CS/DS 552: Class 5

Jacob Whitehill

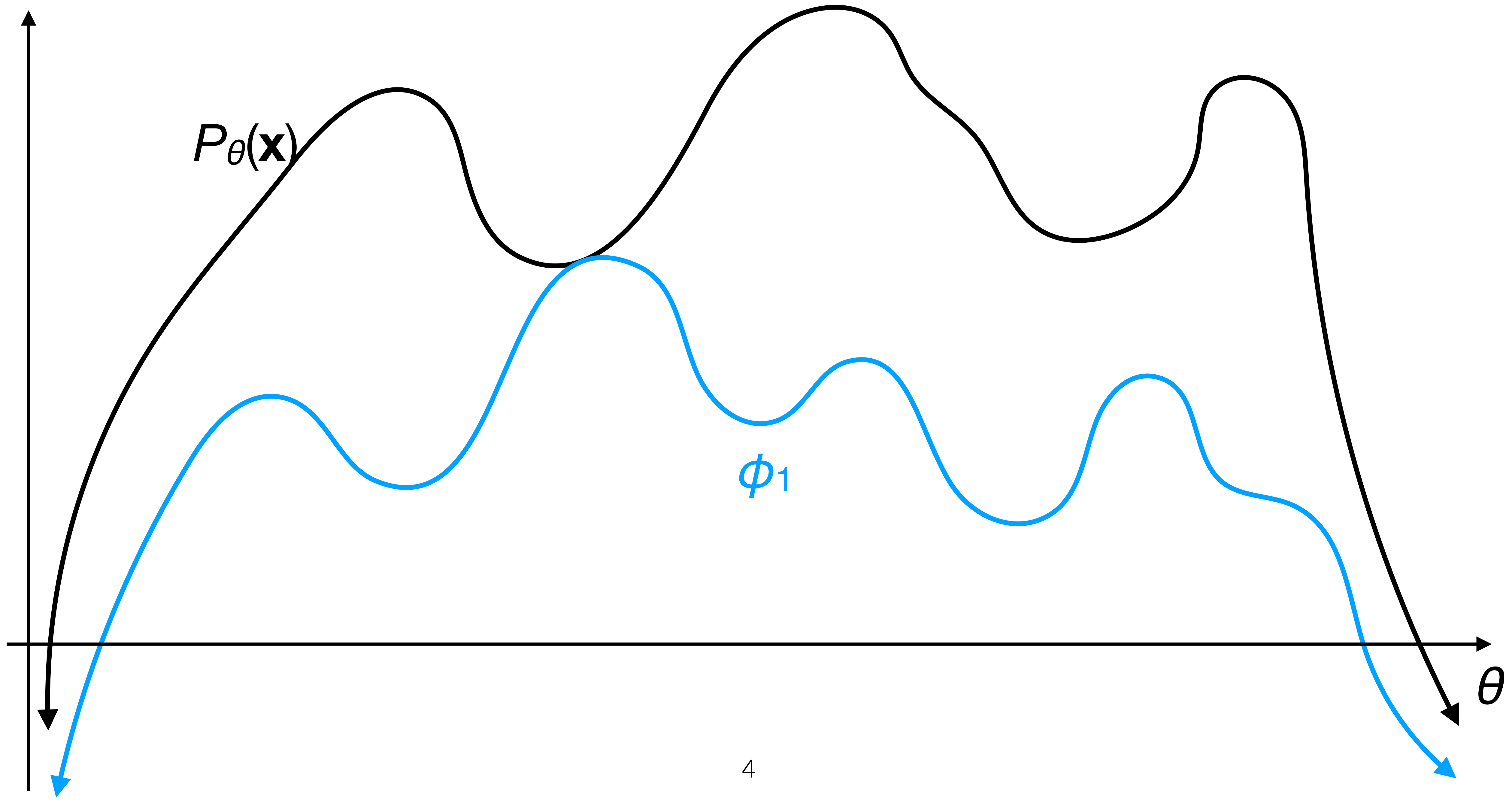
Variational auto-encoders (VAEs)

Group exercise

- Please do **not** use ChatGPT (or another AI tool) on this exercise.
- Download `vae_exercise.py` from Canvas->Files.
- Answer the following questions:
  1. Where/what are all the bugs in the code?
  2. Which lines (which might not be contiguous!) of code implement the expectation $\mathbb{E}_{Q(\mathbf{z} \mid \mathbf{x})}[P(\mathbf{x} \mid \mathbf{z})]$, and how many samples is the expectation calculated over?
  3. Which lines (which might not be contiguous!) of code implement the KL divergence? Where are the inputs to the KL term calculated, and how do they relate to the formula in the slides?
  4. What is the dimension *d* of the code **z**?
  5. How many NN linear layers are traversed in the encoder to process each **x** to produce the mean of $P(\mathbf{z} \mid \mathbf{x})$?
  6. In terms of the VAE definition from the slides, what probability distribution does `VAE.reparameterize()` sample from?

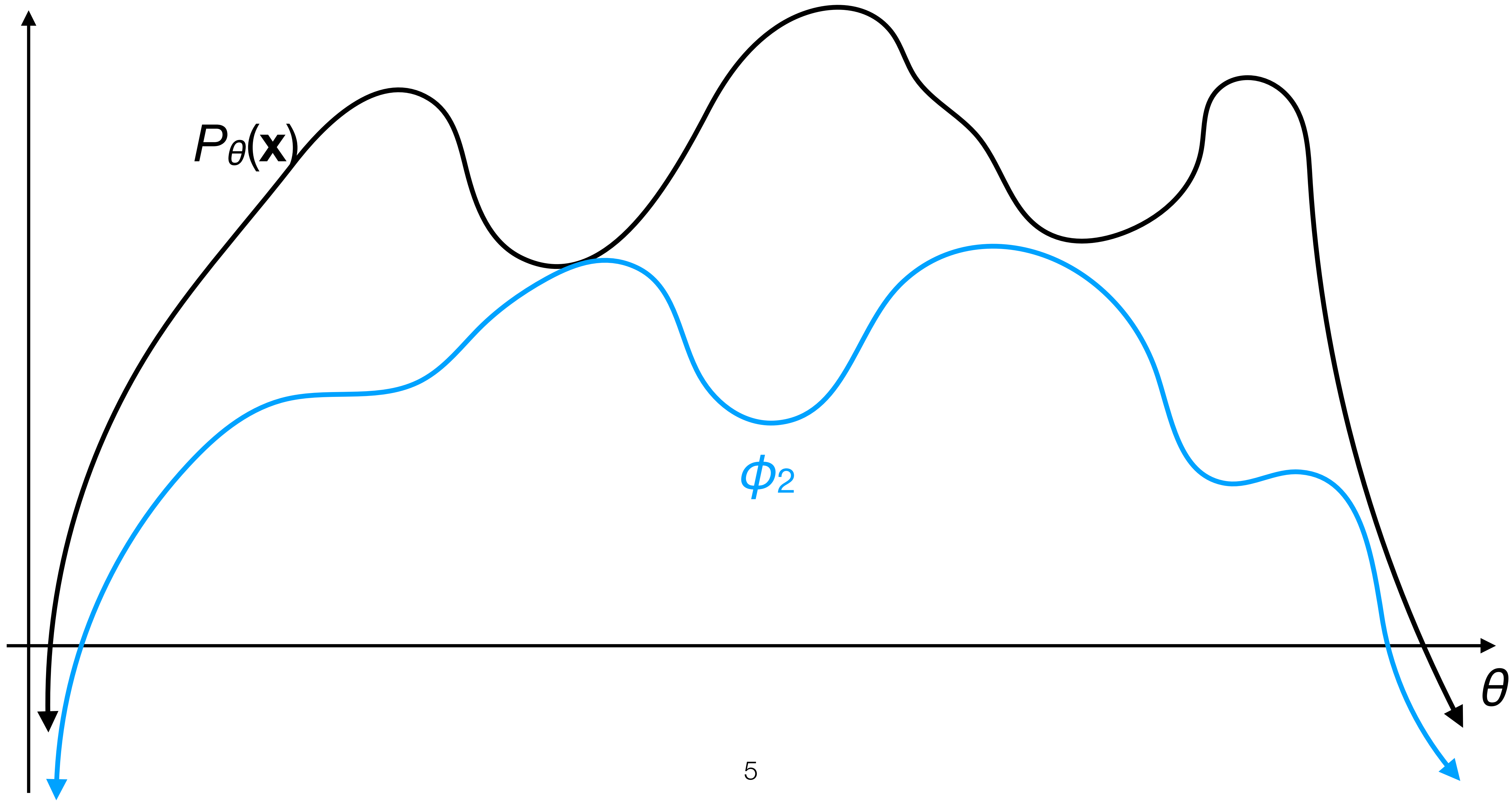
Maximizing the lower bound

- We can approximately maximize $P_\theta(\mathbf{x})$ w.r.t. θ by maximizing the lower bound w.r.t. θ and ϕ.



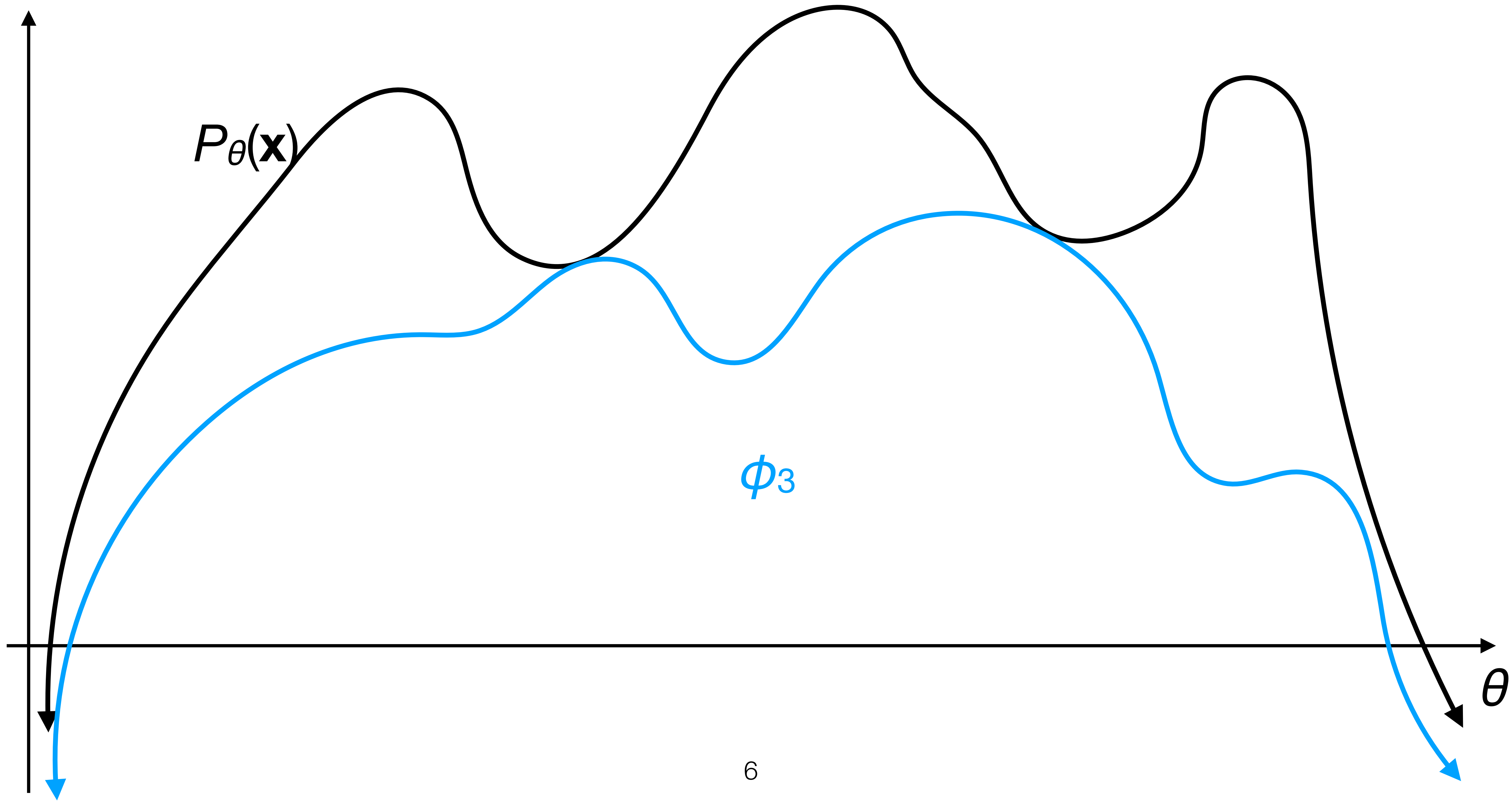
Maximizing the lower bound

- By iteratively updating Q w.r.t. ϕ, we can increase the upper bound (though it will never exceed $P_\theta(\boldsymbol{x})$).



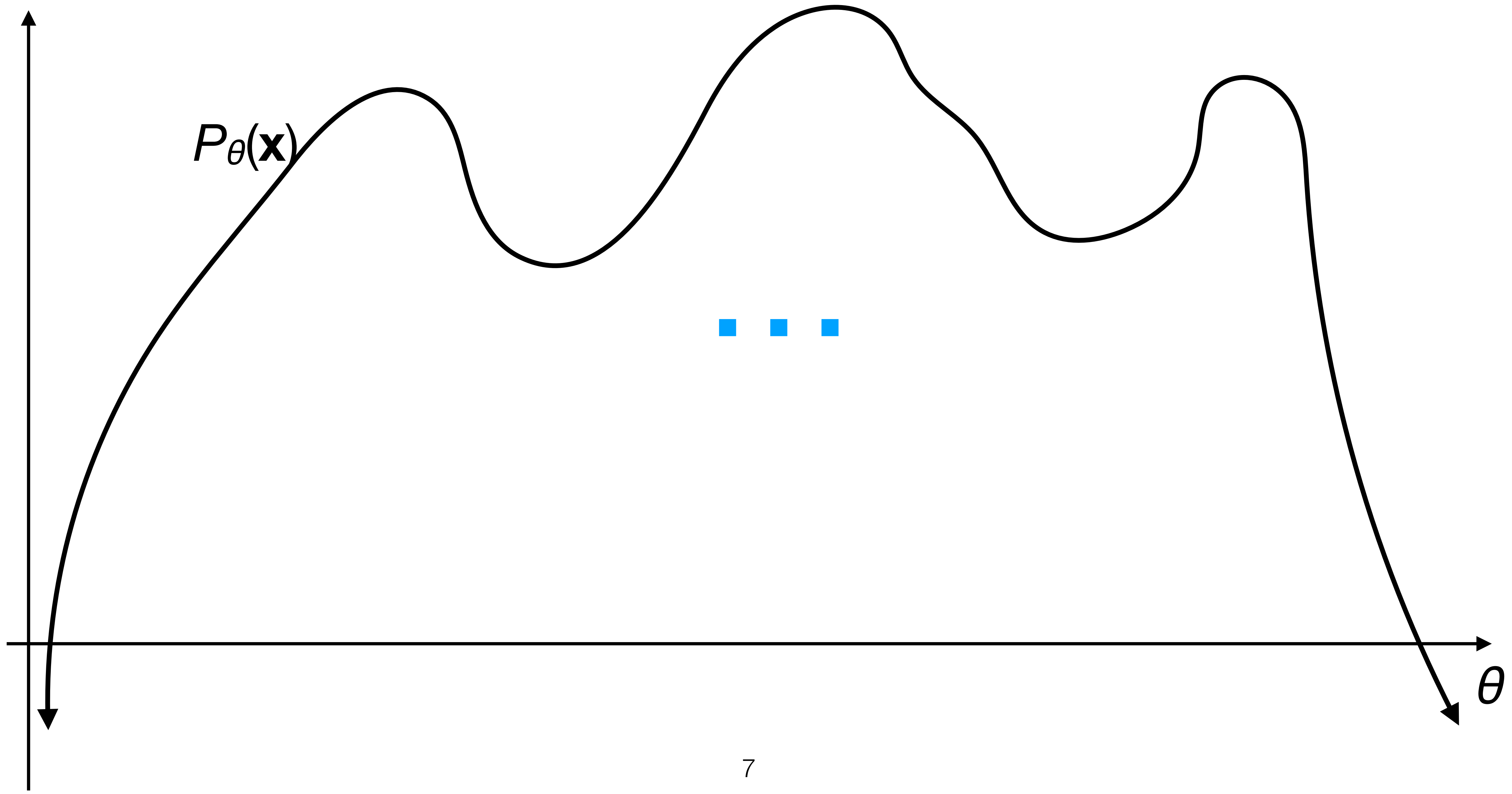
Maximizing the lower bound

- By iteratively updating Q w.r.t. ϕ, we can increase the upper bound (though it will never exceed $P_\theta(\mathbf{x})$).



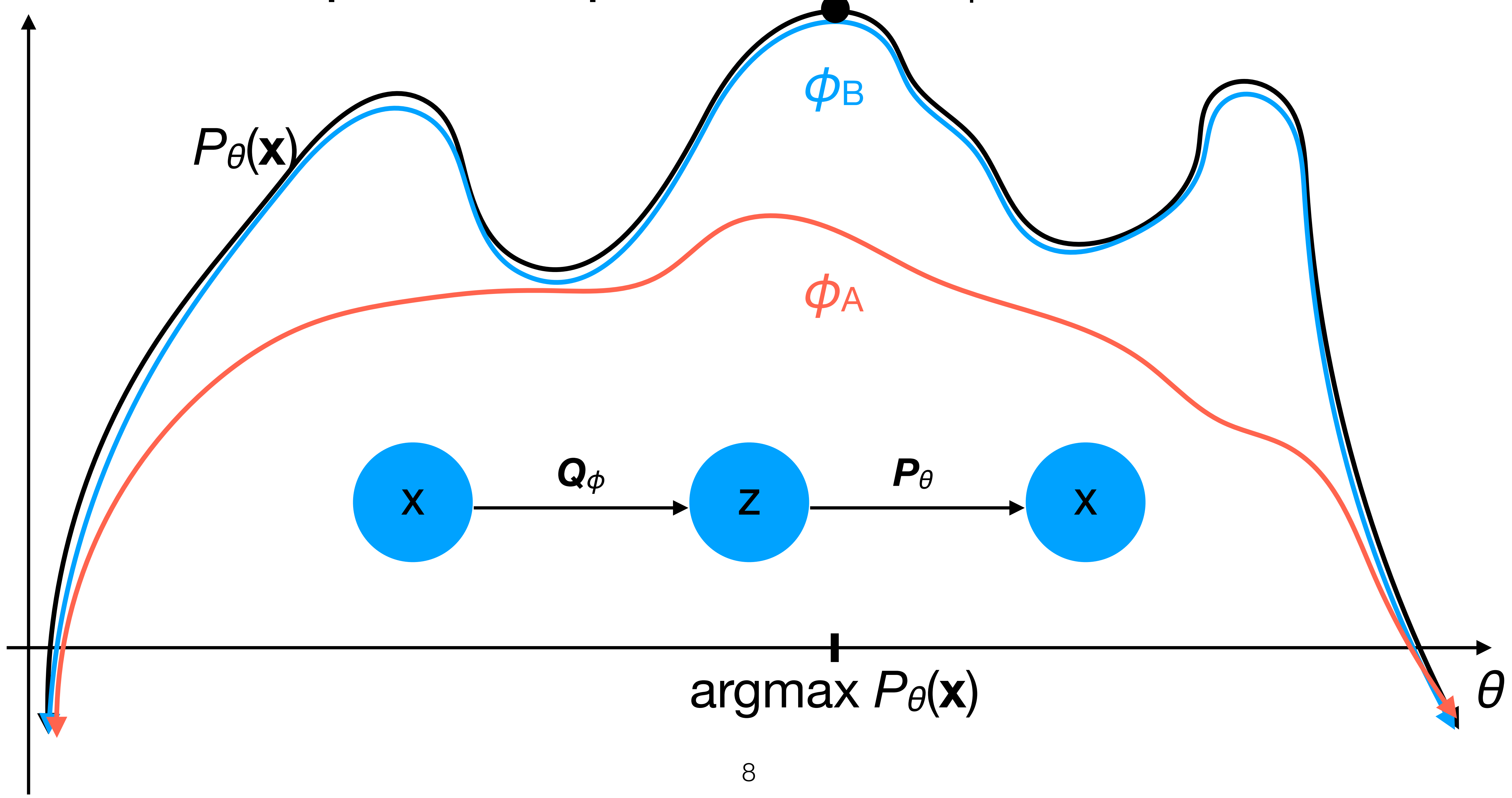
Maximizing the lower bound

- By iteratively updating Q w.r.t. ϕ, we can increase the upper bound (though it will never exceed $P_\theta(\mathbf{x})$).



Solution

- **However**: if the approximation is "too tight", then $Q(\mathbf{z} \mid \mathbf{x}) \approx P(\mathbf{z})$ — i.e., Q "ignores" $\mathbf{x}$ altogether and adheres "too much" to $P(\mathbf{z})$.
- P thus receives no specific information about $\mathbf{x}$ to reconstruct it with — despite the tightness of the bound, SGD has no ability to find the "good" values for θ.
- This is called **posterior collapse** and is a common problem with VAEs.



Avoiding Posterior Collapse

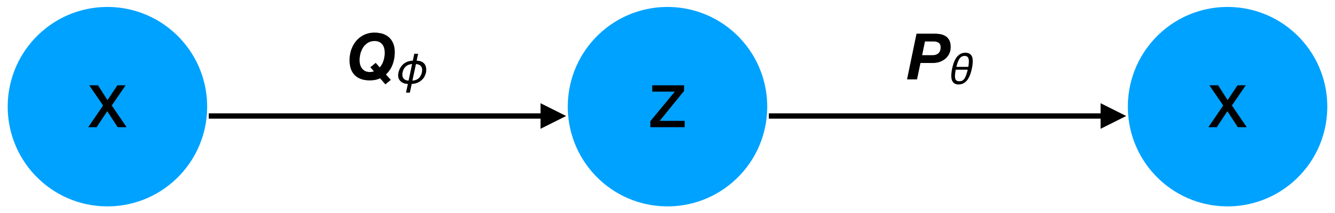
- It is common practice when training VAEs to weight (with hyperparameter *β*) how much we care about the DL term versus the reconstruction term:
    - Modified ELBO:

$$-\beta D_{\mathrm{KL}}(Q_\phi(\mathbf{z} \mid \mathbf{x}) \parallel P(\mathbf{z})) + \log P_\theta(\mathbf{x} \mid \mathbf{z})$$

Blurry images

- VAEs are known to produce blurry images. Why?

- Consider a decoder likelihood model $P(\mathbf{x} \mid \mathbf{z}) = \mathcal{N}(\mu_{\mathbf{x}}, \mathbf{I})$ – i.e., based on a latent **z**, the mean $\mu_{\mathbf{x}}$ of a Gaussian is chosen as the expected generated image.



Blurry images

- Now, suppose two training images **x**, **x**' are similar to each other and have identical latent codes **z***.

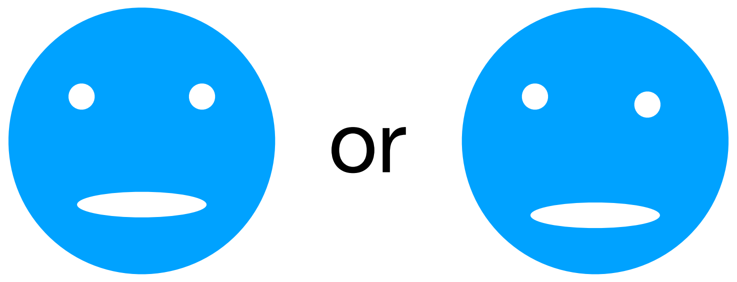
  - Why? Because in practice, it's difficult for latent **z** to perfectly encode all features of an image **x**.

$$\begin{tikzcd} x \arrow[r, "Q_\phi"] & z \arrow[r, "P_\theta"] & x \end{tikzcd}$$

Blurry images

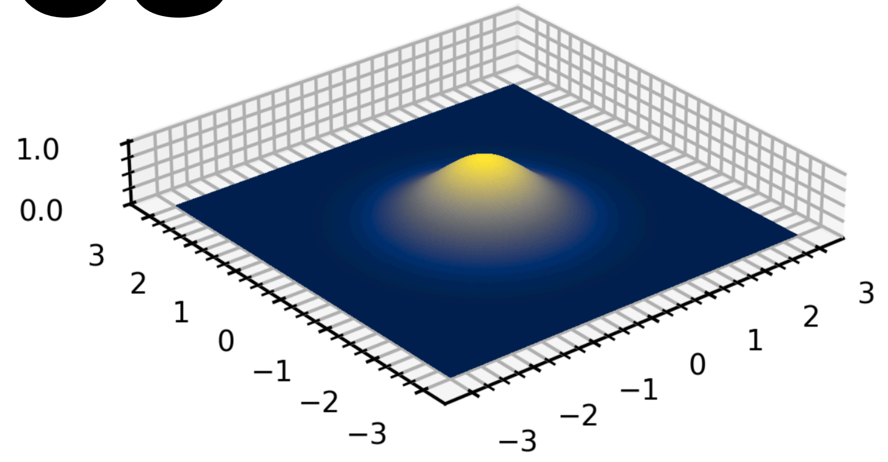
- Ideally, when sampling from $P(\mathbf{x} \mid \mathbf{z}^*)$, we would obtain either of the two images, and they would both be "sharp" images.

$$\mathbf{x} \sim P(\mathbf{x} \mid \mathbf{z}^*)$$



$$\begin{tikzcd} x \arrow[r, "Q_\phi"] & z \arrow[r, "P_\theta"] & x \end{tikzcd}$$

Blurry images



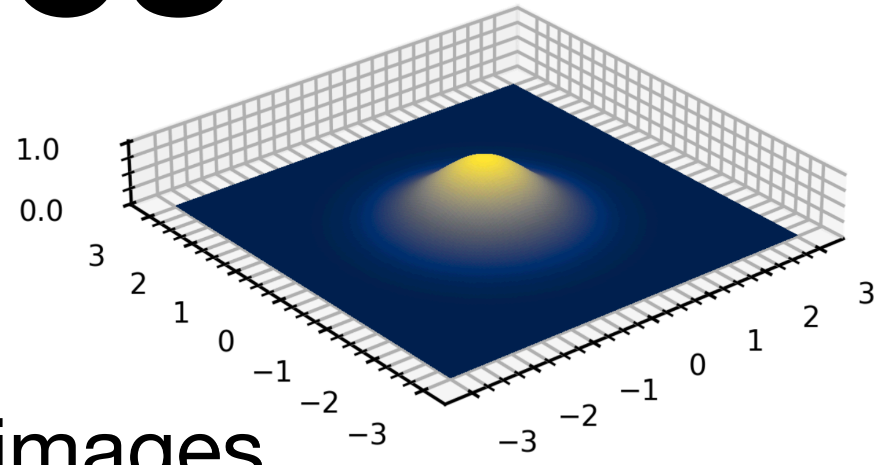
- However, a Gaussian model

$$\mathcal{N}(\mu_{\mathbf{x}}, \mathbf{I}) \propto \exp\left(-\frac{1}{2}\,|\,\mathbf{x} - \mu_{\mathbf{x}}\,|^2\right)$$

  cannot express this — it is a unimodal distribution where likelihood vanishes as **x** deviates from the mean $\boldsymbol{\mu}_{\mathbf{x}}$.

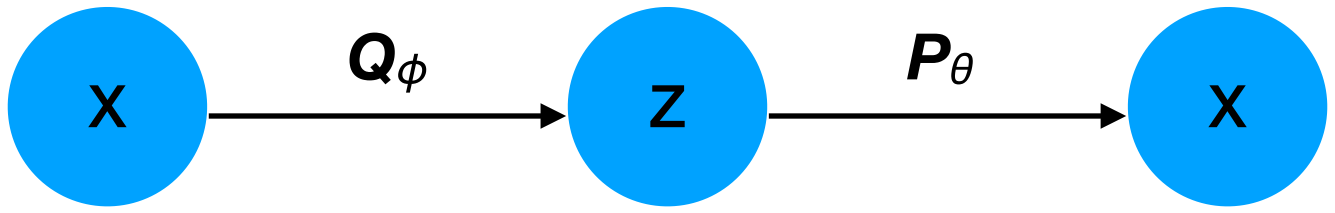
$$\begin{tikzcd} X \arrow[r, "\boldsymbol{Q}_\phi"] & Z \arrow[r, "\boldsymbol{P}_\theta"] & X \end{tikzcd}$$

Blurry images



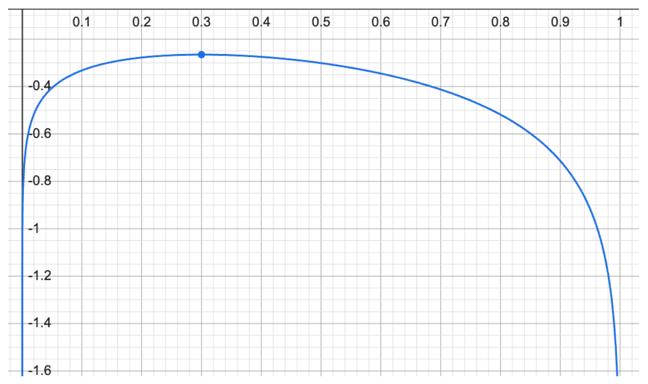
- For this reason, the decoder will maximize the likelihood over {**x**, **x**'} by picking $\boldsymbol{\mu_x}$ to be the *average* of the two images (see homework 1, part 4), creating blurry predictions.

$$\mathbf{x} \sim P(\mathbf{x} \mid \mathbf{z}^*)$$



Blurry images

- The same problem exists for other likelihood models and reconstruction losses as well, e.g., binary cross-entropy.



$$\mathbf{x} \sim P(\mathbf{x} \mid \mathbf{z}^*)$$

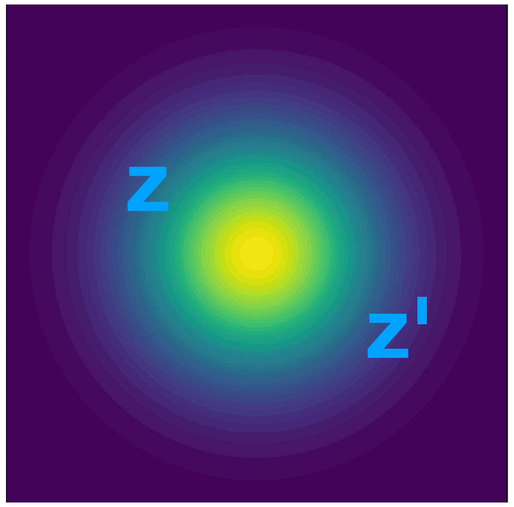
$$\begin{tikzcd} \mathrm{X} \arrow[r, "Q_\phi"] & \mathrm{Z} \arrow[r, "P_\theta"] & \mathrm{X} \end{tikzcd}$$

Vector Quantized (VQ) VAEs

(But let’s be discrete about it…)

Why discrete representations?

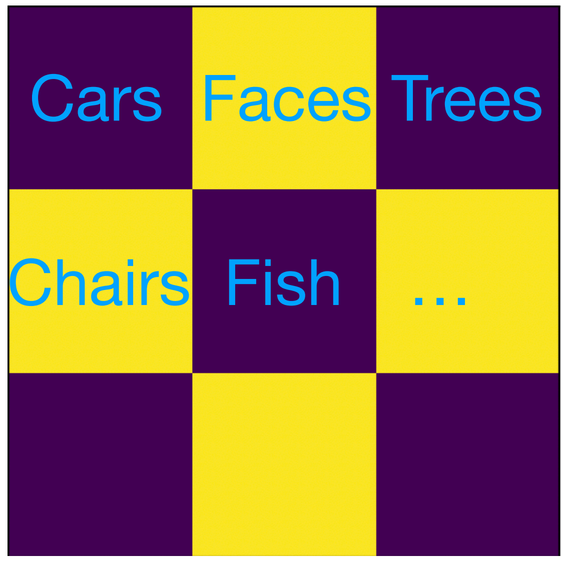
- By making small changes to latent code **z**, the VAE decoder *P*(**x** | **z**) can produce images that vary smoothly in their style & semantics, e.g.:
  - Faces that change from happy (**z**) to sad (**z**').
  - Cars that change from SUV (**z**) to compact car (**z**').



Latent space $\mathbb{R}^d$

Why discrete representations?

- But can a car really "morph" smoothly into a face?
- <u>Maybe there are</u> actually multiple, disjoint subspaces of the latent space that represent different object classes?

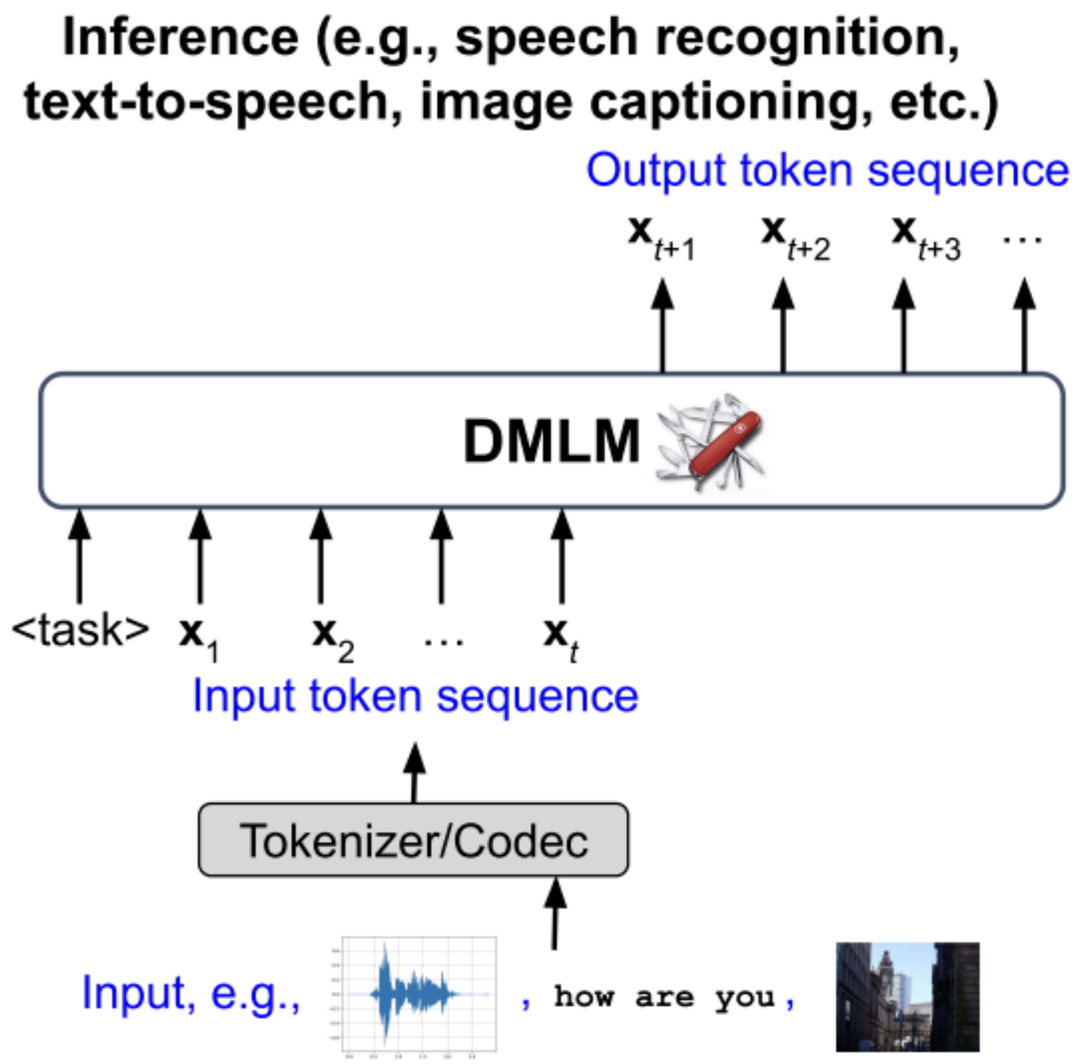


Latent space $\mathbb{R}^d$

Why discrete representations?

- As we'll discuss later, discrete representations are also more convenient for autoregression.

- Representing parts of images, audios, text, etc. as discrete tokens (rather than continuous vectors) allows us to combine multiple modalities in one model — e.g., a multimodal LLM.

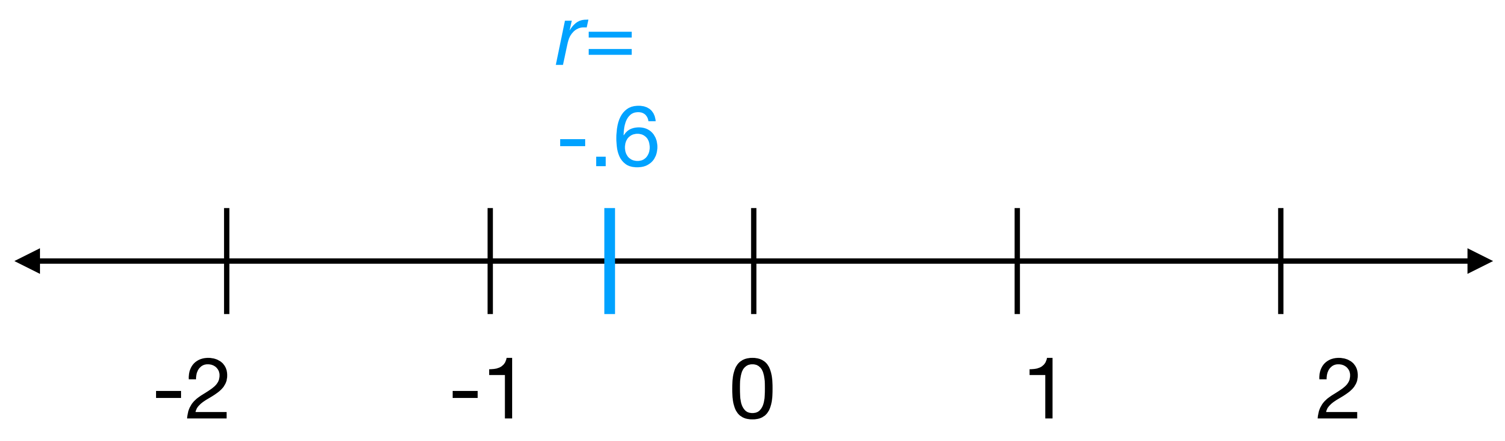

Inference (e.g., speech recognition, text-to-speech, image captioning, etc.)
Output token sequence
$\mathbf{x}_{t+1}$ $\mathbf{x}_{t+2}$ $\mathbf{x}_{t+3}$ …
DMLM
<task> $\mathbf{x}_1$ $\mathbf{x}_2$ … $\mathbf{x}_t$
Input token sequence
Tokenizer/Codec
Input, e.g.,
, how are you ,

Quantization

- **Quantization**: approximate an input *x* from an uncountable/infinite set with an element *y* from a countable/finite set.

- Examples:

    - Round from a real number (uncountable) *r* to its nearest integer *j* (countable).

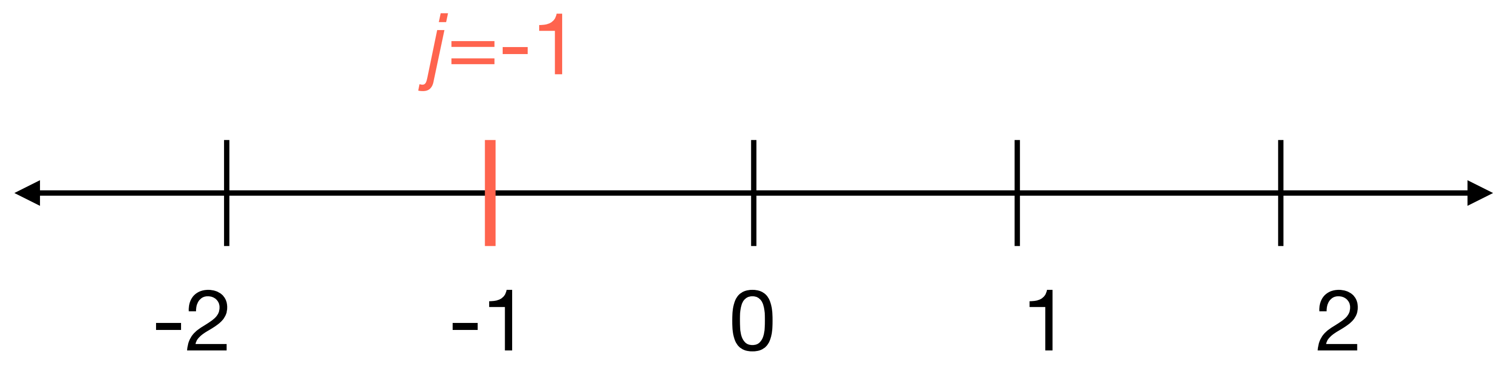


Quantization

- **Quantization**: approximate an input *x* from an uncountable/infinite set with an element *y* from a countable/finite set.

- Examples:

  - Round from a real number (uncountable) *r* to its nearest integer *j* (countable).

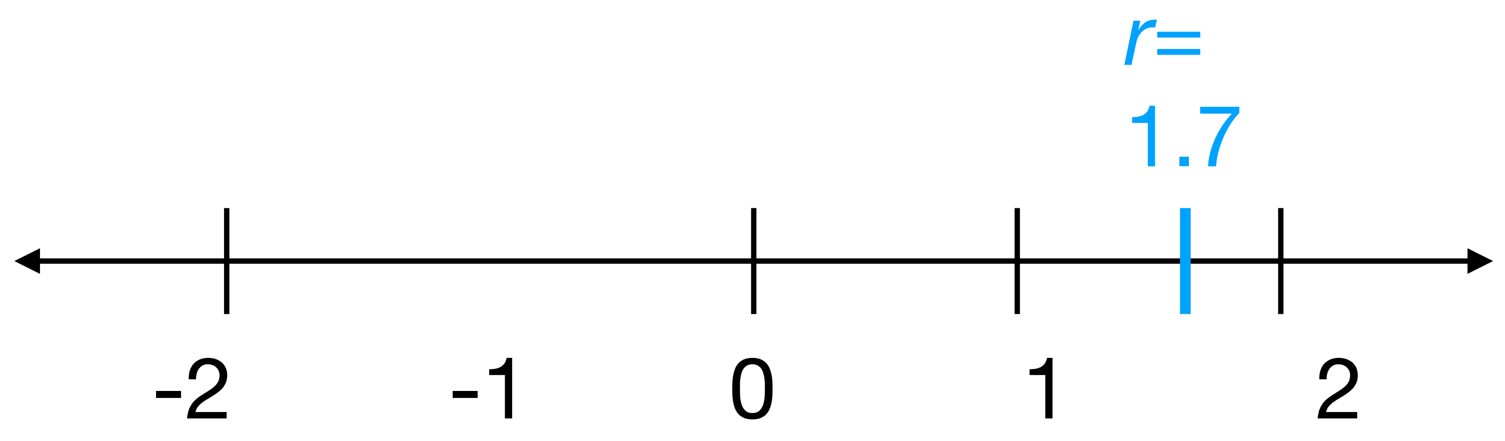


Quantization

- **Quantization**: approximate an input *x* from an uncountable/infinite set with an element *y* from a countable/finite set.

- Examples:

  - Round from a real number (uncountable) *r* to its nearest integer *j* (countable).

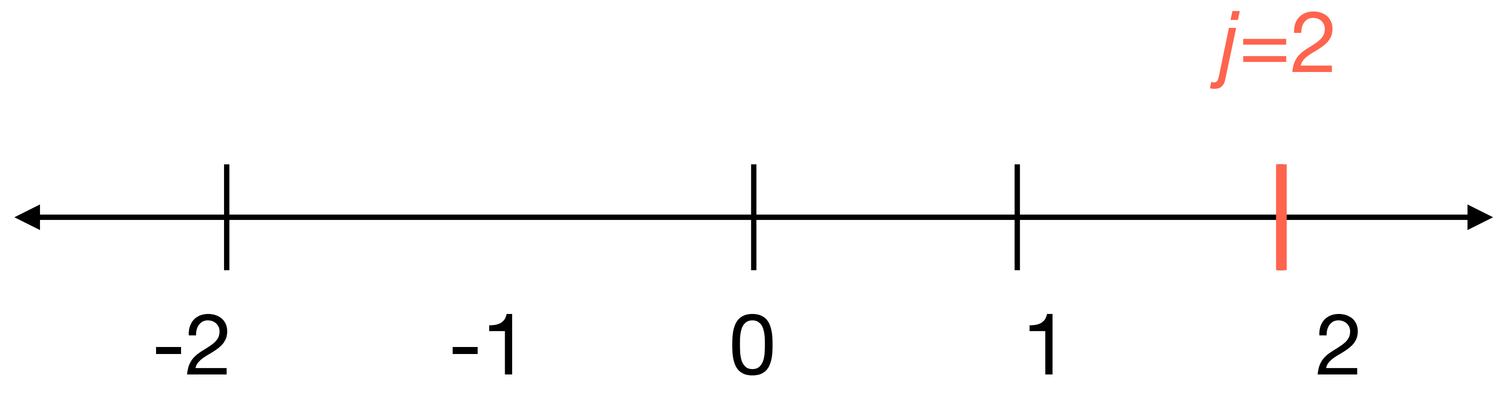


Quantization

- **Quantization**: approximate an input *x* from an uncountable/infinite set with an element *y* from a countable/finite set.

- Examples:

    - Round from a real number (uncountable) *r* to its nearest integer *j* (countable).

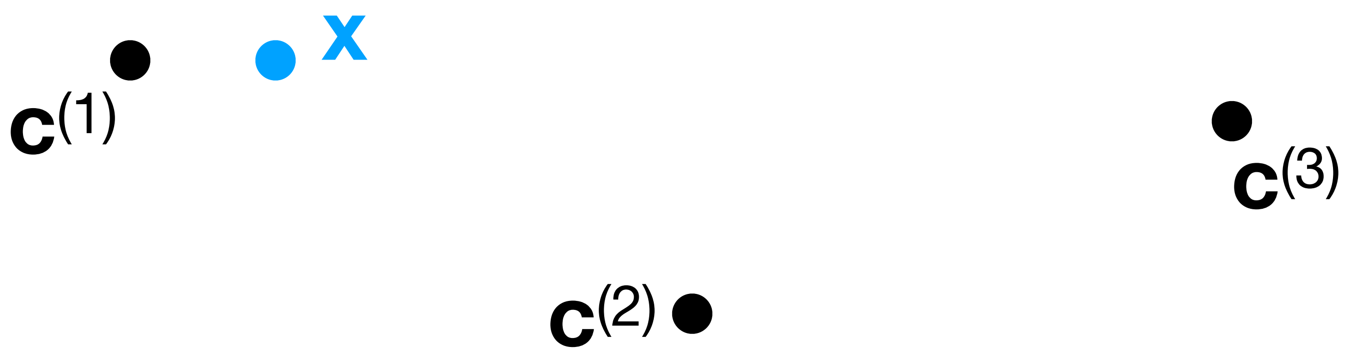


Quantization

- **Quantization**: approximate an input *x* from an uncountable/infinite set with an element *y* from a countable/finite set.

- Examples:

    - Vector quantization (VQ): Map real vector (uncountable) $\mathbf{x} \in \mathbb{R}^m$ to one of a finite set of centroids $\mathbf{c}^1, \ldots, \mathbf{c}^{(K)}$.

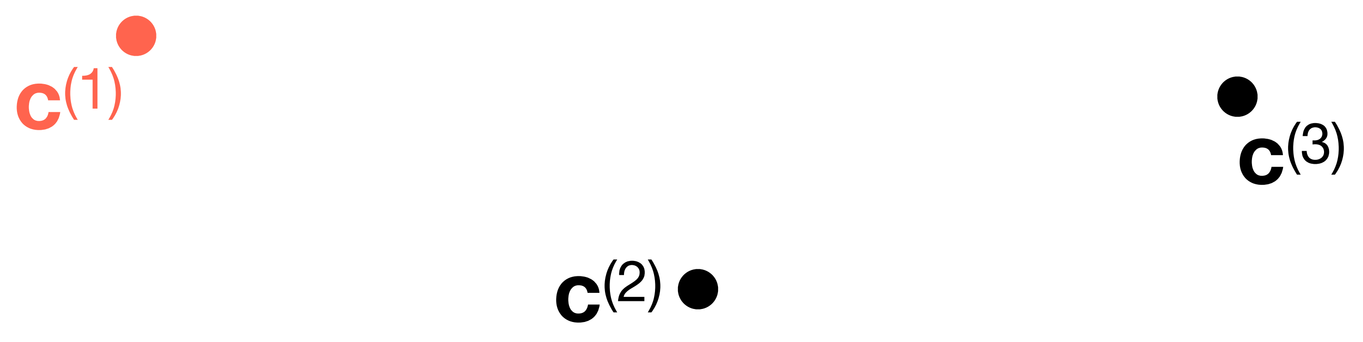


Quantization

- **Quantization**: approximate an input *x* from an uncountable/infinite set with an element *y* from a countable/finite set.

- Examples:

    - Vector quantization (VQ): Map real vector (uncountable) $\mathbf{x} \in \mathbb{R}^m$ to one of a finite set of centroids $\mathbf{c}^1, \ldots, \mathbf{c}^{(K)}$.

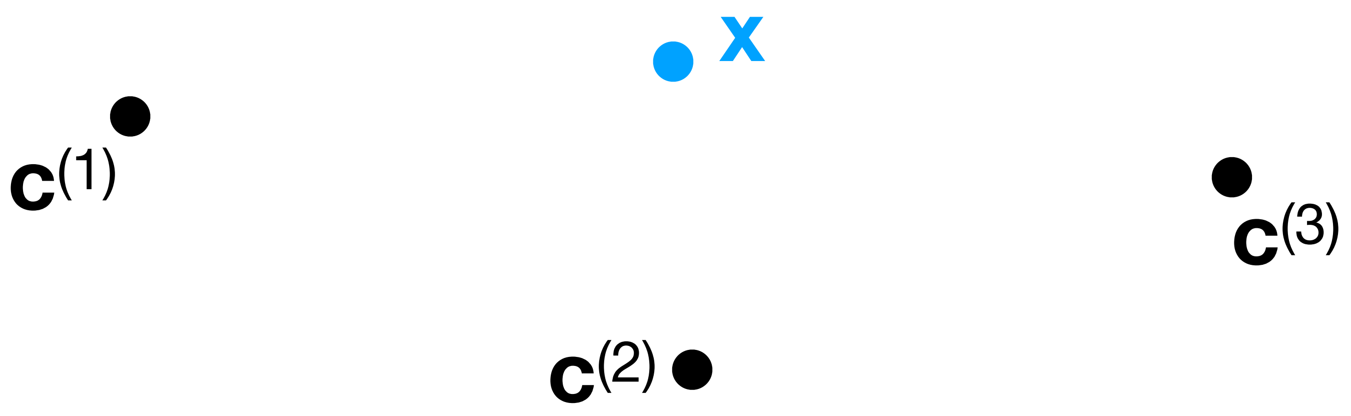


Quantization

- **Quantization**: approximate an input *x* from an uncountable/infinite set with an element *y* from a countable/finite set.

- Examples:

    - Vector quantization (VQ): Map real vector (uncountable) $\mathbf{x} \in \mathbb{R}^m$ to one of a finite set of centroids $\mathbf{c}^1, \ldots, \mathbf{c}^{(K)}$.



Quantization

- **Quantization**: approximate an input *x* from an uncountable/infinite set with an element *y* from a countable/finite set.

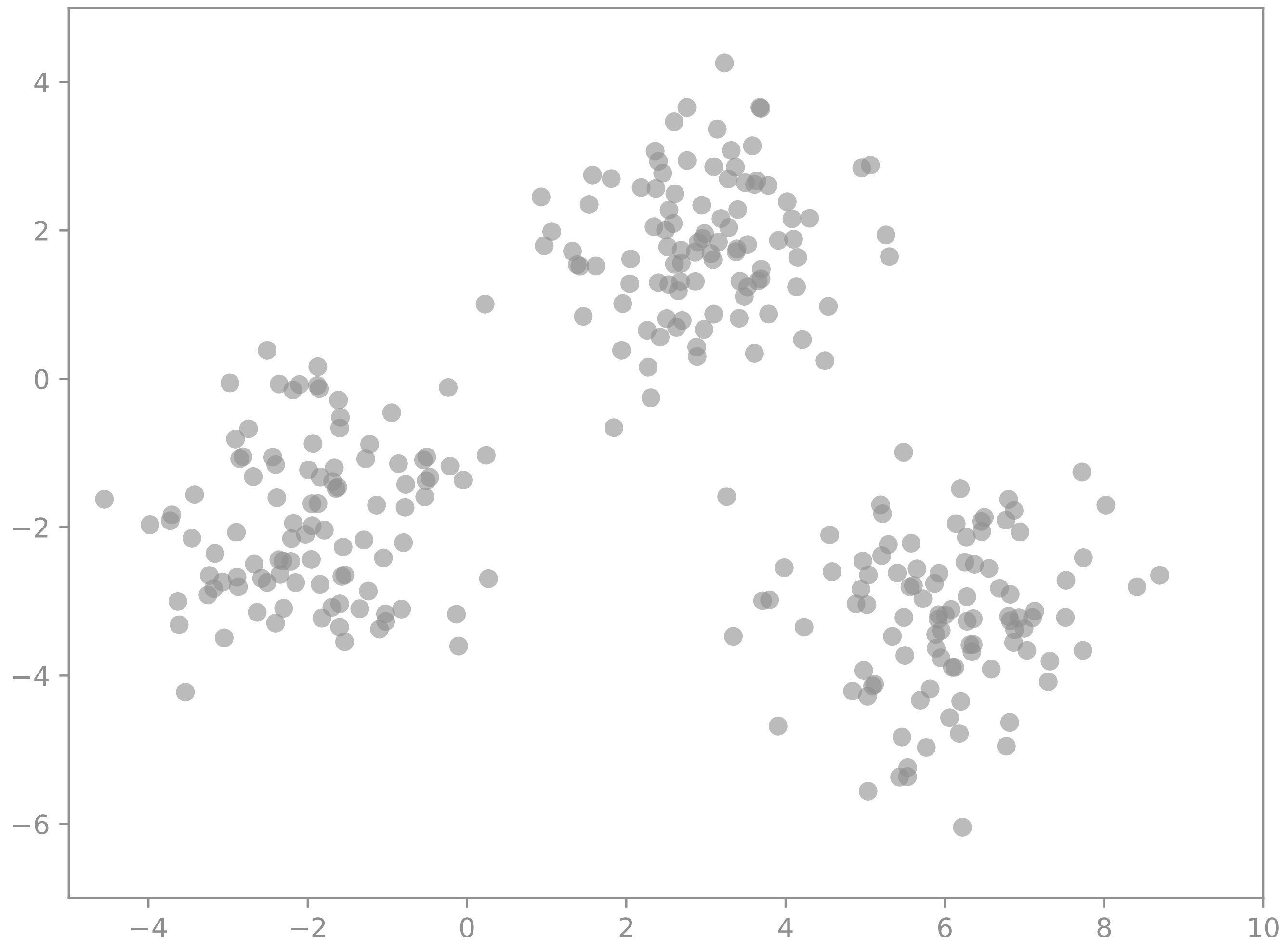
- Examples:

  - Vector quantization (VQ): Map real vector (uncountable) $\mathbf{x} \in \mathbb{R}^m$ to one of a finite set of centroids $\mathbf{c}^1, \ldots, \mathbf{c}^{(K)}$.

$\mathbf{c}^{(1)}$
$\mathbf{c}^{(3)}$
$\mathbf{c}^{(2)}$

Vector Quantization

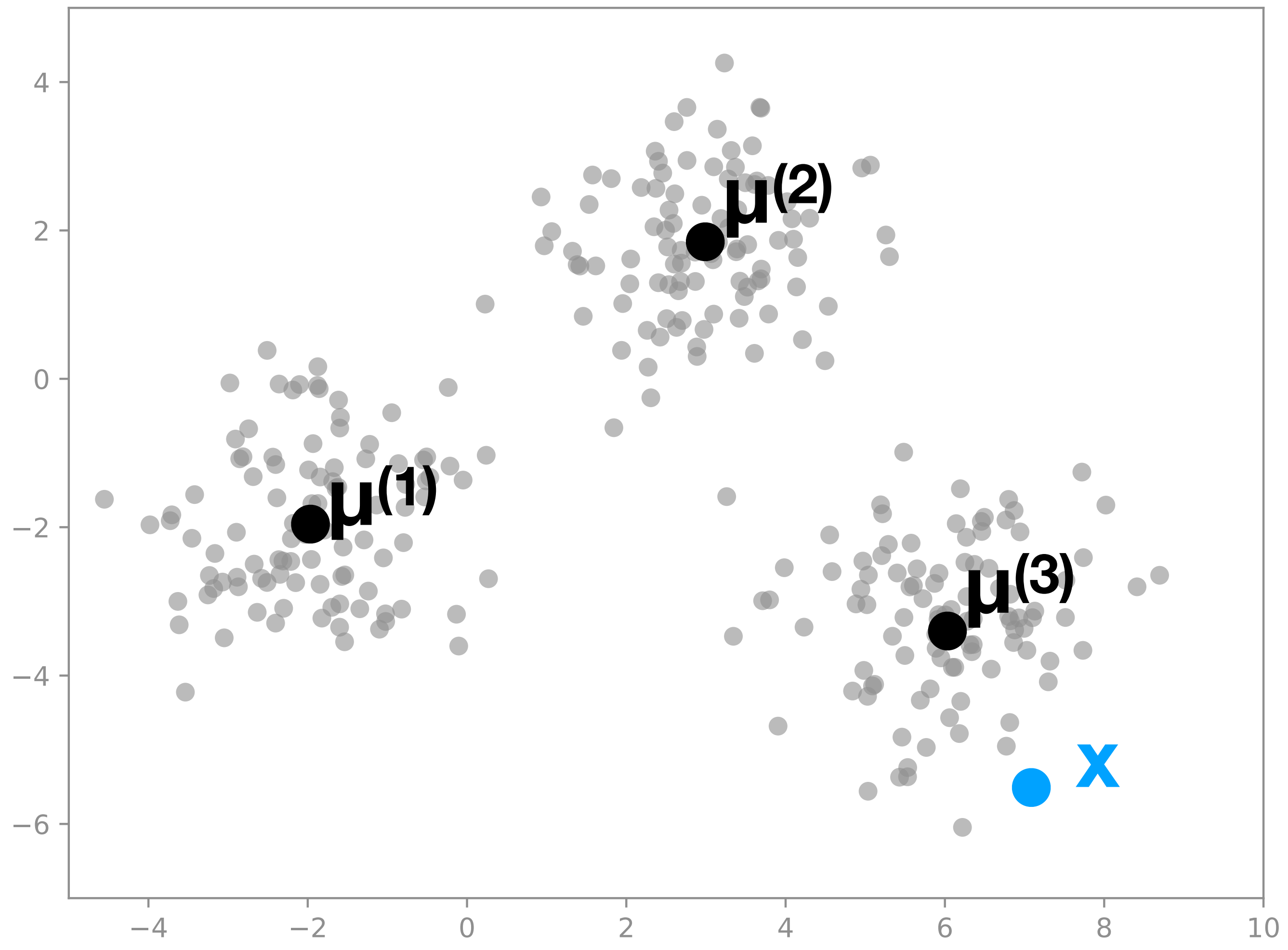
- Consider the set of points { $\mathbf{x}^{(i)}$} shown to the right.



Vector Quantization

- Consider the set of points { $\mathbf{x}^{(i)}$} shown to the right.

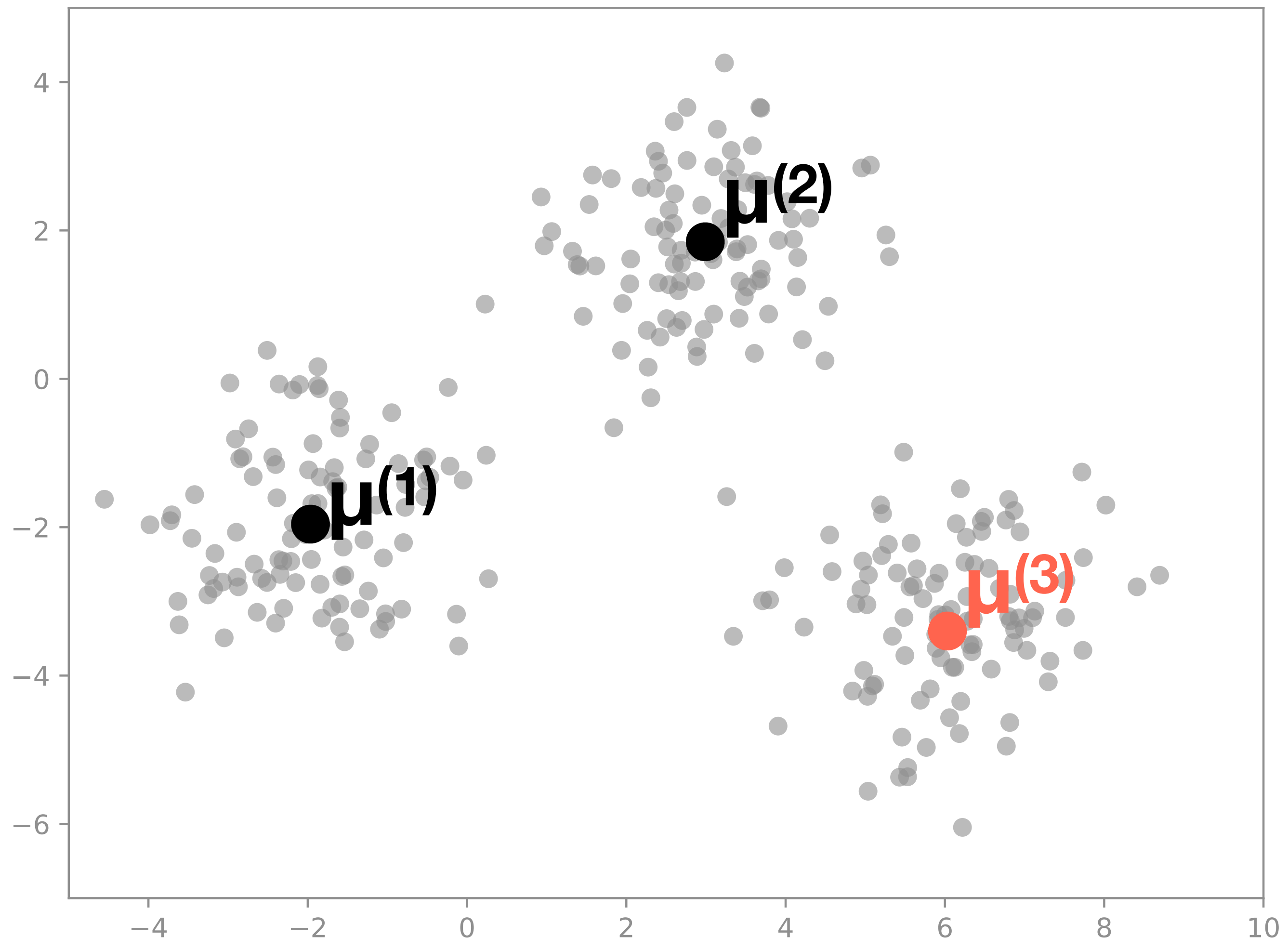
- We could quantize each $\mathbf{x}$ by mapping it to the closest vector from the set { $\boldsymbol{\mu}^{(1)}$, $\boldsymbol{\mu}^{(2)}$, $\boldsymbol{\mu}^{(3)}$ }.



Vector Quantization

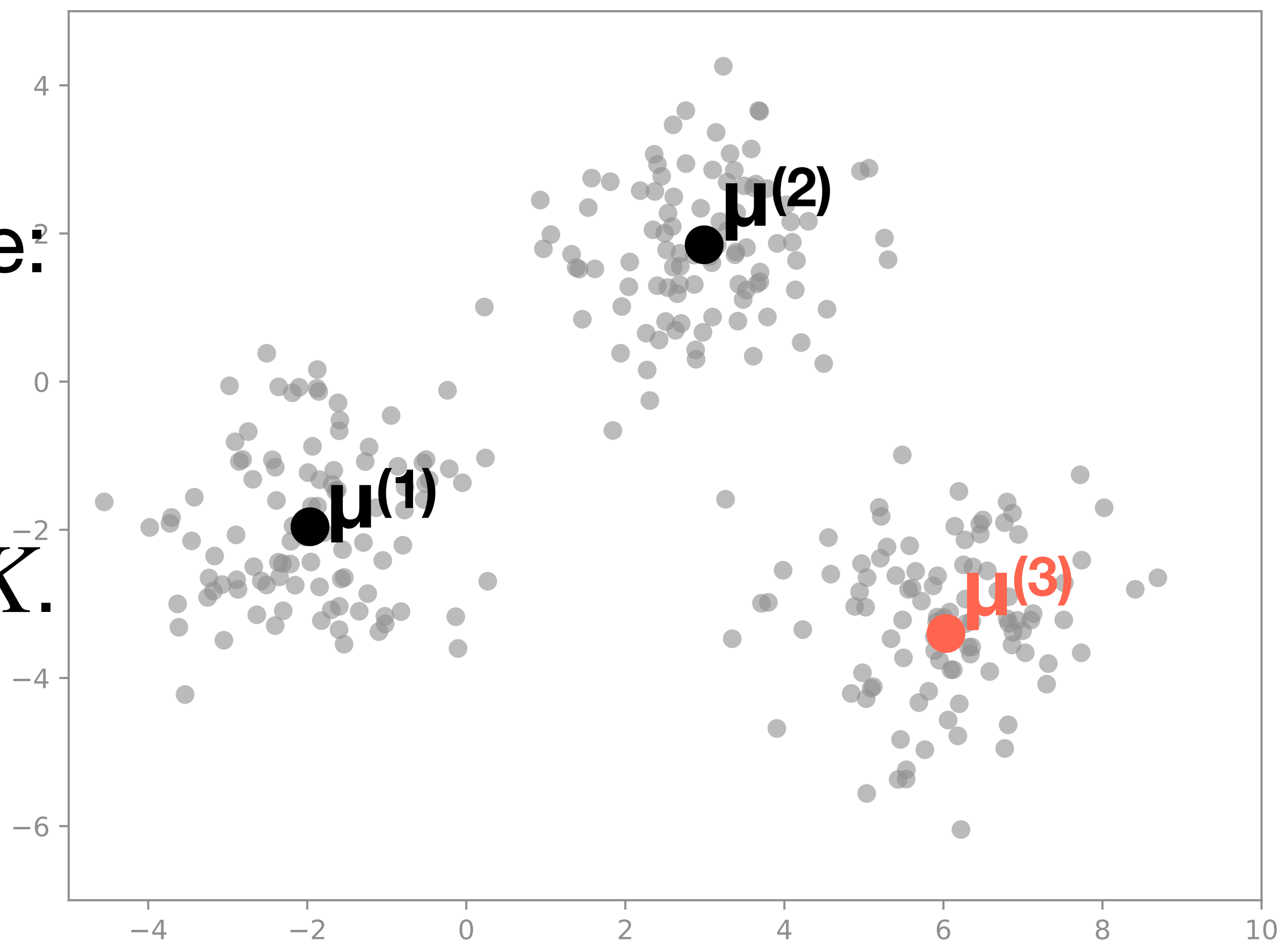
- Consider the set of points { $\mathbf{x}^{(i)}$} shown to the right.

- We could quantize each $\mathbf{x}$ by mapping it to the closest vector from the set { $\boldsymbol{\mu}^{(1)}$, $\boldsymbol{\mu}^{(2)}$, $\boldsymbol{\mu}^{(3)}$ }.



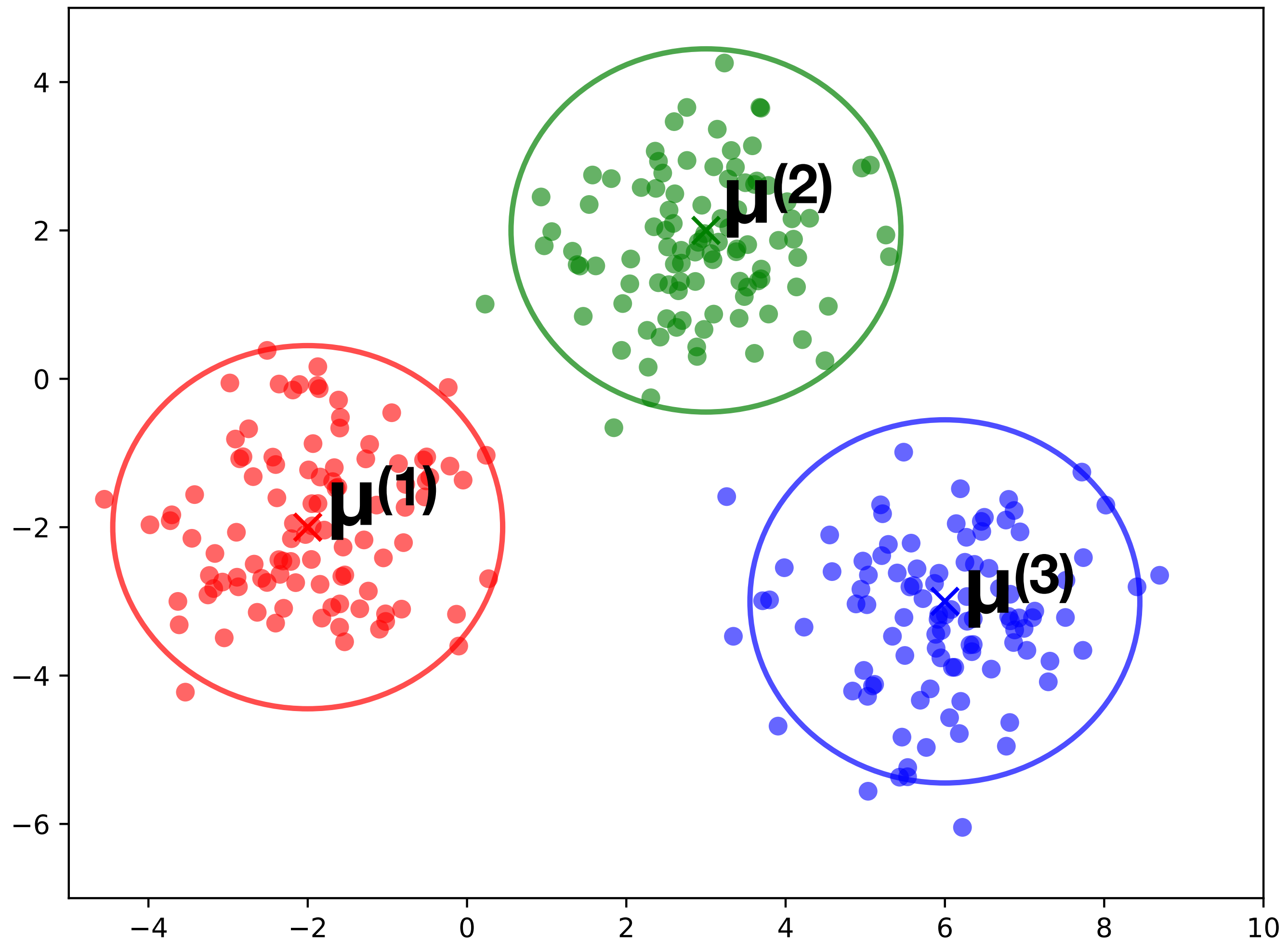
Vector Quantization

- We can (equivalently) consider this process to be:
  - A map from $\mathbb{R}^m$ to $\mathcal{S} \subset \mathbb{R}$, where $|\mathcal{S}| = K$.
    Produces a *vector*
  - A map from $\mathbb{R}^m$ to $\{1, \ldots, K\}$.
    Produces an integer/index



Vector Quantization

- The set of centroids $\{ \boldsymbol{\mu}^{(1)}, \boldsymbol{\mu}^{(2)}, \boldsymbol{\mu}^{(3)} \}$ is typically obtained with *K*-means or a GMM.



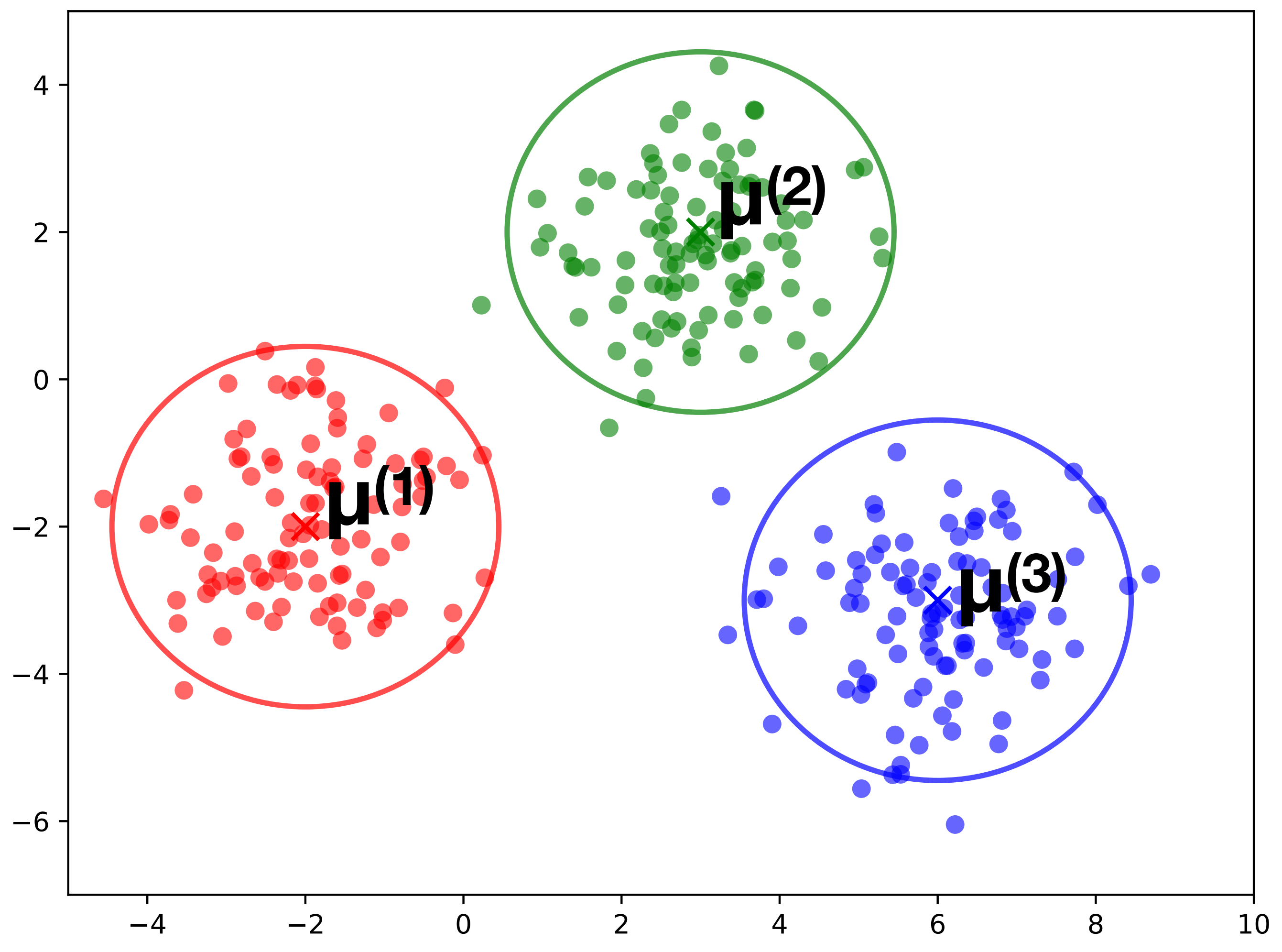
Vector Quantization

- The set of centroids $\{ \boldsymbol{\mu}^{(1)}, \boldsymbol{\mu}^{(2)}, \boldsymbol{\mu}^{(3)} \}$ is typically obtained with *K*-means or a GMM.

- *K*-Means objective:

$$\min_{\{\mu^{(k)}\},\{z^{(i)}\}} \sum_{i=1}^{n} \|\mathbf{x}^{(i)} - \mu^{(z^{(i)})}\|^2$$

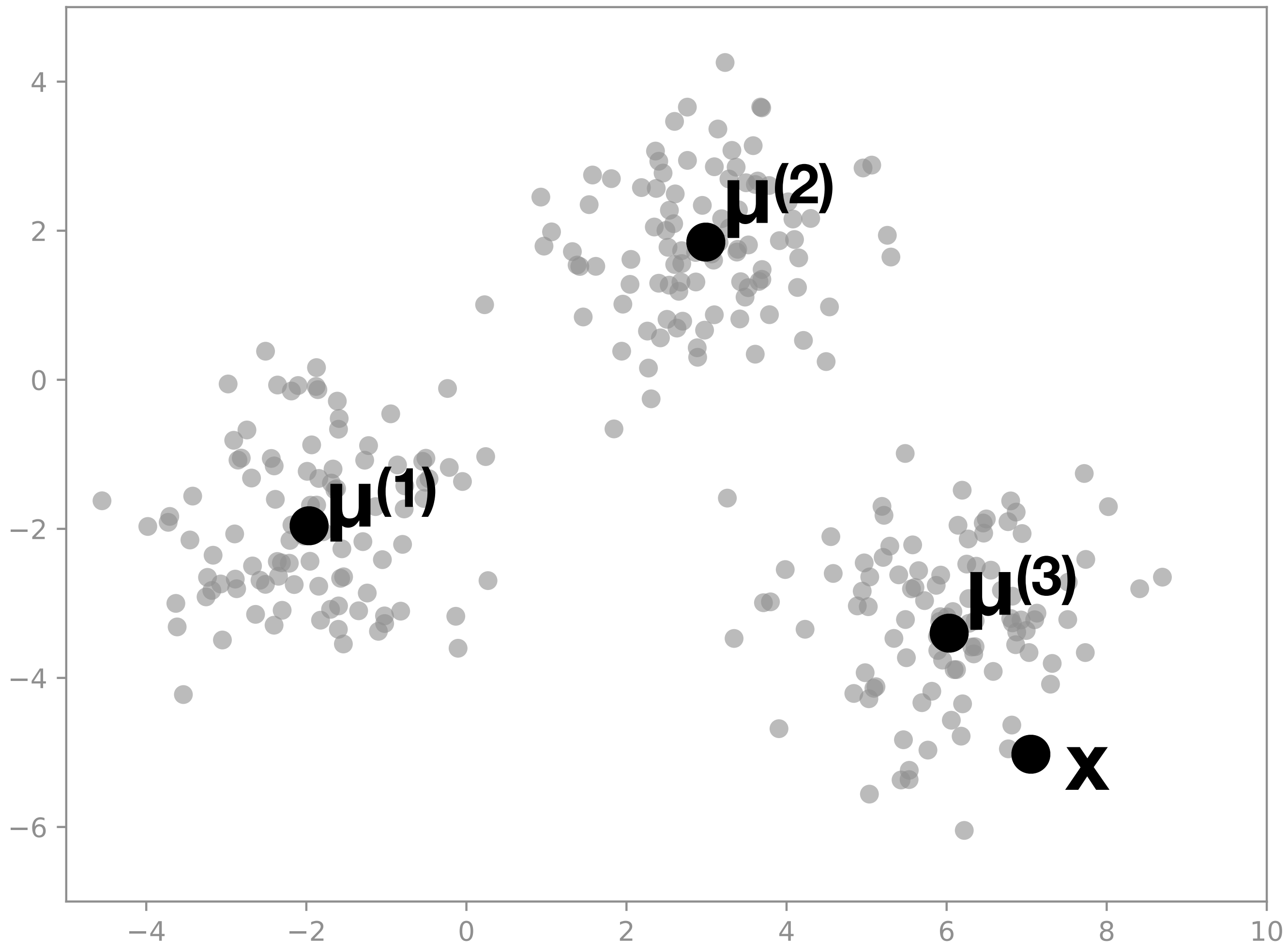
where $z^{(i)} \in \{1, \ldots, K\}$ indicates the cluster that $\mathbf{x}^{(i)}$ belongs to.



Shallow VQ AE

- Based on quantization, we can create a simple (discrete) "auto-encoder" that we train on a dataset { $\mathbf{x}^{(i)}$ }:

1. Q maps $\mathbf{x}$ to the latent code $z \in \{1, \ldots, K\}$ corresponding to the closest centroid $\boldsymbol{\mu}^{(z)}$.



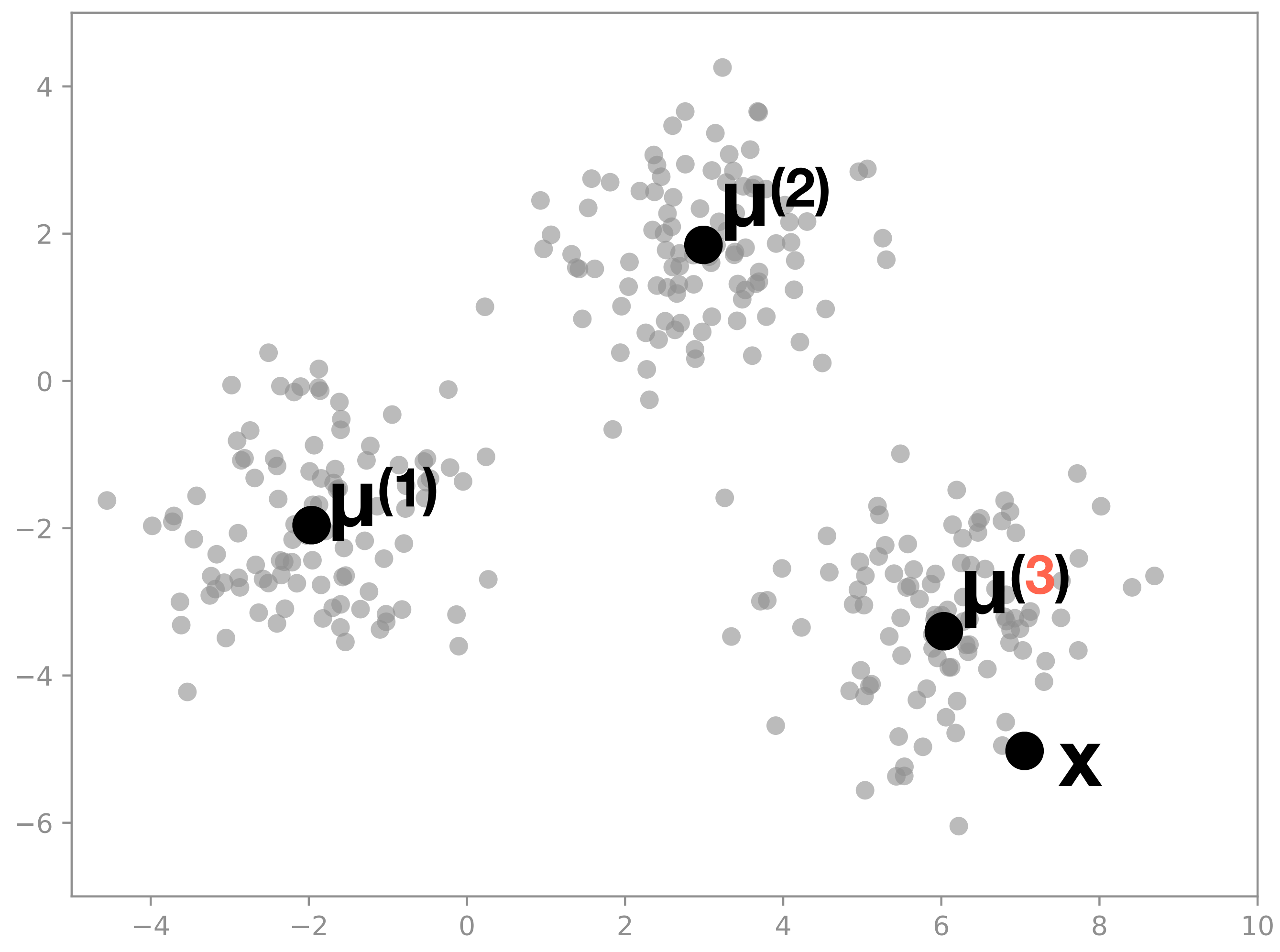
$[7,-5]^T$

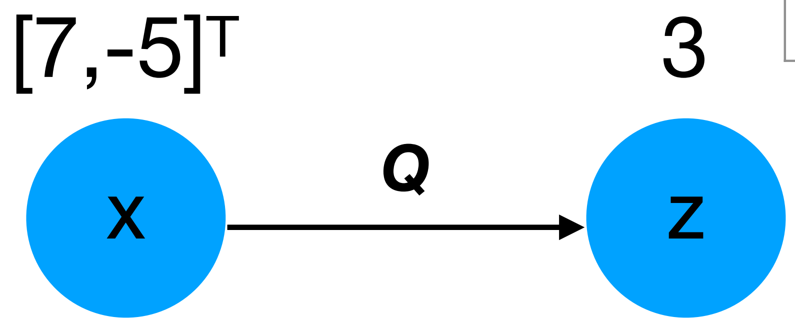
$$\begin{tikzcd} x \arrow[r, "Q"] & z \end{tikzcd}$$

Shallow VQ AE

- Based on quantization, we can create a simple (discrete) "auto-encoder" that we train on a dataset { $\mathbf{x}^{(i)}$ }:

  1. Q maps $\mathbf{x}$ to the latent code $z \in \{1, \ldots, K\}$ corresponding to the closest centroid $\boldsymbol{\mu}^{(z)}$.

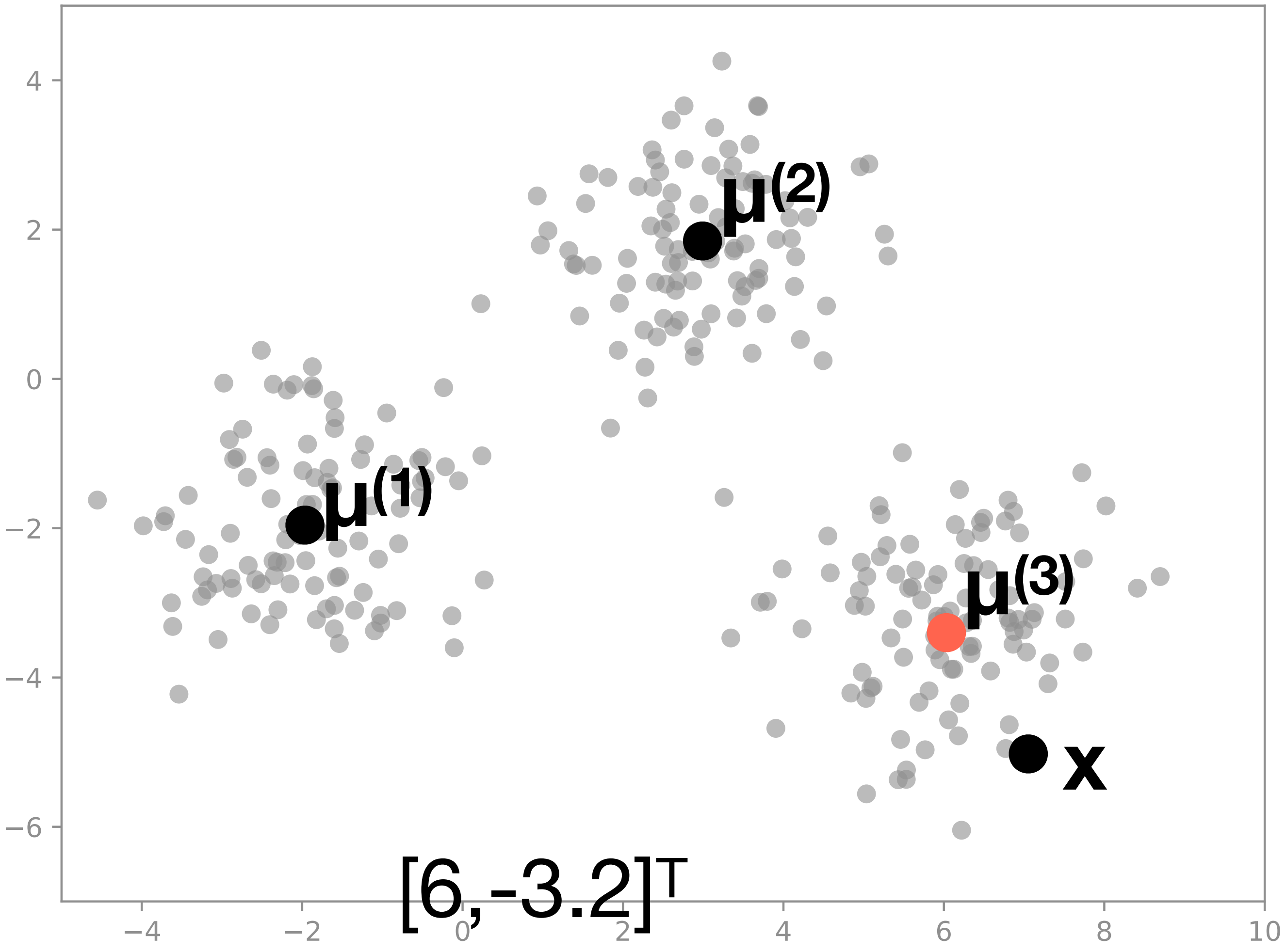




Shallow VQ AE

- Based on quantization, we can create a simple (discrete) "auto-encoder" that we train on a dataset { $\mathbf{x}^{(i)}$ }:

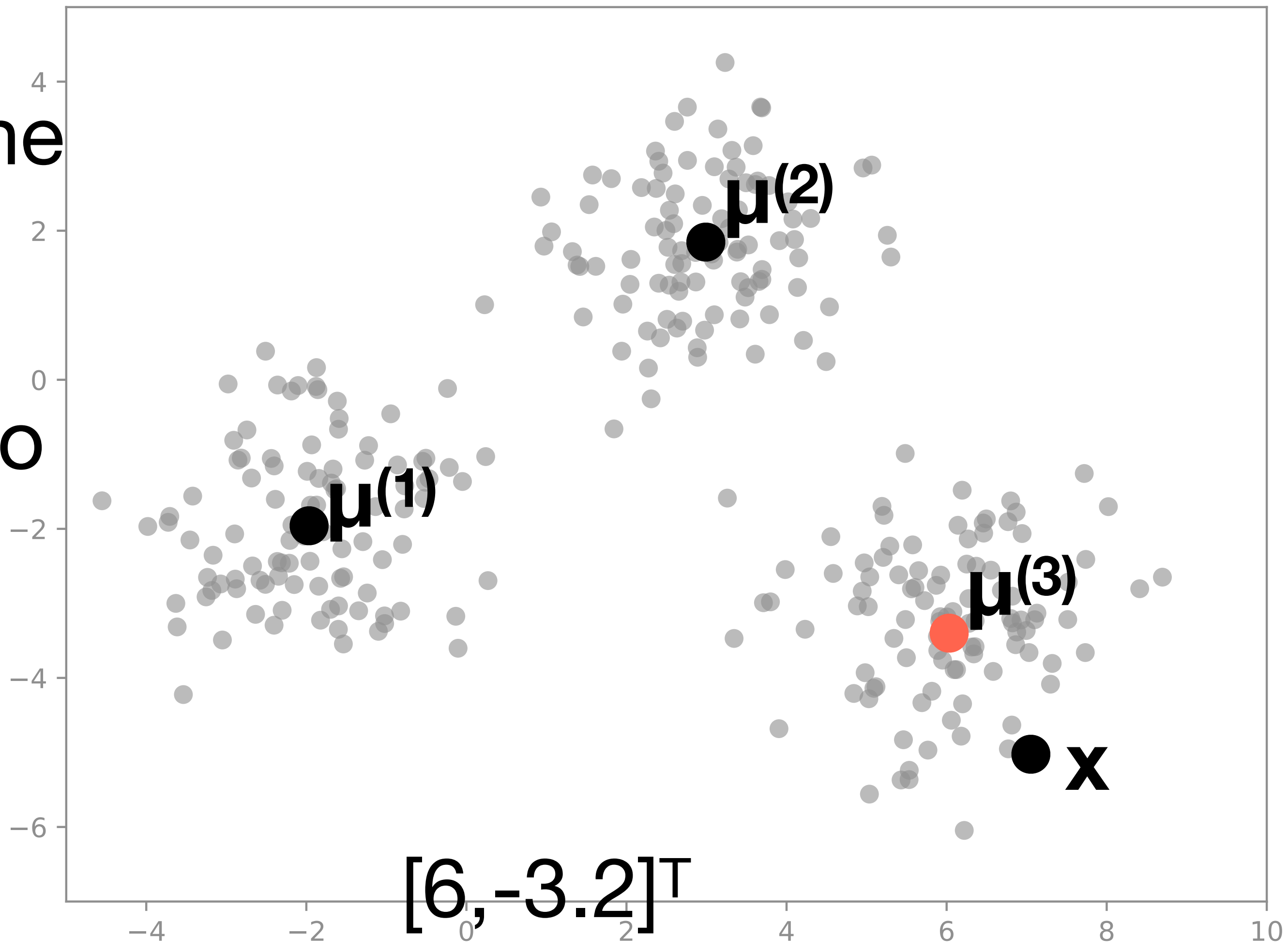
1. *Q* maps **x** to the latent code $z \in \{1, \ldots, K\}$ corresponding to the closest centroid $\boldsymbol{\mu}^{(z)}$.
2. *P* maps *z* to vector $\boldsymbol{\mu}^{(z)}$ as the estimate for **x**.

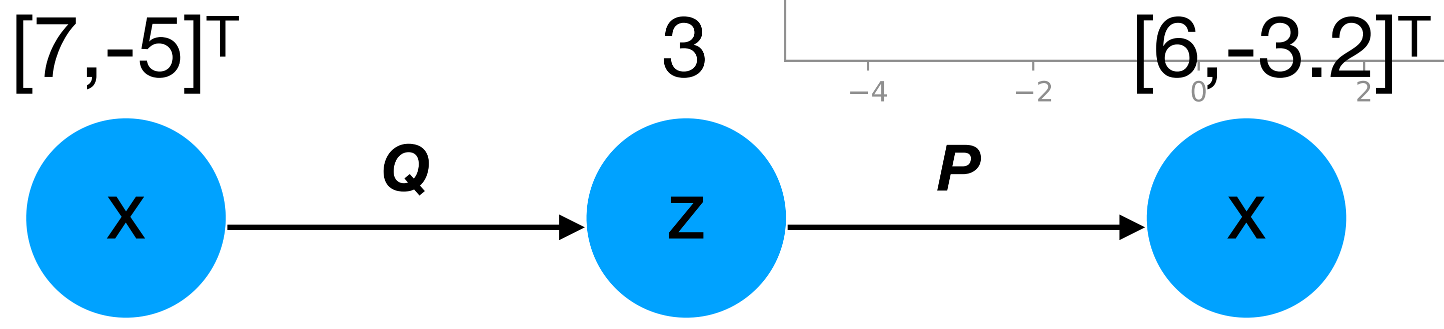


$$\begin{tikzcd}
{[7,-5]^T} & 3 & {[6,-3.2]^T} \\
\mathrm{x} \arrow[r, "Q"] & \mathrm{z} \arrow[r, "P"] & \mathrm{x}
\end{tikzcd}$$

Shallow VQ AE

- We can easily train this simple "VQ AE" using *K*-means.
- It is **discrete** since the latent *z* is from a finite set { 1, …, *K* }.
- The reconstruction error of the training data is just the SSD.
- There are no guarantees as to how many points belong to each cluster.

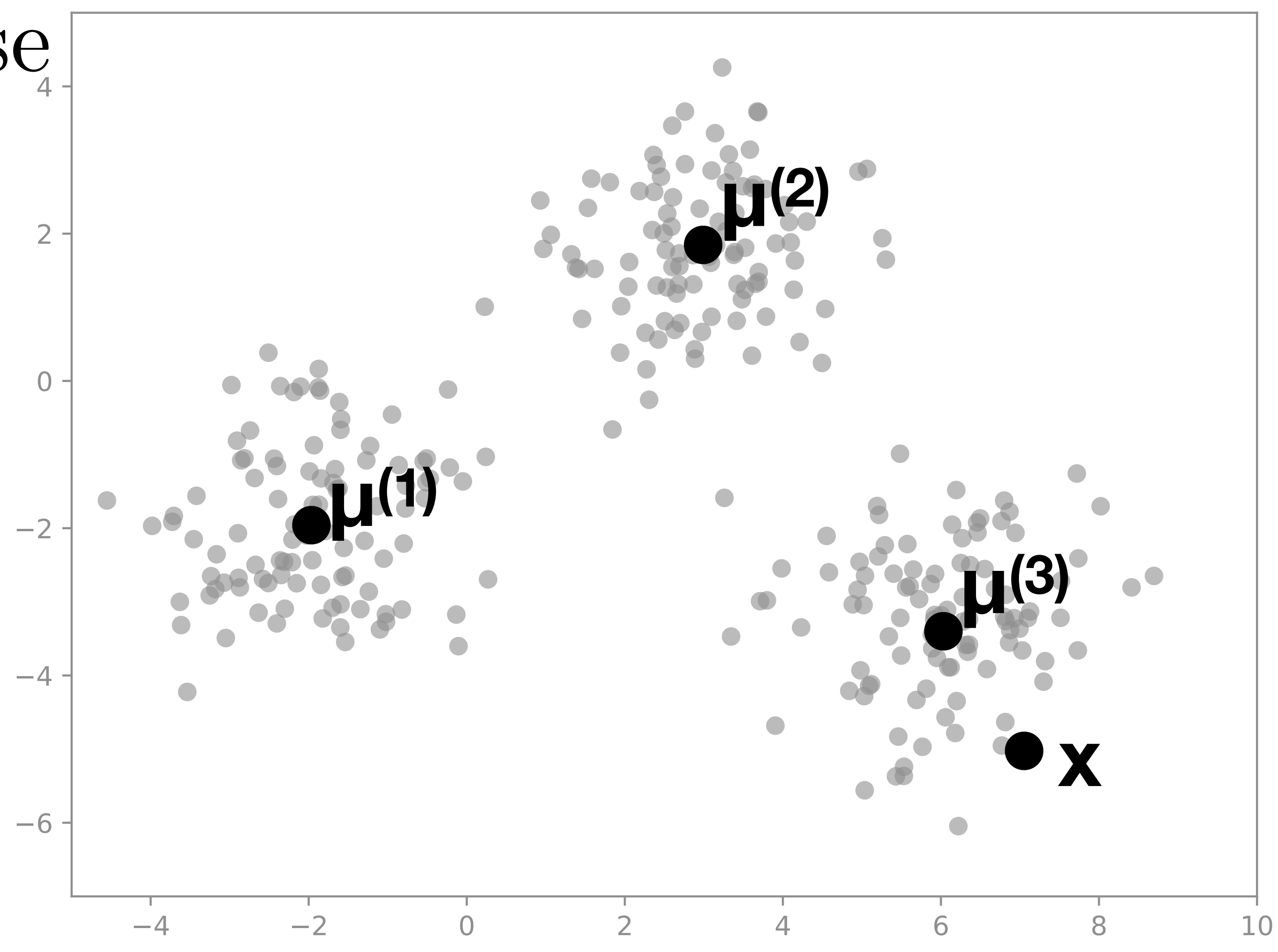




Shallow VQ VAE

- We can also construct a simple discrete "VAE":
    1. Q maps **x** deterministically to index of closest centroid:

$$Q(z \mid \mathbf{x}) = \begin{cases} 1 & \text{if } z = \arg\min_k \|\mathbf{x} - \mu^{(k)}\|^2 \\ 0 & \text{otherwise} \end{cases}$$



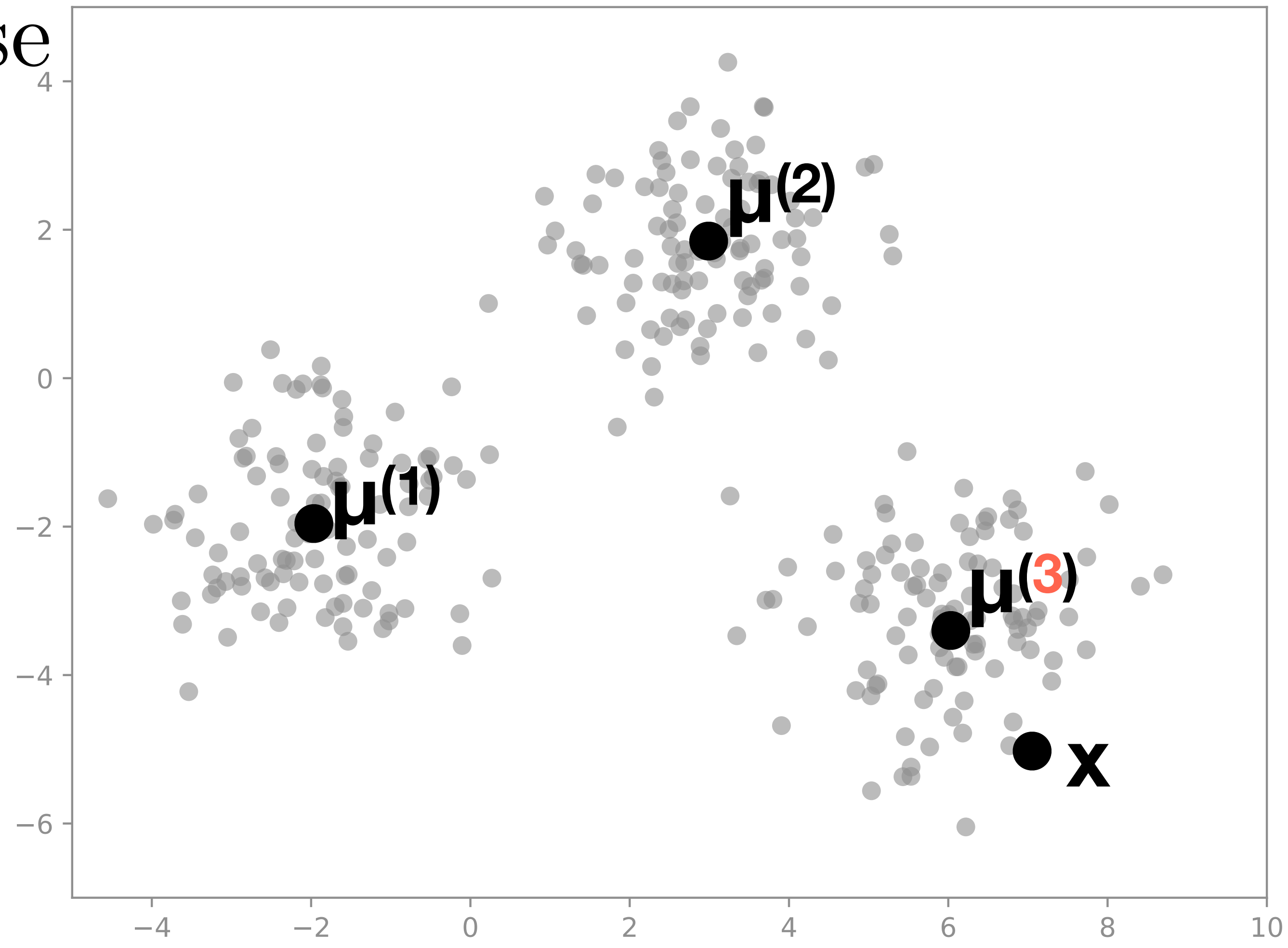
$[7,-5]^T$

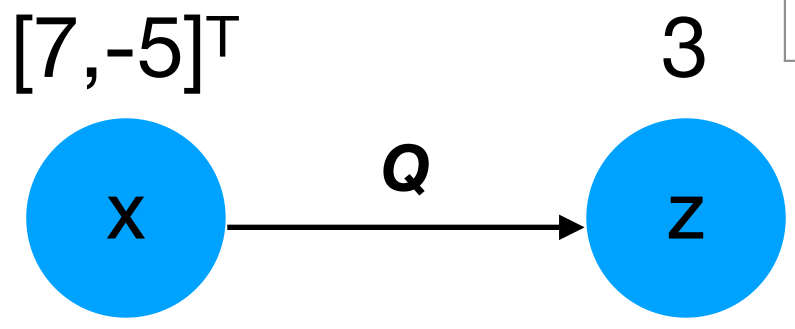
$$\begin{tikzcd} x \arrow[r, "Q"] & z \end{tikzcd}$$

Shallow VQ VAE

- We can also construct a simple discrete “VAE”:
  1. *Q* maps **x** deterministically to index of closest centroid:

$$Q(z \mid \mathbf{x}) = \begin{cases} 1 & \text{if } z = \arg\min_k \|\mathbf{x} - \mu^{(k)}\|^2 \\ 0 & \text{otherwise} \end{cases}$$





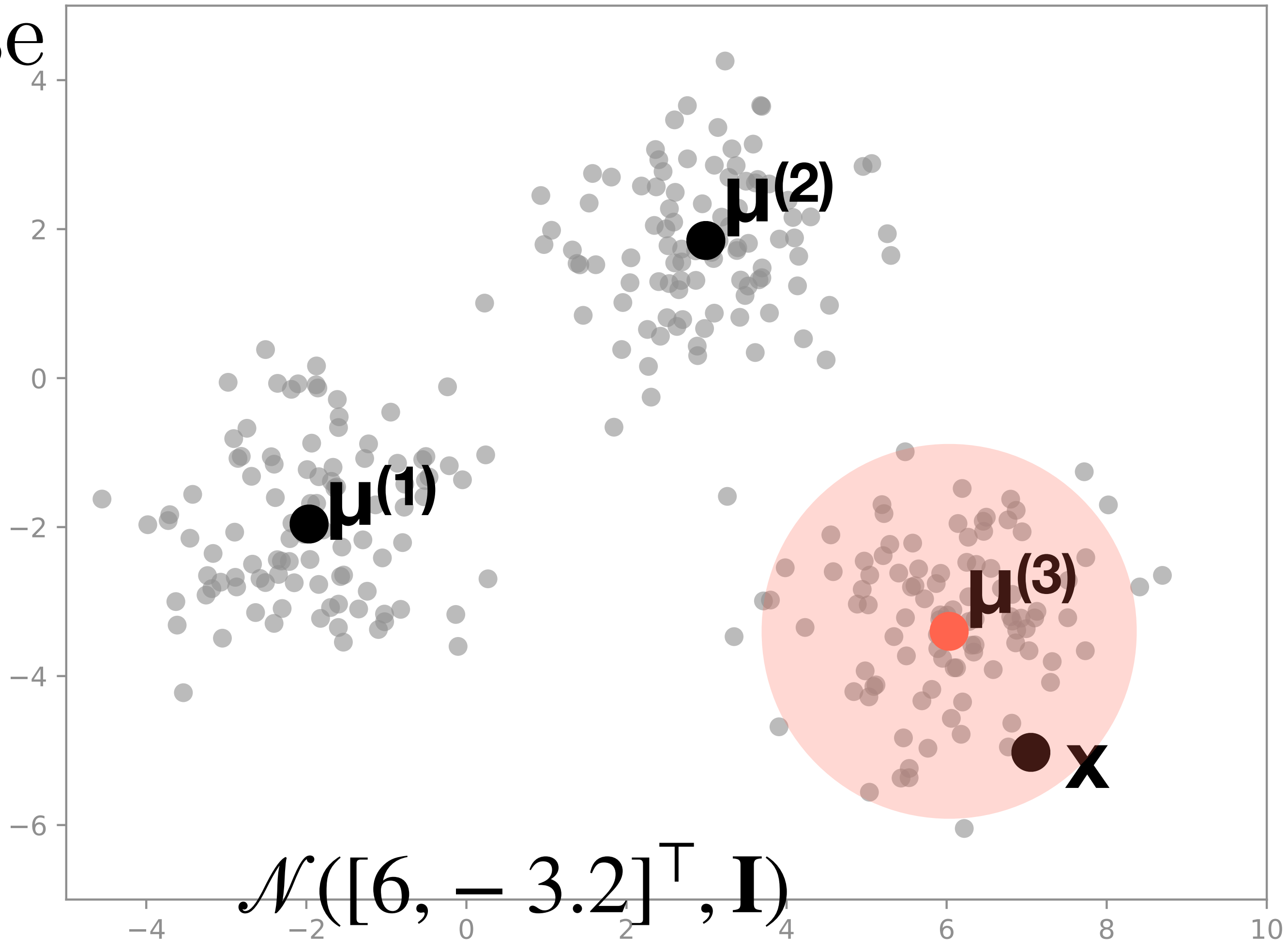
Shallow VQ VAE

- We can also construct a simple discrete "VAE":

1. *Q* maps **x** deterministically to index of closest centroid:

$$Q(z \mid \mathbf{x}) = \begin{cases} 1 & \text{if } z = \arg\min_k \|\mathbf{x} - \mu^{(k)}\|^2 \\ 0 & \text{otherwise} \end{cases}$$

2. *P* maps z to a Gaussian centered at $\boldsymbol{\mu}^{(z)}$, i.e., $P(\mathbf{x} \mid z) = \mathcal{N}(\mu^{(z)}, \mathbf{I})$

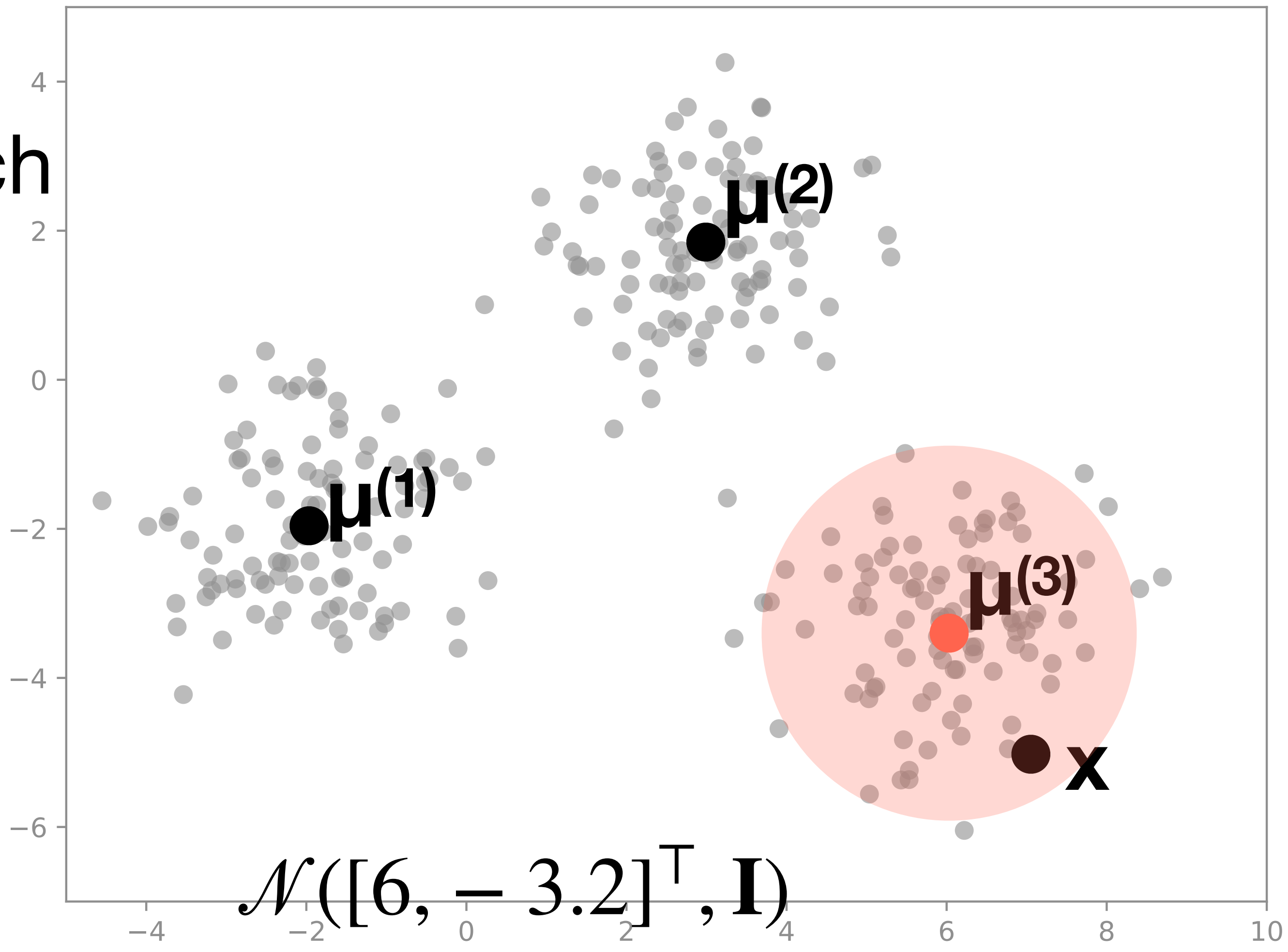


$[7,-5]^\top$ 3 $\mathcal{N}([6,-3.2]^\top, \mathbf{I})$

$$\begin{tikzcd} x \arrow[r, "Q"] & z \arrow[r, "P"] & x \end{tikzcd}$$

Shallow VQ VAE

- We can easily train this simple "VQ VAE" using MLE with a Gaussian Mixture Model (GMM).
- In particular, we can enforce $P(z) = \mathcal{U}(1, \ldots, K)$, i.e., each centroids is equally likely.

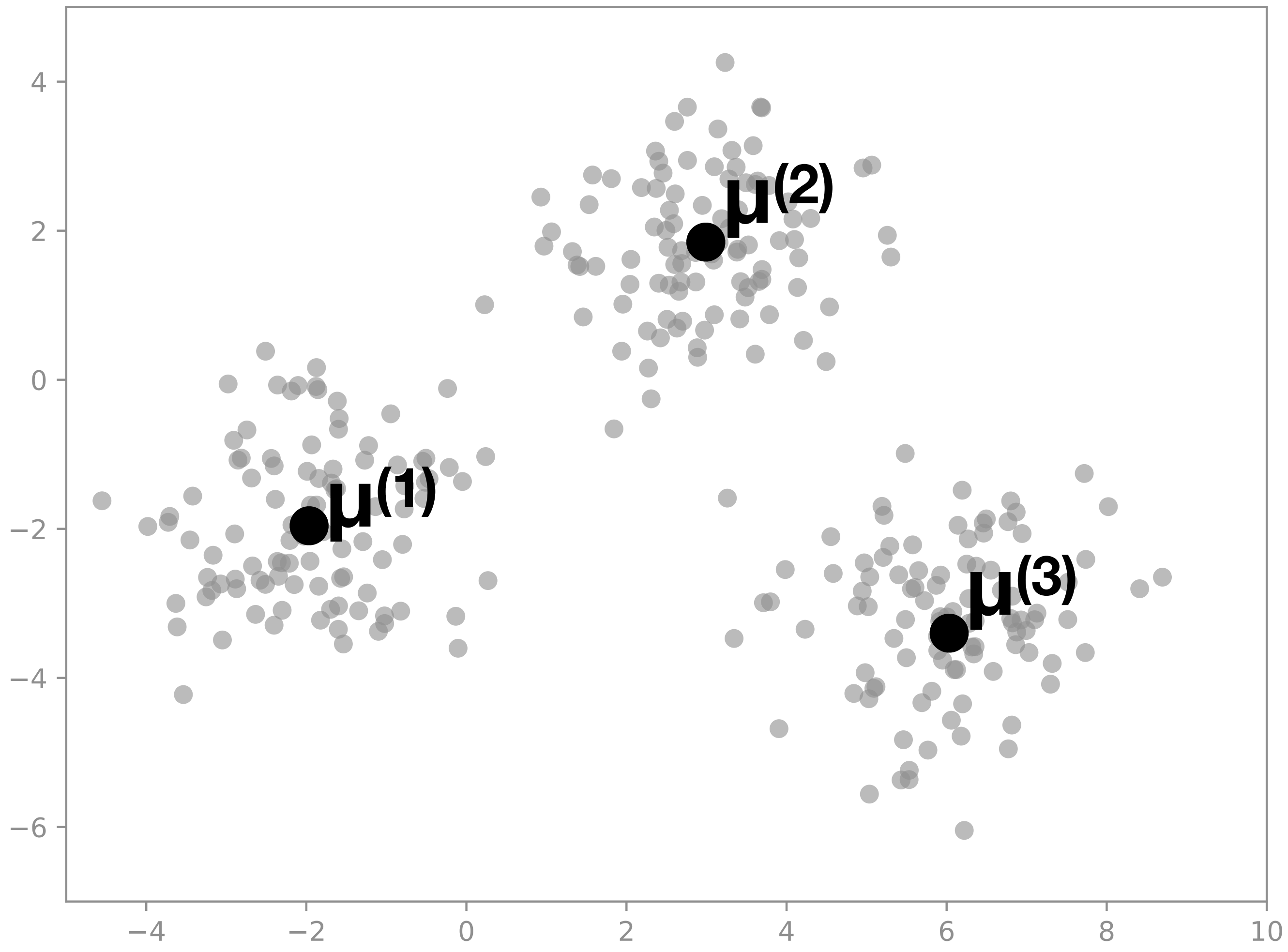


$[7,-5]^\top$ 3 $\mathcal{N}([6, -3.2]^\top, \mathbf{I})$

$$\begin{tikzcd} x \arrow[r, "Q"] & z \arrow[r, "P"] & x \end{tikzcd}$$

Generation

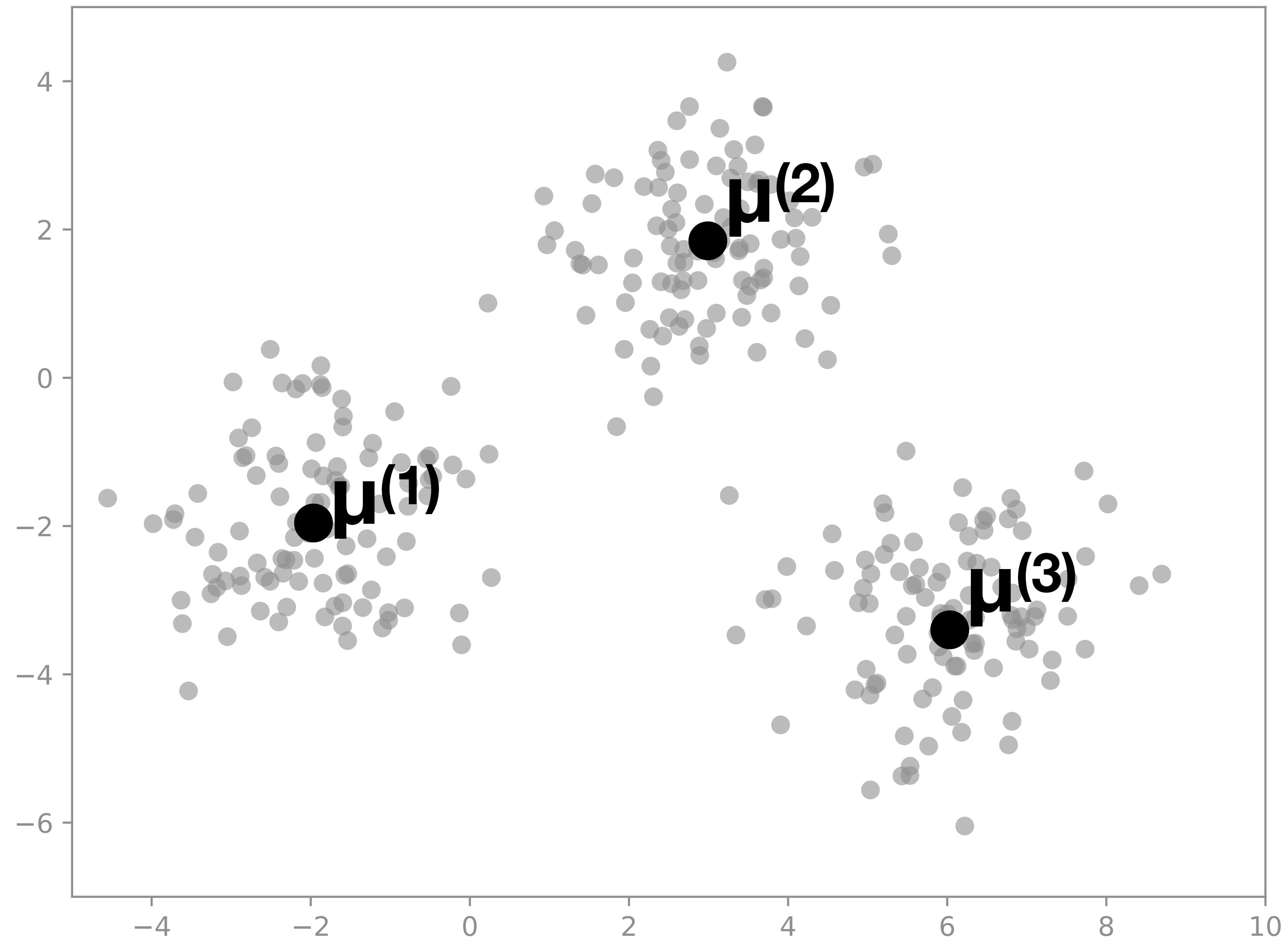
- After training, to generate new data we just:
  1. Sample $z \sim P(z) = \mathcal{U}(1,\ldots,K)$.
  2. Sample $\mathbf{x} \sim P(\mathbf{x} \mid z) = \mathcal{N}(\mu^{(z)}, \mathbf{I})$



$$\begin{tikzcd} x \arrow[r, "Q"] & z \arrow[r, "P"] & x \end{tikzcd}$$

Generation

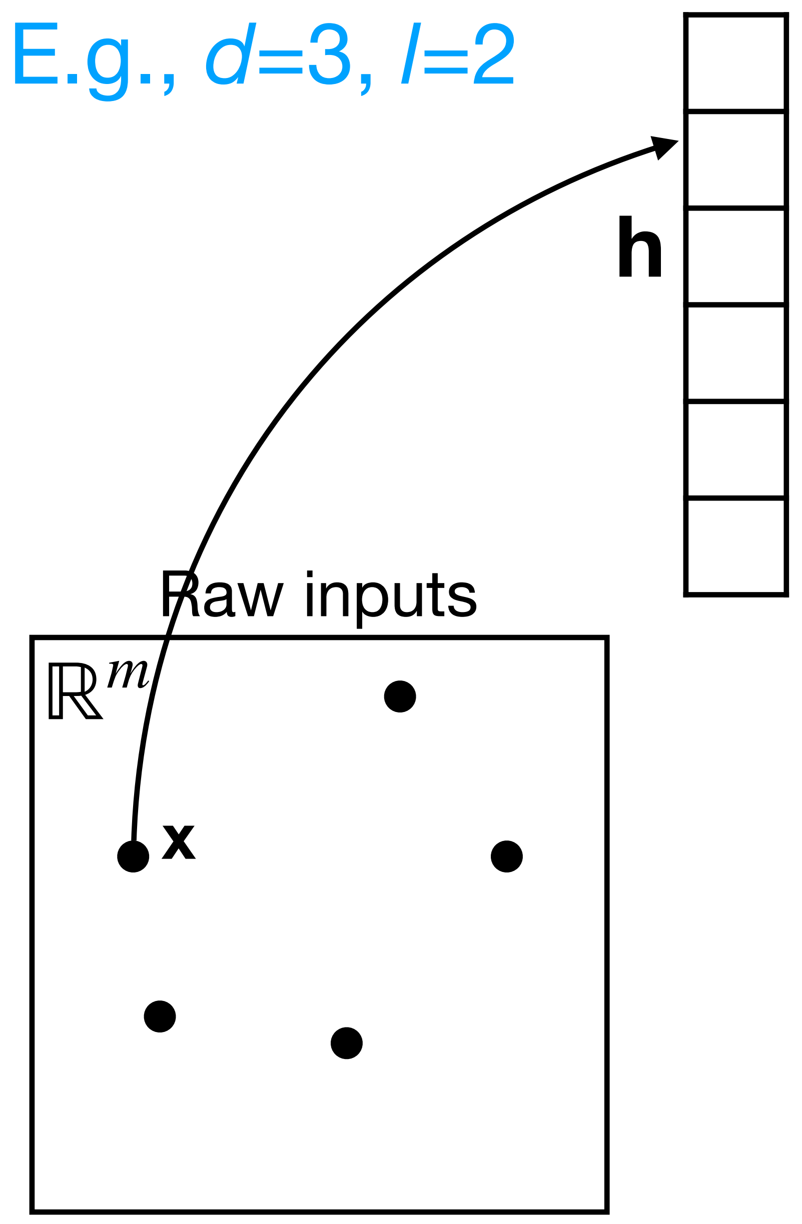
- After training, to generate new data we just:
    1. Sample $z \sim P(z) = \mathcal{U}(1,\ldots,K)$.
    2. Sample $\mathbf{x} \sim P(\mathbf{x} \mid z) = \mathcal{N}(\mu^{(z)}, \mathbf{I})$
- However, this model is very weak — we are just adding noise to centroids in the raw image space.



$$\begin{tikzcd} x \arrow[r, "Q"] & z \arrow[r, "P"] & x \end{tikzcd}$$

(Deep) VQ-VAEs

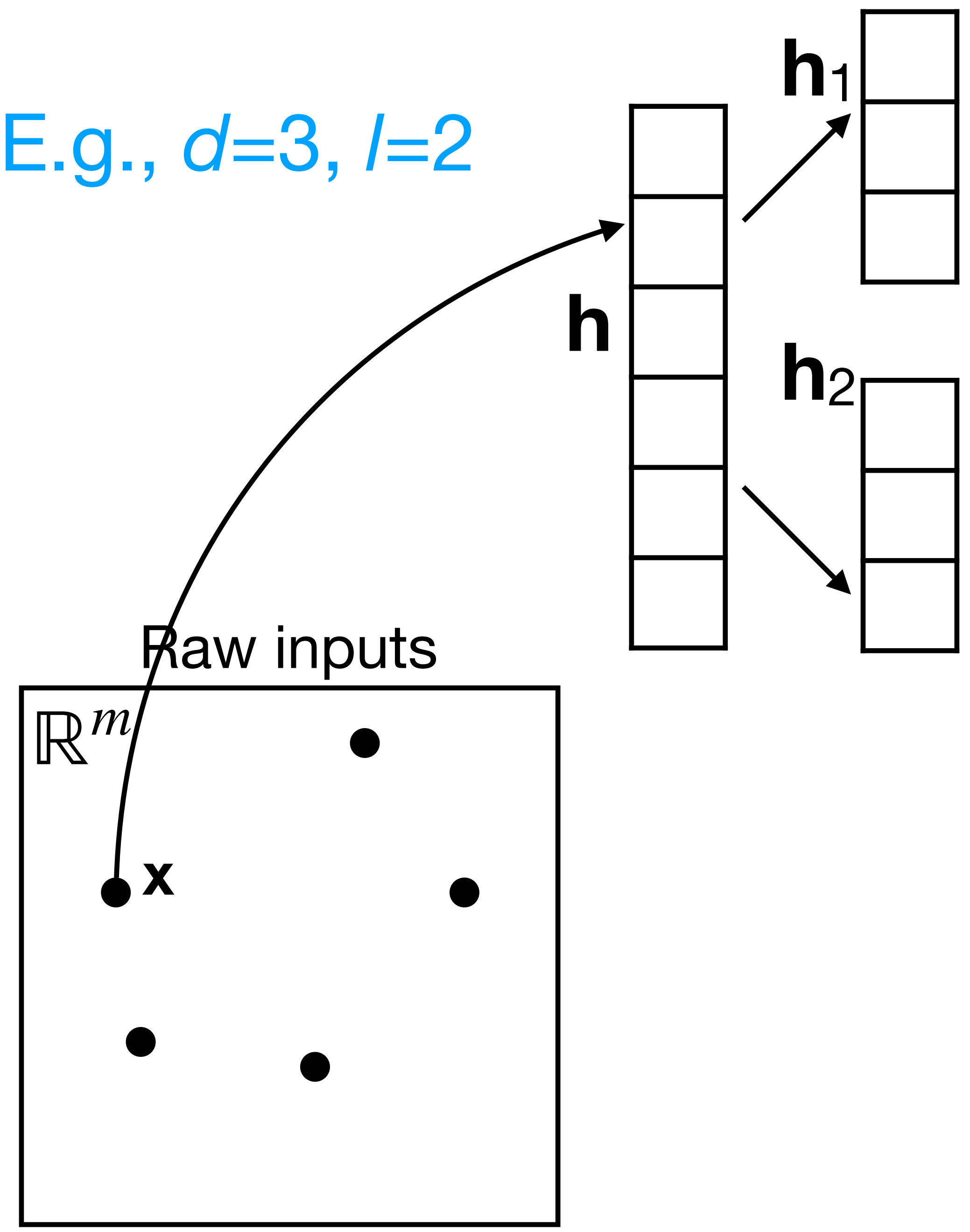
- We can generalize this idea into a deep VQ-VAE:
  1. Transform each $\mathbf{x} \in \mathbb{R}^m$ into a feature vector $\mathbf{h} \in \mathbb{R}^{l \times d}$.



(Deep) VQ-VAEs

- We can generalize this idea into a deep VQ-VAE:
  1. Transform each $\mathbf{x} \in \mathbb{R}^m$ into a feature vector $\mathbf{h} \in \mathbb{R}^{l \times d}$.
  2. Split $\mathbf{h}$ into multiple (l) vectors $\mathbf{h}_1, \ldots, \mathbf{h}_l$.

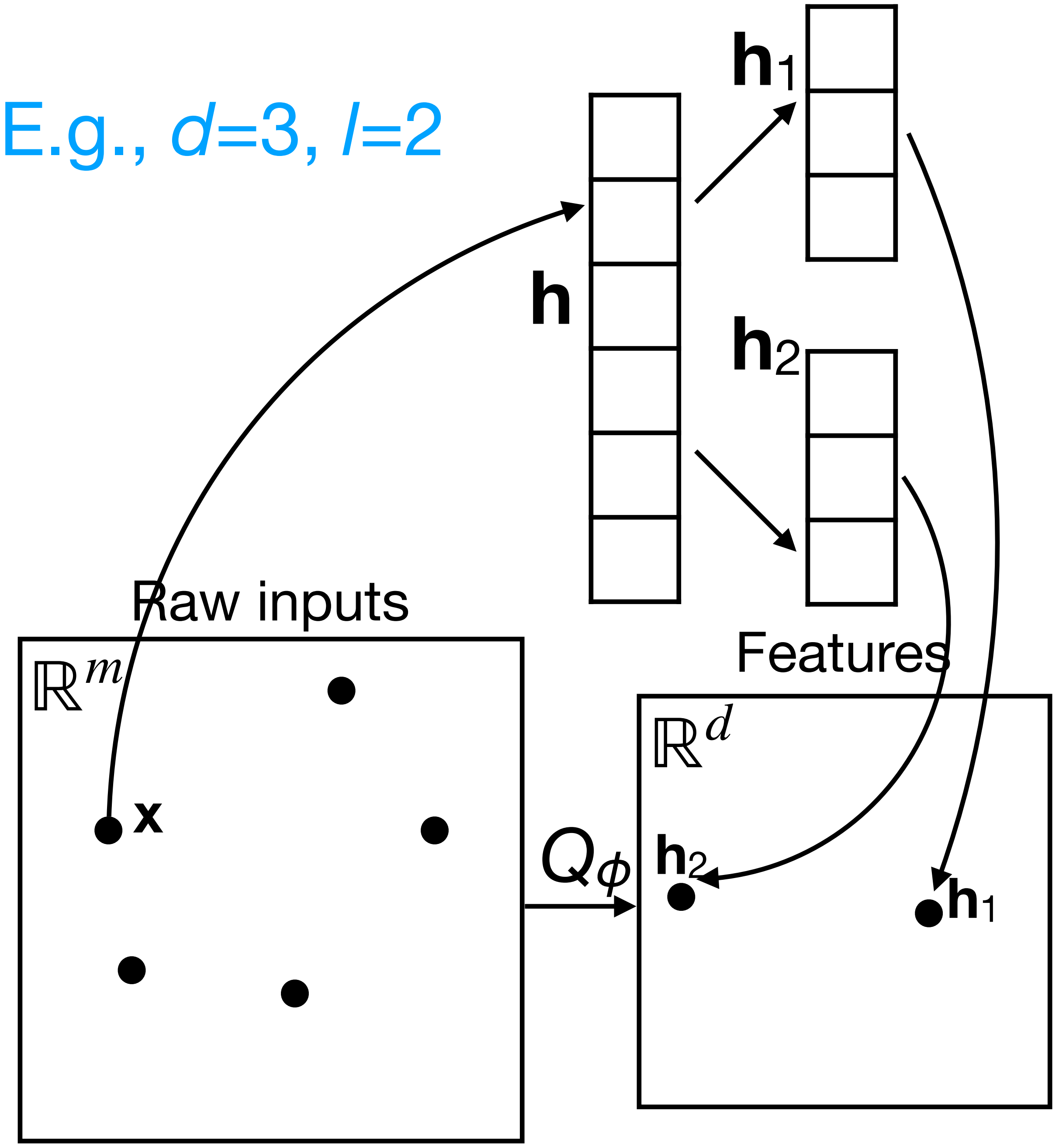
E.g., *d*=3, *l*=2



(Deep) VQ-VAEs

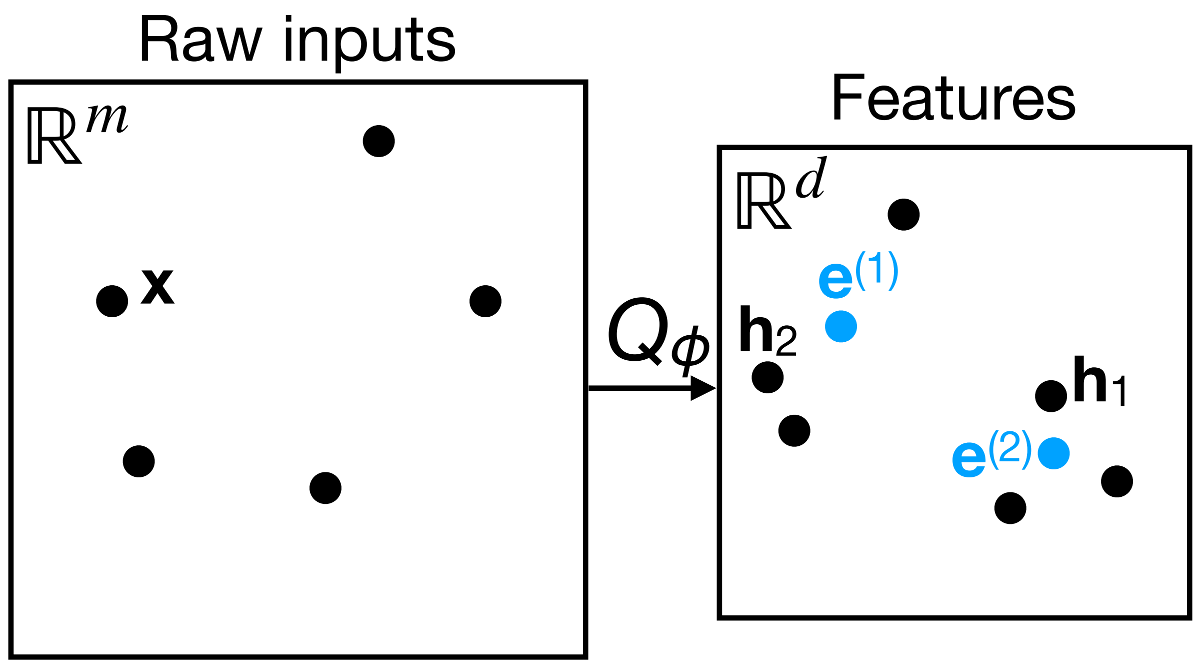
- We can generalize this idea into a deep VQ-VAE:
    1. Transform each $\mathbf{x} \in \mathbb{R}^m$ into a feature vector $\mathbf{h} \in \mathbb{R}^{l \times d}$.
    2. Split $\mathbf{h}$ into multiple (l) vectors $\mathbf{h}_1, \ldots, \mathbf{h}_l$.

E.g., *d*=3, *l*=2



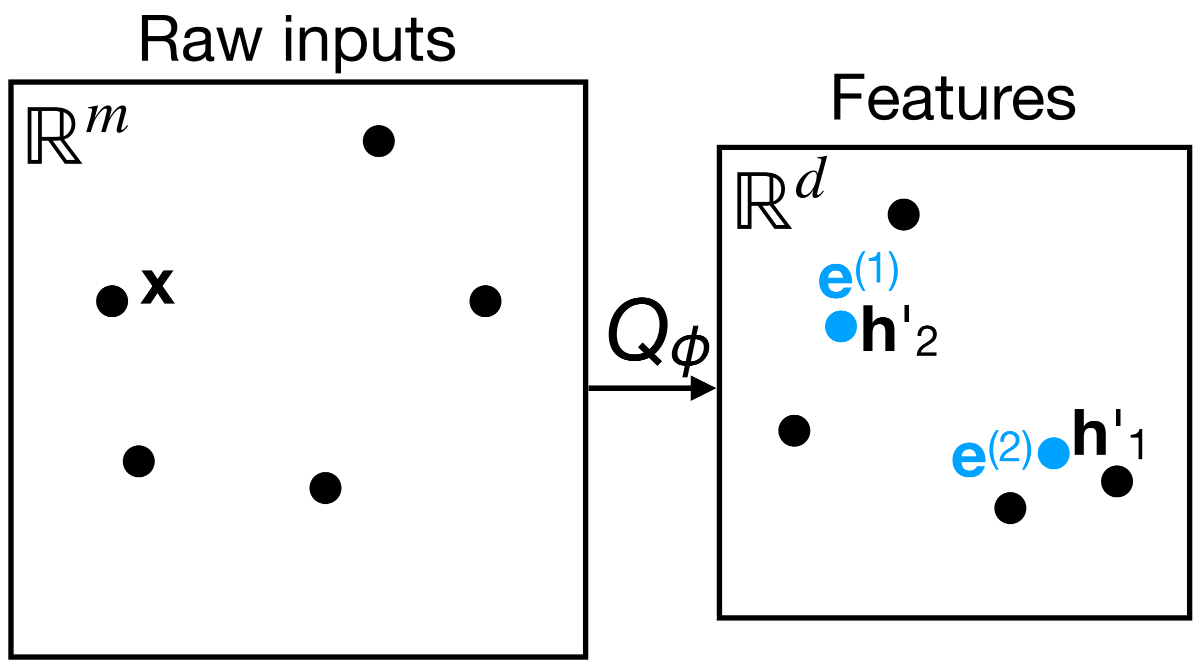
(Deep) VQ-VAEs

- We can generalize this idea into a deep VQ-VAE:
  1. Transform each $\mathbf{x} \in \mathbb{R}^m$ into a feature vector $\mathbf{h} \in \mathbb{R}^{l \times d}$.
  2. Split $\mathbf{h}$ into multiple (l) vectors $\mathbf{h}_1, \ldots, \mathbf{h}_l$.
  3. Using running estimates of K cluster centroids over $\{\mathbf{h}_j^{(i)}\}_{i,j}$,



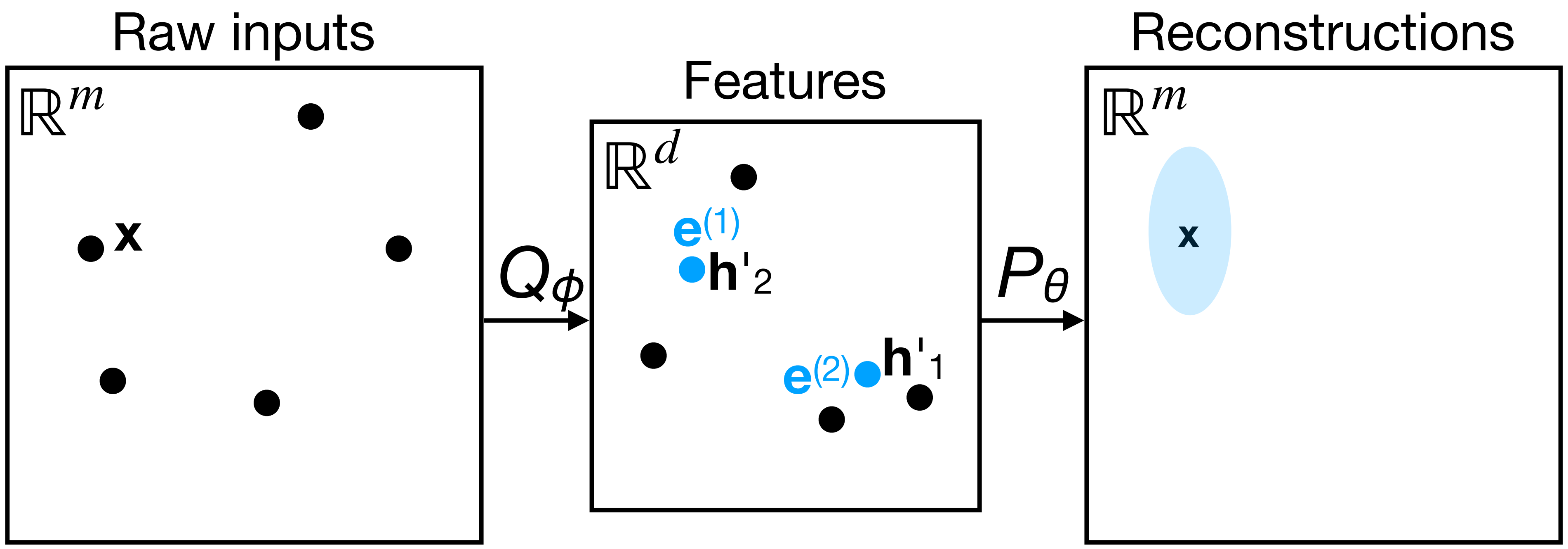
(Deep) VQ-VAEs

- We can generalize this idea into a deep VQ-VAE:
  1. Transform each $\mathbf{x} \in \mathbb{R}^m$ into a feature vector $\mathbf{h} \in \mathbb{R}^{l \times d}$.
  2. Split $\mathbf{h}$ into multiple (l) vectors $\mathbf{h}_1, \ldots, \mathbf{h}_l$.
  3. Using running estimates of K cluster centroids over $\{\mathbf{h}_j^{(i)}\}_{i,j}$, quantize each $\mathbf{h}_j$ into $\mathbf{h}_j'$ using the nearest centroid $\mathbf{e}^{(z)}$.



(Deep) VQ-VAEs

- We can generalize this idea into a deep VQ-VAE:
  1. Transform each $\mathbf{x} \in \mathbb{R}^m$ into a feature vector $\mathbf{h} \in \mathbb{R}^{l \times d}$.
  2. Split $\mathbf{h}$ into multiple (*l*) vectors $\mathbf{h}_1, \ldots, \mathbf{h}_l$.
  3. Using running estimates of *K* cluster centroids over $\{\mathbf{h}_j^{(i)}\}_{i,j}$, quantize each $\mathbf{h}_j$ into $\mathbf{h}_j'$ using the nearest centroid $\mathbf{e}^{(z)}$.
  4. Concatenate the $\mathbf{h}_1', \ldots, \mathbf{h}_l'$ into $\mathbf{h}'$, and then transform $\mathbf{h}'$ into $P(\mathbf{x} \mid \mathbf{h}') = P(\mathbf{x} \mid \{z_j\})$.



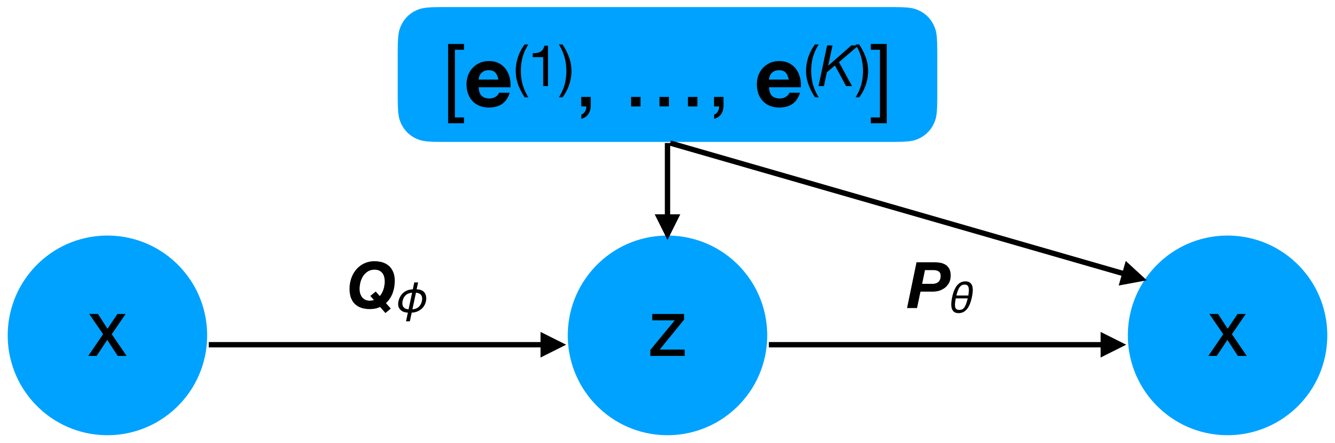
(Deep) VQ-VAEs

- The parameters that must be learned in this VQ-VAE include ϕ (encoder), θ (decoder), and $\mathbf{E}=[\mathbf{e}^{(1)}, \ldots, \mathbf{e}^{(K)}]$ (cluster centroids).

- To optimize these parameters, we will start with an MLE approach but then tweak it as necessary.

- Recall the ELBO for continuous VAEs:

$$-D_{\mathrm{KL}}(Q_\phi(z \mid \mathbf{x}) \mid P(z)) + \mathbb{E}_{Q_\phi}[\log P(\mathbf{x} \mid z)]$$



D_{KL} for VQ-VAEs

- For VQ-VAEs, we want $P(z) = \mathscr{U}(1,\ldots,K) = \dfrac{1}{K}\ \forall z$.

- We have a deterministic encoder:

$$Q(z \mid \mathbf{x}) = \begin{cases} 1 & \text{if } z = \arg\min_k \|\mathbf{x} - \mathbf{e}^{(k)}\|^2 \\ 0 & \text{otherwise} \end{cases}$$

D_{KL} for VQ-VAEs

- For VQ-VAEs, we want $P(z) = \mathscr{U}(1,\ldots,K) = \frac{1}{K}\ \forall z$.
- We have a deterministic encoder:

$$Q(z \mid \mathbf{x}) = \begin{cases} 1 & \text{if } z = \arg\min_k \|\mathbf{x} - \mathbf{e}^{(k)}\|^2 \\ 0 & \text{otherwise} \end{cases}$$

- By definition of KL divergence, we have:

$$D_{\mathrm{KL}}(Q_\phi(z \mid \mathbf{x}) \mid P(z)) = \sum_{z=1}^{K} Q(z \mid \mathbf{x}) \log \frac{Q(z \mid \mathbf{x})}{P(z)}$$

where 0 log 0 is defined to be 0.

D_{KL} for VQ-VAEs

- For VQ-VAEs, we want $P(z) = \mathscr{U}(1,\ldots,K) = \dfrac{1}{K}\ \forall z$.

- We have a deterministic encoder:

$$Q(z \mid \mathbf{x}) = \begin{cases} 1 & \text{if } z = \arg\min_k \|\mathbf{x} - \mathbf{e}^{(k)}\|^2 \\ 0 & \text{otherwise} \end{cases}$$

- By definition of KL divergence, we have:

$$\begin{aligned} D_{\mathrm{KL}}(Q_\phi(z \mid \mathbf{x}) \mid P(z)) &= \sum_{z=1}^{K} Q(z \mid \mathbf{x}) \log \frac{Q(z \mid \mathbf{x})}{P(z)} \\ &= 1 \log \frac{1}{\frac{1}{K}} + \sum_{\ldots} 0 \log \frac{0}{\frac{1}{K}} \\ &= \log K \end{aligned}$$

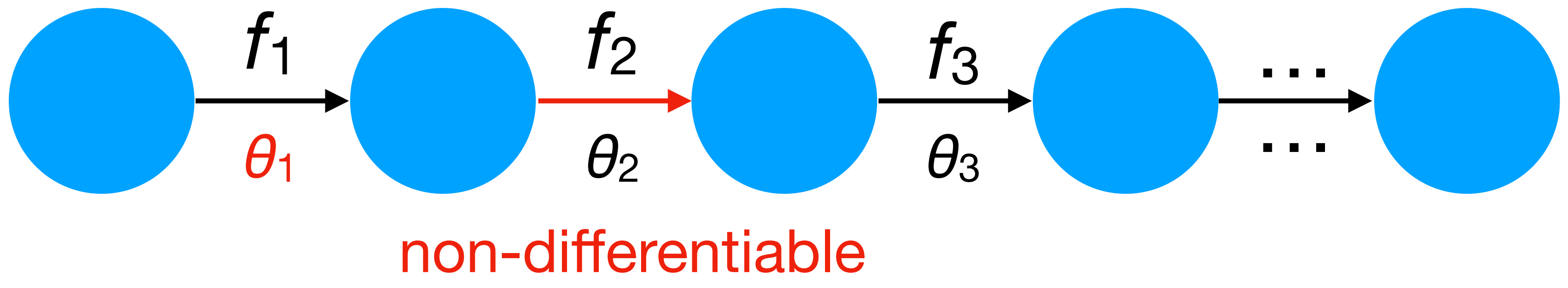
D_{KL} for VQ-VAEs

- Since log *K* does not depend on any of the VQ-VAE's parameters (*ϕ*, *θ*, **E**), it can be ignored from the ELBO.
- That just leaves:

$$\cancel{-D_{\mathrm{KL}}(Q_\phi(z \mid \mathbf{x}) \mid P(z))} + \mathbb{E}_{Q_\phi}[\log P(\mathbf{x} \mid z)]$$

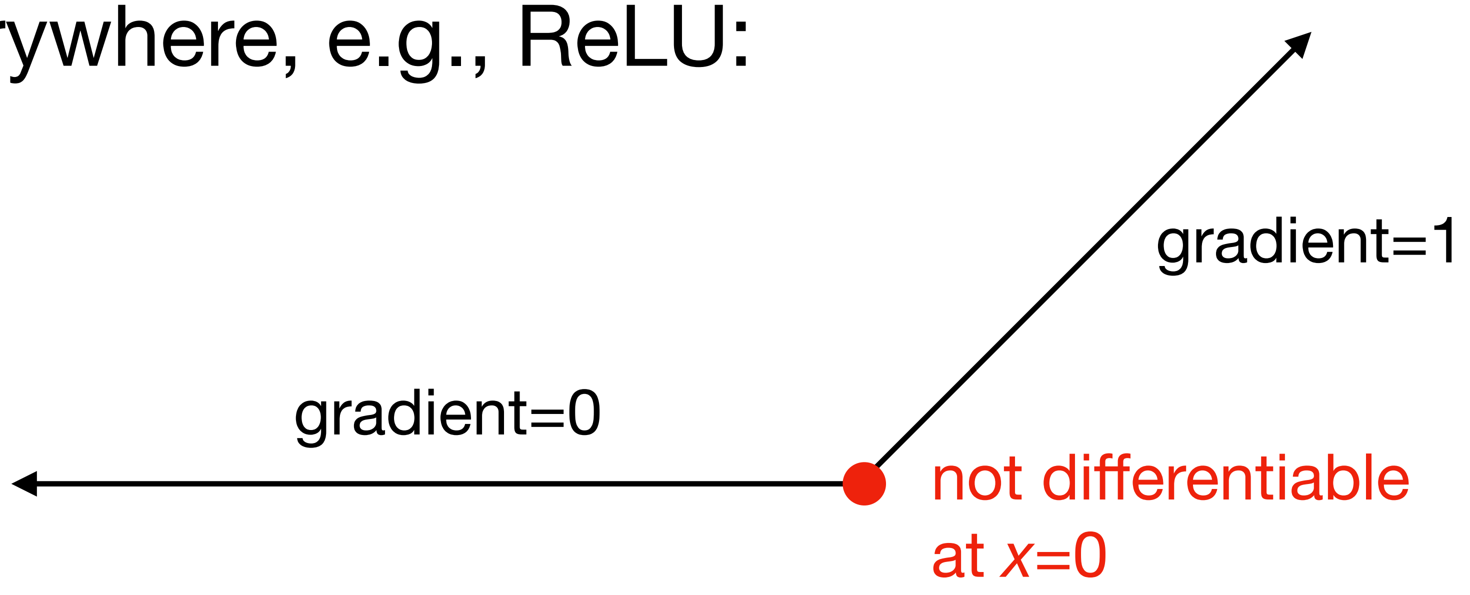
Non-differentiability

- In a computational graph (e.g., a NN), if any operation is non-differentiable, then we cannot use gradient descent to update any parameters before it.



Non-differentiability

- Contrary to classic optimization theory, modern NNs routinely use function that are not differentiable everywhere, e.g., ReLU:



- However, ReLU is differentiable *almost everywhere*, and where the derivative exists, it is “often” non-zero
=> learning can occur.

Exercise

- Consider the following functions:

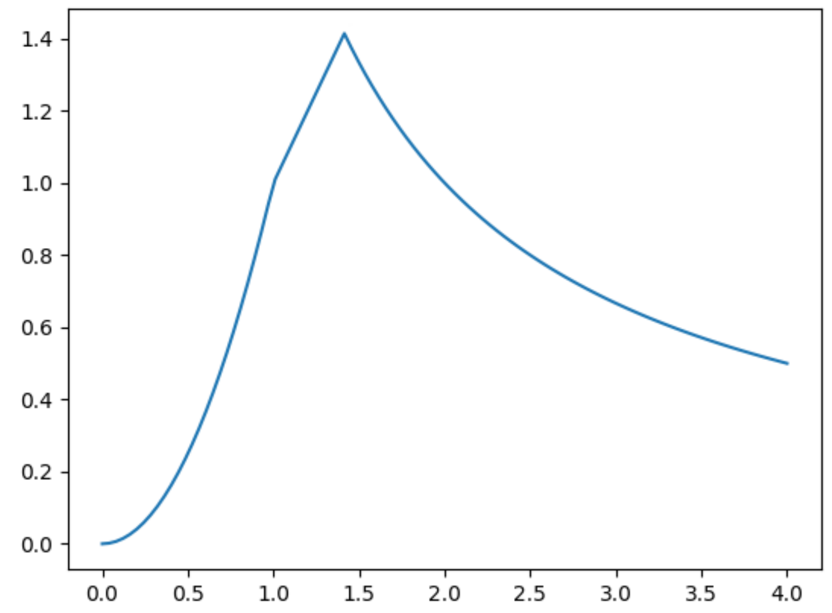
  1. $f(x) = \min\{x,\ x^2,\ 2/x\}$

  2. $g(x) = \arg\min\{x,\ x^2,\ 2/x\}$

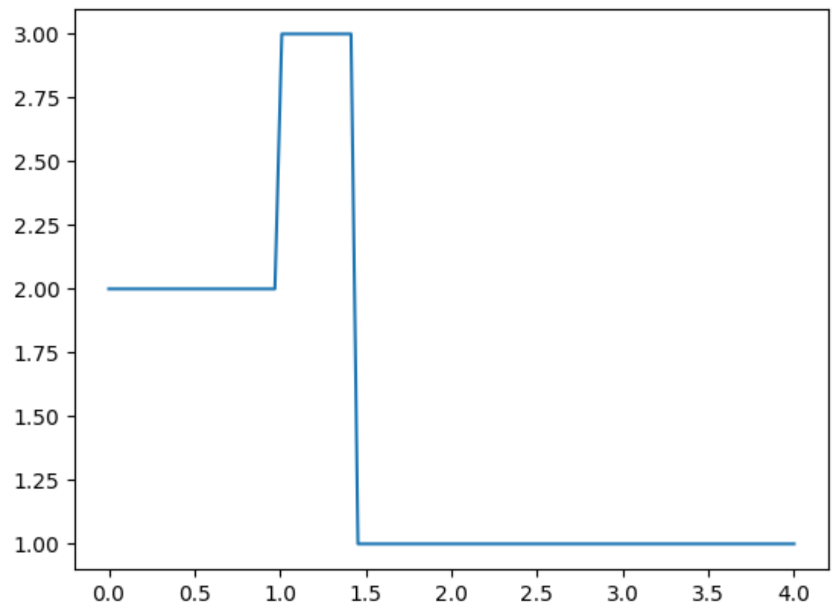
- Could you use (1), (2), or both (1)&(2) within a NN, or would they “break” backpropagation?

Solution

- *f*(*x*) is differentiable almost everywhere and has non-zero gradient where it is defined => we can learn!



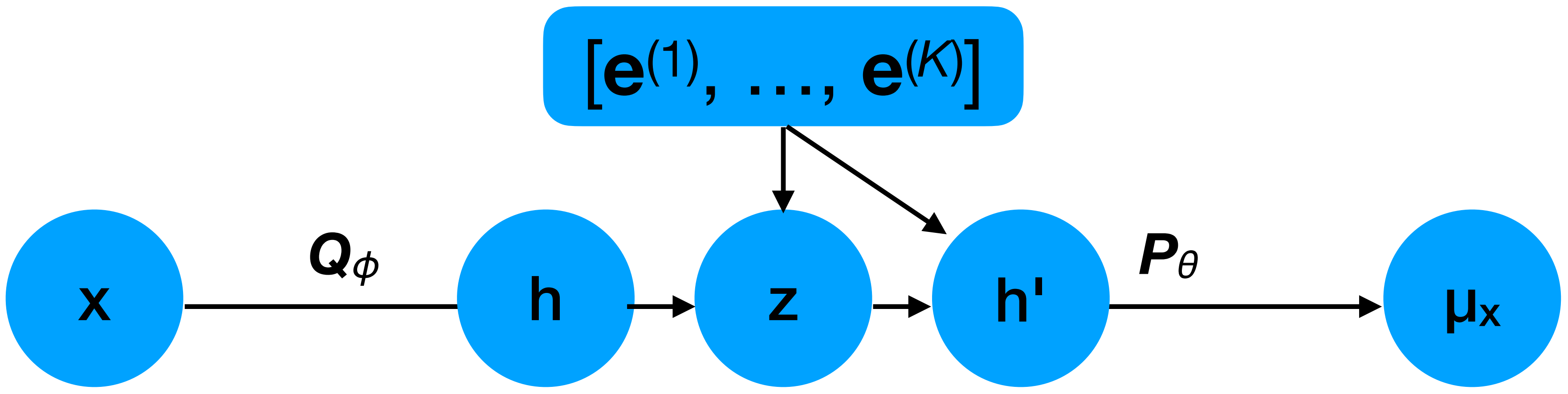
- *g*(*x*) has a gradient that is either 0 or not defined => bad!



D_{KL} for VQ-VAEs

$$-D_{\mathrm{KL}}(Q_\phi(z \mid \mathbf{x}) \mid P(z)) + \mathbb{E}_{Q_\phi}[\log P(\mathbf{x} \mid z)]$$

- Like with a VAE, we could try to estimate the second term in the ELBO by sampling *K*=1 samples, i.e.:
    1. Sample *z* from *Q*(*z* | **x**).
    2. Compute log *P*(**x** | *z*), e.g., $-||\mathbf{x} - \mu_{\mathbf{x}}||^2$



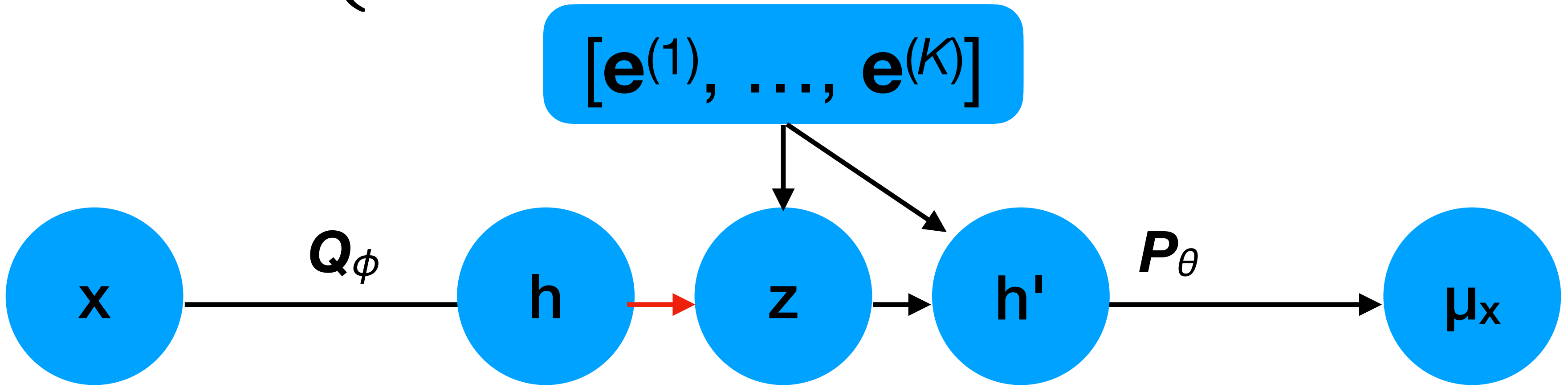
D_{KL} for VQ-VAEs

$$-D_{\mathrm{KL}}(Q_\phi(z \mid \mathbf{x}) \mid P(z)) + \mathbb{E}_{Q_\phi}[\log P(\mathbf{x} \mid z)]$$

- Like with a VAE, we could try to estimate the second term in the ELBO by sampling *K*=1 samples, i.e.:
    1. Sample z from $Q(z \mid \mathbf{x})$.
    2. Compute $\log P(\mathbf{x} \mid z)$, e.g., $-\|\mathbf{x} - \mu_{\mathbf{x}}\|^2$

- Like with continuous VAE, sampling is non-differentiable:

$$Q(z \mid \mathbf{x}) = \begin{cases} 1 & \text{if } z = \arg\min_k \|\mathbf{x} - \mathbf{e}^{(k)}\|^2 \\ 0 & \text{otherwise} \end{cases}$$



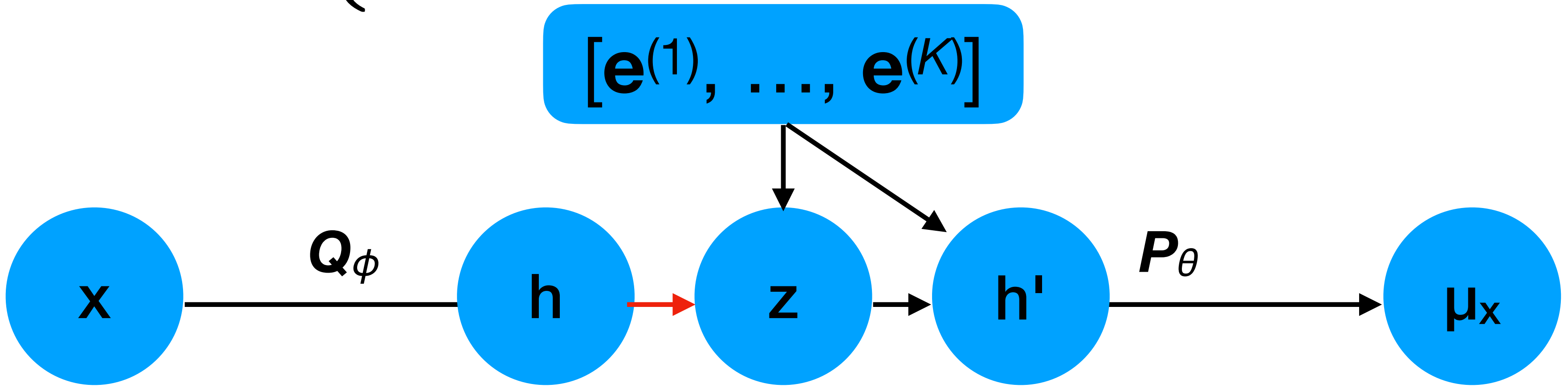
D_{KL} for VQ-VAEs

$$-D_{\mathrm{KL}}(Q_\phi(z \mid \mathbf{x}) \mid P(z)) + \mathbb{E}_{Q_\phi}[\log P(\mathbf{x} \mid z)]$$

- Like with a VAE, we could try to estimate the second term in the ELBO by sampling *K*=1 samples, i.e.:
  1. Sample *z* from *Q*(*z* | **x**).
  2. Compute log *P*(**x** | *z*), e.g., $-\|\mathbf{x} - \mu_{\mathbf{x}}\|^2$

- Unlike continuous VAE, there is no reparam. trick:

$$Q(z \mid \mathbf{x}) = \begin{cases} 1 & \text{if } z = \arg\min_k \|\mathbf{x} - \mathbf{e}^{(k)}\|^2 \\ 0 & \text{otherwise} \end{cases}$$

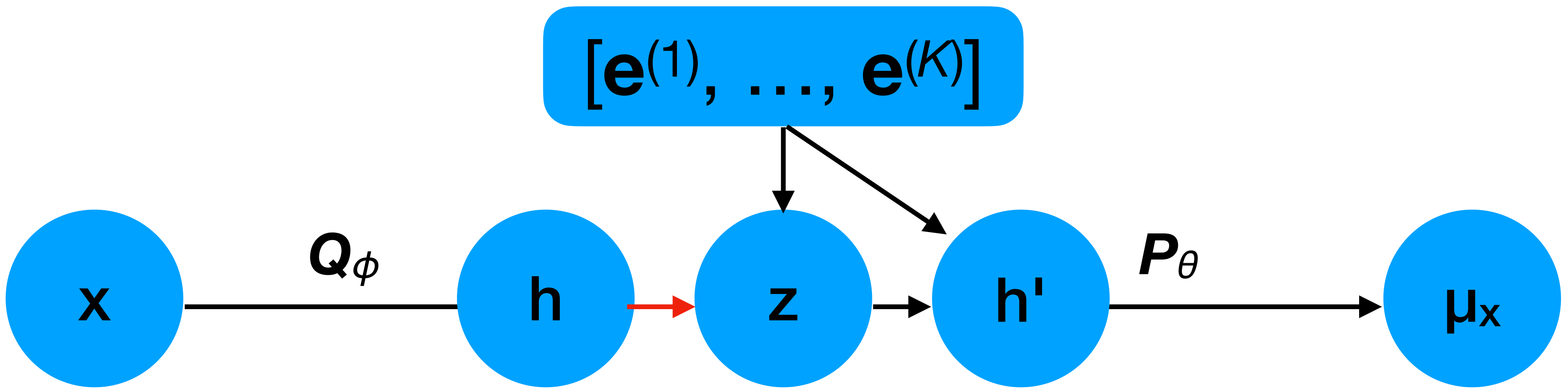


D_{KL} for VQ-VAEs

$$-D_{\mathrm{KL}}(Q_\phi(z \mid \mathbf{x}) \mid P(z)) + \mathbb{E}_{Q_\phi}[\log P(\mathbf{x} \mid z)]$$

- The problem is that the path through the graph to ϕ are blocked due to non-differentiability:

$$\frac{\partial \log P(\mathbf{x} \mid z)}{\partial \mathbf{h}'} \frac{\partial \mathbf{h}'}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \phi}$$

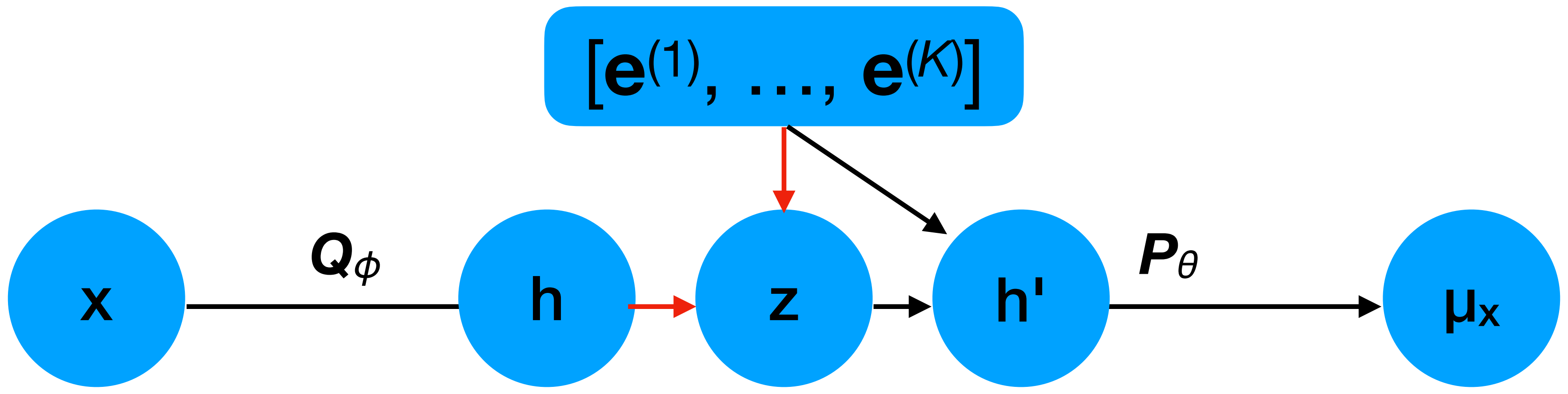


D_{KL} for VQ-VAEs

- For this reason, we have to “give up” on a first-principles approach to maximizing $\mathbb{E}_{Q_\phi}[P(\mathbf{x} \mid z)]$ for VQ-VAEs.
- Instead, we hand-design a loss function with the goals:

  1. Encourage each $\mathbf{h}^{(i)'}$ to give a high $\log P(\mathbf{x} \mid z)$.

$$f_{\text{loss}}(\phi, \theta, \mathbf{E}) = \|\mathbf{x} - \mu_{\mathbf{x}}\|^2$$



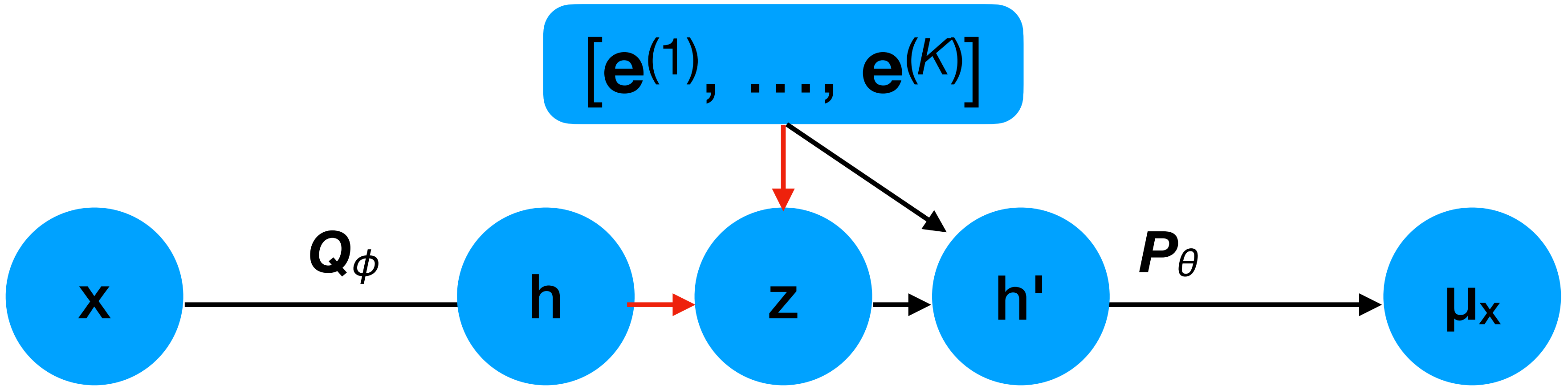
D_{KL} for VQ-VAEs

- For this reason, we have to "give up" on a first-principles approach to maximizing $\mathbb{E}_{Q_\phi}[P(\mathbf{x} \mid z)]$ for VQ-VAEs.

- Instead, we hand-design a loss function with the goals:

  1. Encourage each $\mathbf{h}^{(i)'}$ to give a high $\log P(\mathbf{x} \,|\, z)$.
  2. Encourage the $\mathbf{e}^{(1)}, \ldots, \mathbf{e}^{(K)}$ to be the centroids of *K* clusters among the $\{\mathbf{h}_j^{(i)}\}_{i,j}$.

$$f_{\mathrm{loss}}(\phi, \theta, \mathbf{E}) = \|\mathbf{x} - \mu_{\mathbf{x}}\|^2 + \|\mathbf{h} - \mathbf{e}^{(z)}\|^2$$



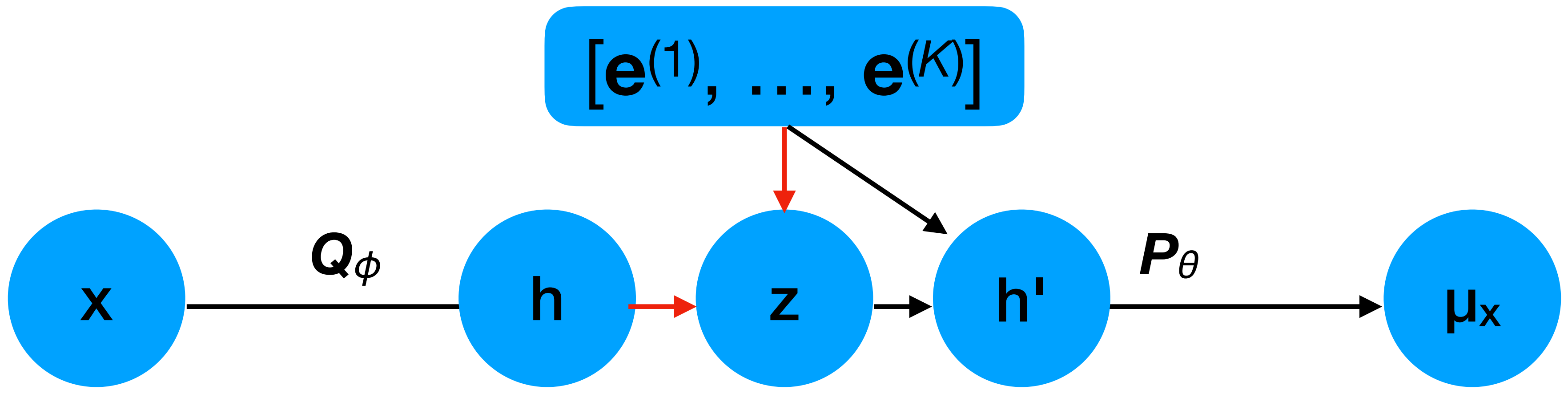
D_{KL} for VQ-VAEs

- For this reason, we have to "give up" on a first-principles approach to maximizing $\mathbb{E}_{Q_\phi}[P(\mathbf{x} \mid z)]$ for VQ-VAEs.

- Instead, we hand-design a loss function with the goals:

  1. Encourage each $\mathbf{h}^{(i)'}$ to give a high $\log P(\mathbf{x} \,|\, z)$.
  2. Encourage the $\mathbf{e}^{(1)}, \ldots, \mathbf{e}^{(K)}$ to be the centroids of *K* clusters among the $\{\mathbf{h}_j^{(i)}\}_{i,j}$.
  3. Encourage each $\mathbf{h}^{(i)}$ to be "close" to its nearest centroid $\mathbf{e}^{(z)}$.

$$f_{\mathrm{loss}}(\phi, \theta, \mathbf{E}) = \|\mathbf{x} - \mu_{\mathbf{x}}\|^2 + \|\mathbf{h} - \mathbf{e}^{(z)}\|^2$$



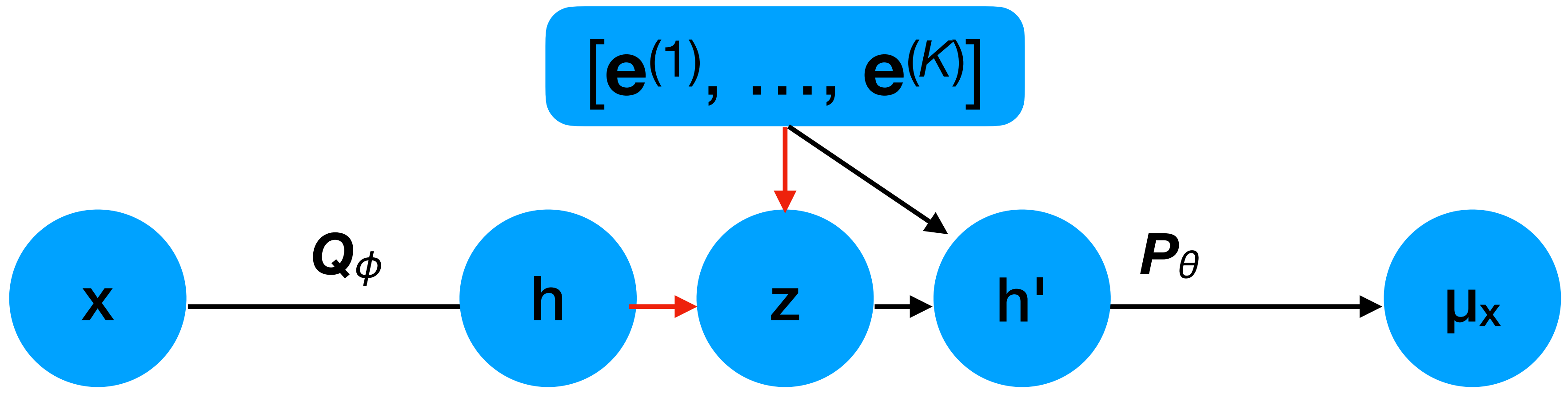
D_{KL} for VQ-VAEs

- But what about the non-differentiable map from **h** to **h**'?

$$\frac{\partial \log P(\mathbf{x} \mid z)}{\partial \mathbf{h}'} \frac{\partial \mathbf{h}'}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \phi}$$

How a change in **h**' affects reconstruction error

$$f_{\text{loss}}(\phi, \theta, \mathbf{E}) = \|\mathbf{x} - \mu_{\mathbf{x}}\|^2 + \|\mathbf{h} - \mathbf{e}^{(z)}\|^2$$



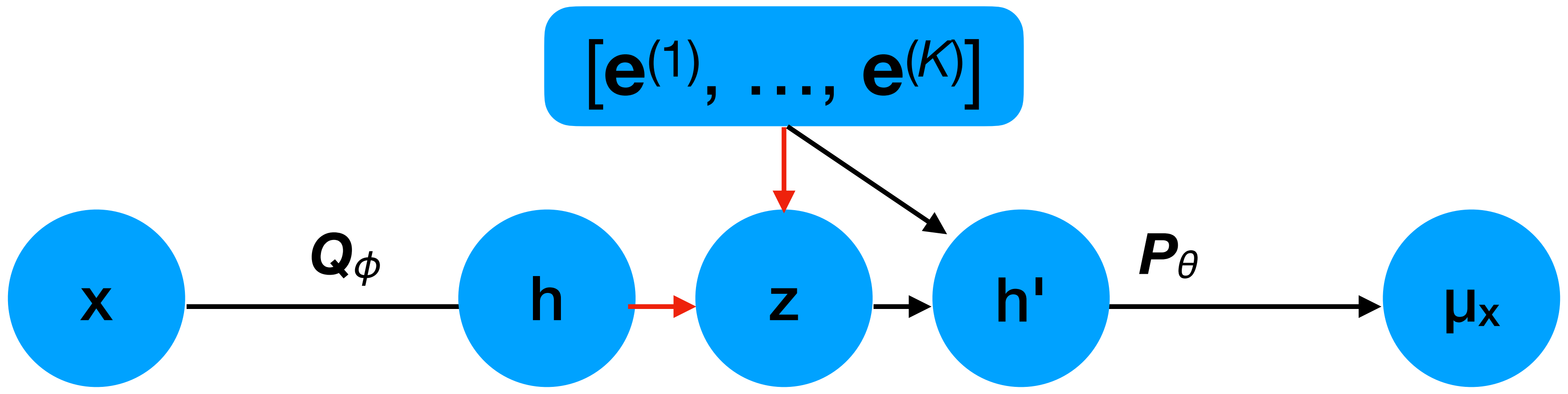
D_{KL} for VQ-VAEs

- But what about the non-differentiable map from **h** to **h**'?

$$\frac{\partial \log P(\mathbf{x} \mid z)}{\partial \mathbf{h}'} \frac{\partial \mathbf{h}'}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \phi}$$

How a change in **h** affects **h**'

$$f_{\text{loss}}(\phi, \theta, \mathbf{E}) = \|\mathbf{x} - \mu_{\mathbf{x}}\|^2 + \|\mathbf{h} - \mathbf{e}^{(z)}\|^2$$



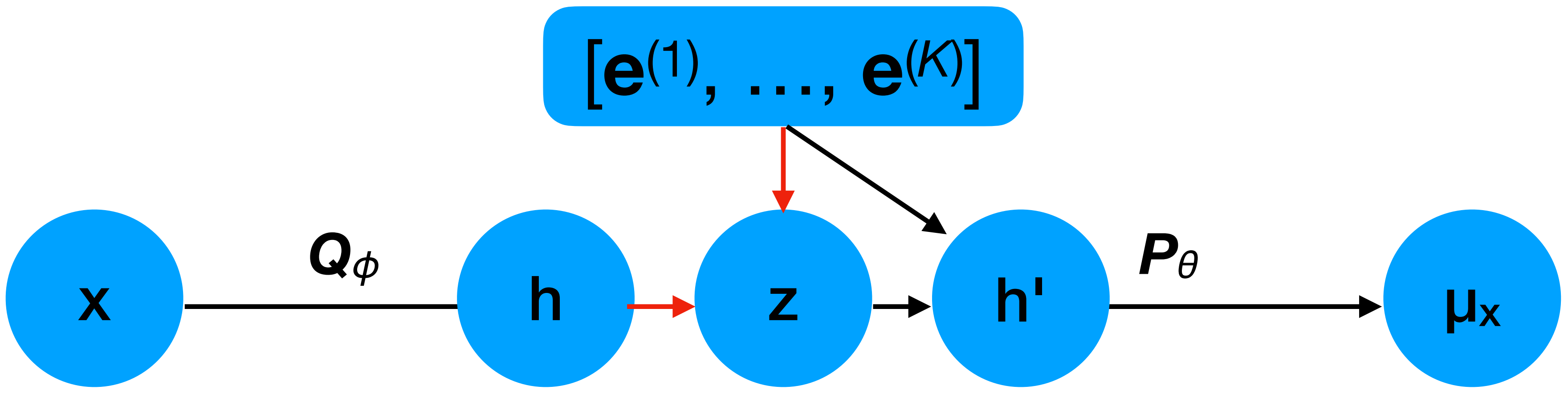
D_{KL} for VQ-VAEs

- But what about the non-differentiable map from **h** to **h**'?

$$\frac{\partial \log P(\mathbf{x} \mid z)}{\partial \mathbf{h}'} \frac{\partial \mathbf{h}'}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \phi}$$

How the parameters of *Q* affect **h**

$$f_{\text{loss}}(\phi, \theta, \mathbf{E}) = \|\mathbf{x} - \mu_{\mathbf{x}}\|^2 + \|\mathbf{h} - \mathbf{e}^{(z)}\|^2$$



D_{KL} for VQ-VAEs

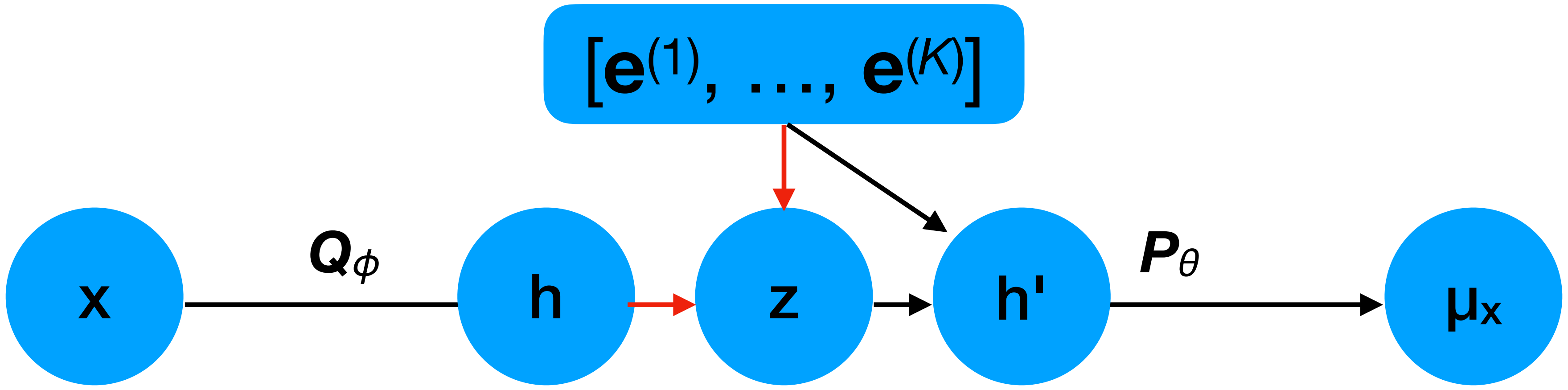
- But what about the non-differentiable map from **h** to **h**'?

$$\frac{\partial \log P(\mathbf{x} \mid z)}{\partial \mathbf{h}'}\mathbf{I}\frac{\partial \mathbf{h}}{\partial \phi}$$

- **Straight-through gradient estimator**: we just ignore it! I.e., we set it to **I**.

- Why is this reasonable? My interpretation: a small change in **h** *does* pull **h**' toward h (though scaled by inverse cluster size).

$$f_{\text{loss}}(\phi, \theta, \mathbf{E}) = \|\mathbf{x} - \mu_{\mathbf{x}}\|^2 + \|\mathbf{h} - \mathbf{e}^{(z)}\|^2$$



[e(1), …, e(K)]
x
Qϕ
h
z
h'
Pθ
μx