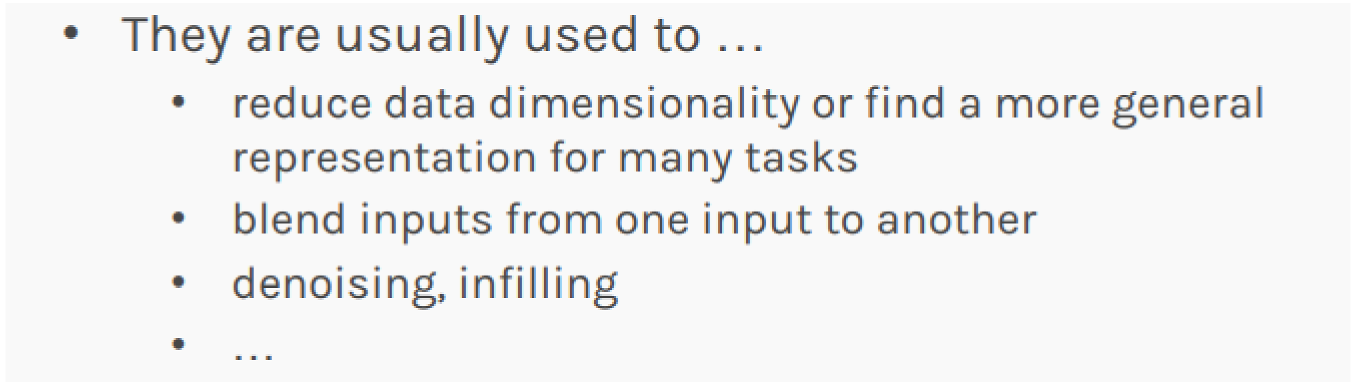
Autoencoders

Autoencoder

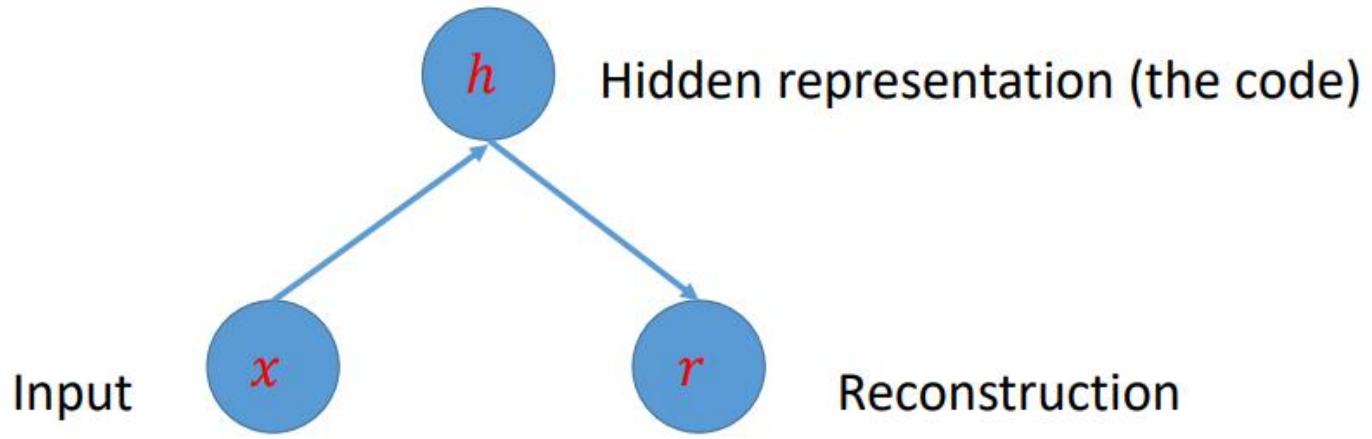
- Neural networks trained to attempt to copy its input to its output

- Contain two parts:
  - Encoder: map the input to a hidden representation
  - Decoder: map the hidden representation to the output



- 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
  - …

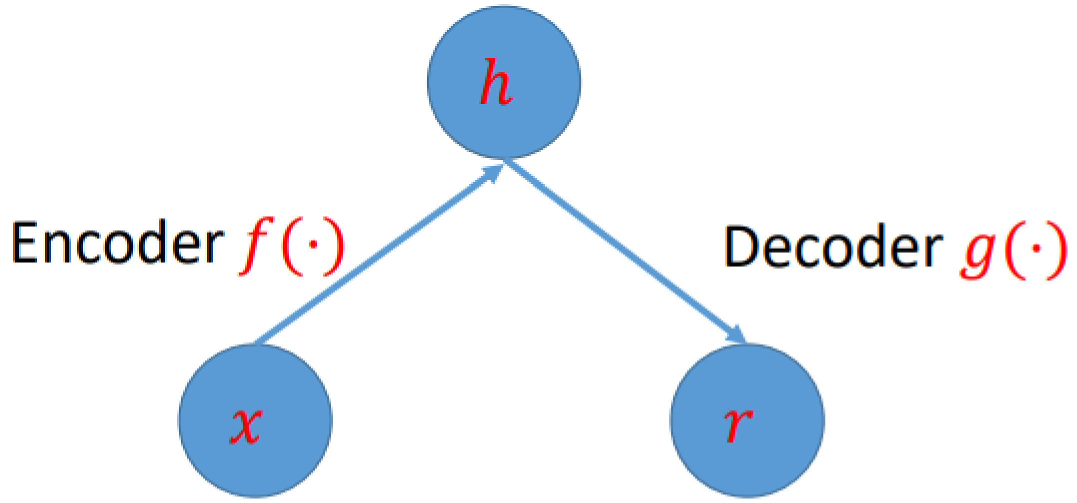
Autoencoder



Given an image of a handwritten digit, an autoencoder first encodes the image into a lower dimensional latent representation, then decodes the latent representation back to an image.

An autoencoder learns to compress the data while minimizing the reconstruction error.

Autoencoder



$$h = f(x), r = g(h) = g(f(x))$$

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

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

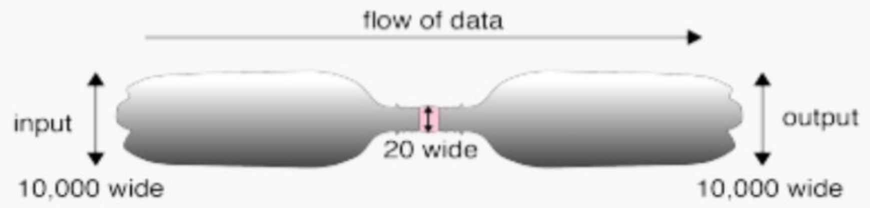
Our goal is to find an estimate of $f(x)$ which we will call $\hat{f}(x)$.

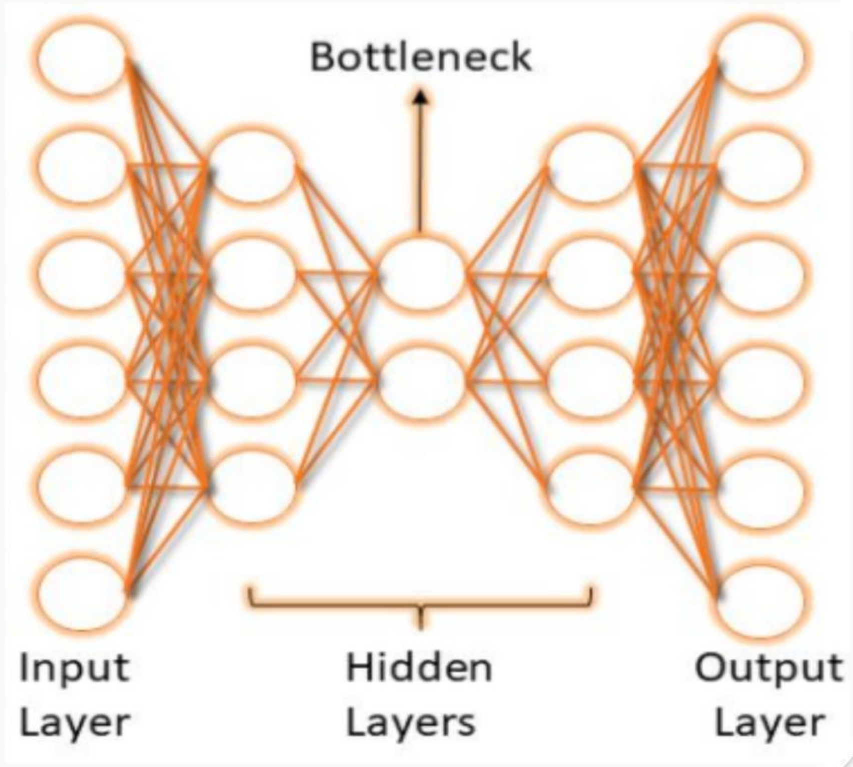
Statistical learning or modeling is the process of finding $\hat{f}(x)$.

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

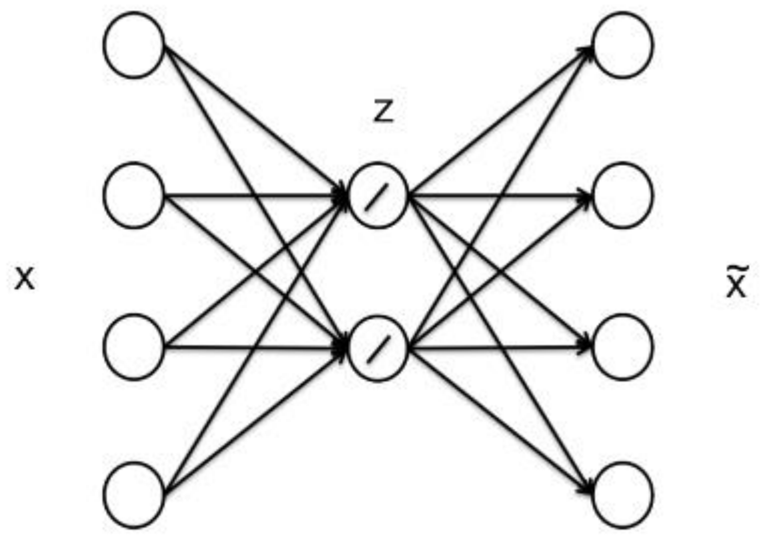
Bottleneck

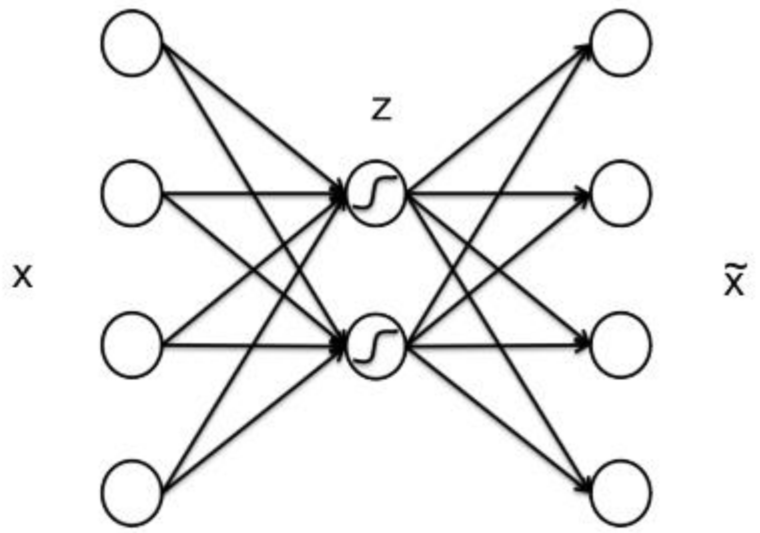
- We start with 10,000 elements
- We have 20 in the middle
- And 10,000 elements again at the end





Linear vs Non- linear Auto encoder



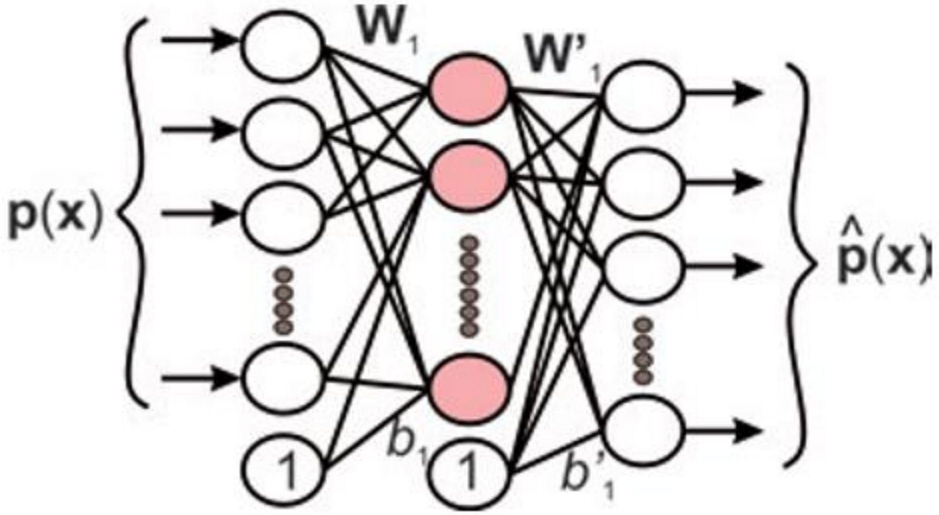


Here, we are trying to map data from 4 dimensions to 2 dimensions using a neural network with one hidden layer. The activation function of the hidden layer is linear and hence the name linear autoencoder.
It works for the case that the data lie on a linear surface.

If the data lie on a nonlinear surface, it makes more sense to use a nonlinear autoencoder.

If the data is highly nonlinear, one could add more hidden layers to the network to have a deep autoencoder

Autoencoder



$$\mathrm{y} = [\mathrm{w}\ \ \mathrm{b}].[\mathrm{p}\ ;\ 1]$$

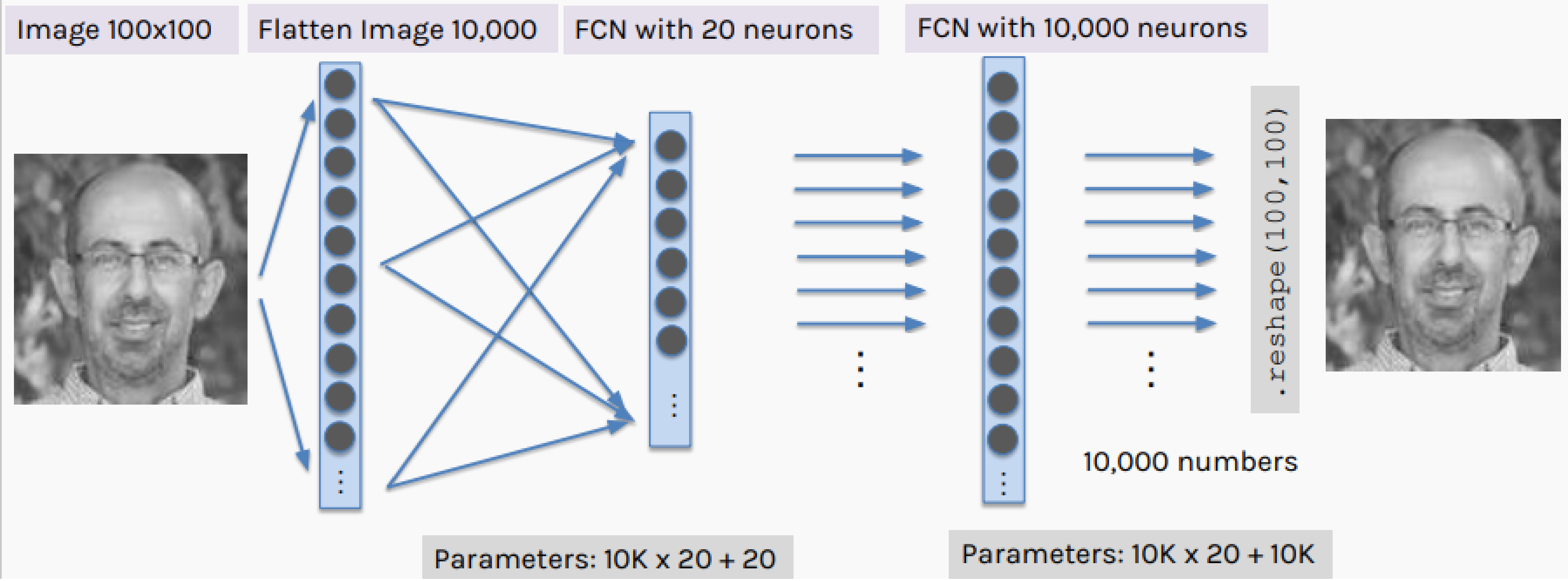
$$\hat{\mathrm{p}} = [\mathrm{w}'\ \ \mathrm{b}'].[\mathrm{y}\ ;\ 1]$$

$$J(\mathbf{W}) = \sum_n \|\mathbf{p}_n - \hat{\mathbf{p}}_n\|$$

$$\{\mathrm{w}, \mathrm{b}, \mathrm{w}', \mathrm{b}'\} = \underset{\{\mathbf{w},\mathbf{b},\mathbf{w}',\mathbf{b}'\}}{\arg\min}\ (J(\mathbf{W}))$$

The simplest autoencoder

Encode and decode together after training



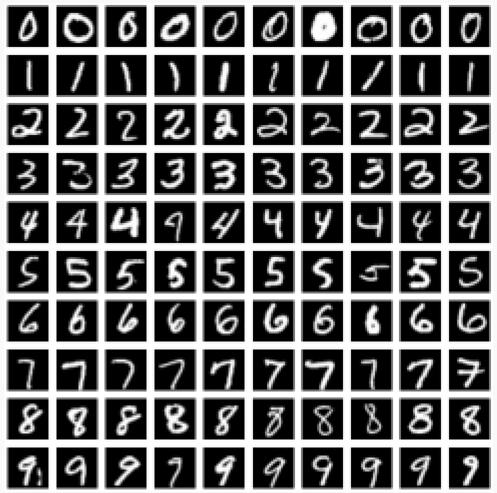
Latent variables and latent layer

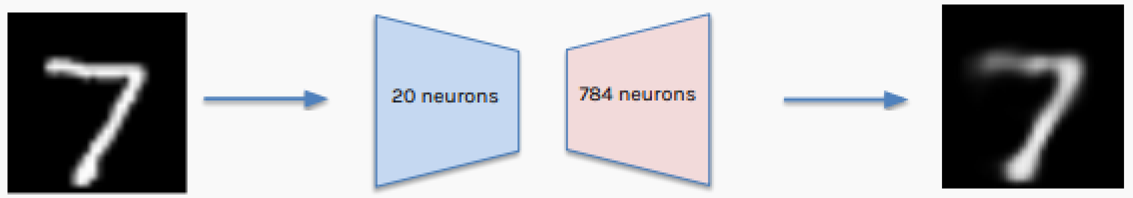
We say that an autoencoder is an example of **semi-supervised or self-supervised** learning.

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.

A better autoencoder

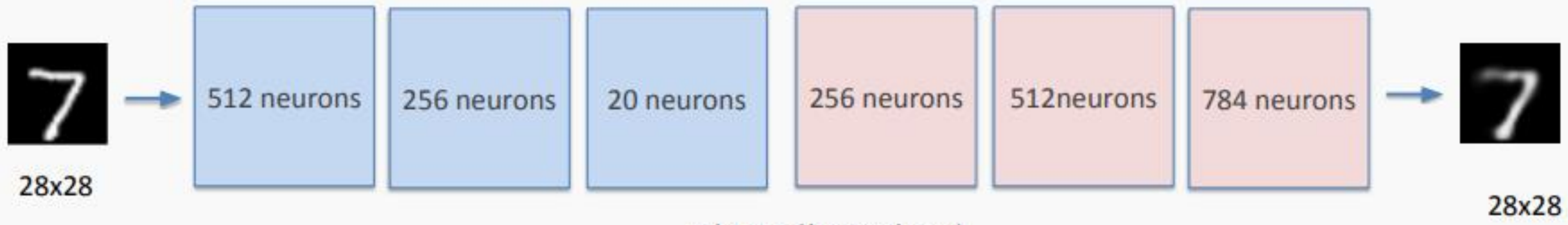
MNIST data: train a simple AE with one-layer FCN encoder and one-layer FCN decoder



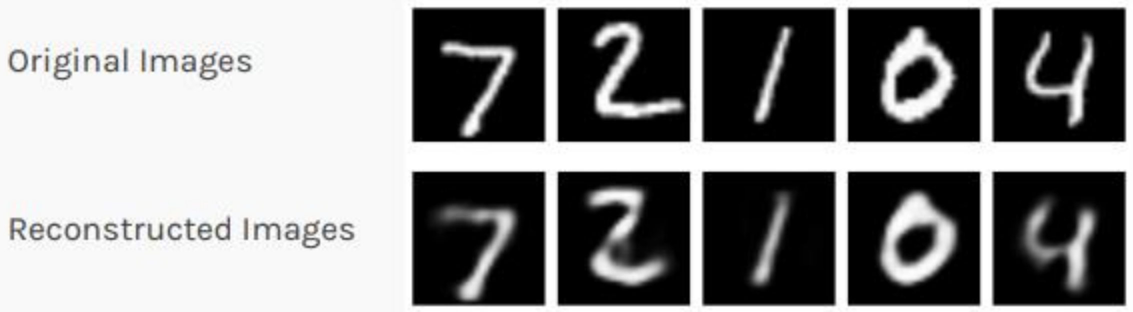


Deeper

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

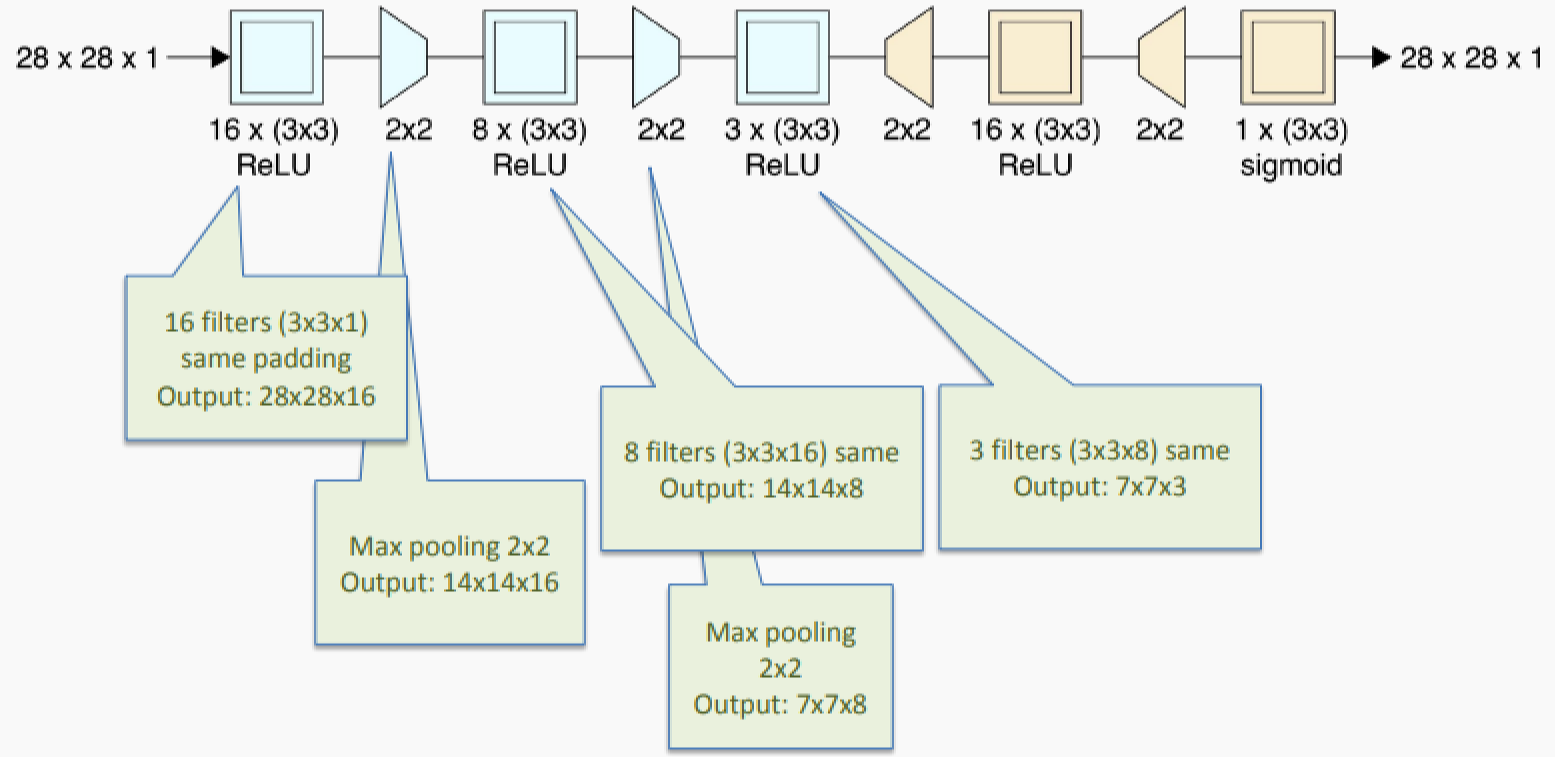


Z is 20 dimensional

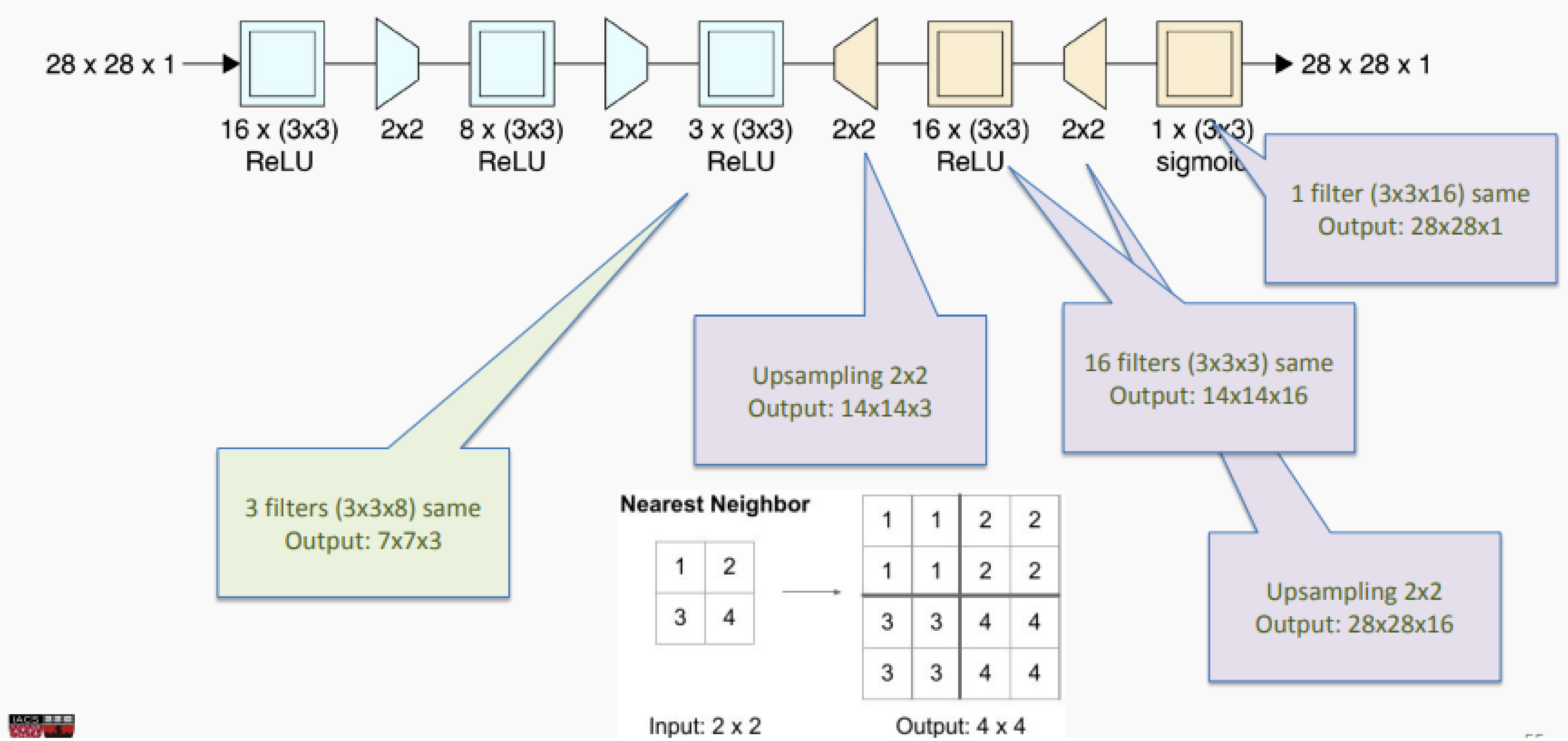


Deeper 20 latent variables
Shallow 20 latent variables

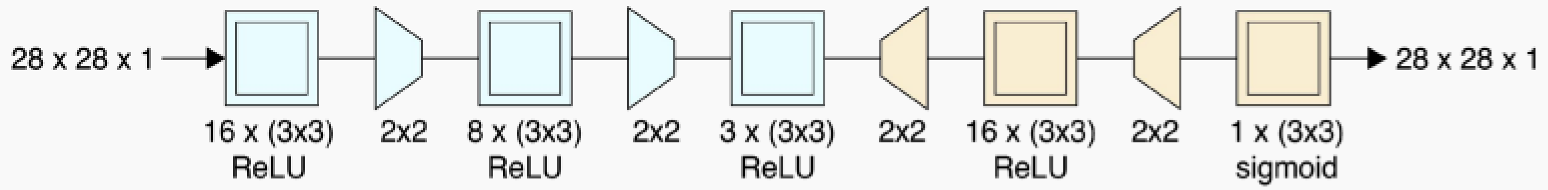
Convolutional Autoencoders


28 x 28 x 1
16 x (3x3) ReLU
2x2
8 x (3x3) ReLU
2x2
3 x (3x3) ReLU
2x2
16 x (3x3) ReLU
2x2
1 x (3x3) sigmoid
28 x 28 x 1
16 filters (3x3x1) same padding Output: 28x28x16
Max pooling 2x2 Output: 14x14x16
8 filters (3x3x16) same Output: 14x14x8
Max pooling 2x2 Output: 7x7x8
3 filters (3x3x8) same Output: 7x7x3

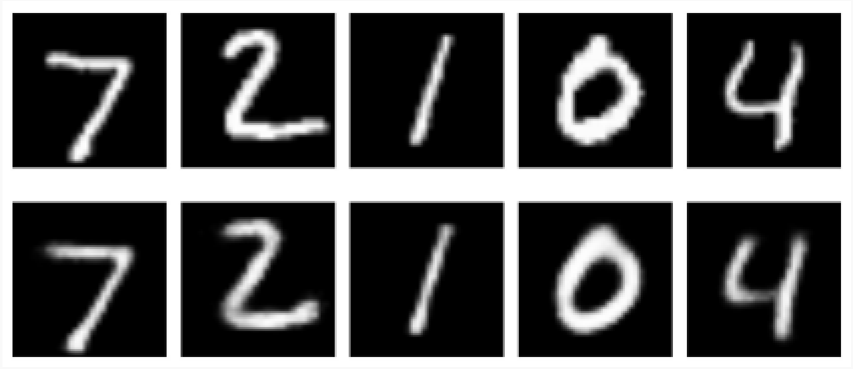
Convolutional Autoencoders



Convolutional Autoencoders



Original Images



Reconstructed Images with Conv AE

ConvAE latent variables

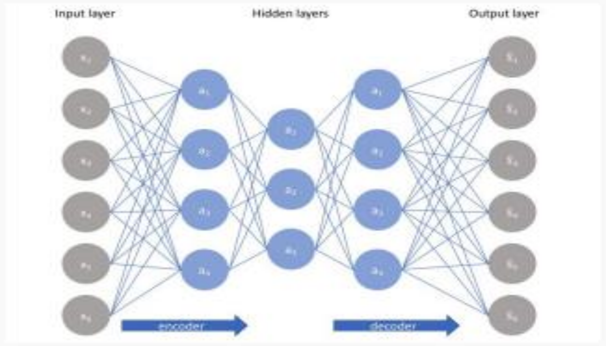
DeepFCN latent variables

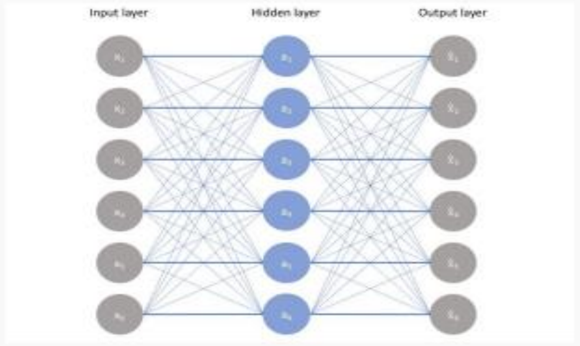
The ideal autoencoder model balances the following:

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

Overcomplete Autoencoders

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

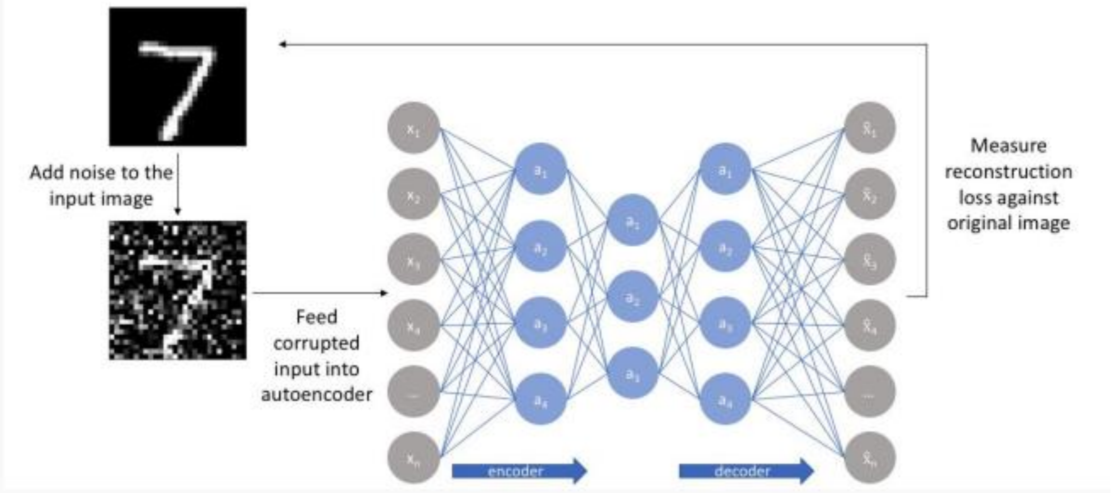




Denoising Autoencoders

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

For each epoch:


Add noise to the input image
Feed corrupted input into autoencoder
Measure reconstruction loss against original image
encoder
decoder

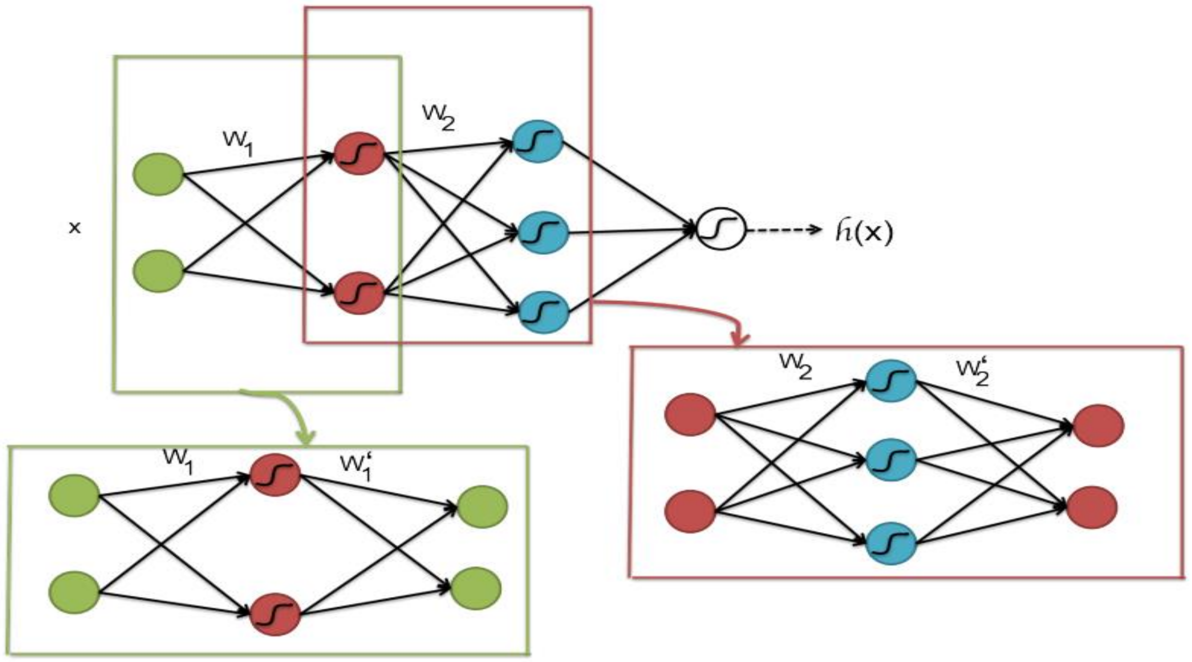
Autoencoders as an initialization method for MLP

- Autoencoders could be used as a way to “pretrain” neural networks. Why? The reason is that training very deep neural networks is difficult

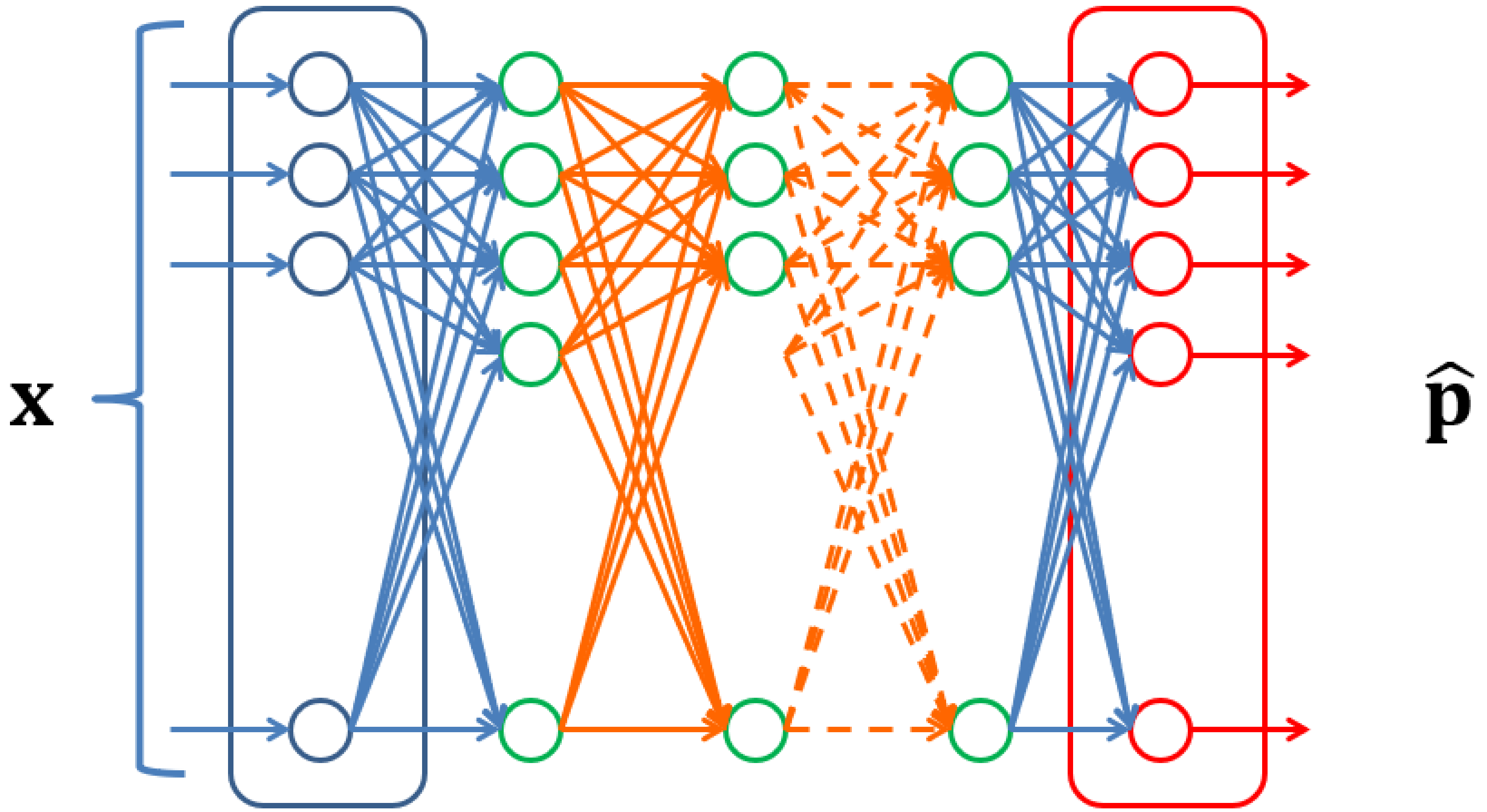
  - The magnitudes of gradients in the lower layers and in higher layers are different,
  - The landscape or curvature of the objective function is difficult for stochastic gradient descent to find a good local optimum,
  - Deep networks have many parameters, which can remember training data and do not generalize well.

The goal of pretraining is to address the above problems. With pretraining, the process of training a deep network is divided in a sequence of steps:

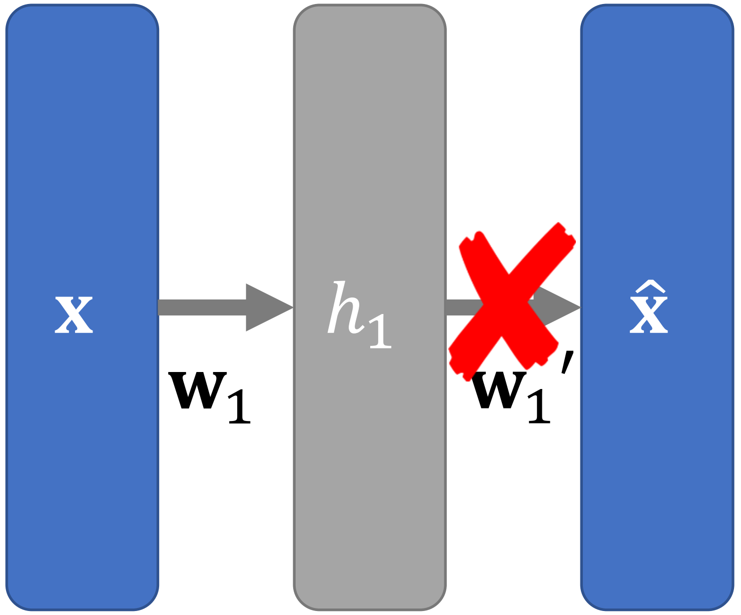
- Pretraining step: train a sequence of shallow autoencoders, greedily one layer at a time, using unsupervised data,
- Fine-tuning step 1: train the last layer using supervised data,
- Fine-tuning step 2: use backpropagation to *fine-tune* the entire network using supervised data.



Autoencoder for MLP Initialization



Autoencoding one Layer at a Time

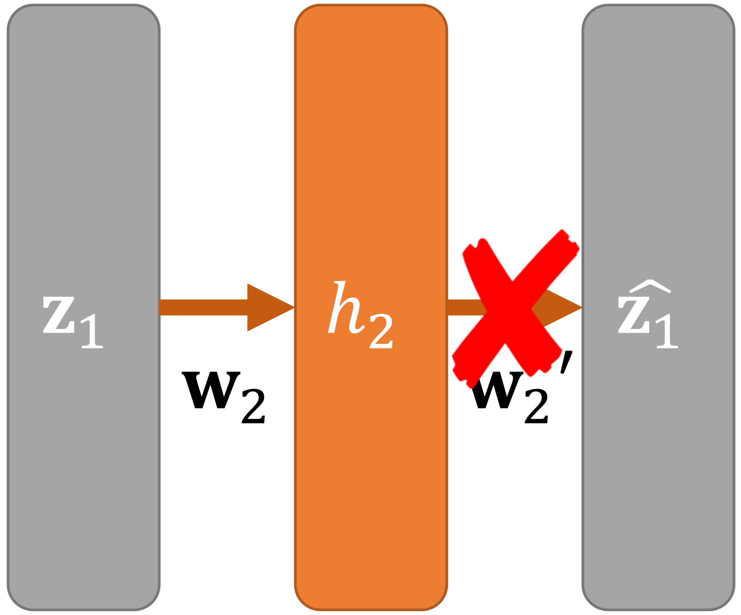


$$\{\mathbf{w}_1, \mathbf{w}_1{}'\} = \operatorname{argmin}\{J(\mathbf{w})\}$$

$$J(\mathbf{w}) = ||\mathbf{x} - \hat{\mathbf{x}}||$$

$$\mathbf{z}_1 = f_{NL}(\mathbf{w}_1 \cdot [\mathbf{x}, 1]^T)$$

Stacking others One Layer at a Time

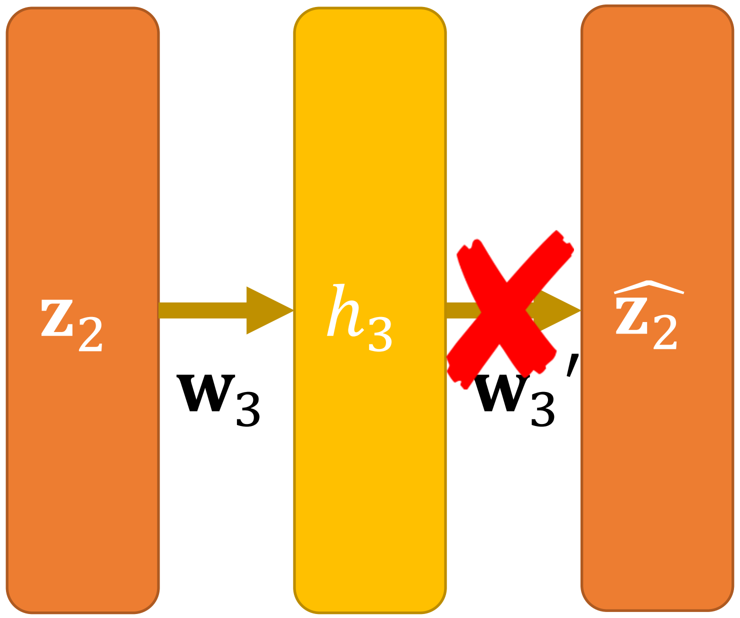


$$\{\mathbf{w}_2, \mathbf{w}_2{}'\} = \text{argmin}\{J(\mathbf{w})\}$$

$$J(\mathbf{w}) = \|\mathbf{z}_1 - \widehat{\mathbf{z}_1}\|$$

$$\mathbf{z}_2 = f_{NL}(\mathbf{w}_2 \cdot [\mathbf{z}_1, 1]^T)$$

Stacking others One Layer at a Time

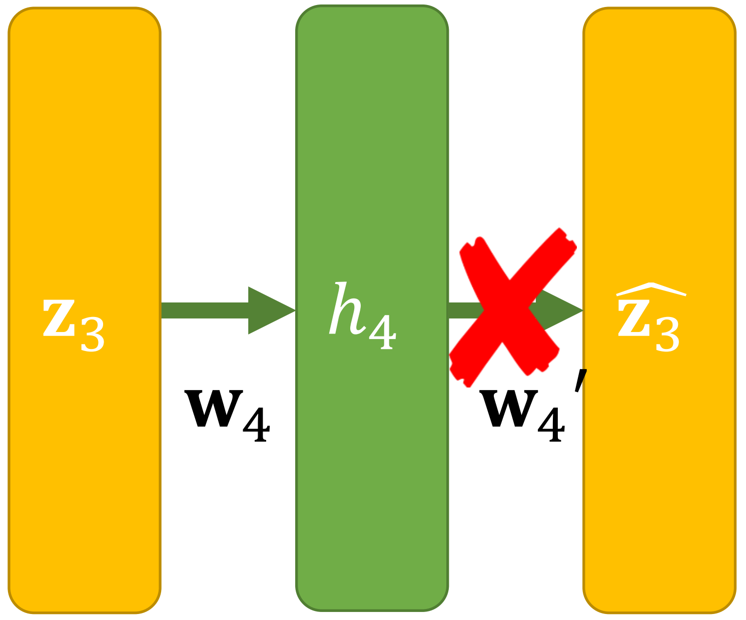


$$\{\mathbf{w}_3, \mathbf{w}_3{}'\} = \operatorname{argmin}\{J(\mathbf{w})\}$$

$$J(\mathbf{w}) = ||\mathbf{z}_2 - \widehat{\mathbf{z}_2}||$$

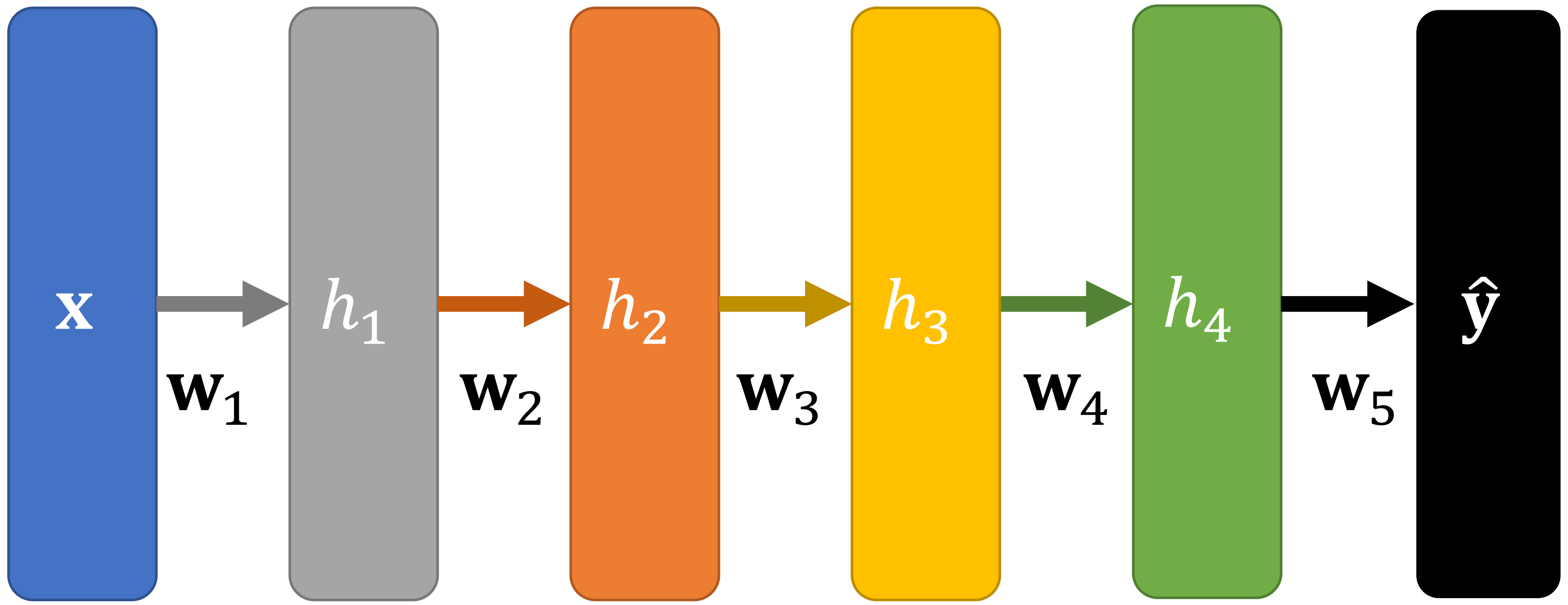
$$\mathbf{z}_3 = f_{NL}(\mathbf{w}_3 \cdot [\mathbf{z}_2, 1]^T)$$

Stacking others One Layer at a Time



$$\{\mathbf{w}_4, \mathbf{w}_4{}'\} = \operatorname{argmin}\{J(\mathbf{w})\}$$
$$J(\mathbf{w}) = \|\mathbf{z}_3 - \widehat{\mathbf{z}_3}\|$$

Supervised Refinement of MLP

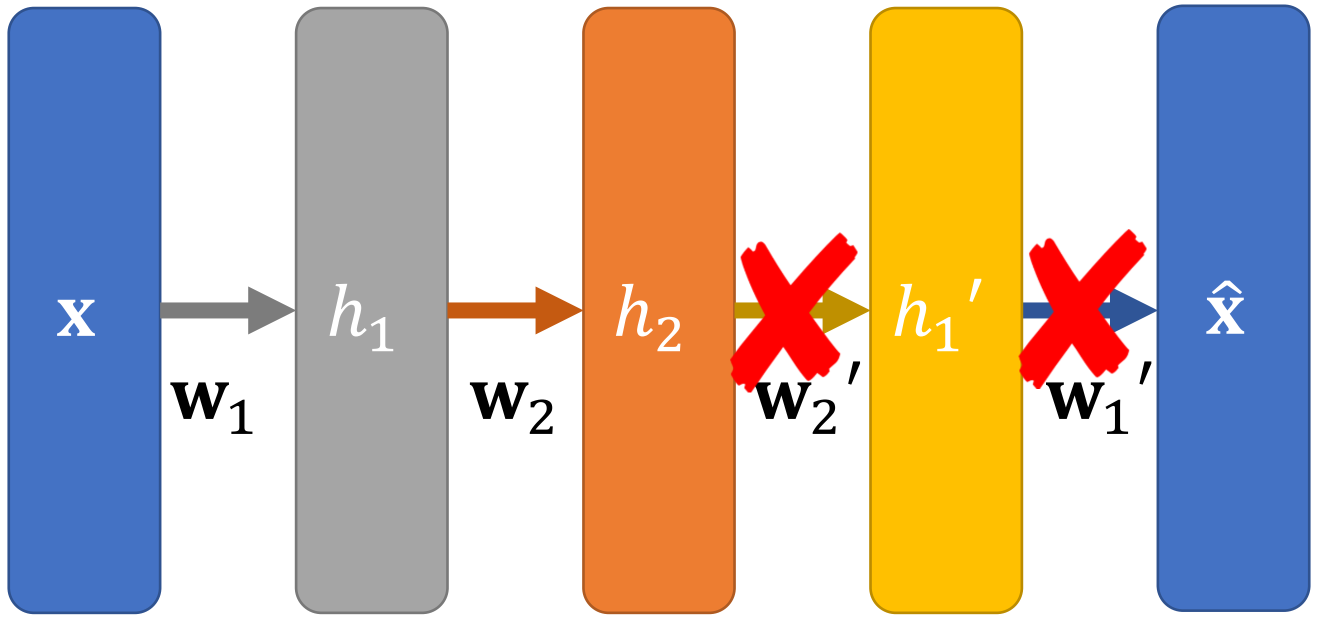


$$\{\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3, \mathbf{w}_4, \mathbf{w}_5\} = \operatorname{argmin}\{J(\mathbf{w})\}$$

$$J(\mathbf{w}) = \|\mathbf{y} - \hat{\mathbf{y}}\|$$

Another Technique

End-to-End Pre-training

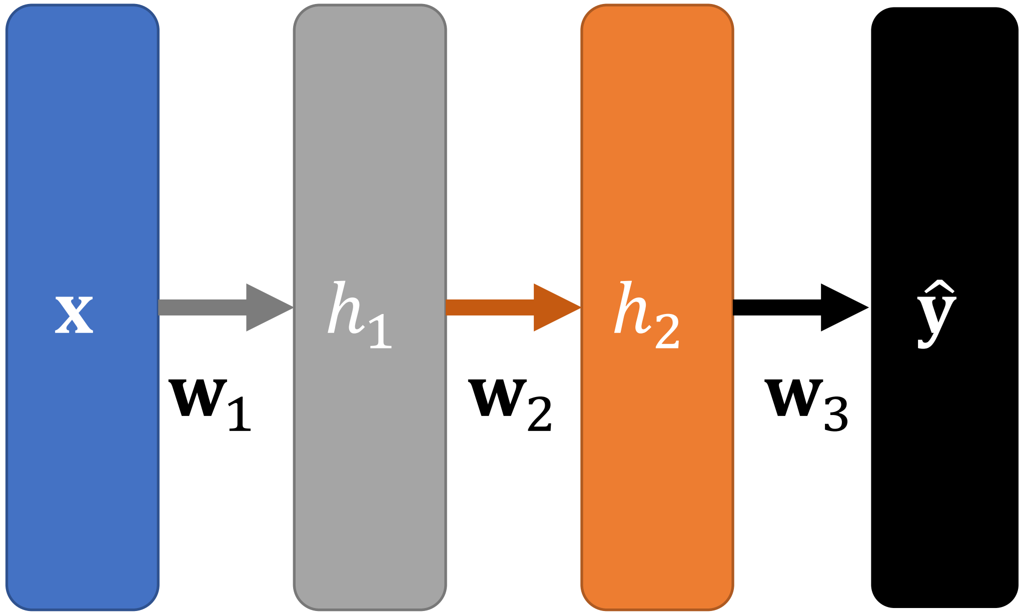


$$\{\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_2', \mathbf{w}_1'\} = \operatorname{argmin}\{J(\mathbf{w})\}$$

$$J(\mathbf{w}) = \|\mathbf{x} - \hat{\mathbf{x}}\|$$

$$\mathbf{z}_2 = f_{NL}\left(\mathbf{w}_2 \cdot [f_{NL}(\mathbf{w}_1 \cdot [\mathbf{x}, 1]), 1] \quad \right)$$

Supervised Refinement of MLP



$$\{\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3\} = \text{argmin}\{J(\mathbf{w})\}$$
$$J(\mathbf{w}) = \|\mathbf{y} - \hat{\mathbf{y}}\|$$