

Expectation Maximization for Gaussian Mixture Models and Hidden Markov Models

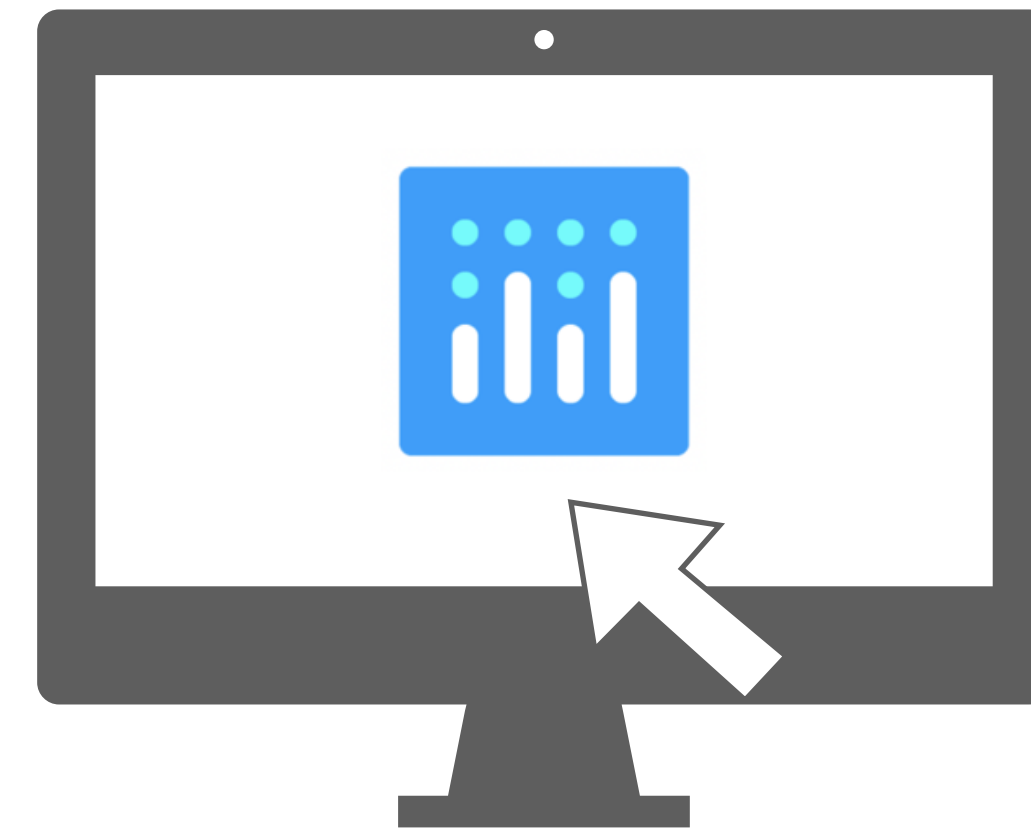
Applications to speech and other examples

EE605 Statistical Sequence Processing



Juan Pablo Zuluaga, Mael Fabien - 07 May 2020

About this work



[https://github.com/
maelfabien/EM_GMM_HMM](https://github.com/maelfabien/EM_GMM_HMM)

```
pip install -r requirements.txt  
python app.py  
http://127.0.0.1:8050/
```

Juan Pablo Zuluaga, Mael Fabien - 07 May 2020

Overview

Part I: EM and GMMs

1. Modeling distributions
2. Reminder on GMMs
3. Motivation for EM
4. EM for GMMs
5. Hard/Viterbi EM
6. Limits of EM
7. Applications of EM for GMM

Overview

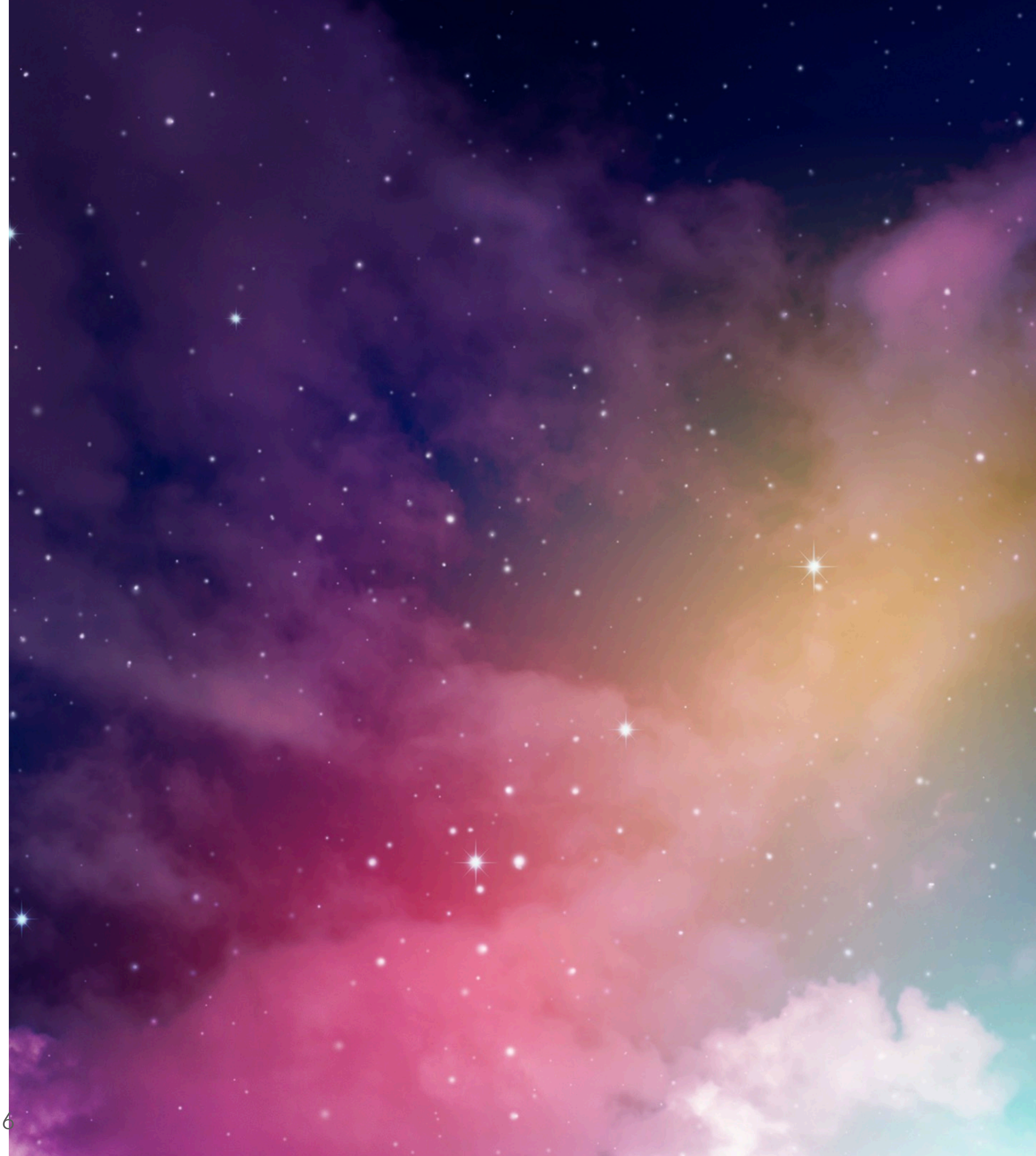
Part II: EM and HMMs

1. Introduction to HMMs
2. HMM training with forward-backward algorithm (Baum Welch)
3. Viterbi Training
4. HMM-GMM training
5. HMM applications

Part I: EM and GMMs

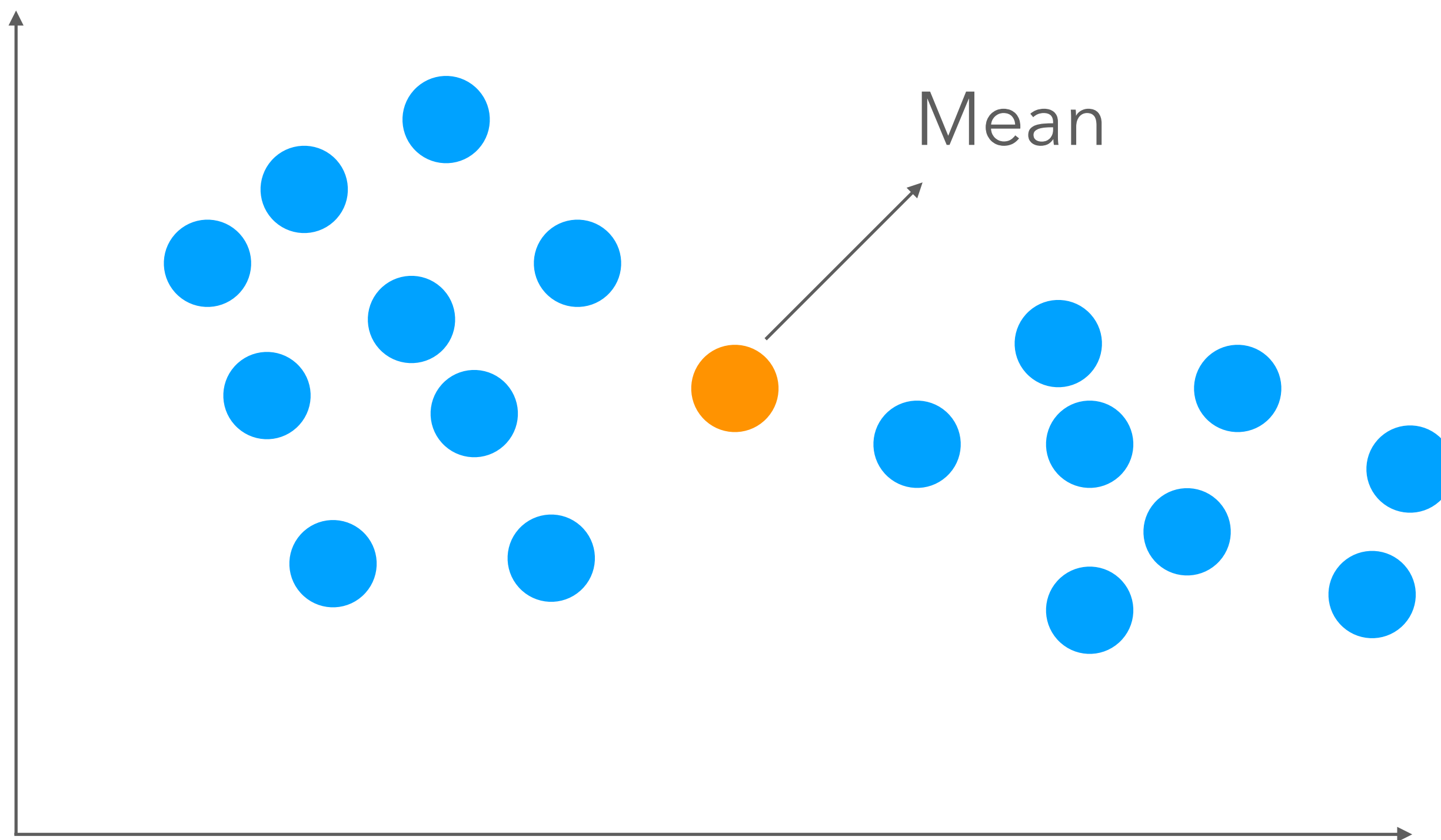
Theory and examples

I. Modeling distributions



I. Modeling distributions with k-Means and GMMs

- Why do we need GMMs?



The data obviously belongs to 2 different clusters, meaning that 2 different sources generate these data. If we fit a single Gaussian, we might end up with a mean not reflecting any of the clusters.

I. Modeling distributions with k-Means and GMMs

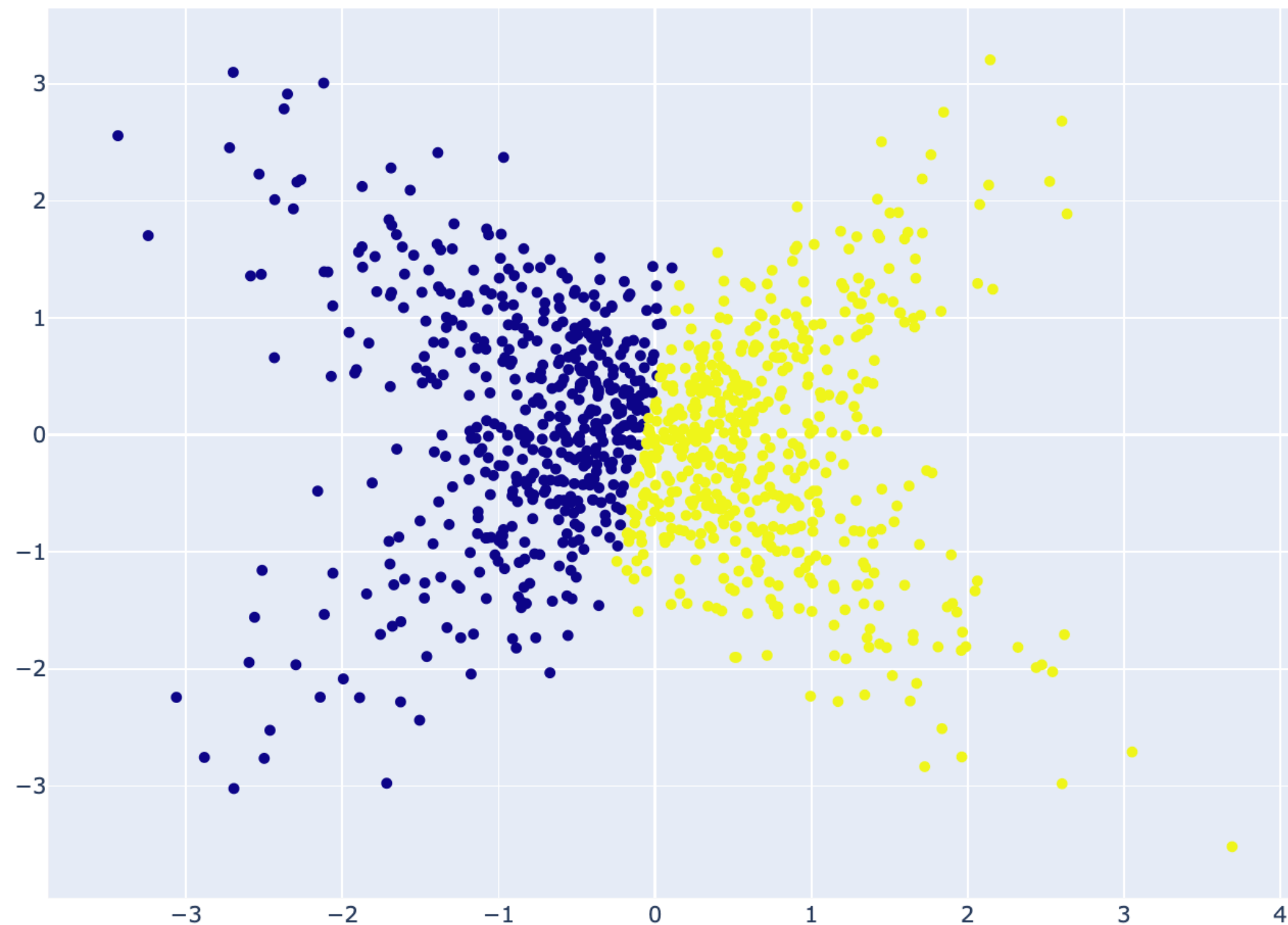
- Why are GMMs a popular choice for modeling distributions? And how does it compare to k-Means? k-Means can be seen as a special case of GMMs.

k-Means	GMM
Clusters are defined by their means	Clusters are defined by their means and their variance, modeled as Gaussians
Limitations if clusters are overlapping	Works if clusters are overlapping
Uses Euclidean distance to the mean	Uses the probability of X belonging to a cluster (generative)

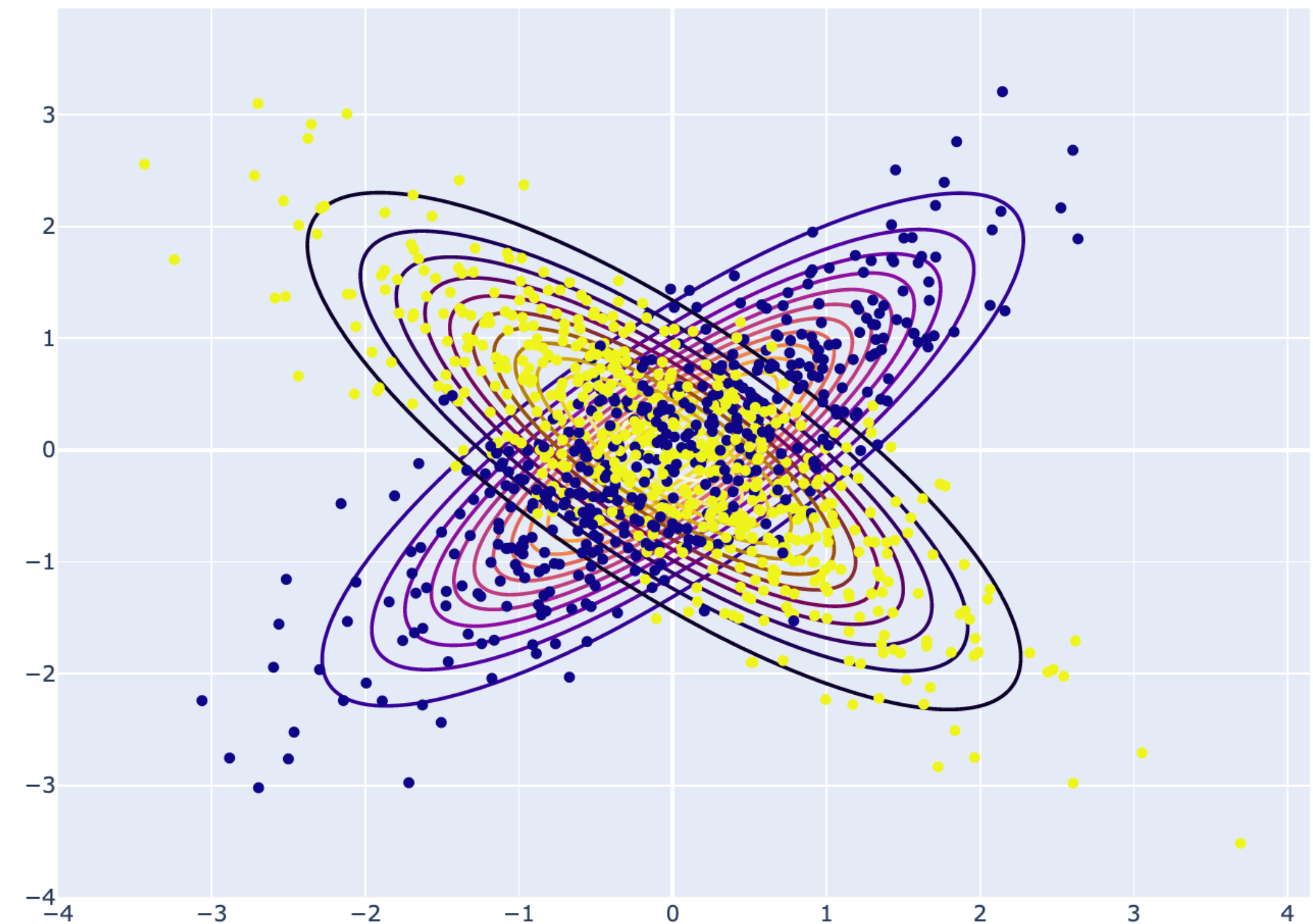
I. Modeling distributions with k-Means and GMMs

- Why are GMMs a popular choice for modeling distributions? And how does it compare to k-Means?

K-Means



GMMs



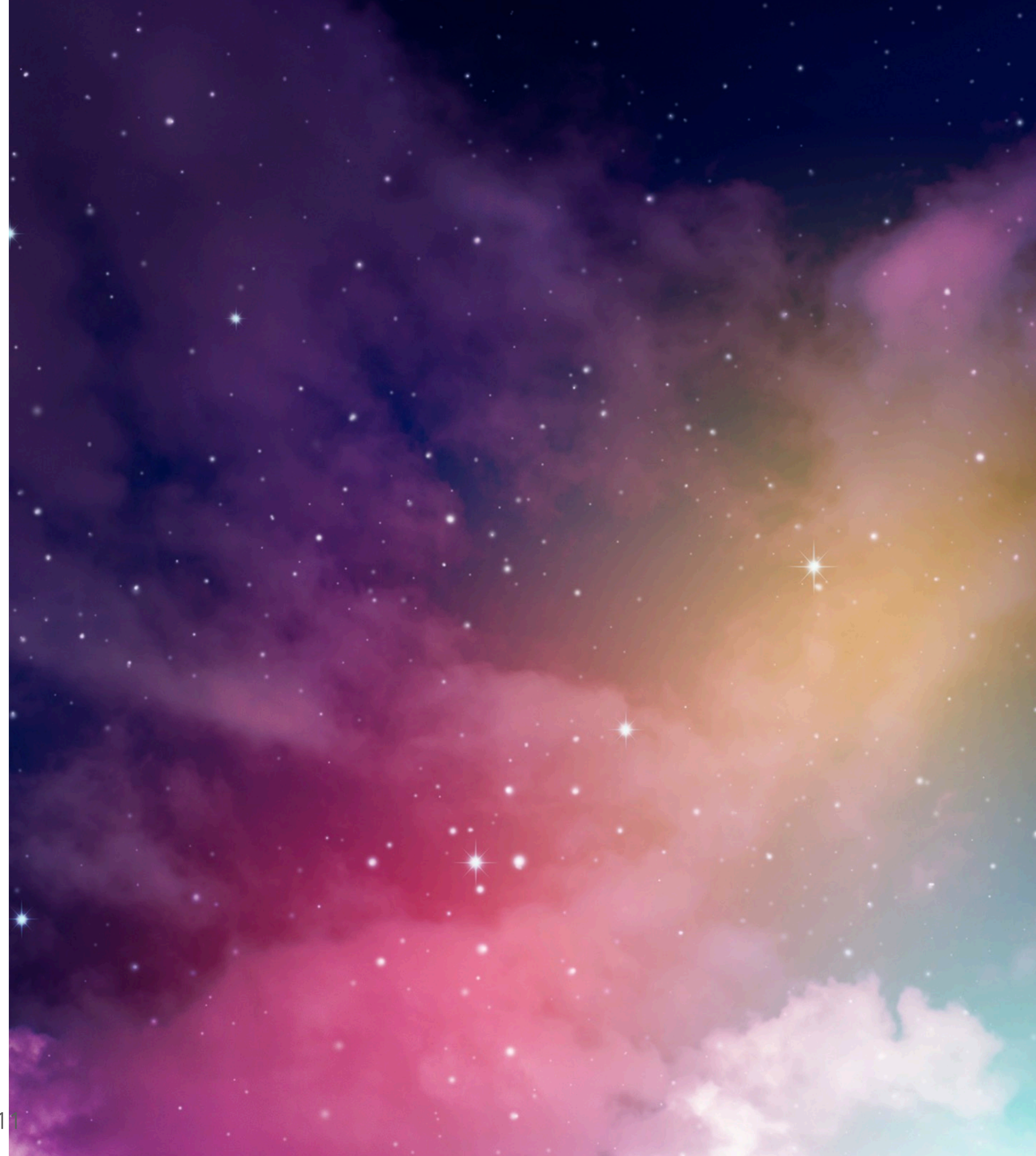
I. Modeling distributions with k-Means and GMMs

Moreover, GMMs, since generative models, have some useful properties, such as:

- Deriving the probability that observations come from a cluster
- Evaluating the similarity between training and testing sets

The k-Means algorithm is in fact a special case of GMM with (hard) EM, where each component is generated by $\mathcal{N}(\mu, I)$, I being the identity matrix

II. Reminder on Gaussian Mixture Models



II. Reminder on Gaussian Mixture Models

- A GMM is a weighted sum of M components Gaussian densities. A density of a Gaussian can be defined as:

$$P(x \mid \lambda) = \sum_{k=1}^M w_k \mathcal{N}(x \mid \mu_k, \Sigma_k)$$

M Components

Weights

Gaussian density

$$\sum_{k=1}^M w_k = 1 \quad \mathcal{N}(x \mid \mu_k, \Sigma_k) = \frac{1}{(2\pi)^{\frac{D}{2}} |\Sigma_k|^{\frac{1}{2}}} \exp^{-\frac{1}{2}(x-\mu_k)^T \Sigma_k^{-1} (x-\mu_k)}$$

II. Reminder on Gaussian Mixture Models

```
from scipy.stats import multivariate_normal

def make_data(n_data, means, covariances, weights):

    n_clusters, n_features = means.shape
    list_clusters = []

    data = np.zeros((n_data, n_features))
    for i in range(n_data):

        k = np.random.choice(n_clusters, size = 1, p = weights)[0]
        list_clusters.append(k)

        x = np.random.multivariate_normal(means[k], covariances[k])
        data[i] = x

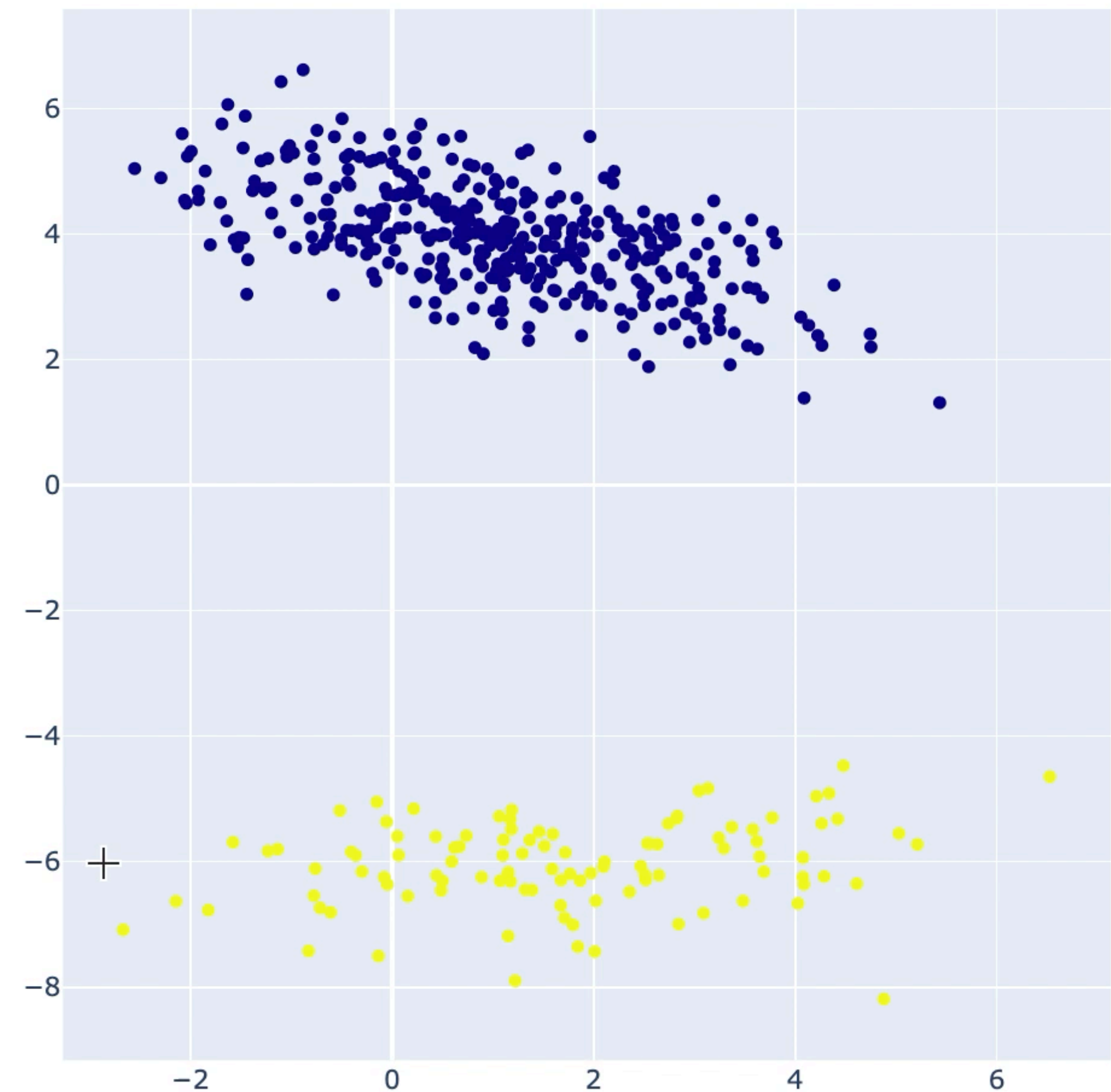
    return data, list_clusters
```

II. Reminder on Gaussian Mixture Models

Parameters

Generating data from GMMs

Number of components



II. Reminder on Gaussian Mixture Models

- The parameters of the GMM are therefore : $\lambda = (w_k, \mu_k, \Sigma_k)$, $k = 1, 2, 3, \dots, M$

How do we solve GMM? Start by solving a ***single gaussian***...

- We apply a Maximum Likelihood Estimation (MLE) to find the parameters:

$$L(\theta \mid X) = \prod_{i=1}^N P(x_i \mid \theta) = \prod_{i=1}^N \frac{1}{\sqrt{(2\pi\sigma^2)}} \exp \frac{-(x_i - \mu)^2}{2\sigma^2} \quad \theta = (\mu, \sigma)$$

$$\theta^\star = \operatorname{argmax}_\theta L(\theta \mid X)$$

II. Reminder on Gaussian Mixture Models

- For convenience, we maximize the log-likelihood since:

$$\operatorname{argmax}_{\theta} L(\theta \mid X) = \operatorname{argmax}_{\theta} \log L(\theta \mid X)$$

- Set the partial derivatives for both parameters to 0:

$$\frac{d}{d\mu} \log L(\theta \mid X) = 0 \quad \cdots \longrightarrow \quad \mu_{MLE} = \frac{1}{N} \sum_{n=1}^N x_n$$

$$\frac{d}{d\sigma} \log L(\theta \mid X) = 0 \quad \cdots \longrightarrow \quad \sigma_{MLE}^2 = \frac{1}{N} \sum_{n=1}^N (x_n - \mu)^2$$

II. Reminder on Gaussian Mixture Models

- To find the MLE of GMM parameters, we need to re-define the likelihood:

$$L(\theta \mid X_1, \dots, X_n) = \prod_{i=1}^N \sum_{k=1}^M w_k \mathcal{N}(x_i; \mu_k; \sigma_k^2)$$

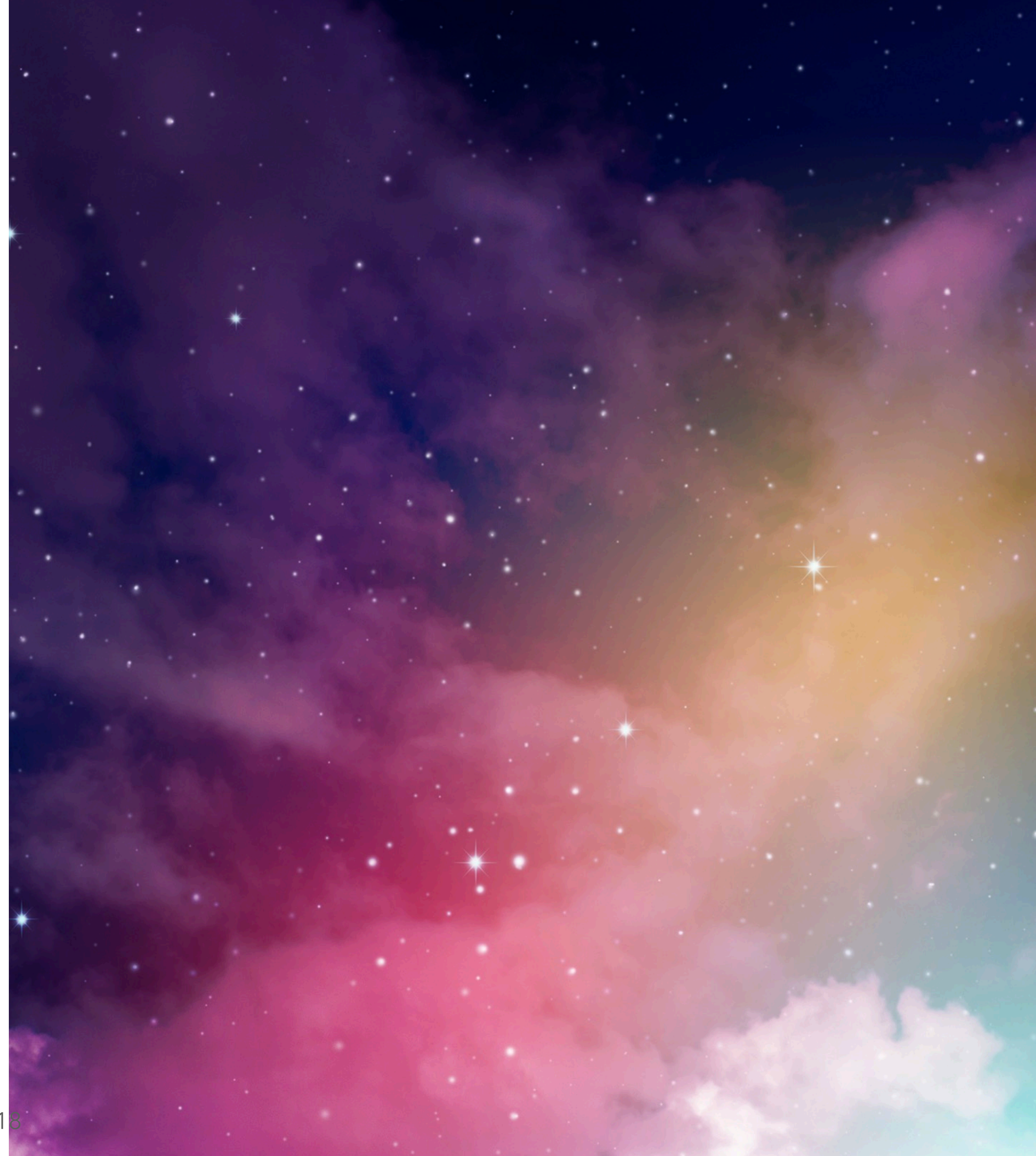
All observations

All components

Component weights

Gaussians

III. Motivation for Expectation Maximization



III. Motivation for EM

- The log-likelihood becomes:

$$l(\theta) = \log L(\theta \mid X_1, \dots, X_n) = \sum_{i=1}^N \log \left(\sum_{k=1}^M w_k \mathcal{N}(x_i, \mu_k, \sigma_k^2) \right)$$

- We now must solve over the M Gaussian components. If we set the derivative to 0 to identify the optimal value of the means μ_k :

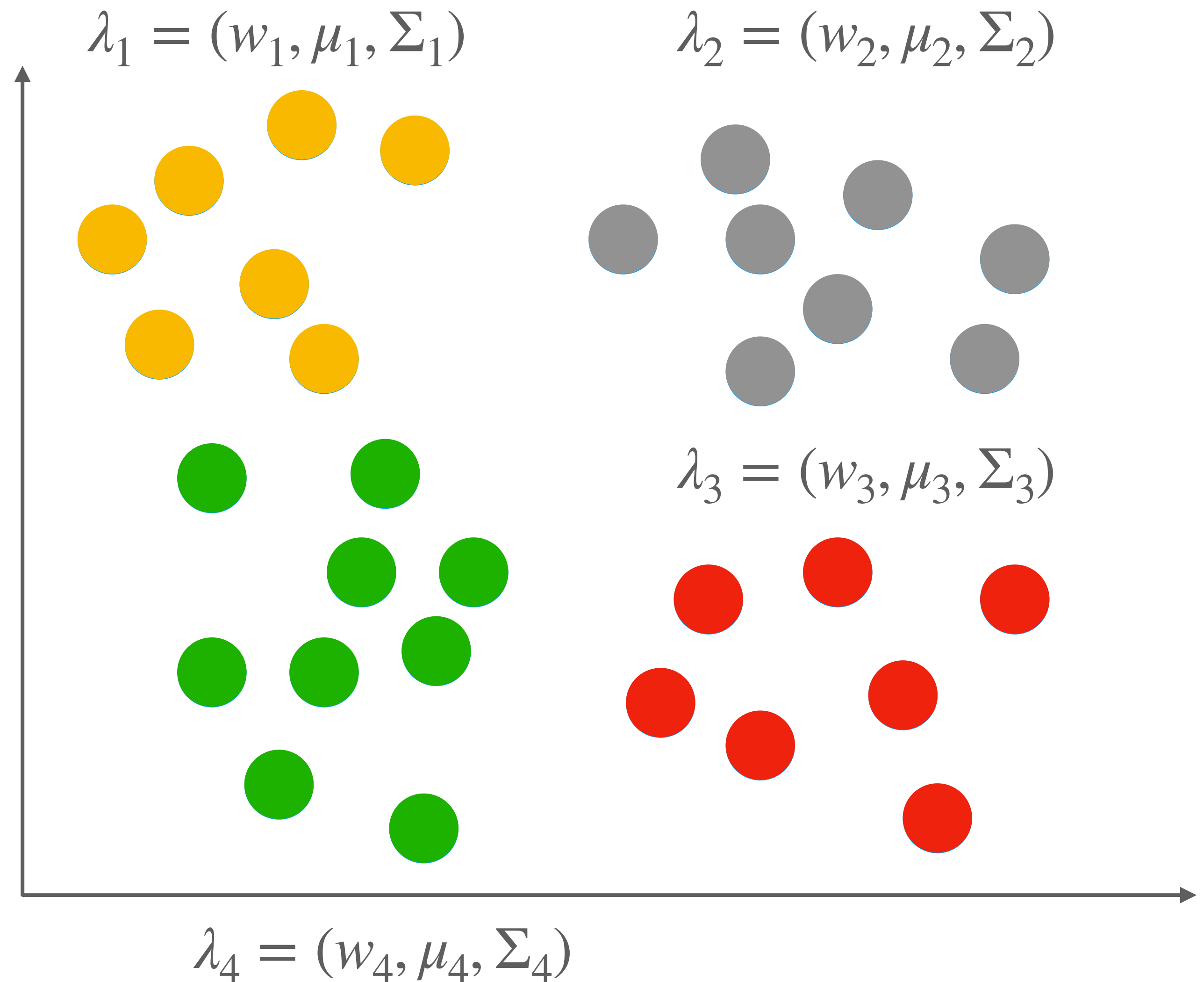
$$\sum_{i=1}^N \frac{1}{\sum_{k=1}^M w_k \mathcal{N}(x_i, \mu_k, \sigma_k)} w_k \mathcal{N}(x_i, \mu_k, \sigma_k) \frac{(x_i - \mu_k)}{\sigma_k^2} = 0$$

III. Motivation for EM

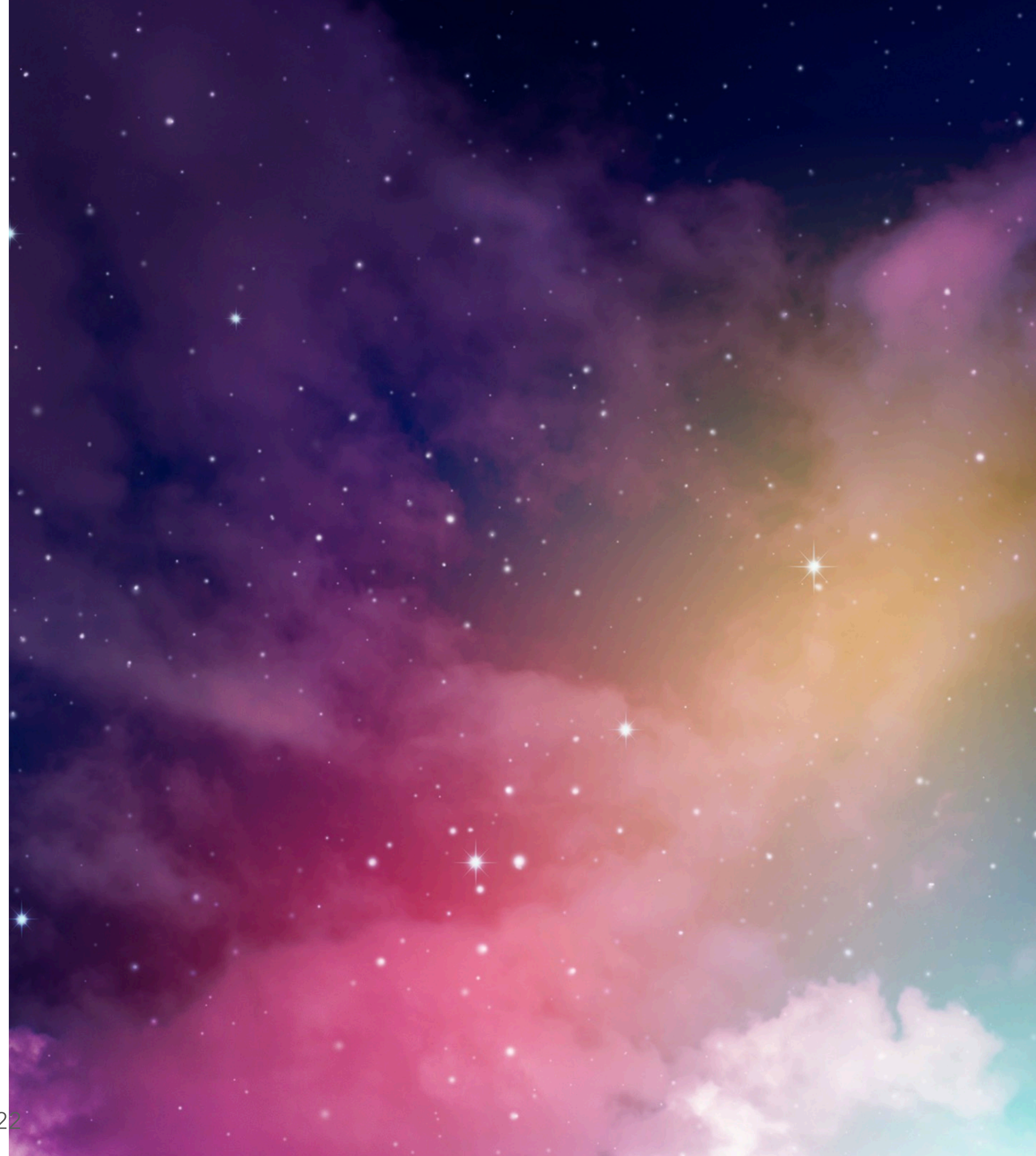
- We cannot solve analytically for μ_k , and therefore we need to find another approach
- What if we knew which Gaussian each data point belongs to? We could solve analytically for the parameters of each Gaussian !
- We can suppose that latent variables Z_i exist, and they describe the component of the mixture to which each observation X_i belongs

III. Motivation for EM

- We can suppose that we know which component each observation X_i belongs to ($Z_i = k$) and it will help us solve each Gaussian. We can iteratively update the parameters using EM in the training cycle.



IV. EM for Gaussian Mixture Models



IV. EM for GMMs

- The main idea behind EM is the following:
 - E-step: We **estimate** the distribution of the hidden variable Z given the data X and the current value of the parameters θ
 - M-step: We **maximize** the joint distribution of the data and the hidden variable and derive the values of the updated parameters
 $\lambda = (w_k, \mu_k, \Sigma_k), k = 1, 2, 3, \dots, M$

IV. EM for GMMs

- EM introduces a latent variable Z corresponding to the component of the GMM to which each observation belongs
- X is now said to be *incomplete data*, and the **complete data** is: (X, Z)
- The joint density is: $P(X, Z | \theta) = P(Z | X, \theta)P(X | \theta)$
- The likelihood $L(\theta | X)$ is now said to be *incomplete*
- The **complete likelihood** now becomes : $L(\theta | X, Z) = P(X, Z | \theta)$

IV. EM for GMMs

- Let us first initialize the parameters of our Gaussians randomly:

$$\theta = (\mu, \sigma, w)$$

- We suppose that we have N observations belonging to M different components

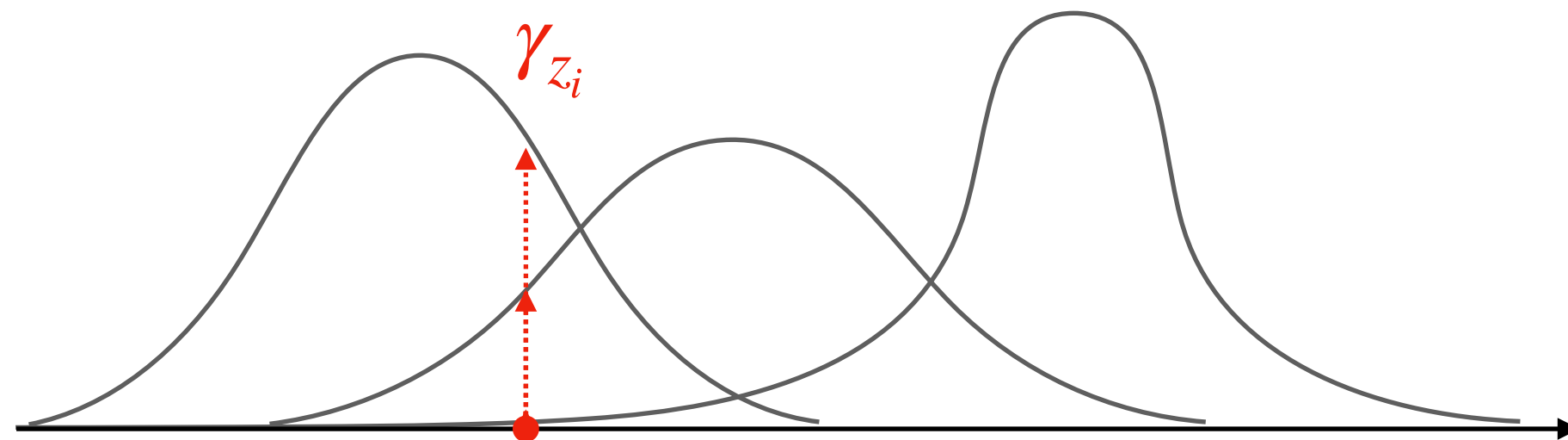
IV. EM for GMMs

- Let us now define $\gamma_{z_i=k}$ as the probability that a given observation belongs to component k
- These « pseudo-posteriors » are defined as:

$$\gamma_{z_i=k} = P(Z_i = k \mid X_i) = \frac{P(X_i \mid Z_i = k)P(Z_i = k)}{P(X_i)} = \frac{w_k \mathcal{N}(x_i, \mu_k, \sigma_k)}{\sum_c w_c \mathcal{N}(x_i, \mu_c, \sigma_c)}$$

Probability of x_i under component k

Sum of probabilities on all components



1. The E-Step

- In the « Estimation » step (E-step), we estimate the value of the auxiliary function:

$$Q(\theta, \theta^{(t)}) = E[\log P(Z \mid \theta) \mid X, \theta^{(t)}]$$

The auxiliary function can be proven to be a lower-bound of the gain of likelihood $L(\theta) - L(\theta^{(t)})$ when updating the parameters (see appendix 1). In other words, if we increase the auxiliary function, we are sure to increase the likelihood.

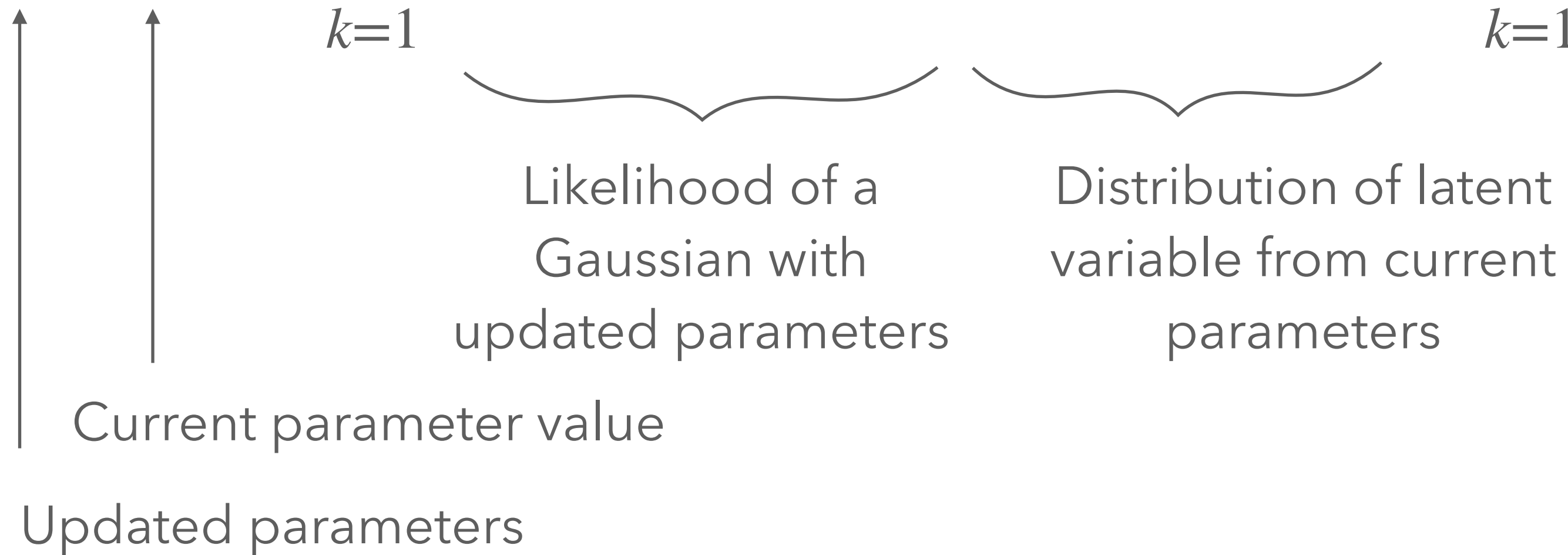
1. The E-Step

- Why don't we do a gradient descent on the Likelihood function?
 - Because it's analytically un-feasible !
 - This is why we need an auxiliary function, a lower-bound of the likelihood gain, which is the only method we know to compute the new value of the parameters

1. The E-Step

- Further expanding the auxiliary function:

$$Q(\theta, \theta^{(t)}) = \sum_{k=1}^M \log L(\theta_k | X, Z) P(Z_k | X, \theta^{(t)}) = \sum_{k=1}^M \log L(\theta_k | X, Z) \gamma_{z_i=k}$$




2. The M-Step

- In the « Maximum » step (M-step), we maximize the value of Q to find the optimal parameter value:

$$\theta^{(t+1)} = \operatorname{argmax}_{\theta} Q(\theta, \theta^{(t)})$$

2. The M-Step

- The update equations of the M-step are defined by setting the derivative of Q to 0 with respect to w, μ, σ . By expanding the expression of the auxiliary function, we get the following expression which can be solved analytically:

$$Q(\theta, \theta^{(t+1)}) = \sum_{k=1}^M \sum_{i=1}^N \log \gamma_k P(Z_k | X_i, \theta^{(t)}) + \sum_{k=1}^M \sum_{i=1}^N \log P(x_i | \theta_k) P(Z_k | X_i, \theta^{(t)})$$

$$\frac{d}{d\mu_k} = 0$$

...

2. The M-Step

- The updated parameters become:

$$\hat{\mu}_k = \frac{\sum_{i=1}^N X_i P(Z_i = k \mid X_i, \theta^{(t)})}{\sum_{i=1}^N P(Z_i = k \mid X_i, \theta^{(t)})} = \frac{1}{N_k} \sum_{i=1}^N \gamma_{Z_i=k} X_i \quad \text{where } N_k = \sum_{i=1}^N \gamma_{Z_i=k}$$

Weighted average of the data with a weight showing how likely the point belongs to the cluster

$$\hat{\sigma}_k^2 = \frac{1}{N_k} \sum_{i=1}^N \gamma_{Z_i=k} (X_i - \mu_k)^2$$

$$\hat{w}_k = \frac{N_k}{N}$$

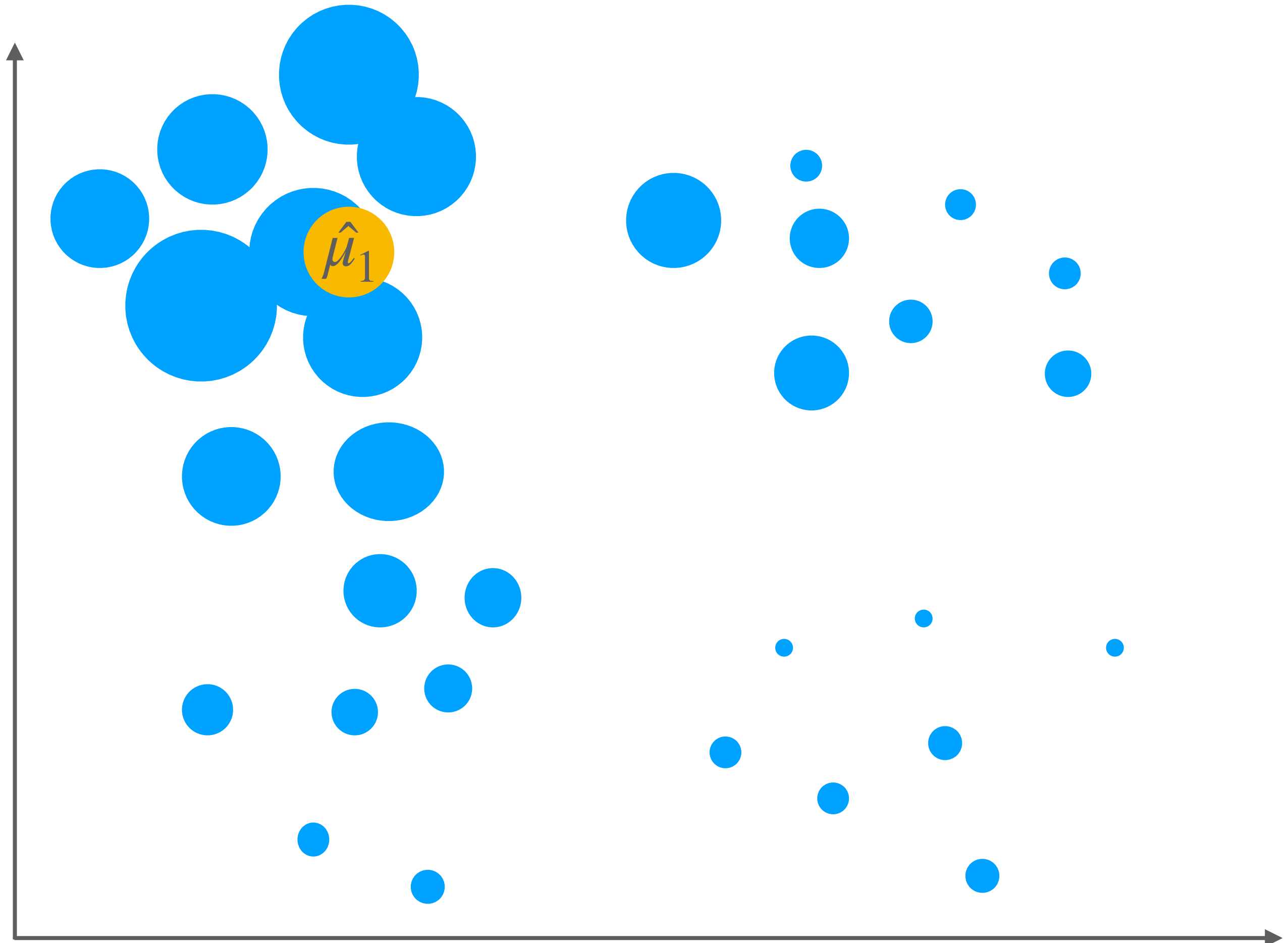
2. The M-Step

The mean update can be seen as a weighted average of the observations by the probability of belonging to the component

$$\hat{\mu}_k = \frac{1}{N_k} \sum_{i=1}^N \gamma_{Z_i=k} X_i$$

$$\hat{\sigma}_k^2 = \frac{1}{N_k} \sum_{i=1}^N \gamma_{Z_i=k} (X_i - \mu_k)^2$$


$$\hat{w}_k = \frac{N_k}{N}$$



3. An iterative process

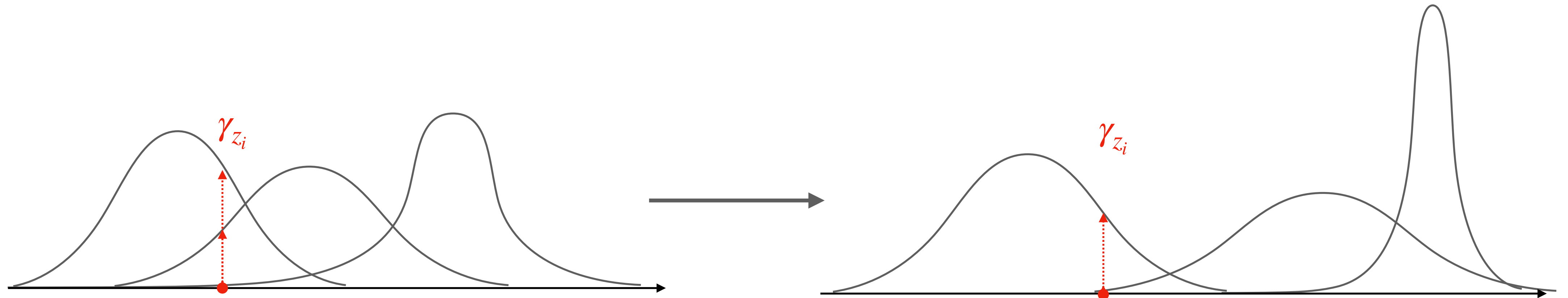
- Using the new values of the parameters $\theta = (\hat{\mu}, \hat{\sigma}, \hat{w})$, inject it in the E-step again:

$$Q(\theta, \theta^{(t+1)}) = \sum_{k=1}^M \log L(\theta_k | X, Z) P(Z_k | X, \theta^{(t+1)})$$

$$\hat{w}, \hat{\mu}, \hat{\sigma} \longrightarrow \gamma_k = \frac{1}{N} \sum_{i=1}^N P(Z_i = k | X, \theta^{(t+1)})$$


3. An iterative process

$$\hat{w}, \hat{\mu}, \hat{\sigma} \longrightarrow \gamma_k = \frac{1}{N} \sum_{i=1}^N P(Z_i = k \mid X, \theta^{(t+1)})$$



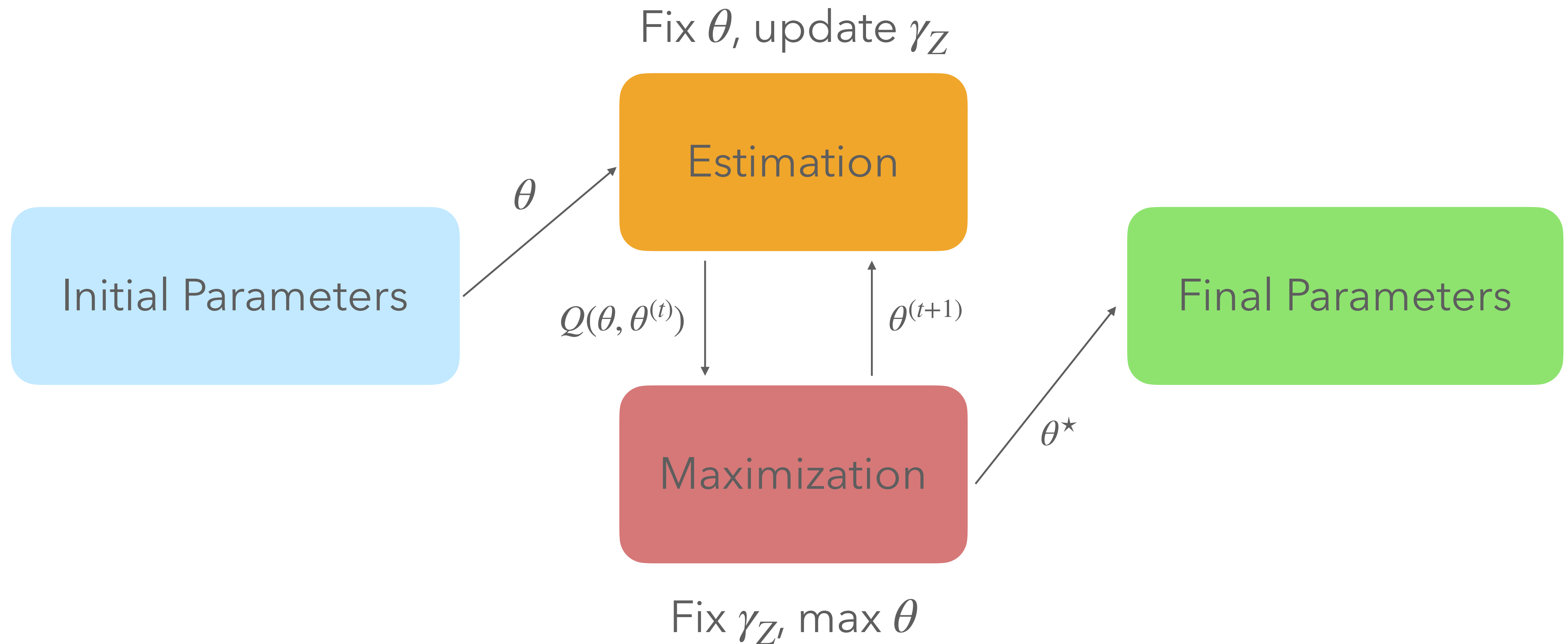
3. An iterative process

- And estimate the value of the optimal parameters again in the M-Step:

$$\theta^{(t+2)} = \operatorname{argmax}_{\theta} Q(\theta, \theta^{(t+1)})$$

3. An iterative process

- The EM cycle can be illustrated as:



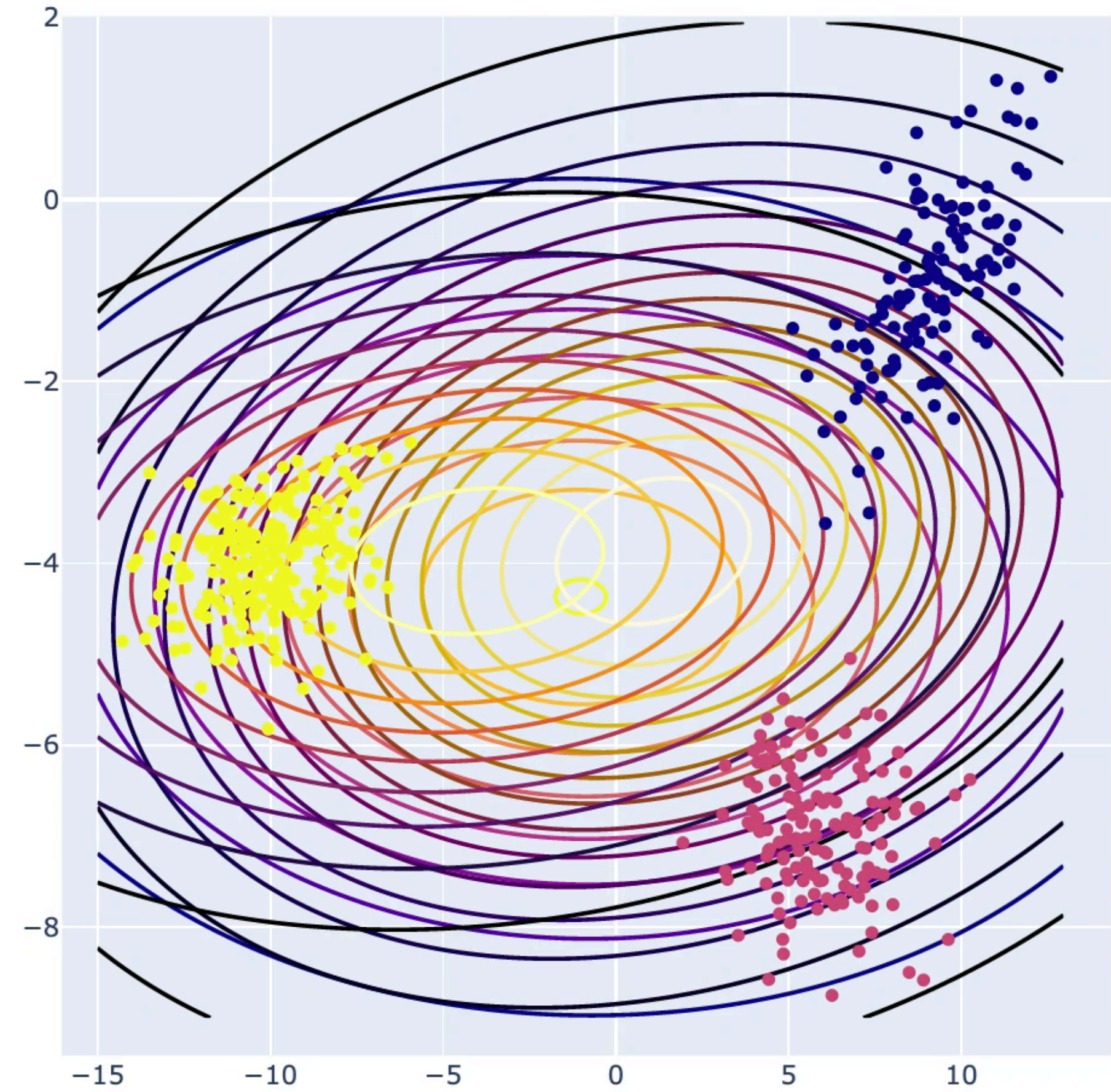
3. An iterative process

Parameters

Number of components

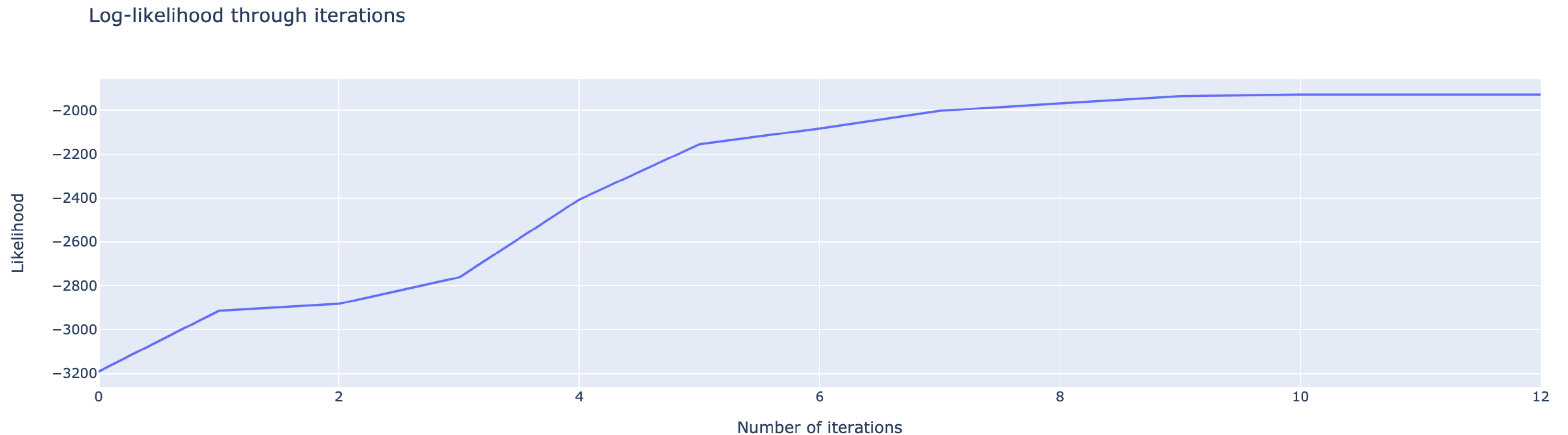


Number of iterations

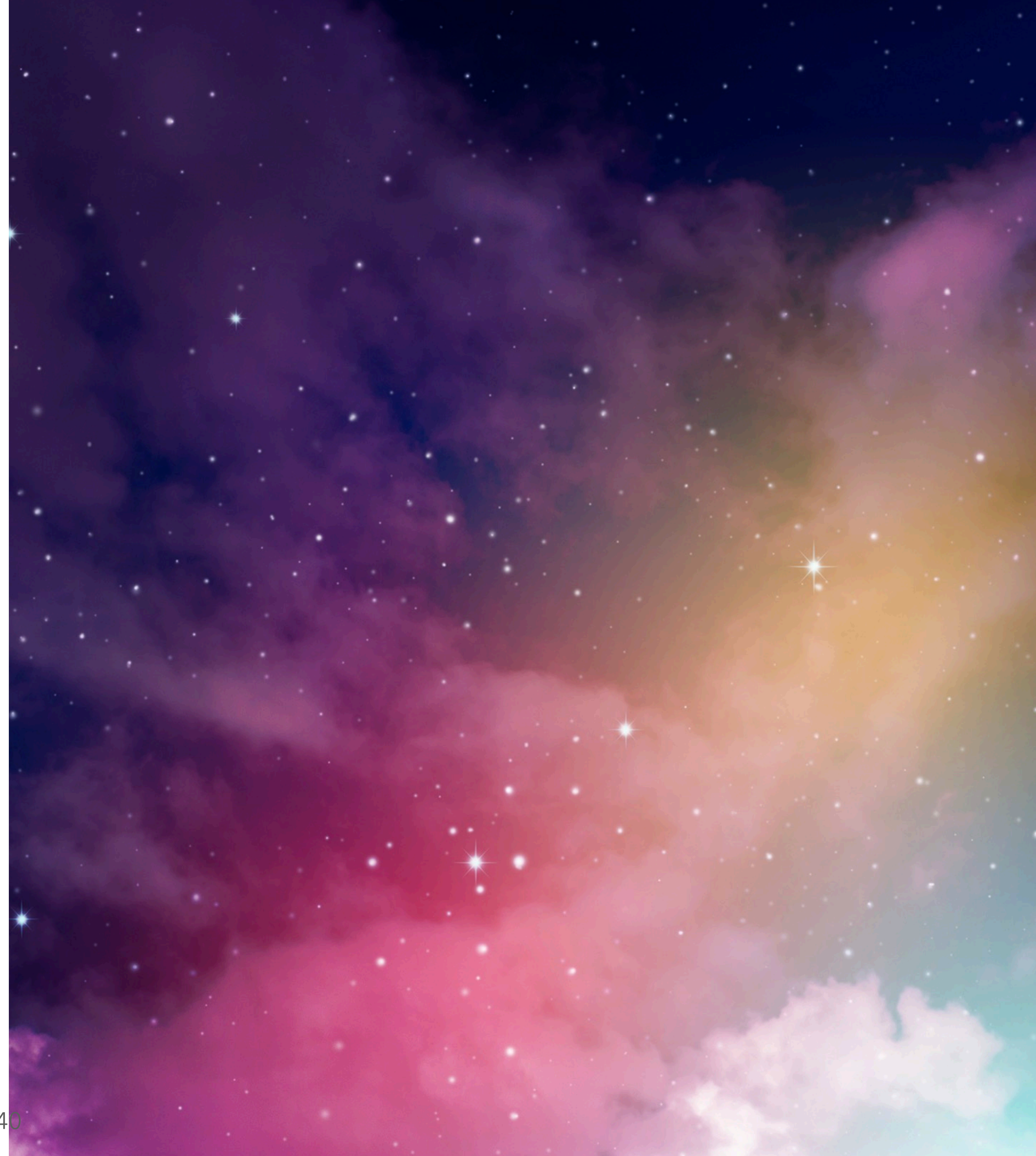


3. An iterative process

- EM is an iterative approach that is guaranteed to increase the likelihood over the number of iterations: $\log p(X | \theta) \geq \log p(X | \theta^{(t)})$

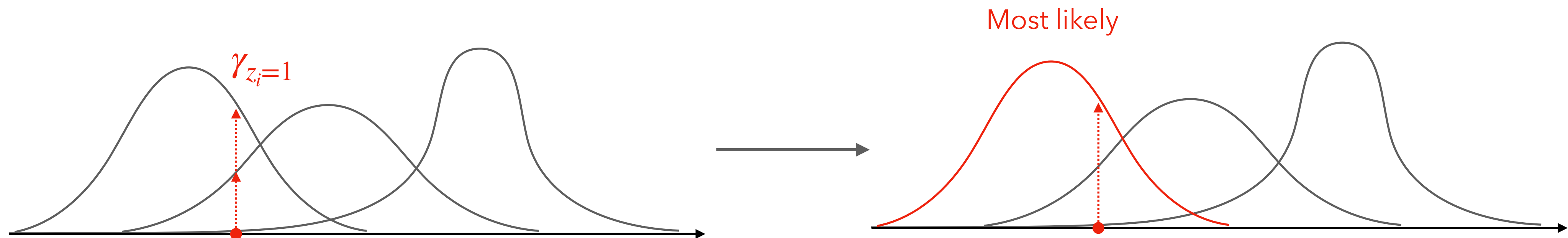


V. Hard / Viterbi EM



V. Hard / Viterbi EM

- So far, what we have seen is called the **Soft EM**. In **Hard EM** or **Viterbi Training**, we make hard decisions for the Z 's: $\max_{\theta, Z} P(X, Z, \theta)$
- In this case, we do not consider a likelihood weighted over all possible \mathbf{Z} with their probabilities, but we simply select the most probable Z and move forward.



V. Hard / Viterbi EM

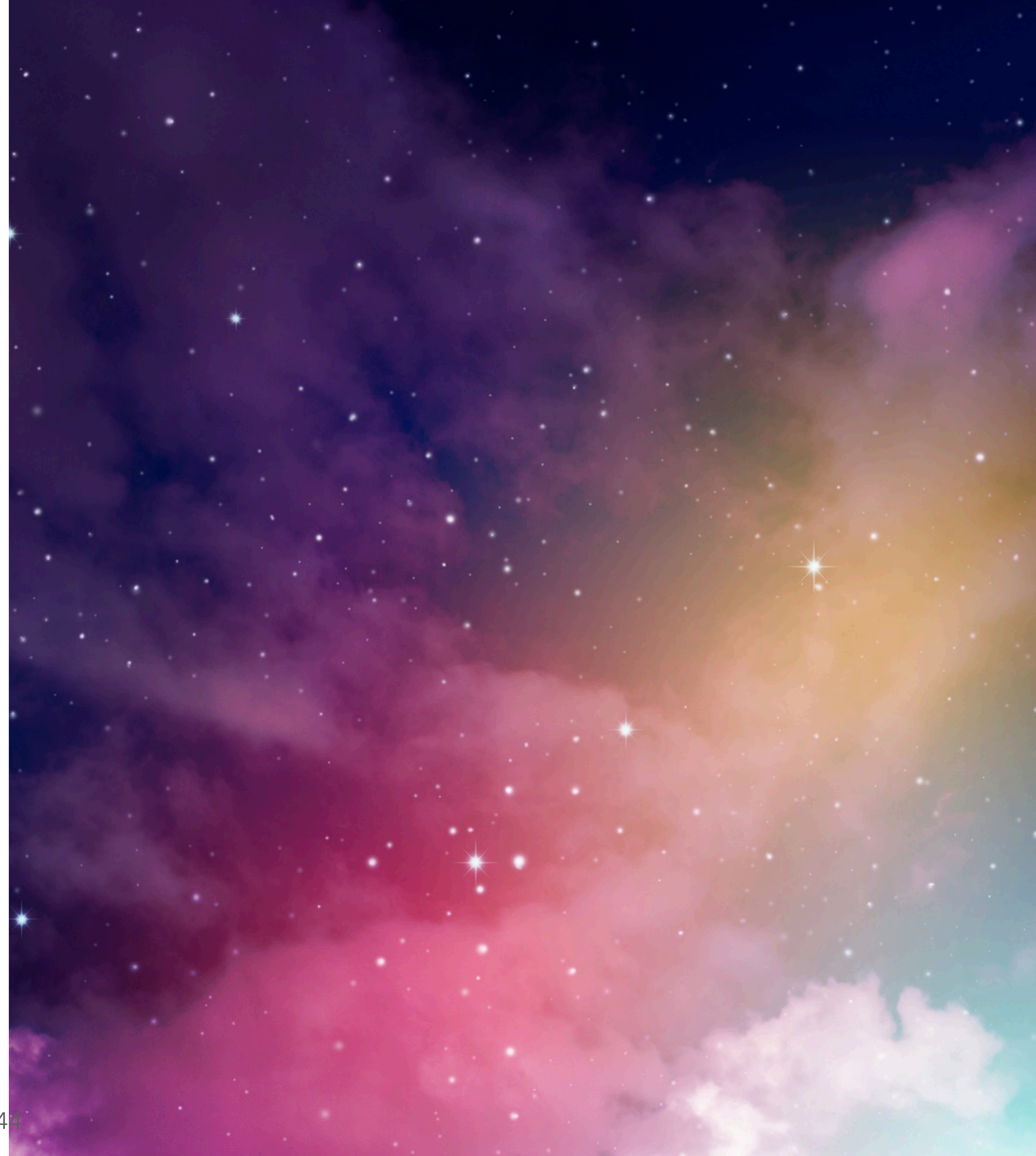
- Hard EM is easier to implement
- But it does not take into account multiple possibilities for Z , which is a problem if our knowledge of Z is limited
- k-Means is actually a special case of Hard EM

Summary of Mean Updates

- Let us compare the Single Gaussian, EM-GMM and Hard EM-GMM:

Single Gaussian	EM-GMM	Hard EM-GMM
$\mu = \frac{1}{N} \sum_{i=1}^N \gamma_1 x_i$ <p style="text-align: center;">↓</p>	$\mu_k = \frac{1}{N_k} \sum_{i=1}^N \gamma_{Z_i=k} X_i$ <p style="text-align: center;">↓</p>	$\mu_k = \frac{1}{N_k} \sum_{i=1}^N \gamma_{Z_i=k} X_i$ <p style="text-align: center;">↓</p>
Probability of 1 to belong to the only cluster	Probability to belong to cluster k	1 if maximum over all k, 0 otherwise

VI. Limits of EM



VI. Limits of EM

- EM is « initialization-dependent », and converges to local optimum
- EM can be initialized with k-Means parameters:
 - The mean of each cluster identified by k-Means gives μ_k
 - We can compute the within-cluster covariances to identify σ_k
 - We can compute the fraction of data attributed to each cluster to identify w_k
- Highly correlated features might prevent the EM from converging

VI. Limits of EM

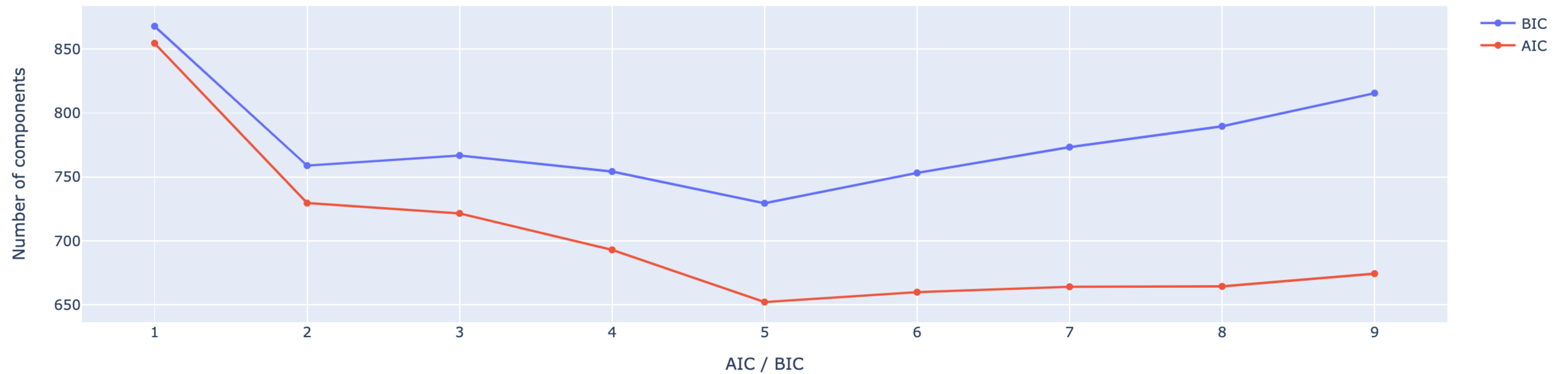
- Since the computation of Jensen's inequality supposes convex functions, EM does not work for all underlying distributions.
- It works for Gaussians, it also works for multinomial distributions, but it cannot be generalized to any distribution

VI. Limits of EM

- Another question is : How to select the right number of components? The same questions arises in k-Means.
- We focus on two criteria:
 - The Aikake Information Criterion (AIC): $AIC = 2p - 2 \ln(L)$
 - The Bayesian Information Criterion (BIC): $BIC = -2 \ln(L) + p \ln(N)$
- Where L is the likelihood of the model, N is the total number of data points, p the total number of estimated parameters. BIC tends to penalize more for model complexity than AIC.
- But it typically requires large computation power

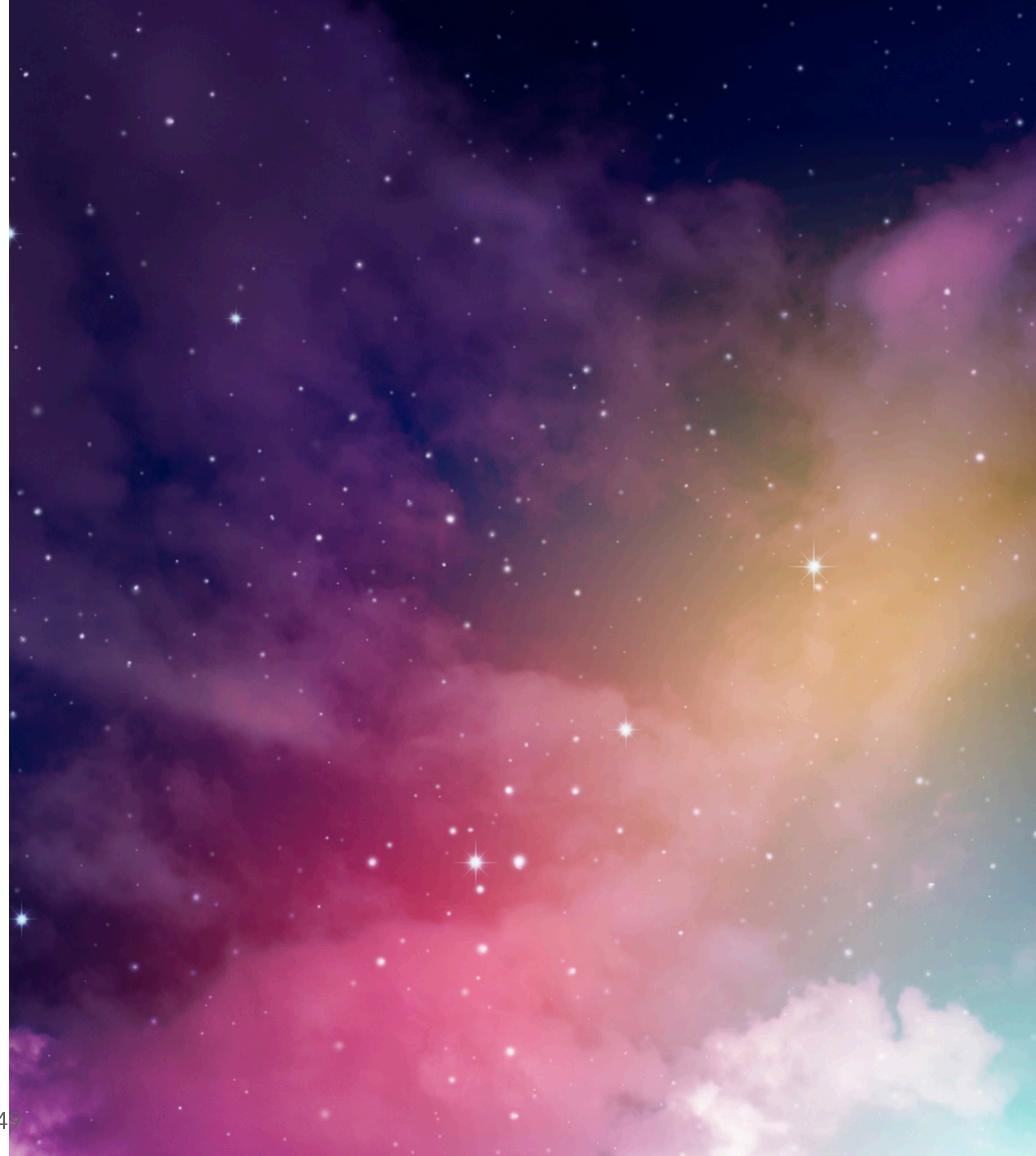
VI. Limits of EM

AIC and BIC over the number of components



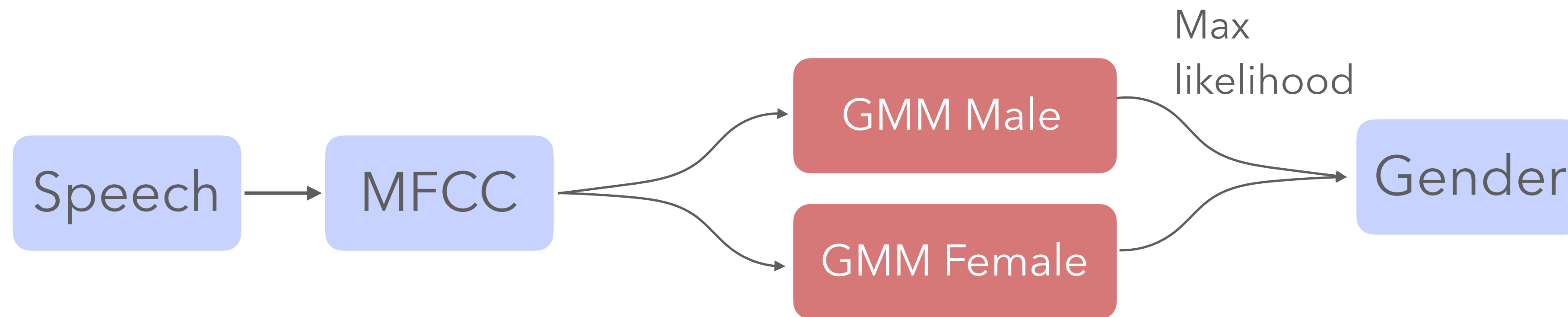
VII.

Applications of EM for GMM



1. GMMs in Speech

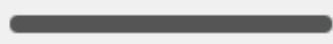

- GMMs are widely used in speech, for example in gender detection, where one GMM for each gender can be fitted on MFCCs, and we attribute the sample to the GMM with the highest likelihood



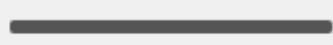
1. GMMs in Speech

Recordings

Male recording

▶ 0:00 / 0:10   ⋮

Female recording

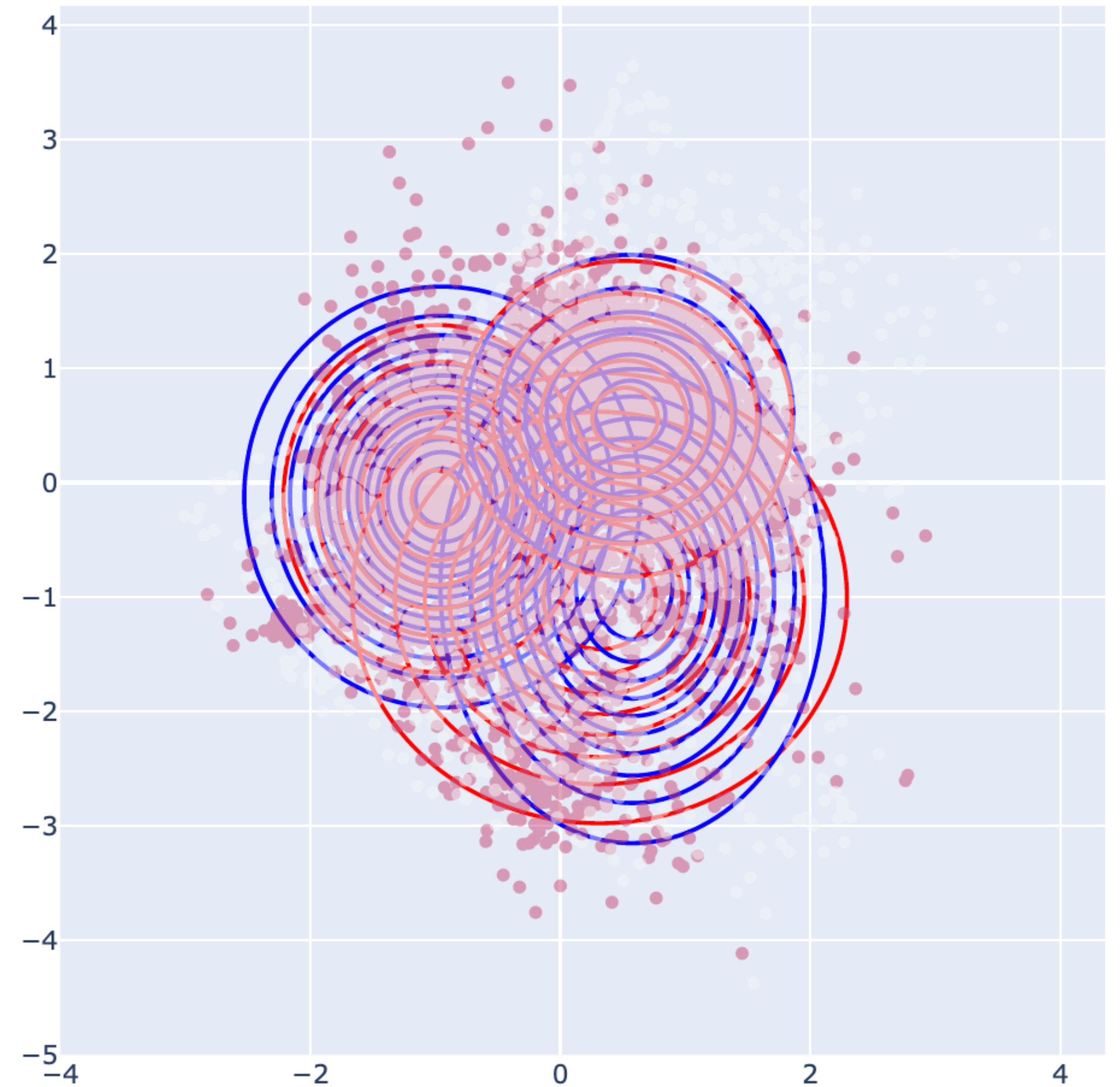
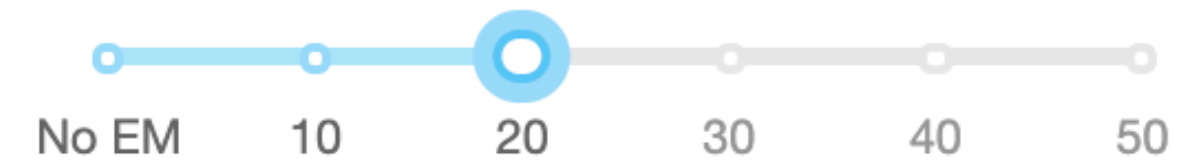
▶ 0:00 / 0:10   ⋮

Parameters

Number of components



Number of iterations



2. k-Means for Vector Quantization

- The k-Means, once applied on images or other signals, are called **Vector Quantization** (VQ) and can be used as compression method for images for example, which prevents from storing the value of each pixel, but simply the clusters and the values identified by EM. To be more specific, k-Means is one of the methods that can be used to perform VQ.

2. k-Means for Vector Quantization



Original image



Compressed image



3. GMMs for Background Subtraction

- GMMs are also used for background subtraction in computer vision for example, where the background is a given cluster, and the objects to keep are another cluster.

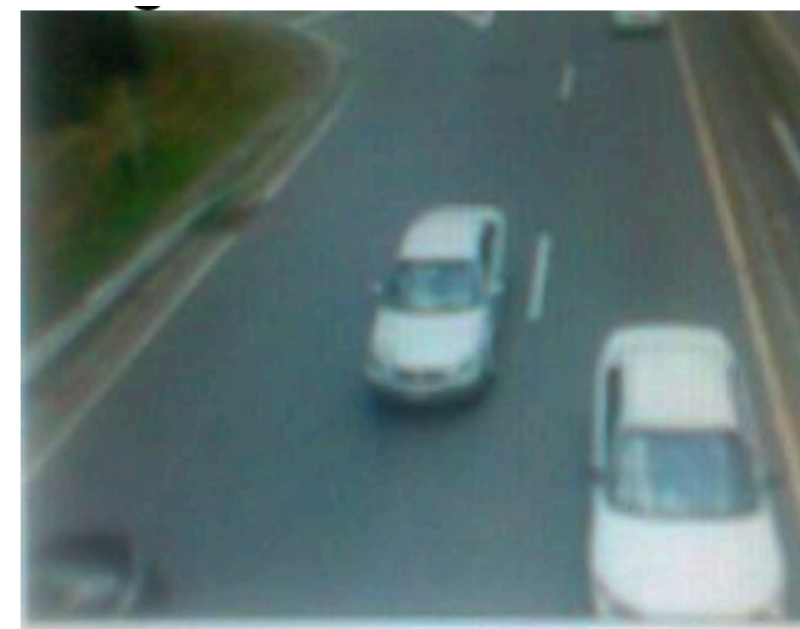


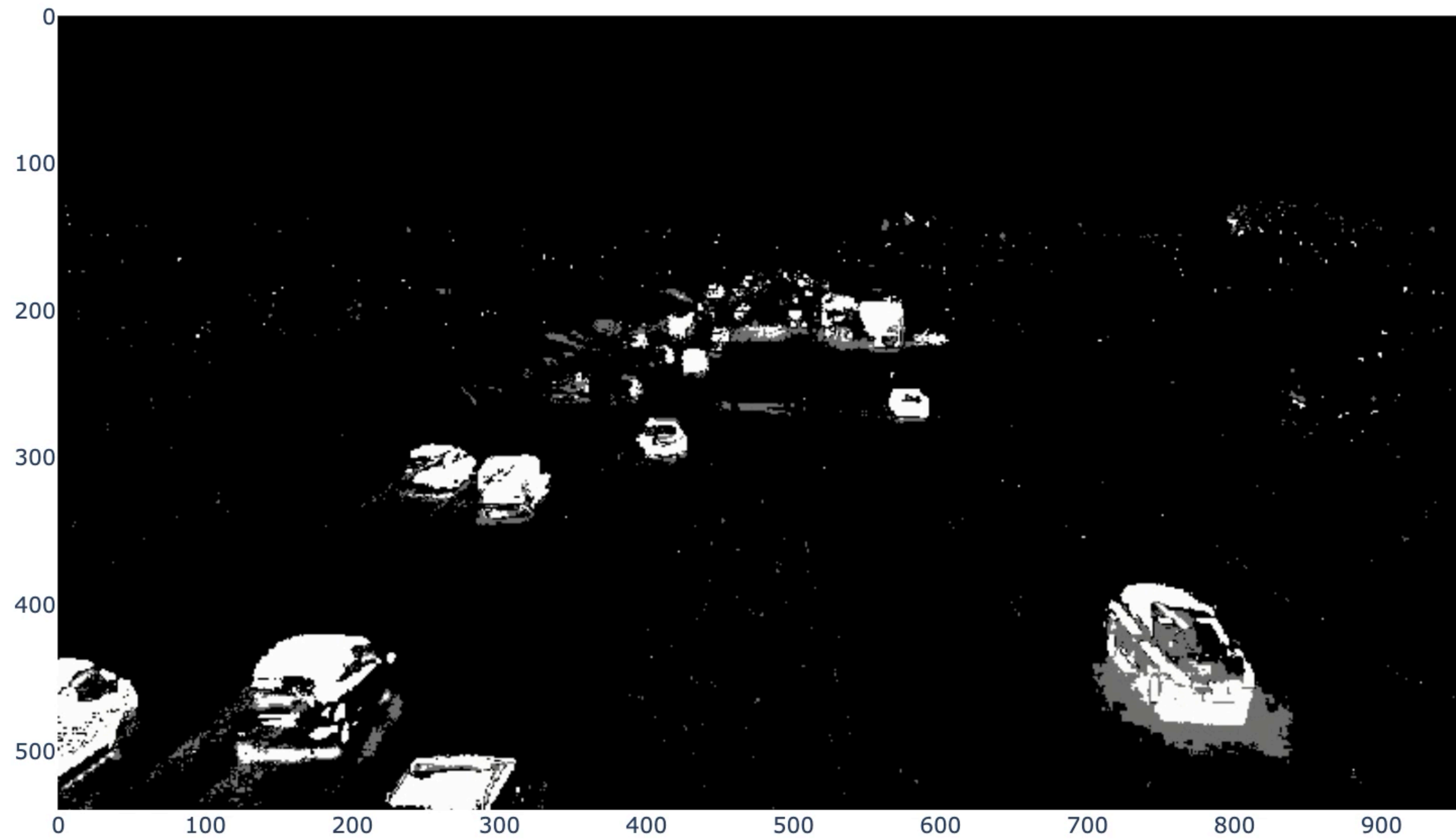
Figure 3.a



Figure 3.b

As shown in the figure 3.a the pixels corresponding to the road which is background in the image do not undergo state changes, as a result the value 0 (black) is attributed and appears black as shown in figure 3.b. The pixels corresponding to cars undergo drastic changes in state, so the value 1 (white) is attributed and cars appears white as shown in figure 3.b

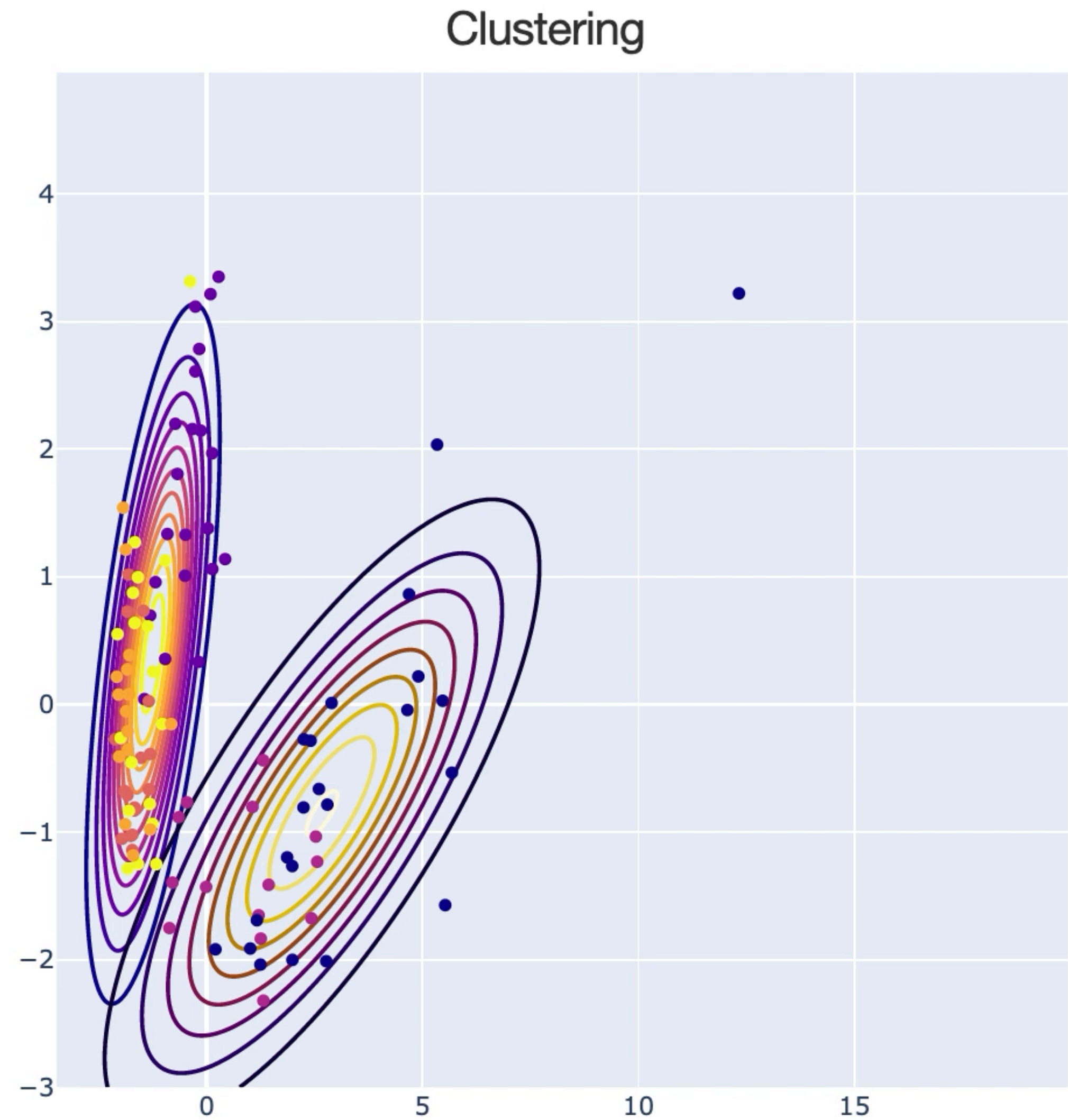
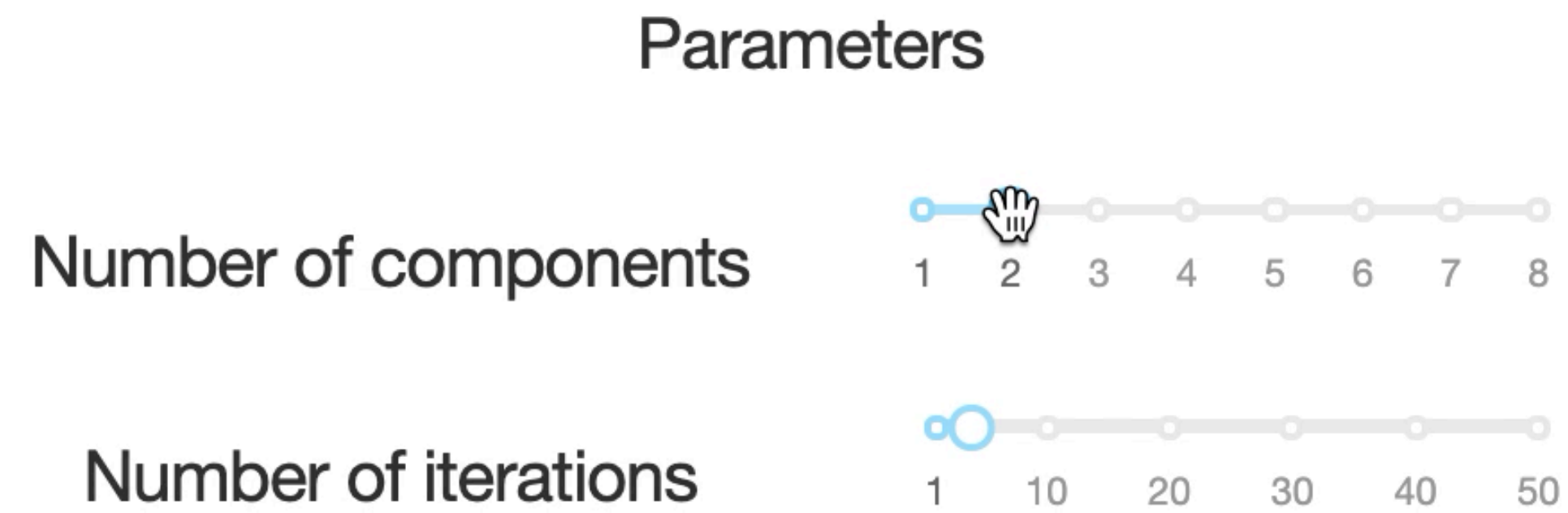
3. GMMs for Background Subtraction



4. GMMs for clustering

- GMMs can be used in unsupervised learning tasks. This is for example the case with health data. We performed a clustering using GMMs on breast cancer data, and displayed the contours of the GMM components. We allow the user to select the number of the components in the GMM.

4. GMMs for clustering

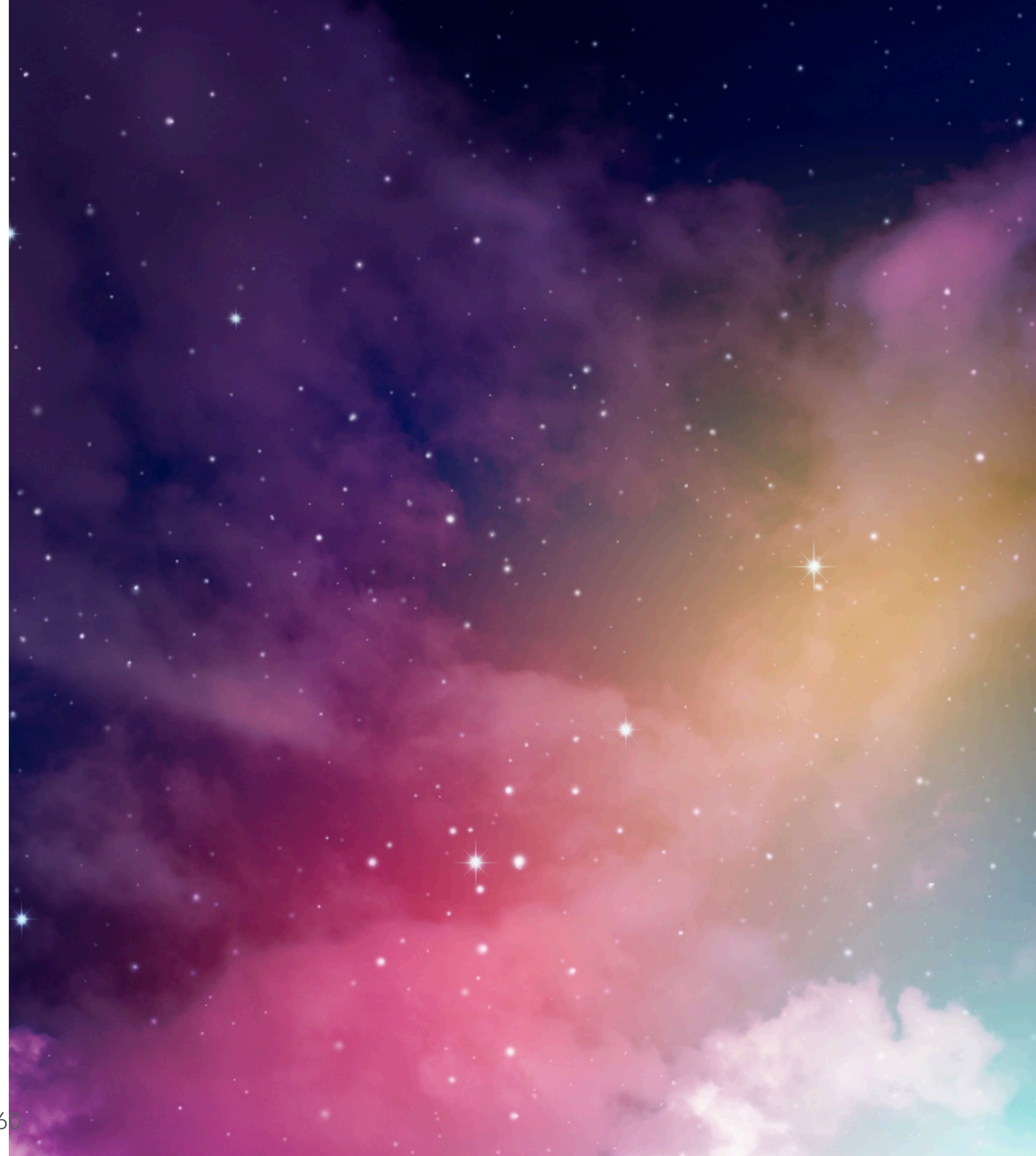


Break / Questions?

Part II: EM and HMMs

Theory and examples

I. Introduction to HMMs

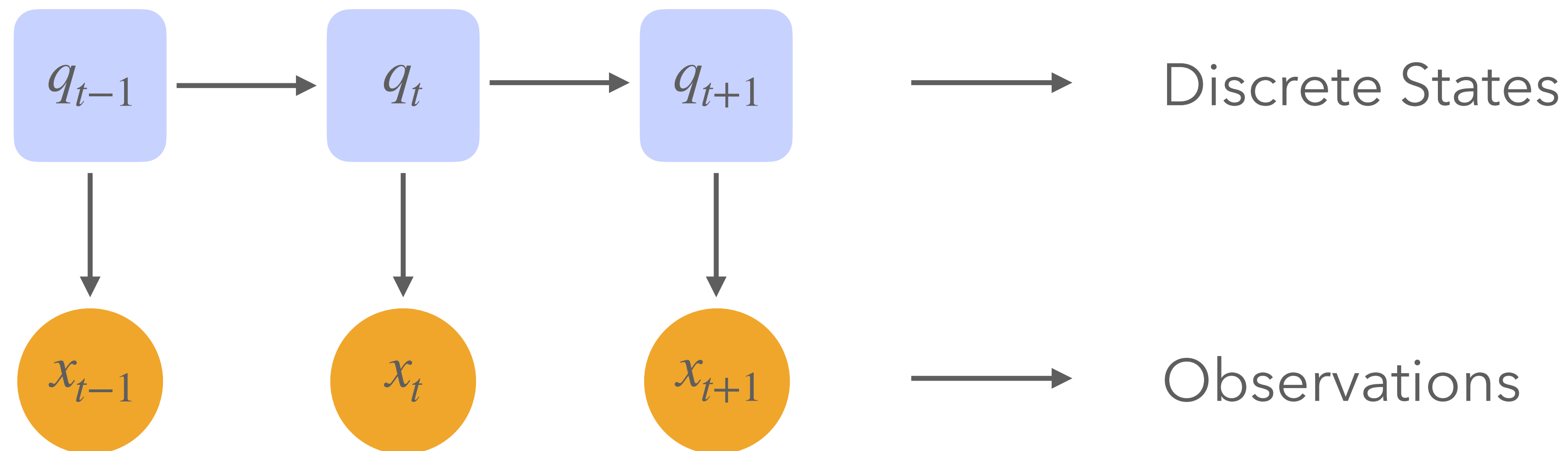


1. Reminder on Hidden Markov Models

- Hidden Markov Models are statistical models for which we collect observations (discrete or continuous) and assume a set of underlying discrete states. The states are said to be hidden because they are a *non-observable stochastic process*.

1. Reminder on Hidden Markov Models

- Let us consider the continuous case of HMMs. The model generates an observation x_t according to a probability distribution (e.g. Gaussian) of the specific state it is in. We suppose that x_t is independent of all other states.



1. Reminder on Hidden Markov Models

- Notation:
 - We have N possible states
 - We have T observations

2. Hidden Markov Models parameters

- HMMs are parametrized as:

$a_{jk} = P(q_{t+1} = j \mid q_t = k)$ \longrightarrow State transition probability

$b_j(x) = P(x \mid q = j)$ \longrightarrow Observation PDF

$\pi_j = P(q_0 = j)$ \longrightarrow Initial state probabilities

$\lambda = \{a_{jk}, b_j(x), \pi_j\}$

2. Hidden Markov Models parameters

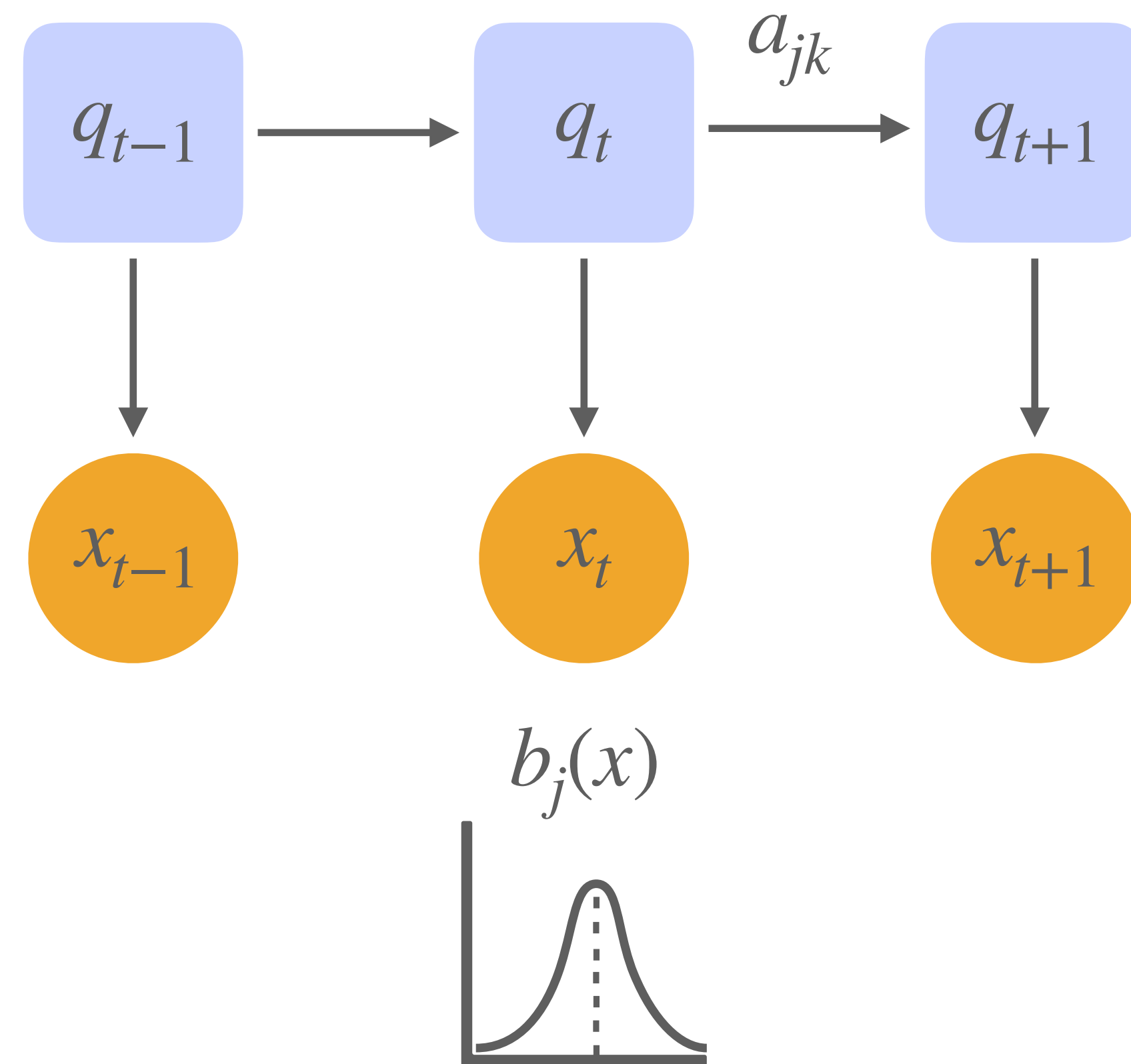
- HMMs are parametrized as:

$$a_{jk} = P(q_{t+1} = j \mid q_t = k)$$

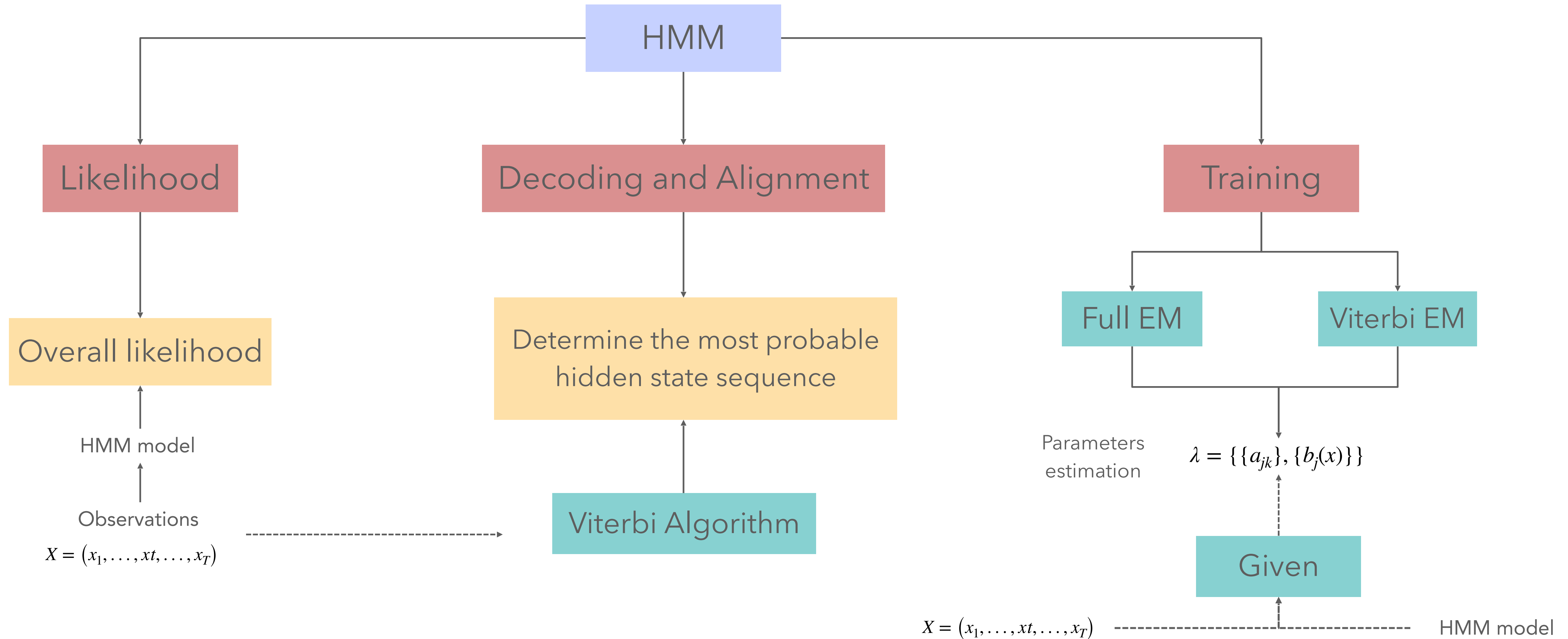
$$b_j(x) = P(x \mid q = j)$$

$$\pi_j = P(q_0 = j)$$

$$\lambda = \{a_{jk}, b_j(x), \pi_j\}$$



3. Major problems in HMMs



4. Sequence Likelihood

- The likelihood of a sequence of observations X and states Q is defined as:

Joint likelihood of X and Q

Initial occupancy probability

Total number of observations

Observation probability given current state

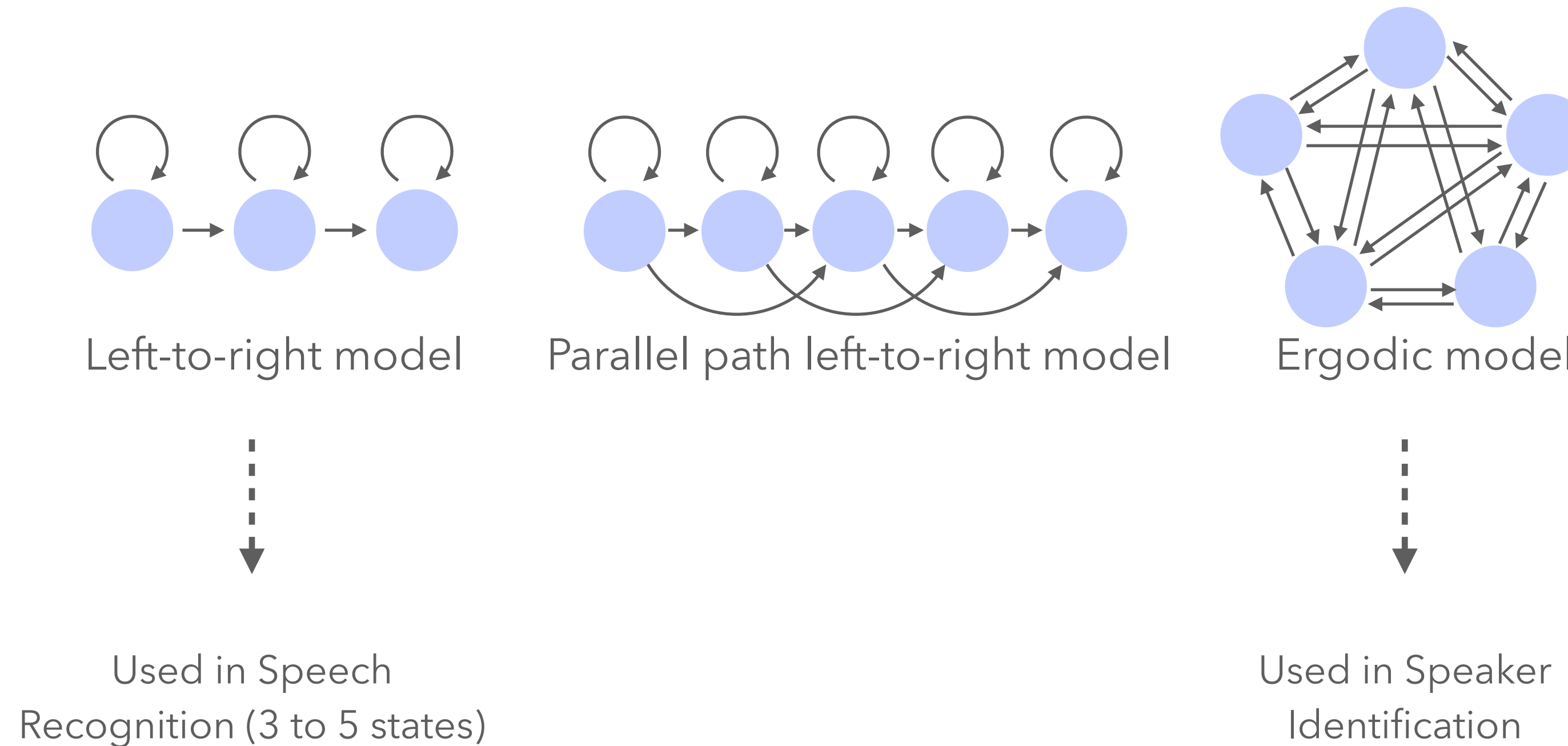
$$P(X, Q; \lambda) = \pi(q_1)P(x_1 | q_1) \prod_{t=2}^T P(q_t | q_{t-1})P(x_t | q_t)$$

Observation probability given current state

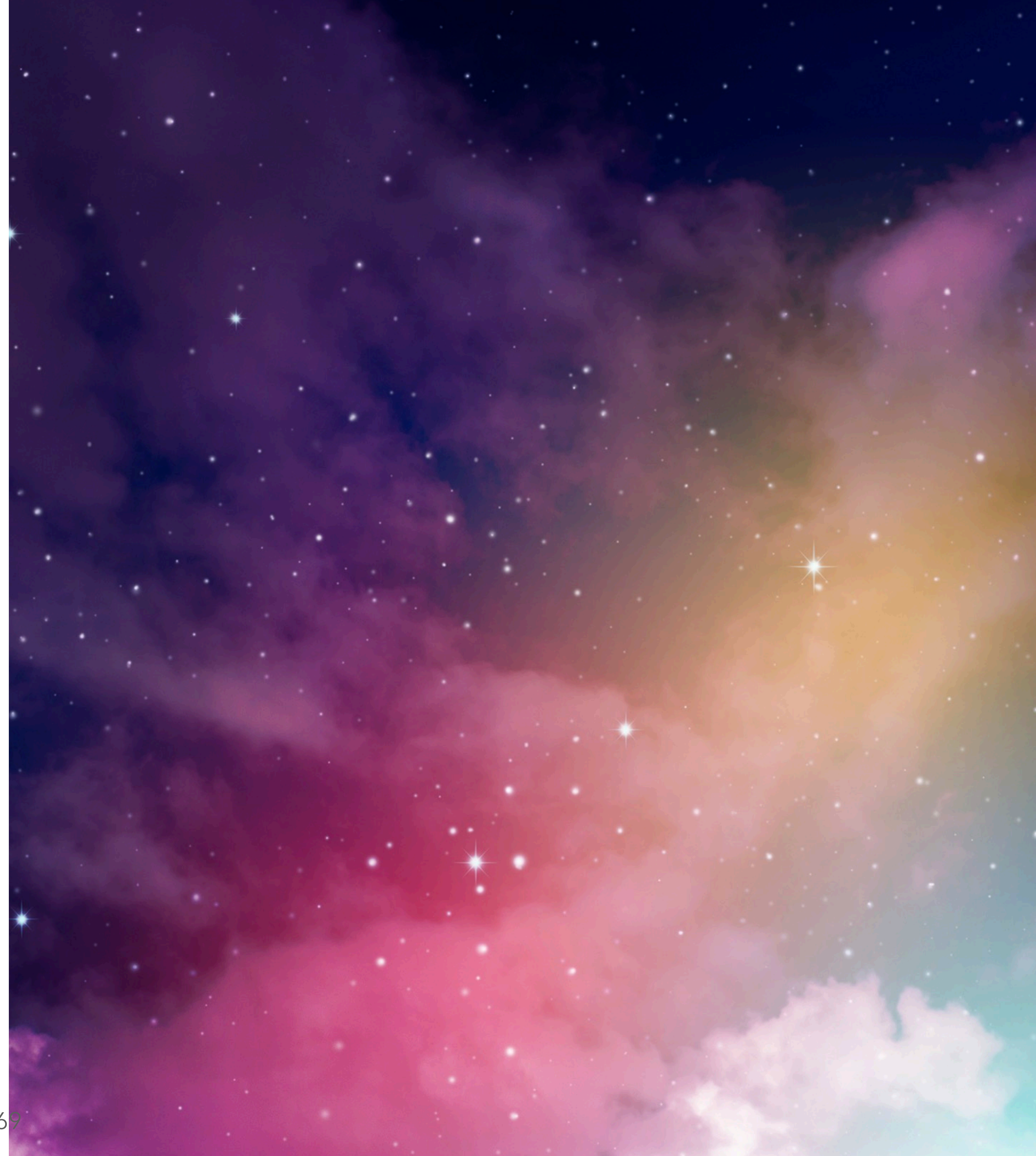
State's probability given a previous state

5. Topology of HMMs

- The topology of HMMs defines the structure of the HMM, and we tend to use special kinds of HMMs in Speech-related tasks. (see appendix 2 for applications to speech)

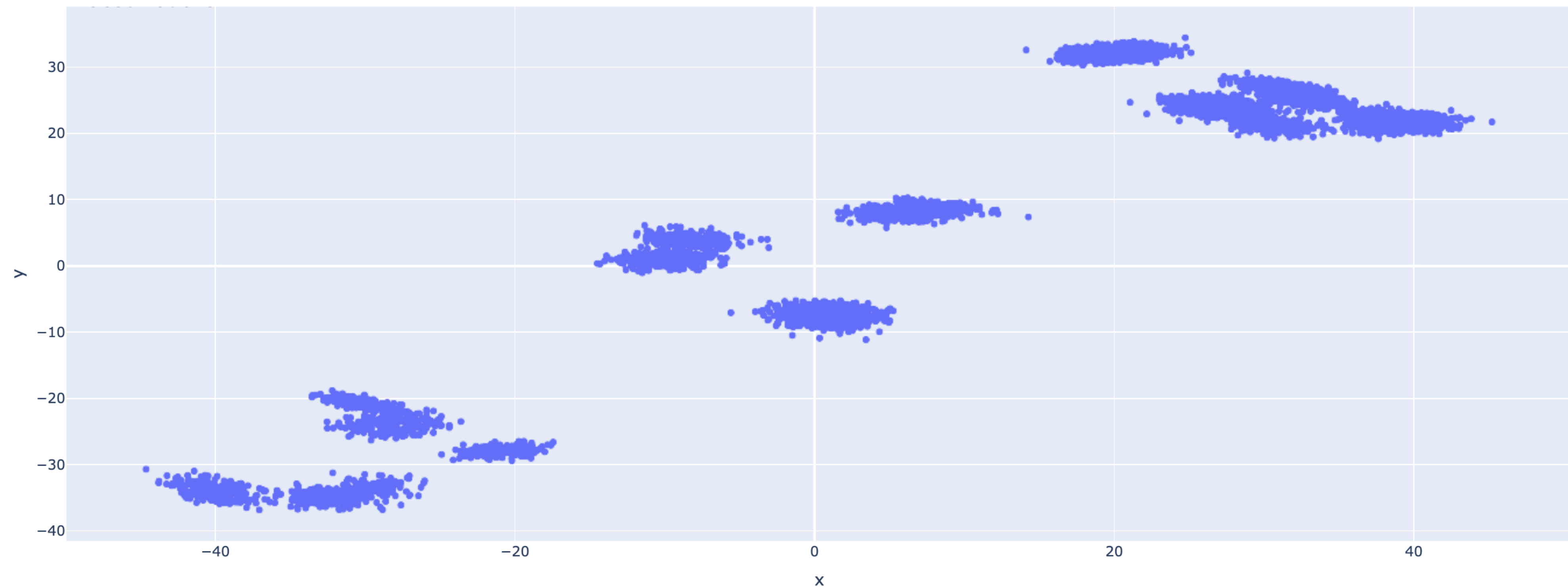


II. HMM training with forward-backward algorithm (Baum-Welch)



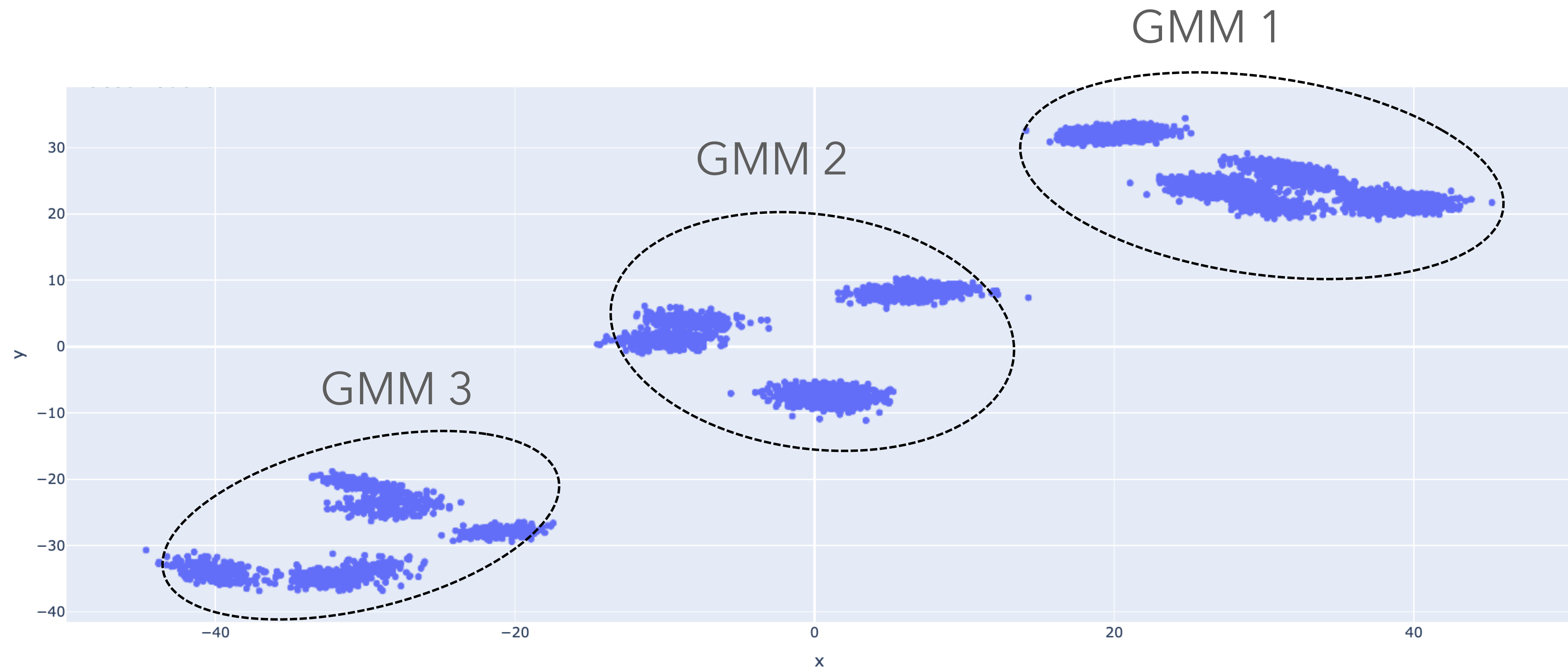
1. What is training?

- What is HMM training? In HMM training, we only see a sequence of observations, and we want to estimate parameters from that sequence of observations. For example, what generated these observations?



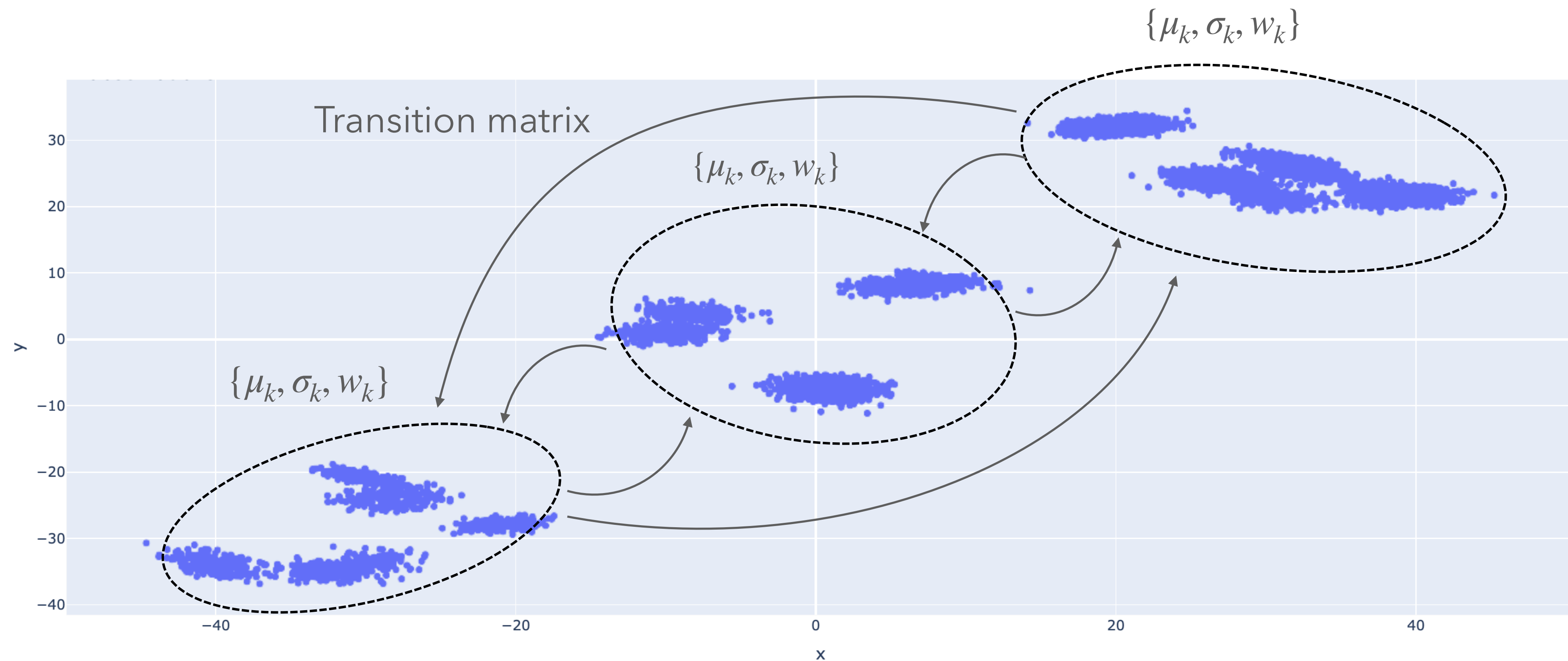
1. What is training?

- These data look like they have been generated by different GMMs, in 3 different states.



1. What is training?

- And « training » an HMM means finding these parameters:



1. What is training?

We can summarize the parameters to learn as:

- A the transition matrix
- π the initial states probability
- B the parameters of the GMMs (means, covariances and weights of each GMM)

On our side, we fix:

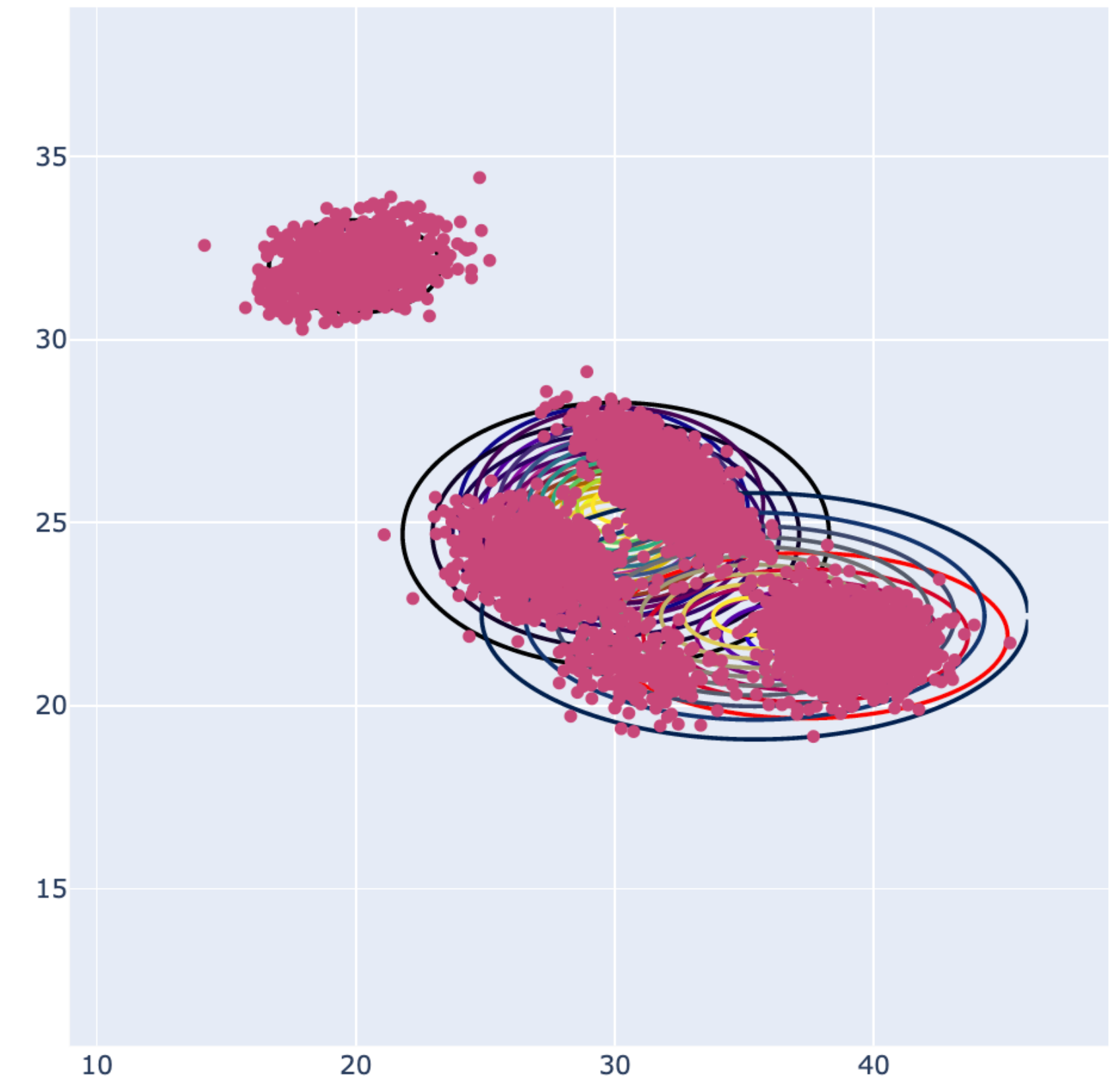
- The number of states (i.e the number of GMMs)
- The number of components for each GMM

Both of the parameters are hard to tune, but in speech recognition, we usually use 5 states of 1024 components.

1. What is training?

Show the GMMs belonging to state:

For example, here are the parameters identified for the GMM corresponding to the first state.



1. What is training?

For everything that comes next, unless specified otherwise, we will suppose a discrete HMM.

2. What's hard about training?

- The whole training cycle became much harder... We need to estimate the states in which our system is through the transition matrix A
- And we need to estimate the PDF of the underlying distribution (Gaussian, GMM) through parameter B

2. What's hard about training?

- Just like in EM for GMMs, life would be easier to estimate parameters if we knew which state generated each data point.
- In that case, estimating the transition probability between i and j would simply be the number of transitions from i to j divided by the number of transitions from i to any other state
- And finding the parameters of each GMM can be done using EM

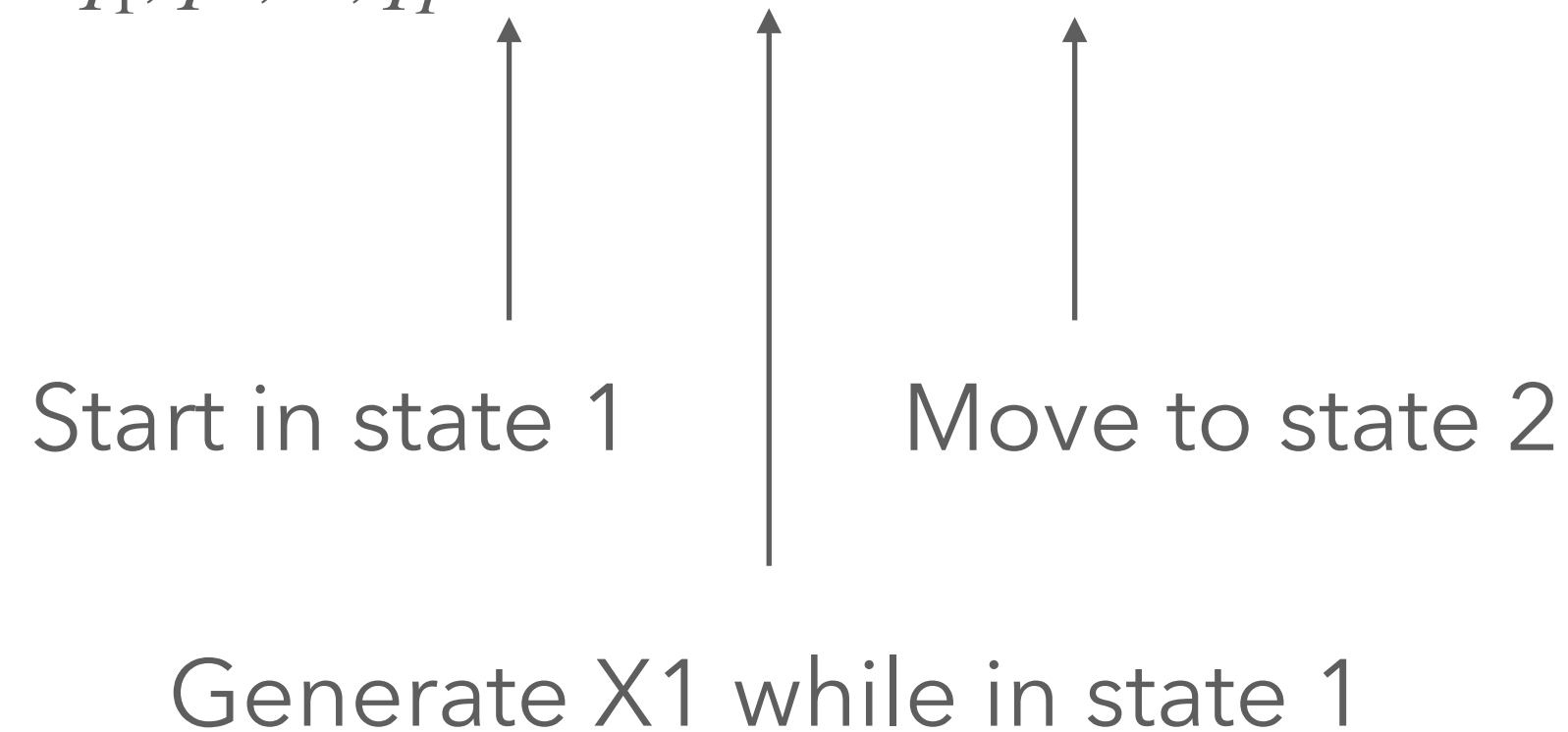
2. What's hard about training?

- More formally, when training HMMs, we try to maximize $P(X \mid \lambda)$ where X is the observation and λ the parameters of the HMM M :

$$\lambda^* = \operatorname{argmax}_{\lambda} P(X \mid \lambda)$$

- One of the problems is that in order to compute $P(X \mid \lambda)$, we would need to compute:

$$P(X \mid \lambda) = \sum_{q_1, q_2, \dots, q_T} \pi_{q_1} b_{q_1}(x_1) a_{q_1 q_2} b_{q_2}(x_2) a_{q_2 q_3} \dots a_{q_{T-1} q_T} b_{q_T}(x_T)$$



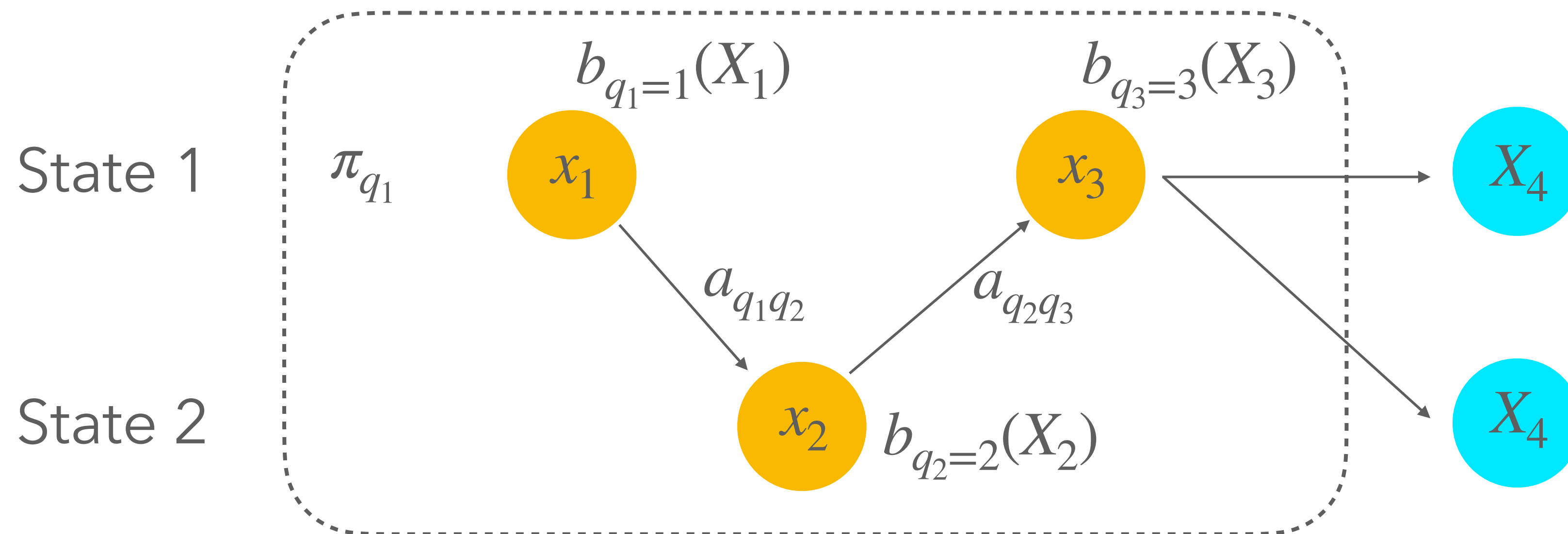
3. The forward algorithm

- This computation is almost infeasible (quickly reaches 10^{70} for simple examples)
- In most paths we compute, there is only 1 value that changes in the chain. Therefore, we introduce the **forward algorithm**, a helper that greatly reduces the computation of the likelihood of a sequence given parameters by storing intermediate values that lead to a given state at a given time. The complexity is reduced from $O(N^T)$ to $O(N^2T)$ where T is the number of observations and N is the number of hidden states.

- It is defined as:

$$\alpha_t(j) = P(x_1, x_2, \dots, x_t, q_t = j \mid \lambda)$$

3. The forward algorithm



Same computation for both paths until $a_{q_1,k}$, so we just need to store and not compute it twice.

3. The forward algorithm

- We can compute the probability of an observation sequence in the following manner:

Initialization: $\alpha_1(i) = \pi_i b_i(x_1)$

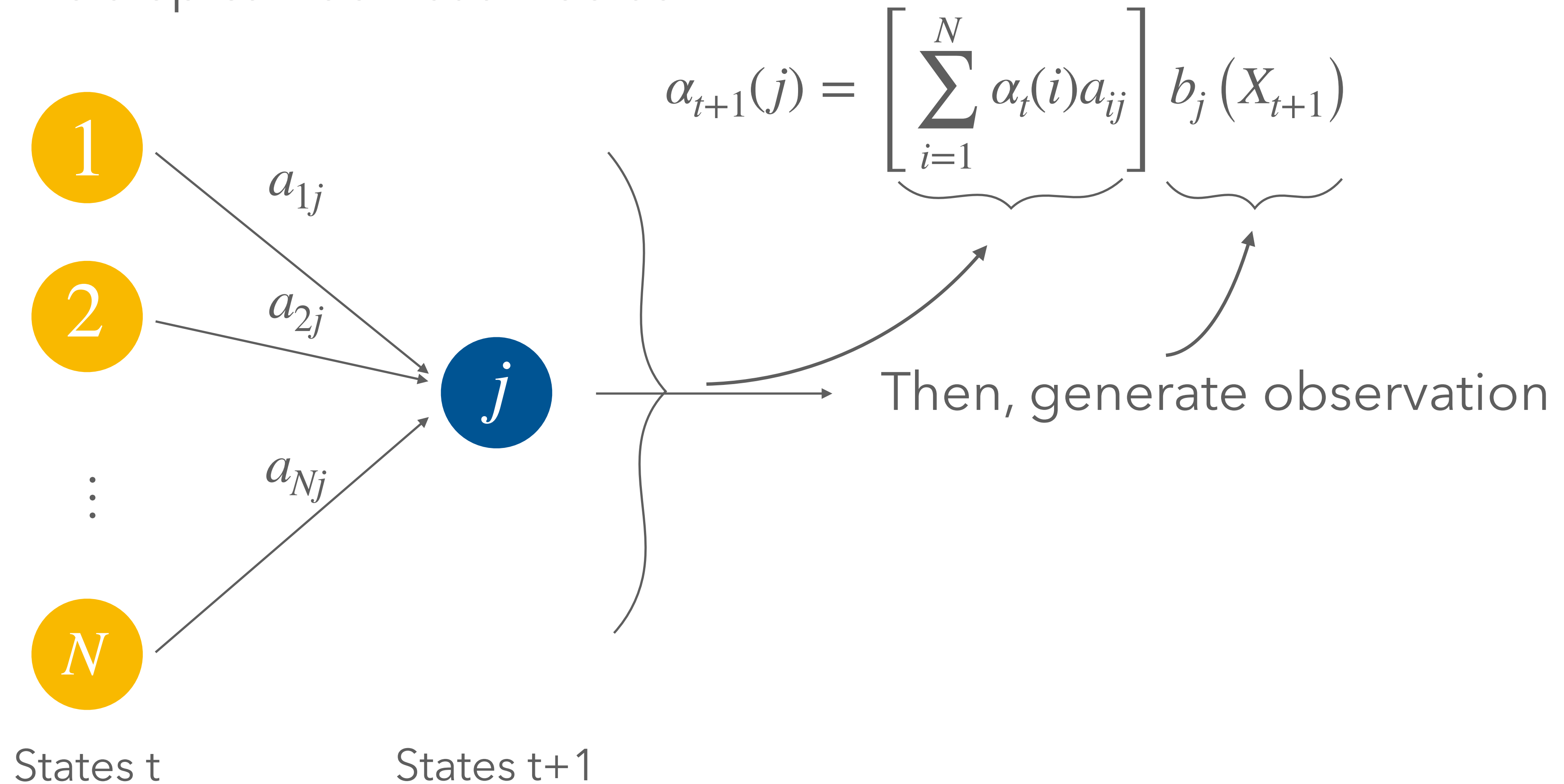
$$\alpha_t(j) = P(x_1, x_2, \dots, x_t, q_t = j \mid \lambda)$$

Iteration: $\alpha_{t+1}(j) = \left[\sum_{i=1}^N \alpha_t(i) a_{ij} \right] b_j(x_{t+1})$ = All possible ways to reach j * Proba. to generate observation

Termination: $\alpha_E = P(X \mid \lambda) = \sum_{i=1}^N \alpha_T(i)$ Sum over all the possible states that we could have ended up in

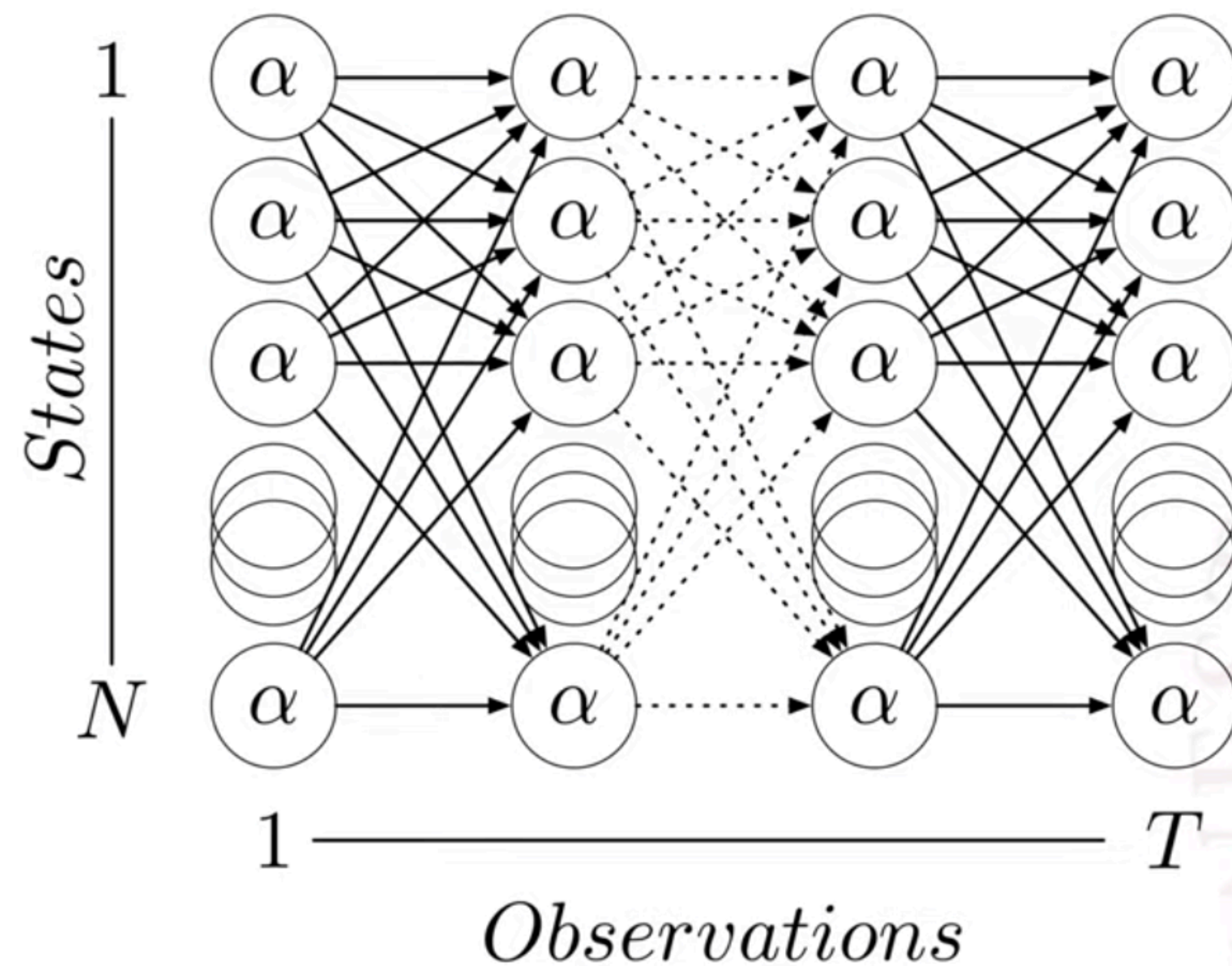
3. The forward algorithm

- The iterative step can be visualized as:



3. The forward algorithm

- The termination step can be visualized as: $\alpha_E = P(X \mid \lambda) = \sum_{i=1}^N \alpha_T(i)$



3. The forward algorithm

Note, the forward algorithm requires transition probabilities. Therefore, we must first initialize the values of the transition matrix A and the observation probabilities B at first.

4. The backward algorithm

- Similarly, we can present the backward algorithm that will be useful for HMM training:

Initialization: $\beta_T(i) = 1$

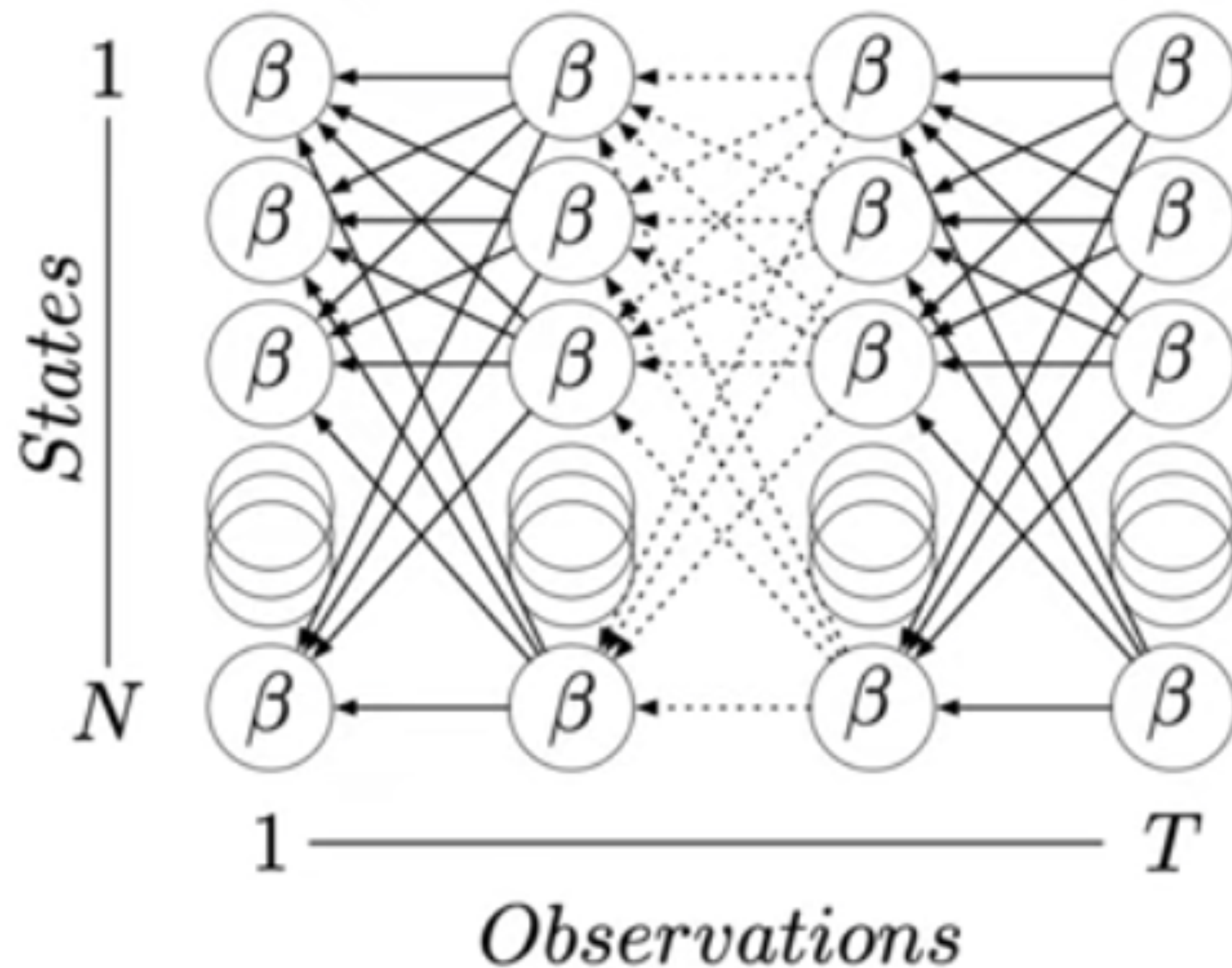
$$\beta_t(j) = P(x_1, x_2, \dots, x_t \mid q_t = i, \lambda)$$

$$\text{Iteration: } \beta_t(j) = \left[\sum_{i=1}^N \beta_{t+1}(i) a_{ij} \right] b_j(x_{t+1})$$

$$\text{Termination: } \beta_0 = P(X \mid \lambda) = \sum_{i=1}^N \pi_i b_i(x_1) \beta_1(i)$$

4. The backward algorithm

- The termination step can be visualized as: $\beta_0 = P(X \mid \lambda) = \sum_{i=1}^N \pi_i b_i(x_1) \beta_1(i)$



4. The backward algorithm

- NB: We can easily notice that: $\beta_0 = \alpha_E = P(X \mid \lambda)$

5. Transition matrix estimation

- The forward and the backward algorithms are used to isolate states within the HMM
- These variable let us estimate transition probabilities between states, and the distribution of the observations in the states

5. Transition matrix estimation

- When training an HMM, say that we want to estimate A , the transition matrix first.
In that case :

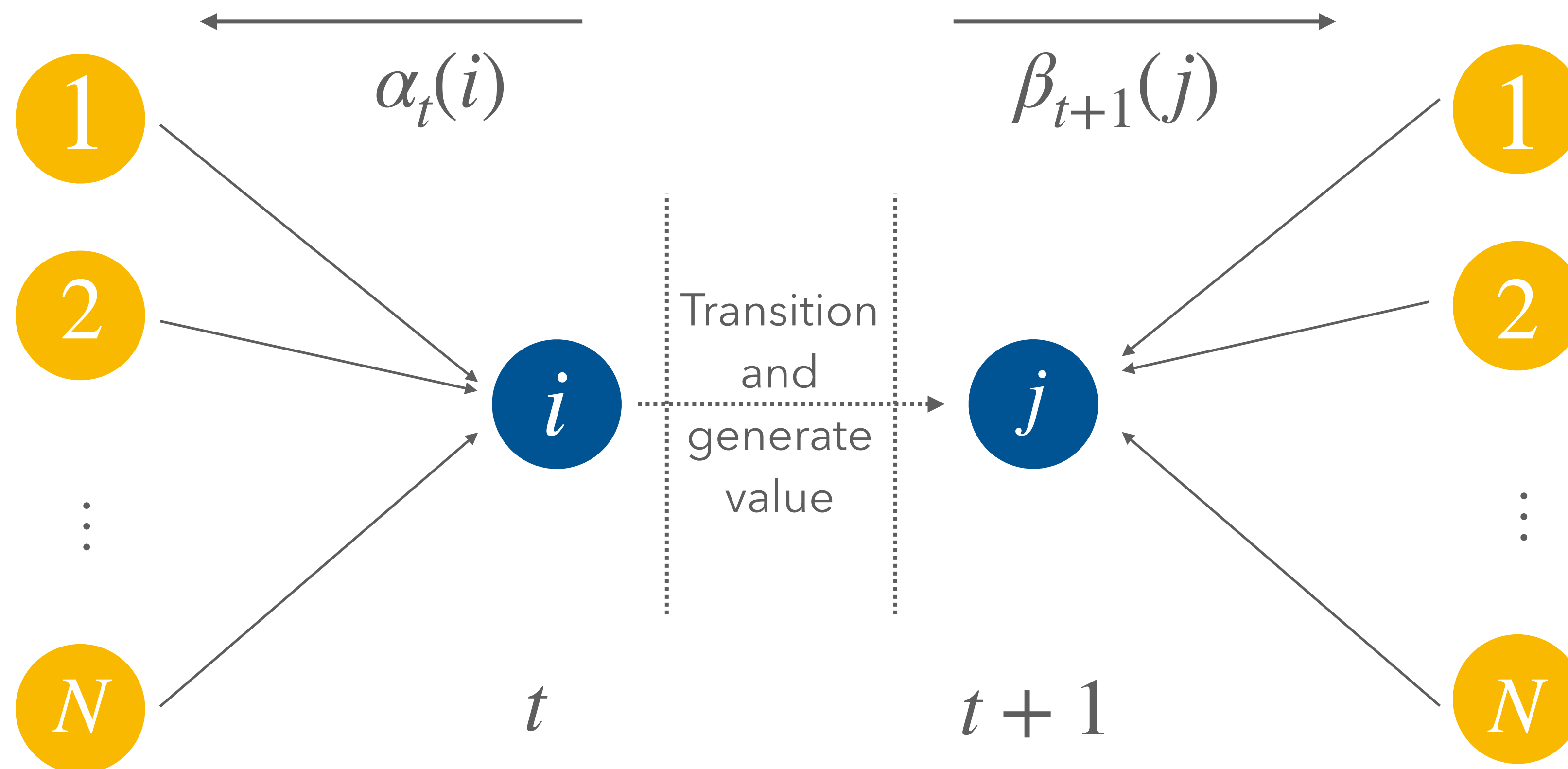
$$\hat{a}_{ij} = \frac{E(\text{Number of transitions from state } i \text{ to } j)}{E(\text{Number of transitions from state } i)}$$

- We can introduce $\xi_t(i, j)$ as being the probability of being in state i at time t and in state j at time $t+1$:

$$\xi_t(i, j) = P(q_t = i, q_{t+1} = j \mid X, \lambda)$$

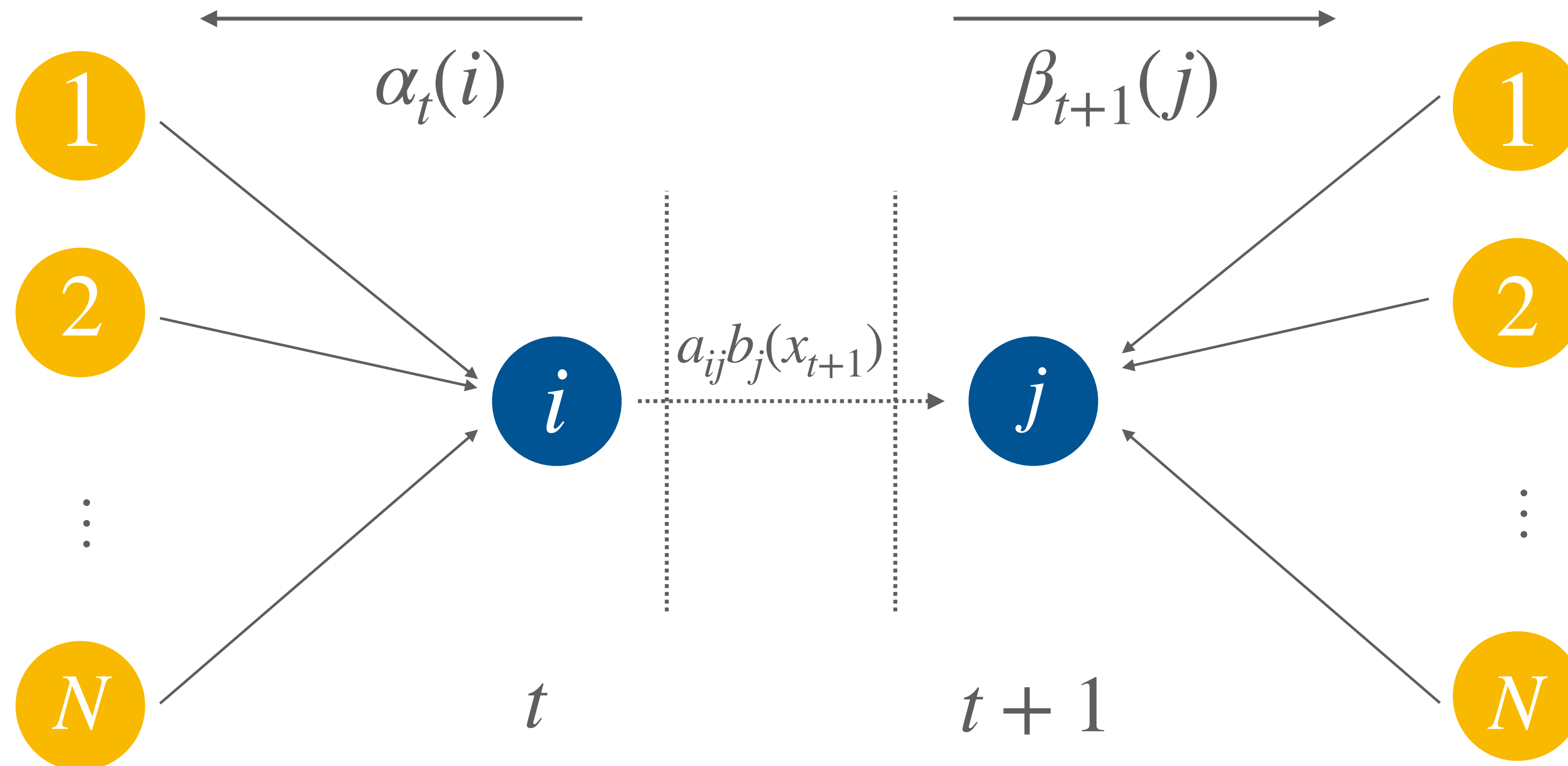
5. Transition matrix estimation

- What's convenient is that $\xi_t(i, j)$ can be expressed in terms of the forward and backward variables: $\xi_t(i, j) = P(q_t = i, q_{t+1} = j \mid X, \lambda)$



5. Transition matrix estimation

- What's convenient is that $\xi_t(i, j)$ can be expressed in terms of the forward and backward variables: $\xi_t(i, j) = P(q_t = i, q_{t+1} = j \mid X, \lambda)$



5. Transition matrix estimation

- Putting it together: $\xi_t(i, j) = P(q_t = i, q_{t+1} = j \mid X, \lambda)$

$$\xi_t(i, j) = \alpha_t(i) a_{ij} b_j(x_{t+1}) \beta_{t+1}(j) / P(X \mid \lambda)$$

Reaching state i with
observation sequence

Moving to j

Generating value

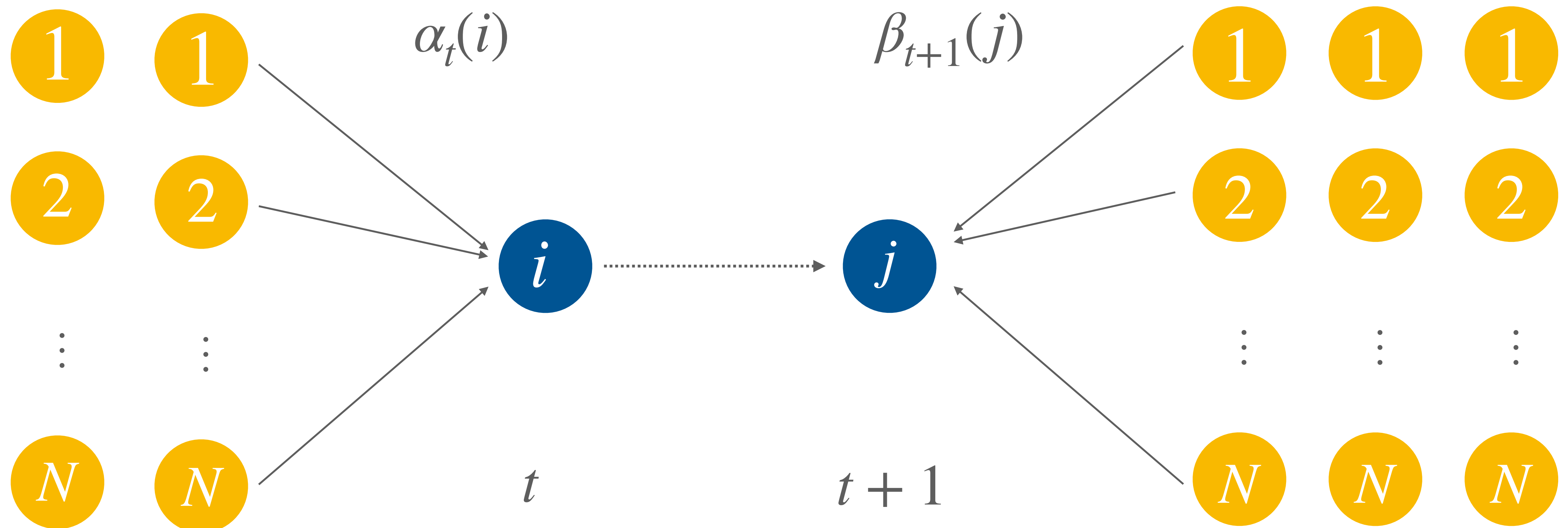
Rest of observation
sequence starting from j

$$\xi_t(i, j) = \frac{\alpha_t(i) a_{ij} b_j(x_{t+1}) \beta_{t+1}(j)}{\sum_{j=1}^N \alpha_t(j) \beta_t(j)}$$

Expected number of
transitions from i to j at time t

5. Transition matrix estimation

- All that we have left to do to estimate \hat{a}_{ij} is to sum over all possible times t



5. Transition matrix estimation

- All that we have left to do to estimate \hat{a}_{ij} is to sum over all possible times t

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \sum_{k=1}^N \xi_t(i, k)}$$

Transition at all times between i and j

Transition at all times between i and any other state

5. Transition matrix estimation

- And there we have it! We compute this for all pairs (i,j) and obtain the transition matrix A

$$\hat{A} = \begin{bmatrix} \hat{a}_{11} & \hat{a}_{12} & \hat{a}_{13} \\ \hat{a}_{21} & \hat{a}_{22} & \hat{a}_{23} \\ \hat{a}_{31} & \hat{a}_{32} & \hat{a}_{33} \end{bmatrix}$$

6. Observation probability

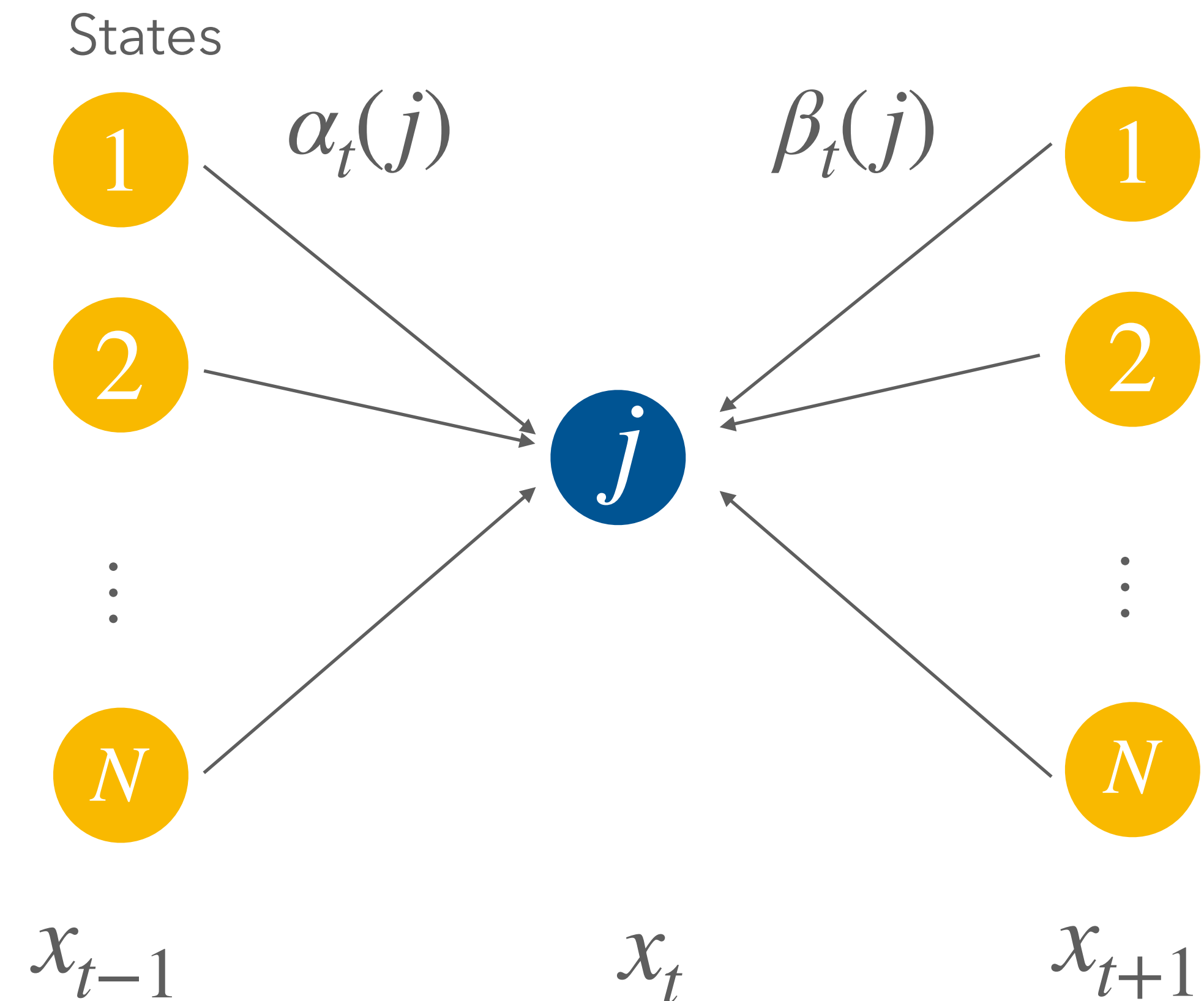
- The first part of the work is now done. Recall that we are in a discrete HMM framework, and our next task will be to estimate the observation probability. We suppose that at each state, we can have an observation x that is a discrete value in the « observation vocabulary » V .
- A single observation from V could be v_k
- Therefore, to characterize our observation probability, we need to compute:

$$\hat{b}_j(v_k) = \frac{E(\text{Number of times in state } j \text{ and observing } v_k)}{E(\text{Number of times in state } j)}$$

6. Observation probability

- We first need to define the probability of being in state j at time t :

$$\begin{aligned}
 \gamma_j(t) &= P(q_t = j \mid X, \lambda) \\
 &\quad \begin{array}{c} \downarrow \text{At time } t \\ \downarrow \text{State } j \end{array} \\
 &= \frac{P(q_t = j, X \mid \lambda)}{P(X \mid \lambda)} \\
 &= \frac{\alpha_t(j)\beta_t(j)}{P(X \mid \lambda)}
 \end{aligned}$$



6. Observation probability

- Deriving $\hat{b}_j(v_k)$ is then straightforward !

$$\hat{b}_j(v_k) = \frac{\sum_{t=1}^T s_{s.t. x_t=v_k} \gamma_t(j)}{\sum_{t=1}^T \gamma_t(j)}$$

—————→ In state j, observe value v_k

—————→ In state j, observe any value

7. Initial state probability

- The simplest value to estimate remains the initial state probability, noted as:

$$\hat{\pi}_j = \gamma_1(j)$$

7. Baum-Welch Algorithm

- With the new values of \hat{a}_{ij} and $\hat{b}_j(v_k)$, we can re-compute $\alpha_t, \beta_t, \gamma_t, \xi_t$, leading to new values of \hat{a}_{ij} and $\hat{b}_j(v_k)$... This is the forward-backward (Baum Welch) algorithm and relies on EM.

Initialize A and B

Iterate until convergence

E-Step

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{\alpha_T(q_F)}$$

$$\xi_t(i, j) = \frac{\alpha_t(i)a_{ij}b_j(x_{t+1})\beta_{t+1}(j)}{\sum_{j=1}^N \alpha_t(j)\beta_t(j)}$$

M-Step

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \sum_{k=1}^N \xi_t(i, k)}$$

$$\hat{b}_j(v_k) = \frac{\sum_{t=1}^T s_{s.t. O_t=v_k} \gamma_t(j)}{\sum_{t=1}^T \gamma_t(j)}$$

8. Gaussian observations

- Now, let's relax the discrete observation assumption and let the observations in each state be generated by Gaussians. In that case, we are not interested in the observation probability, but in the mean and covariance of each Gaussian. In the EM, the \hat{b}_j is replaced by:

$$\hat{\mu}_j = \frac{\sum_{t=1}^T \gamma_j(t) x_t}{\sum_{t=1}^T \gamma_j(t)}$$

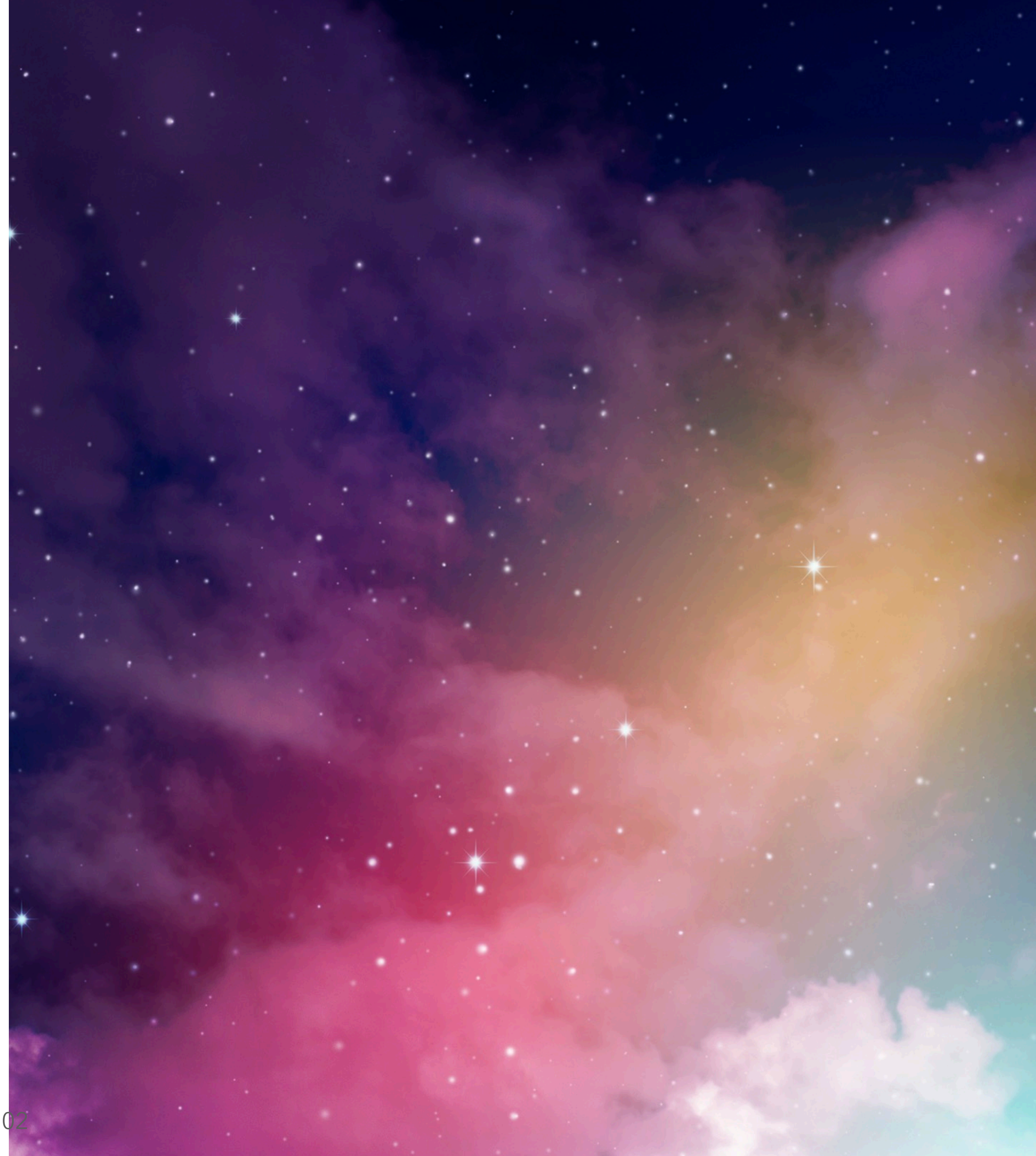
Mean

Weighted average of the value of the observation by the state probability

$$\hat{\Sigma}_j = \frac{\sum_{t=1}^T \gamma_j(t) (x_t - \hat{\mu}_j) (x_t - \hat{\mu}_j)^T}{\sum_{t=1}^T \gamma_j(t)}$$

Covariance

IV. Viterbi training

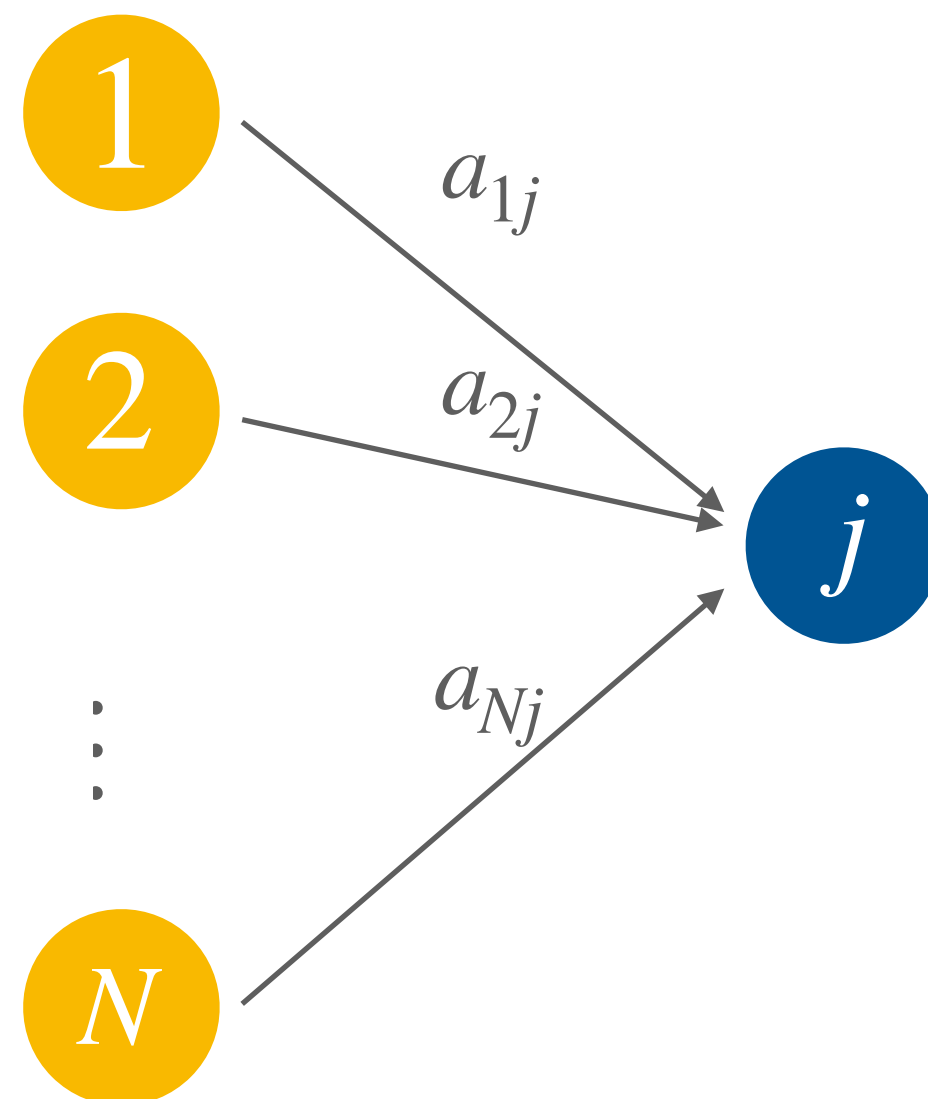


1. Modified forward algorithm

- In Viterbi training, we do not worry about summing over all possible paths, but only on keeping the most likely. Therefore, we can re-define the forward variable as being:

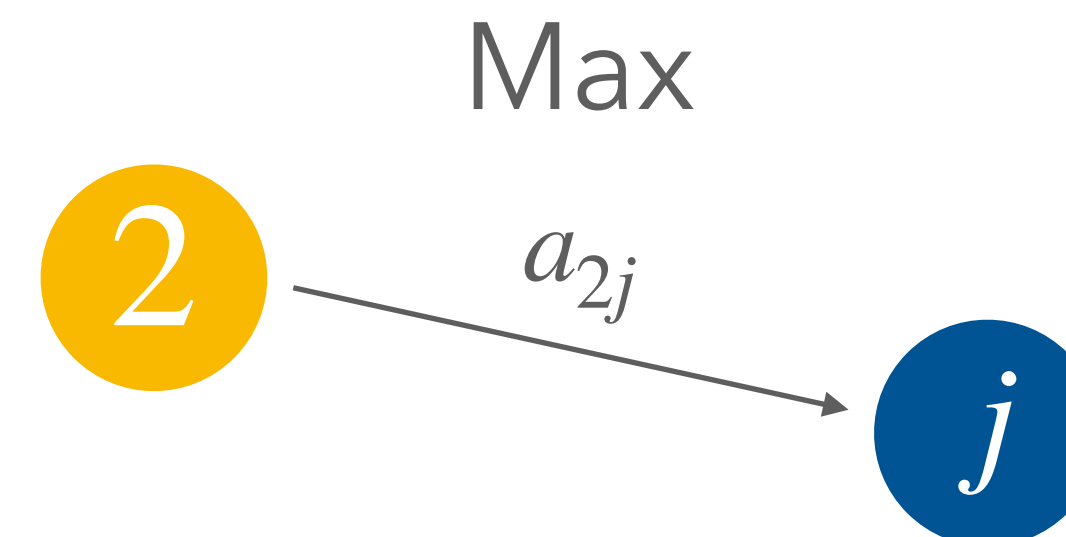
Baum-Welch

$$\alpha_t(j) = P(x_1, x_2, \dots, x_t, q_t = j \mid \lambda)$$



Viterbi

$$\delta_t(j) = \max_{q_1, \dots, q_{t-1}} P(x_1, x_2, \dots, x_t, q_1, q_2, \dots, q_t = j \mid \lambda)$$



1. Modified forward algorithm

- We can compute the probability of an observation sequence in the following manner:

$$\delta_t(j) = \max_{q_1, \dots, q_{t-1}} P(x_1, x_2, \dots, x_t, q_1, q_2, \dots, q_t = j \mid \lambda)$$

Initialization: $\delta_1(i) = \pi_i b_i(x_1)$

Iteration: $\delta_{t+1}(j) = \left[\max_i \delta_t(i) a_{ij} \right] b_j(x_{t+1})$

Termination: $\delta_E = P(X \mid \lambda) = \max_i \delta_T(i)$

2. Viterbi Training

- It is then easy to determine the best state-sequence. From that sequence, we can estimate the updated parameters for our Gaussian HMM:

$\hat{a}_{ij} = \frac{C(i \rightarrow j)}{\sum_k C(i \rightarrow k)}$ <p style="text-align: center;">↑ Count of transition from i-to-k</p>	$\hat{\mu}_j = \frac{\sum_{x \in Z_j} x}{ Z_j }$ <p style="text-align: center;">↓ Estimated mean</p> <p style="text-align: center;">-----> Set of observed features assigned to state 'j'</p>	$\hat{\Sigma}_j = \frac{\sum_{x \in Z_j} (x - \hat{\mu}_j)(x - \hat{\mu}_j)^T}{ Z_j }$ <p style="text-align: center;">↓ Estimated covariance</p>
--	--	--

- And these new parameters are used as new inputs of the forward variable δ_t . We perform these tasks iteratively, and this is the Viterbi Training.

2. Viterbi Training

- Notice how the update parameters are similar to the ones with Full EM except that we use hard priors instead of soft ones.

Full EM

$$\hat{\mu}_j = \frac{\sum_{t=1}^T \gamma_j(t) x_t}{\sum_{t=1}^T \gamma_j(t)}$$

↑
Soft posteriors

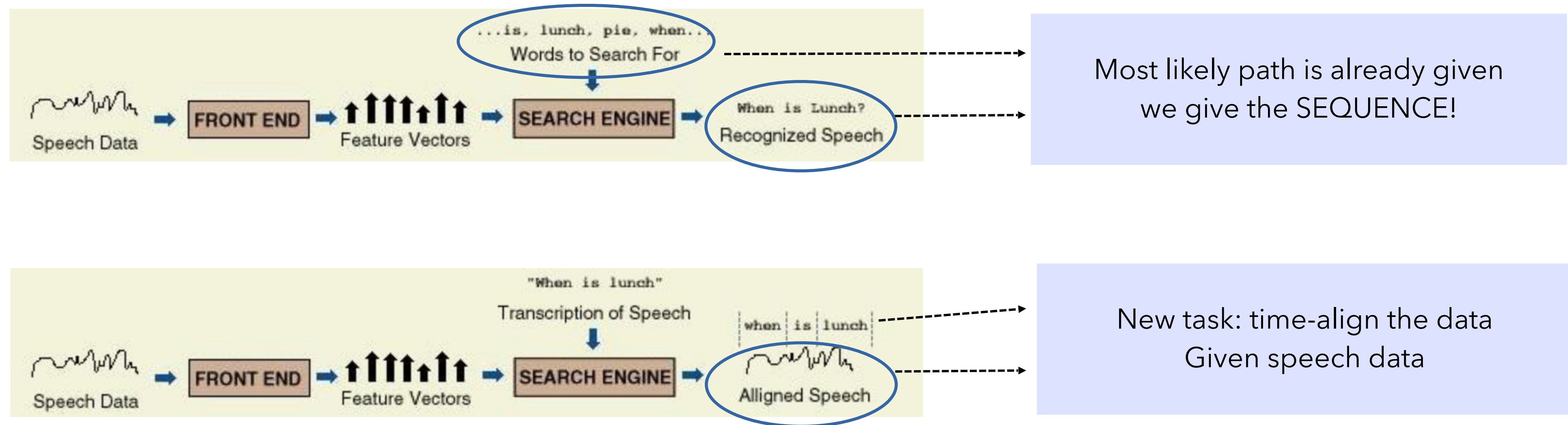
Viterbi Training

$$\hat{\mu}_j = \frac{\sum_{t=1}^T \gamma_j(t) x_t}{\sum_{t=1}^T \gamma_j(t)}$$

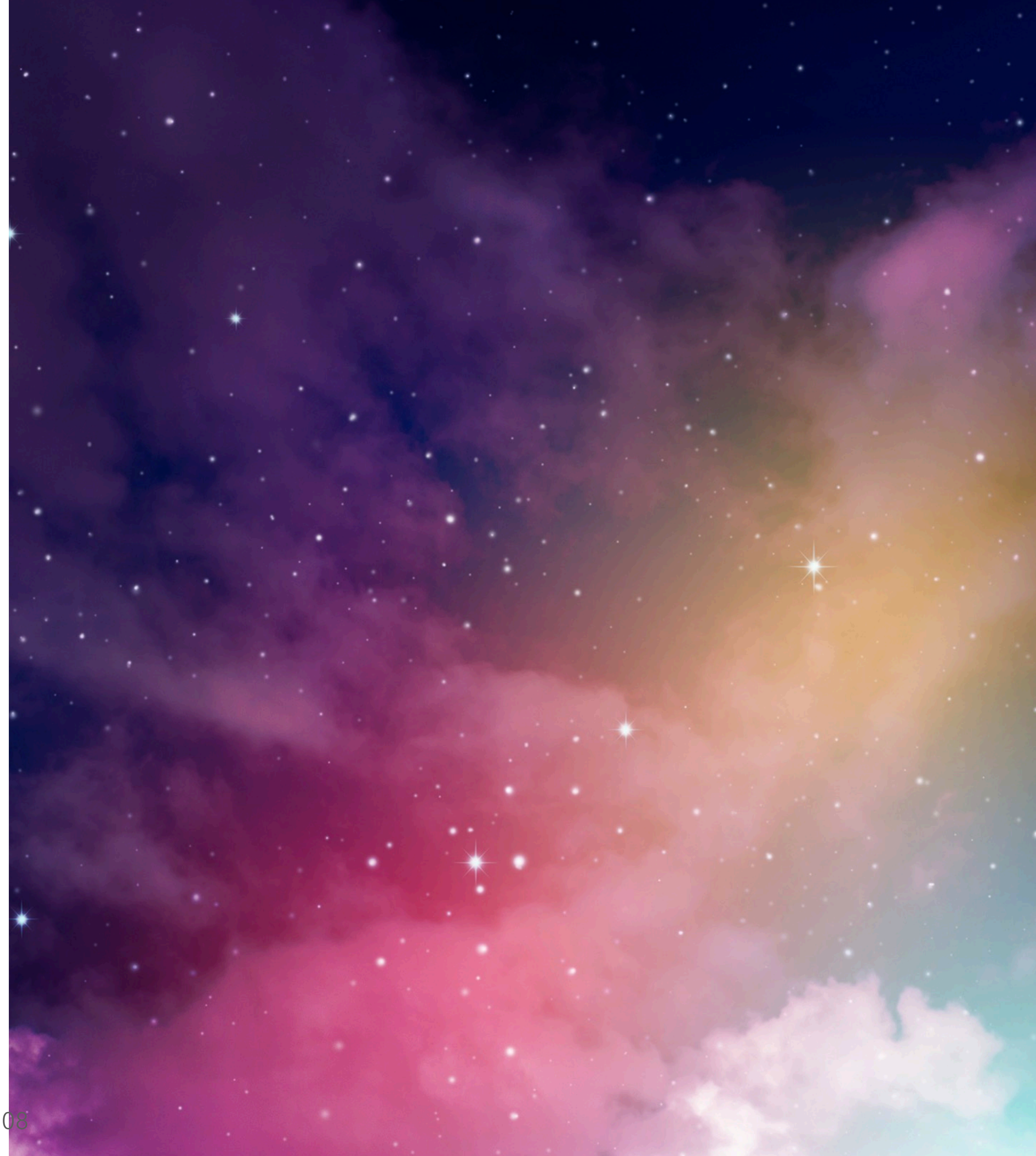
↑
= 1 if in state with maximal
probability, 0 otherwise

2. Viterbi Training

- Example: Embedded Viterbi - Alignments
- Viterbi yields the most likely path that could contain :
 - Acoustic and language model
 - Set of possible words, phonemes or some representation



V. HMM-GMM training



1. HMM-GMM training

- We now define the occupation probability $\gamma_{jm}(t)$ as the probability of occupying mixture component m of state j at time t .

1. HMM-GMM training

We can re-estimate the parameters accordingly

Mean of mixture
component \underline{m} of state \underline{j} :

$$\hat{\mu}_{jm} = \frac{\sum_{t=1}^T \gamma_{jm}(t) x_t}{\sum_{t=1}^T \gamma_{jm}(t)}$$

Covariance matrix of mixture
component \underline{m} of state \underline{j} :

$$\hat{\Sigma}_{jm} = \frac{\sum_{t=1}^T \gamma_{jm}(t) (x_t - \hat{\mu}_{jm}) (x_t - \hat{\mu}_{jm})^T}{\sum_{t=1}^T \gamma_{jm}(t)}$$

Mixture coefficients of
component \underline{m} of state \underline{j} :

$$\hat{c}_{jm} = \frac{\sum_{t=1}^T \gamma_{jm}(t)}{\sum_{\hat{m}=1}^M \sum_{t=1}^T \gamma_{j\hat{m}}(t)}$$

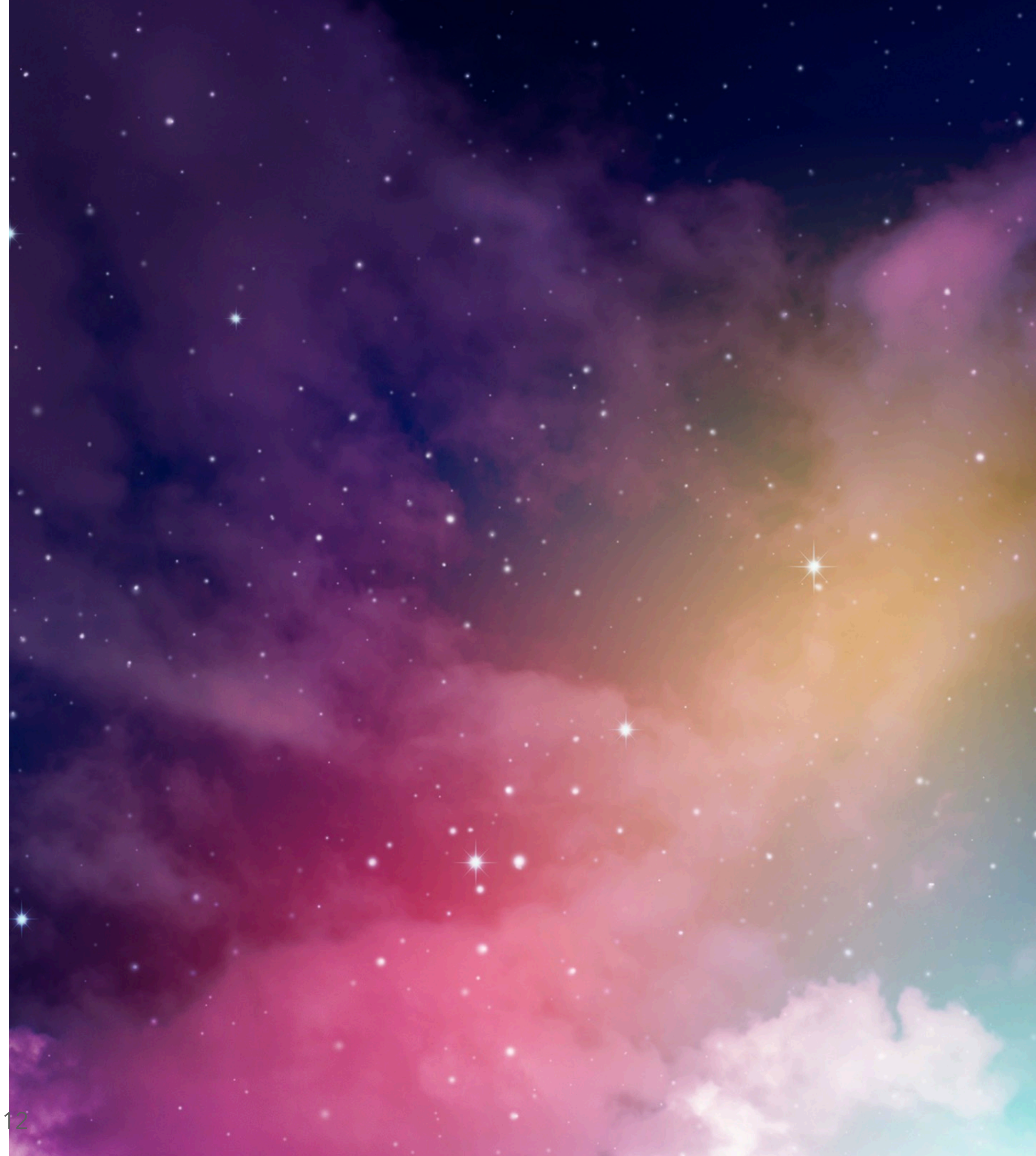
↓
Estimation in a similar way to
'transition probabilities'

2. Computation constraints

The forward-backward and Viterbi recursions result in a long sequences of probabilities being multiplied:

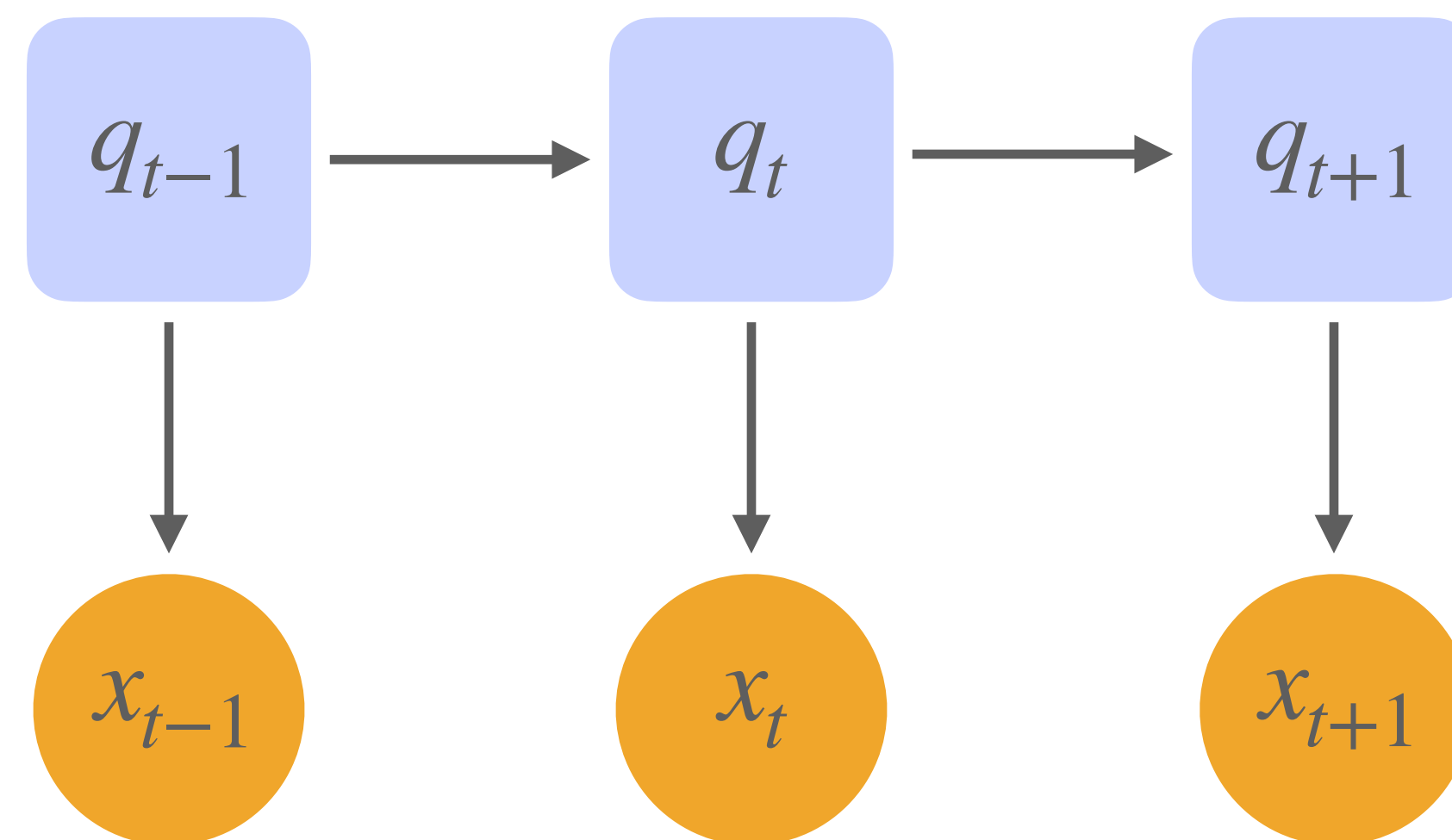
- Could cause floating point underflow problems
- To solve it → calculations are done on log domain
- Log domain: when computing Gaussians the logarithmic power rule transform the power terms in simple summation

VI. HMM applications



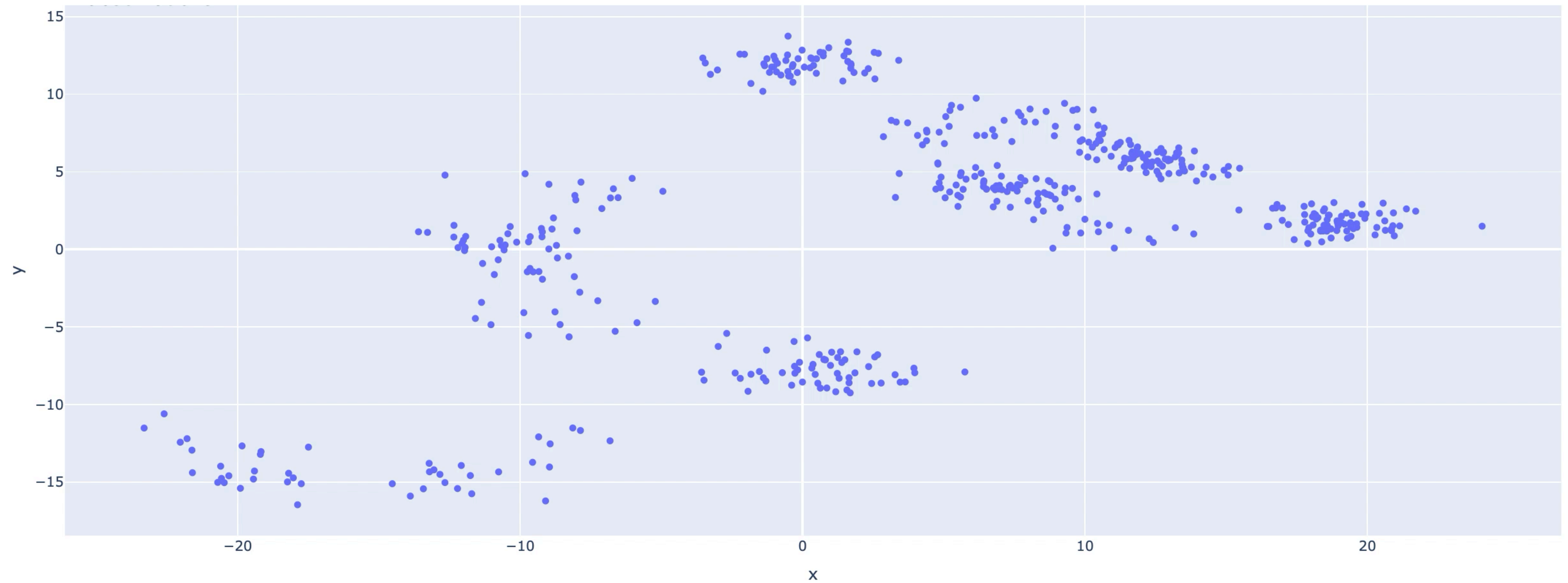
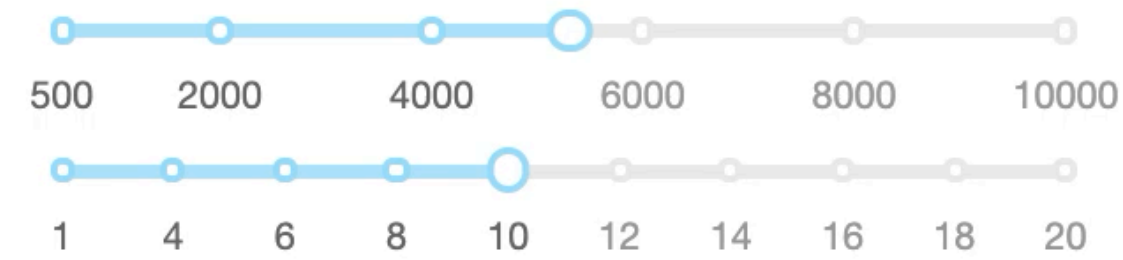
1. Finding states and parameters from observations

- In this first example, we generate data from a Markov Chain (fixed transition probabilities). This gives us a sequence of states. At each state, we generate an observation following a GMM with 6 components. We have 3 states, and therefore 3 GMMs with pre-determined parameters (means, covariances and weights).



1. Finding states and parameters from observations

Length of generated sequence
Number of iterations for HMM training



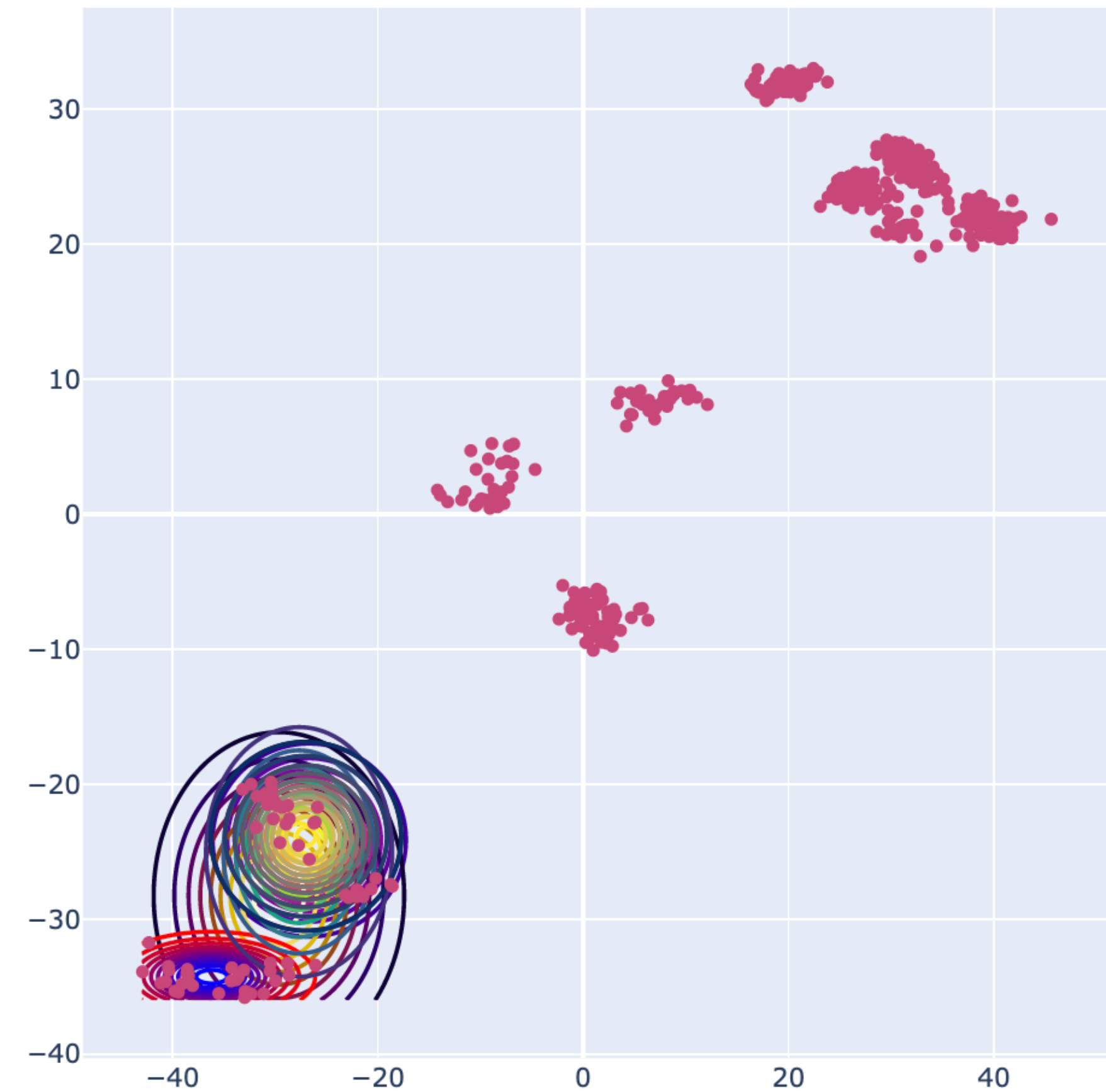
1. Finding states and parameters from observations

- We then are able to train the HMM-GMM using « hmmlearn » library. This heads the predicted transition matrix and the parameters of each GMM. Using the means, the covariances and the weights of the GMM, we can generate the contour plots of each GMM. This is then displayed in an interactive module.

1. Finding states and parameters from observations

Show the GMMs belonging to state:

GMM 1

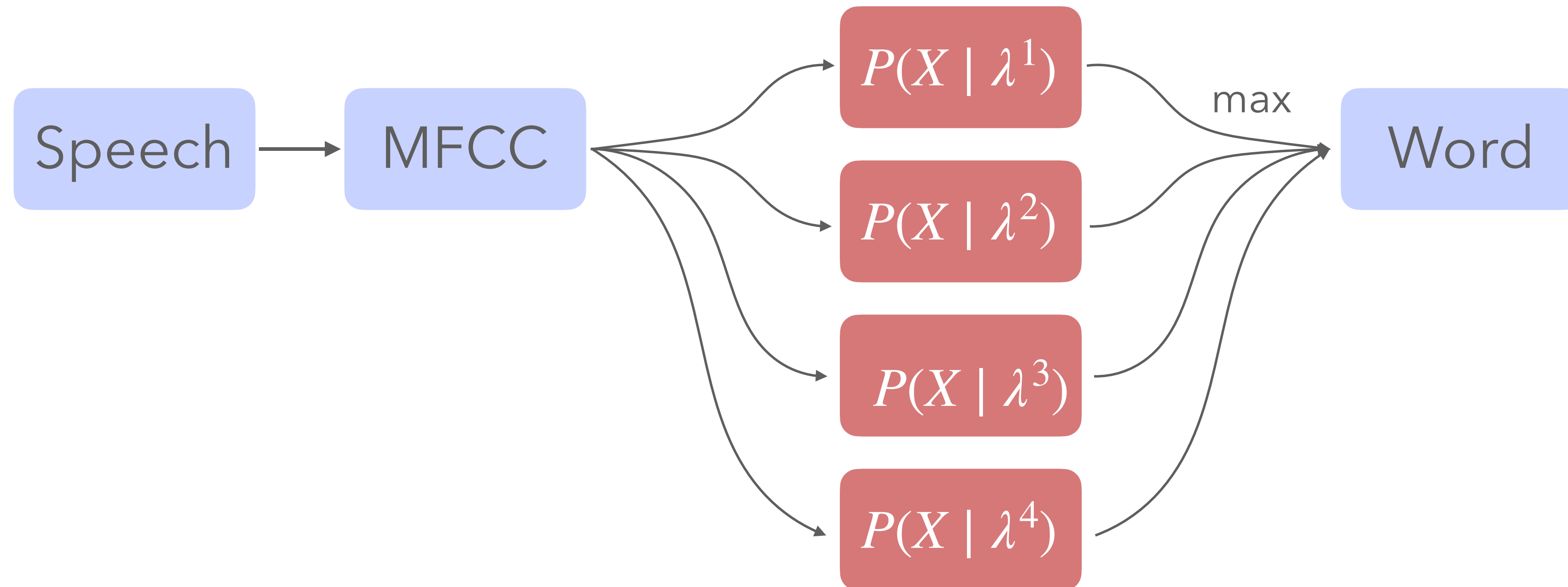


2. Isolated Word Recognition

- HMMs are widely used in Speech Recognition tasks. For example, in Isolated Word Recognition, we assume a vocabulary of size V , and we model each word in the vocabulary by a distinct HMM. We assume that we have K occurrences of each spoken word as training.
- The Speech Recognition system is made of the following steps:
 - Extract features (typically MFCCs) for each training sample
 - Train an HMM λ for each of the words in the vocabulary (estimate A, B, π)
 - For each sample in test, extract features and estimate likelihood to belong to each HMM : $v^{\star} = \operatorname{argmax}_{1 \leq v \leq V} P(O \mid \lambda^v)$

2. Isolated Word Recognition

- Typically, each HMM has 5 states. The isolated word recognition pipeline can be presented as such:



2. Isolated Word Recognition

- In the following example, we will build a single **digit** recognizer using open-source training data (available from <https://github.com/moebg/spoken-digit-recognition/tree/master/data/recordings>)
- The whole code is presented in appendix 3.

2. Isolated Word Recognition

- This function is used to train the GMM-HMM:

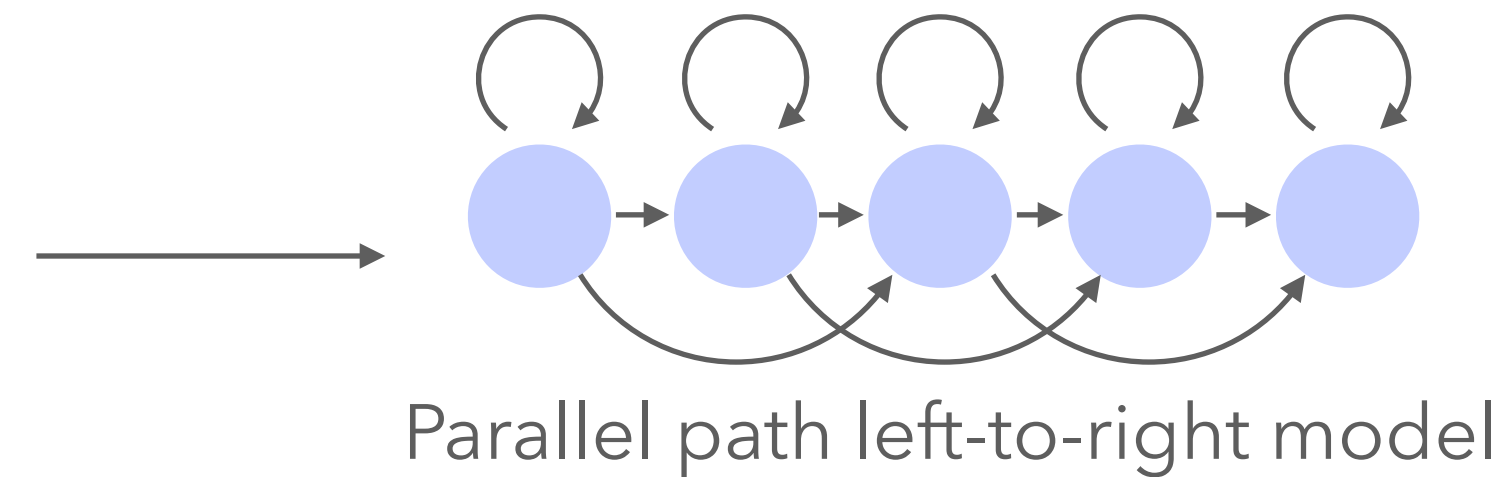
```
def train_GMMHMM(dataset):
    GMMHMM_Models = {}
    states_num = 5
    GMM_mix_num = 6
    tmp_p = 1.0/(states_num-2)
    transmatPrior = np.array([[tmp_p, tmp_p, tmp_p, 0, 0], \
                               [0, tmp_p, tmp_p, tmp_p, 0], \
                               [0, 0, tmp_p, tmp_p, tmp_p], \
                               [0, 0, 0, 0.5, 0.5], \
                               [0, 0, 0, 0, 1]], dtype=np.float)

    startprobPrior = np.array([0.5, 0.5, 0, 0, 0], dtype=np.float)

    for label in dataset.keys():
        model = hmm.GMMHMM(n_components=states_num, n_mix=GMM_mix_num, \
                           transmat_prior=transmatPrior, startprob_prior=startprobPrior, \
                           covariance_type='diag', n_iter=10)


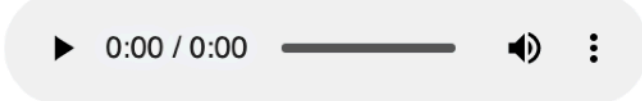

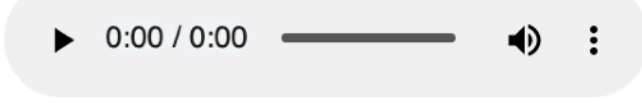
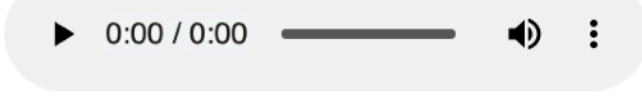
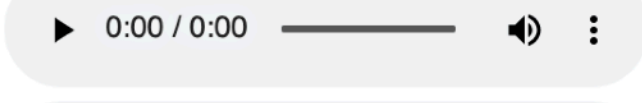
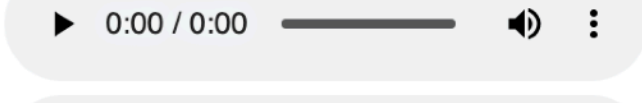
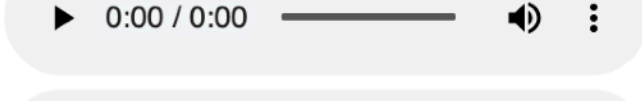
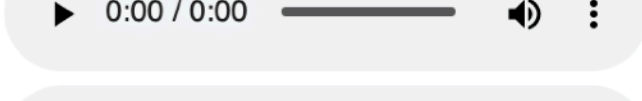
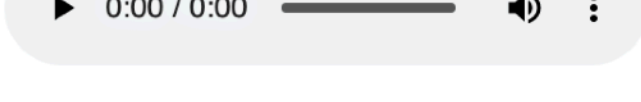
        trainData = dataset[label]
        length = np.zeros([len(trainData)], dtype=np.int)
        for m in range(len(trainData)):
            length[m] = trainData[m].shape[0]
        trainData = np.vstack(trainData)
        model.fit(trainData, lengths=length)
        GMMHMM_Models[label] = model

    return GMMHMM_Models
```



2. Isolated Word Recognition

- The training data is quite heterogenous, and the sound would need better preprocessing. However, the accuracy achieved varies between 30 and 60%, far from the state-of-the-art, but it's still a useful example.

Recording	Audio File	ASR Prediction
0		8
1		1
2		2
3		3
4		4
5		5
6		8
7		7
8		8
9		9

3. Other examples

- Other examples are mentioned in appendix 4:
 - Animal movement prediction with HMMs
 - Playlist prediction with HMMs
 - Person finding using HMMs

Thank you for your attention
Questions?

References

References (1)

- MLE of single Gaussian, <http://jrmeyer.github.io/machinelearning/2017/08/18/mle.html>
- MLE of GMMs, https://stephens999.github.io/fiveMinuteStats/intro_to_em.html
- EM algorithm and variants: an informal tutorial, *Alexis Roche*
- (Hard) Expectation Maximization, *David McAllester*, <https://ttic.uchicago.edu/~dmcallester/ttic101-07/lectures/em/em.pdf>
- Short Note on EM, *Brendan O'Connor*, https://www.cs.cmu.edu/~tom/10601_fall2012/recitations/em.pdf
- Vector Quantization, *David Forsyth*, <http://luthuli.cs.uiuc.edu/~daf/courses/CS-498-DAF-PS/Lecture%2012%20-%20K-means,%20GMMs,%20EM.pdf>
- Background subtraction with GMMs, *D. Hari Hara Santosh, P. Venkatesh, P. Poornesh, L. Narayana Rao, N. Arun Kumar*, <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.649.8642&rep=rep1&type=pdf>

References (2)

- A tutorial on hidden Markov models and selected applications in speech recognition, *Lawrence R. Rabiner*, <https://www.ece.ucsb.edu/Faculty/Rabiner/ece259/Reprints/tutorial%20on%20hmm%20and%20applications.pdf>
- Isolated digit recognition adapted from : https://github.com/wblgers/hmm_speech_recognition_demo
- Westmont College Introduction to Forward-Backward, *Patterson*, <https://www.youtube.com/watch?v=gYma8Gw38Os>
- Hidden Markov Models, Speech and Language Processing, *Daniel Jurafsky & James H. Martin*. <https://web.stanford.edu/~jurafsky/slp3/A.pdf>

References (3)

- Ellis, D. P. (2008). An introduction to signal processing for speech.
- Peter Bell. (2020). Automatic Speech Recognition (ASR) 2019-20: Lectures. University of Edinburgh.
- Daniel J. and James H. Martin. (2020). Speech and Language Processing, draft 3rd Edition
- Application-(a): Adam, T., Griffiths, et al. (2019). Joint modelling of multi-scale animal movement data using hierarchical hidden Markov models. *Methods in Ecology and Evolution*, 10(9), 1536-1550.
- Application-(b): Li, T., Choi, M., Fu, K., & Lin, L. (2018). Music sequence prediction with mixture hidden markov models. *arXiv preprint arXiv:1809.00842*.
- Application-(c): Bayoumi, A., Karkowski, P., & Bennewitz, M. (2019). Speeding up person finding using hidden Markov models. *Robotics and Autonomous Systems*, 115, 40-48.

Appendix

1. Where does the auxiliary function come from?

- The origin of the auxiliary function is linked to the Jensen's inequality:

$$\begin{aligned} L(\theta) - L(\theta^{(t)}) &= \log \frac{p(X | \theta)}{p(X | \theta^{(t)})} = \log \int \frac{p(Z, X | \theta)}{p(X | \theta^{(t)})} dz \\ &= \log \int \frac{p(Z, X | \theta)}{p(Z, X | \theta^{(t)})} P(Z | X, \theta^{(t)}) dz = \log \int \frac{p(Z | \theta)}{p(Z | \theta^{(t)})} P(Z | X, \theta^{(t)}) dx \\ &\geq \underbrace{\int \log \frac{p(Z | \theta)}{p(Z | \theta^{(t)})} P(Z | X, \theta^{(t)}) dx}_{\text{Auxiliary function } Q(\theta, \theta^{(t)})} \end{aligned}$$

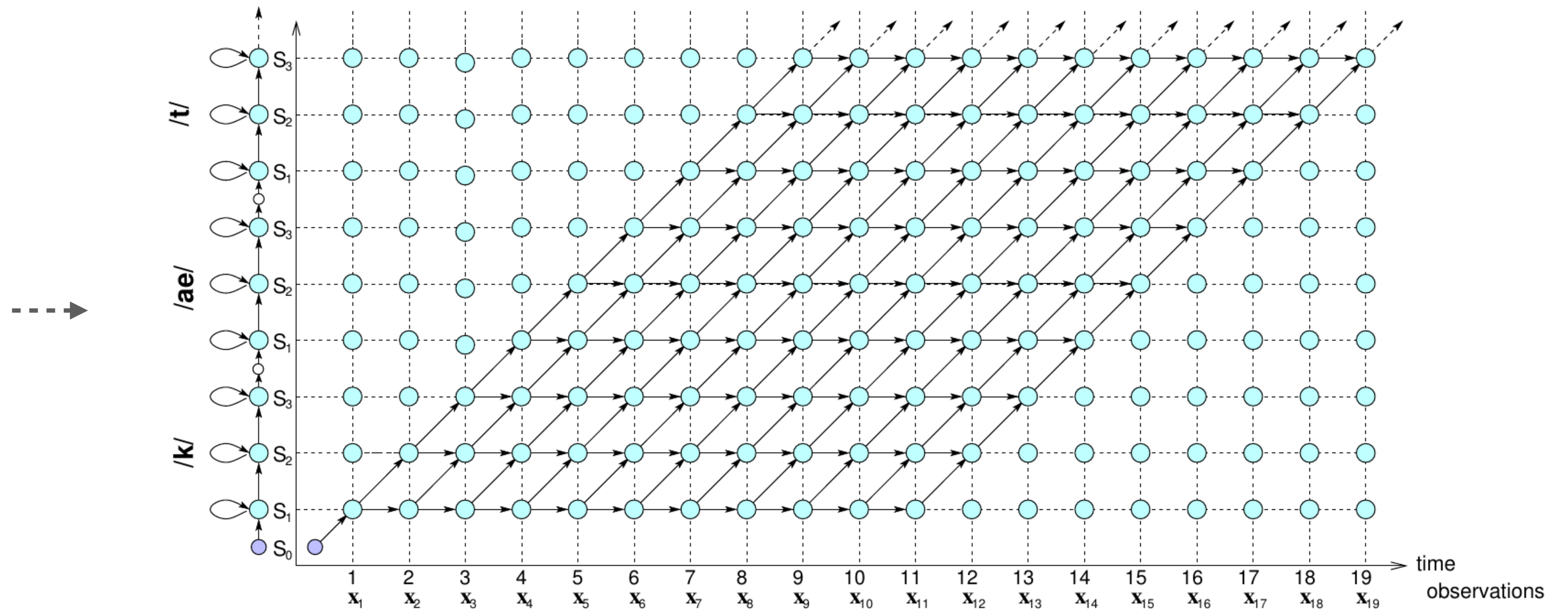
← Jensen's Inequality

The true likelihood variation is always **greater** than the variation of the auxiliary function

2. Cat word Trellis mesh

- Example of the Trellis mesh for the word CAT:

- Word mapped to phones
- 3-states for each phone
- y-axis: current state
- x-axis: observations



3. Isolated Word Recognition

- We can extract MFCCs in just a few lines of code:

```
import python_speech_features as mfcc
from scipy.io import wavfile
import os
import python_speech_features as mfcc
from scipy.io import wavfile
from hmmlearn import hmm
import numpy as np

def extract_mfcc(full_audio_path):
    sample_rate, wave = wavfile.read(full_audio_path)
    mfcc_features = mfcc.mfcc(wave, sample_rate, 0.025, 0.01, 20, appendEnergy = False)
    return mfcc_features
```

3. Isolated Word Recognition

- Then, we build a training data from the training samples folder

```
def build_data(dir):  
  
    fileList = [dir+f for f in os.listdir(dir) if os.path.splitext(f)[1] == '.wav']  
    dataset = {}  
    for fileName in fileList:  
        label = fileName.split("/")[-1].split('_')[0]  
        feature = extract_mfcc(fileName)  
        if label not in dataset.keys():  
            dataset[label] = []  
            dataset[label].append(feature)  
        else:  
            exist_feature = dataset[label]  
            exist_feature.append(feature)  
            dataset[label] = exist_feature  
  
    return dataset
```

3. Isolated Word Recognition

- Then comes the moment to train the GMMs for each label

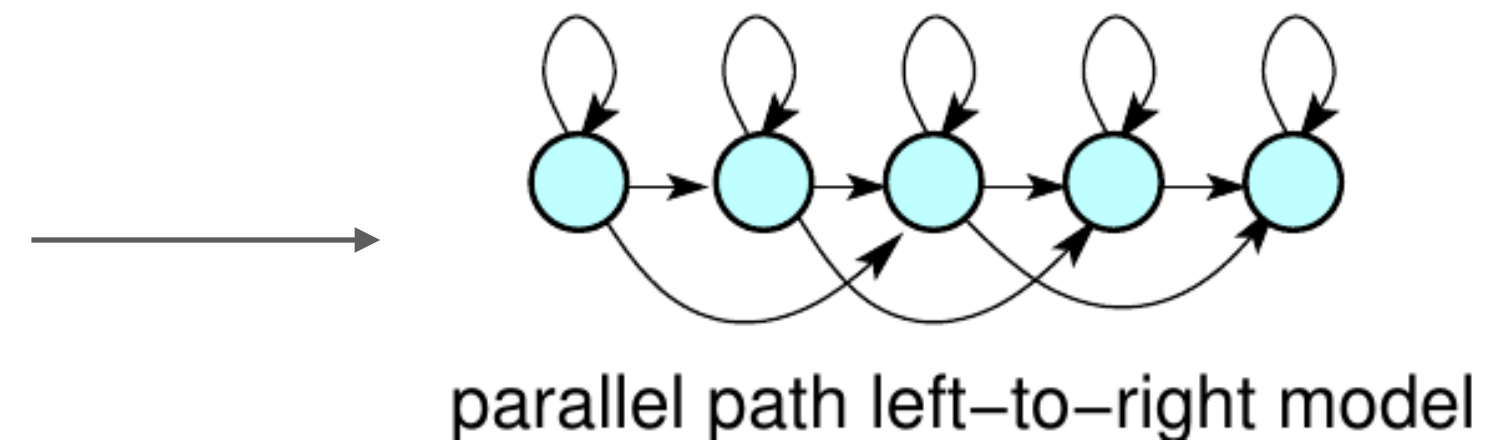
```
def train_GMMHMM(dataset):  
    GMMHMM_Models = {}  
    states_num = 5  
    GMM_mix_num = 6  
    tmp_p = 1.0/(states_num-2)  
    transmatPrior = np.array([[tmp_p, tmp_p, tmp_p, 0, 0], \  
                              [0, tmp_p, tmp_p, tmp_p, 0], \  
                              [0, 0, tmp_p, tmp_p, tmp_p], \  
                              [0, 0, 0, 0.5, 0.5], \  
                              [0, 0, 0, 0, 1]], dtype=np.float)
```

```
    startprobPrior = np.array([0.5, 0.5, 0, 0, 0], dtype=np.float)
```

```
    for label in dataset.keys():  
        model = hmm.GMMHMM(n_components=states_num, n_mix=GMM_mix_num, \  
                           transmat_prior=transmatPrior, startprob_prior=startprobPrior, \  
                           covariance_type='diag', n_iter=10)
```

```
        trainData = dataset[label]  
        length = np.zeros([len(trainData)], dtype=np.int)  
        for m in range(len(trainData)):  
            length[m] = trainData[m].shape[0]  
        trainData = np.vstack(trainData)  
        model.fit(trainData, lengths=length)  
        GMMHMM_Models[label] = model
```

```
    return GMMHMM_Models
```



3. Isolated Word Recognition

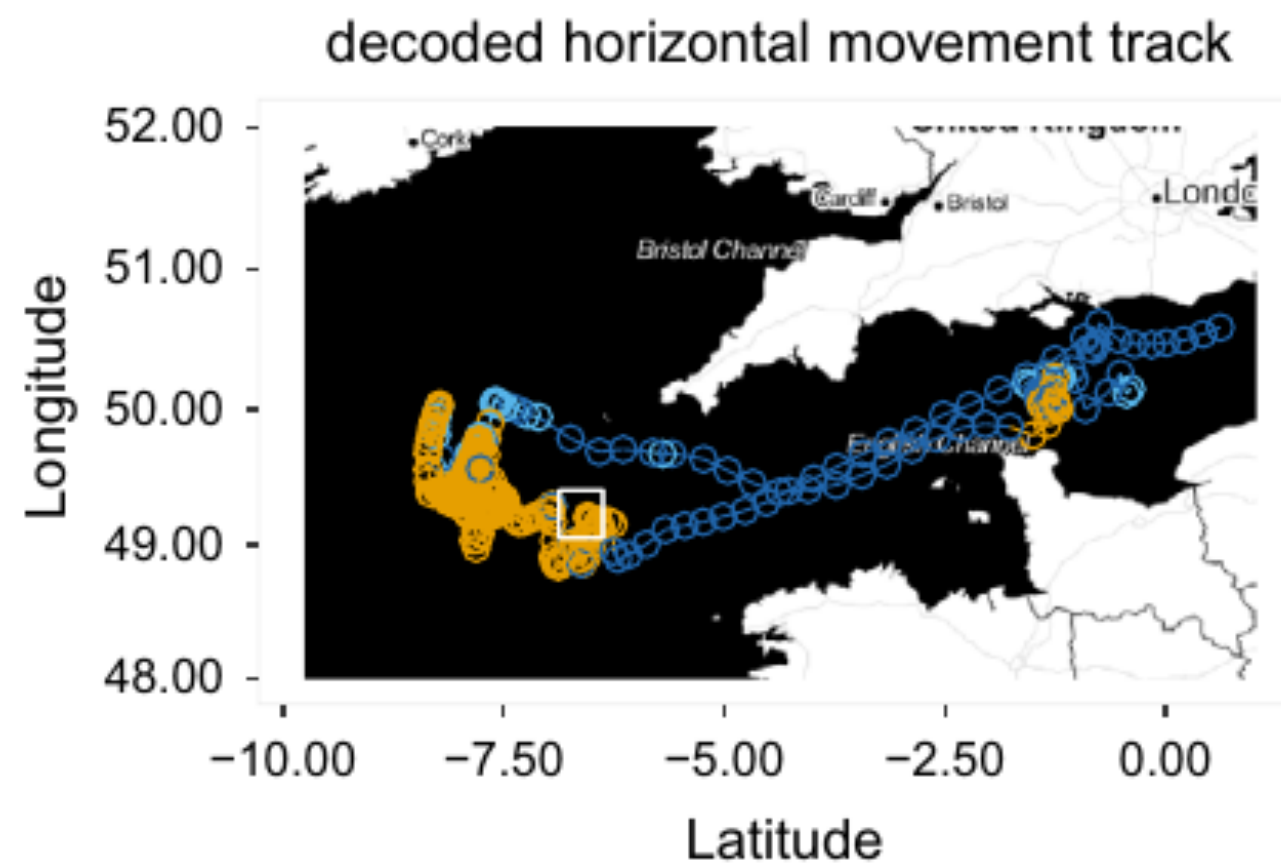
- Finally, we run the pipeline and estimate the accuracy of the system

```
def main():
    trainDir = './train_audio/'
    trainDataSet = build_data(trainDir)
    print("Finish prepare the training data")
    hmmModels = train_GMMHMM(trainDataSet)
    print("Finish training of the GMM_HMM models for digits 0-9")

    testDir = './test_audio/'
    testDataSet = build_data(testDir)

    score_cnt = 0
    for label in testDataSet.keys():
        feature = testDataSet[label]
        scoreList = {}
        for model_label in hmmModels.keys():
            model = hmmModels[model_label]
            score = model.score(feature[0])
            scoreList[model_label] = score
        predict = max(scoreList, key=scoreList.get)
        print("Test on true label ", label, ": predict result label is ", predict)
        if predict == label:
            score_cnt+=1
    print("Final recognition rate is %.2f"%(100.0*score_cnt/len(testDataSet.keys())), "%")
```


4. Selected practical applications of HMMs

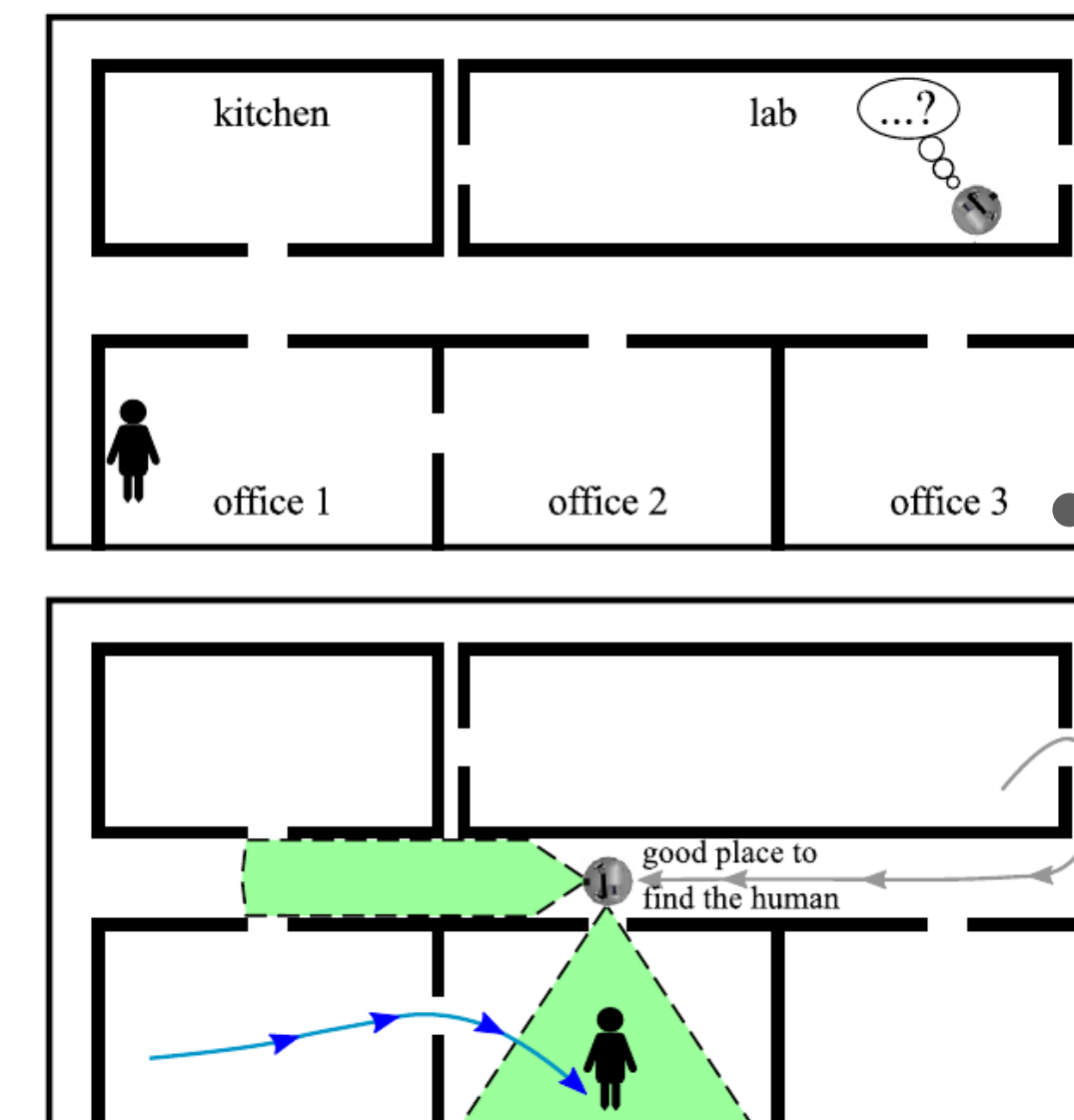


- (a) Animal movement modeling with HMMs



(b) Playlist prediction with HMMs

Applications



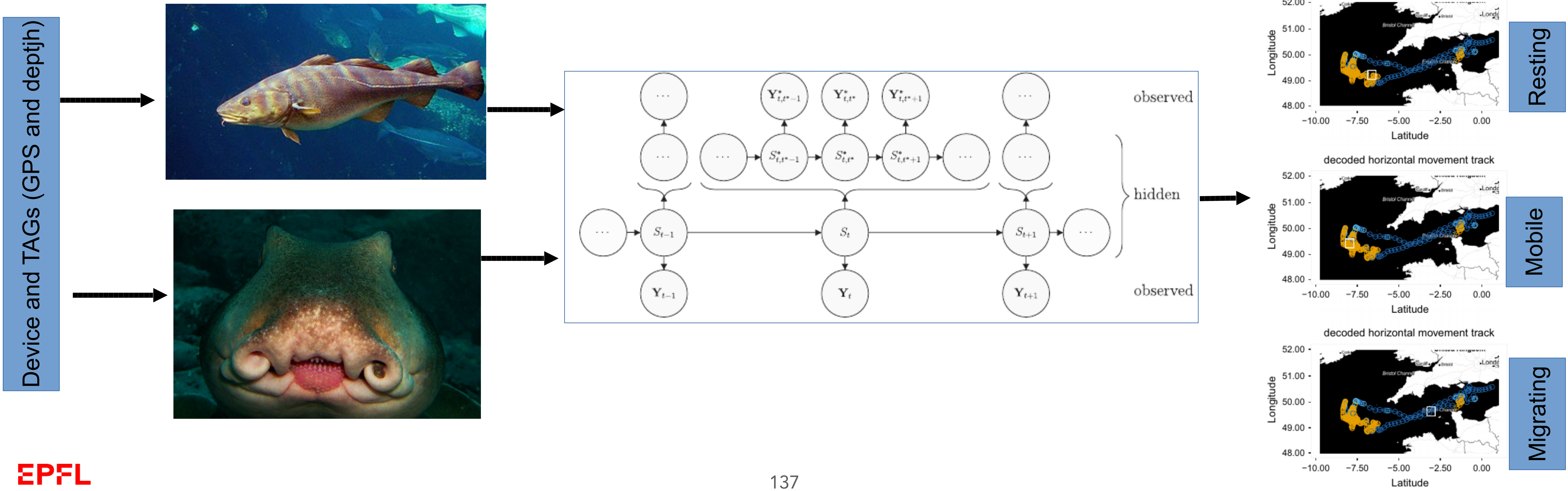
(c) Person finding with HMMs

4.a. Joint modeling of multi-scale animal movement data using hierarchical hidden Markov models

- They use a hierarchical hidden Markov model (HHMM) as a statistical framework to jointly model multi-temporal data streams, collected at different temporal resolutions:
 - HHMM: multi-level stochastic processes, each hidden state is a probabilistic model (emit sequences)
 - Big picture: movements patterns involving long-term vs short-term movement patterns
- Problem of initialization:
 - (1) Random range of different set of initial values
 - (2) pick largest likelihood
- To obtain the 'path': decoded with Viterbi Algorithm
- Input sequences:
 - Vertical movements: 10-min intervals (log-vertical movements)
 - Horizontal movements: 1-day intervals
- The idea is to know whether the fish is moving (3-states)
 - State 1: static behavior
 - State 2: moving (mobile) behavior
 - State 3: moving but more prone to 'horizontal' movement

4.a. Joint modeling of multi-scale animal movement data using hierarchical hidden Markov models

- Cod & Horn shark
- Hierarchical hidden Markov model
- Decoding the movement (Viterbi)



4.b. Music Sequence Prediction with Mixture Hidden Markov Models

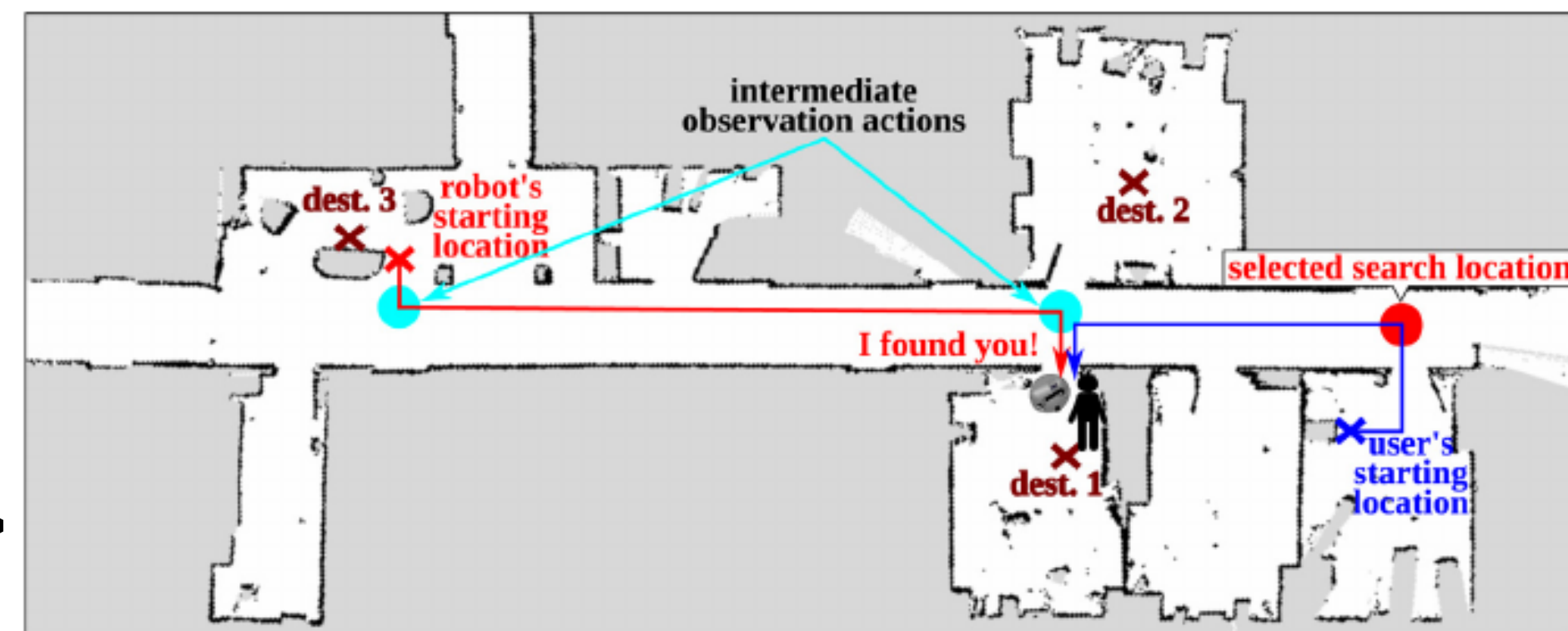
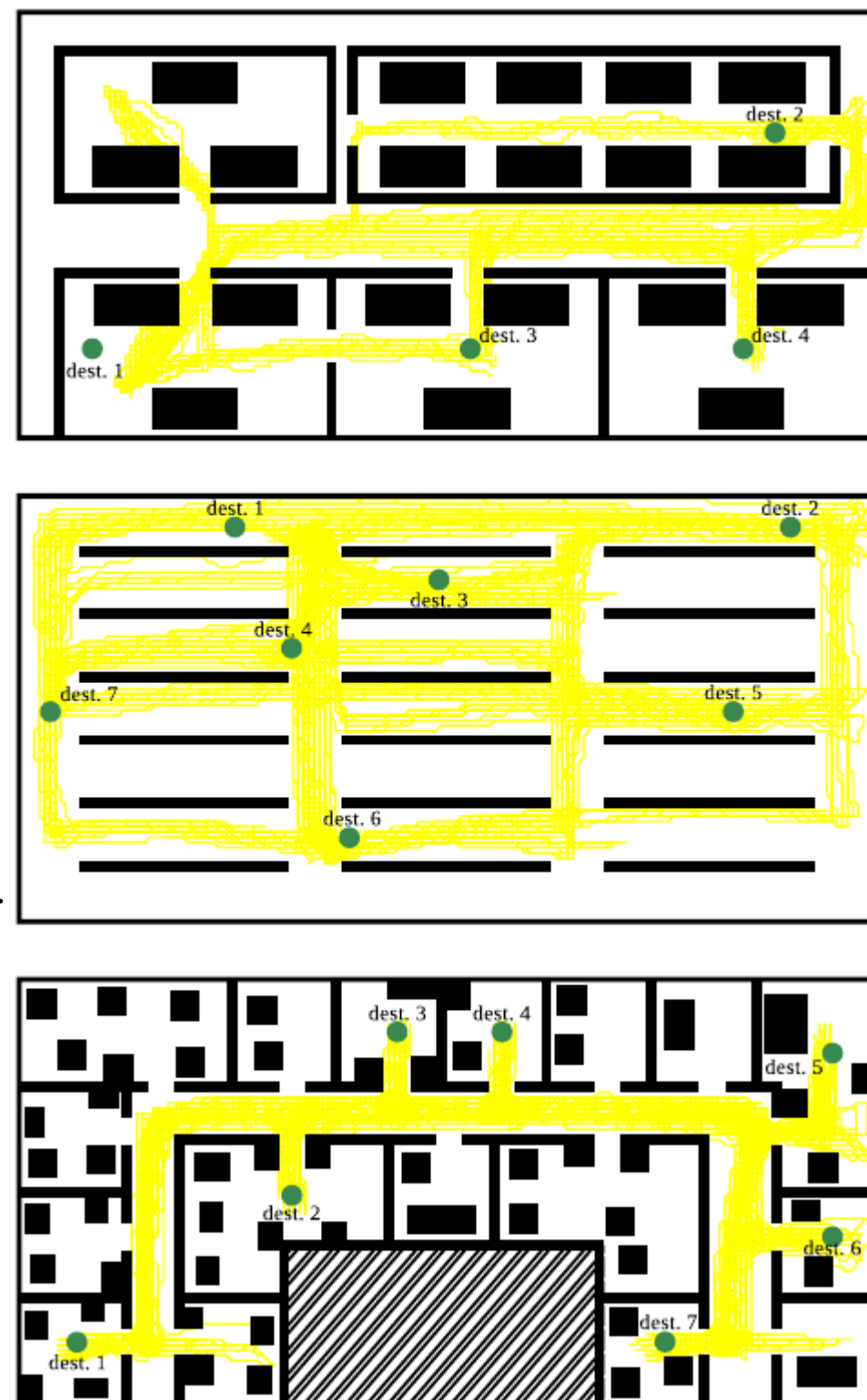
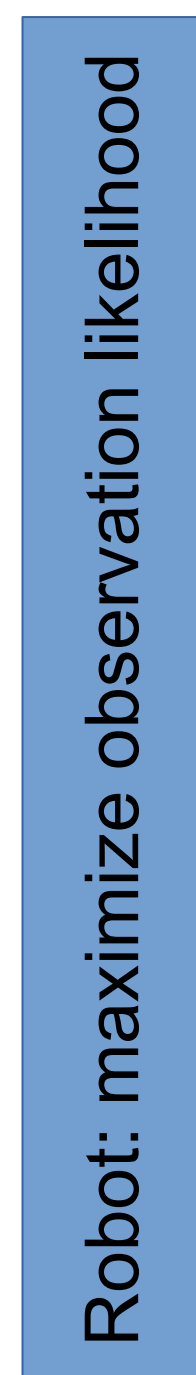
- Kaggle competition – recommendation system for playlist
- Currently, systems are item- or user-based (collaborative filtering)
- Idea: System based on HMM and GMM/HMM
 - Music recommendation → more challenging than movies:
 - Users don't tend to repeat song by song, BUT SEQUENCES!
 - Goal: predict the 30th artist that a given user will play, given the 29 previous artist
- They proposed models where use more 'steps' before to yield a better generalization/score
- Introduce the explore/exploit tradeoff:
 - High-performance model for only a given constrained task, instead the real world needs more robust models
- Ranked 1 on the competition

4.c. Speeding up person finding using hidden Markov models

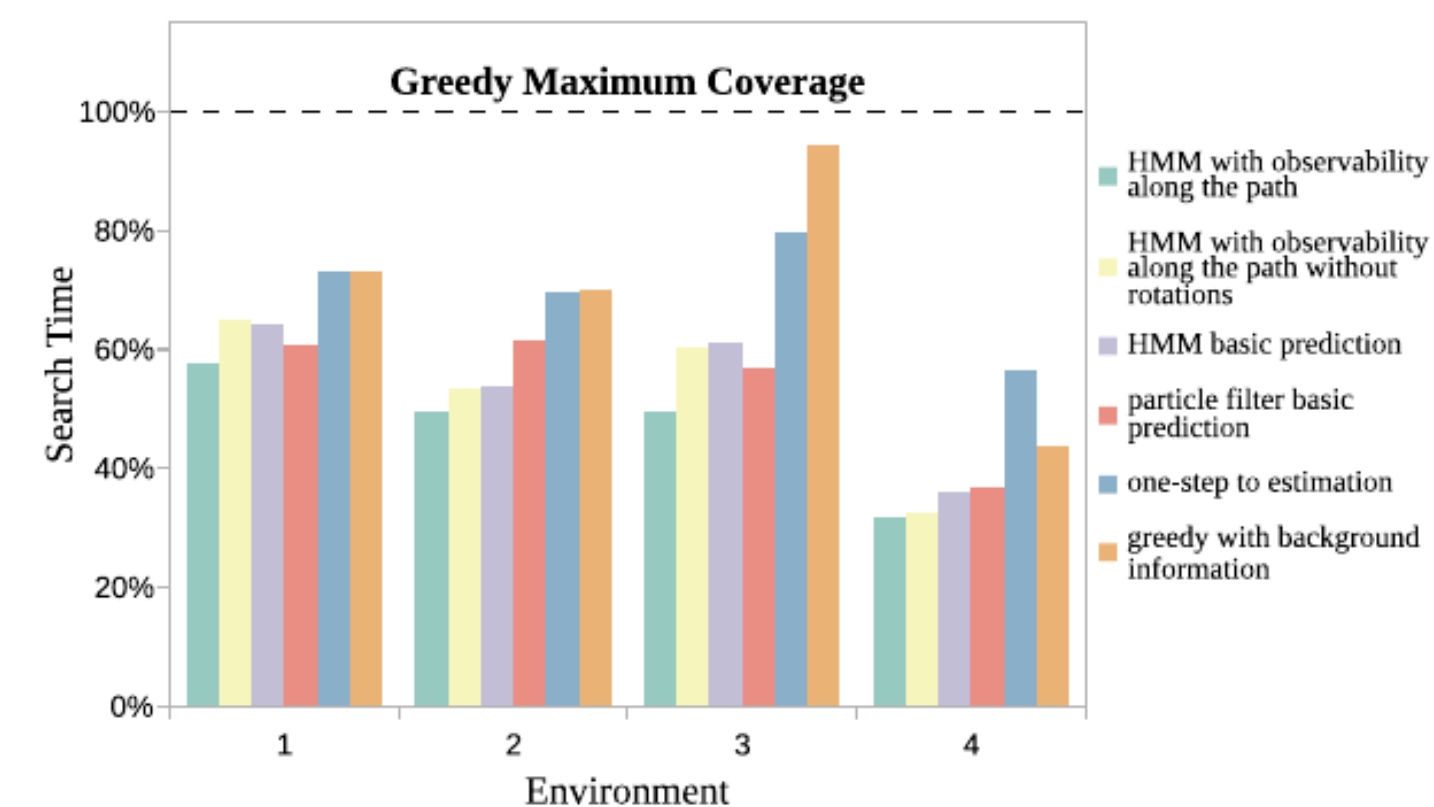
- Ability to efficiently search for a person in a given environment, challenging:
 - Freely moving on the environment
 - Constrained field of view
- Idea: simulates the user's presence locations based on HMMs:
 - Predict user's motion
 - Compute the likelihood at different locations given the predictions
 - Maximize the user's expected location (likelihood of observation)
- The robot will search for the users along the shorter path to a probable location, if not found, set a new probable location and search:
 - After each step, it estimates the best path again, cause it could find the user before reaching the end

4.c. Speeding up person finding using hidden Markov models

- Random generated user's paths on the given environment



- Robots and user's path (updated step by steps)
- Robot drives along the shortest path
- Robot could finish before reaching the end (updating its state)



Path used for HMM training