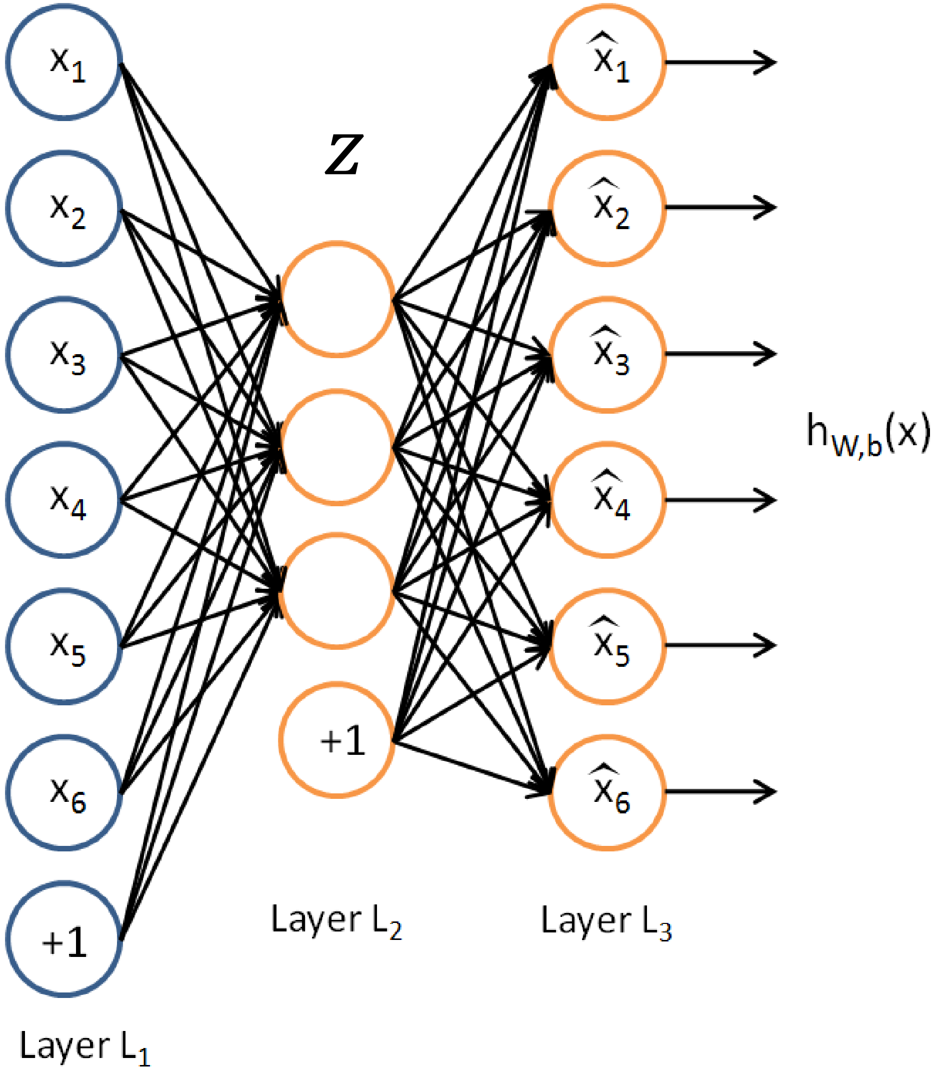
Encoder-decoder model

Biplab Banerjee

Deep CNN based image segmentation

- Auto-encoder
- Regularized auto-encoder
- Class-encoder
- Image segmentation

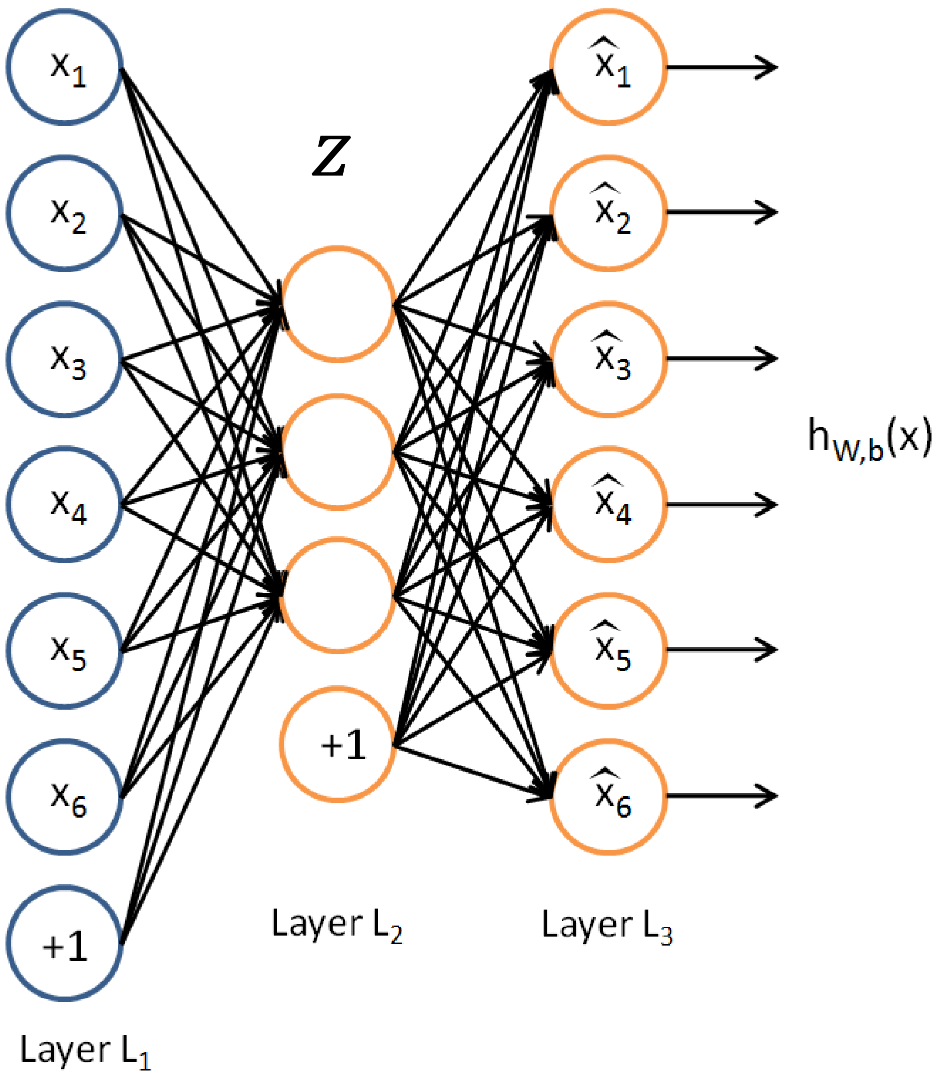
Traditional Autoencoder



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. (later)



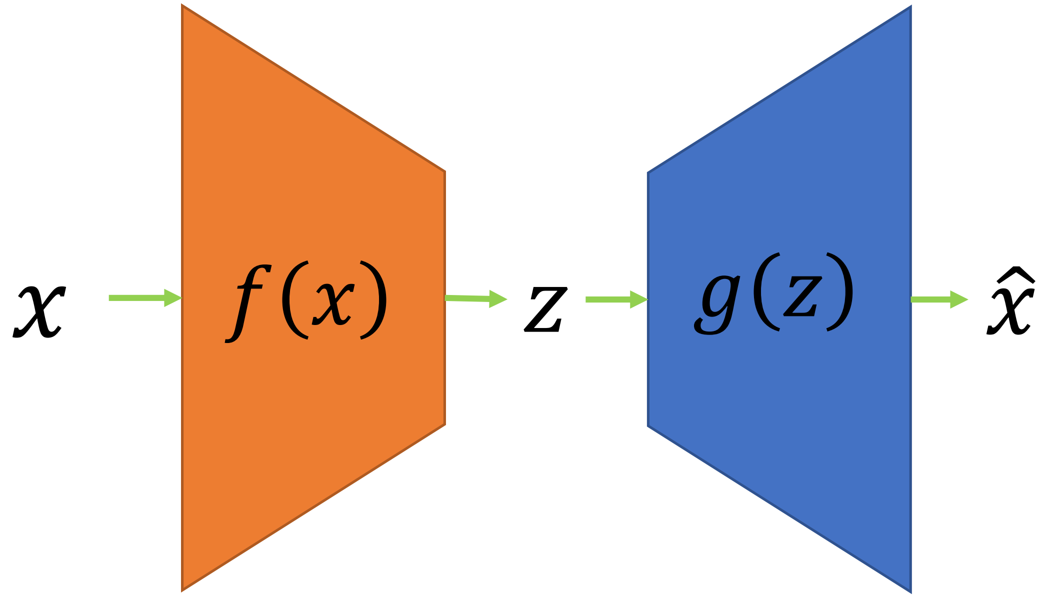
Uses

- 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.
- Representation learning

Simple Idea

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



$$f(x) = s(wx + b) = z$$

and

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

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

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

(z is some **latent** representation or **code** and s is a non-linearity such as the sigmoid)

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

Training the AE

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

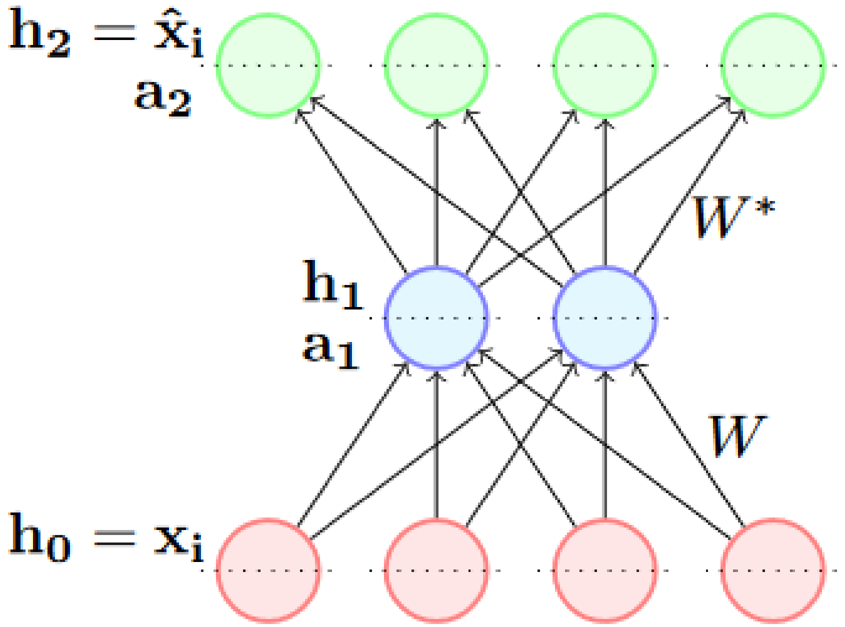
- Traditionally with *squared error* loss function

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

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

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

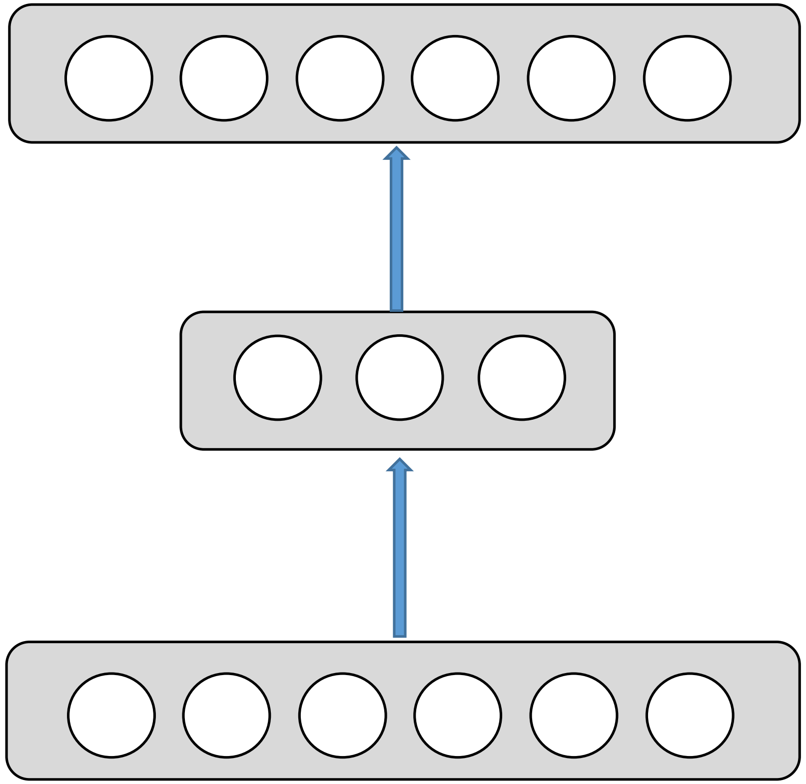
$$\mathscr{L}(\theta) = (\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}_\mathbf{i})^T(\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}_\mathbf{i})$$

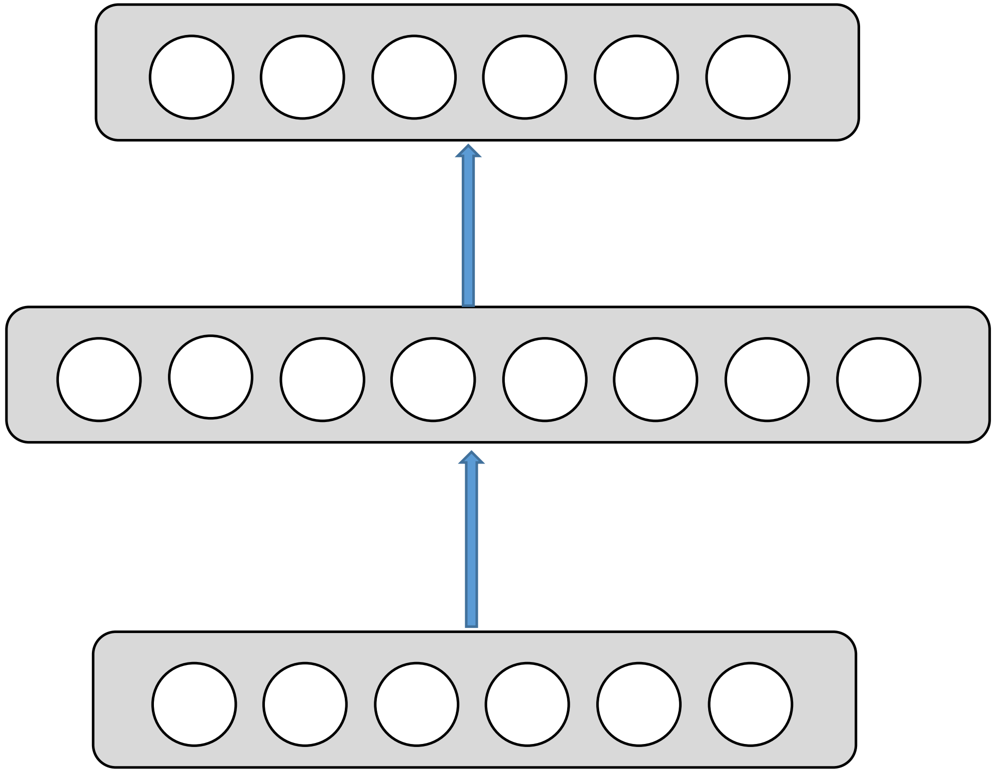


- $$\frac{\partial \mathscr{L}(\theta)}{\partial W^*} = \frac{\partial \mathscr{L}(\theta)}{\partial \mathbf{h_2}} \boxed{\frac{\partial \mathbf{h_2}}{\partial \mathbf{a_2}} \frac{\partial \mathbf{a_2}}{\partial W^*}}$$
- $$\frac{\partial \mathscr{L}(\theta)}{\partial W} = \frac{\partial \mathscr{L}(\theta)}{\partial \mathbf{h_2}} \boxed{\frac{\partial \mathbf{h_2}}{\partial \mathbf{a_2}} \frac{\partial \mathbf{a_2}}{\partial \mathbf{h_1}} \frac{\partial \mathbf{h_1}}{\partial \mathbf{a_1}} \frac{\partial \mathbf{a_1}}{\partial W}}$$

Undercomplete AE VS overcomplete AE

We distinguish between two types of AE structures:

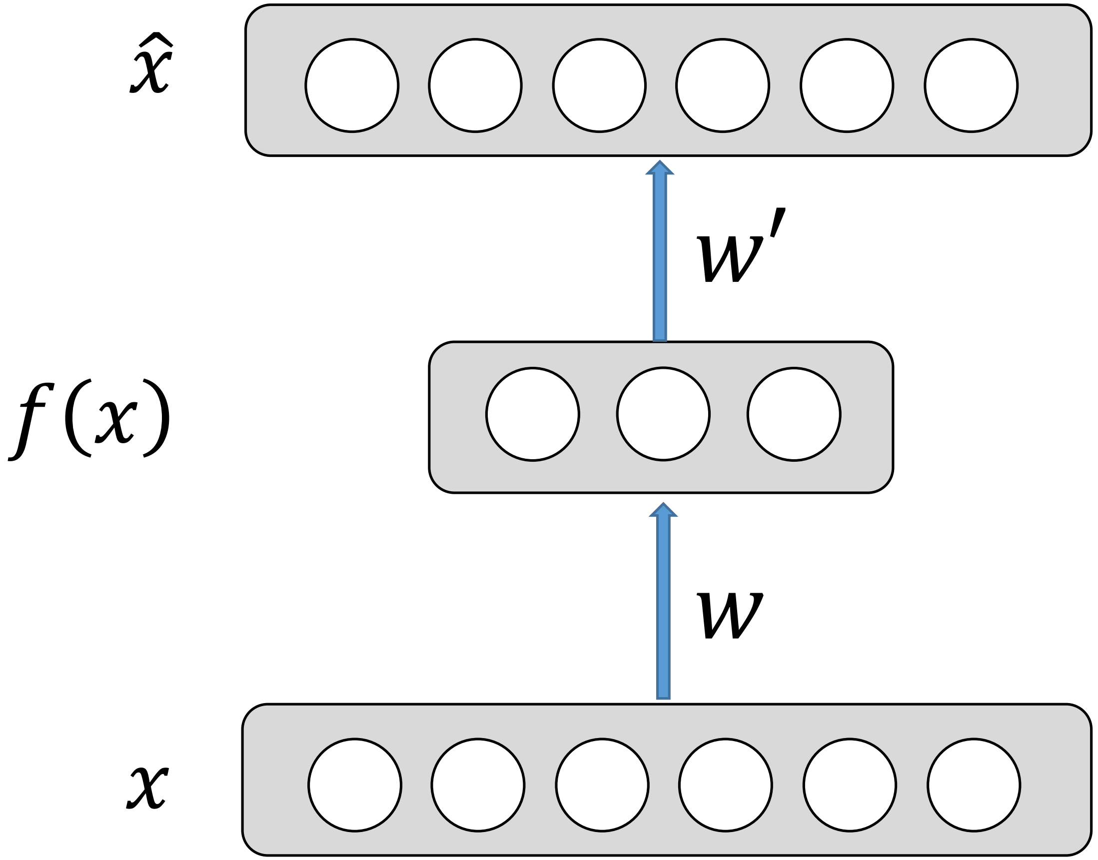




Undercomplete AE

- Hidden layer is **Undercomplete** if smaller than the input layer
  - ❑Compresses the input
  - ❑Compresses well only for the training dist.

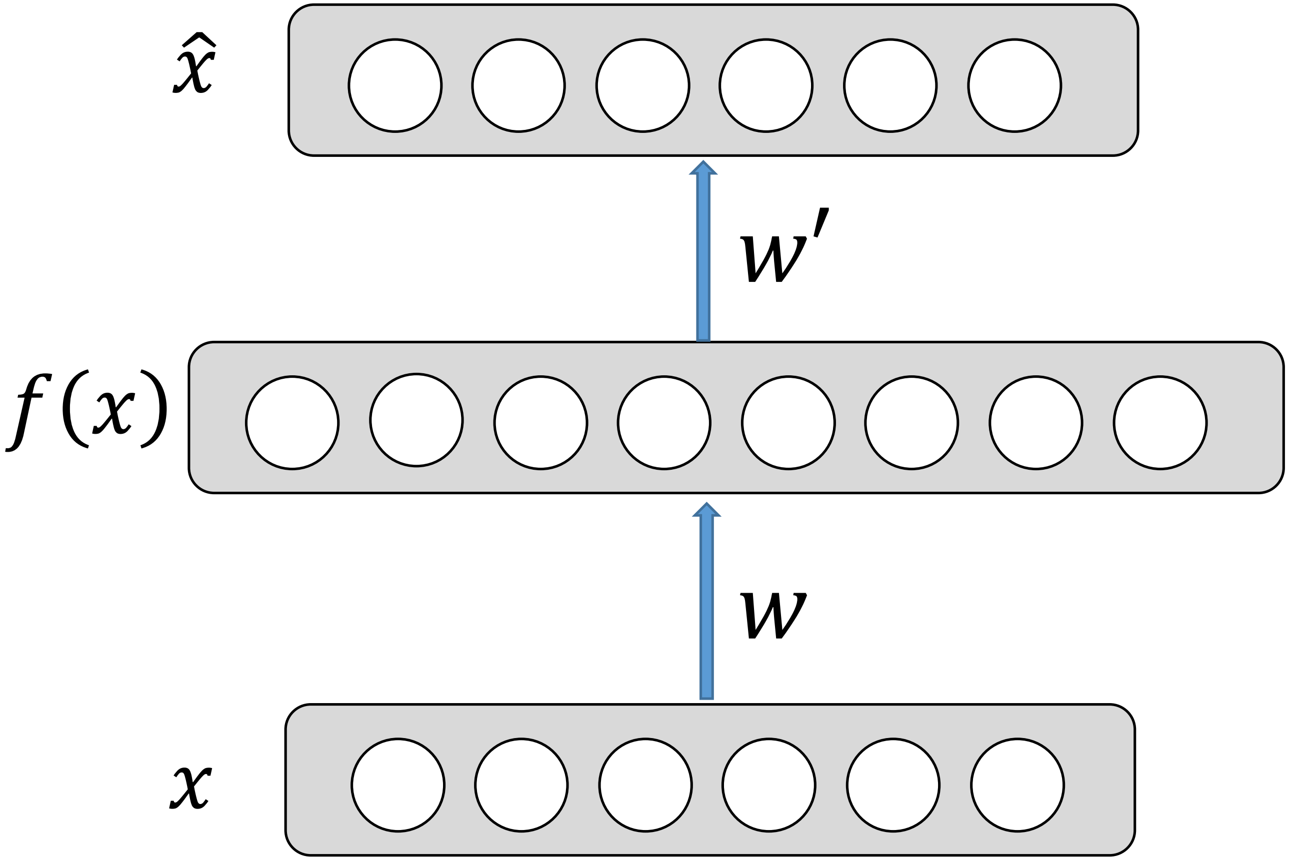
- Hidden nodes will be
  - ❑Good features for the training distribution.
  - ❑Bad for other types on input



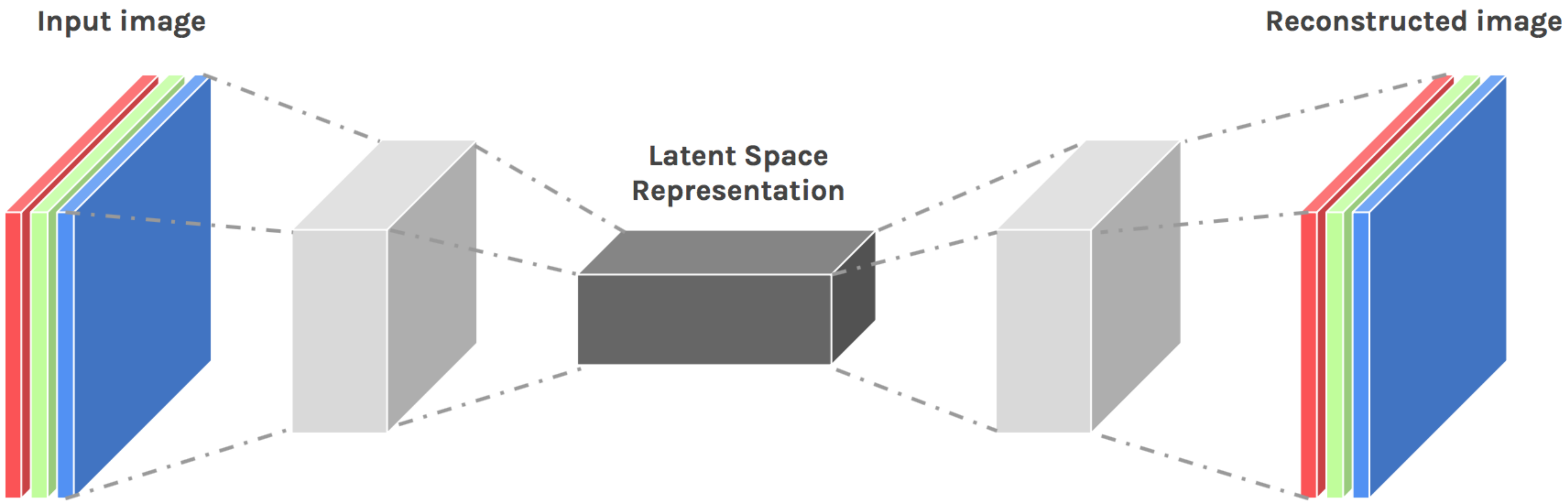
Overcomplete AE

- 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.

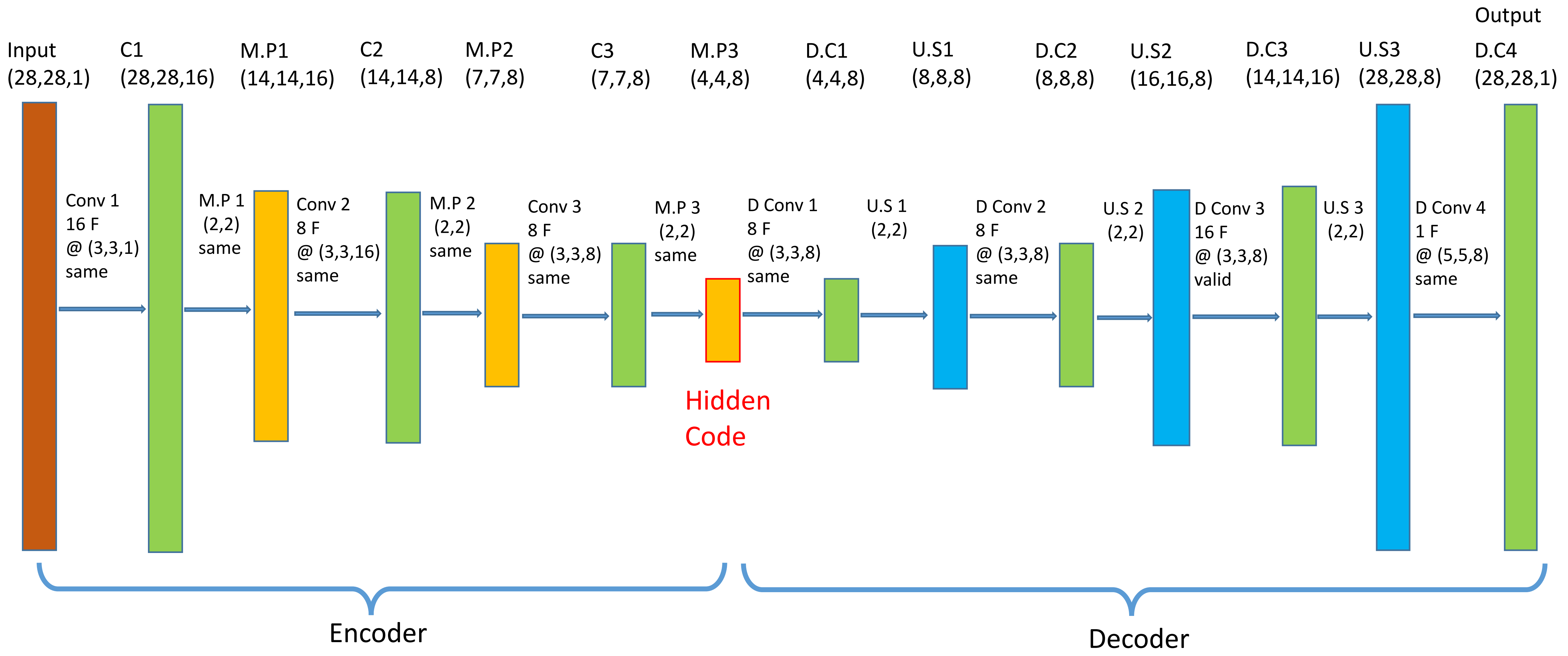


Convolutional AE



* Input values are normalized
* All of the conv layers activation functions are relu except for the last conv which is sigm

Convolutional AE



Regularization

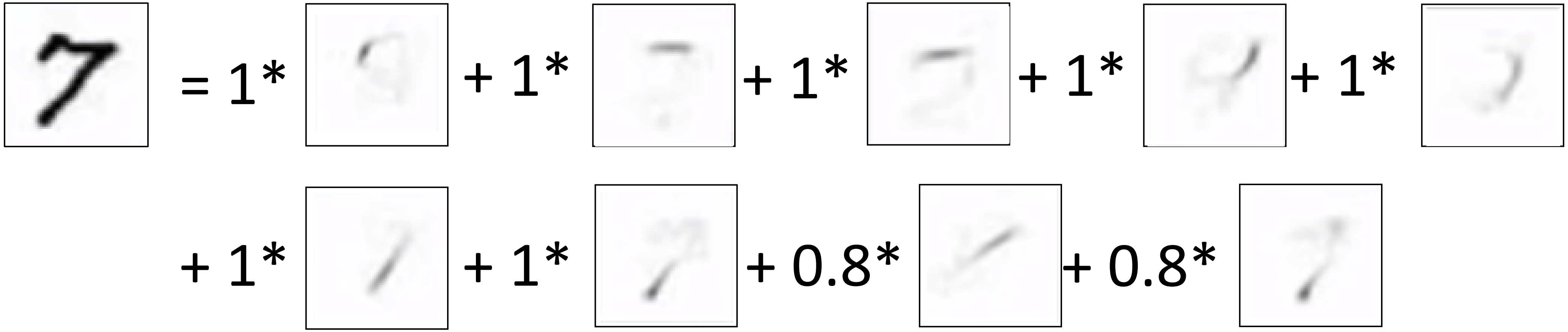
Motivation:

- We would like to learn meaningful features **without** altering the code’s dimensions (Overcomplete or Undercomplete).

The solution: imposing other constraints on the network.

Sparsely Regulated Autoencoders

- We want our learned features to be as **sparse** as possible.
- With sparse features we can generalize better.



Sparsely Regulated Autoencoders

a_j is defined to be the activation of the jth hidden unit (bottleneck) of the autoencoder.

Let $a_j(x)$ be the activation of this specific node on a given input x.

Sparsely Regulated Autoencoders

Further let,

$$\hat{\rho}_j = \frac{1}{m}\sum_{i=1}^{m}\left[a_j\left(x^{(i)}\right)\right]$$

be the average activation of hidden unit j (over the training set).

Thus we would like to force the constraint:

$$\hat{\rho}_j = \rho$$

where ρ is a “sparsity parameter”, typically small. In other words, we want the average activation of each neuron j to be close to ρ.

Sparsely Regulated Autoencoders

- We need to penalize $\hat{\rho}_j$ for deviating from ρ.
- Many choices of the penalty term will give reasonable results.

- For example: $$\sum_{j=1}^{Bn} KL\left(\rho|\hat{\rho}_j\right)$$

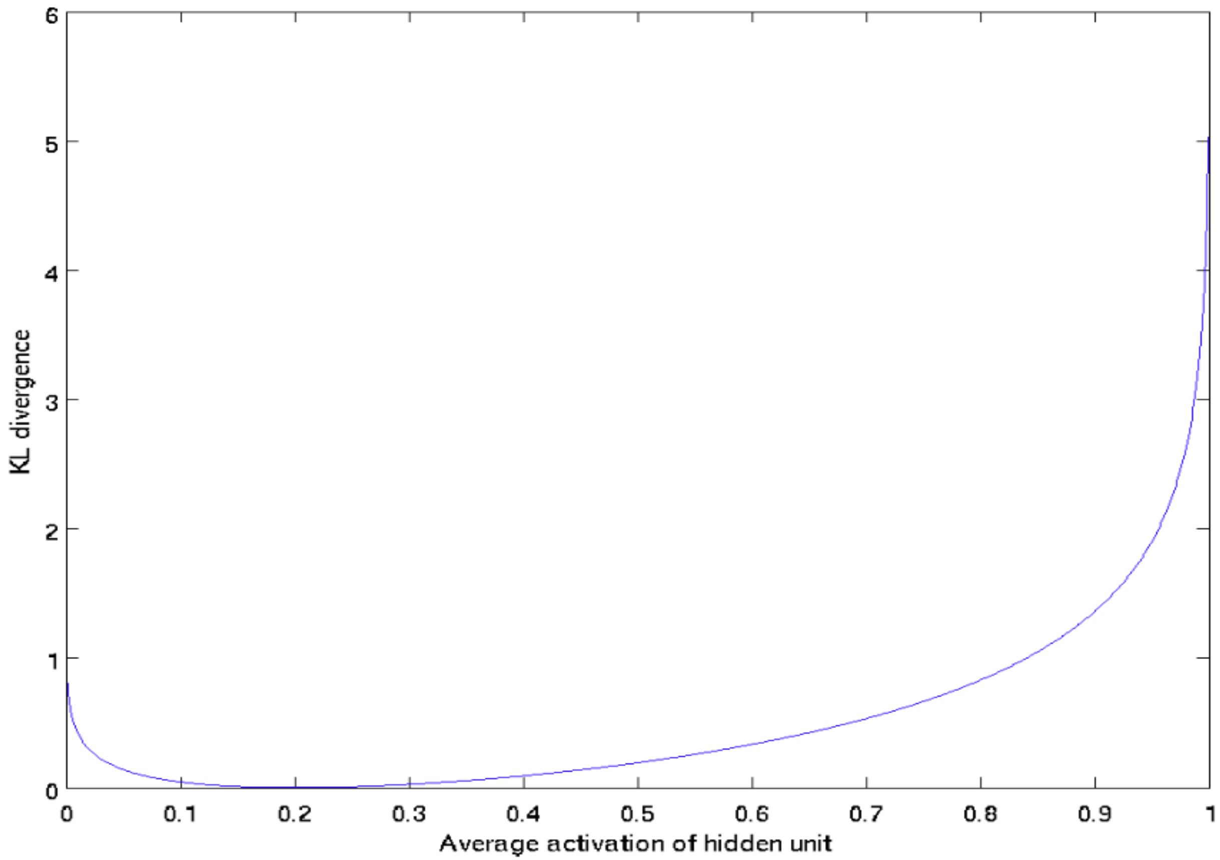
where $KL\left(\rho|\hat{\rho}_j\right)$ is a Kullback-Leibler divergence function.

Sparsely Regulated Autoencoders

- A reminder:
  - KL is a standard function for measuring how different two distributions are, which has the properties:

$$KL(\rho|\hat{\rho}_j) = 0 \text{ if } \hat{\rho}_j = \rho$$

otherwise it is increased monotonically.



$\rho = 0.2$

Sparsely Regulated Autoencoders

- Our overall cost functions is now:

$$J_S(W,b) = J(W,b) + \beta \sum_{j=1}^{Bn} KL\left(p|\hat{\rho}_j\right)$$

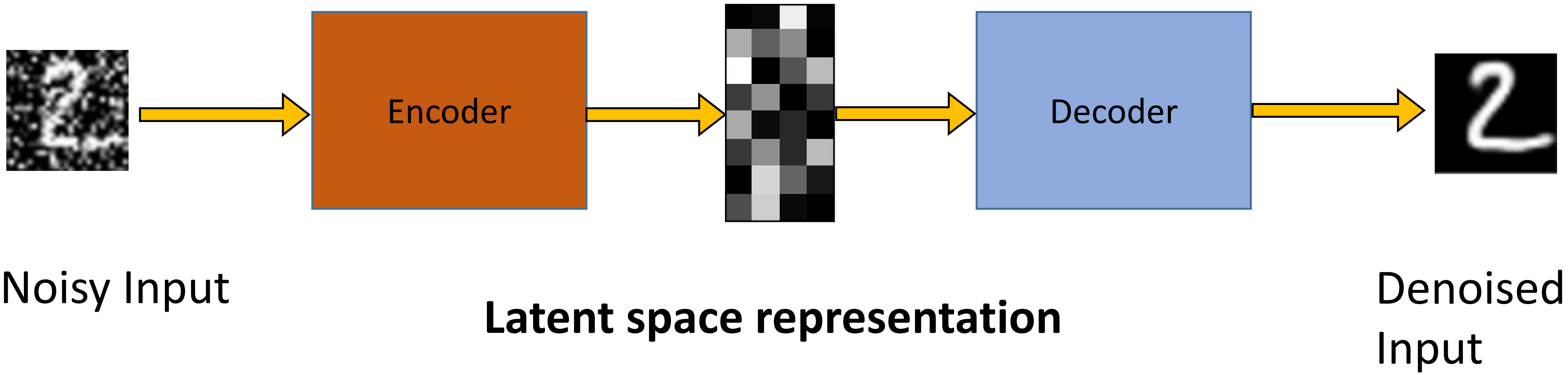
*Note: We need to know $\hat{\rho}_j$ before hand,
so we have to compute a forward pass on all the training set.

Denoising Autoencoders

<u>Intuition:</u>

- 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.

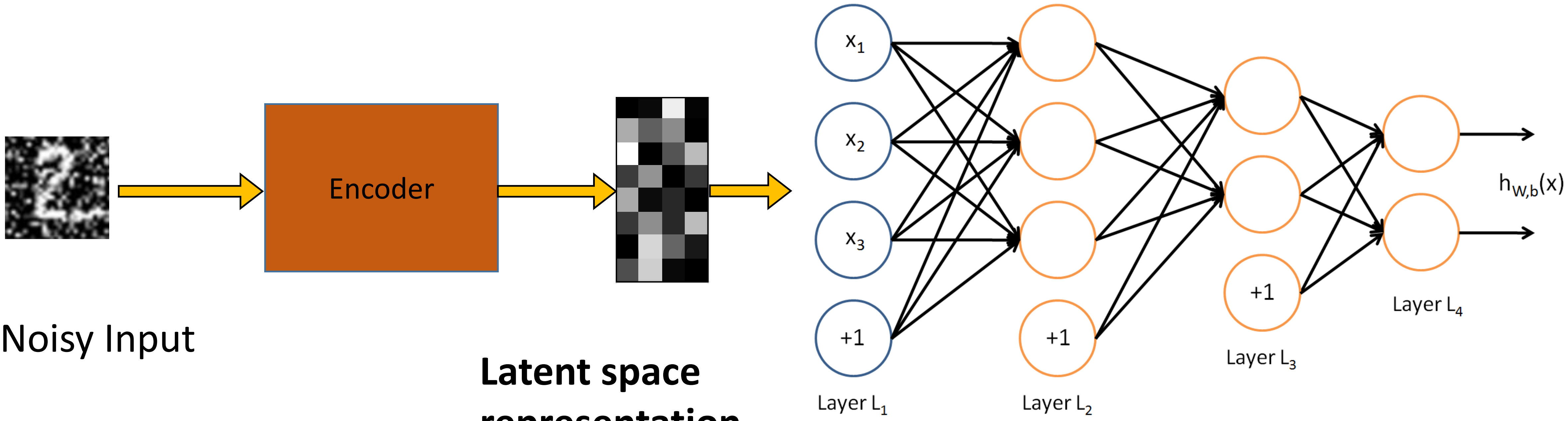
A more robust model



Denoising Autoencoders

Use Case:

- Extract robust representation for a NN classifier.



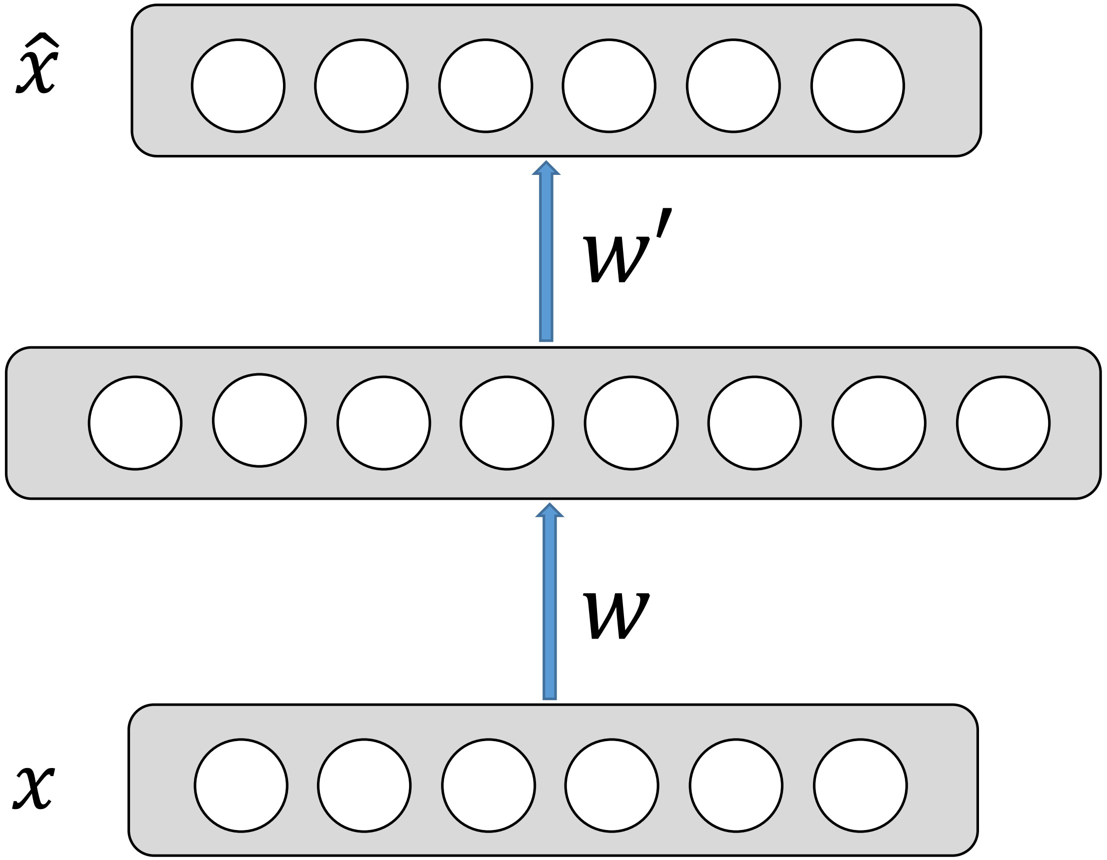
Noisy Input

Latent space representation

Contractive autoencoders

- 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 should maintain that property in the latent space.



Contractive autoencoders

- Definitions and reminders:
- - Frobenius norm (L2): $\|A\|_F = \sqrt{\Sigma_{i,j}|a_{ij}|^2}$
- - Jacobian Matrix: $J_f(x) = \dfrac{\partial f(x)}{\partial x} = \begin{bmatrix} \dfrac{\partial f(x)_1}{\partial x_1} & \cdots & \dfrac{\partial f(x)_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f(x)_m}{\partial x_1} & \cdots & \dfrac{\partial f(x)m}{\partial x_n} \end{bmatrix}$

Contractive autoencoders

- Our new loss function would be:
- $$L^*(x) = L(x) + \lambda\Omega(x)$$

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

and where λ controls the balance of our reconstruction objective and the hidden layer “flatness”.

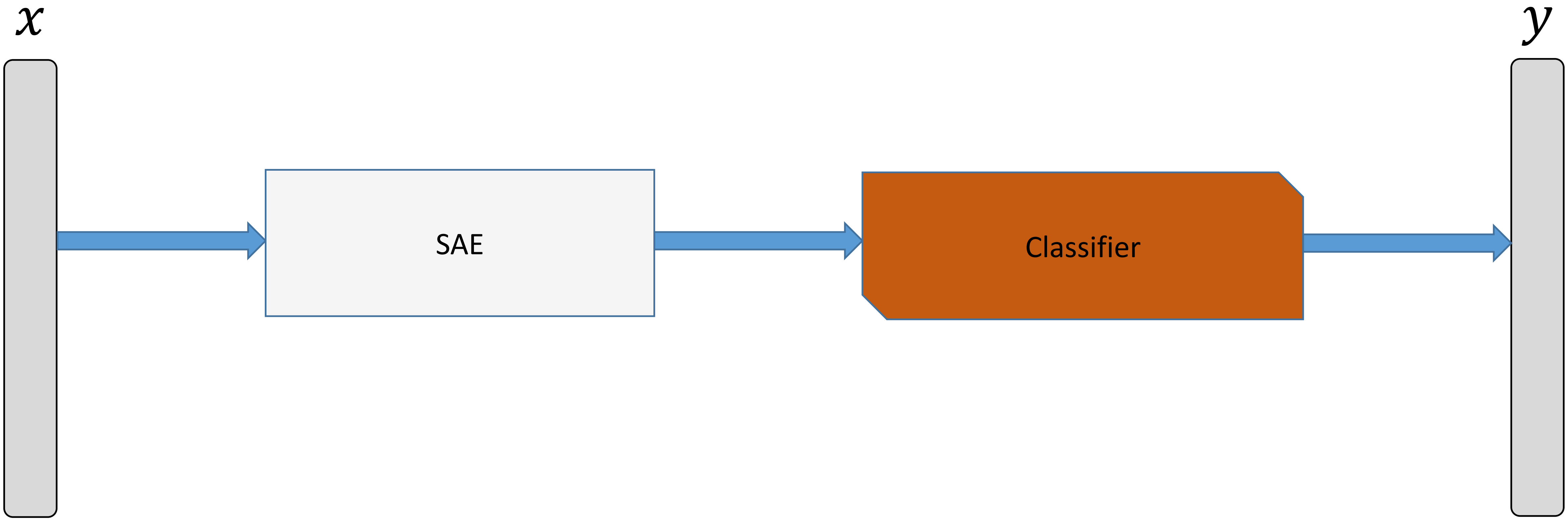
$$\boxed{\begin{aligned} Z_j &= W_i X_i \\ h_j &= \phi(Z_j) \end{aligned}}$$

$$\begin{aligned} \frac{\partial h_j}{\partial X_i} &= \frac{\partial \phi(Z_j)}{\partial X_i} \\ &= \frac{\partial \phi(W_i X_i)}{\partial W_i X_i} \frac{\partial W_i X_i}{\partial X_i} \\ &= [\phi(W_i X_i)(1 - \phi(W_i X_i))] \, W_i \\ &= [h_j(1 - h_j)] \, W_i \end{aligned}$$

$$\boxed{\frac{\partial h}{\partial X} = diag[h(1 - h)] \, W^T}$$

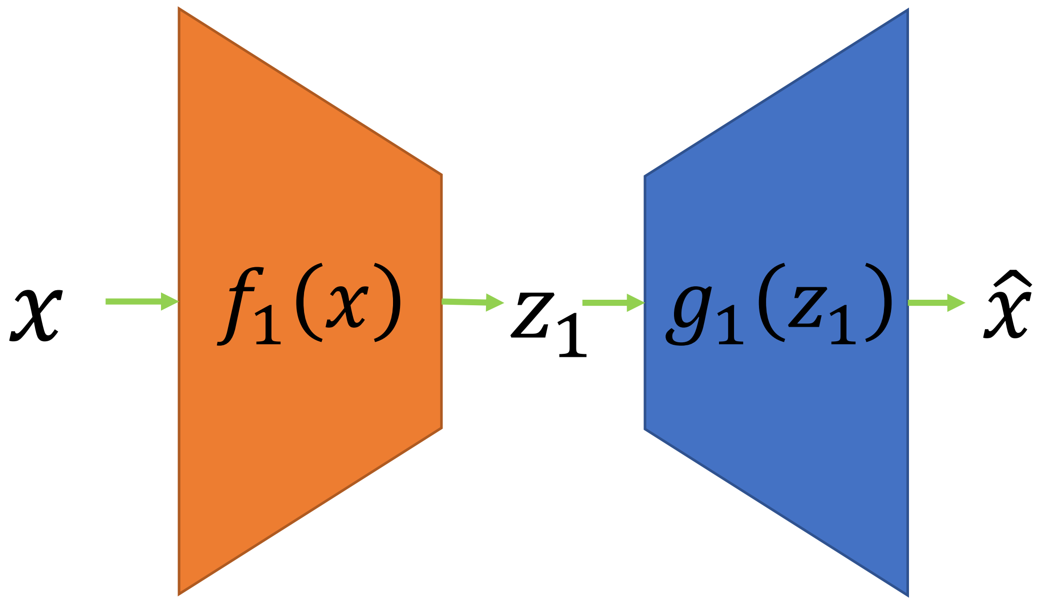
$$\boxed{\begin{aligned} \|J_h(X)\|_F^2 &= \sum_{ij} \left(\frac{\partial h_j}{\partial X_i} \right)^2 \\ &= \sum_i \sum_j [h_j(1 - h_j)]^2 (W_{ji}^T)^2 \\ &= \sum_j [h_j(1 - h_j)]^2 \sum_i (W_{ji}^T)^2 \end{aligned}}$$

Stacked AE



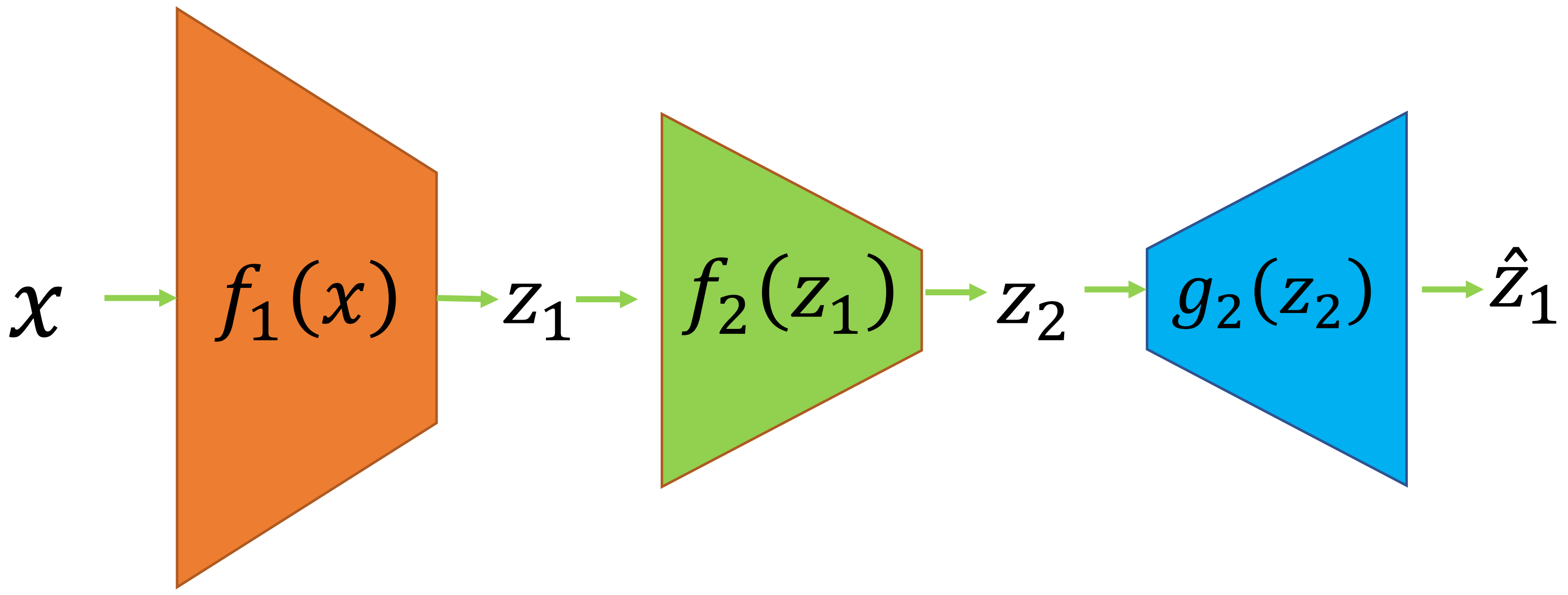
Stacked AE – train process

First Layer Training (AE 1)



Stacked AE – train process

Second Layer Training (AE 2)



Stacked AE – train process

Add any classifier

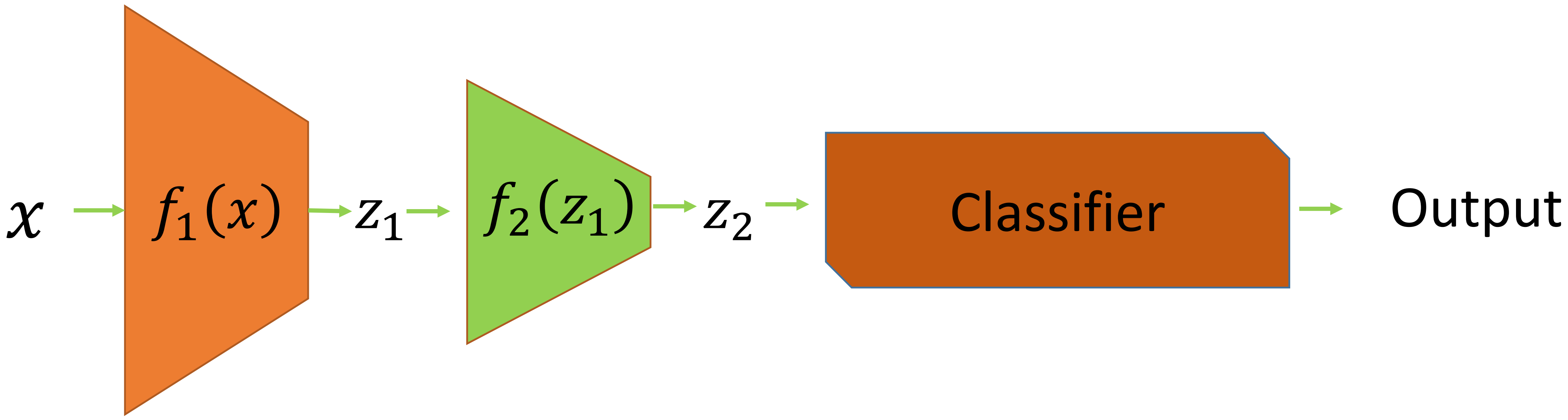


Image segmentation - segnet

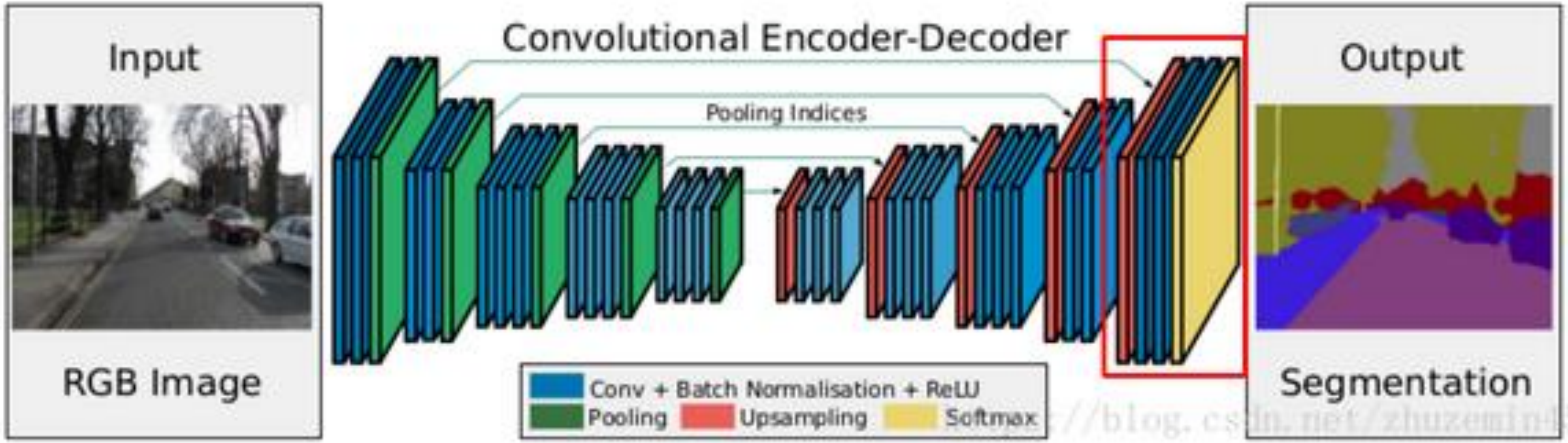


Image segmentation U-net

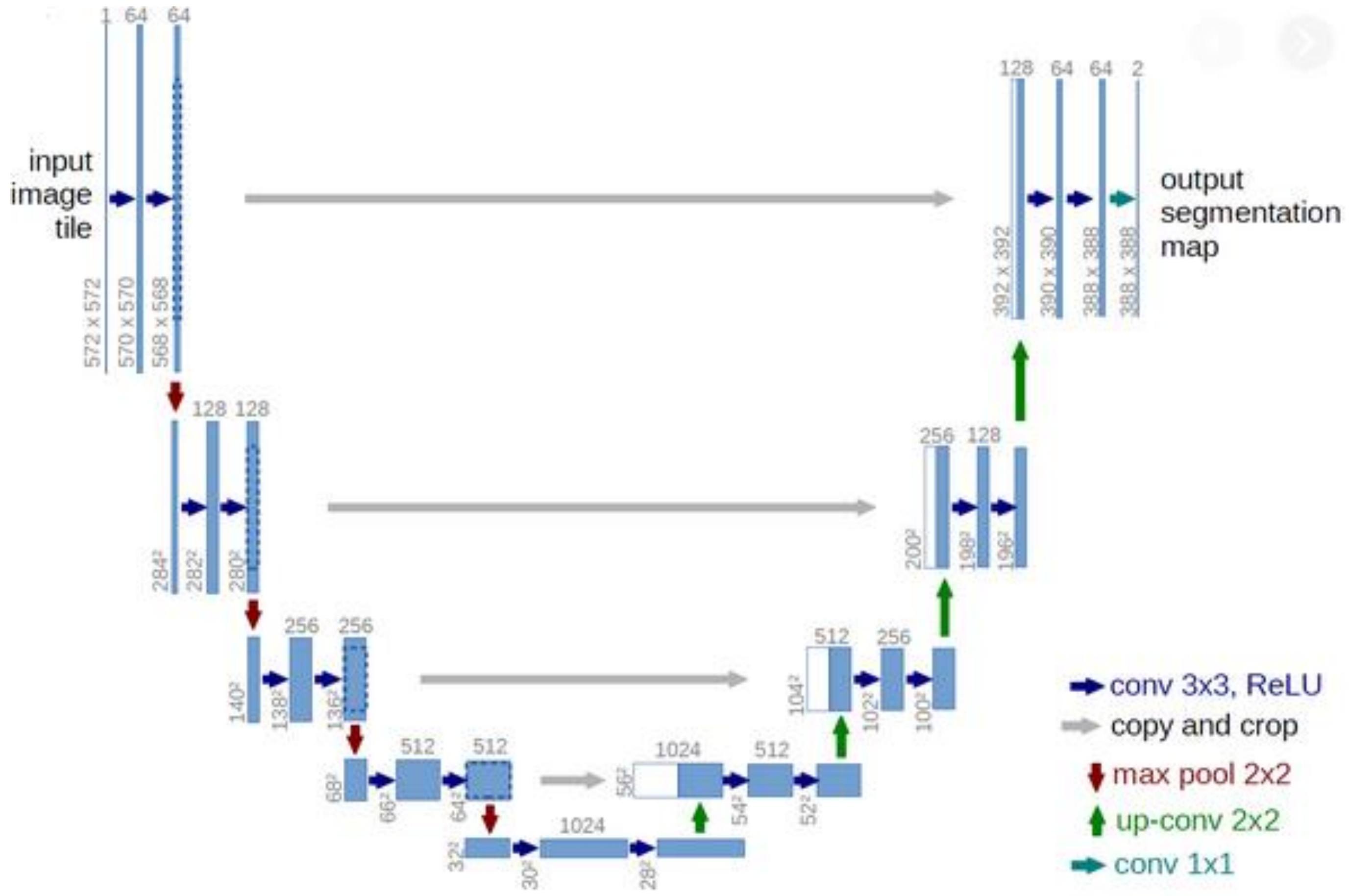
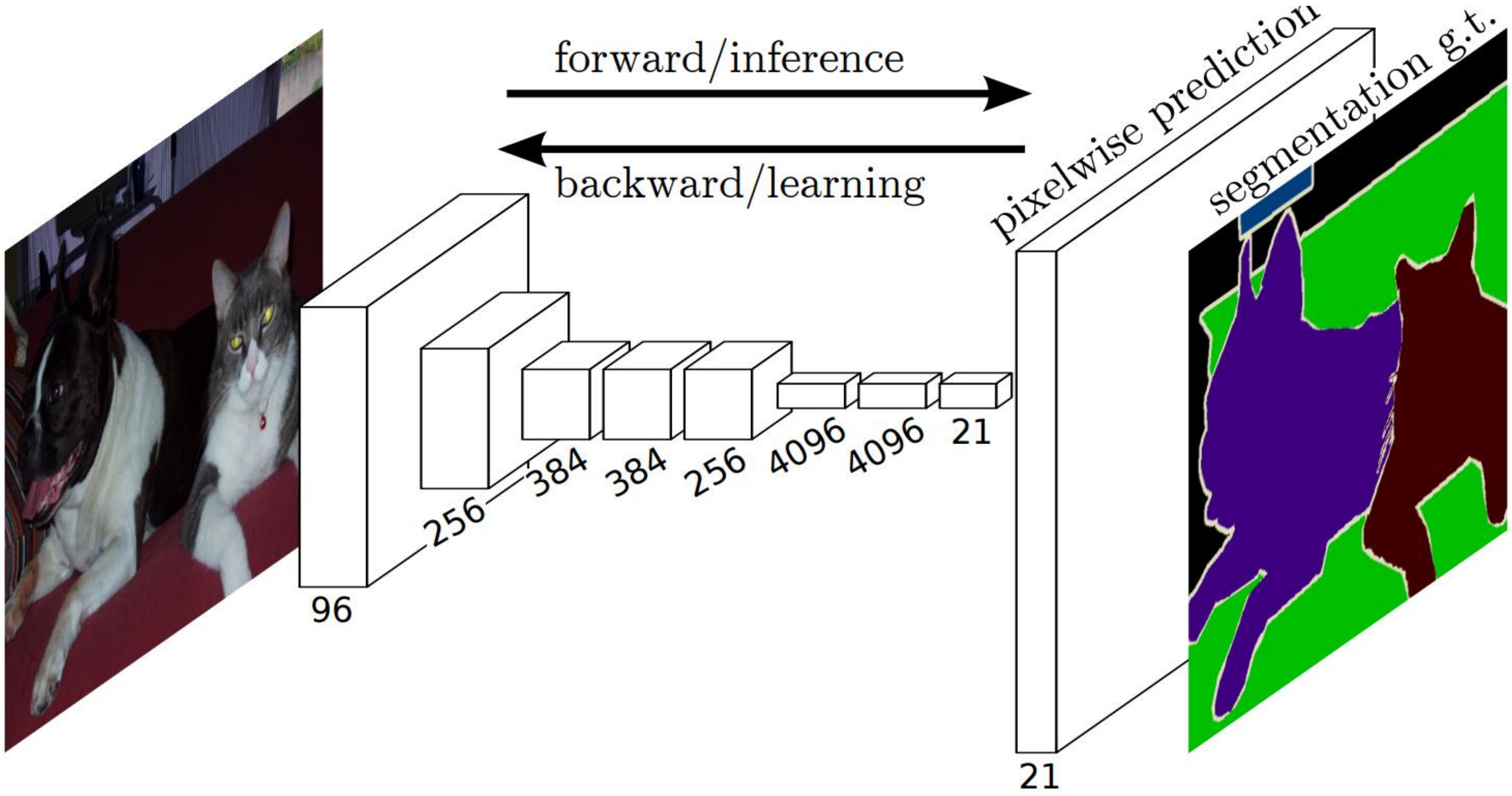


Image segmentation FCN



Instance segmentation