Machine Learning
W10 Tutorial

COMP30027 | Sandy Luo

Overview

Hidden Markov Models

Concept, code

Perceptron

General

01

Sequential Model

02

HMM

Hidden Markov Models (HMM)

Sequential Model

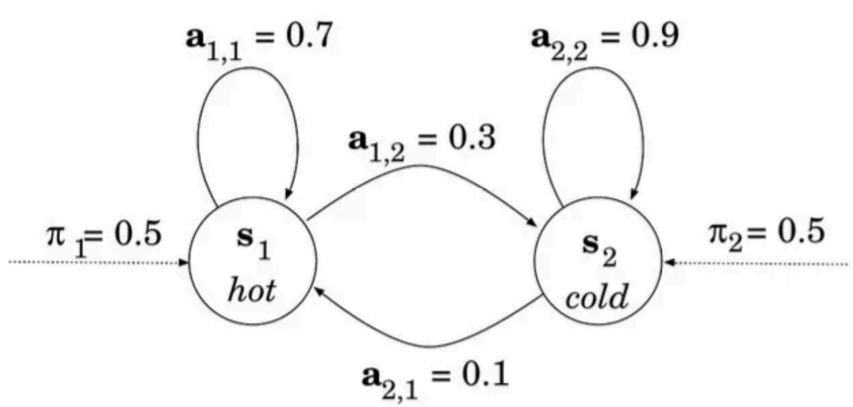
- Previously, treated data instances as independent
  - IRL instances are often related / structured
  - Can be sequential, e.g. time-related, speech etc.

- Sequential models:
  - Statistical models that aim to capture patterns & dependencies in sequential data
  - e.g. Markov models

Markov Chain

Models a system that transitions b/w states via probabilistic rules

1. **States:** $S = \{s_i\}$
2. **Initial state distribution:** $\Pi = \{\pi_i\}, \sum_i \pi_i = 1$
3. **Transition probability matrix:** $\mathrm{A} = \{a_{ij}\}, \sum_j a_{ij} = 1 \text{ for } \forall i$

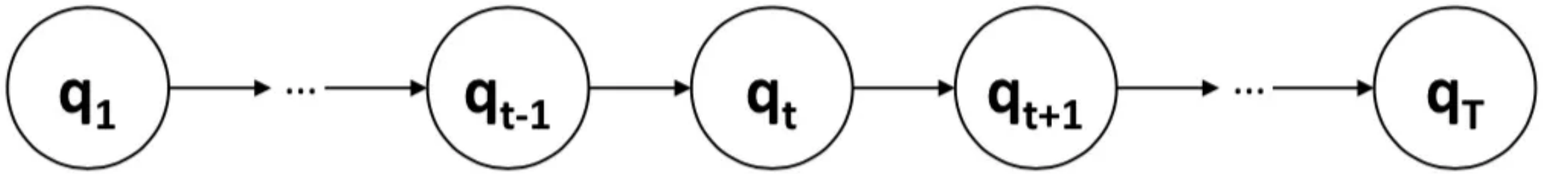


$$\mathrm{S} = \{\text{hot}, \text{cold}\}$$

$$\mathrm{A} = \begin{bmatrix} 0.7 & 0.3 \\ 0.1 & 0.9 \end{bmatrix} \qquad \pi = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

Markov Chain Assumption

- The current state, given all its preceding states, only depend on the immediate prior state



$$P(q_t|q_1, \ldots, q_{t-1}) = P(q_t|q_{t-1})$$

01

Sequential Model

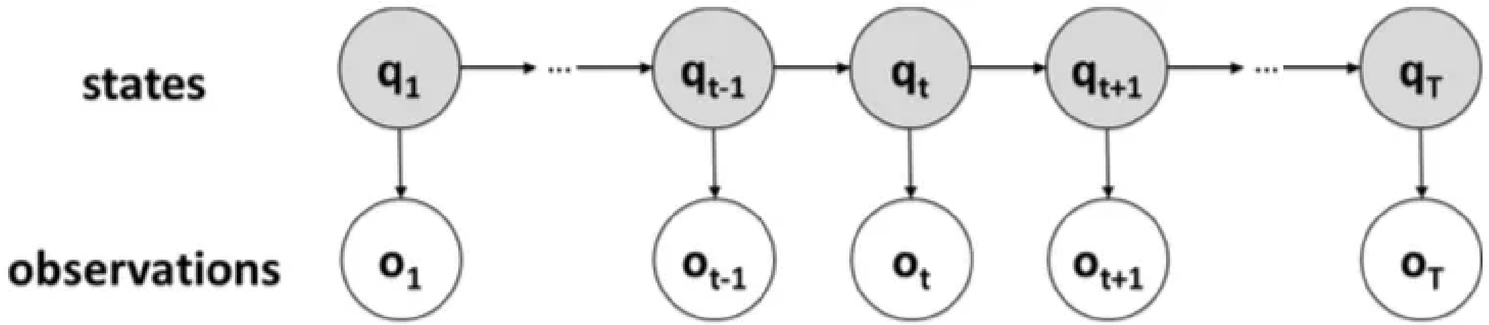
02

HMM

Hidden Markov Models (HMM)

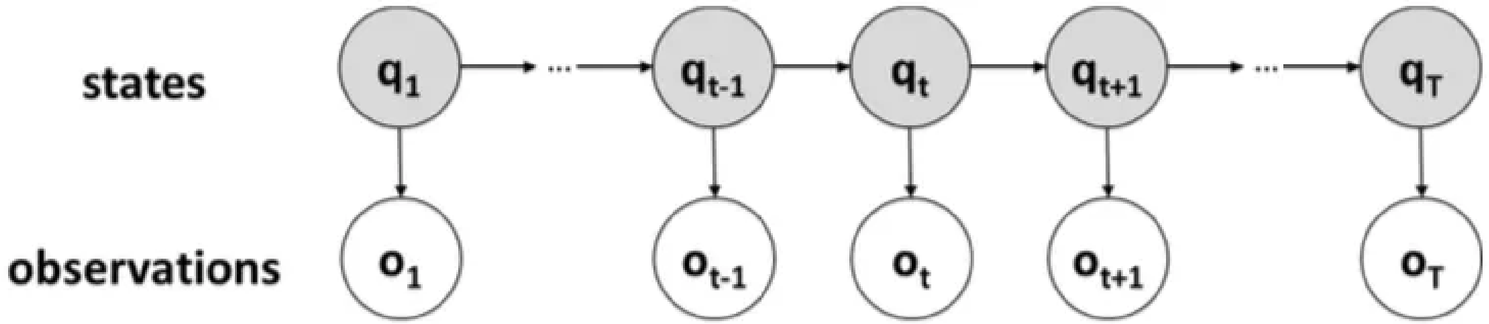
Hidden Markov Model

- Statistical model → models by following Markov chain assumptions
  - Except… **states are *hidden*** (unknown)
  - Have “observations” of the state that give some hints about the state



Assumptions

1. The *current* state is **only** dependent on the immediate *preceding* state
   - $P(q_t|q_1, ..., q_{t-1}, o_1, ..., o_{t-1}) = P(q_t|q_{t-1})$
2. The *current* observation is **only** dependent on the *current* (hidden) state
   - $P(o_t|q_1, ..., q_{t-1}, o_1, ..., o_{t-1}) = P(o_t|q_t)$



Q1:

Hidden Markov Models (HMMs) are best used when the observables are a **univariate time series**:

- Just observing a single variable, which changes over time due to some factor that can be estimated from previous observations.

Q1(a):

Hidden Markov Models (HMMs) are best used when the observables are a **univariate time series**:

- Just observing a single variable, which changes over time due to some factor that can be estimated from previous observations.

1. Explain the two main assumptions (Markov, output independence) that are built into an HMM.

Q1(a):

- Explain the two main assumptions (Markov, output independence) that are built into an HMM.

1. **Markov assumption:** The likelihood of transitioning into a given state depends only on the current state, and not the previous states (or outputs)
    - $P(q_t|q_1, ..., q_{t-1}, o_1, ..., o_{t-1}) = P(q_t|q_{t-1})$
2. **Output independence assumption:** The likelihood of a state producing an observation (as output) does not depend on any other states or outputs
    - $P(o_t|q_1, ..., q_{t-1}, o_1, ..., o_{t-1}) = P(o_t|q_t)$

Q1(b):

Hidden Markov Models (HMMs) are best used when the observables are a **univariate time series**:

- Just observing a single variable, which changes over time due to some factor that can be estimated from previous observations.

2. Could we construct the HMM in such a way as to relax these assumptions? What would the model look like, and what is the major downside?

Q1(b):

2. Could we construct the HMM in such a way as to relax these assumptions? What would the model look like, and what is the major downside?

- **Can introduce more information (state / observations)**
  - **e.g. pairs of states in the conditions for transition probability matrix A**
  - **e.g. pairs of states in the conditions for output probability matrix B**

- **Will vastly increase the number of parameters in the model**

Q1(c):

Hidden Markov Models (HMMs) are best used when the observables are a **univariate time series**:

- Just observing a single variable, which changes over time due to some factor that can be estimated from previous observations.

3. Could we build an HMM for a *multivariate* time series, where we have a number of observed variables for a given (hidden) state?

Q1(c):

3. Could we build an HMM for a *multivariate* time series, where we have a number of observed variables for a given (hidden) state?

- **Yes!**
  - **Observations → vector of measurements, rather than a single variable.**
  - **Vastly increasing the number of parameters**

- **However, sometimes coupling the outputs like this is unnecessary; it might be possible to just generate independent HMMs for each output.**

Main Tasks

1. **Evaluation:**
   - Estimate the likelihood of an ***observation*** sequence
   - Given an HMM μ and observation sequence Ω, determine the likelihood $P(\Omega|\mu)$

   - **Forward algorithm**

2. **Decoding:**
   - Find the most probable ***state*** sequence
   - Given an HMM μ and observation sequence Ω, determine the most probable hidden state sequence Q

   - **Viterbi algorithm**

3. **Learning:**
   - Estimate parameters of HMM
   - Given observation sequence Ω, the set of possible states S and observations O in an HMM, learn parameters $\mu = (A, B, \Pi)$

Evaluation - Forward Algorithm

- Aim: Find the likelihood of an **observation** sequence
- Forward probability:
  - The probability of the partial observation sequence, (o_1,o_2,...,o_t) and state s_i at time t, given the model.
  - Calculate forward probabilities and reuse them to calculate t+1

$$\alpha_t(i) = P(o_1, o_2, ..., o_t, q_t = s_i|\mu)$$

1. **Initialisation:**
   - $t = 1$, state $i = 1, \ldots, N$
   - $\alpha_1(i) = \pi_i b_i(o_1)$
2. **Induction:**
   - $t = 1, \ldots, T-1$, state $i = 1, \ldots, N$
   - $\alpha_{t+1}(i) = (\sum_{j=1}^{N} \alpha_t(j) a_{ji}) b_i(o_{t+1})$
3. **Termination:**
   - $P(\Omega|\mu) = \sum_{i=1}^{N} \alpha_T(i)$

Decoding - Viterbi Algorithm

- Aim: Find the most probable (hidden) **state** sequence - maximum probability for a partial sequence (q_1,q_2,...,q_{t-1}, q_t=s_i) along a single path
  - Backpointer: the hidden state at t-1 = $\psi_t(i)$

1. **Initialisation:**
   - $t = 1$, state $i = 1, \ldots, N$

     $\delta_1(i) = \pi_i b_i(o_1)$

     $\psi_1(i) = 0$
2. **Induction:**
   - $t = 1, \ldots, T-1$, state $i = 1, \ldots, N$

     $\delta_{t+1}(i) = \max_{1 \le j \le N} (\delta_t(j) a_{ji})\, b_i(o_{t+1})$

     $\psi_{t+1}(i) = \arg\max_{1 \le j \le N} (\delta_t(j) a_{ji})$
3. **Termination:**

   $$P_{best} = \max_{1 \le i \le N} \delta_T(i)$$

   $$q_T^* = \arg\max_{1 \le i \le N} \delta_T(i)$$
4. **Backtrack:**

   $$q_t^* = \psi_{t+1}(q_{t+1}^*) \text{ for } t = T-1, T-2, \ldots, 1$$

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

Q2(a):

- How might we go about obtaining the values in the matrices \Pi, A, and B given above, in a supervised context?

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

Q2(a):

- A: Transition Probability Matrix
  - Each element a_{ij} = count of number of times the **state sequence** (i,j) was observed in the labelled data, out of the number of times the state i was observed.
    - i.e. probability of state transition for (i,j) from state (i)
  - In this case, this is a pair of part–of–speech tags (adj, verb, or noun).

Q2(a):

- B: Output Probability Matrix
  - Each element b_{ik} = count of how many times the **observation** k was observed labelled with state i in the training data, out of the number of times the state i was observed.
    - i.e. probability of state i producing the observation k
  - In this case, this is a word being labelled as the equivalent part–of–speech.

Q2(a):

- Π: Initial state distribution
  - Each element π_i = count of how many times state i was the start of an observation sequence, out of the number of observation sequences.
    - i.e. literally the distribution of starting states
  - In this case, this is where some part–of–speech starts a sentence.

Q2(b):

- Use the **forward** algorithm to find the probability of the "sentence" `*brown leaves turn*`

1. **Initialisation:**
   - $t = 1$, state $i = 1, \ldots, N$
   - $\alpha_1(i) = \pi_i b_i(o_1)$
2. **Induction:**
   - $t = 1, \ldots, T-1$, state $i = 1, \ldots, N$
   - $\alpha_{t+1}(i) = (\sum_{j=1}^{N} \alpha_t(j) a_{ji}) b_i(o_{t+1})$
3. **Termination:**
   - $P(\Omega|\mu) = \sum_{i=1}^{N} \alpha_T(i)$

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

Q2(b): “brown”

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. **Initialisation:**
   - $t = 1$, state $i = 1, \ldots, N$
   - $\alpha_1(i) = \pi_i b_i(o_1)$

		1:brown	2:leaves	3:turn
J:	J	$\pi[J]B[J,brown]$ $0.3(0.8) = 0.24$		
N:	N	$\pi[N]B[B,brown]$ $0.4(0.3) = 0.12$		
V:	V	$\pi[V]B[V,brown]$ $0.3(0.1) = 0.03$		

Q2(b): “leaves”

2. **Induction:**
   - $t = 1, \ldots, T-1$, state $i = 1, \ldots, N$
   - $\alpha_{t+1}(i) = (\sum_{j=1}^{N} \alpha_t(j) a_{ji}) b_i(o_{t+1})$

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:brown		2:leaves	3:turn
J:	0.24	J -> J: $\alpha_1(J) = 0.24$ N -> J: $\alpha_1(N) = 0.12$ V -> J: $\alpha_1(V) = 0.03$	$\alpha_1(J) \times A[J,J] = 0.24(0.4) = 0.096$ $\alpha_1(N) \times A[N,J] = 0.12(0.1) = 0.012$ $\alpha_1(V) \times A[V,J] = 0.03(0.4) = 0.012$ $(0.096 + 0.012 + 0.012) \times 0.1 = 0.012$	
N:	0.12	J -> N: $\alpha_1(J) = 0.24$ N -> N: $\alpha_1(N) = 0.12$ V -> N: $\alpha_1(V) = 0.03$	$\alpha_1(J) \times A[J,N] = 0.24(0.5) = 0.12$ $\alpha_1(N) \times A[N,N] = 0.12(0.4) = 0.048$ $\alpha_1(V) \times A[V,N] = 0.03(0.5) = 0.015$ $(0.12 + 0.048 + 0.015) \times 0.4 = 0.0732$	
V:	0.03	J -> V: $\alpha_1(J) = 0.24$ N -> V: $\alpha_1(N) = 0.12$ V -> V: $\alpha_1(V) = 0.03$	$\alpha_1(J) \times A[J,V] = 0.24(0.1) = 0.024$ $\alpha_1(N) \times A[N,V] = 0.12(0.5) = 0.06$ $\alpha_1(V) \times A[V,V] = 0.03(0.1) = 0.003$ $(0.024 + 0.06 + 0.003) \times 0.3 = 0.0261$	

Q2(b): “turn”

2. **Induction:**
   - $t = 1, \ldots, T-1$, state $i = 1, \ldots, N$
   - $\alpha_{t+1}(i) = (\sum_{j=1}^{N} \alpha_t(j) a_{ji}) b_i(o_{t+1})$

	1:brown	2:leaves		3:turn
J:	0.24	0.012	J -> J: $\alpha_2(J) = 0.012$ N -> J: $\alpha_2(N) = 0.0732$ V -> J: $\alpha_2(V) = 0.0261$	$\alpha_2(J) \times A[J,J] = 0.012(0.4) = 0.0048$ $\alpha_2(N) \times A[N,J] = 0.0732(0.1) = 0.00732$ $\alpha_2(V) \times A[V,J] = 0.0261(0.4) = 0.01044$ $(0.0048 + 0.00732 + 0.01044) \times 0.1 = 0.002256$
N:	0.12	0.0732	J -> N: $\alpha_2(J) = 0.012$ N -> N: $\alpha_2(N) = 0.0732$ V -> N: $\alpha_2(V) = 0.0261$	$\alpha_2(J) \times A[J,N] = 0.012(0.5) = 0.006$ $\alpha_2(N) \times A[N,N] = 0.0732(0.4) = 0.02928$ $\alpha_2(V) \times A[V,N] = 0.0261(0.5) = 0.01305$ $(0.006 + 0.02928 + 0.01305) \times 0.3 = 0.014499$
V:	0.03	0.0261	J -> V: $\alpha_2(J) = 0.012$ N -> V: $\alpha_2(N) = 0.0732$ V -> V: $\alpha_2(V) = 0.0261$	$\alpha_2(J) \times A[J,V] = 0.012(0.1) = 0.0012$ $\alpha_2(N) \times A[N,V] = 0.0732(0.5) = 0.0366$ $\alpha_2(V) \times A[V,V] = 0.0261(0.1) = 0.00261$ $(0.0012 + 0.0366 + 0.00261) \times 0.6 = 0.024246$

3. **Termination:**
   - $P(\Omega|\mu) = \sum_{i=1}^{N} \alpha_T(i)$
   - Sum last column → 0.002256 + 0.014499 + 0.024246 = 0.041001

Q2(c):

- Use the **Viterbi** algorithm to find the most likely state sequence for the sentence `*brown leaves turn*`

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

Q2(c): “brown”

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. **Initialisation:**
   - $t = 1$, state $i = 1, \ldots, N$

$$\delta_1(i) = \pi_i b_i(o_1)$$

$$\psi_1(i) = 0$$

		1:brown	2:leaves	3:turn
J:	J	$\pi[J]B[J,brown]$ $0.3(0.8) = 0.24$		
N:	N	$\pi[N]B[B,brown]$ $0.4(0.3) = 0.12$		
V:	V	$\pi[V]B[V,brown]$ $0.3(0.1) = 0.03$		

Q2(c): “leaves”

2. **Induction:**
   - $t = 1, \ldots, T-1$, state $i = 1, \ldots, N$

$$\delta_{t+1}(i) = \max_{1 \le j \le N} (\delta_t(j) a_{ji})\, b_i(o_{t+1})$$

$$\psi_{t+1}(i) = \arg\max_{1 \le j \le N} (\delta_t(j) a_{ji})$$

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:brown		2:leaves	3:turn
J:	0.24	J -> J: $\alpha_1(J) = 0.24$ N -> J: $\alpha_1(N) = 0.12$ V -> J: $\alpha_1(V) = 0.03$	$\alpha_1(J) A[J,J] B[J,leaves]$ = 0.24 x 0.4 x 0.1 = **0.0096** $\alpha_1(N) A[N,J] B[J,leaves]$ = 0.12 x 0.1 x 0.1 = 0.0012 $\alpha_1(V) A[V,J] B[J,leaves]$ = 0.03 x 0.4 x 0.1 = 0.0012	
N:	0.12	J -> N: $\alpha_1(J) = 0.24$ N -> N: $\alpha_1(N) = 0.12$ V -> N: $\alpha_1(V) = 0.03$	$\alpha_1(J) A[J,N] B[N,leaves]$ = 0.24 x 0.5 x 0.4 = **0.048** $\alpha_1(N) A[N,N] B[N,leaves]$ = 0.12 x 0.4 x 0.4 = 0.0192 $\alpha_1(V) A[V,N] B[N,leaves]$ = 0.03 x 0.5 x 0.4 = 0.006	
V:	0.03	J -> V: $\alpha_1(J) = 0.24$ N -> V: $\alpha_1(N) = 0.12$ V -> V: $\alpha_1(V) = 0.03$	$\alpha_1(J) A[J,V] B[V,leaves]$ = 0.24 x 0.1 x 0.3 = 0.0072 $\alpha_1(N) A[N,V] B[V,leaves]$ = 0.12 x 0.5 x 0.3 = **0.018** $\alpha_1(V) A[V,V] B[V,leaves]$ = 0.03 x 0.1 x 0.3 = 0.0009	

Q2(c): “turn”

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

2. **Induction:**

- $t = 1, \ldots, T-1$, state $i = 1, \ldots, N$

$$\delta_{t+1}(i) = \max_{1 \le j \le N} (\delta_t(j) a_{ji})\, b_i(o_{t+1})$$

$$\psi_{t+1}(i) = \arg\max_{1 \le j \le N} (\delta_t(j) a_{ji})$$

	1:brown	2:leaves		3:turn
J:	0.24	0.0096 J -> J	J -> J: $\alpha_2(J) = 0.0096$ N -> J: $\alpha_2(N) = 0.048$ V -> J: $\alpha_2(V) = 0.018$	$\alpha_2(J) A[J,J] B[J,turn]$ = 0.0096 x 0.4 x 0.1 = 0.000384 $\alpha_2(N) A[N,J] B[J,turn]$ = 0.048 x 0.1 x 0.1 = 0.00048 $\alpha_2(V) A[V,J] B[J,turn]$ = 0.018 x 0.4 x 0.1 = **0.00072**
N:	0.12	0.048 J -> N	J -> N: $\alpha_2(J) = 0.0096$ N -> N: $\alpha_2(N) = 0.048$ V -> N: $\alpha_2(V) = 0.018$	$\alpha_2(J) A[J,N] B[N,turn]$ = 0.0096 x 0.5 x 0.3 = 0.00144 $\alpha_2(N) A[N,N] B[N,turn]$ = 0.048 x 0.4 x 0.3 = **0.00576** $\alpha_2(V) A[V,N] B[N,turn]$ = 0.018 x 0.5 x 0.3 = 0.0027
V:	0.03	0.018 J -> V	J -> V: $\alpha_2(J) = 0.0096$ N -> V: $\alpha_2(N) = 0.048$ V -> V: $\alpha_2(V) = 0.018$	$\alpha_2(J) A[J,V] B[V,turn]$ = 0.0096 x 0.1 x 0.6 = 0.000576 $\alpha_2(N) A[N,V] B[V,turn]$ = 0.048 x 0.5 x 0.6 = **0.0144** $\alpha_2(V) A[V,V] B[V,turn]$ = 0.018 x 0.1 x 0.6 = 0.00108

- Backtrack to find most likely state sequence

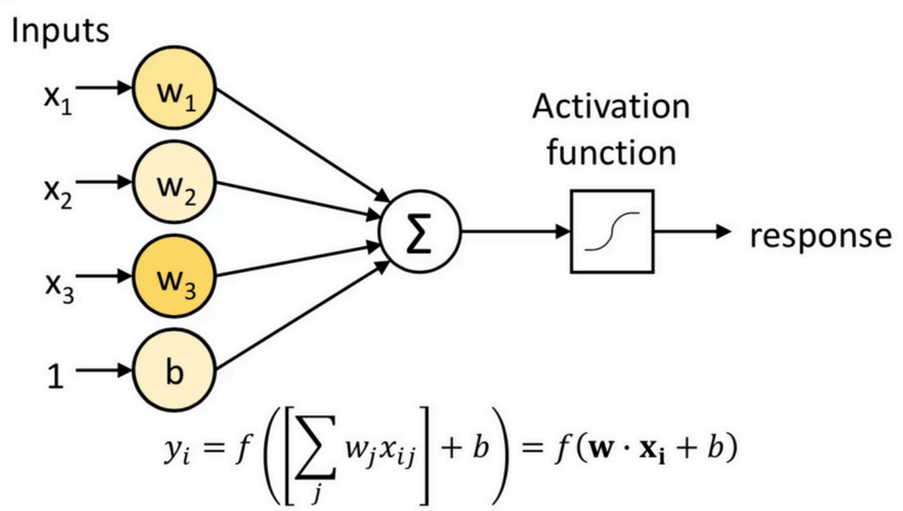
3. **Termination:**

$$P_{best} = \max_{1 \le i \le N} \delta_T(i)$$

$$q_T^* = \arg\max_{1 \le i \le N} \delta_T(i)$$

Perceptron

Perceptron



- Neural Network with a single neutron
  a. Input a vector of features
  b. Each feature has an associated weight
  c. Computes a weighted sum of the inputs
  d. Apply an activation function to the sum to produce output

- Perceptrons are binary linear classifiers:
  - Generally uses step function as activation function

$$f(\mathbf{w} \cdot \mathbf{x_i} + b) = \begin{cases} 1 & if\ \mathbf{w} \cdot \mathbf{x_i} + b \geq 0 \\ 0 & otherwise \end{cases}$$

Q5(a):

- Assume the initial weights are w = {w_0,w_1,w_2} = {0.2, -0.4, 0.1}
- Activation function: step function
  - Outputs 1 if the weighted sum is > 0 and 0 otherwise.

- **Calculate the accuracy of the perceptron on the training data.**

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

Q5(a):

- w = {w_0,w_1,w_2} = {0.2, -0.4, 0.1}

$$f(\mathbf{w}\cdot\mathbf{x_i}+b)=\begin{cases}1 & if\ \mathbf{w}\cdot\mathbf{x_i}+b\geq 0\\ 0 & otherwise\end{cases}$$

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

- **Calculate weighted sum based on input feature values and weight**

(x_1,x_2)	$w_0 + w_1x_1 + w_2x_2$	predicted y
(0,0)	0.2 - 0.4(0) + 0.1(0) = 0.2	1
(0,1)	0.2 - 0.4(0) + 0.1(1) = 0.3	1
(1,1)	0.2 - 0.4(1) + 0.1(1) = -0.1	0

- **Only instance 2 is correct → accuracy = 1/3**

Q5(b):

- Assume the initial weights are w = {w_0,w_1,w_2} = {0.2, -0.4, 0.1}
- Activation function: step function
  - Outputs 1 if the weighted sum is > 0 and 0 otherwise.

- **Using the perceptron learning rule & learning rate of gamma = 0.2, train the perceptron for one epoch. What are the weights after the training?**

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

$$w_j \leftarrow w_j + \lambda(y^i - \hat{y}^i)x_j^i$$

Q5(a):

- **w = {w_0,w_1,w_2} = {0.2, -0.4, 0.1}**
- **gamma = 0.2**

$$w_j \leftarrow w_j + \lambda(y^i - \hat{y}^i)x_j^i$$

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

(x_1,x_2)	$w_0 + w_1x_1 + w_2x_2$	$\hat{y}$	y
(0,0)	0.2 - 0.4(0) + 0.1(0) = 0.2	1	0
	Update w:		
	$w_0 = 0.2 + 0.2(0-1)1 = 0$		
	$w_1 = -0.4 + 0.2(0-1)1 = -0.4$		
	$w_2 = 0.2 + 0.2(0-1)1 = 0.1$		
(0,1)	0 - 0.4(0) + 0.1(1) = 0.1	1	1
	Correct prediction, no update		
(1,1)	0 - 0.4(1) + 0.1(1) = -0.3	0	1
	Update w:		
	$w_0 = 0 + 0.2(1-0)1 = 0.2$		
	$w_1 = -0.4 + 0.2(1-0)1 = -0.2$		
	$w_2 = 0.2 + 0.2(1-0)1 = 0.3$		

Q5(c):

- Assume the initial weights are w = {w_0,w_1,w_2} = {0.2, -0.4, 0.1}
- Activation function: step function
  - Outputs 1 if the weighted sum is > 0 and 0 otherwise.

- **What is the accuracy of the perceptron on the training data after training for one epoch? Did the accuracy improve?**

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

	Update w:		
	$w_0 = 0 + 0.2(1-0)1 = 0.2$		
	$w_1 = -0.4 + 0.2(1-0)1 = -0.2$		
	$w_2 = 0.2 + 0.2(1-0)1 = 0.3$		

Q5(c):

- w_1 = {w_0,w_1,w_2} = {0.2, -0.2, 0.3}

$$f(\mathbf{w}\cdot\mathbf{x_i}+b)=\begin{cases}1 & if\ \mathbf{w}\cdot\mathbf{x_i}+b\geq 0\\ 0 & otherwise\end{cases}$$

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

- Calculate weighted sum based on input feature values and weight

(x_1,x_2)	$w_0 + w_1x_1 + w_2x_2$	predicted y
(0,0)	0.2 - 0.2(0) + 0.3(0) = 0.2	1
(0,1)	0.2 - 0.2(0) + 0.3(1) = 0.5	1
(1,1)	0.2 - 0.2(1) + 0.3(1) = 0.1	1

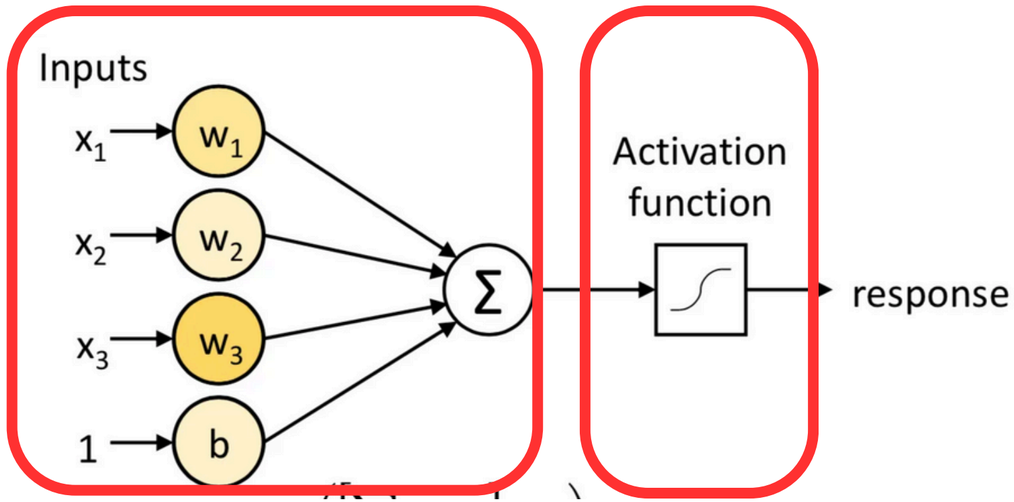
- Now instances 2 and 3 are correct → accuracy = 2/3

Q6:

The perceptron is closely related to logistic regression. Explain the condition(s) under which both are equivalent.

Q6:

The perceptron is closely related to logistic regression. Explain the condition(s) under which both are equivalent.



linear combination of input features...

same as for logistic regression

Standard perceptron uses step function,

but if we use a sigmoid activation function

(1/(1+e^{-x})) instead on the weighted sum

→ equations similar

Q6:

The perceptron is closely related to logistic regression. Explain the condition(s) under which both are equivalent.

Objective function (loss) for optimising weights:

- Logistic regression: Cross-entropy loss (negative log-likelihood)
- Perceptron: Counting errors

→ Change perceptron’s objective function to cross-entropy loss

Q6:

The perceptron is closely related to logistic regression. Explain the condition(s) under which both are equivalent.

Also differences in weight updates (doesn’t affect results)

- Logistic regression typically updates weights after iterating over all training instances once
- Perceptron typically updates weights per instance / in batches