Neural Networks

Variational Autoencoders

(based on slides from Dan Schwarts and Tom Manzini)

Attendance @1758

Recap

- Neural networks are universal approximators

- They can model
  - Boolean functions
  - Classification functions
  - Regressions

- They can be
  - Feature extractors
  - Classifiers
  - Predictors

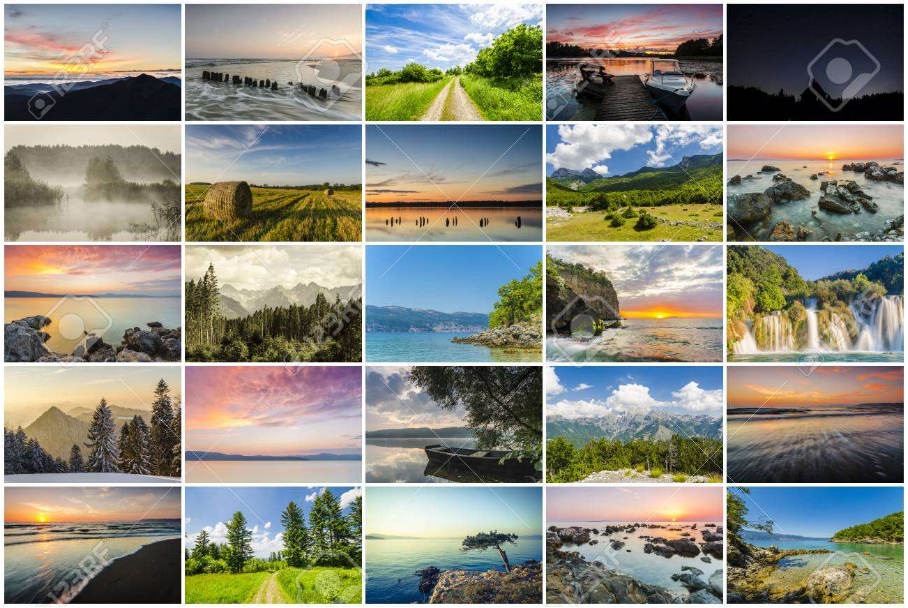
A new problem

- All of the previous cases considered neural networks that are functions
  - They can *operate on*, or *process* a given input data
  - They can learn to perform these tasks from data
- Can networks also ***generate*** data?
  - And learn to do so from examples
  - Topic for next series of lectures

A new problem

- From a large collection of images of faces, can a network learn to *generate* a new portrait
  - Generate samples from the distribution of "face" images
    - How do we even characterize this distribution?

A new problem



- From a large collection of landscapes, can a network learn to *generate* new landscape pictures
  – Generate samples from the distribution of "landscape" images
    - How do we even characterize this distribution?

Neural nets as generative models

- We've seen how neural nets can perform classification or regression
  - MLPs, CNNs, RNNs..

- Next step: NNs as generic generative models
  - Model the distribution of *any* data
  - Such that we can draw samples from it

But first…

The story of generative models

- What are generative models
- How to estimate them
  - *Expectation maximization*

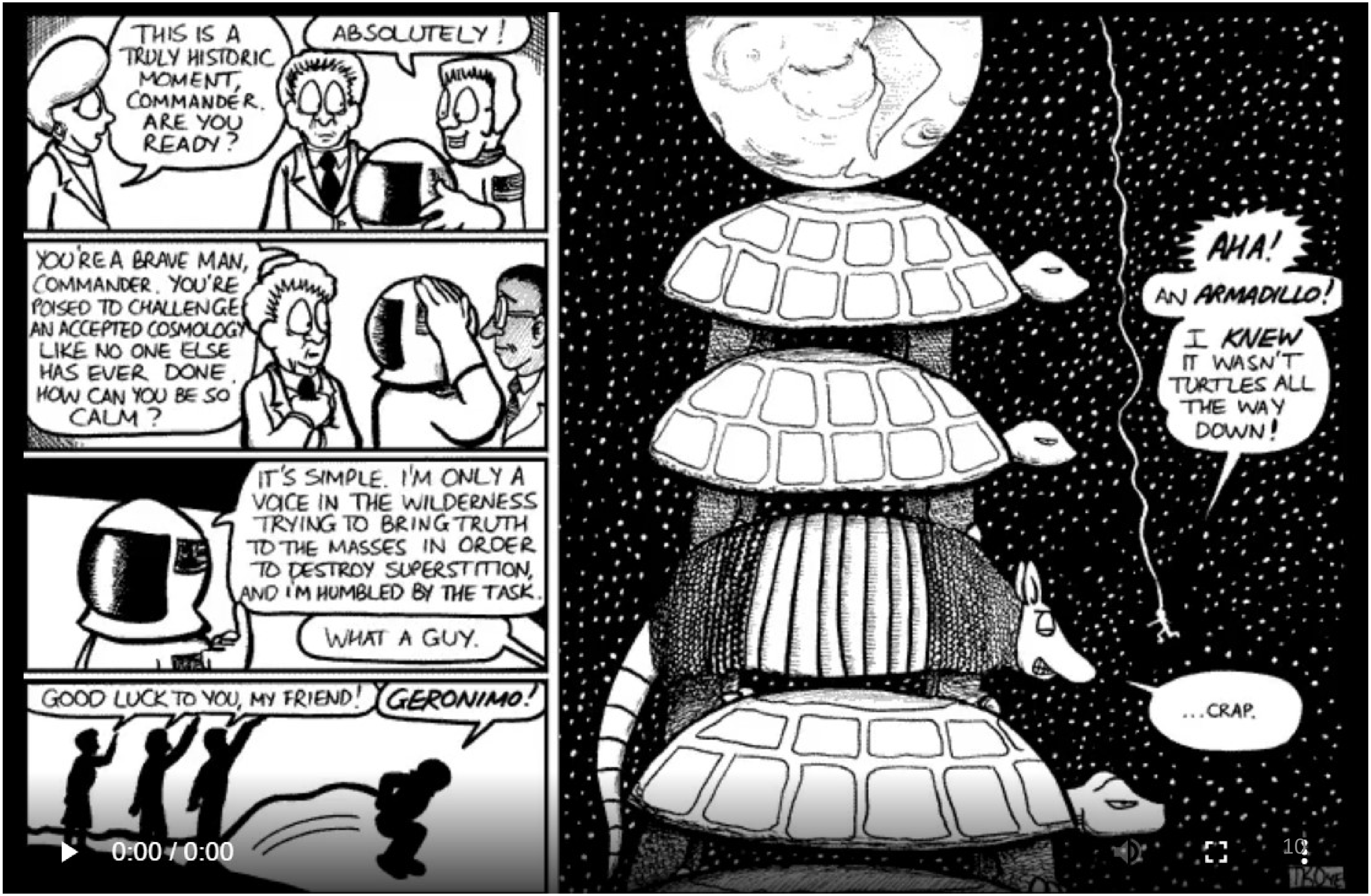
What is a generative model

- A model for the probability distribution of a data x
  - E.g. a multinomial, Gaussian etc.

- Computational equivalent: a model that can be used to “generate” data with a distribution similar to the given data x
  - Typical setting: a box that takes in random seeds and outputs random samples like x

  - Meta question that will matter later: how do we generate the random seeds…

It's turtles all the way down (kinda)…



Some "simple" generative models

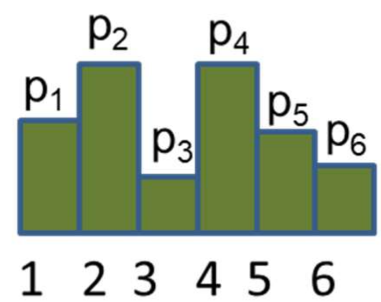
- The category PMF

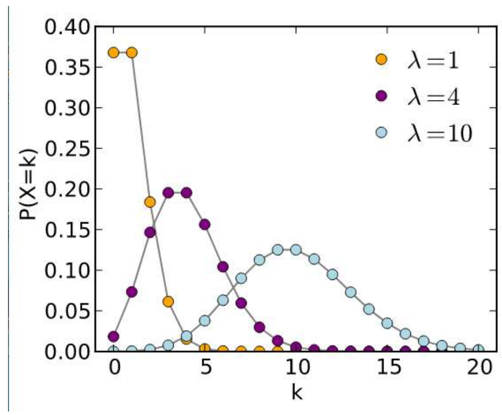
$$P(x = v) \equiv P(v)$$

  - For discrete data
    - v belongs to a discrete set
  - Can be expressed as a table of probabilities if the set of possible vs is finite
  - Else, requires a parametric form, e.g. Poisson

$$P(x = k) = \frac{\lambda^k e^{-\lambda}}{k!} \quad for\ k \geq 0$$

    - λ is the Poisson parameter



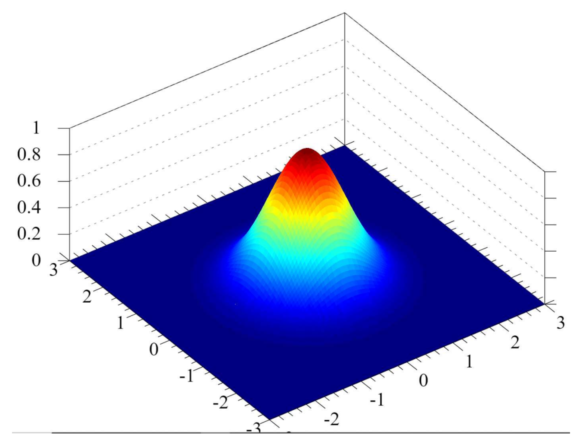


- The Gaussian PDF

$$P(x = v)$$

$$= \frac{1}{\sqrt{2\pi|\Sigma|}^D} \exp(-0.5(x-\mu)^T \Sigma^{-1}(x-\mu))$$

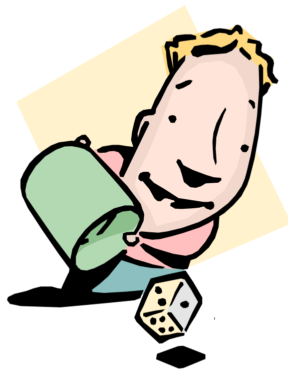
  - For continuous-valued data
  - μ is the mean of the distribution
  - Σ is the Covariance matrix



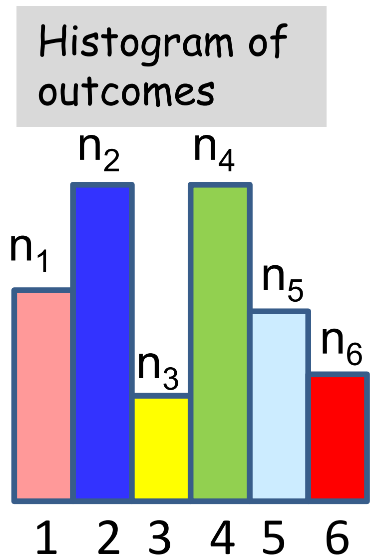
Learning a generative model for data

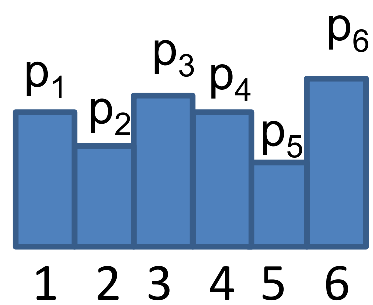
- You are given some set of observed data $X = \{x\}$.
- You choose a model $P(x; \theta)$ for the distribution of x
  - θ are the parameters of the model
- Estimate the θ such that $P(x; \theta)$ best "fits" the observations $X = \{x\}$
  - Hoping it will also represent data outside the training set.

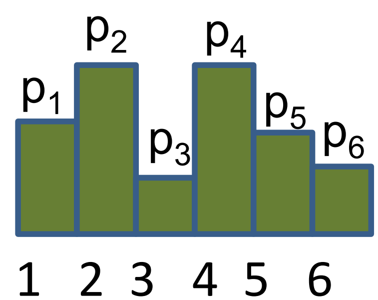
An example: Multinomials



6 3 1 5 4 1 2 4 …

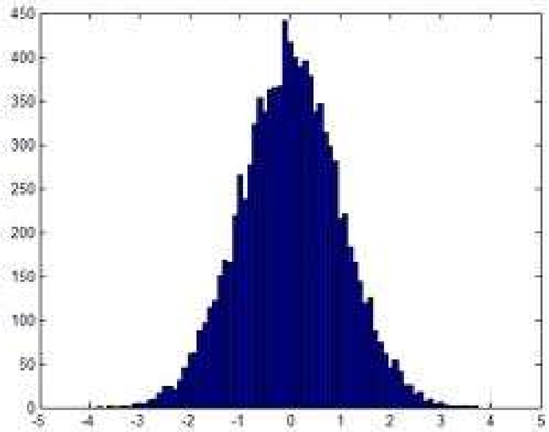


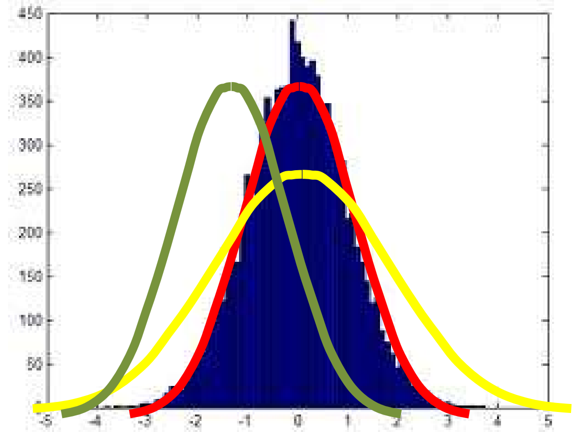




- A dice roller rolls dice and you plot the histogram of outcomes
  - Shown to right
- The distribution is a multinomial
  - Parameters to be learned: $p_1, p_2, p_3, p_4, p_5, p_6$
- Which of the two probability distributions shown to the right is more likely to be the distribution for the dice?
  - Why?

An example





- The left figure shows the histogram of a collection of observations
- We decide to model the distribution as Gaussian
  - Parameters: Mean μ and variance σ^2

- Which of the three Gaussians shown in the right figure is most likely to be the actual PDF of the RV?
  - Why?

Defining "Best Fit": Maximum likelihood

- The data are generated by draws from the distribution
  - I.e. the generating process draws from the distribution

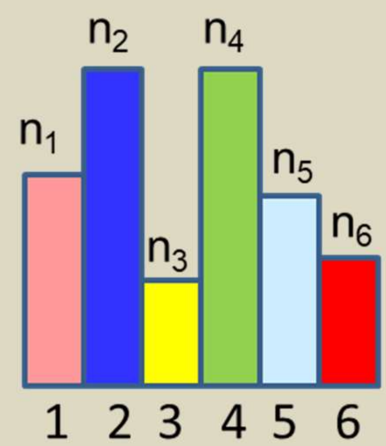
- Assumption: The world is a boring place
  - The data you have observed are very typical of the process

- Consequent assumption: The distribution has a high probability of generating the observed data
  - Not necessarily true

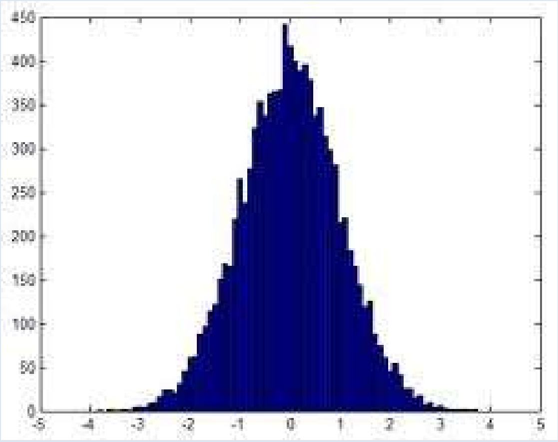
- Select the distribution that has the *highest* probability of generating the data
  - Should assign lower probability to less frequent observations and vice versa

Maximum likelihood

- The maximum likelihood principle:
  - $\underset{\theta}{\operatorname{argmax}} P(X;\theta) = \underset{\theta}{\operatorname{argmax}} \log(P(X;\theta))$

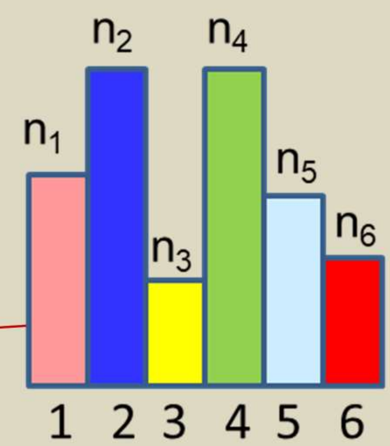


- For the Gaussian
  - $\underset{\theta}{\operatorname{argmax}} \log(\prod_{x \in X} P(x;\theta))$

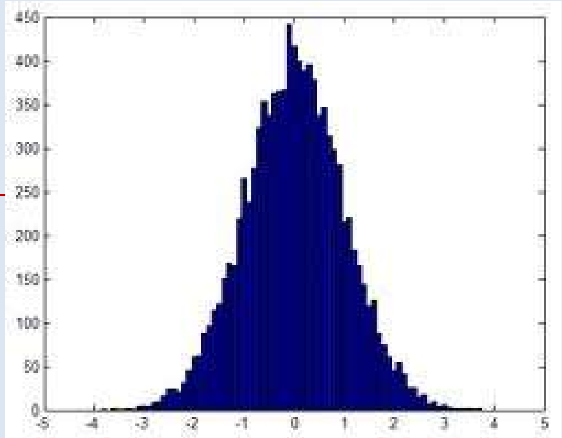


Maximum likelihood

- The maximum likelihood principle:
  - $\underset{\theta}{\operatorname{argmax}} P(X;\theta) = \underset{\theta}{\operatorname{argmax}} \log(P(X;\theta))$
- For the histogram
  - $\underset{\{p_1,p_2,p_3,p_4,p_5,p_6\}}{\operatorname{argmax}} \log(\prod_{x \in X} P(x))$

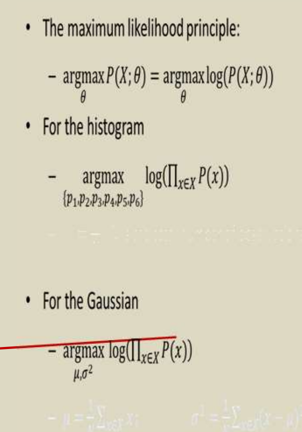


- For the Gaussian
  - $\underset{\mu,\sigma^2}{\operatorname{argmax}} \log(\prod_{x \in X} P(x))$



Maximum likelihood

- The maximum likelihood principle:
  - $\underset{\theta}{\operatorname{argmax}} P(X;\theta) = \underset{\theta}{\operatorname{argmax}} \log(P(X;\theta))$

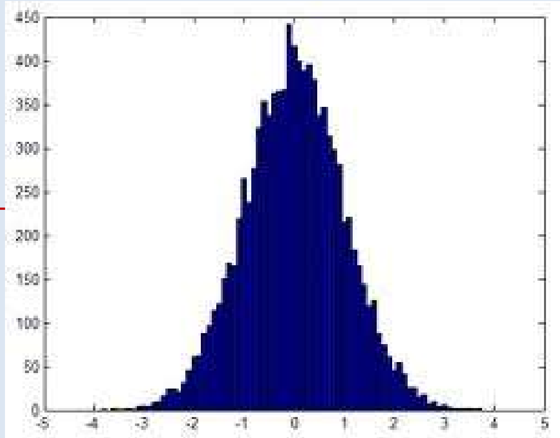


- For the histogram
  - $\underset{\{p_1,p_2,p_3,p_4,p_5,p_6\}}{\operatorname{argmax}} \log(\prod_{x\in X} P(x))$

Can be grouped by value (every instance of i has the same probability)

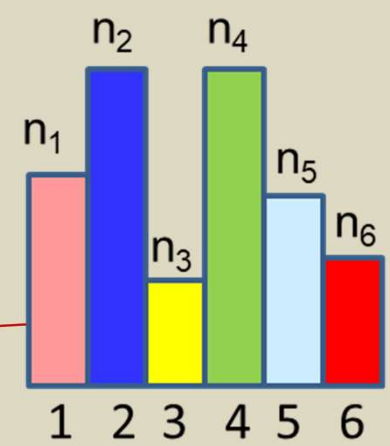
- For the Gaussian
  - $\underset{\mu,\sigma^2}{\operatorname{argmax}} \log(\prod_{x\in X} P(x))$

This probability is a Gaussian

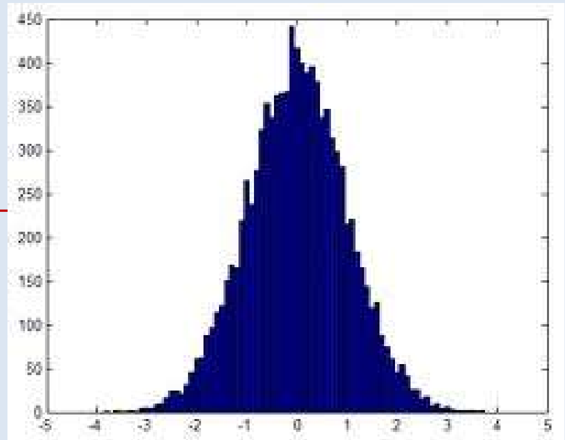


Maximum likelihood

- The maximum likelihood principle:
  - $\underset{\theta}{\operatorname{argmax}} P(X;\theta) = \underset{\theta}{\operatorname{argmax}} \log(P(X;\theta))$
- For the histogram
  - $\underset{\{p_1,p_2,p_3,p_4,p_5,p_6\}}{\operatorname{argmax}} \log\left(\prod_i p_i^{n_i}\right)$

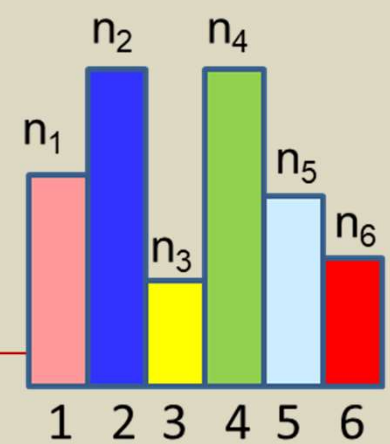


- For the Gaussian
  - $\underset{\mu,\sigma^2}{\operatorname{argmax}} \log(\prod_{x \in X} Gaussian(x;\mu,\sigma^2))$



Maximum likelihood

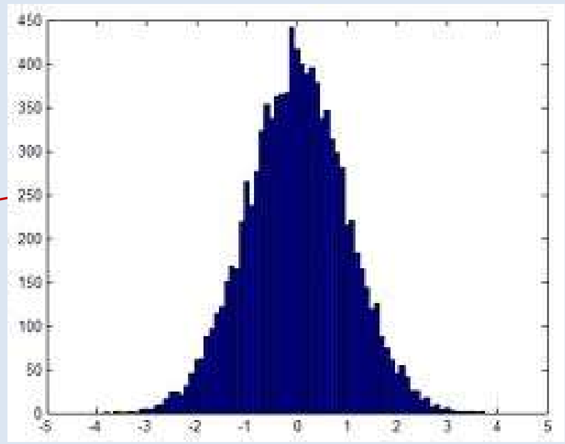
- The maximum likelihood principle:
  - $\underset{\theta}{\operatorname{argmax}} P(X;\theta) = \underset{\theta}{\operatorname{argmax}} \log(P(X;\theta))$
- For the histogram



  - $\underset{\{p_1,p_2,p_3,p_4,p_5,p_6\}}{\operatorname{argmax}} \sum_i n_i \log(p_i)$

  $\Rightarrow p_i = \frac{n_i}{N}$ (N is the total number of observations)

- For the Gaussian



  - $\underset{\mu,\sigma^2}{\operatorname{argmax}} \sum_{x\in X}\left(-0.5\log(2\pi\sigma^2) - \frac{(x-\mu)^2}{2\sigma^2}\right)$

  $\Rightarrow \mu = \frac{1}{N}\sum_{x\in X} x\,;$ $\quad\quad \sigma^2 = \frac{1}{N}\sum_{x\in X}(x-\mu)^2$

Poll 1 (@1759, @1760)

Maximum-likelihood estimation of probability distributions is based on the theory that the world is a terribly boring place

- True
- False

Maximum-likelihood estimation estimates the values of the parameters of a probability distribution such that they maximize the probability of the training data

- True
- False

Poll 1

Maximum-likelihood estimation of probability distributions is based on the theory that the world is a terribly boring place

- **True**
- False

Maximum-likelihood estimation estimates the values of the parameters of a probability distribution such that they maximize the probability of the training data

- **True**
- False

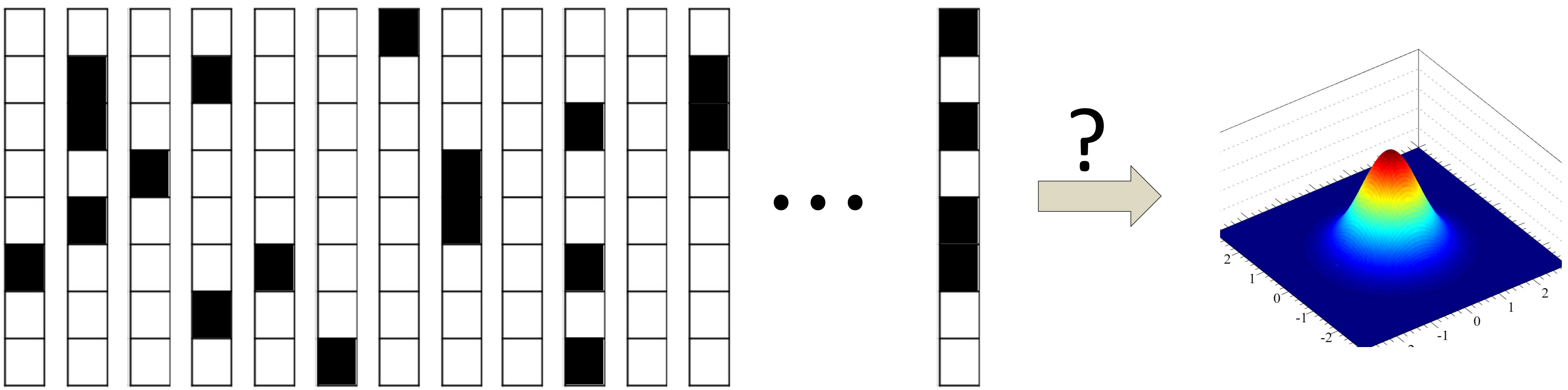
Maximum Likelihood Estimation

- Sometimes the data provided may be incomplete
  - May be insufficient to write out the complete log probability
  - Insufficient to estimate your model parameters directly
- This could be because the data themselves have missing components
  - E.g. Data vectors have some missing components
- Or because of the structure of the model
  - Mixture models, multi-stage Generative models

Maximum Likelihood Estimation

- Sometimes the data provided may be incomplete
  - May be insufficient to write out the complete log probability
  - Insufficient to estimate your model parameters directly
- This could be because the data themselves have missing components
  - E.g. Data vectors have some missing components
- Or because of the structure of the model
  - Mixture models, multi-stage Generative models

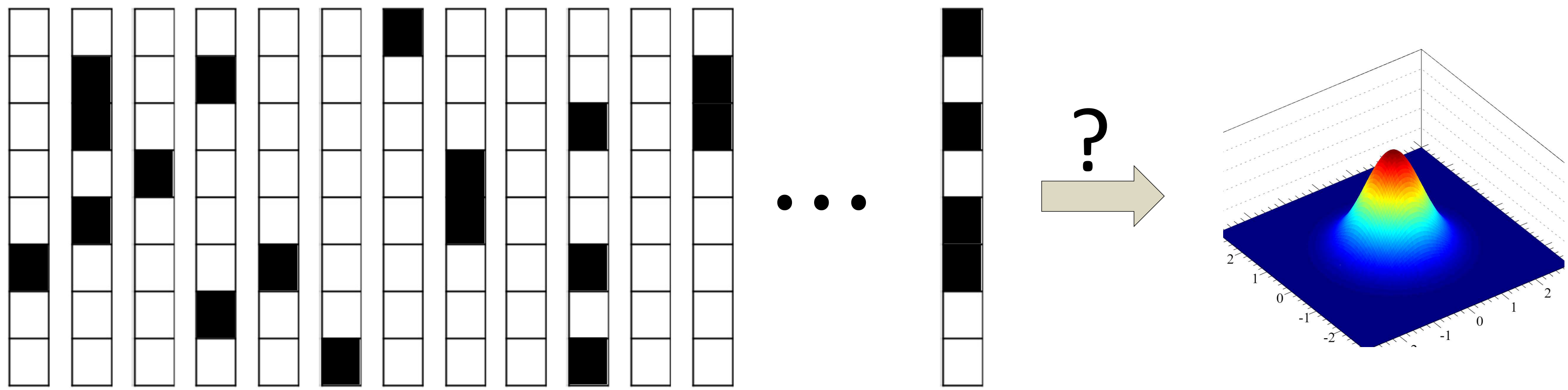
Examples of incomplete data: missing data



Blacked-out components are missing from data

- Objective: Estimate a Gaussian distribution from a collection of vectors
- Problem: Several of the vector components are missing
- Must estimate the mean and covariance of the Gaussian with these incomplete data
  - What would be a good way of doing this?

Maximum likelihood estimation with incomplete data



Blacked-out components are missing from data

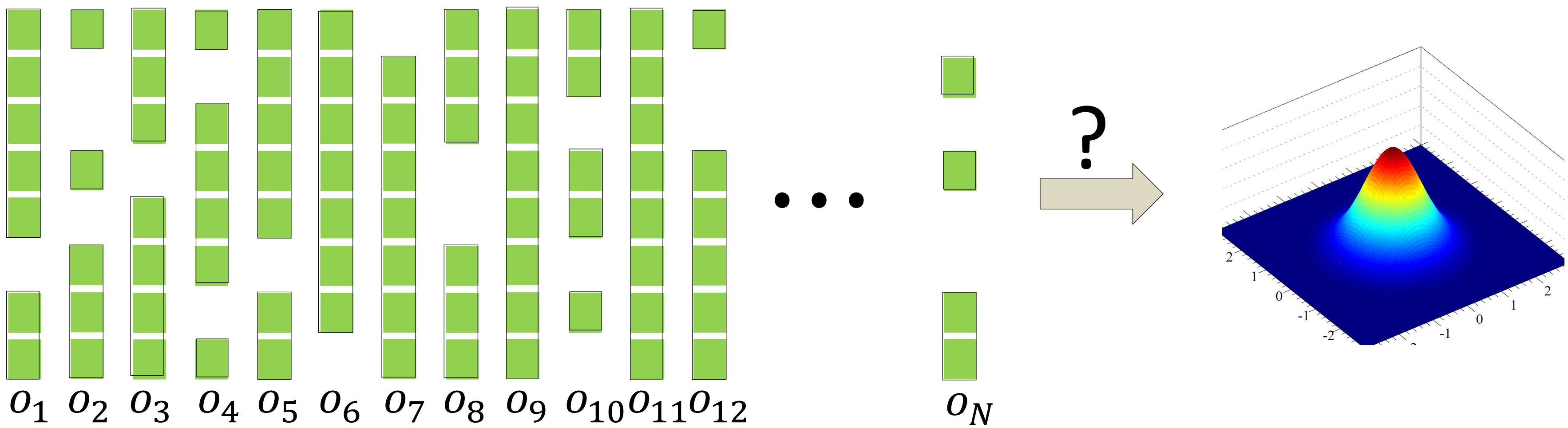
- Original problem: Estimate the Gaussian given a collection $X = \{x\}$ of *complete* vectors

$$\underset{\mu,\sigma^2}{\operatorname{argmax}} \log(P(X)) \quad \text{where X is the entire data}$$

$$= \underset{\mu,\sigma^2}{\operatorname{argmax}} \sum_{x \in X} \log P(x) \quad \text{where P() is a Gaussian}$$

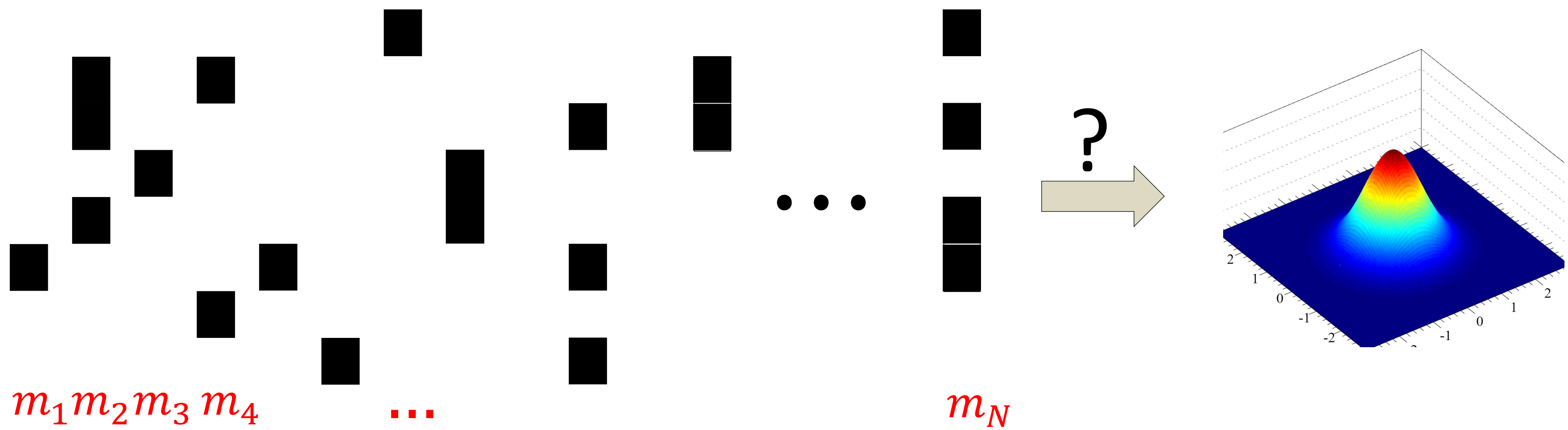
- Unfortunately, many components of each vector are missing in our data

Maximum likelihood estimation with incomplete data



- These are the actual data we have: A set $O = \{o_1, \dots, o_N\}$ of *incomplete* vectors
  - Comprising only the *observed* components of the data

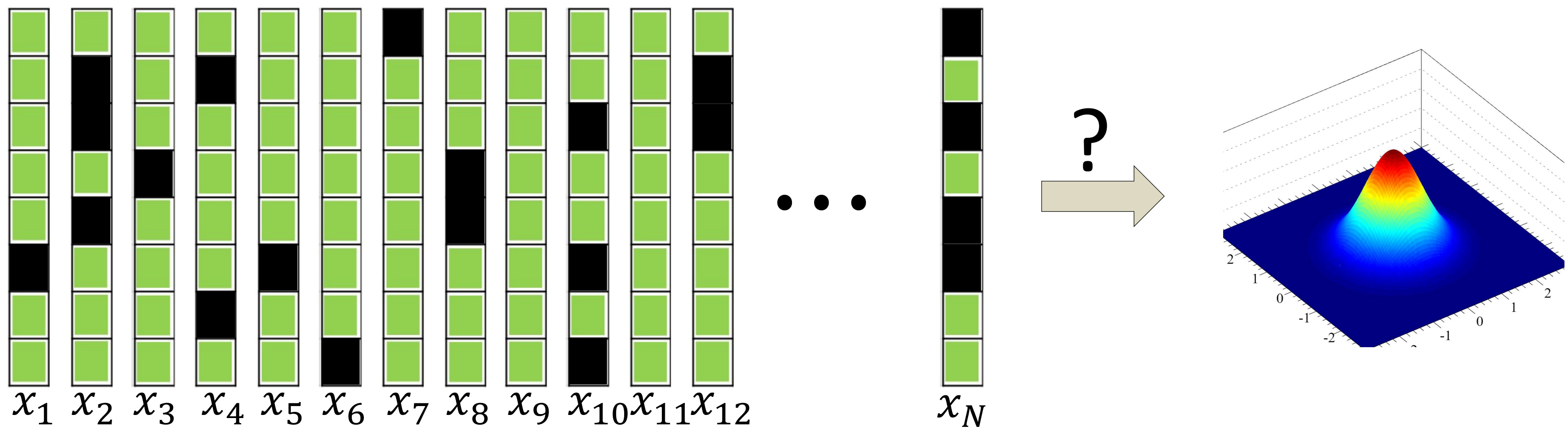
Maximum likelihood estimation with incomplete data



- These are the actual data we have: A set $O = \{o_1, \ldots, o_N\}$ of *incomplete* vectors
  - Comprising only the *observed* components of the data

- We are *missing* the data $M = \{m_1, \ldots, m_N\}$
  - Comprising the *missing* components of the data

Maximum likelihood estimation with incomplete data

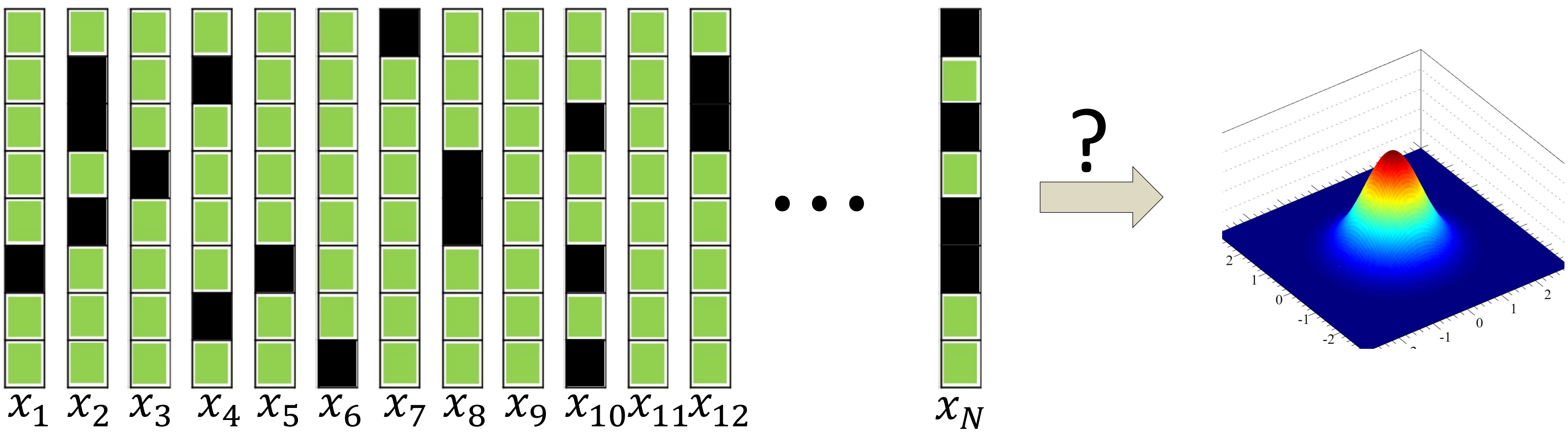


- These are the actual data we have: A set $O = \{o_1, \ldots, o_N\}$ of *incomplete* vectors
  - Comprising only the *observed* components of the data
- We are *missing* the data $M = \{m_1, \ldots, m_N\}$
  - Comprising the *missing* components of the data
- The *complete* data includes both the observed and missing components

$$X = \{x_1, \ldots, x_N\}, \qquad x_i = (o_i, m_i)$$

  - Keep in mind that at the complete data are *not* available (the missing components are missing)

Maximum likelihood estimation with incomplete data



- Maximum likelihood estimation: Maximize the likelihood of the *observed* data
  - That is all we really have

$$\underset{\mu,\sigma^2}{\operatorname{argmax}} \log(P(O)) = \underset{\mu,\sigma^2}{\operatorname{argmax}} \sum_{o \in O} \log P(o)$$

- Unfortunately, the Gaussian is defined on the *complete* vector :
  - $P(x) = Gaussian(x; \mu, \sigma^2)$
  - In order to compute $P(o)$ we must *derive* it from $P(x)$

The log likelihood of incomplete data

- The probability of any vector x with observed and missing parts o and m

$$P(x) = P(o, m)$$

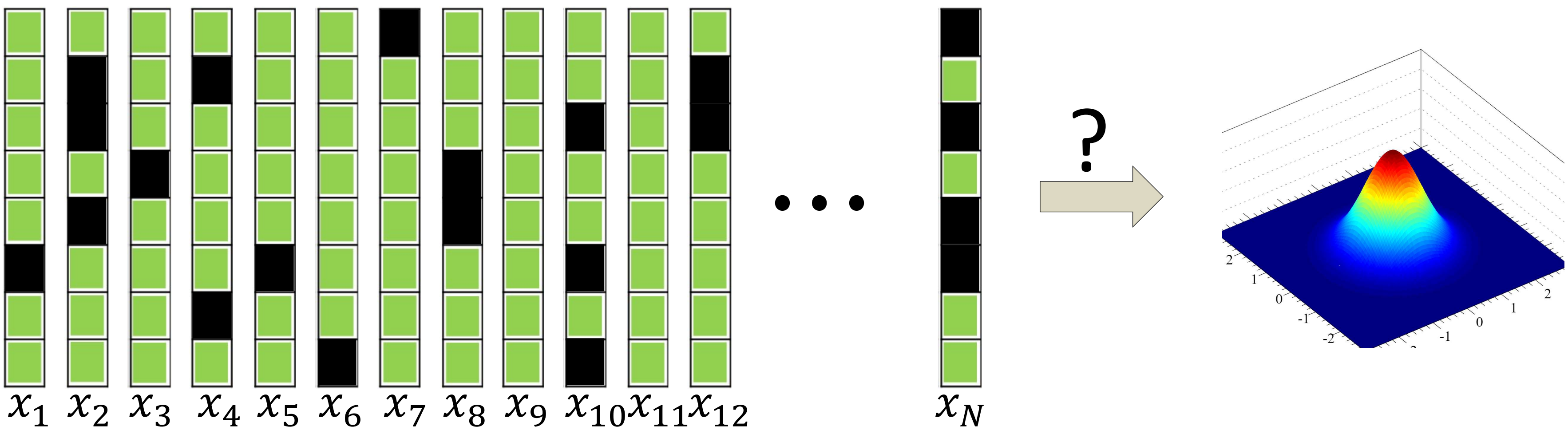
- Compute the probability of the observed components by marginalizing out the missing components

$$P(o) = \int_{-\infty}^{\infty} P(x)dm = \int_{-\infty}^{\infty} P(o, m)dm$$

- The log probability of the *entire observed training data:*

$$\sum_{o \in O} \log \int_{-\infty}^{\infty} P(o, m)dm$$

Maximum likelihood estimation with incomplete data



- Maximum likelihood estimation: Maximize the likelihood of the *observed* data

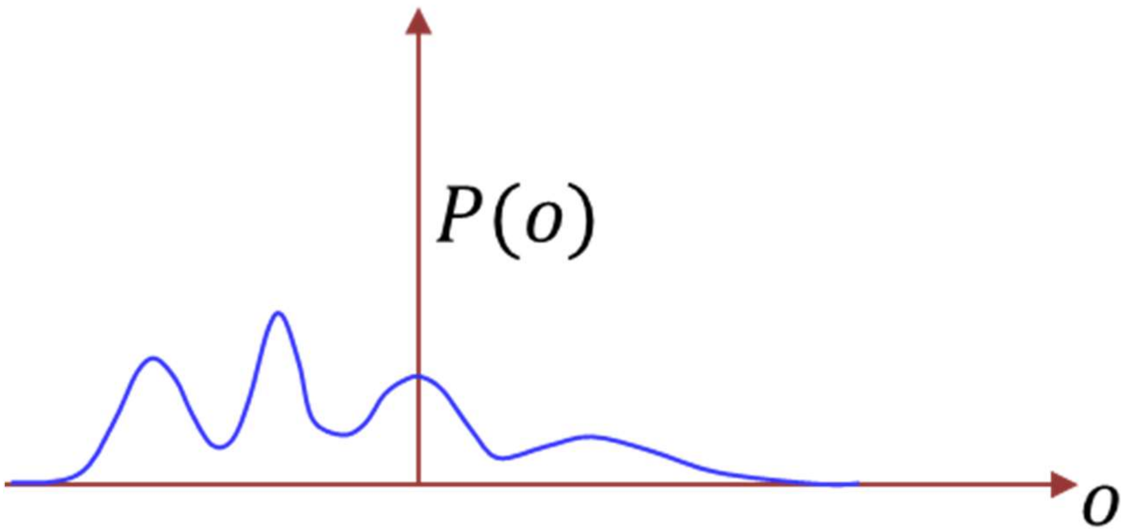
$$\underset{\mu,\sigma^2}{\operatorname{argmax}} \log(P(O)) = \underset{\mu,\sigma^2}{\operatorname{argmax}} \sum_{o \in O} \log \int_{-\infty}^{\infty} P(o,m)\,dm$$

- This requires the maximization of the log of an integral!
  - No closed form
  - Challenging on a good day, impossible on a bad one

Maximum Likelihood Estimation

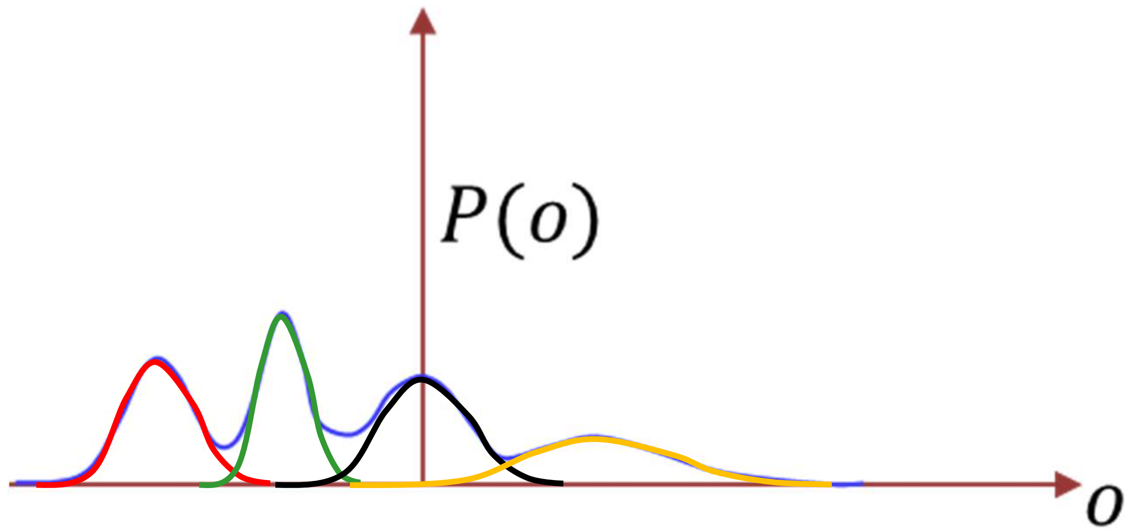
- Sometimes the data provided may be incomplete
  - Insufficient to estimate your model parameters directly
- This could be because the data themselves have missing components
  - E.g. Data vectors have some missing components
- Or because of the structure of the model
  - Mixture models, multi-stage Generative models

The Gaussian Mixture



- Often, when trying to model a complicated distribution

The Gaussian Mixture

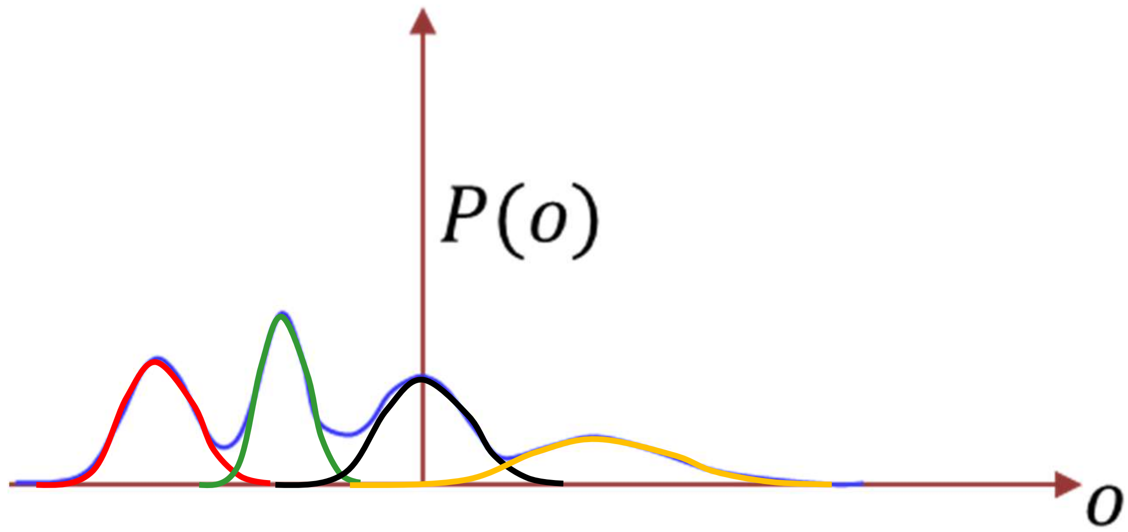


- Often, when trying to model a complicated distribution, we model it as a *mixture of Gaussians* (GMM)
  - A weighted sum of Gaussians

$$P(o) = \sum_k P(k) N(o; \mu_k, \sigma_k^2)$$

  - The weights sum to 1.0

The Gaussian Mixture

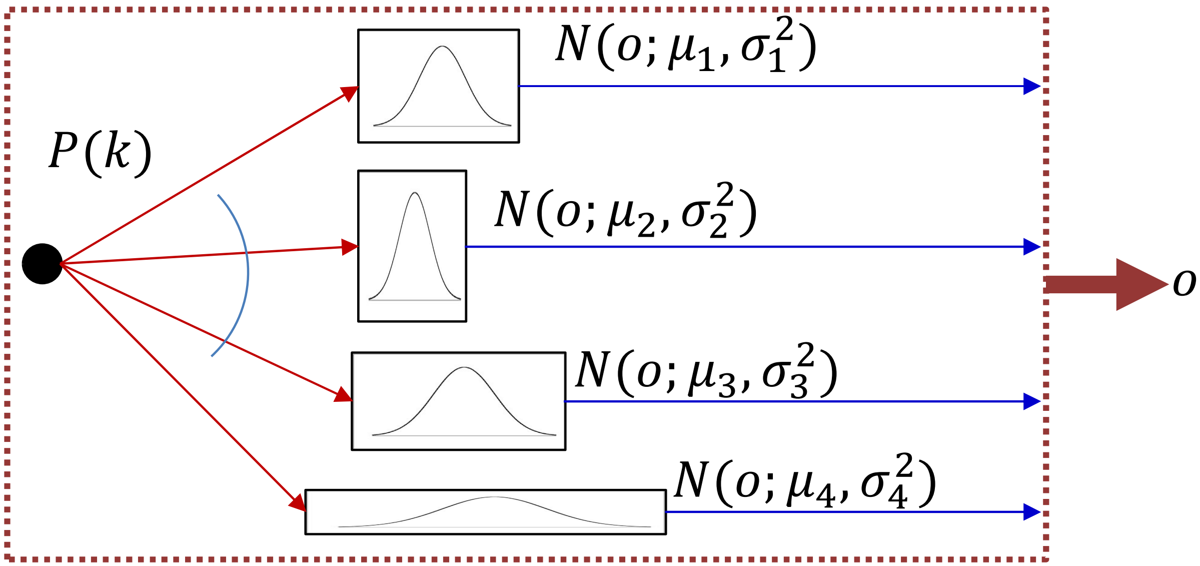


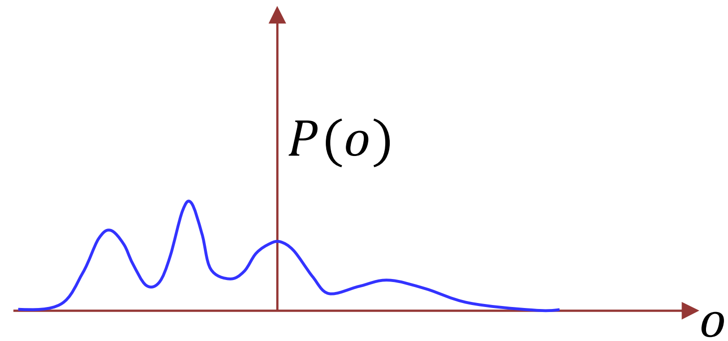
- Often, when trying to model a complicated distribution, we model it as a *mixture of Gaussians* (GMM)
  - A weighted sum of Gaussians

$$P(o) = \sum_k P(k) N(o; \mu_k, \sigma_k^2)$$

  - The weights sum to 1.0
- Problem: Given a number of samples from the original (complicated) distribution, how to determine the parameters of the parameters of the GMM to best fit them

Examples of incomplete data: missing information in Gaussian mixtures





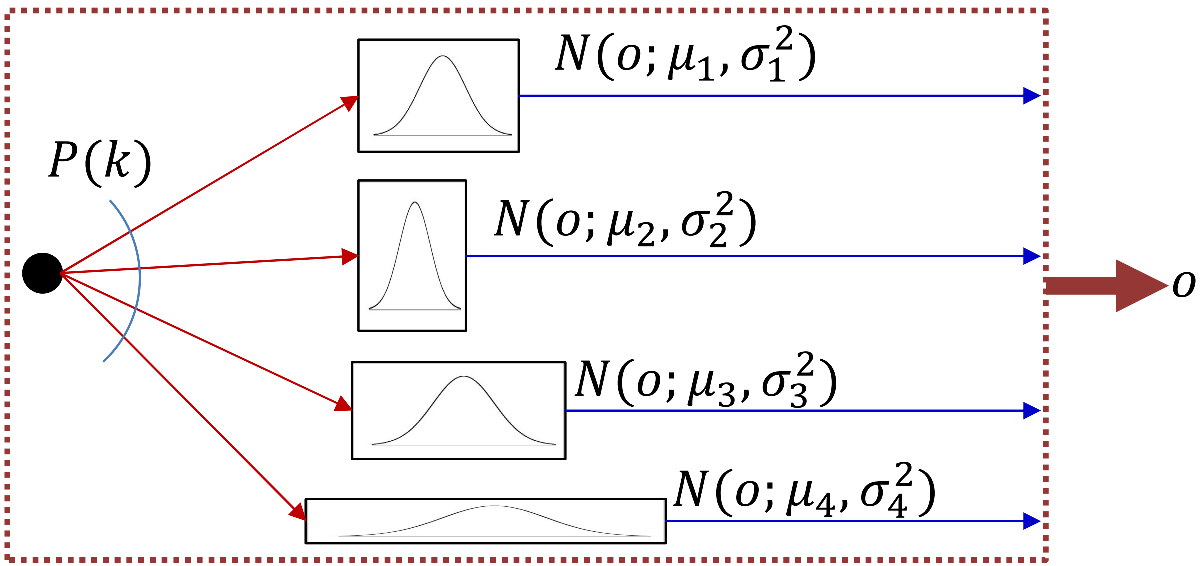
$$P(o) = \sum_k P(k)N(o;\mu_k,\sigma_k^2)$$

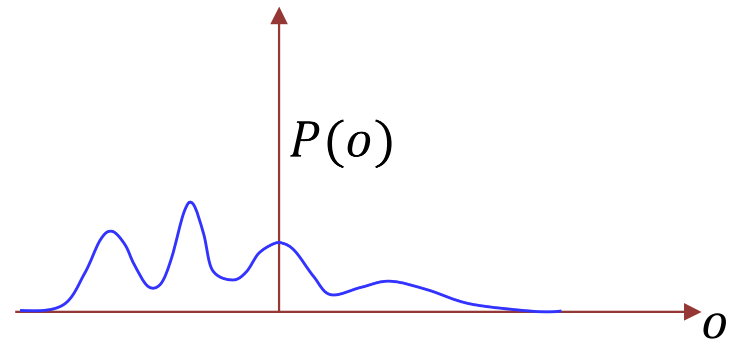
Mixture Gaussian

- The generative model characterizes the data as the outcome of a two-level process
  - In the first step the process chooses a Gaussian from a collection
  - In the second, it draws the vector o from the chosen Gaussian
  - The overall model is a *mixture Gaussian*

- Objective: Learn the parameters of all the Gaussians from training data
  - Learn the means and variances of the individual Gaussians
    - And also the probability with which each Gaussian is selected for the draw

The Gaussian Mixture generative model





$$P(o) = \sum_k P(k)N(o;\mu_k,\sigma_k^2)$$

Mixture Gaussian

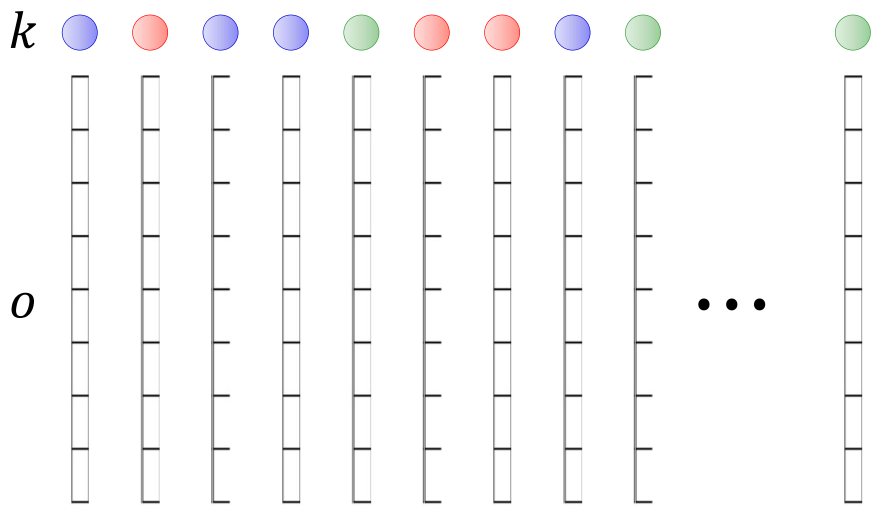
- Note, the process actually draws *two* variables for each observation, k and o.
- The probability of a particular draw is actually the joint probability of both variables

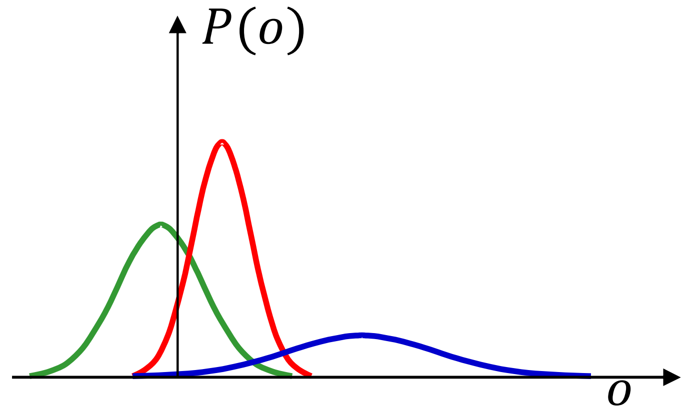
$$P(k,o) = P(k)P(o|k) = P(k)N(o;\mu_k,\sigma_k^2)$$

- To compute the probability of obtaining any observation o, we are *marginalizing out* the Gaussian index variable

$$P(o) = \sum_k P(k,o) = \sum_k P(k)N(o;\mu_k,\sigma_k^2)$$

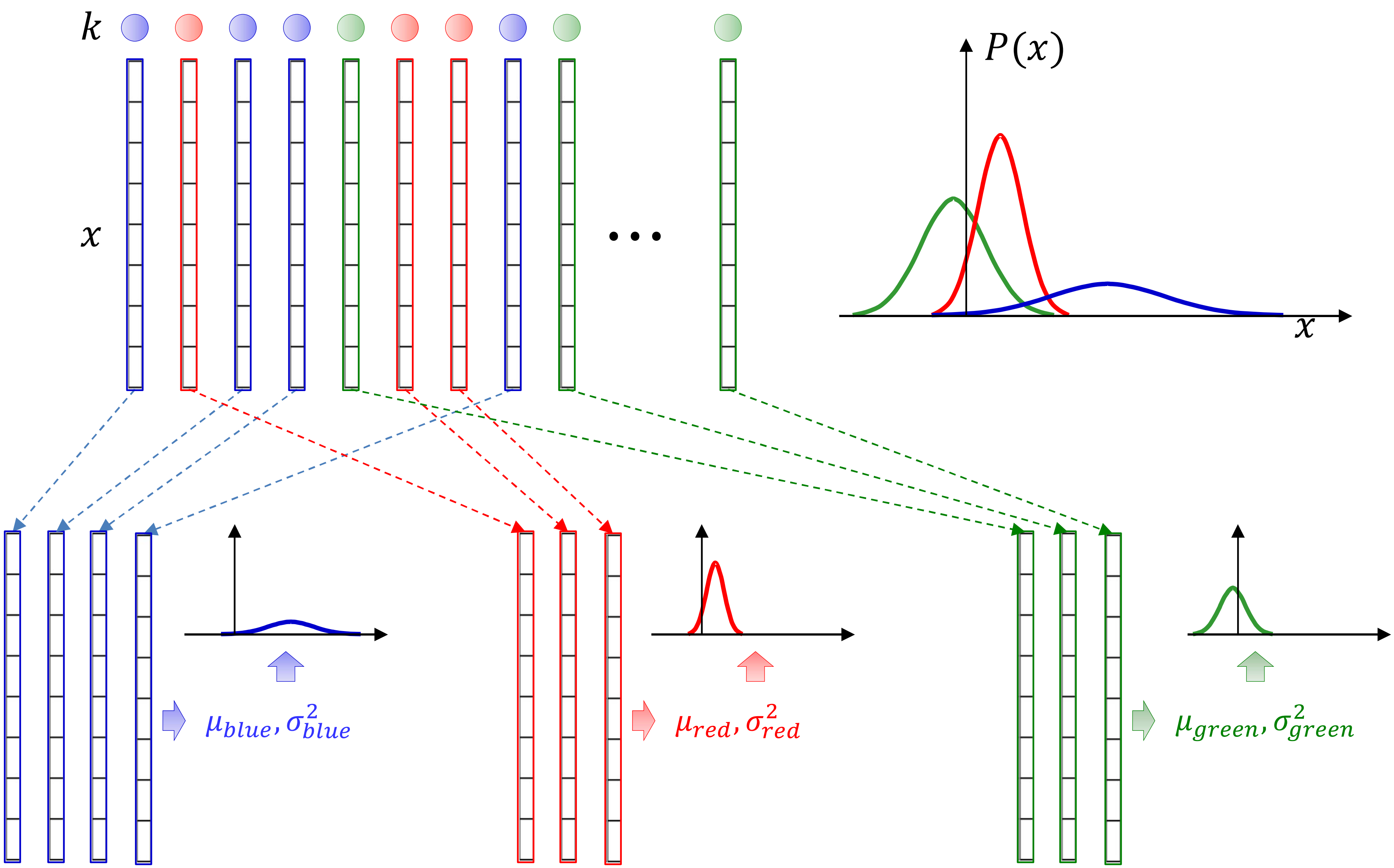
The *complete* data needed to precisely learn the model



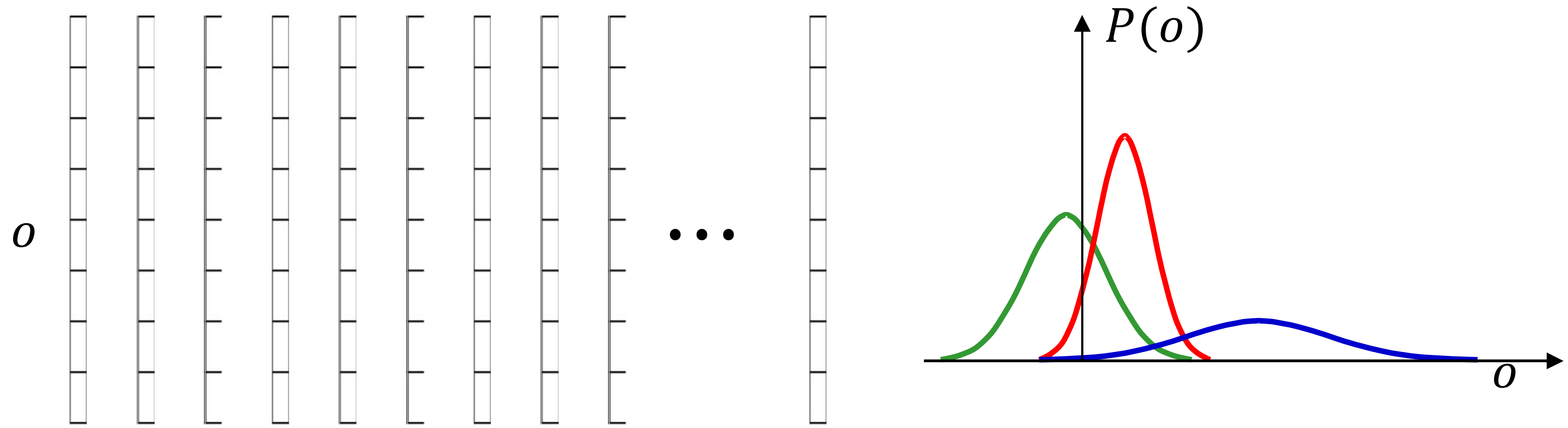


- Ideal data: Each training instance includes both the data vector o and the Gaussian k it was drawn from
  - In order to estimate the parameters of any Gaussian, you only need to segregate the training instances from that Gaussian, and compute the mean and variance from them

Learning a GMM with "complete" data



The GMM problem of incomplete data: missing information



- Problem : We are not given the actual Gaussian for each observation
  - Our data are incomplete

- What we want : $(o_1, k_1), (o_2, k_2), (o_3, k_3)$ ...
- What we have: o_1, o_2, o_3 ...

ML estimation with only *observed data*

- The maximum likelihood estimation problem:
  - Given *observed data* $\mathrm{O} = \{o_1, o_2, o_3 \ldots\}$,
  - estimate $\{(\mu_k, \sigma_k^2), \forall k\}$ – the parameters of all the Gaussians

$$\underset{\{(\mu_k,\sigma_k^2),\forall k\}}{\operatorname{argmax}} \log(P(O)) = \underset{\{(\mu_k,\sigma_k^2),\forall k\}}{\operatorname{argmax}} \sum_{o \in O} \log P(o)$$

ML estimation with only *observed data*

- The maximum likelihood estimation problem:
  - Given *observed data* $\mathrm{O} = \{o_1, o_2, o_3 \dots\}$,
  - estimate $\{(\mu_k, \sigma_k^2), \forall k\}$ – the parameters of all the Gaussians

$$\underset{\{(\mu_k,\sigma_k^2),\forall k\}}{\operatorname{argmax}} \log(P(O)) = \underset{\{(\mu_k,\sigma_k^2),\forall k\}}{\operatorname{argmax}} \sum_{o \in O} \log P(o)$$

- The probability of an individual vector:

$$P(o) = \sum_{k} P(k, o)$$

ML estimation with only *observed data*

- The maximum likelihood estimation problem:
  - Given *observed data* $O = \{o_1, o_2, o_3 \ldots\}$,
  - estimate $\{(\mu_k, \sigma_k^2), \forall k\}$ – the parameters of all the Gaussians

$$\underset{\{(\mu_k,\sigma_k^2),\forall k\}}{\operatorname{argmax}} \log(P(O)) = \underset{\{(\mu_k,\sigma_k^2),\forall k\}}{\operatorname{argmax}} \sum_{o \in O} \log P(o)$$

- The probability of an individual vector:

$$P(o) = \sum_k P(k, o) = \sum_k P(k) N(o; \mu_k, \sigma_k^2)$$

ML estimation with only *observed data*

- The maximum likelihood estimation problem:
  - Given *observed data* $O = \{o_1, o_2, o_3 \ldots\}$,
  - estimate $\left\{\left(\mu_k, \sigma_k^2\right), \forall k\right\}$ – the parameters of all the Gaussians

$$\underset{\{(\mu_k,\sigma_k^2),\forall k\}}{\operatorname{argmax}} \log(P(O)) = \underset{\{(\mu_k,\sigma_k^2),\forall k\}}{\operatorname{argmax}} \sum_{o \in O} \log P(o)$$

- The probability of an individual vector:

$$P(o) = \sum_k P(k) N(o; \mu_k, \sigma_k^2)$$

- The maximum likelihood estimation again

$$\underset{\{(\mu_k,\sigma_k^2),\forall k\}}{\operatorname{argmax}} \sum_{o \in O} \log \sum_k P(k) N(o; \mu_k, \sigma_k^2)$$

ML estimation with only *observed data*

- The maximum likelihood estimation problem:
  - Given *observed data* $O = \{o_1, o_2, o_3 \ldots\}$,
  - estimate $\{(\mu_k, \sigma_k^2), \forall k\}$ – the parameters of all the Gaussians

$$\underset{\{(\mu_k, \sigma_k^2), \forall k\}}{\operatorname{argmax}} \log(P(O)) = \underset{\{(\mu_k, \sigma_k^2), \forall k\}}{\operatorname{argmax}} \sum_{o \in O} \log P(o)$$

- The probability of an individual vector:

$$P(o) = \sum_k P(k) N(o; \mu_k, \sigma_k^2)$$

- The maximum likelihood estimation again

$$\underset{\{(\mu_k, \sigma_k^2), \forall k\}}{\operatorname{argmax}} \sum_{o \in O} \log \sum_k P(k) N(o; \mu_k, \sigma_k^2)$$

- This includes the log of a sum, which defies direct optimization

The general form of the problem

- The "presence" of missing data or variables requires them to be marginalized out of your probability
  - By summation or integration

- This results in a maximum likelihood estimate of the form

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \sum_{o} \log \sum_{h} P\left(h, o; \theta\right)$$

  - The inner summation may also be an integral in some problems
  - Explicitly introducing θ in the RHS to show that the probability is computed by a model with parameter θ which must be estimated

- The log of a sum (or integral) makes estimation challenging
  - No closed form solution
  - Need efficient iterative algorithms

The general form of the problem

- The "presence" of missing data or variables requires them to be marginalized out of your probability
  - By summation or integration

Can we get an approximation to this that is more tractable?
(i.e without a summation or integral within the log)

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \sum_{o} \log \sum_{h} P(h, o)$$

  - The inner summation may also be an integral in some problems

- The log of a sum (or integral) makes estimation challenging
  - No closed form solution
  - Need efficient iterative algorithms

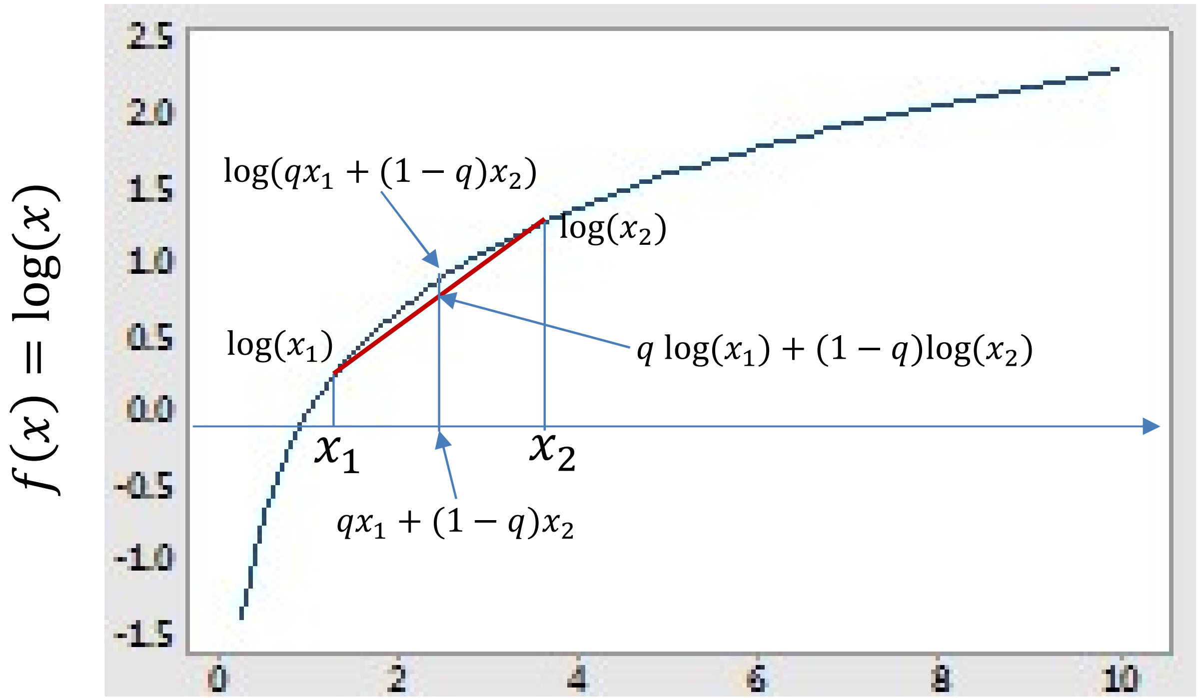
The variational lower bound

- We can rewrite

$$\log P(o) = \log \sum_h P(h, o) = \log \sum_h Q(h) \frac{P(h, o)}{Q(h)}$$

  - Where $Q(h)$ is some function such that $Q(h) \geq 0$ and $\sum_h Q(h) = 1$
    - I.e. a probability distribution

The logarithm is a concave function



- For any x_1 and x_2, for any $0 \leq q \leq 1$,
$$\log(qx_1 + (1-q)x_2) \geq q\log(x_1) + (1-q)\log(x_2)$$

- More generally for any set of $\{x_i\}$, and any weights $\{q_i\}$ s.t. $q_i \geq 0$ and $\sum_i q_i = 1$

$$\log\left(\sum_i q_i x_i\right) \geq \sum_i q_i \log(x_i)$$

The variational lower bound

- By the concavity of the log function

$$\log \sum_h Q(h) \, \frac{P(h,o)}{Q(h)} \geq \sum_h Q(h) \log \frac{P(h,o)}{Q(h)}$$

  – For any $Q(h) \geq 0$ and $\sum_h Q(h) = 1$
  – Note, the LHS is exactly equal to $\log P(o)$

- This is the *variational lower bound* on $\log P(o)$
  – Also called the Evidence Lower BOund, or ELBO

Or more explicitly

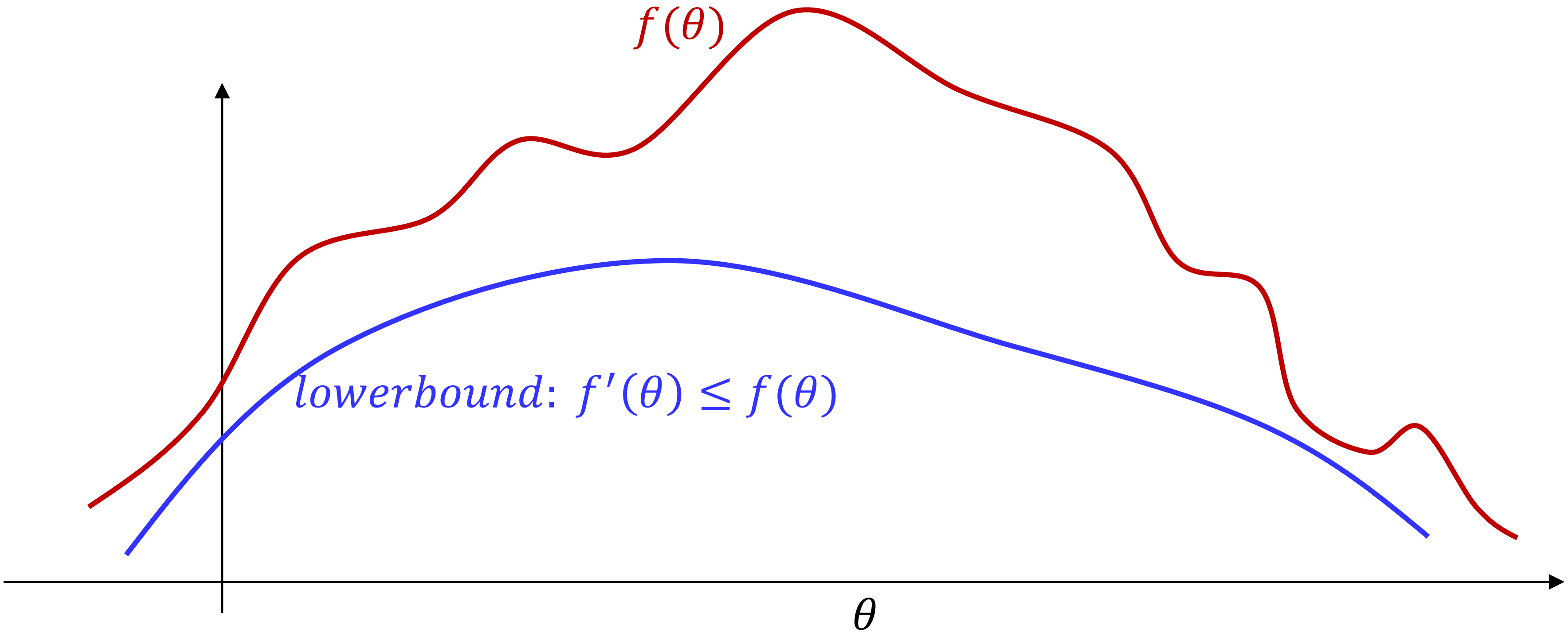
- By the concavity of the log function

$$\log P(o;\theta) \geq \sum_h Q(h) \log \frac{P(h,o;\theta)}{Q(h)}$$

  – Explicitly showing that the probability is computed by a model with parameter θ
    - We must maximize $P(o;\theta)$ w.r.t θ

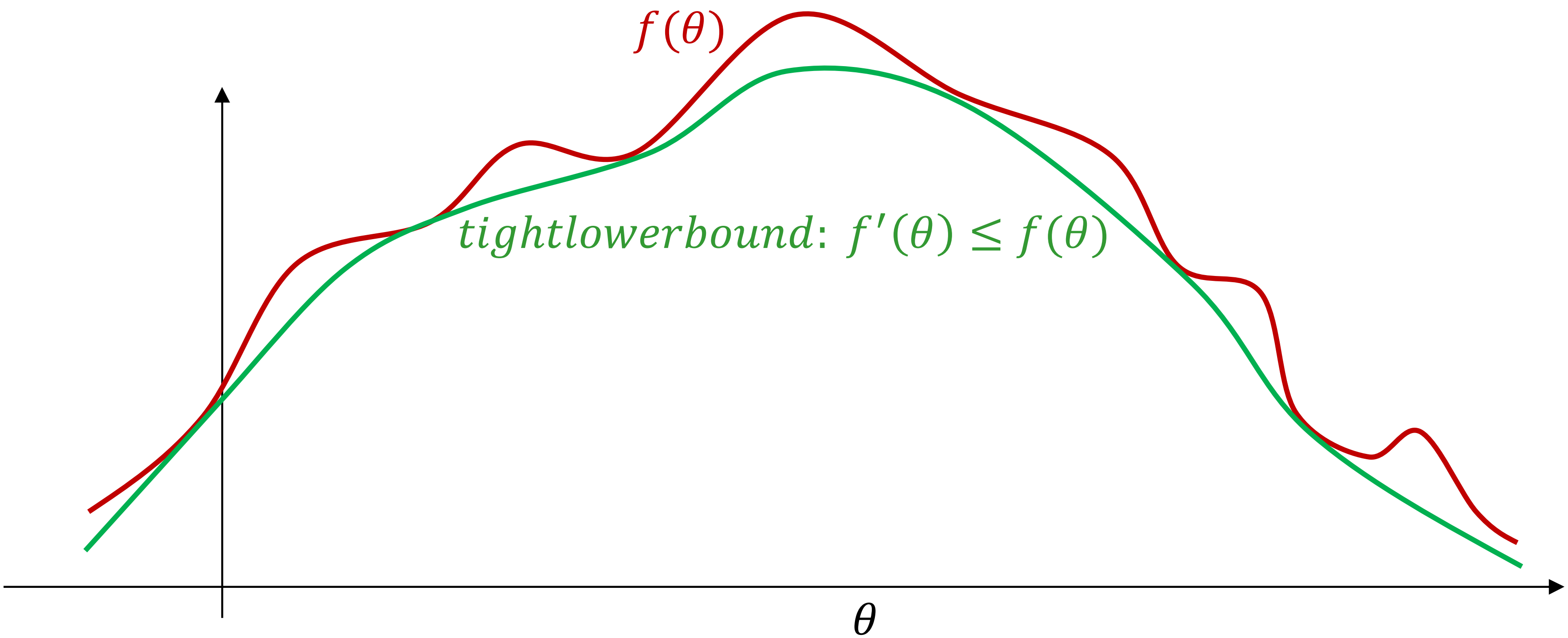
- This is the *variational lower bound* or ELBO on $\log P(o;\theta)$

The (variational) lower bound



- The lower bound is always at or below the original function

The (variational) lower bound



- The lower bound is always at or below the original function

- If it is a tight lower bound, the max of the lower bound can be expected to be near the max of the function
  - To make the lower bound tight, we need to choose $Q(h)$ properly

The two-step process

- By the concavity of the log function

$$\log P(o;\theta) \geq \sum_h Q(h) \log \frac{P(h,o;\theta)}{Q(h)}$$

- **Step 1:** Determine a $Q(h)$ that maximizes the RHS, using the current estimate of θ
  - Makes the bound tight
- **Step 2:** Fix $Q(h)$ and maximize the RHS with respect to θ to get the next estimate

The two-step process

- By the concavity of the log function

$$\log P(o;\theta) \geq \sum_h Q(h) \log \frac{P(h,o;\theta)}{Q(h)}$$

- **Step 1:** Determine a $Q(h)$ that maximizes the RHS, using the current estimate of θ
  - Makes the bound tight
- **Step 2:** Fix $Q(h)$ and maximize the RHS with respect to θ to get the next estimate

Maximizing w.r.t. Q(h)

$$\sum_h Q(h) \log \frac{P(h, o; \theta)}{Q(h)}$$

- Take the derivative w.r.t. $Q(h)$ for all h and equate to 0
  - With the constraint that $Q(h) \geq 0, \sum_h Q(h) = 1$

- If $Q(h)$ is specifically modeled by a neural net or some other restricted function, then we cannot simply take the derivative and equate to 0
  - We may need gradient descent, with backpropagation

- Note: The optimized $Q(h)$ depends on $P(h, o; \theta)$ and is a function of θ

Choosing a good $Q(h)$

- For any $P(h, o; \theta)$, the optimal $Q(h) = P(h|o; \theta)$:

$$\sum_h Q(h) \log \frac{P(h, o; \theta)}{Q(h)} = \sum_h P(h|o; \theta) \log \frac{P(h, o; \theta)}{P(h|o; \theta)}$$

$$= \sum_h P(h|o; \theta) \log P(o; \theta)$$

$$= \log P(o; \theta) \sum_h P(h|o; \theta) \quad = \log P(o; \theta)$$

- At this value of $Q(h)$ the variational lower bound achieves its maximum possible value

Choosing a good $Q(h)$

- Let $Q(h) = P(h|o;\theta')$

$$\log P(o;\theta) \geq \sum_h P(h|o;\theta') \, \log \frac{P(h,o;\theta)}{P(h|o;\theta')}$$

- Let

$$J(\theta,\theta') = \sum_h P(h|o;\theta') \log \frac{P(h,o;\theta)}{P(h|o;\theta')}$$

- We get

$$\log P(o;\theta) \geq J(\theta,\theta')$$

- And

$$\log P(o;\theta) = J(\theta,\theta)$$

Choosing a good $Q(h)$

- Let $Q(h) = P(h|o;\theta')$

$$\log P(o;\theta) \geq \sum_h P(h|o;\theta') \log \frac{P(h,o;\theta)}{P(h|o;\theta')}$$

- Let

$$J(\theta,\theta') = \sum_h P(h|o;\theta') \log \frac{P(h,o;\theta)}{P(h|o;\theta')}$$

- We get

$$\log P(o;\theta) \geq J(\theta,\theta')$$

- And

$$\log P(o;\theta) = J(\theta,\theta)$$

$$P(o;\theta) = J(\theta,\theta)$$

$$J(\theta,\theta) = \sum_h P(h|o;\theta) \log \frac{P(h,o;\theta)}{P(h|o;\theta)}$$

$$= \sum_h P(h|o;\theta) \log P(o;\theta)$$

$$\log P(o;\theta) \sum_h P(h|o;\theta) \quad = \log P(o;\theta)$$

Expectation Maximization

- We have

$$J(\theta, \theta') = \sum_h \overbrace{P(h|o; \theta')}^{Q(h)} \log \frac{P(h, o; \theta)}{P(h|o; \theta')}$$

- where

$$\log P(o; \theta) \geq J(\theta, \theta')$$

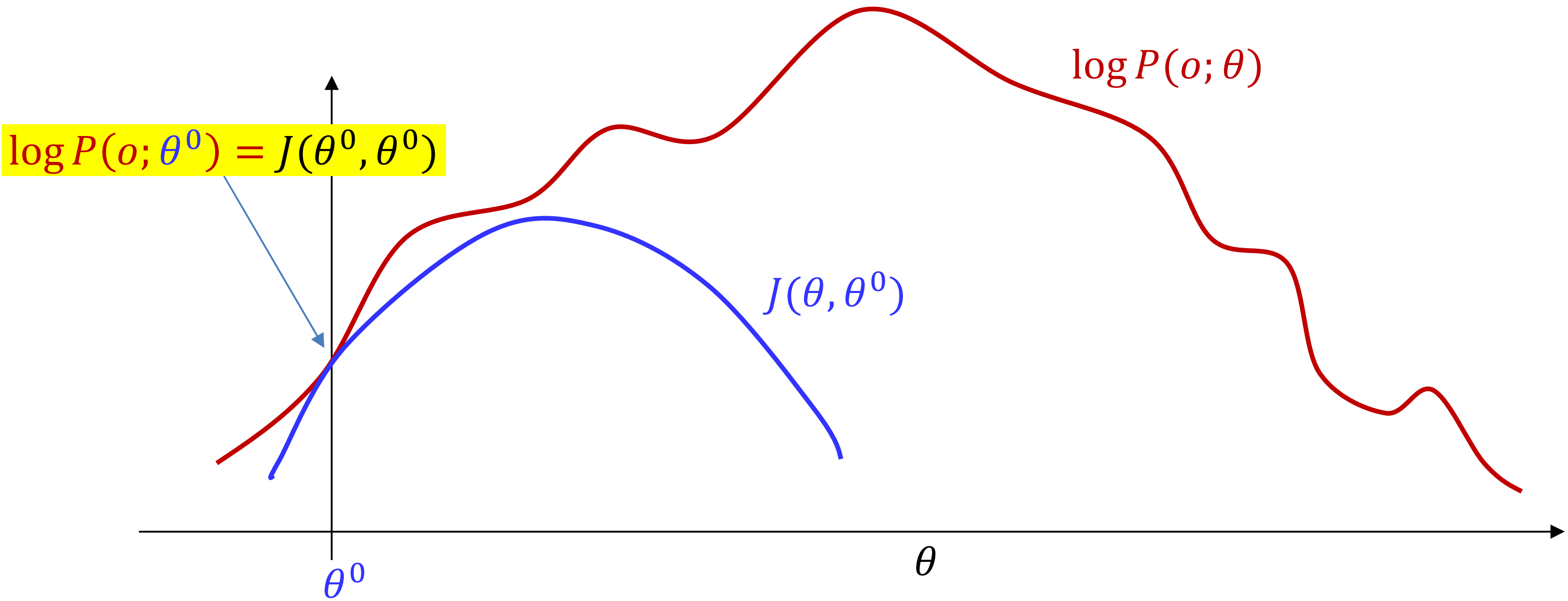
- And

$$\log P(o; \theta) = J(\theta, \theta)$$

- This gives us the following iterative algorithm that guarantees non-decreasing $P(o; \theta)$ with iterations:

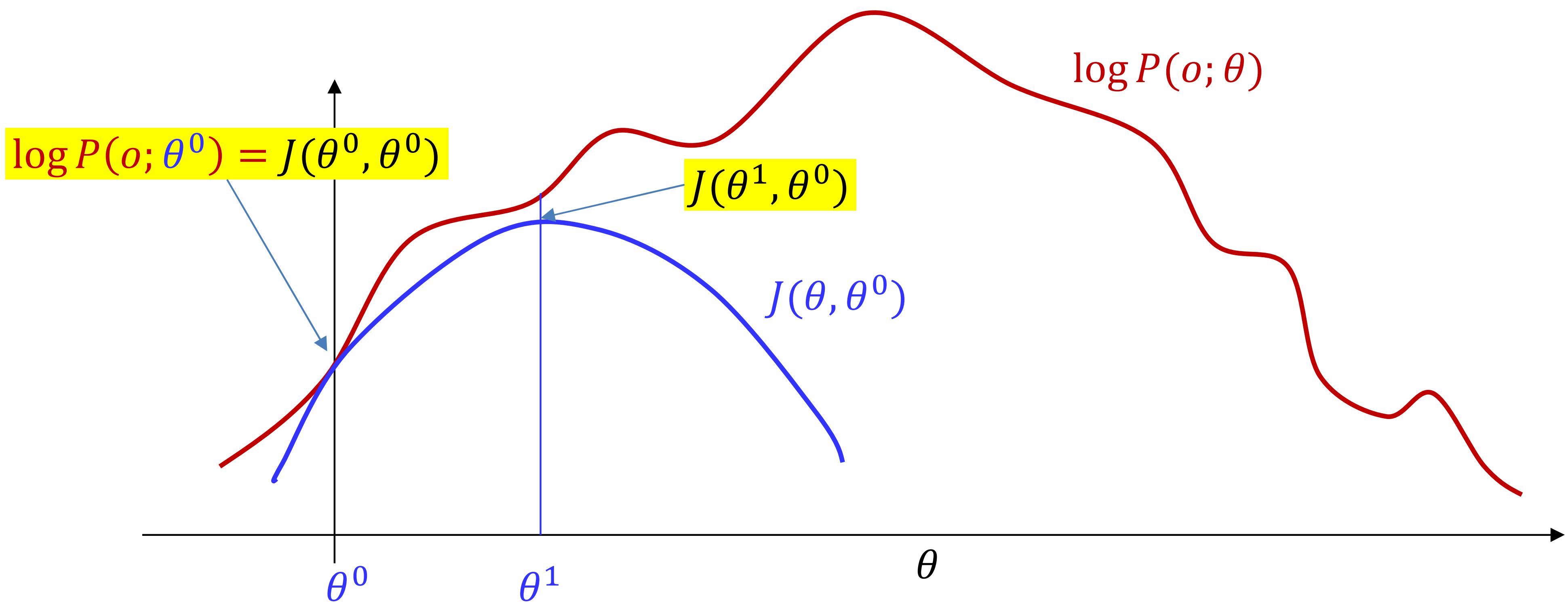
$$\theta^{k+1} \leftarrow \operatorname*{argmax}_\theta J\left(\theta, \theta^k\right)$$

$$\theta^{k+1} \leftarrow \underset{\theta}{\operatorname{argmax}} J(\theta, \theta')$$



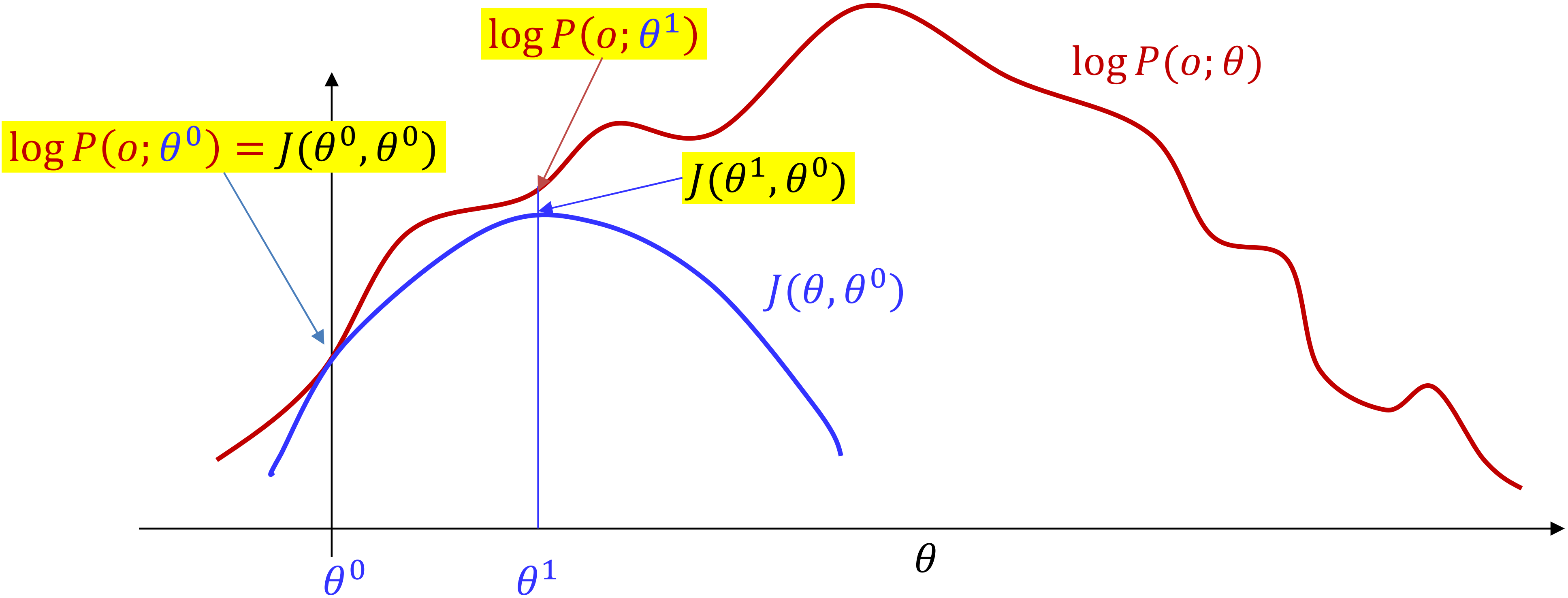
- Initialize θ^0
- Construct $J(\theta, \theta^0)$
  - It touches $\log P(o;\theta)$ at θ^0 because $\log P(o;\theta^0) = J(\theta^0,\theta^0)$

$$\theta^{k+1} \leftarrow \underset{\theta}{\operatorname{argmax}} J(\theta, \theta')$$



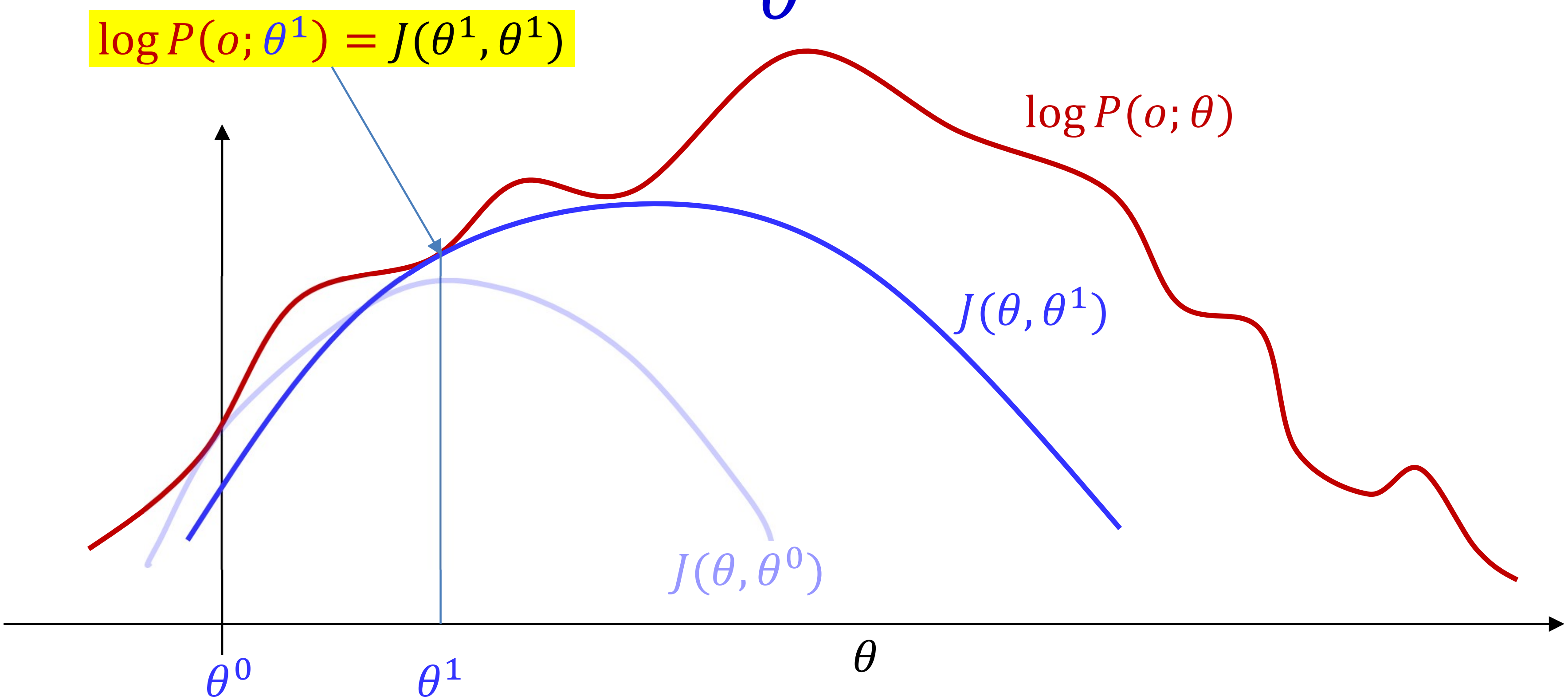
- Find $\theta^1 = \underset{\theta}{\operatorname{argmax}} J(\theta, \theta^0)$
  - $J(\theta^1, \theta^0) \geq J(\theta^0, \theta^0)$ (since you're maximizing $J(\theta, \theta^0)$ w.r.t θ)

$$\theta^{k+1} \leftarrow \underset{\theta}{\operatorname{argmax}} J(\theta, \theta')$$



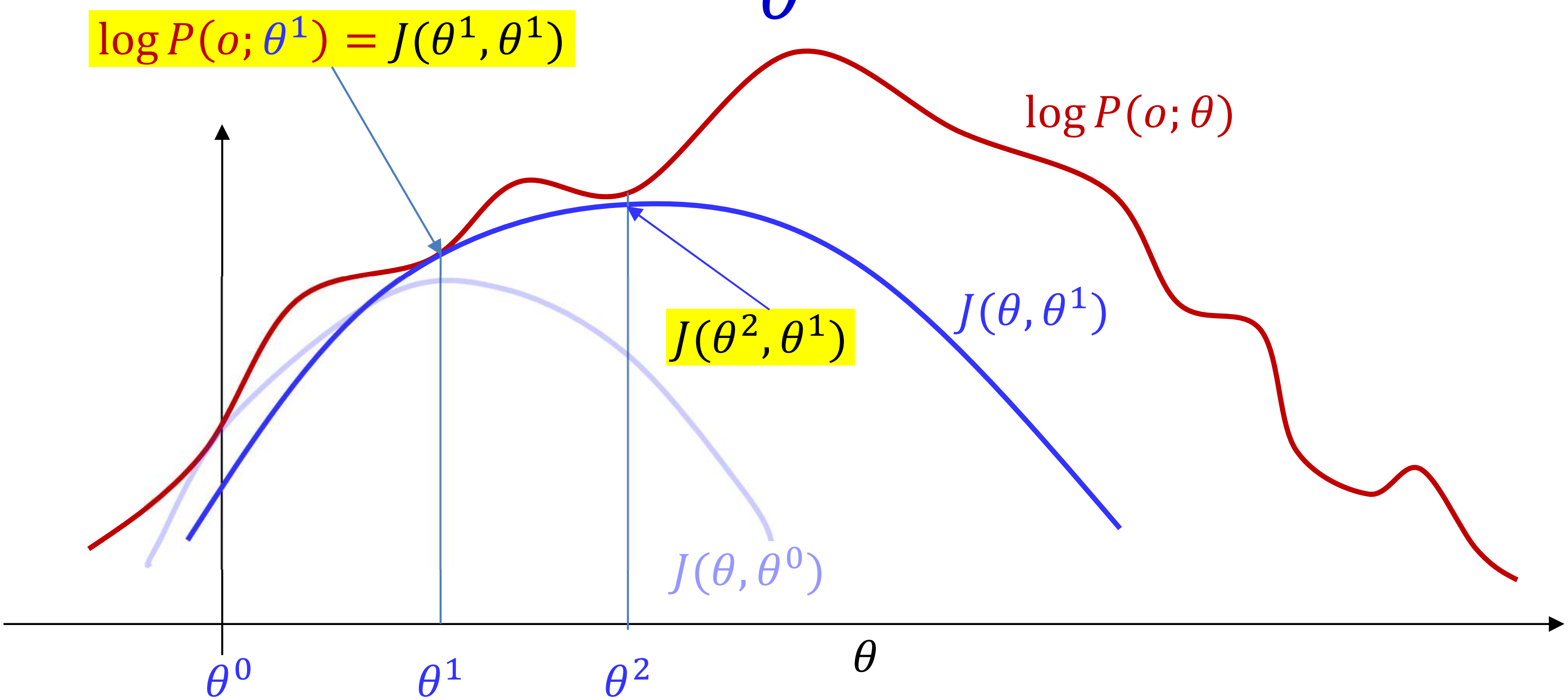
- Find $\theta^1 = \underset{\theta}{\operatorname{argmax}} J(\theta, \theta^0)$
  - $J(\theta^1, \theta^0) \geq J(\theta^0, \theta^0)$ (since you're maximizing $J(\theta, \theta^0)$ w.r.t θ)
- $\log P(o; \theta^1) \geq J(\theta^1, \theta^0)$
  - since $J(\theta, \theta^0)$ is a lower bound on $\log P(o; \theta)$
- So the iteration increases $\log P(o; \theta)$

$$\theta^{k+1} \leftarrow \underset{\theta}{\operatorname{argmax}} J(\theta, \theta')$$



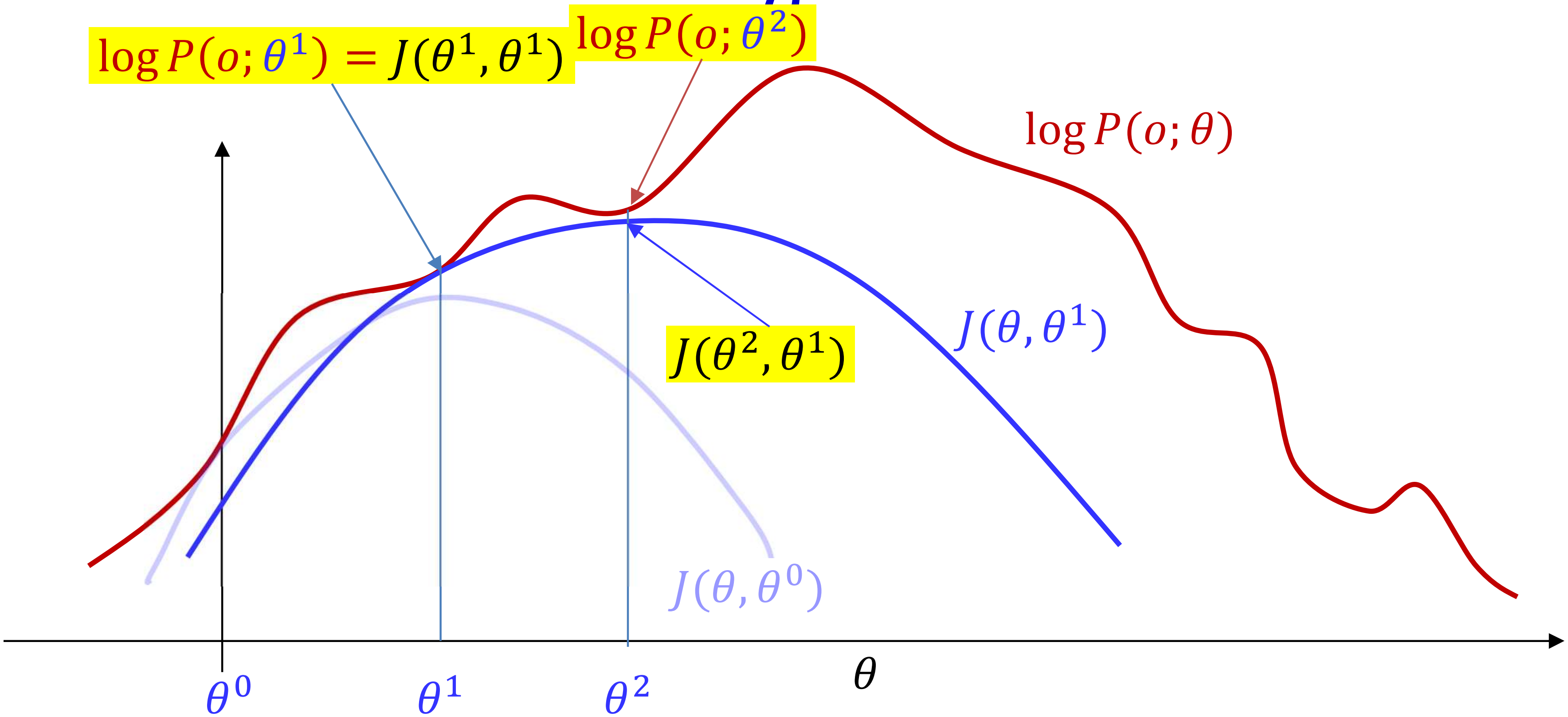
- Construct $J(\theta, \theta^1)$
  - It touches $\log P(o; \theta)$ at θ^1 because $\log P(o; \theta^1) = J(\theta^1, \theta^1)$

$$\theta^{k+1} \leftarrow \underset{\theta}{\operatorname{argmax}} J(\theta, \theta')$$



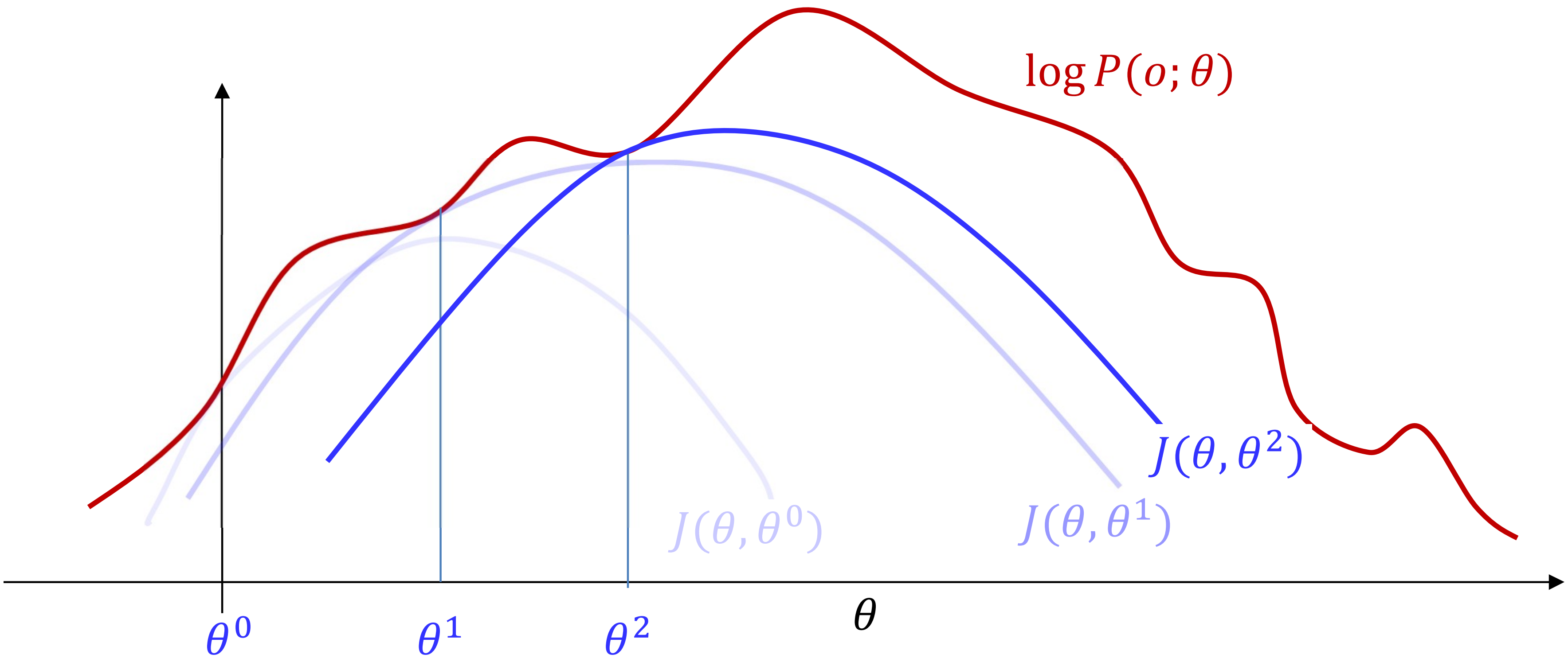
- Find $\theta^2 = \underset{\theta}{\operatorname{argmax}} J(\theta, \theta^1)$
  - $J(\theta^2, \theta^1) \geq J(\theta^1, \theta^1)$ (since you're maximizing $J(\theta, \theta^1)$ w.r.t θ)

$$\theta^{k+1} \leftarrow \underset{\theta}{\operatorname{argmax}} J(\theta, \theta')$$



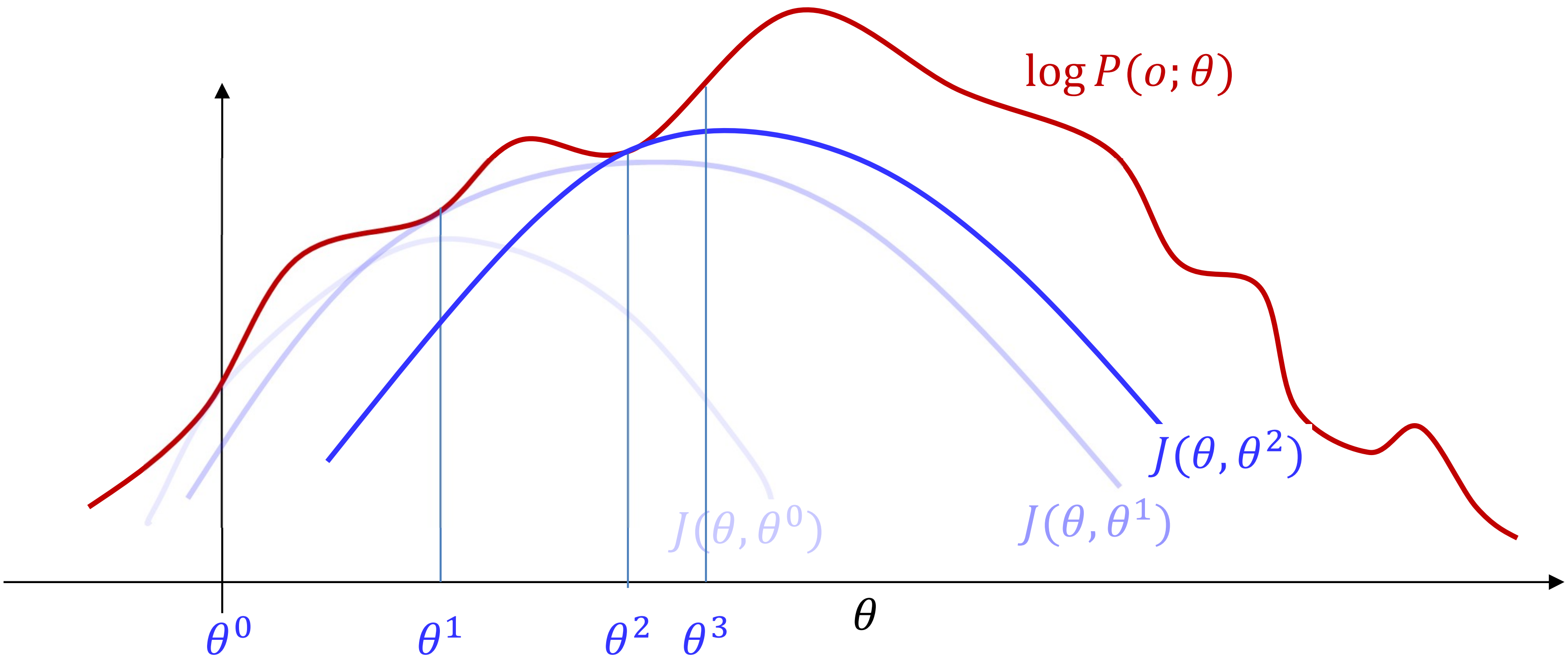
- Find $\theta^2 = \underset{\theta}{\operatorname{argmax}} J(\theta, \theta^1)$
  - $J(\theta^2, \theta^1) \geq J(\theta^1, \theta^1)$ (since you're maximizing $J(\theta, \theta^1)$ w.r.t θ)
- $\log P(o; \theta^2) \geq J(\theta^2, \theta^1)$
  - Since $J(\theta, \theta^1)$ is a lower bound on $\log P(o; \theta)$
- So the iteration increases $\log P(o; \theta)$

$$\theta^{k+1} \leftarrow \underset{\theta}{\operatorname{argmax}} J(\theta, \theta')$$



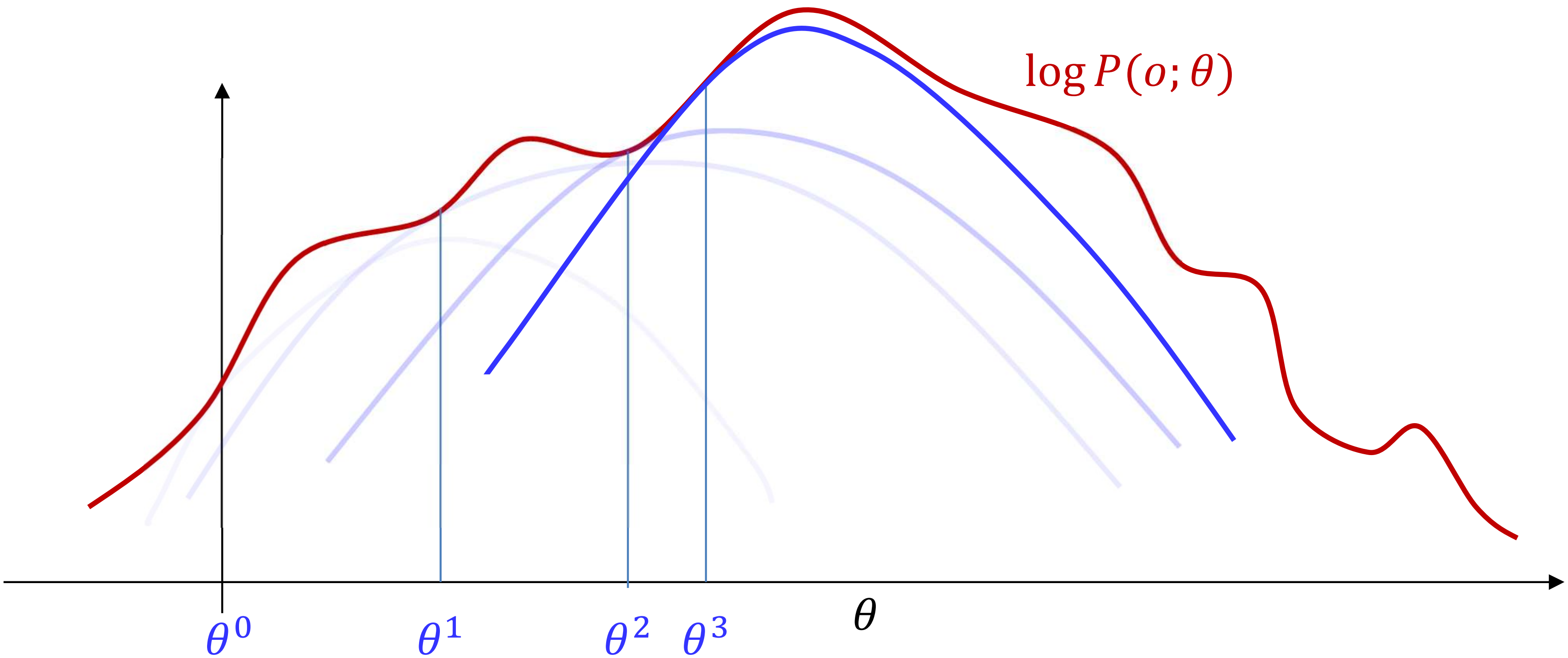
- Repeat the steps:
  - Compose $J(\theta, \theta^k)$ to "touch" $\log P(o;\theta)$ at the current estimate θ^k
  - Set $\theta^{k+1} \leftarrow \underset{\theta}{\operatorname{argmax}} J(\theta, \theta^k)$
- Each step is guaranteed to increase (or at least not decrease) $\log P(o;\theta)$
  - Stop when $\log P(o;\theta)$ stops increasing

$$\theta^{k+1} \leftarrow \underset{\theta}{\operatorname{argmax}} J(\theta, \theta')$$



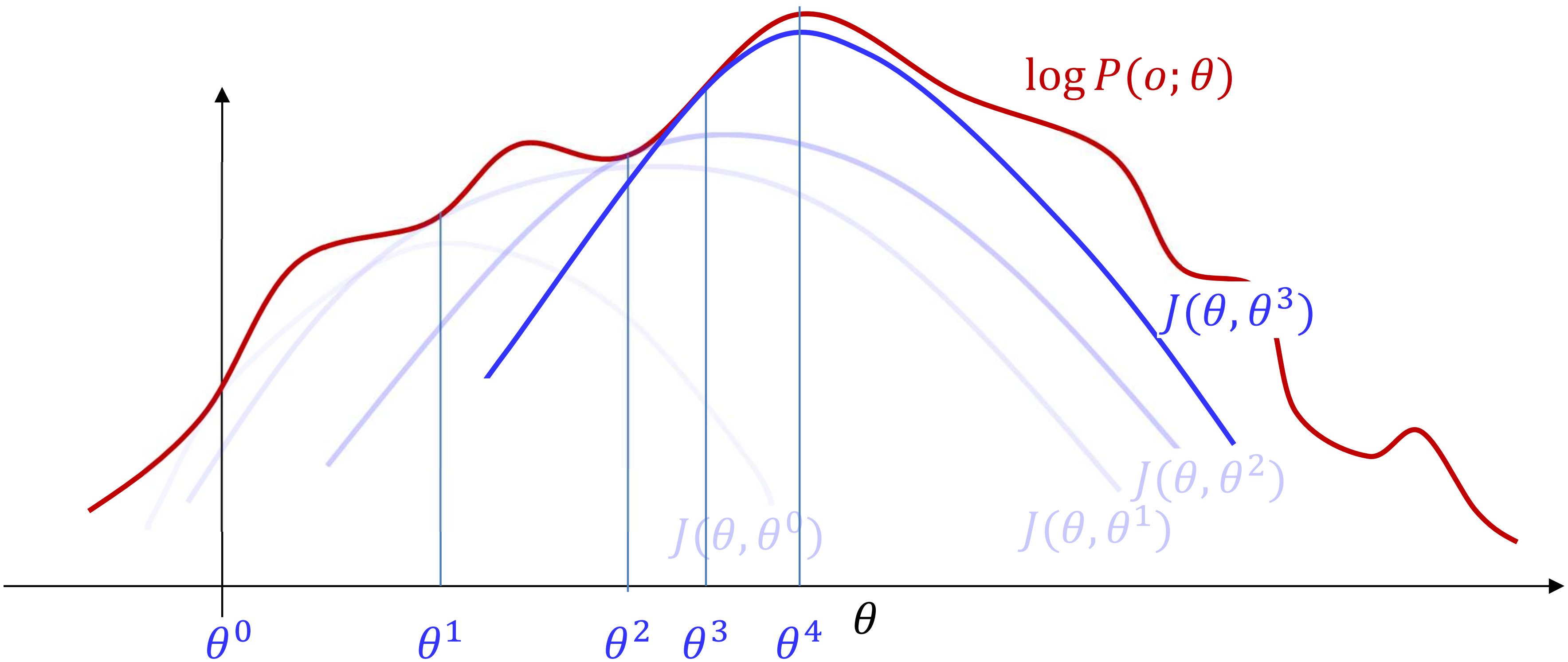
- Repeat the steps:
  - Compose $J(\theta, \theta^k)$ to "touch" $\log P(o;\theta)$ at the current estimate θ^k
  - Set $\theta^{k+1} \leftarrow \underset{\theta}{\operatorname{argmax}} J(\theta, \theta^k)$
- Each step is guaranteed to increase (or at least not decrease) $\log P(o;\theta)$
  - Stop when $\log P(o;\theta)$ stops increasing

$$\theta^{k+1} \leftarrow \underset{\theta}{\operatorname{argmax}} J(\theta, \theta')$$



- Repeat the steps:
  - Compose $J(\theta, \theta^k)$ to “touch” $\log P(o;\theta)$ at the current estimate θ^k
  - Set $\theta^{k+1} \leftarrow \underset{\theta}{\operatorname{argmax}} J(\theta, \theta^k)$
- Each step is guaranteed to increase (or at least not decrease) $\log P(o;\theta)$
  - Stop when $\log P(o;\theta)$ stops increasing

$$\theta^{k+1} \leftarrow \underset{\theta}{\operatorname{argmax}} J(\theta, \theta')$$



- Repeat the steps:
  - Compose $J(\theta, \theta^k)$ to "touch" $\log P(o;\theta)$ at the current estimate θ^k
  - Set $\theta^{k+1} \leftarrow \underset{\theta}{\operatorname{argmax}} J(\theta, \theta^k)$
- Each step is guaranteed to increase (or at least not decrease) $\log P(o;\theta)$
  - Stop when $\log P(o;\theta)$ stops increasing

Expectation Maximization

- Initialize θ^0
- $k = 0$
- Iterate (over k) until $\log P(O; \theta)$ converges:
  - Construct ELBO function

$$J(\theta, \theta^k) = \sum_{o \in O} \sum_{h} P(h|o; \theta^k) \log \frac{P(h, o; \theta)}{P(h|o; \theta^k)}$$

  - Maximization step

$$\theta^{k+1} \leftarrow \underset{\theta}{\operatorname{argmax}} J(\theta, \theta^k)$$

- Let's simplify a bit

Expectation Maximization

- Initialize θ^0
- $k = 0$
- Iterate (over k) until $\log P(O;\theta)$ converges:
  - Construct ELBO function

$$J(\theta,\theta^k) = \sum_{o\in O}\sum_{h} P(h|o;\theta^k)\log P(h,o;\theta) - \sum_{o\in O}\sum_{h} P(h|o;\theta^k)\log P(h|o;\theta^k)$$

  - Maximization step

$$\theta^{k+1} \leftarrow \operatorname*{argmax}_{\theta} J(\theta,\theta^k)$$

Expectation Maximization

- Initialize θ^0
- $k = 0$
- Iterate (over k) until $\log P(O;\theta)$ converges:
  - Construct ELBO function

Not a function of θ

$$J(\theta, \theta^k) = \sum_{o \in O} \sum_h P(h|o;\theta^k) \log P(h,o;\theta) - \sum_{o \in O} \sum_h P(h|o;\theta^k) \log P(h|o;\theta^k)$$

Can be ignored for maximization

  - Maximization step

$$\theta^{k+1} \leftarrow \underset{\theta}{\operatorname{argmax}}\, J(\theta, \theta^k)$$

Expectation Maximization

- Initialize θ^0
- $k = 0$
- Iterate (over k) until $\log P(O;\theta)$ converges:
  - Expectation Step:

    Compute $P(h|o;\theta^k)$ for all $o \in O$ for all h

  - Maximization step

$$\theta^{k+1} \leftarrow \underset{\theta}{\operatorname{argmax}} \sum_{o \in O} \sum_{h} P(h|o;\theta^k) \log P(h,o;\theta)$$

The two-step process

- By the concavity of the log function

$$\log P(o;\theta) \geq \sum_h Q(h) \log \frac{P(h,o;\theta)}{Q(h)}$$

- **Step 1:** Determine a $Q(h)$ that maximizes the RHS, using the current estimate of θ
  - The best case value is $Q(h) = P(h|o;\theta)$ using the current estimate of θ
- **Step 2:** Fix $Q(h)$ and maximize the RHS with respect to θ to get the next estimate

The two-step process

- By the concavity of the log function

$$\log P(o;\theta) \geq \sum_h Q(h) \log \frac{P(h,o;\theta)}{Q(h)}$$

- **Step 1:** Determine a $Q(h)$ that maximizes the RHS, using the current estimate of θ
  - The best case value is $Q(h) = P(h|o;\theta)$ using the current estimate of θ
- **Step 2:** Fix $Q(h)$ and maximize the RHS with respect to θ to get the next estimate

$$\theta \leftarrow \arg\max_\theta \sum_h Q(h)(\log P(h,o;\theta) - \log Q(h))$$

The two-step process

- By the concavity of the log function

$$\log P(o;\theta) \geq \sum_h Q(h) \log \frac{P(h,o;\theta)}{Q(h)}$$

- **Step 1:** Determine a $Q(h)$ that maximizes the RHS, using the current estimate of θ
  - The best case value is $Q(h) = P(h|o;\theta)$ using the current estimate of θ
- **Step 2:** Fix $Q(h)$ and maximize the RHS with respect to θ to get the next estimate

$$\theta \leftarrow \arg\max_\theta \sum_h Q(h) \log P(h,o;\theta)$$

The two-step process

- By the concavity of the log function

$$\log P(o;\theta) \geq \sum_h Q(h) \log \frac{P(h,o;\theta)}{Q(h)}$$

Training by maximizing a variational lower bound

- **Step 1:** Determine a $Q(h)$ that maximizes the RHS, using the current estimate of θ
  – The best case value is $Q(h) = P(h|o;\theta)$ using the current estimate of θ
- **Step 2:** Fix $Q(h)$ and maximize the RHS with respect to θ to get the next estimate

$$\theta \leftarrow \arg\max_\theta \sum_h Q(h) \log P(h,o;\theta)$$

Special case: Expectation Maximization

- Initialize θ^0
- $k = 0$
- Iterate (over k) until $\log P(O;\theta)$ converges:
  - Expectation Step: $Q(h) = P(h|o;\theta^k)$

    Compute $P(h|o;\theta^k)$ for all $o \in O$ for all h

  - Maximization step

$$\theta^{k+1} \leftarrow \underset{\theta}{\operatorname{argmax}} \sum_{o \in O} \sum_{h} P(h|o;\theta^k) \log P(h,o;\theta)$$

Poll 2 (@1761,@1762)

Mark all that are correct about the EM algorithm

- It is an iterative algorithm that can be used to estimate probability distributions when the data are incomplete and have missing components or variables
- It iteratively maximizes an "ELBO" function with respect to model parameters
- It provides a closed form formula to estimate the parameters of the distribution

Mark all that are true of the ELBO (Empirical Lower Bound) function

- It is a lower bound on the actual log probability of the training data as computed by the model
- It is a function of the model parameters
- There are some settings of the model parameters where the ELBO can be greater than the log probability of the training data

Poll 2

Mark all that are correct about the EM algorithm

- **It is an iterative algorithm that can be used to estimate probability distributions when the data are incomplete and have missing components or variables**
- **It iteratively maximizes an "ELBO" function with respect to model parameters**
- It provides a closed form formula to estimate the parameters of the distribution

Mark all that are true of the ELBO (Empirical Lower Bound) function

- **It is a lower bound on the actual log probability of the training data as computed by the model**
- **It is a function of the model parameters**
- There are some settings of the model parameters where the ELBO can be greater than the log probability of the training data

That's so much math, but what does it really do?

- What does EM practically do when we have missing data?
  - What is the intuition behind how it resolves the problem?

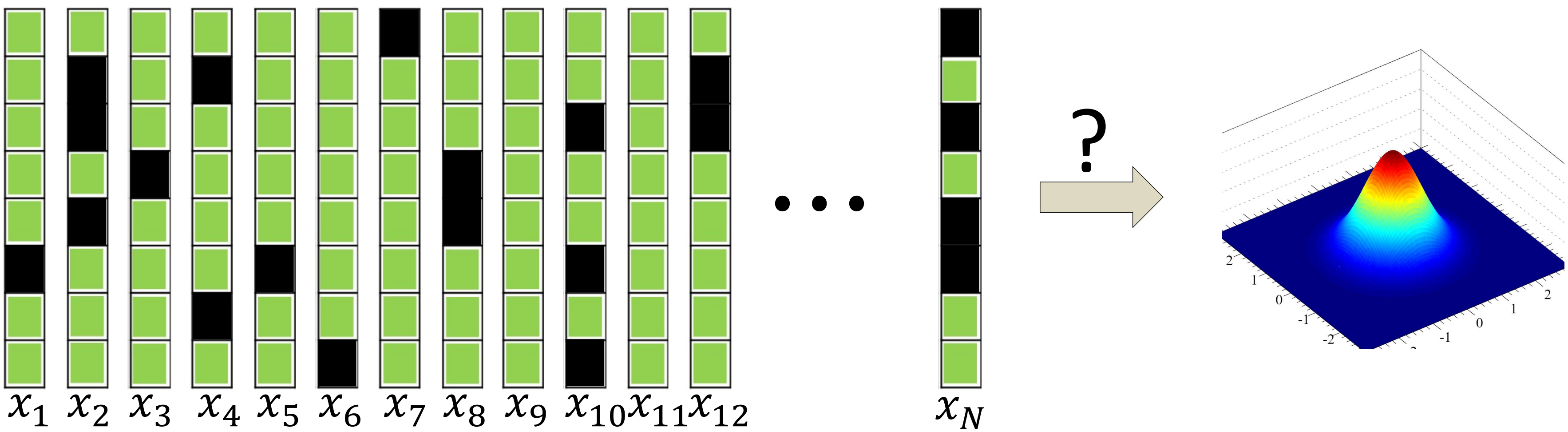
Recap: Maximum Likelihood Estimation

- Sometimes the data provided may be incomplete
  - Insufficient to estimate your model parameters directly

- This could be because the data themselves have missing components
  - E.g. Data vectors have some missing components

- Or because of the structure of the network
  - Mixture models, multi-stage Generative models

Recall this: Gaussian estimation with incomplete vectors



- These are the actual data we have: A set $O = \{o_1, \ldots, o_N\}$ of *incomplete* vectors
  - Comprising only the *observed* components of the data

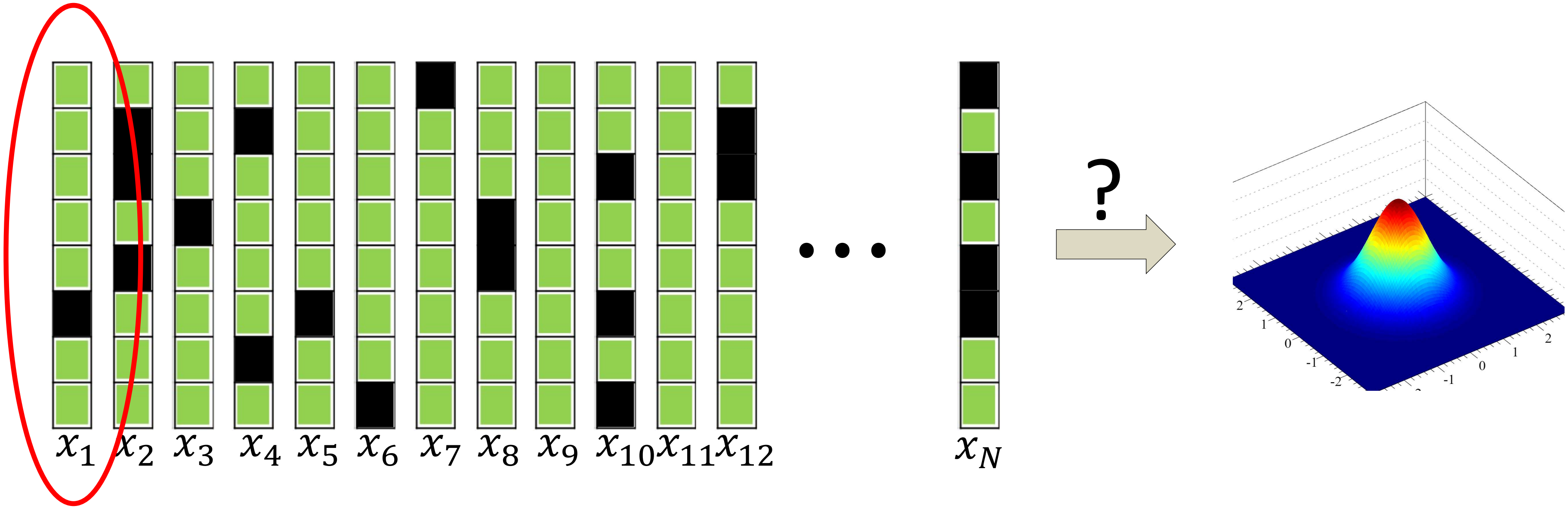
- We are *missing* the data $M = \{m_1, \ldots, m_N\}$
  - Comprising the *missing* components of the data

- The *complete* data includes both the observed and missing components

$$X = \{x_1, \ldots, x_N\}, \qquad x_i = (o_i, m_i)$$

  - Keep in mind that at the complete data are *not* available (the missing components are missing)

Let's look at a single vector

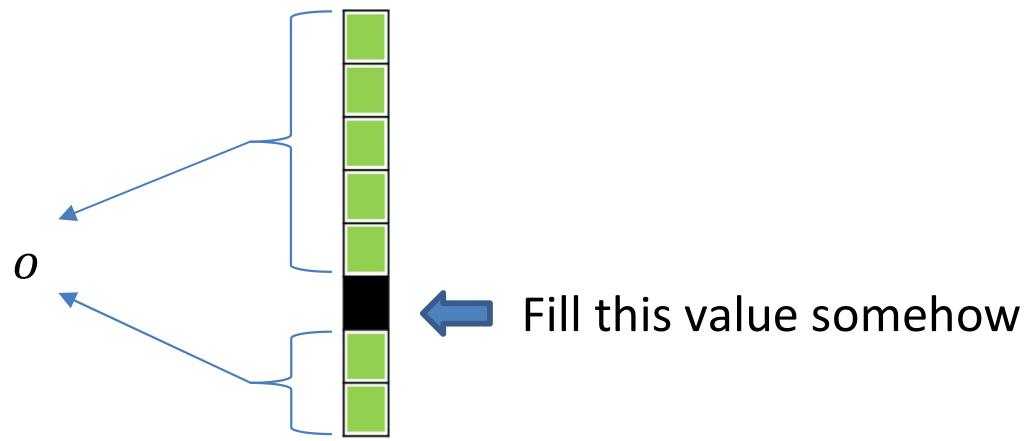


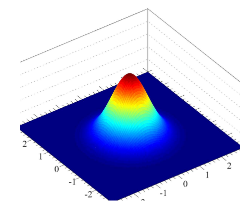
- These are the actual data we have: A set $O = \{o_1, \dots, o_N\}$ of *incomplete* vectors
  - Comprising only the *observed* components of the data
- We are *missing* the data $M = \{m_1, \dots, m_N\}$
  - Comprising the *missing* components of the data
- The *complete* data includes both the observed and missing components

$$X = \{x_1, \dots, x_N\}, \qquad x_i = (o_i, m_i)$$

  - Keep in mind that at the complete data are *not* available (the missing components are missing)

Lets look at a single vector

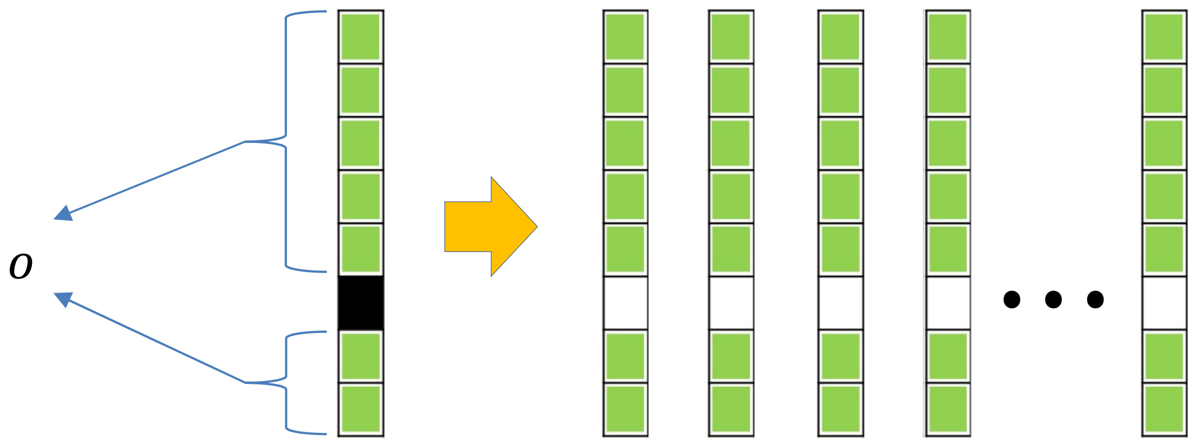


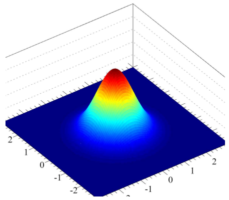


- We will try to complete the vector by filling in the missing value with *plausible* values that match the observed components

- Plausible: Values that "go with" the observed values, according to the distribution of the data

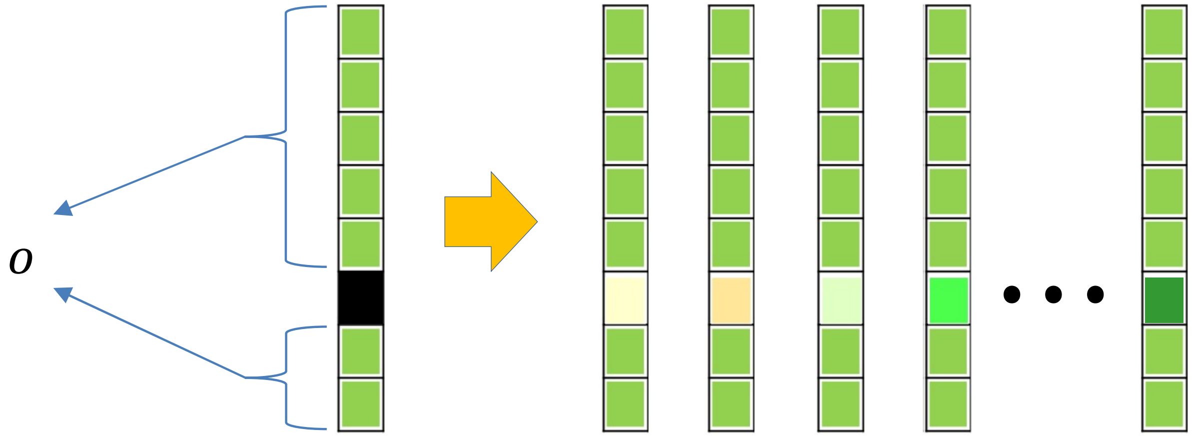
Lets look at a single vector

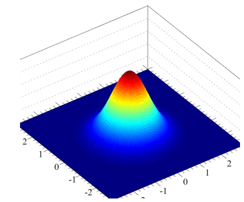




- Question: If we have a very large number of vectors from the Gaussian, all with the same observed components o, what would their missing components be?

Let's look at a single vector

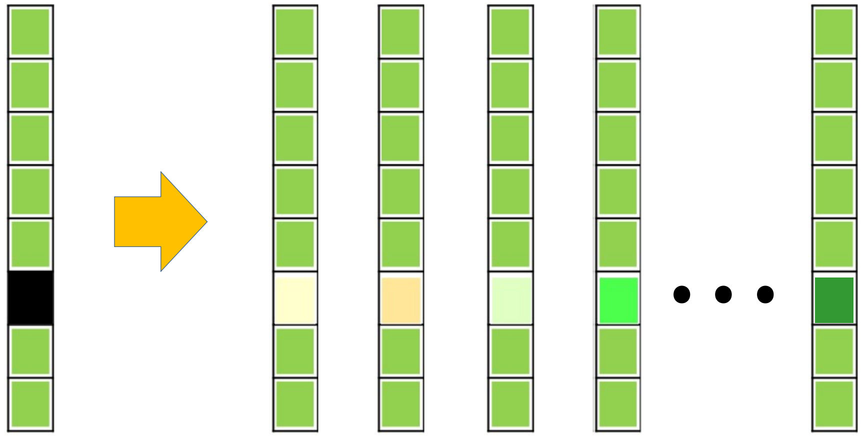




- Question: If we have a very large number of vectors from the Gaussian, all with the same observed components o, what would their missing components be?

- We would see every possible value, but in proportion to their probability: $P(m|o)$ (conditioned on the observations)

Completing incomplete vectors

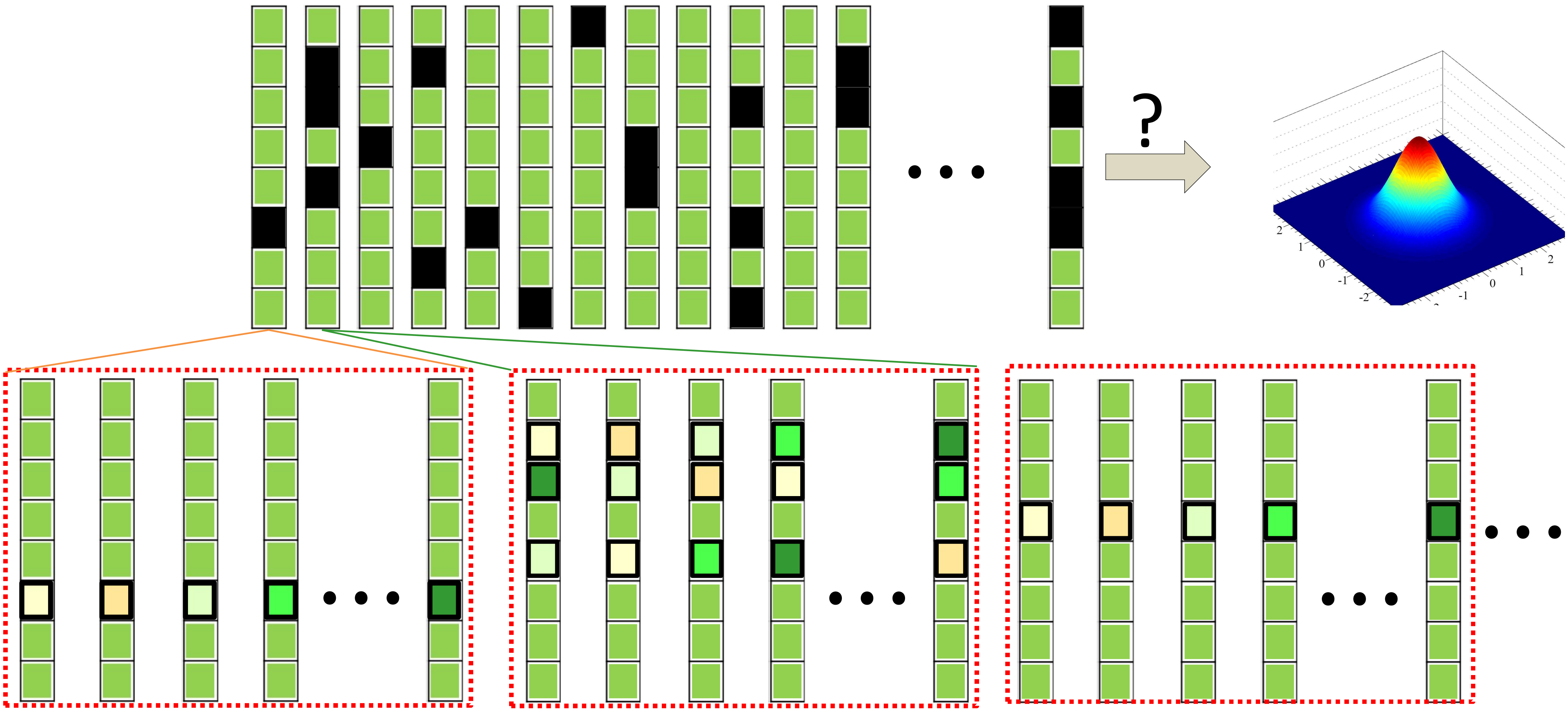


$in\ proportion: P(*|o)$

- Complete vector by filling up the missing components with *every possible value*
  - I.e. make many complete "clones" of the incomplete vector

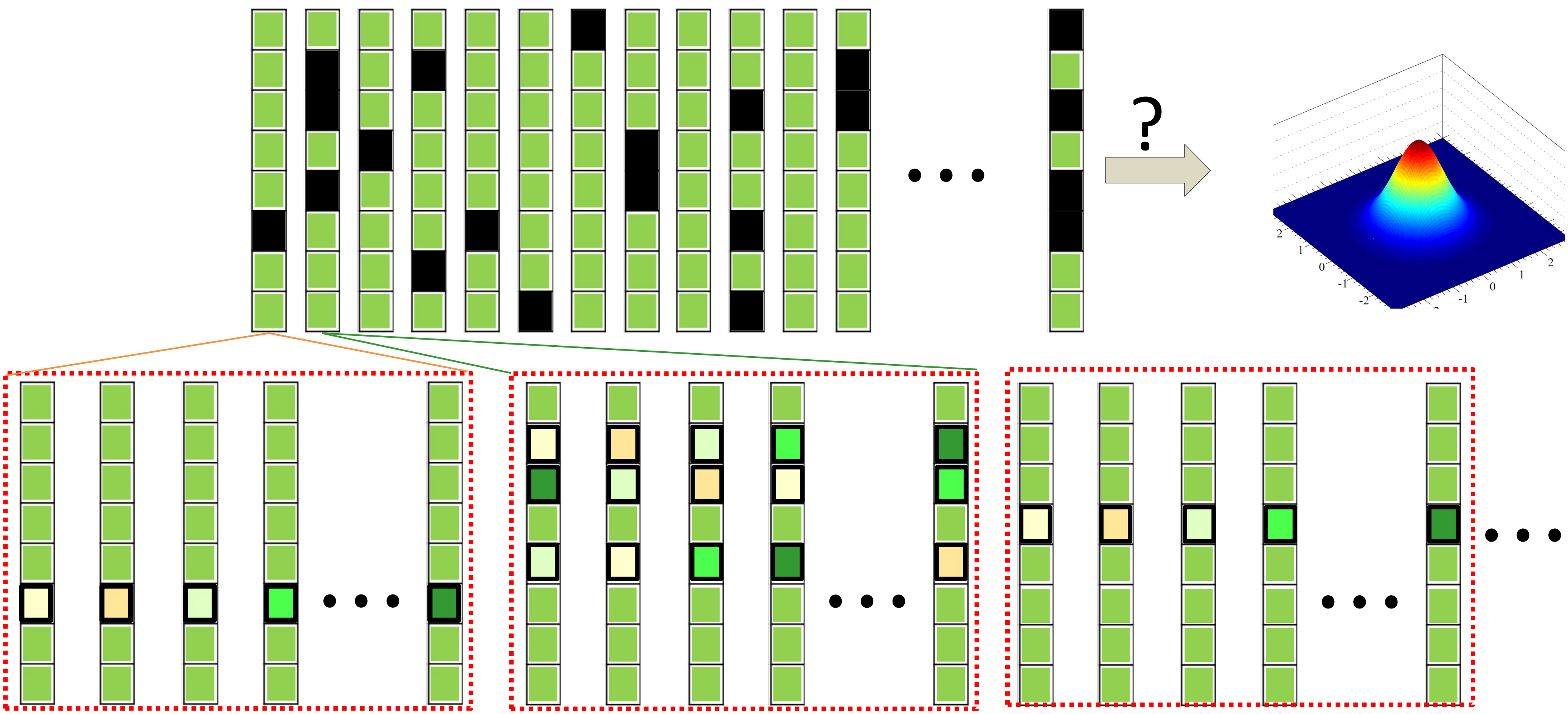
- But assign a *proportion* to each value
  - Proportion is $P(m|o)$
    - Which can be computed if we know $P(x) = P(o, m)$

Gaussian estimation with incomplete vectors



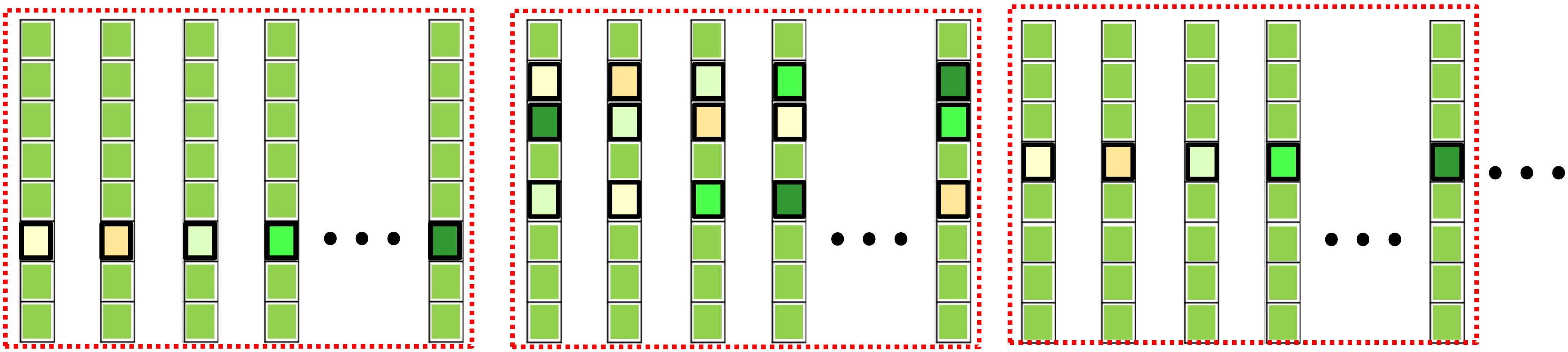
- "Expand" every incomplete vector out into all possibilities
  - In appropriate proportions $P(m|o)$
  - For already complete observations, there is no expansion
- Estimate the statistics from the expanded data

Gaussian estimation with incomplete vectors



- "Expand" every incomplete vector out into all possibilities
  - In appropriate proportions $P(m|o)$ From a previous estimate of the model
  - For already complete observations, there is no expansion
- Estimate the statistics from the expanded data

Estimating the Gaussian Parameters



- Compute the statistics from the (proportionately) expanded set
- Let $x_i(m)$ be the "completed" version of the observation o_i, when the missing components are filled with value m

$$x_i(m) = (m, o_i)$$

  – There will be one such vector for every value of m

$$\mu^{k+1} = \frac{1}{N} \sum_{x_i(m)} x_i(m)$$

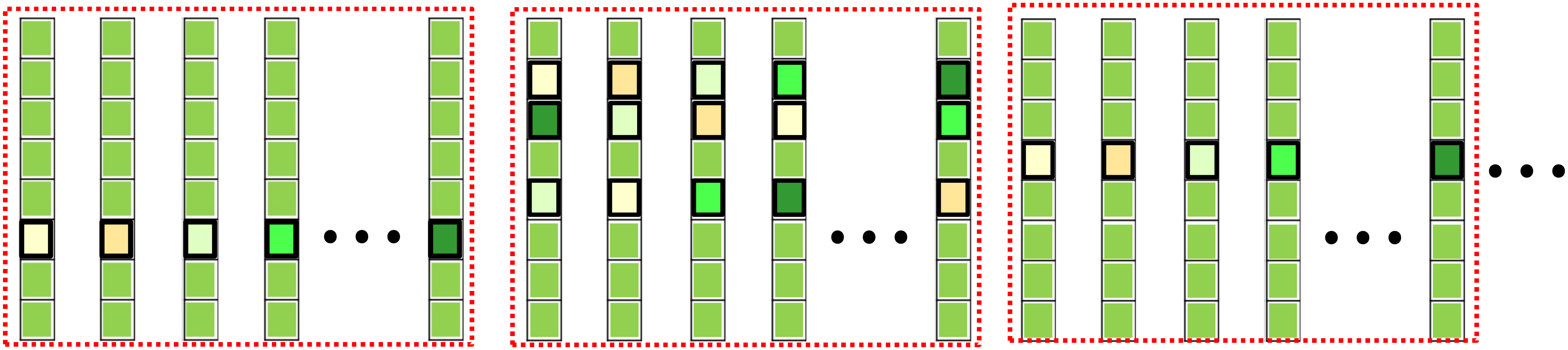
- We have several $x_i(m)$ for each o_i. Group the sum by o_i.
- Recall that for each o_i, the number of $x_i(m)$ for each m is proportion to $P(m|o;\theta^k)$.

$$\mu^{k+1} = \frac{1}{N_o N_{m|o}} \sum_{o \in O} \sum_m P(m|o;\theta^k) x_i(m) = \frac{1}{N_o} \sum_{o \in O} \frac{1}{N_{m|o}} \sum_m P(m|o;\theta^k) x_i(m)$$

- In the limit, if we consider *every* value of m

$$\mu^{k+1} = \frac{1}{N_o} \sum_{o \in O} \int_{-\infty}^{\infty} P\big(m \big| o;\theta^k\big) x_i(m) dm$$

Estimating the Gaussian Parameters



- Compute the statistics from the (proportionately) expanded set
- Let $x_i(m)$ be the "completed" version of the observation o_i, when the missing components are filled with value m

$$x_i(m) = (m, o_i)$$

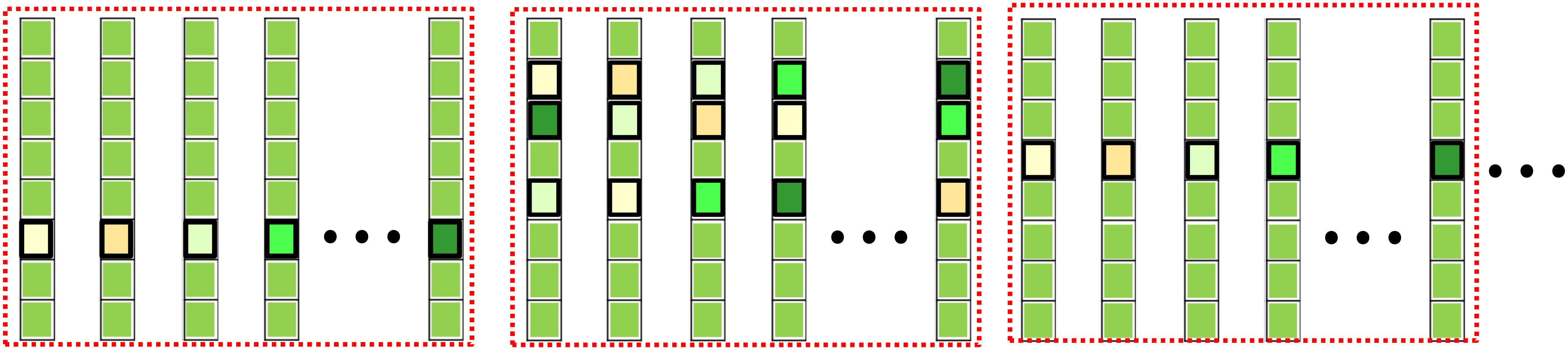
  – There will be one such vector for every value of m

- Estimate the statistics from the expanded data

$$\mu^{k+1} = \frac{1}{N}\sum_{o \in O}\int_{-\infty}^{\infty} P\left(m \middle| o; \theta^k\right) x_i(m)\, dm$$

$$\Sigma^{k+1} = \frac{1}{N}\sum_{o \in O}\int_{-\infty}^{\infty} P\left(m \middle| o; \theta^k\right)\left(x_i(m) - \mu^{k+1}\right)\left(x_i(m) - \mu^{k+1}\right)^T dm$$

EM for computing the Gaussian Parameters



- Initial $\theta^0 = (\mu^0, \Sigma^0)$
- Until $P(O;\theta)$ converges:

$$\mu^{k+1} = \frac{1}{N}\sum_{o\in O}\int_{-\infty}^{\infty} P\left(m\middle|o;\theta^k\right) x_i(m)\,dm$$

$$\Sigma^{k+1} = \frac{1}{N}\sum_{o\in O}\int_{-\infty}^{\infty} P\left(m\middle|o;\theta^k\right)(x_i(m)-\mu^{k+1})(x_i(m)-\mu^{k+1})^T\,dm$$

Where $x_i(m) = (m, o_i)$ and the parameters of $P\left(m\middle|o;\theta^k\right)$ are derived from the $P\left(x;\theta^k\right) = Gaussian(x;\mu^k,\Sigma^k)$

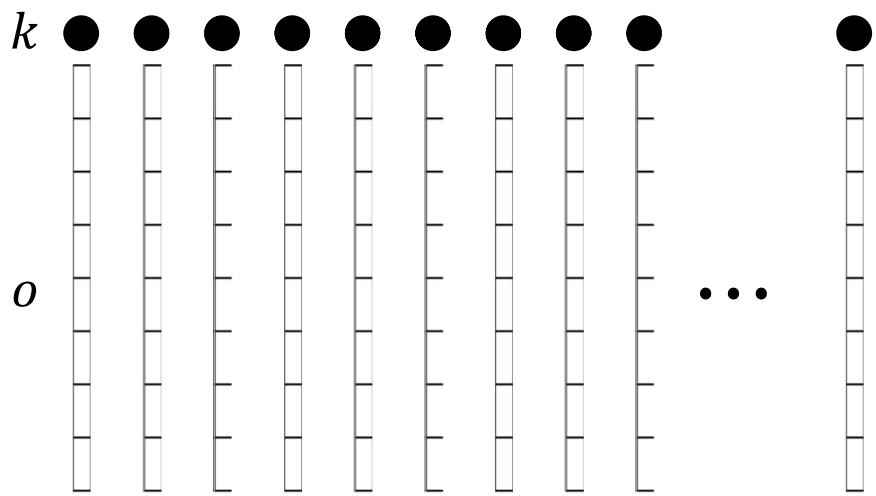
Recap: Maximum Likelihood Estimation

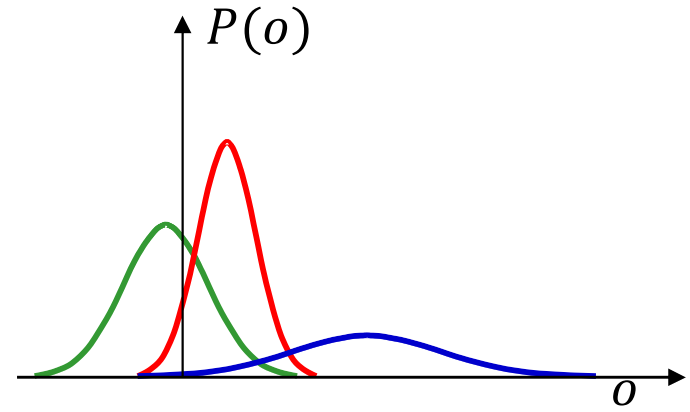
- Sometimes the data provided may be incomplete
  - Insufficient to estimate your model parameters directly

- This could be because the data themselves have missing components
  - E.g. Data vectors have some missing components

- Or because of the structure of the network
  - Mixture models, multi-stage Generative models

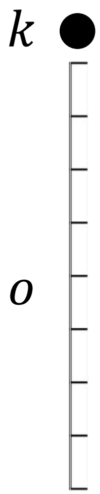
The GMM problem of incomplete data: missing information

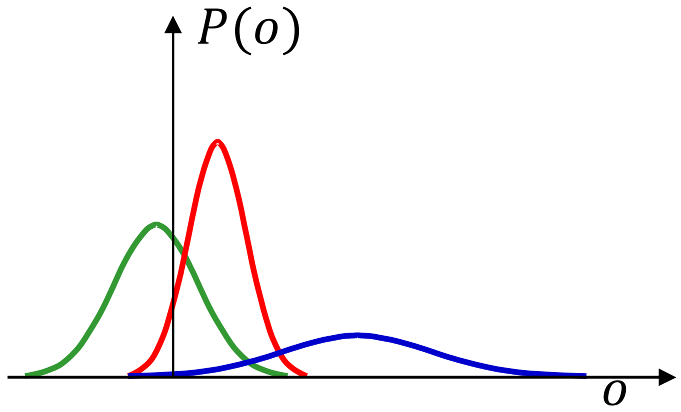




- Problem : We are not given the actual Gaussian for each observation
  - Our data are incomplete
- What we want : $(o_1, k_1), (o_2, k_2), (o_3, k_3)$ ...
- What we have: o_1, o_2, o_3 ...

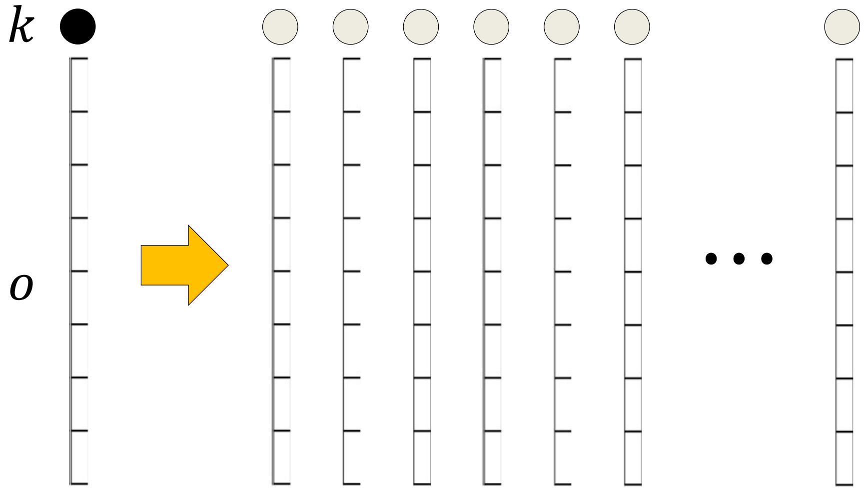
Consider a single vector

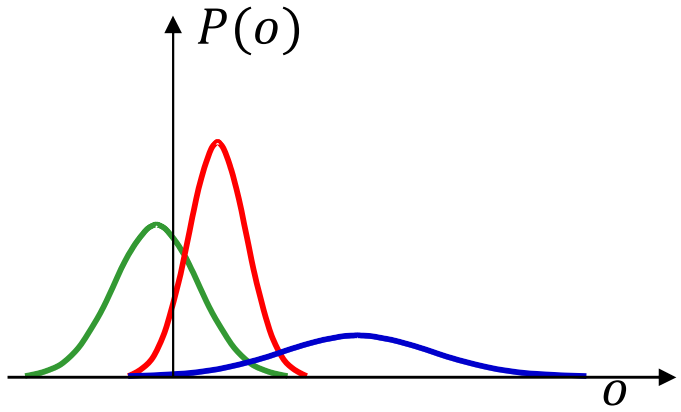




- *Every* Gaussian is capable of generating this vector
  – With different probabilities

Consider a single vector

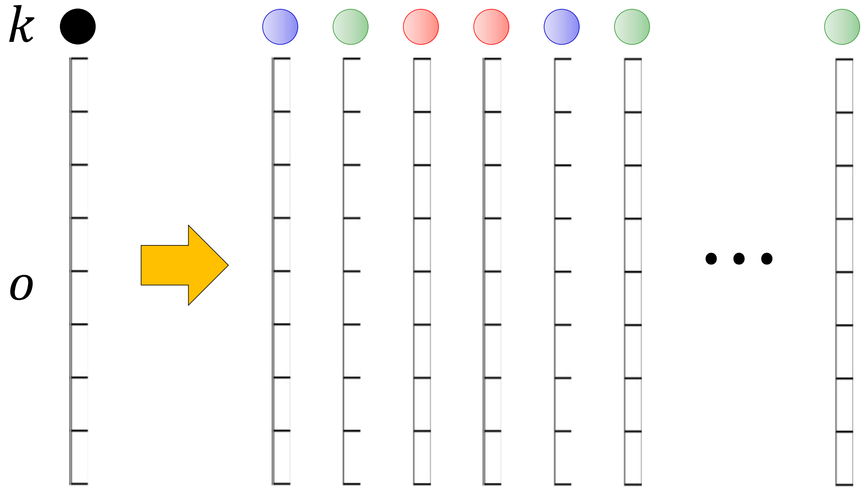


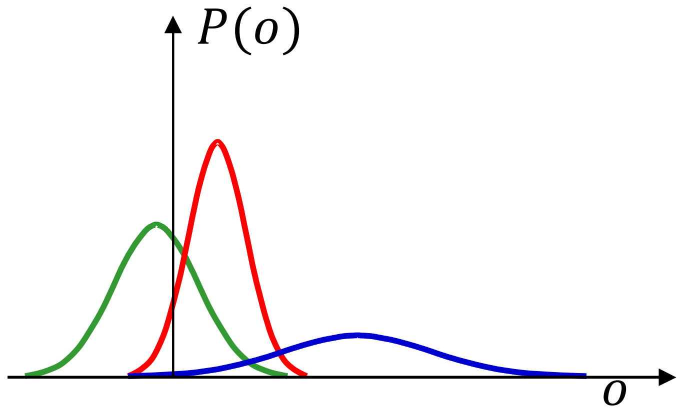


- *Every* Gaussian is capable of generating this vector
  - With different probabilities

- If we saw a large number of these vectors, how many of these would have come from each Gaussian?

Consider a single vector



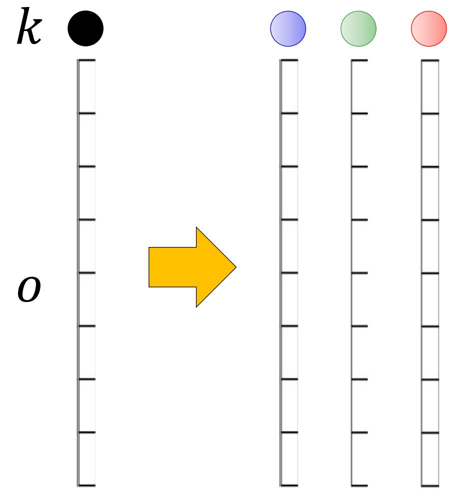


- *Every* Gaussian is capable of generating this vector
  – With different probabilities

- If we saw a large number of these vectors, how many of these would have come from each Gaussian

- All of them, but in proportion to $P(k|o)$

Completing incomplete vectors



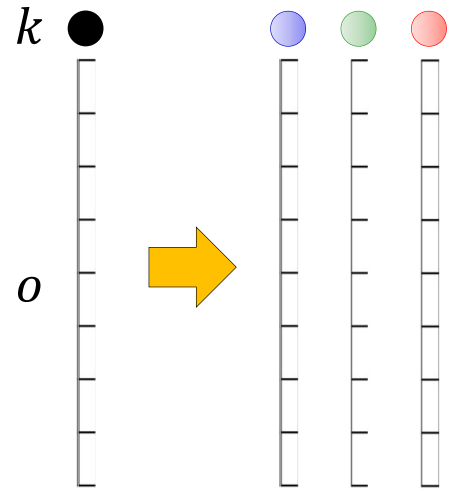
$in\ proportion\colon\ P(*\,|o)$

- Complete the data by attributing to *every Gaussian*
  – I.e. make many complete "clones" of the data

- But assign a *proportion* to each completed vector
  – Proportion is $P(k|o)$
    - Which can be computed if we know $P(k)$ and $P(o|k)$

- Then estimate the parameters using the complete data

Completing incomplete vectors

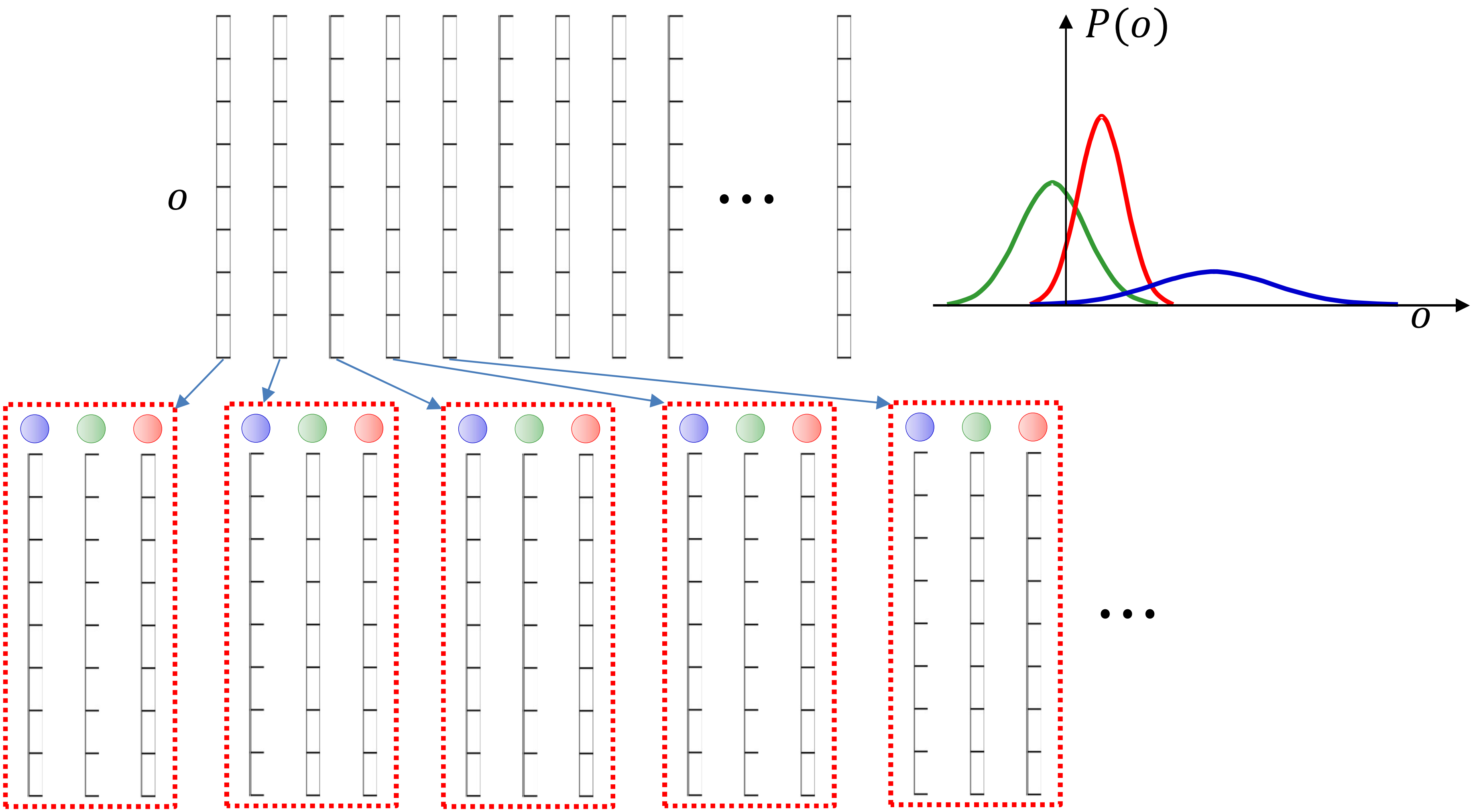


$in\ proportion:\ P(*\,|o)$

- Complete the data by attributing to *every Gaussian*
  - I.e. make many complete "clones" of the data
- But assign a *proportion* to each completed vector
  - Proportion is $P(k|o)$
    - Which can be computed if we know $P(k)$ and $P(o|k)$
- Then estimate the parameters using the complete data

From previous estimate of model

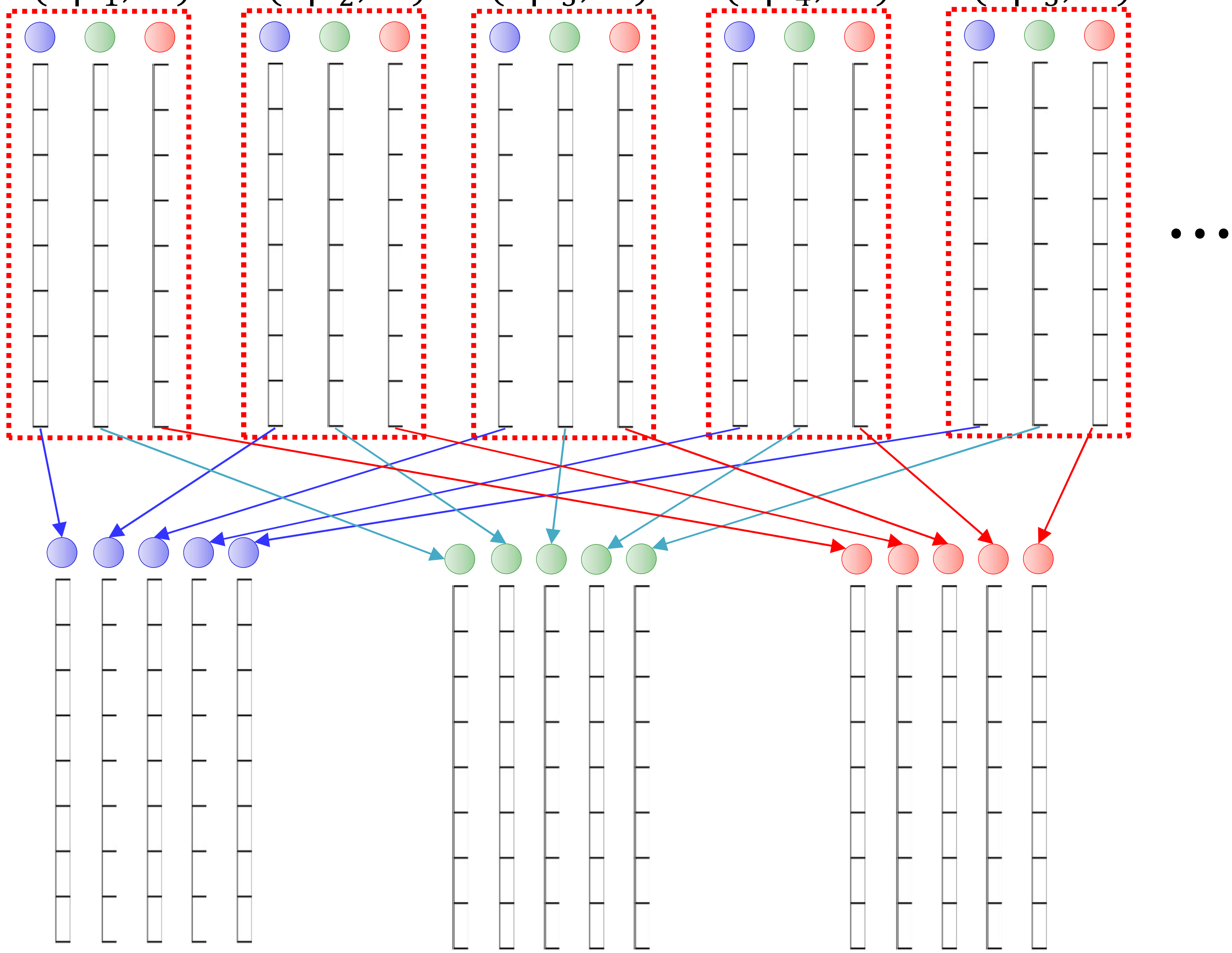
EM for GMMs



- “Complete” each vector in every possible way:
  - assign each vector to every Gaussian
  - In proportion $P(k|o;\theta^l)$ (computed from current model estimate)
- Compute statistics from “completed” data

EM for GMMs

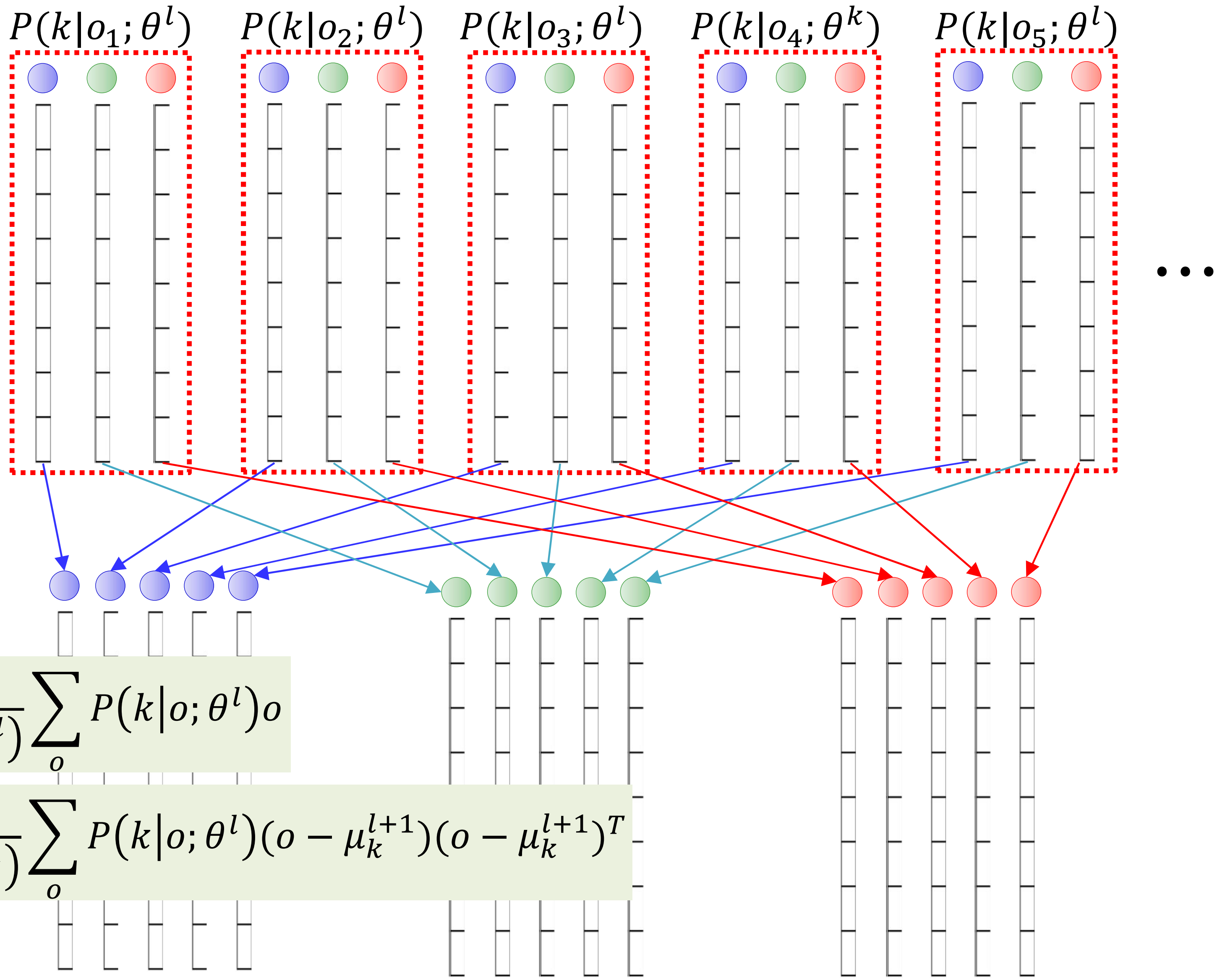
In proportion to $P(k|o_1;\theta^l)$ $P(k|o_2;\theta^l)$ $P(k|o_3;\theta^l)$ $P(k|o_4;\theta^k)$ $P(k|o_5;\theta^l)$



- *Now* you can segregate the vectors by Gaussian
  - The number of segregated complete vectors from each observation will be in proportion to $P(k|o;\theta^l)$

EM for GMMs

In proportion to

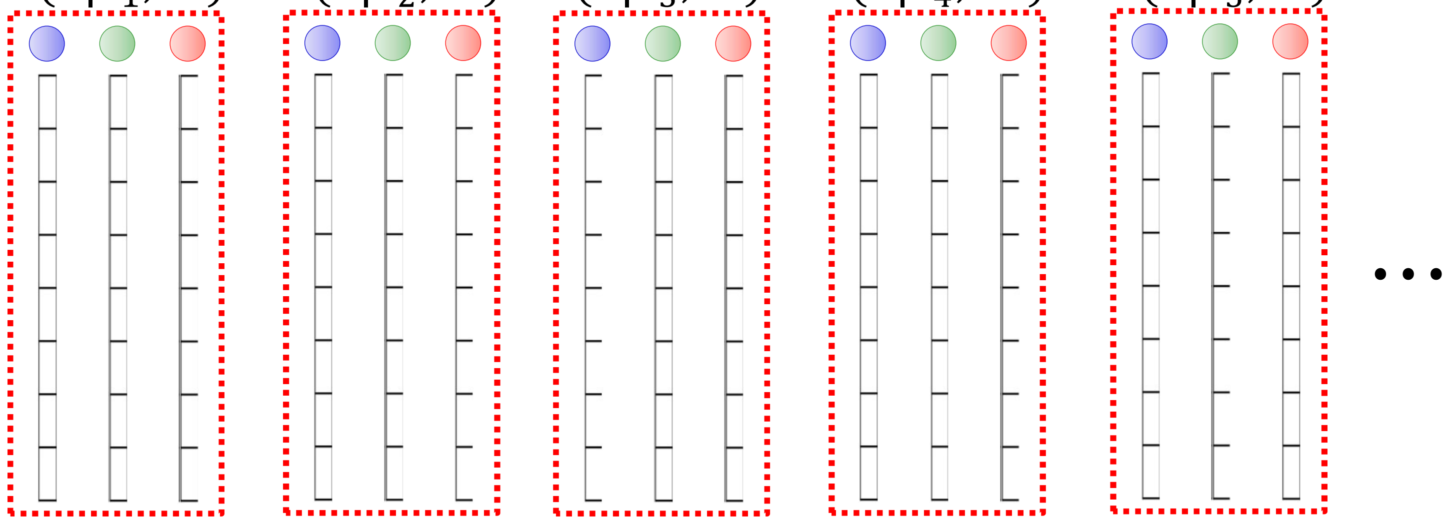


$$\mu_k^{l+1} = \frac{1}{\sum_o P(k|o;\theta^l)} \sum_o P(k|o;\theta^l) o$$

$$\Sigma_k^{l+1} = \frac{1}{\sum_o P(k|o;\theta^l)} \sum_o P(k|o;\theta^l)(o-\mu_k^{l+1})(o-\mu_k^{l+1})^T$$

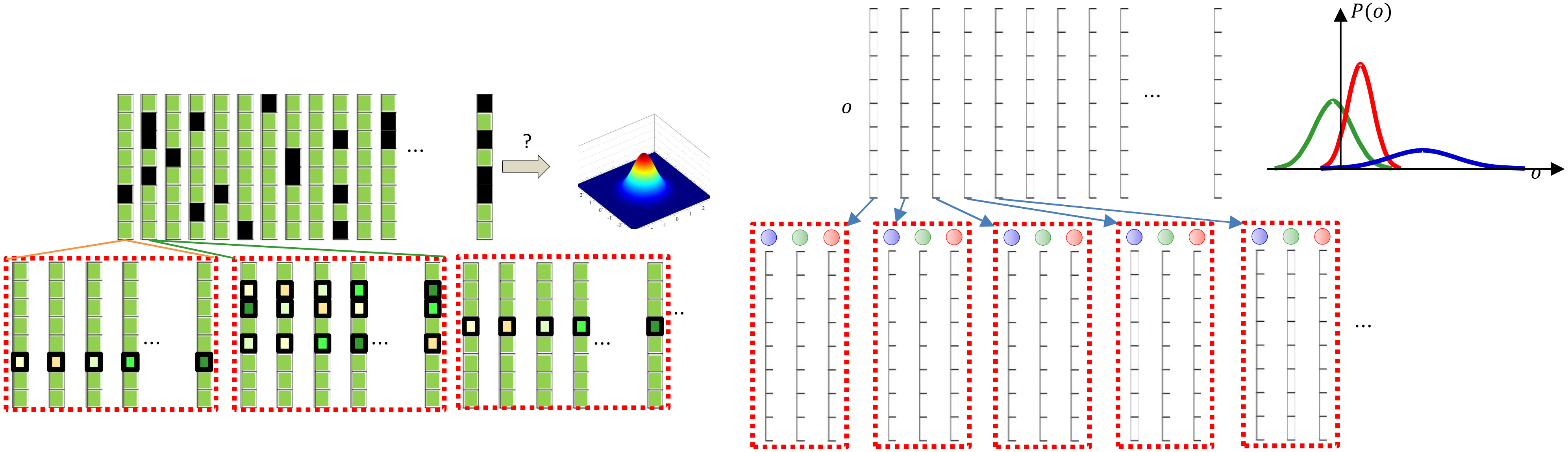
- *Now* you can segregate the vectors by Gaussian
  - The number of segregated complete vectors from each observation will be in proportion to $P(k|o;\theta^l)$

EM for GMMs



- Initialize μ_k^0 and Σ_k^0 for all k
- Iterate (over l):
  - Compute $P(k|o;\theta^l)$ for all o
    - Compute the proportions by which o is assigned to all Gaussians
  - Update:
  - $\mu_k^{l+1} = \frac{1}{\sum_o P\left(k\middle|o;\theta^l\right)} \sum_o P\left(k\middle|o;\theta^l\right) o$
  - $\Sigma_k^{l+1} = \frac{1}{\sum_o P\left(k\middle|o;\theta^l\right)} \sum_o P\left(k\middle|o;\theta^l\right)(o-\mu_k^{l+1})(o-\mu_k^{l+1})^T$

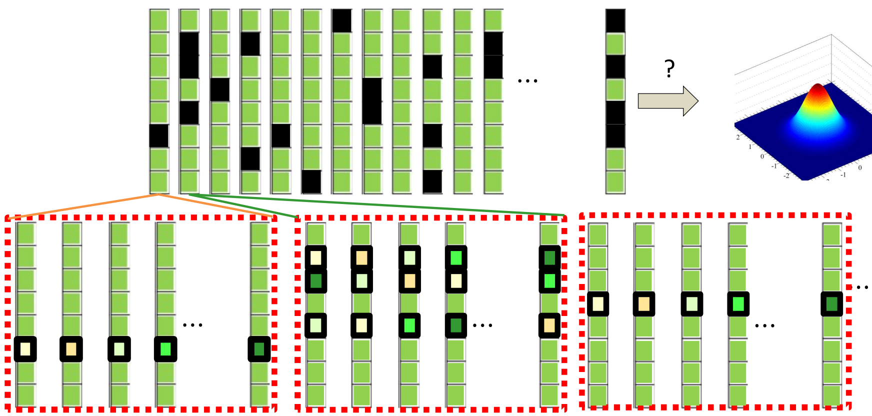
General EM principle

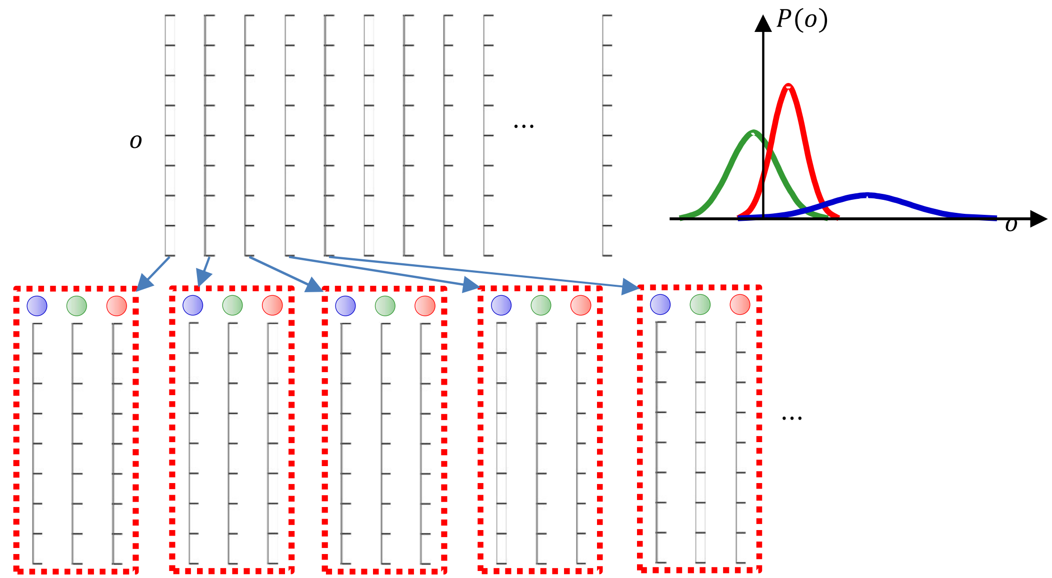


- "Complete" the data by considering *every* possible value for missing data/variables
  - In proportion to their posterior probability, given the observation, $P(m|o)$ (or $P(k|o)$)

- Reestimate parameters from the "completed" data

General EM principle

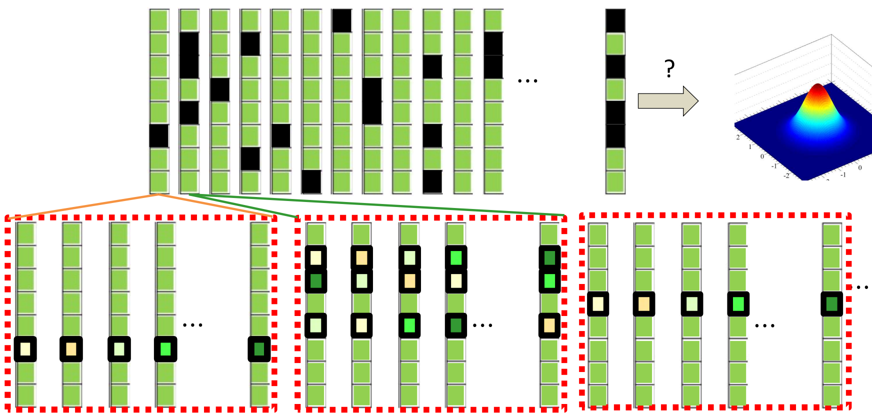


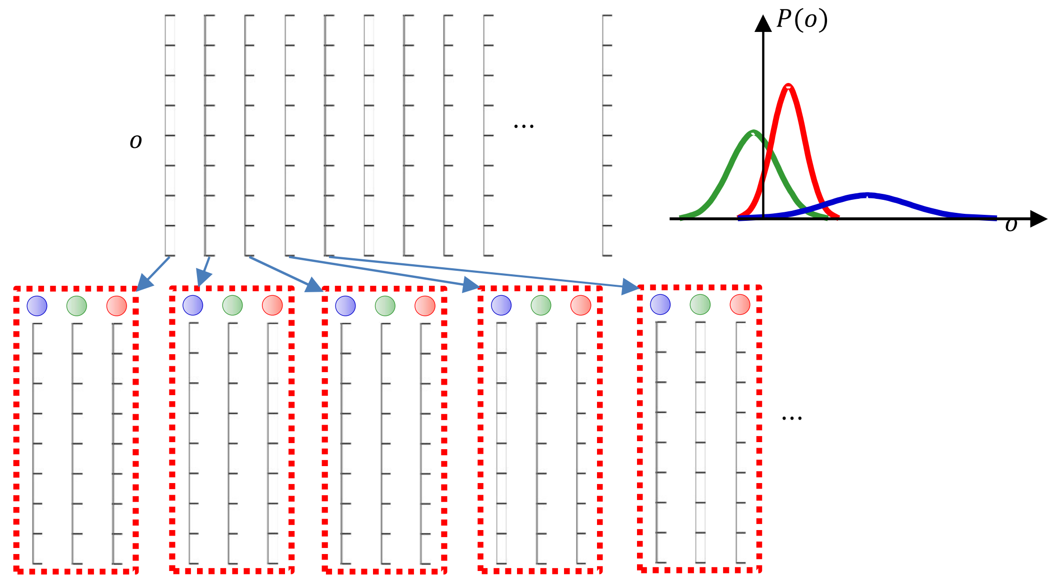


- "Complete" the data by considering *every* possible value for missing data/variables
  - In proportion to their posterior probability, given the observation, $P(m|o)$ (or $P(k|o)$)

- Reestimate parameters from the "completed" data

General EM principle

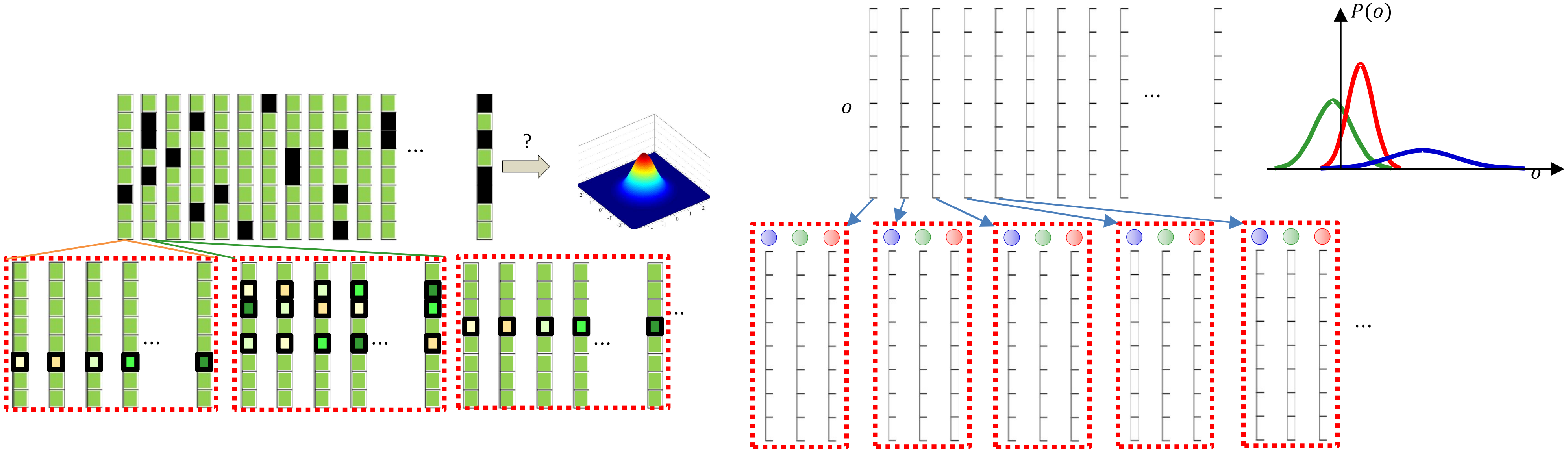




- "Complete" the data by considering *every* possible value for missing data/variables
  - In proportion to their posterior probability, given the observation, $P(m|o)$ (or $P(k|o)$)

Sufficient to "complete" the data by *sampling* missing values from the posterior $P(m|o)$ (or $P(k|o)$) instead

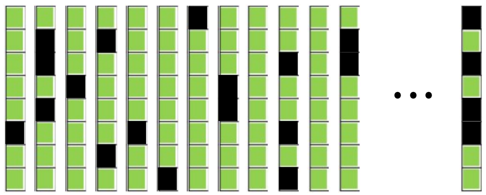
Alternate EM principle

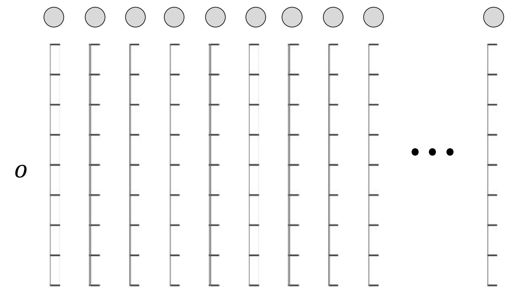


- "Complete" the data by *sampling* possible value for missing data/variables from $P(m|o)$ (or $P(k|o)$)

- Reestimate parameters from the "completed" data

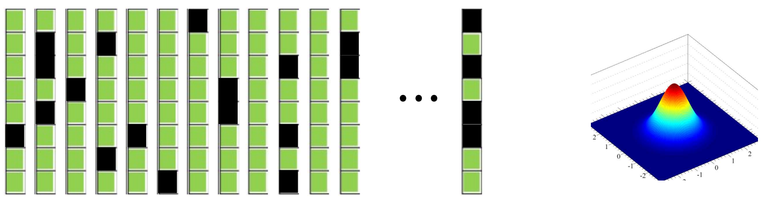
Overall EM principle: Remember this

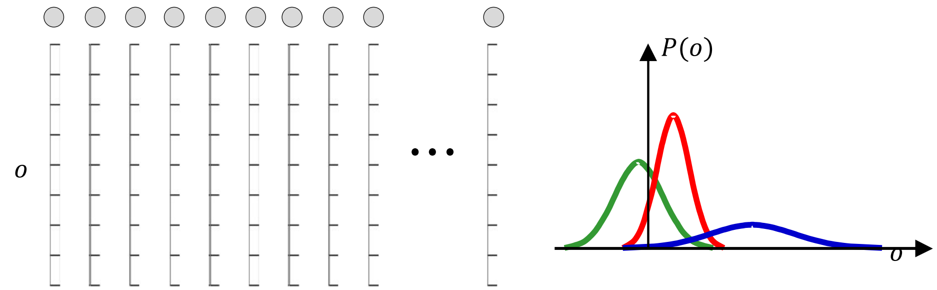




- Initially, some data/information are missing

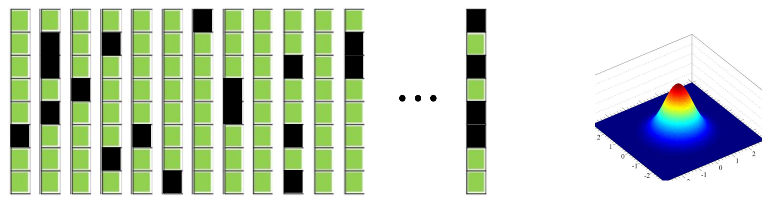
Overall EM principle: Remember this

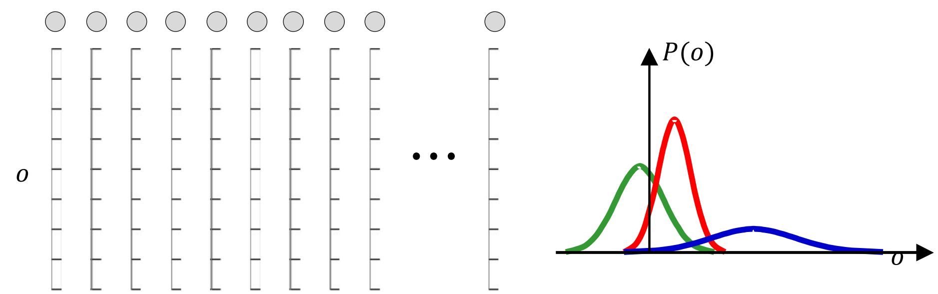




- Initially, some data/information are missing
- ***Initialize model parameters***

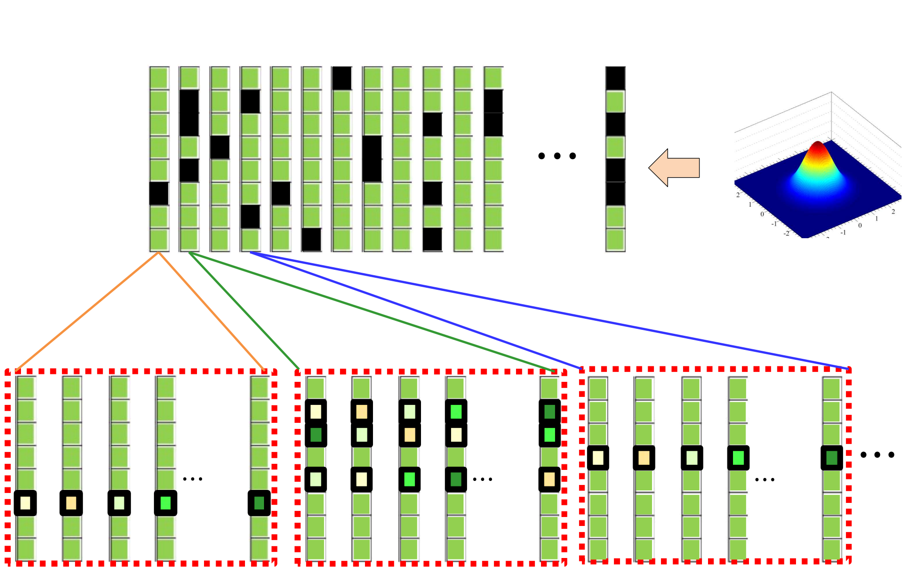
Overall EM principle: Remember this

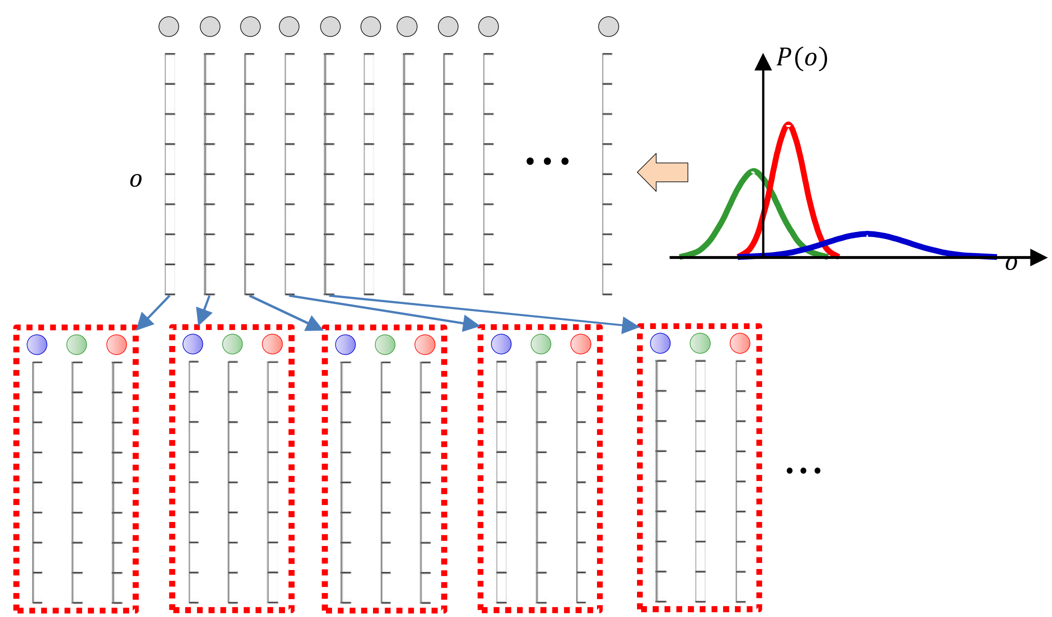




- Initially, some data/information are missing
- Initialize model parameters
- **Iterate:**

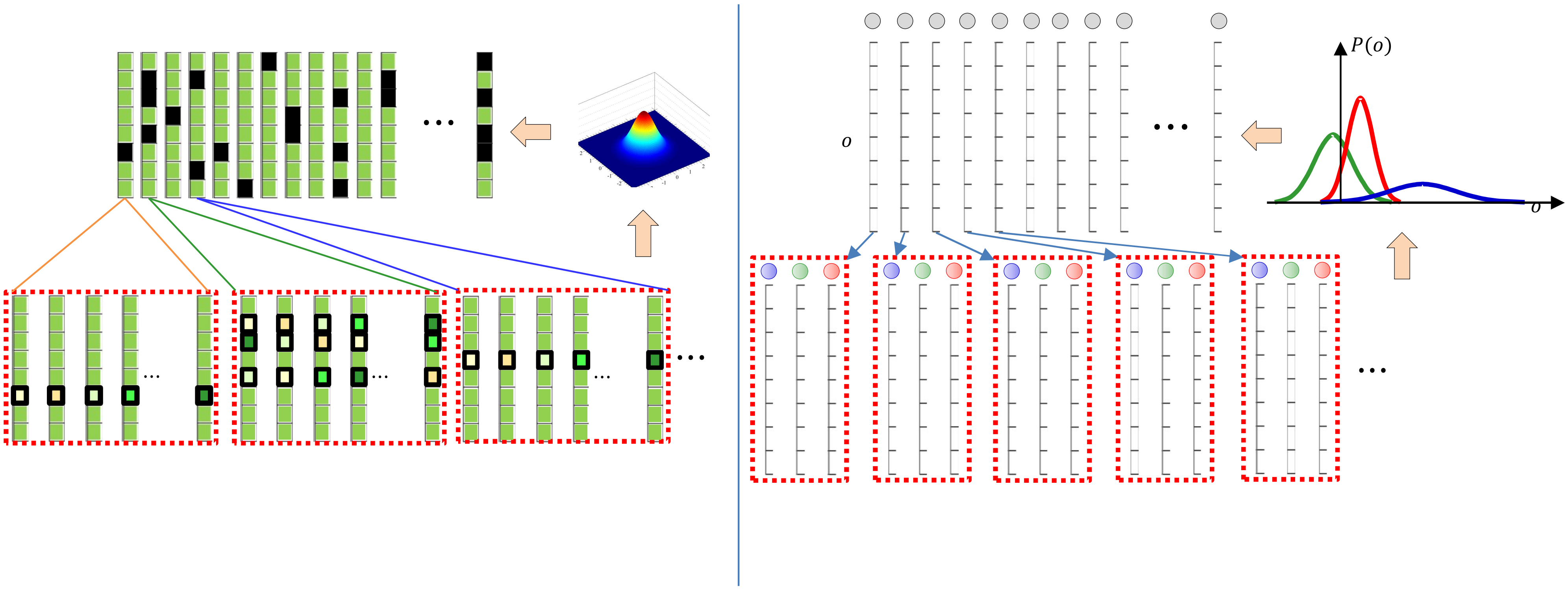
Overall EM principle: Remember this





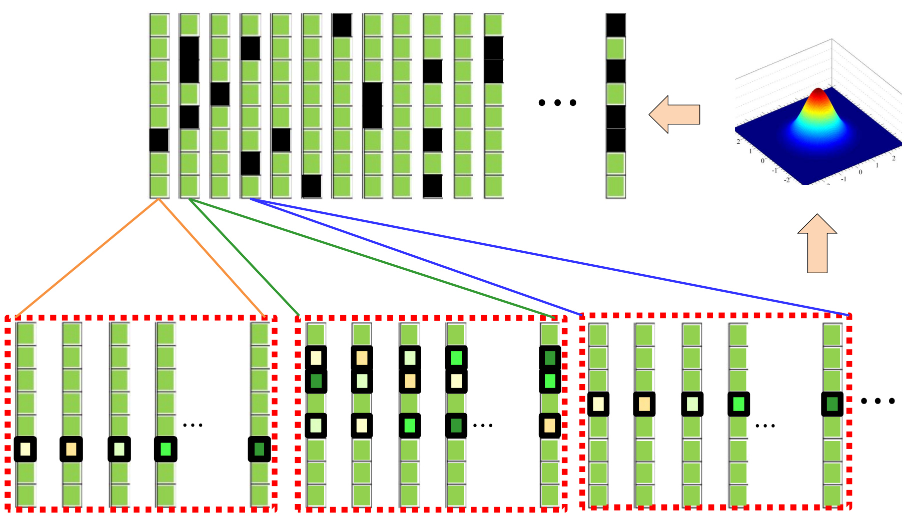
- Initially, some data/information are missing
- Initialize model parameters
- Iterate
  - **Complete the data according to the posterior probabilities $P(m|o)$ computed by the current model**
    - By implicitly considering every possible value, with its posterior-based proportionality
    - Or by explicit completion through sampling the posterior probability distribution $P(m|o)$

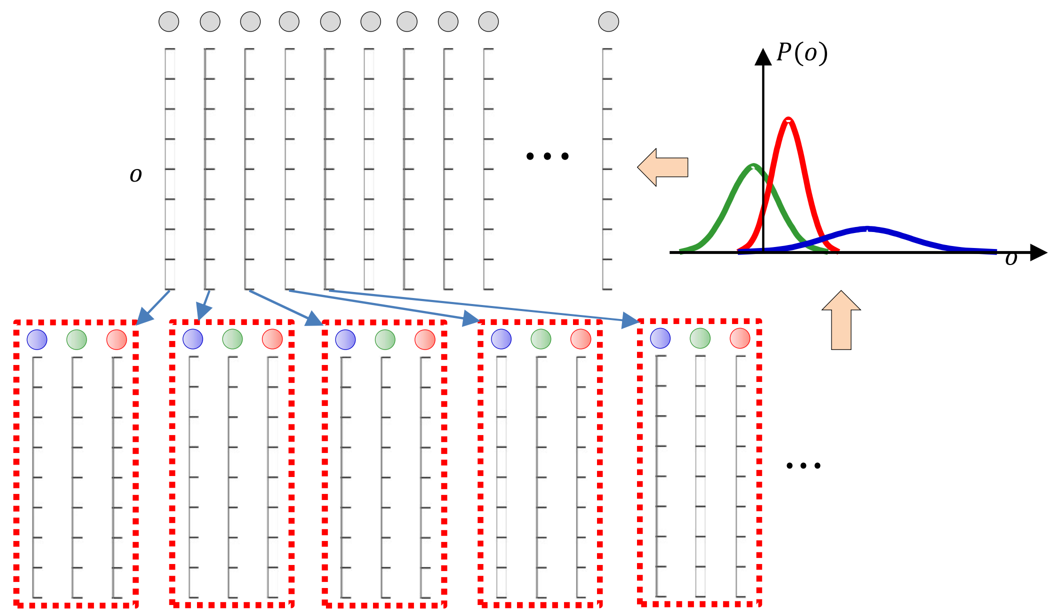
Overall EM principle: Remember this



- Initially, some data/information are missing
- Initialize model parameters
- Iterate
  - Complete the data according to the posterior probabilities $P(m|o)$ computed by the current model
    - By considering every possible value, with its posterior-based proportionality
    - Or by sampling the posterior probability distribution $P(m|o)$
  - **Reestimate the model**

Overall EM principle: Remember this





- Initially, some data/information are missing
- Initialize model parameters
- Iterate
  - Complete the data according to the posterior probabilities $P(m|o)$ computed by the current model
    - By implicitly considering every possible value, with its posterior-based proportionality
    - Or by explicit completion through sampling the posterior probability distribution $P(m|o)$
  - Reestimate the model

Poll 3 (@1763)

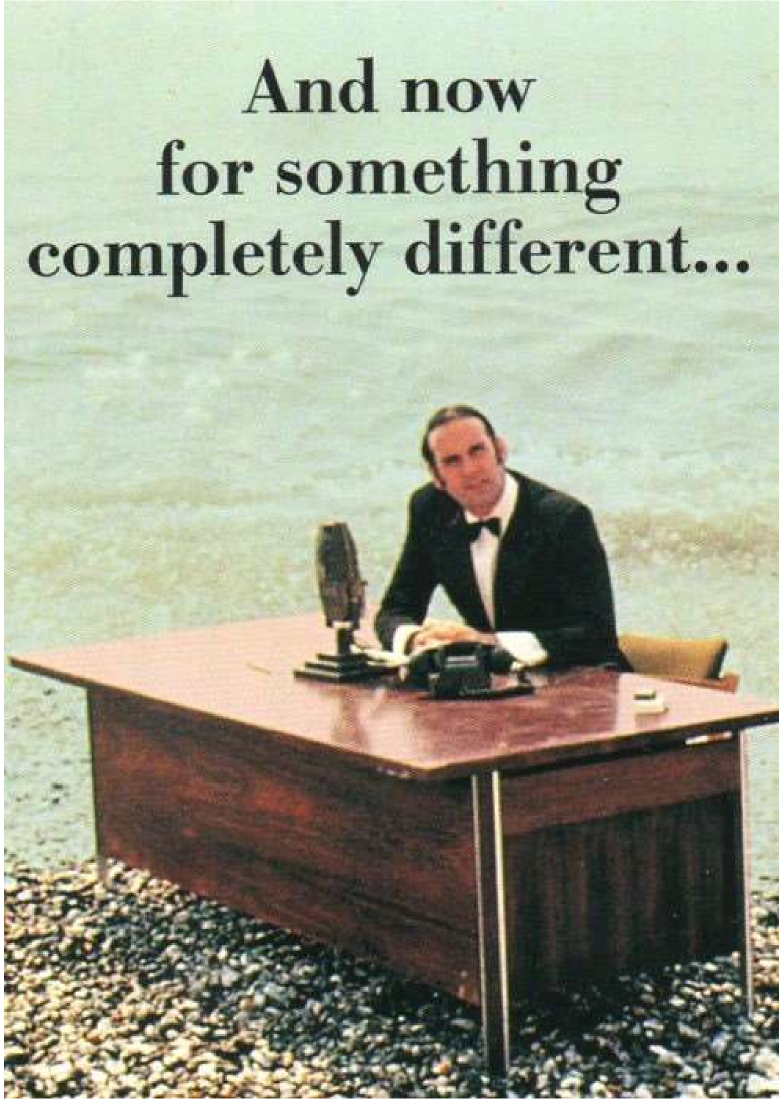
Select all that are true of EM estimation

- In each iteration we “complete” the data, by filling in the missing components/variables, and estimate parameters from the entire completed data
- A data instance can be completed by filling in the missing terms with every possible value, in proportion to their a-posteriori probability, given the observed components of the data
- A data instance can be completed by randomly drawing samples of the missing components from their a-posteriori probability distribution, given the observed components of the data
- “Data completion” must be performed only once during the entire training (with EM)

Poll 3

Select all that are true of EM estimation

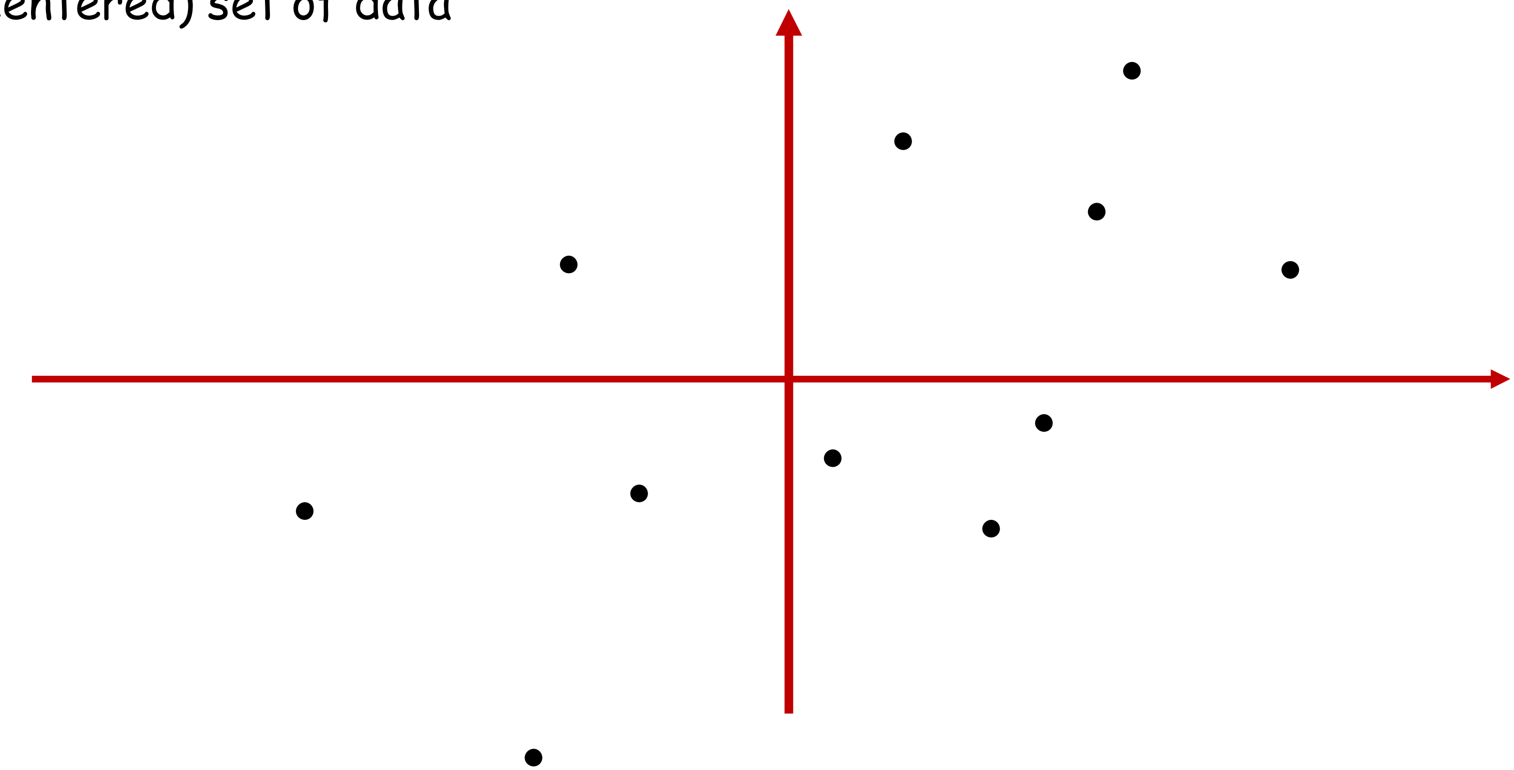
- **In each iteration we "complete" the data, by filling in the missing components/variables, and estimate parameters from the entire completed data**
- **A data instance can be completed by filling in the missing terms with every possible value, in proportion to their a-posteriori probability, given the observed components of the data**
- **A data instance can be completed by randomly drawing samples of the missing components from their a-posteriori probability distribution, given the observed components of the data**
- "Data completion" must be performed only once during the entire training (with EM)



Principal Component Analysis

Principal Component Analysis

Given a (centered) set of data

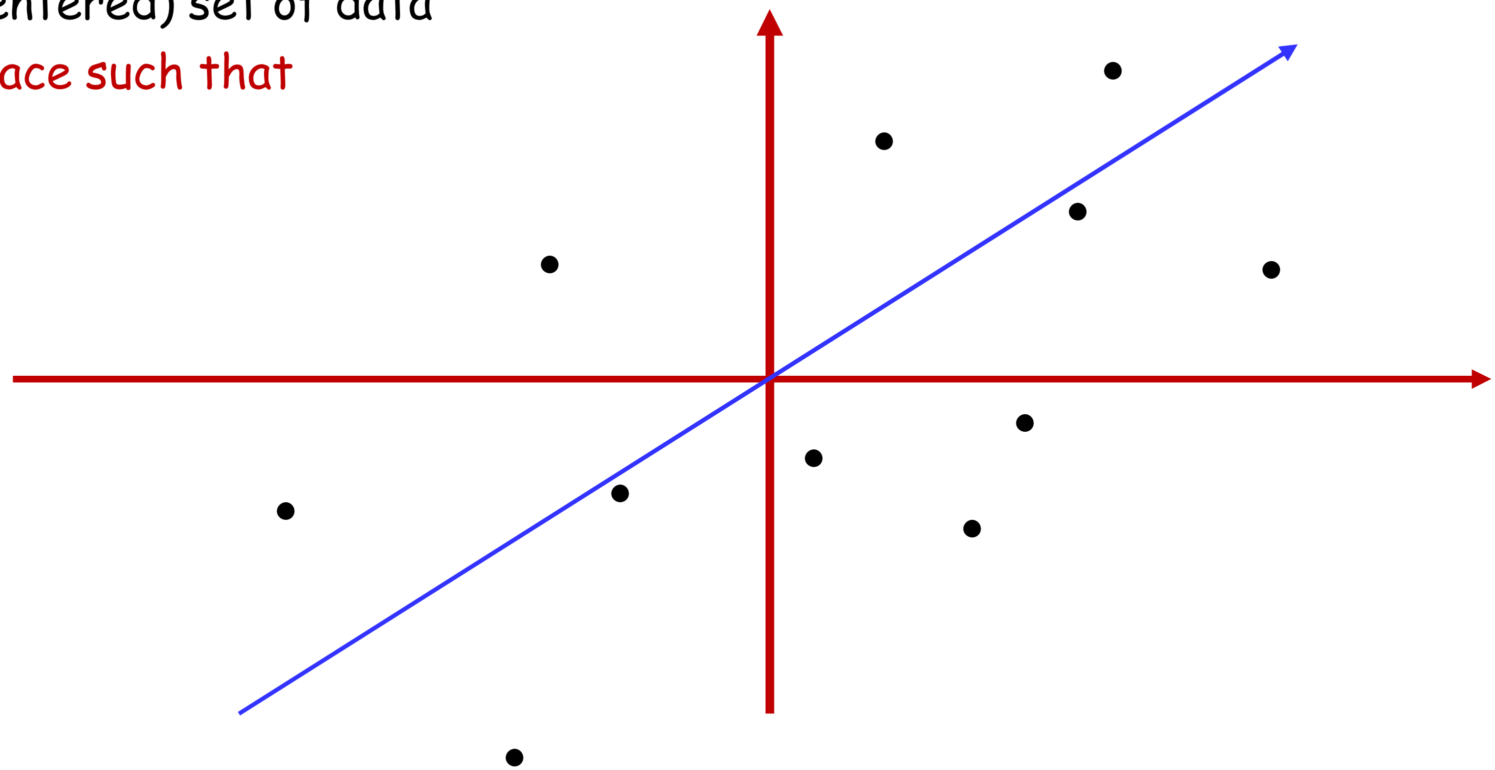


- Find the principal subspace such that when all vectors are approximated as lying on that subspace, the approximation error is minimal
  - Assuming "centered" (zero-mean) data

Principal Component Analysis

Given a (centered) set of data

find subspace such that

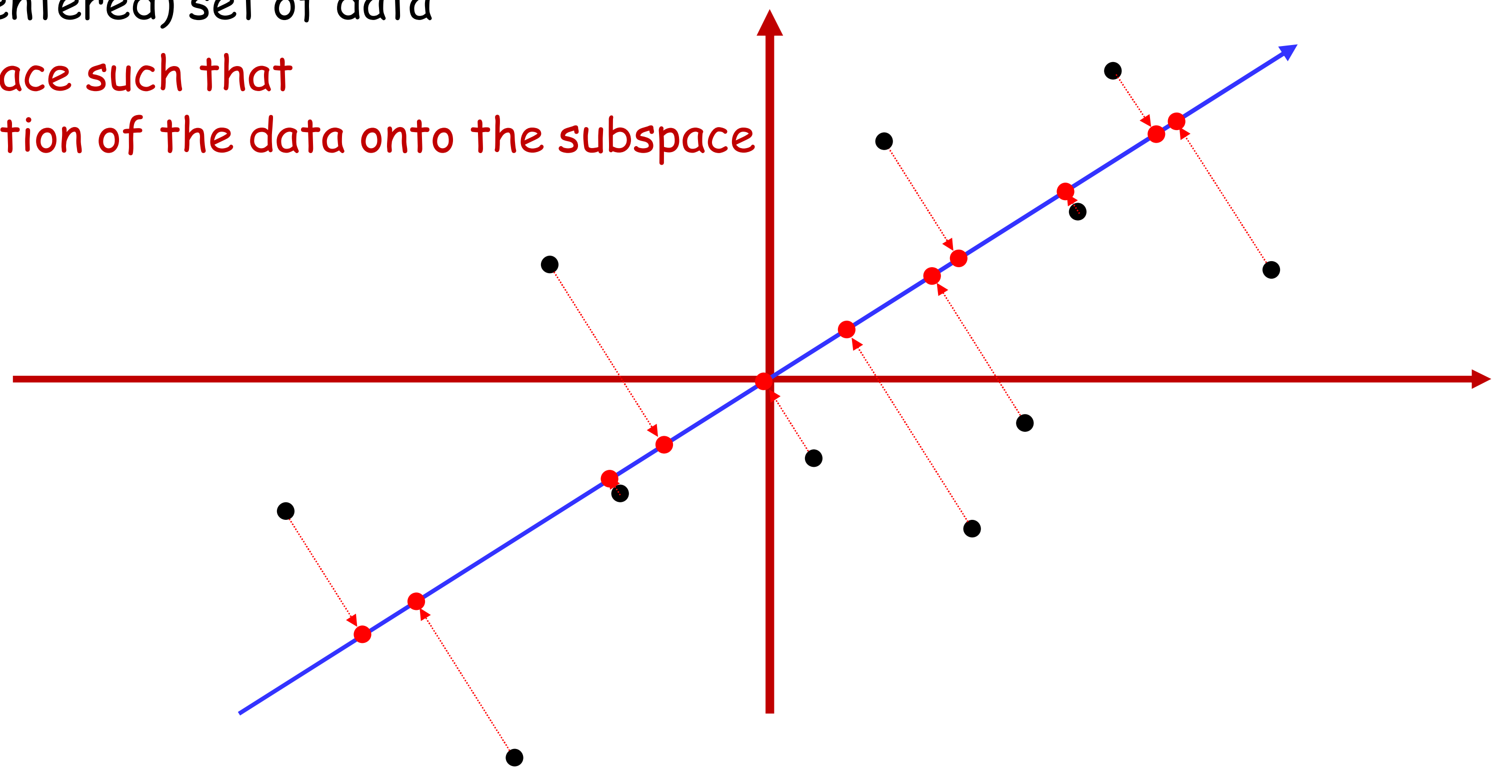


- Find the principal subspace such that when all vectors are approximated as lying on that subspace, the approximation error is minimal
  - Assuming "centered" (zero-mean) data

Principal Component Analysis

Given a (centered) set of data

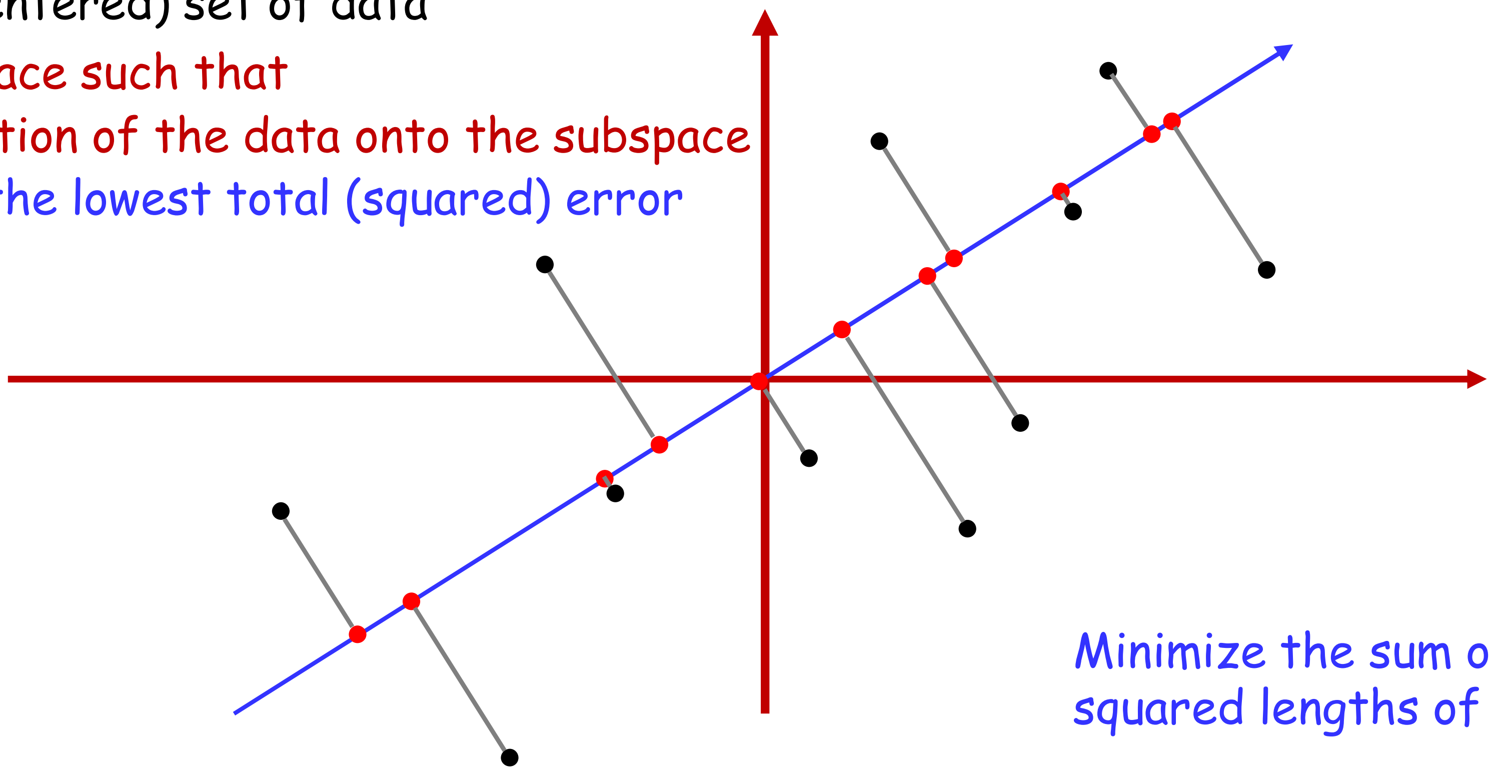
find subspace such that
the projection of the data onto the subspace



- Find the principal subspace such that when all vectors are approximated as lying on that subspace, the approximation error is minimal
  - Assuming "centered" (zero-mean) data

Principal Component Analysis

Given a (centered) set of data
find subspace such that
the projection of the data onto the subspace
results in the lowest total (squared) error

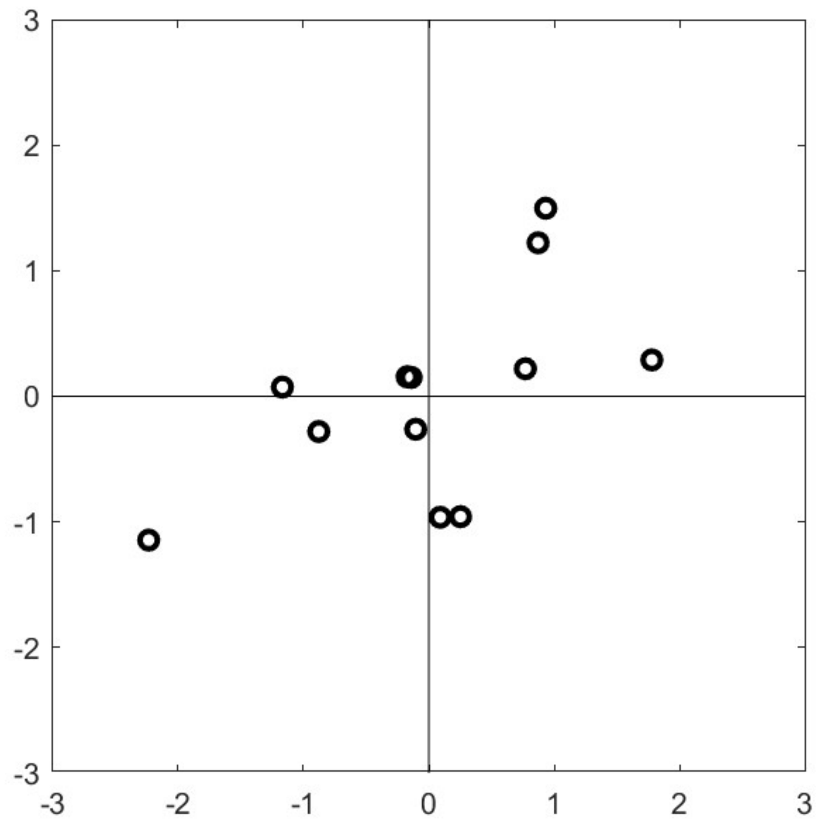


Minimize the sum of the
squared lengths of these lines

- Find the principal subspace such that when all vectors are approximated as lying on that subspace, the approximation error is minimal
  - Assuming "centered" (zero-mean) data

Principal Component Analysis

Animation:
Original centered data

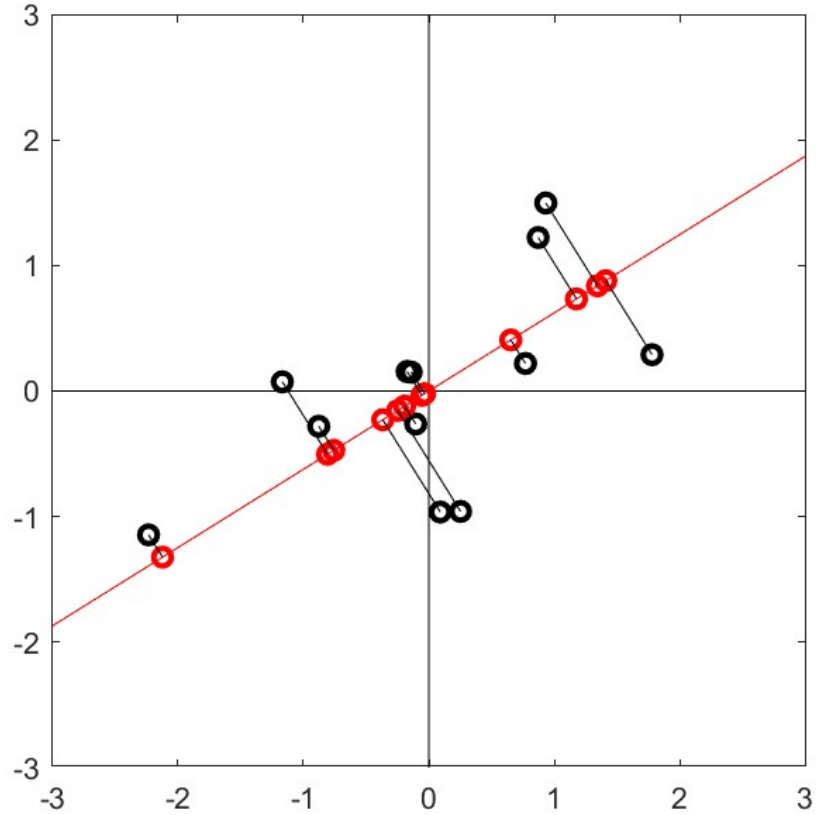


- Find the principal subspace such that when all vectors are approximated as lying on that subspace, the approximation error is minimal
  - Assuming "centered" (zero-mean) data

Principal Component Analysis

Animation:
Original centered data

Principal axis we're searching for

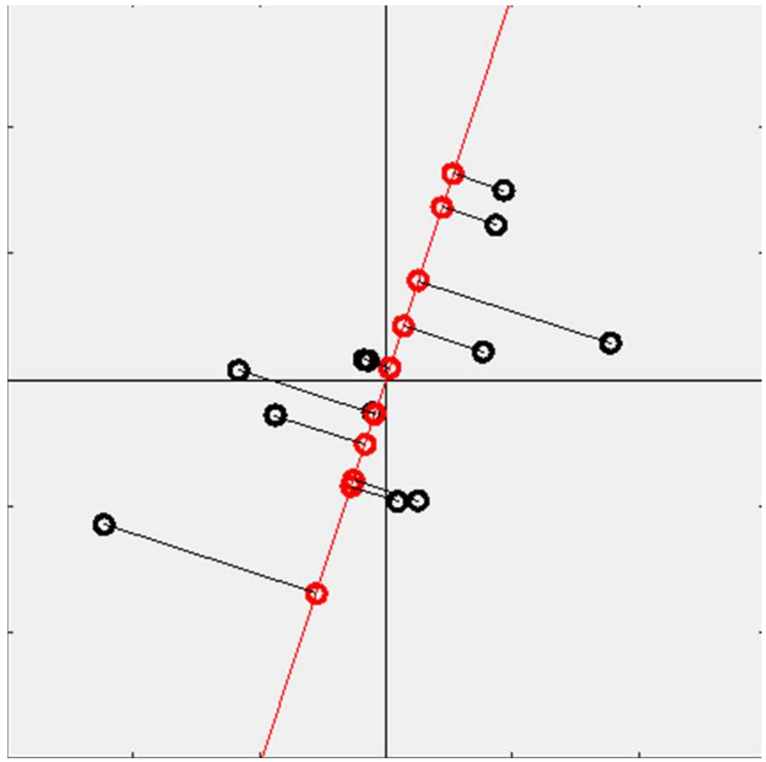


- Find the principal subspace such that when all vectors are approximated as lying on that subspace, the approximation error is minimal
  - Assuming "centered" (zero-mean) data

Principal Component Analysis

Animation:
Original centered data

Principal axis we're searching for

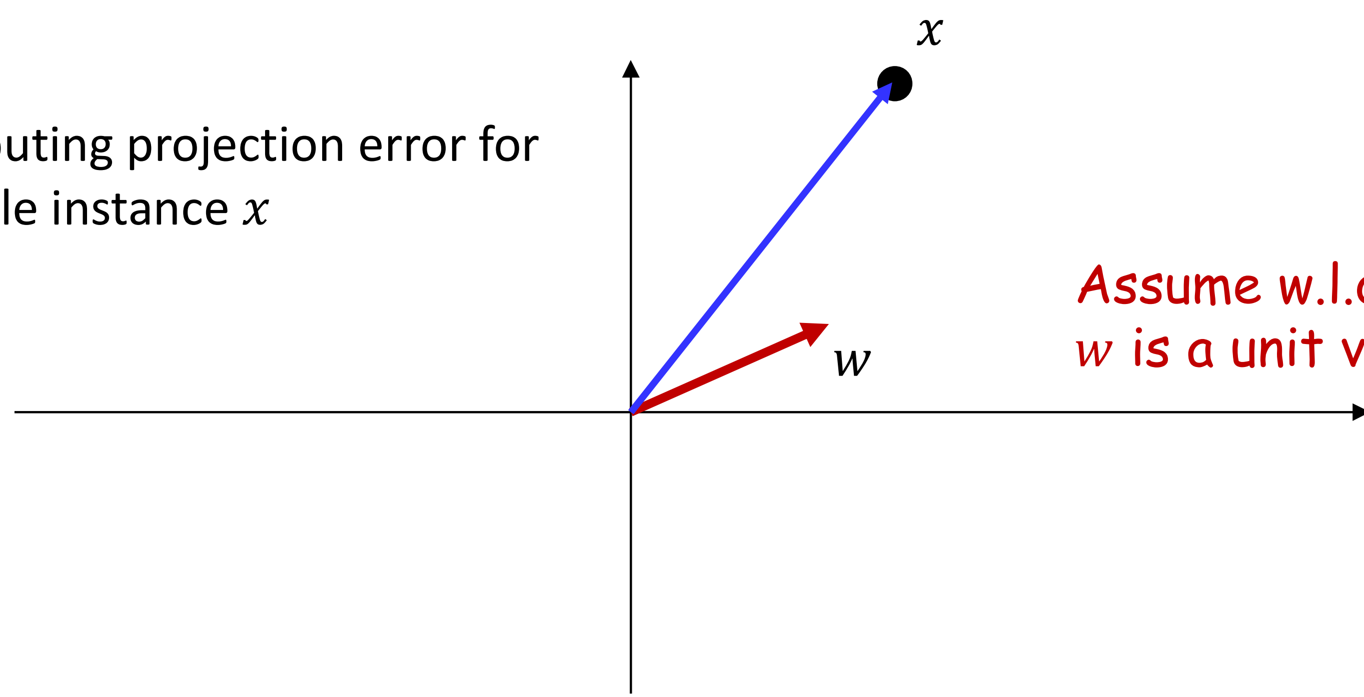


Search through all subspaces to find the one with minimum projection error

- Find the principal subspace such that when all vectors are approximated as lying on that subspace, the approximation error is minimal
  - Assuming "centered" (zero-mean) data

Can be done in closed form

Computing projection error for a single instance x

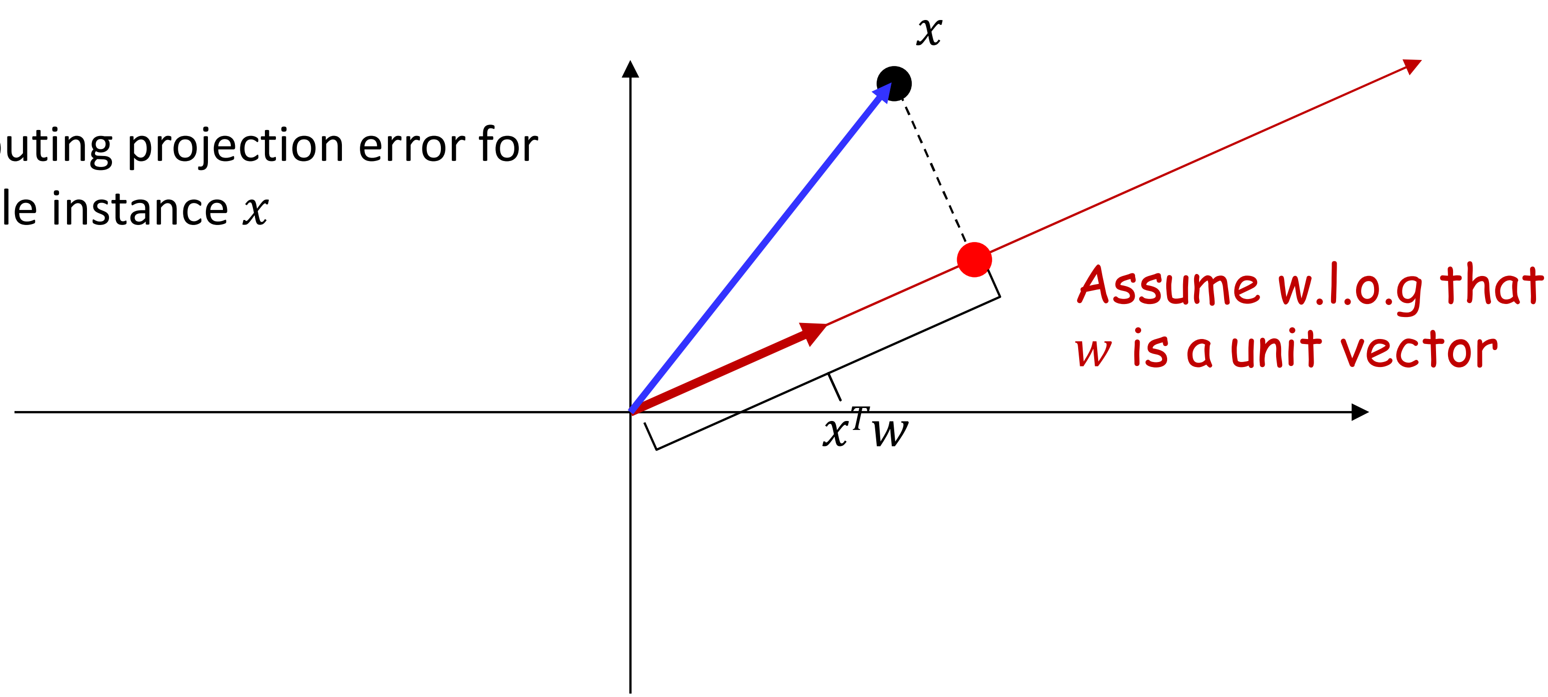


Assume w.l.o.g that w is a unit vector

- Since we're minimizing quadratic L_2 error, we can find a closed form solution

Can be done in closed form

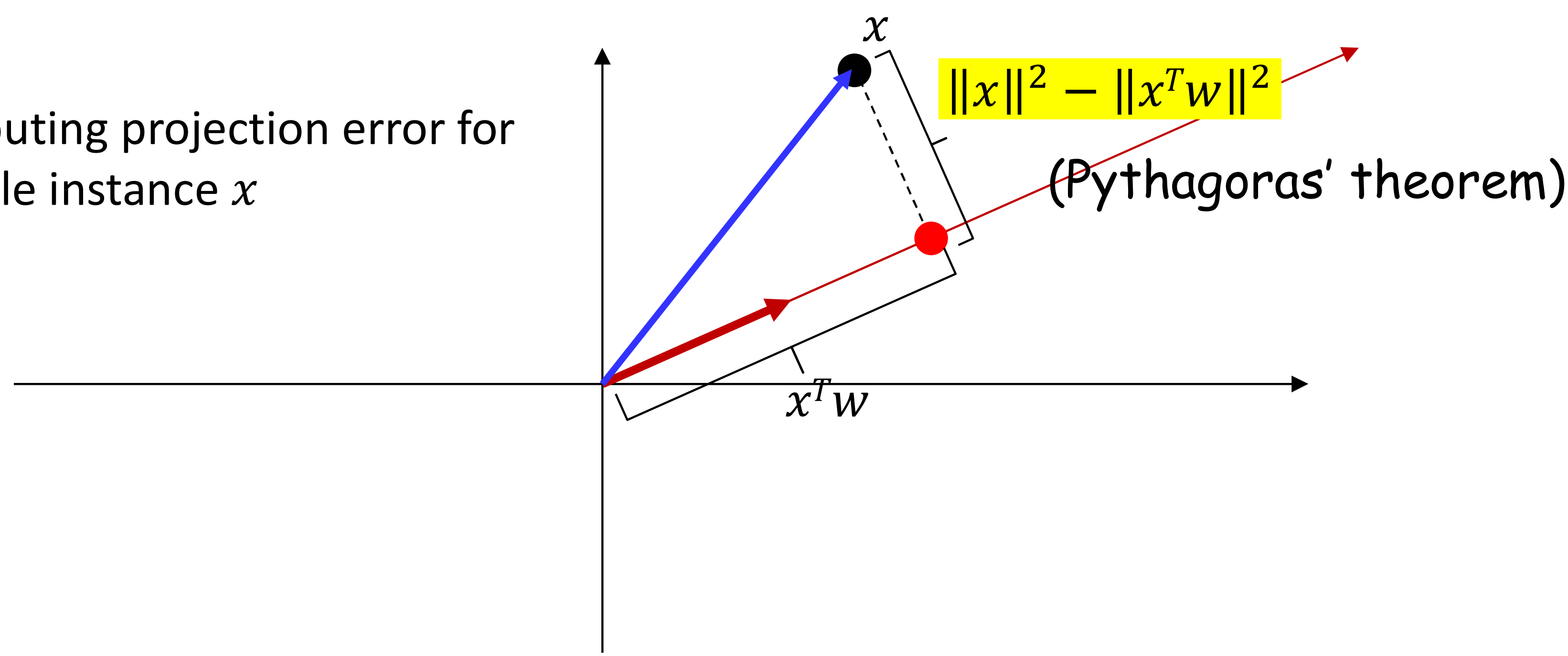
Computing projection error for a single instance x



- Since we're minimizing quadratic L_2 error, we can find a closed form solution

Can be done in closed form

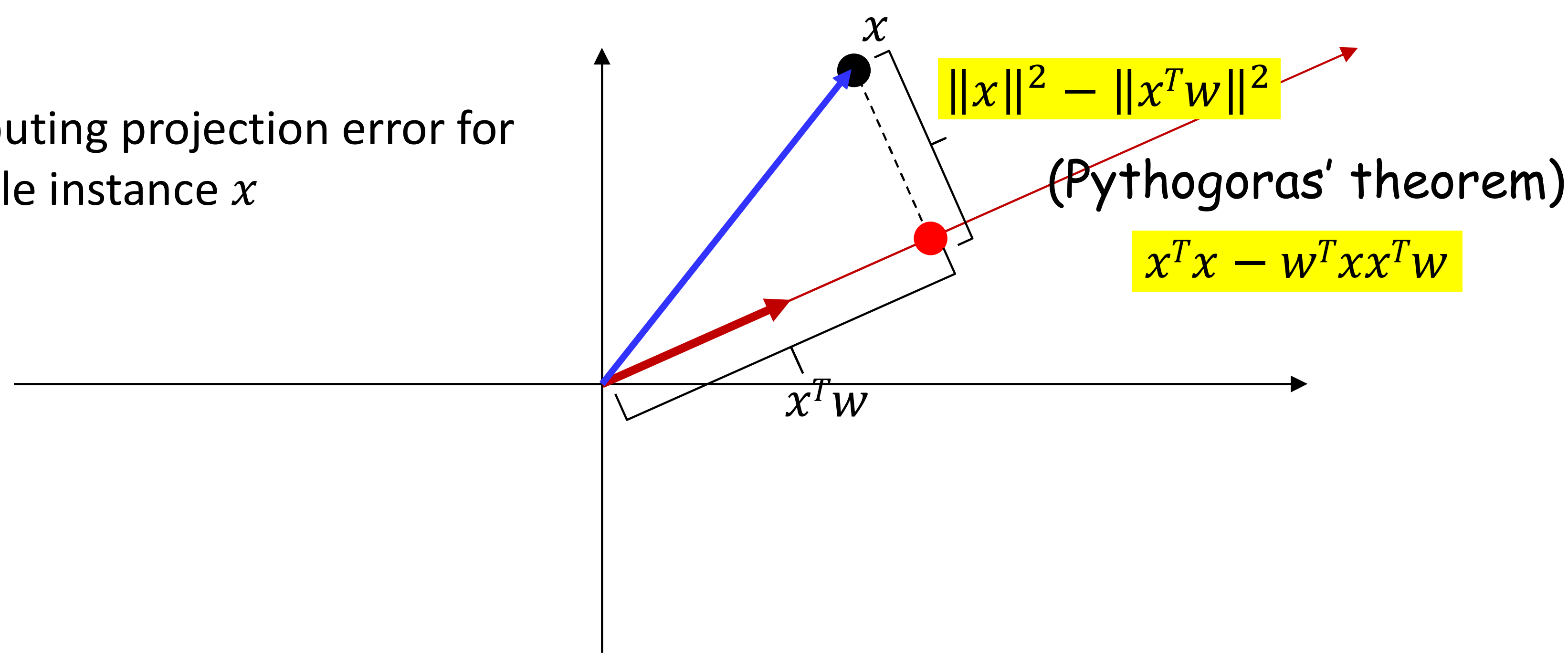
Computing projection error for a single instance x



- Since we're minimizing quadratic L_2 error, we can find a closed form solution

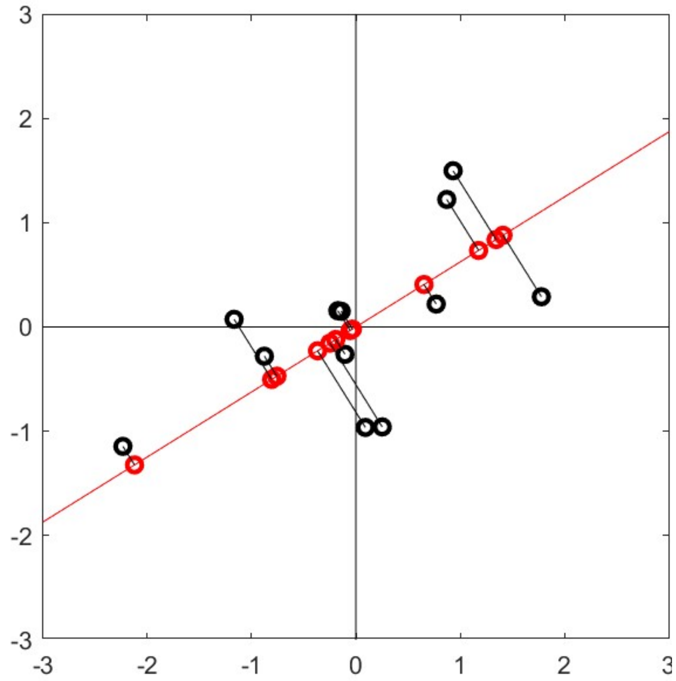
Can be done in closed form

Computing projection error for a single instance x



- Since we're minimizing quadratic L_2 error, we can find a closed form solution

Can be done in closed form



- Since we’re minimizing quadratic L_2 error, we can find a closed form solution
- Total projection error for all data:

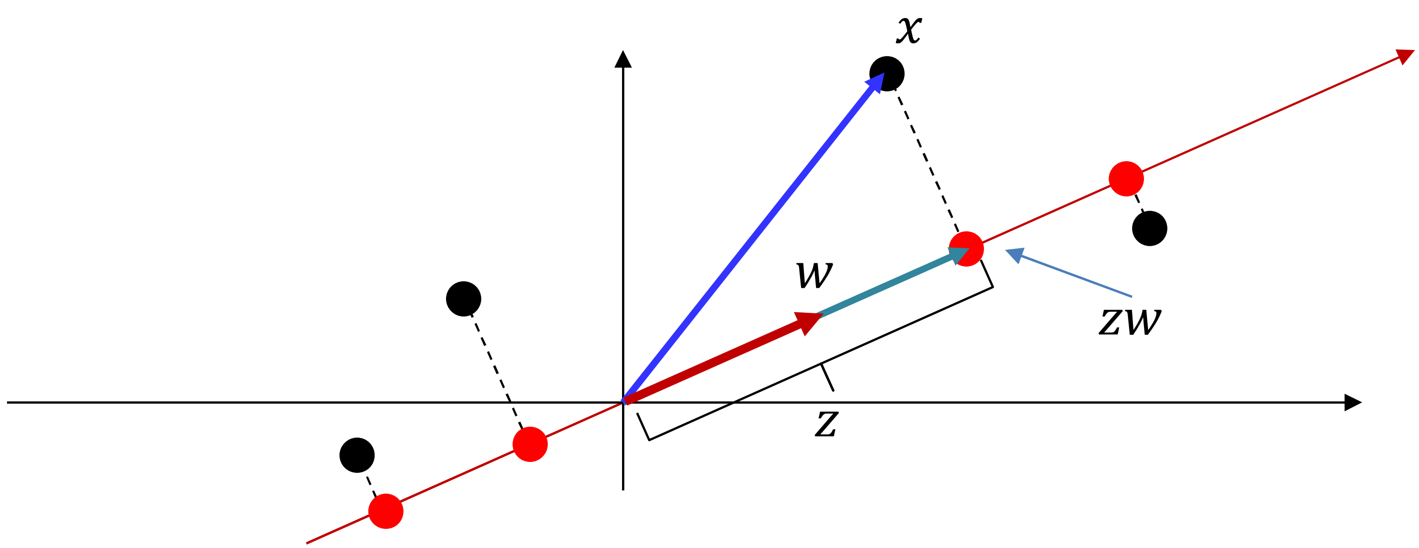
$$L = \sum_x x^T x - wTxx^T w$$

- Minimizing this w.r.t w (subject to w = unit vector) gives you the Eigenvalue equation

$$\left(\sum_x x^T x\right) w = \lambda w$$

- This can be solved to find the principal subspace

There's also an iterative solution

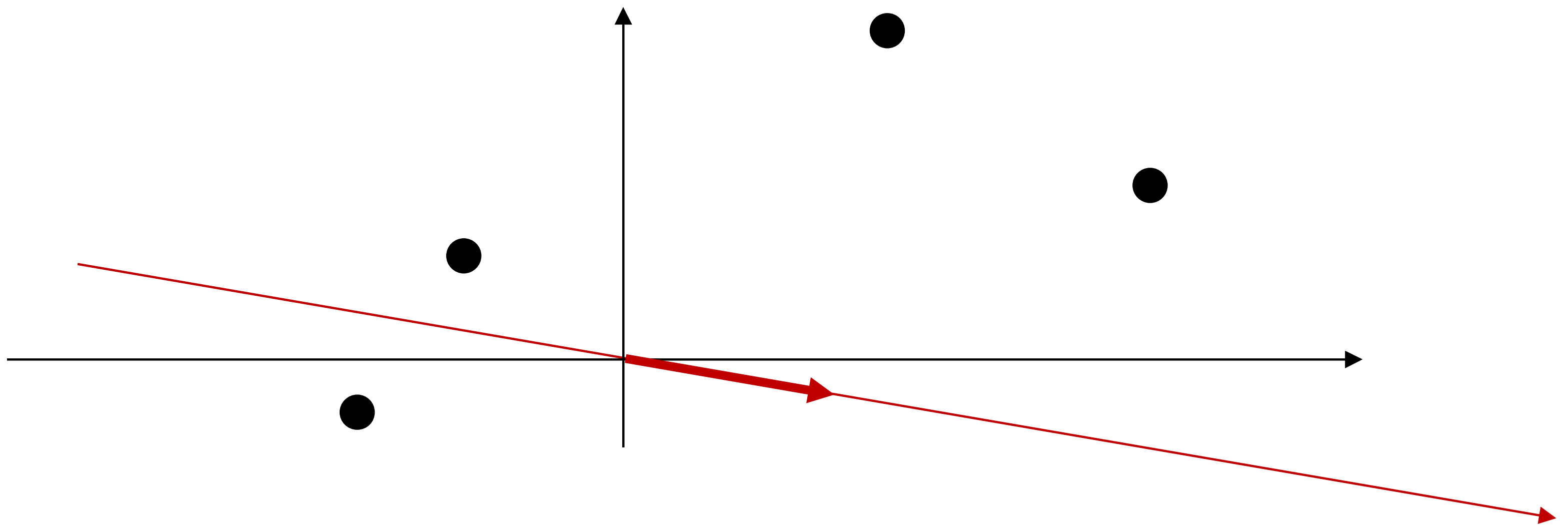


- Objective: find a vector (subspace) w and a *position* z on w such that $zw \approx x$ most closely (in an L_2 sense) for the entire (training) data

- Let $X = [x_1 x_2 \dots x_N]$ be the entire training set (arranged as a matrix)
  - Objective: find vector bases (for the subspace) W and the set of *position vectors* $Z = [z_1 z_2 \dots z_N]$ for all vectors in X such that $WZ \approx X$

- Initialize W
- Iterate until convergence:
  - Given W, find the best position vectors Z: $Z \leftarrow W^+ X$
  - Given position vectors Z, find the best subspace: $W \leftarrow XZ^+$
  - Guaranteed to find the principal subspace

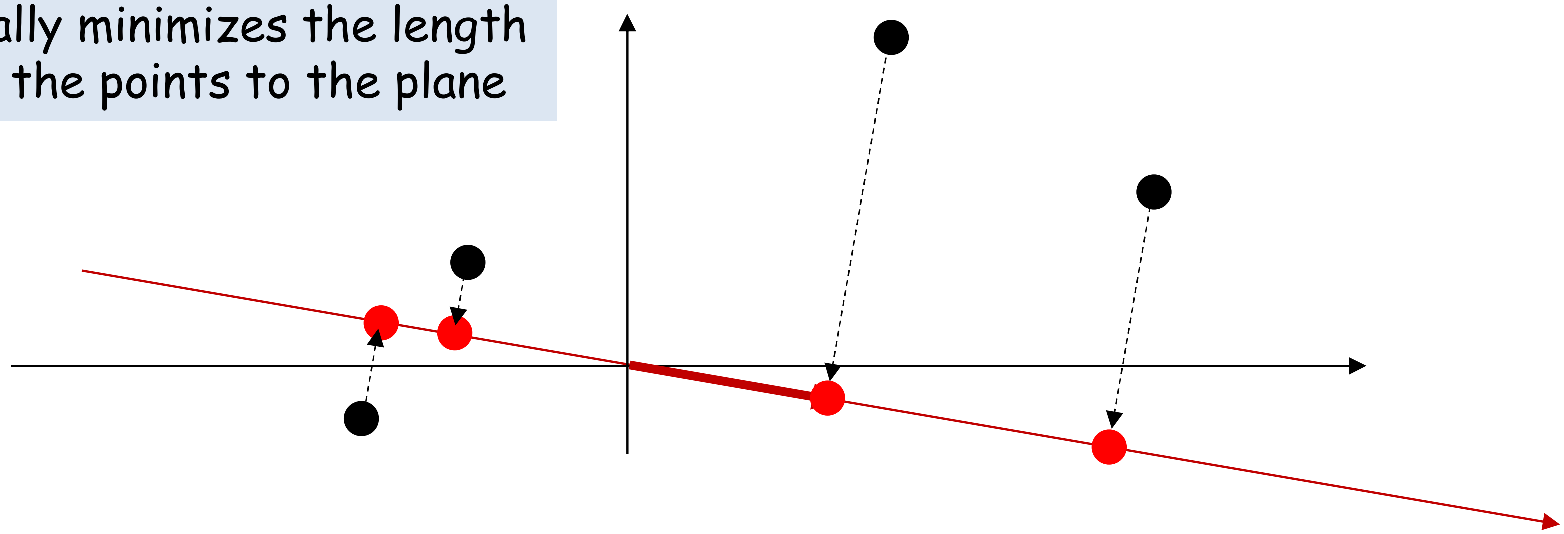
The iterative algorithm



- Initialize a subspace (the basis w)

The iterative algorithm

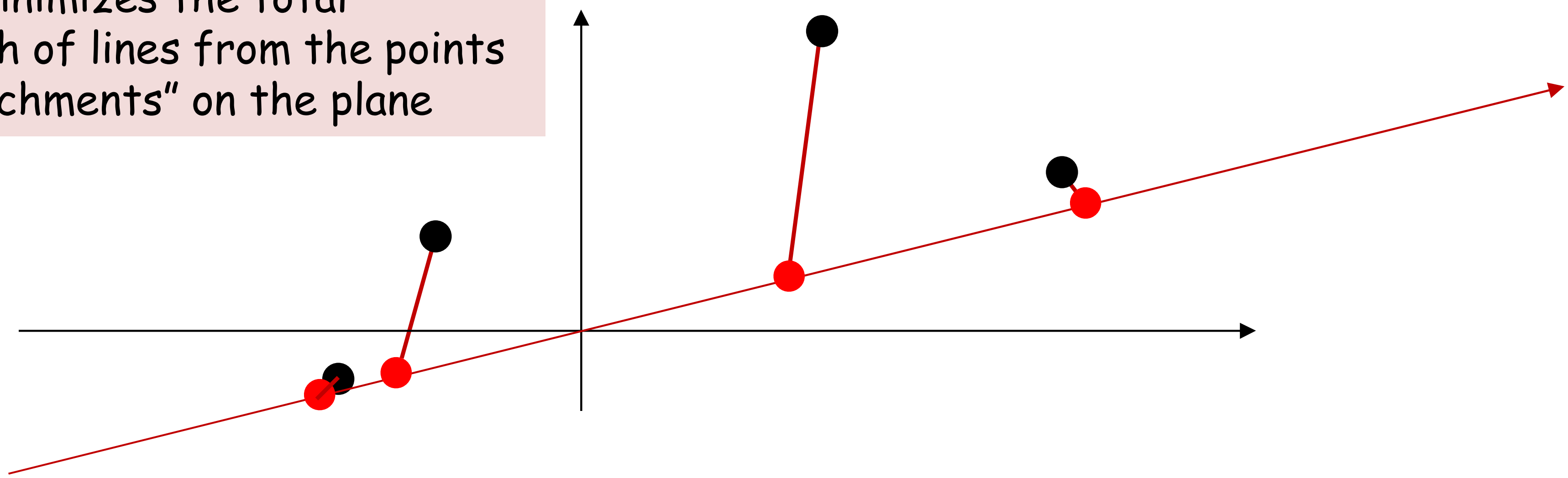
This individually minimizes the length of lines from the points to the plane



- Initialize a subspace (the basis w)
- Iterate until convergence:
  - Find the best position vectors Z on the W subspace for each training instance
    - Find the location on W that is *closest* to each instance, i.e. the perpendicular projection

The iterative algorithm

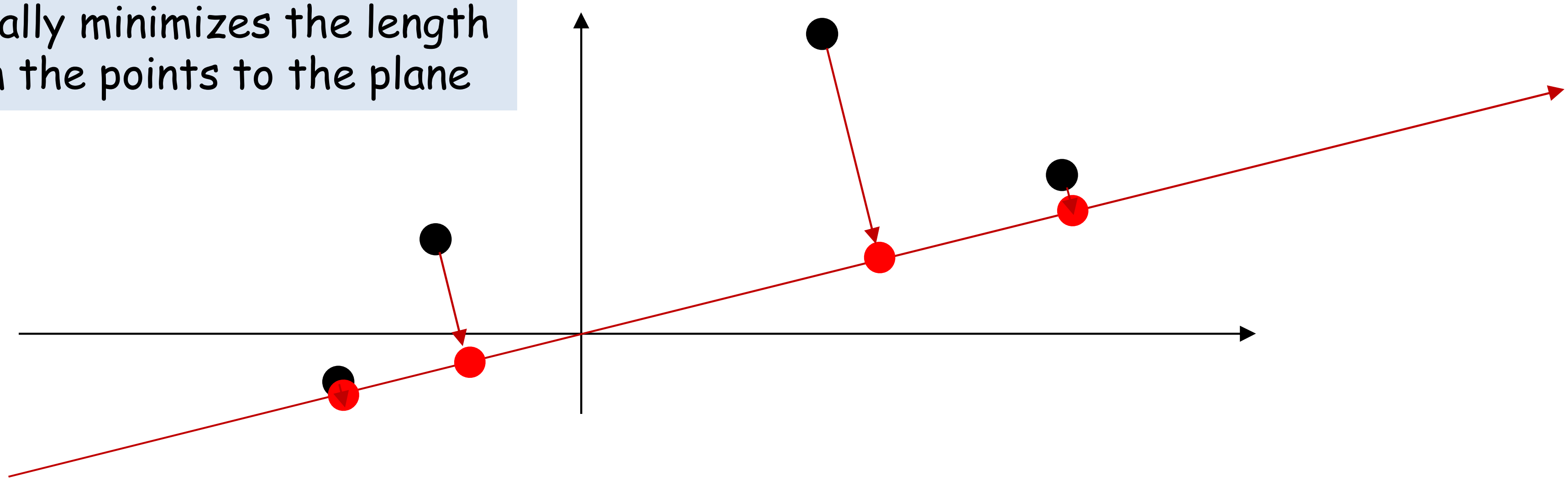
This *jointly* minimizes the total squared length of lines from the points to their "attachments" on the plane



- Initialize a subspace (the basis w)
- Iterate until convergence:
  - Find the best position vectors Z on the W subspace for each training instance
    - Find the location on W that is *closest* to each instance, i.e. the perpendicular projection
  - Let W rotate and stretch/shrink, keeping the arrangement of Z locations fixed
    - Minimize the total square length of the lines attaching the projection on the place to the instance

The iterative algorithm

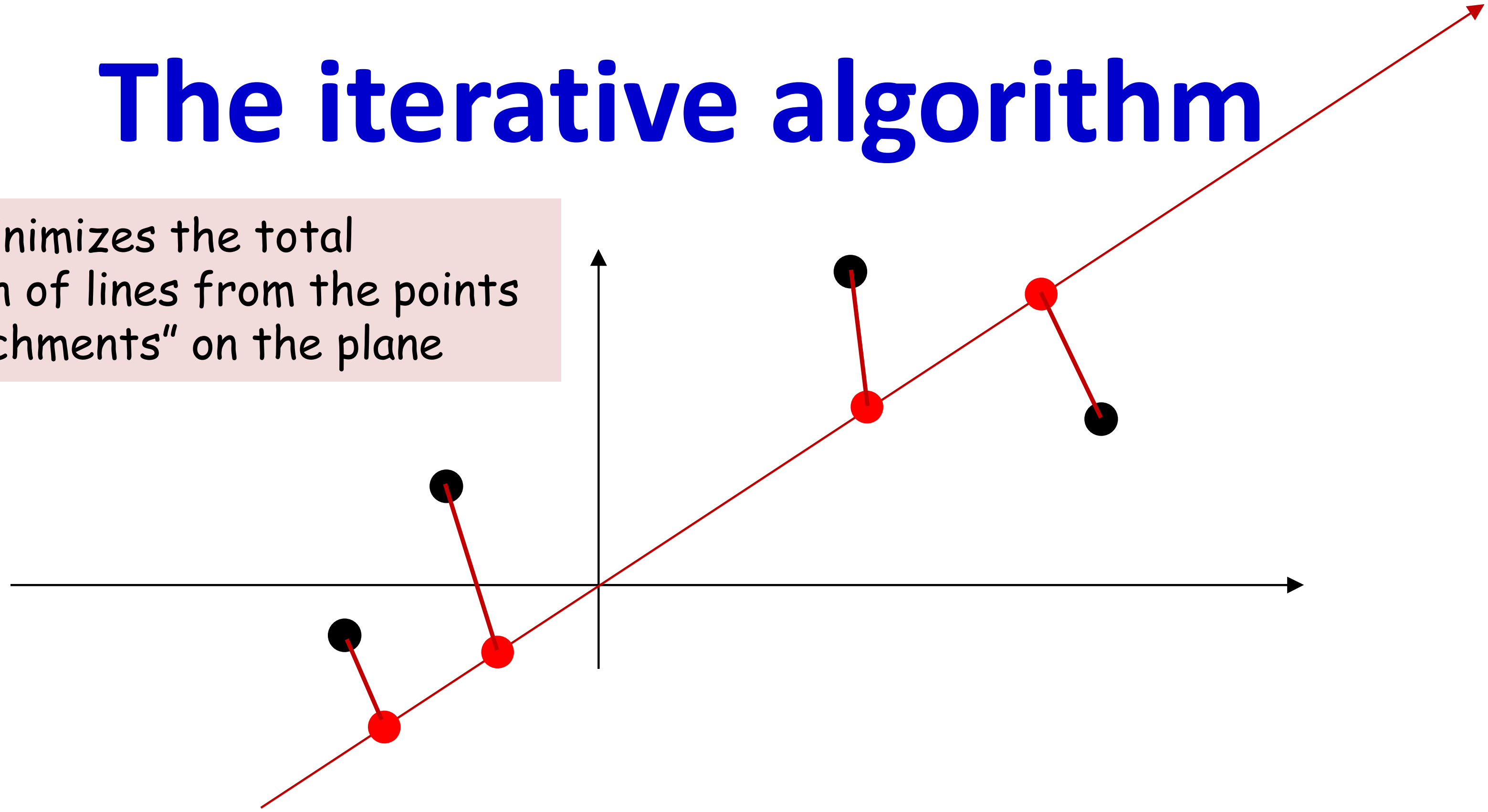
This individually minimizes the length of lines from the points to the plane



- Initialize a subspace (the basis w)
- Iterate until convergence:
  - Find the best position vectors Z on the W subspace for each training instance
    - Find the location on W that is *closest* to each instance, i.e. the perpendicular projection
  - Let W rotate and stretch/shrink, keeping the arrangement of Z locations fixed
    - Minimize the total square length of the lines attaching the projection on the place to the instance

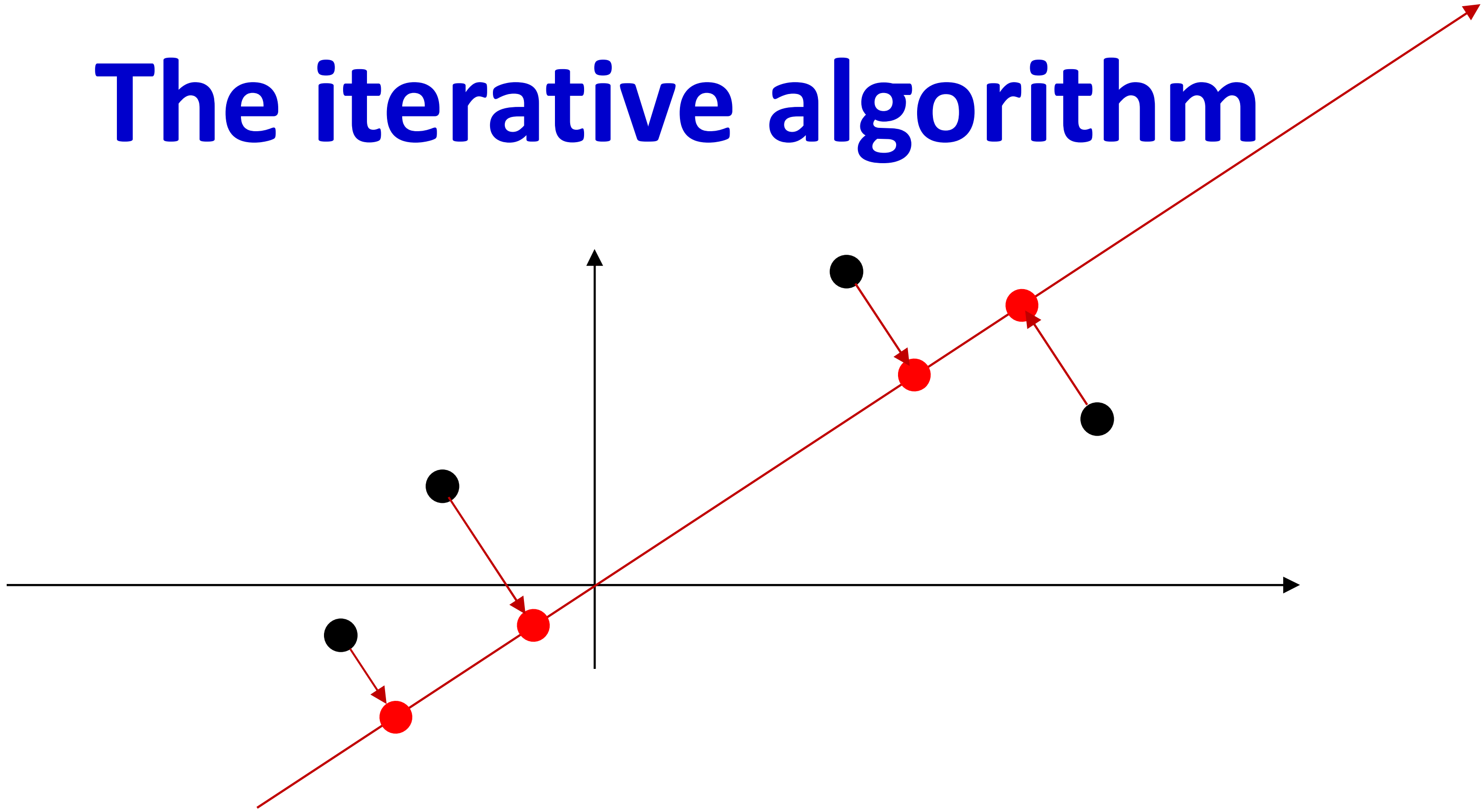
The iterative algorithm

This *jointly* minimizes the total squared length of lines from the points to their "attachments" on the plane



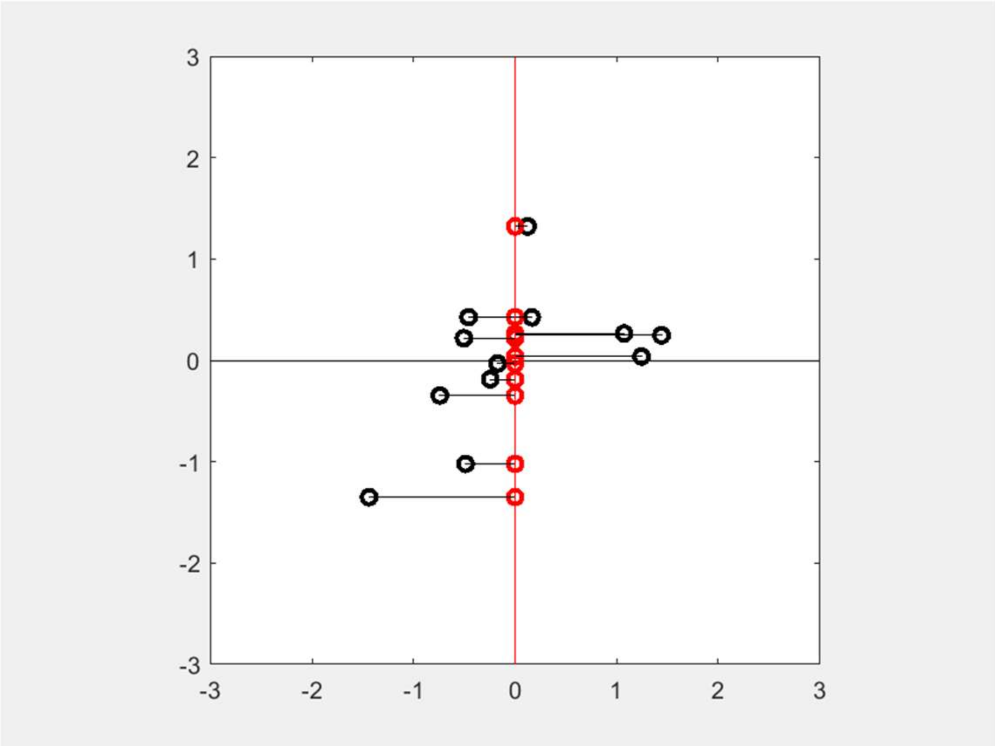
- Initialize a subspace (the basis w)
- Iterate until convergence:
  - Find the best position vectors Z on the W subspace for each training instance
    - Find the location on W that is *closest* to each instance, i.e. the perpendicular projection
  - Let W rotate and stretch/shrink, keeping the arrangement of Z locations fixed
    - Minimize the total square length of the lines attaching the projection on the place to the instance

The iterative algorithm



- Initialize a subspace (the basis w)
- Iterate until convergence:
  - Find the best position vectors Z on the W subspace for each training instance
    - Find the location on W that is *closest* to each instance, i.e. the perpendicular projection
  - Let W rotate and stretch/shrink, keeping the arrangement of Z locations fixed
    - Minimize the total square length of the lines attaching the projection on the place to the instance

A failed attempt at animation



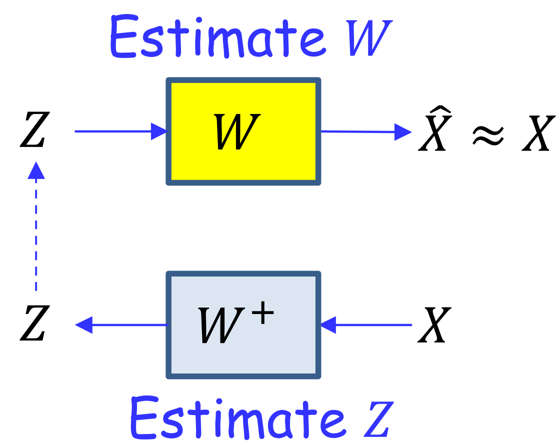
- Someone with animated-gif generation skills, help me…

A cartoon view of Iterative PCA

Iterate :

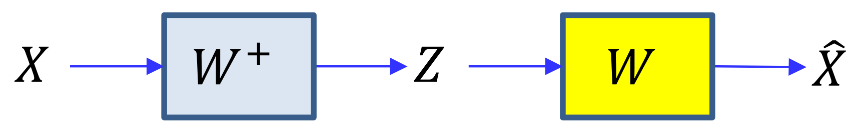
$Z \leftarrow W^{+}X$

$W \leftarrow XZ^{+}$



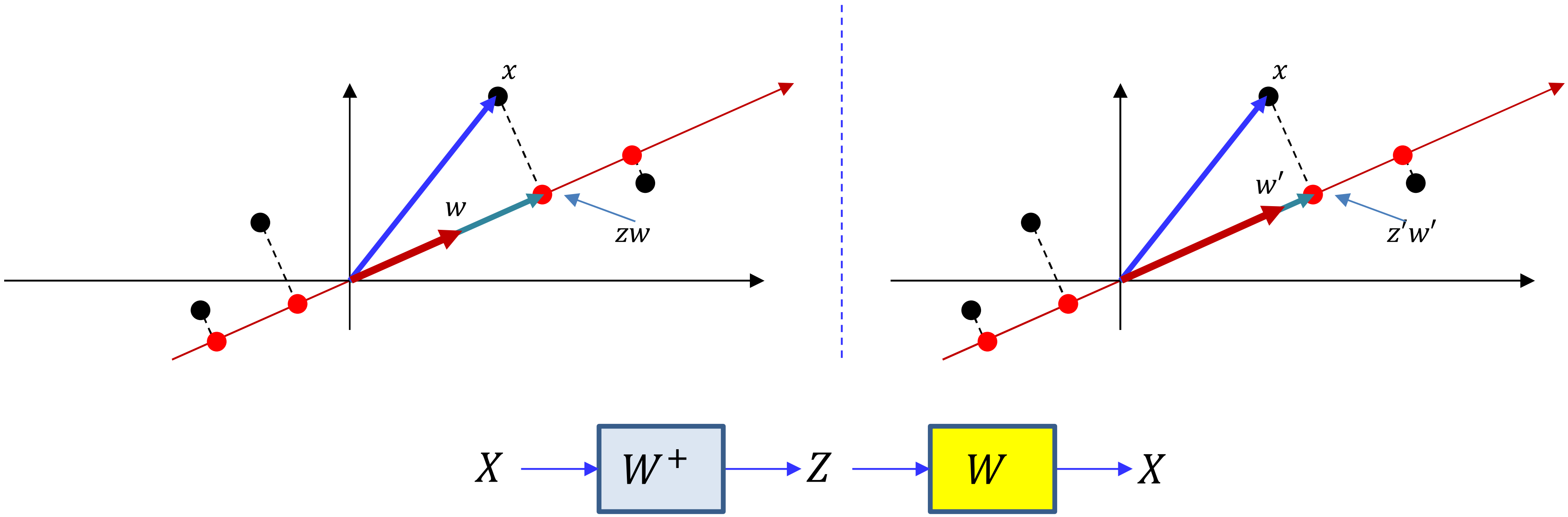
- Note that the real problem in estimating Z is computing W^{+}
  - If you know W^{+}, Z is obtained by a direct matrix multiply

Drawing this differently



- Look familiar?
- An autoencoder with linear activations
- Backprop actually works by simultaneously updating Z (implicitly) and W in tiny increments

A minor issue: Scaling invariance

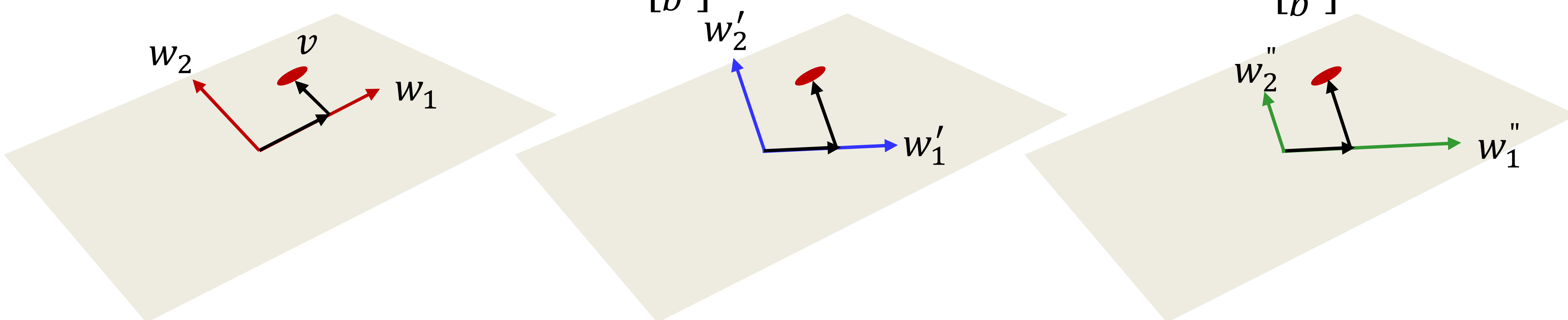


- The estimation is scale invariant
- We can increase the length of w, and compensate for it by reducing z
- The solution is not unique!

Rotation/scaling invariance

$v = aw_1 + bw_2$ $\qquad$ $v = a'w_1' + b'w_2'$ $\qquad$ $v = a''w_1'' + b''w_2''$

$z = \begin{bmatrix} a \\ b \end{bmatrix}$ $\qquad$ $z = \begin{bmatrix} a' \\ b' \end{bmatrix}$ $\qquad$ $z = \begin{bmatrix} a'' \\ b'' \end{bmatrix}$



- We can rotate and scale the vectors in W without changing the actual subspace they compose
- The representation of any point in the hyperspace in terms of these vectors will also change
  - The zs in the two cases will be related through a linear transform
- The subspace is invariant to transformations of z

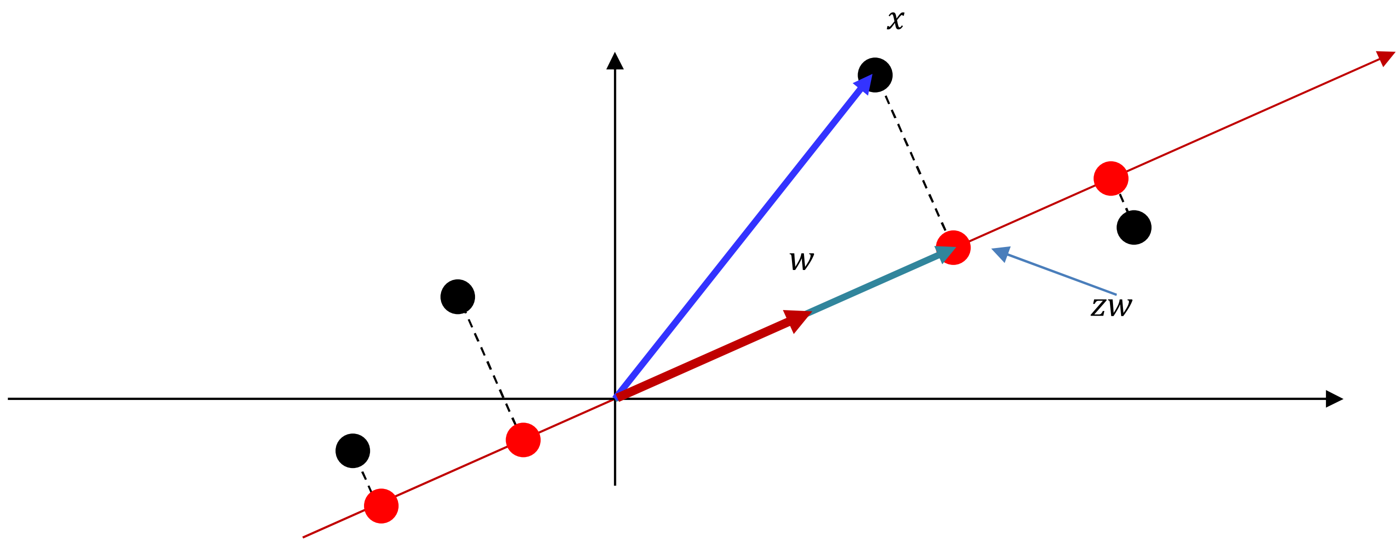
Transformation invariance

$$X \rightarrow \boxed{W^+} \rightarrow Z \rightarrow \boxed{W} \rightarrow \hat{X} \qquad X \approx WZ$$

- We can modify W to $W' = WB$, and Z to $Z' = B^{-1}Z$ such that $WZ = W'Z'$ $(WB)(B^{-1}Z)$
  - A different set of bases for the same subspace
- We can modify Z to $Z' = BZ$, and W to $W' = WB^{-1}$ such that $WZ = W'Z'$ $(WB^{-1})(BZ)$
  - A different set of bases for the same subpace

- The representation is invariant to invertible transforms of either W or Z
  - Although we will always find the same *subspace,* the *bases* and the *representations* in terms of these bases are not unique
  - I.e. there is no guarantee of which of the infinite possible solutions we will actually find

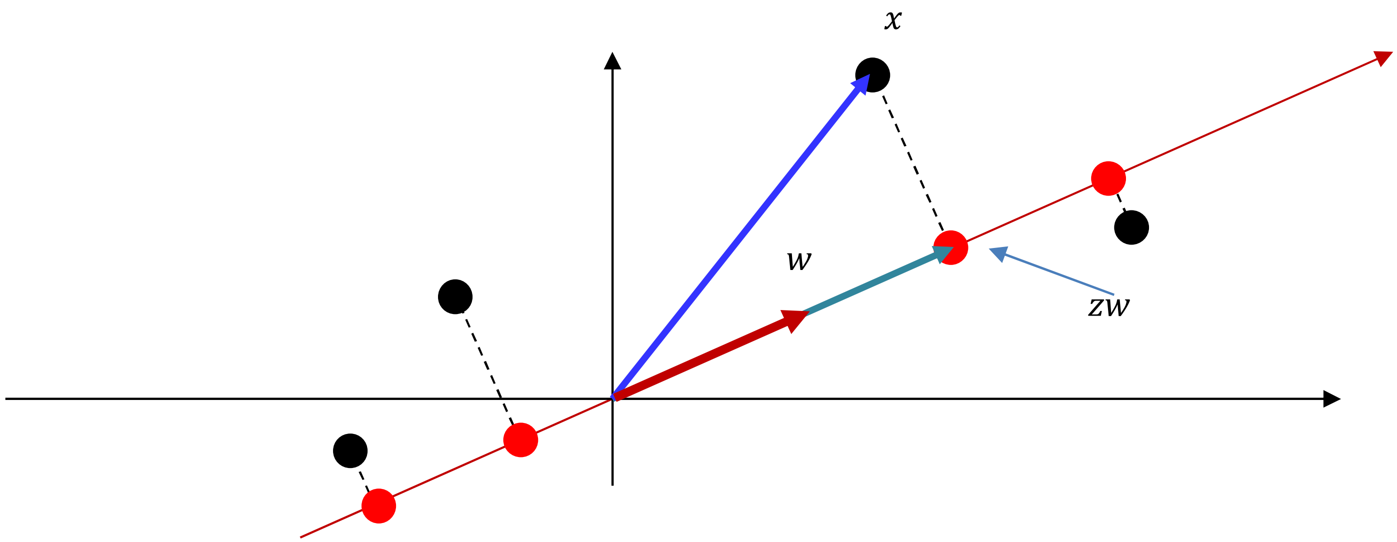
Resolving this issue



- A unique solution can be found by either
  - Requiring the vectors in W to be unit length and orthogonal
    - Standard “closed” form PCA
  - Constraining the variance of Z to be unity

- While the Ws estimated with the two solutions will be different, the resulting discovered principal subspace will be the same

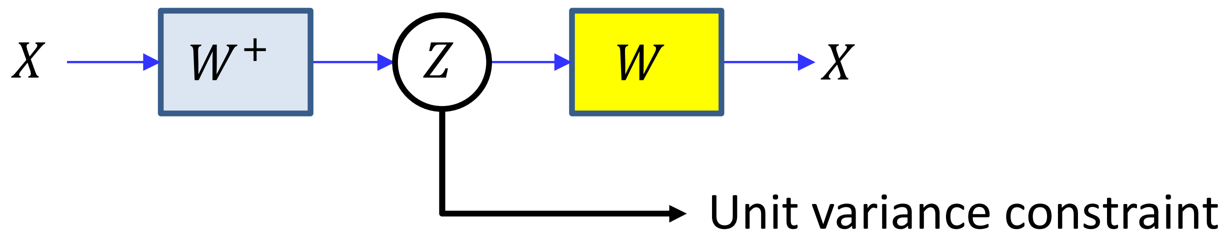
Resolving this issue



- A unique solution can be found by either
  - Requiring the vectors in W to be unit length and orthogonal
    - Standard “closed” form PCA
  - Constraining the variance of Z to be unity

- While the Ws estimated with the two solutions will be different, the resulting discovered principal subspace will be the same

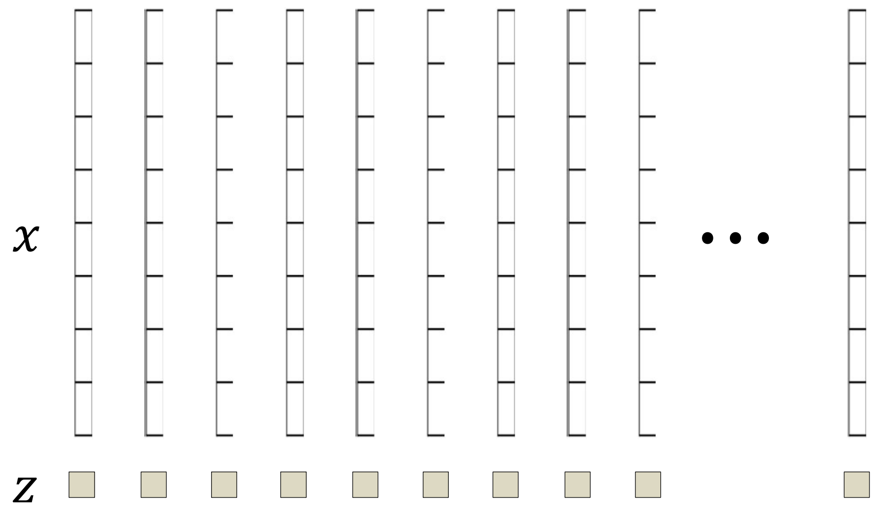
Constraining the linear AE

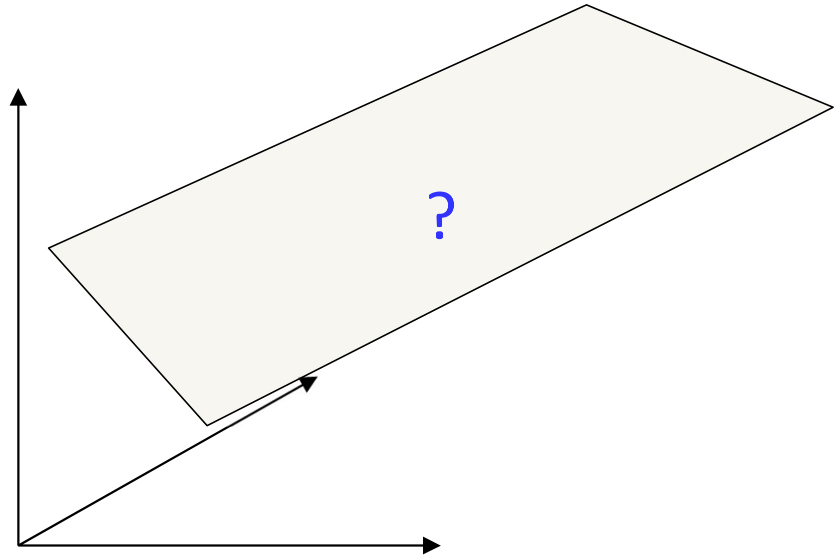


- The linear AE can be constrained to give you a unique(ish) solution

- Impose a unity constraint on the variance of Z
  - How?

So what are we doing in the iterative solution?

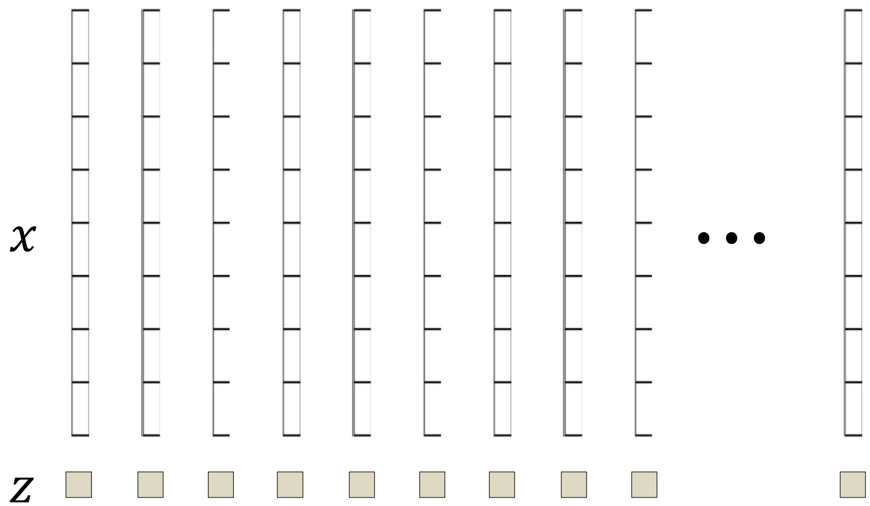


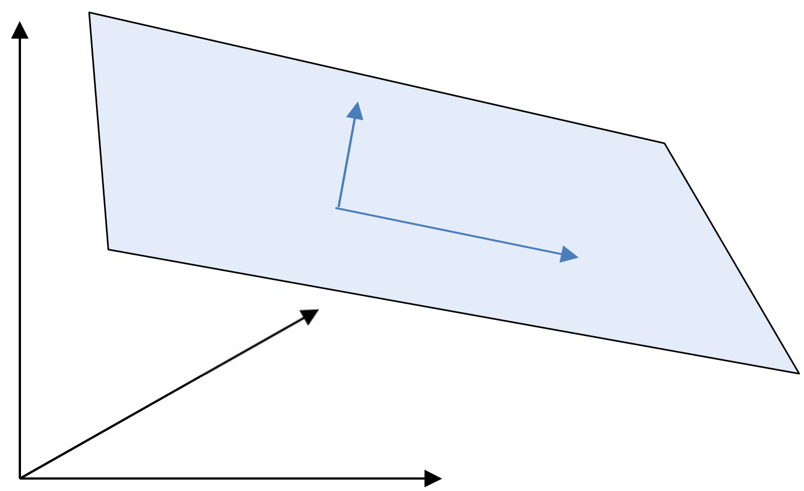


- For every training vector x, we are missing the information z about where the vector lies on the principal subspace hyperplane

- If we had z, we could uniquely identify the plane

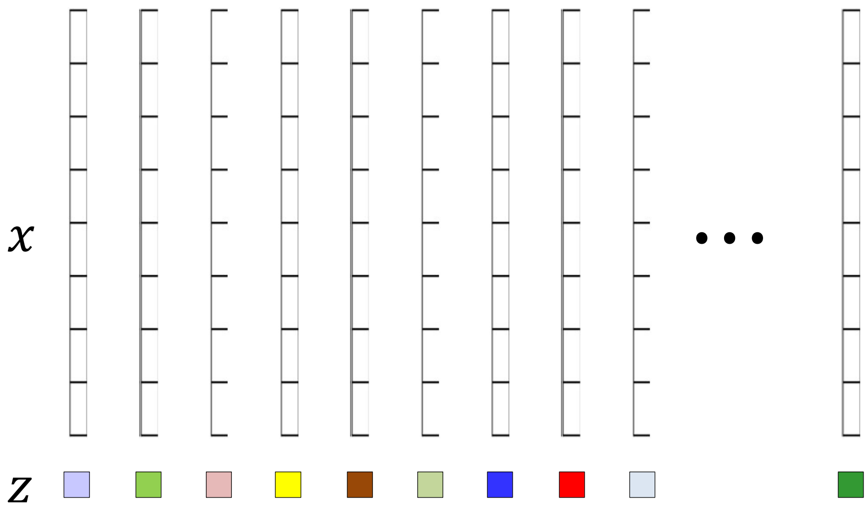
Iterative solution

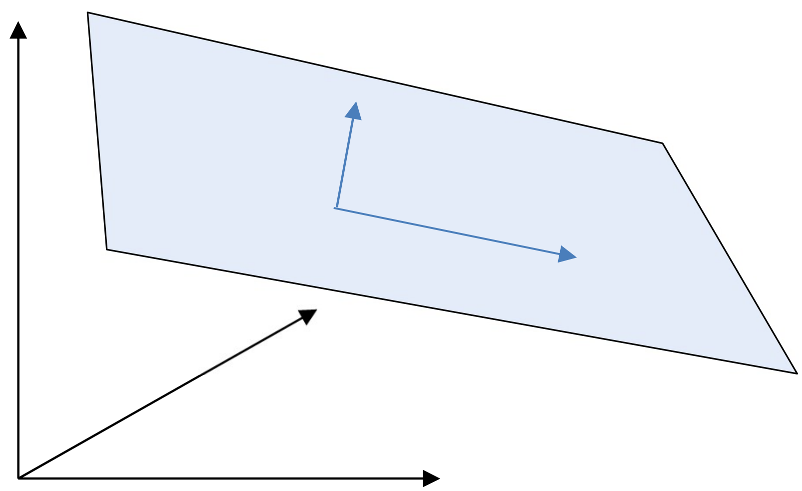




- Initialize the plane
  - Or rather, the bases for the plane

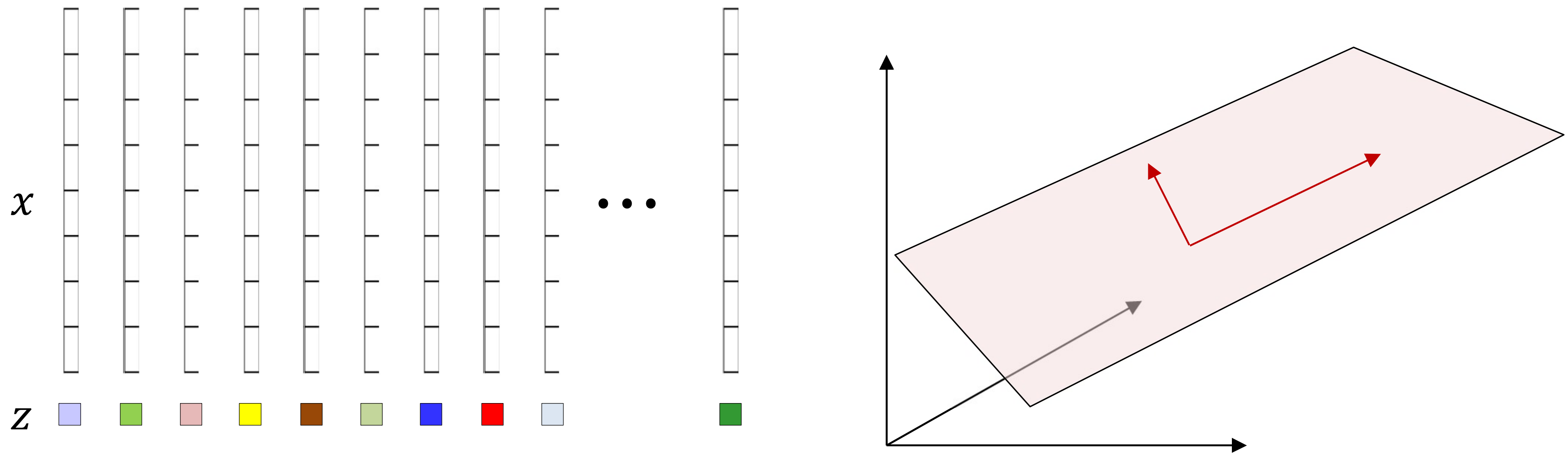
Iterative solution





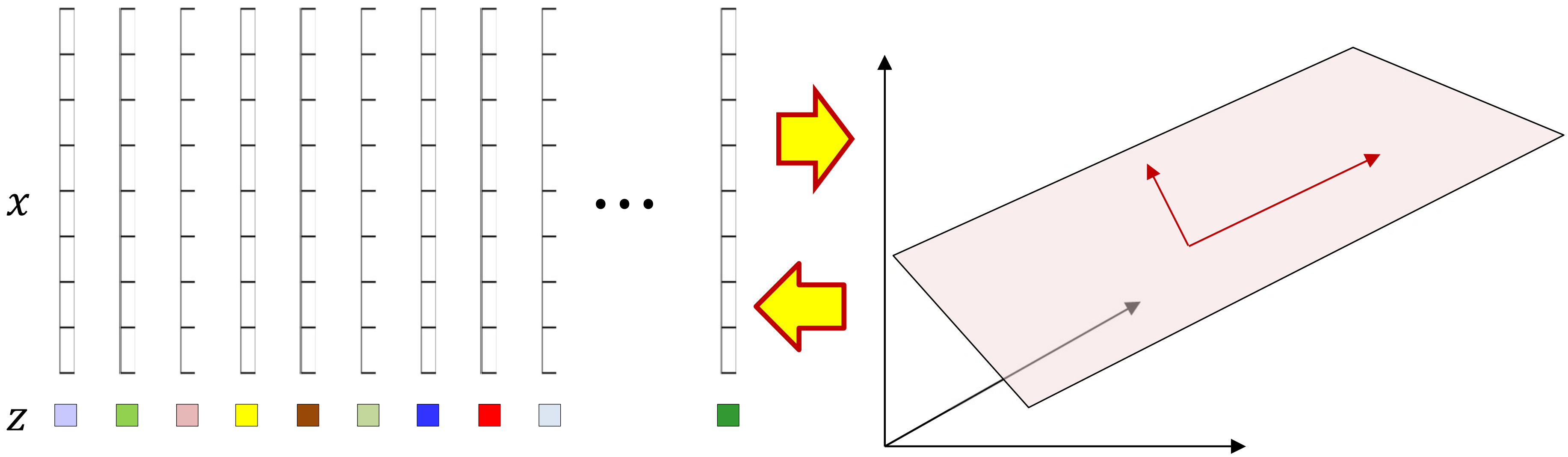
- Initialize the plane
  - Or rather, the bases for the plane
- "Complete" the data by computing the appropriate zs for the plane

Iterative solution



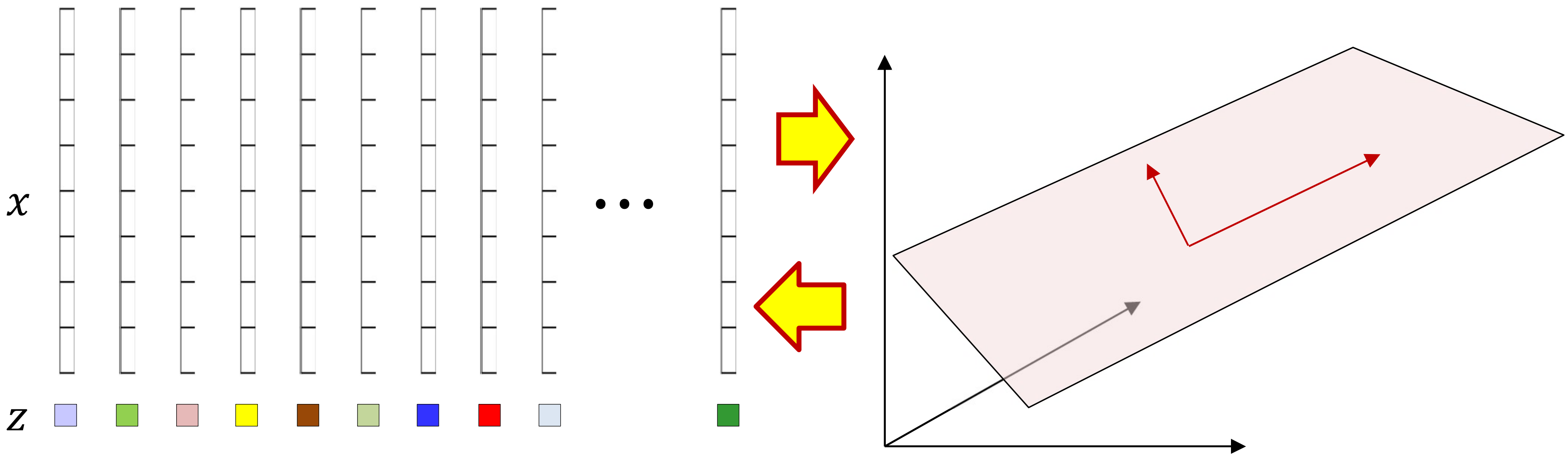
- Initialize the plane
  - Or rather, the bases for the plane
- "Complete" the data by computing the appropriate zs for the plane
- Reestimate the plane using the zs

Iterative solution



- Initialize the plane
  - Or rather, the bases for the plane
- "Complete" the data by computing the appropriate zs for the plane
- Reestimate the plane using the zs
- Iterate

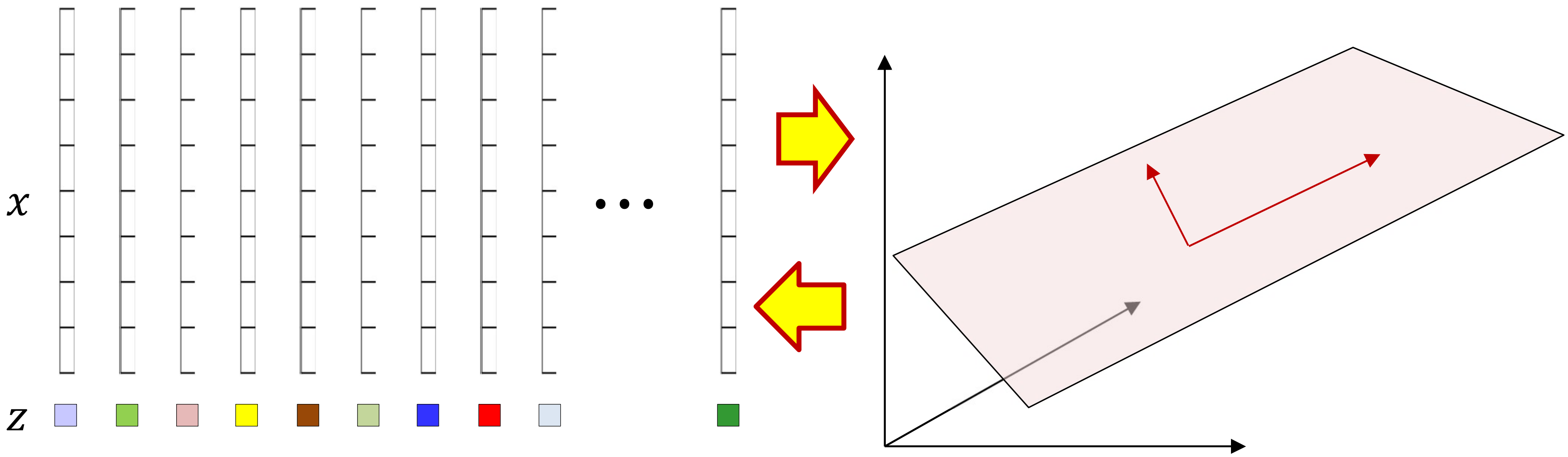
Iterative solution



- Initialize the plane
  - Or rather, the bases for the plan
- "Complete" the da
- Reesti
- Iterate

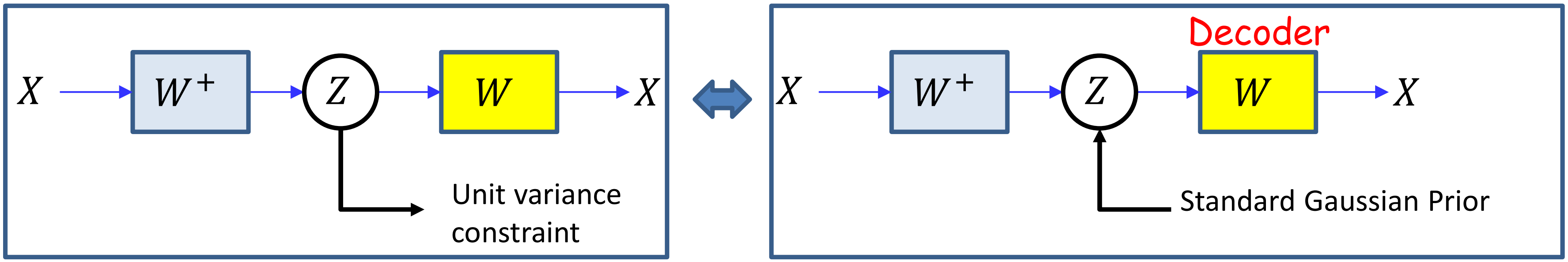
Look Familiar?

Iterative solution



- This looks like EM
  - In fact it is
- But what is the generative model?
- And what distribution is this encoding?

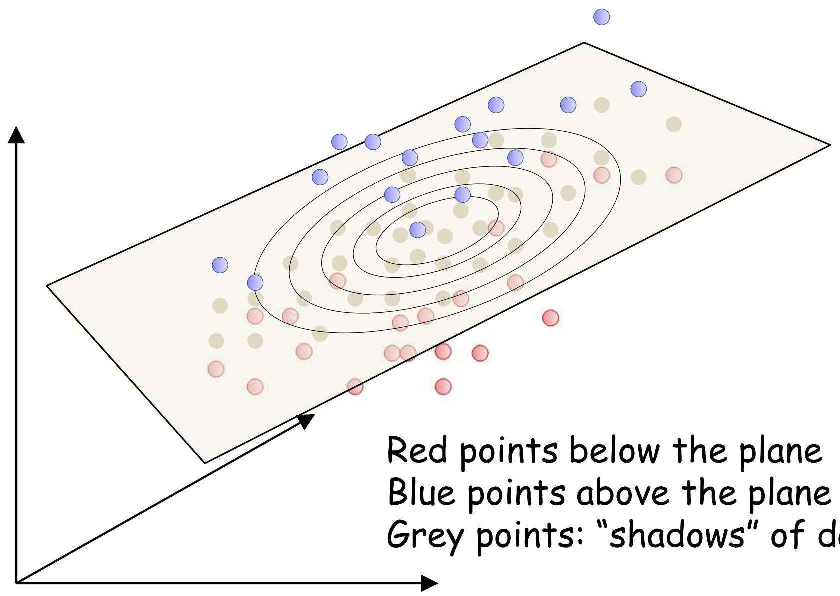
Constraining the linear AE

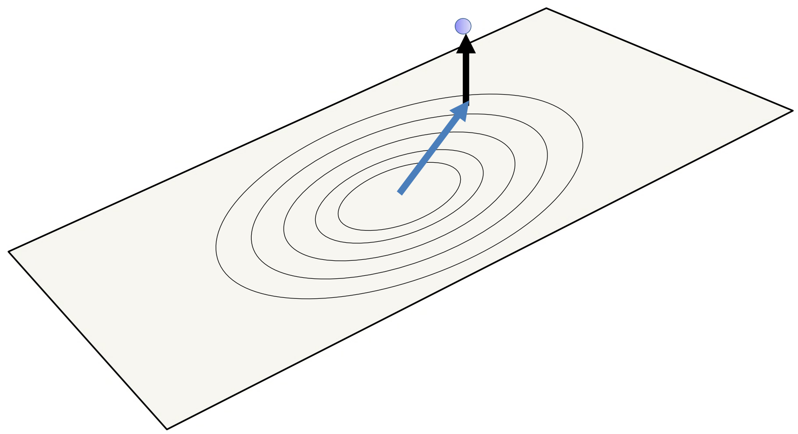


- Imposing the constraint that z must have unit variance is the same as assuming that z is drawn from a standard Gaussian
  – 0 mean, unit variance!

- The decoder of the AE with the unit-variance constraint on z is in fact a Generative model

The *generative* story behind PCA (linear AEs)



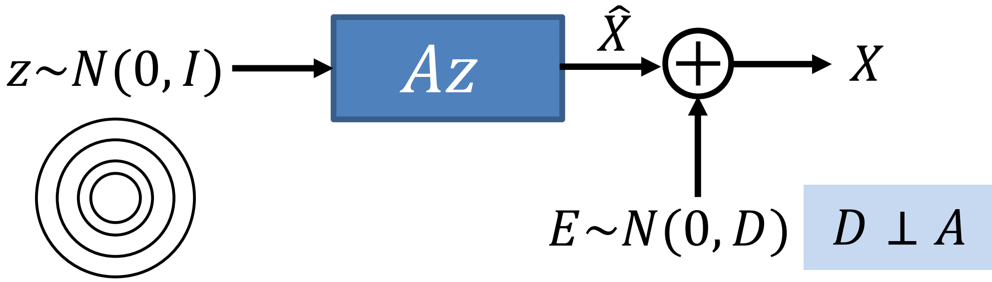


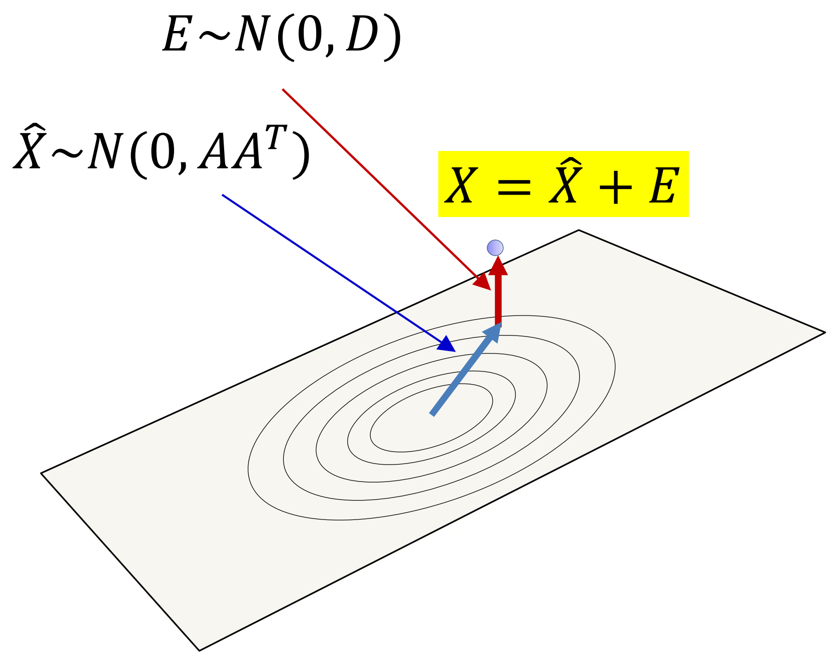
- Linear AEs actually have a generative story
- In order to generate any point
  - We first take a Gaussian step on the principal plane
  - Then we take an orthogonal *Gaussian* step from where we land to generate a point
  - PCA / Linear AEs find the plane and the characteristics of the Gaussian steps from the data

The *generative* story behind PCA (linear AEs)

$z \sim N(0, I)$
$E \sim N(0, D)$

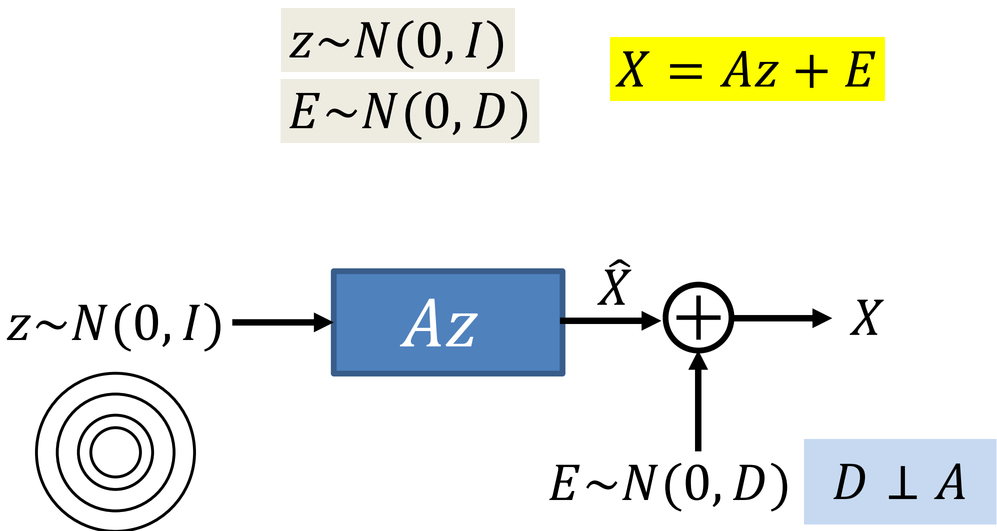
$X = Az + E$

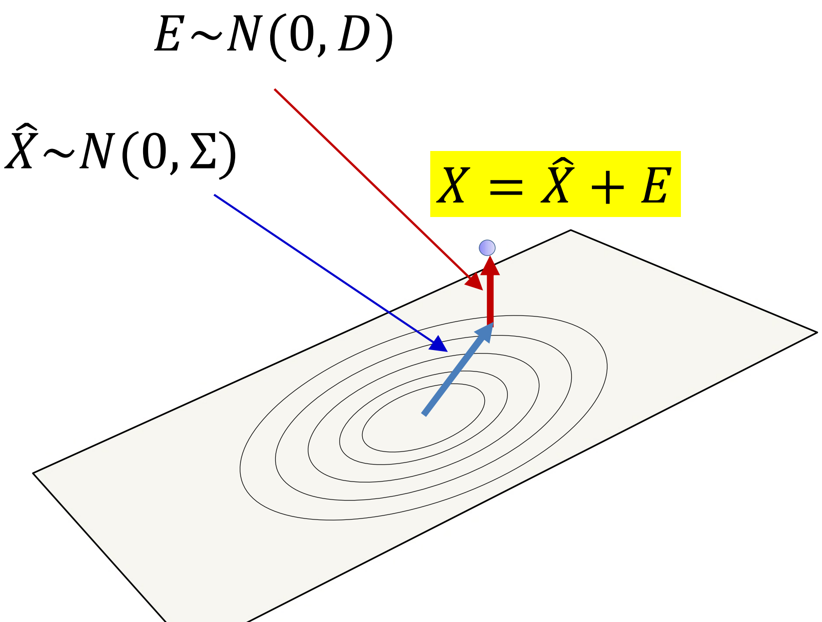




- **Generative story for PCA:**
  - z is drawn from a K-dim isotropic Gaussian
    - K is the dimensionality of the principal subspace
  - A is "basis" matrix
    - Matrix of principal Eigen vectors scaled by Eigen values
  - E is a 0-mean Gaussian noise that is orthogonal to the principal subspace
    - **The covariance of the Gaussian is low-rank and orthogonal to the principal subspace!**

The *generative* story behind PCA (linear AEs)



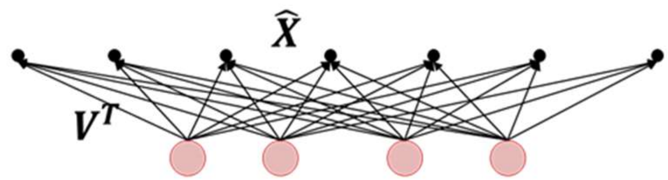


PCA implicitly obtains maximum likelihood estimate of A and D, from training data X

- **Generative story for PCA:**
  - z is drawn from a K-dim isotropic Gaussian
    - K is the dimensionality of the principal subspace
  - A is "basis" matrix
    - Matrix of principal Eigen vectors scaled by Eigen values
  - E is a 0-mean Gaussian noise that is orthogonal to the principal subspace
    - **The covariance of the Gaussian is low-rank and orthogonal to the principal subspace!**

The *generative* (PCA) story of linear AEs

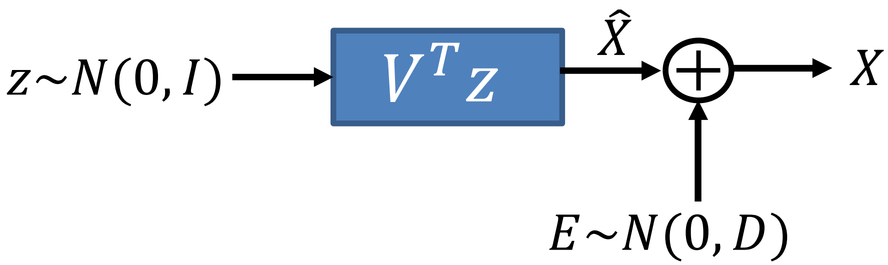
Changed notation
$V^T = A$
$\hat{X} = Az$



Note: the generative model is the *decoder*

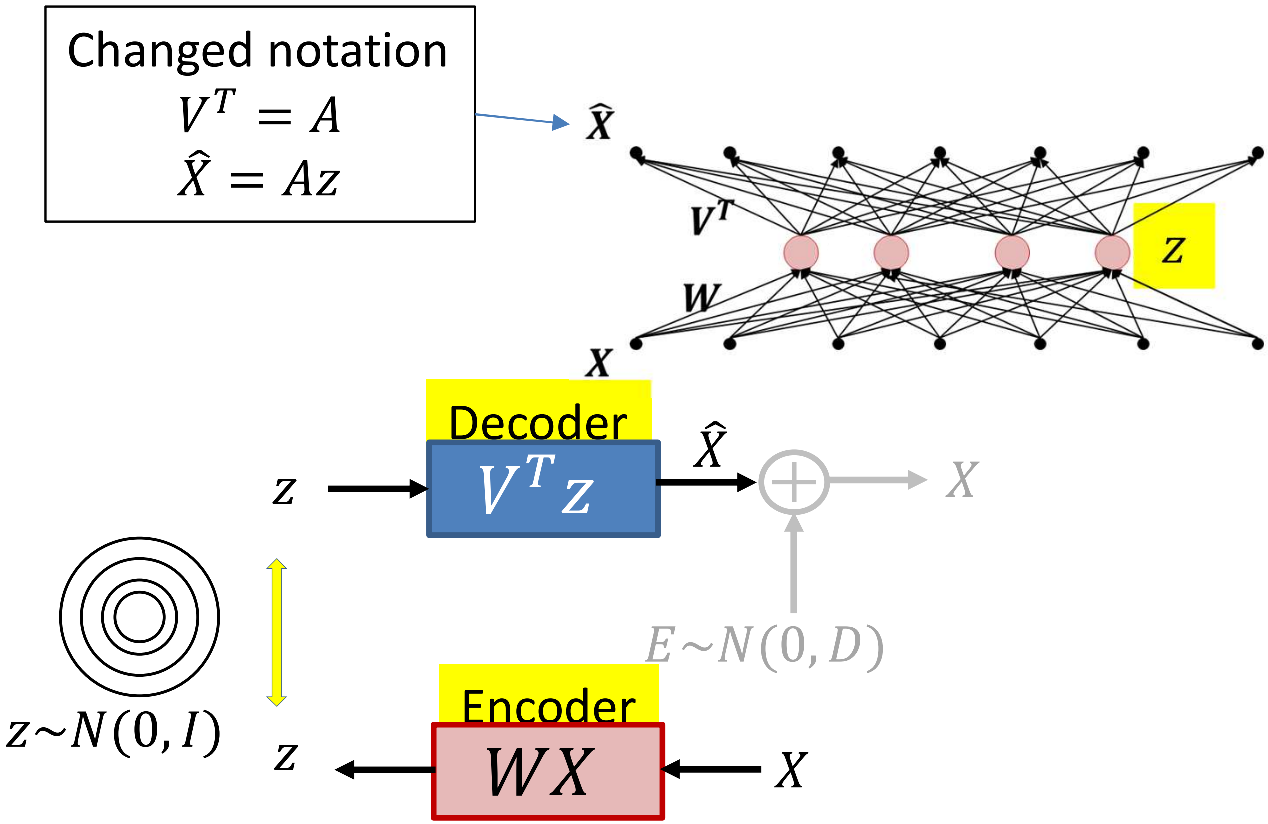
$z \sim N(0, I)$ $\hat{X} = V^T z$

$E \sim N(0, D)$



- The decoder weights are just the PCA basis matrix

The *generative* (PCA) story of linear AEs



Note: the generative model is the *decoder*

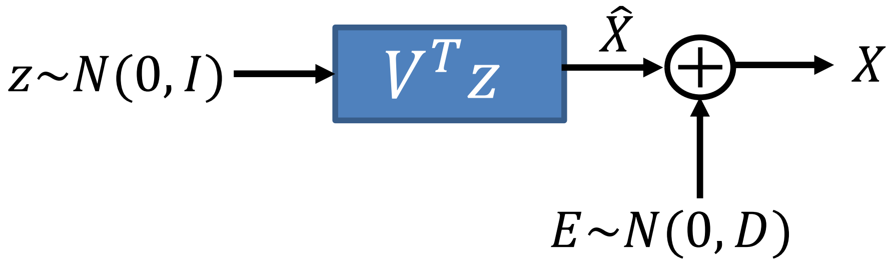
The *encoder* finds the specific z for any input x

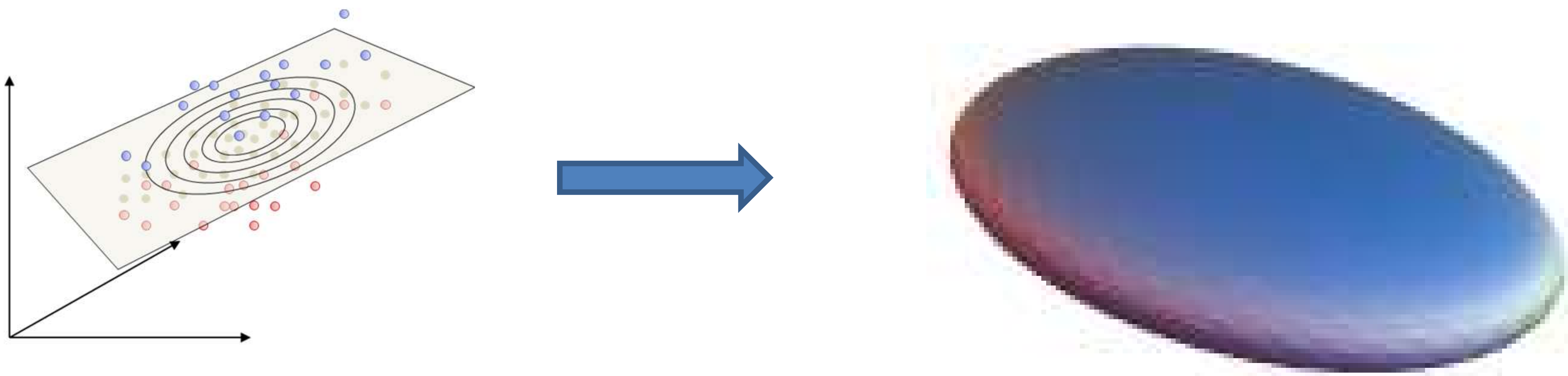
$z \sim N(0, I)$ $\hat{X} = V^T z$

$z = WX$

- The decoder weights are just the PCA basis matrix
- The encoder only projects the data into latent Gaussian position variable z
- Encoder: transforms input X into Gaussian z
- Decoder: transforms Gaussian z into principal subspace reconstruction $\hat{X}$

The distribution modelled by PCA





- If z is Gaussian, $\hat{X}$ is Gaussian
- $\hat{X}$ and E are Gaussian => X is Gaussian
- PCA model: The observed data are Gaussian
  - Gaussian data lying very close to a principal subspace
  - Comprising "clean" Gaussian data on the subspace plus orthogonal noise

Poll 4 (@1764)

Select all that are true about PCA

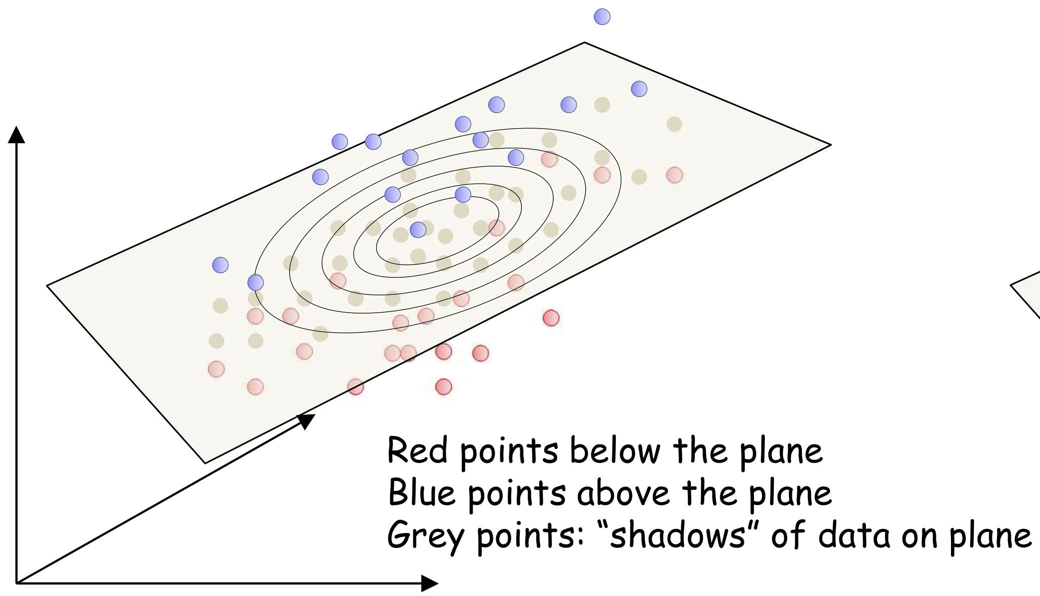
- PCA finds the principal subspace, such that approximating all training data by their projections onto this subspace results in the lowest error
- An optimal autoencoder with linear activations reconstructs all data as their projections on the principal subspace
- The bases of this subspace can be uniquely estimated without constraints
- One way to uniquely estimate the subspace is to require the bases of the subspace (the decoder weights of the AE) to be orthonormal
- Another way to estimate the subspace uniquely is to require the distribution of the latent variable Z to be standard Gaussian
- The decoder weights estimated using both above solutions will be the same

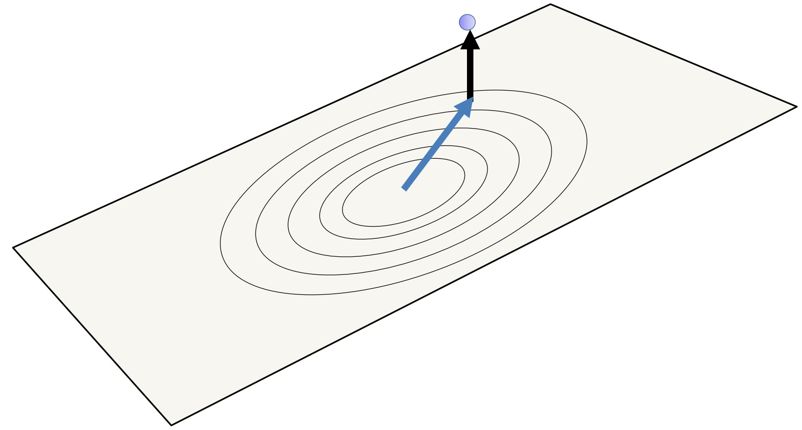
Poll 4

Select all that are true about PCA

- **PCA finds the principal subspace, such that approximating all training data by their projections onto this subspace results in the lowest error**
- **An optimal autoencoder with linear activations reconstructs all data as their projections on the principal subspace**
- The bases of this subspace can be uniquely estimated without constraints
- **One way to uniquely estimate the subspace is to require the bases of the subspace (the decoder weights of the AE) to be orthonormal**
- **Another way to estimate the subspace uniquely is to require the distribution of the latent variable Z to be standard Gaussian**
- The decoder weights estimated using both above solutions will be the same

Can we do better?





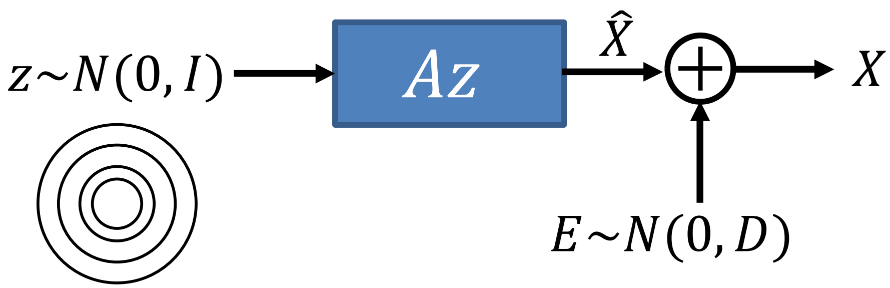
- PCA assumes the noise is always orthogonal to the data
  – Not always true
  – Noise in images can look like images, random noise can sound like speech, etc.
- Let us generalize the model to permit non-orthogonal noise

The Linear Gaussian Model

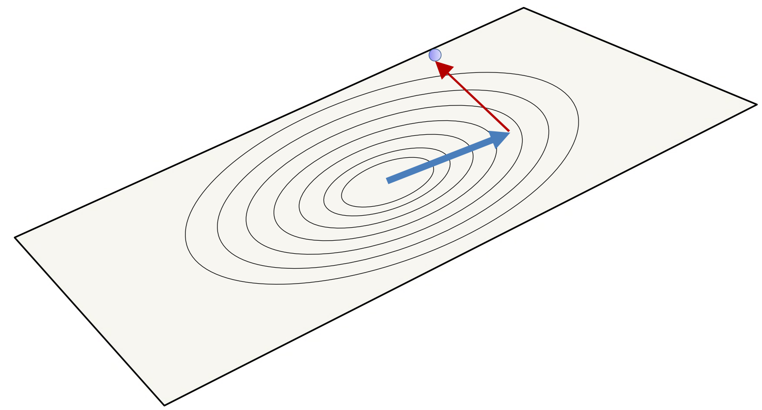
$z \sim N(0, I)$

$E \sim N(0, D)$

$X = Az + E$



D is full rank



- Update the model: The noise added to the output of the encoder can lie in *any* direction
  - Uncorrelated, but not just orthogonal to the principal subspace

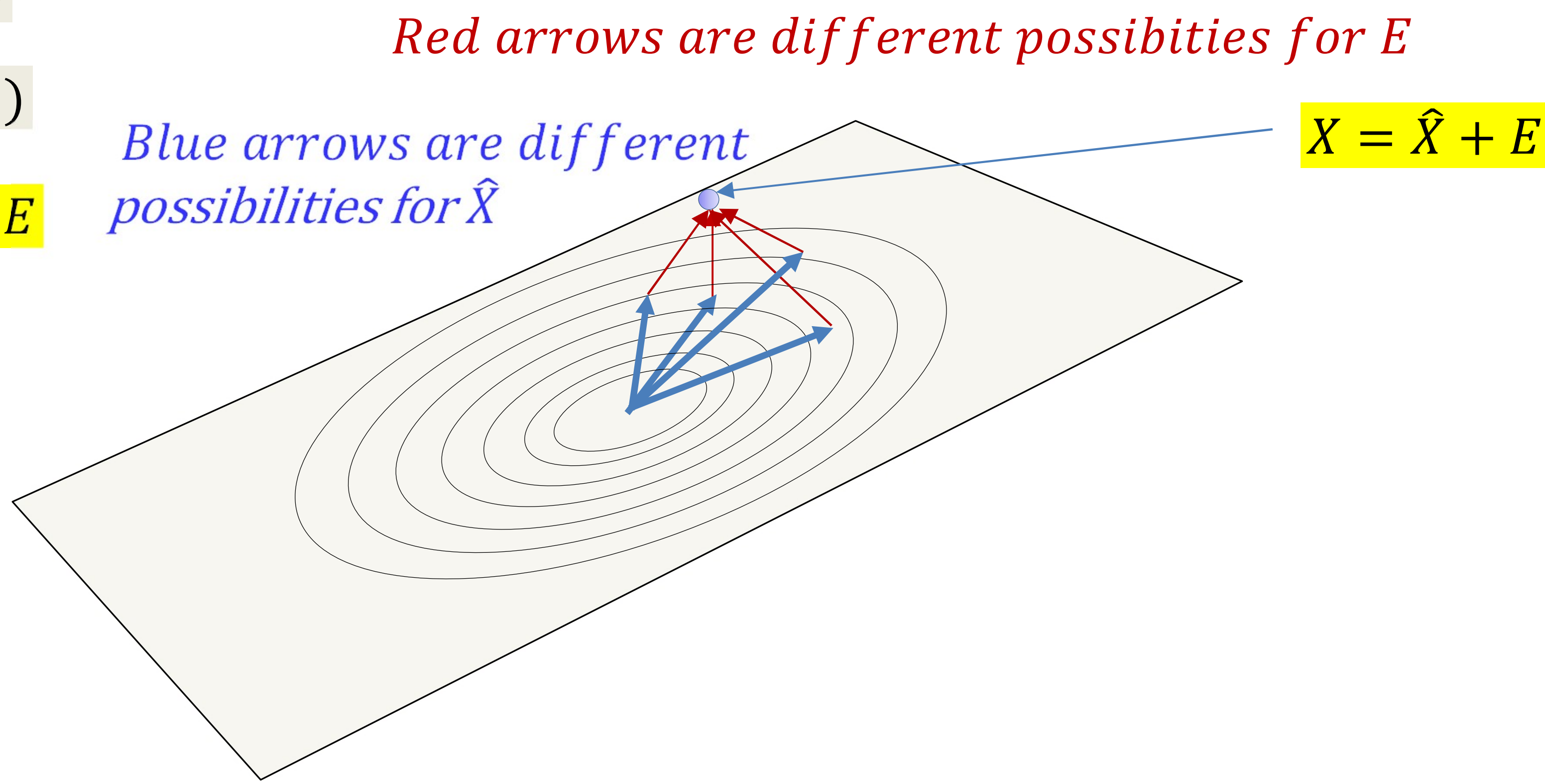
- Generative model: to generate any point
  - Take a Gaussian step on the hyperplane
  - Add *full-rank* Gaussian uncorrelated noise that is independent of the position on the hyperplane
    - Uncorrelated: diagonal covariance matrix
    - Direction of noise is unconstrained
      - Need not be orthogonal to the plane

The linear Gaussian model

$z \sim N(0, I)$

$E \sim N(0, D)$

$X = Az + E$



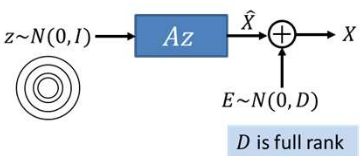
- The way to produce any data instance is no longer unique
  - though different corrections may have different probabilities

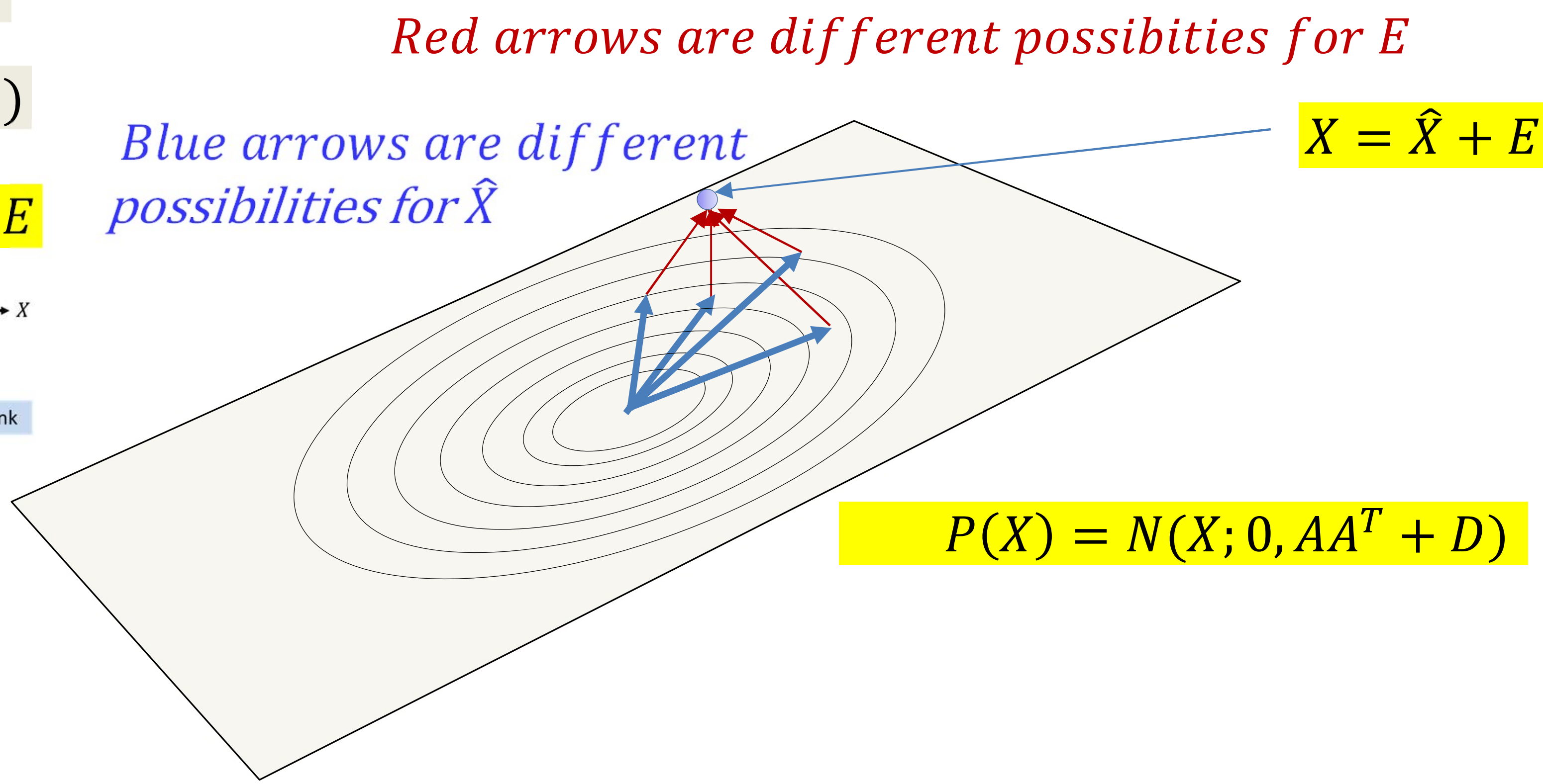
The linear Gaussian model

$z \sim N(0, I)$

$E \sim N(0, D)$

$X = Az + E$





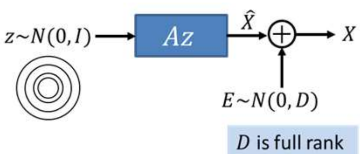
- The way to produce any data instance is no longer unique
  - though different corrections may have different probabilities
- This is still a parametric model for a Gaussian distribution
  - Parameters are A and D (assuming 0 mean)

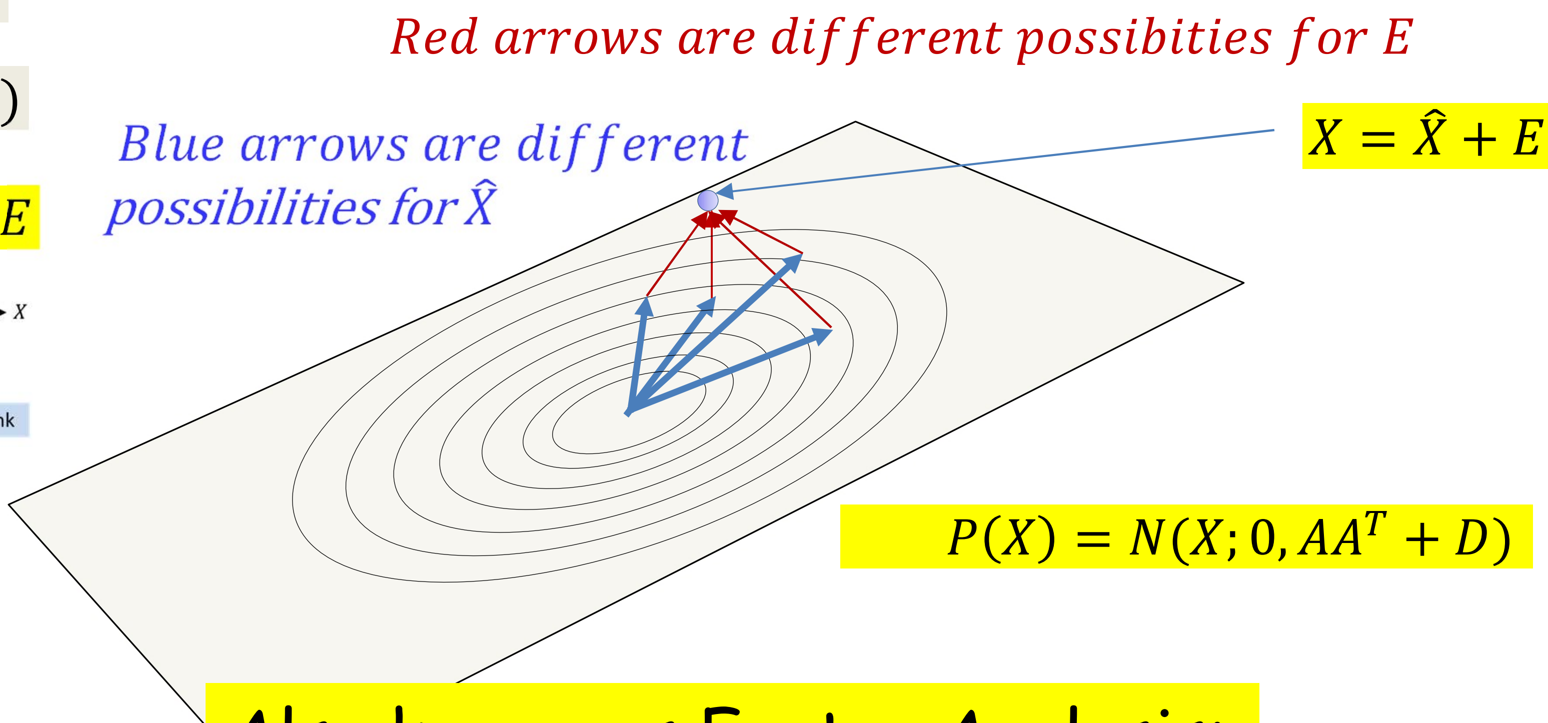
The linear Gaussian model

$z \sim N(0, I)$

$E \sim N(0, D)$

$X = Az + E$

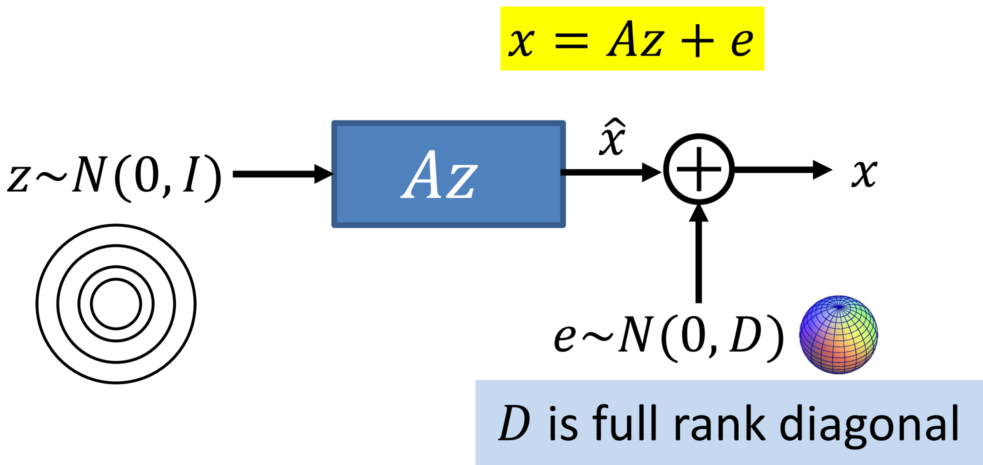


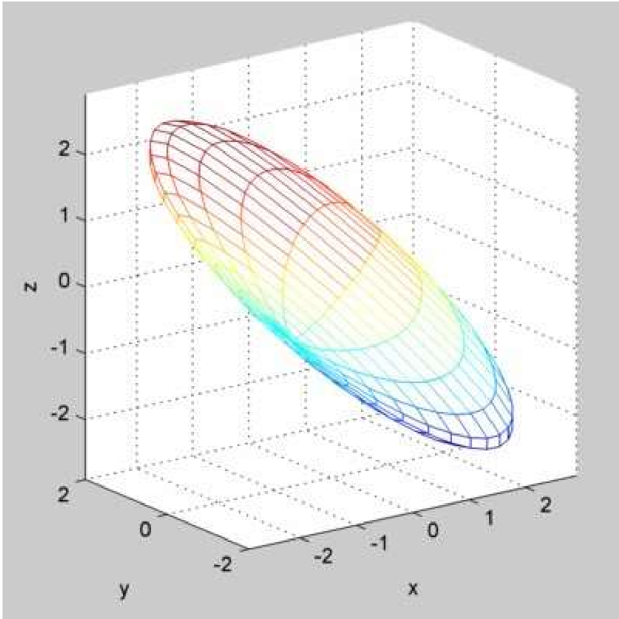


Also known as Factor Analysis:
A is the loading matrix
z are the factors
D is diagonal

- The way to p ... er unique
  - though diff ... babilities
- This is in fact ... distribution
  - Parameters are A and D (assuming 0 mean)

The probability distribution modelled by the LGM





- The noise added to the output of the encoder can lie in *any* direction
- The probability density of x is Gaussian lying mostly close to a hyperplane
  - With uncorrelated Gaussian noise

Story for the day

- EM: An iterative technique to estimate probability models for data with missing components or information
  - By iteratively "completing" the data and reestimating parameters

- PCA: Is actually a generative model for Gaussian data
  - Data lie close to a linear manifold, with orthogonal noise

- Factor Analysis: Also a generative model for Gaussian data
  - Data lie close to a linear manifold
  - Like PCA, but without directional constraints on the noise

- Will continue with FA and Variational AutoEncoders in the next class