CISC6000 Deep Learning
Autoencoder

Dr. Yijun Zhao

Fordham University

Motivations

- **Pre-training for supervised learning**
- Data Encoding (Compression)
- Denoising
- Others

Traditional Neural Network

- Could have L hidden layers:
  - layer input activation for $k>0$ $(\mathbf{h}^{(0)}(\mathbf{x}) = \mathbf{x})$

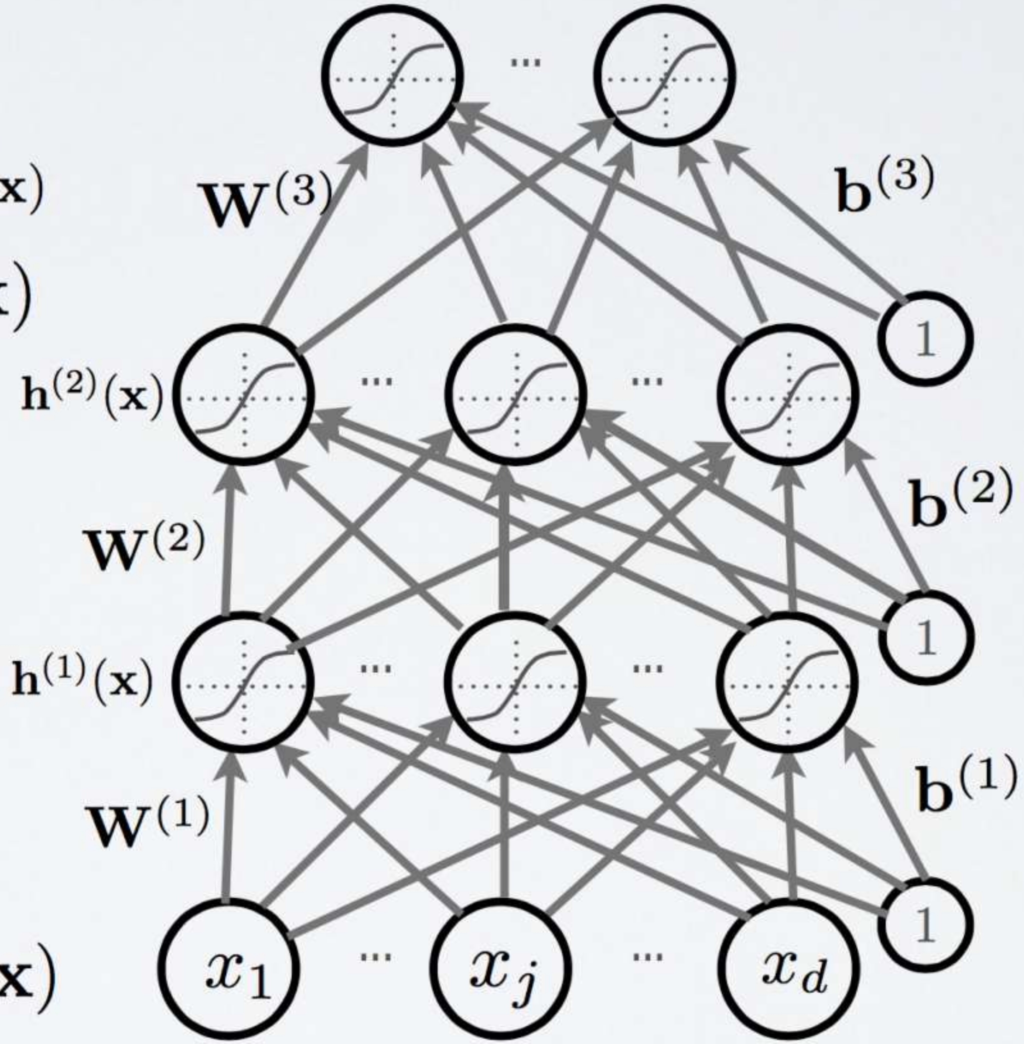
$$\mathbf{a}^{(k)}(\mathbf{x}) = \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\mathbf{h}^{(k-1)}(\mathbf{x})$$

  - hidden layer activation (k from 1 to L):

$$\mathbf{h}^{(k)}(\mathbf{x}) = \mathbf{g}(\mathbf{a}^{(k)}(\mathbf{x}))$$

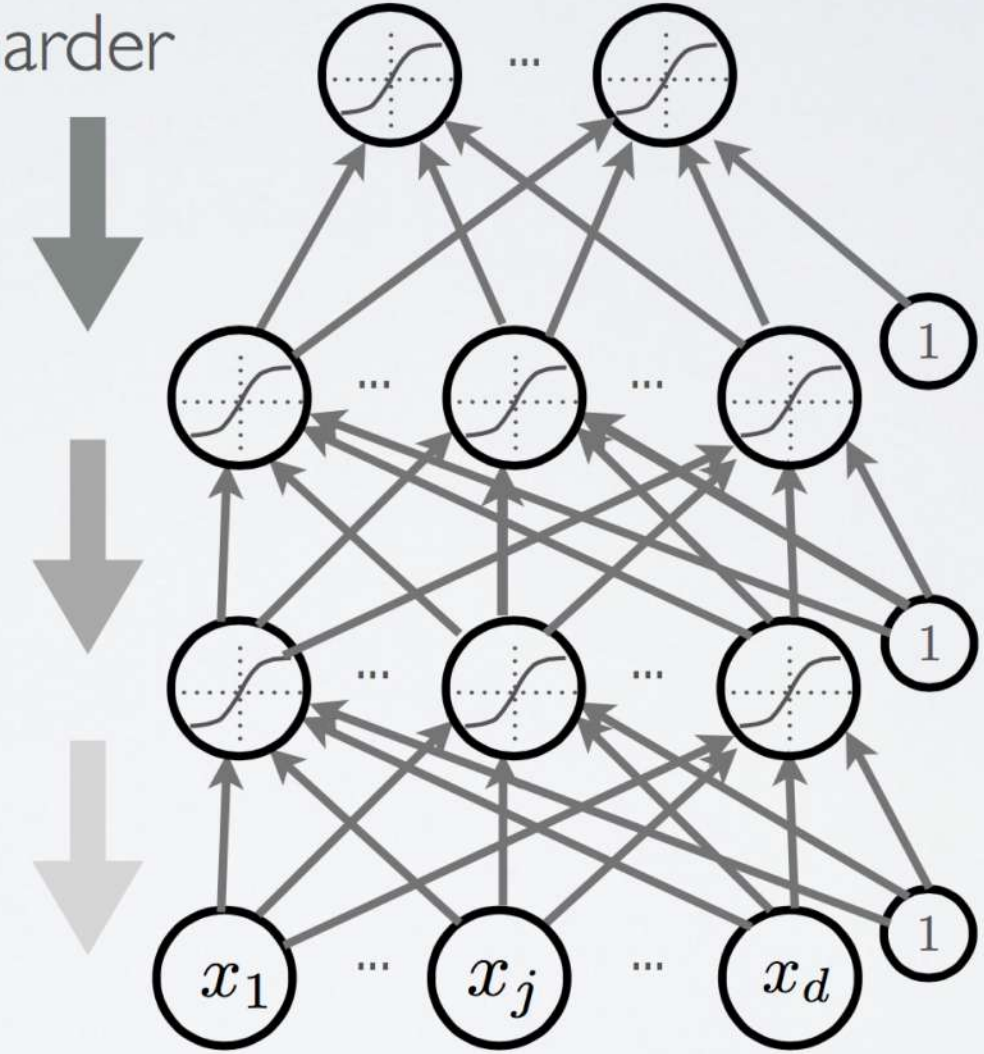
  - output layer activation ($k=L+1$):

$$\mathbf{h}^{(L+1)}(\mathbf{x}) = \mathbf{o}(\mathbf{a}^{(L+1)}(\mathbf{x})) = \mathbf{f}(\mathbf{x})$$



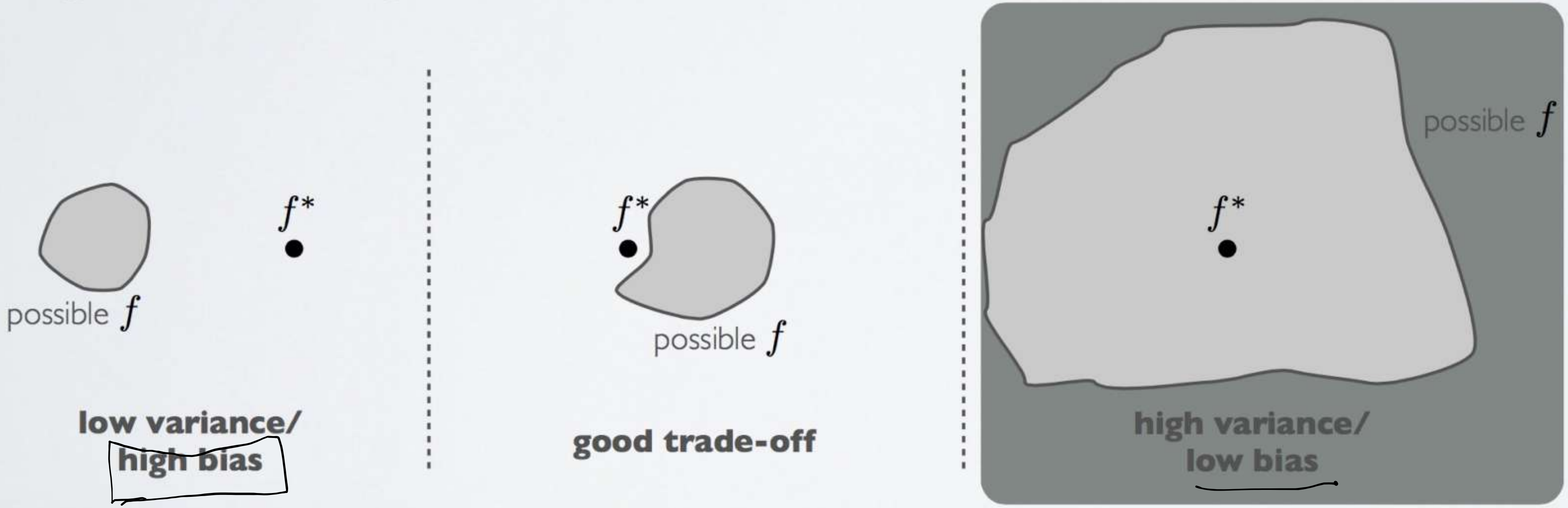
Why Training NN Is Hard?

- First hypothesis: optimization is harder (underfitting)
  - vanishing gradient problem
  - saturated units block gradient propagation
- This is a well known problem in recurrent neural networks



Why Training NN Is Hard?

- Second hypothesis: overfitting
  - we are exploring a space of complex functions
  - deep nets usually have lots of parameters
- Might be in a high variance / low bias situation



Why Training NN Is Hard?

- Depending on the problem, one or the other situation will tend to dominate

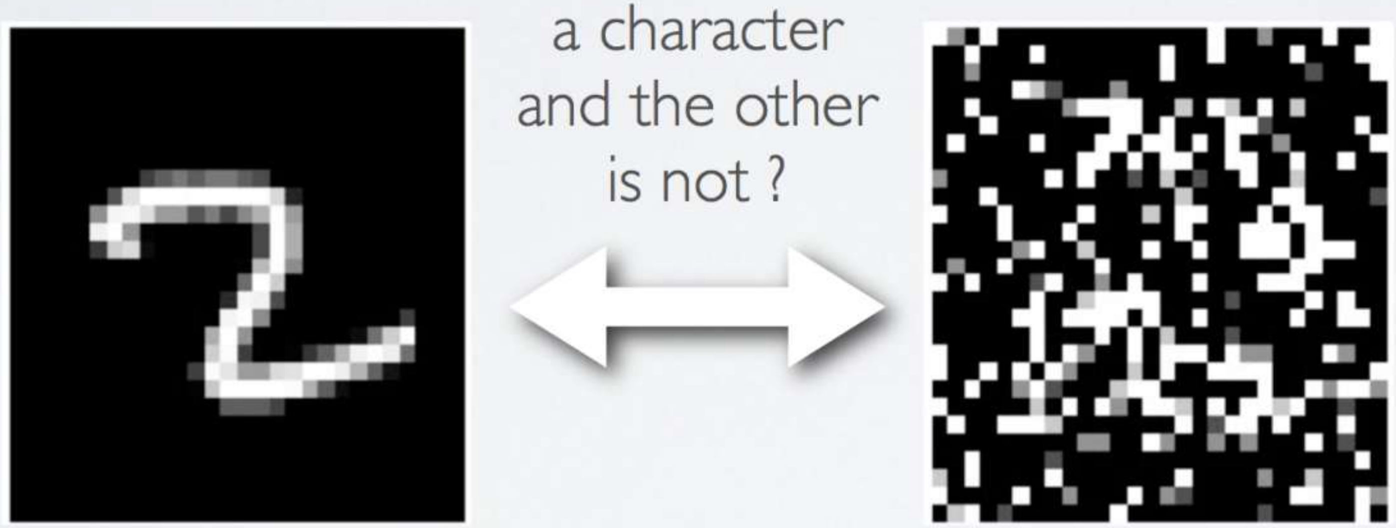
- If first hypothesis (underfitting): use better optimization
  - this is an active area of research

- If second hypothesis (overfitting): use better regularization
  - unsupervised learning
  - stochastic «dropout» training

Why Training NN Is Hard?

- Solution: initialize hidden layers using unsupervised learning
  - force network to represent latent structure of input distribution

Why is one
a character
and the other
is not ?



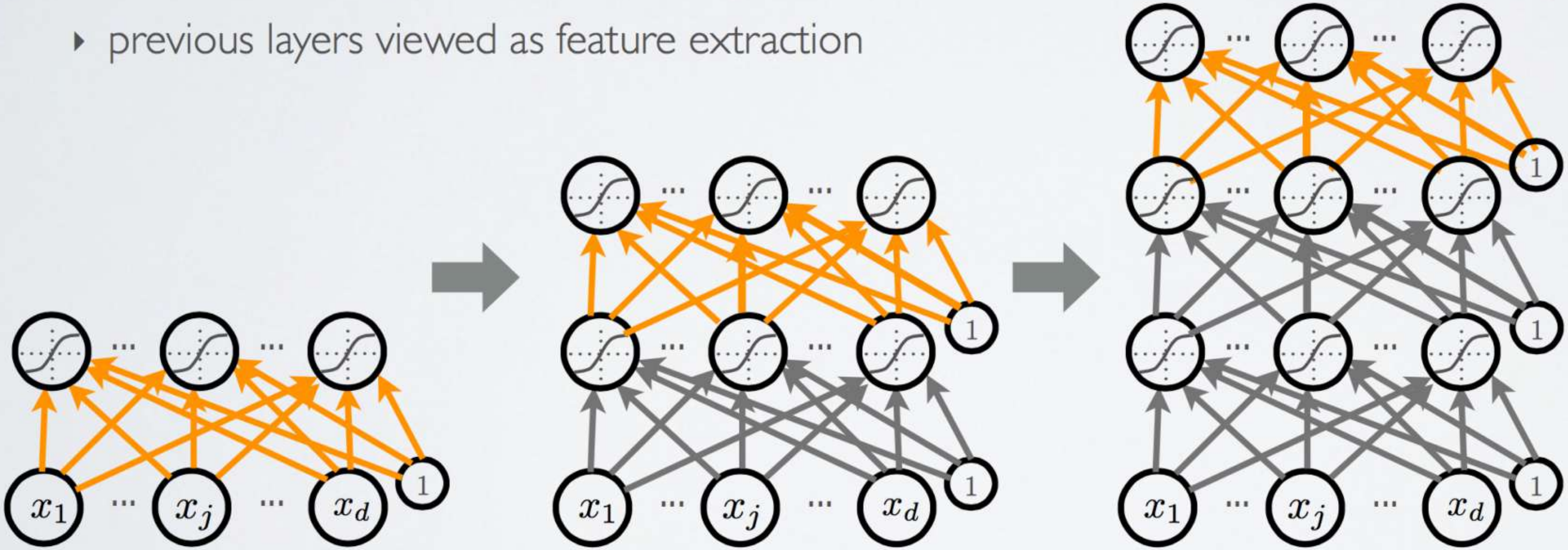
character image

random image

  - encourage hidden layers to encode that structure

Unsupervised Pre-training

- We will use a greedy, layer-wise procedure
  - train one layer at a time, from first to last, with unsupervised criterion
  - fix the parameters of previous hidden layers
  - previous layers viewed as feature extraction



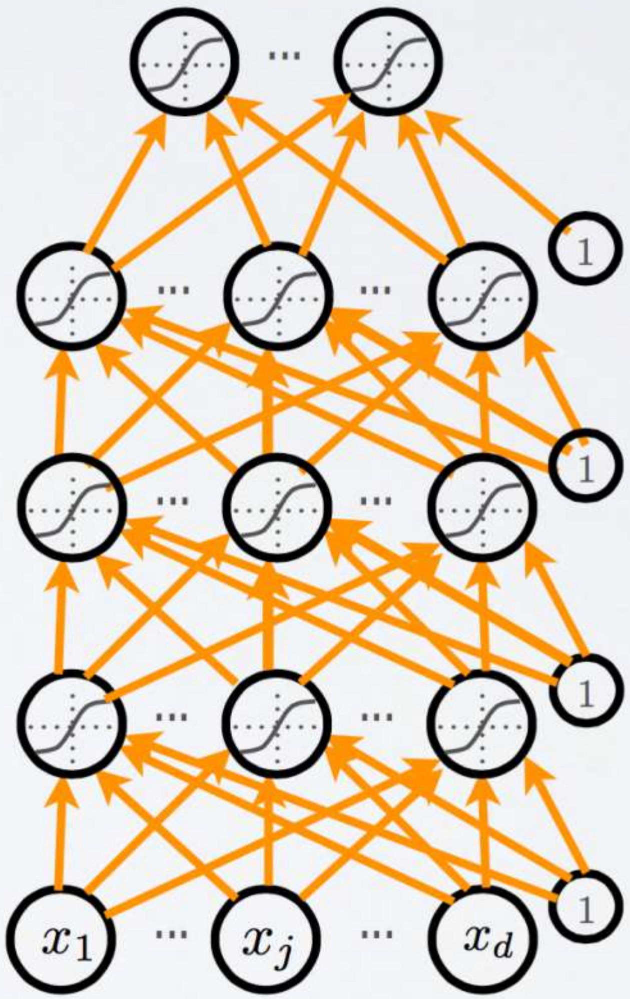
Why Training NN Is Hard?

- We call this procedure unsupervised pre-training
  - **first layer:** find hidden unit features that are more common in training inputs than in random inputs
  - **second layer:** find *combinations* of hidden unit features that are more common than random hidden unit features
  - **third layer:** find *combinations of combinations* of ...
  - etc.

- Pre-training initializes the parameters in a region such that the near local optima overfit less the data

Fine-Tuning

- Once all layers are pre-trained
  - add output layer
  - train the whole network using supervised learning
- Supervised learning is performed as in a regular feed-forward network
  - forward propagation, backpropagation and update
- We call this last phase fine-tuning
  - all parameters are "tuned" for the supervised task at hand
  - representation is adjusted to be more discriminative



Deep Learning Pseudo Code

- for l=1 to L
  - build unsupervised training set (with $\mathbf{h}^{(0)}(\mathbf{x}) = \mathbf{x}$):

$$\mathcal{D} = \left\{ \mathbf{h}^{(l-1)}(\mathbf{x}^{(t)}) \right\}_{t=1}^{T}$$

  - train "greedy module" (RBM, autoencoder) on $\mathcal{D}$
  - use hidden layer weights and biases of greedy module to initialize the deep network parameters $\mathbf{W}^{(l)}$, $\mathbf{b}^{(l)}$

pre-training

- Initialize $\mathbf{W}^{(L+1)}$, $\mathbf{b}^{(L+1)}$ randomly (as usual)
- Train the whole neural network using (supervised) stochastic gradient descent (with backprop)

fine-tuning

What Kind of Unsupervised Learning?

- Stacked restricted Boltzmann machines:
  - Hinton, Teh and Osindero suggested this procedure with RBMs
    - A fast learning algorithm for deep belief nets.
      Hinton, Teh, Osindero., 2006.
    - To recognize shapes, first learn to generate images.
      Hinton, 2006.
- Stacked autoencoders:
  - Bengio, Lamblin, Popovici and Larochelle studied and generalized the procedure to autoencoders
    - Greedy Layer-Wise Training of Deep Networks.
      Bengio, Lamblin, Popovici and Larochelle, 2007.
  - Ranzato, Poultney, Chopra and LeCun also generalized it to sparse autoencoders
    - Efficient Learning of Sparse Representations with an Energy-Based Model.
      Ranzato, Poultney, Chopra and LeCun, 2007.

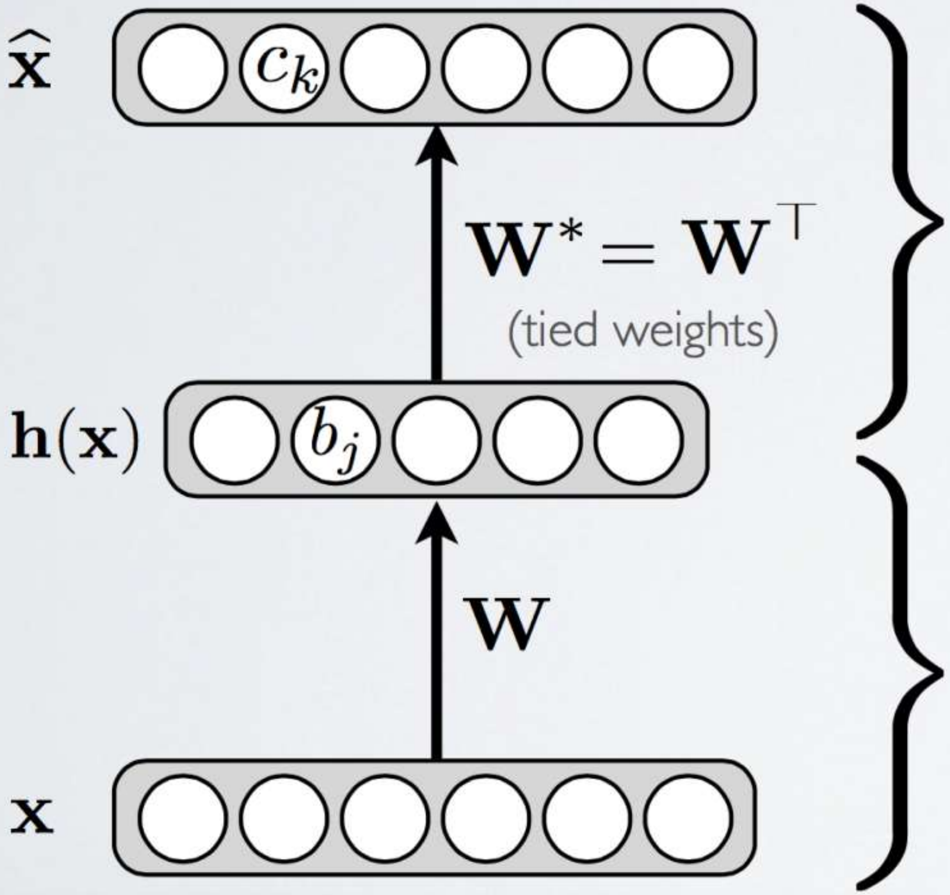
What Kind of Unsupervised Learning?

- Stacked denoising autoencoders:
  - proposed by Vincent, Larochelle, Bengio and Manzagol
    - Extracting and Composing Robust Features with Denoising Autoencoders, Vincent, Larochelle, Bengio and Manzagol, 2008.
- And more:
  - stacked semi-supervised embeddings
    - Deep Learning via Semi-Supervised Embedding. Weston, Ratle and Collobert, 2008.
  - stacked kernel PCA
    - Kernel Methods for Deep Learning. Cho and Saul, 2009.
  - stacked independent subspace analysis
    - Learning hierarchical invariant spatio-temporal features for action recognition with independent subspace analysis. Le, Zou, Yeung and Ng, 2011.

Autoencoders

Definition

- Feed-forward neural network trained to reproduce its input at the output layer



Decoder

$$\begin{aligned}\widehat{\mathbf{x}} &= o(\widehat{\mathbf{a}}(\mathbf{x})) \\ &= \underbrace{\mathrm{sigm}}_{\text{for binary inputs}}(\mathbf{c} + \mathbf{W}^*\mathbf{h}(\mathbf{x}))\end{aligned}$$

Encoder

$$\begin{aligned}\mathbf{h}(\mathbf{x}) &= g(\mathbf{a}(\mathbf{x})) \\ &= \mathrm{sigm}(\mathbf{b} + \mathbf{W}\mathbf{x})\end{aligned}$$

Loss Function

$$f(\mathbf{x}) \equiv \widehat{\mathbf{x}}$$

- For binary inputs:

$$l(f(\mathbf{x})) = -\sum_k \left(x_k \log(\widehat{x}_k) + (1 - x_k)\log(1 - \widehat{x}_k)\right)$$

  - cross-entropy (more precisely: sum of Bernoulli cross-entropies)

- For real-valued inputs:

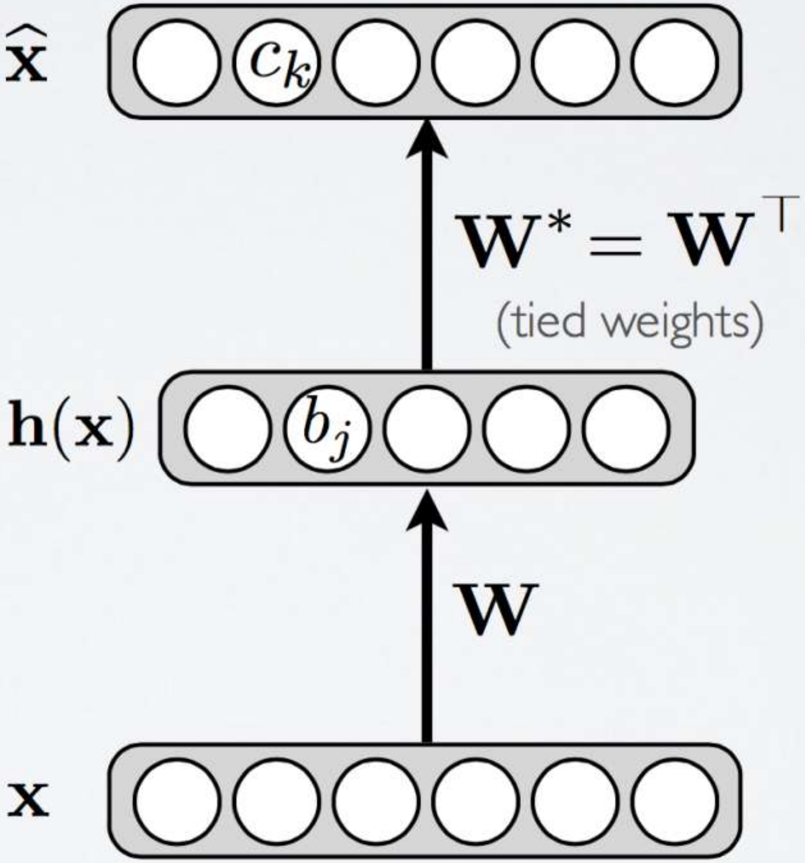
$$l(f(\mathbf{x})) = \frac{1}{2}\sum_k (\widehat{x}_k - x_k)^2$$

  - sum of squared differences (squared euclidean distance)
  - we use a linear activation function at the output

Undercomplete Autoencoder

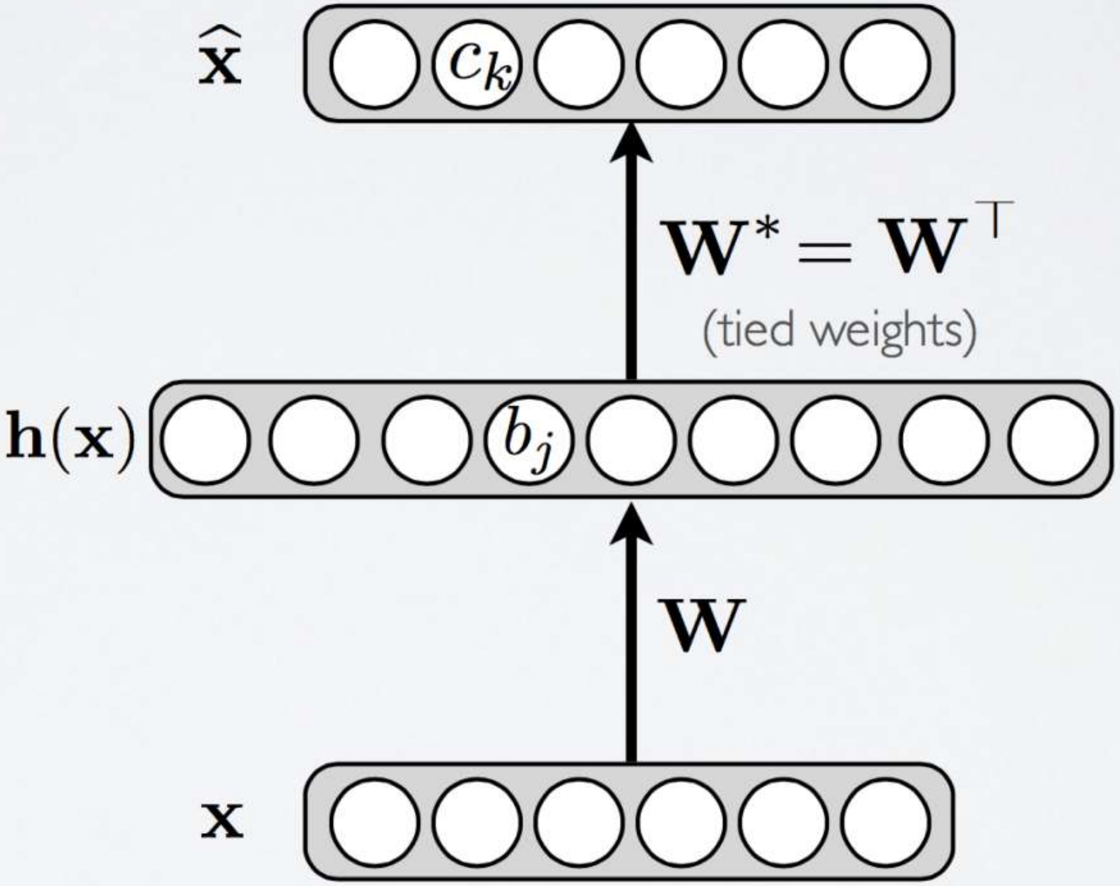
- Hidden layer is undercomplete if smaller than the input layer
  - hidden layer "compresses" the input
  - will compress well only for the training distribution
- Hidden units will be
  - good features for the training distribution

  - but bad for other types of input



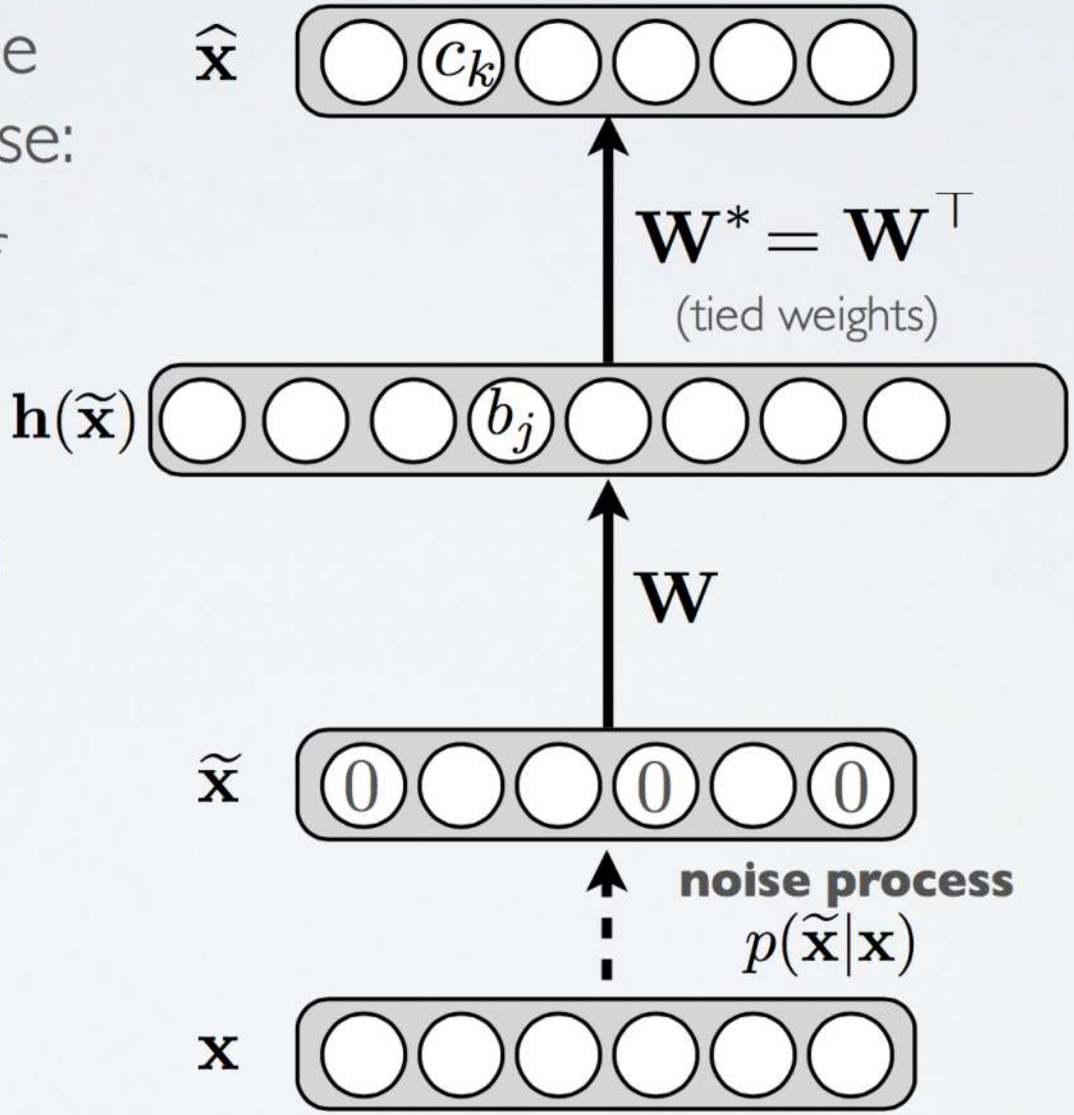
Overcomplete Autoencoder

- 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

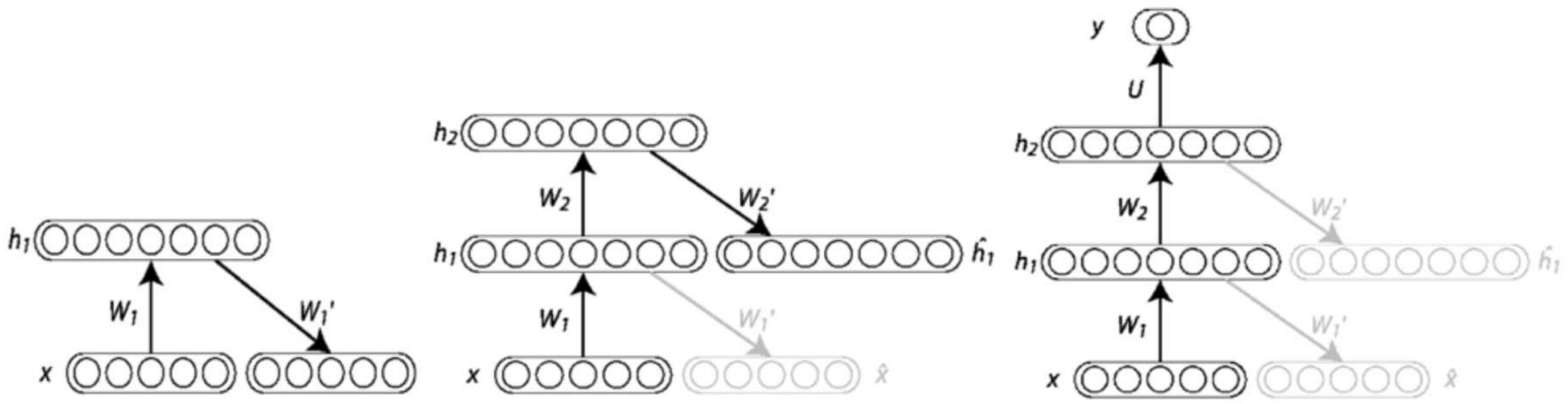


Denoising Autoencoder

- Idea: representation should be robust to introduction of noise:
  - random assignment of subset of inputs to 0, with probability ν
  - Gaussian additive noise
- Reconstruction $\widehat{\mathbf{x}}$ computed from the corrupted input $\widetilde{\mathbf{x}}$
- Loss function compares $\widehat{\mathbf{x}}$ reconstruction with the **noiseless input** $\mathbf{x}$



Deep Learning Using Stacked Autoencoder



from: Bengio ICML 2009

Deep Learning Using Stacked Autoencoder

Network Type	Depth	MNIST-small classif. test error	MNIST-rotation classif. test error
Deep net	1	**4.14** % ± 0.17	15.22 % ± 0.31
	2	**4.03** % ± 0.17	**10.63** % ± 0.27
	3	**4.24** % ± 0.18	11.98 % ± 0.28
	4	4.47 % ± 0.18	11.73 % ± 0.29
Deep net + autoencoder	1	3.87 % ± 0.17	11.43% ± 0.28
	2	**3.38** % ± 0.16	9.88 % ± 0.26
	3	**3.37** % ± 0.16	**9.22** % ± 0.25
	4	**3.39** % ± 0.16	**9.20** % ± 0.25
Deep net + RBM	1	3.17 % ± 0.15	10.47 % ± 0.27
	2	**2.74** % ± 0.14	9.54 % ± 0.26
	3	**2.71** % ± 0.14	**8.80** % ± 0.25
	4	**2.72** % ± 0.14	**8.83** % ± 0.24

An Empirical Evaluation of Deep Architectures on Problems with Many Factors of Variation
Larochelle, Erhan, Courville, Bergstra and Bengio, 2007

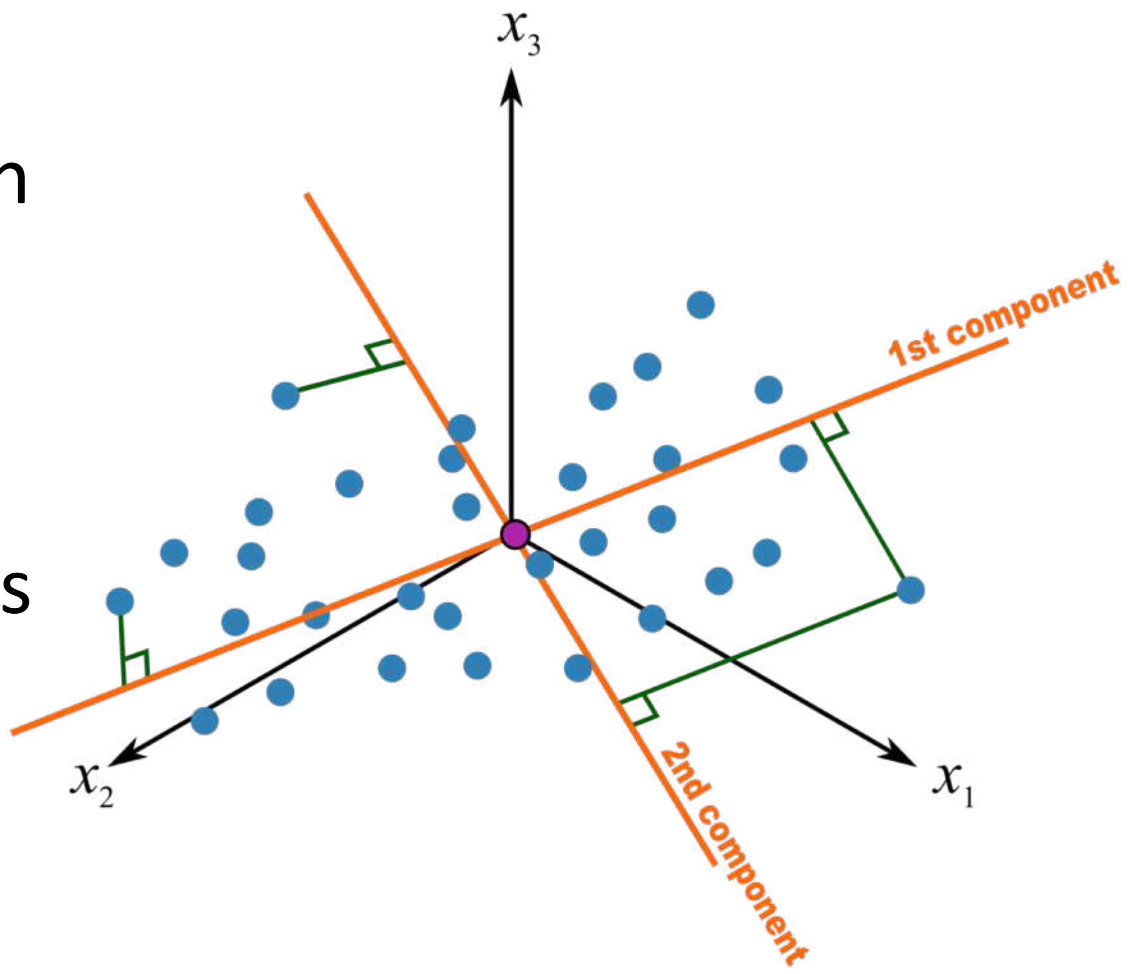
Motivations

- Pre-training for supervised learning
- **Data Encoding (Compression)**
- Denoising
- Others

Principal Component Analysis (PCA)

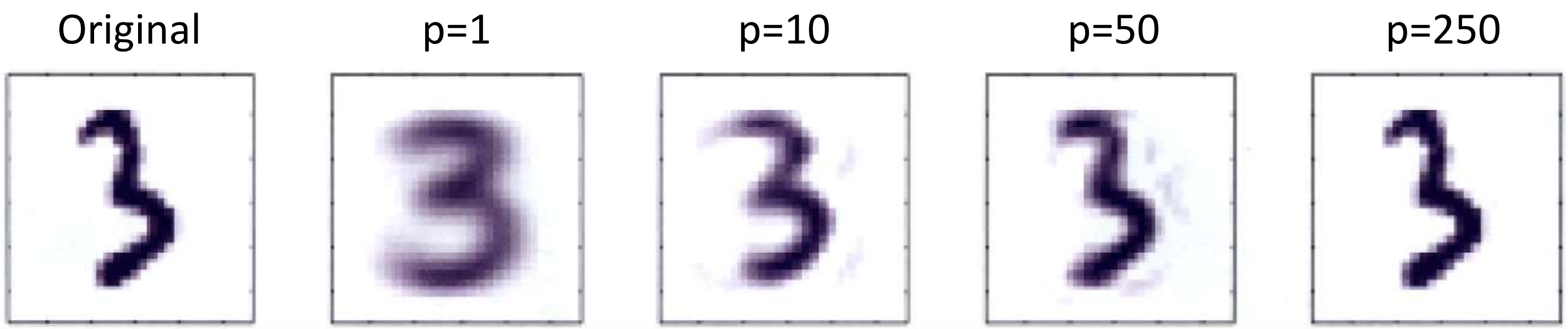
- Statistical approach for data compression and visualization
- Invented by Karl Pearson in 1901

- Weakness: linear components only.



Principal Component Analysis (PCA)

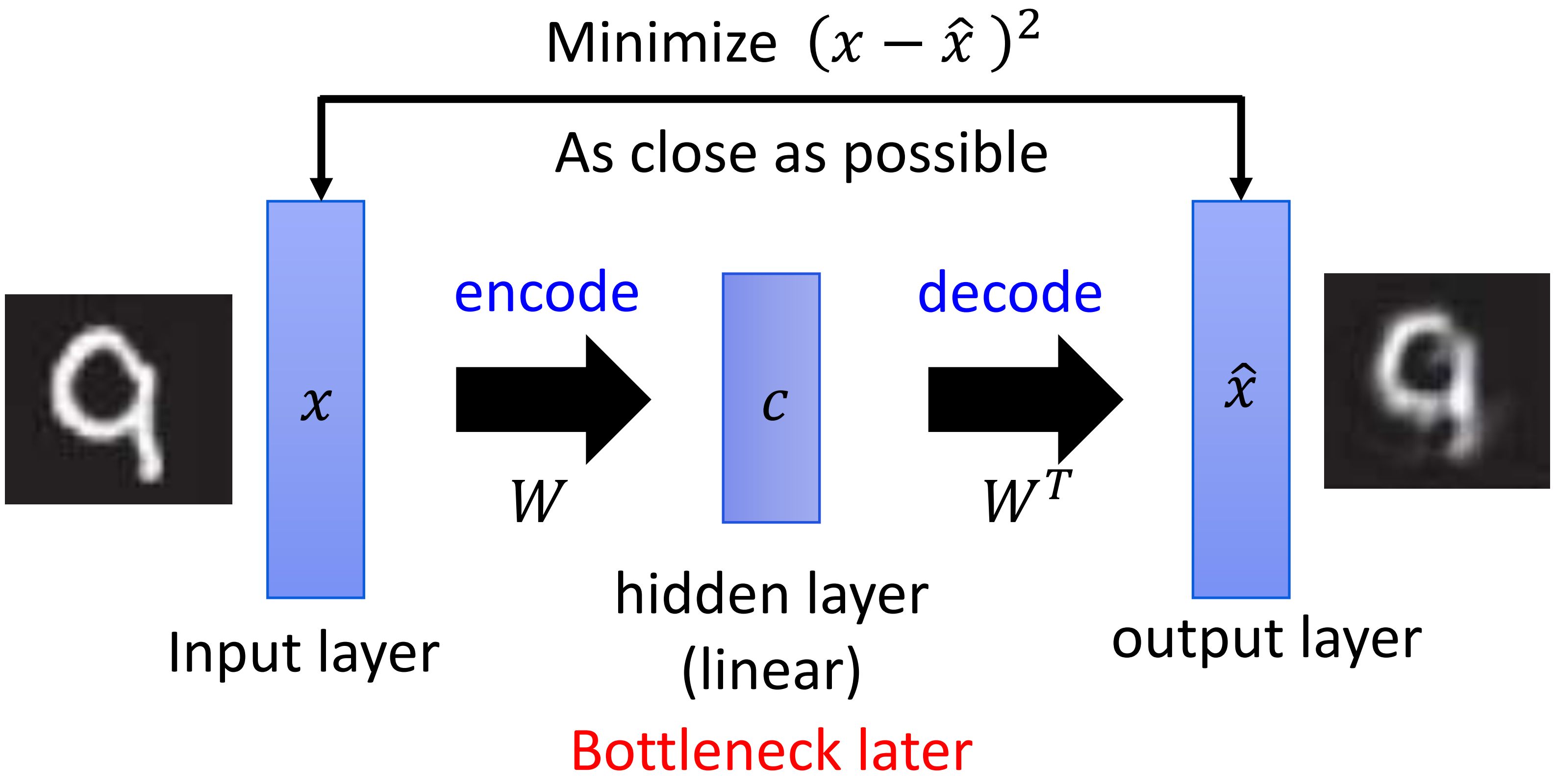
MNIST digit reconstruction using p principal components



Pattern Recognition and Machine Learning. Bishop, 2006. page 567

Autoencoder Example

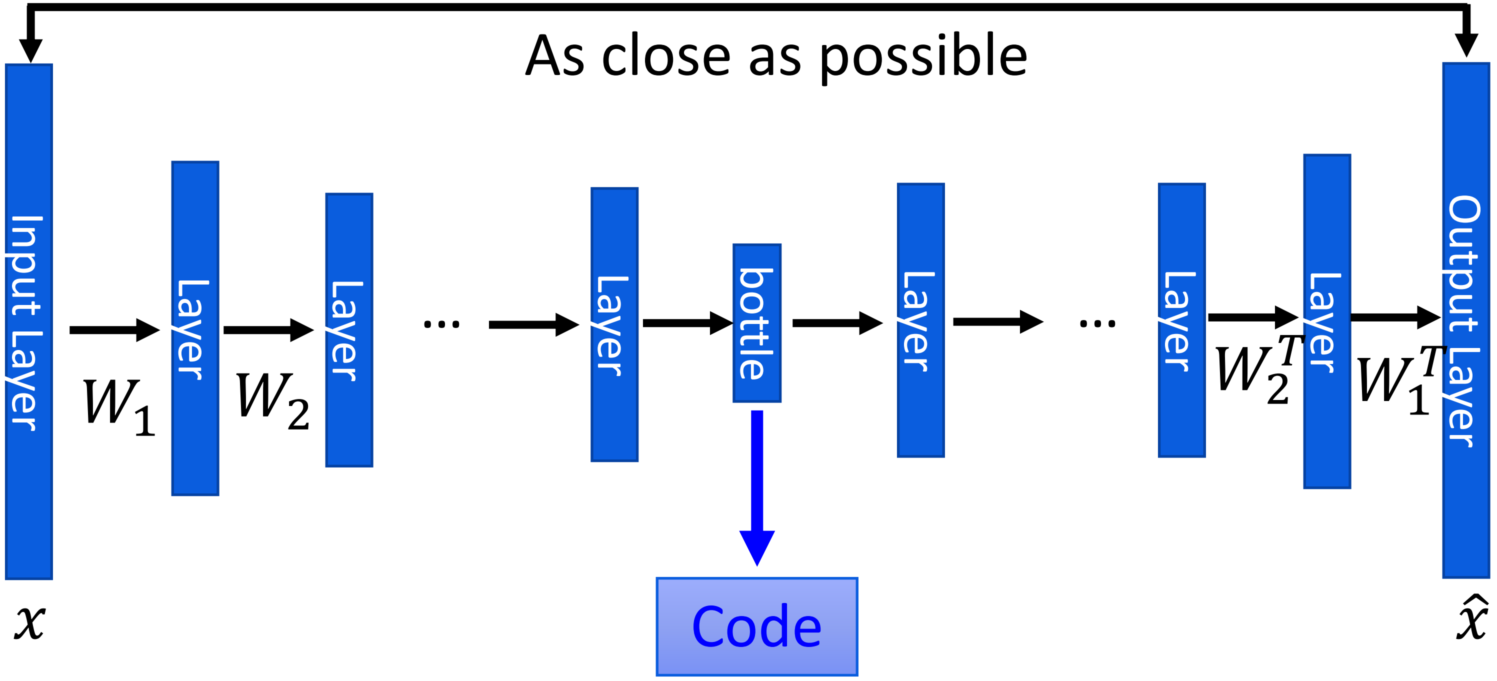
PCA



- It has been shown that an AE without non-linear activation functions achieves the **PCA** capacity.

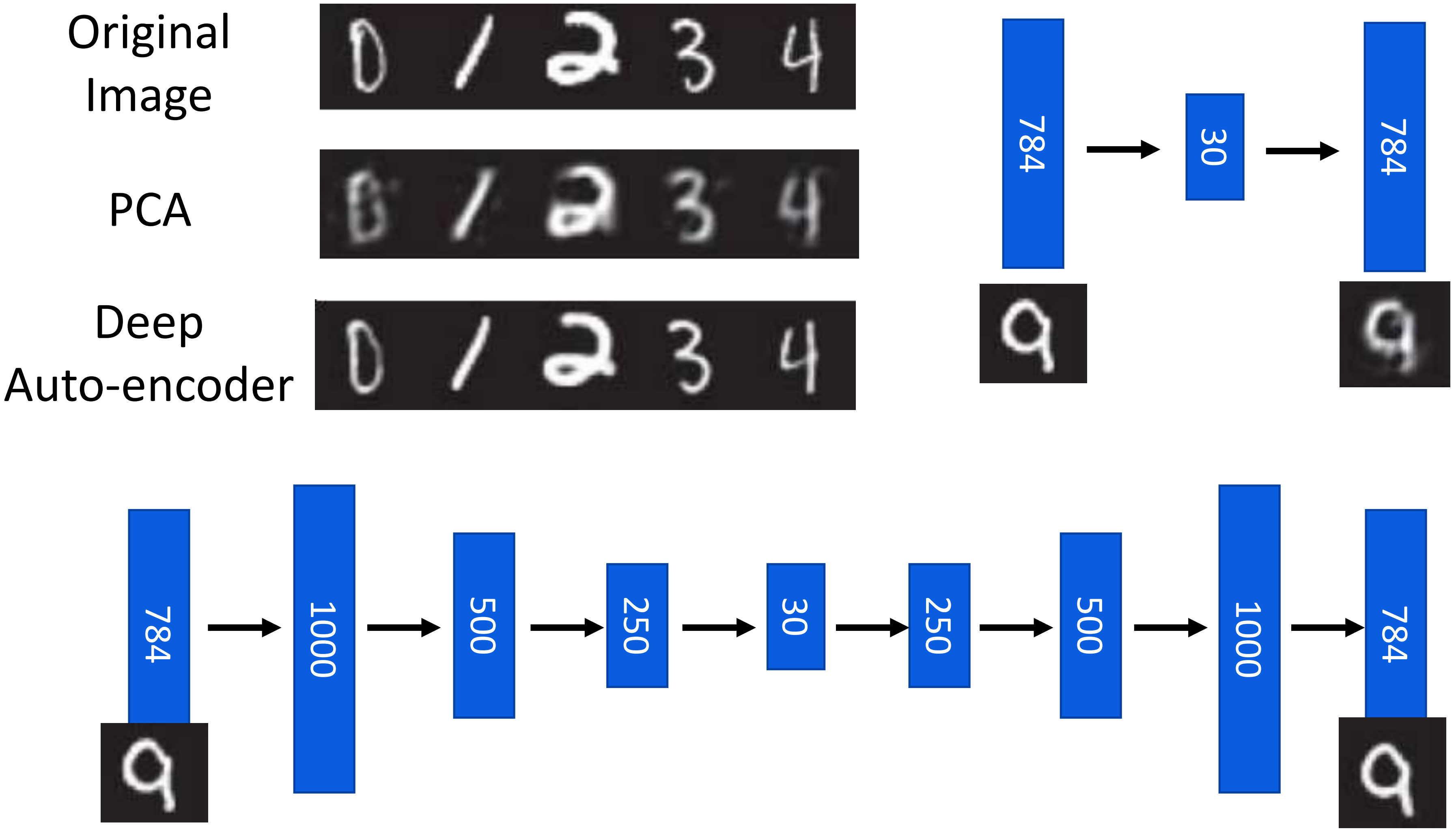
Autoencoder Example

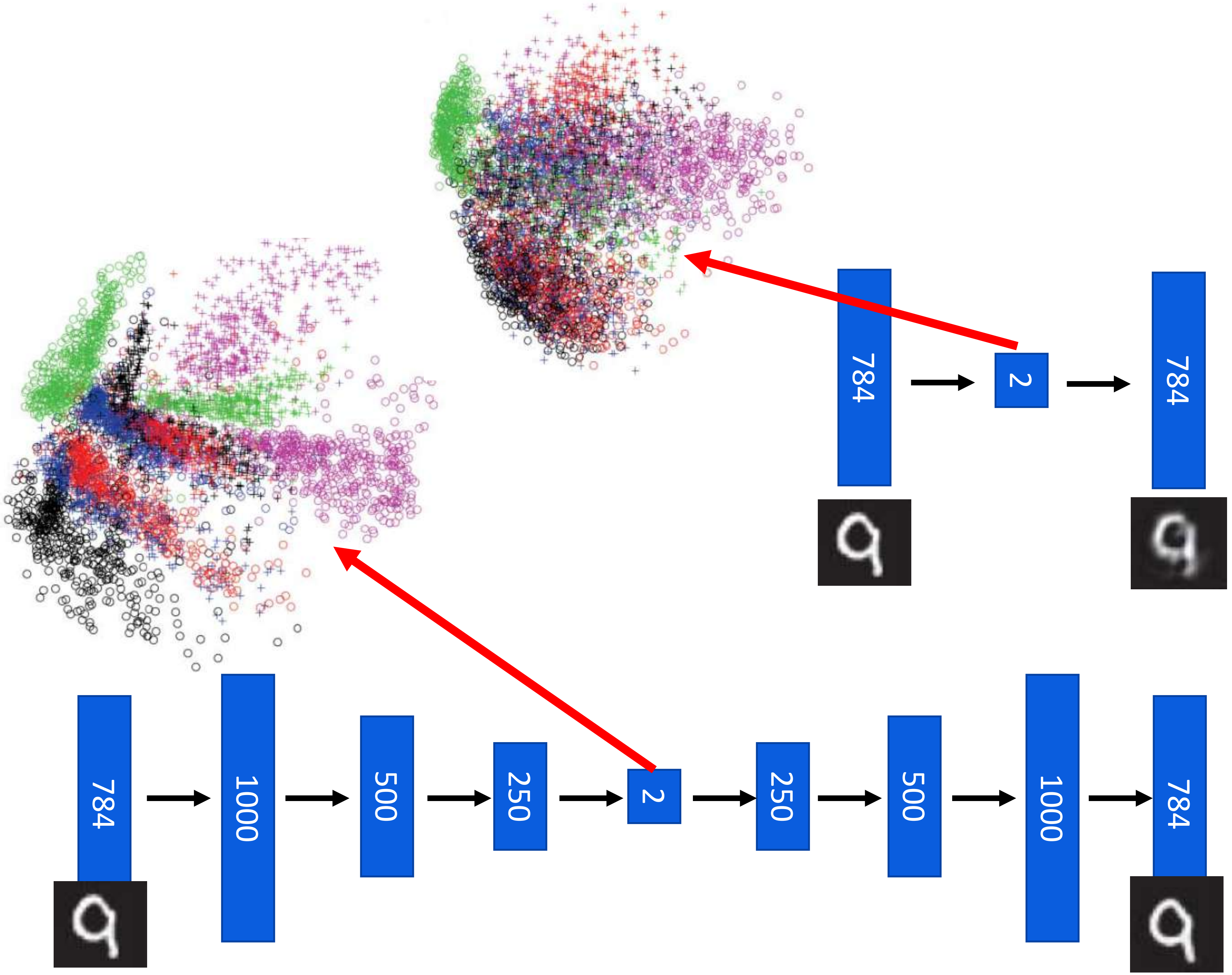
Deep Auto-encoder



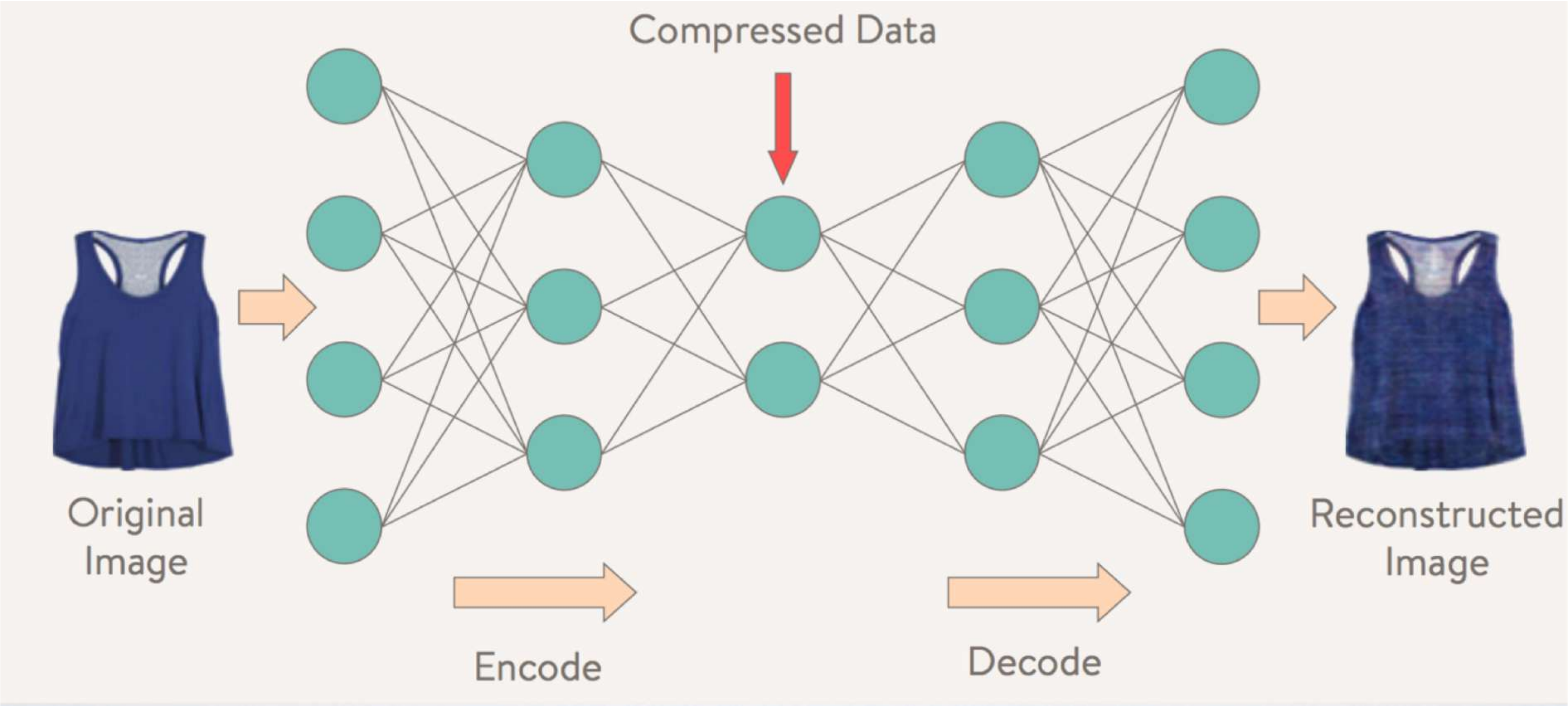
Reference: Hinton, Geoffrey E., and Ruslan R. Salakhutdinov. "Reducing the dimensionality of data with neural networks." *Science* 313.5786 (2006): 504-507

Autoencoder Example




784
2
784
784
1000
500
250
2
250
500
1000
784

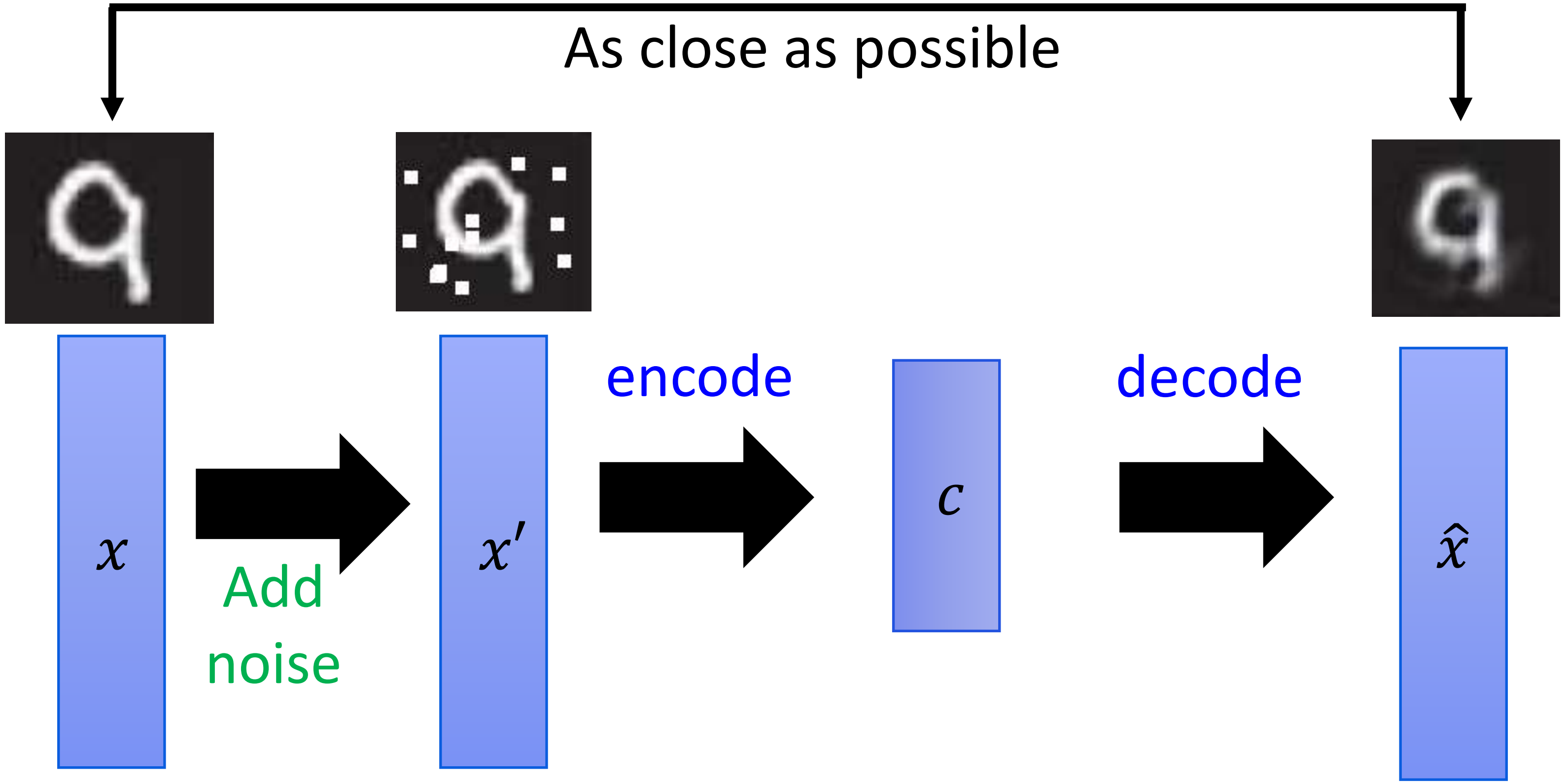
Compress Data



Motivations

- Pre-training for supervised learning
- Data Encoding (Compression)
- **Denoising**
- Others

Denoising Autoencoder



Vincent, Pascal, et al. "Extracting and composing robust features with denoising autoencoders." *ICML,* 2008.

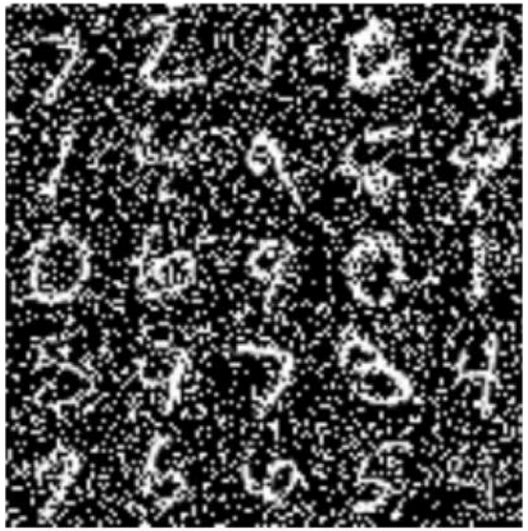
Denoising Autoencoder

Denoising Autoencoder

- Corrupt input images with noisy patterns that you would expect to see in real life
- Network learns to remove this noise

Original

Corrupted



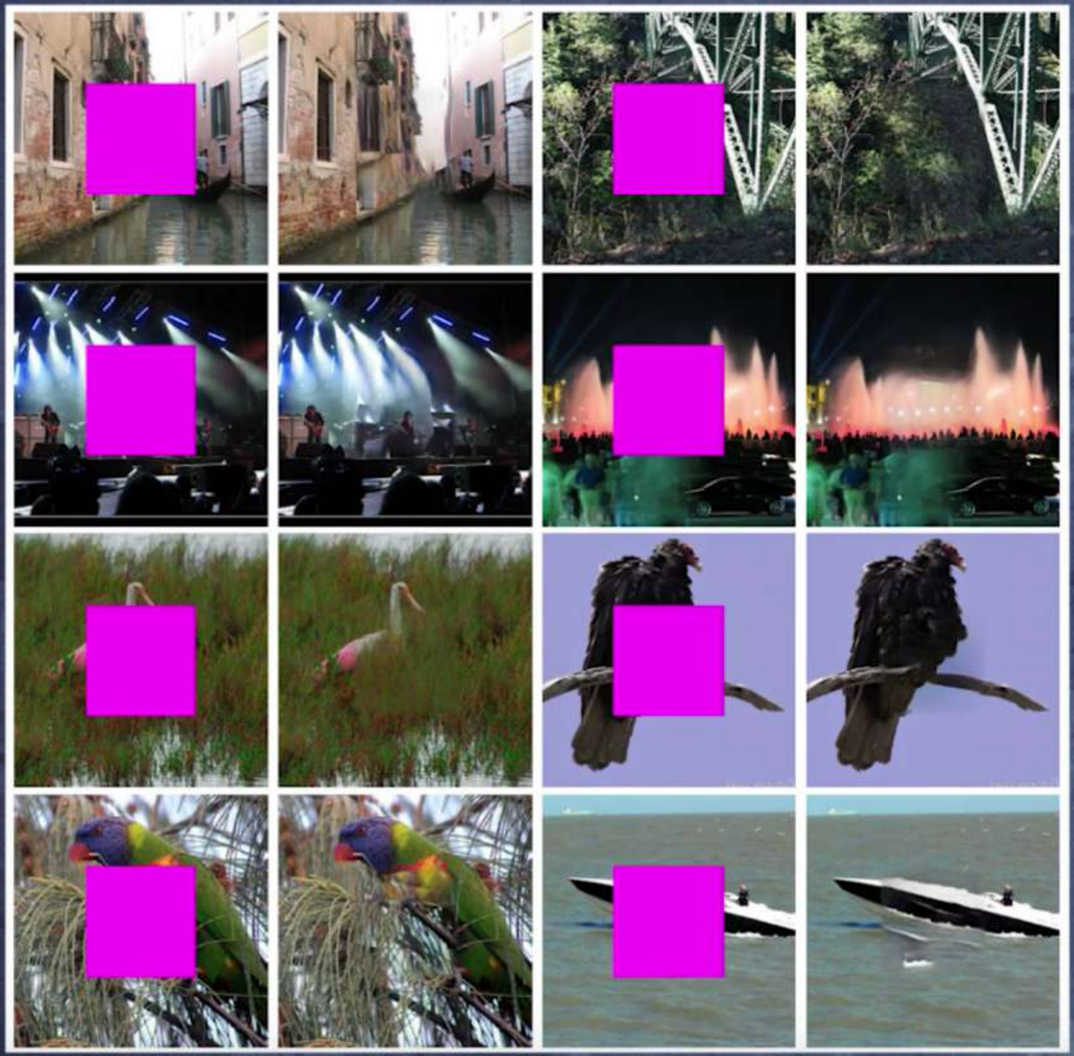
Restored

Figures: https://towardsdatascience.com/denoising-autoencoders-explained-dbb82467fc2

Motivations

- Pre-training for supervised learning
- Data Encoding (Compression)
- Denoising
- **Others**

Neural Inpainting



(source: Github -- Faster-High-Res-Neural-Inpainting)

Impute Missing Values

- **MIDA: Multiple Imputation using Denoising Autoencoders**

Lovedeep Gondara, Ke Wang (PAKDD 2018)

https://github.com/Oracen/MIDAS

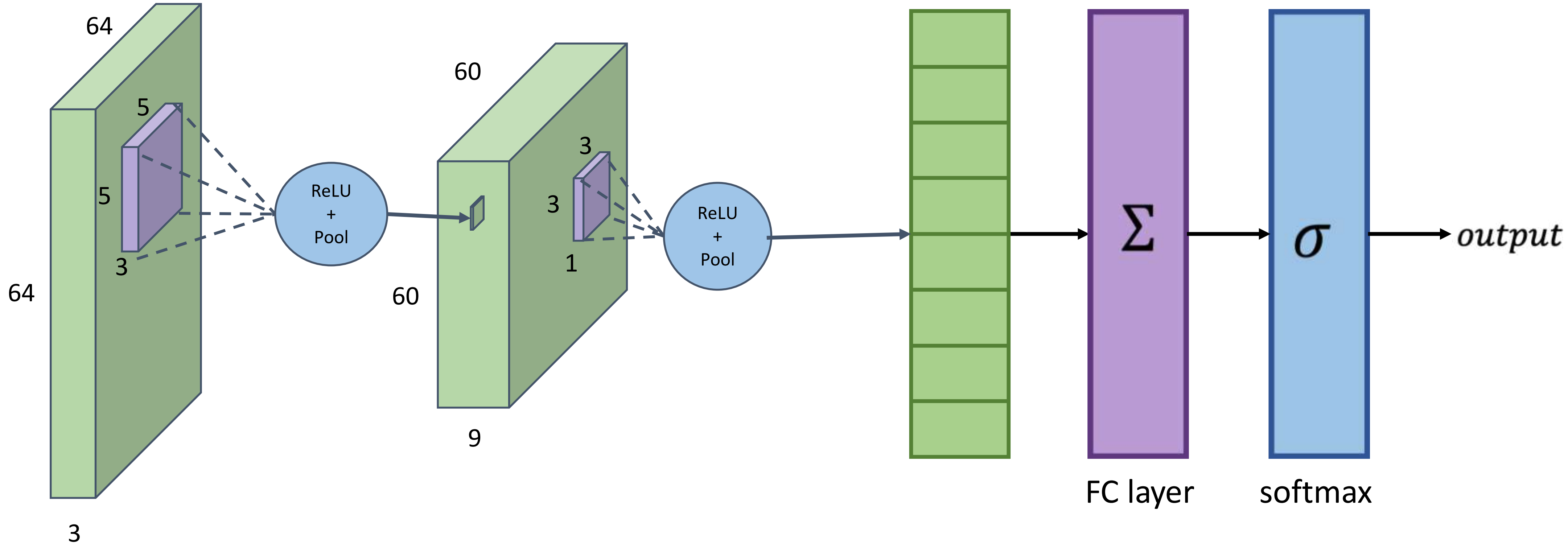
- **MISSING DATA IMPUTATION IN THE ELECTRONIC HEALTH RECORD USING DEEPLY LEARNED AUTOENCODERS***

  BRETT K. BEAULIEU-JONES, JASON H. MOORE

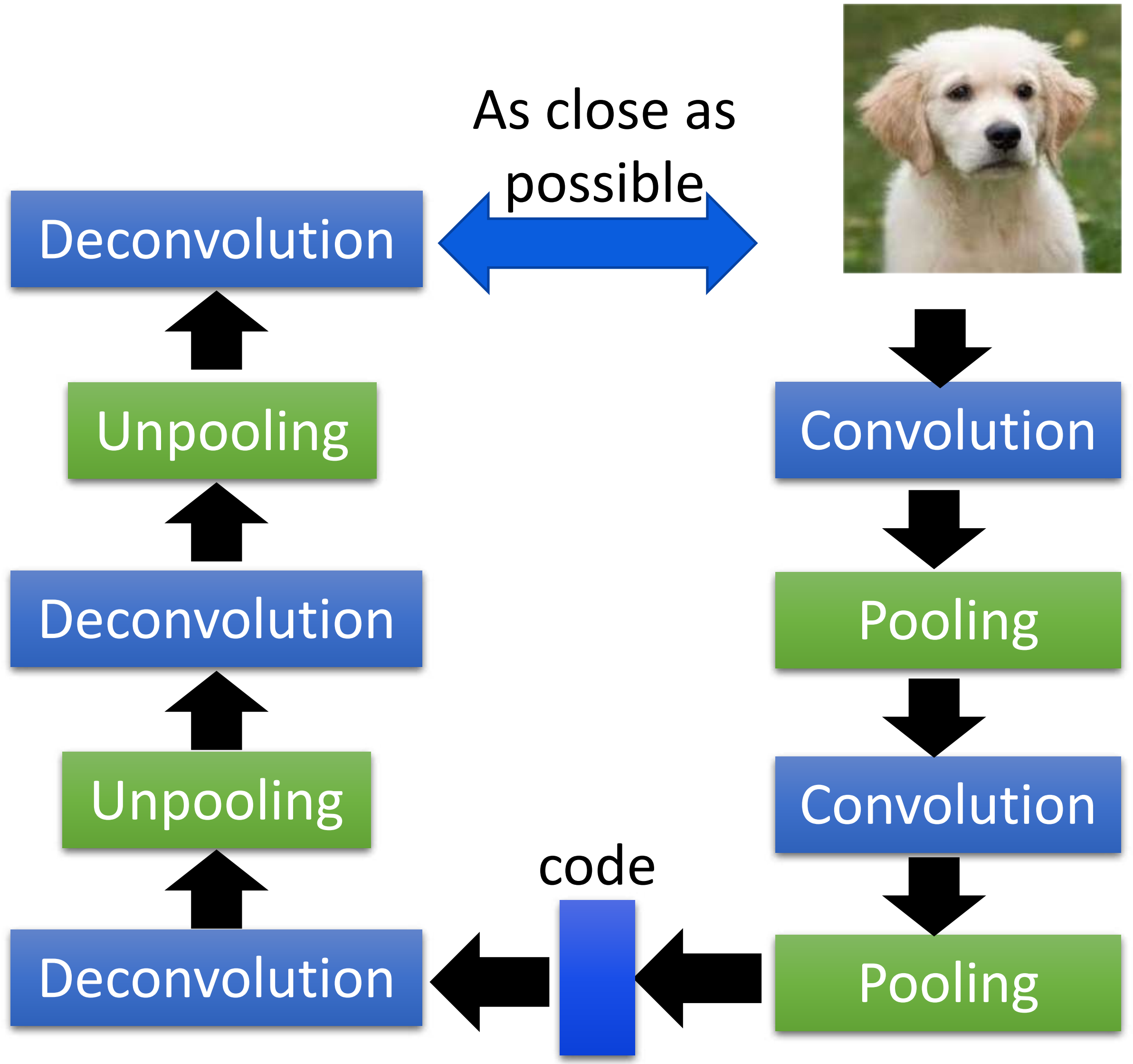
Convolutional Autoencoder

Convolutional Autoencoder

Recall CNN

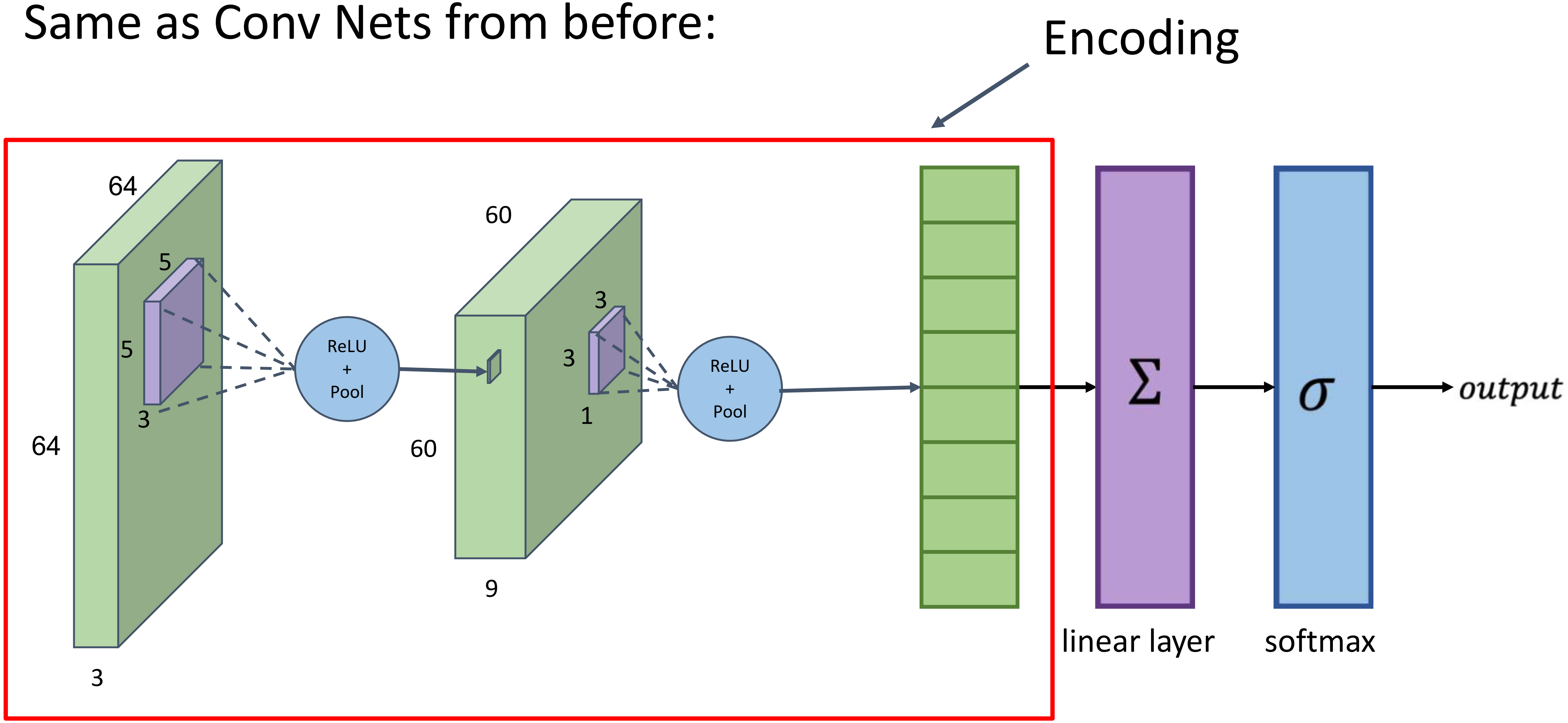


Convolutional Autoencoder



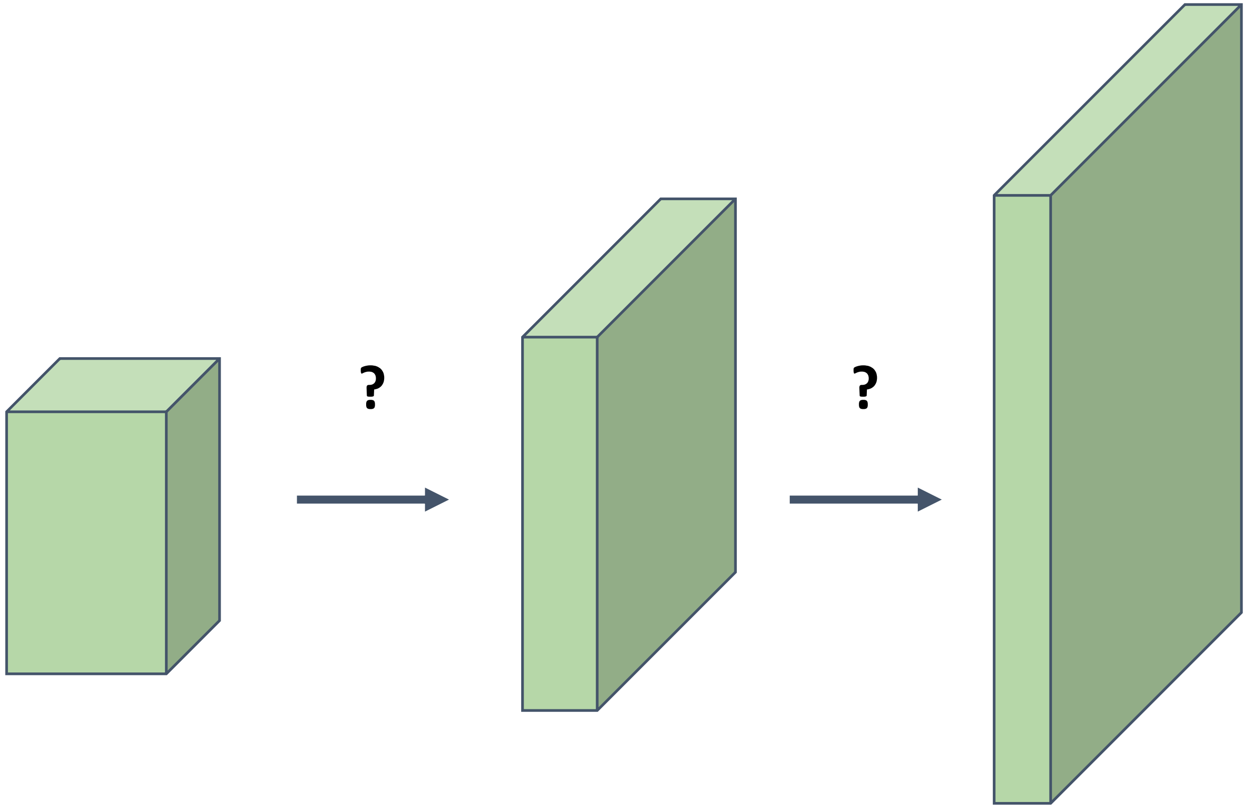
Convolutional Autoencoder

Same as Conv Nets from before:



Convolutional Autoencoder

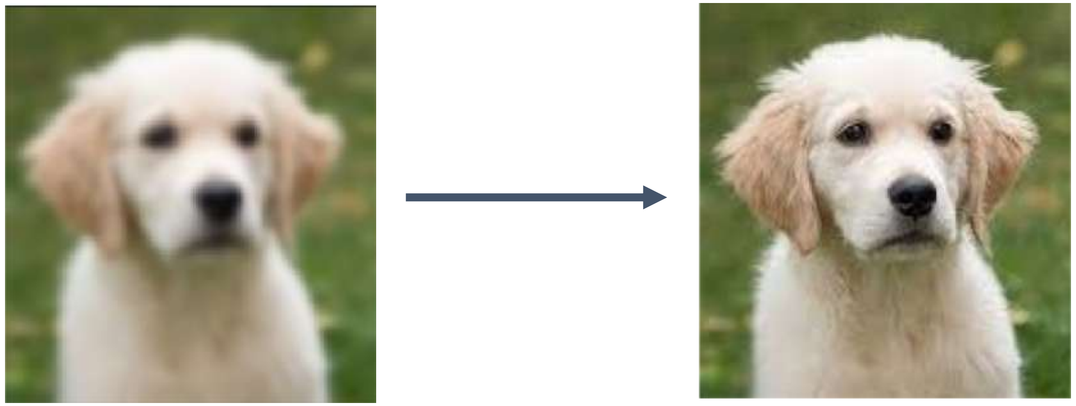
Decoding: How do we go up in size?



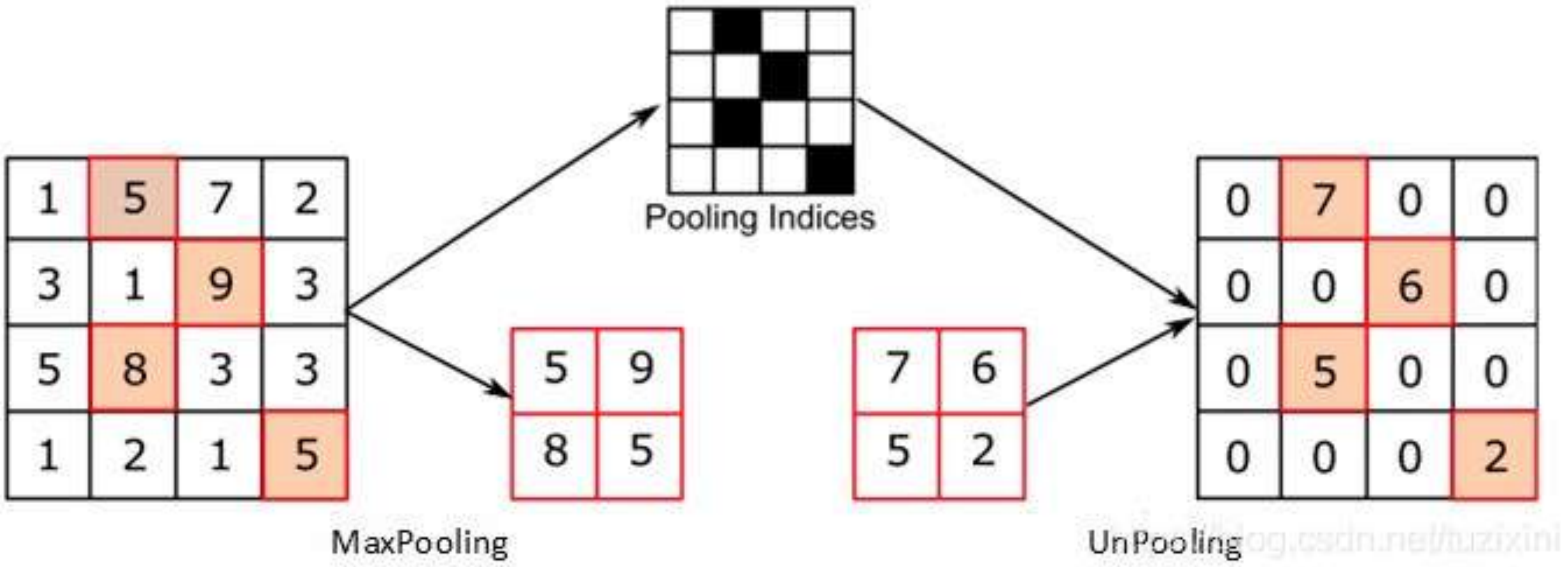
Convolutional Autoencoder: Decoding

Multiple names for the operation:

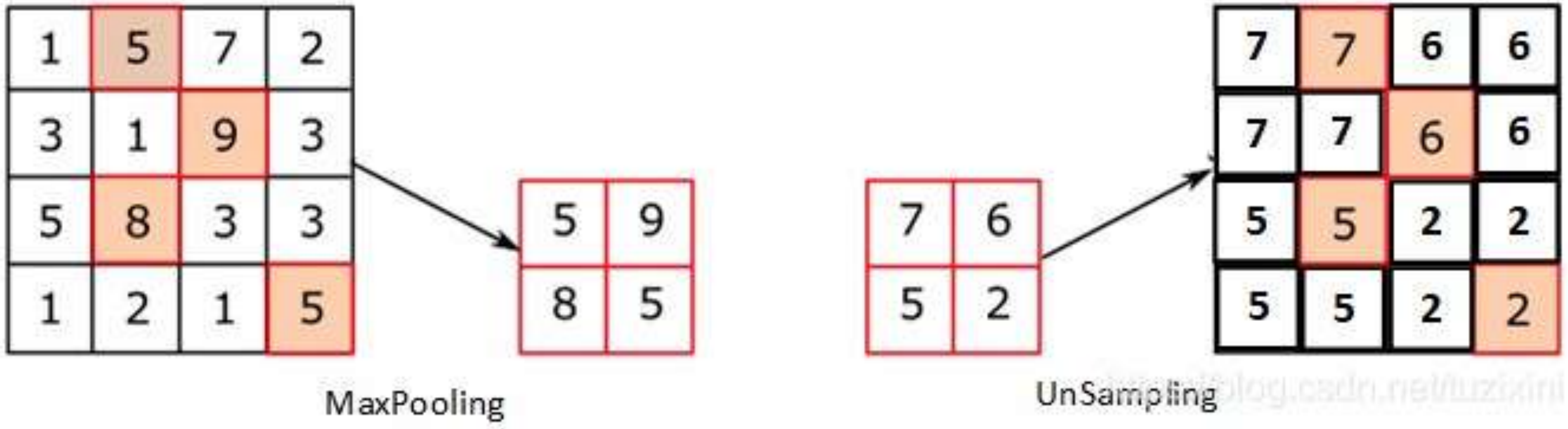
- Up-convolution (fine, but not precise)
- Fractionally-strided convolution (i.e., stride of ½)
- **Transpose convolution**



Convolutional Autoencoder: UpPooling



Convolutional Autoencoder: UpSampling



Transpose Convolution

Convolution can be viewed as a matrix multiplication

How do we represent it this way?

2	1	0	3
0	0	1	2
3	1	2	0
0	2	2	1

Input

?

1	2	3
4	5	6
7	8	9

=

57	50
66	61

Output

Transpose Convolution

Step 1: Flatten the image into a column vector

2	1	0	3
0	0	1	2
3	1	2	0
0	2	2	1

2
1
0
3
0
0
1
2
3
1
2
0
0
2
2
1

Transpose Convolution

Step 2: Flatten out the kernel/filter

1	2	3
4	5	6
7	8	9

1	2	3	0	4	5	6	0	7	8	9	0	0	0	0	0
0	1	2	3	0	4	5	6	0	7	8	9	0	0	0	0
0	0	0	0	1	2	3	0	4	5	6	0	7	8	9	0
0	0	0	0	0	1	2	3	0	4	5	6	0	7	8	9

Transpose Convolution

Step 3: Matrix multiply flattened kernel with flattened image

$$
\begin{bmatrix}
1 & 2 & 3 & 0 & 4 & 5 & 6 & 0 & 7 & 8 & 9 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 2 & 3 & 0 & 4 & 5 & 6 & 0 & 7 & 8 & 9 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 2 & 3 & 0 & 4 & 5 & 6 & 0 & 7 & 8 & 9 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 2 & 3 & 0 & 4 & 5 & 6 & 0 & 7 & 8 & 9
\end{bmatrix}
\times
\begin{bmatrix} 2 \\ 1 \\ 0 \\ 3 \\ 0 \\ 0 \\ 1 \\ 2 \\ 3 \\ 1 \\ 2 \\ 0 \\ 0 \\ 2 \\ 2 \\ 1 \end{bmatrix}
=
\begin{bmatrix} 57 \\ 50 \\ 66 \\ 61 \end{bmatrix}
$$

Transpose Convolution

Step 4: Finally reshape the output back into a grid

57
50
66
61

Final Output Image

57	50
66	61

Transpose Convolution

To upsample an image, we just do the inverse of this operation.

What matrix do we use?

The **transpose** of the big convolution matrix

Input image flattened to column vector

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 2 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 4 & 0 & 1 & 0 \\ 5 & 4 & 2 & 1 \\ 6 & 5 & 3 & 2 \\ 0 & 6 & 0 & 3 \\ 7 & 0 & 4 & 0 \\ 8 & 7 & 5 & 4 \\ 9 & 8 & 6 & 5 \\ 0 & 9 & 0 & 6 \\ 0 & 0 & 7 & 0 \\ 0 & 0 & 8 & 7 \\ 0 & 0 & 9 & 8 \\ 0 & 0 & 0 & 9 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 0 \\ 6 \\ 10 \\ 14 \\ 3 \\ 15 \\ 22 \\ 26 \\ 6 \\ 14 \\ 23 \\ 26 \\ 9 \end{bmatrix}$$

Transpose Convolution

Reshape the output vector into a grid to get the final output image:

1
2
3
0
6
10
14
3
15
22
26
6
14
23
26
9

→

Final output image

1	2	3	4
6	10	14	3
15	1	22	26
14	23	26	9

Transpose Convolution in Tensorflow

```
tf.nn.conv2d_transpose(
    input,
    filters,
    output_shape,
    strides,
    padding='SAME',
    data_format='NHWC',
    dilations=None,
    name=None
)
```

An optional string from: "NHWC" or "NCHW".
Defaults to "NHWC".
"NHWC": batch_shape + [height, width, channels].
"NCHW": batch_shape + [channels, height, width].

Documentation here:
https://www.tensorflow.org/api_docs/python/tf/nn/conv2d_transpose