COMP30027 MACHINE LEARNING TUTORIAL

Workshop - 9

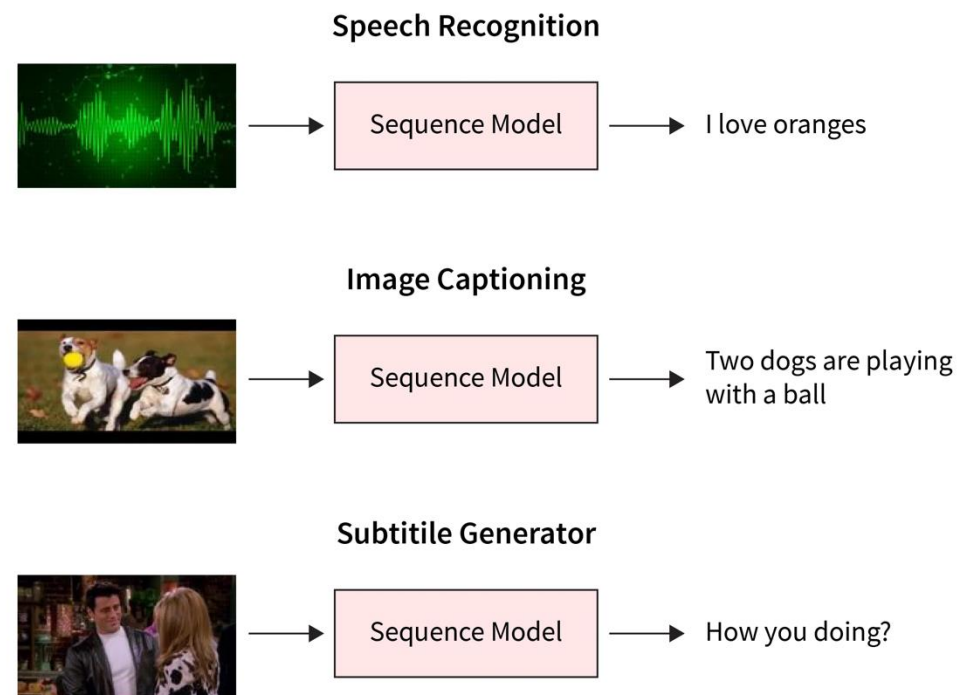# Sequence Modeling and Perceptron

We'll examine Hidden Markov Models, their algorithms, and perceptrons.

Sequential Models – What & Why

A **sequential model** is any model where the **output or current state depends on previous elements in a sequence**. Sequential models are used when **data is ordered** – e.g., speech, text, time series, video.

Example: "I am going to _" – the blank depends on previous words.

Common use-cases:

Speech Recognition

Sequence Model → I love oranges

Image Captioning

Sequence Model → Two dogs are playing with a ball

Subtitle Generator

Sequence Model → How you doing?

Hidden Markov Model Fundamentals

A Hidden Markov Model (HMM) is a probabilistic sequence model that assumes the current hidden state only depends on the immediately previous state.

- We can see a sequence of observations, but the sequence of states is hidden
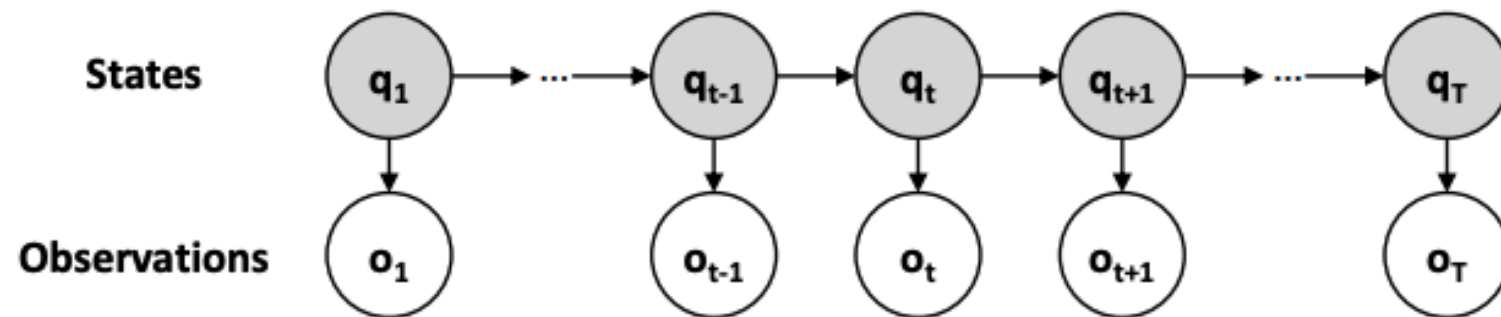
- Hidden Markov Models (HMM): $\mu = (A, B, \Pi)$
  - **States**: a set $S = \{s_i\}$ with $|S|$ unique values
  - **Observations**: a set $Y = \{y_k\}$ with K unique values
  - **Initial state distribution**: $\Pi = \{\pi_i\}, \sum_i \pi_i = 1$
  - **Transition probability matrix**: $A = \{a_{ij}\}, \sum_j a_{ij} = 1$ for $\forall i$ —→ (likelihood of going from one state to another)
  - **Output probability matrix**: $B = \{b_{ik}\}, \sum_k b_{ik} = 1$ for $\forall i$ —→ (How likely it is to "see" a certain word (or observation), given that you're in a specific hidden state.)
 $b_{ik} = b_i(o_t) = P(o_t = y_k | q_t = s_i)$

HMM helps us to **infer hidden information** based on observed data.

HMM – Sample Example

In HMMs of textual data or natural language processing (NLP), **hidden states are categorical variables** (e.g., part-of-speech (POS) tags or speech phonemes) that we don't observe directly, but which influence what we **do** observe (e.g., words or sounds).

Let's say your sentence is:

"She eats apples"
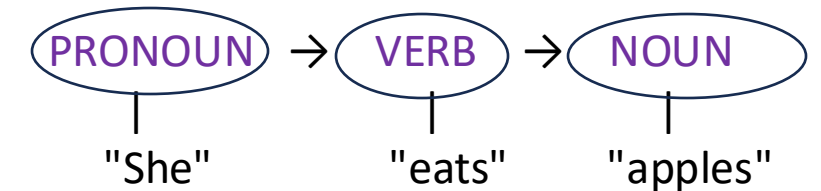
For a Hidden Markov Model in the domain of NLP, we have the following values:

- **Observations (Y):** ["She", "eats", "apples"]

- **Hidden states (S —POS tags):** [PRONOUN, VERB, NOUN]

- **Transition probabilities** $P(s_{i+1} | s_i)$: How likely is it to go from s_i to s_{i+1}?; e.g. $P(VERB | NOUN)$ is the probability that current tag is Verb given previous tag is a Noun.

- **Output probabilities** $P(y_i | s_i)$: Probability of a word (y_i) being generated from a POS tag (s_i); e.g. $P(eats | VERB)$ is the probability that the word *eats* is a Verb.

- **Initial probabilities** $\pi(si)$: Probability of a POS tag being the first tag; e.g. $\pi(VERB)$ is the probability of in the training set

PRONOUN → VERB → NOUN

"She"   "eats"   "apples"

- Each arrow between states = **transition probability** (e.g., P(VERB | PRONOUN))

- Each arrow from state to word = **Output probability** (e.g., P("eats" | VERB))

Two Main Assumptions in HMMs

HMMs are **generative models**: they try to model the **joint probability** of hidden states and observations. They assume:

1. Markov Assumption (First-Order Markov Property)

The likelihood of transitioning into a state at time t depends **only on the previous state** q_{t-1}, not on any older states or outputs.

Formally:

$$P(q_t \mid q_1, q_2, \ldots, q_{t-1}) \approx P(q_t \mid q_{t-1})$$

This means:

- The **hidden state sequence** is modeled as a **first-order Markov chain**
- The model assumes **"limited memory"** — it only "remembers" the last state

Example:

Suppose you're tagging: "**The cat sat on the _____**"

You're trying to predict the POS tag for the word "**mat**".

Under the **Markov assumption**, the HMM assumes:

The tag of "**mat**" depends **only on the tag of "the" (Determiner)**,

and **not** on the fact that the previous words were "The cat sat on".

2. Output Independence (Emission Independence Assumption)

The likelihood of generating observation o_t at time t depends **only on the current hidden state** q_t — and not on the full sequence of previous/future observations or states.

Formally:

$$P(o_t \mid q_1, \ldots, q_t, o_1, \ldots, o_{t-1}) \approx P(o_t \mid q_t)$$

Example:

Imagine a **weather HMM**:

- Hidden state = today's weather
- Observation = clothes you see people wearing

Output independence says:

Once I know it's rainy today, the clothes people wear (like umbrellas or raincoats) depend only on today's weather — not on what the weather was yesterday.

HMM - Overall Process

1. Training data

- Input to a **Supervised HMM** is a set of **observation sequences** (e.g., words in sentences) and **their corresponding hidden states** (e.g., POS tags).

2. Train the HMM:

- From this training data, HMM model computes:

Initial state distribution (π) — $\dfrac{\textit{Count of how often each state appears in the }\textbf{first position}\textit{ of training sequences}}{\textit{Total number of sequences.}}$

Transition matrix (A) — $\dfrac{\textit{Count of how many times the state transition from i to j (e.g., from Verb to Noun) occurred in the training data}}{\textit{Total number of times the state i (e.g., Verb) occurs in the training dataset}}$

Output probability matrix (B) — $\dfrac{\textit{Count of how many times the observation (e.g., words) was observed labelled with state (e.g., POS tag) in the training data}}{\textit{Total number of times the state (e.g., POS tag) occurs in the training dataset}}$

Computed from the dataset

3. Use the model

- Now, you can give a **new observation sequence** (like a sentence or audio signal), and ask the HMM to do one of the following tasks:

    How likely is this sequence generated by the model? — **Evaluation** (Forward Algorithm)

    What is the most likely sequence of hidden states that could produce this observation? — **Decoding** (Viterbi Algorithm)

Example 1: HMM for Text Classification

Imagine you have 3 datasets containing texts from financial news, sports, and entertainment, respectively. Given a new sentence, you have to determine whether it belongs to **financial news**, **sports, or entertainment.**

To determine **which category a sentence belongs to**, you train a separate HMM for each category using its respective sentences.

Each HMM will learn:

π (initial state probabilities)

A (state transition matrix) — how tags (e.g., POS patterns) follow each other

B (emission matrix) — how words are emitted from states

You now have 3 models: HMM_finance, HMM_sports, HMM_films

Now, given a new sentence:

"The bank interest rates are increasing."

You want to know:

Which HMM is most likely to have generated this sentence?

We use **Forward Algorithm** to compute **p(sentence|** HMM_finance), **p(sentence|** HMM_sports), and **p(sentence|** HMM_films)

Choose the category with the **highest probability**

Forward Algorithm: Probability Calculation

This algorithm computes the **total probability of observing the sequence**

In the context of HMMs, **"t" is the time step** — meaning the **position in the sequence of observations**.

Steps:

1. **Initialization (t = 1):**

   - Multiply the **initial state probability** π and **emission probability** for the first word.

2. **Recursion (t = 2 to T):**

   - For each state s_j, sum over all **previous states** s_i:

$$\alpha_t(s_j) = B(s_j, o_t) \cdot \sum_{s_i} \alpha_{t-1}(s_i) \cdot A(s_i, s_j)$$

total probability of seeing first t words/observations and ending in state s_j

3. **Termination:**

   - Sum the final values α_T to get total probability of the sequence.

In the case of the previous text classification example, for the model HMM_finance at t = T, α_T has the value for **p(sentence| HMM_finance)**

Example 2: HMM for Weather Prediction

Think of a weather system where you only observe what people carry or wear (umbrellas, T-shirt, etc) but not the actual weather. You want to guess the weather (hidden state) from the presence of umbrellas, T-shirt, etc (observation)

1. **Training Data (**You collect data for several days):

Day	Actual Weather (**Hidden state**)	What people wore (**observation**)
1	Sunny	Sunglasses
2	Sunny	T-shirt
3	Rainy	Umbrella
4	Cloudy	Raincoat
5	Sunny	Sunglasses

2. From this, the model **learns**:

   - How weather **transitions** over time e.g., Sunny → Rainy (**Transition matrix**)

   - What clothing is usually seen under each weather condition e.g., Sunny — Sunglasses (**Output probability matrix**)

3. **Now, given a sequence of observations, like:**

   ["Umbrella", "Raincoat", "T-shirt"]

   You want to know:

   • What's the most likely sequence of weather or **hidden states** (e.g., Rainy → Cloudy → Sunny) that caused these observations?

   Viterbi Algorithm

Viterbi Algorithm

This finds the **most likely sequence of states** that produced the observations.

Steps:

the best scores of the path to reach a state s_j at a given time t

1. **Initialization (t = 1):** $\delta_1(s_j) = \pi(s_j) \cdot B(s_j, o_1)$

   - Same as forward, but store only the **maximum** probability per state.

2. **Recursion (t = 2 to T):** $\delta_t(s_j) = B(s_j, o_t) \cdot \max_{s_i \in S} [\delta_{t-1}(s_i) \cdot A(s_i, s_j)]$

   Unlike forward algorithm which stores the total sum of probability of all paths to the state, Viterbi only considers the path with maximum probability

   - For each current state, keep the **maximum path** coming into it (instead of sum).

3. **Backtracking:**

   - Start from the most likely final state and trace back the **most likely path** of states.

   Track the path using **backpointers** (Used to trace the best path backwards). It stores the **index of the best previous state** that led to each current state

So, if you saw ["Umbrella", "Raincoat", "T-shirt"]— Viterbi might tell you the most likely weather sequence was: Rainy → Cloudy → Sunny. You're not just asking "could this sequence happen?" (as in the forward algorithm), you're asking "**what sequence of hidden states probably did happen**?"

Q2

Natural language processing is one common application for HMMs: we have a single observation (a "word") that varies over time (a "sentence" or "document"), where each observation is associated with some property (like "part of speech").

Consider the following HMM: $\Pi[J, N, V] = [0.3, 0.4, 0.3]$

A	J (adj)	N (noun)	V (verb)
J	0.4	0.5	0.1
N	0.1	0.4	0.5
V	0.4	0.5	0.1

B	brown	leaves	turn
J	0.8	0.1	0.1
N	0.3	0.4	0.3
V	0.1	0.3	0.6

1. How might we go about obtaining the values in the matrices Π, A, and B given above, in a supervised context?
2. Use the **forward** algorithm to find the probability of the "sentence" `brown leaves turn`.
3. Use the **Viterbi** algorithm to find the most likely state sequence for the sentence `brown leaves turn`.

Q1 How do we obtain the values of Π, A, and B in a supervised setting?

In a **supervised context**, you are given both:

- the **observation sequences** (like: "brown leaves turn"), and

- the **true state sequences** (like: "Adj Noun Verb").

1. Initial State Distribution (Π)

$$\Pi[s] = \frac{\text{Number of sentences that start with state } s}{\text{Total number of sentences}}$$
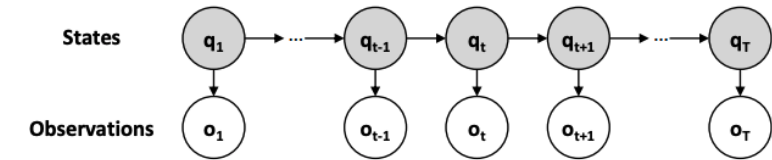
So, if out of 10 sentences:

- 3 start with Adjective (J)

- 4 start with Noun (N)

- 3 start with Verb (V)

  Then:

$$\Pi = [0.3, 0.4, 0.3]$$

- We can see a sequence of observations, but the sequence of states is hidden

States: $q_1 \to \cdots \to q_{t-1} \to q_t \to q_{t+1} \to \cdots \to q_T$

Observations: $o_1, o_{t-1}, o_t, o_{t+1}, o_T$

- Hidden Markov Models (HMM): $\mu = (A, B, \Pi)$
  - **States**: a set $S = \{s_i\}$ with $|S|$ unique values
  - **Observations**: a set $Y = \{y_k\}$ with K unique values
  - **Initial state distribution**: $\Pi = \{\pi_i\}, \sum_i \pi_i = 1$
  - **Transition probability matrix**: $A = \{a_{ij}\}, \sum_j a_{ij} = 1$ for $\forall i$
  - **Output probability matrix**: $B = \{b_{ik}\}, \sum_k b_{ik} = 1$ for $\forall i$
    $b_{ik} = b_i(o_t) = P(o_t = y_k | q_t = s_i)$

2. Transition Matrix (A)

$$A[i][j] = \frac{\text{Number of transitions from state } i \text{ to state } j}{\text{Number of times state } i \text{ occurs}}$$

Example: If N appears 10 times (not at the end) and you see 4 times it transitions to V, 3 to N, and 3 to J:

$$A[N][V] = \frac{4}{10}, A[N][N] = \frac{3}{10}, A[N][J] = \frac{3}{10}$$

3. Output Probability Matrix (B)

$$B[s][w] = \frac{\text{Number of times word } w \text{ is observed when the tag is } s}{\text{Total number of times state } s \text{ appears}}$$

In simpler words:

- **Numerator**: Count of how many times "brown" is labeled as adjective (J)

- **Denominator**: Count of **all words** labeled as adjective (J), regardless of which word they were

Q2. Use the Forward Algorithm to compute P(brown leaves turn)

Given:

- **States (S)** = [J, N, V]
- **Observations** = ["brown", "leaves", "turn"]
- **Π** = [0.3, 0.4, 0.3]

A (Transition Matrix)

From ↓ / To →	J	N	V
J	0.4	0.5	0.1
N	0.1	0.4	0.5
V	0.4	0.5	0.1

B (Emission Matrix):

State ↓ / Word →	brown	leaves	turn
J	0.8	0.1	0.1
N	0.3	0.4	0.3
V	0.1	0.3	0.6

This algorithm computes the **probability of observing the sequence.**

Steps:

1. **Initialization (t = 1):**

   - Multiply the **initial state probability** π and **emission probability** for the first word.

2. **Recursion (t = 2 to T):**

   - For each state s_j, sum over all **previous states** s_i:

$$\alpha_t(s_j) = B(s_j, o_t) \cdot \sum_{s_i} \alpha_{t-1}(s_i) \cdot A(s_i, s_j)$$

3. **Termination:**

   - Sum the final values α_T to get total probability of the sequence.

Given:

- **States (S) = [J, N, V]**

- **Observations = ["brown", "leaves", "turn"]**

- Π = [0.3, 0.4, 0.3]

$\alpha_t(s)$ = Total probability of all paths to a state s at time t

Step 1: Initialization (t=1, word = "brown")

$$\alpha_1(J) = \Pi(J) \cdot B(J,\text{"brown"}) = 0.3 \cdot 0.8 = 0.24$$

$$\alpha_1(N) = 0.4 \cdot 0.3 = 0.12$$

$$\alpha_1(V) = 0.3 \cdot 0.1 = 0.03$$

A (Transition Matrix)

From ↓ / To →	J	N	V
J	0.4	0.5	0.1
N	0.1	0.4	0.5
V	0.4	0.5	0.1

B (Observation Matrix)

State ↓ / Word →	brown	leaves	turn
J	0.8	0.1	0.1
N	0.3	0.4	0.3
V	0.1	0.3	0.6

Step 1: Initialization (t=1, word = "brown")

$$\alpha_1(J) = \Pi(J) \cdot B(J,"brown") = 0.3 \cdot 0.8 = 0.24$$

$$\alpha_1(N) = 0.4 \cdot 0.3 = 0.12$$
$$\alpha_1(V) = 0.3 \cdot 0.1 = 0.03$$

Step 2: t = 2, word = "leaves"

$$\alpha_2(J) = B(J,"leaves") \cdot [\alpha_1(J) \cdot A[J][J] + \alpha_1(N) \cdot A[N][J] + \alpha_1(V) \cdot A[V][J]]$$

$$= 0.1 \cdot (0.24 \cdot 0.4 + 0.12 \cdot 0.1 + 0.03 \cdot 0.4)$$
$$= 0.1 \cdot (0.096 + 0.012 + 0.012) = 0.1 \cdot 0.12 = 0.012$$

$$\alpha_2(N) = 0.4 \cdot (0.24 \cdot 0.5 + 0.12 \cdot 0.4 + 0.03 \cdot 0.5)$$
$$= 0.4 \cdot (0.12 + 0.048 + 0.015) = 0.4 \cdot 0.183 = 0.0732$$

$$\alpha_2(V) = 0.3 \cdot (0.24 \cdot 0.1 + 0.12 \cdot 0.5 + 0.03 \cdot 0.1)$$
$$= 0.3 \cdot (0.024 + 0.06 + 0.003) = 0.3 \cdot 0.087 = 0.0261$$

This algorithm computes the **probability of observing the sequence.**

Steps:

1. **Initialization (t = 1):**
   - Multiply the **initial state probability** π and **emission probability** for the first word.

2. **Recursion (t = 2 to T):**
   - For each state s_j, sum over all **previous states** s_i:

   $$\alpha_t(s_j) = B(s_j, o_t) \cdot \sum_{s_i} \alpha_{t-1}(s_i) \cdot A(s_i, s_j)$$

3. **Termination:**
   - Sum the final values α_T to get total probability of the sequence.

A (Transition Matrix)

From ↓ / To →	J	N	V
J	0.4	0.5	0.1
N	0.1	0.4	0.5
V	0.4	0.5	0.1

B (Observation Matrix)

State ↓ / Word →	brown	leaves	turn
J	0.8	0.1	0.1
N	0.3	0.4	0.3
V	0.1	0.3	0.6

Step 3: t = 3, word = "turn"

$$\alpha_3(J) = 0.1 \cdot (0.012 \cdot 0.4 + 0.0732 \cdot 0.1 + 0.0261 \cdot 0.4)$$
$$= 0.1 \cdot (0.0048 + 0.00732 + 0.01044) = 0.1 \cdot 0.02256 = 0.002256$$

$$\alpha_3(N) = 0.3 \cdot (0.012 \cdot 0.5 + 0.0732 \cdot 0.4 + 0.0261 \cdot 0.5)$$
$$= 0.3 \cdot (0.006 + 0.02928 + 0.01305) = 0.3 \cdot 0.04833 = 0.014499$$

$$\alpha_3(V) = 0.6 \cdot (0.012 \cdot 0.1 + 0.0732 \cdot 0.5 + 0.0261 \cdot 0.1)$$
$$= 0.6 \cdot (0.0012 + 0.0366 + 0.00261) = 0.6 \cdot 0.04041 = 0.024246$$

Final: Sum over all α₃ values

$$P(\text{brown leaves turn}) = 0.002256 + 0.014499 + 0.024246 = \boxed{0.041001}$$

This algorithm computes the **probability of observing the sequence.**

Steps:

1. **Initialization (t = 1):**

   - Multiply the **initial state probability** π and **emission probability** for the first word.

2. **Recursion (t = 2 to T):**

   - For each state s_j, sum over all **previous states** s_i:

   $$\alpha_t(s_j) = B(s_j, o_t) \cdot \sum_{s_i} \alpha_{t-1}(s_i) \cdot A(s_i, s_j)$$

3. **Termination:**

   - Sum the final values α_T to get total probability of the sequence.

A (Transition Matrix)

From ↓ / To →	J	N	V
J	0.4	0.5	0.1
N	0.1	0.4	0.5
V	0.4	0.5	0.1

B (Observation Matrix)

State ↓ / Word →	brown	leaves	turn
J	0.8	0.1	0.1
N	0.3	0.4	0.3
V	0.1	0.3	0.6

Q3. Use Viterbi Algorithm to find the most likely state sequence

We follow the same structure, but take **max instead of sum**, and keep **backpointers**.

Given:

- **States (S)** = [J, N, V]

- **Observations** = ["brown", "leaves", "turn"]

- Π = [0.3, 0.4, 0.3]

This finds the **most likely sequence of states** that produced the observations.

Steps:

1. **Initialization (t = 1):** $\delta_1(s_j) = \pi(s_j) \cdot B(s_j, o_1)$

   - Same as forward, but store only the **maximum** probability per state.

2. **Recursion (t = 2 to T):** $\delta_t(s_j) = B(s_j, o_t) \cdot \max_{s_i \in S} [\delta_{t-1}(s_i) \cdot A(s_i, s_j)]$

   - For each current state, keep the **maximum path** coming into it (instead of sum).

3. **Backtracking:**

   - Start from the most likely final state and trace back the **most likely path** of states

Step 1: Initialization

$$\delta_1(s_j) = \pi(s_j) \cdot B(s_j, \text{"brown"})$$

$$\boxed{\delta_1(J) = 0.3 \cdot 0.8 = 0.24}$$
$$\delta_1(N) = 0.4 \cdot 0.3 = 0.12$$
$$\delta_1(V) = 0.3 \cdot 0.1 = 0.03$$

A (Transition Matrix)

From ↓ / To →	J	N	V
J	0.4	0.5	0.1
N	0.1	0.4	0.5
V	0.4	0.5	0.1

B (Observation Matrix)

State ↓ / Word →	brown	leaves	turn
J	0.8	0.1	0.1
N	0.3	0.4	0.3
V	0.1	0.3	0.6

$$\delta_1(J) = 0.3 \cdot 0.8 = 0.24$$
$$\delta_1(N) = 0.4 \cdot 0.3 = 0.12$$
$$\delta_1(V) = 0.3 \cdot 0.1 = 0.03$$

This finds the **most likely sequence of states** that produced the observations.

Steps:

1. **Initialization (t = 1):** $\quad \delta_1(s_j) = \pi(s_j) \cdot B(s_j, o_1)$

   - Same as forward, but store only the **maximum** probability per state.

2. **Recursion (t = 2 to T):** $\quad \delta_t(s_j) = B(s_j, o_t) \cdot \max_{s_i \in S} \left[\delta_{t-1}(s_i) \cdot A(s_i, s_j) \right]$

   - For each current state, keep the **maximum path** coming into it (instead of sum).

3. **Backtracking:**

   - Start from the most likely final state and trace back the **most likely path** of states

Step 2: t=2 (word = leaves)

$\delta_2(J)$

$= 0.1 \cdot \max(0.24 \cdot 0.4, 0.12 \cdot 0.1, 0.03 \cdot 0.4) = 0.1 \cdot \max(0.096, 0.012, 0.012) = 0.1 \cdot 0.096 = 0.0096$

$\delta_2(N)$

$= 0.4 \cdot \max(0.24 \cdot 0.5, 0.12 \cdot 0.4, 0.03 \cdot 0.5) = 0.4 \cdot \max(0.12, 0.048, 0.015) = 0.4 \cdot 0.12 = 0.048$

$\delta_2(V)$

$= 0.3 \cdot \max(0.24 \cdot 0.1, 0.12 \cdot 0.5, 0.03 \cdot 0.1) = 0.3 \cdot \max(0.024, 0.06, 0.003) = 0.3 \cdot 0.06 = 0.018$

Step 3: t=3 (word = turn)

$\delta_3(J)$

$= 0.1 \cdot \max(0.0096 \cdot 0.4, 0.048 \cdot 0.1, 0.018 \cdot 0.4) = 0.1 \cdot \max(0.00384, 0.0048, 0.0072) = 0.1 \cdot 0.0072 = 0.00072$

$\delta_3(N)$

$= 0.3 \cdot \max(0.0096 \cdot 0.5, 0.048 \cdot 0.4, 0.018 \cdot 0.5) = 0.3 \cdot \max(0.0048, 0.0192, 0.009) = 0.3 \cdot 0.0192 = 0.00576$

$\delta_3(V)$

$= 0.6 \cdot \max(0.0096 \cdot 0.1, 0.048 \cdot 0.5, 0.018 \cdot 0.1) = 0.6 \cdot \max(0.00096, 0.024, 0.0018) = 0.6 \cdot 0.024 = 0.0144$

A (Transition Matrix)

From ↓ / To →	J	N	V
J	0.4	0.5	0.1
N	0.1	0.4	0.5
V	0.4	0.5	0.1

B (Observation Matrix)

State ↓ / Word →	brown	leaves	turn
J	0.8	0.1	0.1
N	0.3	0.4	0.3
V	0.1	0.3	0.6

$$\delta_1(J) = 0.3 \cdot 0.8 = 0.24$$
$$\delta_1(N) = 0.4 \cdot 0.3 = 0.12$$
$$\delta_1(V) = 0.3 \cdot 0.1 = 0.03$$

backtracking

This finds the **most likely sequence of states** that produced the observations.

Steps:

1. **Initialization (t = 1):** $\delta_1(s_j) = \pi(s_j) \cdot B(s_j, o_1)$
   - Same as forward, but store only the **maximum** probability per state.

2. **Recursion (t = 2 to T):** $\delta_t(s_j) = B(s_j, o_t) \cdot \max_{s_i \in S} [\delta_{t-1}(s_i) \cdot A(s_i, s_j)]$
   - For each current state, keep the **maximum path** coming into it (instead of sum).

3. **Backtracking:**
   - Start from the most likely final state and trace back the **most likely path** of states

Step 2: t=2 (word = leaves)

$\delta_2(J)$

$= 0.1 \cdot \max(0.24 \cdot 0.4, 0.12 \cdot 0.1, 0.03 \cdot 0.4) = 0.1 \cdot \max(0.096, 0.012, 0.012) = 0.1 \cdot 0.096 = 0.0096$

$\delta_2(N)$

(J,   N,   V)

$= 0.4 \cdot \max(0.24 \cdot 0.5, 0.12 \cdot 0.4, 0.03 \cdot 0.5) = 0.4 \cdot \max(0.12, 0.048, 0.015) = 0.4 \cdot 0.12 = 0.048$

$\delta_2(V)$

$= 0.3 \cdot \max(0.24 \cdot 0.1, 0.12 \cdot 0.5, 0.03 \cdot 0.1) = 0.3 \cdot \max(0.024, 0.06, 0.003) = 0.3 \cdot 0.06 = 0.018$

backtracking

Step 3: t=3 (word = turn)

$\delta_3(J)$

$= 0.1 \cdot \max(0.0096 \cdot 0.4, 0.048 \cdot 0.1, 0.018 \cdot 0.4) = 0.1 \cdot \max(0.00384, 0.0048, 0.0072) = 0.1 \cdot 0.0072 = 0.00072$

$\delta_3(N)$

$= 0.3 \cdot \max(0.0096 \cdot 0.5, 0.048 \cdot 0.4, 0.018 \cdot 0.5) = 0.3 \cdot \max(0.0048, 0.0192, 0.009) = 0.3 \cdot 0.0192 = 0.00576$

$\delta_3(V)$

(J,   N,   V)

$= 0.6 \cdot \max(0.0096 \cdot 0.1, 0.048 \cdot 0.5, 0.018 \cdot 0.1) = 0.6 \cdot \max(0.00096, 0.024, 0.0018) = 0.6 \cdot 0.024 = 0.0144$

V

A **backpointer**: the state that led to the max score

A (Transition Matrix)

From ↓ / To →	J	N	V
J	0.4	0.5	0.1
N	0.1	0.4	0.5
V	0.4	0.5	0.1

B (Observation Matrix)

State ↓ / Word →	brown	leaves	turn
J	0.8	0.1	0.1
N	0.3	0.4	0.3
V	0.1	0.3	0.6

So the most likely state sequence is:

(J ⟶ N ⟶ V)

Summary Table

Task	Main Algorithm	What it gives you
Decoding	Viterbi algorithm	Best sequence of hidden states for the observations
Evaluation	Forward algorithm	Total probability of the observation sequence

Text classification Observation: "Bank interest rates are increasing."

"How likely is this sentence in financial news?" → Forward algorithm

"What are the most likely tags (hidden states) for each word?" → Viterbi algorithm

Weather prediction observation: ["Umbrella", "Raincoat", "T-shirt"]

"*What is the total probability that this sequence of observations came from the model***?"** → Forward Algorithm

"What's the most likely sequence of weather (hidden states) for these observations? → Viterbi algorithm

Can We Relax Markov Assumptions in HMM?

Yes, we can — but it comes at a **cost**.

Relaxing the **Markov assumption:**

We move from **first-order** to **higher-order HMMs**:

- Use **pairs or sequences of previous states**:

$$P(q_t \mid q_{t-1}, q_{t-2}) \quad \text{(second-order HMM)}$$

Well, we could have pairs of states in the conditions for our transition probability matrix A, and pairs of states in the conditions for our output probability matrix B, but this will vastly increase the number of parameters in the model.

Can HMMs Handle Multivariate Observations?

Yes — this leads to **Multivariate or Vector HMMs**.

Instead of having a **scalar observation o_t**, each observation becomes a **vector**:

$$\vec{o}_t = \left(o_{t1}, o_{t2}, \ldots, o_{td}\right)$$

For example, in activity recognition from sensors:

- $\vec{o}_t = [\text{acceleration, rotation, orientation}]$

Again, at the cost of vastly increasing the number of parameters.

The Single-Layer Perceptron Architecture

A **perceptron** is a basic unit of a neural network. It performs binary classification: $f(\mathbf{w} \cdot \mathbf{x_i} + b) = \begin{cases} 1 & if\ \mathbf{w} \cdot \mathbf{x_i} + b \geq 0 \\ 0 & otherwise \end{cases}$

b

X₁

w1

X₂

w2

w3

X₃

$\sum$

Input Sum

Activation
Function
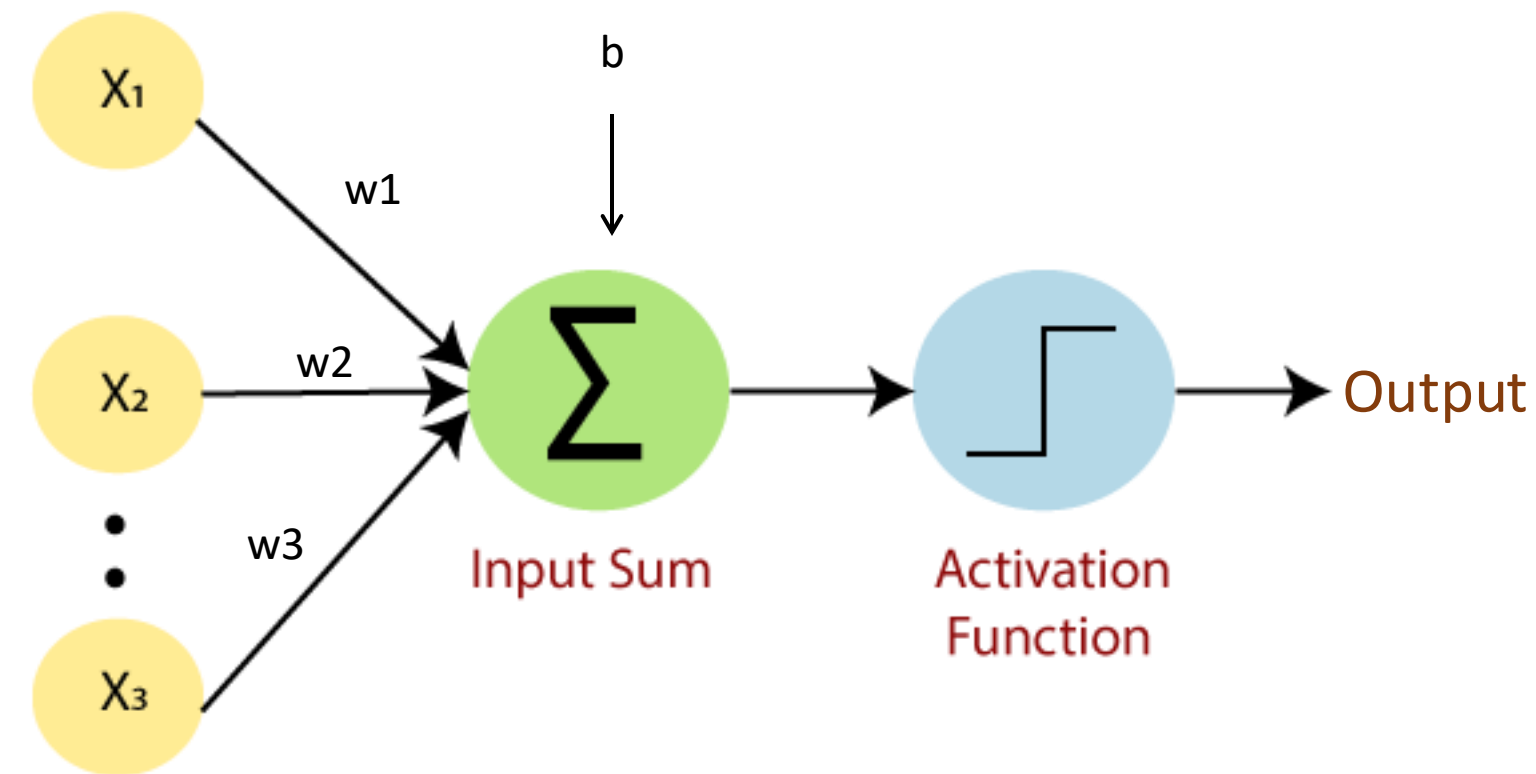
Output

Input Features

Vector of numeric values

Weighted Sum

Combines inputs with The associated weights (w1, w2, and w3) assigned to these inputs indicate their relevance.

Step Activation Function

Thresholds weighted sum

Binary Output

Classification result (0 or 1)

Activation function
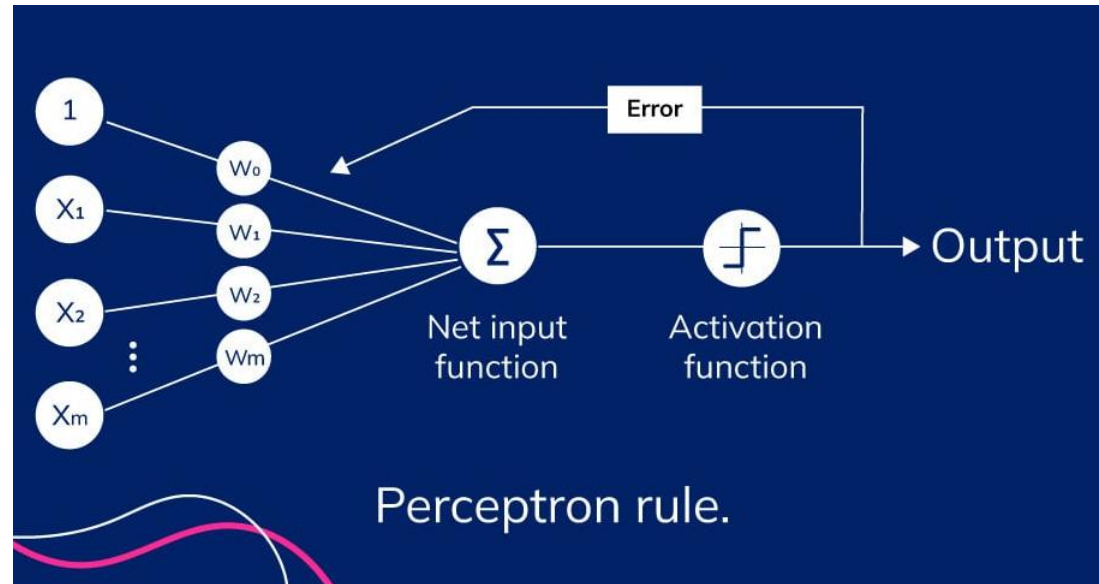
Perceptron rule.

- **Common choices:**
- (logistic) sigmoid (σ):

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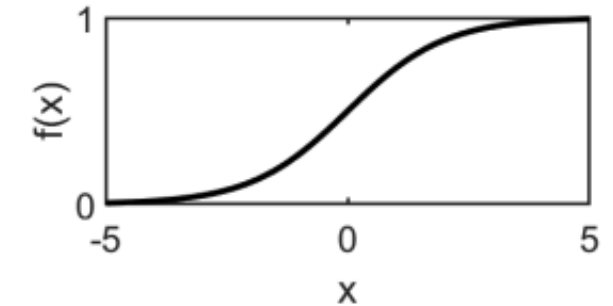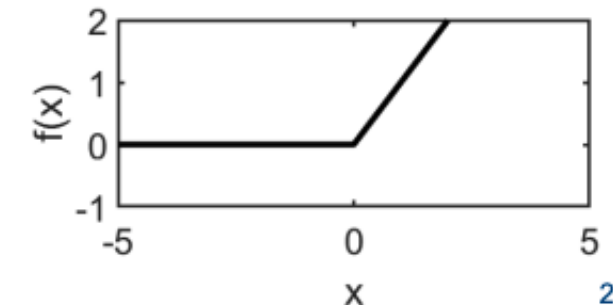

- hyperbolic tan (tanh):

$$f(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$
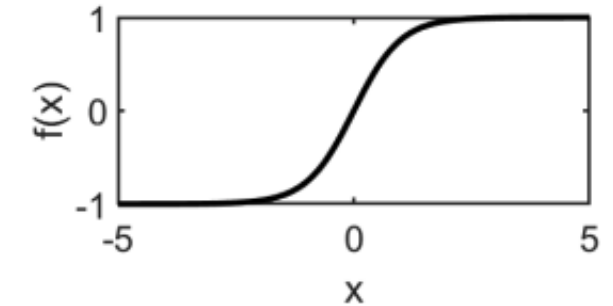
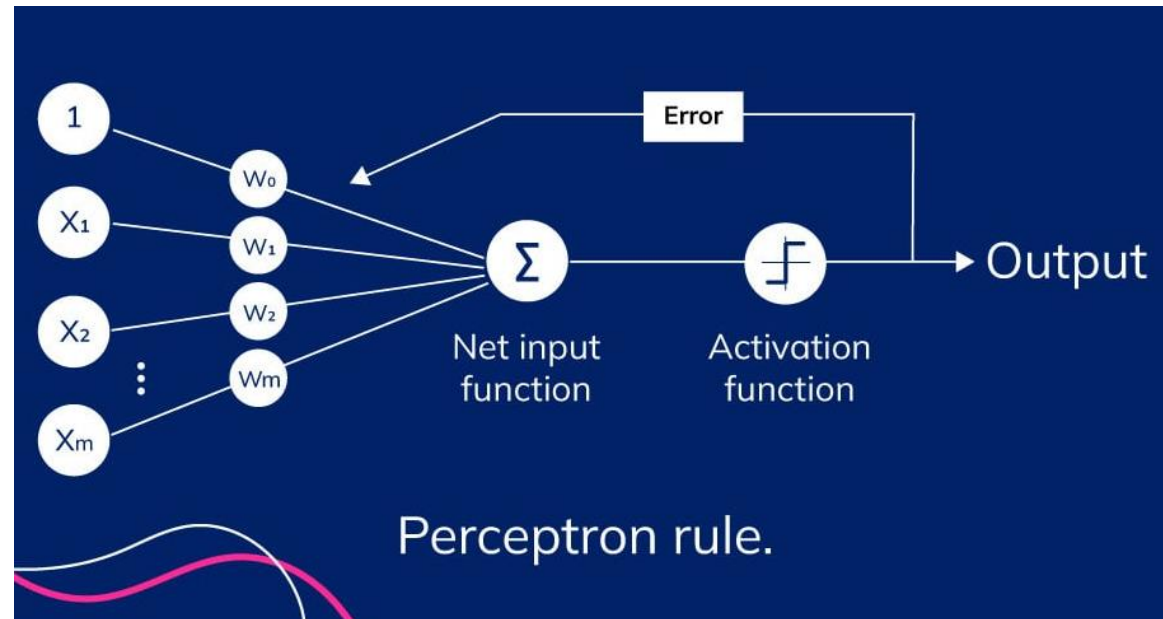- rectified linear unit (ReLU):

$$f(x) = \max(0, x)$$

https://www.lucentinnovation.com/blogs/technology-posts/understanding-the-perceptron

Perceptron Learning Rule: Weight Updates

Perceptron rule.

Initialize weight vector **w** to random values
for each training instance (**x$_i$**, y_i) do:

1   Calculate Error

Error = Desired Output – Actual Output $(y_i - \hat{y}_i)$.

2   Update Rule

$$w_j = w_j + \lambda(y_i - \hat{y}_i)x_{ij}$$

Learning rate

3   Weight Adjustment

New weights move decision boundary toward correct classification.

4   Convergence

Learning stops when all training samples are correctly classified.

Perceptron

Q5

Consider the following traning set:

(x_1,x_2)	y
(0,0)	0
(0,1)	1
(1,1)	1

Assume the initial weights are $w = \{w\_0, w\_1, w\_2\} = \{0.2, -0.4, 0.1\}$ and the activation function of the perceptron is the step function which outputs 1 if the weighted sum is > 0 and 0 otherwise.

1. Calculate the accuracy of the perceptron on the training data.
2. Using the perceptron learning rule and the learning rate of $\gamma = 0.2$. Train the perceptron for one epoch. What are the weights after the training?
3. What is the accuracy of the perceptron on the training data after training for one epoch? Did the accuracy improve?

Initial weights: $w = (w_0, w_1, w_2) = (0.2, -0.4, 0.1)$

Activation function: Step function

$$\text{Output} = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + w_2 x_2 > 0 \\ 0 & \text{otherwise} \end{cases}$$

Learning rate: $\gamma = 0.2$

Input (x_1, x_2)	Label y
(0, 0)	0
(0, 1)	1
(1, 1)	1

1: Initial Accuracy

Sample 1: (0, 0)

$z = 0.2 + (-0.4)(0) + 0.1(0) = 0.2 > 0 \Rightarrow \hat{y} = 1$ (Wrong, actual y = 0)

Input (x_1, x_2)	Label y
(0, 0)	0
(0, 1)	1
(1, 1)	1

Sample 2: (0, 1)

$z = 0.2 + (-0.4)(0) + 0.1(1) = 0.3 > 0 \Rightarrow \hat{y} = 1$ (Correct)

Sample 3: (1, 1)

$z = 0.2 + (-0.4)(1) + 0.1(1) = 0.2 - 0.4 + 0.1 = -0.1 \leq 0 \Rightarrow \hat{y} = 0$ (Wrong, actual y = 1)

Initial accuracy = 1 correct out of 3 = **33.3%**

2: One Epoch Training (Perceptron Rule)

Perceptron update rule (for binary labels 0/1):

$$w_j := w_j + \gamma(y - \hat{y})x_j$$

Where $x_0 = 1$ for the bias.

Go through each sample and update if prediction is incorrect:

Sample 1: (0, 0), y = 0, $\hat{y} = 1$

$$\Delta w = \gamma(y - \hat{y})x = 0.2(0 - 1) \cdot [1, 0, 0] = [-0.2, 0, 0]$$

New weights:

$$w = (0.2, -0.4, 0.1) + (-0.2, 0, 0) = (0.0, -0.4, 0.1)$$

Sample 2: (0, 1), y = 1

$$z = 0.0 + 0 + 0.1 = 0.1 > 0 \Rightarrow \hat{y} = 1 \quad \text{Correct} \rightarrow \text{no update}$$

Sample 3: (1, 1), y = 1

$$z = 0.0 + (-0.4)(1) + 0.1(1) = -0.3 \Rightarrow \hat{y} = 0 \quad \text{Wrong}$$

$$\Delta w = \gamma(1 - 0)[1, 1, 1] = 0.2[1, 1, 1] = [0.2, 0.2, 0.2]$$

New weights:

$$w = (0.0, -0.4, 0.1) + (0.2, 0.2, 0.2) = (0.2, -0.2, 0.3)$$

Final weights after 1 epoch:

$$w = (w_0, w_1, w_2) = (0.2, -0.2, 0.3)$$

3: Accuracy After One Epoch

Test with new weights:

Sample 1: (0, 0)

$$z = 0.2 + (-0.2)(0) + 0.3(0) = 0.2 \Rightarrow \hat{y} = 1 \quad \text{(Wrong)}$$

Sample 2: (0, 1)

$$z = 0.2 + 0 + 0.3 = 0.5 \Rightarrow \hat{y} = 1 \quad \text{(Correct)}$$

Sample 3: (1, 1)

$$z = 0.2 + (-0.2)(1) + 0.3(1) = 0.2 - 0.2 + 0.3 = 0.3 \Rightarrow \hat{y} = 1 \quad \text{(Correct)}$$

New accuracy = 2 correct out of 3 = **66.7%**

Perceptron Training Accuracy Factors

Data Linearity

The perceptron algorithm is guaranteed to converge for linearly-separable data, but the convergence point (class boundary) will depend on:
• The initial values of the weights and bias
• The learning rate

Not guaranteed to converge over non-linearly separable data

Perceptron vs. Logistic Regression

The perceptron and logistic regression are **closely related**, but they differ in:

- **Activation function**

- **Loss/objective function**
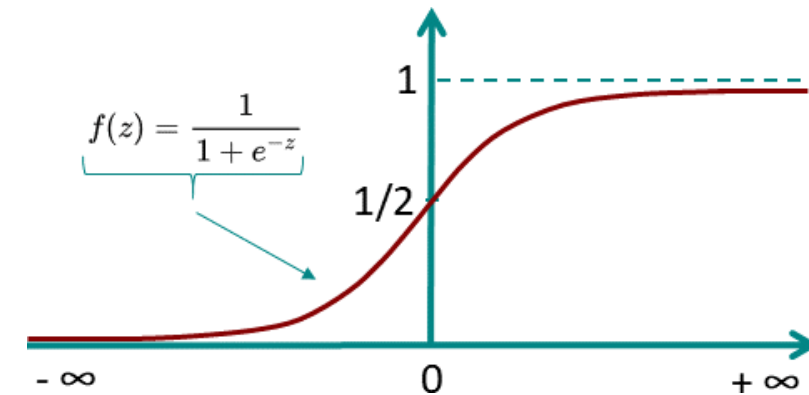
- **Learning mechanism**

Perceptron

- Binary step activation

- Error count

- Updates its weights after processing each instance, but only if the prediction is incorrect.

Logistic Regression

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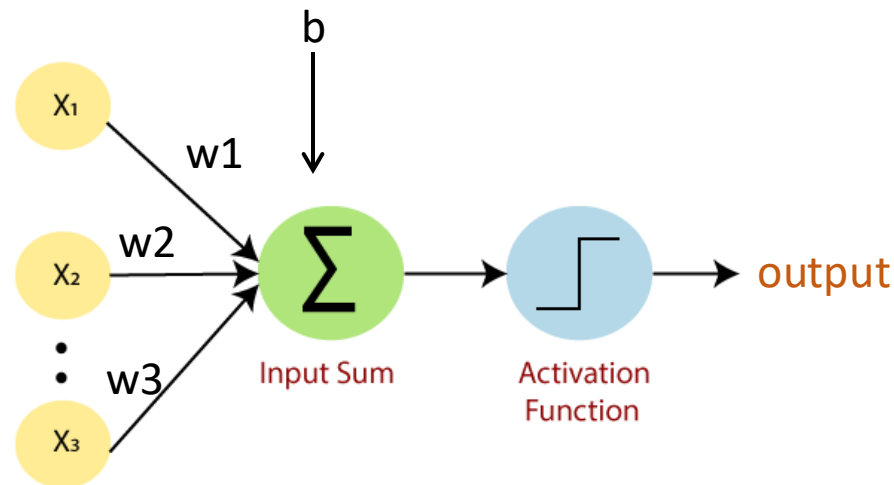
- Sigmoid activation

- Cross-entropy loss (negative log-likelihood)

- Weights are typically updated after all the training instances have been processed

The perceptron and logistic regression will only be completely equivalent if we change (1**) the objective function to the cross-entropy loss and (2) the activation function to the sigmoid.**

Multilayer Perceptron: Breaking Linearity Limits
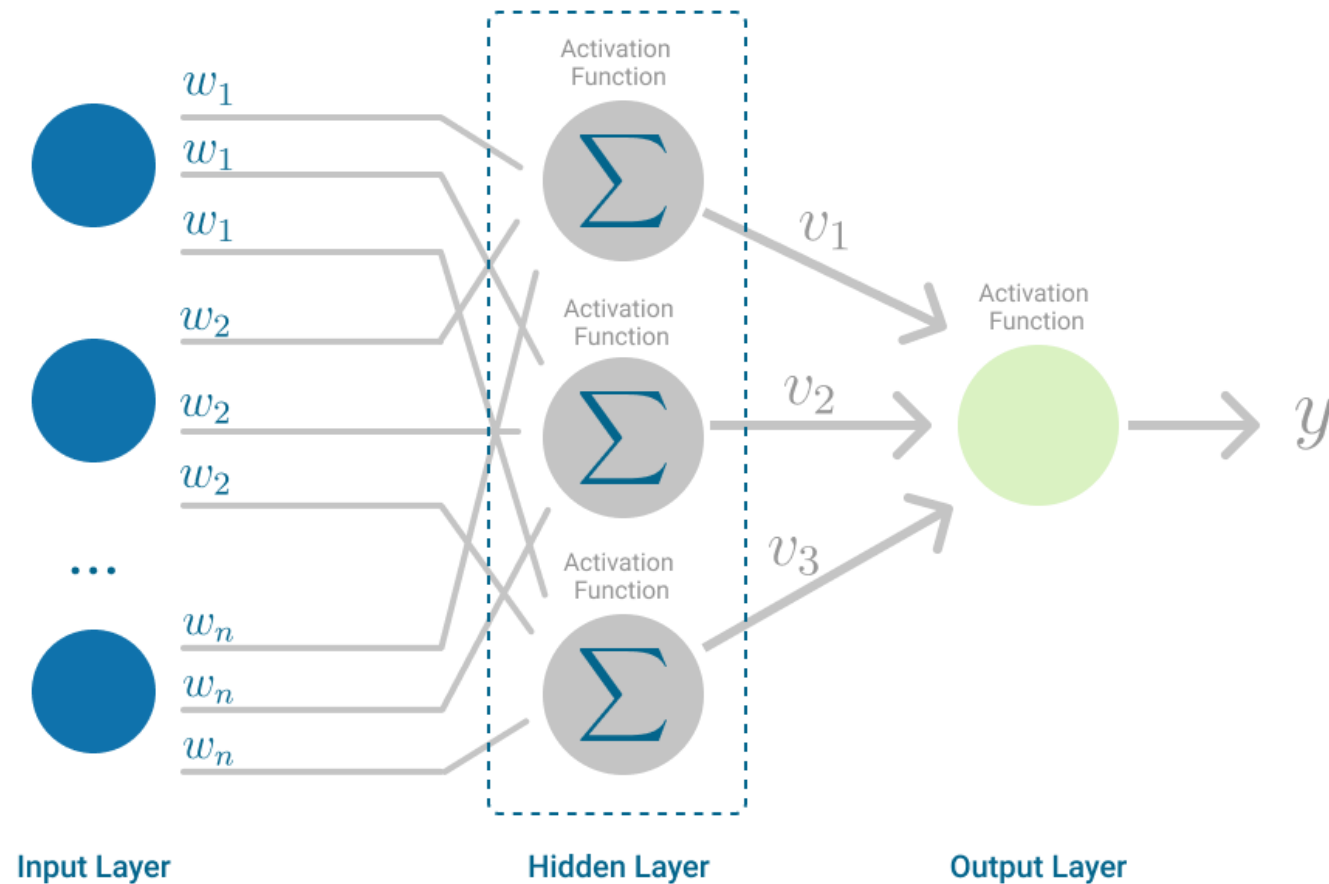
Input Layer

Receives raw feature values

Hidden Layer(s)

Learns complex feature representations. The activation functions between the hidden layers introduces non-linearity to the model

Output Layer

Produces final predictions