Deep Learning
Foundations and Algorithms

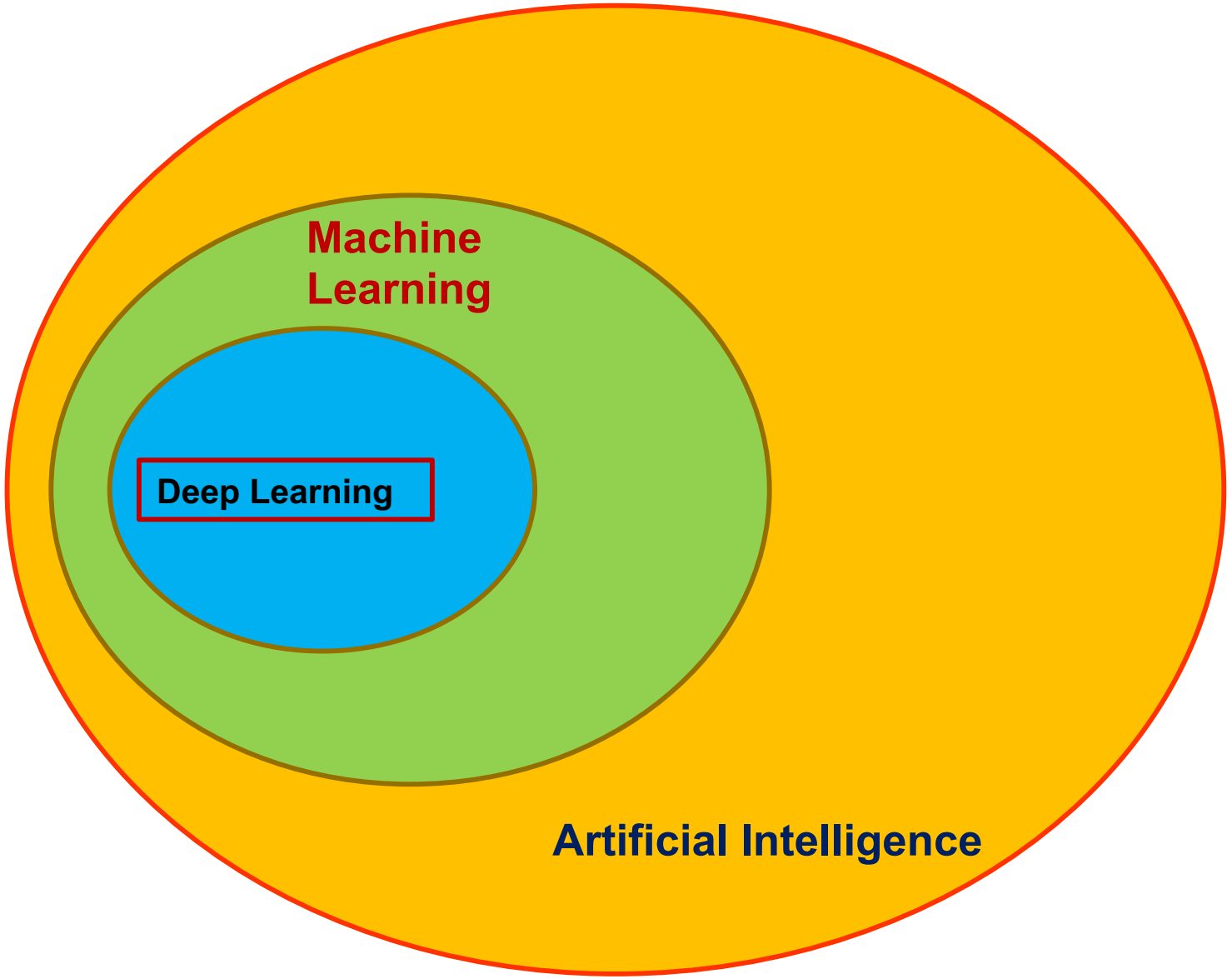
ADMSI -- Talent Sprint – Module 4.1A

Balaji Srinivasan
Mechanical Engineering
IIT-Madras

Topics for Today

- Recap + History of Neural Networks (Biological inspiration)
- Gradient Descent → Linear Regression
  → Logistic Regression
- Multiclass Classification
- Neural Networks (FCN, MLP)
  Next week – Training (Back prop)
  – Nonlinearities
  – Variants of GD
  – Regularization

Recall – The AI Venn diagram



Let us now look at Deep Learning through some simple examples…

A fundamental "trick" in DL

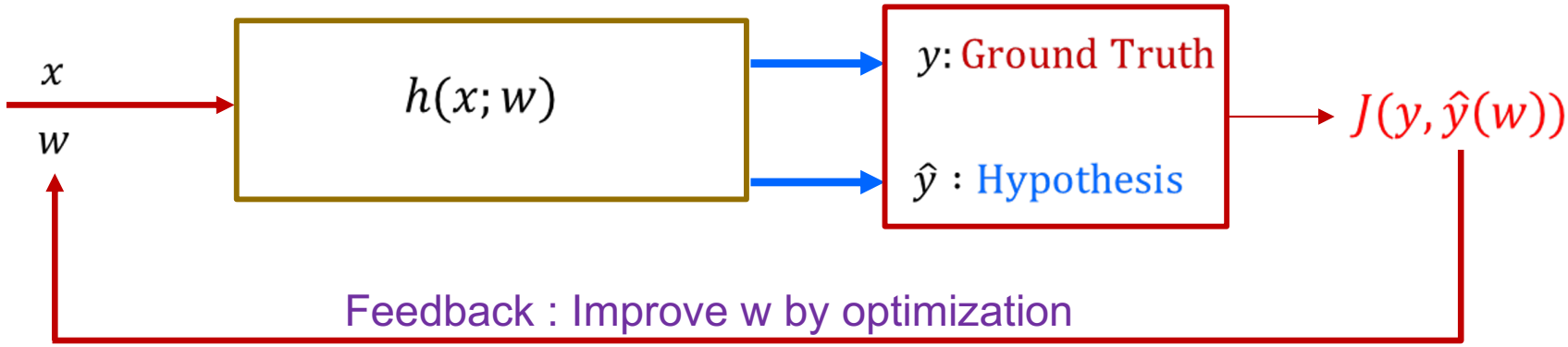
- All problems are data, all solutions are functions/maps

- Cognitive tasks -- Humans get sensory inputs as qualia
  - We must convert these qualitative inputs into **numbers** – Input Vectors
  - Similarly, outputs that humans give must also be converted into numbers – Output/Target vectors

- Determining appropriate inputs and outputs for a machine learning task is an essential part of the process

- Often the "Learning Task" is learning the mapping from input to output.

Learning the parameters via feedback



- To learn the parameters (given x,y find w), we follow this paradigm
- Collect lots of data pairs $(\text{Input Vector}, \text{Output Vector}) = (\boldsymbol{x}, \boldsymbol{y})$
- Guess for the form of the hypothesis function $h(x;w)$
  - Example : $h(x;w) = w_0 + w_1 x_1 + w_2 x_2$
- For an arbitrary guess for w
  - We will get some $\hat{\boldsymbol{y}} = h(\boldsymbol{x};\boldsymbol{w})$ which will not match the ground truth $\boldsymbol{y}$
- Define a cost function $J(y,\hat{y}(w))$ depending on the difference
- Find optimal w by minimizing $J(\mathrm{w})$ → Feedback process
  - By using some optimization procedure such as Gradient Descent

Summary

Two main ideas in this session

1. What enables breadth of application?

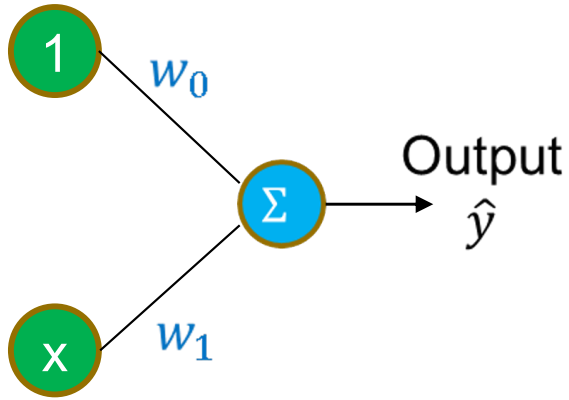
   By converting even qualitative problems into data problems and treating every rule which we wish to learn as a mathematical function.

Input —Function→ **Output**

2. How does (supervised) learning happen?
   1. We collect data as (Input, Output) pairs.
   2. We provide a hypothesis as a form of the function connecting the inputs to the outputs
   3. Learn (improve) the parameters based on mathematical feedback from data via loss function.

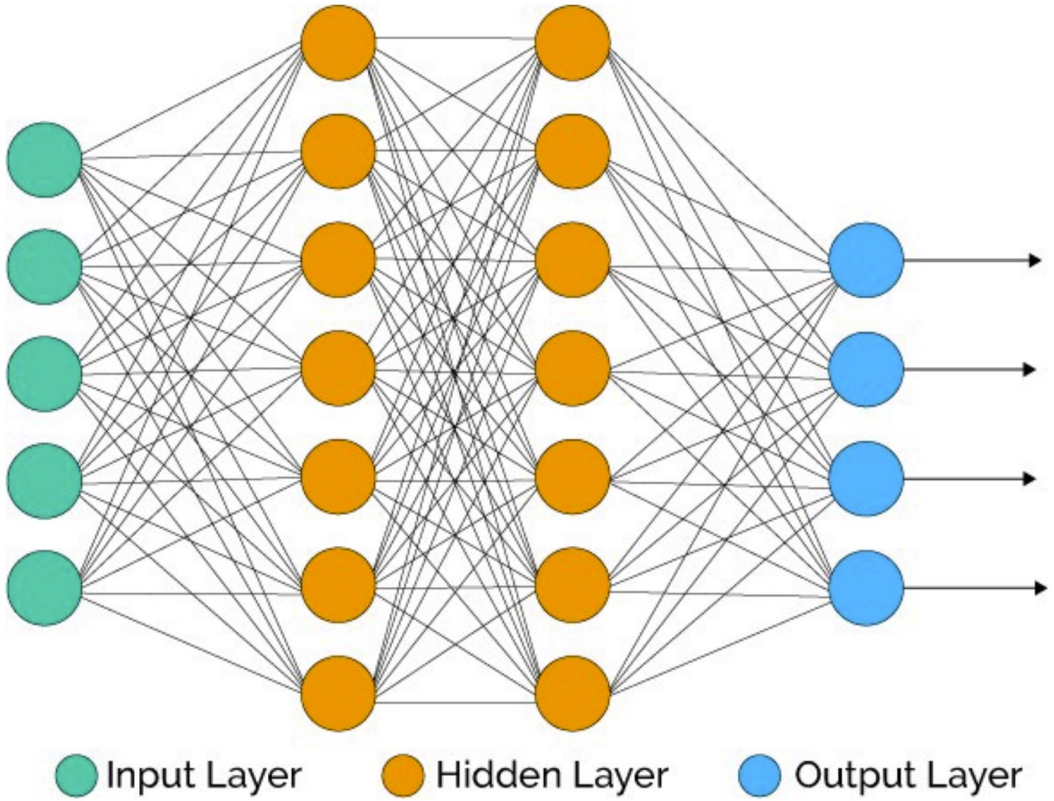
Neural Networks

Linear Model

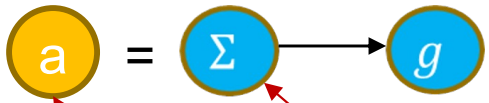


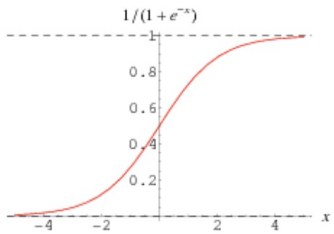
$$\hat{y} = w_0 + w_1 x = \Sigma w_i x_i$$

Neural Network Model



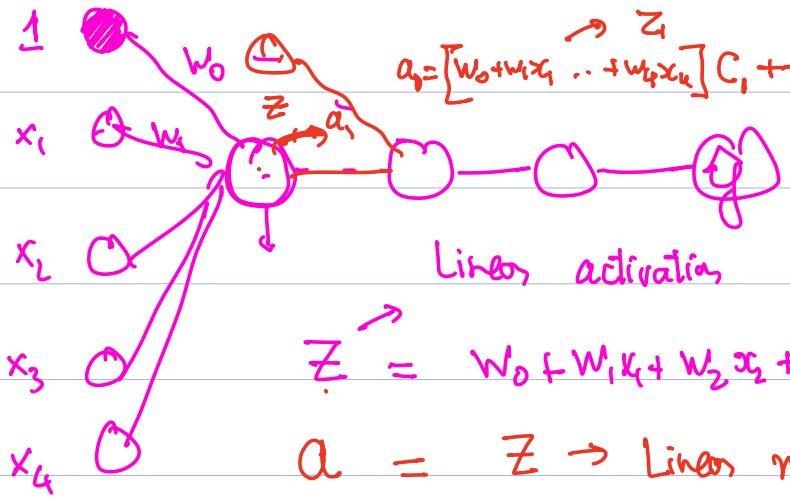
$$a = g(\Sigma w_i x_i)$$





A Neuron has a linear and a nonlinear operation

Example nonlinear function is Sigmoid(x) = $\frac{1}{1+\exp(-x)}$



Linear activation

$$z = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4$$

$a = z \rightarrow$ Linear model $\quad C_1 z + C_2$

$a = g(z)$ activation function

Examples $\quad g(z) = z^2$

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

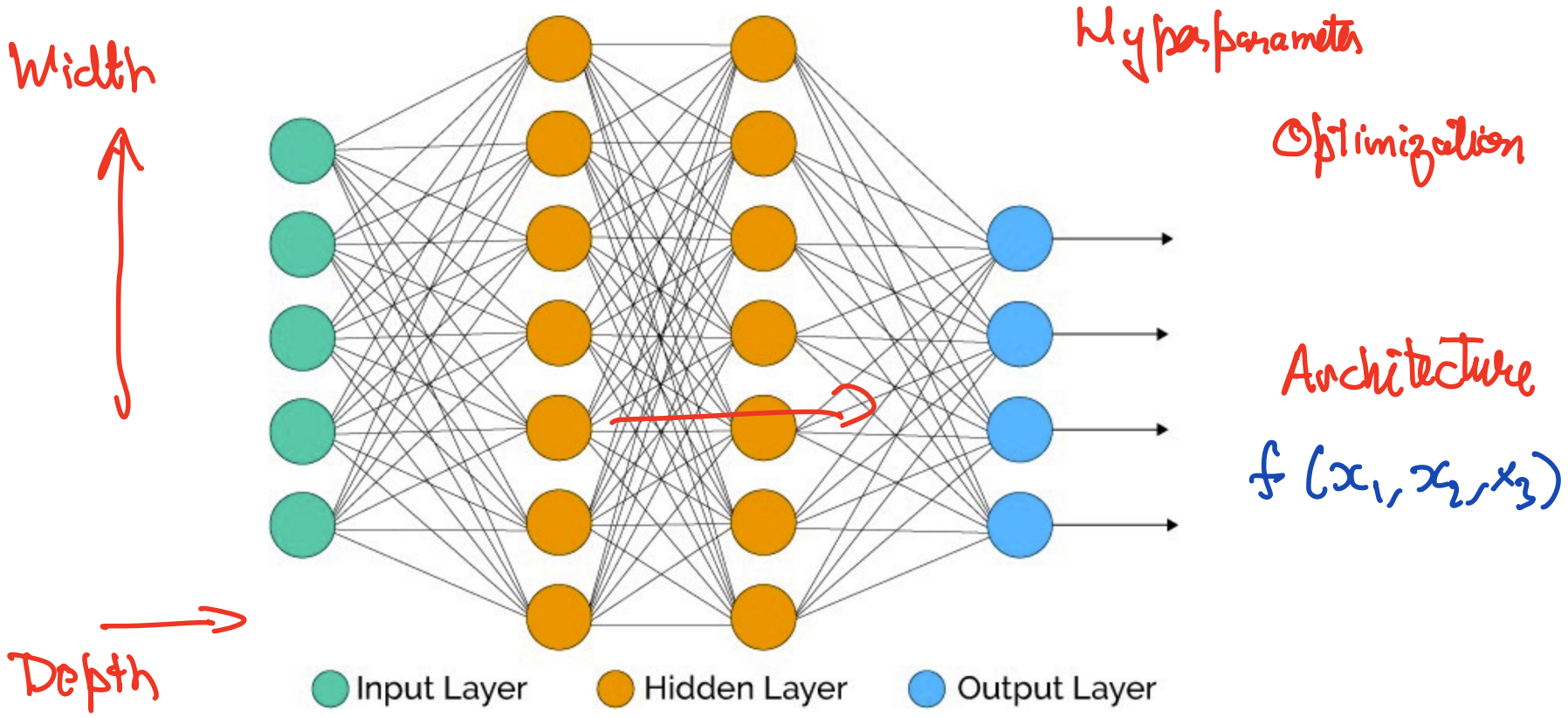
$$= \sin(z)$$

Why?

No nonlinearity $\quad a_1 = (w_1 x_1 + w_2 x_2 \dots + w_3 x_3 + w_0)$

$$a_2 = w_5 a_1 = (w_5 w_1) x_1 + (w_5 w_2) x_2 + (w_5 w_3) x_3 + (w_5 w_4) x_4 \rightarrow \text{Still linear in } x$$

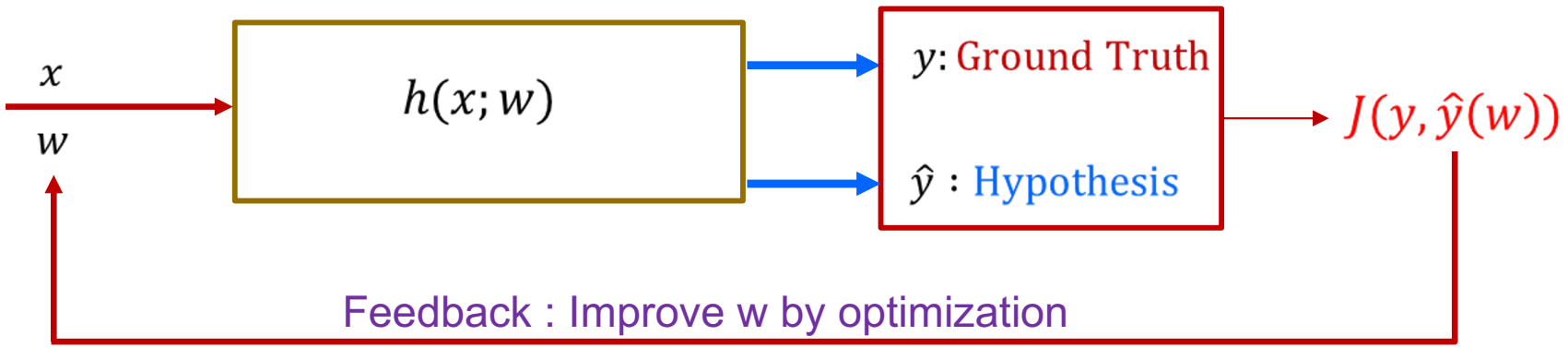
Universal Approximation Theorem



Theorem -- Given sufficient data and neurons, a Neural Network can approximate any function to any desired accuracy

Heart of modern Deep Learning – Lots of data, lots of layers => Lots of learning

Recall : Learning the parameters via feedback



To learn the parameters, we follow this paradigm

- Guess for the form of the hypothesis function $h(x;w)$

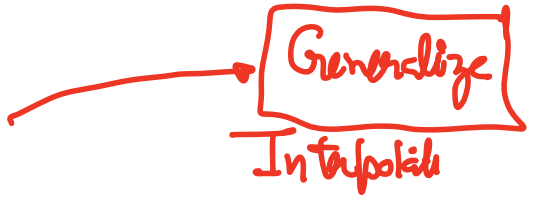
- For an arbitrary guess for w
  - We will get some $\hat{y} = h(x;w)$ which will not match the ground truth y
  - Define a cost function $J(y,\hat{y}(w))$ depending on the difference

- Find optimal w by minimizing $J(w)$ → Feedback process
  - Requires **Backpropagation** for $\frac{\partial J}{\partial w}$

Even complicated Neural Networks work in exactly the same way!

What the DL engineer must provide

- Appropriate decisions for input and output vectors (x, y)
  - Recall : All problems are data, all solutions are functions/maps
- Choosing appropriate datasets



- Some appropriate form of the forward model $\hat{y} = h(x; w)$

NN architecture

  - Example : Linear Model $\hat{y} = w_0 + w_1 x_1 + w_2 x_2 \ldots + w_n x_n$
- Form of the Loss function
  - Example : Least Squares $J(y, \hat{y}) = (y - \hat{y})^2$
- Optimization algorithm – Example: Gradient Descent
  - And associated hyperparameters such as α

Machine Learning is not magic! It requires a lot of input from the engineer

Summary of Deep Learning

- Learning algorithms involve various types of hypothesis functions $\hat{y} = h(x; w)$.

- Each of these have their own purpose and domain where they work well
  - Linear Regression – For simple polynomial regression problems
  - Logistic Regression – For two-class (binary) classification problems
  - Deep Neural Networks – For any general, non-linear problem
    - There is a theorem that assures us that sufficiently large neural network will approximate *any* function
  - Convolutional Neural Networks – For vision/image based problems
  - Recurrent Neural Networks – For sequential/time-series problems

- There are also appropriate loss functions for each.

- It is possible that you might have a better model than these for your problem. The general procedure outlined here remains the same.

Calculation in a single neuron

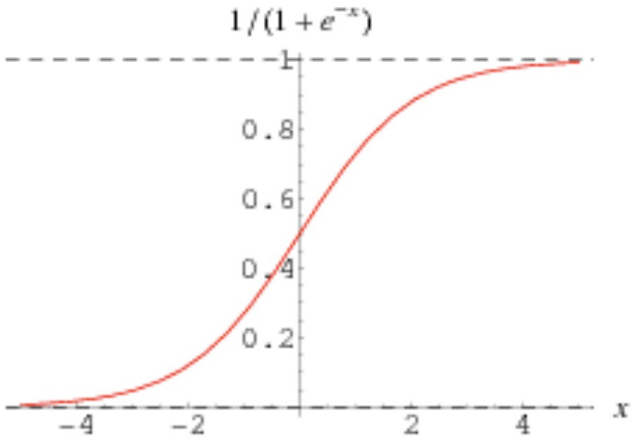
Recall : Neural Networks

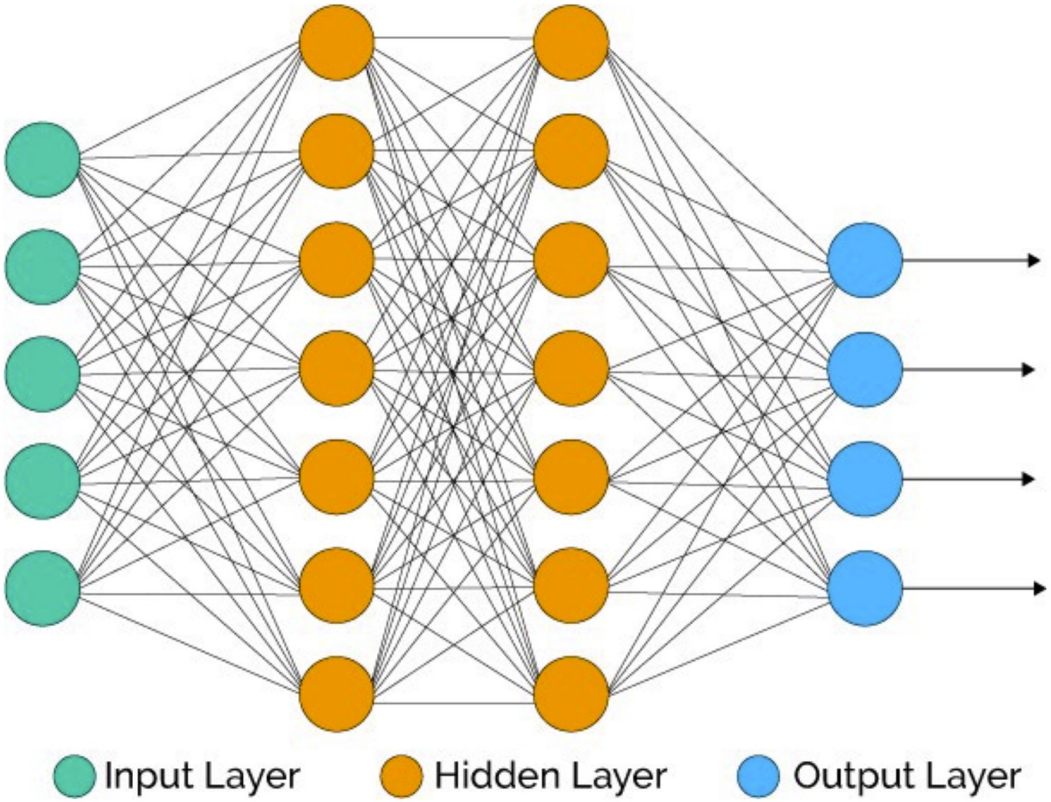
Find σ(1)

Neural Network Model

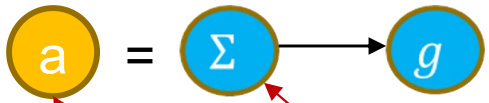
Common Activation function

$$\text{Sigmoid}(x) = \frac{1}{1+\exp(-x)}$$





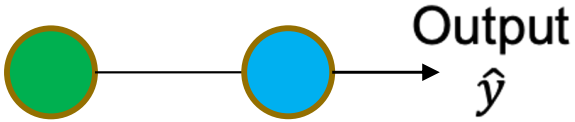
$$a = g(\Sigma w_i x_i)$$

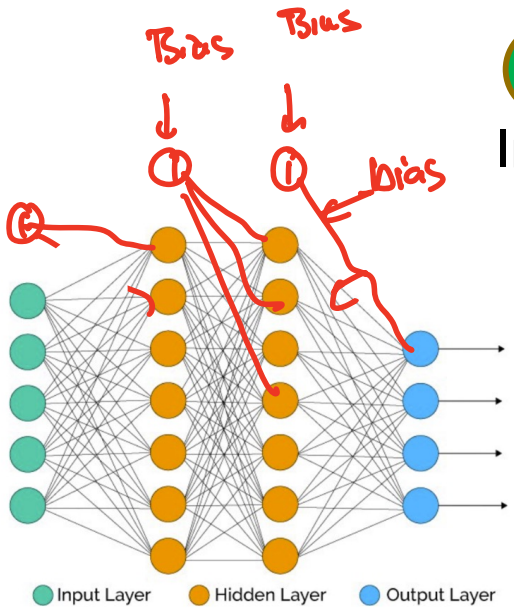


A Neuron has a linear and a nonlinear operation

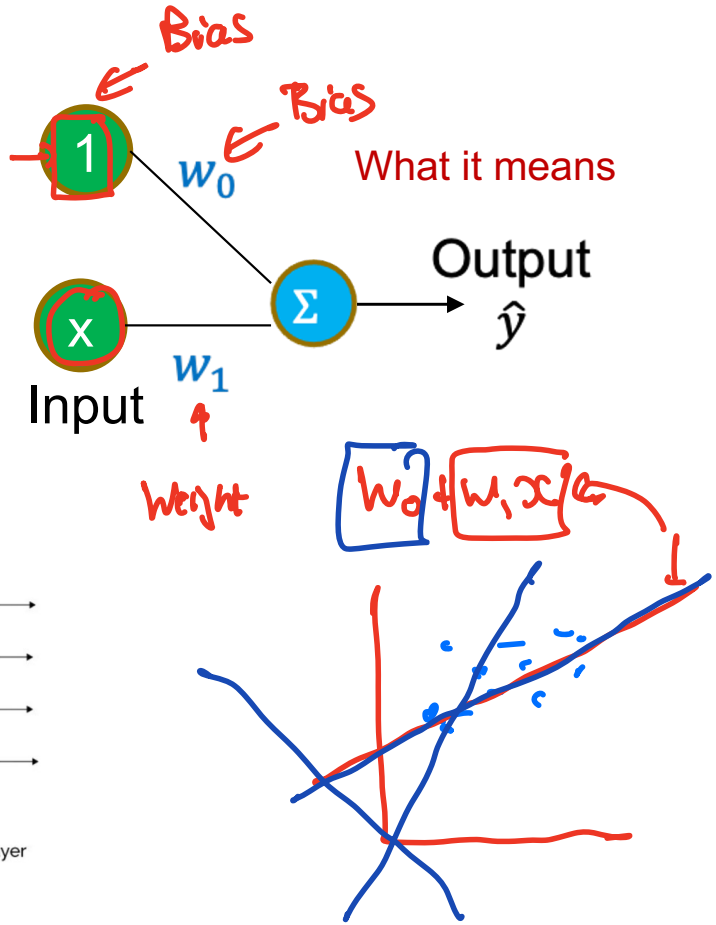
Weights and biases

What is shown



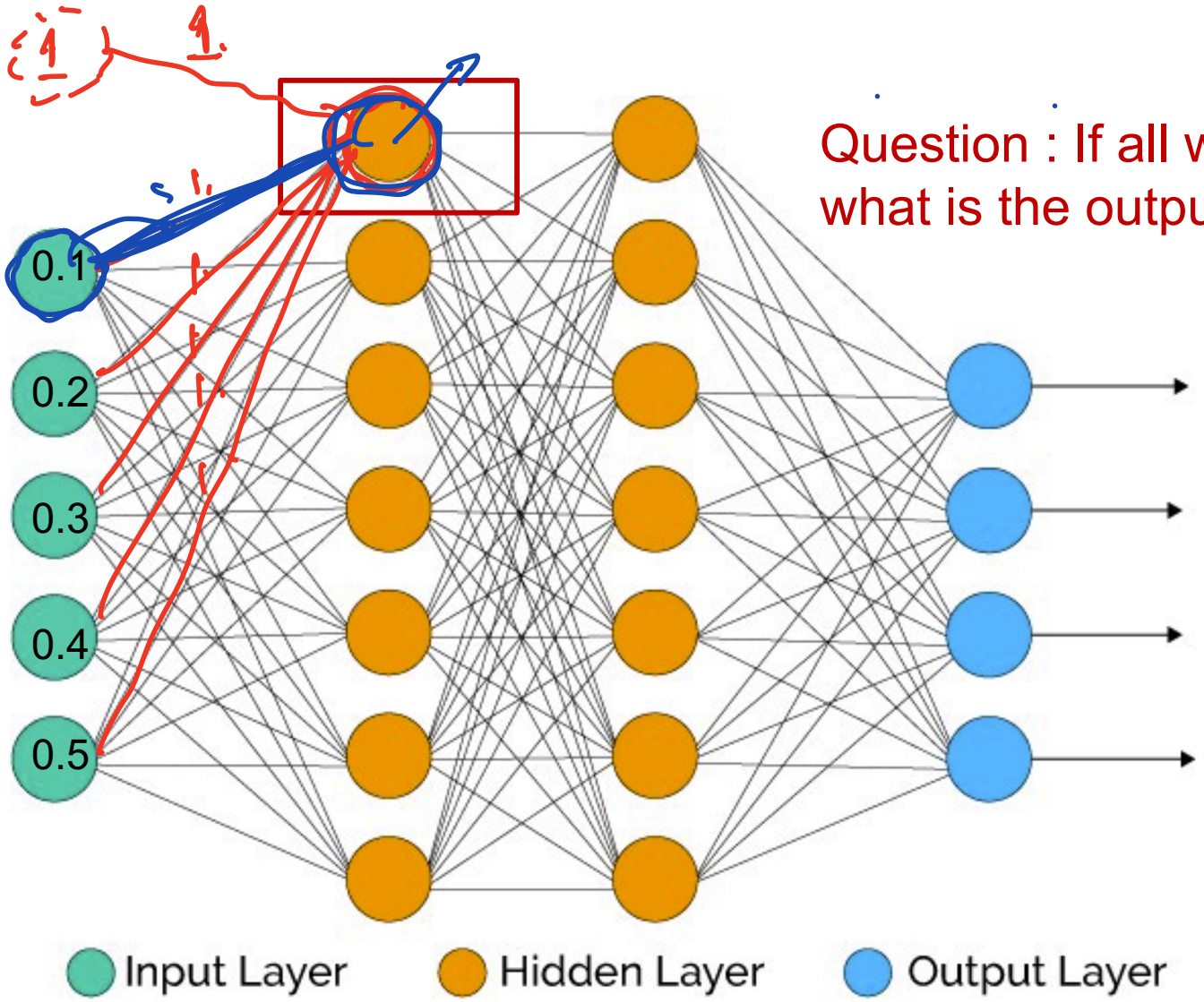


What it means



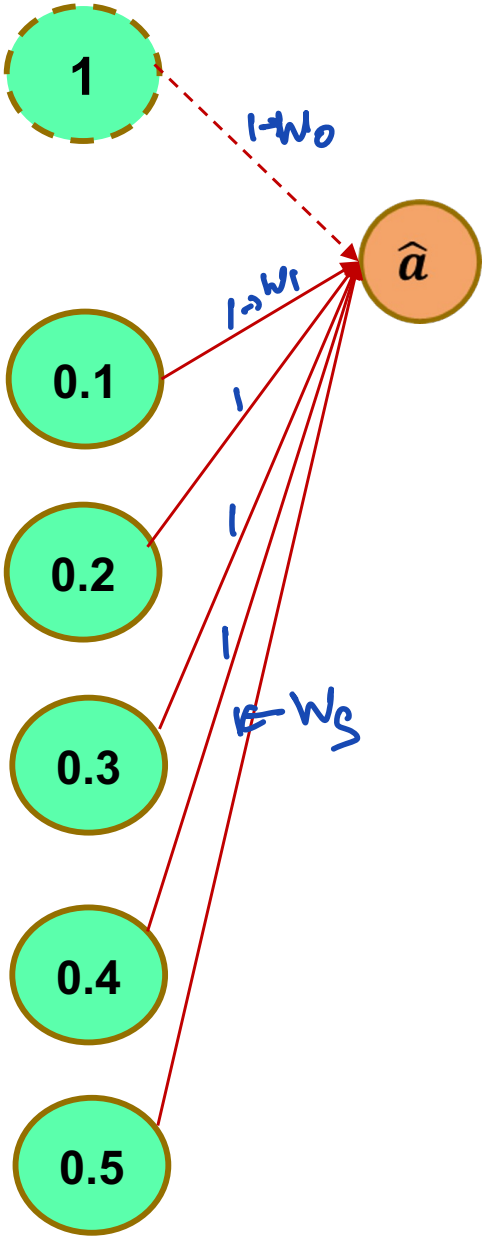
- Constant neurons representing a “feature” with 1 are usually not depicted in the pictorial representation.
- The parameters going from the constant neuron is called a bias and the neuron is called a “bias unit”
- The other parameters are called “weights”

The output of a single neuron



Question : If all weights are 1, then what is the output of the shown neuron?

$g(z) = \sigma(z)$



Find the output of the neuron $\hat{a}$

All weights are equal to 1

Ans: The output has two parts

Linear -- $z = \Sigma w_i x_i$

$$z = w_0 x_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5$$

$$z = 1 + 0.1 + 0.2 + 0.3 + 0.4 + 0.5 = 2.5$$

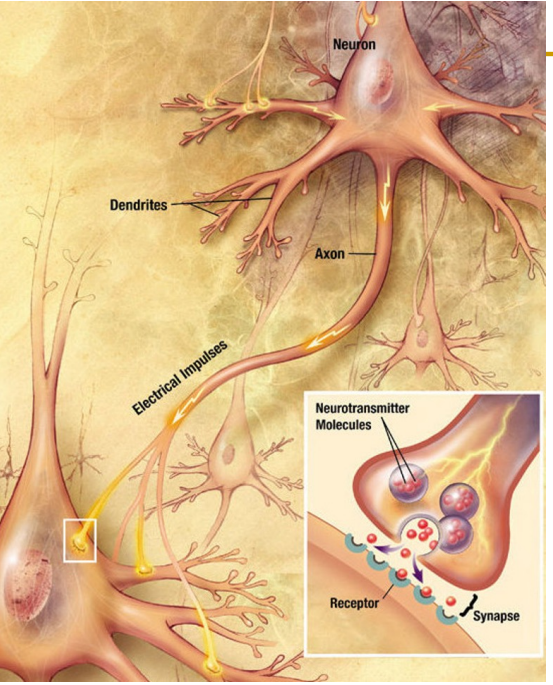
Nonlinear Activation -- $\hat{a} = g(z)$

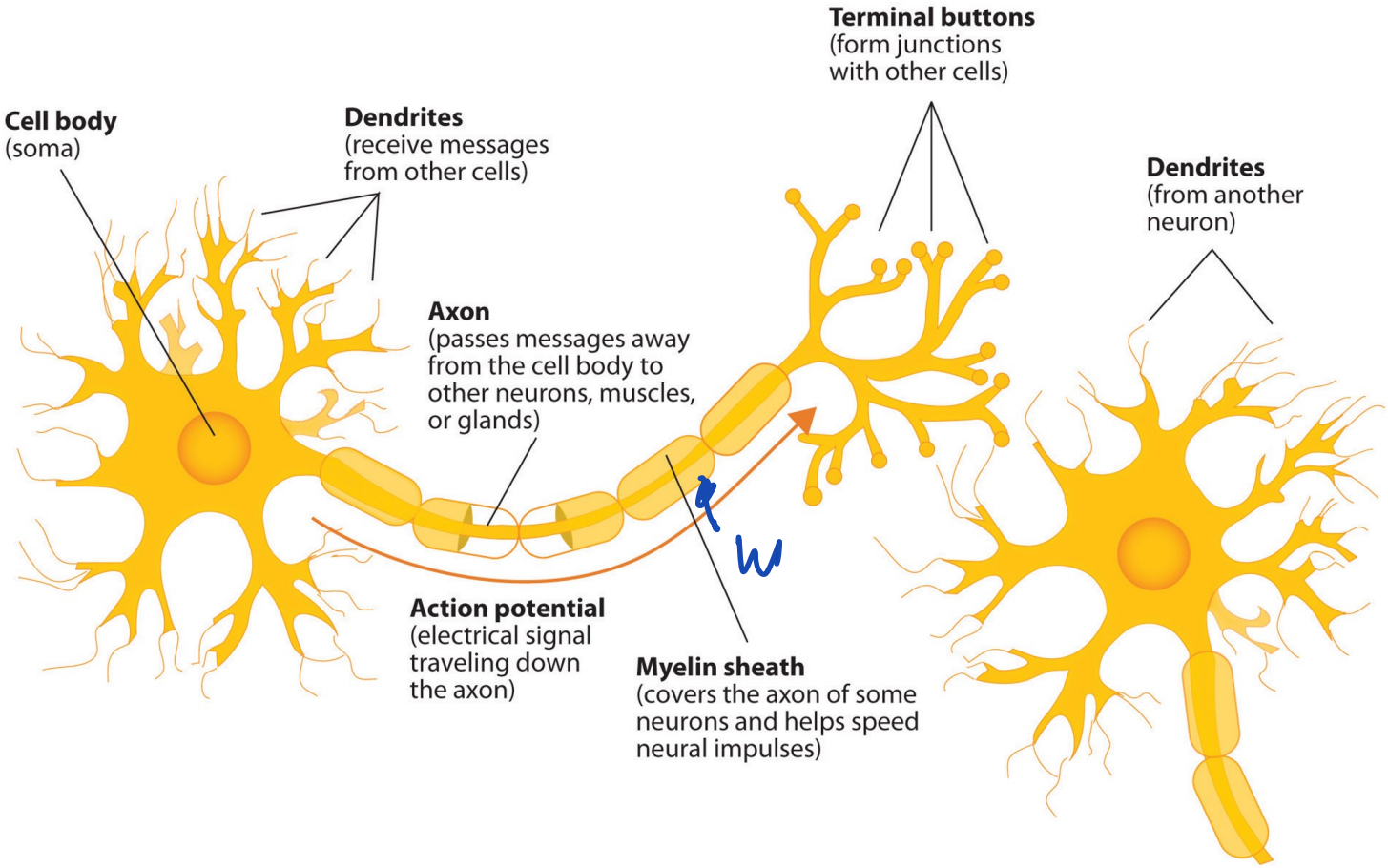
$$= \frac{1}{(1 + \exp(-2.5))}$$

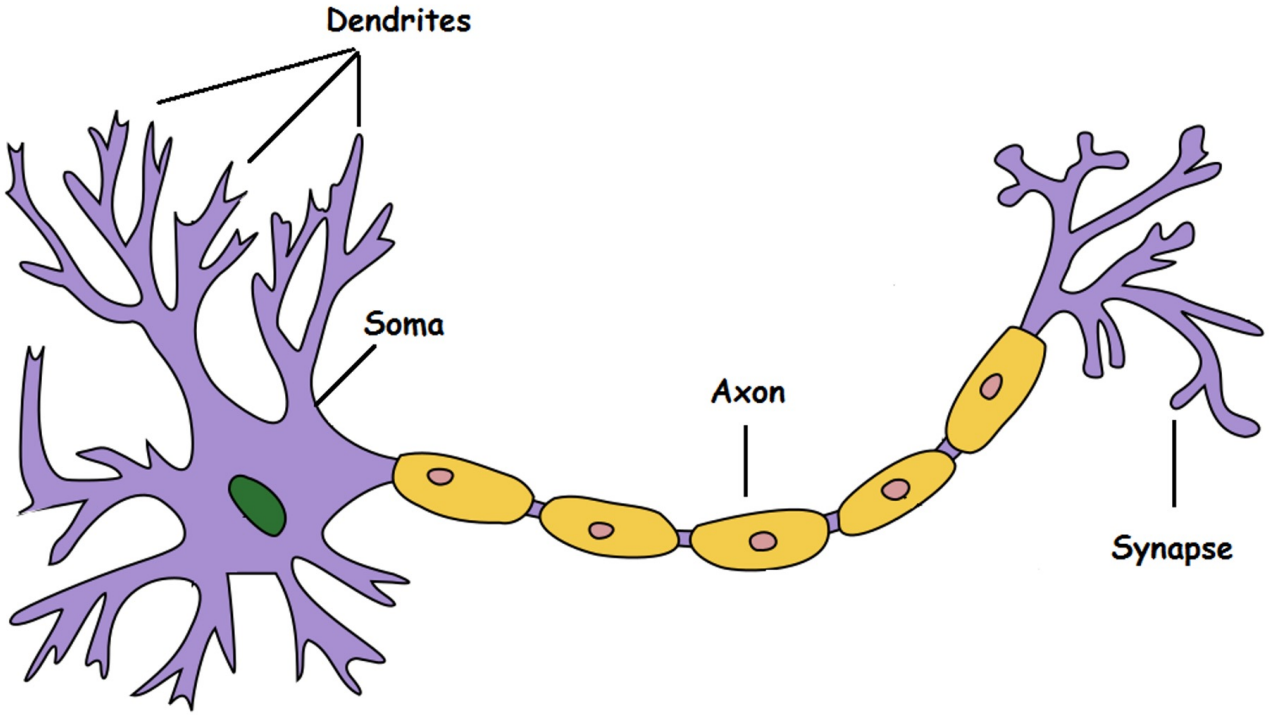
$$= 0.9241$$

Don't forget the bias unit!

From biology to computation

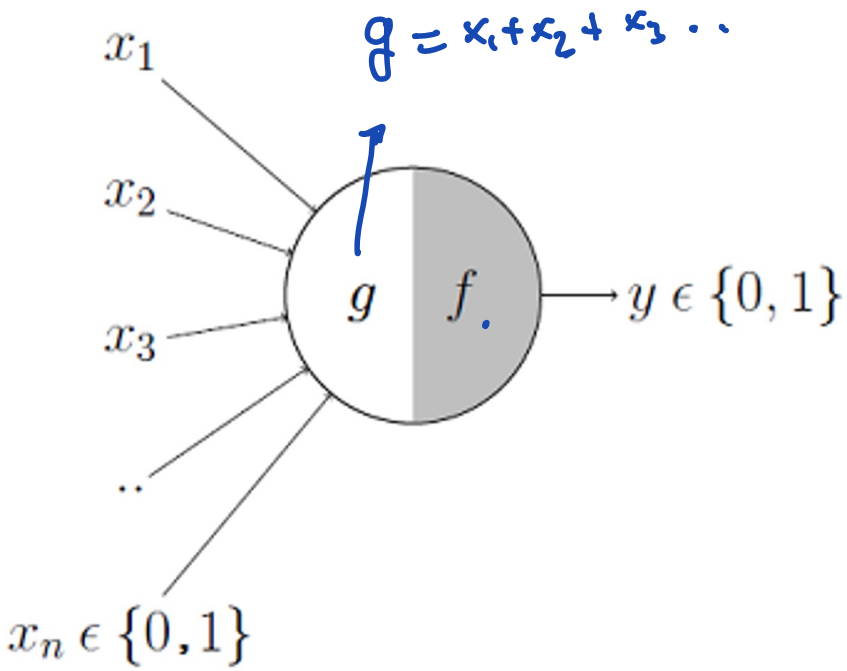

Neuron
Dendrites
Axon
Electrical Impulses
Neurotransmitter Molecules
Receptor
Synapse


Terminal buttons
(form junctions with other cells)
Cell body
(soma)
Dendrites
(receive messages from other cells)
Dendrites
(from another neuron)
Axon
(passes messages away from the cell body to other neurons, muscles, or glands)
Action potential
(electrical signal traveling down the axon)
Myelin sheath
(covers the axon of some neurons and helps speed neural impulses)


Dendrites
Soma
Axon
Synapse

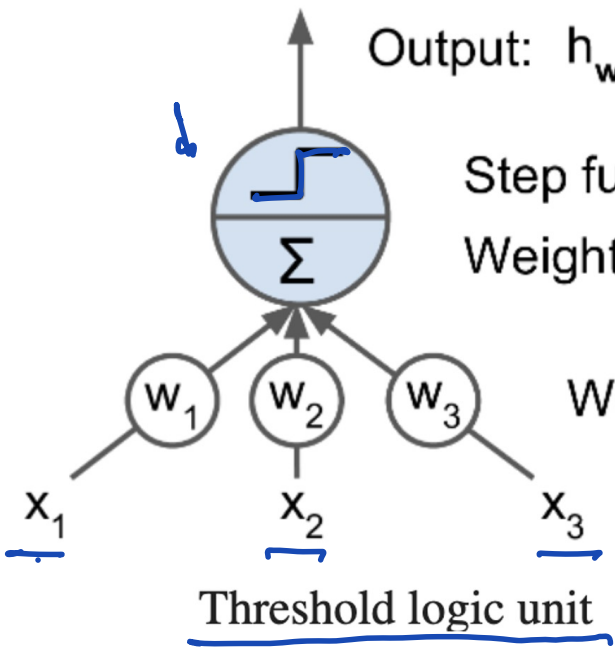
McCulloch-Pitts Neuron

- First Mathematical model of Neuron (1943)

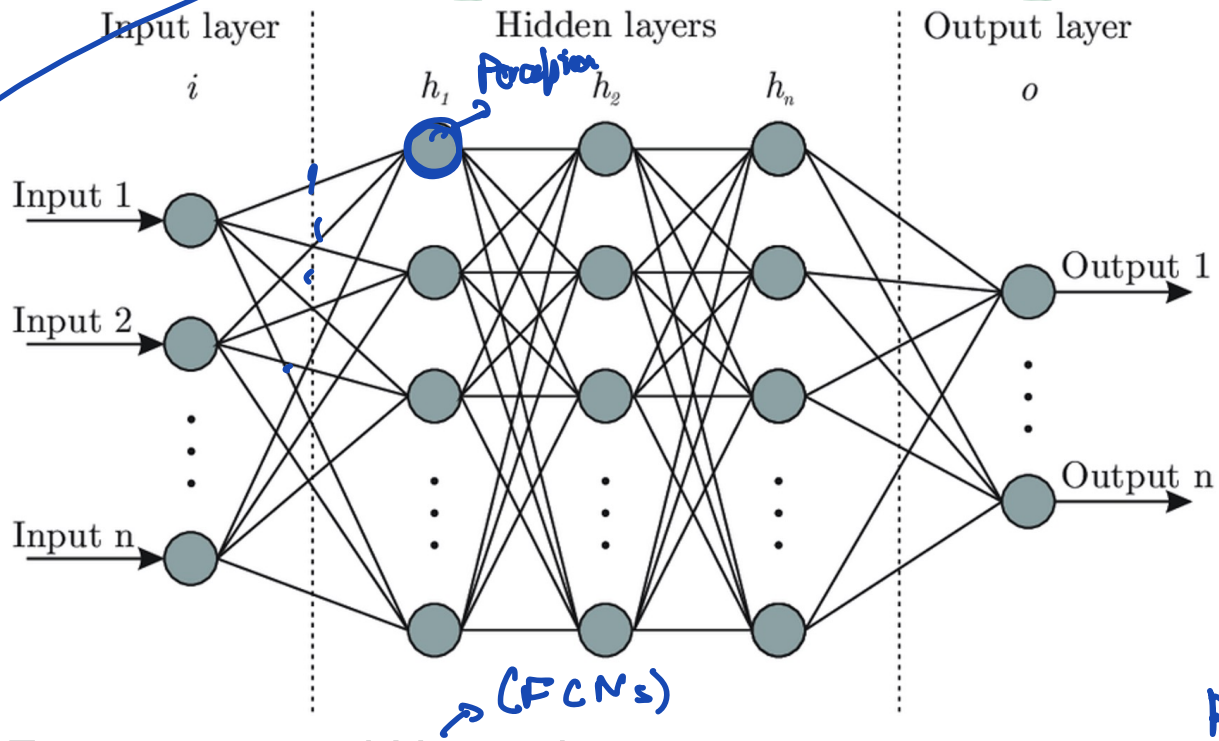


Logic Gate

Rosenblatt Perceptron (1957)



Multilayer Perceptrons/Deep NNs



- MLPs or Fully connected Networks are very common
  - At least as parts of other, more sparse networks

- Satisfy the Universal Approximation Theorem
  - Can approximate any function to arbitrary accuracy with just one layer.

Image from kdnuggets.com

Types of Neural Networks

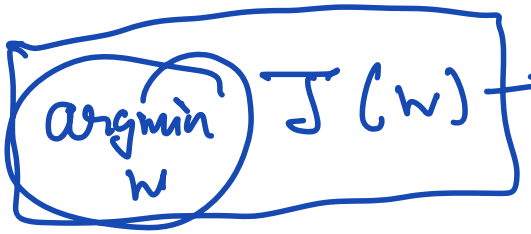
- Artificial Neural Networks
  - For general numerical data
- Convolutional Neural Networks (CNNs)
  - For image based data
- Recurrent Neural Networks (RNNs)
  - For sequential data

Transformers

Detailed Gradient Descent Derivation

For Linear Regression

Introduction



- Gradient Descent is widely used in machine learning algorithms.
- Used to minimize cost functions and optimize models.
- In this presentation, we delve deep into its derivation for linear regression with one variable.

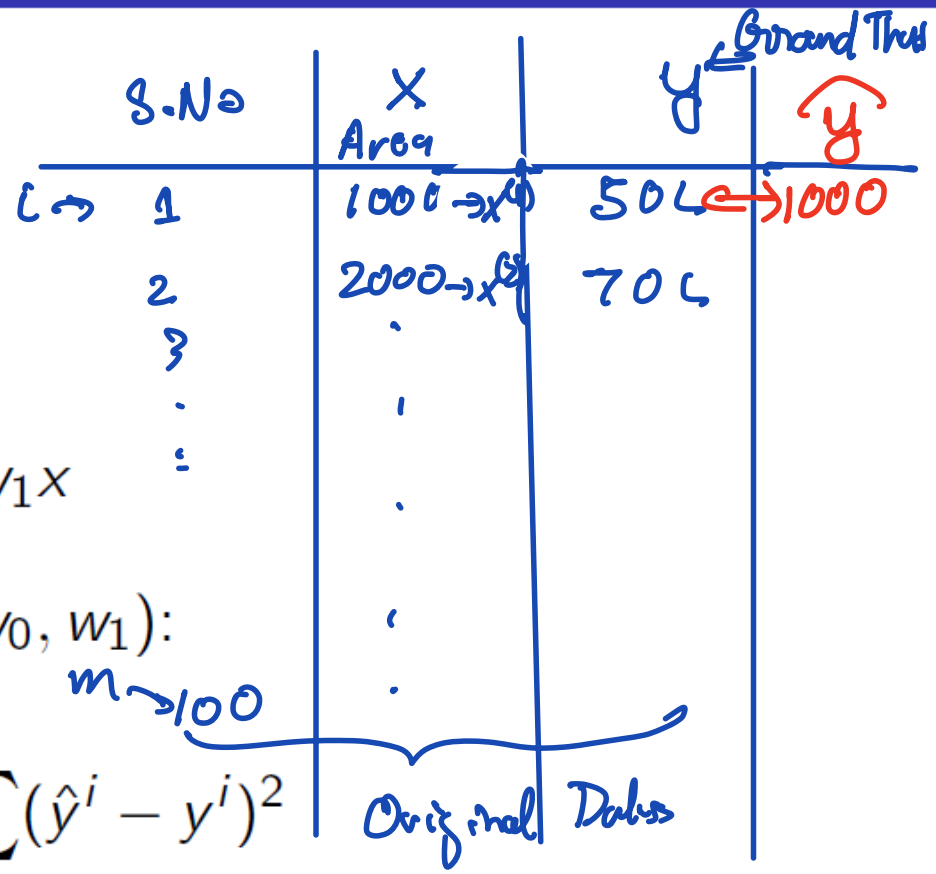
$$J(w_1, w_2) = (w_1 - 1)^2 + (w_2 - 2)^2$$

Value that J takes a min

$$\min J = 0 \rightarrow$$ What is the w that led to min J

$$(w_1 = 1, w_2 = 2) \rightarrow \text{argmin}$$

Linear Regression Model with One Variable



The hypothesis for our model:

$$\hat{y} = w_0 + w_1 x$$

Aiming to minimize the cost function $J(w_0, w_1)$:

MSE avg cost

$$J(w_0, w_1) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}^i - y^i)^2$$

$m = 100$

i^{th} data pt.

$x^{(i)}$

i^{th} data pt

$y^{(i)}$

y — Ground truth

m = # of data samples in the training set

Derivative with respect to w_0

$\frac{\partial J}{\partial w_0}$, $\frac{\partial J}{\partial w_1}$

Differentiating the squared error with respect to w_0:

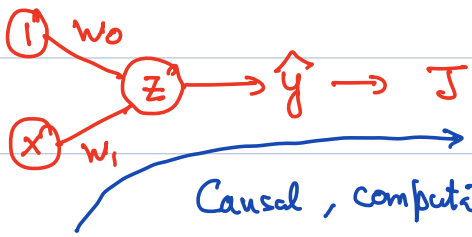
$$\frac{\partial}{\partial w_0}(\hat{y}^i - y^i)^2 = 2(\hat{y}^i - y^i)\frac{\partial \hat{y}^i}{\partial w_0}$$

Given:

$$\frac{\partial \hat{y}^i}{\partial w_0} = 1$$

We have:

$$\frac{\partial}{\partial w_0} J(w_0, w_1) = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}^i - y^i)$$



Causal, computational chain which led from $w_0, w_1 \rightarrow J$.

$z = w_0 + w_1 x$ $\quad \frac{\partial z}{\partial w_0} = 1$ $\quad \frac{\partial z}{\partial w_1} = x$

$\hat{y} = z \rightarrow$

$J = \frac{1}{2}(\hat{y} - y)^2$

$$\frac{\partial J}{\partial w_0} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial w_0}$$

$(\hat{y} - y)$ $\quad 1$ $\quad 1$

$$\frac{\partial J}{\partial w_0} = (\hat{y} - y) * 1$$

"DELTA" RULE

$$\frac{\partial J}{\partial w_j} = e * x_j$$

e: Error in output; x_j: Corresponding input

$$\frac{\partial J}{\partial w_1} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial w_1}$$

$(\hat{y} - y)$ $\quad x$

$$\frac{\partial J}{\partial w_j} = (\hat{y} - y) x_j$$

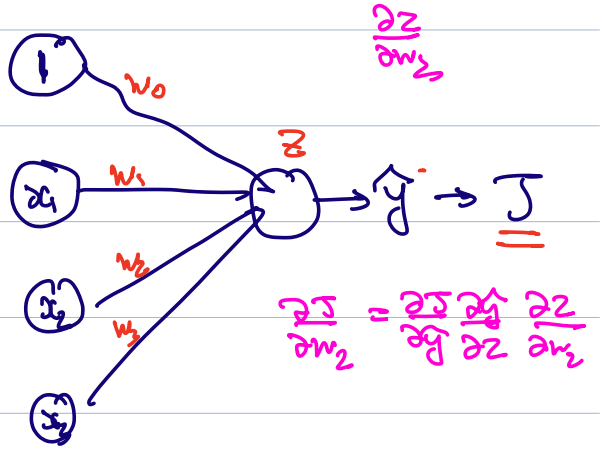
$(\hat{y} - y)$: Error in the output; x_j: Activation of the input

$z = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3$

$\frac{\partial z}{\partial w_2} = x_2$

$z = w_0 * 1 + w_1 x_1 + w_2 * x_2 + w_3 * x_3$

$\frac{\partial z}{\partial w_2}$



$\frac{\partial J}{\partial w_0} = \underbrace{(\hat{y} - y)}_{e}$

$\frac{\partial J}{\partial w_0} = e * 1$ $\quad \frac{\partial J}{\partial w_1} = e * x_1$ $\quad \frac{\partial J}{\partial w_2} = e * x_2$ $\quad \frac{\partial J}{\partial w_3} = e * x_3$

$$\frac{\partial J}{\partial w_2} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial w_2}$$

Derivative with respect to w_1

Using the chain rule:

$$\frac{\partial}{\partial w_1}(\hat{y}^i - y^i)^2 = 2(\hat{y}^i - y^i)x^i$$

Given:

$$\frac{\partial}{\partial w_1}\hat{y}^i = x^i$$

We get:

$$\frac{\partial}{\partial w_1}J(w_0, w_1) = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}^i - y^i)x^i$$

Derivative with respect to w_1

Using the chain rule:

$$\frac{\partial}{\partial w_1}(\hat{y}^i - y^i)^2 = 2(\hat{y}^i - y^i)x^i$$

Given:

$$\frac{\partial}{\partial w_1}\hat{y}^i = x^i$$

We get:

$$\frac{\partial}{\partial w_1}J(w_0, w_1) = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}^i - y^i)x^i$$

Final Expressions for linear regression

$$\text{Hypothesis:} \hat{y} = w_0 + w_1 x$$

$$\text{Cost Fn:} J = \frac{1}{2m} \sum (\hat{y}^i - y^i)^2$$

$$\text{Grad w.r.t } w_0 : \frac{\partial J}{\partial w_0} = \frac{1}{m} \sum (\hat{y}^i - y^i)$$

$$\text{Grad w.r.t } w_1 : \frac{\partial J}{\partial w_1} = \frac{1}{m} \sum (\hat{y}^i - y^i) x^i$$

Gradient Descent Updates

$$w_0 \leftarrow w_0 - \alpha \frac{\partial J}{\partial w_0}$$

$$w_1 \leftarrow w_1 - \alpha \frac{\partial J}{\partial w_1}$$

Quadratic Regression: Final Expressions

Lin. Model

S.No	x_1	x_1^2	y	$\hat{y}$
	.	.		
	.	.		
	.	.		
	.	.		
	.	.		
	.	.		

Features are non lin

x_0 x_1 x_2

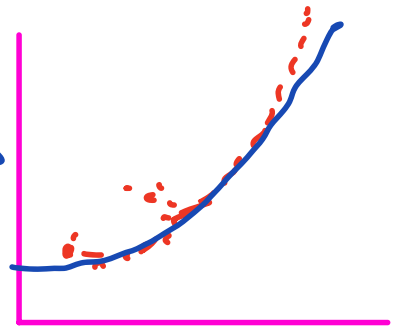
$$\text{Hypothesis: } \hat{y} = w_0 + w_1 x + w_2 x^2$$

Lin in weights

$$\text{Grad w.r.t } w_0 : \frac{\partial J}{\partial w_0} = \frac{1}{m}\sum(\hat{y}^i - y^i)$$

$$\text{Grad w.r.t } w_1 : \frac{\partial J}{\partial w_1} = \frac{1}{m}\sum(\hat{y}^i - y^i)x^i$$

$$\text{Grad w.r.t } w_2 : \frac{\partial J}{\partial w_2} = \frac{1}{m}\sum(\hat{y}^i - y^i)(x^i)^2$$



Nonlinear model

w_1 w_2

$\hat{y} = w_0 + w_1 x + w_2 \ln(x) + w_3 e^x$ → Lin as model

Feature Engineering

Gradient Descent Updates

$$w_0 \leftarrow w_0 - \alpha \frac{\partial J}{\partial w_0}$$

$$w_1 \leftarrow w_1 - \alpha \frac{\partial J}{\partial w_1}$$

$$w_2 \leftarrow w_2 - \alpha \frac{\partial J}{\partial w_2}$$

General n Feature Case: Final Expressions

$$\text{Hypothesis: } \hat{y} = w_0 + \sum_{j=1}^{n} w_j x_j$$

n # features

$$\text{Cost Fn: } J(\mathbf{w}) = \frac{1}{2m} \sum (\hat{y}^i - y^i)^2$$

$$\text{Gradient w.r.t } w_j : \frac{\partial J}{\partial w_j} = \frac{1}{m} \sum (\hat{y}^i - y^i) x_j^i \quad \text{for } j = 0, 1, \ldots, n$$

Data pts

Gradient Descent Updates

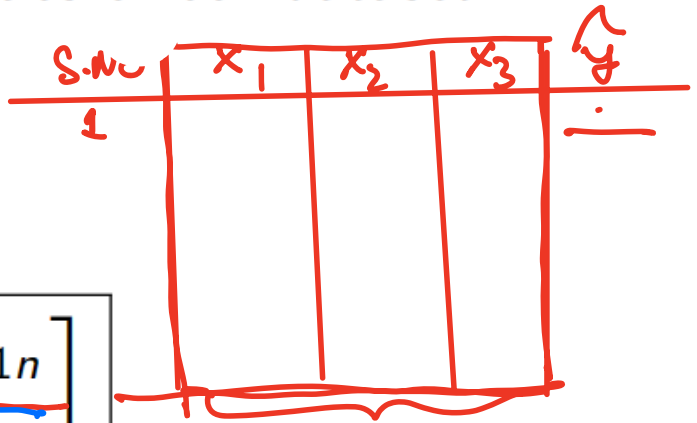
$$w_j \leftarrow w_j - \alpha \frac{\partial J}{\partial w_j} \quad \text{for each } j = 0, 1, \ldots, n$$

Matrix Notation for Linear Regression

Design Matrix (X)

The design matrix **X** organizes the feature values of our dataset:

- Each row corresponds to an observation.
- Each column corresponds to a feature.



X_{ij} → j^{th} feature, i^{th} data pt

X_{32} ← 2nd feature, 3rd data pt

Const feature → Bias

$$X = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1n} \\ 1 & x_{21} & x_{22} & \dots & x_{2n} \\ \vdots & \vdots & x_{32} & \ddots & \vdots \\ 1 & x_{m1} & x_{m2} & \dots & x_{mn} \end{bmatrix}$$

Matrix of inputs arranged column wise

The column of ones allows for the inclusion of the bias term in matrix notation.

Parameters (w)

$y = w^T x$

The parameters or weights, including the bias term, are represented as:

$$w = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_n \end{bmatrix}$$

w_0
w_1
w_2
w_3

$$X = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1n} \\ 1 & x_{21} & x_{22} & & x_{2n} \end{bmatrix}$$

$$\hat{y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \end{bmatrix}$$

- w_0 is the bias term.
- $w_1, \ldots, w_n$ are the weights for each feature.

Hypothesis Function ($\hat{y}$)

The model prediction or hypothesis function is denoted by $\hat{y}$ and is given by:

$\hat{y} = X * w$ → Matrix mult

$100 \times 1 \quad 100 \times 4 \quad 4 \times 1$

$m \times 1 \quad m \times (n+1) \quad (n+1) \times 1$

$$\hat{y} = Xw$$ → Matrix multiplication → Entire dataset

This is a vector where each element is the predicted response for an observation.

$$\hat{y}^{(i)} = w_0 + w_1 x_1^{(i)} + w_2 x_2^{(i)} \ldots + w_n x_n^{(i)}$$

$$= w_0 + w_1 x_{11} + w_2 x_{12} + w_3 x_{13} \ldots + w_n x_{1n}$$

$m = 100$

$n = 3$

↳ Does not include bias

Model Representation with Bias Included

With the bias w_0 included in the parameters:

$$\hat{y} = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1n} \\ 1 & x_{21} & x_{22} & \dots & x_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{m1} & x_{m2} & \dots & x_{mn} \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_n \end{bmatrix}$$

This compact form is efficient for computations and modeling.

Model Representation with Separate Bias

Alternatively, keeping the bias term separate:

$$\boxed{\hat{y} = Xw + b}$$

Here, **b** is a vector containing the bias term repeated for each observation.

- This form is mathematically equivalent to the previous one.
- The choice of representation depends on computational or conceptual preferences.

Derivation of the General Gradient Expression for the Linear Model in Linear Regression

Linear Regression Model

- Let's consider a dataset with m training examples and n features.
- Our design matrix X is $m \times (n+1)$ including a bias term.
- The parameter vector w is $(n+1) \times 1$.
- The observed value vector y is $m \times 1$.

The linear hypothesis function is given by:

$$\hat{y} = Xw$$

$m \times 1$

The goal is to minimize the cost function, which in the case of linear regression is often the Mean Squared Error (MSE).

Cost Function

The Mean Squared Error (MSE) cost function is defined as:

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

$$J(w) = \frac{1}{2m} (Xw - y)^\top (Xw - y)$$

To minimize $J(w)$, we will calculate its gradient $\nabla_w J(w)$.

Gradient Derivation: Step 1

Start with applying the chain rule to the cost function:

$$\nabla_w J(w) = \frac{1}{2m} \nabla_w [(Xw - y)^\top (Xw - y)]$$

Gradient Derivation: Step 2

Exploit the fact that for a scalar c, $\nabla_w c = 0$, and use the identity $\nabla_w a^\top w = a$ for a independent of w:

$$\nabla_w J(w) = \frac{1}{2m} \nabla_w [w^\top X^\top X w - 2 y^\top X w + y^\top y]$$

Note: $y^\top y$ is a constant with respect to w.

Gradient Derivation: Step 3

We find the gradient of the quadratic form $w^\top X^\top X w$ and linear terms $y^\top X w$:

$$\nabla_w J(w) = \frac{1}{2m}[2X^\top X w - 2X^\top y]$$

Gradient Expression

$X^T X w \simeq X^T y$

After simplifying, the gradient of the cost function with respect to w is:

$$\nabla_w J(w) = \frac{1}{m} X^\top (Xw - y)$$

This is the direction of the steepest ascent in the cost function surface, and we need to move in the opposite direction to minimize the cost.

$\nabla_w J = 0 \Rightarrow$

$X^T X w - X^T y = 0$

$X^T X w = X^T y$ → Normal Eqns

$w = (X^T X)^{-1} X^T y$

Parameter Update Rule

In the gradient descent algorithm, we iteratively update w using the update rule:

$$w := w - \alpha \nabla_w J(w)$$

where α is the learning rate.

Conclusion

- We derived the expression for the gradient of the cost function in linear regression.
- The gradient is used to update the parameters w in the direction that minimizes the cost.
- Gradient descent is an iterative optimization algorithm used for finding the values of parameters w that minimize a cost function $J(w)$.

Gradient Descent Algorithm

1. **Initialization:** Choose an initial vector of parameters $\mathbf{w}$ and set learning rate α.
2. **Repeat Until Convergence:**
   1. Compute the gradient:

$$\Delta \mathbf{w} = \frac{\partial J}{\partial \mathbf{w}} = \frac{1}{m} X^T (X\mathbf{w} - \mathbf{y})$$

   2. Update the weights:

$$\mathbf{w} = \mathbf{w} - \alpha \Delta \mathbf{w}$$

3. **Termination:** The algorithm converges when the cost function decreases by an amount smaller than a pre-defined threshold or after a pre-defined number of iterations.

General Supervised Learning Algorithm

- Step 1 : Data Collection – Collect Data in (x, y) pairs
- Step 2 : Parameter Initialization – Guess for w
- Step 3 : Forward Pass – For all data points in the dataset find $\hat{y}$ using a hypothesis Model
- Step 4 : Gradient Descent – Improve w using gradient descent. $w_j = w_j - \alpha \frac{\partial J}{\partial w_j}$
- Step 5 : Check exit criterion.
  - If not satisfied, go to Step 3 with the updated parameters

$$\mathbf{x}^{(1)} = \left(x_1^{(1)}, x_2^{(1)}, .., x_n^{(1)}\right)$$ no. of features

$$\mathbf{y}^{(1)} = \left(y_1^{(1)}, y_2^{(1)}, .., y_K^{(1)}\right)$$ Classes

$$\hat{\mathbf{y}}^{(1)} = \left(\hat{y}_1^{(1)}, \hat{y}_2^{(1)}, .., \hat{y}_K^{(1)}\right)$$

x	y	$\hat{y}$	J	
$\mathbf{x}^{(1)}$	$\mathbf{y}^{(1)}$	$\hat{\mathbf{y}}^{(1)}$	$J^{(1)}$	
$\mathbf{x}^{(2)}$	$\mathbf{y}^{(2)}$	$\hat{\mathbf{y}}^{(2)}$	$J^{(2)}$	
$\mathbf{x}^{(3)}$	$\mathbf{y}^{(3)}$	$\hat{\mathbf{y}}^{(3)}$	$J^{(3)}$	
$\mathbf{x}^{(4)}$	$\mathbf{y}^{(4)}$	$\hat{\mathbf{y}}^{(4)}$	$J^{(4)}$	
$\mathbf{x}^{(5)}$	$\mathbf{y}^{(5)}$	$\hat{\mathbf{y}}^{(5)}$	$J^{(5)}$	
…	…	…	…	
$\mathbf{x}^{(m)}$	$\mathbf{y}^{(m)}$	$\hat{\mathbf{y}}^{(m)}$	$J^{(m)}$	$J = \Sigma J^{(i)}$

Linear Regression – Learning Algorithm

- Step 1 : Data Collection – Collect Data in (x, y) pairs
- Step 2 : Parameter Initialization – Guess for w
- Step 3 : Forward Pass – For all data points in the dataset find $\hat{y}$ using
  - The Linear Model $y^{(j)} = \Sigma_{i=0}^{n} w_i x_i^{(j)}$
- Step 4 : Gradient Descent – Improve w using gradient descent. $w_j = w_j - \alpha \frac{\partial J}{\partial w_j}$
  - Gradient $\frac{\partial J}{\partial w_j} = \frac{1}{m}\Sigma_{i=1}^{m}\left(\hat{y}^{(i)} - y^{(i)}\right)x_j^{(i)}$
- Step 5 : Check exit criterion.
  - If not satisfied, go to Step 3 with the updated parameters

x	y	$\hat{y}$	J
$x^{(1)}$	$y^{(1)}$	$\hat{y}^{(1)}$	$J^{(1)}$
$x^{(2)}$	$y^{(2)}$	$\hat{y}^{(2)}$	$J^{(2)}$
$x^{(3)}$	$y^{(3)}$	$\hat{y}^{(3)}$	$J^{(3)}$
$x^{(4)}$	$y^{(4)}$	$\hat{y}^{(4)}$	$J^{(4)}$
$x^{(5)}$	$y^{(5)}$	$\hat{y}^{(5)}$	$J^{(5)}$
…	…	…	…
$x^{(m)}$	$y^{(m)}$	$\hat{y}^{(m)}$	$J^{(m)}$

$J = \frac{1}{m}\Sigma J^{(i)}$

Practical Consideration – Batching Gradient Descent

Gradient Update

$$w_j = w_j - \alpha \frac{\partial J}{\partial w_j}$$

x	y	$\hat{y}$	J
$\mathbf{x}^{(1)}$	$\mathbf{y}^{(1)}$	$\hat{\mathbf{y}}^{(1)}$	$J^{(1)}$
$\mathbf{x}^{(2)}$	$\mathbf{y}^{(2)}$	$\hat{\mathbf{y}}^{(2)}$	$J^{(2)}$
$\mathbf{x}^{(3)}$	$\mathbf{y}^{(3)}$	$\hat{\mathbf{y}}^{(3)}$	$J^{(3)}$
$\mathbf{x}^{(4)}$	$\mathbf{y}^{(4)}$	$\hat{\mathbf{y}}^{(4)}$	$J^{(4)}$
$\mathbf{x}^{(5)}$	$\mathbf{y}^{(5)}$	$\hat{\mathbf{y}}^{(5)}$	$J^{(5)}$
…	…	…	…
$\mathbf{x}^{(m)}$	$\mathbf{y}^{(m)}$	$\hat{\mathbf{y}}^{(m)}$	$J^{(m)}$

$$\frac{\partial \mathrm{J}}{\partial w_j} = \frac{1}{\mathrm{b}} \frac{\Sigma \partial J^{(i)}}{\partial w_j}$$

b is the size of the batch

- Default option – Each update in gradient descent requires to see the whole dataset
  - That is, one update requires us to average the updates of every point in the whole dataset
  - This is called **Batch Gradient Descent** where b = m. One epoch has one update
  - Problems : (i) It is slow (ii) It is expensive for each update (iii) Data cannot be dynamic
- Stochastic Gradient Descent
  - $b = 1$. One epoch has m updates. Many more updates per epoch
  - Can handle dynamic incoming data but loss convergence is more erratic
- Mini-Batch Gradient Descent
  - $1 < b < m$. One epoch has $\frac{m}{b}$ updates. More than 1 update per epoch
  - Loss convergence more erratic than Batch and less erratic than stochastic

Data Normalization and Its Impact on Gradient Descent for Linear Regression

Introduction

Data normalization and scaling, key preprocessing steps in machine learning, standardize the range of independent variables or features of the data. Especially for linear regression with gradient descent, these techniques can significantly enhance convergence speed and stability.

Machine Precision $\rightarrow 10^{-16}$

What is Data Normalization?

Normalization scales the dataset to have a mean of zero and a standard deviation of one. For a feature x, its normalized version x' is:

$$x' = \frac{x - \mu}{\sigma}$$

Batch norm in NN

Where:

- μ is the mean of the feature.
- σ is the standard deviation of the feature.

What is Min-Max Scaling?

Min-Max scaling transforms features to lie between 0 and 1. For a feature x, its scaled version x' is:

$$x' = \frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}}$$

Where:

- x_{min} is the minimum value of the feature.
- x_{max} is the maximum value of the feature.

Why Normalize or Scale Data for Gradient Descent?

- **Faster Convergence:** Different feature scales result in skewed contour plots, leading to slower convergence.
- **Avoidance of Numerical Instability:** Large magnitude features can induce numerical issues.
- **Improved Model Interpretability:** Scaled features lead to more interpretable coefficients.
- **Consistent Regularization Impact:** Equal treatment of all features by regularization techniques.

Impact on Gradient Descent in Linear Regression

The cost function J in linear regression is:

$$J = \frac{1}{2m} \sum (\hat{y}^i - y^i)^2$$

Feature scales affect gradient magnitudes:

- Different scales lead to oscillatory behavior.
- Normalized or scaled data ensures balanced gradients and uniform weight updates.

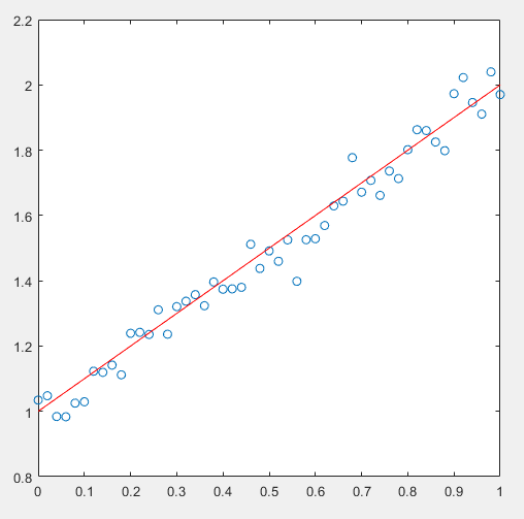
Conclusion

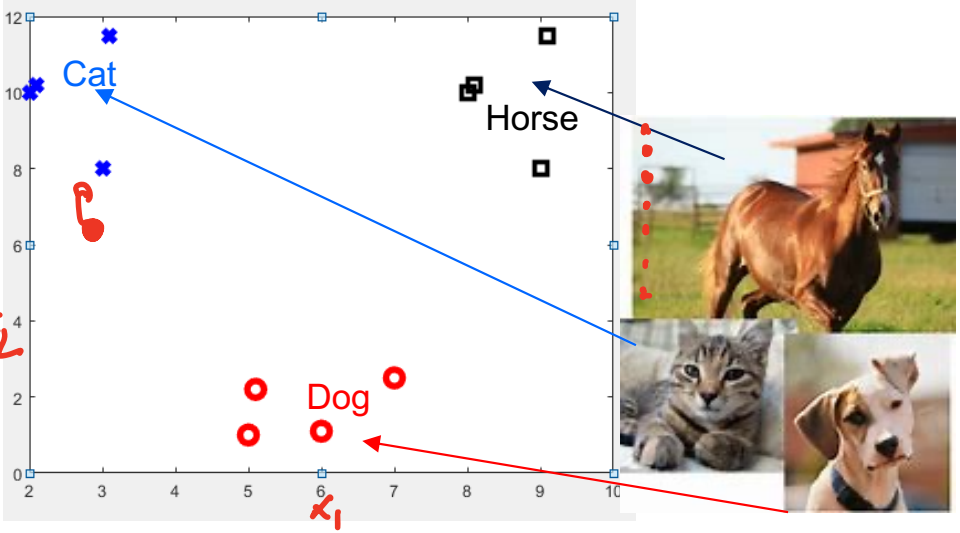
- Data normalization and scaling are crucial for efficient gradient descent in linear regression.
- Both techniques ensure faster and more reliable convergence.
- While multiple methods exist, normalization and Min-Max scaling are among the most popular for linear regression tasks.

Common non-linearities

Classification

Two problems in Supervised Learning





Regression	Classification
Fit it	Split it
Real number data	Discrete or Categorical data.
Has associated number	Has category associated
Example : Prediction of stock market	Example : Tumor classification
Model : Linear Model	**Model** : Logistic/Softmax

Binary Classification

Multiclass Classification

Definition of Machine Learning

- Simple Definition -- Using Data to answer questions
- Study of computer algorithms
  - that improve automatically
  - through experience.
- Formally, a computer program is said to learn from experience *E* with respect to some class of tasks *T* and performance measure *P* if its performance at tasks in *T*, as measured by *P*, improves with experience *E*.
- ML Recipe – Combine
  - Dataset representation
  - Cost function
  - Optimization procedure
  - Model

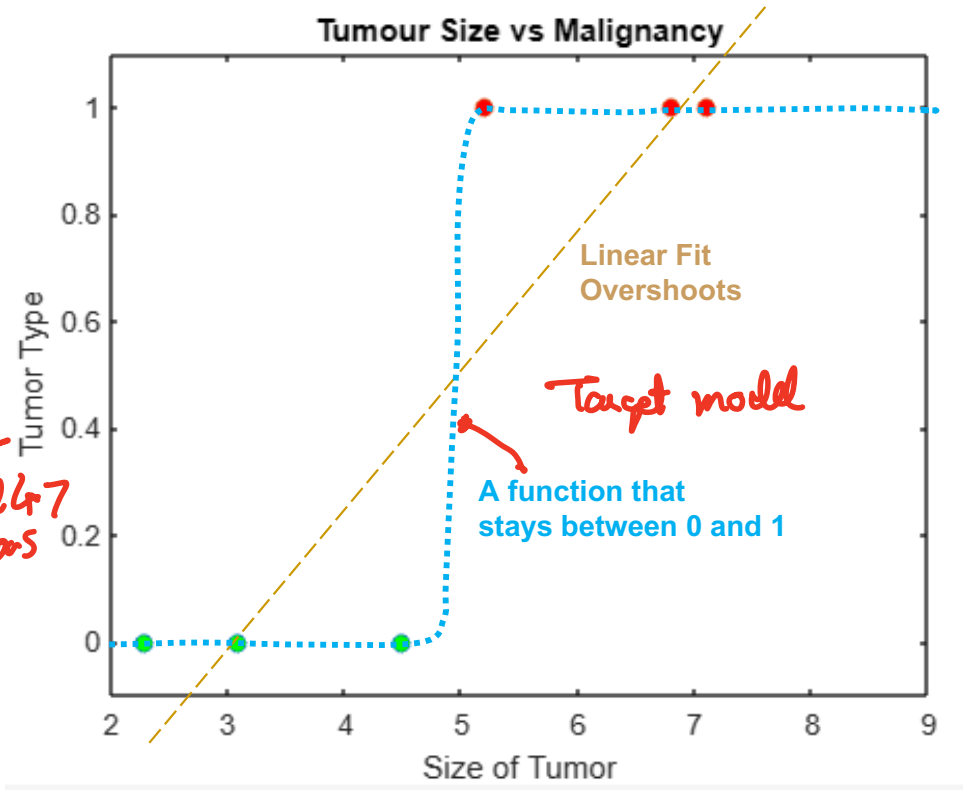
A simple binary classification problem

- Binary classification – Distinguishing between two classes
  - Benign/Malignant, Cat/Dog, Spam/Not Spam
- Imagine we are to characterize a tumor as benign or cancerous based on tumor size alone. Some prior data is given and we have to build a predictive model now
- Step 1 : Data Representation
  - Use numerical values for output – Benign = 0, Cancer = 1

Tumor Size (x in mm)	Outcome (y)
2.3	0
3.1	0
4.5	0
5.2	1
6.3	1
7.1	1

The Forward Model

Tumor Size (x in mm)	Outcome (y)
2.3	0
3.1	0
4.5	0
5.2	1
6.8	1
7.1	1

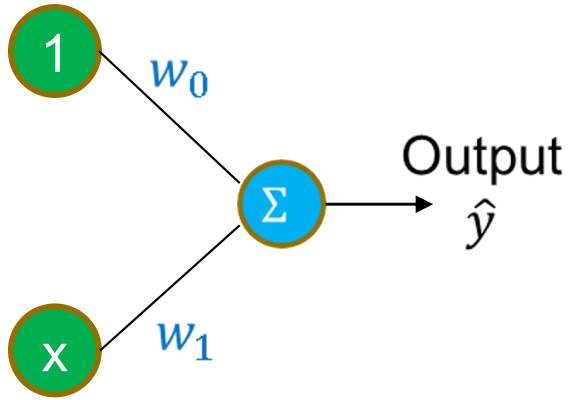


Step 2 : Create Forward Model

- We need a model that respects the data to be replicated
- We want the output to always be either 0 or 1
  - Or at least lie between 0 and 1
- A simple linear model $\hat{y} = w_0 + w_1 x$ will not obey this.
- So, we try the logistic model

From Linear to Logistic

Linear Model



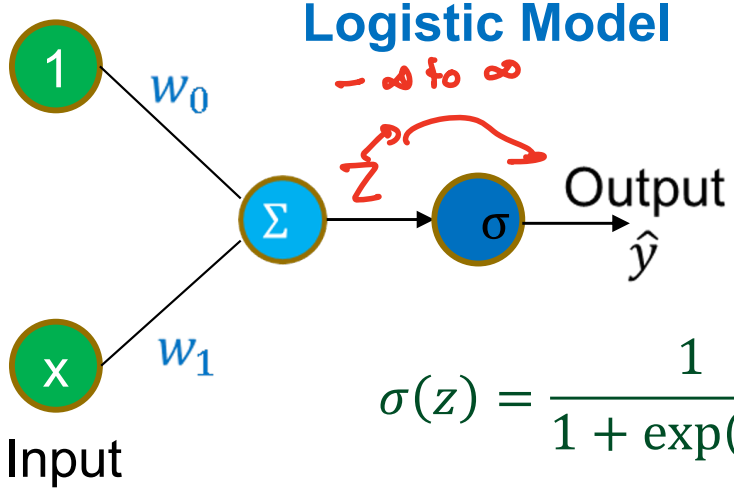
$Z \rightarrow -\infty$ to ∞

0 to 1

$$\hat{y} = w_0 + w_1 x = \Sigma w_i x_i$$

Output can be any real number

Logistic Model

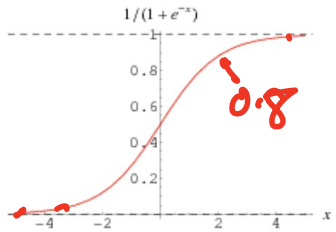


$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

$$z = w_0 + w_1 x = \Sigma w_i x_i$$

Logit/
Linear Activation

$$\hat{y} = \sigma(z)$$



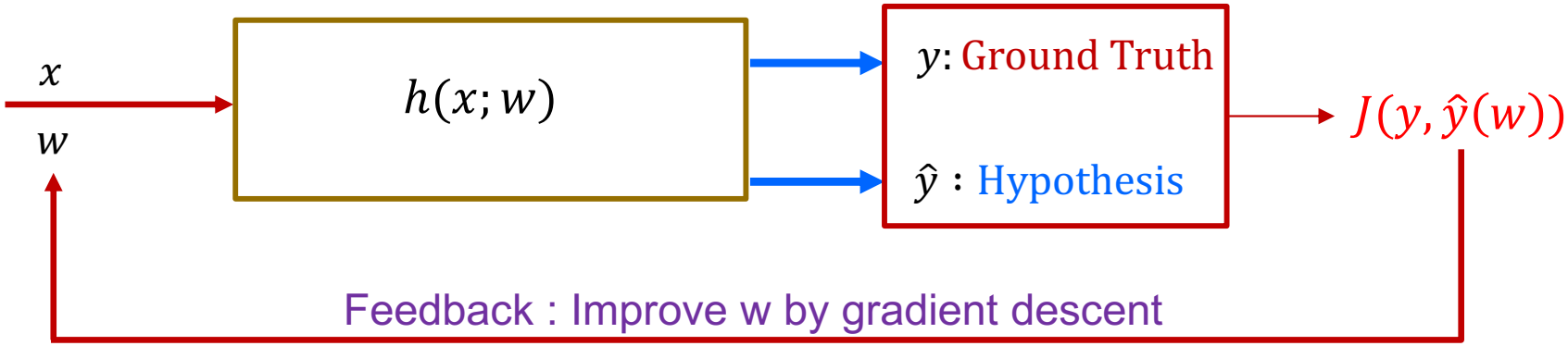
0.8

$\hat{y} = \sigma(w_0 + w_1 x)$ → Math model

Output will lie between 0 and 1 and can model probability p(y=1| x)

Interpretable

Completing the learning recipe



$y \rightarrow$ Either 0 or 1

- ML Recipe for Logistic Regression for classification
  - ✓ Dataset representation – x, y Numerical representation
  - ✓ Model -- $\hat{y} = h(x; w)$
  - ❑ Cost function -- $J(y, \hat{y})$
  - ❑ Optimization procedure

We need these to finish the recipe

Gradient descent

Least Square Loss
$(y - \hat{y})^2$

$\hat{y}$: Between 0 & 1

- The **cost function** $J(y, \hat{y})$ needs to satisfy the following properties

1. Correct Classifications should be as cheap as possible

   That is, if $\hat{y} = y$ then $J = 0$ → Least Square Loss satisfies this

2. Incorrect Classifications should be as expensive as possible

   If $y = 1$ but $\hat{y} \rightarrow 0$ then $J \rightarrow \infty$
   If $y = 0$ but $\hat{y} \rightarrow 1$ then $J \rightarrow \infty$ → Least Square Loss does not satisfy this

NOTE: Least square still works, despite this problem, but is slower compared to the Loss function we will use

Binary Cross-Entropy (BCE) Loss Function

- Recall that $J(y, \hat{y})$ the loss function quantifies the gap between the model output and the true output
- The Least Square error $J^{LSE}(y, \hat{y}) = (y - \hat{y})^2$
  - Can have a maximum value of only 1 for any given data point
  - Does not penalize incorrect classifications heavily enough
  - Derived from Gaussian distribution which is not accurate for binary data
- So, we introduce the binary cross entropy loss function

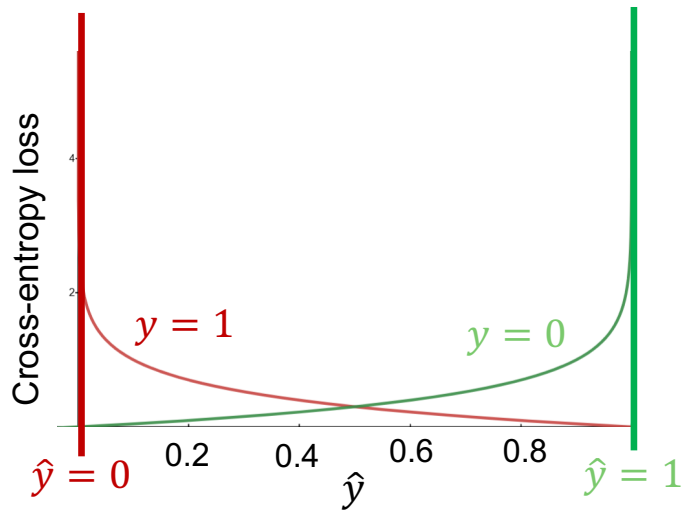
$$J^{BCE}(y, \hat{y}) = -y \, ln \, \hat{y} - (1 - y) \, ln \, (1 - \hat{y})$$

- This has the properties that
  - $J = 0 \; if \; y = \hat{y}$
  - $J \to \infty \; if \; y$ is totally misclassified
  - J is never negative
  - We will show these in the next slide

Binary Cross Entropy Properties

$$J^{BCE}(y,\hat{y}) = \begin{cases} -\log \hat{y} & if\ y = 1 \\ -\log(1-\hat{y}) & if\ y = 0 \end{cases}$$

$$J^{BCE}(y,\hat{y}) = -y \log \hat{y} - (1-y)\log(1-\hat{y})$$

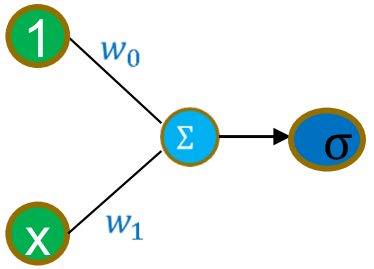


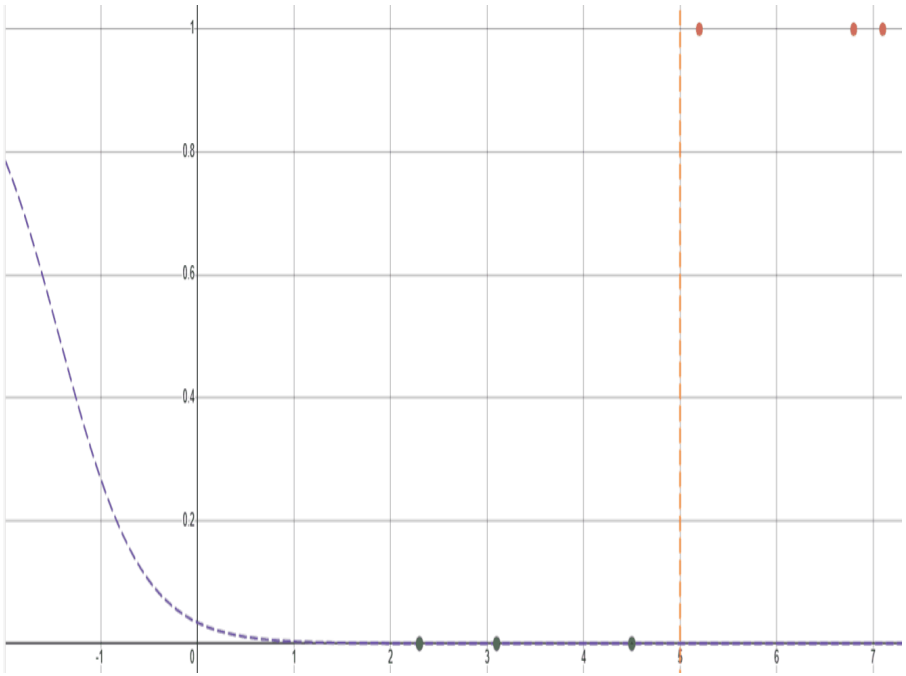
Properties of *ln*

- Logarithm to the base e. Inverse of the exponential. That is, $\log(e^x) = 1$
- We can see that $\log(1) = 0$ and $\log(0) \to -\infty$

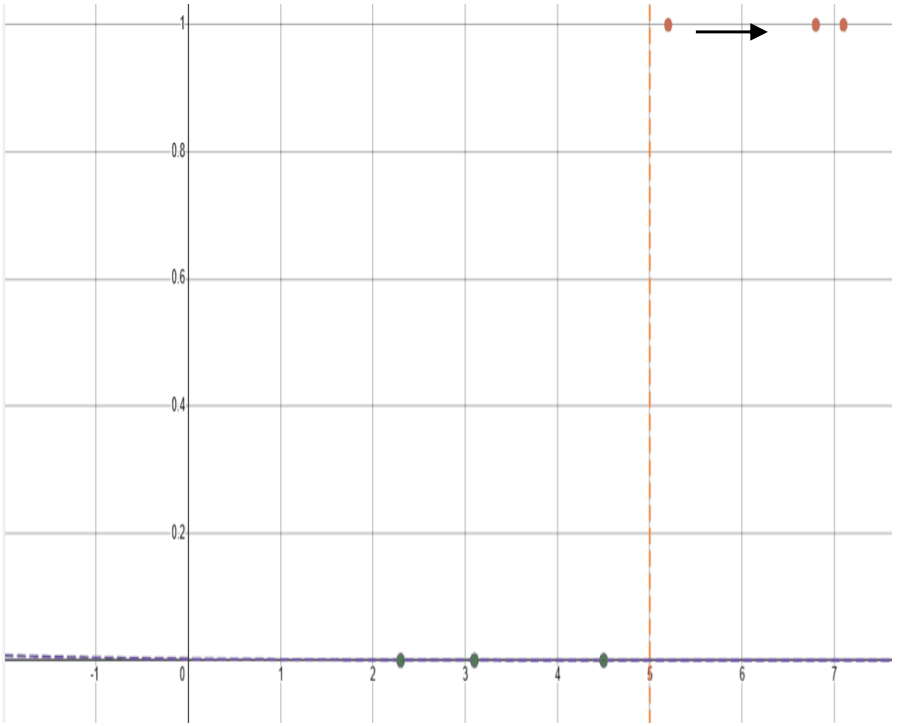
S.No	y	$\hat{y}$	$J(y,\hat{y}) = -y \log \hat{y} - (1-y)\log(1-\hat{y})$	Comments
1	1	≈ 1	$\approx -1\times\log(1) - 0 \to 0$	Correct classification incurs no cost
2	0	≈ 0	$\approx -0 - 1\times\log(1) \to 0$	Correct classification incurs no cost
3	1	≈ 0	$\approx -1\times\log(0) - 0 \to \infty$	Incorrect classification incurs infinite cost
4	0	≈ 1	$\approx -0 - 1\times\log(0) \to \infty$	Incorrect classification incurs infinite cost

Playing with the sigmoid – Varying the parameters





Changing w_0 with fixed w_1



Changing w_0 with fixed w_1

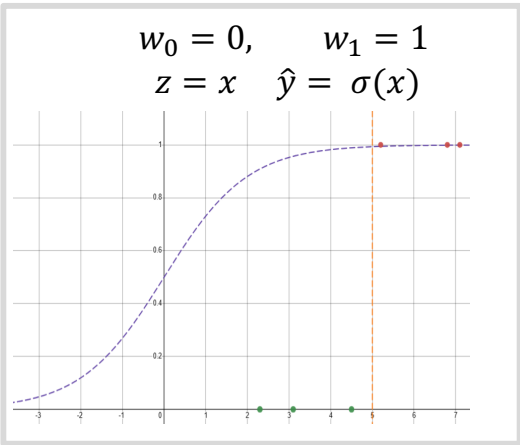
$$z = w_0 + w_1 x \quad \hat{y} = \sigma(z)$$

Changing w_0 seems to change the critical point and w_1 affects the sharpness

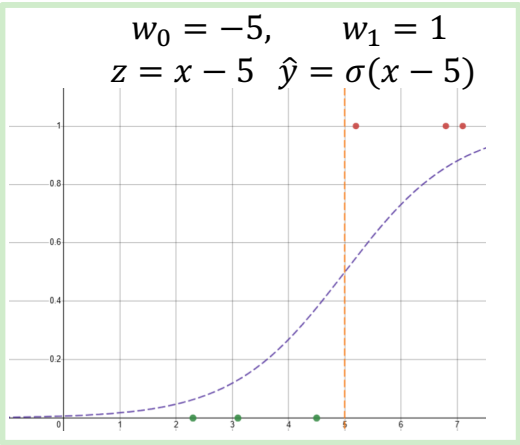
Playing with the sigmoid

$z = w_0 + w_1 x \quad \hat{y} = \sigma(z)$

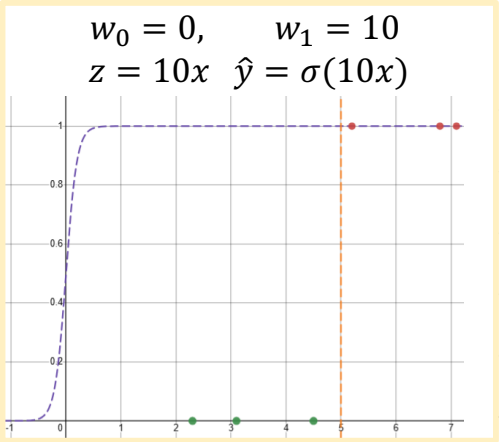
Let us try various values of the parameters to interpret what is happening



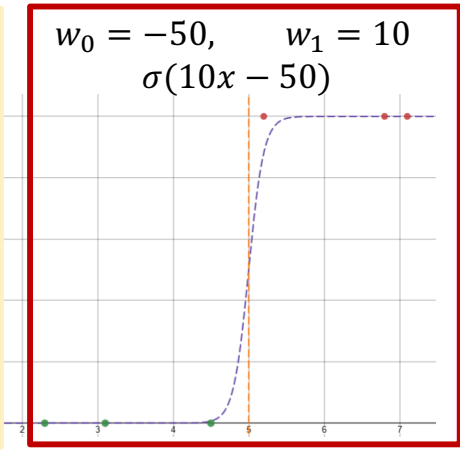
Case 1



Case 2



Case 3



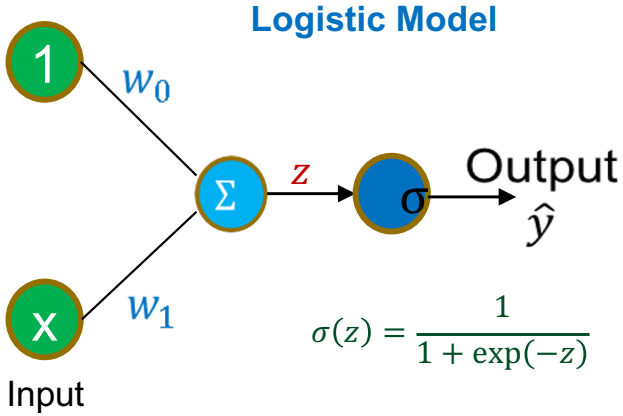
Case 4

Case #	w_0	w_1	z	Function $\hat{y}$	Comments
1	0	1	x	$\sigma(x)$	Does not capture the critical point and not sharp
2	-5	1	$x - 5$	$\sigma(x - 5)$	Captures critical point but not sharp
3	0	10	$10x$	$\sigma(10x)$	Sharp but underpredicts critical point
4	-50	10	$10x - 50$	$\sigma(10x - 50)$	Sharp and captures critical point reasonably

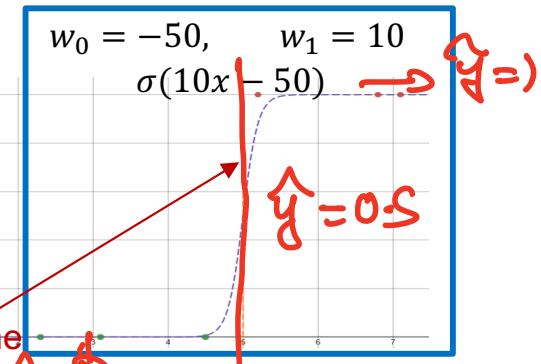
Let us now interpret the term $\sigma(z)$ and the logistic model so that we can systematize this

Interpreting the logistic model

$z = w_0 + w_1 x \quad \hat{y} = \sigma(z)$



Tumor Size (x in mm)	Outcome (y)
2.3	0
3.1	0
4.5	0
5.2	1
6.8	1
7.1	1



The critical/threshold/classification line separates the two classes in the binary problem

The criterion for the classifying line in terms of z (the network variable)

Far to the left of it we have $\hat{y} \to 0$ which means $z \to -\infty$

Far to the right of it we have $\hat{y} \to 1$ which means $z \to \infty$

- Right at the classifying line, we will have $\hat{y} \to 0.5$, i.e. $\boxed{z_c = 0}$

Classification criterion

The criterion for the classifying line in terms of x (the physical variable)

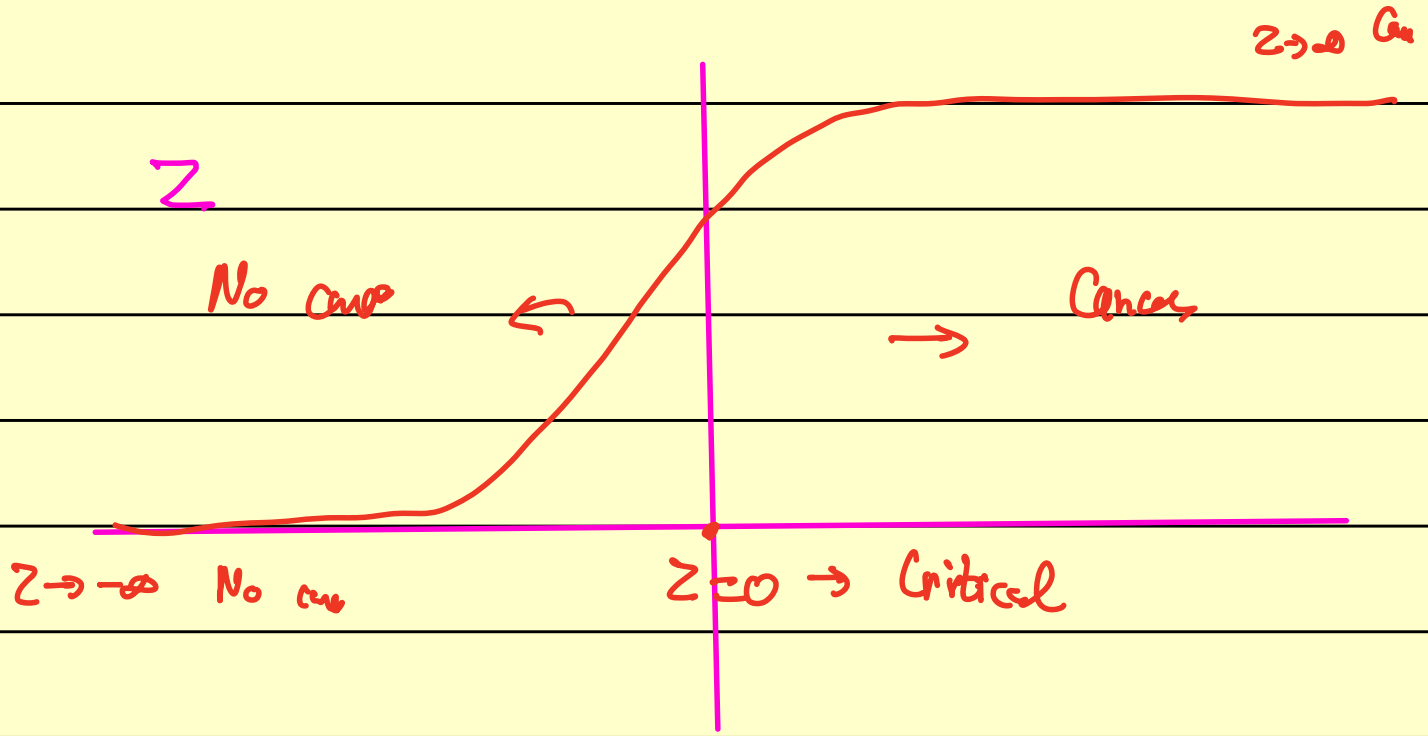
$$z = w_0 + w_1 x \Rightarrow z_c = w_0 + w_1 x_c \Rightarrow x_c = (z_c - w_0)/w_1$$

$$\text{Since } z_c = 0 \Rightarrow \boxed{x_c = -w_0/w_1}$$

We obtained $w_0 \approx -50, w_1 \approx 10$ manually $\Rightarrow x_c \approx -5$ which matches the data

Instead of obtaining these manually, can we automatically get these parameters via the "learning" process?

$z = w_0 + w_1 x$ → Change of variable



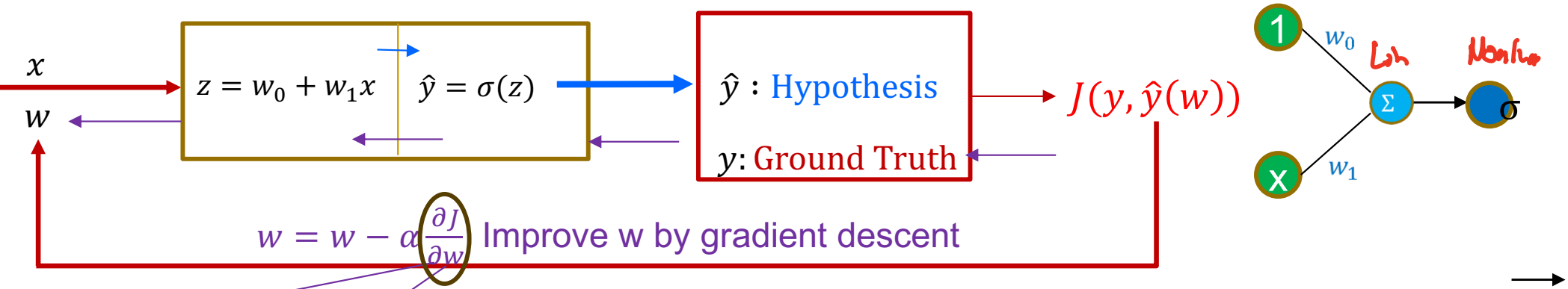
w_0, w_1 → Gradient Descent

$x = 3mm$

$w_0 + w_1 x = z = 1.1$

$\hat{y} = \sigma(z) = \sigma(1.1) \cong 0.83$

Gradient Operator for logistic regression



We need $\frac{\partial J}{\partial w_0}$ and $\frac{\partial J}{\partial w_1}$. Reverse the order of calculations to calculate these derivatives

Forward Prop $\rightarrow x; w \rightarrow z \rightarrow \hat{y} \rightarrow J$

Backward Prop $\rightarrow \Delta J \rightarrow \Delta\hat{y} \rightarrow \Delta z \rightarrow \Delta w$

Calculating $\frac{\partial J}{\partial \hat{y}}$

Use $\frac{d}{dx}\log(x) = \frac{1}{x}$

$$J = -y \log \hat{y} - (1-y)\log(1-\hat{y})$$

$$\frac{\partial J}{\partial \hat{y}} = -\frac{y}{\hat{y}} + \frac{(1-y)}{1-\hat{y}}$$

$$\frac{\partial J}{\partial w_0} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial w_0}$$

$$\frac{\partial J}{\partial w_1} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial w_1}$$

Common term $\frac{\partial J}{\partial z}$

$$\Rightarrow \frac{\partial J}{\partial z} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} = \hat{y} - y$$

Calculating $\frac{\partial \hat{y}}{\partial z}$ $\rightarrow \hat{y} = \frac{1}{1+\exp(-z)}$

$$\Rightarrow \frac{d\hat{y}}{dz} = \frac{\exp(-z)}{(1+\exp(-z))^2}$$

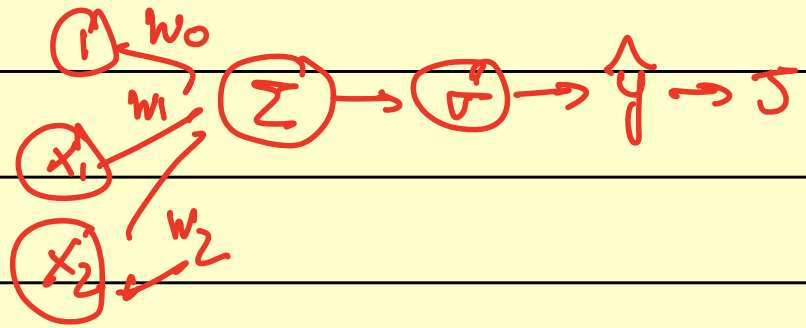
$$\Rightarrow \frac{d\hat{y}}{dz} = \frac{\exp(-z)}{(1+\exp(-z))} \times \frac{1}{(1+\exp(-z))}$$

$$\Rightarrow \frac{d\hat{y}}{dz} = \hat{y} \times (1-\hat{y})$$

$$\frac{\partial J}{\partial w_0} = \frac{\partial J}{\partial z} \frac{\partial z}{\partial w_0} = (\hat{y} - y) \times 1$$

$$\frac{\partial J}{\partial w_1} = \frac{\partial J}{\partial z} \frac{\partial z}{\partial w_1} = (\hat{y} - y) \times x$$

DELTA RULE



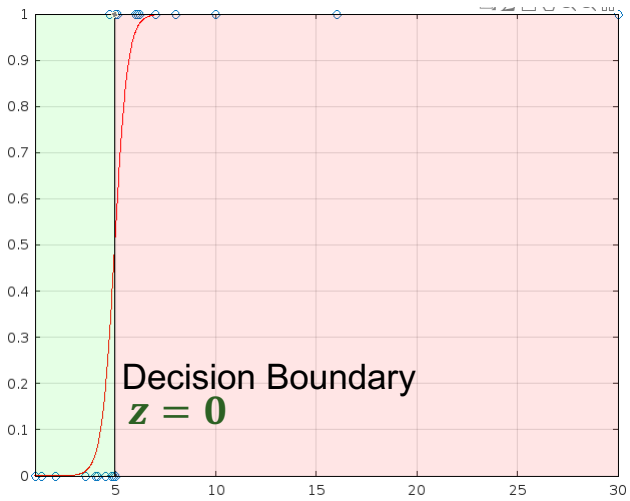
$$\frac{\partial J}{\partial w_2} = (\hat{y} - y) x_2$$

Binary Classification Algorithm

- Step 1 : Data Collection – Collect Data in (x, y) pairs
  - Denote y only as 0 or 1 (Binary)
- Step 2 : Parameter Initialization – Guess for w
  - For a 1 feature model, this is w_0 and w_1
- Step 3 : Forward Pass – For all data points in the dataset find $\hat{y}$
  - For a 1 feature model, this is $\hat{y}_i = \sigma(w_0 + w_1 x_i)$
- Step 4 : Gradient Descent – Improve w using gradient descent
  - $w_j = w_j - \alpha \frac{\partial J}{\partial w_j}$
  - For a 1 feature model, $\frac{\partial J}{\partial w_j} = (\hat{y} - y) \times x_j$
- Step 5 : Check exit criterion.
  - If not satisfied, go to Step 3 with the updated parameters

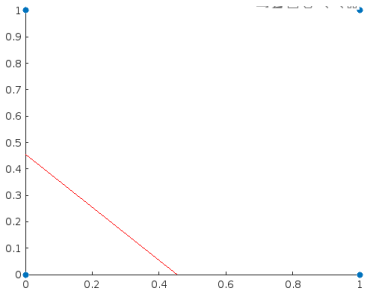
epochs

Tumor Size (x in mm)	True Outcome (y)	Model Outcome $\hat{y}$	Data Loss J_i
2.3	0		
3.1	0		
4.5	0		
5.2	1		
6.8	1		
7.1	1		

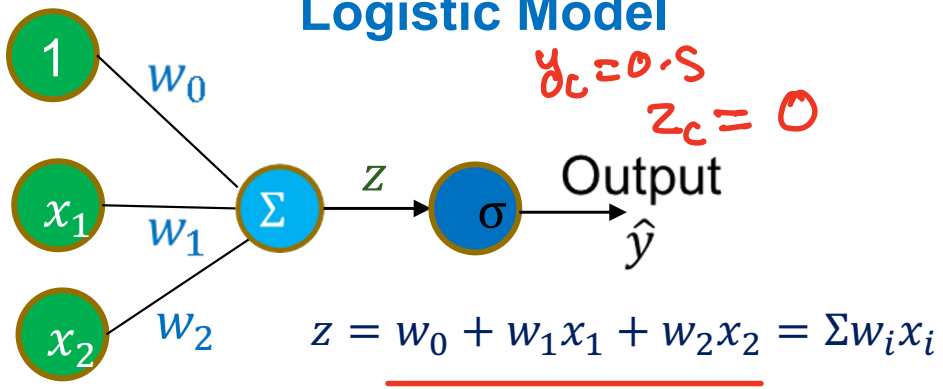


Binary Classification for more than one feature

	x_1	x_2	y
1	0	0	0
2	1	0	1
3	0	1	1
4	1	1	1



OR gate as a classification problem



- The technique for modeling more than 1 features remains the same.

$$z = Xw \quad \hat{y} = \sigma(z)$$

- For two features this becomes

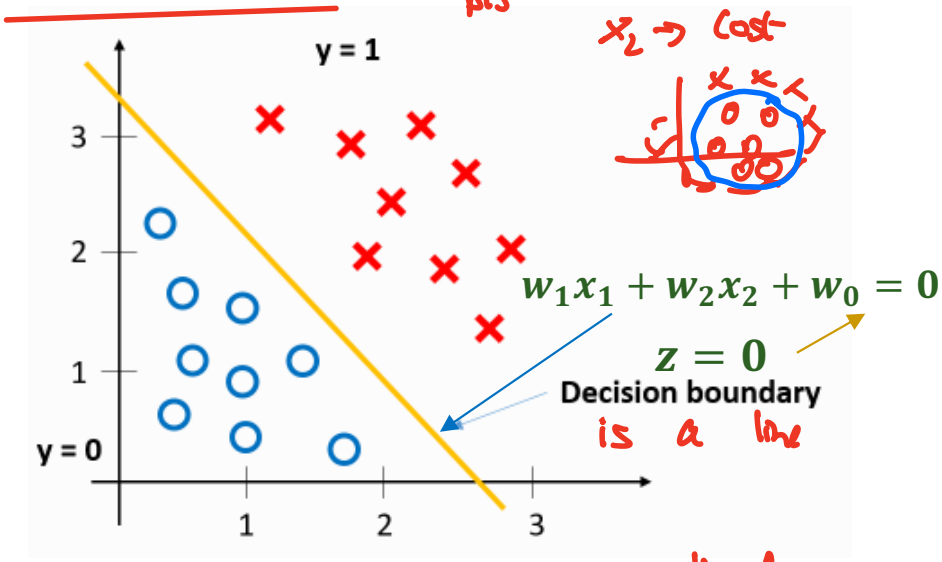
$$z = w_0 + w_1x_1 + w_2x_2 \quad \hat{y} = \sigma(z)$$

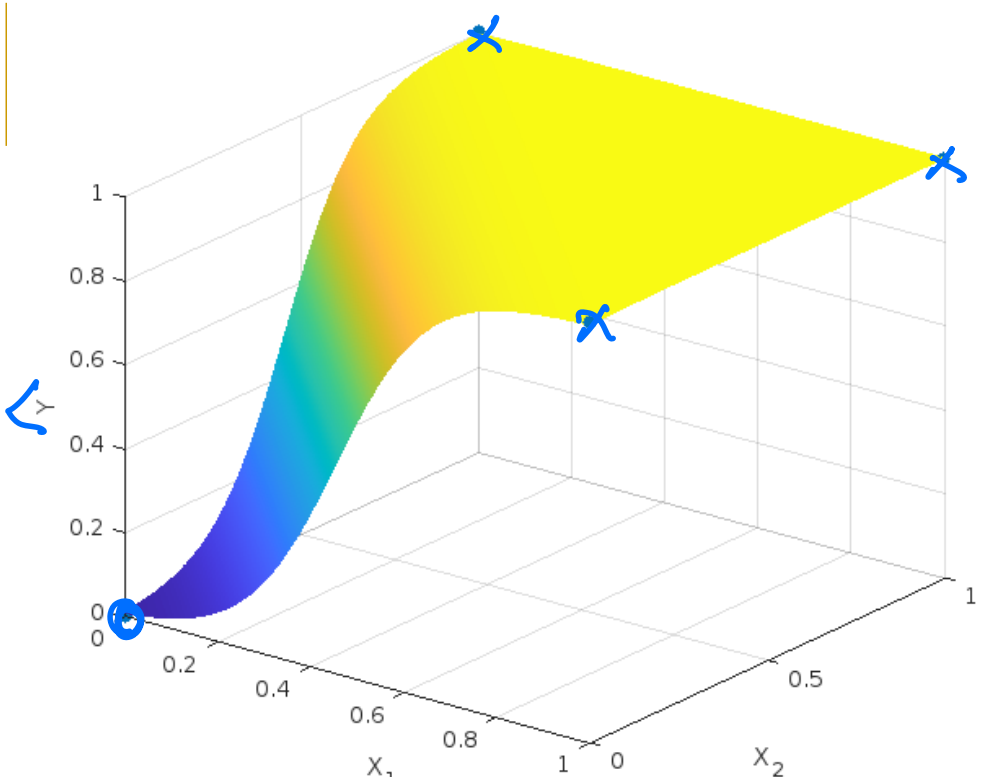
- The decision boundary remains $z = 0$
  - This means that the **boundary is always a line**
- Gradients follow the "Delta Rule"

$$\frac{\partial J}{\partial w_j} = (\hat{y} - y) \times x_j$$

Input

$$\hat{y} = \sigma(z)$$



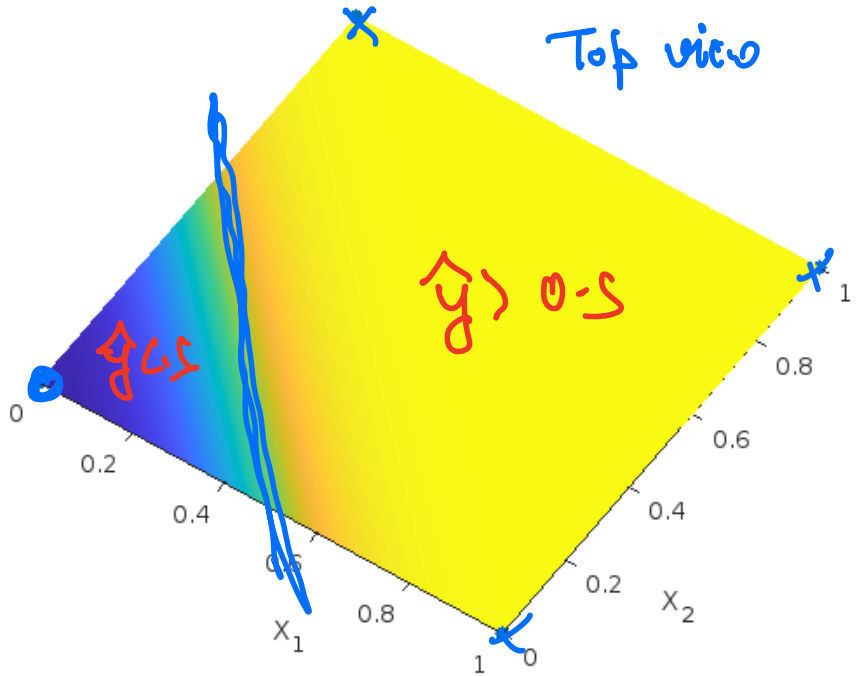


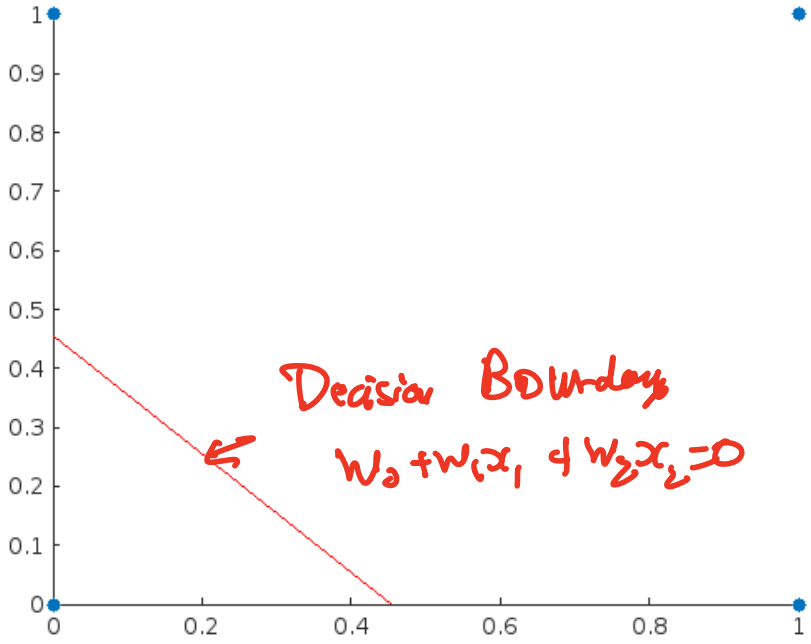
$m = 4$

	x_1	x_2	y
1	0	0	0
2	1	0	1
3	0	1	1
4	1	1	1

$$z = w_0 + w_1 x_1 + w_2 x_2 = \Sigma w_i x_i$$

$$\hat{y} = \sigma(z)$$



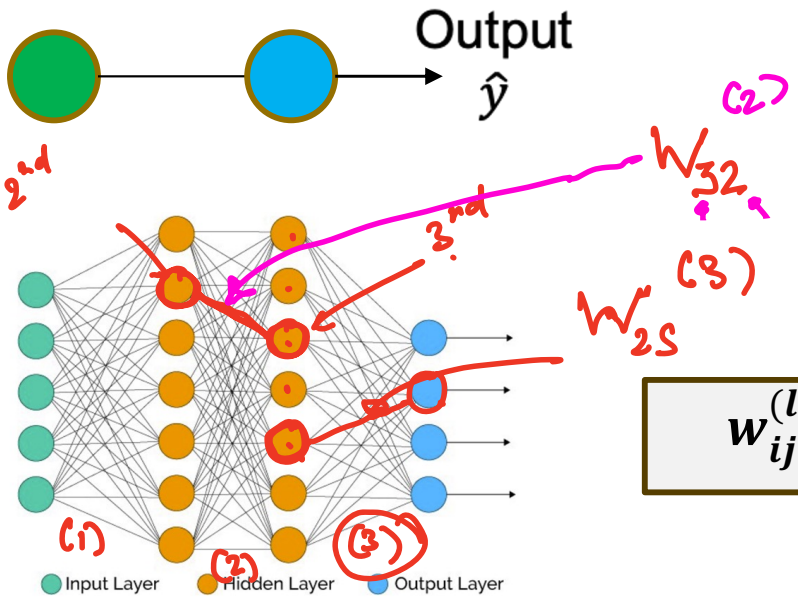


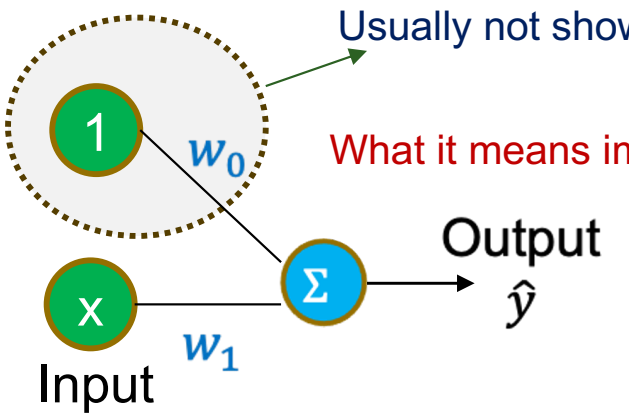
MULTICLASS CLASSIFICATION

(Multinomial Classification)

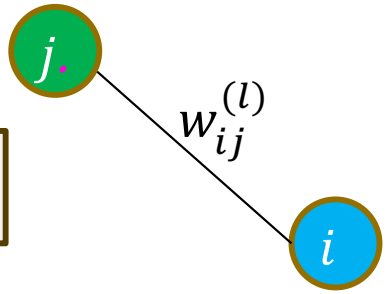
Weights notation

What is shown explicitly





$$w_{ij}^{(l)} \rightarrow w_{\text{to,from}}^{(layer)}$$



- Constant neurons representing a “feature” with 1 are usually not depicted in the pictorial representation.
- The parameters going from the constant neuron is called a bias and the neuron is called a “bias unit”
- The other parameters are called “weights”

Multiclass classification

$K = 2$ Binary Classification

$K = 3$ Multiclass Classification

- K : Number of classes.
  - K =2 Binary classification
  - K > 2 Multiclass (Multinomial) classification
- ML Recipe – Requires the following
  - Data Representation – How do we represent the output (it cannot be 0 or 1)
  - Model – What is the nonlinearity instead of sigmoid? (That was used to lie between 0 and 1)
  - Loss – What is the Loss function?
  - Optimization – Gradient Descent but What is the gradient for the new model?

Let us look at these issues one by one

Data Representation – One Hot Vector

$K = 2$ Binary Classification

$y = 0$ $y = 1$

Binary Representation

Scalar

p(Dog)

$K = 3$ Multiclass Classification

Ground Truth y

1	0	0

0	**1**	0

0	0	**1**

One Hot Representation

Image Not

- "One-Hot Representation" – Represent the output as a vector instead of a scalar
- Each output class is represented by a 1 in one location and 0 in others – One Hot
- We can interpret the entries in each box as the probability of the class

Prob distribution

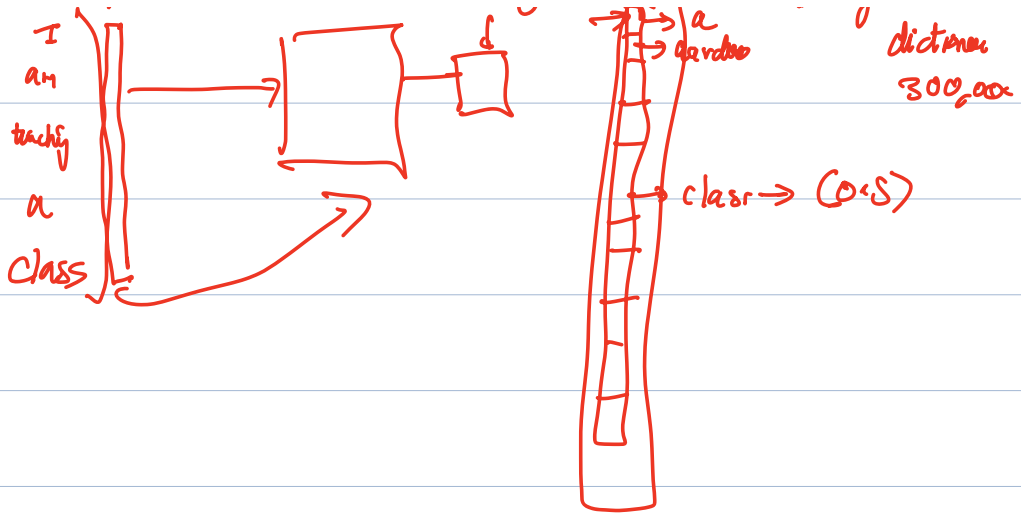
0.4	**0.5**	**0.1**

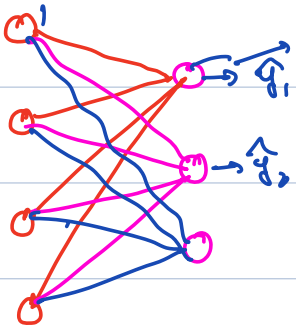
Model Output $\hat{y}$ representation

p(Cat) p(Dog) p(Horse)

For both y and $\hat{y}$, every entry represents $p(y_i = 1|x)$

$\hat{y}$ $K = ?$ Size of



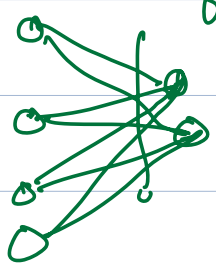


$Z_1 = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4$ $\quad \hat{y}_1 = \sigma(Z_1)$ $\in [0,1]$

$\hat{y}_2$ $Z_2 = w_5 x_1 \ldots + w x_4$ $\quad \hat{y}_2 = \sigma(Z_2)$ $\in [0,1]$

$Z_3 = w_9 x_1 \quad + w_{12} x_4$ $\quad \hat{y}_3 = \sigma(Z_3)$

9 weights



$J(\hat{y}) =$

$\hat{y}(z)$

Model Nonlinearity -- Softmax

0.4	0.5	0.1
p(Cat)	p(Dog)	p(Horse)

$$\hat{y} = [0.4 \quad 0.5 \quad 0.1]$$

$$= [\hat{y_1} \quad \hat{y_2} \quad \hat{y_3}]$$

	z	$\sigma(z)$	Softmax(z)
z_1	1	0.7311	0.1191
z_2	3	0.9526	0.8801
z_3	-4	0.0180	0.0008
SUM Σ		1.7016	1.0000

Not = 1

- We can define $z = \Sigma w_i x_i$ as before
- However, we cannot use $\hat{y} = \sigma(z)$ since we would not have $\Sigma \hat{y_i} = 1$
- To guarantee that the output can be interpreted as a probability we introduce softmax.
- This is defined as

$$sf_1 = \frac{e^{z_1}}{D}, \quad sf_2 = \frac{e^{z_2}}{D}, \quad sf_3 = \frac{e^{z_3}}{D} \qquad sf_1 + sf_2 + sf_3 = \frac{D}{D} = 1$$

$$\text{softmax}(z_1) = \frac{\exp(z_1)}{(\exp(z_1) + \exp(z_2) + \exp(z_3)} \qquad D$$

Normalizing const

- This has all outputs between 0 and 1 as well as $\Sigma \hat{y_i} = 1$

Categorical Cross-Entropy (CCE) Loss Function

- Recall the binary cross entropy loss function

$$J^{BCE}(y, \hat{y}) = -y \ln \hat{y} - (1 - y) \ln (1 - \hat{y})$$

- We use a generalization of this for the multiclass case

$$J^{CCE}(y, \hat{y}) = -\Sigma_{i=1}^{K} (y_i \ln \hat{y}_i)$$

Categorical Cross Entropy

- What happens if apply CCE to the binary case with one-hot?

- $J^{CCE}(y, \hat{y}) = -\Sigma_{i=1}^{K} (y_i \ln \hat{y}_i) = -y_1 ln\hat{y_1} - y_2 ln\hat{y_2}$
- In the binary case $\hat{y_2} = (1 - \hat{y_1})$ and $y_2 = (1 - y_1)$

$$\Rightarrow J^{CCE}(y, \hat{y}) = -y_1 ln\hat{y_1} - (1 - y_1) \ln(1 - \hat{y_1})$$

- This is identical to the binary cross entropy loss!

1 Binary 0

1 0 One-hot 0 1

y_1 y_2 y_1 y_2

$y_1 = 1 - y_2$; $y_2 = 1 - y_1$

$\hat{y}_1 = 1 - \hat{y}_2$; $\hat{y}_2 = 1 - \hat{y}_1$

$$-\sum_{i=1}^{2} y_i \ln \hat{y}_i = -[y_1 \ln \hat{y}_1 + y_2 \ln \hat{y}_2]$$

Multiclass

$$= -[y_1 \ln \hat{y}_1 + (1-y_1) \ln (1-\hat{y}_1)]$$

$$= -y_1 \ln \hat{y}_1 - (1-y_1) \ln (1-\hat{y}_1)$$

BCE

$$-y_1 \ln \hat{y}_1 - y_2 \ln \hat{y}_2 - y_3 \ln \hat{y}_3$$

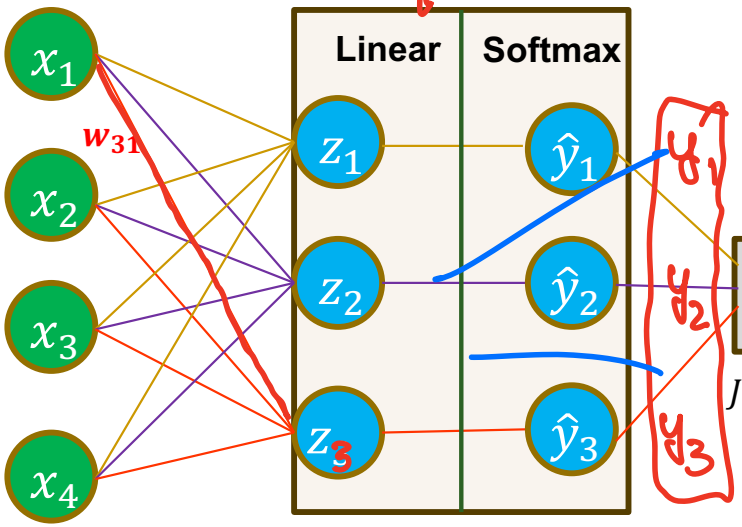
$$[1-(y_1+y_2)] \qquad [1-(\hat{y}_1+\hat{y}_2)]$$

$$0 \le \text{Sf}(z) \le 1$$

$$\frac{e^{z_i}}{\sum e^{z_i}}$$

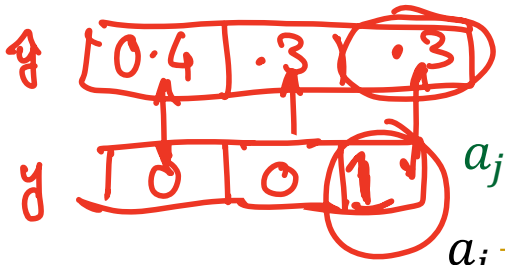
$$\frac{\partial J}{\partial w_j} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial w_j}$$

Optimization – Gradients in the Multiclass case



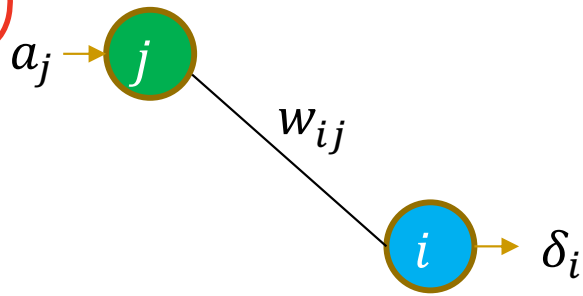
Neuron

$J^{CCE}(y, \hat{y}) = -\Sigma_{i=1}^{K}(y_i \ln \hat{y}_i)$

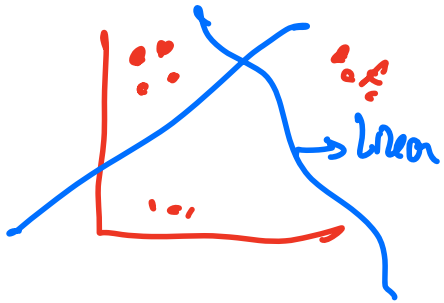


$\delta_3 = (.3-1) = -.7$ #255

$a_j = x_j$ Activation of Input



$\delta_i = \hat{y}_i - y_i$ Error in output



- It can be shown that $\frac{\partial J}{\partial z_1} = \hat{y}_1 - y_1$. Similarly, $\frac{\partial J}{\partial z_i} = \hat{y}_i - y_i$

Not derived

  - Same $\frac{\partial J}{\partial z}$ expression as linear and logistic

- $\frac{\partial J}{\partial w_{31}}$ = Error in Output 3×Activation of Input 1=$\delta_3 \times a_1 = (\hat{y}_3 - y_3) \times x_1$

Error in output

General Delta Rule $\frac{\partial J}{\partial w_{ij}} = \delta_i a_j$

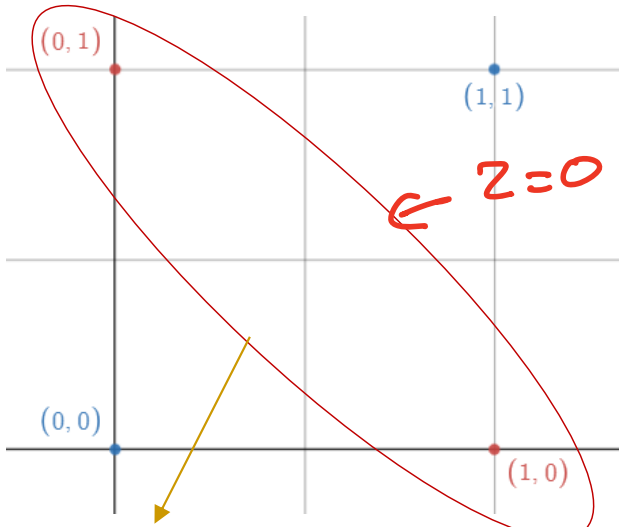
Activation of corresponding input

No hidden layers → No nonlinear decision bds

The XOR problem – Need for deep networks

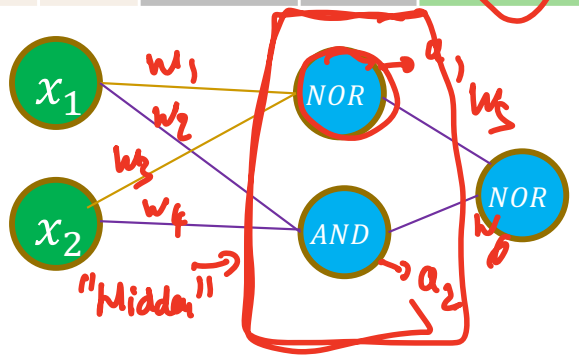
	x_1	x_2	y
1	0	0	0
2	1	0	1
3	0	1	1
4	1	1	0

$\hat{y} = x_1 + x_2 - 2x_1x_2$



$z = 0$ decision boundary is not a line

x_1	x_2	NOR a_1	AND a_2	NOR(NOR,AND) NOR(a_1,a_2)
0	0	1	0	0
1	0	0	0	1
0	1	0	0	1
1	1	0	1	0



- In all cases so far, we map the input to $z = w_0 + w_1x_1 + \cdots$
- The decision boundary, $z = 0$ will, therefore, always be a line
  - If we directly map $x \rightarrow z$ like so far, it only works for **linearly separable data**
- Two solutions
  - Use nonlinear features. Example -- $\hat{y} = w_0 + w_1x_1 + w_2x_2 + w_3x_1x_2$
  - Use "deep" networks.

$x_3 = x_1x_2$ Nonlinear feature

- We can use $XOR(x_1, x_2) = NOR\big(NOR(x_1, x_2), AND(x_1, x_2)\big)$ for example
- But, can we automatically come up with this representation via learning?

Two approaches

(1) Fix the input → Nonlinear features

SVM → "Kernel" trick

Feature Engineering

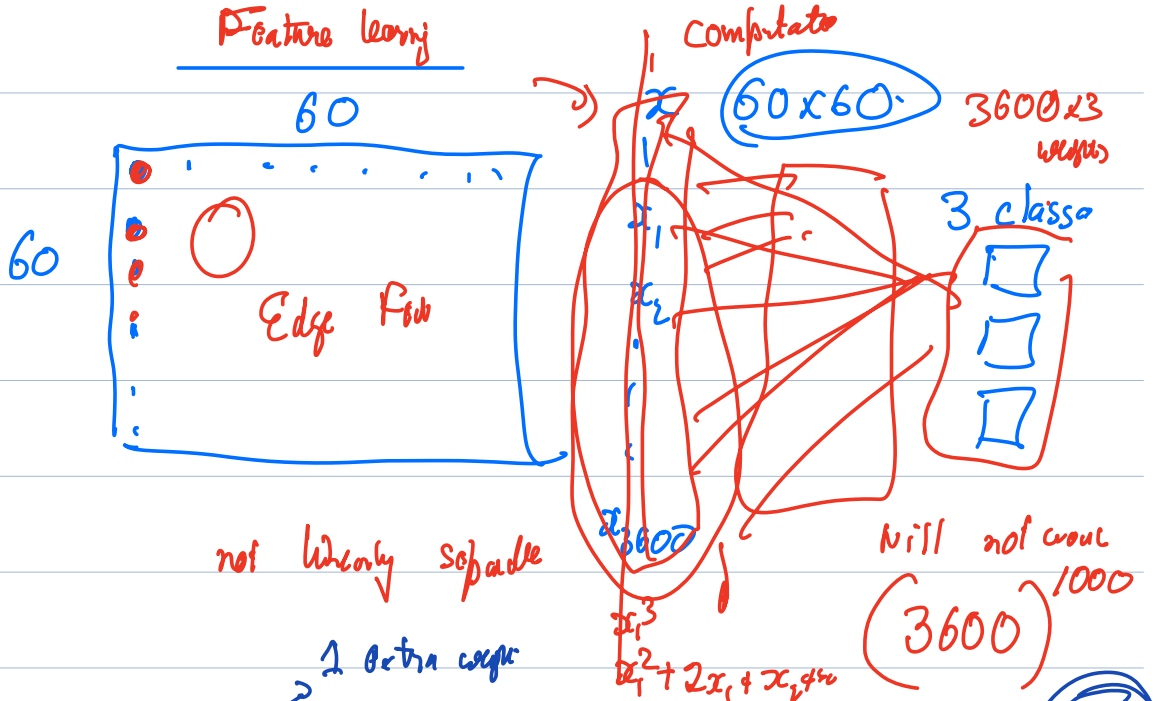
x_1, x_2, x_3

x_1x_2 x_2x_3 x_3x_1 → Curse of dimensionality

x_1^2 x_2^2 x_3^2

(2) Fix the process → Deep networks

↗ → Add more "intermediate"



1 extra weight

1 feature $w_0 + w_1 x + w_2 x^2$

↳ Quadratic

1 → 3 → 6

N^D

3 features

$x_1^2, x_2^2, x_3^2, x_1x_2, x_2x_3, x_3x_1$

6.

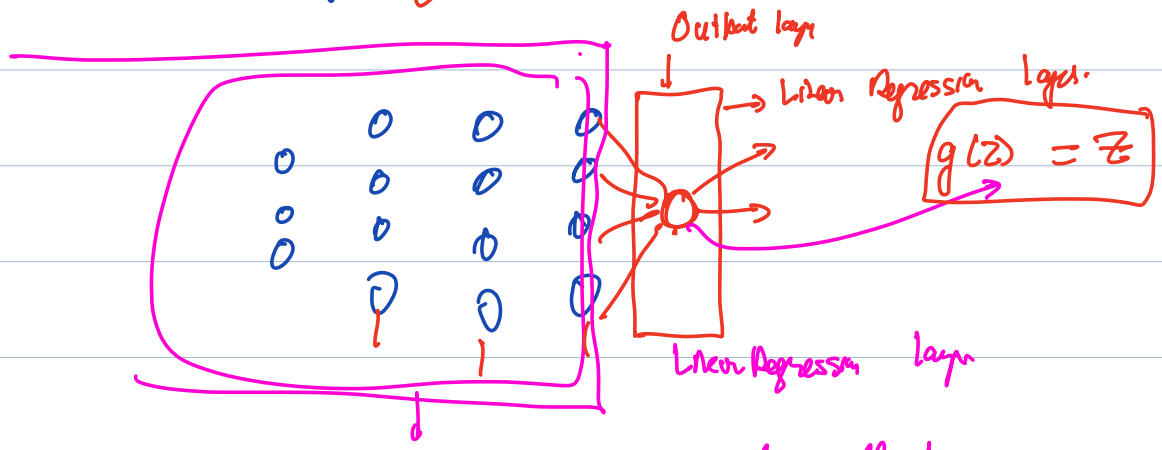
2 features x_1, x_2 → $w_0 + w_1x_1 + w_2x_2$

3 for quadratic

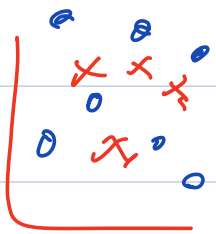
Quadratic $w_0 + w_1x_1 + w_2x_2$

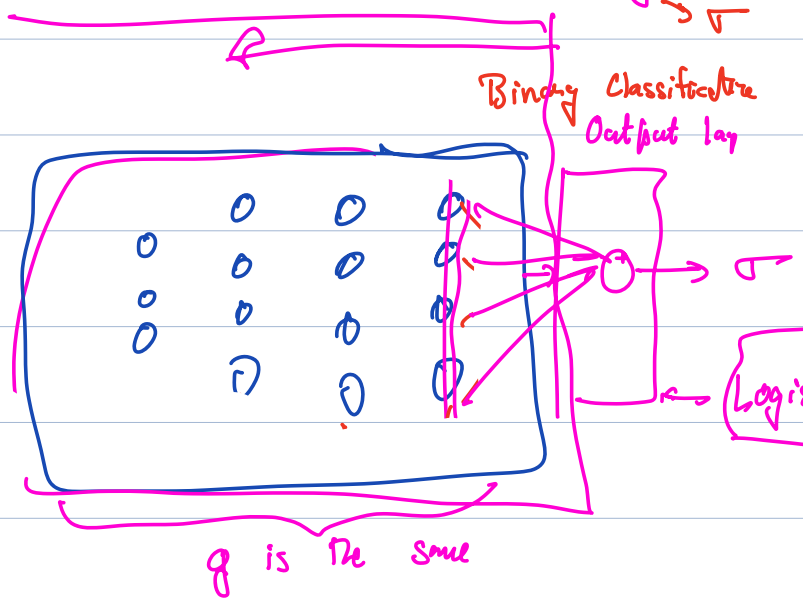
$+ w_3x_1^2 + w_4x_2^2 + w_5x_1x_2$

Regression, no limits on $\hat{y}$.



Use the same g for all layers





g is the same