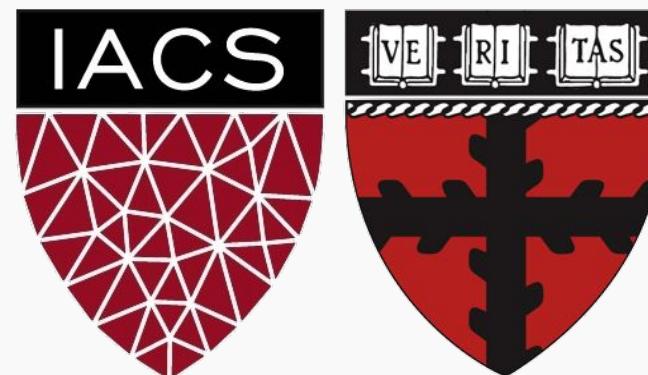
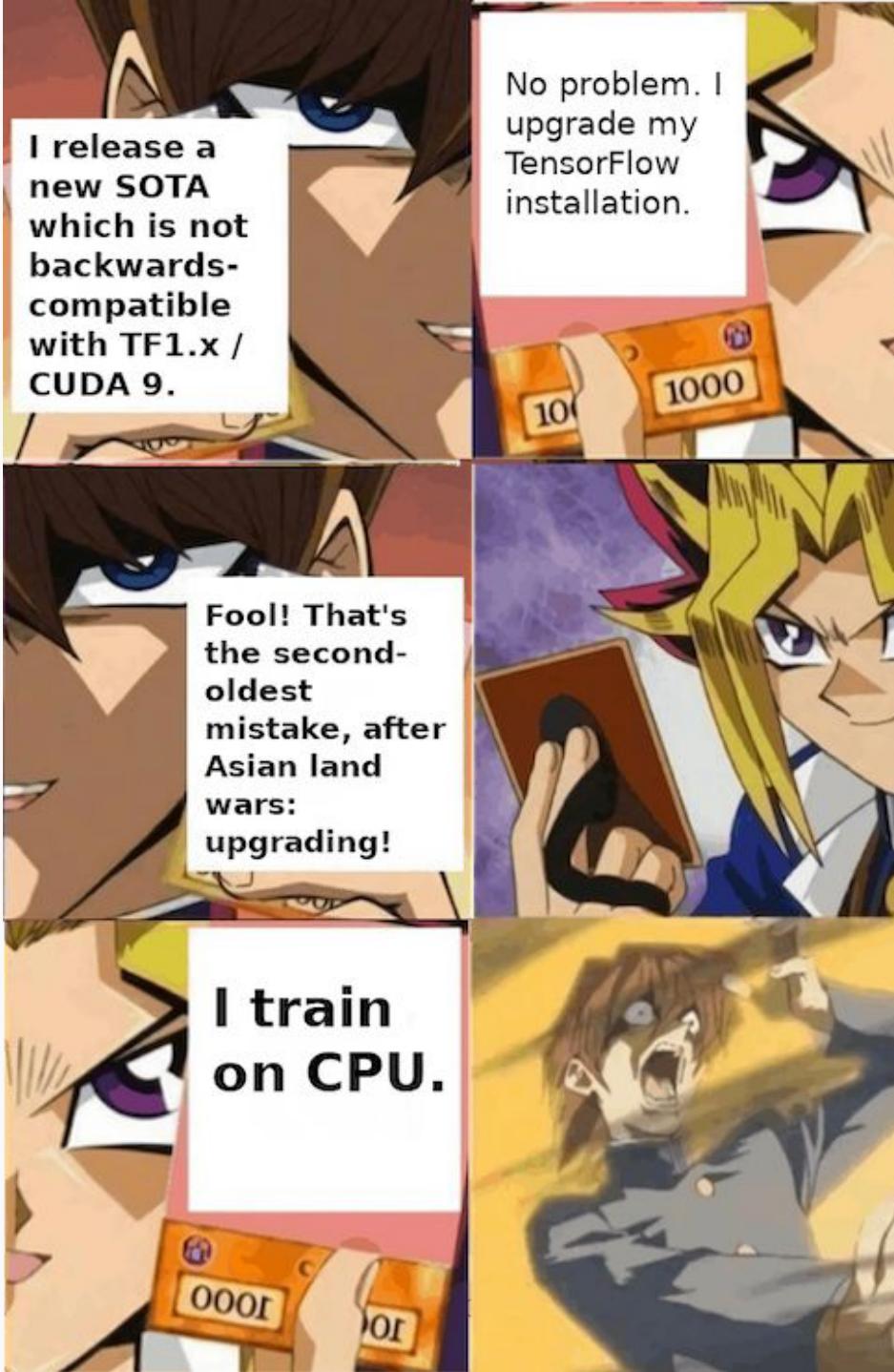


# Lecture 12: Autoencoders

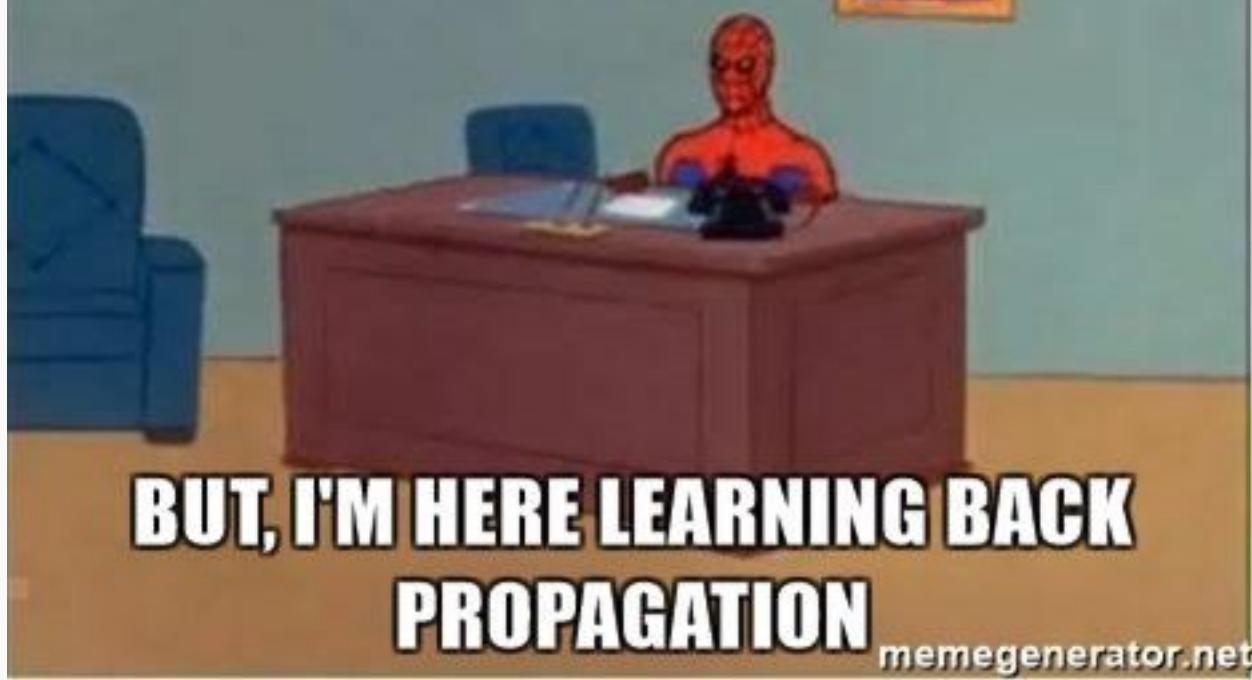
CS109B Data Science 2

Pavlos Protopapas, Mark Glickman, Chris Tanner





**IT IS FRIDAY, ALL MY FRIENDS ARE  
AT PARTY**



**BUT, I'M HERE LEARNING BACK  
PROPAGATION**

memegenerator.net

## Announcements:

- Homework 3 deadline ...
- Project group have been formed and will be announced ...



### ■ "Pure" Reinforcement Learning (cherry)

- ▶ The machine predicts a scalar reward given once in a while.
- ▶ **A few bits for some samples**



### ■ Supervised Learning (icing)

- ▶ The machine predicts a category or a few numbers for each input
- ▶ Predicting human-supplied data
- ▶ **10→10,000 bits per sample**

### ■ Unsupervised/Predictive Learning (cake)

- ▶ The machine predicts any part of its input for any observed part.
- ▶ Predicts future frames in videos
- ▶ **Millions of bits per sample**

■ (Yes, I know, this picture is slightly offensive to RL folks. But I'll make it up)

Original LeCun cake analogy slide presented at NIPS 2016, the highlighted area has now been updated.

# How Much Information is the Machine Given during Learning?

- ▶ “Pure” Reinforcement Learning (**cherry**)
  - ▶ The machine predicts a scalar reward given once in a while.

**A few bits for some samples**



- ▶ Supervised Learning (**icing**)
  - ▶ The machine predicts a category or a few numbers for each input
  - ▶ Predicting human-supplied data
  - ▶ **10→10,000 bits per sample**

- ▶ Self-Supervised Learning (**cake génoise**)
  - ▶ The machine predicts any part of its input for any observed part.
  - ▶ Predicts future frames in videos
  - ▶ **Millions of bits per sample**

© 2019 IEEE International Solid-State Circuits Conference

1.1: Deep Learning Hardware: Past, Present, & Future

59

LeCun updated his cake recipe last week at the 2019 International Solid-State Circuits Conference (ISSCC) in San Francisco, replacing “unsupervised learning” with “self-supervised learning,” a variant of unsupervised learning where the data provides the supervision.

# Outline

---

- What are autoencoders?
- Brief history of encoding/decoding.
- Inside autoencoders.
- Regularization of autoencoders.
- Applications
  - Denoising
  - Blending



# Outline

---

- **What are autoencoders?**
- Brief history of encoding/decoding.
- Inside autoencoders.
- Regularization of autoencoders.
- Applications
  - Denoising
  - Blending



# Quick Review. Neural Networks as function approximation.

Given an input  $x$  and an output  $y$  there exists a mapping from input space to output space as follows:

$$\begin{aligned}x &\rightarrow y \\y &= f(x) + \epsilon\end{aligned}$$

Our goal is to find an estimate of  $f(x)$  which we will call  $\hat{f}(x)$ .

Statistical learning or modeling is the process of finding  $\hat{f}(x)$ .

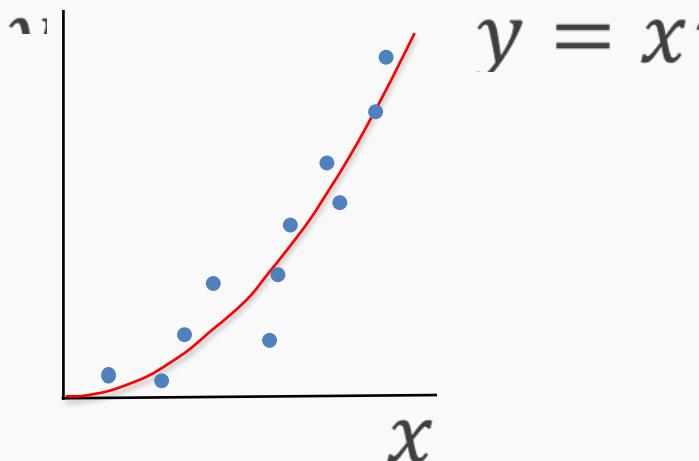
Neural networks are one of many possible methods we can use to obtain the estimate  $\hat{f}(x)$ .

# Representational Learning

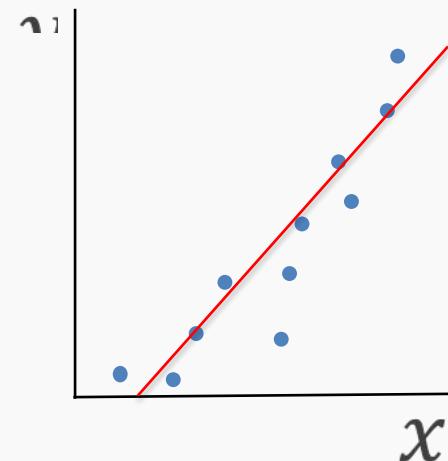
Representation Matters



Fit quadratic function

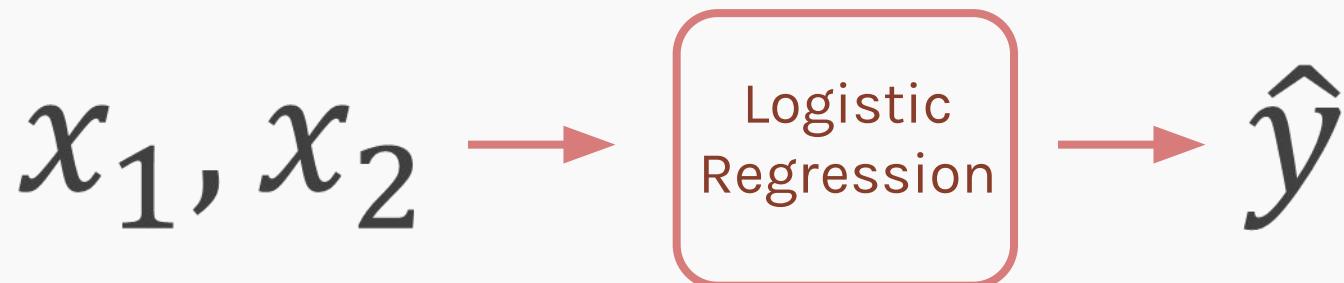


Do your best !

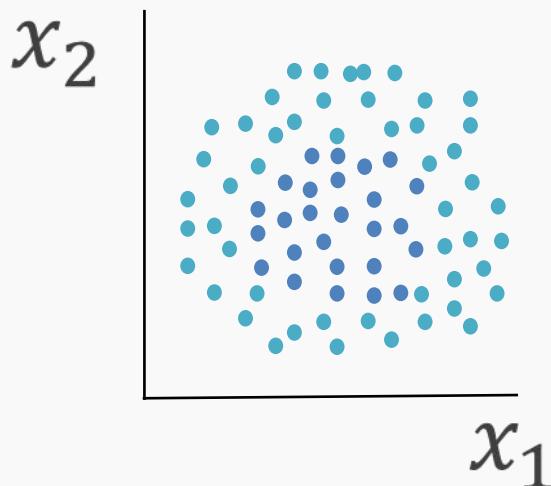


# Representational Learning

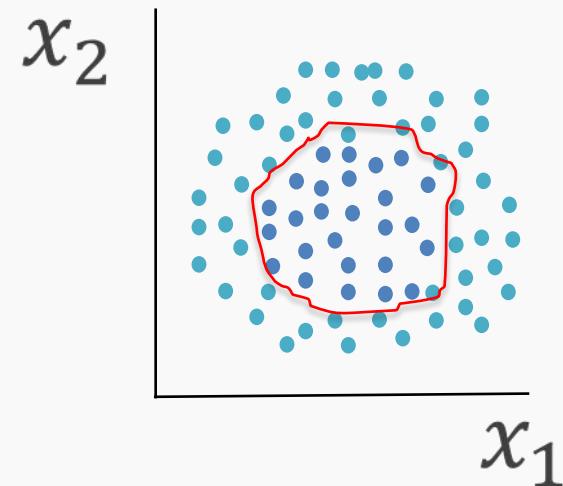
Representation Matters



Fit polynomial function

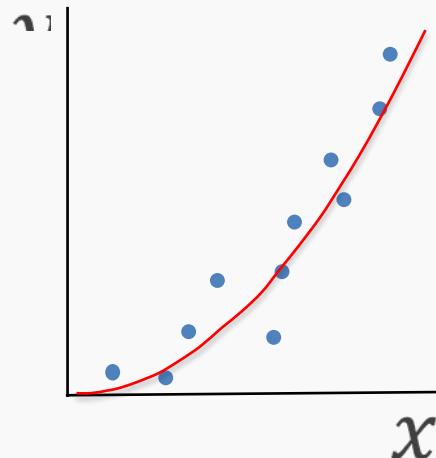
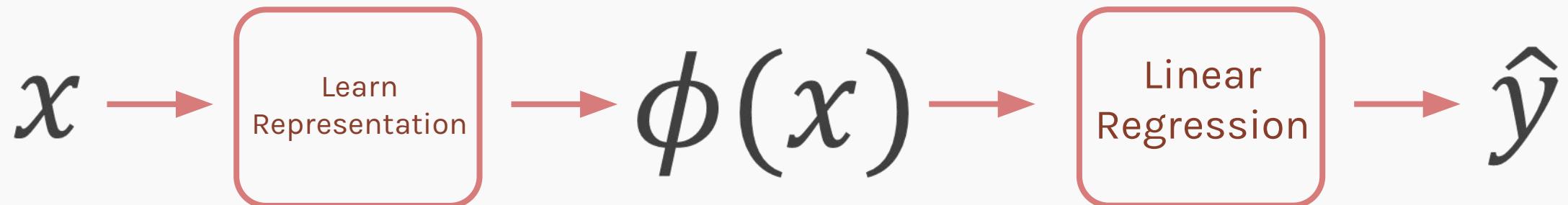


Do your best !

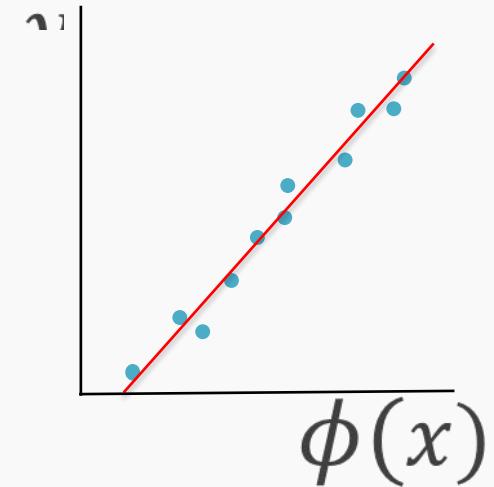


# Representational Learning

Representation Matters

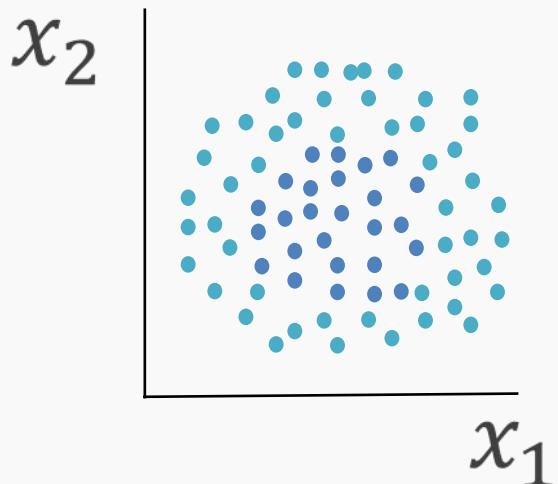
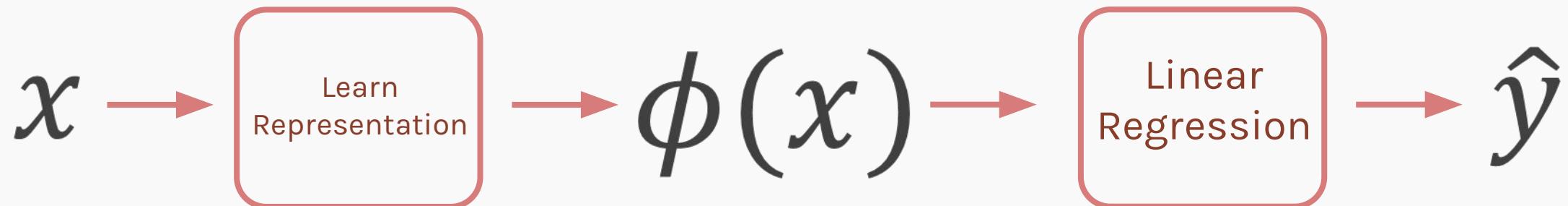


$$\phi(x) = x^2$$



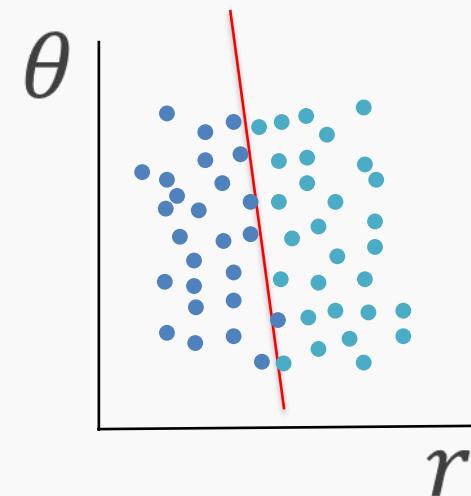
# Representational Learning

Representation Matters



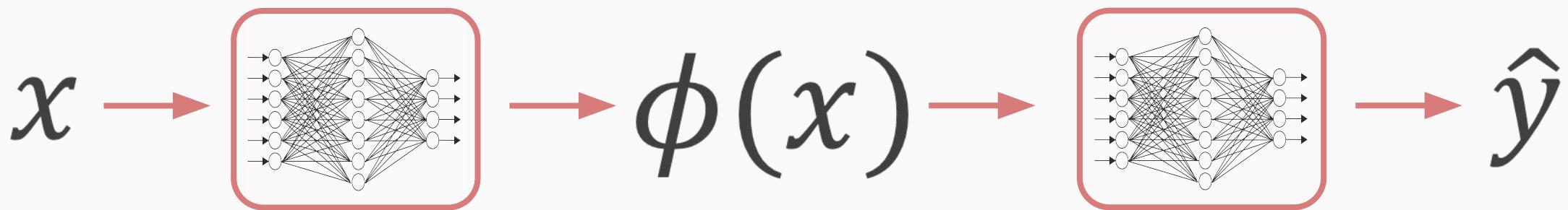
$\phi(x)$

$$r^2 = x_1^2 + x_2^2$$
$$\theta = \tan^{-1} \frac{x_2}{x_1}$$



# Representational Learning

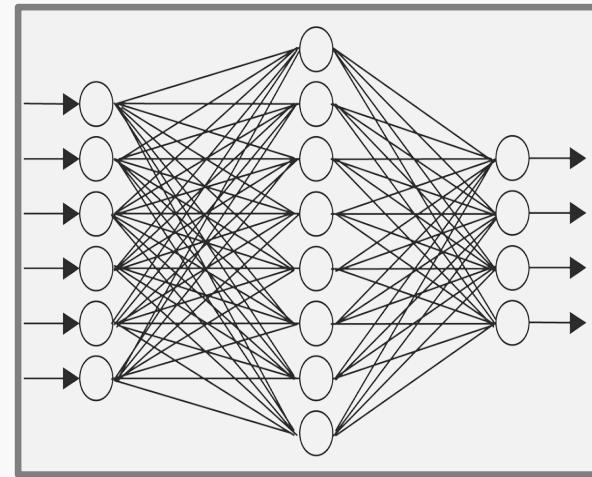
Representation Matters



# Representational Learning: **Supervised Learning**

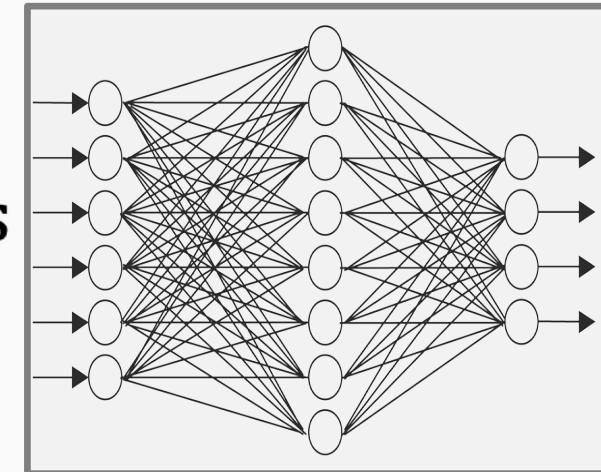
We train the two networks by minimizing the loss function (**cross entropy loss**)

$X$



Feature Discovery Network

$Y$

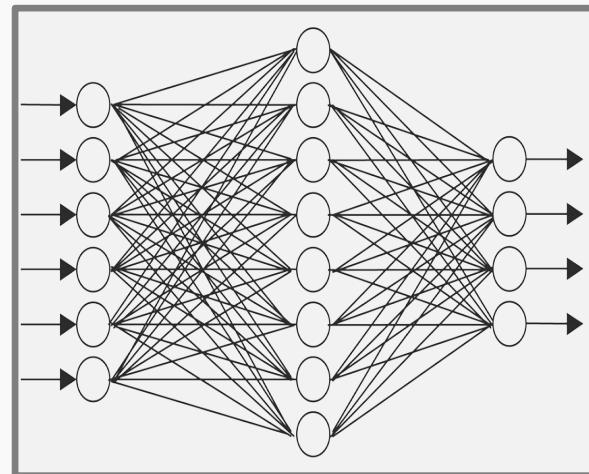


Classification Network

# Representational Learning: **Self-supervised Learning**

We train the two networks by minimizing the cross entropy loss.

$X$



Feature Discovery Network

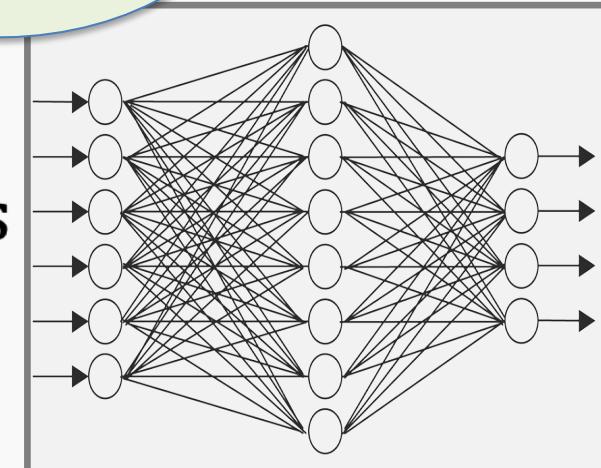
Specialized  
features for  
Task

**Features**  
 $\phi(x)$

Cross entropy loss.

No labels

$Y$



Classification Network

{Cat,Dog}

# Representational Learning: **Self-supervised Learning**

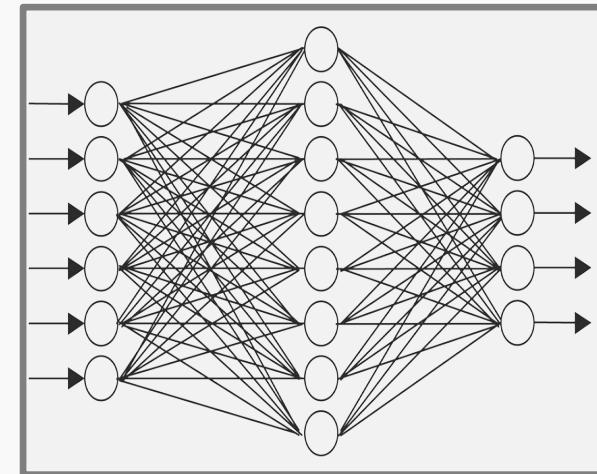
We train the two networks by minimizing the **reconstruction** loss function:  $\mathcal{L} = \sum(x_i - \hat{x}_i)^2$

$X$



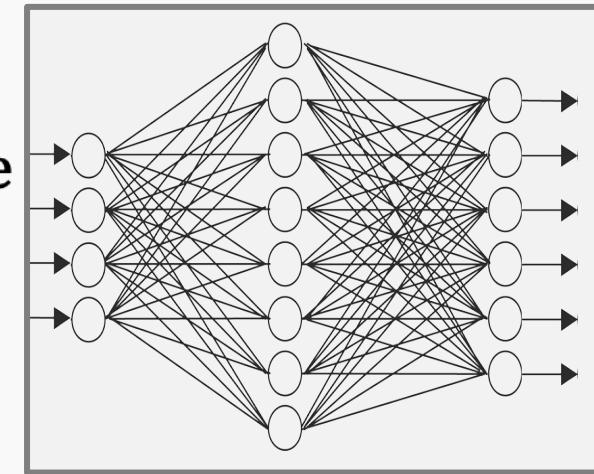
## AUTOENCODER

$\hat{X}$



Latent Space  
Features  
 $\phi(x)$

Feature Discovery Network



Decoder Network

This is an **autoencoder**. It gets that name because it automatically finds the best way to encode the input so that the decoded version is as close as possible to the input.

# Brief history of encoding/decoding

---

Is this a new idea?

- **MP3** can compress music files by a factor of 10 enabling digital storage and transmission large volumes of audio.
- **JPG** compresses images by a factor of 10-20 and enables storage and transmission of image data.
- These technologies led the way to the image-rich web and abundance of music that we enjoy today.

# Brief history of encoding/decoding (cont)

---

We say that both MP3 and JPG take an input (a music or image file), and **encode** it into a **compressed** form.

Then we **decode** or **decompress** the intermediate version to some lower quality original version.

# Lossless and Lossy Encoding

---

The greater the difference between the original version and the version post-decompression the greater the **loss**.

**Example:** Imagine you are in Boston and you want to write a birthday text to a special friend while walking home.

**HANNAH HAPPY BIRTHDAY I LOVE YOU DAN**

In the freezing Boston winter (-20C) you do not want to have your hands out in the open, so you shorten the message as much as possible using text-speak:

**H HBD ILY D**

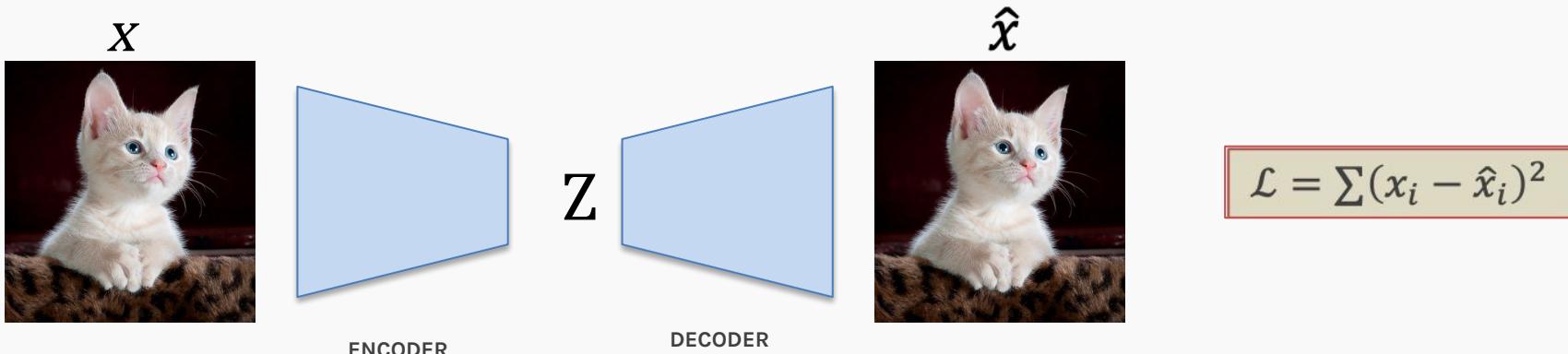
Your 36 characters message is compressed to 11 characters

# Lossless and Lossy Encoding (cont)

**Question:** Is this an example lossy or lossless compression?

**HPD** is unambiguous given that it is her birthday today, but **D** could mean **Dan or David or Donny** ... You can imagine the potential drama.

A way to test if a transformation is **lossy** or **lossless** is to consider if it can be inverted, or run backwards, to provide us with the original data. In autoencoders this is the loss function.



# What are autoencoders?

---

- A particular kind of learning architecture.
- A mechanism of compressing inputs into a form that can later be decompressed similar to the way MP3 compresses audio and JPG compresses images.
- Autoencoders are more general than either MP3 or JPG.
- They are usually used to ...
  - reduce data dimensionality or find a more general representation for many tasks
  - blend inputs from one input to another
  - denoising, infilling
  - ...

# MP3 and JPG Image Compression



original



MP3



JPG

Original image 256x256=262,000, MP3=37,000, and JPG=26,000

# MP3 and JPG Image Compression(cont)



original



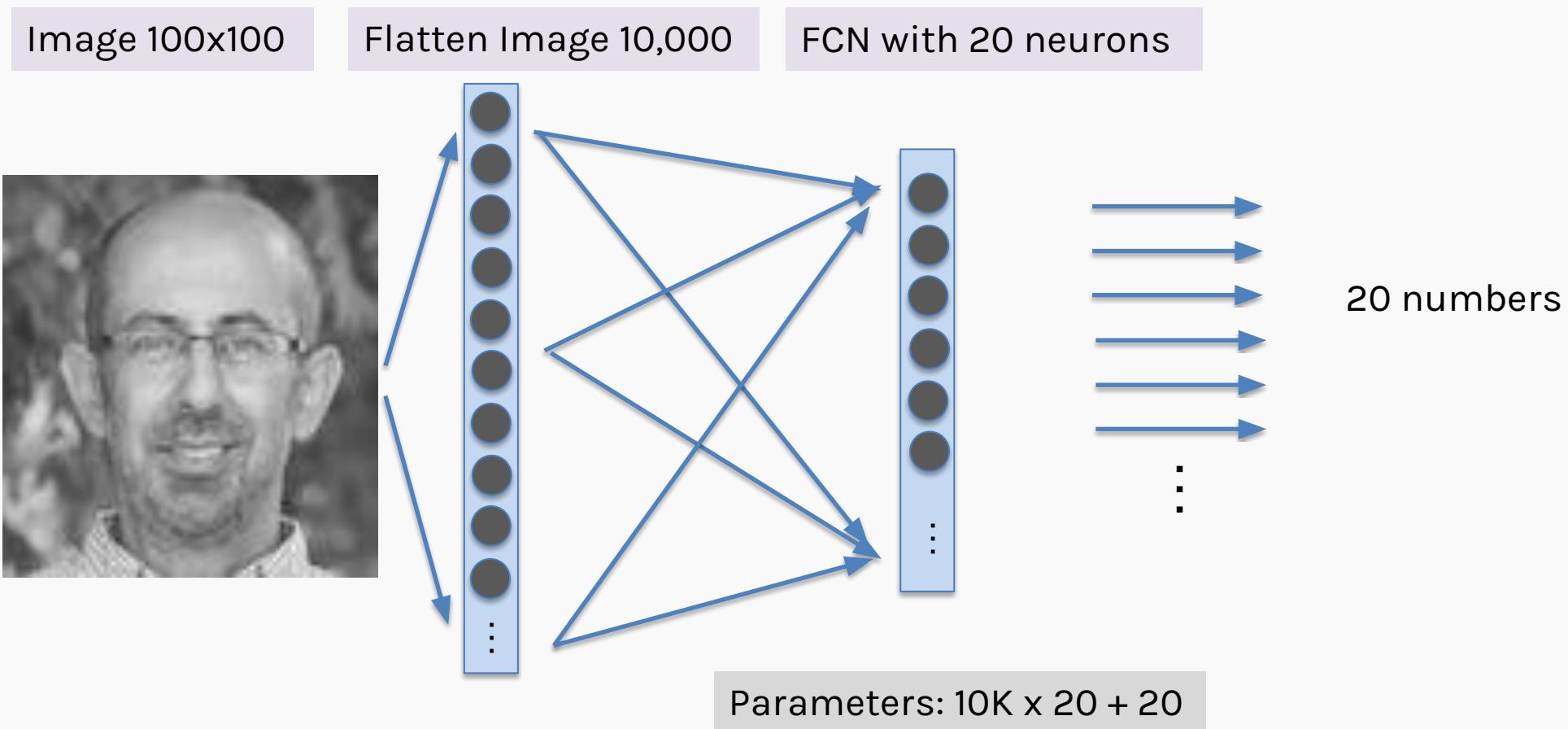
MP3



JPG

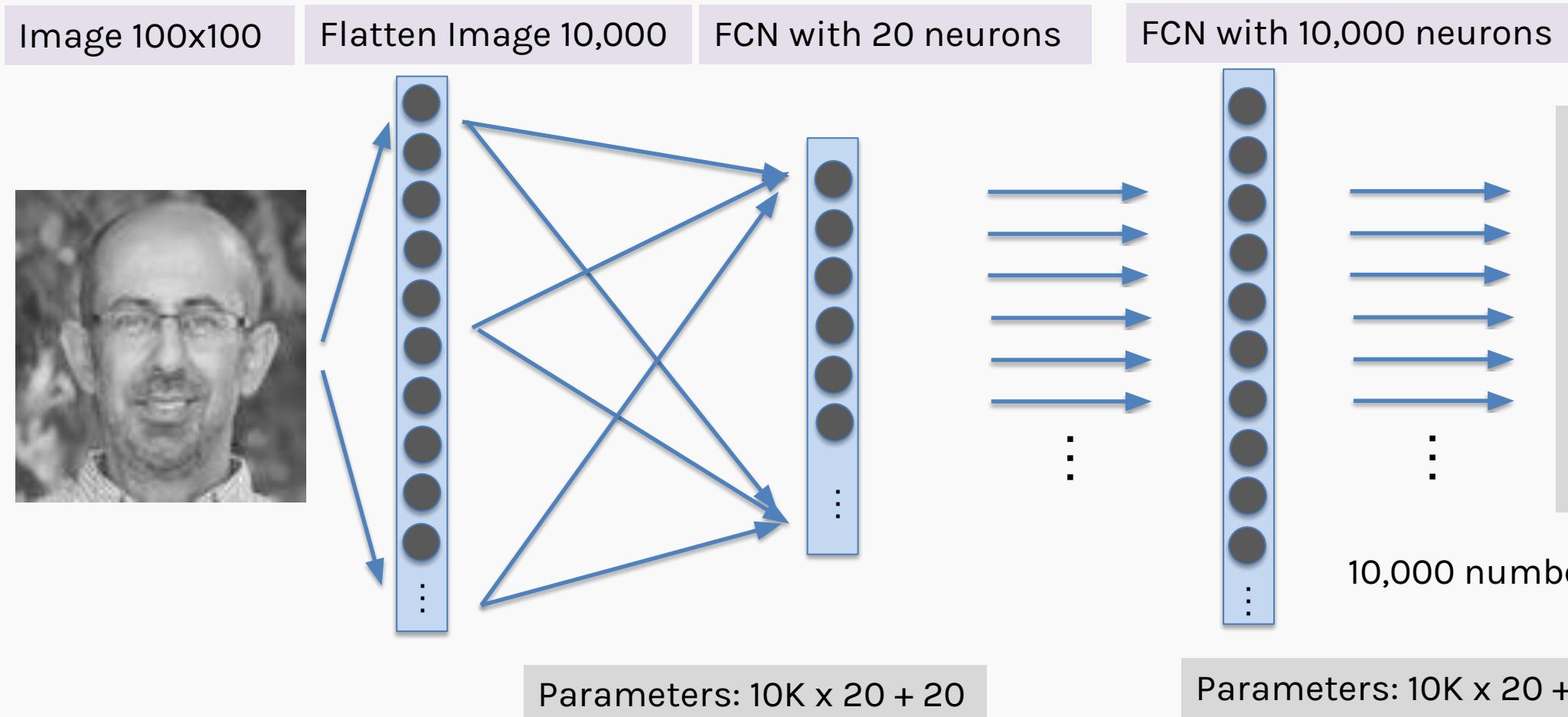
# The simplest autoencoder

**Encode with a simple fully connected network (FCN)**



# The simplest autoencoder

Encode and decode together after training



# Autoencoders in action

Comparing the input and output pixel by pixel.

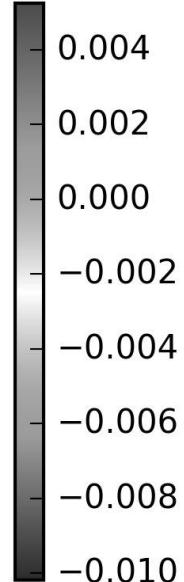
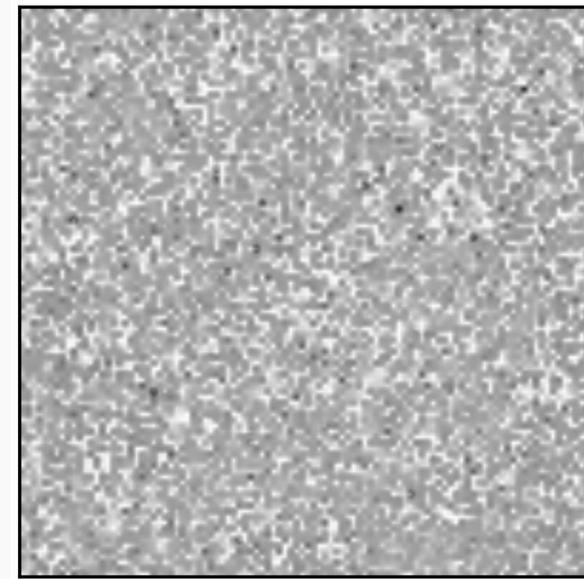
Input Image 100x100



Output Image 100x100

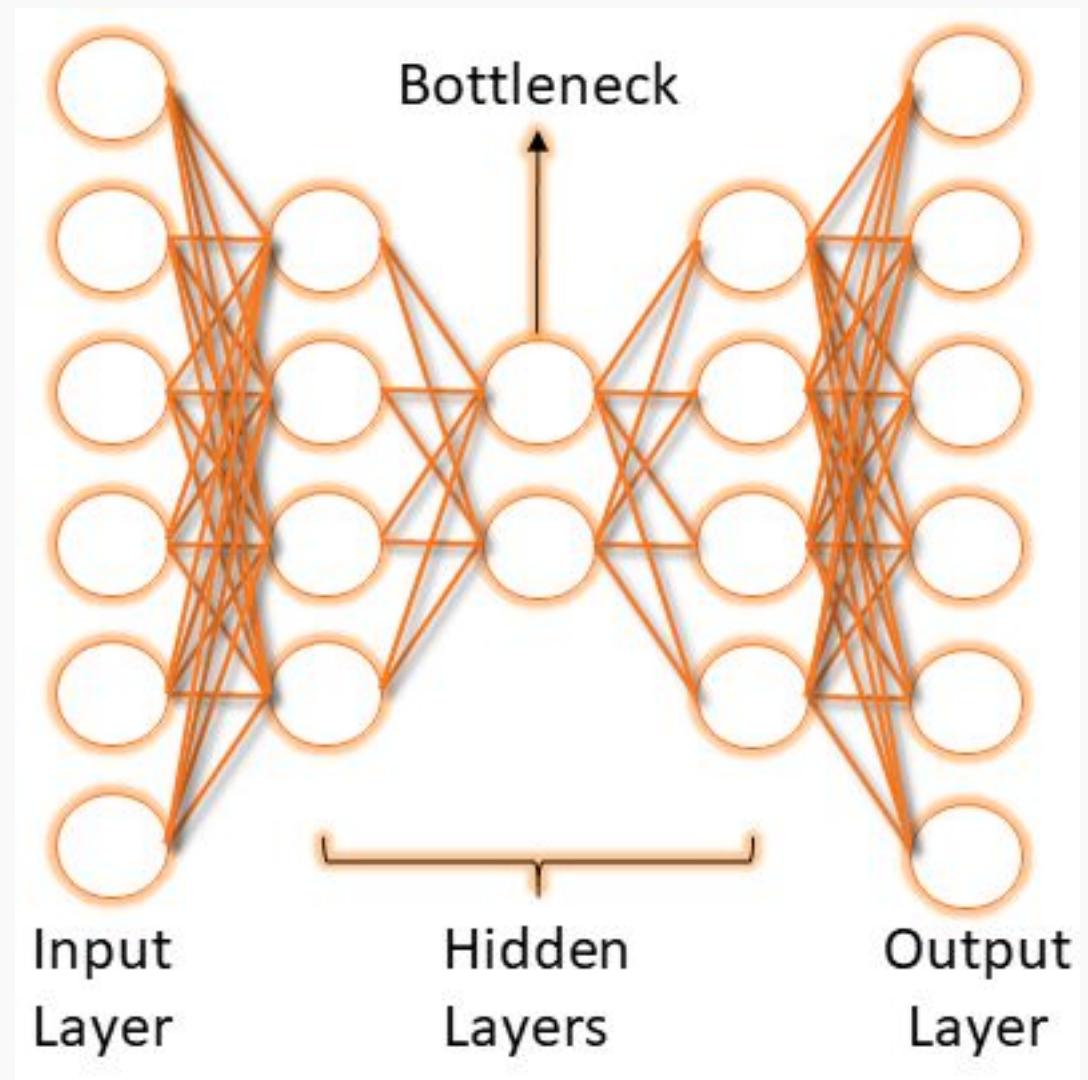
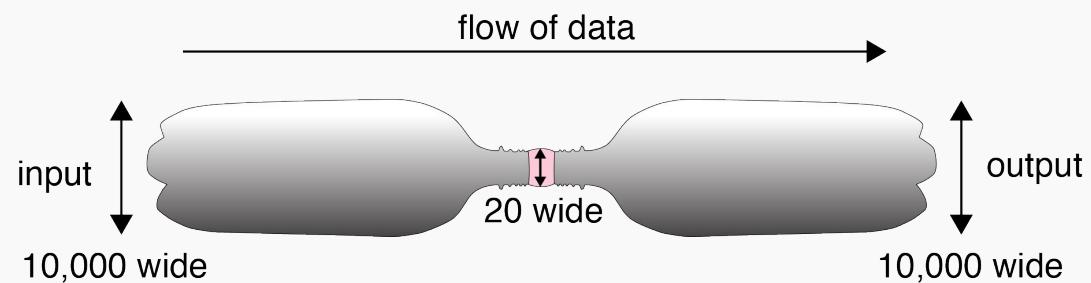


Residuals 100x100

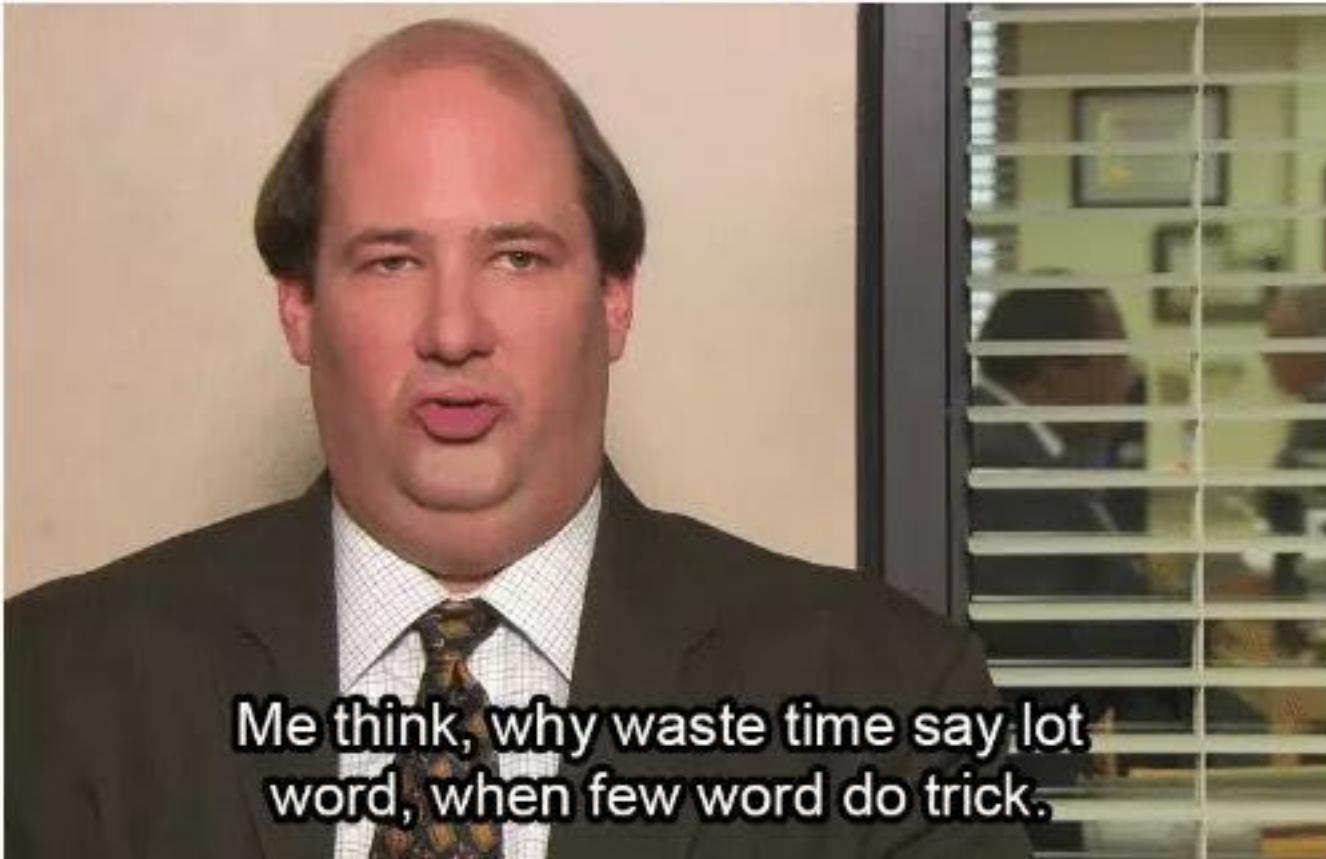


# Bottleneck

- We start with 10,000 elements
- We have 20 in the middle
- And 10,000 elements again at the end



# When you penalize your Natural Language Generation model for large sentence lengths



# Latent variables and latent layer

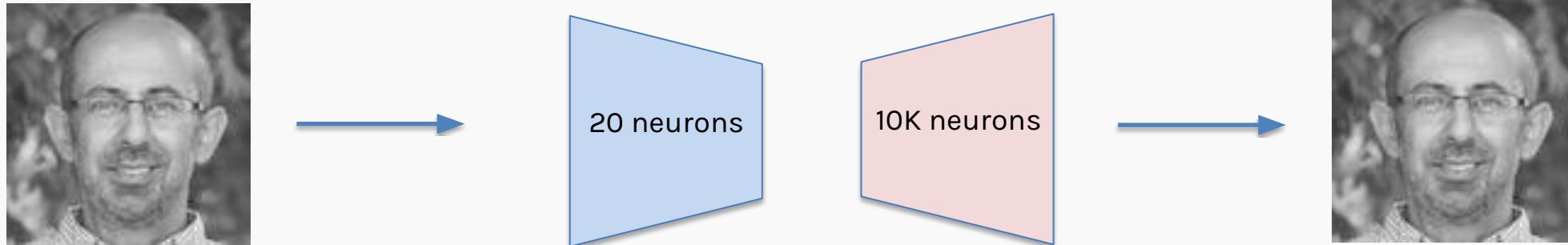
---

We say that an autoencoder is an example of **semi-supervised or self-supervised learning**.

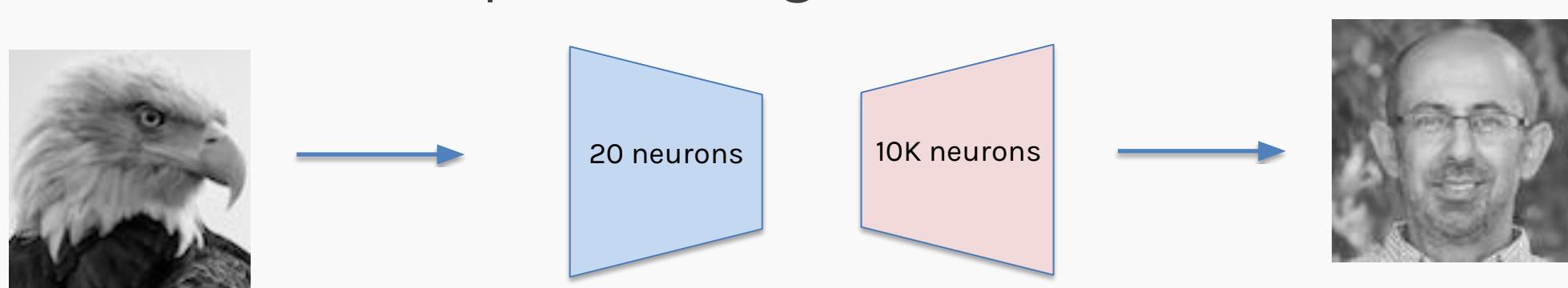
It sort-of is **supervised** learning because we give the system explicit goal data (the output should be the same as the input), and it sort-of isn't supervised learning because we don't have any manually determined labels or targets on the inputs.

# Autoencoders in action (cont)

Passing "Pavlos" to the trained autoencoder returns:



How about if we input the “Eagle”?



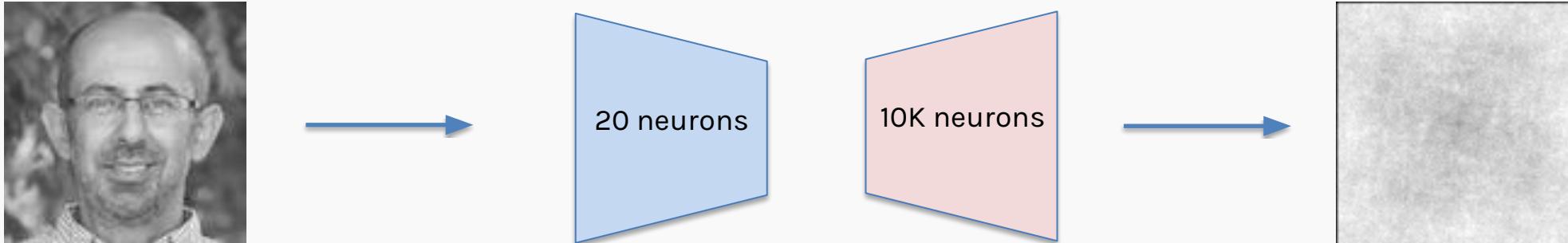
# Autoencoders in action (cont)

We must **train** with a variety of images.



# Autoencoders in action (cont)

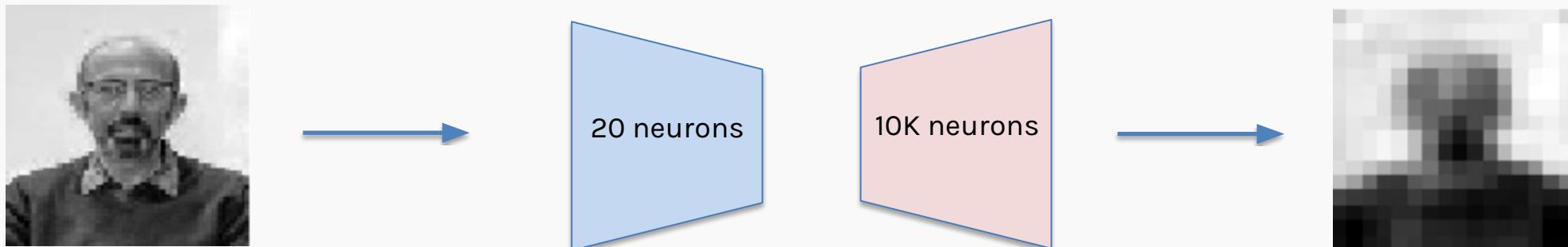
After training with those images, let's test how well it generalizes:



Network never seen anything like this, so it is no surprise that could not reconstruct this.

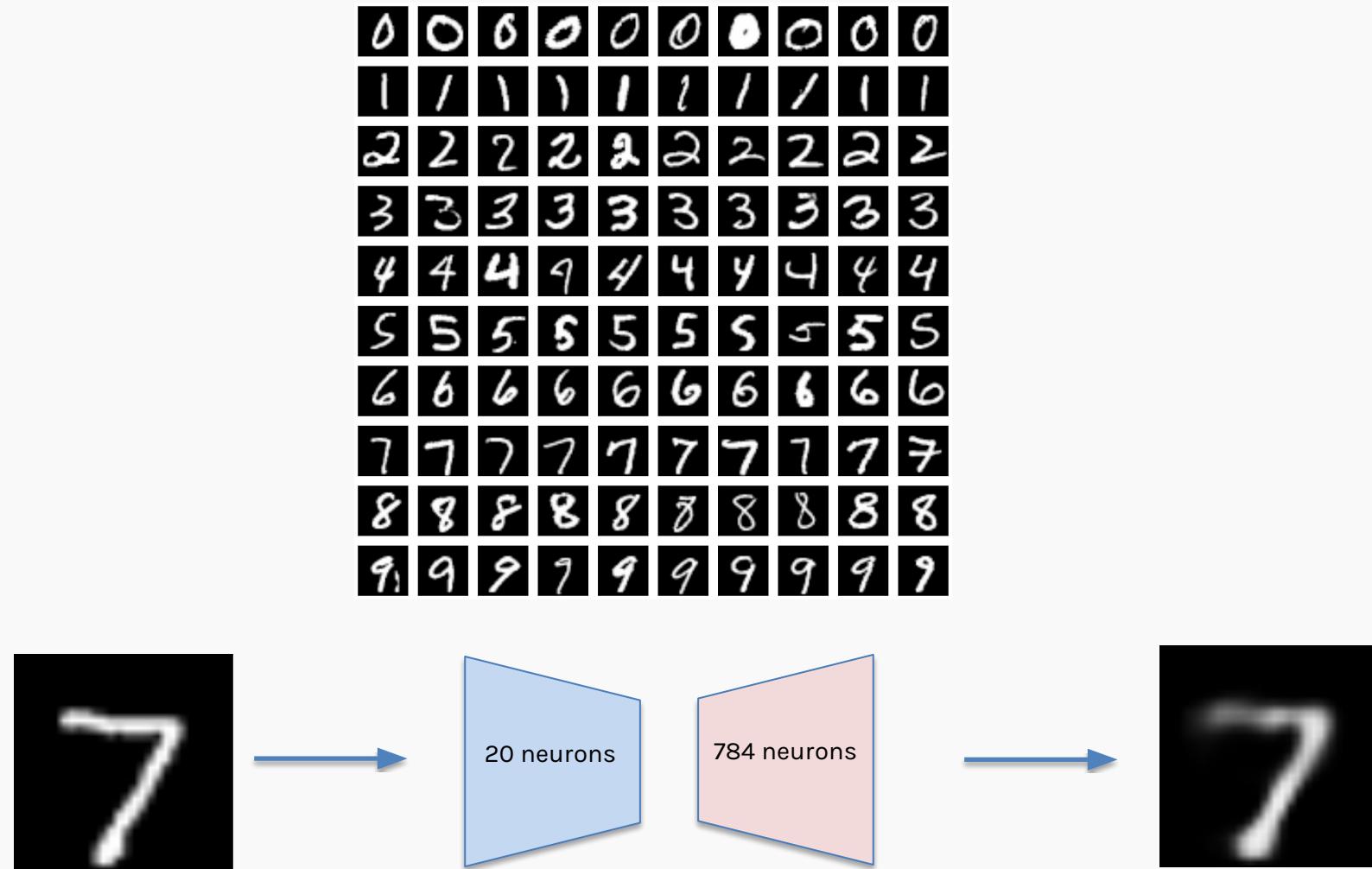
# Autoencoders in action (cont)

We can use a better training set such as the Olivetti faces



# A better autoencoder

MNIST data: train a simple AE with one-layer FCN encoder and one-layer FCN decoder



20 latent variables

original



reconstructed



10 latent variables

original



reconstructed

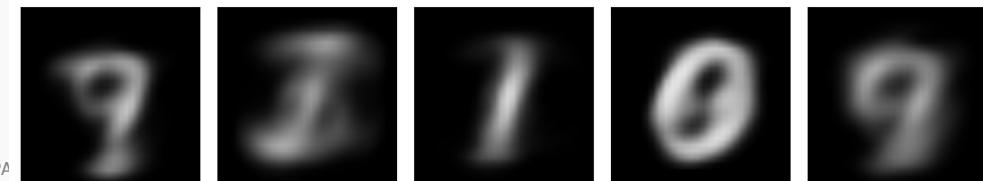


2 latent variables

original



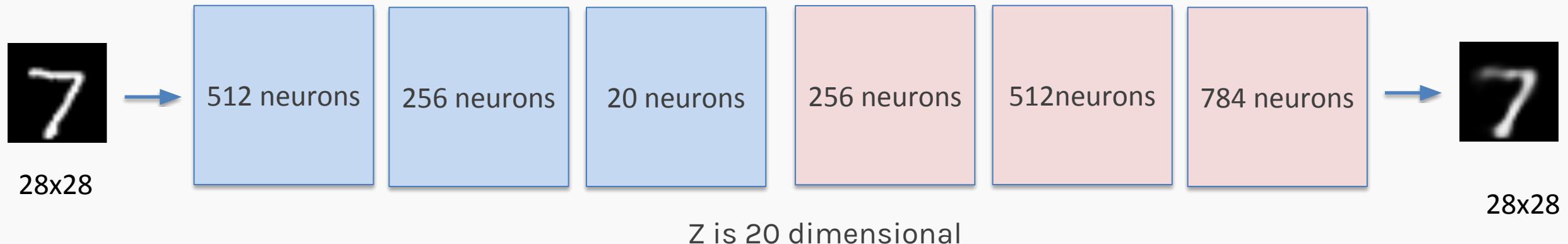
reconstructed



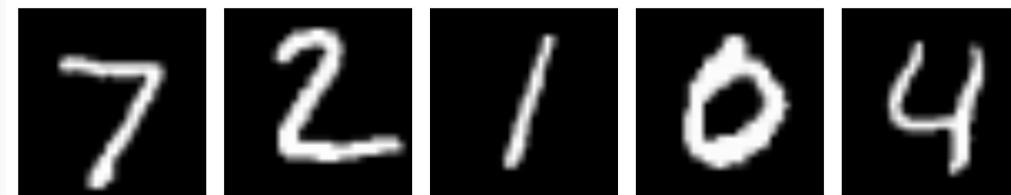
## WHY DOES IT WORK?

# Deeper

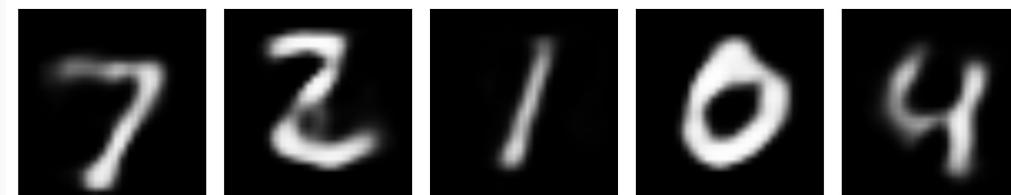
For a better representation we can add neurons in one layer or go deeper.



Original Images



Reconstructed Images

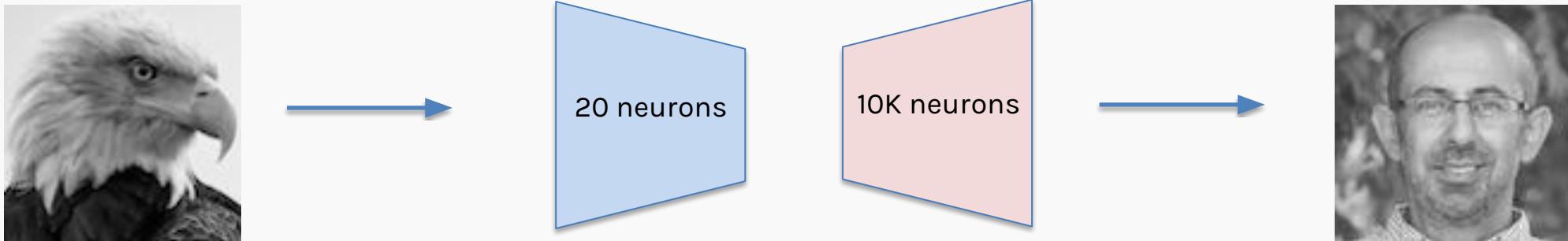
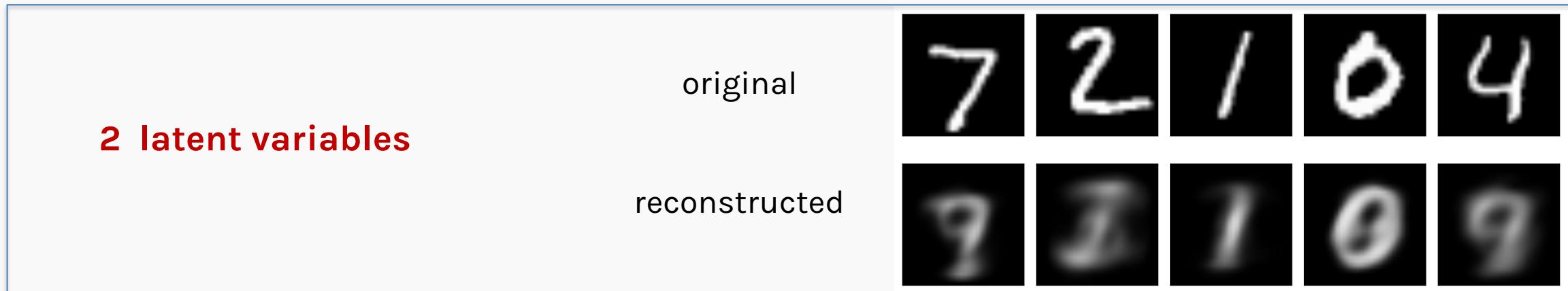


Deeper 20 latent variables  
Shallow 20 latent variables

**DEEPER IS BETTER**

# Exploring autoencoders

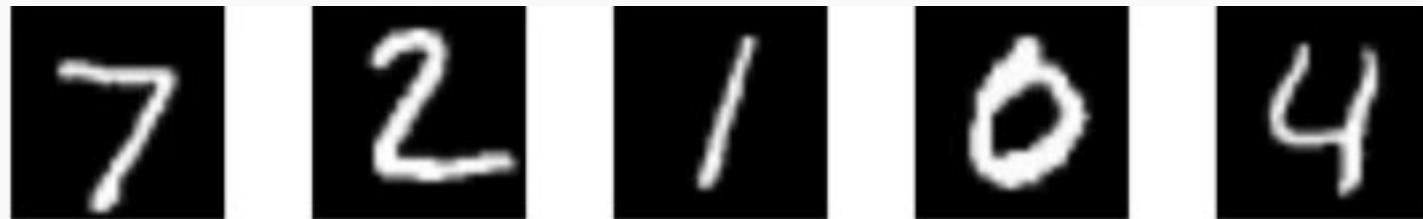
How can we be sure that there is information in the latent space?



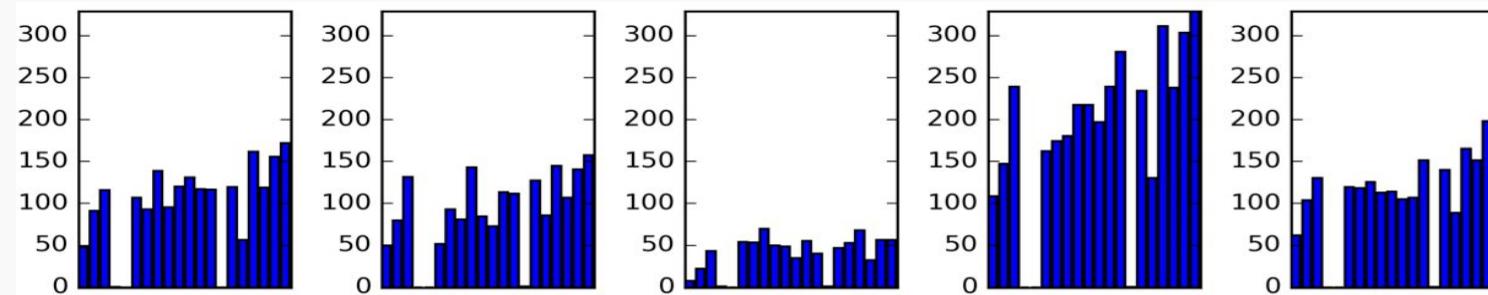
# Exploring autoencoders (cont)

For that let's explore the actual latent space.

Input



Latent space



Each plot a hist of  
the values of the 20  
Z's

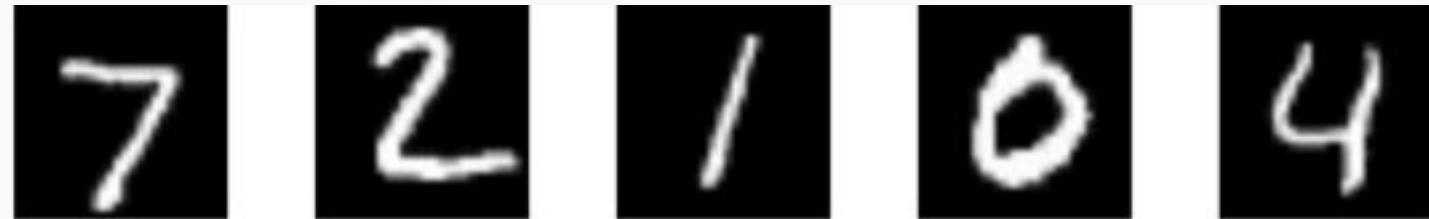
Output



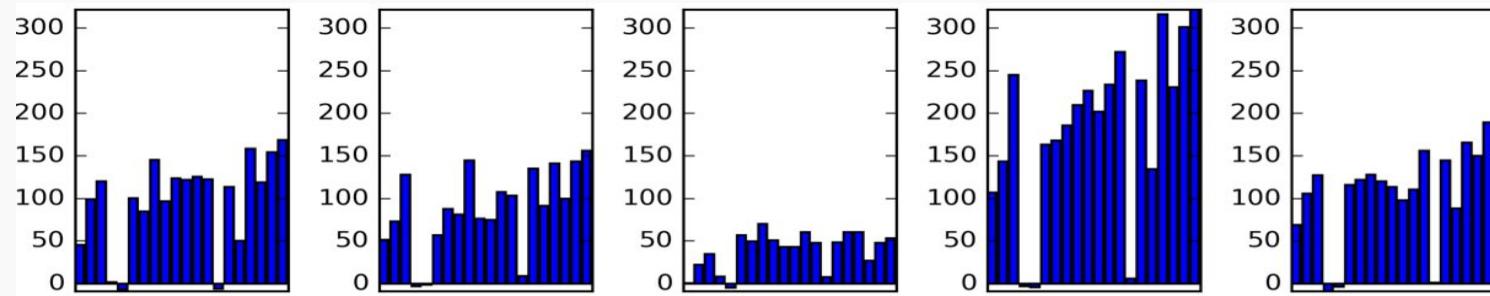
# Exploring autoencoders (cont)

If latent space does not contain any information, then changing the values randomly won't change the output or it will not be robust.

Input



Latent space



Add random noise  
 $\pm 10$  to  $z$ 's

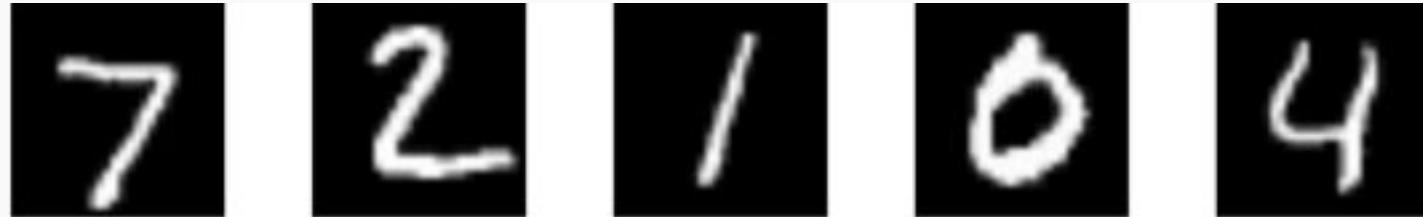
Output



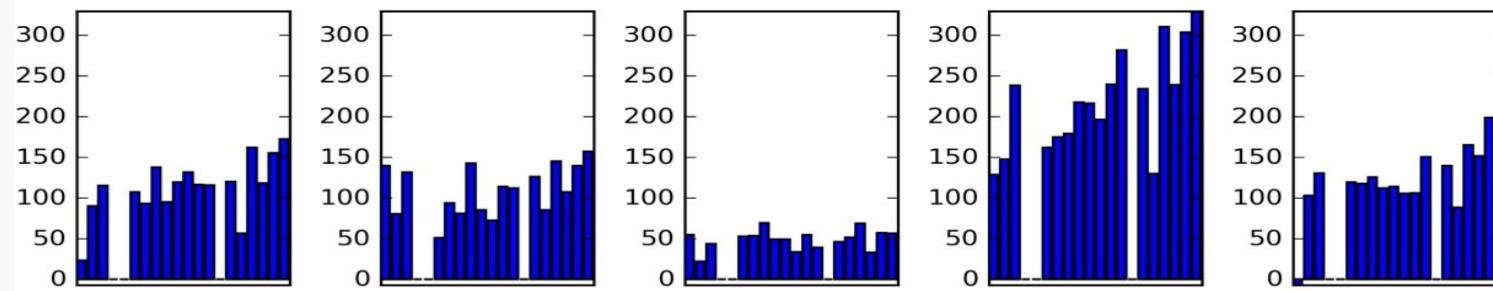
# Exploring autoencoders (cont)

If latent space does not contain any information, then changing the values randomly won't change the output or it will not be robust.

Input



Latent space



Add random noise  
+/-100 to z's

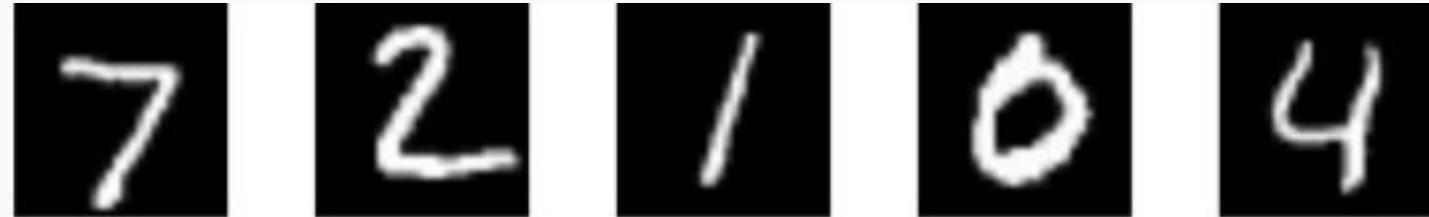
Output



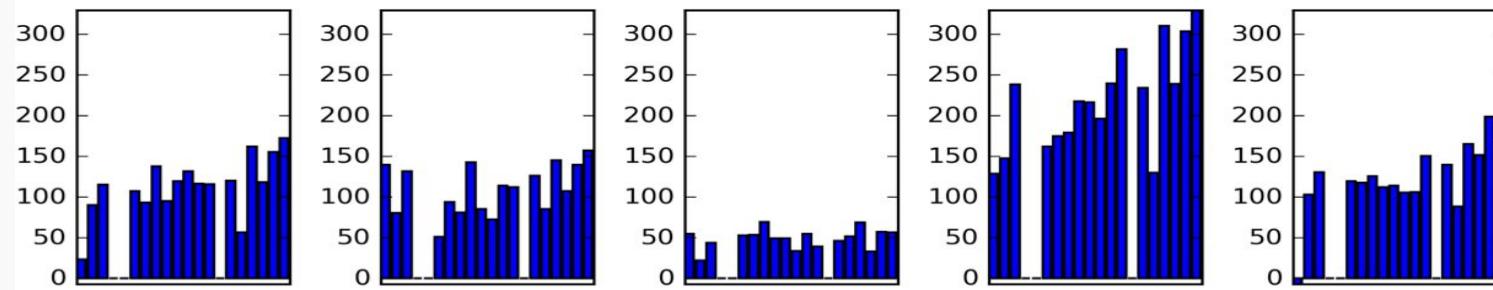
# Exploring autoencoders (cont)

If latent space does not contain any information, then changing the values randomly won't change the output or it will not be robust.

Input



Latent space



Add random noise  
+/-100 to  $z$ 's

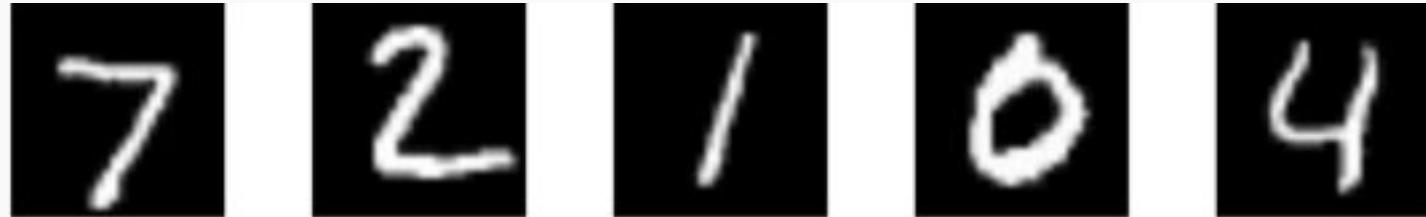
Output



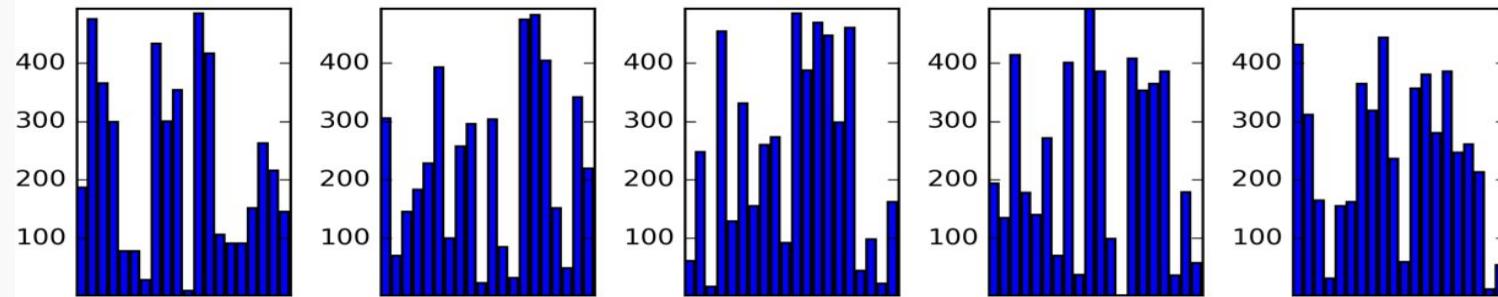
# Exploring autoencoders (cont)

If latent space does not contain any information, then changing the values randomly won't change the output or it will not be robust.

Input



Latent space



Just noise

Output

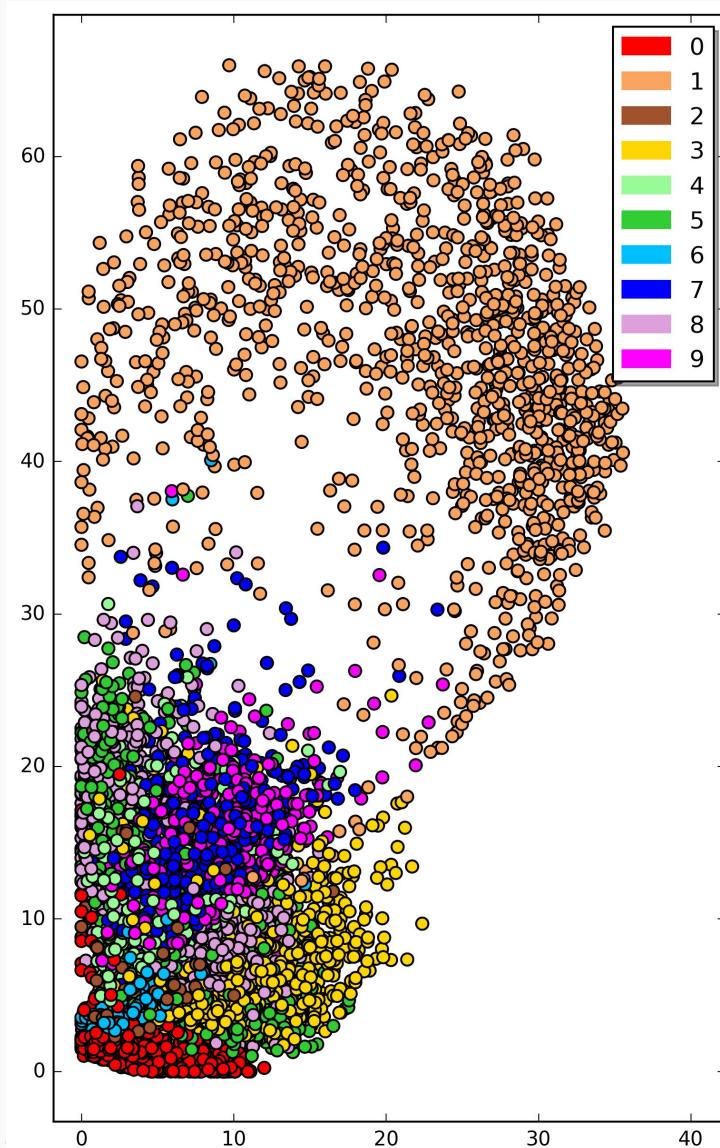


# Parameter space of autoencoder

Further examine the latent space. Is there any separation of the different classes? If the AE learned the “essence” of the MNIST images, similar images should be close to each other.

Plot the latent space and examine the separation.

Here we plot the 2 PCA components of the latent space.



# Blending

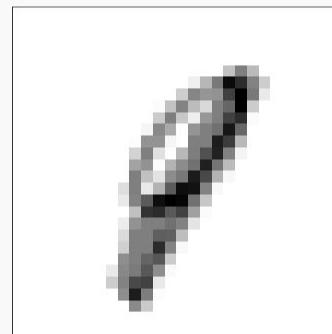
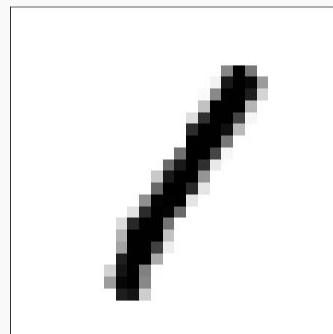
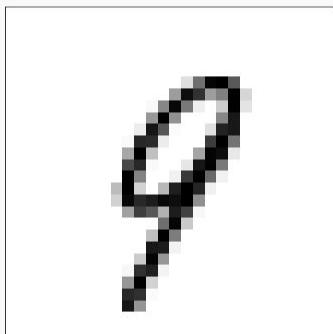
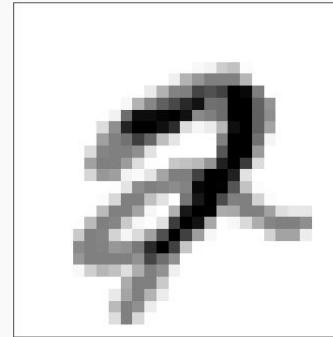
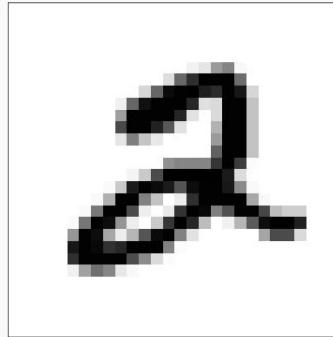
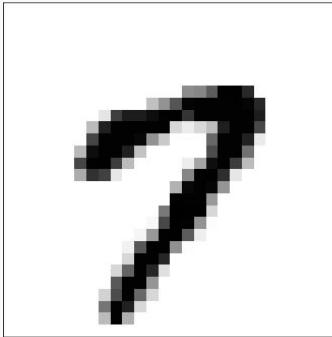
We blend inputs to create new data that is similar to the input data, but not exactly the same.

One example of blending is **content blending** where the content of two pieces of data is directly blended. An example is if we overlay images of a cow and zebra.



# Blending

## Content blending on MNIST images



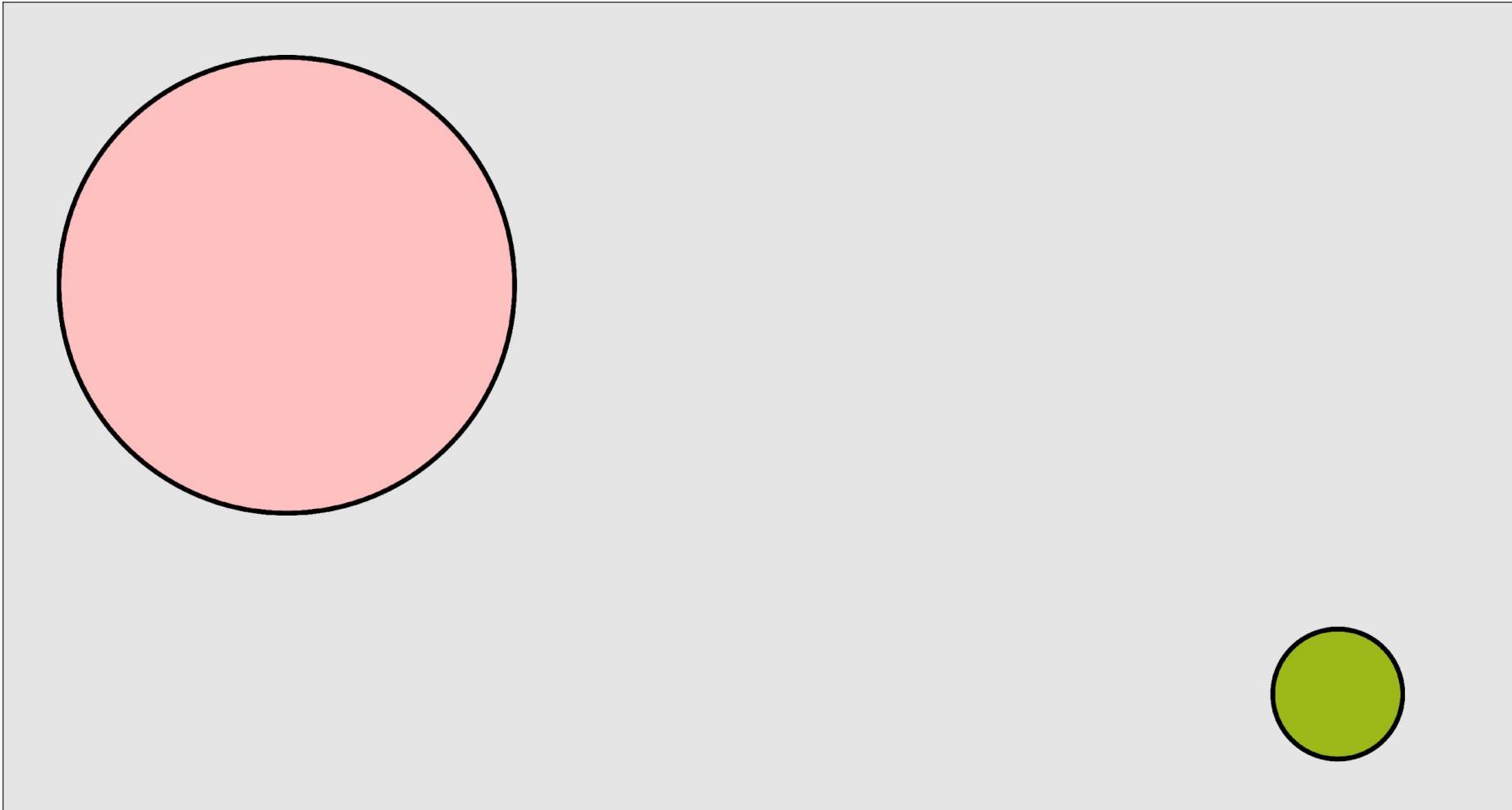
## Blending (cont)

---

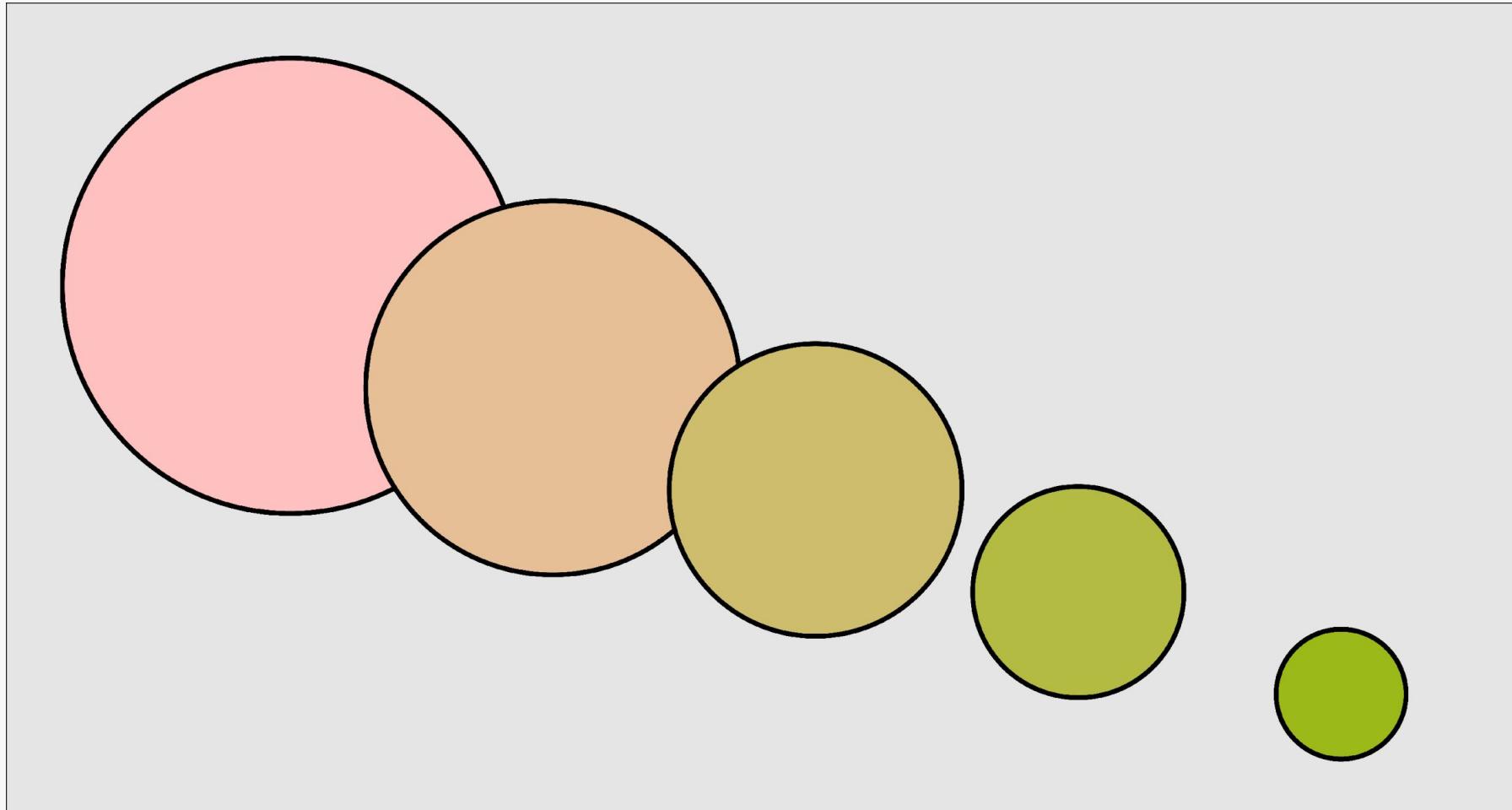
Another type of blending is **parametric** or **representation** blending:

In this type of blending we take advantage of parameterization to describe the objects we're interested in. By engaging in blending in the parameter space, we can create results that blend the inherent qualities of the objects of interest.

# Blending (cont)



# Parametric or representation blending



# Blending in compressed representation

While the blending we've described works well for uncompressed objects what happens when compression is involved?

The compressed form may not be the best representation with what we would like to blend the objects.

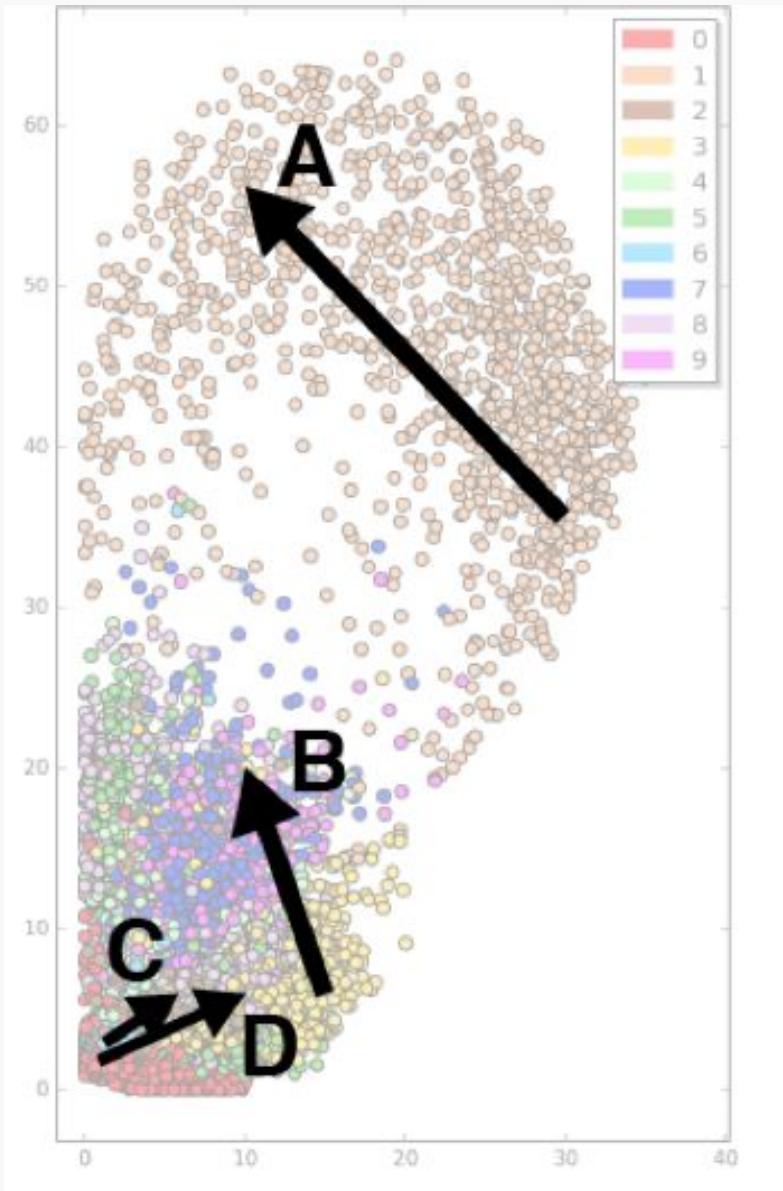
For example, let's take the sounds of the words *cherry* and *orange*.

We can blend these sounds together or we can compress them into written words.

C —— DEFGH**I**JKLMN —— O  
H —— **I**JK**L**MNOPQ —— R  
E———— **D**C**B**———— A  
R———— **Q**P**O**———— N  
R———— QPON**M**L**K**J**I**H —— G  
YXWVUTSRQP**ON**ML**K**J**I**H**G**F**E**



# Blending Latent Variables



Back to the example of MNIST.

1. We start at the start of the arrows in latent space and then move to end of the arrow in 7 steps.
2. For each value of  $z$  we use the already trained decoder to produce an image.

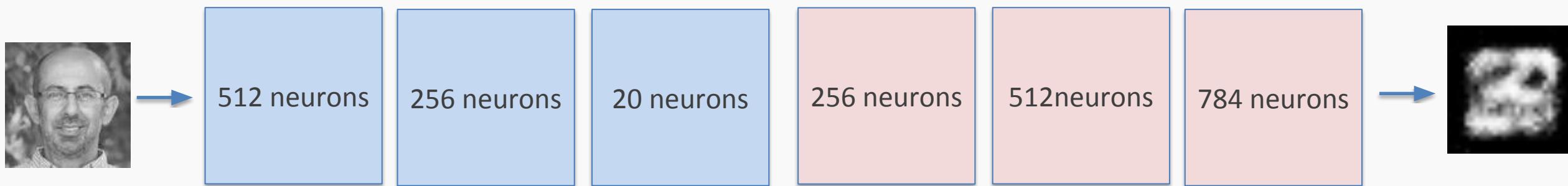


GIFs  
om Reconstructing  
Input Image



# Applying to novel input

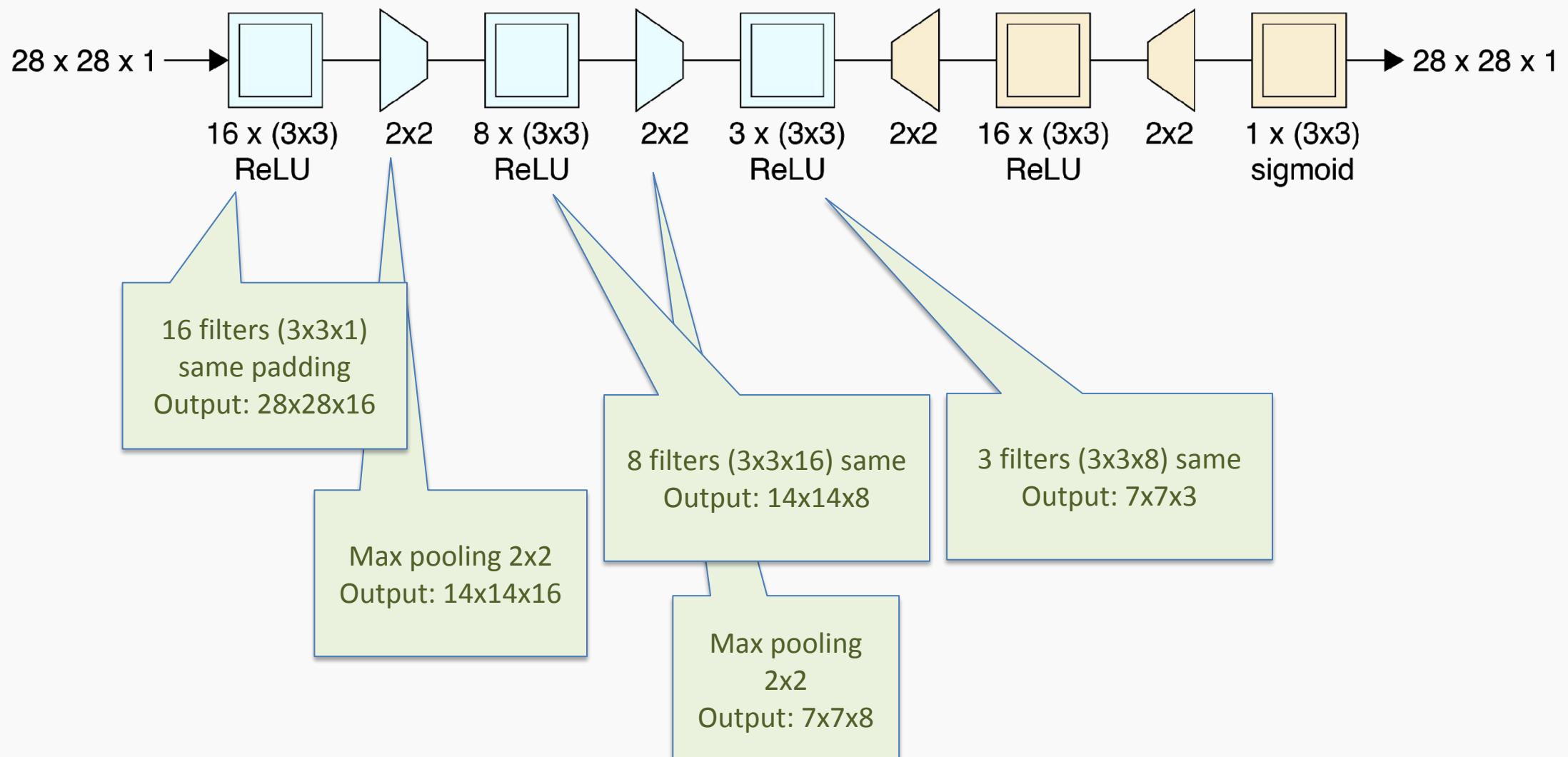
Check again if the deep AE trained on MNIST works with “Pavlos” image?



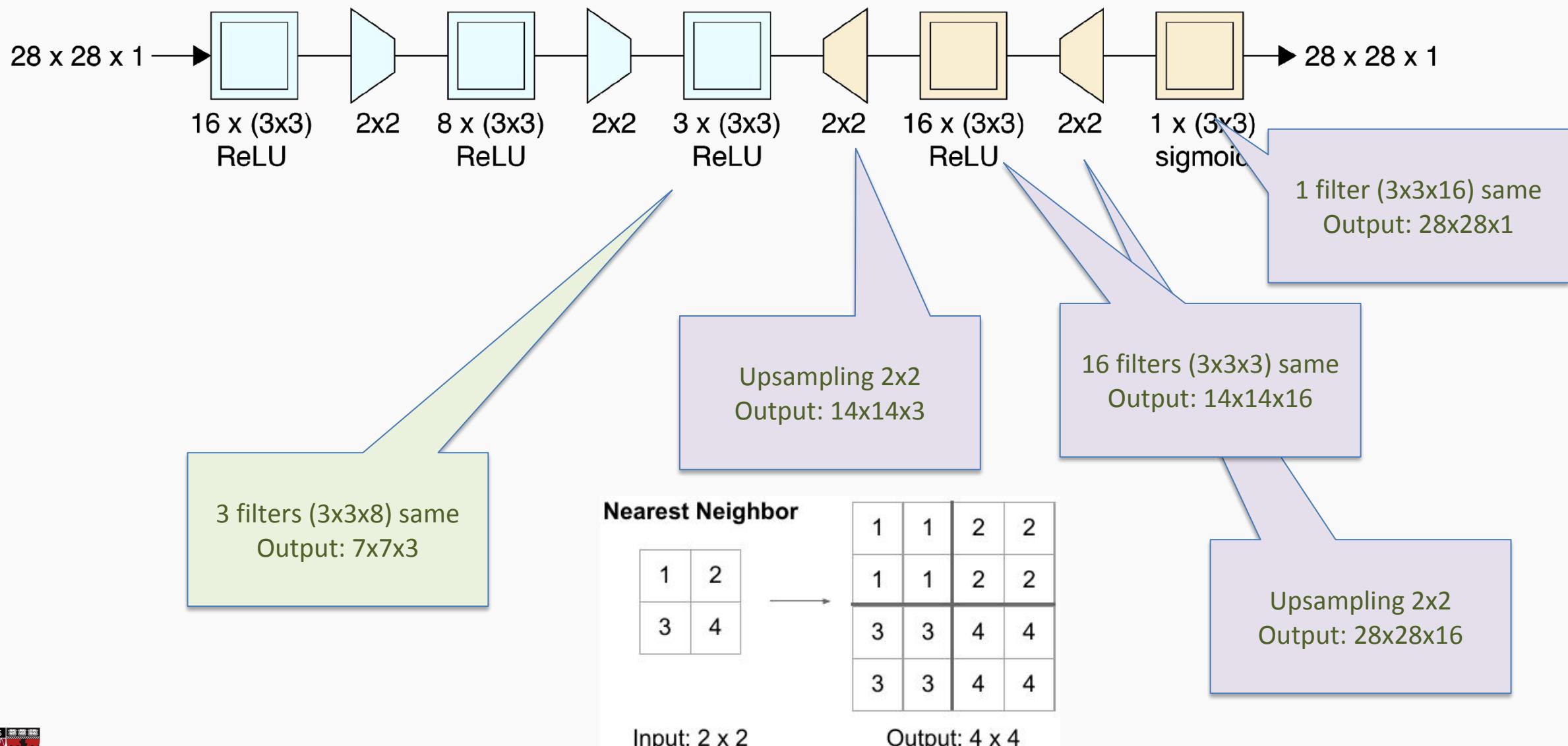
**NO**

Since we are dealing with images, it is best to use CNNs

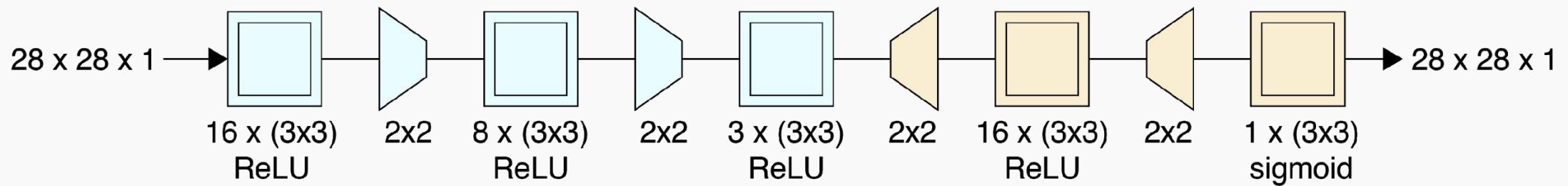
# Convolutional Autoencoders



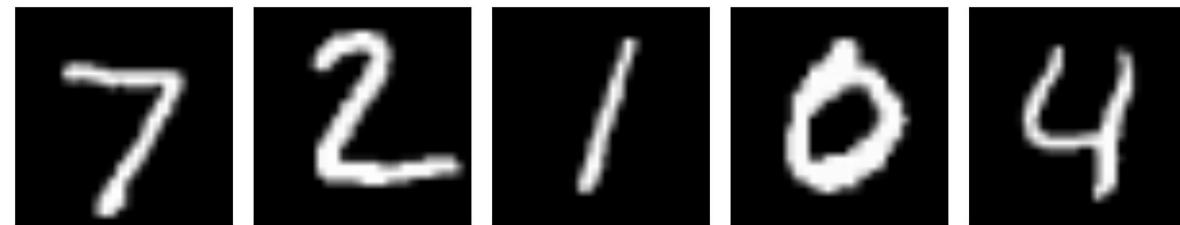
# Convolutional Autoencoders



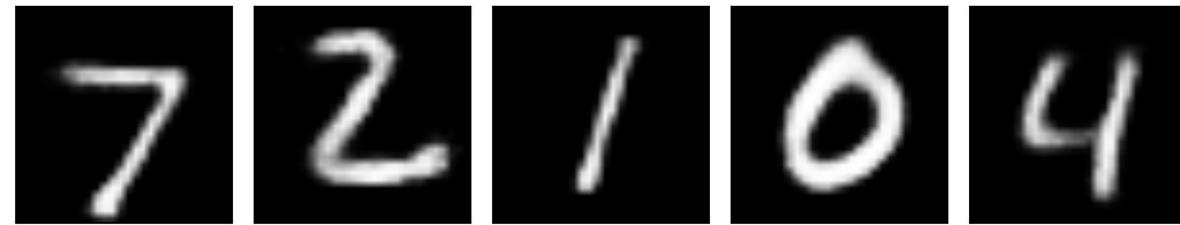
# Convolutional Autoencoders



Original Images



Reconstructed Images  
with Conv AE

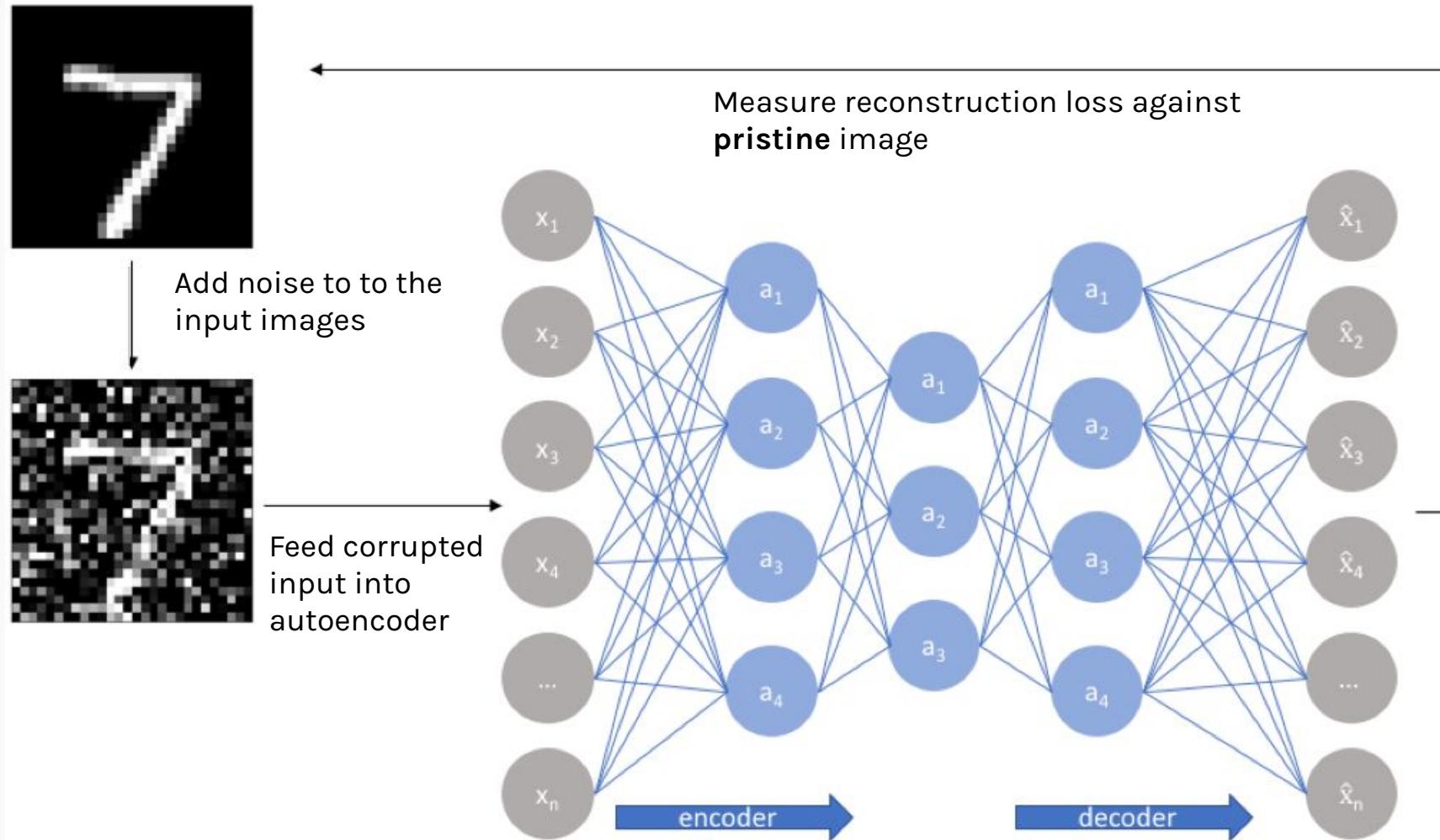


ConvAE latent variables  
DeepFCN latent variables

# Denoising

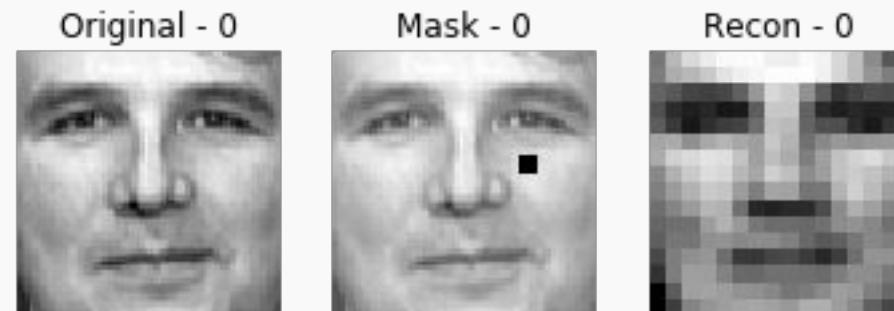
A popular use of autoencoders is to remove noise from samples.

Start with a **pristine** images



# Infilling

Claim is that AE learns the contextual information of the images. That would mean if some parts of the image is missing then



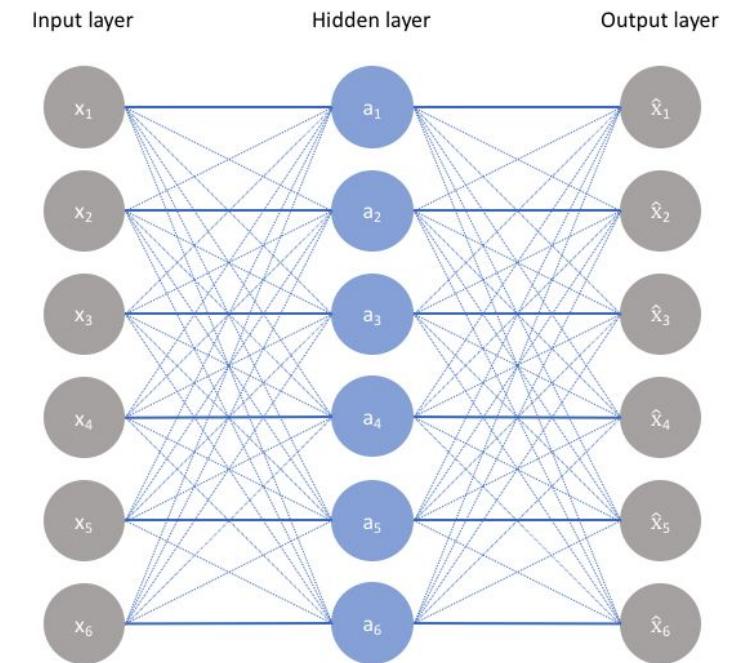
# Overcomplete Autoencoders

Over-complete autoencoder is when  $z$  has **greater dimension** than  $x$ .

Autoencoder may simply copy input to output without learning anything useful – Mark's point.

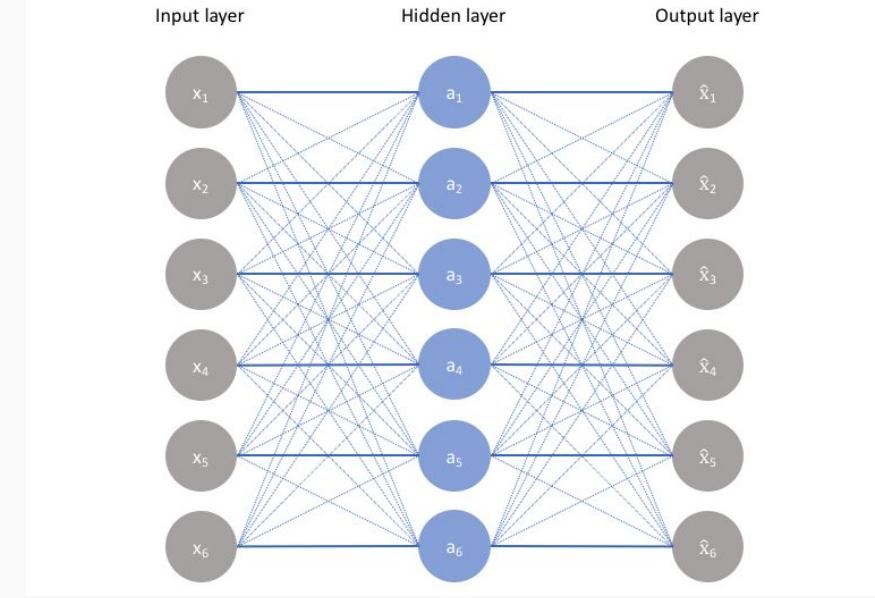
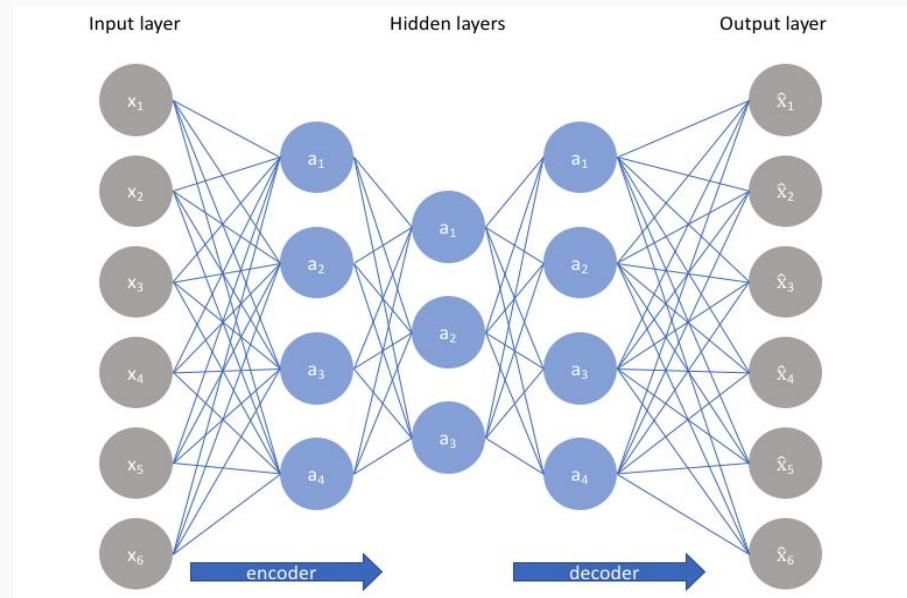
The ideal autoencoder model balances the following:

1. Sensitive to the inputs enough to accurately build a reconstruction.
2. Insensitive enough to the inputs that the model doesn't simply memorize or overfits the training data.



# Overcomplete Autoencoders

- We've assumed so far that the size of the bottleneck is smaller than the size of the inputs - this is called an **undercomplete autoencoder**.
- The case in which the size of the bottleneck is greater than or equal to the number of inputs we call an **overcomplete autoencoder**.

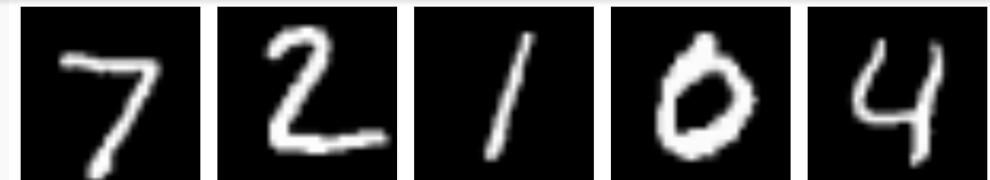


# Overcomplete Autoencoders

The size of the bottleneck (i.e. the number of latent variables) makes a difference!

20 latent variables

original

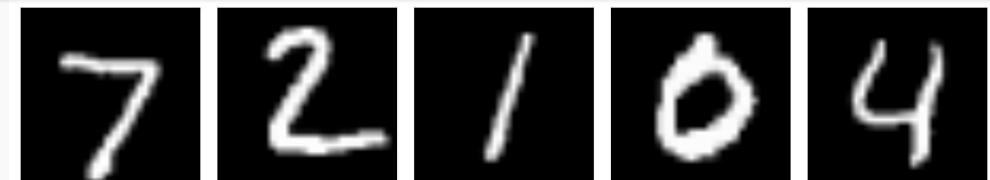


reconstructed

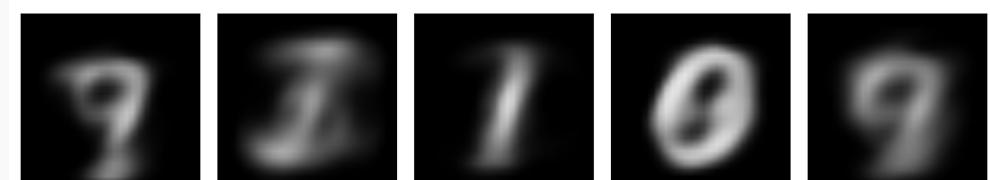


2 latent variables

original



reconstructed



61

# Regularized Autoencoders

---

- Sparse autoencoders
- Contractive autoencoders
- Denoising autoencoders and dropout



# Sparse Autoencoders

---

This trade-off requires the model to maintain only the variations in the data required to reconstruct the input without holding on to redundancies within the input.

**Question:** How to achieve this?

For most cases, this involves constructing a loss function where one term encourages our model to be sensitive to the inputs (ie. reconstruction loss  $\mathcal{L}(x, \hat{x})$ ) and a second term discourages memorization/overfitting (ie. an added regularizer).

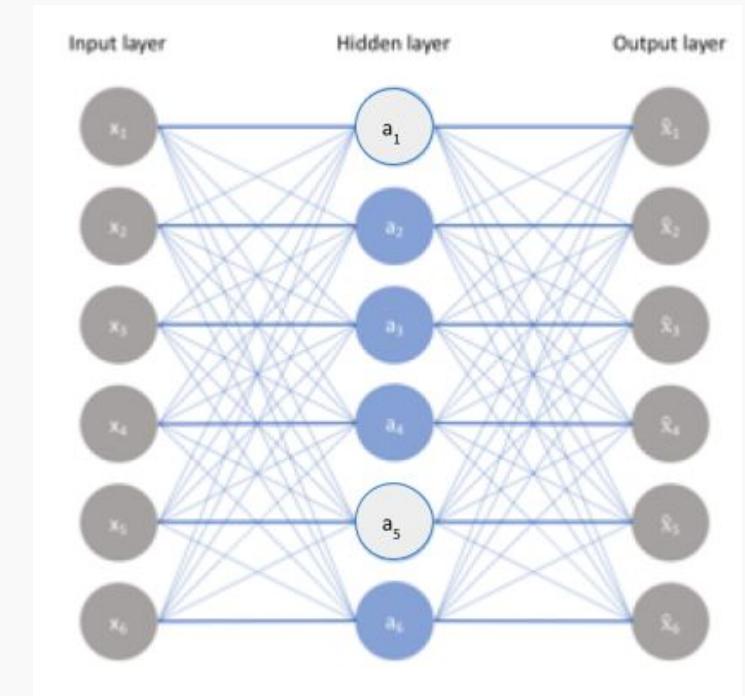
$$\mathcal{L}(x, \hat{x}) + \text{reg}$$



# Sparse Autoencoders

- We allow our network to sensitize individual hidden layer nodes toward specific attributes of the input data.
- A sparse autoencoder is selectively activate regions of the network depending on the input data.
- Limiting the network's capacity to memorize the input data without limiting the networks capability to extract features from the data.

$$\mathcal{L}(x, \hat{x}) + \lambda \sum_i |z_i|$$



# Contractive Autoencoders

One would expect that **for very similar inputs, the learned encoding would also be very similar.**

We can explicitly train our model for this to be the case by requiring that the derivative of the hidden layer activations are small with respect to the input.

**Question:** How do we find how much the encoded space would change if the input changes?

# Contractive Autoencoders

---

## Derivatives

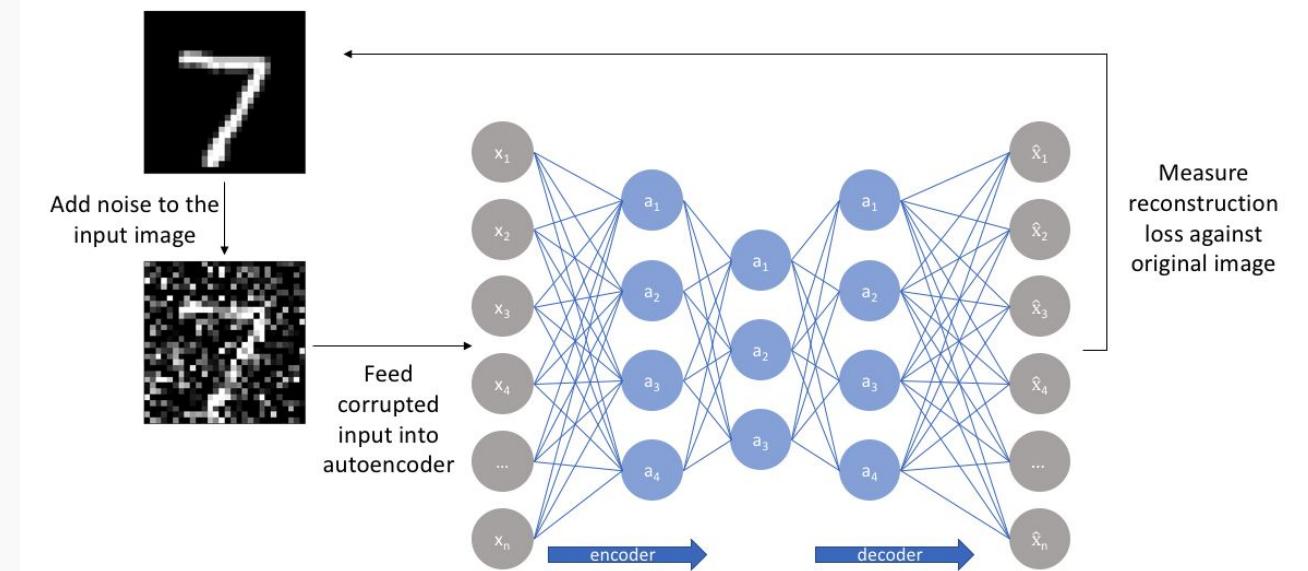
$$\mathcal{L}(x, g(f(x))) + \lambda \sum_i \|\nabla_x z_i\|^2$$

This forces the model to learn a function that does not change much when  $x$  changes slightly. Because this penalty is applied only at training examples, it forces the autoencoder to learn features that capture information about the training distribution.

# Denoising Autoencoders

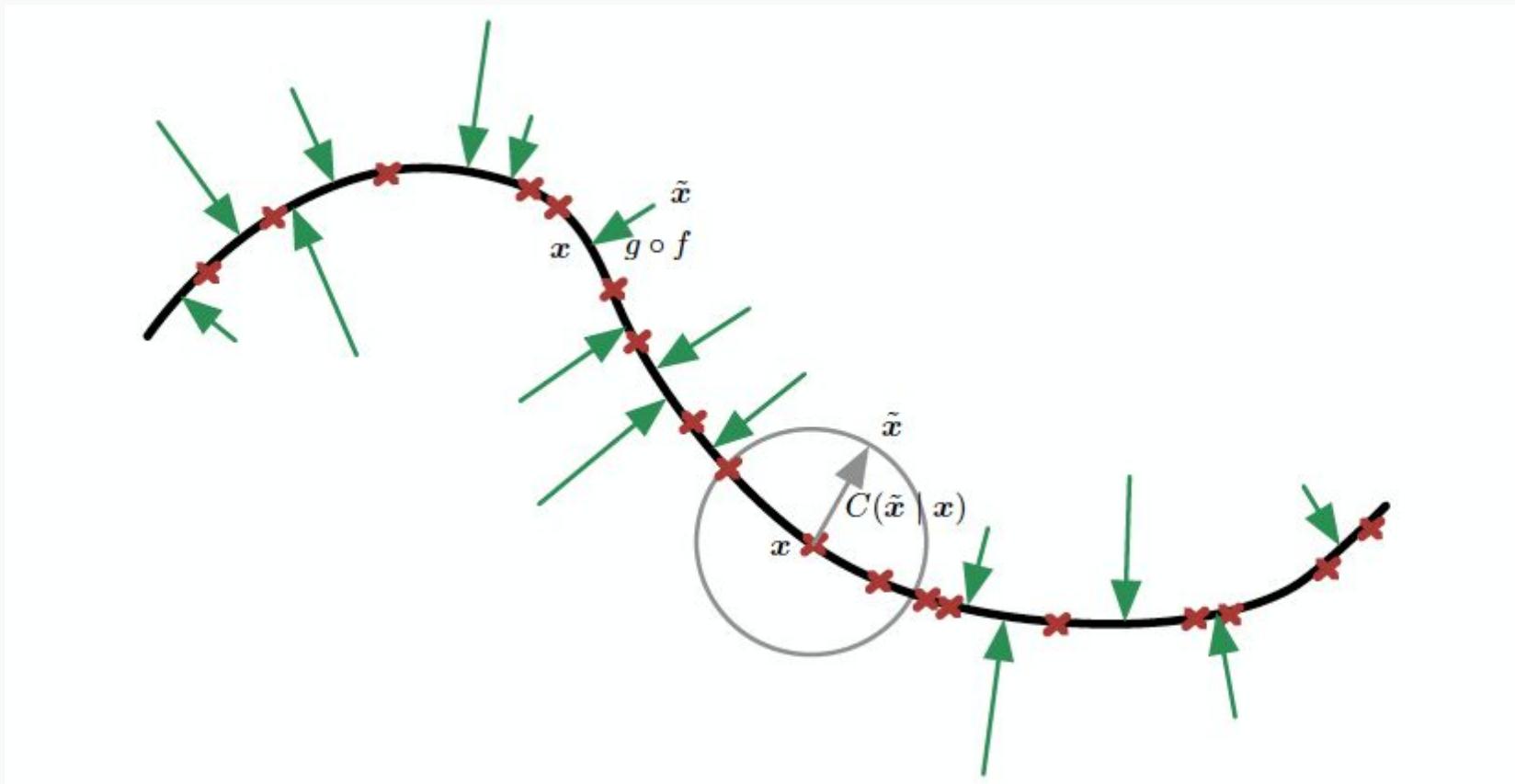
The denoising autoencoder is an autoencoder that receives a corrupted data point as input and is trained to predict the original, uncorrupted data point as its output.

For each epoch:



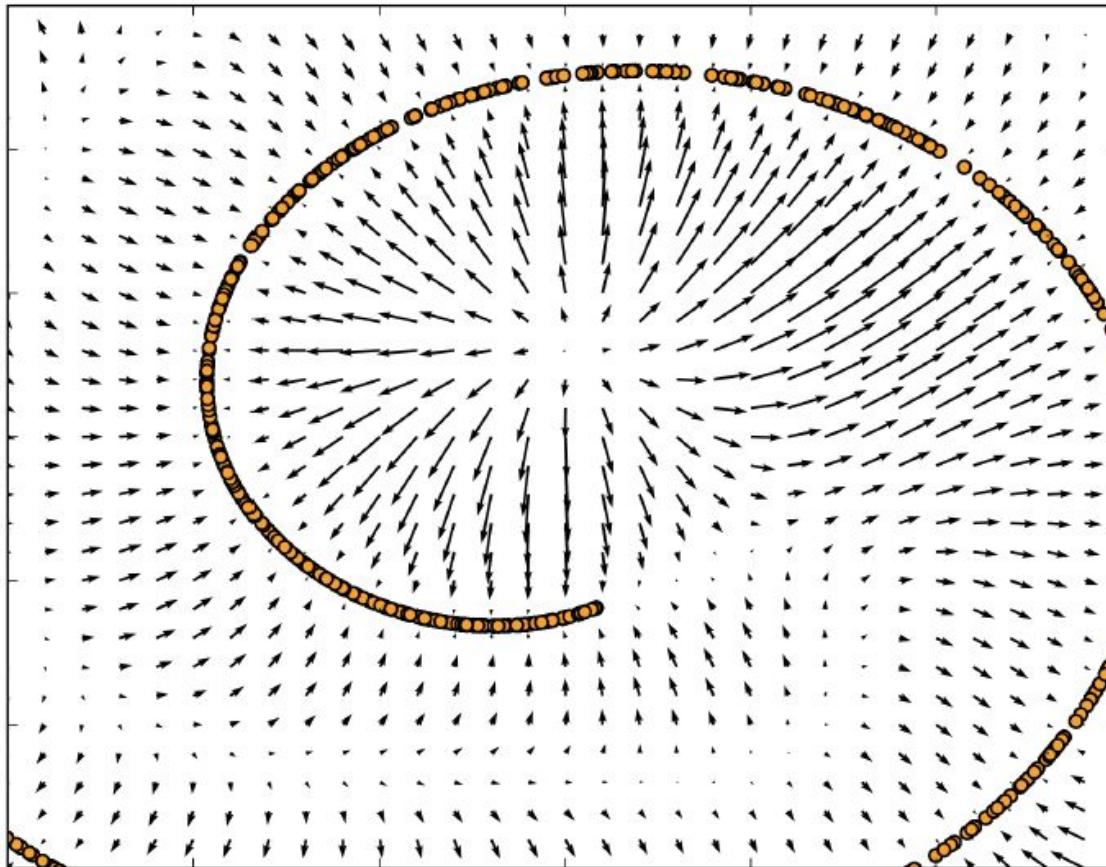
# Denoising Autoencoders (cont)

Denoising autoencoders learn a manifold. Vector field learned by denoising autoencoder. Each arrow is proportional to  $g(f(x)) - x$

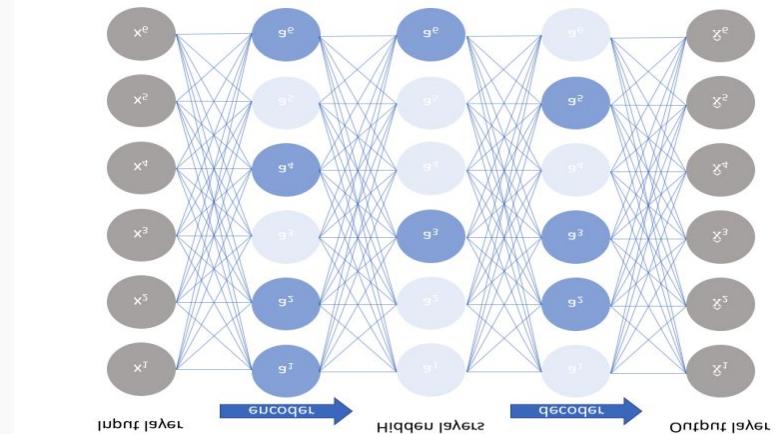


# Denoising Autoencoders (cont)

Vector field learned by denoising autoencoder. Each arrow is proportional to  $g(f(x)) - x$



We allow our network to sensitize individual hidden layer nodes toward specific attributes of the input data.



# Problems with Autoencoders

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

