

# Deep Learning Basics

簡韶逸 Shao-Yi Chien

Department of Electrical Engineering  
National Taiwan University

# References and Slide Credits

- Slides from *Deep Learning for Computer Vision*, Prof. Yu-Chiang Frank Wang, National Taiwan University
- Slides from Machine Learning, Prof. Hung-Yi Lee, EE, National Taiwan University
- Slides from *CE 5554 / ECE 4554: Computer Vision*, Prof. J.-B. Huang, Virginia Tech
- <http://cs231n.stanford.edu/syllabus.html>
- Marc'Aurelio Ranzato, Tutorial in CVPR2014
- Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep Learning*
  - <https://www.deeplearningbook.org/>
- Bishop, *Pattern Recognition and Machine Learning*
- Reference papers

# Outline

- Introduction of neural network
- Go deeper
- Introduction of convolutional neural network (CNN)
- Modern CNN models

# History of Neural Network and Deep Learning

[Prof. Hung-Yi Lee]

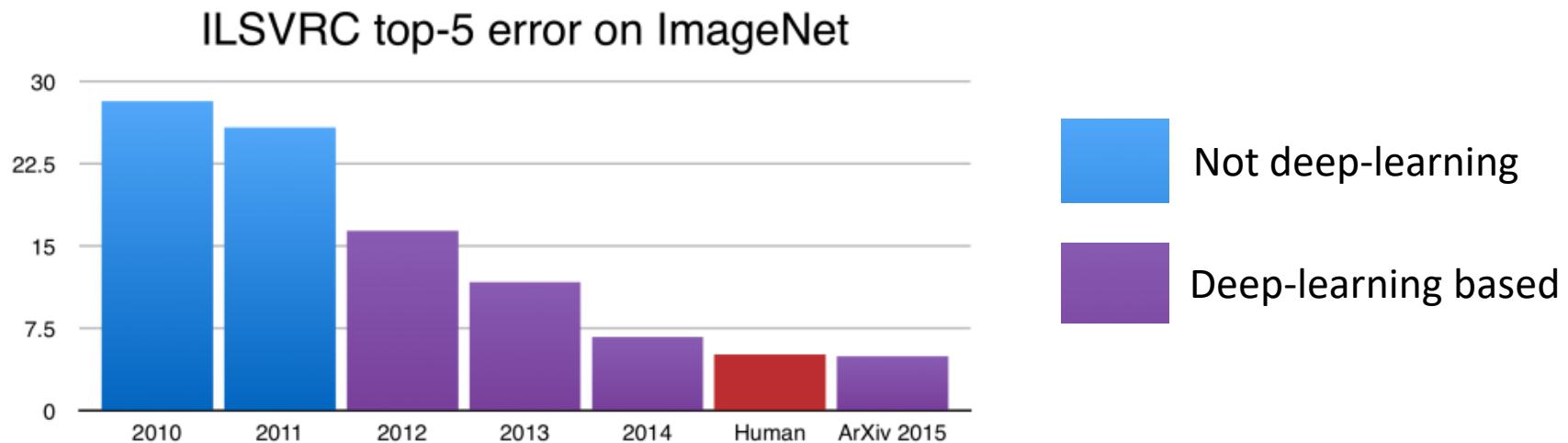
- 1958: Perceptron (linear model)
- 1969: Perceptron has limitation
- 1980s: Multi layer perceptron
  - Do not have significant difference from DNN today
- 1986: Backpropagation
  - Usually more than 3 hidden layers is not helpful
- 1989: 1 hidden layer is “good enough”, why deep?
- 2006: RBM initialization (breakthrough)
- 2009: GPU
- 2011: Start to be popular in speech recognition
- 2012: win ILSVRC image competition



Geoffrey Hinton

LeCun, Yann; Bengio, Yoshua; Hinton, Geoffrey, “Deep learning,” *Nature*, 2015.

# How Powerful? Object Recognition

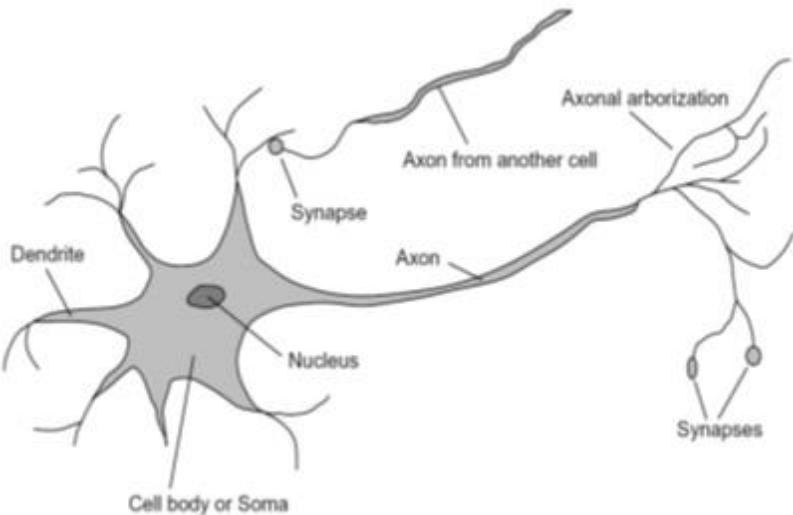


Source:

<https://devblogs.nvidia.com/parallelforall/mocha-jl-deep-learning-julia/>

<https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/>

# Biological neuron and Perceptrons



A biological neuron

Input

Weights

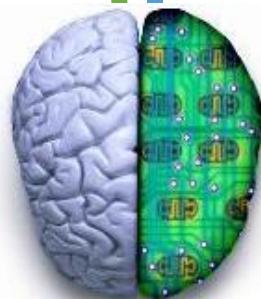
$$\begin{array}{l} x_1 \xrightarrow{w_1} \\ x_2 \xrightarrow{w_2} \\ x_3 \xrightarrow{w_3} \\ \vdots \\ \vdots \\ x_d \xrightarrow{w_d} \end{array}$$

Output:  $\sigma(w \cdot x + b)$

Sigmoid function:

$$\sigma(t) = \frac{1}{1 + e^{-t}}$$

An artificial neuron (Perceptron)  
- a linear classifier

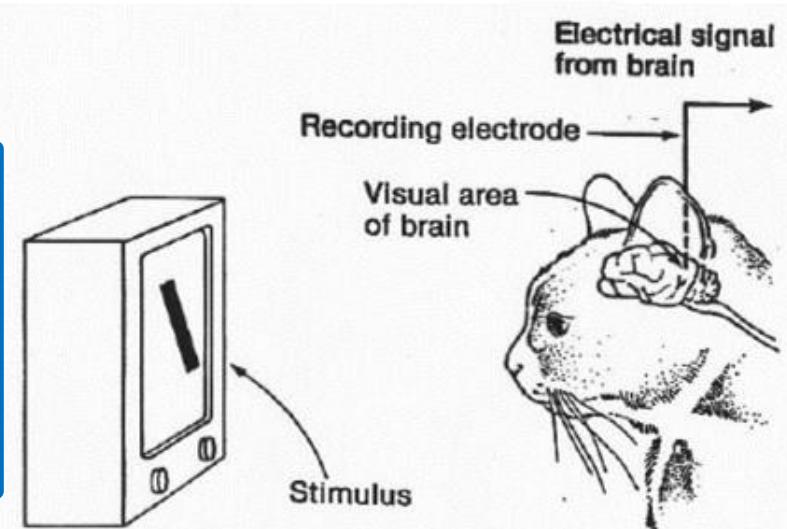


# Simple, Complex and Hypercomplex cells



David H. Hubel and Torsten Wiesel

Suggested a **hierarchy of feature detectors** in the visual cortex, with higher level features responding to patterns of activation in lower level cells, and propagating activation upwards to still higher level cells.

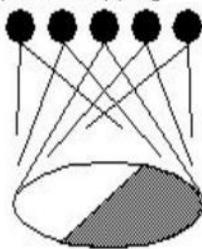


David Hubel's [Eye, Brain, and Vision](#)

# Hubel/Wiesel Architecture and Multi-layer Neural Network

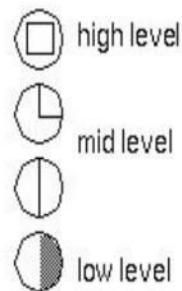
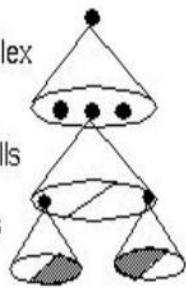
Hubel & Weisel

topographical mapping

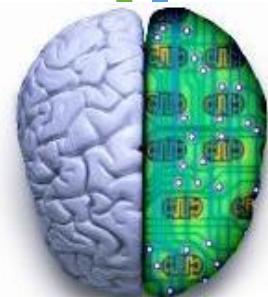


featural hierarchy

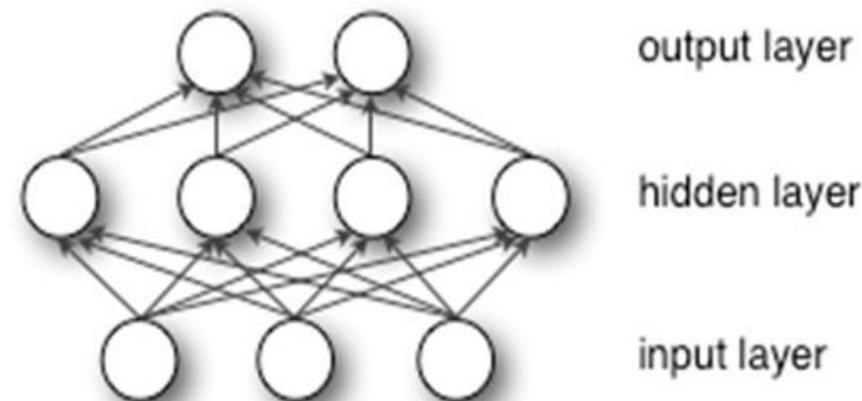
hyper-complex  
cells  
complex cells  
simple cells



Hubel and Weisel's architecture

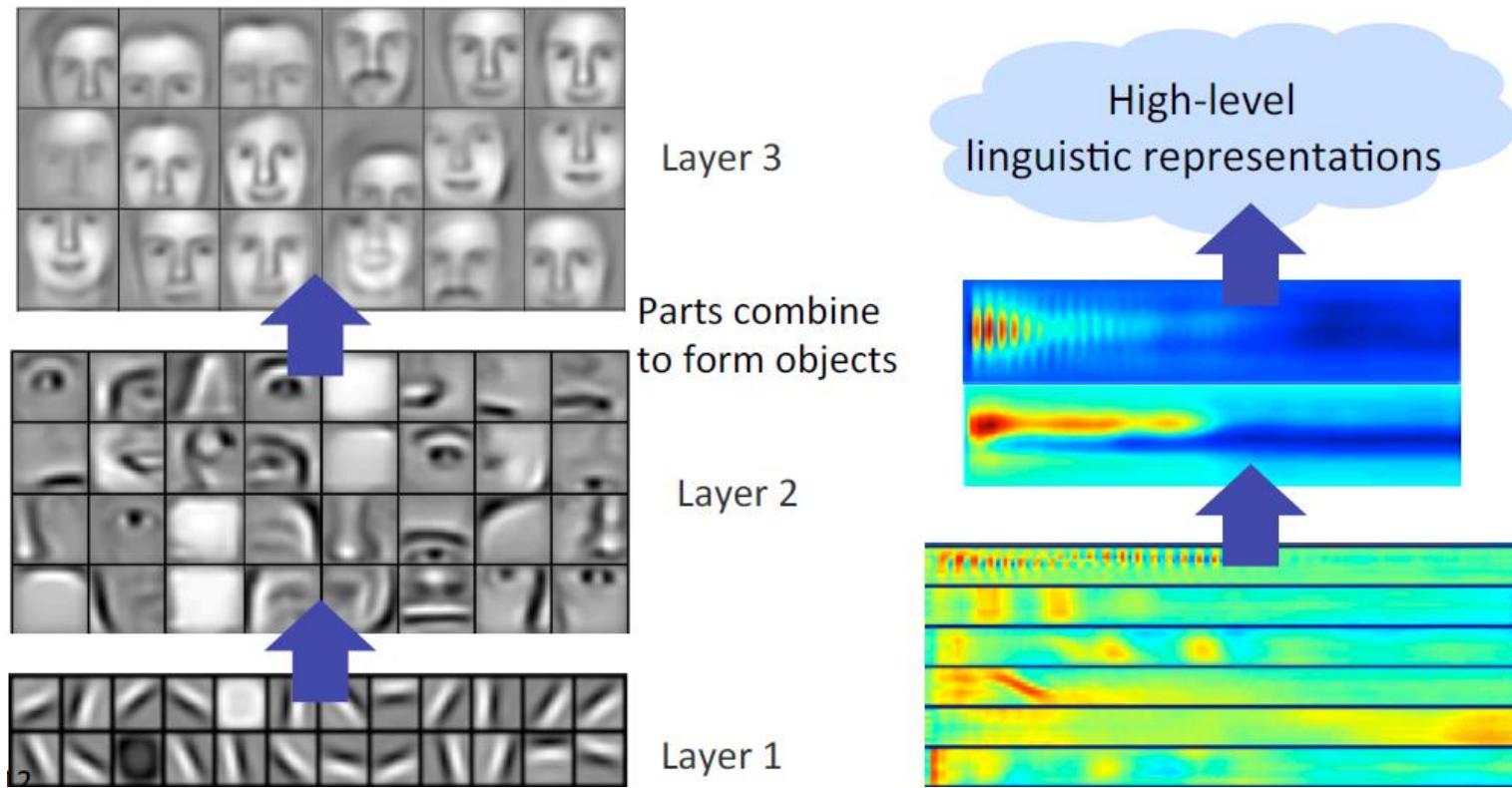


Multi-layer Neural Network  
- A *non-linear* classifier



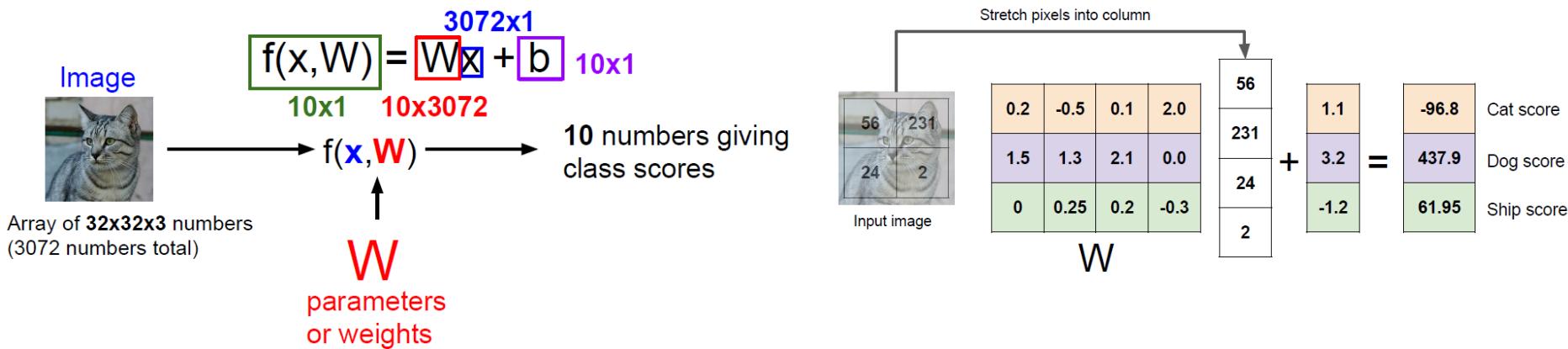
# Hierarchical Representation Learning

- Successive model layers learn deeper intermediate representations.

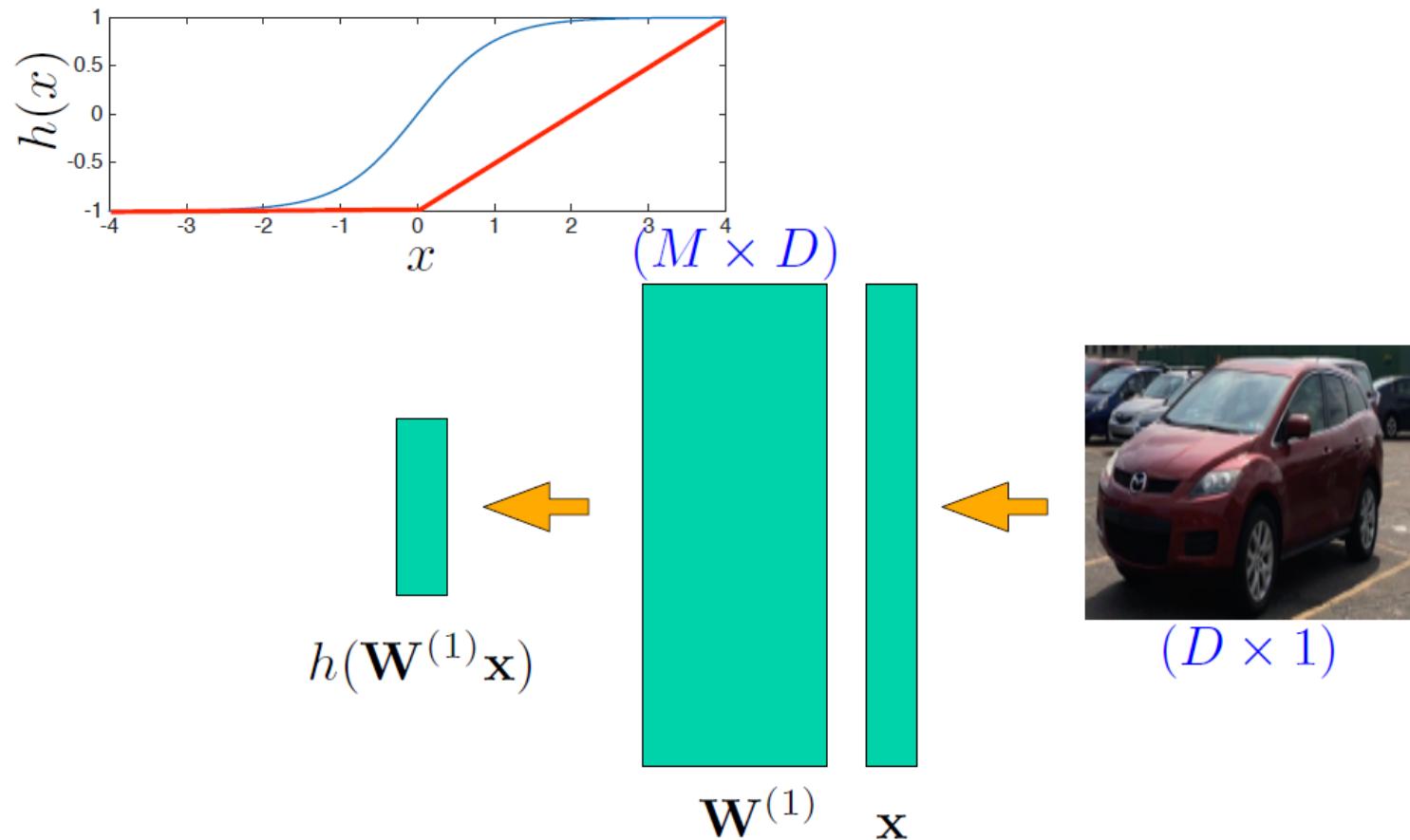


# Recap: Linear Classification

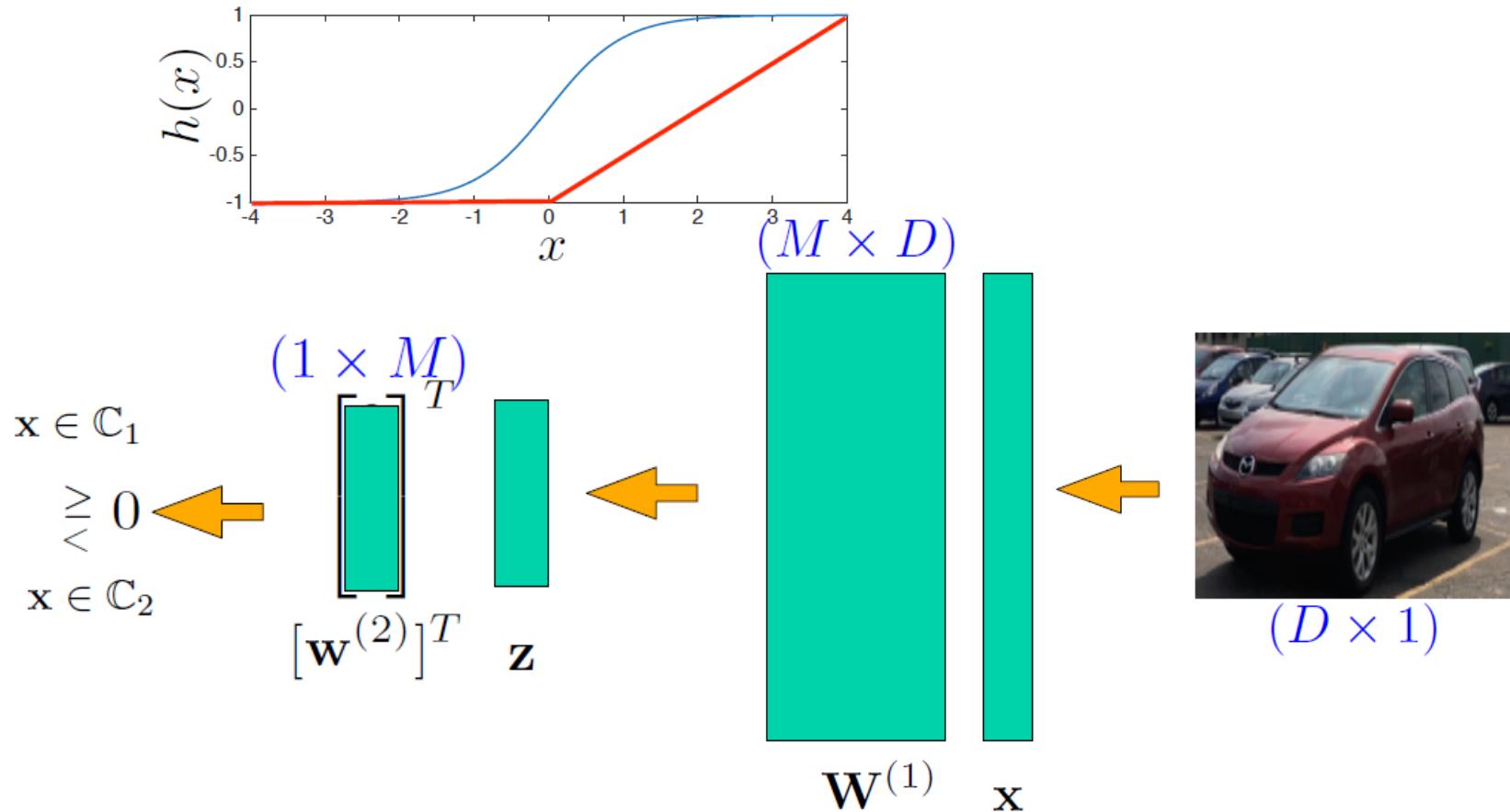
- Linear Classifier
  - Let's take the input image as  $\mathbf{x}$ , and the linear classifier as  $\mathbf{W}$ . We need  $\mathbf{y} = \mathbf{Wx} + \mathbf{b}$  as a 10-dimensional output vector, indicating the score for each class.
  - For example, an image with  $2 \times 2$  pixels & 3 classes of interest we need to learn a linear classifier  $\mathbf{W}$  (plus a bias  $\mathbf{b}$ ), so that desirable outputs  $\mathbf{y} = \mathbf{Wx} + \mathbf{b}$  can be expected.



# Multi-Layer Perceptron: A Nonlinear Classifier

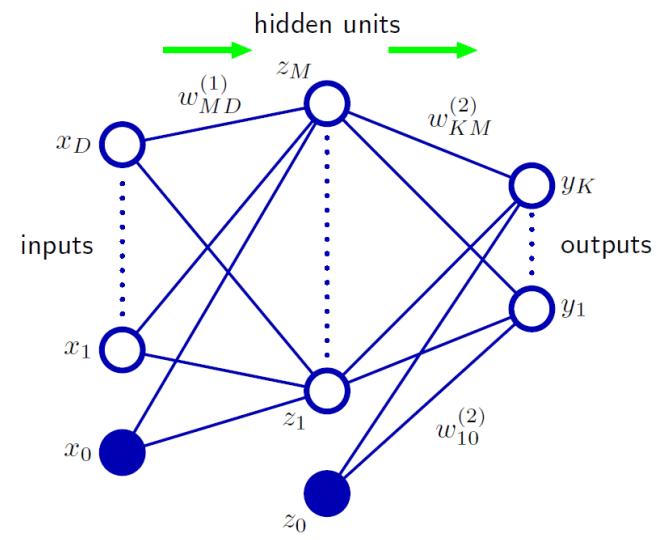


# Multi-Layer Perceptron: A Nonlinear Classifier (cont'd)



# Layer 1 in MLP

$$\mathbf{z} = \begin{bmatrix} z_1 \\ \vdots \\ z_M \end{bmatrix} \leftarrow \begin{bmatrix} h[\mathbf{x}^T \mathbf{w}_1^{(1)}] \\ \vdots \\ h[\mathbf{x}^T \mathbf{w}_M^{(1)}] \end{bmatrix}$$

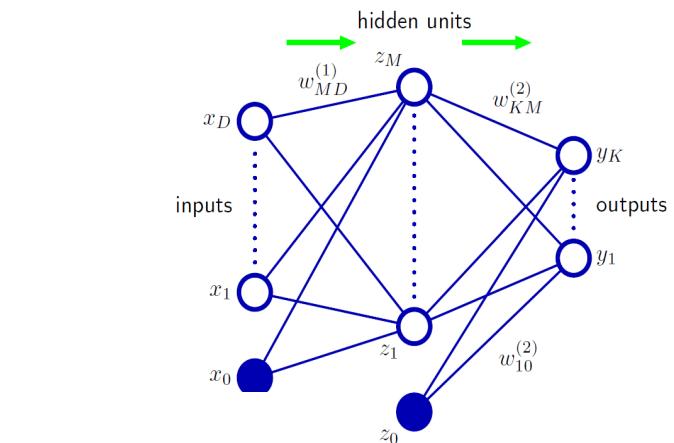


$h()$  = non-linear function

$[\mathbf{w}_1^{(1)}, \dots, \mathbf{w}_M^{(1)}]$  = 1st layer's  $D \times M$  weights

$\mathbf{x} = D \times 1$  raw input

# Layer 2 in MLP



$$\mathbf{x} \in \mathbb{R}^D$$

$$\mathbf{z} \in \mathbb{C}_1$$

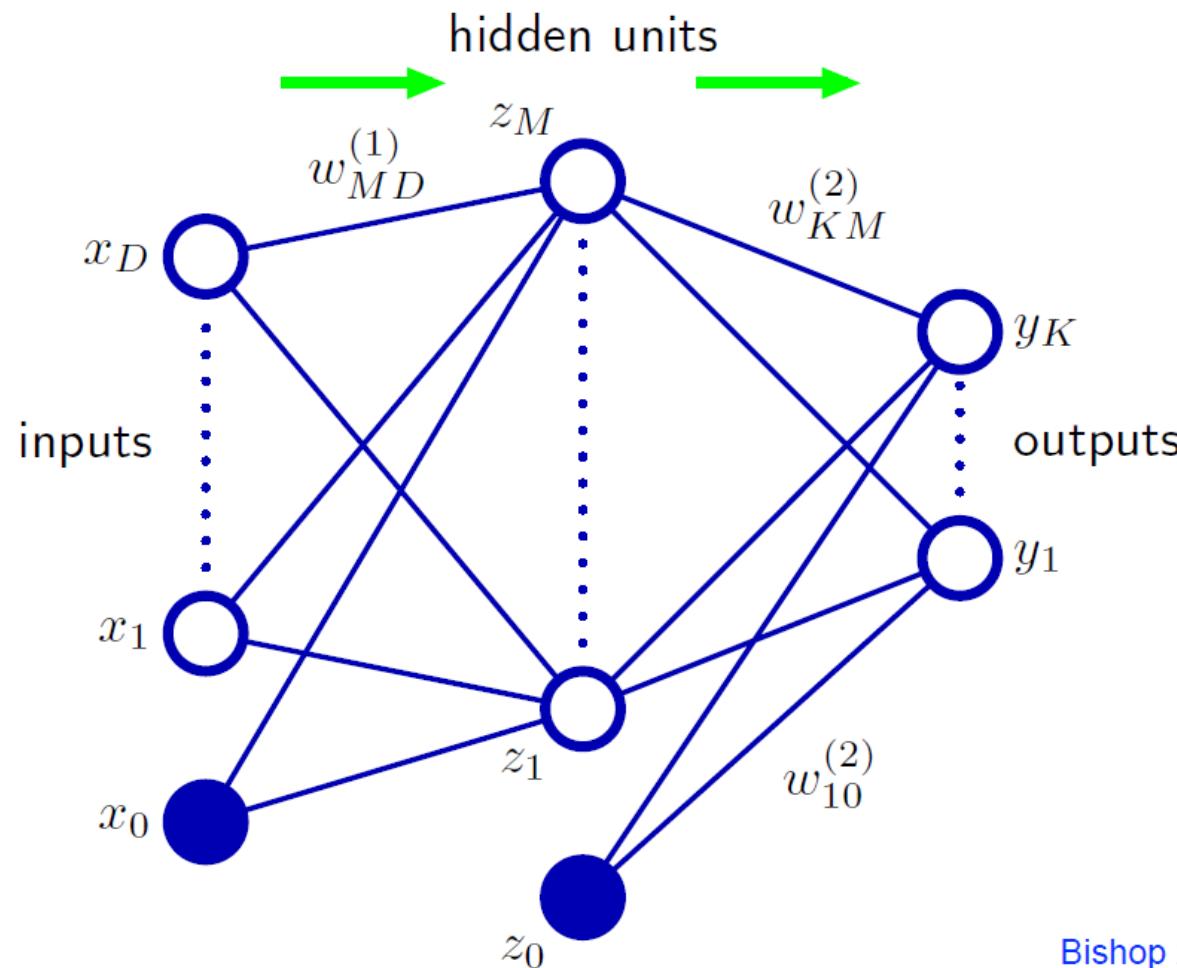
$$\mathbf{z}^T \mathbf{w}^{(2)} \geq 0$$

$$\mathbf{z} \in \mathbb{C}_2$$

$\mathbf{z} = M \times 1$  output of layer 1

$\mathbf{w}^{(2)} = 2\text{nd layer's } M \times 1$  weight vector

# Multi-Layer Perceptron: A Nonlinear Classifier (cont'd)

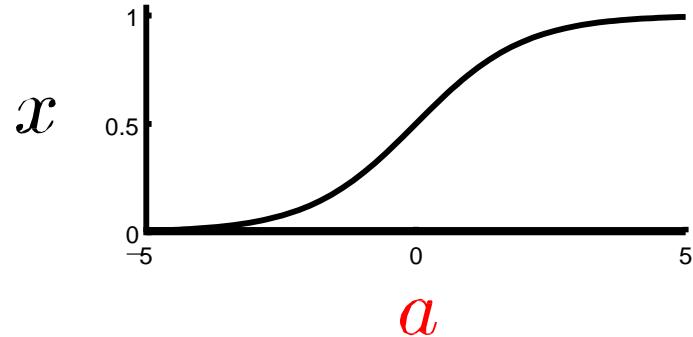
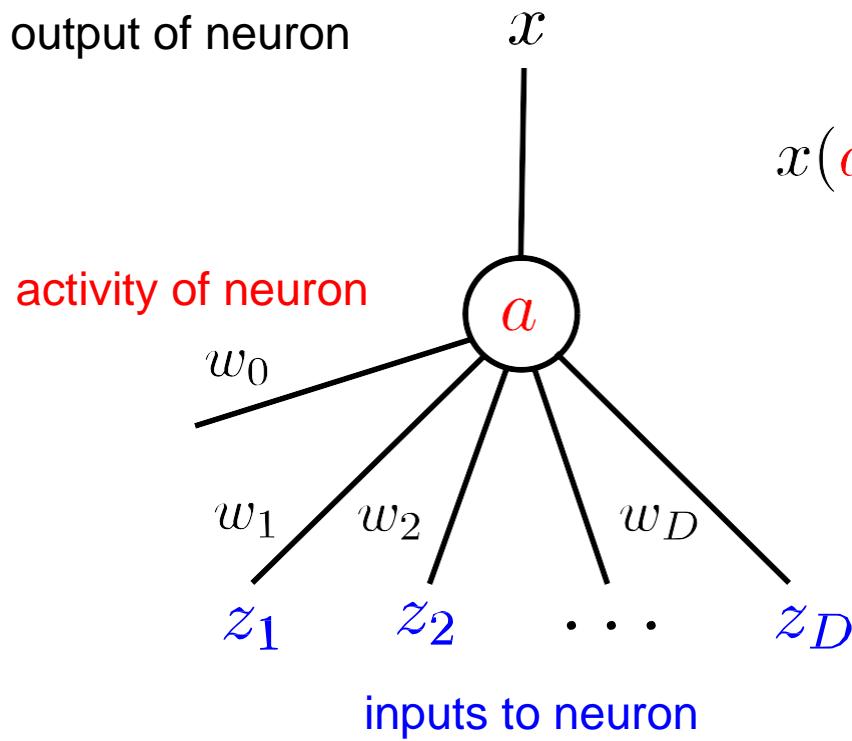


Bishop 2006

# Let's Get a Closer Look...

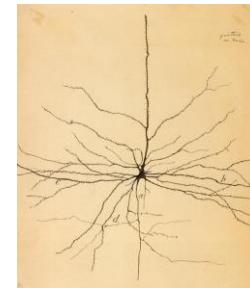
- A single neuron

output of neuron

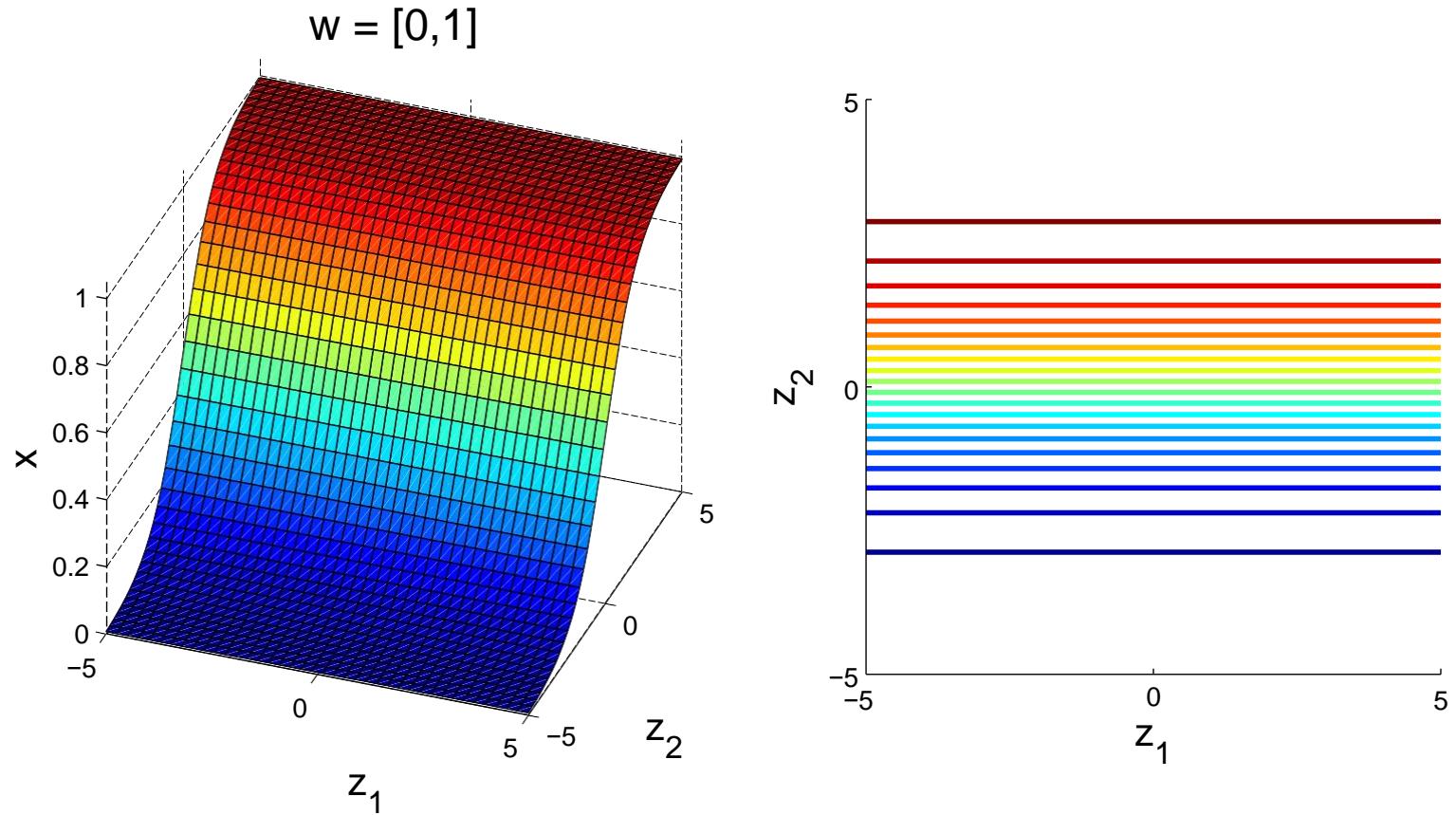


$$x(a) = \frac{1}{1+\exp(-a)} \quad x \in (0, 1)$$

$$\begin{aligned} a &= w_0 + \sum_{d=1}^D w_d z_d \\ &= \sum_{d=0}^D w_d z_d \end{aligned}$$

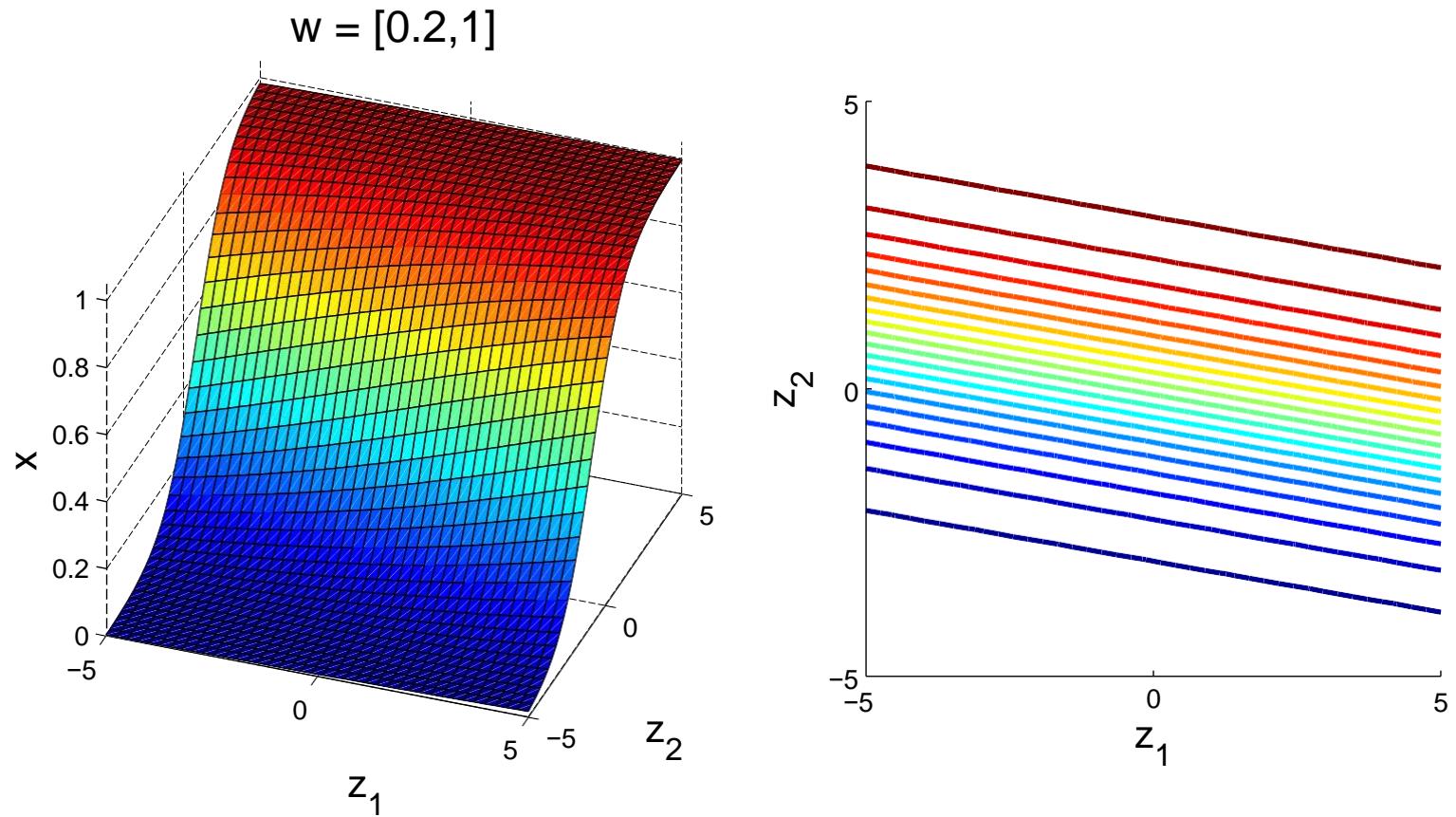


# Input-Output Function of a Single Neuron



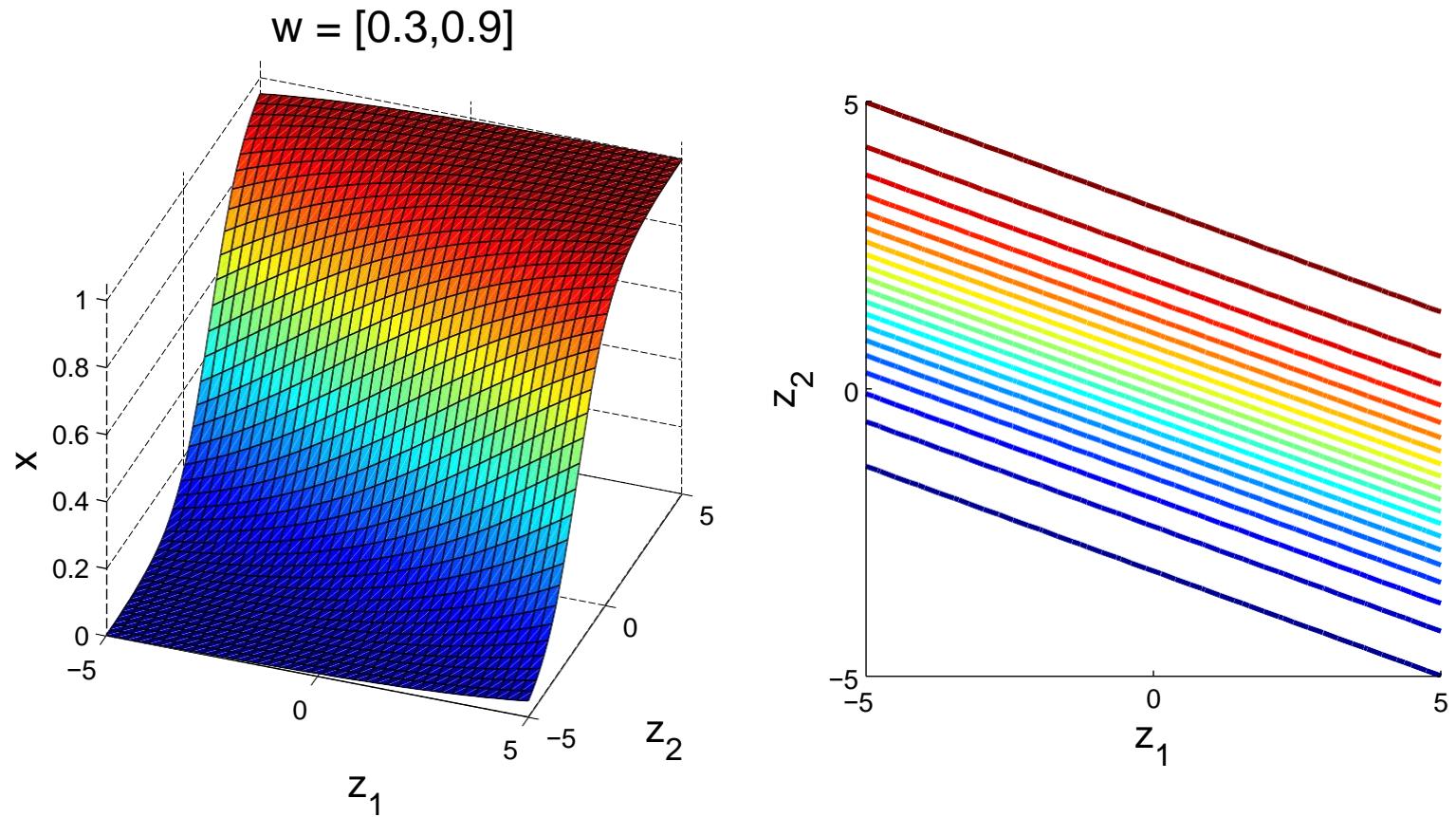
$$x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)}$$

# Input-Output Function of a Single Neuron



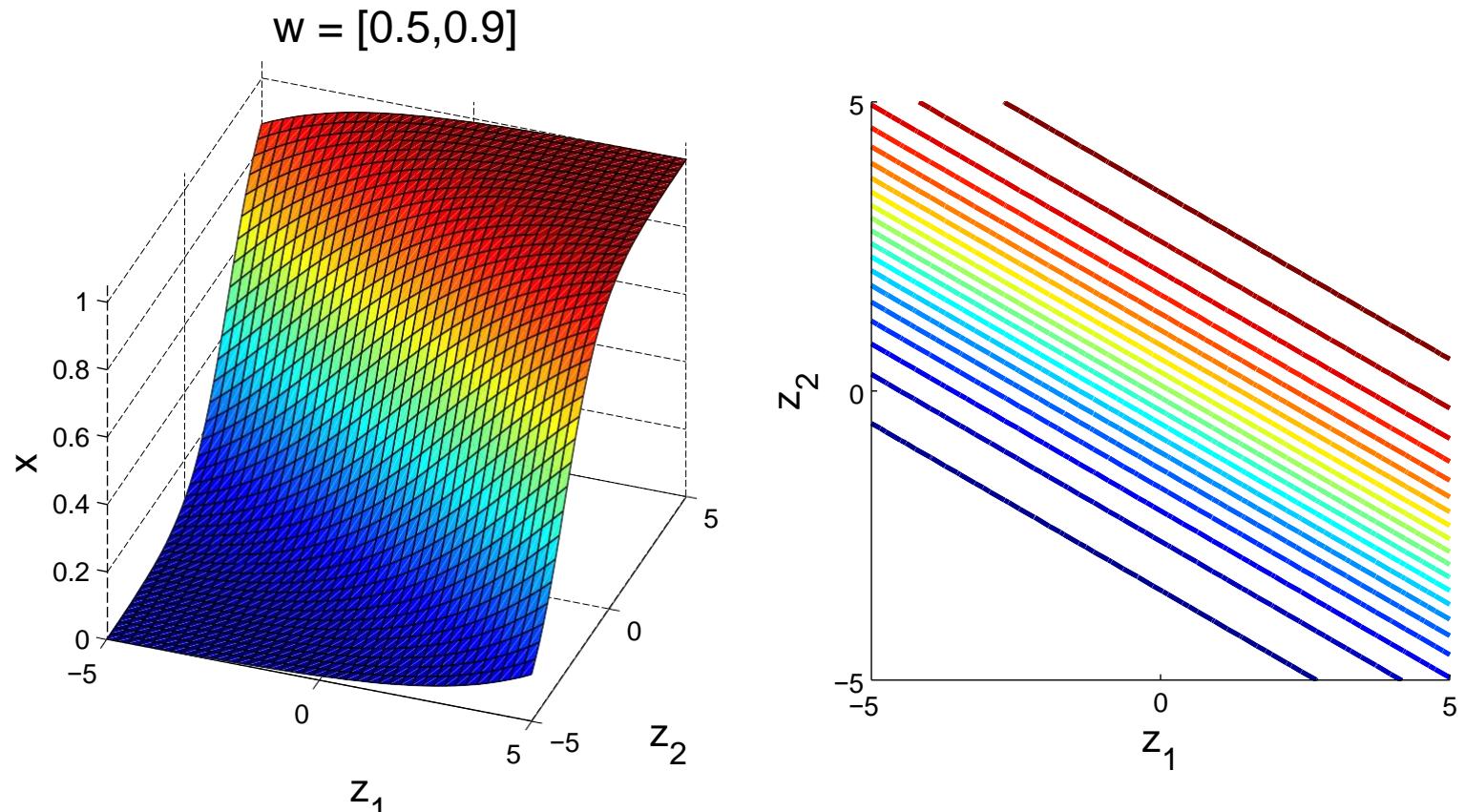
$$x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)}$$

# Input-Output Function of a Single Neuron



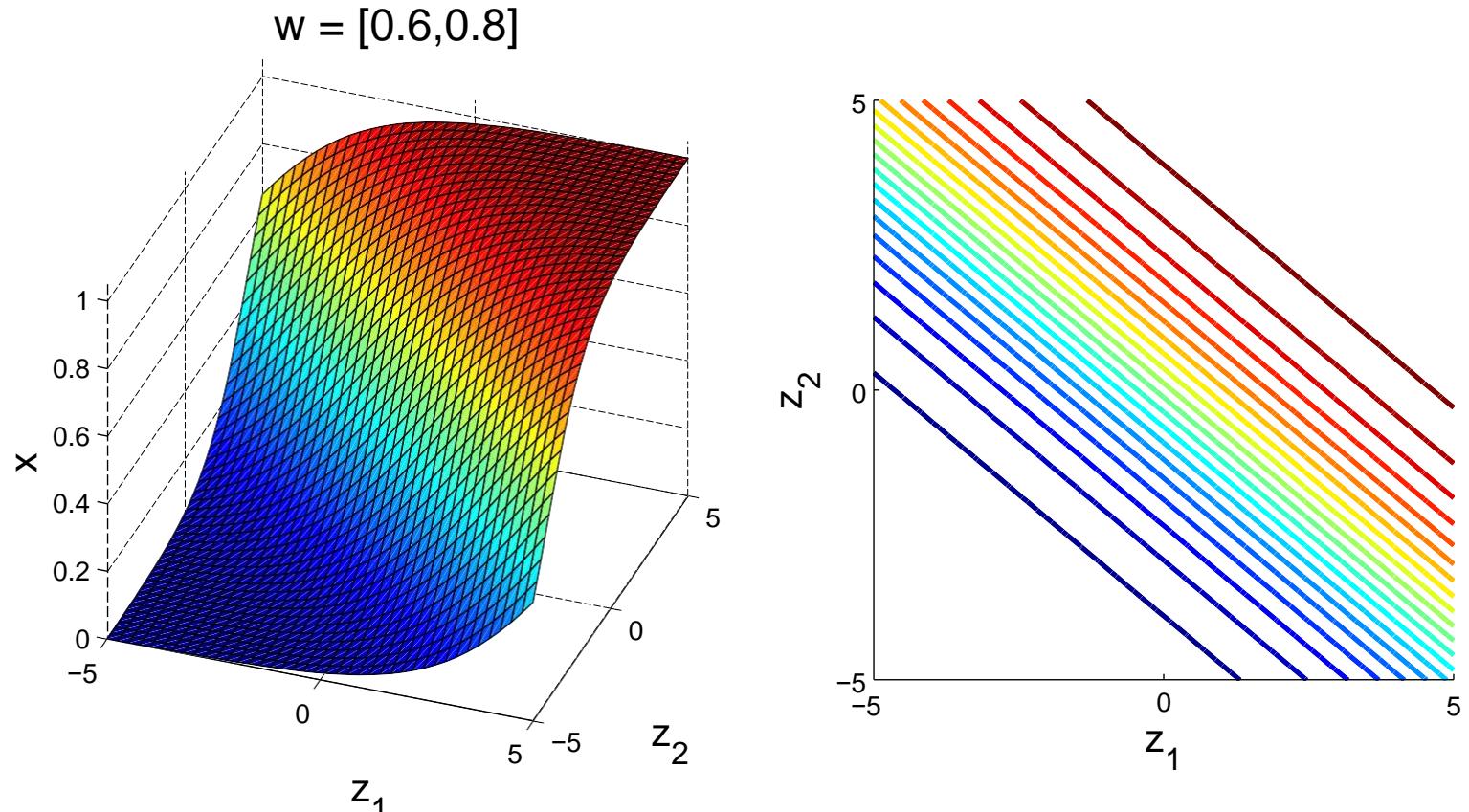
$$x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)}$$

# Input-Output Function of a Single Neuron



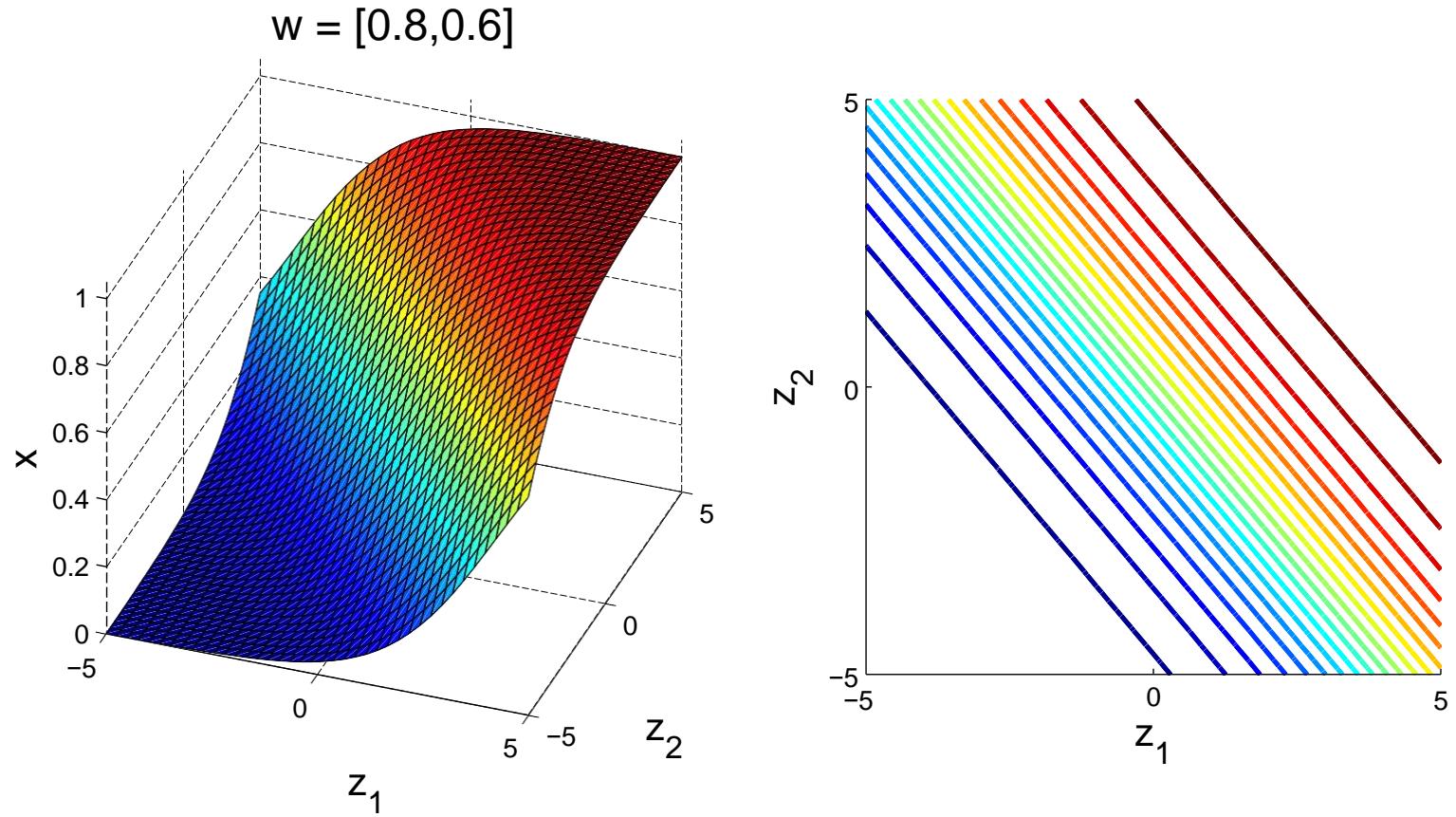
$$x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)}$$

# Input-Output Function of a Single Neuron



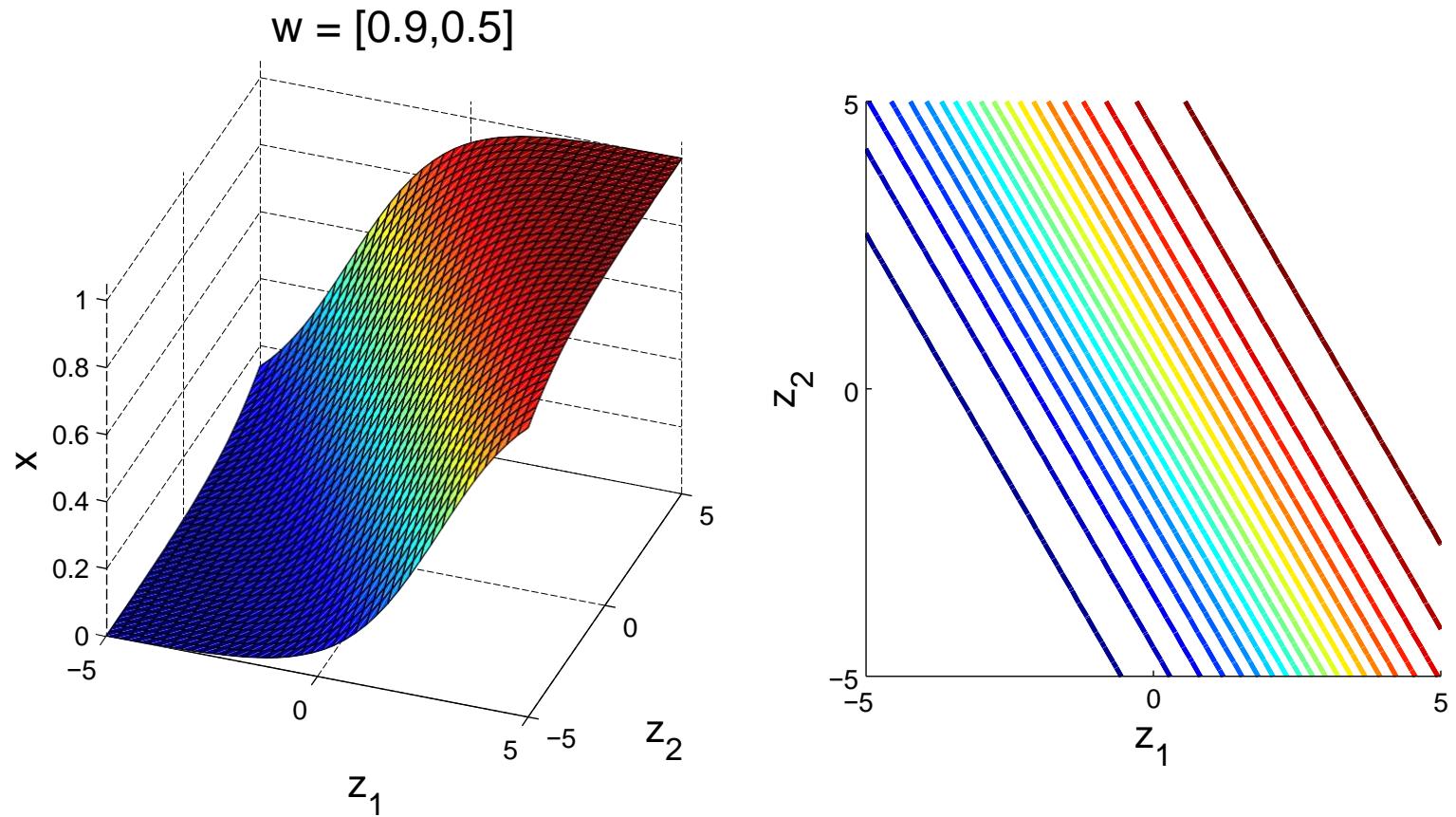
$$x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)}$$

# Input-Output Function of a Single Neuron



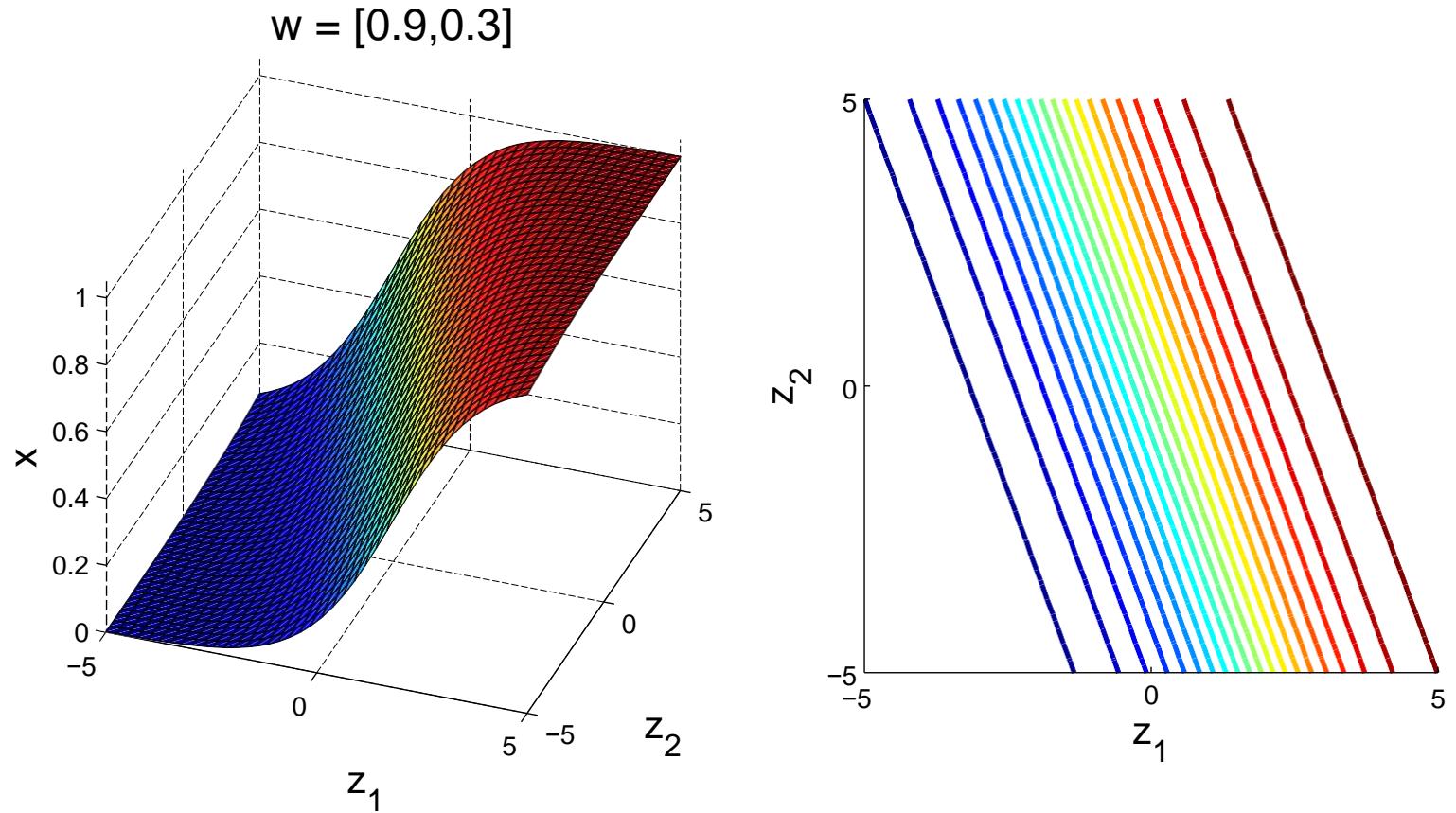
$$x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)}$$

# Input-Output Function of a Single Neuron



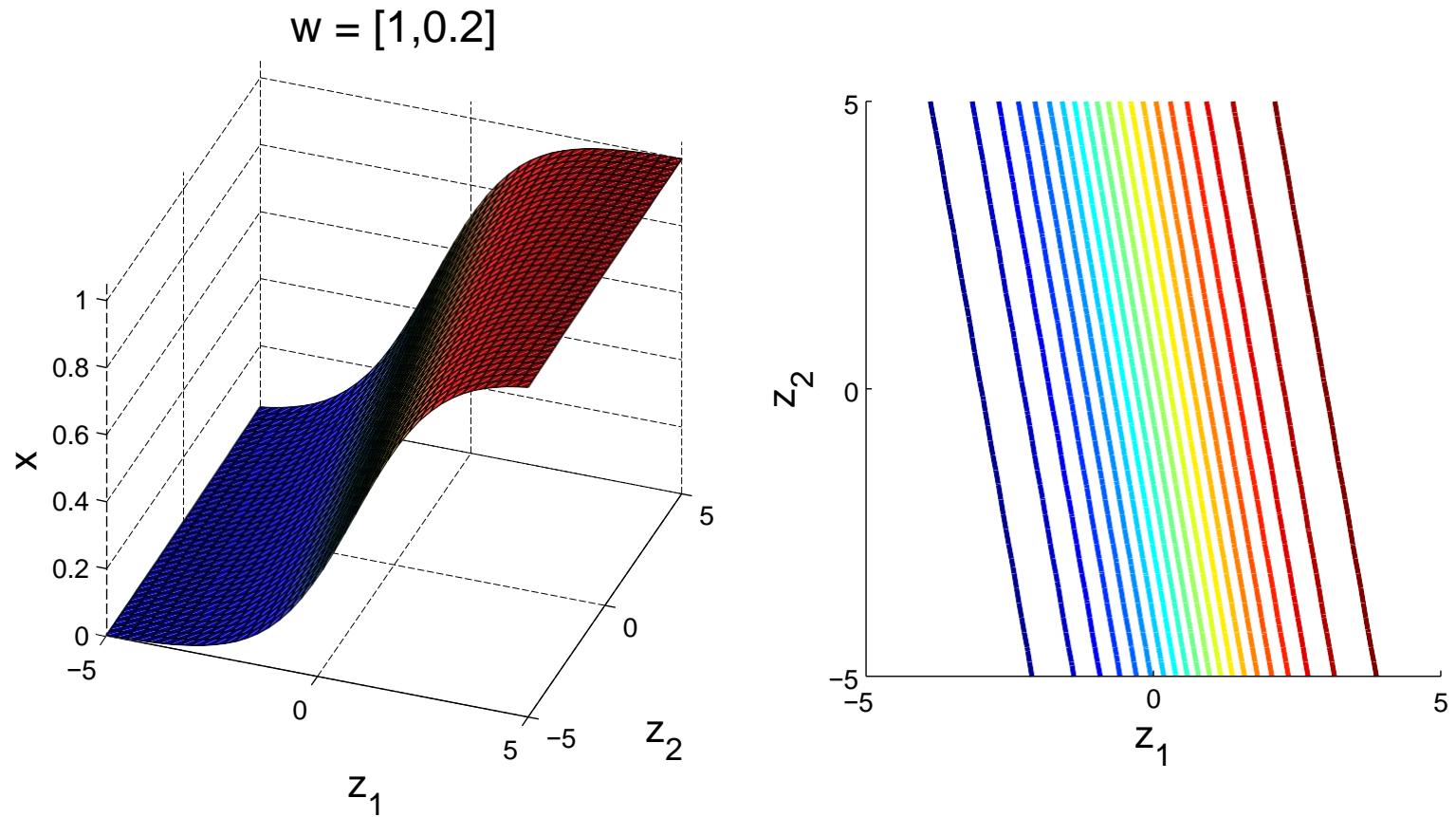
$$x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)}$$

# Input-Output Function of a Single Neuron



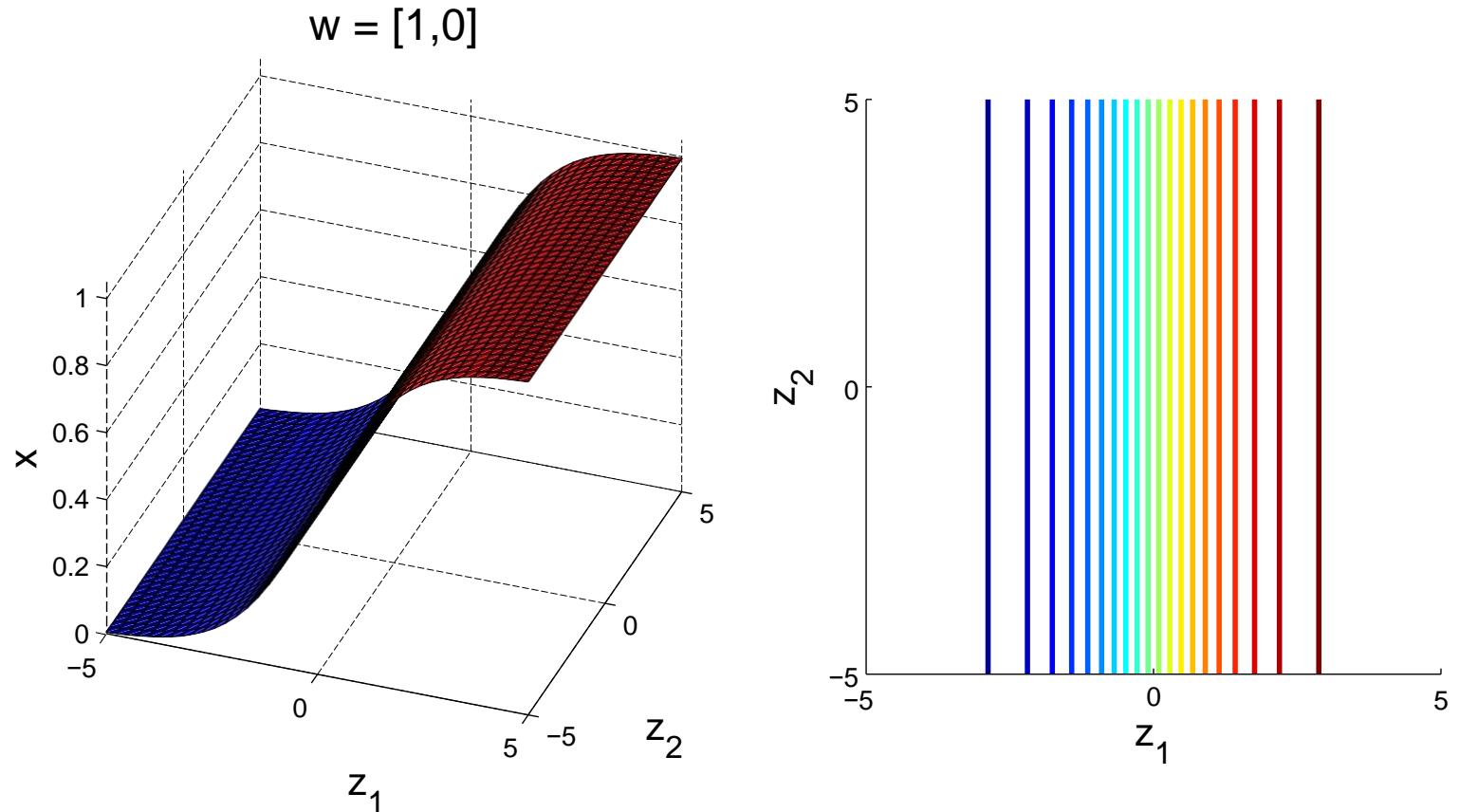
$$x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)}$$

# Input-Output Function of a Single Neuron



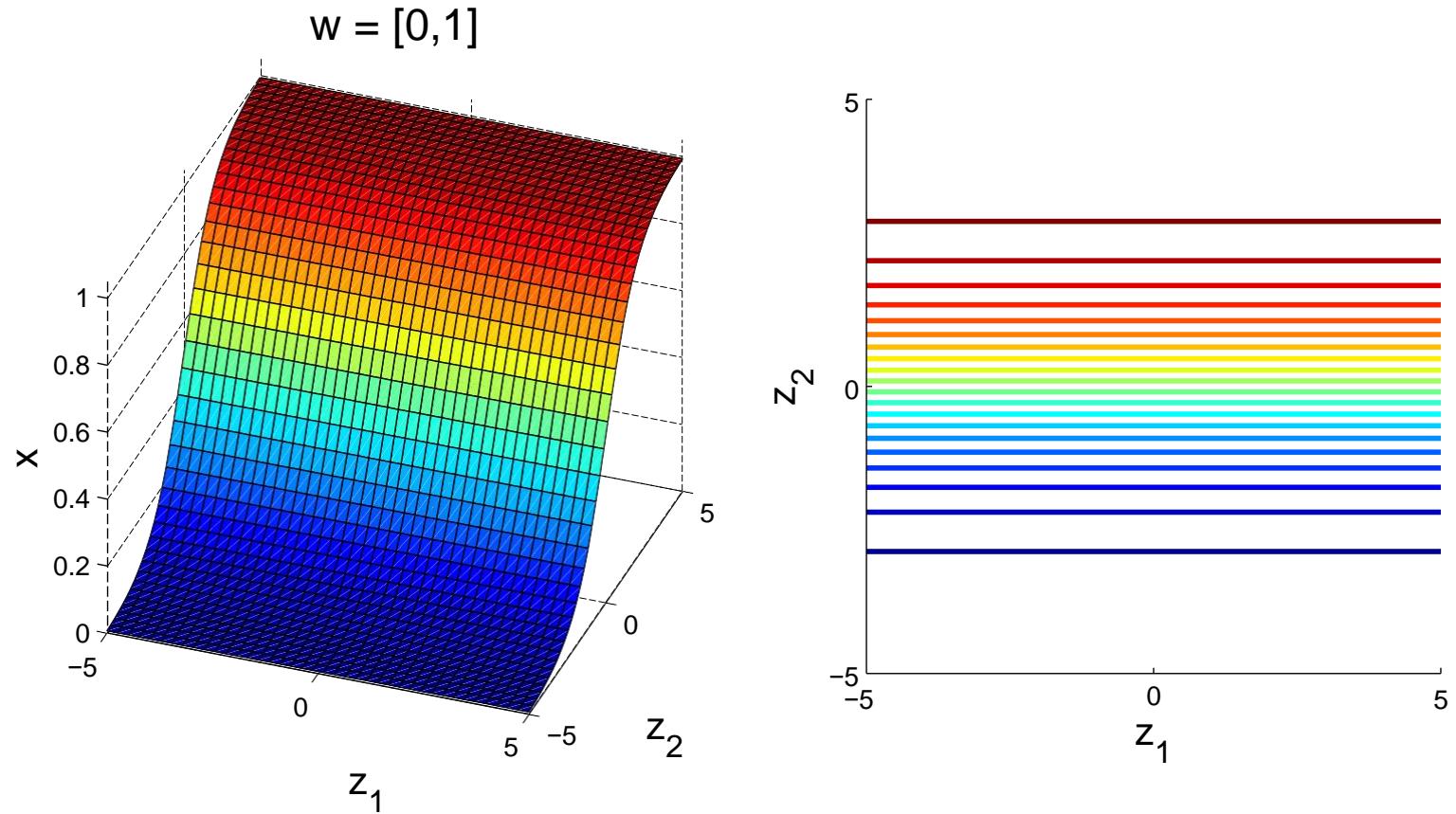
$$x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)}$$

# Input-Output Function of a Single Neuron



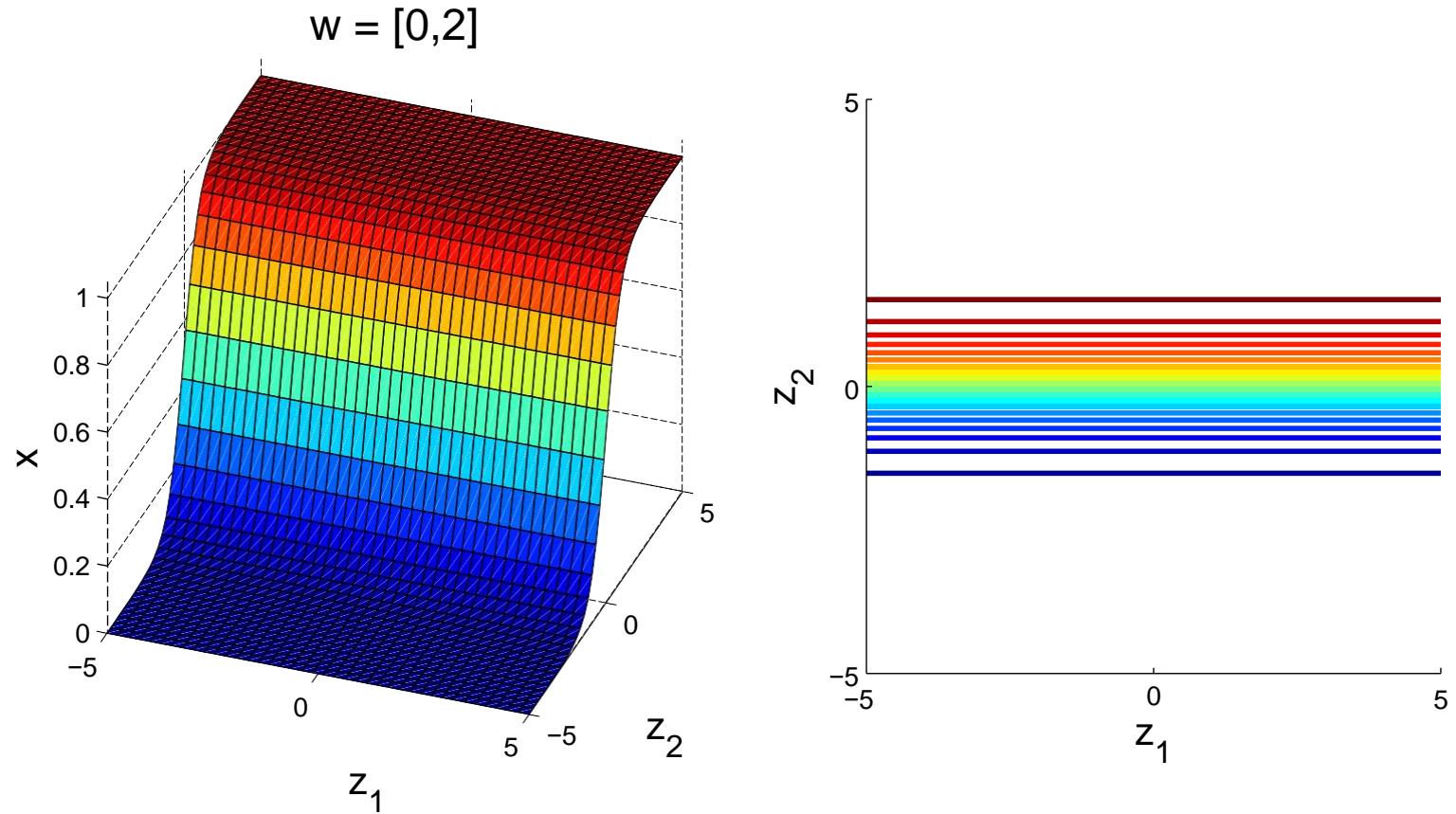
$$x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)}$$

# Input-Output Function of a Single Neuron (cont'd)



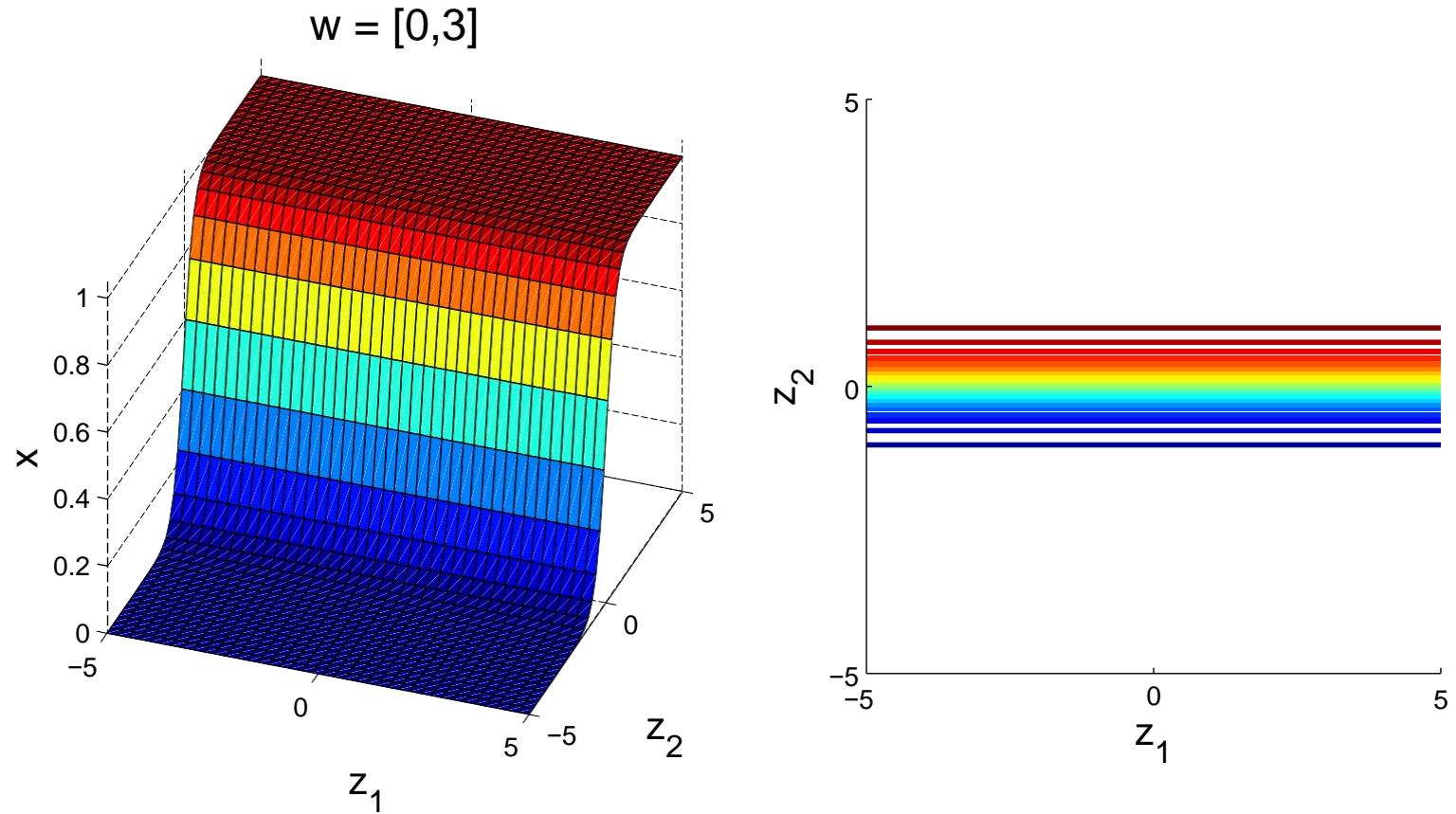
$$x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)}$$

# Input-Output Function of a Single Neuron (cont'd)



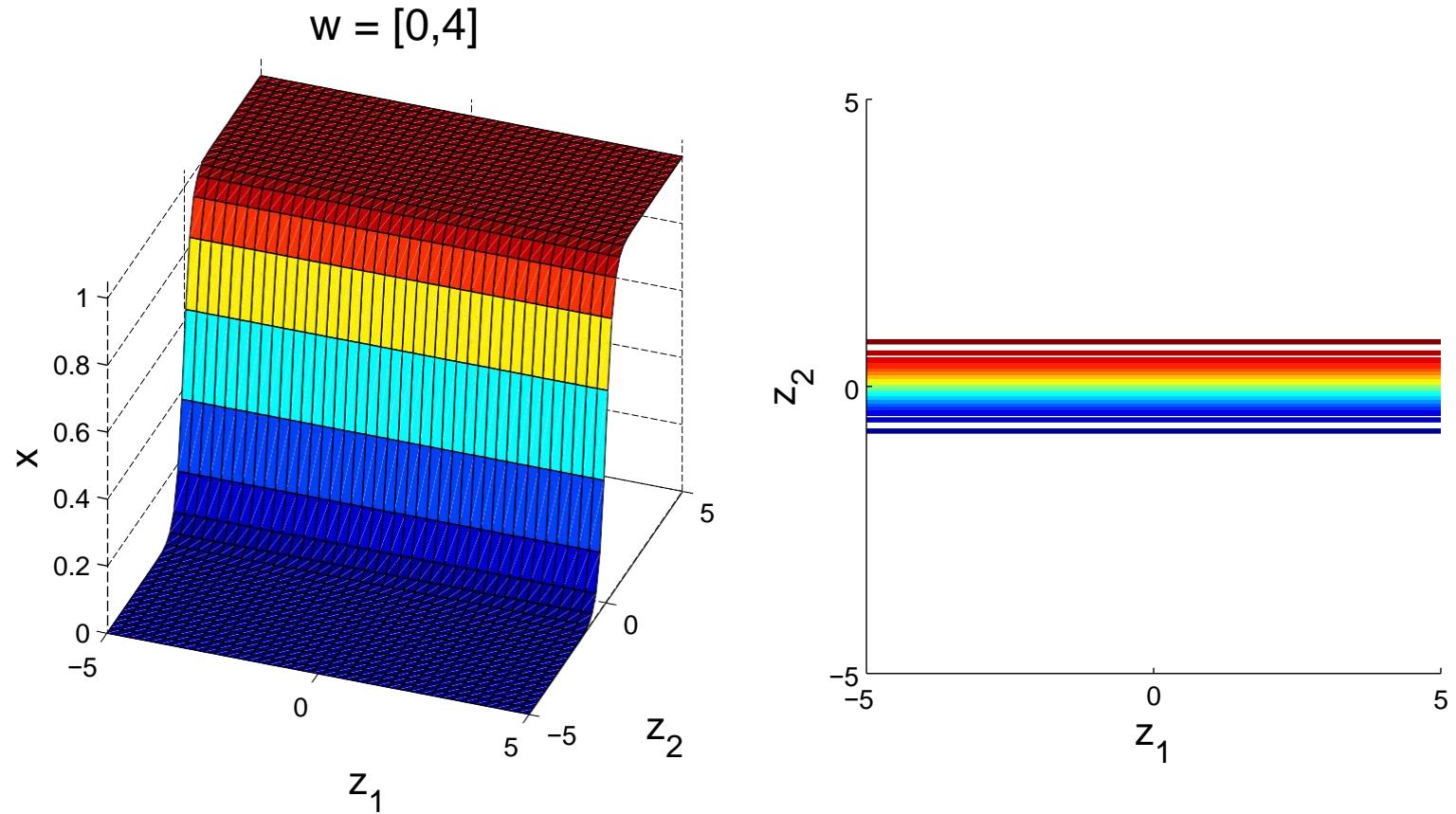
$$x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)}$$

# Input-Output Function of a Single Neuron (cont'd)



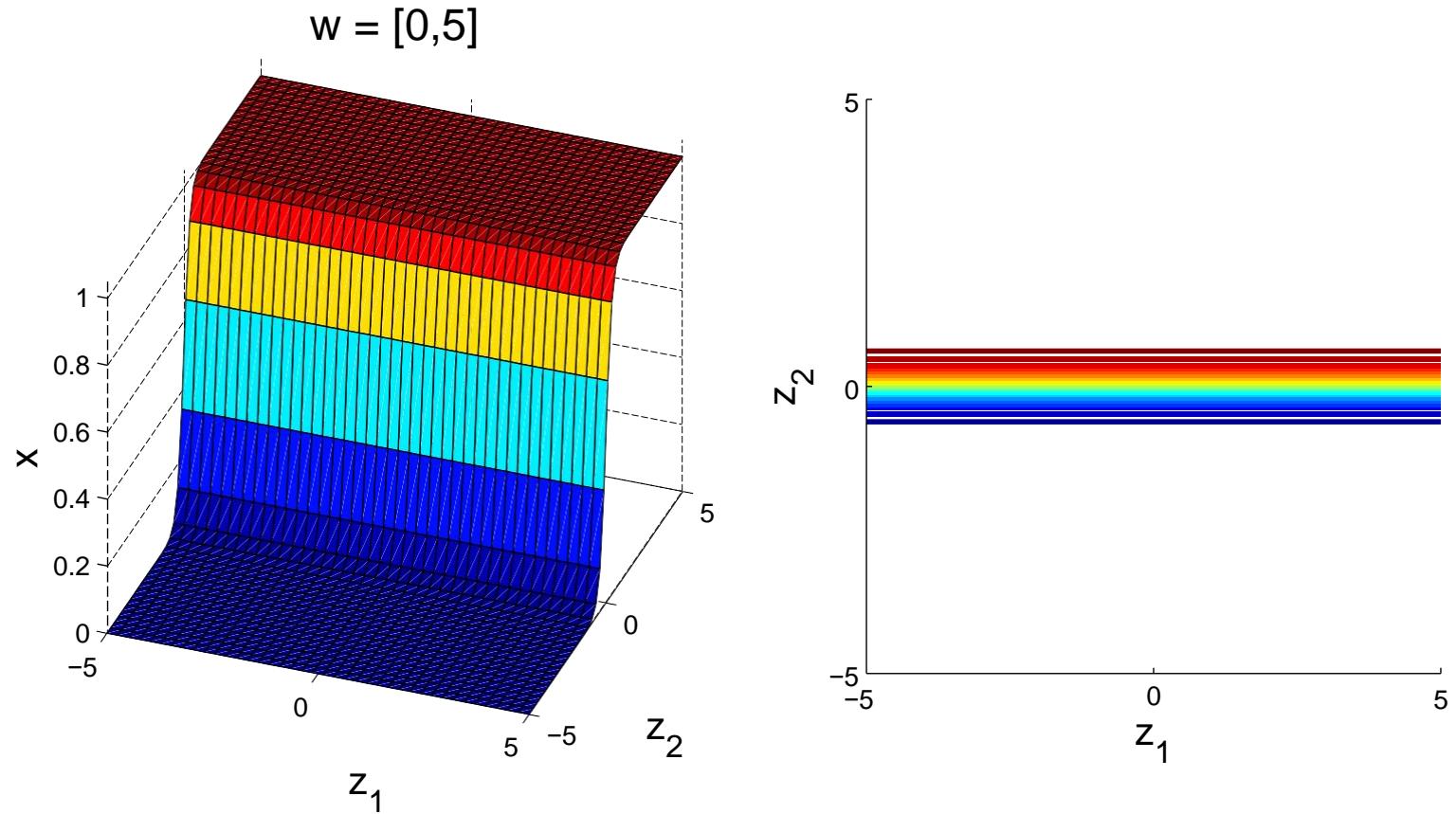
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

# Input-Output Function of a Single Neuron (cont'd)



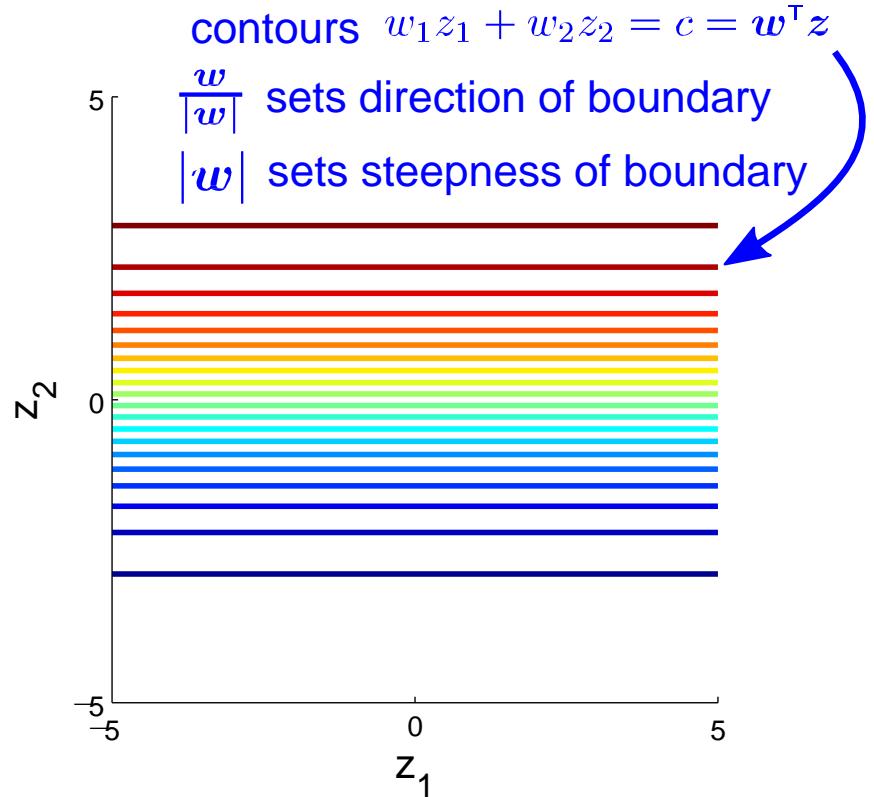
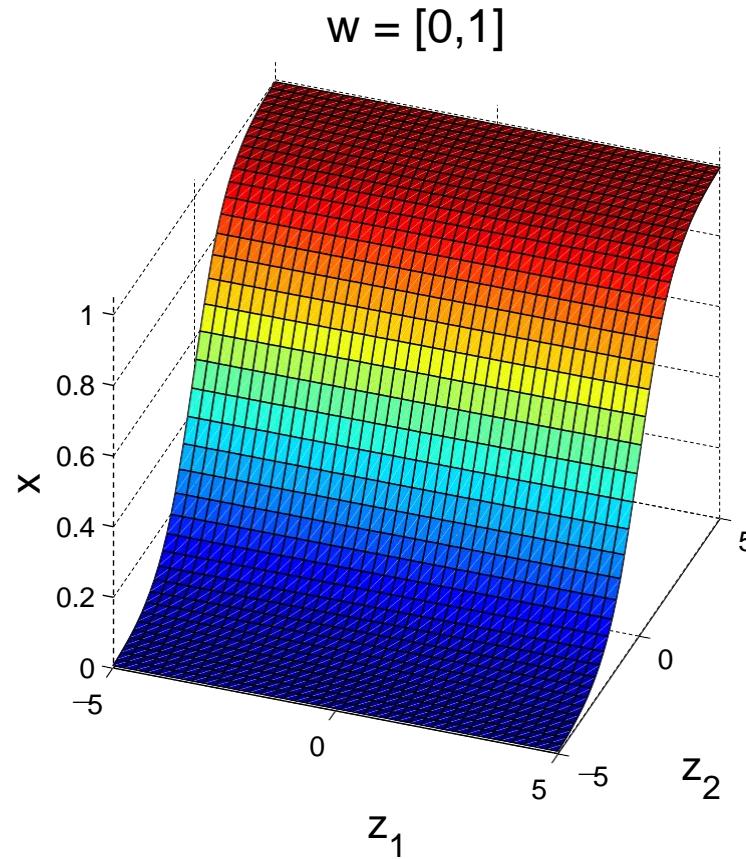
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1z_1-w_2z_2)}$$

# Input-Output Function of a Single Neuron (cont'd)



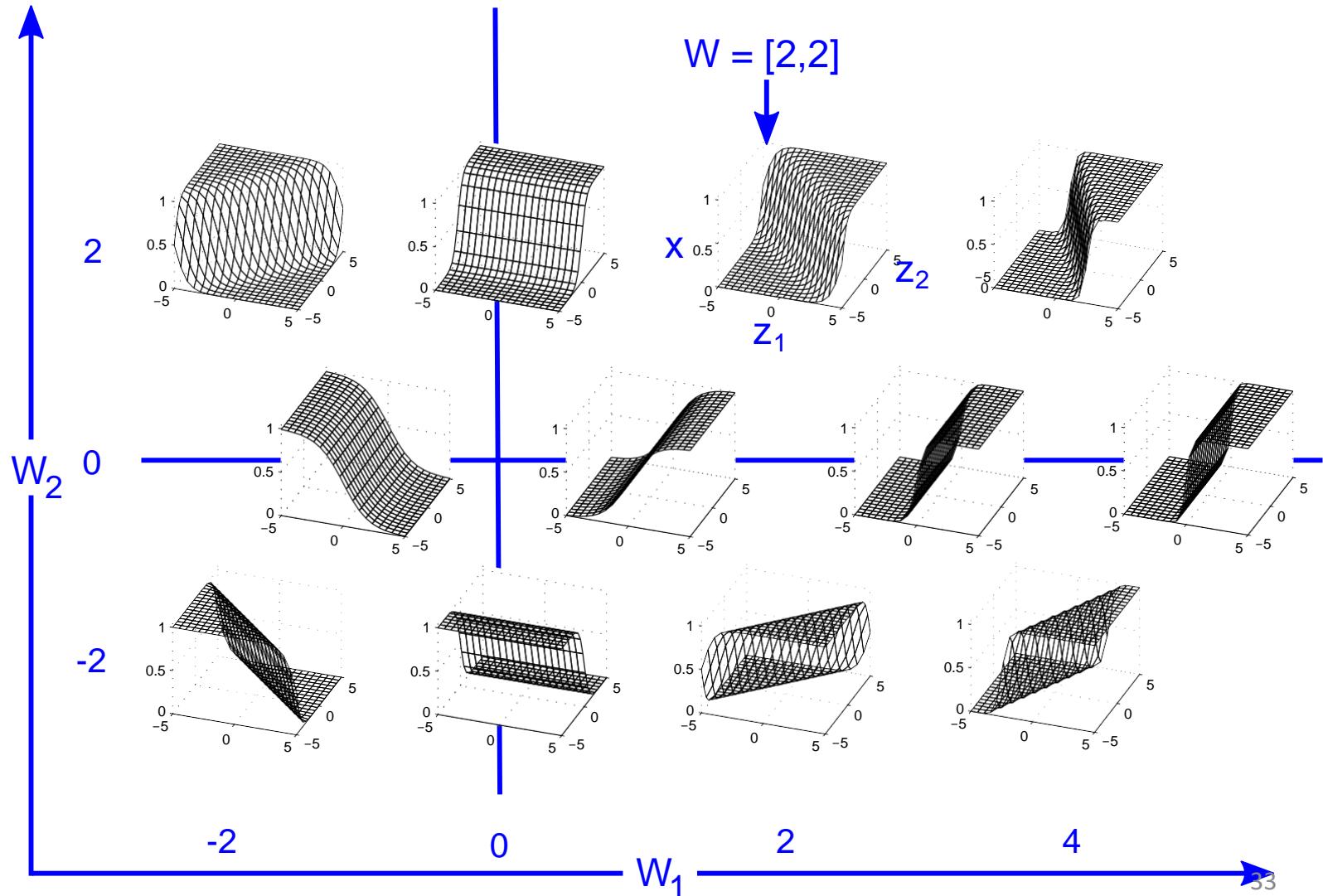
$$x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)}$$

# Input-Output Function of a Single Neuron (cont'd)

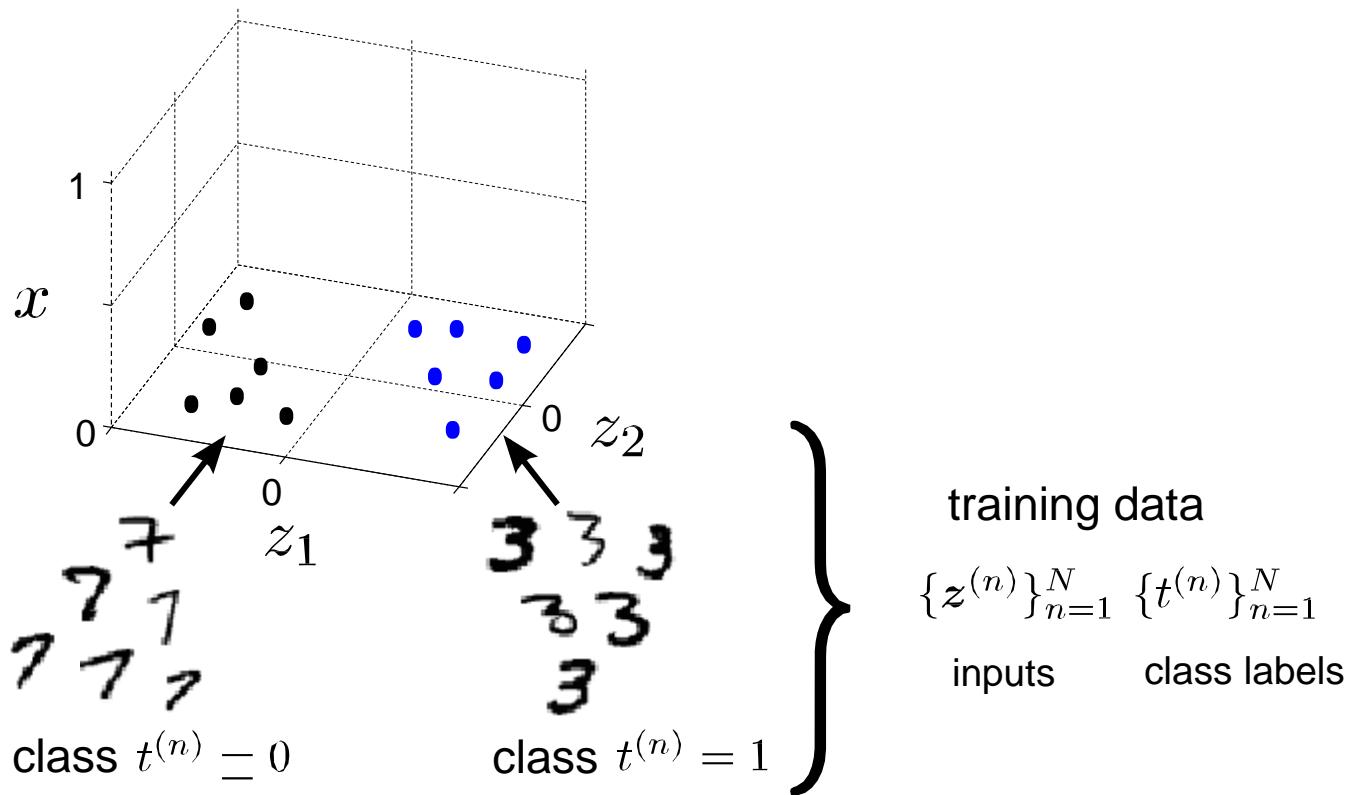


$$x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)}$$

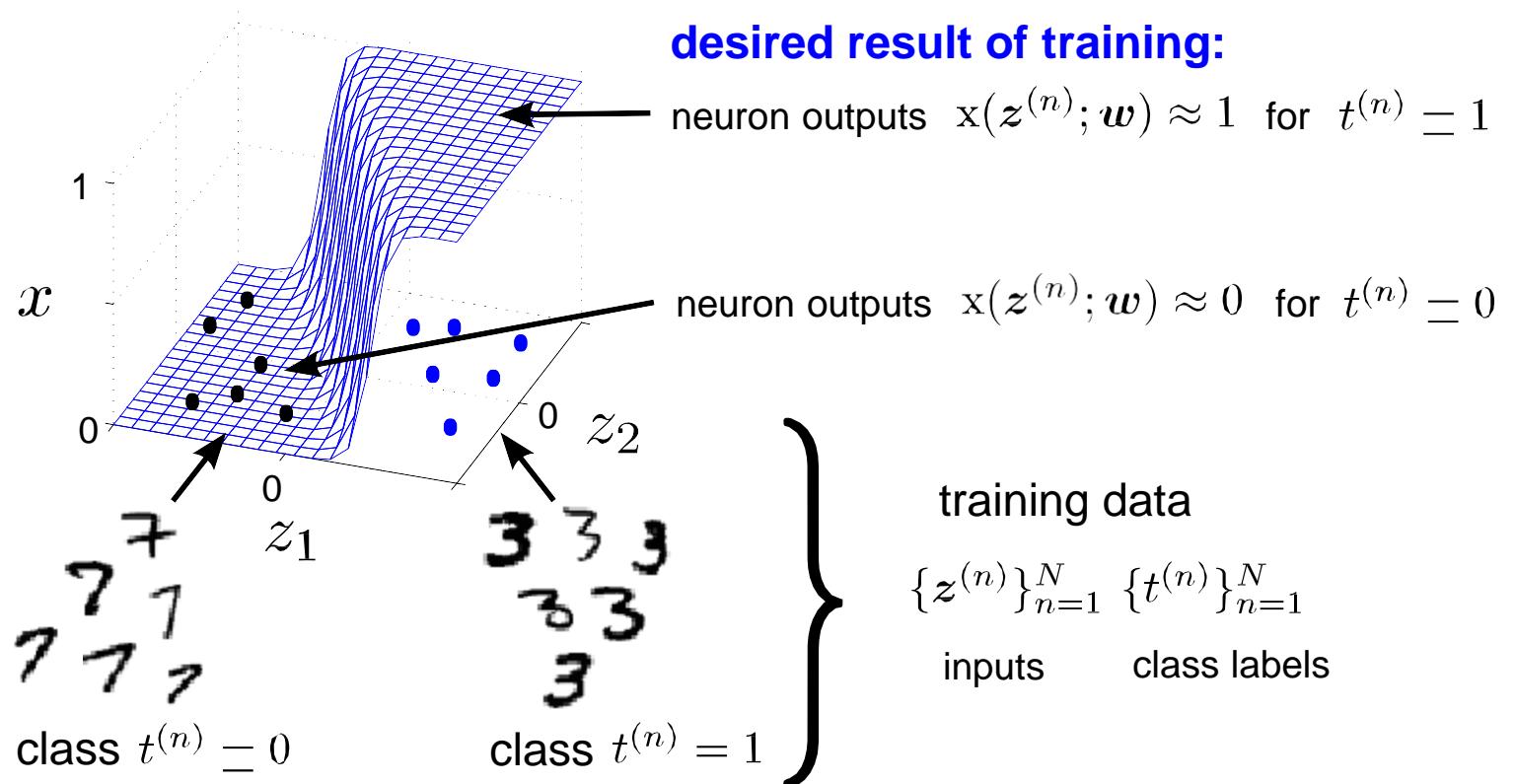
# Weight Space of a Single Neuron



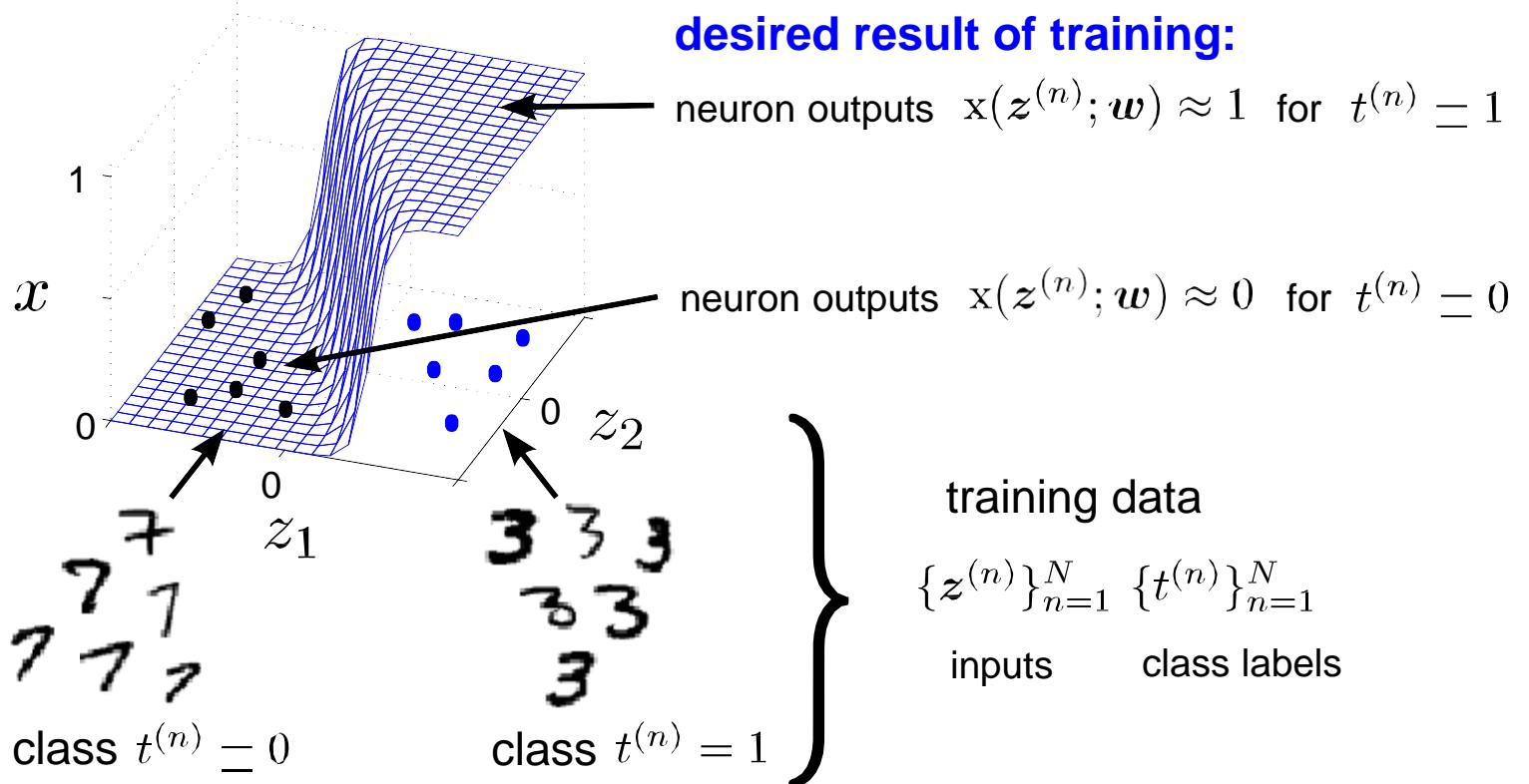
# Training a Single Neuron



# Training a Single Neuron



# Training a Single Neuron



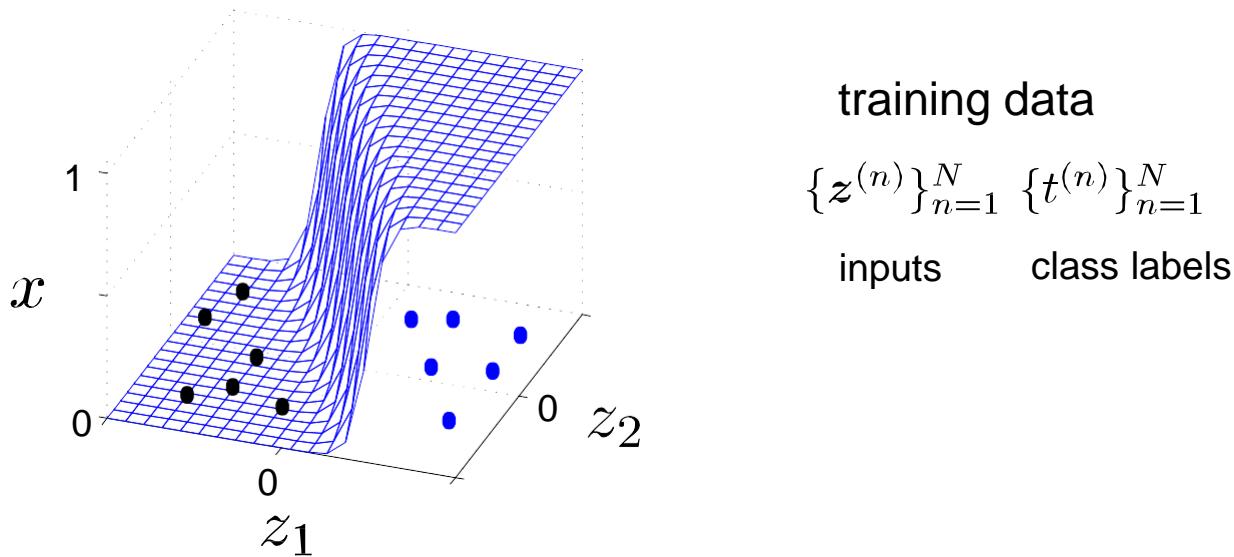
## objective function:

$$G(\mathbf{w}) = - \sum_n [t^{(n)} \log x(z^{(n)}; \mathbf{w}) + (1 - t^{(n)}) \log (1 - x(z^{(n)}; \mathbf{w}))] \geq 0$$

surprise –  $\log p(\text{outcome})$  when observing  $t^{(n)}$   
relative entropy between  $x(z^{(n)}; \mathbf{w})$  and  $t^{(n)}$

} encourages neuron output  
to match training data

# Training a Single Neuron



training data

$$\{z^{(n)}\}_{n=1}^N \quad \{t^{(n)}\}_{n=1}^N$$

inputs

class labels

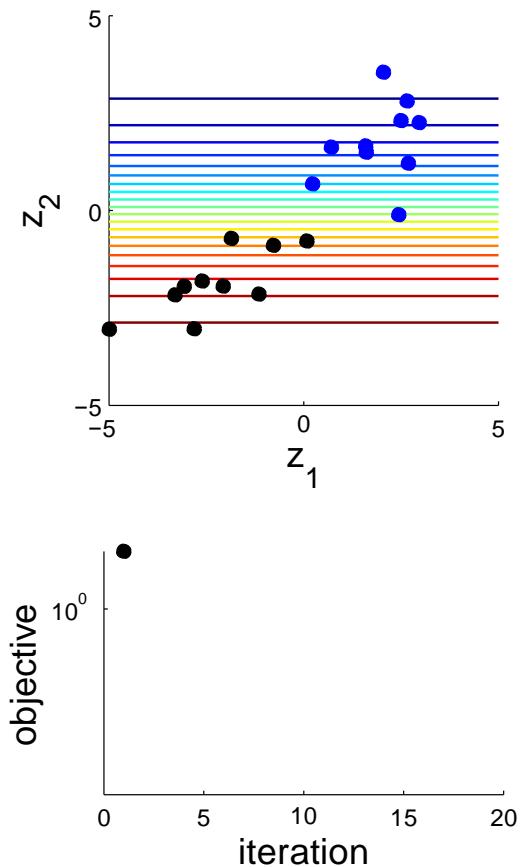
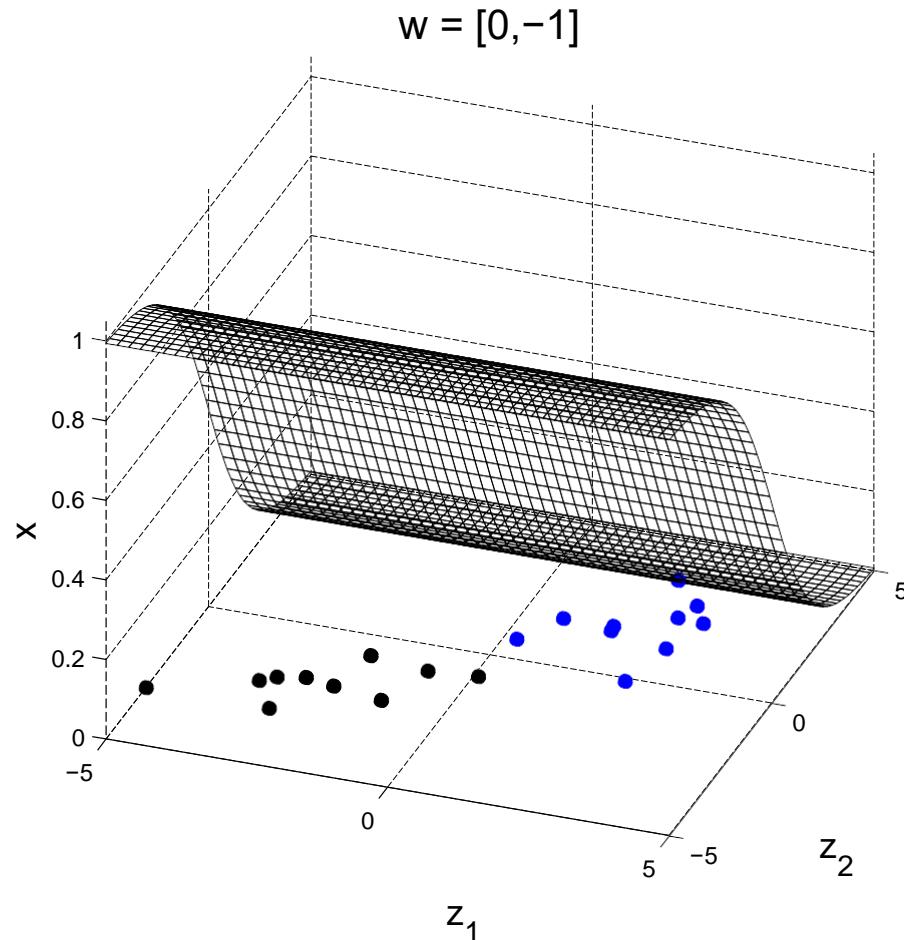
**objective function:**

$$G(\mathbf{w}) = - \sum_n [t^{(n)} \log x(\mathbf{z}^{(n)}; \mathbf{w}) + (1 - t^{(n)}) \log (1 - x(\mathbf{z}^{(n)}; \mathbf{w}))] \geq 0$$

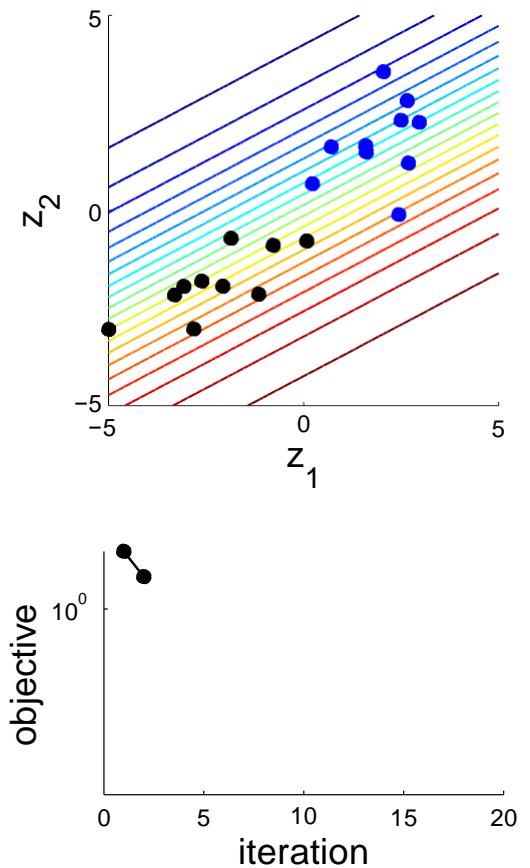
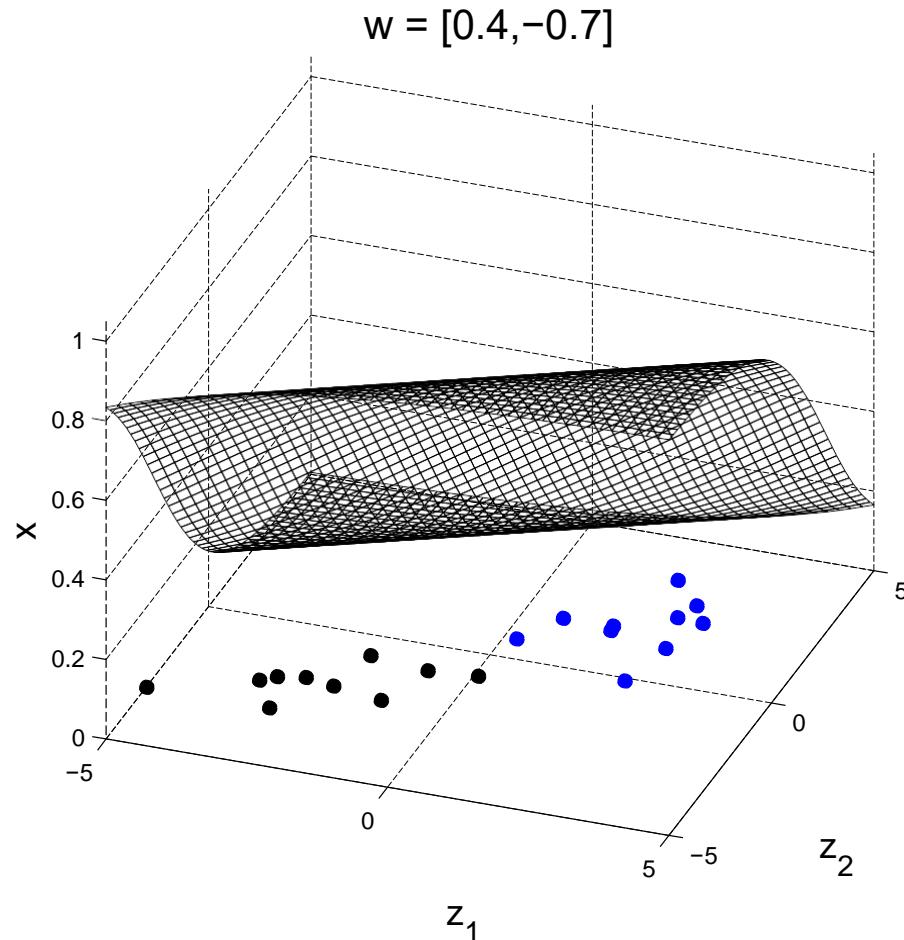
$$\mathbf{w}^* = \arg \min_{\mathbf{w}} G(\mathbf{w}) \quad \text{choose the weights that minimise the network's surprise about the training data}$$

$$\frac{d}{d\mathbf{w}} G(\mathbf{w}) = \sum_n \frac{dG(\mathbf{w})}{dx^{(n)}} \frac{dx^{(n)}}{d\mathbf{w}} = - \sum_n (t^{(n)} - x^{(n)}) \mathbf{z}^{(n)} = \text{prediction error} \times \text{feature}$$
$$\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{d}{d\mathbf{w}} G(\mathbf{w}) \quad \text{iteratively step down the objective (gradient points up hill)}$$

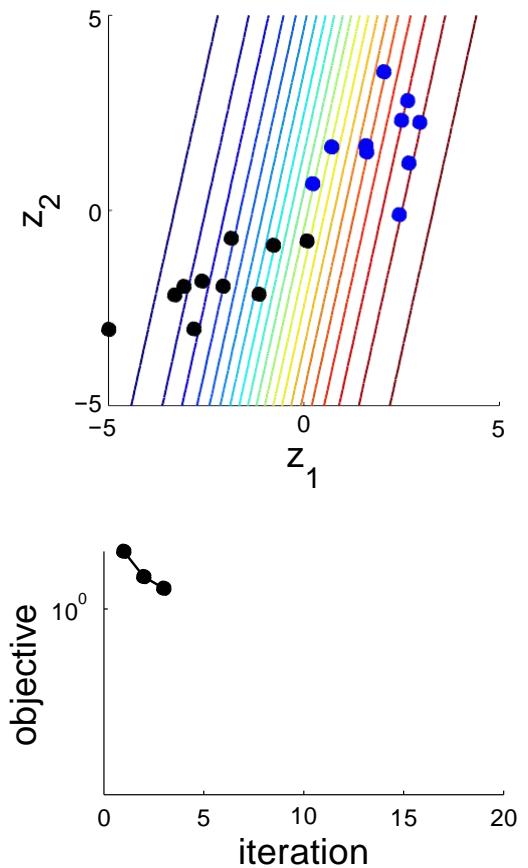
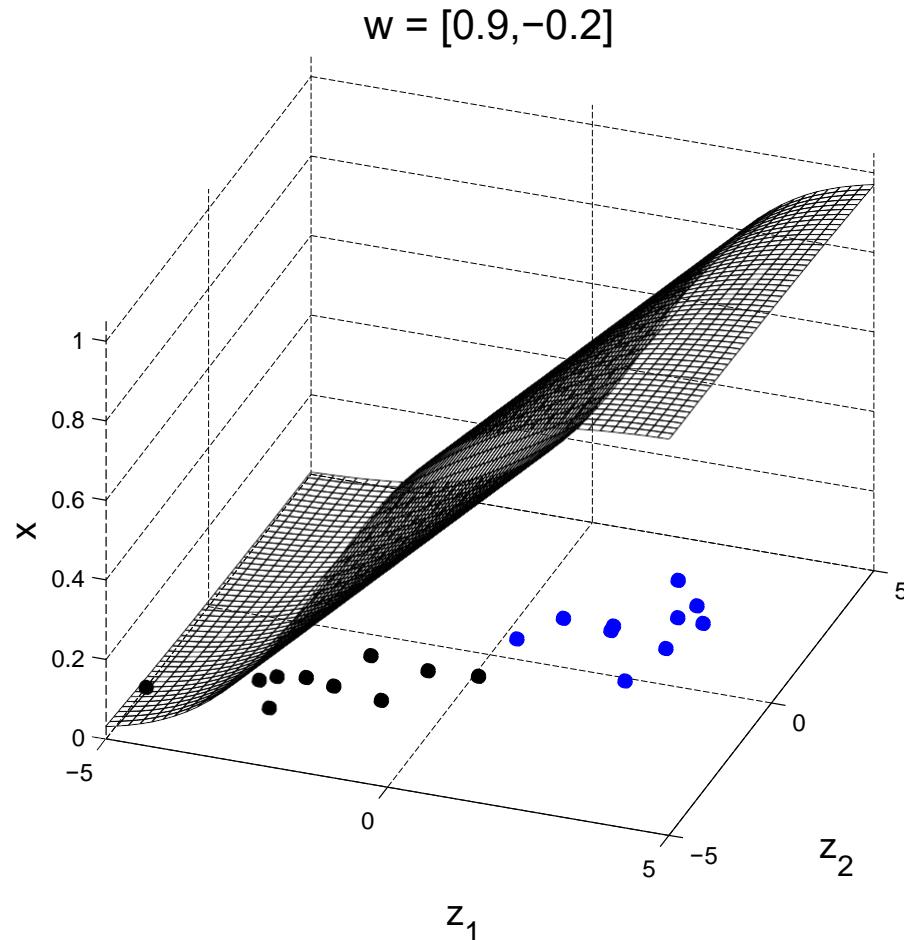
# Training a Single Neuron



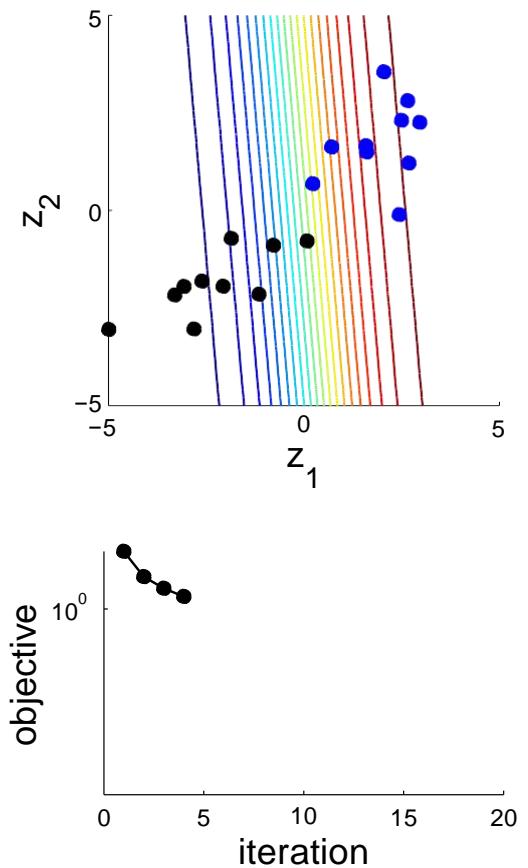
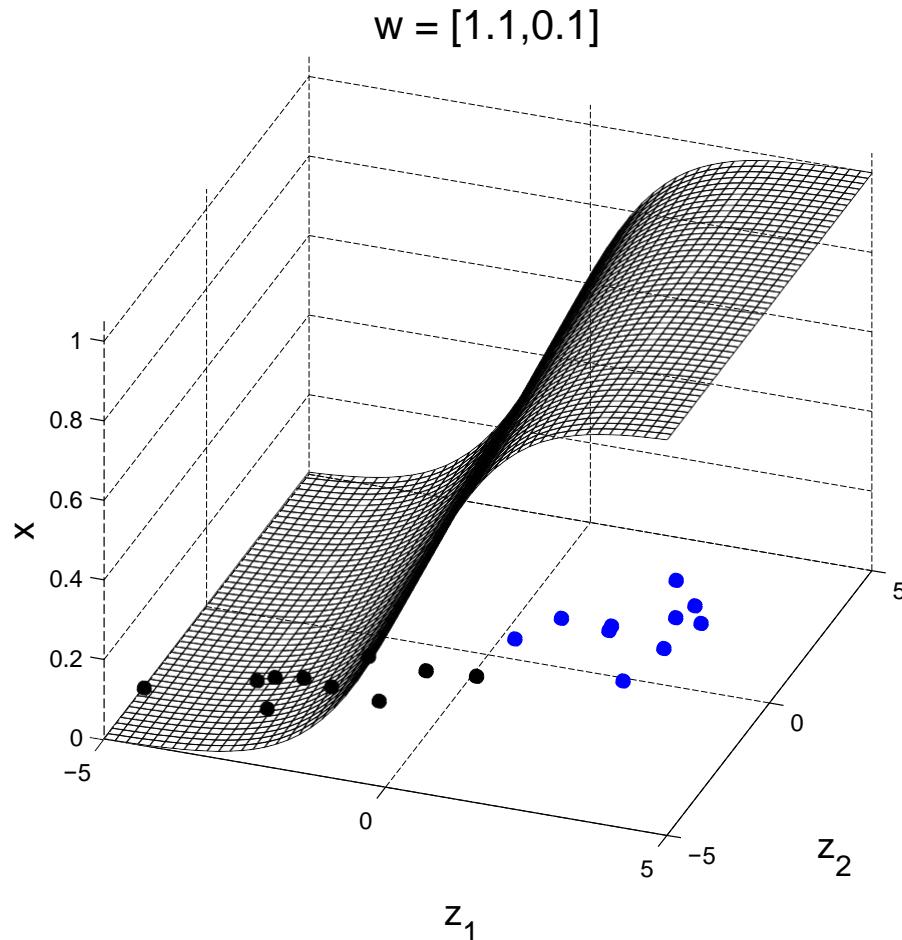
# Training a Single Neuron



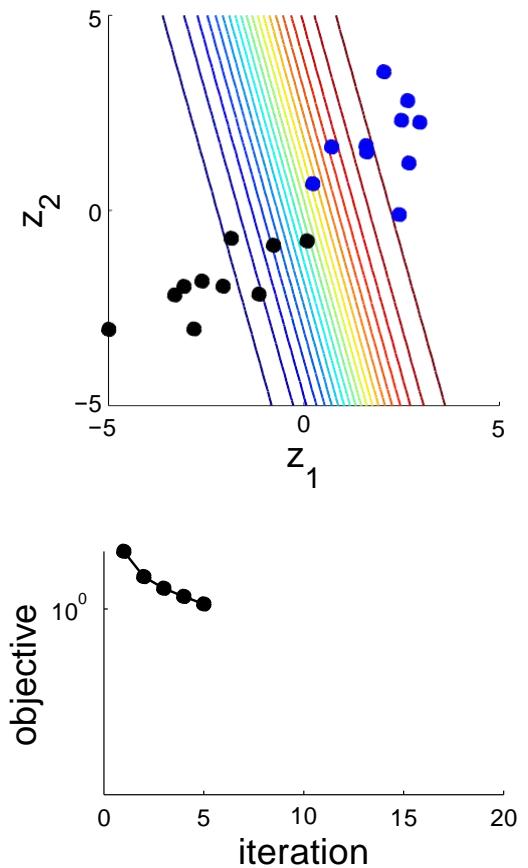
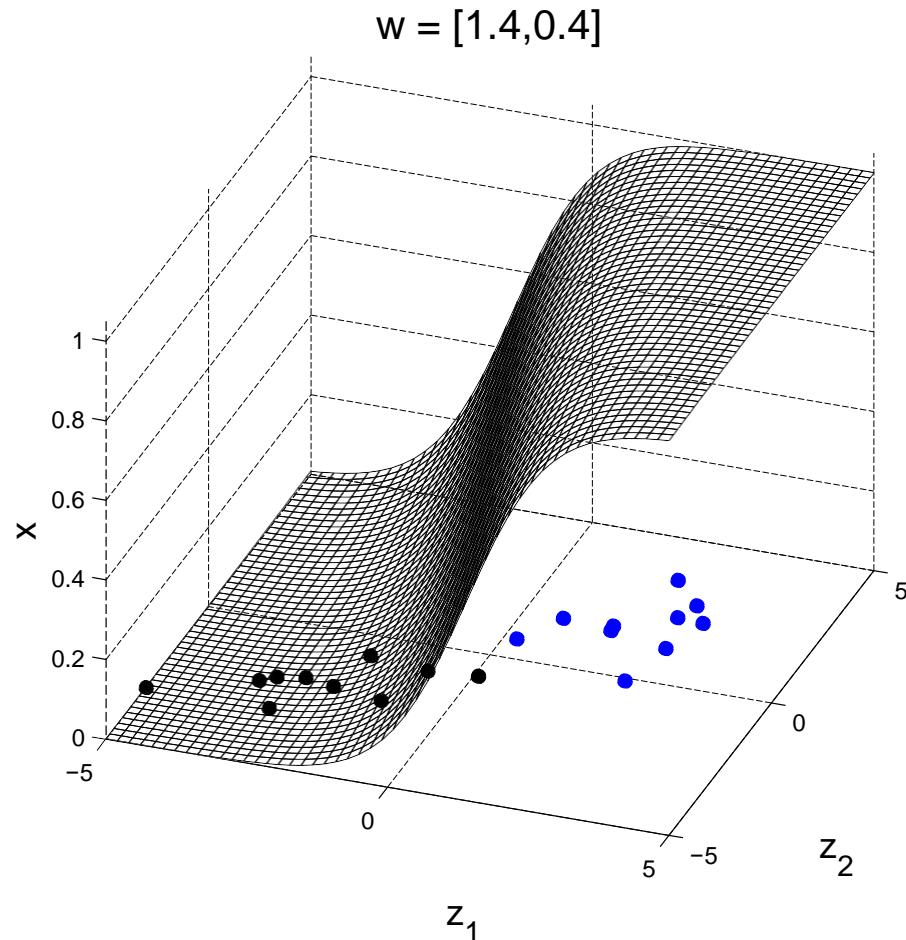
# Training a Single Neuron



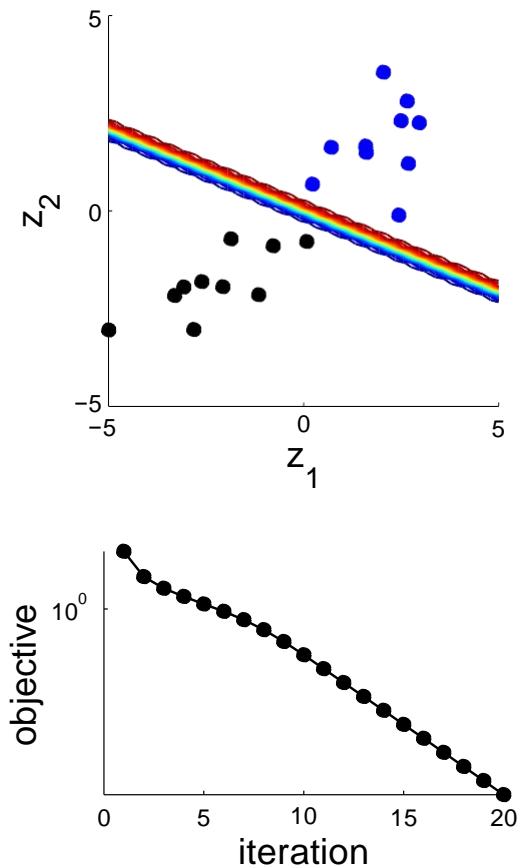
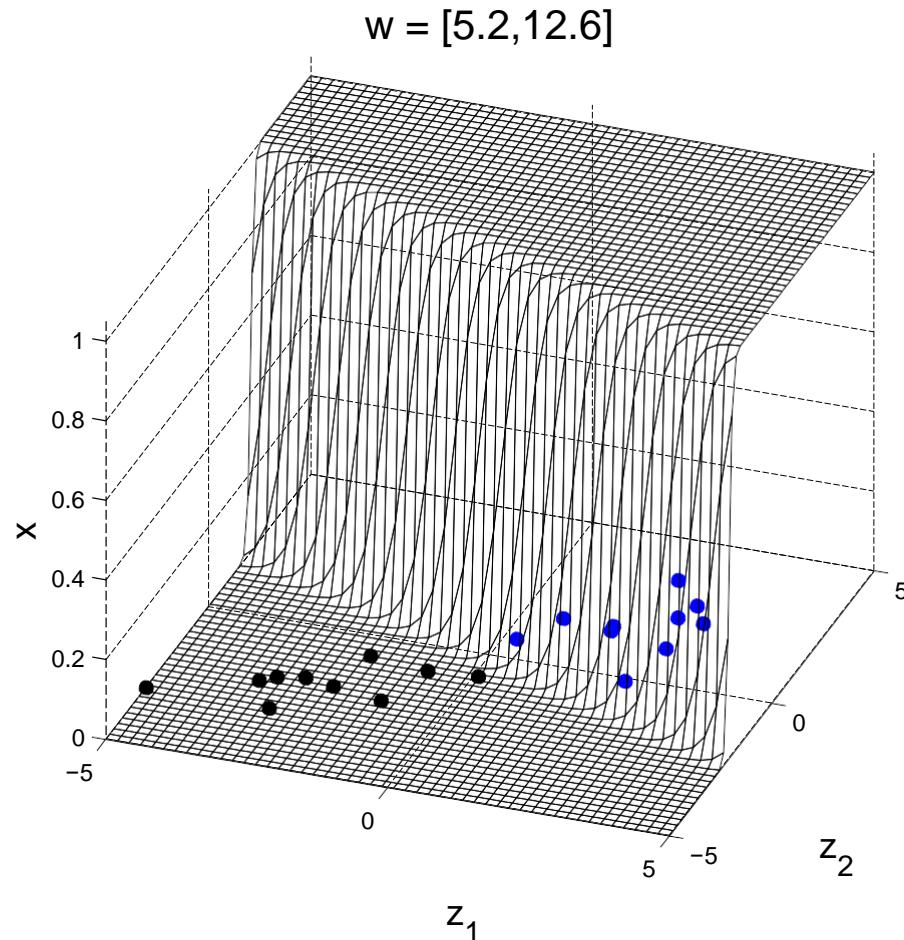
# Training a Single Neuron



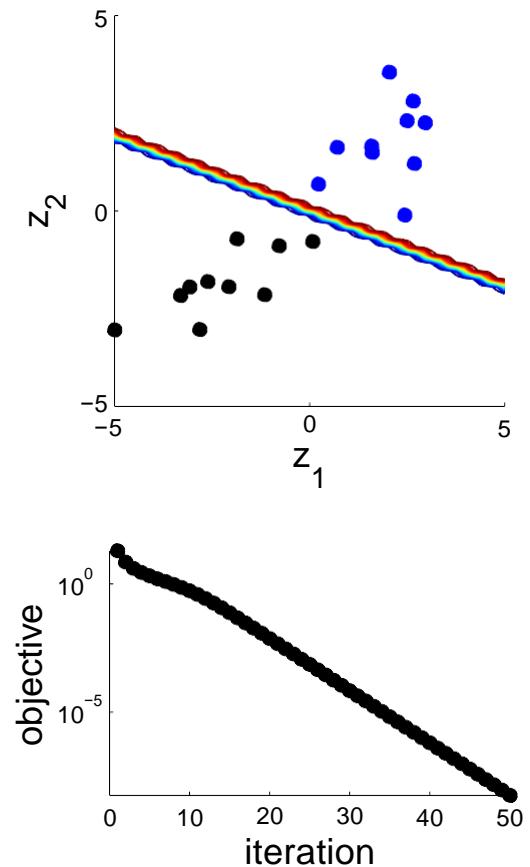
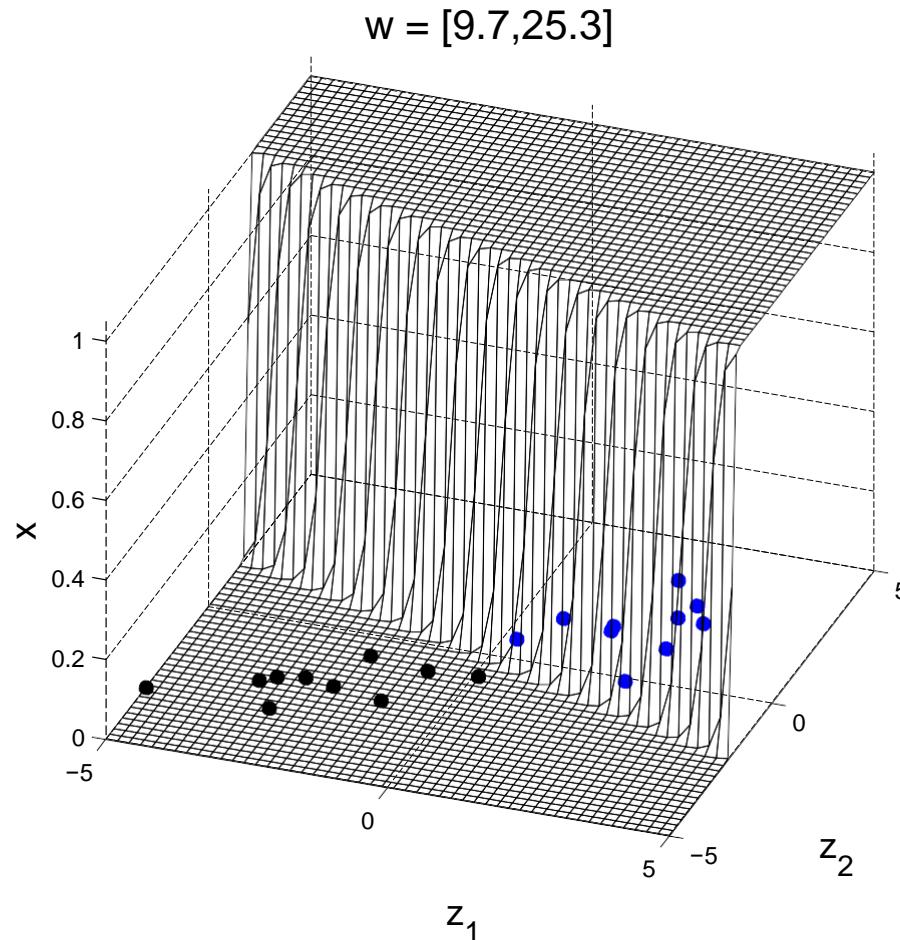
# Training a Single Neuron



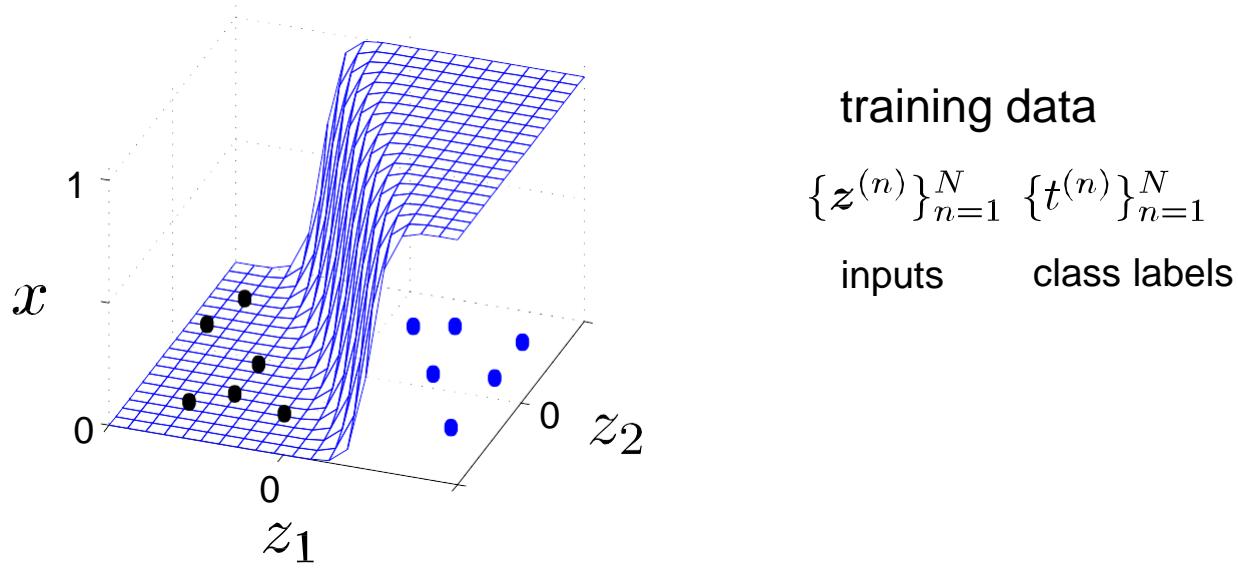
# Training a Single Neuron



# Training a Single Neuron



# Overfitting and Weight Decay



training data

$$\{\mathbf{z}^{(n)}\}_{n=1}^N \quad \{t^{(n)}\}_{n=1}^N$$

inputs

class labels

**objective function:**

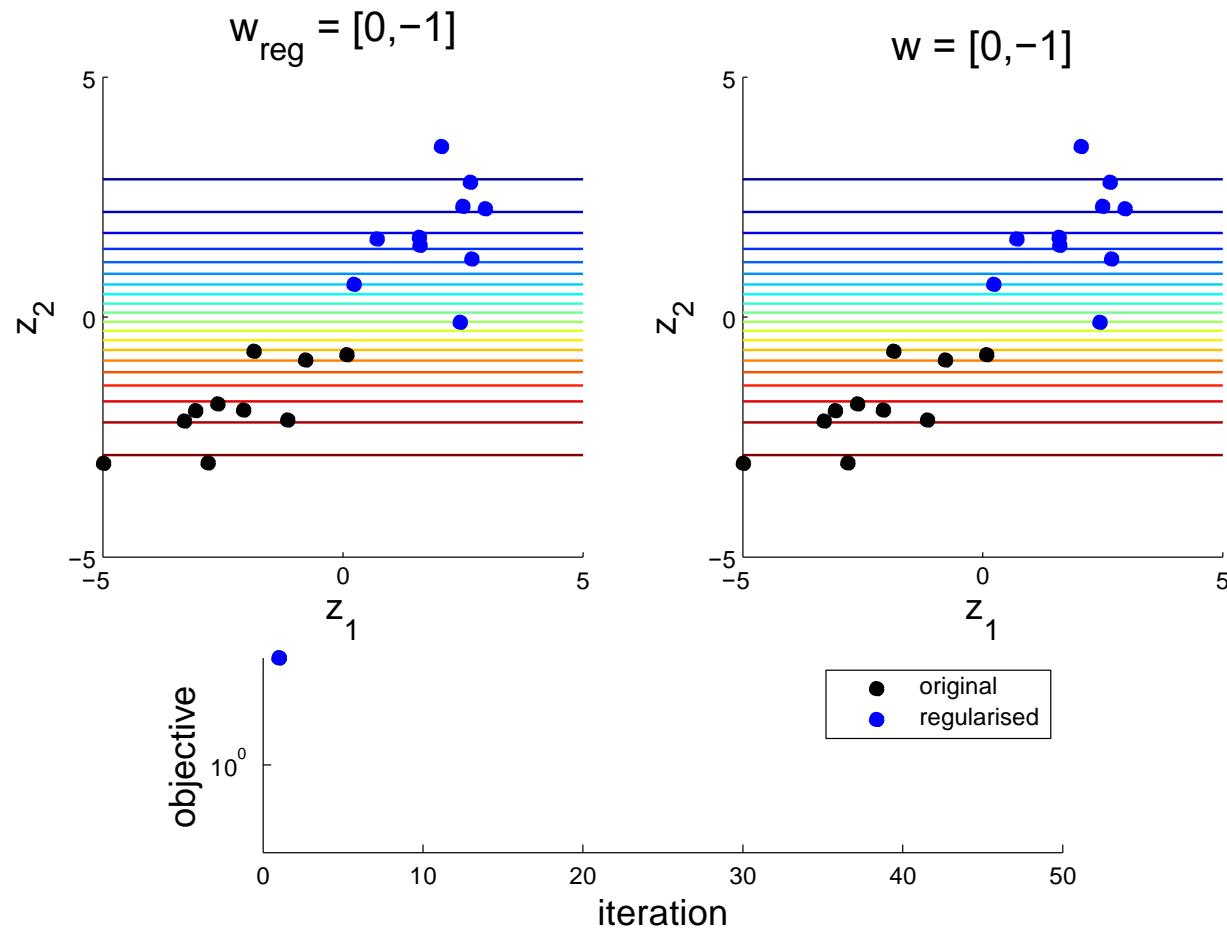
$$G(\mathbf{w}) = - \sum_n [t^{(n)} \log x(\mathbf{z}^{(n)}; \mathbf{w}) + (1 - t^{(n)}) \log (1 - x(\mathbf{z}^{(n)}; \mathbf{w}))]$$

$$E(\mathbf{w}) = \frac{1}{2} \sum_i w_i^2 \quad \text{regulariser discourages the network using extreme weights}$$

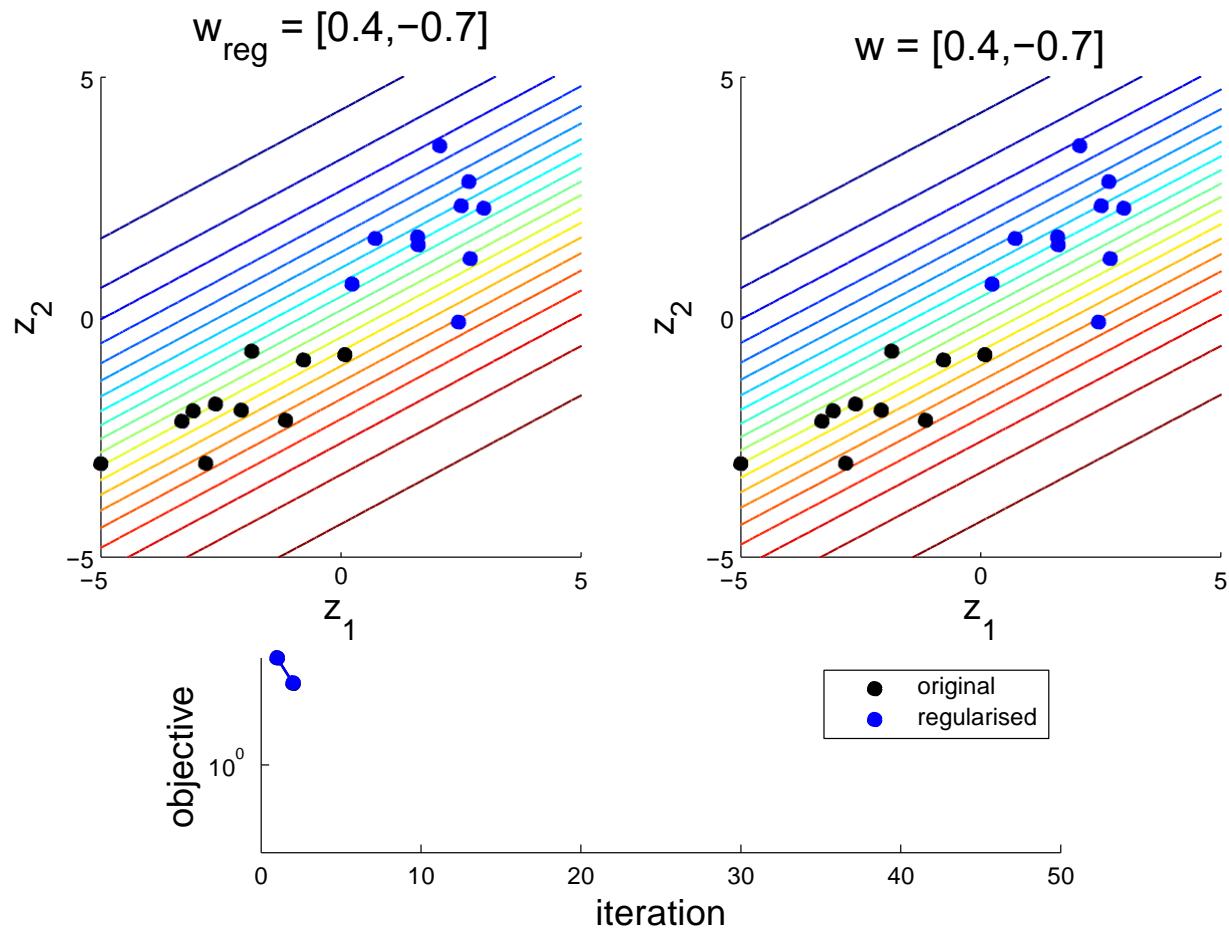
$$\mathbf{w}^* = \arg \min_{\mathbf{w}} M(\mathbf{w}) = \arg \min_{\mathbf{w}} [G(\mathbf{w}) + \alpha E(\mathbf{w})]$$

$$\frac{d}{d\mathbf{w}} M(\mathbf{w}) = - \sum_n (t^{(n)} - x^{(n)}) \mathbf{z}^{(n)} + \alpha \mathbf{w} \quad \text{weight decay - shrinks weights towards zero}$$

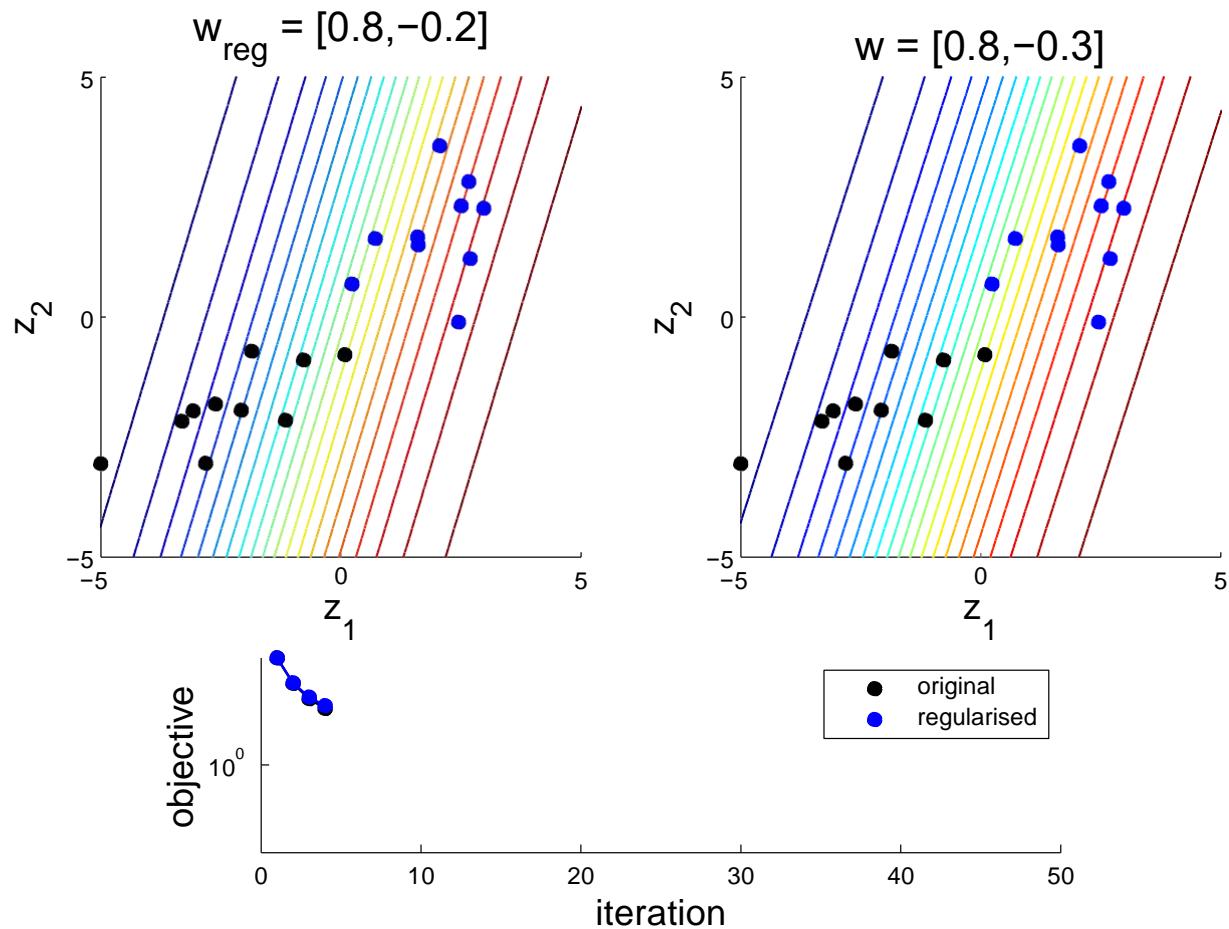
# Training a Single Neuron (cont'd)



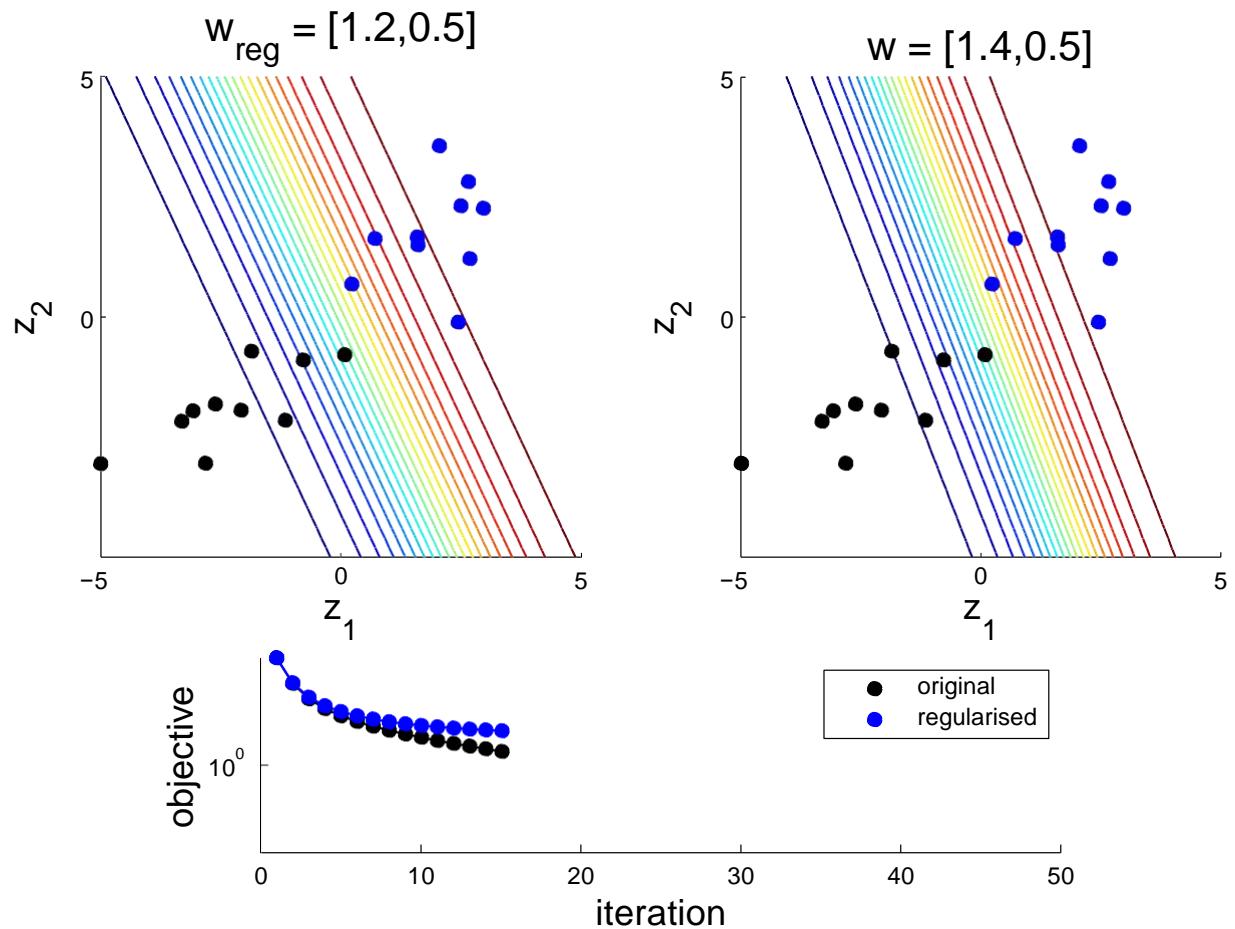
# Training a Single Neuron (cont'd)



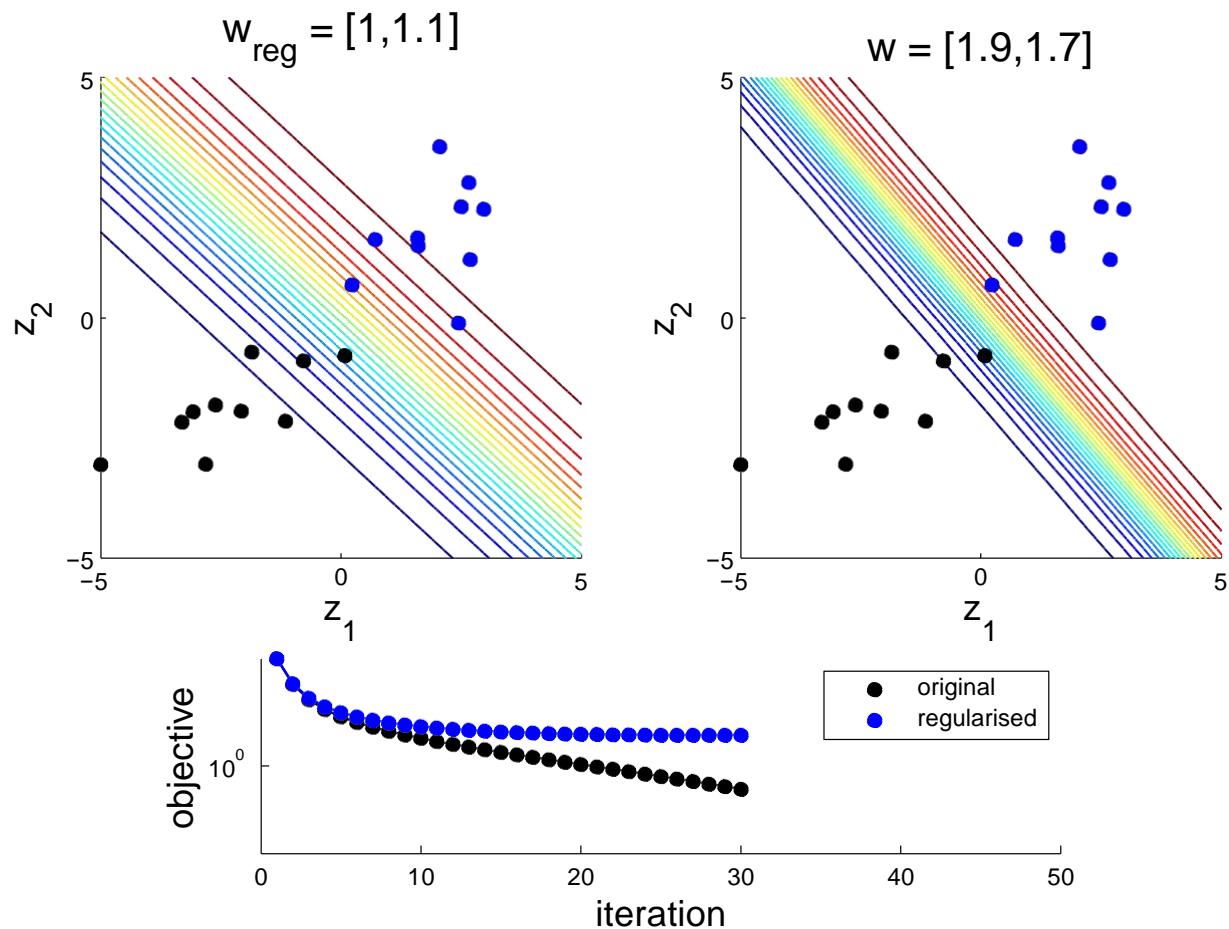
# Training a Single Neuron (cont'd)



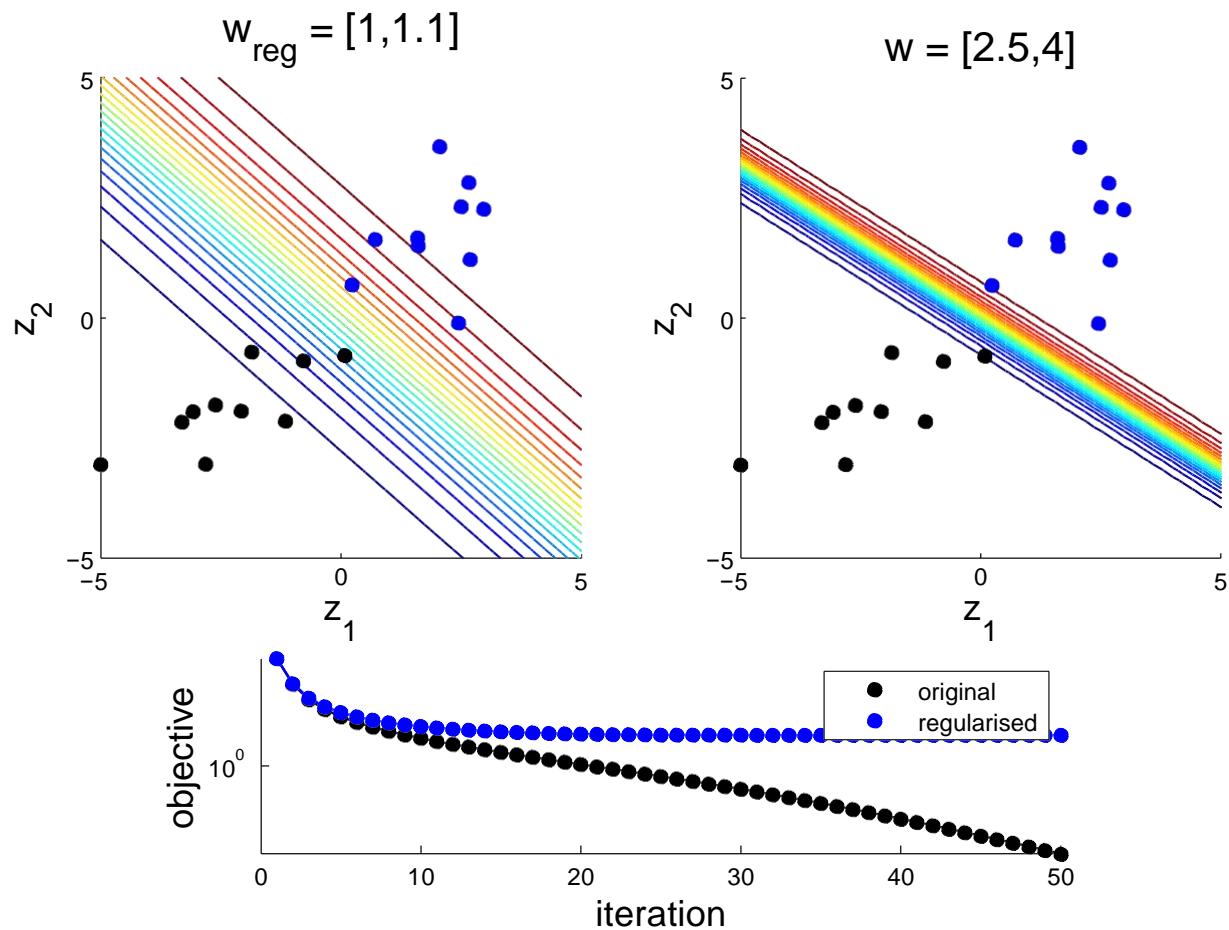
# Training a Single Neuron (cont'd)



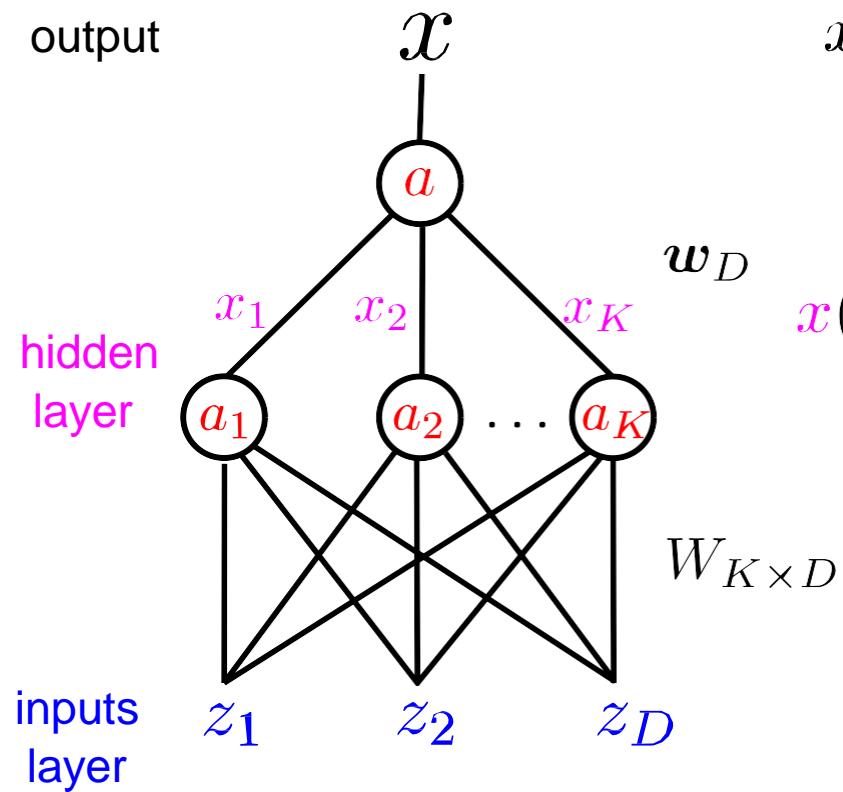
# Training a Single Neuron (cont'd)



# Training a Single Neuron (cont'd)



# Single Hidden Layer Neural Networks

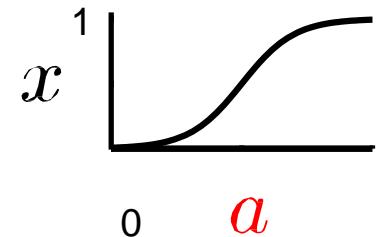


$$x(\textcolor{red}{a}) = \frac{1}{1 + \exp(-\textcolor{red}{a})}$$

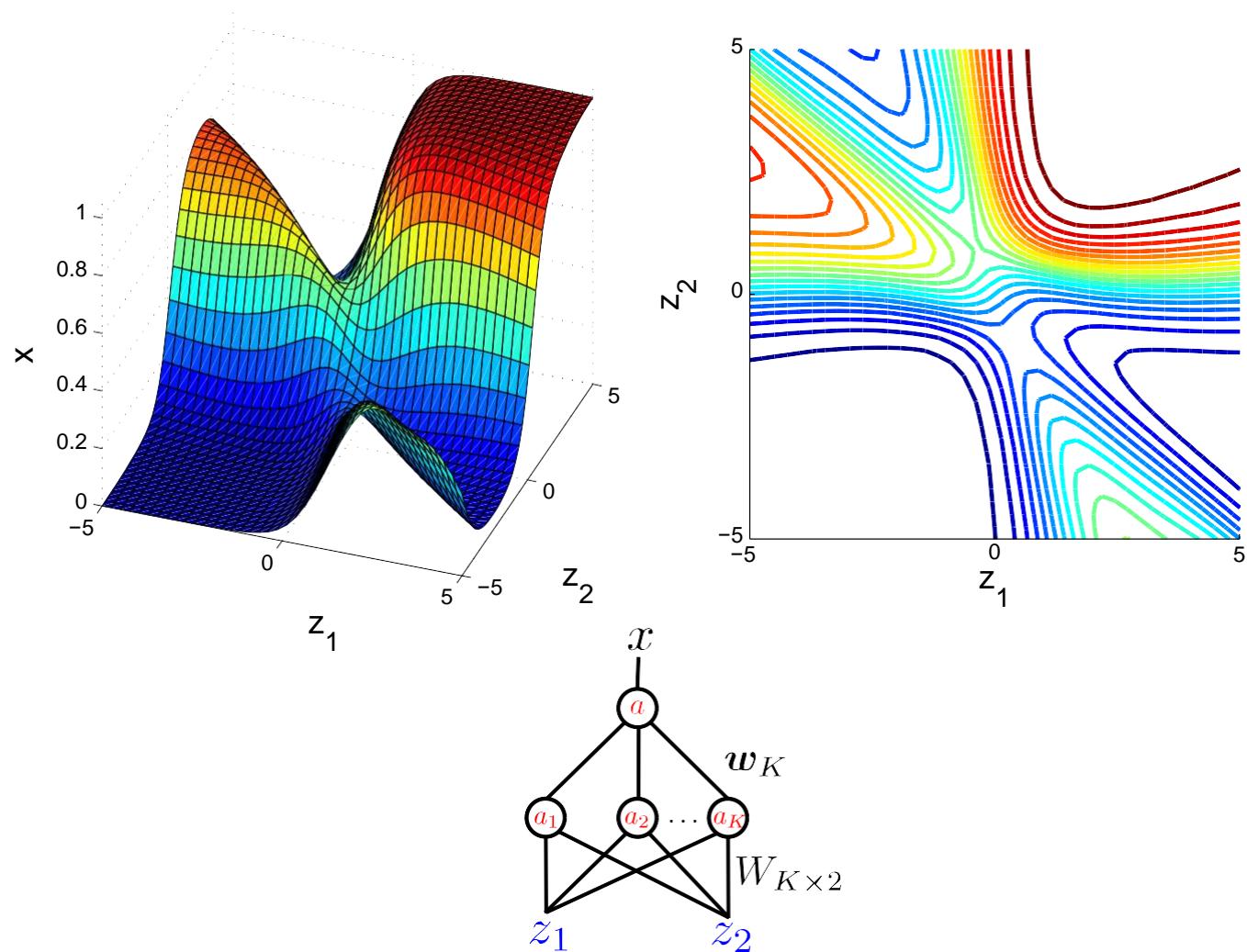
$$\textcolor{red}{a} = \sum_{k=1}^K w_k \textcolor{magenta}{x}_k$$

$$x(\textcolor{red}{a}_k) = \frac{1}{1 + \exp(-\textcolor{red}{a}_k)}$$

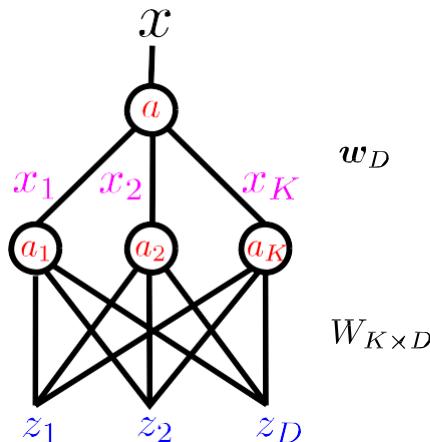
$$a_k = \sum_{d=1}^D W_{k,d} \textcolor{blue}{z}_d$$



# Sampling Random Neural Network Classifiers



# Training a Neural Network with a Single Hidden Layer



$$x(\textcolor{red}{a}) = \frac{1}{1 + \exp(-\textcolor{red}{a})}$$

$$\textcolor{red}{a} = \sum_{k=1}^K w_k \textcolor{violet}{x}_k$$

$$\textcolor{violet}{x}(\textcolor{red}{a}_k) = \frac{1}{1 + \exp(-\textcolor{red}{a}_k)}$$

$$a_k = \sum_{d=1}^D W_{k,d} \textcolor{blue}{z}_d$$

**objective function:**

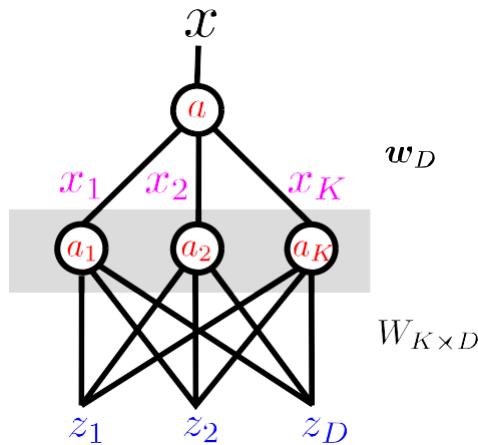
$$G(W, \mathbf{w}) = - \sum_n [t^{(n)} \log x^{(n)} + (1 - t^{(n)}) \log (1 - x^{(n)})] \quad \text{likelihood same as before}$$

$$E(W, \mathbf{w}) = \frac{1}{2} \sum_i w_i^2 + \frac{1}{2} \sum_{ij} W_{ij}^2 \quad \text{regulariser discourages extreme weights}$$

$$\{W, \mathbf{w}^*\} = \arg \min_{W, \mathbf{w}} M(W, \mathbf{w}) = \arg \min_{W, \mathbf{w}} [G(W, \mathbf{w}) + \alpha E(W, \mathbf{w})]$$

# Training a Neural Network with a Single Hidden Layer

Networks with hidden layers can be fit using gradient descent using an algorithm called **back-propagation**.



$$x(\mathbf{a}) = \frac{1}{1 + \exp(-\mathbf{a})}$$

$$\mathbf{a} = \sum_{k=1}^K w_k \mathbf{x}_k$$

$$\mathbf{x}(\mathbf{a}_k) = \frac{1}{1 + \exp(-\mathbf{a}_k)}$$

$$a_k = \sum_{d=1}^D W_{k,d} z_d$$

## objective function:

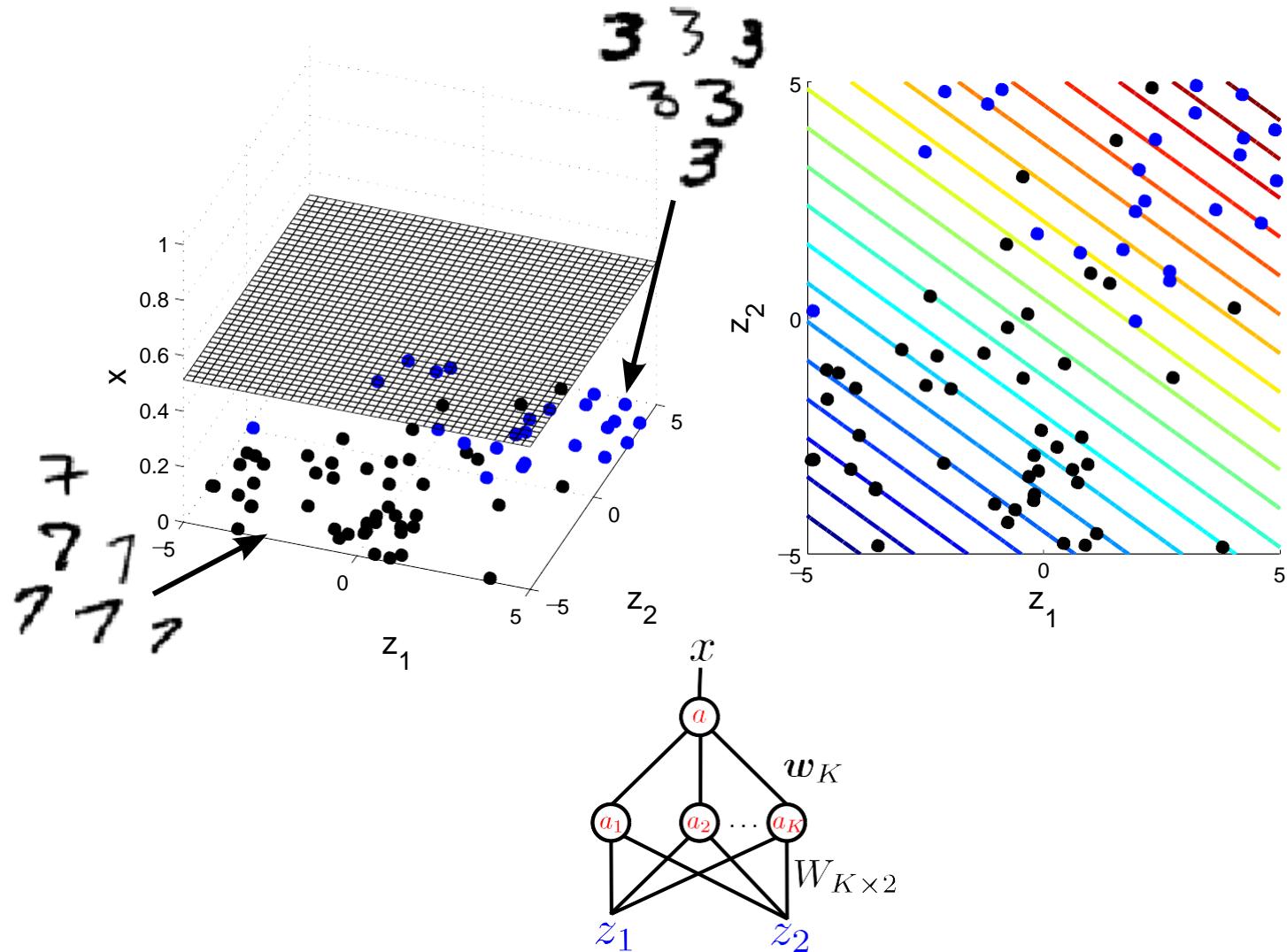
$$G(W, \mathbf{w}) = - \sum_n [t^{(n)} \log \mathbf{x}^{(n)} + (1 - t^{(n)}) \log (1 - \mathbf{x}^{(n)})] \quad \text{likelihood same as before}$$

$$E(W, \mathbf{w}) = \frac{1}{2} \sum_i w_i^2 + \frac{1}{2} \sum_{ij} W_{ij}^2 \quad \text{regulariser discourages extreme weights}$$

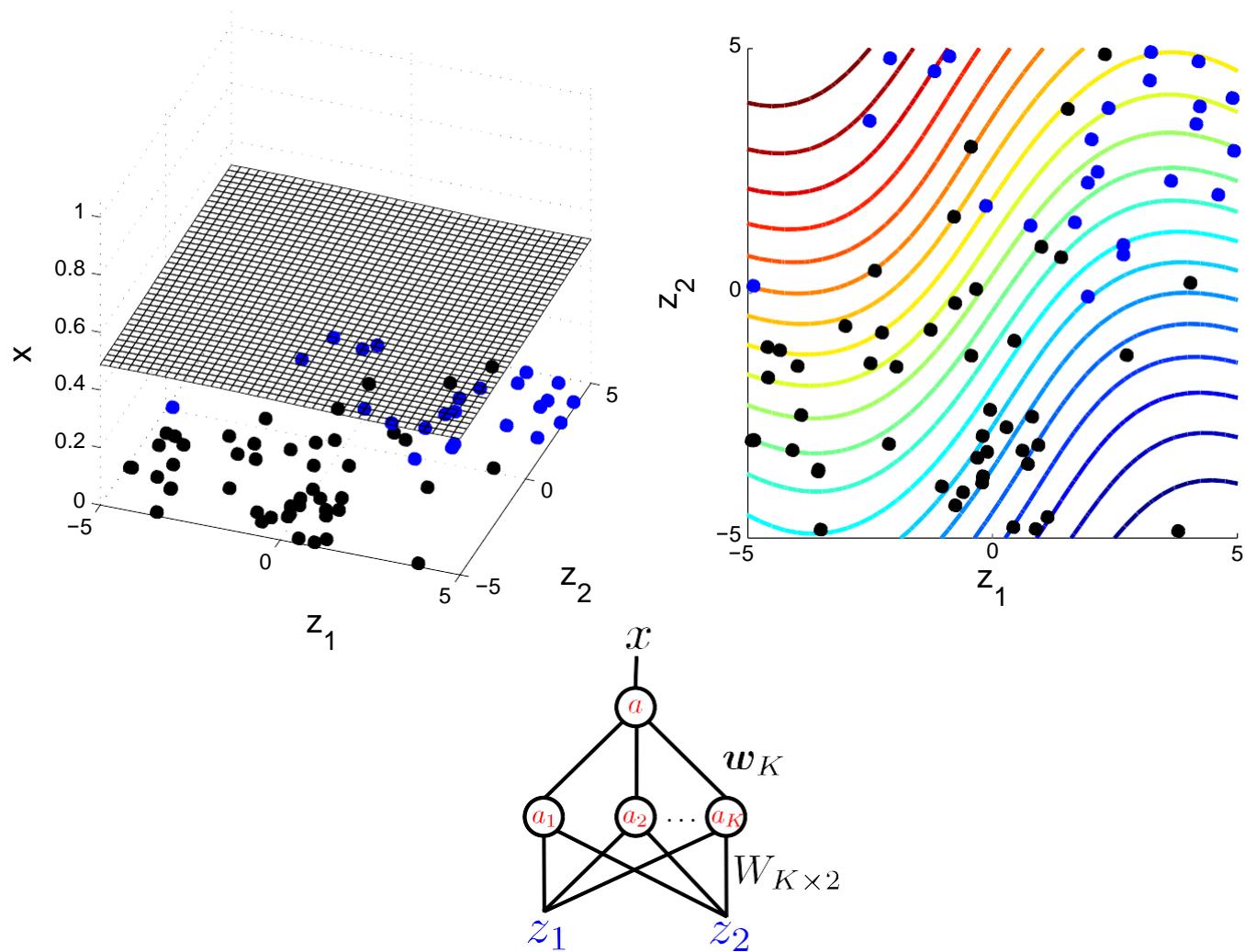
$$\{W, \mathbf{w}^*\} = \arg \min_{W, \mathbf{w}} M(W, \mathbf{w}) = \arg \min_{W, \mathbf{w}} [G(W, \mathbf{w}) + \alpha E(W, \mathbf{w})]$$

$$\begin{aligned} \frac{dG(W, \mathbf{w})}{dW_{ij}} &= \sum_n \frac{dG(W, \mathbf{w})}{dx^{(n)}} \frac{dx^{(n)}}{dW_{ij}} = \sum_n \frac{dG(W, \mathbf{w})}{dx^{(n)}} \frac{dx^{(n)}}{da^{(n)}} \frac{da^{(n)}}{dW_{ij}} \\ &= \sum_n \frac{dG(W, \mathbf{w})}{dx^{(n)}} \frac{dx^{(n)}}{da^{(n)}} \frac{da^{(n)}}{dx_i^{(n)}} \frac{dx_i^{(n)}}{dW_{ij}} = \sum_n \frac{dG(W, \mathbf{w})}{dx^{(n)}} \frac{dx^{(n)}}{da^{(n)}} \frac{da^{(n)}}{dx_i^{(n)}} \frac{dx_i^{(n)}}{da_i^{(n)}} \frac{da_i^{(n)}}{dW_{ij}} \end{aligned}$$

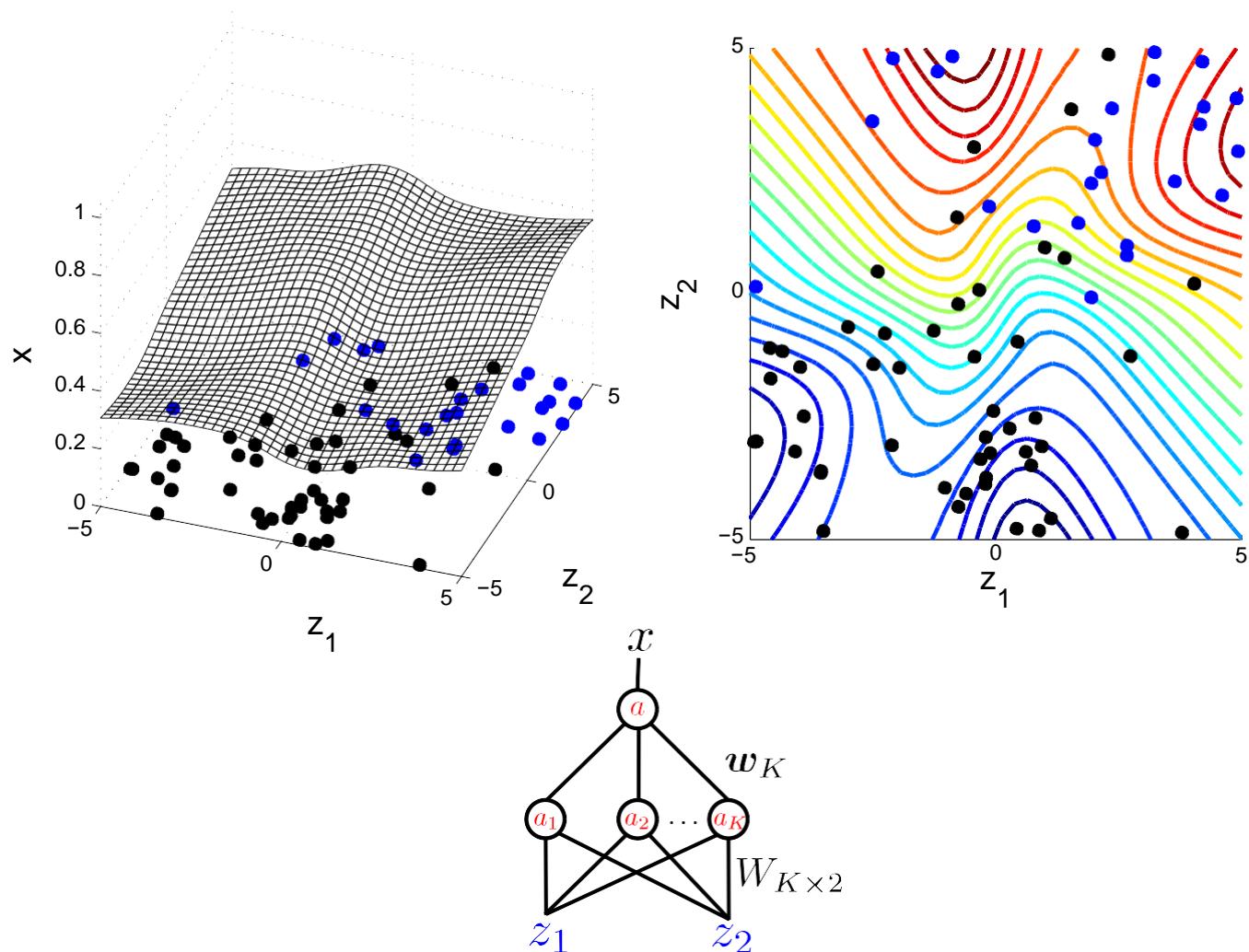
# Training a Neural Network with a Single Hidden Layer



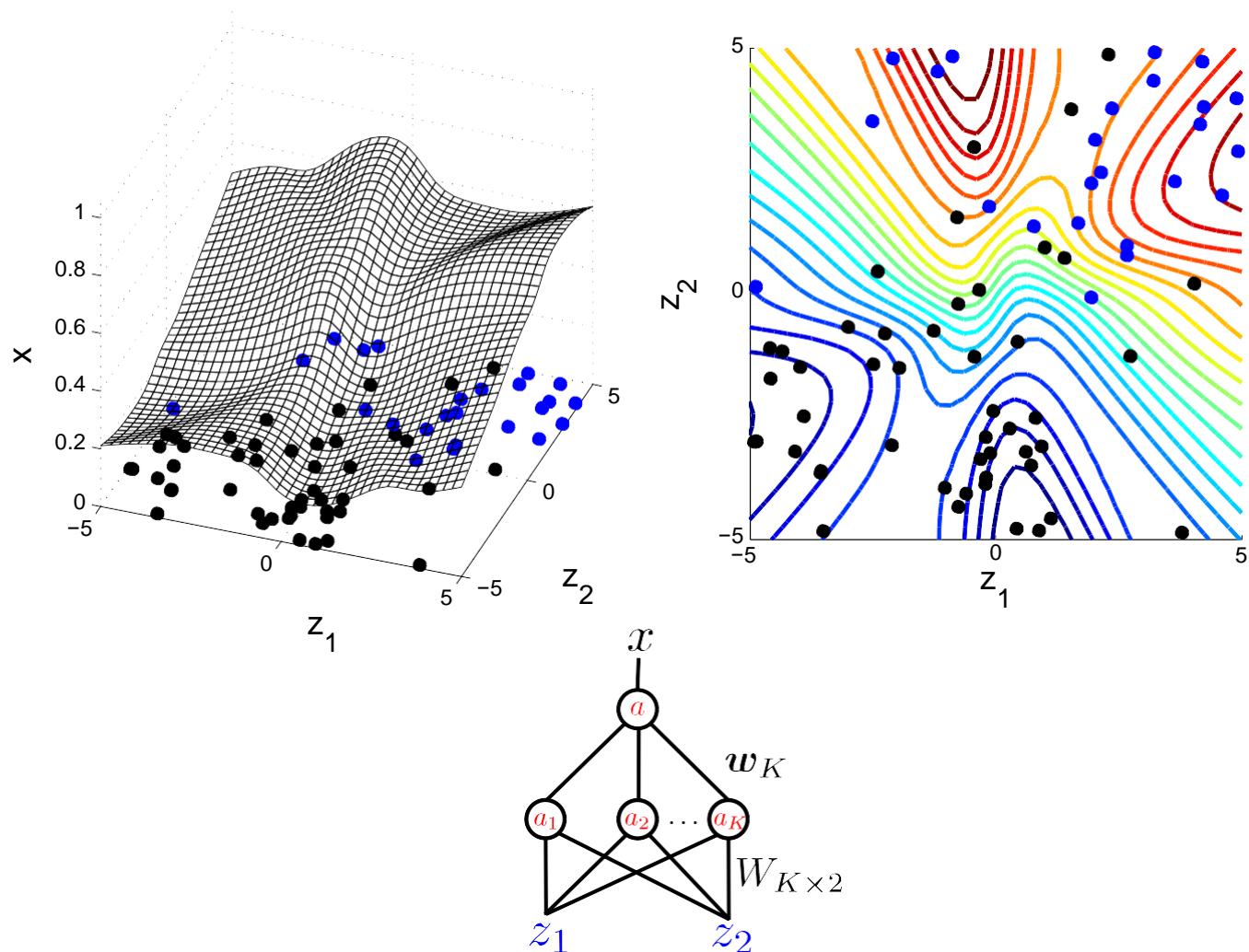
# Training a Neural Network with a Single Hidden Layer



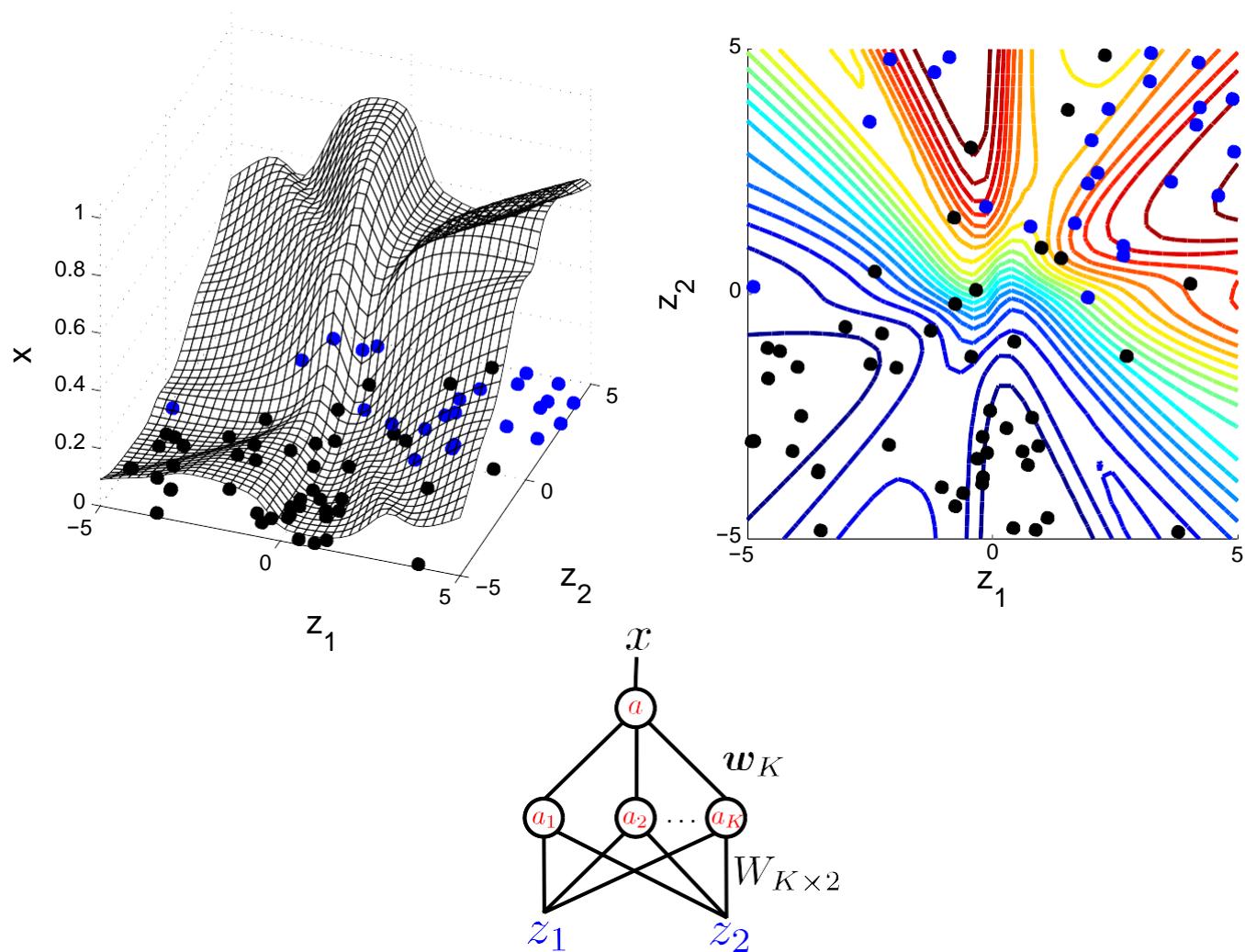
# Training a Neural Network with a Single Hidden Layer



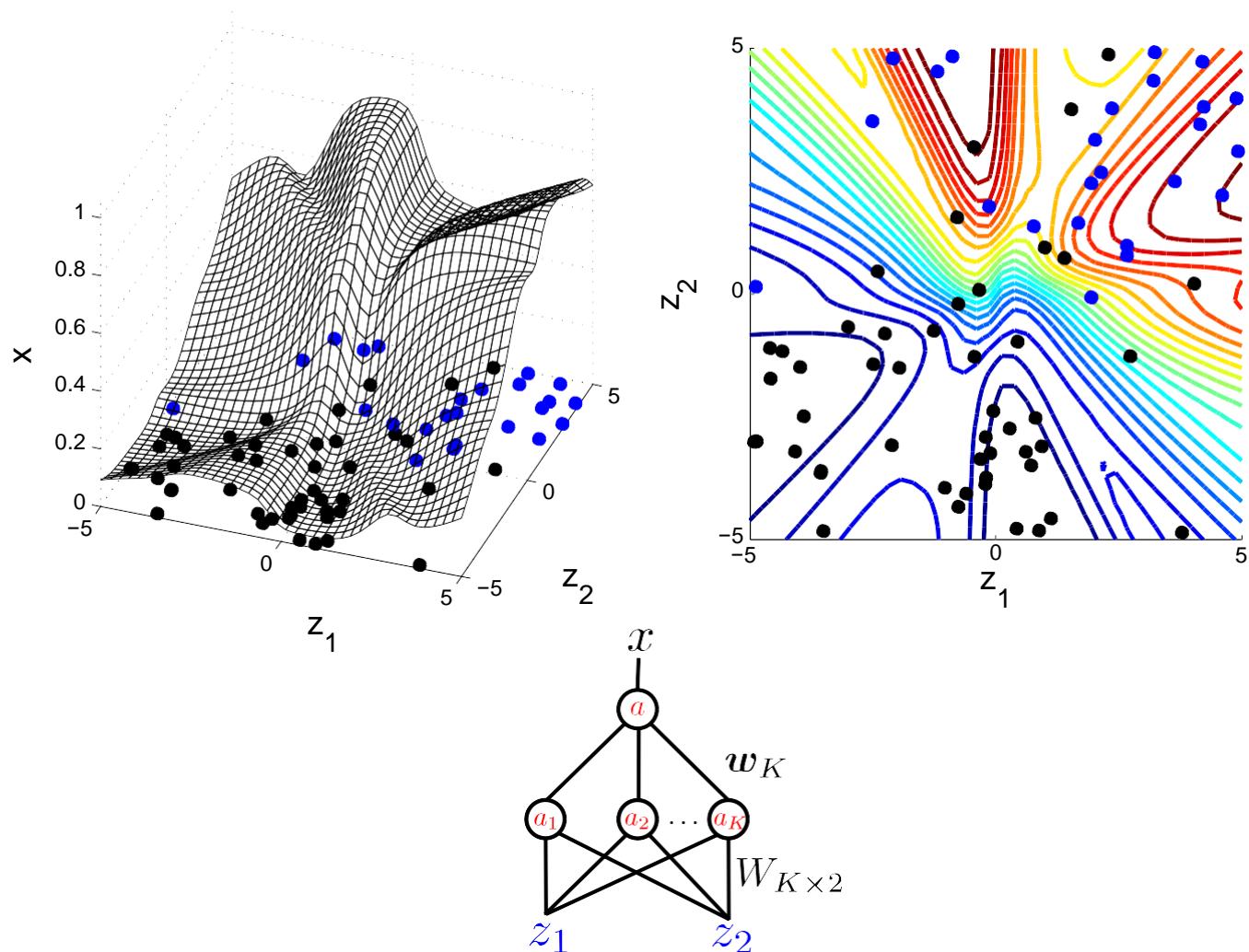
# Training a Neural Network with a Single Hidden Layer



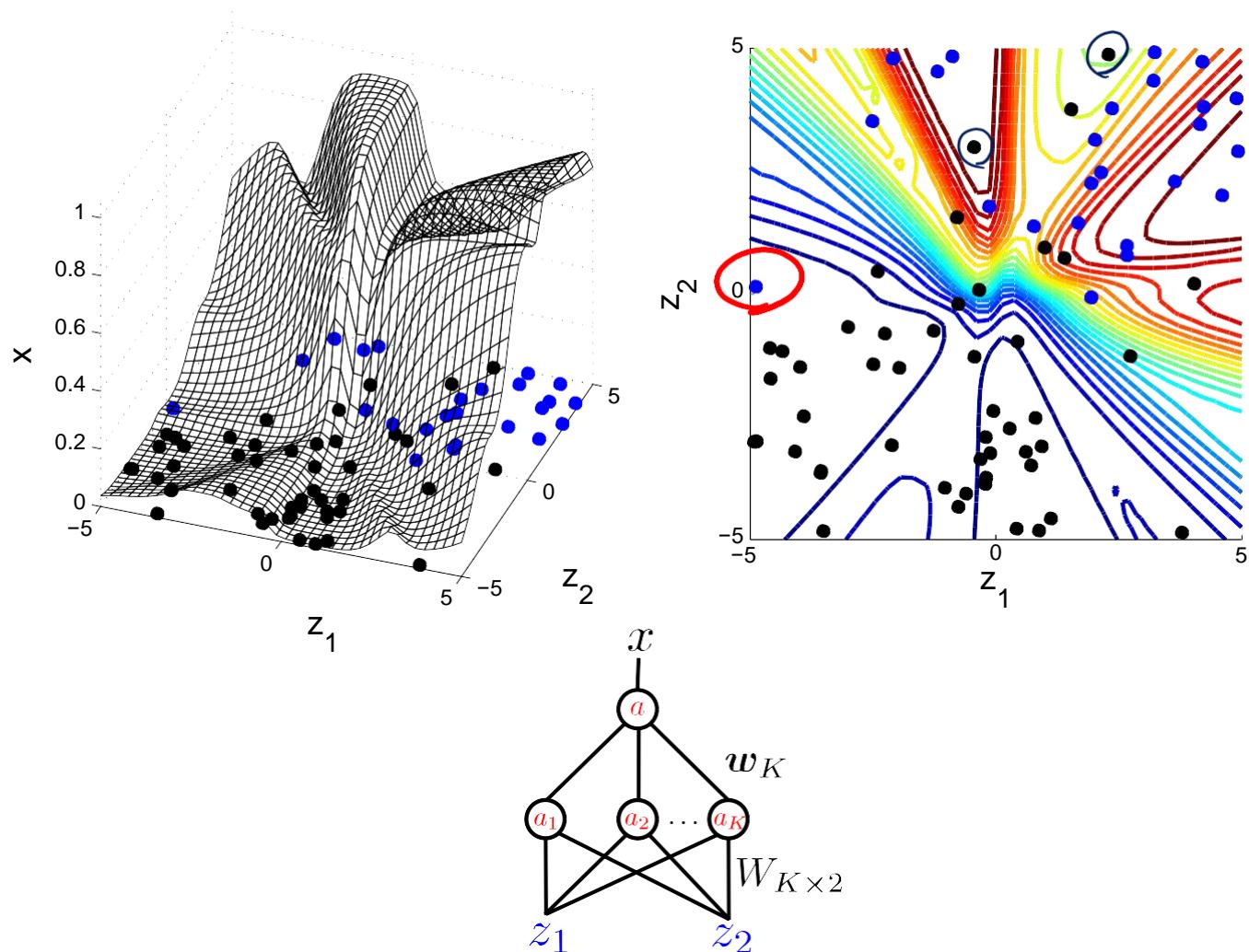
# Training a Neural Network with a Single Hidden Layer



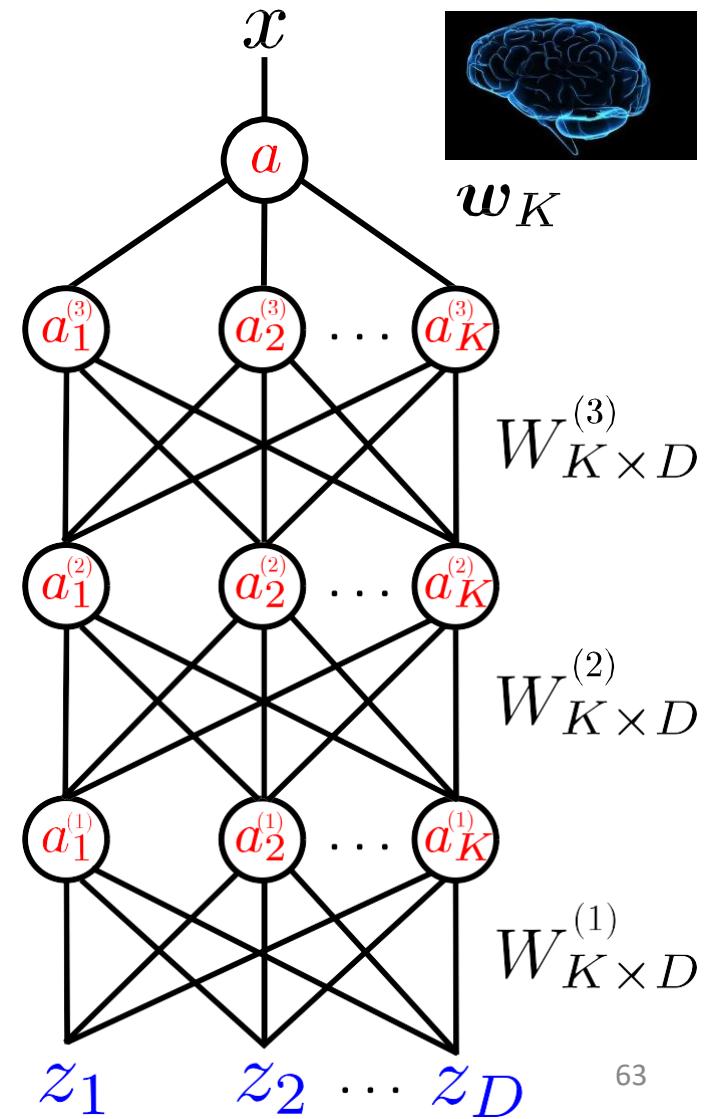
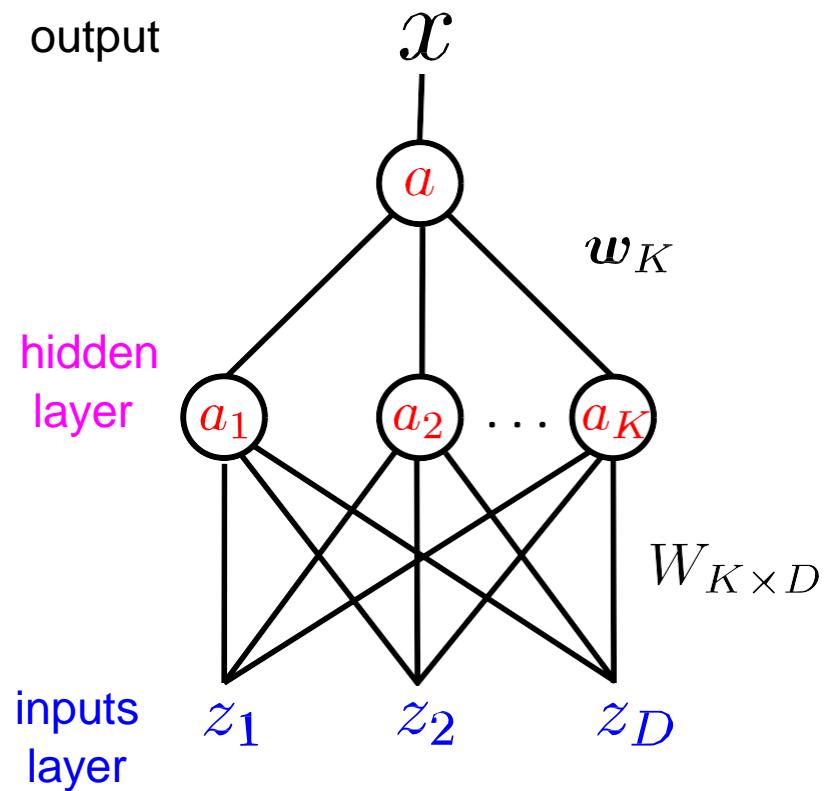
# Training a Neural Network with a Single Hidden Layer



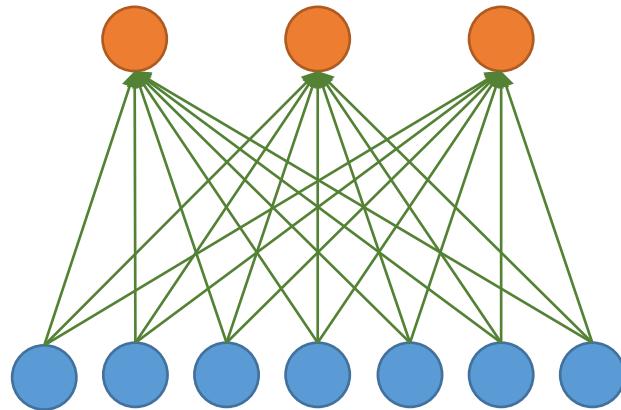
# Training a Neural Network with a Single Hidden Layer



# Hierarchical Models with Many Layers



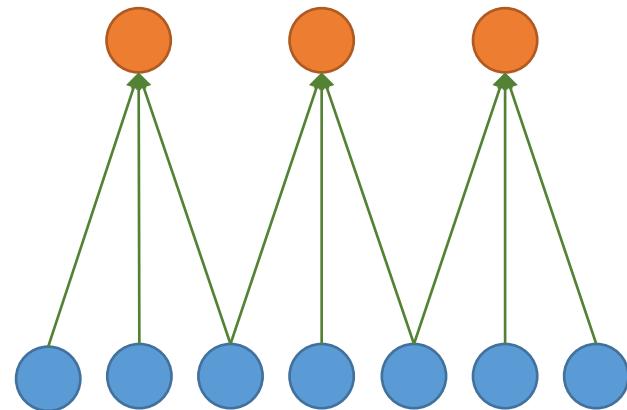
# Convolutional Neural Networks (CNN): Local Connectivity



**Global** connectivity

Hidden layer

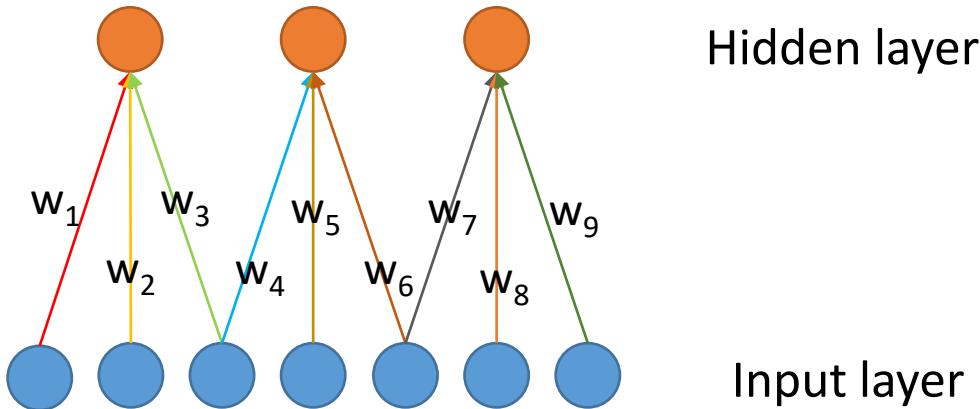
Input layer



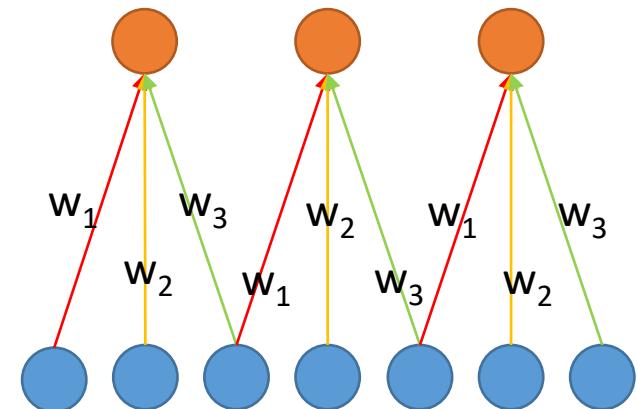
**Local** connectivity

- # input units (neurons): 7
- # hidden units: 3
- Number of parameters
  - Global connectivity: 21
  - Local connectivity: 9

# Convolutional Neural Networks (CNN): Weight Sharing



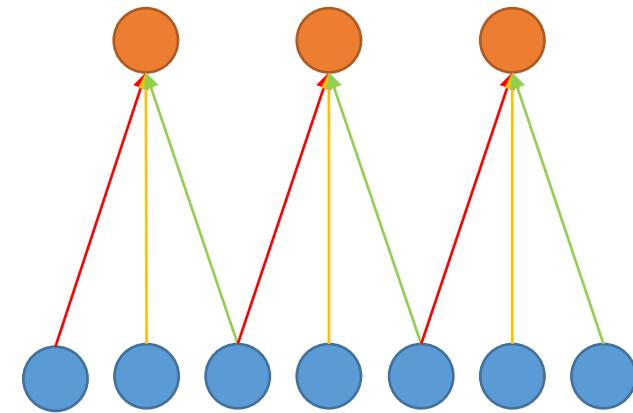
**Without** weight sharing



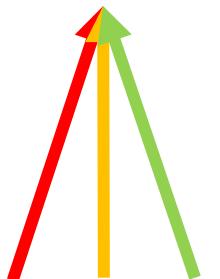
**With** weight sharing

- # input units (neurons): 7
- # hidden units: 3
- Number of parameters
  - Without weight sharing: 9
  - With weight sharing : 3

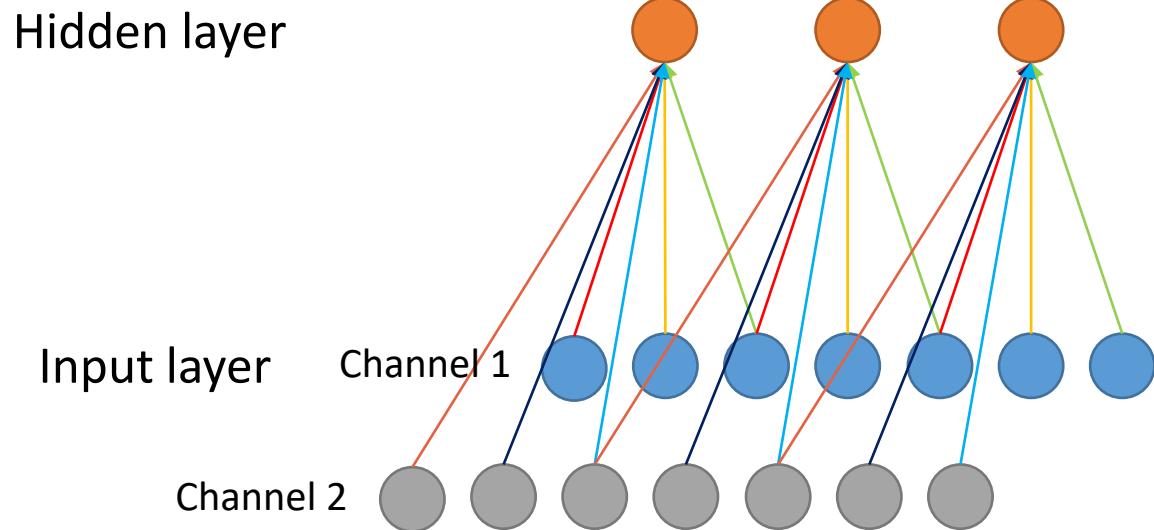
# CNN with Multiple Input Channels



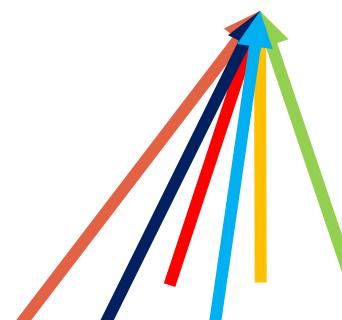
**Single input channel**



**Filter weights**

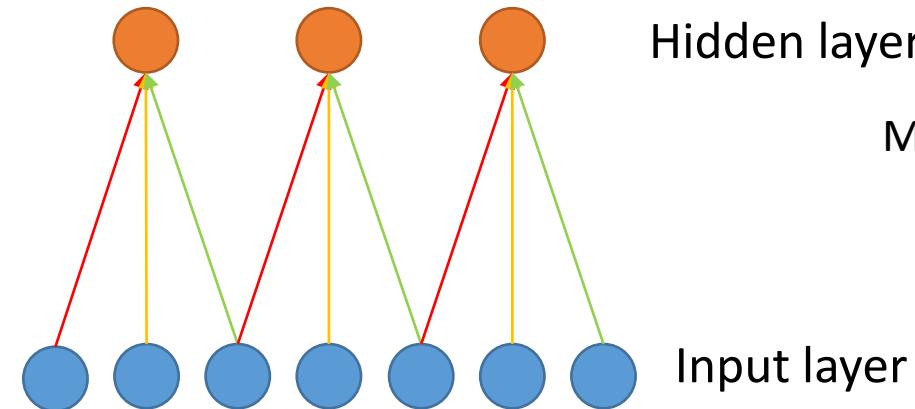


**Multiple input channels**



**Filter weights**

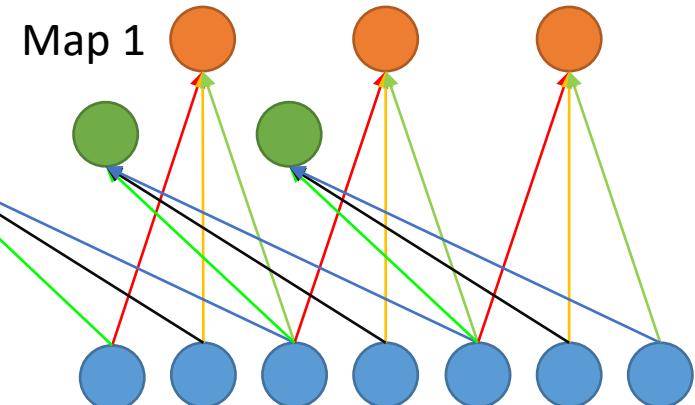
# CNN with Multiple Output Maps



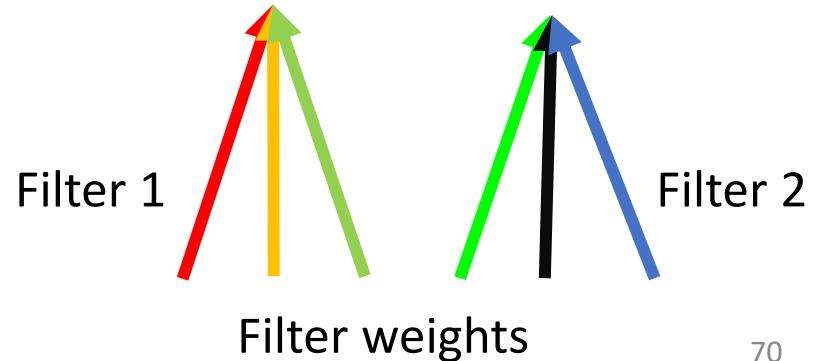
**Single output map**



Hidden layer  
Map 2  
Input layer

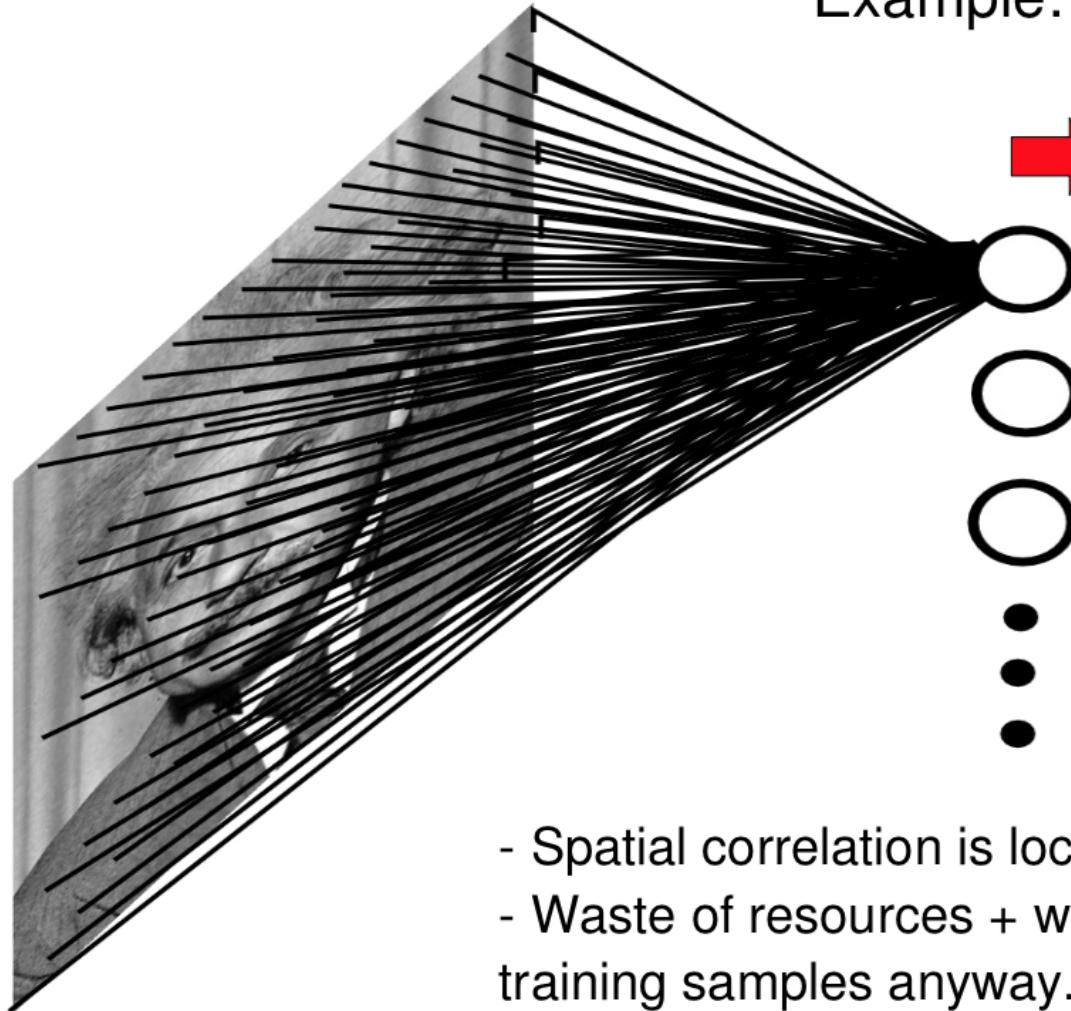


**Multiple output maps**



Generalized to 2D Cases:

## Fully Connected Layer



Example: 200x200 image

40K hidden units

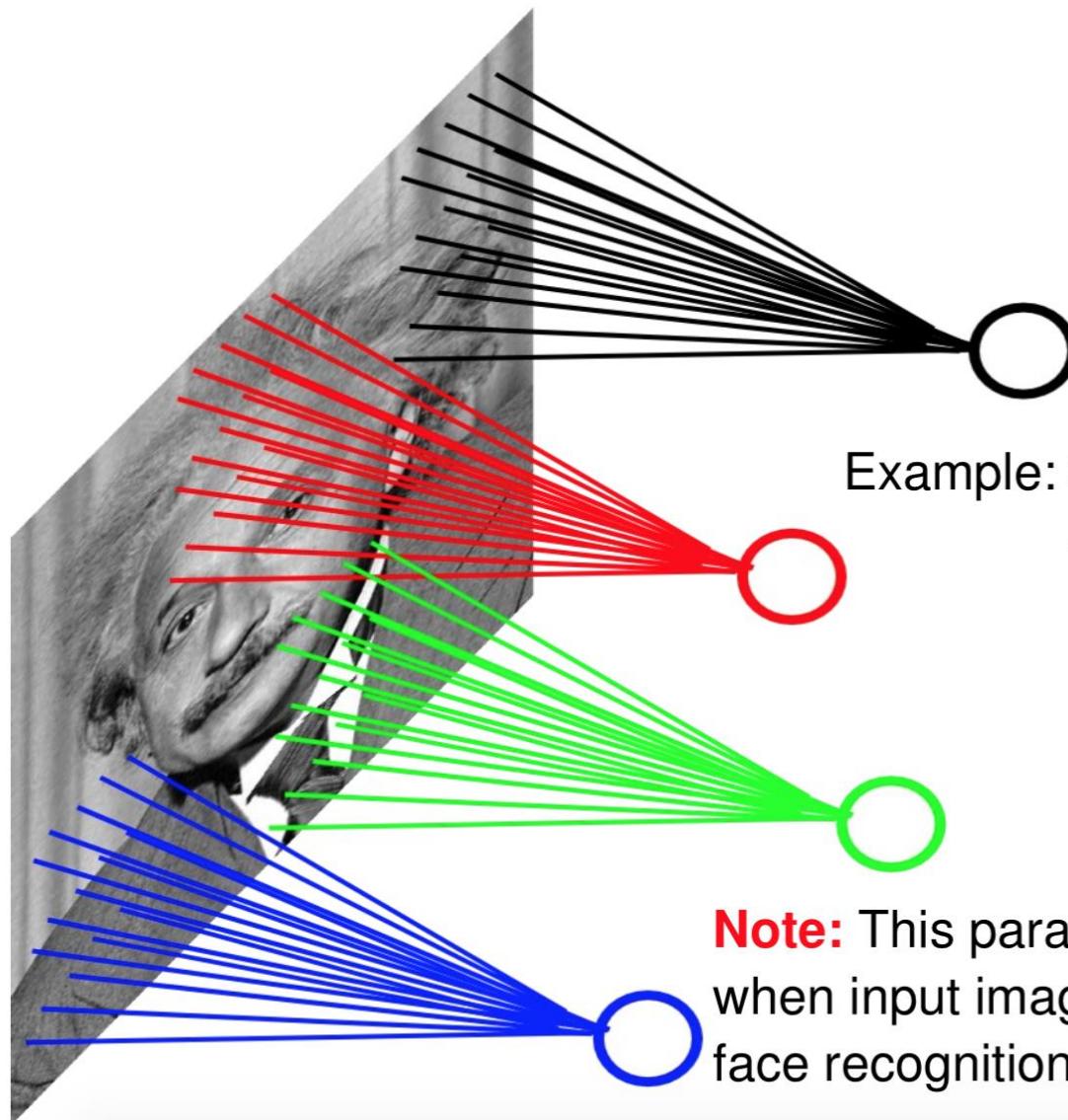
→ **~2B parameters!!!**

- Spatial correlation is local
- Waste of resources + we have not enough training samples anyway..

33  
71

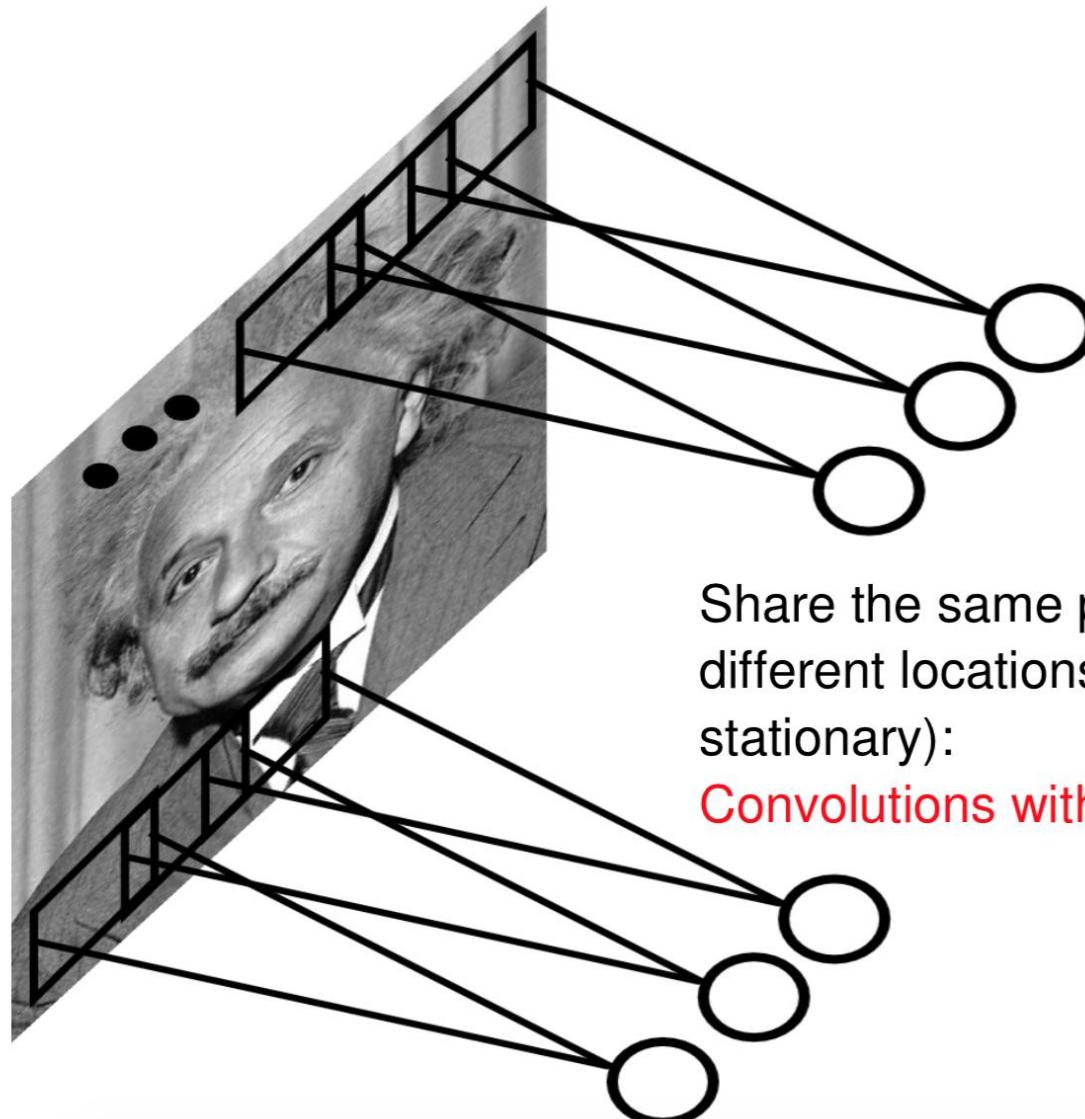
# Generalized to 2D Cases:

## Locally Connected Layer



Generalized to 2D Cases:

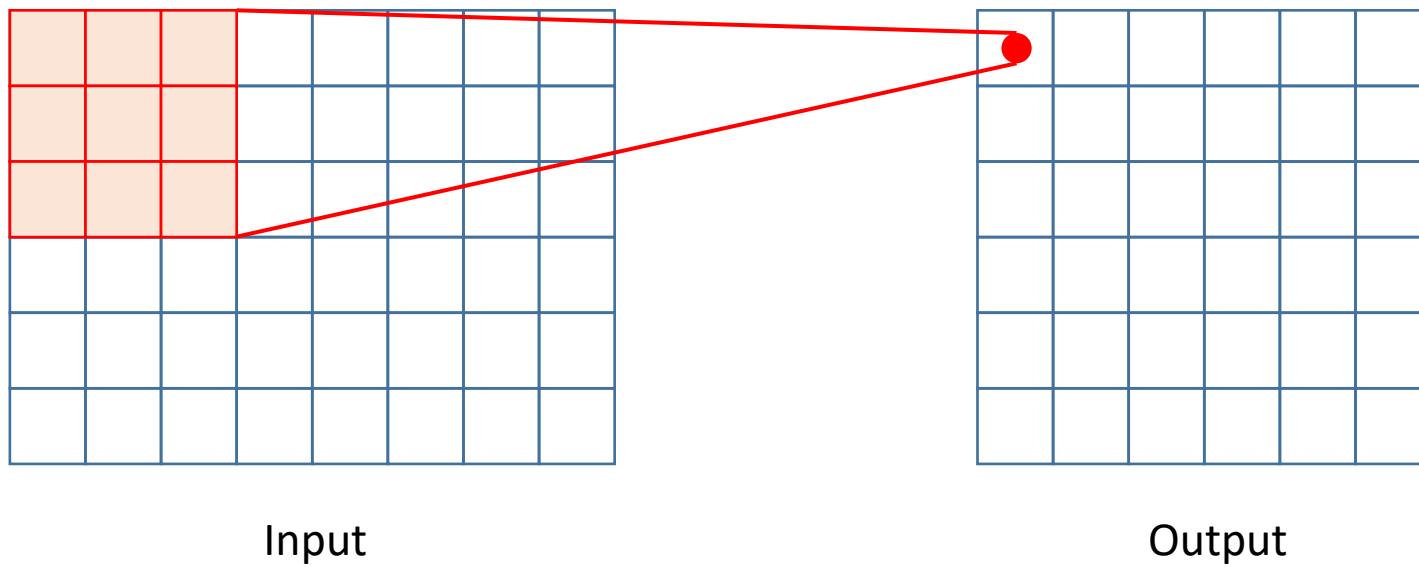
## Convolutional Layer



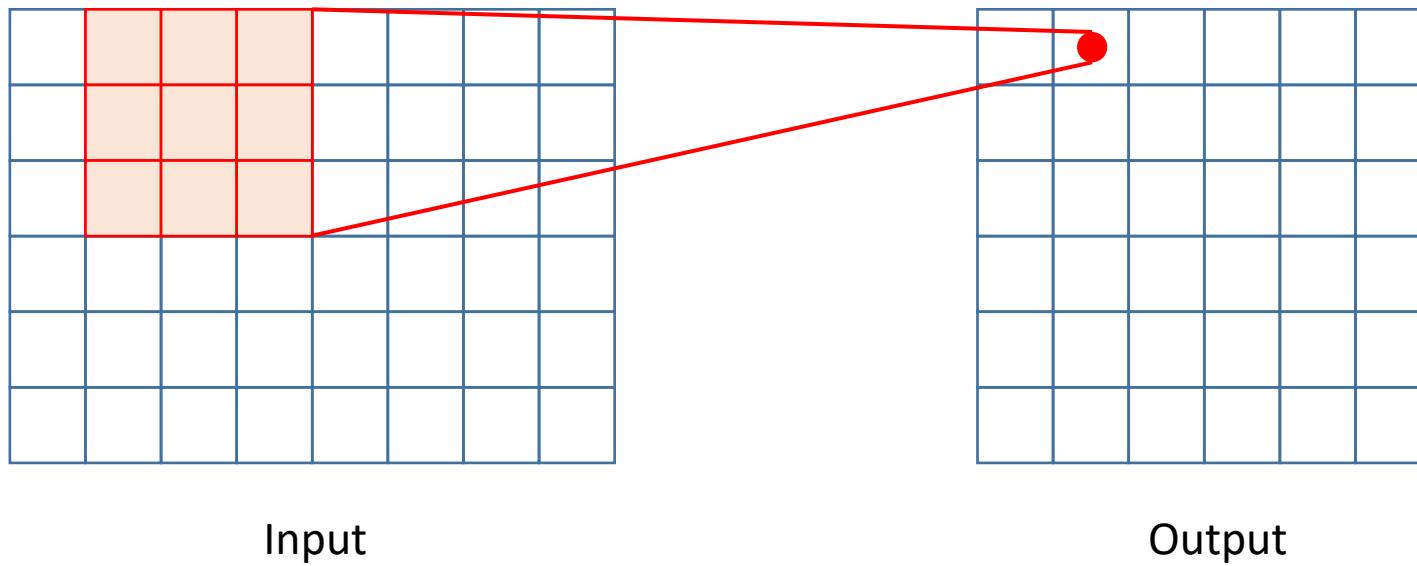
Share the same parameters across  
different locations (assuming input is  
stationary):  
**Convolutions with learned kernels**

36

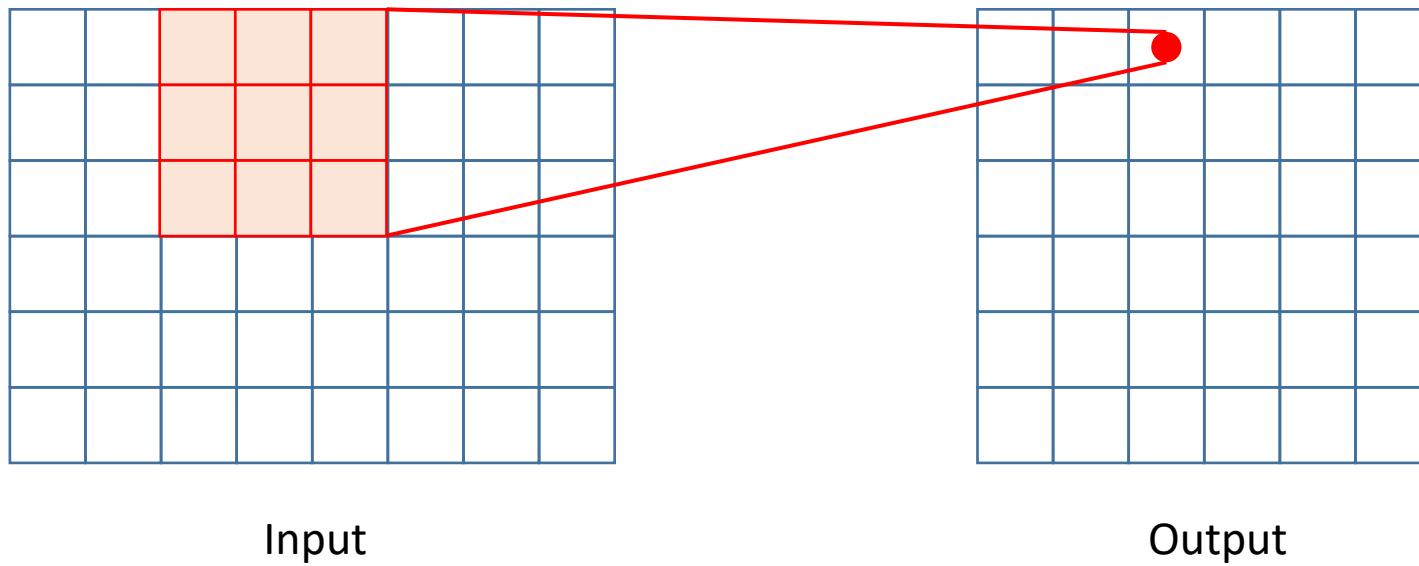
# Convolutional Layer



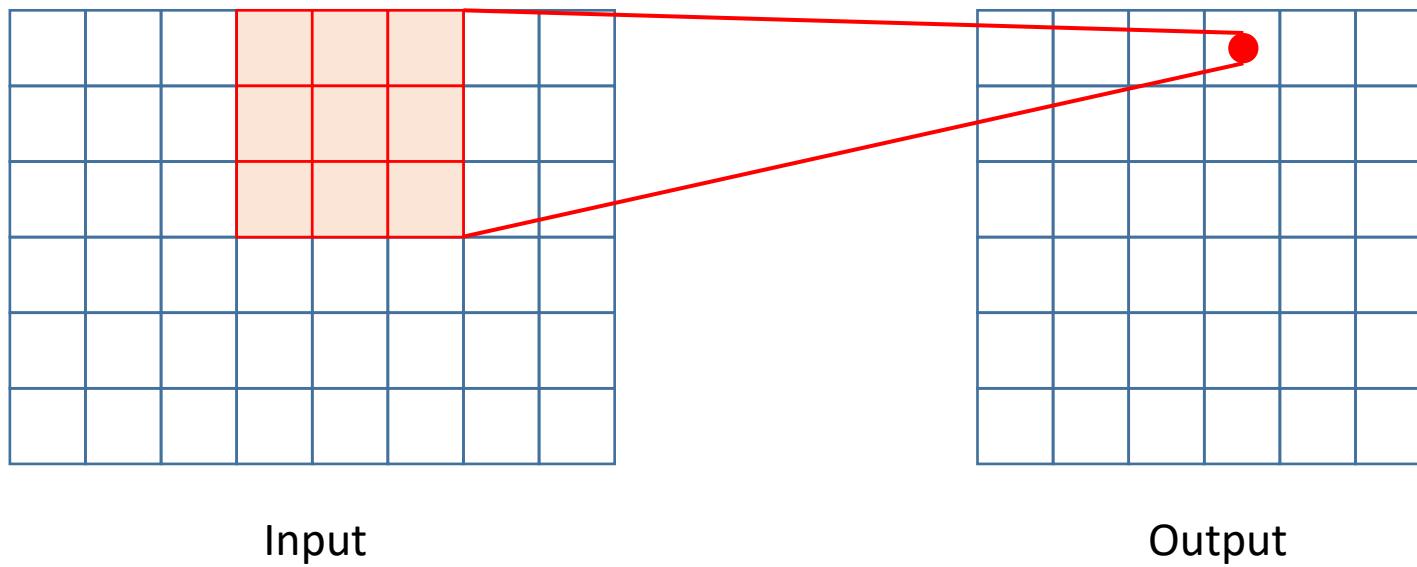
# Convolutional Layer



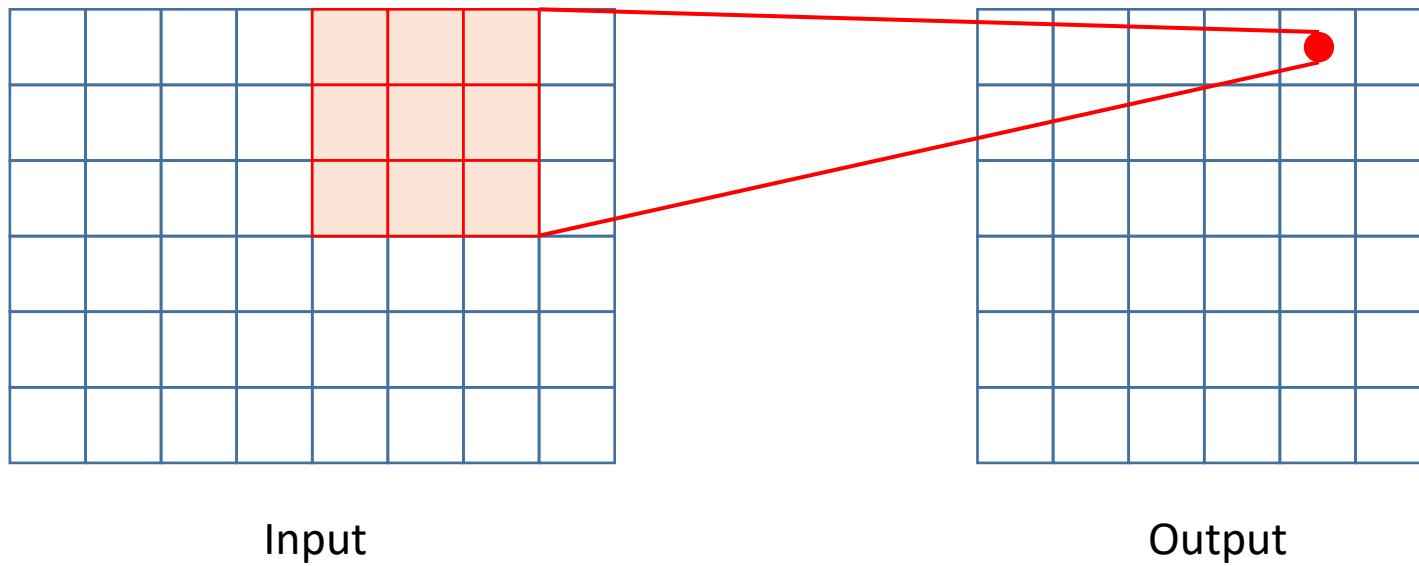
# Convolutional Layer



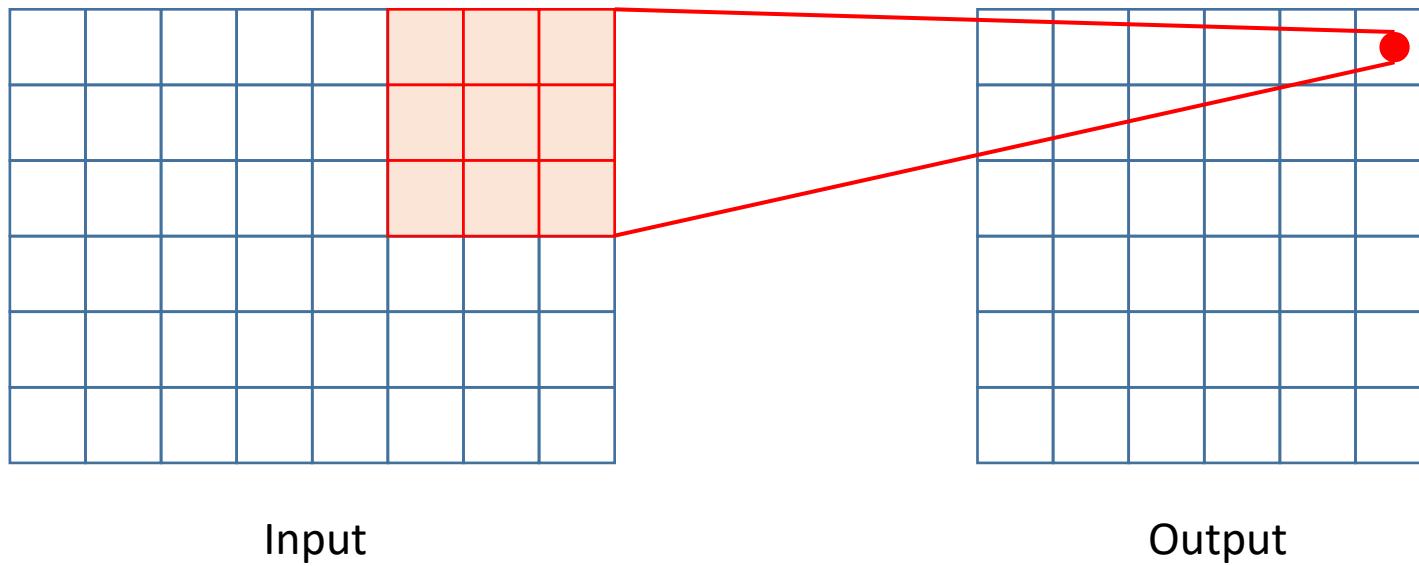
# Convolutional Layer



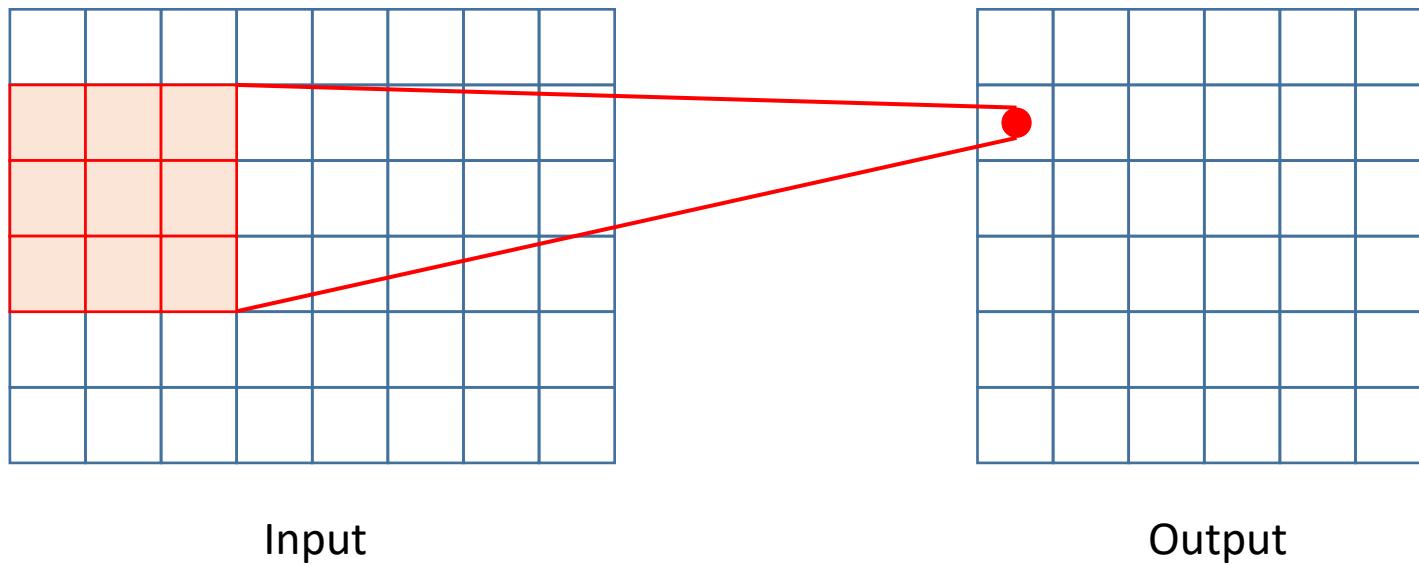
# Convolutional Layer



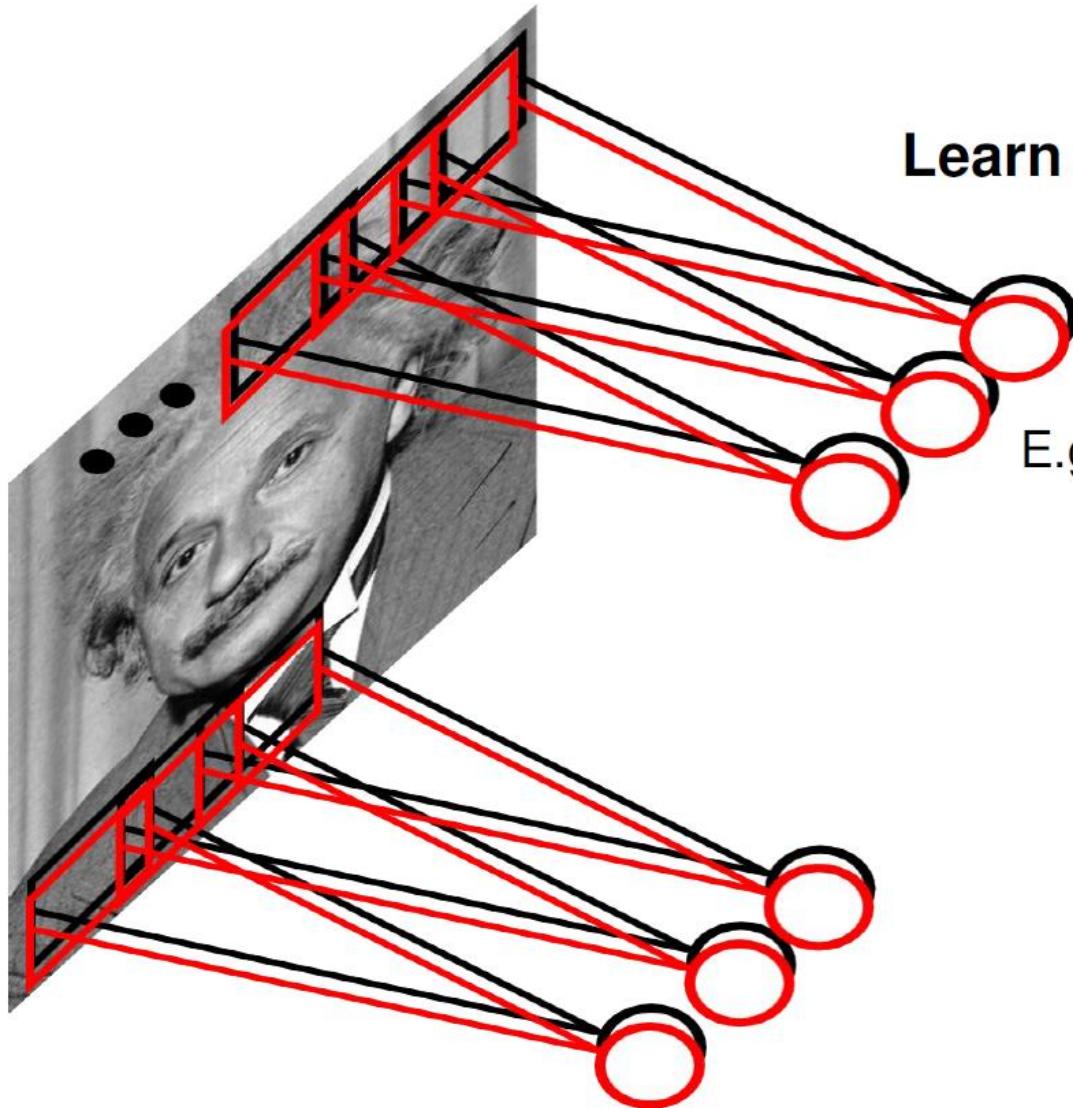
# Convolutional Layer



# Convolutional Layer



# Convolutional Layer



**Learn** multiple filters.

E.g.: 200x200 image  
100 Filters  
Filter size: 10x10  
10K parameters

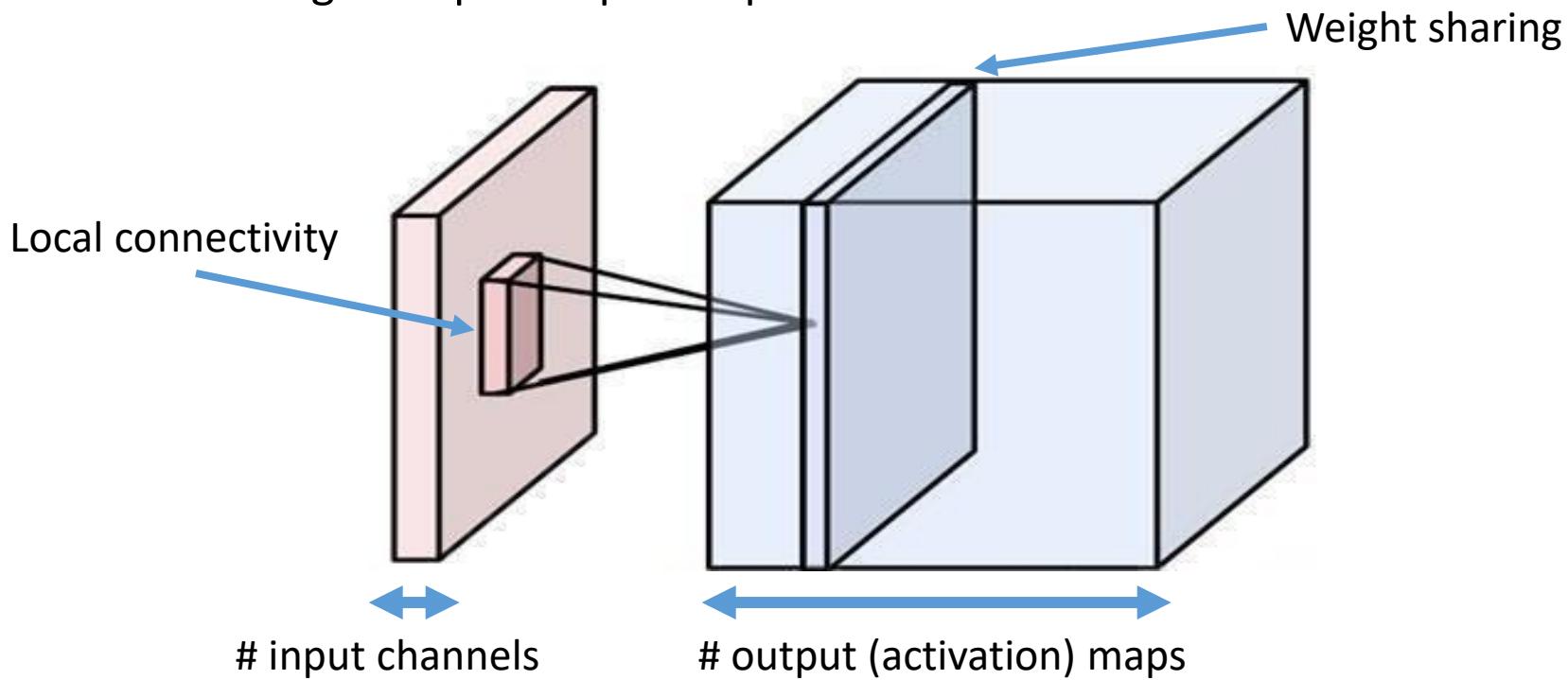
61

Ranzato

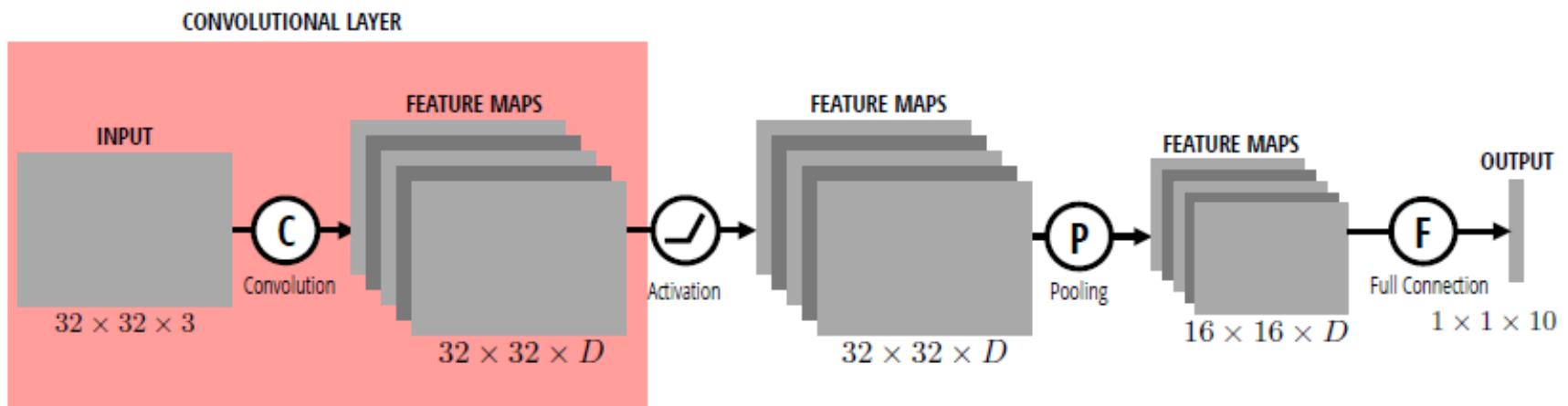
81

# Putting them together → CNN

- Local connectivity
- Weight sharing
- Handling multiple input channels
- Handling multiple output maps

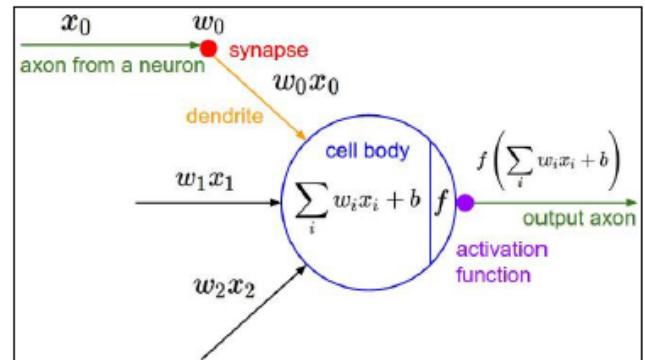
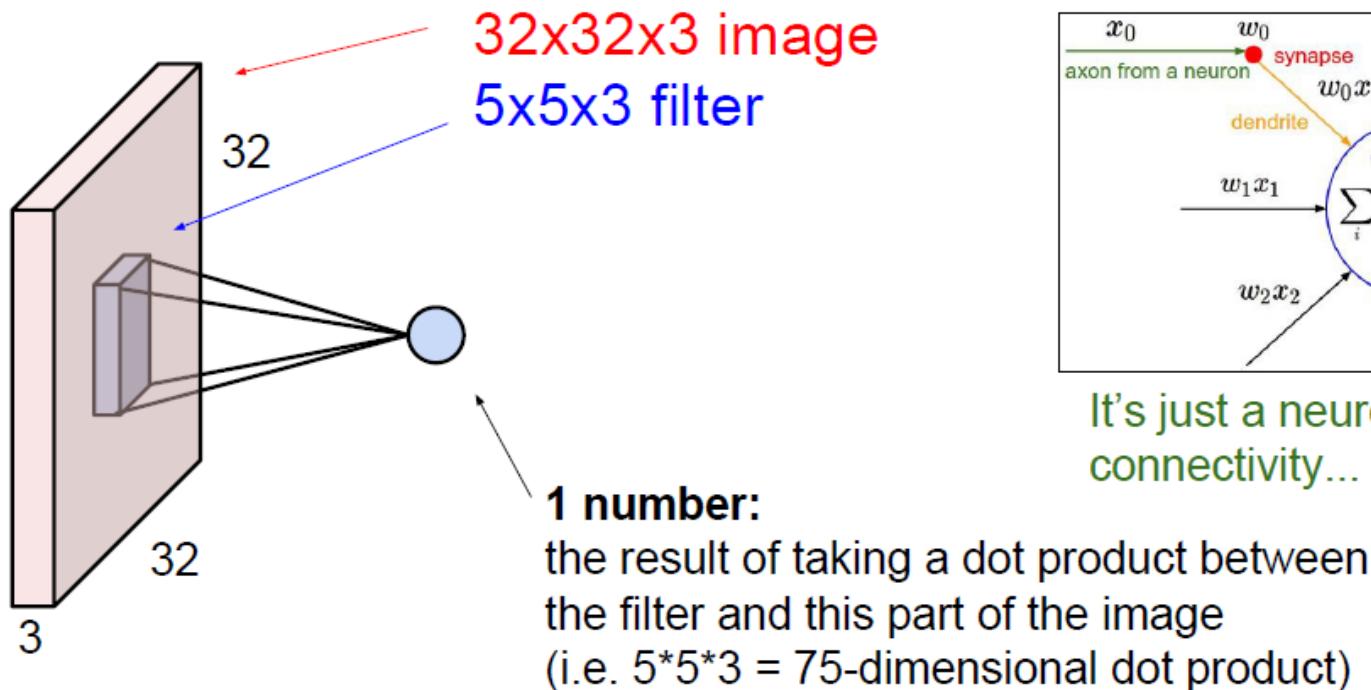


# Convolution Layer in CNN



# Putting them together (cont'd)

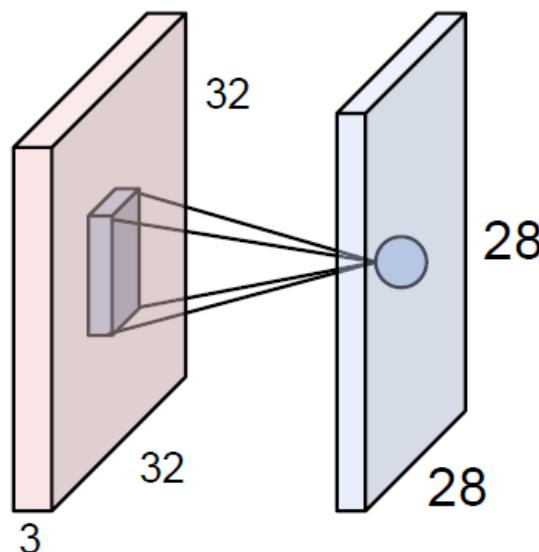
- The brain/neuron view of CONV layer



It's just a neuron with local connectivity...

# Putting them together (cont'd)

- The brain/neuron view of CONV layer



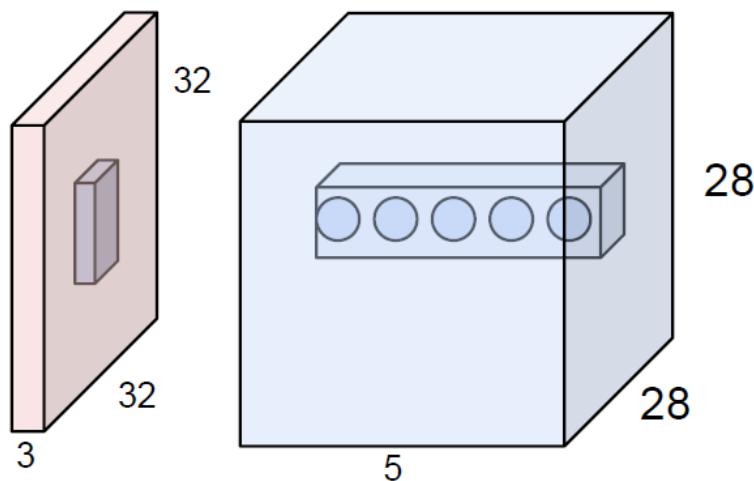
An activation map is a  $28 \times 28$  sheet of neuron outputs:

- Each is connected to a small region in the input
- All of them share parameters

“5x5 filter” -> “5x5 receptive field for each neuron”

# Putting them together (cont'd)

- The brain/neuron view of CONV layer

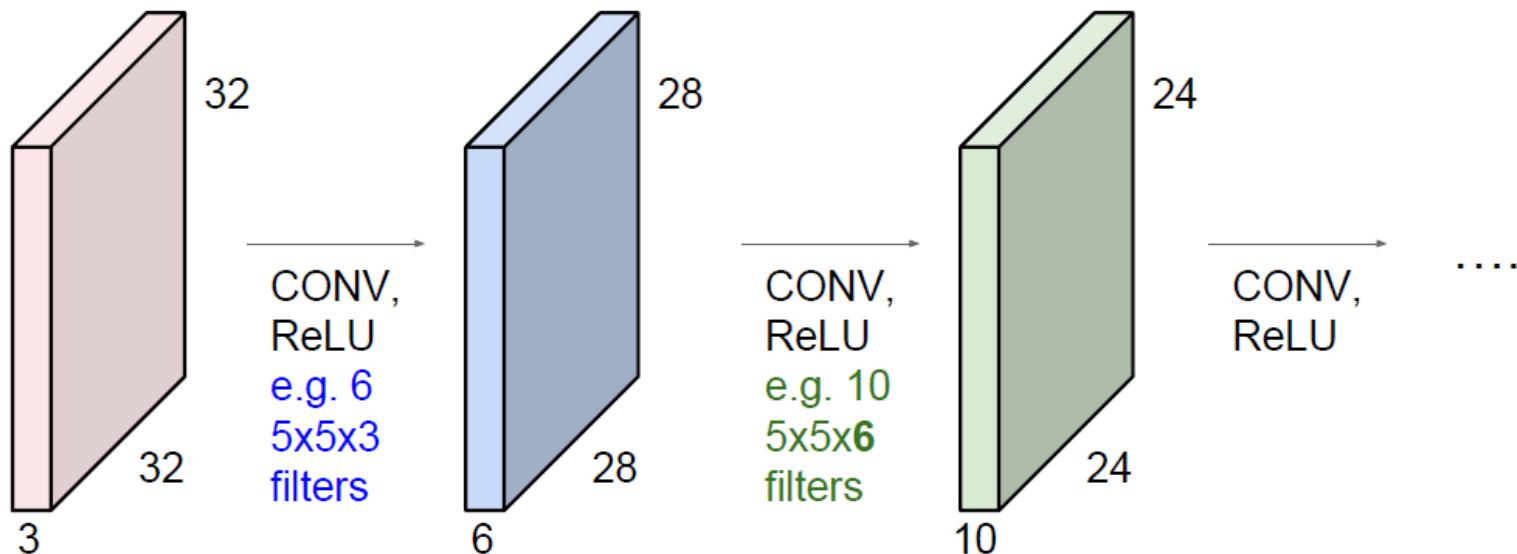


E.g. with 5 filters,  
CONV layer consists of  
neurons arranged in a 3D grid  
(28x28x5)

There will be 5 different  
neurons all looking at the same  
region in the input volume

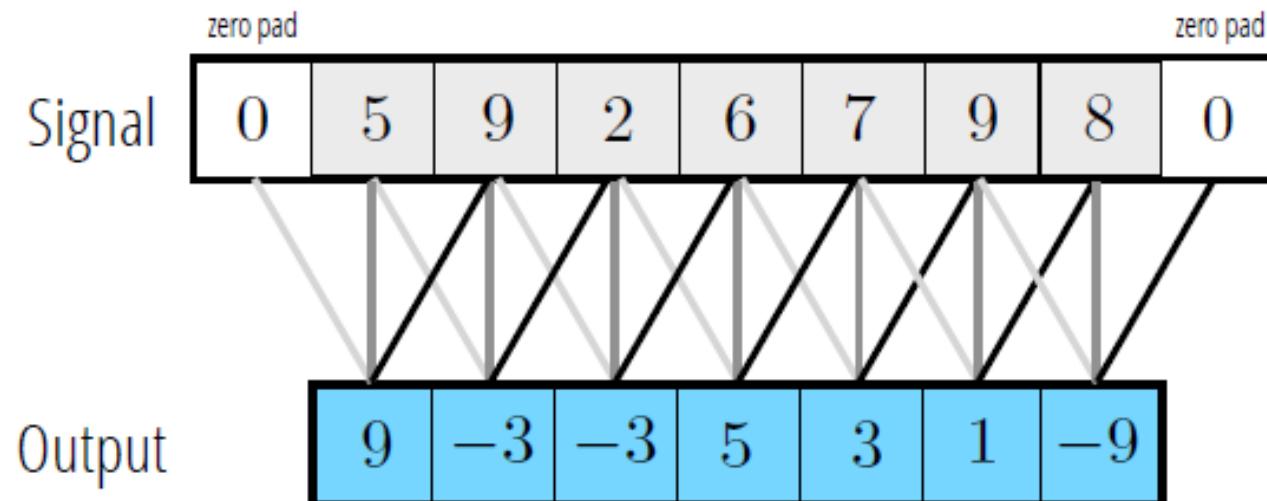
# Putting them together (cont'd)

- Image input with  $32 \times 32$  pixels convolved repeatedly with  $5 \times 5 \times 3$  filters shrinks volumes spatially ( $32 \rightarrow 28 \rightarrow 24 \rightarrow \dots$ ).



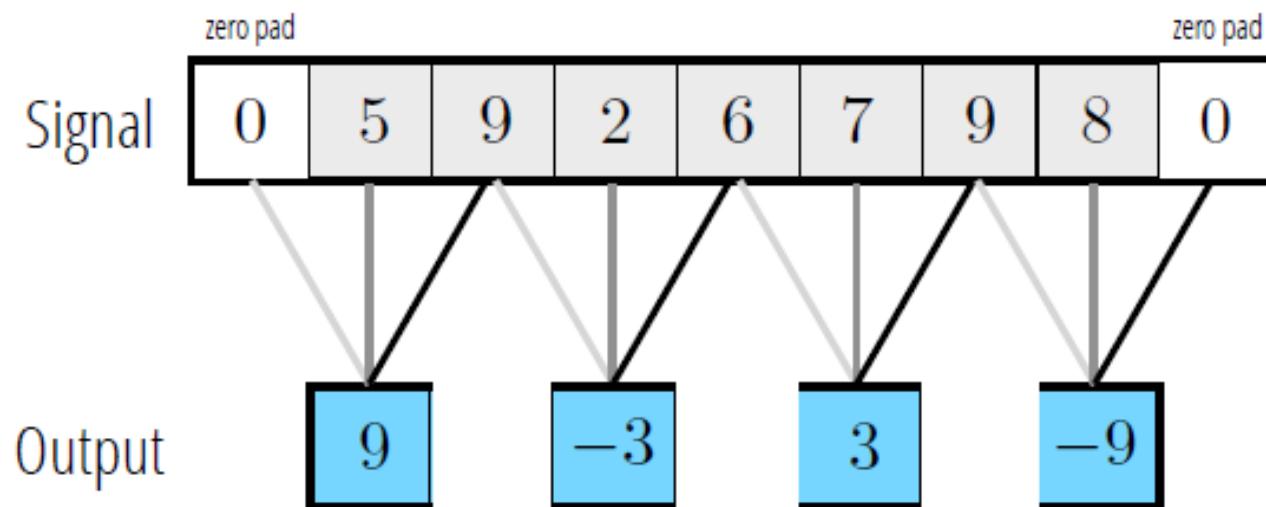
# Variations of Convolution

- Zero Padding
  - Output is the same size as input (doesn't shrink as the network gets deeper).



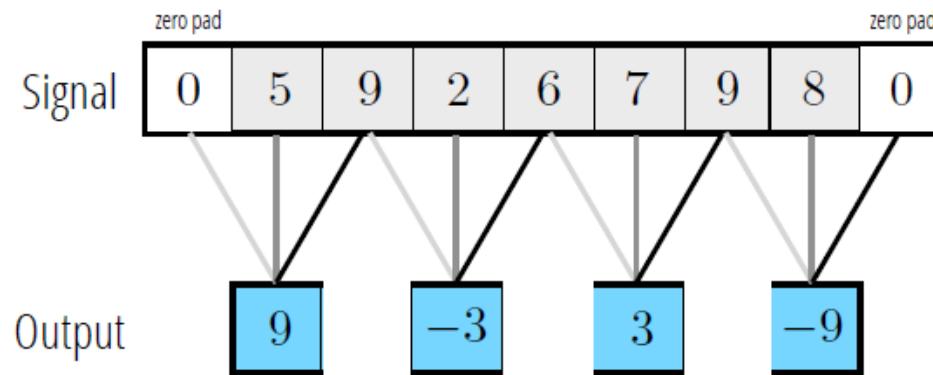
# Variations of Convolution

- Stride
  - Step size across signals



# Variations of Convolution

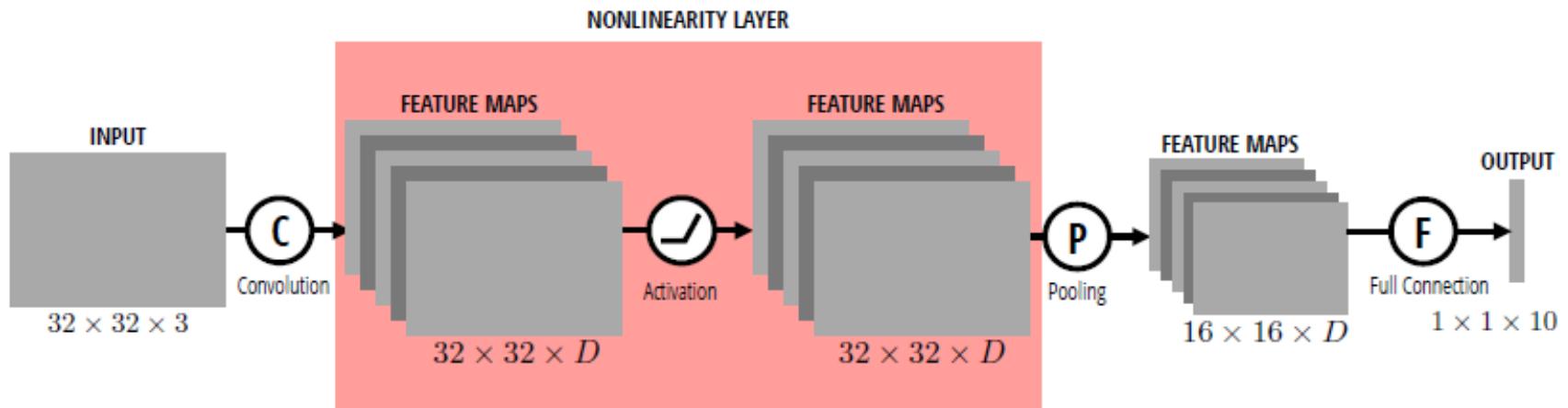
- Stride
  - Step size across signals



$$\frac{N - c}{s} + 1$$

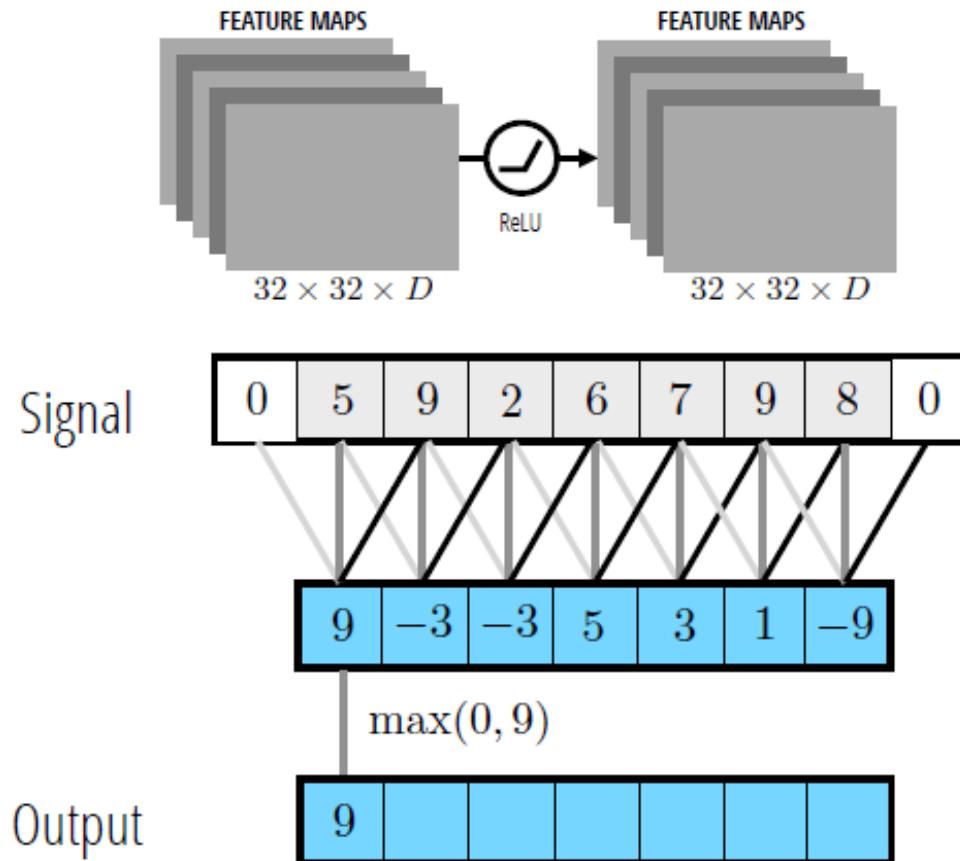
Input Size                  Filter Size  
Output Size  
Stride: step size across the signal

# Nonlinearity Layer in CNN



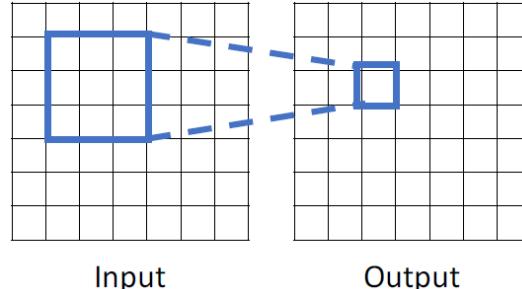
# Nonlinearity Layer

- E.g., ReLU (Rectified Linear Unit)
  - Pixel by pixel computation of  $\max(0, x)$

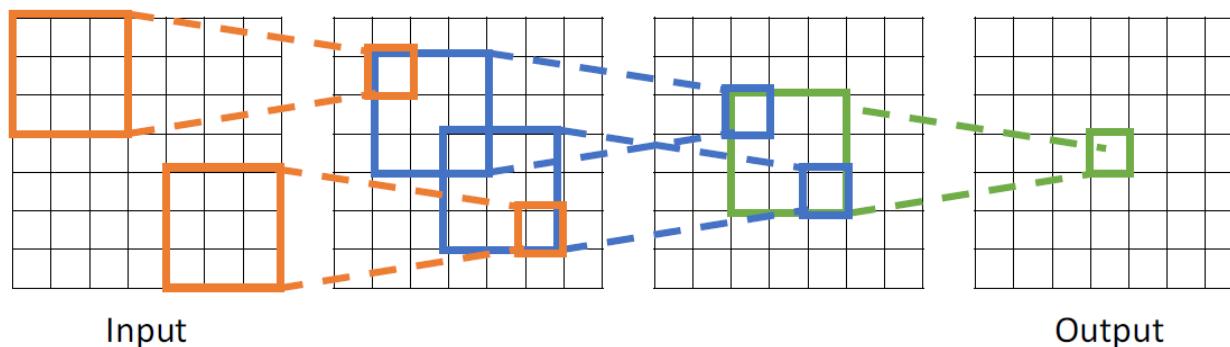


# Receptive Field

- For convolution with kernel size  $n \times n$ , each entry in the output layer depends on a  $n \times n$  receptive field in the input layer.

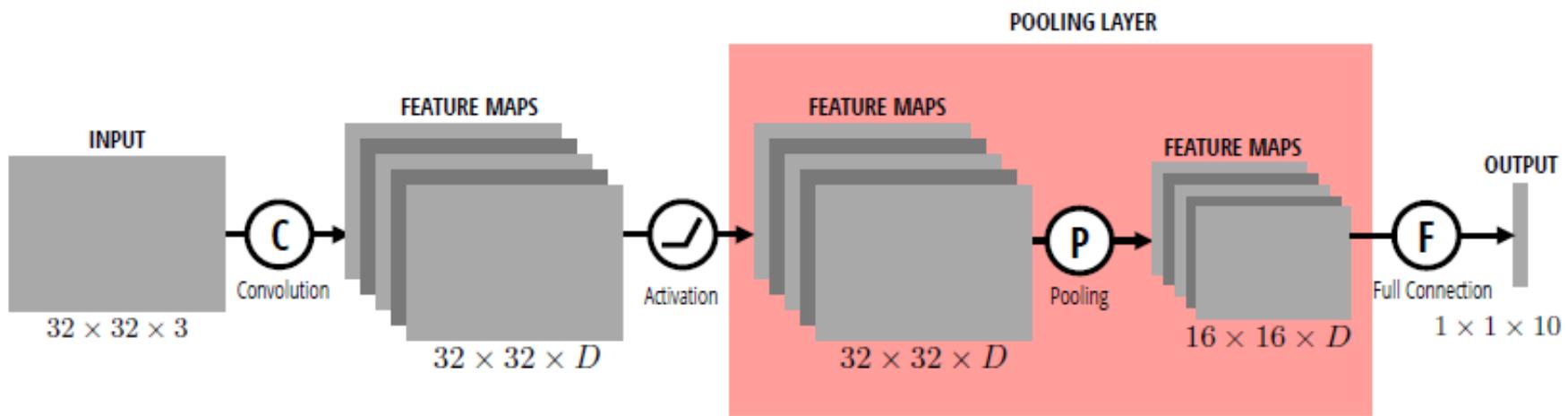


- Each successive convolution adds  $n-1$  to the receptive field size. With a total of  $L$  layers, the receptive field size would be  $1 + L * (n-1)$ .



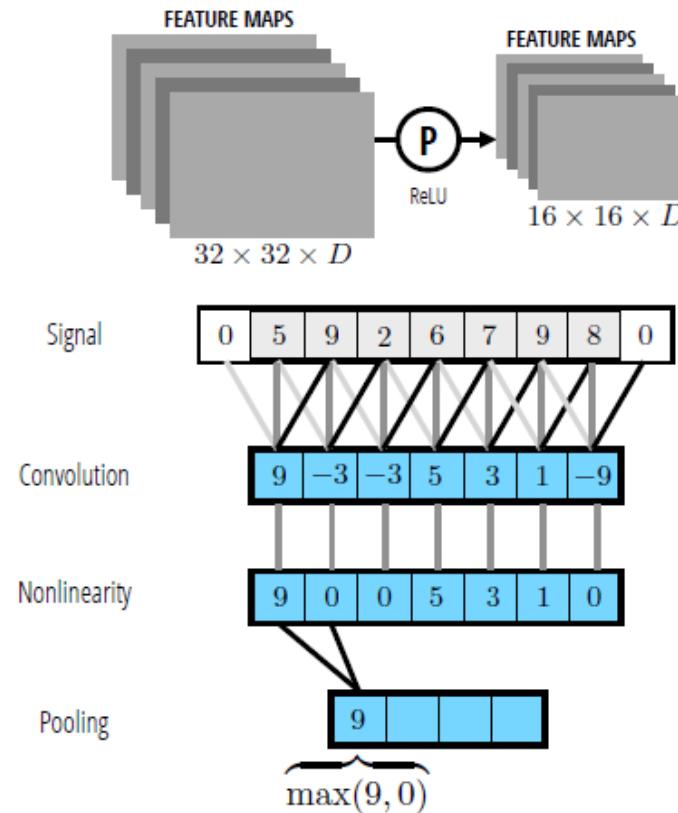
- Thus, for large images, we need many layers for each entry in output to “see” the entire input image. Possible solution → **downsample** the image/feature map (see **pooling layer** next)

# Pooling Layer in CNN



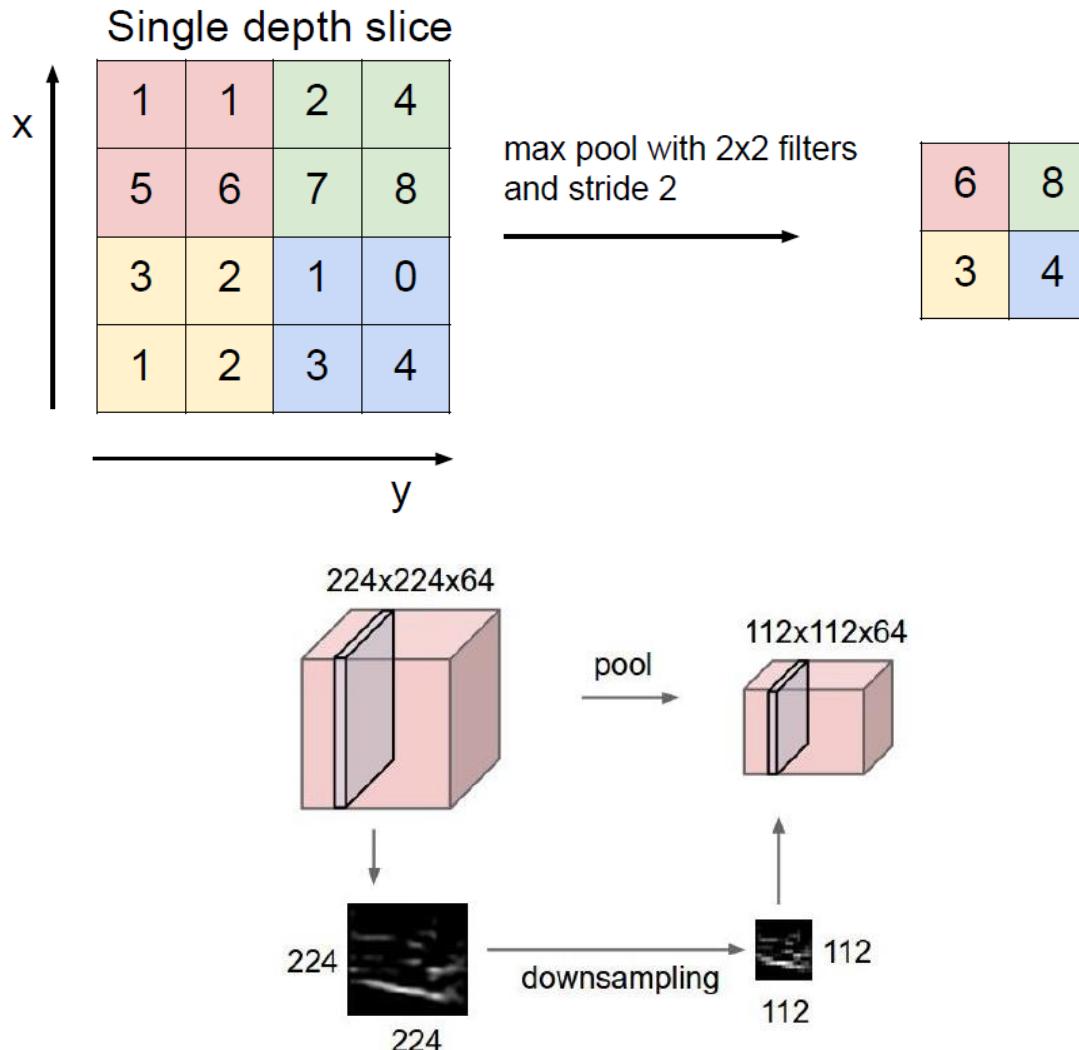
# Pooling Layer

- Makes the representations smaller and more manageable
- Operates over each activation map independently
- E.g., Max Pooling

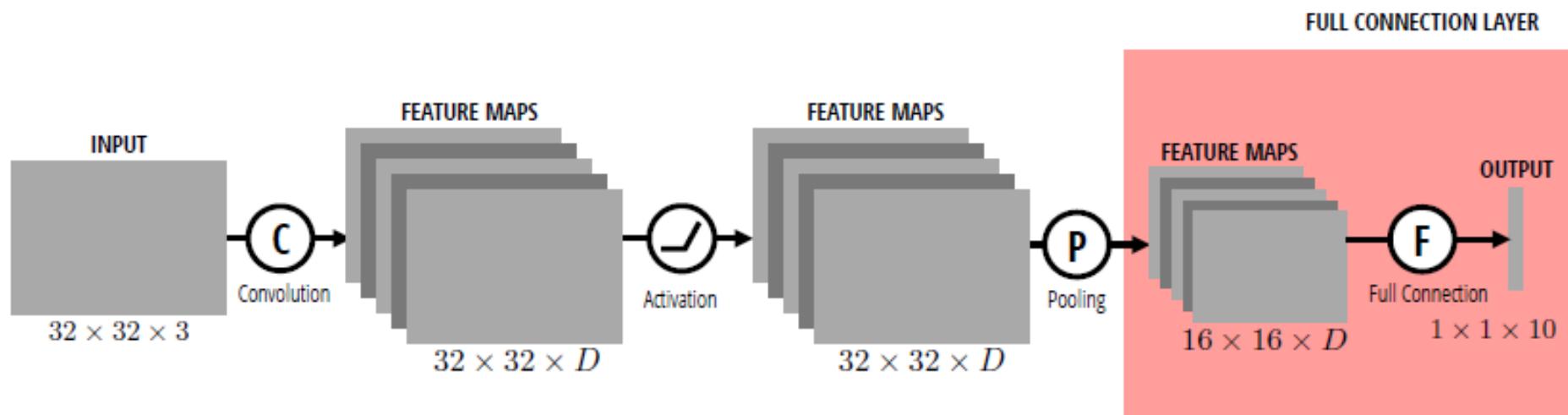


# Pooling Layer for 2D Cases

- Reduces the spatial size and provides spatial invariance

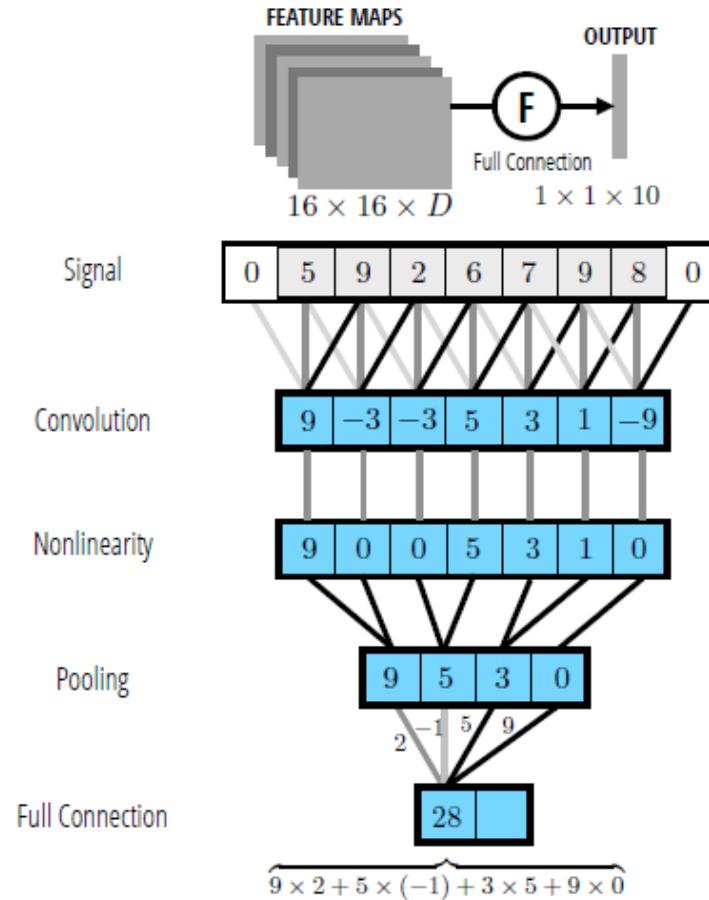


# Fully Connected (FC) Layer in CNN



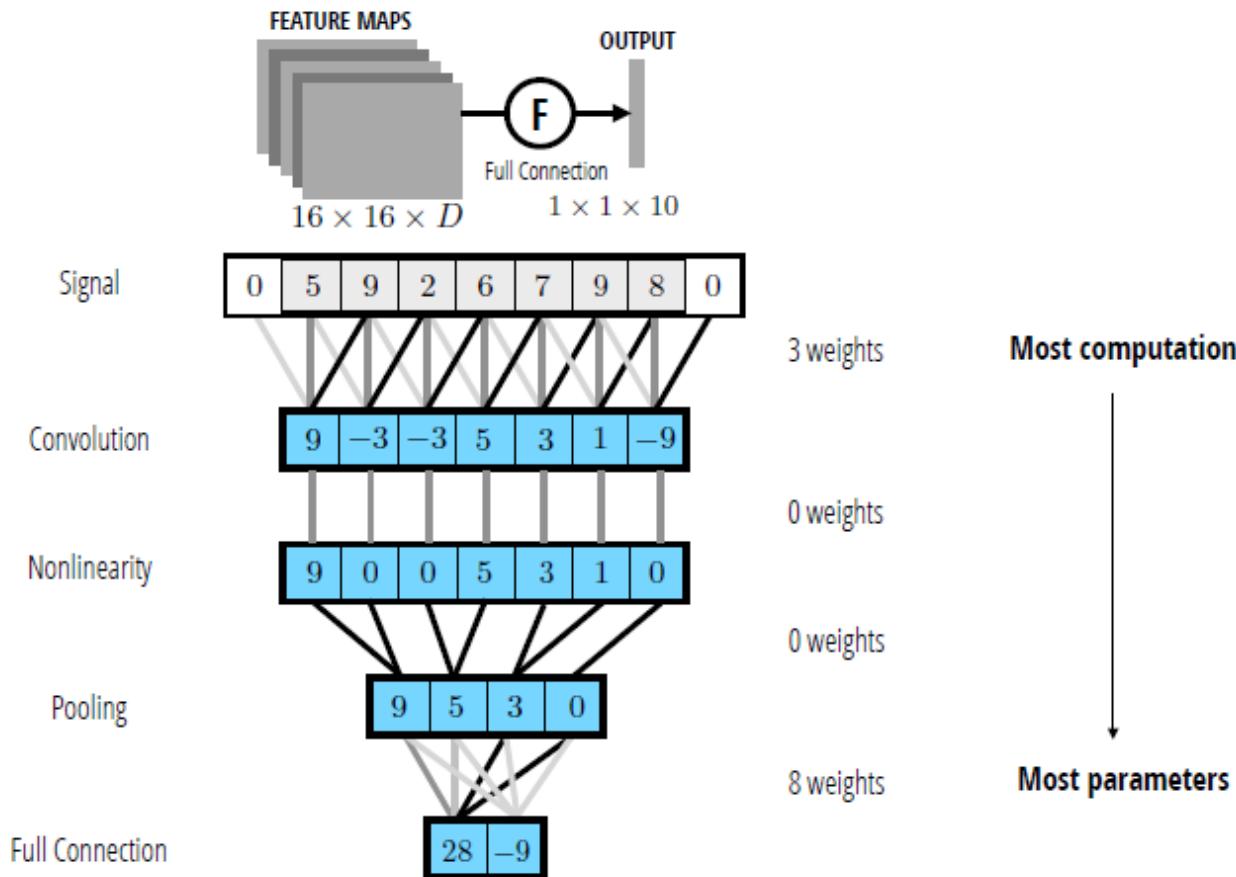
# FC Layer

- Contains neurons that connect to the entire input volume, as in ordinary neural networks



# FC Layer

- Contains neurons that connect to the entire input volume, as in ordinary neural networks



# CNN

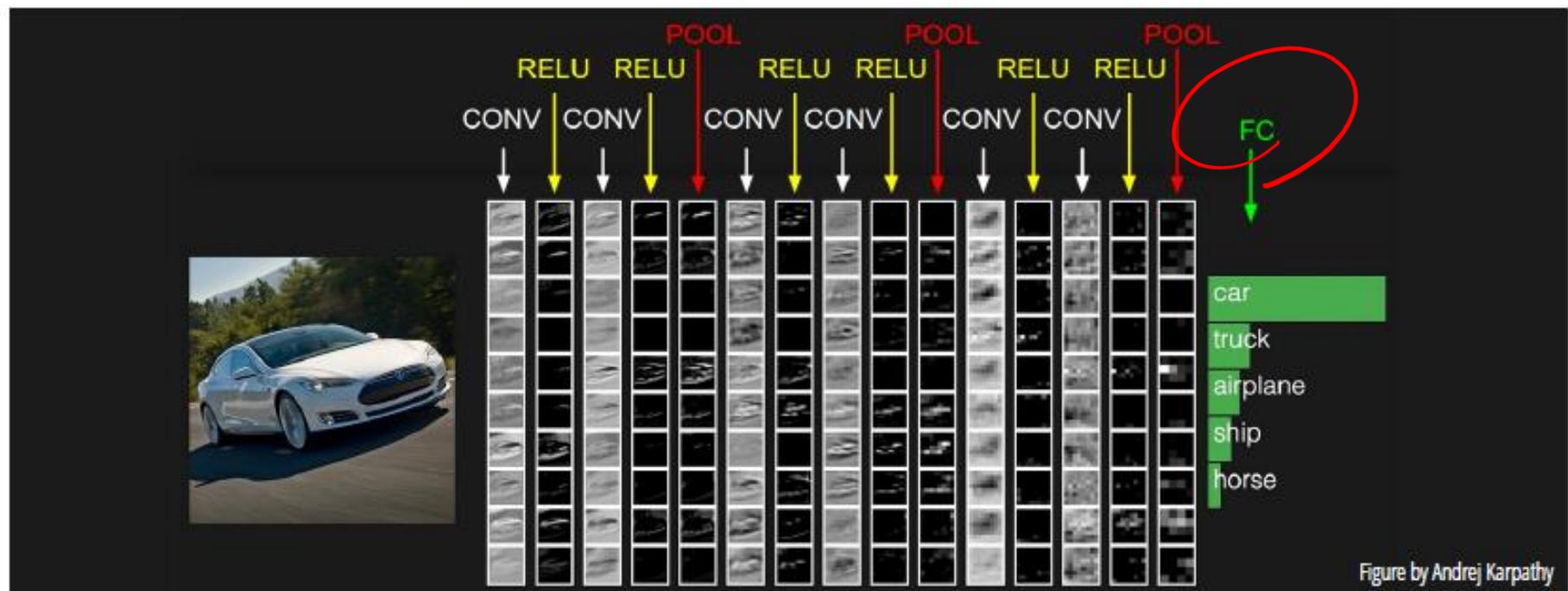
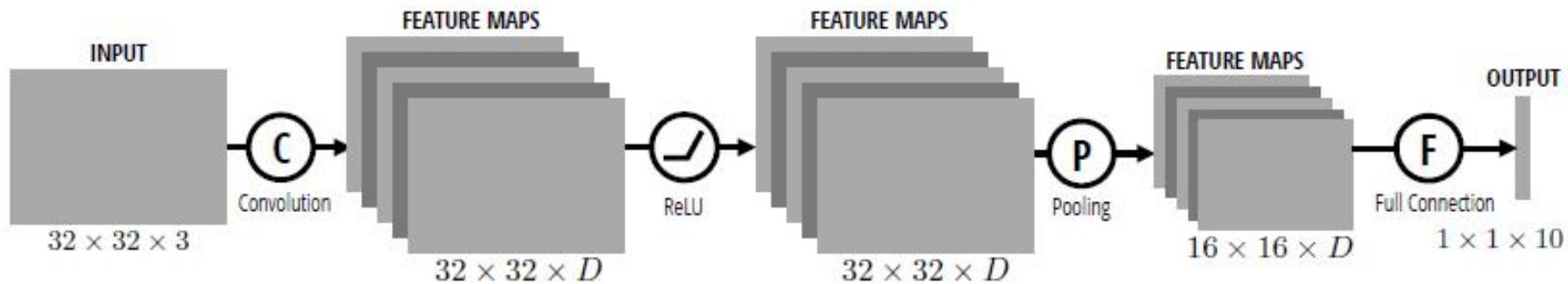
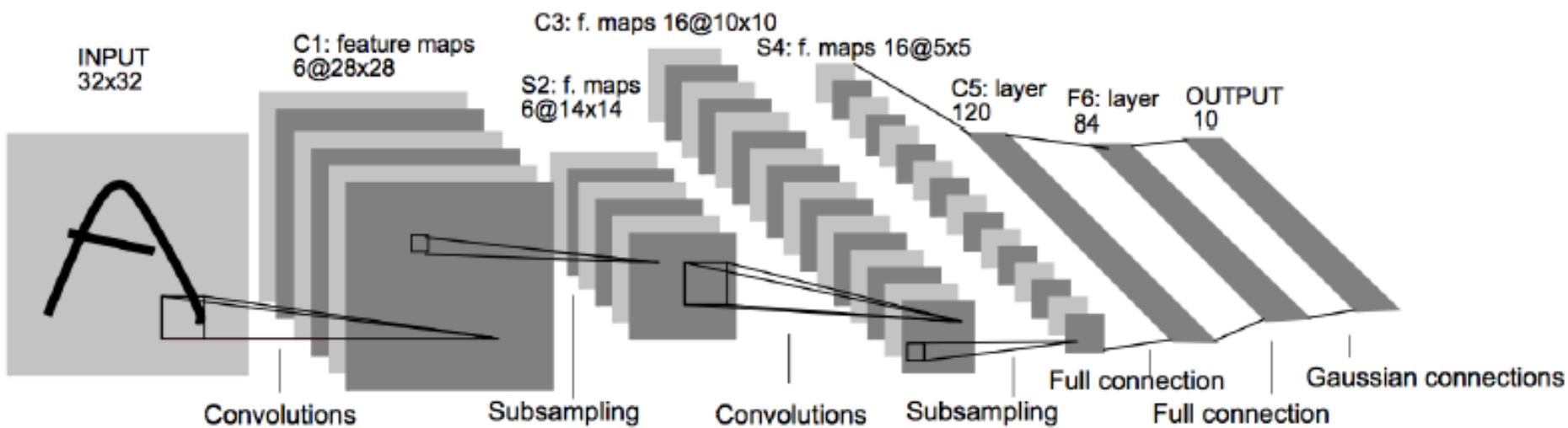


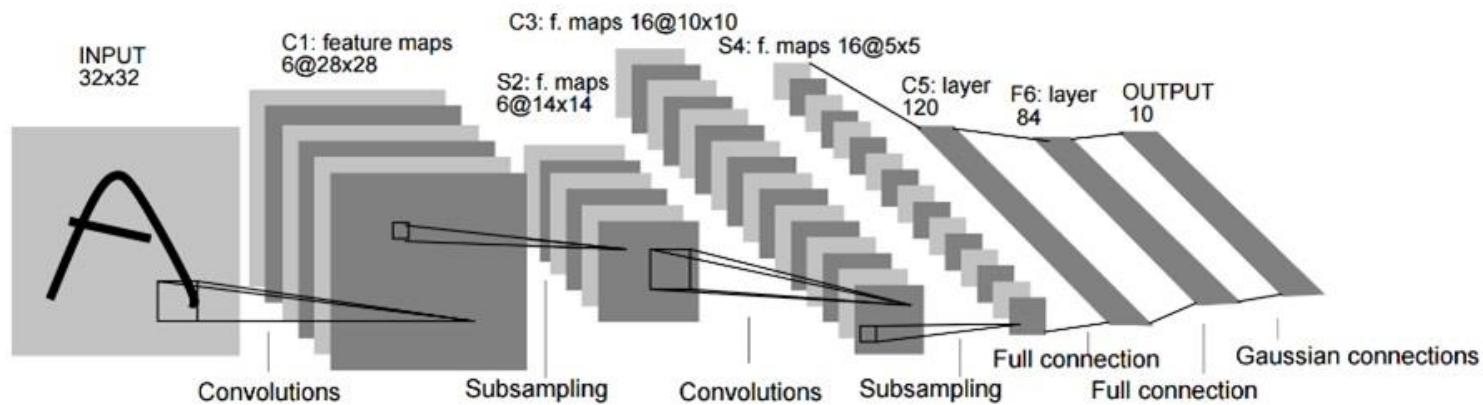
Figure by Andrej Karpathy

# LeNet

- Presented by Yann LeCun during the 1990s for reading digits
- Has the elements of modern architectures



# LeNet [LeCun et al. 1998]



Gradient-based learning applied to document recognition

[LeCun, Bottou, Bengio, Haffner 1998]

# New Driving Forces

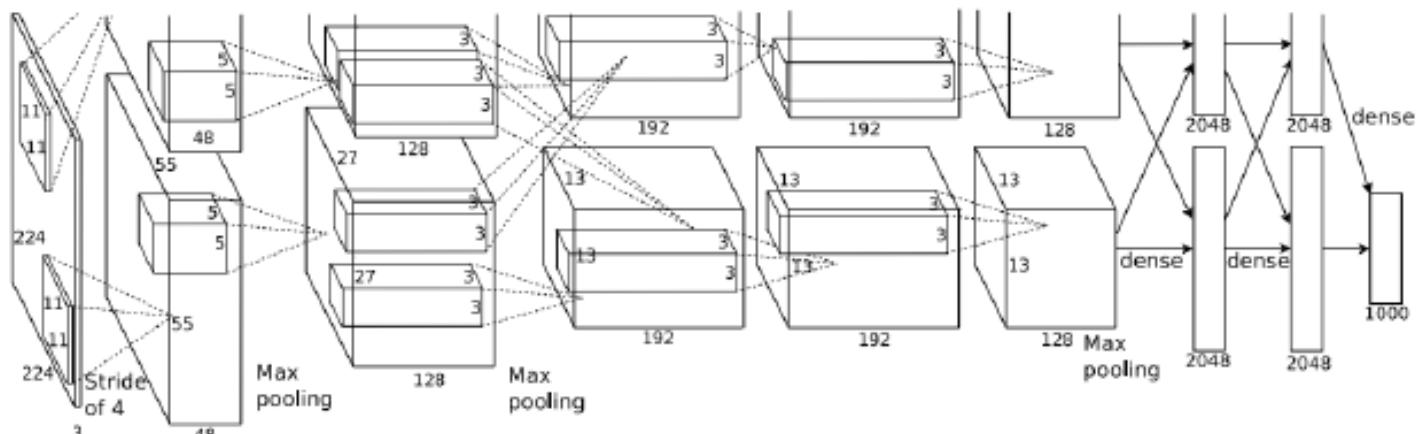
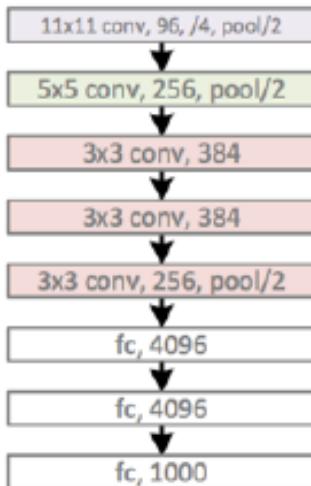
- CPU/GPU computing
  - Personal super computer
- Internet → big data → large datasets become available

# AlexNet [Krizhevsky et al., 2012]

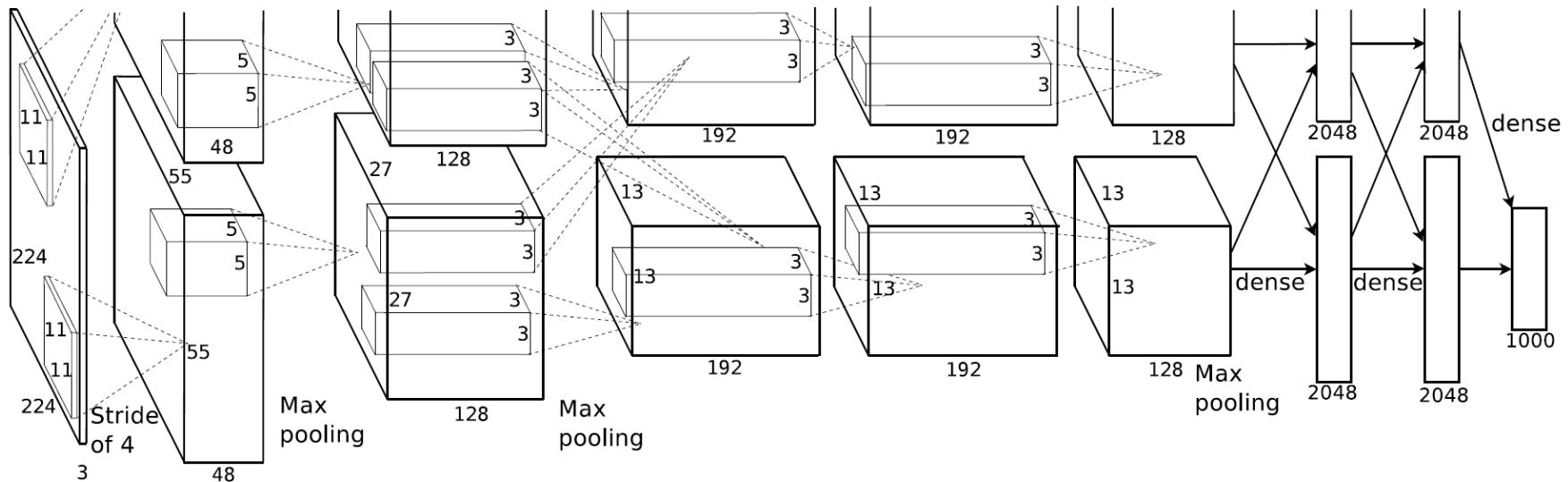
- Repopularized CNN by winning the ImageNet Challenge 2012
- 7 hidden layers, 650,000 neurons, 60M parameters
- Error rate of 16% vs. 26% for 2<sup>nd</sup> place.

Full (simplified) AlexNet architecture:

- [227x227x3] INPUT
- [55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
- [27x27x96] MAX POOL1: 3x3 filters at stride 2
- [27x27x96] NORM1: Normalization layer
- [27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
- [13x13x256] MAX POOL2: 3x3 filters at stride 2
- [13x13x256] NORM2: Normalization layer
- [13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
- [13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
- [13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
- [6x6x256] MAX POOL3: 3x3 filters at stride 2
- [4096] FC6: 4096 neurons
- [4096] FC7: 4096 neurons
- [1000] FC8: 1000 neurons (class scores)



# AlexNet



- Parameters
  - Convolution: 1.89M parameters = 7.56MB
  - Fully connected: 58.62M parameters = 234.49MB
- Computation
  - Convolution: 591M Floating MAC
  - Fully connected: 58.62M Floating MAC
  - Full-HD 30fps: 805 GFLOPS (no overlap)

# Deep or Not?

- Depth of the network is critical for performance.



**AlexNet:** 8 Layers with 18.2% top-5 error

**Removing Layer 7** reduces 16 million parameters, but only 1.1% drop in performance!

**Removing Layer 6 and 7** reduces 50 million parameters, but only 5.7% drop in performance

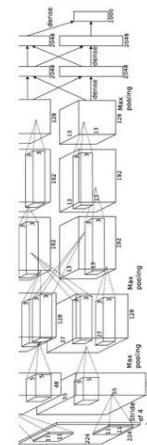
**Removing middle conv layers** reduces 1 million parameters, but only 3% drop in performance

**Removing feature & conv layers** produces a **33% drop** in performance

# Ultra Deep Network

[http://cs231n.stanford.edu/slides/winter1516\\_lecuture8.pdf](http://cs231n.stanford.edu/slides/winter1516_lecuture8.pdf)

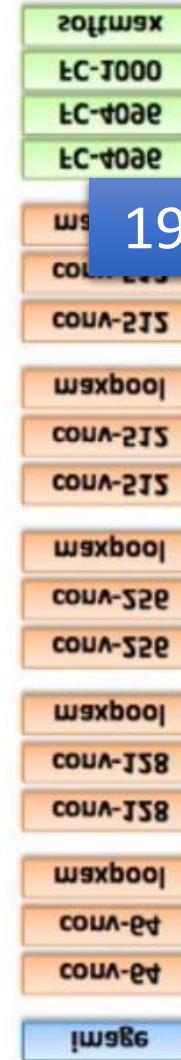
16.4%



AlexNet (2012)

8 layers

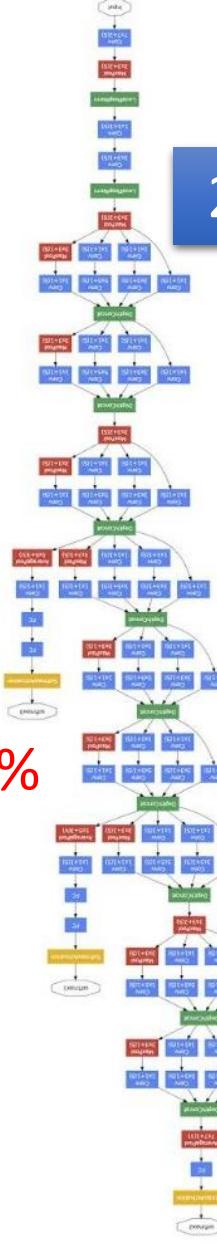
7.3%



VGG (2014)

19 layers

6.7%



GoogleNet (2014)

22 layers

# Ultra Deep Network

[Prof. H.-Y. Lee]

152 layers

101 layers

This ultra deep network  
have special structure.

3.57%

16.4%

7.3%

6.7%

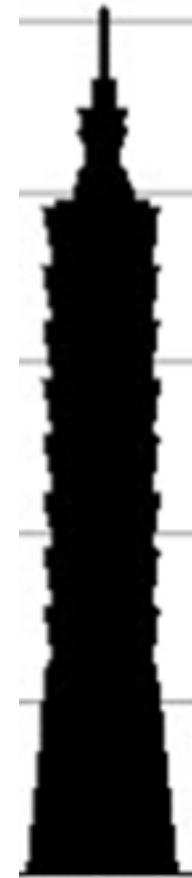
AlexNet  
(2012)

VGG  
(2014)

GoogleNet  
(2014)

Residual Net  
(2015)

Taipei  
101



# VGG (2014)

- Parameters:

- Convolution: ~14M, 56MB
- Fully connected: ~124M, 496MB

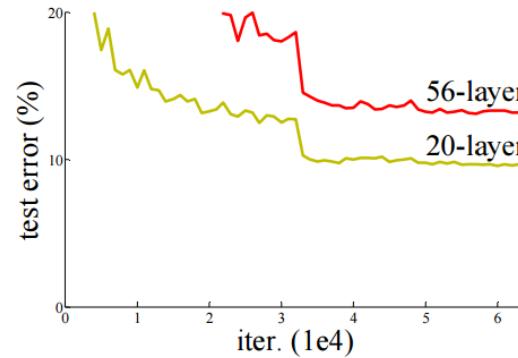
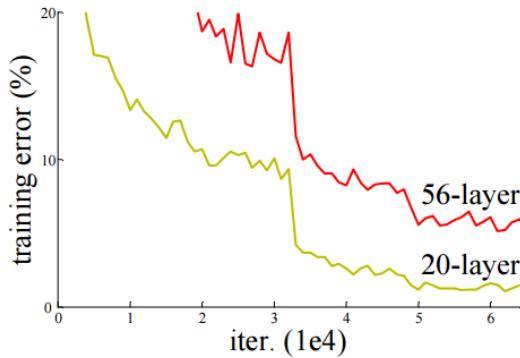
- Computation:

- Convolution: 15.52G Floating MAC
- Fully connected: 123.63M Floating MAC
- Full-HD 30fps: 19.3TFLOPS(no overlap)

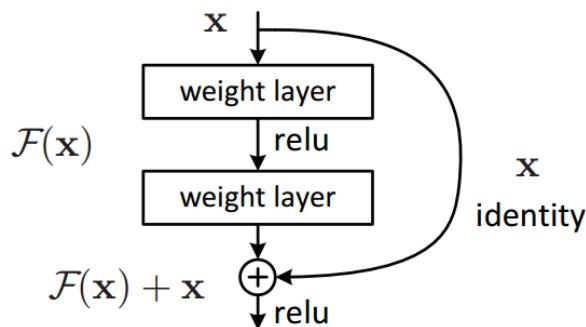


# ResNet (2016)

- Can we just increase the #layer?



- How can we train very deep network?
  - Residual learning

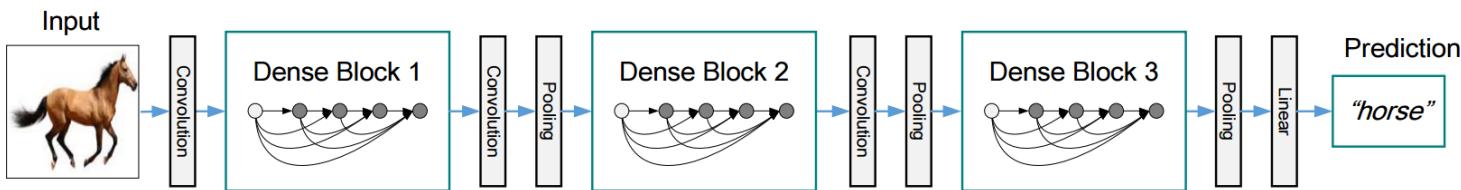
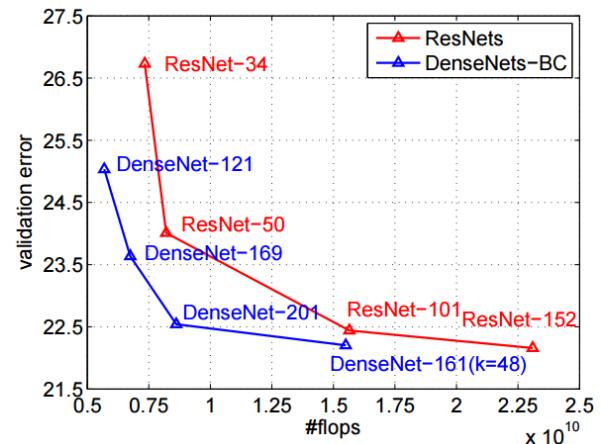
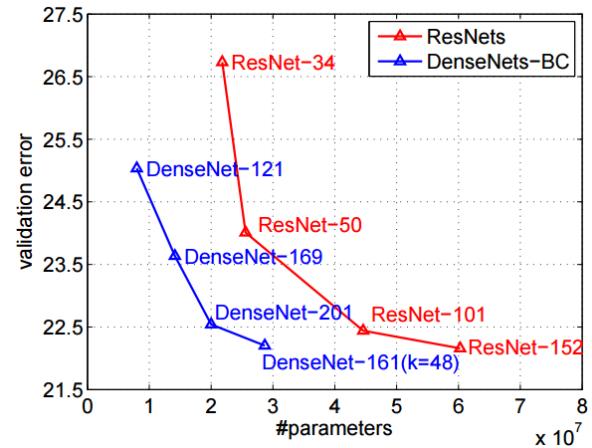
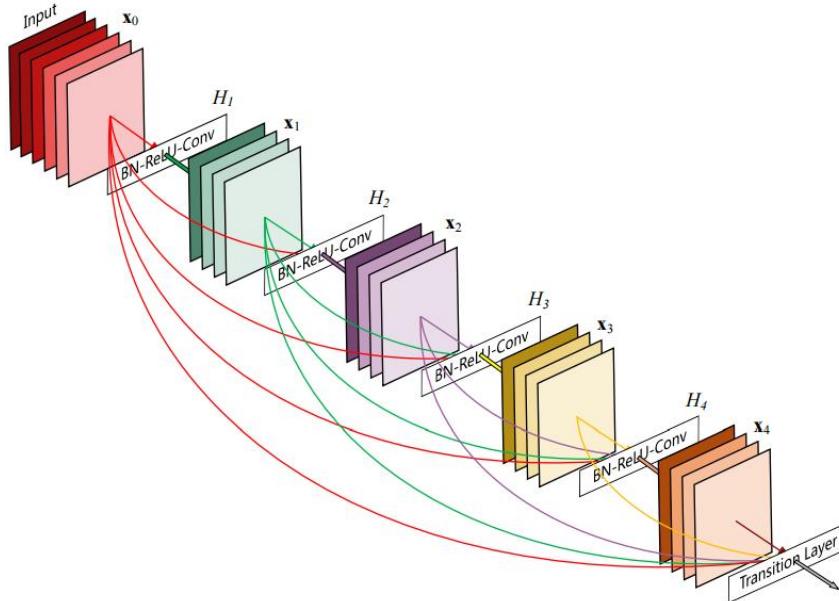


method	top-5 err. (test)
VGG [41] (ILSVRC'14)	7.32
GoogLeNet [44] (ILSVRC'14)	6.66
VGG [41] (v5)	6.8
PReLU-net [13]	4.94
BN-inception [16]	4.82
<b>ResNet (ILSVRC'15)</b>	<b>3.57</b>

Ref: He, Kaiming, et al. "Deep residual learning for image recognition." CVPR, 2016.

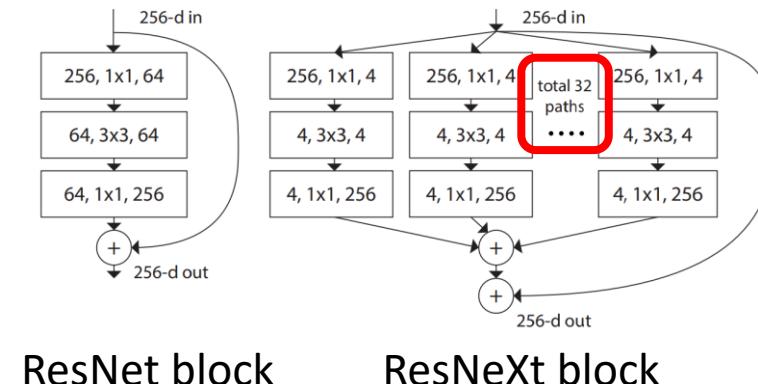
# DenseNet (2017)

- Shorter connections (like ResNet) help
- Why not just connect them all?

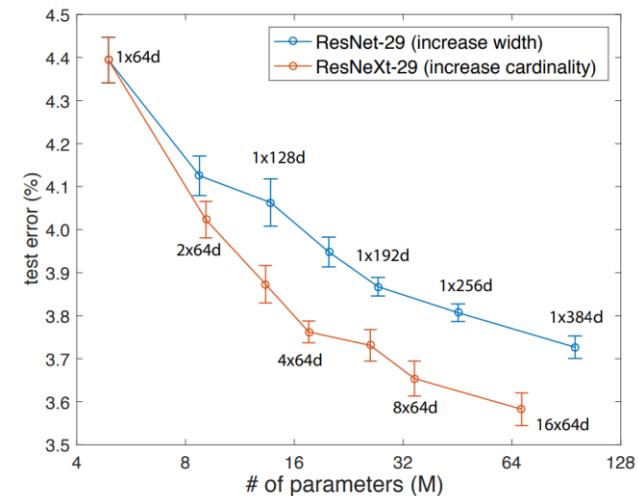


# ResNeXT (2017)

- Deeper and wider → better...what else?
  - Increase cardinality

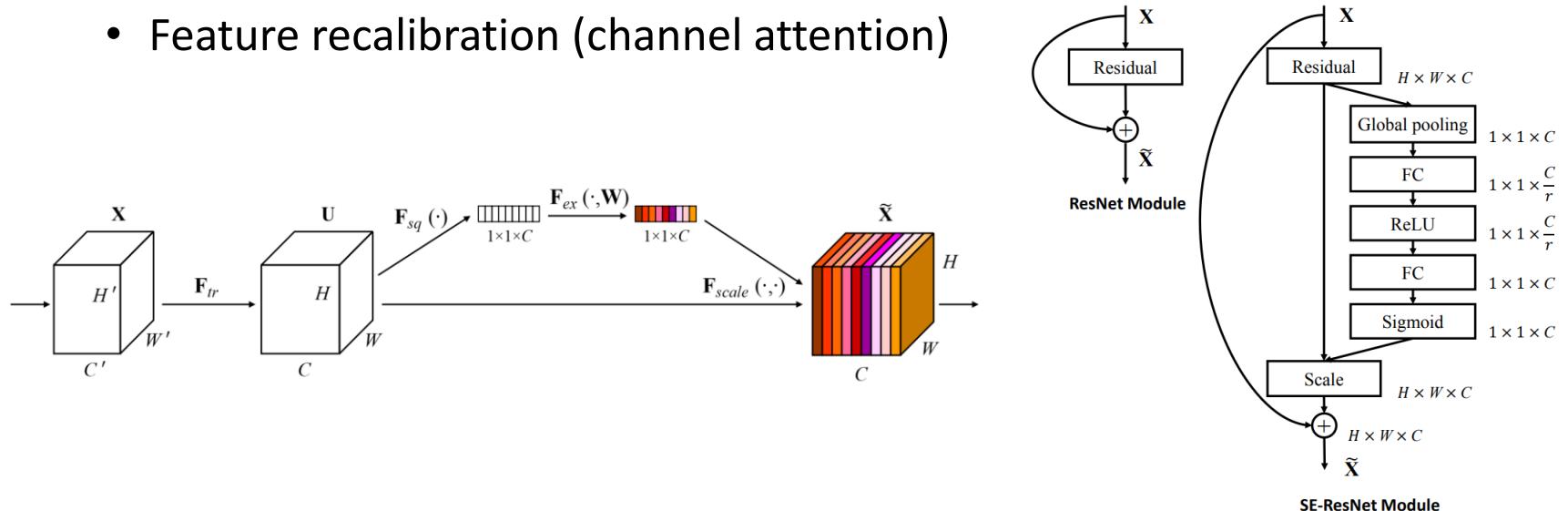


	setting	top-1 error (%)
ResNet-50	$1 \times 64d$	23.9
ResNeXt-50	$2 \times 40d$	23.0
ResNeXt-50	$4 \times 24d$	22.6
ResNeXt-50	$8 \times 14d$	22.3
ResNeXt-50	$32 \times 4d$	<b>22.2</b>
ResNet-101	$1 \times 64d$	22.0
ResNeXt-101	$2 \times 40d$	21.7
ResNeXt-101	$4 \times 24d$	21.4
ResNeXt-101	$8 \times 14d$	21.3
ResNeXt-101	$32 \times 4d$	<b>21.2</b>



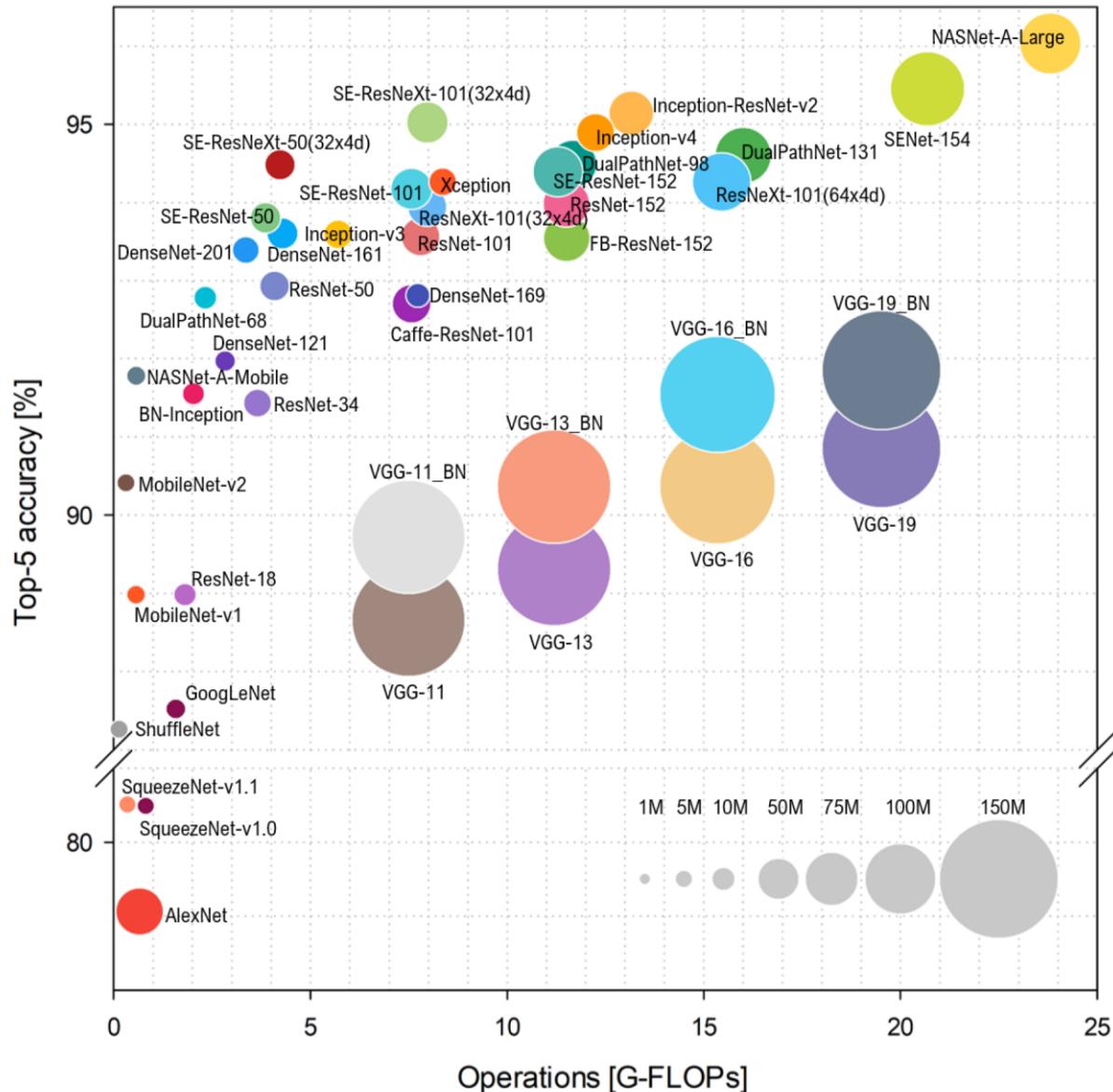
# Squeeze-and-Excitation Net (SENet)

- How to improve acc. without much overhead?
  - Feature recalibration (channel attention)



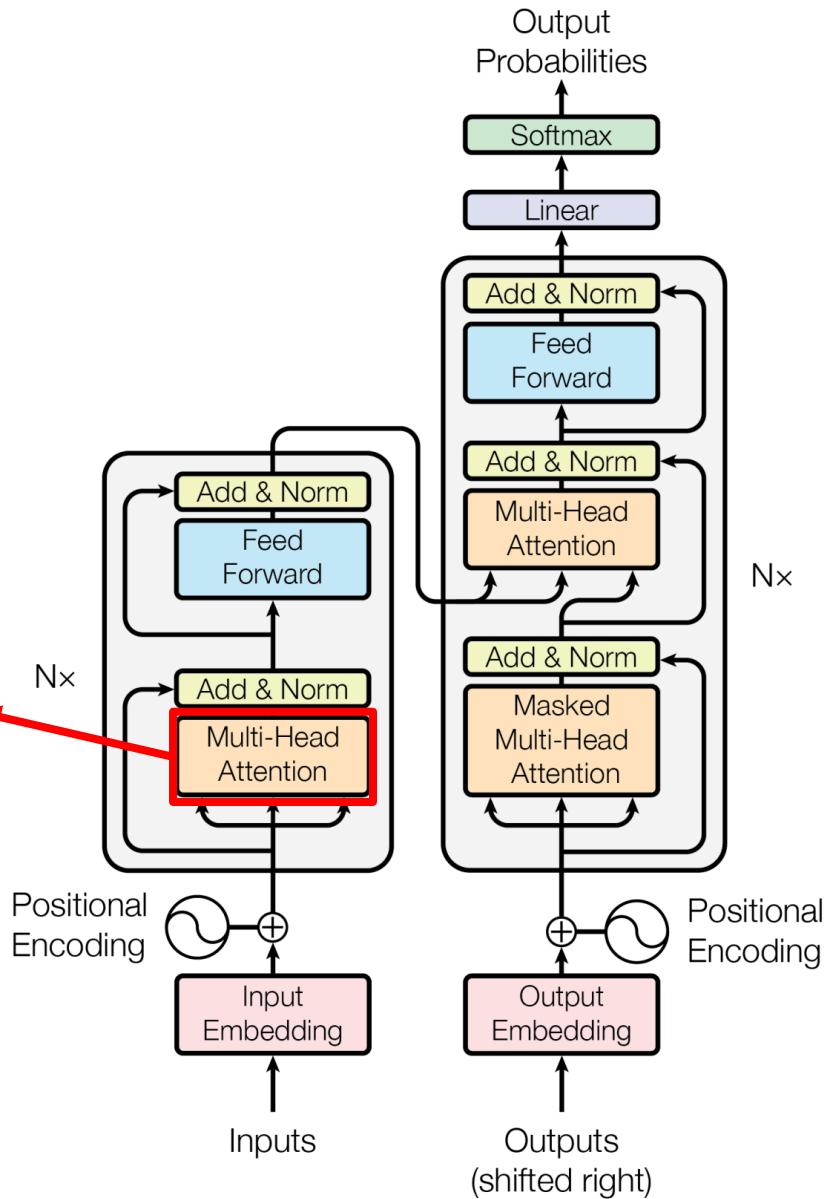
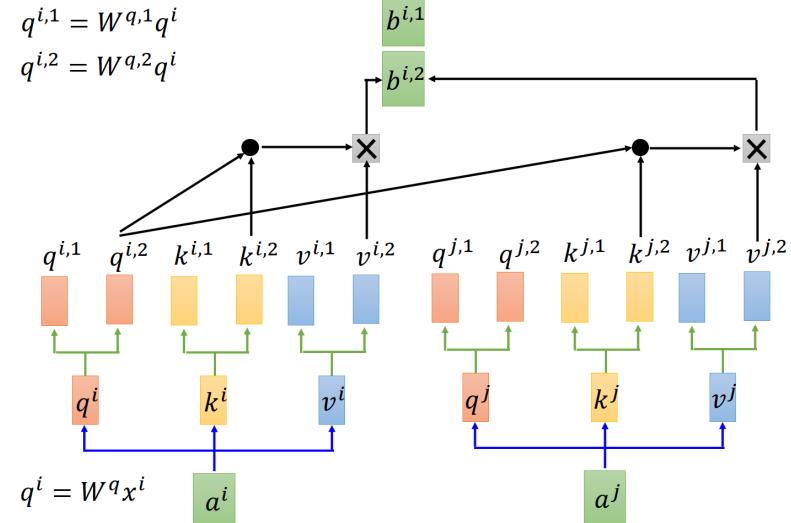
	original		re-implementation			SENet		
	top-1 err.	top-5 err.	top-1 err.	top-5 err.	GFLOPs	top-1 err.	top-5 err.	GFLOPs
ResNet-50 [13]	24.7	7.8	24.80	7.48	3.86	23.29 <sub>(1.51)</sub>	6.62 <sub>(0.86)</sub>	3.87
ResNet-101 [13]	23.6	7.1	23.17	6.52	7.58	22.38 <sub>(0.79)</sub>	6.07 <sub>(0.45)</sub>	7.60
ResNet-152 [13]	23.0	6.7	22.42	6.34	11.30	21.57 <sub>(0.85)</sub>	5.73 <sub>(0.61)</sub>	11.32
ResNeXt-50 [19]	22.2	-	22.11	5.90	4.24	21.10 <sub>(1.01)</sub>	5.49 <sub>(0.41)</sub>	4.25
ResNeXt-101 [19]	21.2	5.6	21.18	5.57	7.99	20.70 <sub>(0.48)</sub>	5.01 <sub>(0.56)</sub>	8.00
VGG-16 [11]	-	-	27.02	8.81	15.47	25.22 <sub>(1.80)</sub>	7.70 <sub>(1.11)</sub>	15.48
BN-Inception [6]	25.2	7.82	25.38	7.89	2.03	24.23 <sub>(1.15)</sub>	7.14 <sub>(0.75)</sub>	2.04
Inception-ResNet-v2 [21]	19.9 <sup>†</sup>	4.9 <sup>†</sup>	20.37	5.21	11.75	19.80 <sub>(0.57)</sub>	4.79 <sub>(0.42)</sub>	11.76

# Various Deep Learning Models...



# Transformer

## Multi-head Self-attention

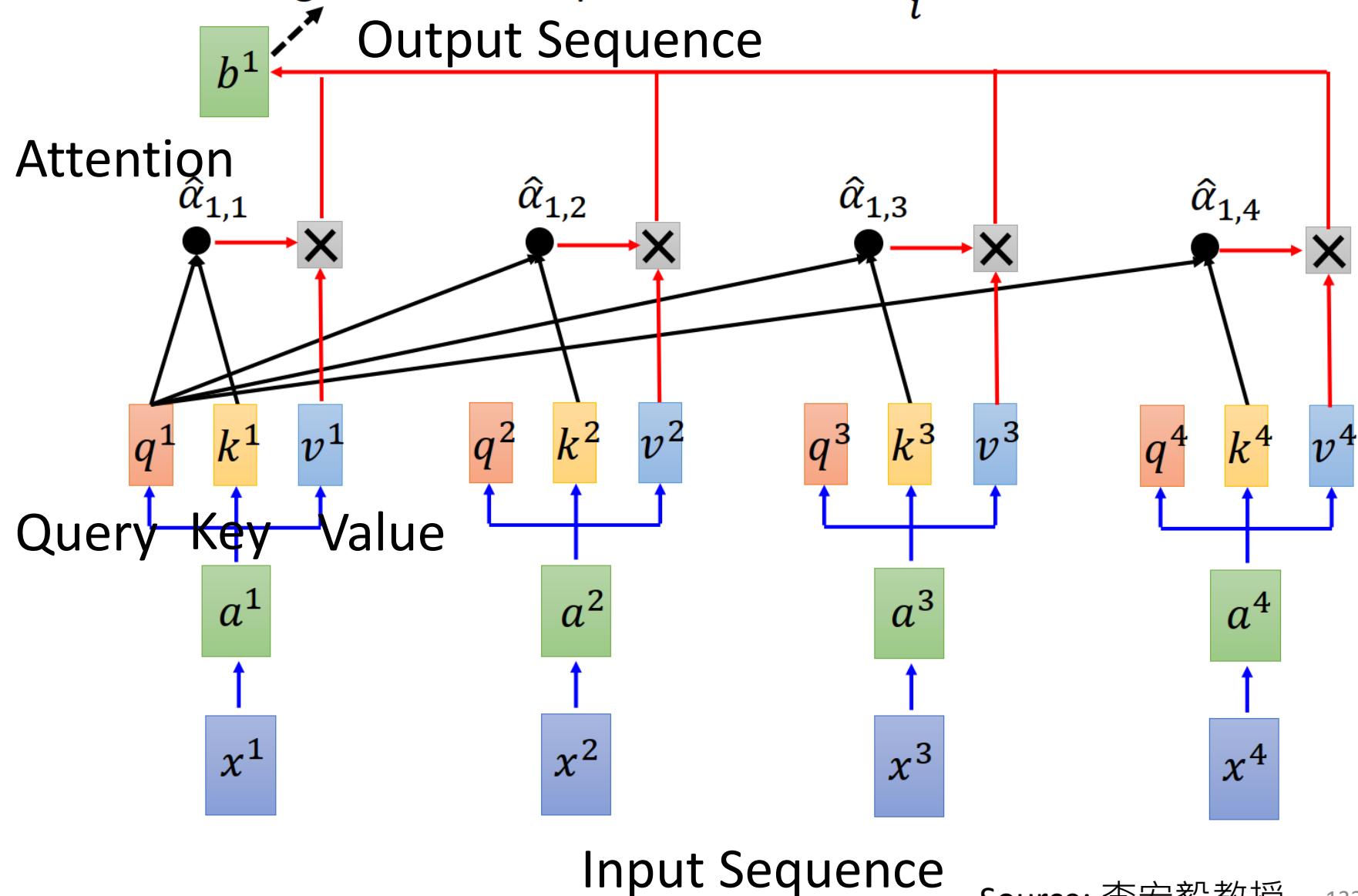


Ref: Vaswani et al., "Attention Is All You Need," NIPS2017.

## Self-attention

Considering the whole sequence

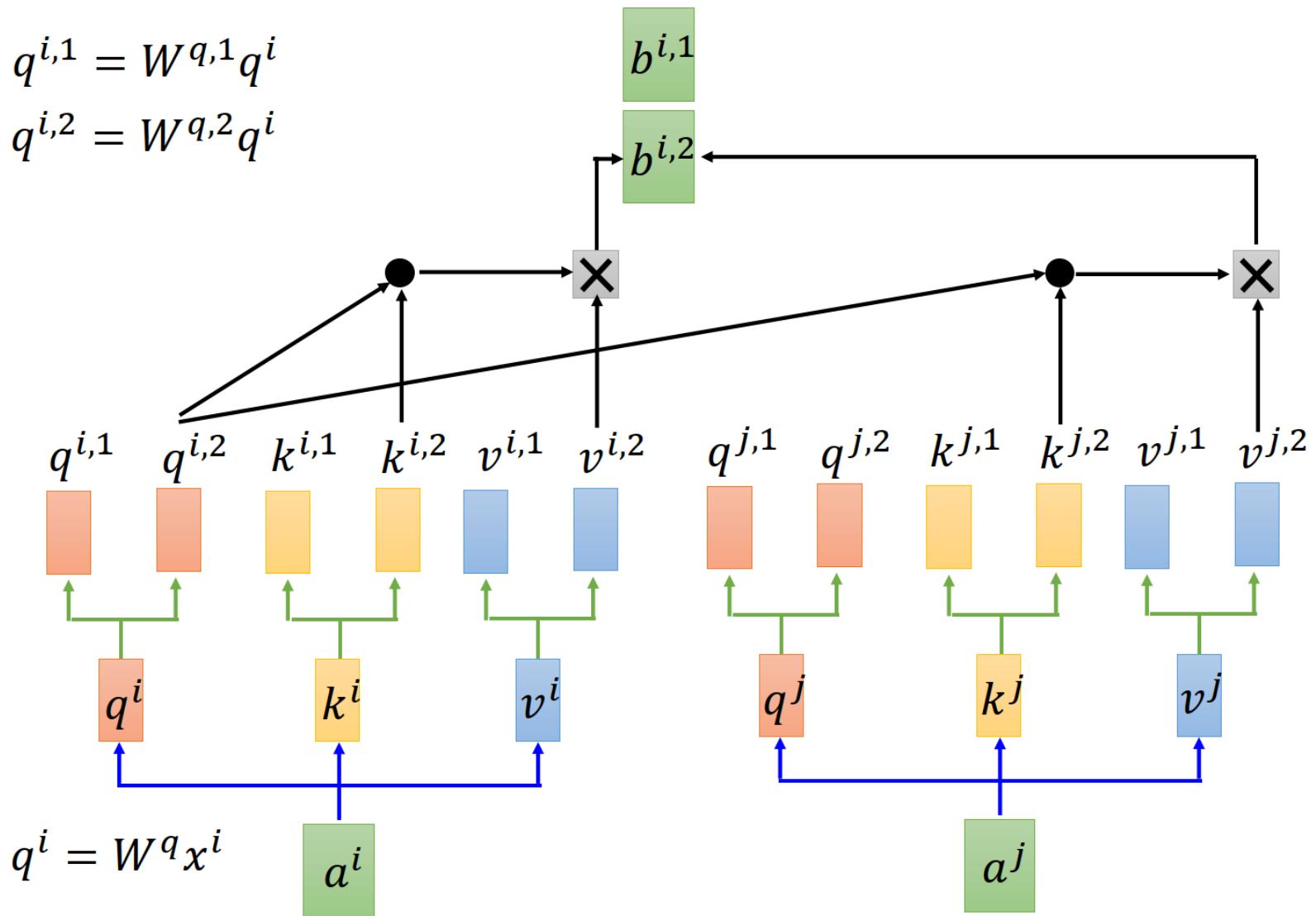
$$b^1 = \sum_i \hat{\alpha}_{1,i} v^i$$



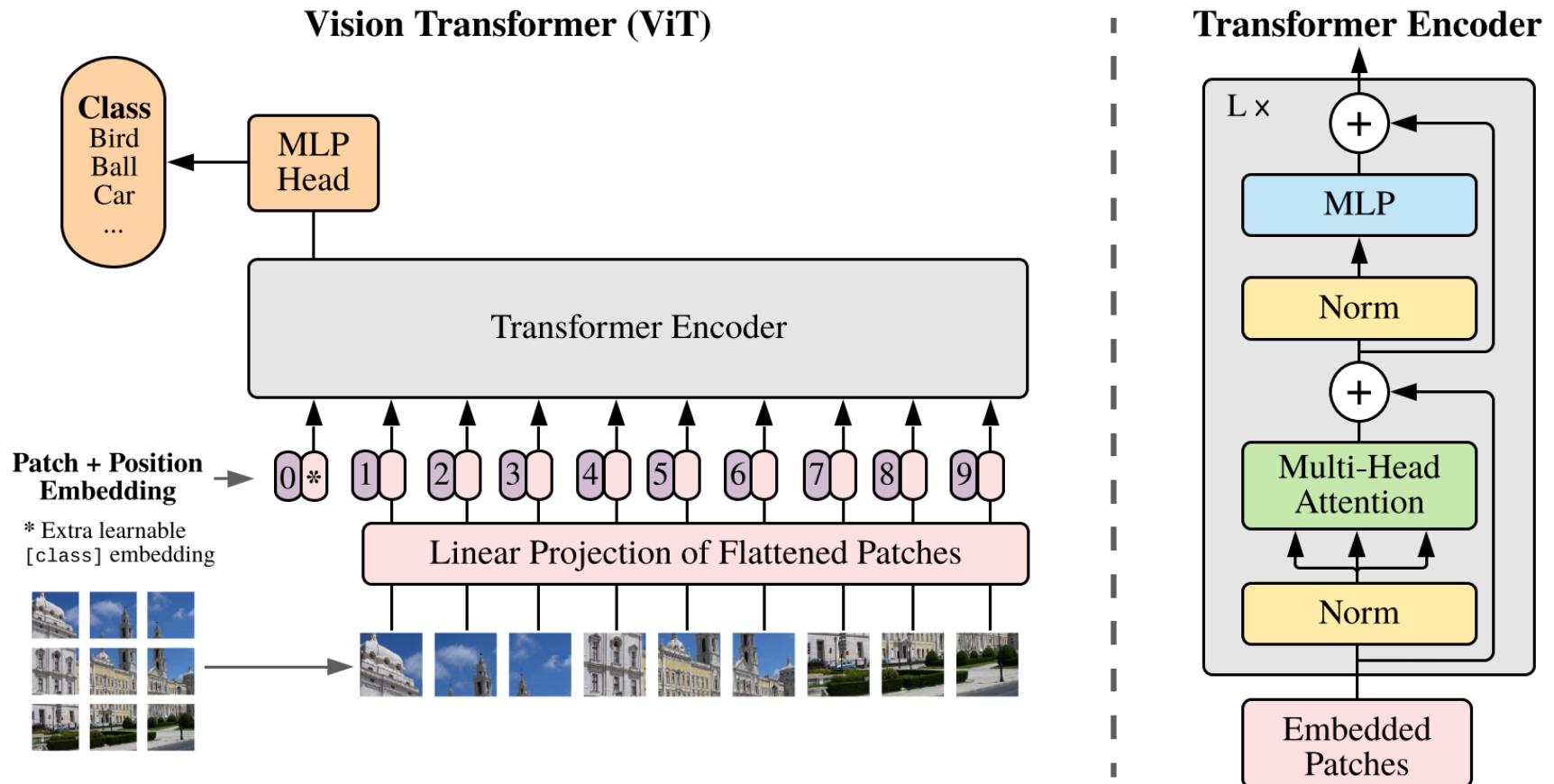
# **Multi-head Self-attention** (2 heads as example)

$$q^{i,1} = W^{q,1} q^i$$

$$q^{i,2} = W^{q,2} q^i$$



# Visual Transformer (ViT)



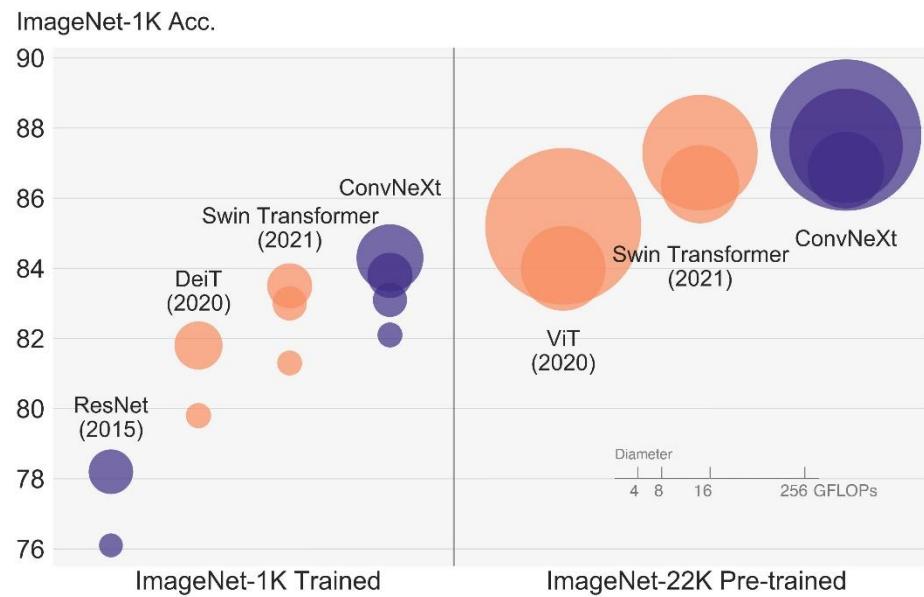
Ref: Dosovitskiy et al., "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," *ICLR2021*.

# Visual Transformer (ViT)

(a) Regular ImageNet-1K trained models					
method	image size	#param.	FLOPs	throughput (image / s)	ImageNet top-1 acc.
RegNetY-4G [48]	224 <sup>2</sup>	21M	4.0G	1156.7	80.0
RegNetY-8G [48]	224 <sup>2</sup>	39M	8.0G	591.6	81.7
RegNetY-16G [48]	224 <sup>2</sup>	84M	16.0G	334.7	82.9
EffNet-B3 [58]	300 <sup>2</sup>	12M	1.8G	732.1	81.6
EffNet-B4 [58]	380 <sup>2</sup>	19M	4.2G	349.4	82.9
EffNet-B5 [58]	456 <sup>2</sup>	30M	9.9G	169.1	83.6
EffNet-B6 [58]	528 <sup>2</sup>	43M	19.0G	96.9	84.0
EffNet-B7 [58]	600 <sup>2</sup>	66M	37.0G	55.1	84.3
ViT-B/16 [20]	384 <sup>2</sup>	86M	55.4G	85.9	77.9
ViT-L/16 [20]	384 <sup>2</sup>	307M	190.7G	27.3	76.5
DeiT-S [63]	224 <sup>2</sup>	22M	4.6G	940.4	79.8
DeiT-B [63]	224 <sup>2</sup>	86M	17.5G	292.3	81.8
DeiT-B [63]	384 <sup>2</sup>	86M	55.4G	85.9	83.1
Swin-T	224 <sup>2</sup>	29M	4.5G	755.2	81.3
Swin-S	224 <sup>2</sup>	50M	8.7G	436.9	83.0
Swin-B	224 <sup>2</sup>	88M	15.4G	278.1	83.5
Swin-B	384 <sup>2</sup>	88M	47.0G	84.7	84.5

(b) ImageNet-22K pre-trained models					
method	image size	#param.	FLOPs	throughput (image / s)	ImageNet top-1 acc.
R-101x3 [38]	384 <sup>2</sup>	388M	204.6G	-	84.4
R-152x4 [38]	480 <sup>2</sup>	937M	840.5G	-	85.4
ViT-B/16 [20]	384 <sup>2</sup>	86M	55.4G	85.9	84.0
ViT-L/16 [20]	384 <sup>2</sup>	307M	190.7G	27.3	85.2
Swin-B	224 <sup>2</sup>	88M	15.4G	278.1	85.2
Swin-B	384 <sup>2</sup>	88M	47.0G	84.7	86.4
Swin-L	384 <sup>2</sup>	197M	103.9G	42.1	87.3

Table 1. Comparison of different backbones on ImageNet-1K classification. Throughput is measured using the GitHub repository of [68] and a V100 GPU, following [63].



Ref: Liu et al., “A ConvNet for the 2020s,” CVPR2022.

Ref: Liu et al., “Swin Transformer: Hierarchical Vision Transformer using Shifted Windows,” ICCV2021.