

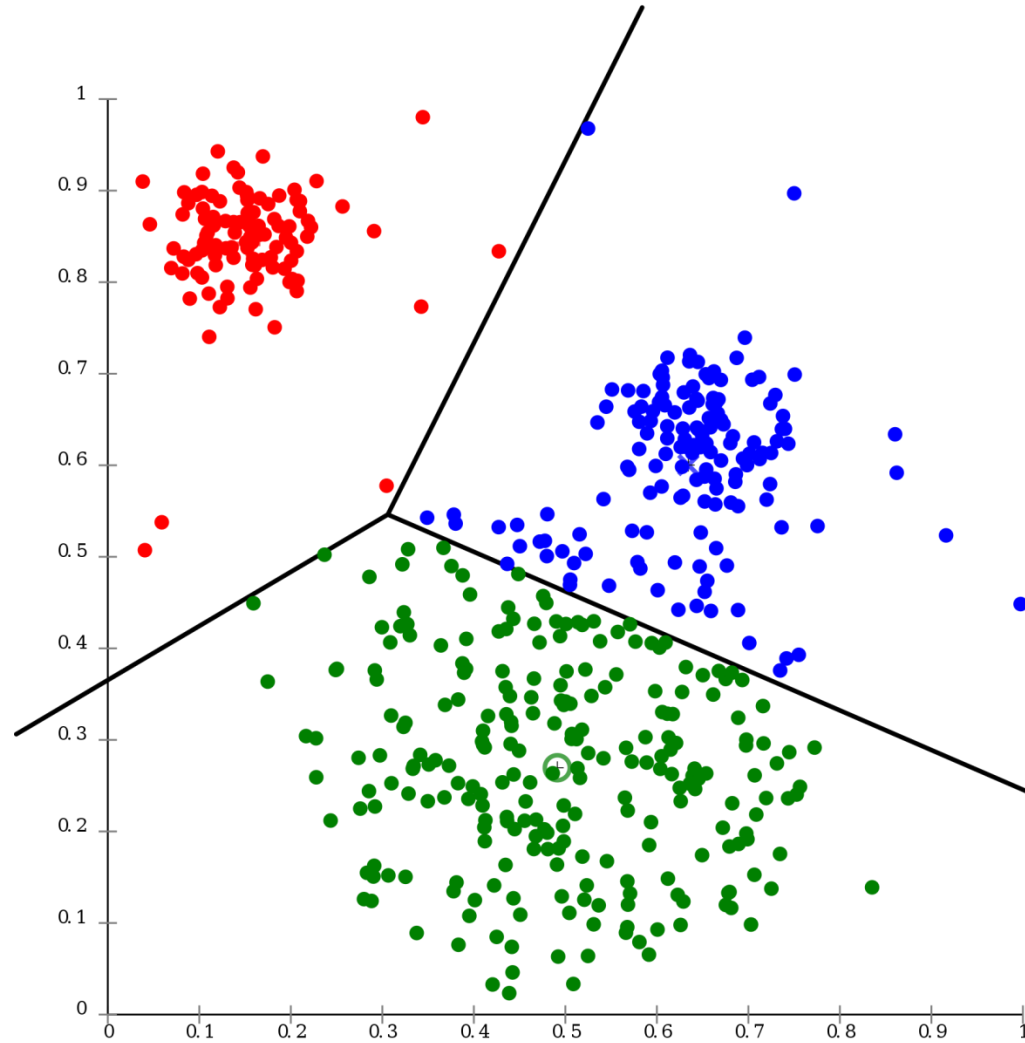
# AUTOENCODERS



# AGENDA

- Unsupervised Learning (Introduction)
- Autoencoder (AE)
- Convolutional AE
- Regularization: Sparse
- Denoising AE
- Stacked AE

# INTRODUCTION TO UNSUPERVISED LEARNING



# SUPERVISED LEARNING

## Supervised Learning

Data:  $(X, Y)$

Goal: Learn a Mapping  
Function  $f$  where:

$$f(X) = Y$$

**Classification**



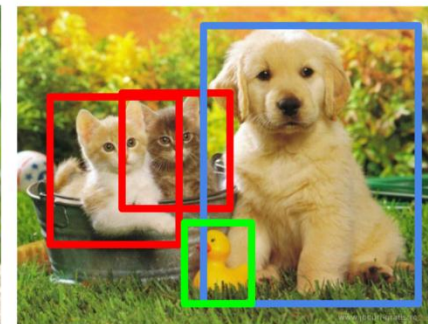
CAT

**Classification  
+ Localization**



CAT

**Object Detection**



CAT, DOG, DUCK

**Instance  
Segmentation**



CAT, DOG, DUCK

Single object

Multiple objects

# SUPERVISED LEARNING

## Examples: Classification.

Decision Trees

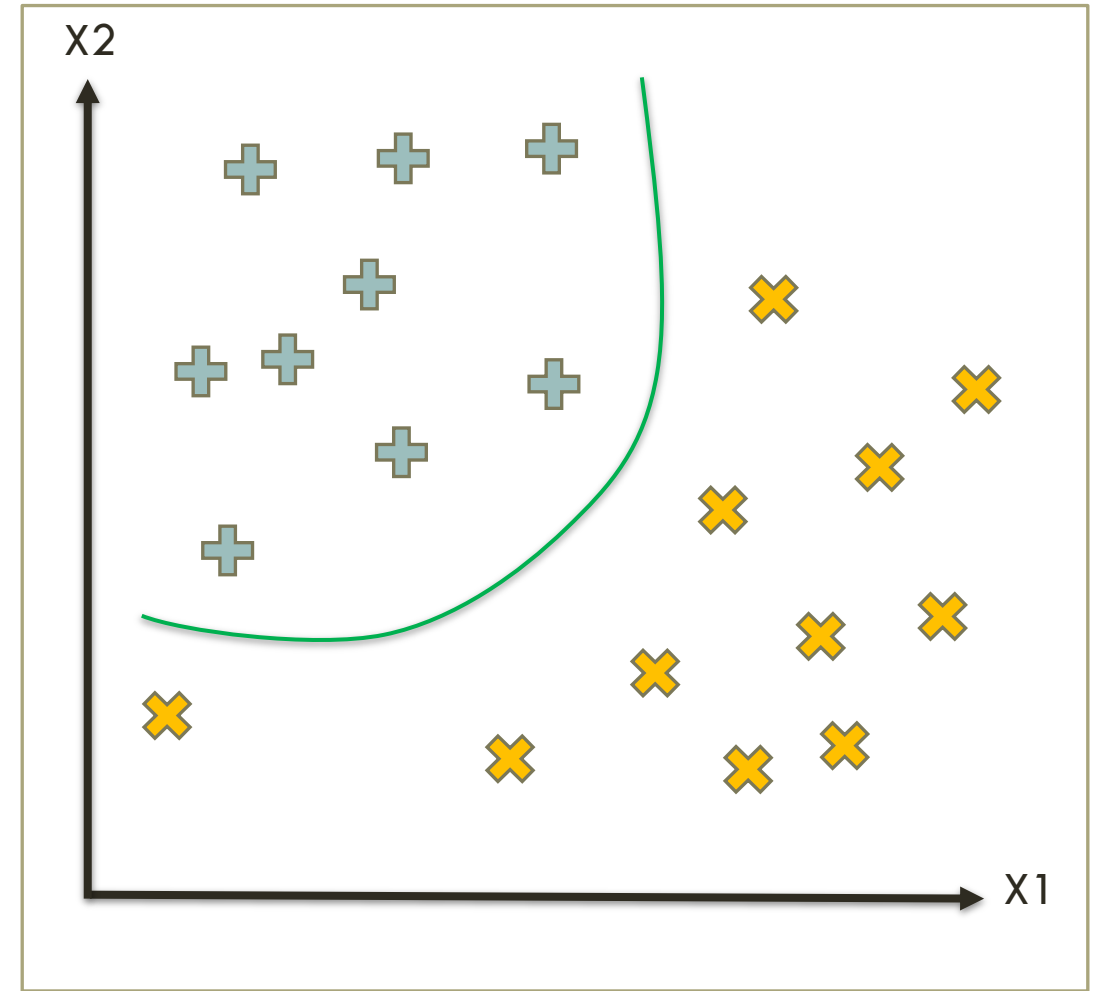
Naïve Bayes

KNN

SVM

Perceptron

Multi Layer  
Perceptron



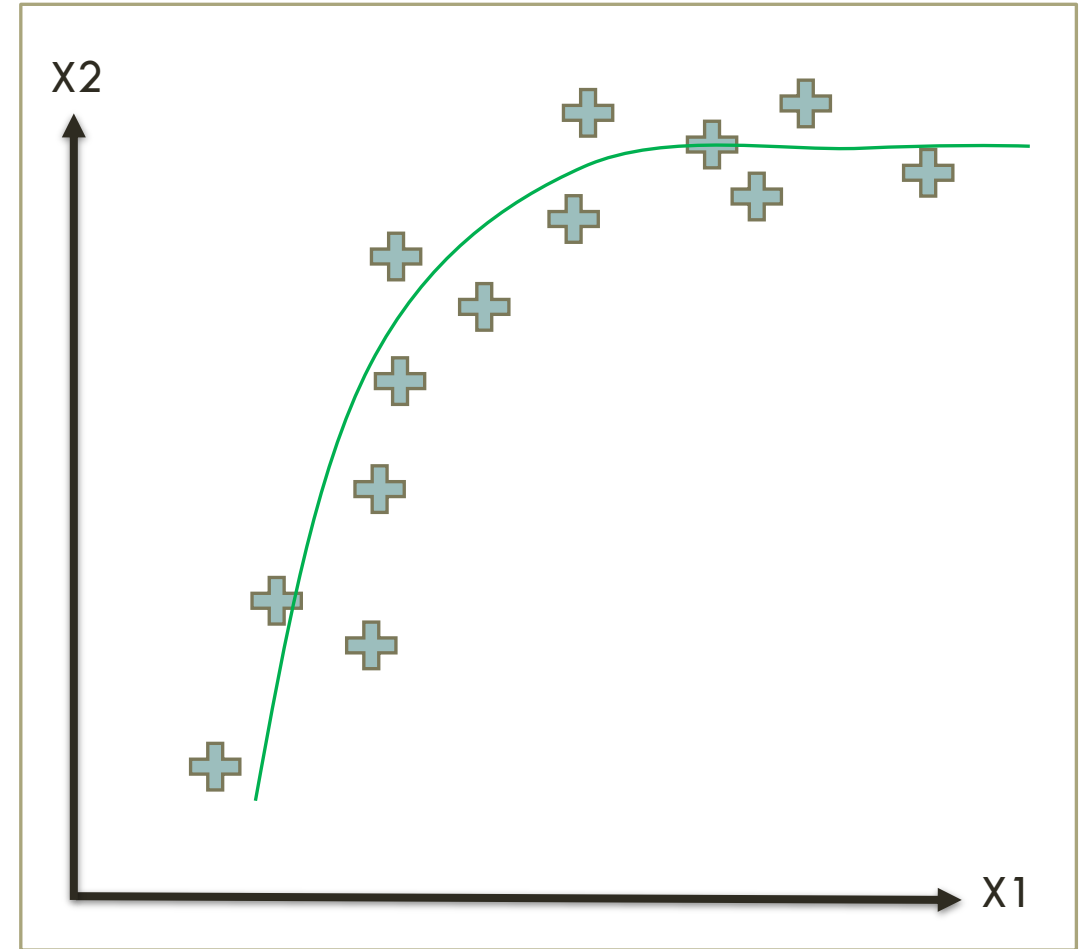
Classification

# SUPERVISED LEARNING

Examples: Regression.

Linear Regression

Logistic Regression



Regression

# SUPERVISED LEARNING VS UNSUPERVISED LEARNING

01

What happens when our labels are noisy?

- Missing values.
- Labeled incorrectly.

02

What happens where we don't have labels for training **at all**?

# SUPERVISED LEARNING VS UNSUPERVISED LEARNING

---

Up until now we have encountered mostly **Supervised Learning** problems and algorithms.

---

Lets talk about **Unsupervised Learning**

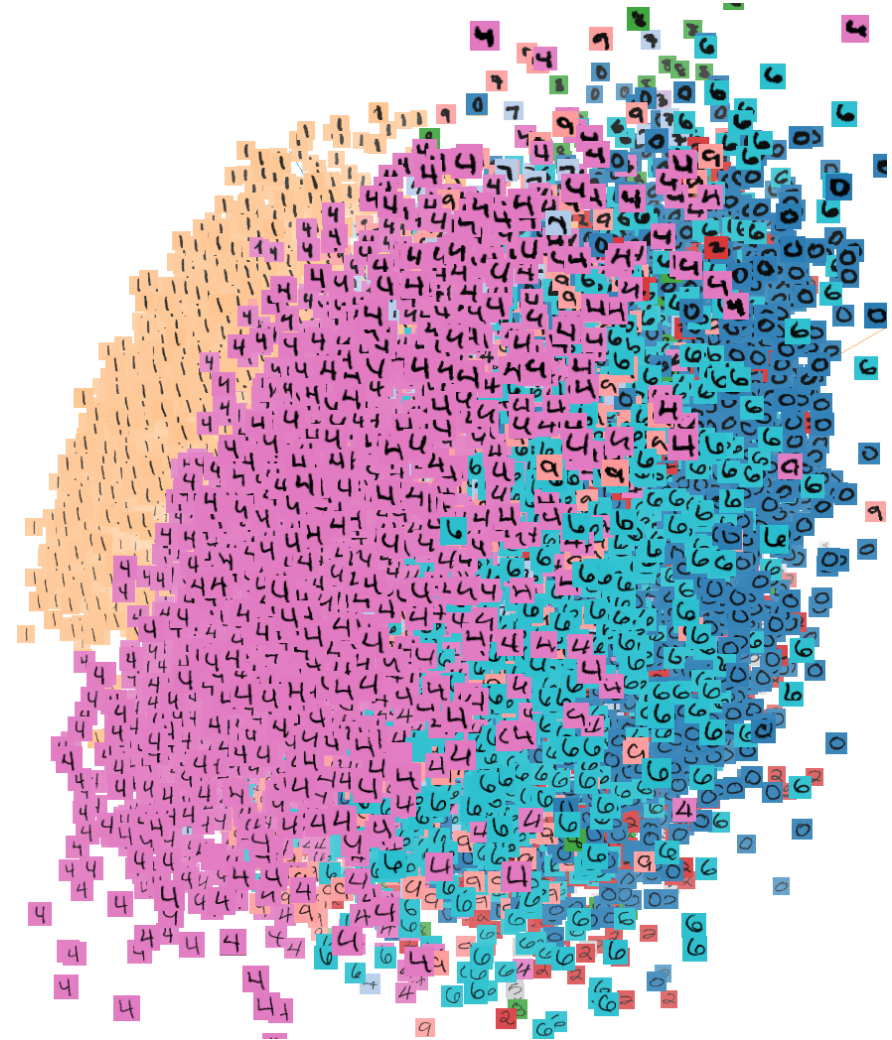


# UNSUPERVISED LEARNING

Unsupervised Learning

Data:  $X$  (no labels!)

Goal: Learn the structure of the data  
(learn correlations between features)

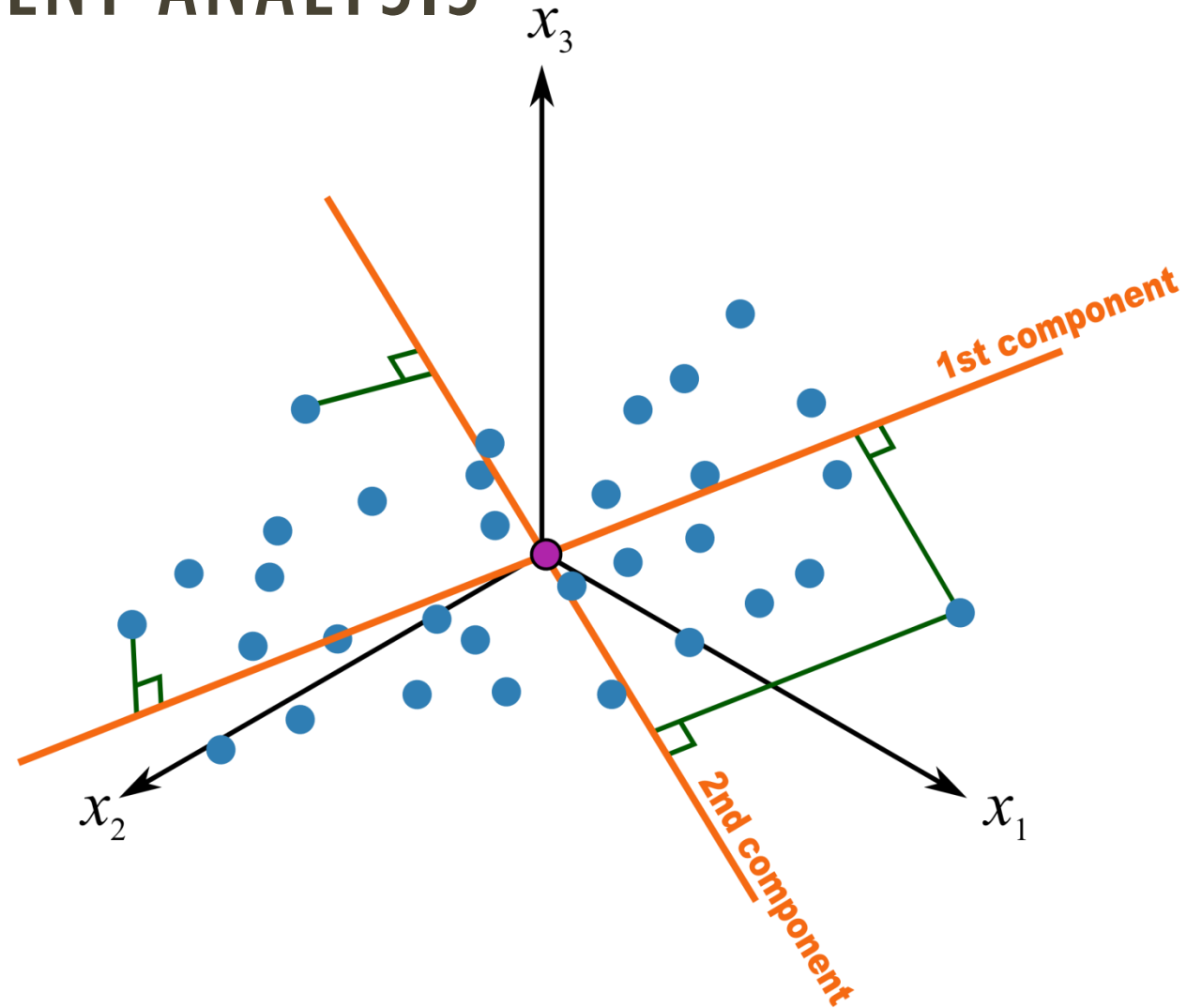


# UNSUPERVISED LEARNING

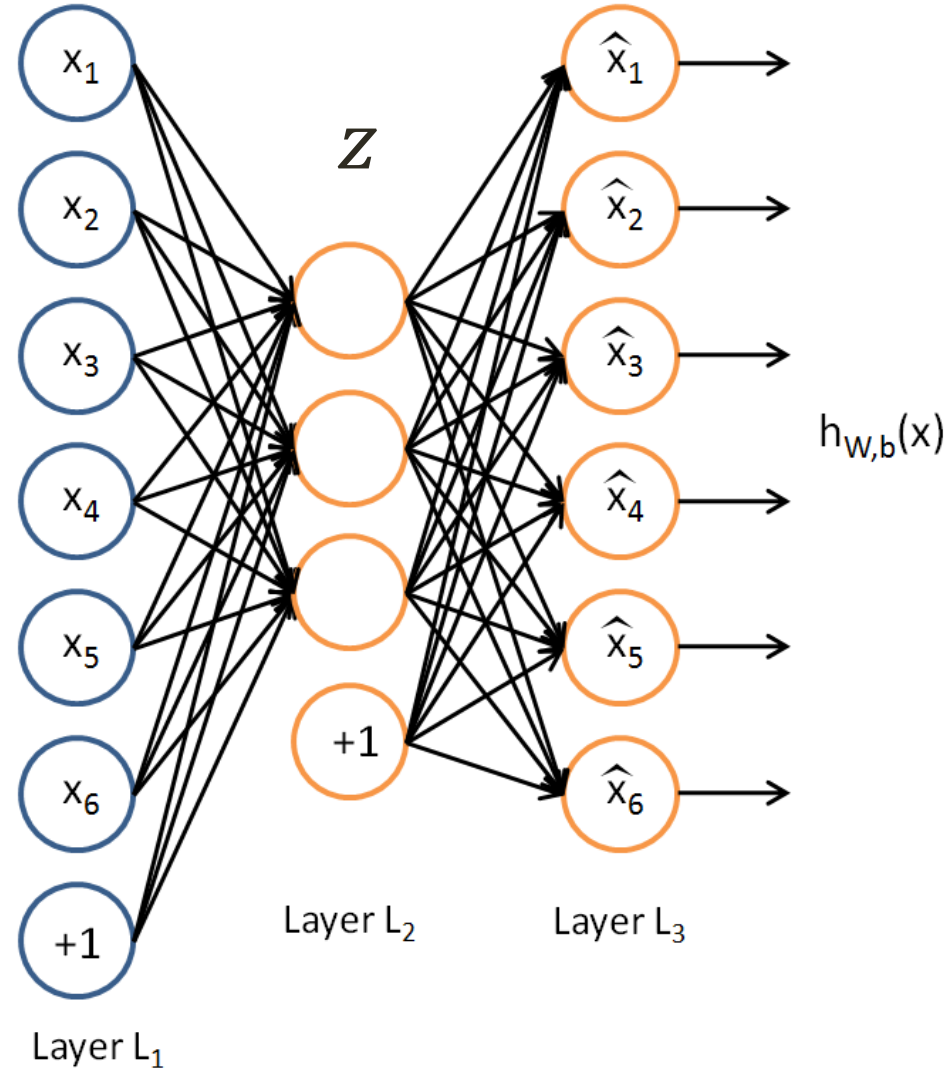
Examples: Clustering, **Compression, Feature & Representation learning, Dimensionality reduction**, Generative models ,etc.

# PCA — PRINCIPAL COMPONENT ANALYSIS

- Statistical approach for data compression and visualization
- Invented by Karl Pearson in 1901
- Weakness: linear components only.

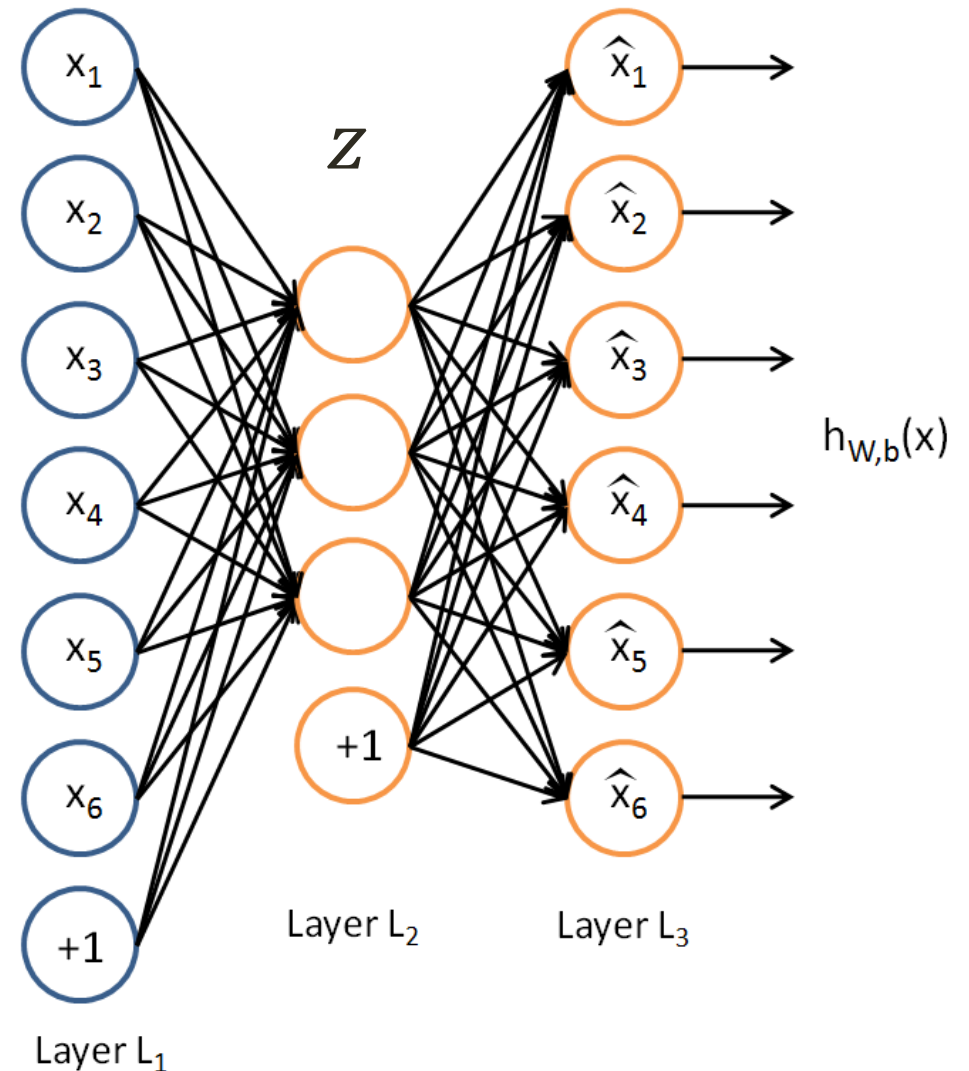


# TRADITIONAL AUTOENCODER



# TRADITIONAL AUTOENCODER

- Unlike the **PCA** now we can use activation functions to achieve non-linearity.
- It has been shown that an AE without activation functions achieves the **PCA** capacity.



# USES

- The autoencoder idea was a part of NN history for decades (LeCun et al, 1987).
- Traditionally an autoencoder is used for **dimensionality reduction** and **feature learning**.
- Recently, the connection between autoencoders and latent space modeling has brought autoencoders to the front of generative modeling.

- **Not used for compression.**
  - Data specific compression.
  - Lossy.

# SIMPLE IDEA

- Given data  $x$  (no labels) we would like to learn the functions  $f$  (encoder) and  $g$  (decoder) where:

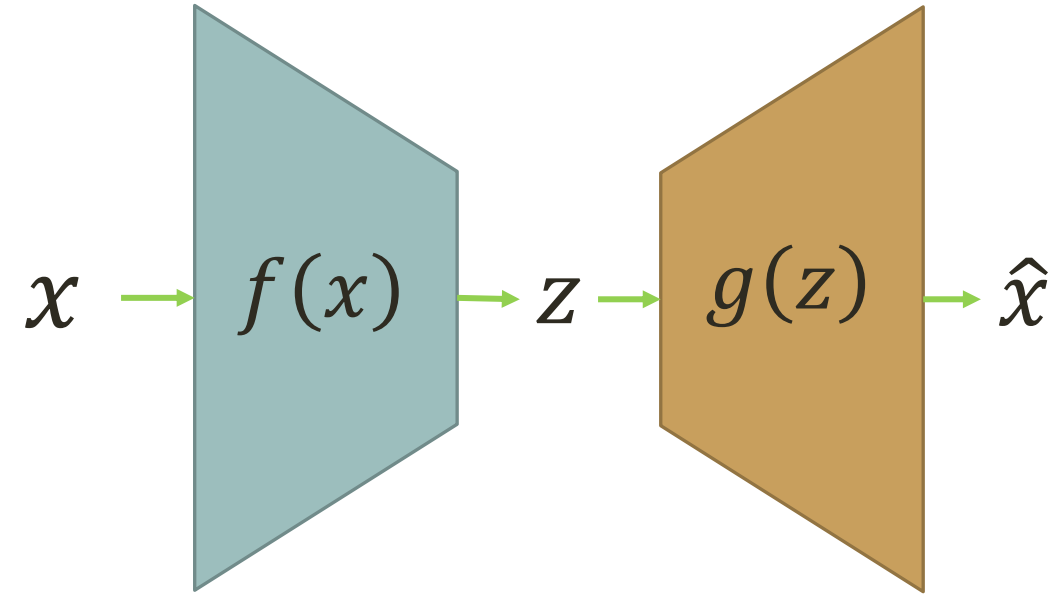
$$f(x) = \text{act}(wx + b) = z$$

and

$$g(z) = \text{act}(w'z + b') = \hat{x}$$

$$\text{s.t } h(x) = g(f(x)) = \hat{x}$$

where  $h$  is an **approximation** of the identity function.




( $z$  is some **latent** representation or **code** and “act” is a non-linearity such as the sigmoid)

( $\hat{x}$  is  $x$ 's reconstruction)

# SIMPLE IDEA

Learning the identity function seems trivial, but with added constraints on the network (such as limiting the number of hidden neurons or regularization) we can learn information about the structure of the data.



Trying to capture the distribution of the data (data specific!)



# TRAINING THE AE

Using **Gradient Descent** we can simply train the model as any other FC NN with:

- Traditionally with squared error loss function

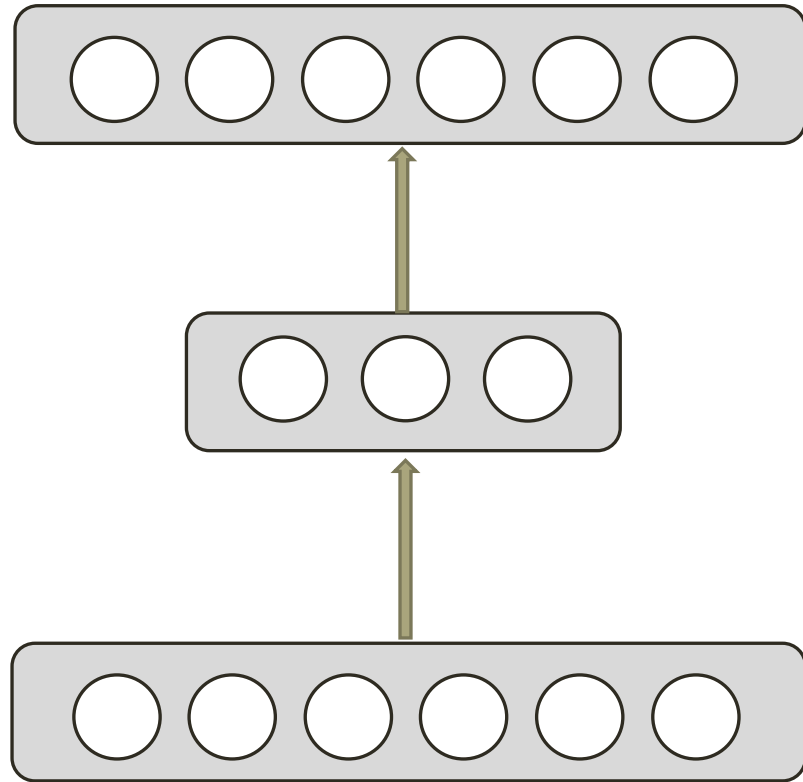
$$L(x, \hat{x}) = \|x - \hat{x}\|^2$$

- If our input is interpreted as bit vectors or vectors of bit probabilities the cross entropy can be used

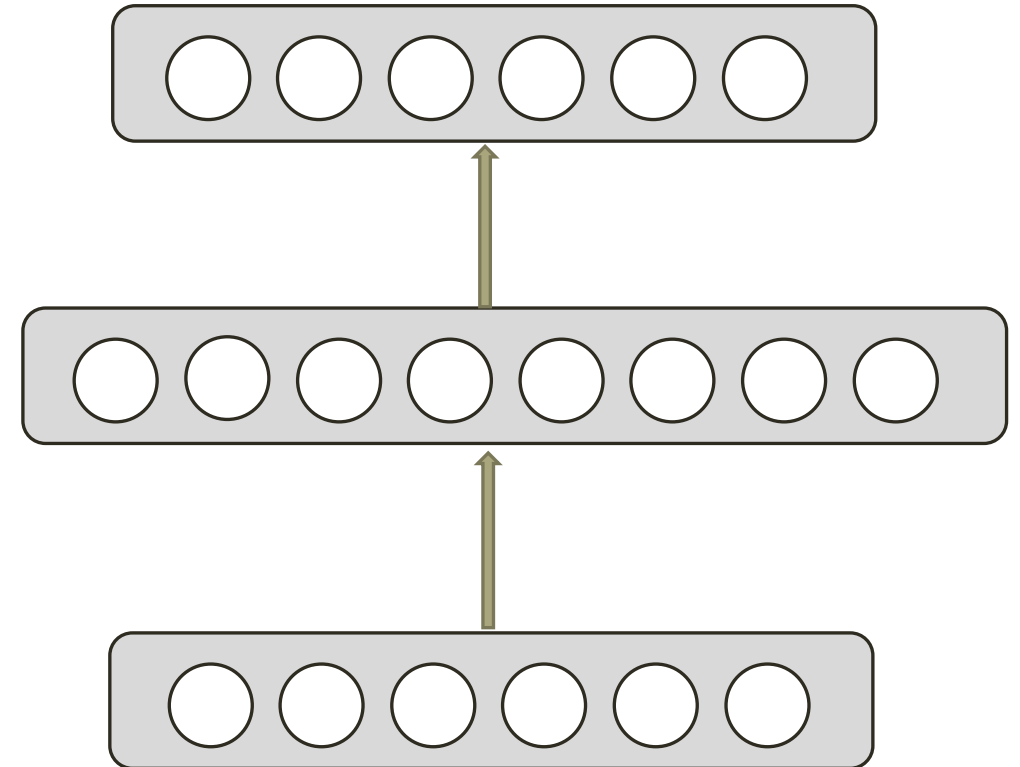
$$H(p, q) = - \sum_x p(x) \log q(x)$$

# UNDERCOMPLETE AE VS OVERCOMPLETE AE

We distinguish between two types of AE structures:



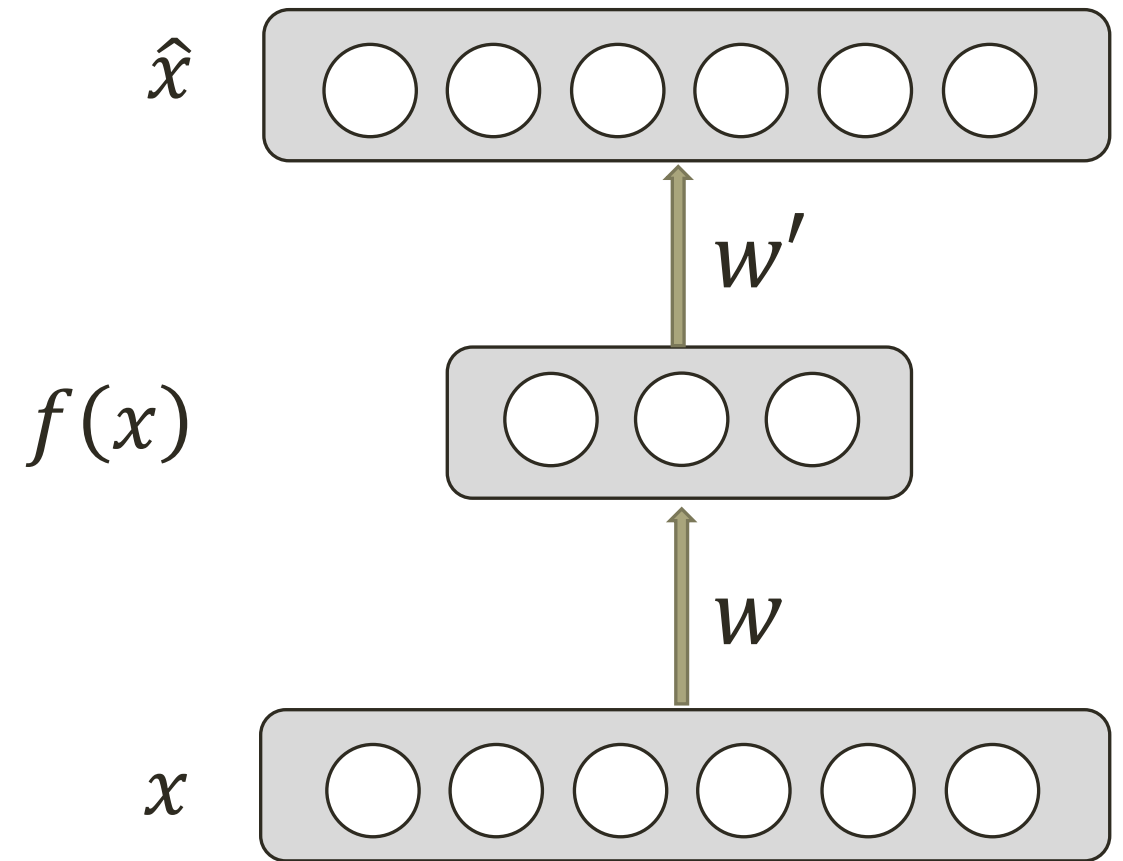
**Under Complete**



**Over Complete**

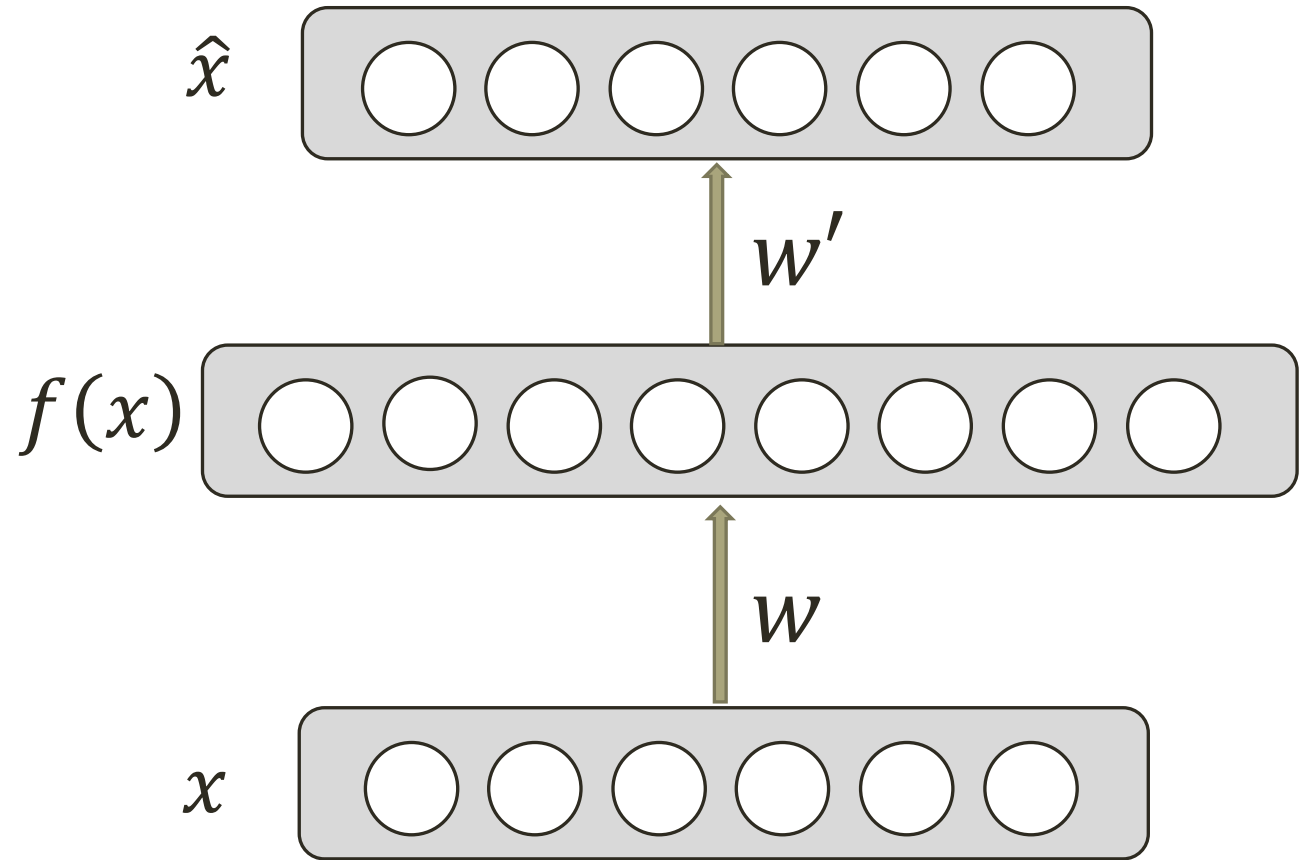
# UNDERCOMPLETE AE

- Hidden layer is **Undercomplete** if smaller than the input layer
  - ❑ Compresses the input
  - ❑ Compresses well only for the training dist.
- Hidden nodes will be
  - ❑ Good features for the training distribution.
  - ❑ Bad for other types on input



# OVERCOMPLETE AE

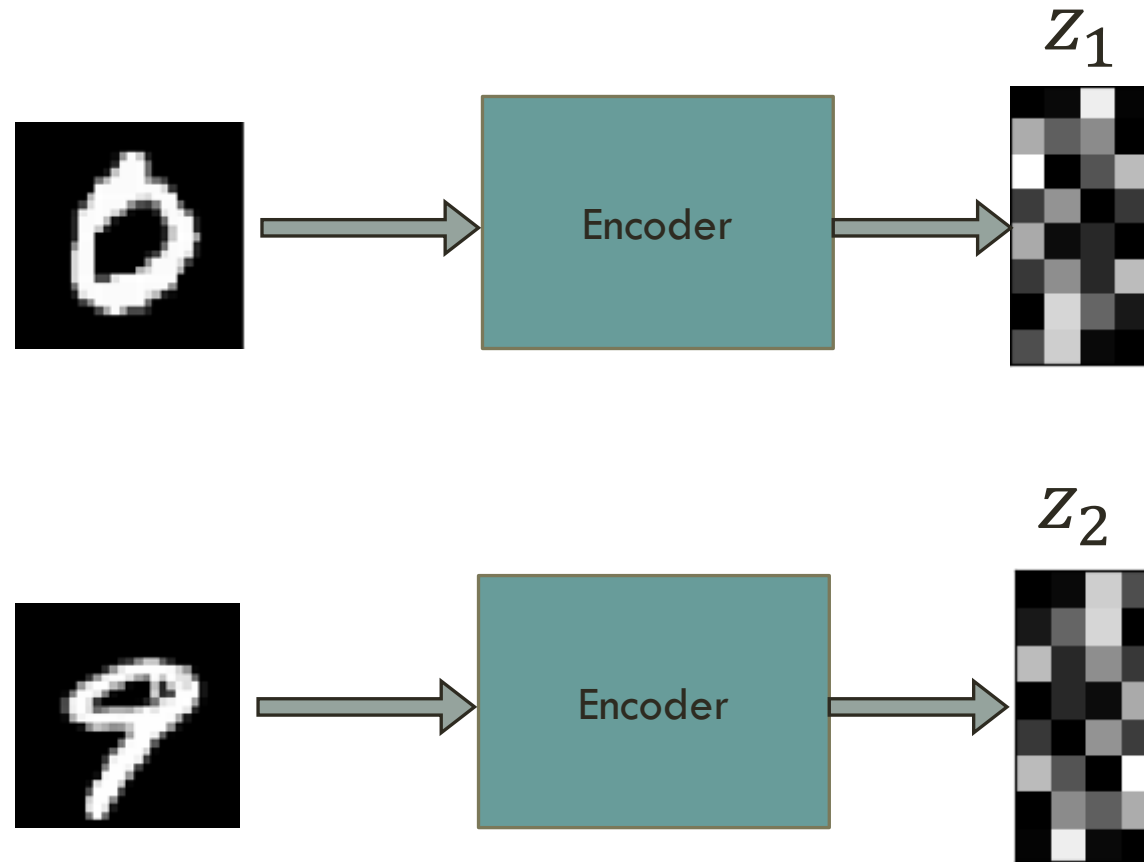
- Hidden layer is **Overcomplete** if greater than the input layer
  - ❑ No compression in hidden layer.
  - ❑ Each hidden unit could copy a different input component.
- No guarantee that the hidden units will extract meaningful structure.
- Adding dimensions is good for training a linear classifier (XOR case example).
- A higher dimension code helps model a more complex distribution.



# DEEP AUTOENCODER DEMO

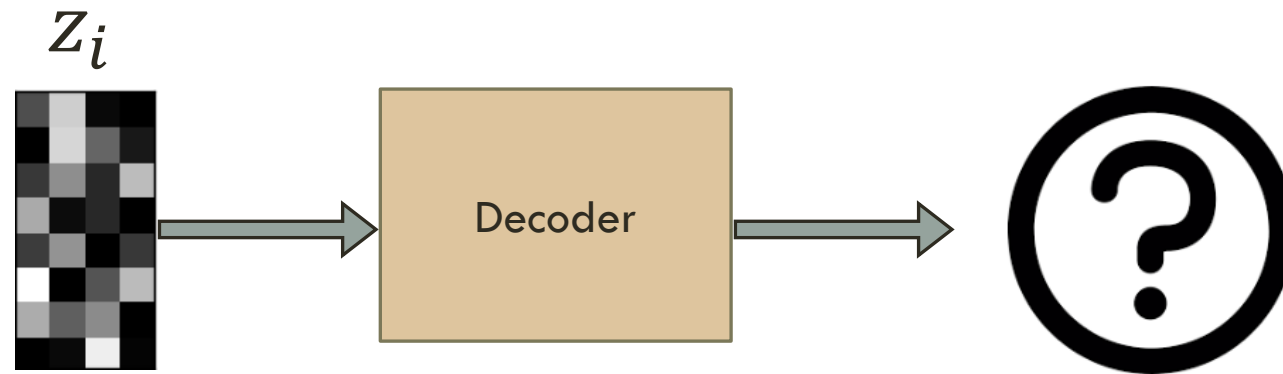
<https://cs.stanford.edu/people/karpathy/convnetjs/demo/autoencoder.html>

# SIMPLE LATENT SPACE INTERPOLATION

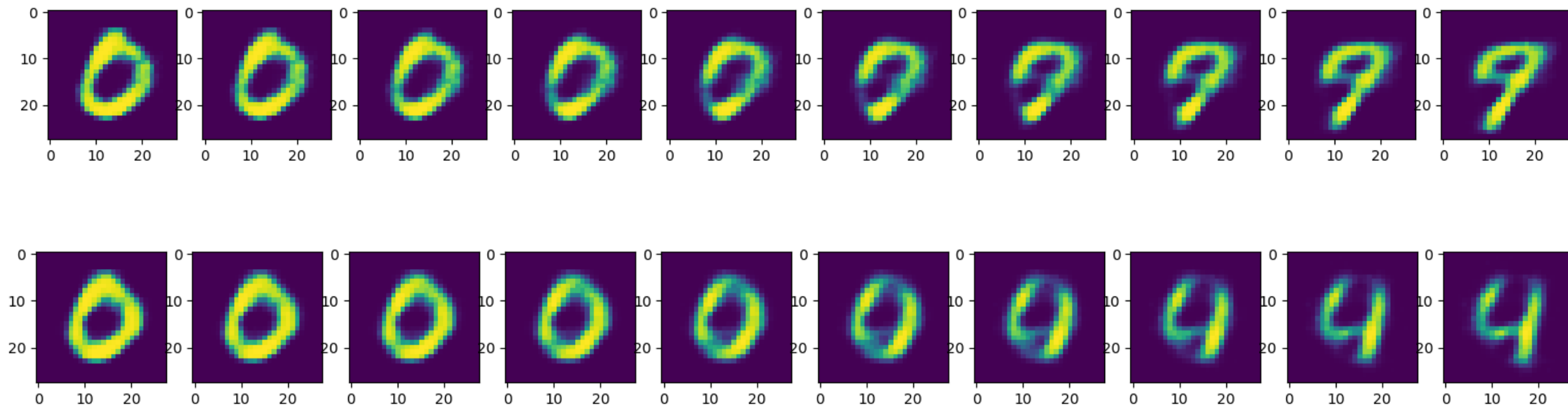


# SIMPLE LATENT SPACE INTERPOLATION

$$z_i = \alpha \begin{matrix} z_1 \\ \text{[noise pattern]} \end{matrix} + (1 - \alpha) \begin{matrix} z_2 \\ \text{[noise pattern]} \end{matrix}$$

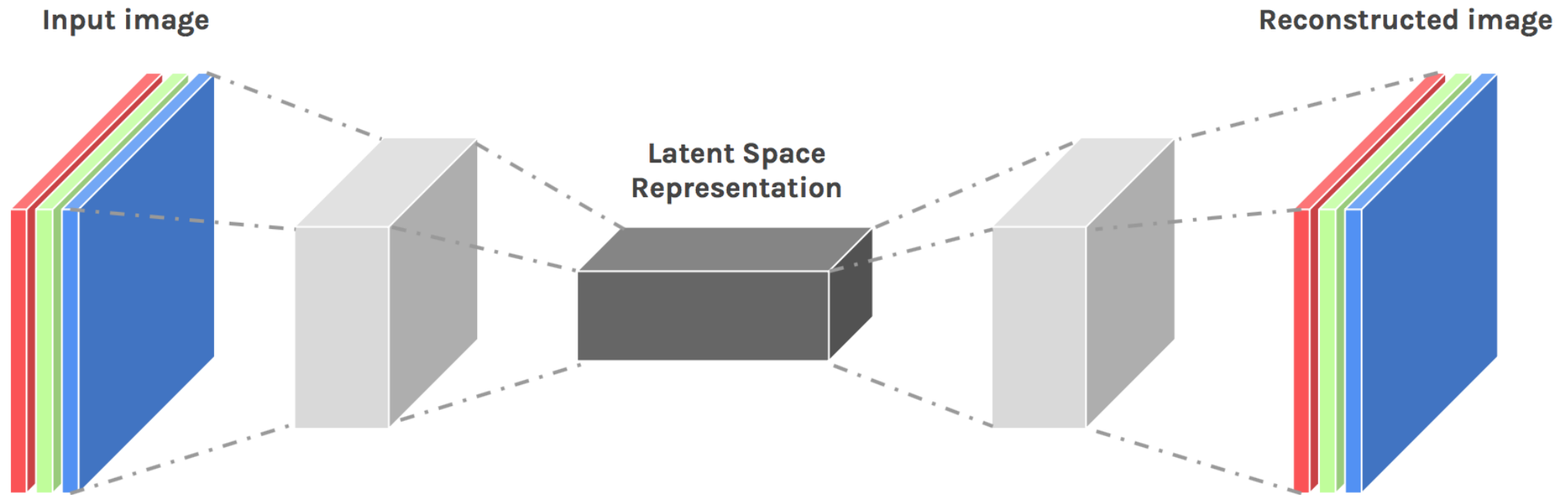


# SIMPLE LATENT SPACE — INTERPOLATION





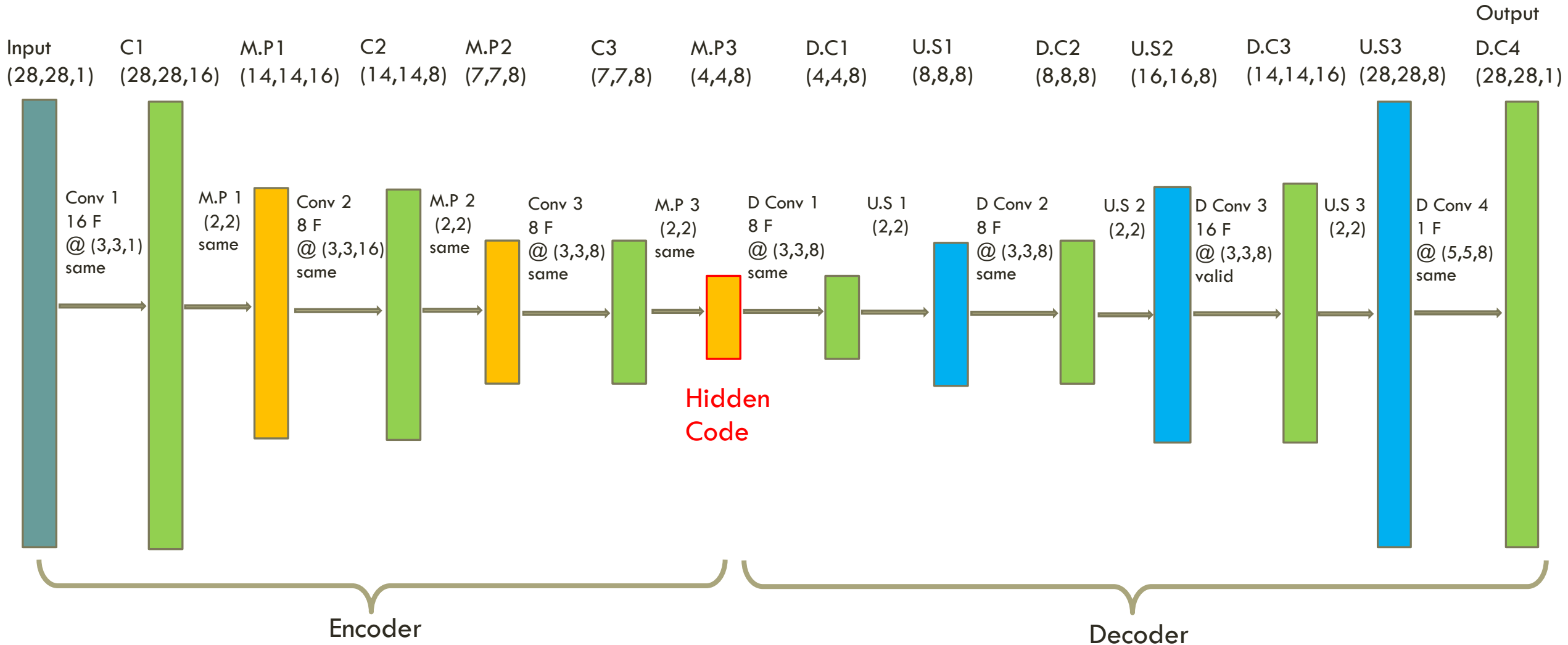
# CONVOLUTIONAL AE



\* Input values are normalized

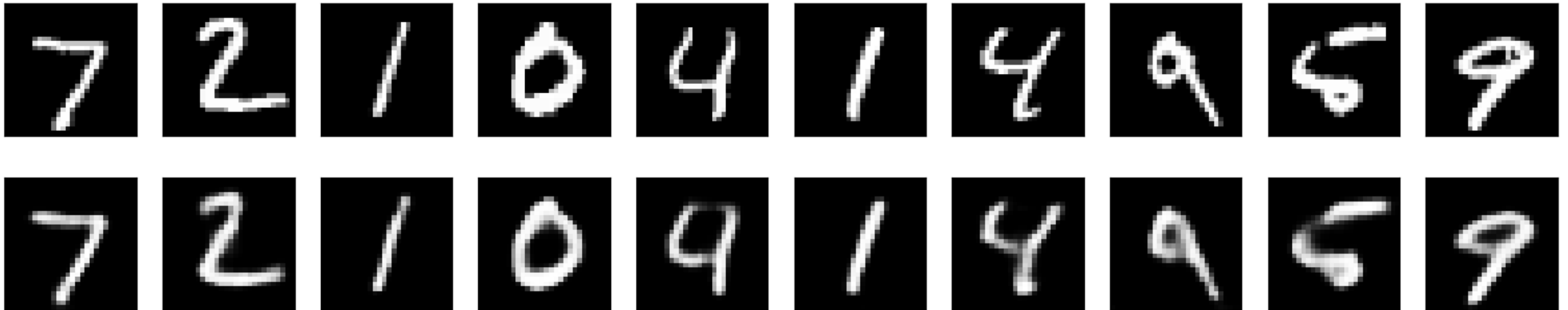
\* All of the conv layers activation functions are relu except for the last conv which is sigmoid

# CONVOLUTIONAL AE



# CONVOLUTIONAL AE — KERAS EXAMPLE RESULTS

- 50 epochs.
- 88% accuracy on validation set.

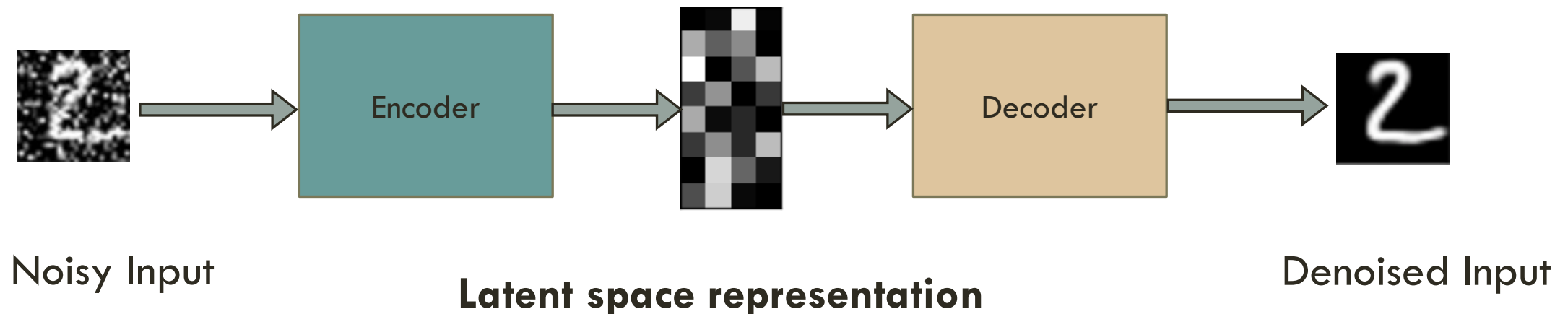


# DENOISING AUTOENCODERS

## Intuition:

- We still aim to encode the input and to NOT mimic the identity function.
- We try to undo the effect of *corruption* process stochastically applied to the input.

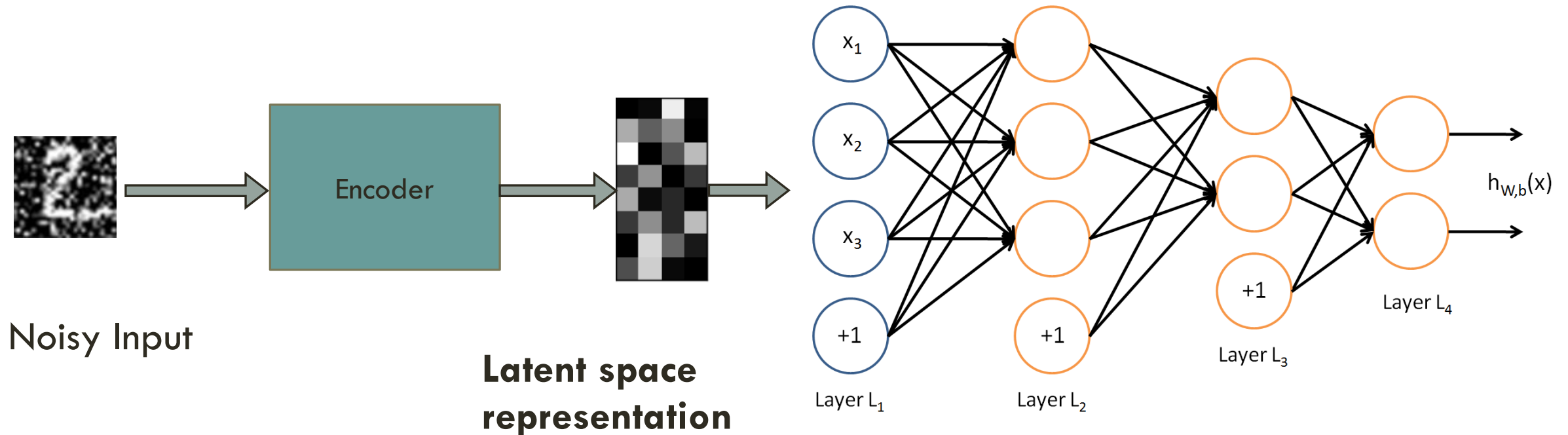
**A more robust model**



# DENOISING AUTOENCODERS

Use Case:

- Extract robust representation for a NN classifier.



# DENOISING AUTOENCODERS

Instead of trying to mimic the identity function by minimizing:

$$L(x, g(f(x)))$$

where  $L$  is some loss function

A **DAE** instead minimizes:

$$L(x, g(f(\tilde{x})))$$

where  $\tilde{x}$  is a copy of  $x$  that has been corrupted by some form of noise.

# DENOISING AUTOENCODERS

Idea: A robust representation against noise:

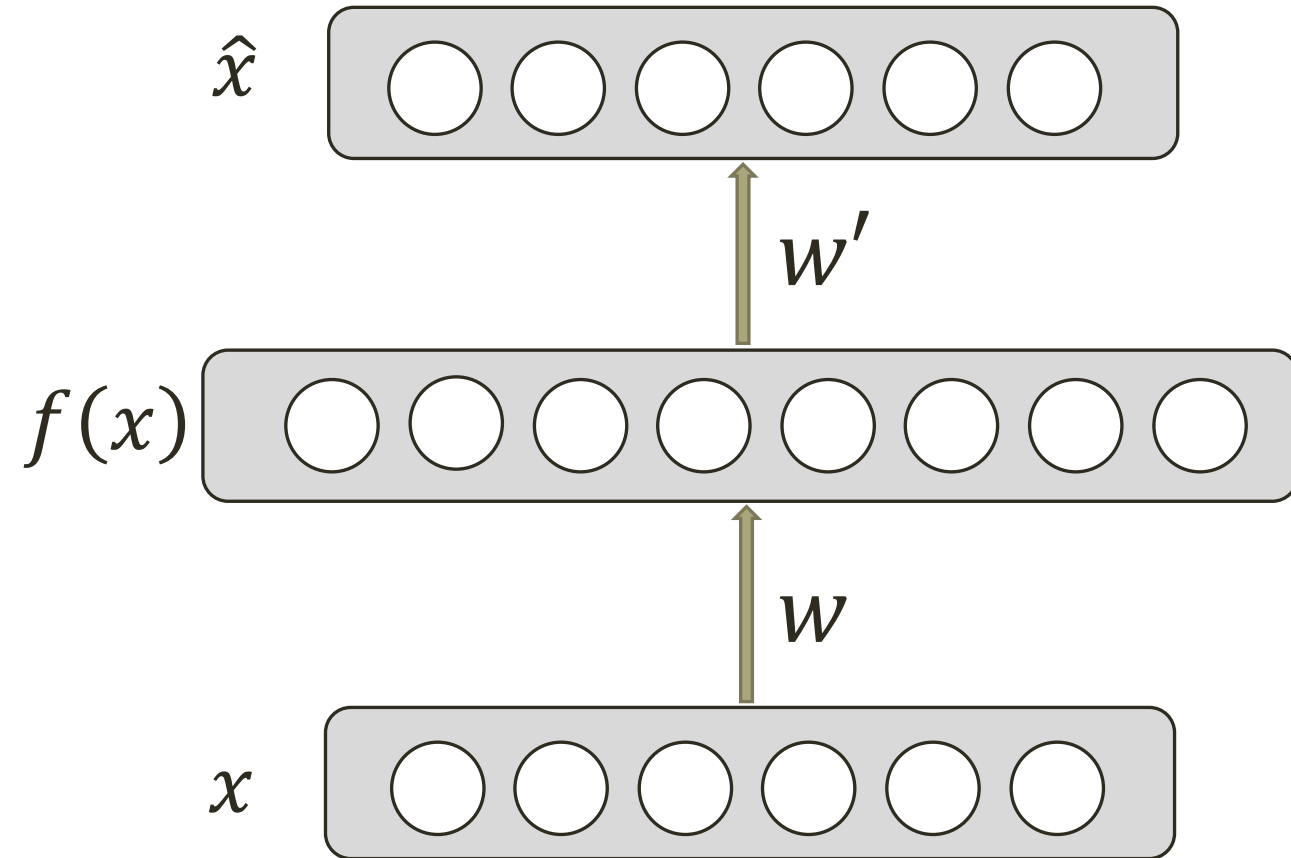
- Random assignment of subset of inputs to 0, with probability  $v$ .
- Gaussian additive noise.



(a)

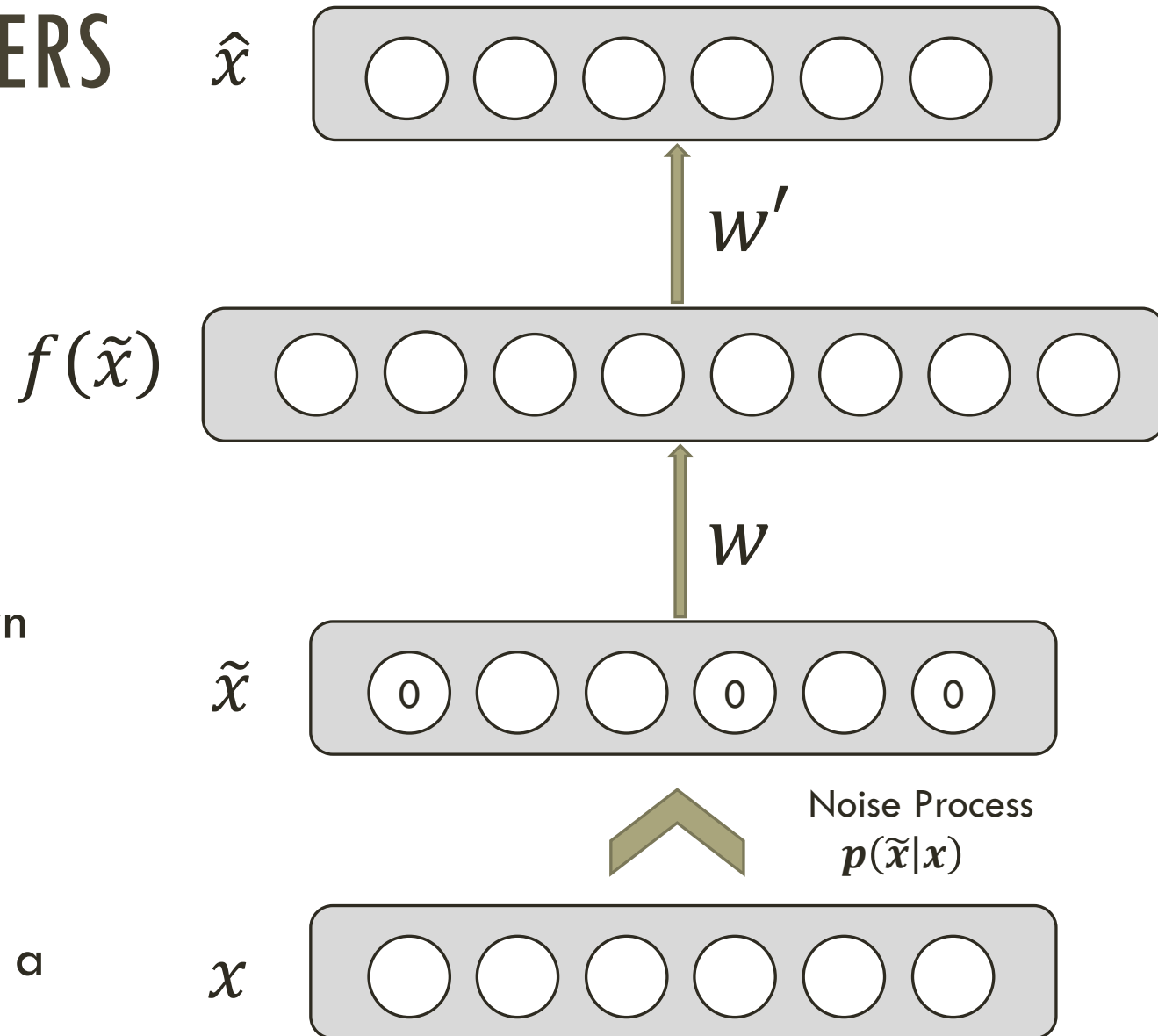


(b)



# DENOISING AUTOENCODERS

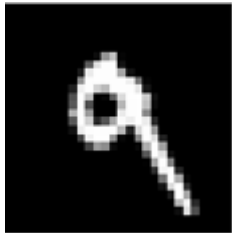
- Reconstruction  $\hat{x}$  computed from the corrupted input  $\tilde{x}$ .
- Loss function compares  $\hat{x}$  reconstruction with the noiseless  $x$ .
- ❖ The autoencoder cannot fully trust each feature of  $x$  independently so it must learn the correlations of  $x$ 's features.
- ❖ Based on those relations we can predict a more 'not prone to changes' model.
- We are forcing the hidden layer to learn a generalized structure of the data.





# DENOISING AUTOENCODERS - PROCESS

Taken some input  $x$



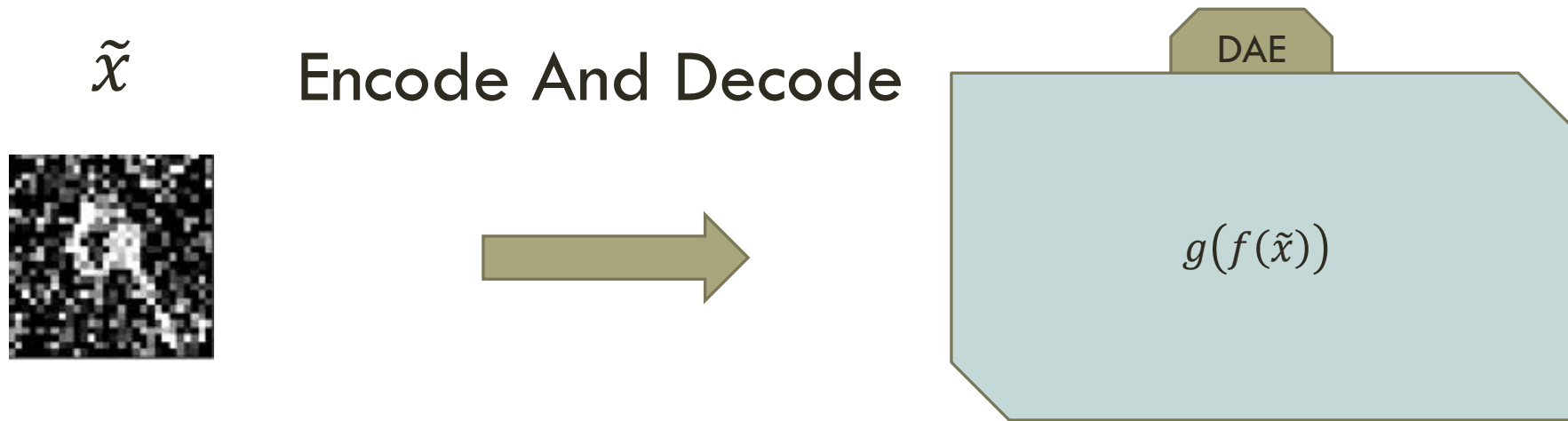
Apply Noise



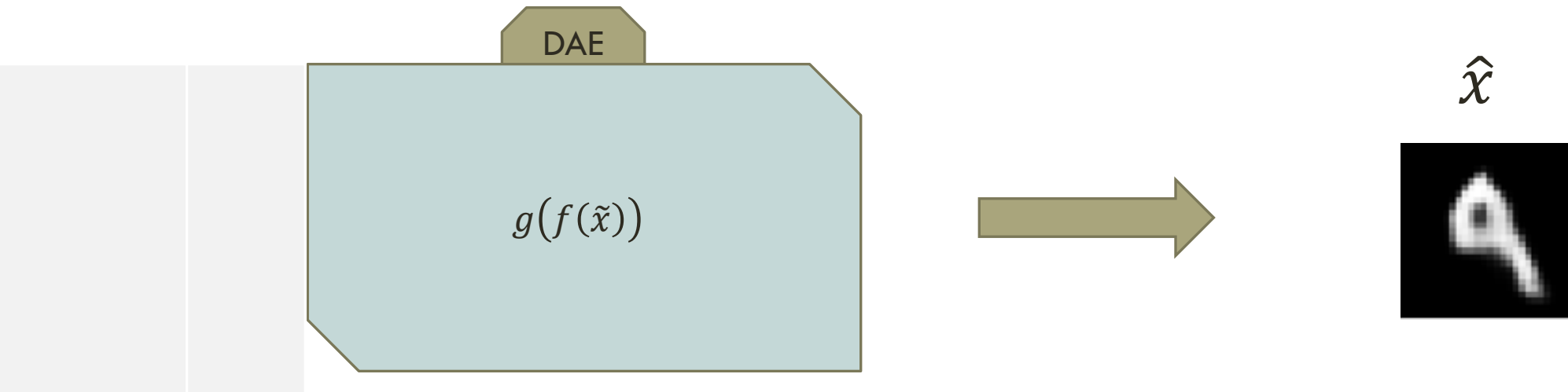
$\tilde{x}$



# DENOISING AUTOENCODERS - PROCESS

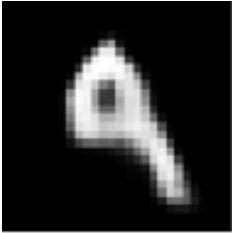


# DENOISING AUTOENCODERS - PROCESS

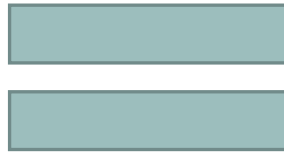


# DENOISING AUTOENCODERS - PROCESS

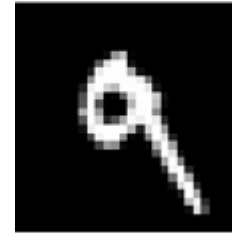
$\hat{x}$



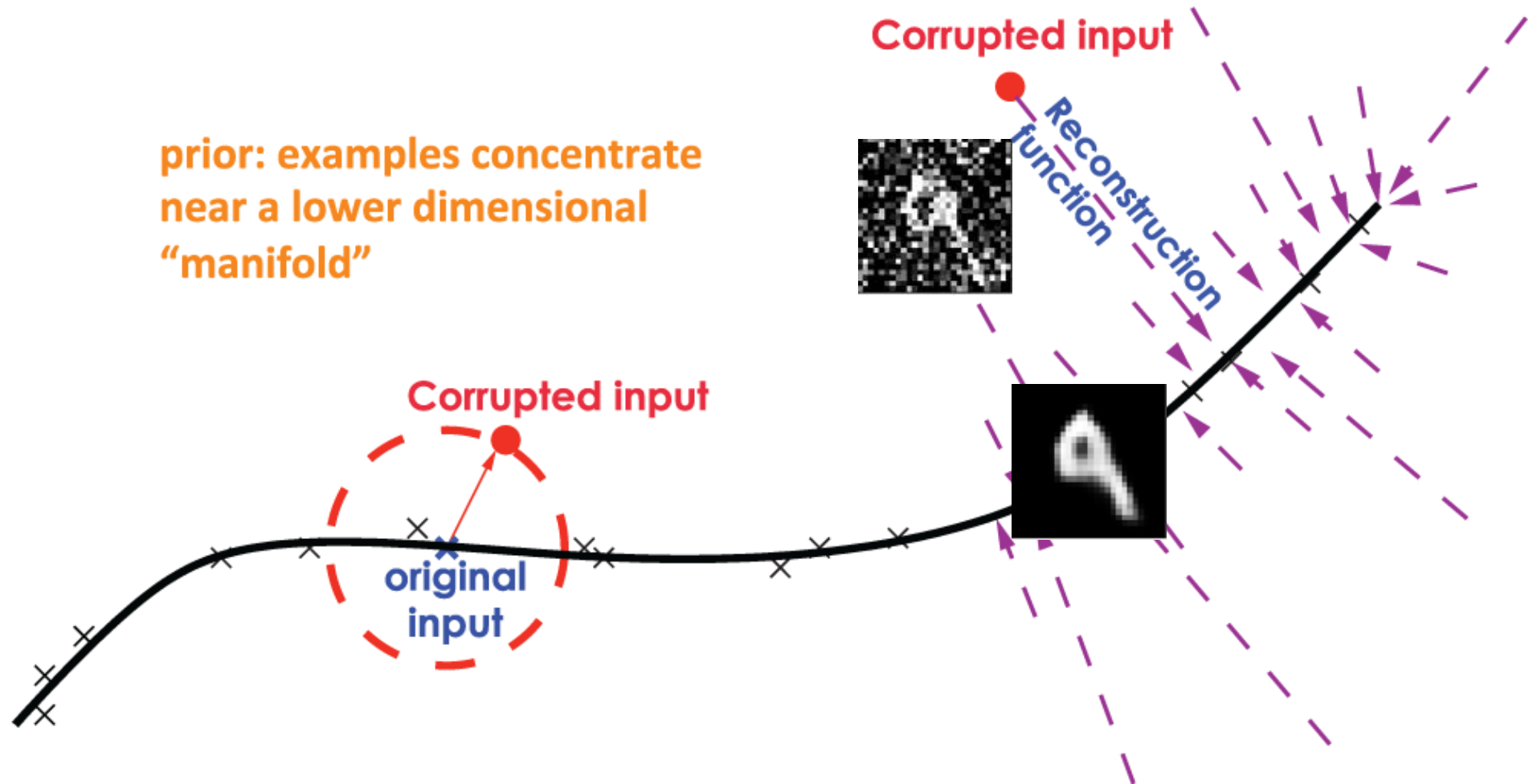
Compare



$x$

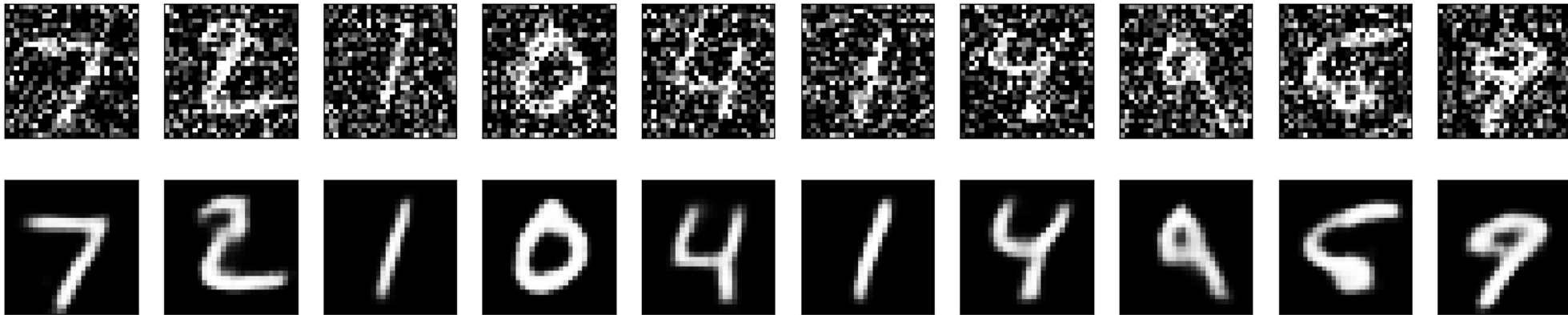


# DENOISING AUTOENCODERS



# DENOISING CONVOLUTIONAL AE — KERAS

- 50 epochs.
- Noise factor 0.5
- 92% accuracy on validation set.



# STACKED AE

## - Motivation:

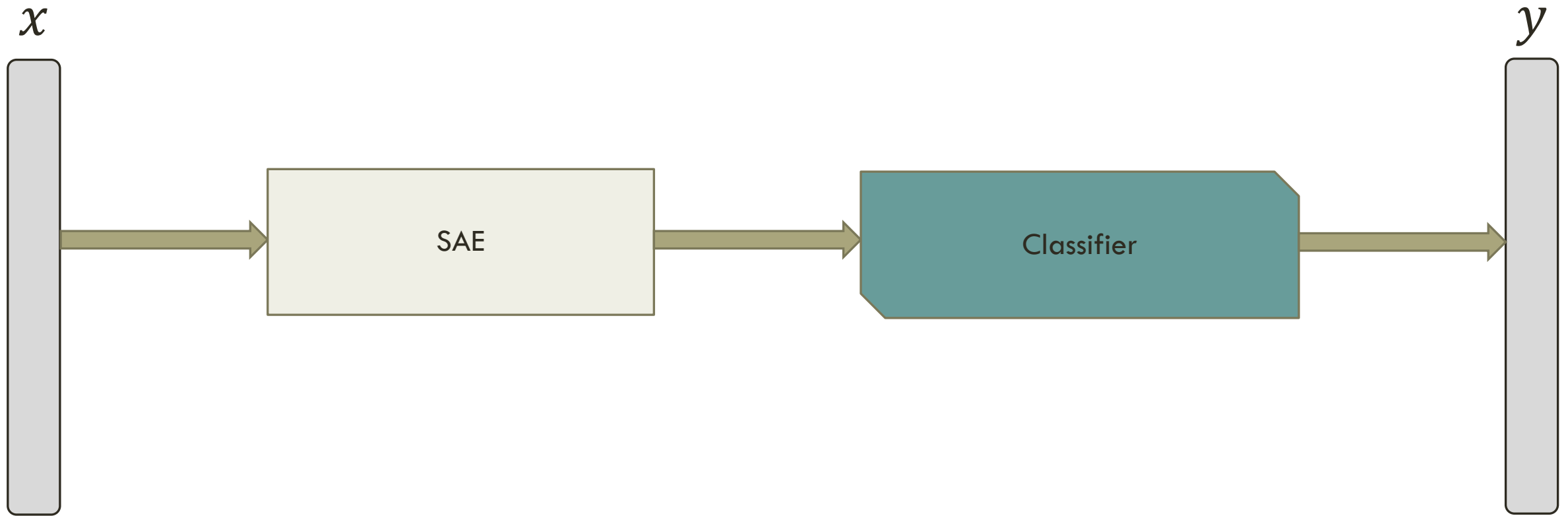
- ❑ We want to harness the feature extraction quality of a AE for our advantage.
- ❑ For example: we can build a deep supervised classifier where it's input is the output of a SAE.
- ❑ The benefit: our deep model's  $W$  are not randomly initialized but are rather “smartly selected”
- ❑ Also using this unsupervised technique lets us have a larger unlabeled dataset.

# STACKED AE

- Building a SAE consists of two phases:
  - 1. Train each AE layer one after the other.
  - 2. Connect any classifier (SVM / FC NN layer etc.)

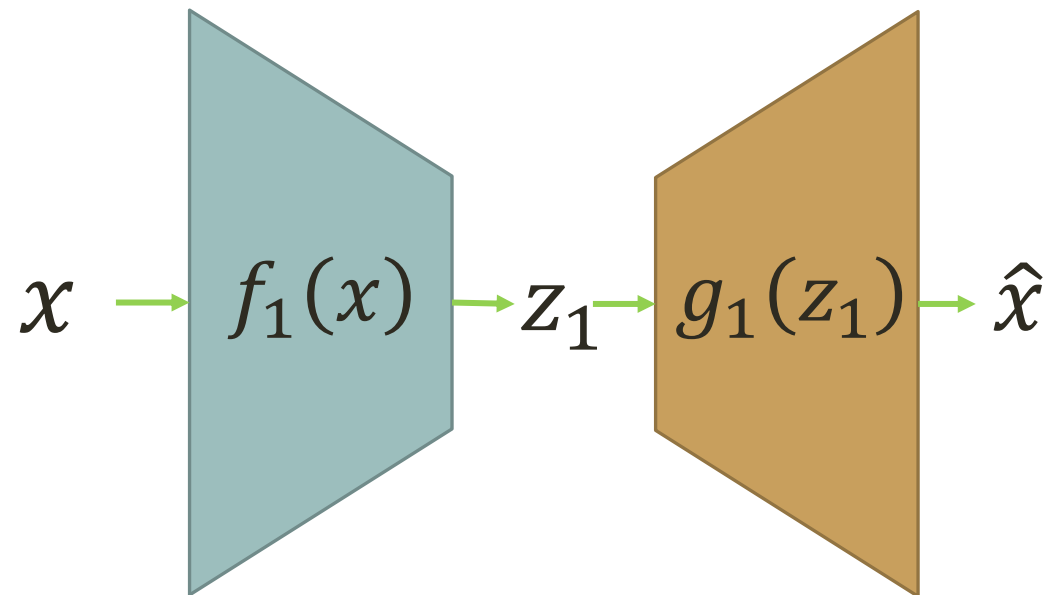


# STACKED AE



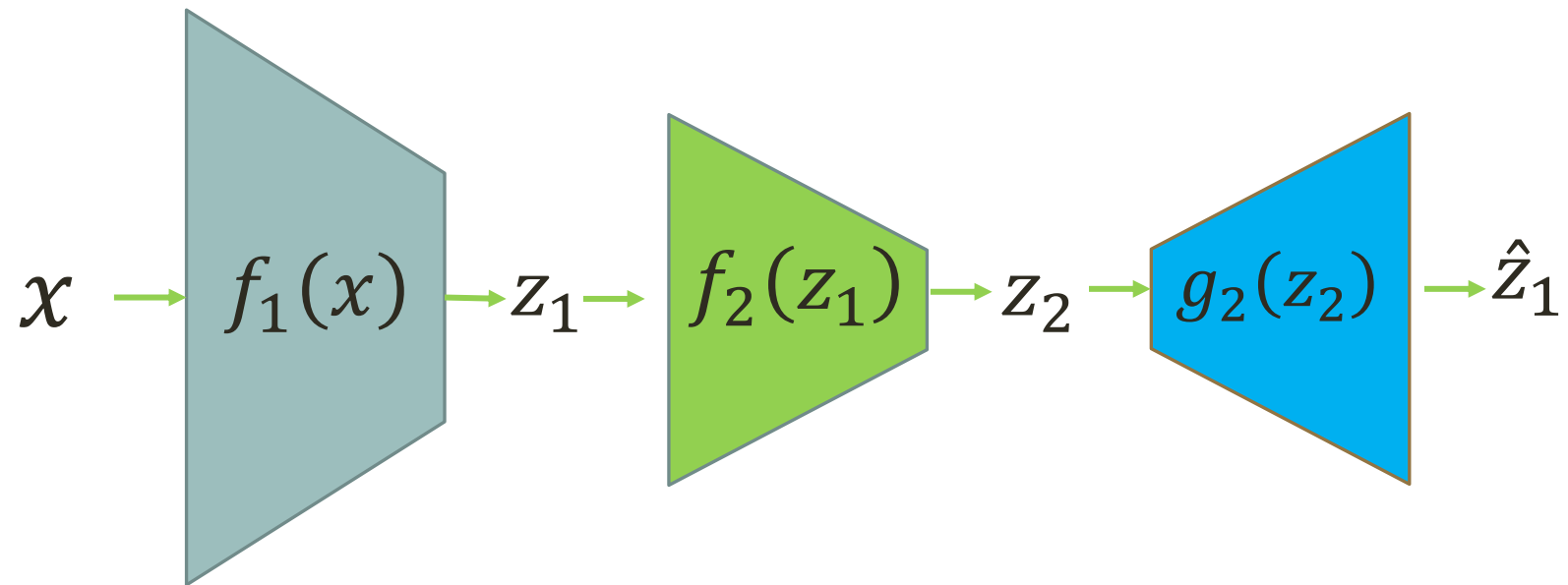
# STACKED AE — TRAIN PROCESS

First Layer Training (AE 1)



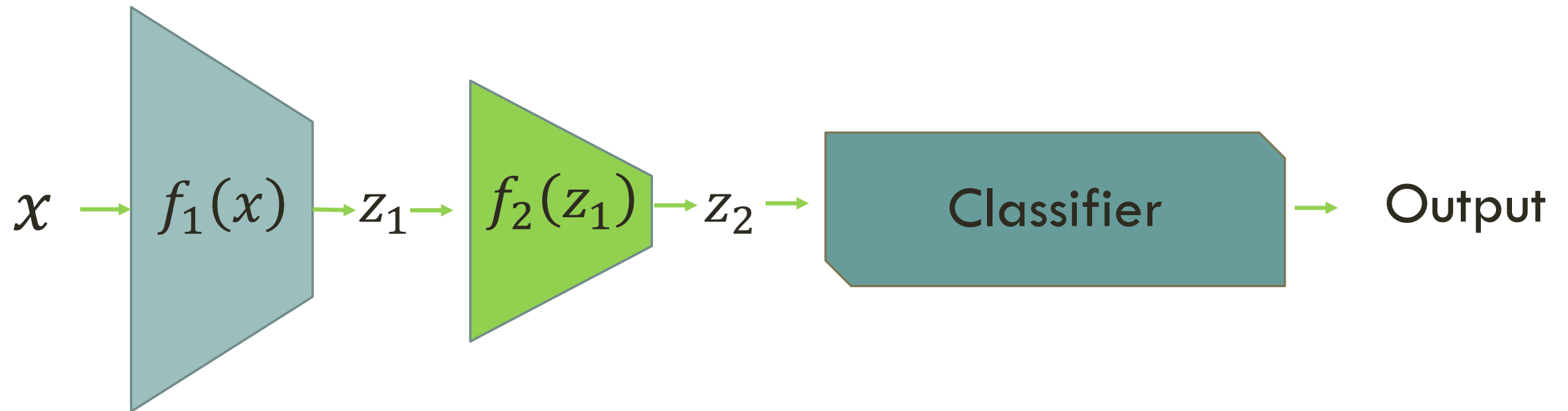
# STACKED AE — TRAIN PROCESS

## Second Layer Training (AE 2)



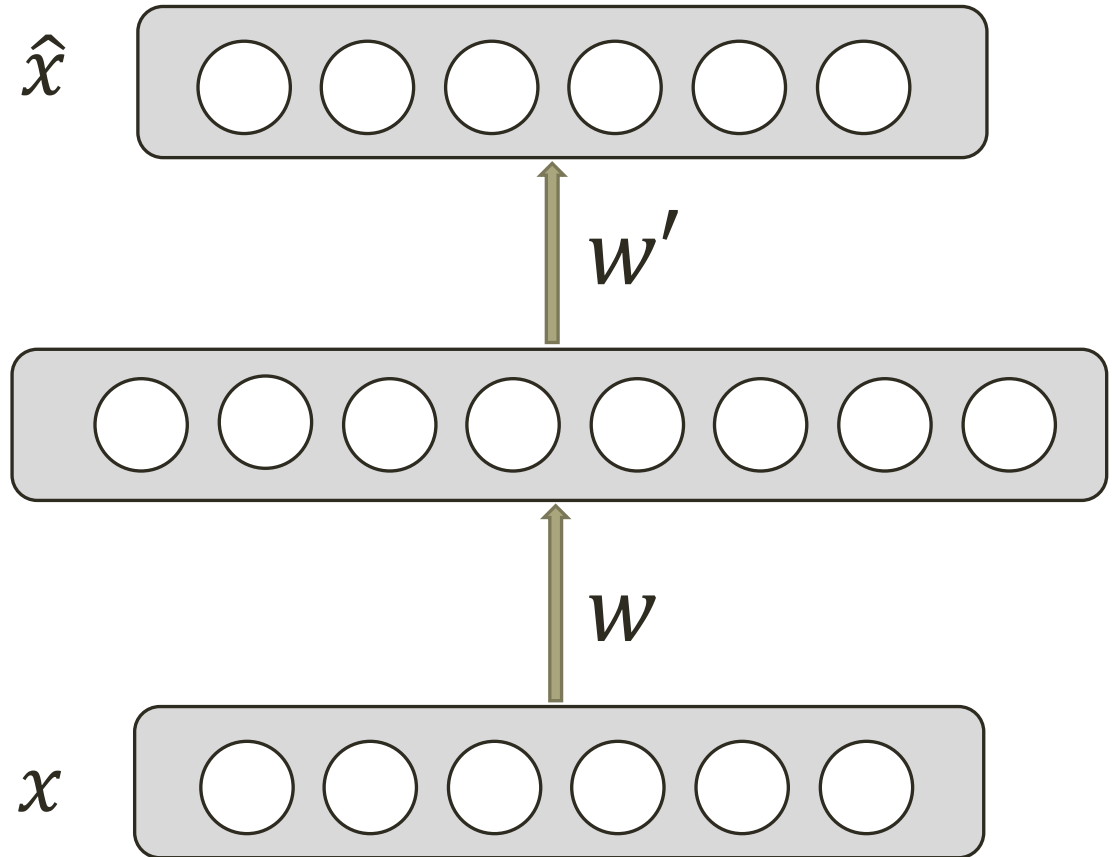
# STACKED AE — TRAIN PROCESS

Add any classifier



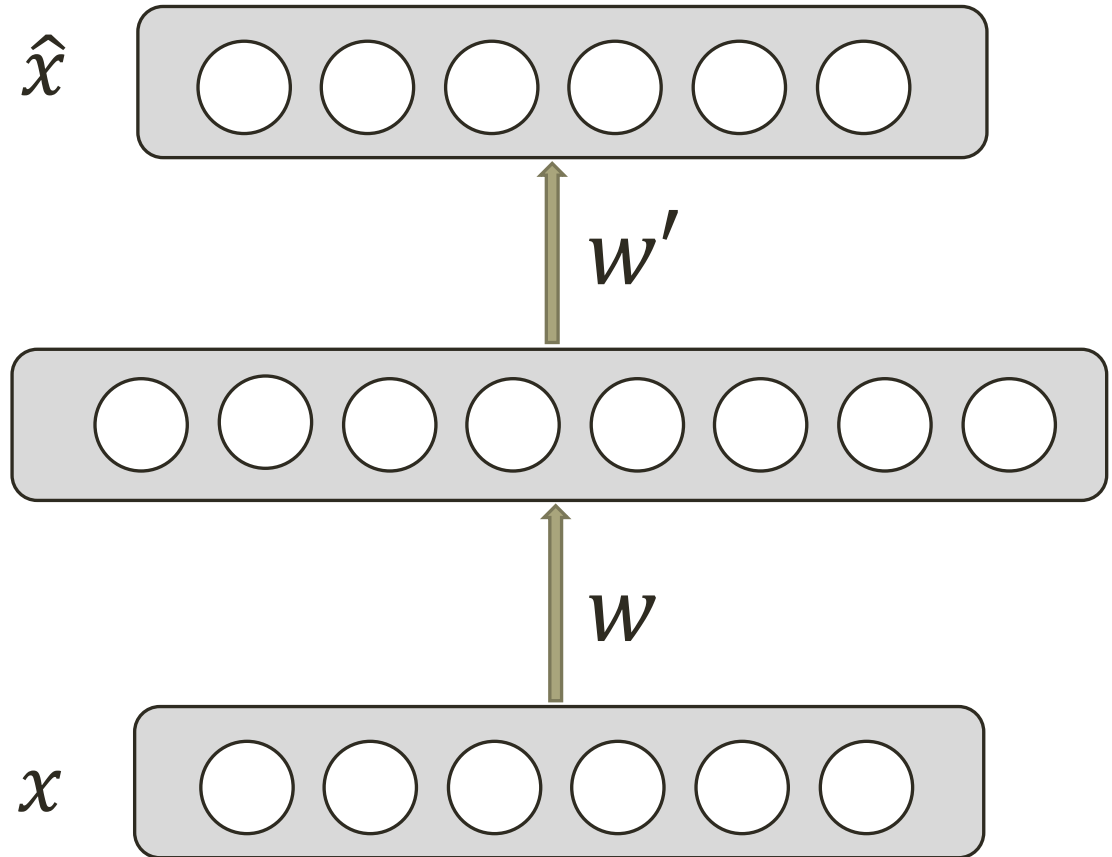
# CONTRACTIVE AUTOENCODERS

- We are still trying to avoid uninteresting features.
- Here we add a regularization term  $\Omega(x)$  to our loss function to limit the hidden layer.



# CONTRACTIVE AUTOENCODERS

- Idea: We wish to extract features that **only** reflect variations observed in the training set. We would like to be invariant to the other variations.
- Points close to each other in the input space maintain that property in the latent space.



# CONTRACTIVE AUTOENCODERS

Definitions and reminders:

- Frobenius norm (L2):  $\|A\|_F =$

$$\sqrt{\sum_{i,j} |a_{ij}|^2}$$

- Jacobian Matrix:  $J_f(x) = \frac{\partial f(x)}{\partial x} = \begin{bmatrix} \frac{\partial f(x)_1}{\partial x_1} & \dots & \frac{\partial f(x)_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f(x)_m}{\partial x_1} & \dots & \frac{\partial f(x)_m}{\partial x_n} \end{bmatrix}$

# CONTRACTIVE AUTOENCODERS

Our new loss function would be:

$$L^*(x) = L(x) + \lambda \Omega(x)$$

where  $\Omega(x) = \|J_f(x)\|_F^2$  or simply:  $\sum_{i,j} \left( \frac{\partial f(x)_j}{\partial x_i} \right)^2$

and where  $\lambda$  controls the balance of our reconstruction objective and the hidden layer “flatness”.



# CONTRACTIVE AUTOENCODERS

Our new loss function would be:

$$L^*(x) = L(x) + \lambda \Omega(x)$$

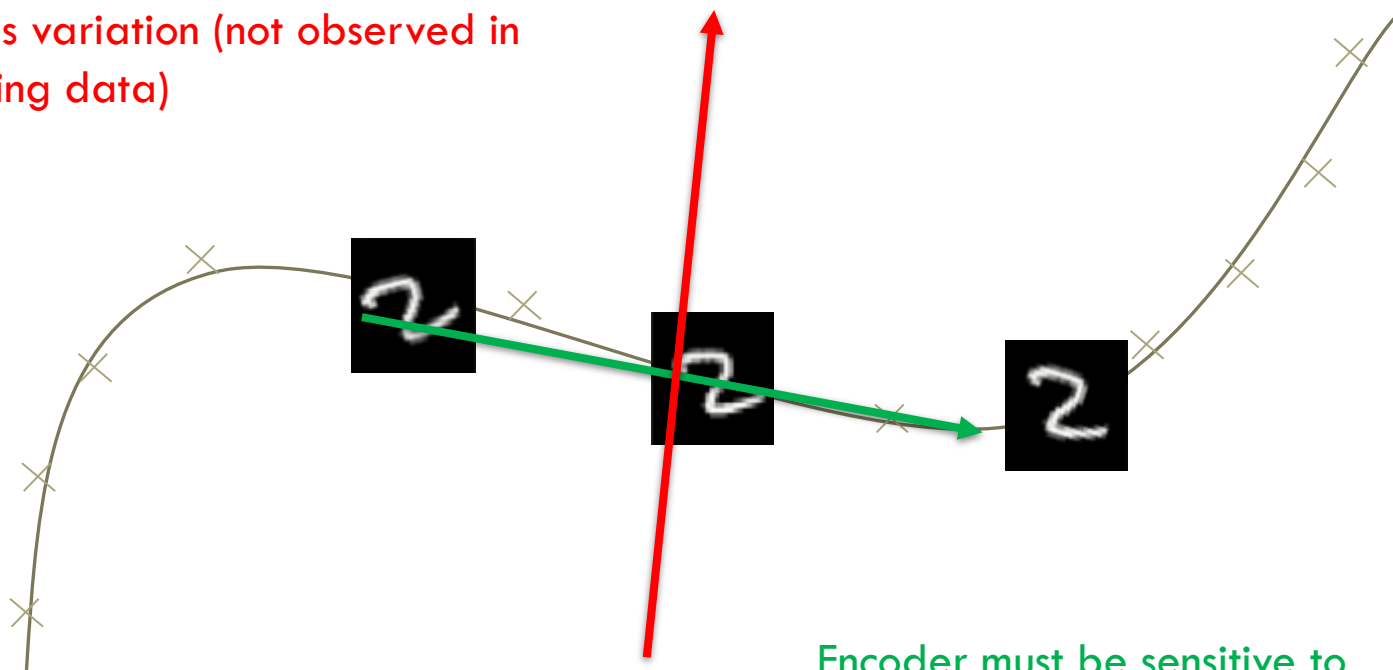
$L(x)$  - would be an encoder that keeps good information ( $\lambda \rightarrow 0$ )

$\Omega(x)$  - would be an encoder that throws away all information ( $\lambda \rightarrow \infty$ )

Combination  
would be an  
encoder that  
keeps **only** good  
information.

# CONTRACTIVE AUTOENCODERS

Encoder doesn't need to be sensitive to this variation (not observed in training data)



Encoder must be sensitive to this variation to reconstruct well

# WHICH AUTOENCODER?

- DAE make the **reconstruction function** resist small, finite sized perturbations in input.
- CAE make the **feature encoding function** resist small, infinitesimal perturbations in input.
- Both denoising AE and contractive AE perform well!

# WHICH AUTOENCODER?

- Advantage of DAE: simpler to implement
  - Requires adding one or two lines of code to regular AE.
  - No need to compute Jacobian of hidden layer.
- Advantage of CAE: gradient is deterministic.
  - might be more stable than DAE, which uses a sampled gradient.
  - one less hyper-parameter to tune (noise-factor)

# REFERENCES

1. <https://arxiv.org/pdf/1206.5538.pdf>
2. <http://www.deeplearningbook.org/contents/autoencoders.html>
3. <http://deeplearning.net/tutorial/dA.html>
4. <http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/>
5. [http://ufldl.stanford.edu/wiki/index.php/Stacked\\_Autoencoders](http://ufldl.stanford.edu/wiki/index.php/Stacked_Autoencoders)
6. <http://www.jmlr.org/papers/volume11/vincent10a/vincent10a.pdf>
7. <https://codeburst.io/deep-learning-types-and-autoencoders-a40ee6754663>



**THANK YOU**