Autoencoders

Deep Learning & Applications

Learning Problems

Supervised Learning:

- Learn model using dataset with **data-label pairs** $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$
- Examples: Classification, regression, structured prediction

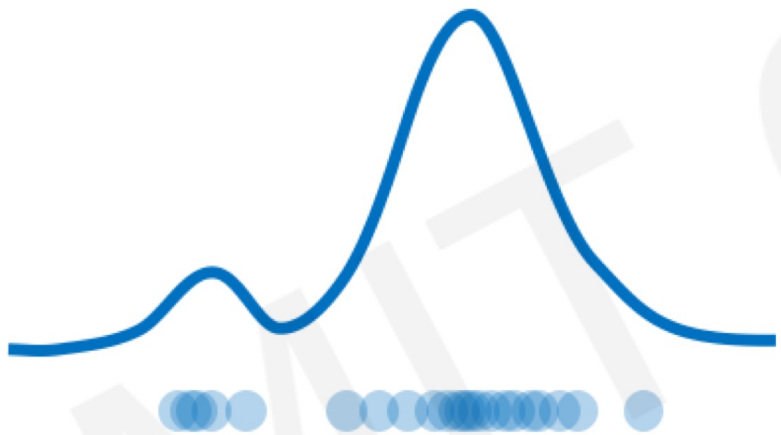
Unsupervised Learning:

- Learn model using dataset **without labels** $\{\mathbf{x}_i\}_{i=1}^N$
- Examples: Clustering, dimensionality reduction, generative models

Generative modeling

Goal: Take as input training samples from some distribution and learn a model that represents that distribution

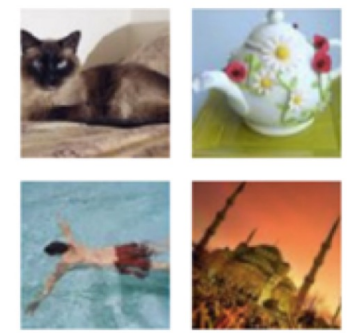
Density Estimation



Sample Generation

Input samples

Training data ~ $P_{data}(x)$



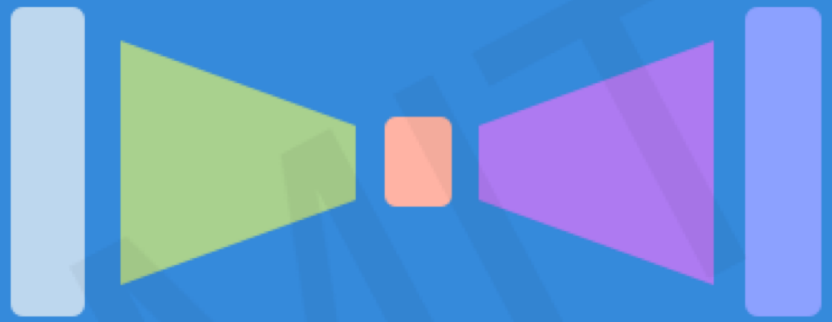
Generated samples

Generated ~ $P_{model}(x)$

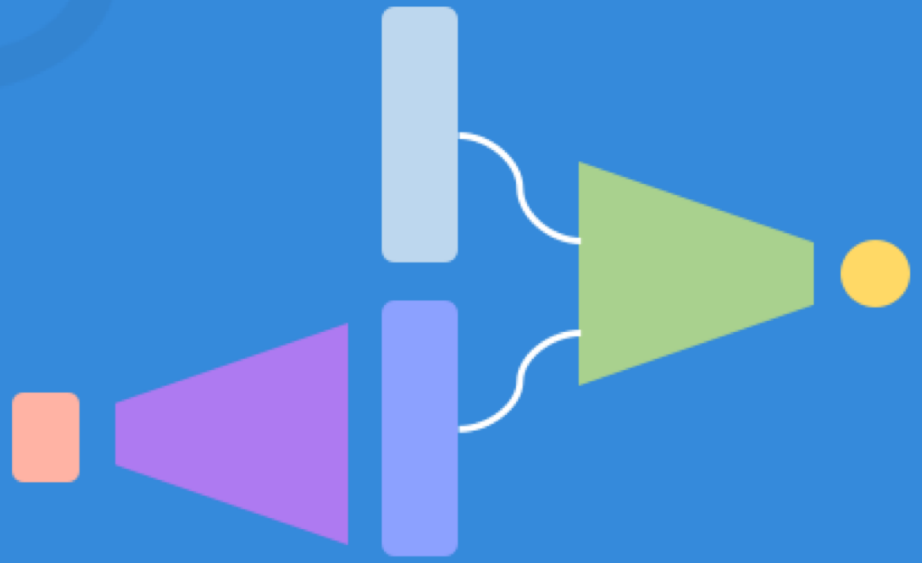
How can we learn $P_{model}(x)$ similar to $P_{data}(x)$?

Latent variable models

Autoencoders and Variational Autoencoders (VAEs)



Generative Adversarial Networks (GANs)



What is a latent variable?

Myth of the Cave

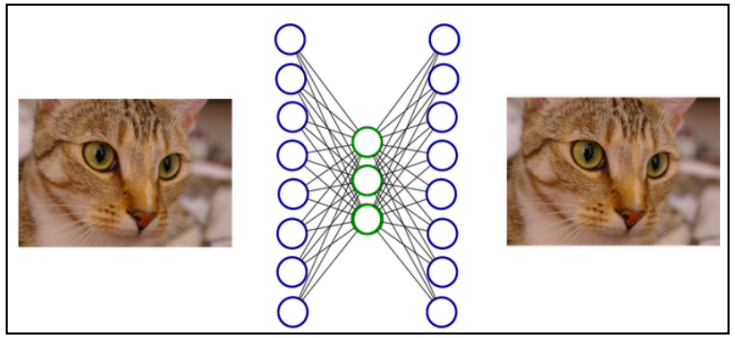
What is a latent variable?

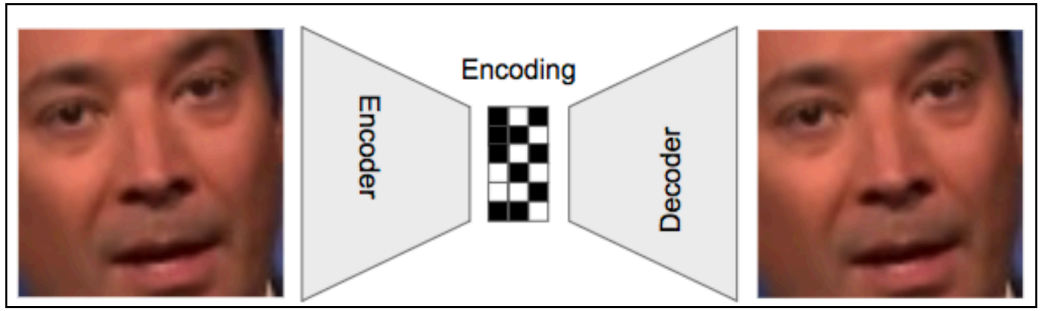
Can we learn the **true explanatory factors**, e.g. latent variables, from only observed data?

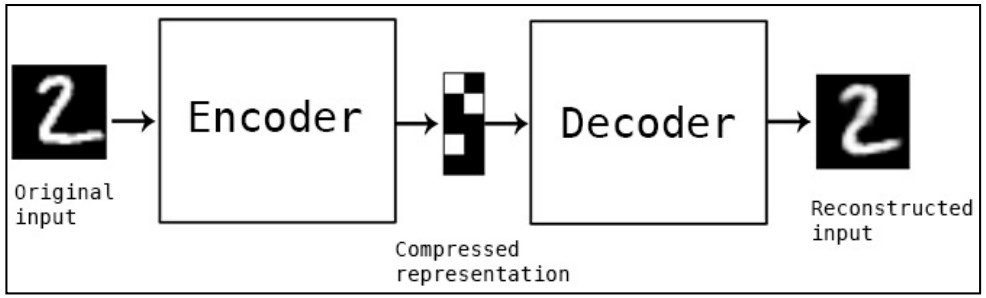
What is an Autoencoder?

A neural network trained using unsupervised learning

- Trained to copy its input to its output



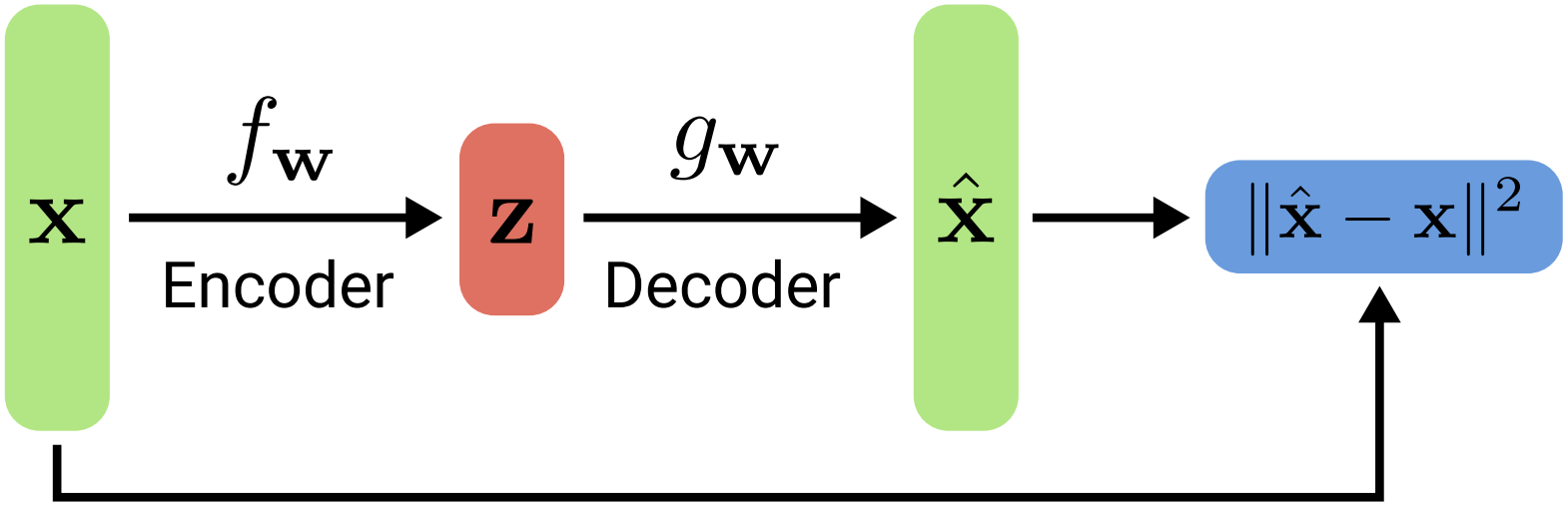




What is an Autoencoder?

Autoencoders comprise an **encoder** $f_{\mathbf{w}}$ as well as a **decoder** $g_{\mathbf{w}}$:

$$f_{\mathbf{w}} : \mathbf{x} \mapsto \mathbf{z} \qquad g_{\mathbf{w}} : \mathbf{z} \mapsto \hat{\mathbf{x}}$$



- Models of this type are called **autoencoders** as they predict their input as output

Auto encoders differ from General Data Compression

- Autoencoders are data-specific
  - i.e., only able to compress data similar to what they have been trained on
- This is different from, say, MP3 or JPEG compression algorithm
  - Which make general assumptions about "sound/images", but not about specific types of sounds/images
  - Autoencoder for pictures of cats would do poorly in compressing pictures of trees
    - Because features it would learn would be cat-specific
- Autoencoders are lossy
  - which means that the decompressed outputs will be degraded compared to the original inputs (similar to MP3 or JPEG compression).
  - This differs from lossless arithmetic compression
- Autoencoders are learnt

What does an Autoencoder Learn?

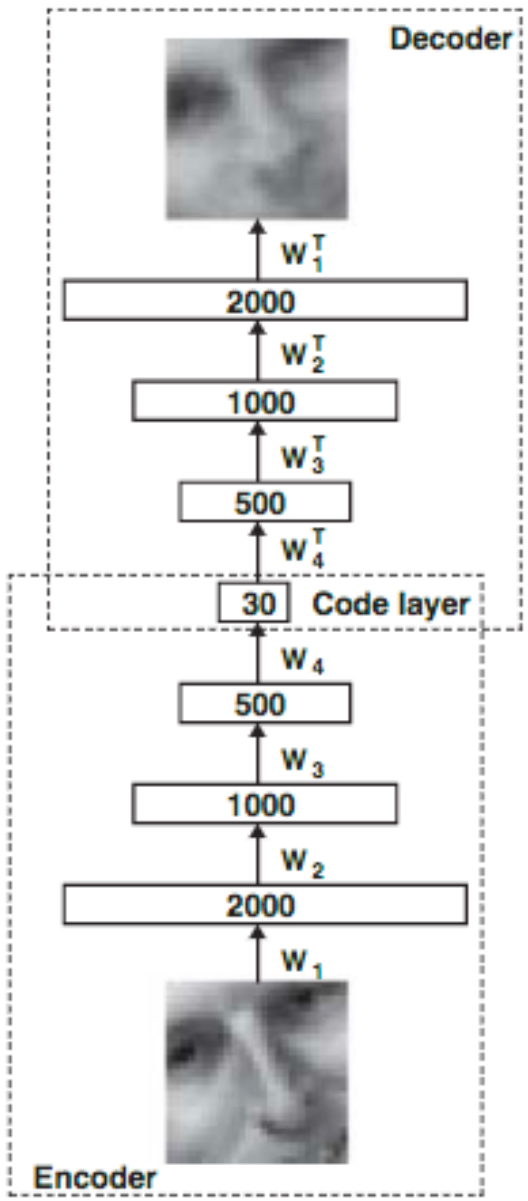
- Learning $g(f(\boldsymbol{x}))=\boldsymbol{x}$ everywhere is not useful
- Autoencoders are designed to be unable to copy perfectly
  - Restricted to copy only approximately
- Autoencoders learn useful properties of the data
  - Being forced to prioritize which aspects of input should be copied

- Maps an input $\boldsymbol{x}$ to an output $\boldsymbol{r}$ (called reconstruction) through an internal representation code $\boldsymbol{h}$
  - It has a hidden layer $\boldsymbol{h}$ that describes a code used to represent the input
- The network has two parts
  - The encoder function $\boldsymbol{h}=f(\boldsymbol{x})$
  - A decoder that produces a reconstruction $\boldsymbol{r}=g(\boldsymbol{h})$

An autoencoder architecture

Decoder g

Encoder f



Undercomplete autoencoder

- Constrain $\boldsymbol{h}$ to have lower dimension than $\boldsymbol{x}$
- Force it to capture most salient features of training data

Weights W are learnt using:

1. Training samples, and
2. a loss function

Autoencoder is a feed-forward non-recurrent neural net

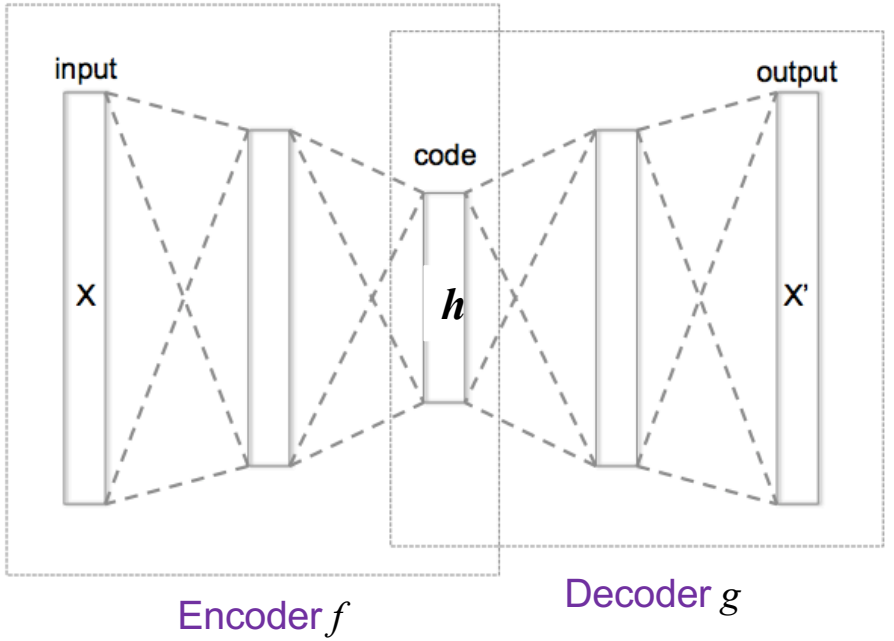
- With an input layer, an output layer and one or more hidden layers
- Can be trained using the same techniques
  - Compute gradients using back-propagation
  - Followed by minibatch gradient descent

Autoencoder training using a loss function

- Encoder f and decoder g

$$f : X \rightarrow \boldsymbol{h}$$
$$g : \boldsymbol{h} \rightarrow X$$
$$\arg\min_{f,g} \left\| X - (f \circ g)X \right\|^2$$

Autoencoder with 3 fully connected hidden layers



- One hidden layer
  - Non-linear encoder
  - Takes input $\boldsymbol{x}$ ε R^d
  - Maps into output $\boldsymbol{h}$ ε R^p

$$\boldsymbol{h} = \sigma_1(W\boldsymbol{x} + \boldsymbol{b})$$
$$\boldsymbol{x}' = \sigma_2(W'\boldsymbol{h} + \boldsymbol{b}')$$

σ is an element-wise activation function such as sigmoid or Relu

Trained to minimize reconstruction error (such as sum of squared errors)

$$L(\boldsymbol{x}, \boldsymbol{x}') = \left\| \boldsymbol{x} - \boldsymbol{x}' \right\|^2 = \left\| \boldsymbol{x} - \sigma_2(W^t(\sigma_1(W\boldsymbol{x} + \boldsymbol{b})) + \boldsymbol{b}') \right\|^2$$

Provides a compressed representation of the input $\boldsymbol{x}$

Cases when Autoencoder Learning Fails

Where autoencoders fail to learn anything useful:

1. Capacity of encoder/decoder f/g is too high
    - Capacity controlled by depth
2. Hidden code $\boldsymbol{h}$ has dimension equal to input $\boldsymbol{x}$
3. *Overcomplete* case: where hidden code $\boldsymbol{h}$ has dimension greater than input $\boldsymbol{x}$
    - Even a linear encoder/decoder can learn to copy input to output without learning anything useful about data distribution

Right Autoencoder Design: Use regularization

Encoder/Decoder Capacity

If encoder f and decoder g are allowed too much capacity

- autoencoder can learn to perform the copying task without learning any useful information about distribution of data

Autoencoder with a one-dimensional code and a very powerful nonlinear encoder can learn to map $\boldsymbol{x}^{(i)}$ to code i.

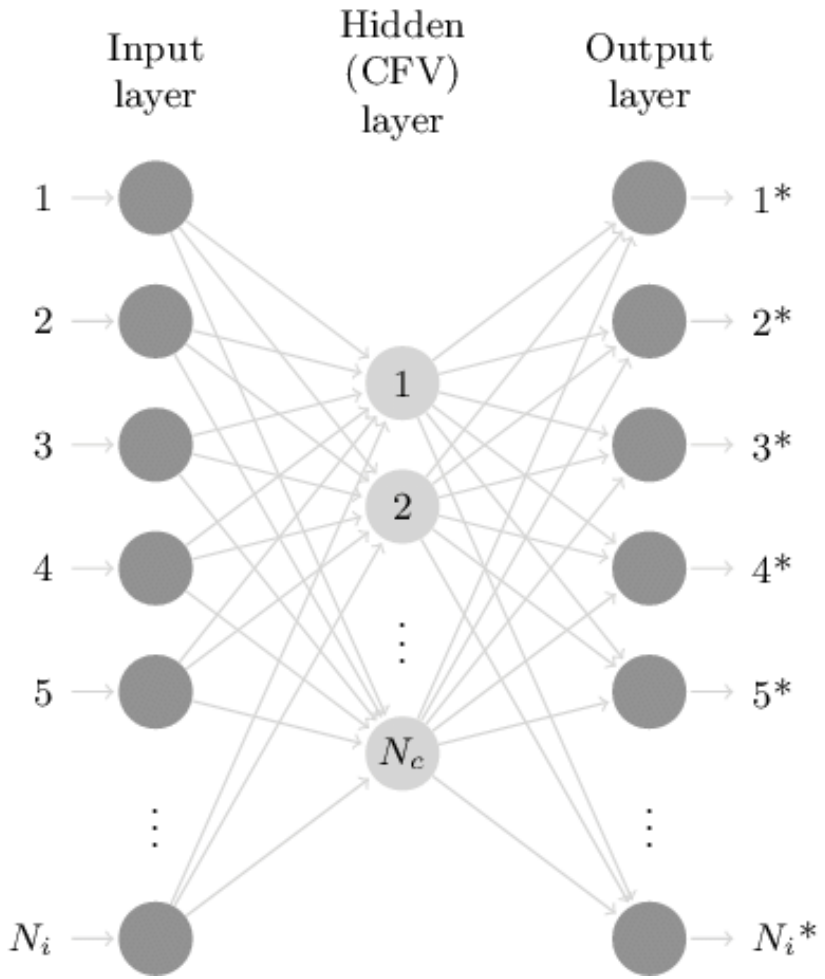
- The decoder can learn to map these integer indices back to the values of specific training examples

Autoencoder trained for copying task fails to learn anything useful if f/g capacity is too great

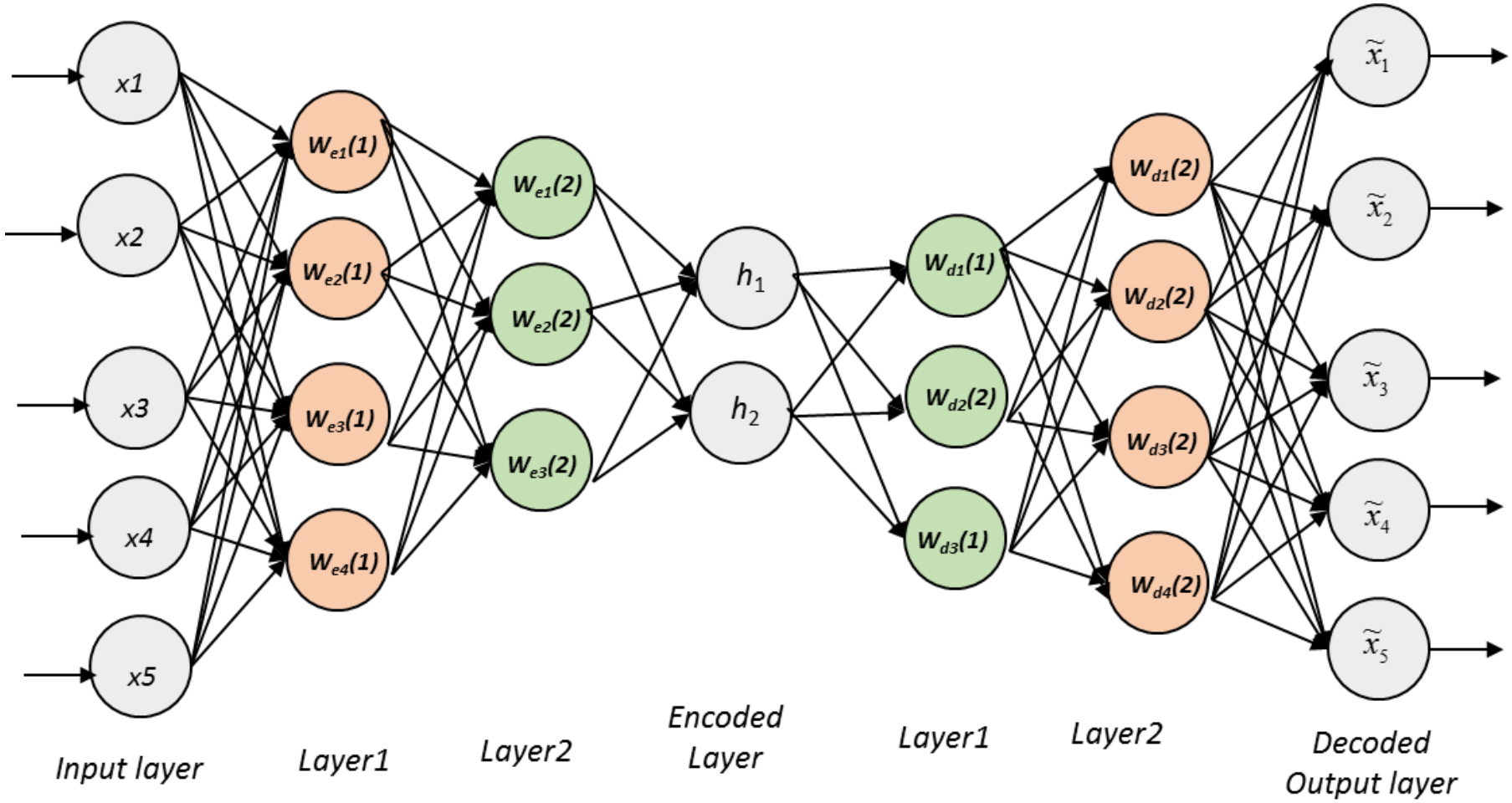
Types of Autoencoders

- Vanilla autoencoder
- Multilayer autoencoder
- Stacked autoencoder
- Deep autoencoder
- Convolutional autoencoder
- Regularized autoencoder

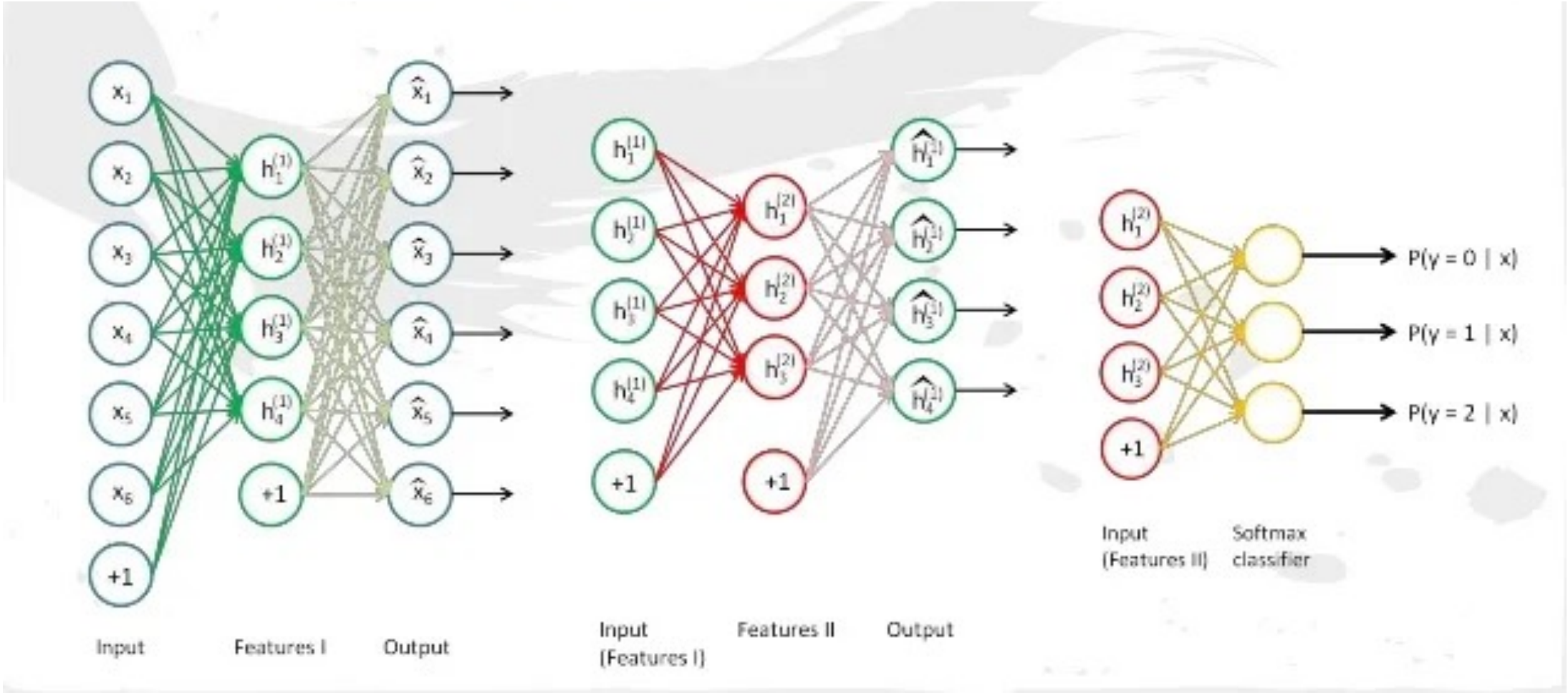
Vanilla autoencoder



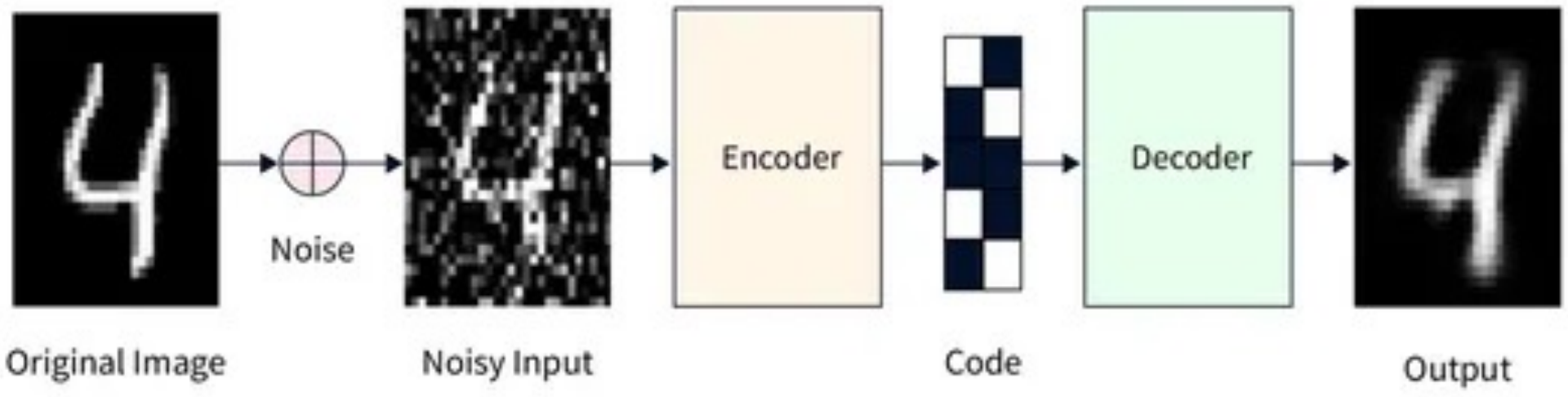
Multilayer Autoencoder



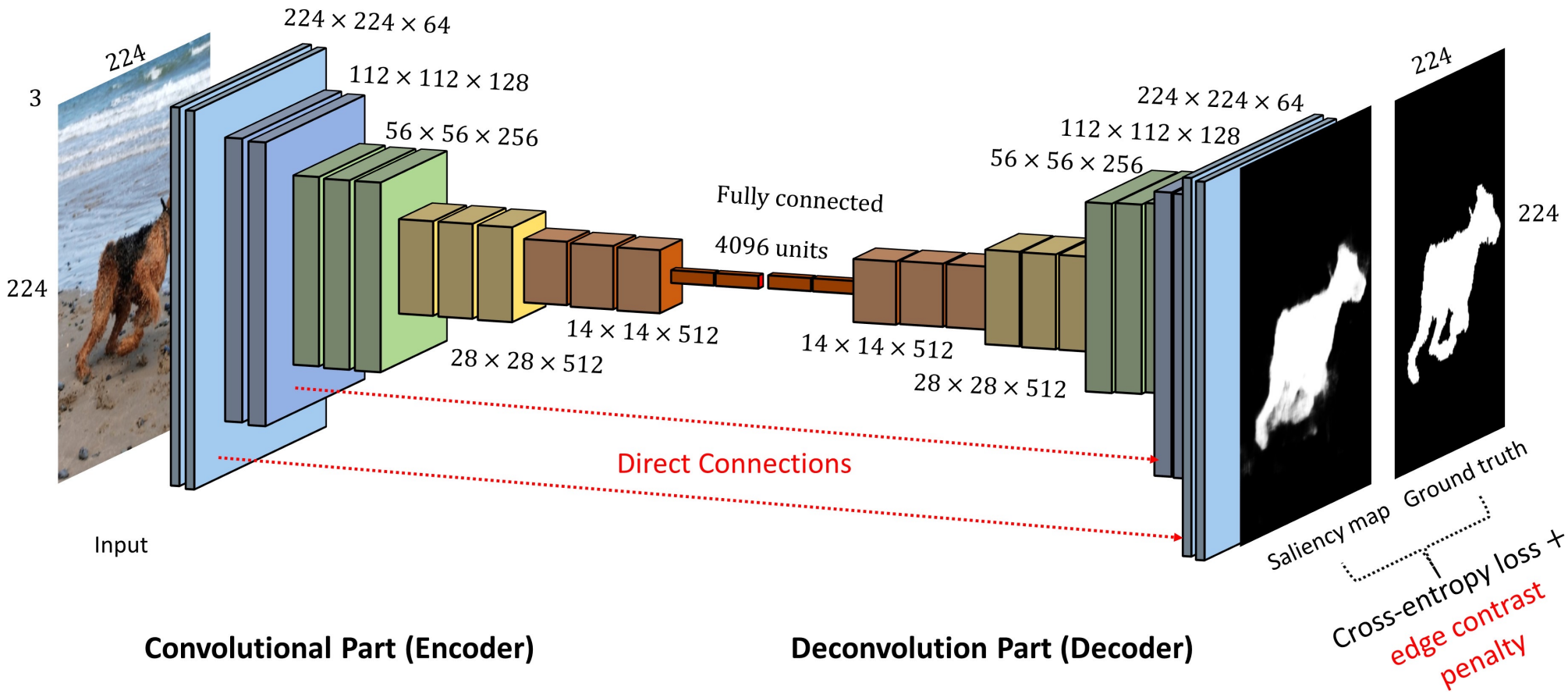
Stacked Autoencoder



Denoising Autoencoder

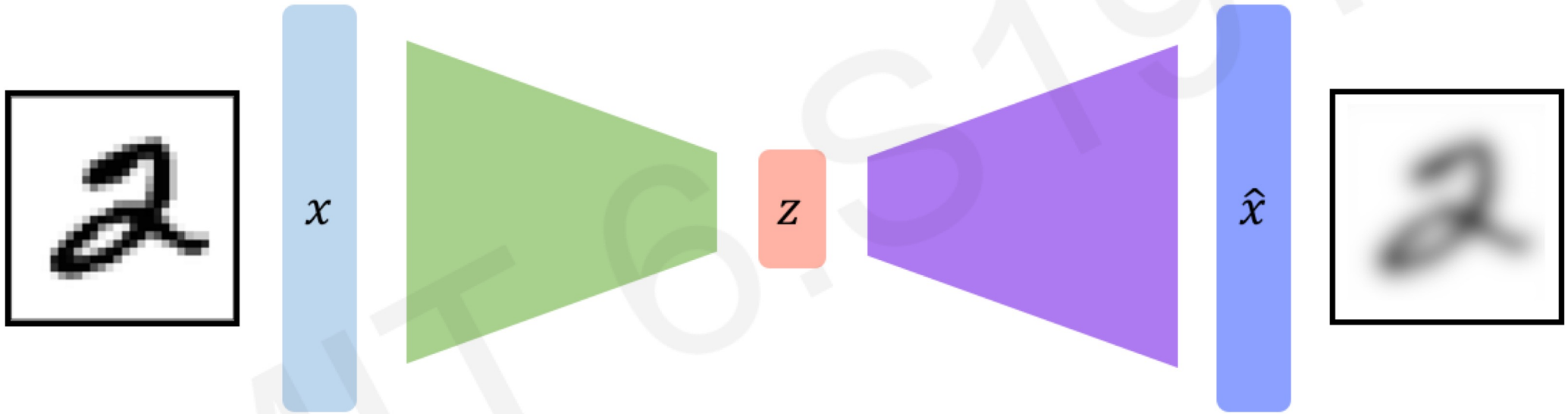


Convolutional Autoencoder

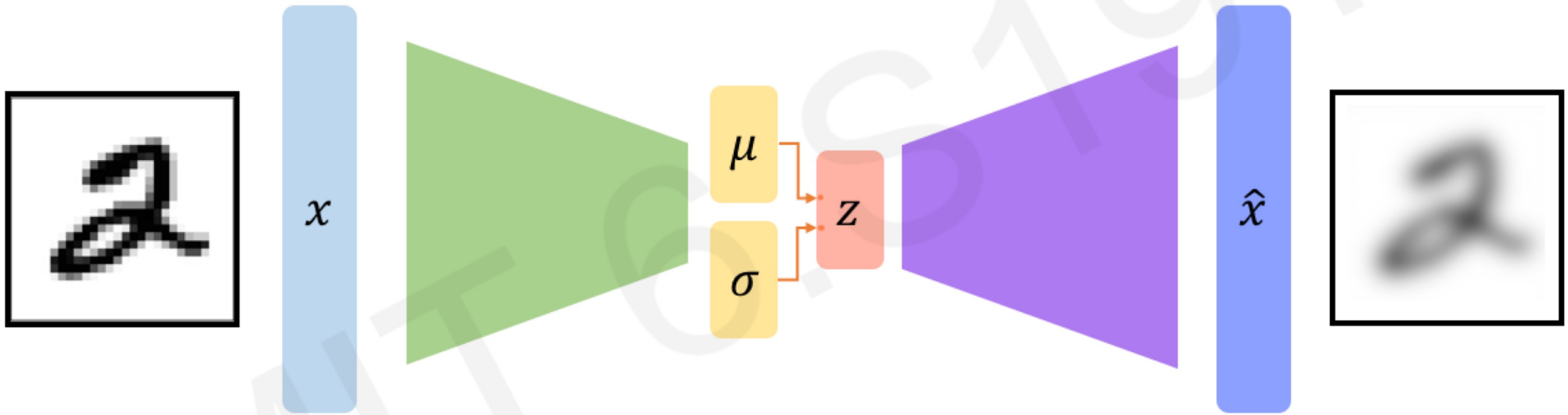


Variational Autoencoders (VAEs)

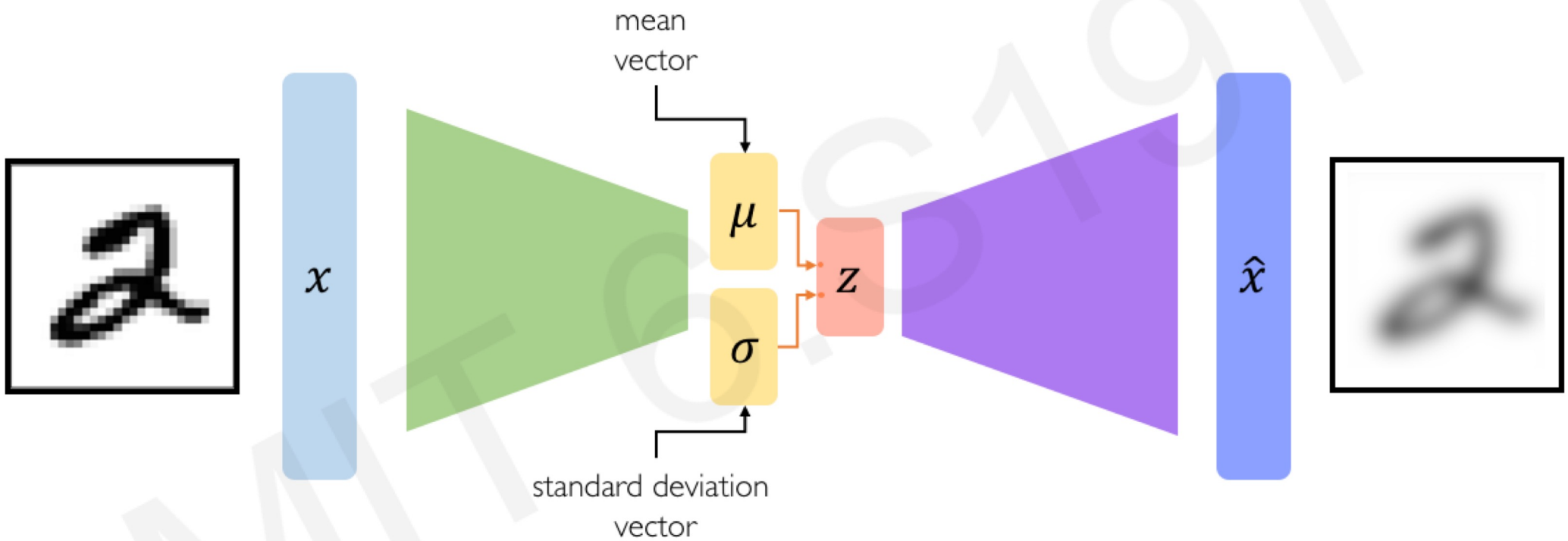
Traditional autoencoders



VAEs: key difference with traditional autoencoder



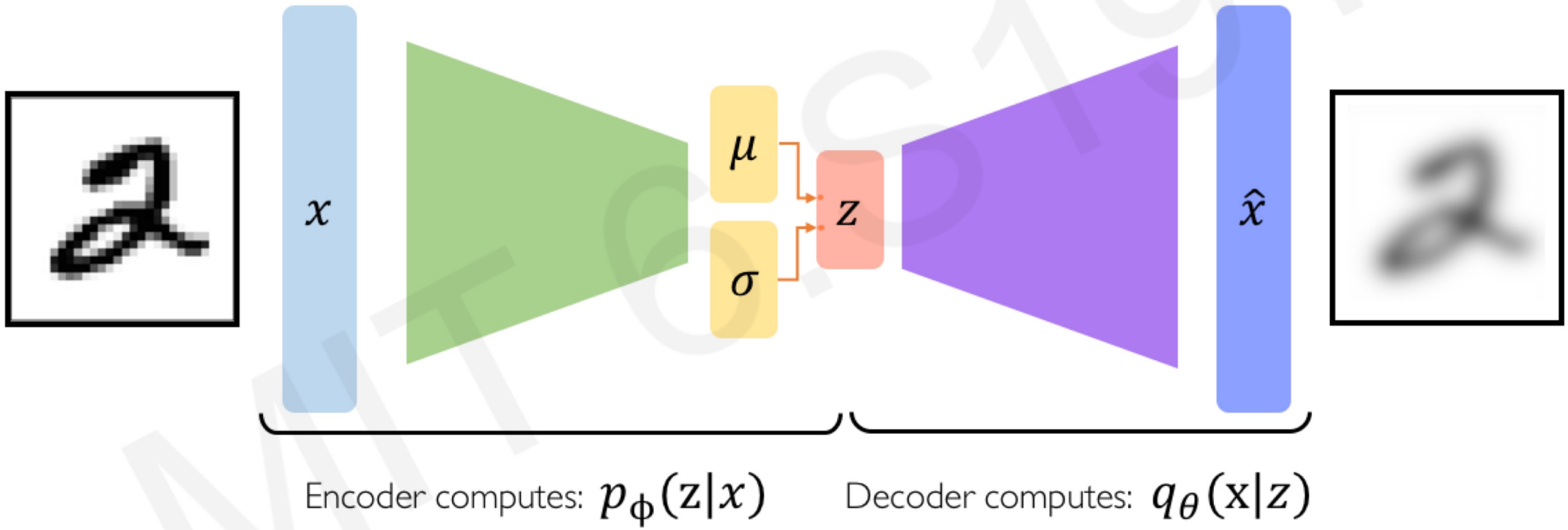
VAEs: key difference with traditional autoencoder



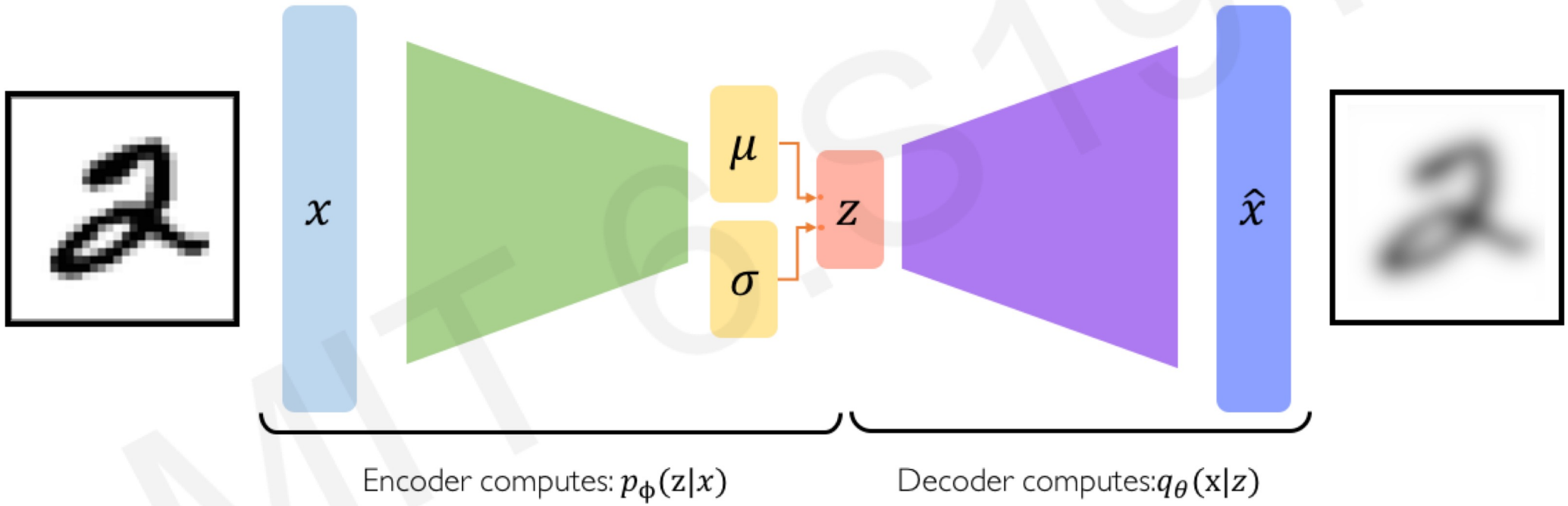
Variational autoencoders are a probabilistic twist on autoencoders!

Sample from the mean and standard dev. to compute latent sample

VAE optimization

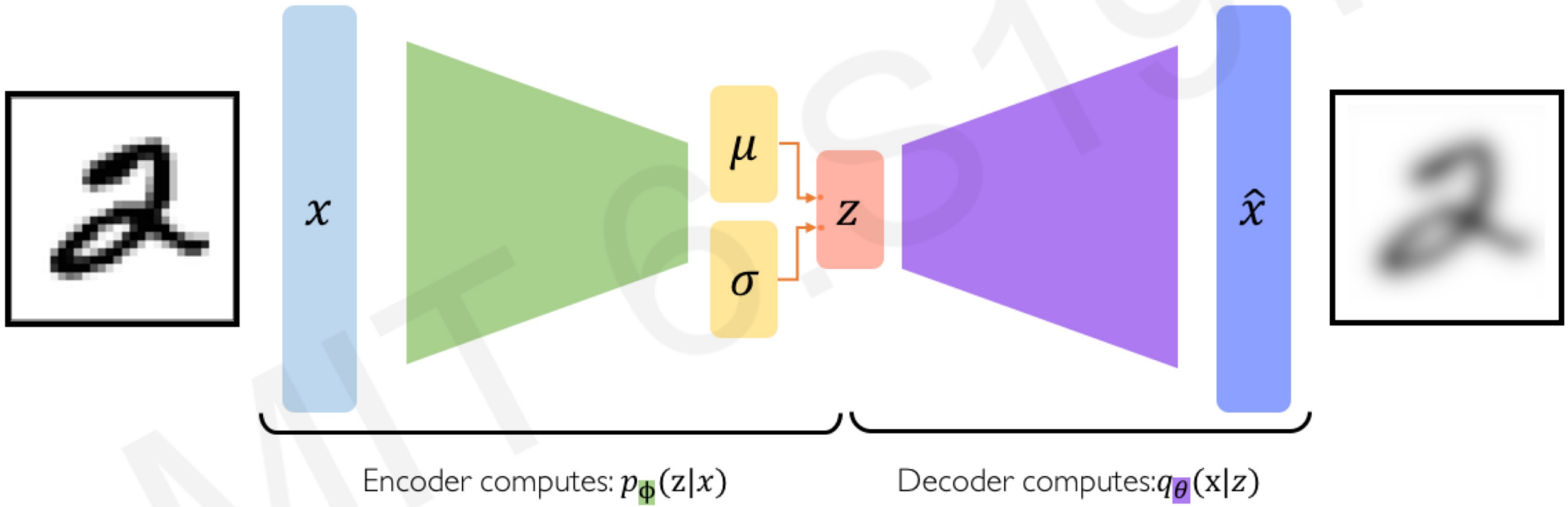


VAE optimization



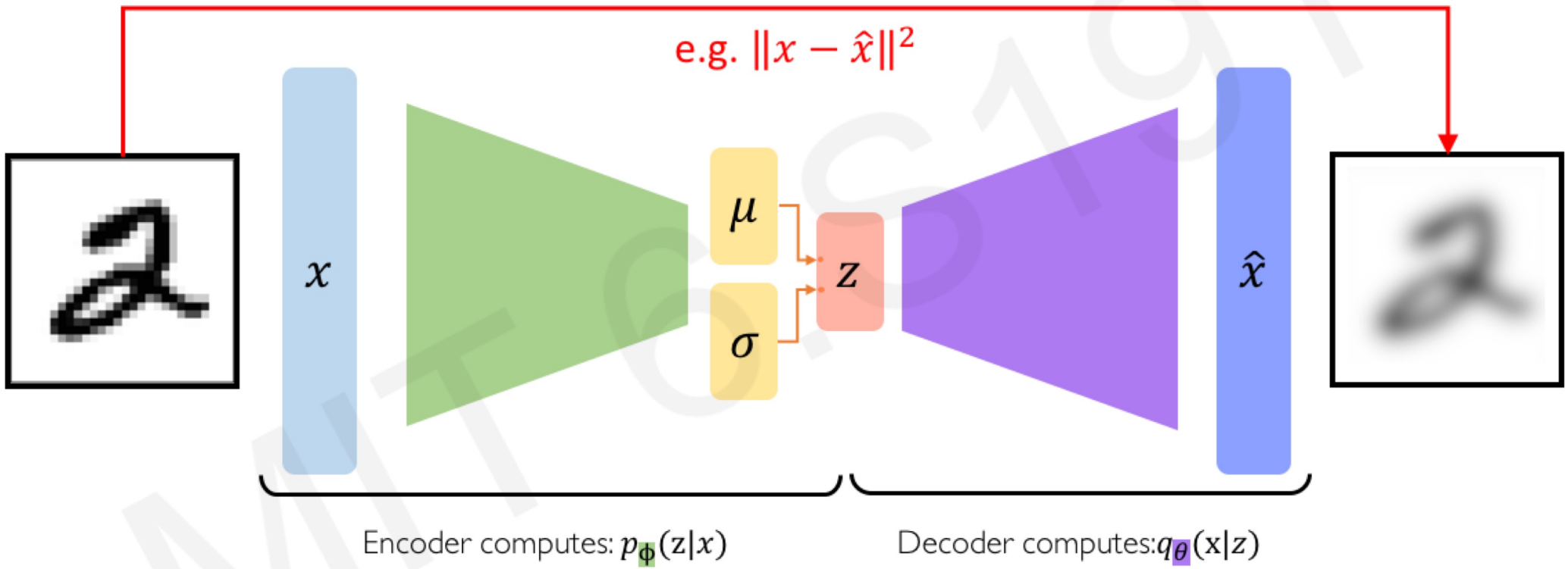
$$\mathcal{L}(\phi, \theta) = \text{(reconstruction loss)} + \text{(regularization term)}$$

VAE optimization



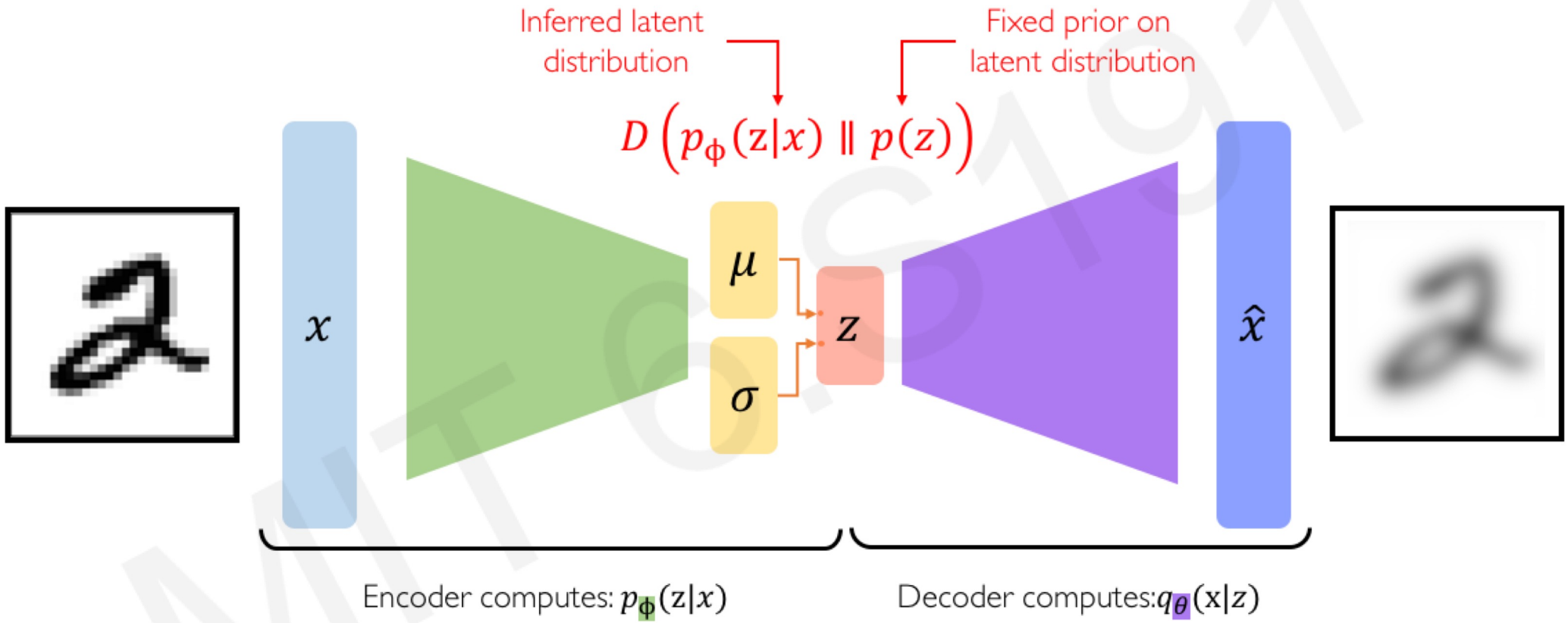
$$\mathcal{L}(\phi, \theta, x) = \text{(reconstruction loss)} + \text{(regularization term)}$$

VAE optimization



$$\mathcal{L}(\phi, \theta, x) = \boxed{\text{(reconstruction loss)}} + \text{(regularization term)}$$

VAE optimization



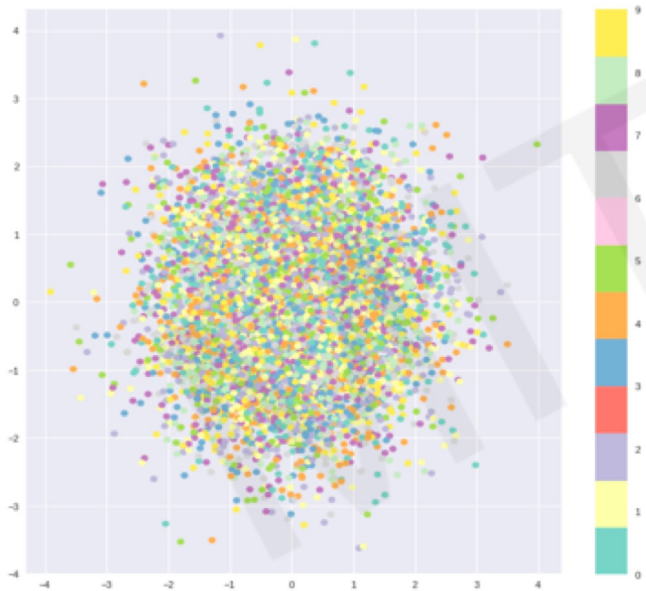
$$\mathcal{L}(\phi, \theta, x) = \text{(reconstruction loss)} + \boxed{\text{(regularization term)}}$$

Priors on the latent distribution

$$D\left(p_\phi(z|x) \parallel p(z)\right)$$

Inferred latent distribution

Fixed prior on latent distribution



Common choice of prior:

$$p(z) = \mathcal{N}(\mu = 0, \sigma^2 = 1)$$

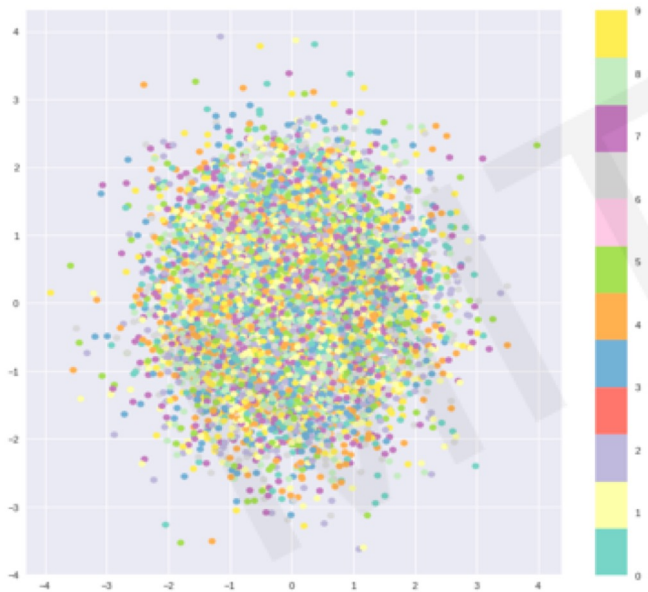
- Encourages encodings to distribute encodings evenly around the center of the latent space
- Penalize the network when it tries to "cheat" by clustering points in specific regions (ie. memorizing the data)

Priors on the latent distribution

$$D\left(p_\phi(z|x) \parallel p(z)\right)$$

$$= -\frac{1}{2}\sum_{j=0}^{k-1}\left(\sigma_j + \mu_j^2 - 1 - \log\sigma_j\right)$$

KL-divergence between the two distributions

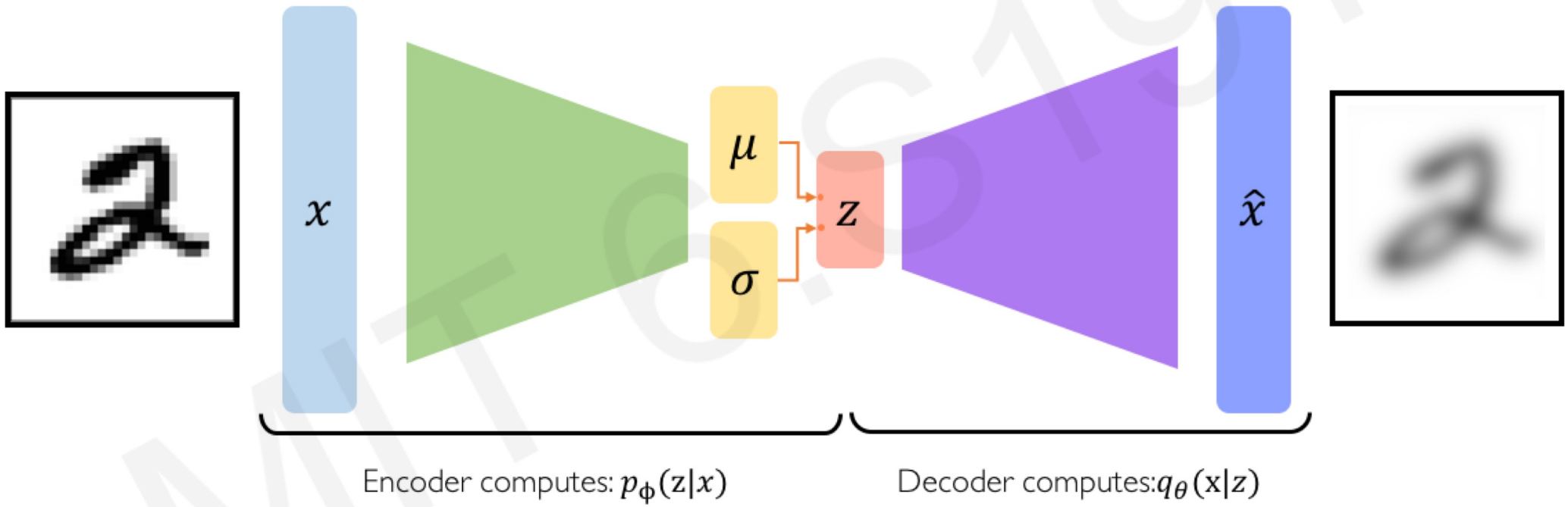


Common choice of prior:

$$p(z) = \mathcal{N}(\mu = 0, \sigma^2 = 1)$$

- Encourages encodings to distribute encodings evenly around the center of the latent space
- Penalize the network when it tries to "cheat" by clustering points in specific regions (ie. memorizing the data)

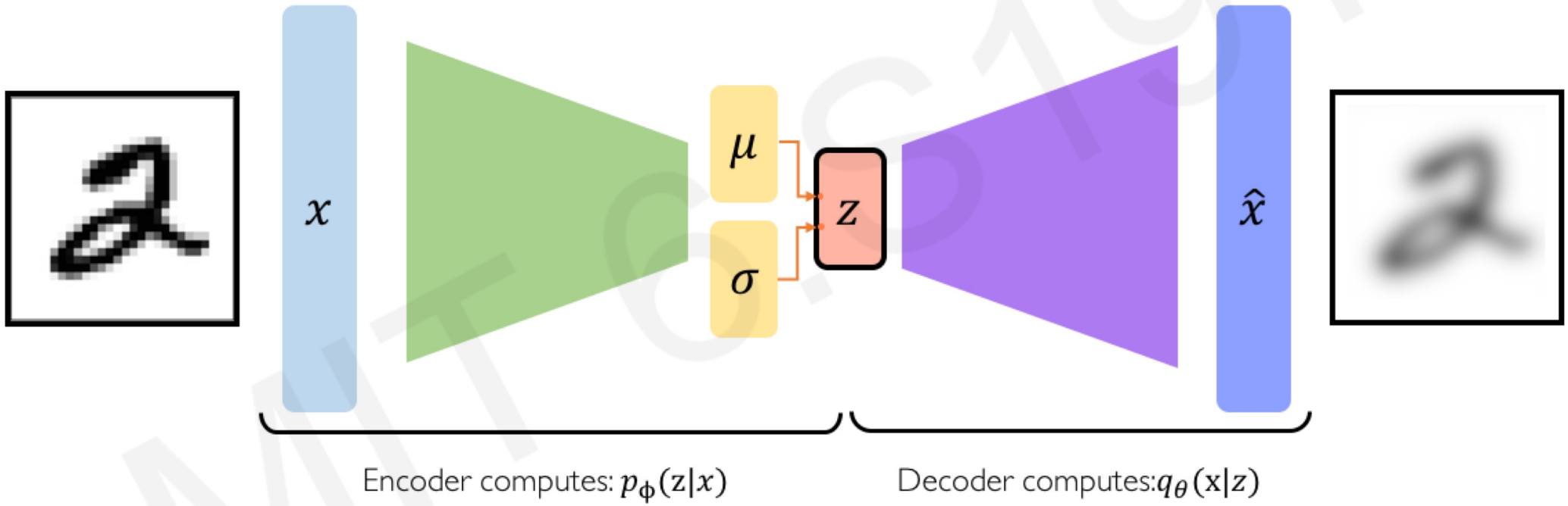
VAEs computation graph



$\mathcal{L}(\phi, \theta, x) =$ (reconstruction loss) + (regularization term)

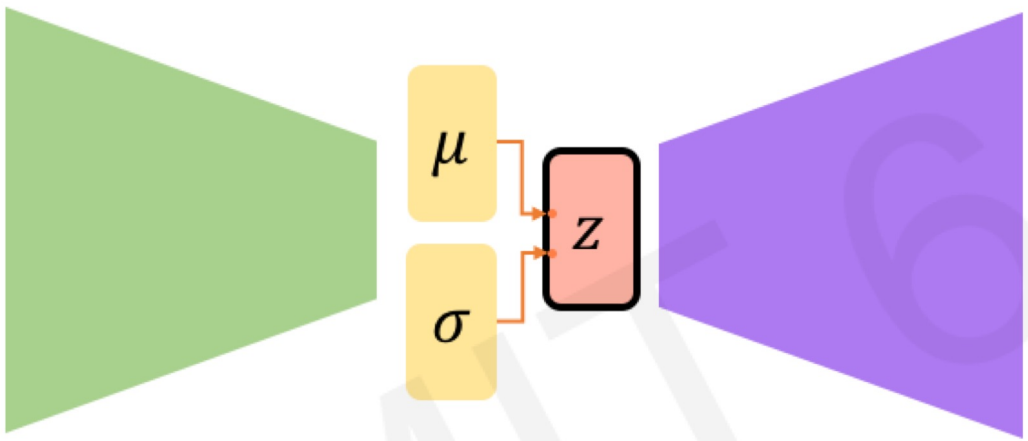
VAEs computation graph

Problem: We cannot backpropagate gradients through sampling layers!



$\mathcal{L}(\phi, \theta, x) =$ (reconstruction loss) + (regularization term)

Reparametrizing the sampling layer



Key Idea:

~~$z \sim \mathcal{N}(\mu, \sigma^2)$~~

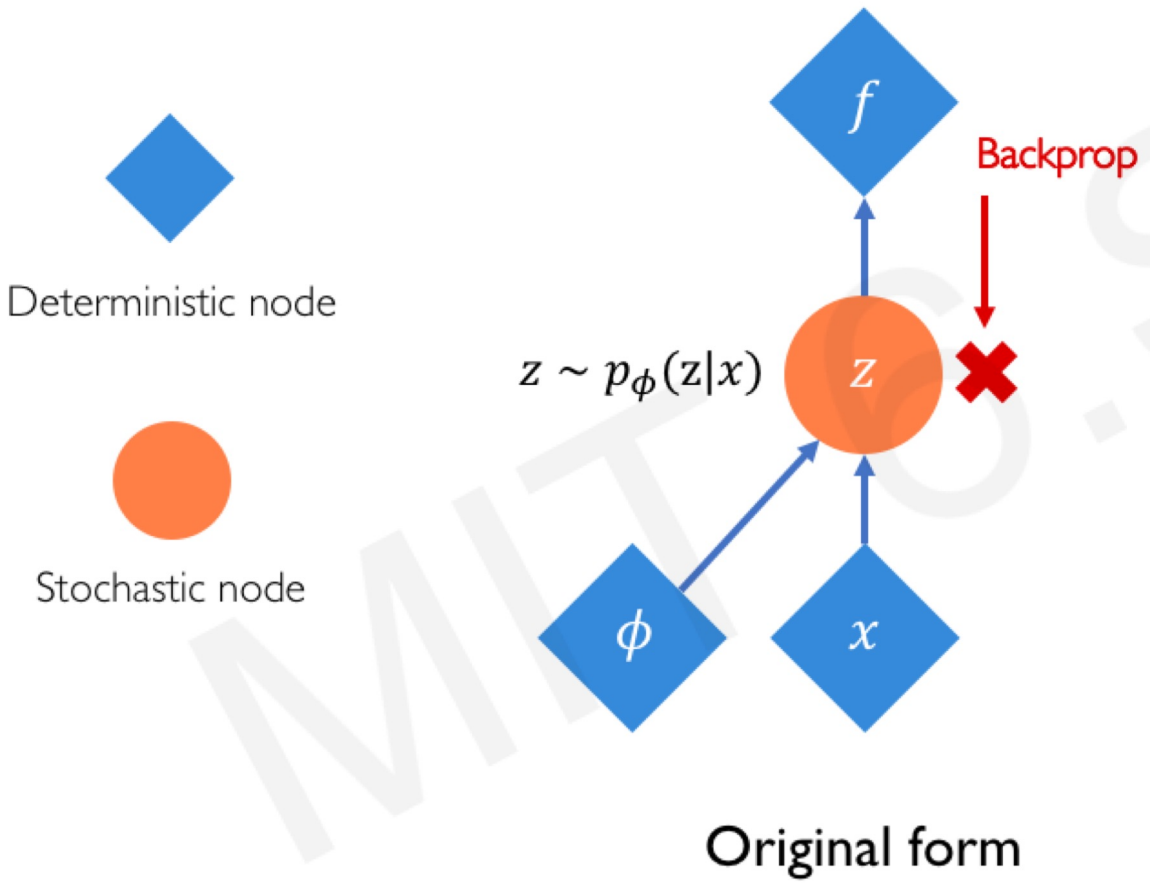
Consider the sampled latent vector z as a sum of

- a fixed μ vector,
- and fixed σ vector, scaled by random constants drawn from the prior distribution

$$\Rightarrow z = \mu + \sigma \odot \varepsilon$$

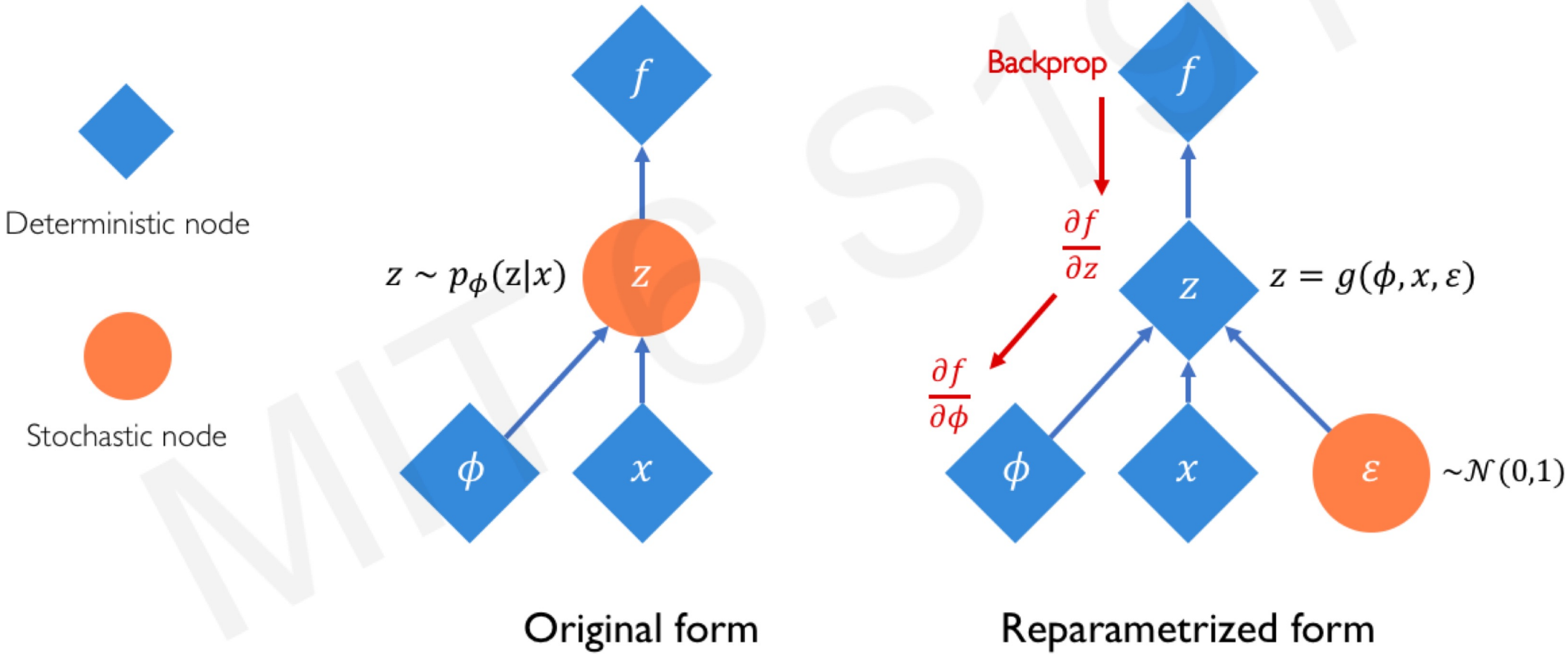
where $\varepsilon \sim \mathcal{N}(0,1)$

Reparametrizing the sampling layer



Original form

Reparametrizing the sampling layer



VAEs: Latent perturbation

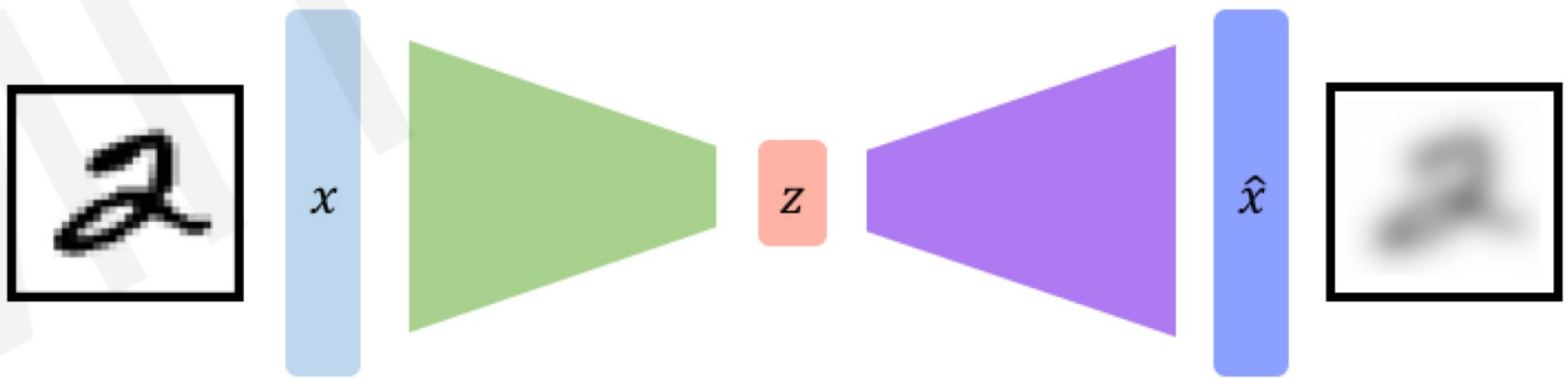
Slowly increase or decrease a **single latent variable**
Keep all other variables fixed

Head pose

Different dimensions of z encodes **different interpretable latent features**

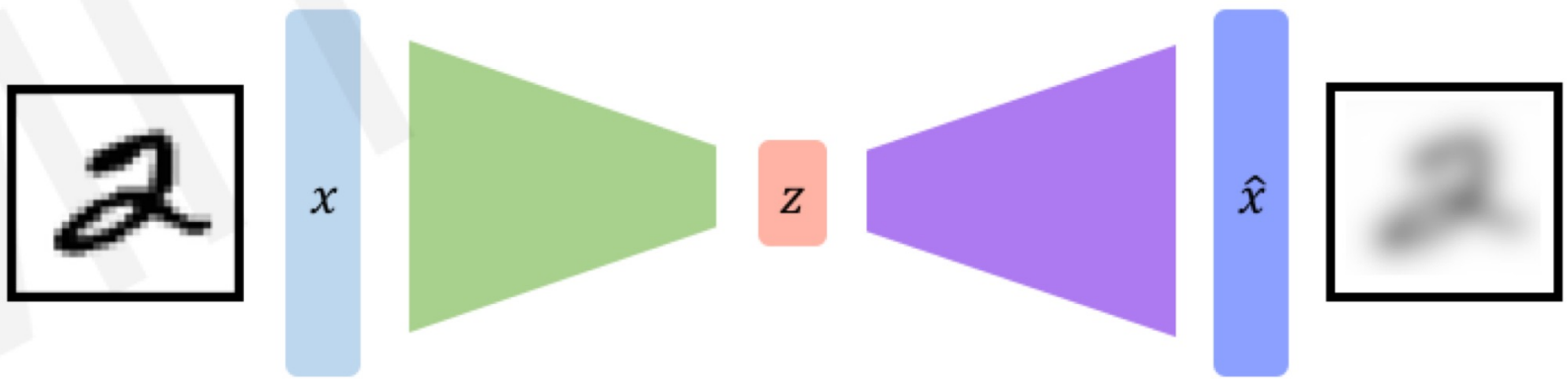
VAE summary

. Compress representation of world to something we can use to learn



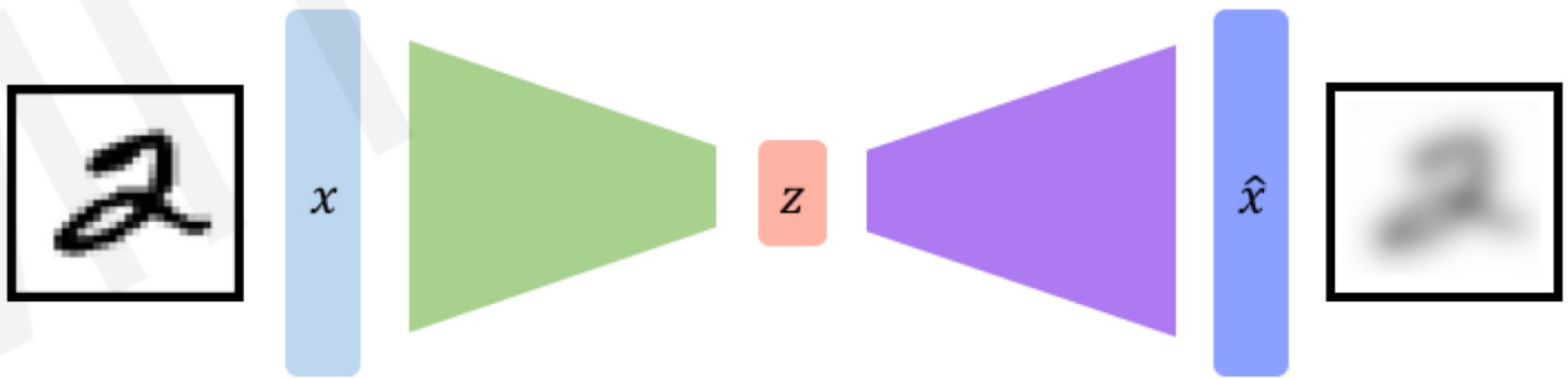
VAE summary

1. Compress representation of world to something we can use to learn
2. Reconstruction allows for unsupervised learning (no labels!)



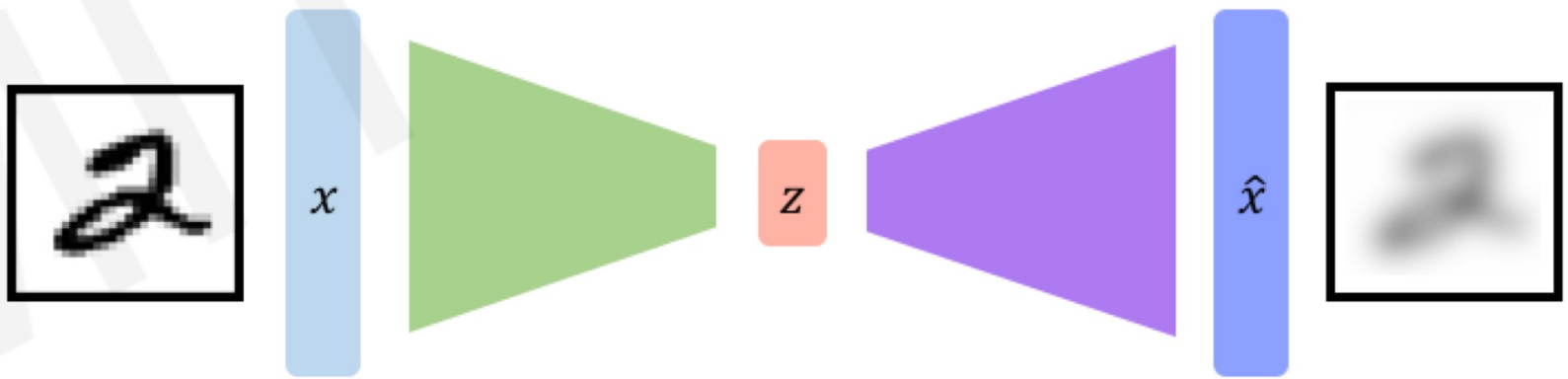
VAE summary

1. Compress representation of world to something we can use to learn
2. Reconstruction allows for unsupervised learning (no labels!)
3. Reparameterization trick to train end-to-end



VAE summary

1. Compress representation of world to something we can use to learn
2. Reconstruction allows for unsupervised learning (no labels!)
3. Reparameterization trick to train end-to-end
4. Interpret hidden latent variables using perturbation



VAE summary

1. Compress representation of world to something we can use to learn
2. Reconstruction allows for unsupervised learning (no labels!)
3. Reparameterization trick to train end-to-end
4. Interpret hidden latent variables using perturbation
5. Generating new examples