ICDSS Lecture 4: Neural Networks

Instructor: *Myungchan Kim,*
Chemical Engineering

"Pre-requisites" of L4

Not a mathematically rigorous lecture. We assume you know:

- Matrix Multiplication
- Chain Rule
- Linear and Logistic Regression

$$\begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix} \begin{bmatrix} b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 \\ b_7 & b_8 & b_9 \end{bmatrix} = \begin{bmatrix} c_1 & c_2 & c_3 \\ c_4 & c_5 & c_6 \\ c_7 & c_8 & c_9 \end{bmatrix}$$

$$\frac{dL}{dW_1} = \frac{dL}{da_2}\frac{da_2}{dz_2}\frac{dz_2}{da_1}\frac{da_1}{dz_1}\frac{dz_1}{dW_1}$$

Credit: https://charchithowitzer.medium.com/matrix-multiplication-why-is-it-a-big-deal-cc8ef7490008

Overview of L4

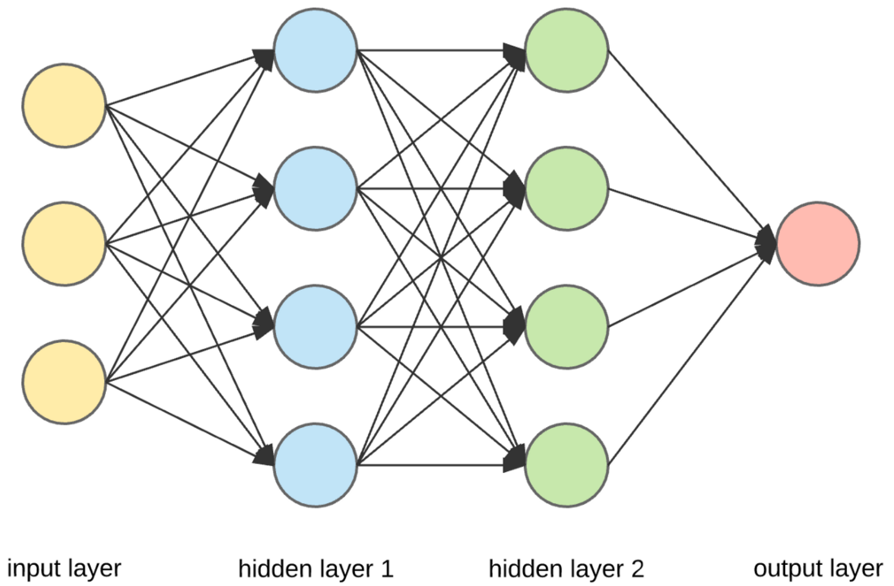
1 Linear Regression as a NN

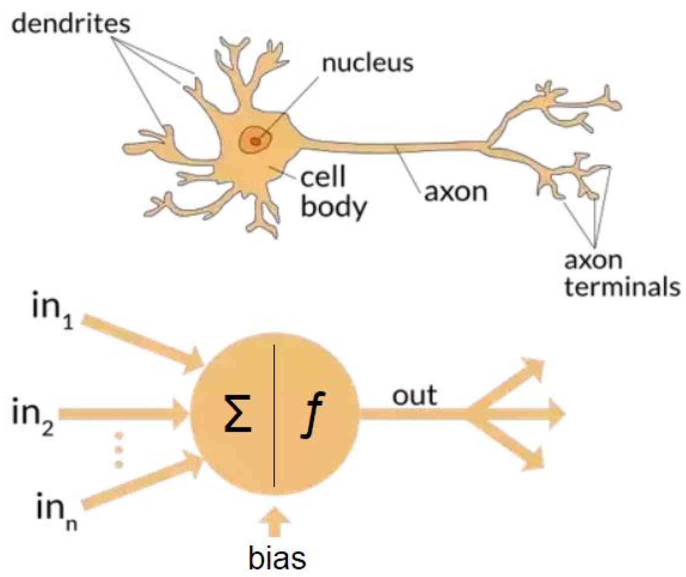
2 Logistic Regression as a NN

3 General NNs

Why are they called Neural Networks?

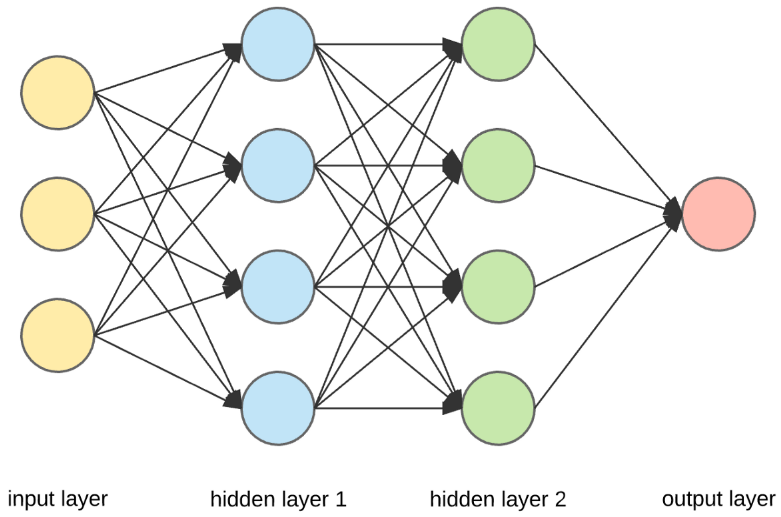
- Mimics the human brain (not exactly!)
- Output signals from other neurons as the input
- Each signal has a different “weighting”
- If signal is above some threshold value, it is “activated.”

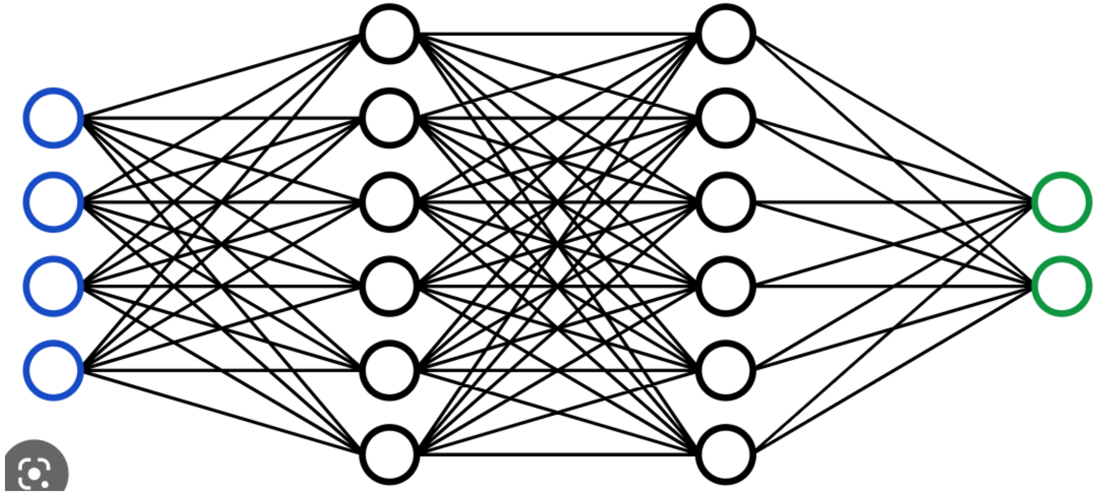




Credit: https://charchithowitzer.medium.com/matrix-multiplication-why-is-it-a-big-deal-cc8ef7490008

Some Notation

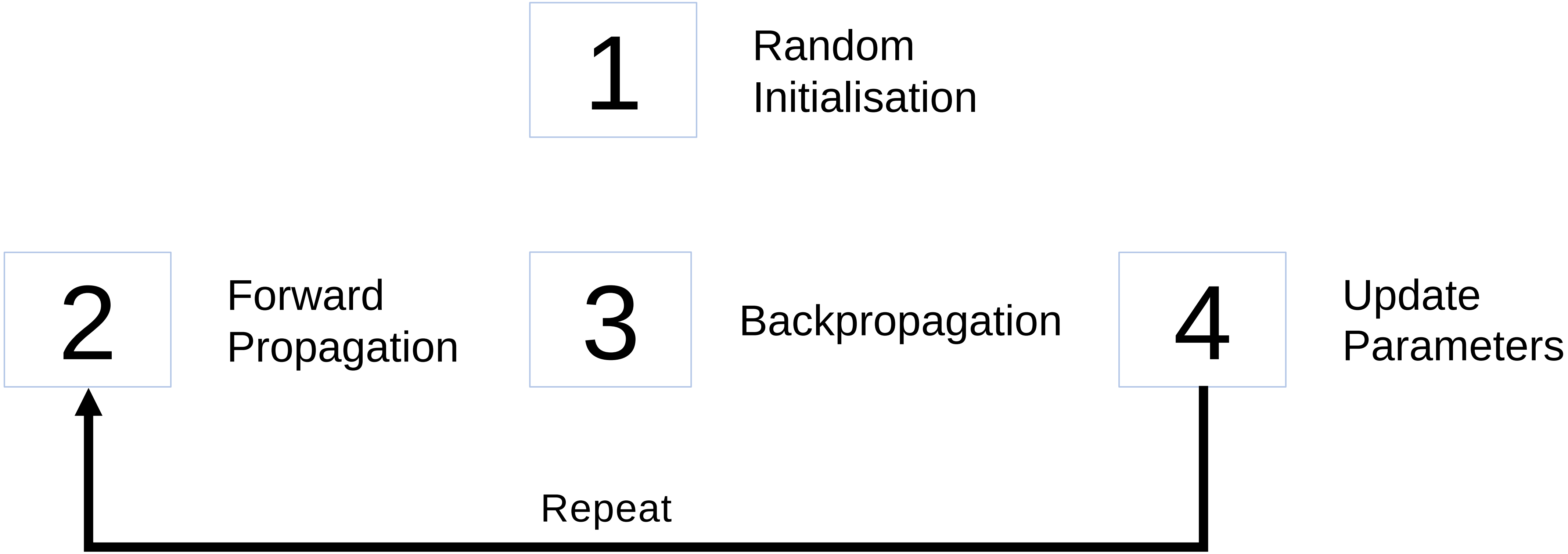




- **Input, Label Matrix:** $\boldsymbol{X, Y}$
- **Prediction Matrix:** $\boldsymbol{\hat{y}}$
- Number Of Layers: All layers excluding input
- Input Layer: Layer 0
- **Number of examples: m**
- Input, Label Size: n_x, n_y

Credit: https://medium.com/@marcellvollmer/how-to-make-it-simple-to-explain-ai-ml-dl-and-data-science-a49e54d54a12

Neural Networks: General Steps



Linear Regression (Single Feature)

Linear Regression (Single Feature):

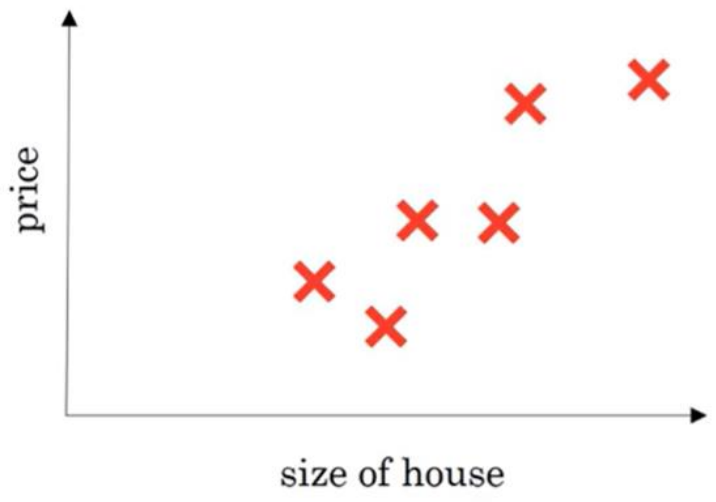
Consider a single feature linear regression problem:

- Predicting House Price based on Total Size

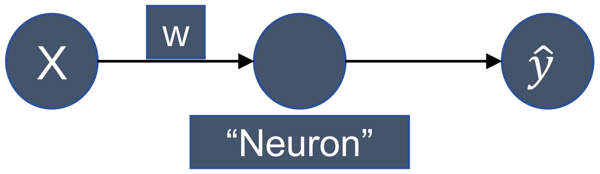
Linear Regression attempts to fit data into the form:

$$y = wx + b$$

w: Weighting, b: Bias



This can be visualised as a “Neural Network”



Credit: deeplearning.ai

Linear Regression (Single Feature):

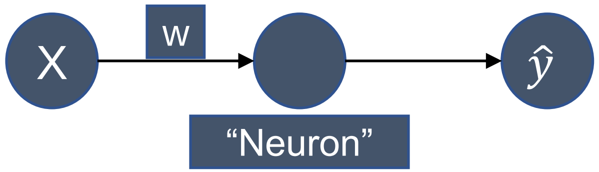
First, the "network" will randomly guess w and b.

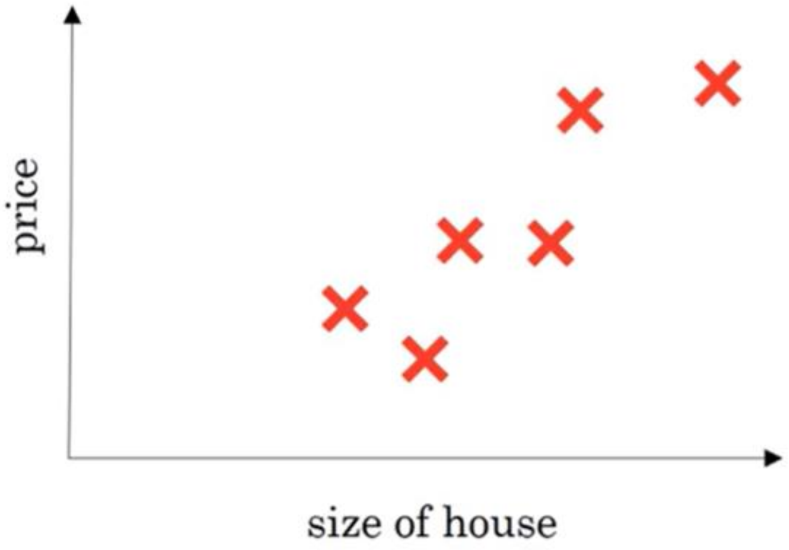
Then, the model will calculate:

$$Z = wx + b$$

This value acts as the input of the Neuron

- **Forward Propagation!**





In Linear Regression, there is no special “**activation function:**”

$$\hat{y} = Z = wx + b$$

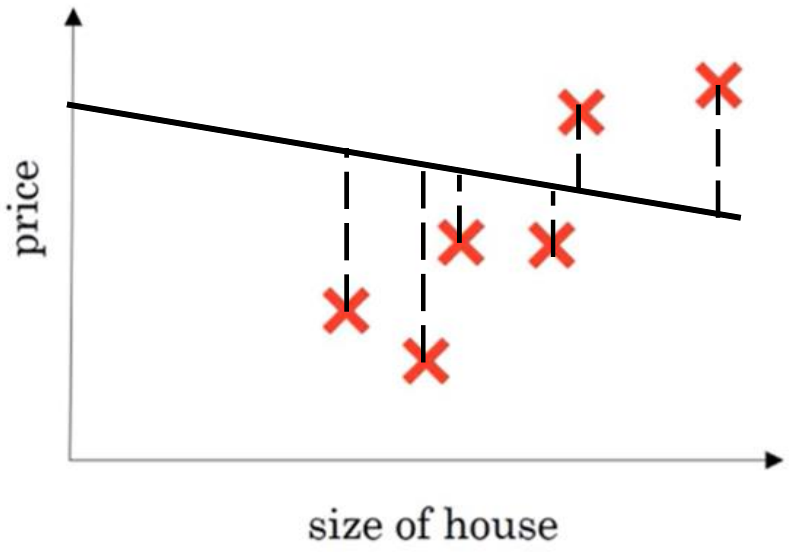
Credit: deeplearning.ai

Linear Regression (Single Feature):

The "network" just guessed values, predictions can't be accurate

- A loss function (performance measure) needs to be defined and minimised
- Use Mean Squared Error (MSE)

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$



How do we change the parameters (such as w) so that we minimize the MSE?

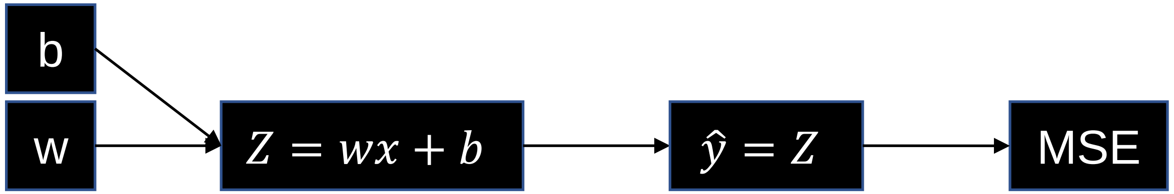
- Update the value of the parameter using its gradient **(gradient descent)**

Credit: deeplearning.ai

Linear Regression (Single Feature):

How do we change the parameters (such as w) of the network to minimize the MSE?

- Examine how MSE is calculated in terms of a **computation diagram**



Consider derivatives of MSE with respect to w and b

- Use chain rule to get

$$\frac{d(MSE)}{dw} = \frac{d(MSE)}{d\hat{y}} \frac{d\hat{y}}{dZ} \frac{dZ}{dw} \qquad \frac{d(MSE)}{db} = \frac{d(MSE)}{d\hat{y}} \frac{d\hat{y}}{dZ} \frac{dZ}{db}$$

See that we are calculating derivatives from the output layer to the input layer (backwards)

Backpropagation!

Credit: deeplearning.ai

Linear Regression (Single Feature):

Consider derivatives of MSE with respect to w and b

- Use chain rule to get:

$$\frac{d(MSE)}{dw} = \frac{d(MSE)}{d\hat{y}}\frac{d\hat{y}}{dZ}\frac{dZ}{dw} \qquad \frac{d(MSE)}{db} = \frac{d(MSE)}{d\hat{y}}\frac{d\hat{y}}{dZ}\frac{dZ}{db}$$

$$\frac{d(MSE)}{d\hat{y}} = \frac{d}{d\hat{y}}\left(\frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2\right) \longrightarrow \frac{d(MSE)}{d\hat{y}} = \frac{1}{m}\sum_{i=1}^{m}\left(\frac{d}{d\hat{y}}(y_i - \hat{y}_i)^2\right)$$

$$\frac{d(MSE)}{d\hat{y}} = -\frac{2}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)$$

Linear Regression (Single Feature):

Consider derivatives of MSE with respect to w and b

$$Z = wx + b$$

$$\frac{d(MSE)}{d\hat{y}} = -\frac{2}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i) \qquad \frac{dZ}{d\hat{y}} = 1; \frac{d\hat{y}}{dw} = x\,; \frac{d\hat{y}}{db} = 1$$

$$\frac{d(MSE)}{dw} = \frac{d(MSE)}{d\hat{y}}\frac{d\hat{y}}{dZ}\frac{dZ}{dw} = -\frac{2}{m}x\sum_{i=1}^{m}(y_i - \hat{y}_i)$$

$$\frac{d(MSE)}{db} = \frac{d(MSE)}{d\hat{y}}\frac{d\hat{y}}{dZ}\frac{dZ}{db} = -\frac{2}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)$$

Linear Regression (Single Feature):

The gradients can be used to **update parameters** (also known as “learning”)

$$\frac{d(MSE)}{dw} = \frac{d(MSE)}{d\hat{y}}\frac{d\hat{y}}{dZ}\frac{dZ}{dw} \qquad \frac{d(MSE)}{db} = \frac{d(MSE)}{d\hat{y}}\frac{d\hat{y}}{dZ}\frac{dZ}{db}$$

Update as following:

$$w \rightarrow w - \alpha\frac{d(MSE)}{dw} \qquad b \rightarrow b - \alpha\frac{d(MSE)}{db}$$

α Is the **learning rate**

- It is a **hyperparameter**: it needs to be specified when the network is created

Also, see that the negative value of the gradient is taken

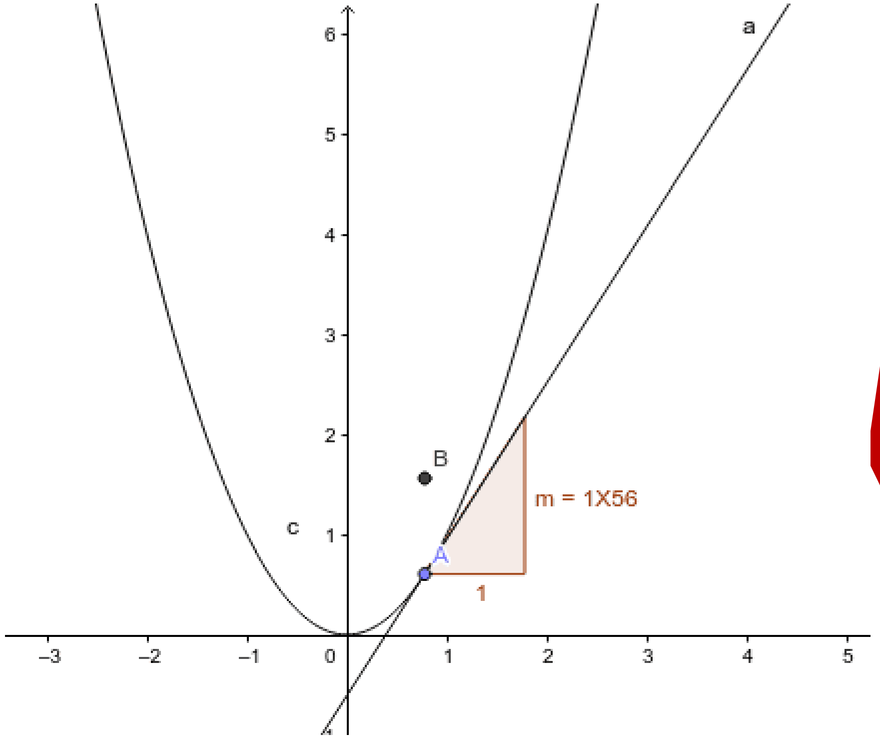
Linear Regression (Single Feature):

Why are the negative values of gradients used?

$$w \rightarrow w - \alpha \frac{d(MSE)}{dw} \qquad\qquad b \rightarrow b - \alpha \frac{d(MSE)}{db}$$

- The gradient points towards the direction with the steepest increase
- To get direction of steepest decrease, the negative value should be taken

Forward, Backpropagation and Updating of parameters will repeat for a set number of "**epochs**"

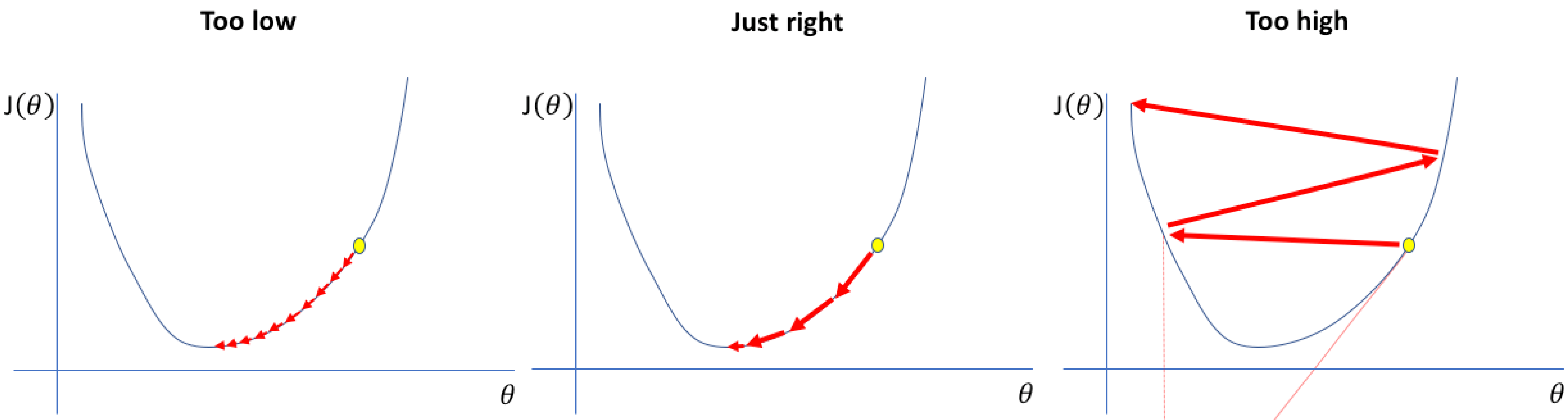


Credit: deeplearning.ai

Linear Regression (Single Feature):

Finding the right value for the learning rate can be an iterative process

$$w \rightarrow w - \alpha \frac{d(MSE)}{dw} \qquad\qquad b \rightarrow b - \alpha \frac{d(MSE)}{db}$$



A small learning rate requires many updates before reaching the minimum point

The optimal learning rate swiftly reaches the minimum point

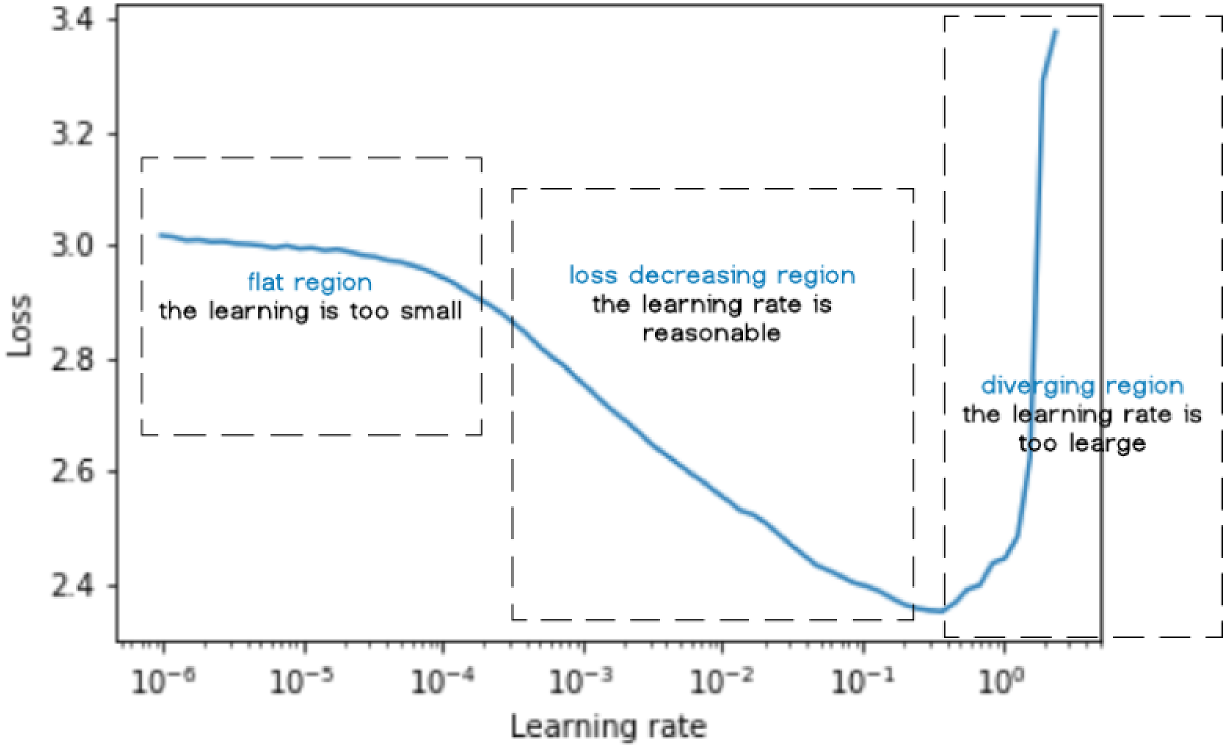
Too large of a learning rate causes drastic updates which lead to divergent behaviors

Credit:. https://www.jeremyjordan.me/nn-learning-rate/ i

Linear Regression (Single Feature):

Starting from very small values, evaluate the loss of the model using a validation set

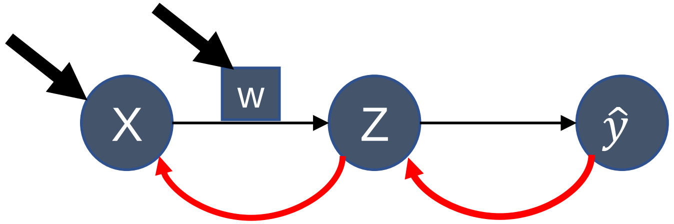
$$w \rightarrow w - \alpha \frac{d(MSE)}{dw} \qquad b \rightarrow b - \alpha \frac{d(MSE)}{db}$$



Credit:. https://medium.com/deep-learning-hk/some-techniques-in-deep-learning-optimization-1-learning-rate-b4669d5bb568#:~:text=Learning%20Rate%20Range%20Test,-

Linear Regression (Single Feature):

In Summary:



1. **Random Initialisation:** Neural Network assigns random values to W and b.

2. **Forward Propagation**: The Network Calculates:

$$Z = wx + b \; ; \; \hat{y} = Z$$

3. **Backpropagation**: The network calculates derivatives to minimise the loss function

$$\frac{d(MSE)}{dw} = \frac{d(MSE)}{d\hat{y}} \frac{d\hat{y}}{dZ} \frac{dZ}{dw} \qquad \frac{d(MSE)}{db} = \frac{d(MSE)}{d\hat{y}} \frac{d\hat{y}}{dZ} \frac{dZ}{db}$$

4. **Update Parameters**: Using the gradients, parameters (w,b) are updated

$$w \rightarrow w - \alpha \frac{d(MSE)}{dw} \qquad b \rightarrow b - \alpha \frac{d(MSE)}{db}$$

5. **Repeat 2-4 for a set number of “Epochs”**

Linear Regression (Multiple Features)

Linear Regression:

What if there are multiple features?

- Predicting House Price based on House Size (x_1), Number of Toilets (x_2),…,(x_n)

Linear Regression attempts to fit data into the form:

$$y = w_1x_1 + w_2x_2 + w_3x_3 + \cdots + w_nx_n + \boldsymbol{b}$$

Let W be a vector containing all weightings and X be a vector containing all features:

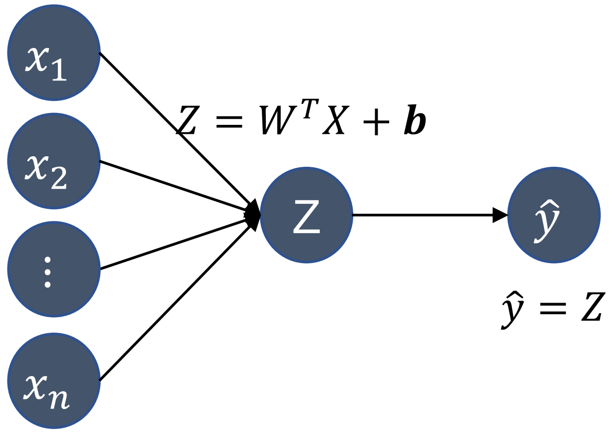
$$W = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix}; X = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

In terms of Matrix Multiplication:

$$Y = W^T X + \boldsymbol{b}$$

Linear Regression Example:

This can be illustrated as a neural network:



Forward Propagation:

$$w_1 x_1 + w_2 x_2 + w_3 x_3 + \cdots + w_n x_n + \boldsymbol{b}$$

Just like before, the network will first assign random values to $w_1, w_2, \ldots, w_n, b$

Credit: deeplearning.ai

Linear Regression Example:

$w_1, w_2, \dots, w_n, b$ need to be modified

$$\frac{d(MSE)}{dw_1}; \frac{d(MSE)}{dw_2}; \dots; \frac{d(MSE)}{dw_n}; \frac{d(MSE)}{db}$$

Follow similar workings to single variable linear regression:

$$\frac{d(MSE)}{dw_i} = \frac{d(MSE)}{d\hat{y}} \frac{d\hat{y}}{dZ} \frac{dZ}{dw_i}$$

$$\frac{d(MSE)}{dw_i} = \frac{d(MSE)}{d\hat{y}} \frac{d\hat{y}}{dZ} \frac{dZ}{dw_i} = -\frac{2}{m} x_i \sum_{k=1}^{m} (y_k - \hat{y}_k)$$

$$\frac{d(MSE)}{db} = \frac{d(MSE)}{d\hat{y}} \frac{d\hat{y}}{dZ} \frac{dZ}{db} = -\frac{2}{m} \sum_{k=1}^{m} (y_k - \hat{y}_k)$$

Linear Regression Example:

With the gradients and the learning rate, update the parameters

$$w_1 \rightarrow w_1 - \alpha \frac{d(MSE)}{dw_1} \qquad w_2 \rightarrow w_2 - \alpha \frac{d(MSE)}{dw_2} \qquad w_n \rightarrow w_n - \alpha \frac{d(MSE)}{dw_n} \qquad b \rightarrow b - \alpha \frac{d(MSE)}{db}$$

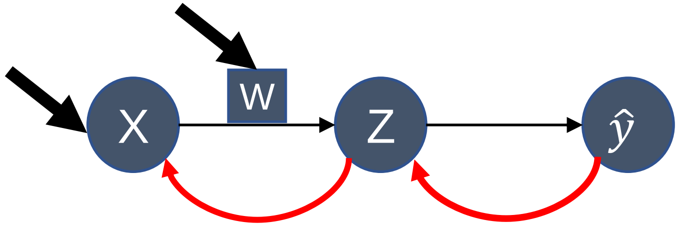
In terms of vectors:

$$\begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix} \rightarrow \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix} - \alpha \begin{pmatrix} \frac{d(MSE)}{dw_1} \\ \frac{d(MSE)}{dw_2} \\ \vdots \\ \frac{d(MSE)}{dw_n} \end{pmatrix} \qquad if\ dW = \begin{pmatrix} \frac{d(MSE)}{dw_1} \\ \frac{d(MSE)}{dw_2} \\ \vdots \\ \frac{d(MSE)}{dw_n} \end{pmatrix} \equiv \nabla MSE$$

$$\mathrm{W} \rightarrow W - \alpha dW$$

Linear Regression

In Summary:



1. **Random Initialisation:** Neural Network assigns random values to W and b.

2. **Foreward Propagation**: The Network Calculates:

$$Z = W^T X + b \; ; \; \hat{y} = Z$$

3. **Backpropagation**: The network calculates derivatives to minimise the loss function

$$\frac{d(MSE)}{dw_i} = \frac{d(MSE)}{d\hat{y}} \frac{d\hat{y}}{dZ} \frac{dZ}{dw_i} \qquad \frac{d(MSE)}{db} = \frac{d(MSE)}{d\hat{y}} \frac{d\hat{y}}{dZ} \frac{dZ}{db}$$

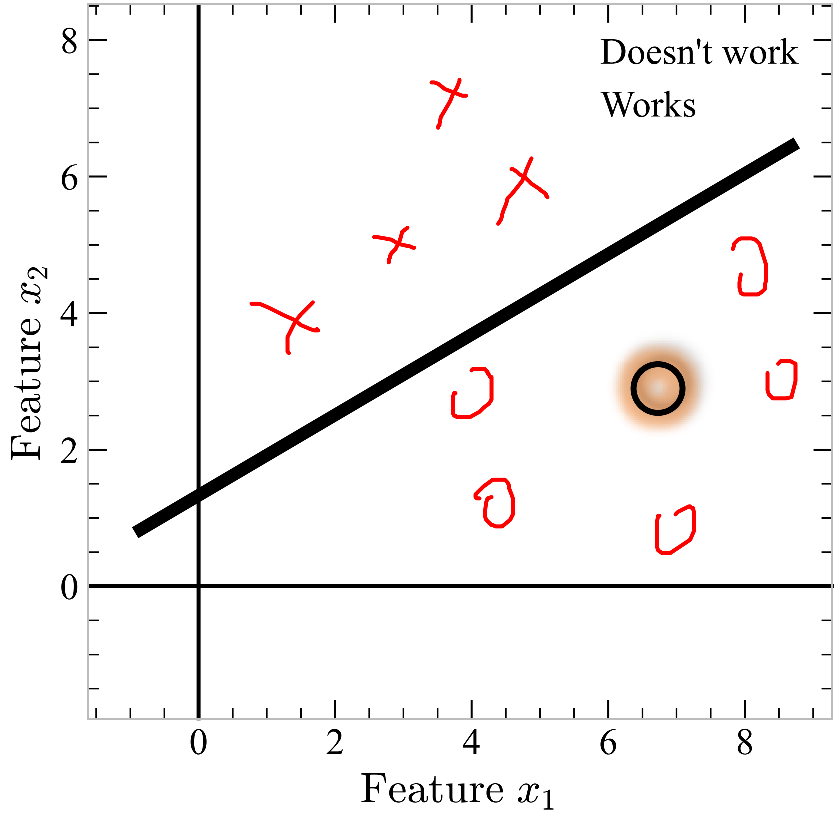
4. **Update Parameters**: Using the gradients, parameters (w,b) are updated

$$W \rightarrow W - \alpha dW \qquad b \rightarrow b - \alpha \frac{d(MSE)}{db}$$

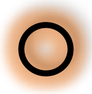
5. **Repeat 2-4 for a set number of “Epochs”**

Logistic Regression

Logistic Regression: Recap



We want to predict the probability that the engine will work.



Will this engine work?

Do this using **Logistic Regression**

Linear regression or Logistic Regression?

We want to predict a probability that is either 0 = 0% or 1 = 100%.

Linear regression gave us a value that **is not between 0 and 1**

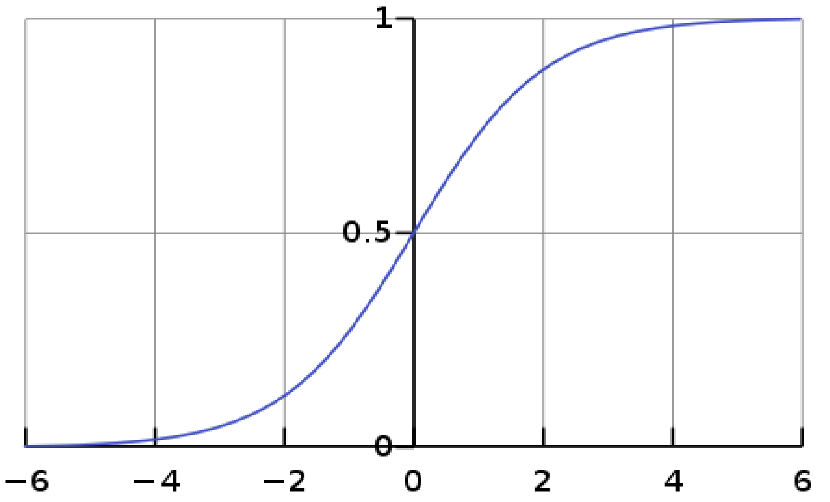
$$\hat{y} = w_1x_1 + w_2x_2 + w_3x_3 + \cdots + w_nx_n + \boldsymbol{b} = W^TX + b$$

Wrap the result of the linear regression with another function.

- In a neural network, this is called the **Activation Function**

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

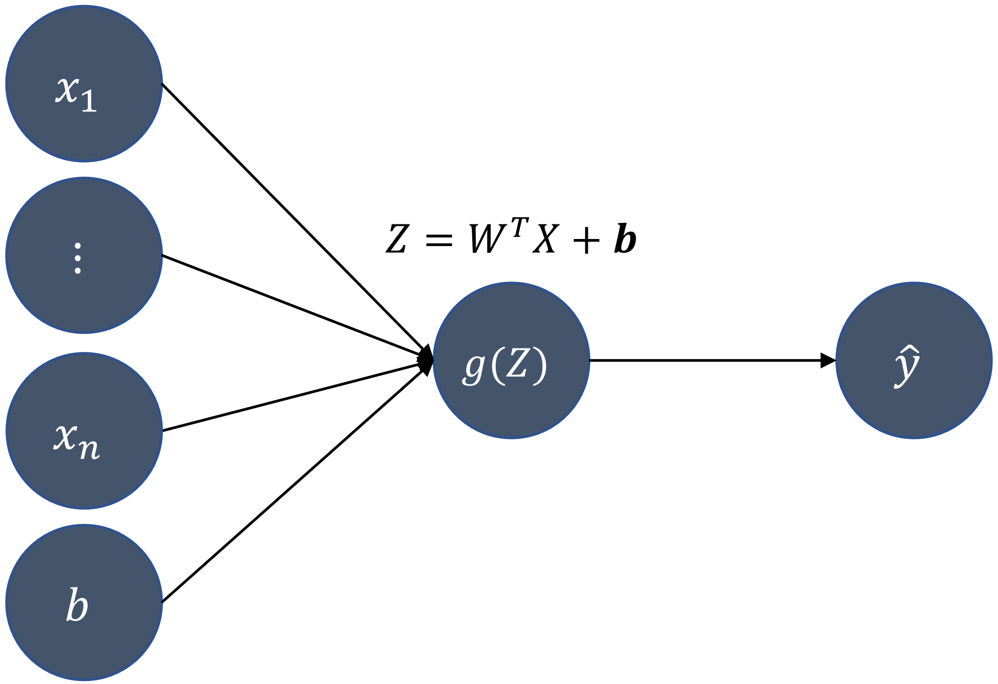
Logistic Function



Logistic Regression as a NN

We can represent Logisitic Regression as a Simple Neural Network

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

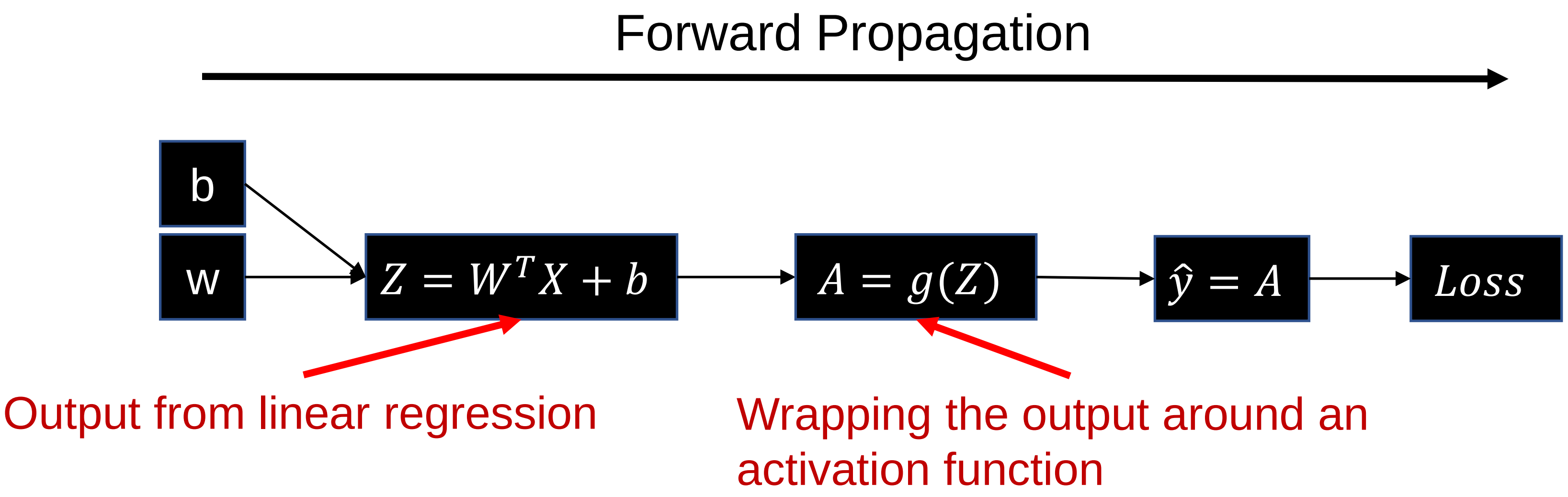


Logistic regression: the model

Logistic regression model:

Sigmoid function:

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



The probability that the engine works is represented by g(Z)

We still need to define a Loss function

Logistic Regression Loss Function

A measure of how "badly" the model performed.

- Logistic Regression uses a "Cross Entropy" Loss Function

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

y_i Label for the i-th example

$\hat{y}_i$ Prediction for the i-th example

Credit: https://towardsdatascience.com/step-by-step-guide-to-building-your-own-neural-network-from-scratch-df64b1c5ab6e

Loss Function

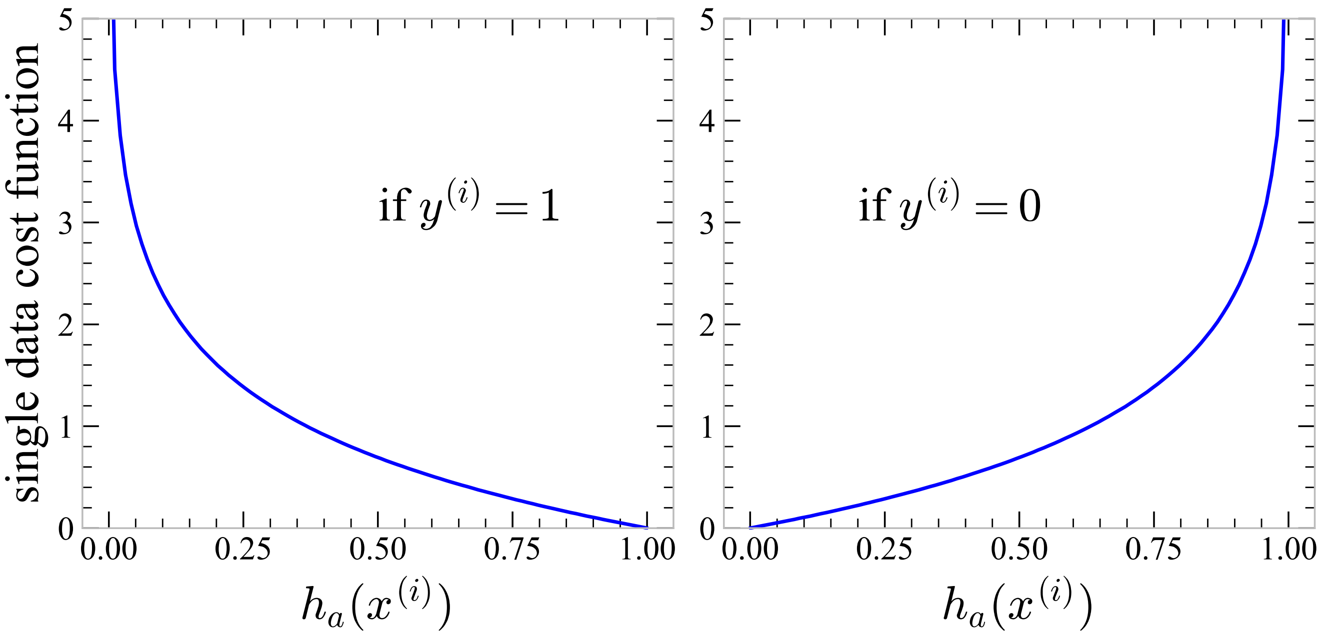
$$J = -\underbrace{\frac{1}{m}\sum_{i=1}^{m}}_{\text{Average over all } m \text{ examples}} \underbrace{y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)}_{\text{Loss for one example}}$$

Note that: $0 \leq \hat{y}_i \leq 1$

- Meaning both $\log \hat{y}_i \;\; and \;\; \log(1 - \hat{y}_i)$ are negative
- A larger value of J should mean worse performance: minus sign added

Loss Function

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



Backpropagation in Logistic Regression

$$b,\ W \rightarrow Z = W^T X + b \rightarrow A = g(Z) \rightarrow \hat{y} = A \rightarrow J(\hat{y})$$

We need to compute:

$$\frac{d(J)}{dw_1};\frac{d(J)}{dw_2};\dots;\frac{d(J)}{dw_n};\frac{d(J)}{db}$$

$$\frac{d(J)}{dw_i} = \frac{d(J)}{d\hat{y}}\frac{d\hat{y}}{dA}\frac{dA}{dZ}\frac{dZ}{dw_i}$$

$$\frac{d(J)}{db} = \frac{d(J)}{d\hat{y}}\frac{d\hat{y}}{dA}\frac{dA}{dZ}\frac{dZ}{db}$$

$$\begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix} \rightarrow \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix} - \alpha \begin{pmatrix} \frac{d(J)}{dw_1} \\ \frac{d(J)}{dw_2} \\ \vdots \\ \frac{d(J)}{dw_n} \end{pmatrix} \Longrightarrow$$

$$\mathrm{W} \rightarrow W - \alpha dW$$

$$b \rightarrow b - \alpha \frac{d(MSE)}{db}$$

Loss Function Derivative

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

Determine $\frac{dJ}{d\hat{y}}$: Let $L(y_i, \hat{y}_i) = y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i)$

$$\frac{dJ}{d\hat{y}} = \frac{d}{d\hat{y}}\left(-\frac{1}{m}\sum_{i=1}^{m} L(y_i, \hat{y}_i)\right) \longrightarrow \frac{dJ}{d\hat{y}} = -\frac{1}{m}\sum_{i=1}^{m} \frac{d}{d\hat{y}}\left(L(y_i, \hat{y}_i)\right)$$

$$\frac{d}{d\hat{y}}\left(L(y_i, \hat{y}_i)\right) = \frac{y_i}{\hat{y}_i} - \frac{(1 - y_i)}{1 - \hat{y}_i}$$

$$\boldsymbol{\frac{dJ}{d\hat{y}} = -\frac{1}{m}\sum_{i=1}^{m}\left\{\frac{y_i}{\hat{y}_i} - \frac{(1 - y_i)}{1 - \hat{y}_i}\right\}}$$

Backpropagation in Logistic Regression

$$\begin{tikzcd} b \arrow[dr] & & & & \\ W \arrow[r] & Z = W^T X + b \arrow[r] & A = g(Z) \arrow[r] & \hat{y} = A \arrow[r] & J(\hat{y}) \end{tikzcd}$$

Now compute the derivatives

$$\frac{d(J)}{dw_i} = \frac{d(J)}{d\hat{y}} \frac{d\hat{y}}{dA} \frac{dA}{dZ} \frac{dZ}{dw_i} \qquad \frac{d(J)}{db} = \frac{d(J)}{d\hat{y}} \frac{d\hat{y}}{dA} \frac{dA}{dZ} \frac{dZ}{db}$$

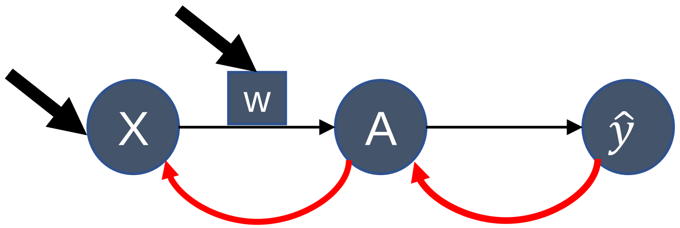
$$\frac{dJ}{d\hat{y}} = -\frac{1}{m} \sum_{i=1}^{m} \left\{ \frac{y_i}{\hat{y}_i} - \frac{(1 - y_i)}{1 - \hat{y}_i} \right\} \qquad \frac{d\hat{y}}{dA} = 1; \frac{dA}{dZ} = A(1-A); \frac{dZ}{dw_i} = x_i; \frac{dZ}{db} = 1$$

$$\frac{d(J)}{dw_i} = -\frac{1}{m} A(1-A) x_i \sum_{k=1}^{m} \left\{ \frac{y_k}{\hat{y}_k} - \frac{(1 - y_k)}{1 - \hat{y}_k} \right\}$$

$$\frac{d(J)}{db} = -\frac{1}{m} A(1-A) \sum_{k=1}^{m} \left\{ \frac{y_k}{\hat{y}_k} - \frac{(1 - y_k)}{1 - \hat{y}_k} \right\}$$

Logistic Regression

In Summary:



1. **Random Initialisation:** Neural Network assigns random values to W and b.

2. **Forward Propagation**: The Network Calculates:

$$Z = W^T X + b\ ; A = g(Z);\ \hat{y} = A$$

3. **Backpropagation**: The network calculates derivatives to minimise the loss function

$$\frac{d(J)}{dw_i} = \frac{d(J)}{d\hat{y}}\frac{d\hat{y}}{dA}\frac{dA}{dZ}\frac{dZ}{dw_i} \qquad \frac{d(MSE)}{db} = \frac{d(MSE)}{d\hat{y}}\frac{d\hat{y}}{dA}\frac{dA}{dZ}\frac{dZ}{db}$$

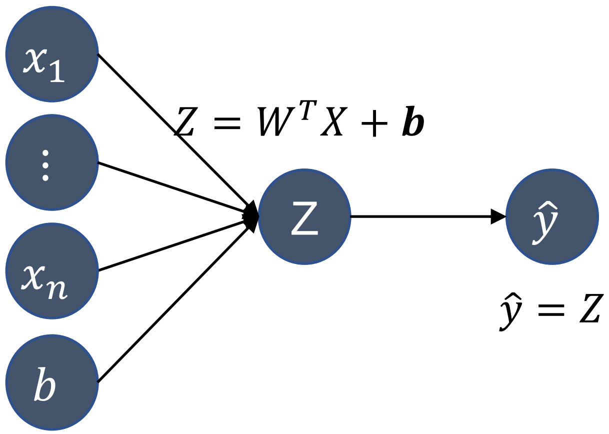
4. **Update Parameters**: Using the gradients, parameters (w,b) are updated

$$w\ \rightarrow w - \alpha dW \qquad b \rightarrow b - \alpha\frac{d(MSE)}{db}$$

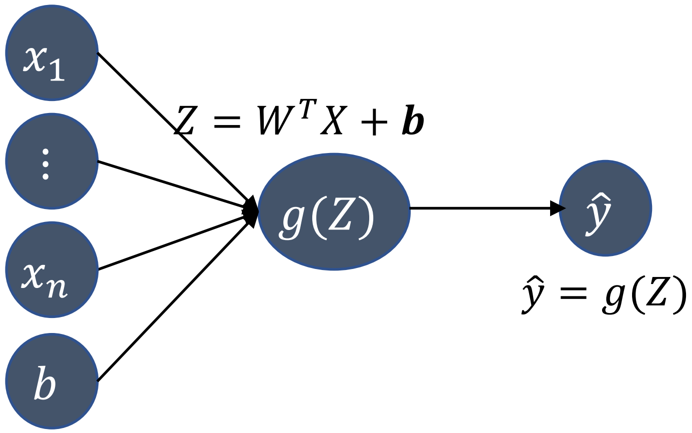
5. **Repeat 2-4 for a set number of "Epochs"**

What about General NNs?

Linear Regression



Logistic Regression (Classification)



See that a neural network can function as a regressor or a classifier

It only depends on which activation function we are using!

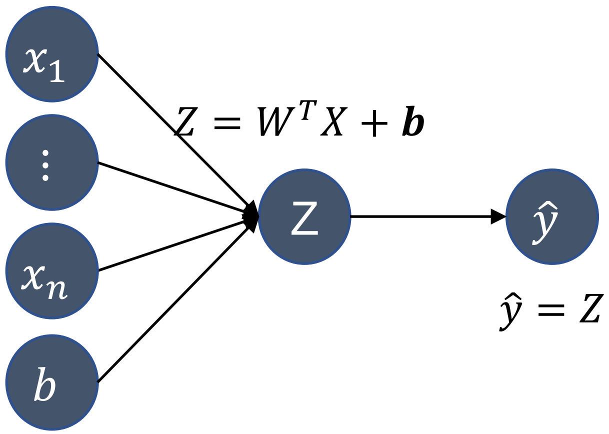
Credit: https://medium.com/@marcellvollmer/how-to-make-it-simple-to-explain-ai-ml-dl-and-data-science-a49e54d54a12

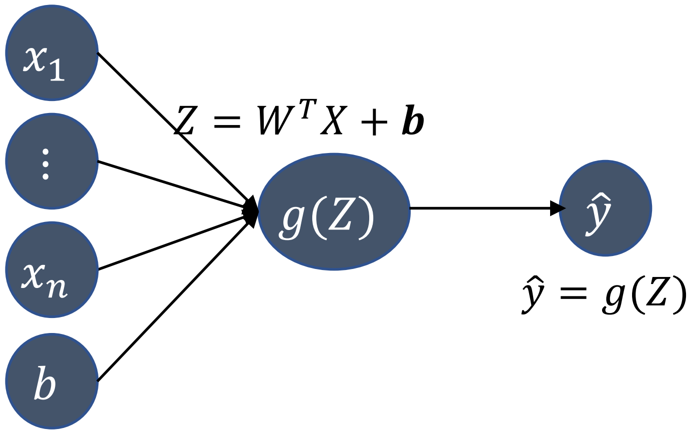
General NNs

What about General NNs?

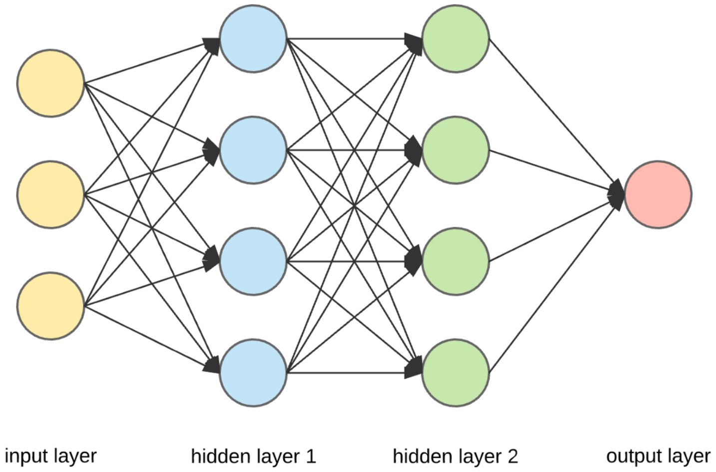
Until now, we only considered problems where there are only 1 neuron and 1 hidden layer.

- This is also known as a **perceptron**



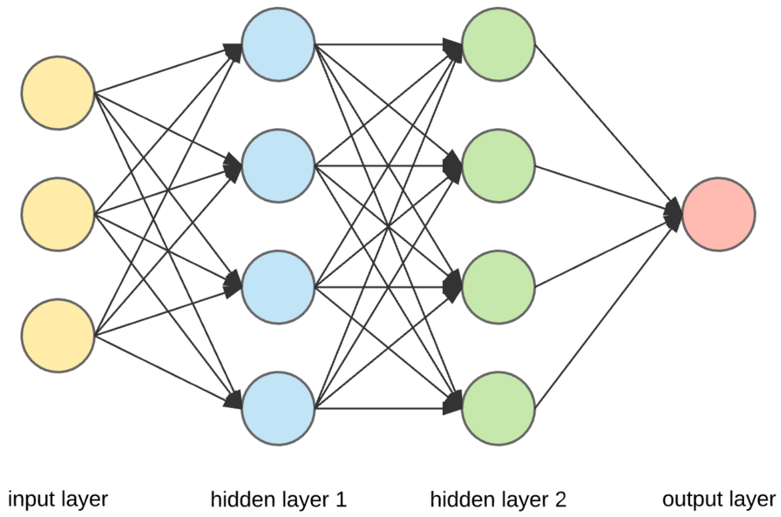


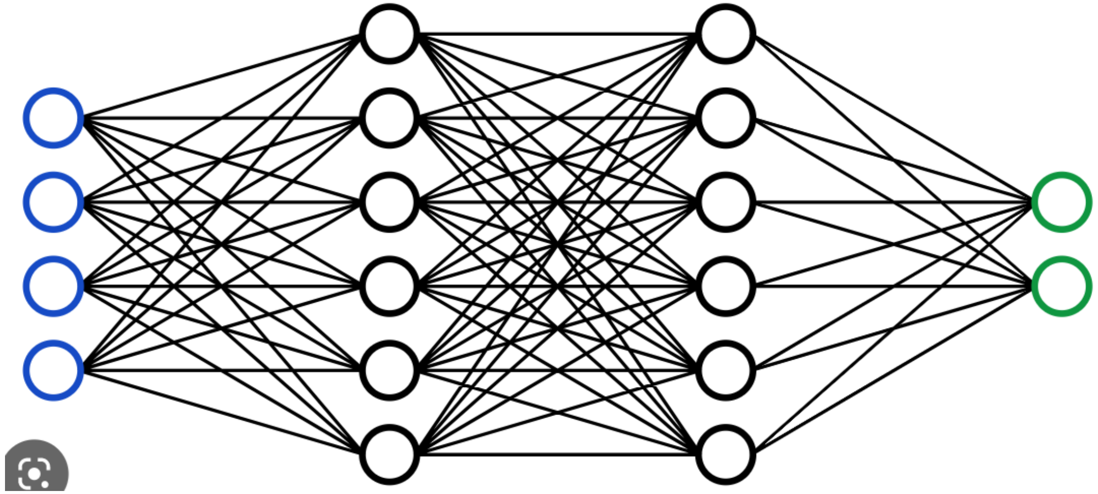
What about general neural networks?



Credit: https://medium.com/@marcellvollmer/how-to-make-it-simple-to-explain-ai-ml-dl-and-data-science-a49e54d54a12

Some (More) Notation



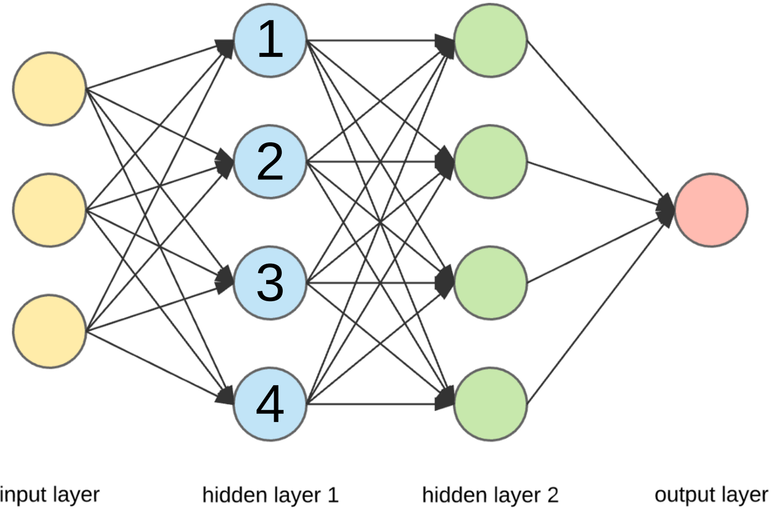


- **Input, Label Matrix: $\boldsymbol{X}, \boldsymbol{Y}$**
- **Prediction Matrix: $\hat{\boldsymbol{y}}$**
- Number Of Layers: All layers excluding input
- Input Layer: Layer 0
- **Number of examples: m**
- Input, Label Size: n_x, n_y
- Values in layer L is denoted by: $W^{[L]}$
- Number of neurons in (hidden) layer L: $n_h^{[L]}$

Credit: https://medium.com/@marcellvollmer/how-to-make-it-simple-to-explain-ai-ml-dl-and-data-science-a49e54d54a12

General NNs

Consider Input Layer to hidden layer 1 and number each hidden neuron 1 to 4 with some activation function $g^{[1]}(x)$



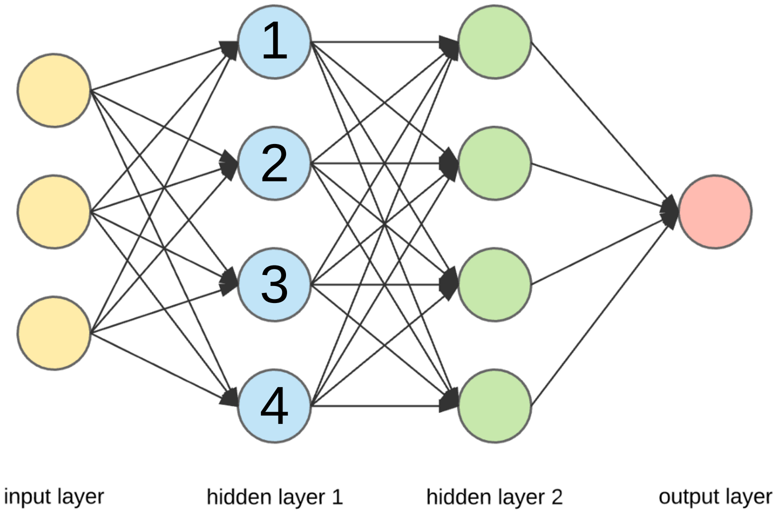
$$A_1^{[1]} = g^{[1]}\left(Z_1^{[1]}\right); where\ Z_1^{[1]} = w_{1,1}x_1 + w_{2,1}x_2 + w_{3,1}x_3 + b_1 = W_1^{[1]T}X + b_1$$

$$A_2^{[1]} = g^{[1]}\left(Z_2^{[1]}\right); where\ Z_2^{[1]} = w_{1,2}x_1 + w_{2,2}x_2 + w_{3,2}x_3 + b_2 = W_2^{[1]T}X + b_2$$

$$A_3^{[1]} = g^{[1]}\left(Z_3^{[1]}\right); where\ Z_3^{[1]} = w_{1,3}x_1 + w_{2,3}x_2 + w_{3,3}x_3 + b_3 = W_3^{[1]T}X + b_3$$

$$A_4^{[1]} = g^{[1]}\left(Z_4^{[1]}\right); where\ Z_4^{[1]} = w_{1,4}x_1 + w_{2,4}x_2 + w_{3,4}x_3 + b_4 = W_4^{[1]T}X + b_4$$

Consider Input Layer to hidden layer 1 and number each hidden neuron 1 to 4 with some activation function g(x)

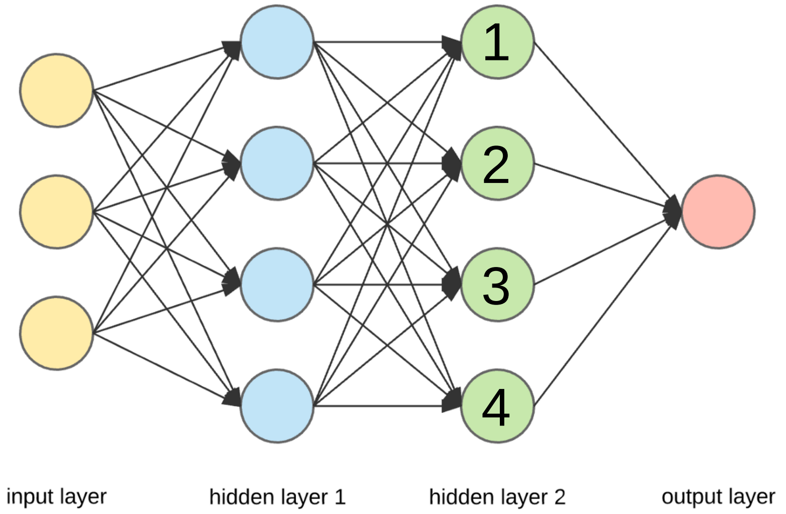


$$A^{[1]} = \begin{bmatrix} A_1^{[1]} \\ A_2^{[1]} \\ A_3^{[1]} \\ A_4^{[1]} \end{bmatrix} = \begin{bmatrix} g^{[1]}(W_1^{[1]T} X + b_1^{[1]}) \\ g^{[1]}(W_2^{[1]T} X + b_2^{[1]}) \\ g^{[1]}(W_3^{[1]T} X + b_3^{[1]}) \\ g^{[1]}(W_4^{[1]T} X + b_4^{[1]}) \end{bmatrix}$$

Now $A^{[1]}$ acts as the input for hidden layer 2

General NNs

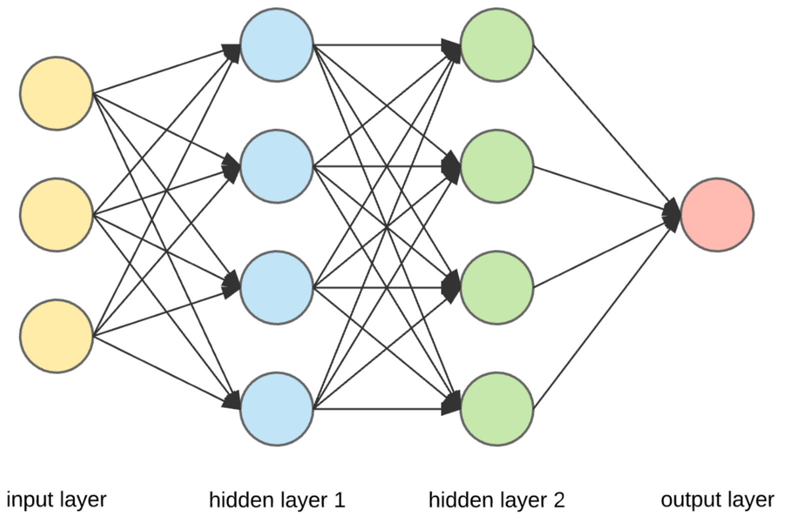
Now consider Hidden Layer 1 to Hidden Layer 2. Follow similar logic to previous slides, using $g^{[2]}(x)$ as the activation function



$$A^{[2]} = \begin{bmatrix} A_1^{[2]} \\ A_2^{[2]} \\ A_3^{[2]} \\ A_4^{[2]} \end{bmatrix} = \begin{bmatrix} g^{[2]}(W_1^{[2]T} A_1^{[1]} + b_1^{[2]}) \\ g^{[2]}(W_2^{[2]T} A_2^{[1]} + b_2^{[2]}) \\ g^{[2]}(W_3^{[2]T} A_3^{[1]} + b_3^{[2]}) \\ g^{[2]}(W_4^{[2]T} A_4^{[1]} + b_4^{[2]}) \end{bmatrix}$$

General NNs

Finally for the output layer with activation function $g^{[3]}(x)$:



$$\hat{y} = g^{[3]}(W^{[3]T} A^{[2]} + b^{[3]})$$

A general neural network will have to assign random values for: $W^{[1]}; W^{[2]}; W^{[3]}; b^{[1]}\,; b^{[2]}; b^{[3]}$

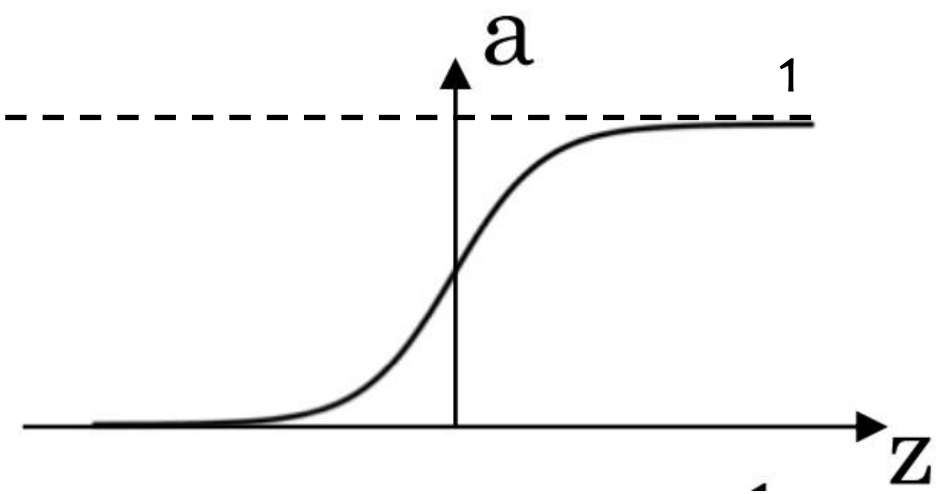
And then calculate derivatives for all parameters and then update using:

$$\mathrm{W} \rightarrow W - \alpha dW \qquad b \rightarrow b - \alpha \frac{d(Loss)}{db}$$

Activation Functions

Activation Functions

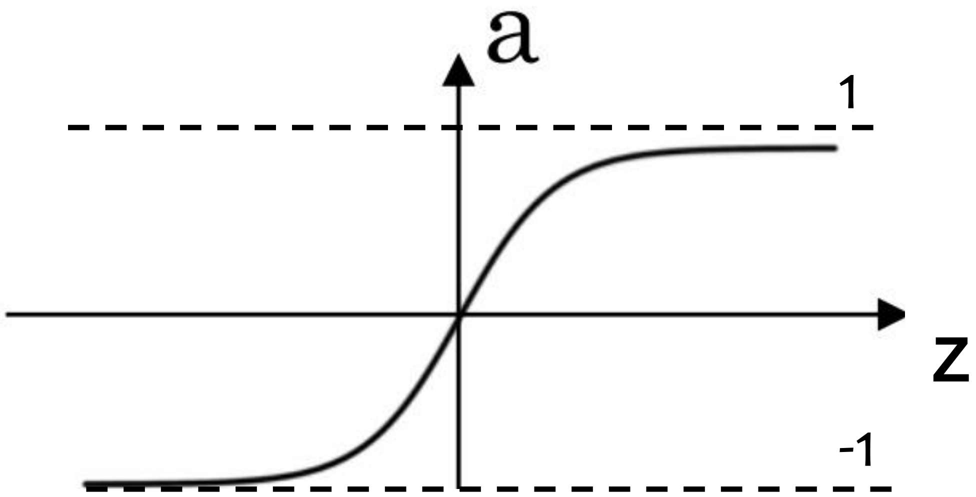
Sigmoid Function



$$g(Z) = \frac{1}{1+e^{-Z}}$$

Generally, **only** used for the output layer in a binary classification problem

$\tanh(Z)$

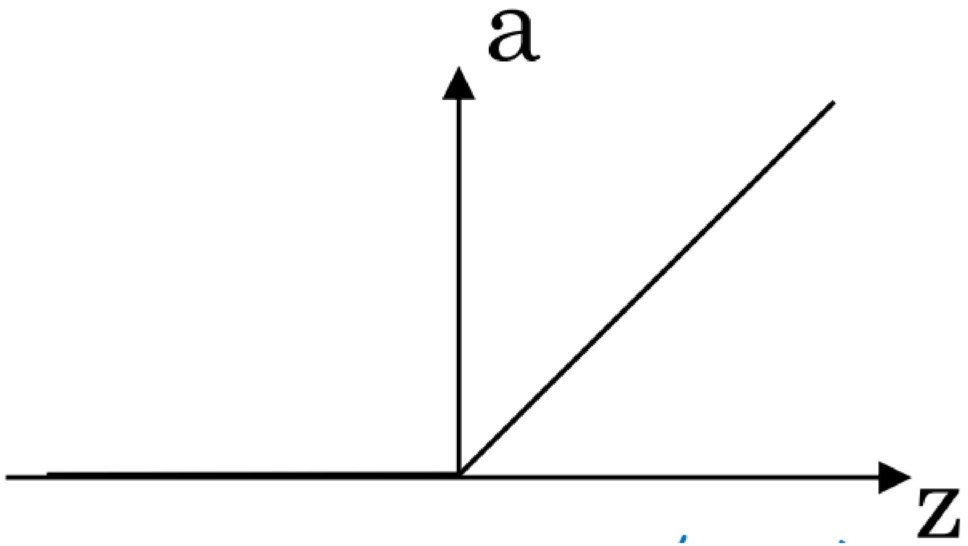


$$g(Z) = \tanh(Z) = \frac{e^{Z} - e^{-Z}}{e^{Z} + e^{-Z}}$$

Credit: deeplearning.ai

Activation Functions

ReLU

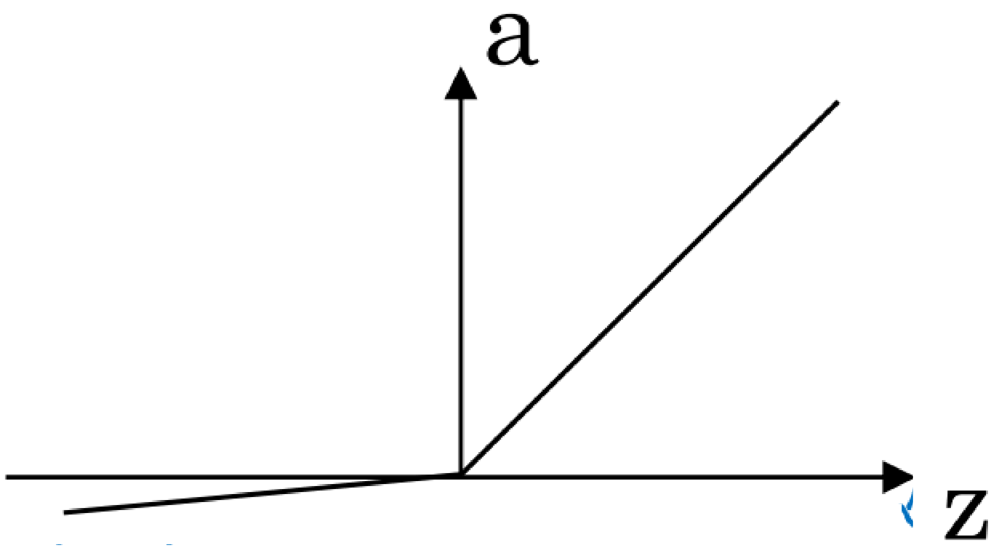


$$g(Z) = \begin{cases} Z \text{ } if \text{ } Z \geq 0 \\ 0 \text{ } if \text{ } Z < 0 \end{cases}$$

When in doubt, use ReLU

Easy to implement, closest to how a Neuron actually works

Leaky ReLU



$$g(Z) = \begin{cases} Z \text{ } if \text{ } Z \geq 0 \\ \beta Z \text{ } if \text{ } Z < 0 \end{cases}$$

Tends to perform better than just ReLU

But 1 more hyperparameter to tune

Credit: deeplearning.ai

Questions?

Extra reading

Learning rate does not always have to be constant

- Optimisers such as “Adam” or “RMSProp” slowly change the value of the learning rate
- https://medium.com/analytics-vidhya/a-complete-guide-to-adam-and-rmsprop-optimizer-75f4502d83be

To prevent overfitting: you can modify the Loss function using L1 or L2 Regularisation

- https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c

Matrix Calculus: This lecture used a simplified version of Neural networks to keep the maths simple. In reality, Matrix Calculus is needed.

- https://explained.ai/matrix-calculus/

Credit: deeplearning.ai