Neural Networks

Variational Autoencoders
Part 2

(based in part on slides from Dan Schwartz and Tom Manzini)

Attendance: @1848

Recap: 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

Recap: 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?

Recap: Generative models and Maximum Likelihood Estimation

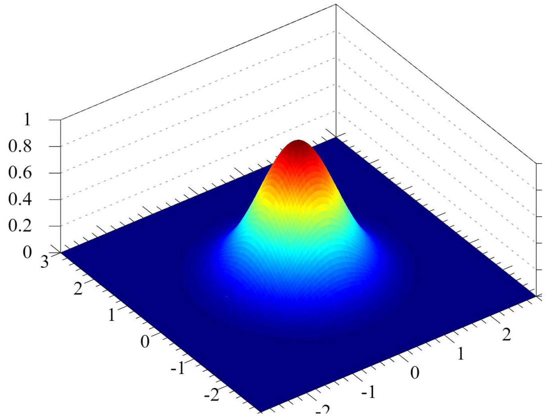
What is a generative model

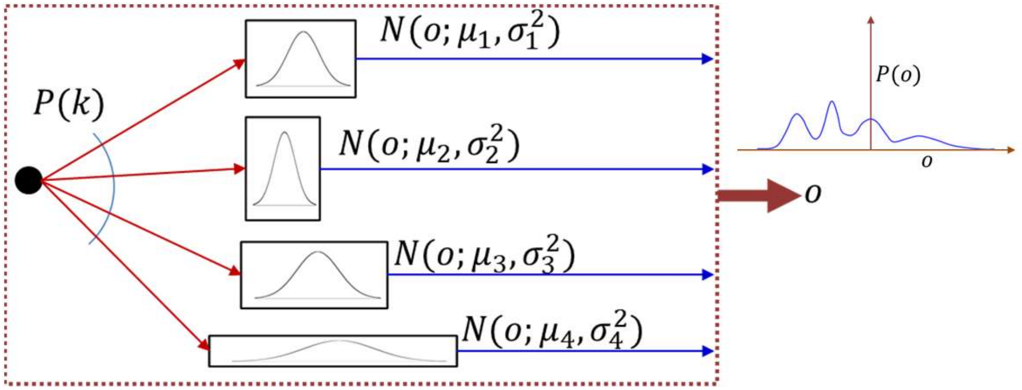
- In statistical estimation, a *generative model* is a functional or computational model for the probability distribution of a given data
  - Can be represented generically as $P(x;\theta)$, where x represents a data instance and θ are the *parameters* of the model
  - But actually encodes a *generative story* for how the data were produced

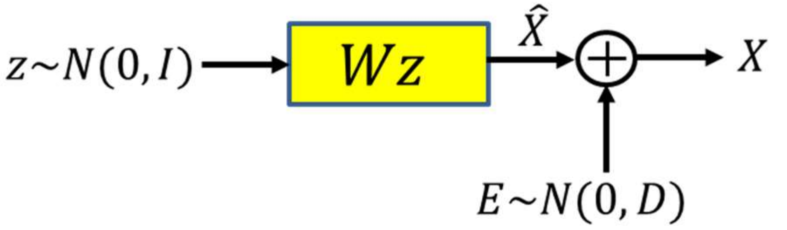
- Utility of the model
  - Can compute the probability of observing a given value x
  - Can also be used to *generate* samples of (or statistically similar to) the data

Recap: Examples of Generative Models

- Generative models can be simple, one step models of generation
  - E.g. Gaussians, Category distributions
- Or a multi-step generating process
  - E.g. Gaussian Mixtures
  - E.g. Linear Gaussian Models







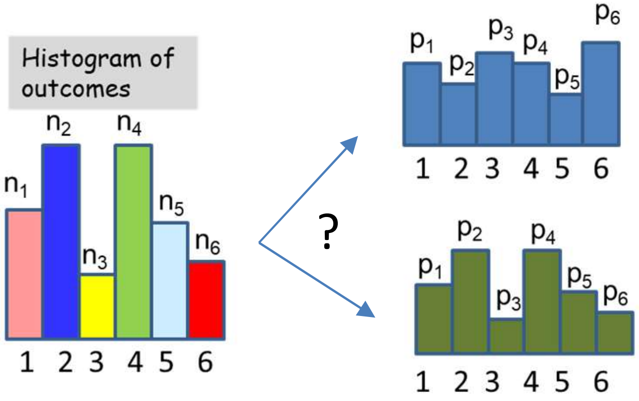
Recap: ML Estimation of Generative Models

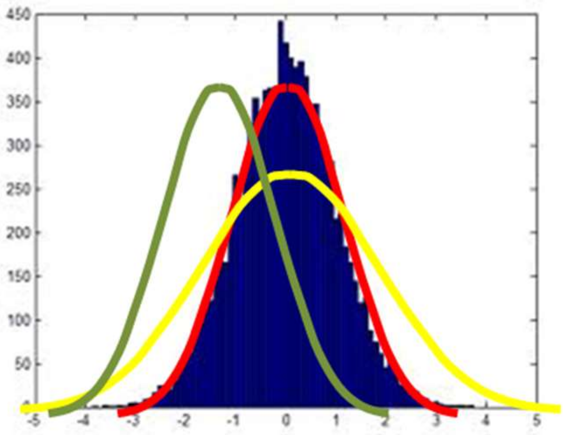
- Must estimate the parameters of the model from observed data

- Maximum likelihood estimation: Choose parameters to maximize the (log) likelihood of observed data

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \log(P(X;\theta))$$

$$= \underset{\theta}{\operatorname{argmax}} \sum_{x \in X} \log(P(x;\theta))$$





Recap: ML estimation from incomplete data

- In many situations, our observed data are missing information
  - E.g. components of the data
  - E.g. "inside" information about how the data are drawn by the model

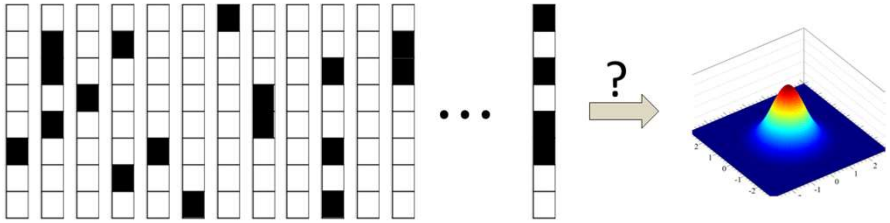
- In these cases, the ML estimate must only consider the *observed* data O

$$\underset{\theta}{\operatorname{argmax}} \sum_{o \in O} \log P(o; \theta)$$

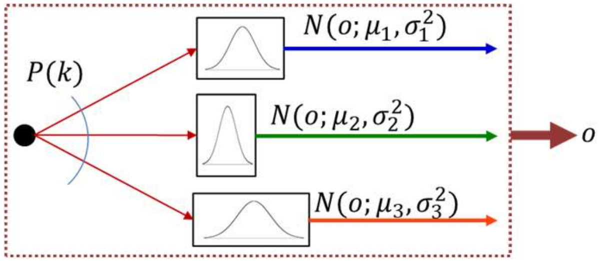
  - But the observed data are incomplete

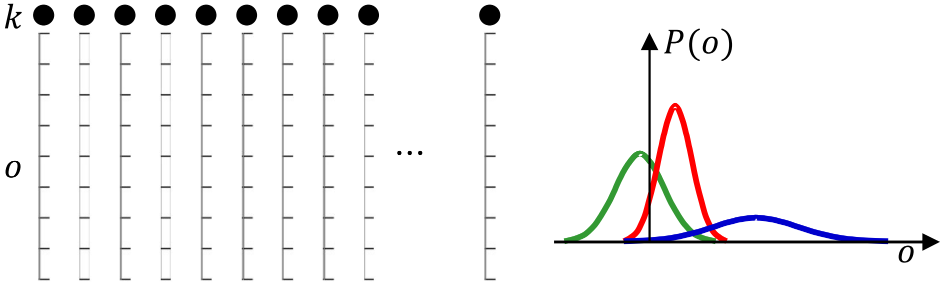
- Observation probability $P(o)$ must be obtained from the *complete* data probability, by marginalizing out missing components
  - This can cause ML estimation to become challenging

Observation O consists of only the unblackened components



Blacked-out components are missing from data





Recap: ML estimation from incomplete data

- ML estimate from *observed* data O

$$\underset{\theta}{\operatorname{argmax}} \sum_{o \in O} \log P(o;\theta)$$

- $P(o)$ is obtained by marginalizing out missing components

$$P(o;\theta) = \sum_{h} P(h,o;\theta)$$

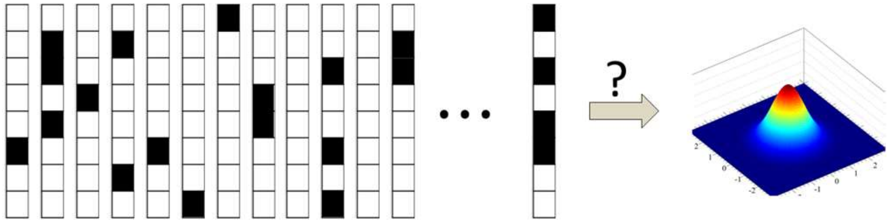
- The ML estimate becomes

$$\underset{\theta}{\operatorname{argmax}} \sum_{o \in O} \log \sum_{h} P(h,o;\theta)$$

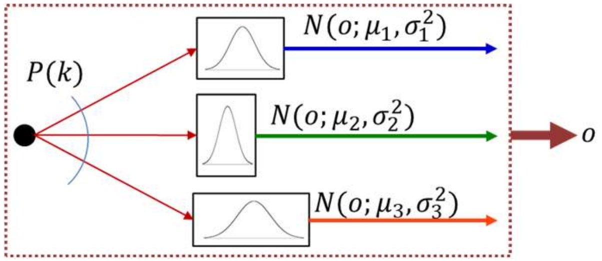
  - h represent the hidden or missing components

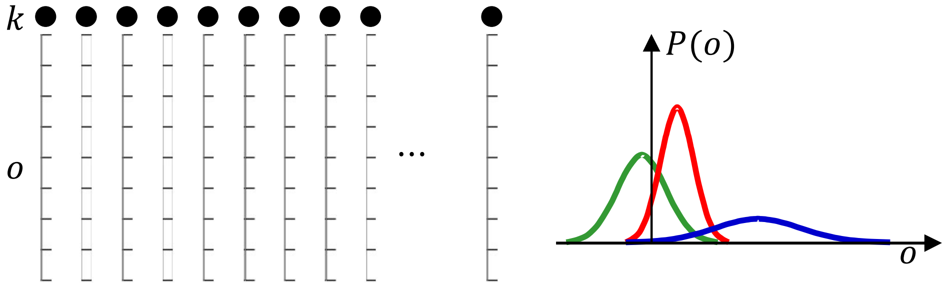
- Minimizing the log of a sum of ugly functions usually doesn't have nice solutions

Observation O consists of only the unblackened components



Blacked-out components are missing from data





Recap: The ELBO function

- We can define an *Empirical Lower Bound* (or ELBO) for the log probability $\log(P(o))$ as:

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

  - Holds for *any* probability distribution $Q(h)$
  - The bound is tightest when $Q(h) = P(h|o;\theta)$

- We get a nice iterative ML estimator if we maximize the ELBO instead of $\log(P(o))$ directly

The two-step process

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

- **Step 1:** Determine a $Q(h)$ that maximizes $L(\theta)$, fixing θ
  - Makes the bound tight

$$\hat{Q}(h) = \mathrm{argmax}_{Q(h)} \sum_h Q(h)\ \log \frac{P(h,o;\theta^{cur})}{Q(h)}$$

  - The$\hat{Q}(h)$ is a function of the *current* value of θ

The two-step process

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

- **Step 2:** Fix $Q(h) = \hat{Q}(h)$ and maximize $L(\theta)$ with respect to θ to get the next estimate of θ

$$\hat{\theta} = \mathrm{argmax}_{\theta} \sum_h \hat{Q}(h) \log P(h,o;\theta)$$

In summary

- Define the *auxiliary* function:

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

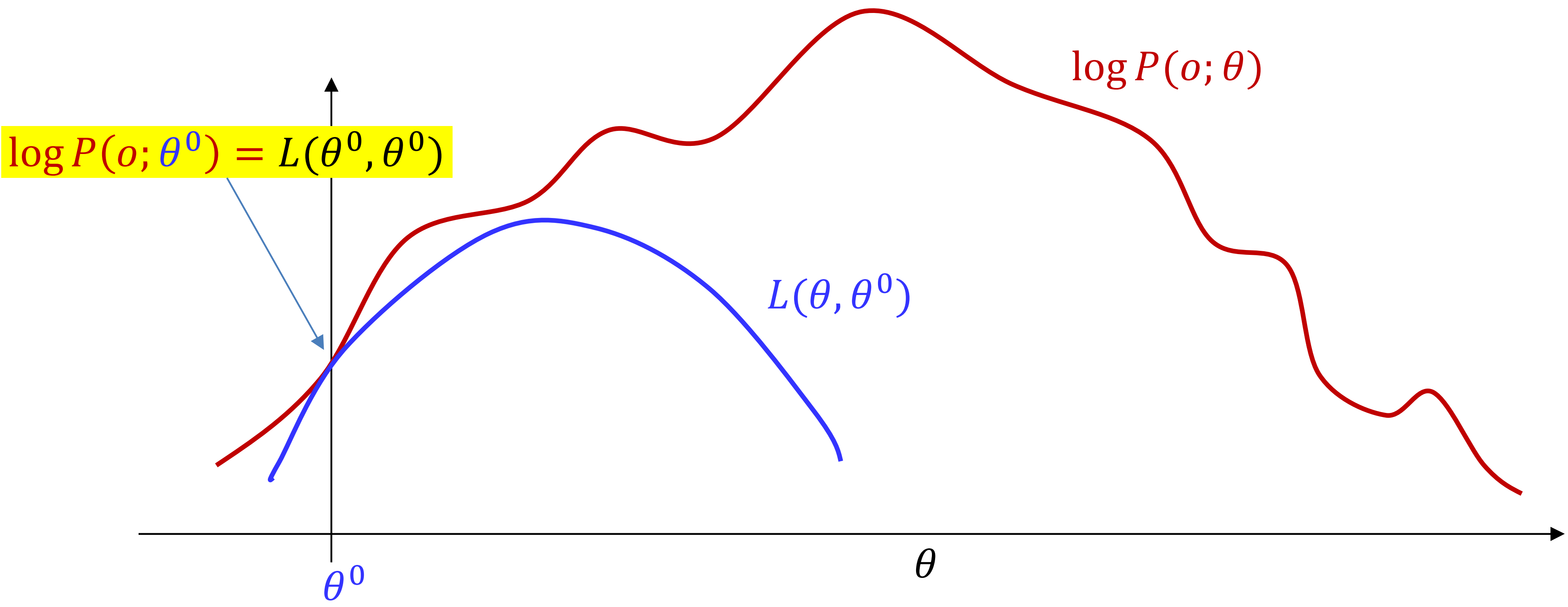
  - Which is the ELBO plus a term that doesn't depend on θ
  - For the optimal $\boldsymbol{Q(h; \theta^k) = P(h|o; \theta^k)}$ this is EM

- Iteratively compute

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

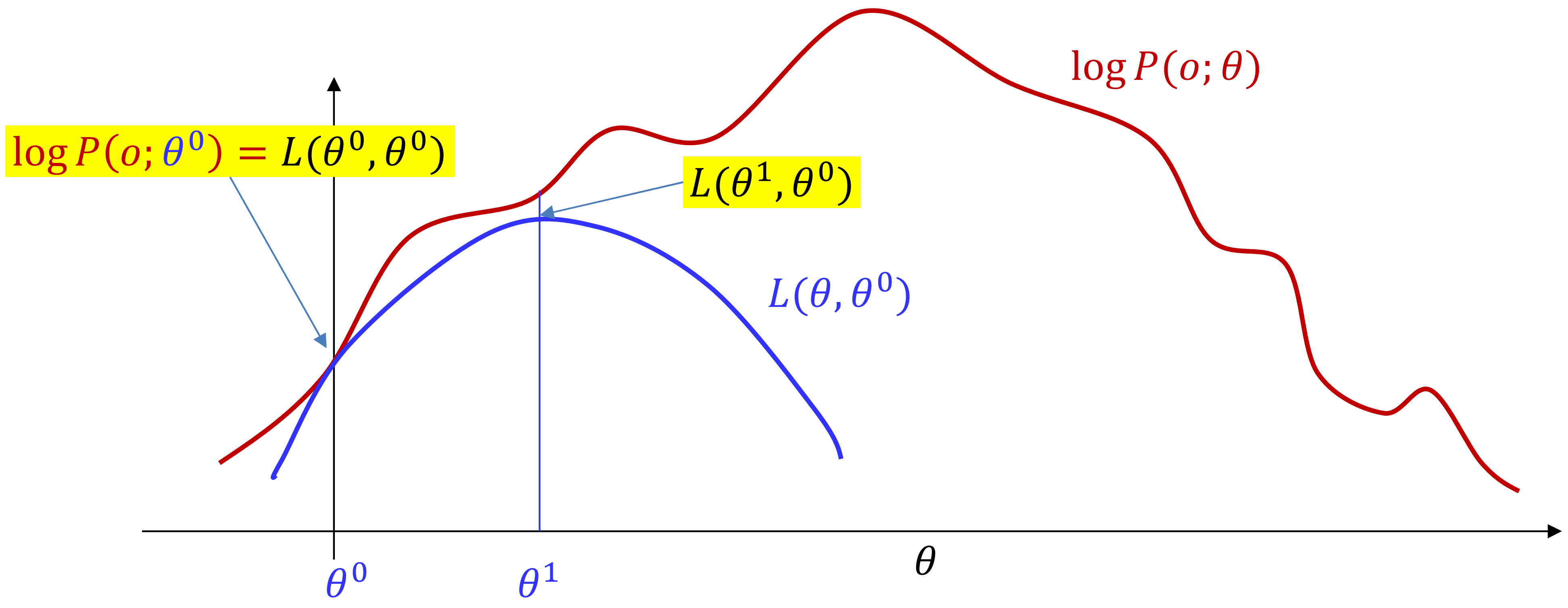
- Expected (or guaranteed) to increase $\log P(o)$ with every iteration

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



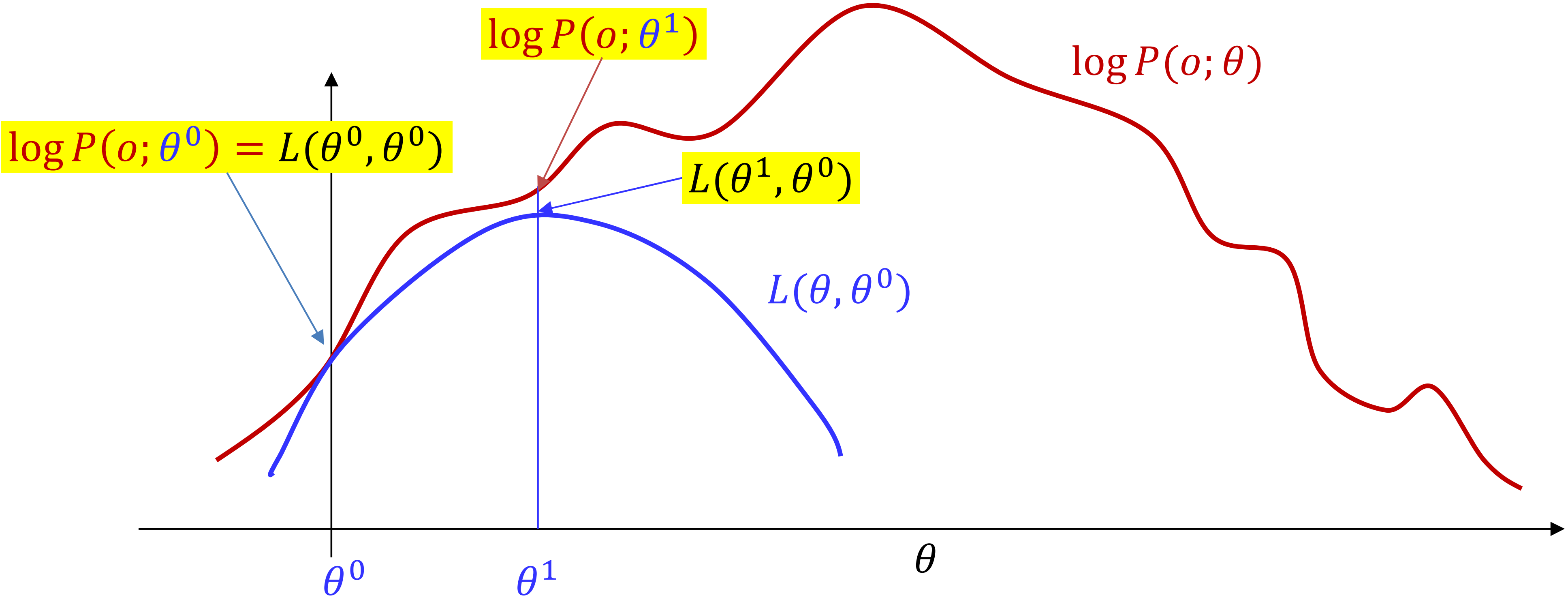
- Initialize θ^0
- Construct $L(\theta,\theta^0) = \sum_h Q(h;\theta^0)\ \log P(h,o;\theta)$
  - For EM it touches $\log P(o;\theta)$ at θ^0 because $\log P(o;\theta^0) = L(\theta^0,\theta^0)$
  - More generally, it tries to get as close to $\log P(o;\theta^0)$ as possible

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



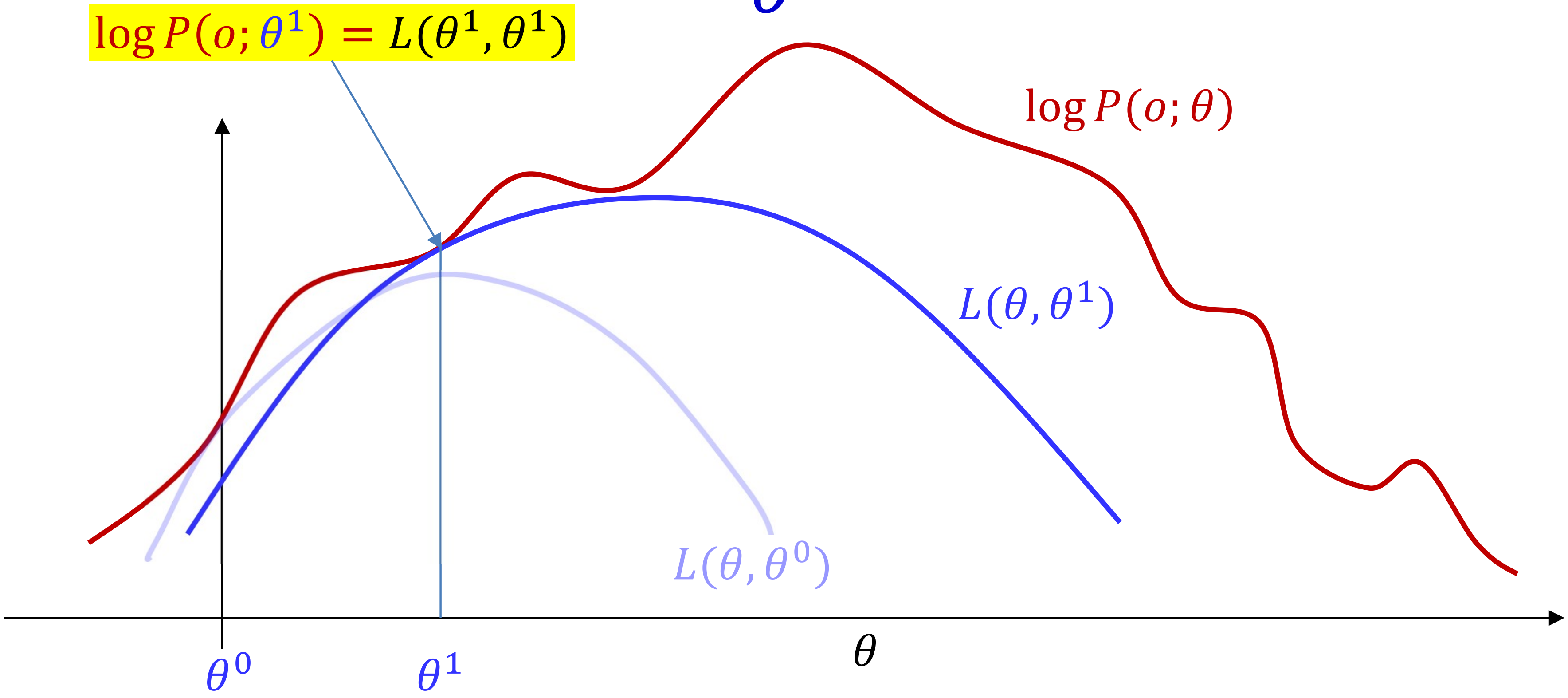
- Find $\theta^1 = \underset{\theta}{\operatorname{argmax}}\, L(\theta, \theta^0)$
  - $L(\theta^1, \theta^0) \geq L(\theta^0, \theta^0)$ (since you're maximizing $L(\theta, \theta^0)$ w.r.t θ)

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



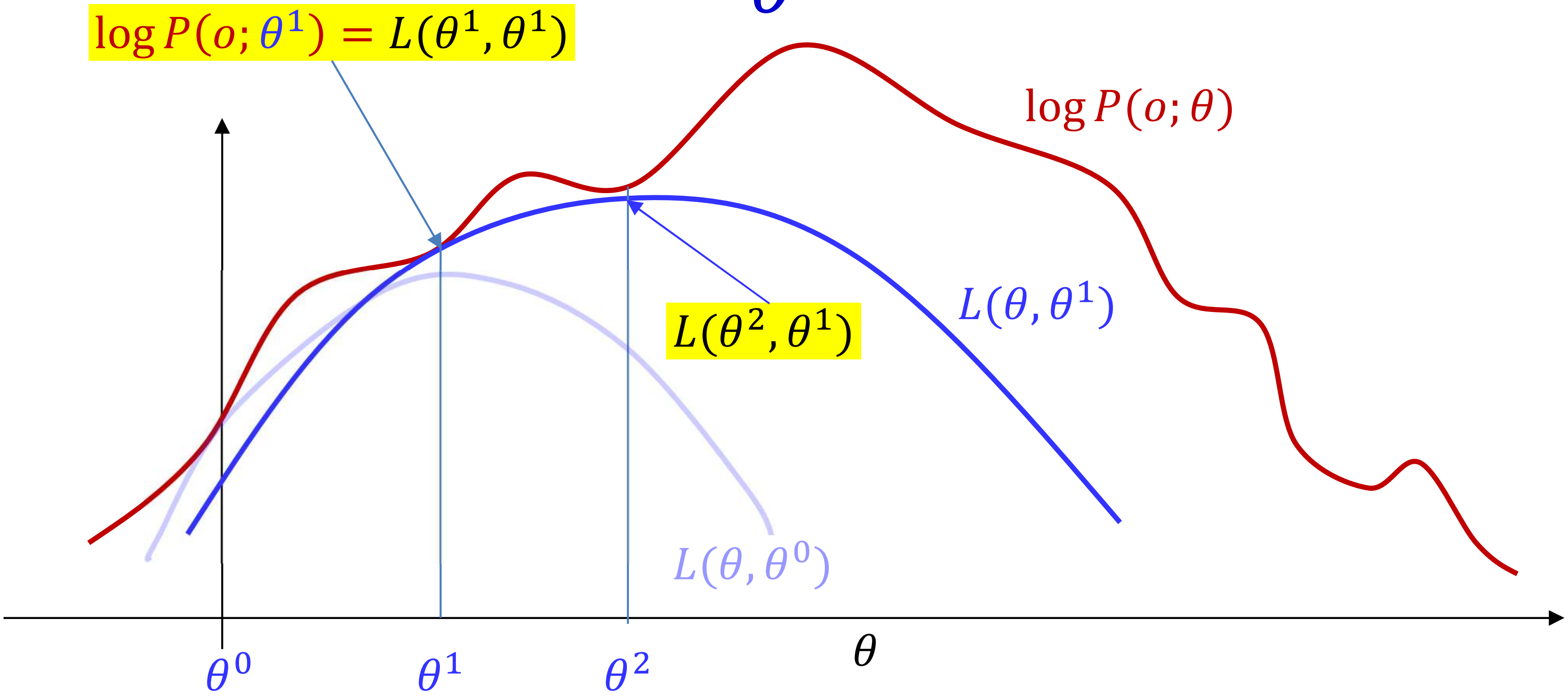
- Find $\theta^1 = \underset{\theta}{\operatorname{argmax}}\, L(\theta, \theta^0)$
  - $L(\theta^1,\theta^0) \geq L(\theta^0,\theta^0)$ (since you're maximizing $L(\theta,\theta^0)$ w.r.t θ)
- $\log P(o;\theta^1) \geq L(\theta^1,\theta^0)$
  - since $L(\theta,\theta^0)$ is a lower bound on $\log P(o;\theta)$
- So the iteration may be expected to increase $\log P(o;\theta)$
  - Guaranteed non-decreasing for EM

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



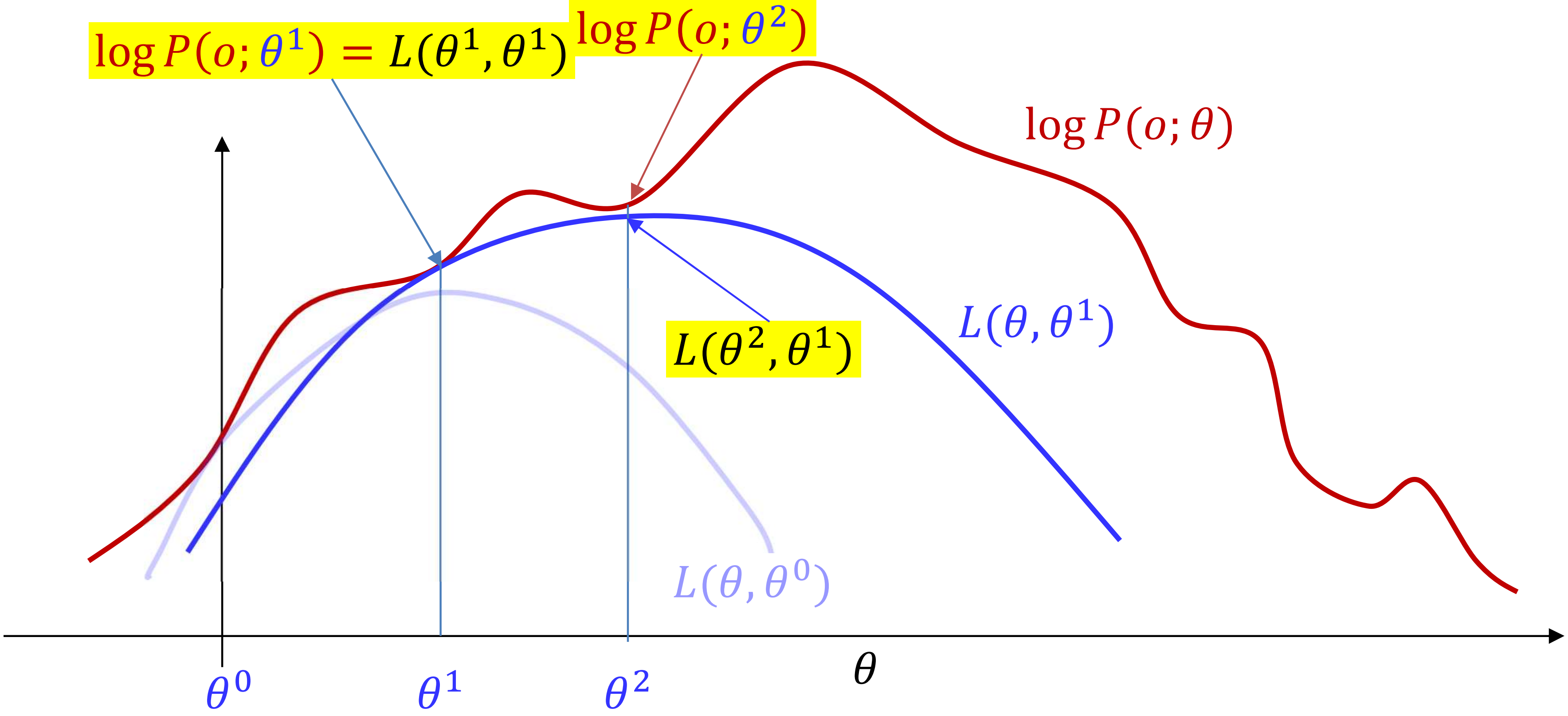
- Construct $L(\theta, \theta^1)$
  - – It gets close to (or touches) $\log P(o; \theta)$ at θ^1
    - Because the $Q(h; \theta^1)$ is now a function of θ^1

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



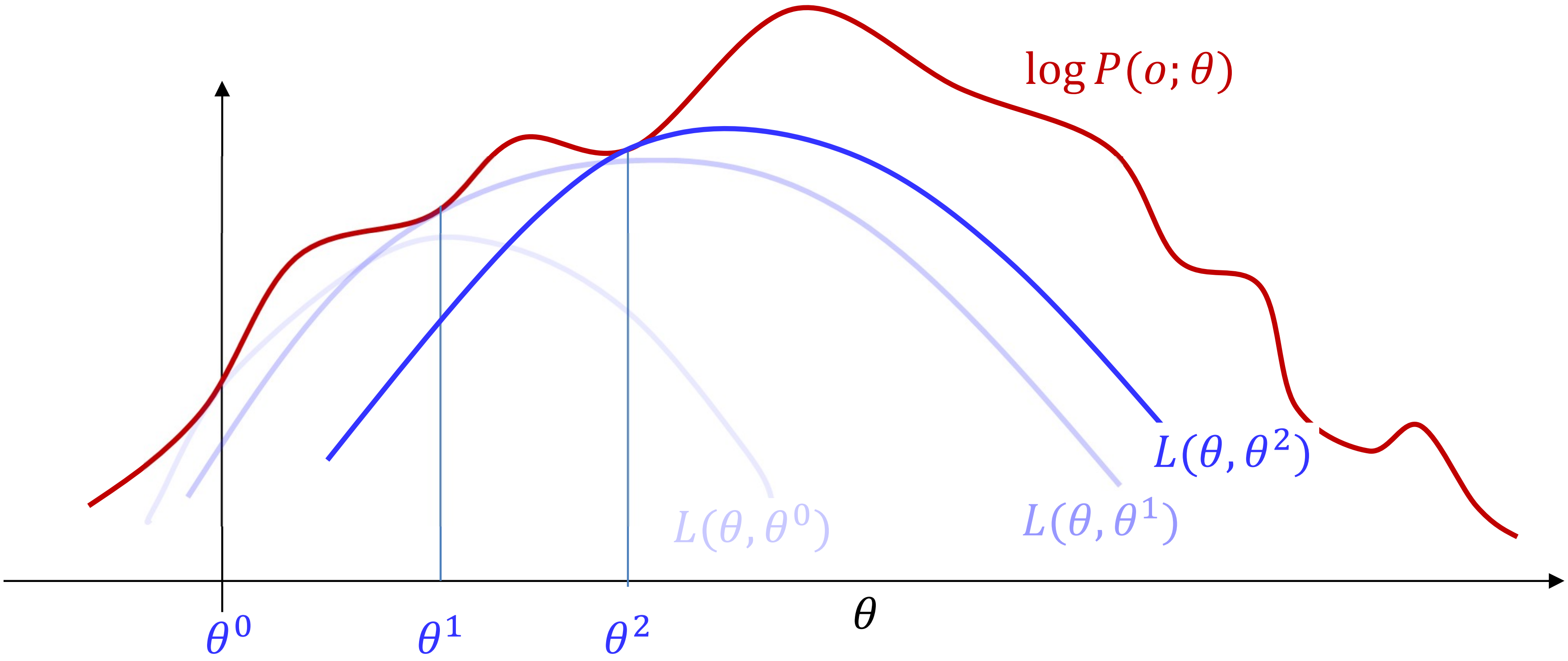
- Find $\theta^2 = \underset{\theta}{\operatorname{argmax}}\, L(\theta,\theta^1)$
  - $L(\theta^2,\theta^1) \geq L(\theta^1,\theta^1)$ (since you're maximizing $L(\theta,\theta^1)$ w.r.t θ)

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



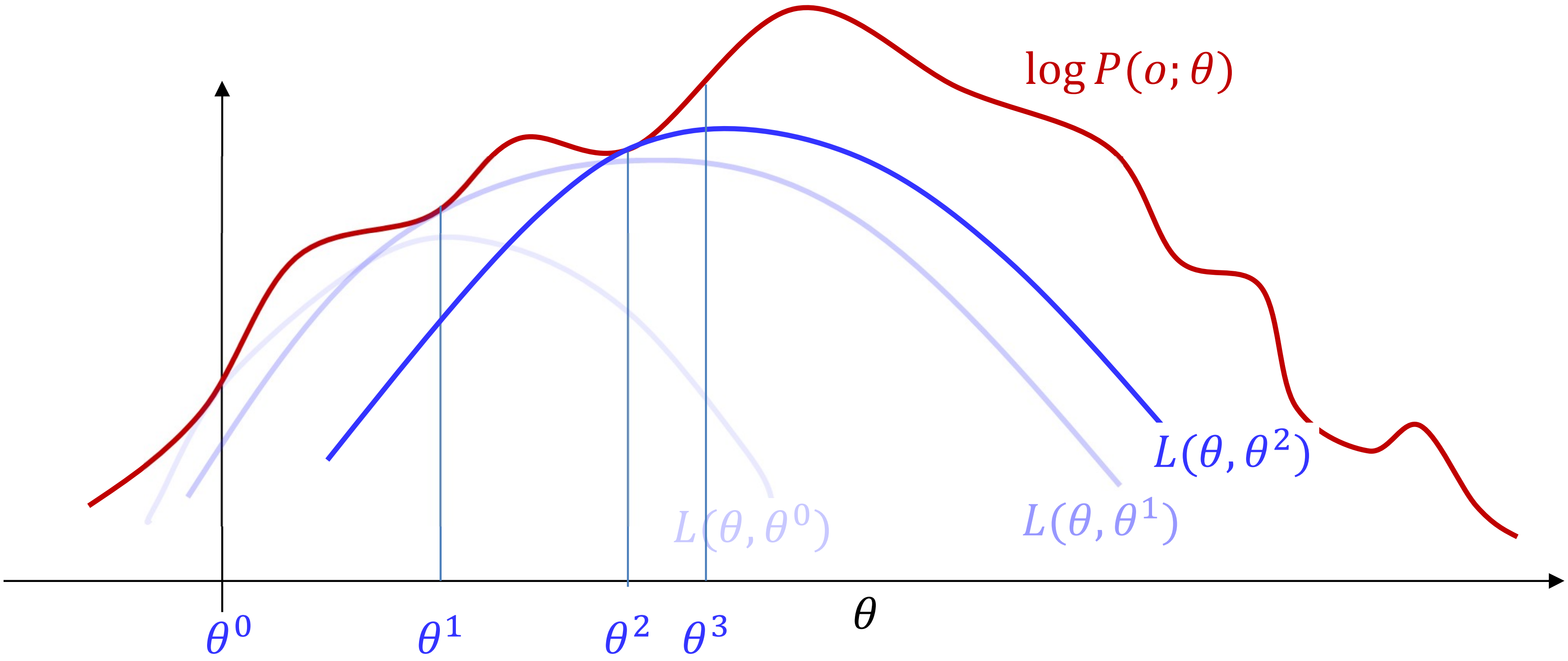
- Find $\theta^2 = \underset{\theta}{\operatorname{argmax}}\, L(\theta,\theta^1)$
  - $L(\theta^2,\theta^1) \geq L(\theta^1,\theta^1)$ (since you're maximizing $L(\theta,\theta^1)$ w.r.t θ)
- Ideally $\log P(o;\theta^2) \geq L(\theta^2,\theta^1)$
  - Since $L(\theta,\theta^1)$ is a lower bound on $\log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

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



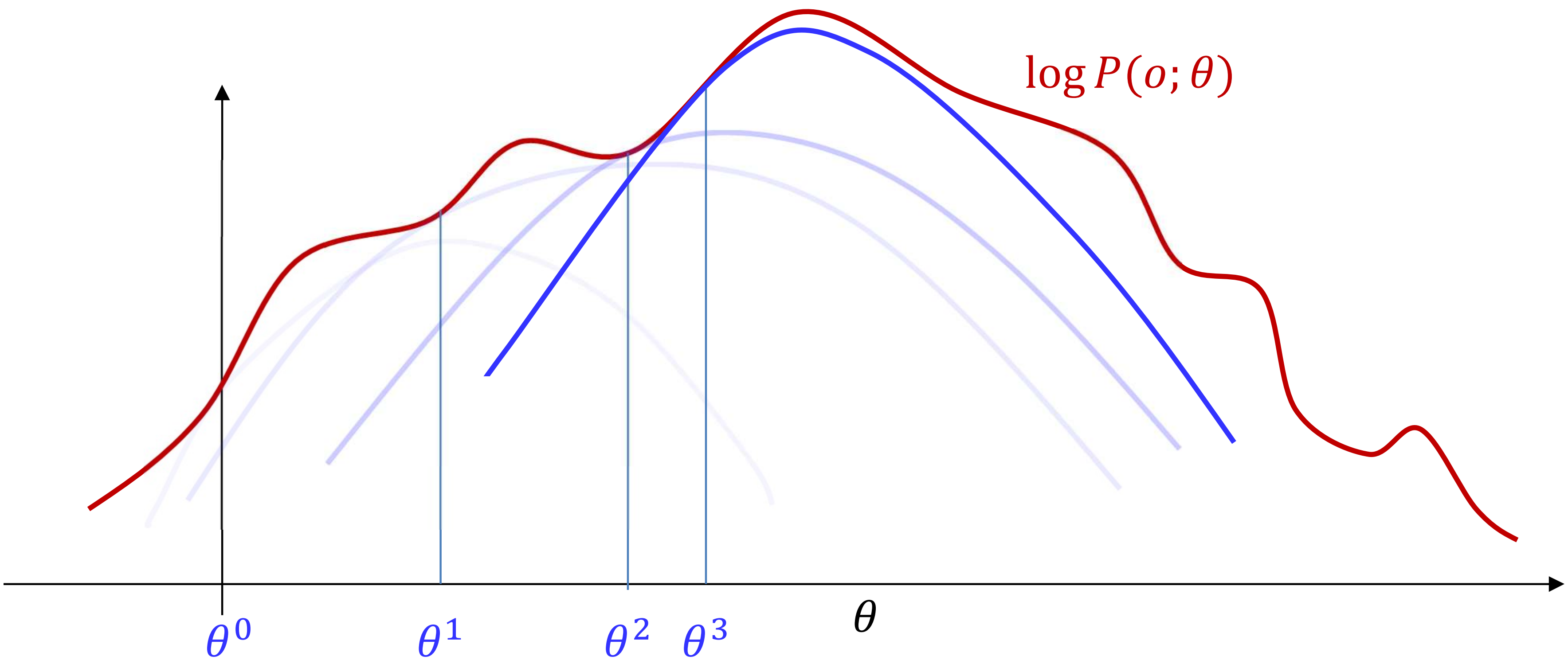
- Repeat the steps:
  - Compose $L(\theta, \theta^k)$ to “touch” $\log P(o;\theta)$ at the current estimate θ^k
  - Set $\theta^{k+1} \leftarrow \underset{\theta}{\operatorname{argmax}}\, L(\theta, \theta^k)$
- Each step is guaranteed to increase (or at least not decrease) $\log P(o;\theta)$
  - Stop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

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



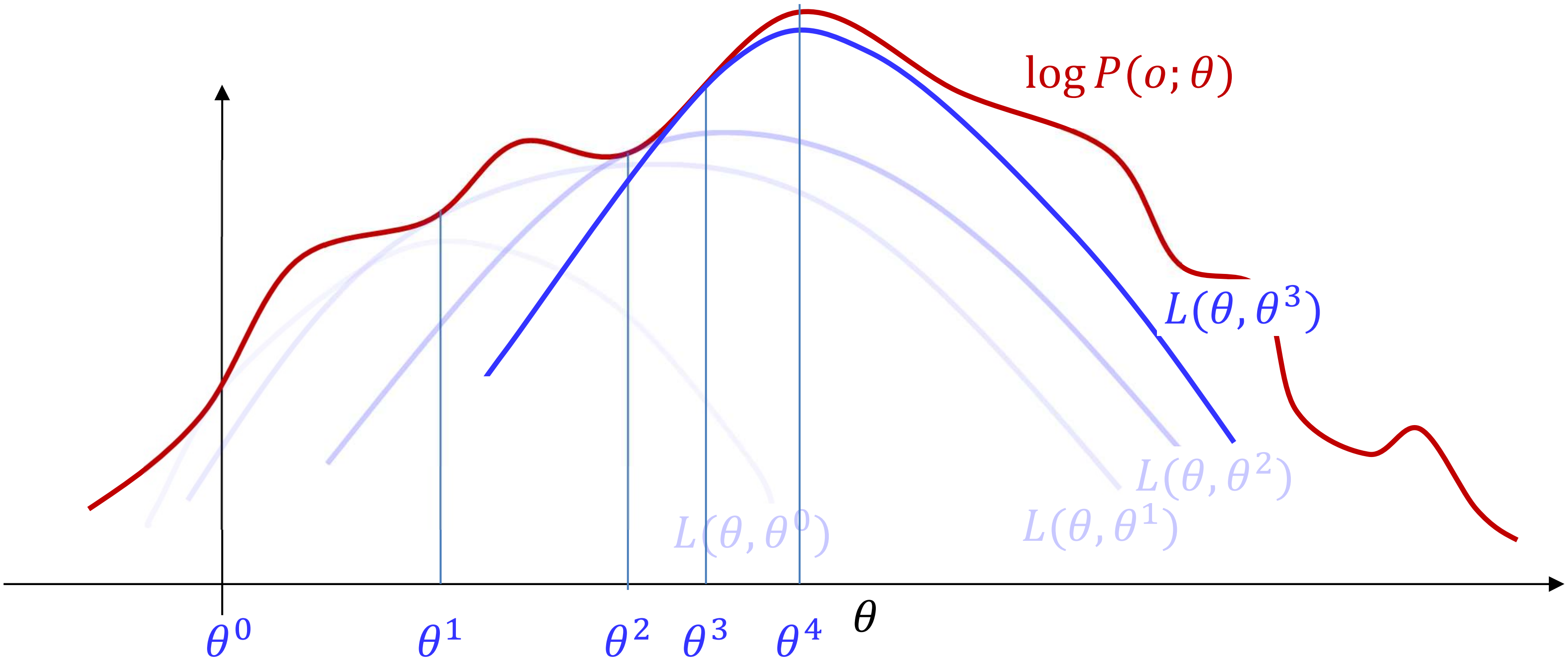
- Repeat the steps:
  - Compose $L(\theta, \theta^k)$ to “touch” $\log P(o;\theta)$ at the current estimate θ^k
  - Set $\theta^{k+1} \leftarrow \underset{\theta}{\operatorname{argmax}}\, L(\theta, \theta^k)$
- Each step is guaranteed to increase (or at least not decrease) $\log P(o;\theta)$
  - Stop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

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



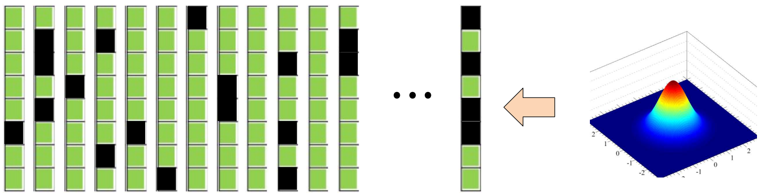
- Repeat the steps:
  - Compose $L(\theta, \theta^k)$ to “touch” $\log P(o;\theta)$ at the current estimate θ^k
  - Set $\theta^{k+1} \leftarrow \underset{\theta}{\operatorname{argmax}} L(\theta, \theta^k)$
- Each step is guaranteed to increase (or at least not decrease) $\log P(o;\theta)$
  - Stop when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

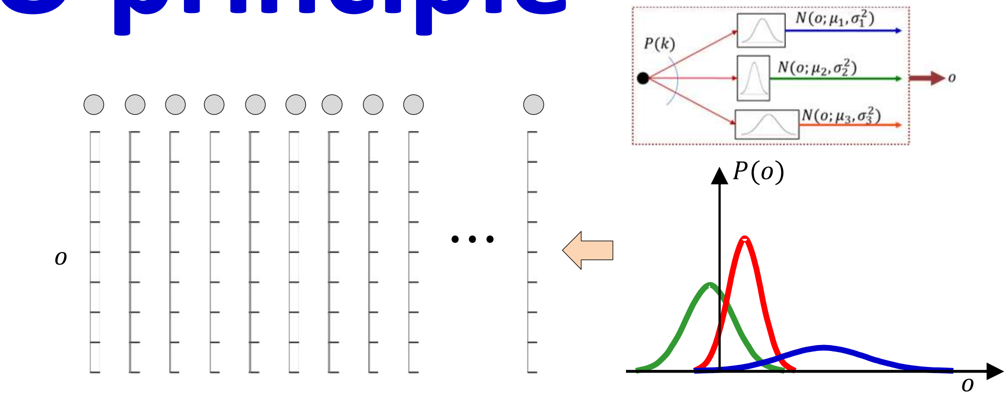
$$\theta^{k+1} \leftarrow \underset{\theta}{\operatorname{argmax}}\, \boldsymbol{L}(\theta, \theta')$$



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

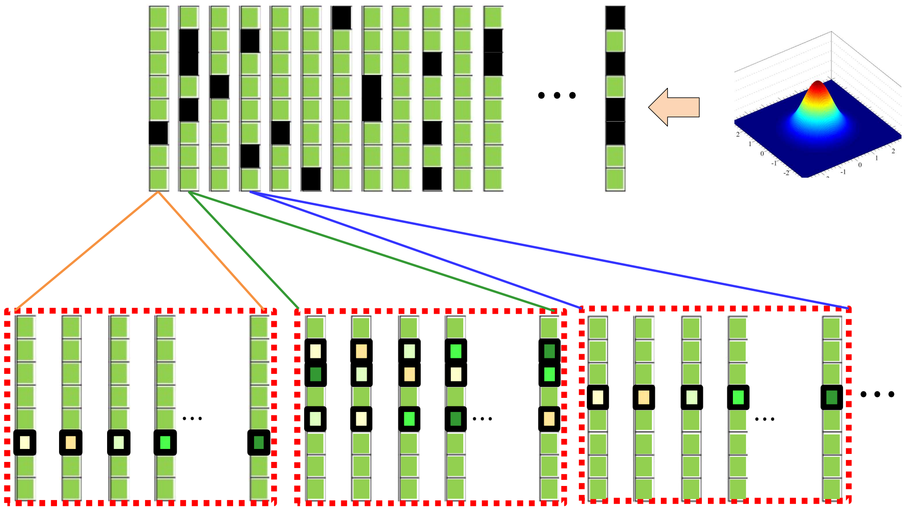
Recap: ELBO principle

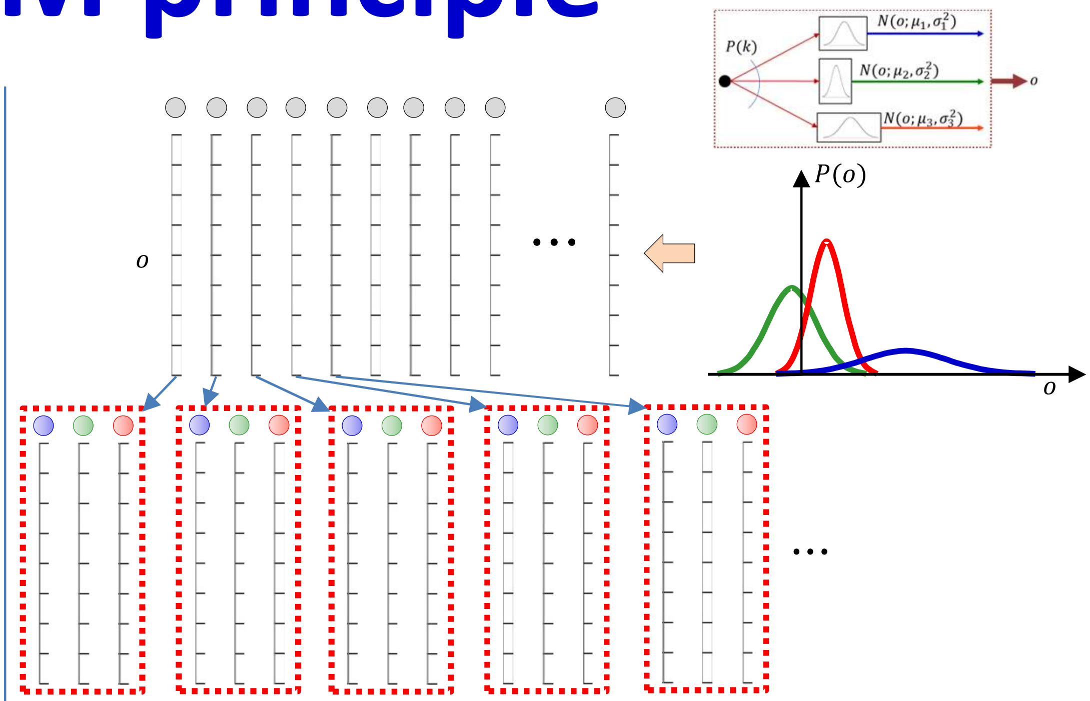




- Problem: Train generative model using incomplete data
  - Gaussian from vectors with missing components
  - GMMs, where the Gaussian that generated each observation is unknown
  - Etc…
- ELBO approach: “Complete” the data by “filling in” the missing components
  - Estimate models from “completed” data
  - Question: How to “complete” data

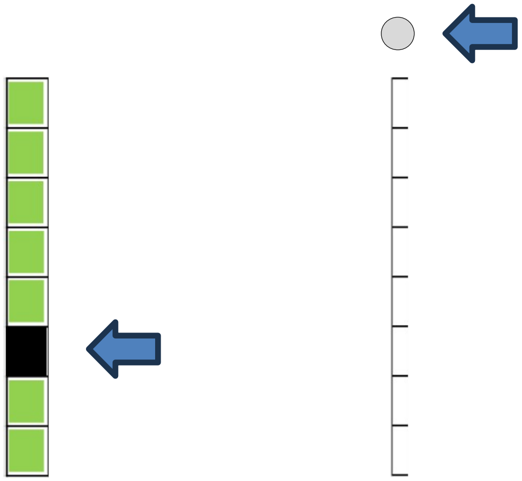
Recap: EM principle





- Iteratively:
- **Complete the data according to $Q(h)$, which approximates the posterior probabilities $P(m|o)$ computed by the current model**
  - By explicitly considering every possible value, with its posterior-based proportionality
  - Or by sampling the posterior probability distribution $Q(h)$
    - Upon completion each incomplete observation implicitly or explicitly becomes many (potentially infinite) complete observations
- ***Reestimate the model from completed data***

Re-Recap : What are we *really doing*



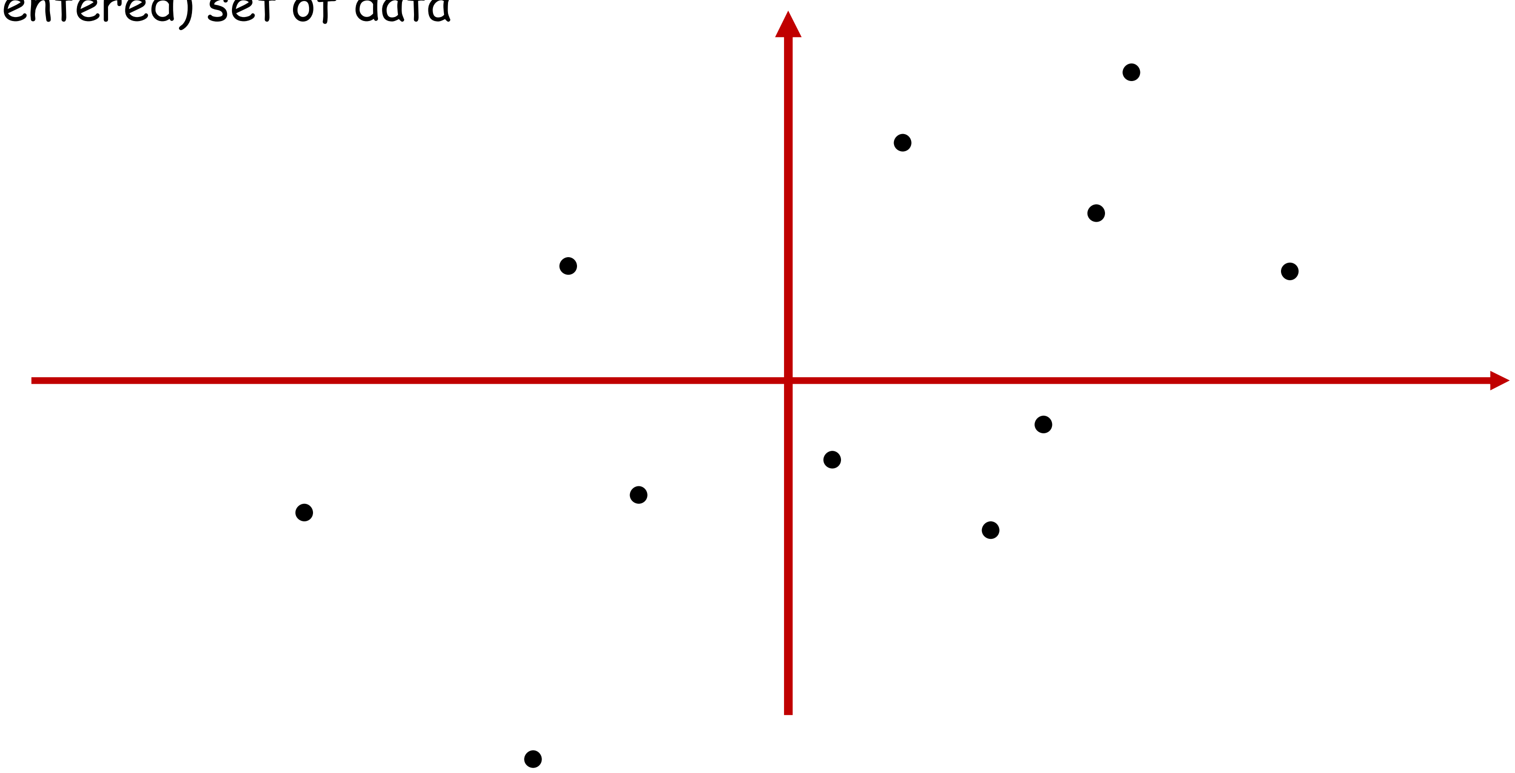
- Since key variables are missing, we *fill them up* by guessing their value and “complete” the data
  - Then estimate whatever statistic we wish to estimate

- “Filling them up”:
  - Sample from P(h|o), or some approximation Q(h) to P(h|o)
  - Take the expected value of the statistic over *all* values of h, with distribution P(h|o) (or Q(h))

Questions?

Principal Component Analysis

Given a (centered) set of data

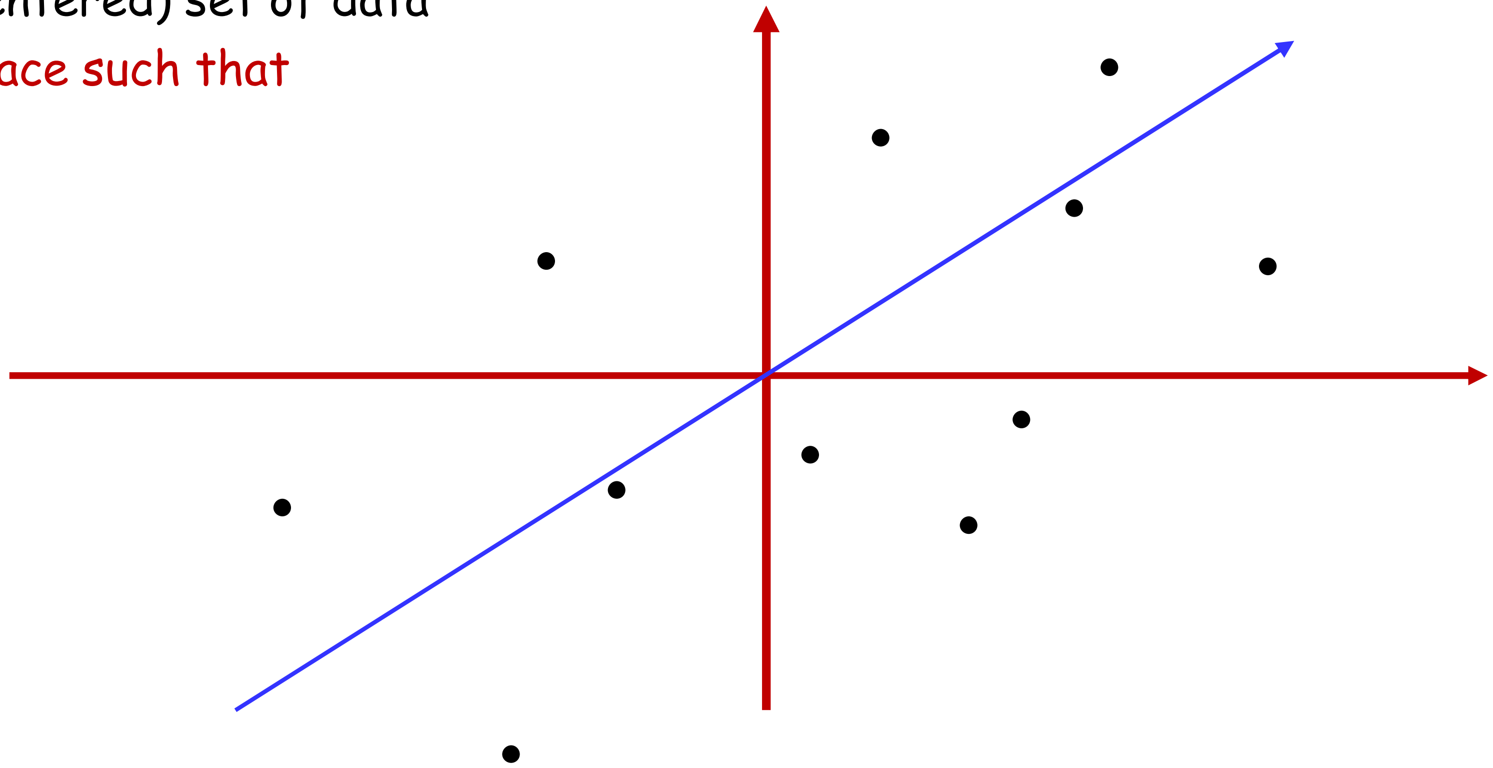


- Find the principal subspace such that when all vectors are approximated as lying on that subspace, the approximation error is minimal
  - Assuming "centered" (zero-mean) data

Principal Component Analysis

Given a (centered) set of data

find subspace such that

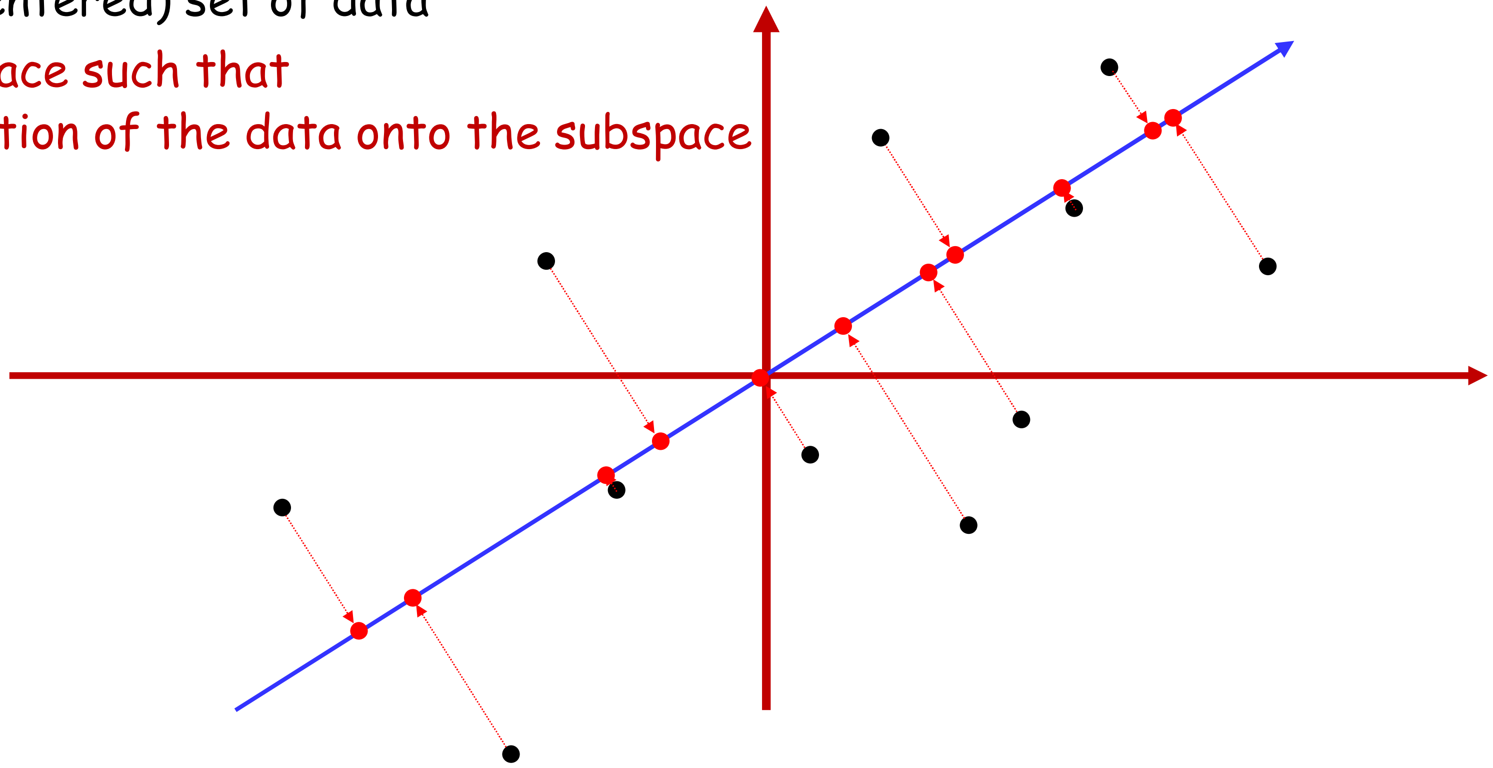


- Find the principal subspace such that when all vectors are approximated as lying on that subspace, the approximation error is minimal
  - Assuming "centered" (zero-mean) data

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

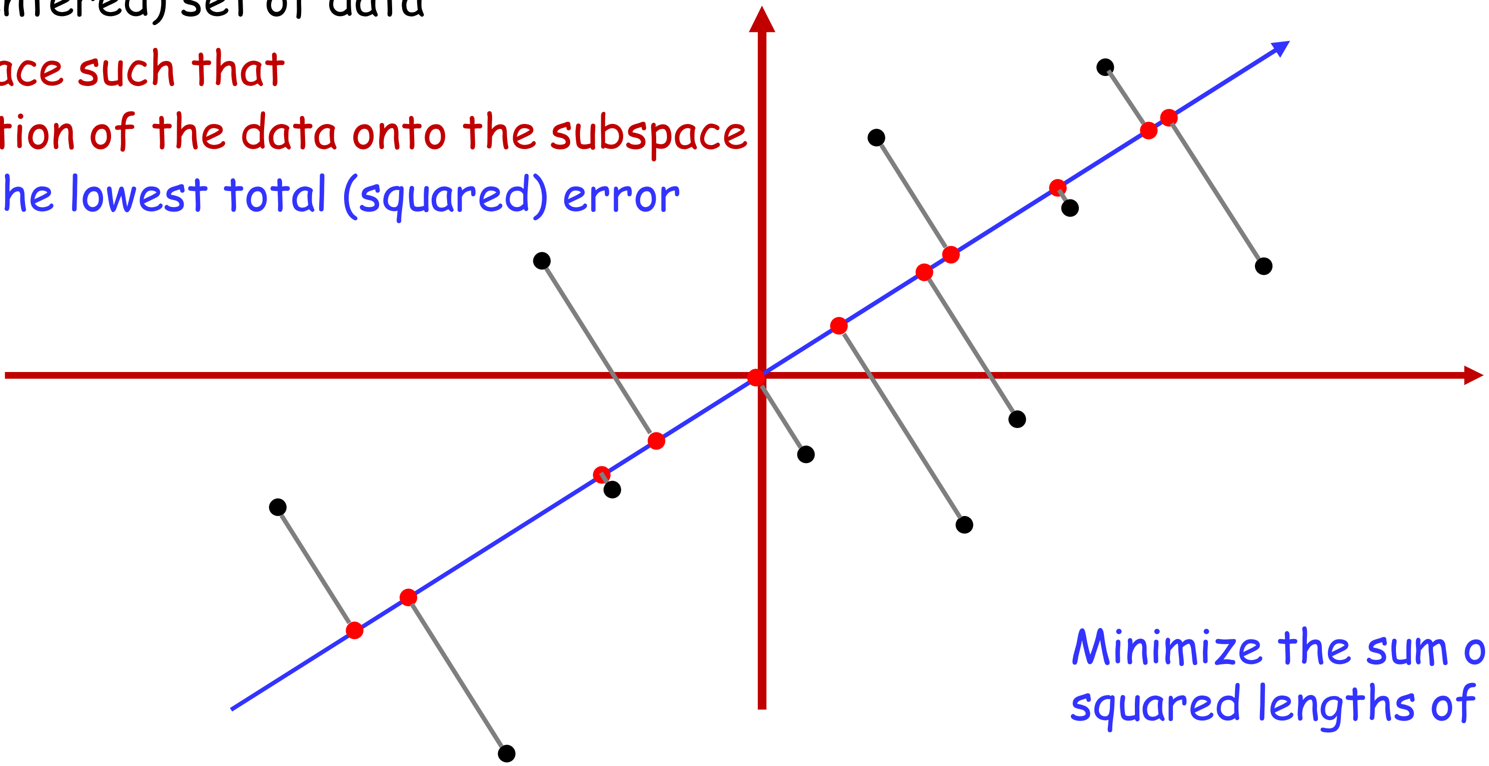
Principal Component Analysis

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

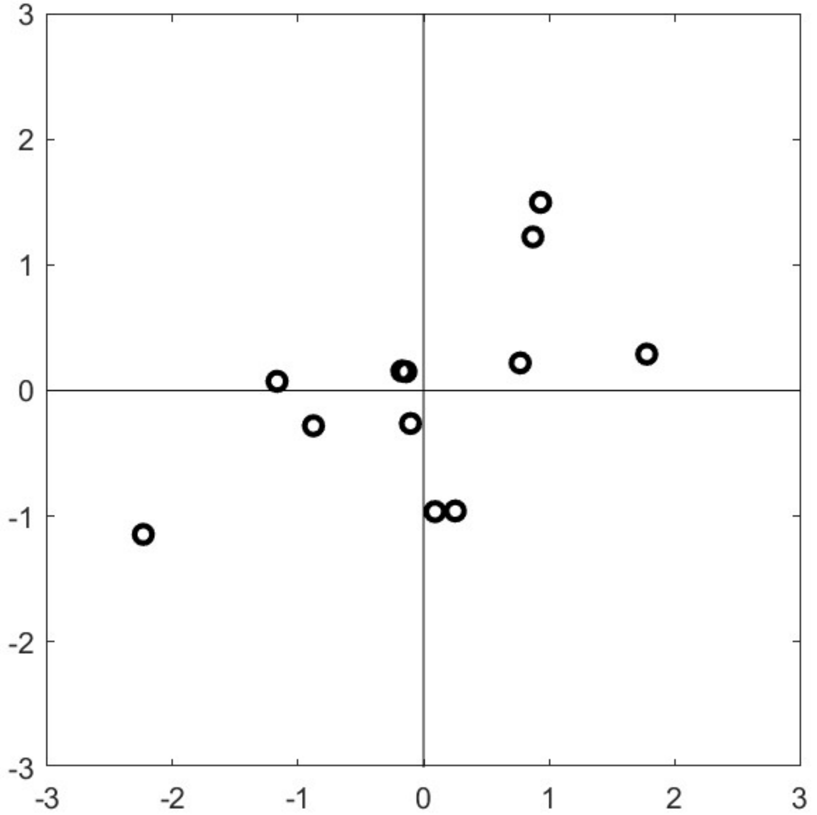


Minimize the sum of the squared lengths of these lines

- 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

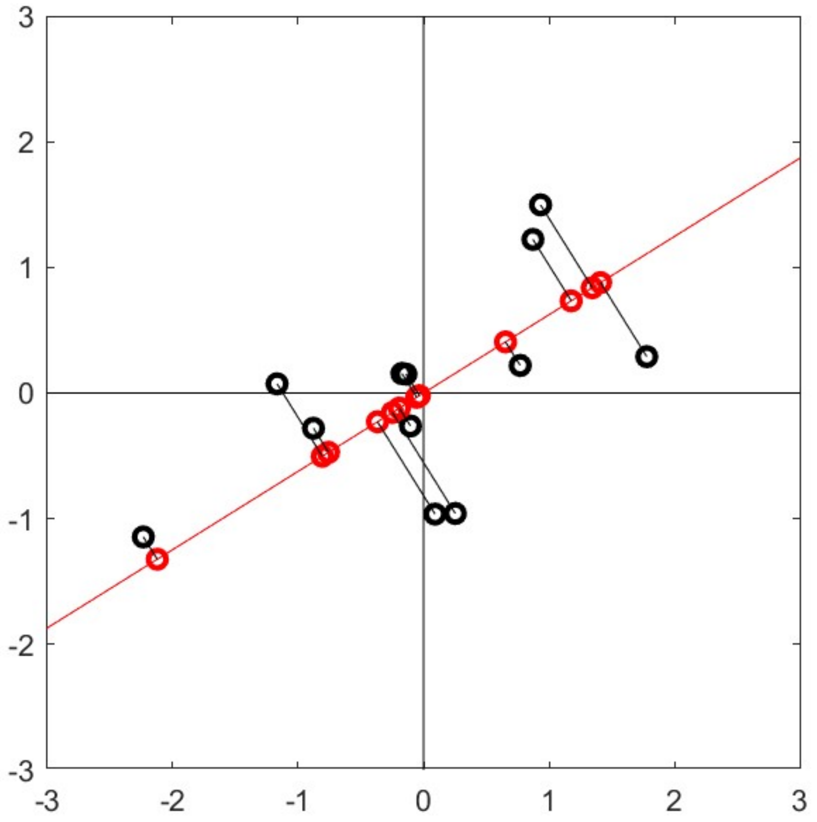


- Find the principal subspace such that when all vectors are approximated as lying on that subspace, the approximation error is minimal
  - Assuming “centered” (zero-mean) data

Principal Component Analysis

Animation:
Original centered data

Principal axis we're searching for

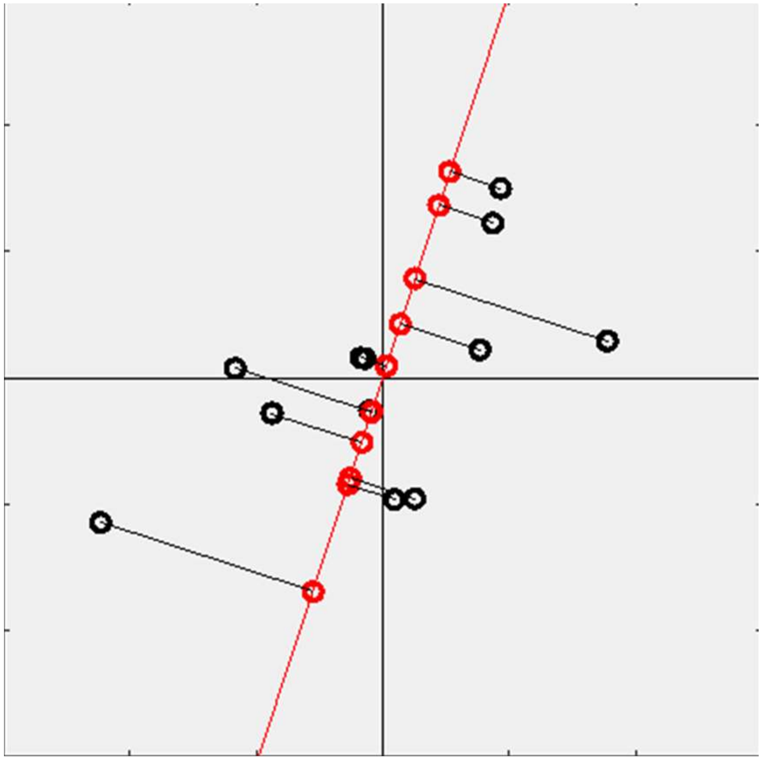


- Find the principal subspace such that when all vectors are approximated as lying on that subspace, the approximation error is minimal
  - Assuming "centered" (zero-mean) data

Principal Component Analysis

Animation:
Original centered data

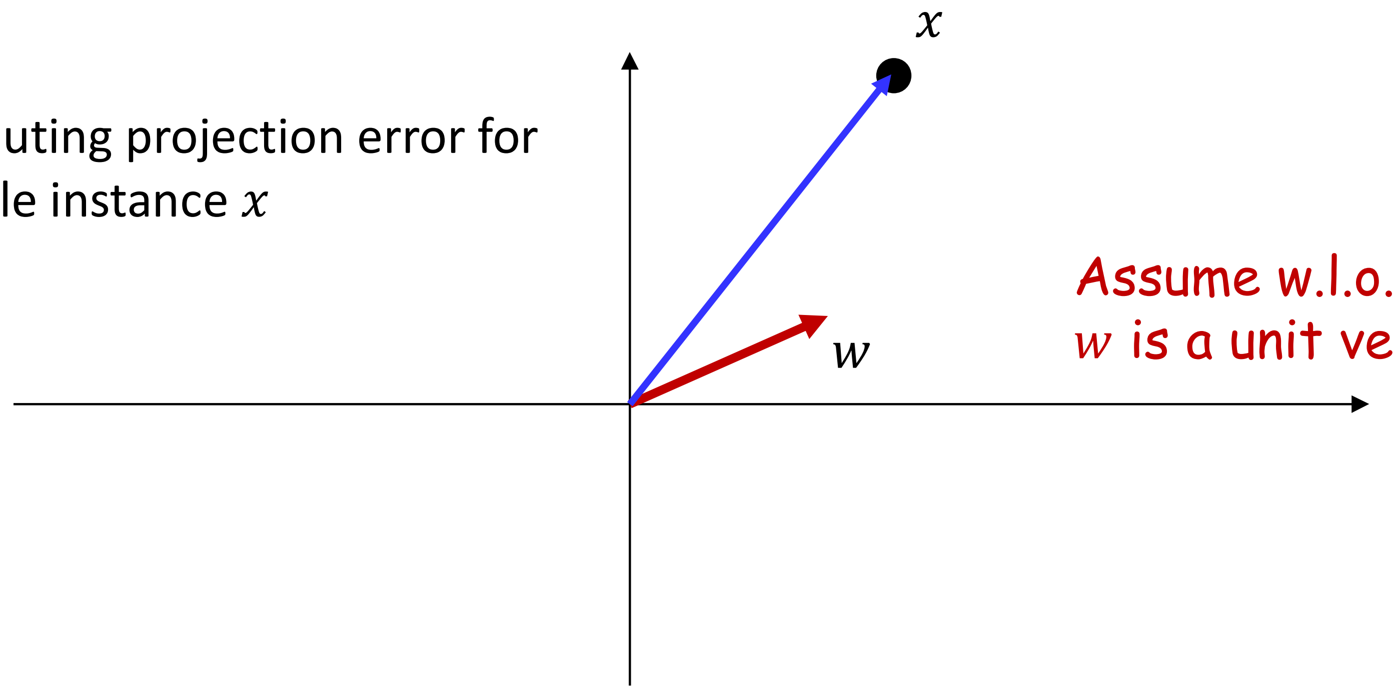
Principal axis we're searching for



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



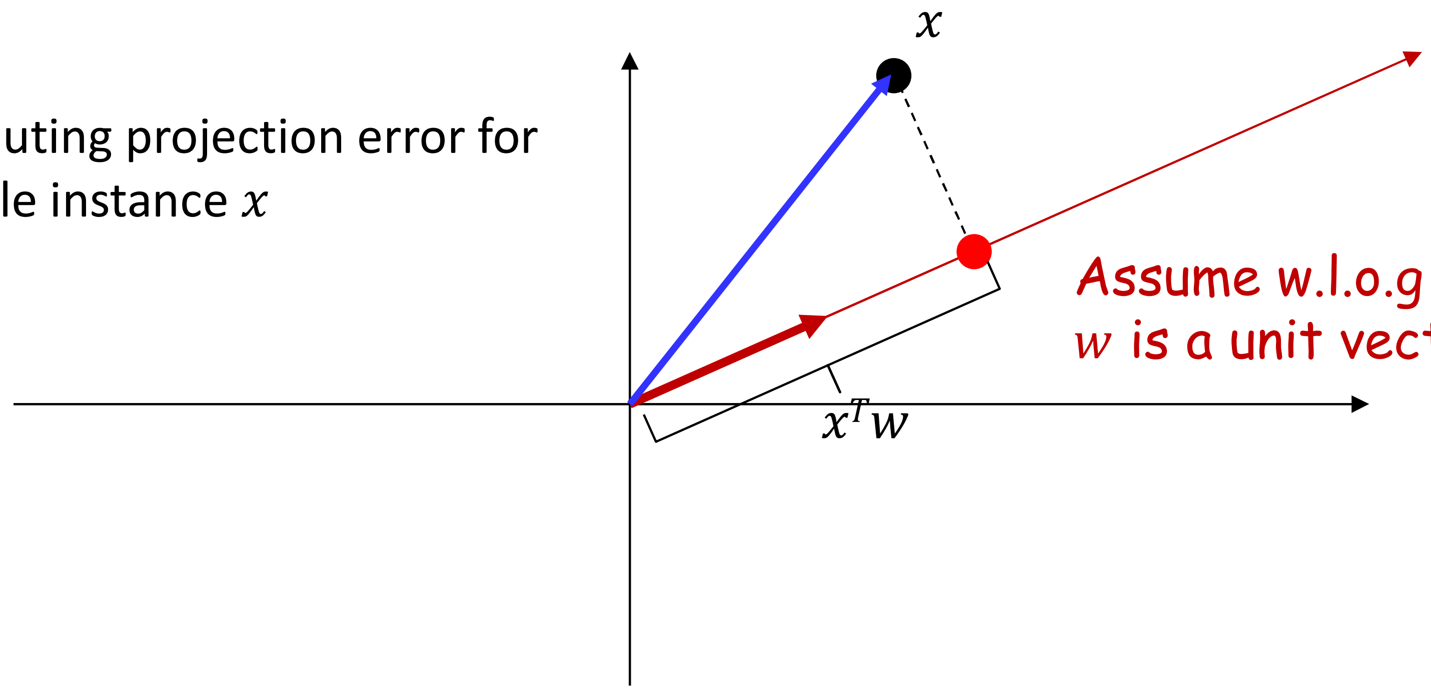
Computing projection error for a single instance x

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

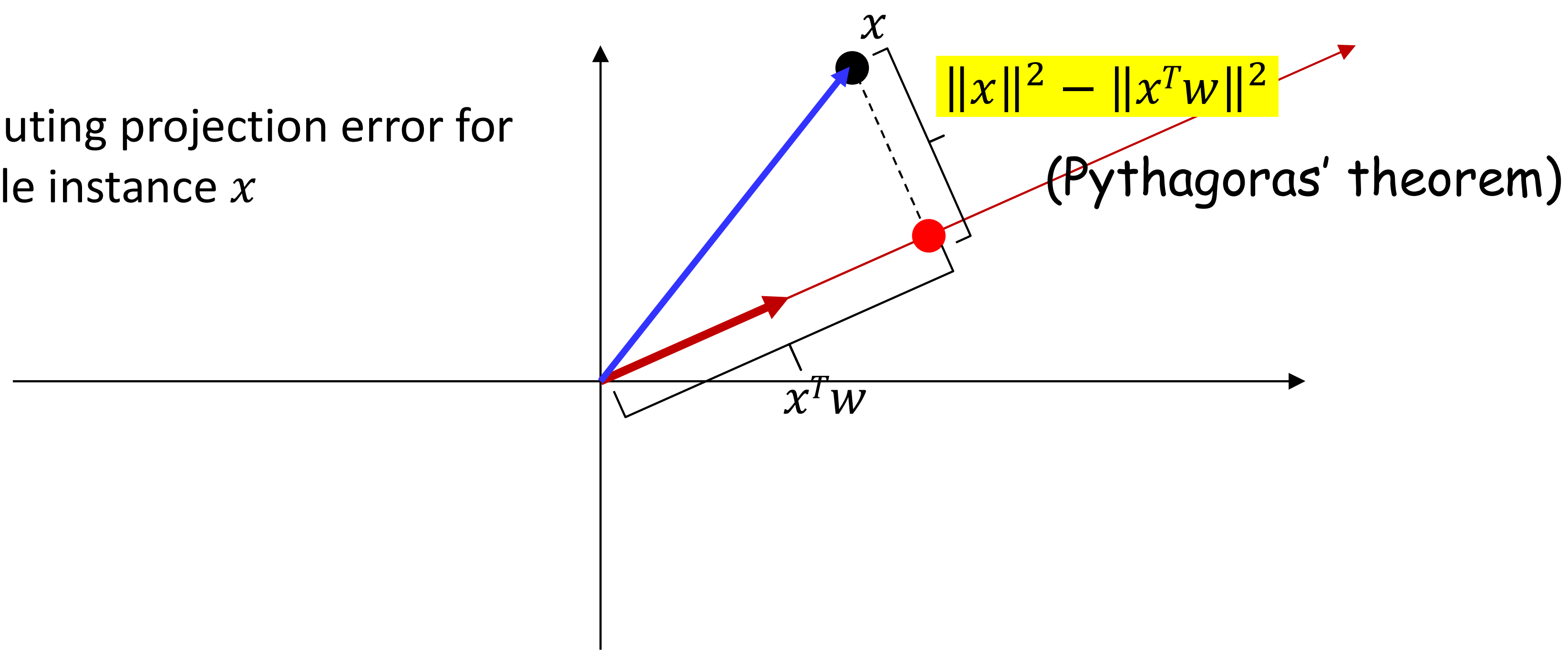


Assume w.l.o.g that w is a unit vector

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

Can be done in closed form

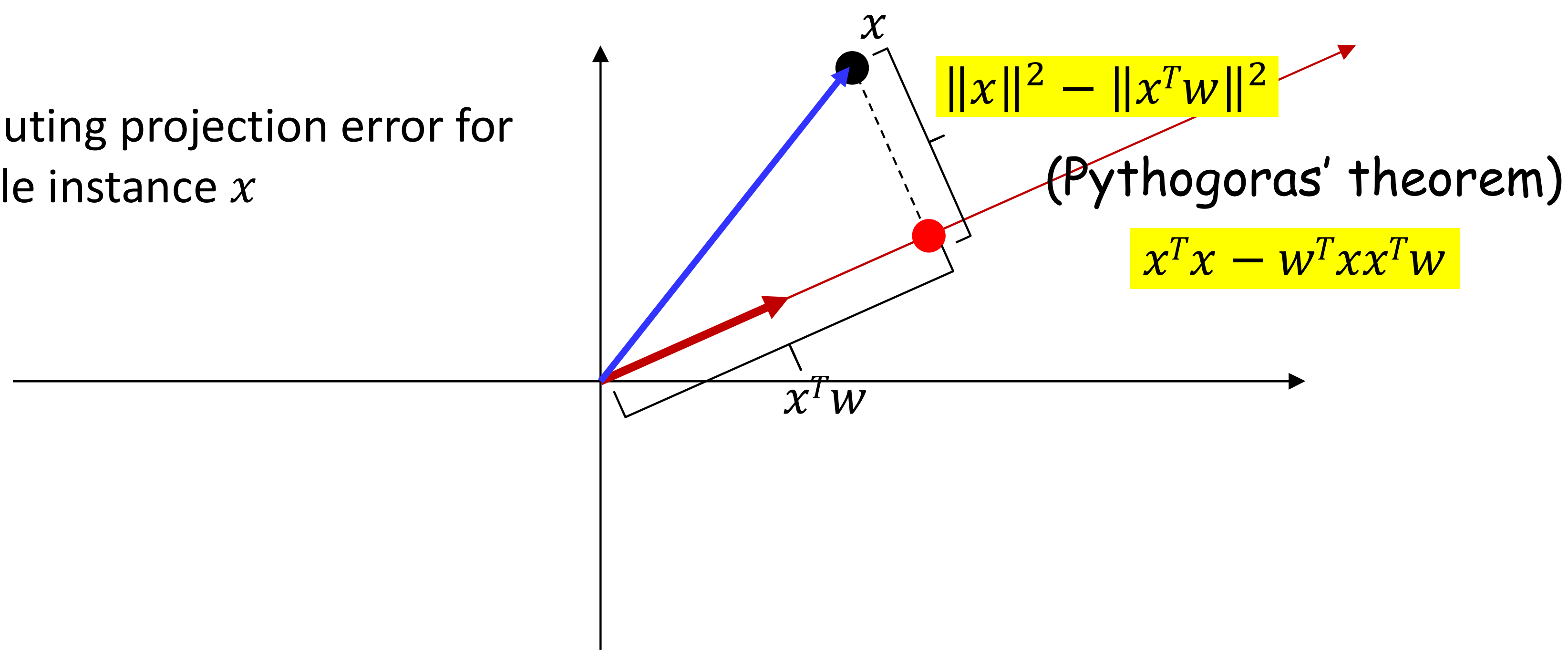
Computing projection error for a single instance x



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

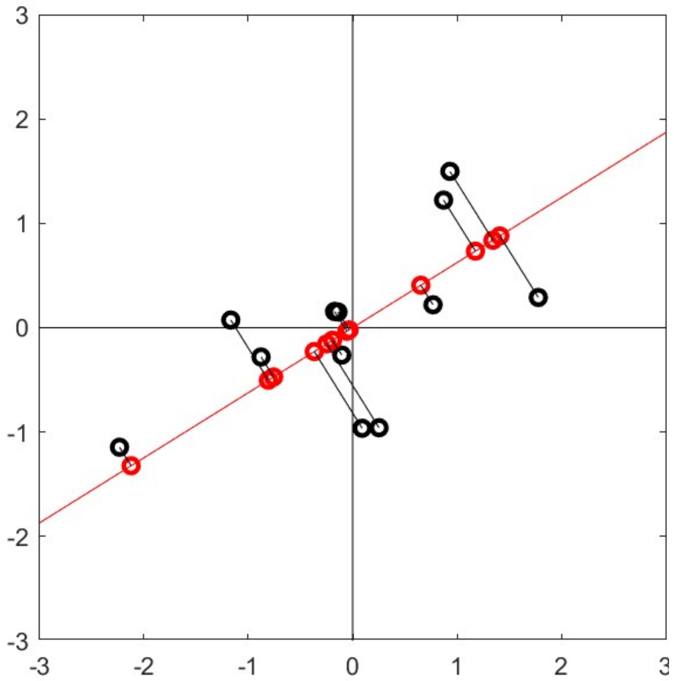
Can be done in closed form

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



- Since we're minimizing quadratic L$_2$ error, we can find a closed form solution
- Total projection error for all data:

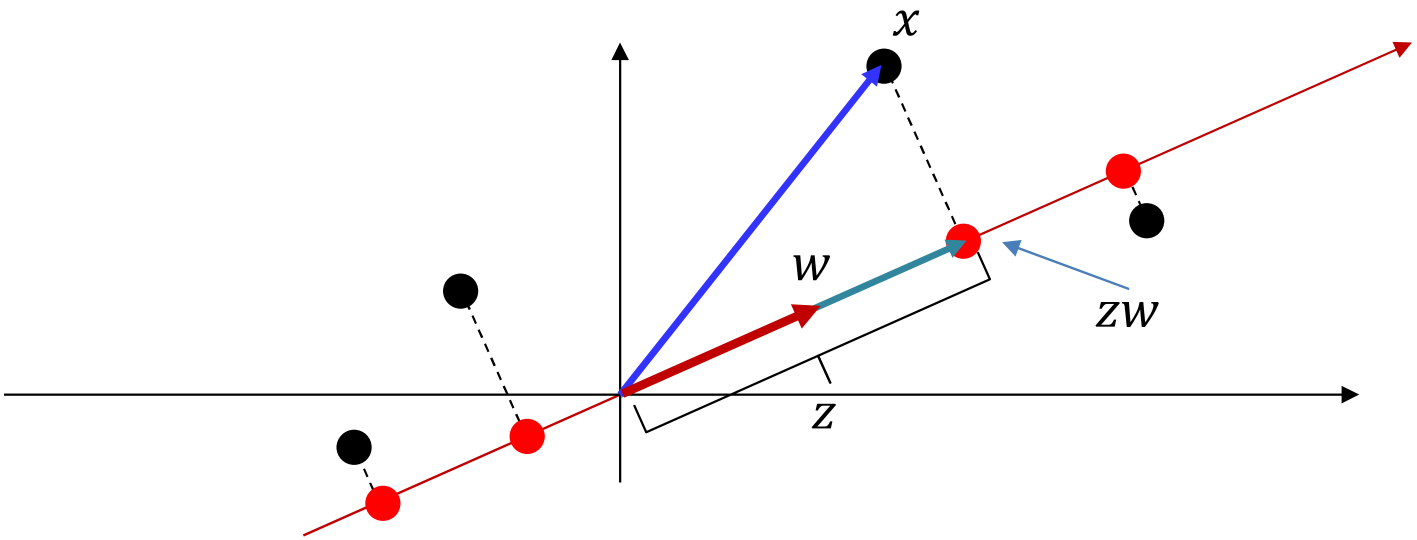
$$L = \sum_x x^T x - w^T x x^T w$$

- 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

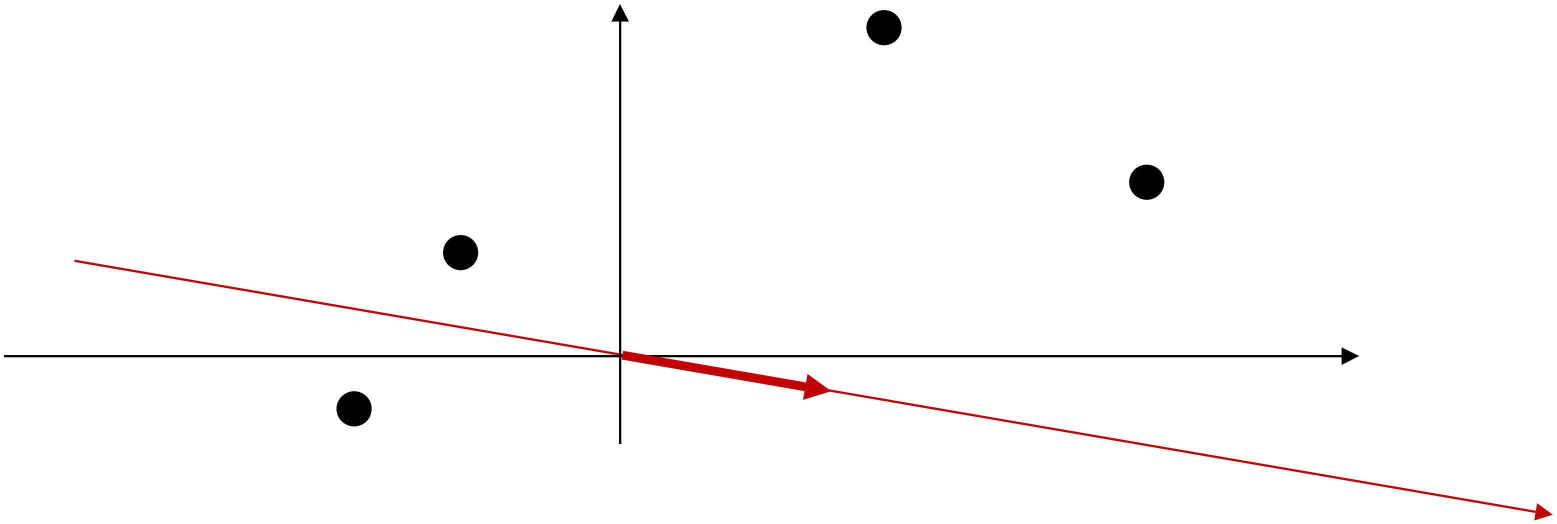


- Objective: find a vector (subspace) w and a *position* z on w such that $zw \approx x$ most closely (in an L_2 sense) for the entire (training) data

- 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

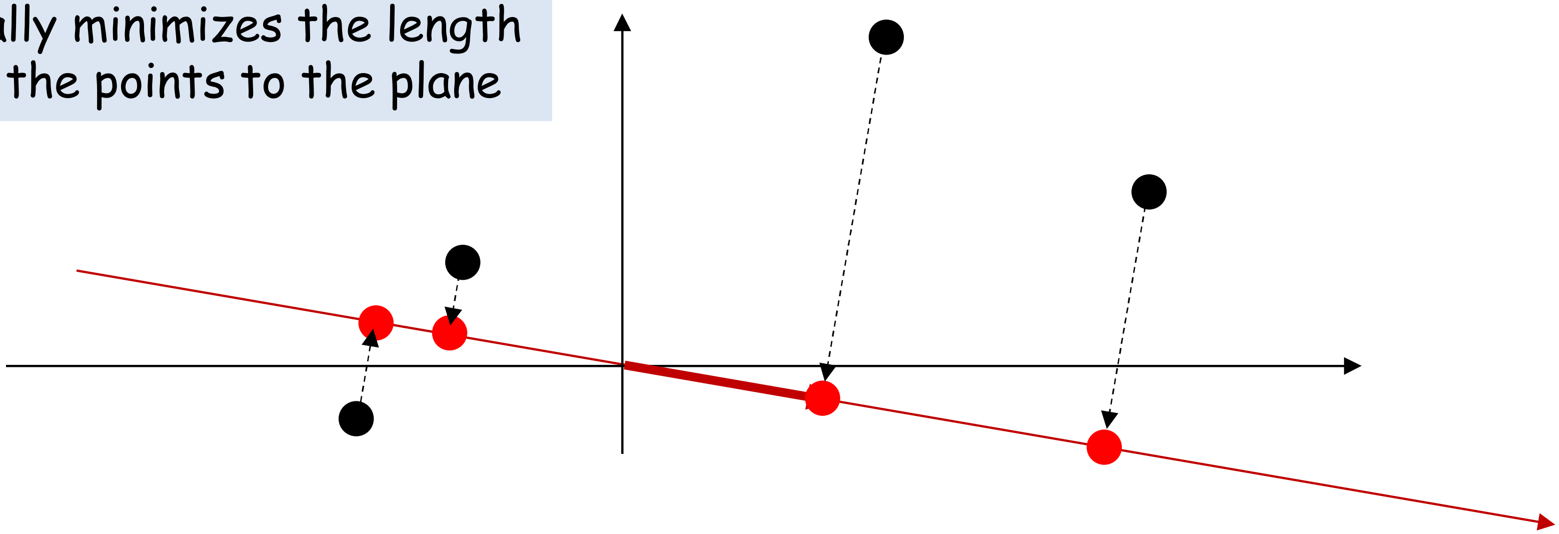
The iterative algorithm



- Initialize a subspace (the basis w)

The iterative algorithm

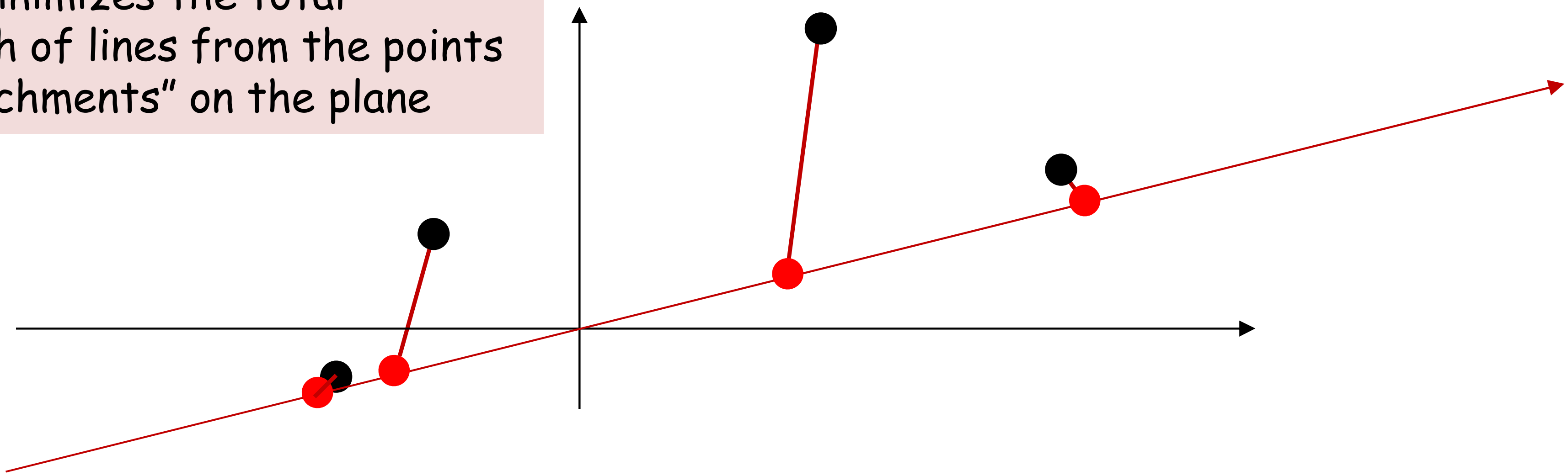
This individually minimizes the length of lines from the points to the plane



- 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

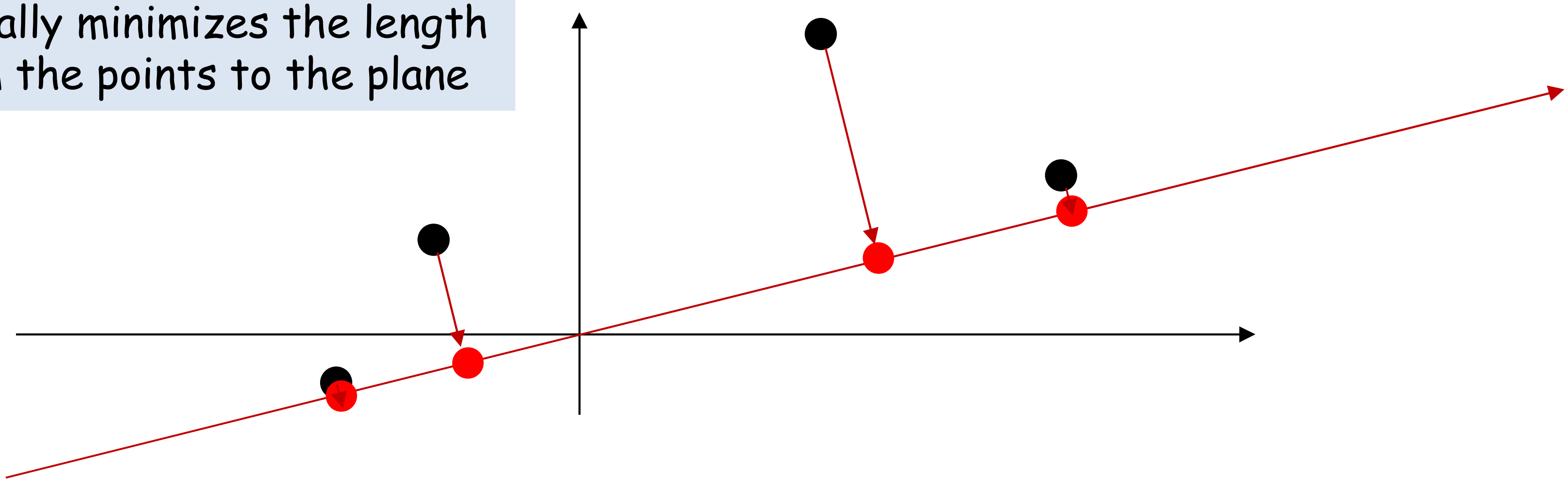
This *jointly* minimizes the total squared length of lines from the points to their "attachments" on the plane



- 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

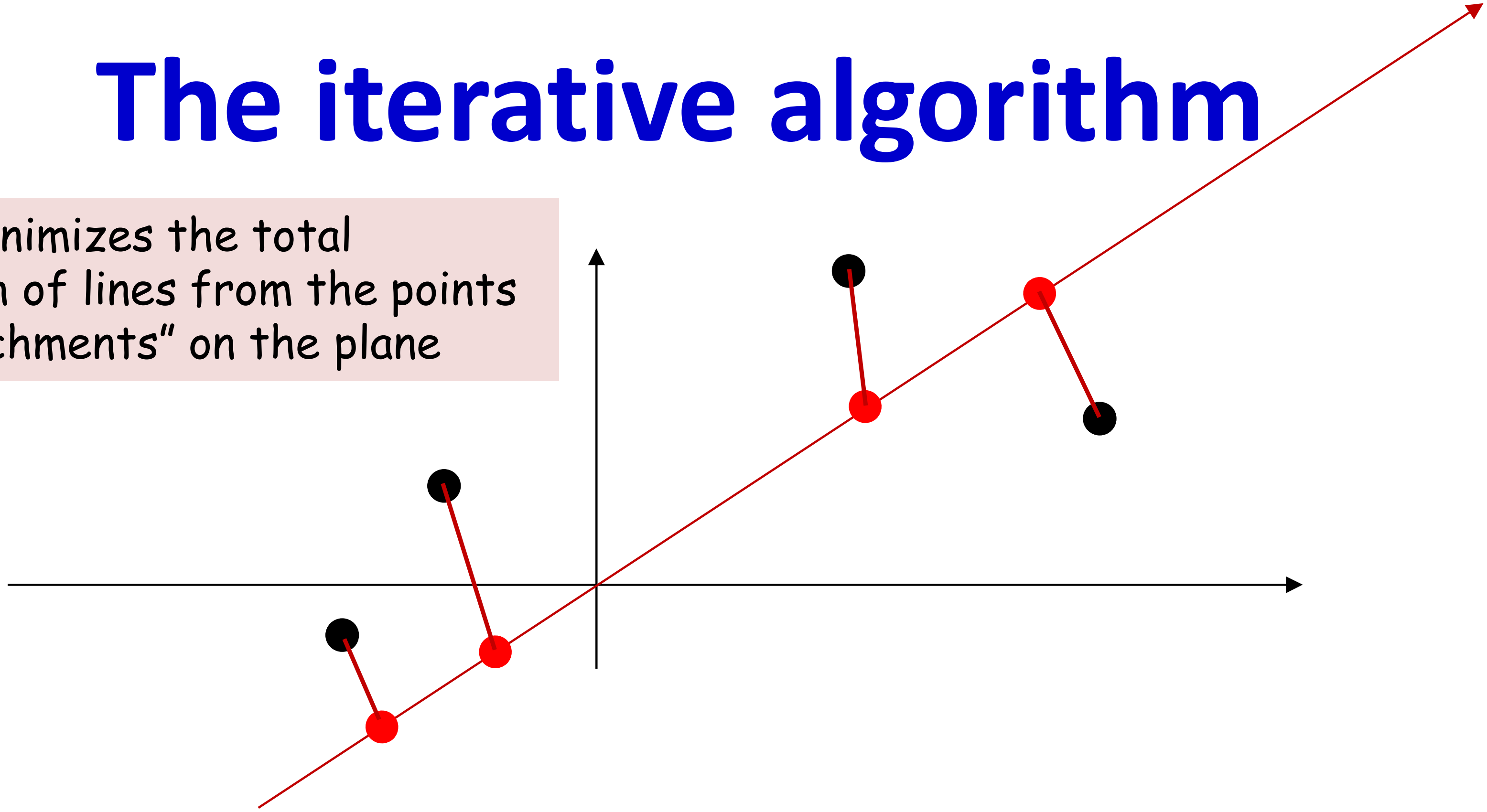
This individually minimizes the length of lines from the points to the plane



- 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

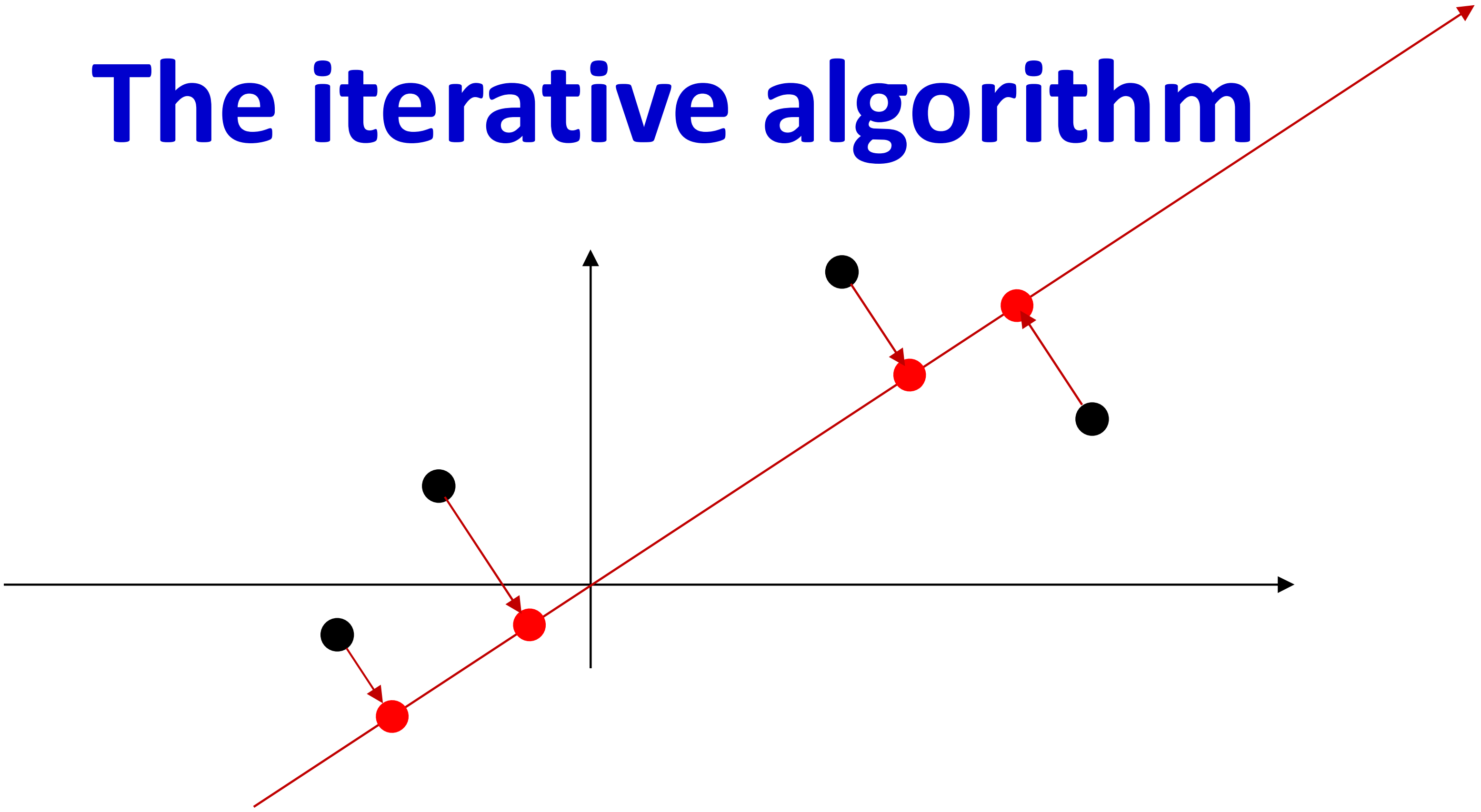
The iterative algorithm

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



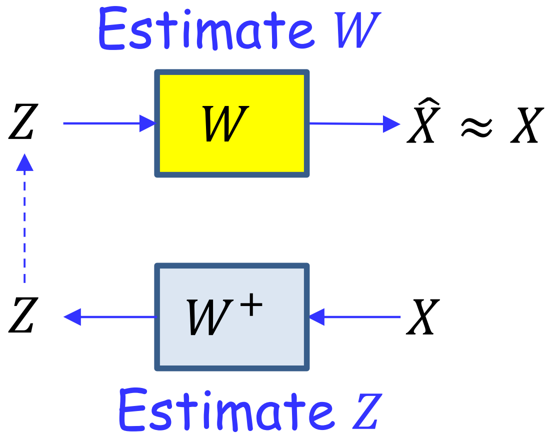
- Initialize a subspace (the basis w)
- Iterate until convergence:
  - Find the best position vectors Z on the W subspace for each training instance
    - Find the location on W that is *closest* to each instance, i.e. the perpendicular projection
  - Let W rotate and stretch/shrink, keeping the arrangement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

A cartoon view of Iterative PCA

Iterate :

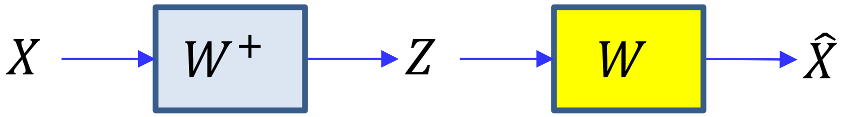
$Z \leftarrow W^{+}X$

$W \leftarrow XZ^{+}$



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

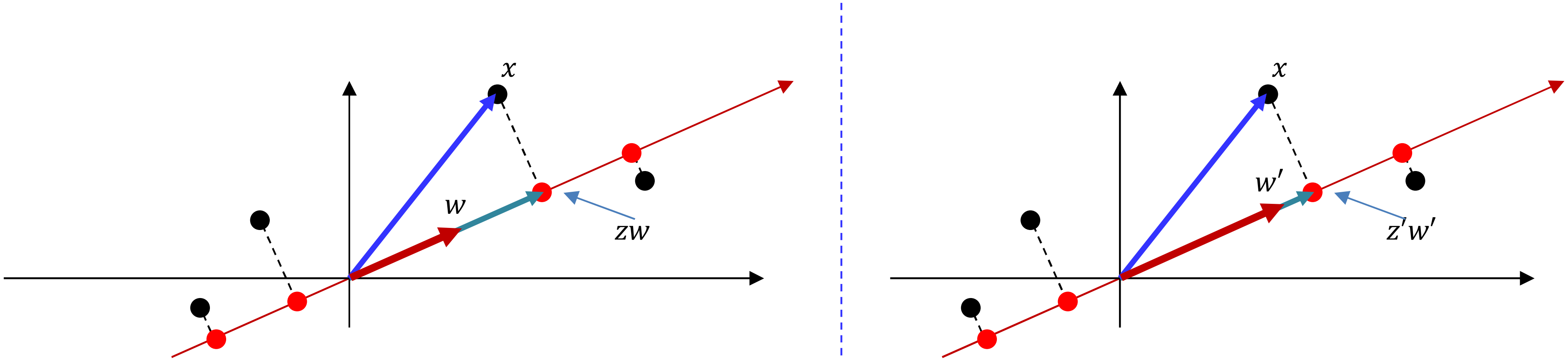
Drawing this differently

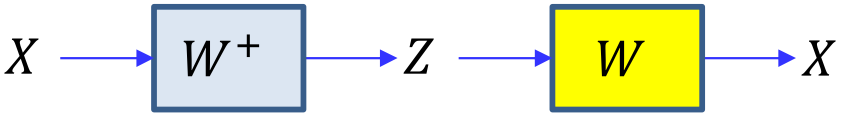


- This is just an autoencoder with linear activations

- Backprop actually works by simultaneously updating Z (implicitly) and W in tiny increments

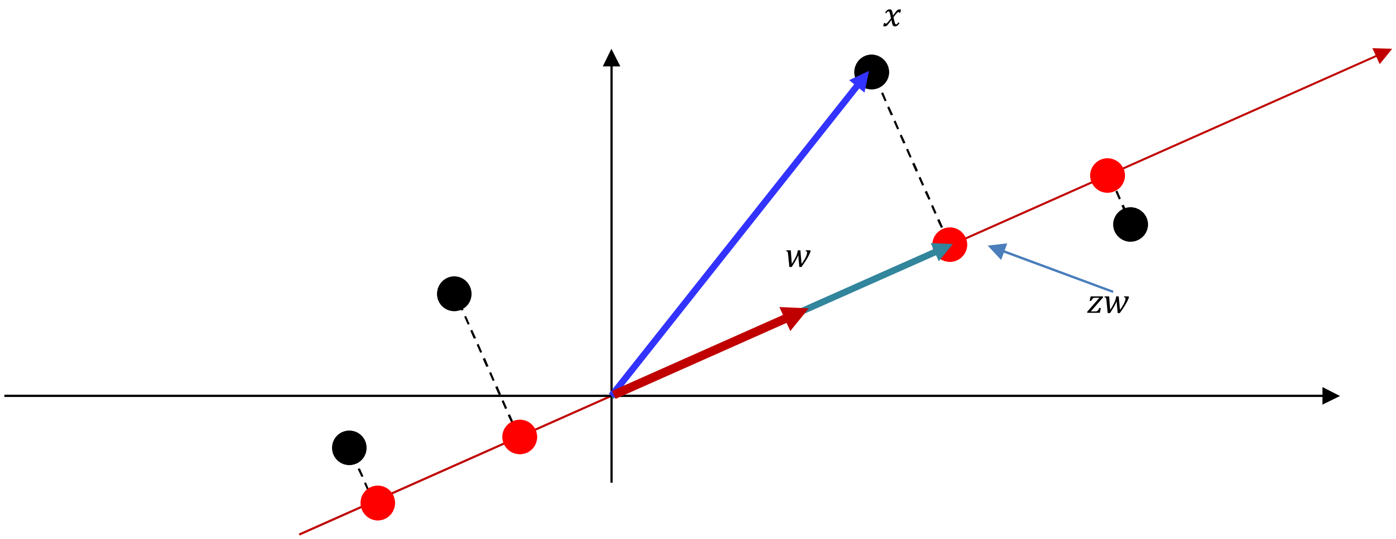
A minor issue: Scaling invariance





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

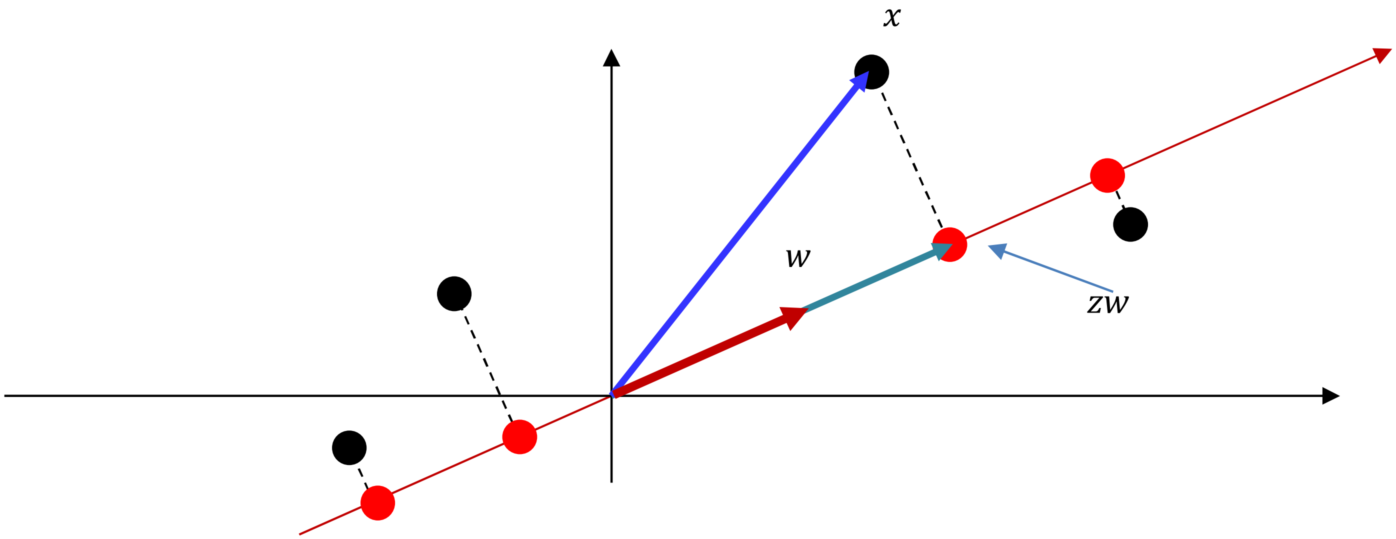
Resolving this issue



- 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

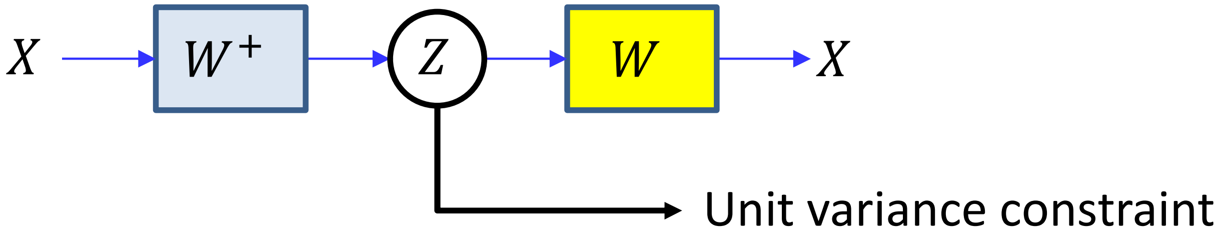
Resolving this issue



- 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

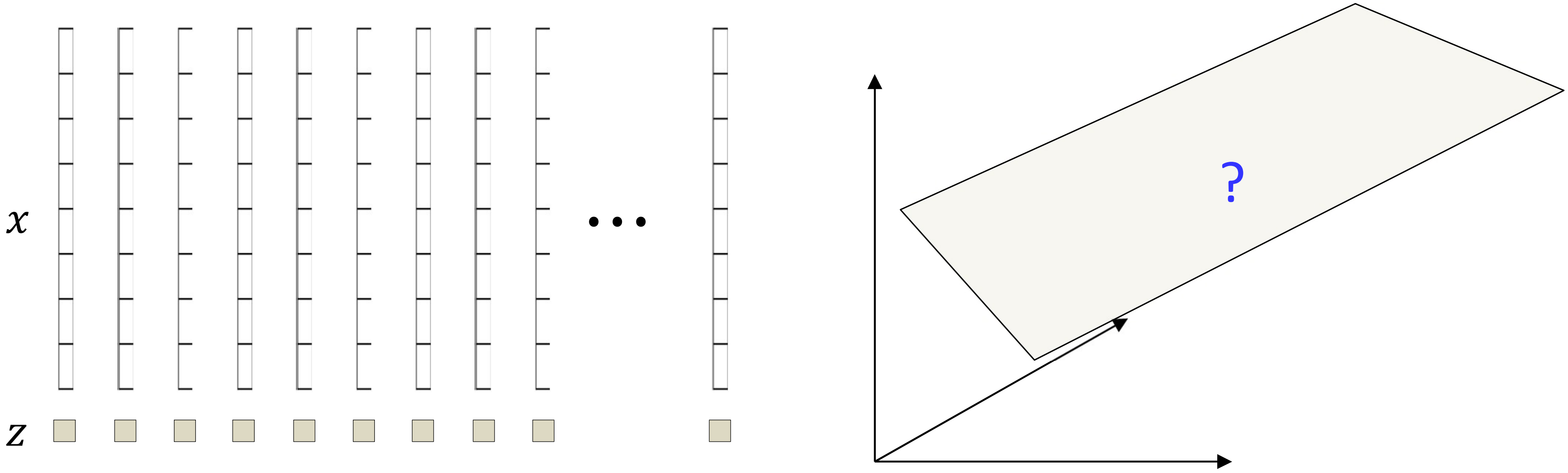
Constraining the linear AE



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

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

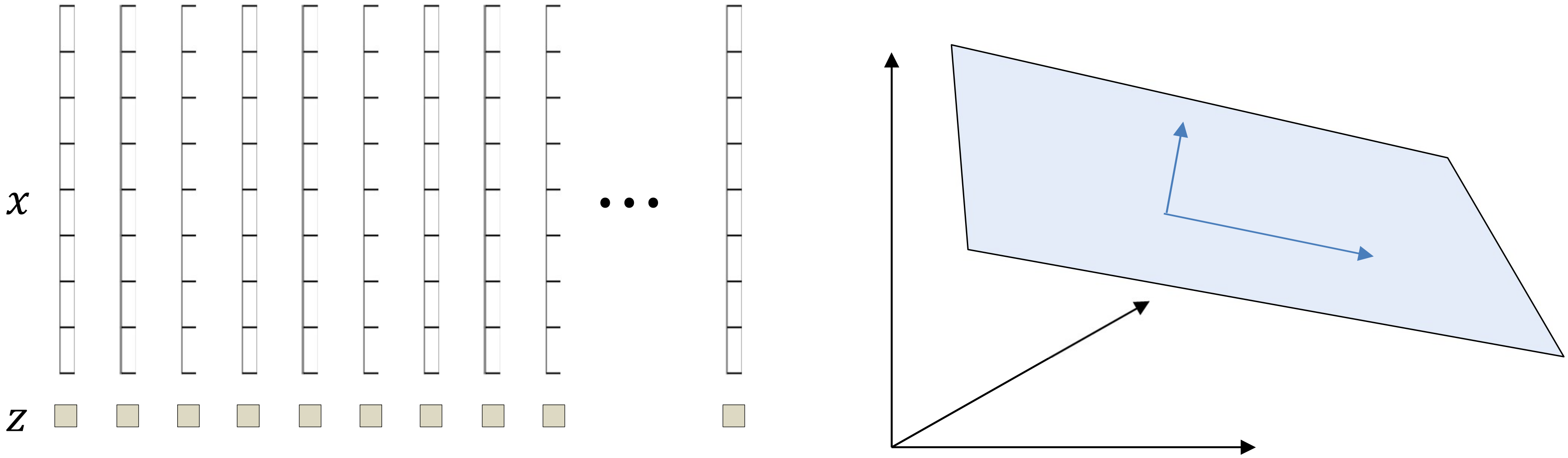
So what are we doing in the iterative solution?



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

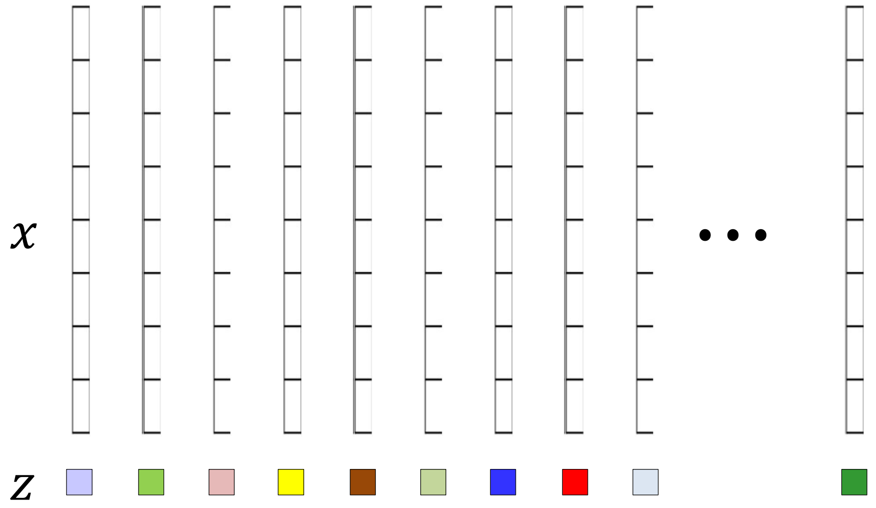
- If we had z, we could uniquely identify the plane

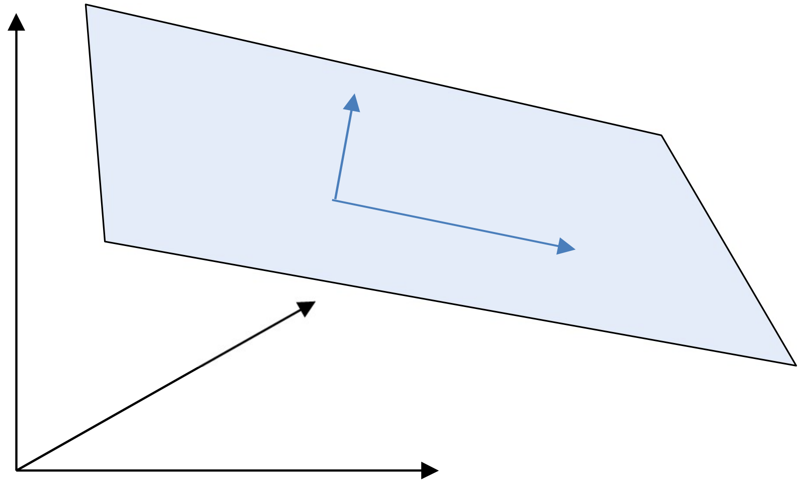
Iterative solution



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

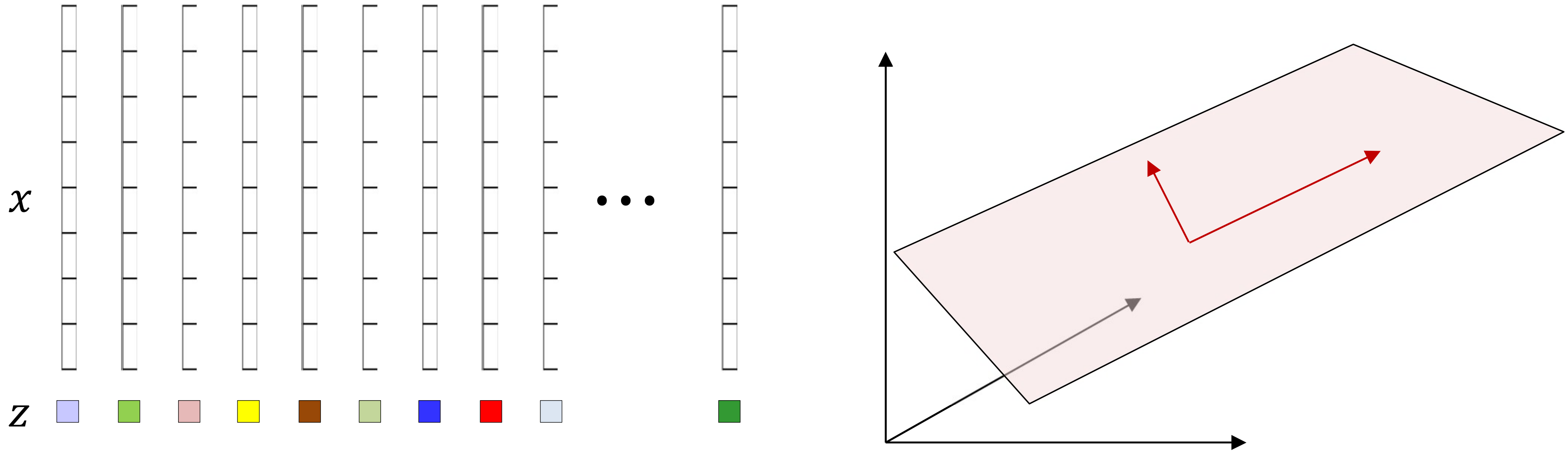
Iterative solution





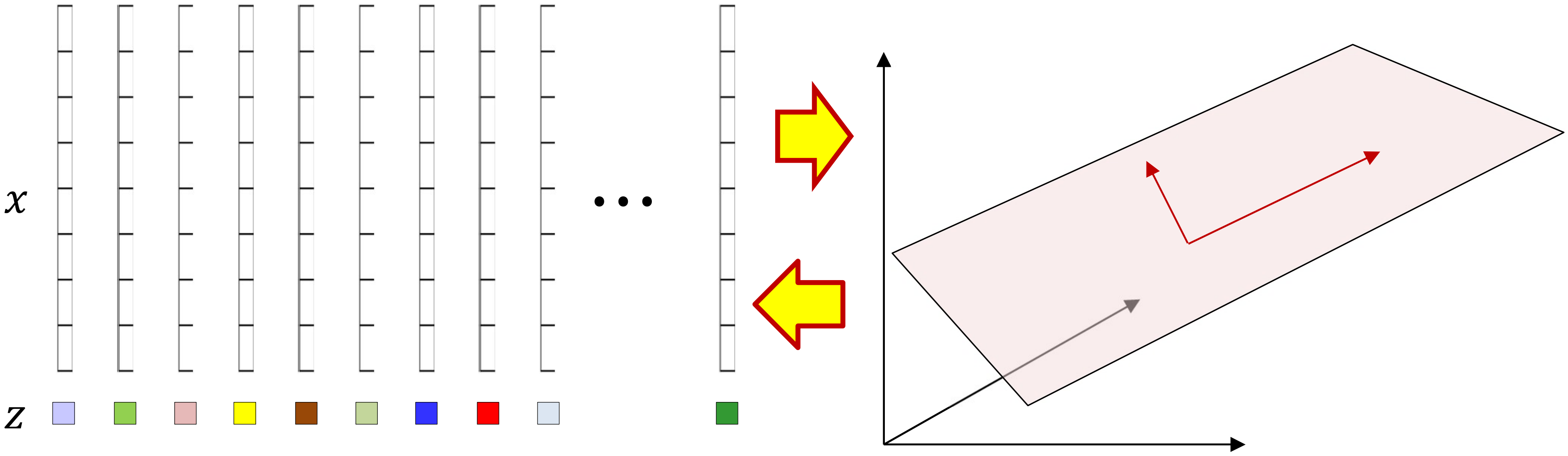
- Initialize the plane
  - Or rather, the bases for the plane
- "Complete" the data by computing the appropriate zs for the plane

Iterative solution



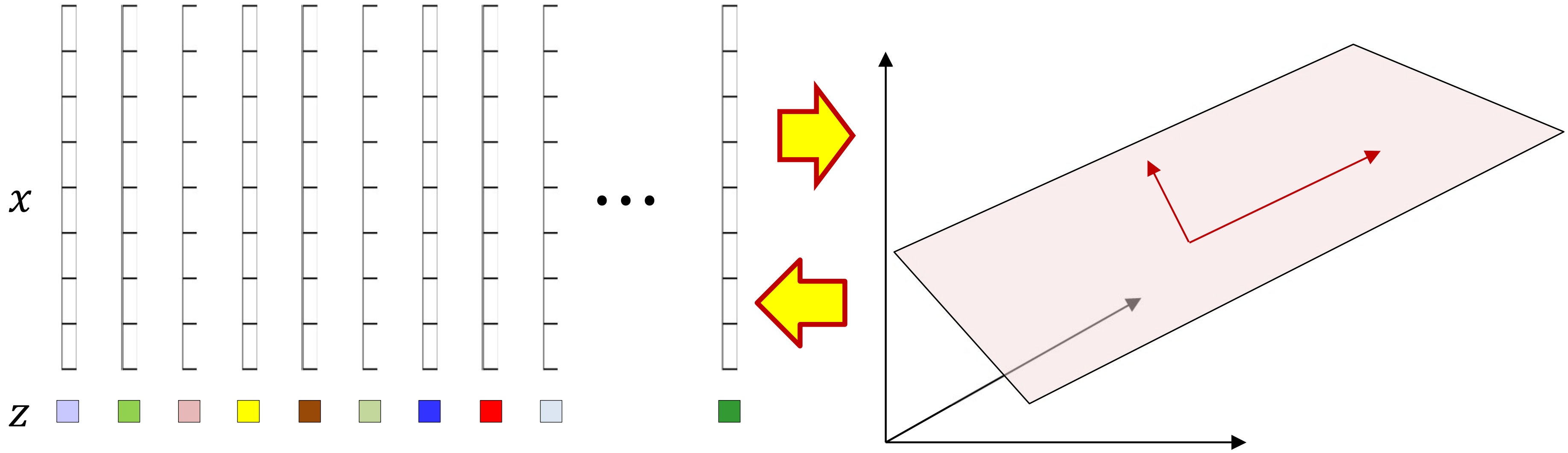
- Initialize the plane
  - – Or rather, the bases for the plane
- “Complete” the data by computing the appropriate zs for the plane
- Reestimate the plane using the zs

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

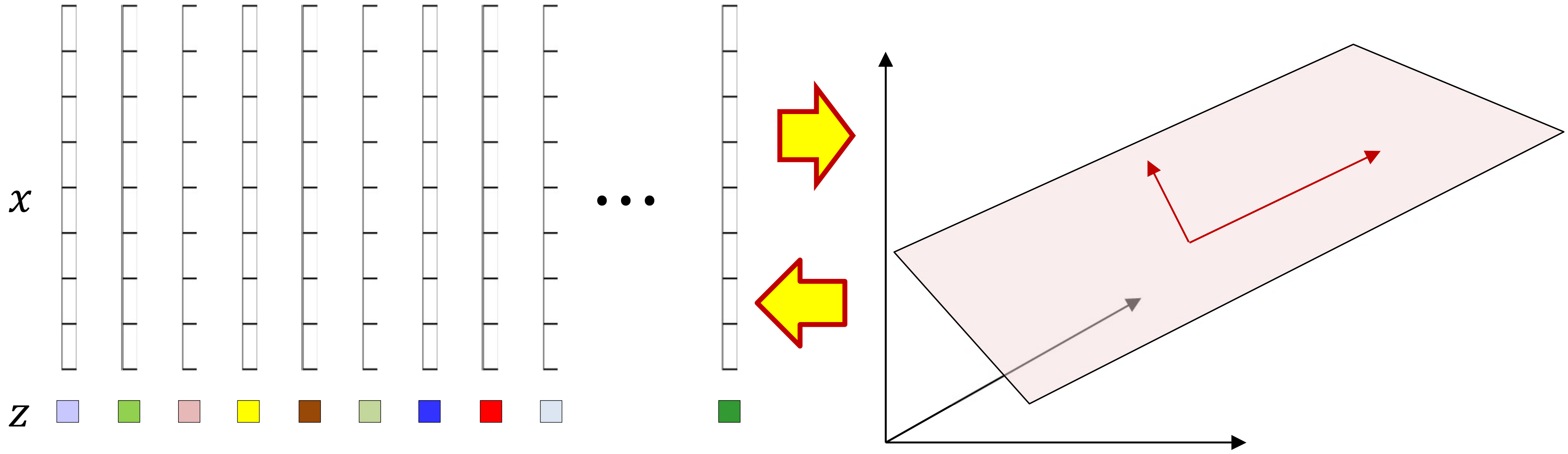
Iterative solution



- Initialize the plane
  - Or rather, the bases for the plan
- “Complete” the d
- Reesti
- Iterate

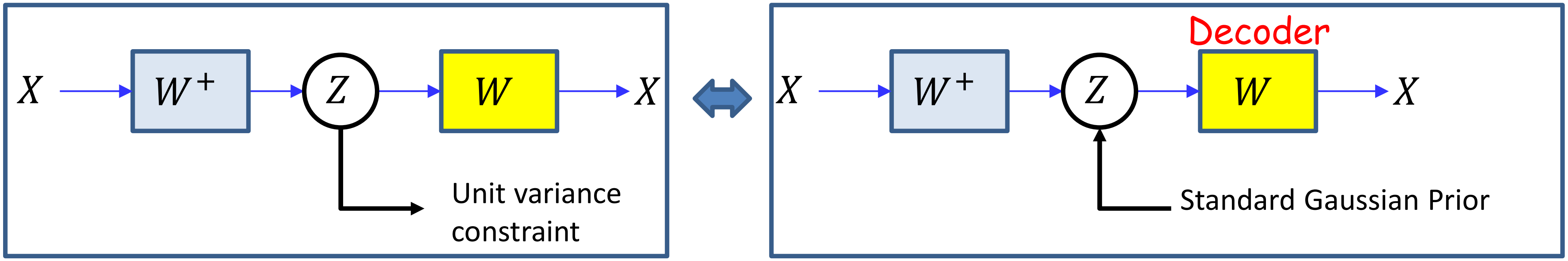
Look Familiar?

Iterative solution



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

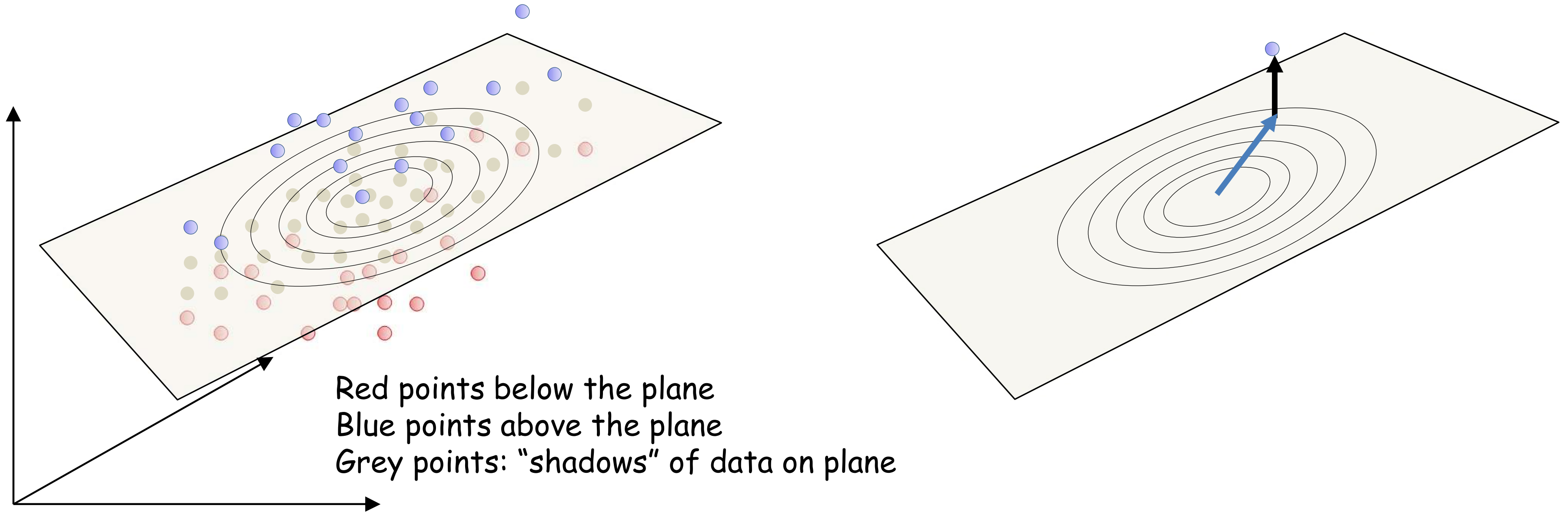
Constraining the linear AE



- 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)



Red points below the plane
Blue points above the plane
Grey points: “shadows” of data on plane

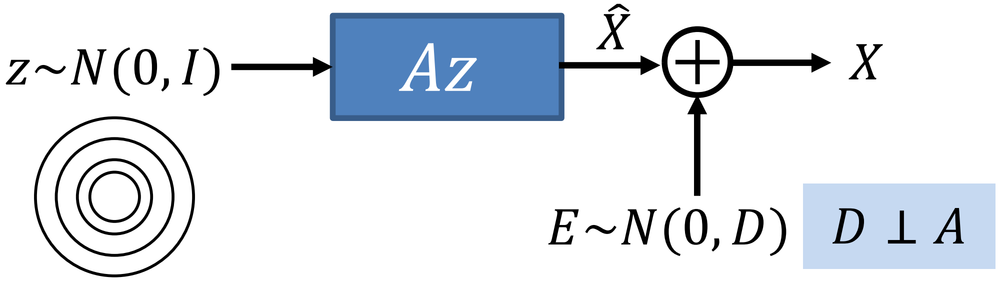
- Linear AEs actually have a generative story
- In order to generate any point
  - We first take a Gaussian step on the principal plane
  - Then we take an orthogonal *Gaussian* step from where we land to generate a point
  - PCA / Linear AEs find the plane and the characteristics of the Gaussian steps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

The *generative* story behind PCA (linear AEs)

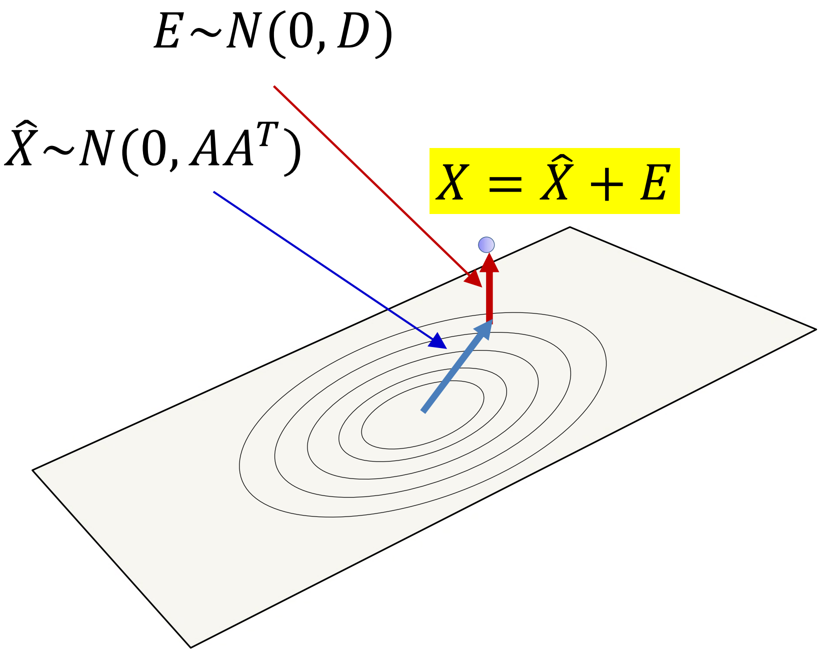
$z \sim N(0, I)$

$E \sim N(0, D)$

$X = Az + E$



$D \perp A$



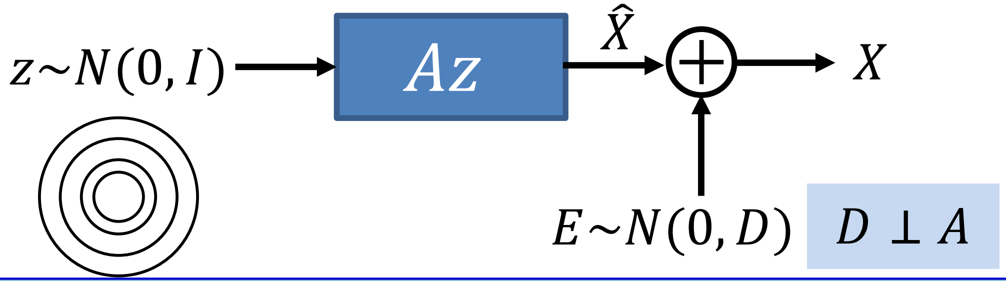
- **Generative story for PCA:**
  - z is drawn from a K-dim isotropic Gaussian
    - K is the dimensionality of the principal subspace
  - A is "basis" matrix for the subspace
  - E is a 0-mean Gaussian noise that is orthogonal to the principal subspace
    - **The covariance of the Gaussian is low-rank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

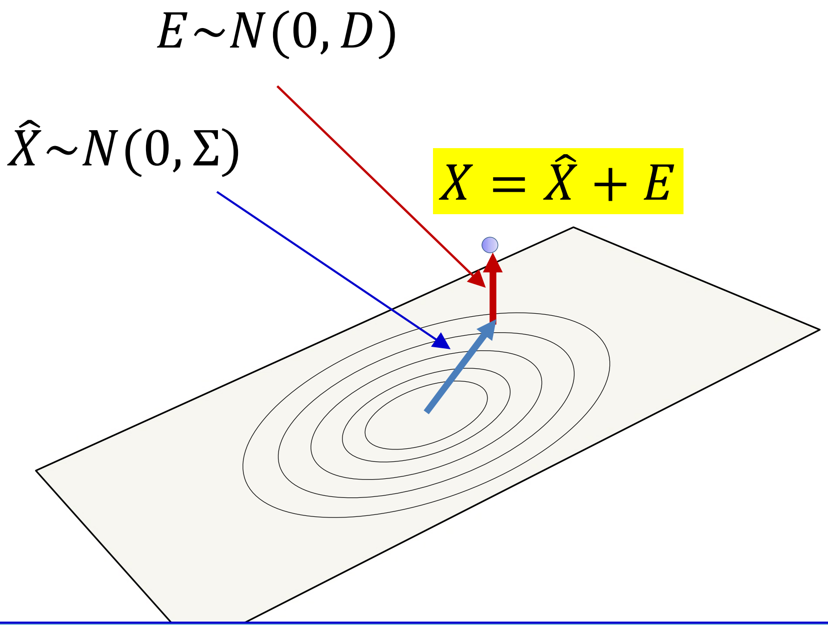
The *generative* story behind PCA (linear AEs)

$z \sim N(0, I)$

$E \sim N(0, D)$

$X = Az + E$

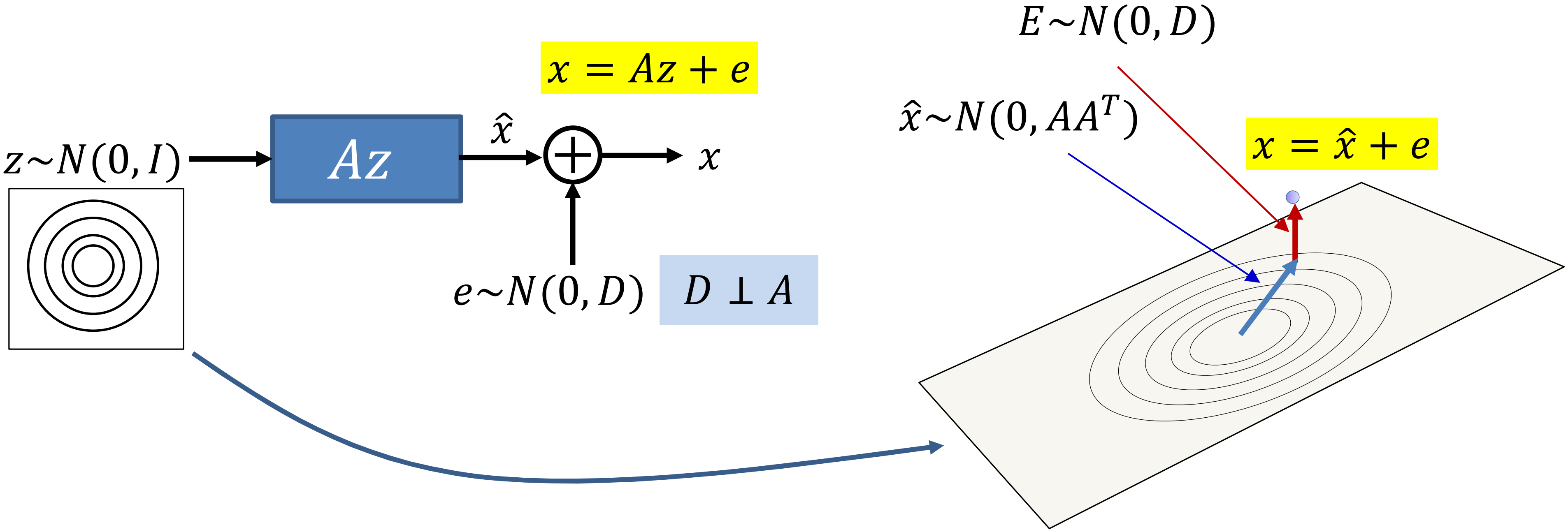




PCA implicitly obtains maximum likelihood estimate of A and D, from training data X

- **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 for the subspace
  - 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



$x = Az + e$

$z\sim N(0,I)$ $\longrightarrow$ $\boxed{Az}$ $\xrightarrow{\hat{x}}$ $\oplus$ $\longrightarrow$ x

$e\sim N(0,D)$

$D \perp A$

$E\sim N(0,D)$

$\hat{x}\sim N(0,AA^T)$

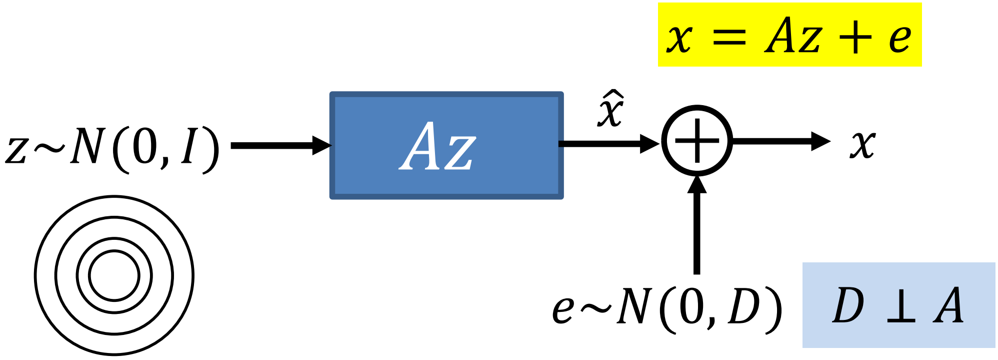
$x = \hat{x} + e$

- Alternate view: Az stretches and rotates the K-dimensional planar space of z into a K-dimensional planar subspace (manifold) of the data space

- The circular distribution of z in the K-dimensional z space transforms into an ellipsoidal distribution on a K-dimensional hyperplane the data space

- Samples are drawn from the ellipsoidal distribution on the hyperplane, and noise is added to them

The probability modelled by PCA



$x = Az + e$

$z \sim N(0, I)$

Az

$\hat{x}$

x

$e \sim N(0, D)$

$D \perp A$

- **PCA models a Gaussian distribution:**

$$\hat{x} = Az \Rightarrow \qquad P(\hat{x}) = N(0, AA^T)$$

$$x = \hat{x} + E \Rightarrow \qquad P(x) = N(0, AA^T + D)$$

- The probability density of x is Gaussian lying mostly close to a hyperplane
  - With correlated structure on the plane
  - And uncorrelated components orthogonal to the plane

Poll 1 (@1844)

Choose all that are true about PCA

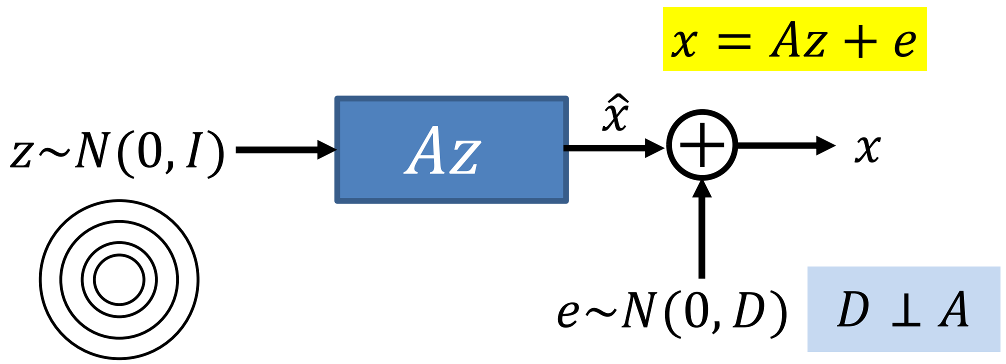
- It actually performs Maximum Likelihood estimation of a generative model for the data
- The generative model for PCA is that in order to generate any point, the process first takes a Gaussian step on the principal hyperplane, followed by a Gaussian step perpendicular to the hyperplane
- It can also be iteratively estimated using Expectation Maximization
- It assumes the distribution of the data is a Gaussian that is centered on the principal hyperplane

Poll 1

Choose all that are true about PCA

- It actually performs Maximum Likelihood estimation of a generative model for the data
- The generative model for PCA is that in order to generate any point, the process first takes a Gaussian step on the principal hyperplane, followed by a Gaussian step perpendicular to the hyperplane
- It can also be iteratively estimated using Expectation Maximization
- It assumes the distribution of the data is a Gaussian that is centered on the principal hyperplane

Missing information for PCA



$x = Az + e$

$z \sim N(0, I)$

Az

$\hat{x}$

x

$e \sim N(0, D)$

$D \perp A$

- There is missing information about the observation X
  - Information about intermediate values drawn in generating X
  - We don’t know z

- If we knew z for each X, estimating A (and D) would be simple

PCA with complete information

$$x = Az + E$$
$$P(x|z) = N(Az, D)$$

- Given complete information $(x_1, z_1), (x_2, z_2), \dots$
  - Representing $X = [x_1, x_2, \dots], \;\; Z = [z_1, z_2, \dots]$

$$\underset{A,D}{\operatorname{argmax}} \sum_{(x,z)} \log P(x,z) \; = \underset{A,D}{\operatorname{argmax}} \sum_{(x,z)} \log P(x|z)$$

$$= \underset{A,D}{\operatorname{argmax}} \sum_{(x,z)} \log \frac{1}{\sqrt{(2\pi)^d |D|}} \exp(-0.5(x - Az)^T D^{-1}(x - Az))$$

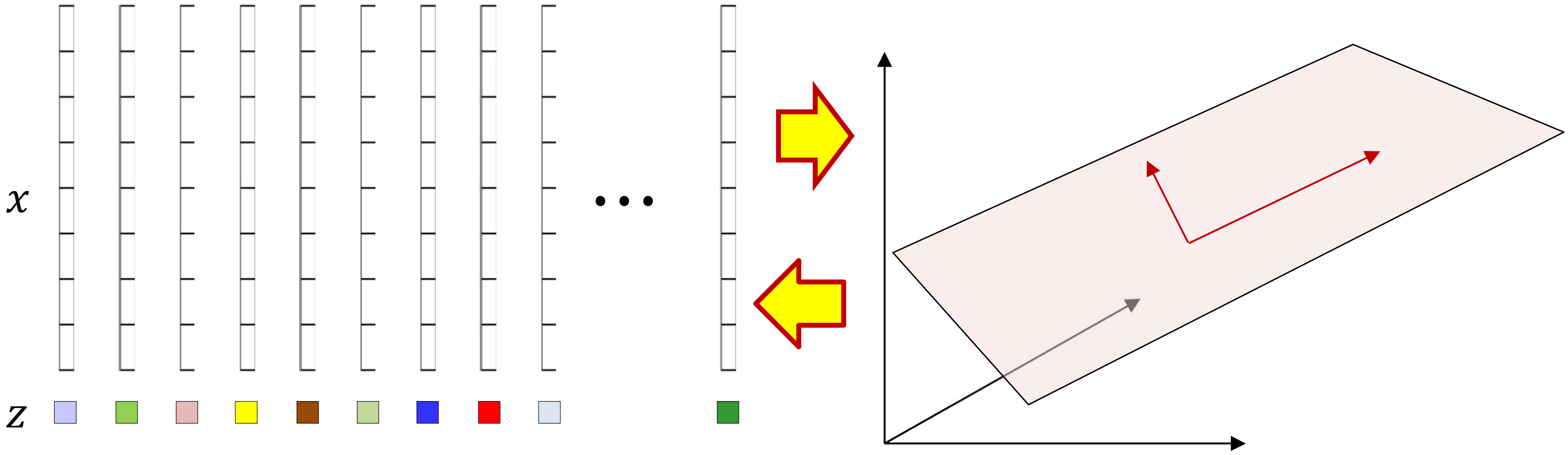
- Differentiating w.r.t A and equating to 0, we get the easy solution

$$A = XZ^{+}$$

  - (Some sloppy math (D is not invertible), but the solution is right)

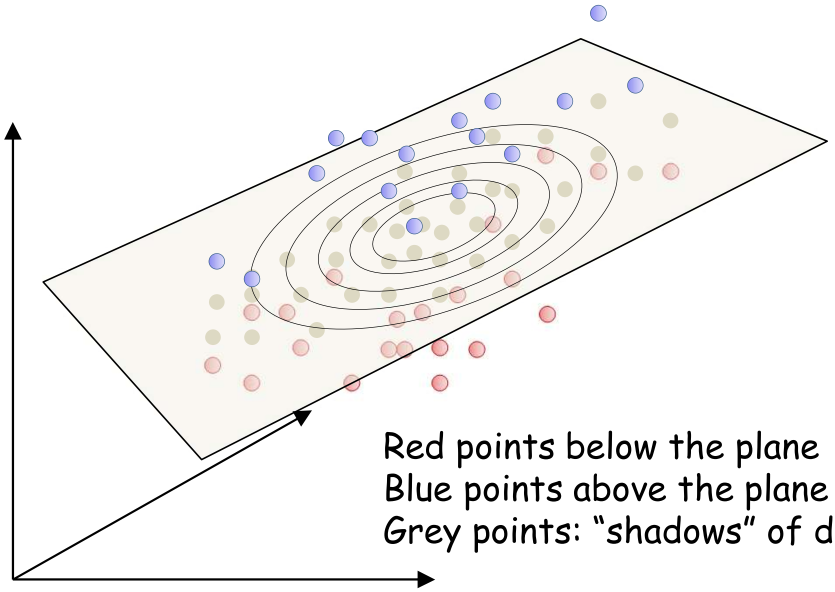
But we don't have z. It is missing

EM for PCA

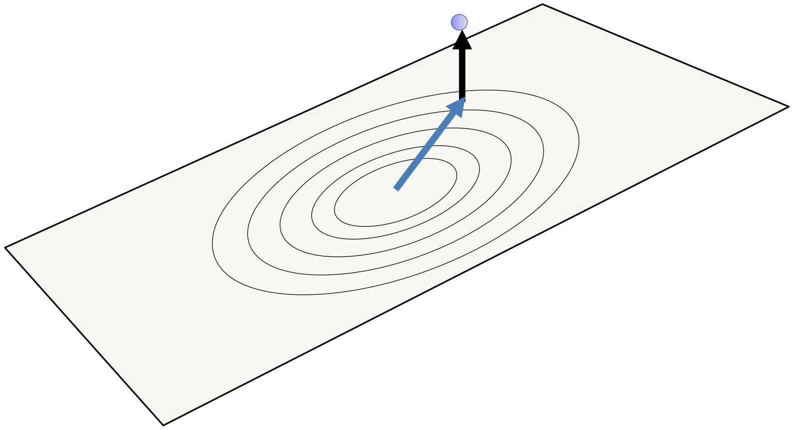


- Initialize the plane
  - Or rather, the bases for the plane
- "Complete" the data by computing the appropriate zs for the plane
  - $P(z|X;\ A)$ is a delta, because E is orthogonal to A
- Reestimate the plane using the zs
- Iterate

Improving on PCA



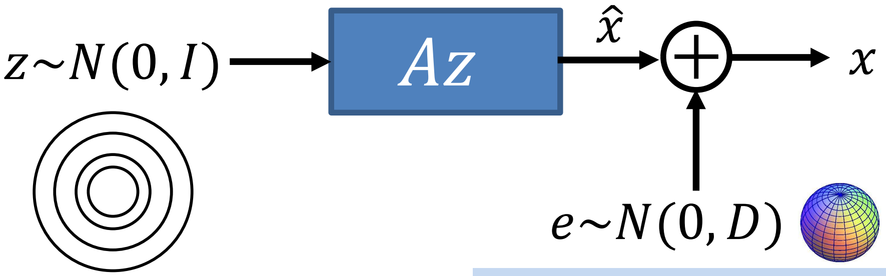
Red points below the plane
Blue points above the plane
Grey points: “shadows” of data on plane



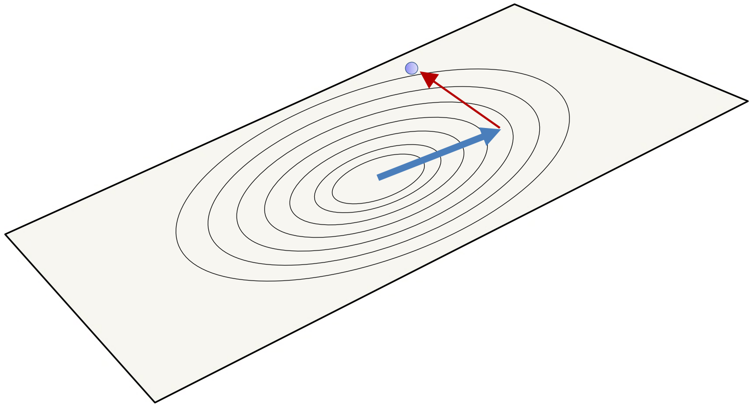
- PCA assumes the noise is always orthogonal to the data
  - Not always true
  - Noise in images can look like images, random noise can sound like speech, etc.
- Let us generalize the model to permit non-orthogonal noise

The Linear Gaussian Model

$$x = Az + e$$



D is full rank diagonal

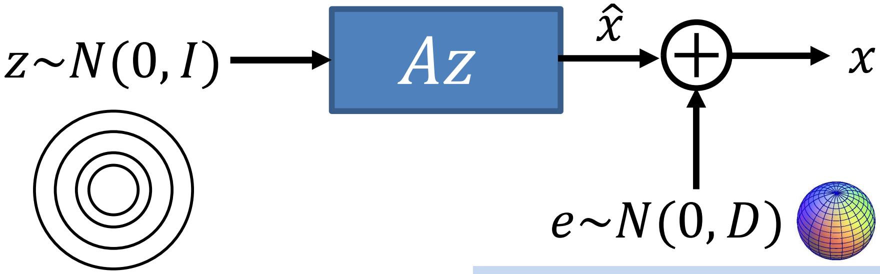


- The noise added to the output of the encoder can lie in *any* direction
  - Uncorrelated, but not just orthogonal to the principal subspace

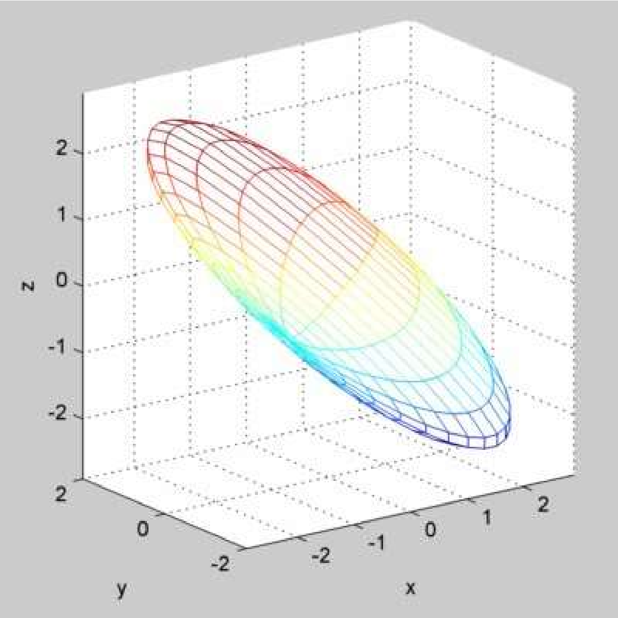
- Generative model: to generate any point
  - Take a Gaussian step on the hyperplane
  - Add *full-rank* Gaussian uncorrelated noise ***that is independent of the position on the hyperplane***
    - Uncorrelated: diagonal covariance matrix
    - Direction of noise is unconstrained
      - Need not be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

The probability distribution modelled by the LGM

$x = Az + e$



D is full rank diagonal



- The noise added to the output of the encoder can lie in *any* direction

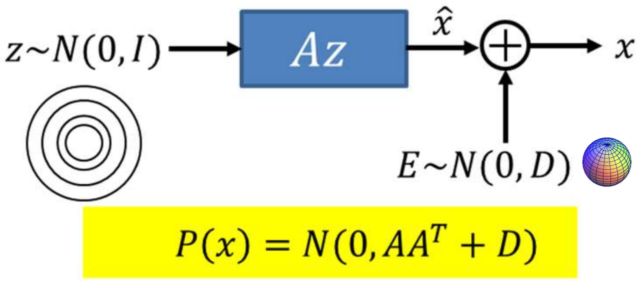
$$\hat{x} = Az \Rightarrow \qquad P(\hat{x}) = N(0, AA^T)$$

$$x = \hat{x} + E \Rightarrow \qquad P(x) = N(0, AA^T + D)$$

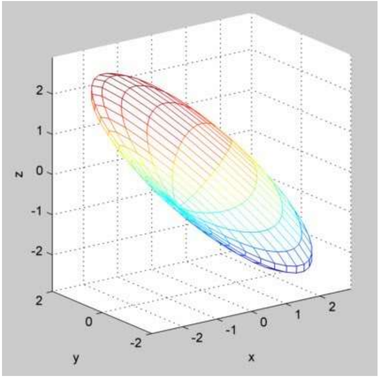
- The probability density of x is Gaussian lying mostly close to a hyperplane
  - With uncorrelated Gaussian
- Also

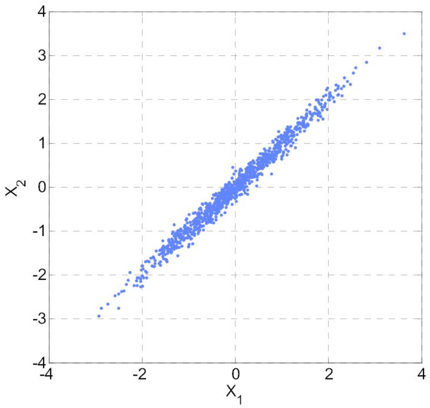
$$P(x|z) = N(Az, D)$$

The linear Gaussian model



$$P(x) = N(0, AA^T + D)$$

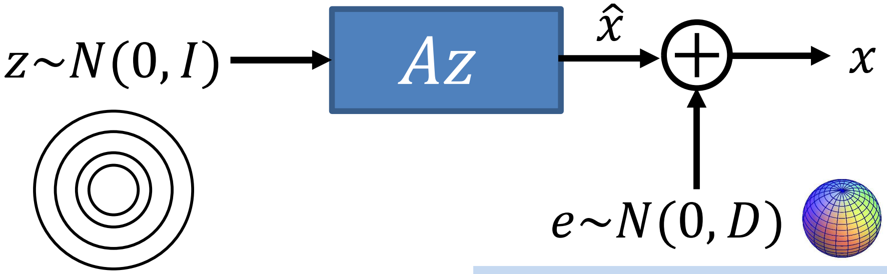




- Is a generative model for Gaussians
- Data distribution are Gaussian lying largely on a hyperplane with some Gaussian “fuzz”
  - Only components on the plane are correlated with one another
    - No correlations off the plane
  - Which allows us to model *some* correlations between components
    - Halfway between a Gaussian with a diagonal covariance, and one with a full covariance

ML estimation of LGM parameters

$$x = Az + e$$



$z \sim N(0, I)$

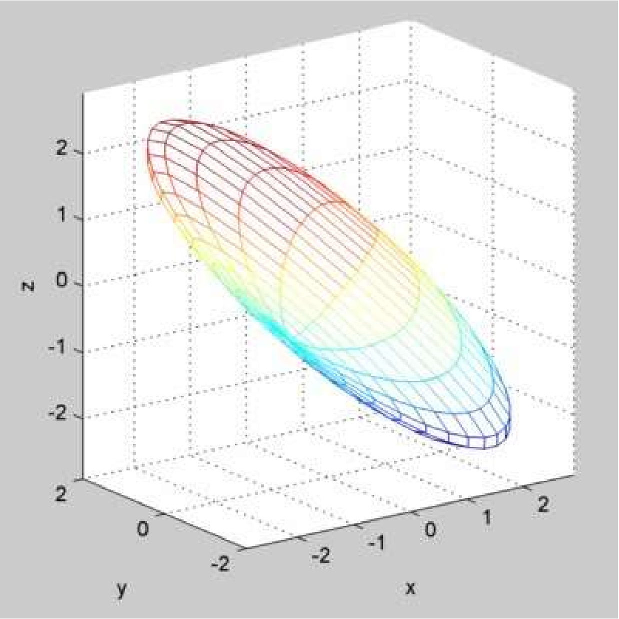
Az

$\hat{x}$

x

$e \sim N(0, D)$

D is full rank diagonal



$$P(x) = N(0, AA^T + D)$$

- The parameters of the LGM generative model are A and D
- The ML estimator is

$$\underset{A,D}{\operatorname{argmax}} \sum_x \log \frac{1}{\sqrt{(2\pi)^d |AA^T + D|}} \exp(-0.5x^T (AA^T + D)^{-1} x)$$

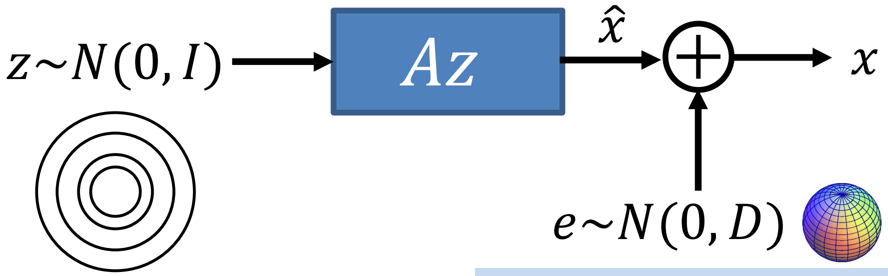
  - Where d is the dimensionality of the space

- As it turns out, this does *not* have a nice closed form solution
  - Because D is full rank

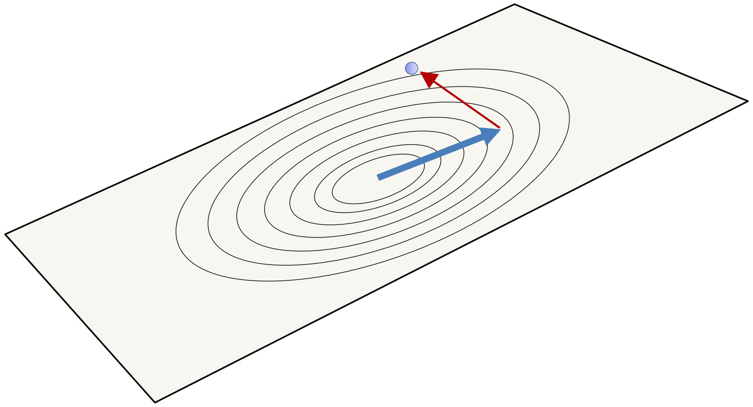
- Will require EM

Missing information for LGMs

$x = Az + e$



D is full rank diagonal



- There is missing information about the observation X
  - Information about intermediate values drawn in generating X
  - We don't know z
- If we knew the z for each X, estimating A (and D) would be very simple

LGM with complete information

$$x = Az + e$$
$$P(x|z) = N(Az, D)$$

- Given complete information $X = [x_1, x_2, \dots], \;\; Z = [z_1, z_2, \dots]$

$$\underset{A,D}{\operatorname{argmax}} \sum_{(x,z)} \log P(x, z) \;= \underset{A,D}{\operatorname{argmax}} \sum_{(x,z)} \log P(x|z)$$

$$= \underset{A,D}{\operatorname{argmax}} \sum_{(x,z)} \log \frac{1}{\sqrt{(2\pi)^d |D|}} \exp(-0.5(x - Az)^T D^{-1} (x - Az))$$

$$= \underset{A,D}{\operatorname{argmax}} \sum_{(x,z)} -\frac{1}{2} \log|D| - 0.5(x - Az)^T D^{-1} (x - Az)$$

- Differentiating w.r.t A and D equating to 0, we get an easy solution

LGM with complete information

$$\underset{A,D}{\operatorname{argmax}} \sum_{(x,z)} -\frac{1}{2}\log|D| - 0.5(x - Az)^T D^{-1}(x - Az)$$

- Differentiating w.r.t A and $D$and equating to 0, we get an easy solution
- Solution for A

$$\nabla_A \sum_{(x,z)} 0.5(x - Az)^T D^{-1}(x - Az) = 0 \quad \Rightarrow$$

$$\sum_{(x,z)} (x - Az)z^T = 0 \quad \Rightarrow \quad A = \left(\sum_{(x,z)} xz^T\right)\left(\sum_{z} zz^T\right)^{-1}$$

- Solution for D

$$\nabla_D \sum_{(x,z)} \frac{1}{2}\log|D| + 0.5(x - Az)^T D^{-1}(x - Az) = 0 \quad \Rightarrow$$

$$D = diag\left(\frac{1}{N}\left(\sum_{x} xx^T - A\sum_{(x,z)} xz^T\right)\right)$$

LGM with complete information

$$\underset{A,D}{\operatorname{argmax}} \sum_{(x,z)} -\frac{1}{2}\log|D| - 0.5(x - Az)^T D^{-1}(x - Az)$$

- Differentiating w.r.t A and $D$and equating to 0, we get an easy solution
- Solution for A

$$\nabla_A \sum_{(x,z)} 0.5(x - Az)^T D^{-1}(x - Az) = 0 \quad \Rightarrow$$

$$\sum_{(x,z)} (x - Az)z^T = 0 \quad \Rightarrow \quad A = \left(\sum_{(x,z)} xz^T\right)\left(\sum_{z} zz^T\right)^{-1}$$

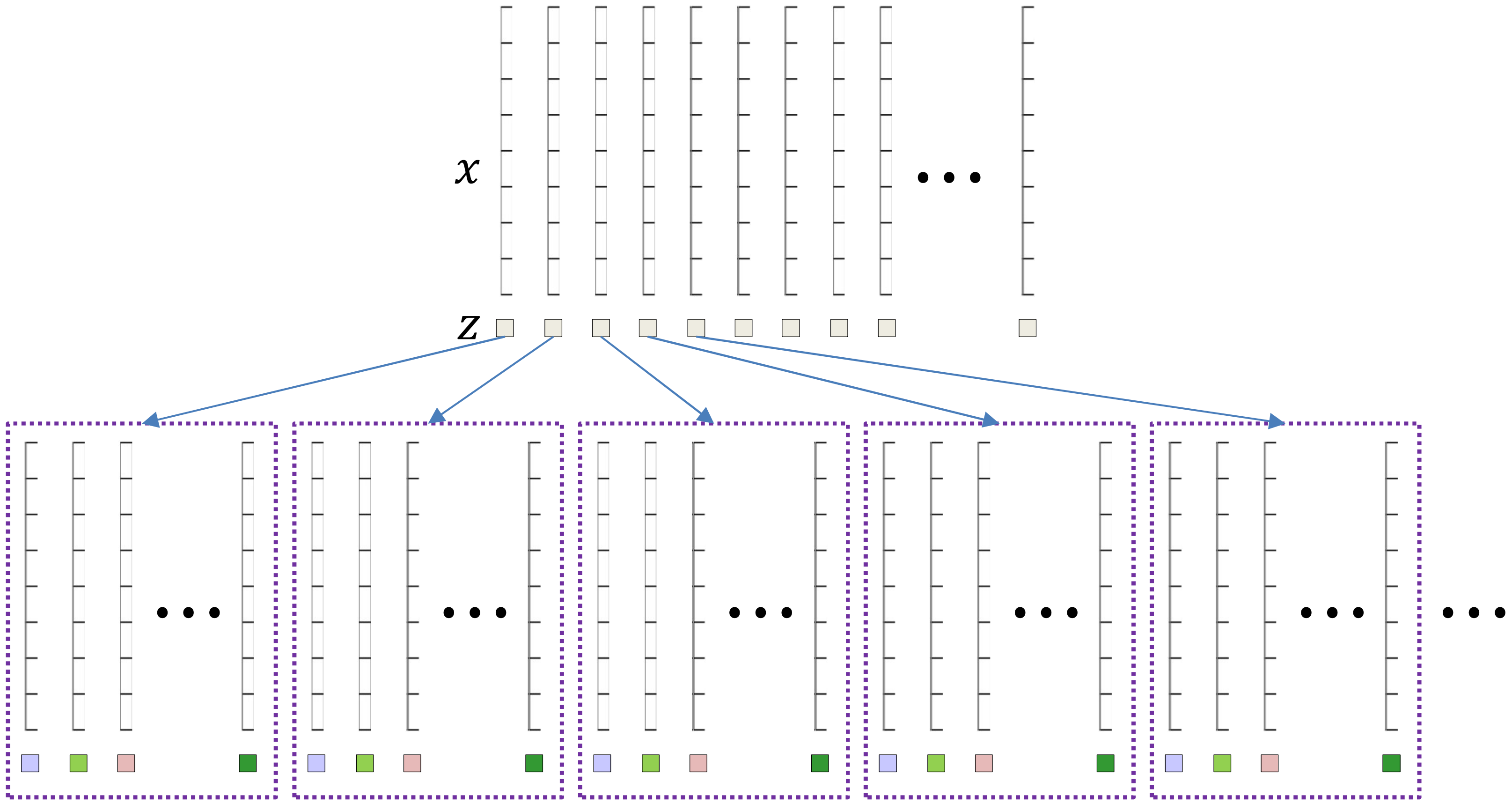
- Solution for D

$$\nabla_D \sum_{(x,z)} \frac{1}{2}\log|D| + 0.5(x - Az)^T D^{-1}(x - Az) = 0 \quad \Rightarrow$$

$$D = diag\left(\frac{1}{N}\left(\sum_{x} xx^T - A\sum_{(x,z)} xz^T\right)\right)$$

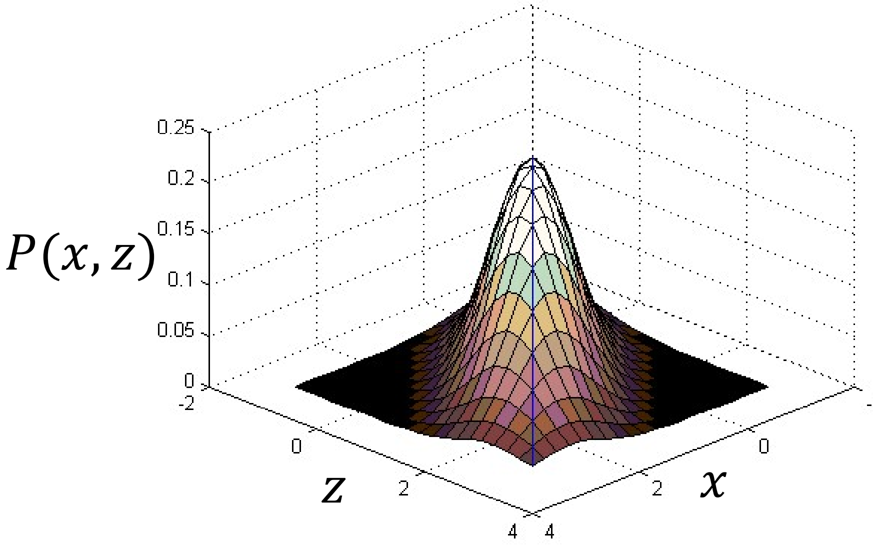
Unfortunately we do not observe z.
It is missing; the observations are incomplete

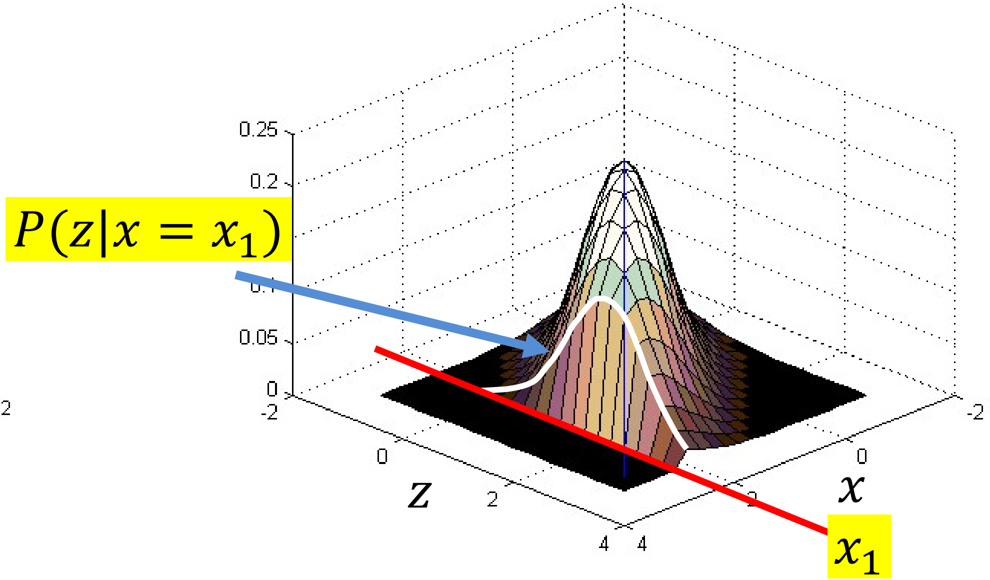
Expectation Maximization for LGM



- *Complete* the data
- Option 1:
  - In *every possible way* proportional to $P(z|x)$
  - Compute the solution from the completed data

The posterior $P(z|x)$



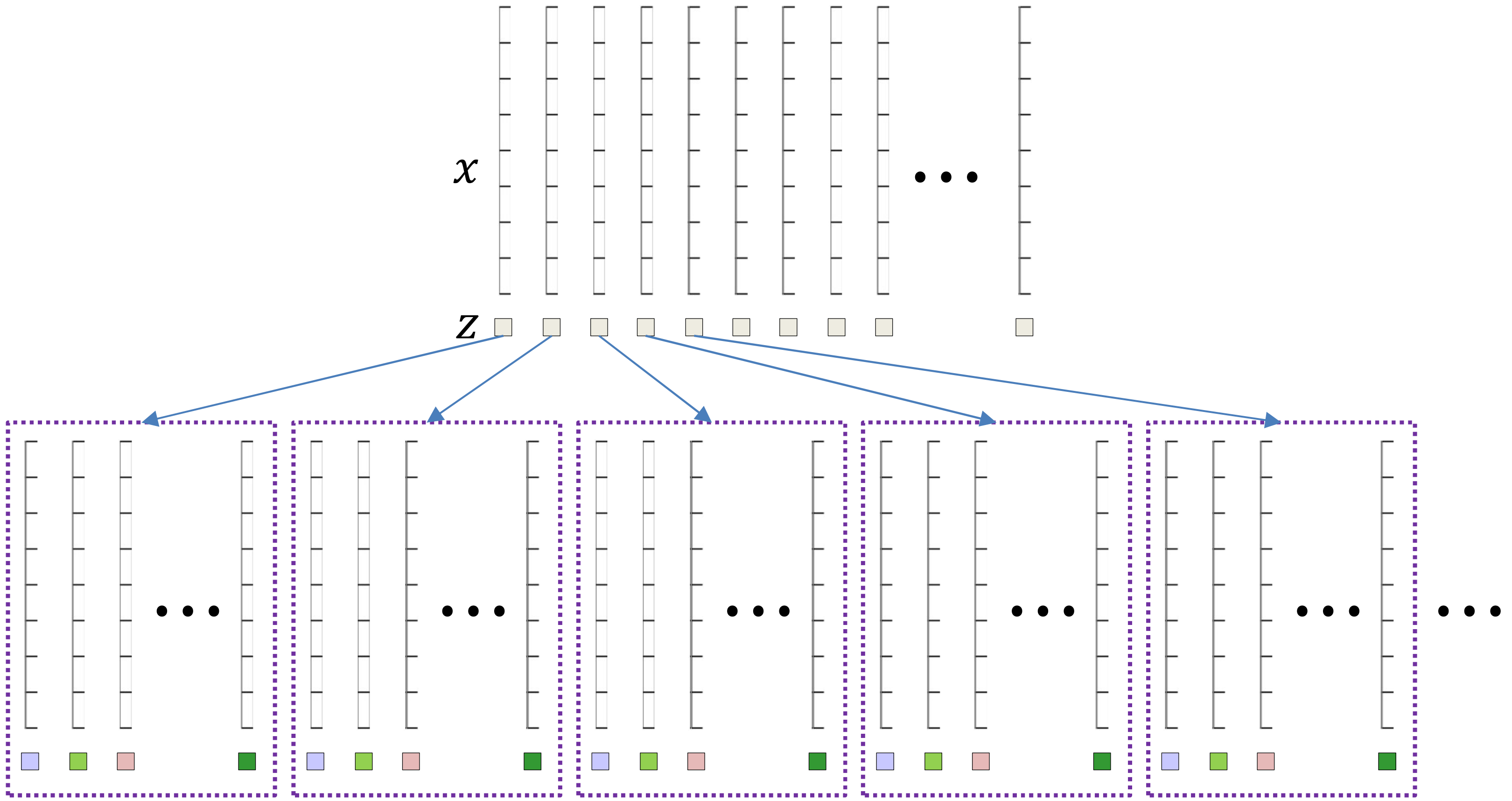


- $P(x)$ is Gaussian
  - We saw this
- The *joint* distribution of x and z is also Gaussian
  - Trust me
- The *conditional* distribution of z given x is also Gaussian

$$P(z|x) = N(z; A^T(AA^T + D)^{-1}x, I - A^T(AA^T + D)^{-1}A)$$

  - Trust me

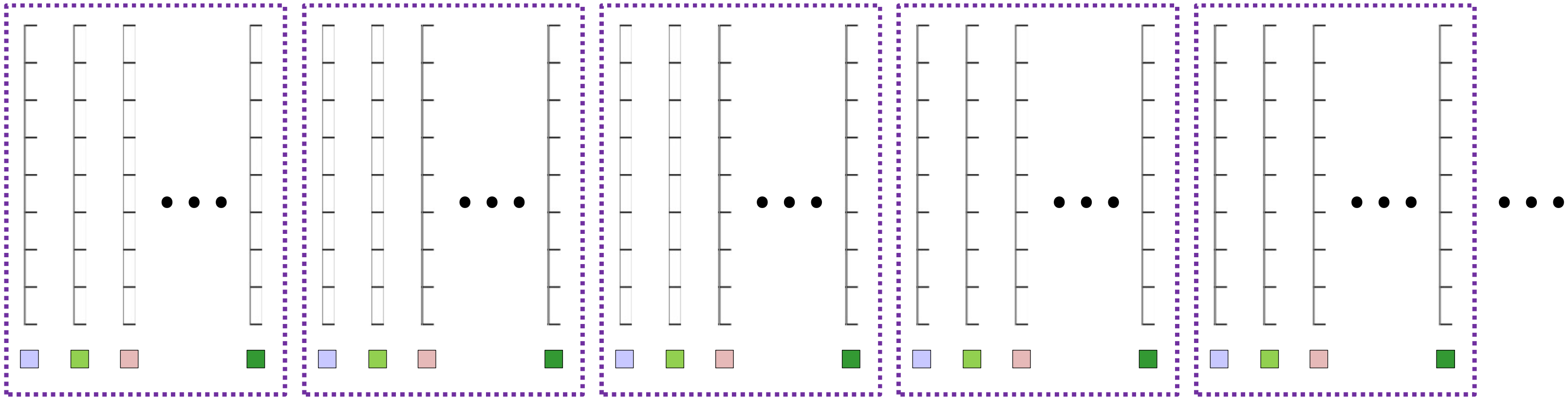
Expectation Maximization for LGM



$$P(z|x) = N(z; A^T(AA^T + D)^{-1}x, I - A^T(AA^T + D)^{-1}A)$$

- *Complete* the data
- Option 1:
  - In *every possible way* proportional to $P(z|x)$
  - Compute the solution from the completed data

Expectation Maximization for LGM



- *Complete* the data in *every possible way* proportional to $P(z|x)$
  - Compute the solution from the completed data
  - $\underset{A,D}{\operatorname{argmax}} \sum_{(x,z)} -\frac{1}{2}\log|D| - 0.5(x - Az)^T D^{-1}(x - Az)$
- The z values for each x are distributed according to $P(z|x)$. Segregating the summation by x

$$\underset{A,D}{\operatorname{argmax}} \sum_{x} \int_{-\infty}^{\infty} p(z|x)\left(-\frac{1}{2}\log|D| - 0.5(x - Az)^T D^{-1}(x - Az)\right) dz$$

LGM with incomplete information

$$\underset{A,D}{\operatorname{argmax}} \sum_x \int_{-\infty}^{\infty} p(z|x) \left(-\frac{1}{2} \log|D| - 0.5(x - Az)^T D^{-1} (x - Az) \right) dz$$

- Differentiating w.r.t A and $D$and equating to 0, we get an easy solution
- Solution for A

$$\nabla_A \sum_x \int_{-\infty}^{\infty} p(z|x)(x - Az)^T D^{-1}(x - Az)dz = 0 \quad \Rightarrow$$

$$\sum_x \int_{-\infty}^{\infty} p(z|x)(x - Az)z^T dz = 0 \quad \Rightarrow \quad A = \left(\sum_x \int_{-\infty}^{\infty} p(z|x) x z^T dz \right) \left(\sum_x \int_{-\infty}^{\infty} p(z|x) z z^T dz \right)^{-1}$$

- Solution for D

$$\nabla_D \left(N \log|D| + \sum_x \int_{-\infty}^{\infty} p(z|x)(x - Az)^T D^{-1}(x - Az)dz \right) = 0 \quad \Rightarrow$$

$$D = diag\left(\frac{1}{N} \left(\sum_x x x^T - A \sum_x \int_{-\infty}^{\infty} p(z|x) x z^T dz \right) \right)$$

These are closed form solutions, the details of which are not relevant to us.

Key: All terms integrate over *all possible completion of incomplete observations*, where the proportionality attached to any completion of x is P(z|x)

LGM with incomplete information

- It is actually an iterative algorithm (EM):
- Solution for A

$$A^{k+1} = \left(\sum_x \int_{-\infty}^{\infty} p(z|x; A^k, D^k) x z^T dz\right)\left(\sum_x \int_{-\infty}^{\infty} p(z|x; A^k, D^k) z z^T dz\right)^{-1}$$

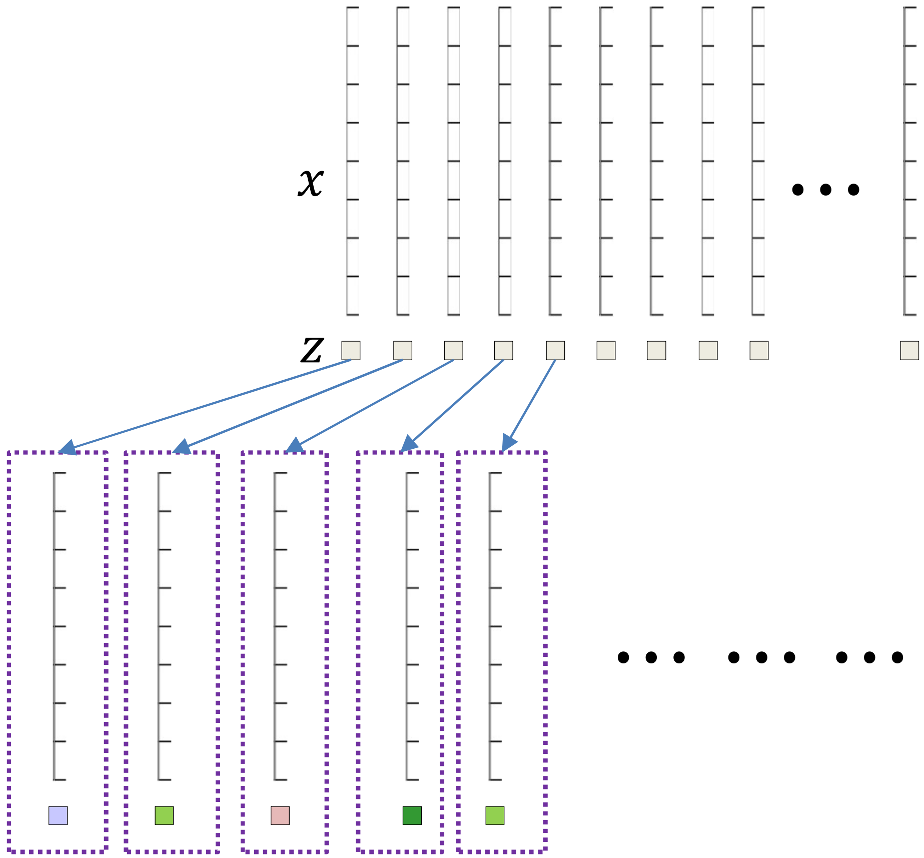
- Solution for D

$$D = diag\left(\frac{1}{N}\left(\sum_x x x^T - A \sum_x \int_{-\infty}^{\infty} p(z|x; A^k, D^k) x z^T dz\right)\right)$$

These are closed form solutions, the details of which are not relevant to us.

Key: All terms integrate over *all possible completion of incomplete observations*, where the proportionality attached to any completion of x is P(z|x)

Expectation Maximization for LGM



- *Complete* the data

$$P(z|x) = N(z; A^T(AA^T + D)^{-1}x, I - A^T(AA^T + D)^{-1}A)$$

- Option 2:
  - ***By drawing samples from*** $P(z|x)$
  - Compute the solution from the completed data

LGM from drawn samples

- Since we now have a collection of *complete vectors,* we can use the usual complete-data formulae
- Solution for A

$$A^{k+1} = \left(\sum_{(x,z)} xz^T\right)\left(\sum_{z} zz^T\right)^{-1}$$

- Solution for D

$$D^{k+1} = diag\left(\frac{1}{N}\left(\sum_{x} xx^T - A^k \sum_{(x,z)} xz^T\right)\right)$$

These are closed form solutions

Draw missing components from P(z|x; A^k, D^k) to *complete the data*

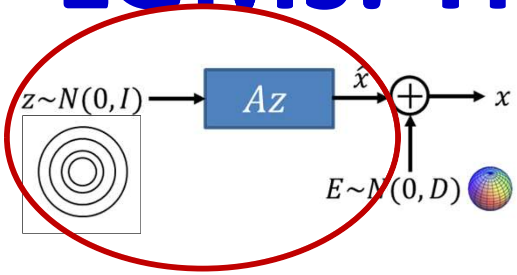
Estimate parameters from completed data

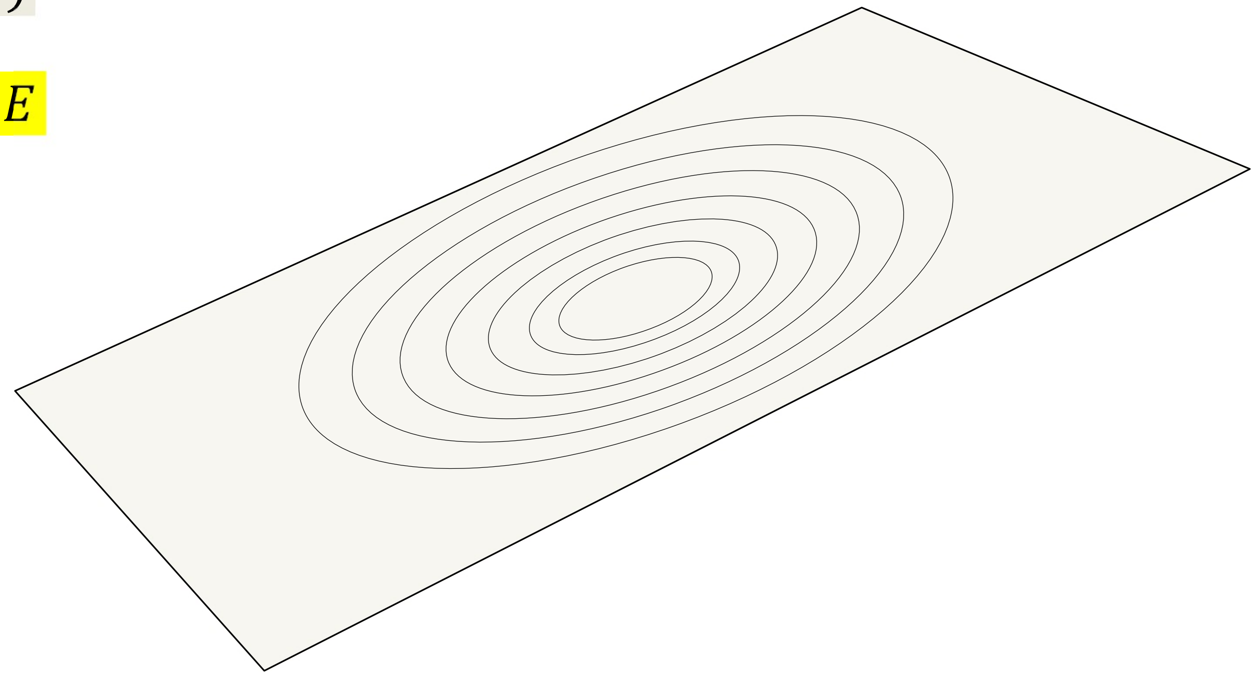
LGMs: The intuition

$z \sim N(0, I)$

$E \sim N(0, D)$

$x = Az + E$





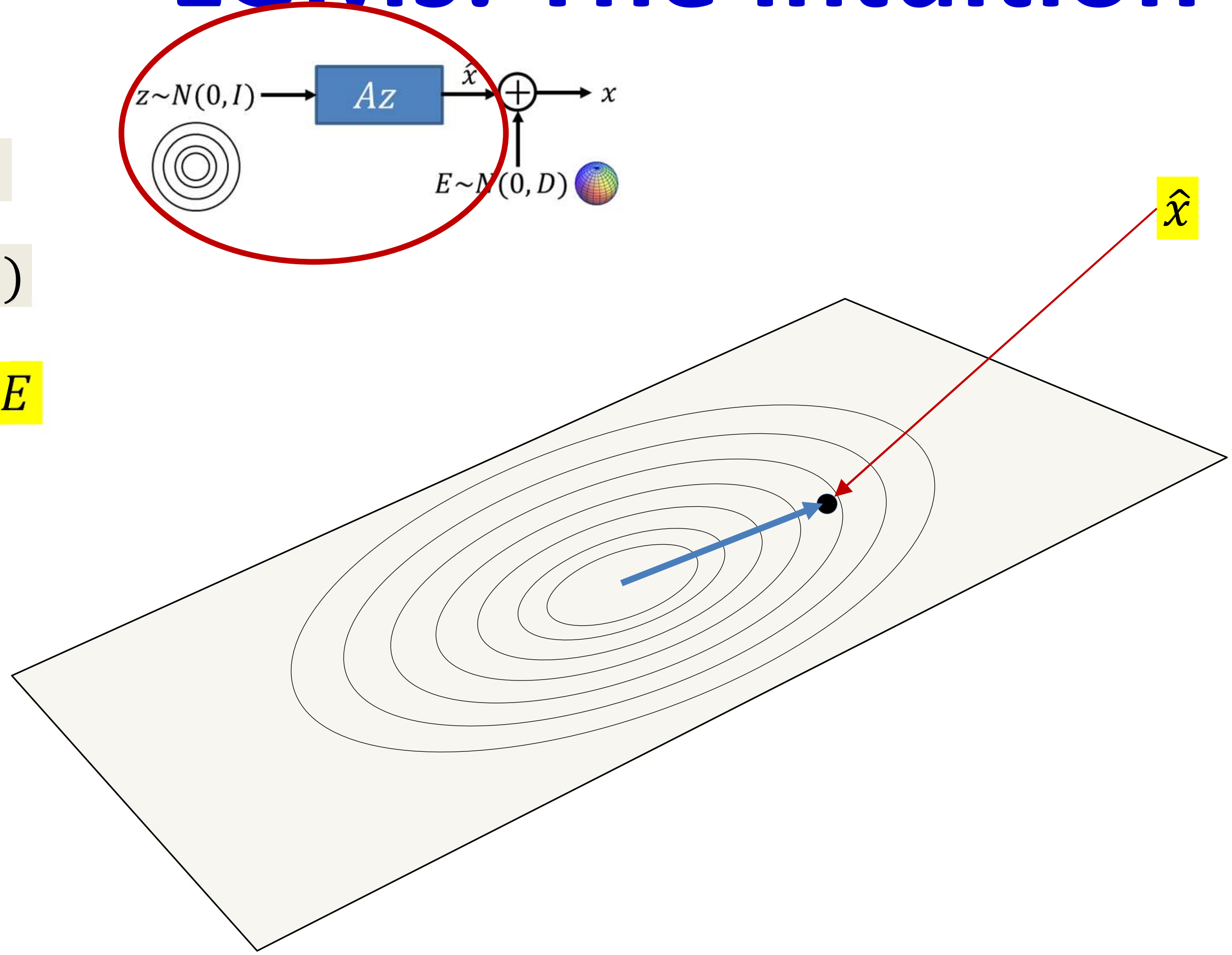
- The linear transform stretches and rotates the K-dimensional input space onto a K-dimensional hyperplane in the data space
- The isotropic Gaussian in the input space becomes a stretched and rotated Gaussian on the hyperplane

LGMs: The intuition

$z \sim N(0, I)$

$E \sim N(0, D)$

$x = Az + E$



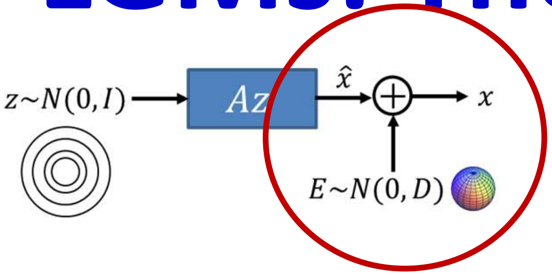
- Drawing samples: The first step places the z somewhere on the plane described by A
  - The distribution of points on the plane is also Gaussian

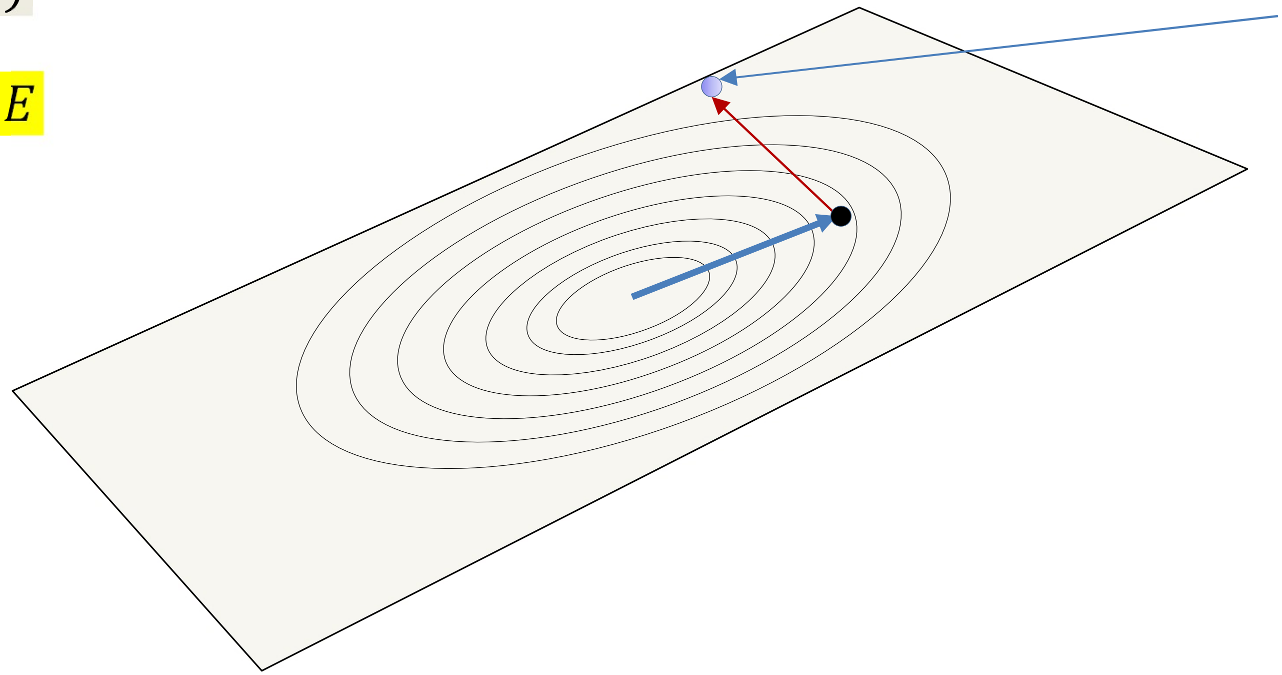
LGMs: The intuition

$z \sim N(0, I)$

$E \sim N(0, D)$

$X = Az + E$





$x = \hat{x} + E$

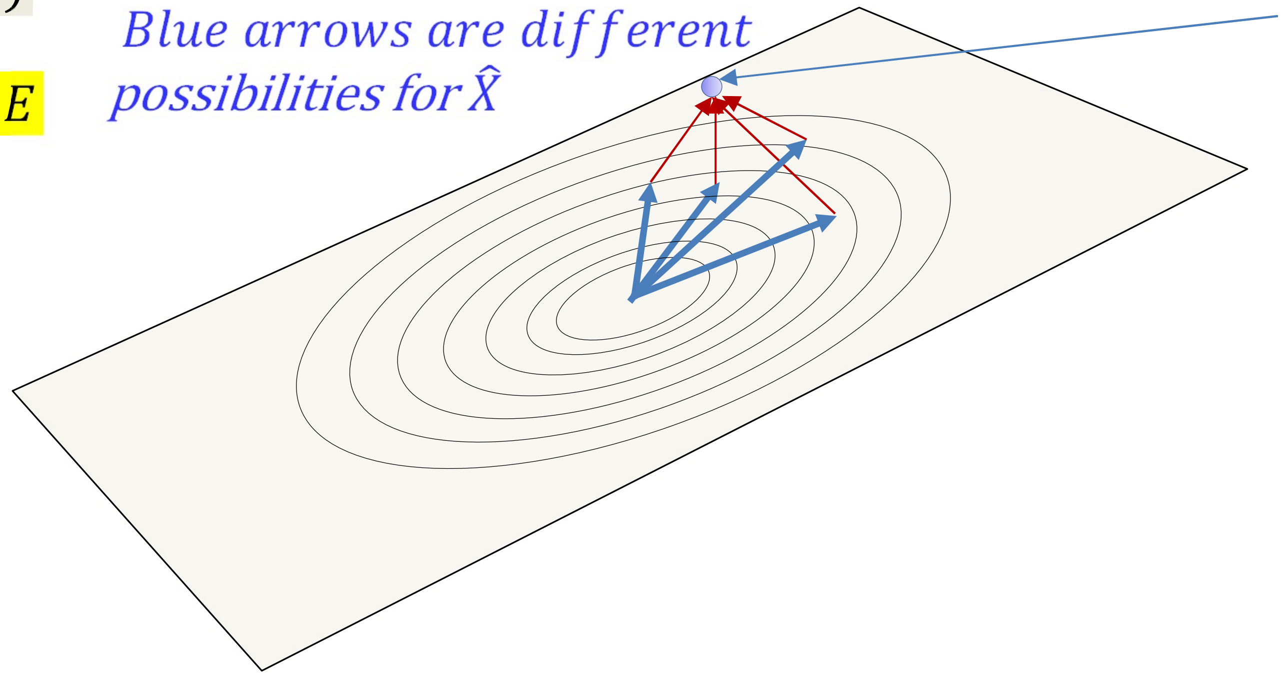
- LGM model: The first step places the z somewhere on the plane described by A
  - The distribution of points on the plane is also Gaussian
- Second step: Add Gaussian noise to produce points that aren't necessarily on the plane
  - Noise added is not revealed

EM for LGMs: The intuition

$z \sim N(0, I)$

$E \sim N(0, D)$

$X = Az + E$



Red arrows are different possibities for E

Blue arrows are different possibilities for $\hat{X}$

$X = \hat{X} + E$

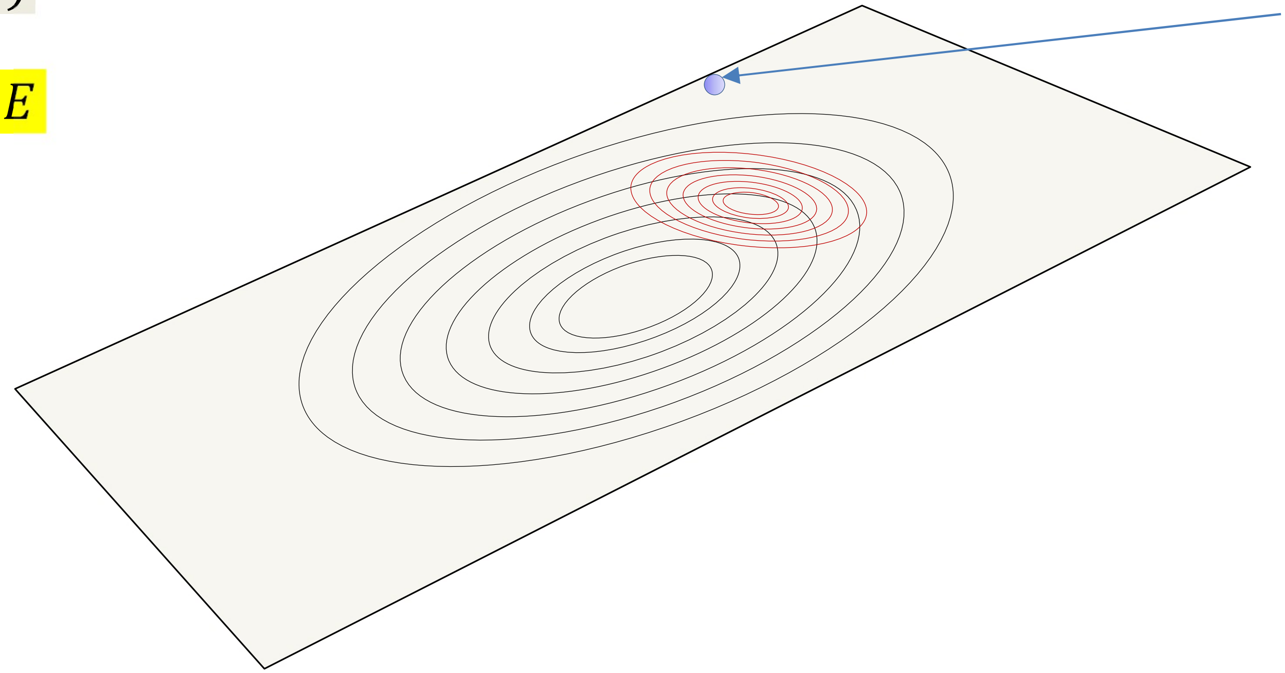
- In an LGM the way to produce any data instance is not unique
- Conversely, given only the data point, the “shadow” on the principal plane cannot be uniquely known

EM Solution

$z \sim N(0, I)$

$E \sim N(0, D)$

$x = Az + E$



$x = \hat{x} + E$

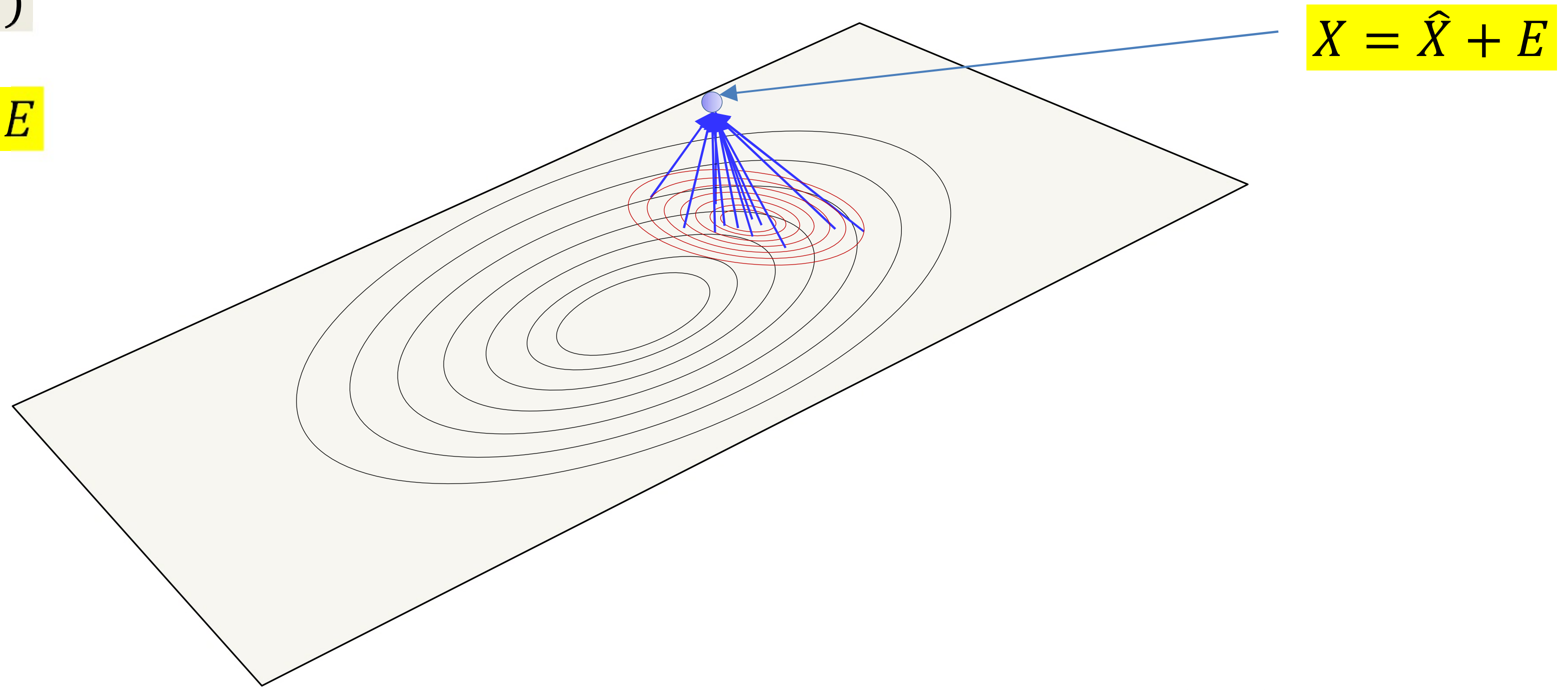
- The posterior probability $P(z|x)$ gives you the location of all the points on the plane that *could* have generated x and their probabilities

EM Solution

$z \sim N(0, I)$

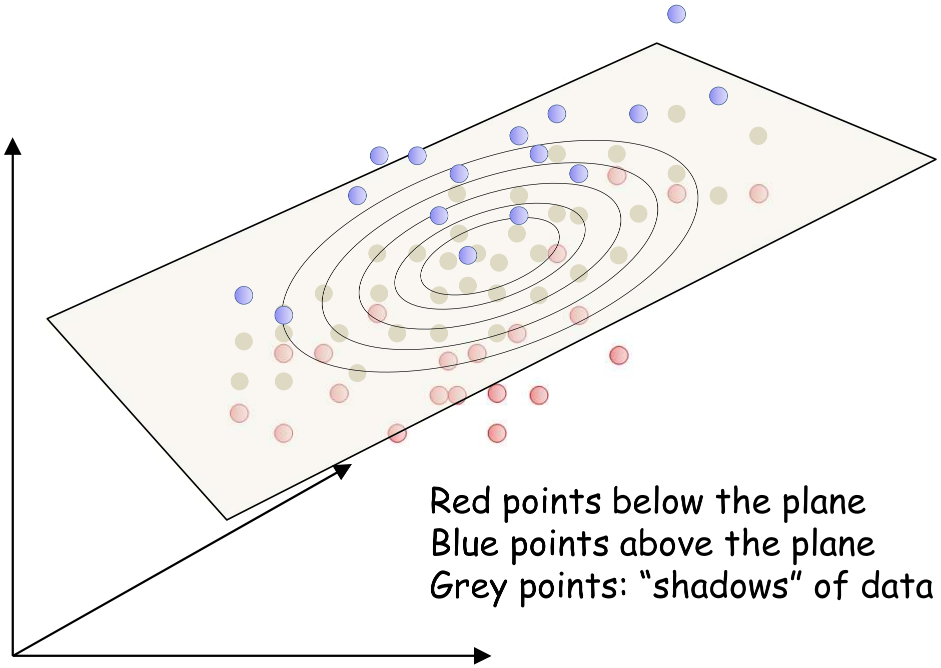
$E \sim N(0, D)$

$X = Az + E$


$X = \hat{X} + E$

- Attach the point to *every* location on the plane, according to $P(z|x)$
  - Or to a sample of points on the plane drawn from $P(z|x)$
- There will be more attachments where $P(z|x)$ is higher, and fewer where it is lower

EM Solution

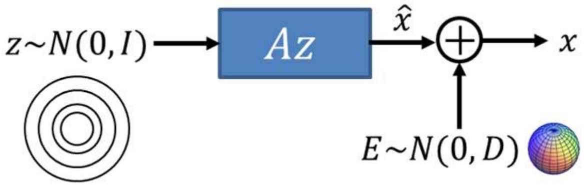


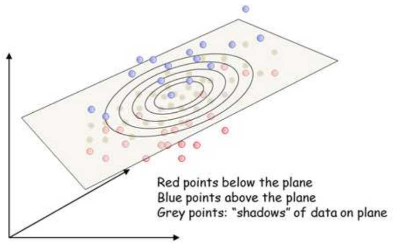
Red points below the plane
Blue points above the plane
Grey points: “shadows” of data on plane

- Attach *every* training point in this manner
- Let the plane rotate and stretch until the total tension (sum squared length) of all the attachments is minimize
- Repeat attachment and rotation until convergence…

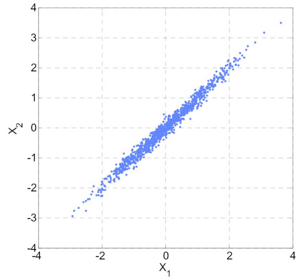
Summarizing LGMs

- LGMs are models for *Gaussian* distributions
- Specifically, they model the distribution of data as Gaussian, where most of the variation is along a *linear* manifold
  - They do this by transforming a Gaussian RV z through a linear transform $f(z) = Az$ that transforms the K-dim input space of z into a K-dimensional hyperplane (linear manifold) in the data space

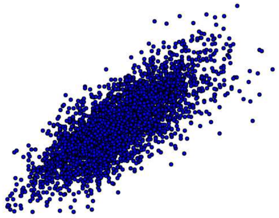


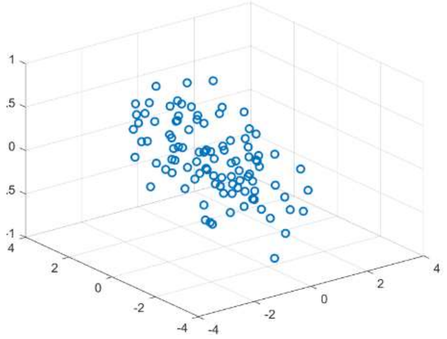


Red points below the plane
Blue points above the plane
Grey points: "shadows" of data on plane



- They are excellent models for data that actually fit these assumptions
  - Often, we can simply assume that data lie near linear manifolds and model them with LGMs
  - PCA, an instance of LGMs, is very popular





Poll 2 (@1845)

Choose all that are true about Factor Analysis (Linear Gaussian Models)

- It models the distribution of the data as a Gaussian centered on a principal (hyper)plane
- The generative model is that in order to generate any point, the process first takes a Gaussian step on the principal hyperplane, followed by addition of Gaussian noise.
- The parameters of the distribution are the bases of the hyperplane and the covariance of the noise.
- The parameters can be easily estimated if the location of the first step on the principal hyperplane is known for every data point
- The actual estimation is performed using EM, which iteratively “completes” each data instance with the location of this first step, and then estimates the parameters

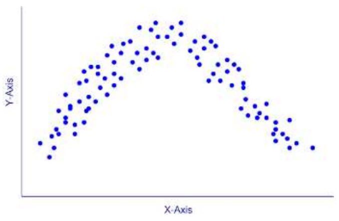
Poll 2

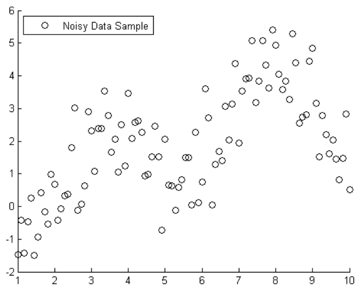
Choose all that are true about Factor Analysis (Linear Gaussian Models)

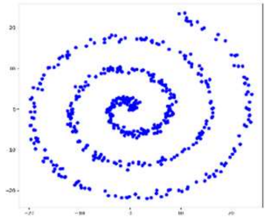
- It models the distribution of the data as a Gaussian centered on a principal (hyper)plane
- The generative model is that in order to generate any point, the process first takes a Gaussian step on the principal hyperplane, followed by addition of Gaussian noise.
- The parameters of the distribution are the bases of the hyperplane and the covariance of the noise.
- The parameters can be easily estimated if the location of the first step on the principal hyperplane is known for every data point
- The actual estimation is performed using EM, which iteratively "completes" each data instance with the location of this first step, and then estimates the parameters

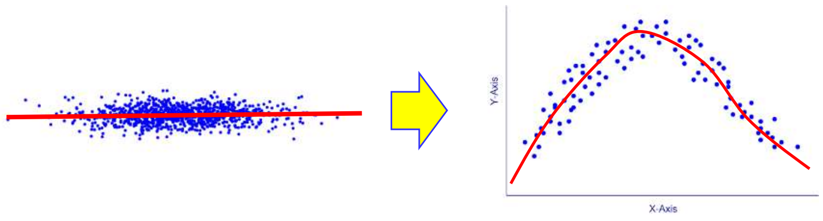
Where LGMs fail

- What about data that are not Gaussian distributed close to a plane
  - The distributions lie close to a curved or otherwise non-linear manifold?
- You can model these as Gaussian data centered on a plane that has been *warped into the observed shape*



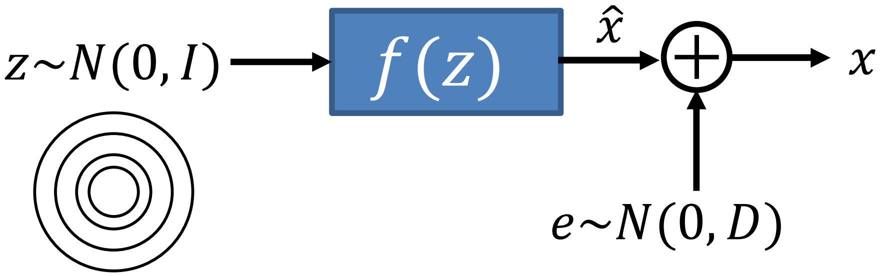






NLGMs

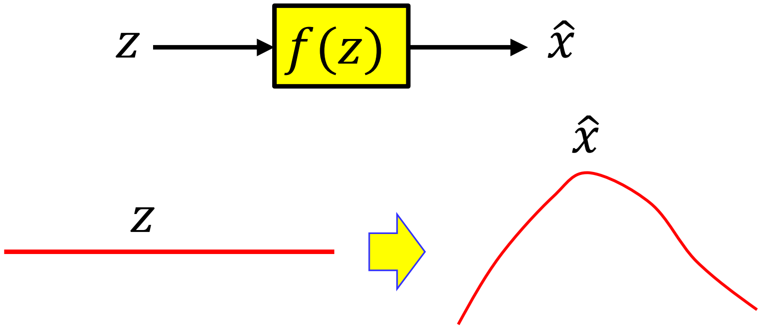
$x = f(z) + e$



- The *non-linear* Gaussian model
- $f(z)$ is a *non-linear function* that produces a *curved* manifold
  - Like the decoder of a non-linear AE
- The samples of z are placed on this curved manifold
- The actual data are produced by adding noise to samples on the manifold

NLGMs

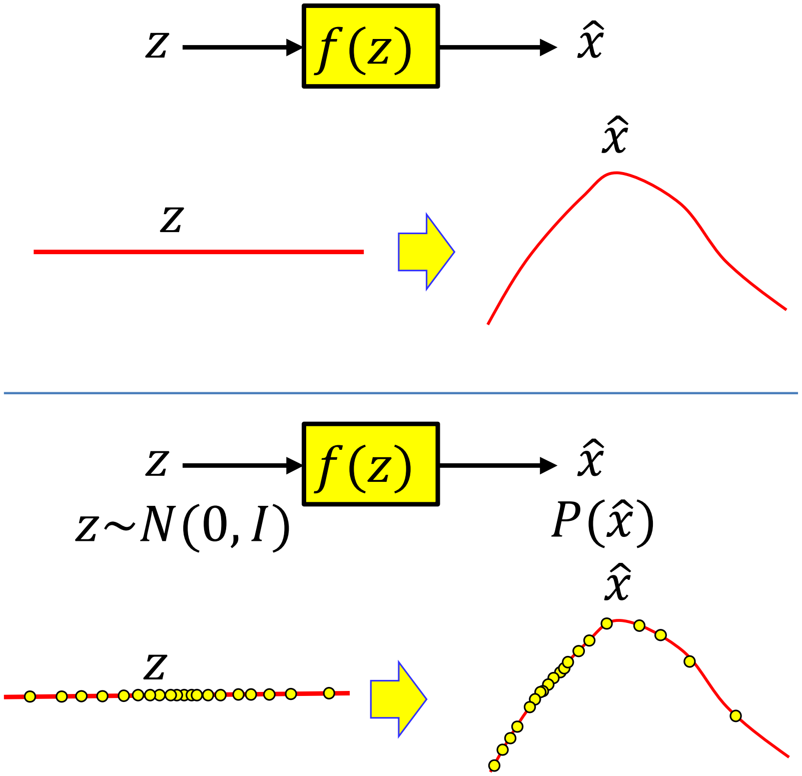
- **Step1:** Find a function that warps a lower-dimensional input plane to the target manifold in the data space
  - The non-linear version of the linear transform in the LGM



NLGMs

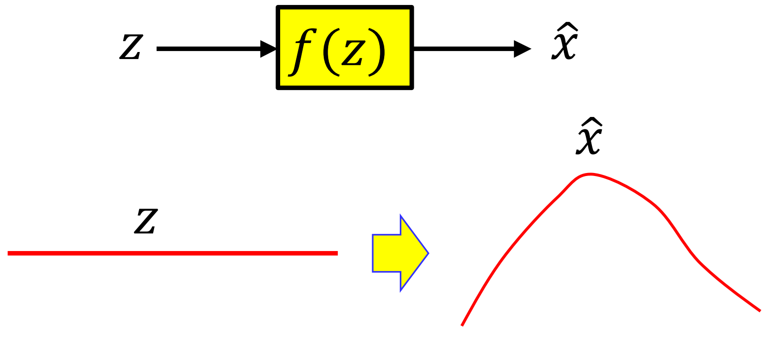
- **Step1:** Find a function that warps a lower-dimensional input plane to the target manifold in the data space
  - The non-linear version of the linear transform in the LGM

- **Step2:** Transform a Gaussian distribution on the input plane to a distribution on the curved manifold

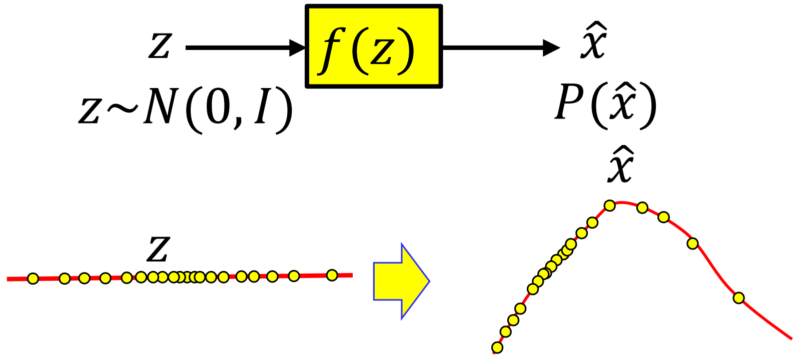


NLGMs

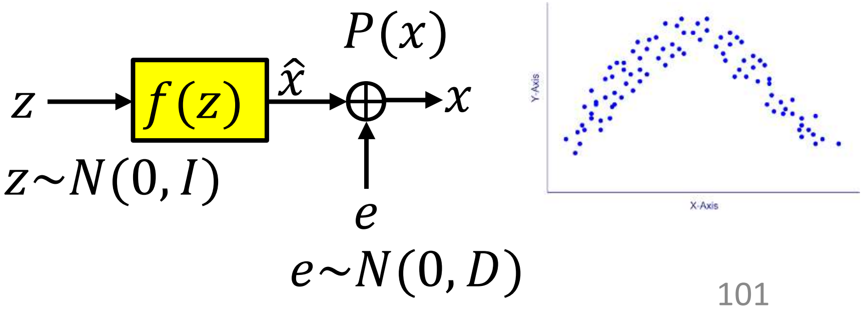
- **Step1:** Find a function that warps a lower-dimensional input plane to the target manifold in the data space
  - The non-linear version of the linear transform in the LGM



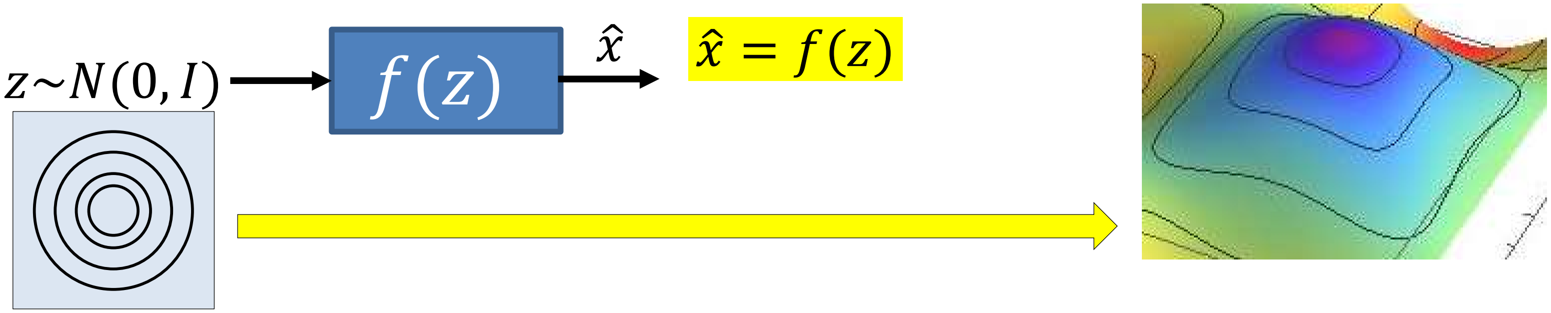
- **Step2:** Transform a Gaussian distribution on the input plane to a distribution on the curved manifold



- **Step3:** Add some uncorrelated Gaussian "fuzz" to account for off-manifold variations



NLGMs



$z \sim N(0, I)$

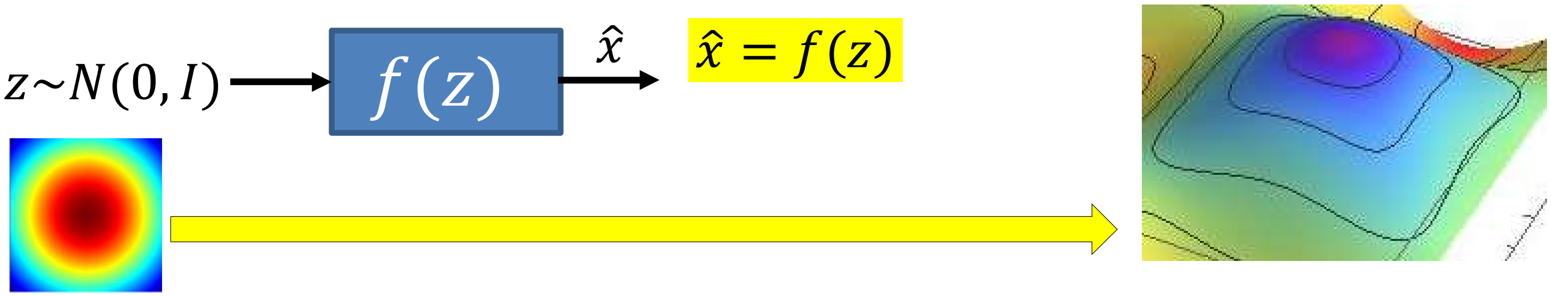
$f(z)$

$\hat{x}$

$\hat{x} = f(z)$

- The non-linear function warps the input space into a curved manifold in the data space

NLGMs



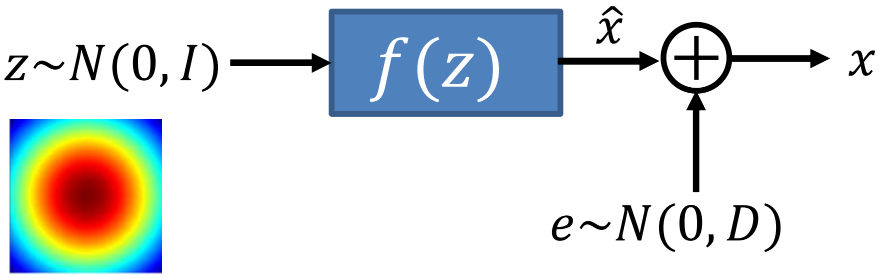
$z \sim N(0, I)$ → $f(z)$ → $\hat{x}$

$\hat{x} = f(z)$

- The non-linear function warps the input space into a curved manifold in the data space
  - The variable z at the input is transformed to the variable $\hat{x}$ on this manifold
  - The distribution of $\hat{x}$ on the manifold will follow the distribution of z
    - High-density regions of $\hat{x}$ correspond to high-density regions of z

NLGMs

$x = f(z) + e$



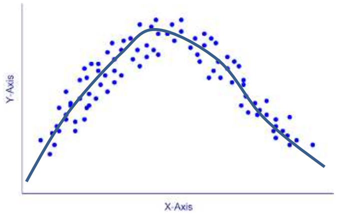
$z \sim N(0, I)$

$f(z)$

$\hat{x}$

x

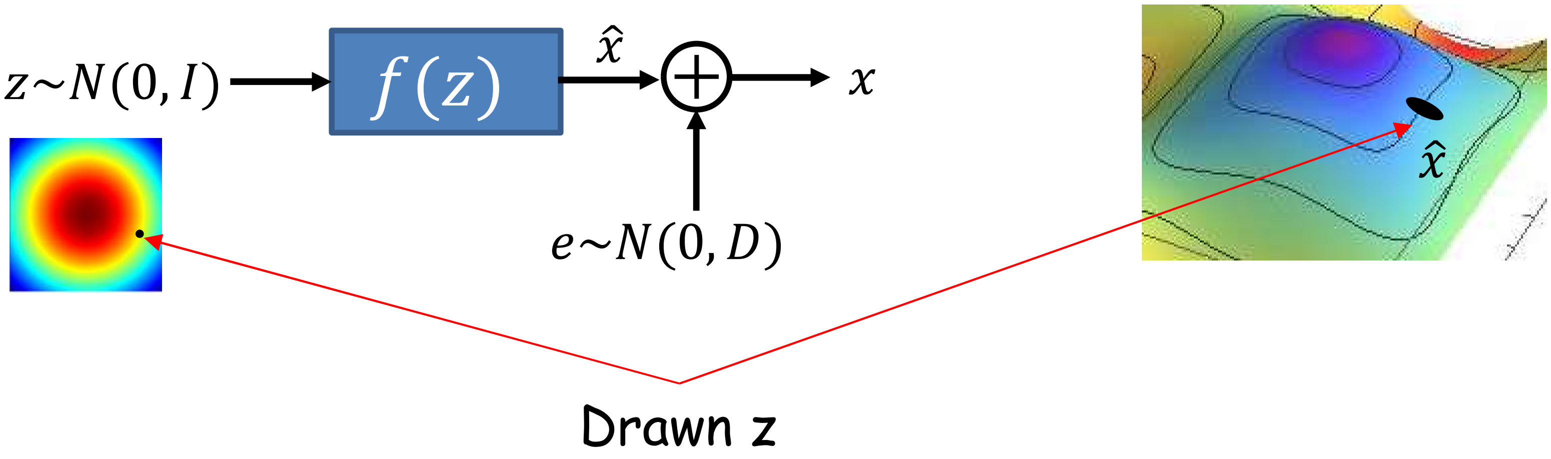
$e \sim N(0, D)$



- The non-linear function warps the input space into a curved manifold in the data space
  - The variable z at the input is transformed to the variable $\hat{x}$ on this manifold
  - The distribution of $\hat{x}$ on the manifold will follow the distribution of z
    - High-density regions of $\hat{x}$ correspond to high-density regions of z
- The final observations are obtained by adding uncorrelated full-dimensional Gaussian noise to $\hat{x}$

NLGM Generating Process

$x = f(z) + e$

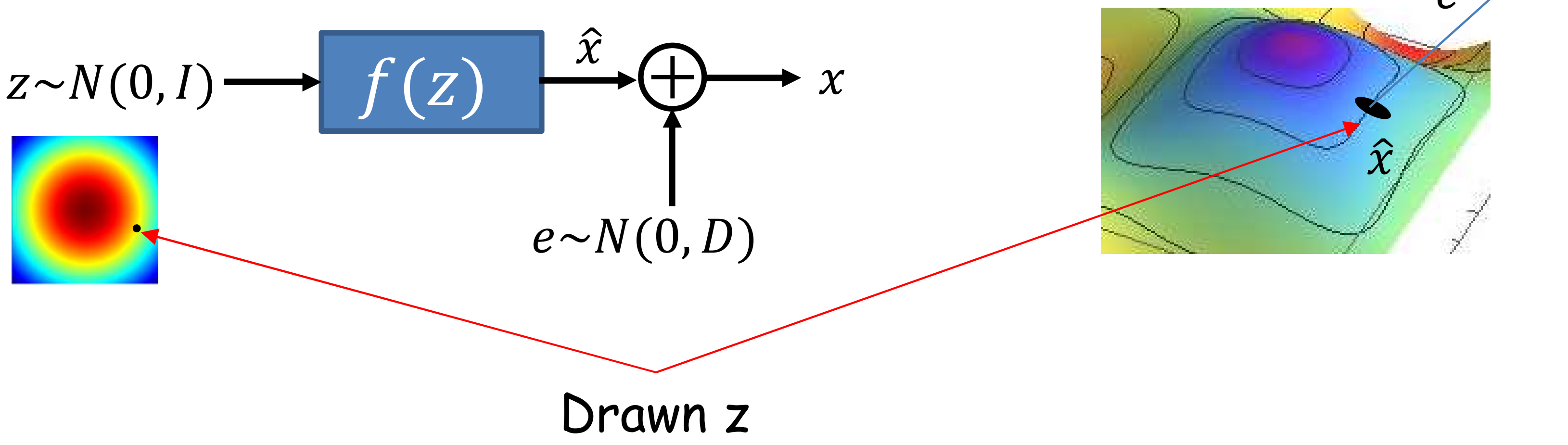


Drawn z

- Generating process:
  - Draw a sample z from a Standard Gaussian
  - Transform z by $f(z)$
    - This places z on the curved manifold

NLGMs Generating Process

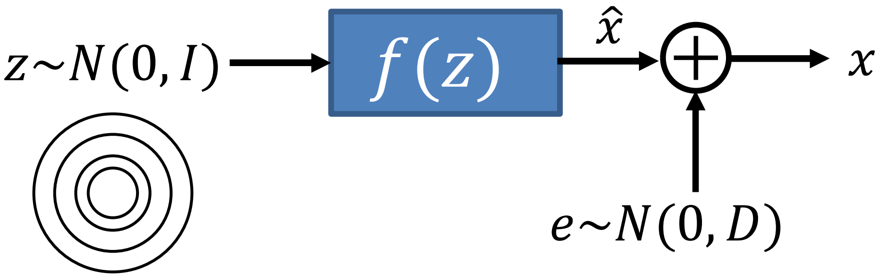
$$x = f(z) + e$$

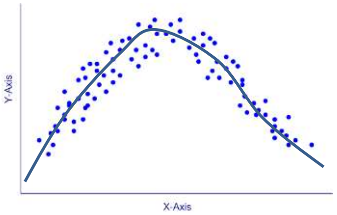


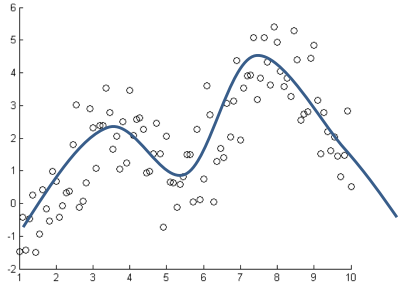
- Generating process:
  - Draw a sample z from a Standard Gaussian
  - Transform z by $f(z)$
    - This places z on the curved manifold
  - Add uncorrelated Gaussian noise to get the final observation

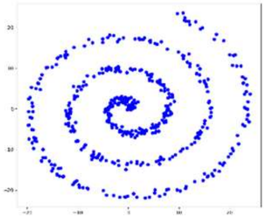
NLGMs

$x = f(z) + e$





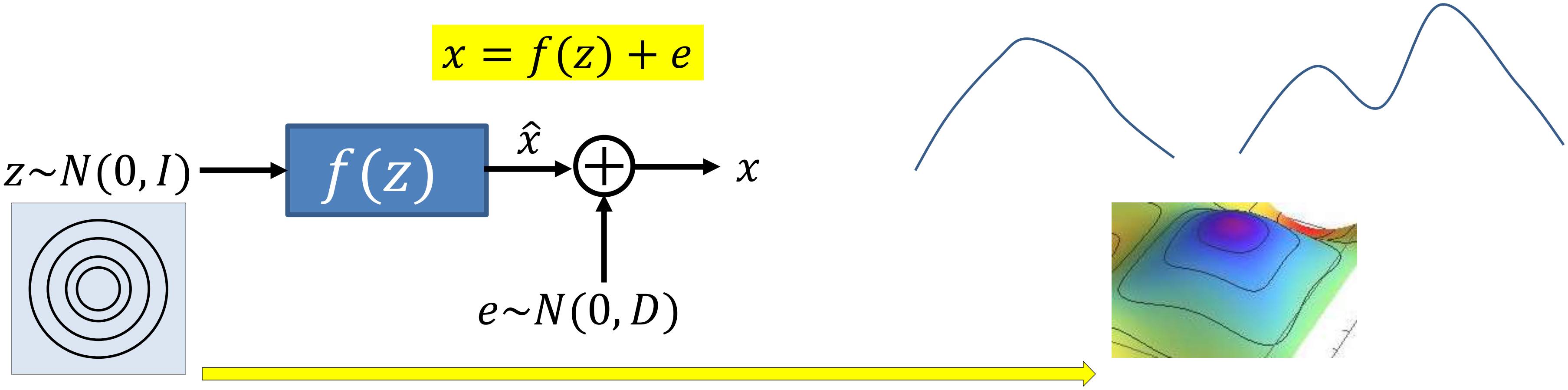




- The NLGM can model very complicated distributions
  - Distributions that may be viewed as lying close to a curved K-dimensional surface in the data space
    - Or even a linear surface: $f(z) = Az$ is a special case
    - K is the dimensionality of z

NLGMs

$$x = f(z) + e$$



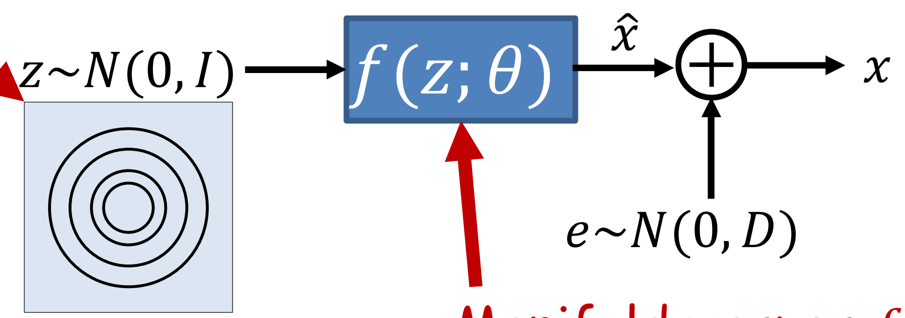
- The NLGM can model very complicated distributions
  - Distributions that may be viewed as lying close to a curved K-dimensional surface in the data space
    - Or even a linear surface: $f(z) = Az$ is a special case
    - K is the dimensionality of z

- Key requirement:
  - Identifying the dimensionality K of the curved manifold
  - Having a function $f(z)$ that can transform the (linear) K-dimensional input space (space of z) to the desired K-dimensional manifold in the data space

Designing NLGMs

K = dimensionality of z

$x = f(z) + e$



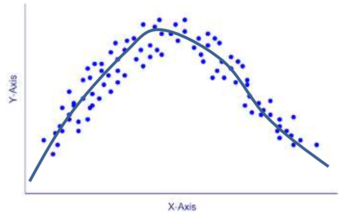
$z \sim N(0, I)$

$f(z; \theta)$

$\hat{x}$

x

$e \sim N(0, D)$

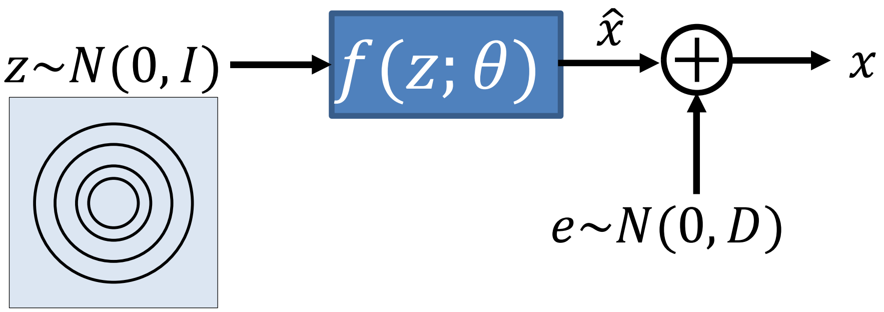


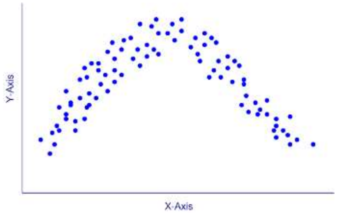
Manifold warper $f(z)$

- Key design issues:
  - – Select (or guess) the dimensionality of the manifold
    - This is the dimensionality of z
  - – Choosing the right function $f(z)$ that is capable of *learning* the shape of the manifold
    - We will choose a Neural Network $f(z; \theta)$

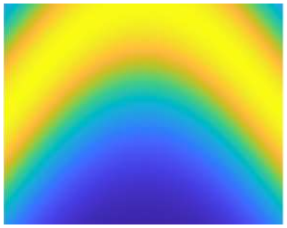
Learning the NLGM

$x = f(z) + e$



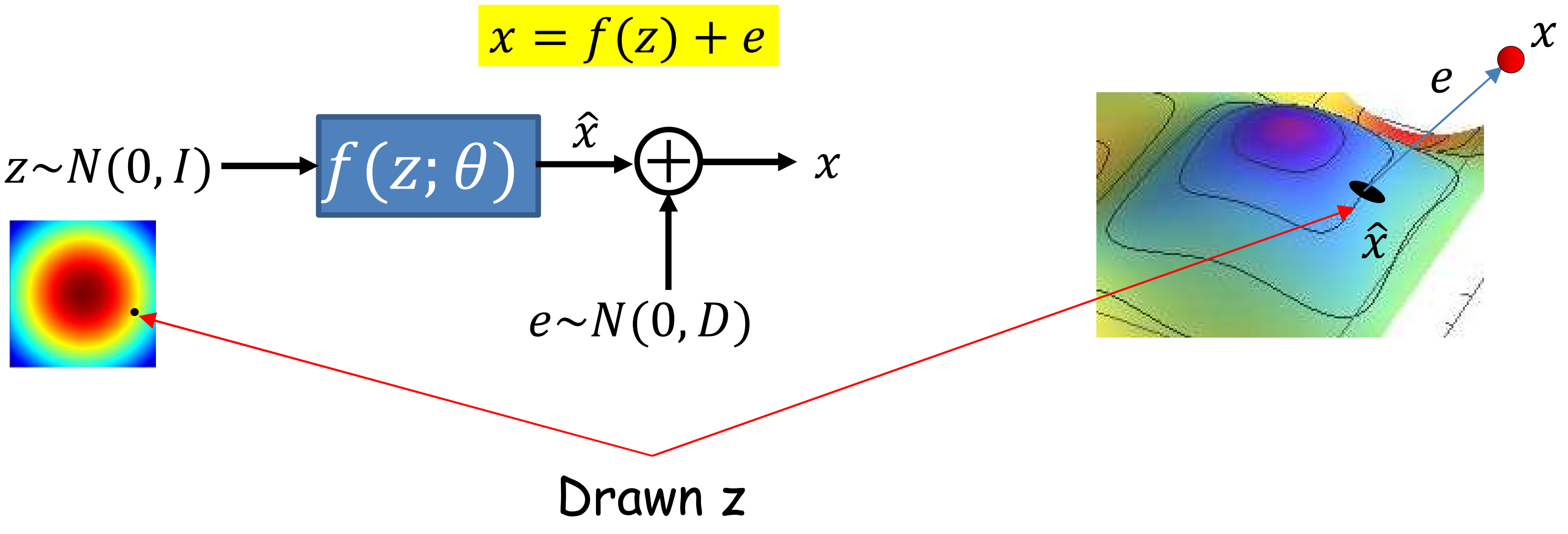


P(x)



- Given a collection of training data $X = \{x\}$
  - Estimate the parameters θ of $f(z; \theta)$
  - Estimate D
- The NLGM is a generative model that actually models a distribution
  - The distribution obtained when $N(0, I)$ is transformed by $f(z)$
- We will use ML estimation to learn its parameters to best match the training data

Probabilities modelled by the NLGM



$x = f(z) + e$

$z \sim N(0, I) \rightarrow \boxed{f(z;\theta)} \xrightarrow{\hat{x}} \oplus \rightarrow x$

$e \sim N(0, D)$

Drawn z

- The conditional probability of x given z

$$P(x|z) = N(x;\ f(z;\theta), D)$$

- The marginal probability of x

$$P(x) = \int_{-\infty}^{\infty} P(x|z)P(z)dz = \int_{-\infty}^{\infty} N(x;\ f(z;\theta), D)\ N(z; 0, D)\ dz$$

  - For most nonlinear functions $f(z;\theta)$ this math is not tractable, and we cannot get a closed form for $P(x)$
  - ***That won't prevent us from being able to estimate θ and D***

Poll 3 (@1846)

Choose all that are true of Nonlinear Gaussian Models

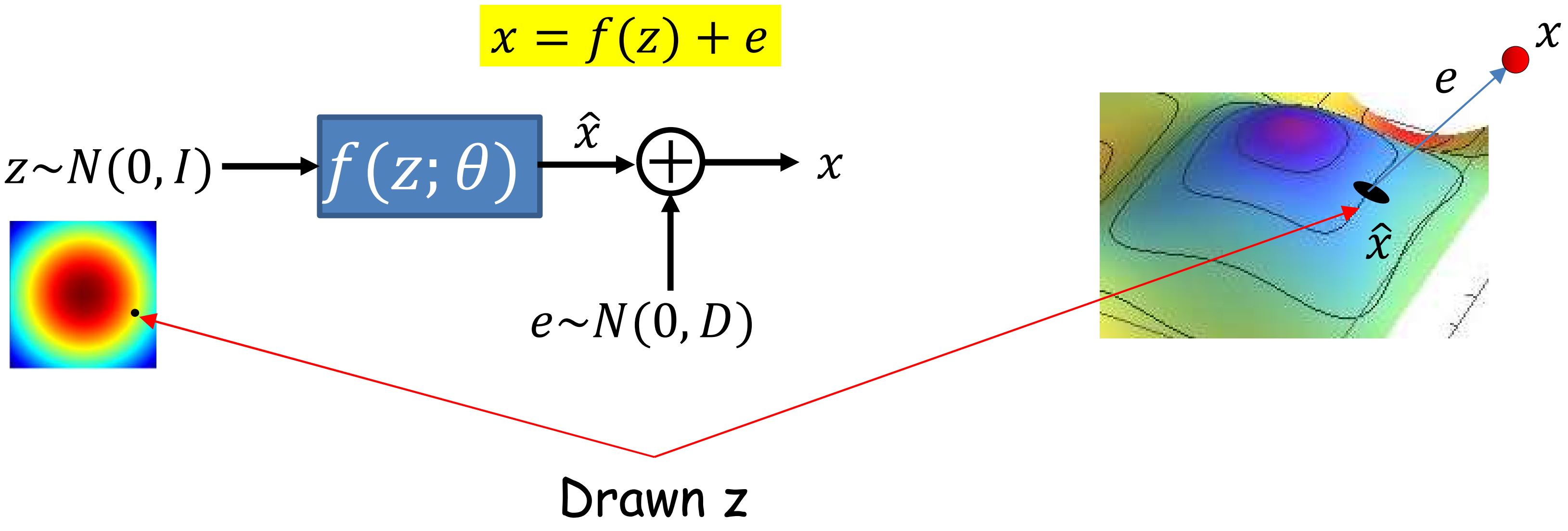
- They are generative models
- They model the distribution of the data as Gaussian distributed about a curved manifold
- The generative model is that in order to generate any point, the process first takes a Gaussian step on along the curved manifold, followed by the addition of Gaussian noise
- The parameters of the distribution are the parameters of the function that transform a K-dimensional plane into a K-dimensional manifold, and the covariance of the noise
- The NLGM is the decoder component of a variational autoencoder

Poll 3

Choose all that are true of Nonlinear Gaussian Models

- **They are generative models**
- **They model the distribution of the data as Gaussian distributed about a curved manifold**
- **The generative model is that in order to generate any point, the process first takes a Gaussian step on along the curved manifold, followed by the addition of Gaussian noise**
- **The parameters of the distribution are the parameters of the function that transform a K-dimensional plane into a K-dimensional manifold, and the covariance of the noise**
- **The NLGM is the decoder component of a variational autoencoder**

Learning the NLGM with *complete data*



$x = f(z) + e$

$z \sim N(0, I)$

$f(z;\theta)$

$\hat{x}$

x

$e \sim N(0, D)$

e

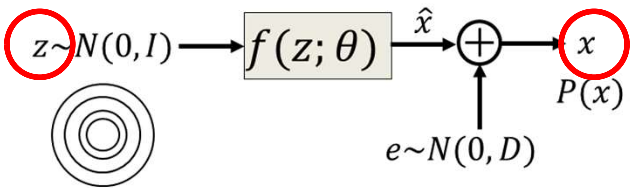
x

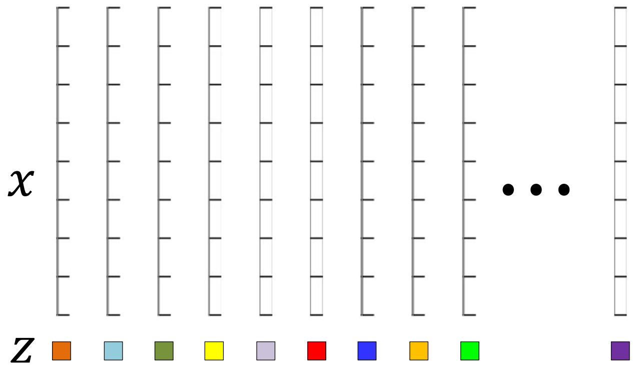
$\hat{x}$

Drawn z

- Drawing a sample from the NLGM is a two-step process
  - First a z is drawn
    - And transformed
  - Then an e is drawn
    - And added
- The complete data to describe any draw are the outcomes of every stage of the drawing process, i.e. (x, z)
  - Actually (x, e, z), but let's work without e

NLGM with complete data

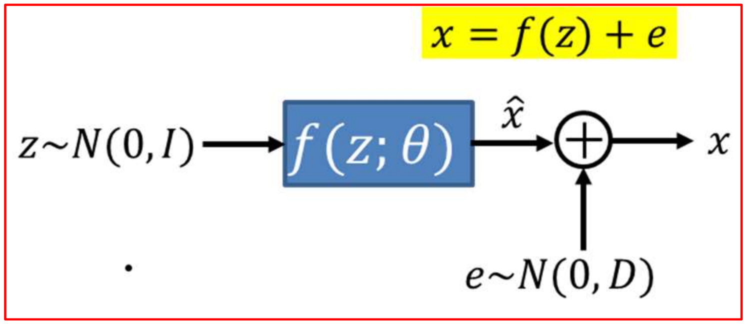




- Let us first consider a *glass-box* process that gives us *complete* data
  - The output and the intermediate steps of the generation process
  - I.e. both the x and the z for every draw

- We will derive estimation rules for the model parameters using the complete data

ML estimation with complete information



$$x = f(z;\theta) + e$$
$$P(x|z) = N(f(z;\theta), D)$$

- Given complete information $X = [x_1, x_2, \ldots],\ \ Z = [z_1, z_2, \ldots]$

$$\theta^*, D^* = \underset{\theta,D}{\operatorname{argmax}} \sum_{(x,z)} \log P(x,z) \ = \underset{\theta,D}{\operatorname{argmax}} \sum_{(x,z)} \log P(x|z)$$

$$= \underset{\theta,D}{\operatorname{argmax}} \sum_{(x,z)} \log \frac{1}{\sqrt{(2\pi)^d |D|}} \exp(-0.5(x - f(z;\theta))^T D^{-1} (x - f(z;\theta)))$$

$$= \underset{\theta,D}{\operatorname{argmax}} \sum_{(x,z)} -\frac{1}{2} \log|D| - 0.5(x - f(z;\theta))^T D^{-1} (x - f(z;\theta))$$

NLGM with complete information

$$\theta^*, D^* = \underset{\theta,D}{\operatorname{argmax}} \sum_{(x,z)} -\frac{1}{2}\log|D| - 0.5(x - f(z;\theta))^T D^{-1}(x - f(z;\theta))$$

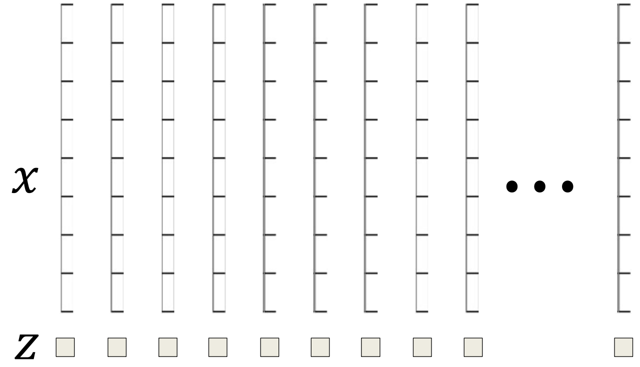
- There isn't a nice closed form solution, but we could learn the parameters using backpropagation, which minimizes the following loss

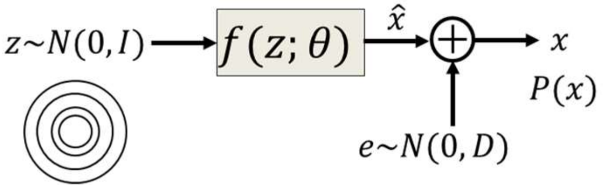
$$L(\theta, D) = \sum_{(x,z)} \frac{1}{2}\log|D| + 0.5(x - f(z;\theta))^T D^{-1}(x - f(z;\theta))$$

$$\theta^*, D^* = \underset{\theta,D}{\operatorname{argmin}}\, L(\theta, D)$$

Unfortunately we do not observe z.
It is missing; the observations are incomplete

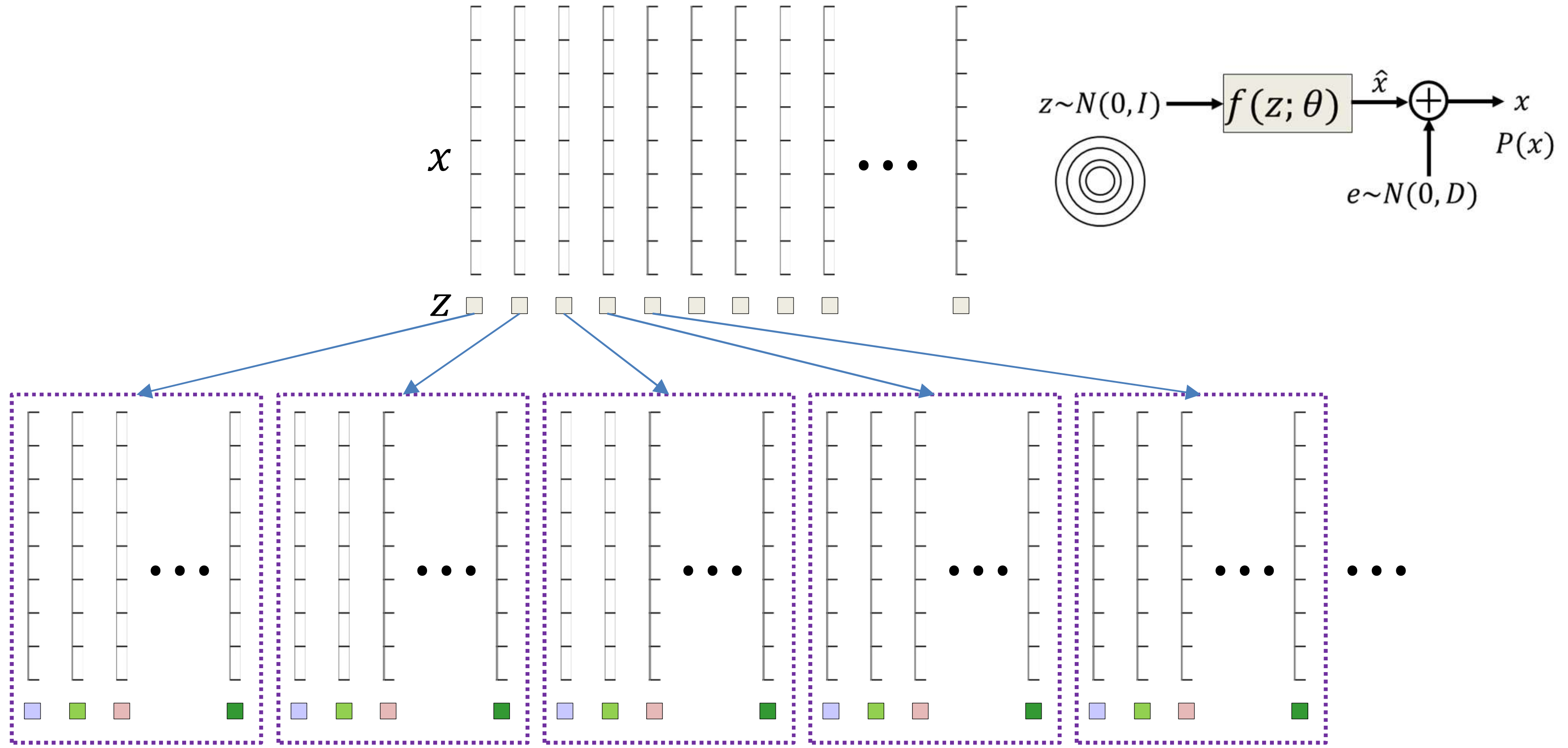
NLGM with incomplete data





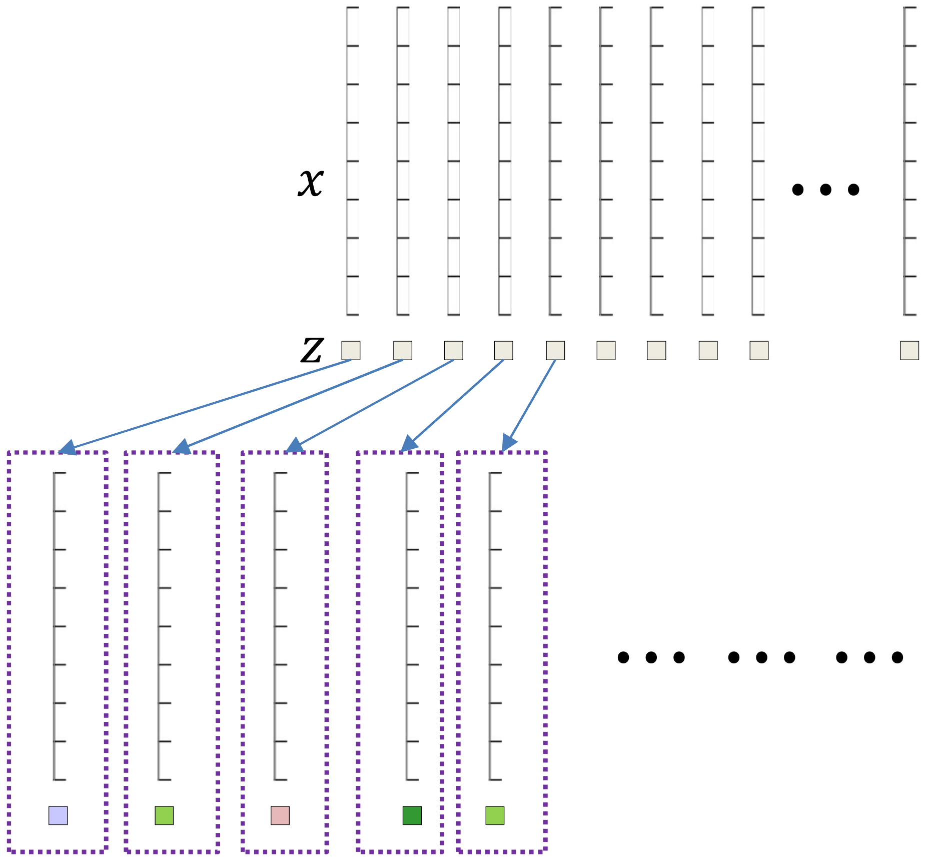
- We could estimate the model parameters if z were known for every data observation
  - I.e. if the data were complete
- Unfortunately, we don't know z
  - The data are incomplete
- Solution: *Complete the data*

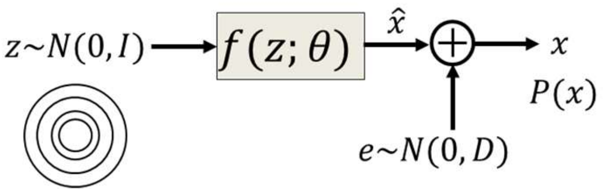
Expectation Maximization for NLGM



- *Complete* the data
- Option 1:
  - In *every possible way* proportional to $P(z|x)$
  - Compute the solution from the completed data

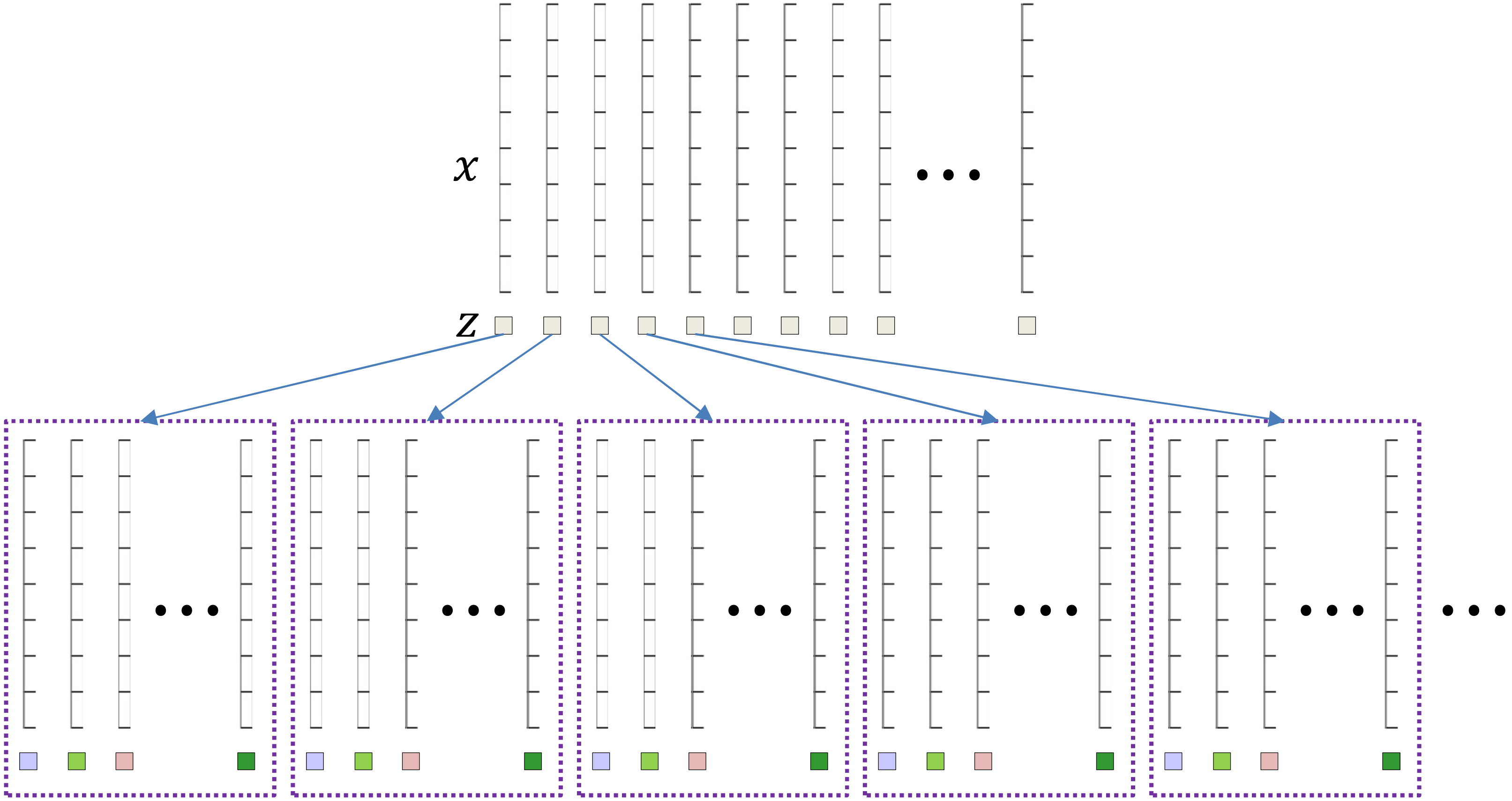
Expectation Maximization for LGM





- *Complete* the data
- Option 2:
  - ***By drawing samples from*** $P(z|x)$
  - Compute the solution from the completed data

Expectation Maximization for LGM



- *Complete* the data

Using *every* possible value (option 1) is to be preferred over sampling (option 2) if the former produces tractable closed form solutions. Otherwise we must use option 2.

- Option 2:
  - ***By drawing samples from*** $P(z|x)$
  - Compute the solution from the completed data

Problem with completing the data

- The posterior probability is given by

$$P(z|x) = \frac{P(x|z)P(z)}{P(x)}$$

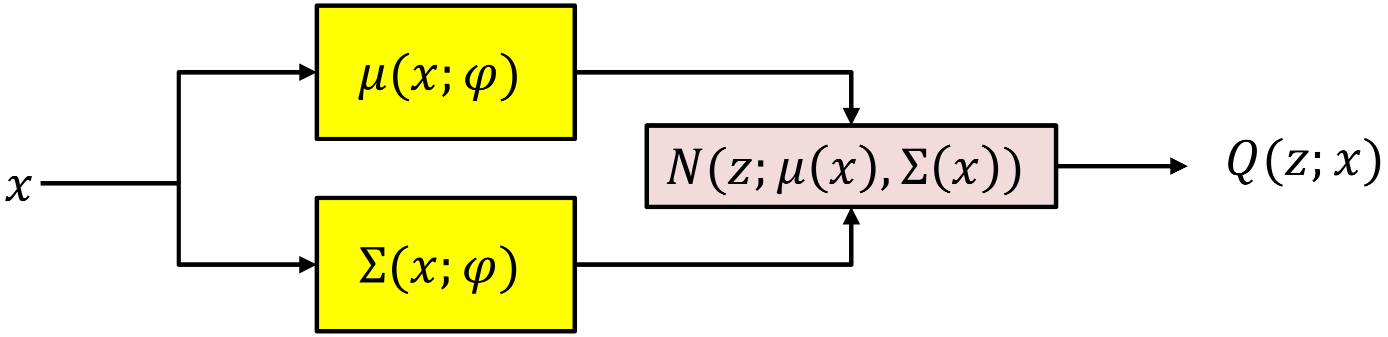
- The denominator

$$P(x) = \int_{-\infty}^{\infty} N(x;\ f(z;\theta), D)\ N(z; 0, D)\ dz$$

  - This is intractable to compute in closed form for most $f(z;\theta)$

- $P(z|x)$ is intractable as a closed form solution
  - Makes it challenging to integrate over it or draw samples from it
  - But we could try to *approximate it with a* $Q(z)$

Approximating $P(z|x)$



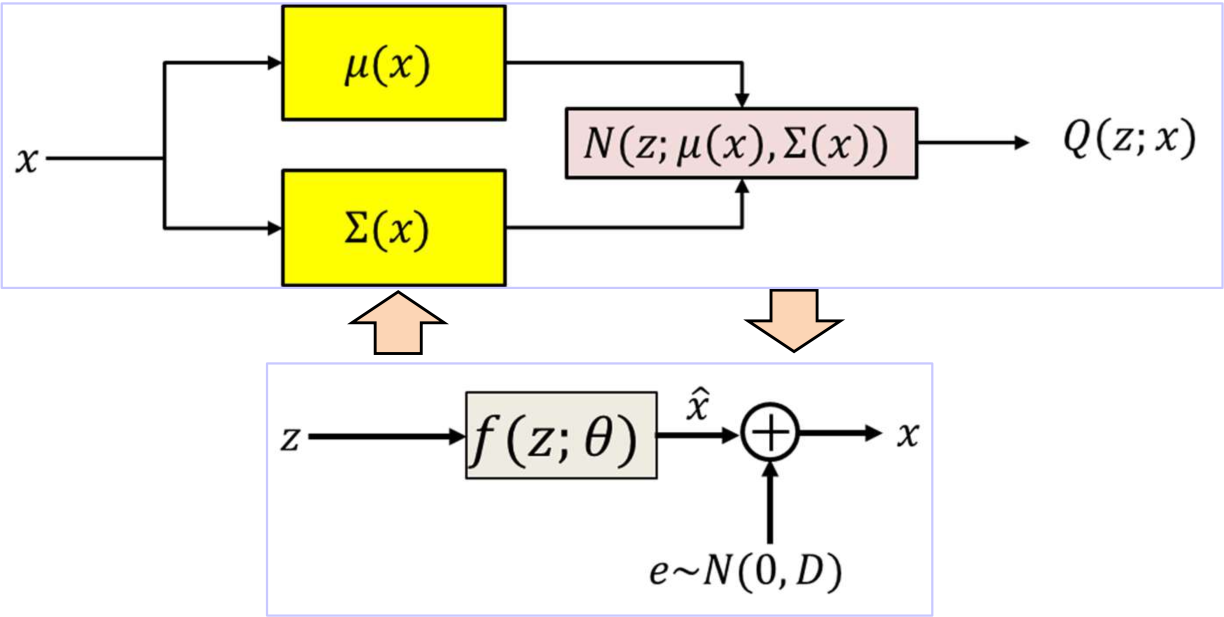
$\mu(x;\varphi)$ and $\Sigma(x;\varphi)$ are parametric functions of x, with parameters that we jointly represent as φ

- We will approximate $P(z|x)$ as

$$P(z|x) \approx Q(z,x) \ = \ Gaussian\ N(z;\mu(x),\Sigma(x))$$

  - where $\mu(x;\varphi)$ and $\Sigma(x;\varphi)$ are estimated such that $Q(z,x)$ approximates $P(z|x)$ as closely as possible
  - For convenience, we will assume $\Sigma(x;\varphi)$ is a diagonal matrix, represented entirely by its diagonal elements
- We will use $Q(z,x)$ as our proxy for $P(z|x)$

Overall Solution

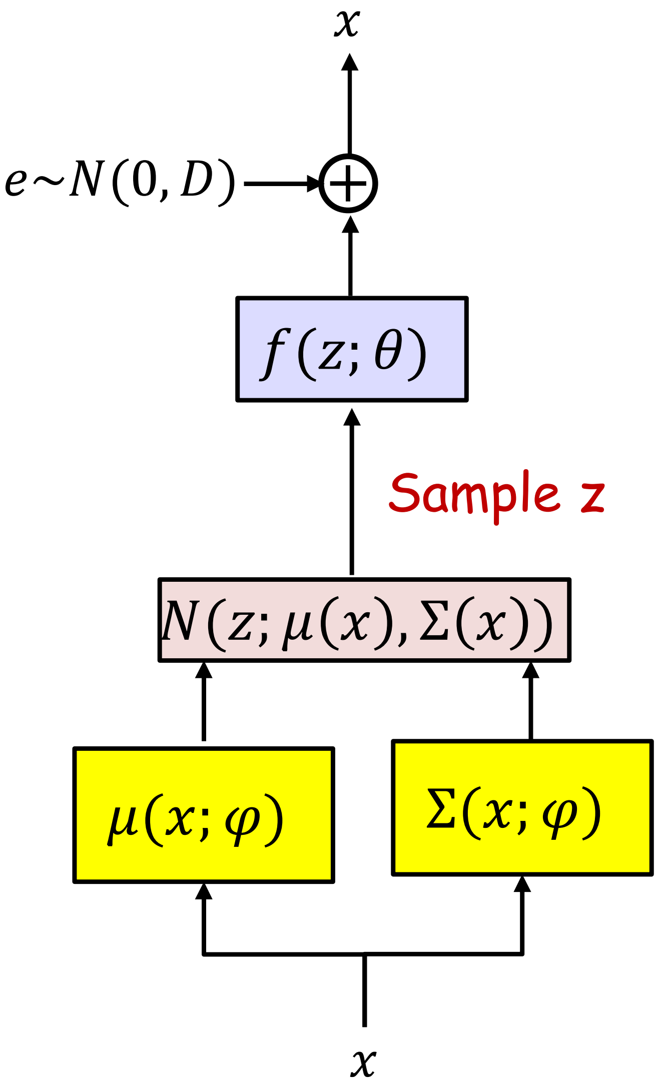


- Initialize $f(z;\theta)$
- Iterate:
  - Estimate $\mu(x;\varphi)$ and $\Sigma(x;\varphi)$ to give you the best $Q(x,z)$
  - “Complete” the data using $Q(z,x)$
  - Reestimate $f(z;\theta)$

The complete pipeline

- Initialize θ and φ

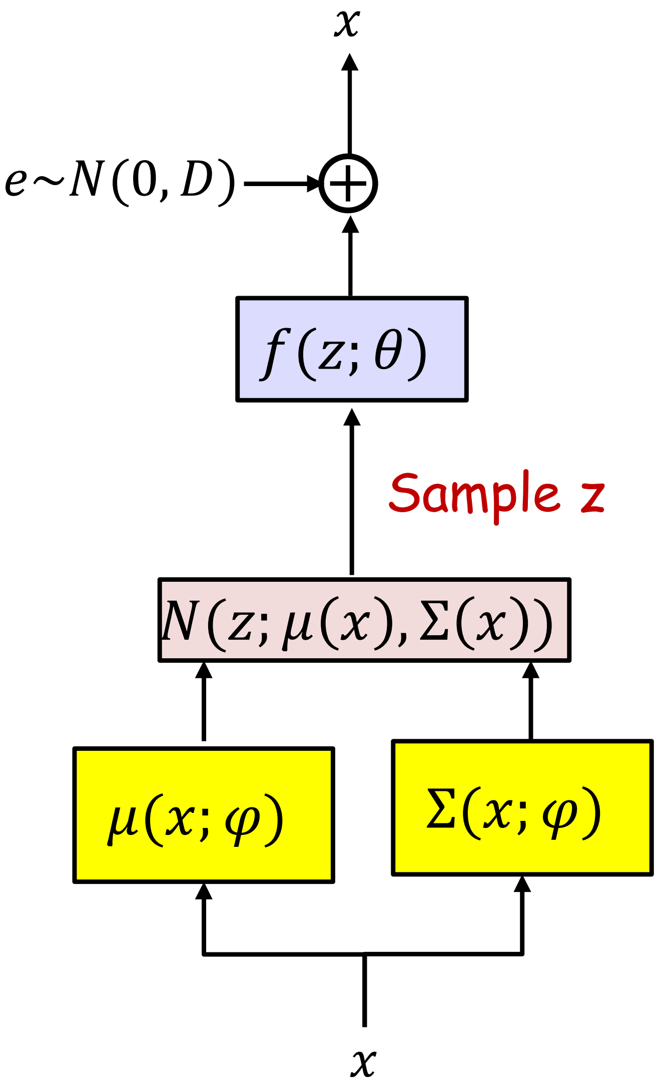
- Iterate:
  - Sample z from $N(z; \mu(x;\varphi), \Sigma(x;\varphi))$ for each training instance
    - "Completing" the data
  - Reestimate θ from the entire "complete" data
  - Estimate φ using the entire "complete" data



The complete pipeline

- Initialize θ and φ

- Iterate:
  - Sample z from $N(z; \mu(x;\varphi), \Sigma(x;\varphi))$ for each training instance
    - "Completing" the data
  - Reestimate θ from the entire "complete" data
  - Estimate φ using the entire "complete" data



Sampling z

- *Sample z from* $N(z;\mu(x;\varphi),\Sigma(x;\varphi))$ *for each training instance*
  - *"Completing" the data*

- We use a standard "reparametrization" step to sample z
  - Sample z from a standard Gaussian, and scale and shift it such that it appears as a sample from $N(z;\mu(x;\varphi),\Sigma(x;\varphi))$

- For each training instance x
  - Compute $\mu(x;\varphi)$ and $\Sigma(x;\varphi)$
  - Draw one or more samples from the Gaussian $N(z;\mu(x;\varphi),\Sigma(x;\varphi))$
    - Draw K-dimensional vector ε from $N(0,I)$
    - Compute $z=\mu(x;\varphi)+\Sigma(x;\varphi)^{0.5}\varepsilon$

Sampling z

- *Sample z from $N(z;\mu(x;\varphi),\Sigma(x;\varphi))$ for each training instance*
  - *"Completing" the data*

- We use a standard "reparametrization" step to sample z
  - Sample z from a standard Gaussian, and scale and shift it such that it appears as a sample from $N(z;\mu(x;\varphi),\Sigma(x;\varphi))$

- For each training instance x
  - Compute $\mu(x;\varphi)$ and $\Sigma(x;\varphi)$
  - Draw one or more samples from the Gaussian $N(z;\mu(x;\varphi),\Sigma(x;\varphi))$
    - Draw K-dimensional vector ε from $N(0,I)$
    - Compute $z=\mu(x;\varphi)+\Sigma(x;\varphi)^{0.5}\varepsilon$

Remember this one $\nabla_\varphi z=\nabla_\varphi\mu(x;\varphi)+diag(\varepsilon)\nabla_\varphi\Sigma(x;\varphi)^{0.5}$

This will be specific to x and to the *specific sample of* z for that x (via ε)

The complete pipeline

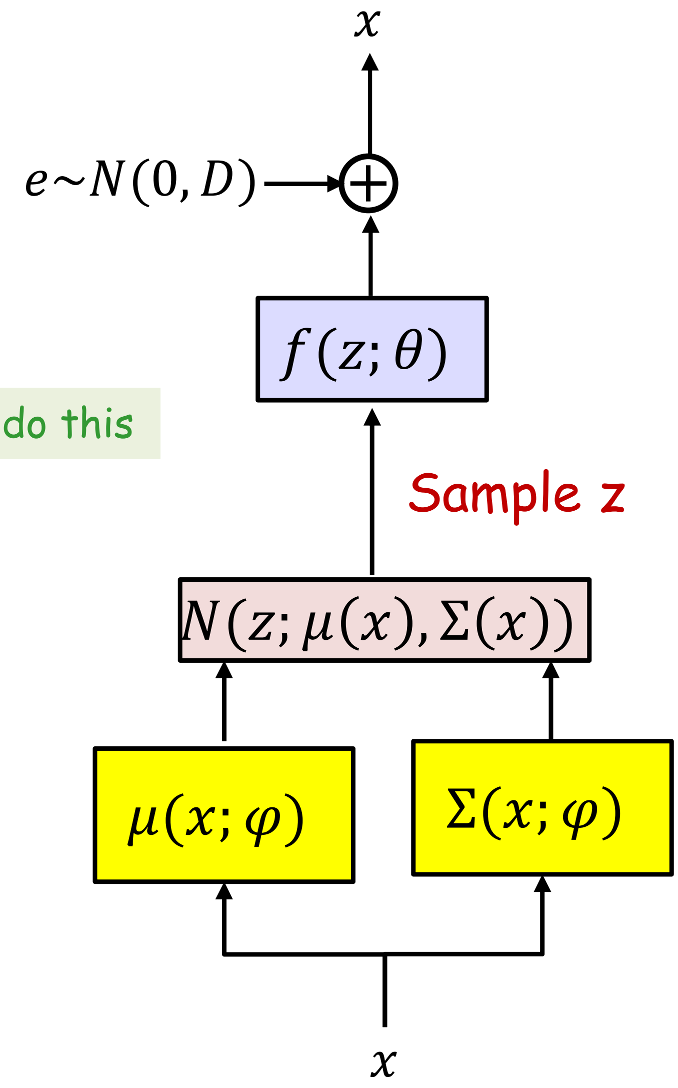
- Initialize θ and φ

- Iterate:
  - Sample z from $N(z;\mu(x;\varphi),\Sigma(x;\varphi))$ for each training instance
    - "Completing" the data

    We know how to do this

  - Reestimate θ from the entire "complete" data

  - Estimate φ using the entire "complete" data



NLGM with complete information

$$\theta^*, D^* = \underset{\theta, D}{\operatorname{argmax}} \sum_{(x,z)} -\frac{1}{2}\log|D| - 0.5(x - f(z;\theta))^T D^{-1}(x - f(z;\theta))$$

- We can learn the parameters using backpropagation, which minimizes the following loss

$$L(\theta, D) = \sum_{(x,z)} \log|D| + (x - f(z;\theta))^T D^{-1}(x - f(z;\theta))$$

$$\theta^*, D^* = \underset{\theta, D}{\operatorname{argmin}}\, L(\theta, D)$$

- It is common to assume that all the (diagonal) entries of D are identical, with value σ^2

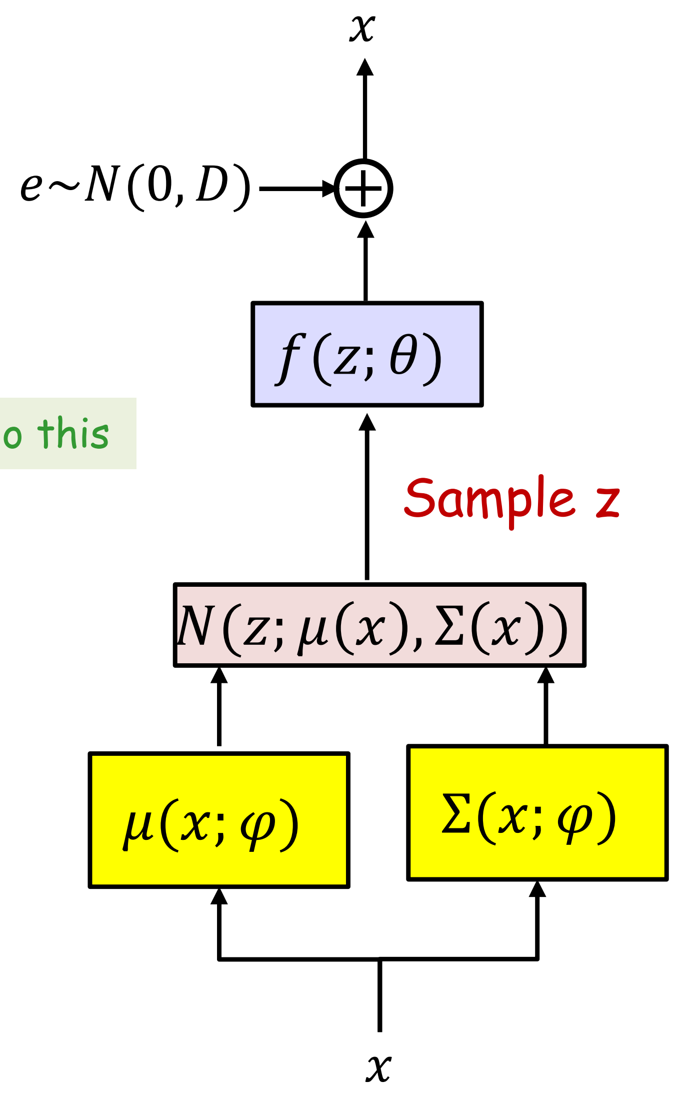
$$L(\theta, \sigma^2) = d \log \sigma^2 + \sum_{(x,z)} \frac{1}{\sigma^2}\|x - f(z;\theta)\|^2$$

- The derivative of this w.r.t θ and σ^2 is trivially computed for backprop

The complete pipeline

- Initialize θ and φ

- Iterate:
  - Sample z from $N(z; \mu(x;\varphi), \Sigma(x;\varphi))$ for each training instance We know how to do this
    - "Completing" the data
  - Reestimate θ from the entire "complete" data We know how to do this
  - Estimate φ using the entire "complete" data



Approximating $P(z|x)$ by $Q(z,x)$

- Recall $Q(z,x) = N(z;\mu(x;\varphi),\Sigma(x;\varphi))$must approximate $P(z|x)$ as closely as possible

- Estimate φ to minimize the error between $Q(z,x)$ and $P(z|x)$
  - Define a divergence between $Q(z,x)$ and $P(z|x)$ and minimize it w.r.t. φ
  - Following the literature, we will use the KL divergence
    - Then I will give you a simpler explanation

Approximating $P(z|x)$ by $Q(z,x)$

$$KL(Q(z,x)P(z|x)) = E_{z\sim Q}\log\frac{Q(z,x)}{P(z|x)}$$

$$= E_{z\sim Q}\log Q(z,x) - E_{z\sim Q}\log P(z|x)$$

$$= E_{z\sim Q}\log Q(z,x) - E_{z\sim Q}\log\frac{P(z)P(x|z)}{P(x)}$$

$$= E_{z\sim Q}\log Q(z,x) - E_{z\sim Q}\log P(z) - E_{z\sim Q}\log P(x|z) + E_{z\sim Q}\log P(x)$$

$$= KL\big(Q(z,x),P(z)\big) - E_{z\sim Q}\log P(x|z) + E_{z\sim Q}\log P(x)$$

- $Q(z,x)$ is a function of φ. Minimizing the loss w.r.t. φ we get

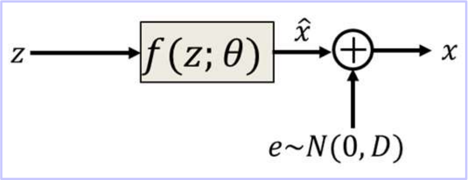
$$\varphi^* = \underset{\varphi}{\operatorname{argmin}}\, KL(Q(z,x)P(z|x))$$

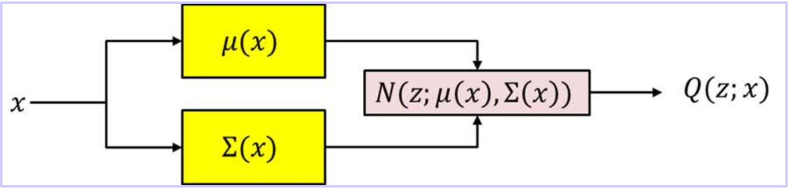
$$= \underset{\varphi}{\operatorname{argmin}}\, KL\big(Q(z,x),P(z)\big) - E_{z\sim Q}\log P(x|z)$$

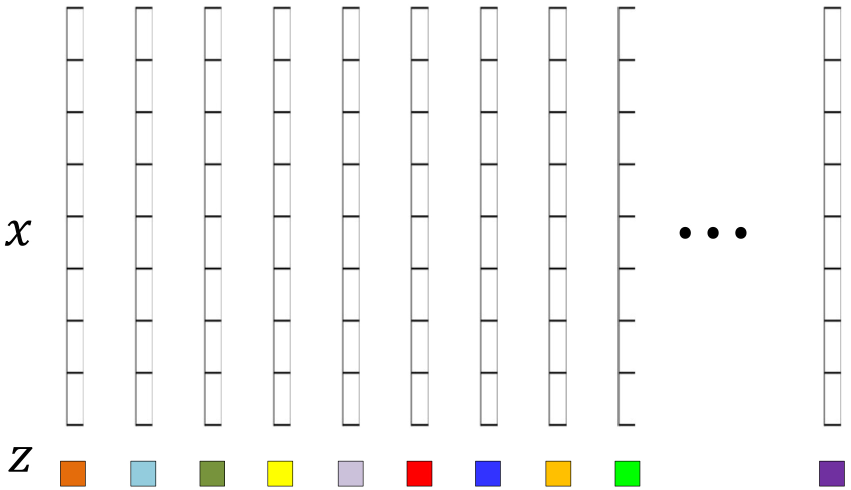
Find φ to minimize the (empirical estimate of the) KL divergence between Q(z,x) and P(z) while simultaneously maximizing the (empirical estimate of) the expectation of log P(x|z)

Let’s try that again…

NLGM with complete data

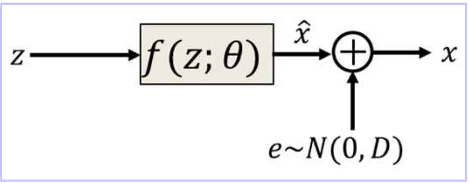


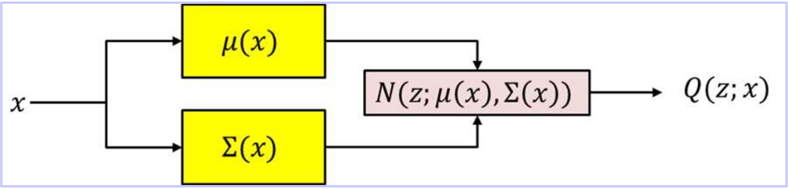


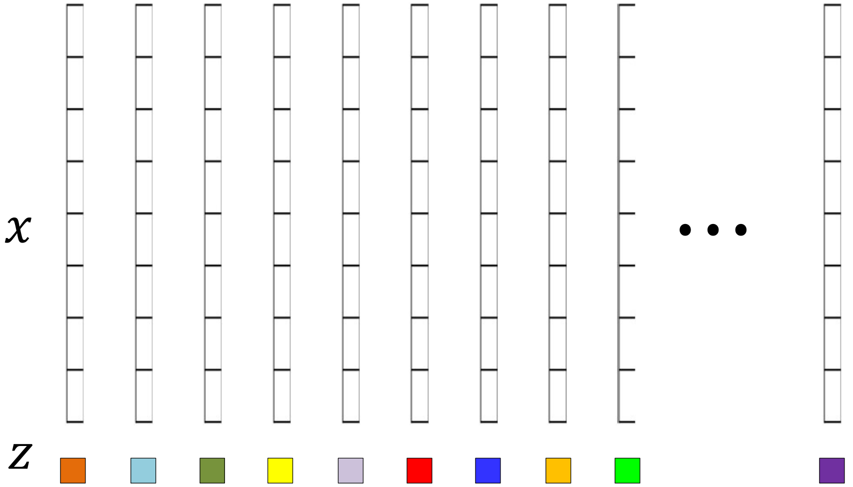


- Assume we have completed the data using $Q(z,x)$
  - We have a collection of (x,z) pairs
    - More precisely denoted as $(x, z_{x,j})$ since the value $z_{x,j}$ used to complete the observation is specific to x (subscript x)
      - Also, a single x may be completed in multiple ways (subscript j)
    - We will use (x,z) as our shorthand notation, though
  - We can work with complete data!

NLGM with complete data







- Representing the completed data as $[X,Z] = \{(x,z)\}$, the actual posterior probability for Z given X as computed by the model is

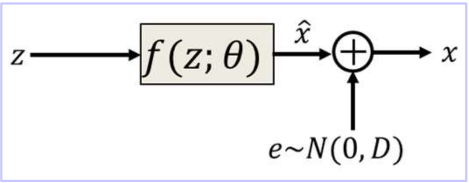
$$P(Z|X;\theta) = \prod_{(x,z)\in[X,Z]} P(z|x;\theta)\,, \qquad \log P(Z|X;\theta) = \sum_{(x,z)\in[X,Z]} \log P(z|x;\theta)$$

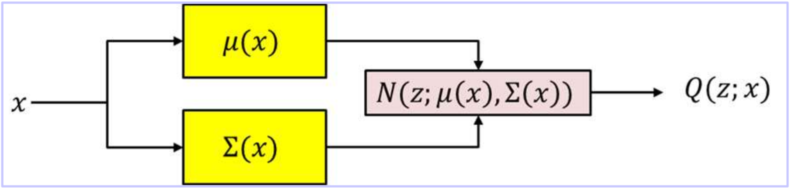
  - Because the observations are independent

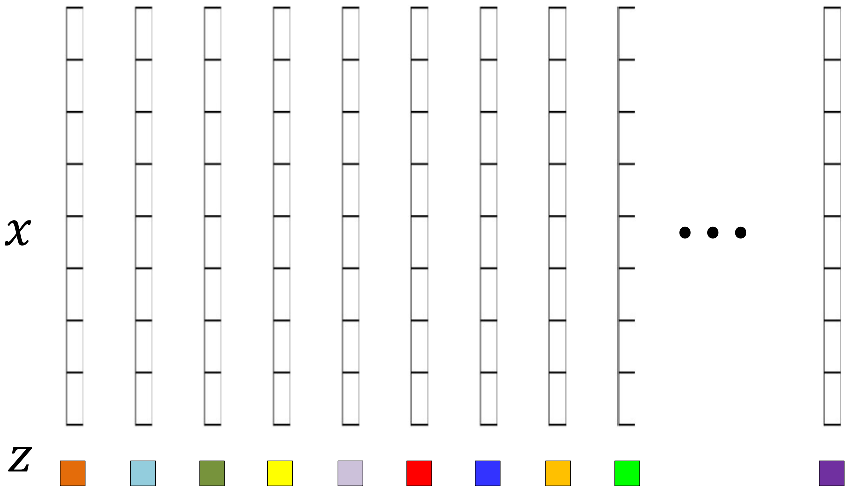
- The approximation using $Q(Z,X)$ is

$$Q(Z,X;\varphi) = \prod_{(x,z)\in[X,Z]} Q(z,x;\varphi)\,, \qquad \log Q(Z,X;\varphi) = \sum_{(x,z)\in[X,Z]} \log Q(z,x;\varphi)$$

NLGM with complete data







- Representing the completed data as $[X,Z] = \{(x,z)\}$, the actual posterior probability for Z given X as computed by the model is

We will estimate φ to minimize the discrepancy between these two probabilities

$$\log P(Z|X;\theta) = \sum_{(x,z)\in[X,Z]} \log P(z|x;\theta)$$

$$\Updownarrow$$

$$\log Q(Z,X;\varphi) = \sum_{(x,z)\in[X,Z]} \log Q(z,x;\varphi)$$

Estimating $\boldsymbol{\varphi}$

- We will minimize the following error

$$\log Q(Z,X;\varphi) - \log P(Z|X;\theta)$$

$$= \sum_{(x,z)\in[X,Z]} \log Q(z,x;\varphi) - \log P(z|x;\theta)$$

Estimating φ

- We will minimize the following error

$$\log Q(Z, X; \varphi) - \log P(Z|X; \theta)$$

$$= \sum_{(x,z)\in[X,Z]} \log Q(z, x; \varphi) - \log P(z|x; \theta)$$

- By Bayes rule $P(z|x; \theta) = P(z)P(x|z)/P(x)$
- The error becomes

$$\sum_{(x,z)\in[X,Z]} \log Q(z, x; \varphi) - \log P(z) - \log P(x|z; \theta) + \log P(x; \theta)$$

Estimating $\boldsymbol{\varphi}$

- We will minimize the following error

$$\log Q(Z, X; \varphi) - \log P(Z|X; \theta)$$

$$= \sum_{(x,z)\in[X,Z]} \log Q(z, x; \varphi) - \log P(z|x; \theta)$$

- By Bayes rule $P(z|x; \theta) = P(z)P(x|z)/P(x)$
- The error becomes

$$\sum_{(x,z)\in[X,Z]} \log Q(z, x; \varphi) - \log P(z) - \log P(x|z; \theta) + \log P(x; \theta)$$

- φ influences $Q(z, x; \varphi)$ directly and z, because it is sampled from $Q(z, x; \varphi)$. $P(x; \theta)$ is not related to either φ or z and can be ignored.

Estimating φ

- We will minimize the following error

$$\log Q(Z,X;\varphi) - \log P(Z|X;\theta)$$

$$= \sum_{(x,z)\in[X,Z]} \log Q(z,x;\varphi) - \log P(z|x;\theta)$$

- By Bayes rule $P(z|x;\theta) = P(z)P(x|z)/P(x)$
- The error becomes

$$\sum_{(x,z)\in[X,Z]} \log Q(z,x;\varphi) - \log P(z) - \log P(x|z;\theta) + \log P(x;\theta)$$

- φ influences $Q(z,x;\varphi)$ directly and z, because it is sampled from $Q(z,x;\varphi)$.
- $P(x;\theta)$ is not related to either φ or z and can be ignored.
- This gives us the loss function

$$L_Q(\varphi) = \sum_{(x,z)\in[X,Z]} \log Q(z,x;\varphi) - \log P(z) - \log P(x|z;\theta)$$

- This must be minimized w.r.t. φ

Estimating φ

- We will minimize the following error

$$\log Q(Z, X; \varphi) - \log P(Z|X; \theta)$$

$$= \sum_{(x,z)\in[X,Z]} \log Q(z, x; \varphi) - \log P(z|x; \theta)$$

- By Bayes rule $P(z|x; \theta) = P(z)P(x|z)/P(x)$
- The error becomes

$$\sum_{(x,z)\in[X,Z]} \log Q(z, x; \varphi) - \log P(z) - \log P(x|z; \theta) + \log P(x; \theta)$$

- φ influences $Q(z, x; \varphi)$ directly and z, because it is sampled from $Q(z, x; \varphi)$.
- $P(x; \theta)$ is not related to either φ or z and can be ignored.
- This giv

How do we complete the data for learning φ?

$$L_Q(\varphi) = \sum_{(x,z)\in[X,Z]} \log Q(z, x; \varphi) - \log P(z) - \log P(x|z; \theta)$$

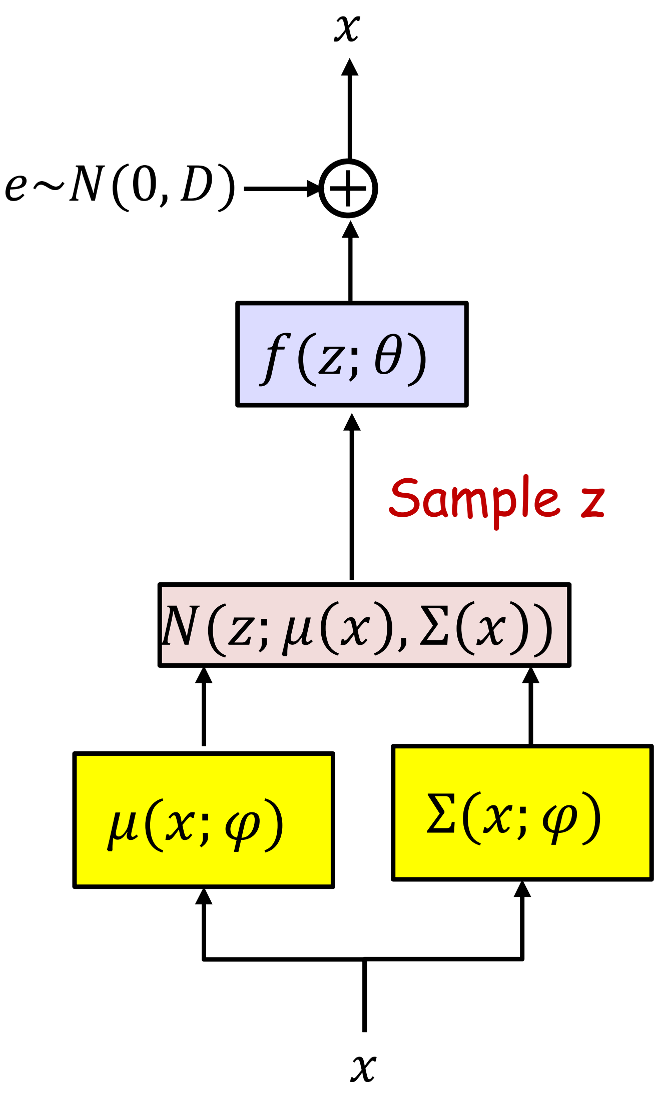
- This must be minimized w.r.t. φ

The complete training pipeline

- For θ : Must sample z from $Q(Z;\varphi)$
  - These are used to learn θ

- For φ : But to learn φ *given* θ, find how the sampled z must be perturbed to make the corresponding x more plausible
  - While keeping the PDF of the adjusted zs as standard Gaussian

- Or more precisely, how to adjust ϕ to make the adjusted z more likely



Estimating φ

- The loss function

$$L_Q(\varphi) = \sum_{(x,z)\in[X,Z]} \log Q(z,x;\varphi) - \log P(z) - \log P(x|z;\theta)$$

- Multiple choices for data completion

- Simple option: Simply use samples drawn from $Q(z,x;\varphi)$
  - You can skip the next couple of slides if you do

$$\nabla_\varphi L_Q(\varphi) = \sum_{(x,z)\in[X,Z]} \nabla_\varphi \log Q(z,x;\varphi) + \nabla_z \log Q(z,x;\varphi)\, \nabla_\varphi z \\ - \nabla_z \log P(z) \nabla_\varphi z - \nabla_z \log P(x|z;\theta)\, \nabla_\varphi z$$

Estimating $\boldsymbol{\varphi}$

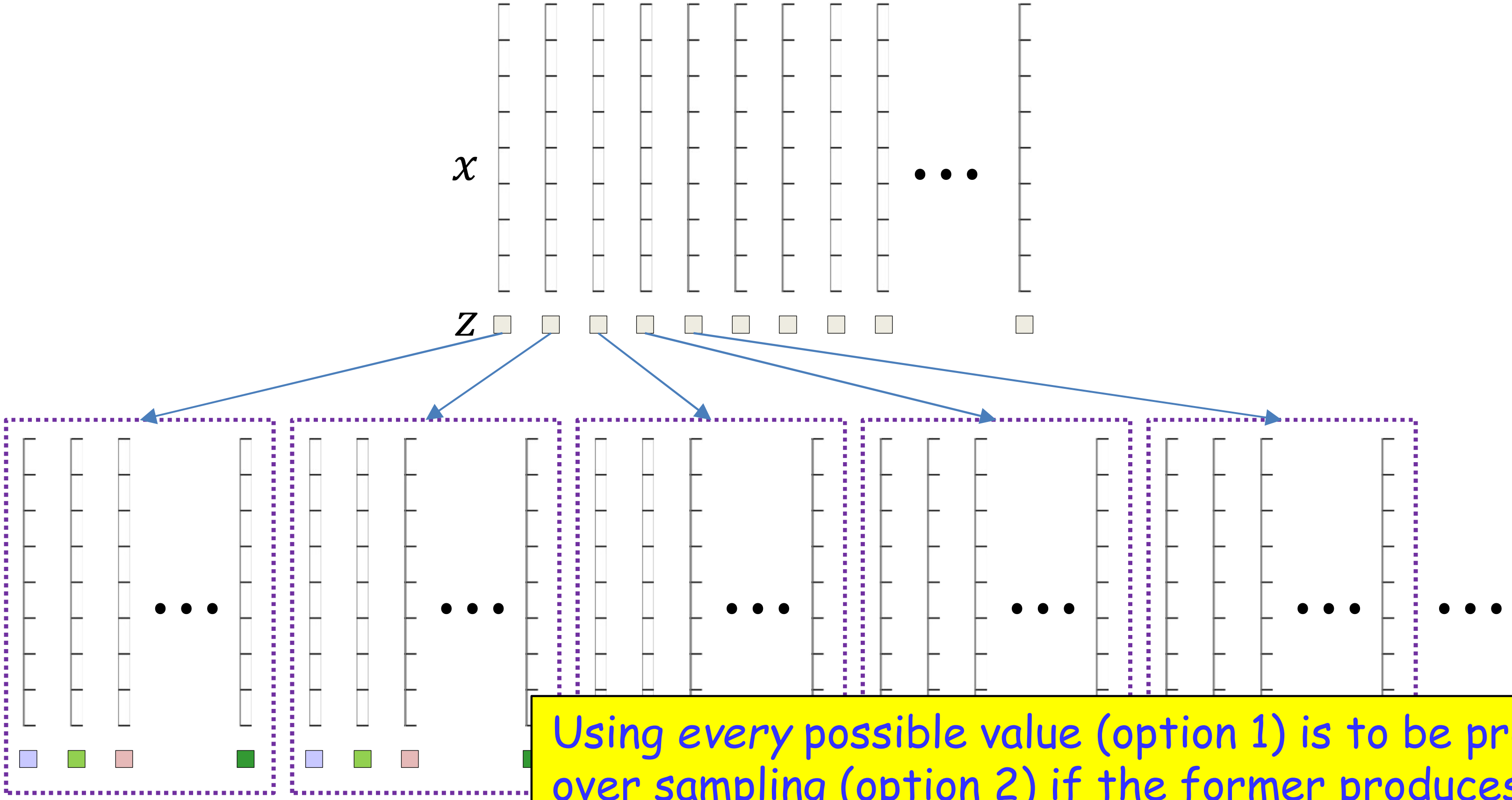
- The loss function

$$L_Q(\varphi) = \sum_{(x,z)\in[X,Z]} \log Q(z,x;\varphi) - \log P(z) - \log P(x|z;\theta)$$

- Multiple choices for data completion
- Simple option: Simply use samples drawn from $Q(z,x;\varphi)$
  - You can skip the next couple of slides if you do

- Orrrr try to be more precise….

Expectation Maximization for LGM



Using *every* possible value (option 1) is to be preferred over sampling (option 2) if the former produces tractable closed form solutions. Otherwise we must use option 2.

- *Complete* the data
  - Option 1 : Consider *every possible value for z*
  - Option 2: ***By drawing samples from*** $P(z|x)$
- Compute the solution from the completed data

Estimating φ

- The loss function

$$L_Q(\varphi) = \sum_{(x,z)\in[X,Z]} \underbrace{\log Q(z,x;\varphi) - \log P(z)}_{} - \underbrace{\log P(x|z;\theta)}_{}$$

- It turns out that the portion underlined in blue can be computed in closed form if you consider every possible value of z

- The portion underlined in red cannot

- So sum the first portion over all possible values of z from and the second one over only the drawn samples

$$L_Q(\varphi) = \sum_{(x)\in[X]} \int_{-\infty}^{\infty} Q(z,x;\varphi)(\log Q(z,x;\varphi) - \log P(z))\, dz - \sum_{(x,z)\in[X,Z]} \log P(x|z;\theta)$$

Estimating φ

- The loss function

$$L_Q(\varphi) = \sum_{(x,z)\in[X,Z]} \underbrace{\log Q(z,x;\varphi) - \log P(z)}_{} - \underbrace{\log P(x|z;\theta)}_{}$$

- It turns out that the portion underlined in blue can be computed in closed form if you consider every possible value of z

- The portion underlined in red cannot

- So sum the first portion over all possible values of z from and the second one over only the drawn samples

$$L_Q(\varphi) = \sum_{(x)\in[X]} \underbrace{\int_{-\infty}^{\infty} Q(z,x;\varphi)(\log Q(z,x;\varphi) - \log P(z))\, dz}_{\boxed{KL(Q(z,x;\varphi),P(z))}} - \sum_{(x,z)\in[X,Z]} \log P(x|z;\theta)$$

Estimating $\boldsymbol{\varphi}$

- The loss function

$$L_Q(\varphi) = \sum_{x \in X} KL(Q(z, x; \varphi), P(z)) - \sum_{(x,z) \in [X,Z]} \log P(x|z; \theta)$$

- We have:

$$Q(z, x) = N\big(z; \mu(x; \varphi), \Sigma(x; \varphi)\big), \qquad\qquad P(z) = N(0, I)$$

- The KL between the two Gaussians works out to

$$KL\big(Q(z, x; \varphi), P(z)\big) = \frac{1}{2}\Big(tr\big(\Sigma(x; \varphi)\big) + \mu(x; \varphi)^T\big(\mu(x; \varphi) - d - \log|\Sigma(x; \varphi)|\big)\Big)$$

- We have

$$\log P(x|z; \theta) = \sum_{(x,z)} -\frac{1}{2}\log|D| - 0.5(x - f(z; \theta))^T D^{-1}(x - f(z; \theta))$$

- Plugging it all in:

$$L_Q(\varphi)$$

$$= \sum_{x \in X} \frac{1}{2}\Big(tr\big(\Sigma(x; \varphi)\big) + \mu(x; \varphi)^T\big(\mu(x; \varphi) - d - \log|\Sigma(x; \varphi)|\big)\Big)$$

$$+ \sum_{(x,z) \in [X,Z]} \frac{1}{2}\log|D| + 0.5(x - f(z; \theta))^T D^{-1}(x - f(z; \theta))$$

Estimating $\boldsymbol{\varphi}$

- So we finally have the loss function (ignoring unnecessary terms and factors)

$$L_Q(\varphi)$$

$$= \sum_{x \in X} \left(tr\big(\Sigma(x;\varphi)\big) + \mu(x;\varphi)^T \big(\mu(x;\varphi) - d - \log|\Sigma(x;\varphi)|\big) \right)$$

$$+ \sum_{(x,z) \in [X,Z]} (x - f(z;\theta))^T D^{-1} (x - f(z;\theta))$$

- Assuming that $D$is diagonal with identical values σ^2 for the diagonal elements gives us the simplification

$$L_Q(\varphi) = \sum_{x \in X} \left(tr\big(\Sigma(x;\varphi)\big) + \mu(x;\varphi)^T \big(\mu(x;\varphi) - d - \log|\Sigma(x;\varphi)|\big) \right) + \frac{1}{\sigma^2} \sum_{(x,z) \in [X,Z]} \|(x - f(z;\theta))\|^2$$

- To estimate φ we will compute

$$\varphi^* = \underset{\varphi}{\operatorname{argmin}}\, L_Q(\varphi)$$

- To perform the minimization we will use gradient descent

Estimating $\boldsymbol{\varphi}$

$$\varphi^* = \underset{\varphi}{\operatorname{argmin}} L_Q(\varphi)$$

- To perform the minimization we will use gradient descent

$$\nabla_\varphi L_Q(\varphi)$$

$$= \sum_{x \in X} \nabla_\varphi \left(tr\big(\Sigma(x;\varphi)\big) + \mu(x;\varphi)^T \big(\mu(x;\varphi) - d - \log|\Sigma(x;\varphi)|\big) \right)$$

$$+ \frac{1}{\sigma^2} \sum_{(x,z) \in [X,Z]} \nabla_z \|(x - f(z;\theta))\|^2 \nabla_\varphi z$$

Recap from earlier

$$\nabla_\varphi z = \nabla_\varphi \mu(x;\varphi) + diag(\varepsilon) \nabla_\varphi \Sigma(x;\varphi)^{0.5}$$

This will be specific to x and to the *specific sample of* z for that x (via ε)

The complete training pipeline

- Initialize θ and φ
- Iterate:
  - Sample $z_{x,\varepsilon}$ from $N(z;\mu(x;\varphi),\Sigma(x;\varphi))$ for each training instance
    - “Completing” the data
  - Reestimate θ from the entire “complete” data

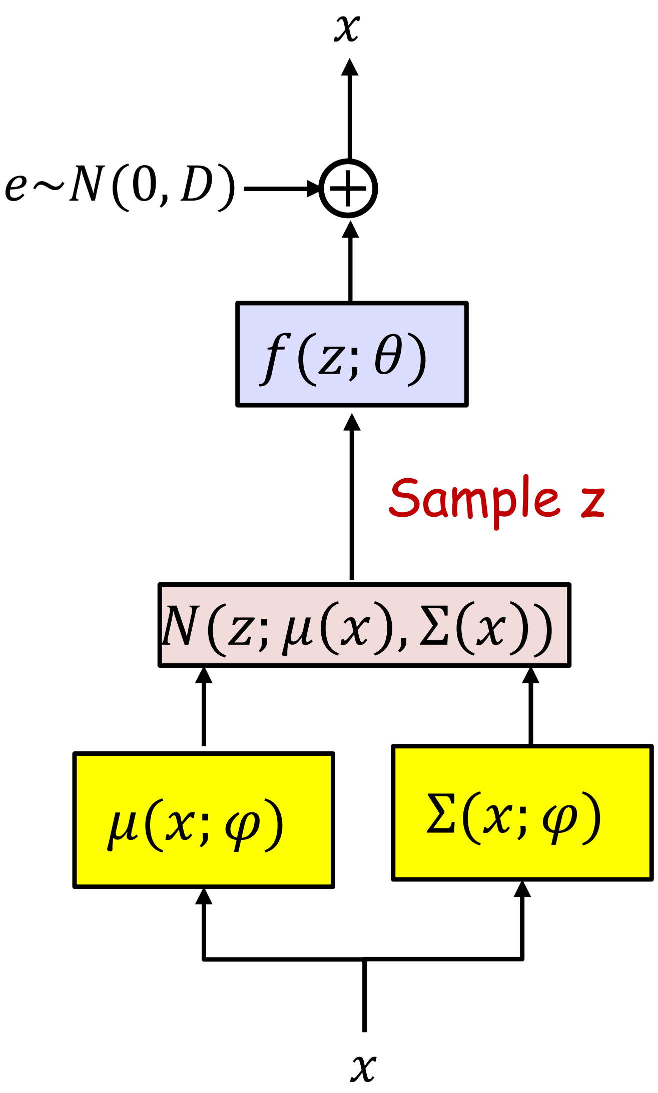
$$L(\theta,\sigma^2) = d\log\sigma^2 + \frac{1}{\sigma^2}\sum_{(x,z)}\|x - f(z;\theta)\|^2$$

  - Estimate φ using the entire “complete” data

$$L_Q(\varphi)$$

$$= \sum_{x\in X}\Big(tr\big(\Sigma(x;\varphi)\big) + \mu(x;\varphi)^T\big(\mu(x;\varphi) - d - \log|\Sigma(x;\varphi)|\big)\Big)$$

$$+ \frac{1}{\sigma^2}\sum_{(x,z)\in[X,Z]}\|(x - f(z;\theta))\|^2$$



The complete training pipeline: Single step update

- Initialize θ and φ

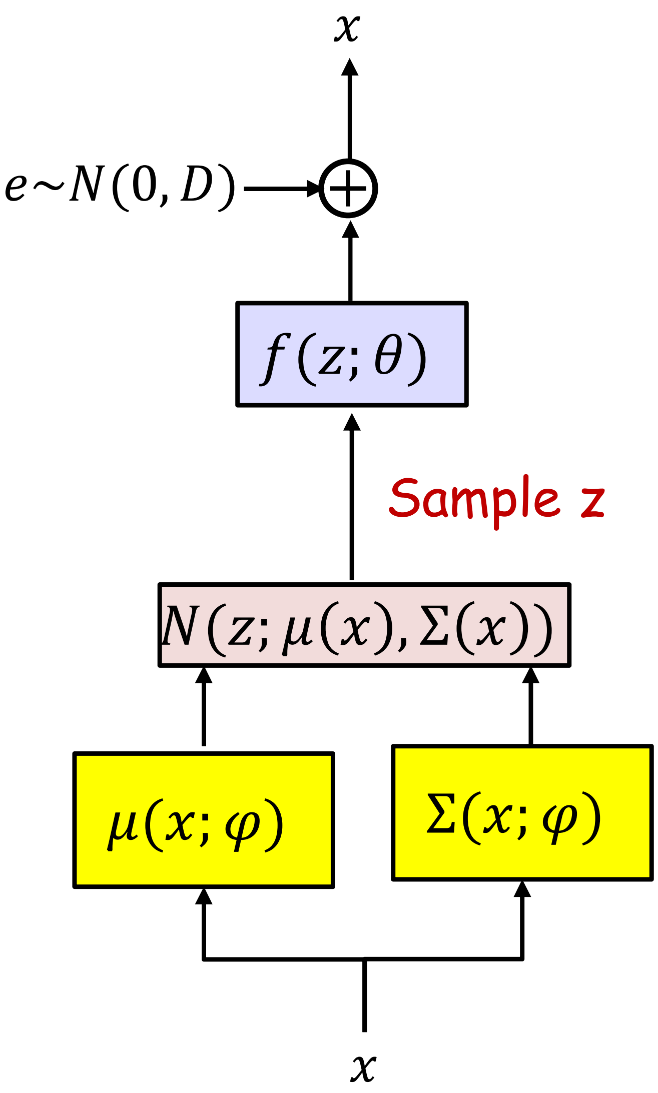
- Iterate:
  - Sample $z_{x,\varepsilon}$ from $N(z;\mu(x;\varphi),\Sigma(x;\varphi))$ for each training instance
    - "Completing" the data
  - Reestimate θ and φ from the entire "complete" data

$$L(\theta,\sigma^2,\varphi)$$

$$= \sum_{x\in X}\Big(tr\big(\Sigma(x;\varphi)\big) + \mu(x;\varphi)^T\big(\mu(x;\varphi) - d - \log|\Sigma(x;\varphi)|\big)\Big)$$

$$+ \frac{1}{\sigma^2}\sum_{(x,z)\in[X,Z]}\|(x - f(z;\theta))\|^2 + d\log\sigma^2$$

- (Merged the updates of θ and φ into a single step)
  - Gradient computation doesn't change

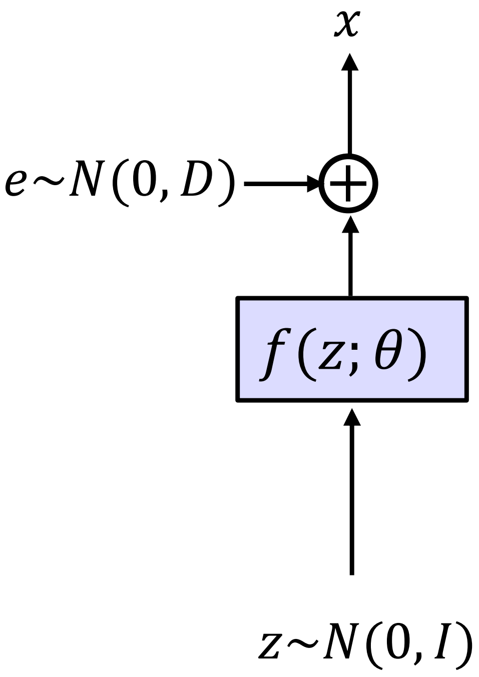


The complete training pipeline

- Once trained the approximation function $Q(z, x)$ can be discarded

- The rest of the function gives us a *generative* model for x

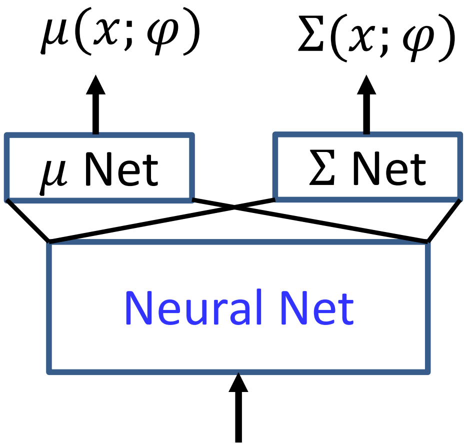
- Generating data using this part of the model should (ideally) give us data similar to the training data

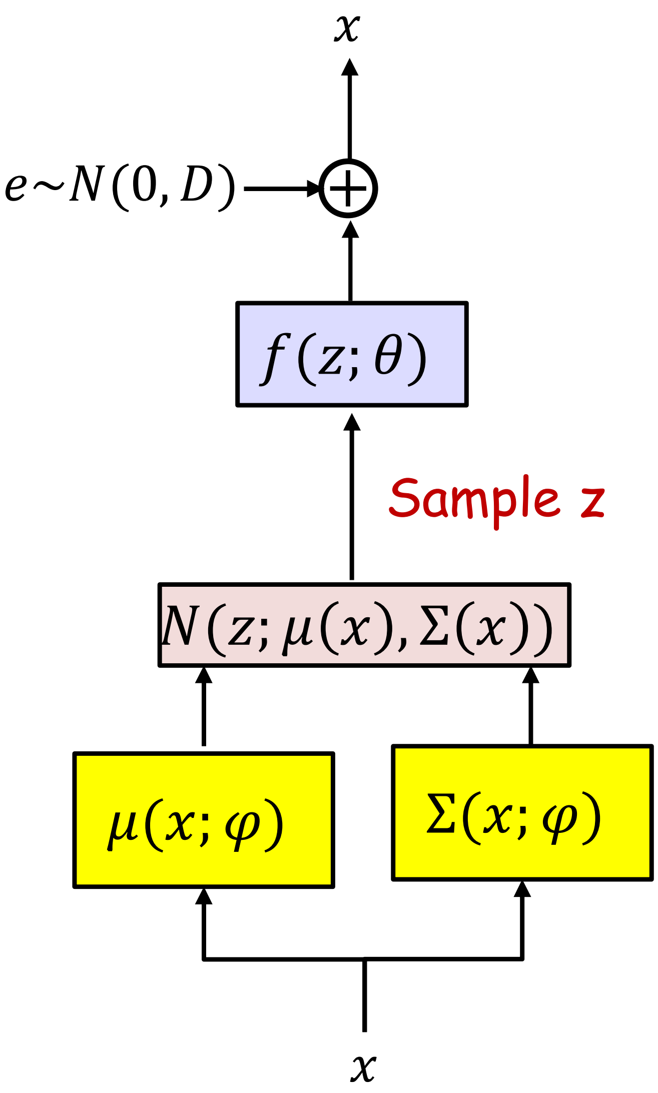


But where are the neural nets?

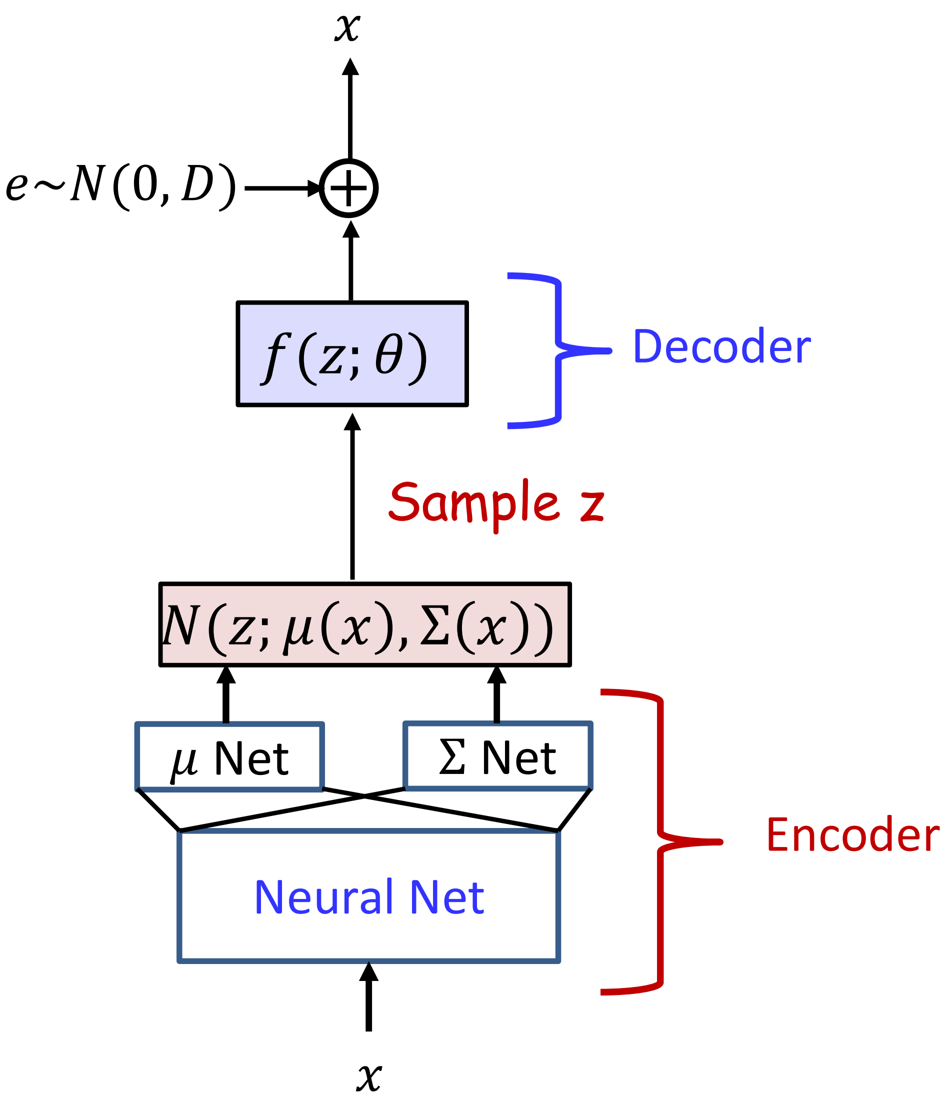
- $f(z;\theta)$ is generally modelled by a neural network

- $\mu(x;\varphi)$ and $\Sigma(x;\varphi)$ are generally modelled by a *common* network with two outputs
  - The combined parameters of the network are φ





The Variational AutoEncoder



The decoder is the actual generative model

The encoder is primarily needed for training
It can also be used to generate the (approximate) distribution of latent space representations conditioned on specific inputs input
(much like a regular autoencoder)

z is a *latent-space* representation of the data
$\mu(x)$ can also be used as a *expected latent* representation of x

Poll 4 (@1847)

Mark all that are true

- The decoder in a Variational Auto Encoder is a non-linear Gaussian model
- The NLGM in the VAE is estimated using EM
- The encoder in a VAE is a module that generates the samples of z needed to complete the data, in order to estimate the parameters of the NLGM (decoder)
- The encoder approximates P(z|x) to enable sampling of z, to complete the data

Poll 4

Mark all that are true

- **The decoder in a Variational Auto Encoder is a non-linear Gaussian model**
- **The NLGM in the VAE is estimated using EM**
- **The encoder in a VAE is a module that generates the samples of z needed to complete the data, in order to estimate the parameters of the NLGM (decoder)**
- **The encoder approximates P(z|x) to enable sampling of z, to complete the data**

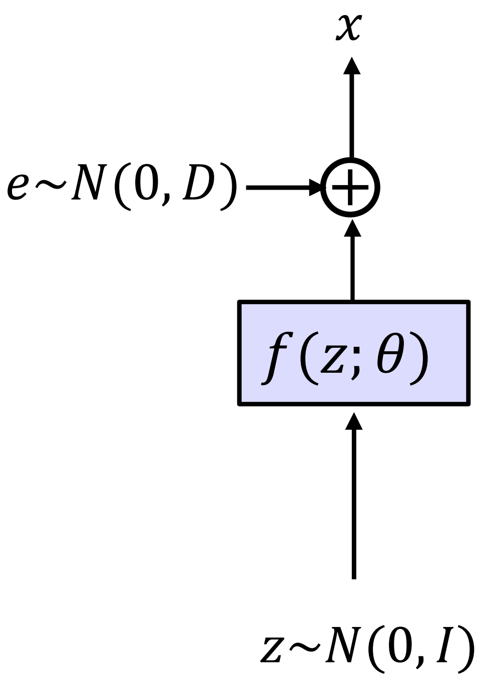
VAEs

- VAEs are, unfortunately, *strictly* generative models

- They can be used to *generate samples* of the data

- But they cannot be used to *compute the likelihood* of data
  - At least not directly
  - Because $P(x;\theta)$ is generally intractable

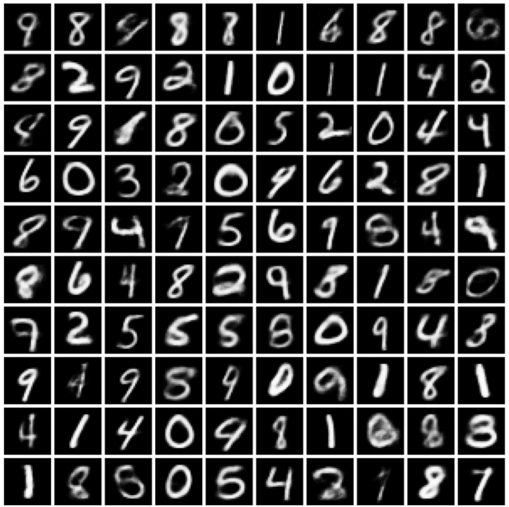
- Nevertheless, they are highly effective as generators
  - They can learn highly complex distributions



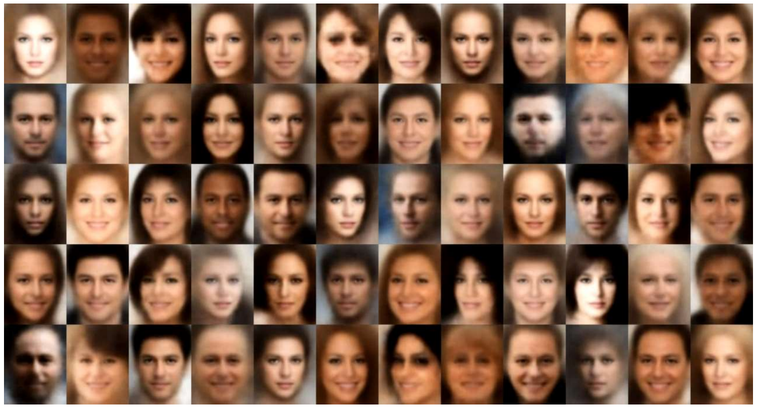
VAE examples

- Top: VAE trained on MNIST and used to generate new data

- Below: VAE trained on faces, and used to generate new data



From C. Doersch



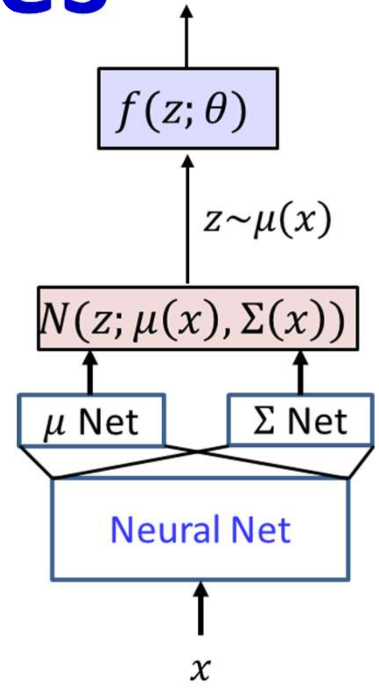
From J. Rocco

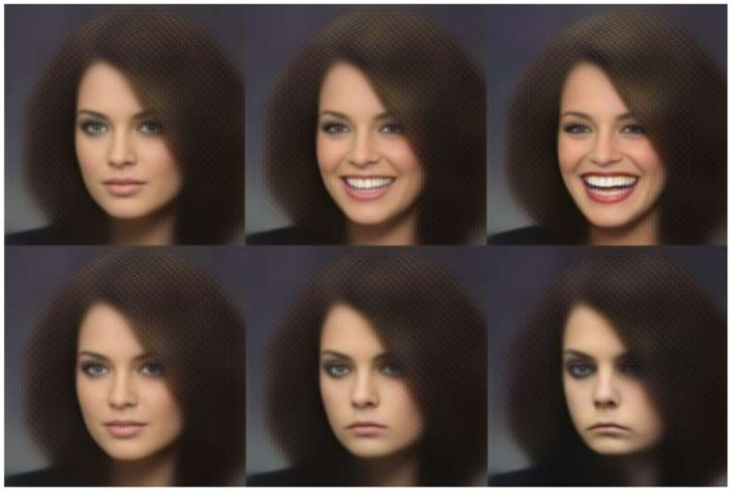
VAE and latent spaces

- The latent space z often captures underlying structure in the data x in a smooth manner

- Varying z continuously in different directions can result in plausible variations in the drawn output
  - Typically manipulations are performed by wiggling z around its expected value $\mu(x)$

- Typically, in these draws, you do not add the noise e
  - The output is the *expected* generation for a given latent value z





VAE conclusions

- Simple non-linear extensions of linear Gaussian models

- Excellent generative models for the distribution of data $P(x)$
  - Various extensions such as Conditional VAEs, which model *conditional* distributions, such as $P(x|y)$
    - Straight-forward extension where the conditioning variable y is an additional input to the encoder and decoder

- Have also been successfully embedded into dynamical system models
  - $P(z)$ now becomes a mixture, or a Markov model instead of $N(0, I)$

- In all cases, the arithmetic for learning is similar to that presented here
- Read the literature on the topic, it is vast