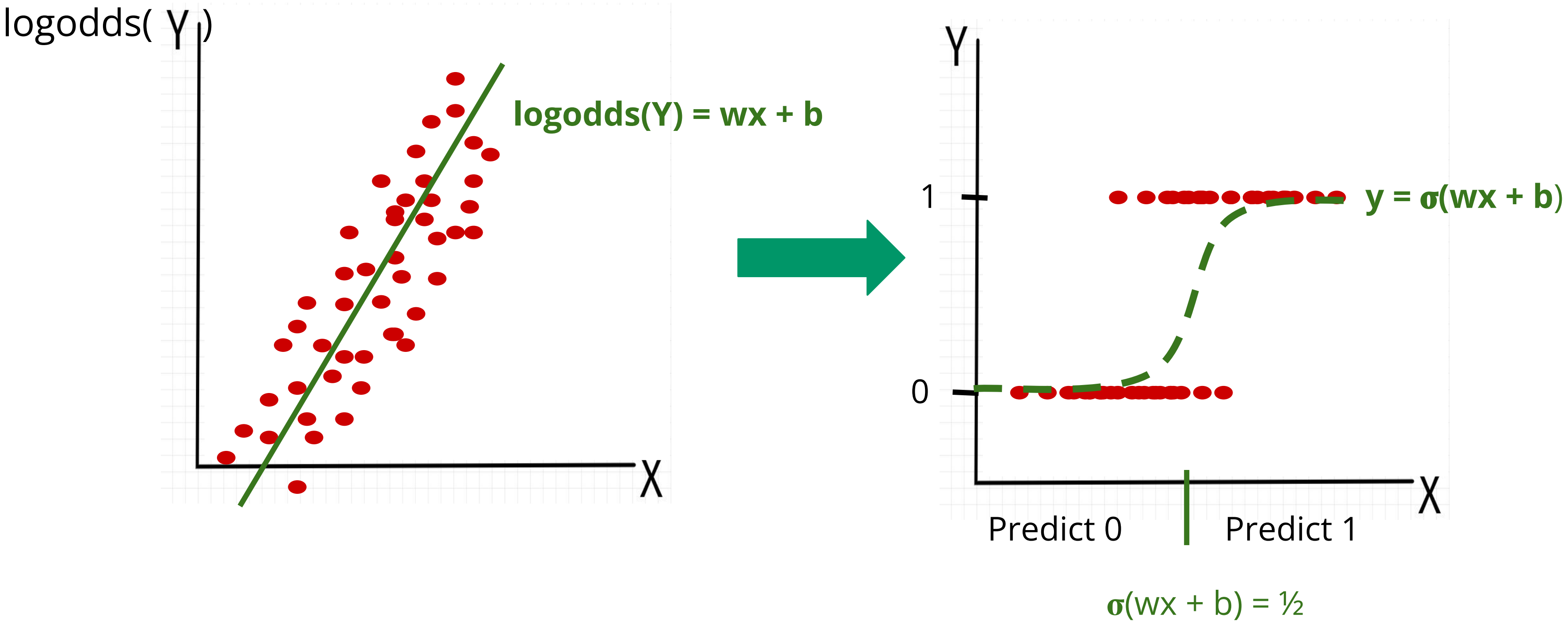
Neural Networks

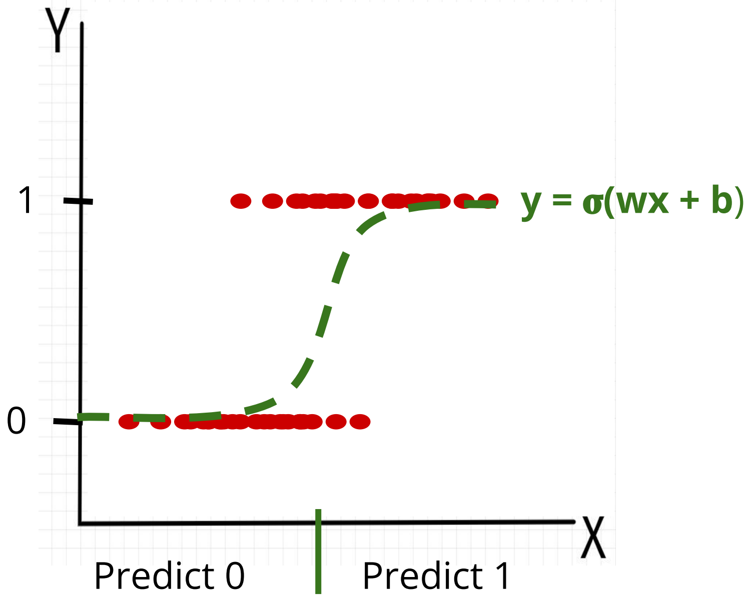
Boston University CS 506 - Lance Galletti

DECISION RULE:
IF P(Y=1|X) > ½ THEN 1 ELSE 0

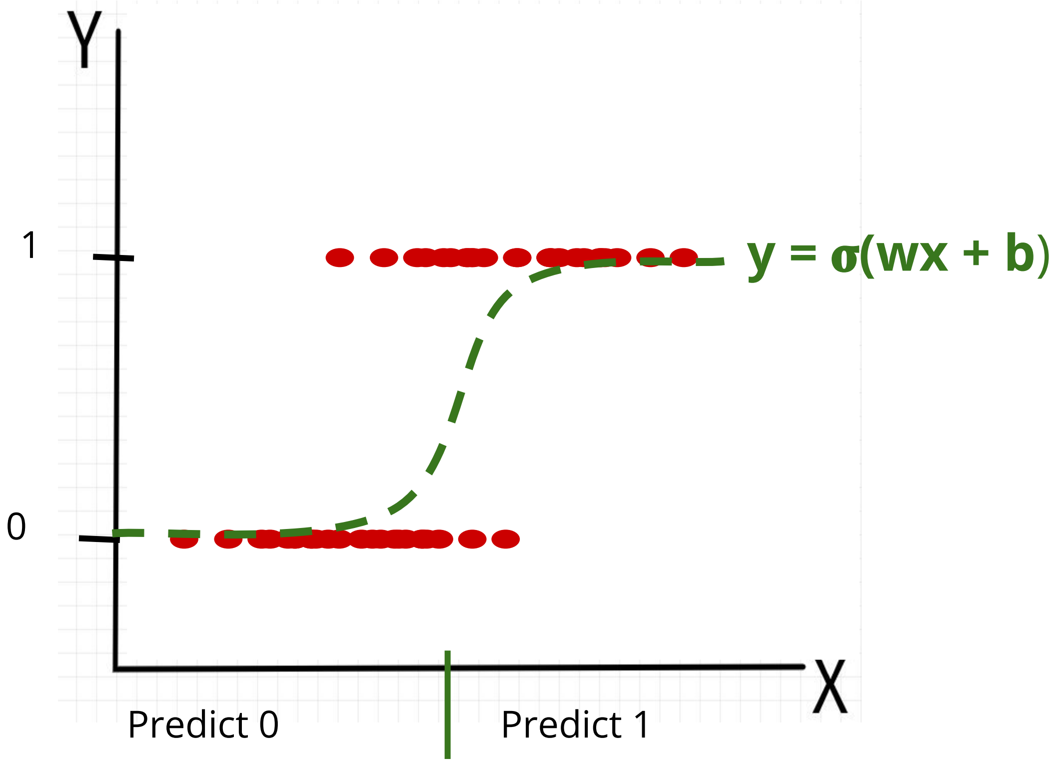


DECISION RULE:

IF P(Y=1|X) > ½ THEN 1 ELSE 0



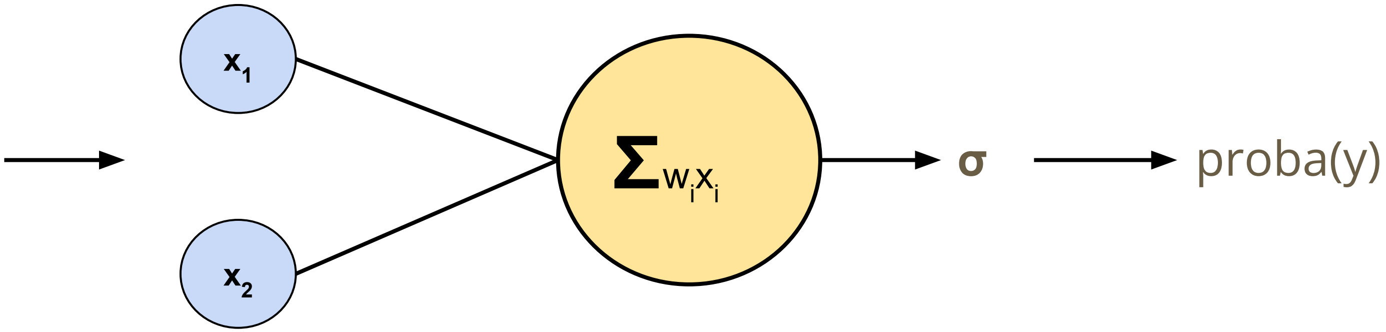
σ(wx + b) = ½



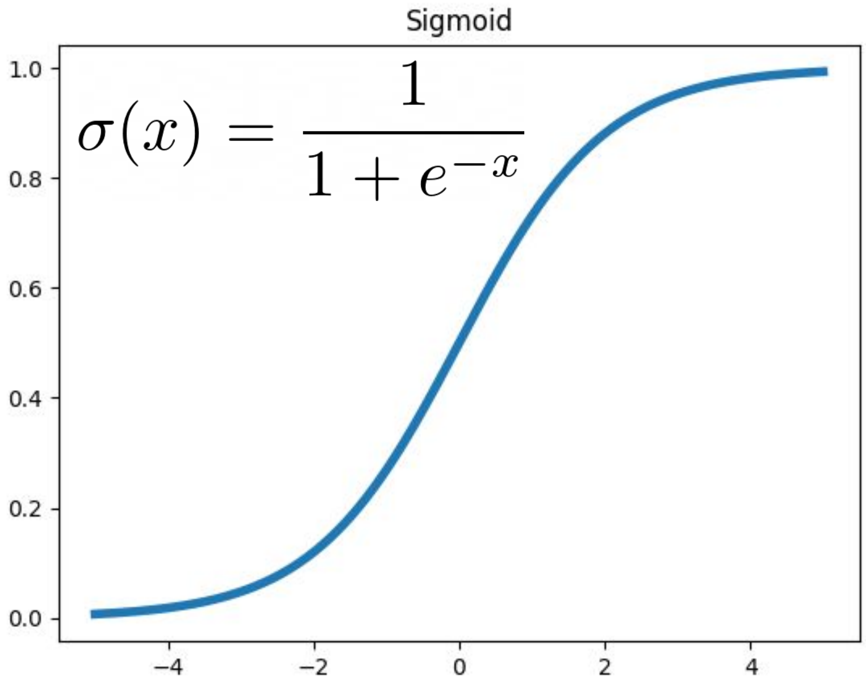
$\sigma(wx + b) = ½$

Logistic Regression Revisited

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



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

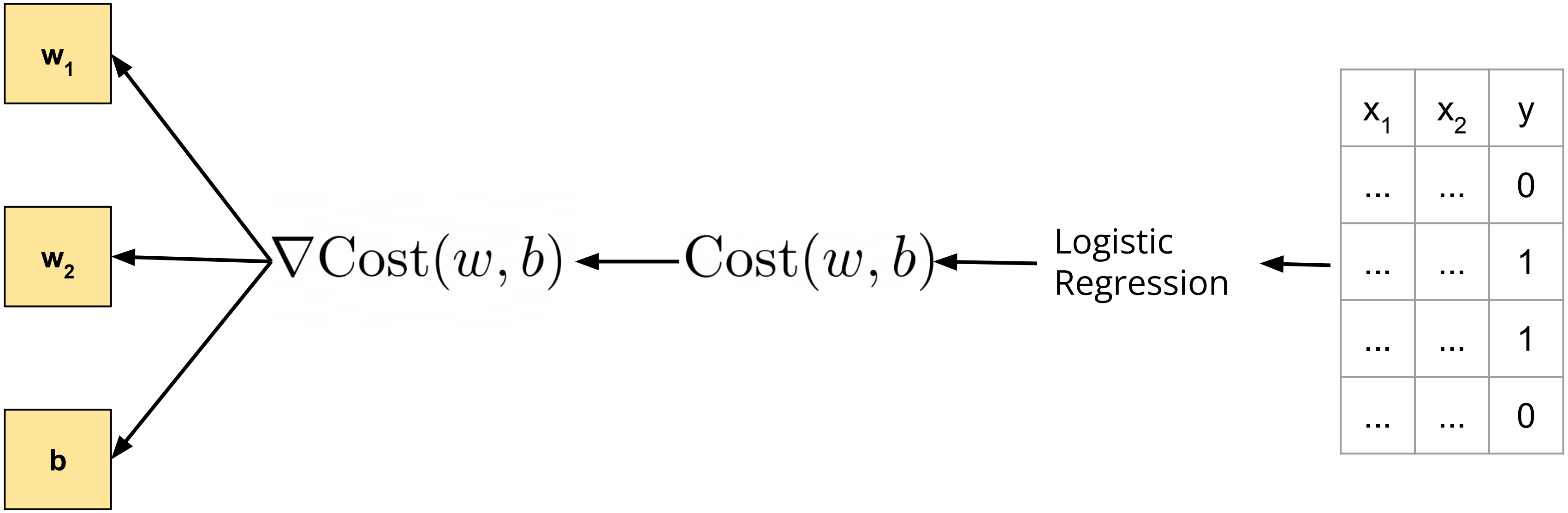

Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$
1.0
0.8
0.6
0.4
0.2
0.0
−4
−2
0
2
4

$$\max \prod_{i=1}^{n} P(y_i|x_i) = \prod_{i} (logit^{-1}(w^T x_i + b))^{y_i} (1 - logit^{-1}(w^T x_i + b))^{1-y_i}$$

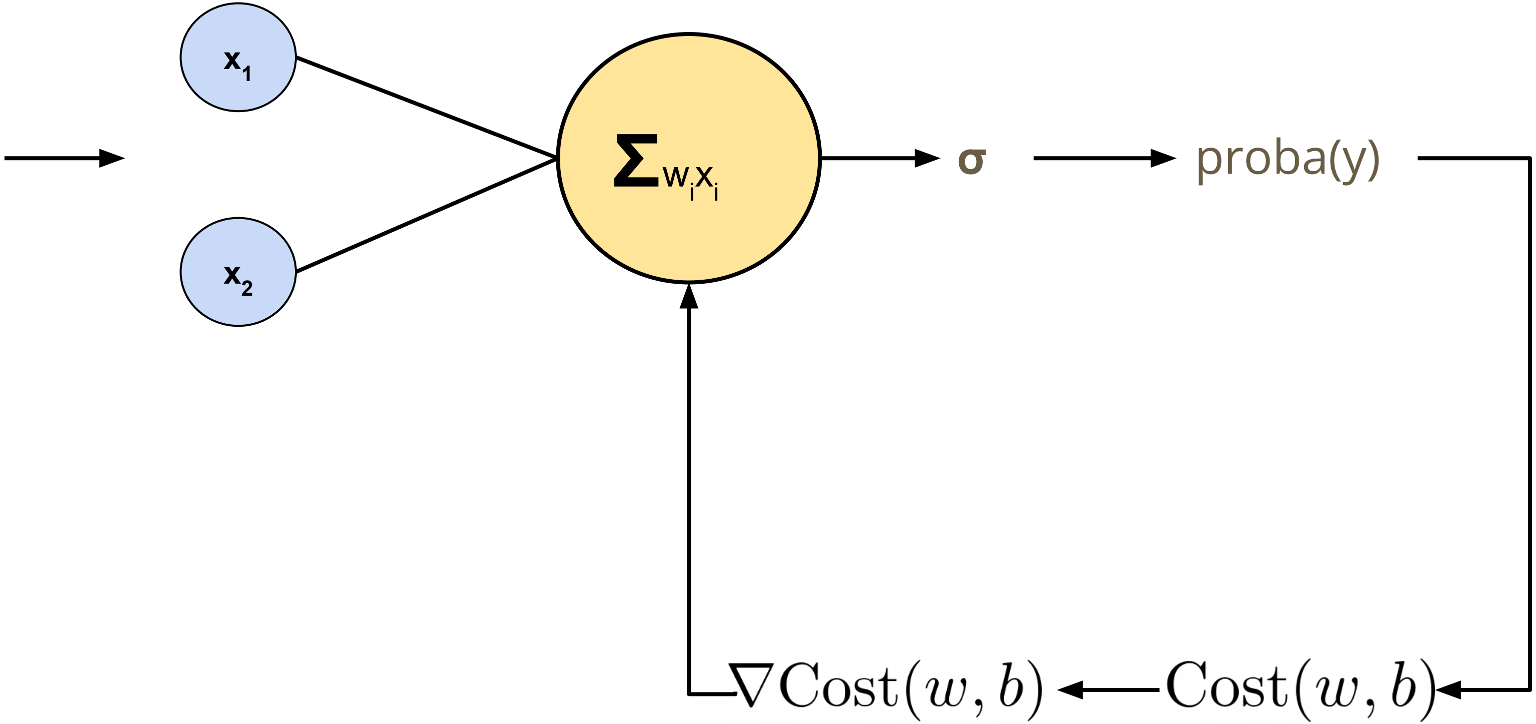
$$= \min -\frac{1}{n} \sum_{i=1}^{n} [yi \log(\sigma(-w^T x_i + b)) + (1 - y_i) \log(1 - \sigma(-w^T x_i + b))]$$

$$= \min \mathrm{Cost}(w, b)$$

Gradient Descent



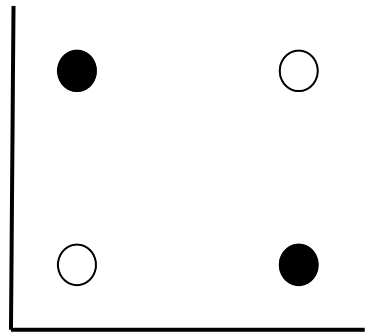
x_1	x_2	y
...	...	0
...	...	1
...	...	1
...	...	0



Logistic Regression Revisited

XOR function

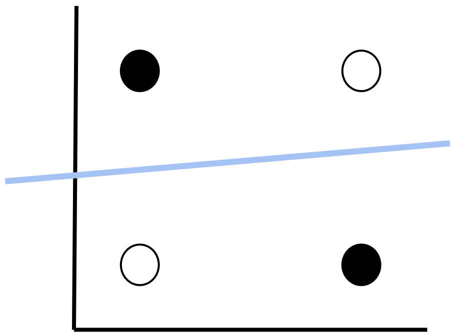
x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0



Logistic Regression Revisited

XOR function

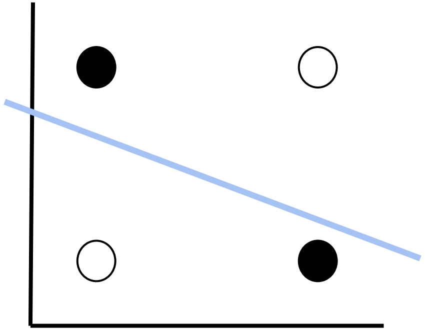
x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0



Logistic Regression Revisited

XOR function

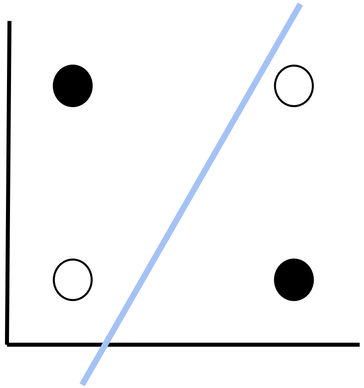
x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0



Logistic Regression Revisited

XOR function

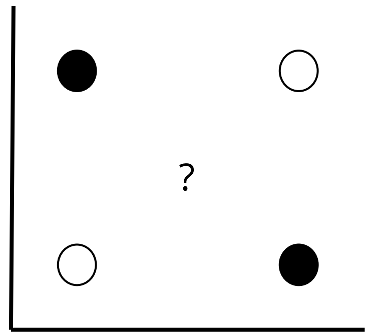
x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0



Logistic Regression Revisited

XOR function

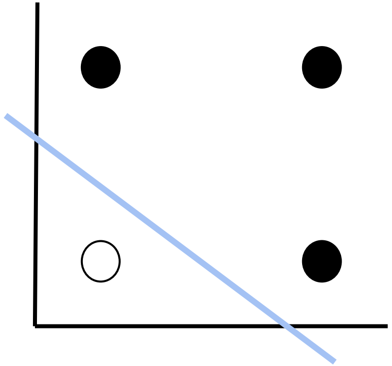
x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0



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

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

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

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

Logistic Regression Revisited

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

$$= (x_1 = 0 \land x_2 = 1) \lor (x_1 = 1 \land x_2 = 0)$$

$$= h_1 \lor h_2$$

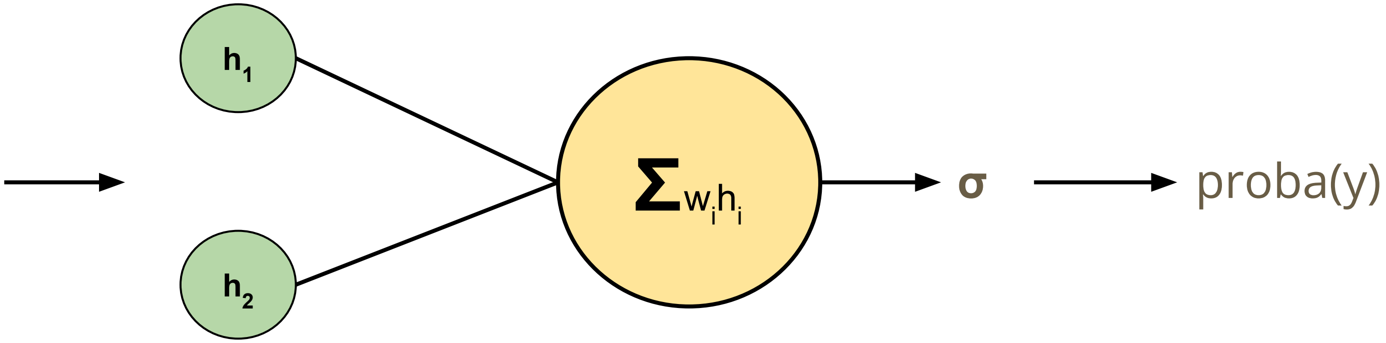
h_1 = **AND**($x_1 = 0$, $x_2 = 1$)

h_2 = **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

h_1	h_2	y
0	0	0
0	1	1
1	0	1
0	0	0

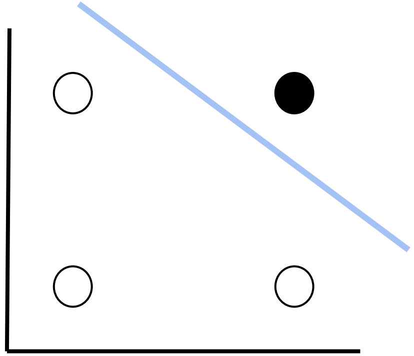


where $$\sigma(x) = \frac{1}{1+e^{-x}}$$

Logistic Regression Revisited

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

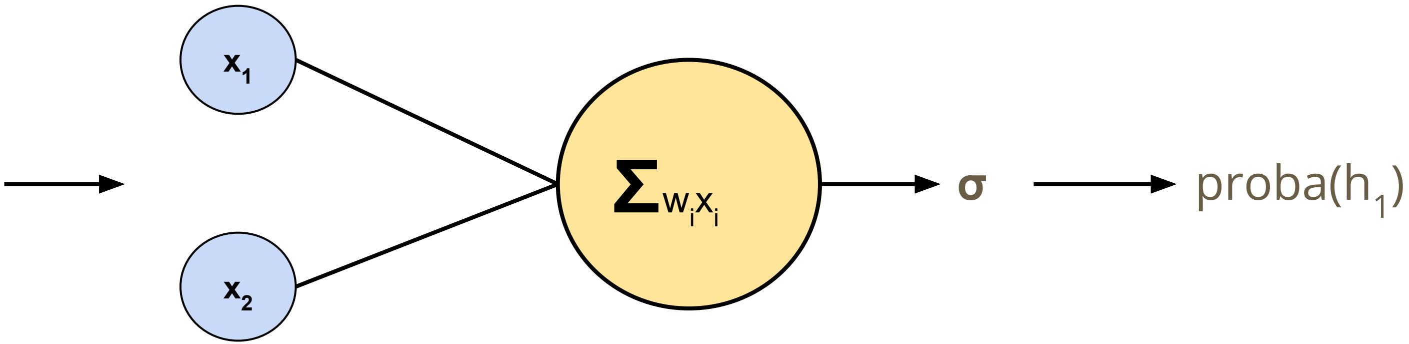
x_1	x_2	y
0	0	0
1	0	0
0	1	0
1	1	1



Logistic Regression Revisited

Since we can learn h_1 and h_2 automatically through logistic regression

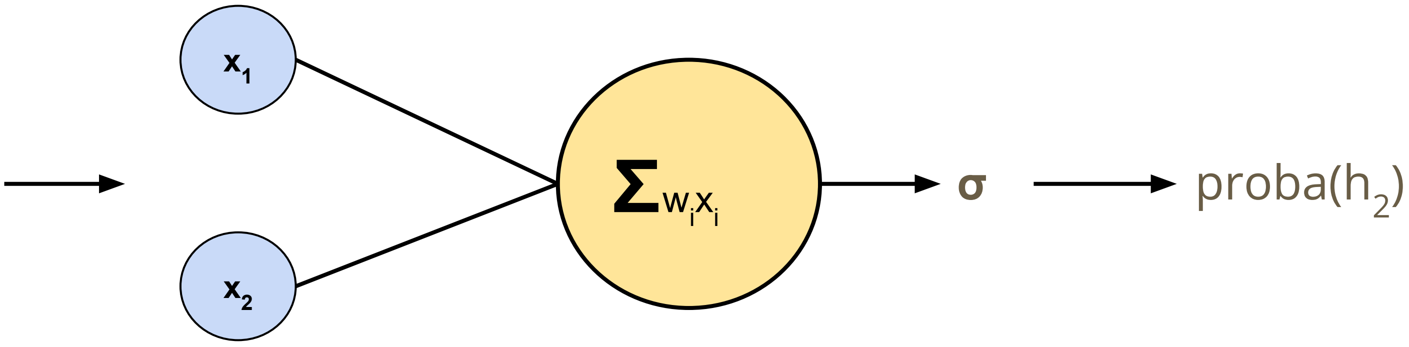
x_1	x_2	h_1
0	0	0
0	1	0
1	0	1
1	1	0



Logistic Regression Revisited

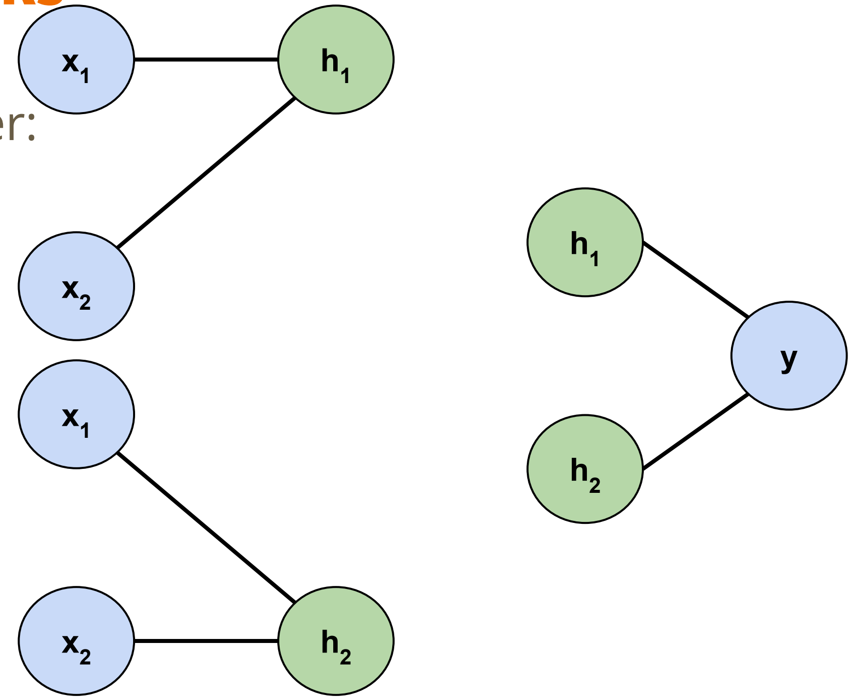
Since we can learn h_1 and h_2 automatically through logistic regression

x_1	x_2	h_2
0	0	0
0	1	1
1	0	0
1	1	0



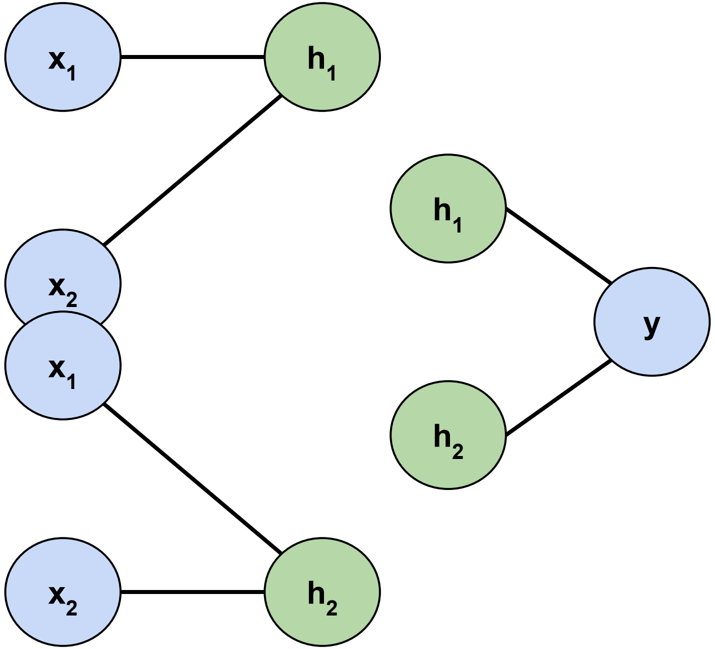
Neural Networks

Putting it all together:



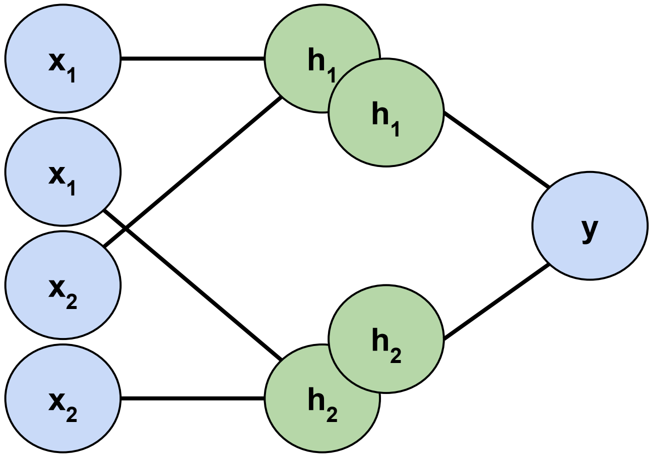
Neural Networks

Putting it all together:



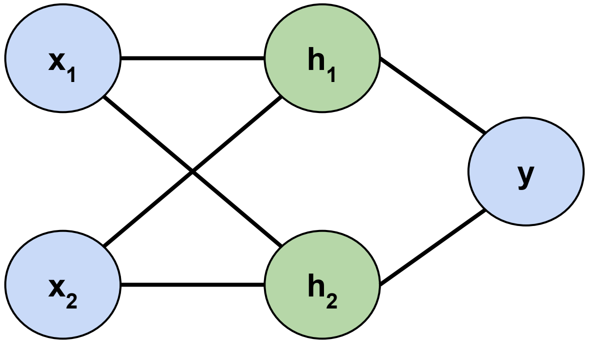
Neural Networks

Putting it all together:

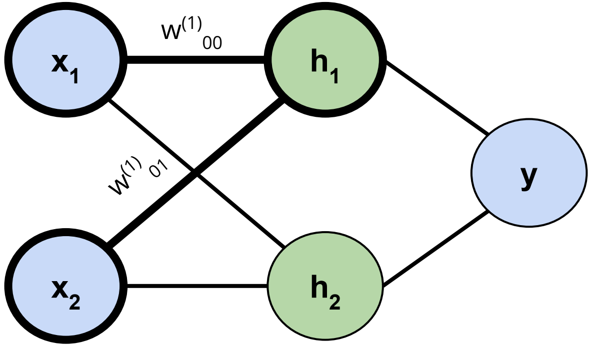


Neural Networks

Putting it all together:

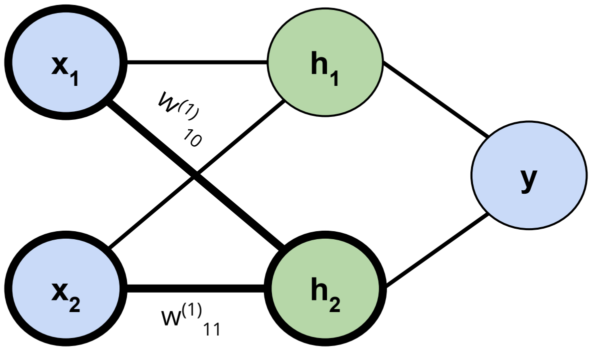


Neural Networks



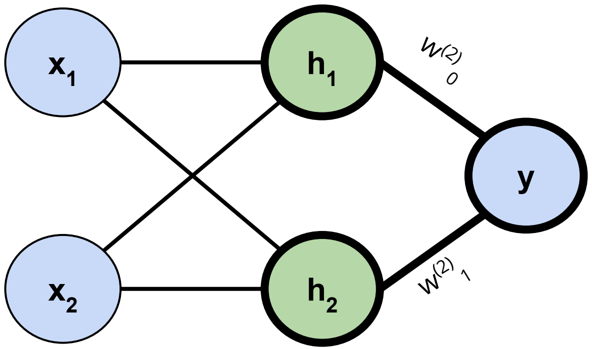
$$h_1 = \boldsymbol{\sigma}(\ w^{(1)}_{00}\ x_1 + w^{(1)}_{01}\ x_2 + b^{(1)}_1)$$

Neural Networks



$h_2 = \boldsymbol{\sigma}(w^{(1)}_{10}\, x_1 + w^{(1)}_{11}\, x_2 + b^{(1)}_{2})$

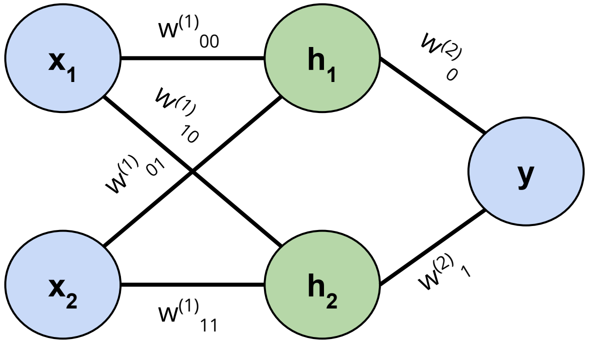
Neural Networks



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

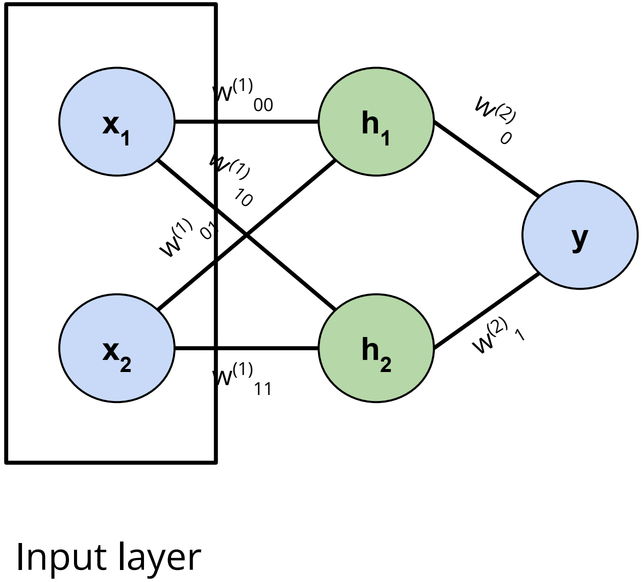
Neural Networks

Putting it all together:



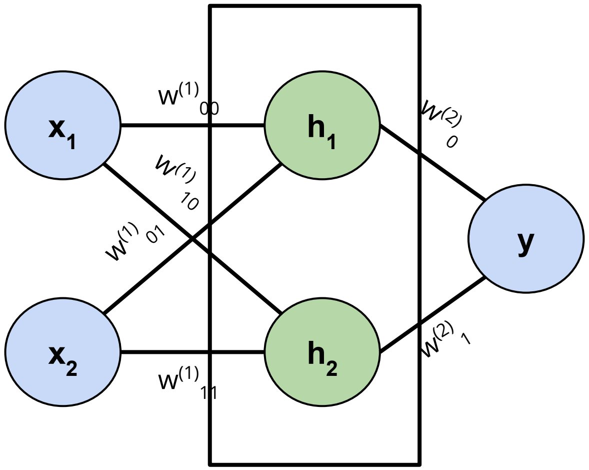
Neural Networks

Putting it all together:



Neural Networks

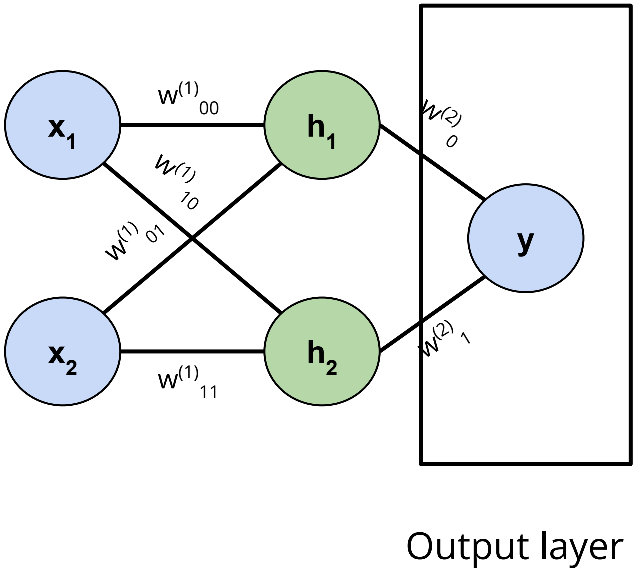
Putting it all together:



Hidden layer

Neural Networks

Putting it all together:



Neural Networks

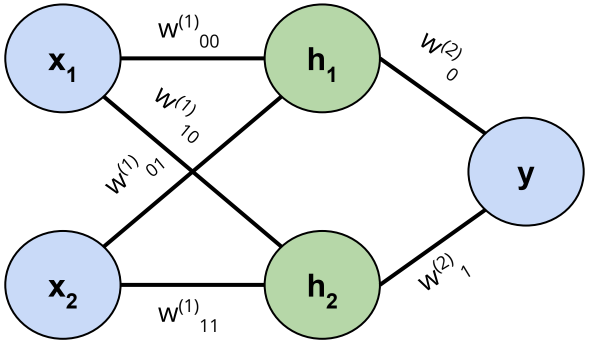
It’s all about learning features (created in the hidden layer(s)) automatically

Neural Networks

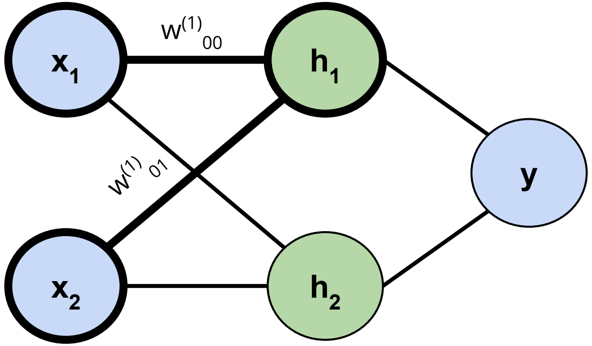
We need to define:

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

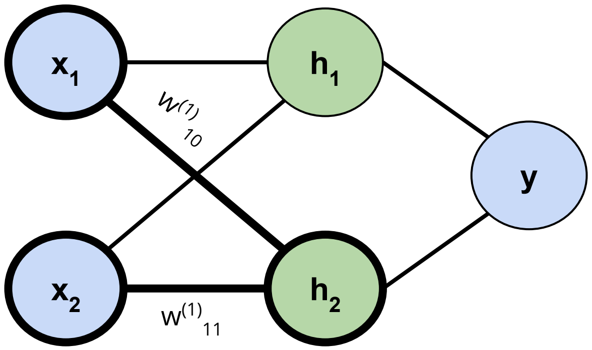


Neural Networks - Forward Propagation



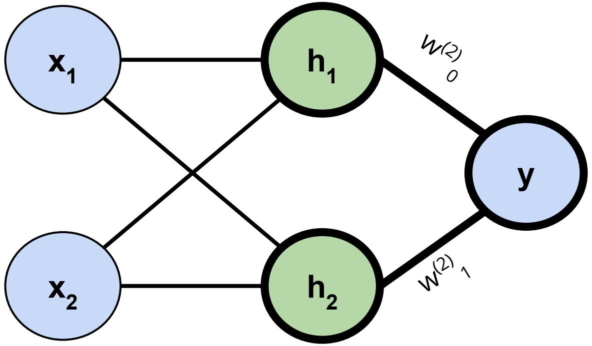
$h_1 = \boldsymbol{\sigma}(w^{(1)}_{00}\, x_1 + w^{(1)}_{01}\, x_2 + b^{(1)}_1)$

Neural Networks - Forward Propagation



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

Neural Networks - Forward Propagation



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

Neural Networks - Forward Propagation

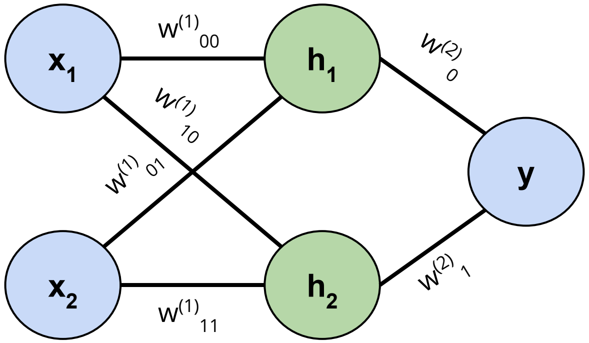
Using matrix notation:

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

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

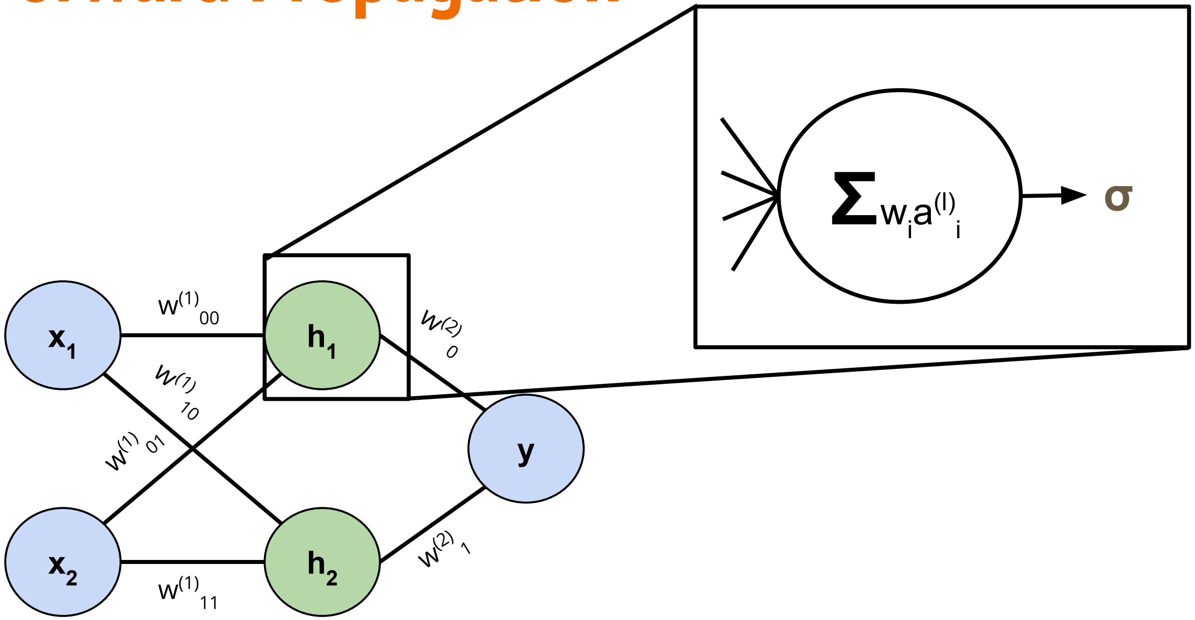
Neural Networks - Forward Propagation

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



Neural Networks - Forward Propagation

Q: what happens if we don't have **σ** in the hidden layer here?



Neural Networks - Forward Propagation

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

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

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

Then

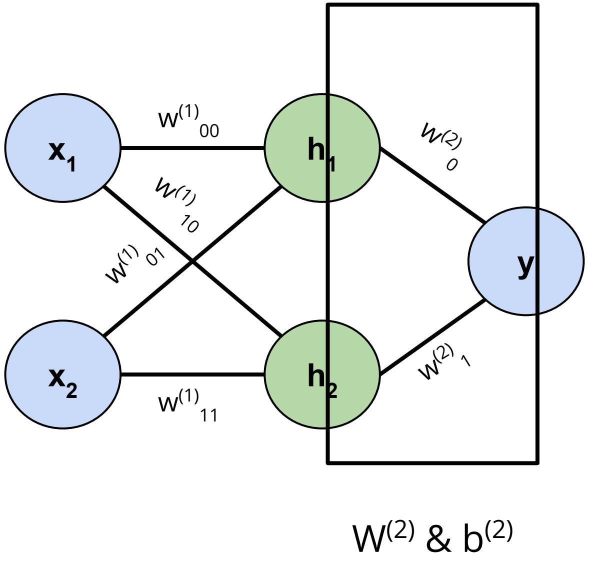
$y = \boldsymbol{\sigma}(\, w^{(2)}_0\, h_1 + w^{(2)}_1\, h_2 + b^{(2)}_1)$

$\quad = \boldsymbol{\sigma}(\, w^{(2)}_0(w^{(1)}_{00}\, x_1 + w^{(1)}_{01}\, x_2 + b^{(1)}_1) + w^{(2)}_1(w^{(1)}_{10}\, x_1 + w^{(1)}_{11}\, x_2 + b^{(1)}_2) + b^{(2)}_1)$

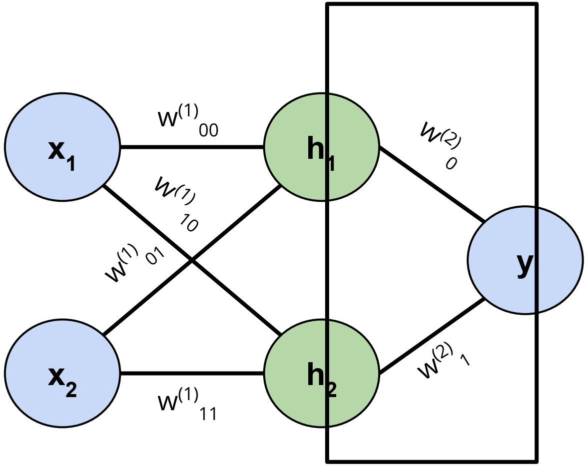
$\quad = \boldsymbol{\sigma}(\, w_1\, x_1 + w_2\, x_2 + b_2)$

Neural Networks - BackPropagation

How do weights and biases get updated?



Neural Networks - BackPropagation

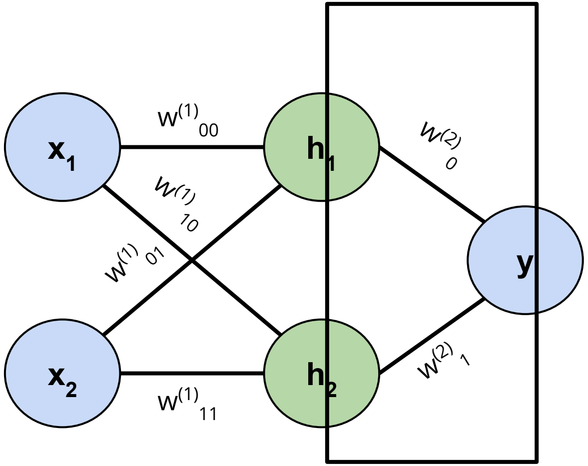


This is the same update from logistic regression except relative to the learned features **h**

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

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

Neural Networks - BackPropagation

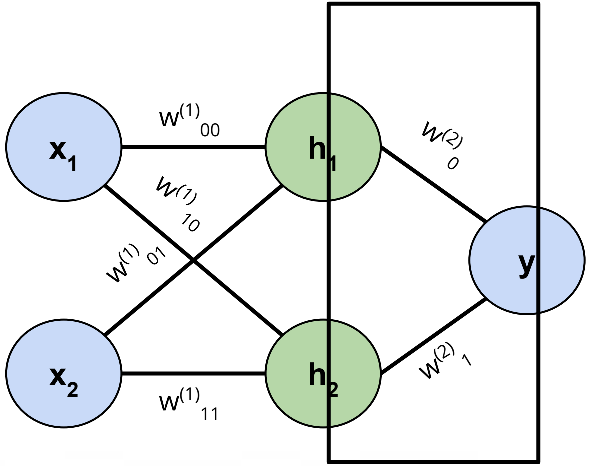


$$\nabla \mathrm{Cost}(w, b) = \left[\frac{\partial}{\partial w} \mathrm{Cost}, \frac{\partial}{\partial b} \mathrm{Cost} \right]$$

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

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

Neural Networks - BackPropagation



Using the chain rule:

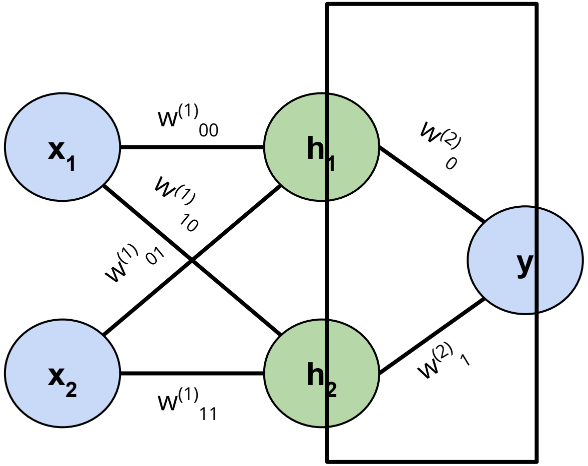
$$\frac{\partial C}{\partial W^{(2)}} = \frac{\partial C}{\partial u^{(2)}} \frac{\partial u^{(2)}}{\partial W^{(2)}} \quad \text{where} \quad u^{(2)} = W^{(2)} h + b^{(2)}$$

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

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

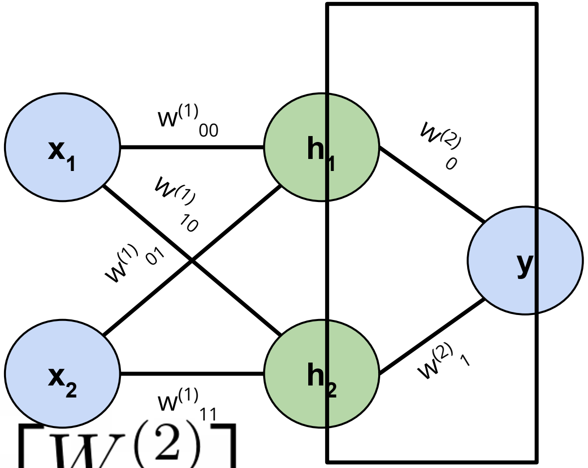
Neural Networks - BackPropagation

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 $b^{(1)}$

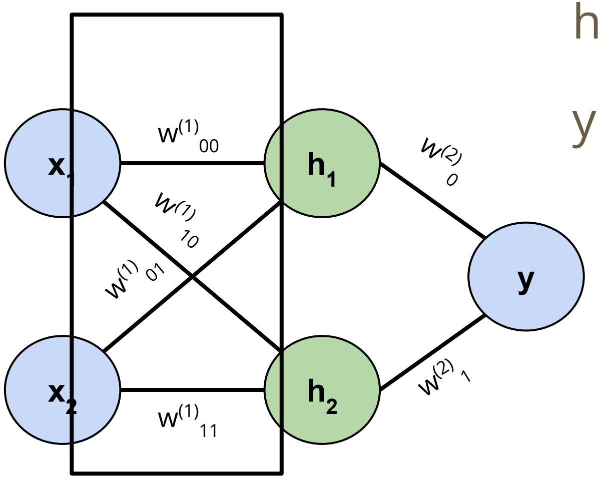
Neural Networks - BackPropagation

How do weights and biases get updated?

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

$$h_2 = \boldsymbol{\sigma}(w^{(1)}_{10}\, x_1 + w^{(1)}_{11}\, x_2 + b^{(1)}_{2})$$

$$y = \boldsymbol{\sigma}(w^{(2)}_{0}\, h_1 + w^{(2)}_{1}\, h_2 + b^{(2)}_{1})$$



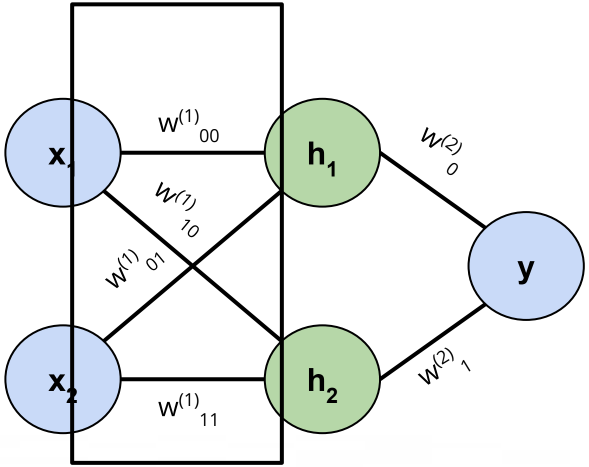
$W^{(1)}$ & $b^{(1)}$

Neural Networks - BackPropagation

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

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

Neural Networks - BackPropagation



Using the chain rule:

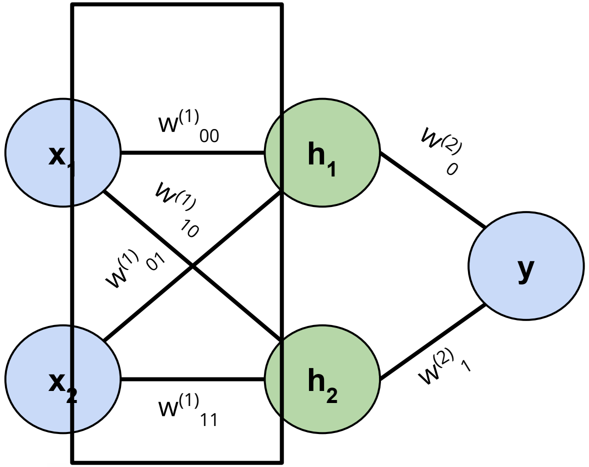
$$\frac{\partial C}{\partial W^{(1)}} = \frac{\partial C}{\partial h} \cdot \frac{\partial h}{\partial W^{(1)}} = \frac{\partial C}{\partial h} \cdot \frac{\partial h}{\partial u^{(1)}} \cdot \frac{\partial u^{(1)}}{\partial W^{(1)}} \quad \text{where} \quad u^{(1)} = w^{(1)}x + b^{(1)}$$

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

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

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:

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

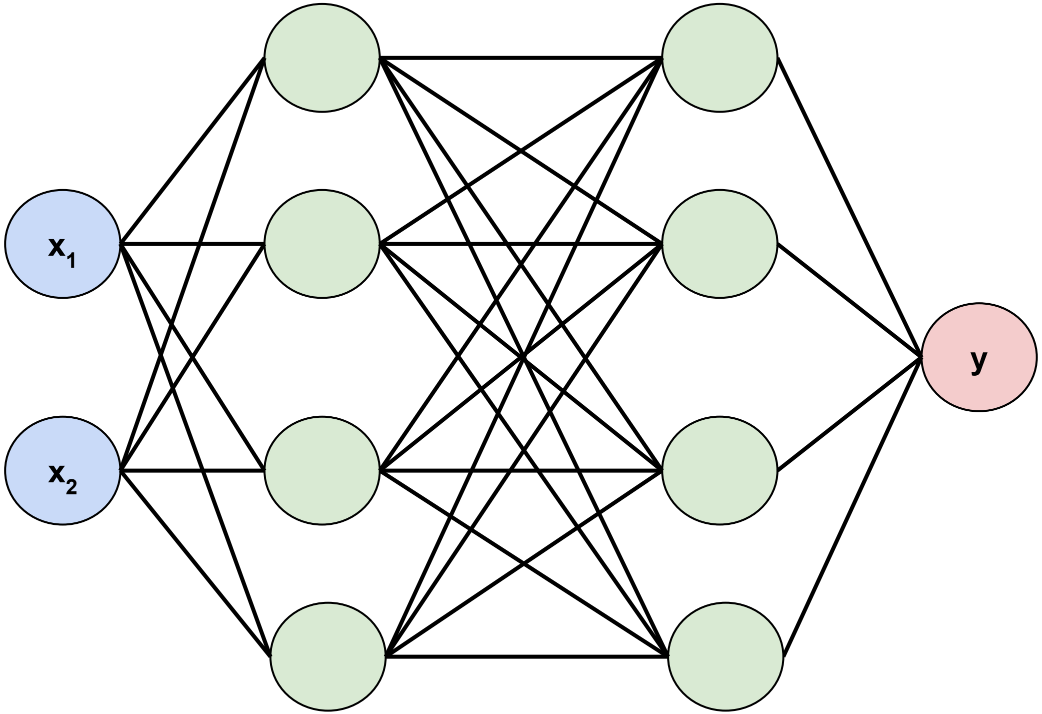
$$= \frac{\partial C}{\partial u^{(2)}} \cdot \frac{\partial u^{(2)}}{\partial h} \cdot \frac{\partial h}{\partial u^{(1)}} \cdot \frac{\partial u^{(1)}}{\partial W^{(1)}} = \frac{\partial C}{\partial u^{(2)}} \cdot W^{(2)} \cdot \sigma'(u^{(1)}) \cdot x$$

Depends on both data and weights

Initializing all weights to zero then is not a good idea

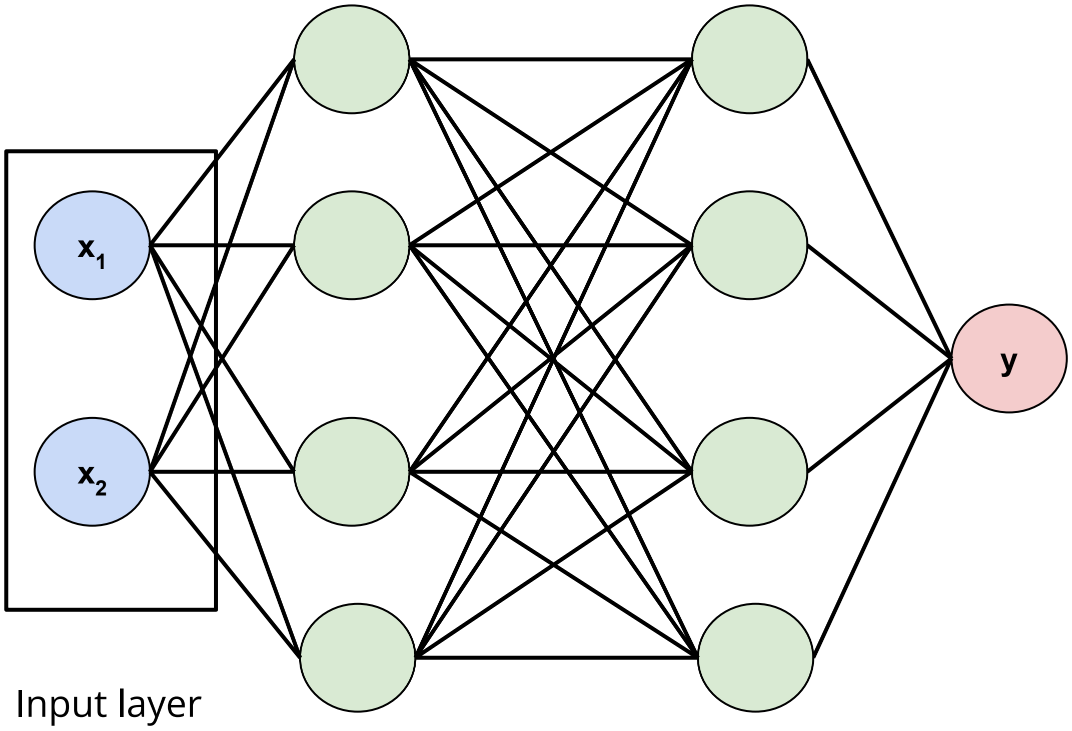
Feedforward Neural Networks

In general:



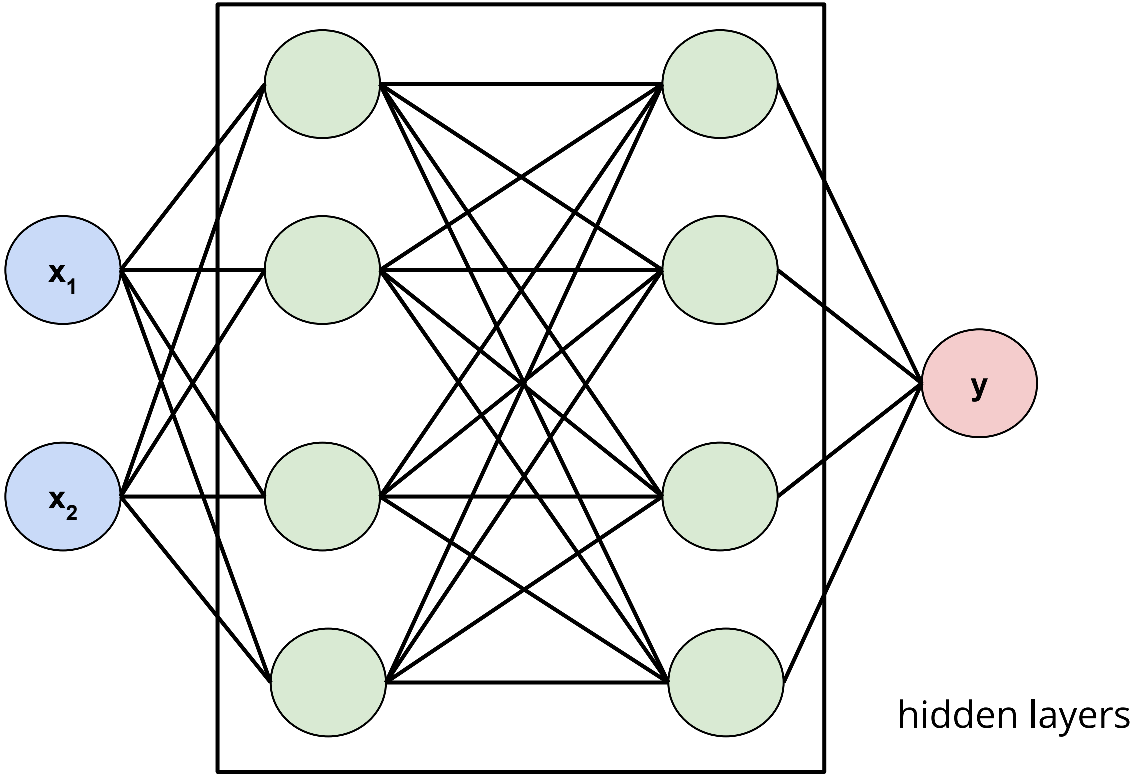
Feedforward Neural Networks

In general:



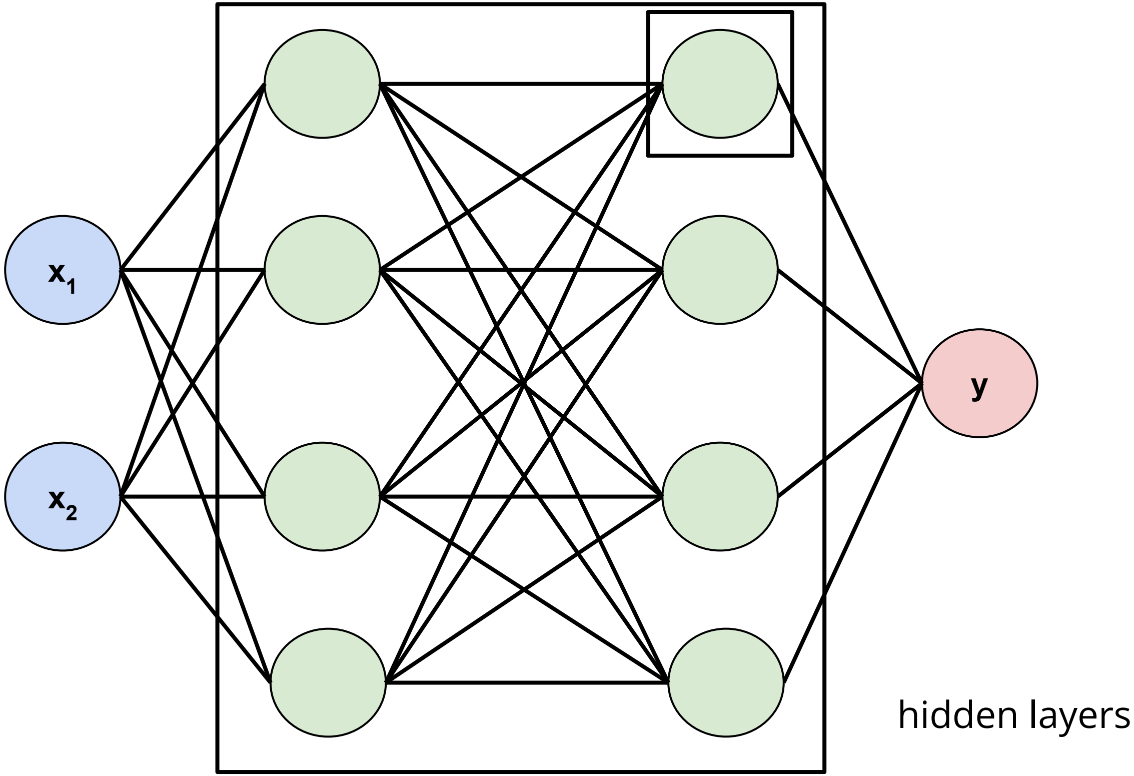
Feedforward Neural Networks

In general:



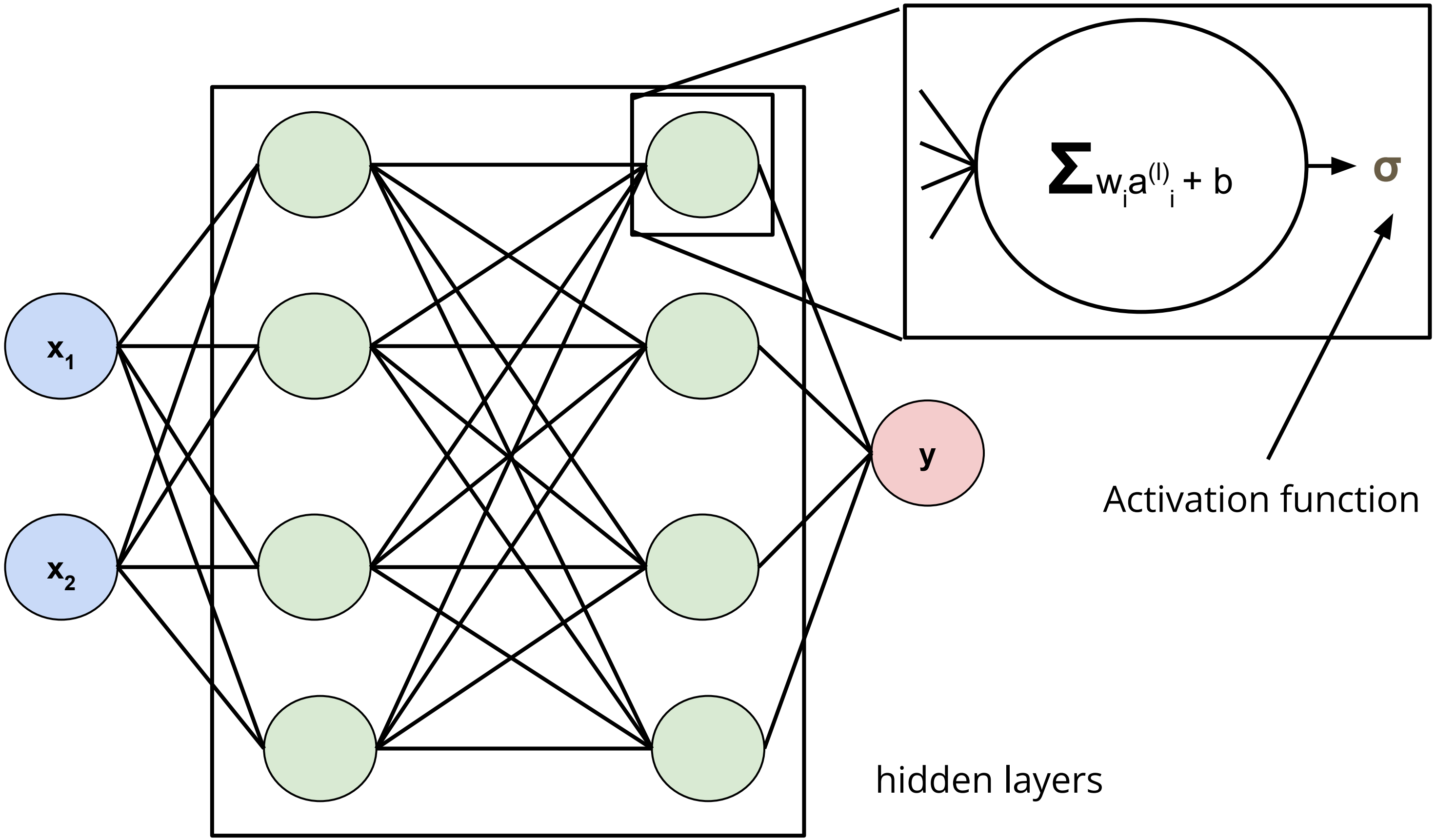
Feedforward Neural Networks

In general:



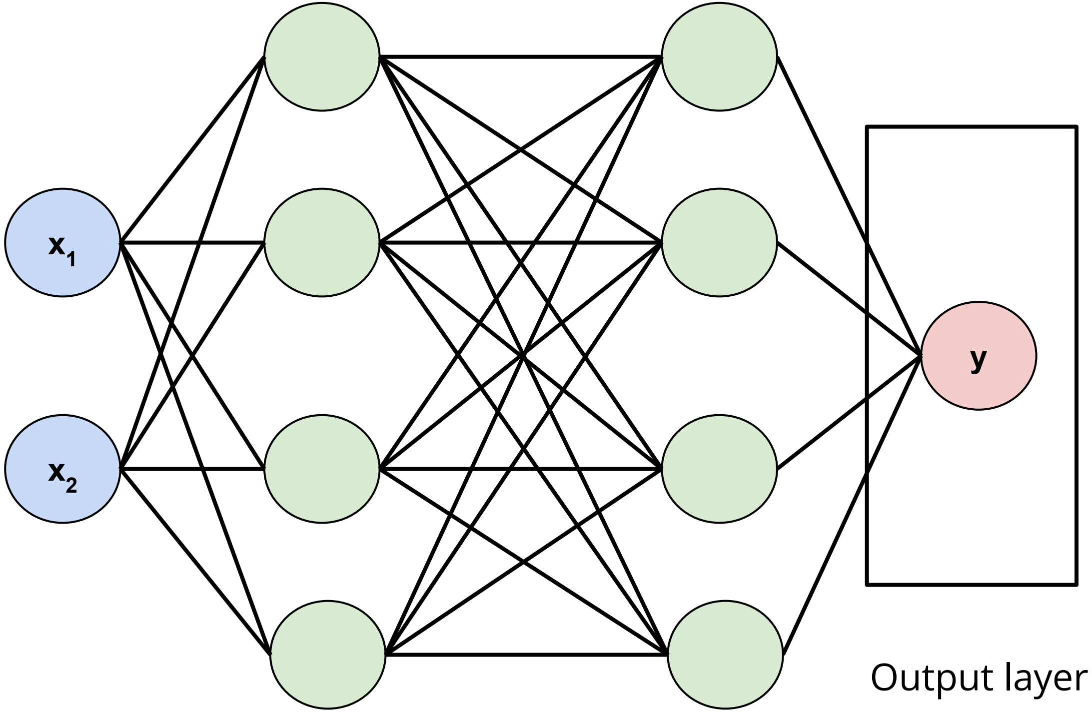
Feedforward Neural Networks

In general:



Feedforward Neural Networks

In general:



Feedforward Neural Networks

The hope:

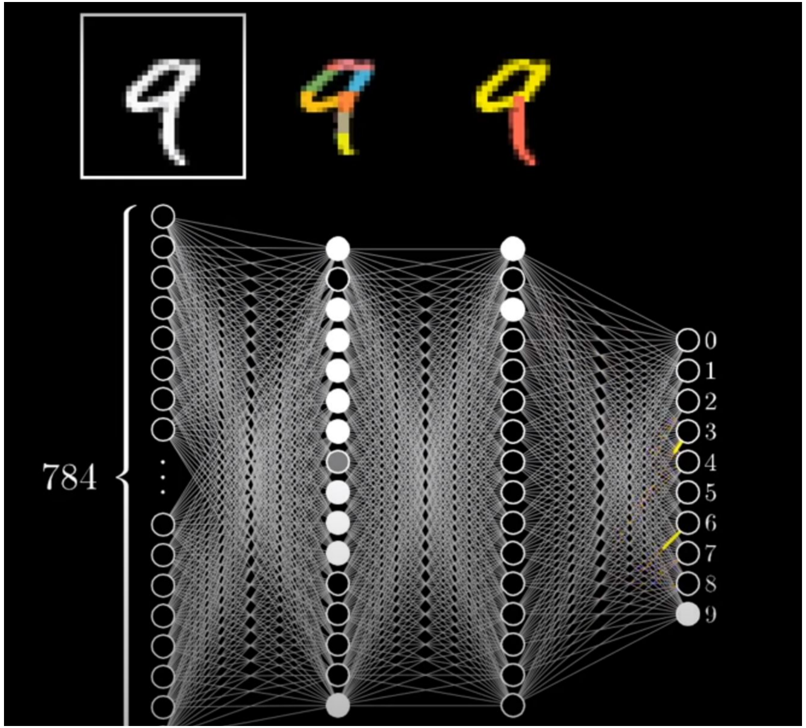


Image from 3b1b

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.

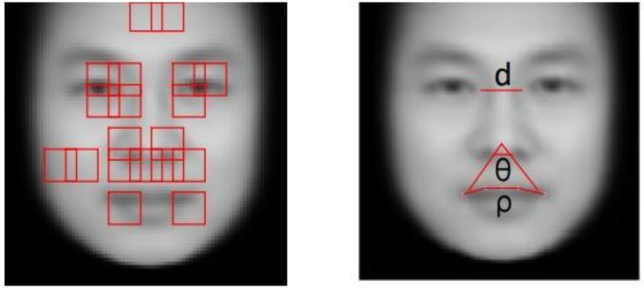
Image from "Why Should I Trust You?": Explaining the Predictions of Any Classifier (2016)Marco Tulio Ribeiro, Sameer Singh, Carlos Guestrin

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



(a) (b)

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

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

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

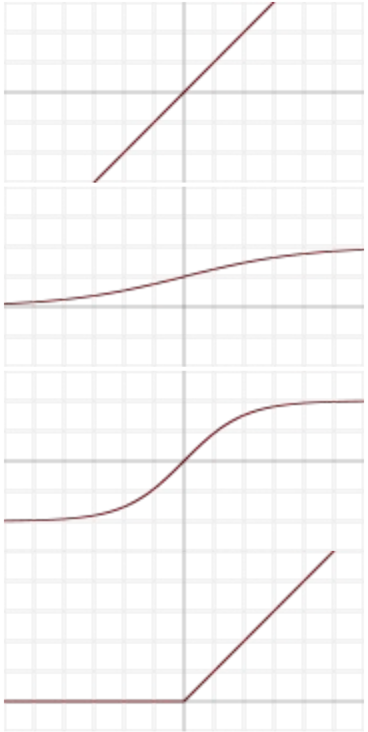
Neural Networks

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 -> **σ**(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

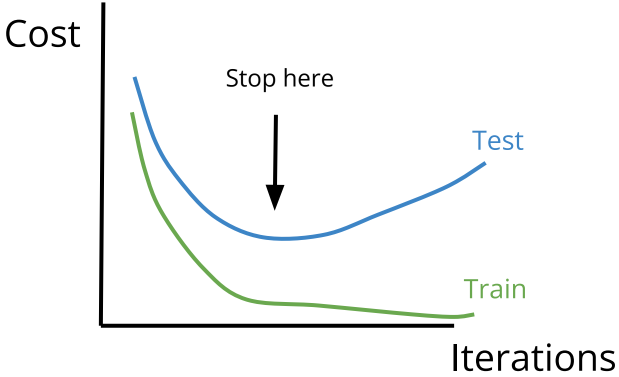
Neural Networks - Challenges

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

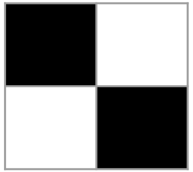
https://medium.com/mlearning-ai/tuning-neural-networks-part-ii-considerations-for-initialization-4f82e525da69

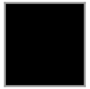
Neural Networks - Activation Functions

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

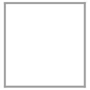
Logistic Regression Revisited

Given a 2 x 2 grid where each cell a_{ij} can take on one of two colors c_1 and c_2, find a function that can identify the following diagonal pattern:





$= c_1 = -1$

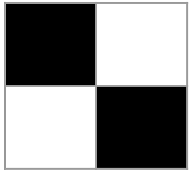


$= c_2 = 1$

Logistic Regression Revisited

That is, find **f** such that

$$\mathbf{f}\left(\begin{array}{|c|c|}\hline a_{00} & a_{01} \\ \hline a_{10} & a_{11} \\ \hline \end{array}\right) = \begin{cases} \checkmark \text{ if } \begin{array}{|c|c|}\hline a_{00} & a_{01} \\ \hline a_{10} & a_{11} \\ \hline \end{array} = \begin{array}{|c|c|}\hline c_1 & c_2 \\ \hline c_2 & c_1 \\ \hline \end{array} = \\ \times \text{ otherwise} \end{cases}$$



We can define: ✔ = 1 and ✘ = 0

Logistic Regression Revisited

We can assign weights to each cell

w_1	w_2
w_3	w_4

Logistic Regression Revisited

$$\begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} \otimes \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} = w_1 a_{00}$$

Logistic Regression Revisited

$$\begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} \otimes \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} = w_1 a_{00} + w_2 a_{01}$$

Logistic Regression Revisited

$$\begin{array}{|c|c|} \hline w_1 & w_2 \\ \hline w_3 & w_4 \\ \hline \end{array} \otimes \begin{array}{|c|c|} \hline a_{00} & a_{01} \\ \hline a_{10} & a_{11} \\ \hline \end{array} = w_1a_{00} + w_2a_{01} + w_3a_{10}$$

Logistic Regression Revisited

$$\begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} \otimes \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} = w_1 a_{00} + w_2 a_{01} + w_3 a_{10} + w_4 a_{11}$$

Logistic Regression Revisited

We can assign weights to each cell

w_1	w_2
w_3	w_4

such that:

$$w_1a_{00} + w_2a_{01} + w_3a_{10} + w_4a_{11} = b \quad \text{if diagonal pattern found}$$

w_1	w_2
w_3	w_4

$\otimes$

a_{00}	a_{01}
a_{10}	a_{11}

Logistic Regression Revisited

For example:

$$\begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} = \begin{bmatrix} -2 & 2 \\ 2 & -2 \end{bmatrix}$$

What value b do we get when applied to the diagonal pattern?

$$\begin{bmatrix} -2 & 2 \\ 2 & -2 \end{bmatrix} \otimes \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} = ?$$

Logistic Regression Revisited

Any other pattern will have a value lower:

$$\begin{bmatrix} -2 & 2 \\ 2 & -2 \end{bmatrix} \otimes \begin{bmatrix} -1 & -1 \\ 1 & -1 \end{bmatrix} = ?$$

Logistic Regression Revisited

Equivalently we can decide to move the value b to the left of the equation in order for the weighted sum to reveal a diagonal pattern at 0:

$$w_1 a_{00} + w_2 a_{01} + w_3 a_{10} + w_4 a_{11} + b = 0 \quad \text{if diagonal pattern found}$$

Logistic Regression Revisited

We could then find a function $\boldsymbol{\sigma}$ to apply to the result of this sum in order to get probabilities of being diagonal:

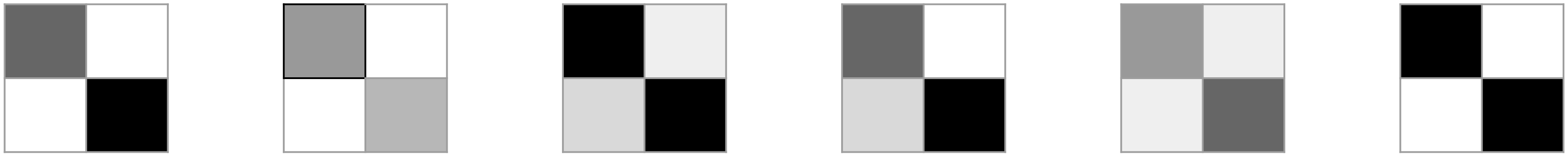
$$\boldsymbol{\sigma}(w_1a_{00} + w_2a_{01} + w_3a_{10} + w_4a_{11} + b) > ½ \text{ if } w_1a_{00} + w_2a_{01} + w_3a_{10} + w_4a_{11} + b > 0$$

Logistic Regression Revisited

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

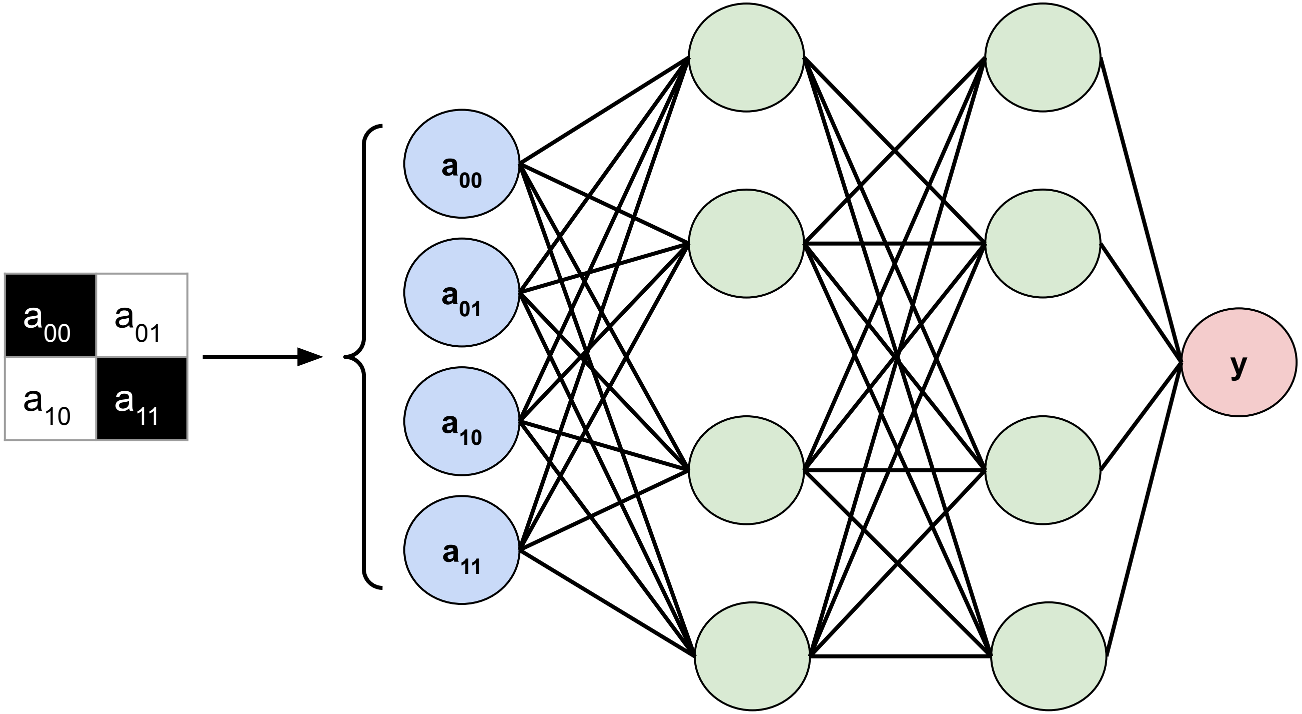
When **σ** is the logit^{-1} (also called sigmoid) function, this is Logistic Regression.

So for each cell we're looking to learn a weight w_i that makes **σ** larger for diagonal patterns. The bias term b lets us account for systemic dimming or brightening of cells (i.e. when the data is not normalized).

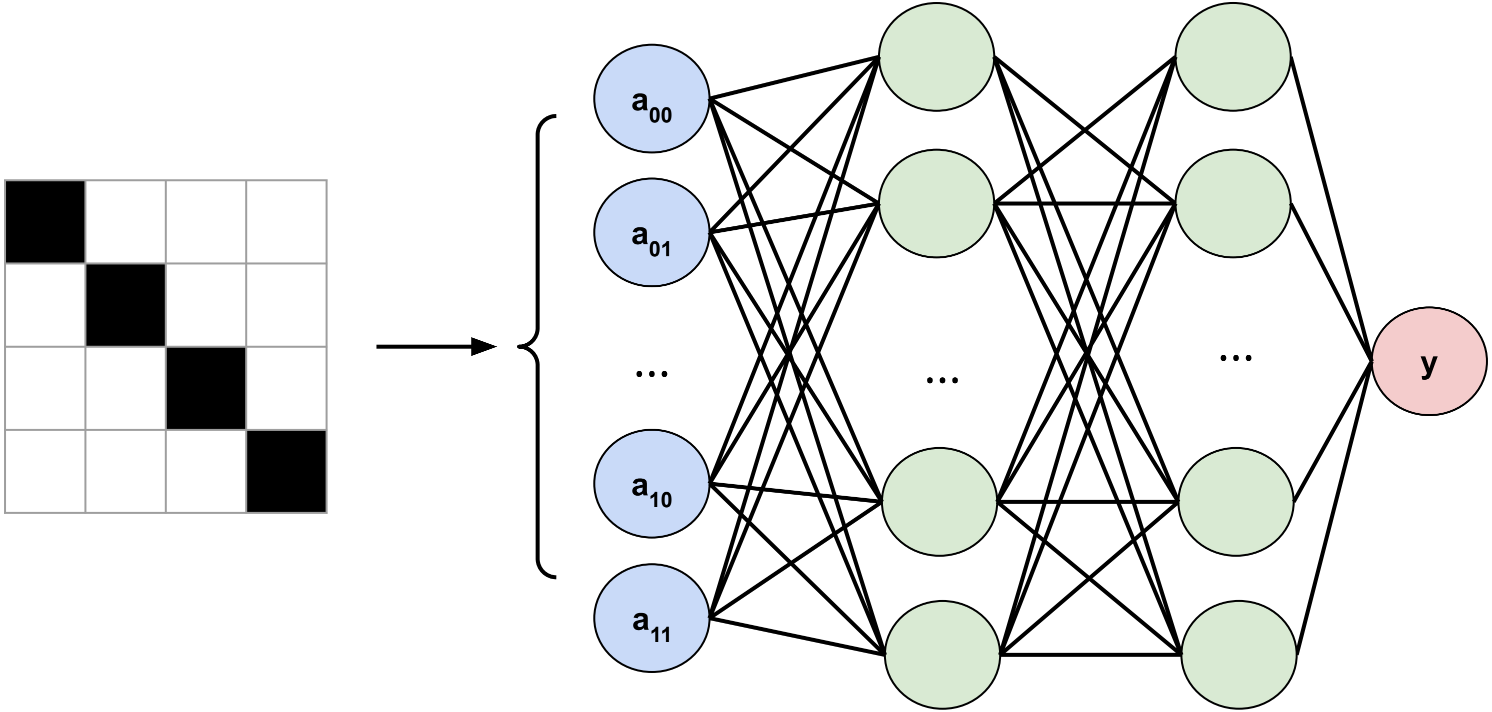


.92 .81 .95 .73 .68 .99

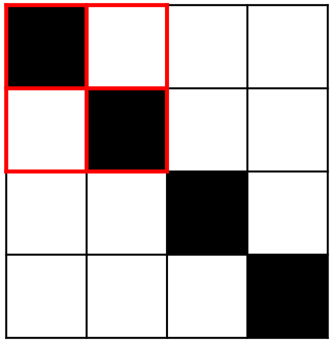
Neural Networks - Convolutional Neural Networks



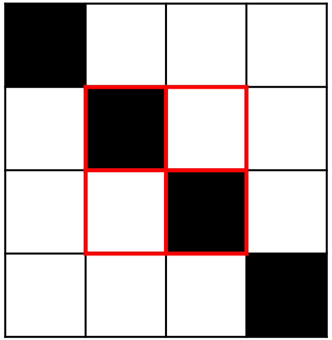
Neural Networks - Convolutional Neural Networks



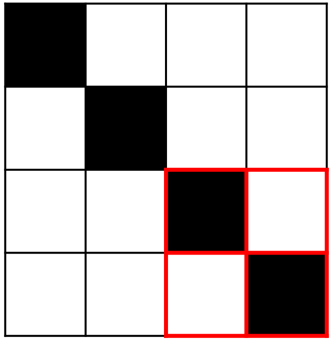
Neural Networks - Convolutional Neural Networks



Neural Networks - Convolutional Neural Networks



Neural Networks - Convolutional Neural Networks



Neural Networks - Convolutional Neural Networks

Recall: Our network learns weights for each cell

w_1	w_2
w_3	w_4

Neural Networks - Convolutional Neural Networks

$$\begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} \otimes \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} = w_1 a_{00}$$

Neural Networks - Convolutional Neural Networks

$$\begin{bmatrix} w_1 & \boxed{w_2} \\ w_3 & w_4 \end{bmatrix} \otimes \begin{bmatrix} a_{00} & \boxed{a_{01}} \\ a_{10} & a_{11} \end{bmatrix} = w_1 a_{00} + w_2 a_{01}$$

Neural Networks - Convolutional Neural Networks

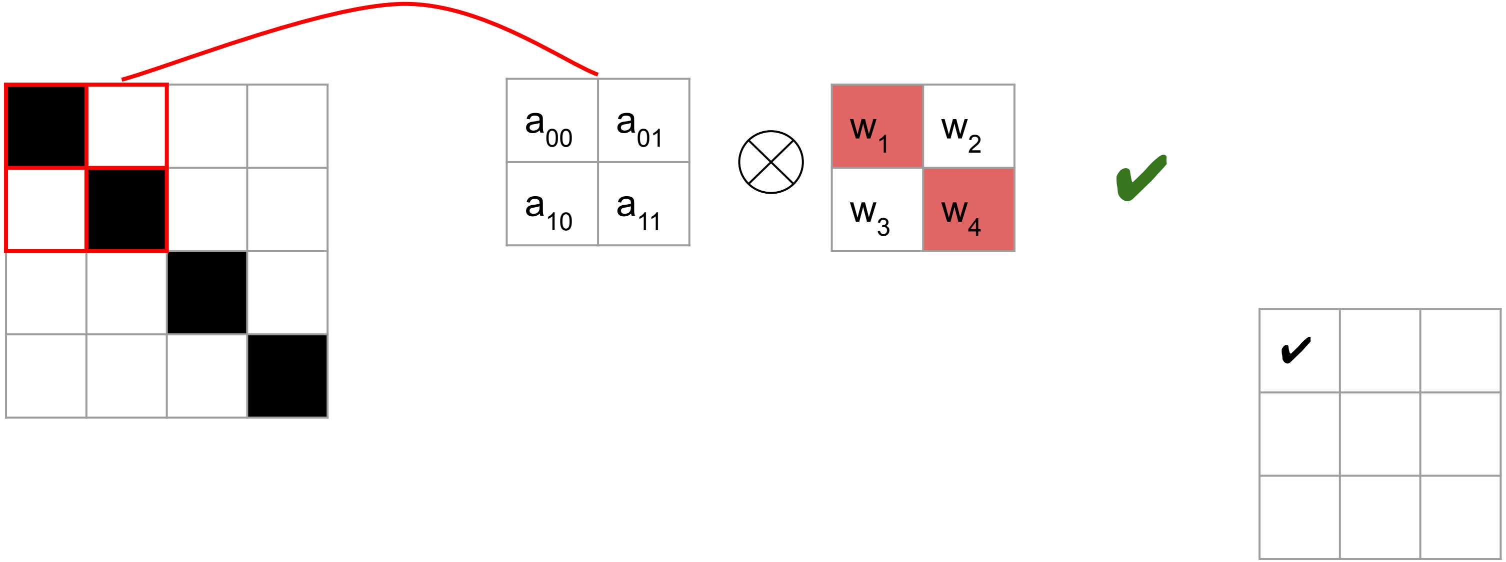
$$\begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} \otimes \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} = w_1 a_{00} + w_2 a_{01} + w_3 a_{10}$$

Neural Networks - Convolutional Neural Networks

$$\begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} \otimes \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} = w_1 a_{00} + w_2 a_{01} + w_3 a_{10} + w_4 a_{11}$$

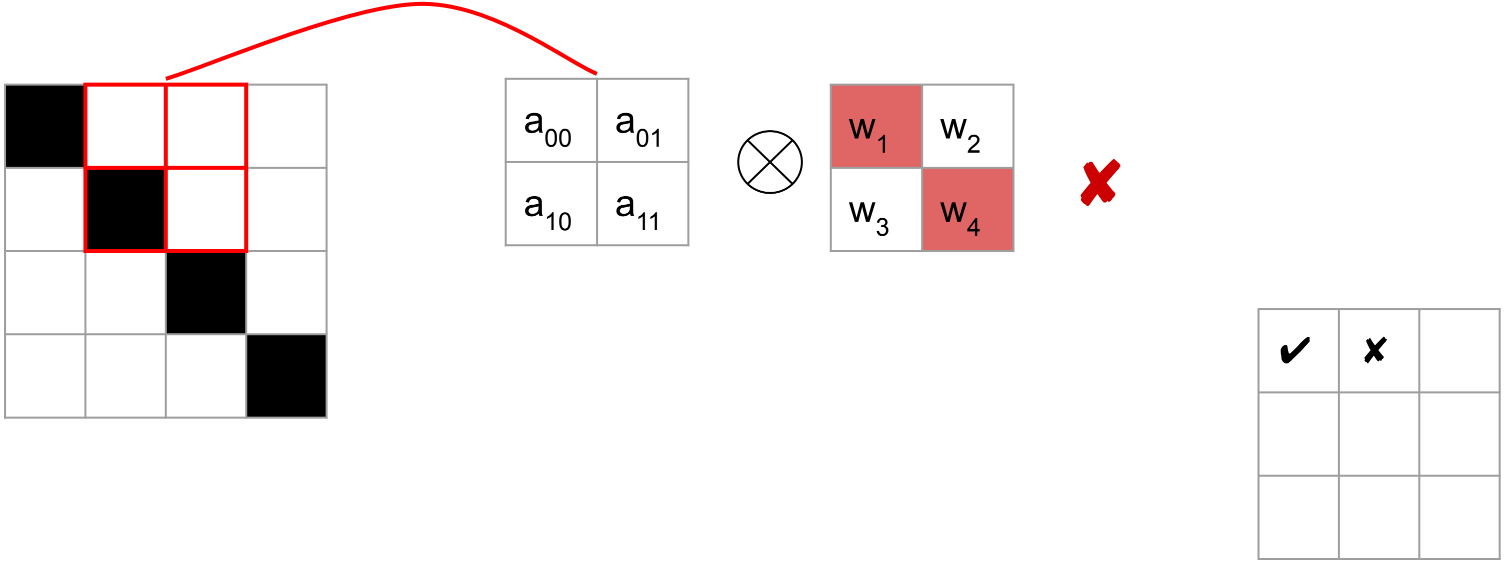
Neural Networks - Convolutional Neural Networks

Knowing this, what happens if we slide this filter across the larger diagonal?



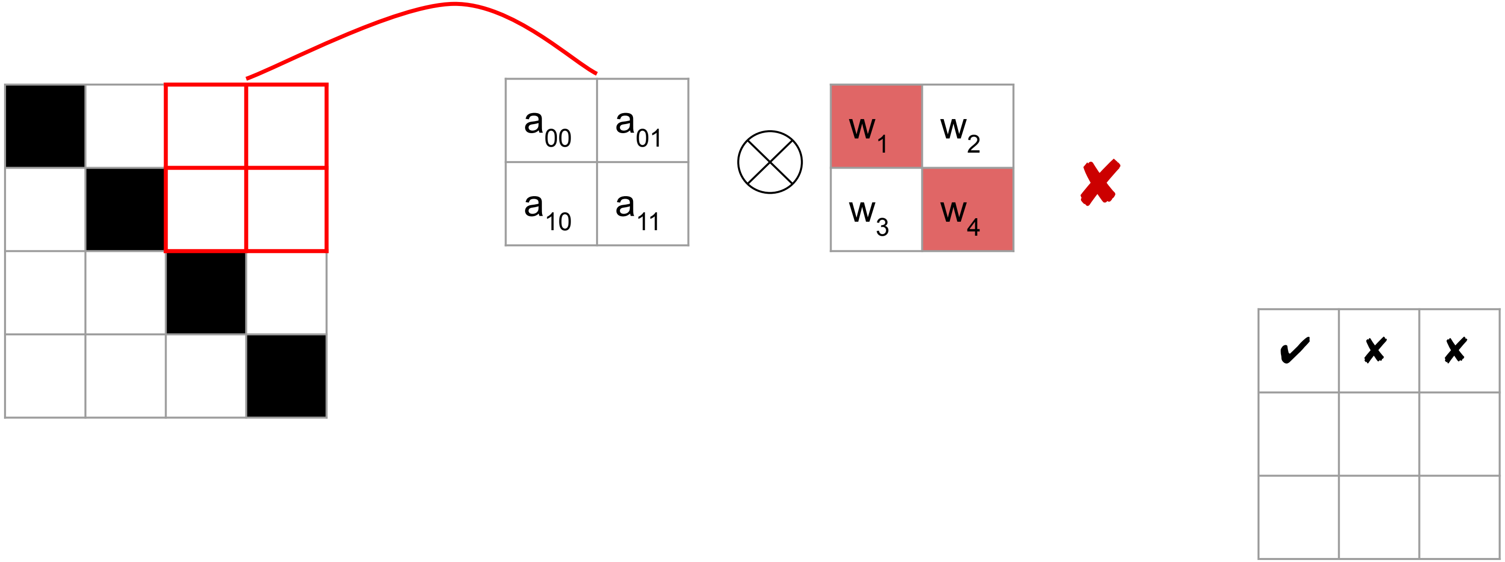
Neural Networks - Convolutional Neural Networks

Knowing this, what happens if we slide this filter across the larger diagonal?



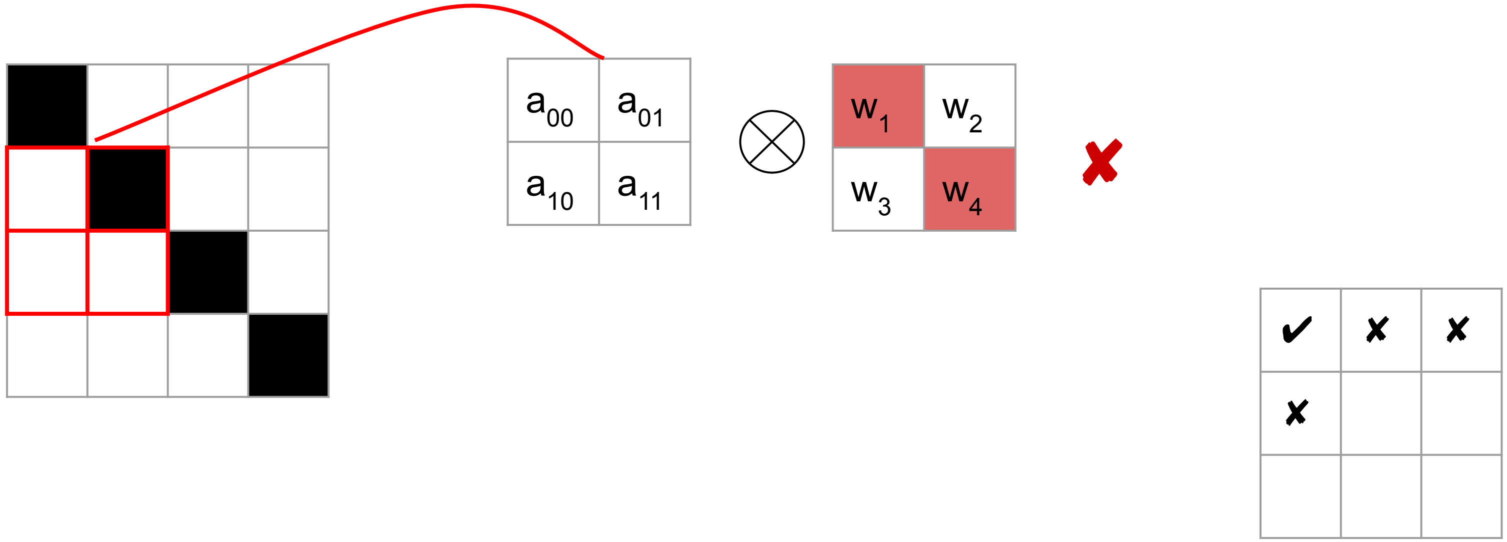
Neural Networks - Convolutional Neural Networks

Knowing this, what happens if we slide this filter across the larger diagonal?



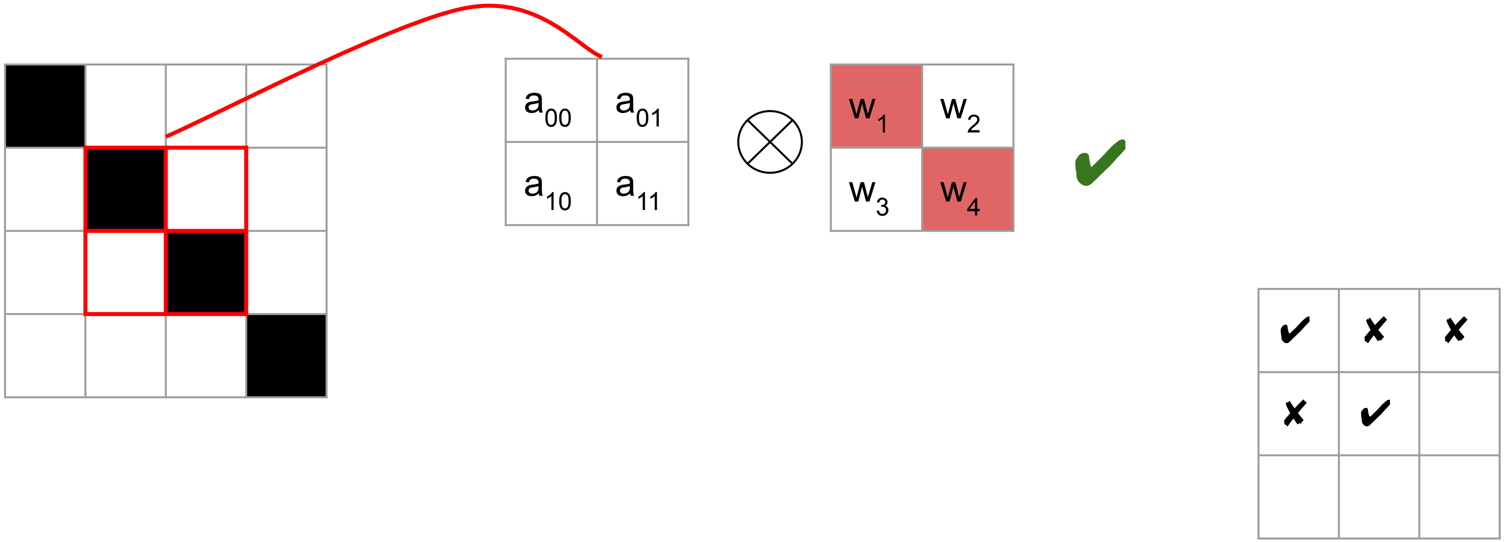
Neural Networks - Convolutional Neural Networks

Knowing this, what happens if we slide this filter across the larger diagonal?



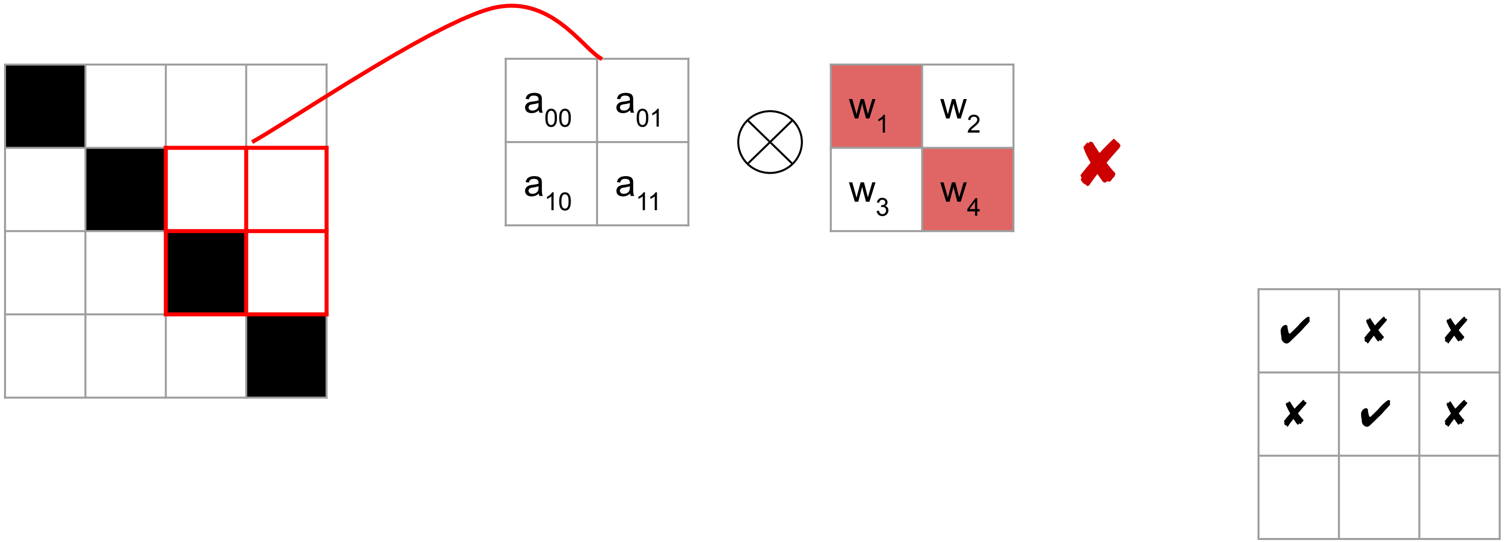
Neural Networks - Convolutional Neural Networks

Knowing this, what happens if we slide this filter across the larger diagonal?



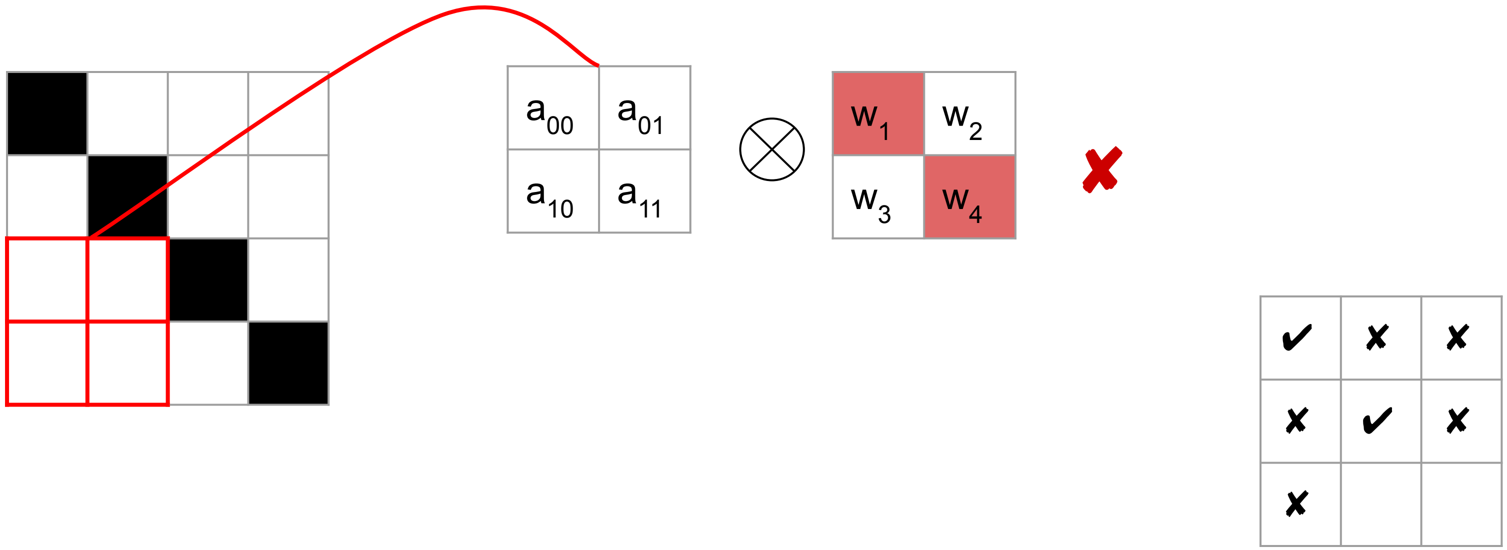
Neural Networks - Convolutional Neural Networks

Knowing this, what happens if we slide this filter across the larger diagonal?



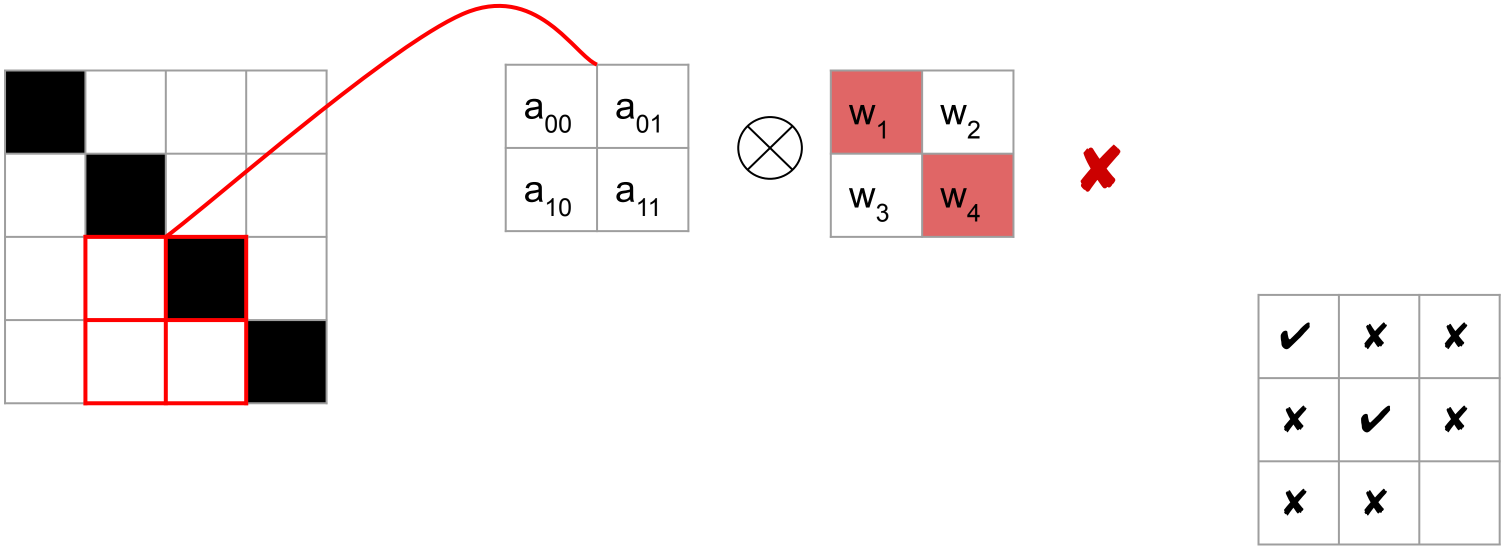
Neural Networks - Convolutional Neural Networks

Knowing this, what happens if we slide this filter across the larger diagonal?



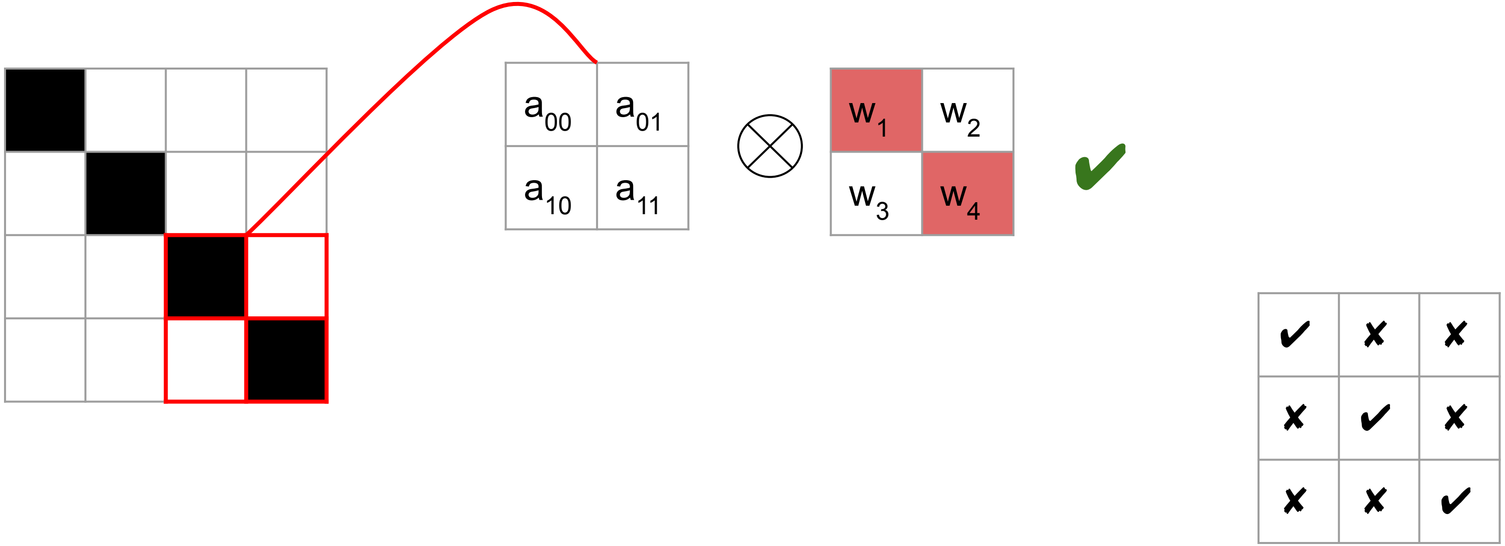
Neural Networks - Convolutional Neural Networks

Knowing this, what happens if we slide this filter across the larger diagonal?



Neural Networks - Convolutional Neural Networks

Knowing this, what happens if we slide this filter across the larger diagonal?



Neural Networks - Convolutional Neural Networks

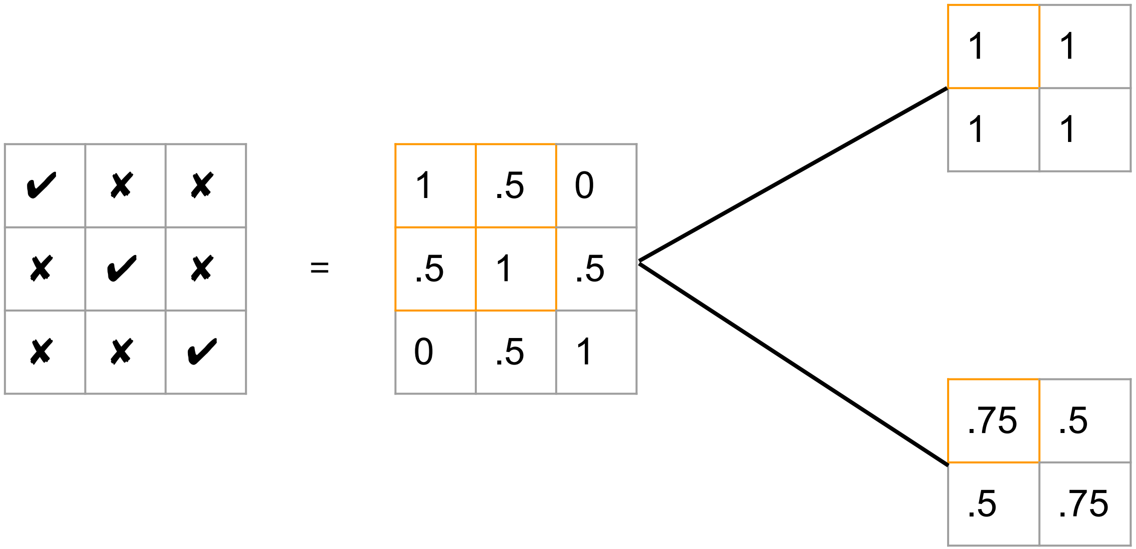
Creating such a filter allows us to:

1. Reduce the number of weights
2. Capture features all over the image

The process of applying a filter (or kernel) is called a convolution

Neural Networks - Convolutional Neural Networks

To reduce the weights even further, another phase is done after convolution called Pooling:



Max Pooling

Average Pooling

Neural Networks - Convolutional Neural Networks

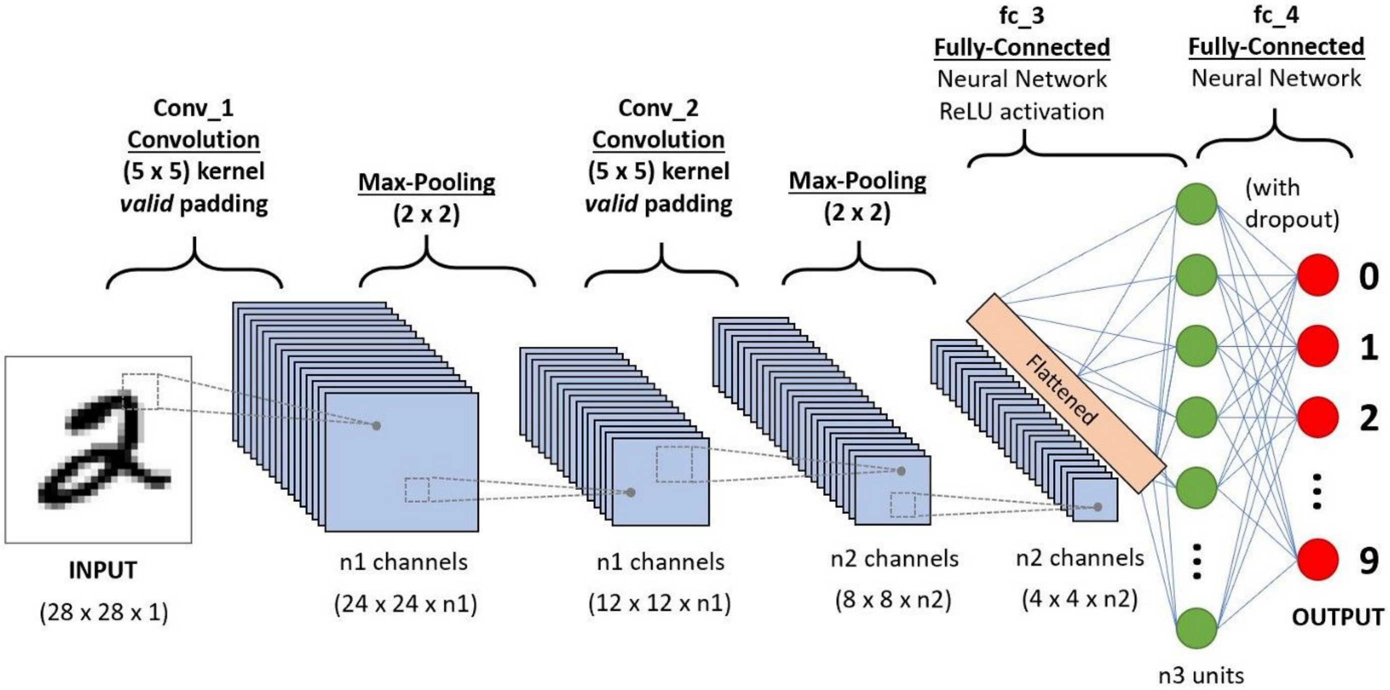


Image from https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Neural Networks - Convolutional Neural Networks

Main application: Computer vision

Recurrent Neural Networks

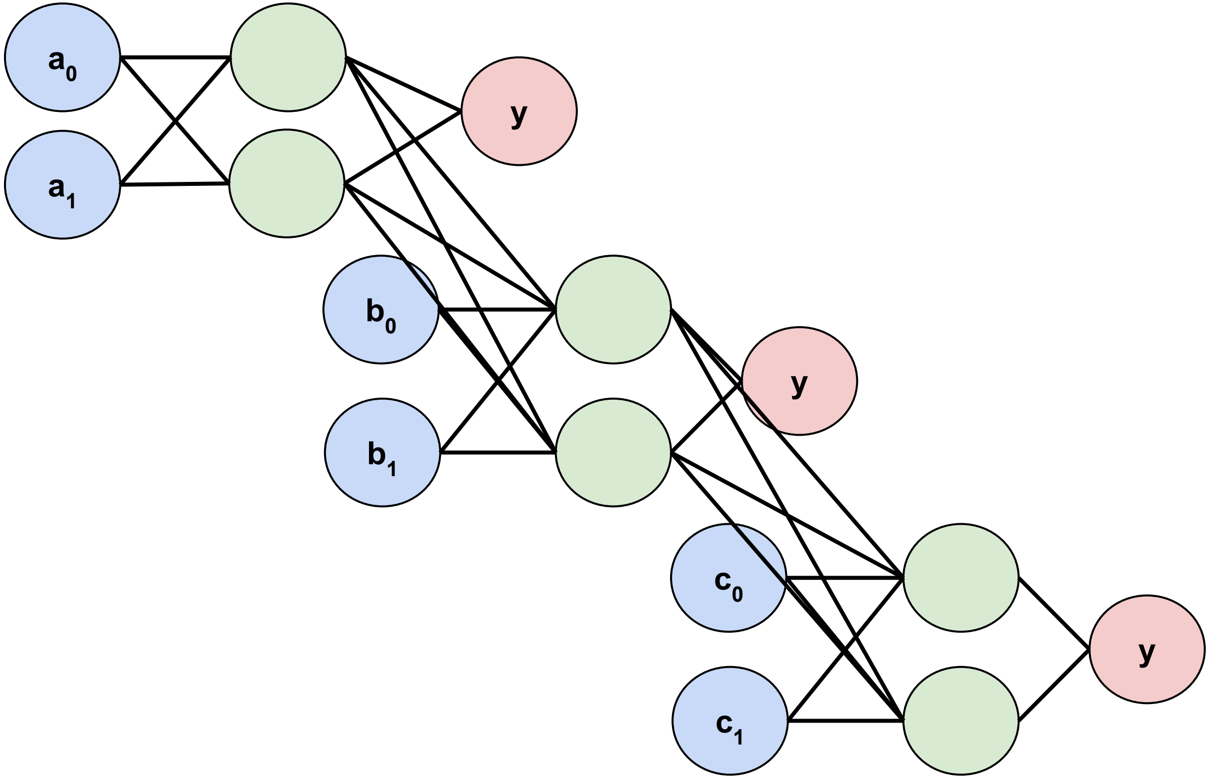
Handling sequences of input.

Intuition: What a word is / might be in a sentence is easier to figure out if you know the words around it.

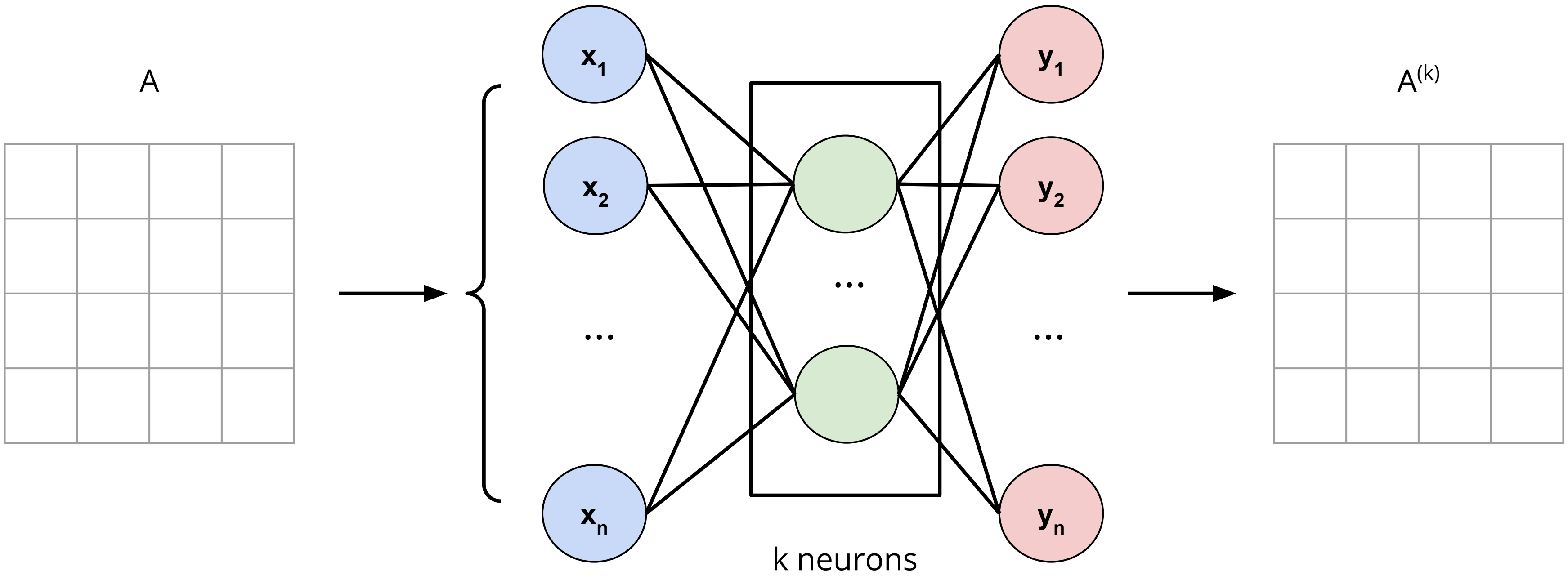
Applications:

1. Predicting the next word
2. Translation
3. Speech Recognition
4. Video Tagging

Recurrent Neural Networks



Neural Networks - Auto Encoders



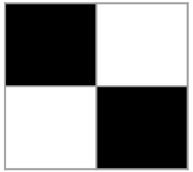
Intro to Neural Networks

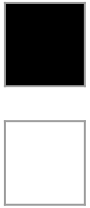
https://medium.com/@gallettilance/list/introducing-neural-networks-d74f0dc25400

EXTRA

Logistic Regression Revisited

Given a 2 x 2 grid where each cell a_{ij} can take on one of two colors c_1 and c_2, find a function that can identify the following diagonal pattern:





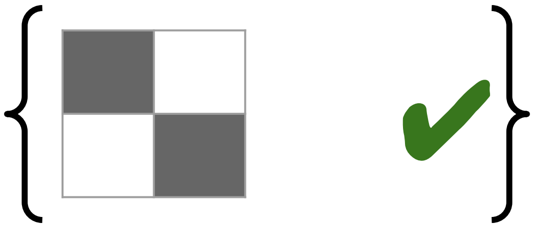
$= c_1 = -1$

$= c_2 = 1$

Gradient Descent

Let's apply this to our diagonal problem to find the weights and bias for logistic regression.

Assume we have the following dataset:



$[0\ 1\ 1\ 0]^T$

Gradient Descent

Recall:

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

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

Gradient Descent

We need to compute ∇Cost(w, b):

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

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

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

Gradient Descent

 $= [0\ 1\ 1\ 0]^T$

1. Start with random w and b:

$w = [0\ 0\ 0\ 0]^T$, $b = 0$

Note: $\boldsymbol{\sigma}(0) = 0.5$

Gradient Descent

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

2. Compute the Cost(w, b)

Cost($[0\ 0\ 0\ 0]^T$, 0) = -1 log(**σ**(0)) = -log(0.5)

Gradient Descent

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

= [0 1 1 0]^T

3. Compute the gradient ∇Cost at (w, b)

$$\frac{\partial}{\partial w}\text{Cost} = \frac{1}{1}\sum_{i=1}^{1} \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} (1 - \sigma(0)) = \begin{bmatrix} 0 \\ 1/2 \\ 1/2 \\ 0 \end{bmatrix}$$

Recall we only have one data point

Gradient Descent

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

 = [0 1 1 0]^T

3. Compute the gradient ∇Cost at (w, b)

$$\frac{\partial}{\partial b}\text{Cost} = \frac{1}{1}\sum_{i=1}^{1}(\sigma(0) - 1) = -\frac{1}{2}$$

Recall we only have one data point

Gradient Descent

4. Adjust w & b by taking α steps in the direction of $-\nabla \text{Cost}_{(w,\,b)}$

$$w_{\text{new}} = -\alpha \begin{bmatrix} 0 \\ 1/2 \\ 1/2 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ -\alpha/2 \\ -\alpha/2 \\ 0 \end{bmatrix}$$

$$b_{\text{new}} = \alpha \frac{1}{2} + 0 = \frac{\alpha}{2}$$

Gradient Descent

5. Compute the updated Cost

$$\text{Cost}\left(\begin{bmatrix} 0 \\ -\alpha/2 \\ -\alpha/2 \\ 0 \end{bmatrix}, \frac{\alpha}{2}\right) = -\log\left(\sigma\left(\alpha + \frac{1}{2}\right)\right)$$

For what values of α is the Cost reduced?