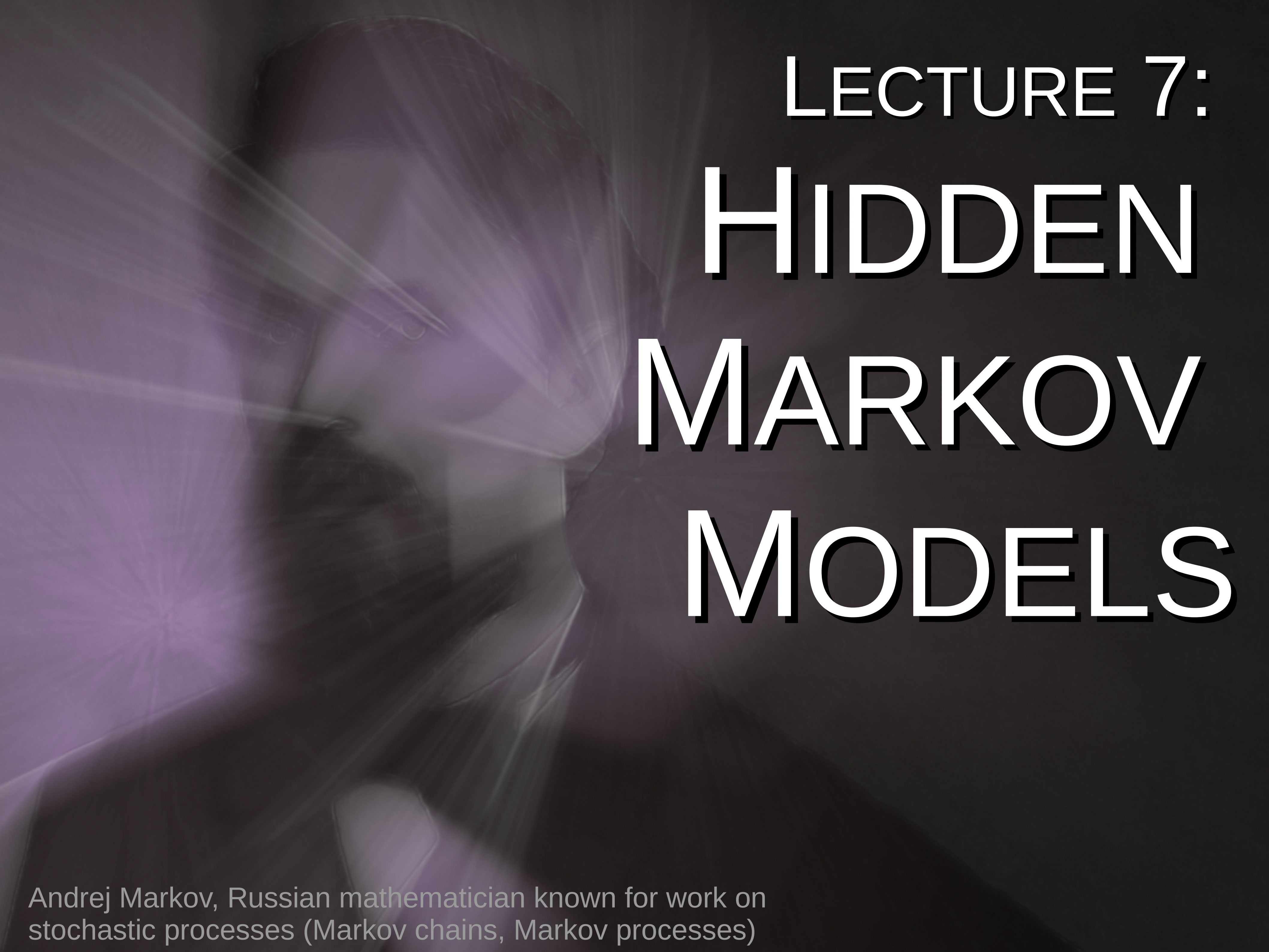
# Lecture 7: Hidden Markov Models



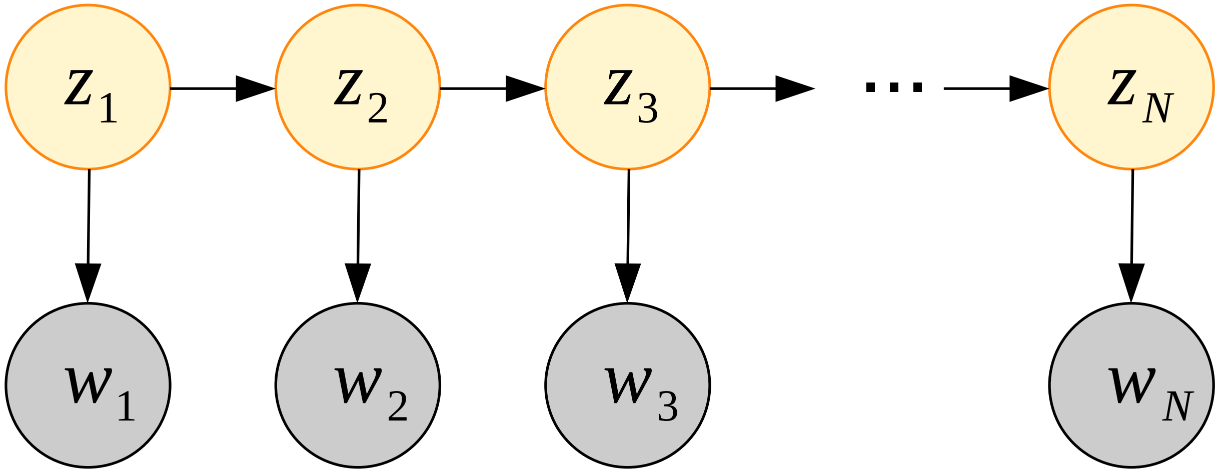
Andrej Markov, Russian mathematician known for work on stochastic processes (Markov chains, Markov processes)

# Hidden Markov models

- N-grams represent Markov model type relationships between observed words in a sequence
- Topic models represent how observed words can arise from underlying latent variables, but the ones we discussed did not consider sequence information of the words
- The two ideas can be combined: **hidden Markov models** (HMMs) can represent observations that arise as a sequence, from latent variables that change as a sequence

# Hidden Markov models

- Graphical representation of the idea:



- Observed words do not directly depend on each other: they are indepent given the latent state, but the latent states depend on each other
- HMMs can be used as a model on their own but they are often also used as a part of a bigger model

# Hidden Markov models

- Hidden Markov models are defined by:
  - A set of **hidden states** *S*, indexed by $k=1,\ldots,N_S$
  - A set of possible **observed values** (output alphabet).
  - **Transition probability distributions**: $\theta_{j|i}=p\left(z_{t+1}=j|z_t=i\right)$ probability of all possible next states j, given current states i.
  - **Initial state distribution**: probability $\pi_k=p\left(z_1=k\right)$ to start from each possible state $k=1,\ldots,N_S$
  - **Emission distributions**: given each possible current state i, probability $\beta_i(\boldsymbol{x})$ to emit each possible observation $\boldsymbol{x}$ .
- HMMs are used in many domains. In audio processing (e.g. speech recognition) they are often used with Gaussian emission distributions to emit momentary audio features
- We will use discrete outputs: the set of possible observations is the word vocabulary, i.e. each ***x*** is a word from the vocabulary.

# Hidden Markov models

- Generating a new sequence of observations (e.g. a sequence of observed words) from a Hidden Markov model is easy:

1. Initially t=1. Sample the initial hidden state $z_1$ according the multinomial initial state distribution probabilities $\{\pi_k = p(z_1 = k)\}_{k=1,\dots,N_S}$
2. Let's say the value was $z_t = i$ . Then sample the current observation $\mathbf{x}_t$ from the distribution $\beta_i$ (probabilities $\beta_i(\mathbf{x})$ for each possible observation ***x***).
3. Sample the next hidden state $z_{t+1}$ according to the transition probabilities. If the previous state was $z_t = i$, use the transition probabilities $\{\theta_{j|i} = p(z_{t+1} = j | z_t = i)\}_{j=1,\dots,N_S}$
4. Increase t, and return to step 2.
5. Repeat steps 2-4 as long as needed.

# Hidden Markov models

- Learning in Hidden Markov models is done by three algorithms:
  - The **Forward-Backward algorithm**: computes the probability of a sequence of observations, given the current parameter values
  - The **Viterbi algorithm**: computes the most likely sequence of hidden states that could have generated a sequence of observations
  - The **Baum-Welch algorithm**: an algorithm for maximum likelihood optimization of the parameters of the HMM. It is a special case of the Expectation-Maximization algorithm

# Forward-Backward algorithm

- Suppose we have a sequence of observations $\boldsymbol{x}_1,\ldots,\boldsymbol{x}_N$. In our case they will simply be words (vocabulary indices), but we will use the general notation as much as possible.
- We want to compute the probability of the observation sequence under the HMM model.
- The joint probability of the observation sequence and the underlying latent sequence of states $z_1,\ldots,z_N$ is

$$p([\boldsymbol{x}_1,\ldots,\boldsymbol{x}_N],[z_1,\ldots,z_N])=(p(\boldsymbol{x}_1|z_1)\,p(z_1))\prod_{t=2}^{N}(p(\boldsymbol{x}_t|z_t)\,p(z_t|z_{t-1}))$$

$$=(\beta_{z_1}(\boldsymbol{x}_1)\pi_{z_1})\prod_{t=2}^{N}(\beta_{z_t}(\boldsymbol{x}_t)\theta_{z_t|z_{t-1}})$$

# Forward-Backward algorithm

- Consider a **forward term**: the probability of observations up to time t, and that the state at time t is k:

$$p_{\text{Forward}}(t,k)=p([\mathbf{x}_1,\ldots,\mathbf{x}_t],z_t=k)$$

- Consider also a **backward term**: the probability of observations after time t, given that the state at time t is k:

$$p_{\text{Backward}}(t,k)=p([\mathbf{x}_{t+1},\ldots,\mathbf{x}_N]\|z_t=k)$$

- The forward probabilities can be defined inductively:

$$p_{\text{Forward}}(1,k) \;=\; p(\mathbf{x}_1,z_1=k) \;=\; p(\mathbf{x}_1|z_1=k)\,p(z_1=k) \;=\; \beta_k(\mathbf{x}_1)\pi_k$$

$$p_{\text{Forward}}(t,k)=p([\mathbf{x}_1,\ldots,\mathbf{x}_t],z_t=k)$$

$$=\sum_{j=1}^{N_S} p([\mathbf{x}_1,\ldots,\mathbf{x}_{t-1}],z_{t-1}=j,\mathbf{x}_t,z_t=k)$$

$$=\sum_{j=1}^{N_S} p([\mathbf{x}_1,\ldots,\mathbf{x}_{t-1}],z_{t-1}=j)\,p(z_t=k|z_{t-1}=j)\,p(\mathbf{x}_t|z_t=k)$$

$$=\sum_{j=1}^{N_S} p_{\text{Forward}}(t-1,j)\theta_{k|j}\beta_k(\mathbf{x}_t)$$

# Forward-Backward algorithm

- The backward probabilities can also be defined inductively:

$p_{\text{Backward}}(N,k)=1$ (because $p_{\text{Backward}}(t,k)=p([\mathbf{x}_{t+1},\ldots,\mathbf{x}_N]|z_t=k)$ but there are no observations after N so there is nothing to generate when t is N.)

$$p_{\text{Backward}}(t,k)=p([\mathbf{x}_{t+1},\ldots,\mathbf{x}_N]|z_t=k)$$

$$=\sum_{m=1}^{N_S} p(\mathbf{x}_{t+1},[\mathbf{x}_{t+2},\ldots,\mathbf{x}_N],z_{t+1}=m|z_t=k)$$

depends only on $z_{t+1}$

$$=\sum_{m=1}^{N_S} p(\mathbf{x}_{t+1}|[\mathbf{x}_{t+2},\ldots,\mathbf{x}_N],z_{t+1}=m,z_t=k)\cdot$$

$$p([\mathbf{x}_{t+2},\ldots,\mathbf{x}_N]|z_{t+1}=m,z_t=k)\,p(z_{t+1}=m|z_t=k)$$

depends only on $z_{t+1}$

$$=\sum_{m=1}^{N_S} p(\mathbf{x}_{t+1}|z_{t+1}=m)\,p(z_{t+1}=m|z_t=k)\,p([\mathbf{x}_{t+2},\ldots,\mathbf{x}_N]|z_{t+1}=m)$$

$$=\sum_{m=1}^{N_S} \beta_m(\mathbf{x}_{t+1})\,\theta_{m|k}\,p_{\text{Backward}}(t+1,m)$$

# Forward-Backward algorithm

- The probability of the observation sequence can then be written using the forward and backward probabilities as

$$p(\mathbf{x}_1,\ldots,\mathbf{x}_N)=p([\mathbf{x}_1,\ldots,\mathbf{x}_t],[\mathbf{x}_{t+1},\ldots,\mathbf{x}_N])$$

for any choice of t

$$=\sum_{k=1}^{N_S} p([\mathbf{x}_1,\ldots,\mathbf{x}_t],z_t=k,[\mathbf{x}_{t+1},\ldots,\mathbf{x}_N])$$

$$=\sum_{k=1}^{N_S} p([\mathbf{x}_1,\ldots,\mathbf{x}_t],z_t=k)\,p([\mathbf{x}_{t+1},\ldots,\mathbf{x}_N]\,|\,[\mathbf{x}_1,\ldots,\mathbf{x}_t],z_t=k)$$

depends only on $z_t$

$$=\sum_{k=1}^{N_S} p([\mathbf{x}_1,\ldots,\mathbf{x}_t],z_t=k)\,p([\mathbf{x}_{t+1},\ldots,\mathbf{x}_N]\,|\,z_t=k)$$

$$=\sum_{k=1}^{N_S} p_{\text{Forward}}(t,k)\,p_{\text{Backward}}(t,k)$$

- In particular, if we choose t=N, then $p_{\text{Backward}}(N,k)=1$ and

$$p(\mathbf{x}_1,\ldots,\mathbf{x}_N)=\sum_{k=1}^{N_S} p_{\text{Forward}}(N,k)$$

# Viterbi algorithm

- Next, suppose we have observed the sequence $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ and want to find the sequence of states $z_1, \ldots, z_N$ that has the maximal probability $p(z_1, \ldots, z_N | \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)$ given the observations.
- Finding the maximum a posteriori sequence is important in some applications such as speech recognition where the states have linguistic meaning, but they can be also used just to inspect what kinds of explanations the HMM finds for observed sequences.
- We want to find $z_1^*, \ldots, z_N^* = argmax_{z_1, \ldots, z_N} p(z_1, \ldots, z_N, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)$
- Define $\mathrm{Best}(t, k) = max_{z_1, \ldots, z_{t-1}} p(z_1, \ldots, z_{t-1}, z_t = k, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_t)$ which is the highest probability of a state sequence up to time t ending in state k, and the corresponding observations up to t.

# Viterbi algorithm

- We can compute it recursively:

$$\text{Best}(t,k) = max_{z_1,\dots,z_{t-1}} p(z_1,\dots,z_{t-1},z_t=k,\boldsymbol{x}_1,\dots,\boldsymbol{x}_t)$$

$$= max_{z_1,\dots,z_{t-2}} max_{j\in[1,\dots,N_S]} p(z_1,\dots,z_{t-2},z_{t-1}=j,z_t=k,\boldsymbol{x}_1,\dots,\boldsymbol{x}_t)$$

$$= max_{j\in[1,\dots,N_S]} max_{z_1,\dots,z_{t-2}} p(z_1,\dots,z_{t-2},z_{t-1}=j,\boldsymbol{x}_1,\dots,\boldsymbol{x}_{t-1},z_t=k,\boldsymbol{x}_t)$$

if t > 1

$$= max_{j\in[1,\dots,N_S]} [ max_{z_1,\dots,z_{t-2}} p(z_1,\dots,z_{t-2},z_{t-1}=j,\boldsymbol{x}_1,\dots,\boldsymbol{x}_{t-1}) p(z_t=k|z_{t-1}=j) ] p(\boldsymbol{x}_t|z_t=k)$$

$$= max_{j\in[1,\dots,N_S]} [\text{Best}(t-1,j)\theta_{k|j}] \beta_k(\boldsymbol{x}_t)$$

if t = 1

$$= max_{j\in[1,\dots,N_S]} [p(z_1=k)] p(\boldsymbol{x}_1|z_1=k) = \pi_k \beta_k(\boldsymbol{x}_1)$$

- The maximum over the whole sequence is then

$$max_{z_1,\dots,z_N} p(z_1,\dots,z_N,\boldsymbol{x}_1,\dots,\boldsymbol{x}_N) = max_{k\in[1,\dots,N_S]} \text{Best}(N,k)$$

- The sequence of states that yields the above maximum value can also be computed recursively

# Viterbi algorithm

- Let's use a structure "BestZ" which keeps track of what index maximized each maximum operation.
- Initialize by

$$\text{Best}(1,k)=\pi_k\,\beta_k(\boldsymbol{x}_1)$$

$$\text{BestZ}(1,k)=0$$

- Then for t > 1 we set

$$\text{Best}(t,k)=max_{j\in[1,\ldots,N_S]}[\text{Best}(t-1,j)\theta_{k|j}]\beta_k(\boldsymbol{x}_t)$$

$$\text{BestZ}(t,k)=argmax_{j\in[1,\ldots,N_S]}[\text{Best}(t-1,j)\theta_{k|j}]$$

- We recover the final state of the highest-probability sequence:

$$z_N^*=argmax_{k\in[1,\ldots,N_S]}\text{Best}(N,k)$$

- We then backtrack through the maximums in t=N down to 2:

$$z_{t-1}^*=argmax_{j\in[1,\ldots,N_S]}[\text{Best}(t-1,j)\theta_{k=z_t^*|j}]=\text{BestZ}(t,z_t^*)$$

# Baum-Welch algorithm

- The remaining problem is how to fit the parameters: transition probabilities and emission probabilities, for a given observation sequence, by maximum likelihood. The Baum-Welch algorithm does this as a special case of the Expectation-Maximimization algorithm for HMMs.

- The Baum-Welch algorithm makes use of the probabilities we computed previously:

$$p_{\text{Forward}}(t,j) = p([\boldsymbol{x}_1,\ldots,\boldsymbol{x}_t], z_t = j)$$

$$p_{\text{Backward}}(t,j) = p([\boldsymbol{x}_{t+1},\ldots,\boldsymbol{x}_N] | z_t = j)$$

- The E-step then computes two kinds of conditional probabilities

$$\kappa(t,j) = p(z_t = j | \boldsymbol{x}_1,\ldots,\boldsymbol{x}_N) = \frac{p(z_t = j, \boldsymbol{x}_1,\ldots,\boldsymbol{x}_N)}{p(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_N)}$$

$$= \frac{p([\boldsymbol{x}_1,\ldots,\boldsymbol{x}_t], z_t = j)\, p([\boldsymbol{x}_{t+1},\ldots,\boldsymbol{x}_N] | z_t = j)}{p(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_N)} = \frac{p_{\text{Forward}}(t,j)\, p_{\text{Backward}}(t,j)}{\sum_{l=1}^{N_S} p_{\text{Forward}}(N,l)}$$

# Baum-Welch algorithm

- and

$$\nu(t,j,k)=p(z_t=j,z_{t+1}=k|\boldsymbol{x}_1,\ldots,\boldsymbol{x}_N)=\frac{p(z_t=j,z_{t+1}=k,\boldsymbol{x}_1,\ldots,\boldsymbol{x}_N)}{p(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_N)}$$

$$=\frac{p([\boldsymbol{x}_1,\ldots,\boldsymbol{x}_t],z_t=j)\,p(z_{t+1}=k|z_t=j)\,p(\boldsymbol{x}_{t+1}|z_{t+1})\,p([\boldsymbol{x}_{t+2},\ldots,\boldsymbol{x}_N]|z_{t+1}=k)}{p(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_N)}$$

$$=\frac{p_{\text{Forward}}(t,j)\,\theta_{k|j}\,\beta_k(\boldsymbol{x}_{t+1})\,p_{\text{Backward}}(t+1,k)}{\sum_{l=1}^{N_S}p_{\text{Forward}}(N,l)}$$

# Baum-Welch algorithm

- The M-step then updates the parameters: initial state probabilities, state transition probabilities, and any parameters that control the emission probabilities
- The initial state probabilities are updated as

$$\pi_k := p(z_1 = k | \mathbf{x}_1, \ldots, \mathbf{x}_N) = \kappa(1, k)$$

- State transition probabilities are updated as

$$\theta_{k|j} := p(z_{\text{next}} = k | z_{\text{current}} = j, \mathbf{x}_1, \ldots, \mathbf{x}_N) = \frac{p(z_{\text{next}} = k, z_{\text{current}} = j, \mathbf{x}_1, \ldots, \mathbf{x}_N)}{p(z_{\text{current}} = j, \mathbf{x}_1, \ldots, \mathbf{x}_N)}$$

$$= \frac{\sum_{t=1}^{N-1} p(z_{t+1} = k, z_t = j | \mathbf{x}_1, \ldots, \mathbf{x}_N)}{\sum_{t=1}^{N-1} p(z_t = j | \mathbf{x}_1, \ldots, \mathbf{x}_N)} = \frac{\sum_{t=1}^{N-1} \nu(t, j, k)}{\sum_{t=1}^{N-1} \kappa(t, j)}$$

- The update of the emission probabilities depends on what type of observations are used: we describe the case when the observations are discrete

# Baum-Welch algorithm

- For discrete observations (such as observed words) the parameters are simply, for each state, the probabilities of each observation: $\beta_j(v) = p(v|z=j)$ for each word v=1,...,V in the vocabulary.
- They are updated as

$$\beta_j(v) := \frac{p(\mathbf{x}=v, z=j|\mathbf{x}_1, \ldots, \mathbf{x}_N)}{p(z=j|\mathbf{x}_1, \ldots, \mathbf{x}_N)} = \frac{\sum_{t=1}^{N} p(\mathbf{x}_t=v, z_t=j|\mathbf{x}_1, \ldots, \mathbf{x}_N)}{\sum_{t=1}^{N} p(z_t=j|\mathbf{x}_1, \ldots, \mathbf{x}_N)}$$

$$= \frac{\sum_{t=1}^{N} \delta(\mathbf{x}_t=v) p(z_t=j|\mathbf{x}_1, \ldots, \mathbf{x}_N)}{\sum_{t=1}^{N} p(z_t=j|\mathbf{x}_1, \ldots, \mathbf{x}_N)} = \frac{\sum_{t=1}^{N} \delta(\mathbf{x}_t, v) \kappa(t, j)}{\sum_{t=1}^{N} \kappa(t, j)}$$

where $\delta(\mathbf{x}_t, v) = 1$ if the word at position t is v and zero otherwise.

# Baum-Welch algorithm

- The previous equations assume a single observation sequence. When there are multiple (M > 1) sequences , we compute the forward and backward probabilities and E-step conditional probabilities for each sequence m=1,...,M, denoted as

$$p_{\text{Forward}}^{(m)}(t,j)=p([\boldsymbol{x}_1^{(m)},\ldots,\boldsymbol{x}_t^{(m)}],z_t^{(m)}=j)$$

$$p_{\text{Backward}}^{(m)}(t,j)=p([\boldsymbol{x}_{t+1}^{(m)},\ldots,\boldsymbol{x}_{N^{(m)}}^{(m)}]|z_t^{(m)}=j)$$

computed by running the Forward-Backward algorithm for each sequence

$$\kappa^{(m)}(t,j)=\frac{p_{\text{Forward}}^{(m)}(t,j)\,p_{\text{Backward}}^{(m)}(t,j)}{\sum_{l=1}^{N_S}p_{\text{Forward}}^{(m)}(N,l)}$$

$$\nu^{(m)}(t,j,k)=\frac{p_{\text{Forward}}^{(m)}(t,j)\,\theta_{k|j}\,\beta_k(\boldsymbol{x}_{t+1}^{(m)})\,p_{\text{Backward}}^{(m)}(t+1,k)}{\sum_{l=1}^{N_S}p_{\text{Forward}}^{(m)}(N,l)}$$

# Baum-Welch algorithm

- Parameters are then updated as

$$\pi_k := \frac{1}{M} \sum_{m=1}^{M} \kappa^{(m)}(1,k)$$

$$\theta_{k|j} := \frac{\sum_{m=1}^{M} \sum_{t=1}^{N^{(m)}-1} \nu^{(m)}(t,j,k)}{\sum_{m=1}^{M} \sum_{t=1}^{N^{(m)}-1} \kappa^{(m)}(t,j)}$$

$$\beta_j(v) := \frac{\sum_{m=1}^{M} \sum_{t=1}^{N^{(m)}} \delta(\boldsymbol{x}_t^{(m)}, v)\, \kappa^{(m)}(t,j)}{\sum_{m=1}^{M} \sum_{t=1}^{N^{(m)}} \kappa^{(m)}(t,j)}$$

# HMMs and topic models

- In a topic model like Latent Dirichlet Allocation, words were generated from underlying topics, as a bag of words
- For a single sequence of words, this can be seen as a special case of a HMM: when the transition probabilities are constrained to be independent of the previous state (i.e. same as the initial probabilities), the "transition" just becomes an independent sampling of states for each word.
- However, in e.g. LDA, each document had its own overall distribution of topics, from which topics were sampled: in the HMM there is one set of initial and transition probabilities
- Solution: learn separate initial probabilities for each sequence, but a single set of emission probabilities for all sequences. Learn the intitial probabilities as

$$\theta_{k|j}^{(m)} = \pi_k^{(m)} := p\left(z=k \,|\, \boldsymbol{x}_1^{(m)}, \ldots, \boldsymbol{x}_{N^{(m)}}^{(m)}\right) = \frac{1}{N^{(m)}} \sum_{t=1}^{N^{(m)}} \kappa^{(m)}(t,k)$$

and use them in the E-step computations of sequence m

# HMMs and part-of-speech tagging

- In purely unsupervised HMMs the meaning of the different latent states is not known: one has to analyze the resulting model (transition probabilities and emission probabilities) to understand what meaning each state might have.
- The hope is that if the HMM is trained well, the found states will turn out to have syntactic or semantic meaning.
- A HMM can be used for **part-of-speech tagging** of sentences: The part of speech assigned to a word can depend on its context: e.g. "r**ound** down" (verb), "a **round** cylinder" (adjective), "a **round** of golf" (noun).
  - The hope is that trained latent states HMM states will correspond to parts of speech.
- One could classify each HMM state into a part of speech after an unsupervised training stage, by looking at which words have been emitted from that state, and what actual parts-of-speech they had: then assign the state into the majority class.

# HMMs and part-of-speech tagging

- HMMs can also involve **supervised training** from labeled sequences where the part of speech is known for every word: suppose for some sequences m we have observed both the observation sequence $\mathbf{x}_1^{(m)},\ldots,\mathbf{x}_{N^{(m)}}^{(m)}$ and the part-of-speech sequence $z_1^{(m)},\ldots,z_{N^{(m)}}^{(m)}$ .
- Then for those sequences the E-step probabilities are simple:

$$\kappa^{(m)}(t,j)=p\left(z_t^{(m)}=j|\mathbf{x}_1^{(m)},\ldots,\mathbf{x}_{N^{(m)}}^{(m)}\right)=\delta\left(z_t^{(m)},j\right)$$

$$\nu^{(m)}(t,j,k)=p\left(z_t^{(m)}=j,z_{t+1}^{(m)}=k|\mathbf{x}_1^{(m)},\ldots,\mathbf{x}_{N^{(m)}}^{(m)}\right)=\delta\left(z_t^{(m)},j\right)\delta\left(z_{t+1}^{(m)},k\right)$$

- Both kinds of sequences (labeled and unlabeled) can be used in training.
- Unsupervised sequences use the normal E-step, and the M-step is the same.
- After training, the Viterbi algorithm is used for part-of-speech tagging of new word sequences.

# HMMs in Python

- NLTK includes a tagging module based on HMMs: `nltk.tag.hmm.HiddenMarkovModelTagger`
- Example of use:

```
import nltk

# NLTK includes an example corpus of 3914 tagged sentences

mytaggedsentences=nltk.corpus.treebank.tagged_sents()

myhmmtrainer=nltk.tag.hmm.HiddenMarkovModelTrainer()

# Training using labeled and possibly also unlabeled data
# unfortunately the unsupervised training seems to be buggy!

mytrainedhmm = myhmmtrainer.train( \
    labeled_sequences=mytaggedsentences,unlabeled_sequences=None)

# Alternative: training using "labelled" data only

mytrainedhmm = myhmmtrainer.train_supervised( \
    labelled_sequences=mytaggedsentences)

# Tag a sentence

mytrainedhmm.tag(("The hope is that trained HMM states will " + \
     "correspond to parts of speech.").split())
```

# HMMs in Python

- There is another HMM library HMMlearn, used like this:

```
pip install hmmlearn
import numpy, hmmlearn, hmmlearn.hmm
# HMMlearn expects the data to be provided
# as a (nsamples,1) 2D-array, where the 2nd dimension has just
# one element, containing list of indices into a vocabulary,
# all documents concatenated together, and separately a
# list of lenghts of the individual documents.
# Create concatenated index list from previously
# crawled and processed documents
concatenated_data=[]
documentlengths=[]
for k in range(len(myindices_in_unifiedvocabulary)):
    concatenated_data.extend(myindices_in_unifiedvocabulary[k])
    documentlengths.append(len(myindices_in_unifiedvocabulary[k]))
concatenated_data=numpy.matrix(concatenated_data).T
# Fit the model
myhmm = hmmlearn.hmm.MultinomialHMM(n_components=10, n_iter=100,
verbose=True)
myhmm_fitted=myhmm.fit(concatenated_data,lengths=documentlengths)
# Inspect start, transition, and emission probabilities
myhmm_fitted.startprob_
myhmm_fitted.emissionprob_
myhmm_fitted.transmat_
```

# In brief: unsupervised document classification

- **Document classification** can be done using unsupervised generative models. Suppose you have categories c=1,...,C, and a set of training data (documents belonging to the category) for each of the categories.
  - For each category, train a generative model for the data of that category: for example a n-gram model, or a topic model, or a HMM.
  - Since the models are generative models, they can calculate the likelihood (or log-likelihood) of a new document.
  - Given a new document, calculate its log-likelihood based on each trained model. Assign the document to the category whose model gave the highest likelihood.
- Example use cases: Spam versus non-spam detection, authorship attribution (categories = different possible authors)