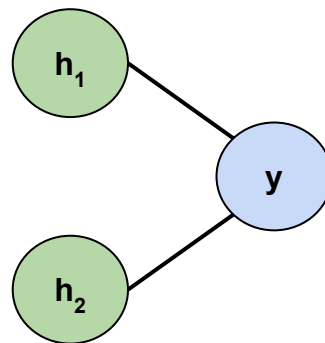
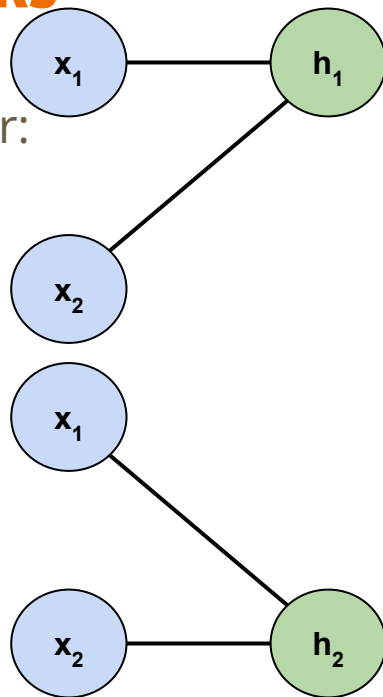
Logistic Regression Revisited

Since we can learn h_1 and h_2 automatically through logistic regression



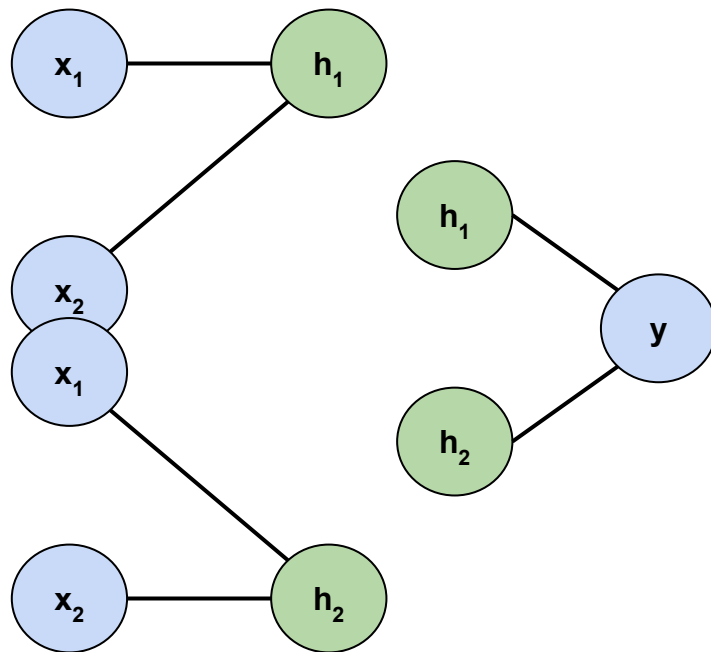
Neural Networks

Putting it all together:



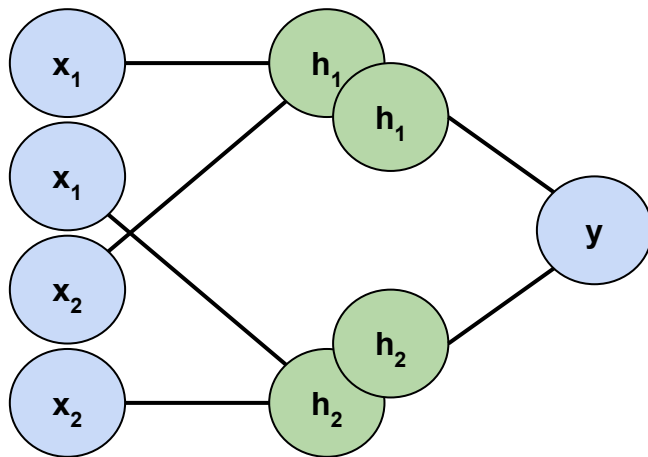
Neural Networks

Putting it all together:



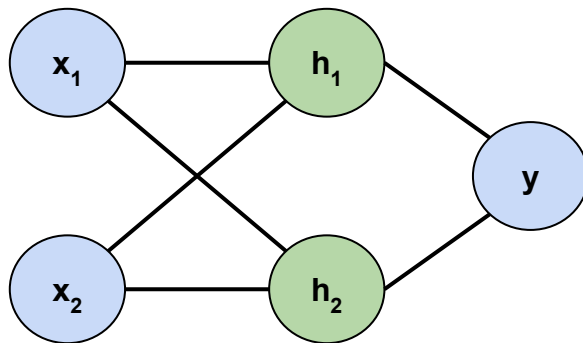
Neural Networks

Putting it all together:

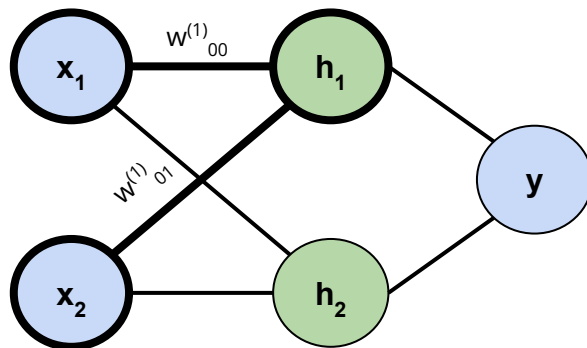


Neural Networks

Putting it all together:

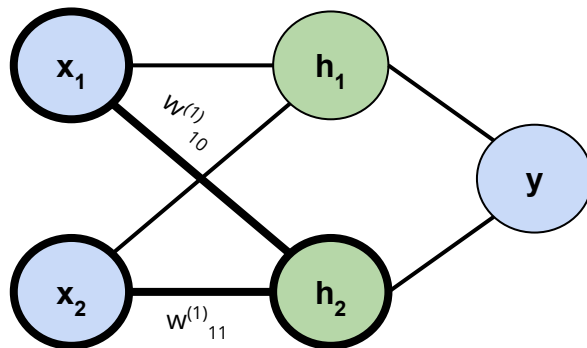


Neural Networks



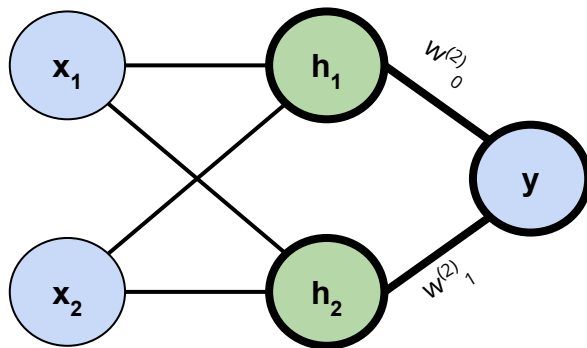
$$h_1 = \sigma(w^{(1)}_{00} x_1 + w^{(1)}_{01} x_2 + b^{(1)}_1)$$

Neural Networks



$$h_2 = \sigma(w_{10}^{(1)} x_1 + w_{11}^{(1)} x_2 + b_{2}^{(1)})$$

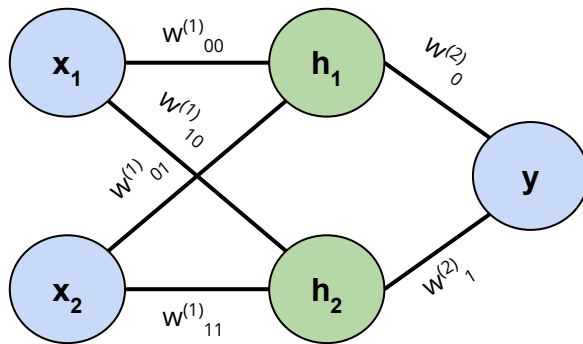
Neural Networks



$$y = \sigma(w^{(2)}_0 h_1 + w^{(2)}_1 h_2 + b^{(2)}_1)$$

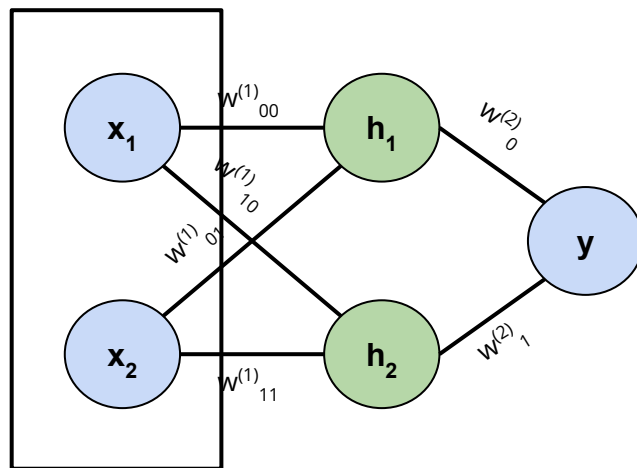
Neural Networks

Putting it all together:



Neural Networks

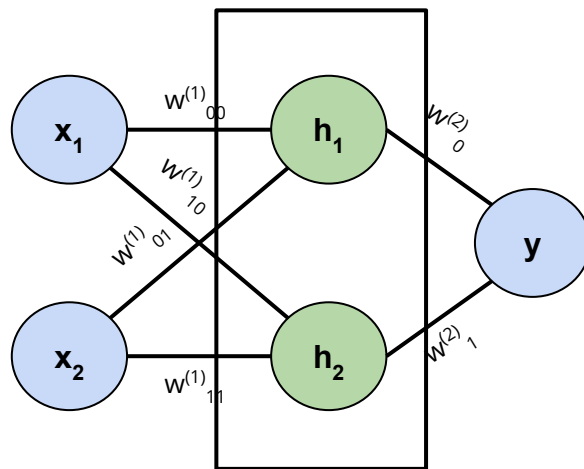
Putting it all together:



Input layer

Neural Networks

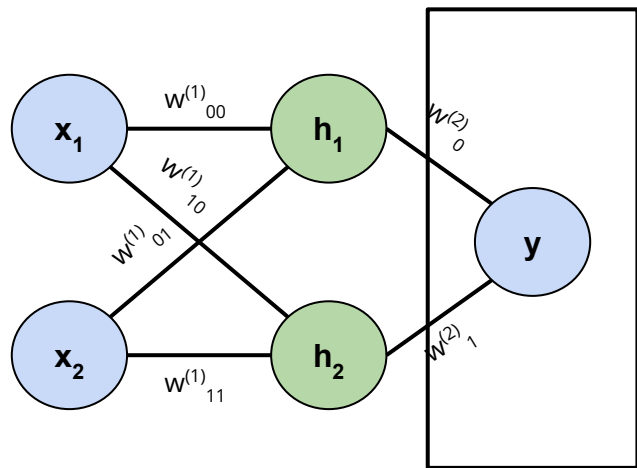
Putting it all together:



Hidden layer

Neural Networks

Putting it all together:



Output layer

Neural Networks

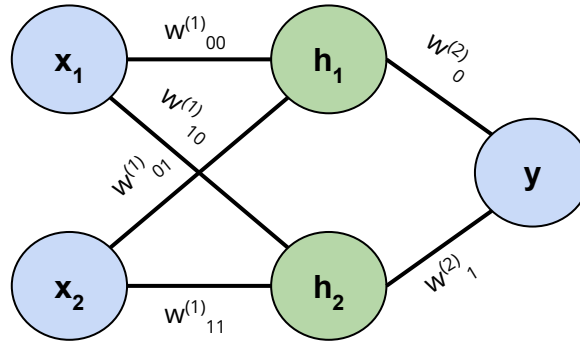
It's all about learning features (created in the hidden layer(s)) automatically

Neural Networks

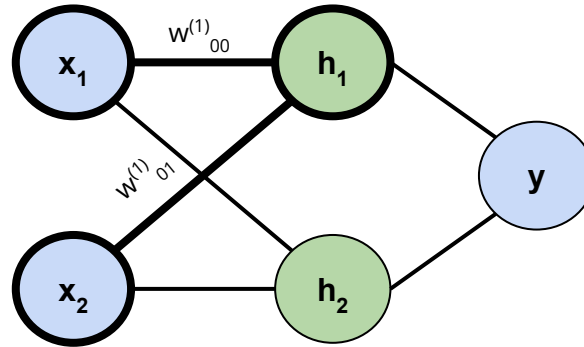
We need to define:

1. How input flows through the network to get the output (forward propagation)
2. How the weights and biases gets updated (Backpropagation)

Neural Networks - Forward Propagation

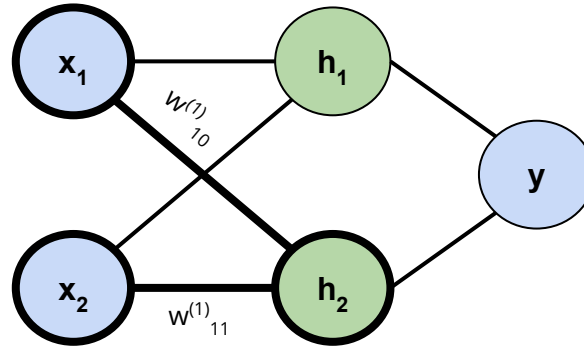


Neural Networks - Forward Propagation



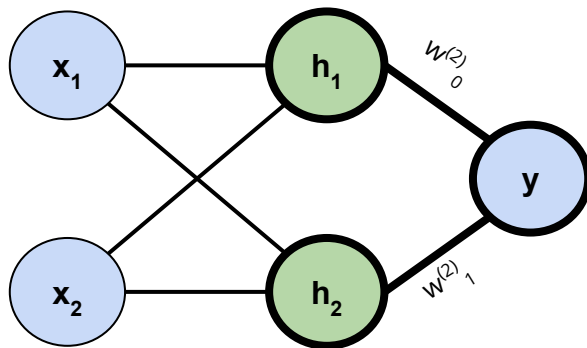
$$h_1 = \sigma(w_{00}^{(1)} x_1 + w_{01}^{(1)} x_2 + b_1^{(1)})$$

Neural Networks - Forward Propagation



$$h_2 = \sigma(w_{10}^{(1)} x_1 + w_{11}^{(1)} x_2 + b_{2}^{(1)})$$

Neural Networks - Forward Propagation



$$y = \sigma(w^{(2)}_0 h_1 + w^{(2)}_1 h_2 + b^{(2)}_1)$$

Neural Networks - Forward Propagation

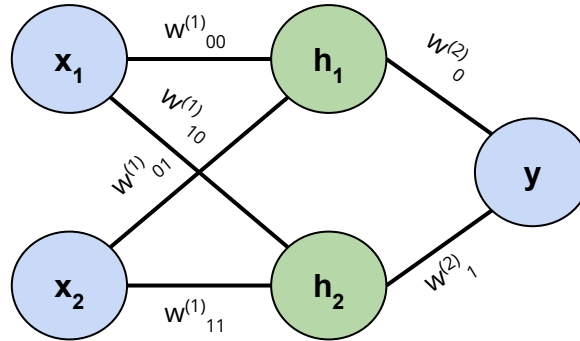
Using matrix notation:

$$\begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = \sigma \left(\begin{bmatrix} w_{00}^{(1)} & w_{01}^{(1)} \\ w_{10}^{(1)} & w_{11}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \end{bmatrix} \right)$$

$$y = \sigma \left(\begin{bmatrix} w_{00}^{(2)} \\ w_{01}^{(2)} \end{bmatrix}^T \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} + b^{(2)} \right)$$

Neural Networks - Forward Propagation

Q: if all the weights and biases are initialized to 0, what will be the output of the network?



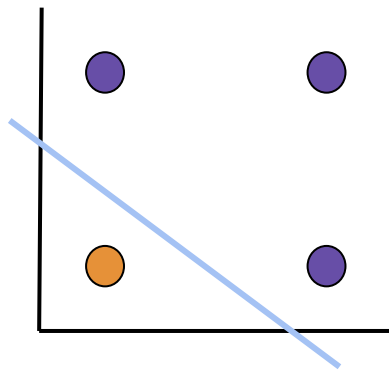
Recall in Logistic Regression

Decision Boundary is where

$$\sigma(\mathbf{w}\mathbf{x}+\mathbf{b}) = \frac{1}{2}$$

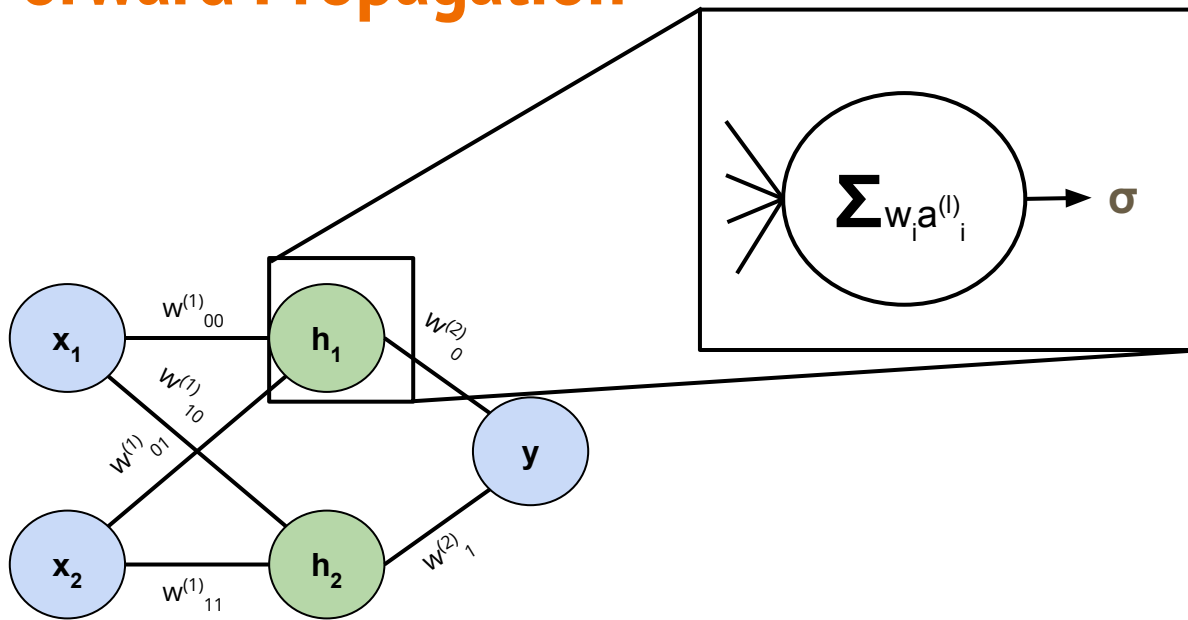
which is exactly where

$$\mathbf{w}\mathbf{x}+\mathbf{b} = 0$$



Neural Networks - Forward Propagation

Q: what happens if we don't have σ in the hidden layer here? What will the decision boundary look like? What will our features be?



Neural Networks - Forward Propagation

If we don't, we just end up with normal logistic regression on x_1 and x_2 .

$$h_1 = w_{00}^{(1)} x_1 + w_{01}^{(1)} x_2 + b_1^{(1)}$$

$$h_2 = w_{10}^{(1)} x_1 + w_{11}^{(1)} x_2 + b_2^{(1)}$$

Then

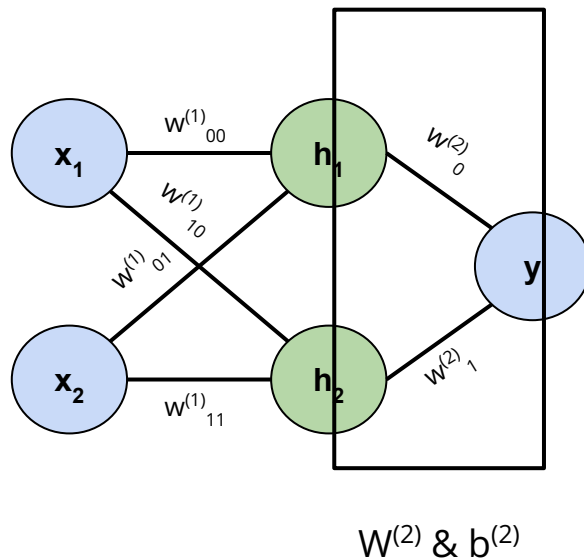
$$y = \sigma(w_0^{(2)} h_1 + w_1^{(2)} h_2 + b_1^{(2)})$$

$$= \sigma(w_0^{(2)}(w_{00}^{(1)} x_1 + w_{01}^{(1)} x_2 + b_1^{(1)}) + w_1^{(2)}(w_{10}^{(1)} x_1 + w_{11}^{(1)} x_2 + b_2^{(1)}) + b_1^{(2)})$$

$$= \sigma(w_1 x_1 + w_2 x_2 + b_2)$$

Neural Networks - BackPropagation

How do weights and biases get updated?

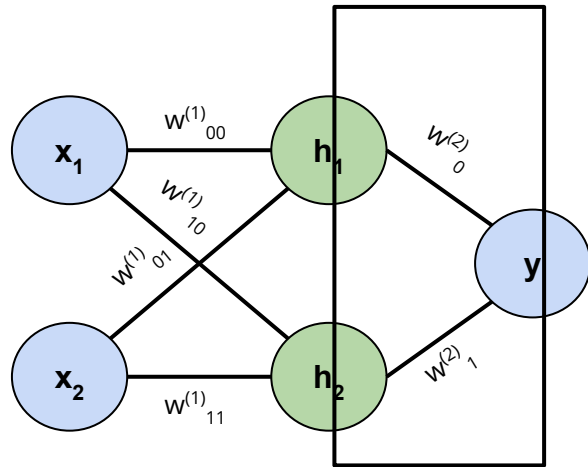


Neural Networks - BackPropagation

This is the same update from logistic regression except relative to the learned features \mathbf{h}

$$\text{Cost}(w, b)$$

$$= -\frac{1}{n} \sum_{i=1}^n [y_i \log(\sigma(-w^T h_i + b)) + (1 - y_i) \log(1 - \sigma(-w^T h_i + b))]$$

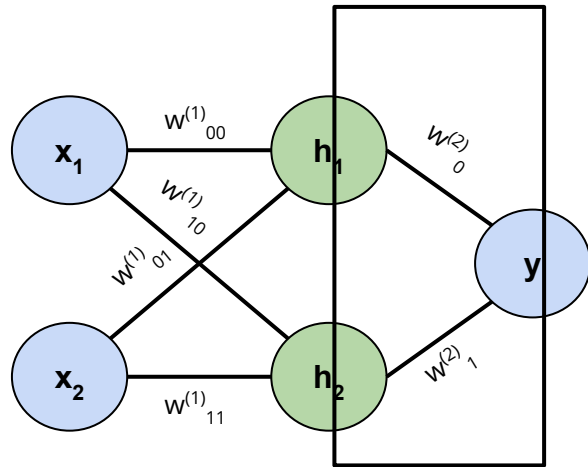


Neural Networks - BackPropagation

$$\nabla \text{Cost}(w, b) = \left[\frac{\partial}{\partial w} \text{Cost}, \frac{\partial}{\partial b} \text{Cost} \right]$$

$$\frac{\partial}{\partial w} \text{Cost} = \frac{1}{n} \sum_{i=1}^n h_i (y_i - \sigma(-w^T h_i + b))$$

$$\frac{\partial}{\partial b} \text{Cost} = \frac{1}{n} \sum_{i=1}^n \sigma(-w^T h_i + b) - y_i$$



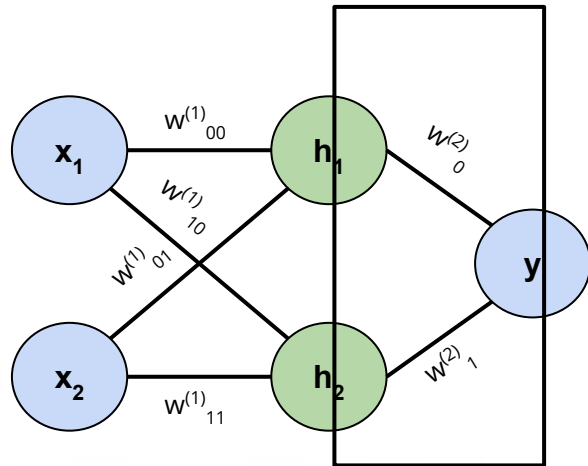
Neural Networks - BackPropagation

Using the chain rule:

$$\frac{\partial C}{\partial W^{(2)}} = \frac{\partial C}{\partial u^{(2)}} \frac{\partial u^{(2)}}{\partial W^{(2)}}$$

where $u^{(2)} = W^{(2)}h + b^{(2)}$

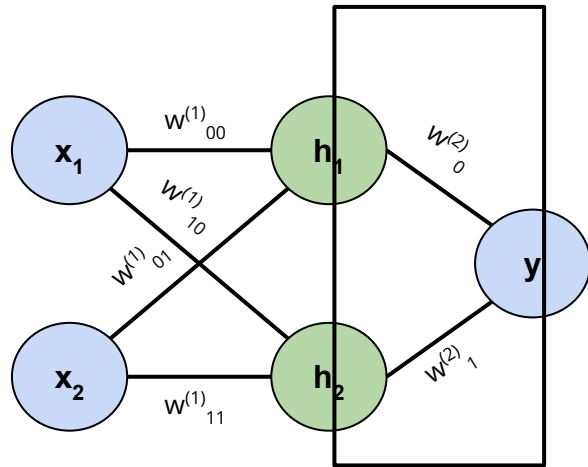
$$= \frac{\partial C}{\partial u^{(2)}} \cdot h = \frac{1}{n} \sum_{i=1}^n h(y_i - \sigma(u^{(2)}))$$



$$h = \sigma(W^{(1)}X + b^{(1)})$$

Neural Networks - BackPropagation

Similarly:

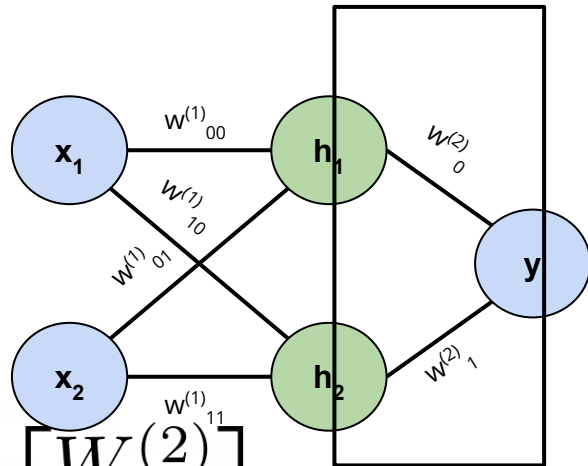


$$\frac{\partial C}{\partial b^{(2)}} = \frac{\partial C}{\partial u^{(2)}} \frac{\partial u^{(2)}}{\partial b^{(2)}} = \frac{1}{n} \sum_{i=1}^n y_i - \sigma(u^{(2)})$$

Neural Networks - BackPropagation

So we can update $W^{(2)}$ and $b^{(2)}$ as follows:

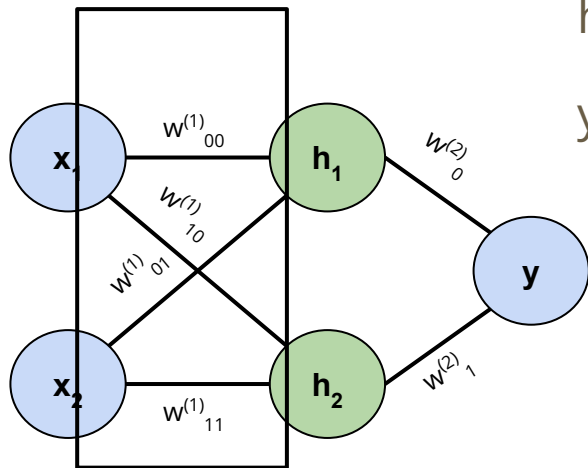
$$\begin{bmatrix} W_{new}^{(2)} \\ b_{new}^{(2)} \end{bmatrix} = -\alpha \begin{bmatrix} \frac{\partial C}{\partial W^{(2)}} \\ \frac{\partial C}{\partial b^{(2)}} \end{bmatrix} + \begin{bmatrix} W^{(2)} \\ b^{(2)} \end{bmatrix}$$



So far this is identical to logistic regression. But how do we update $W^{(1)}$ and $b^{(1)}$

Neural Networks - BackPropagation

How do weights and biases get updated?



$$h_1 = \sigma(w^{(1)}_{00} x_1 + w^{(1)}_{01} x_2 + b^{(1)}_1)$$

$$h_2 = \sigma(w^{(1)}_{10} x_1 + w^{(1)}_{11} x_2 + b^{(1)}_2)$$

$$y = \sigma(w^{(2)}_0 h_1 + w^{(2)}_1 h_2 + b^{(2)}_1)$$

Neural Networks - BackPropagation

$$\text{Cost}(w, b)$$

$$= -\frac{1}{n} \sum_{i=1}^n [y_i \log(\sigma(-w^T h_i + b)) + (1 - y_i) \log(1 - \sigma(-w^T h_i + b))]$$

Neural Networks - BackPropagation

$$\text{Cost}(w, b)$$

$$= -\frac{1}{n} \sum_{i=1}^n [y_i \log(\sigma(-w^T h_i + b)) + (1 - y_i) \log(1 - \sigma(-w^T h_i + b))]$$

Neural Networks - BackPropagation

$$\text{Cost}(w, b)$$

$$= -\frac{1}{n} \sum_{i=1}^n [y_i \log(\sigma(-w^T h_i + b)) + (1 - y_i) \log(1 - \sigma(-w^T h_i + b))]$$


$$h_i = \sigma(w_{i0}^{(1)} x_1 + w_{i1}^{(1)} x_2 + b_i^{(1)})$$

Neural Networks - BackPropagation

Using the chain rule:

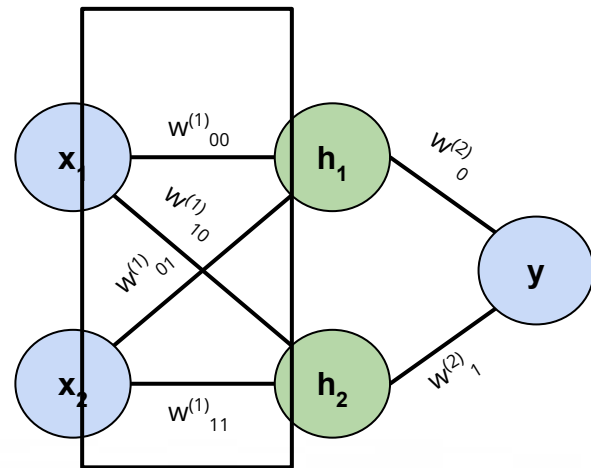
$$\frac{\partial C}{\partial W^{(1)}} = \frac{\partial C}{\partial h} \cdot \frac{\partial h}{\partial W^{(1)}} = \frac{\partial C}{\partial h} \cdot \frac{\partial h}{\partial u^{(1)}} \cdot \frac{\partial u^{(1)}}{\partial W^{(1)}} \quad \text{where } u^{(1)} = w^{(1)}x + b^{(1)}$$

$W^{(1)} \text{ \& } b^{(1)}$

$$= \frac{\partial C}{\partial u^{(2)}} \cdot \frac{\partial u^{(2)}}{\partial h} \cdot \frac{\partial h}{\partial u^{(1)}} \cdot \frac{\partial u^{(1)}}{\partial W^{(1)}} = \frac{\partial C}{\partial u^{(2)}} \cdot W^{(2)} \cdot \sigma'(u^{(1)}) \cdot x$$



Already computed



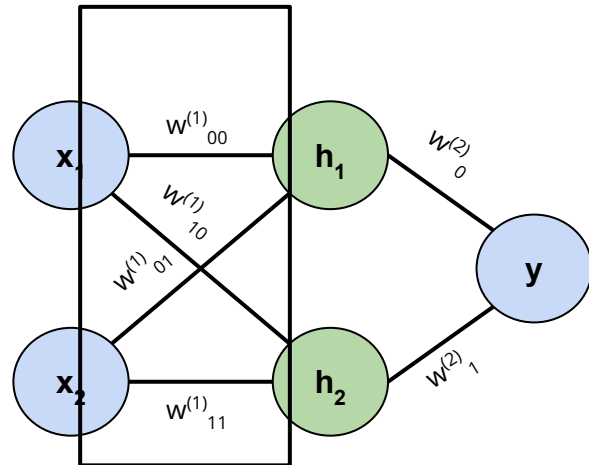
Neural Networks - BackPropagation

Similarly:

$$\frac{\partial C}{\partial b^{(1)}} = \frac{\partial C}{\partial u^{(2)}} \cdot W^{(2)} \cdot \sigma'(u^{(1)})$$



Already computed



Neural Networks - BackPropagation

Backpropagation: update $W^{(1)}$ and $b^{(1)}$ without recomputing values that are computed when getting the gradients of the previously updated layer.

<http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf>

Neural Networks - BackPropagation

Important Note:

$$\frac{\partial C}{\partial W^{(1)}} = \frac{\partial C}{\partial h} \cdot \frac{\partial h}{\partial W^{(1)}} = \frac{\partial C}{\partial h} \cdot \frac{\partial h}{\partial u^{(1)}} \cdot \frac{\partial u^{(1)}}{\partial W^{(1)}} \quad \text{where} \quad u^{(1)} = w^{(1)}x + b^{(1)}$$

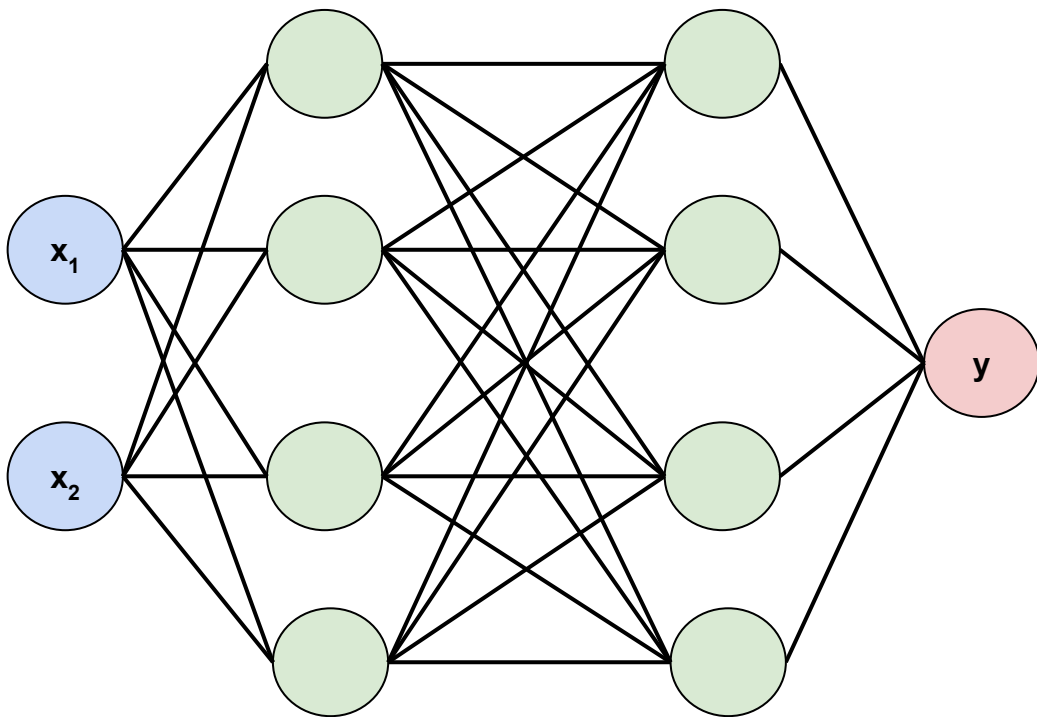
$$= \frac{\partial C}{\partial u^{(2)}} \cdot \frac{\partial u^{(2)}}{\partial h} \cdot \frac{\partial h}{\partial u^{(1)}} \cdot \frac{\partial u^{(1)}}{\partial W^{(1)}} = \frac{\partial C}{\partial u^{(2)}} \cdot W^{(2)} \cdot \sigma'(u^{(1)}) \cdot x$$



Depends on both data and weights
Initializing all weights to zero then is not a good idea

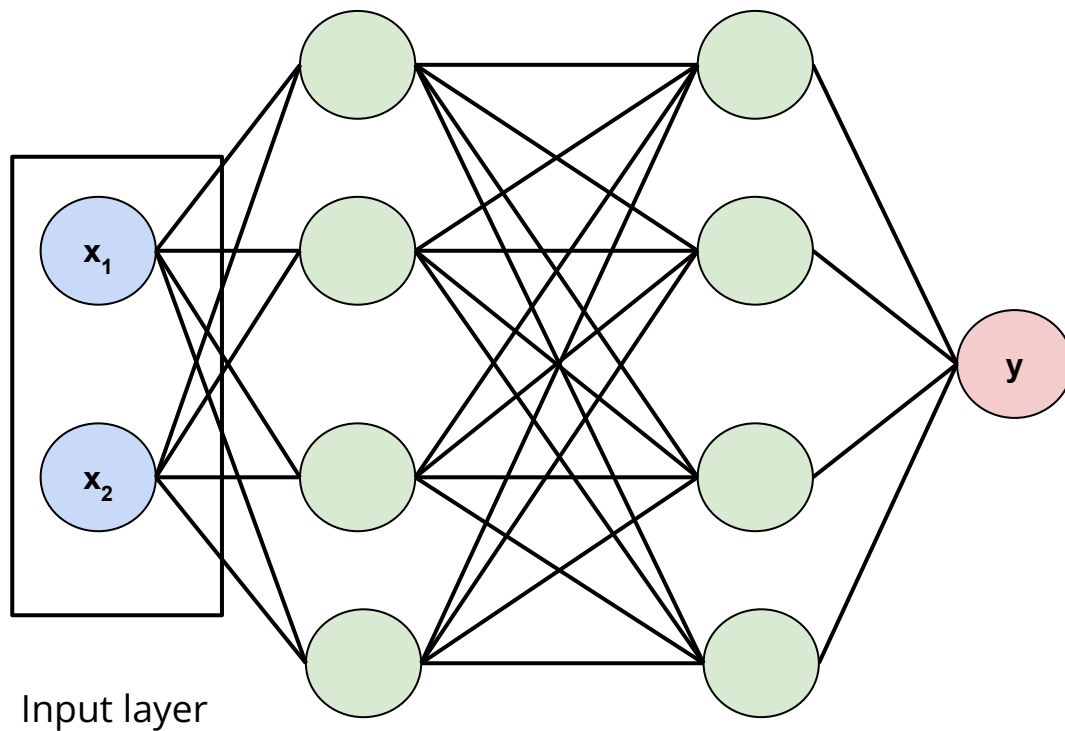
Feedforward Neural Networks

In general:



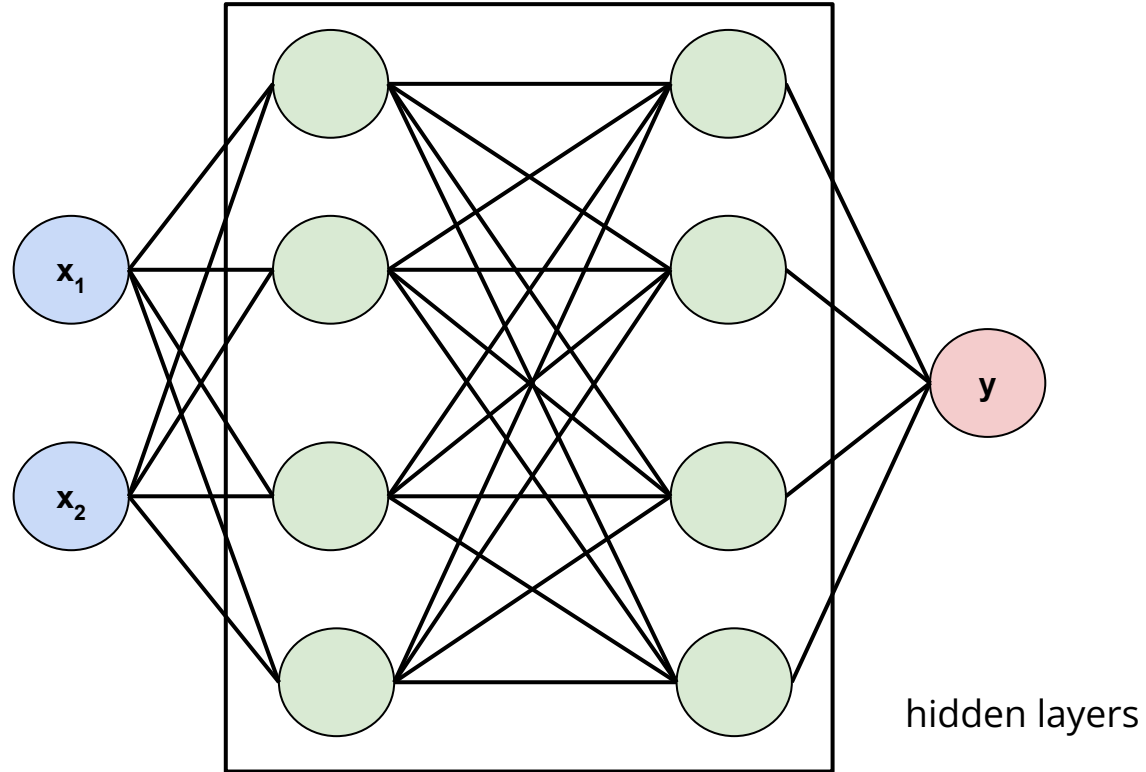
Feedforward Neural Networks

In general:



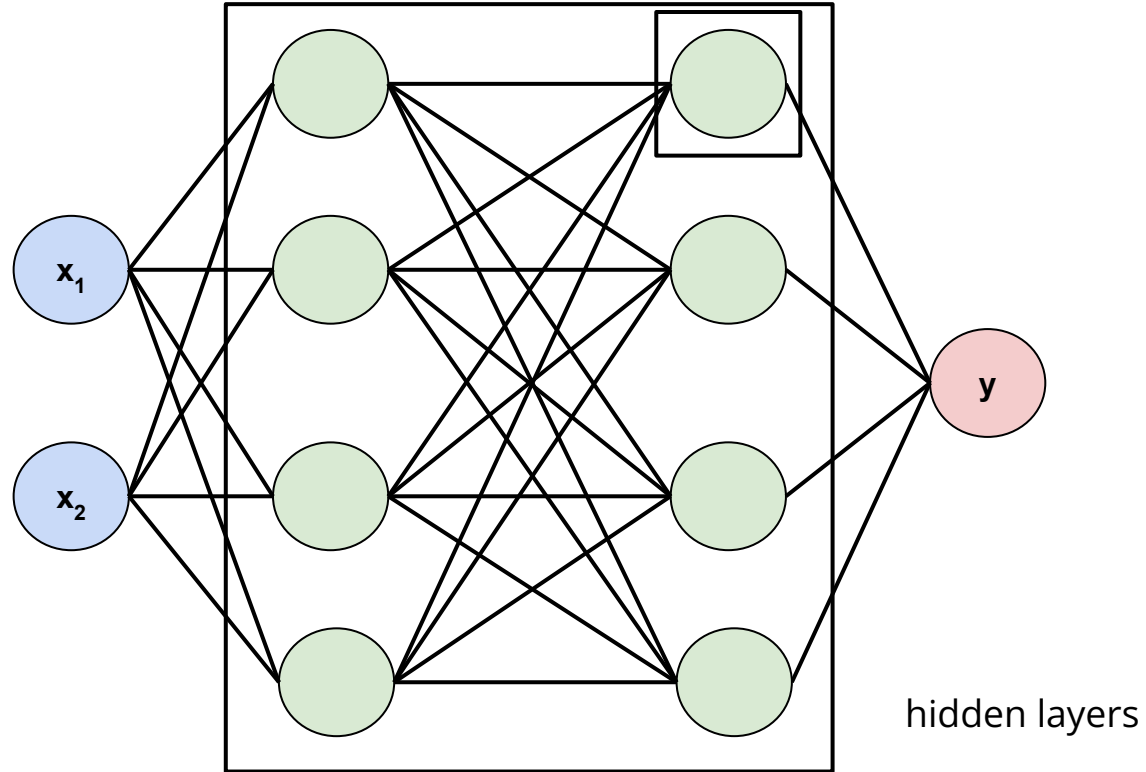
Feedforward Neural Networks

In general:



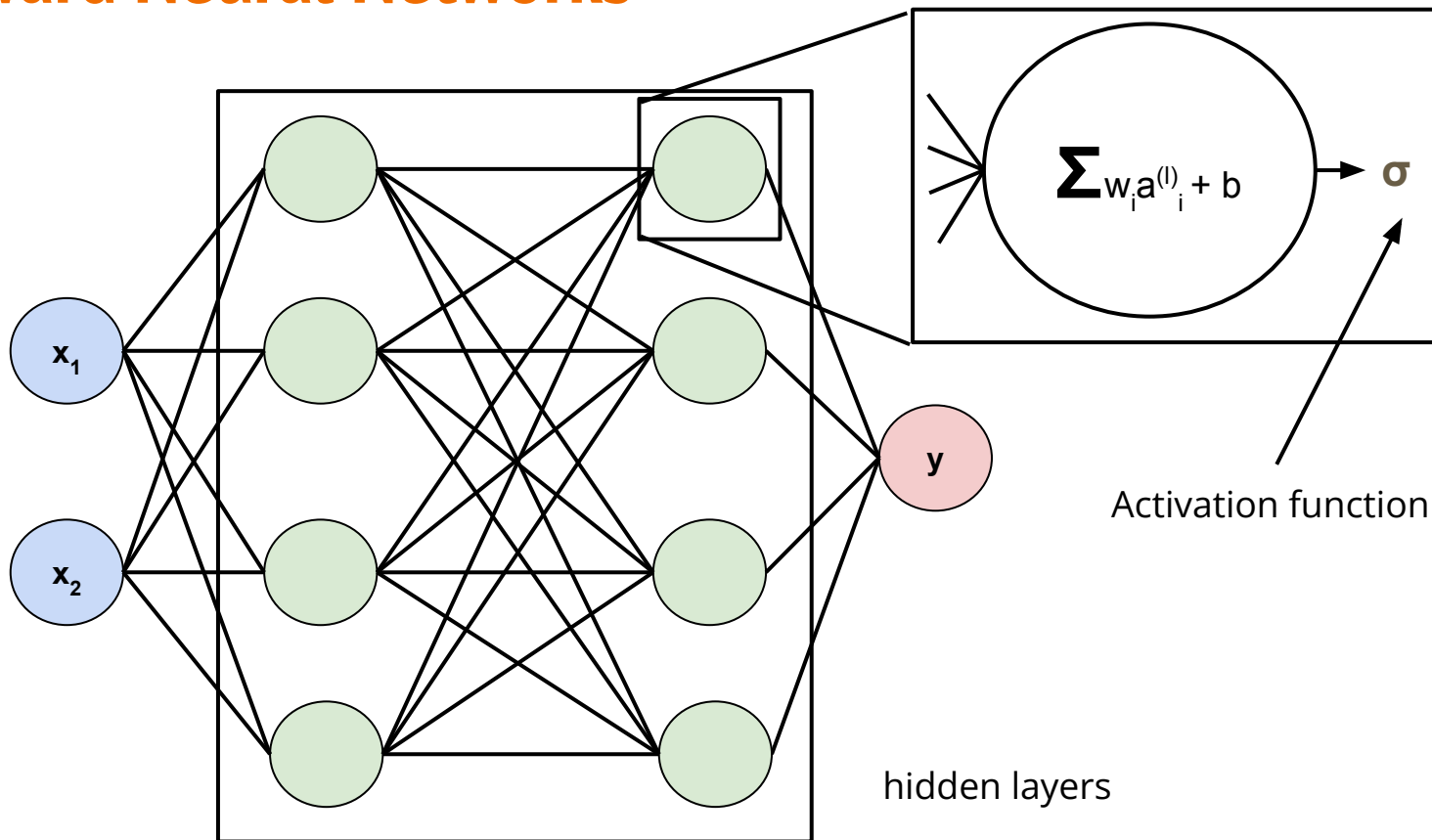
Feedforward Neural Networks

In general:



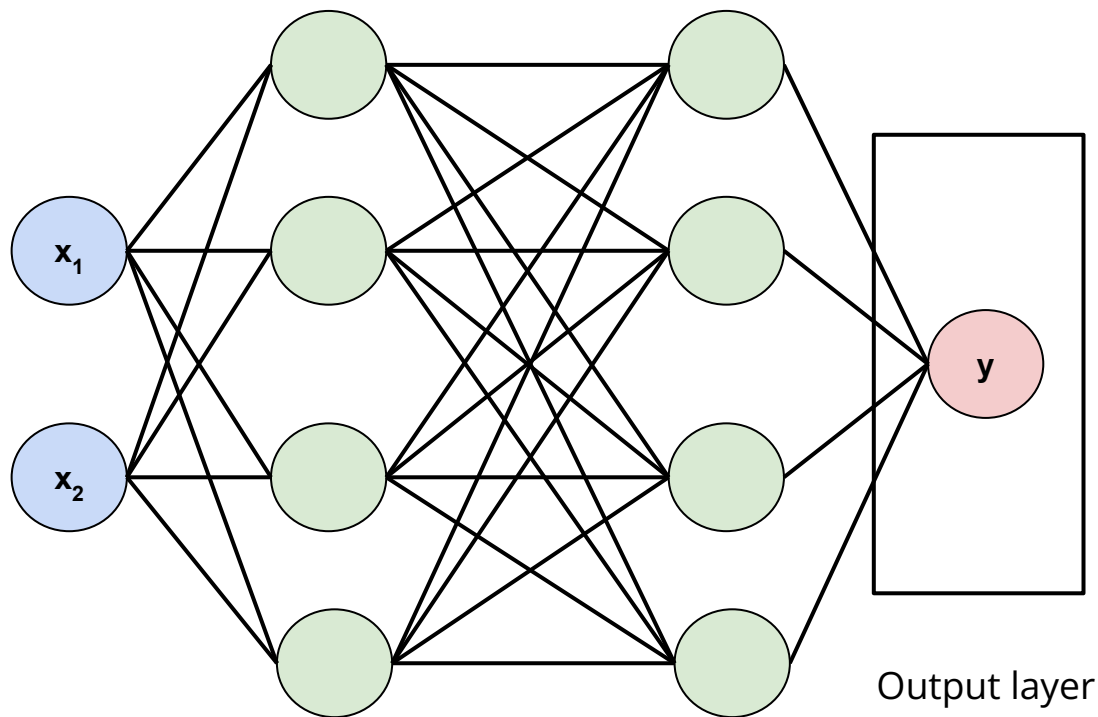
Feedforward Neural Networks

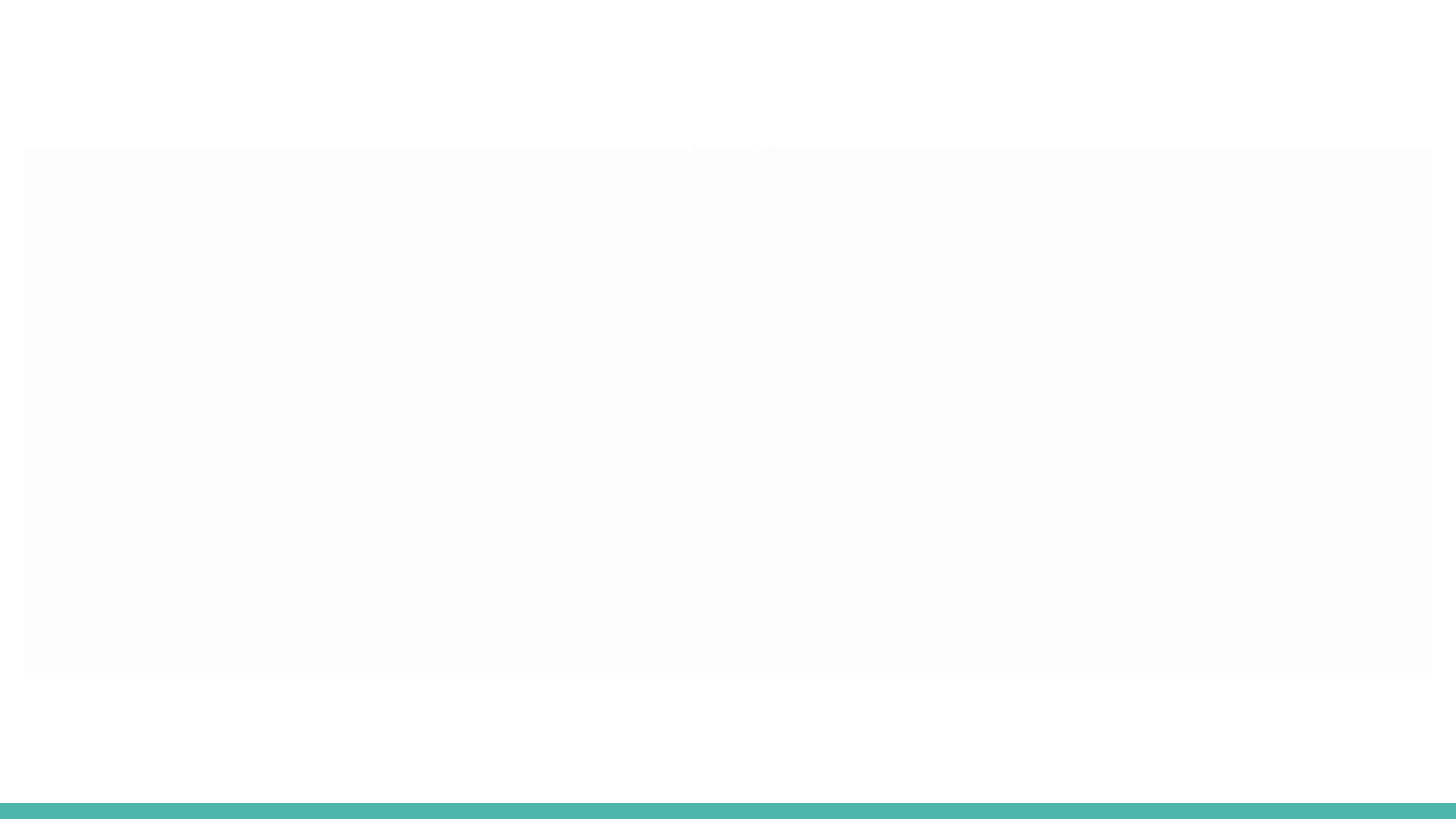
In general:



Feedforward Neural Networks

In general:





Feedforward Neural Networks

The hope:

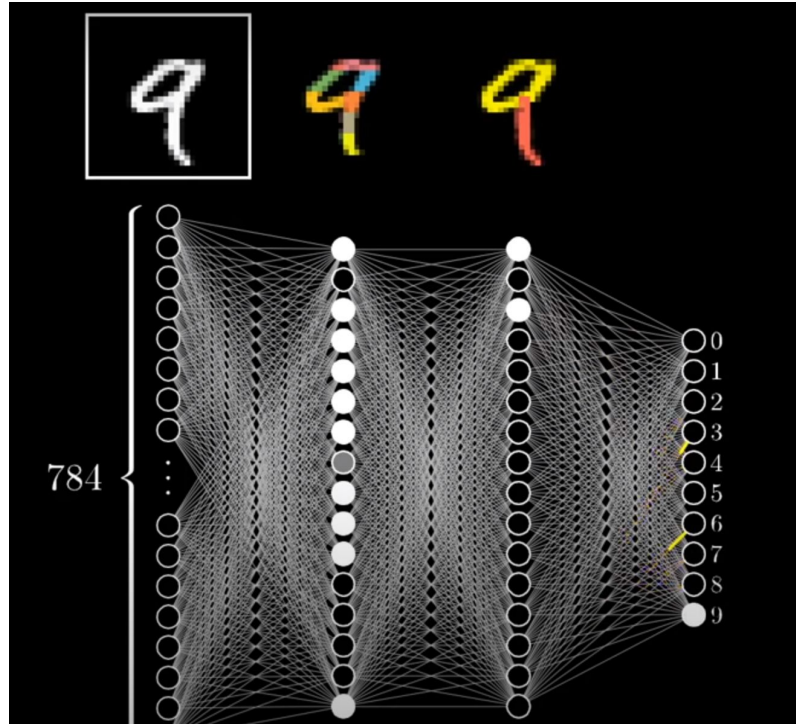
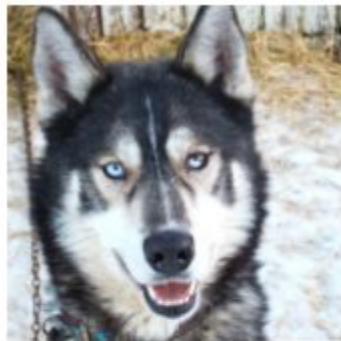


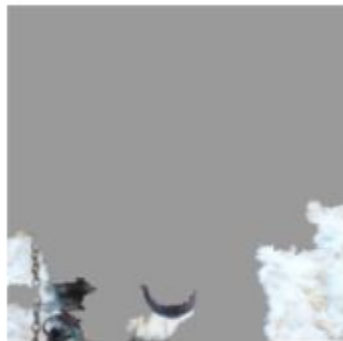
Image from 3b1b

Feedforward Neural Networks

The reality:



(a) Husky classified as wolf



(b) Explanation

Figure 11: Raw data and explanation of a bad model's prediction in the "Husky vs Wolf" task.

	Before	After
Trusted the bad model	10 out of 27	3 out of 27
Snow as a potential feature	12 out of 27	25 out of 27

Table 2: "Husky vs Wolf" experiment results.

Image from "Why Should I Trust You?": Explaining the Predictions of Any Classifier (2016) Marco Tulio Ribeiro, Sameer Singh, Carlos Guestrin

Feedforward Neural Networks

The scary reality:

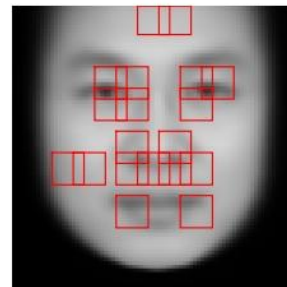


(a) Three samples in criminal ID photo set S_c .

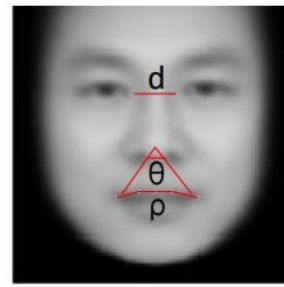


(b) Three samples in non-criminal ID photo set S_n .

from "Automated Inference on
Criminality using Face Images",
Xiaolin Wu, Xi Zhang



(a)



(b)

Figure 8. (a) FGM results; (b) Three discriminative features ρ , d and θ .

According to this model, if you don't smile, you're a criminal

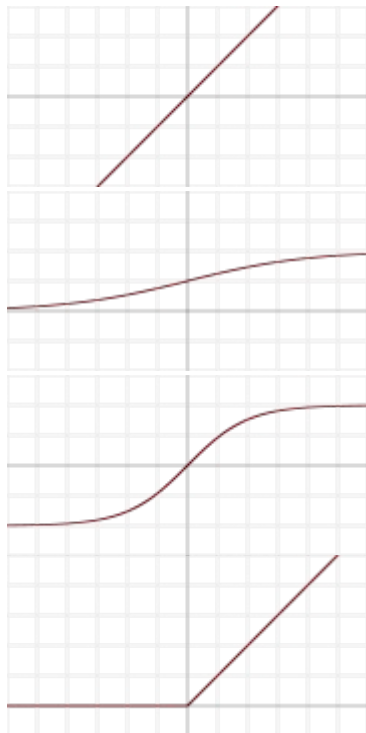
Neural Networks

Can do both **Classification** and **Regression**

Neural Networks - Tuning Parameters

1. Step size α
2. Number of BackPropagation iterations
3. Batch Size
4. Number of hidden layers
5. Size of each hidden layer
6. Activation function used in each layer
7. Cost function
8. Regularization (to avoid overfitting)

Activation Functions



Identity -> x

Sigmoid -> $\sigma(x)$

Tanh -> $\tanh(x)$

ReLU -> $\max(0, x)$

Note: can use any function you want in order to introduce non-linearity. These are just the popular ones that have been shown to work in practice.

Tuning the activation function is equivalent to feature engineering.

Demo

Universal Approximation Theorem

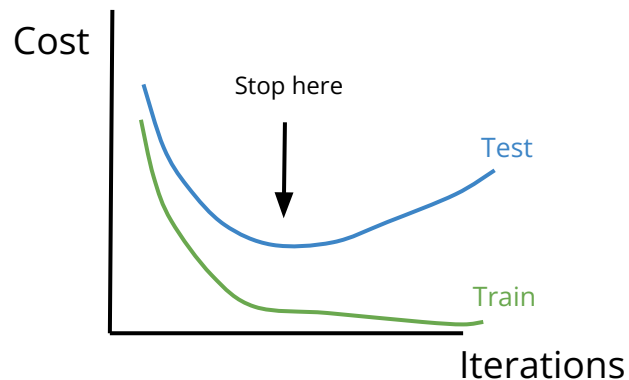
Neural Networks - Challenges

1. High risk of overfitting as you're optimizing on the training set.
2. As the dimensionality of the input increases:
 - a. So does the number of weights
 - b. The gradients typically get smaller: Vanishing gradient problem
3. Doesn't do well for computer vision where the object of detection can be anywhere in the image
4. Doesn't handle sequences of inputs (i.e. providing context for data)

Neural Networks - Regularization

Two main ways:

1. Early termination of weight / bias updates



2. Dropout - kill neurons (by setting them to 0) randomly

Neural Networks

First: Normalize your data

<https://medium.com/mlearning-ai/tuning-neural-networks-part-i-normalize-your-data-6821a28b2cd8>

Neural Networks - Initialization Gotchas

Divide and conquer

<https://medium.com/mlearning-ai/tuning-neural-networks-part-ii-considerations-for-initialization-4f82e525da69>

Zero initialization:

If $\text{act}(0) \neq 0$ then weights all move together (same as constant init)

If $\text{act}(0) = 0$ then no updates can be made

Neural Networks - Activation Functions

<https://medium.com/@gallettilance/tuning-neural-networks-part-iii-43dfd0c8600f>