Lecture 18: Variational Autoencoders

CS109B Data Science 2

Pavlos Protopapas, Mark Glickman, and Chris Tanner

Outline

Motivation for Variational Autoencoders (VAE)

Mechanics of VAE

Separability of VAE

The math behind everything

Generative models

Outline

Motivation for Variational Autoencoders (VAE)

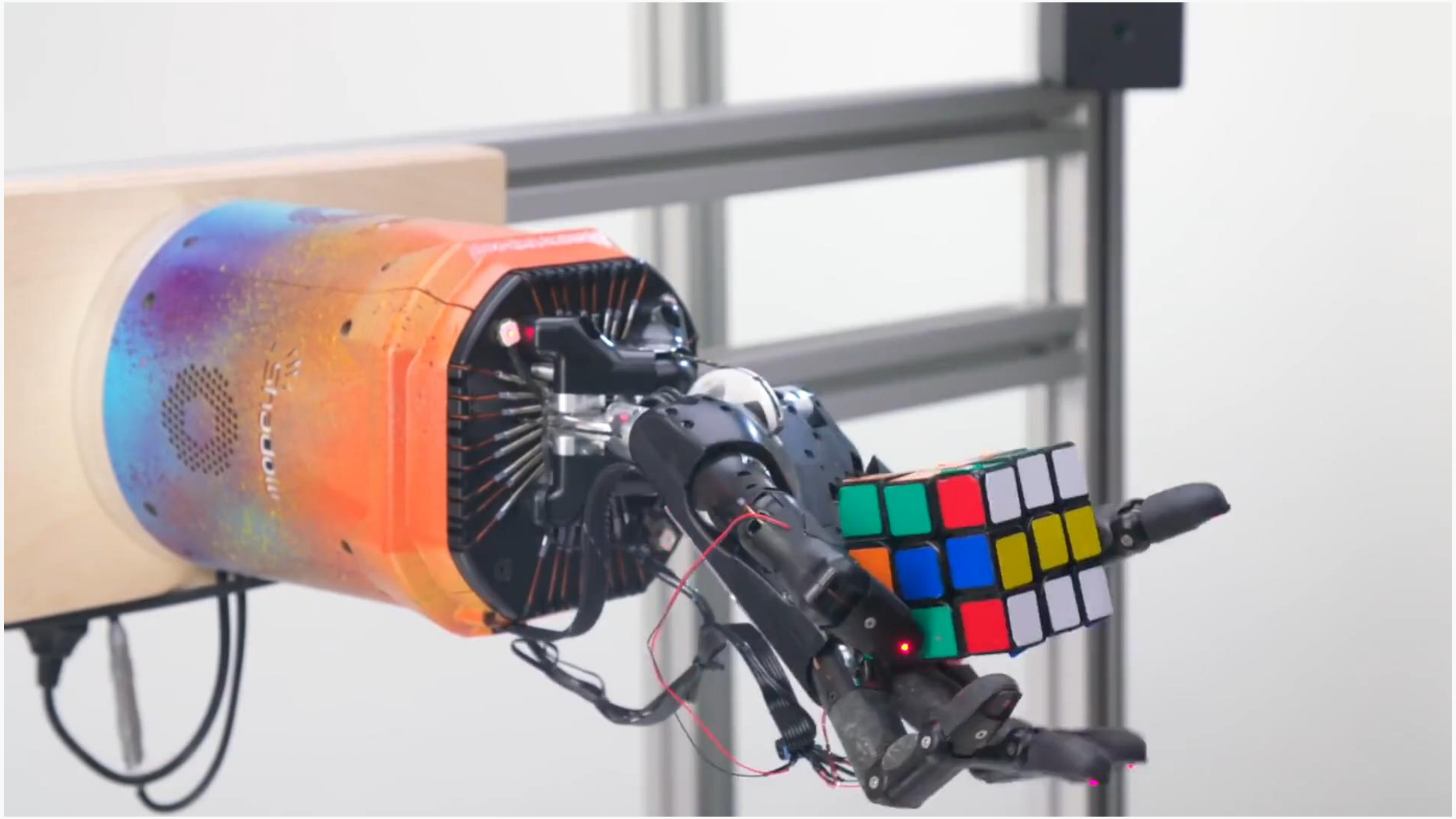
Mechanics of VAE

Separability of VAE

The math behind everything

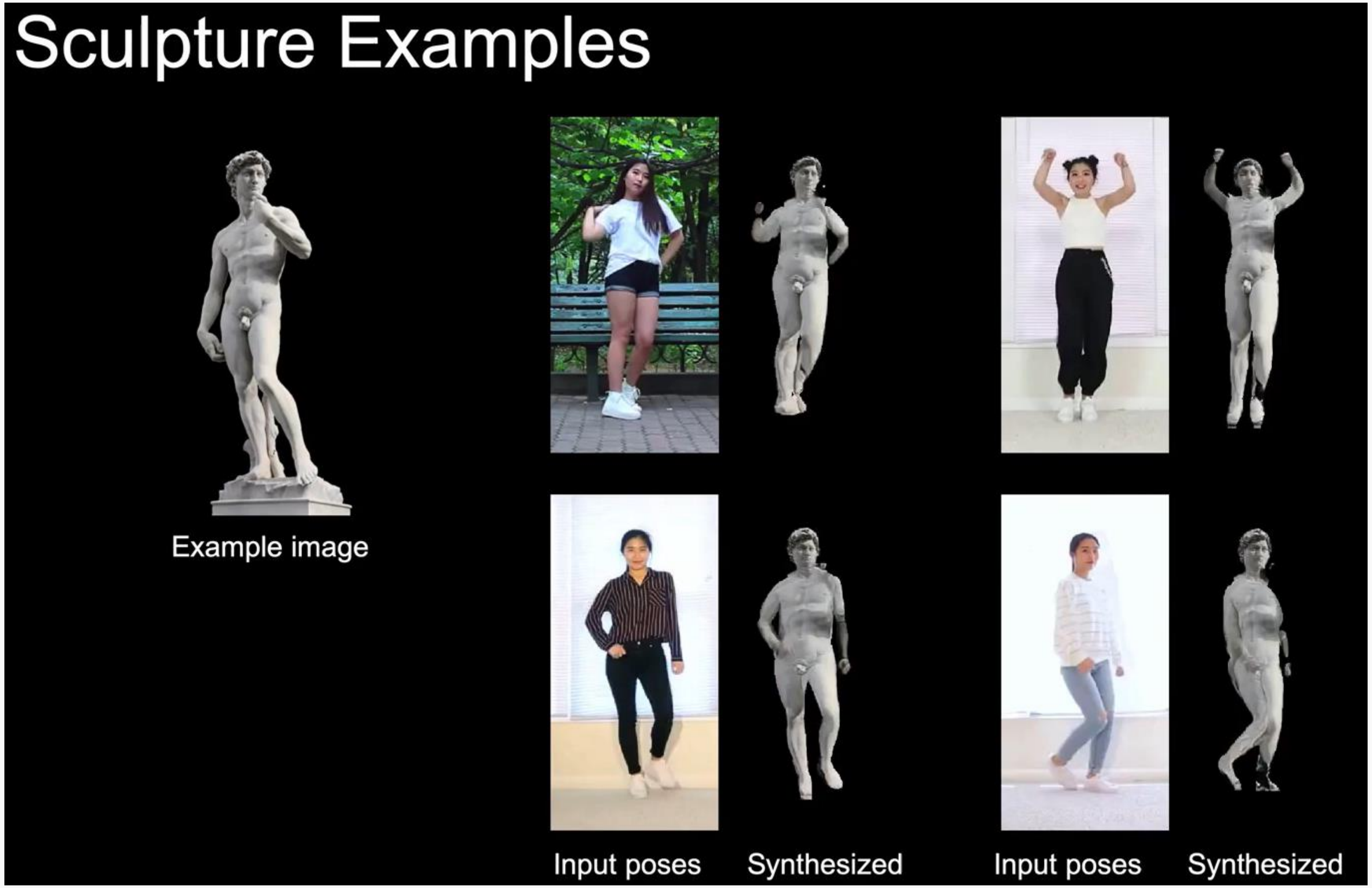
Generative models

State of the Art in AI



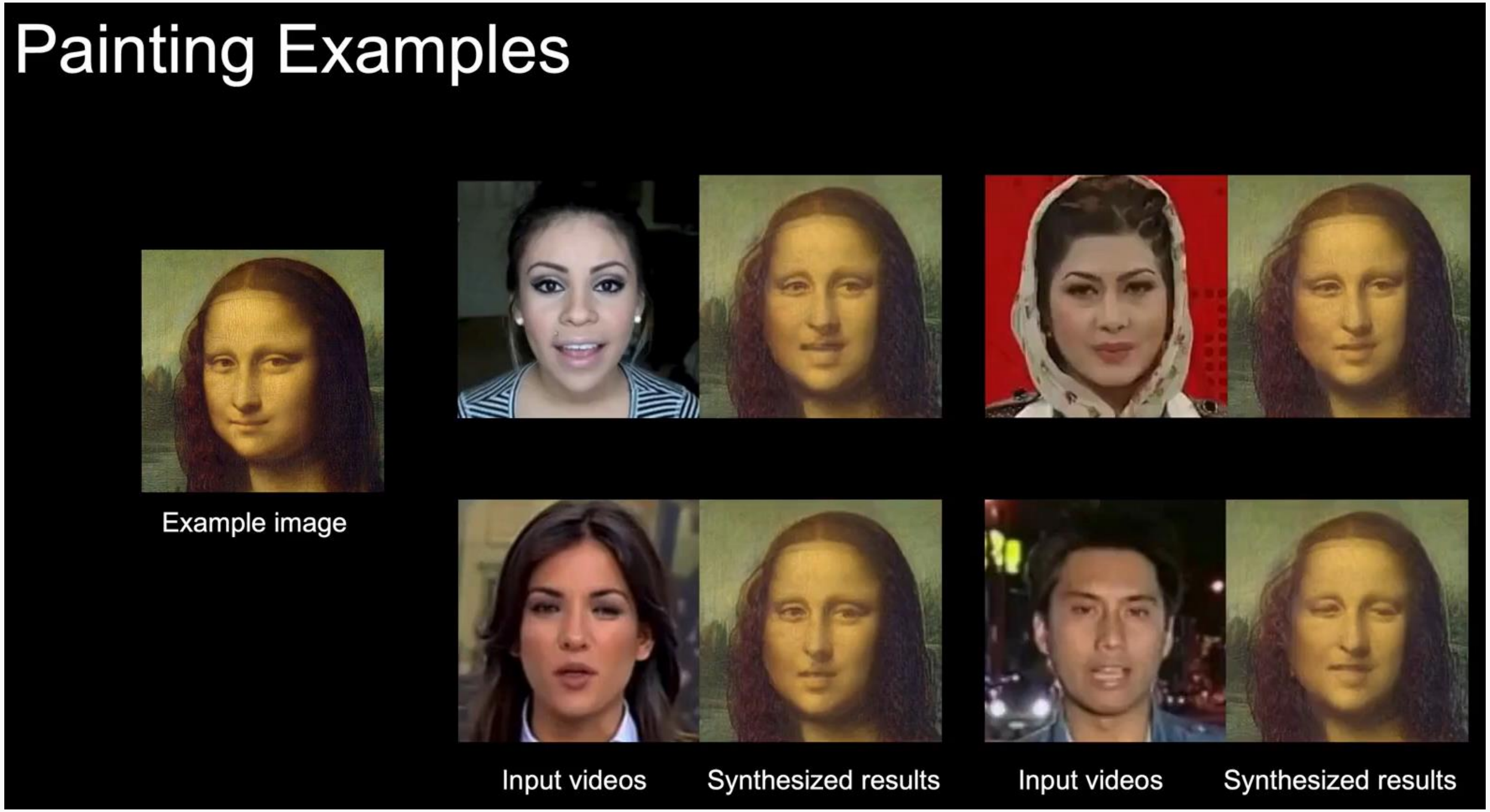
https://openai.com/blog/solving-rubiks-cube/

State of the Art in AI



https://nvlabs.github.io/few-shot-vid2vid/

State of the Art in AI



https://nvlabs.github.io/few-shot-vid2vid/

Generative Modeling

"What I cannot create, I do not understand."
- Richard Feynman

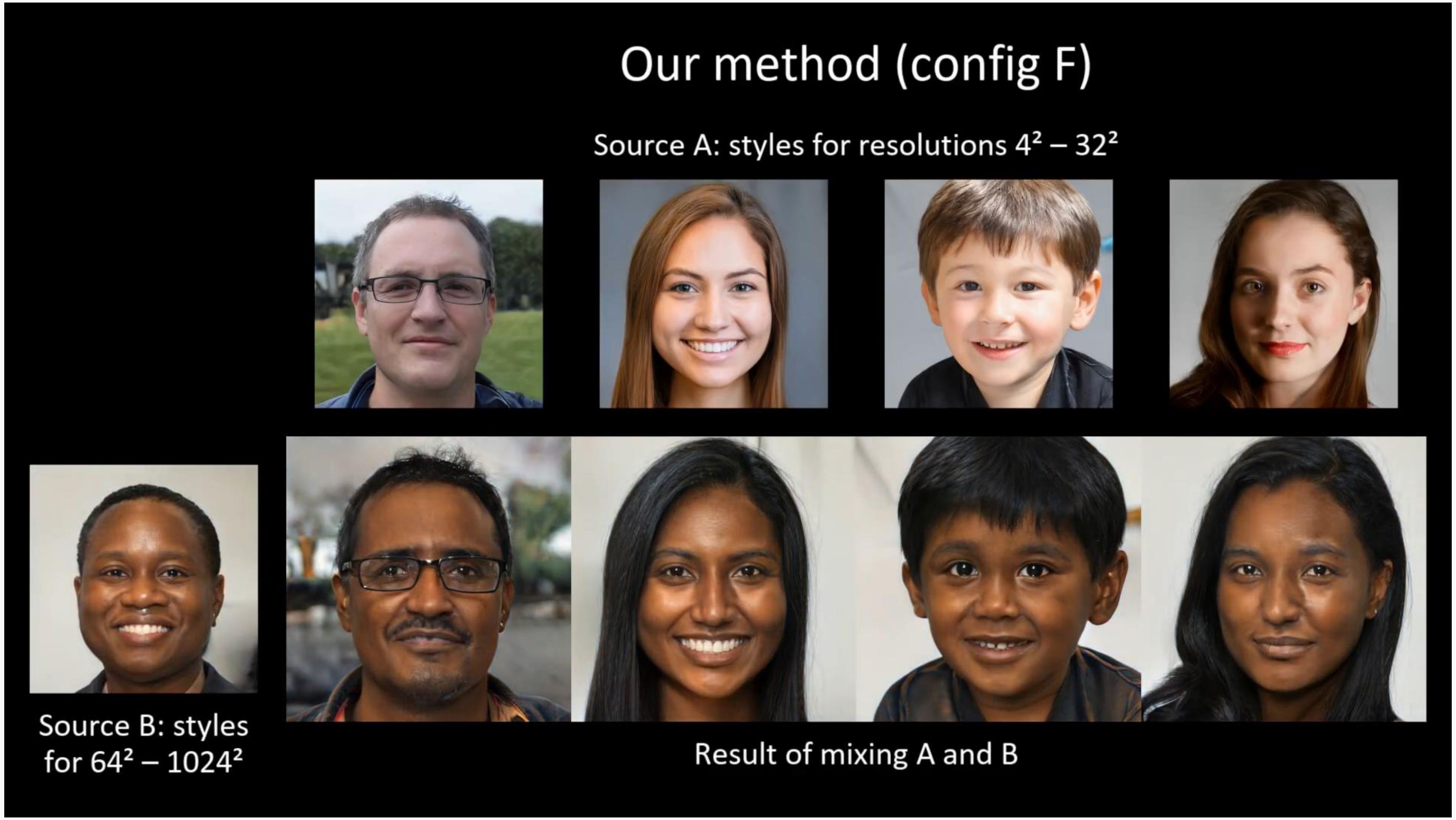
Generative Modeling

https://github.com/tkarras/progressive_growing_of_gans

Generative Modeling

https://github.com/tkarras/progressive_growing_of_gans

Generative Modeling



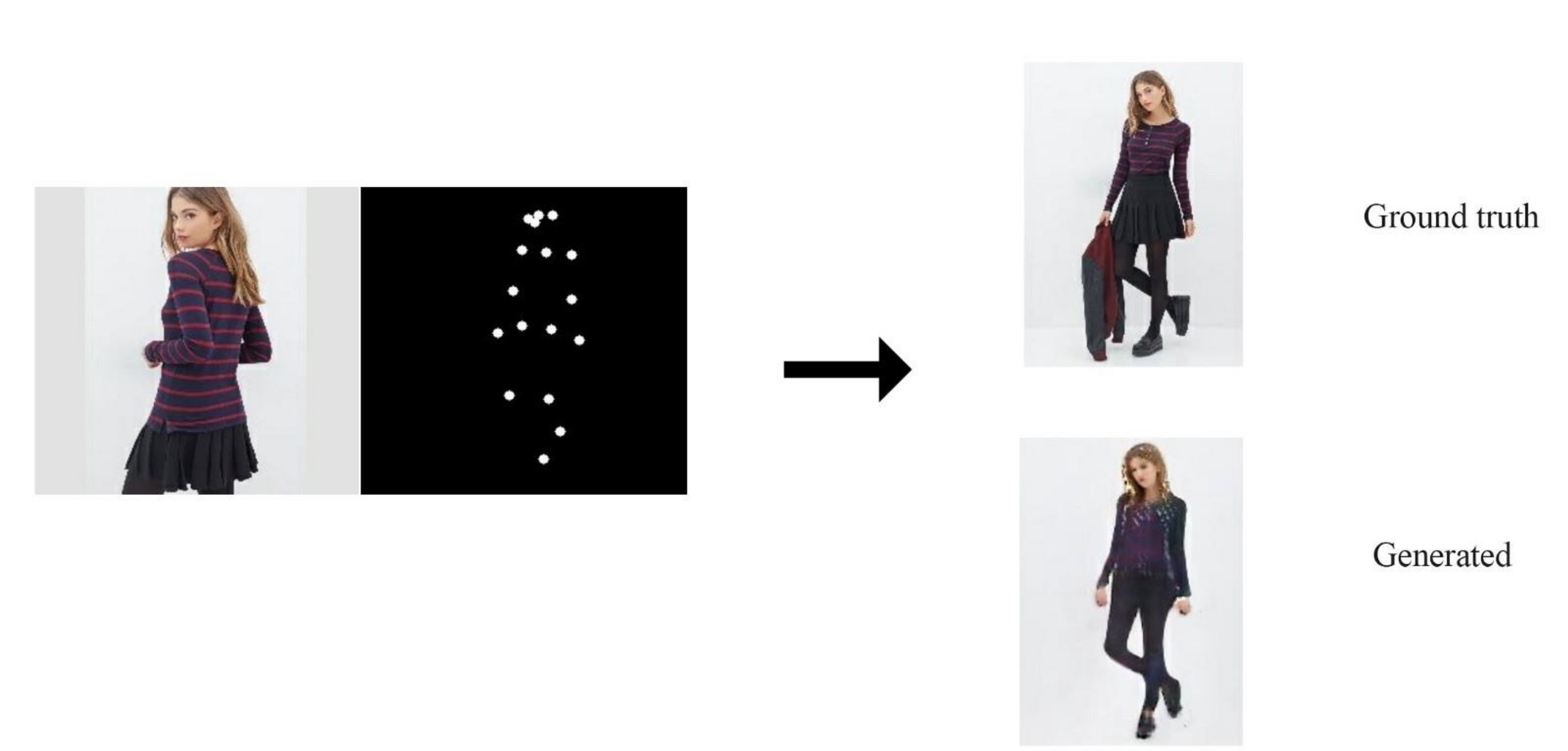
https://github.com/NVlabs/stylegan2

Generative Modeling

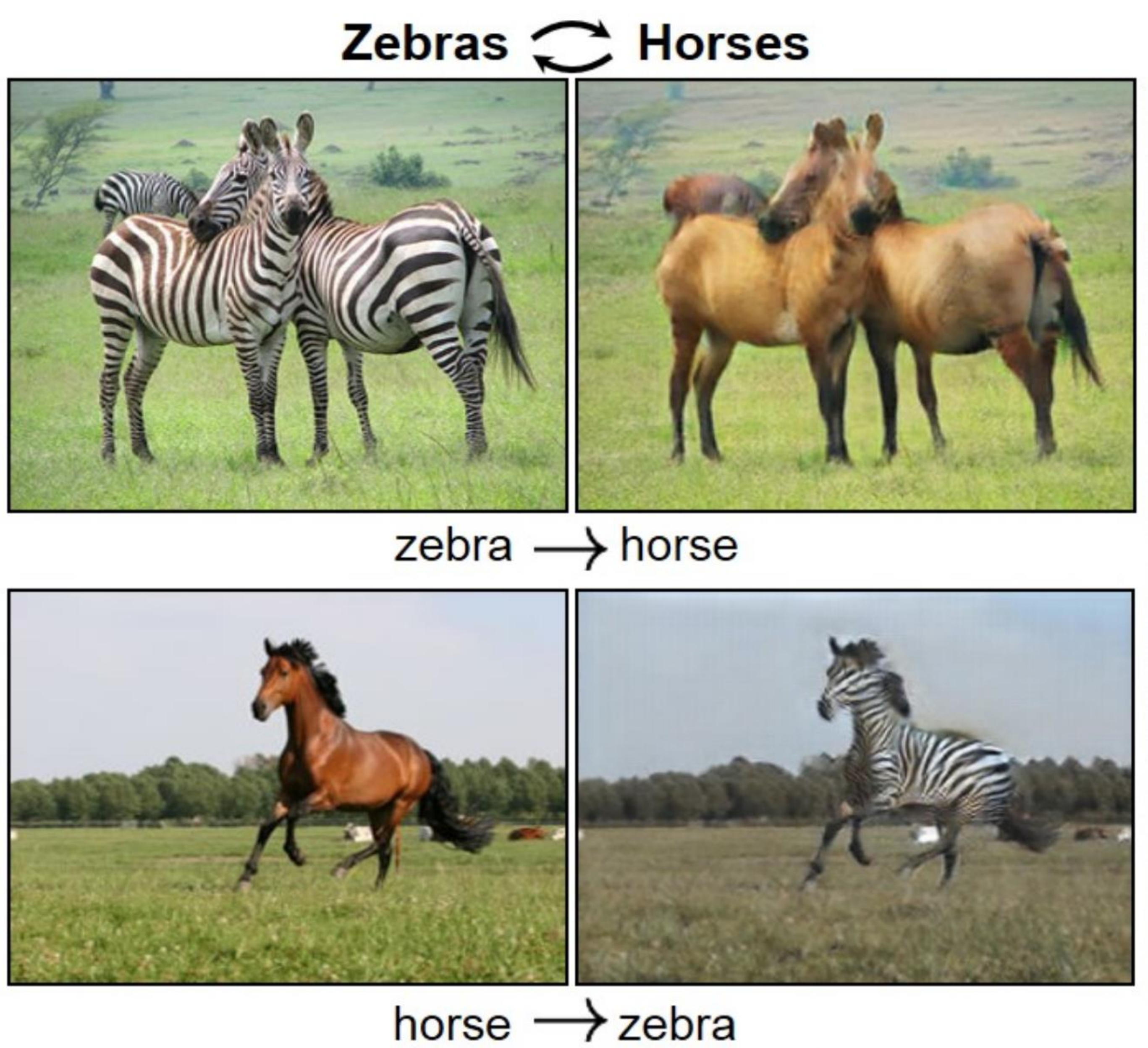
Figure 7: Generated samples

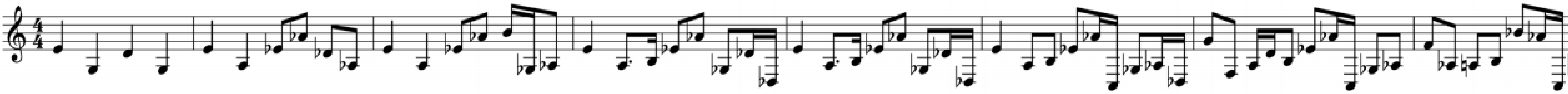
https://arxiv.org/pdf/1708.05509.pdf

Generative Modeling

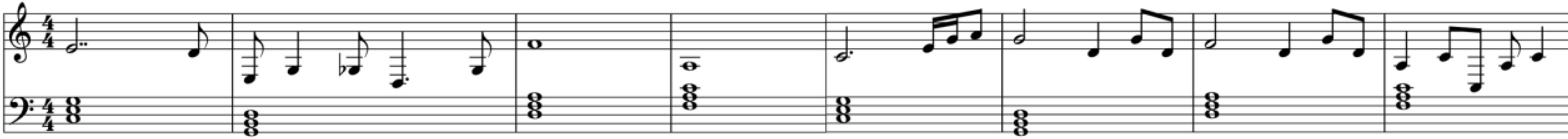


Generative Modeling

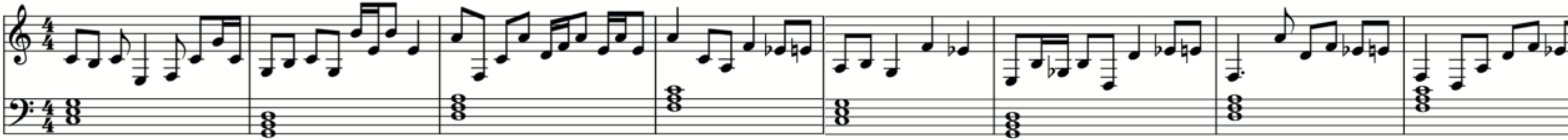




(a) MidiNet model 1



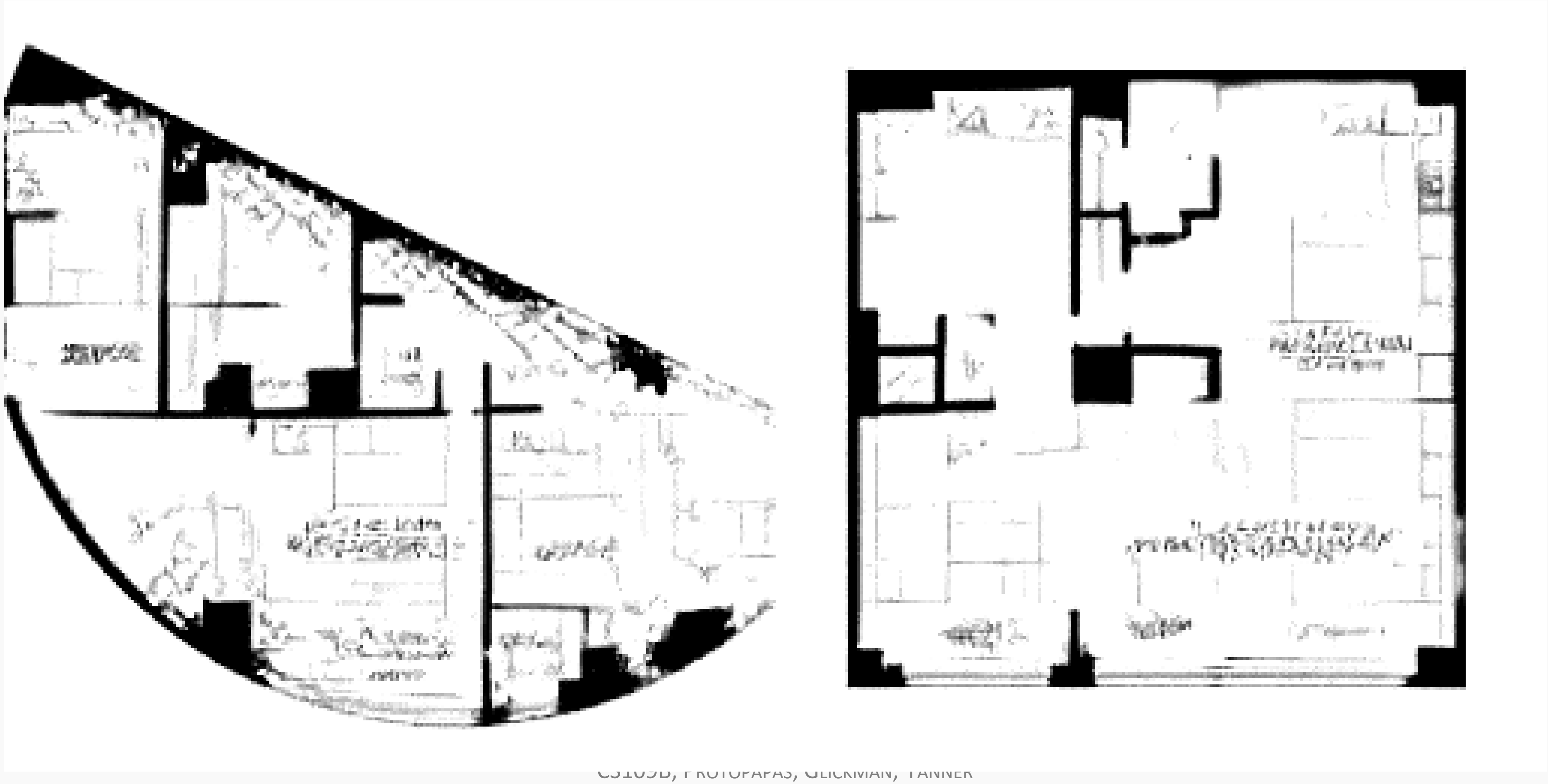
(b) MidiNet model 2



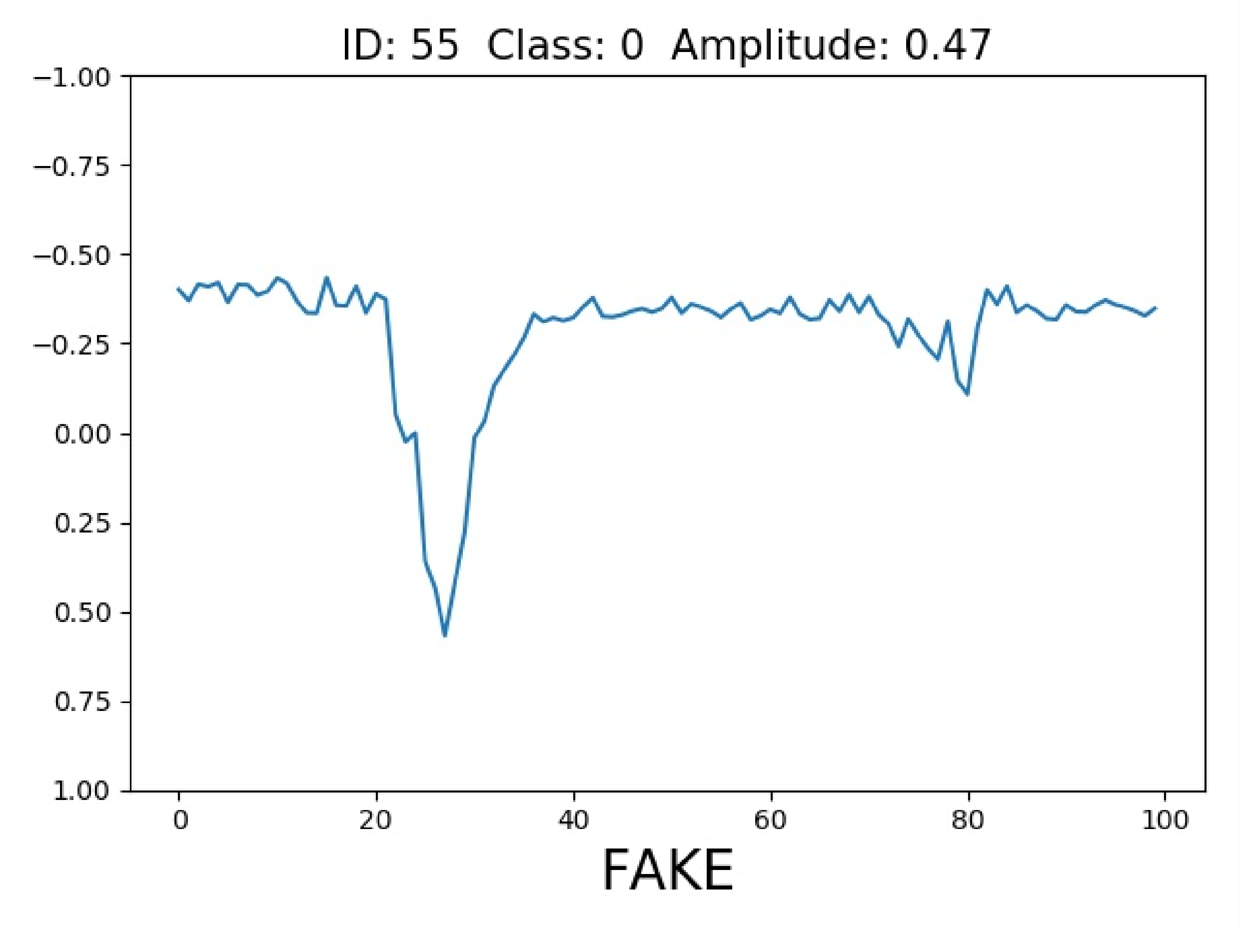
(c) MidiNet model 3

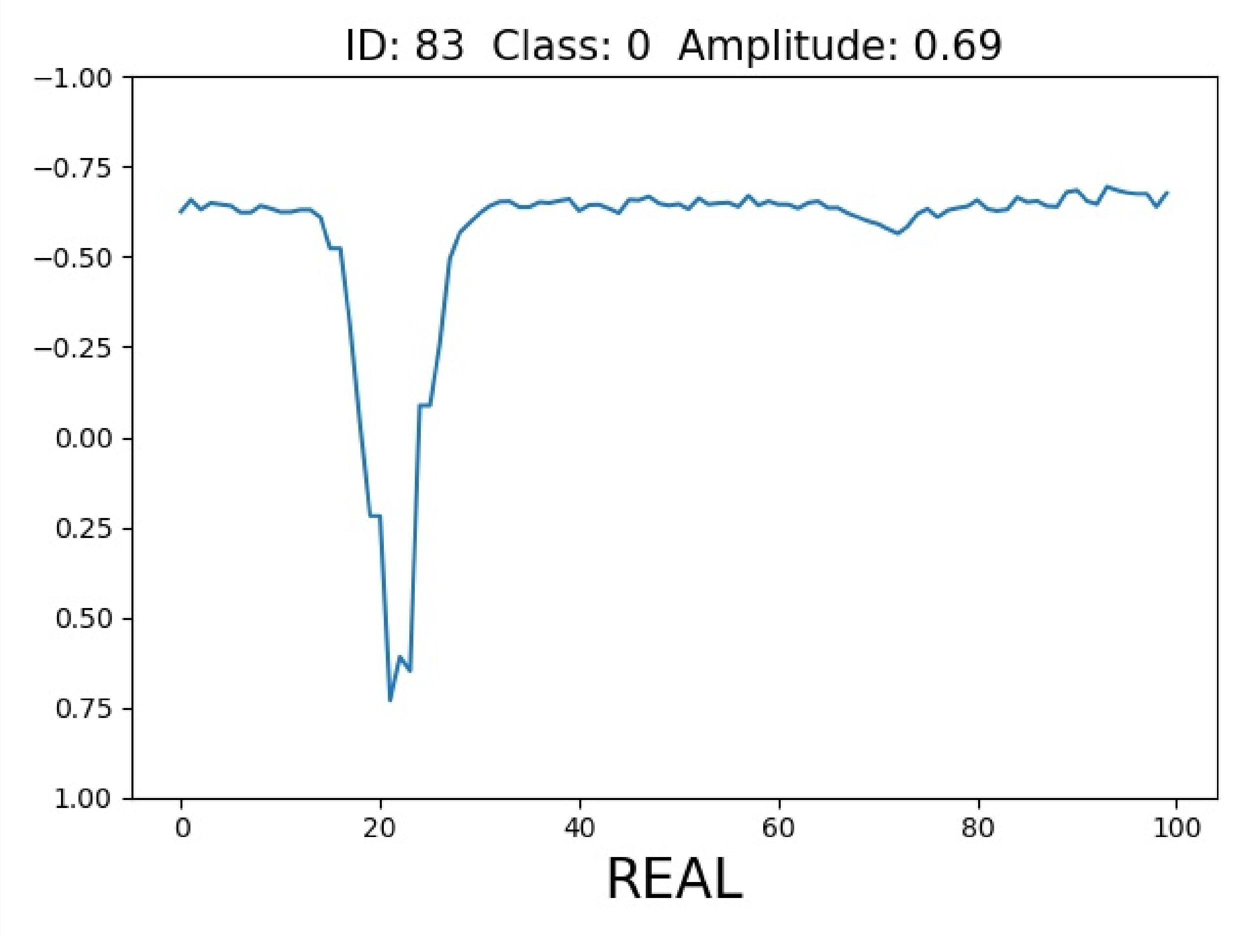
Figure 3. Example result of the melodies (of 8 bars) generated by different implementations of MidiNet.

Another use of generating new data is to give us ideas and options. Suppose we're planning a house. We can give the computer the space we have available, and its location. From this, the computer can can give us some ideas.



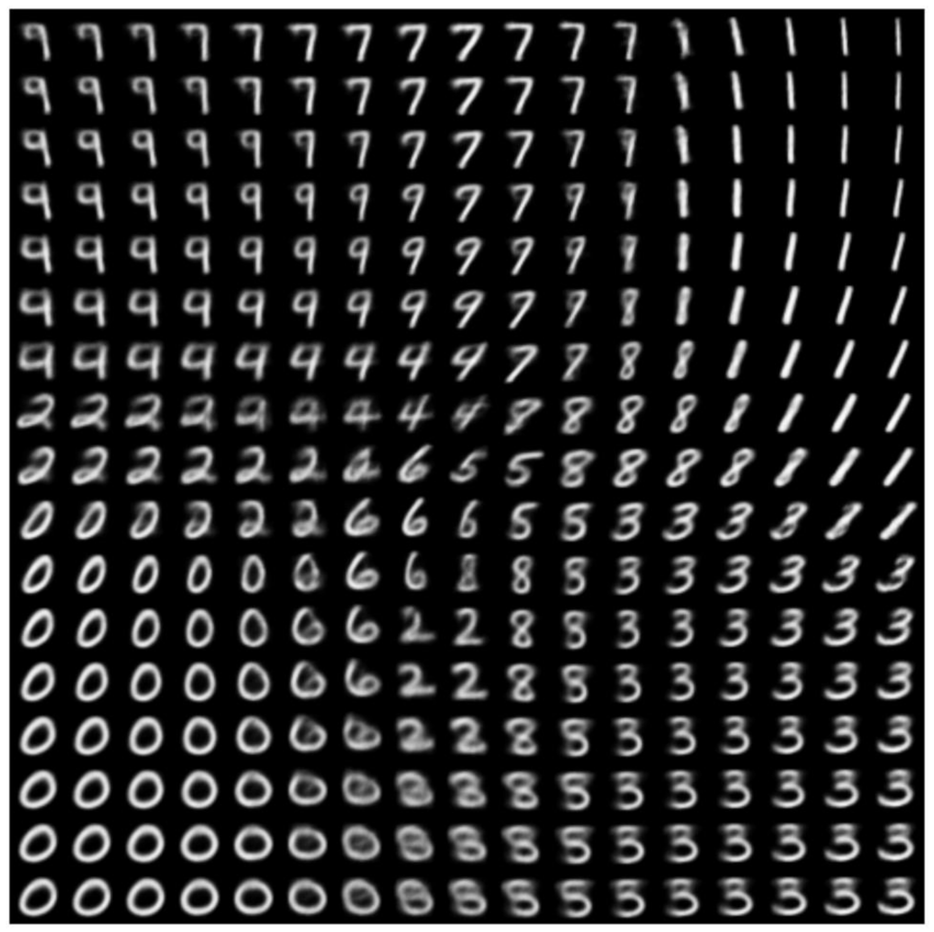
Big networks require big data, and getting high-quality, labeled data is difficult. If we're generating that data our selves, we can make as much of it as we like.





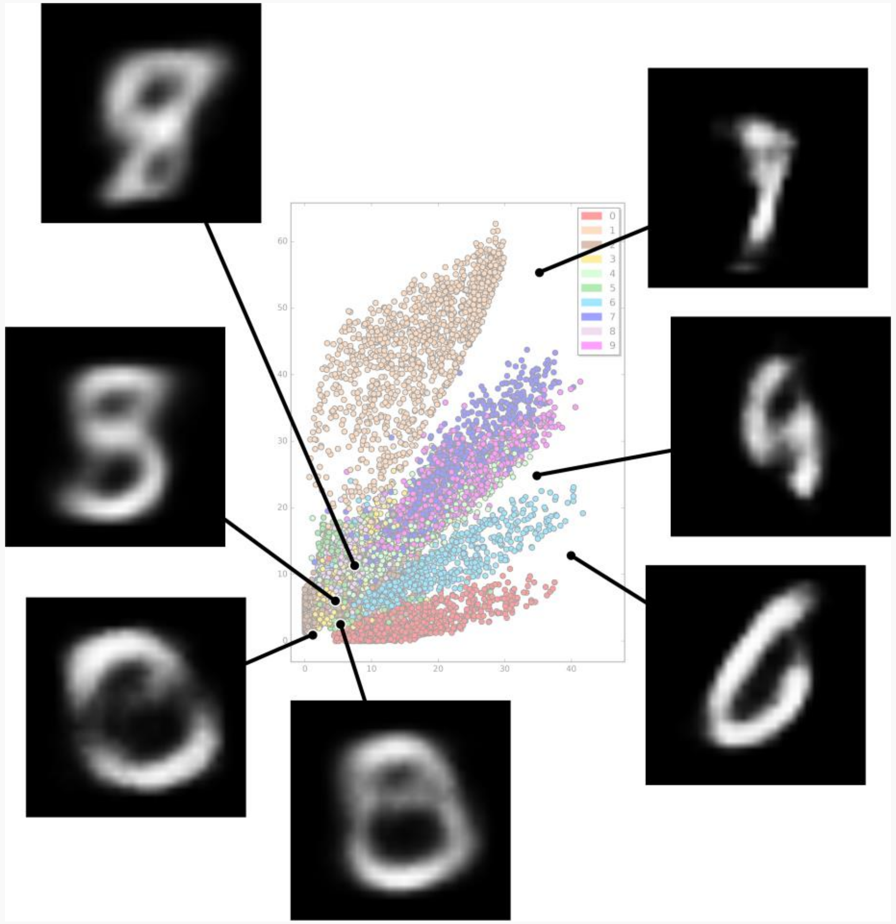
Generating Data

We saw how to generate new data with a AE in Lecture 12.



Problems with Autoencoders (from lecture 12)

- Gaps in the latent space
- Discrete latent space
- Separability in the latent space



Outline

Motivation for Variational Autoencoders (VAE)

Mechanics of VAE

Separability of VAE

The math behind everything

Generative models

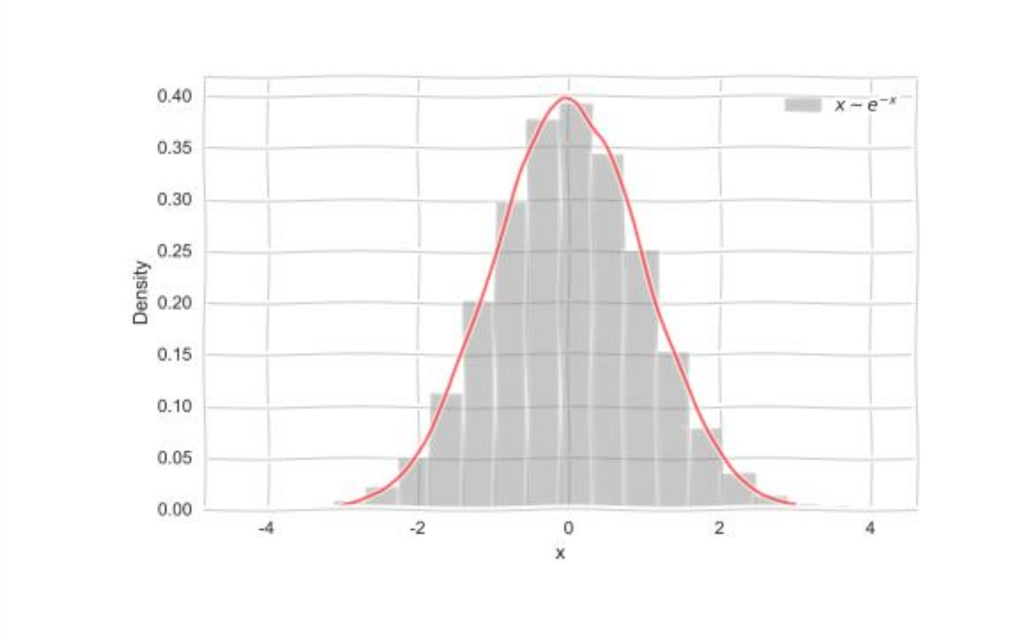
Generative models

Imagine we want to generate data from a distribution,

$$x \sim p(x)$$

e.g.

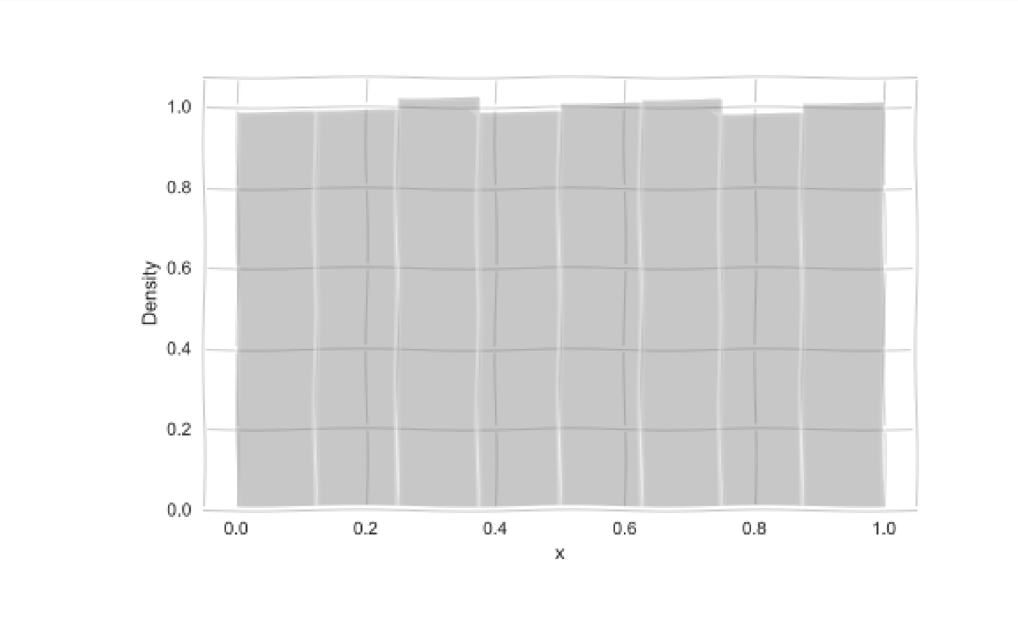
$$x \sim \mathcal{N}(\mu, \sigma)$$



Generative models

But how do we generate such samples?

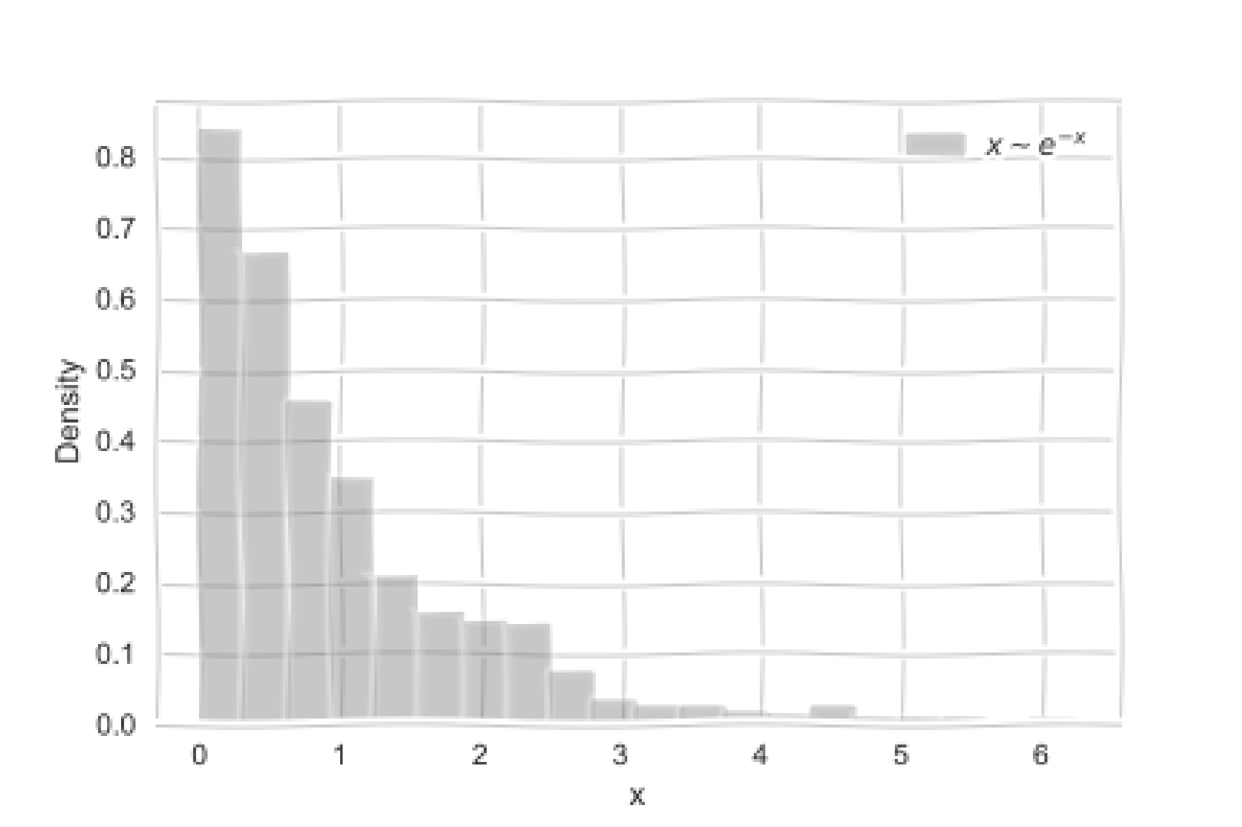
$$z \sim \mathrm{Unif}(0, 1)$$



Generative models

But how do we generate such samples?

$$z \sim \mathrm{Unif}(0,1) \quad \mathrm{x} = \ln \mathrm{z}$$



Generative models

In other words we can think that if we choose $\mathbf{z} \sim \boldsymbol{Uniform}$ then there is a mapping:

$$x = f(z)$$

such as:

$$x \sim p(x)$$

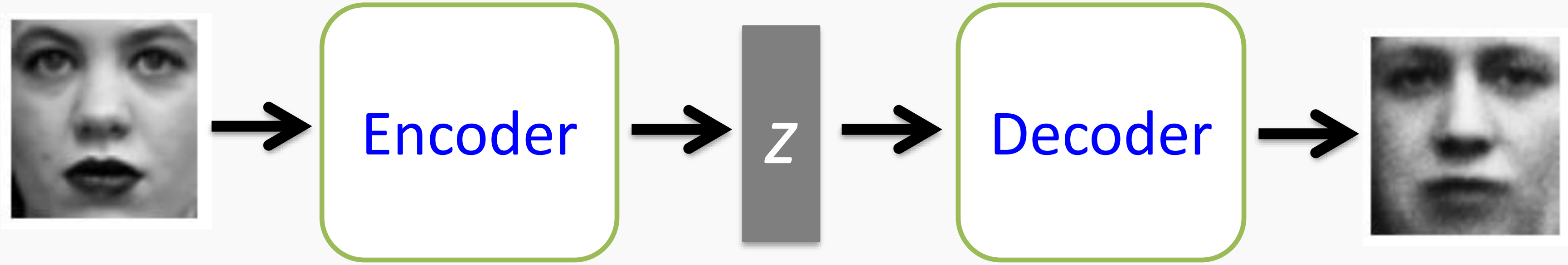
where in general f is some complicated function.

We already know that **Neural Networks are great in learning complex functions**.

$$\boxed{z \sim g(z)} \longrightarrow \boxed{x = f(z)} \longrightarrow \boxed{x \sim p(x)}$$

Traditional Autoencoders

In traditional autoencoders, we can thing of encoder and decoders as some function mapping.

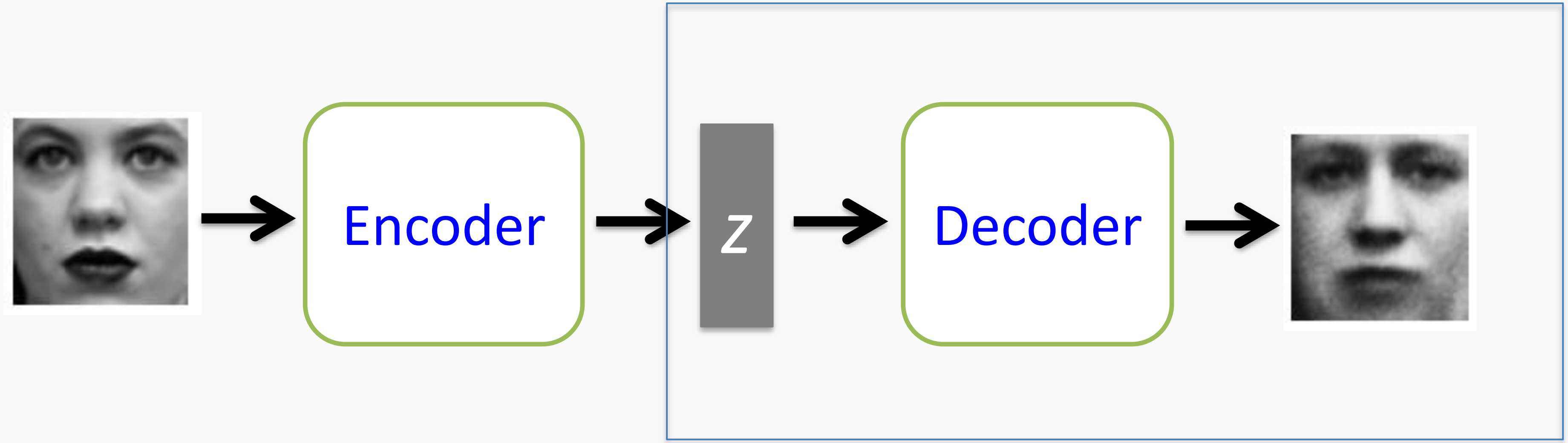


$z = h(x)$

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

Variational Autoencoders

To go to variational autoencoders, we need to first add some stochasticity and think of it as a probabilistic modeling.



Variational Autoencoders

Sample from $g(z)$
e.g. Standard Gaussian

z → Decoder $P(\hat{x}|z)$ → 

$z \sim g(z)$ $\hat{x} = f(z)$ $\hat{x} \sim P(x|z)$

Variational Autoencoders

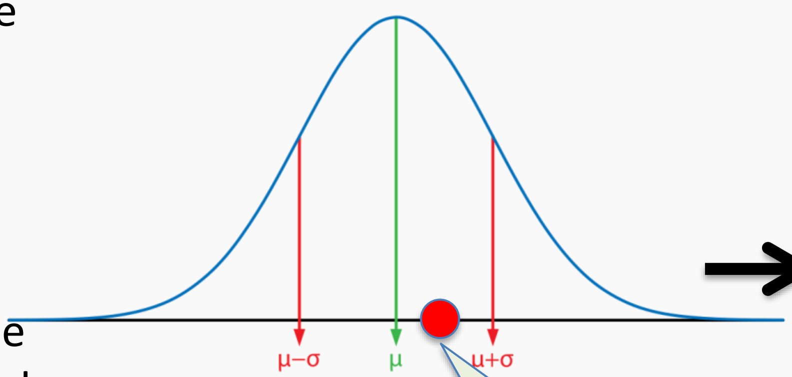
Traditional AE

→ Encoder → z

→ Encoder → z_1, z_2

Consider this to be the mean of a normal μ

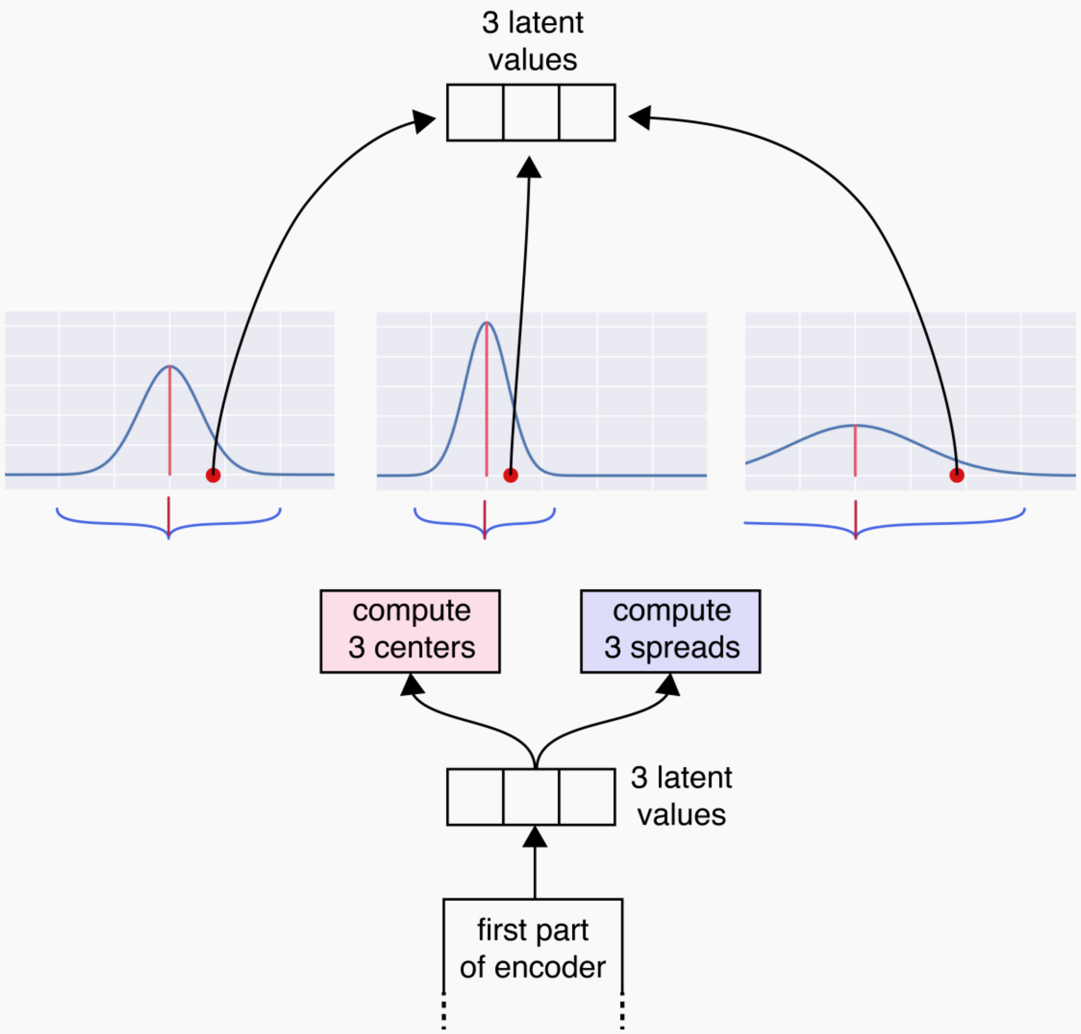
Consider this to be the std of a normal σ



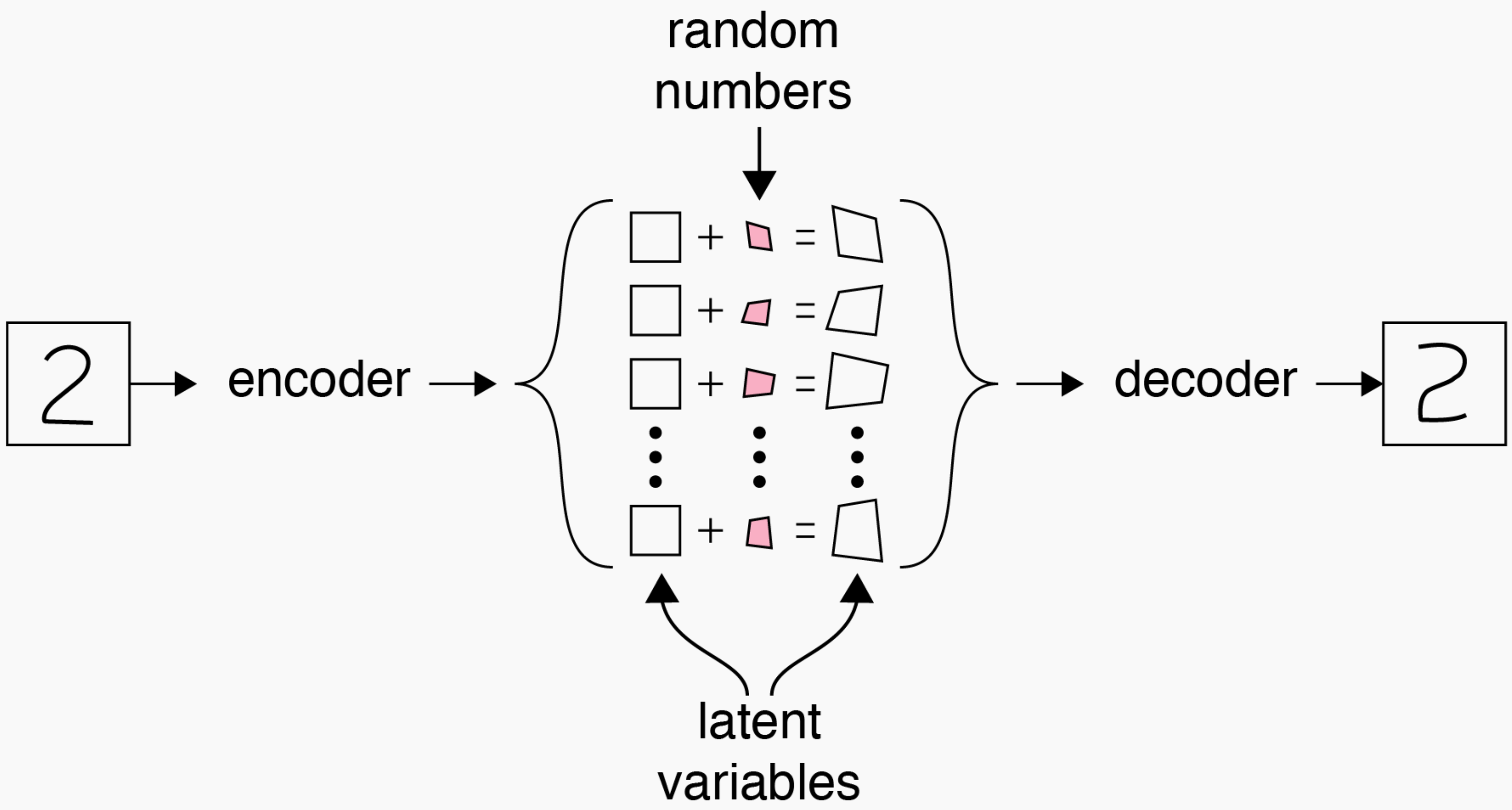
Randomly chosen value Latent value, z

→ Decode

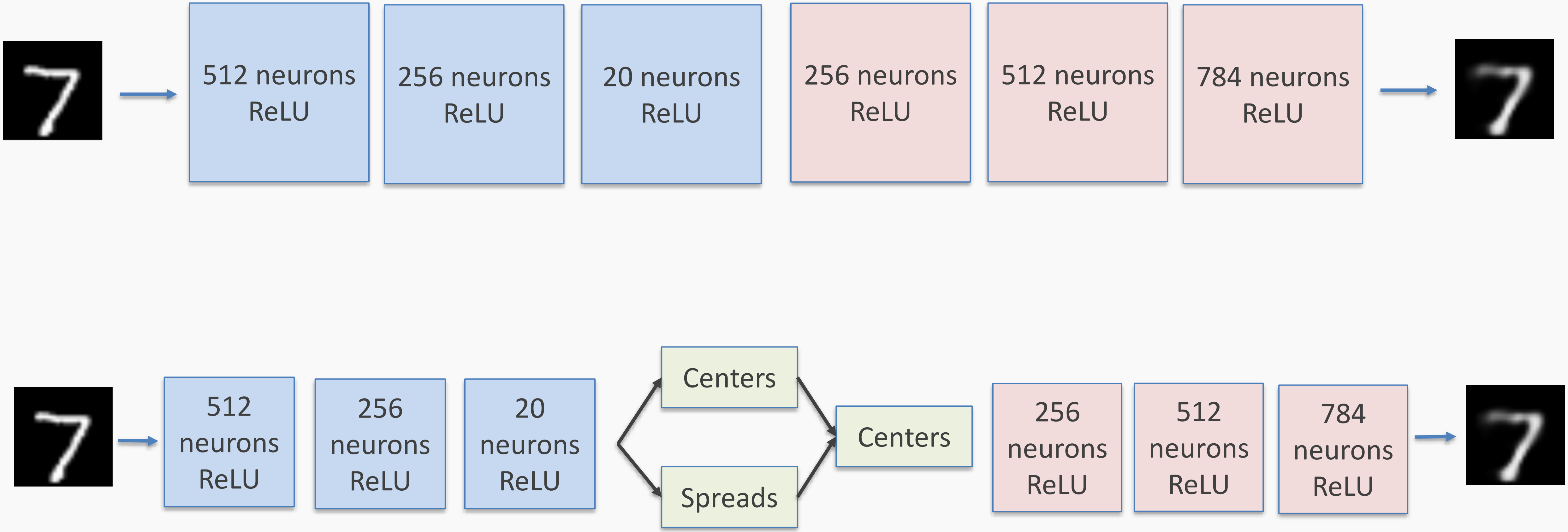
Variational Autoencoders



Variational Autoencoders



Variational Autoencoders



Outline

Motivation for Variational Autoencoders (VAE)

Mechanics of VAE

Separability of VAE

The math behind everything

Generative models

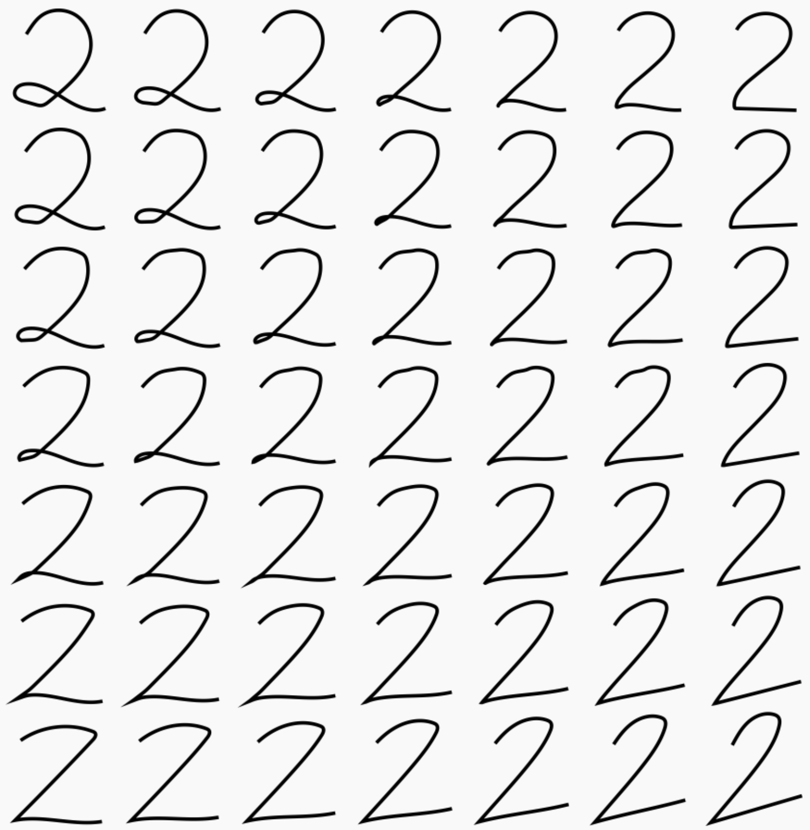
Separability in Variational Autoencoders

Separability is not only between classes but we also want similar items in the same class to be near each other.

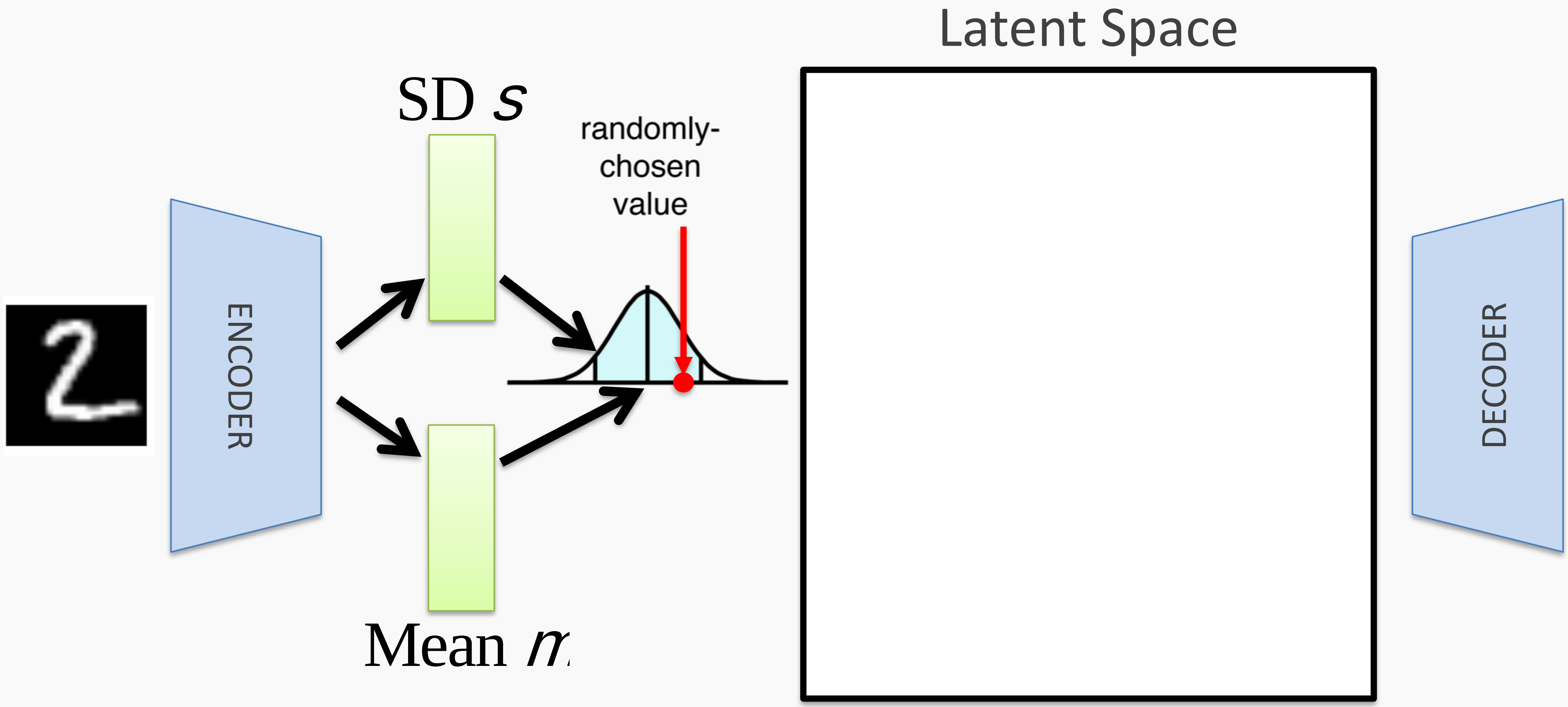
This is similar to word encoding we have talked in the previous lecture.

For example, there are different ways of writing “2”, we want similar styles to end up near each other.

Let’s examine VAE, there is something magic happening once we add stochasticity in the latent space.

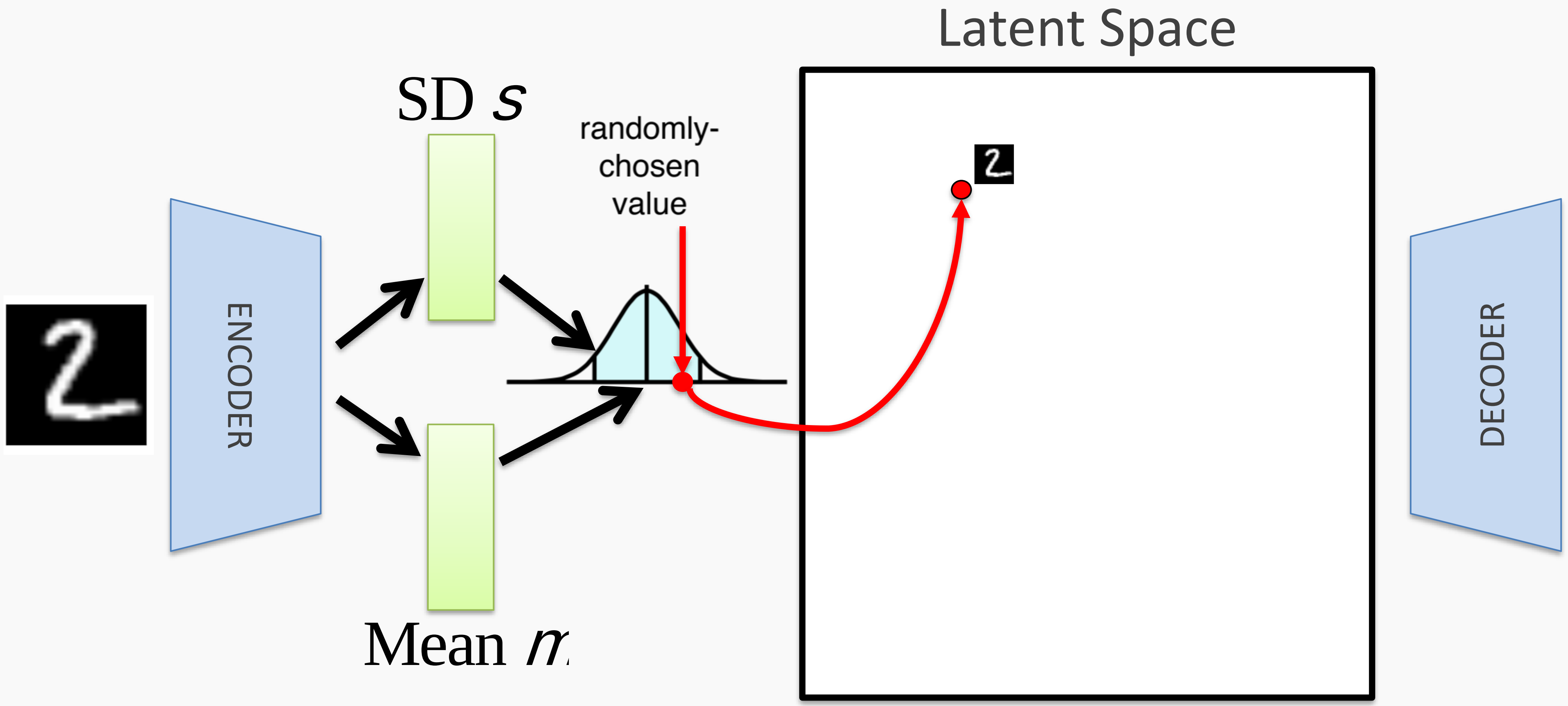


Separability in Variational Autoencoders



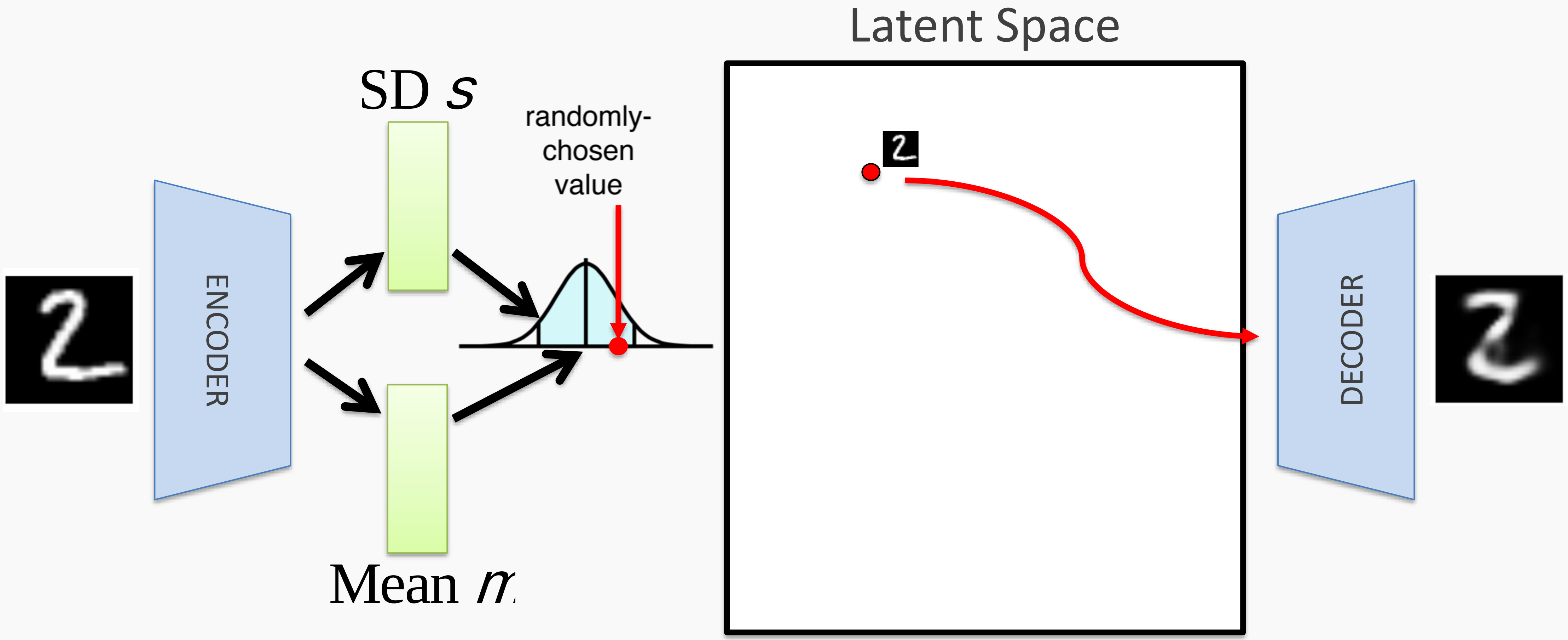
Encode the first sample (a “2”) and find μ_1, σ_1

Separability in Variational Autoencoders



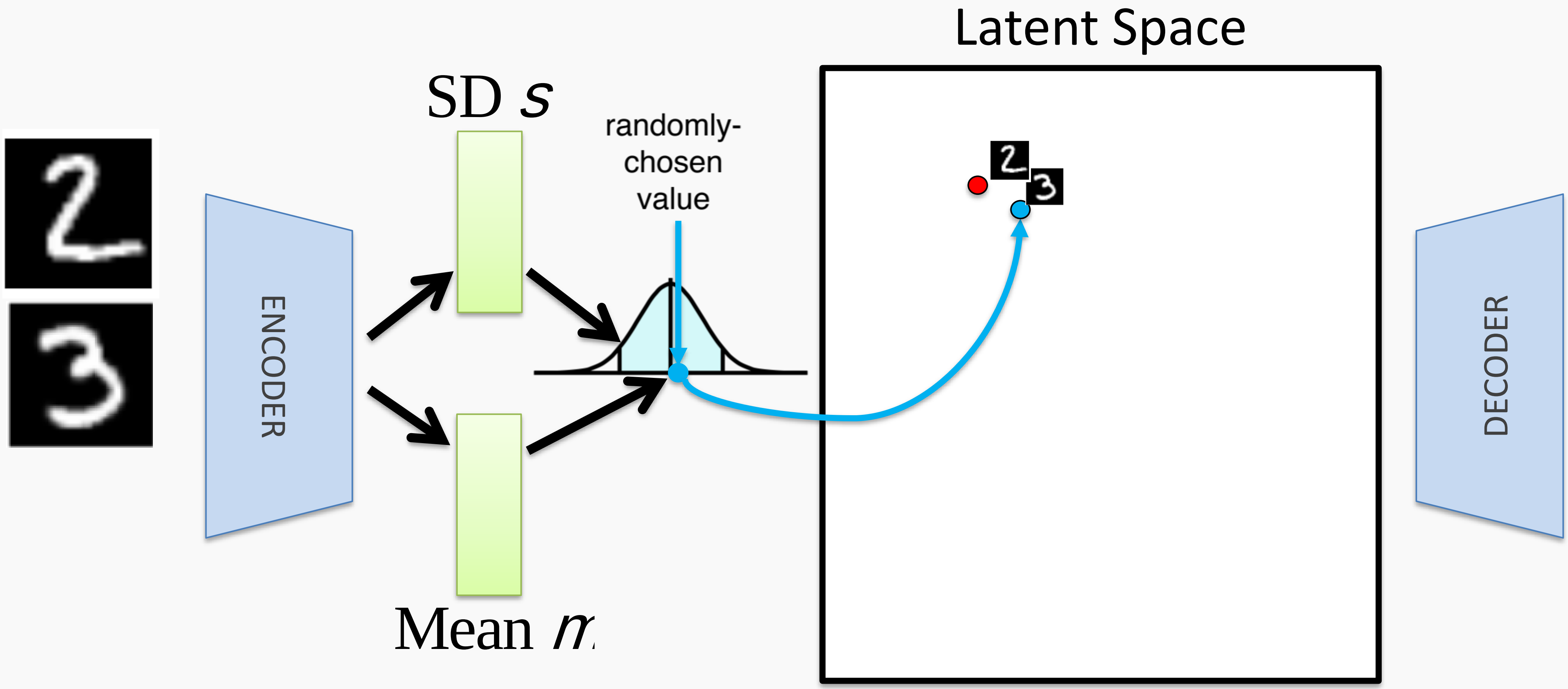
Sample $z_1 \sim N(\mu_1, \sigma_1)$

Blending Latent Variables



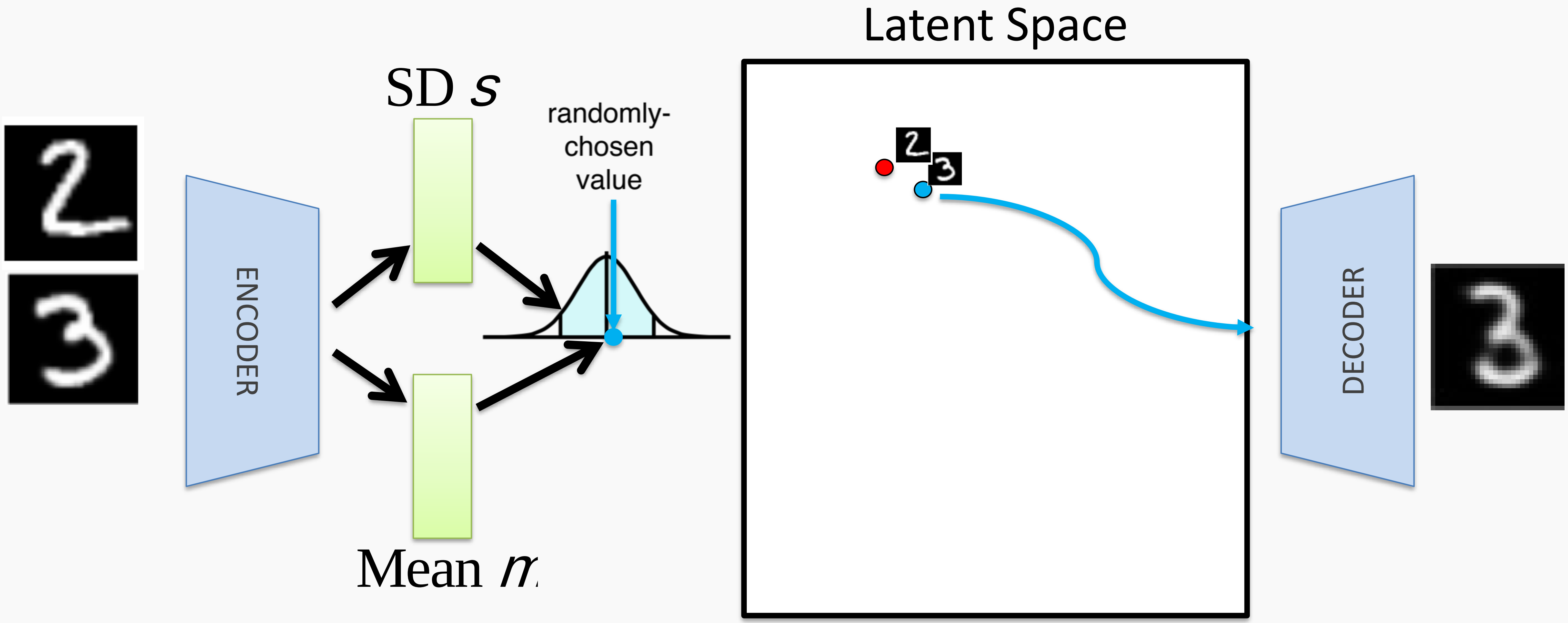
Decode to $\hat{x}_1$

Separability in Variational Autoencoders



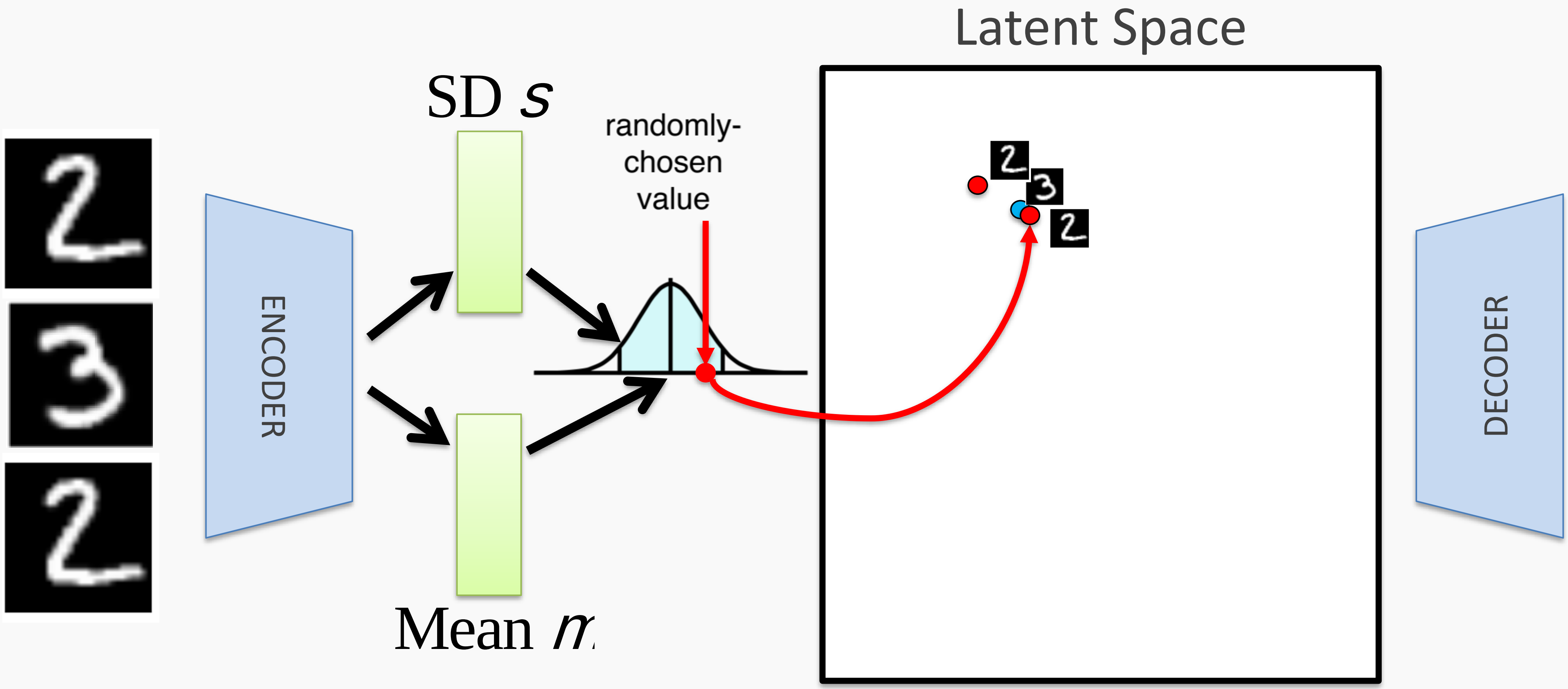
Encode the second sample (a "3") find μ_2, σ_2. Sample $z_2 \sim N(\mu_2, \sigma_2)$

Separability in Variational Autoencoders



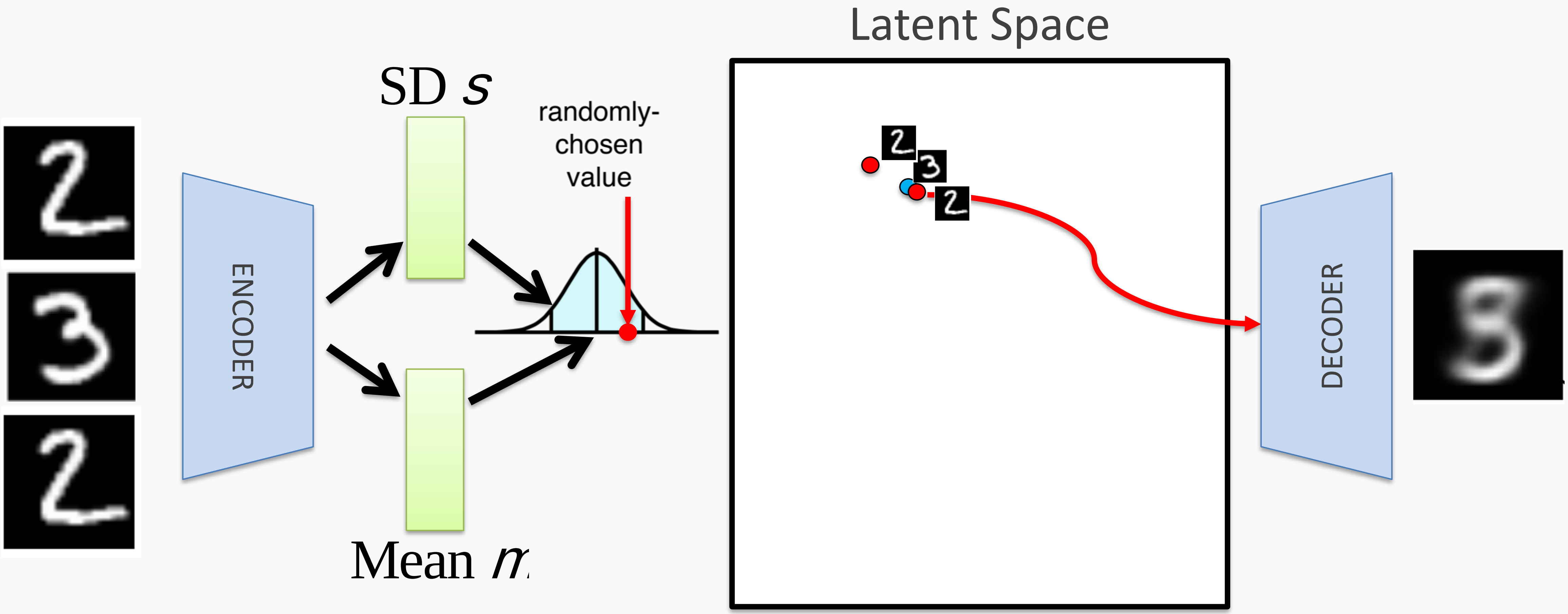
Decode to $\hat{x}_2$

Separability in Variational Autoencoders



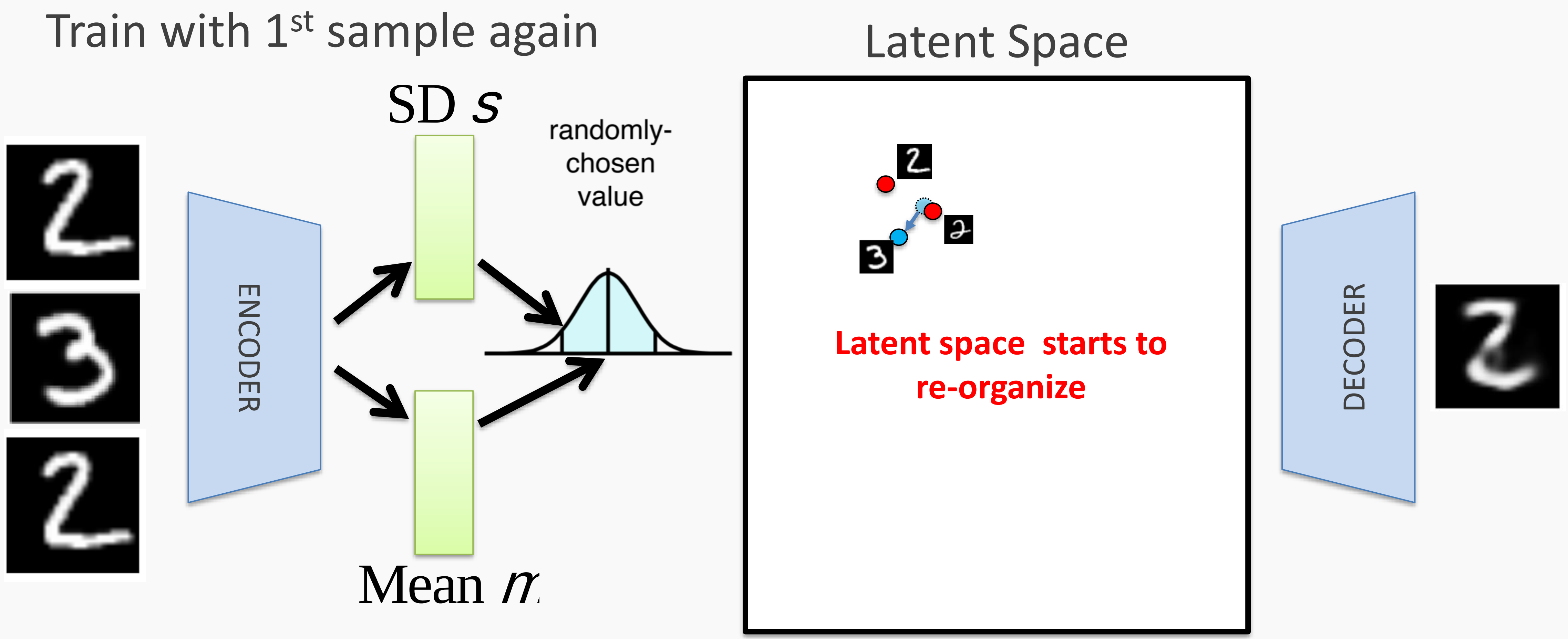
Train with the first sample (a "2") again and find μ_1, σ_1. However $z_1 \sim N(\mu_1, \sigma_1)$ **will not be the same**. *It can happen to be close to the "3" in latent space.*

Separability in Variational Autoencoders



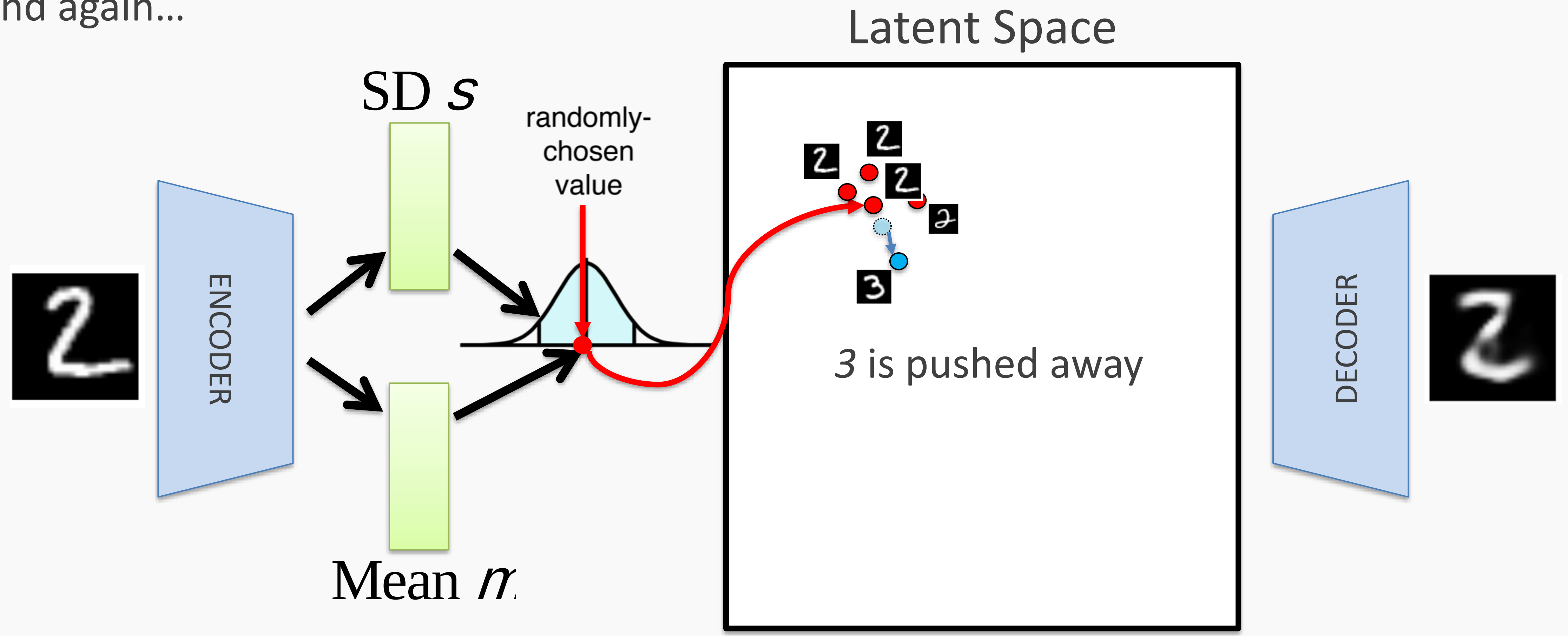
Decode to $\hat{x}_1$. *Since the decoder only knows how to map from latent space to* $\hat{x}$ *space, it will return a "3".*

Separability in Variational Autoencoders



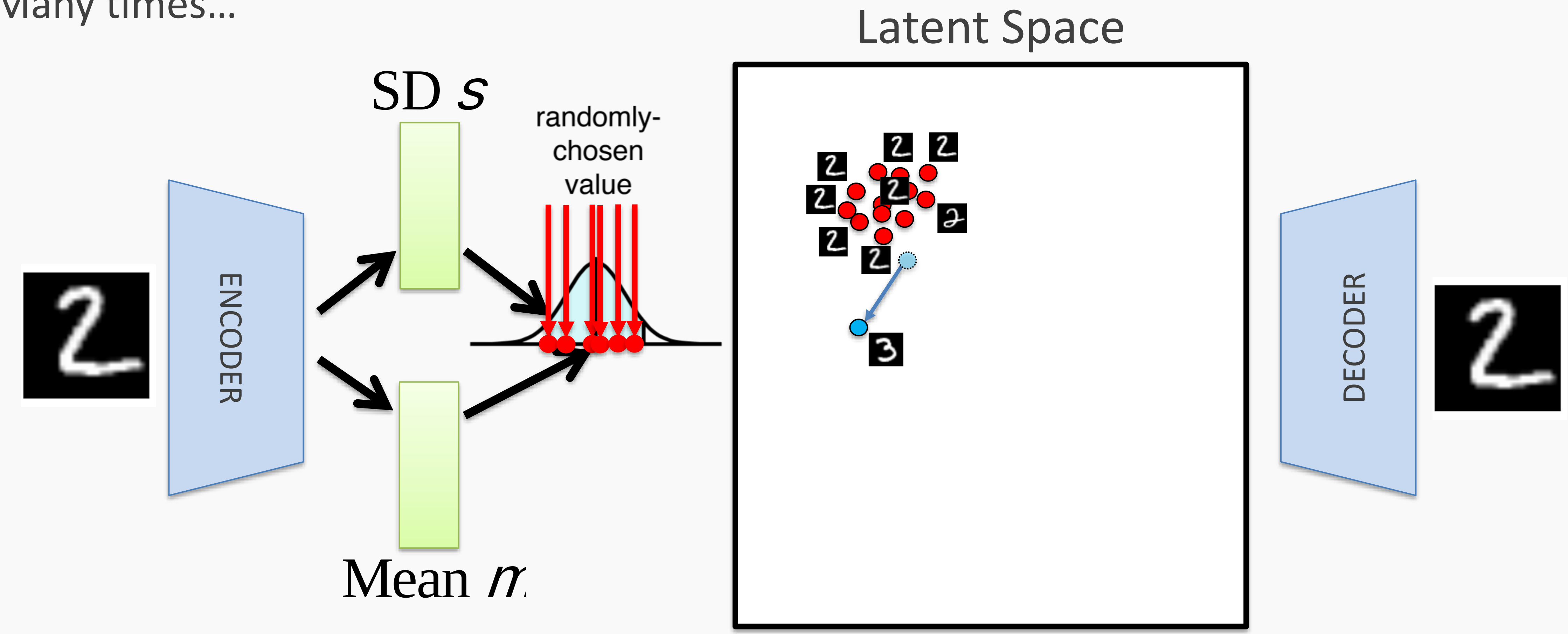
Separability in Variational Autoencoders

And again…



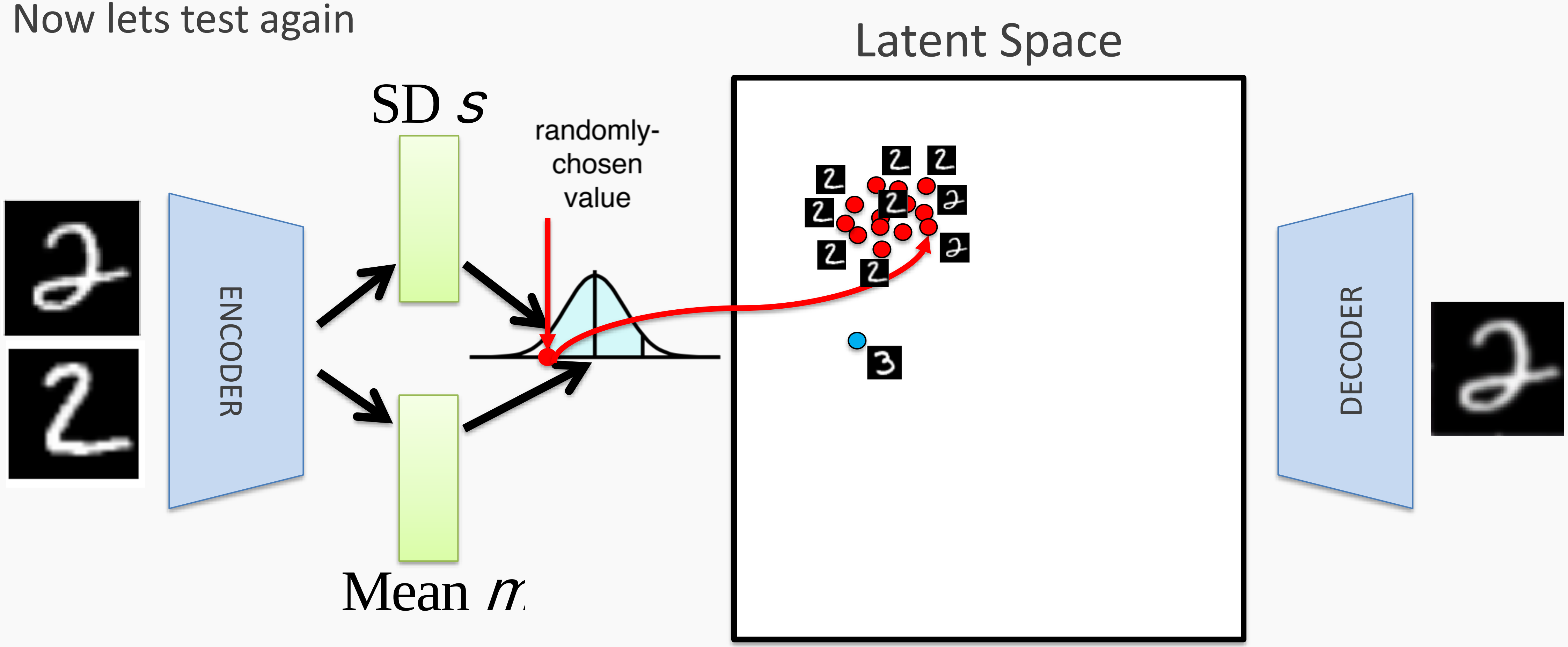
Separability in Variational Autoencoders

Many times...



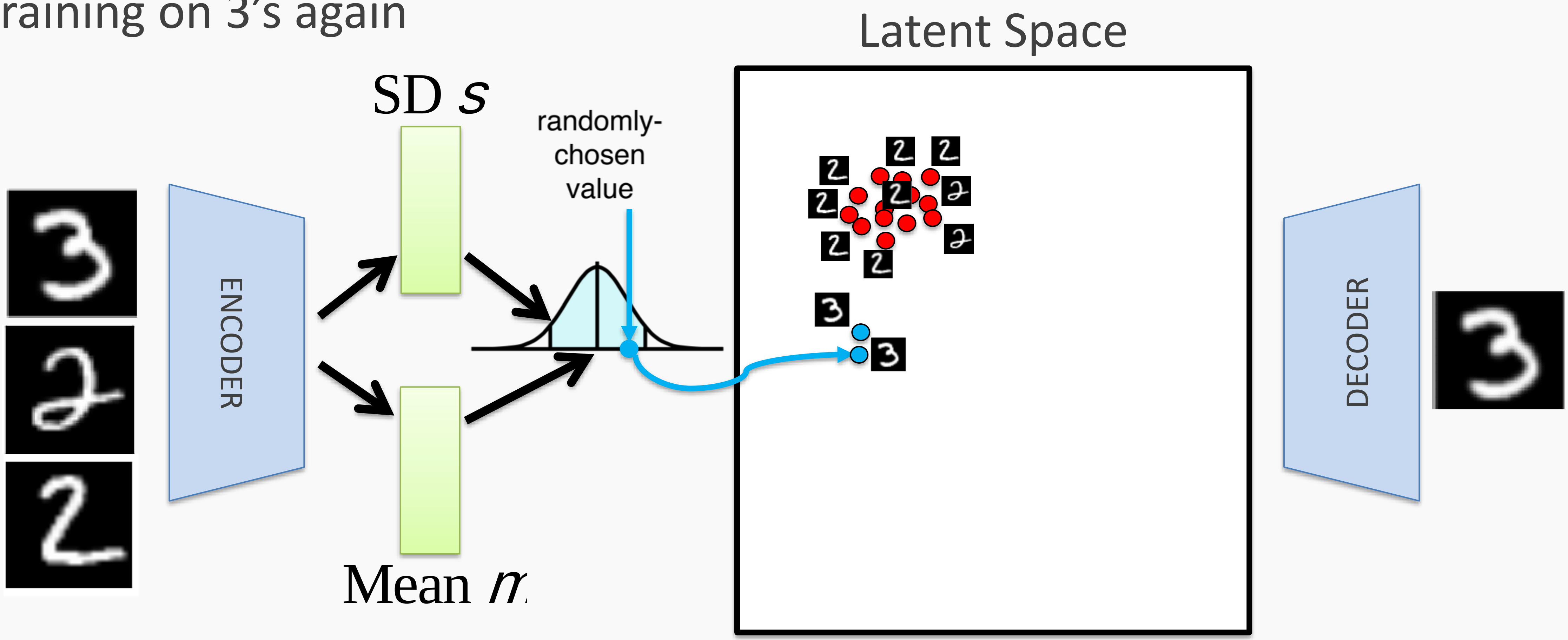
Separability in Variational Autoencoders

Now lets test again



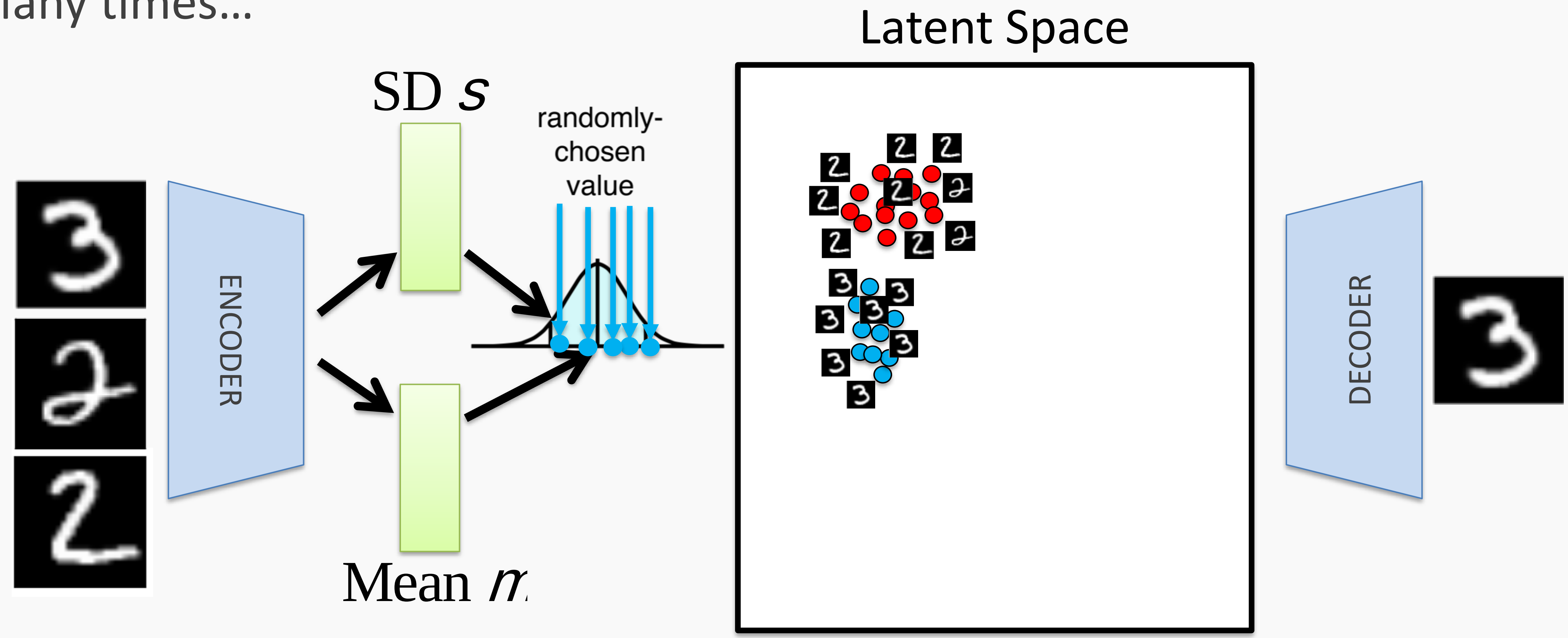
Separability in Variational Autoencoders

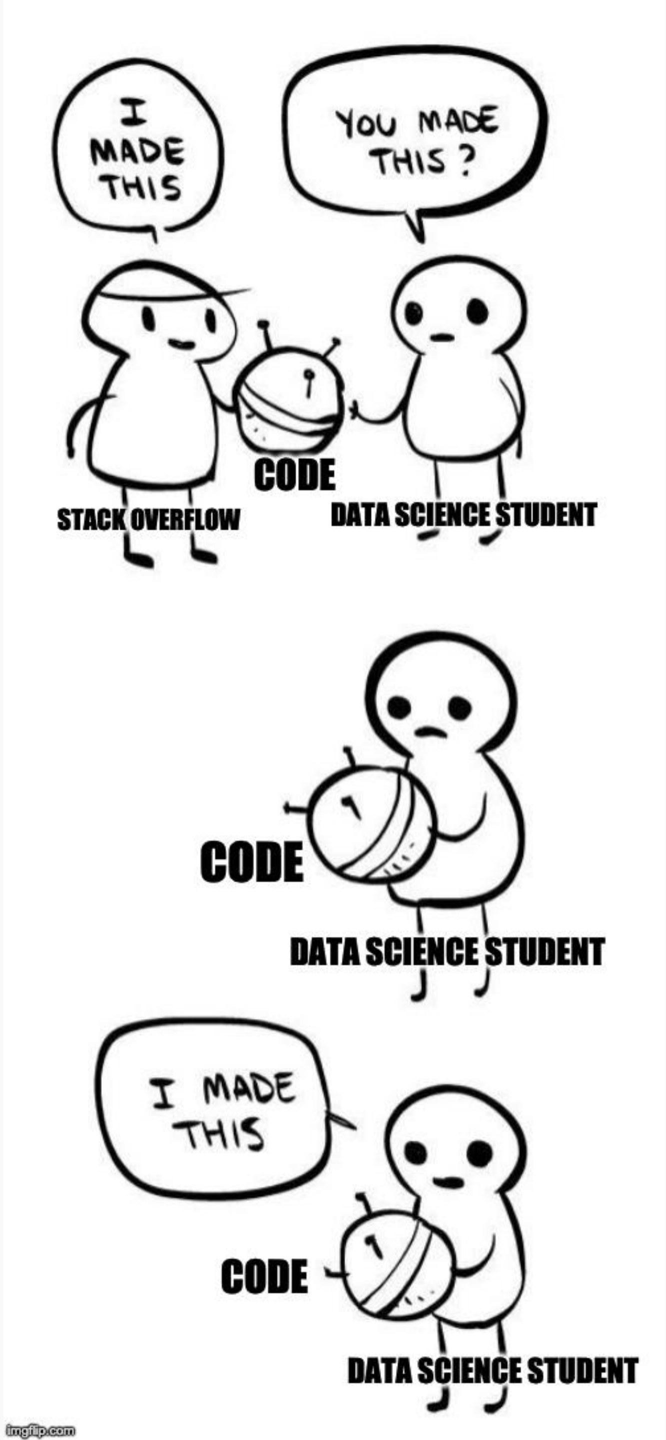
Training on 3's again



Separability in Variational Autoencoders

Many times…




I MADE THIS
YOU MADE THIS ?
CODE
STACK OVERFLOW
DATA SCIENCE STUDENT
CODE
DATA SCIENCE STUDENT
I MADE THIS
CODE
DATA SCIENCE STUDENT
imgflip.com

Outline

Motivation for Variational Autoencoders (VAE)

Mechanics of VAE

Separability of VAE

The math behind everything

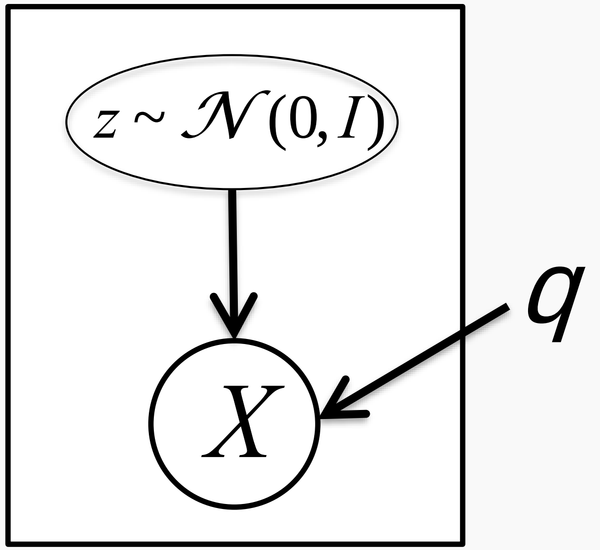
Generative models

Training a VAE using the likelihood

Neural network

$$p_\theta(x) = \int_z p_\theta\left(x \middle| z\right) \boxed{p_\theta(z)} dz$$

Difficult to approximate in high dim through sampling



For most *z* values *p*(*x*|*z*) is close to 0

Training

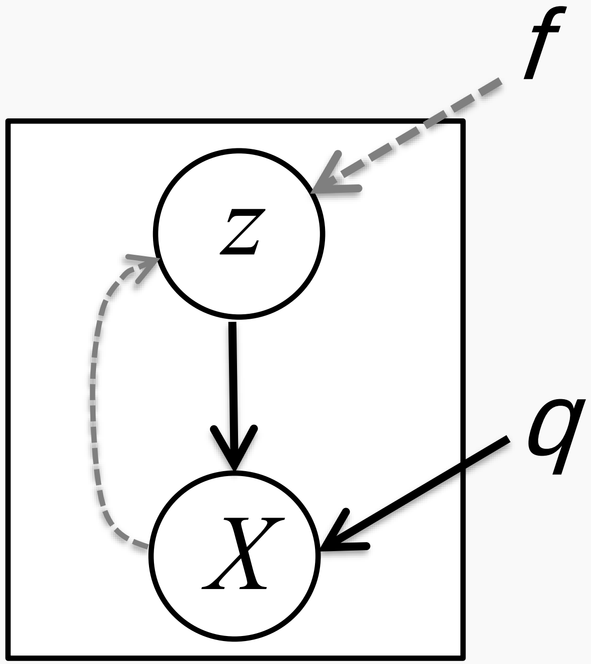
VAE Likelihood

Another neural net

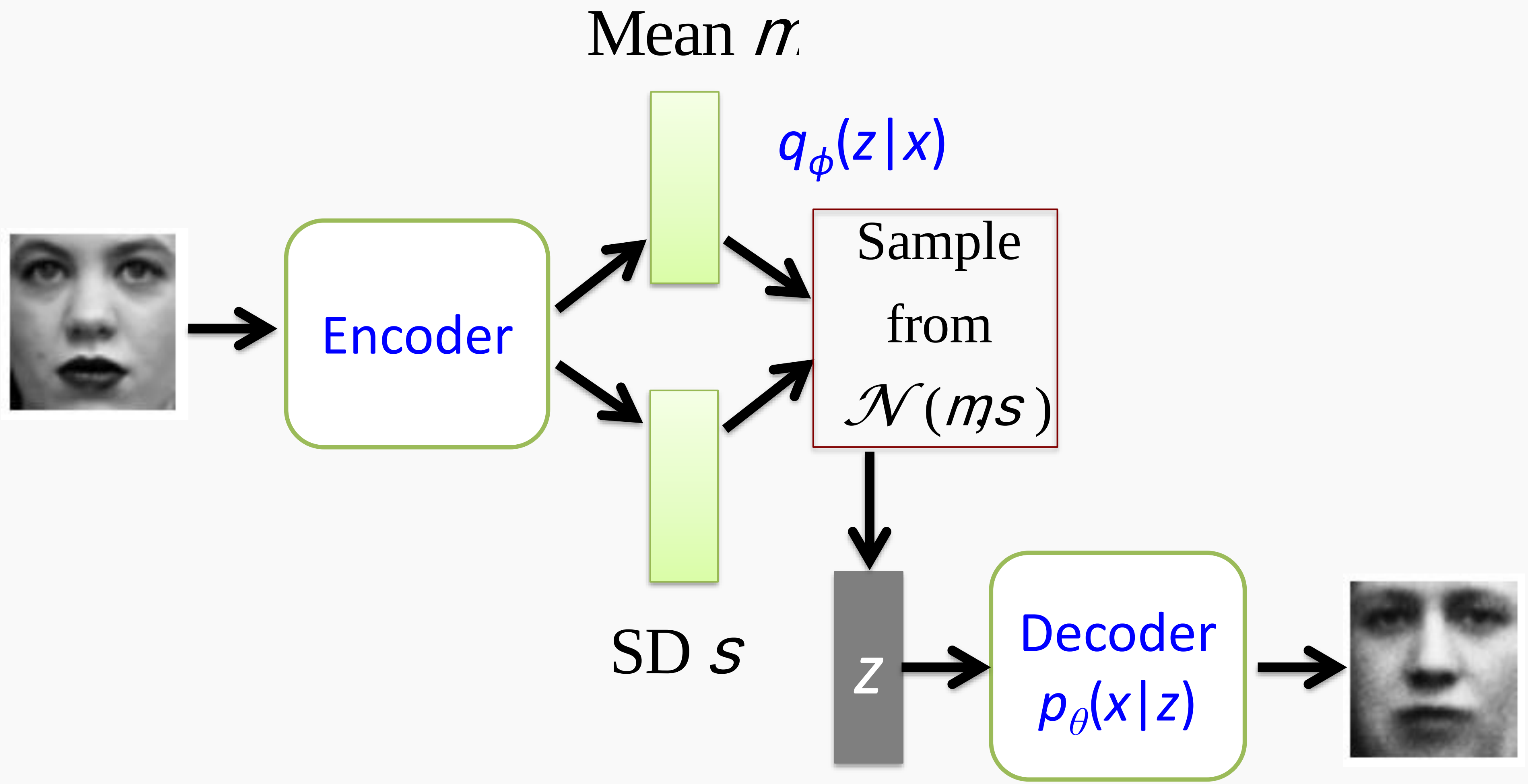
$$p_\theta(x) = \int_z p_\theta(x|z)\, \boxed{q_\phi(z|x)}\, dz$$

Proposal distribution:

Likely to produce values of *x* for which *p*(*x*|*z*) is non-zero



VAE Architecture



VAE Loss

Reconstruction Loss

$$- \mathbf{E}_{z \sim q_f(z|x)} \log\left(p_q(x|z)\right)$$

VAE Loss

Reconstruction Loss

Proposal distribution should resemble a Gaussian

$$- \mathbf{E}_{z \sim \boxed{q_f(z|x)}} \log\left(p_q(x|z)\right) + KL\left(\boxed{q_f(z|x)} \,\|\, p_q(z)\right)$$

VAE Loss

Reconstruction Loss

Proposal distribution should resemble a Gaussian

$$- \mathbf{E}_{z \sim \boxed{q_f(z|x)}} \log\left(p_q(x|z)\right) + KL\left(\boxed{q_f(z|x)} \| p_q(z)\right)$$

$$3 \; - \log p_q(x)$$

Variational upper bound on loss we care about!

```python
def make_VAE(input_dim, latent_dim, bottleneck_dim, reg=0.01, dropout_rate=0.0):
    flat_dim = latent_dim[0] * latent_dim[1] * latent_dim[2]

    x = layers.Input(shape=input_dim, batch_size=batch_size)
    xe = ConvEncoder(input_shape=input_dim, dropout_rate=dropout_rate)(x)
    xe = layers.Flatten()(xe)
    z_mean = layers.Dense(bottleneck_dim[0], activation='linear')(xe)
    z_log_var = layers.Dense(bottleneck_dim[0], activation='linear')(xe)
    z = Sampling()([z_mean, z_log_var])
    xr = layers.Dense(flat_dim, activation='relu')(z)
    xr = layers.Reshape(latent_dim)(xr)
    xr = ConvDecoder(input_shape=latent_dim, dropout_rate=dropout_rate)(xr)

    encoder = models.Model(inputs=x, outputs=z)
    VAE = models.Model(inputs=x, outputs=xr)

    if reg > 0.0:
        kl_loss = - 0.5 * tf.reduce_mean(z_log_var - tf.square(z_mean) - tf.exp(z_log_var) + 1)
        VAE.add_loss(reg * kl_loss)

    opt = optimizers.Adam(learning_rate=1e-4)
    loss = losses.MeanSquaredError()

    VAE.compile(optimizer=opt, loss=loss)
    VAE.summary()

    return VAE, encoder
```

Training VAE

Apply stochastic gradient descent (SGD)

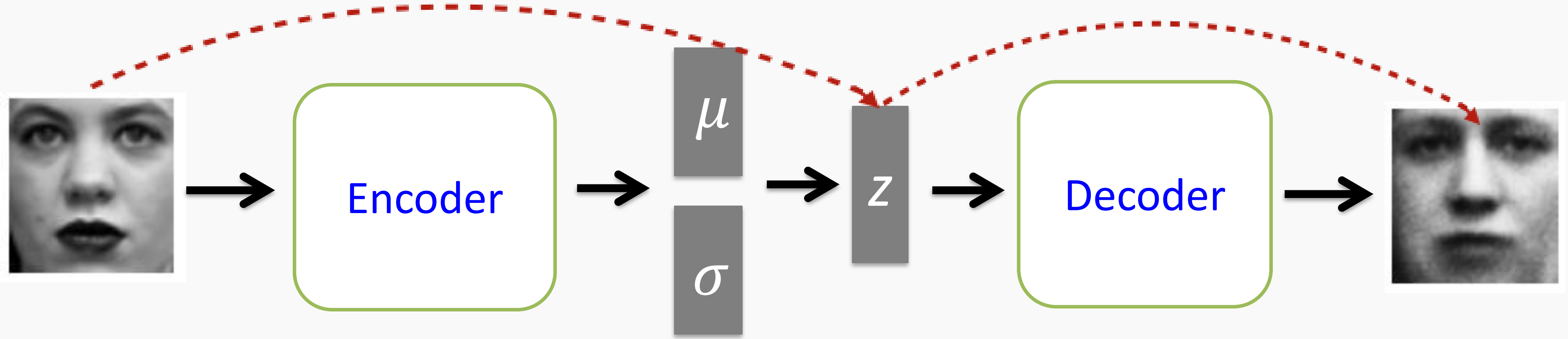
Problem:

Sampling step not differentiable

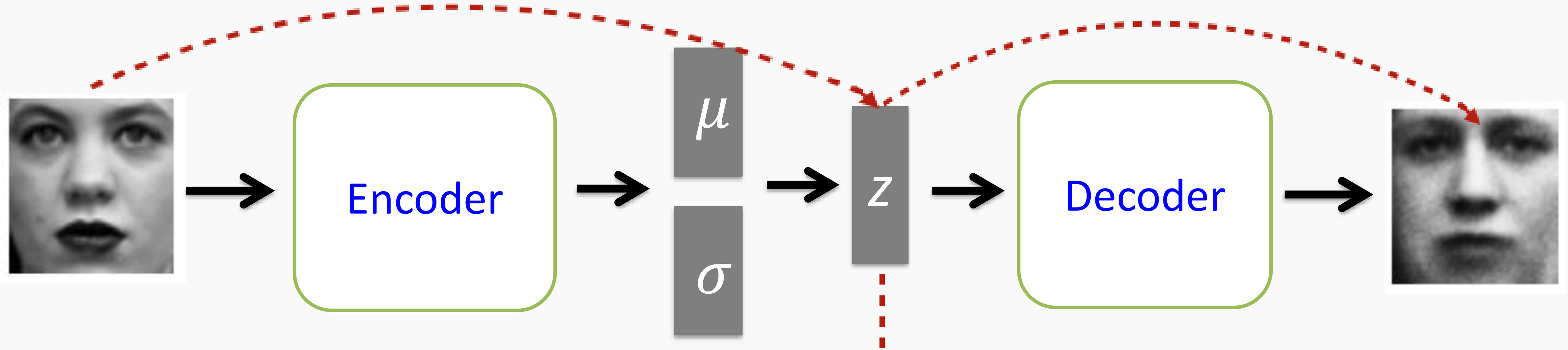
Use a re-parameterization trick

– Move sampling to input layer, so that the sampling step is independent of the model

Reparametrization Trick

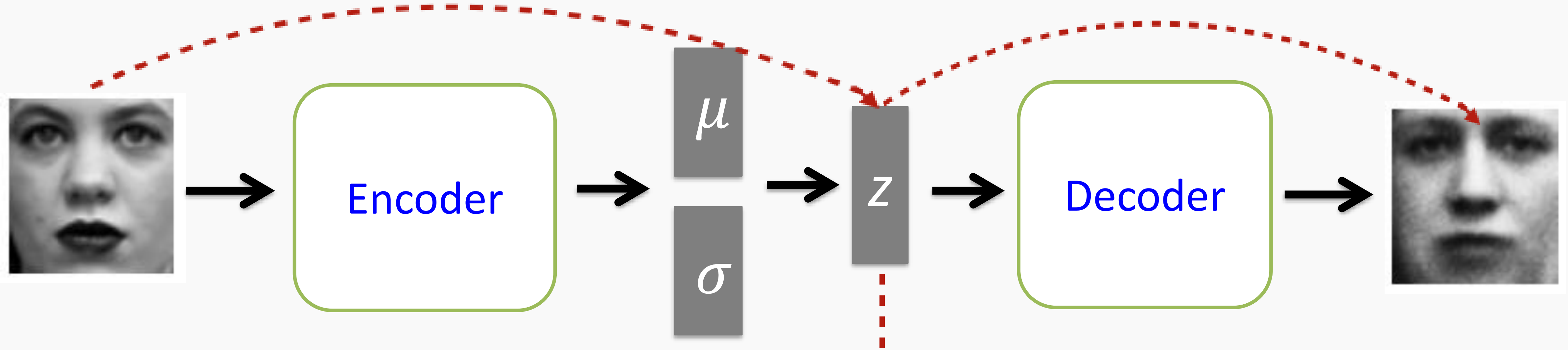


Reparametrization Trick



$$\boxed{Z = \mu + \varepsilon \circ \sigma}$$

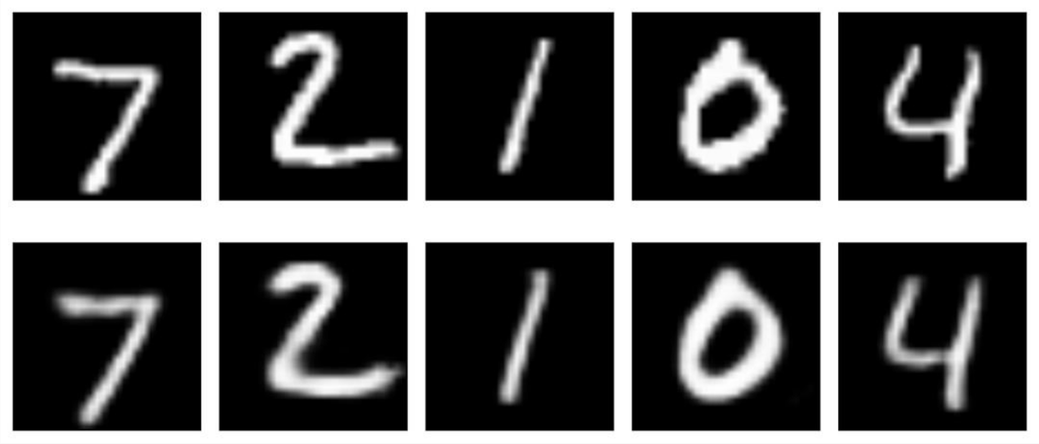
Reparametrization Trick

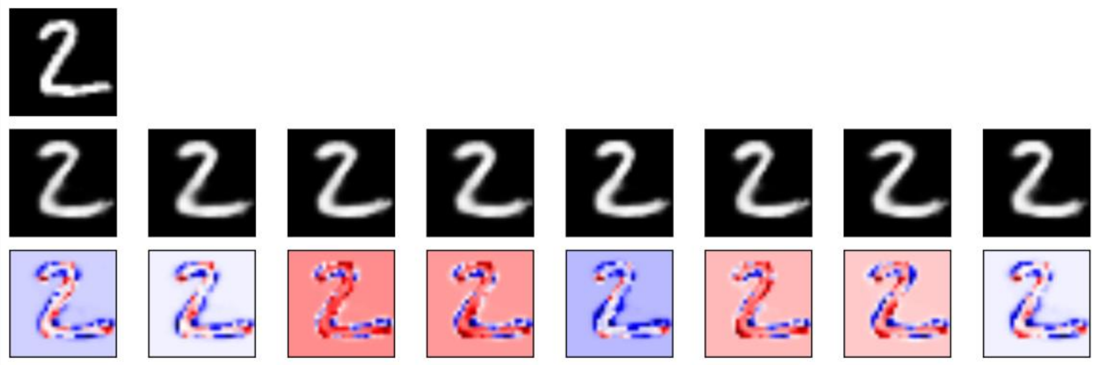


$$Z = \mu + \varepsilon \circ \sigma$$

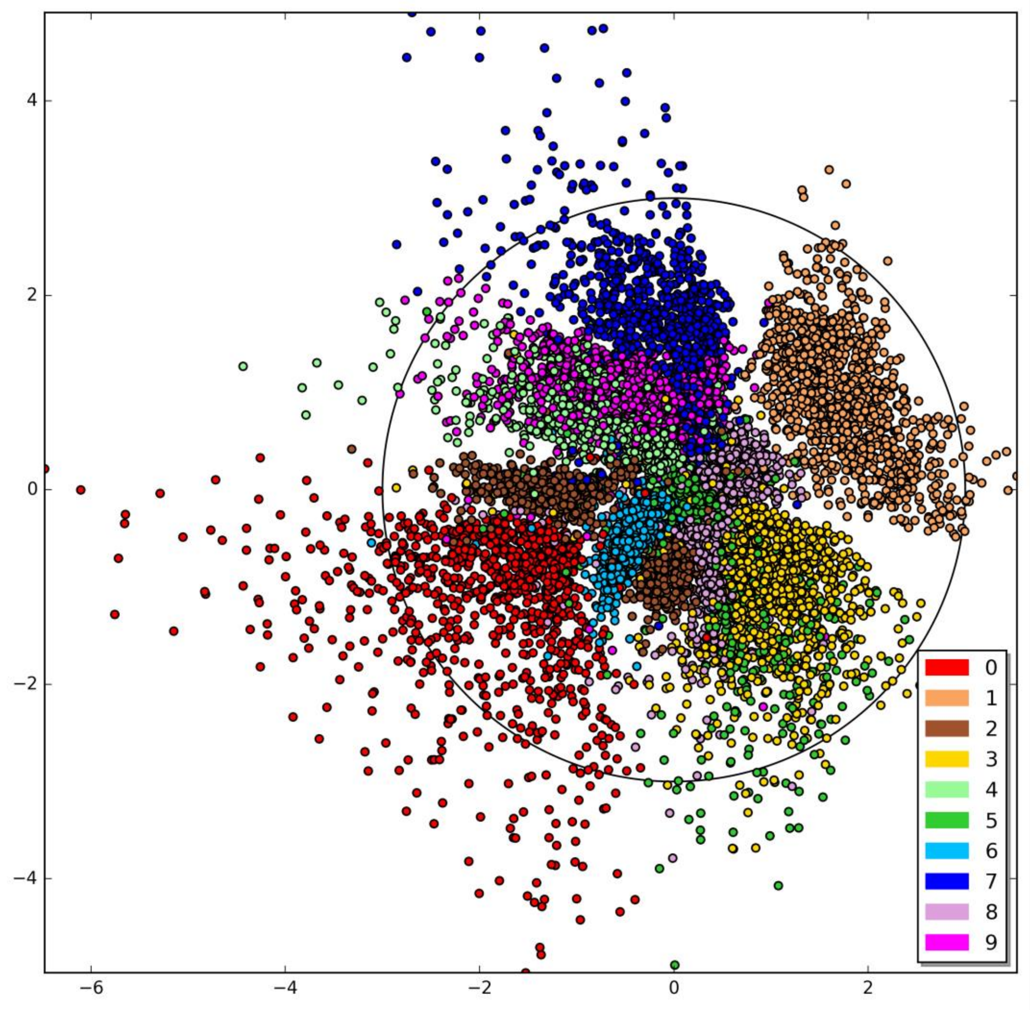
$$\varepsilon \sim N(0, I)$$

Training VAE





Parameter space VAE



Parameter space VAE