CSCI 599: Deep Learning and its Applications

Lecture 6

Spring 2019
Shao-Hua Sun

Today's agenda

- **Part 1: Deep Learning Framework**
  - TensorFlow
  - PyTorch
- **Part 2: Cloud Service**
  - Google Cloud
  - Amazon Web Services

Today's agenda

- **Part 1: Deep Learning Framework** (Shao-Hua Sun)
  - TensorFlow (Shao-Hua Sun)
  - PyTorch
- **Part 2: Cloud Service**
  - Google Cloud
  - Amazon Web Services

Today's agenda

- **Part 1: Deep Learning Framework** (Shao-Hua Sun)
  - TensorFlow (Shao-Hua Sun)
  - PyTorch (Te-Lin Wu)
- **Part 2: Cloud Service**
  - Google Cloud
  - Amazon Web Services

Today's agenda

- **Part 1: Deep Learning Framework** (Shao-Hua Sun)
  - TensorFlow (Shao-Hua Sun)
  - PyTorch (Te-Lin Wu)
- **Part 2: Cloud Service** (Hanpeng Liu)
  - Google Cloud
  - Amazon Web Services

Today's agenda

- **Part 1: Deep Learning Framework**
  - TensorFlow
  - PyTorch
- Part 2: Cloud Service
  - Google Cloud
  - Amazon Web Services

Deep Learning Framework

- TensorFlow
- PyTorch
- Torch
- Caffe
- Caffe2
- Theano
- Keras
- Etc.

Deep Learning Framework

Deep Learning Framework

torch

NYU/Facebook

Deep Learning Framework

torch
NYU/Facebook

Deep Learning Framework

torch
NYU/Facebook

PYTORCH
Facebook

Deep Learning Framework

torch
NYU/Facebook

PYTORCH
Facebook

theano
U Montreal

Deep Learning Framework

torch
NYU/Facebook

PYTORCH
Facebook

theano
U Montreal

Deep Learning Framework

torch
NYU/Facebook

PYTORCH
Facebook

theano
U Montreal

TensorFlow
Google

Deep Learning Framework

torch
NYU/Facebook

PYTORCH
Facebook

theano
U Montreal

TensorFlow
Google

Caffe
UC Berkeley

Deep Learning Framework

torch
NYU/Facebook → PYTORCH
Facebook

theano
U Montreal → TensorFlow
Google

Caffe
UC Berkeley →

Deep Learning Framework

torch NYU/Facebook	- - - →	PYTORCH Facebook
theano U Montreal	→	TensorFlow Google
Caffe UC Berkeley	→	Caffe2 Facebook

Deep Learning Framework

- Model specification
  - Configuration file
    - Caffe
  - Programmatic generation
    - PyTorch, Theano, TensorFlow

Deep Learning Framework

- Language
  - C++
    - Caffe
  - Lua
    - Torch
  - Python
    - PyTorch, Theano, TensorFlow

Today's agenda

- **Part 1: Deep Learning Framework**
  - TensorFlow
  - PyTorch
- Part 2: Cloud Service
  - Google Cloud
  - Amazon Web Services

TensorFlow

TensorFlow

- A deep learning library

TensorFlow

- A deep learning library
- Open-sourced by Google

TensorFlow

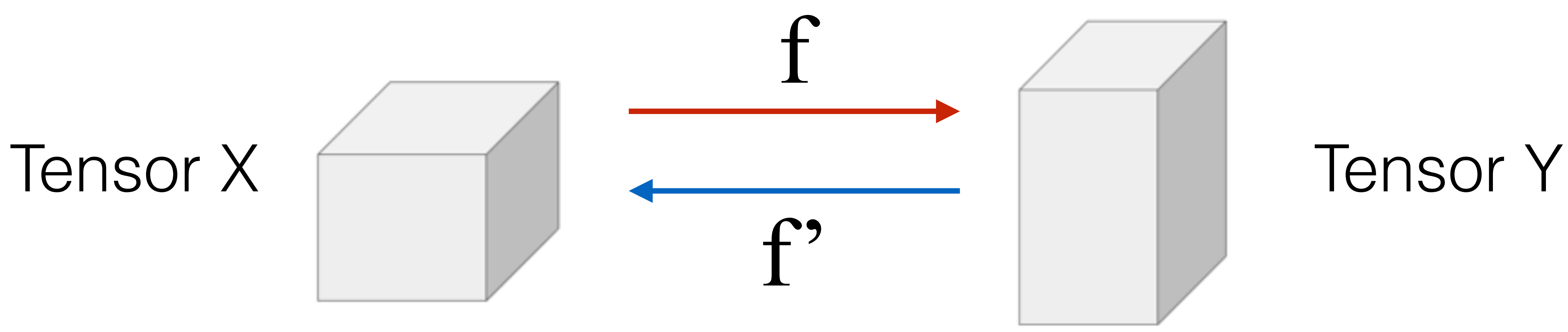
- A deep learning library
- Open-sourced by Google
- Written with a Python API over a C/C++ engine

TensorFlow

- A deep learning library
- Open-sourced by Google
- Written with a Python API over a C/C++ engine
- Provides primitives for defining functions on tensors and automatically computing their derivatives

TensorFlow

- A deep learning library
- Open-sourced by Google
- Written with a Python API over a C/C++ engine
- Provides primitives for defining functions on tensors and automatically computing their derivatives



Why is it named TensorFlow?

TensorFlow

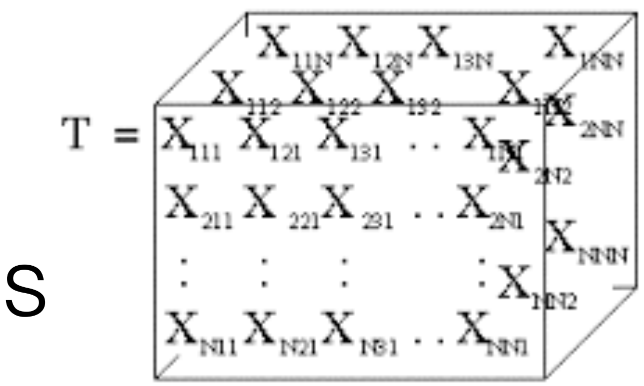
Tensor Flow

TensorFlow

- What is a **Tensor**?

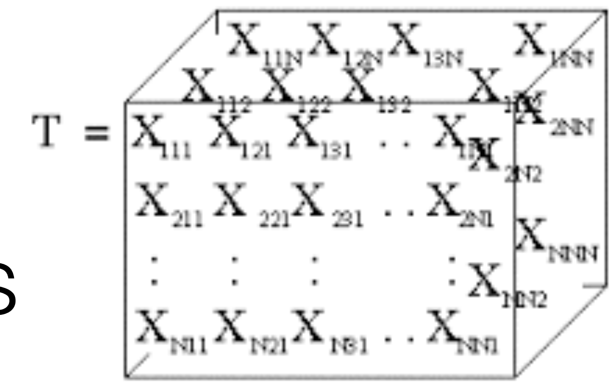
TensorFlow

- What is a **Tensor**?
  - A **multidimensional array** of numbers



TensorFlow

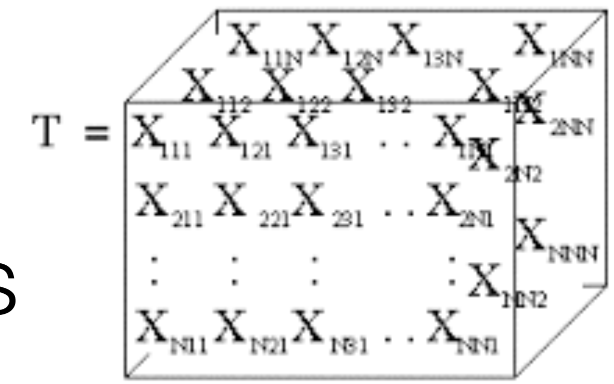
- What is a **Tensor**?
  - A **multidimensional array** of numbers



- A scalar is a tensor

TensorFlow

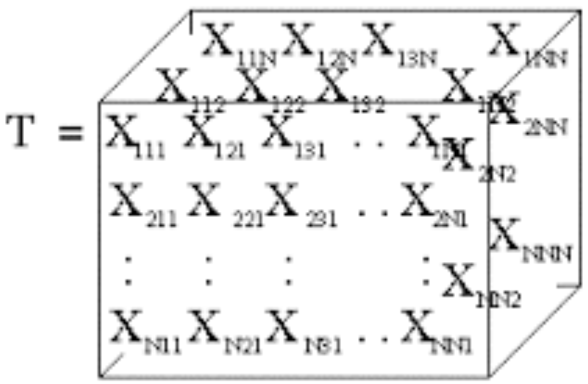
- What is a **Tensor**?
  - A **multidimensional array** of numbers



- A scalar is a tensor
- A vector is a tensor

TensorFlow

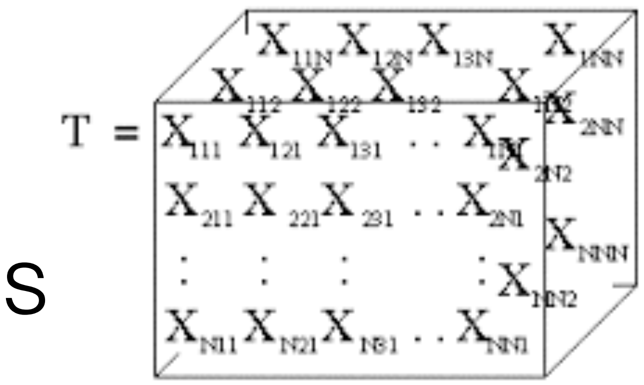
- What is a **Tensor**?
  - A **multidimensional array** of numbers



- A scalar is a tensor
- A vector is a tensor
- A matrix is a tensor

TensorFlow

- What is a **Tensor**?
  - A **multidimensional array** of numbers



- A scalar is a tensor
- A vector is a tensor
- A matrix is a tensor

We use **Tensor** structure to represent **data**

TensorFlow

- Why **Flow**?

TensorFlow

- Why **Flow**?
  - Linear classification

$$\underbrace{\begin{bmatrix} 0.1 & -0.15 & 0 & 0.3 \\ -0.5 & 0.2 & 0.1 & 0.35 \\ -0.45 & 0.4 & 0.1 & 0.15 \end{bmatrix}}_{W} \underbrace{\begin{bmatrix} 0.3 \\ 0.15 \\ 1.5 \\ 1.91 \end{bmatrix}}_{x} + \underbrace{\begin{bmatrix} 0.15 \\ -0.1 \\ -0.2 \end{bmatrix}}_{b} = \underbrace{\begin{bmatrix} \boxed{0.73} \\ 0.598 \\ 0.161 \end{bmatrix}}_{f(x;\, W,\, b)} \begin{array}{l} \text{Cat} \\ \text{Car} \\ \text{Airplane} \end{array}$$

TensorFlow

- Why **Flow**?
  - Linear classification: can be represented as

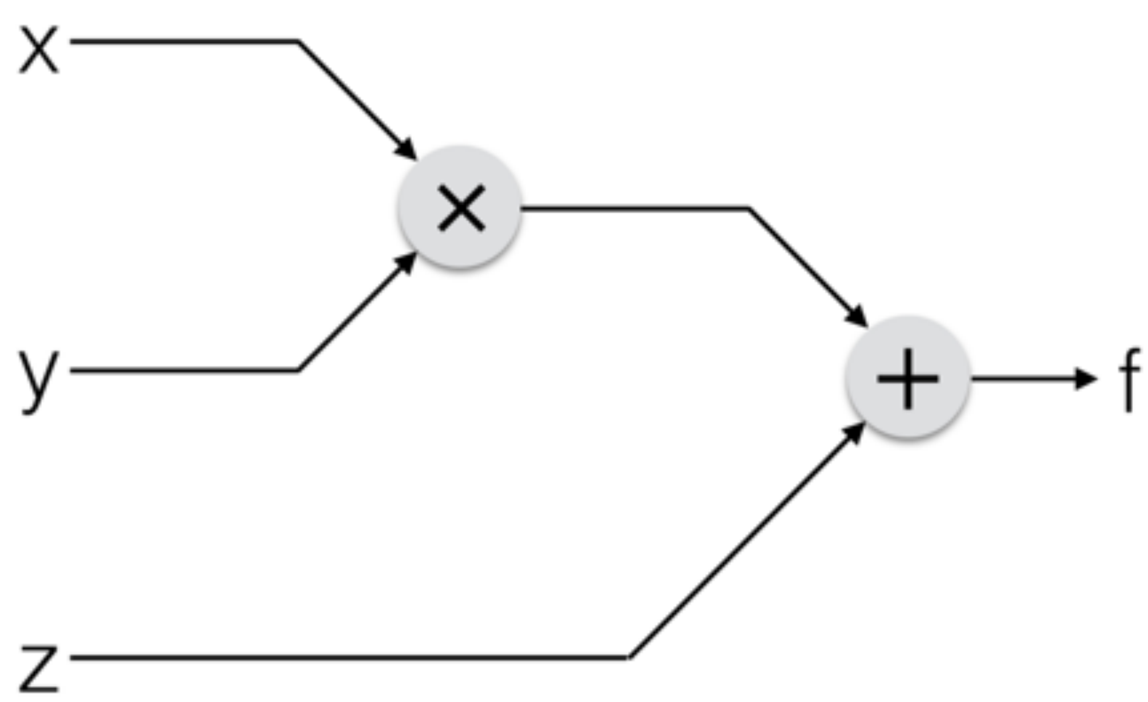
$$\underbrace{\begin{bmatrix} 0.1 & -0.15 & 0 & 0.3 \\ -0.5 & 0.2 & 0.1 & 0.35 \\ -0.45 & 0.4 & 0.1 & 0.15 \end{bmatrix}}_{W} \underbrace{\begin{bmatrix} 0.3 \\ 0.15 \\ 1.5 \\ 1.91 \end{bmatrix}}_{x} + \underbrace{\begin{bmatrix} 0.15 \\ -0.1 \\ -0.2 \end{bmatrix}}_{b} = \underbrace{\begin{bmatrix} \boxed{0.73} \\ 0.598 \\ 0.161 \end{bmatrix}}_{f(x;\, W,\, b)} \begin{array}{l} \text{Cat} \\ \text{Car} \\ \text{Airplane} \end{array}$$

TensorFlow

- Why **Flow**?
  - Data flow graph

TensorFlow

- Why **Flow**?
  - Data flow graph
    - Remember the **computational graph**?

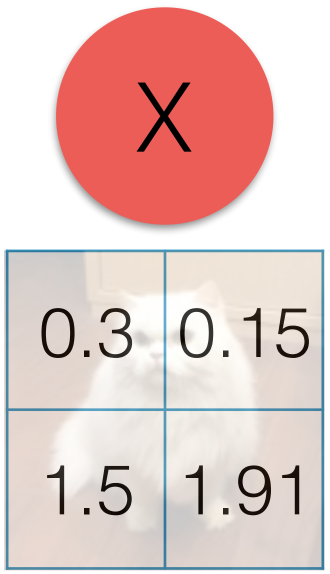


TensorFlow

- Why **Flow**?
  - Data flow graph

TensorFlow

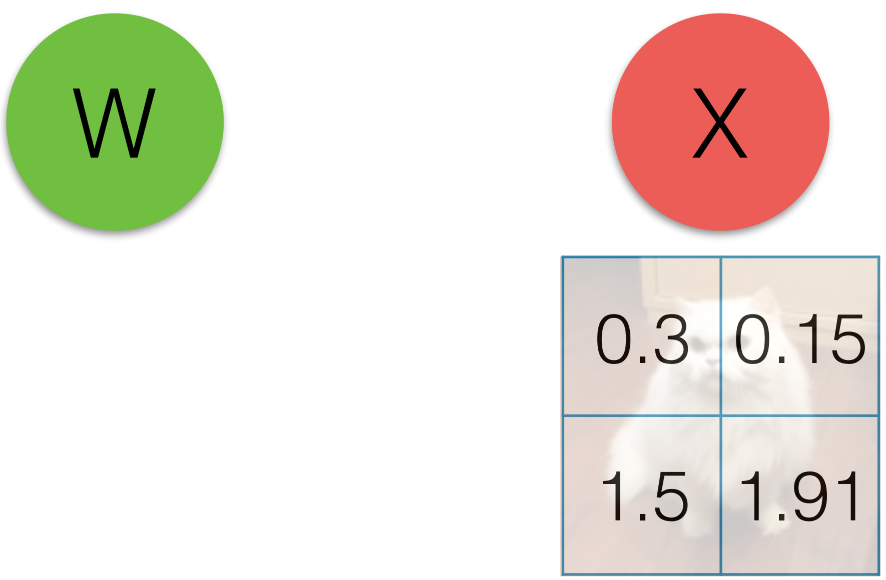
- Why **Flow**?
    - Data flow graph



TensorFlow

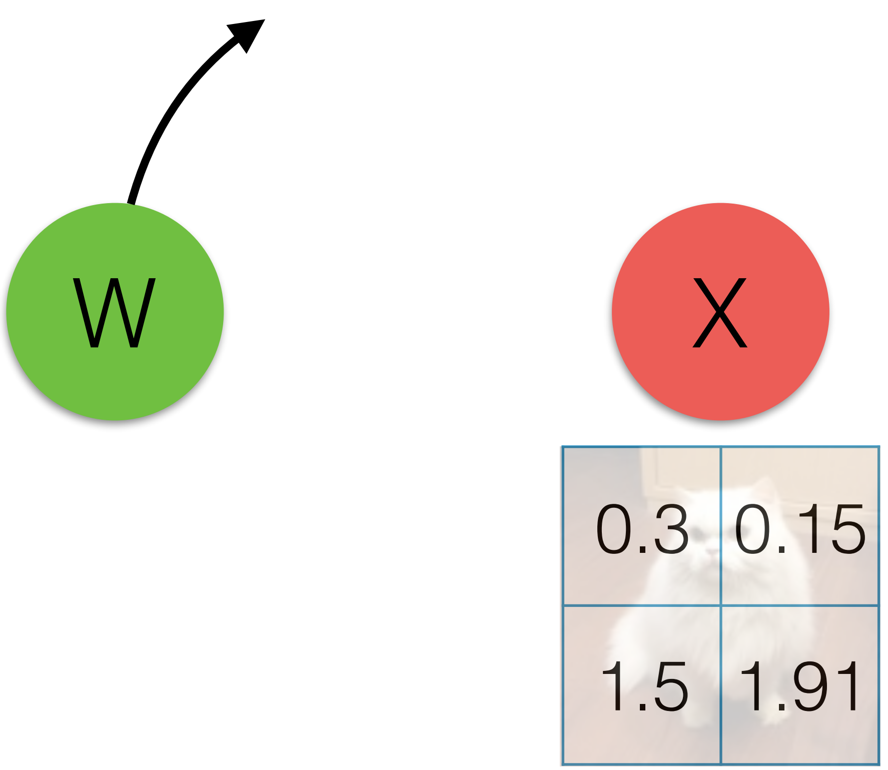
- Why **Flow**?

  - Data flow graph



TensorFlow

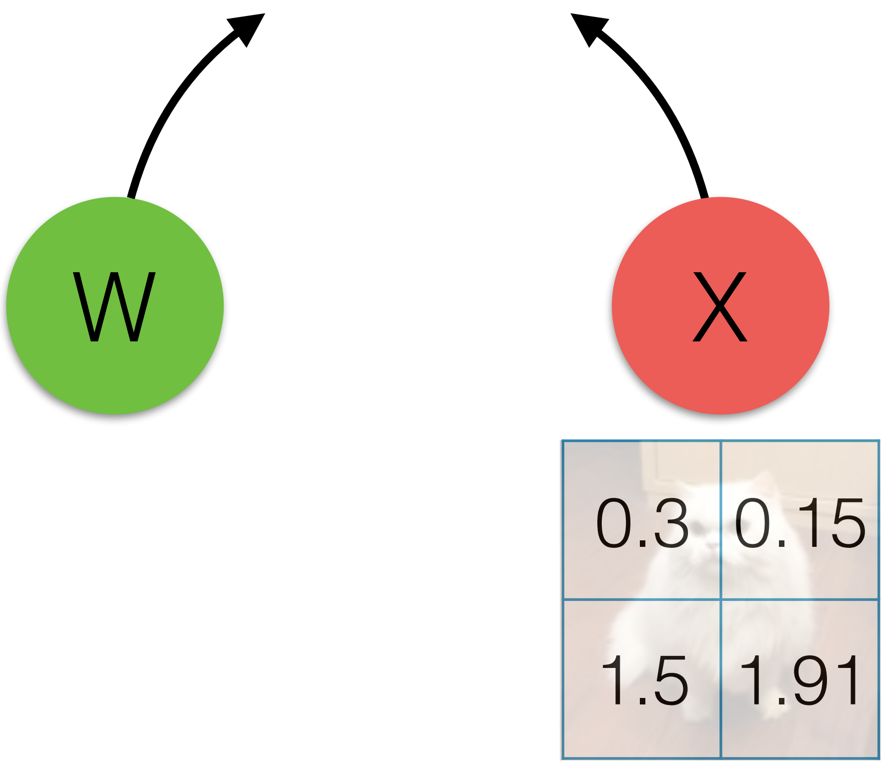
- Why **Flow**?
  - Data flow graph



TensorFlow

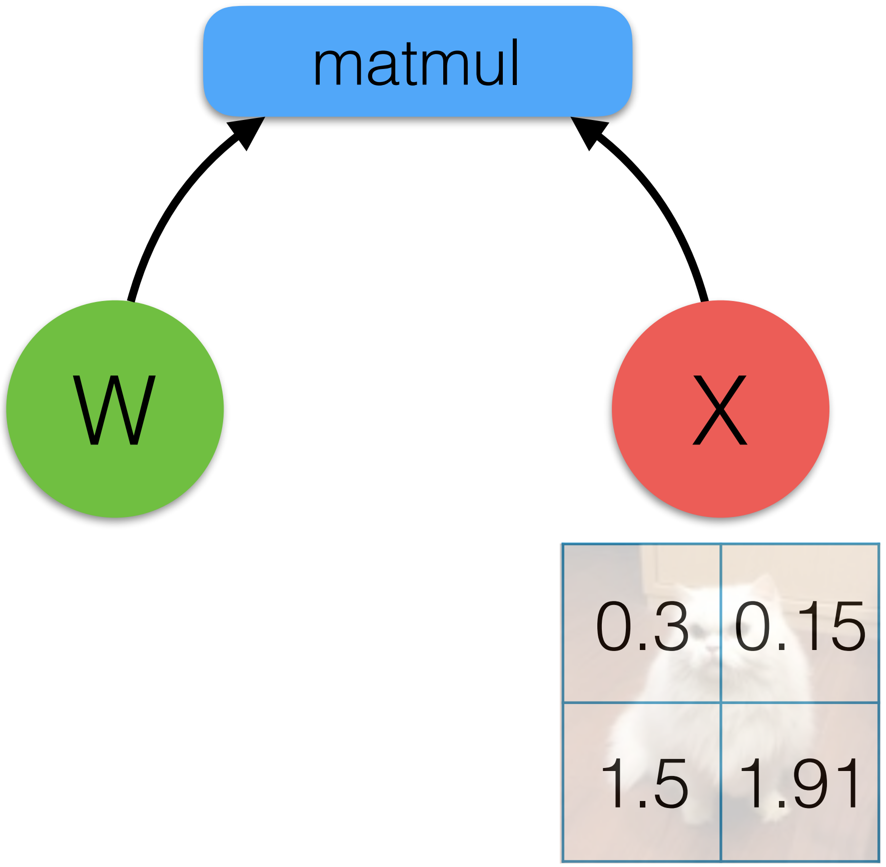
- Why **Flow**?

  - Data flow graph



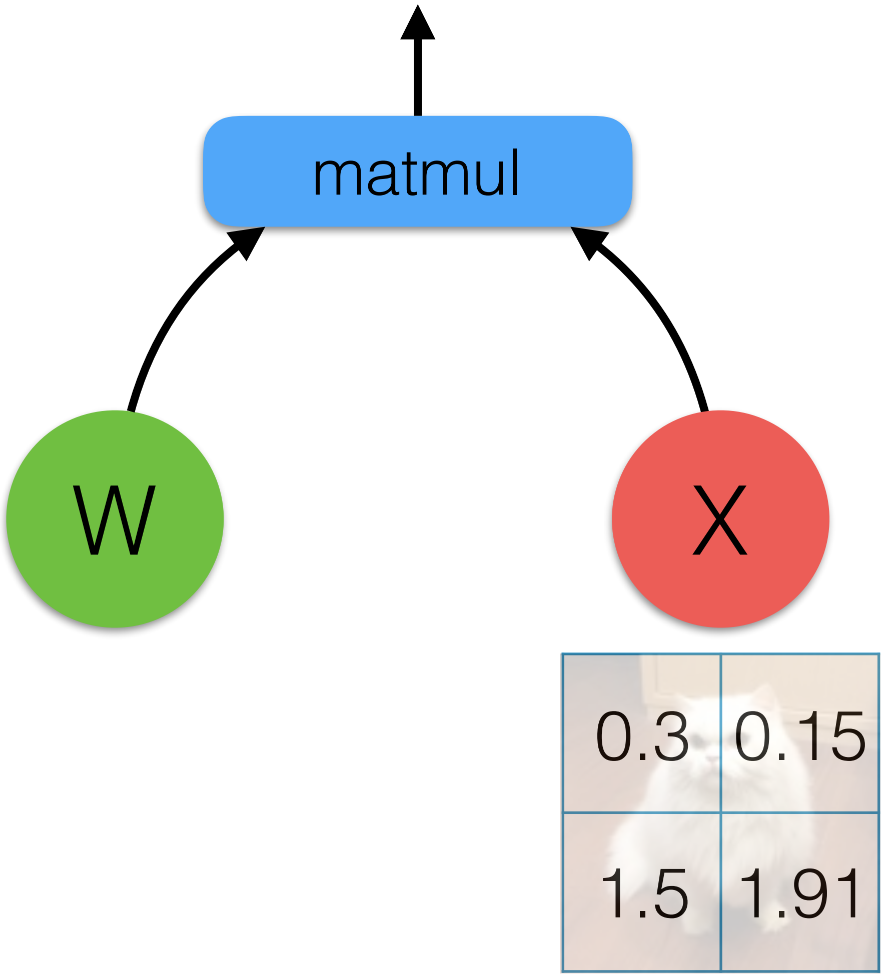
TensorFlow

- Why **Flow**?
  - Data flow graph



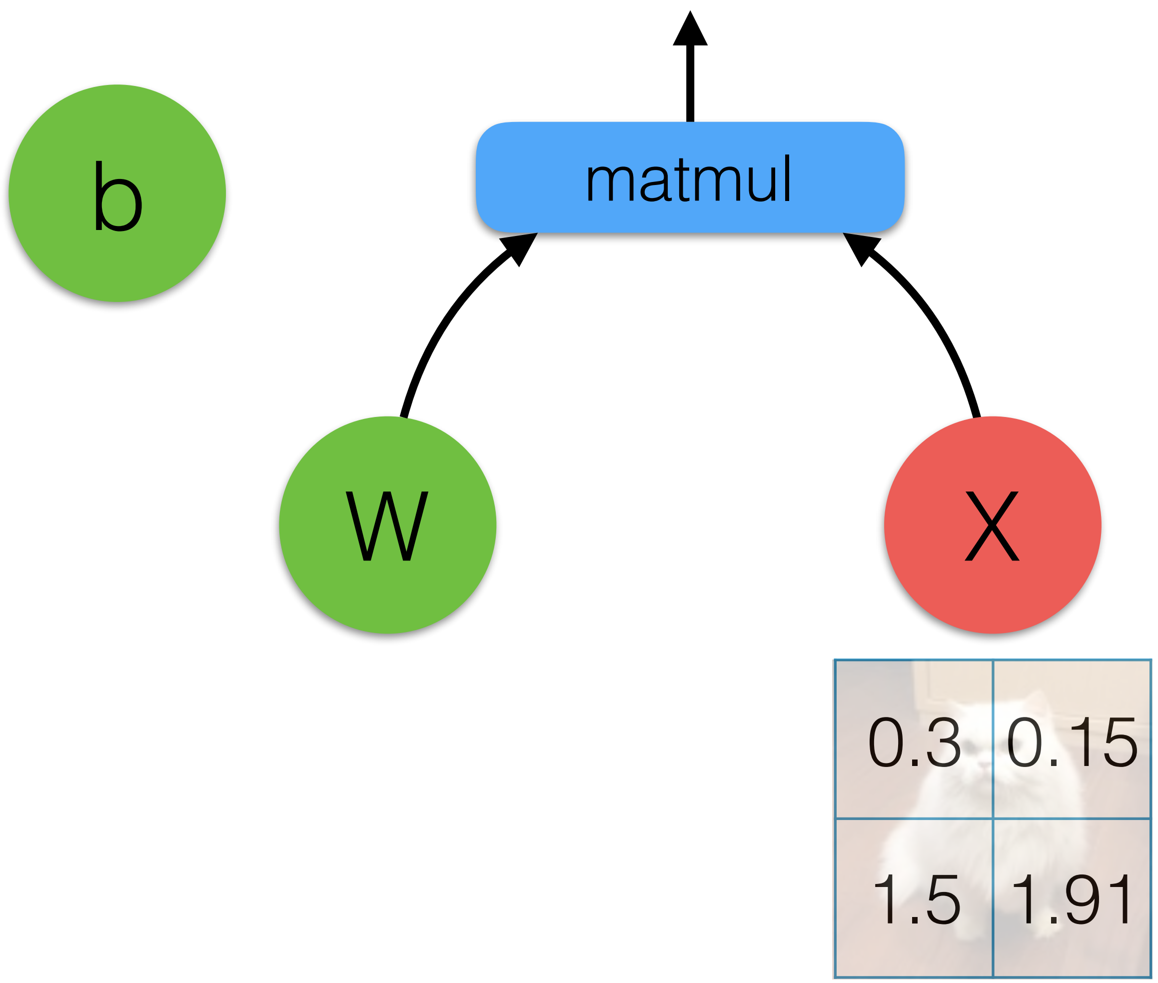
TensorFlow

- Why **Flow**?
  - Data flow graph



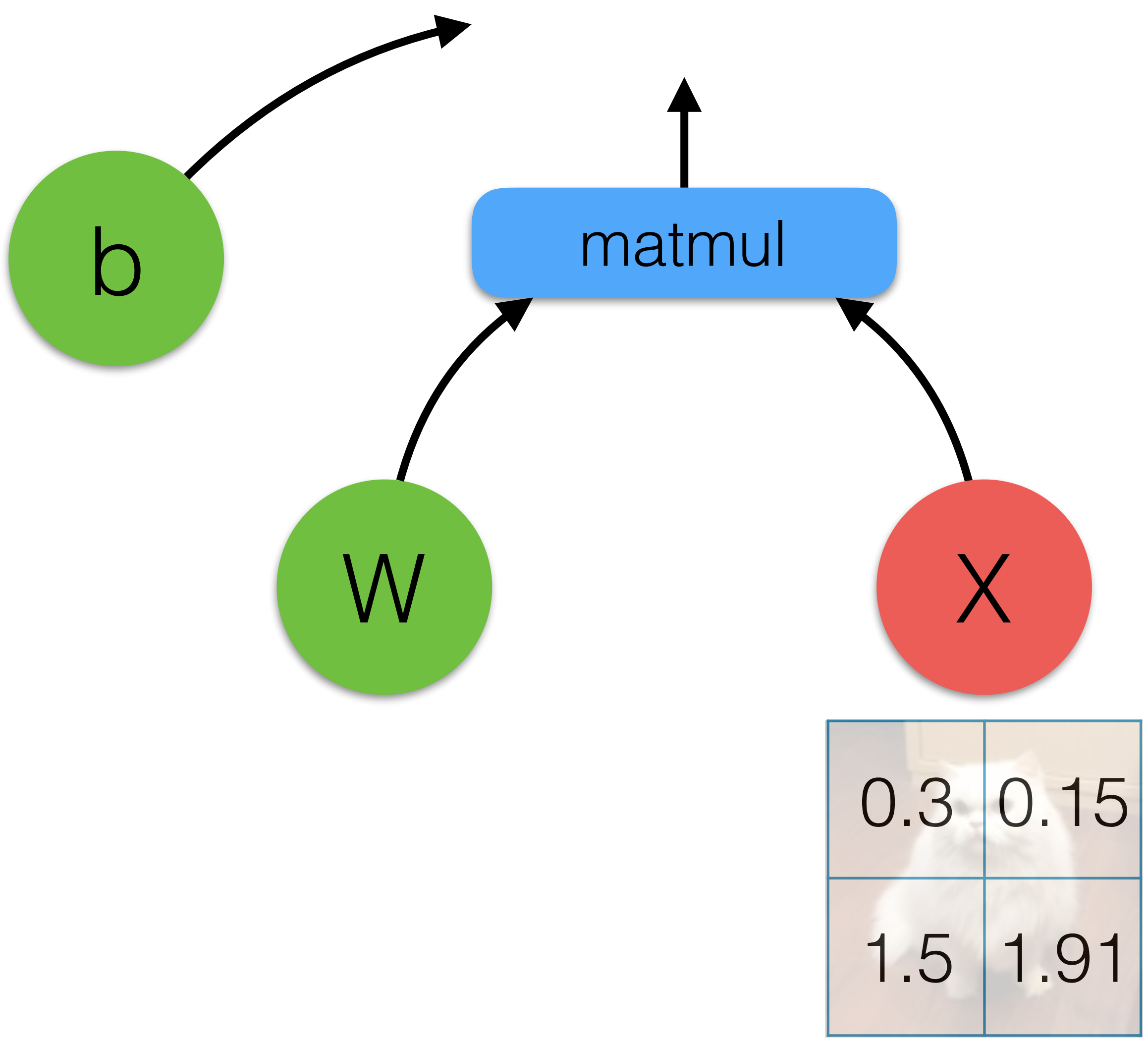
TensorFlow

- Why **Flow**?
  - Data flow graph



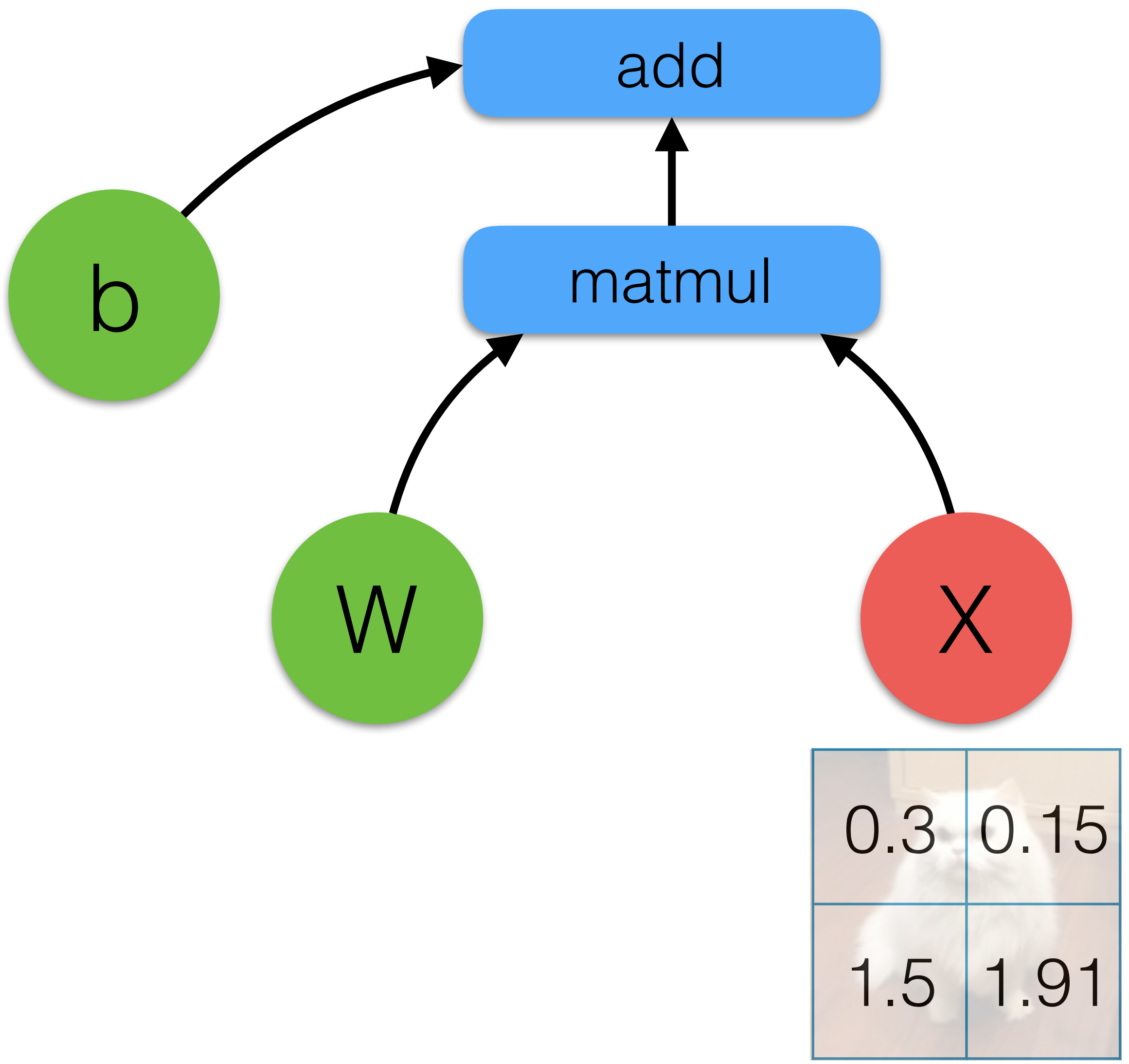
TensorFlow

- Why **Flow**?
  - Data flow graph



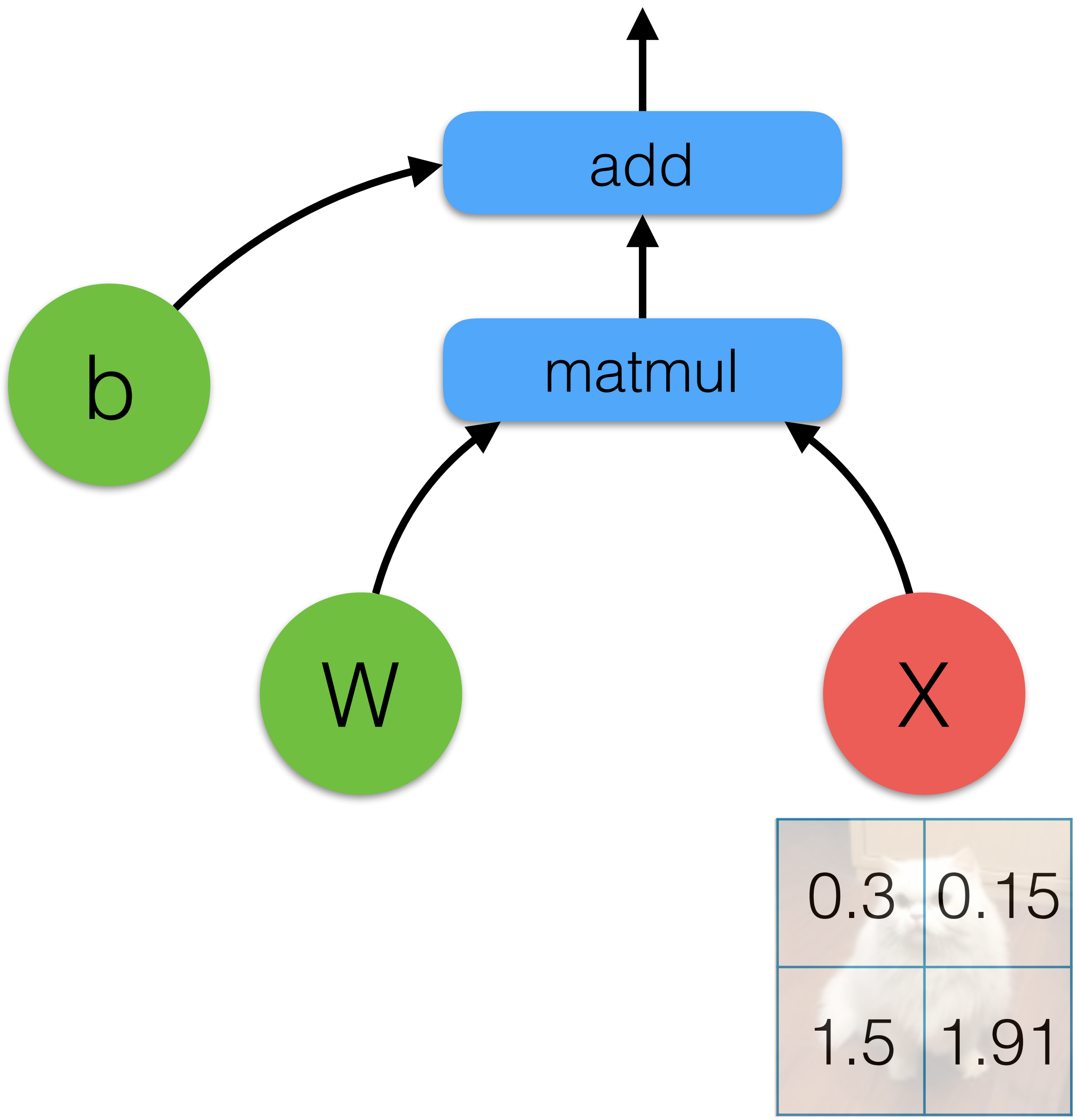
TensorFlow

- Why **Flow**?
    - Data flow graph



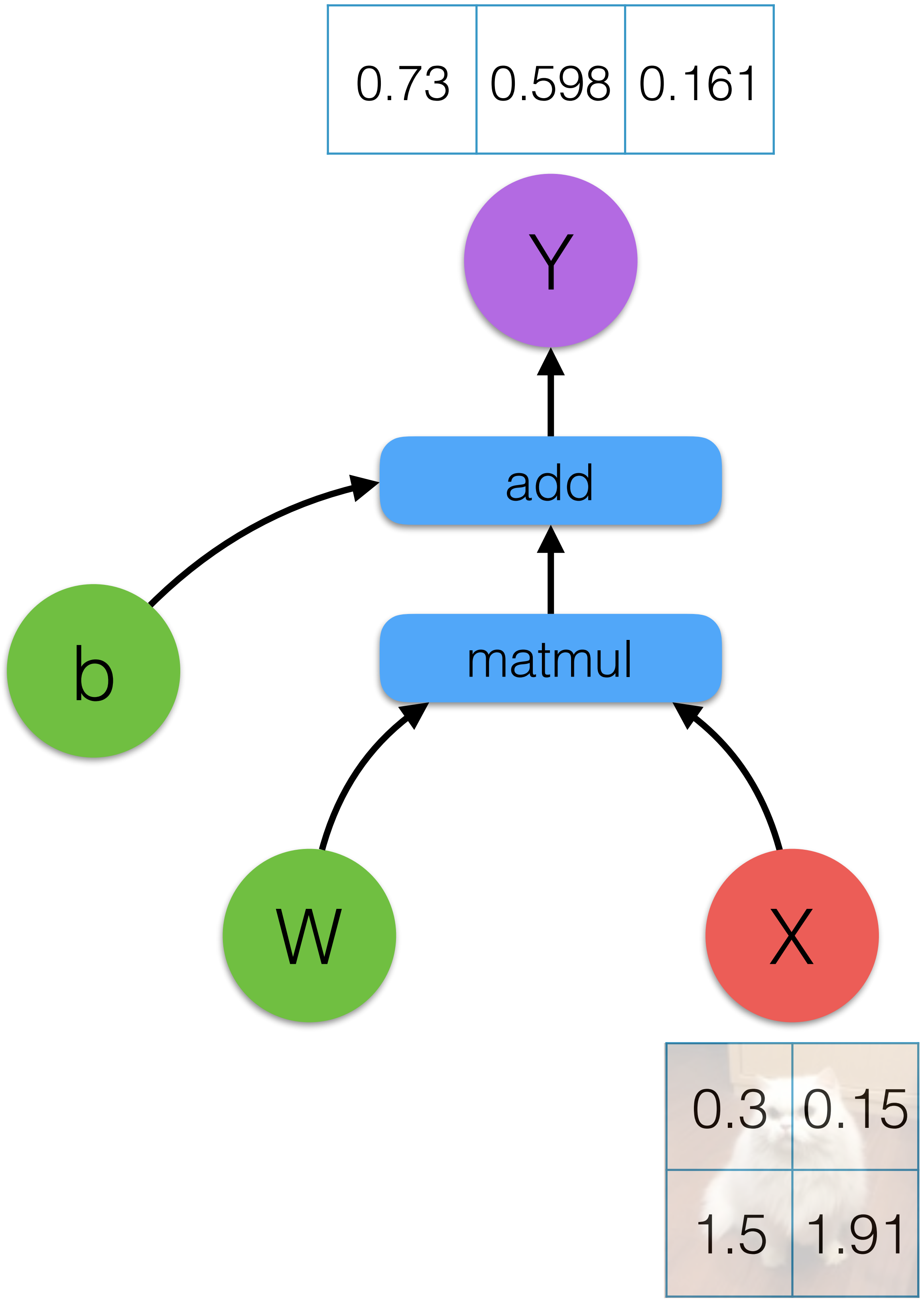
TensorFlow

- Why **Flow**?
  - Data flow graph



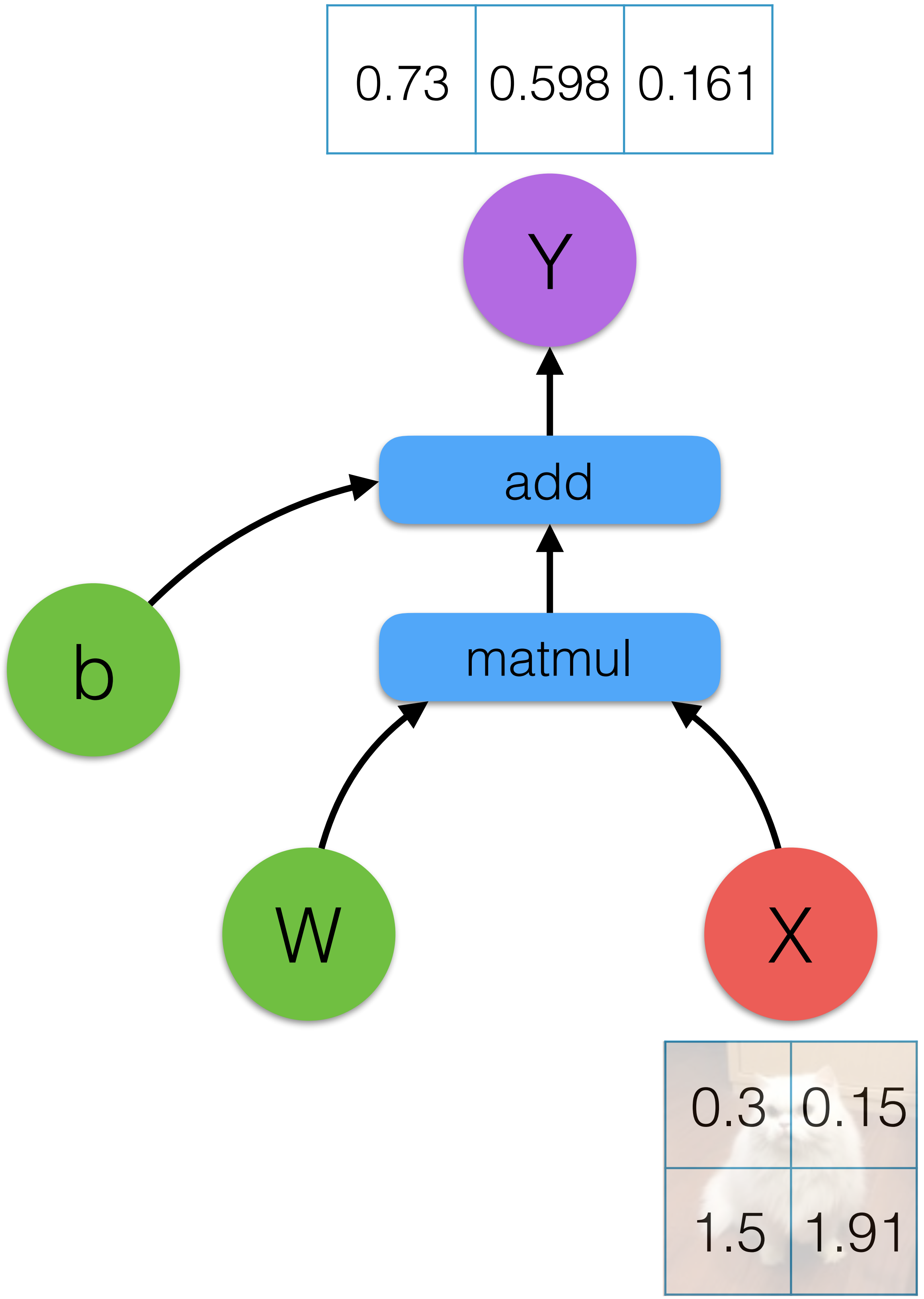
TensorFlow

- Why **Flow**?
  - Data flow graph



TensorFlow

- Why **Flow**?
  - Data flow graph
    - Nodes: operations
    - Edges: tensors (data)



Tensors Flow

Tensors Flow

Data

Tensors

Data

Flow

Computation

Now you know why it is named TensorFlow

How does it work?

Survey

How much do you know about TensorFlow?

- Never heard about it
- Never used it
- Played with it a little bit
- Worked on a project with it

How does it work?

TensorFlow Core tutorial

This gives Python access to all of TensorFlow's classes, methods, and symbols

```
import tensorflow as tf
```

TensorFlow tutorial

The Computational Graph

Two sections

- Building the computational graph
- Running the computational graph

TensorFlow tutorial

The Computational Graph

- Building the computational graph

TensorFlow tutorial

The Computational Graph

- Building the computational graph

```
node1 = tf.constant(3.0, dtype=tf.float32)
node2 = tf.constant(4.0) # also tf.float32 implicitly
print(node1, node2)
```

TensorFlow tutorial

The Computational Graph

- Building the computational graph

```
node1 = tf.constant(3.0, dtype=tf.float32)
node2 = tf.constant(4.0) # also tf.float32 implicitly
print(node1, node2)
```

TensorFlow tutorial

The Computational Graph

- Building the computational graph

```
node1 = tf.constant(3.0, dtype=tf.float32)
node2 = tf.constant(4.0) # also tf.float32 implicitly
print(node1, node2)
```

Output

TensorFlow tutorial

The Computational Graph

- Building the computational graph

```
node1 = tf.constant(3.0, dtype=tf.float32)
node2 = tf.constant(4.0) # also tf.float32 implicitly
print(node1, node2)
```

Output

```
Tensor("Const:0", shape=(), dtype=float32) Tensor("Const_1:0", shape=(), dtype=float32)
```

TensorFlow tutorial

The Computational Graph

- Building the computational graph

```
node1 = tf.constant(3.0, dtype=tf.float32)
node2 = tf.constant(4.0) # also tf.float32 implicitly
print(node1, node2)
```

Output

```
Tensor("Const:0", shape=(), dtype=float32) Tensor("Const_1:0", shape=(), dtype=float32)
```

- Running the computational graph

TensorFlow tutorial

The Computational Graph

- Building the computational graph

```
node1 = tf.constant(3.0, dtype=tf.float32)
node2 = tf.constant(4.0) # also tf.float32 implicitly
print(node1, node2)
```

Output

```
Tensor("Const:0", shape=(), dtype=float32) Tensor("Const_1:0", shape=(), dtype=float32)
```

- Running the computational graph

```
sess = tf.Session()
print(sess.run([node1, node2]))
```

TensorFlow tutorial

The Computational Graph

- Building the computational graph

```
node1 = tf.constant(3.0, dtype=tf.float32)
node2 = tf.constant(4.0) # also tf.float32 implicitly
print(node1, node2)
```

Output

```
Tensor("Const:0", shape=(), dtype=float32) Tensor("Const_1:0", shape=(), dtype=float32)
```

- Running the computational graph

```
sess = tf.Session()
print(sess.run([node1, node2]))
```

Output

TensorFlow tutorial

The Computational Graph

- Building the computational graph

```
node1 = tf.constant(3.0, dtype=tf.float32)
node2 = tf.constant(4.0) # also tf.float32 implicitly
print(node1, node2)
```

Output

```
Tensor("Const:0", shape=(), dtype=float32) Tensor("Const_1:0", shape=(), dtype=float32)
```

- Running the computational graph

```
sess = tf.Session()
print(sess.run([node1, node2]))
```

Output

```
[3.0, 4.0]
```

TensorFlow tutorial

Implementation Demo

Implementation Demo

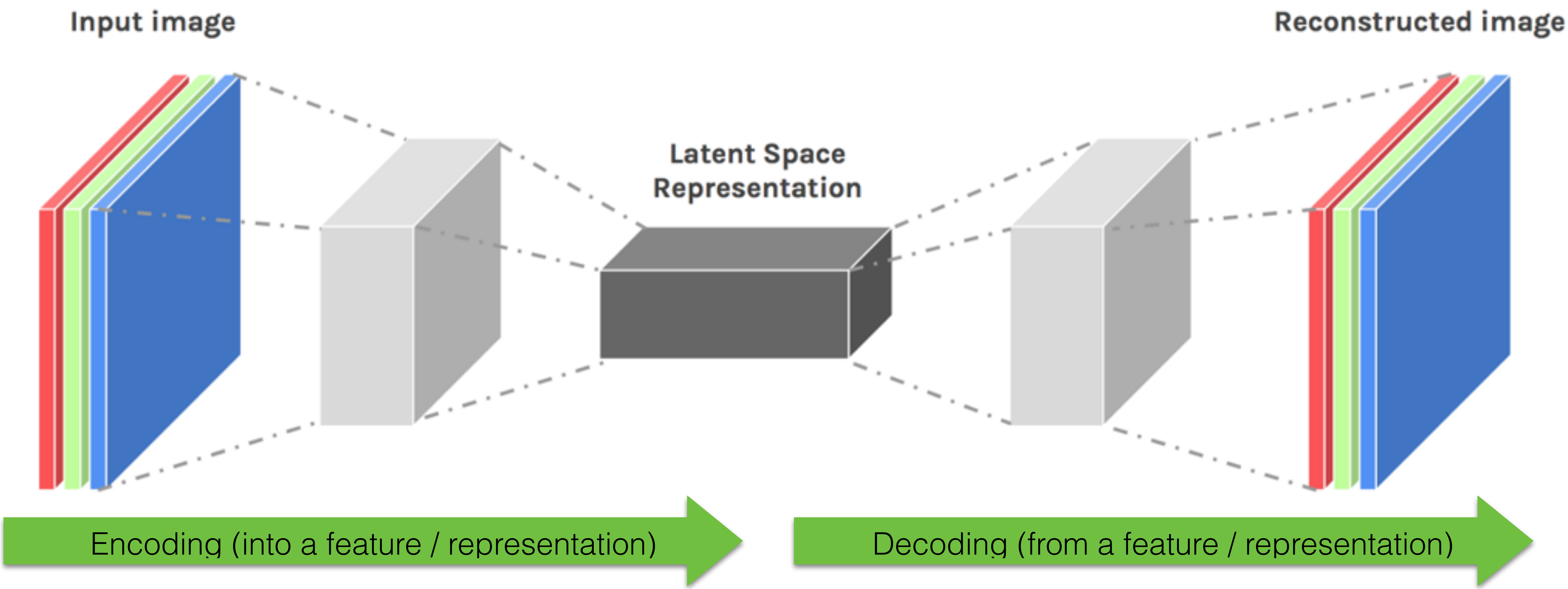
- Building phase
  - Build a network
- Running phase
  - Train the network
  - Test the network

Implementation Demo

- Variational autoencoder
  - Background
  - Model architecture
- Demo
  - Build a network
  - Train the network
  - Test the network

Autoencoder

- Autoencoder



PC: https://hackernoon.com/latent-space-visualization-deep-learning-bits-2-bd09a46920df

Autoencoder

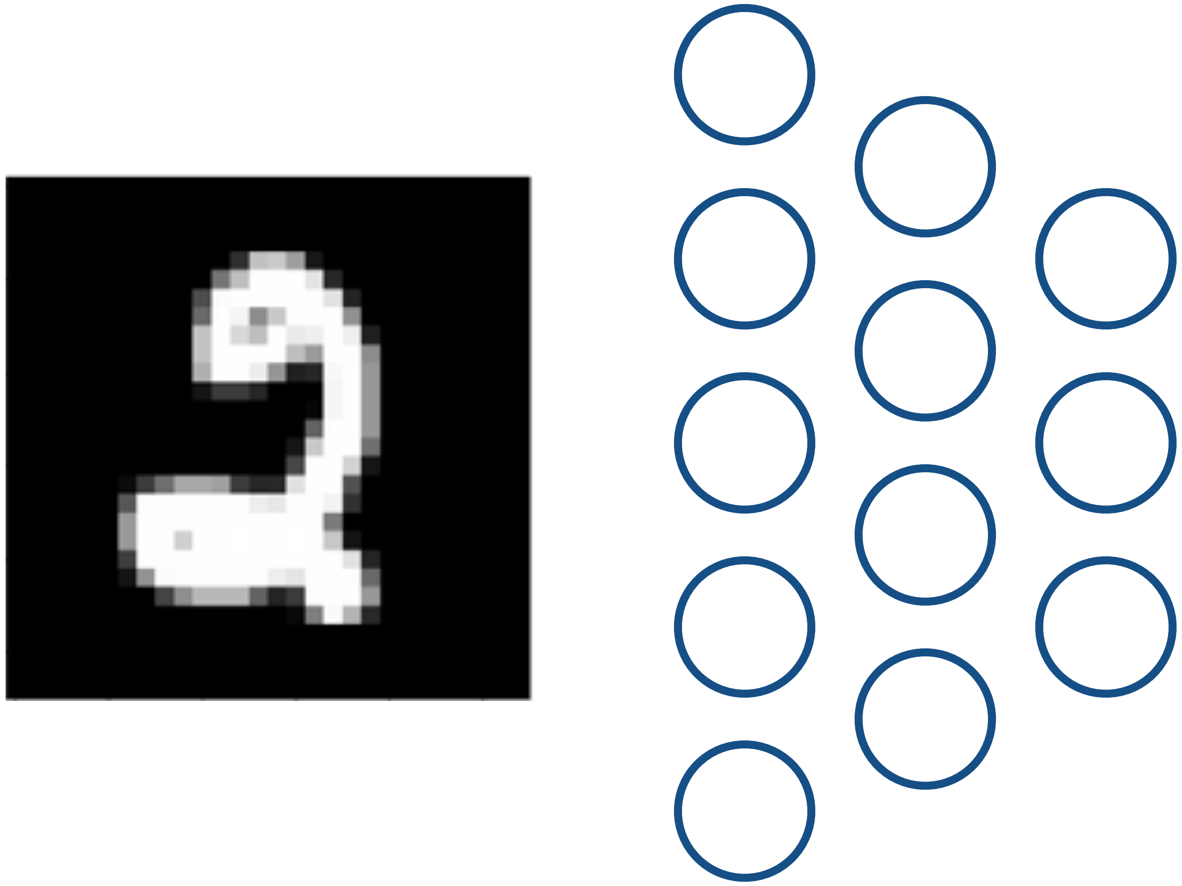
- Autoencoder
  - Compression

Autoencoder

- Autoencoder
  - Compression

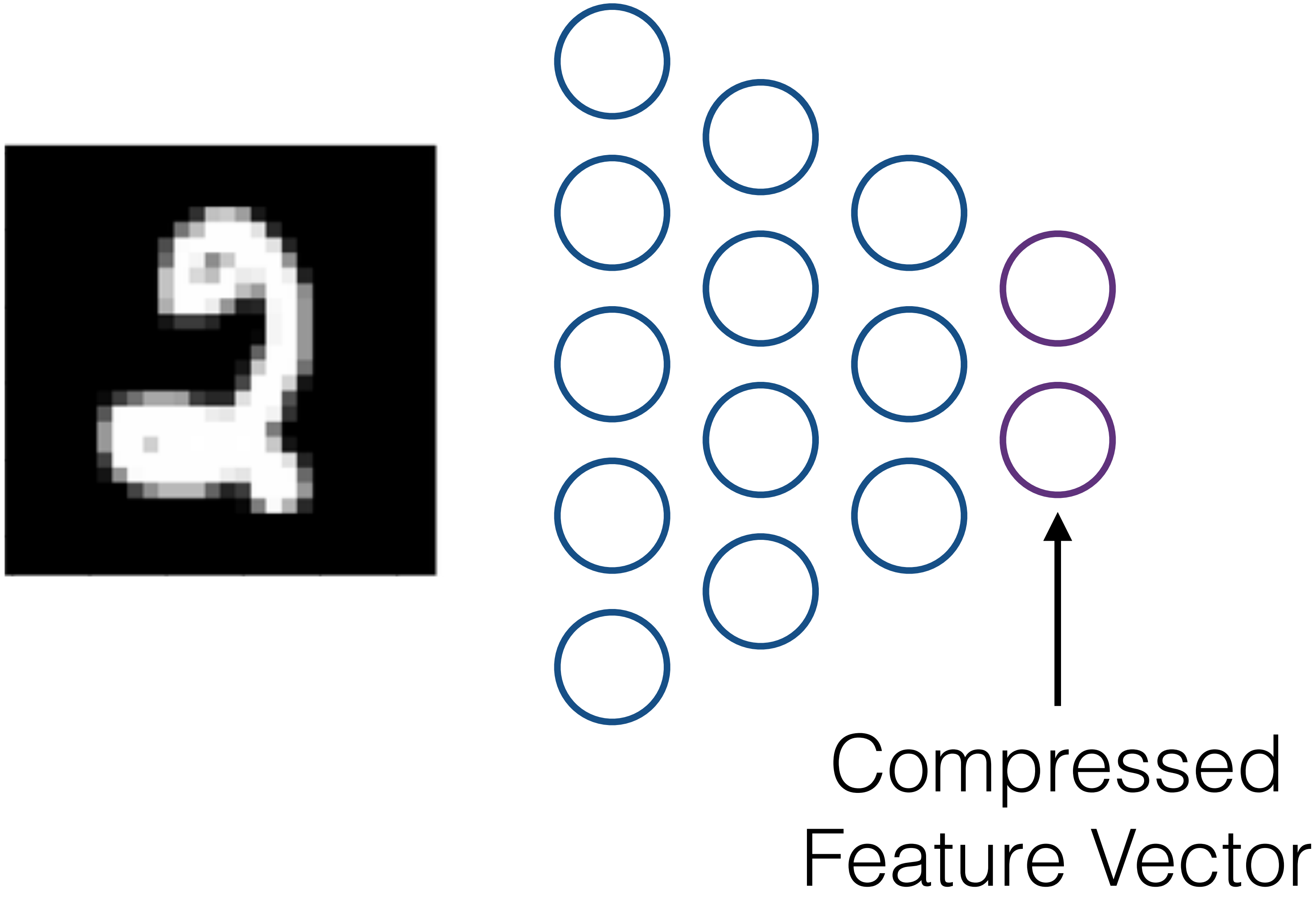
Autoencoder

- Autoencoder
    - Compression



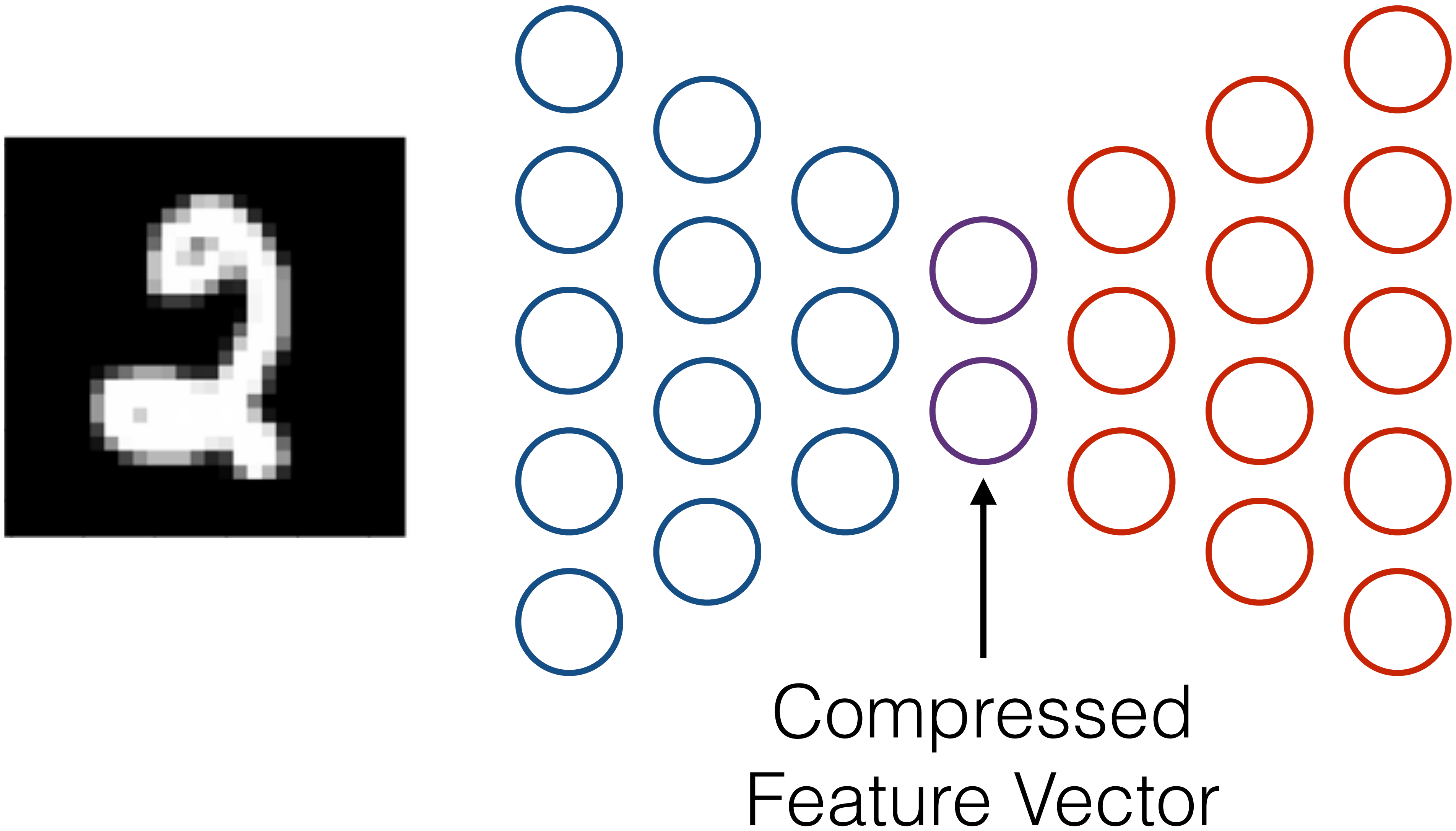
Autoencoder

- Autoencoder
  - Compression



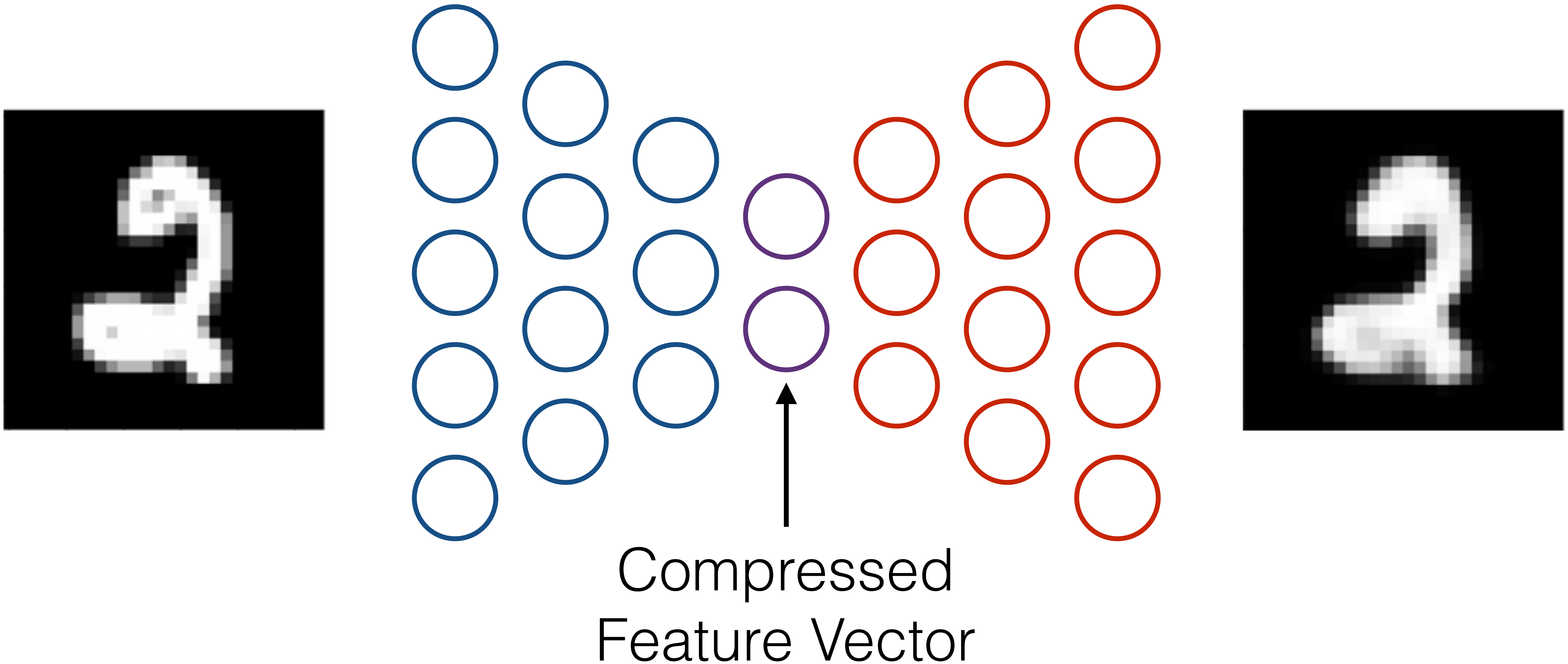
Autoencoder

- Autoencoder
  - Compression



Autoencoder

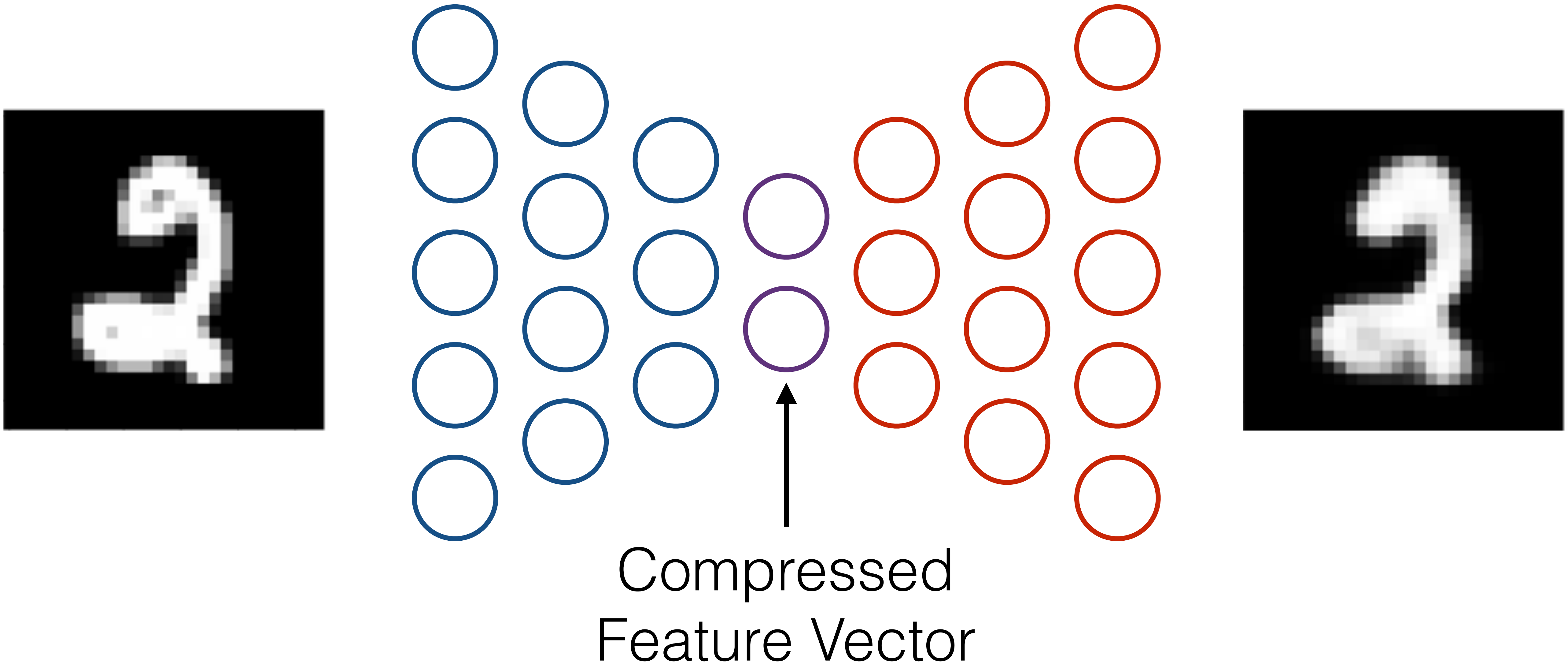
- Autoencoder
  - Compression



Autoencoder

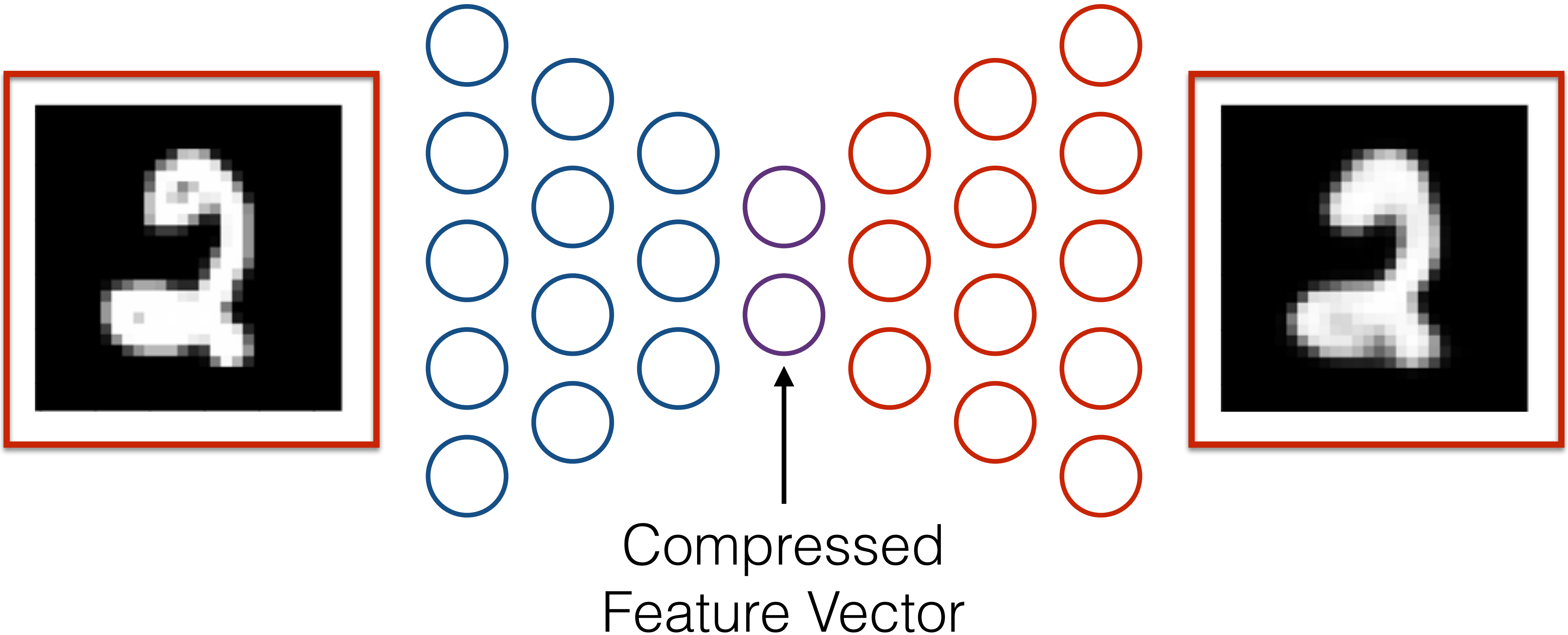
- Loss

  - Reconstruction loss



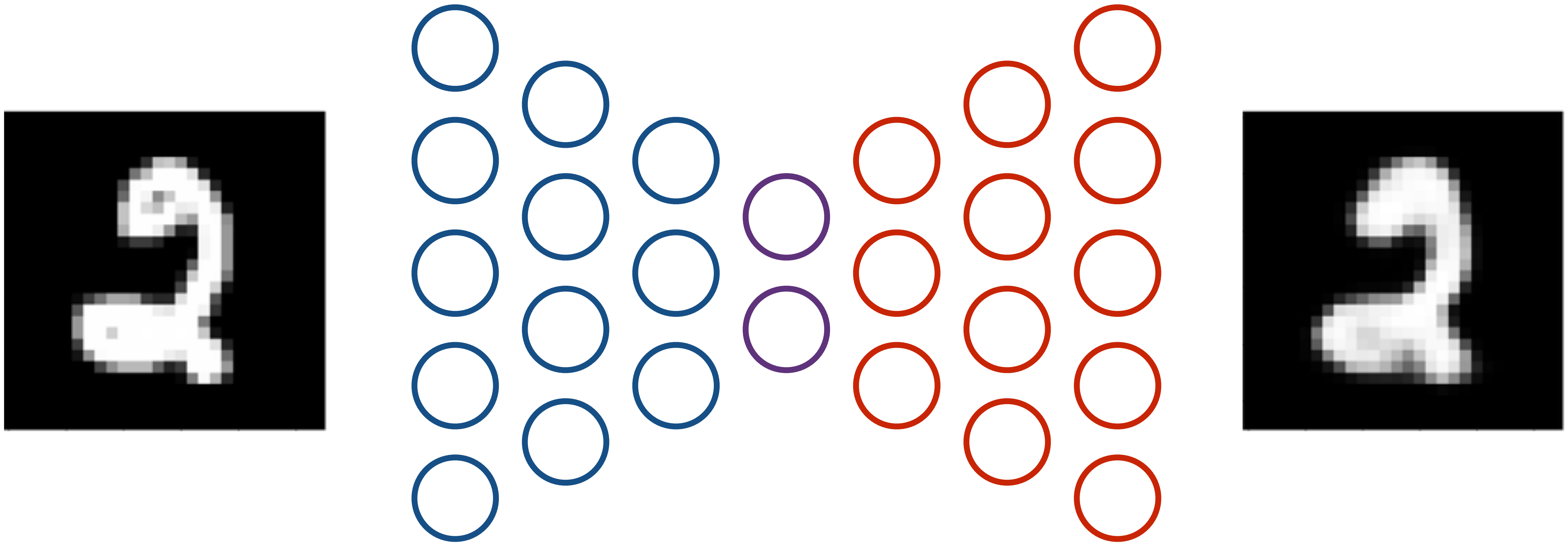
Autoencoder

- Loss
  - Reconstruction loss



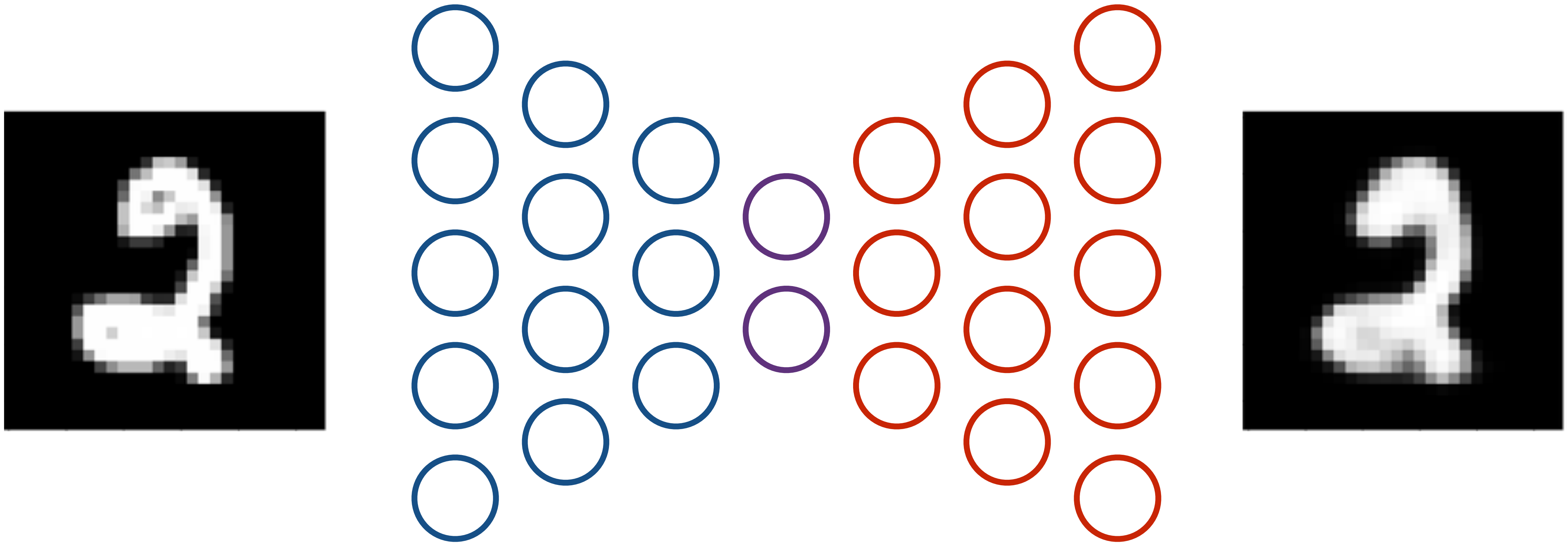
Autoencoder

- Model
  - Decoder (3 layers)
  - Encoder (3 layers)



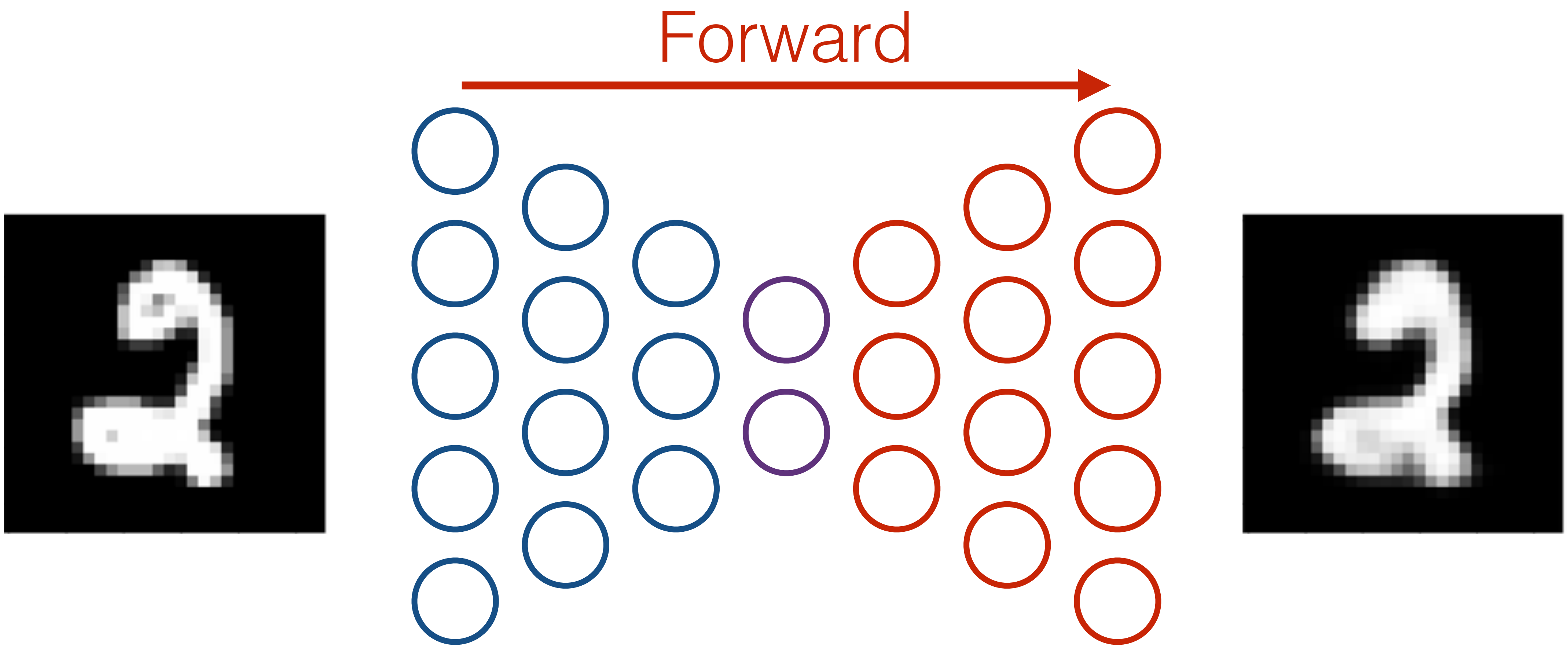
Autoencoder

- Function: training phase
  - Train the model given a batch by minimizing the reconstruction loss



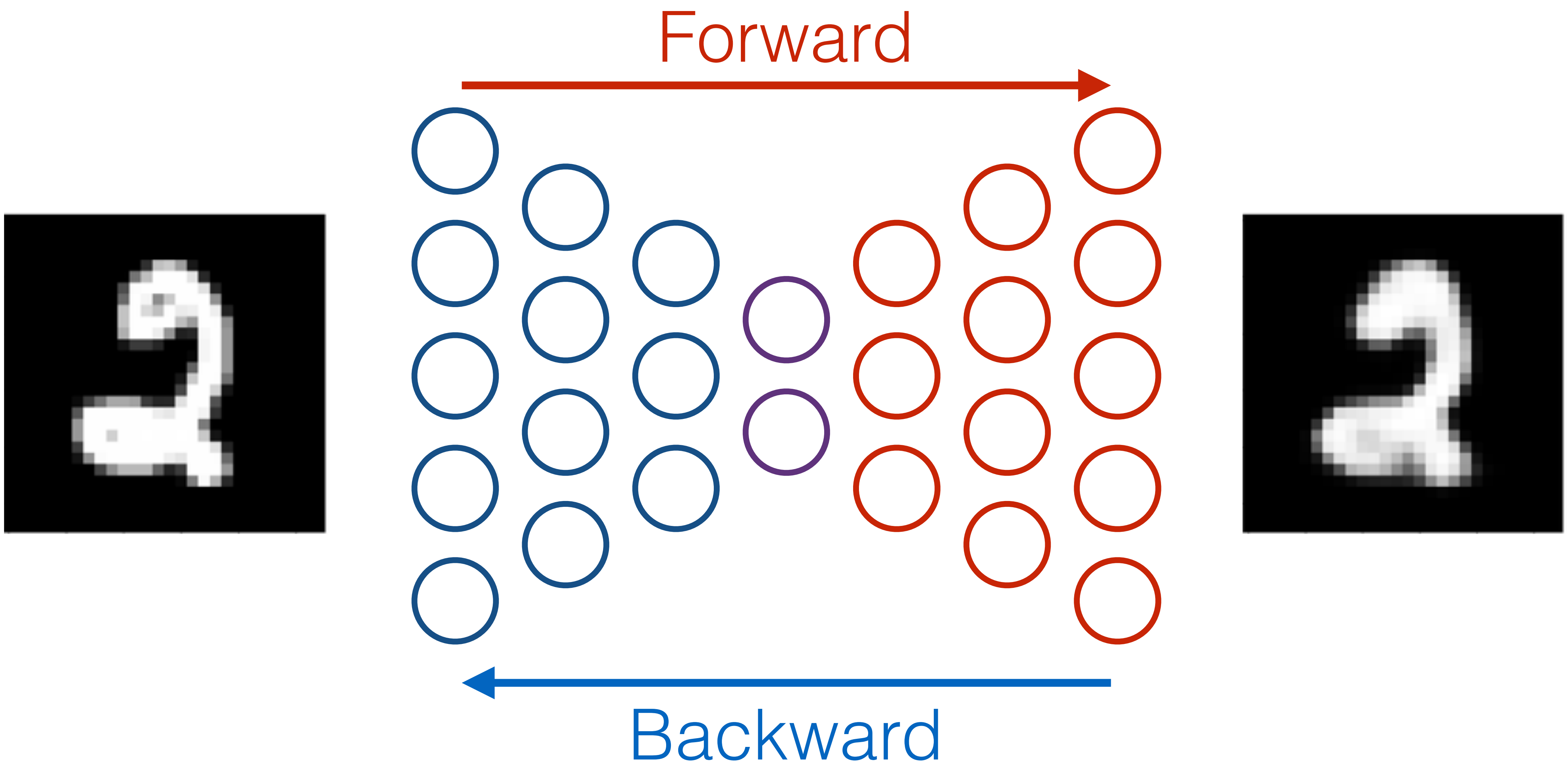
Autoencoder

- Function: training phase
  - Train the model given a batch by minimizing the reconstruction loss



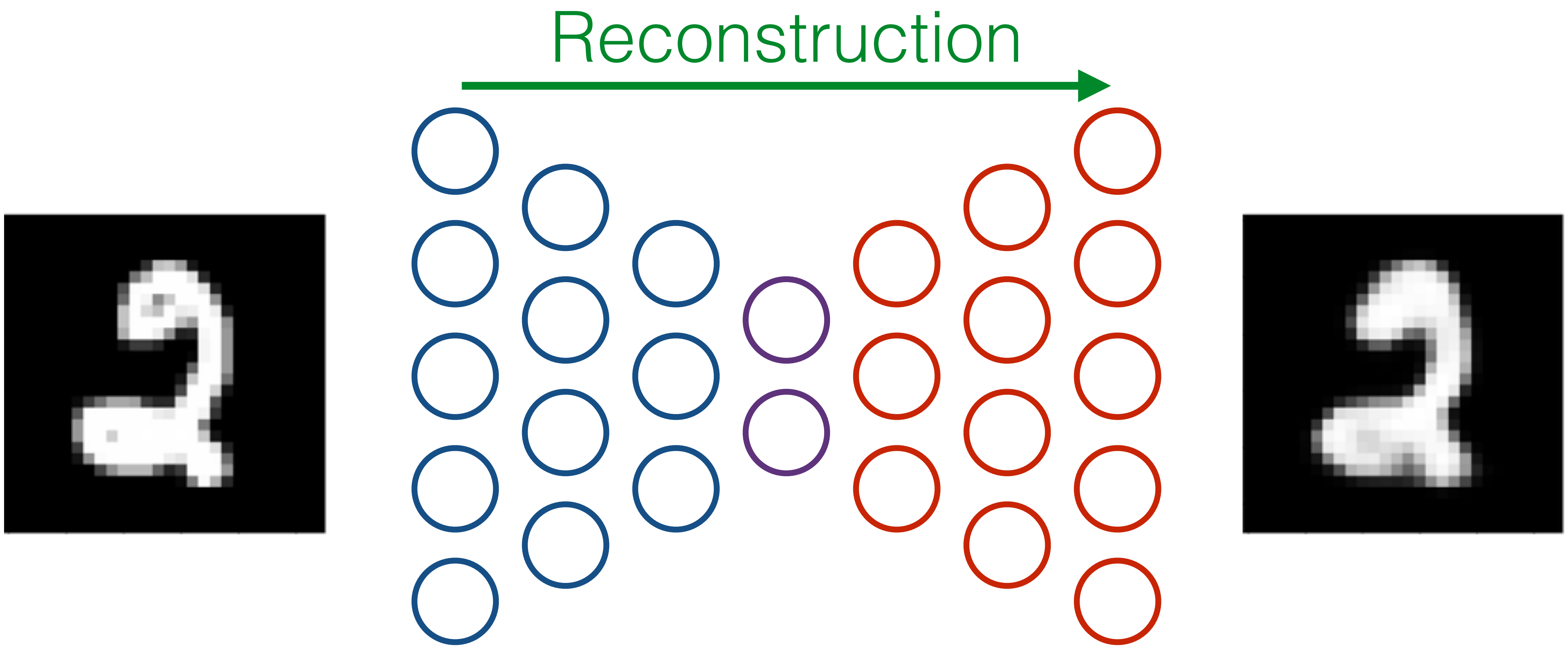
Autoencoder

- Function: training phase
  - Train the model given a batch by minimizing the reconstruction loss



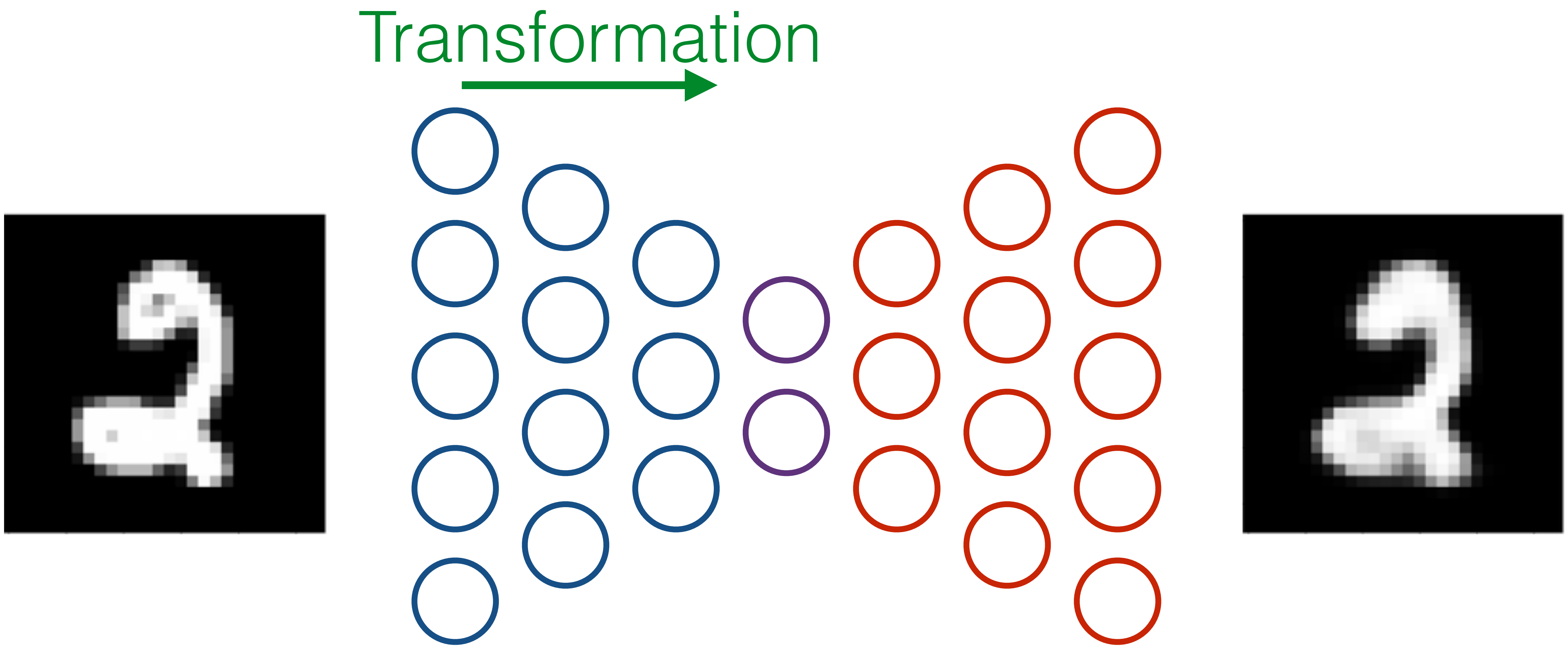
Autoencoder

- Function: testing phase
  - Reconstruction



Autoencoder

- Function: testing phase
  - Transformation

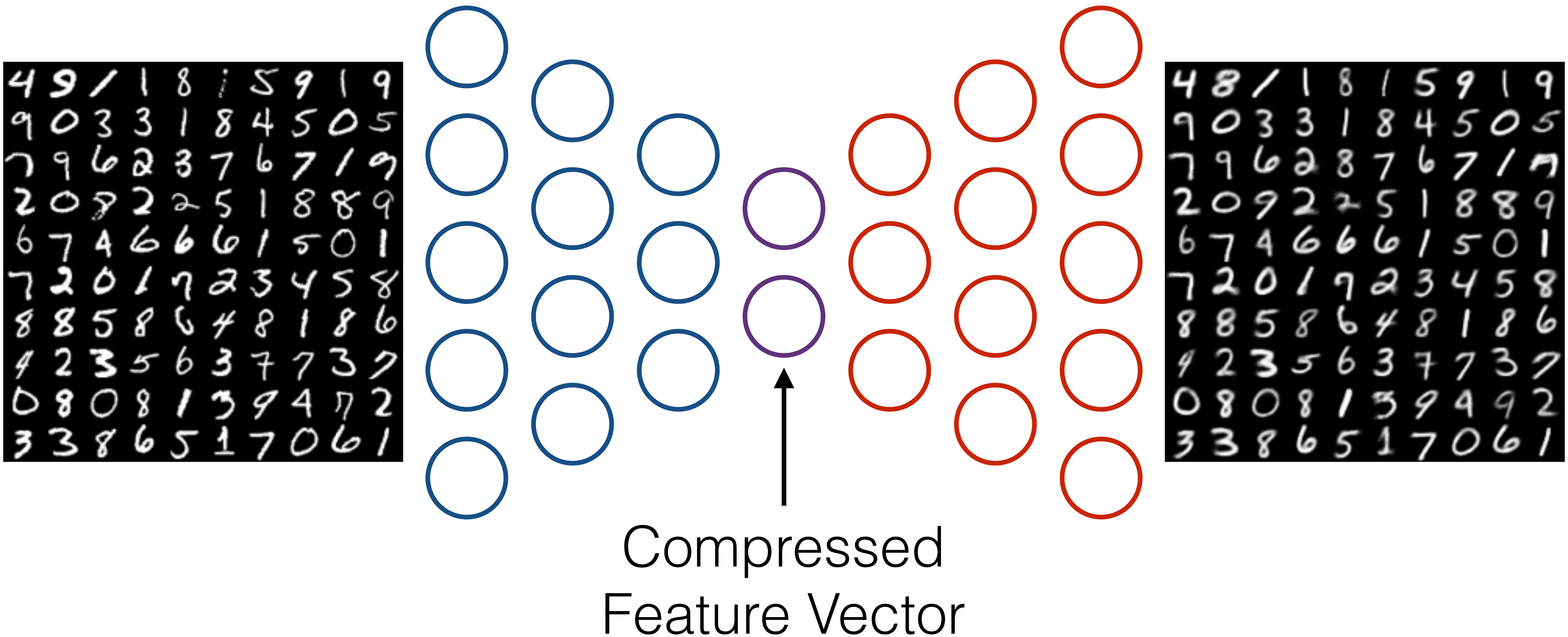


Coding Session

Autoencoder

Autoencoder

- Results
  - Looks pretty good! Doesn't it?



Autoencoder

- What about generation?

Autoencoder

- What about generation?

Images in the dataset

Autoencoder

- What about generation?

Images in the dataset

Autoencoder

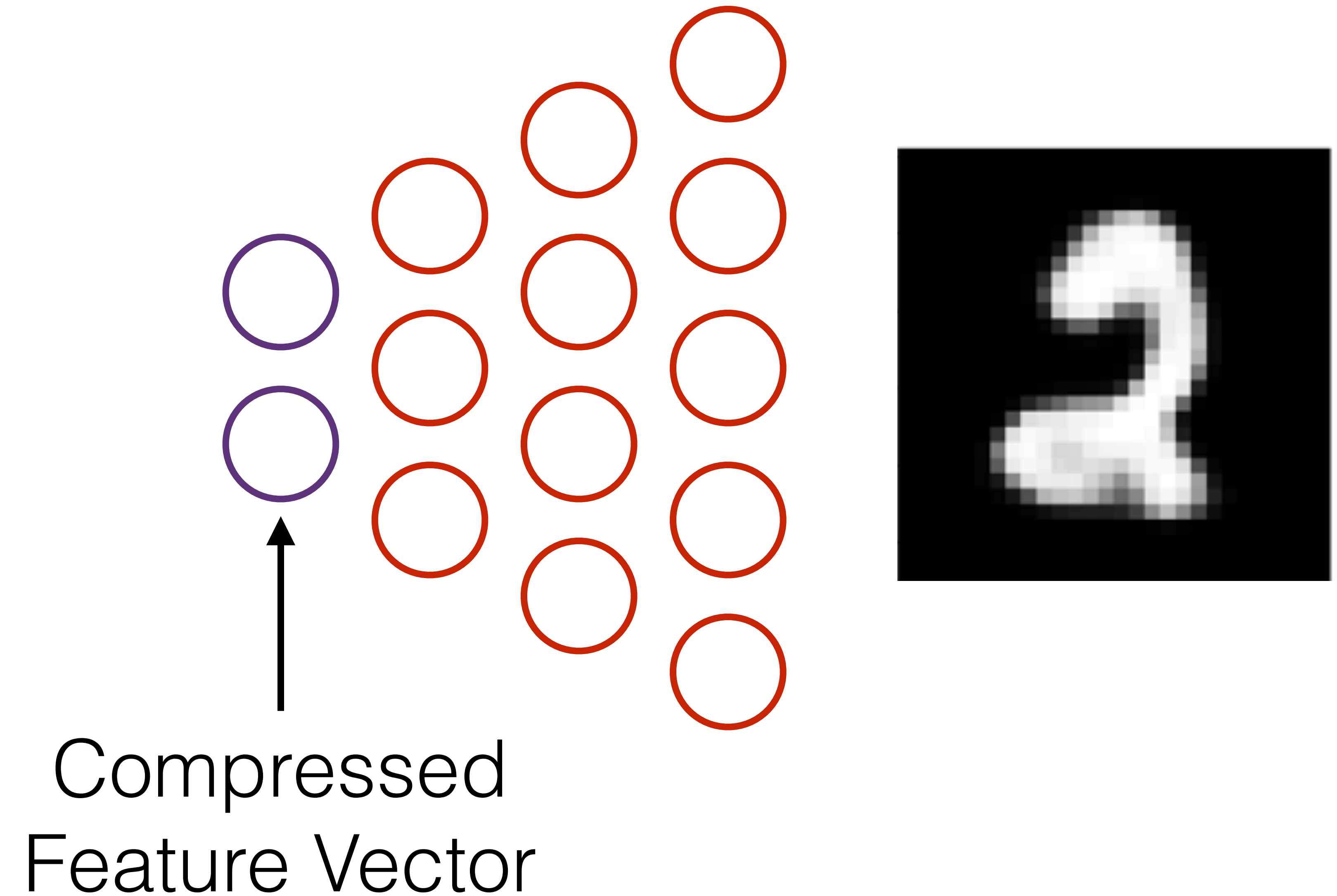
- What about generation?

Images in the dataset

Generated images

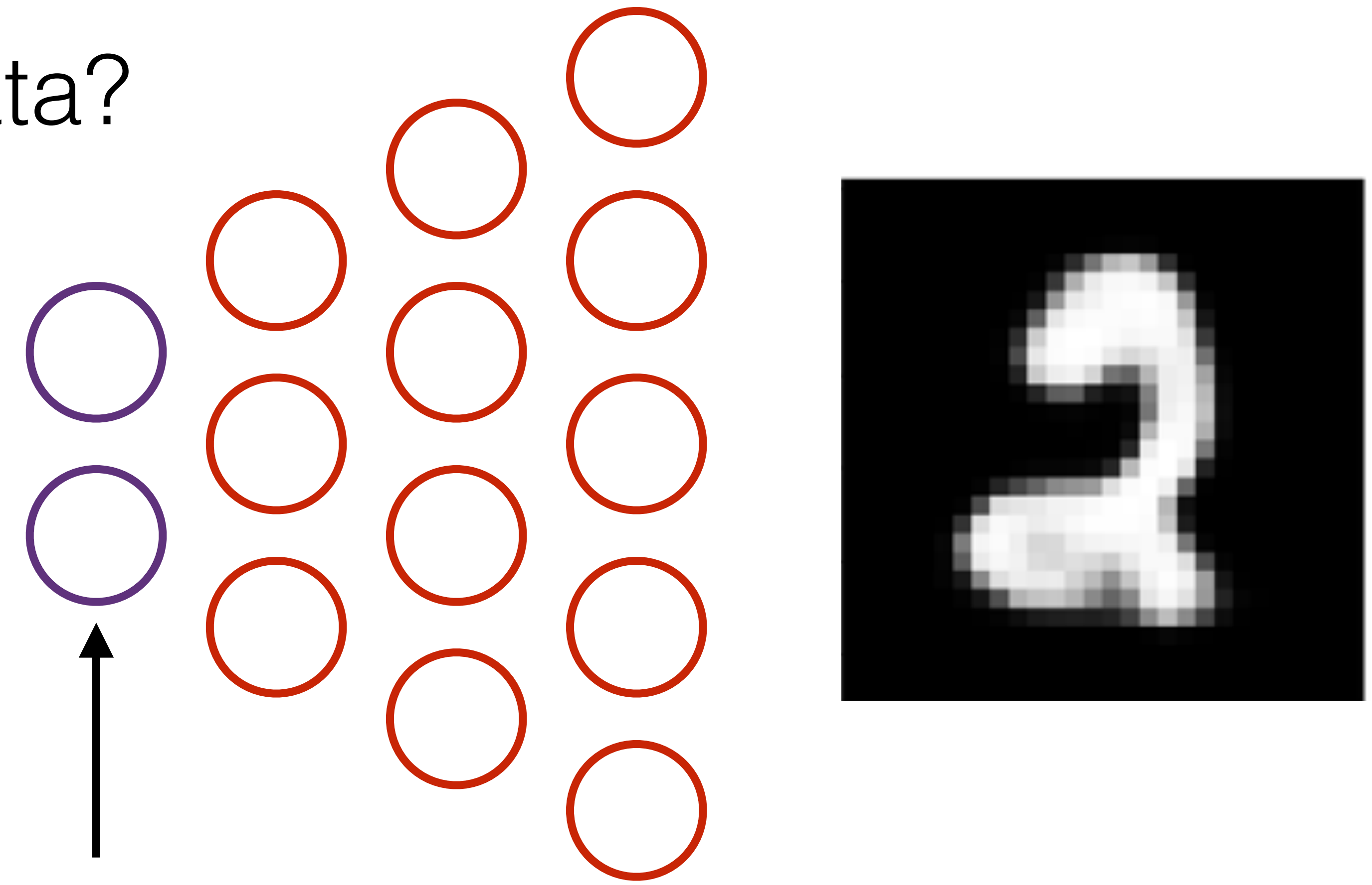
Autoencoder

- What about generation?



Autoencoder

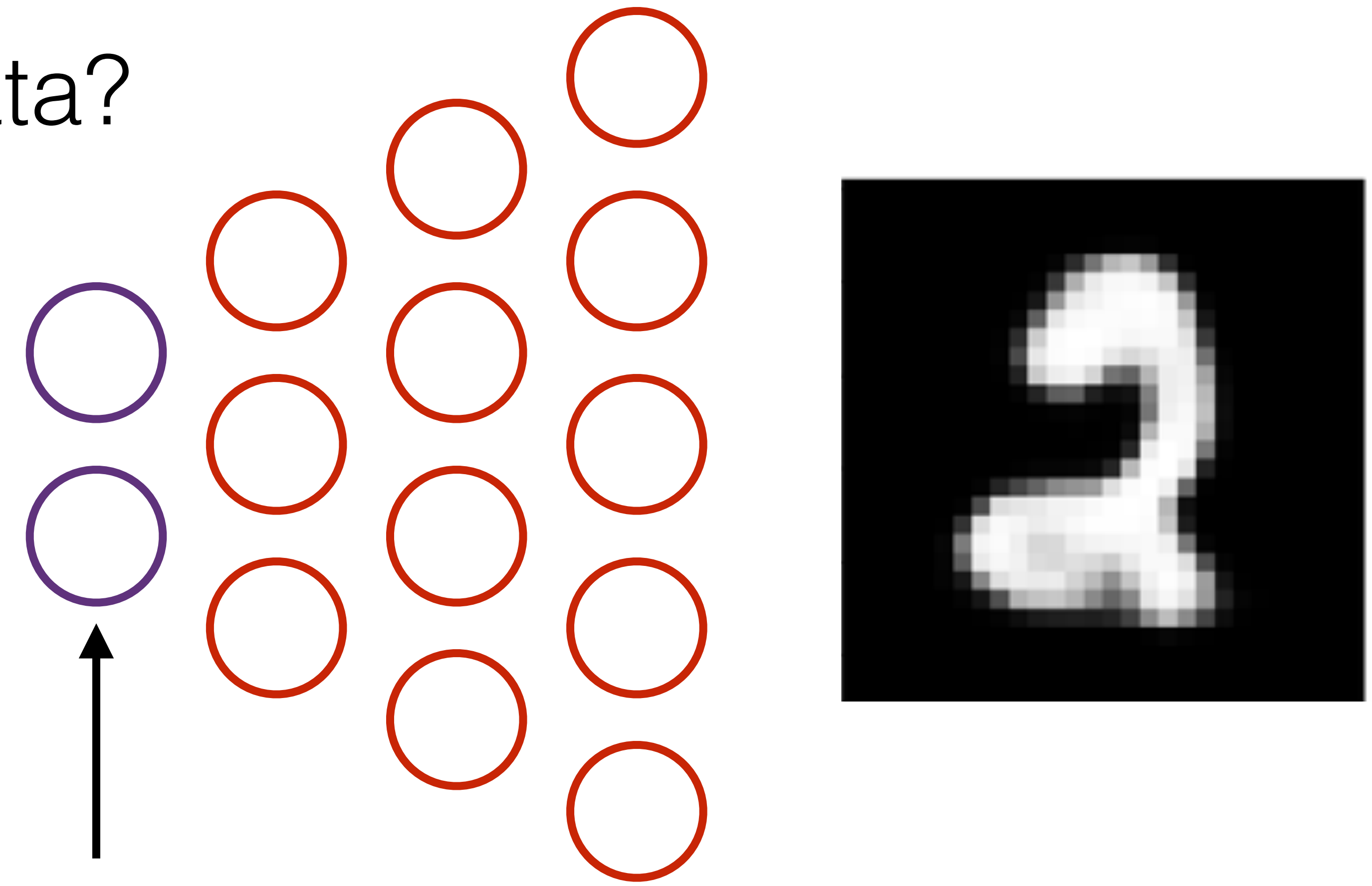
- What about generation?
  - Given feature vectors
  - Can we generate data?



Compressed
Feature Vector

Autoencoder

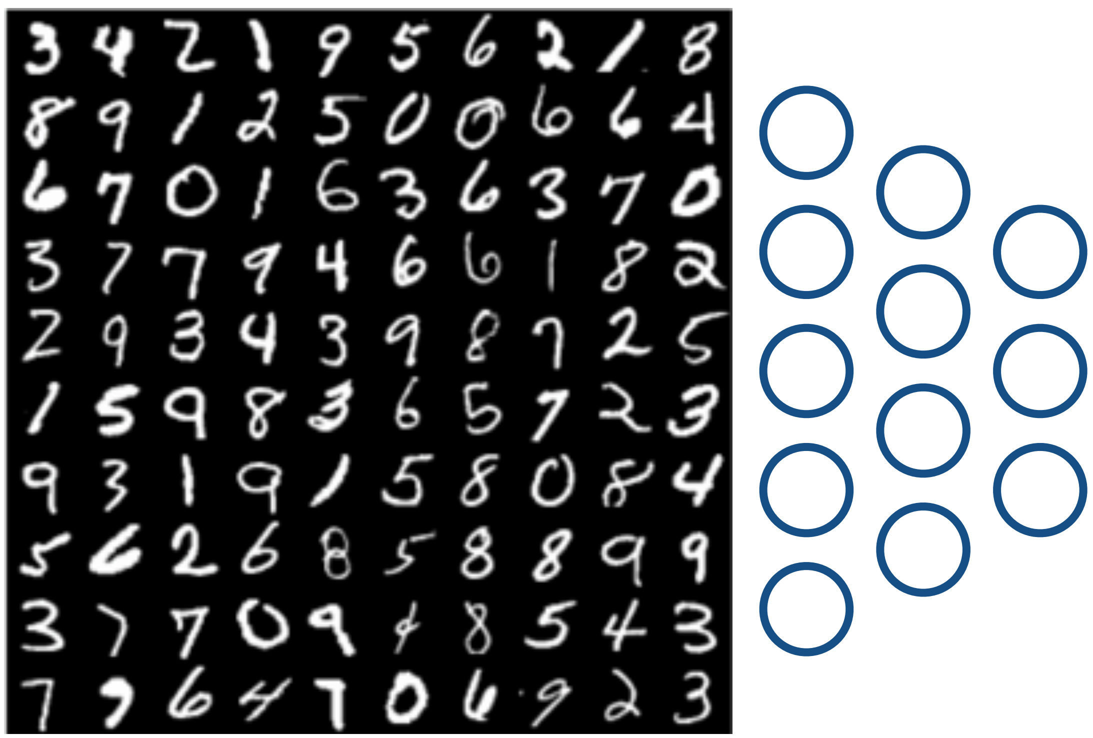
- What about generation?
    - Given feature vectors
    - Can we generate data?
- No!
    - What are the valid feature vectors?



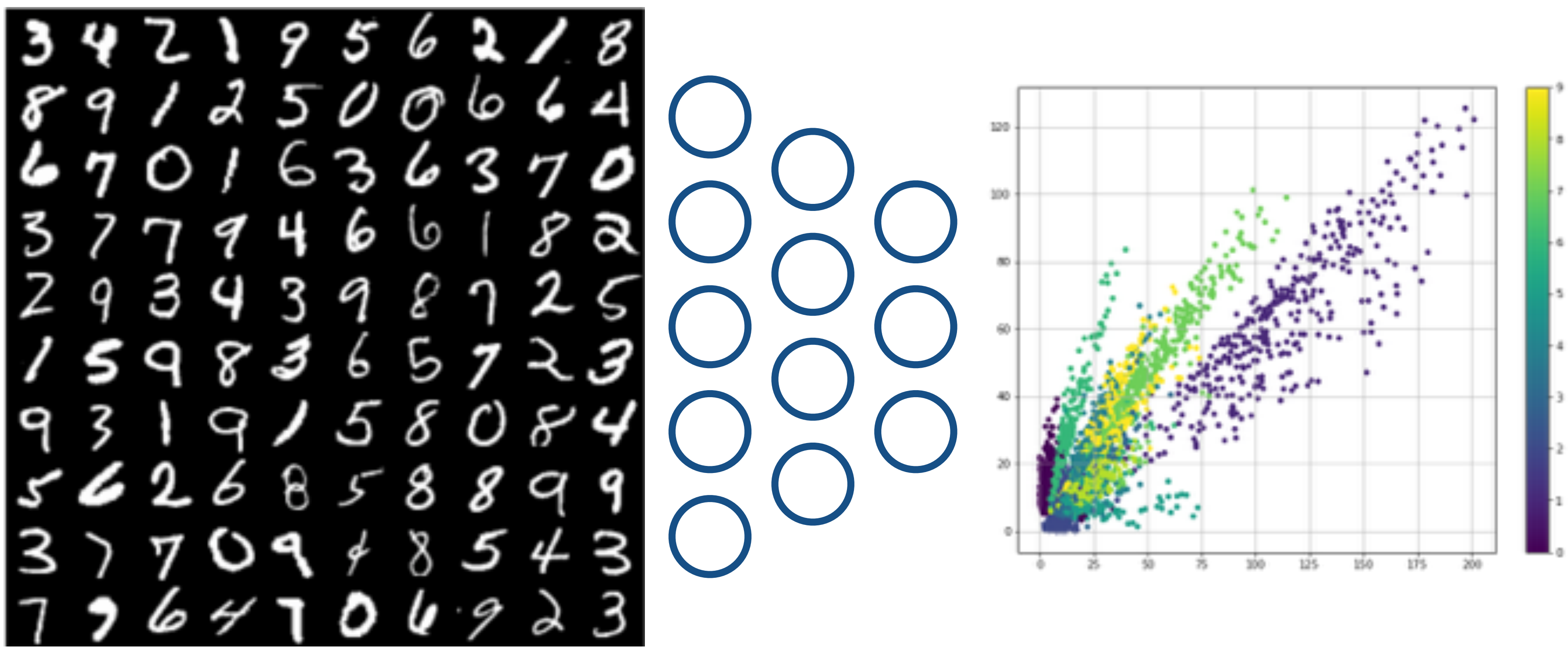
Compressed Feature Vector

Autoencoder

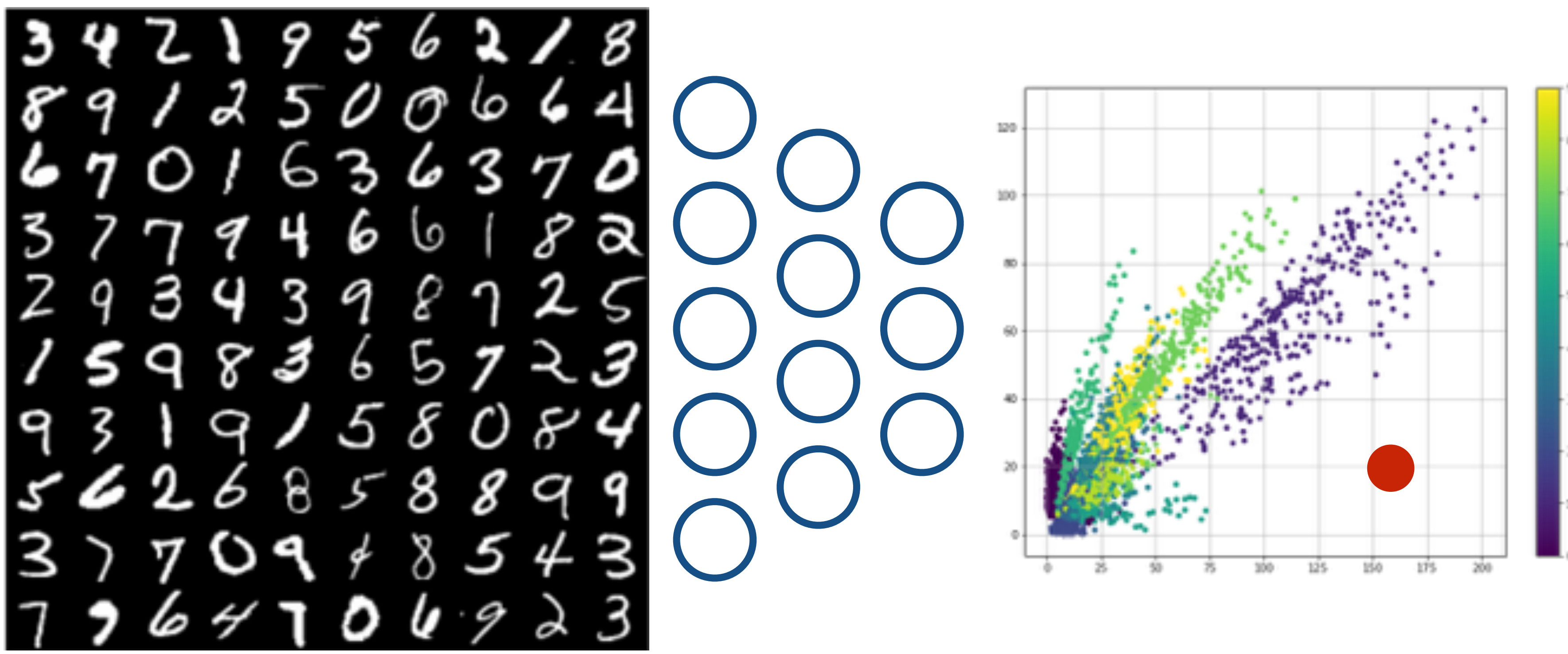
Autoencoder



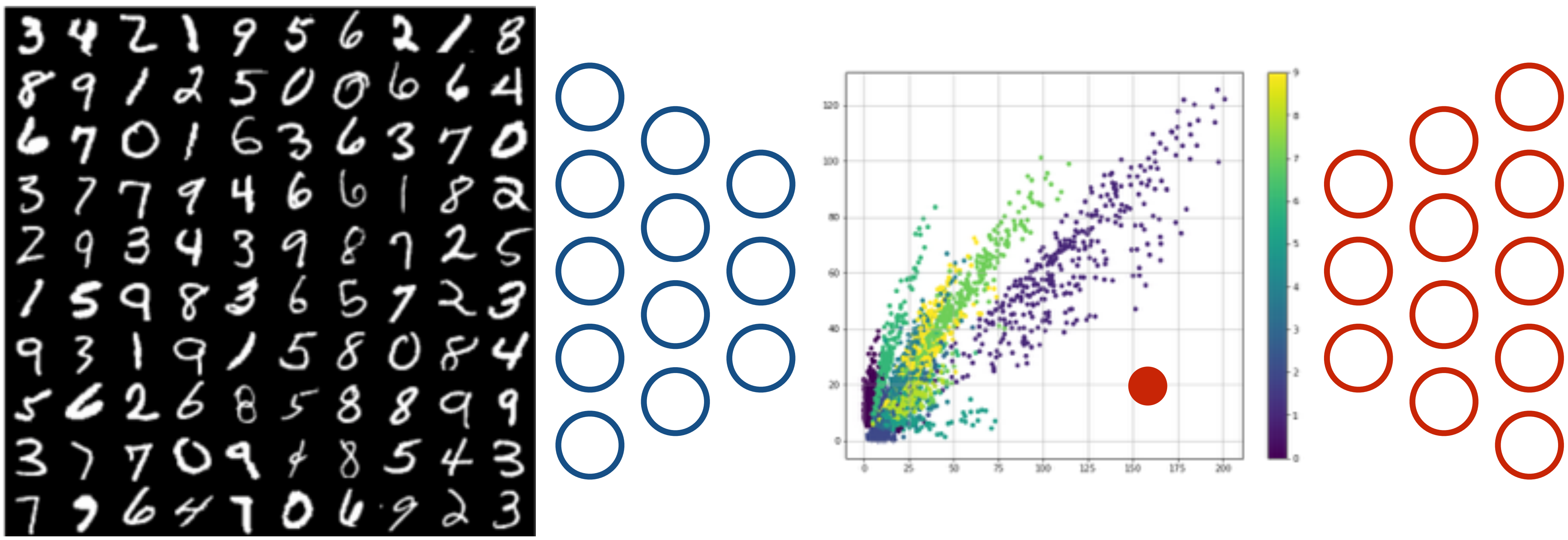
Autoencoder



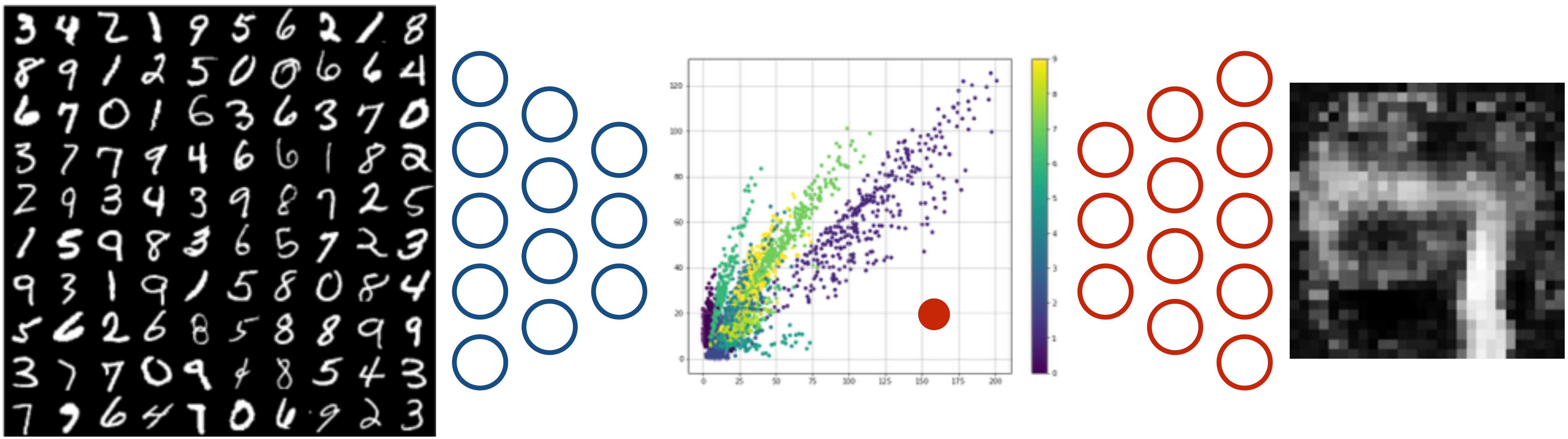
Autoencoder



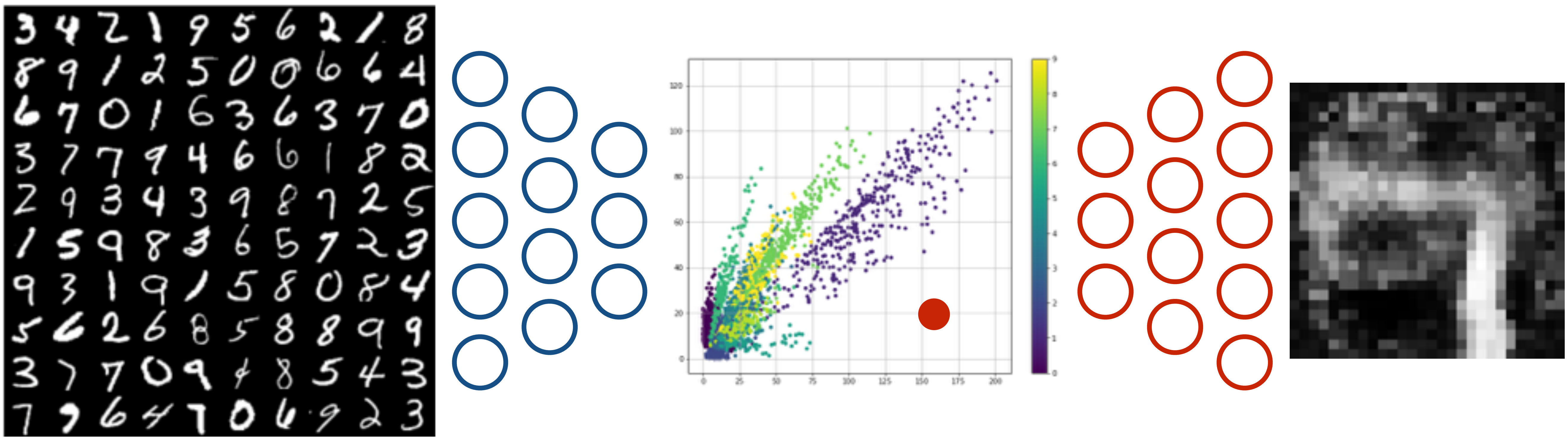
Autoencoder



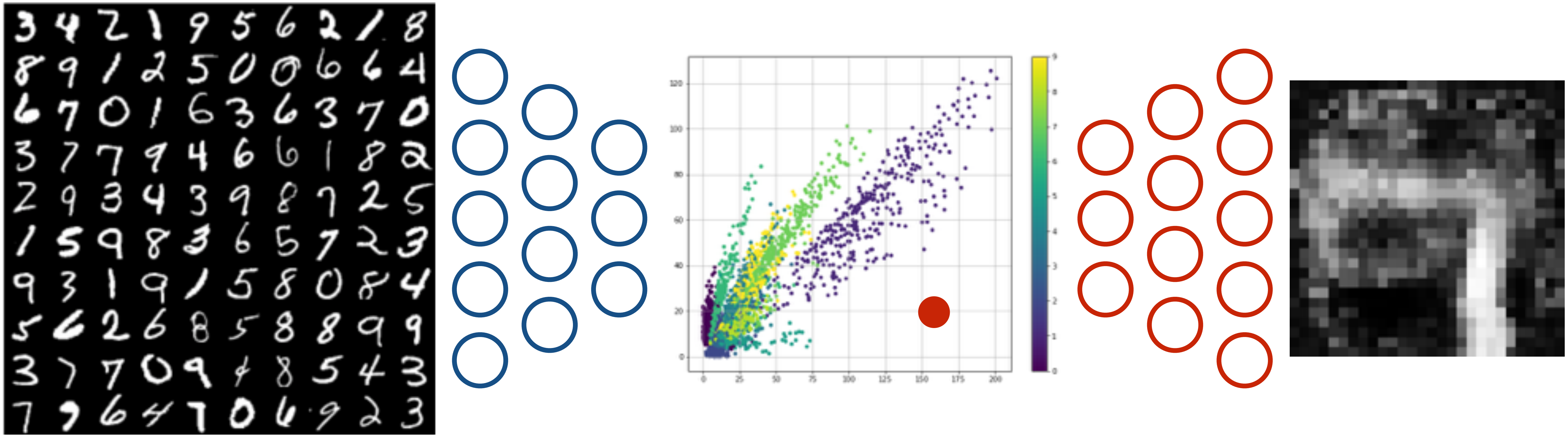
Autoencoder



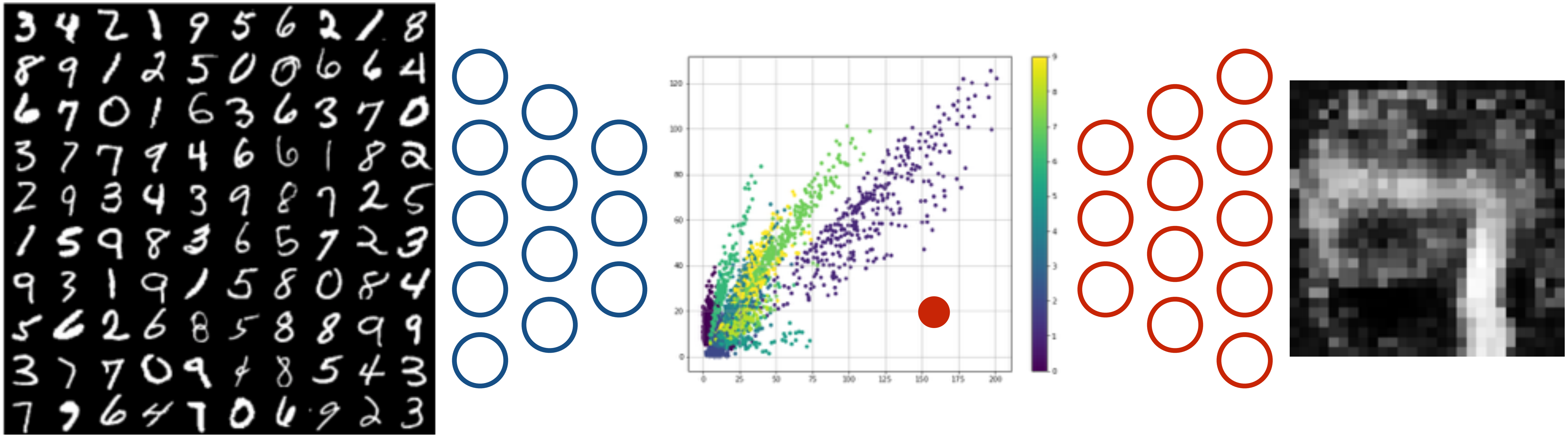
Autoencoder



Autoencoder

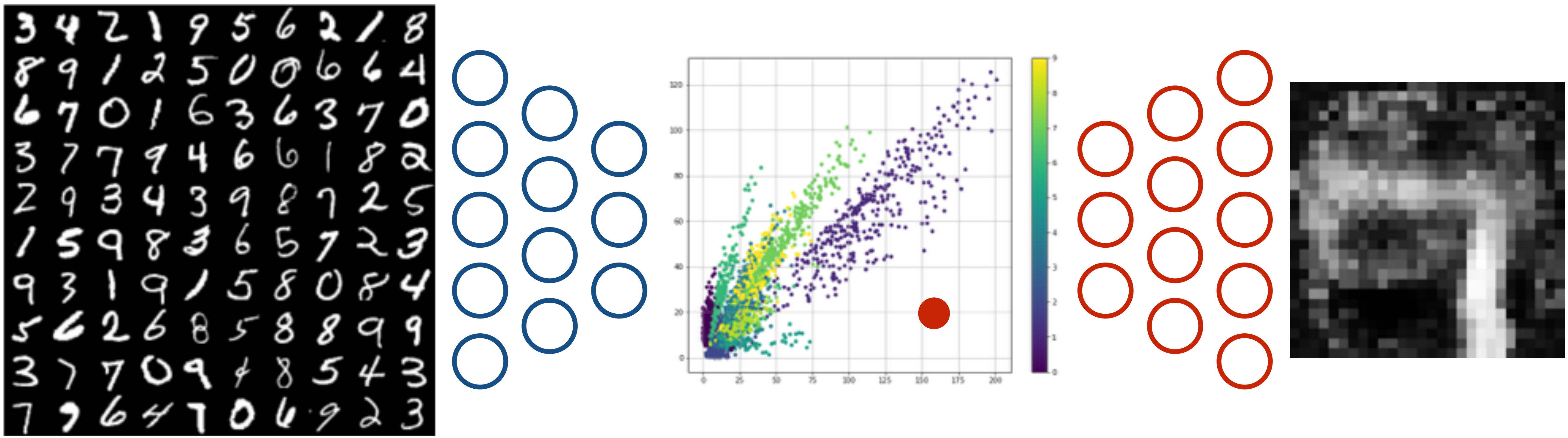


Autoencoder



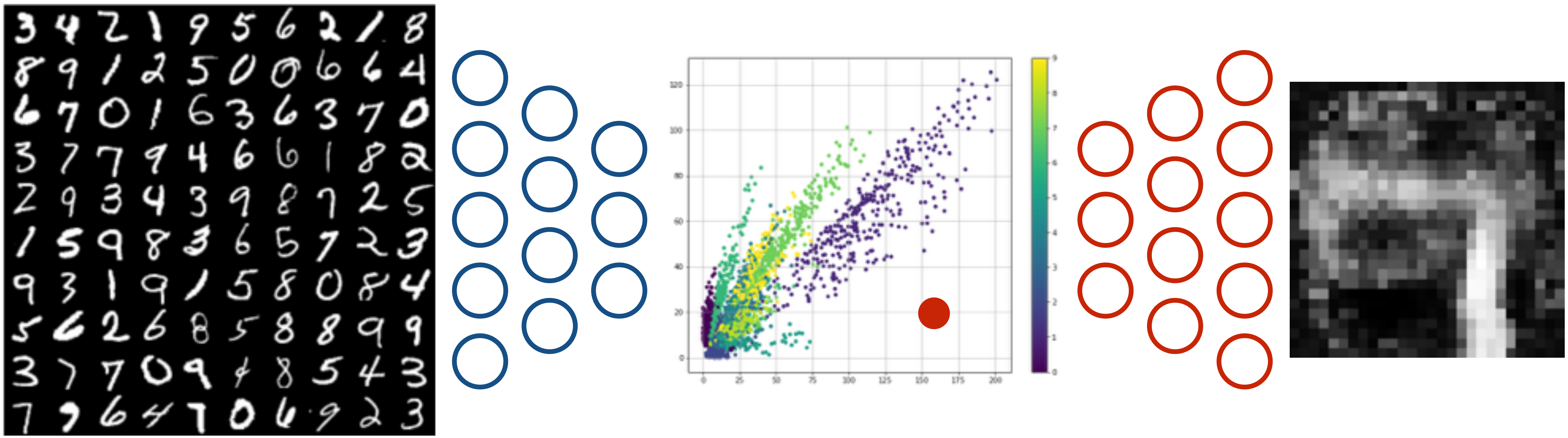
What if…

Autoencoder

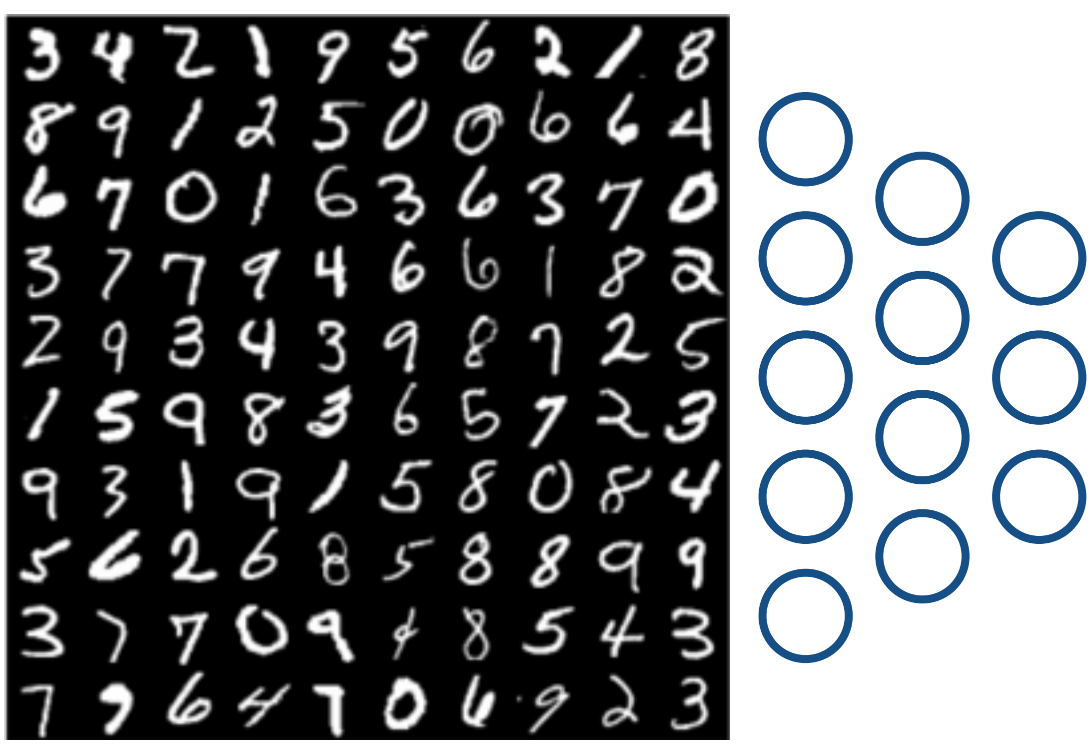


What if…

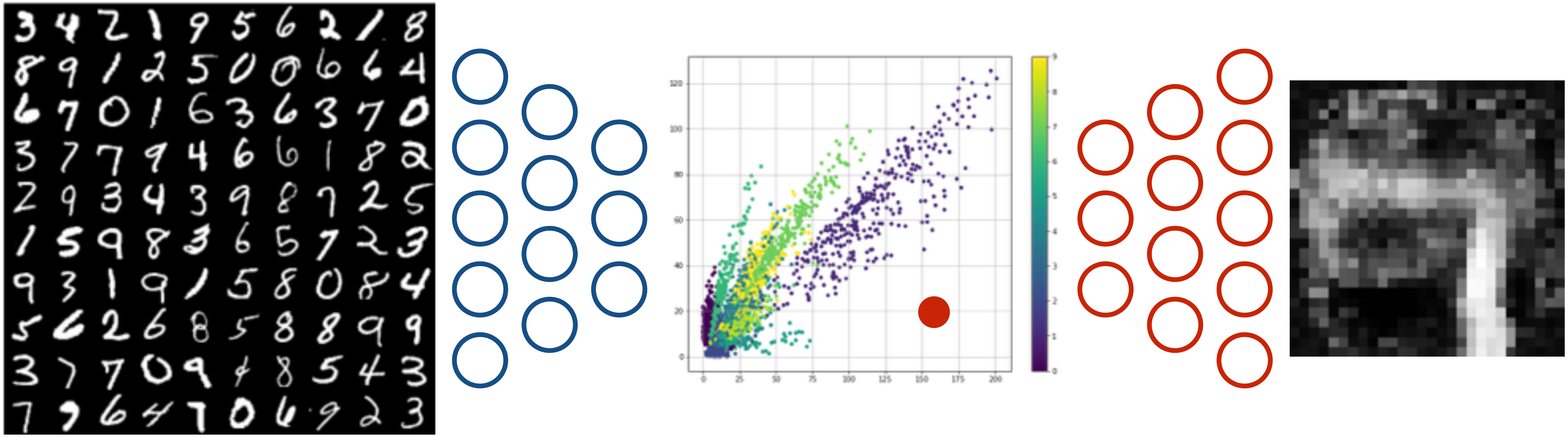
Autoencoder



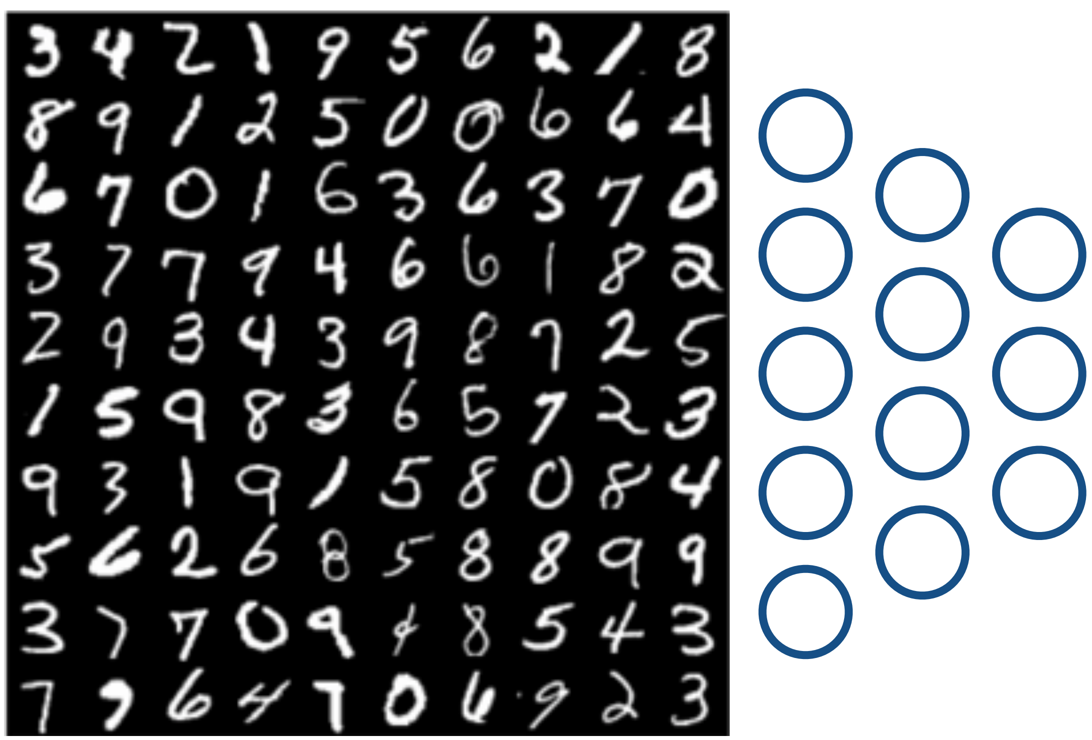
What if…

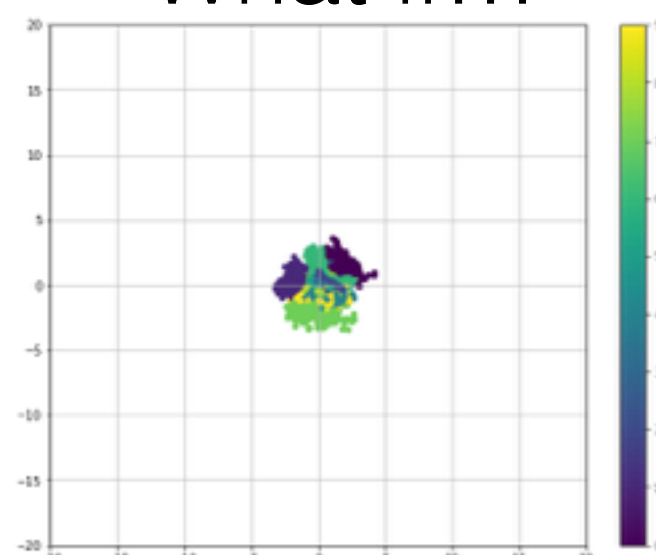


Autoencoder

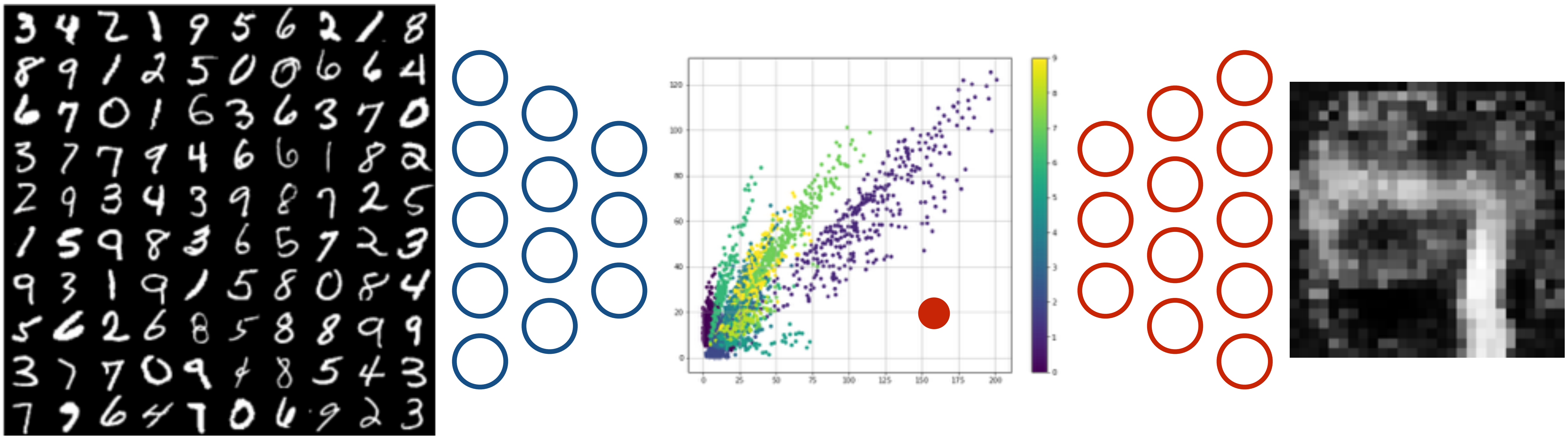


What if…

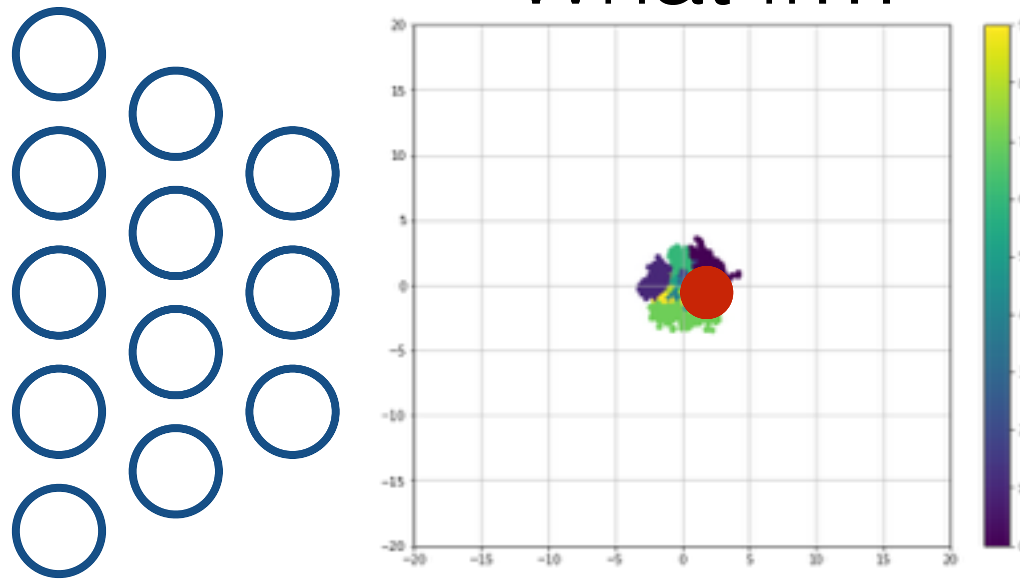




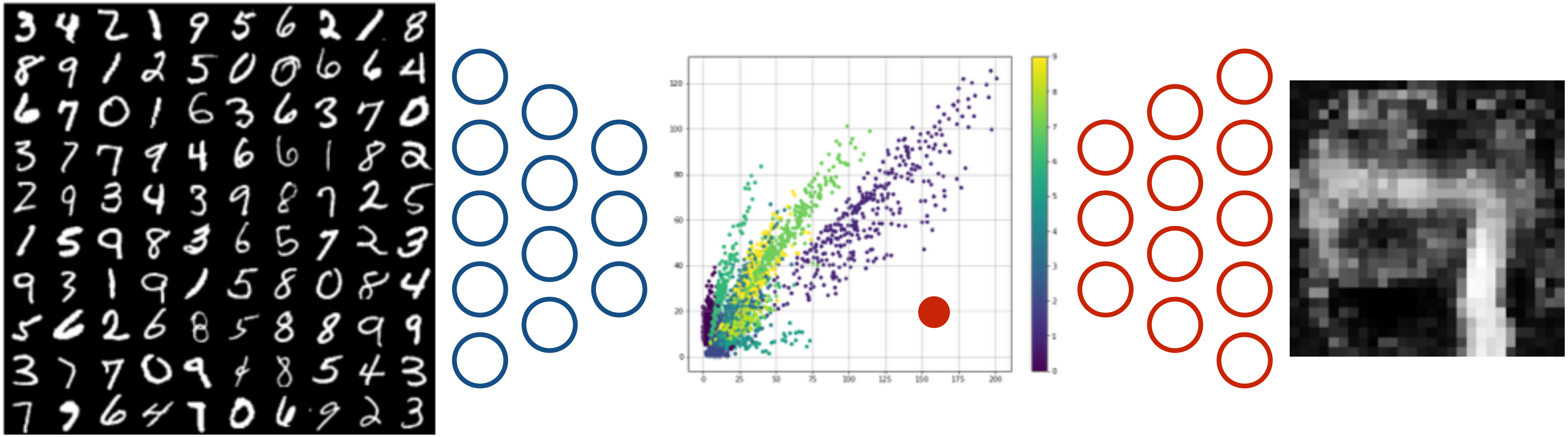
Autoencoder



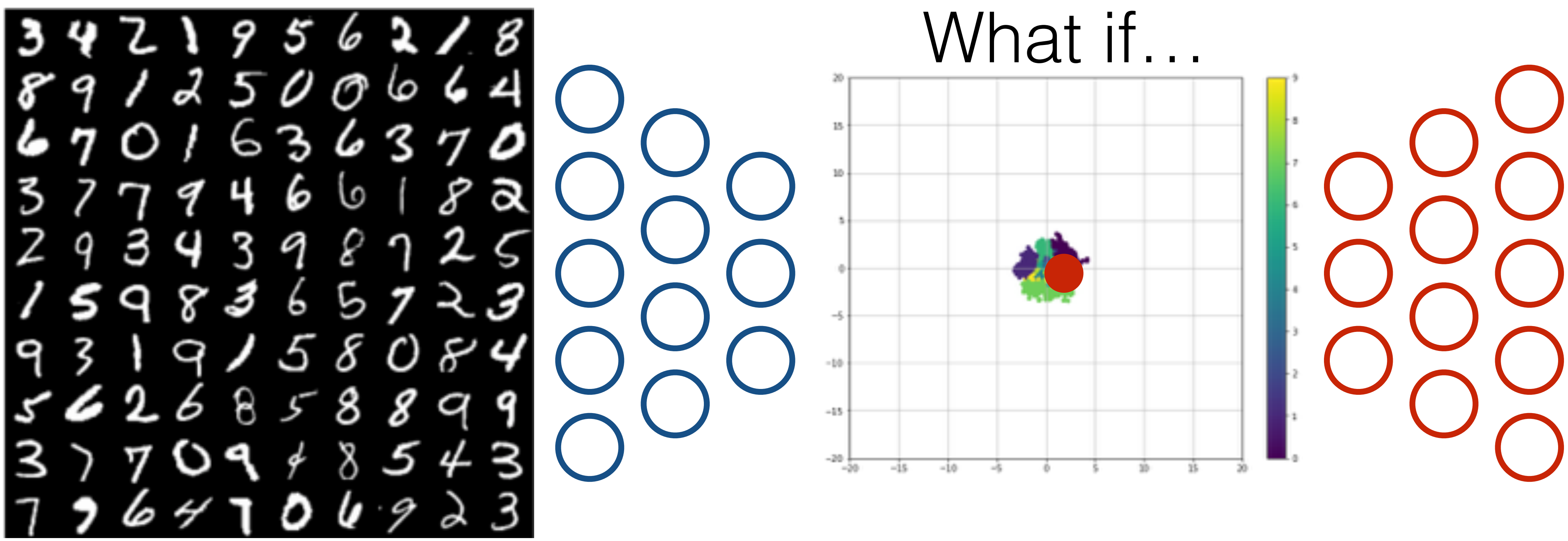
What if…



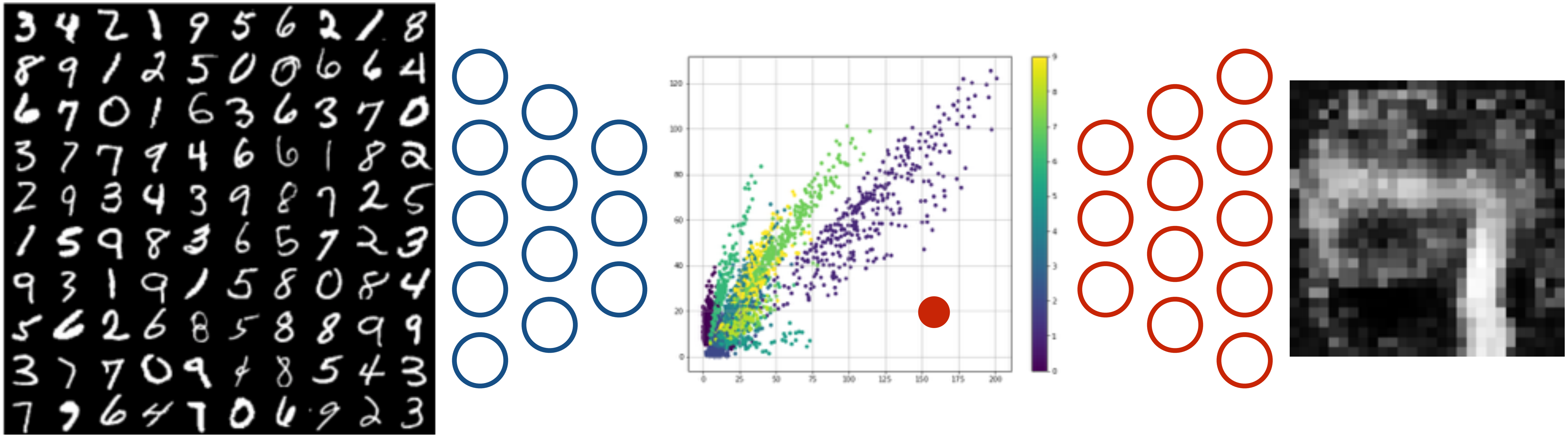
Autoencoder



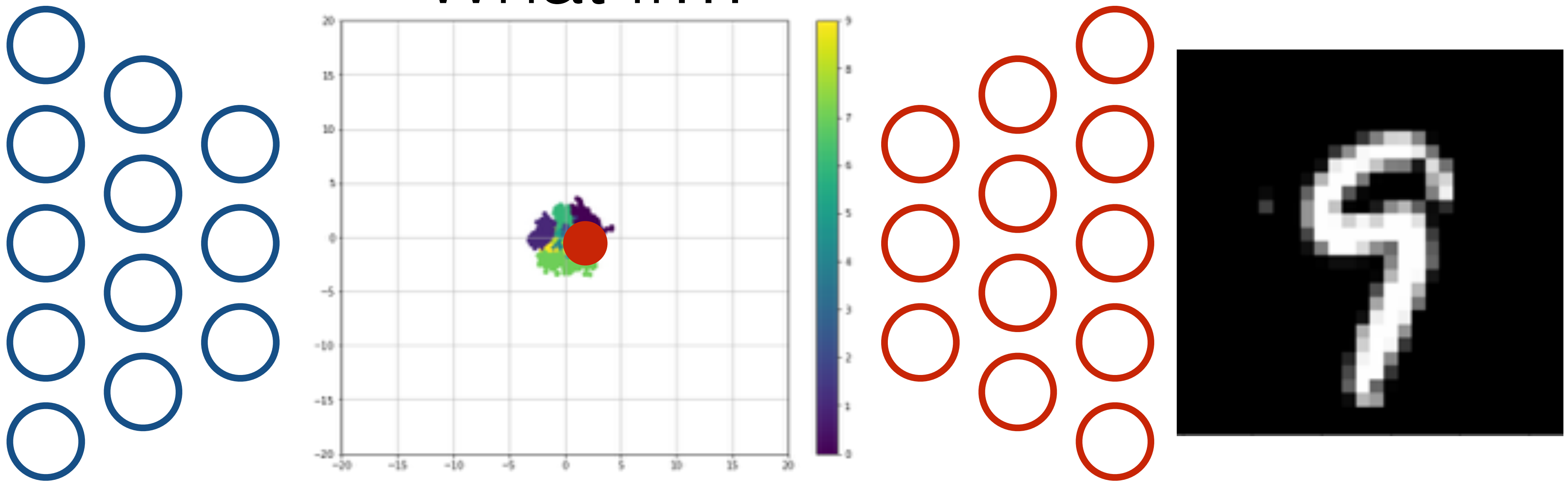
What if…



Autoencoder

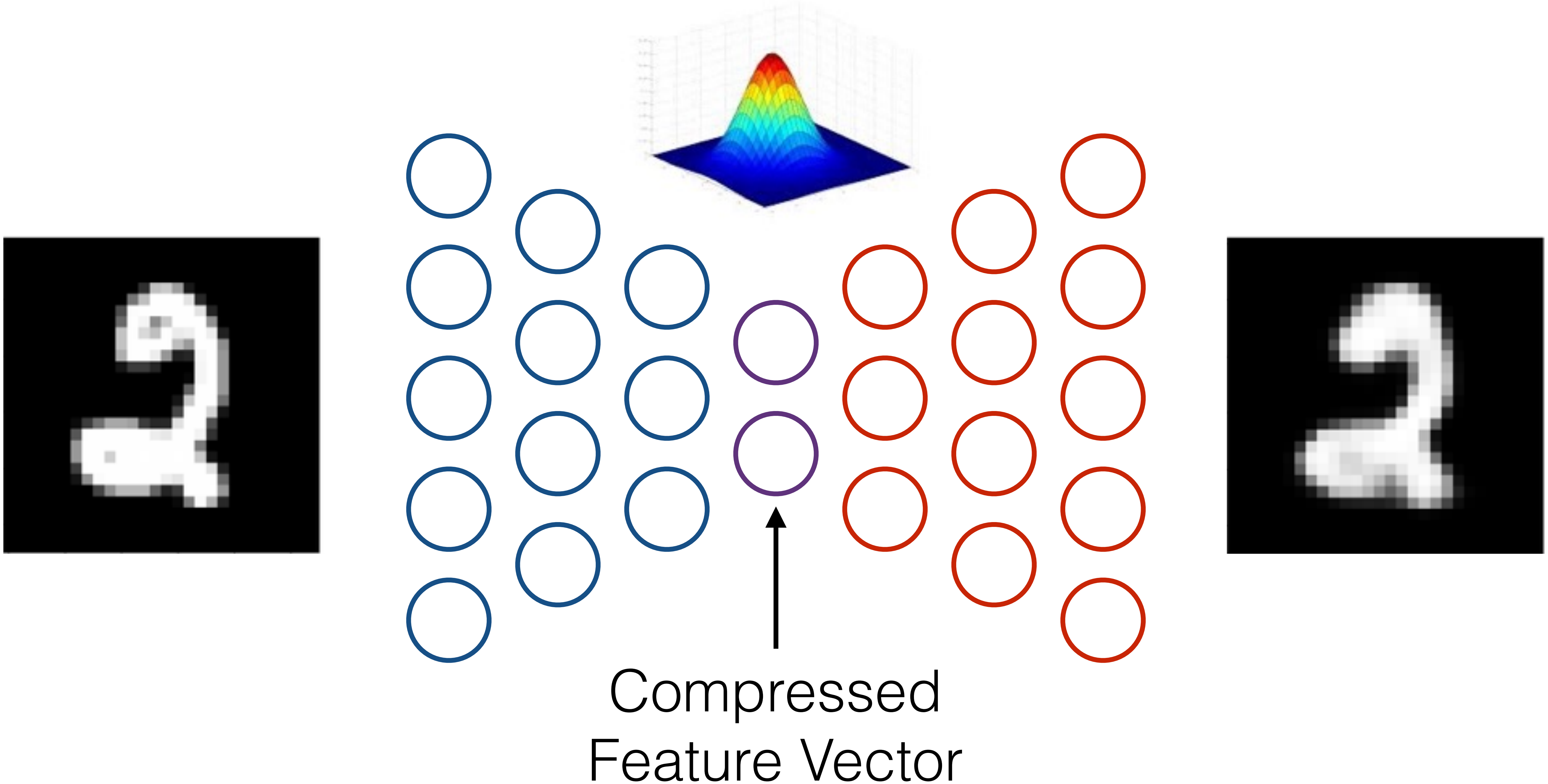


What if…



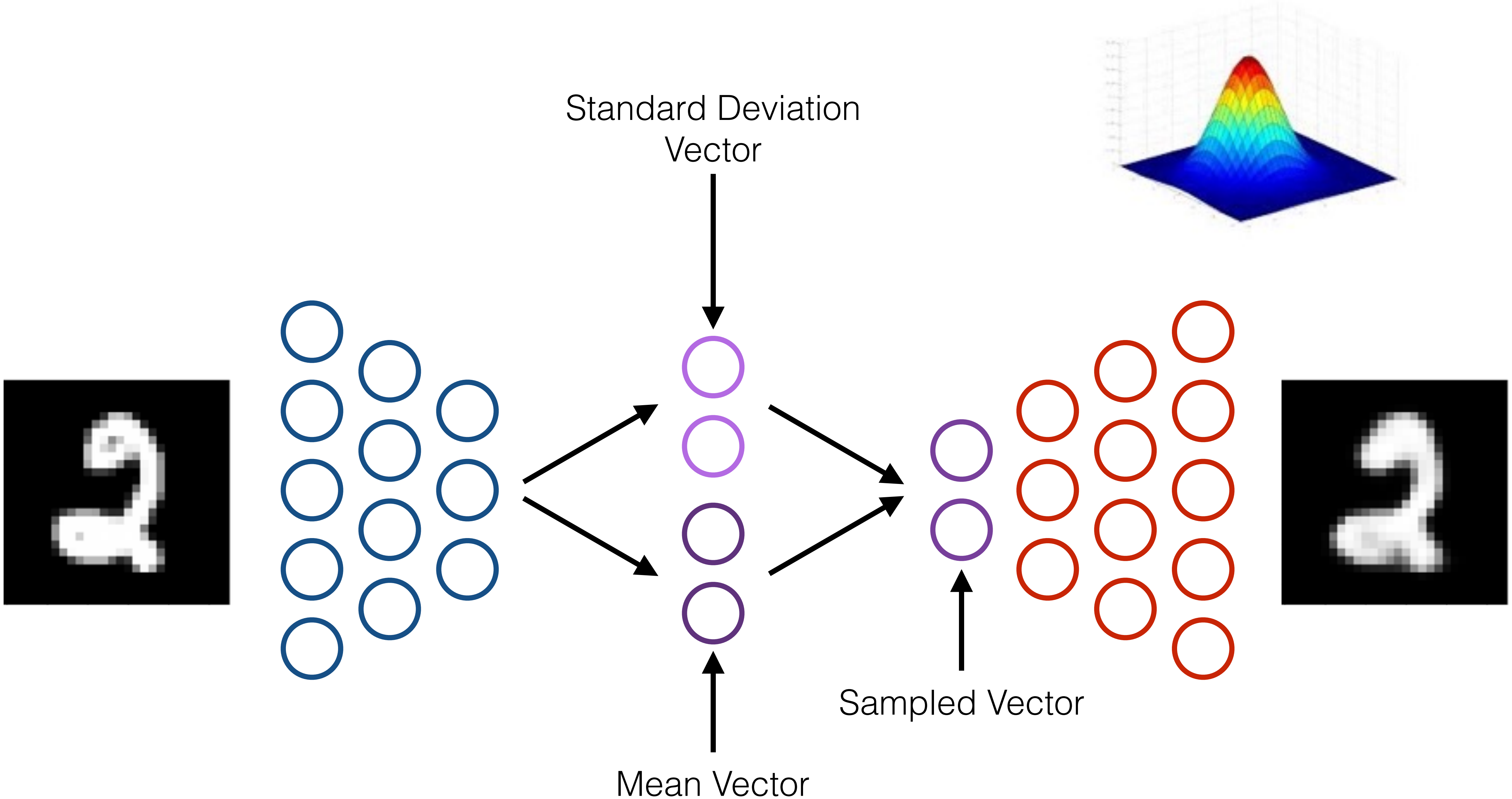
Variational Autoencoder

- Constrain the latent distribution



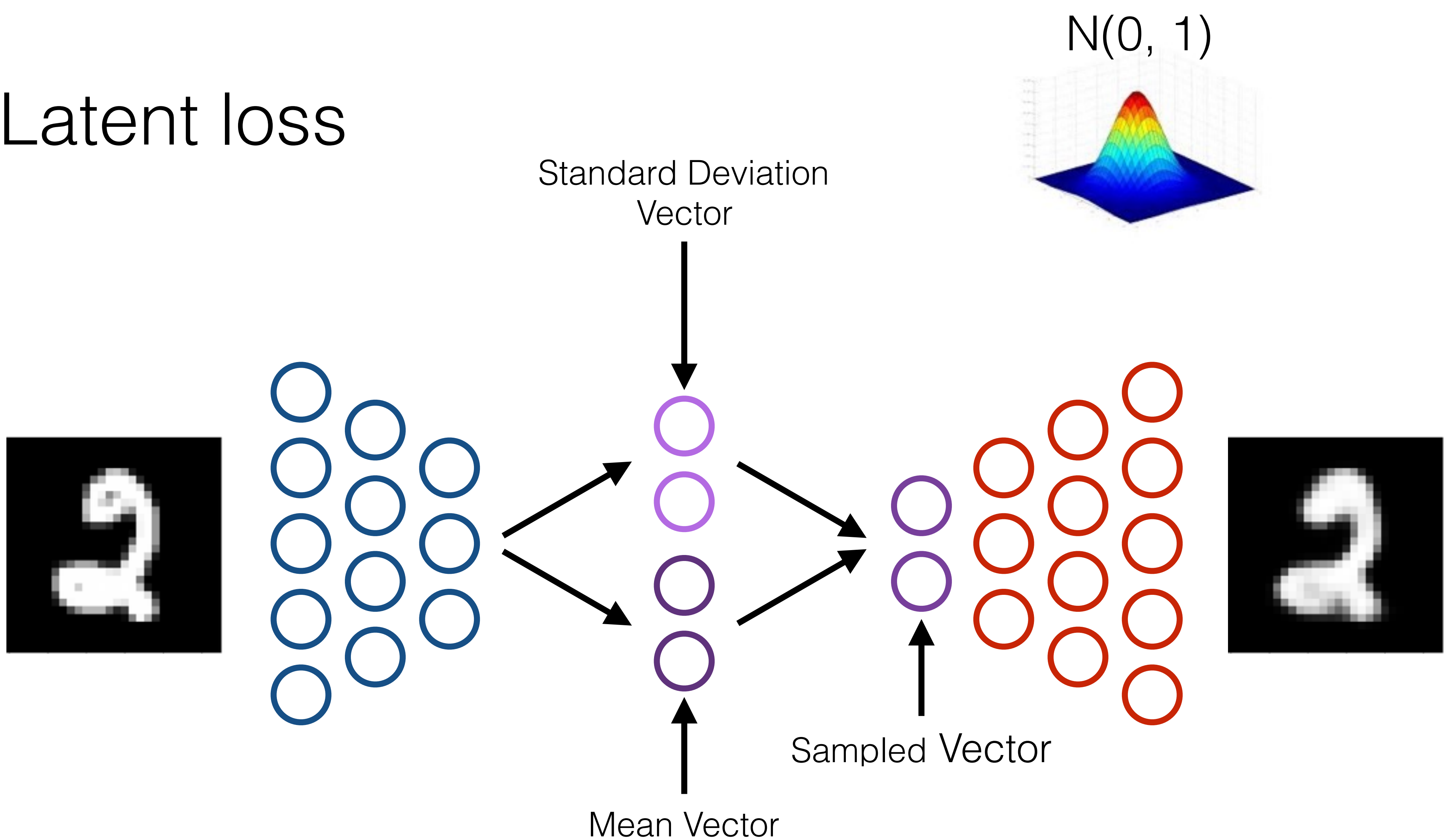
Variational Autoencoder

- Constrain the latent distribution: a Gaussian distribution



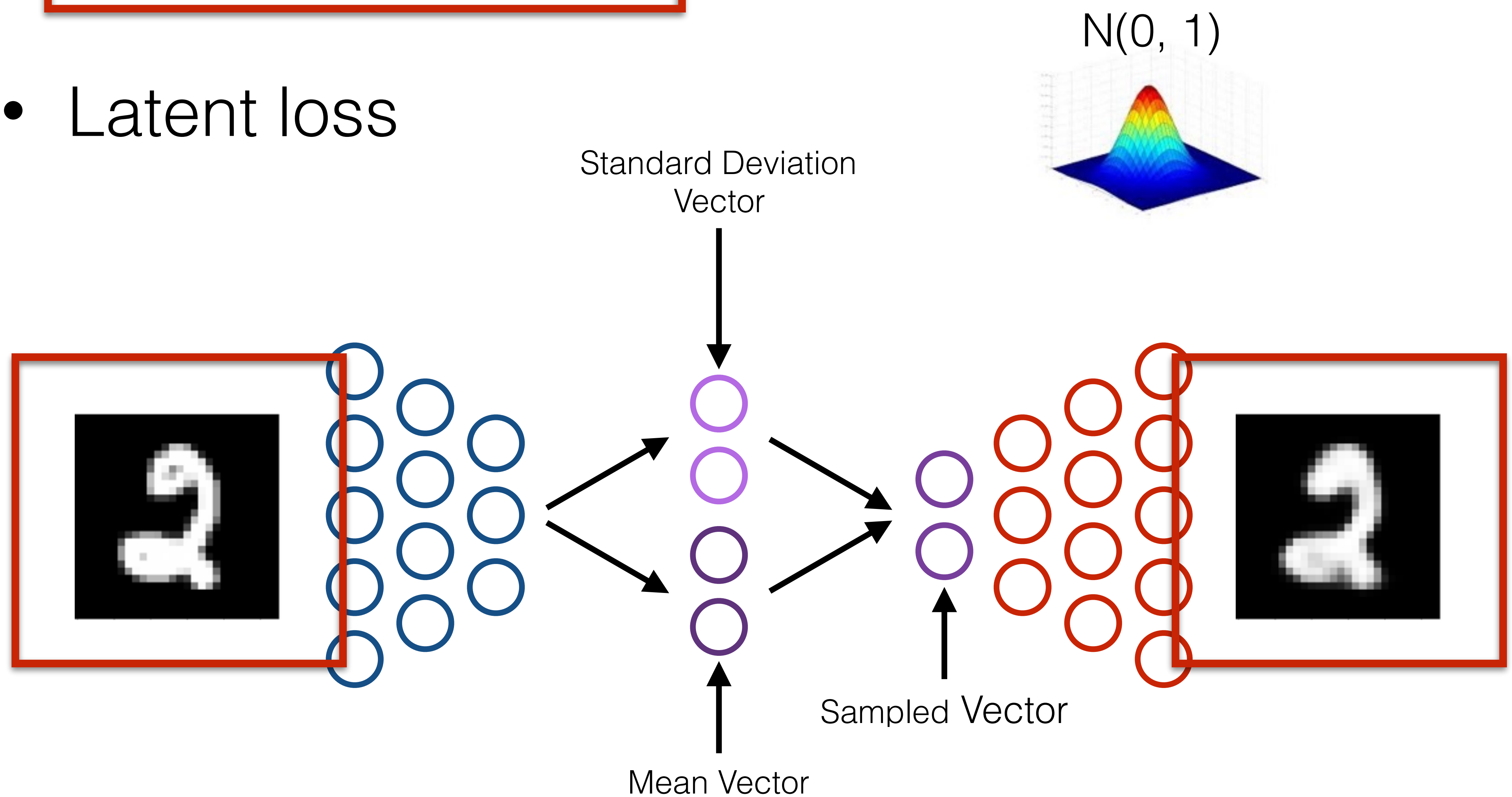
Variational Autoencoder

- Loss
  - Reconstruction loss
  - Latent loss



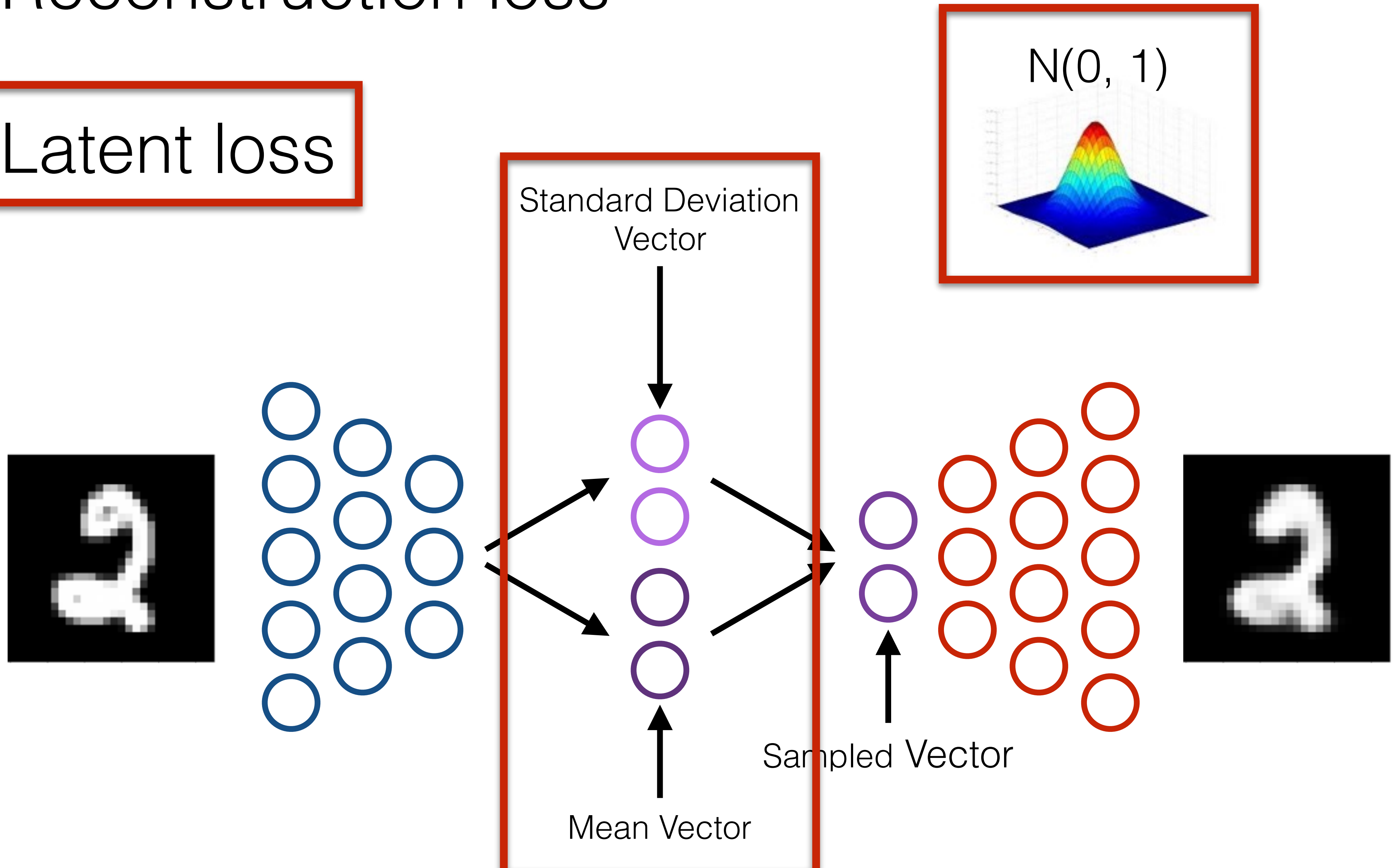
Variational Autoencoder

- Loss
  - Reconstruction loss
  - Latent loss



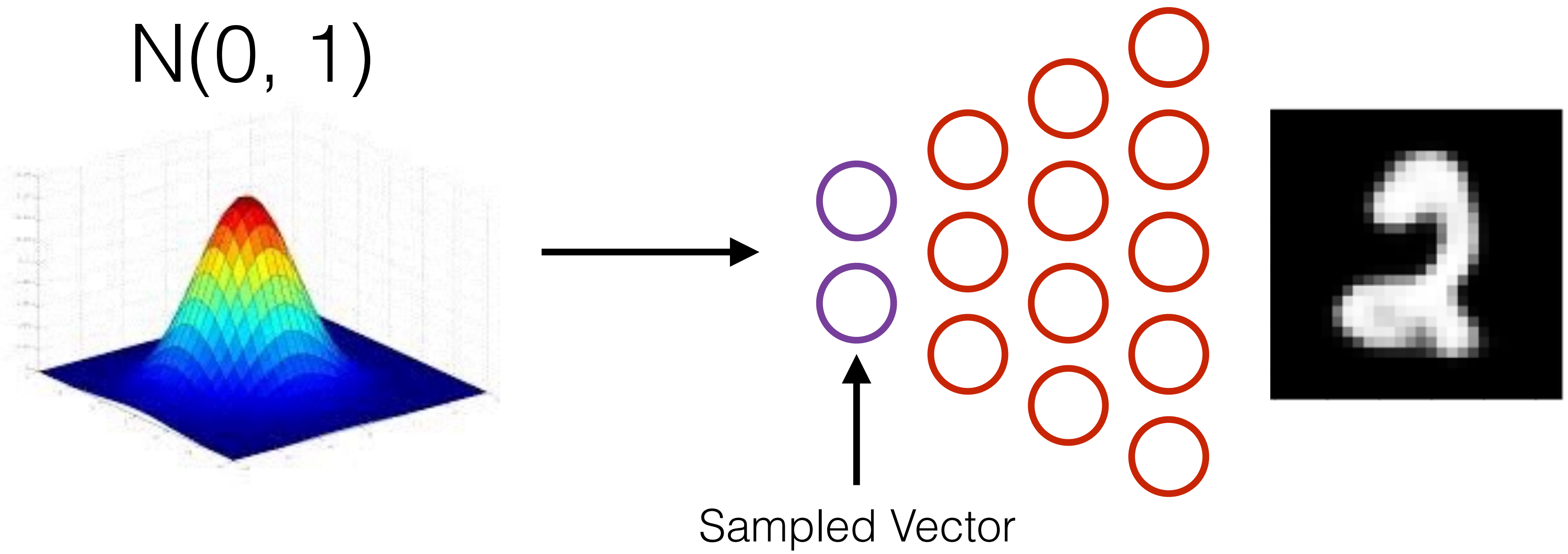
Variational Autoencoder

- Loss
  - Reconstruction loss
  - Latent loss



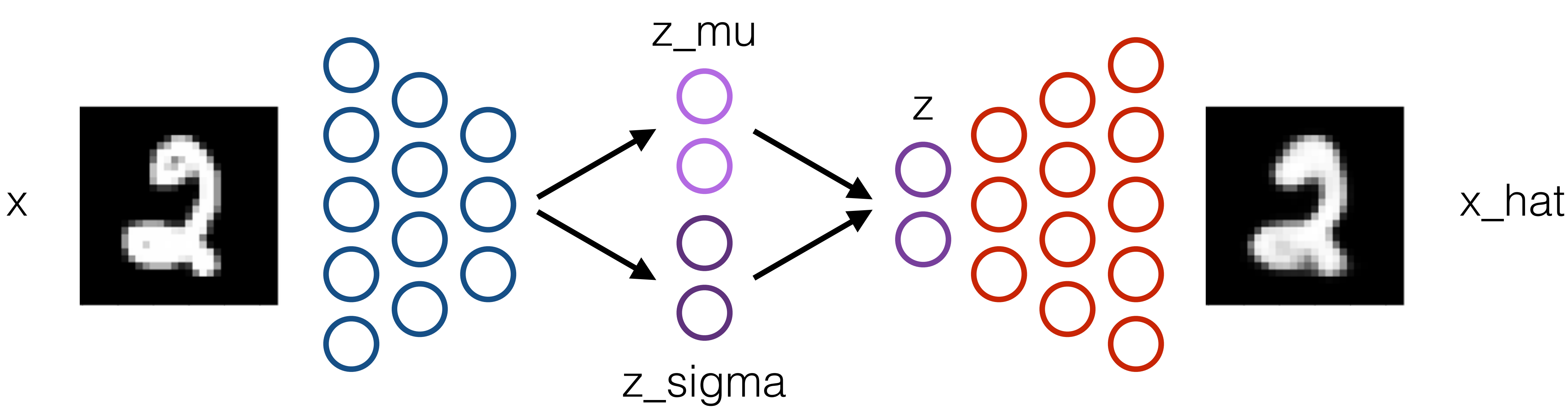
Variational Autoencoder

- Generation



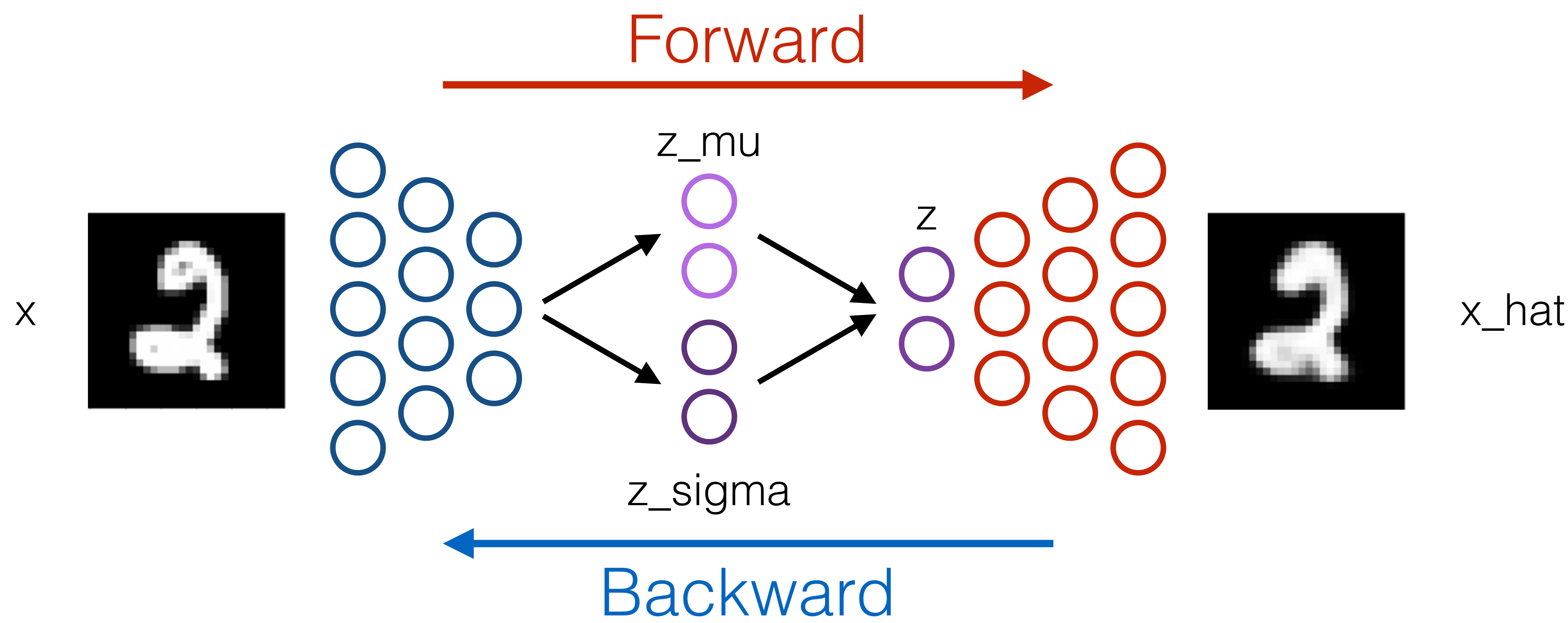
Variational Autoencoder

- Model
  - Decoder (3 layers)
  - Encoder (3 layers)



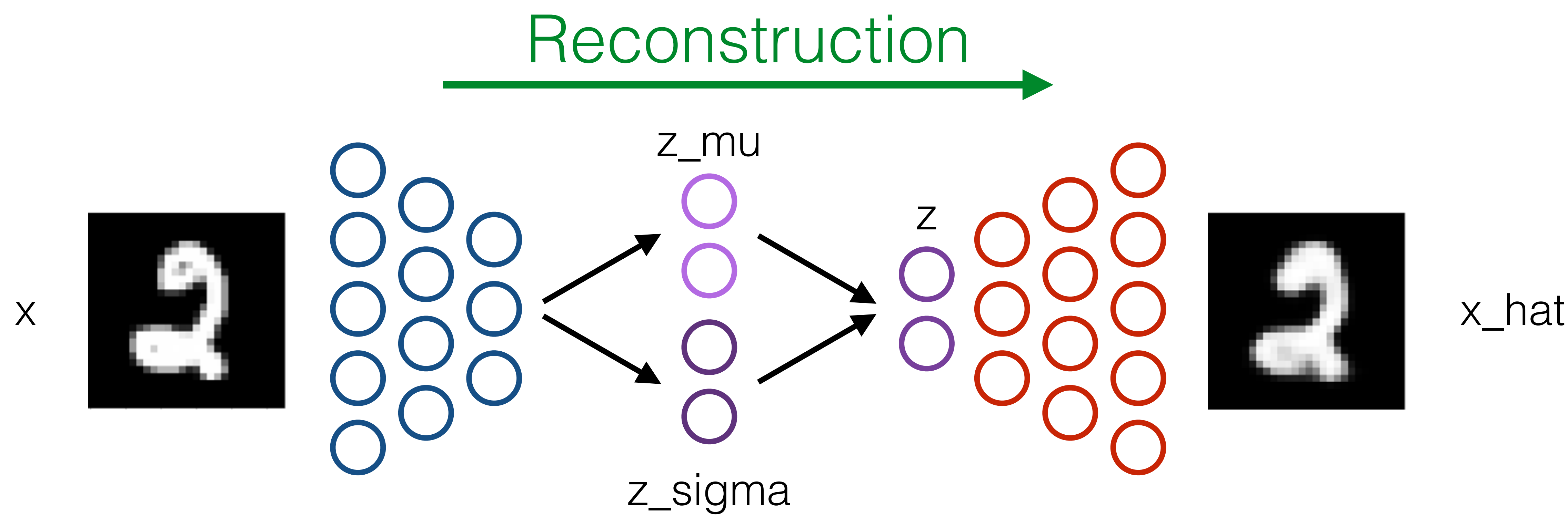
Variational Autoencoder

- Function: training phase
  - Train the model given a batch



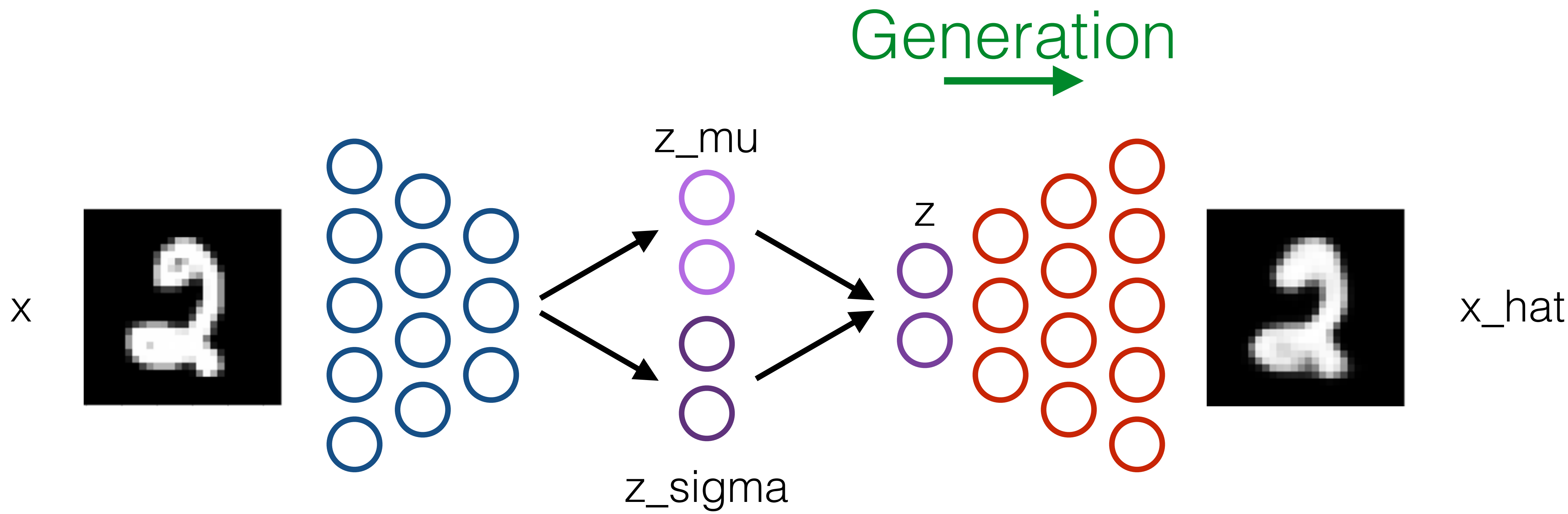
Variational Autoencoder

- Function: testing phase
  - Reconstruction



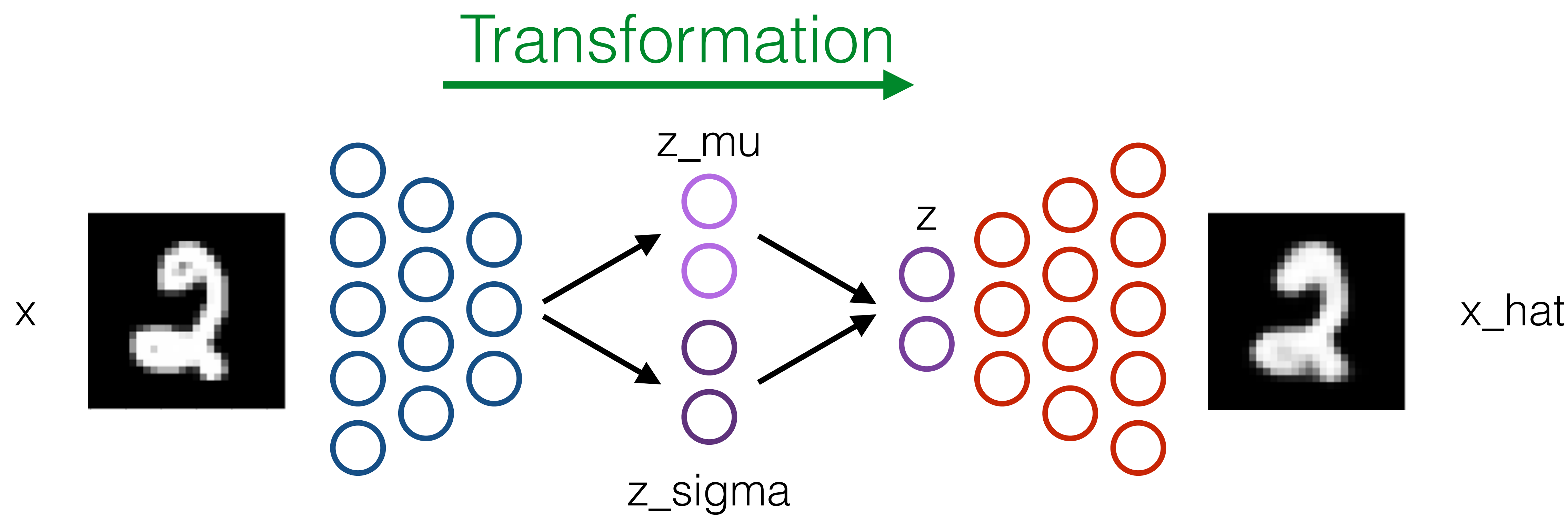
Variational Autoencoder

- Function: testing phase
  - Generation



Variational Autoencoder

- Function: testing phase
  - Transformation



Coding Session

Variational Autoencoder

Review

Import

```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets('MNIST_data', one_hot=True)
num_sample = mnist.train.num_examples
input_dim = mnist.train.images[0].shape[0]
w = h = int(np.sqrt(input_dim))
```

Data

```
import time
import numpy as np
import tensorflow as tf
from tensorflow.contrib.slim import fully_connected as fc
import matplotlib.pyplot as plt
%matplotlib inline
```

Review

Model

```
class VariantionalAutoencoder(object):

    def __init__(self, learning_rate=1e-4, batch_size=64, n_z=16):
        self.learning_rate = learning_rate
        self.batch_size = batch_size
        self.n_z = n_z

        tf.reset_default_graph()
        self.build()

        self.sess = tf.InteractiveSession()
        self.sess.run(tf.global_variables_initializer())
```

Review

Build the network: the encoder and decoder

```
# Build the netowrk and the loss functions
def build(self):
    self.x = tf.placeholder(name='x', dtype=tf.float32, shape=[None, input_dim])

    # Encode
    # x -> z_mean, z_sigma -> z
    f1 = fc(self.x, 256, scope='enc_fc1', activation_fn=tf.nn.elu)
    f2 = fc(f1, 128, scope='enc_fc2', activation_fn=tf.nn.elu)
    f3 = fc(f2, 64, scope='enc_fc3', activation_fn=tf.nn.elu)
    self.z_mu = fc(f3, self.n_z, scope='enc_fc4_mu', activation_fn=None)
    self.z_log_sigma_sq = fc(f3, self.n_z, scope='enc_fc4_sigma', activation_fn=None)
    eps = tf.random_normal(
        shape=tf.shape(self.z_log_sigma_sq),
        mean=0, stddev=1, dtype=tf.float32)
    self.z = self.z_mu + tf.sqrt(tf.exp(self.z_log_sigma_sq)) * eps

    # Decode
    # z -> x_hat
    g1 = fc(self.z, 64, scope='dec_fc1', activation_fn=tf.nn.elu)
    g2 = fc(g1, 128, scope='dec_fc2', activation_fn=tf.nn.elu)
    g3 = fc(g2, 256, scope='dec_fc3', activation_fn=tf.nn.elu)
    self.x_hat = fc(g3, input_dim, scope='dec_fc4', activation_fn=tf.sigmoid)
```

Review

Build the network: the loss

```
# Loss
# Reconstruction loss
# Minimize the cross-entropy loss
# H(x, x_hat) = -\Sigma x*log(x_hat) + (1-x)*log(1-x_hat)
epsilon = 1e-10
recon_loss = -tf.reduce_sum(
    self.x * tf.log(epsilon+self.x_hat) + (1-self.x) * tf.log(epsilon+1-self.x_hat),
    axis=1
)
self.recon_loss = tf.reduce_mean(recon_loss)

# Latent loss
# KL divergence: measure the difference between two distributions
# Here we measure the divergence between the latent distribution and N(0, 1)
latent_loss = -0.5 * tf.reduce_sum(
    1 + self.z_log_sigma_sq - tf.square(self.z_mu) - tf.exp(self.z_log_sigma_sq), axis=1)
self.latent_loss = tf.reduce_mean(latent_loss)

self.total_loss = self.recon_loss + self.latent_loss
self.train_op = tf.train.AdamOptimizer(
    learning_rate=self.learning_rate).minimize(self.total_loss)

self.losses = {
    'recon_loss': self.recon_loss,
    'latent_loss': self.latent_loss,
    'total_loss': self.total_loss,
}
```

Review

Training and testing functions

```python
# Execute the forward and the backward pass
def run_single_step(self, x):
    _, losses = self.sess.run(
        [self.train_op, self.losses],
        feed_dict={self.x: x}
    )
    return losses

# x -> x_hat
def reconstructor(self, x):
    x_hat = self.sess.run(self.x_hat, feed_dict={self.x: x})
    return x_hat

# z -> x
def generator(self, z):
    x_hat = self.sess.run(self.x_hat, feed_dict={self.z: z})
    return x_hat

# x -> z
def transformer(self, x):
    z = self.sess.run(self.z, feed_dict={self.x: x})
    return z
```

Review

Train a model

```
def trainer(model_object, learning_rate=1e-4, batch_size=64, num_epoch=100, n_z=16):
    model = model_object(
        learning_rate=learning_rate, batch_size=batch_size, n_z=n_z)

    for epoch in range(num_epoch):
        start_time = time.time()
        for iter in range(num_sample // batch_size):
            # Get a batch
            batch = mnist.train.next_batch(batch_size)
            # Execute the forward and backward pass and report computed losses
            losses = model.run_single_step(batch[0])
        end_time = time.time()

        if epoch % 5 == 0:
            log_str = '[Epoch {}] '.format(epoch)
            for k, v in losses.items():
                log_str += '{}: {:.3f}  '.format(k, v)
            log_str += '({:.3f} sec/epoch)'.format(end_time - start_time)
            print(log_str)

    print('Done!')
    return model
```

```
# Train a model
model = trainer(VariantionalAutoencoder)
```

Review

Train a model

```
[Epoch 0] recon_loss: 182.894  latent_loss: 9.692  total_loss: 192.586  (2.595 sec/epoch)
[Epoch 5] recon_loss: 113.784  latent_loss: 14.704  total_loss: 128.488  (2.470 sec/epoch)
[Epoch 10] recon_loss: 108.452  latent_loss: 17.072  total_loss: 125.523  (2.724 sec/epoch)
[Epoch 15] recon_loss: 101.748  latent_loss: 17.579  total_loss: 119.328  (2.027 sec/epoch)
[Epoch 20] recon_loss: 91.644  latent_loss: 18.083  total_loss: 109.727  (2.467 sec/epoch)
[Epoch 25] recon_loss: 94.685  latent_loss: 18.548  total_loss: 113.234  (2.504 sec/epoch)
[Epoch 30] recon_loss: 87.946  latent_loss: 18.468  total_loss: 106.414  (2.454 sec/epoch)
[Epoch 35] recon_loss: 90.971  latent_loss: 18.891  total_loss: 109.862  (2.588 sec/epoch)
[Epoch 40] recon_loss: 84.645  latent_loss: 18.783  total_loss: 103.428  (2.144 sec/epoch)
[Epoch 45] recon_loss: 85.681  latent_loss: 19.136  total_loss: 104.818  (2.589 sec/epoch)
[Epoch 50] recon_loss: 84.453  latent_loss: 19.142  total_loss: 103.595  (2.524 sec/epoch)
[Epoch 55] recon_loss: 86.023  latent_loss: 19.744  total_loss: 105.767  (2.755 sec/epoch)
[Epoch 60] recon_loss: 84.479  latent_loss: 19.800  total_loss: 104.280  (2.405 sec/epoch)
[Epoch 65] recon_loss: 86.094  latent_loss: 19.502  total_loss: 105.596  (2.068 sec/epoch)
[Epoch 70] recon_loss: 84.493  latent_loss: 19.866  total_loss: 104.359  (2.738 sec/epoch)
[Epoch 75] recon_loss: 83.497  latent_loss: 19.736  total_loss: 103.233  (2.659 sec/epoch)
[Epoch 80] recon_loss: 84.940  latent_loss: 20.300  total_loss: 105.240  (2.567 sec/epoch)
[Epoch 85] recon_loss: 83.345  latent_loss: 19.918  total_loss: 103.263  (2.544 sec/epoch)
[Epoch 90] recon_loss: 83.698  latent_loss: 20.042  total_loss: 103.740  (2.691 sec/epoch)
[Epoch 95] recon_loss: 84.776  latent_loss: 19.934  total_loss: 104.710  (2.520 sec/epoch)
Done!
```

Review

Test the model: **reconstruction**

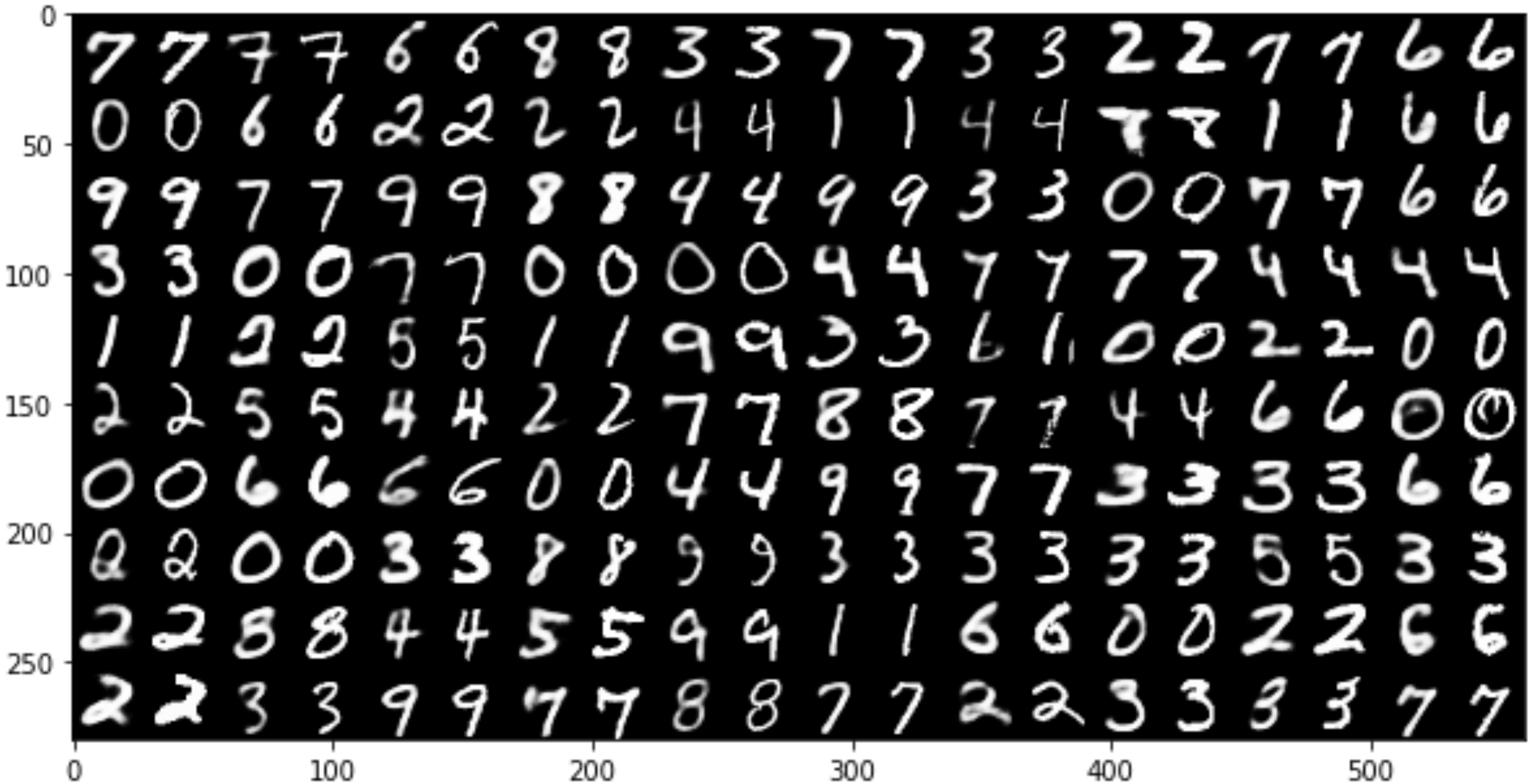
```
def test_reconstruction(model, mnist, h=28, w=28, batch_size=100):
    # Test the trained model: reconstruction
    batch = mnist.test.next_batch(batch_size)
    x_reconstructed = model.reconstructor(batch[0])

    n = np.sqrt(batch_size).astype(np.int32)
    I_reconstructed = np.empty((h*n, 2*w*n))
    for i in range(n):
        for j in range(n):
            x = np.concatenate(
                (x_reconstructed[i*n+j, :].reshape(h, w),
                 batch[0][i*n+j, :].reshape(h, w)),
                axis=1
            )
            I_reconstructed[i*h:(i+1)*h, j*2*w:(j+1)*2*w] = x

    plt.figure(figsize=(10, 20))
    plt.imshow(I_reconstructed, cmap='gray')
```

Review

Test the model: **reconstruction**



Review

Test the model: **generation**

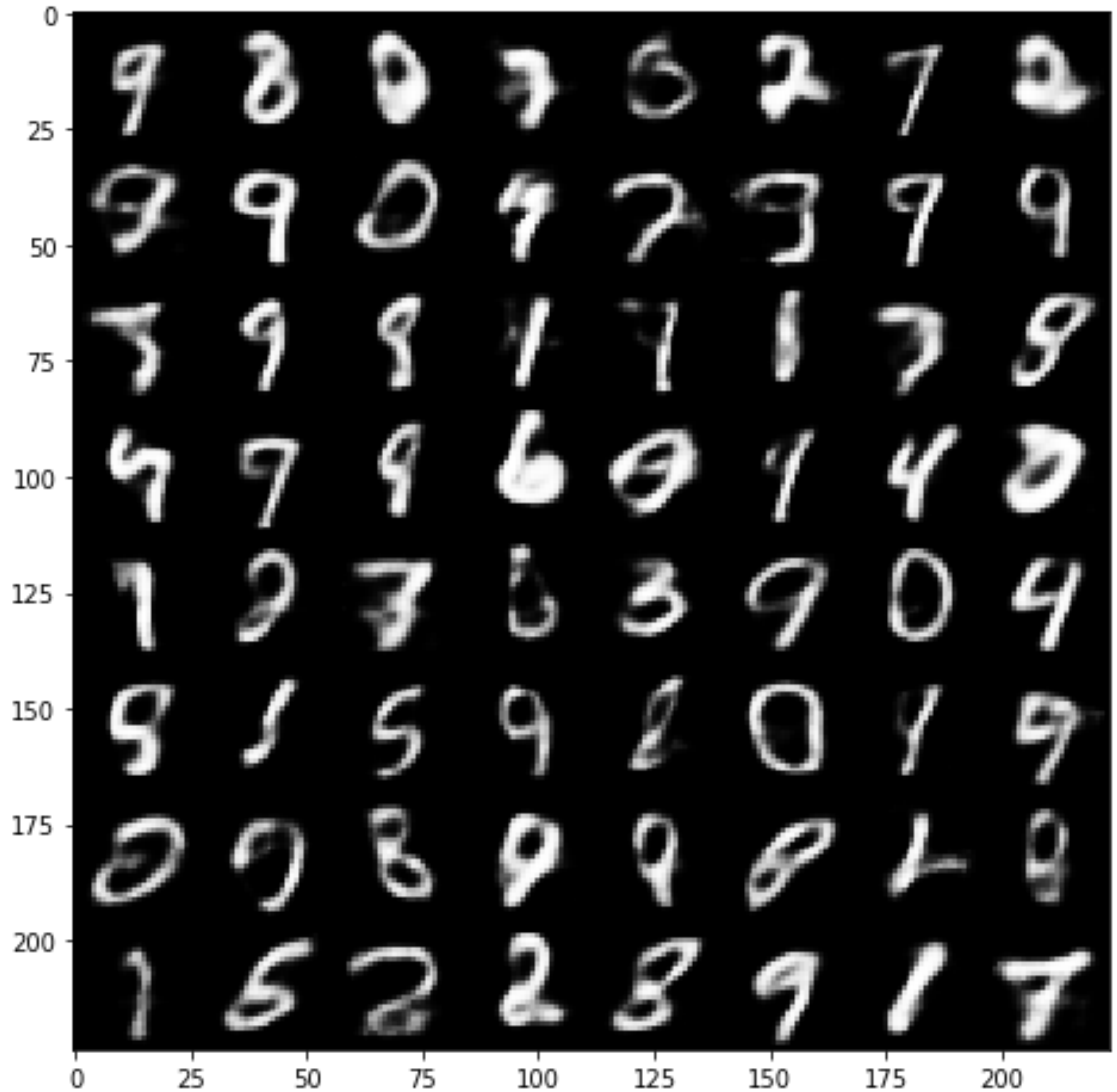
```
# Test the trained model: generation
# Sample noise vectors from N(0, 1)
z = np.random.normal(size=[model.batch_size, model.n_z])
x_generated = model.generator(z)

n = np.sqrt(model.batch_size).astype(np.int32)
I_generated = np.empty((h*n, w*n))
for i in range(n):
    for j in range(n):
        I_generated[i*h:(i+1)*h, j*w:(j+1)*w] = x_generated[i*n+j, :].reshape(h, w)

plt.figure(figsize=(8, 8))
plt.imshow(I_generated, cmap='gray')
```

Review

Test the model: **generation**



Review

Train a 2d model

```
# Train a model with 2d latent space
model_2d = trainer(VariantionalAutoencoder, n_z=2)
```

```
[Epoch 0] recon_loss: 199.729  latent_loss: 4.342  total_loss: 204.071  (2.159 sec/epoch)
[Epoch 5] recon_loss: 167.280  latent_loss: 4.728  total_loss: 172.008  (2.601 sec/epoch)
[Epoch 10] recon_loss: 145.253  latent_loss: 5.516  total_loss: 150.769  (2.589 sec/epoch)
[Epoch 15] recon_loss: 144.056  latent_loss: 5.380  total_loss: 149.436  (2.580 sec/epoch)
[Epoch 20] recon_loss: 148.198  latent_loss: 5.860  total_loss: 154.058  (2.712 sec/epoch)
[Epoch 25] recon_loss: 152.884  latent_loss: 5.783  total_loss: 158.667  (2.075 sec/epoch)
[Epoch 30] recon_loss: 147.188  latent_loss: 5.782  total_loss: 152.970  (2.813 sec/epoch)
[Epoch 35] recon_loss: 144.544  latent_loss: 5.971  total_loss: 150.515  (2.479 sec/epoch)
[Epoch 40] recon_loss: 152.908  latent_loss: 5.965  total_loss: 158.873  (2.542 sec/epoch)
[Epoch 45] recon_loss: 132.893  latent_loss: 6.022  total_loss: 138.916  (2.746 sec/epoch)
[Epoch 50] recon_loss: 136.282  latent_loss: 6.303  total_loss: 142.585  (2.093 sec/epoch)
[Epoch 55] recon_loss: 148.406  latent_loss: 6.235  total_loss: 154.640  (2.798 sec/epoch)
[Epoch 60] recon_loss: 131.068  latent_loss: 6.062  total_loss: 137.130  (2.410 sec/epoch)
[Epoch 65] recon_loss: 135.531  latent_loss: 6.250  total_loss: 141.781  (2.393 sec/epoch)
[Epoch 70] recon_loss: 129.911  latent_loss: 6.276  total_loss: 136.187  (2.722 sec/epoch)
[Epoch 75] recon_loss: 141.734  latent_loss: 6.252  total_loss: 147.986  (2.587 sec/epoch)
[Epoch 80] recon_loss: 149.359  latent_loss: 6.349  total_loss: 155.708  (2.023 sec/epoch)
[Epoch 85] recon_loss: 138.324  latent_loss: 6.197  total_loss: 144.521  (2.484 sec/epoch)
[Epoch 90] recon_loss: 130.314  latent_loss: 6.396  total_loss: 136.711  (2.500 sec/epoch)
[Epoch 95] recon_loss: 133.127  latent_loss: 6.579  total_loss: 139.706  (2.602 sec/epoch)
Done!
```

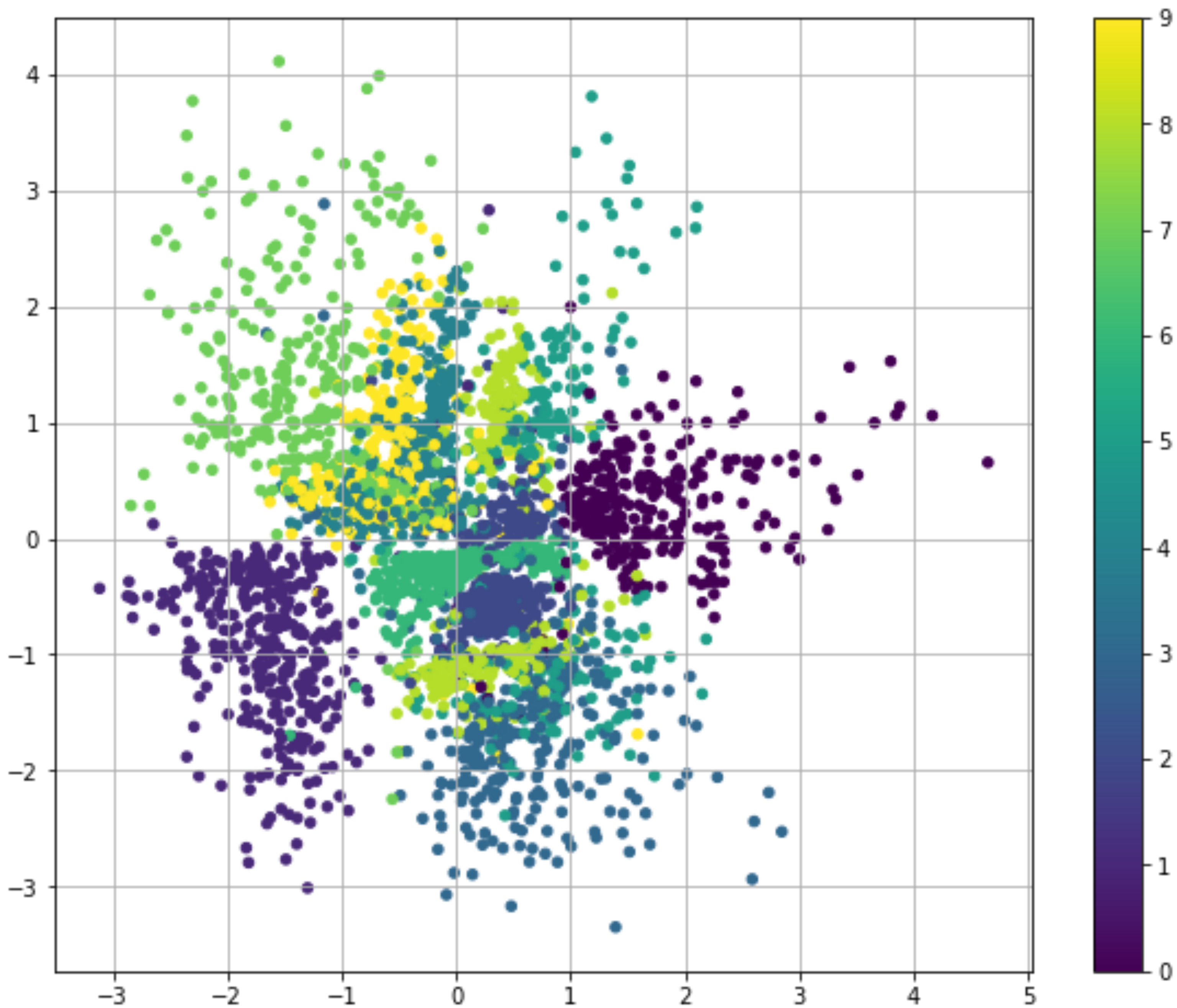
Review

Test the 2d model: **transformation**

```
def test_transformation(model_2d, mnist, batch_size=3000):
    # Test the trained model: transformation
    assert model.n_z == 2
    batch = mnist.test.next_batch(batch_size)
    z = model_2d.transformer(batch[0])
    plt.figure(figsize=(10, 8))
    plt.scatter(z[:, 0], z[:, 1], c=np.argmax(batch[1], 1), s=20)
    plt.colorbar()
    plt.grid()
```

Review

Test the 2d model: **transformation**



Review

Test the 2d model: **latent space**

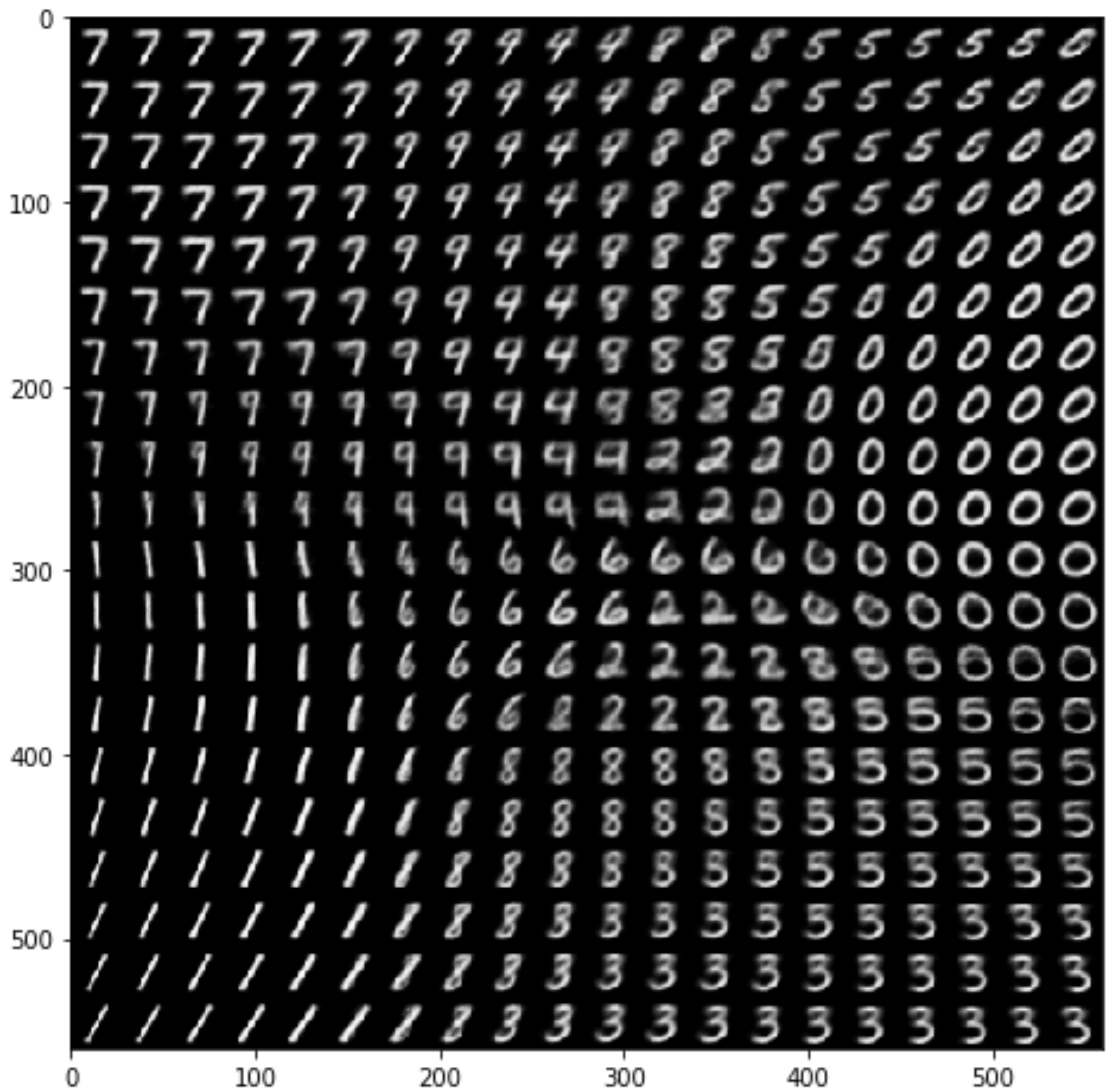
```
# Test the trained model: transformation
n = 20
x = np.linspace(-2, 2, n)
y = np.linspace(-2, 2, n)

I_latent = np.empty((h*n, w*n))
for i, yi in enumerate(x):
    for j, xi in enumerate(y):
        z = np.array([[xi, yi]]*model_2d.batch_size)
        x_hat = model_2d.generator(z)
        I_latent[(n-i-1)*h:(n-i)*h, j*w:(j+1)*w] = x_hat[0].reshape(h, w)

plt.figure(figsize=(8, 8))
plt.imshow(I_latent, cmap="gray")
```

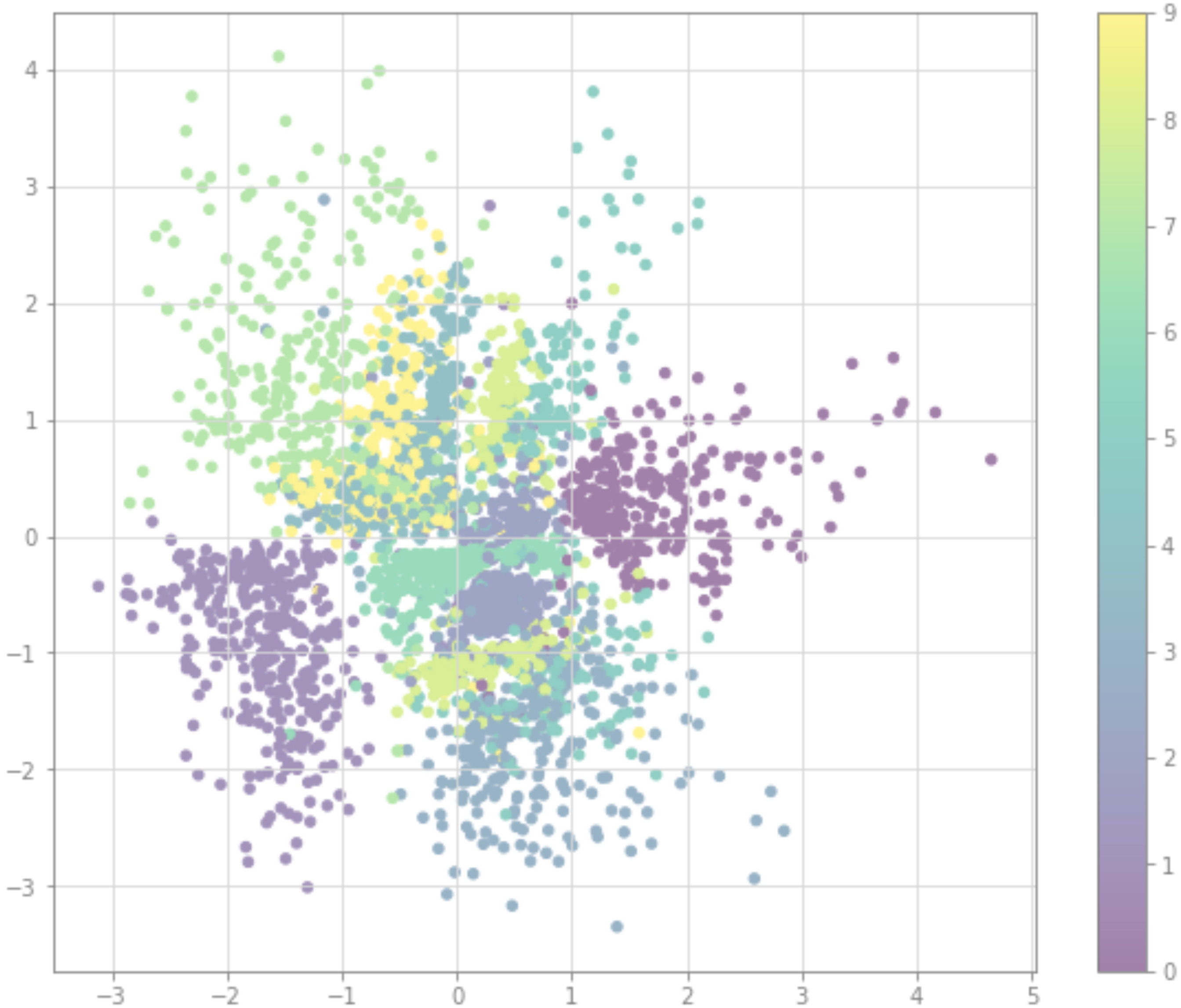
Review

Test the 2d model: **latent space**



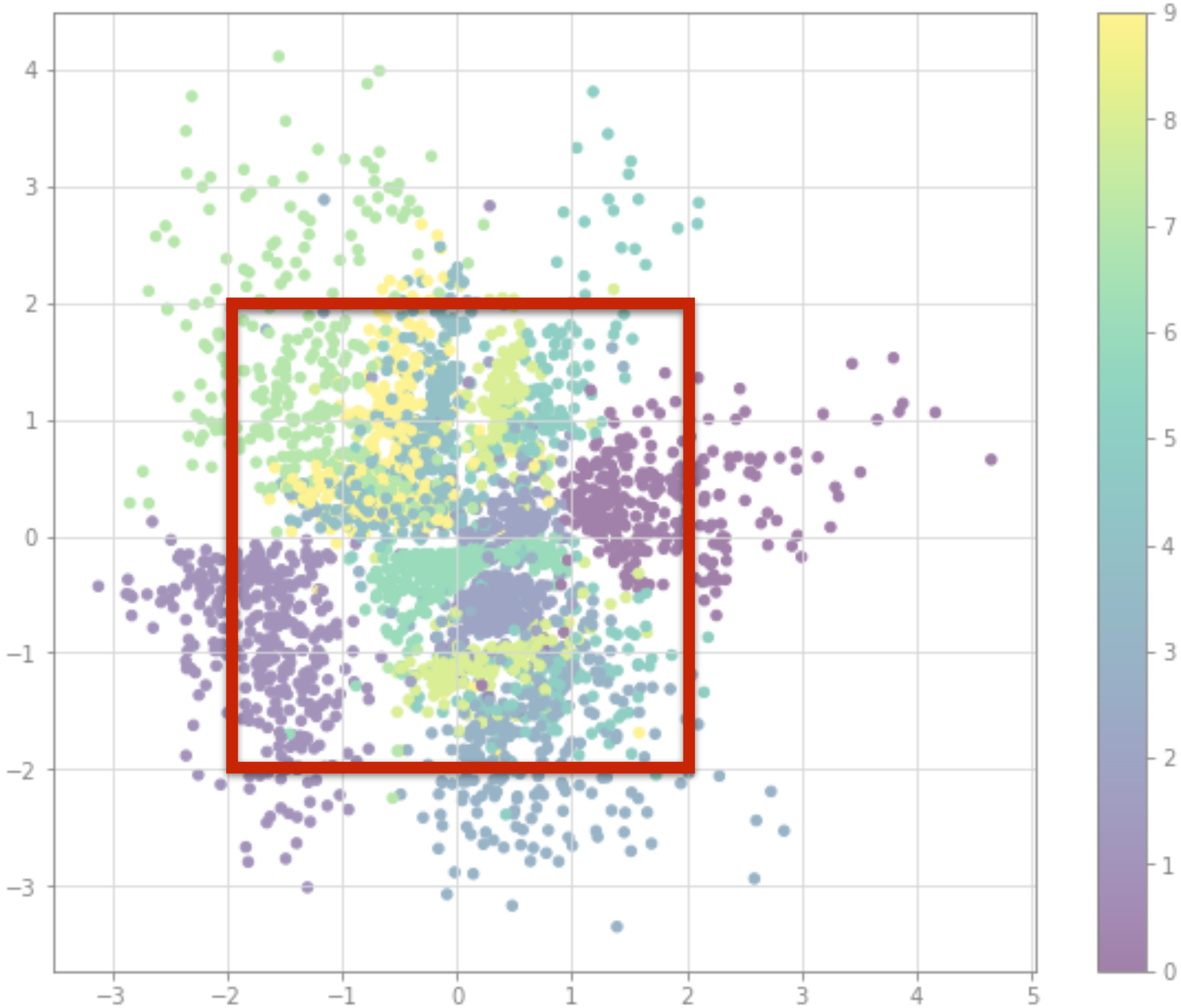
Review

Test the 2d model: **latent space**



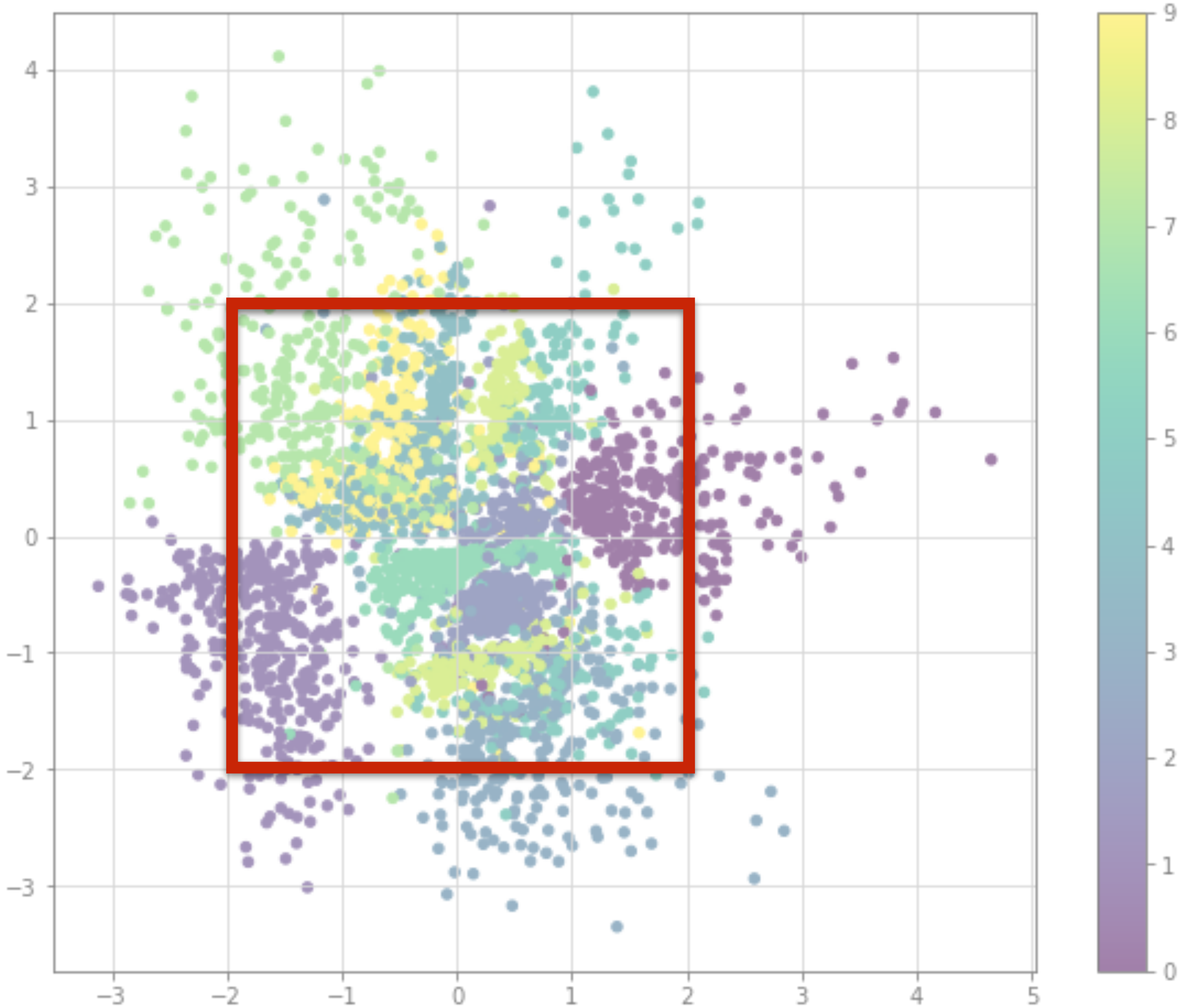
Review

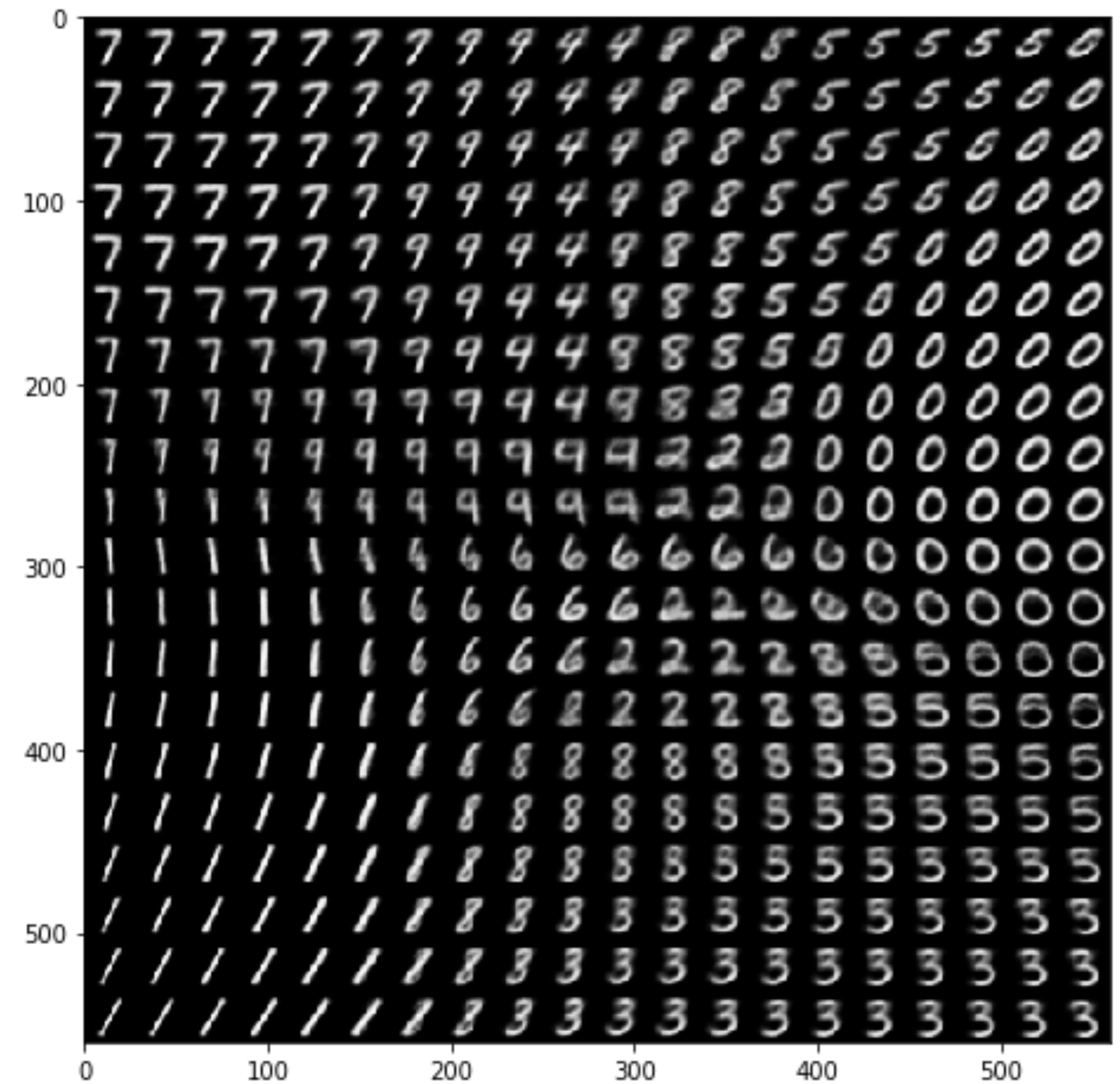
Test the 2d model: **latent space**



Review

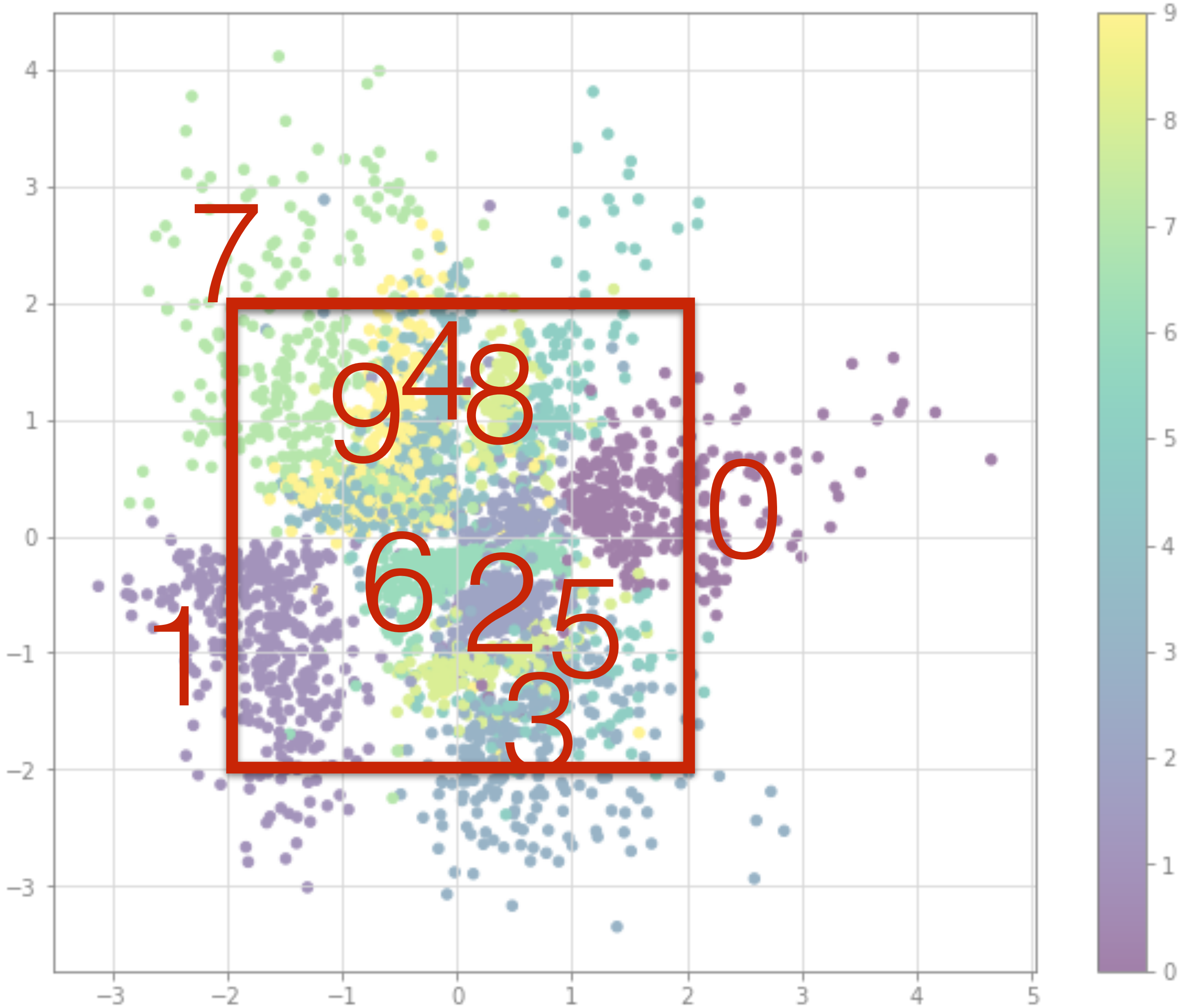
Test the 2d model: **latent space**

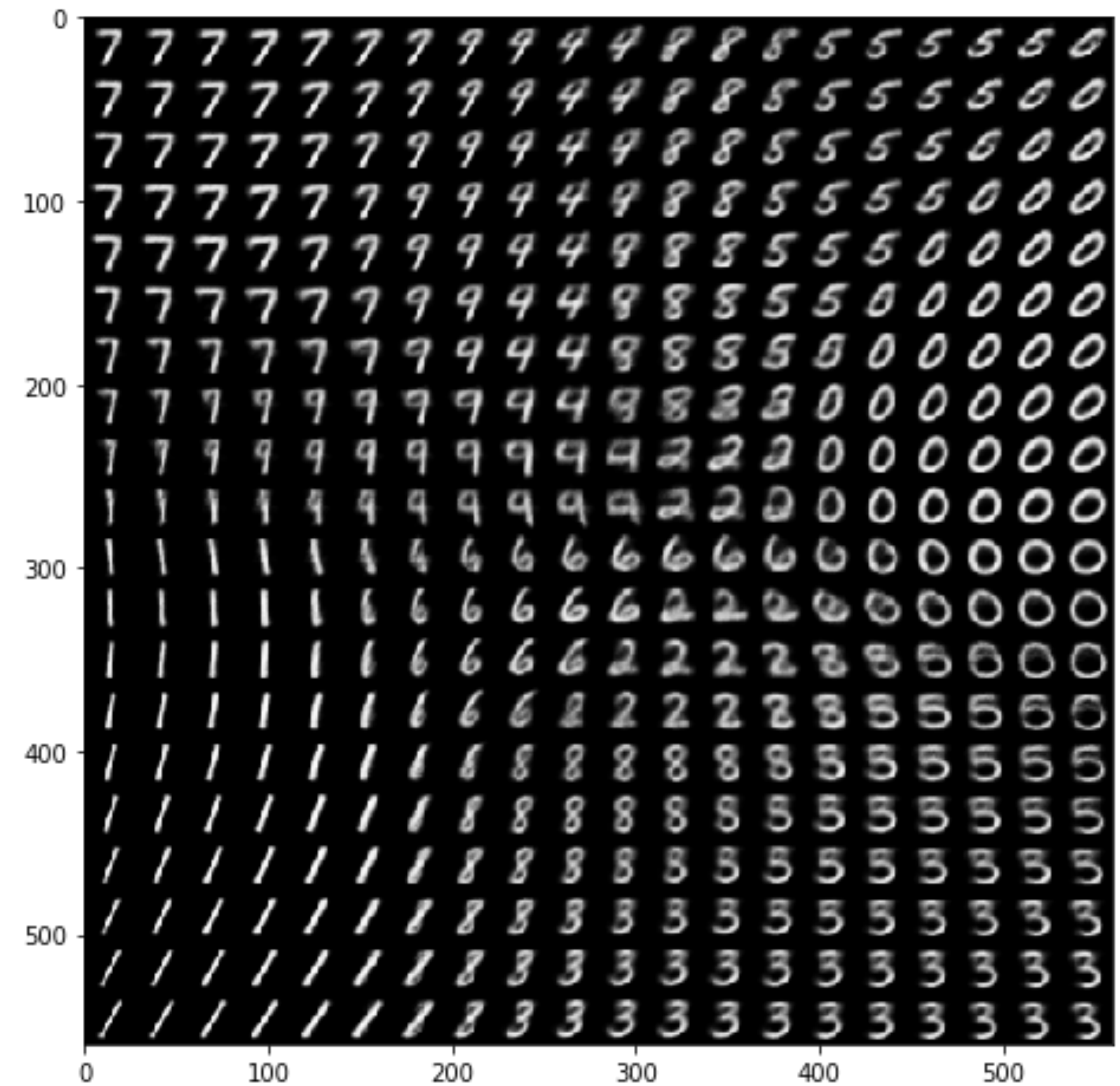




Review

Test the 2d model: **latent space**





Summary

TensorFlow

- Build the graph
- Run the graph

Variational Autoencoder

- Generative model
- [A blog post about VAEs]
- Demo code (https://github.com/shaohua0116/VAE-Tensorflow)

Questions?

Today’s agenda

- **Part 1: Deep Learning Framework**
  - TensorFlow
  - PyTorch
- **Part 2: Cloud Service**
  - Google Cloud
  - Amazon Web Services