Unsupervised Learning and Auto-Encoders

Neural Computation Course 2022

Konstantinos Kamnitsas

Intro to Unsupervised Learning

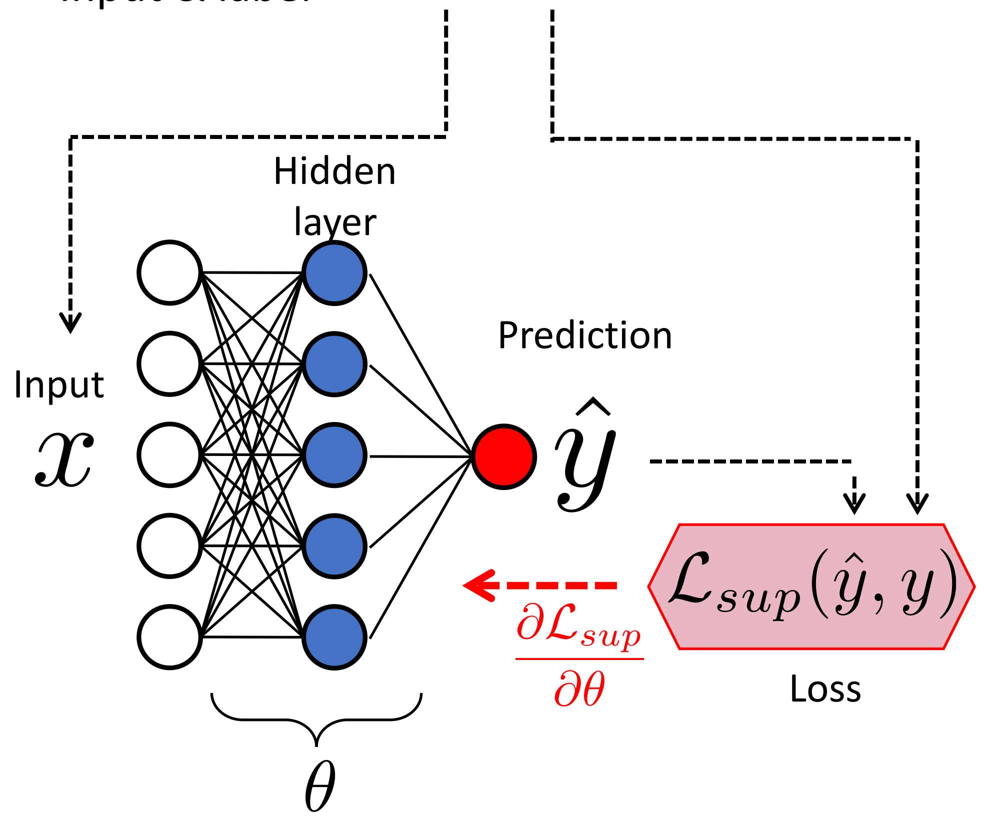
Neural Computation Course 2022

Konstantinos Kamnitsas

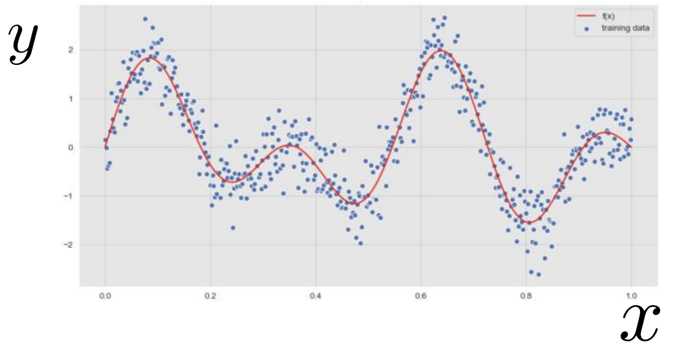
Previously: Supervised Learning

Goal: learn a function to map $f_\theta : \mathcal{X} \rightarrow \mathcal{Y}$

Given for training: $(x_1, y_1), \ldots, (x_N, y_N) \sim p_{data}(x, y)$
Input & label

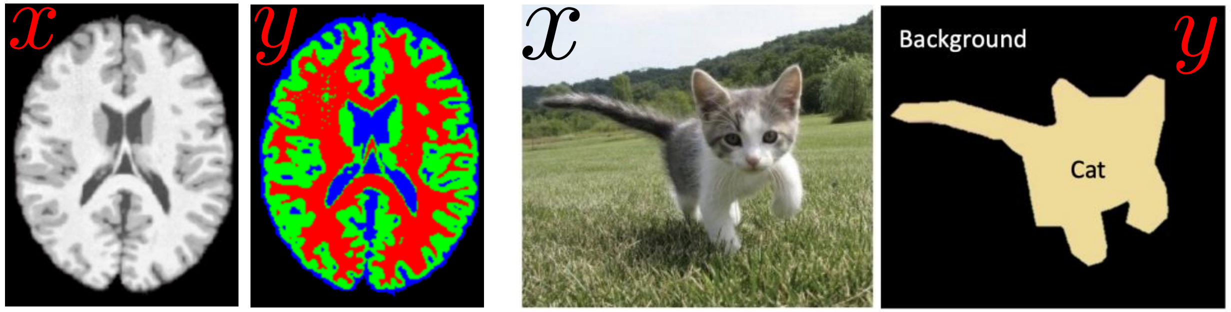


Regression



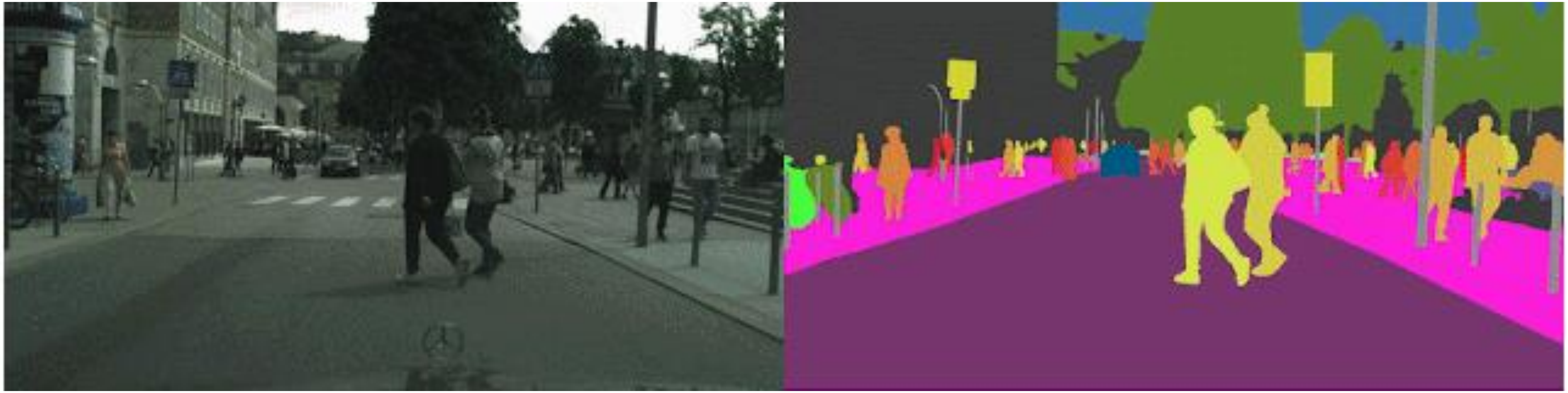
Classification

Segmentation



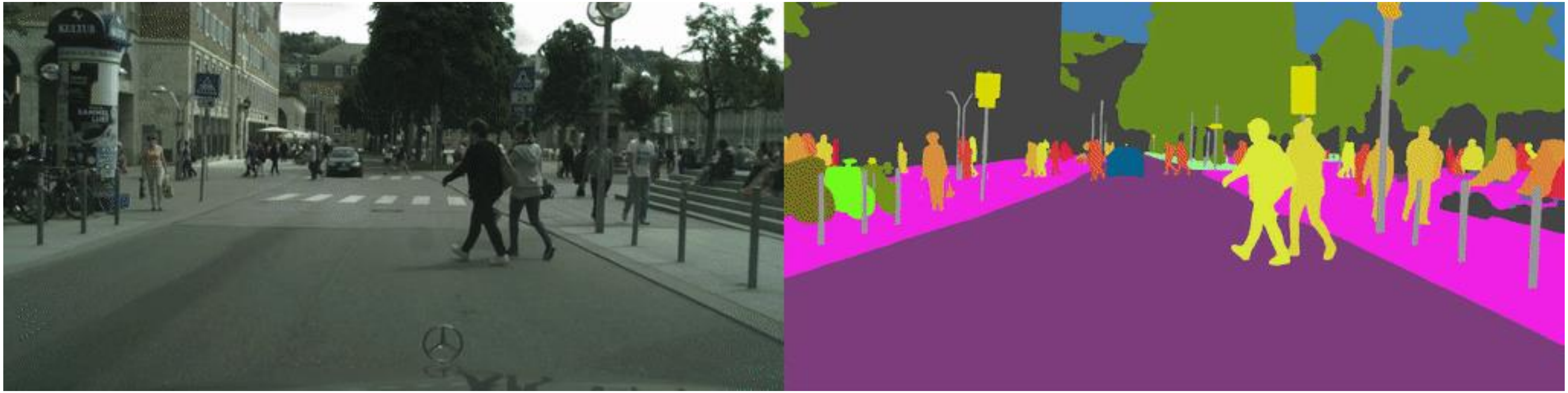
Creating labelled data is challenging

x y



Creating labelled data is challenging

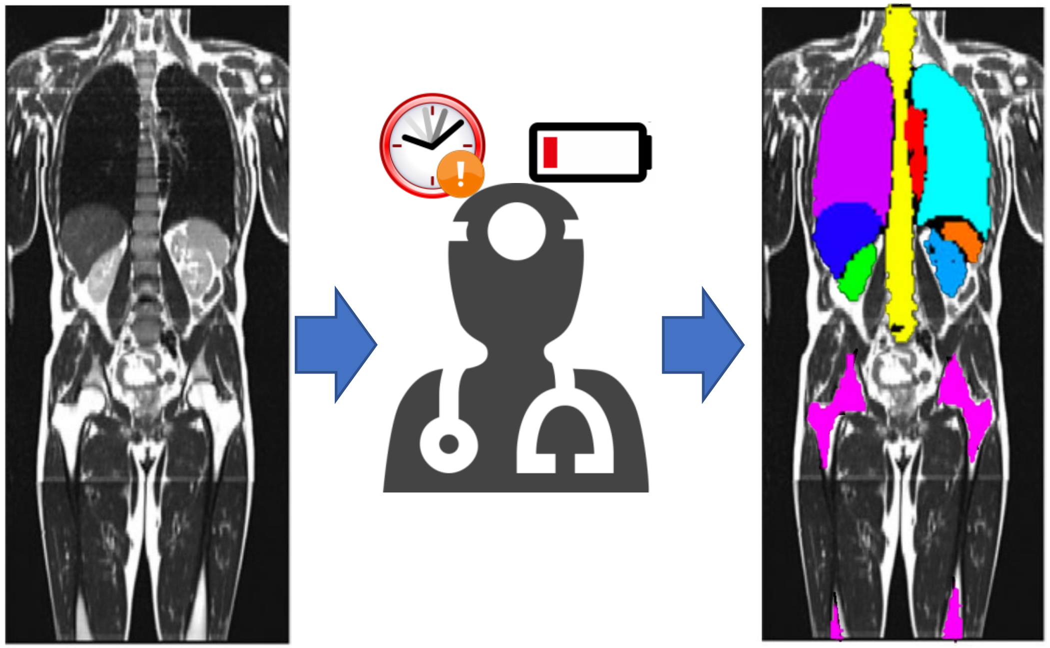
x y

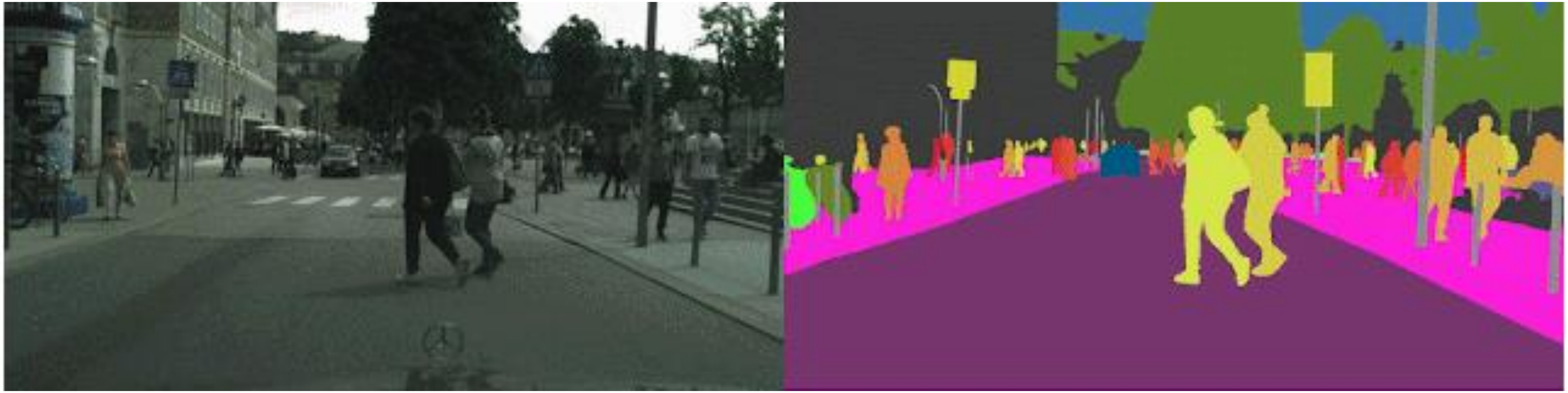


Labelled data are limited

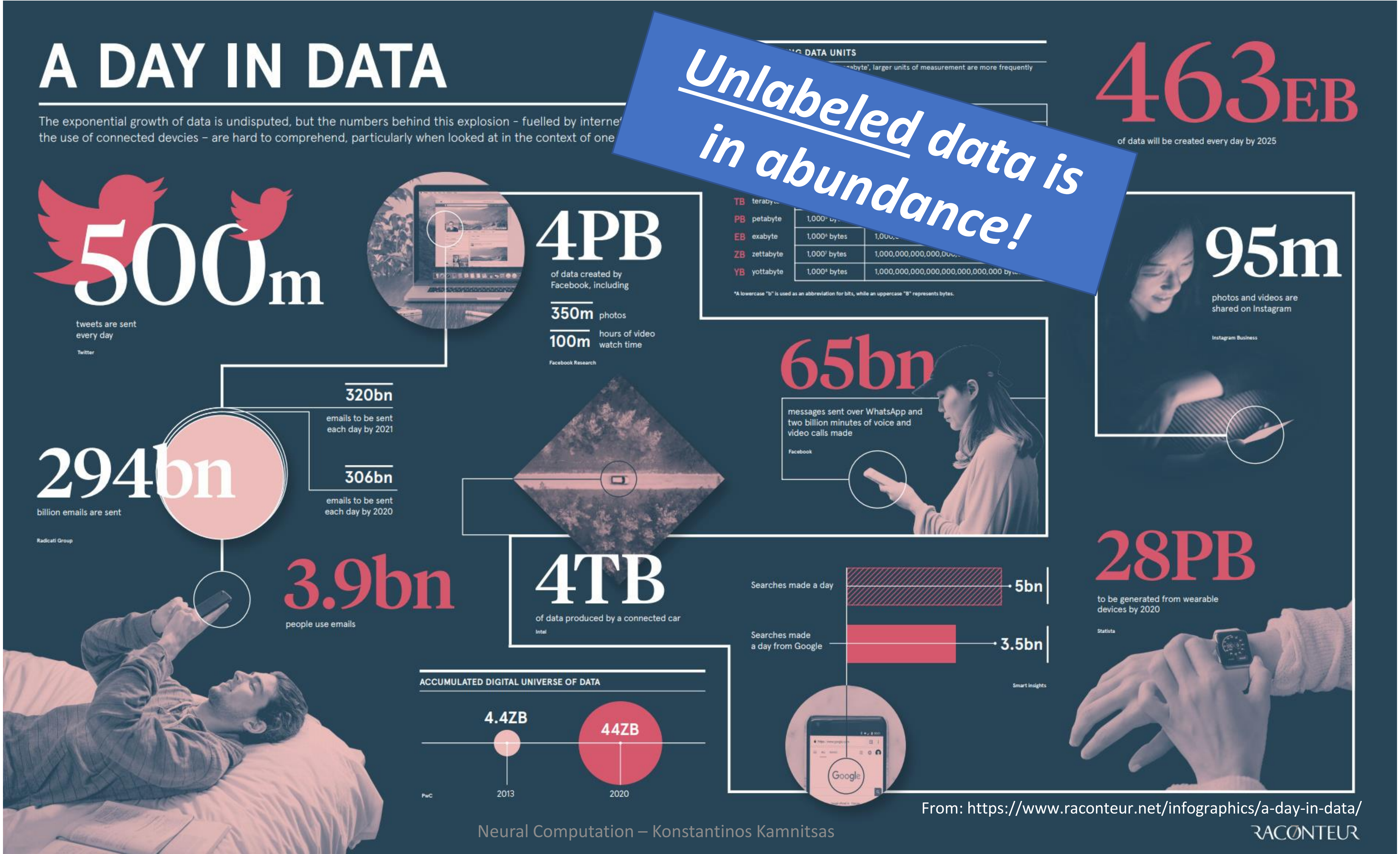
Creating labelled data requires:
- Time consuming manual work
- Humans are expensive
- Labelling may require expertise (e.g. healthcare)

Consequence: Labelled databases are small





Unlabeled data is in abundance!



From: https://www.raconteur.net/infographics/a-day-in-data/

Unsupervised Learning

Available data: $x_1, ..., x_N \sim p_{data}(x)$

Goal: Learn "**useful features**" of the data / Learn "**the structure**" of the data.

Unsupervised Learning

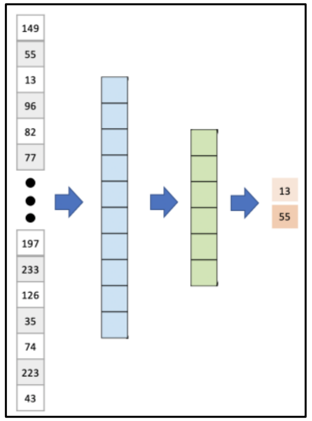
Available data: $x_1, \ldots, x_N \sim p_{data}(x)$

Goal: Learn "**useful features**" of the data / Learn "**the structure**" of the data.

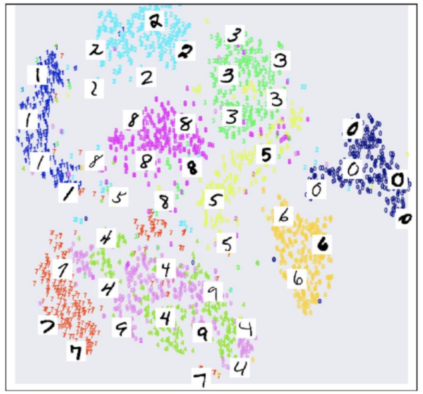
Useful for:
- Dimensionality Reduction (compression)
- Clustering
- Probability Density Estimation
- Generation / Synthesis
- Learn from loads unlabeled data, when labeled data are limited
- …

Generation / Synthesis

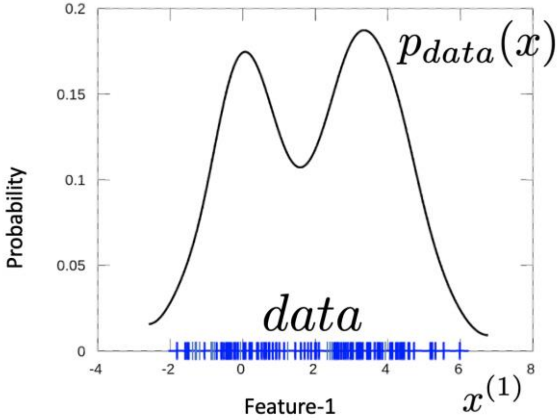
Dimensionality Reduction



Clustering



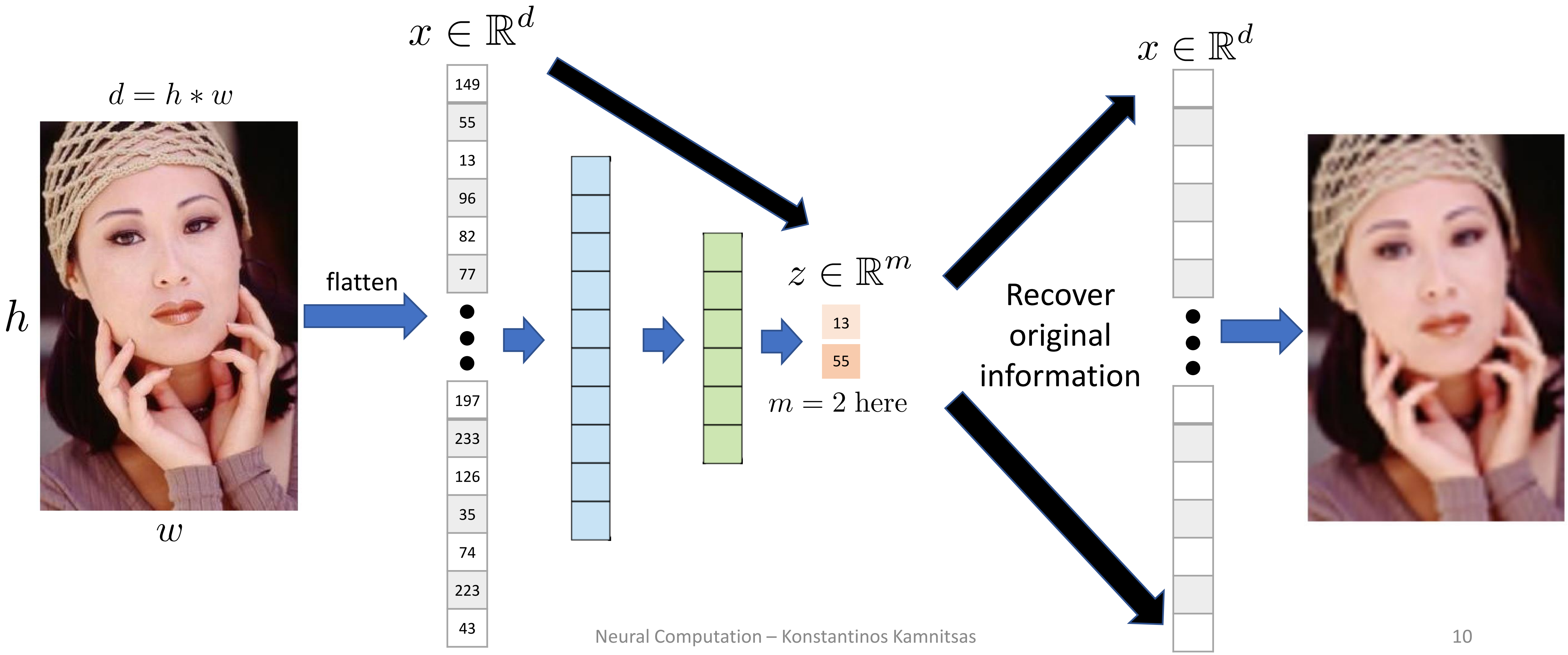
Probability Density Estimation



Dimensionality Reduction / Compression

Learn $f : \mathbb{R}^d \rightarrow \mathbb{R}^m$, where $d > m$

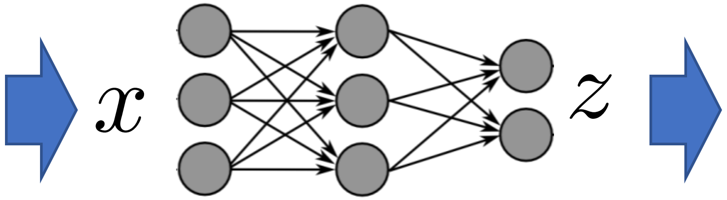
Requirement:
Preserve important information

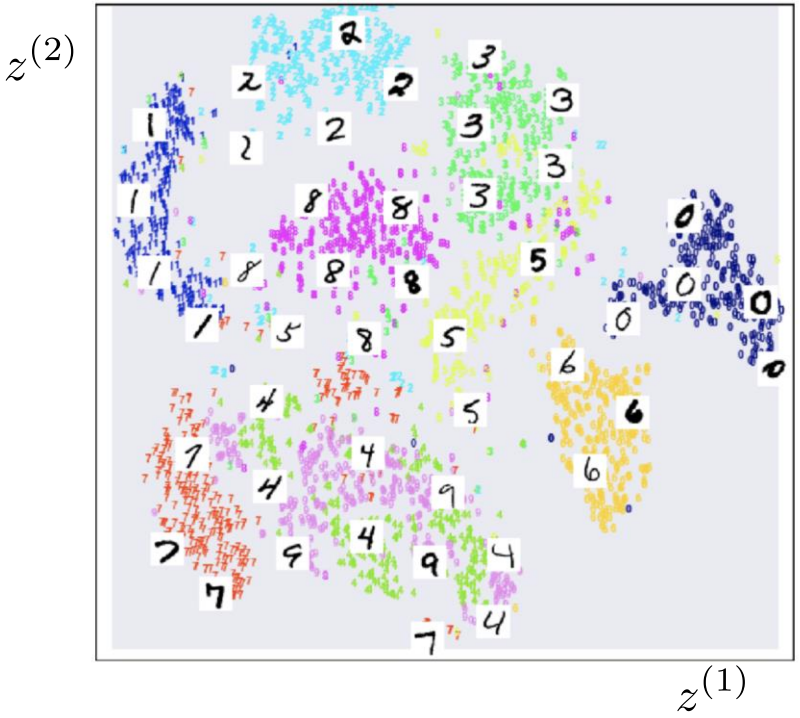


Clustering

What is clustering: Discover or form groups of samples based on their similarity. Similar samples should be grouped together, dissimilar samples should be separated.

Example: Assume 10 “classes” of digits. Learn a function (model) that embeds (maps) samples of the same class to similar values (features), whereas samples of different classes are embedded to different values (features)





Clustering

What is clustering: Discover or form groups of samples based on their similarity. Similar samples should be grouped together, dissimilar samples should be separated.

Example: Assume 10 “classes” of digits. Learn a function (model) that embeds (maps) samples of the same class to similar values (features), whereas samples of different classes are embedded to different values (features)

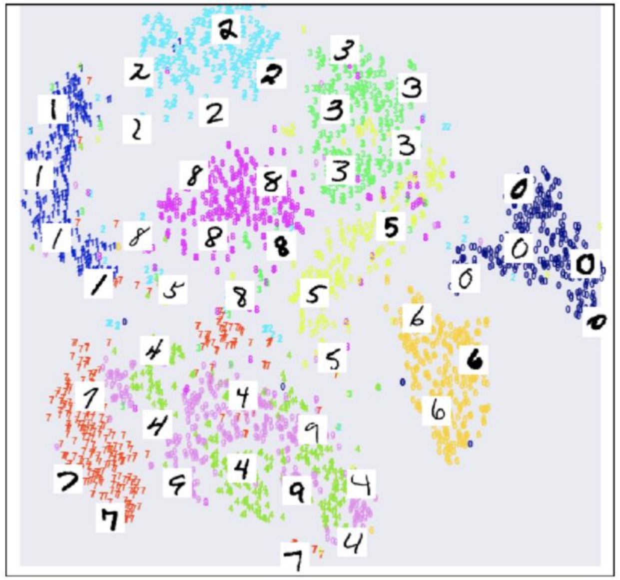
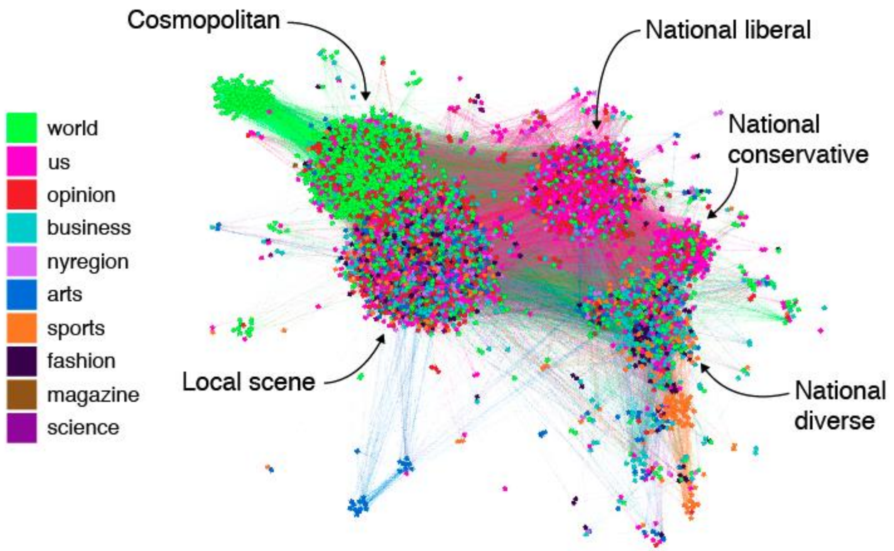
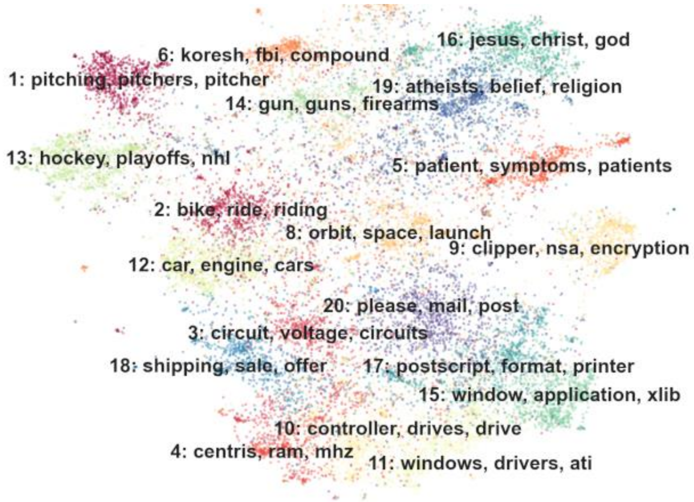


Image (digit) analysis



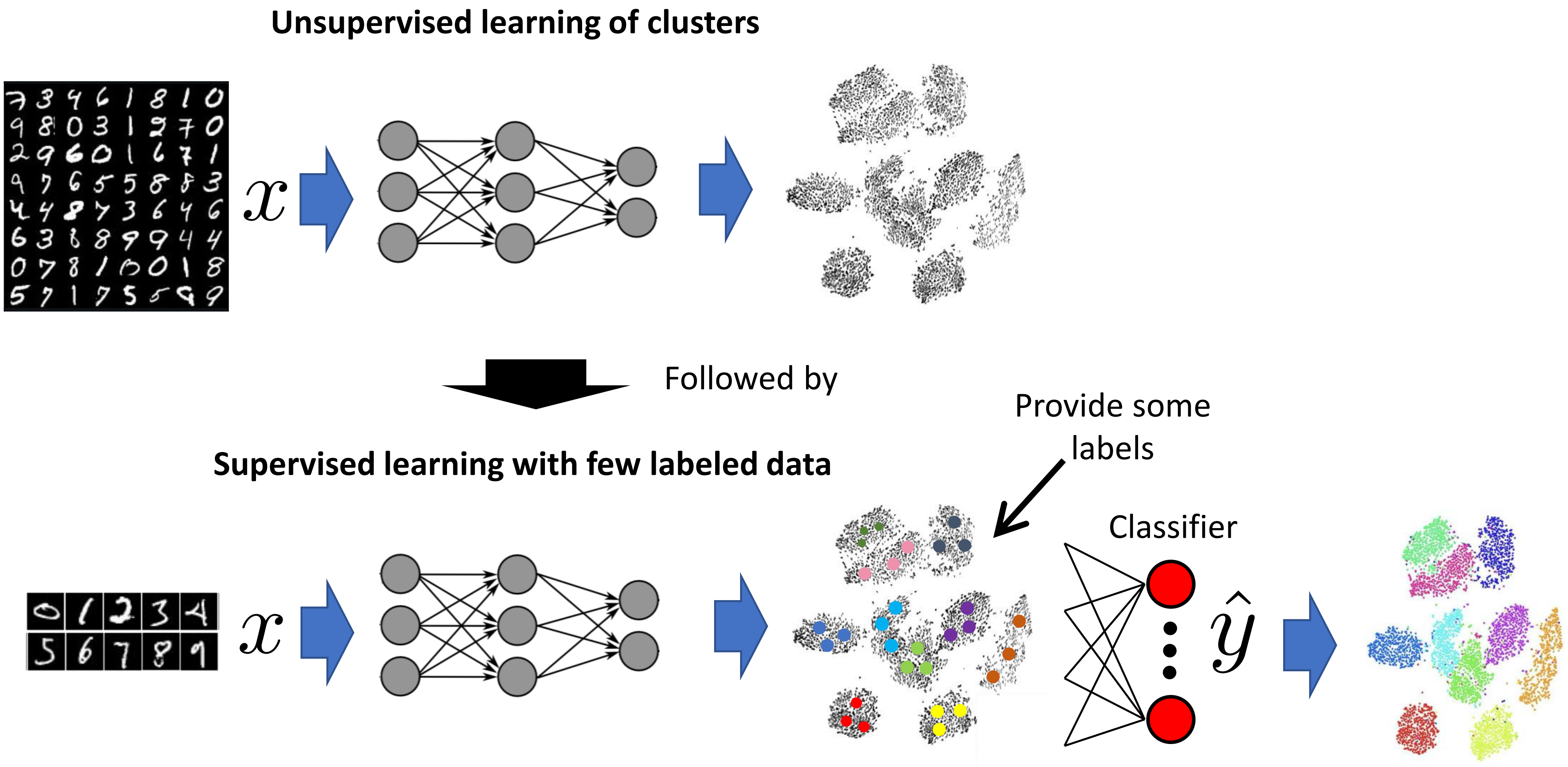
From: www.zdnet.com/article/new-study-maps-social-identity-clusters-on-twitter/

Social media users



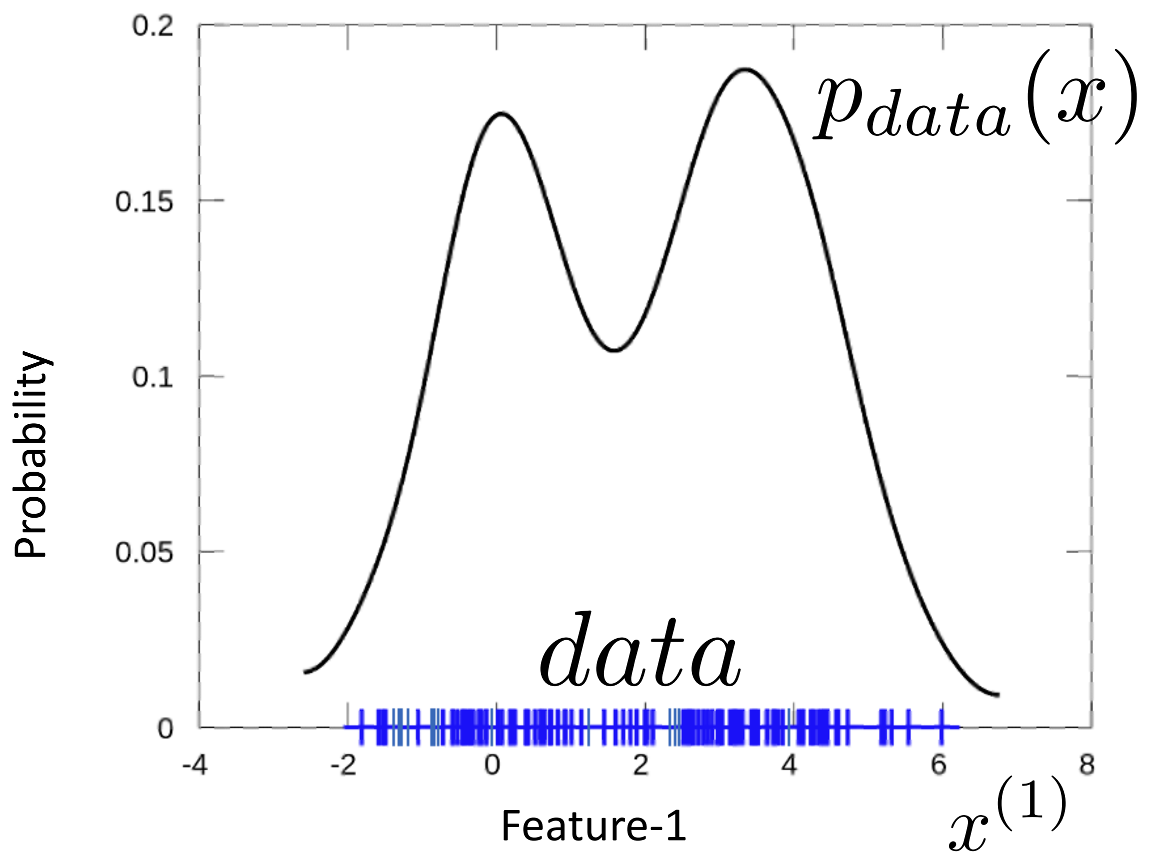
Text analysis

Learn from **unlabeled data** to improve Supervised Learning **when labelled data are limited**



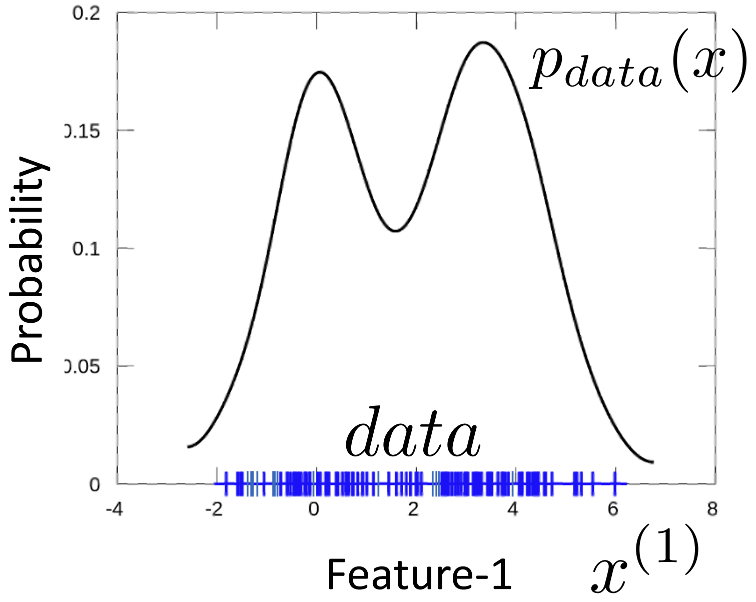
Probability Density Estimation

1-dimensional data:

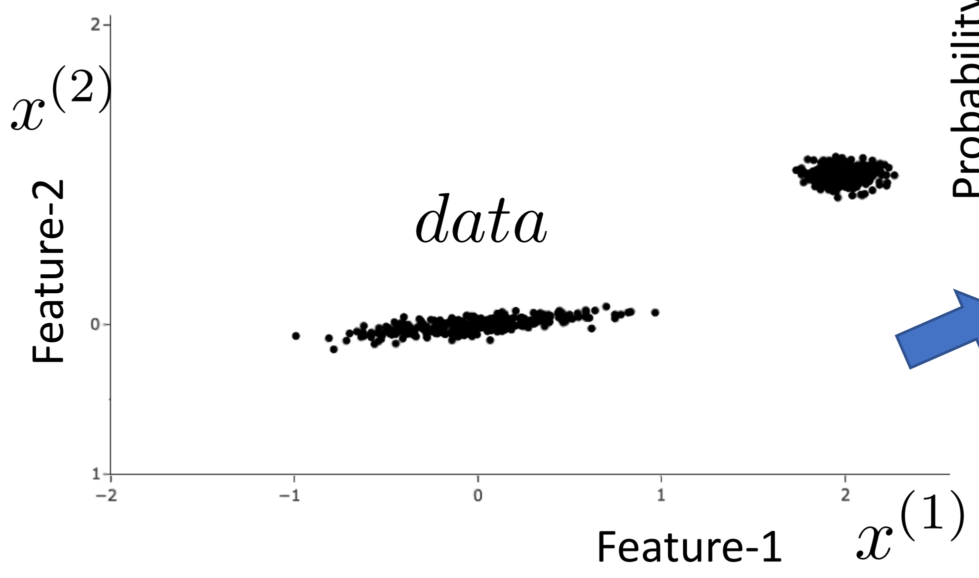


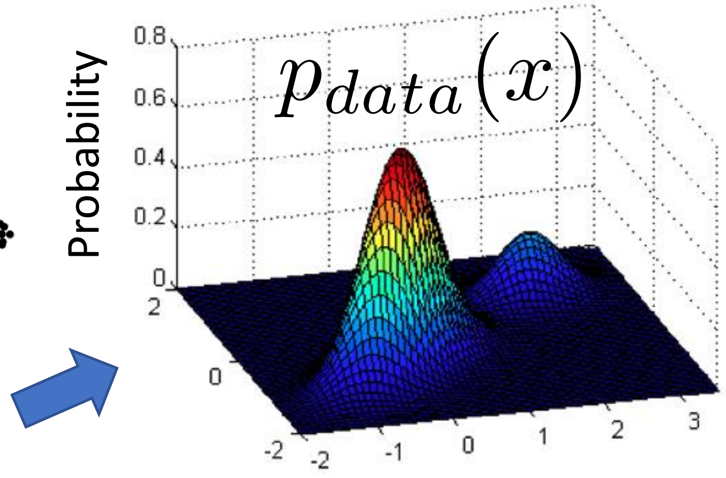
Probability Density Estimation

1-dimensional data:



2-dimensional data:





Generation / Synthesis

https://thispersondoesnotexist.com/

Generation / Synthesis

State of the Art 2014

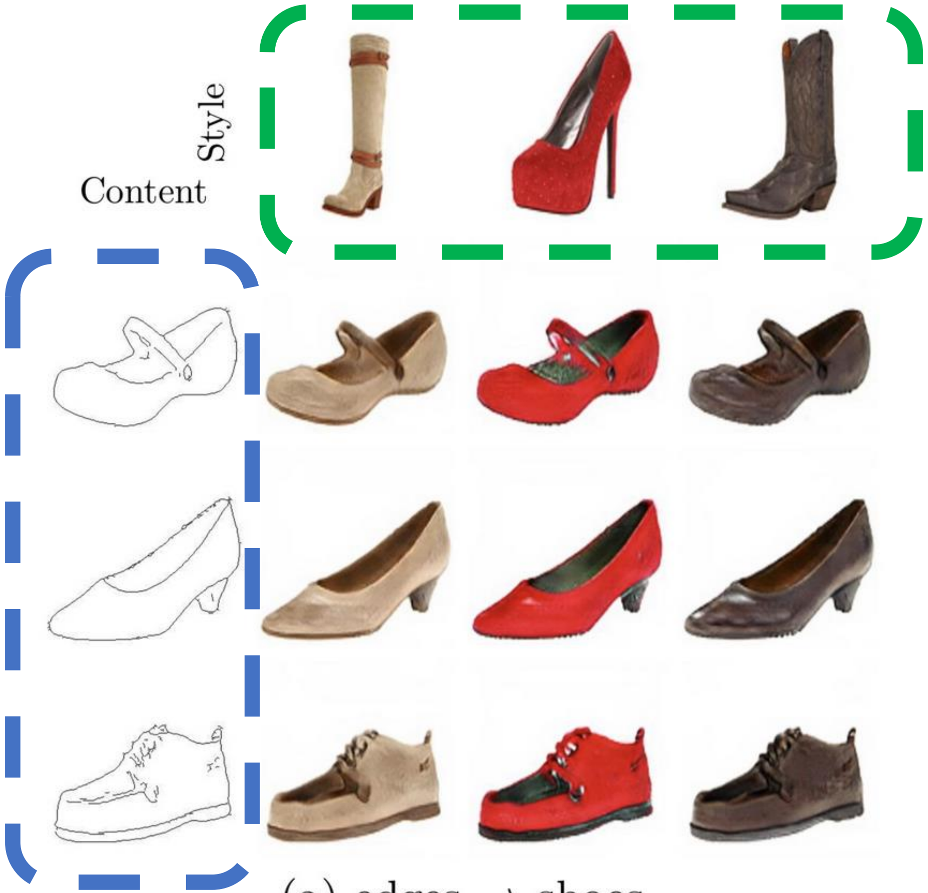
State of the Art 2018

Generation / Synthesis

Karas et al, StyleGAN2, 2019

Synthesis with Style Transfer

Example-guided Image Translation



(a) edges → shoes

Huang et al, Multimodal Unsupervised Image-to-Image Translation, 2018
Huang et al, Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization, 2017

Style Transfer

Winter — Real Image / Synthetic — Summer

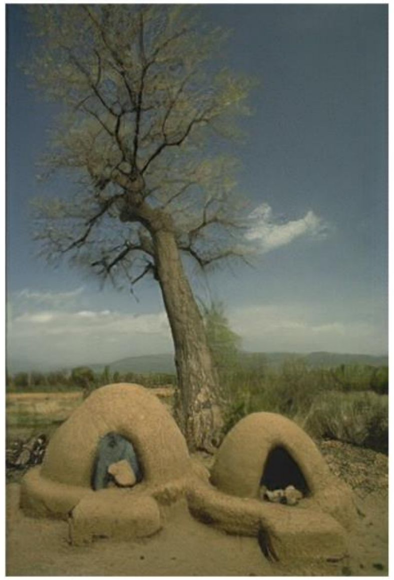
Summer — Real Image / Synthetic — Winter

Huang et al, Multimodal Unsupervised Image-to-Image Translation, 2018
Huang et al, Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization, 2017

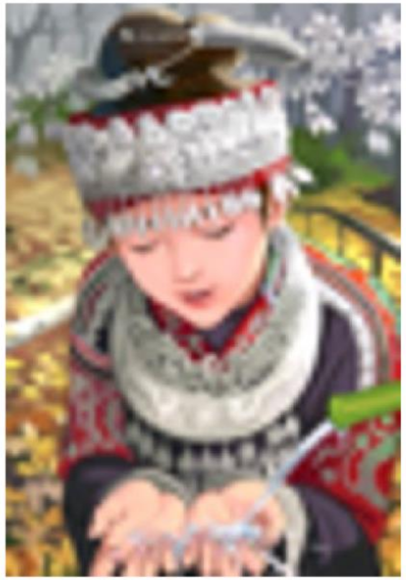
Image Enhancement

Down-sampled

GAN enhancement



Down-sampled



GAN enhancement

Down-sampled

GAN enhancement

Ledig et al, CVPR 2017

How do we learn from unlabeled data?

Next part:

Auto-Encoders (1)

Auto-Encoders (Part 1)

Neural Computation Course 2022

Konstantinos Kamnitsas

What we will learn about Auto-Encoders:

Part 1 (this part):

- What is the basic auto-encoder
- How to train auto-encoders
- What is an AE with a bottleneck layer and why to use it
- What features do they learn

Part 2:

- What can they be used for (dimensionality reduction, clustering, pretraining,…)
- What AEs are not good at and why

Supervised learning:
From observations (x) to labels (y), given labeled data

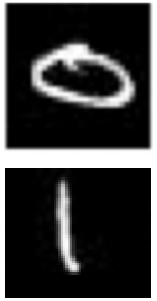
$x \in \mathcal{X}$

Observed variable: Because we see this data, both at training and at testing. E.g. an image, or the vector representing a data point.

$y \in \mathcal{Y}$

Labeled variable: Used for variables for which a human created manual annotations. Usually the prediction **target** for **supervised** learning.

Observation: **Label:**



"zero"

"one"

Unsupervised Learning:
From observations (x) to latent variables (z), without labeled data

$$x \in \mathcal{X}$$

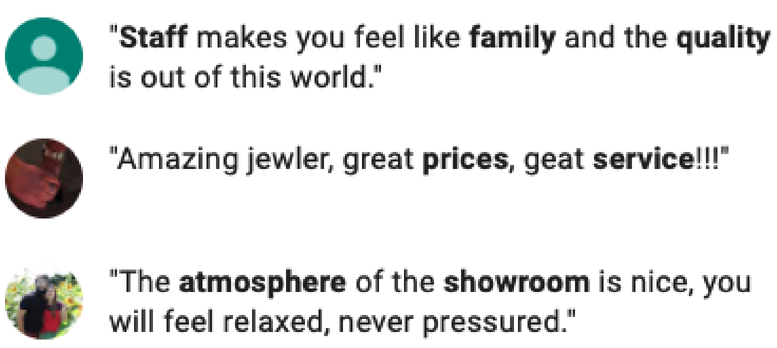
Observed variable: Because we see this data, both at training and at testing.
E.g. an image, or the vector representing a data point.

$$z \in \mathcal{Z}$$

Latent variable: We do not know this information.
Given x, we **have to infer it**.

z: label of digit, thickness, slope …
x: image of digit

z: content, lighting angle, zoom …
x: the photo



z: sentiment, vocabulary, grammar skills …
x: written review

Example of latent (vector) representation

Observed variable

$x =$ Input vector

$x_1 =$  w, h $= \begin{bmatrix} x_1^{(1)} \\ x_1^{(2)} \\ \vdots \\ x_1^{(w*h)} \end{bmatrix}$

$x_2 =$  $= \begin{bmatrix} x_2^{(1)} \\ x_2^{(2)} \\ \vdots \\ x_2^{(w*h)} \end{bmatrix}$

Latent variable

$$z = \begin{bmatrix} z^{(1)} \\ z^{(2)} \\ z^{(3)} \\ z^{(4)} \\ \vdots \\ z^{(v)} \end{bmatrix} = \begin{bmatrix} \text{“}blonde\text{”} \\ \text{“}smiling\text{”} \\ \text{“}looks\_left\text{”} \\ \text{“}looks\_right\text{”} \\ \vdots \\ \text{“}wears\_suit\text{”} \end{bmatrix}$$

$$z_1 = \begin{bmatrix} z_1^{(1)} \\ z_1^{(2)} \\ z_1^{(3)} \\ z_1^{(4)} \\ \vdots \\ z_1^{(v)} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

$$z_2 = \begin{bmatrix} z_2^{(1)} \\ z_2^{(2)} \\ z_2^{(3)} \\ z_2^{(4)} \\ \vdots \\ z_2^{(v)} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

How to learn useful latent representation Z of data?

$$f_\phi : \mathcal{X} \rightarrow \mathcal{Z}$$

…given only input samples x…

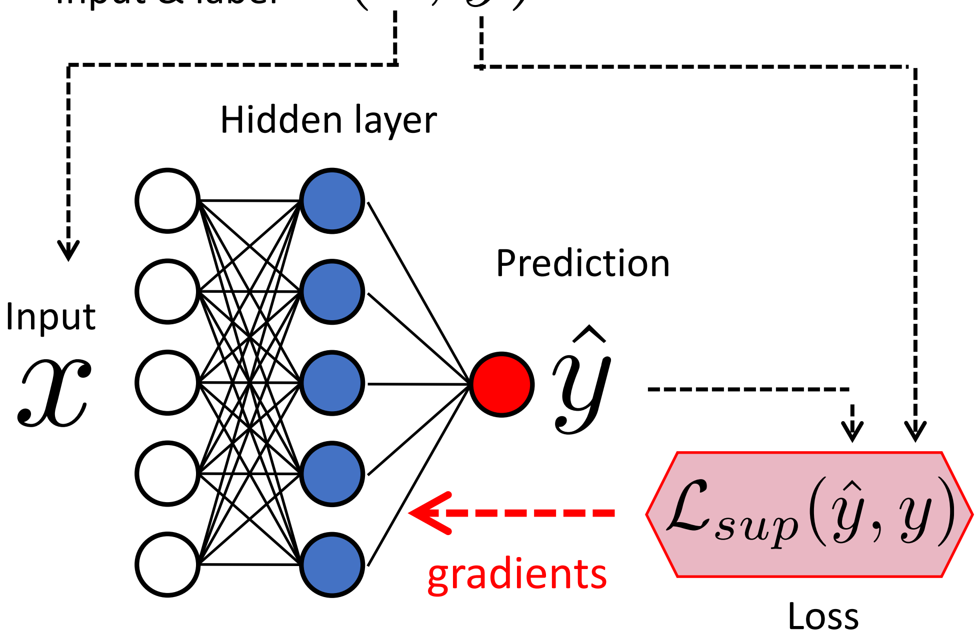
$$z = \begin{bmatrix} \text{“}blonde\text{”} \\ \text{“}smiling\text{”} \\ \text{“}looks\_left\text{”} \\ \text{“}looks\_right\text{”} \\ \vdots \\ \text{“}wears\_suit\text{”} \end{bmatrix}$$

$$z = \begin{bmatrix} \text{“}slope\text{”} \\ \text{“}thickness\text{”} \\ \vdots \\ \text{“}is\_circular\text{”} \end{bmatrix}$$

How to learn useful latent representation (code) of data?

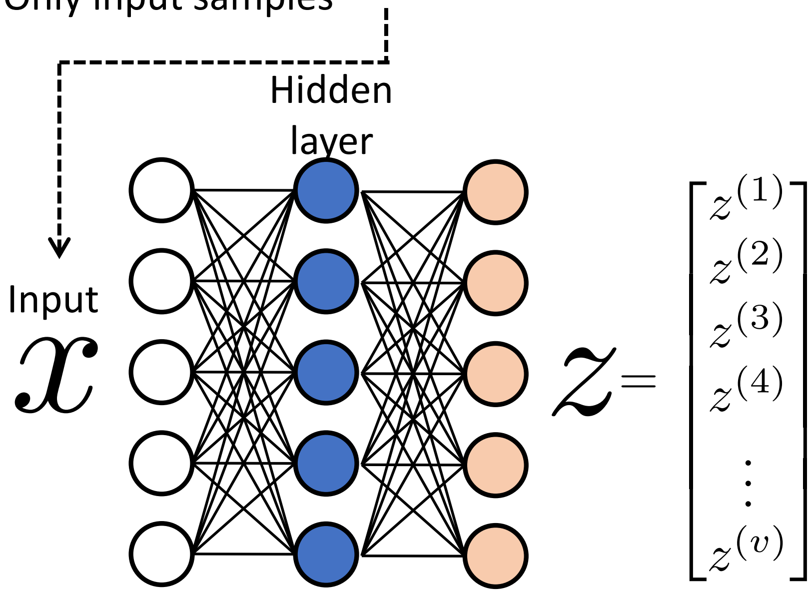
Supervised Classifier: $\mathcal{X} \rightarrow \mathcal{Y}$

Given for training: Input & label (x, y)



Unsupervised <u>Encoder</u>: $\mathcal{X} \rightarrow \mathcal{Z}$

Given for training: Only input samples x



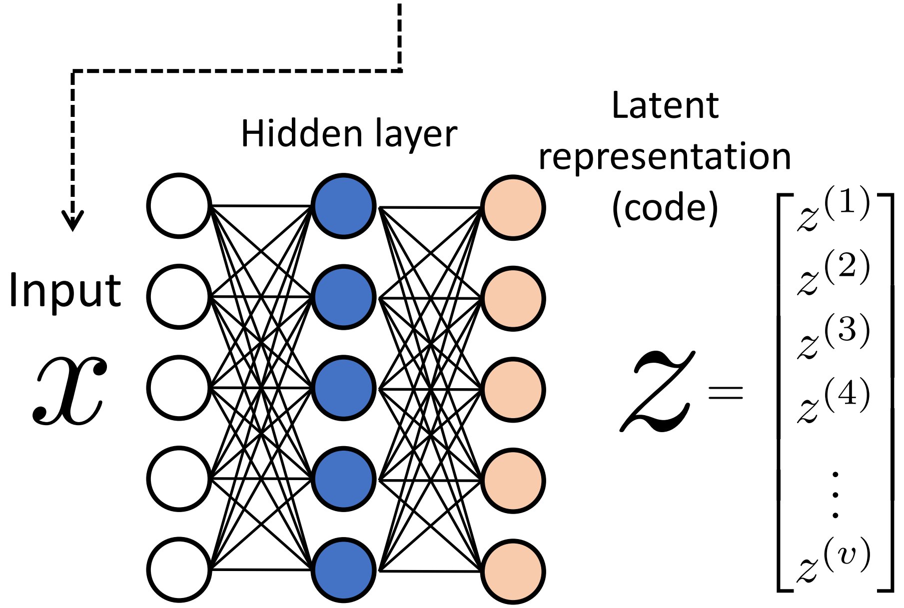
<u>How do we train</u> such an encoder?

<u>What Loss?</u>

How to learn useful latent representation (code) of data?

Unsupervised <u>Encoder</u>: $f_\phi : \mathcal{X} \rightarrow \mathcal{Z}$

Given for training: x



<u>Main idea:</u>

*What **property** do **we require** for latent representation z ?*

To represent x

To **preserve useful info** about x

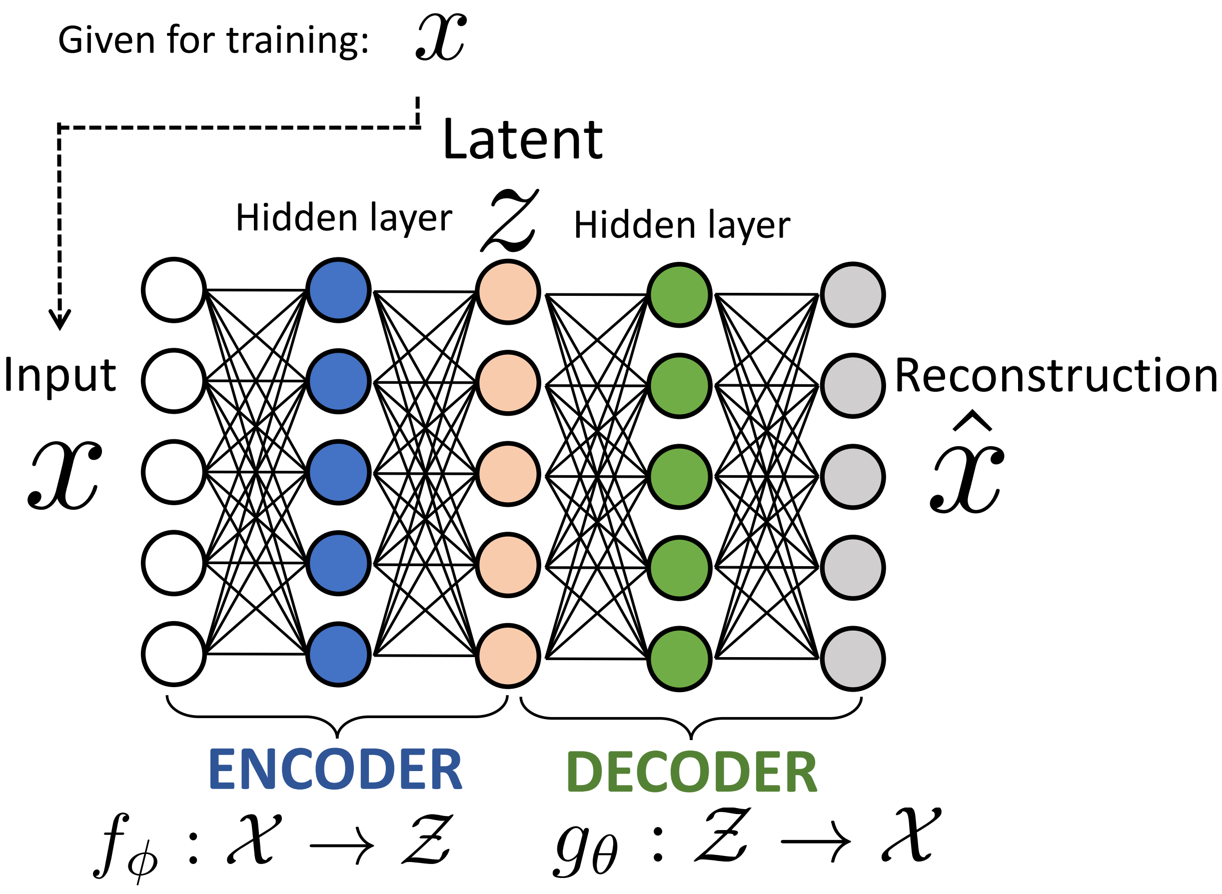
How can we enforce that?!?!

Ensure we can also learn

$$g_\theta : \mathcal{Z} \rightarrow \mathcal{X}$$

How to learn useful latent representation (code) of data?

Unsupervised Auto-Encoder (AE):



Parameters: ϕ θ

Encoder:

Takes input x and encodes it to $z = f_\phi(x)$

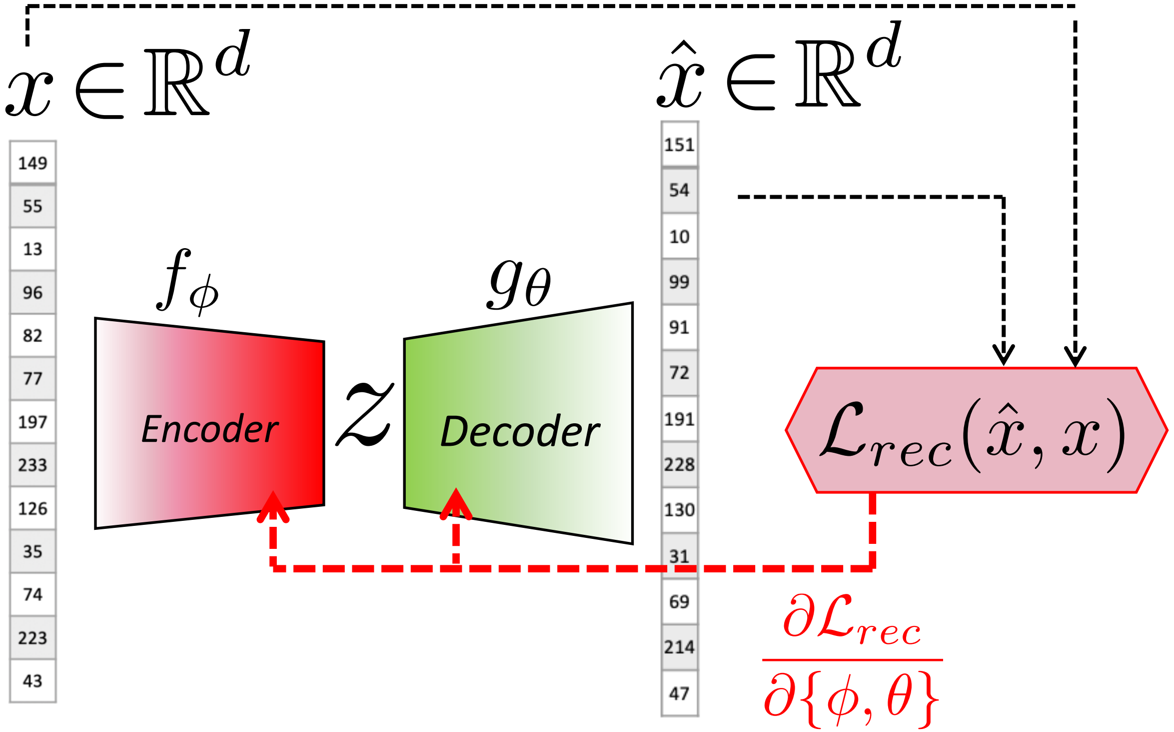
Decoder:

Takes code z and decodes it to re-construct $\hat{x} = g_\theta(f_\phi(x))$

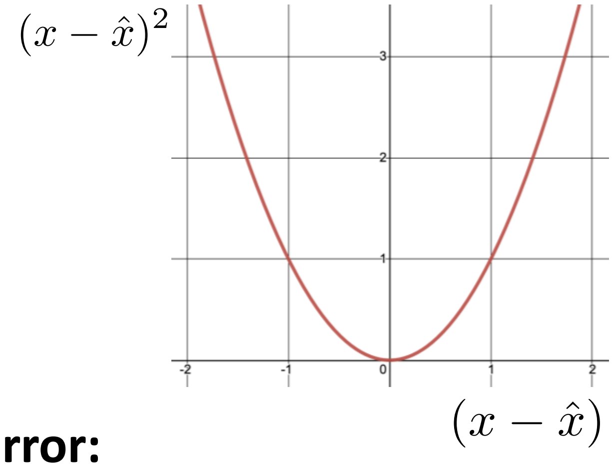
Result:

Good reconstruction $(\hat{x} \approx x)$ is possible only if *code Z preserves info about X*

Training loss



$$\phi', \theta' = arg \min_{\phi,\theta} \mathcal{L}_{rec}$$



Reconstruction Loss
a.k.a. Mean Squared Error:

$$\mathcal{L}_{rec} = \frac{1}{d}\sum_{j=1}^{d}(x^{(j)} - \hat{x}^{(j)})^2$$

$$= \frac{1}{d}\sum_{j=1}^{d}\left(x^{(j)} - g_\theta^{(j)}(f_\phi(x))\right)^2$$

Only one Global Optimum:

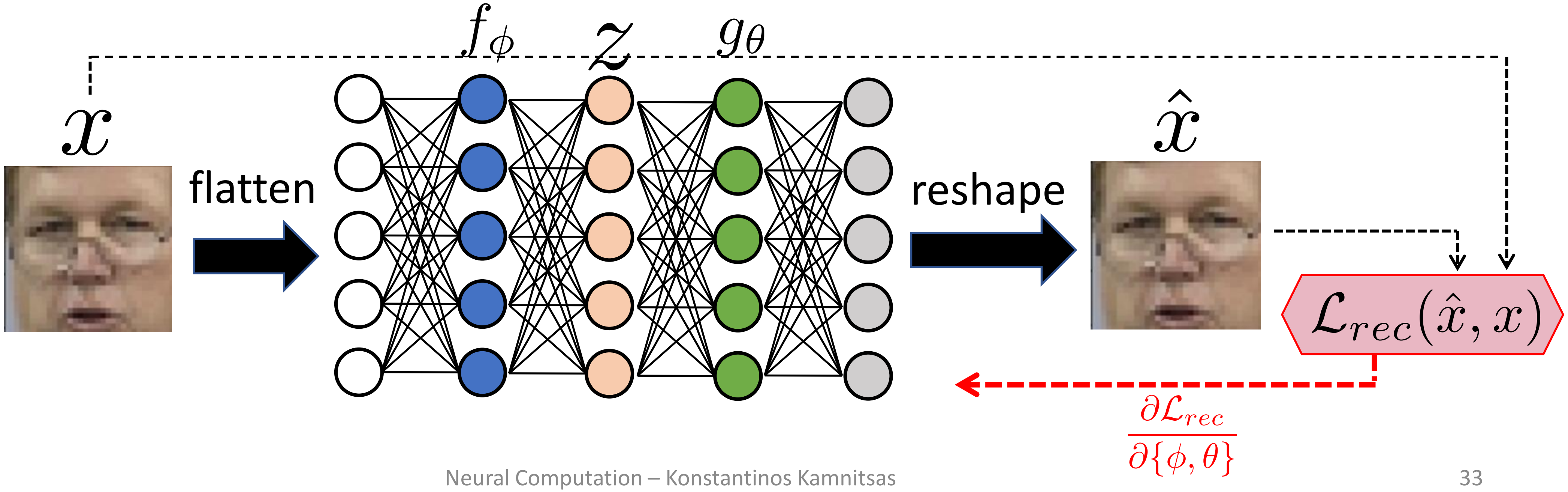
$$\mathcal{L}_{rec} = 0 \Leftrightarrow \hat{x}^{(j)} - x^{(j)} = 0 \;\; \forall j$$

$$\Leftrightarrow \boxed{\hat{x} = x}$$

Learning to reconstruct

The loss is minimized when: $\hat{x} = x \Rightarrow \underbrace{g_\theta(f_\phi(x))}_{} = x$

The Auto-Encoder tries to learn the <u>**identity function**</u>!



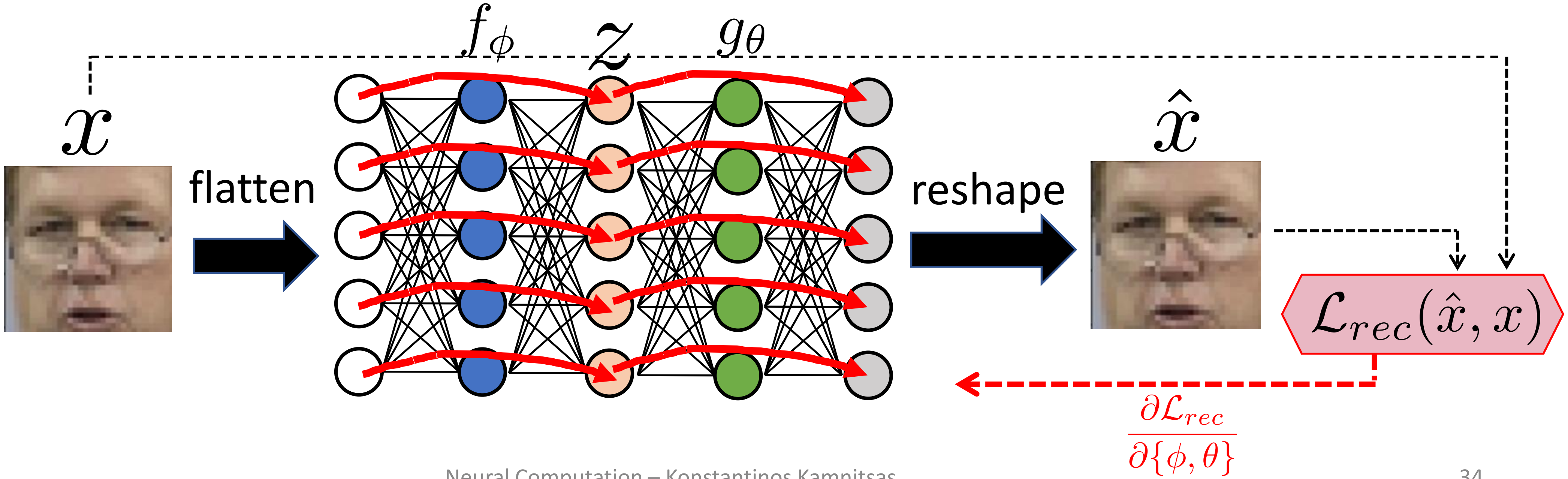
Problem: Trivial Solution / Learning a useless function

The loss is minimized when: $\hat{x} = x \Rightarrow \underbrace{g_\theta(f_\phi(x))}_{} = x$

The Auto-Encoder tries to learn the **identity function**!

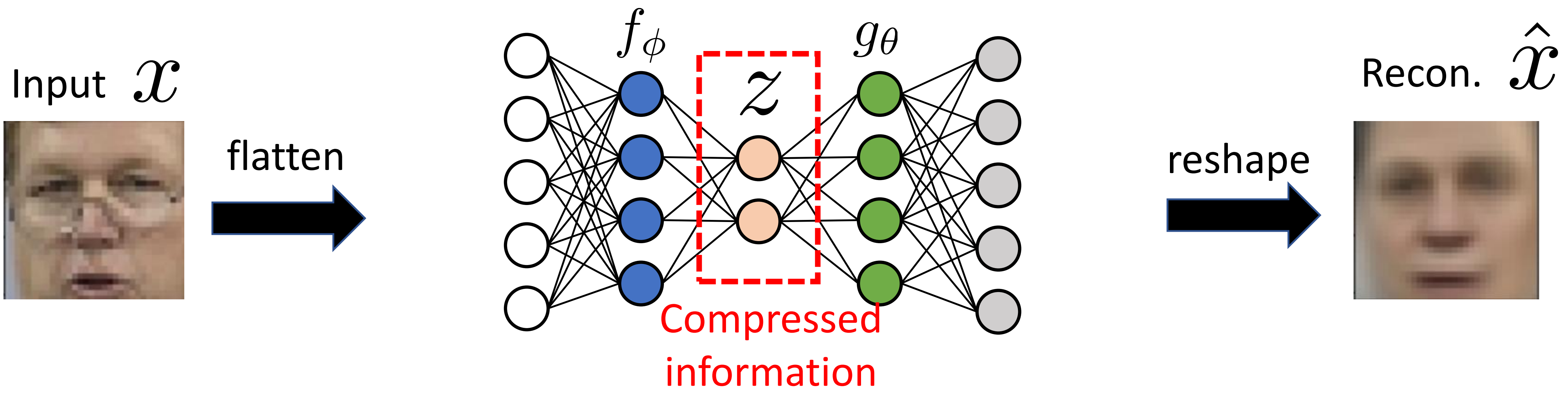
Exists ***trivial solution (and useless model)***:

$$z = f_\phi(x) = x$$
$$\hat{x} = g_\theta(z) = z$$



Solution: ***Bottleneck*** layer

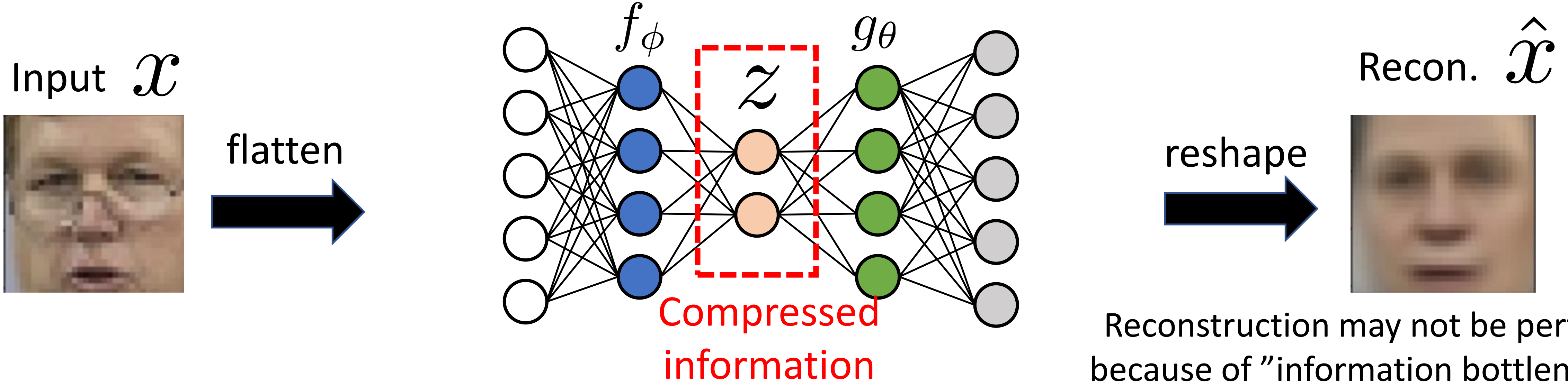
$$f_\phi : \mathcal{X} \in \mathbb{R}^d \rightarrow \mathcal{Z} \in \mathbb{R}^v, \text{ where } v < d$$



”Standard/basic” Auto-Encoder

$$f_\phi : \mathcal{X} \in \mathbb{R}^d \rightarrow \mathcal{Z} \in \mathbb{R}^v, \text{ where } v < d$$

With “standard/basic” Auto-Encoder,
in this class we refer to
AE with Bottleneck

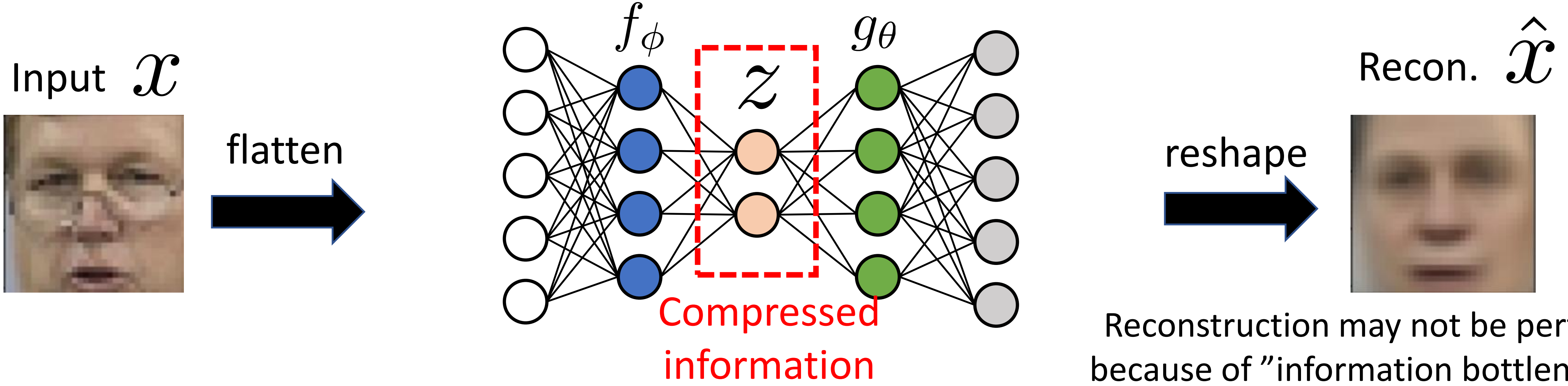


Reconstruction may not be perfect
because of ”information bottleneck”

”Standard/basic” Auto-Encoder

$$f_\phi : \mathcal{X} \in \mathbb{R}^d \rightarrow \mathcal{Z} \in \mathbb{R}^v, \text{ where } v < d$$

With “standard/basic” Auto-Encoder, in this class we refer to AE with Bottleneck

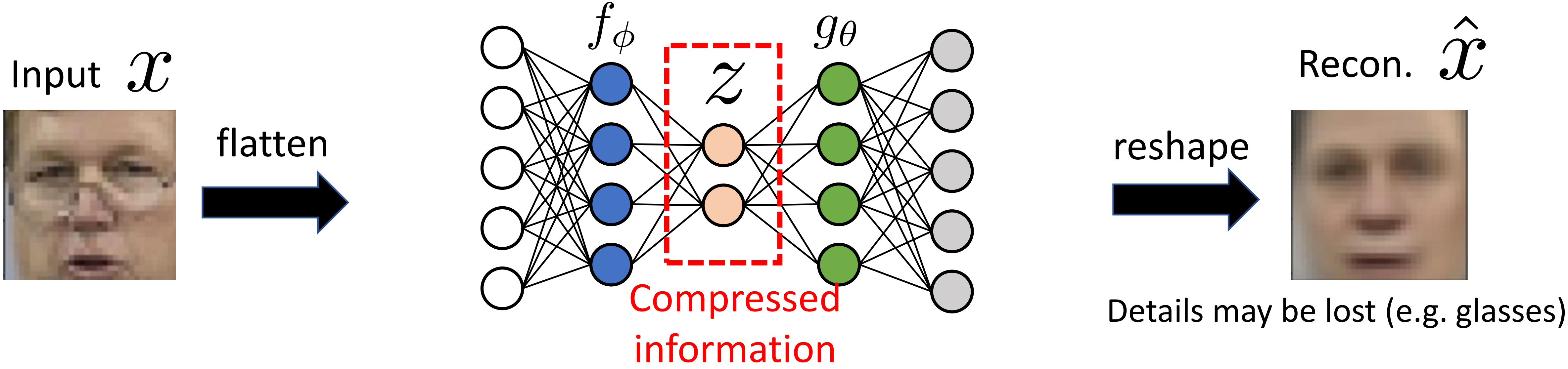


Reconstruction may not be perfect because of ”information bottleneck”

What do you think would the AE learn to encode in Z?

Compression forces encoding the most prominent features

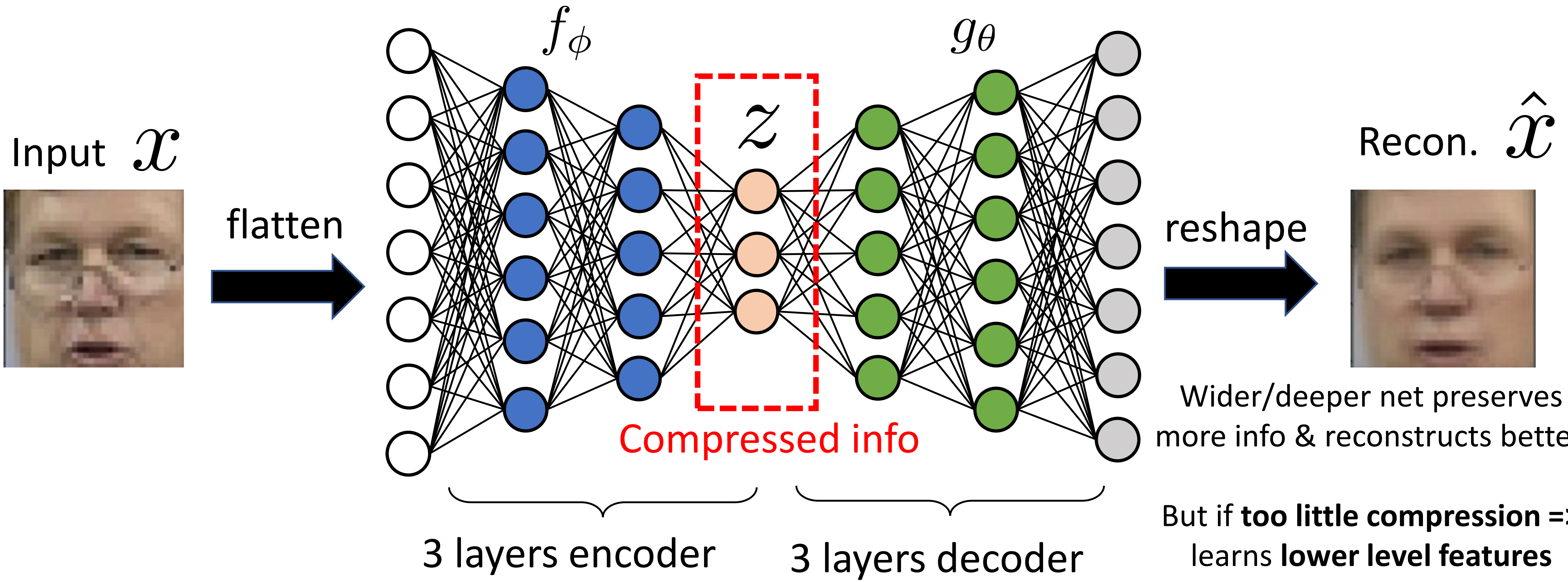
$$f_\phi : \mathcal{X} \in \mathbb{R}^d \to \mathcal{Z} \in \mathbb{R}^v, \text{ where } v < d$$



Reconstruction loss **penalizes wrong pixel intensities**. Therefore:
Encoder usually learns to encode features that **explain intensities of as many pixels as possible**.
Usually these are **"high level"** features, as often called in deep learning:
E.g. here: Skin color (most pixels), location and size of eyes, mouth, nose (dark areas), type of hair, clothing…

Wider bottleneck: Better reconstruction, less compression

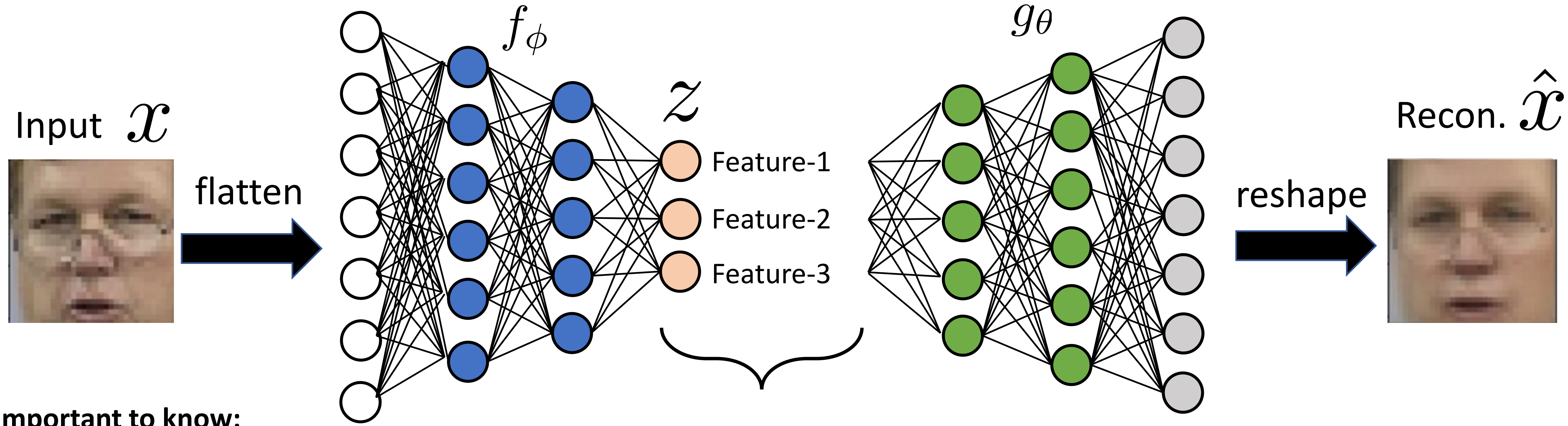
$$f_\phi : \mathcal{X} \in \mathbb{R}^d \rightarrow \mathcal{Z} \in \mathbb{R}^v, \text{ where } v < d$$



Wider/deeper net preserves more info & reconstructs better.

But if **too little compression =>** learns **lower level features** (towards encode all pixels / identity func). It's a **trade-off**.

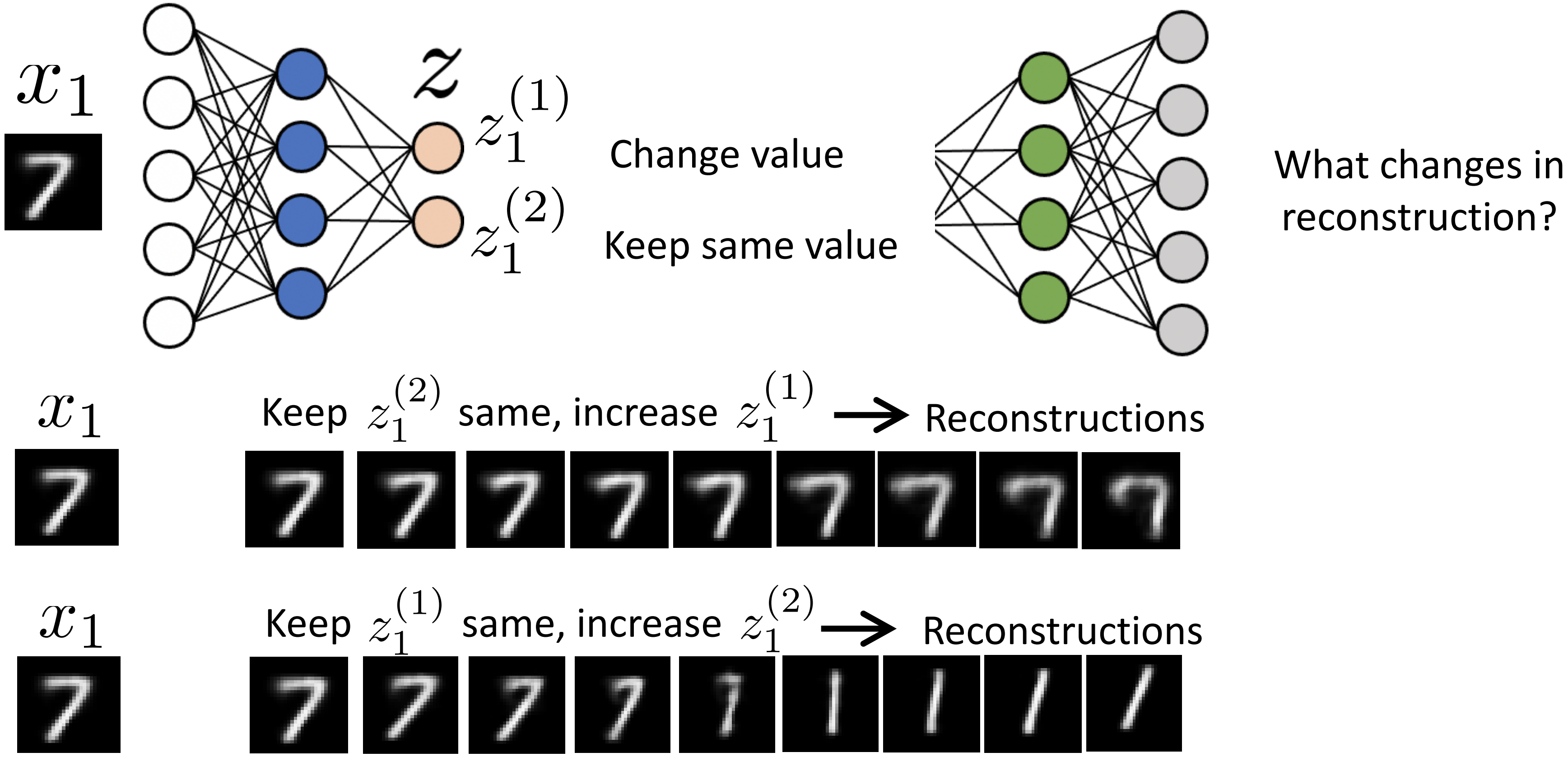
Bottleneck forces the AE to learn to encode the **few most important features** of data. But what are they?



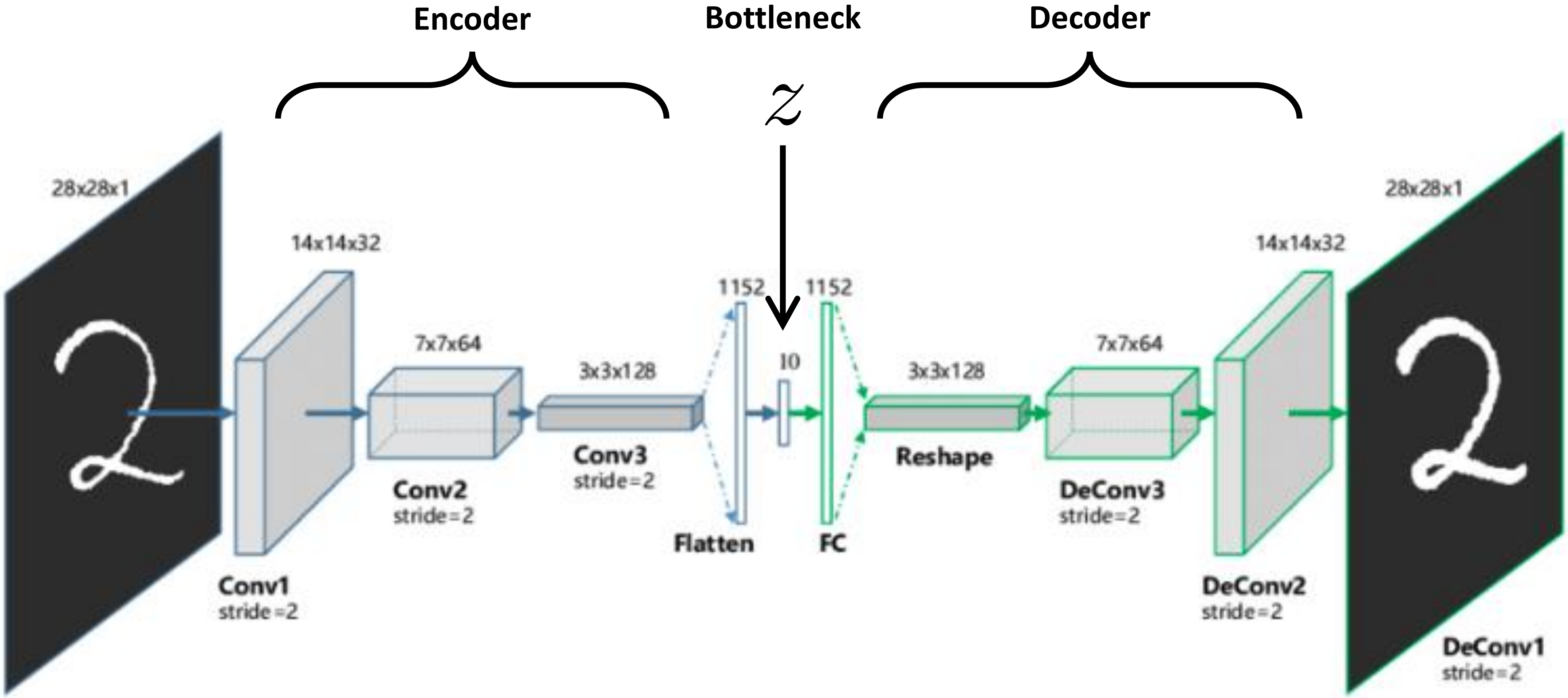
Important to know:

- **We do not control** what features are learned
- Re-training with different random seeds **may learn different features** due to randomness of SGD!
- **We do not know** what features (attributes) of the input are learned
- We can find out by visual inspection: After an AE was trained, encode X and decode it after changing only 1 feature. Then, check what changed in the reconstruction!

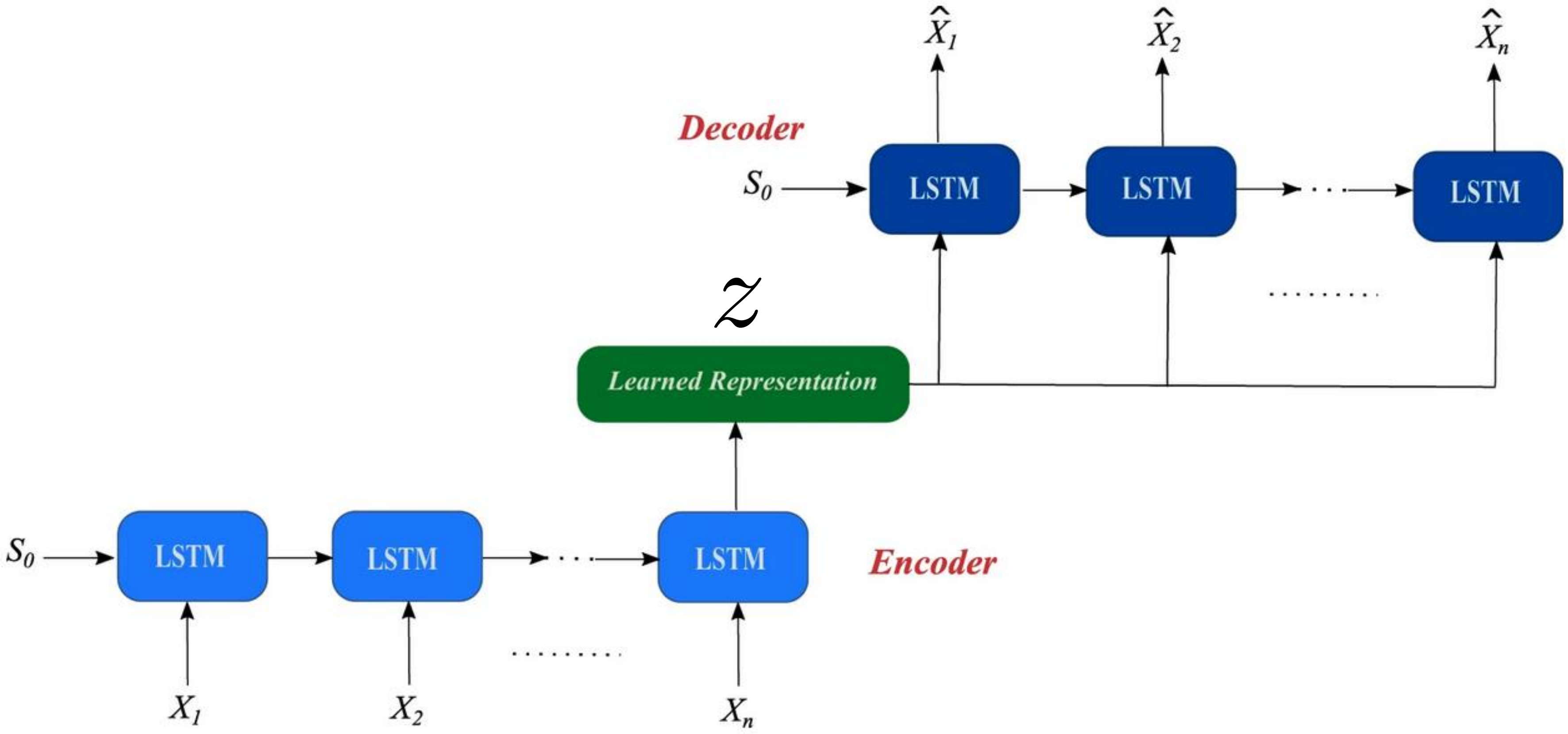
Investigating what the features are by visual inspection



Convolutional Auto-Encoders



RNN-based Auto-Encoders



From: Sagheer & Kotb, "Unsupervised Pre-training of a Deep LSTM-based Stacked Autoencoder for Multivariate Time Series Forecasting Problems", Nature Scientific Reports, 2019

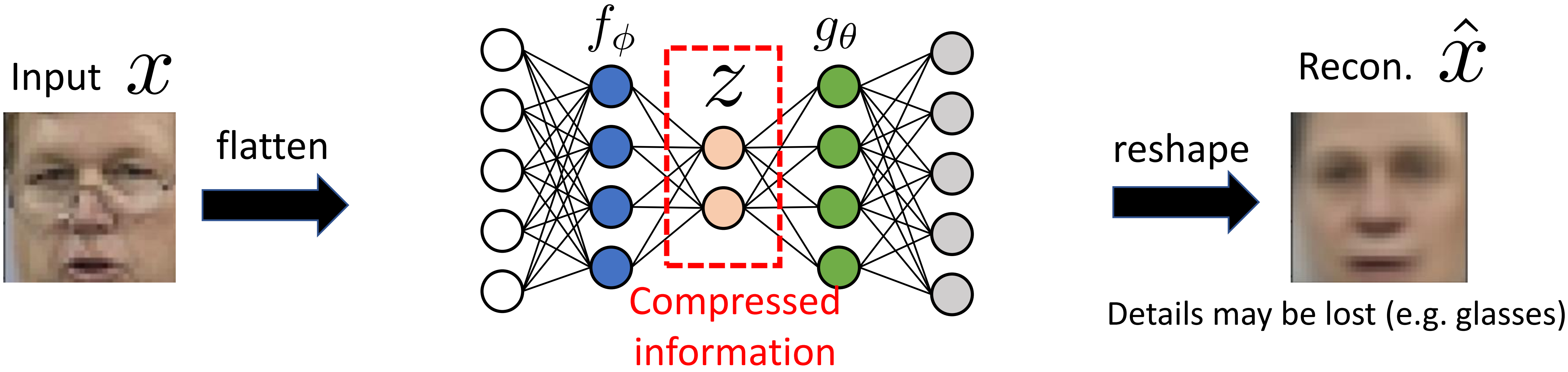
In this part:

- What is an Auto-Encoder
- What is the information bottleneck and why use it
- How to train an Auto-Encoder

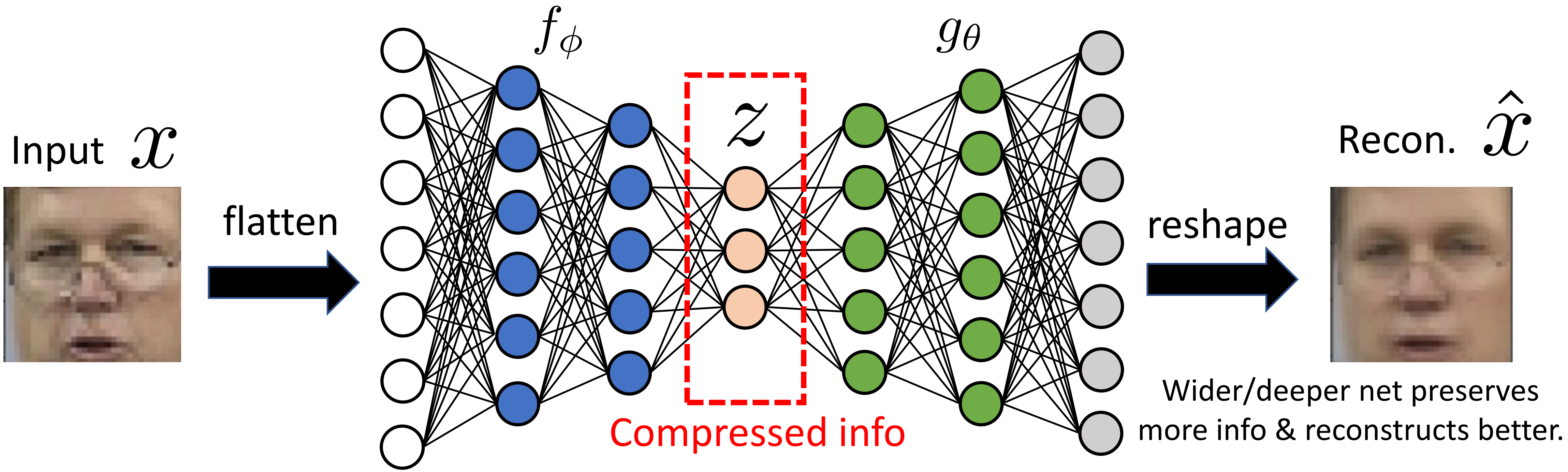
Next part:

- What can we **use AEs for**?
- What are AEs **not good for**?

AE with Bottleneck layer learns to perform Dimensionality Reduction / Compression via Unsupervised Learning



Wider bottleneck: Better reconstruction, less compression

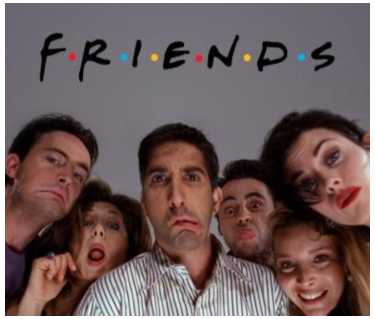


But if too little compression **=>** AE may not learn high level features (towards encoding all pixels / identity func).

It's a **trade-off**.

Use of AEs for compression - Example application

Streaming Company



720

1280

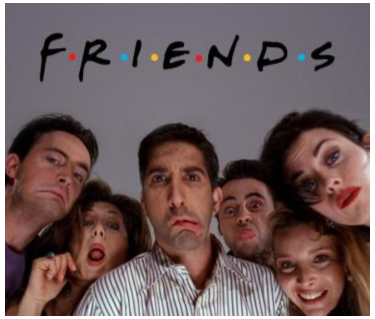
$1280 \times 720 = 921600$ pixels

Streaming 921600 values per frame.
Slow!

Customer

Use of AEs for compression - Example application

(1) Streaming company trains AE on the movie frames:

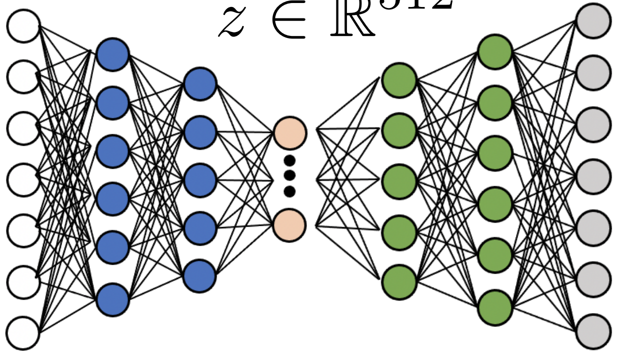


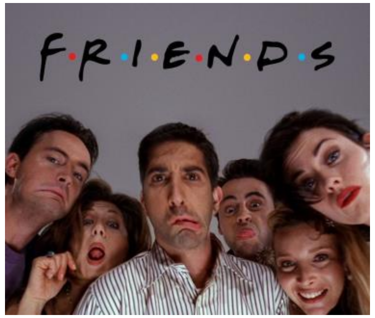
720

1280

$1280 \times 720 = 921600$ pixels

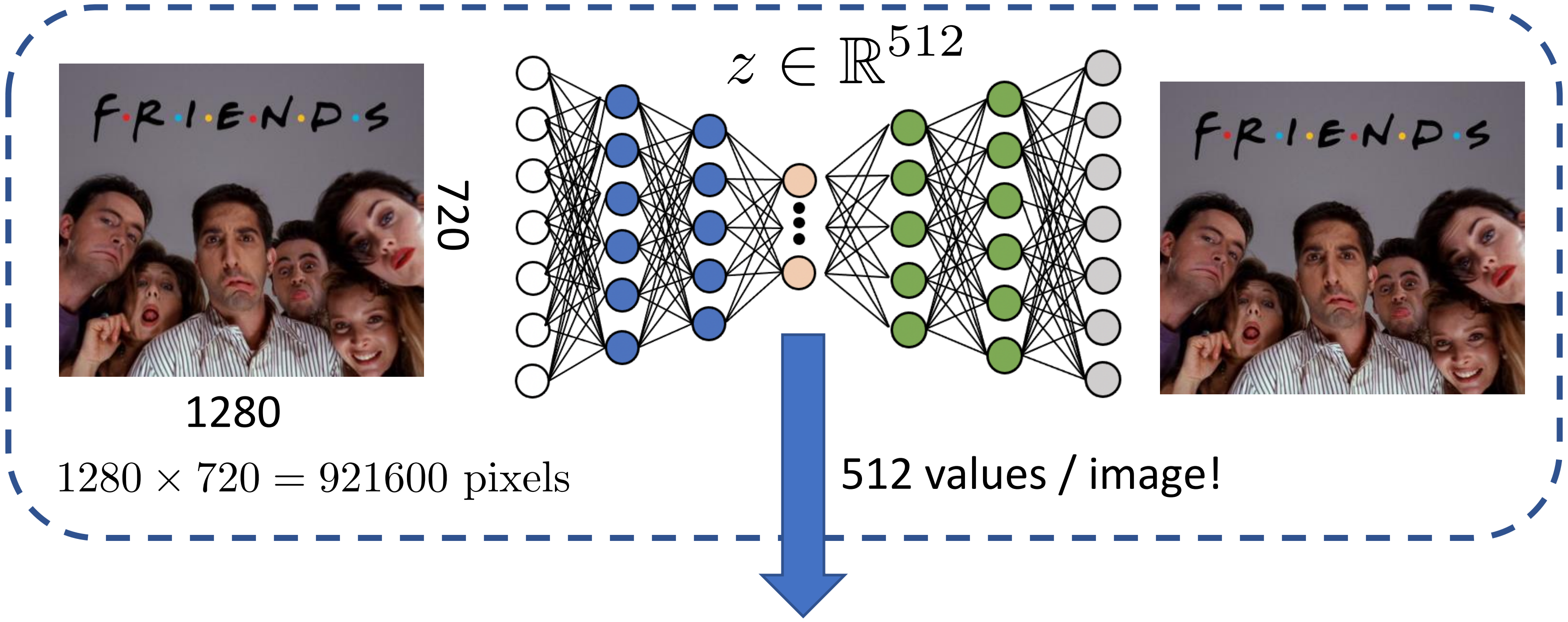
$z \in \mathbb{R}^{512}$



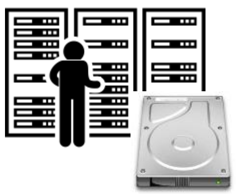


Use of AEs for compression - Example application

(1) Streaming company trains AE on the movie frames:

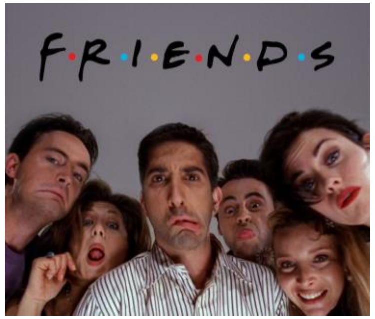


(2) Stores movie
encoded in Z
on own servers



Use of AEs for compression - Example application

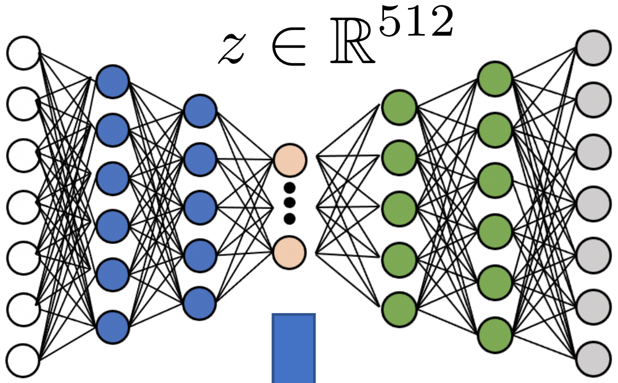
(1) Streaming company trains AE on the movie frames:



720

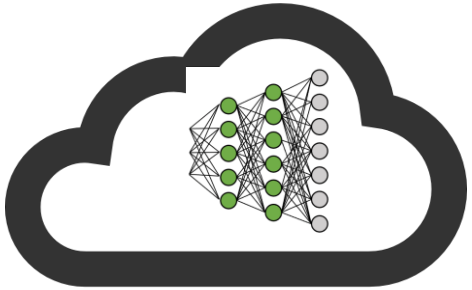
1280

$1280 \times 720 = 921600$ pixels

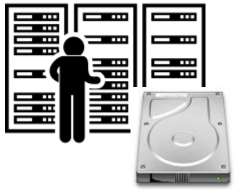


512 values / image!

(3) Deploys decoder on cloud within client

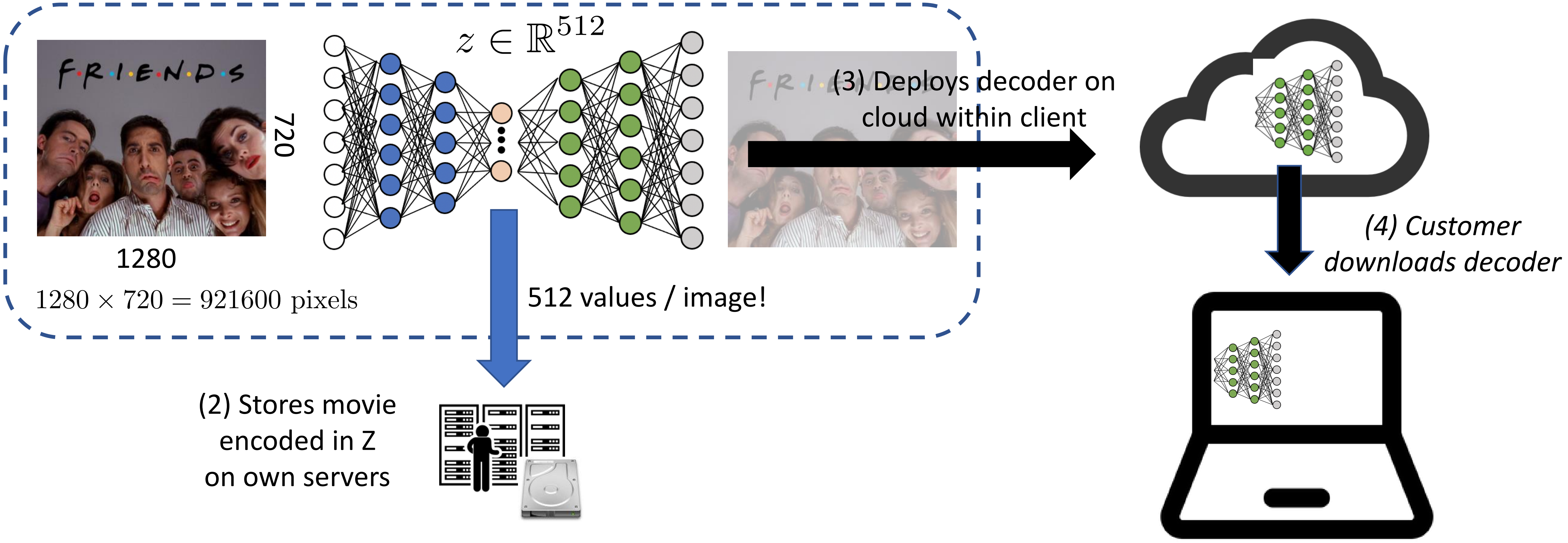


(2) Stores movie encoded in Z on own servers

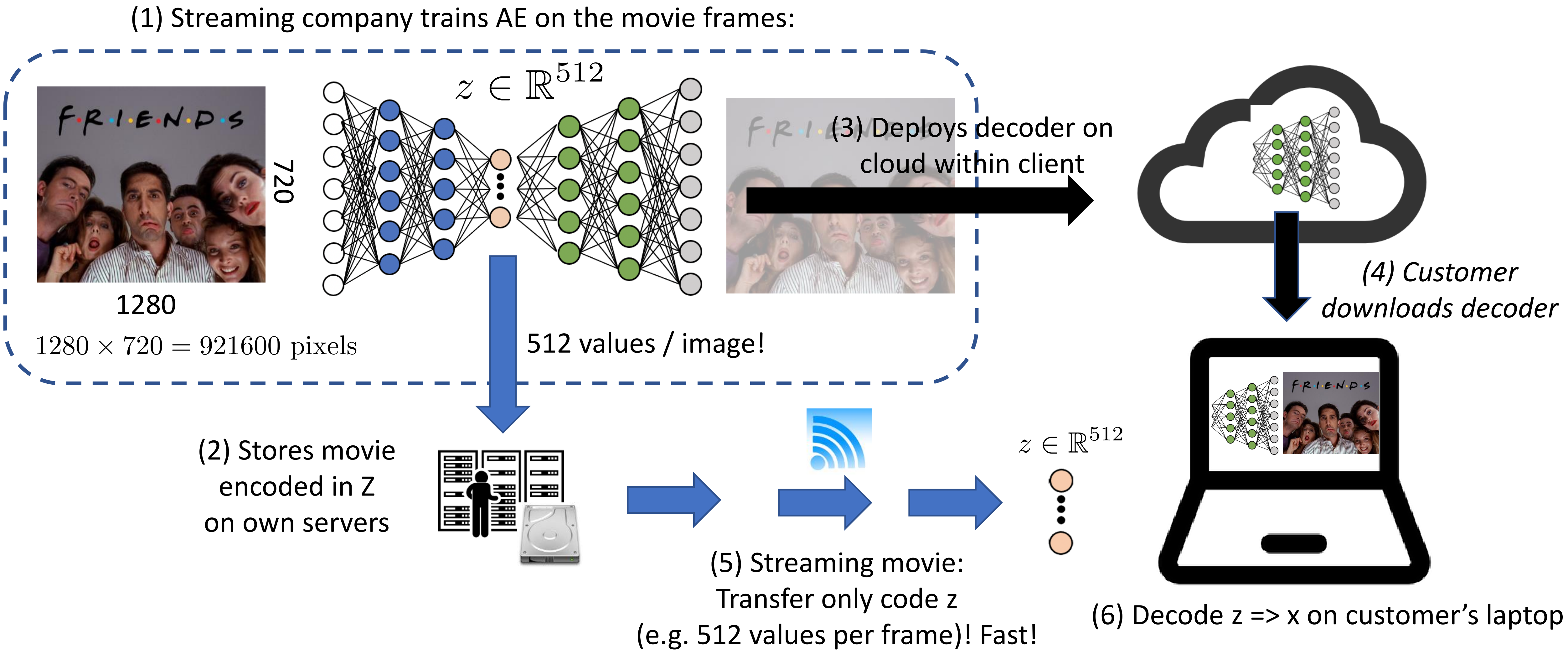


Use of AEs for compression - Example application

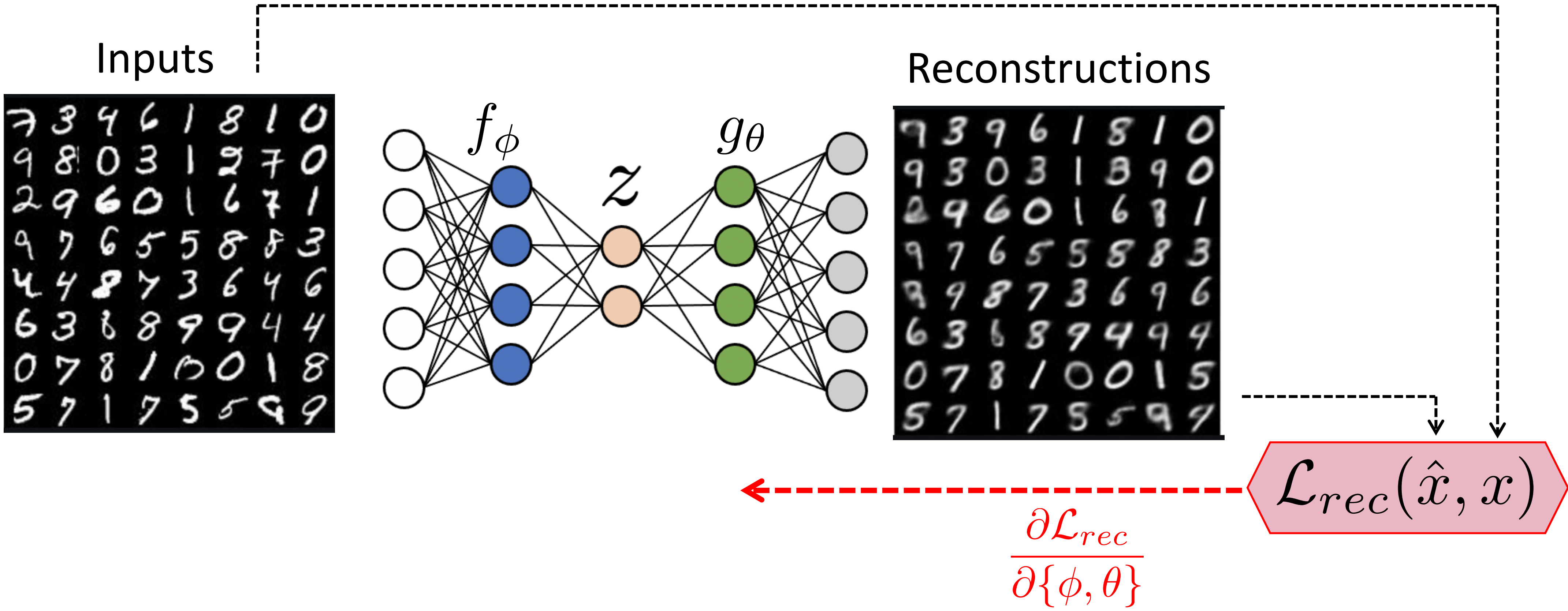
(1) Streaming company trains AE on the movie frames:



Use of AEs for compression - Example application

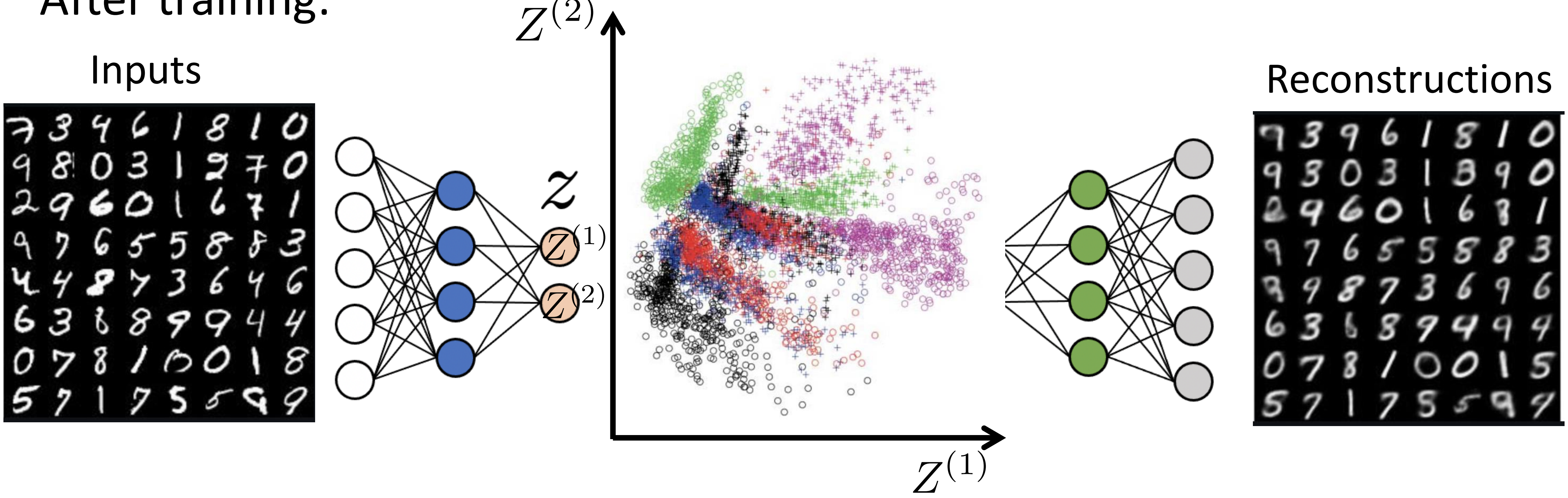


AutoEncoders: Do they learn to encode “useful” info in Z?



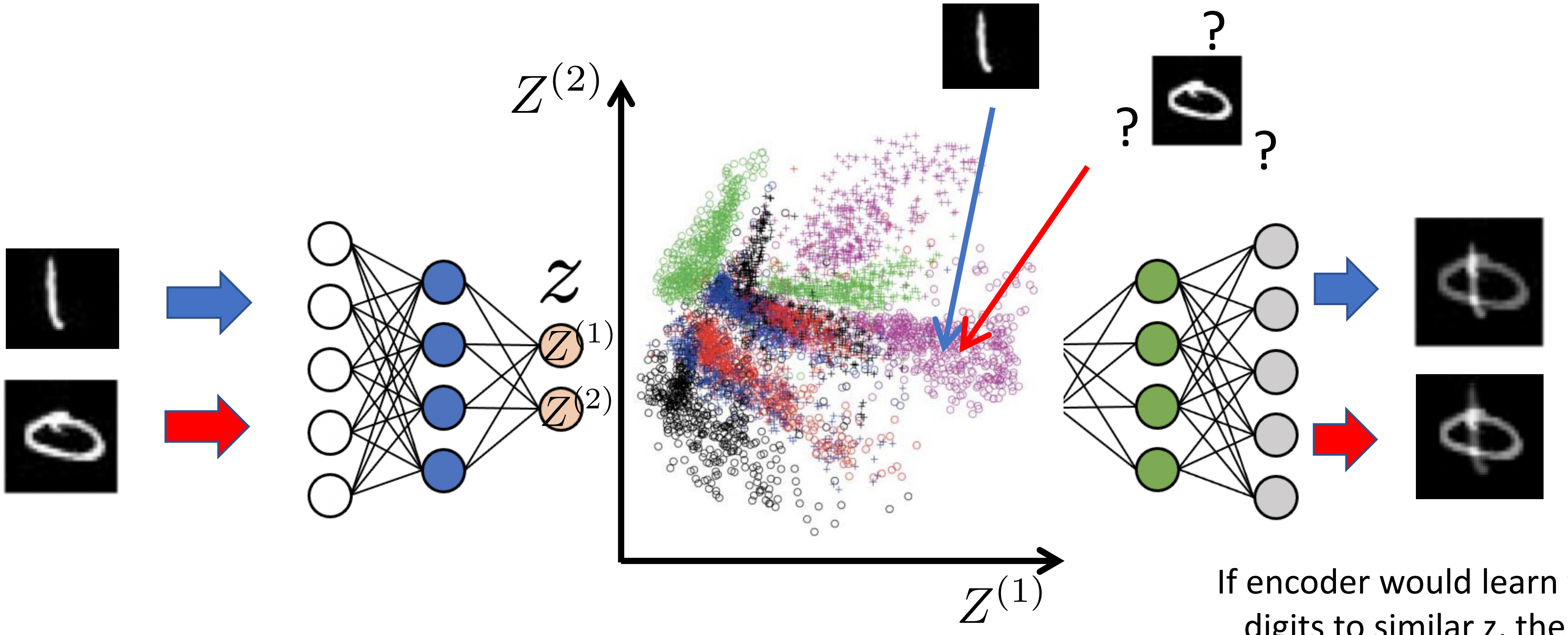
AEs can learn to **cluster** the data in unsupervised manner!

After training:



From: Hinton and Salakhutdinov, Reducing the Dimensionality of Data with Neural Networks, Science, 2006

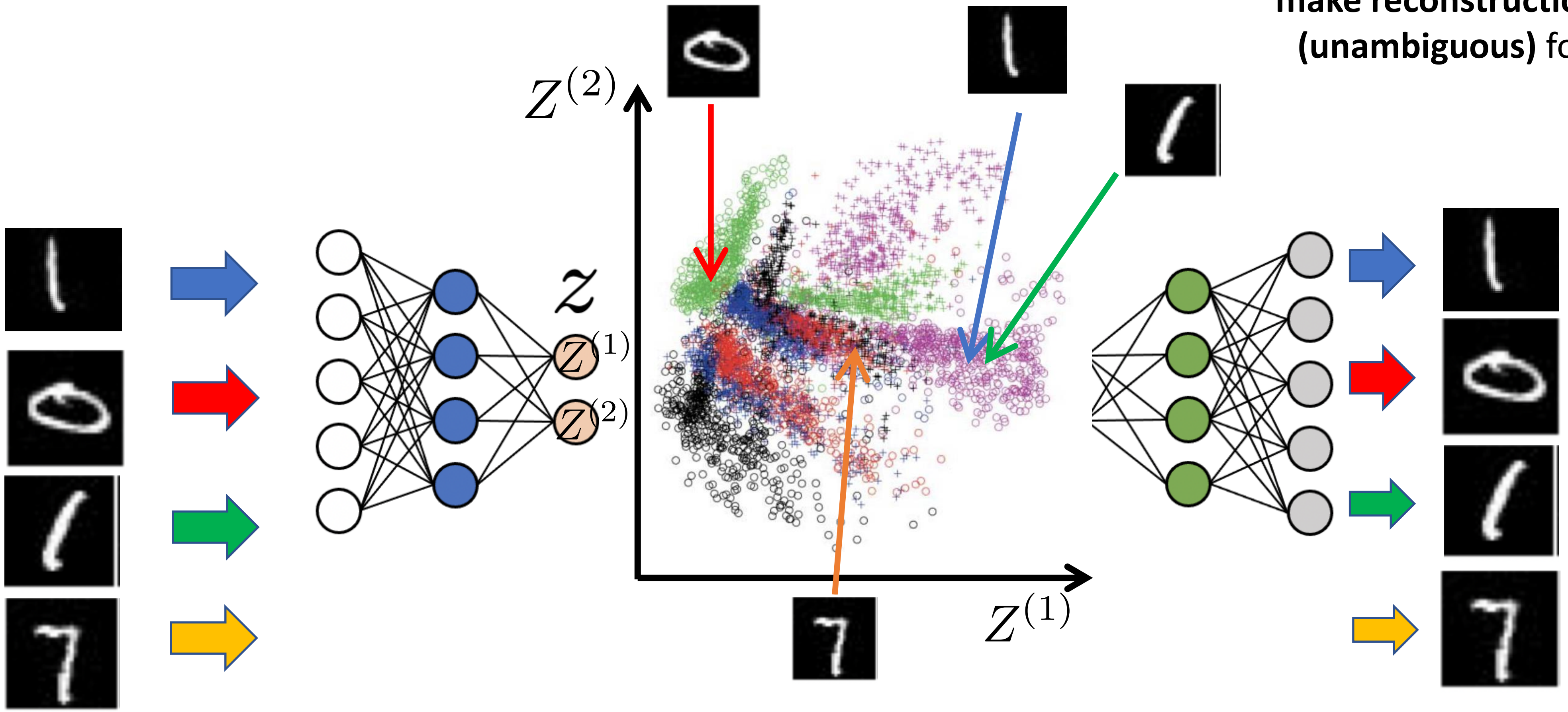
Why learn to cluster data?



If encoder would learn to map 2 different digits to similar z, then reconstruction z-> x would fail for both!
High loss!

Why learn to cluster data?

To minimize the loss, the **Encoder** tries **to make reconstruction z-> x easy (unambiguous)** for Decoder**.**

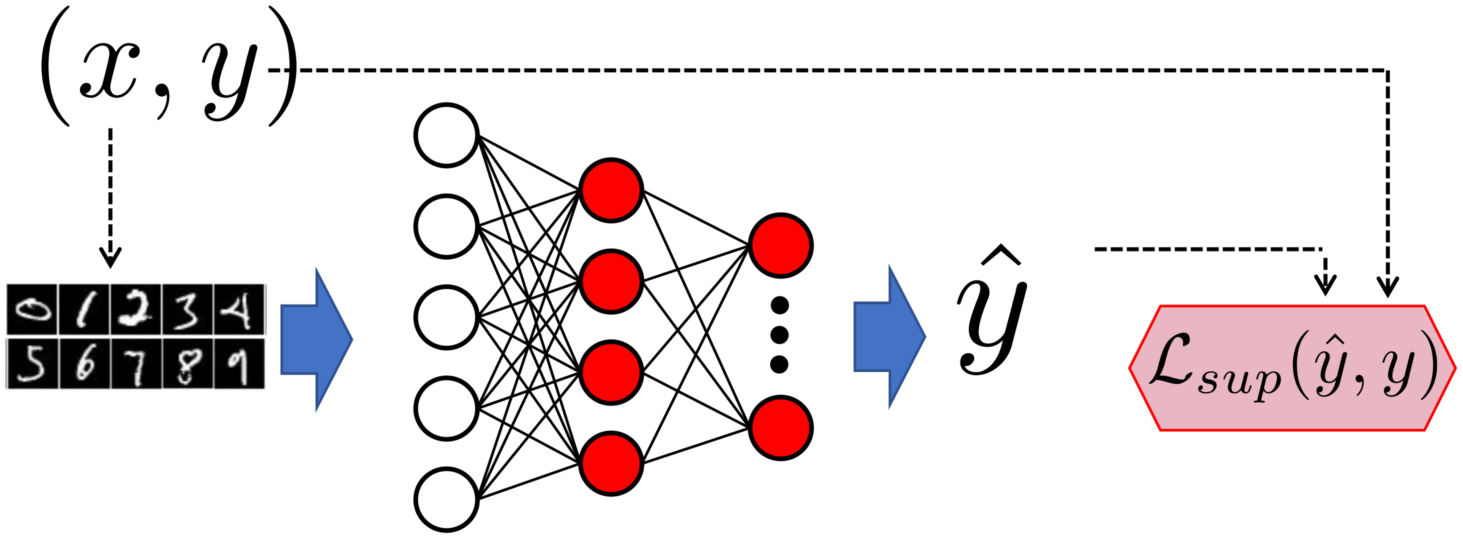


Therefore, **Encoder** learns to **map similar samples close in space of Z, and different samples far away**.

Learn from unlabeled data, when **<u>labels are limited</u>**

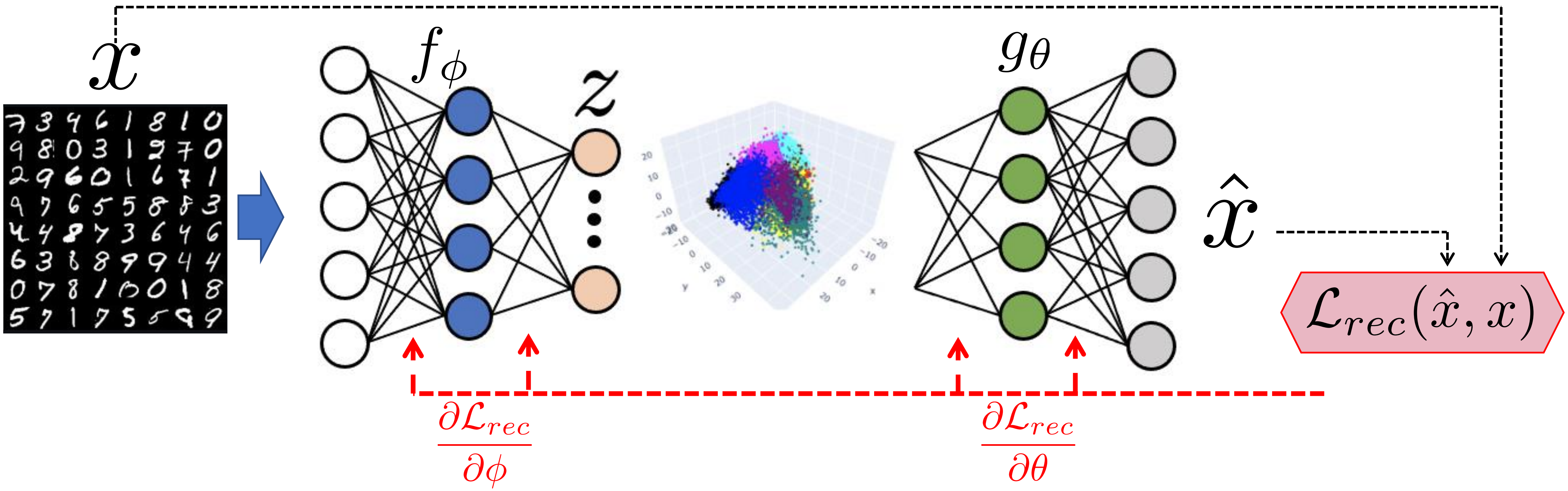
Assume **our ultimate goal is to learn a Classifier with Supervised Learning.**
But number (N) of labeled data is **<u>small</u>**.

$$\mathbf{D} = (\mathbf{X}, \mathbf{Y}) = \{(x_i, y_i)\}_{i=1}^{N}$$

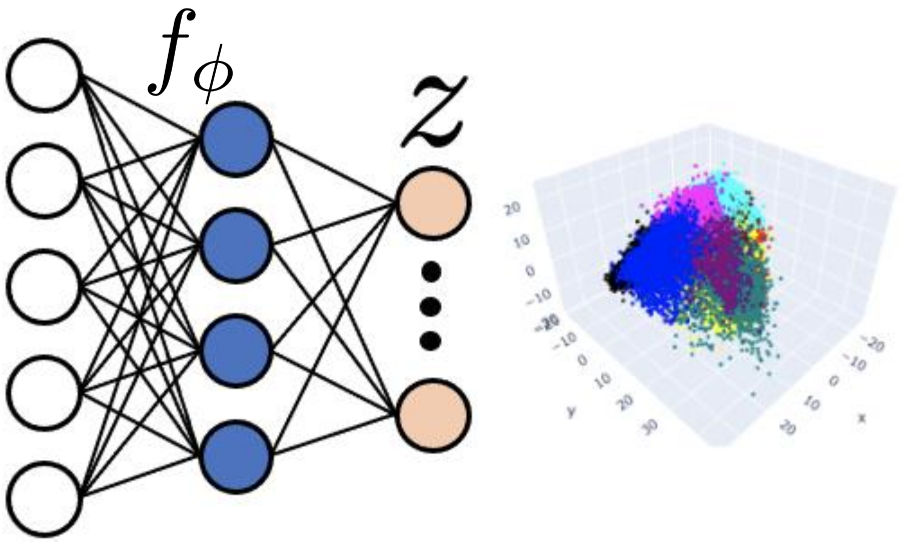


Potential overfit!
Whole image-to-label network (many parameters) trained with little data!

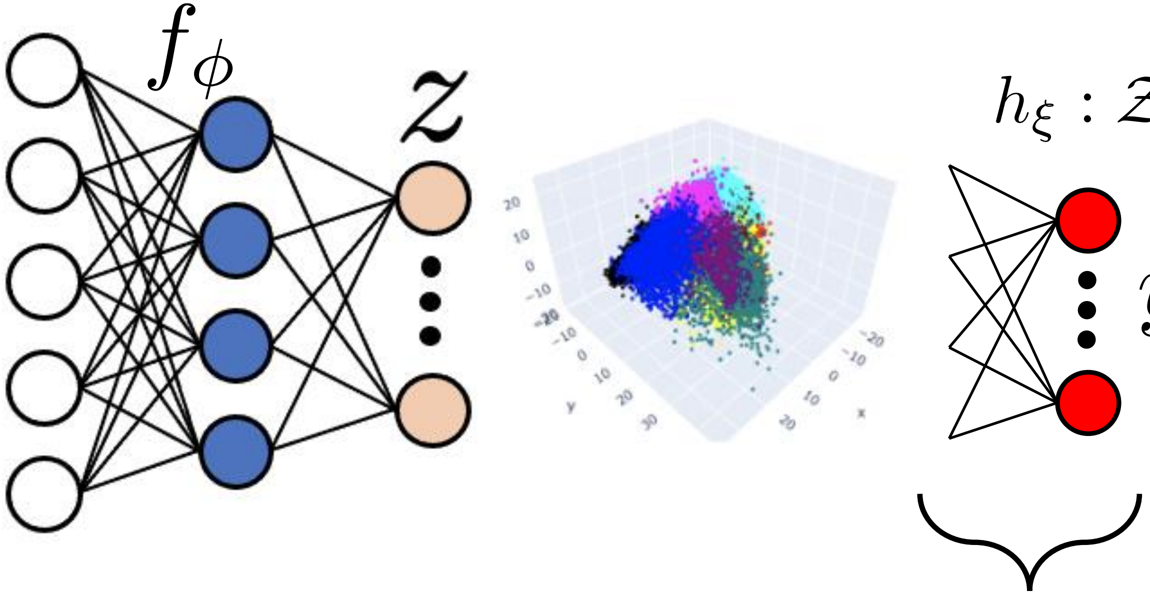
Pre-train with unlabeled data…



Trained Encoder of AE maps data X to space Z where they are **clustered**.
Throw away the decoder…



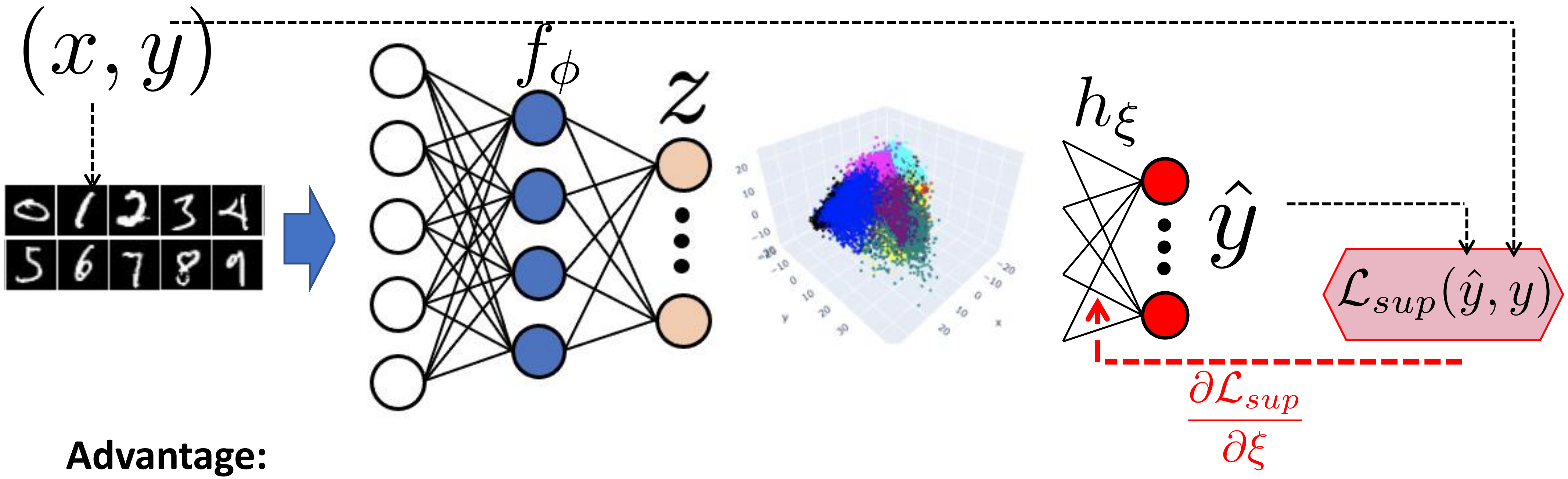
Attach **untrained** classifier on top of encoder (commonly 1-2 layers)



$$\hat{y} = h_\xi(f_\phi(x))$$

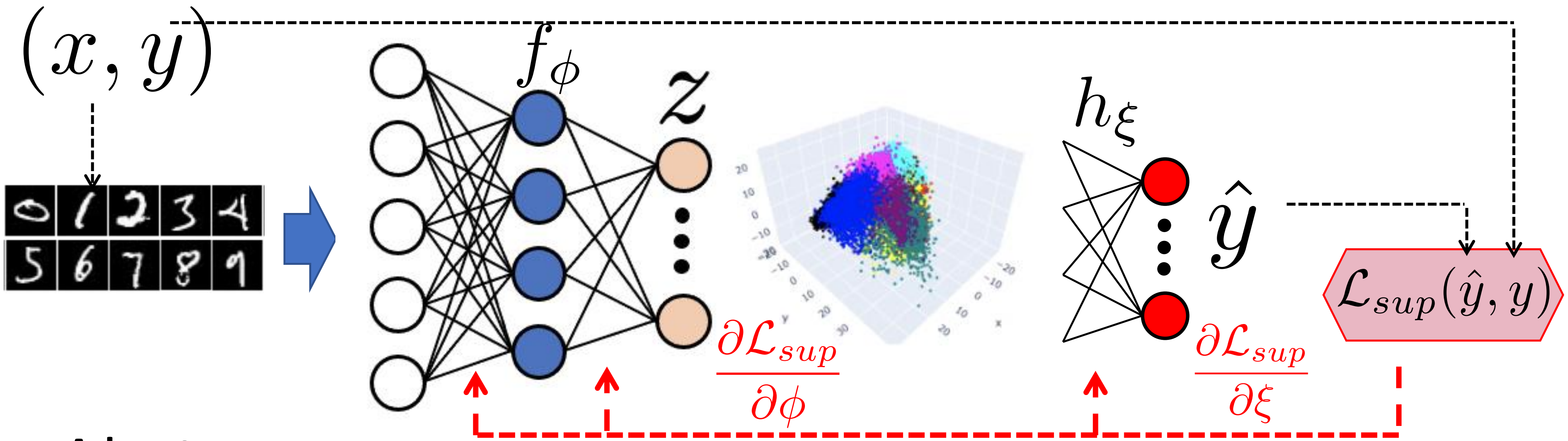
Commonly shallow, 1-2 layers.
Therefore, it has very few parameters

(Approach 1)
Using the **limited labelled data**, train with **Supervised Learning ONLY the classifier**.
Keep parameters of **encoder frozen**.



Advantage:
Trains only the the small classifier with the limited labels. Therefore can avoid over-fit.
Disadvantage:
Encoder is not optimized for labelled data. May be suboptimal.

(Approach 2)
Using the **limited labelled data**, train with **Supervised Learning BOTH the encoder and the classifier** (usually for only few SGD iterations).



Advantage:
Encoder is optimized via labels, which ”may” lead to better representation Z and results.
Disadvantage:
Possibility to overfit as all parameters are trained. Limited GD steps to avoid this.
Number of GD steps must be carefully decided on validation data to avoid overfit.

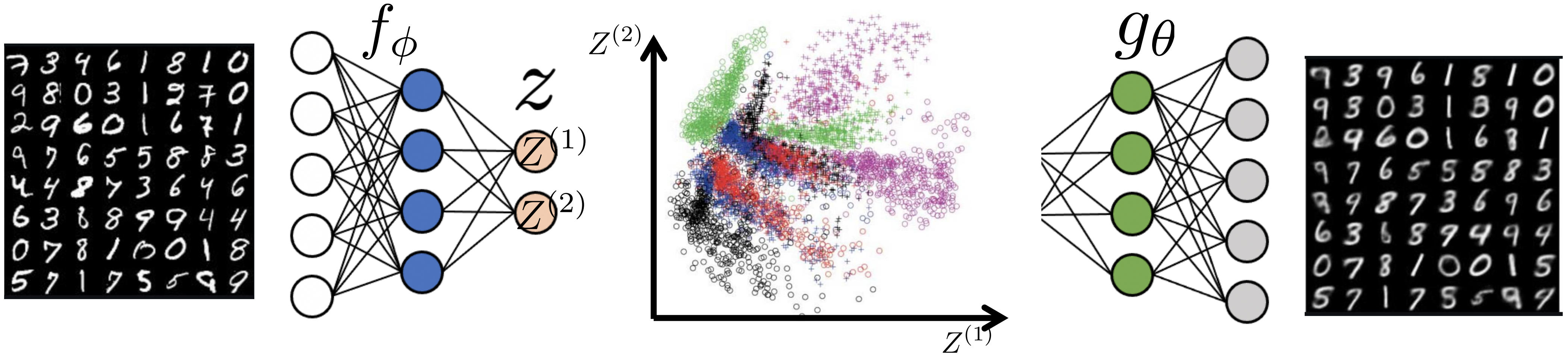
Can we generate (synthesize) new (not real) data with basic AE?

Can we generate (synthesize) new (not real) data with basic AE?

Assume an ***already trained*** Auto-Encoder

$f : \mathcal{X} \rightarrow \mathcal{Z}$

$g : \mathcal{Z} \rightarrow X$

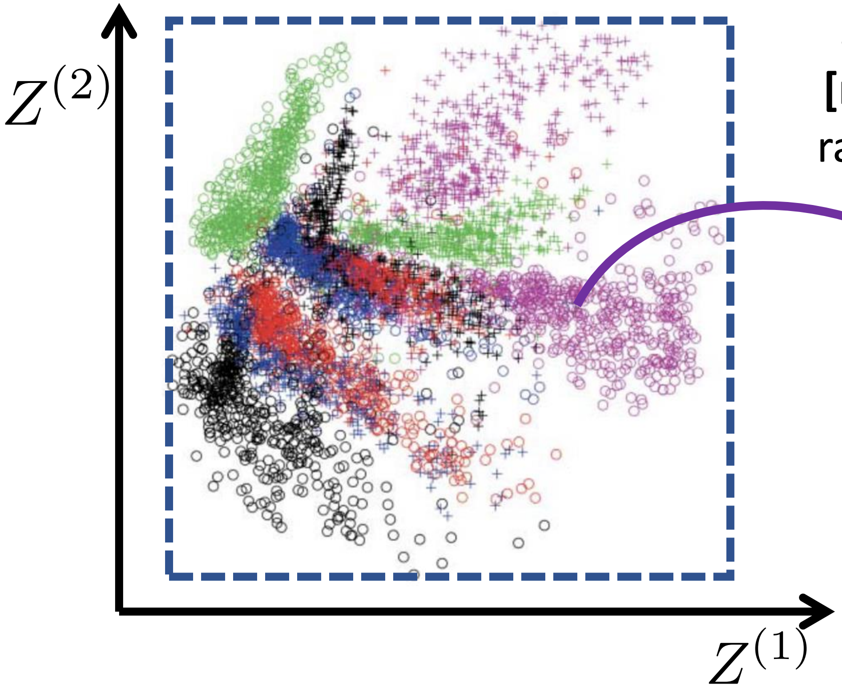


Is it a model appropriate for generating new data?
It is not trained for it..!

Can we generate (synthesize) new (not real) data with basic AE?

Step 1: Sample random z

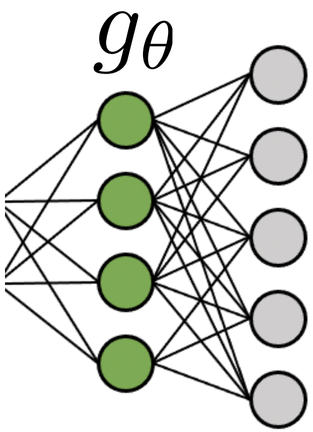
E.g. With uniform probability between
[min, max] values z seen during training



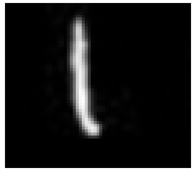
Sample from
[min-z, max-z]
range of values

$$\tilde{z}_1$$

Step 2: Decode



$g_\theta(\tilde{z}_1)$

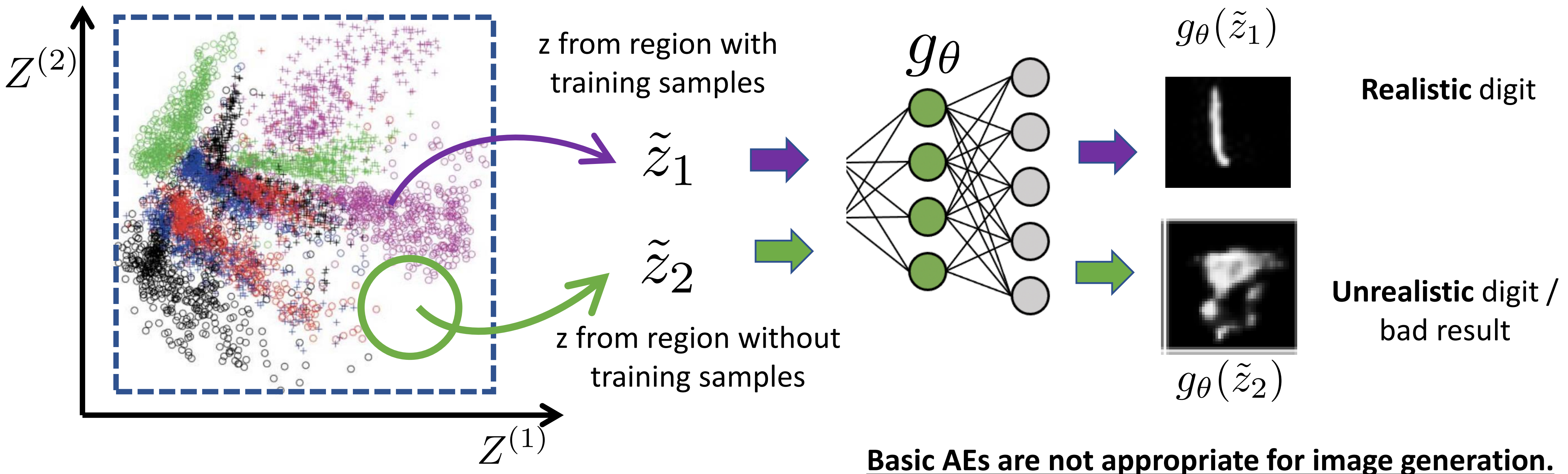


Problems generating new data with basic AE

Step 1: Sample random z

E.g. With uniform probability between [min, max] values z seen during training

Step 2: Decode



Problems:

a) No "**real**" digits were encoded in that area during training. Hence these z values do not encode "realistic" digits.

b) Decoder has not learned to decode such z values

<u>**Basic AEs are not appropriate for image generation.**</u>
<u>**Reconstruction loss does not train AE for generation.**</u>

We will see how ***Generative Models (VAEs and GANs)*** are trained appropriately for generation.

In this video:
- What can we use basic AEs for?
  (e.g. compression, clustering, pretrain/initialize supervised model when labelled data are limited)
- What are “standard” AEs not good for?
  (generation of new samples)

Next week:
- Generative Models
- Variational Auto-Encoders

For further reading (e.g. some more advanced AEs):

- Goodfellow, Bengion & Courville, Deep Learning, Chapter 14
  https://www.deeplearningbook.org/

(However only the material our own slides, videos & tutorials will be assessed)

Thank you very much for your attention