Lecture 8: Mixture Models & Expectation-Maximisation

Matthew Caldwell

COMP0088 Introduction to Machine Learning • UCL Computer Science

Contents

8.1: Latent Variables

COMP0088 Introduction to Machine Learning · UCL Computer Science

Inferring the unmeasurable

- Often the things we want to measure are expressed only indirectly through the measurements we are able to make
- The true governing variables or states are hidden or latent, acting beneath the accessible surface of the data, behind the curtain
- Instead we have to measure some proxies and infer
- This is a fundamental principle for unsupervised learning: if the latent variables were explicit we wouldn't need to infer them

  → And indeed for supervised learning: in that case we are handed explicit labels, but it's not clear how well they correspond to the actual latent structure of the data, and we have to learn some correspondence

- This principle is implicit in PCA
  - → The reduced explanatory dimensions represent more “genuine” axes of variability to which the measured dimensions are related by some latent correlation structure
- And pretty much explicit in clustering
  - → Spatial distribution of features is dependent on some underlying class differences between the samples in each cluster
- In this week’s topics, the latent variables are made more explicit still
  - → Though we’re still, of course, approximating them with models

Missing data

- Hidden state and missing observations are kinda the same thing
- In all cases, the problem would be easy (or at least easier) if we had full disclosure, but the unknowns make it difficult
- The same general process can be applied to both problems
- …but we'll only actually deal with hidden state here

8.2: Gaussian Mixtures

COMP0088 Introduction to Machine Learning · UCL Computer Science

Clustering redux

- Underlying class specifies spatial distribution of features
- We were unspecific about what that distribution was, just that it subdivided feature space by distance
  - → Assumed symmetrical in all directions
  - → Imposed hard boundaries
- We could pose this more generally as drawing from a set of distributions conditioned on the hidden class membership
- Another way of looking at it is that each class has a single “true” feature value plus its own separate noise model

Mixture of Gaussians

- If we assume each class is normally distributed with some mean feature vector mu and covariance sigma, then the observed data distribution can be expressed as a weighted sum of *k* individual Gaussians, where the weights correspond to the probabilities of drawing from each class

$$P(\mathbf{x}) = \sum_{i}^{k} \alpha_i \phi(\mathbf{x}; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$$

  - → This kind of weighted sum can express arbitrarily complex distributions given large enough *k*
  - → As with *k*-means, we'll assume *k* is a hyperparameter and choosing it is beyond the scope of the immediate problem

Mixture of Gaussians

- We'll assume that class membership is distributed multinomially with some class membership probabilities α — these correspond to the weights in the weighted sum
- Of course we don't know α, μ or Σ, nor do we know the latent class of any of the data points
- Our problem is to figure all those things out

- If we knew the class memberships, the problem would be relatively easy, as we could just estimate single Gaussians for each of the known subpopulations, and estimate the α as the fraction of total samples in each class:

$$\boldsymbol{\mu}_j = \frac{\sum_i \mathbf{1}(z_i = j)\mathbf{x}_i}{\sum_i \mathbf{1}(z_i = j)}$$

$$\boldsymbol{\Sigma}_j = \frac{\sum_i \mathbf{1}(z_i = j)(\mathbf{x}_i - \boldsymbol{\mu}_j)(\mathbf{x}_i - \boldsymbol{\mu}_j)^\mathsf{T}}{\sum_i \mathbf{1}(z_i = j)}$$

$$\alpha_j = \frac{\sum_i \mathbf{1}(z_i = j)}{n}$$

- Not knowing that, we wind up with unsolvable likelihood expressions
- The proximate cause of the intractability is that we wind up taking logs of a sum as we marginalise the unknowns
- But the ultimate cause is that the problem recurses: the solution to each part depends on already having solved the other, in an infinite regress

When in doubt, guess

- As with *k*-means, we'll choose to break the impasse by starting with a guess
- And then iteratively refining partial solutions that can be posed in terms of previous partial solutions
- Unlike *k*-means, we're going to make soft predictions rather than hard
  - → ie, for each sample we'll predict a distribution — a vector of probabilities — over the *k* classes rather than a single class assignment
  - → these are aka responsibilities: how much we believe the sample belongs to the class

EM for GMMs

- E-step: estimate responsibilities

$$\gamma_{i,j} = \frac{P(\mathbf{x}_i|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)\alpha_j}{\sum_l P(\mathbf{x}_i|\boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)\alpha_l}$$

- M-step: choose MLE params given responsibilities

$$\boldsymbol{\mu}_j = \frac{\sum_i \gamma_{i,j}\mathbf{x}_i}{\sum_i \gamma_{i,j}}$$

$$\boldsymbol{\Sigma}_j = \frac{\sum_i \gamma_{i,j}(\mathbf{x}_i - \boldsymbol{\mu}_j)(\mathbf{x}_i - \boldsymbol{\mu}_j)^\mathsf{T}}{\sum_i \gamma_{i,j}}$$

$$\alpha_j = \frac{\sum_i \gamma_{i,j}}{n}$$

- GMMs are relatively easy to implement and understand and are widely used
  - → Both for soft clustering (which is basically what we've discussed here) and for approximating complex distributions
  - → ie, when you don't care about class assignment but want to model a spatially complex population that's asymmetric or disconnected or whatever
- Like *k*-means (and many other E-M methods) they are susceptible to getting stuck in local maxima and consequently sensitive to initialisation
- It's common to run multiple times with different initial values and pick the best outcome, and also to run *k*-means (which is faster) as an initialiser (the sklearn GaussianMixture class supports doing both of these automatically)

8.3: Expectation-Maximisation

COMP0088 Introduction to Machine Learning · UCL Computer Science

EM in general

- Both GMM fitting and k-means are instances of a more general optimisation approach known as expectation-maximisation
- The general pattern is that an optimisation is difficult or impossible as stated but would be much easier if some additional information were available
  - → Of course, all problems would be easy if *some* additional information were available — to wit, the answers — so this is not a very precise or necessarily helpful statement
  - → It may be more useful to think of the hard problem as being an incomplete analogue of the easier problem in some straightforward way
- This can be applied to missing data problems, but we will focus on the case of latent state

General form

- Consider the general case model with data **X**, latent variables **Z** and parameters **θ**
- Assume we know the form of the joint probability distribution:

$$P(\mathbf{X}, \mathbf{Z}; \boldsymbol{\theta}) = P(\mathbf{X}|\mathbf{Z}; \boldsymbol{\theta})P(\mathbf{Z}|\boldsymbol{\theta})$$

→ Eg, in the GMM case the first component is Gaussian and the second multinomial

→ (These are expressed slightly hand-wavingly in terms of single point probabilities, but we can calculate for distributions by marginalising, eg over **Z** — which we shortly will)

- This means we also know the form of the likelihood function

$$L(\boldsymbol{\theta}) = P(\mathbf{X}, \mathbf{Z}; \boldsymbol{\theta}) = P(\mathbf{X}|\mathbf{Z}; \boldsymbol{\theta})P(\mathbf{Z}|\boldsymbol{\theta})$$

- And the form of the posterior distribution for the latent variables

$$P(\mathbf{Z}|\mathbf{X}; \boldsymbol{\theta}) = \frac{P(\mathbf{X}|\mathbf{Z}; \boldsymbol{\theta})P(\mathbf{Z}|\boldsymbol{\theta})}{P(\mathbf{X}; \boldsymbol{\theta})}$$

- If we knew **Z**, we could define a likelihood (or log likelihood) and attempt to maximise it to estimate **θ**
- If we knew **θ** we could estimate the distribution of **Z**
- Unfortunately, we only know **X**

- As in earlier cases, we resolve this by just guessing one thing, using that to estimate the other, using that estimate to refine the guess, and so on
- We can formalise this in terms of two iterable steps

E-step

- Estimate a posterior distribution, Q, for the latent state $\mathbf{Z}$, based on previously estimated/guessed $\boldsymbol{\theta}$:

$$Q^{(t)}(\mathbf{Z}) = P(\mathbf{Z}|\mathbf{X}; \boldsymbol{\theta}^{(t-1)})$$

- Formulate the log likelihood of $\boldsymbol{\theta}$ using this distribution:

$$l^{(t)}(\boldsymbol{\theta}) = \sum_{\mathbf{Z}} Q^{(t)}(\mathbf{Z}) \log P(\mathbf{X}, \mathbf{Z}; \boldsymbol{\theta}^{(t-1)})$$

→ Note that the log is inside the summation here

M-step

- Maximise the provisional log likelihood to find improved parameter values:

$$\boldsymbol{\theta}^{(t)} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, l^{(t)}(\boldsymbol{\theta})$$

Do it again and again and again

- These two steps are repeated alternately until we get a solution we’re happy with
  - → Obviously we need to define happiness, but that’s a whole other kettle of worms

Initialisation

- We've defined a recurrence relation with no starting point
- Somehow we need to come up with an initial value $\boldsymbol{\theta}^{(0)}$
- As noted for both k-means and GMMs, this can turn out to be pretty important
- EM optimisations are almost always non-concave and can converge to local maxima
- How you deal with this is very problem dependent, but a crude "try a bunch of different start values" strategy can be effective

Convergence

- It may risk converging to local maxima, but EM is at least guaranteed to converge
- The expected log likelihood function constructed in the E-step will not make things worse
  - → Which may seem like a pretty weak promise, but look around you
  - → There's a proof of this which these lectures are too narrow to contain, but it's readily available in the reading list books

- Actually translating any particular model formulation into the computations stipulated by the EM equations of slides 21 & 22 can be quite tricky, though there are a few relatively straightforward cases, as for GMMs
- In practice, it may be more useful to think of EM as a pattern that is regularly useful, rather than as a recipe for how to solve any particular problem
- Although there turn out to be a number of important and useful EM applications, it's perhaps telling that these were often developed independently as clever ways to solve a specific problem — formulated in somewhat different terms — and only retrospectively recognised as doing the same thing

8.4: Hidden Markov Models

COMP0088 Introduction to Machine Learning · UCL Computer Science

Changing states

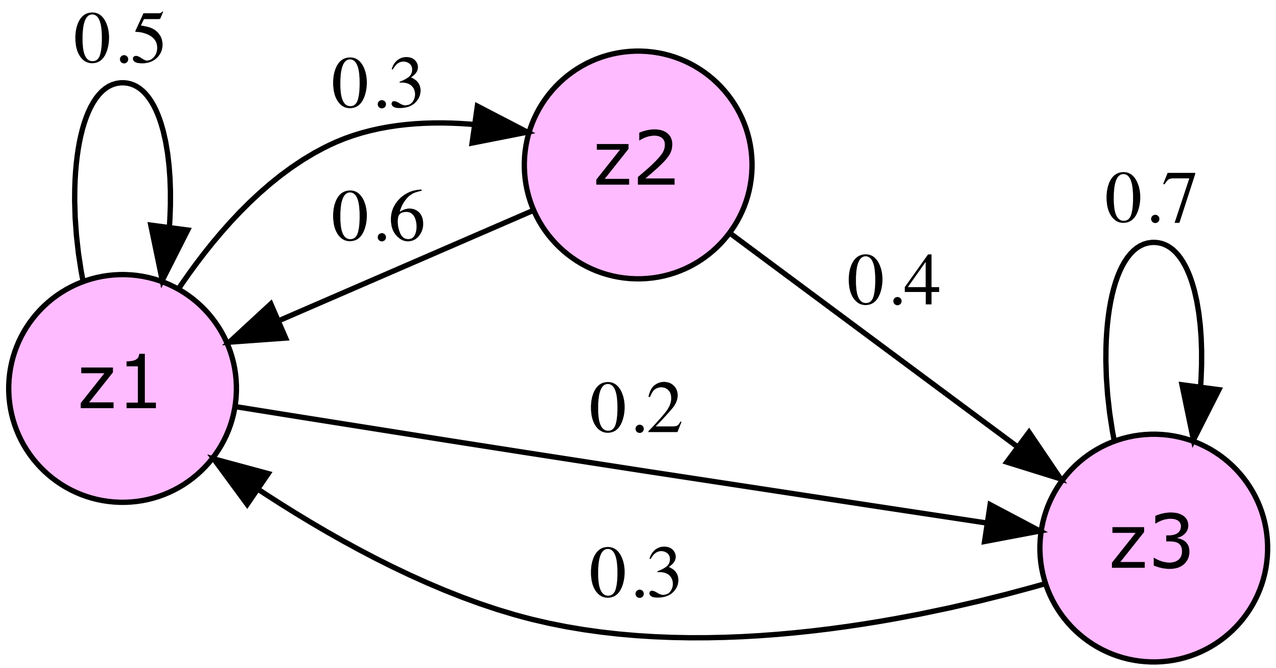
- In the previous discussion we've considered only static models, where all the samples drawn are independent
- We might also want to model sequential data in which our samples are drawn from the same model over time
  - → As with a GMM, observations are conditioned on the hidden state, but that hidden state is subject to change (cf. RNNs)
- If the states at different times are still independent, then there is no difference from multiple parallel draws from, eg, a GMM
- In general we might expect state changes to be conditional on (at least) the current state

Markov property

- If the state changes depend only on the current state, the process is said to be Markovian (or to have the Markov property) — often expressed as being memoryless
- Past and future states are conditionally independent, given the present state
  - → It doesn't matter how we got to the present state, just that we're there
- At each time step, the current state fully determines the transition probabilities to all other states
- If the transition probabilities from each state are unchanging through time, the process is time homogenous
  - → We'll consider only time homogenous Markov models here

Graph representation

- We can represent the latent behaviour as a graph where the nodes are states and the edges represent transitions:



Matrix representation

- Alternatively, we can represent it with a transition matrix

  → The element at *i*,*j* represents the probability of going from state *i* to state *j*

state at *t+1*

state at *t*	z1	z2	z3
z1	0.5	0.3	0.2
z2	0.6	0	0.4
z3	0.3	0	0.7

Model variables & parameters

- *T* sequential observations $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T \in \mathbb{R}^d$
- Conditioned on unobserved states $z_1, z_2, ..., z_T \in \{1, 2, ..., k\}$
- Via emission probability distribution $P(\mathbf{x}|z; \boldsymbol{\theta})$ parameterised by $\boldsymbol{\theta}$
- Initial state probability vector $\boldsymbol{\pi} = \{\pi_1, \pi_2, ..., \pi_k\}$
- Transition matrix **A**, where element $a_{i,j}$ is probability of going from state *i* at any timestep *t* to state *j* at timestep *t*+1
  - → All rows must sum to 1
  - → $a_{i,i}$ is probability of staying in state *i*
- We'll assume **A** doesn't change — process is time homogenous

Kinds of HMM problem

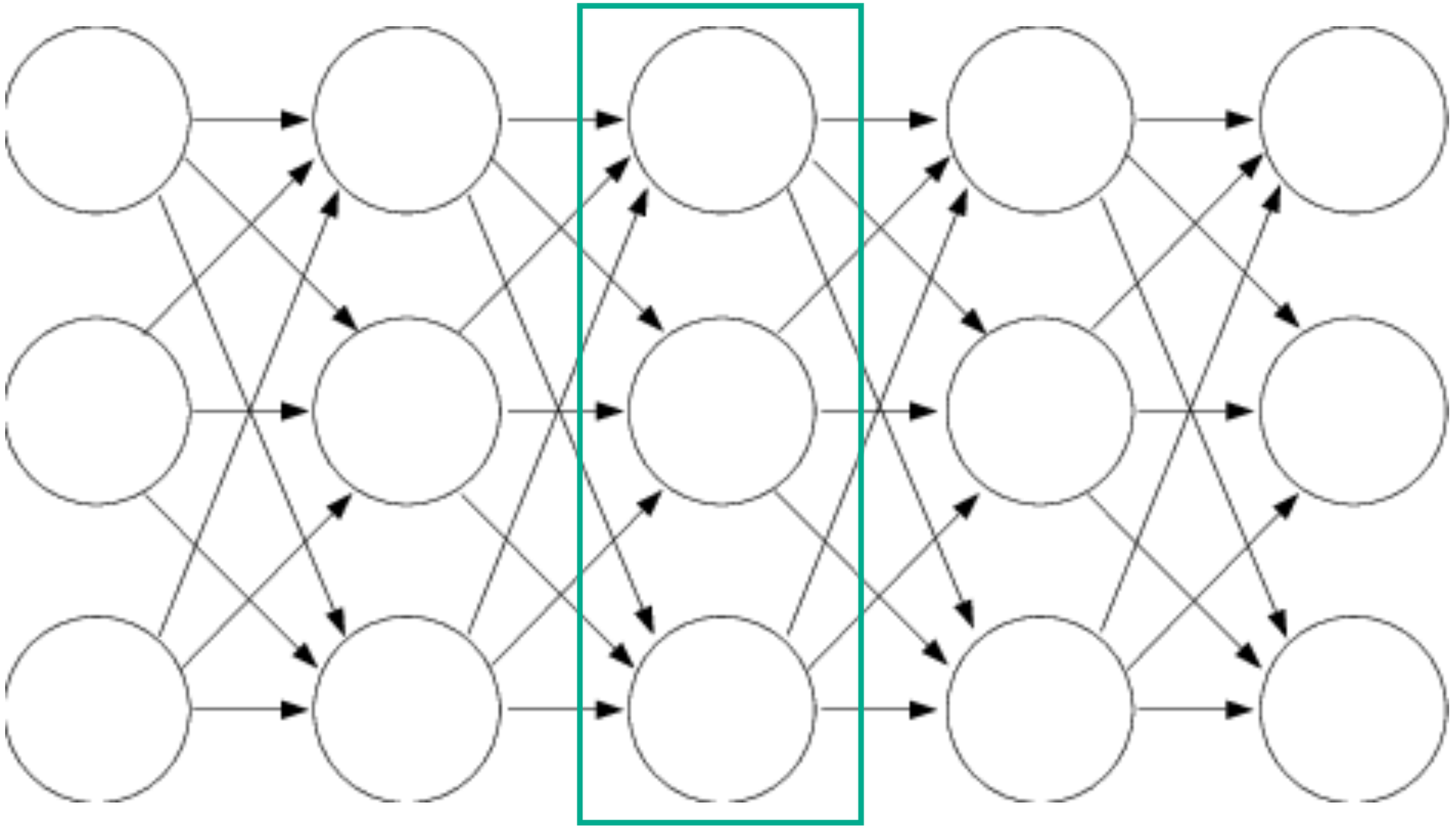
- Probability of an observation or sequence (forward, forward-backward)
  - → given (**X**, **π**, **A**, **θ**), estimate P(**X**)
- Likelihood of parameters (forward-backward)
  - → given (**X**, **π**, **A**, **θ**), estimate L(**π**, **A**, **θ**)
- Parameter fitting (Baum-Welch = EM)
  - → given (maybe lots of) **X**, estimate (**π**, **A**, **θ**)
- Hidden state inference (Viterbi)
  - → given (**X**, **π**, **A**, **θ**), estimate (at least some of) **Z**

Conditional independence

- Processing all possible routes through tree of sequences is intractable due to exponential growth
- But practical algorithms are made possible because the Markov property means the future is conditionally independent of the past, given the present
  - → x_t depends only on z_t
  - → z_t depends only on z_{t-1}
  - → one set of probabilities per timestep: it doesn't matter how you got there

$$P(\mathbf{x}_{1:t}, z_t) = P(\mathbf{x}_t|z_t) \sum_{z_{t-1}} P(z_t|z_{t-1}) P(\mathbf{x}_{1:t-1}, z_{t-1})$$

Trellis diagram



Forward algorithm

- Probabilities/likelihoods at each step based on previous step, transition probabilities and conditional probability of observations
- Start at beginning and iterate
- Considers only past evidence (“filtering”)

$$\alpha_t(j) = P(\mathbf{x}_{1:t}, z_t = j) = \sum_i^k \alpha_{t-1}(i)\, a_{i,j}\, P(\mathbf{x}_t; \boldsymbol{\theta}_j)$$

Forward-backward algorithm

- Same forward pass as forward algorithm
- But also a complementary backward pass calculating probability of having been in starting state at each point given subsequent observations
- Start at end and iterate backwards
- Considers evidence from future
- Combine both to get better estimate (“smoothing”)

$$\beta_t(j) = P(\mathbf{x}_{t:T}, z_t = j) = \sum_i^k \beta_{t+1}(i)\, a_{i,j}\, P(\mathbf{x}_{t+1}; \boldsymbol{\theta}_j)$$

Inferring state sequence: Viterbi algorithm

- Similar to forward algorithm, but computes only most likely route to each state at each step (ie max rather than sum), and records previous state that led to that max

$$\alpha_t(j) = \max_i \ \alpha_{t-1}(i)\, a_{i,j}\, P(\mathbf{x}_t; \boldsymbol{\theta}_j)$$

$$\zeta_t(j) = \operatorname*{argmax}_i \ \alpha_{t-1}(i)\, a_{i,j}\, P(\mathbf{x}_t; \boldsymbol{\theta}_j)$$

- At the end the likeliest final state is determined and follows the backpointers from there to the start to get the overall likeliest hidden state sequence

Parameter fitting: Baum-Welch

- EM algorithm using forward-backward to estimate likelihoods
- Start and transition probabilities are estimated from the trellis of smoothed states — fraction of time you were softly attributed to state *i* at time *t* that you were softly attributed to state *j* at *t*+1
- Emission probability parameters estimated according to the appropriate distribution — eg, for Gaussian outputs the MLE is essentially the same as for the means and covariances of a GMM, for categorical outputs much the same as for the class memberships α for the GMM

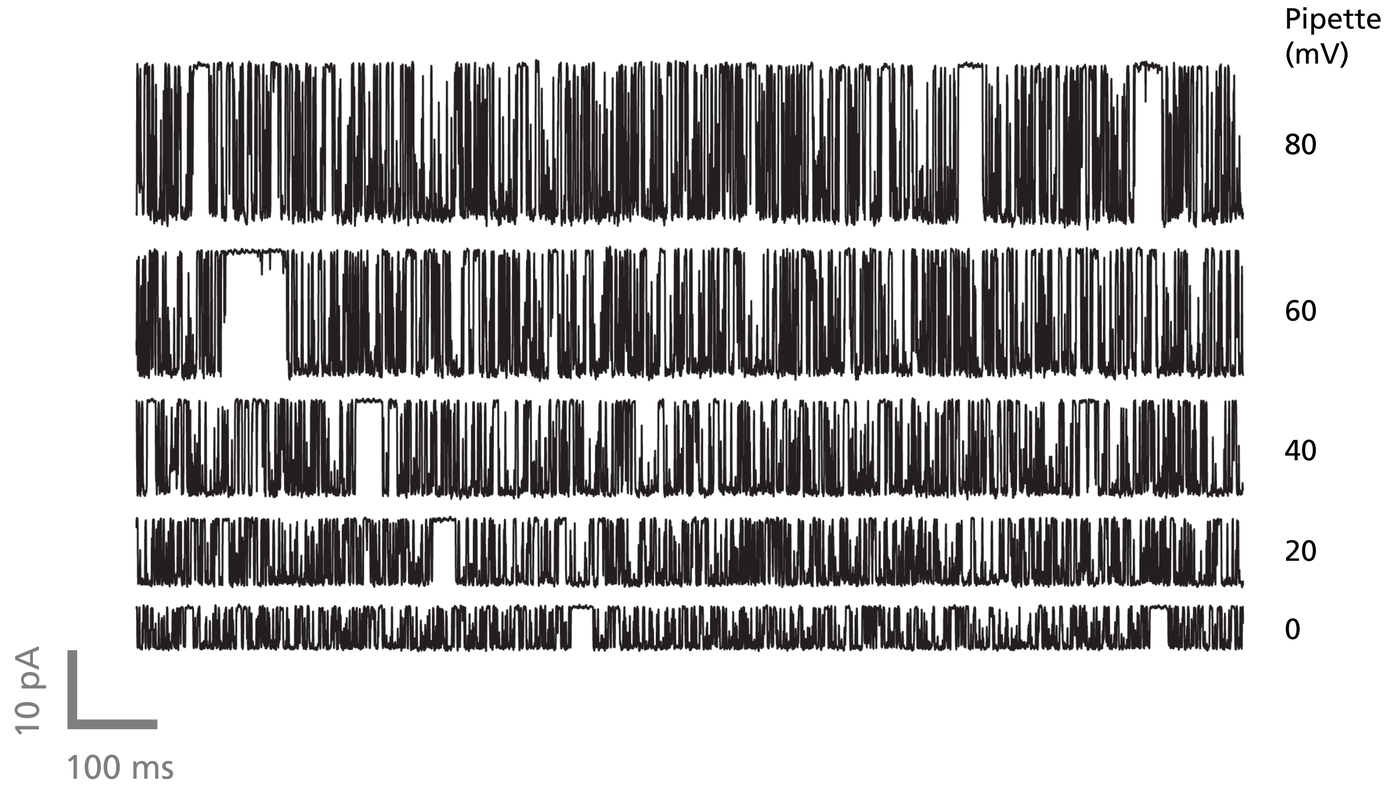
HMM applications

- HMMs originally came to prominence in the context of speech recognition and NLP, though have been overtaken somewhat by RNNs
- Still widely used in bioinformatics for things like gene sequencing
- An example from my own past: estimating ion channel kinetics from single channel recording data

Single channel recording

- As mentioned in week 5, neuronal processing depends on minutely detailed control of ion flows in and out of cells
- Key components in this process are proteins known as ion channels, which traverse cell membranes and allow selective passage of different ion species
- These exist in several different conformations, flicking between them stochastically (but conditioned on factors such as neurotransmitter binding or membrane potential)
- Some conformations allow ions to pass, others don’t

- The current flows through individual channels are incredibly tiny (of the order of trillionths of amps), but can be measured experimentally using very sensitive amplifiers, leading to data like this:



- These recordings show the behaviour of single molecules in real time!
- HMMs can be used to estimate the kinetics of individual channels from such recordings
- This can be important in the context of pharmacology: drugs cause changes in channel kinetics, which can be how they act to affect things like mental state

Wrapping up

- Unsupervised learning requires the imposing some structure on the problem
- Doing so defines the sort of answers we are looking for — it necessarily prejudices the results to some extent
- But it is also what makes it possible to extract useful information from unlabelled data
- Latent variables are an important way of formulating our problem requirements in such a way as to be able to interrogate the data
- The EM process of iterative refinement from an initial guess enables latent variable models to be estimated