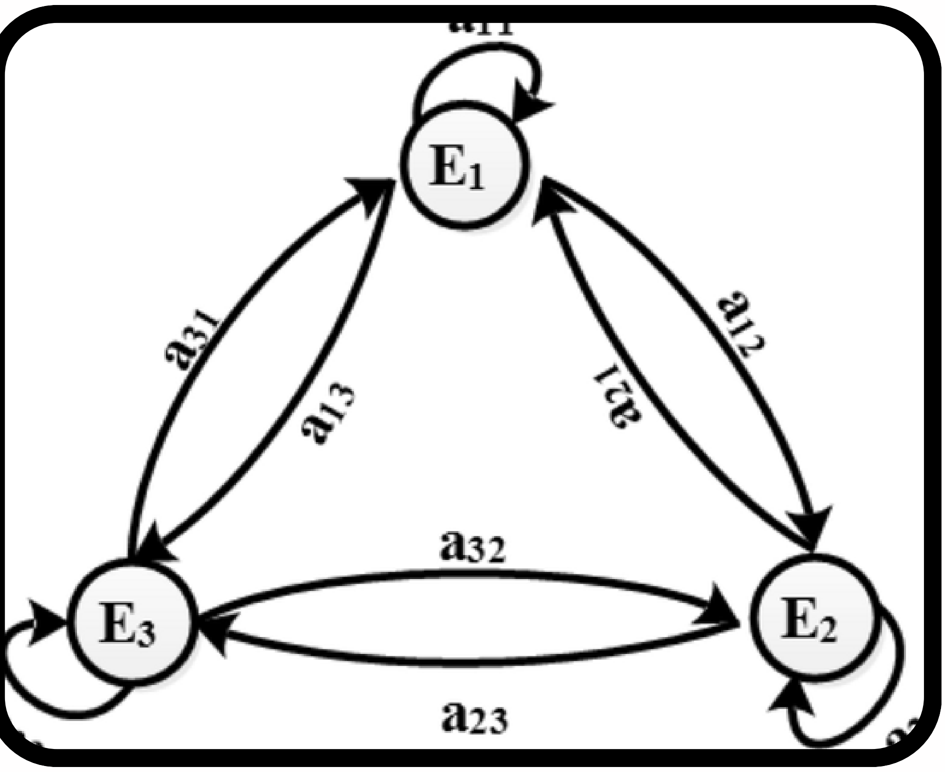
Stochastic Team

Mahmoud Essam(Leader) - Abdullah Hussien - Muhammed Ahmed Hamdi- Muhammed Hassan gaber- Fares Muhammed Fathy - Safy Fathy - Abdelrhman Ashraf



Hidden Markov Chain Model
the main 3 problems of HMM

Introduction to Hidden Markov Chain

Definition 📚

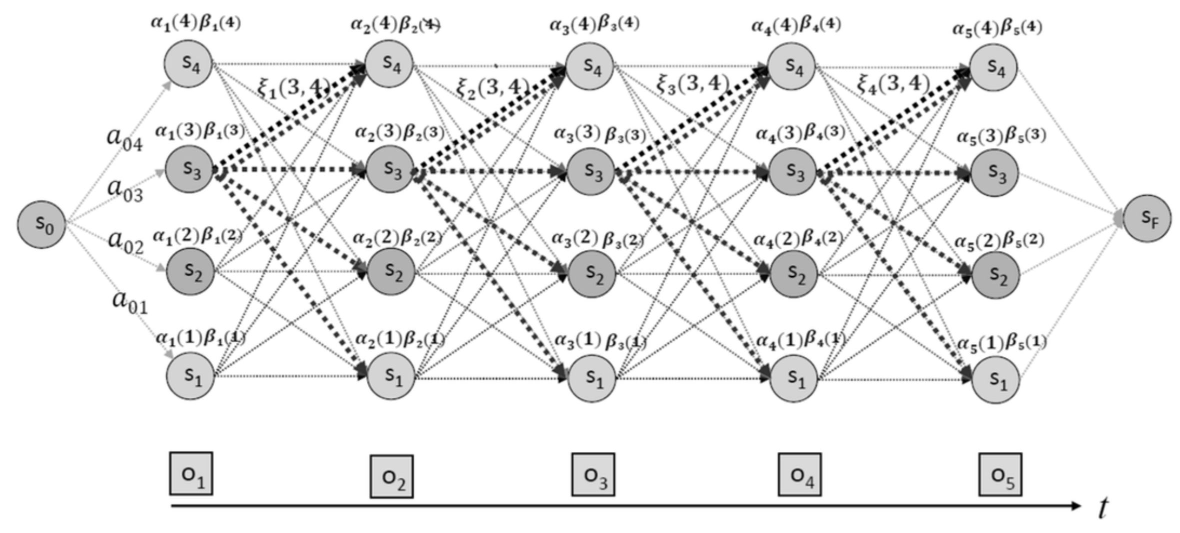
A probabilistic model that describes a system with unobservable states and observable outputs.

Applications 🌐

Speech recognition, natural language processing, and information retrieval.

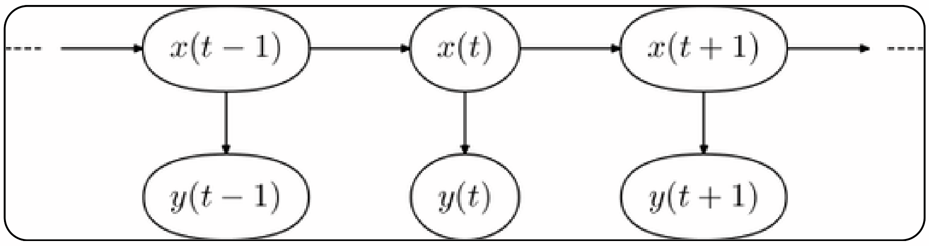
Notable Use Case 🎲

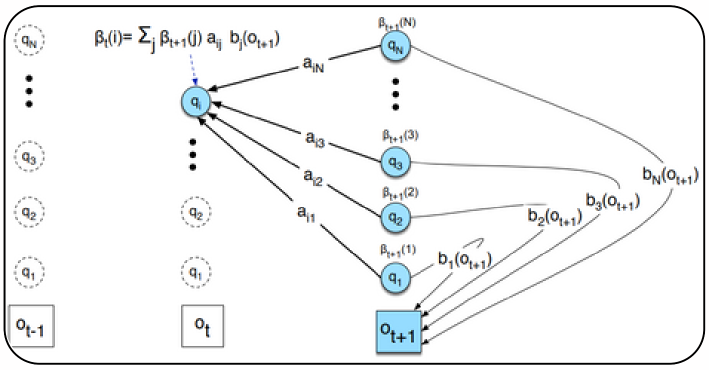
Google Translate's Language Models for word suggestions and corrections.

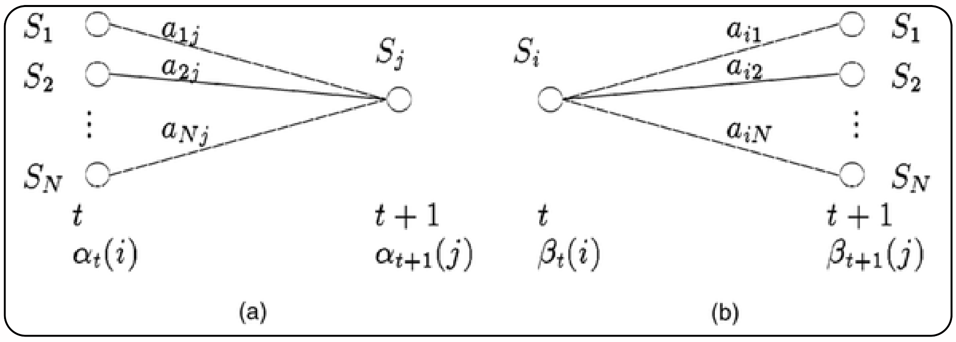


What do you want to start from?

- First Problem: using Forward Algorithm
- Add Backward Algorithm
- Second Problem: Using Viterabi Algorithm
- Third Problem: Using Baum-Welch Algorithm







Forward Algorithm

The Forward Algorithm is a dynamic programming algorithm used to efficiently compute the probability of a sequence of observations in a Hidden Markov Model. It does this by summing over all possible paths through the model that could have generated the observed sequence. The Forward Algorithm is a fundamental part of many other algorithms in HMMs, such as the Viterbi Algorithm and the Baum-Welch Algorithm.

Initialization

$$\alpha_1(i) = p_i \ b_i(o(1)), \quad i=1,\ldots,N$$

Recursion

$$\alpha_{t+1}(i) = [\textstyle\sum_{j-1}^{N} \alpha_t(j) \ a_{ji}]\, b_i(o(t+1))$$

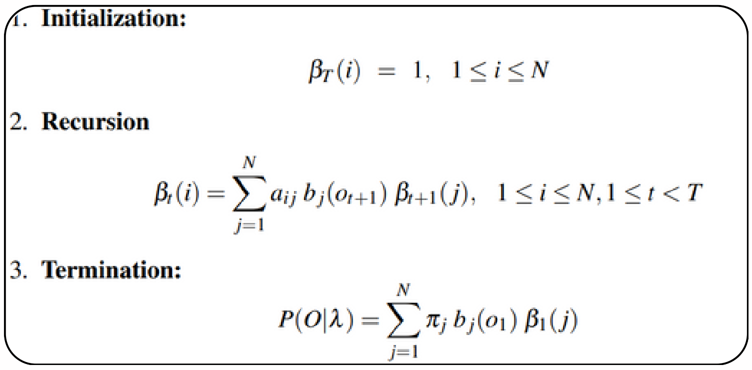
where $i = 1,\ldots,N,\ t=1,\ldots,T-1$

Termination

$$P(o(1)o(2)\ldots o(T)) = \textstyle\sum_{j-1}^{N} \alpha_T(j)$$

Backward Algorithm

The Backward Algorithm is another dynamic programming algorithm used in Hidden Markov Models (HMMs). It calculates the probability of observing a partial or complete sequence of future observations, given a particular state at a specific time. The Backward Algorithm starts from the last observation and works its way backward through time, calculating the probability of being in each state at each time step. This algorithm is useful in various applications, such as speech recognition, natural language processing, and bioinformatics.



1. **Initialization:**

$$\beta_T(i) \;=\; 1, \quad 1 \le i \le N$$

2. **Recursion**

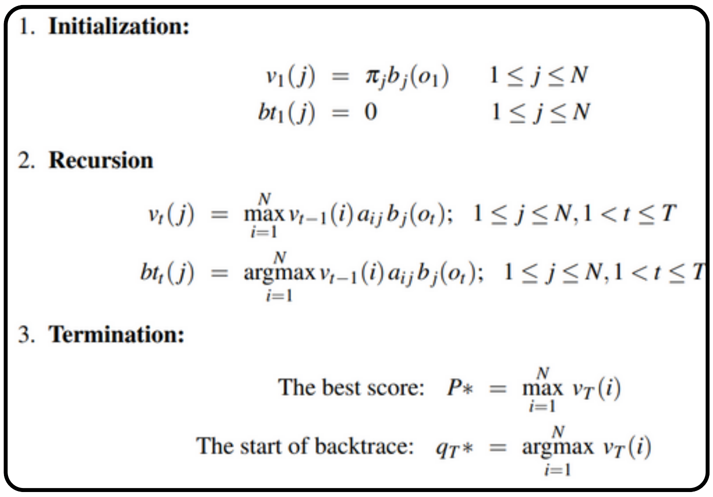
$$\beta_t(i) = \sum_{j=1}^{N} a_{ij}\, b_j(o_{t+1})\, \beta_{t+1}(j), \quad 1 \le i \le N, 1 \le t < T$$

3. **Termination:**

$$P(O|\lambda) = \sum_{j=1}^{N} \pi_j\, b_j(o_1)\, \beta_1(j)$$

Viterabi Algorithm

The Viterabi Algorithm is a dynamic programming algorithm used in Hidden Markov Models (HMMs) to find the most likely sequence of hidden states that generated a given sequence of observations. It calculates the probability of each possible state sequence recursively and selects the one with the highest probability. The Viterabi Algorithm is widely used in various applications such as speech recognition, natural language processing, and bioinformatics.



1. **Initialization:**

$$\begin{aligned} v_1(j) &= \pi_j b_j(o_1) \quad & 1 \le j \le N \\ bt_1(j) &= 0 \quad & 1 \le j \le N \end{aligned}$$

2. **Recursion**

$$\begin{aligned} v_t(j) &= \max_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t); \quad 1 \le j \le N, 1 < t \le T \\ bt_t(j) &= \operatorname*{argmax}_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t); \quad 1 \le j \le N, 1 < t \le T \end{aligned}$$

3. **Termination:**

$$\begin{aligned} \text{The best score:} \quad P* &= \max_{i=1}^{N} v_T(i) \\ \text{The start of backtrace:} \quad q_T* &= \operatorname*{argmax}_{i=1}^{N} v_T(i) \end{aligned}$$

Baum-Welch Algorithm

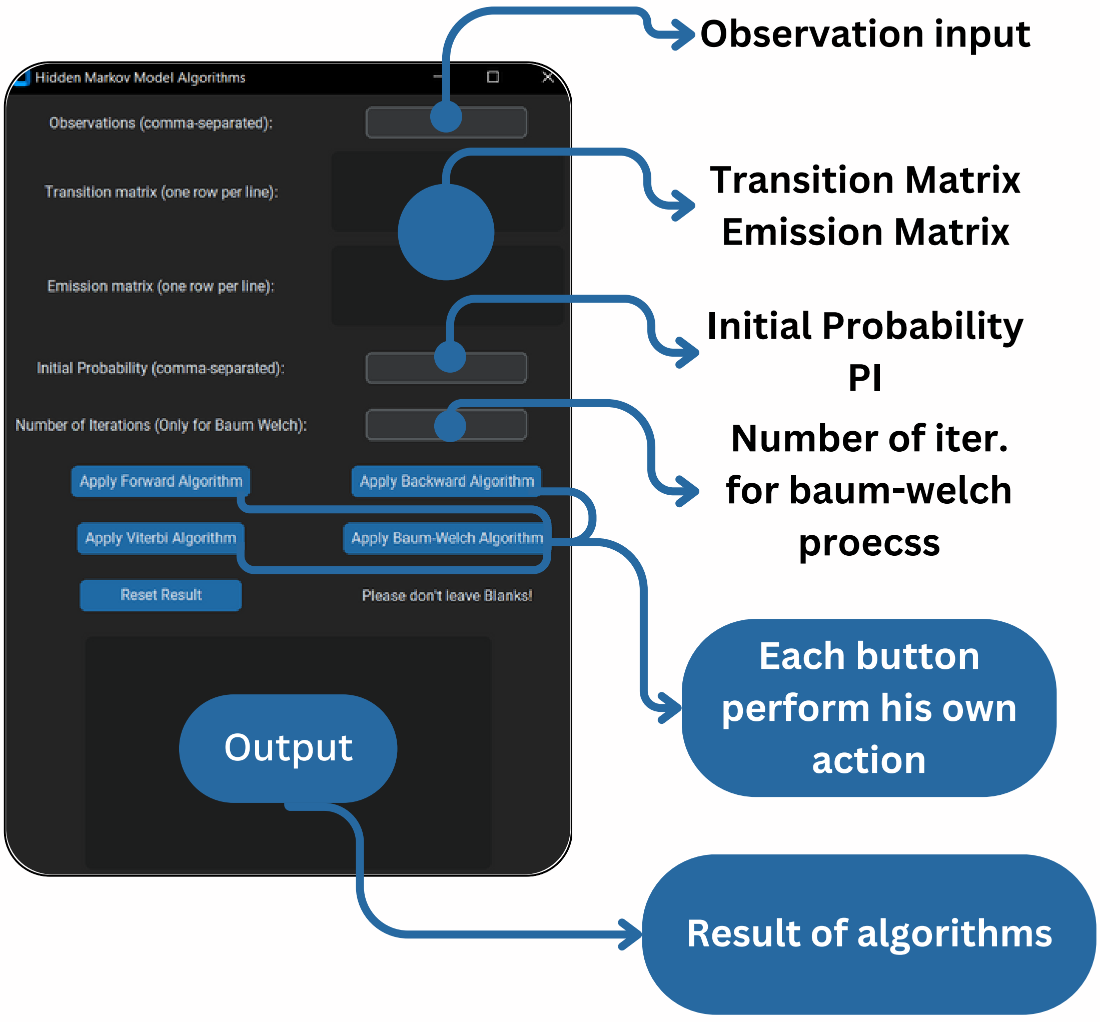
The Baum-Welch algorithm, also known as the forward-backward algorithm, is an iterative algorithm used to estimate the parameters of a hidden Markov model (HMM) given a set of observed data. It is an unsupervised learning algorithm that aims to find the maximum likelihood estimate of the model parameters.

Forward-Backward Algorithm: This step computes the forward and backward probabilities of the HMM, which are used to calculate the expected sufficient statistics for the model parameters.

Parameter Update: This step updates the model parameters based on the expected sufficient statistics obtained from the forward-backward algorithm.

Building up GUI HMM App!

we have tried to make a simple executable app about hidden markov chain, which user can add his inputs and app automatically no need to know how it runs, it's enough to see output with final answers and here's anatomy of App


Hidden Markov Model Algorithms
Observations (comma-separated):
Transition matrix (one row per line):
Emission matrix (one row per line):
Initial Probability (comma-separated):
Number of Iterations (Only for Baum Welch):
Apply Forward Algorithm
Apply Backward Algorithm
Apply Viterbi Algorithm
Apply Baum-Welch Algorithm
Reset Result
Please don't leave Blanks!
Output
Observation input
Transition Matrix
Emission Matrix
Initial Probability
PI
Number of iter.
for baum-welch
proecss
Each button perform his own action
Result of algorithms

Forward Algorithm Performing

The forward algorithm is used to calculate the likelihood of observing a particular sequence of observations in a Hidden Markov Model.

```
def forward_algorithm(observations, P, E, Pi):
    # Number of hidden states
    num_states = len(Pi)
    # Number of observations
    num_observations = len(observations)
    # Initialize the forward matrix
    forward_matrix = np.zeros((num_states, num_observations))
    # Init the first value of each state
    forward_matrix[:, 0] = Pi * E[:, observations[0]]
    # Print the initial forward variables
    print(f"Alpha at time 1:\n{forward_matrix[:, 0]}\n")
    # Fill in the rest of the forward matrix
    for t in range(1, num_observations):
        for j in range(num_states):
            forward_matrix[j, t] = np.sum(
                forward_matrix[i, t - 1] * P[i, j] * E[j, observations[t]]
                for i in range(num_states)
            )

        # Print the forward variables at each time step
        print(f"Alpha at time {t+1}:\n{forward_matrix[:, t]}\n")
    # Termination step: Compute the likelihood of the sequence by summation
of last 2 times
    likelihood = np.sum(forward_matrix[:, num_observations - 1])
    # Print the final forward matrix and likelihood
    print(
        f"Final Forward Matrix: \n{forward_matrix}\n",
        f"Likelihood of the sequence: {likelihood}",
        sep="\n",
    )
    return forward_matrix, likelihood
```

backward Algorithm Performing

This code computes the likelihood of observing a sequence in a Hidden Markov Model using the backward algorithm, focusing on the probabilities of transitioning between hidden states and emitting observations in reverse order.

```python
def backward_algorithm(observations, P, E):
  # Number of hidden states
  num_states = len(P)
  # Number of observations
  num_observations = len(observations)
  # Initialize the backward matrix
  backward_matrix = np.zeros((num_states, num_observations))
  # Initialize the last column of the backward matrix to 1
  backward_matrix[:, num_observations - 1] = 1
  # Print the initial backward variables
  print(
    f"Beta at time {num_observations}:\n{backward_matrix[:, num_observations -
1]}\n"
  )
  # Fill in the rest of the backward matrix
  for t in range(num_observations - 2, -1, -1):
    for i in range(num_states):
      backward_matrix[i, t] = np.sum(
        backward_matrix[j, t + 1] * P[i, j] * E[j, observations[t + 1]]
        for j in range(num_states)
      )

    # Print the backward variables at each time step
    print(f"Beta at time {t+1}:\n{backward_matrix[:, t]}\n")

  # Termination step: Compute the likelihood of the sequence using the initial
probabilities and the first observation
  likelihood = np.sum(
    backward_matrix[i, 0] * P[0, j] * E[j, observations[0]]
    for j in range(num_states)
  )

  # Print the final backward matrix and likelihood
  print(
    f"Final Backward Matrix: \n{backward_matrix}\n",
    f"Likelihood of the sequence: {likelihood}",
    sep="\n",
  )

  return backward_matrix, likelihood
```

Viterabi Algorithm Performing

This code employs the Viterbi algorithm to determine the most probable sequence of hidden states in a Hidden Markov Model given a set of observations, calculating both the best path and its associated probability.

```
def viterbi_algorithm(observations, P, E, Pi):
  # Number of hidden states
  num_states = len(Pi)
  # Number of observations
  num_observations = len(observations)
  # Initialize the Viterbi matrix
  viterbi_matrix = np.zeros((num_states, num_observations))
  # Initialize the backpointer matrix
  path = np.zeros((num_states, num_observations), dtype=int)
  # Initialization step
  viterbi_matrix[:, 0] = Pi * E[:, observations[0]]
  # Recursion step
  for t in range(1, num_observations):
    for j in range(num_states):
      prev_probs = viterbi_matrix[:, t - 1] * P[:, j] * E[j,
observations[t]]
      viterbi_matrix[j, t] = round(max(prev_probs), 5)
      path[j, t] = np.argmax(prev_probs)

  # Termination step
  best_path_prob = max(viterbi_matrix[:, -1])
  best_last_state = np.argmax(viterbi_matrix[:, -1])

  # Backtrace to find the best path
  best_path = [best_last_state]
  for t in range(num_observations - 1, 0, -1):
    best_last_state = path[best_last_state, t]
    best_path.insert(0, best_last_state)

  return best_path, best_path_prob, viterbi_matrix
```

Baum-Welch Performing

This code implements the Baum-Welch algorithm, iterating through the Expectation-Maximization process to estimate the parameters (initial state probabilities, transition probabilities, and emission probabilities) of a Hidden Markov Model based on a given sequence of observations.

```
def baum_welch(Pi,Emission,Transition,observations,
  num_iterations=100):
  state_numbers = len(Pi)
  # Iterate through the Baum-Welch algorithm
  for iteration in range(num_iterations):
    # E-step: Use forward and backward algorithms to calculate expected counts
    forward_matrix, forward_likelihood = forward_algorithm(
      observations, Transition, Emission, Pi)
    backward_matrix, backward_likelihood = backward_algorithm(observations, Transition, Emission)
    # M-step: Update parameters based on expected counts
    # Update initial state probabilities
    Pi = forward_matrix[:, 0] * backward_matrix[:, 0] / forward_likelihood

    # Update transition probabilities
    for i in range(state_numbers):
      for j in range(state_numbers):
        numer = sum(
          forward_matrix[i, t]
          * Transition[i, j]
           * Emission[j, observations[t + 1]]
          * backward_matrix[j, t + 1]
          for t in range(len(observations) - 1)
        )
        denom = sum(
          forward_matrix[i, t] * backward_matrix[i, t]
          for t in range(len(observations) - 1)
        )
        Transition[i, j] = numer / denom
    # Update emission probabilities
    for j in range(state_numbers):
      for k in range(state_numbers):
        numer = sum(
          forward_matrix[i, t] * backward_matrix[i, t]
          if observations[t] == k
          else 0
          for t in range(len(observations) - 1)
        )
        denom = sum(
          forward_matrix[i, t] * backward_matrix[i, t]
          for t in range(len(observations) - 1)
        )
        Emission[j, k] = numer / denom

  return Pi, Transition, Emission
```