Lecture Pattern Analysis

Part 18: Hidden Markov Models (HMMs)

Christian Riess
IT Security Infrastructures Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg
June 17, 2021

Introduction

- Hidden Markov Models (HMMs) are the classic tool to process sequential data[1]
- Applications are for example speech recognition and weather forecasts
- Nowadays, neural networks (NNs) supersede HMMs, but much of the NN design is oftentimes oriented on HMM knowledge
- HMMs are generative probabilistic models, hence, it is even possible to draw new samples from a trained model
- 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

[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 (1/2)

- 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 (2/2)

- Second order Markov chains provide a more expressive model,

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

  which corresponds to the graphical model

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

- Unfortunately, this approach does not scale to N-th order Markov chains: the number of parameters grows exponentially in the order of the Markov chain
- Trick to keep the number of parameters tractable: introduce latent variables $\mathbf{z}_t$
- In a Hidden Markov Model, $\mathbf{z}_t$ encodes **discrete** states
- The states could model for example “April weather”, “Indian summer” and other weather periods of the year

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. distributions of state transitions $p(\mathbf{z}_t|\mathbf{z}_{t-1})$, and
  3. distributions 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

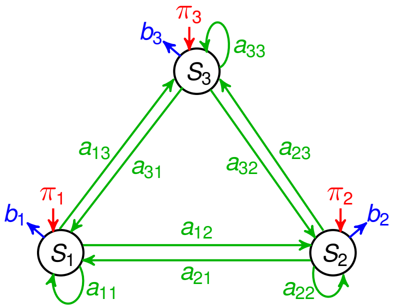
HMMs as State Machines: Notation More Similar to Rabiner

- 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.,
    $$\sum_{j=1}^{N} a_{S_i S_j} = 1 \tag{7}$$
  - $\mathbf{B} = [b_{S_i}(o_v)] \in \mathbb{R}^{N \times V}$ is a matrix of symbol probabilities per state S_i with
    $$\sum_{v=1}^{V} b_{S_i}(o_v) = 1 \tag{8}$$
  - $\boldsymbol{\pi} = [\pi_{S_i}] \in \mathbb{R}^{N}$ is a vector of probabilities that the sequence starts in state i,
    $$\sum_{i=1}^{N} \pi_{S_i} = 1 \tag{9}$$

[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
- Of course, modern speech recognizers use deep neural networks, but they oftentimes draw design inspiration from the theoretically well-founded HMMs