Machine Learning I
Week 14: Sequence Learning Introduction

Alex Graves

Technische Universität München

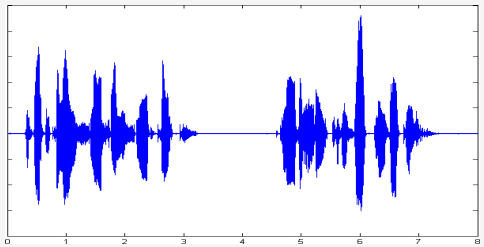
29. January 2009

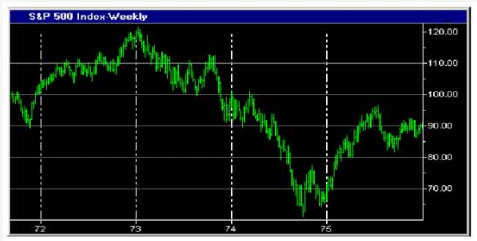
Literature

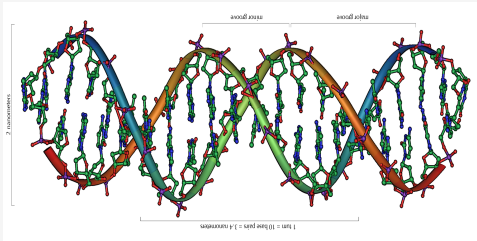
- Pattern Recognition and Machine Learning
  Chapter 13: Sequential Data
  Christopher M. Bishop
- Machine Learning for Sequential Data: A Review
  Thomas G. Dietterich, review paper
- Markovian Models for Sequential Data
  Yoshua Bengio, review paper
- Supervised Sequence Labelling with Recurrent Neural Networks
  Alex Graves, Ph.D. thesis
- On Intelligence
  Jeff Hawkins

What is Sequence Learning?

- Most machine learning algorithms are designed for independent, identically distributed (i.i.d.) data
- But many interesting data types are *not* i.i.d.
- In particular the successive points in sequential data are strongly correlated







- Sequence learning is the study of machine learning algorithms designed for sequential data. These algorithms should
  1. not assume data points to be independent
  2. be able to deal with sequential distortions
  3. make use of context information

What is Sequence Learning Used for?

Time-Series Prediction

Tasks where the history of a time series is used to predict the next point. Applications include stock market prediction, weather forecasting, object tracking, disaster prediction...

Sequence Labelling

Tasks where a sequence of labels is applied to a sequence of data. Applications include speech recognition, gesture recognition, protein secondary structure prediction, handwriting recognition...

- For now we will concentrate on sequence labelling, but most algorithms are applicable to both

Definition of Sequence Labelling

- Sequence labelling is a supervised learning task where pairs $(\mathbf{x}, \mathbf{t})$ of input sequences and target label sequences are used for training
- The inputs $\mathbf{x}$ come from the set $\mathcal{X} = (\mathbb{R}^m)^*$ of sequences of m-dimensional real-valued vectors, for some fixed m
- The targets $\mathbf{t}$ come from the set $\mathcal{T} = L^*$ of strings over the alphabet L of labels used for the task
- In each pair $(\mathbf{x}, \mathbf{t})$ the target sequence is at most as long as the input sequence: $|\mathbf{t}| \leq |\mathbf{x}|$. They are *not necessarily the same length*

Definition (Sequence Labelling)

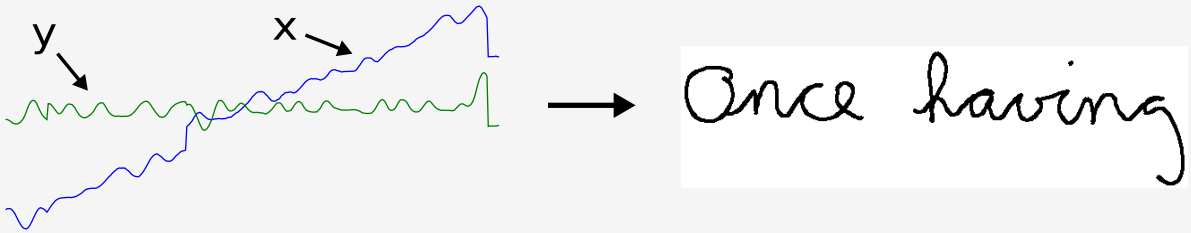
Given a training set A and a test set B, both drawn independently from a fixed distribution $\mathcal{D}_{\mathcal{X} \times \mathcal{T}}$, the goal is to use A to train an algorithm $h : \mathcal{X} \mapsto \mathcal{T}$ to label B in a way that minimises some task-specific error measure

Comments

- We assume that the distribution $\mathcal{D}_{\mathcal{X}\times\mathcal{T}}$ that generates the data is stationary — i.e. the probability of some $(\mathbf{x}, \mathbf{t}) \in \mathcal{D}_{\mathcal{X}\times\mathcal{T}}$ remains constant over time (strictly speaking this does not apply to e.g. financial and weather data, because markets and climates change over time)
- Therefore the sequences (but not the individual data points) are i.i.d. This means that much of the reasoning underlying standard machine learning algorithms also applies here, only at the level of sequences and not points

Motivating Example

- Online handwriting recognition is the recognition of words and letters from sequences of pen positions
- The inputs are the x and y coordinates of the pen, so m is 2

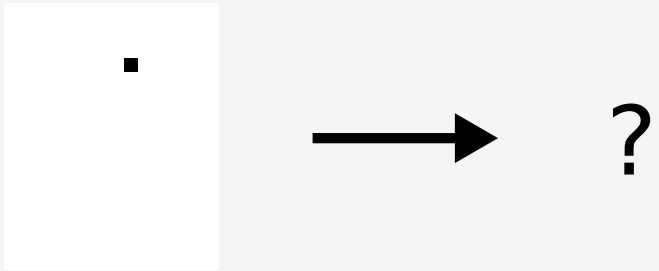


- The label alphabet L is just the usual Latin alphabet, possibly with extra labels for punctuation marks etc.
- The error measure is the edit distance between the output of the classifier and the target sequence

$$\text{input} \longrightarrow \text{itput} \longrightarrow \text{utput} \longrightarrow \text{output}$$

Online Handwriting with Non-Sequential Algorithms

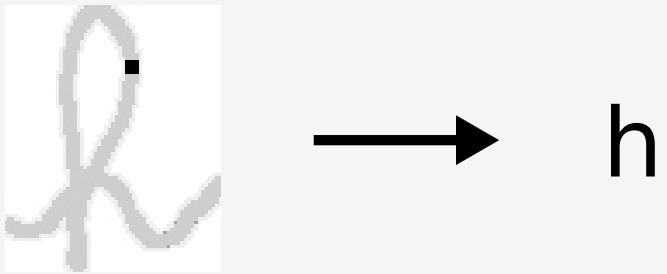
- If we assume the data-points are independent we should classify each co-ordinate separately



- Clearly impossible! Each point is only meaningful in the *context* of its surroundings
- Obvious solution is to classify an input window around each point
- This is the usual approach when standard ML algorithms (SVMs, MLPs etc) are applied to sequential data

Online Handwriting with Non-Sequential Algorithms

- If we assume the data-points are independent we should classify each co-ordinate separately



- Clearly impossible! Each point is only meaningful in the *context* of its surroundings
- Obvious solution is to classify an input window around each point
- This is the usual approach when standard ML algorithms (SVMs, MLPs etc) are applied to sequential data

Context and Input Windows

- One problem is that it is difficult to determine in advance how big the window should be

- Too small gives poor performance, too big is computationally unfeasible (too many parameters)
- Have to hand-tune for the dataset, depending on writing style, input resolution etc.

Context and Input Windows

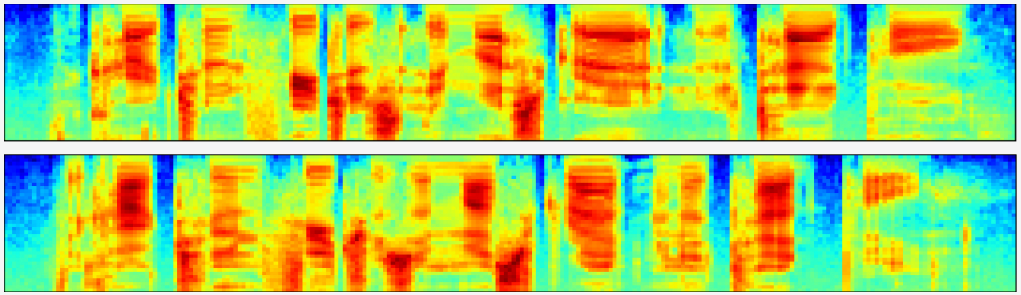
- One problem is that it is difficult to determine in advance how big the window should be

- Too small gives poor performance, too big is computationally unfeasible (too many parameters)
- Have to hand-tune for the dataset, depending on writing style, input resolution etc.

Sequential Distortion and Input Windows

- A deeper problem is that the same patterns often appear stretched or compressed along the time axis in different sequences
- In handwriting this is caused by variations in writing style

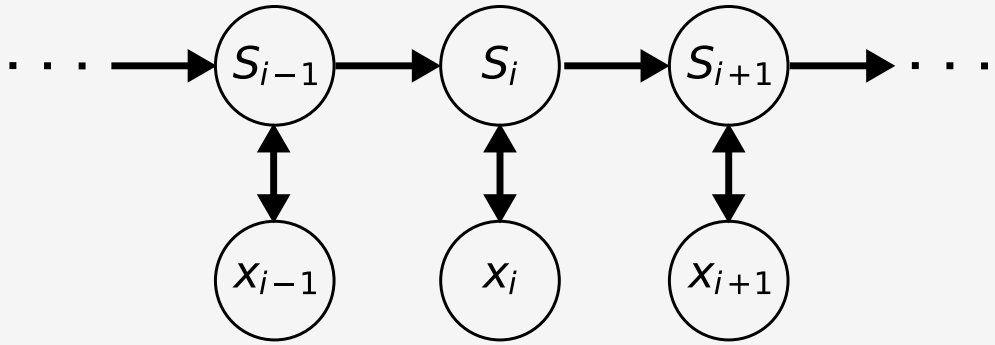
- In speech it comes from variations in speaking rate, prosody etc.



- Input windows are not robust to this because they ignore the relationship between the data-points. Even a 1 pixel shift looks like a completely different image!
- This means poor generalisation and lots of training data needed

Hidden State Architectures

- A better solution is to use an architecture with an internal hidden state that depends on both the *previous* state and the *current* data point



- The chain of hidden states acts like a 'memory' of the data
- Can be extended to look several states back with no change to the basic structure (but an increase in computational cost)
- There are many types of hidden state architecture, including Hidden Markov models, recurrent neural networks, linear dynamical systems, extended Kalman filters...

Advantages of Hidden State Architectures

- Context is passed along by the 'memory' stored in the previous states, so the problem of fixed-size input windows are avoided

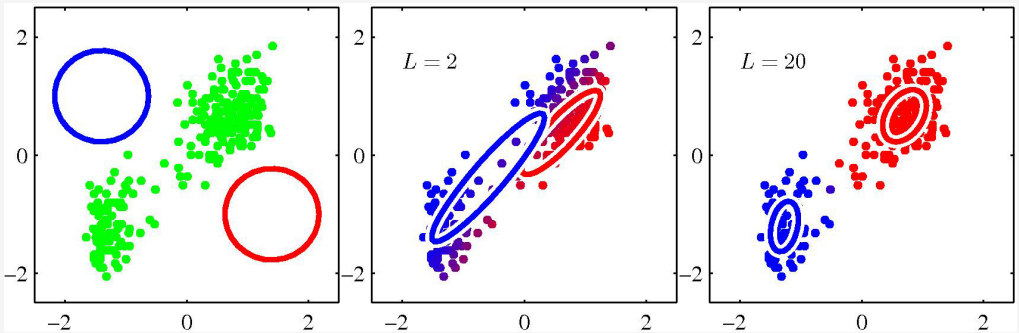
$$\Pr(s_n|x_i) = \Pr(s_n|s_{n-1}) \Pr(s_{n-1}|s_{n-2}) \dots \Pr(s_i|x_i)$$

- And *typically* require fewer parameters than input windows
- Sequential distortions can be accommodated by slight changes to the hidden state sequence. Similar sequences 'look' similar to the algorithm.

- The general principle is that the structure of the architecture matches the structure of the data. Put another way, hidden state architectures are biased towards sequential data.

Hidden Markov Models

- Hidden Markov models (HMMs) are a generative hidden state architecture where sequences of discrete hidden states are matched to observation sequences.
- Recap: generative models attempt to determine the probability of the inputs (observations) given some class or label: $\Pr(x|C_k)$
- Fitting a mixture of Gaussians to data is a well known example of a generative model with a hidden state

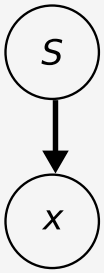


- Can think of HMMs as a sequential version of a mixture model

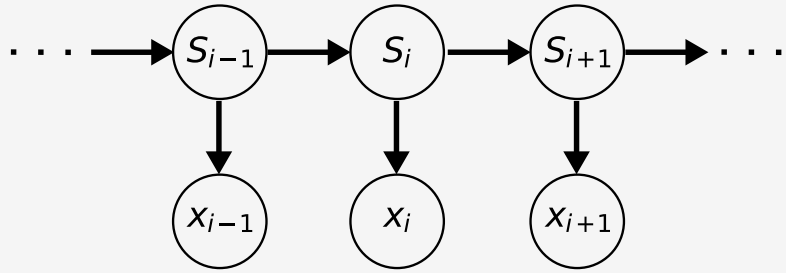
Hidden Markov Models

- In a simple mixture model the observations are conditioned on the states. This is still true for HMMs, but now the states are conditioned on the previous states as well

Mixture of Gaussians



Hidden Markov Model



- This creates the following joint distribution over states and observations

$$\Pr(\mathbf{x}, \mathbf{s}|\theta) = \Pr(s_1|\pi) \prod_{i=2}^{N} \Pr(s_i|s_{i-1}, A) \prod_{i=1}^{N} \Pr(x_i|s_i, \phi)$$

where $\theta = \{\pi, A, \phi\}$ are the HMM parameters and N is the sequence length

HMM Parameters

- $Pr(s_1 = k|\pi) = \pi_k$ are the initial probabilities of the states
- $\Pr(s_i = k|s_{i-1} = j, A) = A_{jk}$ is the matrix of transition probabilities between states. Note that some of its entries may be zero, since not all transitions are necessarily allowed
- $\Pr(x_i|s_i, \phi)$ are the emission probabilities of the observations given the states. The form of $\Pr(x_i|s_i, \phi)$ depends on the task, and the performance of HMMs depends critically on choosing a distribution able to accurately model the data. For cases where a single Gaussian is not flexible enough, mixtures of Gaussian and neural networks are common choices.

$$\Pr(x_i|s_i, \phi) = \prod_{k=1}^{n} a_k^i \mathcal{N}(x_i|\mu_k^i, \Sigma_k^i)$$

Training and Using HMMs

- Like most parametric models, HMMs are trained by adjusting the parameters to maximise the log-likelihood of the training data

$$\log \Pr(A|\theta) = \sum_{(\mathbf{x},\mathbf{t})\in A} \log \sum_{\mathbf{s}} \Pr(\mathbf{x}, \mathbf{s}|\theta)$$

  This can be done efficiently with the Baum-Welch algorithm

- Once trained, we use the HMM to label a new data sequence $\mathbf{x}$ by finding the state sequence $\mathbf{s}^*$ that gives the highest joint probability

$$\mathbf{s}^* = \arg\max_{\mathbf{s}} p(\mathbf{x}, \mathbf{s}|\theta).$$

  This can be done with the Viterbi algorithm

- Note: HMMs can be seen as a special case of probabilistic graphical models (Bishop chapter 8). In this context Baum-Welch and Viterbi are special cases of the sum-product and max-sum algorithms.

Evaluation Problem

Given an observation sequence $\mathbf{x}$ and parameters θ what is the probability $Pr(\mathbf{x}|\theta)$?

First need to compute $Pr(\mathbf{s}|\theta)$. For example, with $\mathbf{s} = s_1 s_2 s_3$:

$$\begin{aligned} Pr(\mathbf{s}|\theta) &= Pr(s_1, s_2, s_3|\theta) \\ &= Pr(s_1|\theta) Pr(s_2, s_3 \mid s_1, \theta) \\ &= Pr(s_1|\theta) Pr(s_2 \mid s_1, \theta) Pr(s_3 \mid s_2, \theta) \\ &= \pi_2 A_{21} A_{11} A_{12} \end{aligned}$$

Then compute $Pr(\mathbf{x}|\mathbf{s}, \theta)$:

$$\begin{aligned} Pr(\mathbf{x} \mid \mathbf{s}, \theta) &= Pr(x_1 x_2 x_3 \mid s_1 s_2 s_3, \theta) \\ &= Pr(x_1 \mid s_1, \theta) Pr(x_2 \mid s_2, \theta) Pr(x_3 \mid s_3, \theta) \qquad (1) \end{aligned}$$

Evaluation Problem

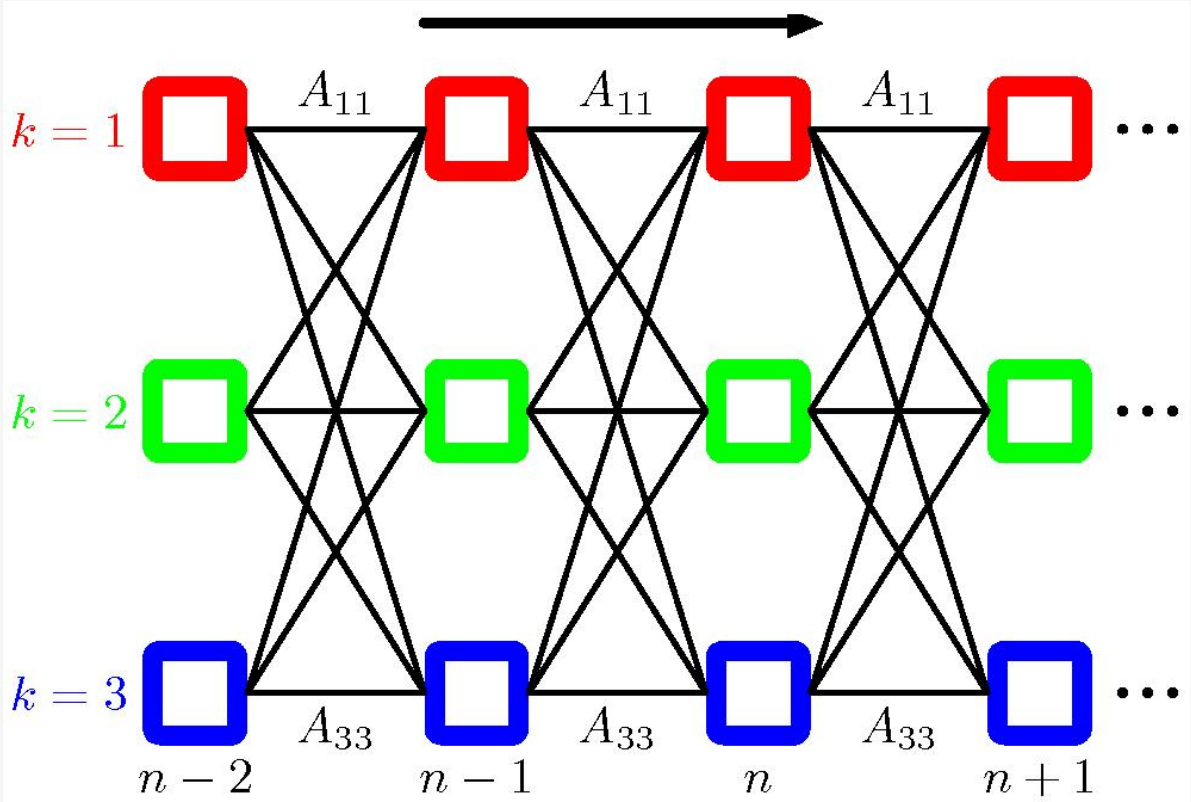
Then use sum rule for probabilities to get

$$Pr(\mathbf{x}|\theta) = \sum_{\mathbf{s}} Pr(\mathbf{s}|\theta) Pr(\mathbf{x}|\mathbf{s}, \theta) \tag{2}$$

PROBLEM: number of possible state sequences $= |\ \mathbf{s}\ |^N$

Unfold the HMM

If we unfold the state transition diagram of the above example, we obtain a lattice, or trellis, representation of the latent states. This makes it easier to understand the following derivations.



A smart solution for the Evaluation Problem

$$
\begin{aligned}
\alpha_t(i) &= Pr(x_1 x_2 \ldots x_t | s_t = i) \\
\alpha_1(i) &= Pr(x_1 | s_1 = i)\pi_i \\
\alpha_{t+1}(j) &= \sum_i A_{ij} Pr(x_{t+1} | s_t = j)\alpha_t(i)
\end{aligned}
$$

Therefore the probability of a given observation sequence ending in state s_i can be computed as follows

$$
\begin{aligned}
Pr(x_1 x_2 \ldots x_t) &= \sum_{i=1}^{N} \alpha_t(i) \\
Pr(s_t = i \mid x_1 x_2 \ldots x_t) &= \frac{\alpha_t(i)}{\sum_{j=1}^{N} \alpha_t(j)}
\end{aligned}
$$

A smart solution for the Evaluation Problem

$$
\begin{aligned}
\alpha_t(i) &= Pr(x_1 x_2 \ldots x_t | s_t = i) \\
\alpha_1(i) &= Pr(x_1 | s_1 = i)\pi_i \\
\alpha_{t+1}(j) &= \sum_i A_{ij} Pr(x_{t+1} | s_t = j)\alpha_t(i)
\end{aligned}
$$

Therefore the probability of a given observation sequence ending in state s_i can be computed as follows

$$
\begin{aligned}
Pr(x_1 x_2 \ldots x_t) &= \sum_{i=1}^{N} \alpha_t(i) \\
Pr(s_t = i \mid x_1 x_2 \ldots x_t) &= \frac{\alpha_t(i)}{\sum_{j=1}^{N} \alpha_t(j)}
\end{aligned}
$$

The Viterbi Algorithm

Given an observation sequence **x**, which is the state sequence **s** with the highest probability?

$$
\begin{aligned}
\arg\max_{\mathbf{s}} \quad & Pr(\mathbf{s}|x_1 x_2 \ldots x_T) \\
& \text{with Bayes} \\
= \quad & \arg\max_{\mathbf{s}} \frac{Pr(x_1 x_2 \ldots x_T \mid \mathbf{s}) Pr(\mathbf{s})}{Pr(x_1 x_2 \ldots x_T)} \\
= \quad & \arg\max_{\mathbf{s}} Pr(x_1 x_2 \ldots x_T \mid \mathbf{s}) Pr(\mathbf{s})
\end{aligned}
$$

Again: Dynamic programming to the rescue!

The Viterbi Algorithm

The variable $\delta_t(i)$ is the maximum probability of

- the existence of the state path $s_1 s_2 \ldots s_{t-1}$
- ending in state i
- and producing the output $Pr(x_1 x_2 \ldots x_t)$

$$\delta_t(i) = \max_{s_1 s_2 \ldots s_{t-1}} Pr(s_1 s_2 \ldots s_{t-1}, s_t = i, x_1 x_2 \ldots x_t)$$

The Viterbi Algorithm

The variable $\delta_t(i)$ is the maximum probability of

- the existence of the state path $s_1 s_2 \ldots s_{t-1}$
- ending in state i
- and producing the output $Pr(x_1 x_2 \ldots x_t)$

$$\delta_t(i) = \max_{s_1 s_2 \ldots s_{t-1}} Pr(s_1 s_2 \ldots s_{t-1}, s_t = i, x_1 x_2 \ldots x_t)$$

The Viterbi Algorithm

So for any $\delta_t(j)$ we are looking for the most probable path of length t that has as the last *two* states i and j.

But this is the most probable path (of length $t-1$) to i followed by the transition from i to j and the corresponding observation x_t. Thus, the most probable path to j has i^* as its penultimate state, with

$$i^* = \arg\max_i \delta_{t-1}(i) A_{ij} Pr(x_t|s_t = j)$$

Hence

$$\delta_t(j) = \delta_{t-1}(i^*) A_{i^*j} Pr(x_t|s_t = j)$$

The Viterbi Algorithm

So for any $\delta_t(j)$ we are looking for the most probable path of length t that has as the last *two* states i and j.

But this is the most probable path (of length $t-1$) to i followed by the transition from i to j and the corresponding observation x_t. Thus, the most probable path to j has i^* as its penultimate state, with

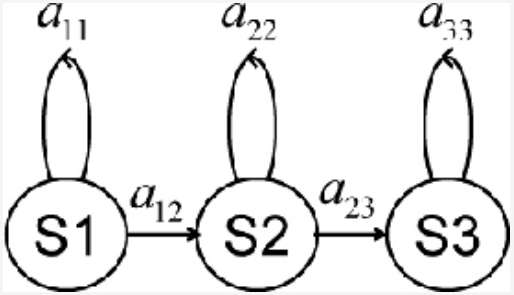
$$i^* = \arg\max_i \delta_{t-1}(i) A_{ij} Pr(x_t|s_t = j)$$

Hence

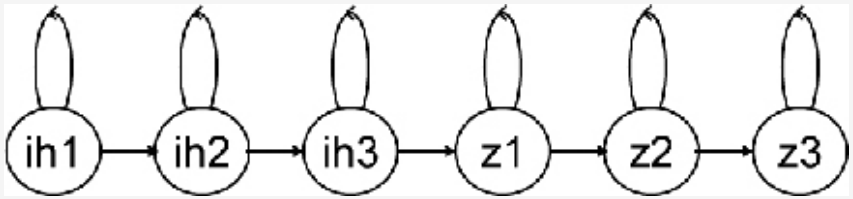
$$\delta_t(j) = \delta_{t-1}(i^*) A_{i^*j} Pr(x_t|s_t = j)$$

Sequence Labelling with HMMs

- Could define a simple HMM with one state per label
- But for most data multi-state models are needed for each label, such as this one used for a phoneme in speech recognition

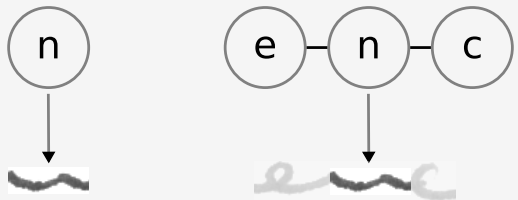


- Note that only left-to-right and self transitions are allowed. This ensures that all observation sequences generated by the label pass through similar 'stages'
- For good performance, the states should correspond to 'independent' observation segments within the label
- Can concatenate label models to get higher level structures, such as words



N-Gram Label Models

- Using separate label models assumes that the observation sequences generated by the labels are independent
- But in practice this often isn't true. e.g. in speech the pronunciation of a phoneme is influenced by those around it (co-articulation)
- Usual solution is to use n-gram label models (e.g. triphones), with the label generating the observations in the centre



- Improves performance, but also increases the number of models ($L \longrightarrow L^n$ for L labels) and amount of training data required
- Can reduce the parameter explosion by tying similar states in different n-grams

Duration Modelling

- The only way an HMM can stay in the same state is by repeatedly making self-transitions
- This means the probability of spending T timesteps in some state k decays expontentially with T

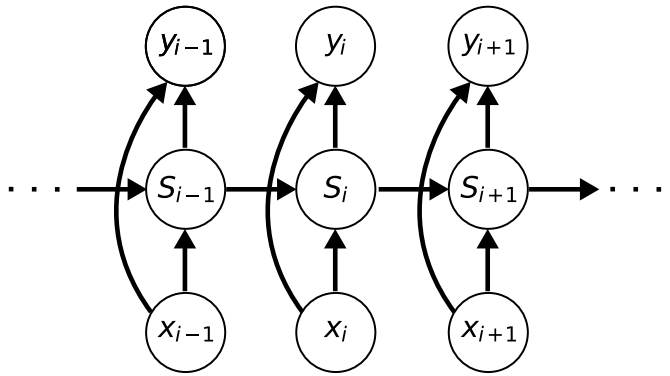
$$\Pr(T) = (A_{kk})^T(1 - A_{kk}) \propto exp(-T \ln A_{kk})$$

- However this is usually an unrealistic model of state duration
- One solution is to remove the self-transitions and model the duration probability $p(T|k)$ explicitly
- When state k is entered, a value for T is first drawn from $p(T|k)$ and T successive observations are then drawn from $p(x|k)$

Variant HMM structures

Many variant architectures can be created by modifying the basic HMM dependency structure. Three of them are described below

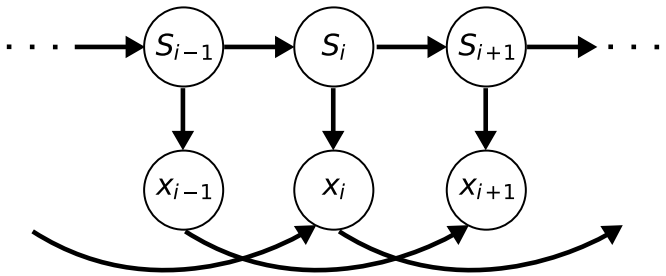
Input-output HMMs



Turn HMMs into a discriminative model by reversing the dependency between the states and the observations and introducing extra output variables

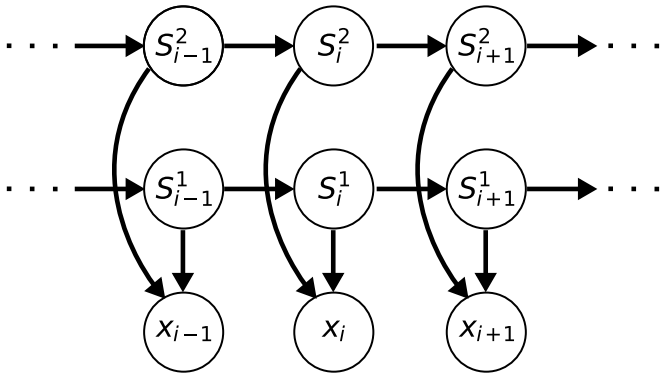
Variant HMM structures

Autoregressive HMMs



Add adding explicit dependencies between the observations to improve long-range context modelling

Factorial HMMs



Add extra chains of hidden states, thereby moving from a single-valued to a distributed architecture and increasing the memory capacity of HMMs: $O(\log N) \rightarrow O(N)$