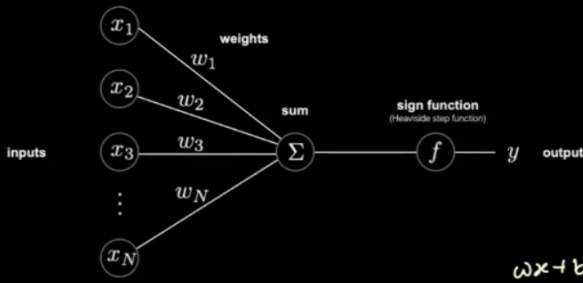
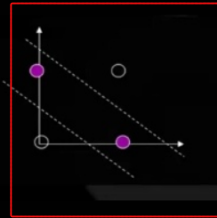


Feed forward Neural Networks

The Perceptron

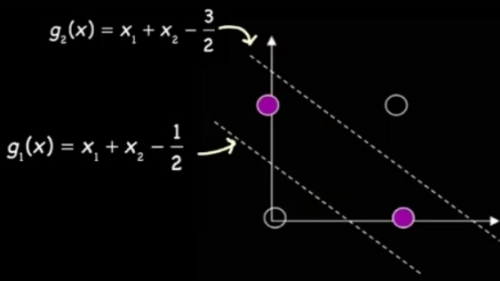


→ Perceptron can solve linearly separable.



XOR problem can't be solved since the data is not linearly separable

→ feed forward neural n/w is an extension of perceptron that can solve such problems.



Using combination of two hyperplane $g_1(x) \neq g_2(x)$

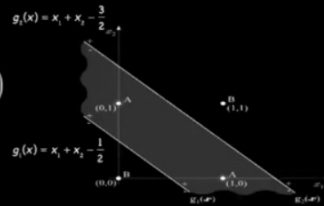
Let $f(x)$ be the unit step activation function:
 $f(x) = 0, x < 0$
 $f(x) = 1, x \geq 0$

Observe that the classification problem is then solved by

$$f\left(y_1 - y_2 - \frac{1}{2}\right)$$

where

$$y_1 = f(g_1(x)) \text{ and } y_2 = f(g_2(x))$$

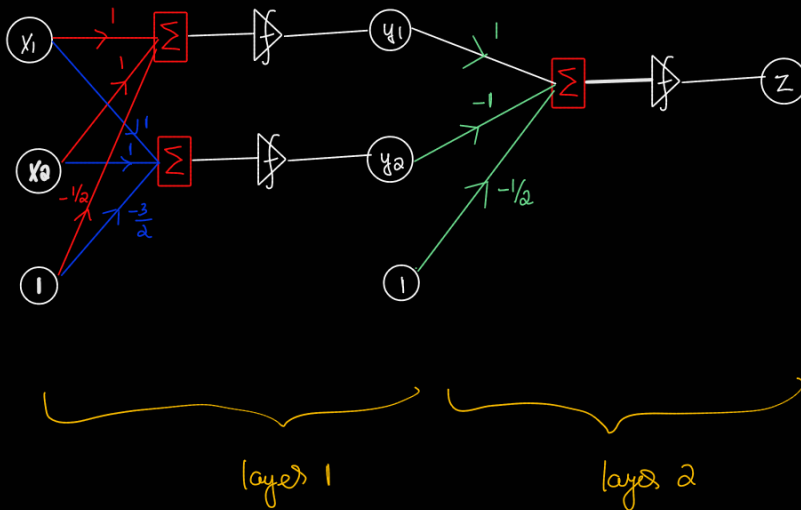


□ This calculation can be implemented sequentially:

1. Compute y_1 and y_2 from x_1 and x_2 .
2. Compute the decision from y_1 and y_2 .

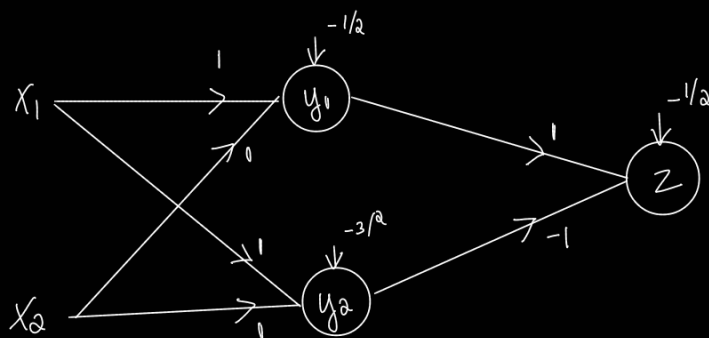
□ Each layer in the sequence consists of one or more linear classifications.

□ This is therefore a two-layer perceptron.

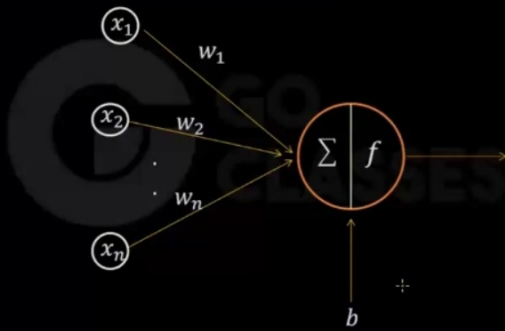


$$z = x_1 \oplus x_2$$

XOR using a 2 layer neural n/w



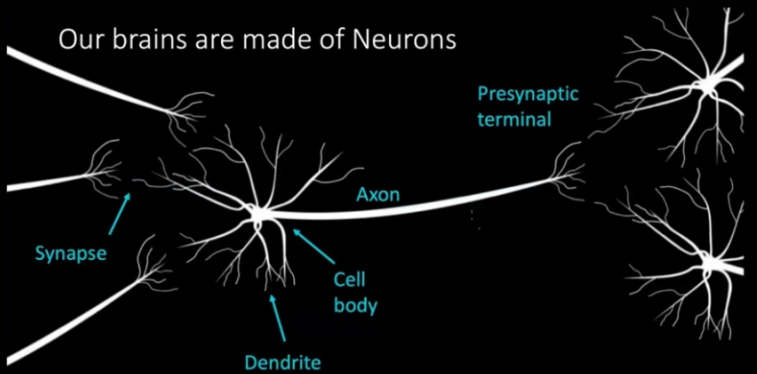
one neuron: (generalisation of perceptron)



f is an activation function.

→ if activation is step function then it is a perceptron

Our brains are made of Neurons



→ Biological neuron was less understood when the perceptron or neuron was first formulated

→ Now, we know the biological neuron is much more complicated than a simple neuron

→ it takes multiple layers of artificial neurons to accurately model even one single biological neuron.

1. Neuron (in Machine Learning)

A neuron (also called an *artificial neuron*) is a basic computational unit in a neural network – inspired by biological neurons.

General structure:

Each neuron:

- Takes multiple inputs x_1, x_2, \dots, x_n
- Multiplies each input by a weight w_i
- Adds a bias b
- Passes the sum through an activation function $f(\cdot)$

$$\text{Output } y = f(w_1x_1 + w_2x_2 + \dots + w_nx_n + b)$$

Key point:

- The activation function can be *non-linear* (like ReLU, sigmoid, tanh, etc.).
- Neurons are used in **multi-layer neural networks**.
- They can represent **complex non-linear relationships**.

2. Perceptron

The **perceptron** is a specific type of neuron – the *simplest form* of an artificial neuron, introduced by Frank Rosenblatt (1958).

Formula:

$$y = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

Characteristics:

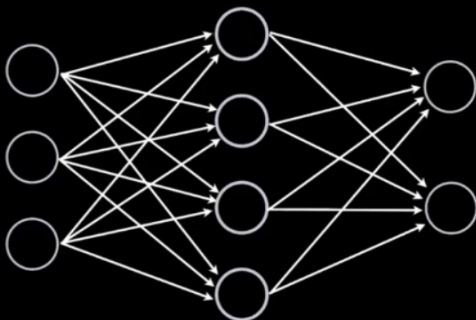
- Uses a **step function** as its activation (a hard threshold).
- Produces a **binary output** (0 or 1).
- Represents a **linear classifier** – it can only separate data with a straight line (or hyperplane).

3. Main Differences Summary

Feature	Perceptron	Neuron (in Neural Networks)
Invented by	Frank Rosenblatt (1958)	Evolved later in multi-layer networks
Activation Function	Step (binary output)	Any differentiable function (sigmoid, ReLU, tanh, etc.)
Output	Binary (0 or 1)	Continuous (real values)
Learning Capability	Linear decision boundary only	Non-linear relationships
Used In	Single-layer models	Multi-layer neural networks (Deep Learning)
Training	Perceptron learning rule	Gradient Descent / Backpropagation


Neural Network

This network is also called a **Multi-layer Perceptron (MLP)** or *feed forward neural network*.



A **Multi-Layer Perceptron (MLP)** is a specific type of **Feed-Forward Neural Network (FNN)**.

In other words:

 **All MLPs are feed-forward neural networks, but not all feed-forward neural networks are MLPs.**

Detailed Comparison

Aspect	Feed-Forward Neural Network (FNN)	Multi-Layer Perceptron (MLP)
Definition	A general class of neural networks where information moves only <i>forward</i> – from input to output – without cycles.	A specific kind of feed-forward network made up of <i>fully connected layers</i> of perceptrons (neurons).
Structure	Can include different kinds of layers (dense, convolutional, dropout, etc.).	Consists only of dense (fully connected) layers.
Flow of data	Always one-way (no feedback or recurrence).	Also one-way; it's a feed-forward architecture.
Connections	May have specialized connections or architectures (e.g., convolutional, residual).	Every neuron in one layer connects to all neurons in the next layer.
Activation Functions	Can vary (ReLU, softmax, etc.).	Typically non-linear (ReLU, sigmoid, tanh, etc.) – but traditionally sigmoid/tanh.
Example Use Cases	Broad: CNNs, transformers (forwards), MLPs all fall under this umbrella.	Classic tabular data, small-scale regression/classification tasks.

Conceptually

- **Feed-Forward Neural Network = the family**
→ describes *how data flows* (only forward).
- **Multi-Layer Perceptron (MLP) = a specific architecture** in that family
→ describes *how layers are connected* (fully connected, stacked perceptrons).

2. Types of Neural Networks That Are FNNs

These are examples that **fall under** the FNN umbrella:

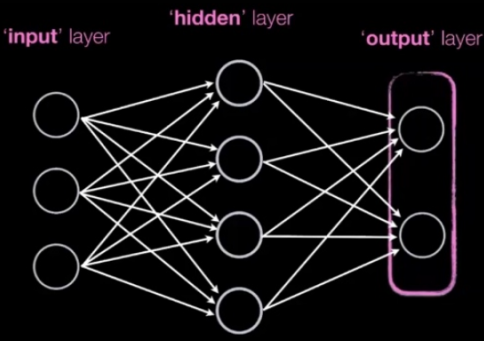
Category	Example	Description
Fully Connected / Dense	Multi-Layer Perceptron (MLP)	Classic architecture; every neuron in one layer connects to all neurons in the next.
Convolutional	CNN (Convolutional Neural Network)	Feed-forward network specialized for images; uses convolution + pooling layers.
Transformer / Attention-based	Transformers, BERT, GPT	Feed-forward sequence models that rely on attention, not recurrence.
Autoencoders	Basic Autoencoder, Variational Autoencoder (VAE)	Encoder-decoder structure, but still feed-forward in flow direction.
Residual / Skip Connection Nets	ResNet	Still feed-forward even with skip connections — because no <i>loops</i> exist.

Even though CNNs and Transformers *look different structurally*, their data still flows **forward only**, so they qualify as FNNs.

3. Neural Networks That Are Not Feed-Forward

These have **feedback loops** or **recurrent connections**, so they are **not** FNNs.

Type	Example	Why not Feed-Forward
Recurrent Neural Networks (RNNs)	RNN, LSTM, GRU	Output from one timestep feeds back into the network for the next timestep.
Recursive Neural Networks	Tree-RNNs (used in NLP parse trees)	Apply same weights recursively along tree structures (feedback-like).
Hopfield Networks	Classical associative memory networks	Have cyclic connections — energy-based, recurrent.
Boltzmann Machines / RBMs	Restricted Boltzmann Machine	Use bidirectional connections — not purely forward.
Echo State Networks / Liquid State Machines	Reservoir computing	Include dynamic feedback within reservoir layer.



fully connected \equiv Dense \equiv MLP

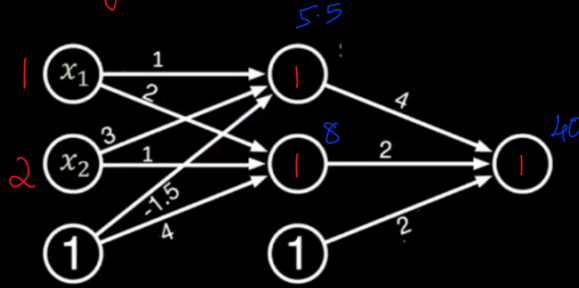
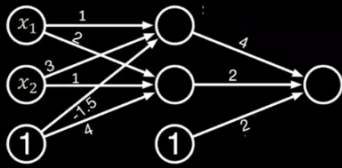
6 neurons \rightarrow Do not count input layer as neuron
 4×3 weights for hidden layers
 4×2 weights for o/p layers
 26 learnable parameters
 $12+8$ weights
 \rightarrow Not counting Bias

#Learnable parameters = #weights + #neurons

#neurons = #bias terms

What is the output of the network for $x_1 = 1, x_2 = 2$.

\rightarrow Assuming step activation & using identity function



identity function
 $f(x) = x$



Activation function

\rightarrow step function / threshold function / Heaviside step / sign

$$f(z) = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0 \end{cases}$$

\rightarrow sigmoid function / logistic function

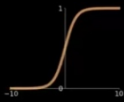
$$f(z) = \frac{1}{1+e^{-z}}$$

\rightarrow identity function / none function

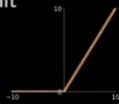
$$f(z) = z$$

Activation Functions

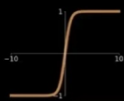
Sigmoid
 $\sigma(x) = \frac{1}{1+e^{-x}}$



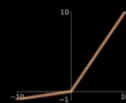
Rectified Linear Unit
ReLU
 $\max(0, x)$



tanh
 $\tanh(x)$



Leaky ReLU
 $\max(0.1x, x)$



ReLU / Leaky ReLU is most popular

0.1 is not fixed but 0.1 is most common

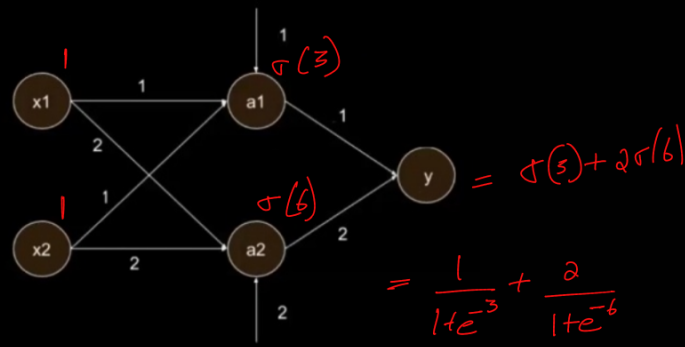
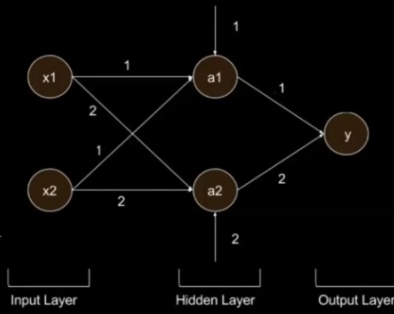
Softmax / Sigmoid is generally used in o/p layer for regression

$\rightarrow \text{Tanh}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

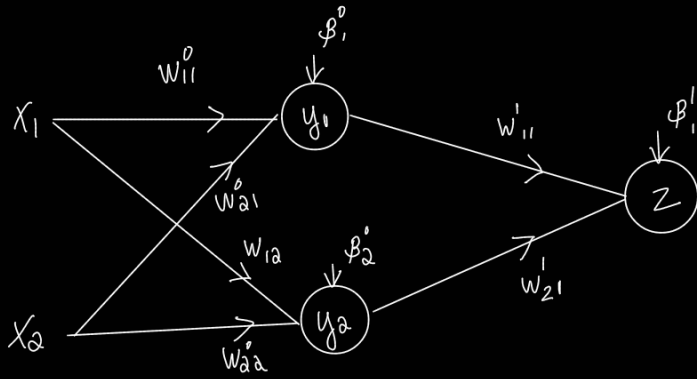
Question:

What is the output of the network for $x_1 = 1, x_2 = 1$?

Assume sigmoid is being used at a_1 and a_2 . and at y , no activation



General case



O/p z is a function of input & weights.

$$\hat{z} = H(x_1, x_2, \underbrace{w_{11}, w_{21}, w_{12}, w_{22}}_{\text{weights}}, \underbrace{\beta_1, \beta_2}_{\text{weights}}, w'_{11}, w'_{21}, \beta_1)$$

where $H \rightarrow$ hypothesis function

\rightarrow Now we have to take a loss function depending on the application

\rightarrow eg. for regression we have MSE as the Loss

$$L(x, w) = \sum_{i=1}^n (z_i - \hat{z}_i)^2$$

$z_i \rightarrow$ label data

\rightarrow general strategy is to differentiate the Loss function to get the update rule for training.

Question 6: Neural Networks (12 points)

Can a neural network be used to model the following machine learning algorithms? If so, state the neural network structure (how many hidden layers are required) and the activation function(s) used at the internal and output nodes. If not, describe why not in one or two sentences.



(a) (3 points) Linear regression

Solution: only one neuron required with identity activation

(b) (3 points) Logistic regression

Solution: only one neuron required with sigmoid activation

(c) (3 points) k-nearest neighbors

Solution: \rightarrow Not possible, since K-NN is non parametric model.

But whatever k-NN can do Neural networks can do

Suppose that we are performing a prediction task using artificial neurons. Which of the following statements about artificial neurons is TRUE?

- Linear regression uses the identity activation function with the squared error loss function.
- Logistic regression uses the identity activation function with the binary cross entropy loss function.

Question:

Consider the unit shown on Figure 1. Suppose that the weights corresponding to the three inputs have the following values:

$$\begin{matrix} w_1 & = & 2 \\ w_2 & = & -4 \\ w_3 & = & 1 \end{matrix}$$

and the activation of the unit is given by the step-function:

$$\varphi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Calculate what will be the output value y of the unit for each of the following input patterns:

Pattern	P_1	P_2	P_3	P_4
x_1	1	0	1	1
x_2	0	1	0	1
x_3	0	1	1	1

↓ ↓ ↓ ↓
1 0 1 0

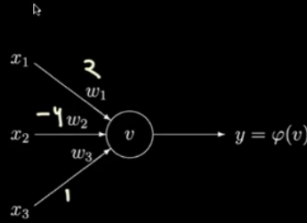


Figure 1: Single unit with three inputs.

★ i. Which of the following activation functions, when applied to the nodes in the hidden layers of a multilayer perceptron, is not useful for a deep learning model? (0.25 points)

- A. $\frac{e^x - e^{-x}}{e^x + e^{-x}}$ — tanh
- B. $-\min(4, x)$
- C. $-0.4x + 0.2$
- D. $\max(0.1x, x)$ — leaky relu

★★★★

Think extreme case → All layers have linear activation
 ⇒ entire NN becomes linear in inputs ⇒ useless — just a linear regression

↔ Linear functions are not helpful
 Use identity why to create burden

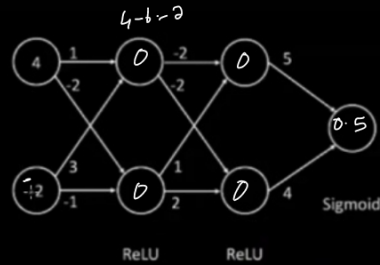
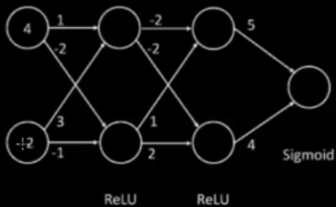
↔ Linear function as activation is redundant

↔ Use of activation is to learn the non linearity in the model

Question 2: Neural Network Output (1 point)

What is the output probability prediction of the following neural network given the input vector shown? Show your work.

Both hidden layers have ReLU activation applied to every node. The output layer uses sigmoid activation (for binary classification).



GATE DA 2024 Sample Paper Question

#weights = $5 \times 10 + 10 \times 3 = 80$

#neurons = 13 = #bias terms

total learnable = #Weights + #Bias
 = 80 + 13
 = 93

Q.43	Consider a Multi-Layer Perceptron (MLP) model with one hidden layer and one output layer. The hidden layer has 10 neurons, and the output layer has 3 neurons. The input to the MLP is a 5-dimensional vector. Each neuron is connected to every neuron in the previous layer, and a bias term is included for each neuron. The activation function used is the sigmoid function. The total number of trainable parameters in this MLP model is _____.
------	--

Q.43 Consider the two neural networks (NNs) shown in Figures 1 and 2, with $ReLU$ activation ($ReLU(z) = \max\{0, z\}, \forall z \in \mathbb{R}$). \mathbb{R} denotes the set of real numbers. The connections and their corresponding weights are shown in the Figures. The biases at every neuron are set to 0. For what values of p, q, r in Figure 2 are the two NNs equivalent, when x_1, x_2, x_3 are positive?

Figure 1

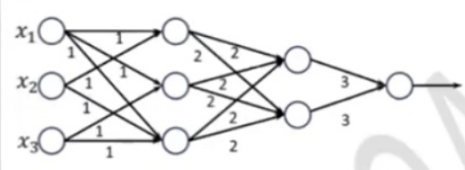
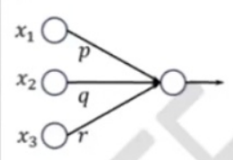


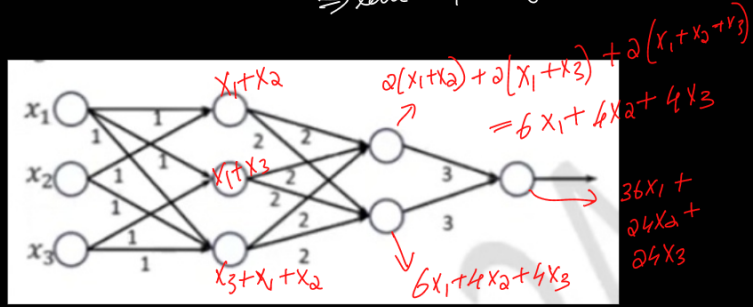
Figure 2



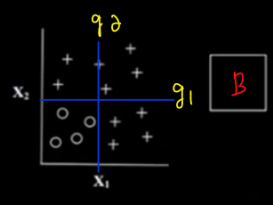
$p = 36$
 $q = 24$
 $r = 24$

- (A) $p = 36, q = 24, r = 24$
- (B) $p = 24, q = 24, r = 36$
- (C) $p = 18, q = 36, r = 24$
- (D) $p = 36, q = 36, r = 36$

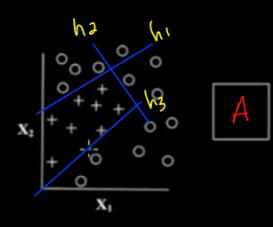
\rightarrow if x_1, x_2, x_3 +ve } \Rightarrow All relu activation will be on the RHS of WX plane
 \rightarrow All weights +ve \Rightarrow relu = identity



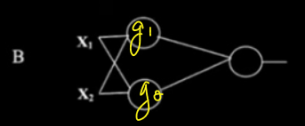
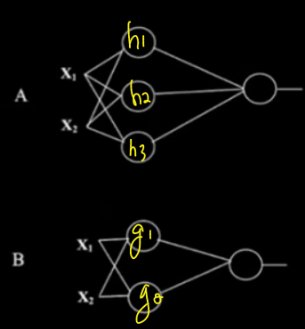
Recall that the output of a perceptron is 0 or 1. For each of the two following data sets, select the perceptron network with the fewest nodes that will separate the classes, and write the corresponding letter in the box. You can use the same network more than once.



B

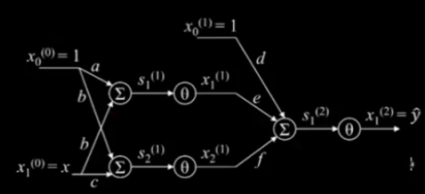


A



\rightarrow Each Neuron in the hidden layers learns one intermediate hyperplane

4. (15 MARKS) Consider below a two-layer neural network with 3 neurons, which takes scalar input x and outputs y . Here Σ denotes the summation operator and θ denotes the activation function. The symbols $a, b, c, d, e,$ and f represent the weights on their respective edges. Note that $x_0^{(1)} = 1$ is the bias input into the second hidden layer, while $x_0^{(2)} = 1$ is the bias input into the first hidden layer.



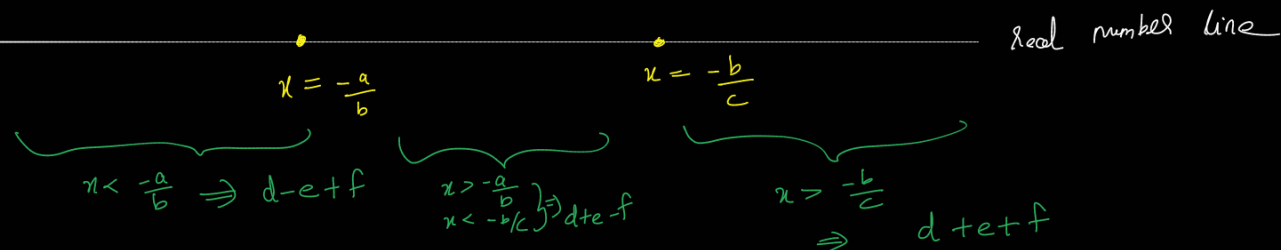
$$x_1^{(1)} = \text{Sign}(a+bx)$$

$$x_2^{(1)} = \text{Sign}(b+cx)$$

$$x_1^{(2)} = \text{Sign}(d + e \text{Sign}(a+bx) + f \text{Sign}(b+cx))$$

Suppose the neural network is used for binary classification, with $\theta(s) = \text{sign}(s) \in \{-1, +1\}$ and $a = -1.5, b = 1, c = 2, d = 1.5, e = 1,$ and $f = -1$. On the real line, sketch the decision rule accomplished by this neural network, i.e., the different regions for input x where we have output $\hat{y} = -1$ and $\hat{y} = +1$.

At DB, $d + e \cdot \text{Sign}(a+bx) + f \cdot \text{Sign}(b+cx) = 0$

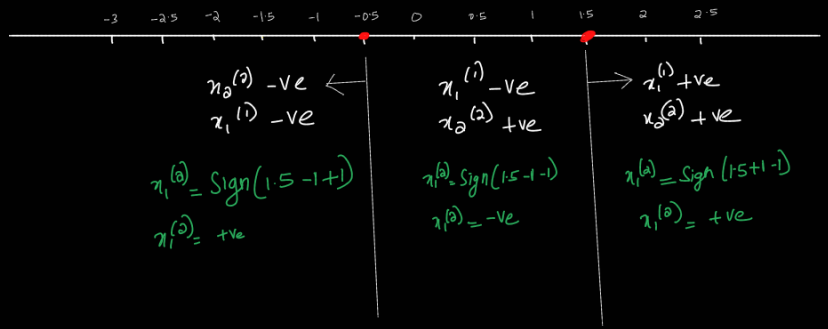


Specific case $a = -1.5, b = 1, c = 2, d = 1.5, e = 1, f = -1$

$$x_1^{(1)} = \text{Sign}(x - 1.5) \quad \text{critical @ } x = 1.5$$

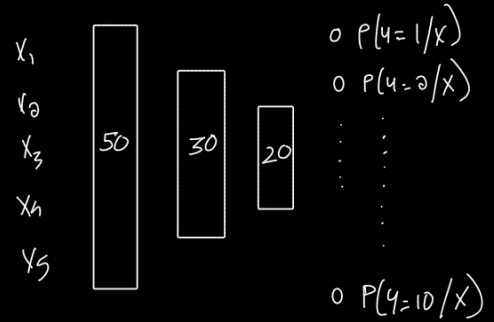
$$x_2^{(1)} = \text{Sign}(2x + 1) \quad \text{critical @ } x = -1/2$$

$$x_1^{(2)} = \text{Sign}(1.5 + \text{Sign}(x - 1.5) - \text{Sign}(2x + 1))$$



iv. Let's say you have a MLP network which is trained on 5-dimensional input data. The network has 3 hidden layers, with 50, 30, and 20 nodes respectively. The network is trained to predict a 10-way classification task using softmax activation. If we ignore bias terms, how many weights does this neural network have? (0.25 points)

- ✓ A. 2,550
- B. 1,500,000
- C. 115
- D. 30,000



$$\# \text{ weights} = (5 \times 5) + (30 \times 5) + (20 \times 30) + (10 \times 20)$$

$$= 250 + 1500 + 600 + 200 = 2550 \text{ weights}$$

$$\# \text{ neurons} = 50 + 30 + 20 + 10 = 110$$

$$\# \text{ learnable parameters} = 2660$$

Output Layer

Objective	op layer	Activation	Loss function	Remarks
Binary classification	1 neuron	Sigmoid - differentiable or step \rightarrow non differentiable	C.E Loss	
Regression	1 neuron	identity	MSE	
Multiclass classification	# Neurons = # Classes	softmax	C.E Loss	

c. Suppose we have a deep feedforward neural network with two layers. The first layer has four artificial neurons, and the second layer has one neuron. We want to use the network to perform binary classification on the data that is not linearly separable. This means that the dataset cannot be separated with a reasonable performance by just using a linear classifier. Which of the following settings can help us achieve our task well?

1.0 point · Multiple choice · 4 alternatives

- Use the tanh activation function for all neurons and use the cross-entropy loss
- Use the identity activation function for all neurons and use the cross-entropy loss
- Use the tanh activation function for all neurons and use the squared error loss
- Use the identity activation function for all neurons and use the squared error loss

→ we have seen that using identity for all neurons will end up becoming a very costly regression algorithm.

Suppose you are training neural networks on 100×100 images to predict 5 classes. Neural network *A* consists of a single linear layer followed by a softmax output activation. Neural network *B* consists of a sequence of layers with dimensions 128, 512, and 32, respectively, followed by a softmax output activation. Assuming that both neural networks are trained using an identical procedure (e.g. batch size, learning rate, epochs, etc), and neither contains hidden activations, what can you generally expect about the relative performance of *A* and *B* on the test data?

★ No hidden activation is another name for identity function

- (a) *A* will outperform *B*
- (b) *B* will outperform *A*
- (c) *A* and *B* will perform roughly the same.

→ No non-linearity introduced ⇒ Just a dumb regressor (both A & B)
B more costly but equally dumb

If a network has no hidden activation, that means: ★

- There are no hidden layers, or
- The activation functions in hidden layers are missing (i.e., purely linear).

So, effectively:

$$\text{Output} = Wx + b$$

Two historians approach you for your deep learning expertise. They want to classify images of historical objects into 3 classes depending on the time they were created:

- Antiquity ($y = 0$)
- Middle Ages ($y = 1$)
- Modern Era ($y = 2$)

→ softmax activation

$$\hat{y} = [1/3, 1/3, 1/3]$$



(A) Class: Antiquity



(B) Class: Middle Ages



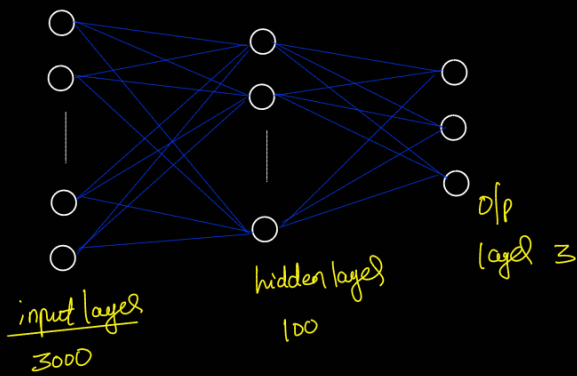
(C) Class: Modern Era

Figure 1: Example of images found in the dataset along with their classes

(3 points) Following the last FC-3 layer of your network, what activation must be applied? Given a vector $a = [0.3, 0.3, 0.3]$, what is the result of using your activation on this vector?

Backpropagation

- Take an neural network with 3000 as input dimension
- 1 hidden layer of 100 dimension
- o/p layer of 3 dimension



Notation for weights W_{ij}^k

k → weight corresponding to k^{th} layer
 $k = 1, 2, 3 \dots \# \text{layers}$

i → weight corresponding to i^{th} input in the previous layer

j → weight corresponding to j^{th} neuron in the k^{th} layer

→ if we consider the input vector as a column vector X then $W_1^T X$ gives the o/p of hidden layer (we are assuming no activation)

→ let $Z = W_1^T X$, then

$$W_2^T Z = Y \quad \text{which is our output}$$

$W_2 \rightarrow 100 \times 3$ matrix

$$\hat{Y} = W_2 \cdot W_1^T X$$

$$3 \times 1 = (3 \times 100) \cdot (100 \times 3000) \cdot (3000 \times 1)$$

→ if we consider our input vector & output vector as row vector

$$\hat{Y} = X W_1 W_2$$

→ if we want we can add bias terms

$$\hat{Y} = W_2^T (W_1^T X + \beta_1) + \beta_2 \quad \text{column notation}$$

$$\hat{Y} = (X W_1 + \beta_1) W_2 + \beta_2 \quad \text{row notation}$$

→ w/o activation

Matrix $W^1 =$

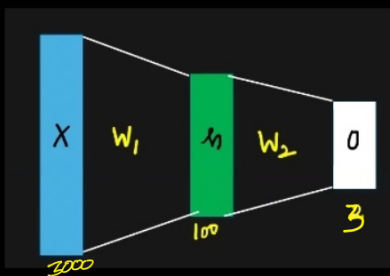
$$\begin{pmatrix} W_{1,1}^1 & W_{1,2}^1 & \dots & W_{1,100}^1 \\ W_{2,1}^1 & W_{2,2}^1 & & W_{2,100}^1 \\ W_{3,1}^1 & W_{3,2}^1 & & W_{3,100}^1 \\ \vdots & \vdots & & \vdots \\ W_{3000,1}^1 & W_{3000,2}^1 & & W_{3000,100}^1 \end{pmatrix} \quad 3000 \times 100$$

W_1 - weight matrix corresponding to 1st layer neuron

→ Each column represents one neuron in the corresponding layer

→ j^{th} column represents j^{th} neuron

→ i^{th} row represents i^{th} input



Introducing Non Linearity

→ Assume some activation, say eg ReLU $f(z) = \max(0, z)$ in the hidden layer

$$z = \max(w_1^T x + \beta_1, 0)$$

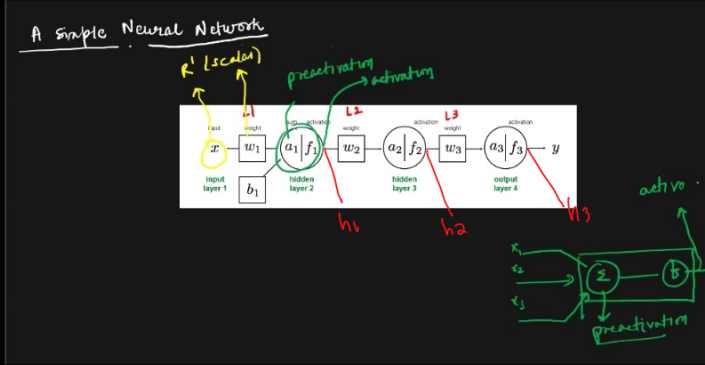
$$\hat{y} = W_a^T z + \beta_2$$

} Combined

$$\hat{y} = W_a^T (\max(w_1^T x + \beta_1, 0)) + \beta_2$$

↳ writing entire neural n/w in one complex equation

Preactivation & Activation



$$a_1 = w_1^T x + \beta_1 \quad h_1 = f_1(a_1) \rightarrow \text{activation}$$

$$a_2 = w_2^T h_1 = w_2^T f_1(a_1) = w_2^T f_1(w_1^T x + \beta_1)$$

$$h_2 = f_2(a_2)$$

$$a_3 = w_3^T h_2 = w_3^T f_2(w_2^T f_1(w_1^T x + \beta_1))$$

↳ Forward pass of neural n/w

$$h_3 = f_3(a_3) = \text{output } \hat{y}$$

$$\hat{y} = f_3(w_3^T f_2(w_2^T f_1(w_1^T x + \beta_1)))$$

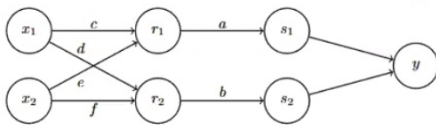
Trainable parameters

[6 pts] Below is a neural network with weights a, b, c, d, e, f . The inputs are x_1 and x_2 . The first hidden layer computes $r_1 = \max(c \cdot x_1 + e \cdot x_2, 0)$ and $r_2 = \max(d \cdot x_1 + f \cdot x_2, 0)$. The second hidden layer computes $s_1 = \frac{1}{1 + \exp(-a \cdot r_1)}$ and $s_2 = \frac{1}{1 + \exp(-b \cdot r_2)}$. The output layer computes $y = s_1 + s_2$. Note that the weights a, b, c, d, e, f are indicated along the edges of the neural network here.

Suppose the network has inputs $x_1 = 1, x_2 = -1$. The weight values are $a = 1, b = 1, c = 4, d = 1, e = 2, f = 2$.

calculate values of:

- ① r_1
- ② r_2
- ③ s_1
- ④ s_2
- ⑤ y



$$r_1 = 4 - 2 = 2$$

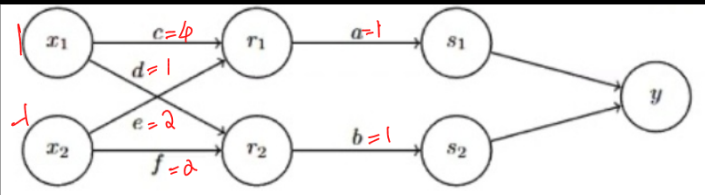
$$r_2 = 2 - 2 = 0$$

$$s_1 = \sigma(2) = \frac{1}{1 + e^{-2}}$$

$$s_2 = \sigma(0) = 1/2$$

$$y = \sigma(2) + \sigma(0) = \frac{1}{1 + e^{-2}} + \frac{1}{2}$$

$$y = 1.380$$



Back propagation

Gradient Descent

1. Initialize the parameters w to some guess (usually all zeros, or random values)
2. Update the parameters:
 $w = w - \eta \nabla L(w)$
3. Update the learning rate η
4. Repeat steps 2-3 until $\nabla L(w)$ is close to zero.

→ Directly using GD in FNN is not practical
→ Calculating the gradient for each trainable parameter will be very time consuming

→ Logistic regression, linear regression } all these shallow learning algo have very convenient form of gradient

→ Each time we change loss function, we have to recalculate gradient

eg. Logistic gradient $x^T (\sigma(w^T x + \beta) - y)$
MSE gradient $2x^T (xw - \hat{y})$
Ridge gradient $2x^T (xw - \hat{y}) + \lambda w$

→ Back propagation makes calculation of gradient very efficient

→ gradient gives in which direction each weight to be updated
 → if we iteratively update weights in the opposite direction of gradient, then this is called training or learning

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

$$w_j = w_j - \alpha \frac{\partial L}{\partial w_j}(x, w)$$

→ BkPg → gives way to efficiently calculate $\frac{\partial L}{\partial w_j}$

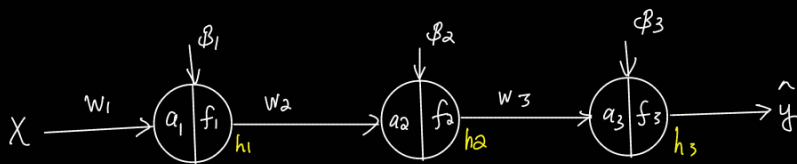
Side quest

<p>Vector Derivatives</p> <p>$x \in \mathbb{R}^N, y \in \mathbb{R}$</p> <p>Derivative is Gradient:</p> <p>$\frac{\partial y}{\partial x} \in \mathbb{R}^N$</p> <p>$(\frac{\partial y}{\partial x})_i = \frac{\partial y}{\partial x_i}$</p> <p>For each element of x, if it changes by a small amount, how much will y change?</p>	<p>$x \in \mathbb{R}^N, y \in \mathbb{R}$</p> <p>Derivative is Gradient:</p> <p>$\frac{\partial y}{\partial x} \in \mathbb{R}^N$</p> <p>$(\frac{\partial y}{\partial x})_i = \frac{\partial y}{\partial x_i}$</p> <p>For each element of x, if it changes by a small amount then how much will y change?</p>	<p>$x \in \mathbb{R}^N, y \in \mathbb{R}^M$</p> <p>Derivative is Jacobian:</p> <p>$\frac{\partial y}{\partial x} \in \mathbb{R}^{N \times M}$</p> <p>$(\frac{\partial y}{\partial x})_{ij} = \frac{\partial y_j}{\partial x_i}$</p> <p>For each element of x, if it changes by a small amount then how much will each element of y change?</p>
--	---	---

→ gradient shows how much the output change if i change each component of x.

→ if y & x is vector, then the gradient matrix is called Jacobian

Consider a simple FNN



$$\hat{y} = f_3(w_3 \cdot f_2(w_2 \cdot f_1(w_1 x + \beta_1) + \beta_2) + \beta_3)$$

$$L = \text{Loss function} = \sum_{i=1}^n (y_i - \hat{y}_i)^2 \rightarrow \text{SSE loss}$$

$$L_i = (y_i - h_3)^2$$

$$h_3 = f_3(a_3)$$

$$a_3 = w_3 h_2 + \beta_3$$

$$\frac{\partial L_i}{\partial w_3} = \frac{\partial L_i}{\partial h_3} \cdot \frac{\partial h_3}{\partial a_3} \cdot \frac{\partial a_3}{\partial w_3} \quad (\text{chain rule})$$

$$\frac{\partial L_i}{\partial h_3} = 2(h_3 - y_i)$$

$$\frac{\partial h_3}{\partial a_3} = 1 \quad (\text{assuming activation identity})$$

$$\frac{\partial a_3}{\partial w_3} = h_2$$

$$\frac{\partial L_i}{\partial w_3} = 2(h_3 - y_i) \cdot h_2$$

$$\frac{\partial L_i}{\partial w_3} = \frac{\partial L_i}{\partial h_3} \cdot \frac{\partial h_3}{\partial a_3} \cdot \frac{\partial a_3}{\partial w_3}$$

$$\frac{\partial L_i}{\partial w_2} = \frac{\partial L_i}{\partial h_3} \cdot \frac{\partial h_3}{\partial a_3} \cdot \frac{\partial a_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial a_2} \cdot \frac{\partial a_2}{\partial w_2}$$

$$\frac{\partial L_i}{\partial w_1} = \frac{\partial L_i}{\partial h_3} \cdot \frac{\partial h_3}{\partial a_3} \cdot \frac{\partial a_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial a_2} \cdot \frac{\partial a_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial a_1} \cdot \frac{\partial a_1}{\partial w_1}$$

$$\frac{\partial L_i}{\partial \beta_1} = \frac{\partial L_i}{\partial h_3} \cdot \frac{\partial h_3}{\partial a_3} \cdot \frac{\partial a_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial a_2} \cdot \frac{\partial a_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial \beta_1} \cdot \frac{\partial \beta_1}{\partial \beta_1}$$

→ The efficiency comes from reusing part of expression

→ This is like dynamic programming.

Gradient-Descent

For each example sample $\{x_i, y_i\}$

1. Forward pass
2. compute loss
3. update

$$\hat{y} = b_{NN}(x_i)$$

$$L_i$$

- a. Back-propagation
- b. gradient-update

$$\frac{\partial L_i}{\partial w_3} = \frac{\partial L_i}{\partial h_3} \cdot \frac{\partial h_3}{\partial a_3} \cdot \frac{\partial a_3}{\partial w_3}$$

$$\frac{\partial L_i}{\partial w_2} = \frac{\partial L_i}{\partial h_3} \cdot \frac{\partial h_3}{\partial a_3} \cdot \frac{\partial a_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial a_2} \cdot \frac{\partial a_2}{\partial w_2}$$

$$\frac{\partial L_i}{\partial w_1} = \frac{\partial L_i}{\partial h_3} \cdot \frac{\partial h_3}{\partial a_3} \cdot \frac{\partial a_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial a_2} \cdot \frac{\partial a_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial a_1} \cdot \frac{\partial a_1}{\partial w_1}$$

$$\frac{\partial L_i}{\partial b_1} = \frac{\partial L_i}{\partial h_3} \cdot \frac{\partial h_3}{\partial a_3} \cdot \frac{\partial a_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial a_2} \cdot \frac{\partial a_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial a_1} \cdot \frac{\partial a_1}{\partial b_1}$$

$$w_3 = w_3 - \eta \nabla w_3$$

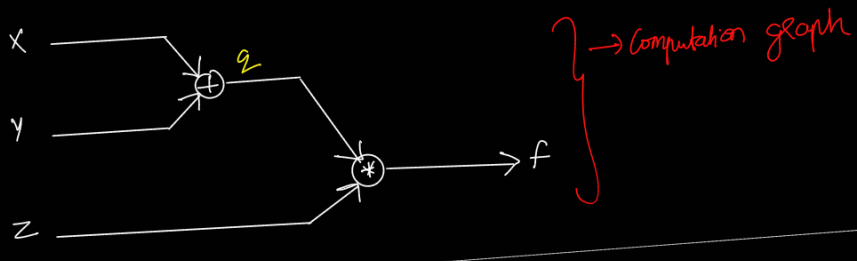
$$w_2 = w_2 - \eta \nabla w_2$$

$$w_1 = w_1 - \eta \nabla w_1$$

$$b = b - \eta \nabla b$$

Computational graph & Back propagation

→ Consider the function $f(x, y, z) = (x+y)z$



Forward pass for $x, y, z = -2, 5, -4$

$$z = x + y = -2 + 5 = 3$$

$$f = z * z = 3 * -4 = -12$$

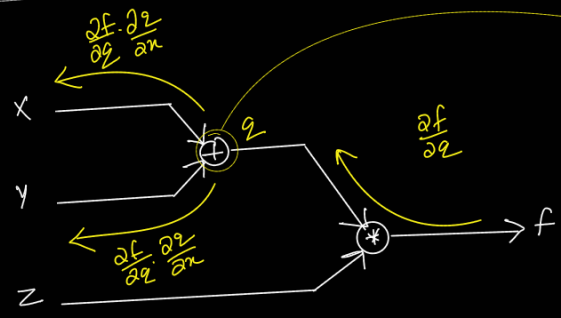
Back propagation

$$\frac{\partial f}{\partial z} = \frac{\partial (z \cdot z)}{\partial z} = z = x + y = 3$$

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial z} \cdot \frac{\partial z}{\partial y} = 3 \cdot 1 = 3$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial z} \cdot \frac{\partial z}{\partial x} = 3 \cdot 1 = 3$$

→ This will be calculated first

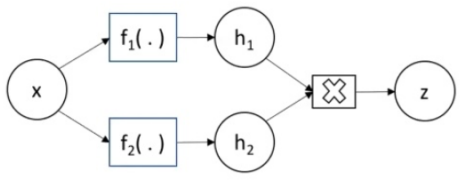


→ Each neuron will have local gradients $\frac{\partial \text{output}}{\partial \text{inputs}}$ (here $\frac{\partial z}{\partial x}$ & $\frac{\partial z}{\partial y}$)
 → The neuron will multiply the back propagating derivatives with local gradient and pass it to the input of the neurons. (here $\frac{\partial f}{\partial z}$)
 ↳ $\frac{\partial f}{\partial z} \cdot \frac{\partial z}{\partial x}$ & $\frac{\partial f}{\partial z} \cdot \frac{\partial z}{\partial y}$

MULTIPLE PATHS IN BACK PROPAGATION

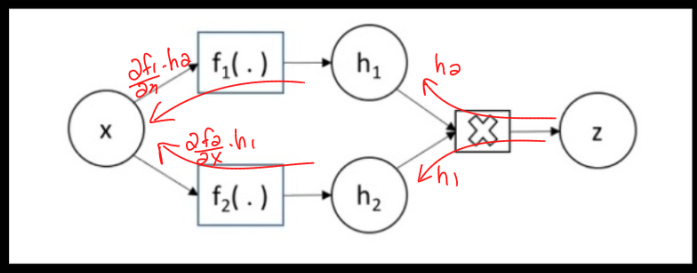
Backpropagation Below is a deep network with input x . Values x, h_1, h_2, z are all scalars.

$$h_1 = f_1(x), h_2 = f_2(x), z = h_1 h_2$$



Q. What of the following options correctly represent the expression of $\frac{\partial z}{\partial h_1}$

- A. h_1
- B. h_2 ✓
- C. $h_1 h_2$
- D. None



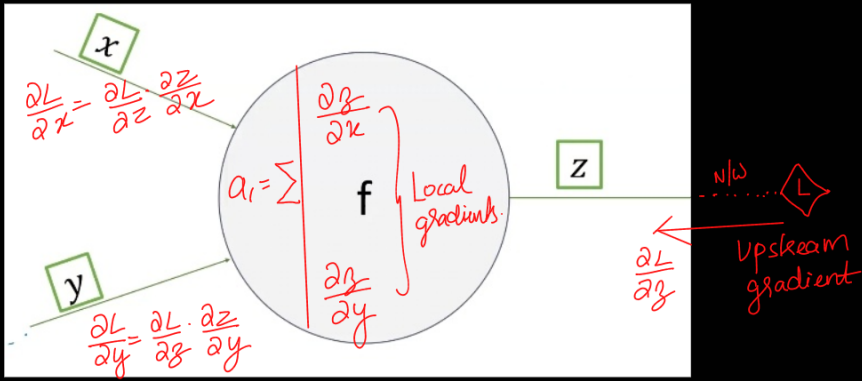
→ Multiplication gates just passes swapped values as partial gradients in back propagation

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial h_1} \cdot \frac{\partial h_1}{\partial x} + \frac{\partial z}{\partial h_2} \cdot \frac{\partial h_2}{\partial x}$$

$$\frac{\partial z}{\partial x} = h_2 \cdot \frac{\partial f_1}{\partial x} + h_1 \cdot \frac{\partial f_2}{\partial x}$$

→ If backpropagation reaches a input or activation using multiple paths, they have to be added.

Local gradient



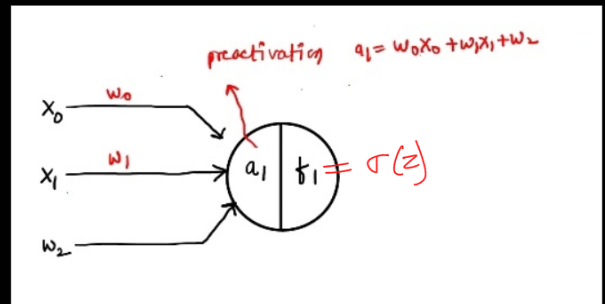
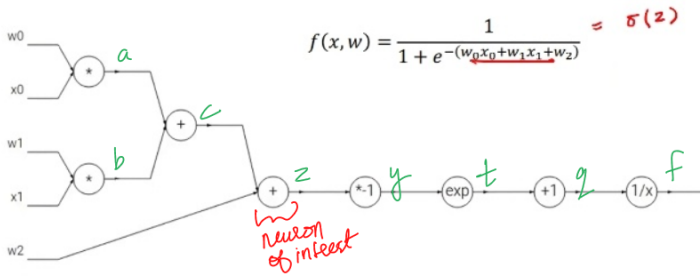
- After the forward pass, we know the inputs & o/p of all neurons
- Local gradient is particular for a single neuron
- So we can evaluate $\frac{\partial \text{output}}{\partial \text{inputs}}$ independent of any other partial gradients immediately after evaluating forward pass.
- We can evaluate these & store in memory before backpropagation

→ During backpropagation → upstream gradient $\frac{\partial L}{\partial z}$ will propagate backwards & reach this particular neuron at its o/p.

→ So the neuron can multiply local gradient with upstream gradient & pass it backwards to the neuron's input layer. so that input gradients are back propagated. } Back propagate

Example

Now let's perform backpropagation through a single neuron of a neural network with a sigmoid activation. Specifically, we will define the pre-activation $z = w_0x_0 + w_1x_1 + w_2$ and we will define the activation value $a = \sigma(z) = 1 / (1 + e^{-z})$. The computation graph is visualized below:



a) Forward pass

$a = w_0x_0$
 $b = w_1x_1$
 $c = w_0x_0 + w_1x_1$
 $z = c + w_2$
 $y = -z$
 $t = e^y$
 $g = t + 1$
 $f = 1/g$

Evaluating @ $w_0 = -2, x_0 = -1, w_1 = -3, x_1 = -2, w_2 = -3$

$a = -2$
 $b = 6$
 $c = 4$
 $z = 1$
 $y = -1$
 $t = e^{-1} = 0.37$
 $g = 1.37$
 $f = 0.73$

b) Calculating local gradient

• $\frac{\partial f}{\partial z} = -1/g^2 = -0.53$

• $\frac{\partial z}{\partial t} = 1$

• $\frac{\partial t}{\partial y} = e^y = 1/e = 0.37$

• $\frac{\partial y}{\partial z} = -1$

• $\frac{\partial z}{\partial w_0} = 1$ $\frac{\partial z}{\partial w_1} = x_1 = -2$ $\frac{\partial z}{\partial w_2} = x_0 = -1$

c) Back propagation w.r.t our neuron

$$\frac{\partial f}{\partial w_0} = \frac{\partial f}{\partial z} \cdot \frac{\partial z}{\partial t} \cdot \frac{\partial t}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial w_0} = (-0.53) \cdot 1 \cdot (0.37) \cdot (-1) \cdot (-1) = -0.39$$

$$\frac{\partial f}{\partial w_1} = \frac{\partial f}{\partial z} \cdot \frac{\partial z}{\partial t} \cdot \frac{\partial t}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial w_1} = (-0.53) \cdot 1 \cdot (0.37) \cdot (-1) \cdot (-2) = 0.79$$

$$\frac{\partial f}{\partial w_2} = \frac{\partial f}{\partial z} \cdot \frac{\partial z}{\partial t} \cdot \frac{\partial t}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial w_2} = (-0.53) \cdot 1 \cdot (0.37) \cdot (-1) \cdot (-1) = -0.39$$

upstream gradient (red arrow)
local gradients (green arrow)

→ Back propagation stores those intermittent values by going in the reverse direction

[6 pts] Below is a neural network with weights a, b, c, d, e, f . The inputs are x_1 and x_2 . The first hidden layer computes $r_1 = \max(c \cdot x_1 + e \cdot x_2, 0)$ and $r_2 = \max(d \cdot x_1 + f \cdot x_2, 0)$. The second hidden layer computes $s_1 = \frac{1}{1 + \exp(-a \cdot r_1)}$ and $s_2 = \frac{1}{1 + \exp(-b \cdot r_2)}$. The output layer computes $y = s_1 + s_2$. Note that the weights a, b, c, d, e, f are indicated along the edges of the neural network here.

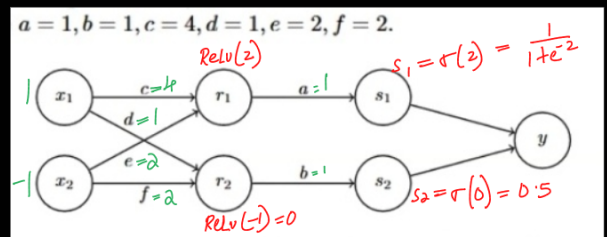
Suppose the network has inputs $x_1 = 1, x_2 = -1$. The weight values are $a = 1, b = 1, c = 4, d = 1, e = 2, f = 2$.

(hw)

Calculate values for each of the derivative

	$\frac{\partial y}{\partial a}$	$\frac{\partial y}{\partial b}$	$\frac{\partial y}{\partial c}$	$\frac{\partial y}{\partial d}$	$\frac{\partial y}{\partial e}$	$\frac{\partial y}{\partial f}$
a)	0.20	0.20	0.20	0.20	0.20	0.20
b)	0.18	0.18	0.18	0.09	0.09	0.18
c)	0.09	0.09	0.09	0.09	0.09	0.09
d)	0	0	0	0	0	0

a) forward pass



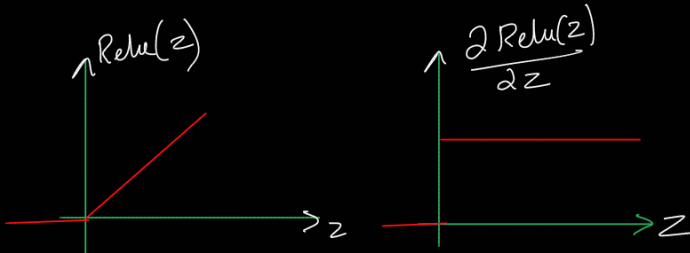
b) local gradients

- $\frac{\partial y}{\partial s_1} = 1, \frac{\partial y}{\partial s_2} = 1, \frac{\partial s_1}{\partial r_1} = \sigma(a r_1) \sigma(-a r_1) \cdot a = \sigma(2) \sigma(-2)$
- $\frac{\partial s_2}{\partial r_2} = \sigma(b r_2) \sigma(-b r_2) \cdot b = \sigma(0) \sigma(0) \cdot 1 = 0.25, \frac{\partial s_1}{\partial r_1} \approx 0.09$
- $\frac{\partial r_1}{\partial x_1} = c = 4, \frac{\partial r_1}{\partial x_2} = e = 2$
- $\frac{\partial r_2}{\partial x_1} = d = 1, \frac{\partial r_2}{\partial x_2} = f = 2$
- $\frac{\partial s_1}{\partial a} = \frac{\partial (\sigma(a r_1))}{\partial a} = \sigma(a r_1) \cdot (1 - \sigma(a r_1)) \cdot r_1 = 2 \cdot \sigma(2) \cdot \sigma(-2) = 0.18$
- $\frac{\partial s_2}{\partial b} = \sigma(b r_2) \cdot (1 - \sigma(b r_2)) \cdot r_2 = 0$

sidequest → Analytical derivative

$$\text{relu}(z) = \begin{cases} 0, & z \leq 0 \\ z, & z > 0 \end{cases}$$

$$\frac{\partial \text{relu}(z)}{\partial z} = \begin{cases} 0, & z \leq 0 \\ 1, & z > 0 \end{cases} \quad \text{Analytical derivative}$$



c) Back propagation

$$\frac{\partial y}{\partial e} = \frac{\partial y}{\partial s_1} \cdot \frac{\partial s_1}{\partial \lambda_1} \cdot \frac{\partial \lambda_1}{\partial e} = (1) \cdot (0.07) \cdot (-1) = -0.07$$

$$\frac{\partial y}{\partial c} = \frac{\partial y}{\partial s_1} \cdot \frac{\partial s_1}{\partial \lambda_1} \cdot \frac{\partial \lambda_1}{\partial c} = (1) \cdot (0.07) \cdot (1) = 0.07$$

$$\frac{\partial y}{\partial a} = \frac{\partial y}{\partial s_1} \cdot \frac{\partial s_1}{\partial a} = (1) \cdot 0.18 = \underline{\underline{0.18}}$$

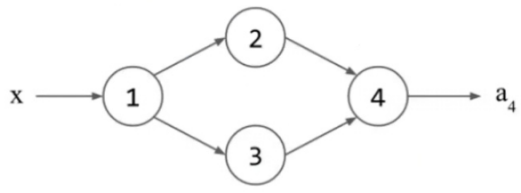
$$\frac{\partial y}{\partial b} = \frac{\partial y}{\partial s_a} \cdot \frac{\partial s_a}{\partial b} = 0$$

$$\frac{\partial y}{\partial f} = \frac{\partial y}{\partial s_a} \cdot \frac{\partial s_a}{\partial \lambda_a} \cdot \frac{\partial \lambda_a}{\partial f} = 0$$

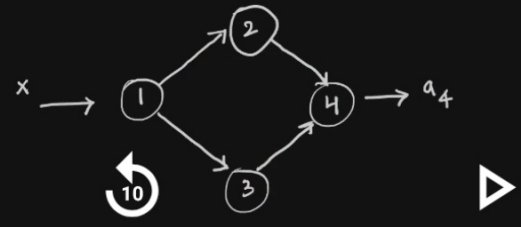
$$\frac{\partial y}{\partial d} = \frac{\partial y}{\partial s_a} \cdot \frac{\partial s_a}{\partial \lambda_a} \cdot \frac{\partial \lambda_a}{\partial d} = 0$$

Consider the neural network below. Note that:

- x , a scalar, is the input. a_4 , also a scalar, is the output.
- Each a_i value is the final output for neuron i in the network.
- Each z_i value is the pre-activation value for neuron i in the network (i.e. the dot product).
- w_1 is the weight used by neuron 1.
- Let w_{ij} be the weight from neuron i to neuron j .
- The activation function for each neuron is the function $g(x) = e^x$.
- $L(y, a)$ is the loss function. It is $(y - a)^2$, where y is the training label.



i. (2.5 pts) Use the Chain Rule and the equations given to calculate $\partial L / \partial z_2$. You may use x and y , along with all of the w , a , and z values. Note that the picture above only shows the neural network, i.e. does not include the loss function L .



ii. (1.5 pts) Use the Chain Rule and the equations given to calculate $\partial L / \partial z_3$. You may use x and y , along with all of the w , a , and z values. (Hint: If you're comfortable with the Chain Rule, you should be able to do this quickly by looking at your solution to Part A.)

A neural network has many layers, with the final two being X (with 4 nodes) and Y (with one node). Four weights $\{w_1 = 0.1, w_2 = -0.5, w_3 = 0.8, w_4 = 0.6\}$ connect x_1, x_2, x_3, x_4 to y_1 .

On the most recent forward pass through the network, the outputted values of X are:

$$x_1 = 0.3, \quad x_2 = -0.7, \quad x_3 = 0.5, \quad x_4 = 0.9.$$

The activation function for node y_1 is:

$$y_1 = \frac{1}{1 + e^{-s}}, \quad \text{where } s \text{ is the sum of weighted inputs into } y_1.$$

The error function is:

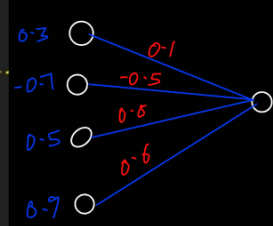
$$\frac{1}{2}(T - y_1)^2,$$

where T is the target value. In this case, $T = 0.8$.

The learning rate is 0.05, and the minibatch size is 1.

Tasks:

1. Compute the value of y_1 during the forward pass.
2. Compute the gradients of w_1, w_2, w_3, w_4 using backpropagation.
3. Find the new values for the four weights after the backward pass.



$$s = 0.03 + -0.35 + 0.4 + 0.54 = 0.62$$

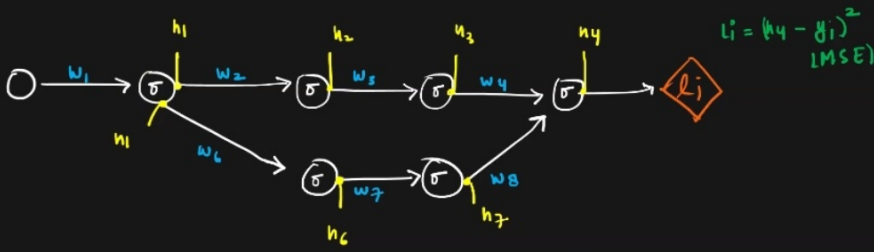
$$y = \sigma(s) = 0.65$$

$$\text{loss} = \frac{1}{2}(T - y_1)^2 = 0.01125$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial s} \cdot \frac{\partial s}{\partial w_1} = -0.034125 \times 0.3 = -0.0102$$

$$\begin{aligned} \frac{\partial L}{\partial y} &= (y - T) \\ \frac{\partial L}{\partial y} &= 0.65 - 0.8 = -0.15 \\ \frac{\partial y}{\partial s} &= \sigma(s) (1 - \sigma(s)) \\ &= 0.65 (1 - 0.65) \\ &= 0.2275 \end{aligned}$$

Vanishing Gradient & Exploding gradient



$$\frac{\partial L_i}{\partial w_1} = \frac{\partial L_i}{\partial h_4} \left[\frac{\partial h_4}{\partial h_3} \cdot \frac{\partial h_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial w_1} + \frac{\partial h_4}{\partial h_7} + \frac{\partial h_7}{\partial h_6} \cdot \frac{\partial h_6}{\partial h_1} \right] \cdot \frac{\partial h_1}{\partial w_1}$$

→ path 1 from \$h_1\$ to \$h_4\$

$$\frac{\partial h_4}{\partial h_1} = \frac{\partial h_4}{\partial h_3} \Big|_{\text{path 1}} + \frac{\partial h_4}{\partial h_7} \Big|_{\text{path 2}} + \frac{\partial h_7}{\partial h_6} \cdot \frac{\partial h_6}{\partial h_1} \Big|_{\text{path 2}}$$

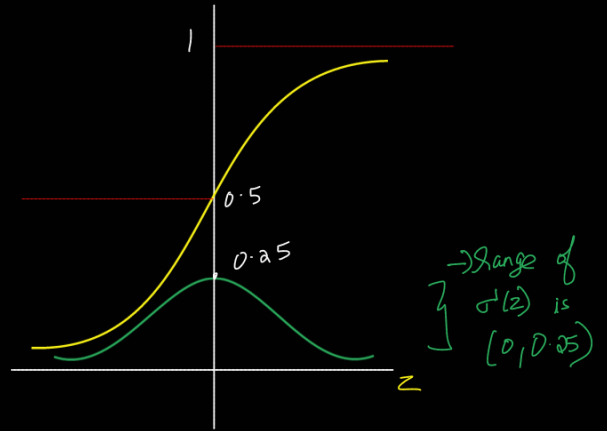
$$\frac{\partial h_4}{\partial h_3} = \frac{\partial \sigma(w_4 h_3)}{\partial h_3} = w_4 \cdot \sigma(w_4 h_3) \cdot (1 - \sigma(w_4 h_3)) = w_4 \cdot \sigma'(w_4 h_3)$$

$$\frac{\partial h_3}{\partial h_2} = w_3 \sigma'(w_3 h_2)$$

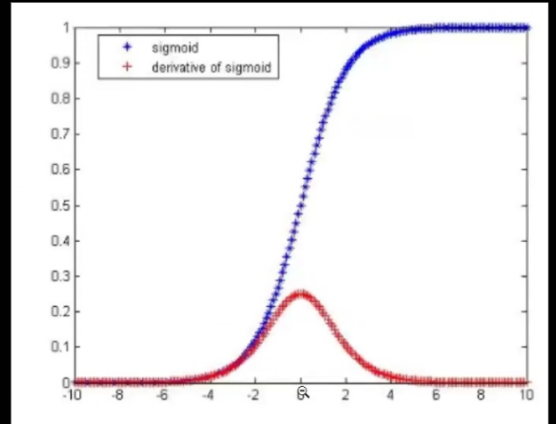
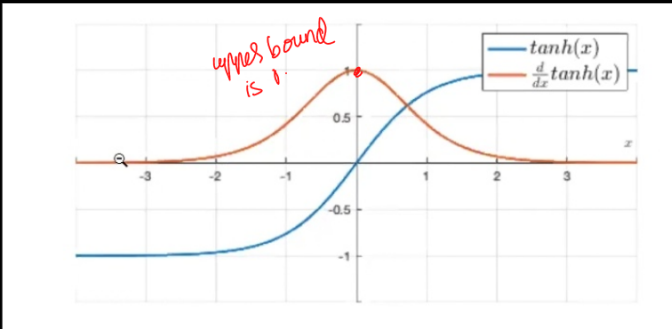
$$\frac{\partial h_2}{\partial h_1} = w_2 \sigma'(w_2 h_1)$$

$$\rightarrow \frac{\partial h_4}{\partial h_1} \Big|_{\text{path 1}} = w_4 \cdot w_3 \cdot w_2 \cdot \sigma'(w_4 h_3) \cdot \sigma'(w_3 h_2) \cdot \sigma'(w_2 h_1) \leq (0.25)(0.25)(0.25)$$

$\sigma(z), \sigma'(z)$



- This is for 3 layers
- Imagine the same form of say 10, 20 layers
- The gradient becomes very small
- Learning becomes very slow
- Same is true for tanh activation function



→ This issue arises because of the way these functions behave saturated for both high values of \$z\$ & extremely negative values of \$z\$ in \$\tanh(z)\$ & \$\sigma(z)\$.

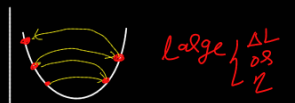
$$\rightarrow \frac{\partial h_4}{\partial h_1} \Big|_{\text{path 1}} = w_4 \cdot w_3 \cdot w_2 \cdot \sigma'(w_4 h_3) \cdot \sigma'(w_3 h_2) \cdot \sigma'(w_2 h_1)$$

↓ → since multiplying weight may become very very large for more layers

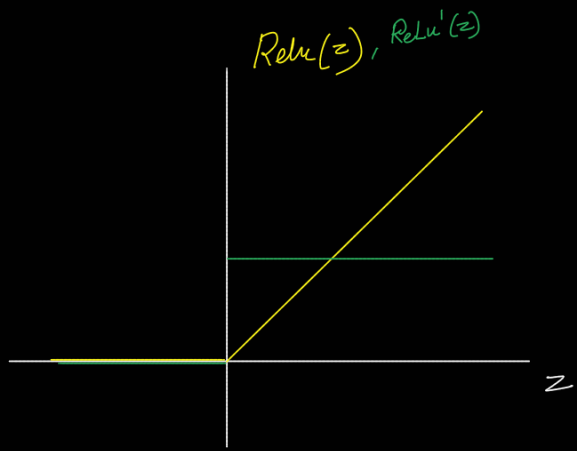
→ This issue is called exploding gradient

$$w_{i, \text{new}} = w_{i, \text{old}} - \eta \nabla L \rightarrow \text{Large gradient} \rightarrow \text{more chance of GD divergence}$$

★ One of the solution is to clip the gradient @ some upper limit



ReLU & Dead Neuron

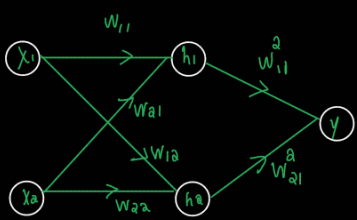


- ReLU helps in solving vanishing gradient
- But what if any of the sub gradient in a path is zero \Rightarrow Entire gradient becomes zero
- **Dead neuron** \rightarrow does not pass forward information in forward path
- Leads to zero gradient in back propagation
- Makes an entire path dead just because one neuron in the path is dead.

Initialisation of Weight

$$\rightarrow \frac{\partial h_1}{\partial w_{11}} = W_4 \cdot W_3 \cdot W_2 \cdot \sigma'(w_4 h_3) \cdot \sigma'(w_3 h_2) \cdot \sigma'(w_2 h_1)$$

$\ll (0.25) (0.25) (0.25)$



all equal wts

$$\rightarrow h_1 = kx_1 + kx_2$$

$$h_2 = kx_1 + kx_2$$

$$\frac{\partial y}{\partial w_{11}} = \frac{\partial y}{\partial h_1} \cdot \frac{\partial h_1}{\partial w_{11}} = kx_1$$

$$\frac{\partial y}{\partial w_{12}} = \frac{\partial y}{\partial h_1} \cdot \frac{\partial h_1}{\partial w_{12}} = kx_1$$

① Not possible to initialise weights all to zero

- \rightarrow All gradients in all paths become zero
- \rightarrow No learning

② Initialise all weights to some constant K.

- \rightarrow Certain combination of weights got updated concurrently
- \rightarrow Take many iterations to get out of this effect and properly learn useful weights
- \rightarrow more aggressive issue in larger N/w with many paths
- \rightarrow Too much symmetry in N/w \neq costs more iterations to break from symmetry

③ Initialise randomly to large values

- \rightarrow leads to exploding gradient

④ Initialise randomly close to zero

- \rightarrow Leads to vanishing gradient
- \rightarrow Too many wasted iterations to get out of small weights

Solution:- ① Xavier Initialisation, ② He initialisation

- \rightarrow Initially randomly
- \rightarrow Activations / are centered 0; (mean)
- \rightarrow $\left\{ \begin{array}{l} \text{variance}(\text{layer } i) = \text{variance}(\text{layer } i-1) \end{array} \right.$

Summary

Vanishing gradient due to -

- choice of activation
- Initialisation of wts close to zero
- Depth of N/w

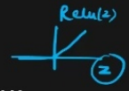
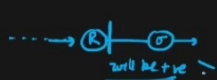
Exploding gradient due to:-

→ Initialisation of weights

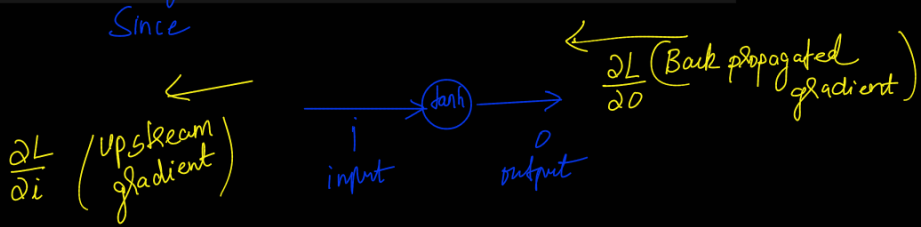
n Epoch of iteration is said to be completed when the learning algorithm of model has seen the entire dataset n times.

Q. Which of the options are correct [MSQ] ?

- a) You are solving a binary classification task, and the final two layers in your network are a ReLU activation followed by a sigmoid activation. This leads to all predictions to be positive.
- b) A 2-layer fully connected neural network with 5 neurons in each layer has a total of 60 parameters (i.e. weights and biases) → Assuming ip layer = 5 dim
- c) Vanishing gradient causes deeper layers to learn more slowly than earlier layers
- d) The backpropagated gradient through a tanh non-linearity is always greater than in magnitude than the upstream gradient.



Affect of weights that connect btw the last two layers ignored



$$\frac{\partial L}{\partial i} = \frac{\partial L}{\partial o} \cdot \frac{\partial o}{\partial i} = \frac{\partial L}{\partial o} \cdot \underbrace{\tanh'(i)}_{\text{Always } \leq 1}$$

Q. Which of the options are correct [MSQ] ?

- a) Maximum value of local gradient when sigmoid is used as an activation function is 0.5
- b) Minimum value of local gradient when sigmoid is used as an activation function is 0.25
- c) Maximum value of local gradient when tanh is used as an activation function is 1
- d) Minimum value of local gradient when Relu is used as an activation function is 0

Which of the following properties, if present in an activation function CANNOT be used in a neural network?

- a. The function is periodic ✓
- b. The function is monotonic
- c. The function is unbounded
- d. Both a and b

Q. Which one of the following statements on initialization is false?

- a) The variances of layer outputs remain unchanged during training when Xavier initialization is used. **T**
- b) Initializing all weights to a positive constant value isn't sufficient to break learning symmetry during training. **F**
- c) Different activation functions may benefit from different types of initializations. **T**
- d) It's possible to break symmetry by initializing weights to be sampled uniformly from the set $\{-1, 1\}$. **T**

☆ Periodic function Not usable in neuron activation.

Q. Which of the following activation functions can lead to vanishing gradients?

- a) ReLU
- b) Tanh
- c) Sigmoid
- d) All of the above

(e) (2 points) Which of the following is true about the vanishing gradient problem? (Circle all that apply)

- (i) Tanh is usually preferred over sigmoid because it doesn't suffer from vanishing gradients. **False**
- (ii) Vanishing gradient causes deeper layers to learn more slowly than earlier layers. **False**
- (iii) Leaky ReLU is less likely to suffer from vanishing gradients than sigmoid. **True**
- (iv) Xavier initialization can help prevent the vanishing gradient problem. **False**
- (v) None of the above

Linear discriminant Analysis

→ Practically not used now
 → Absolute & better methods are available

→ LDA as a classifier
 → dimensionality reduction technique

Baye's optimal classifier

$$P(Y=k|X) = \frac{\overset{\text{prior}}{P(Y=k)} \cdot \overset{\text{likelihood}}{P(X|Y=k)}}{\underset{\text{normalises}}{P(X)}}$$

X → multidimensional i/p
 Y → categorical o/p

$$P(X) = \sum_i^{\text{all classes}} P(\text{class}=i) \cdot P(X|\text{class}=i)$$

→ Baye's optimal classifier is the best that any classifier can perform
 → Baye's optimal error is the minimum error that any generalising model can achieve without overfitting.
 → Logistic regression estimates posterior directly by learning the hyperplane or decision boundary rather than the baye's parameters

→ Naive Baye's assumed conditional independence of attribute to learn a less expensive baye's parameters.

→ Let likelihood of class k - ie $P(X|Y=k) \rightarrow f_k(x)$ a.k.a class conditional density.

→ Let prior of class k - ie $P(Y=k) \rightarrow \pi_k$

$$\Rightarrow P(Y=k|X) = \frac{\pi_k \cdot f_k(x)}{\sum_i \pi_i \cdot f_i(x)} \quad \left. \vphantom{\frac{\pi_k \cdot f_k(x)}{\sum_i \pi_i \cdot f_i(x)}} \right\} \rightarrow \text{change just notation}$$

LDA Assumption aka GDA (Gaussian discriminant analysis)

① Likelihood / class conditional density - $P(X|Y=k) = f_k(x) \sim N(\mu_k, \sigma_k)$, Assume $x \in \mathbb{R}^1$

$$\rightarrow f_k(x) = \frac{1}{\sigma_k \sqrt{2\pi}} \exp\left[-\frac{1}{2} \left(\frac{x-\mu_k}{\sigma_k}\right)^2\right]$$

$$\rightarrow P(Y=k|X) = \frac{\pi_k \cdot \frac{1}{\sigma_k \sqrt{2\pi}} \exp\left[-\frac{1}{2} \left(\frac{x-\mu_k}{\sigma_k}\right)^2\right]}{\sum_i \pi_i \cdot \frac{1}{\sigma_i \sqrt{2\pi}} \exp\left[-\frac{1}{2} \left(\frac{x-\mu_i}{\sigma_i}\right)^2\right]}$$

② All k classes have same variance

$$\rightarrow \sigma_1^2 = \sigma_2^2 = \sigma_3^2 = \dots = \sigma_k^2$$

example - Dataset with 3 classes

- Dog
- Cat
- Neither

→ So assumption says such classes do not exist in the dataset

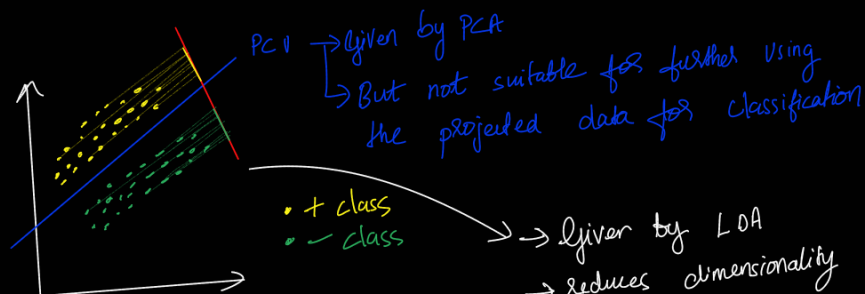
Variance of likelihood similar

can be horse, lion, mouse, cobra

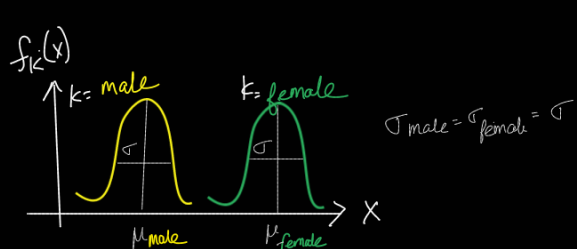
→ very higher variance in class

$$P(Y=k|X) = \frac{\pi_k \cdot \frac{1}{\sigma\sqrt{\pi}} \exp\left(-\frac{1}{2} \cdot \left(\frac{x-\mu_k}{\sigma}\right)^2\right)}{\sum_i \pi_i \cdot \frac{1}{\sigma\sqrt{\pi}} \exp\left(-\frac{1}{2} \left(\frac{x-\mu_i}{\sigma}\right)^2\right)}$$

Intuition



→ given by LOA
 → reduces dimensionality such that the points can be easily classified in the lower dimension



Prediction k*

$$k^* = \operatorname{argmax}_k P(Y=k|X) = \operatorname{argmax}_k \log P(Y=k|X) = \operatorname{argmax}_k \log \left\{ \frac{\pi_k \cdot f_k(x)}{\sum_i \pi_i f_i(x)} \right\}$$

Make maths easier
 Avoids underflow of values

→ denominator doesn't depend on k.

$$k^* = \operatorname{argmax}_k \log(\pi_k \cdot f_k(x))$$

$$k^* = \operatorname{argmax}_k \left\{ \log \pi_k + \log f_k(x) \right\} \rightarrow$$

- Applicable for all classification algorithms in general
- No assumption incorporated right now

→ For any class k the denominator is same

→ So can safely ignore inside the argmax_k

LDA Assumption

$$k^* = \operatorname{argmax}_k \left\{ \log(\pi_k) + \log\left(\frac{1}{\sigma\sqrt{\pi}} \exp\left(-\frac{1}{2} \left(\frac{x-\mu_k}{\sigma}\right)^2\right)\right) \right\}$$

$$k^* = \operatorname{argmax}_k \left\{ \log(\pi_k) - \log(\sigma\sqrt{\pi}) - \frac{1}{2} \left(\frac{x-\mu_k}{\sigma}\right)^2 \right\}$$

→ Constant
 → independent of class

$$k^* = \text{argmax}_k \left\{ \log(\pi_k) - \frac{1}{2} \left(\frac{x^2 + \mu_k^2 - 2x\mu_k}{\sigma^2} \right) \right\}$$

constant give datapoint x
 \rightarrow does not depend of k

$$k^* = \text{argmax}_k \left\{ \log(\pi_k) - \frac{\mu_k^2}{2\sigma^2} + \frac{x\mu_k}{\sigma^2} \right\} \quad k^* = \text{argmax}_k (\delta_k(x))$$

$\delta_k(x) \rightarrow$ called discriminant
 $\rightarrow = \frac{\log \pi_k}{m} - \frac{\mu_k^2}{2n} + \frac{x\mu_k}{w}$

$2k+1$ parameters $\left\{ \begin{array}{l} k \text{ parameters} \leftarrow \rightarrow \text{given entire dataset } \pi_k \text{ can be calculated for each class} \\ k \text{ parameters} \leftarrow \rightarrow \mu_k \text{ can be calculated for each class} \\ 1 \text{ parameter} \leftarrow \rightarrow \sigma \text{ can be calculated} \end{array} \right.$

\Rightarrow given entire dataset & a new unknown point x
 $\delta_k(x) = m-n + xw \quad \left. \begin{array}{l} \rightarrow \text{equation of a line} \\ \rightarrow \text{Linear in } x \end{array} \right\}$

Hence called linear discriminant analysis.

with optimisation for $P(y)$
 $2k$ parameters

Decision Boundary

\rightarrow @ DB value of discriminant is same for any two classes
 $\rightarrow \delta_{k_1}(x) = \delta_{k_2}(x)$ for any two class $k_1 \neq k_2$
 \rightarrow Linear decision boundary \rightarrow No feature transformation assumed

Binary classification & DB

$$\Rightarrow \log \pi_1 - \frac{\mu_1^2}{2\sigma^2} + \frac{x\mu_1}{\sigma^2} = \log \pi_2 - \frac{\mu_2^2}{2\sigma^2} + \frac{x\mu_2}{\sigma^2}$$

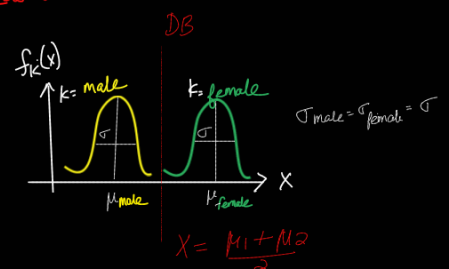
$$\Rightarrow \frac{x(\mu_1 - \mu_2)}{\sigma^2} = \frac{\mu_1^2 - \mu_2^2}{2\sigma^2} + \log \left(\frac{\pi_2}{\pi_1} \right)$$

$$\Rightarrow \frac{x(\mu_1 - \mu_2)}{\sigma^2} = \frac{(\mu_1 - \mu_2)(\mu_1 + \mu_2)}{2\sigma^2} + \log \left(\frac{\pi_2}{\pi_1} \right)$$

Assume $\pi_1 = \pi_2$ $\left\{ \begin{array}{l} \rightarrow \text{unconditionally without giving } x, P(y=1) = P(y=2) \\ \rightarrow \text{dataset has equal examples of } y=1 \text{ \& } y=2 \end{array} \right\} \Rightarrow \log \left(\frac{\pi_2}{\pi_1} \right) = 0$

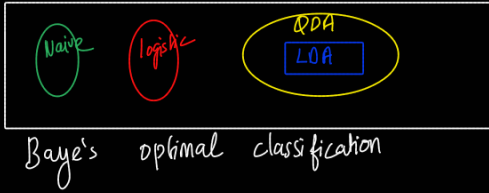
$$\Rightarrow x = \frac{\mu_1 + \mu_2}{2}$$

- \rightarrow Binary classification
- \rightarrow One dimensional feature
- \rightarrow prior is equal
- \rightarrow LDA assumptions
 - \rightarrow shared variances
 - \rightarrow Normal likelihood



→ if the priors are equal & variances are shared then LDA DB turns out to be perpendicular bisector of $\mu_1 + \mu_2$

→ if all classes does not share some variance

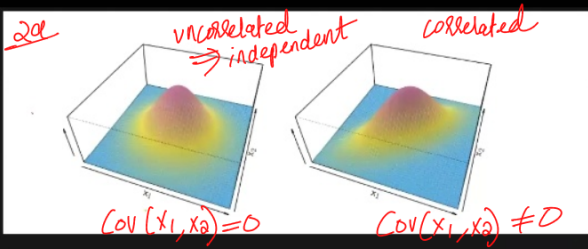


} in terms of assumptions

→ QDA
 → Quadratic discriminant analysis
 → discriminant is quadratic
 $\delta_k(x) \rightarrow Ax^2 + Bx + C$

Multivariate inputs & Multivariate Gaussian

→ for multivariate normal uncorrelated \Rightarrow independent
 → each component feature follows gaussian
 → No linear combination of features gives a non gaussian



$\mu_k \rightarrow$ mean column vector $d \times 1$ | $|\Sigma_k| \Rightarrow$ determinant
 $\Sigma_k \rightarrow$ covariance matrix $d \times d$ | $\Sigma_k \rightarrow$ symmetric matrix

$f_k(x) = p(x|y=k) = N(x|\mu_k, \Sigma_k)$
 $= \frac{1}{(2\pi)^{d/2}} |\Sigma|^{-1/2} \exp\left(-\frac{1}{2}(x-\mu_k)^T \Sigma^{-1}(x-\mu_k)\right)$
 ↓
 $x \in \mathbb{R}^d$
 Column Vector $d \times 1$

$f_k(x) = \frac{1}{(2\pi)^{d/2}} |\Sigma_k|^{-1/2} \exp\left(-\frac{1}{2}(x-\mu_k)^T \Sigma_k^{-1}(x-\mu_k)\right)$

$\Sigma = \begin{bmatrix} x_1 & x_2 & \dots & x_n \\ \text{Var}(x_1) & \text{Cov}(x_1, x_2) & & \\ \text{Cov}(x_1, x_2) & \text{Var}(x_2) & & \\ \vdots & \vdots & \ddots & \vdots \\ \text{Var}(x_n) \end{bmatrix}$

→ diagonal values are variance
 $a_{ii} = \text{Var}(x_i)$
 $a_{ij} = a_{ji} = \text{Cov}(x_i, x_j)$

① Isotropic covariance

→ diagonal entries are equal $\rightarrow \text{Variance}(x_1) = \text{Var}(x_2) = \dots = \text{Var}(x_n)$
 → offdiagonal entries are zero $\rightarrow \text{Cov}(x_i, x_j) = 0, i \neq j$

Spherical contours for $f_k(x) = c$

$\Sigma = \sigma^2 I = \begin{bmatrix} \sigma^2 & & & 0 \\ & \sigma^2 & & \\ & & \ddots & \\ 0 & & & \sigma^2 \end{bmatrix}$

→ Not same as LDA assumption that $P(x|y=k)$ has shared variance among different classes.

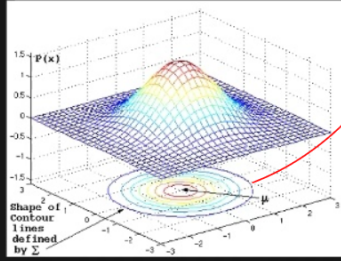
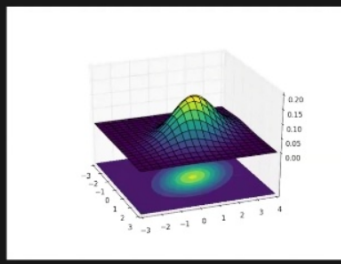
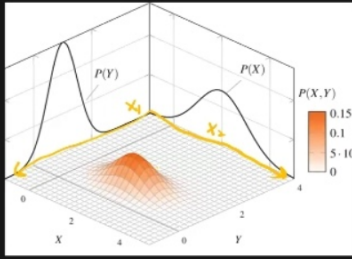
→ Here same variance among different features

② Anisotropic covariance

→ diagonal entries are non zero but not equal

ellipsoidal contours for $f_k(x) = c$

→ off diagonal entries are zero



→ spherical contours → $\sigma_1^2 = \sigma_2^2 = \dots = \sigma_n^2 = \sigma^2$

→ ellipsoidal contours → $\sigma_1^2 \neq \sigma_2^2 \neq \dots \neq \sigma_n^2$

→ Axis aligned ⇒ $Cov(X_i, X_j) = 0, i \neq j$
 → rotated ⇒ $Cov(X_i, X_j) \neq 0$ for some $i \neq j$

For a multivariate Gaussian distribution, the contours of f of x equal to some constant c will be a spherical or circle-shaped if it is isotropic covariance and it will be an ellipsoid if it will be an anisotropic covariance. Is this statement true?

✓ Yes, your statement is true – with a small clarification.

Let's restate it precisely:

For a multivariate Gaussian distribution

$$f(\mathbf{x}) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu})\right)$$

the contours of constant density (i.e., where $f(\mathbf{x}) = c$) satisfy:

$$(\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}) = \text{constant}$$

This defines a quadratic form, whose shape depends on the covariance matrix Σ :



◆ Case 1: Isotropic covariance

If

$$\Sigma = \sigma^2 I$$

then

$$(\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}) = \frac{1}{\sigma^2} \|\mathbf{x} - \boldsymbol{\mu}\|^2 = \text{constant}$$

→ The contours are spheres in higher dimensions (or circles in 2D).

◆ Case 2: Anisotropic covariance

If

$$\Sigma$$

is **not proportional to the identity matrix**, then different directions have different variances (and possibly correlated components).

$$(\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}) = \text{constant}$$

now describes an ellipsoid in general (ellipse in 2D).

✓ Summary:

Covariance type	Contour shape	Description
Isotropic ($\Sigma = \sigma^2 I$)	Sphere / Circle	Equal variance, uncorrelated directions
Anisotropic ($\Sigma \neq \sigma^2 I$)	Ellipsoid	Unequal variance and/or correlated directions

So your statement is **entirely correct** – just note that "spherical" corresponds to higher-dimensional circles (spheres) and "ellipsoid" is the general case.

Further what difference can be observed if the matrix has unequal variance and there is no correlation. means the covariance matrix is entirely diagonal. but now compare this with the unequal variance and correlated components in matrix. so what differences in contour can be observed

Excellent question – this goes one step deeper into understanding the geometry of Gaussian contours

Let's consider the two cases you mentioned carefully.

◆ Case 1: Unequal variances, no correlation

Covariance matrix:

$$\Sigma = \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}$$

Then

$$(\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}) = \frac{(x - \mu_x)^2}{\sigma_x^2} + \frac{(y - \mu_y)^2}{\sigma_y^2} = \text{constant}$$

◆ Contour shape:

- It's an axis-aligned ellipse (ellipse whose axes are parallel to the coordinate axes).
- The major and minor axes correspond to the variances σ_x^2 and σ_y^2 .
- No rotation occurs because there are no cross-correlation terms.

◆ Case 2: Unequal variances, with correlation

Covariance matrix:

$$\Sigma = \begin{bmatrix} \sigma_x^2 & \rho \sigma_x \sigma_y \\ \rho \sigma_x \sigma_y & \sigma_y^2 \end{bmatrix}$$

with $|\rho| > 0$.

Now the term expands to:

$$(\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}) = A(x - \mu_x)^2 + B(y - \mu_y)^2 + 2C(x - \mu_x)(y - \mu_y)$$

◆ Contour shape:

- Still an ellipse, but rotated relative to the coordinate axes.
- The rotation angle depends on the correlation coefficient ρ .
- When $\rho > 0$: the ellipse is tilted along the line $y = x$.
- When $\rho < 0$: it's tilted along $y = -x$.



◆ Summary Table

Covariance Matrix Type	Variances	Correlation	Contour Shape	Orientation
Isotropic ($\sigma_x = \sigma_y$)	Equal	None	Circle	Not applicable
Diagonal, unequal variances	Unequal	None	Ellipse	Aligned with axes
Full (non-diagonal)	Unequal	Present	Ellipse	Rotated toward correlated direction

◆ Intuitive takeaway:

- Unequal variances → Stretching (ellipse instead of circle).
- Non-zero correlation → Rotation (ellipse tilted). ↓
- So correlation affects orientation, variances affect shape/scale.

★ $f_k(x) = N(\mu_k, \Sigma_k) \rightarrow$ multivariate

$\Sigma_k \rightarrow$ covariance given class k

if $\Sigma_k \rightarrow$ all non diagonal entries zero for all class k

} ⇒ features are independent given class
 } ⇒ equivalent to Naive Bayes' assumption

$$P(Y=k|X) = \frac{\pi_k \cdot f_k(x)}{\sum_i \pi_i f_i(x)} = \frac{\pi_k \cdot \frac{1}{(2\pi)^{d/2}} |\Sigma_k|^{-1/2} \cdot \exp\left[-\frac{1}{2} (x-\mu_k)^T \Sigma_k^{-1} (x-\mu_k)\right]}{\sum_i \pi_i \frac{1}{(2\pi)^{d/2}} |\Sigma_i|^{-1/2} \cdot \exp\left[-\frac{1}{2} (x-\mu_i)^T \Sigma_i^{-1} (x-\mu_i)\right]}$$

} QDA

$$= \frac{\pi_k \cdot \frac{1}{(2\pi)^{d/2}} |\Sigma|^{-1/2} \cdot \exp\left[-\frac{1}{2} (x-\mu_k)^T \Sigma^{-1} (x-\mu_k)\right]}{\sum_i \pi_i \frac{1}{(2\pi)^{d/2}} |\Sigma|^{-1/2} \cdot \exp\left[-\frac{1}{2} (x-\mu_i)^T \Sigma^{-1} (x-\mu_i)\right]}$$

} LDA

$$k^* = \operatorname{argmax}_k \left\{ \log(\pi_k) - \frac{1}{2} (x-\mu_k)^T \Sigma^{-1} (x-\mu_k) \right\}$$

→ denominators not dependent on k
 → $\log\left(\frac{1}{(2\pi)^{d/2}} |\Sigma|^{-1/2}\right)$ → doesn't depend on k

$$\rightarrow -\frac{1}{2} (x^T - \mu_k^T) \Sigma^{-1} (x - \mu_k)$$

Both of these terms are equal since Σ^{-1} is symmetric

$$-\frac{1}{2} \left(x^T \Sigma^{-1} x - x^T \Sigma^{-1} \mu_k - \mu_k^T \Sigma^{-1} x + \mu_k^T \Sigma^{-1} \mu_k \right)$$

→ Does not depend on class k.
 → Can be eliminated from argmax_k

$$\Rightarrow k^* = \operatorname{argmax}_k \left\{ \log \pi_k + x^T \Sigma^{-1} \mu_k - \frac{\mu_k^T \Sigma^{-1} \mu_k}{2} \right\}$$

DISCRIMINANT

→ some constant scalars β
 → some constant vectors w } given dataset

$$k^* = \operatorname{argmax}_k \left\{ x^T w + \beta \right\}$$

Decision boundary (Binary)

$$\log \pi_1 + x^T \Sigma^{-1} \mu_1 - \frac{\mu_1^T \Sigma^{-1} \mu_1}{2} = \log \pi_2 + x^T \Sigma^{-1} \mu_2 - \frac{\mu_2^T \Sigma^{-1} \mu_2}{2}$$

$$\Rightarrow x^T \Sigma^{-1} (\mu_1 - \mu_2) = \log \frac{\pi_2}{\pi_1} + \frac{\mu_1^T \Sigma^{-1} \mu_1 - \mu_2^T \Sigma^{-1} \mu_2}{2}$$

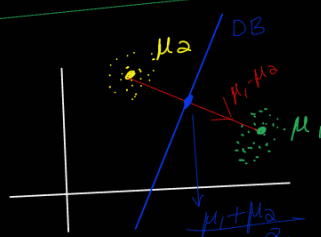
$$x^T \Sigma^{-1} (\mu_1 - \mu_2) = \frac{1}{2} (\mu_1^T \Sigma^{-1} \mu_1 - \mu_2^T \Sigma^{-1} \mu_2)$$

constant vectors constant scalars

$$x^T w = -\beta$$

$$w^T x + \beta = 0$$

⇒ DB is linear in x



sidequest → centroid method

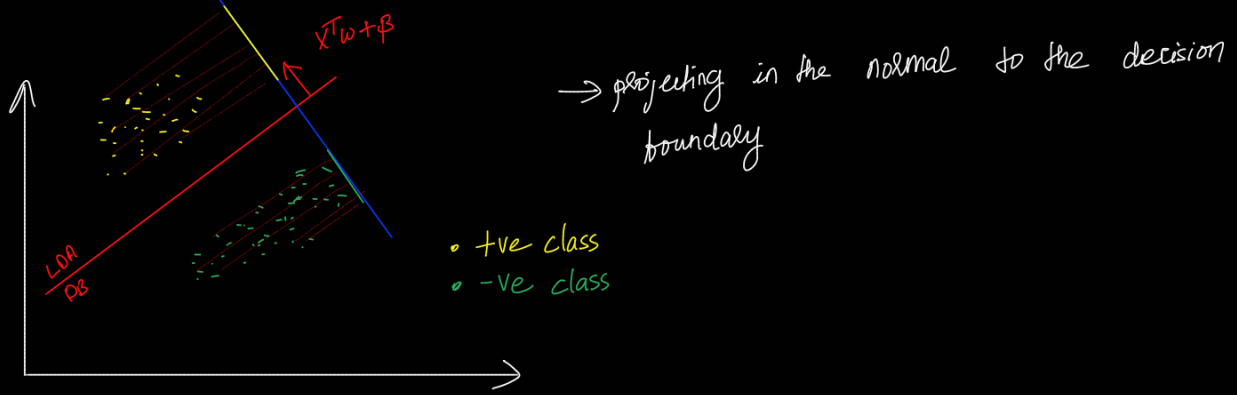
$$I \Sigma^{-1} = \sigma^{-2} I \quad (\text{isotropic}) \quad \Sigma^{-1} = \begin{bmatrix} \sigma^{-2} & 0 & 0 \\ 0 & \sigma^{-2} & 0 \\ 0 & 0 & \sigma^{-2} \end{bmatrix}$$

$$\text{then, } \Sigma^{-1} = \frac{I}{\sigma^2}$$

$$x^T (\mu_1 - \mu_2) = \frac{\mu_1^T \mu_1 - \mu_2^T \mu_2}{2\sigma^2} = \frac{(\mu_1 + \mu_2)^T (\mu_1 - \mu_2)}{2\sigma^2}$$

$$\Rightarrow x^T = \frac{(\mu_1 + \mu_2)^T}{2} + \text{anything } \perp \text{ to } (\mu_1 - \mu_2)$$

☆ We cannot cross $(\mu_1 - \mu_2)$ on both sides since not invertible



Aspect	LDA (Linear Discriminant Analysis)	QDA (Quadratic Discriminant Analysis)
Decision Boundary	Linear: The decision boundary is a hyperplane.	Quadratic: The decision boundary is a quadratic surface.
Covariance Assumption	Assumes all classes share the same covariance matrix (Σ).	Assumes each class has its own covariance matrix (Σ_k).
Overfitting Risk	Less prone to overfitting due to fewer parameters.	More prone to overfitting, especially with limited data, as more parameters are estimated.

parameters = $d \times k + d \times d$

classes = k

input dim = d

$\{ \mu_1, \mu_2, \dots, \mu_k \}$

$\{ \Sigma_1, \Sigma_2, \dots, \Sigma_k \}$

Other choices are possible by constraining Σ_k in various ways:

Covariance matrices	Separate	Shared
Full	QDA	LDA
Diagonal	Naive Bayes	Diagonal LDA
Spherical	'Spherical QDA'	Nearest centroid

$\Sigma = \begin{bmatrix} \sigma_1^2 & & 0 \\ & \ddots & \\ 0 & & \sigma_d^2 \end{bmatrix}$

$\Sigma_k = \begin{bmatrix} \sigma_{k1}^2 & & 0 \\ & \sigma_{k2}^2 & \\ 0 & & \sigma_{kd}^2 \end{bmatrix}$

$\Sigma_1 = \Sigma_2 = \dots = \Sigma_k = \Sigma = \begin{bmatrix} \sigma_1^2 & & \\ & \ddots & \\ & & \sigma_d^2 \end{bmatrix}$

→ if Σ_k is always diagonal $\forall k \Rightarrow$ given class, all features are independent

→ Naive Bayes

→ QDA Naive Bayes or spherical QDA

→ further if $\Sigma_1 = \Sigma_2 = \dots = \Sigma_k = \Sigma \Rightarrow$ diagonal LDA, nearest centroid

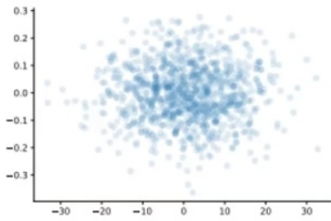
→ further if diagonal entries same \Rightarrow Full diagonal spherical

You want to train a dog identifier with Gaussian discriminant analysis. Your classifier takes an image vector as its input and outputs 1 if it thinks it is a dog, and 0 otherwise. You use a certain dataset, modified such that all the classes that are not "dog" have the label 0. Your training set has 5,000 dog images and 45,000 non-dog ("other") images. Which of the following statements seems likely to be correct?

- A. LDA has an advantage over QDA because the two classes have different numbers of training examples.
- B. QDA has an advantage over LDA because the two classes have different numbers of training examples.
- C. LDA has an advantage over QDA because the two classes are expected to have very different covariance matrices.
- D. QDA has an advantage over LDA because the two classes are expected to have very different covariance matrices.

Both LDA & QDA has no assumption about prior (π_1, π_0)
 LDA assumes same covariance given classes \rightarrow

[4 pts] Below are 1,000 sample points drawn from a two-dimensional multivariate normal distribution. Which of the following matrices could (without extreme improbability) be the covariance matrix of the distribution? (Pay attention to the numbers on the axes!)



A: $\Sigma = \begin{bmatrix} 100 & 0 \\ 0 & 0.01 \end{bmatrix}$

B: $\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \rightarrow$ Variance not same

C: $\Sigma = \begin{bmatrix} 100 & 0 \\ 0 & 0.1 \end{bmatrix}$

D: $\Sigma = \begin{bmatrix} -10 & 0 \\ 0 & -0.1 \end{bmatrix} \rightarrow$ Variance cannot be negative

10 is too low
 diagonal = σ^2

[3 pts] Which of the following are true about two-class Gaussian discriminant analysis? Assume you have estimated the parameters $\hat{\mu}_C, \hat{\Sigma}_C, \hat{\pi}_C$ for class C and $\hat{\mu}_D, \hat{\Sigma}_D, \hat{\pi}_D$ for class D.

If $\hat{\mu}_C = \hat{\mu}_D$ and $\hat{\pi}_C = \hat{\pi}_D$, then the LDA and QDA classifiers are identical

If $\hat{\Sigma}_C = \hat{\Sigma}_D, \hat{\pi}_C = 1/6$, and $\hat{\pi}_D = 5/6$, then the LDA and QDA classifiers are identical

If $\hat{\Sigma}_C = I$ (the identity matrix) and $\hat{\Sigma}_D = 5I$, then the LDA and QDA classifiers are identical

If the LDA and QDA classifiers are identical, then the posterior probability $P(Y = C|X = x)$ is linear in x

discriminant / DB \rightarrow linear \rightarrow LDA
 posterior is not linear \rightarrow quadratic \rightarrow QDA

(d) [4 pts] Which statements are true of Gaussian discriminant analysis for two-class classification, specifically quadratic discriminant analysis (QDA) and linear discriminant analysis (LDA)? (Assume that there are no added features.)

A: QDA for isotropic Gaussians (i.e., with the same variance in all directions) becomes the centroid method when the prior probabilities of the two classes are equal.

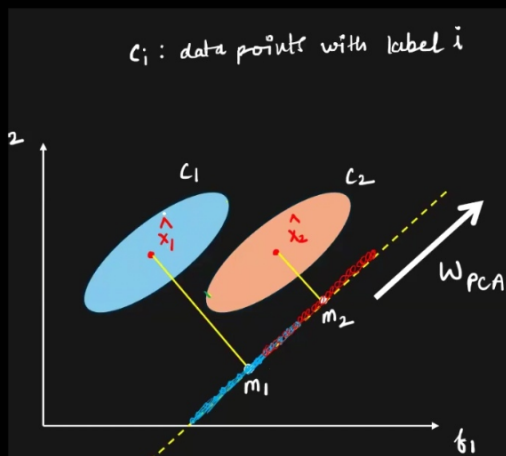
C: LDA for isotropic Gaussians (i.e., with the same variance in all directions) becomes the centroid method when the prior probabilities of the two classes are equal.

B: QDA is more likely to overfit than LDA when the number of training points is small.

D: LDA for anisotropic Gaussians can produce nonlinear decision boundaries.

LDA always Linear DB

LDA as dimensionality reduction tool



C_i : data points with label i

Σ_1 : covariance matrix of C_1

Σ_2 : " of C_2

$\Sigma_1 = \Sigma_2$ assumption

- ① PCA focuses on information retention \rightarrow maximise variance
- ② does not care about class information

observation:

- Losing class information because of overlap \rightarrow for new unseen data in the overlapping region, L cannot predict class
- Direction of projection \neq direction of decision boundary is perpendicular.

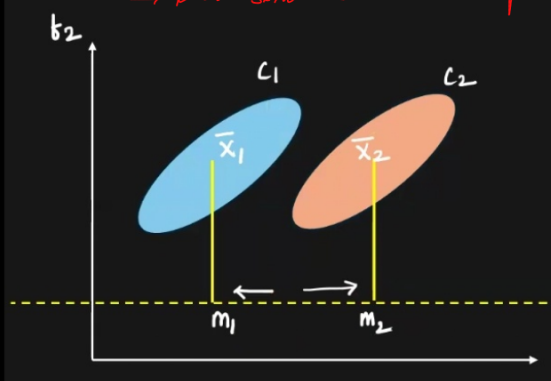
b_1 : i^{th} features

\hat{x}_i = means of points of class $i = \frac{\sum x}{n_i}$ - #pts in class i

m_i = mean vectors in projected space -

\rightarrow To try to reduce overlap, we can try to project such that the projected mean of the classes are as far as possible

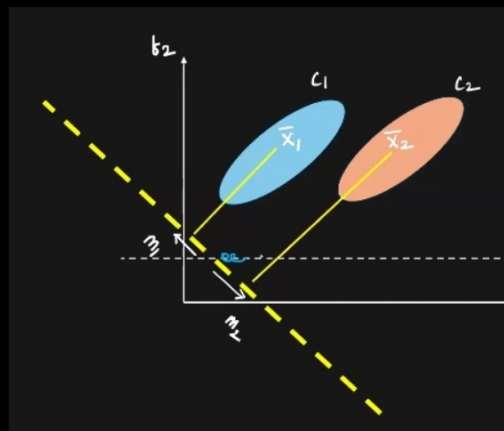
\rightarrow But still there is overlap



\rightarrow For binary classification, this direction is the $(m_2 - m_1)$ direction

optimum direction of projection $(W^*) = \text{argmax}_W (m_1 - m_2)^2$

\rightarrow Another approach is to reduce the scatter/variance of different classes in the projected space so as to try to avoid overlap



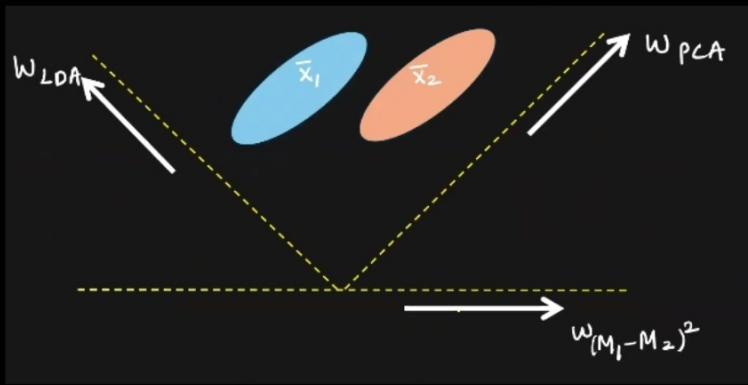
Idea

\rightarrow To reduce overlap, find the projection that tries to place the respective mean of classes as far as possible

\rightarrow Simultaneously this projection should minimise the within class scatter/variance in the projected data

$W^* = \text{argmax}_W \frac{(m_1 - m_2)^2}{S_1^2 + S_2^2}$

S_1, S_2 is the variance in projected space.



NOTATION

- Dataset $D = \{x, y\}_{i=1}^N$
- Let n_0, n_1 be # points in respective classes
- Z_i is scalar projection of x_i on w
 - w → column vectors
 - Z_i → scalars
- Let μ_0, μ_1 be the means of the data corresponding to each class 0 & 1 respectively
- m_0, m_1 → means in projected space
- Variance of data in projected space → S_0^2, S_1^2

$D = \{x, y\}_{i=1}^N$
 $x \in \mathbb{R}^d$
 $y_i \in \{0, 1\}$
 $Z_i = W^T x_i$

$C_0 \rightarrow$ class 0
 $C_1 \rightarrow$ class 1
 $n_0 = |C_0|$
 $n_1 = |C_1|$
 $N = n_0 + n_1$

$\mu_0 = \frac{1}{n_0} \sum_{x_i \in C_0} (x_i)$
 $\mu_1 = \frac{1}{n_1} \sum_{x_i \in C_1} (x_i)$

$m_0 = W^T \mu_0$
 $m_1 = W^T \mu_1$

$S_1^2 = \frac{1}{n_1} \sum_{x_i \in C_1} (W^T x_i - m_1)^2$
 $S_0^2 = \frac{1}{n_0} \sum_{x_i \in C_0} (W^T x_i - m_0)^2$

DERIVATION OF LDA

$$W^* = \underset{W}{\operatorname{argmax}} = \frac{(m_1 - m_0)^2}{S_1^2 + S_0^2} = \frac{(W^T \mu_1 - W^T \mu_0)^2}{\frac{1}{n_0} \sum_{x_i \in C_0} (W^T x_i - W^T \mu_0)^2 + \frac{1}{n_1} \sum_{x_i \in C_1} (W^T x_i - W^T \mu_1)^2} = \frac{[W^T (\mu_1 - \mu_0)]^2}{S_1^2 + S_0^2}$$

$W^* = \underset{W}{\operatorname{argmax}} \left\{ \frac{W^T (\mu_1 - \mu_0) (\mu_1 - \mu_0)^T W}{\frac{1}{n_0} \sum_{x_i \in C_0} (W^T x_i - W^T \mu_0)^2 + \frac{1}{n_1} \sum_{x_i \in C_1} (W^T x_i - W^T \mu_1)^2} \right\}$

- defined as SB
- $d \times d$ matrix $\mu_1, \mu_0 \in \mathbb{R}^{d \times 1}$
- Between class scatter matrix
- Can be thought of as $\operatorname{Var}(E(X|Y))$ or Covariance Matrix $(E(X|Y))$

$$S_0^2 = \frac{1}{n_0} \sum_{x_i \in C_0} (W^T (x_i - \mu_0))^2$$

$$S_0^2 = \frac{1}{n_0} \sum_{x_i \in C_0} W^T (x_i - \mu_0) (x_i - \mu_0)^T W$$

$$S_0^2 = W^T \left[\frac{1}{n_0} \sum_{x_i \in C_0} (x_i - \mu_0) (x_i - \mu_0)^T \right] W$$

$$S_1^2 = W^T \left[\frac{1}{n_1} \sum_{x_i \in C_1} (x_i - \mu_1) (x_i - \mu_1)^T \right] W$$

$$\Rightarrow S_0^2 + S_1^2 = W^T \left[\frac{1}{n_0} \sum_{x_i \in C_0} (x_i - \mu_0) (x_i - \mu_0)^T + \frac{1}{n_1} \sum_{x_i \in C_1} (x_i - \mu_1) (x_i - \mu_1)^T \right] W$$

$$S_0^2 + S_1^2 = W^T S_W W$$

→ Within class Scatter matrix

$$\Rightarrow W^* = \underset{W}{\operatorname{argmax}} \left\{ \frac{W^T S_B W}{W^T S_W W} \right\}$$

→ Convex function

→ convex loss function = $J(w) = \frac{w^T S_B w}{w^T S_W w}$

→ $\frac{\partial J(w)}{\partial w} = \frac{w^T S_W w \cdot (S_B + S_B^T) w - w^T S_B w \cdot (S_W + S_W^T) w}{(w^T S_W w)^2} = 0$

→ S_W & S_B are both symmetric.

⇒ $w^T S_W w \cdot \partial S_B w - w^T S_B w \cdot \partial S_W w = 0$ ⇒ $S_B w = \frac{J(w)}{w^T S_W w} S_W w$
 ↓ scaled value λ

→ Assume S_W to be invertible
 Assume S_W has full rank.
 Assume S_W invertible

← scalar terms
 ← $\frac{w^T S_W w \cdot S_B w}{w^T S_W w} = \frac{w^T S_B w \cdot S_W w}{w^T S_W w}$

→ so can be cancelled

⇒ $S_B w = \frac{w^T S_B w}{w^T S_W w} S_W w$

⇒ $S_W^{-1} S_B w = \lambda w$
 $S w = \lambda w$

→ Generalised for a eigen value problem

$S = S_W^{-1} S_B$
 $\lambda = J(w) = \frac{w^T S_B w}{w^T S_W w}$

→ But we actually want the direction of w that will maximise $J(w)$

→ For that it has to satisfy $S_W^{-1} S_B w = \lambda w$

⇒ $S_W^{-1} (\mu_1 - \mu_0) (\mu_1 - \mu_0)^T w = \lambda w$
 ↓ scalar ⇒ let it be k

Now w will be in the direction of $S_W^{-1} (\mu_1 - \mu_0)$

⇒ $S_W^{-1} (\mu_1 - \mu_0) \cdot k = \lambda w$

⇒ $S_W^{-1} (\mu_1 - \mu_0) = \left(\frac{\lambda}{k}\right) w$ → scaled
 $= \frac{J(w)}{(\mu_1 - \mu_0)^T w} w$

→ To find optimum w , we need to only calculate $S_W^{-1} (\mu_1 - \mu_0)$

Comparison with LDA as a Classifier

→ We remember in multivariate case $\delta_k(w) = \log \pi_k + x^T \Sigma^{-1} \mu_k - \frac{\mu_k^T \Sigma^{-1} \mu_k}{2}$

→ DB was $x^T \Sigma^{-1} (\mu_1 - \mu_0) = \log \left(\frac{\pi_0}{\pi_1}\right) + \frac{\mu_1^T \Sigma^{-1} \mu_1 - \mu_0^T \Sigma^{-1} \mu_0}{2}$

→ This is of the form $x^T w = -\beta$ or $x^T w + \beta = 0$

→ where w is the normal to the decision boundary. $w \propto \Sigma^{-1} (\mu_1 - \mu_0)$

→ Compare with our result $w \propto S_W^{-1} (\mu_1 - \mu_0)$

→ The difference between Σ & S_W is that, in the classifier, Σ was assumed to be class invariant. i.e. $\Sigma_1 = \Sigma_2 = \dots = \Sigma_k$ → all classes have same within class covariance matrix

→ But here S_W is the sum of covariance matrices of different classes, even though they are not class invariant

LDA algorithm

- Calculate class means μ_0, μ_1
- Calculate covariance matrix of each class
- Add and obtain within class scatter matrix S_w
- $W^* = S_w^{-1}(\mu_1 - \mu_0)$

The classes mean are:

$$\mu_1 = \frac{1}{N_1} \sum_{x \in \omega_1} x = \frac{1}{5} \left[\begin{pmatrix} 4 \\ 2 \end{pmatrix} + \begin{pmatrix} 2 \\ 4 \end{pmatrix} + \begin{pmatrix} 3 \\ 6 \end{pmatrix} + \begin{pmatrix} 4 \\ 4 \end{pmatrix} \right] = \begin{pmatrix} 3 \\ 3.8 \end{pmatrix}$$

$$\mu_2 = \frac{1}{N_2} \sum_{x \in \omega_2} x = \frac{1}{5} \left[\begin{pmatrix} 9 \\ 10 \end{pmatrix} + \begin{pmatrix} 6 \\ 8 \end{pmatrix} + \begin{pmatrix} 9 \\ 5 \end{pmatrix} + \begin{pmatrix} 8 \\ 7 \end{pmatrix} + \begin{pmatrix} 10 \\ 8 \end{pmatrix} \right] = \begin{pmatrix} 8.4 \\ 7.6 \end{pmatrix}$$

```

class means
mu1 = mean(X1)';
mu2 = mean(X2)';
    
```

LDA ... Two Classes - Example

- Compute the Linear Discriminant projection for the following two-dimensional dataset.
- Samples for class ω_1 : $X_1 = \{(4,2), (2,4), (3,6), (3,6), (4,4)\}$
- Samples for class ω_2 : $X_2 = \{(9,10), (6,8), (9,5), (8,7), (10,8)\}$

```

samples for class 1
X1 = [4,2;
      2,4;
      3,6;
      3,6;
      4,4];

samples for class 2
X2 = [9,10;
      6,8;
      9,5;
      8,7;
      10,8];
    
```

Covariance matrix of the first class:

$$S_1 = \sum_{x \in \omega_1} (x - \mu_1)(x - \mu_1)^T = \begin{bmatrix} 4-3 & 2-3.8 \\ 2-3 & 4-3.8 \end{bmatrix} + \begin{bmatrix} 2-3 & 4-3.8 \\ 4-3 & 2-3.8 \end{bmatrix} + \begin{bmatrix} 3-3 & 6-3.8 \\ 6-3 & 3-3.8 \end{bmatrix} + \begin{bmatrix} 3-3 & 6-3.8 \\ 6-3 & 3-3.8 \end{bmatrix} + \begin{bmatrix} 4-3 & 4-3.8 \\ 4-3 & 4-3.8 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & -0.25 \\ -0.25 & 2.2 \end{bmatrix} = \frac{1}{5} \sum_{x \in \omega_1} (x - \mu_1)(x - \mu_1)^T$$

```

covariance matrix of the first class
S1 = cov(X1);
    
```

Covariance matrix of the second class:

$$S_2 = \sum_{x \in \omega_2} (x - \mu_2)(x - \mu_2)^T = \begin{bmatrix} 9-8.4 & 10-7.6 \\ 10-7.6 & 6-7.6 \end{bmatrix} + \begin{bmatrix} 6-8.4 & 8-7.6 \\ 8-7.6 & 9-7.6 \end{bmatrix} + \begin{bmatrix} 9-8.4 & 5-7.6 \\ 5-7.6 & 9-7.6 \end{bmatrix} + \begin{bmatrix} 8-8.4 & 7-7.6 \\ 7-7.6 & 8-7.6 \end{bmatrix} + \begin{bmatrix} 10-8.4 & 8-7.6 \\ 8-7.6 & 10-7.6 \end{bmatrix}$$

$$= \begin{bmatrix} 2.3 & -0.05 \\ -0.05 & 3.3 \end{bmatrix}$$

```

covariance matrix of the second class
S2 = cov(X2);
    
```

Within-class scatter matrix:

$$S_w = S_1 + S_2 = \begin{bmatrix} 1 & -0.25 \\ -0.25 & 2.2 \end{bmatrix} + \begin{bmatrix} 2.3 & -0.05 \\ -0.05 & 3.3 \end{bmatrix} = \begin{bmatrix} 3.3 & -0.3 \\ -0.3 & 5.5 \end{bmatrix}$$

```

within-class scatter matrix
Sw = S1 + S2;
    
```

OR we can solve it by using eigen value problem of $S_w^{-1} S_B W = \lambda W$

$\frac{1}{N_0-1} \sum_{x \in \omega_0} (x - \mu_0)(x - \mu_0)^T$

notice the covariance matrix calculation uses discounting the cardinality by 1

Bessel's correction to account for the degree of freedom used by the mean itself.

Without Bessel's correction the covariance would be underestimation.

LDA ... Two Classes - Example

Or directly:

$$w^* = S_w^{-1}(\mu_1 - \mu_2) = \begin{bmatrix} 3.3 & -0.3 \\ -0.3 & 5.5 \end{bmatrix}^{-1} \left[\begin{pmatrix} 3 \\ 3.8 \end{pmatrix} - \begin{pmatrix} 8.4 \\ 7.6 \end{pmatrix} \right]$$

$$= \begin{bmatrix} 0.3045 & 0.0166 \\ 0.0166 & 0.1827 \end{bmatrix} \begin{pmatrix} -5.4 \\ -3.8 \end{pmatrix} = \begin{bmatrix} 0.9088 \\ 0.4173 \end{bmatrix}$$

LDA ... Two Classes - Example

The LDA projection is then obtained as the solution of the generalized eigen value problem $S_w^{-1} S_B W = \lambda W$

$$\Rightarrow |S_w^{-1} S_B - \lambda I| = 0$$

$$\Rightarrow \begin{vmatrix} 3.3 & -0.3 \\ -0.3 & 5.5 \end{vmatrix} \begin{bmatrix} 29.16 & 20.52 \\ 20.52 & 14.44 \end{bmatrix} - \lambda \begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix} = 0$$

$$\Rightarrow \begin{vmatrix} 0.3045 & 0.0166 \\ 0.0166 & 0.1827 \end{vmatrix} \begin{bmatrix} 29.16 & 20.52 \\ 20.52 & 14.44 \end{bmatrix} - \lambda \begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix} = 0$$

$$\Rightarrow \begin{vmatrix} 9.2213 - \lambda & 6.489 \\ 4.2339 & 2.9794 - \lambda \end{vmatrix} = 0$$

$$\Rightarrow (9.2213 - \lambda)(2.9794 - \lambda) - 6.489 \times 4.2339 = 0$$

$$\Rightarrow \lambda^2 - 12.2007\lambda = 0 \Rightarrow \lambda(\lambda - 12.2007) = 0$$

$$\Rightarrow \lambda_1 = 0, \lambda_2 = 12.2007$$

```

computing the LDA projection
invSw_by_Sb = invSw * Sb;
hence
[9.2213 6.489
 4.2339 2.9794] w1 = [w1_1; w1_2]
and
[9.2213 6.489
 4.2339 2.9794] w2 = [12.2007; 0]
Thus:
w1 = [-0.5755; 0.8178] and w2 = [0.9088; 0.4173] = w*
The optimal projection is the one that given maximum lambda = lambda_2
    
```

Both method works but eigen value method additional requires calculating S_B

Lagrangian view point

$$W^* = \operatorname{argmax}_W J(W) = \operatorname{argmax}_W \frac{W^T S_B W}{W^T S_w W}$$

$$\Rightarrow L(W, \lambda) = W^T S_B W - \lambda (W^T S_w W - 1)$$

$$\frac{\partial L}{\partial W} = 2S_B W - \lambda 2S_w W = 0$$

$$\Rightarrow S_B W = \lambda S_w W$$

$$\Rightarrow S_w^{-1} S_B W = \lambda W \quad \text{Eigen value problem}$$

$$\Rightarrow S_w^{-1} (\mu_1 - \mu_0) (\mu_1 - \mu_0)^T W = \lambda W$$

λ scalar

$$\Rightarrow S_w^{-1} (\mu_1 - \mu_0) = \frac{\lambda}{K} W$$

$$\Rightarrow W \propto S_w^{-1} (\mu_1 - \mu_0)$$

Equivalently this can be converted to a constrained optimisation problem s.t

$$W^* = \operatorname{argmax}_W W^T S_B W \quad | \quad W^T S_w W = 1$$

Because we only care about direction not the magnitude.

$$\frac{\partial L}{\partial \lambda} = 0 \Rightarrow -W^T S_w W + 1 = 0$$

$$W^T S_w W = 1$$

At optimum W^*, λ^*

$$W^{*T} S_B W^* = W^{*T} \lambda^* S_w W^* = \lambda^* W^{*T} S_w W^* = \lambda^* = 1$$

$$\Rightarrow W^{*T} S_B W^* = \lambda^*$$

$$J(W^*) = \frac{W^{*T} S_B W^*}{W^{*T} S_w W^*} = \lambda^*$$

The optimum W^*, λ^* is the solution to the generalised eigen value problem

$$S_w^{-1} S_B W = \lambda W$$

$W^* \rightarrow$ Eigen vector corresponding to largest eigen value of $S_w^{-1} S_B$

$\lambda^* \rightarrow$ largest eigen value of $S_w^{-1} S_B$

Multiclass Intuition

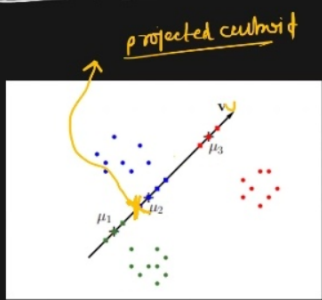
- ① Mean of projected class should be as far as possible
- ② Classes should be kept as tight as possible in projected space.

→ Choose common point → Centroid of class means

→ Sense of distance = $(\mu_1 - \mu_c)^2 + (\mu_2 - \mu_c)^2 + (\mu_3 - \mu_c)^2 + \dots + (\mu_k - \mu_c)^2$

But we need to weight on the number of points otherwise biased towards minority class

Multiclass intuition



- ① means of projected class (separation maximise)
- ② classes should be as tight as possible

$$[(x_1 - \mu) + (x_2 - \mu) + (x_3 - \mu) \dots]$$

$$J(W) = \frac{W^T S_B W}{W^T S_W W}$$

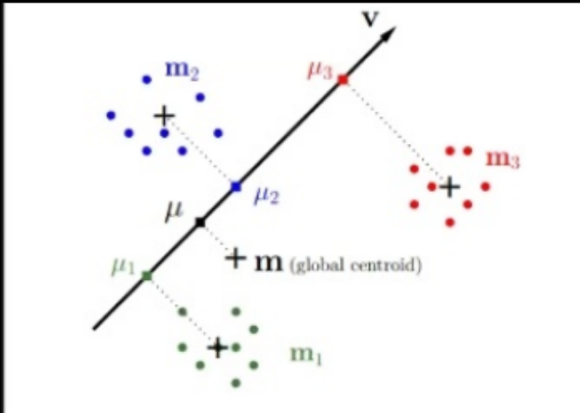
$$\text{distance} = n_1(\mu_1 - \mu_c)^2 + n_2(\mu_2 - \mu_c)^2 + \dots + n_k(\mu_k - \mu_c)^2$$

$$\mu_c = \frac{1}{k} \sum_{i=1}^k \mu_i$$

k = #classes

$$S_B = \sum_{i=1}^k n_i (\mu_i - \mu_c)(\mu_i - \mu_c)^T$$

$$S_W = \sum_{i=1}^k S_i^2$$



Objective function

Since the projection is no longer a scalar (it has C-1 dimensions), we then use the determinant of the scatter matrices to obtain a scalar objective function:

$$J(W) = \frac{|\tilde{S}_B|}{|\tilde{S}_W|} = \frac{|W^T S_B W|}{|W^T S_W W|}$$

W → Matrix

$$\mathbb{R}^n \rightarrow \mathbb{R}^m$$

$$k \rightarrow \# \text{classes}$$

$$N \rightarrow \# \text{datapoints}$$

$$n_i \rightarrow |c_i| \rightarrow \# \text{point in class } i$$

$$W = N \times m \text{ matrix}$$

How many discriminatory directions can we find?

Very important point

one can only find at most c - 1 discriminatory directions.

$$c = k = \# \text{classes}$$

$$\Rightarrow m \leq k - 1$$

$$m \leq c - 1$$



2 classes ⇒ 1 discriminatory direction

Limitations of LDA

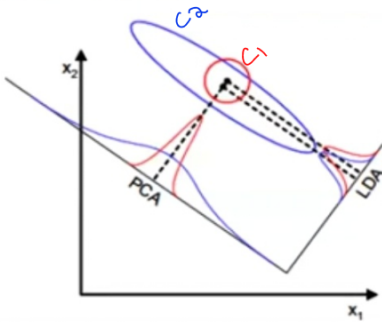
LDA produces at most C-1 feature projections

- If the classification error estimates establish that more features are needed, some other method must be employed to provide those additional features

LDA is a parametric method since it assumes unimodal Gaussian likelihoods

- If the distributions are significantly non-Gaussian, the LDA projections will not be able to preserve any complex structure of the data, which may be needed for classification.

LDA will fail when the discriminatory information is not in the mean but rather in the variance of the data



→ mean μ_1 & μ_2 are almost same
 → $\text{Var}(c_1)$ & $\text{Var}(c_2)$ is the discriminatory information

Comparison between PCA and LDA

	PCA	LDA
Use labels?	no (unsupervised)	yes (supervised)
Criterion	variance	discrimination
#dimensions (k)	any	$\leq c - 1$
Computing	SVD	generalized eigenvectors
Linear projection?	yes $((x - m)^T V)$	yes $(x^T V)$
Nonlinear boundary	can handle*	cannot handle

we showed that discriminant $f_k(x)$ is linear in x
 → cannot handle unless feature transformation

Q.22 For any binary classification dataset, let $S_B \in \mathbb{R}^{d \times d}$ and $S_W \in \mathbb{R}^{d \times d}$ be the between-class and within-class scatter (covariance) matrices, respectively. The Fisher linear discriminant is defined by $u^* \in \mathbb{R}^d$, that maximizes

$$J(u) = \frac{u^T S_B u}{u^T S_W u}$$

If $\lambda = J(u^*)$, S_W is non-singular and $S_B \neq 0$, then (u^*, λ) must satisfy which ONE of the following equations?

Note: \mathbb{R} denotes the set of real numbers.

- (A) $S_W^{-1} S_B u^* = \lambda u^*$ ✓
- (B) $S_W u^* = \lambda S_B u^*$
- (C) $S_B S_W u^* = \lambda u^*$
- (D) $u^{*T} u^* = \lambda^2$