Lecture Pattern Analysis

Part 20: Hidden Markov Models (HMMs)

Christian Riess
IT Security Infrastructures Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg
July 12, 2022

Introduction

- Hidden Markov Models (HMMs) are a classic tool to process sequential data[1]
- HMMs can be seen as a probabilistic state machine that maps observation sequences to states
- The states are hidden variables, because a user only sees input and output but not the states
- Example applications are speech recognition and weather forecasts
- In modern sequence processing, neural networks supersede HMMs, but
  - many modern ideas are inspired on classic HMM knowledge
  - HMMs are highly instructive due to their concise probabilistic model
- HMMs are generative probabilistic models, hence, it is even possible to draw new samples from a trained model

[1] Literature references: Bishop Chap. 13–13.2 covers HMMs, and the content of slide 2–4 is taken from the first few pages. However, the state model view on HMMs is also interesting (and appears very natural to me as a computer scientist), which is why we use from slide 5 on, and in the next videos, the paper and the notation by Rabiner's paper, particularly Sec. II and Sec. III.

Preliminary Considerations on Sequential Data (1/3)

- Consider the task of predicting the weather for each day
- We can make measurements $\mathbf{x}_t$ for each day t like temperature, humidity, ...
- Treating each day's measurements $\mathbf{x}_t$ independently is certainly bad: oftentimes, the weather of day t is the same as on day $t-1$
- So let us condition day t on day $t-1$, which is a **first-order Markov model**,

$$p(\mathbf{x}_1, \ldots, \mathbf{x}_T) = p(\mathbf{x}_1) \cdot \prod_{t=2}^{T} p(\mathbf{x}_t | \mathbf{x}_1, \ldots, \mathbf{x}_{t-1}) \overset{!}{=} p(\mathbf{x}_1) \prod_{t=2}^{T} p(\mathbf{x}_t | \mathbf{x}_{t-1}) \quad (1)$$

- The associated graphical model is

$$\begin{tikzcd} \mathbf{x}_1 \arrow[r] & \mathbf{x}_2 \arrow[r] & \cdots \arrow[r] & \mathbf{x}_T \end{tikzcd}$$

- Inference on such linear chains of dependencies can be done efficiently[2]

[2] If you are curious: a general treatment of inference on chain graphs is in Bishop Sec. 8.4.1–8.4.5

Preliminary Considerations on Sequential Data (2/3)

- One might hypothesize that a model that predicts today's weather from only the previous day quickly runs into a performance saturation
- A direct extension: predict today's weather from the last two days
- Such a second order Markov chain provides a more expressive model,

$$p(\mathbf{x}_1, \ldots, \mathbf{x}_T) = p(\mathbf{x}_1) p(\mathbf{x}_2 | \mathbf{x}_1) \prod_{t=3}^{T} p(\mathbf{x}_t | \mathbf{x}_{t-1}, \mathbf{x}_{t-2}) \qquad (2)$$

which corresponds to the graphical model

$$\begin{tikzcd}
\mathbf{x}_1 \arrow[r] \arrow[rr, bend right] & \mathbf{x}_2 \arrow[r] \arrow[rr, bend right] & \mathbf{x}_3 \arrow[r] \arrow[rr, bend right] & \cdots & \cdots
\end{tikzcd}$$

Preliminary Considerations on Sequential Data (3/3)

- If there are w weather types (e.g., $w = 3$: $\{$“sun”, “rain”, “snow”$\}$), then
  - The first-order model must learn w^2 transitions from $\mathbf{x}_{t-1}$ to $\mathbf{x}_t$
  - The second-order model must learn w^3 transitions from $\mathbf{x}_{t-2}$ and $\mathbf{x}_{t-1}$ to $\mathbf{x}_t$
- This approach does not scale to higher order Markov chains: the number of parameters grow exponentially in the order of the Markov chain

- HMMs use a trick to keep the number of parameters tractable: they use latent variables $\mathbf{z}_t$
- These latent variables $\mathbf{z}_t$ encode **discrete** states
- The states could model for example weather periods of the year, like “April weather”, “Indian summer”, or the seasons
- We can then match a sequence of days with particular weather to a plausible sequence of states

More Formally: HMM Joint Distribution

- The joint distribution is

$$p(\mathbf{x}_1, \ldots, \mathbf{x}_T, \mathbf{z}_1, \ldots, \mathbf{z}_T) = p(\mathbf{x}_1, \ldots, \mathbf{x}_T | \mathbf{z}_1, \ldots, \mathbf{z}_T) \cdot p(\mathbf{z}_1, \ldots, \mathbf{z}_T) \quad (3)$$

with observation sequence $\mathbf{x}_t$ and hidden state sequence $\mathbf{z}_t$ $(1 \leq t \leq T)$

- These assumptions are used to further factorize the model:
  1. Each observation only depends on a single hidden state, i.e.,

$$p(\mathbf{x}_1, \ldots, \mathbf{x}_T | \mathbf{z}_1, \ldots, \mathbf{z}_T) = \prod_{t=1}^{T} p(\mathbf{x}_t | \mathbf{z}_t) \quad (4)$$

  2. Each state only depends on its predecessor (again the Markov assumption!)

$$p(\mathbf{z}_1, \ldots, \mathbf{z}_T) = p(\mathbf{z}_1) \prod_{t=2}^{T} p(\mathbf{z}_t | \mathbf{z}_{t-1}) \quad (5)$$

- Hence, the factorized HMM consists of small terms

$$p(\mathbf{x}_1, \ldots, \mathbf{x}_T, \mathbf{z}_1, \ldots, \mathbf{z}_T) = p(\mathbf{z}_1) \cdot \prod_{t=1}^{T} p(\mathbf{x}_t | \mathbf{z}_t) \cdot \prod_{t=2}^{T} p(\mathbf{z}_t | \mathbf{z}_{t-1}) \quad (6)$$

HMMs as Graphical Models

- The associated graphical model is

$$\begin{tikzcd}
\mathbf{z}_1 \arrow[r] \arrow[d] & \mathbf{z}_2 \arrow[r] \arrow[d] & \cdots \arrow[r] & \mathbf{z}_T \arrow[d] \\
\mathbf{x}_1 & \mathbf{x}_2 & & \mathbf{x}_T
\end{tikzcd}$$

- This is only a minor variation of the previously seen chain of dependencies, and it is well tractable (more on this in the next videos)
- The observation sequence is now indirectly represented in a state sequence
- We will see that the indirection through a state sequence can describe complex data in a **compact representation** via
  1. distributions of the starting state $p(\mathbf{z}_1)$,
  2. likelihoods of state transitions $p(\mathbf{z}_t|\mathbf{z}_{t-1})$, and
  3. likelihoods of observations in each state $p(\mathbf{x}_t|\mathbf{z}_t)$
- There are many analogies to state machines, and we now adopt this view

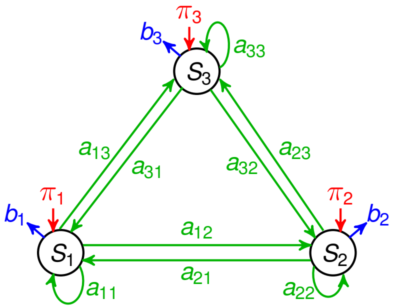
HMMs as State Machines: Notation More Similar to Rabiner

- A HMM consists of N states S_i
- We operate on a sequence of T observations[3], where (for now) each observation is a discrete symbol o_v from an alphabet of size V
- The HMM parameters are $\lambda = (\mathbf{A}, \mathbf{B}, \boldsymbol{\pi})$, where
  - $\mathbf{A} = [a_{S_i S_j}] \in \mathbb{R}^{N \times N}$ is a matrix of state transitions from state S_i to S_j
    The outgoing transition probabilities form a discrete probability distribution, i.e.,
    $$\sum_{j=1}^{N} a_{S_i S_j} = 1 \qquad (7)$$
  - $\mathbf{B} = [b_{S_i}(o_v)] \in \mathbb{R}^{N \times V}$ is a matrix of symbol probabilities per state S_i with
    $$\sum_{v=1}^{V} b_{S_i}(o_v) = 1 \qquad (8)$$
  - $\boldsymbol{\pi} = [\pi_{S_i}] \in \mathbb{R}^{N}$ is a vector of probabilities that the sequence starts in state i,
    $$\sum_{i=1}^{N} \pi_{S_i} = 1 \qquad (9)$$

[3] In other parts of PA, we use N for the number of observations, which is here T (“time”)

HMMs as State Machines: Sketch

- The HMM can be sketched like a classical state machine:



- It calculates a probability that an input sequence matches the learned model by marginalizing over all paths, where one path consists of
  - the starting probability
  - the state transition probabilities
  - the output probabilities per state

Three Algorithms for HMMs

- Training and evaluation of an HMM $\lambda = (\mathbf{A}, \mathbf{B}, \pi)$ requires three algorithms:
  1. Matching: probability that an observation "belongs" to λ, i.e., $p(\mathbf{x}_1, \ldots, \mathbf{x}_T | \lambda)$
     This is done with the forward algorithm (or the analogous backward algorithm)
  2. Finding the most likely state sequence $\underset{z_t = S_j}{\operatorname{argmax}}\, p(\mathbf{x}_1, \ldots, \mathbf{x}_T, z_1, \ldots, z_T | \lambda)$
     This is done with the Viterbi algorithm
  3. HMM training, i.e., determine $\lambda = (\mathbf{A}, \mathbf{B}, \pi)$ from many training sequences $\mathbf{x}_1, \ldots, \mathbf{x}_T$, where T may vary from sequence to sequence
     This is done with the Baum-Welch formulae, which is an EM algorithm
- The conditional dependency on λ in algorithm 1 and 2 emphasizes that these algorithms operate on the trained model; many authors omit them
- We go through these algorithms in the next videos
- For now, let us have a quick look at an example application after noting that
  - HMMs are fully probabilistic,
  - hence also their composition is fully probabilistic

Example HMM Application

- Classical speech recognizers nest HMMs:
  - At the lowest level, each HMM recognizes a single phoneme
    Phonemes are short sounds ($\approx$ 25 ms)
  - At the second level, each HMM recognizes a word from the phonemes
    The vocabulary is application-specific, e.g., “want” “ticket”, “Frankfurt”
  - There may even be a third level (given sufficient training data), to recognize specific sentences
    Again, these sentences are domain-specific, e.g., “I want a ticket to Frankfurt”
- Hence, the speech recognizer consists of a large number of small HMMs

- Outlook: How is speech recognition done today?
  - Modern speech recognizers use transformer networks
  - Transformers generalize the idea of a state machine. Here, different input words trigger different neighborhood searches for matching context

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG

SCHOOL OF ENGINEERING

Lecture Pattern Analysis

Part 21: HMMs Algorithms 1 and 2

Christian Riess

IT Security Infrastructures Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg

July 11, 2022

Algorithm 1: How Well Matches an Input Sequence the HMM?

- Calculate $p(\mathbf{x}_1, \ldots, \mathbf{x}_T | \lambda)$ by marginalizing over all state sequences $z_1, \ldots, z_T$
- Naive marginalization is exponential in the sequence length,

$$p(\mathbf{x}_1, \ldots, \mathbf{x}_T) = \sum_{z_1=S_1}^{S_N} \sum_{z_2=S_1}^{S_N} \ldots \sum_{z_T=S_1}^{S_N} \pi_{z_1} b_{z_1}(\mathbf{x}_1) a_{z_1 z_2} b_{z_2}(\mathbf{x}_2) \ldots a_{z_{T-1} z_T} b_{z_T}(\mathbf{x}_T) \ , \tag{1}$$

  with a computational cost of $O(N^T)$
- However, the chain structure of dependencies admits an efficient dynamic programming solution

$$p(\mathbf{x}_1, \ldots, \mathbf{x}_T) = \sum_{z_1=S_1}^{S_N} \pi_{z_1} b_{z_1}(\mathbf{x}_1) \sum_{z_2=S_1}^{S_N} a_{z_1 z_2} b_{z_2}(\mathbf{x}_2) \ldots \sum_{z_T=S_1}^{S_N} a_{z_{T-1} z_T} b_{z_T}(\mathbf{x}_T) \tag{2}$$

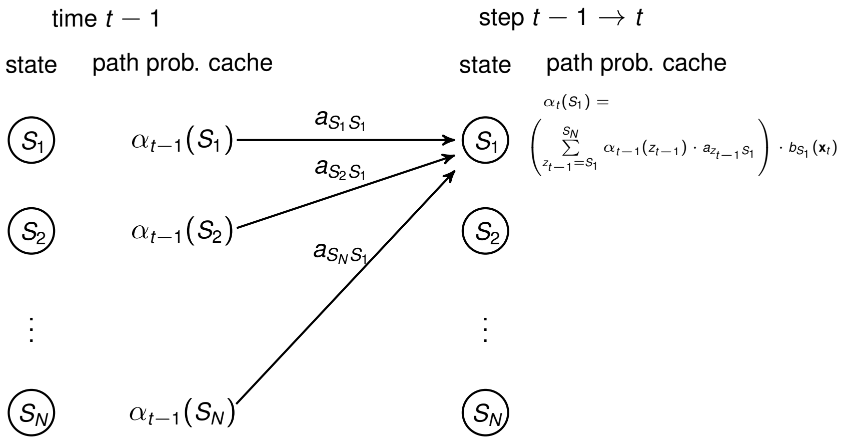
  with caching of the N accumulated state probabilities at time t.
  Each state transition requires $O(N^2)$ evaluations, the total cost is $O(N^2 T)$

The Forward Algorithm and the Backward Algorithm

- The forward algorithm directly implements Eqn. 2:
  1. Initialization (here and below, S_i implicitly creates a list of N entries):
     $t = 1$: $\alpha_1(z_1 = S_i) = \pi_{z_1} b_{z_1}(\mathbf{x}_1)$
  2. Iteration:
     $$2 \leq t \leq T\colon \alpha_t(z_t = S_i) = \left(\sum_{z_{t-1}=S_1}^{S_N} \alpha_{t-1}(z_{t-1}) \cdot a_{z_{t-1} z_t} \right) b_{z_t}(\mathbf{x}_t)$$
  3. Summation over all end points:
     $$t = T\colon \qquad p(\mathbf{x}_1, \ldots, \mathbf{x}_T) = \sum_{z_T=S_1}^{S_N} \alpha_T(z_t)$$
- Training requires the (almost identical) backward alg., starting at time T:
  1. Initialization:
     $t = T$: $\beta_T(z_1 = S_i) = 1$
  2. Iteration: $1 \leq t \leq T\colon \beta_t(z_t = S_i) = \sum_{z_{t+1}=S_1}^{S_N} \beta_{t+1}(z_{t+1}) b_{z_{t+1}}(\mathbf{x}_{t+1}) a_{z_t z_{t+1}}$
- The backward iteration does not include $\mathbf{x}_1$, which is OK for its training use

Forward Algorithm: State-Time Diagram

- This diagram over states (y-axis) and time (x-axis) illustrates the caching:



Algorithm 2: What is the Most Likely State Sequence?

- Finding $\underset{z_t=S_j}{\operatorname{argmax}}\, p(\mathbf{x}_1, \ldots, \mathbf{x}_T, z_1, \ldots, z_T | \lambda)$ is almost identical to forward alg.
  - Variable $\delta_t(z_i)$ caches the likelihood of the most likely path to z_i within time $1..t$
  - Variable $\psi_t(z_i)$ caches the predecessor state to reconstruct the path
- Viterbi algorithm:
  1. Initialization:
     $$t = 1: \quad \delta_1(z_1 = S_i) = \pi_{z_1} b_{z_1}(\mathbf{x}_1)$$
     $$\psi_1(z_1 = S_i) = 0$$
  2. Iteration:
     $$1 \leq t \leq T: \delta_t(z_t = S_i) = \max_{S_1 \leq z_{t-1} \leq S_N} \left(\delta_{t-1}(z_{t-1}) \cdot a_{z_{t-1} z_t}\right) b_{z_t}(\mathbf{x}_t)$$
     $$\psi_t(z_t) = \underset{S_1 \leq z_{t-1} \leq S_N}{\operatorname{argmax}}\, \delta_{t-1}(z_{t-1}) \cdot a_{z_{t-1} z_t}$$
  3. Maximum over all end points:
     $$t = T: \quad p^* = \max_{S_1 \leq z_T \leq S_N} \delta_T(z_T)$$
     $$z_T^* = \underset{S_1 \leq z_T \leq S_N}{\operatorname{argmax}}\, \delta_T(z_T)$$
  4. Path reconstruction: $z_t^* = \psi_{t+1}(z_{t+1}^*)$

Remarks

- The computational complexity of the Viterbi algorithm is also $O(N^2 T)$
- Interpretation of the caches $\alpha_t(S_i)$, $\beta_t(S_i)$, $\delta_t(S_i)$:
    - $\alpha_t(S_i)$ is the sum of probabilities of all paths from time 1 to time t that are at time t in state S_i
    - $\beta_t(S_i)$ is the sum of probabilities of all paths from time T to time t that are at time t in state S_i
    - $\delta_t(S_i)$ is the probability of the single, most likely path from time 1 to time t that is at time t in state S_i

  We will use these interpretations to explain the training procedure
- The Viterbi algorithm has also many other applications, e.g., as a decoder for partially corrupted digital watermarks (see lecture “Multimedia Security”)

Lecture Pattern Analysis

Part 22: HMMs Algorithm 3 and Remarks

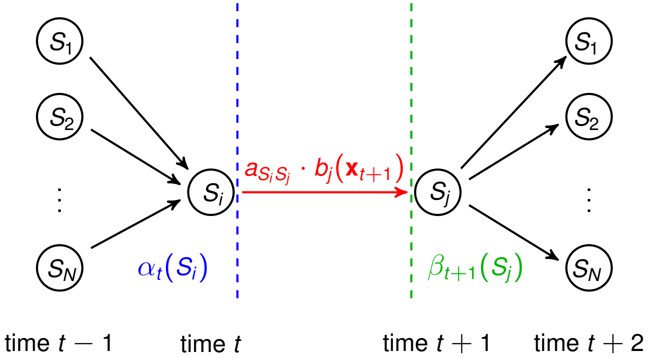
Christian Riess
IT Security Infrastructures Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg
July 12, 2022

Overview

- The HMM training is an Expectation-Maximization algorithm
- For historical reasons, this EM is called Baum-Welch formulae
- Expectation Step:
  - The responsibilities are estimations for the current state or state transition, and their associated expectations
  - Key idea: Forward and backward algorithm marginalize out most parameters, to isolate the parameter of interest
- Maximization Step:
  - The maximization step updates the model parameters
  - Key idea: counting relative frequencies converges to a local optimum
- As in the previous videos, $\mathbf{x}_t$ is a position in one input sequence $\mathbf{x}_1, \ldots, \mathbf{x}_T$
- However, the training is done over a dataset, so mentally add an outer loop around the calculations to accumulate the probabilities of all samples

Towards Responsibilities: Accessing Hidden State Transitions

- Recall that the state sequence is “hidden”, and that we consider for HMM matching all state sequences
- For training, we have to update, e.g., an individual state transition $a_{S_iS_j}$
- This can be accessed by combining $\alpha_t(z_t = S_i)$ and $\beta_{t+1}(z_{t+1} = S_j)$:



Expectation Step: Calculation of the Responsibilities

- Likelihood for a state transition $S_i \to S_j$ at time t:

$$\xi_t(i,j) = p(z_t = S_i, z_{t+1} = S_j) \tag{1}$$

$$= \frac{\alpha_t(S_i) \cdot a_{S_i S_j} \cdot b_{t+1}(\mathbf{x}_{t+1})\beta_{t+1}(S_j)}{\sum_{z_t=S_1}^{S_N} \sum_{z_{t+1}=S_1}^{S_N} \alpha_t(z_t) \cdot a_{z_t z_{t+1}} \cdot b_{t+1}(\mathbf{x}_{t+1})\dot{\beta}_{t+1}(z_{t+1})} \tag{2}$$

- Likelihood for being in state S_i at time t: marginalize out state S_j via

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j) \tag{3}$$

- Expected number of transitions $S_i \to S_j$ at all times: $\sum_{t=1}^{T} \xi_t(i,j)$
- Expected number of times S_i is visited: $\sum_{t=1}^{T} \gamma_t(i)$

Maximization Step

- Update of the starting probabilities:

$$\bar{\pi}_{S_i} = \gamma_1(i) = \text{expected \# times in } S_i \text{ in time } t \text{ over all training data} \qquad (4)$$

- Update of the state transition probabilities:

$$\bar{a}_{S_i S_j} = \frac{\sum_{t=1}^{T} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} = \frac{\text{exp. \# of transitions } S_i \rightarrow S_j}{\text{exp. \# of times in state } S_i} \qquad (5)$$

- Update of the output probabilities (discrete alphabet):

$$\bar{b}_{S_i}(o_v) = \frac{\sum_{t=1}^{T} \gamma_t(i) \cdot \mathbb{I}(\mathbf{x}_t, o_v)}{\sum_{t=1}^{T} \gamma_t(S_i)} = \frac{\text{expected \# times in } S_i \text{ and observing } o_v}{\text{expected \# times in } S_i}$$

where $\mathbb{I}$ is the discrete indicator function (analogous to Dirac δ) (6)

Remarks

- An HMM is called **ergodic** if every state can be reached from every other state
- In practice, **left-right HMMs** are much more commonly used. Left-right HMMs only allow forward edges and self-loops, i.e., $a_{s_i s_j} = 0$ if $i > j$
- It is not difficult to change the discrete alphabet to continuous observations
  - For example a GMM can model how well a state matches an observation (we use such a model in the hand-writing recognition exercise)
  - GMM training and HMM training work well together: both are fully probabilistic models, both are trained via EM
- Good parameter initializations before training can significantly improve results. For example, Rabiner (Sec. VI.F) proposes to pre-cluster speech segments for initial state assignments