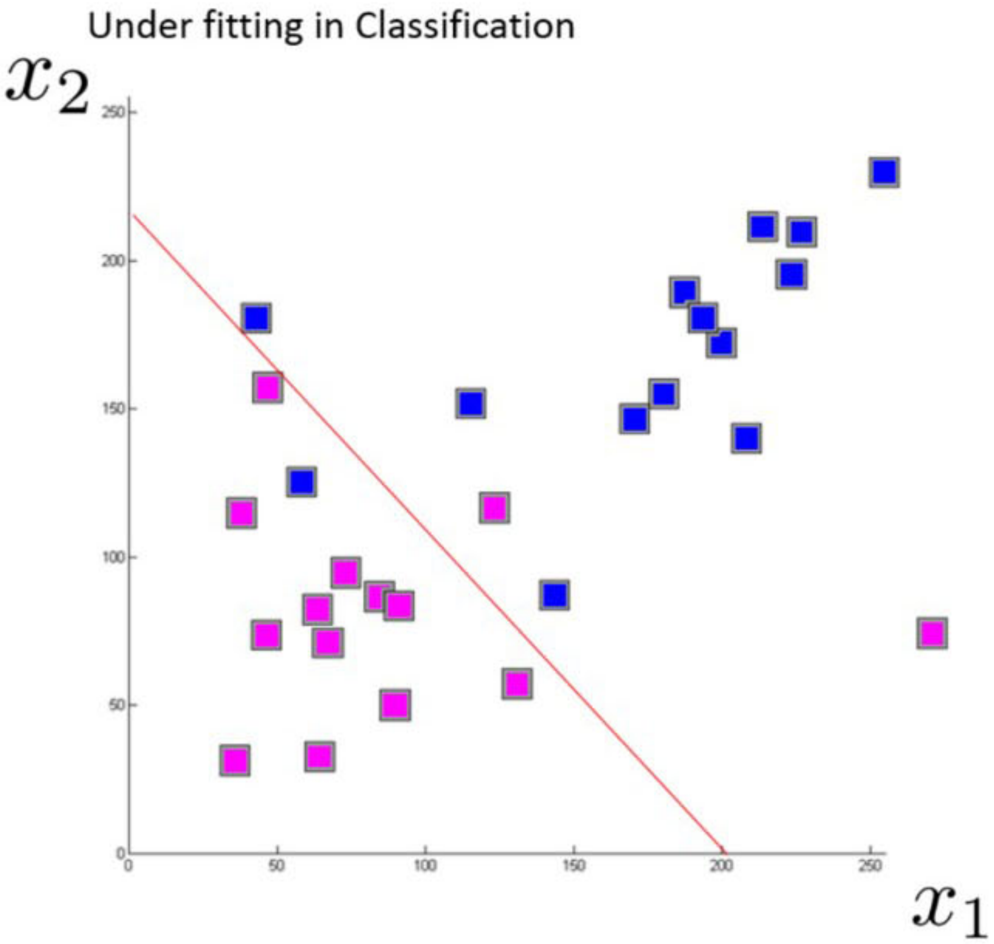
CS 466/566
Introduction to Deep Learning

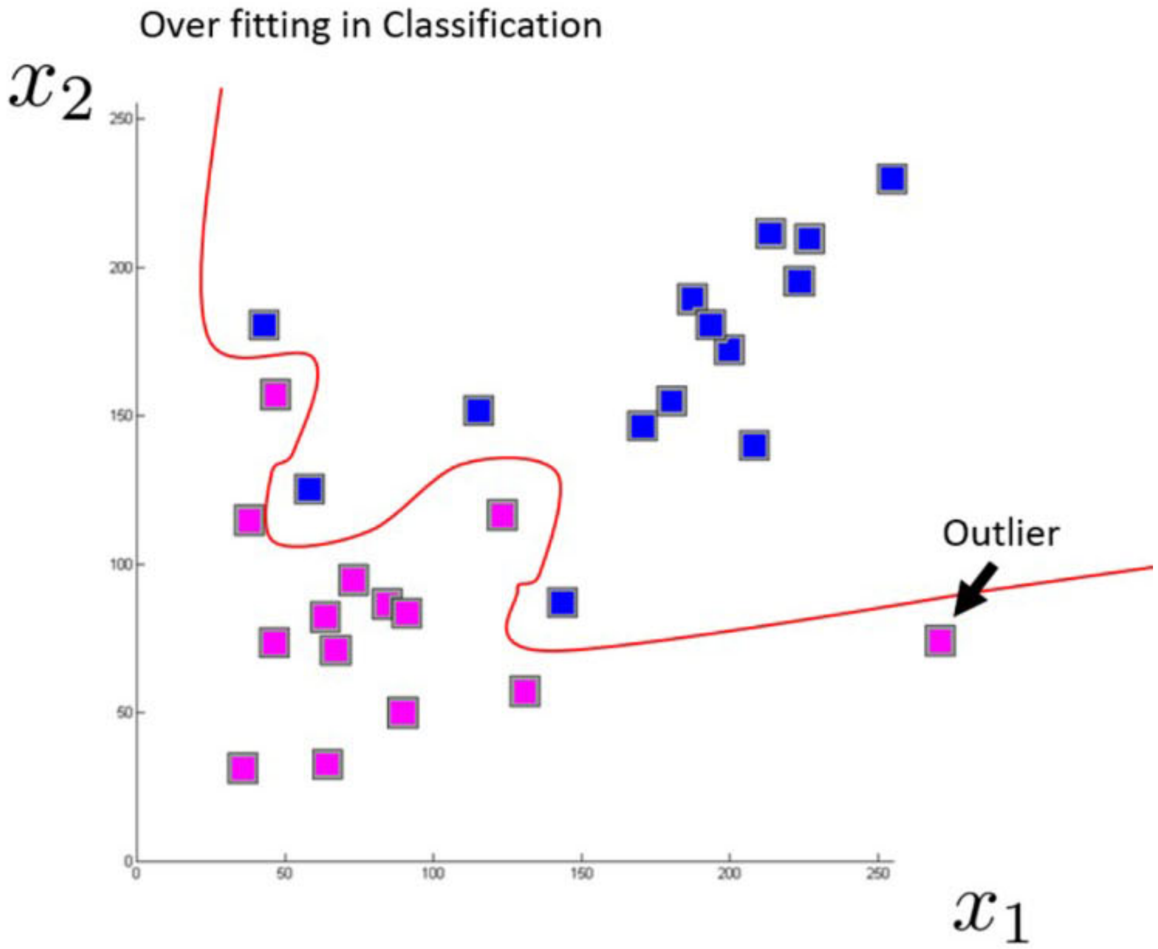
Lecture 3

Introduction to Deep Neural Networks - Part 1

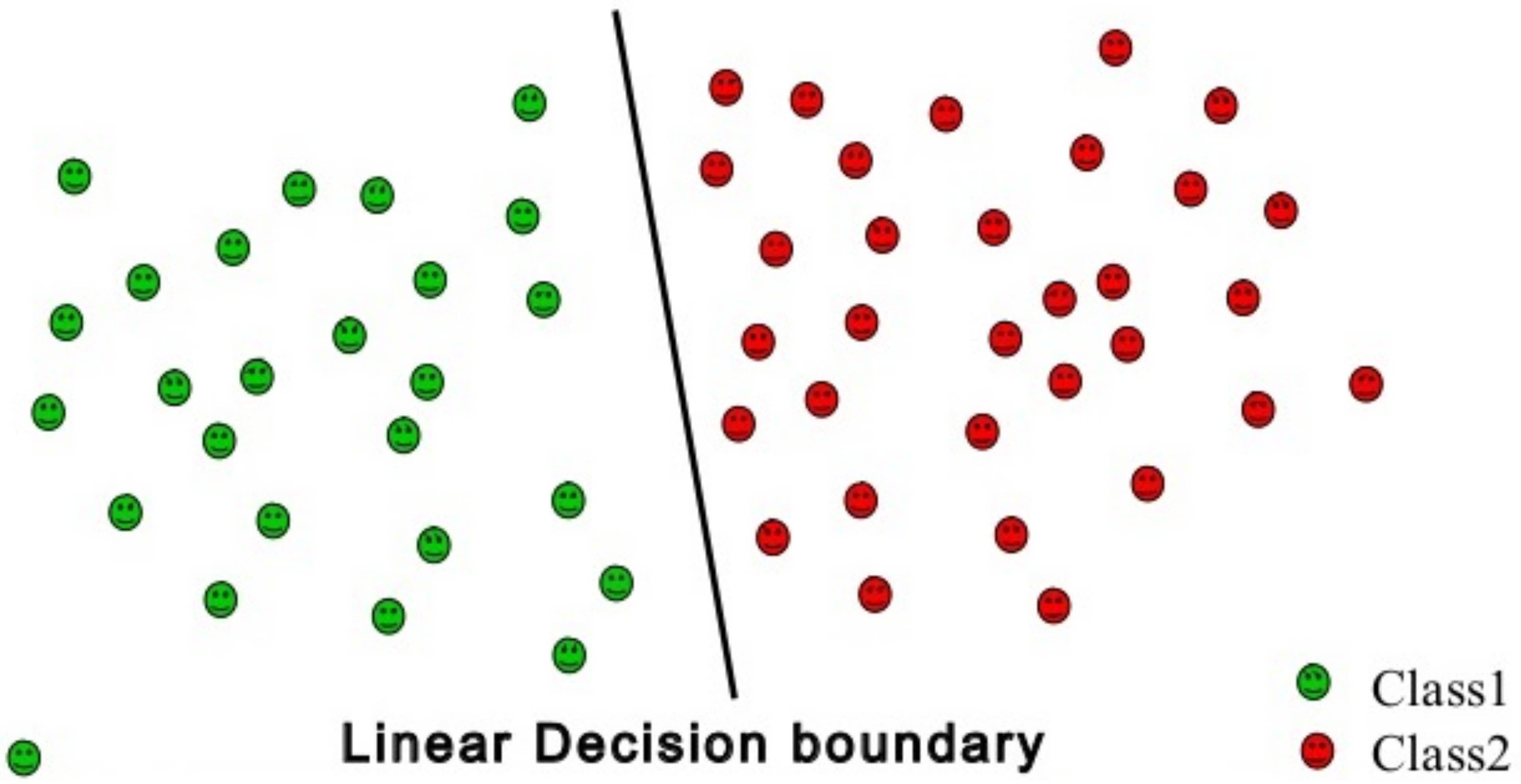
Compiled from various resources

Under-fitting and Over-fitting

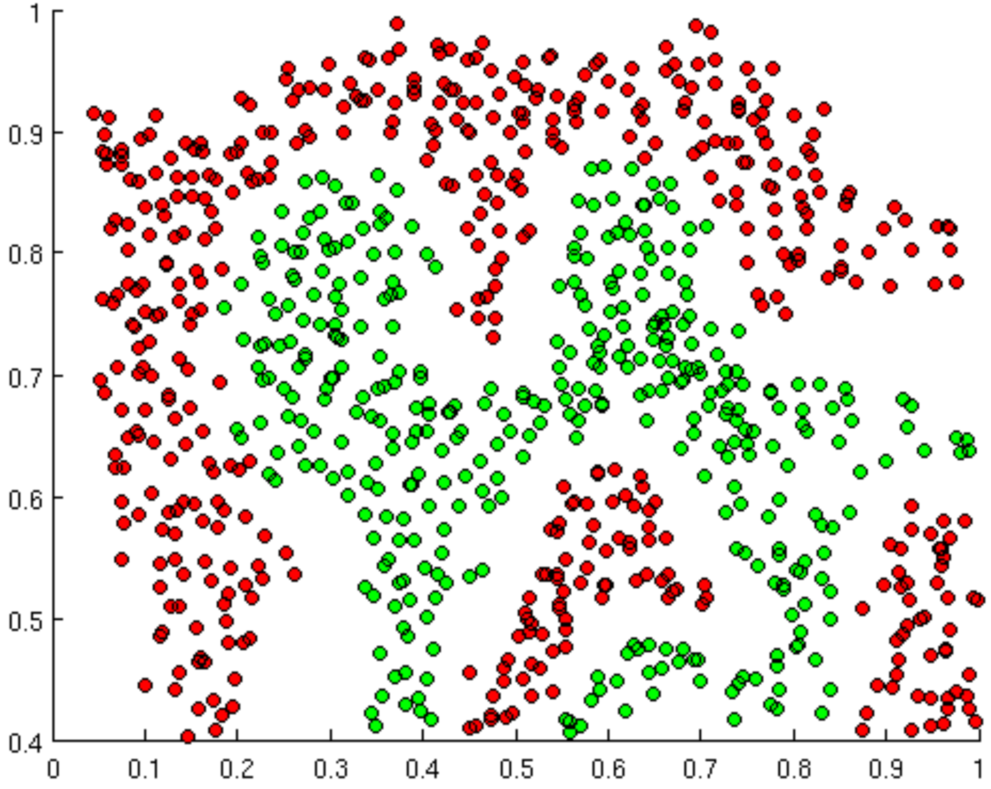


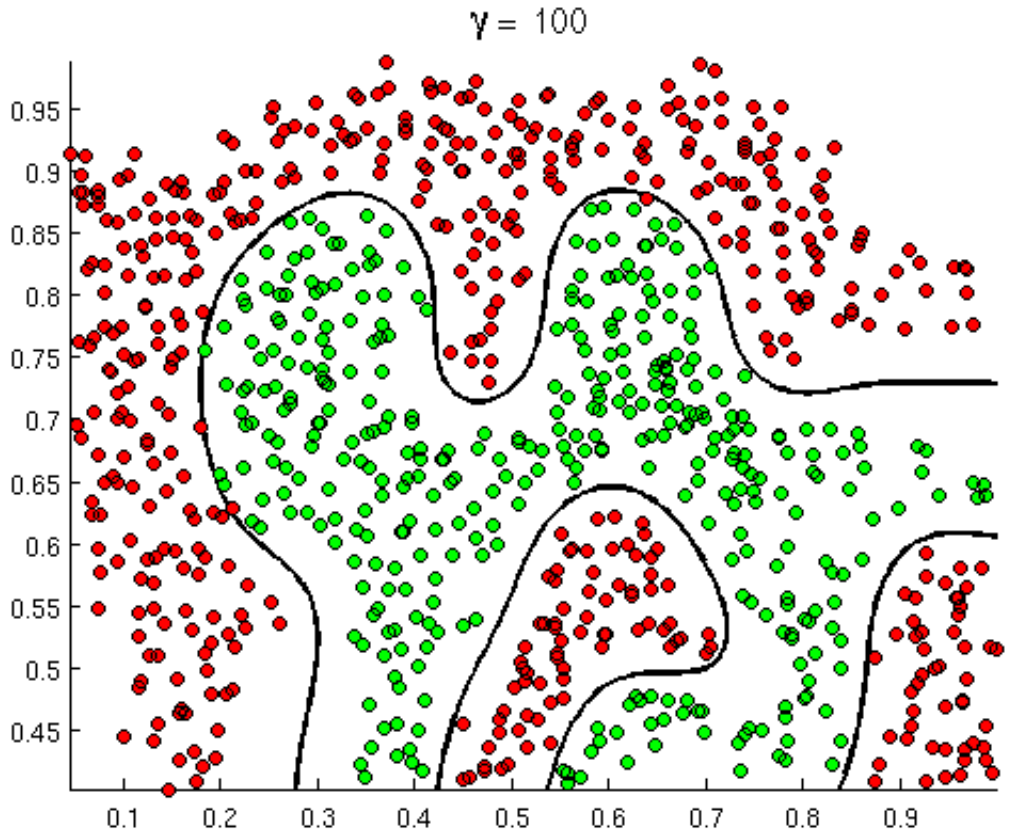


Linearly Separable Data



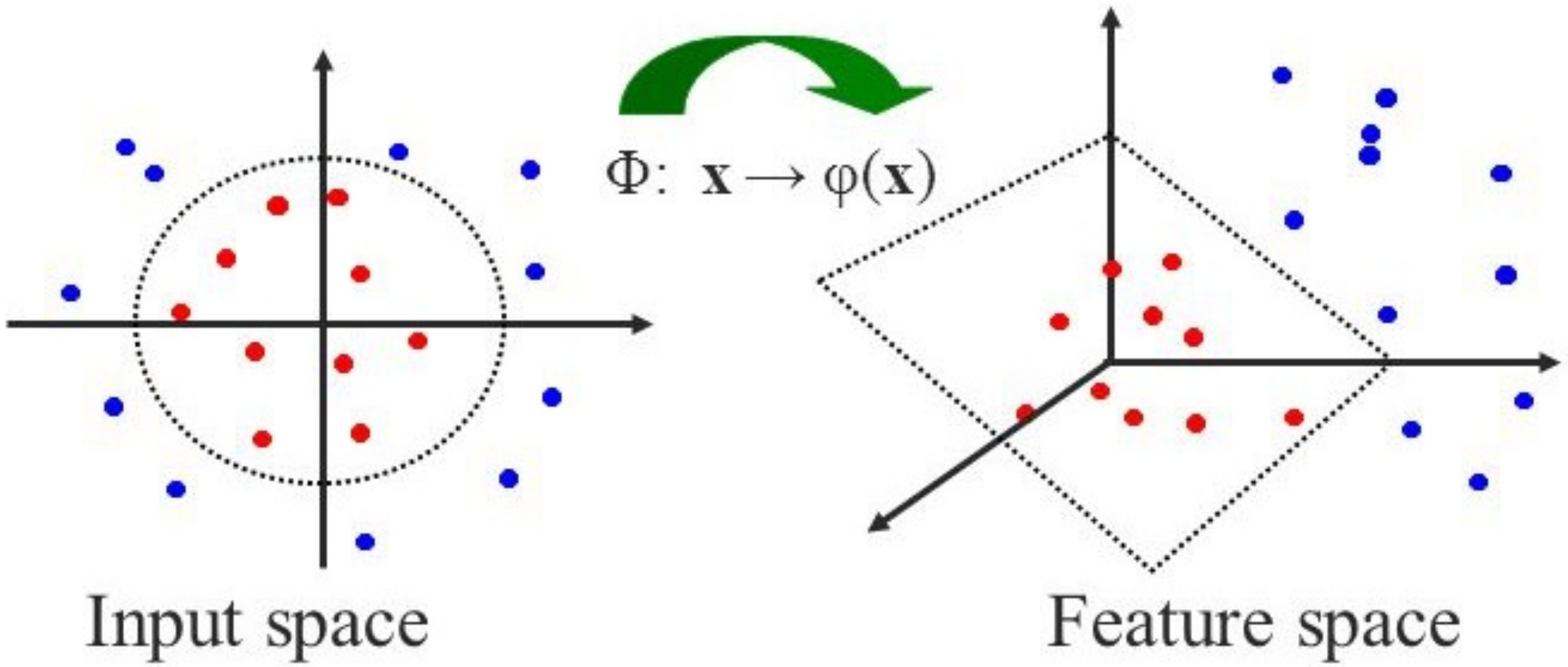
Non-linear Decision Boundaries





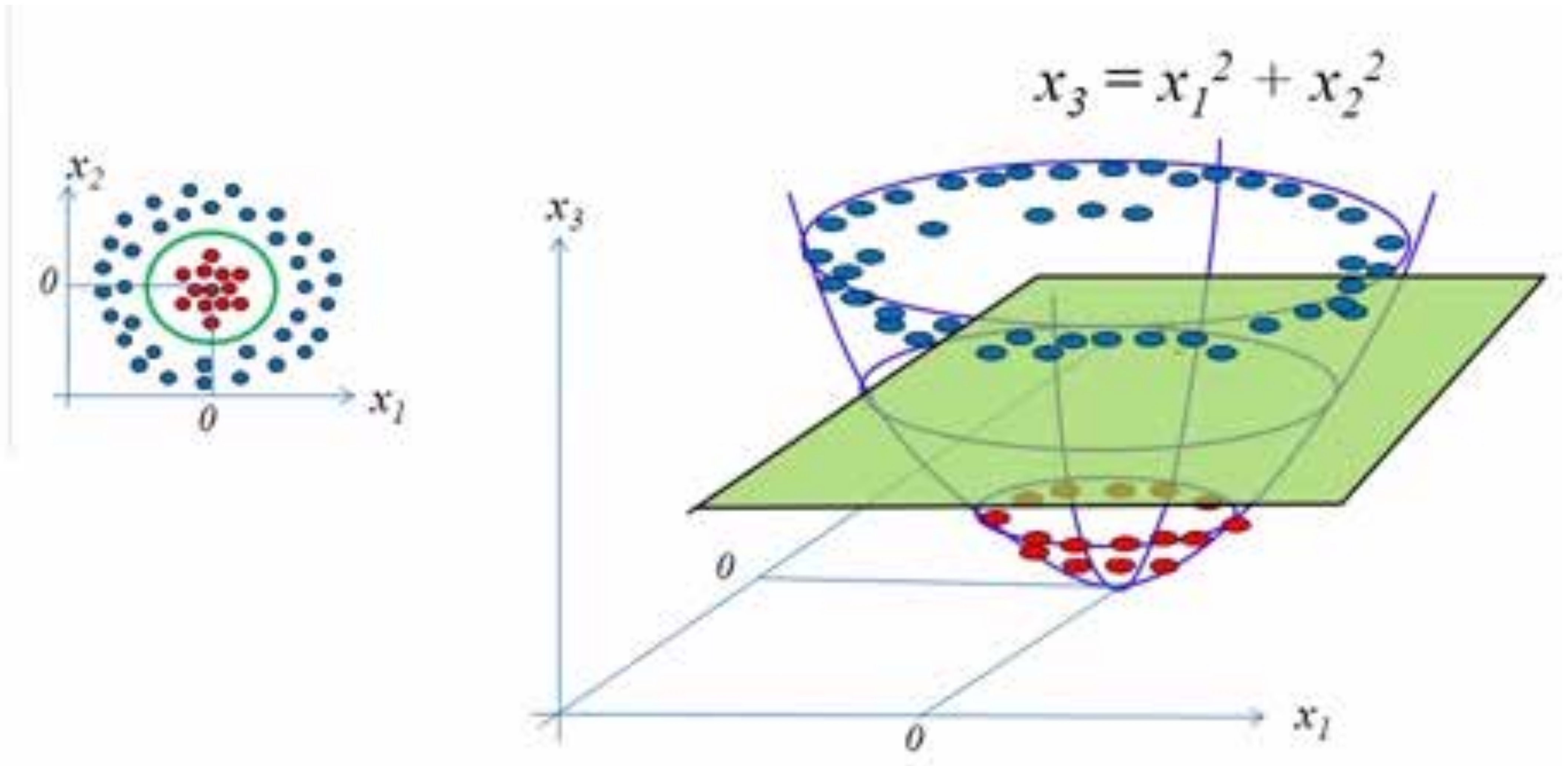
A multi-layer perceptron (MLP) supposedly should have solved this problem, if you could have trained it correctly.

Kernel Trick



Training MLPs was not easy. So people tended to leave NNs and use Kernel Trick instead.

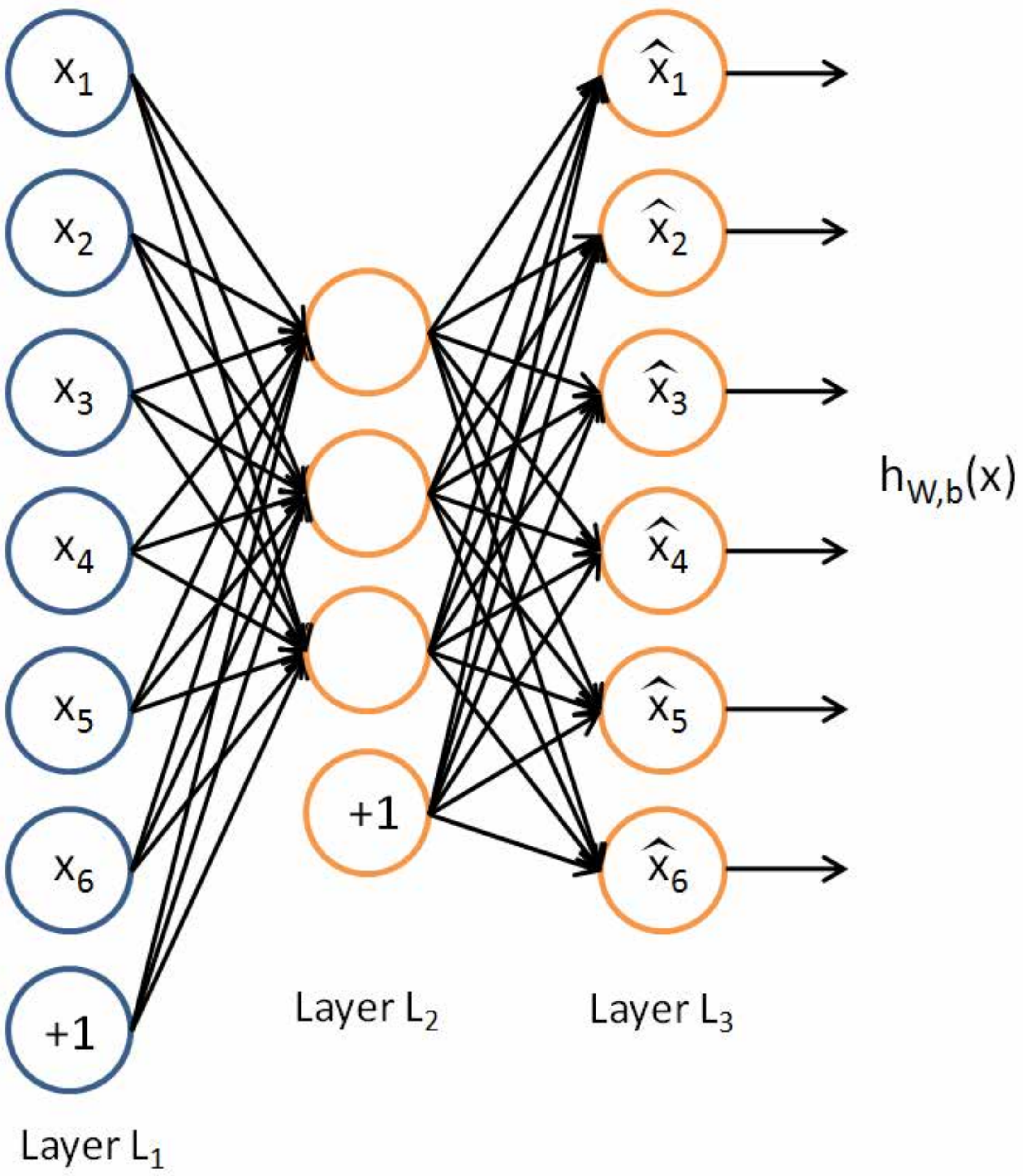
Kernel Trick Illustrated



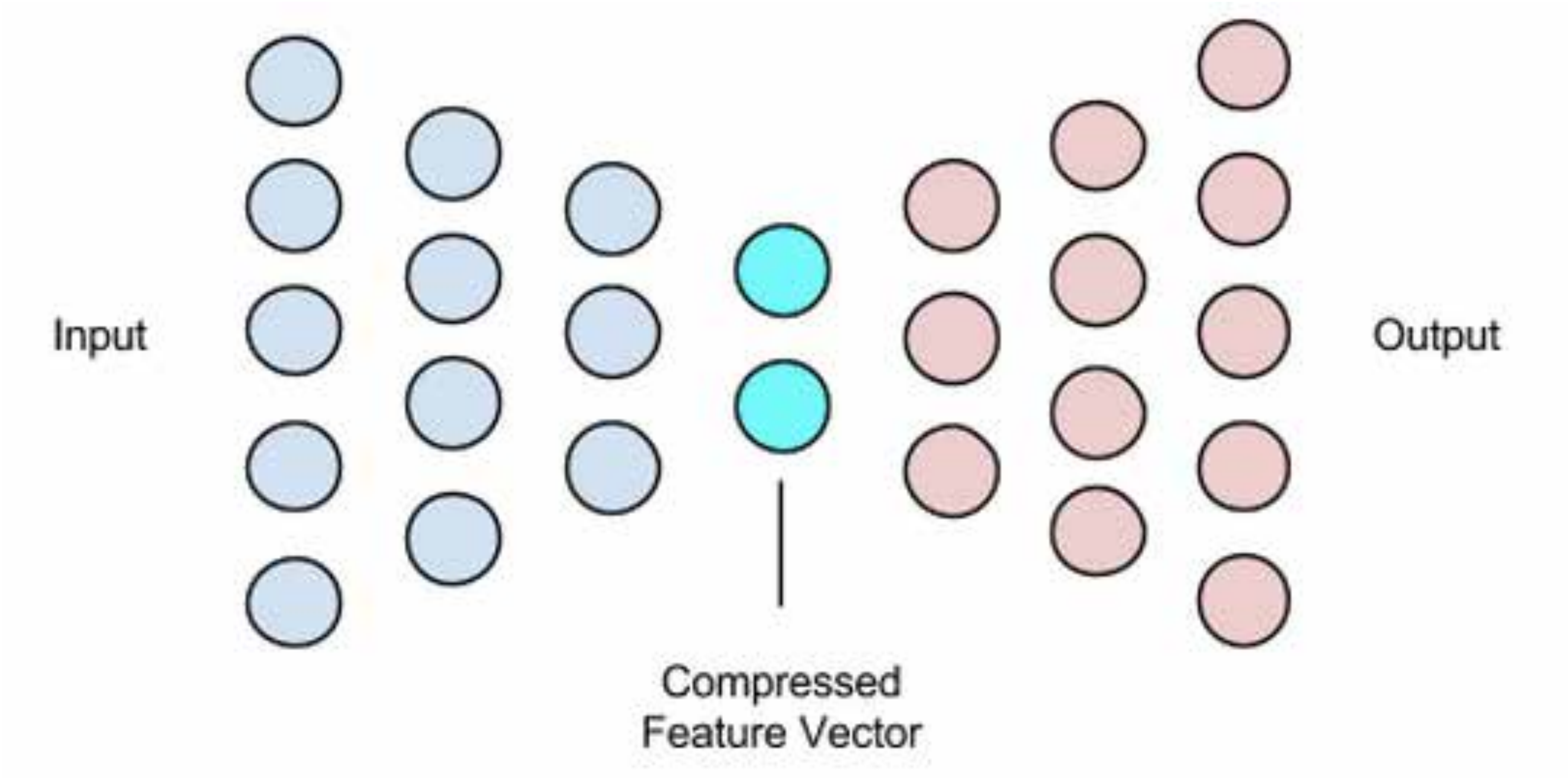
Auto-encoders

- What was hard about training deep MLPs?

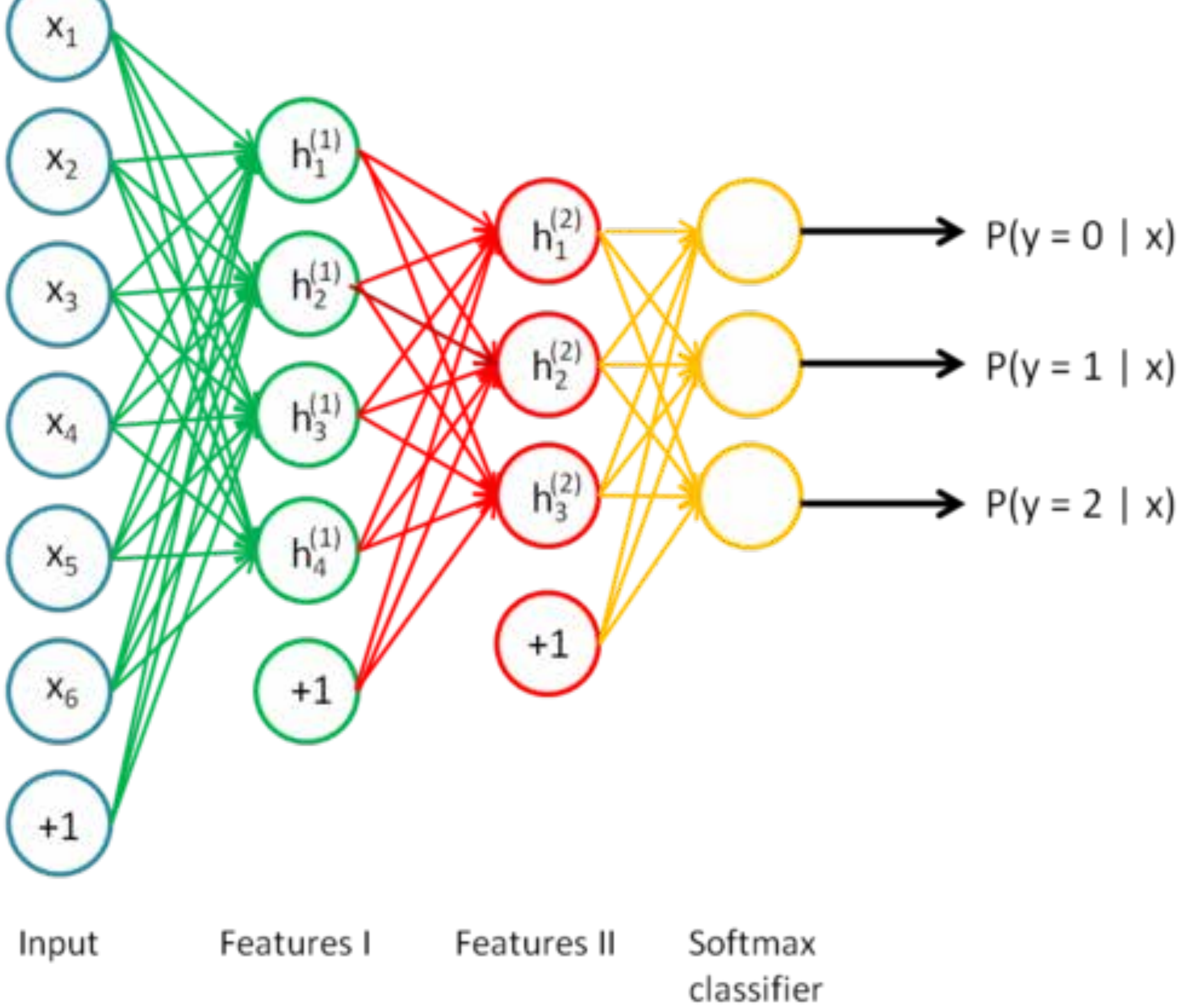
- An auto-encoder tries to match the input to the output!
- Basically, does nothing, right?
- Not right.
- It depends on the number of neurons in hidden layer.

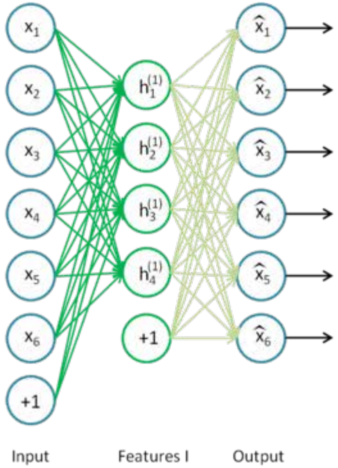


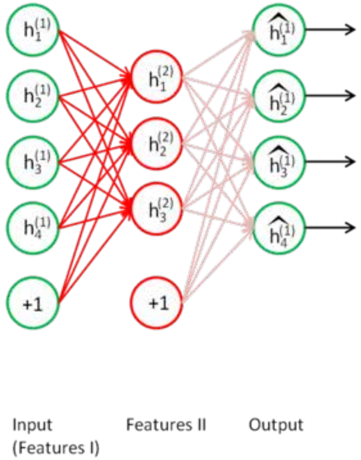
Deep Auto-encoders: Training them is too unstable

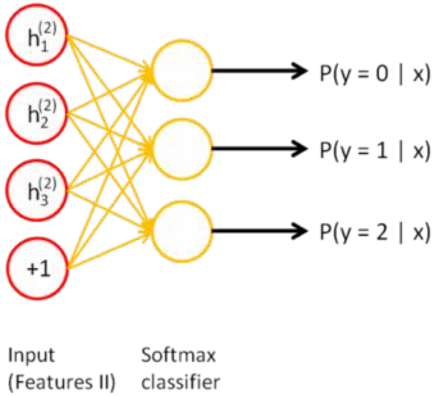


Trick: Stacked Auto-Encoders

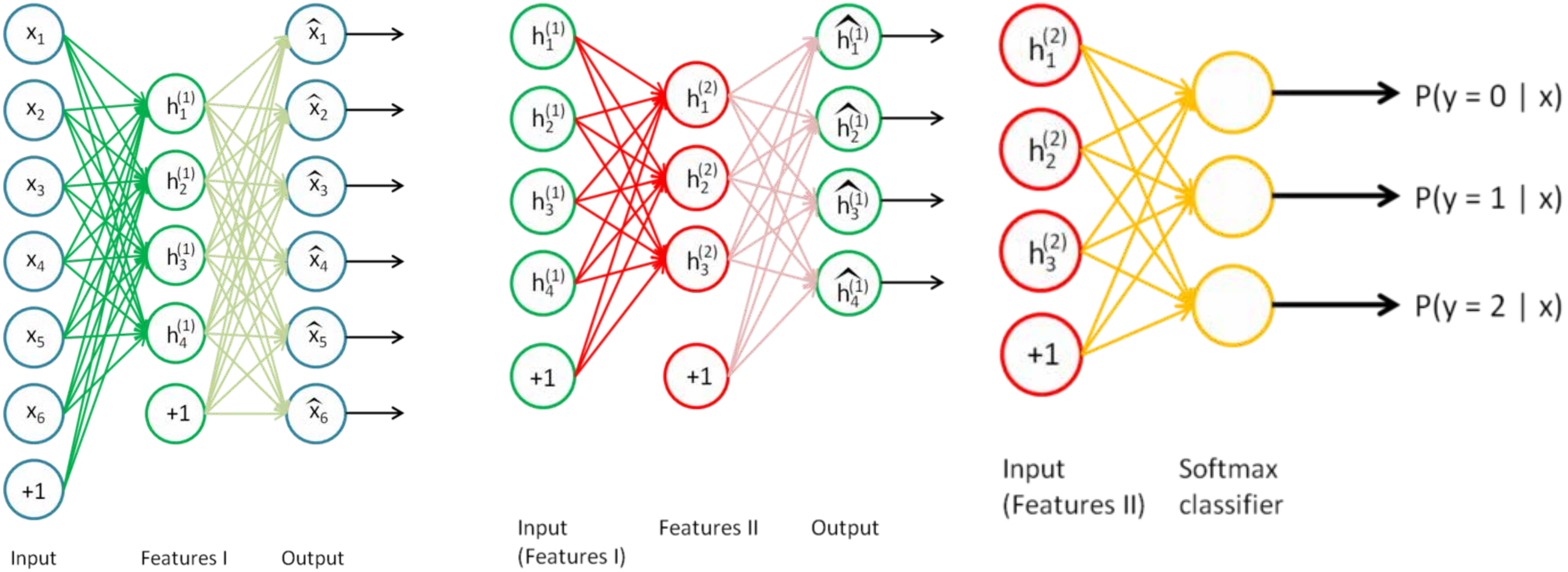








Stacked Auto-Encoders:



Use learned weights as initial weights of real training!

Sparse Auto-Encoder

- If hidden layer has more neurons than its input, it is a redundant layer.
- It is guaranteed to over-fit as there are numerous different ways of producing the same output in this case.
- We need to impose extra restrictions in training so that only some of the neurons will be trained, others will have zero weights.
- If we add an extra penalty term to our cost function back-prop algorithm will also optimize this for us.
- This penalty is normally add the absolute value of all the activations of the layer’s neurons.

Deep Neural Networks - Remember

- DNNs have extremely high complexity
  - Using stacked auto-encoders for initialization solved our training problem, but this doesn’t change the fact that their represented model is too complex.
  - They have a lot of parameters to learn (namely weights and biases)
  - Do we have enough data to feed this hungry network?
  - Even if we have data, do we have the time to train them with so much data?
  - No!
  - We need ways to cut down our DNN complexity.
  - Consider Computer Vision applications (M-NIST, Yale dataset):
    We also want invariance to translation and illumination if possible.

Translation Invariance? (M-NIST Dataset)

(a) (b) (c)

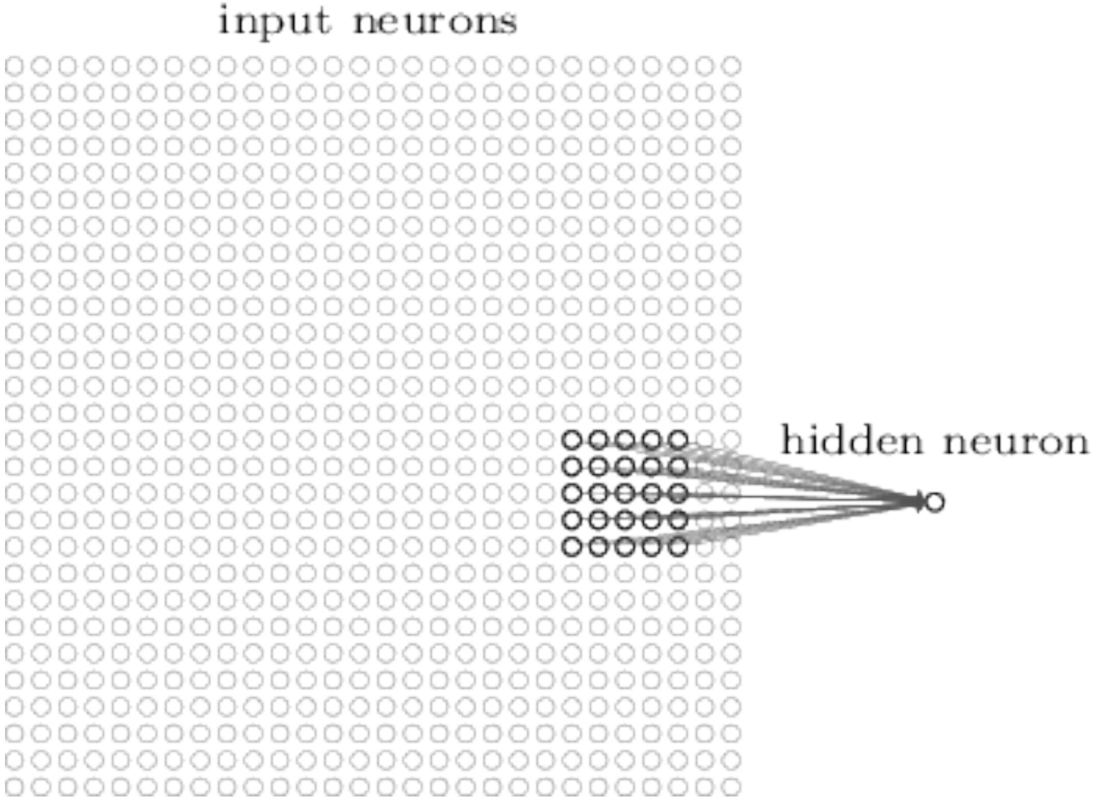
Illumination Invariance? (Yale Dataset)

Original Yale images

Processed Yale images

First simplification: Local Receptive Fields

- Consider a 2D representation of input neurons (e.g. M-NIST data)
- Consider a specific neuron in the hidden layer.
- Now, it will be connected to a local neighborhood only
  - Instead of connected it to each and every input



Beginning of Convolution Sub-module

- Let’s remember convolution and its applications in Computer Vision for a few slides.

- Why?

- Because Convolutional Neural Networks, you know ☺

Image filtering

- Image filtering: compute function of local neighborhood at each position
- Really important!
  - Enhance images
    - Denoise, resize, increase contrast, etc.
  - Extract information from images
    - Texture, edges, distinctive points, etc.
  - Detect patterns
    - Template matching

Example: box filter

$$g[\cdot,\cdot]$$

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Slide credit: David Lowe (UBC)

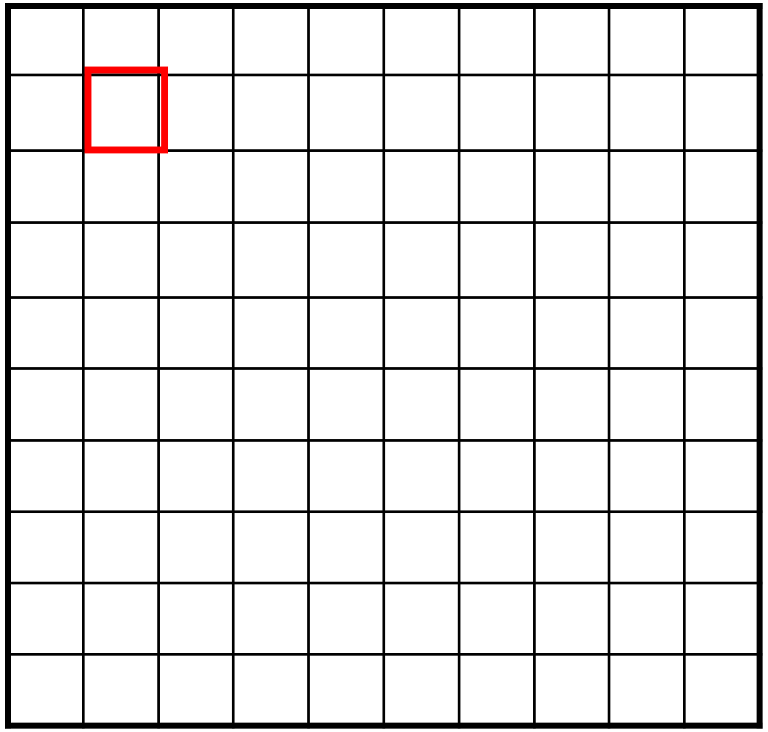
Image filtering

$$g[\cdot,\cdot]\ \frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$$

$f[.,.]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[.,.]$



$$h[m,n]=\sum_{k,l} g[k,l]\, f[m+k,n+l]$$

Credit: S. Seitz

Image filtering

$$g[\cdot\,,\cdot\,]\ \frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$$

$f[.,.]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[.,.]$

	0	10							

$$h[m,n]=\sum_{k,l} g[k,l]\, f[m+k,n+l]$$

Credit: S. Seitz

Image filtering

$g[\cdot,\cdot]\ \frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[.,.]$

	0	10	20						

$$h[m,n]=\sum_{k,l} g[k,l]\, f[m+k,n+l]$$

Credit: S. Seitz

Image filtering

$$g[\cdot\,,\cdot\,]\ \frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

$f[.,.]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[.,.]$

	0	10	20	30					

$$h[m,n]=\sum_{k,l} g[k,l]\, f[m+k,n+l]$$

Credit: S. Seitz

Image filtering

$$g[\cdot\,,\cdot\,]\ \frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

$$f[.,.]$$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$$h[.,.]$$

	0	10	20	30	30				

$$h[m,n]=\sum_{k,l} g[k,l]\, f[m+k,n+l]$$

Credit: S. Seitz

Image filtering

$$g[\cdot\,,\cdot\,]\ \frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

$f[.,.]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[.,.]$

	0	10	20	30	30				
				?					

$$h[m,n]=\sum_{k,l} g[k,l]\, f[m+k,n+l]$$

Credit: S. Seitz

Image filtering

$g[\cdot,\cdot]\ \frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[.,.]$

	0	10	20	30	30				
						?			
				50					

$$h[m,n]=\sum_{k,l} g[k,l]\, f[m+k,n+l]$$

Credit: S. Seitz

Image filtering

$$g[\cdot\,,\cdot\,]\ \frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$$

$f[.,.]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[.,.]$

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

Credit: S. Seitz

Box Filter

What does it do?

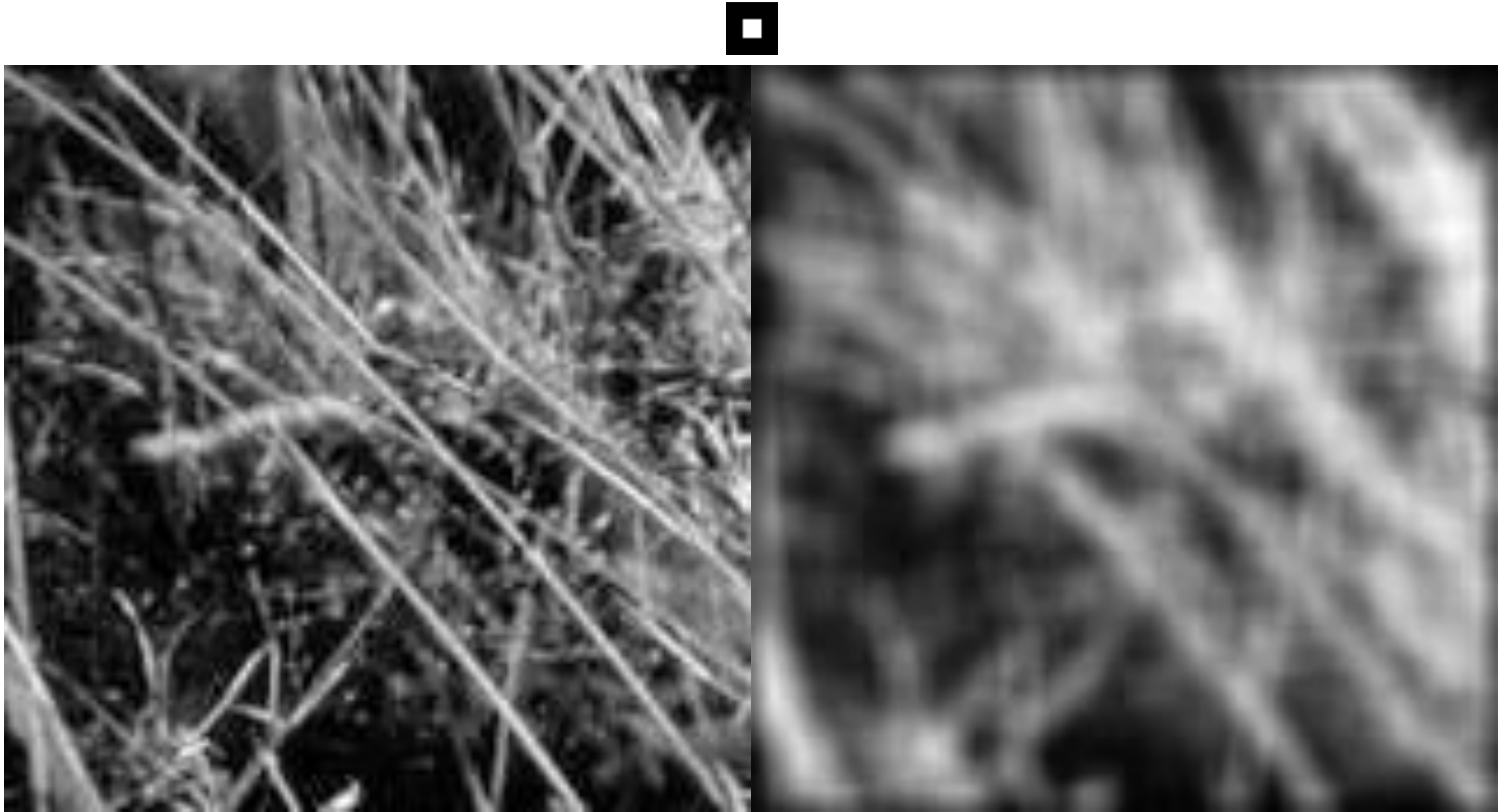
- Replaces each pixel with an average of its neighborhood
- Achieve smoothing effect (remove sharp features)

$$g[\cdot\,,\cdot\,]$$

$$\frac{1}{9}\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline\end{array}$$

Slide credit: David Lowe (UBC)

Smoothing with box filter



Practice with linear filters

Original

0	0	0
0	1	0
0	0	0

?

Source: D. Lowe

Practice with linear filters

Original

0	0	0
0	1	0
0	0	0

Filtered
(no change)

Source: D. Lowe

Practice with linear filters

Original

0	0	0
0	0	1
0	0	0

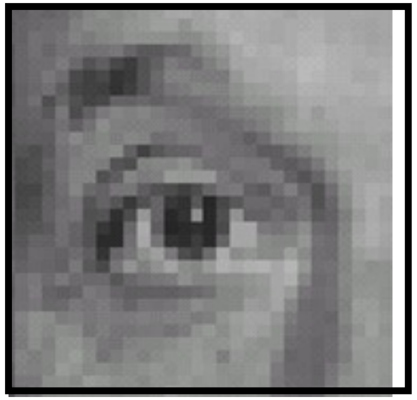
?

Source: D. Lowe

Practice with linear filters

Original

0	0	0
0	0	1
0	0	0



Shifted left
By 1 pixel

Source: D. Lowe

Practice with linear filters

Original

$$\begin{bmatrix}0&0&0\\0&2&0\\0&0&0\end{bmatrix} - \frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix} \qquad ?$$

(Note that filter sums to 1)

Source: D. Lowe

Practice with linear filters

Original

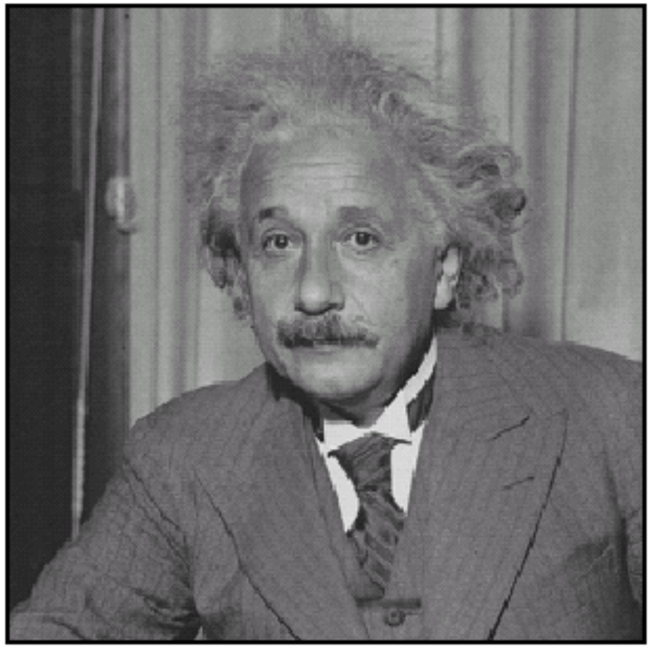
$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Sharpening filter

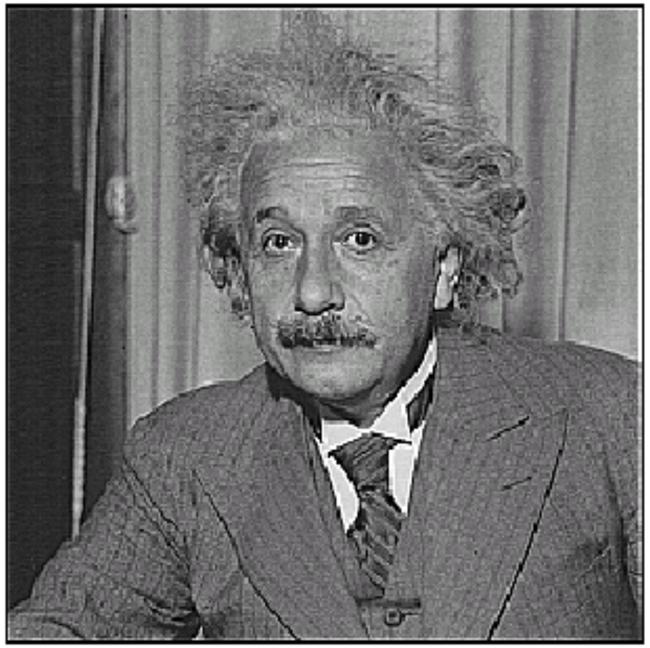
- Accentuates differences with local average

Source: D. Lowe

Sharpening



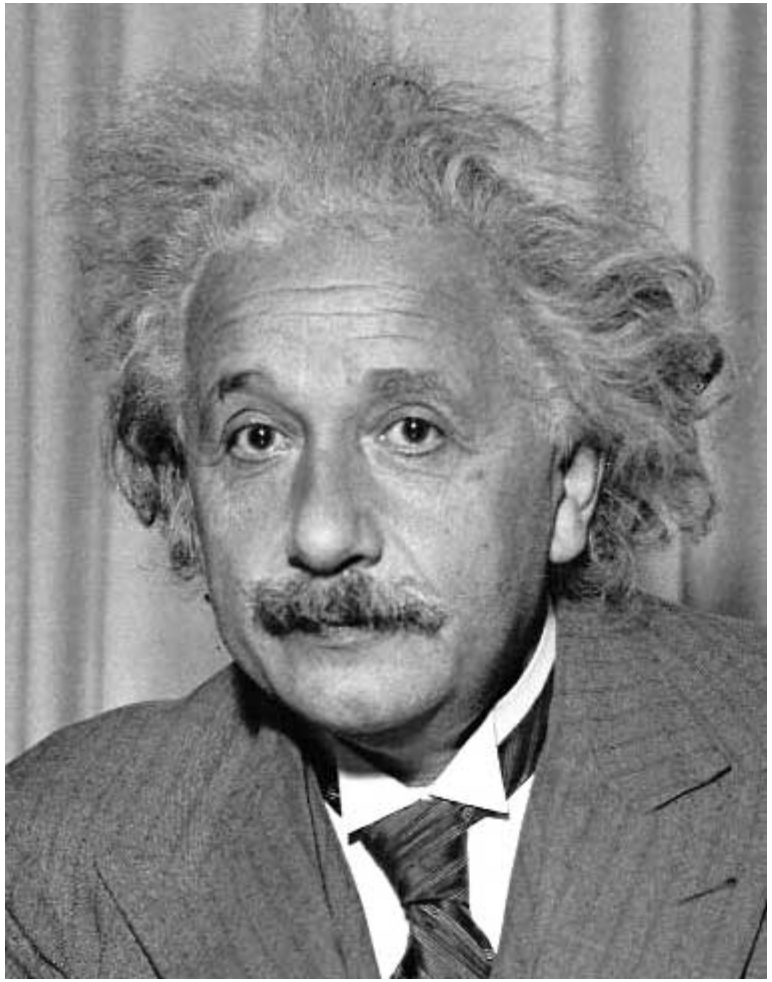
before



after

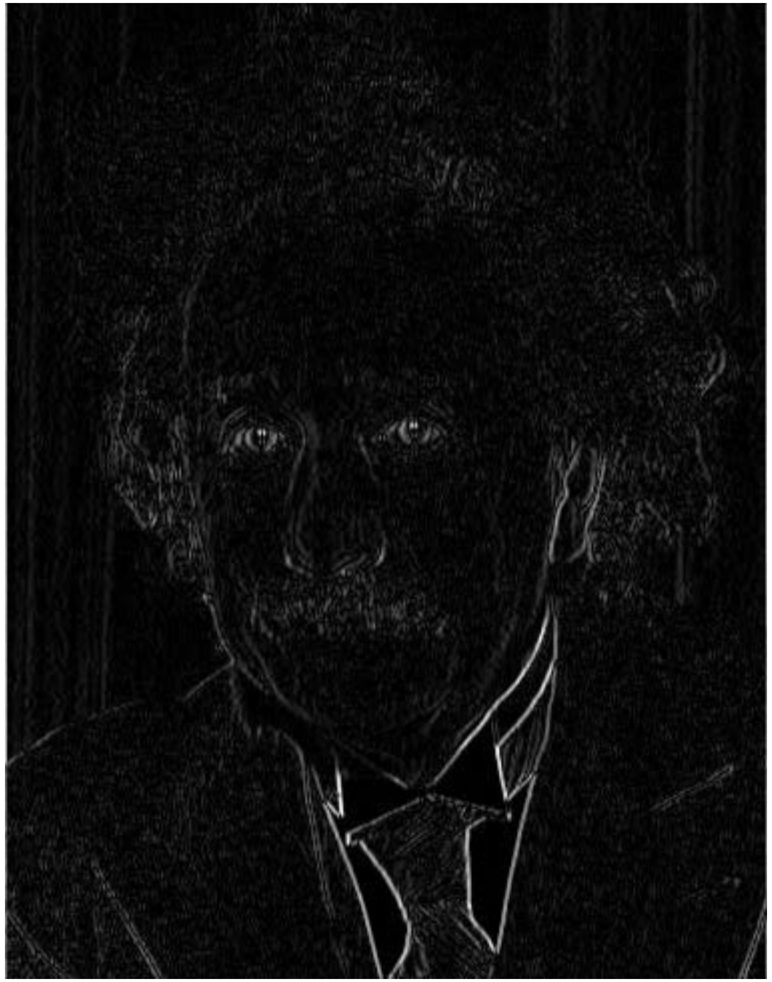
Source: D. Lowe

Other filters



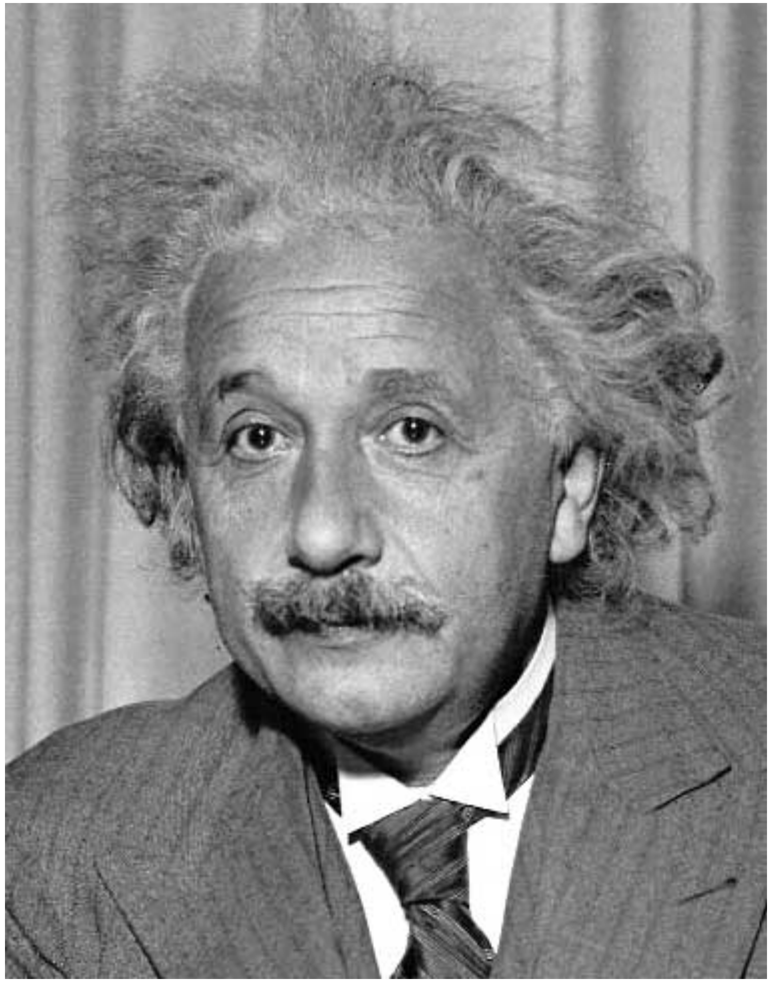
1	0	-1
2	0	-2
1	0	-1

Sobel



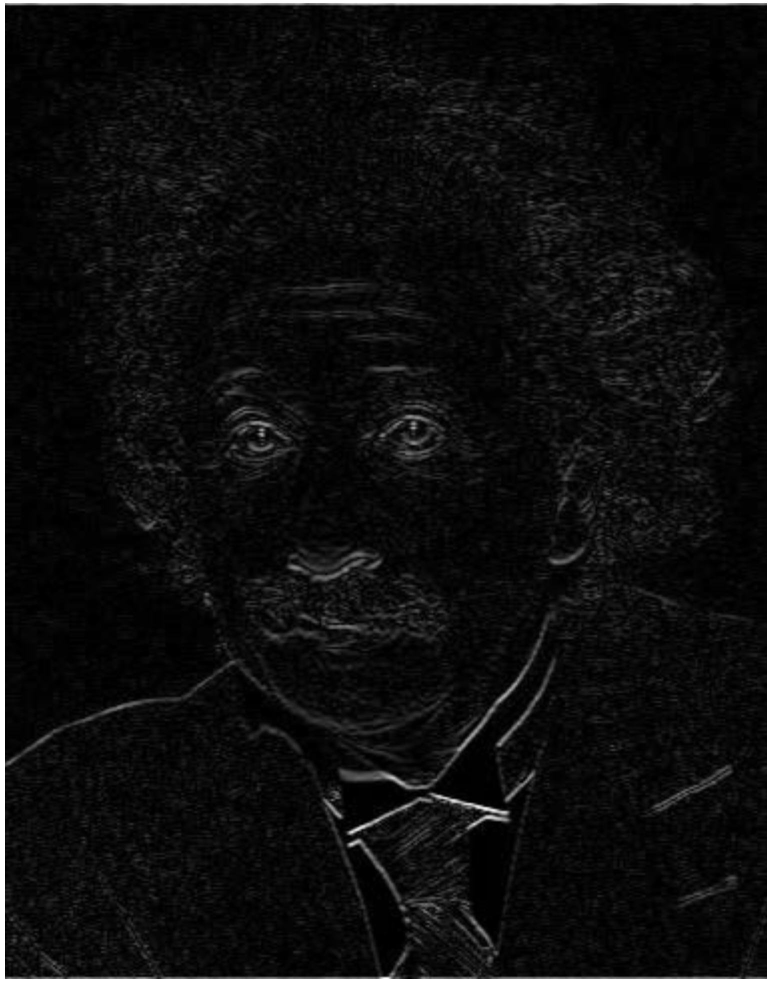
Vertical Edge
(absolute value)

Other filters



1	2	1
0	0	0
-1	-2	-1

Sobel



Horizontal Edge
(absolute value)

Filtering vs. Convolution

f = image g = filter

- 2d filtering

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

- 2d convolution

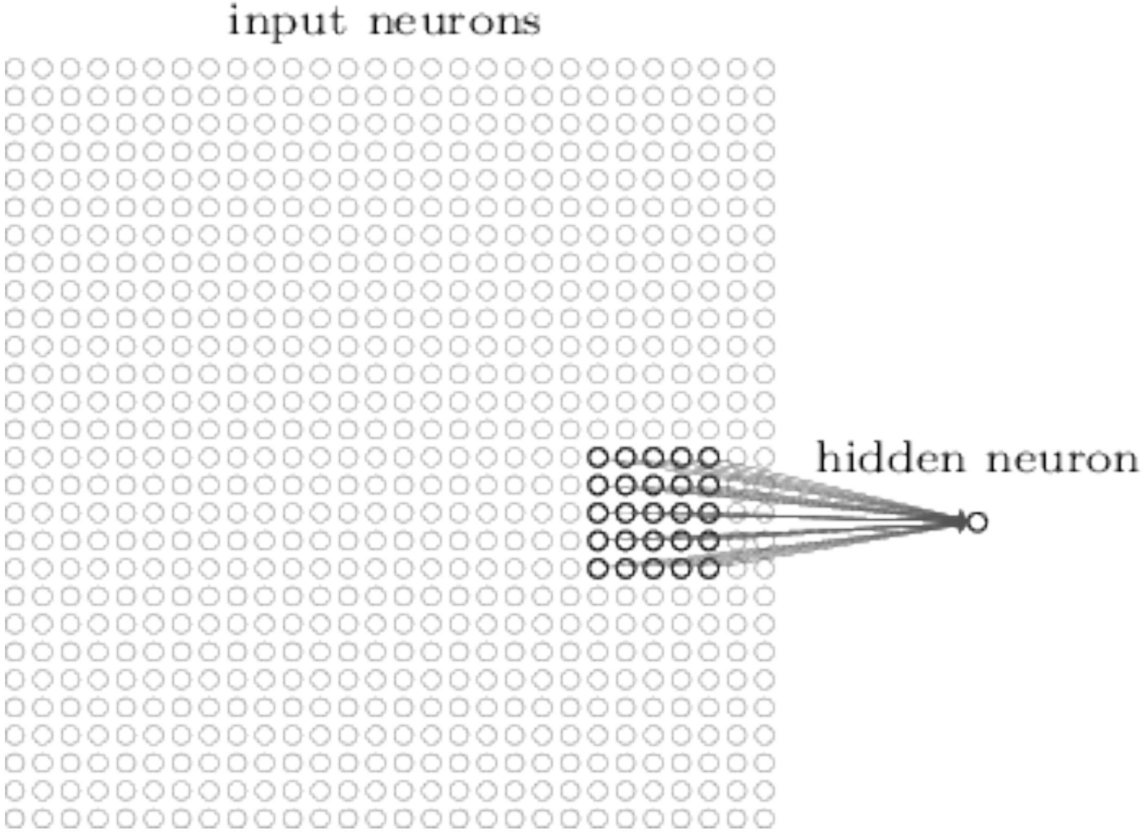
$$h[m,n] = \sum_{k,l} g[k,l]\, f[m-k, n-l]$$

End of Convolution Sub-module

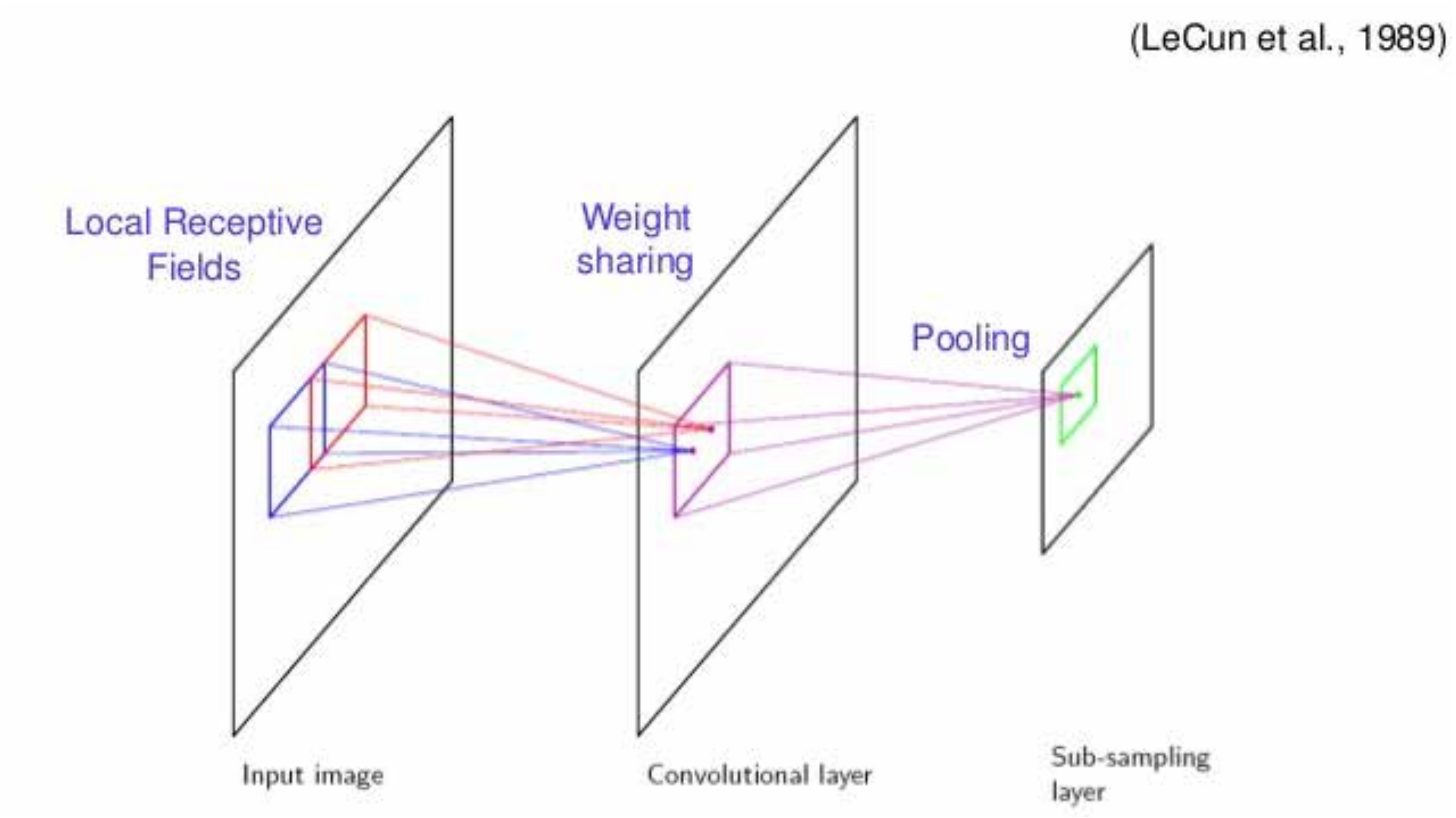
- Let's get to our 2nd simplification about our Deep Neural Network.

First simplification: Local Receptive Fields (AGAIN)

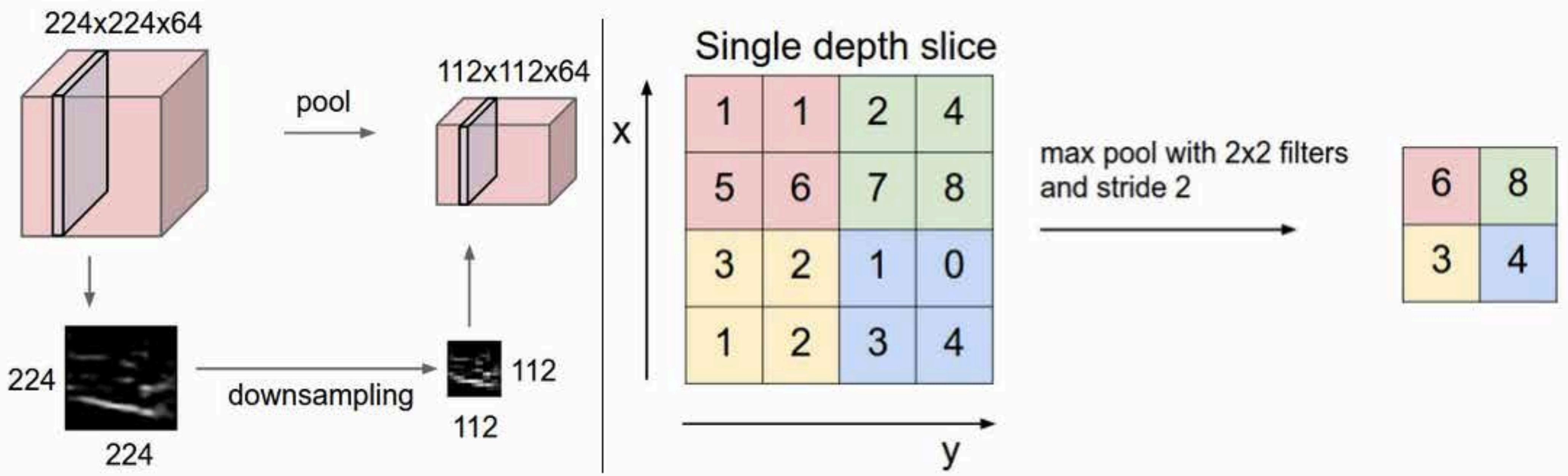
- Consider a 2D representation of input neurons (e.g. M-NIST data)
- Consider a specific neuron in the hidden layer.
- Now, it will be connected to a local neighborhood only
  - Instead of connected it to each and every input



Second Simplification: Shared Weights



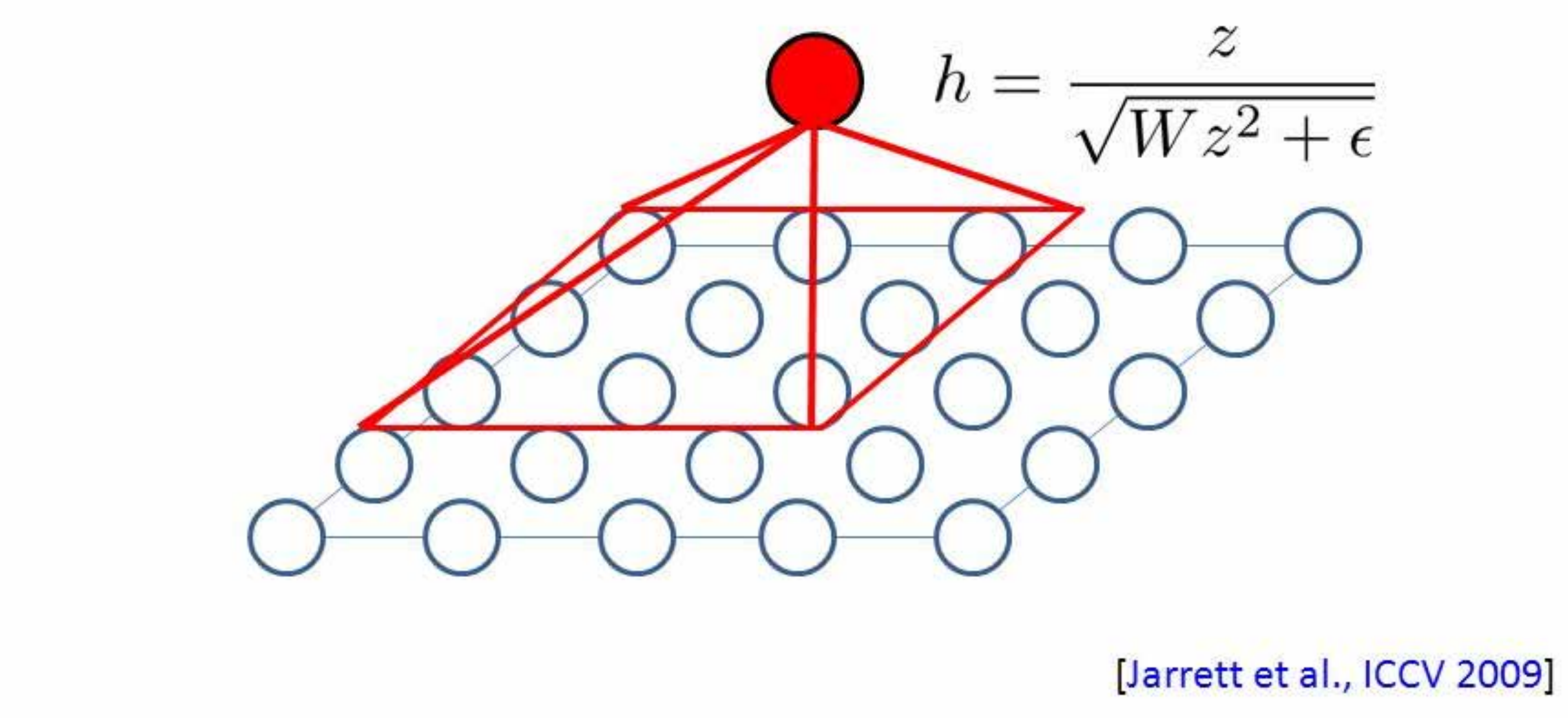
For translation invariance: Max-pooling



Pooling layer downsamples the volume spatially, independently in each depth slice of the input volume. **Left:** In this example, the input volume of size [224x224x64] is pooled with filter size 2, stride 2 into output volume of size [112x112x64]. Notice that the volume depth is preserved. **Right:** The most common downsampling operation is max, giving rise to **max pooling**, here shown with a stride of 2. That is, each max is taken over 4 numbers (little 2x2 square).

For illumination invariance: Local Contrast Normalization

- Empirically useful to soft-normalize magnitude of neurons
- Sometimes we first subtract out the mean, as well.



An example CNN pipeline