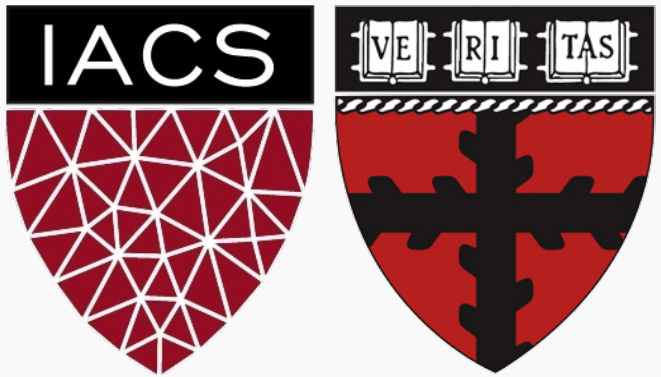
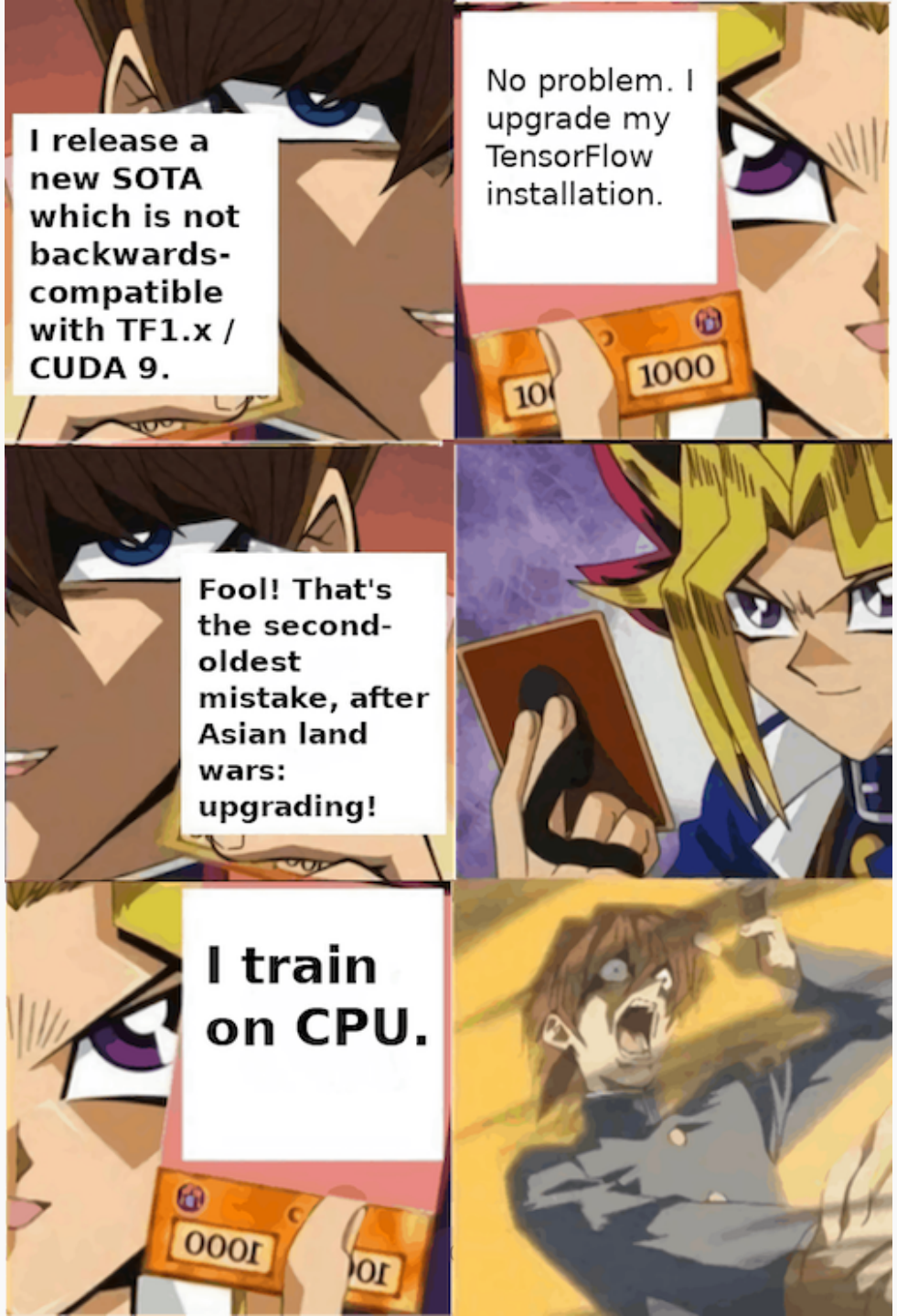
Lecture 12: Autoencoders

CS109B Data Science 2

Pavlos Protopapas and Mark Glickman




I release a new SOTA which is not backwards-compatible with TF1.x / CUDA 9.
No problem. I upgrade my TensorFlow installation.
1000
Fool! That's the second-oldest mistake, after Asian land wars: upgrading!
I train on CPU.


IT IS FRIDAY, ALL MY FRIENDS ARE AT PARTY
BUT, I'M HERE LEARNING BACK PROPAGATION
memegenerator.net

Homework 3: Deadline is …..

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?

Y. LeCun

- **"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**

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

- Brief history of encoding/decoding
- Inside Autoencoders
    - Convolutional 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:

$$x \rightarrow y$$
$$y = f(x) + \epsilon$$

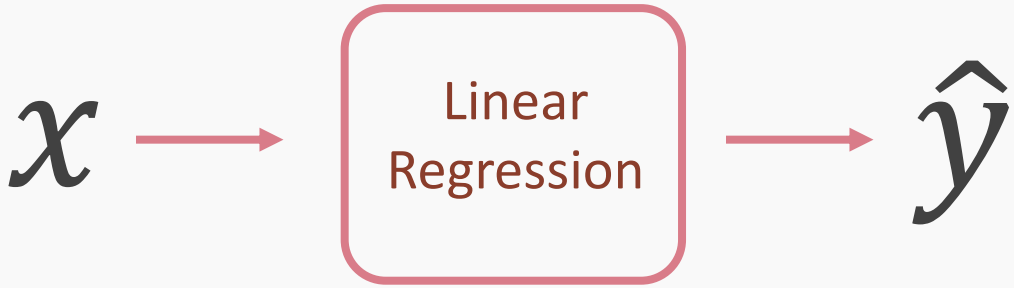
Our goal is to find an estimate of $f(x)$ which we will call $\hat{f}(x)$.

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

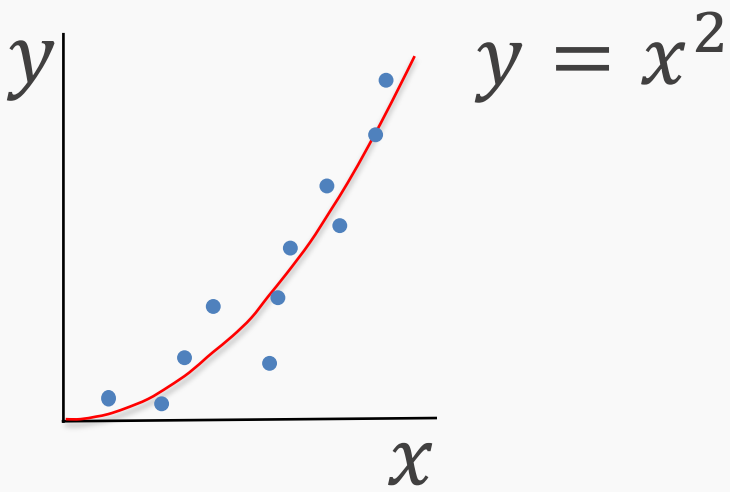
Neural networks are one of many possible methods we can use to obtain the estimate $\hat{f}(x)$.

Representational Learning

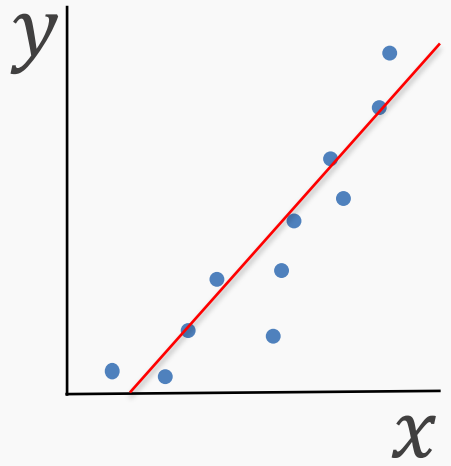
Representation Matters



Fit exponential function



Do your best !

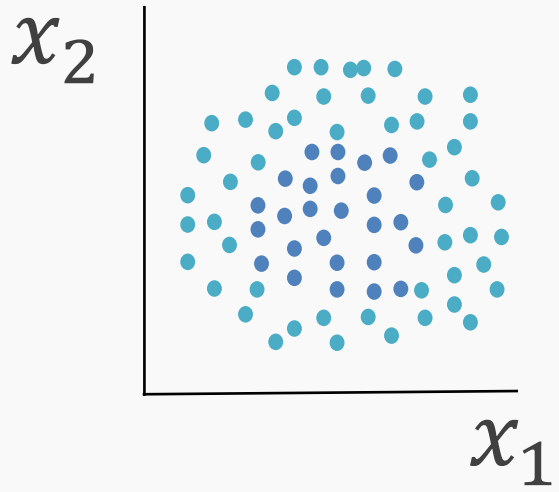


Representational Learning

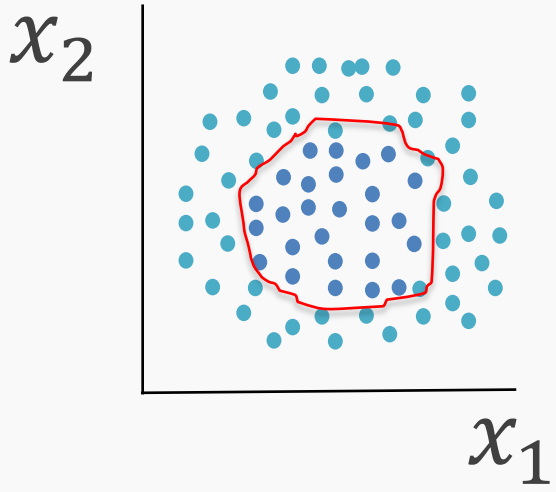
Representation Matters

x_1, x_2 → Logistic Regression → $\hat{y}$

Fit polynomial function

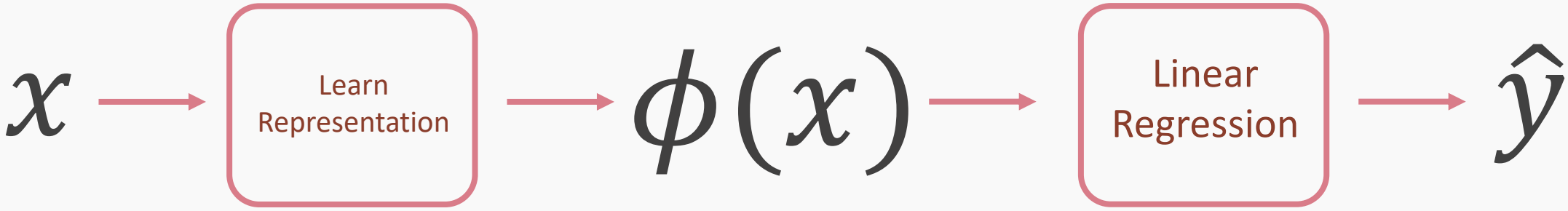


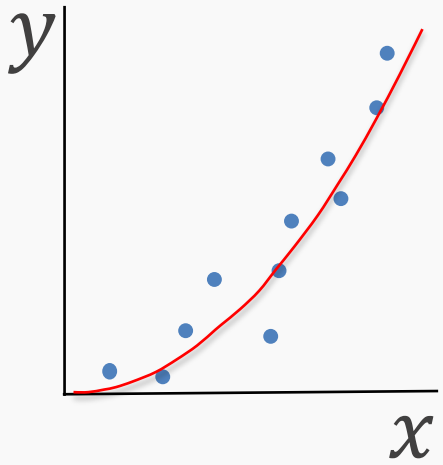
Do your best !



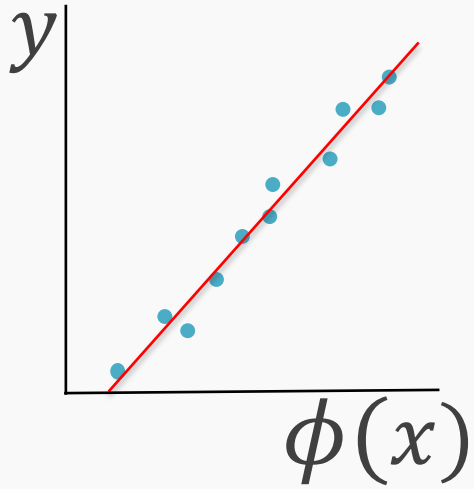
Representational Learning

Representation Matters



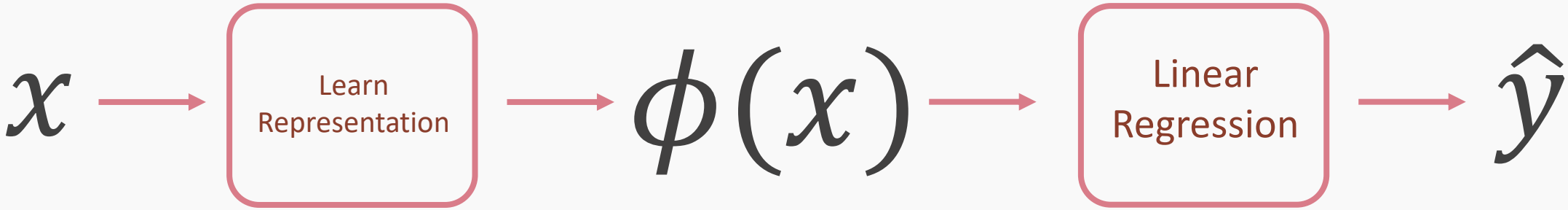


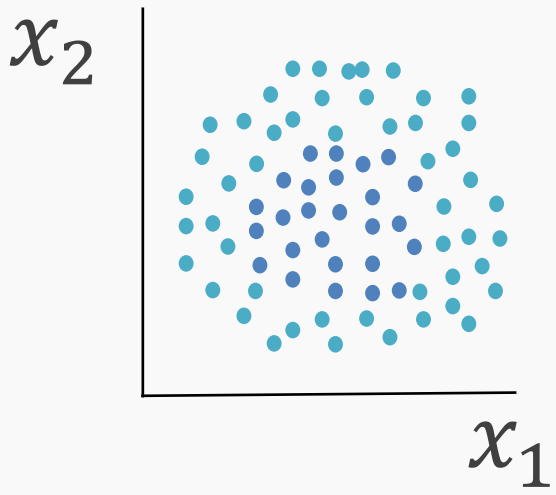
$$\phi(x) = x^2$$



Representational Learning

Representation Matters

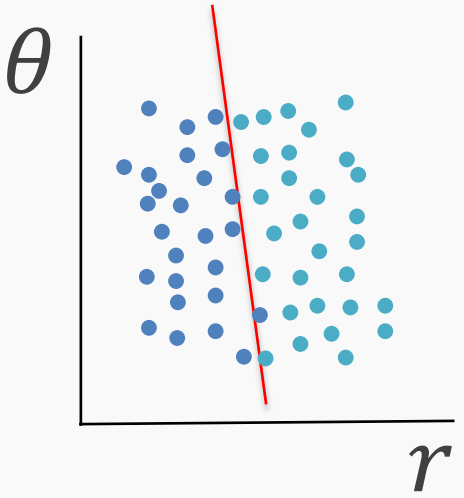




$$\phi(x)$$

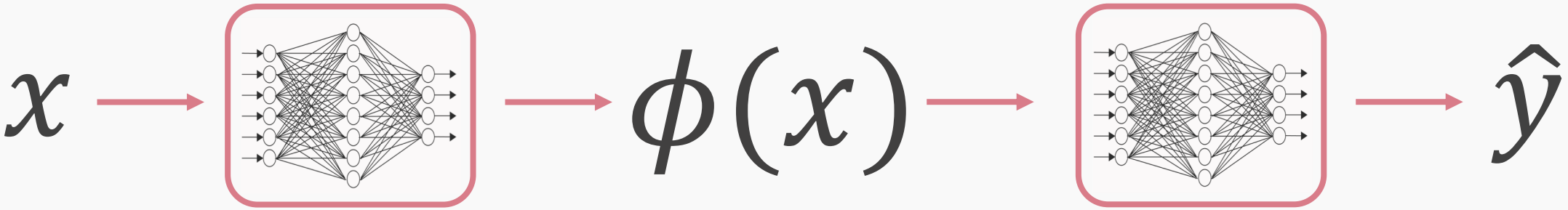
$$r^2 = x_1^2 + x_2^2$$

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



Representational Learning

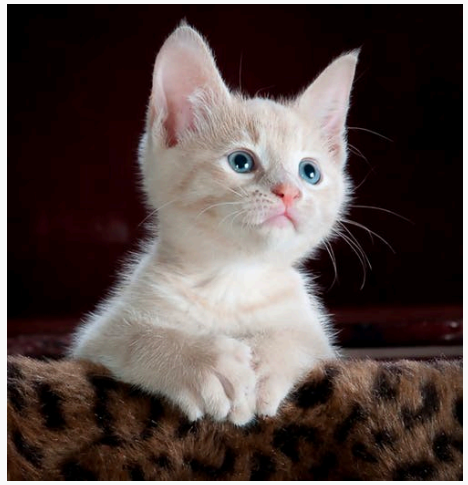
Representation Matters

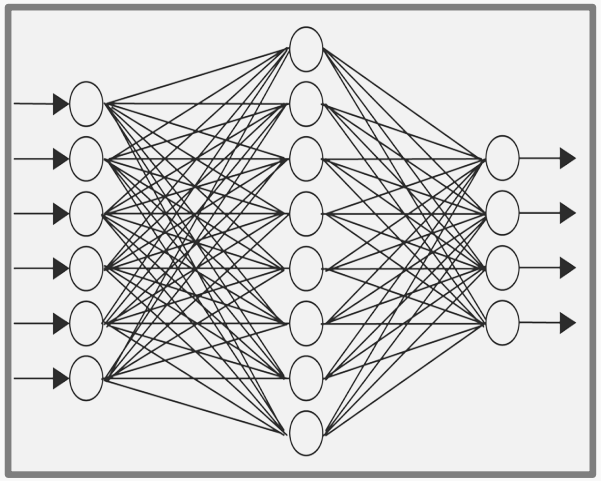


Representational Learning: **Supervised Learning**

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

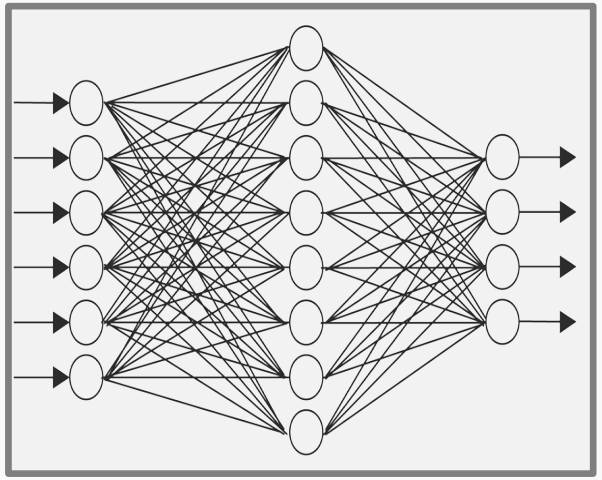
X





Feature Discovery Network

Features
$\phi(x)$



Classification Network

Y

{Cat,Dog}

Representational Learning: **Self-supervised Learning**

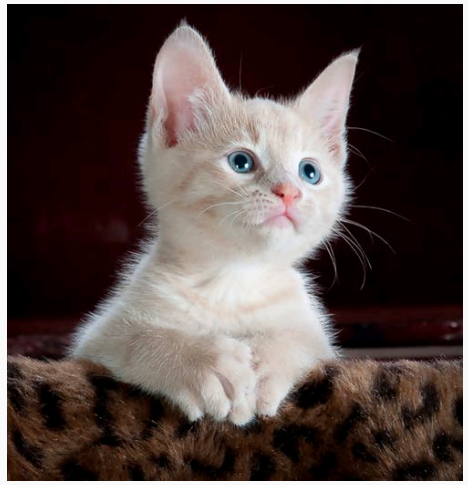
We train the two networks by minimizing th **entropy loss**)

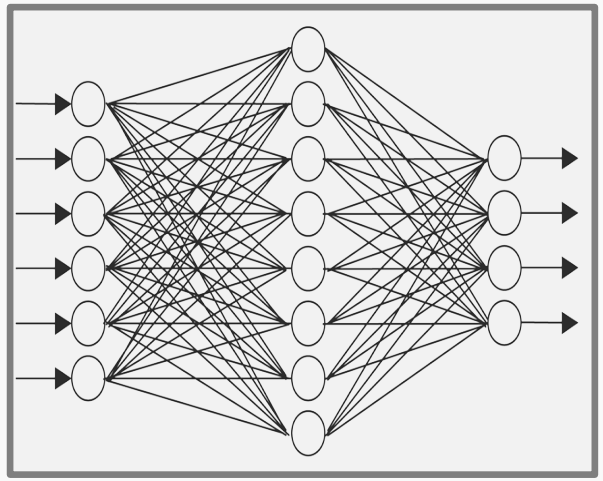
No labels

Specialized features for Task

X

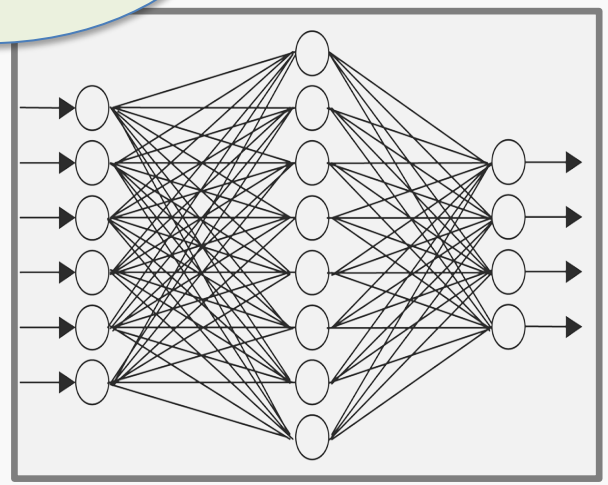
Y





Feature Discovery Network

Features $\phi(x)$



Classification Network

{Cat,Dog}

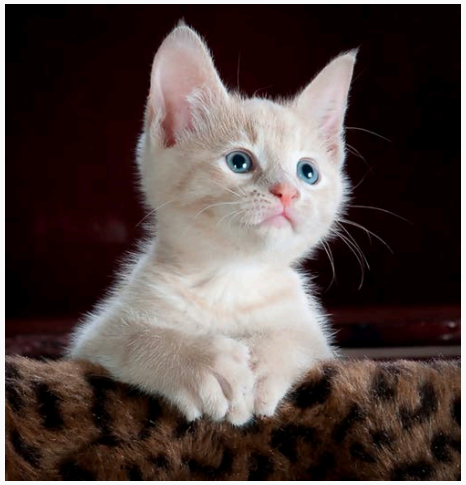
Representational Learning: **Self-supervised Learning**

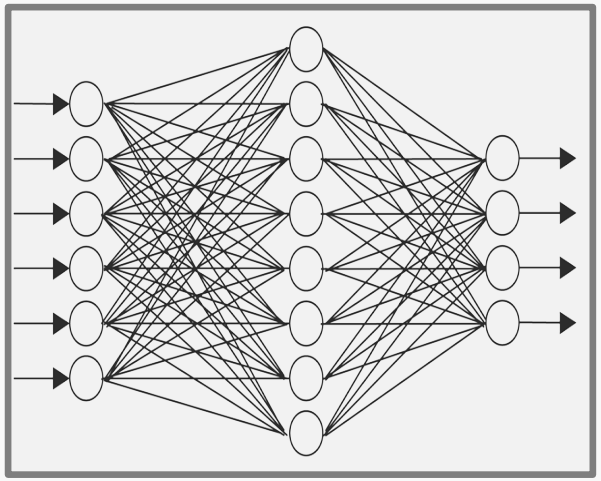
We train the two networks by minimizing the **reconstruction** loss function:

$$\mathcal{L} = \sum(x_i - \hat{x}_i)^2$$

AUTOENCODER

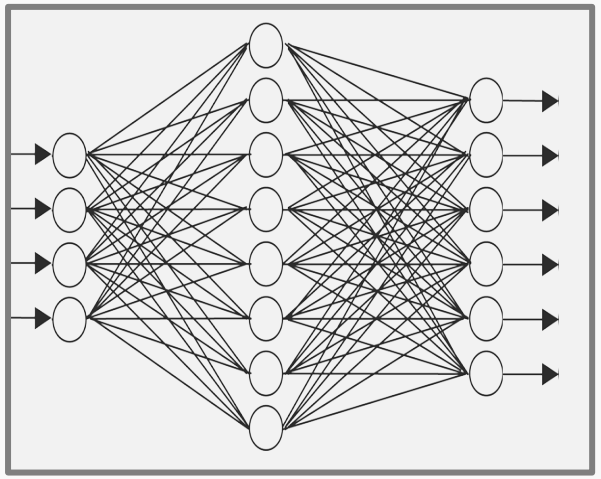
X

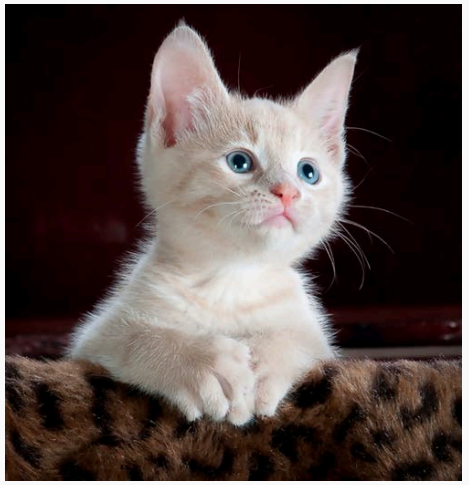




Latent Space Features z

$\phi(x)$





$\hat{X}$

Feature Extractor Network — ENCODER

Second Network — DECODER

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

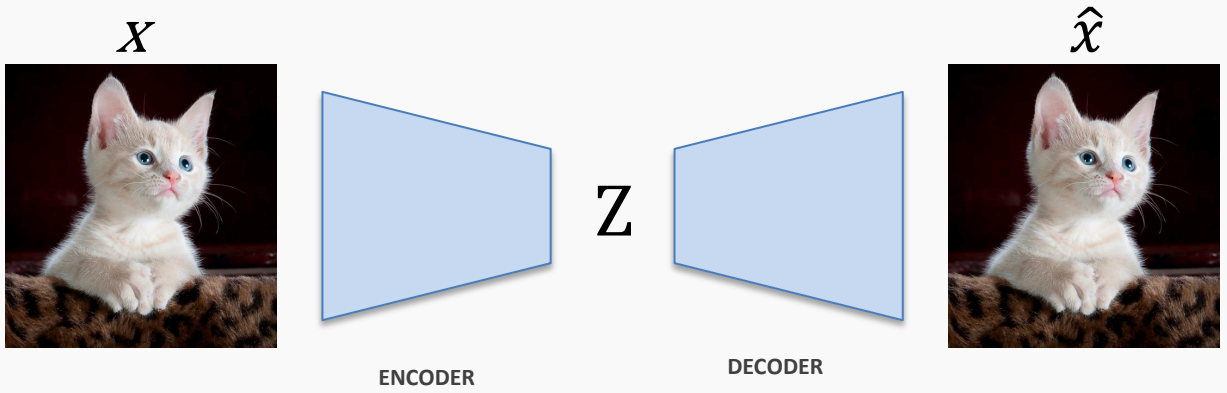
Your 36 characters message is compressed to 11 characters

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.



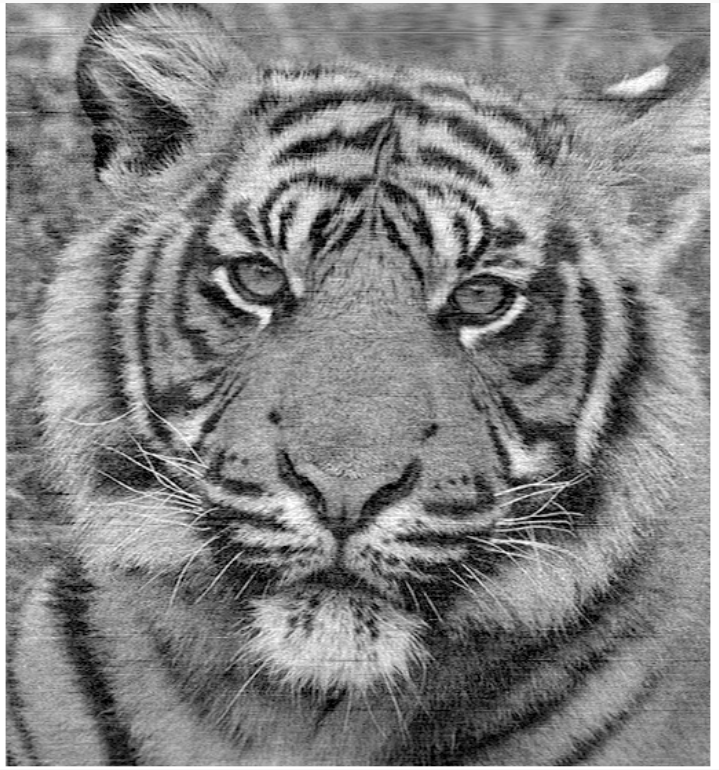
$$\mathcal{L} = \sum(x_i - \hat{x}_i)^2$$

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

Image taken from A. Glassner, Deep Learning, Vol. 2: From Basics to Practice

MP3 and JPG Image Compression(cont)

original MP3 JPG

Image taken from A. Glassner, Deep Learning, Vol. 2: From Basics to Practice

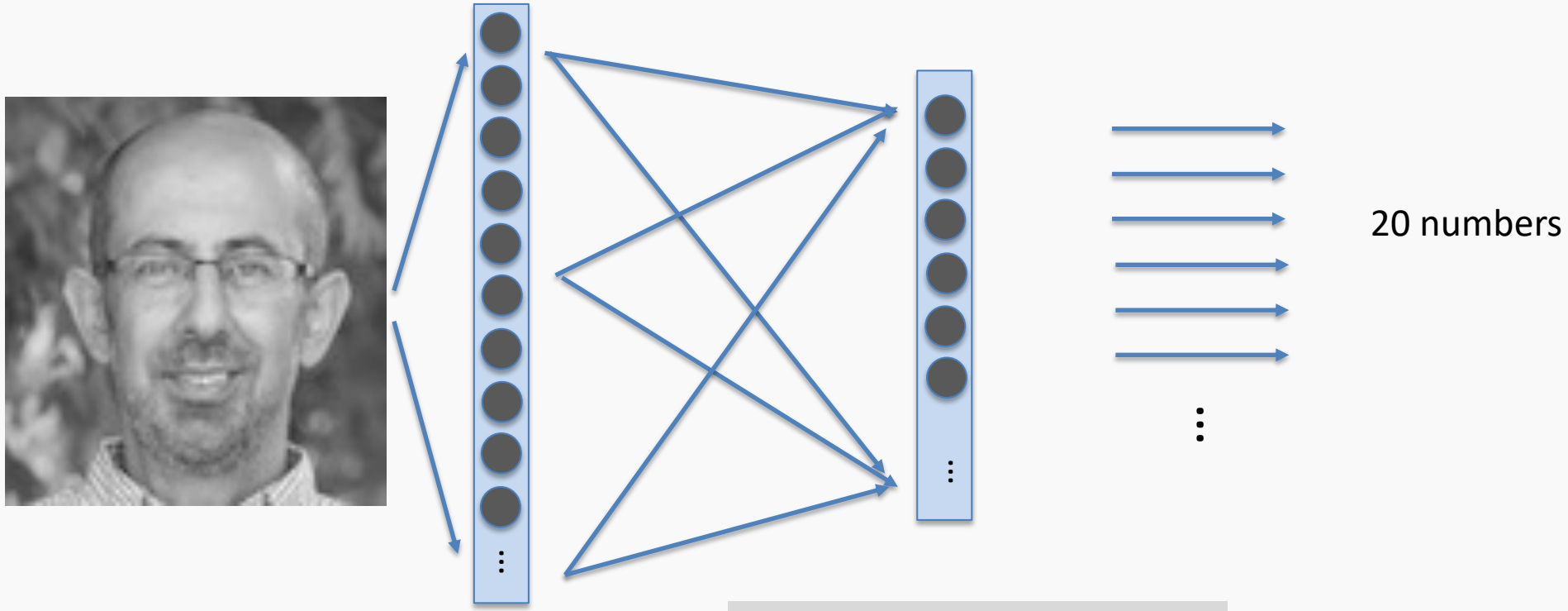
The simplest autoencoder

Encode with a simple fully connected network (FCN)

Image 100x100

Flatten Image 10,000

FCN with 20 neurons



20 numbers

Parameters: 10K x 20 + 20

The simplest autoencoder

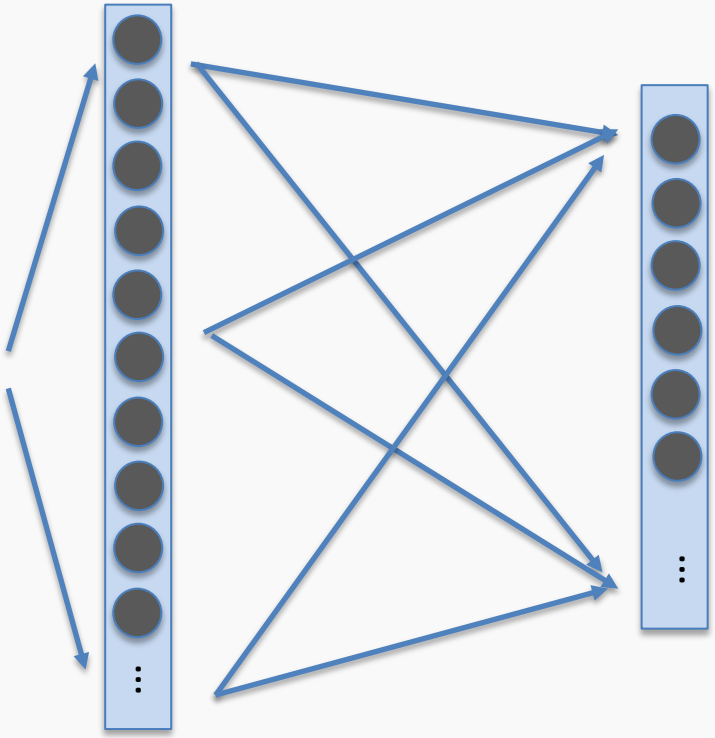
Encode and decode together after training

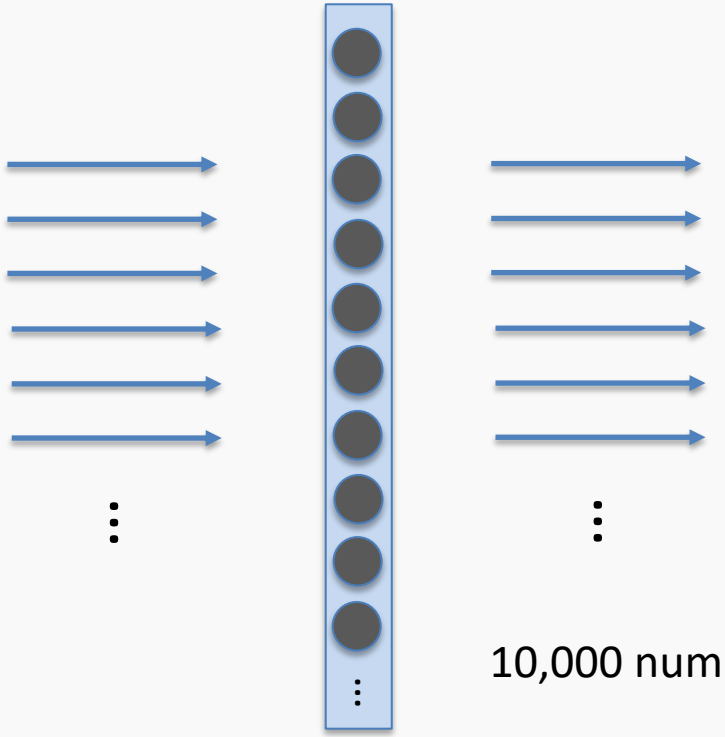
Image 100x100

Flatten Image 10,000

FCN with 20 neurons

FCN with 10,000 neurons





.reshape(100,100)

10,000 numbers

Parameters: 10K x 20 + 20

Parameters: 10K x 20 + 10K

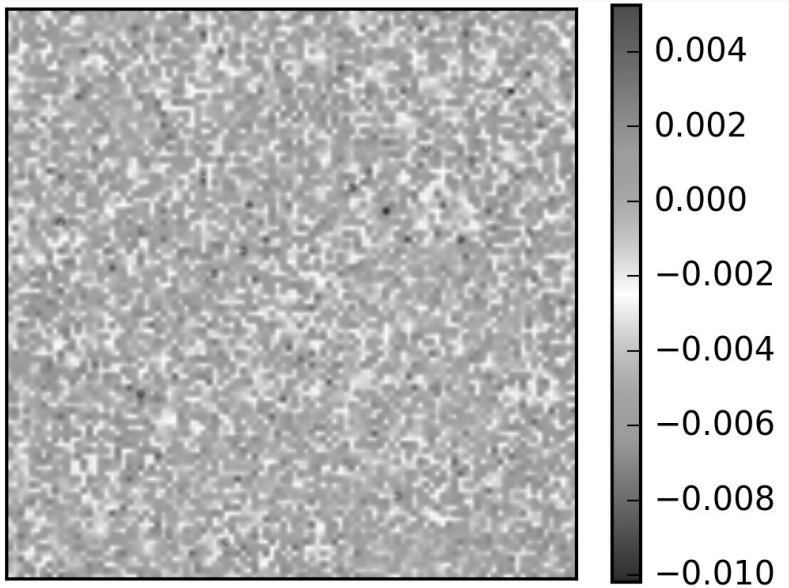
Autoencoders in action

Comparing the input and output pixel by pixel.

Input Image 100x100

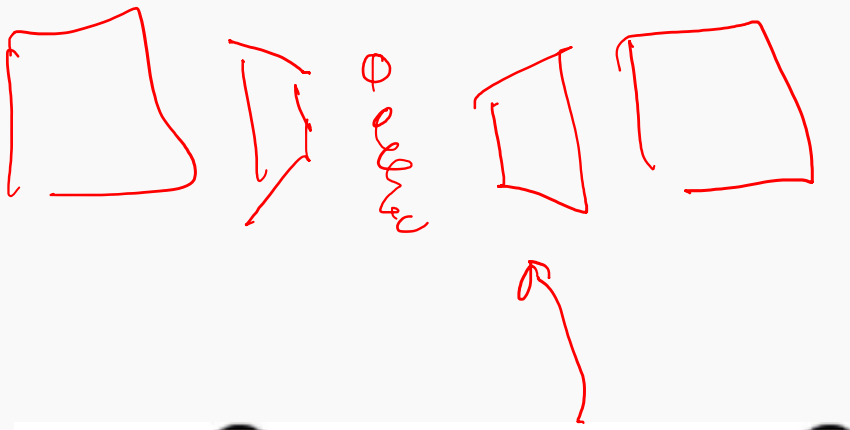
Output Image 100x100

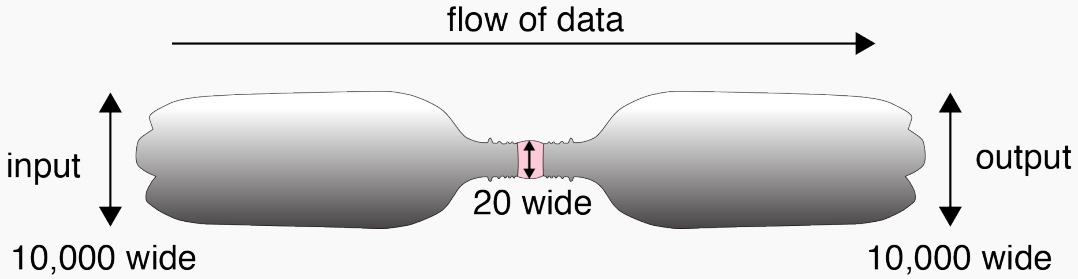
Residuals 100x100

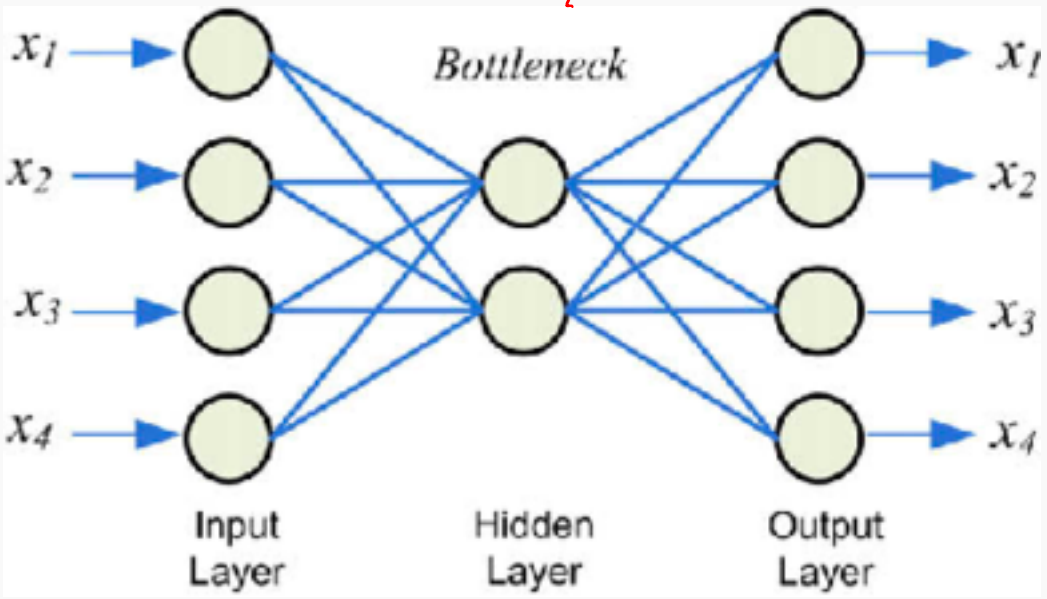


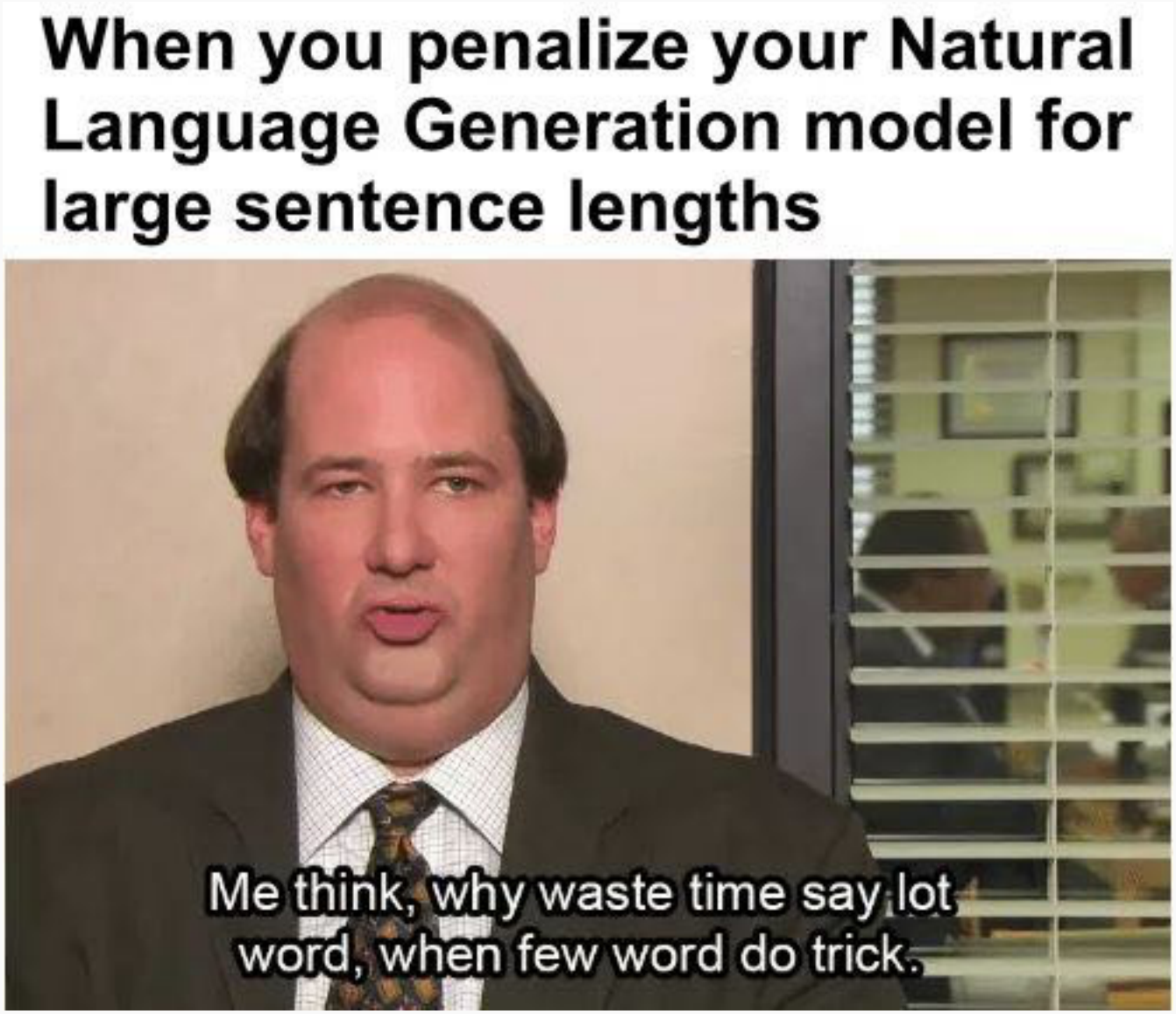
Bottleneck

- 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
Me think, why waste time say lot word, when few word do trick.

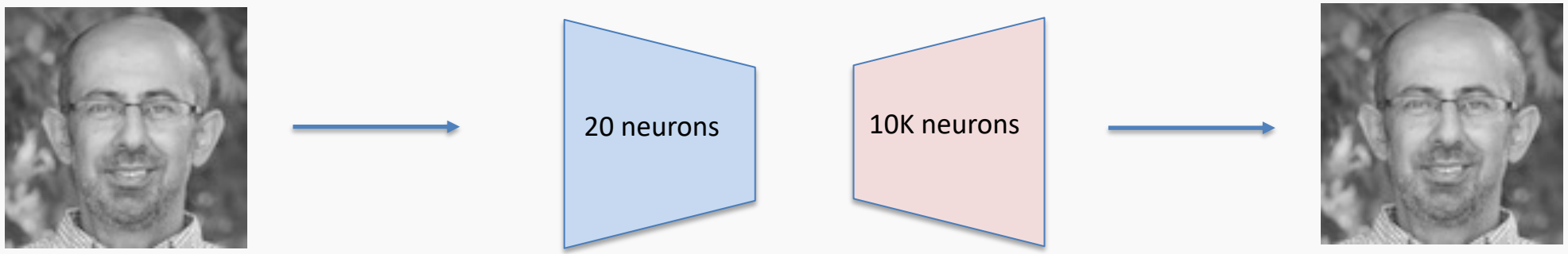
Latent variables and latent layer

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

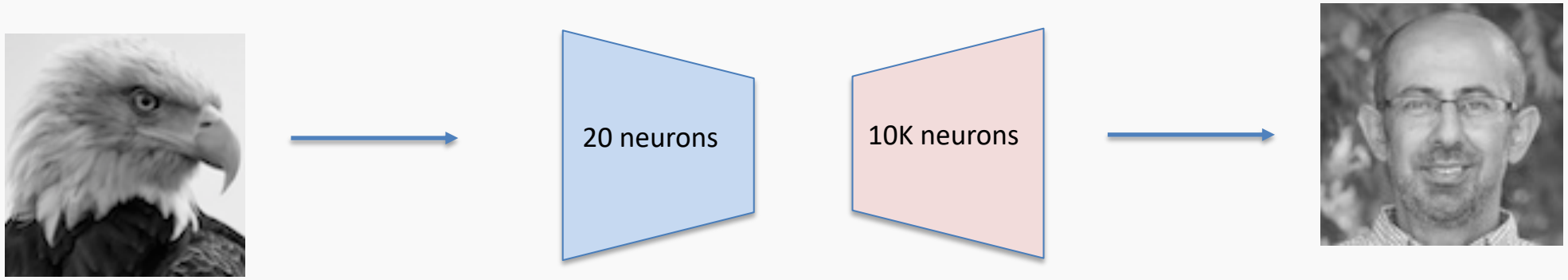
It sort-of is **supervised** learning because we give the system explicit goal data (the output should be the same as the input), and it sort-of isn't supervised learning because we don't have any manually determined labels or targets on the inputs.

Autoencoders in action (cont)

Passing ”Pavlos” to the trained autoencoder returns:

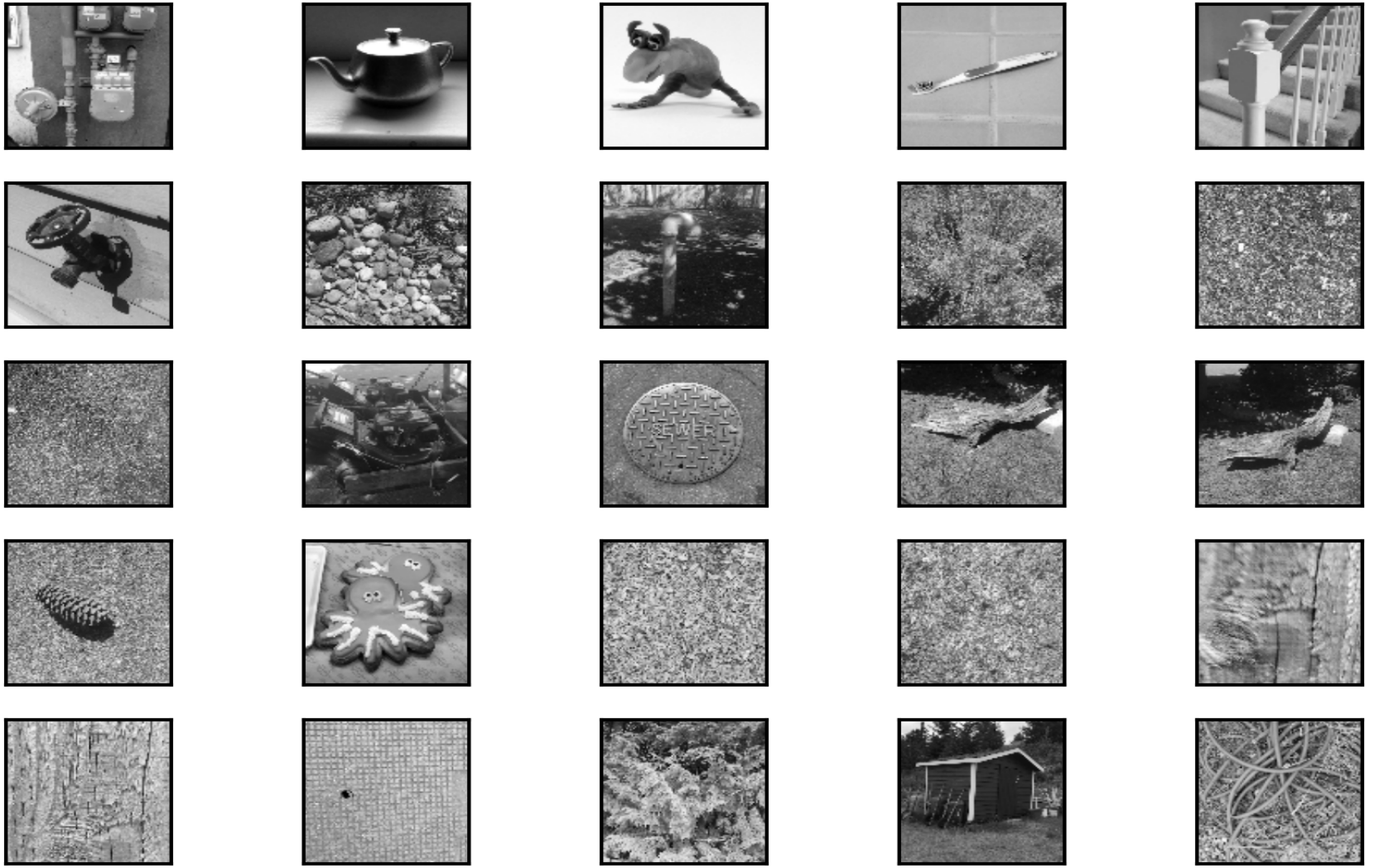


How about if we input the “Eagle”?



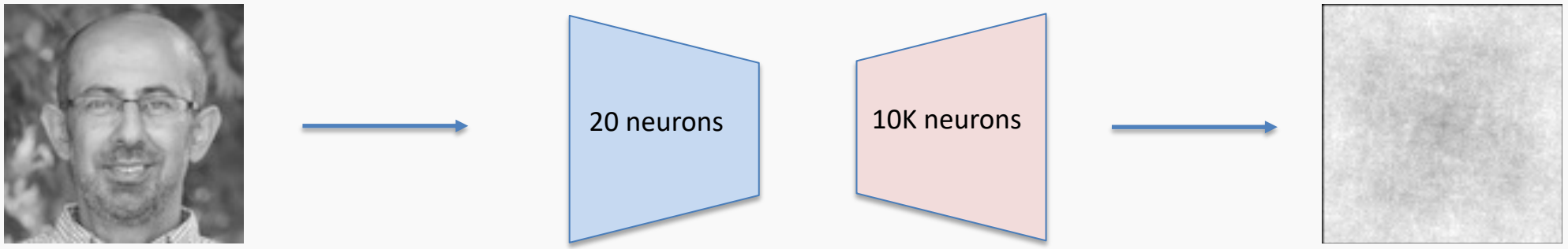
Autoencoders in action (cont)

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



Autoencoders in action (cont)

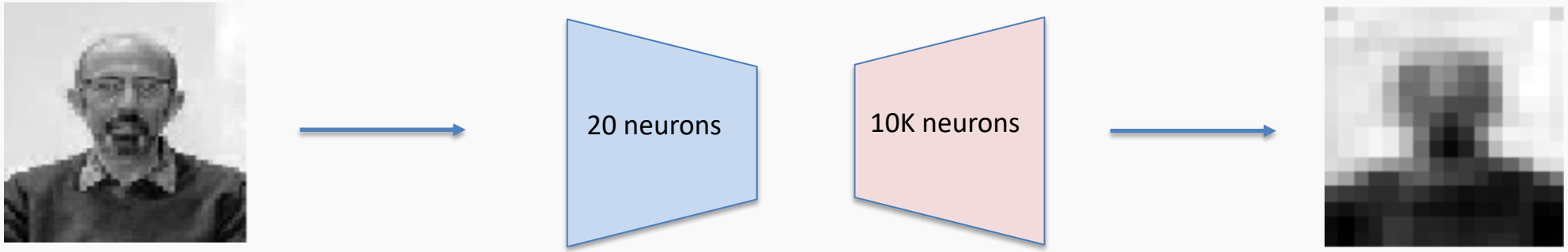
After training with those images, let's test how well it regularizes:



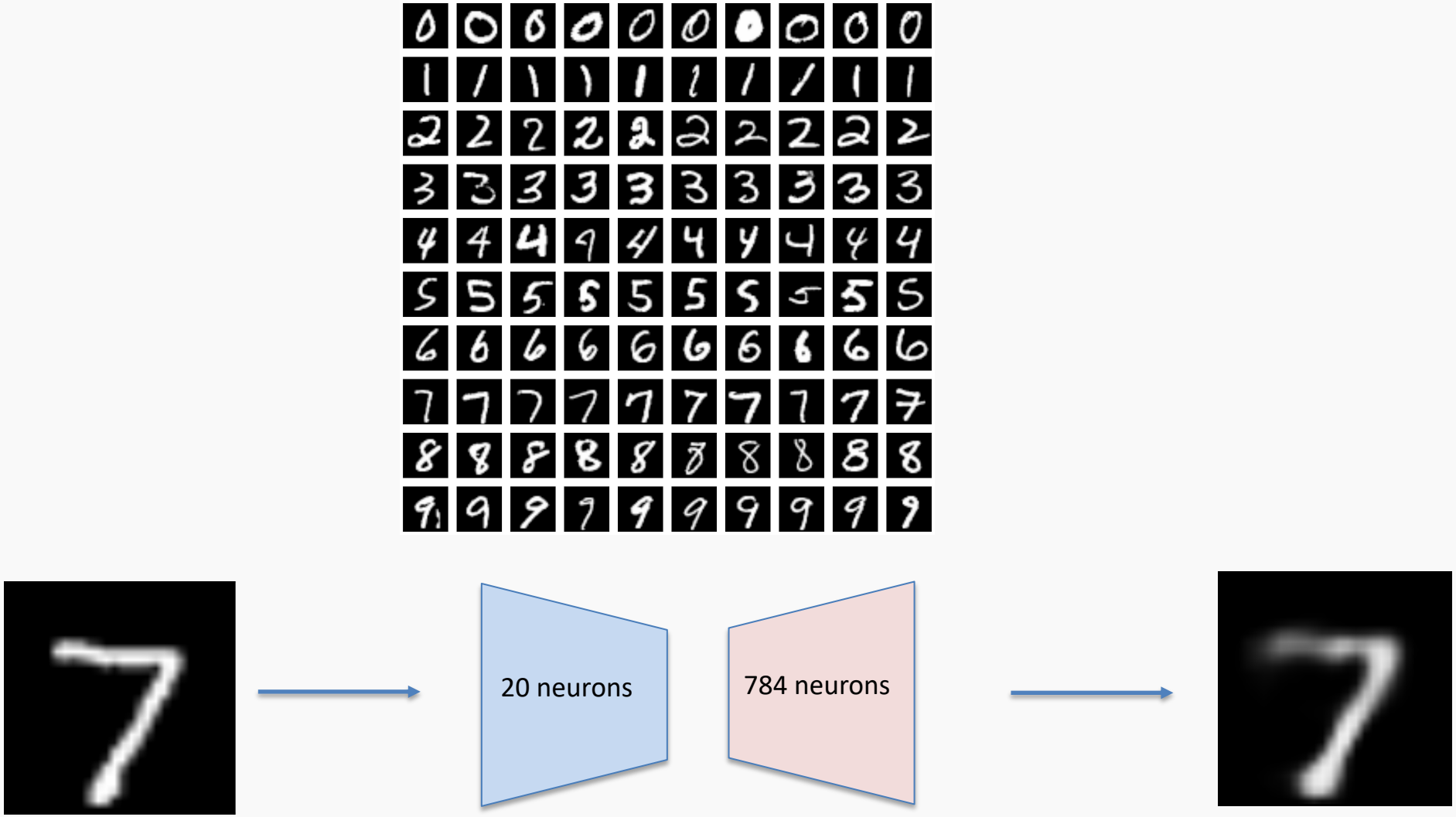
Network never seen anything like this, so it is no surprise that could not reconstruct this.

Autoencoders in action (cont)

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



A better autoencoder



20 latent variables

original

reconstructed

10 latent variables

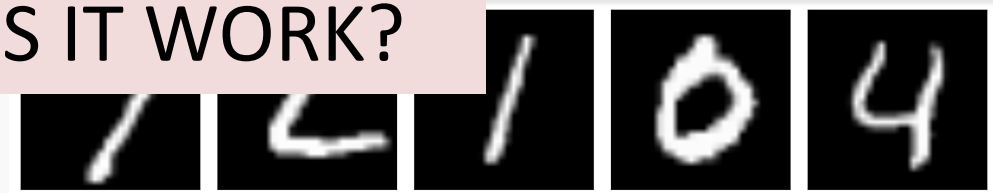
original

reconstructed

WHY DOES IT WORK?

2 latent variables

original

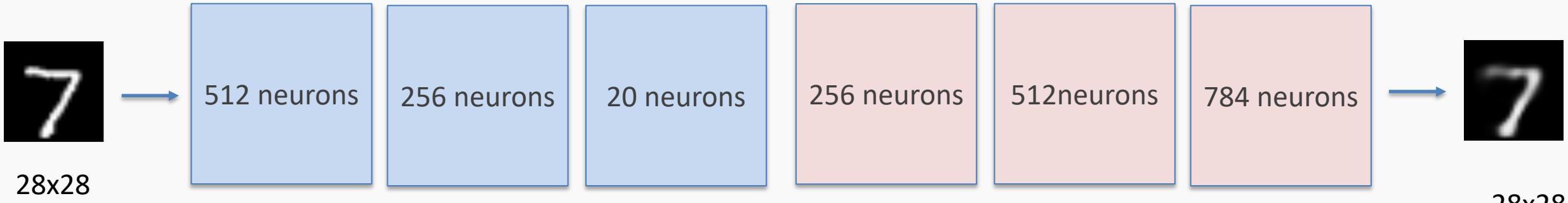


reconstructed

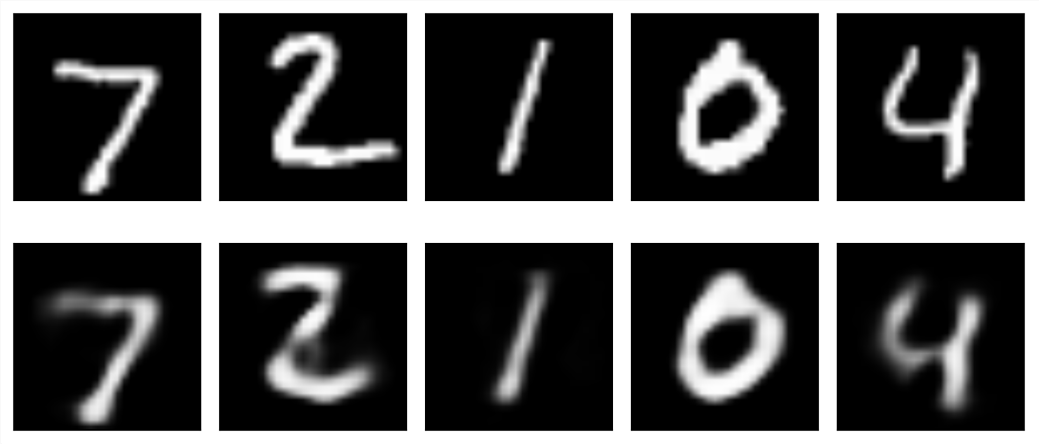
Image taken from A. Glassner, Deep Learning, Vol. 2: From Basics to Practice

Deeper

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



Z is 20 dimensional



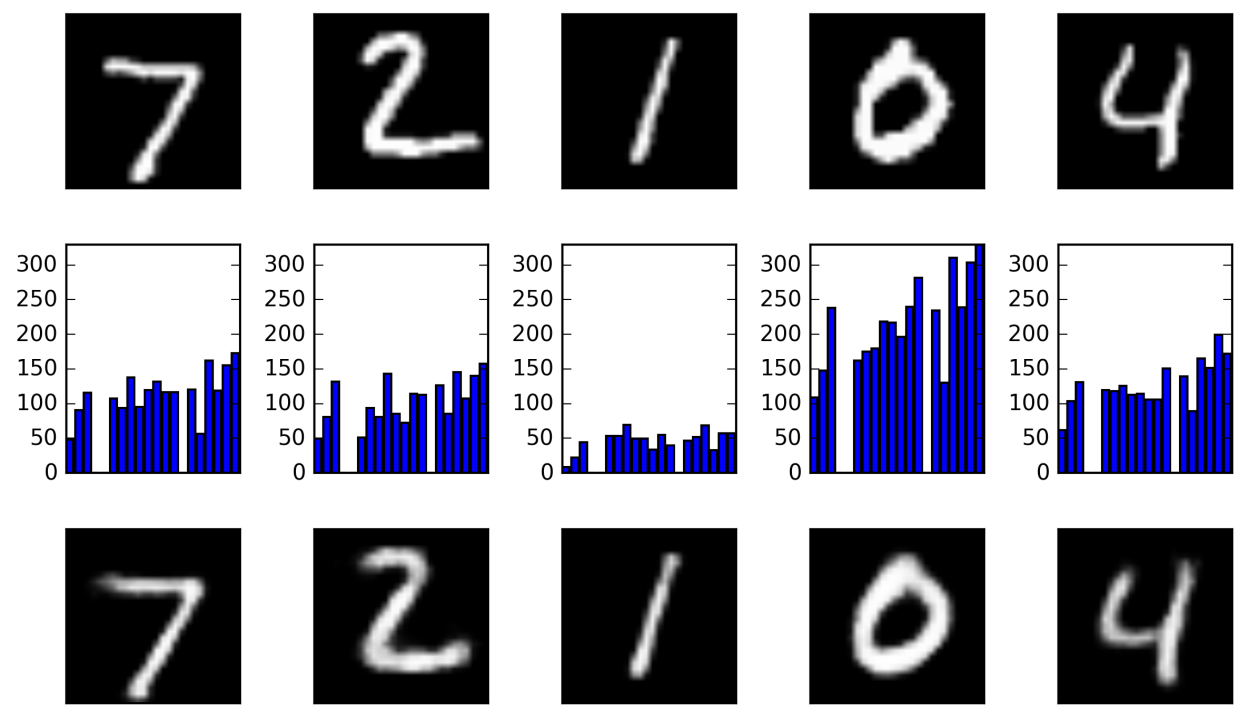
Shallow 20 latent variables

DEEPER IS BETTER


Reconstructing
Input Image

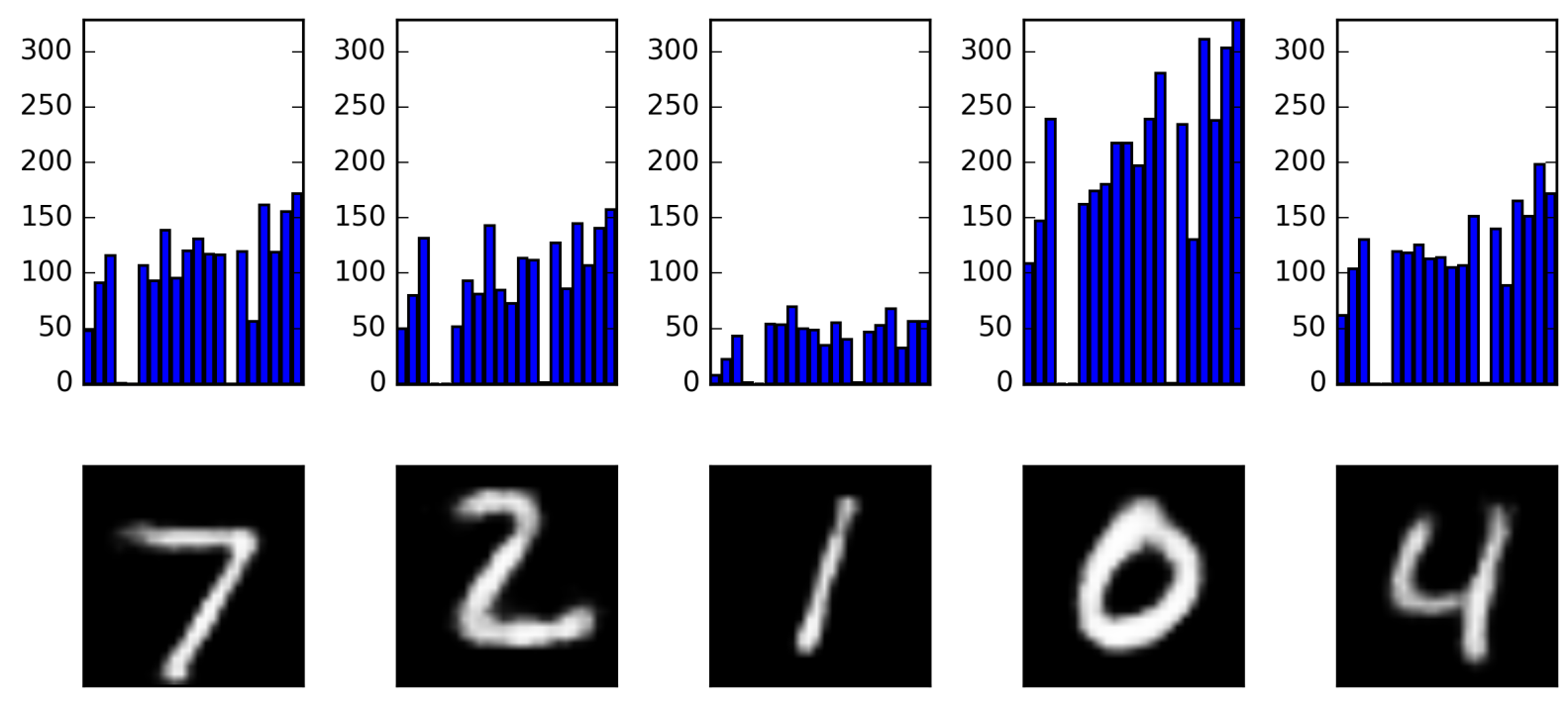
Exploring autoencoders

Explore the weights // slow PAUSE overlaid



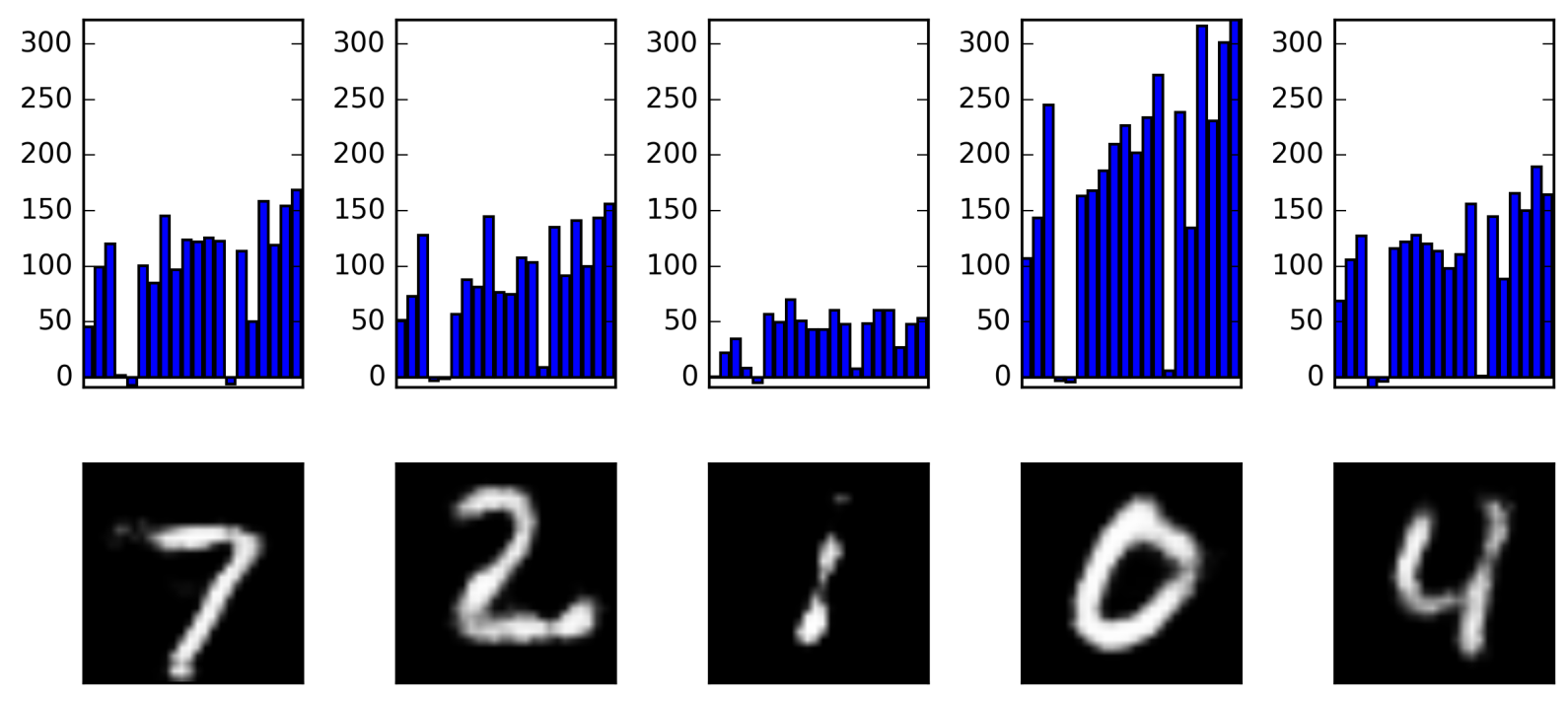
redo this.

Exploring autoencoders (cont)



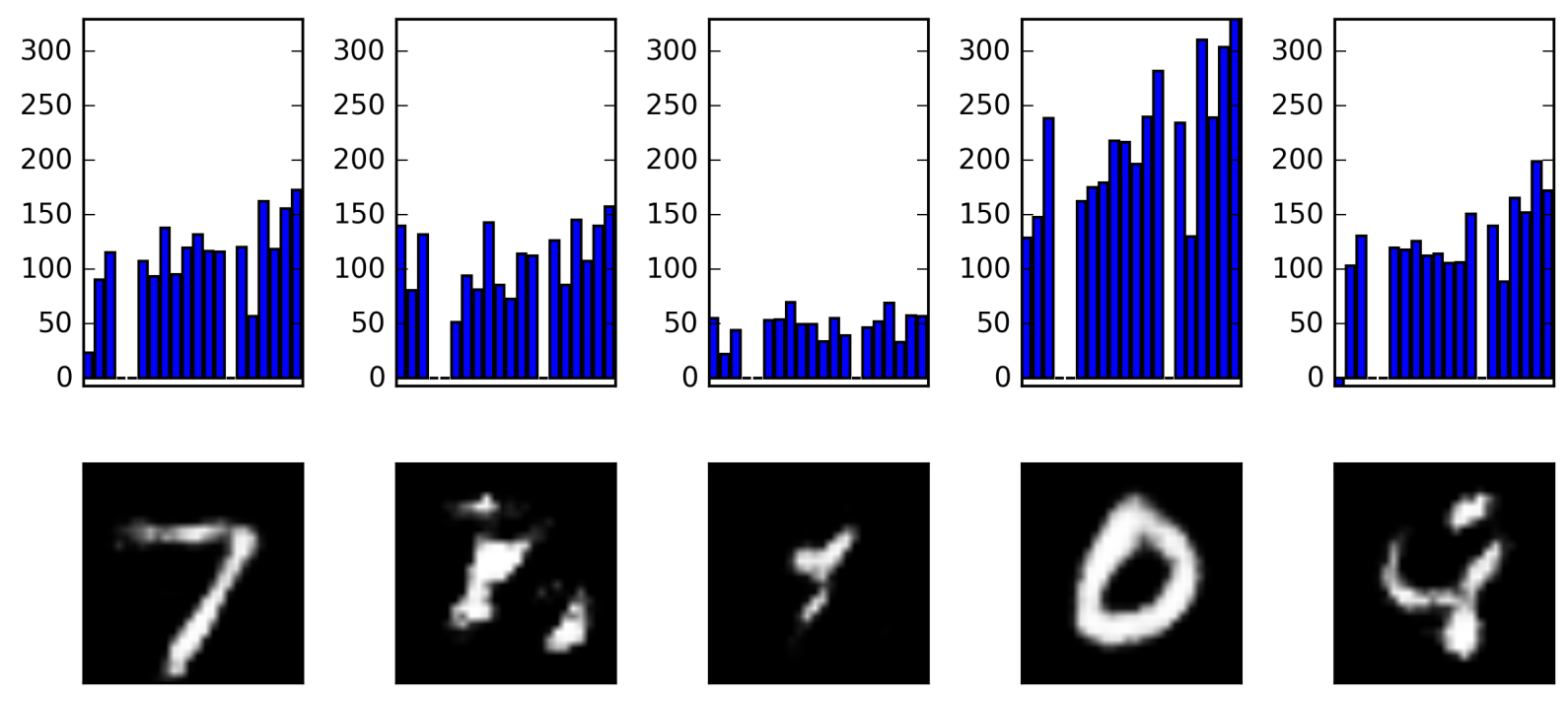
Change +/- 1 the latent variables

Exploring autoencoders (cont)



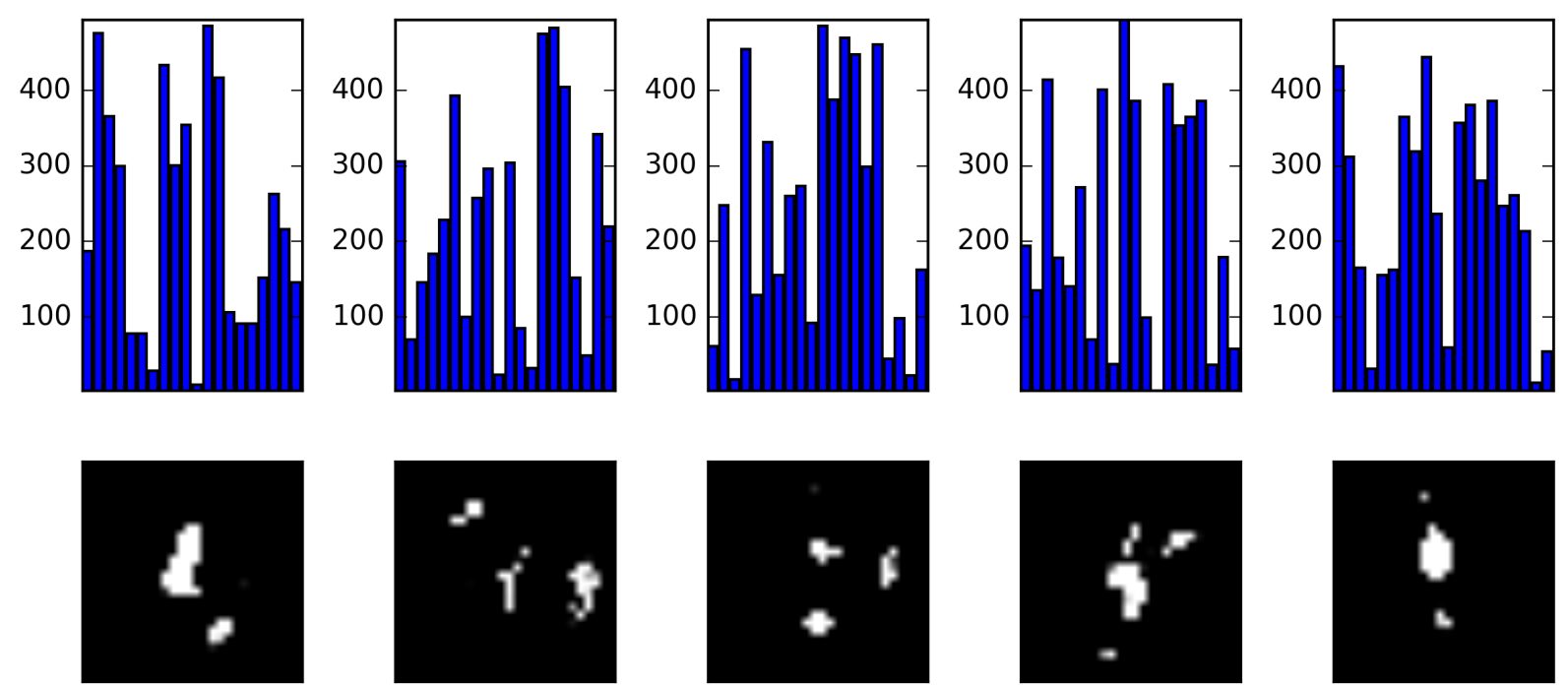
Change +/- 10 the latent variables

Exploring autoencoders (cont)



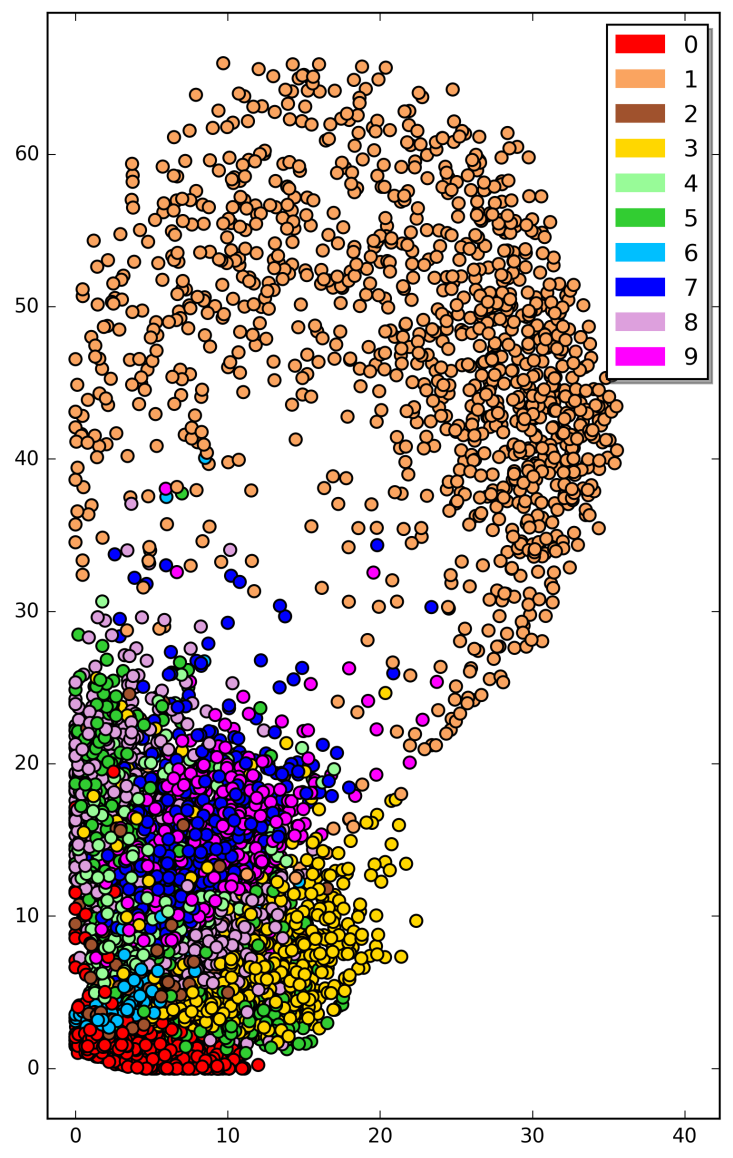
Change +/- 100 the first latent variables

Exploring autoencoders (cont)



Just noise

Parameter space of autoencoder



what is this

2 pca dimensionality

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.



Image taken from A. Glassner, Deep Learning, Vol. 2: From Basics to Practice

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)

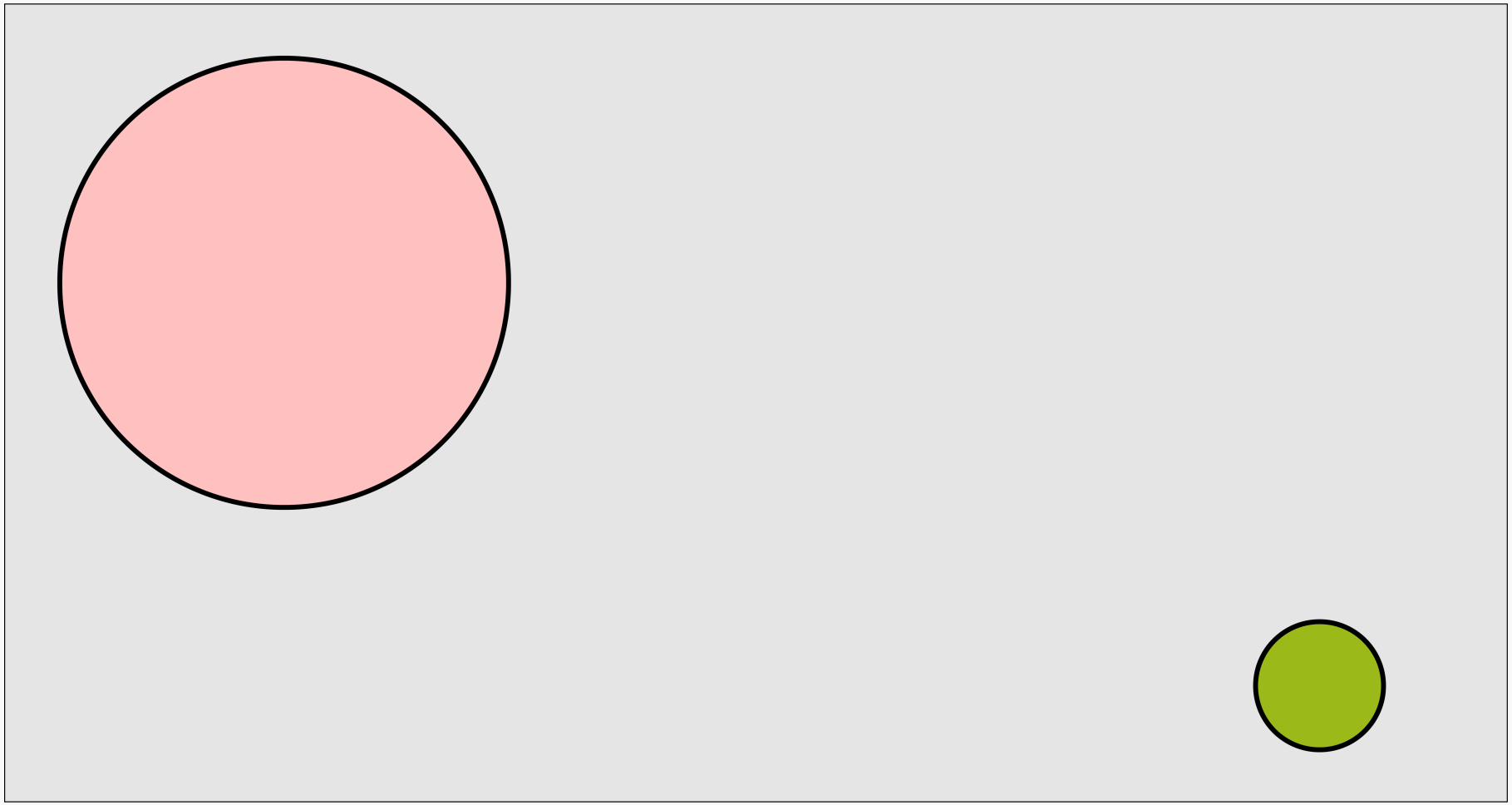


Image taken from A. Glassner, Deep Learning, Vol. 2: From Basics to Practice

Parametric or representation blending

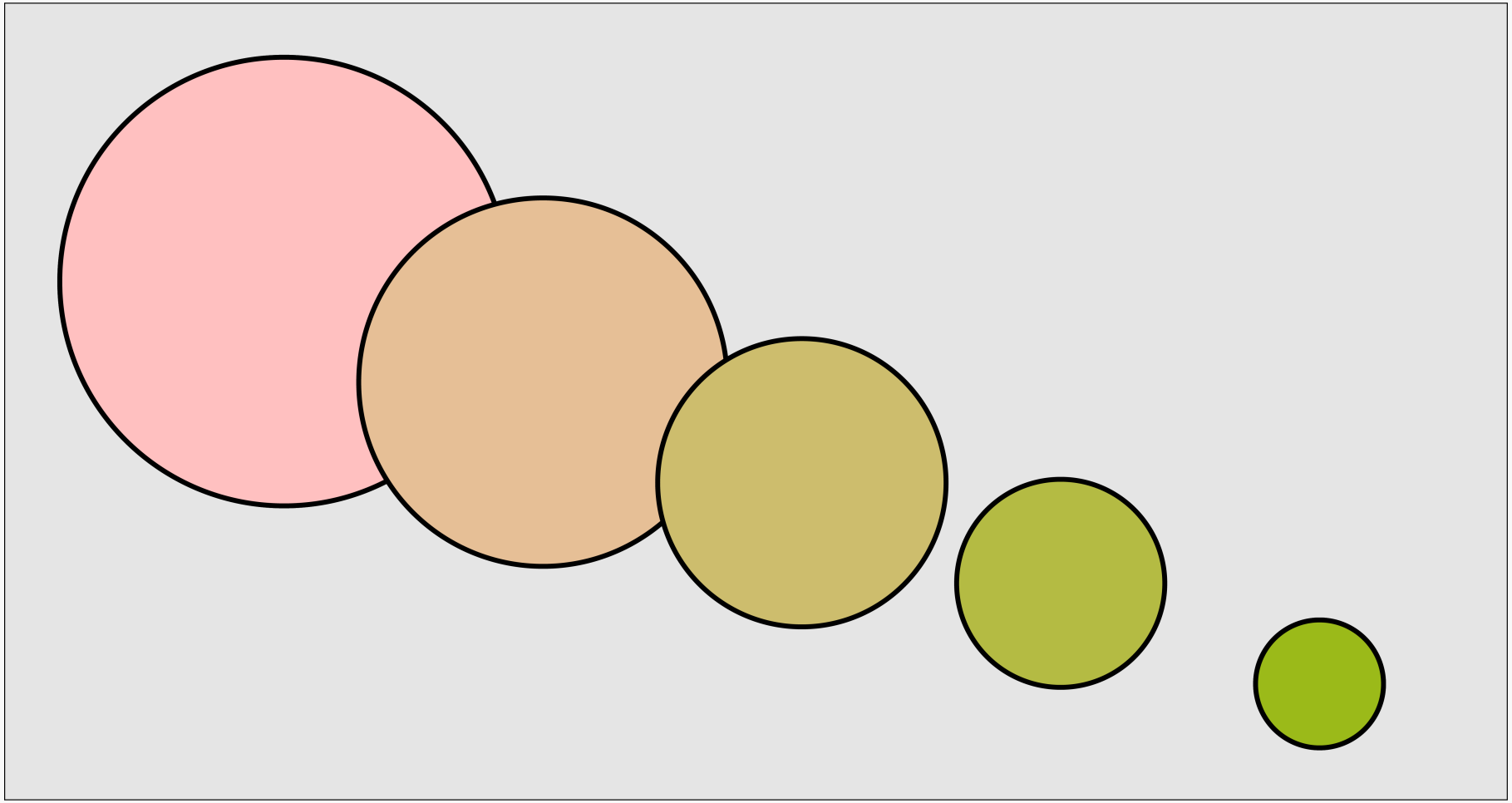


Image taken from A. Glassner, Deep Learning, Vol. 2: From Basics to Practice

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.

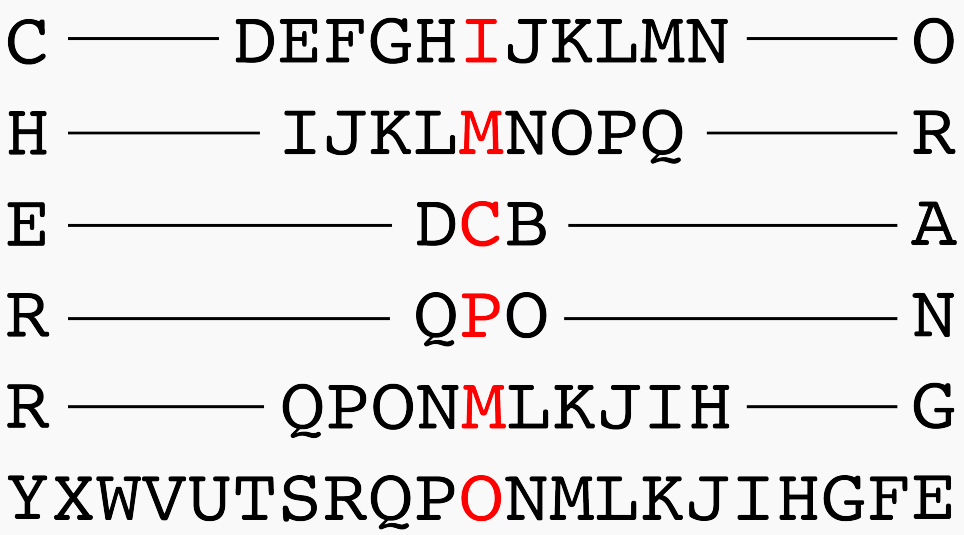
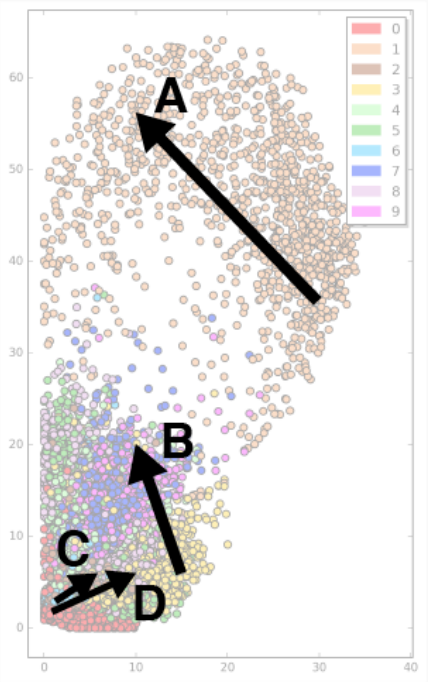
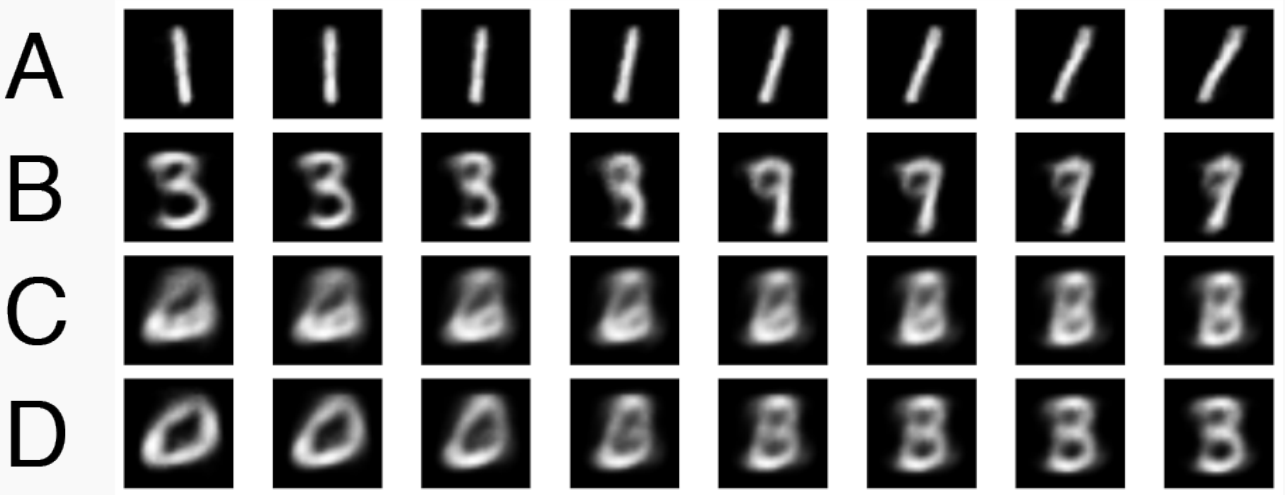


Image taken from A. Glassner, Deep Learning, Vol. 2: From Basics to Practice

Blending Latent Variables

Back to the example of MNIST

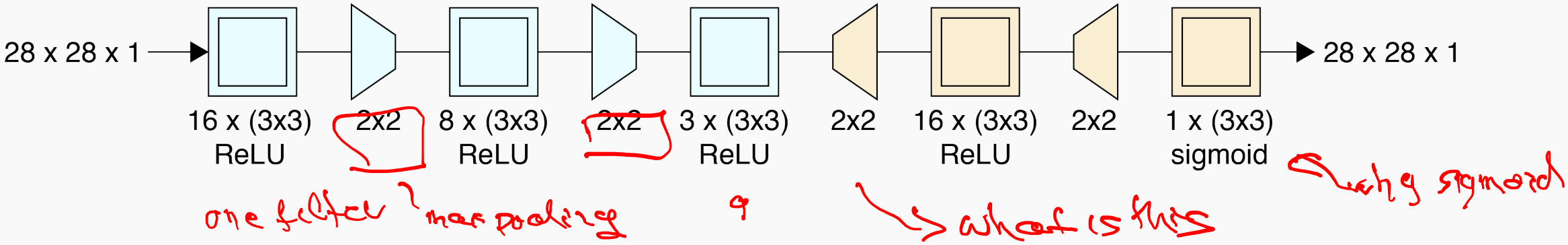


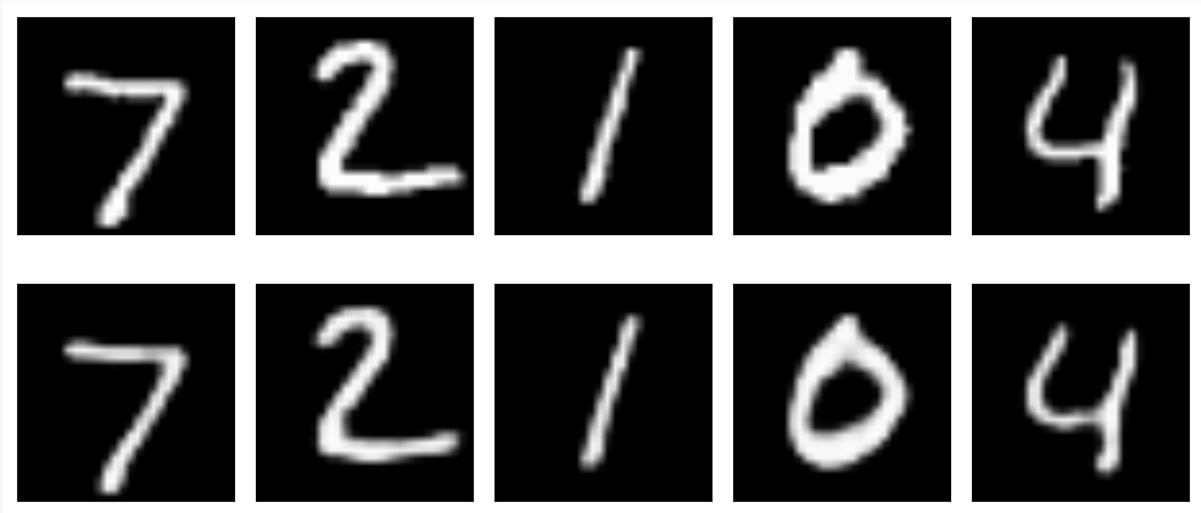


Applying to novel input

Convolutional Autoencoders

Do this step by step



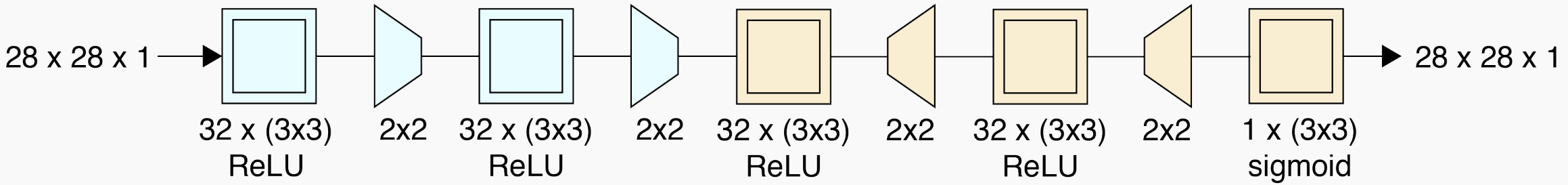


Denoising

→ need transition

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

Denoising (cont)



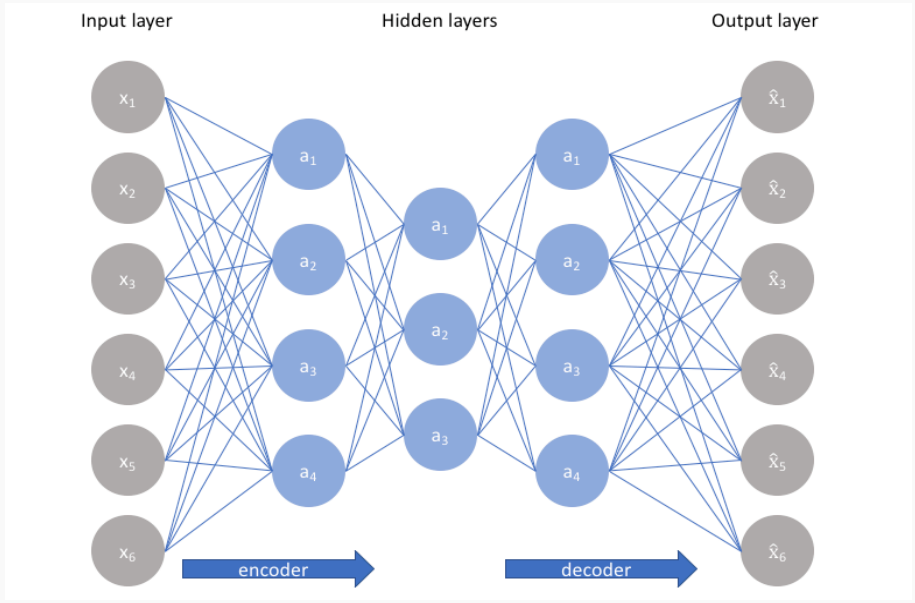
Note that we start with a clean image, add noise and train our networks to return a clean decoded image.

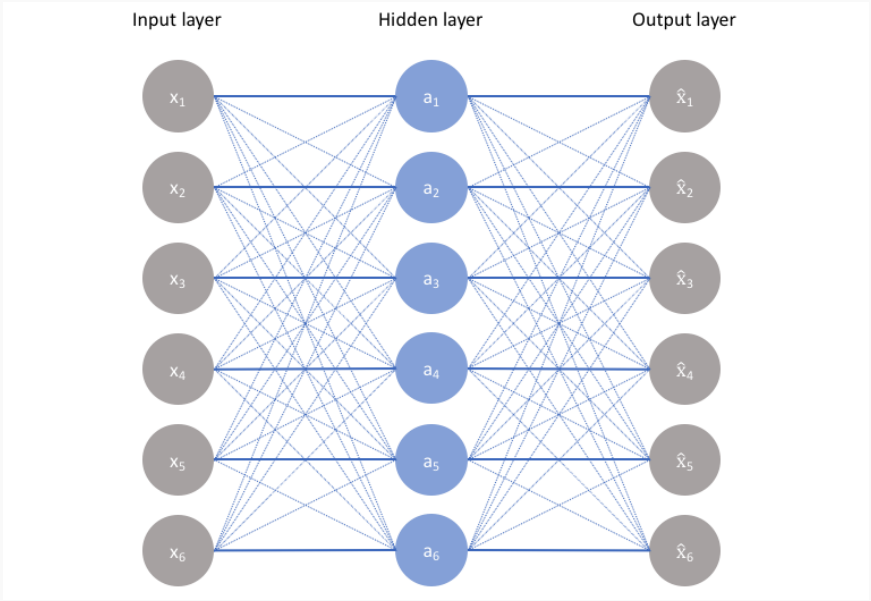
QUIZ?
what is
up scaling

Sparse Autoencoder

Quiz // click on which is overcomplete autoencoder

- 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**





Sparse Autoencoder (cont)

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

58

Regularized Autoencoders

- Sparse autoencoders
- Denoising autoencoders
- Autoencoders with dropout on z
- Contractive autoencoders

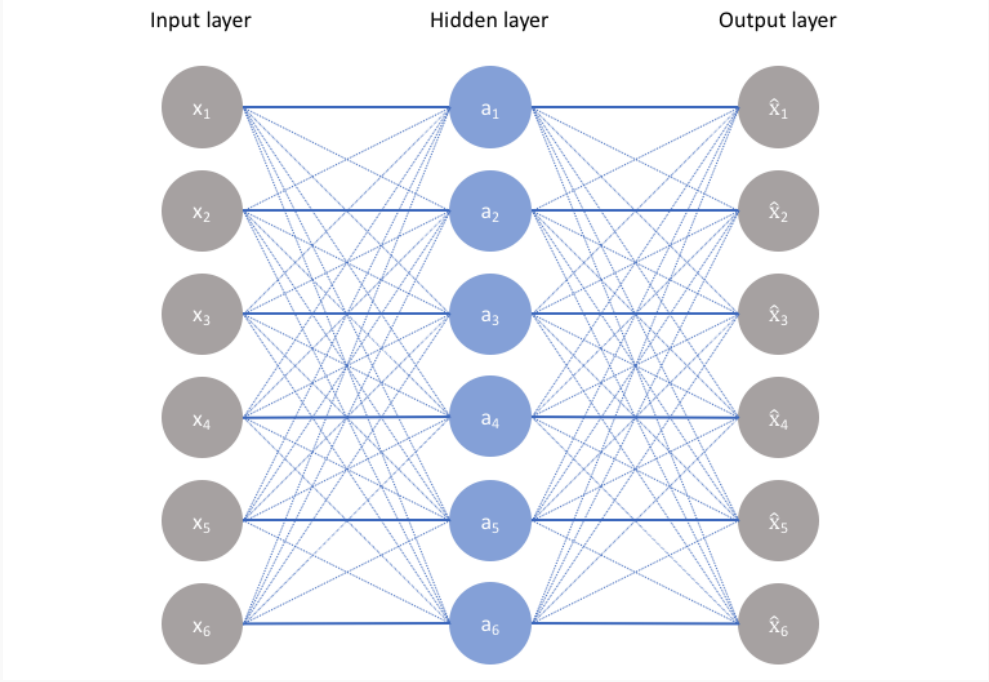
Overcomplete Autoencoders

z has greater dimension than *x*

Autoencoder may simply copy input to output without learning anything useful

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 overfit the training data



↳ infilling ⇒

Regularized Autoencoders (cont)

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).

$$\mathcal{L}\Big(x, g\big(f(x)\big)\Big) + \Omega(z)$$

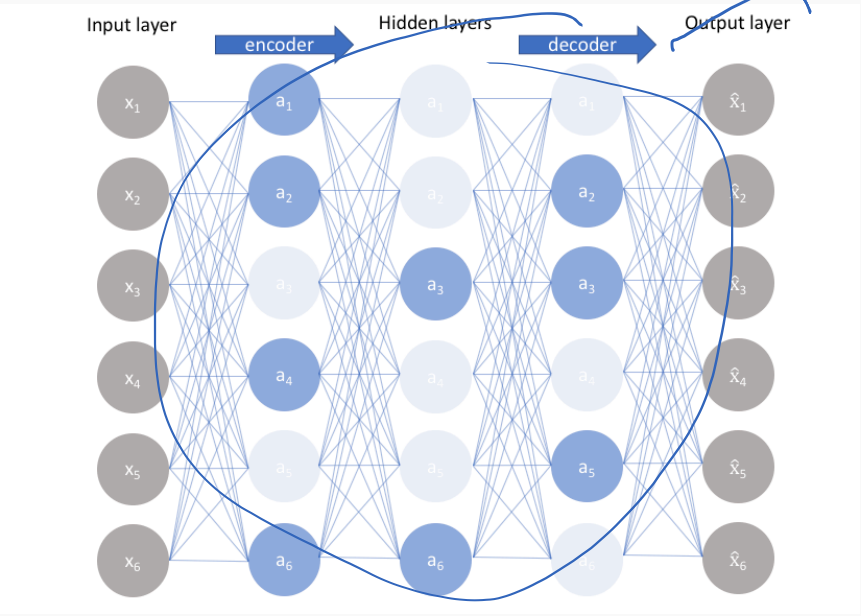
latent space

Regularization on output of encoder, not on network parameters

Sparse Autoencoders

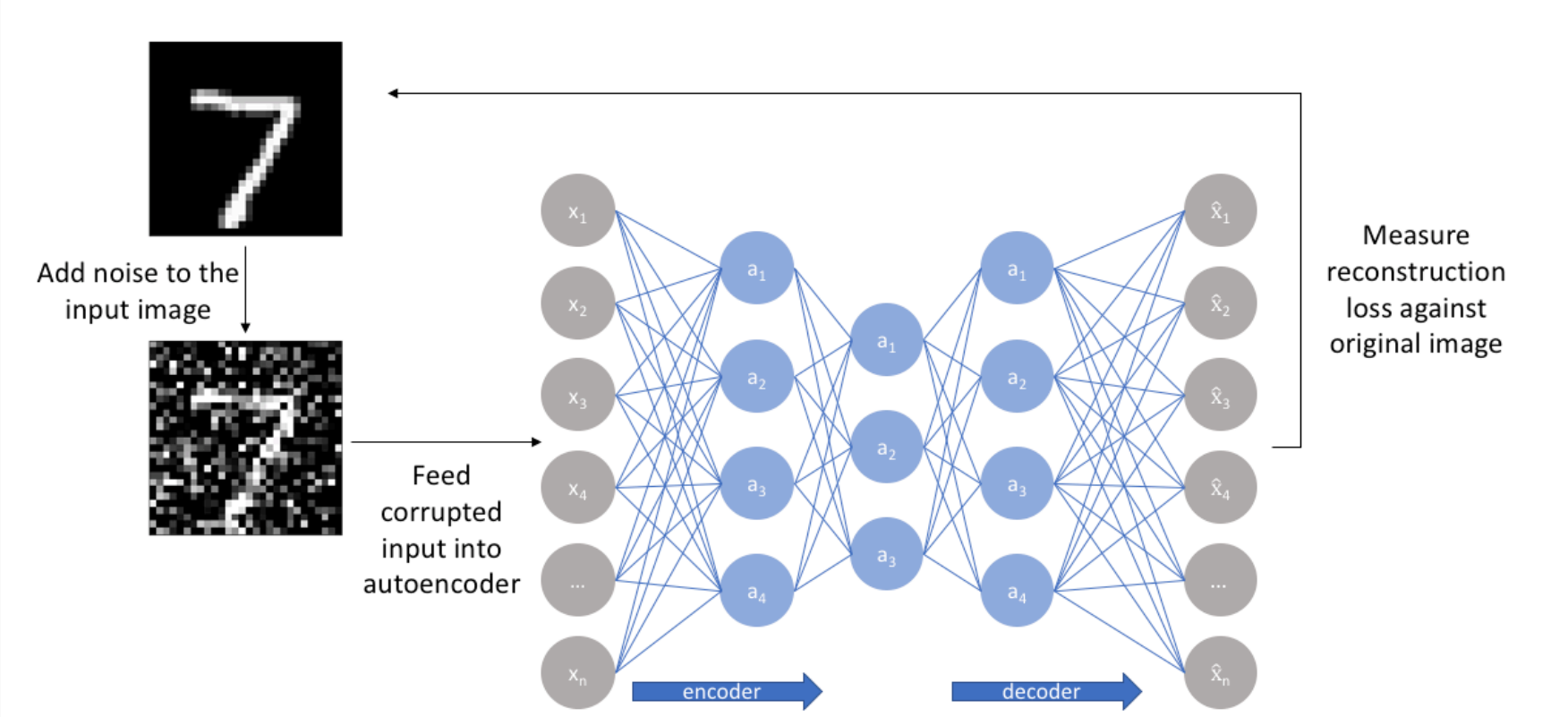
- We **allow our network to sensitize individual hidden layer nodes toward specific attributes of the input data**.
- A sparse autoencoder is selectively activate regions of the network depending on the input data.
- Limiting the network's capacity to memorize the input data without limiting the networks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

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



Denoising Autoencoders

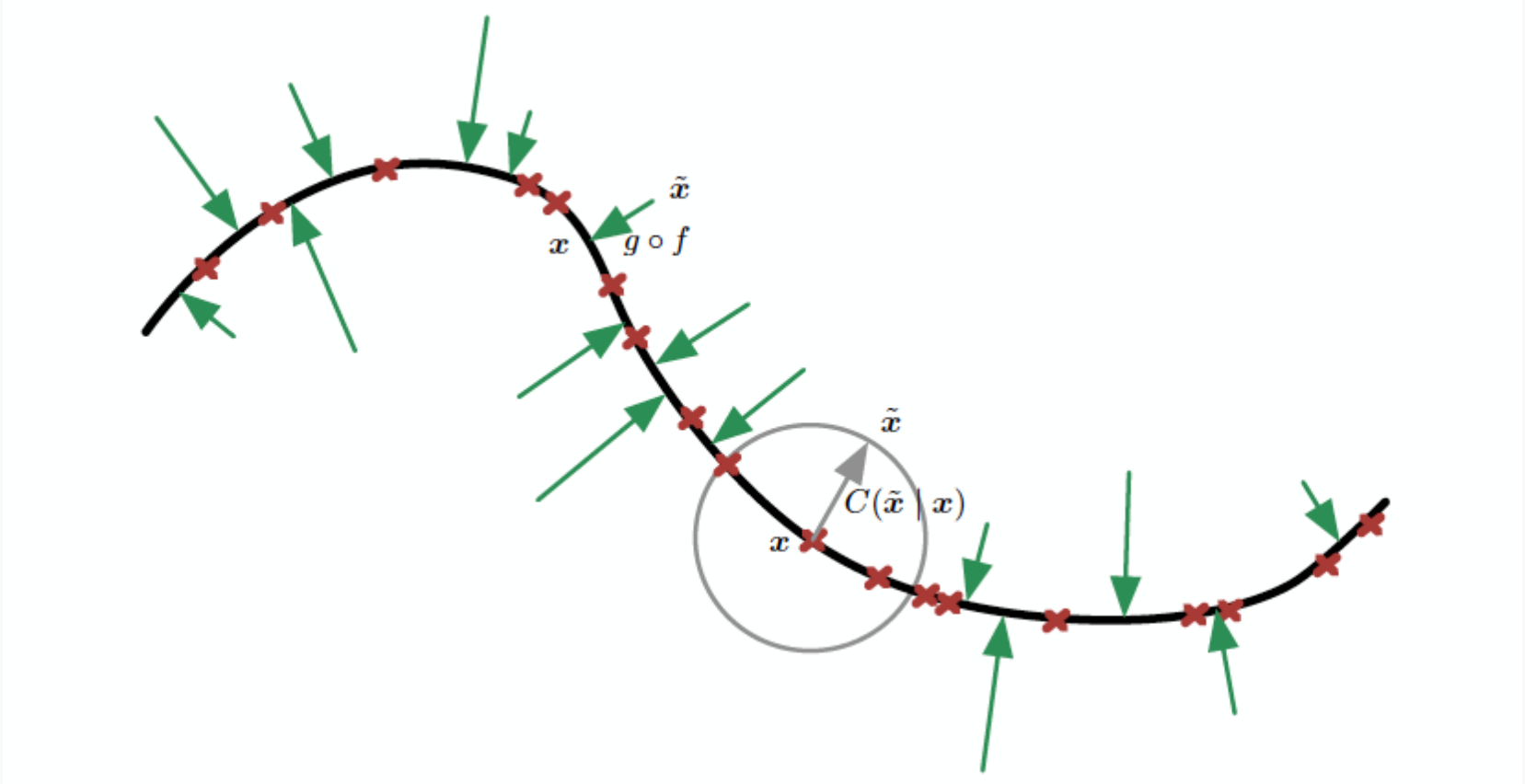
Trained with corrupted data points, but to reconstruct original data points



Denoising Autoencoders (cont)

UNDERSTAND THIS TWO

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



Contractive Autoencoders

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

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

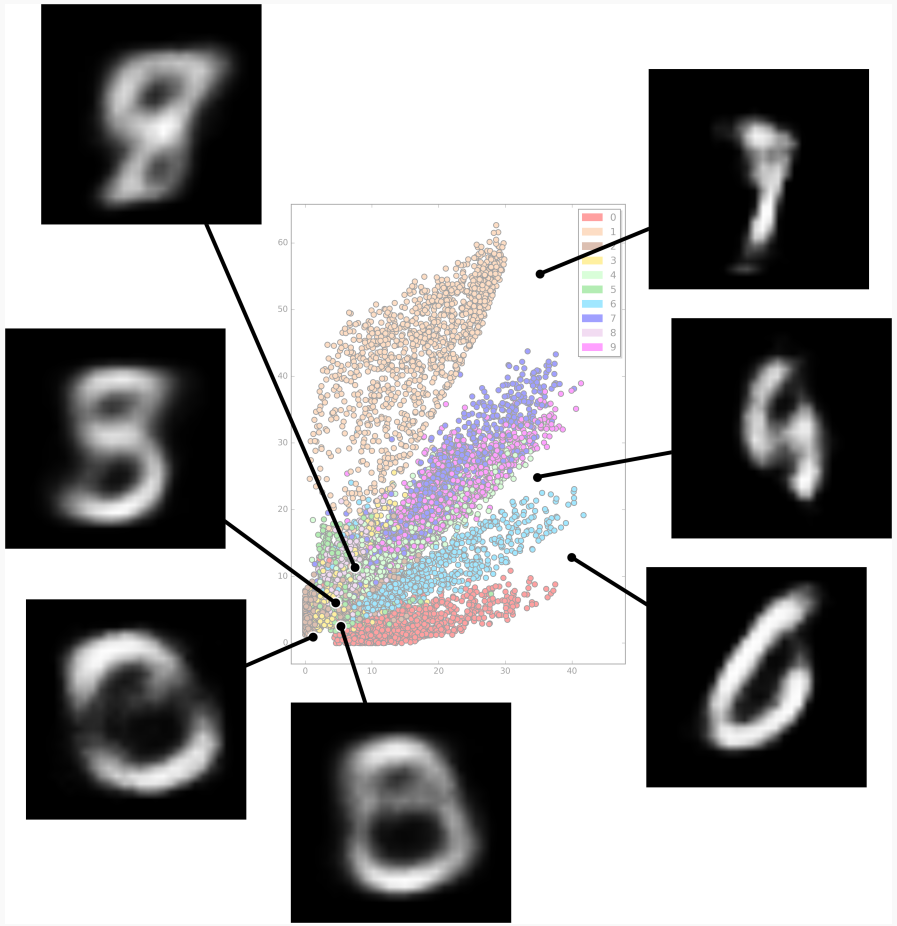
Question: How do we find how much the encoded space would change if the input changes?

Derivatives

$$L(x, g(f(x))) + \lambda \left\| \frac{\partial f(x)}{\partial x} \right\|_F^2$$

Problems with Autoencoders

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



More when we do variational auto-encoders