



Deep Learning Basics

簡韶逸 Shao-Yi Chien

Department of Electrical Engineering

National Taiwan University

Spring 2021

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]

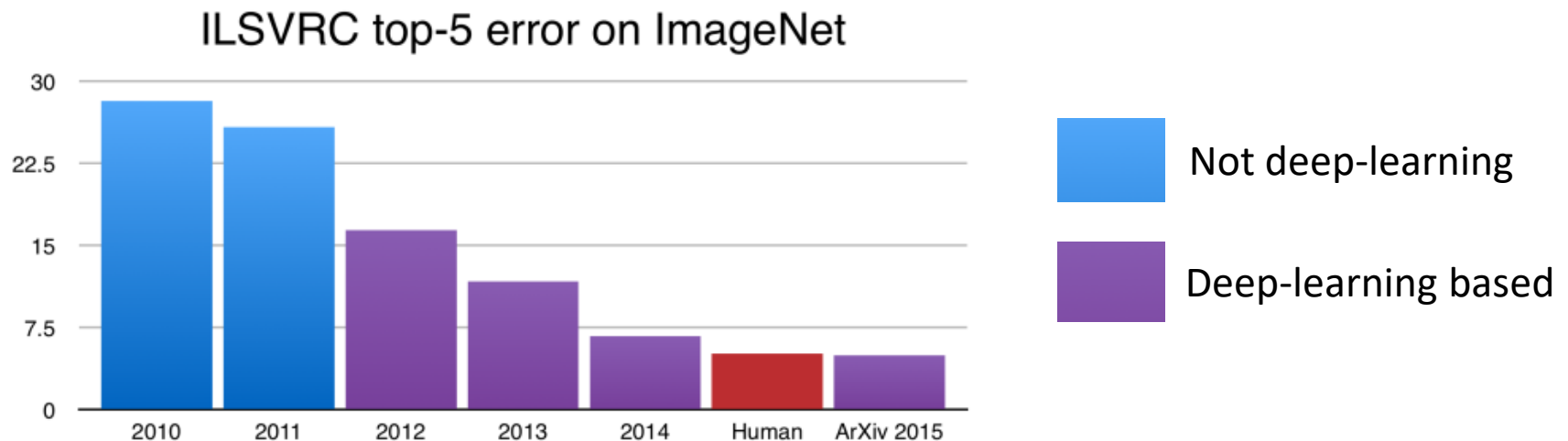
- 1958: Perceptron (linear model)
- 1969: Perceptron has limitation
- 1980s: Multi layer perceptron
 - Do not have significant difference from DNN today
- 1986: Backpropagation
 - Usually more than 3 hidden layers is not helpful
- 1989: 1 hidden layer is “good enough”, why deep?
- 2006: RBM initialization (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

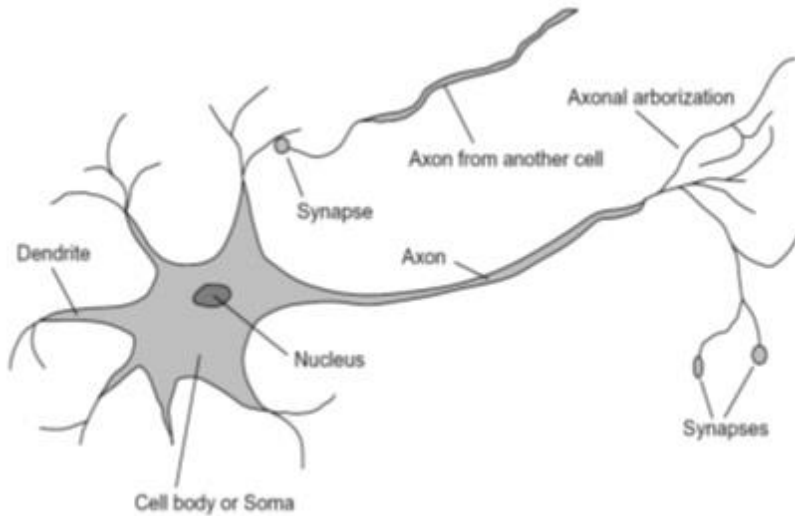


Source:

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

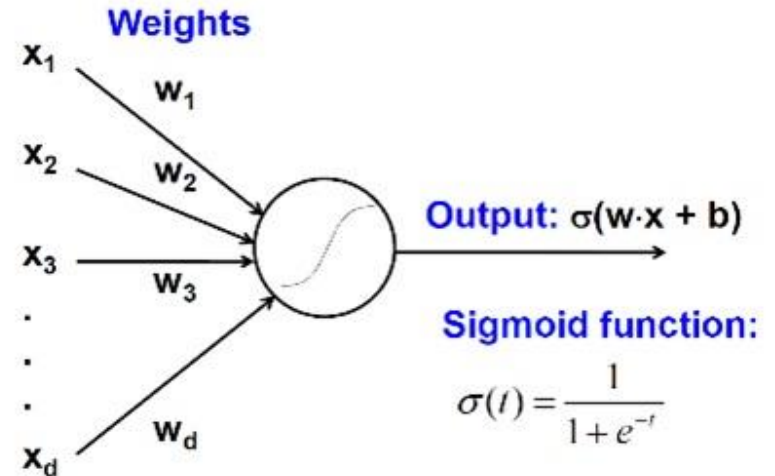
<https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/>

Biological neuron and Perceptrons



A biological neuron

Input



An artificial neuron (Perceptron)
- a linear classifier



Simple, Complex and Hypercomplex cells

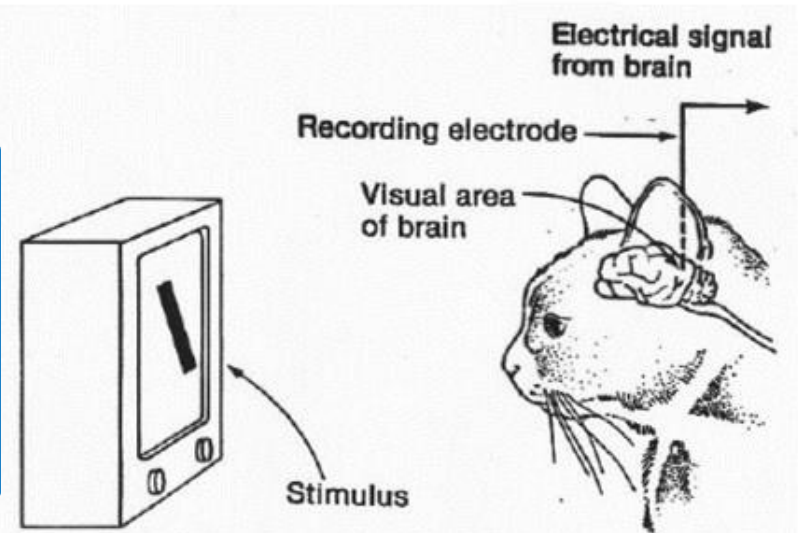


Suggested a **hierarchy of feature detectors** in the visual cortex, with higher level features responding to patterns of activation in lower level cells, and propagating activation upwards to still higher level cells.

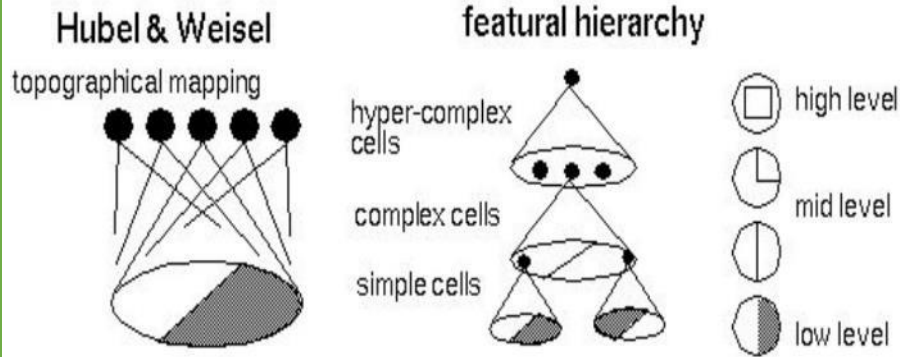
David Hubel's [Eye, Brain, and Vision](#)



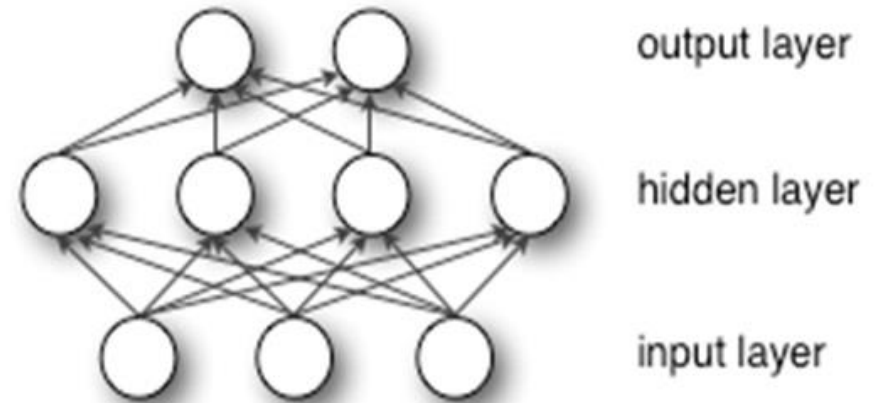
David H. Hubel and Torsten Wiesel



Hubel/Wiesel Architecture and Multi-layer Neural Network



Hubel and Wiesel's architecture

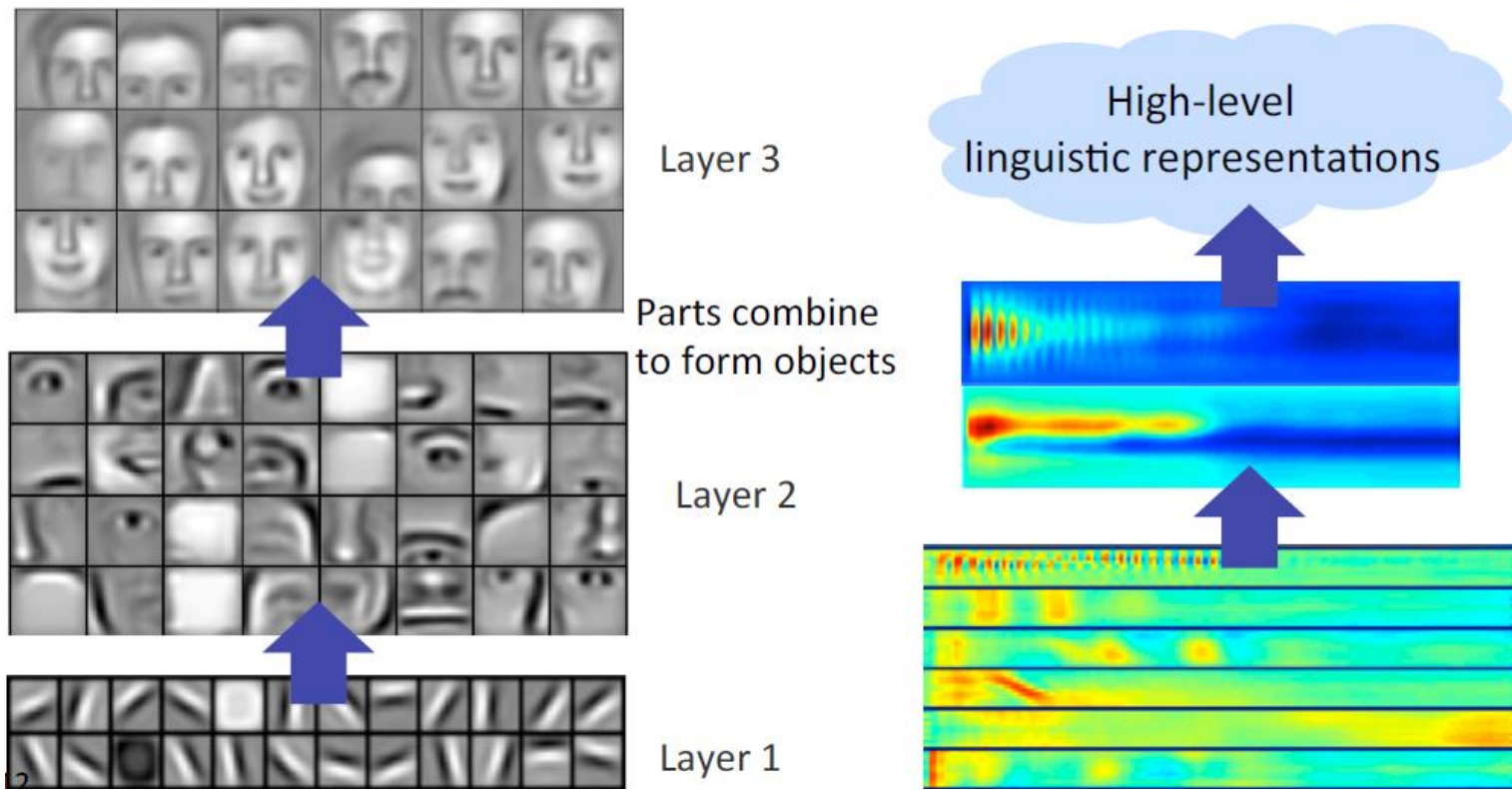


Multi-layer Neural Network
- *A non-linear classifier*



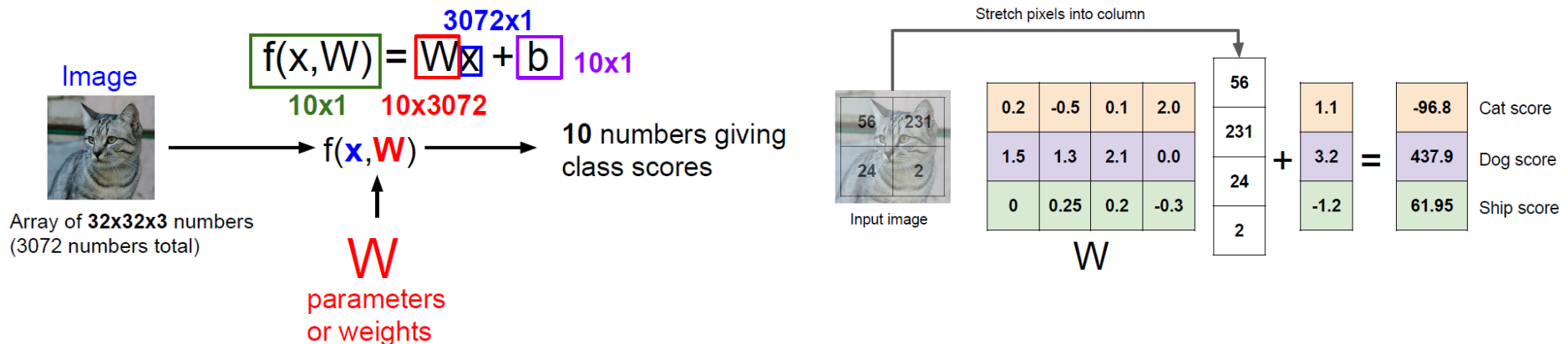
Hierarchical Representation Learning

- Successive model layers learn deeper intermediate representations.

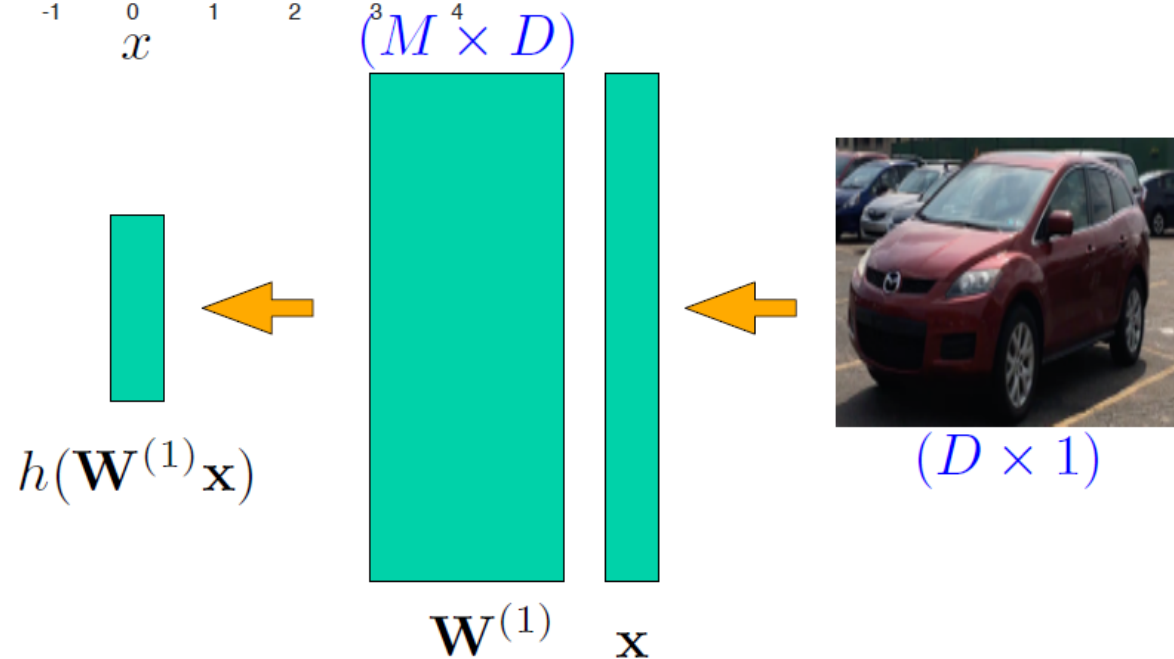
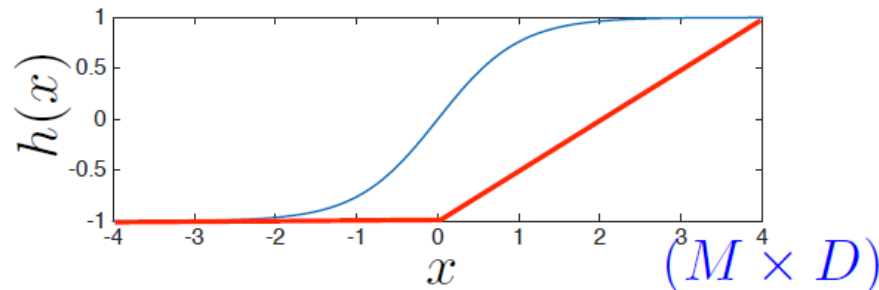


Recap: Linear Classification

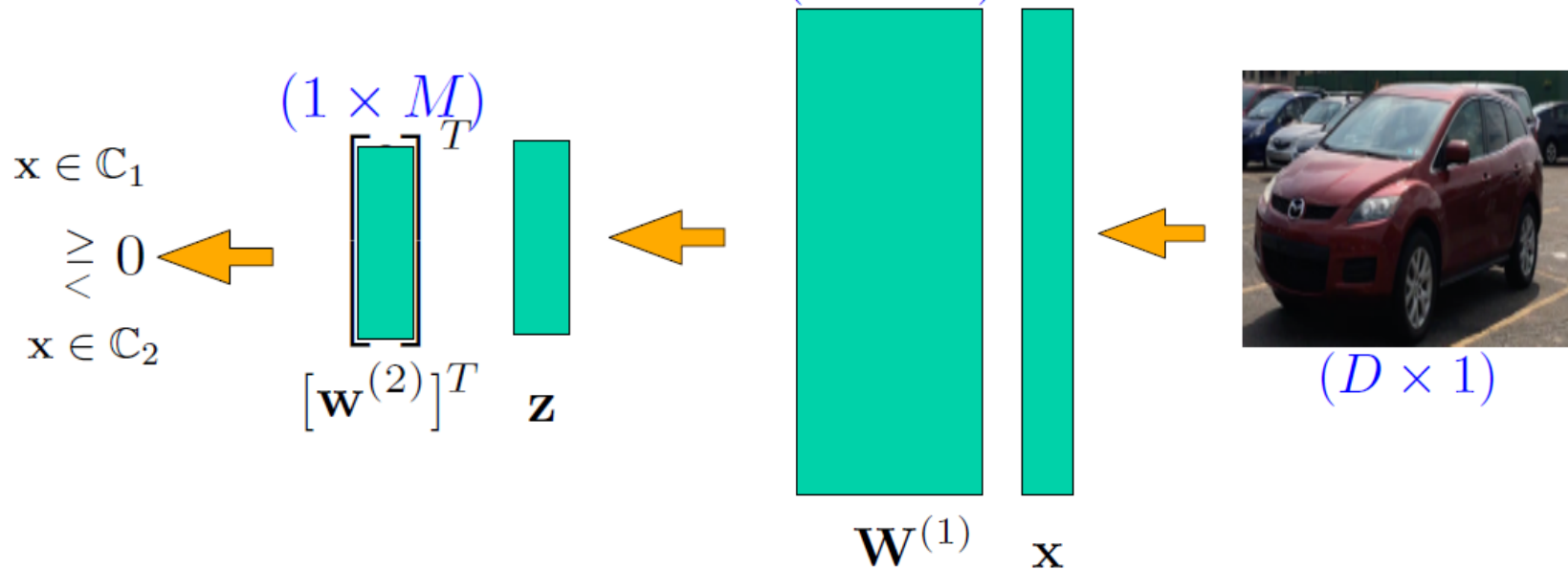
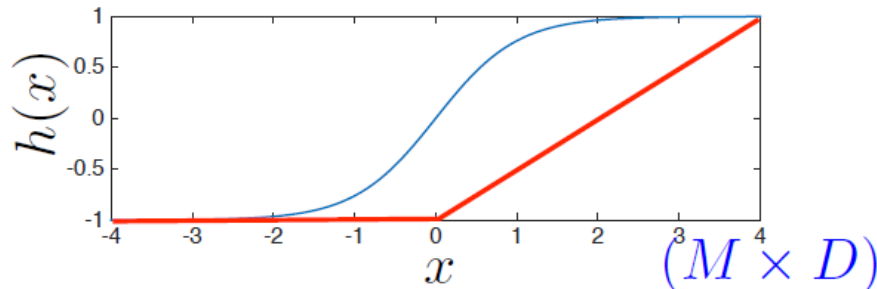
- Linear Classifier
 - Let's take the input image as $\mathbf{x}$, and the linear classifier as $\mathbf{W}$. We need $\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$ as a 10-dimensional output vector, indicating the score for each class.
 - For example, an image with 2 x 2 pixels & 3 classes of interest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



Multi-Layer Perceptron: A Nonlinear Classifier

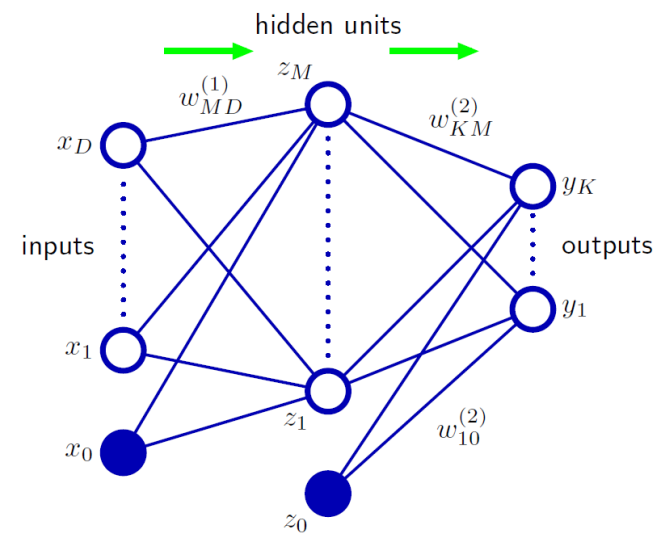


Multi-Layer Perceptron: A Nonlinear Classifier (cont'd)



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



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



Layer 1

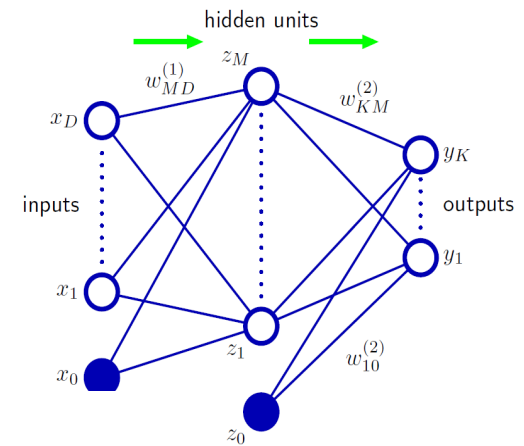


$$\mathbf{z} \in \mathbb{R}^M$$

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

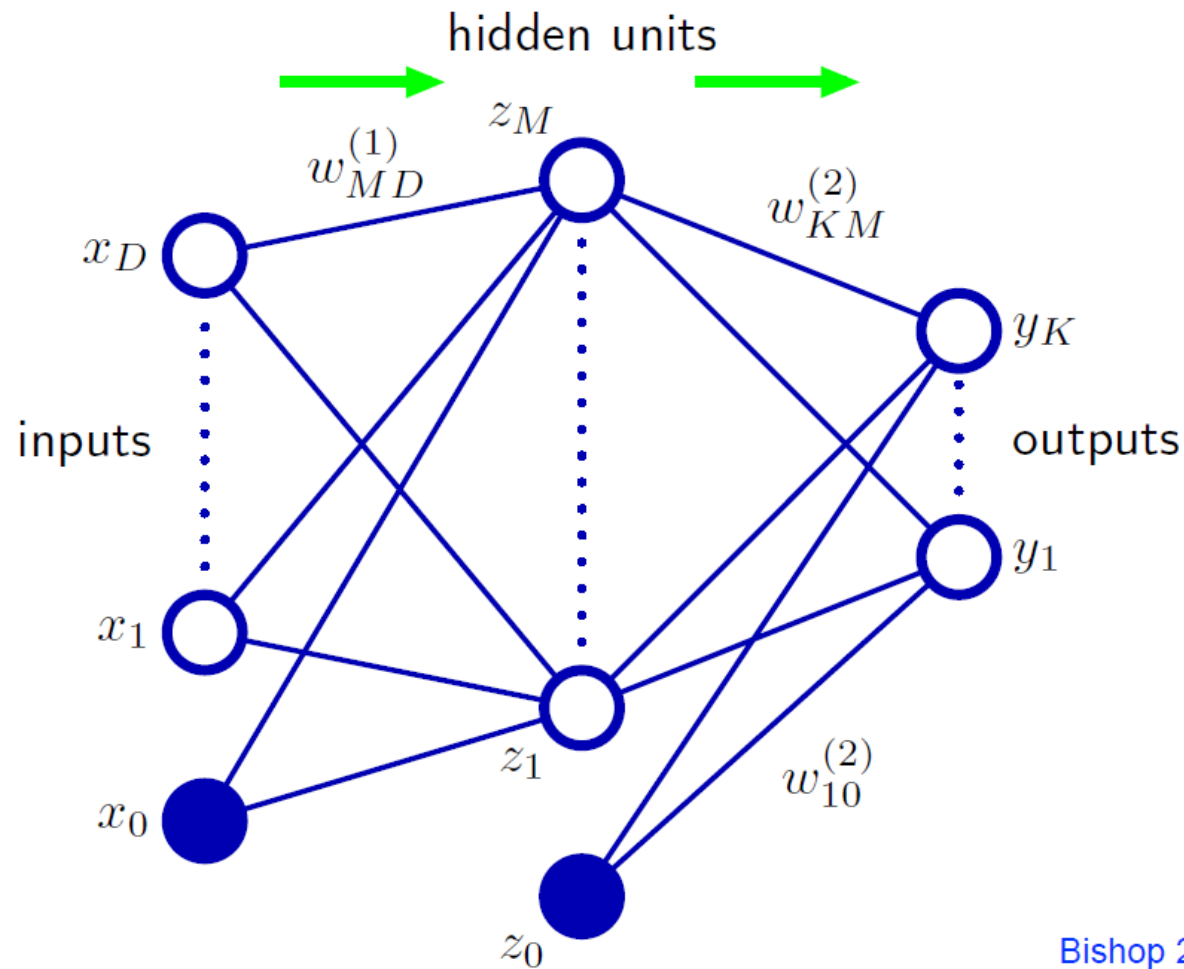
$$\mathbf{z}^T \mathbf{w}^{(2)} \begin{matrix} \geq \\ < \end{matrix} 0$$

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



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

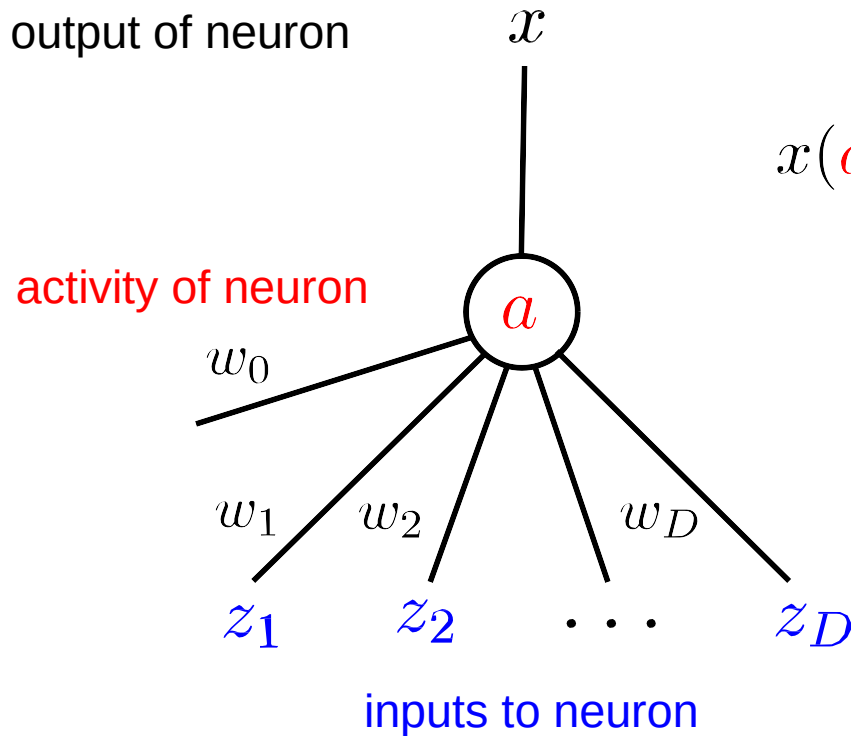
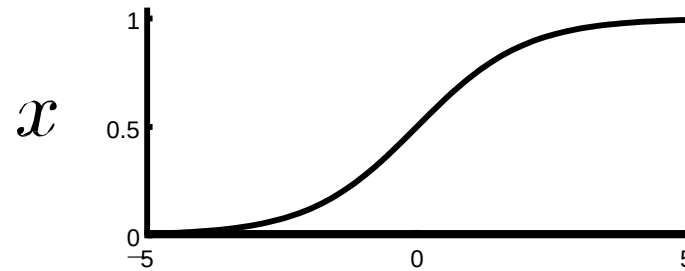
Multi-Layer Perceptron: A Nonlinear Classifier (cont'd)



Bishop 2006

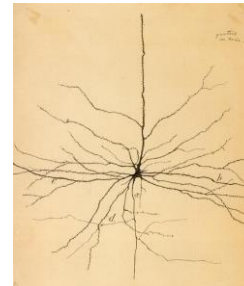
Let's Get a Closer Look...

- A single neuron

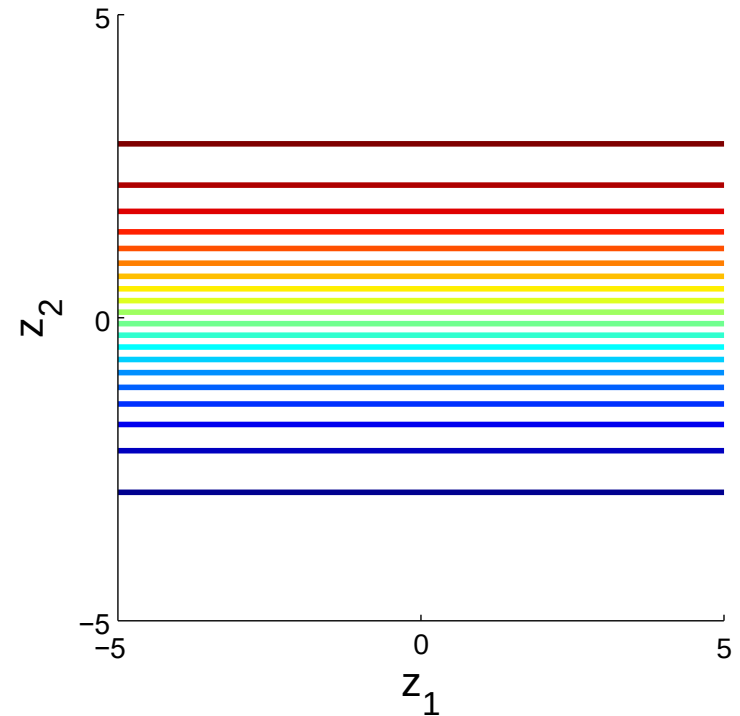
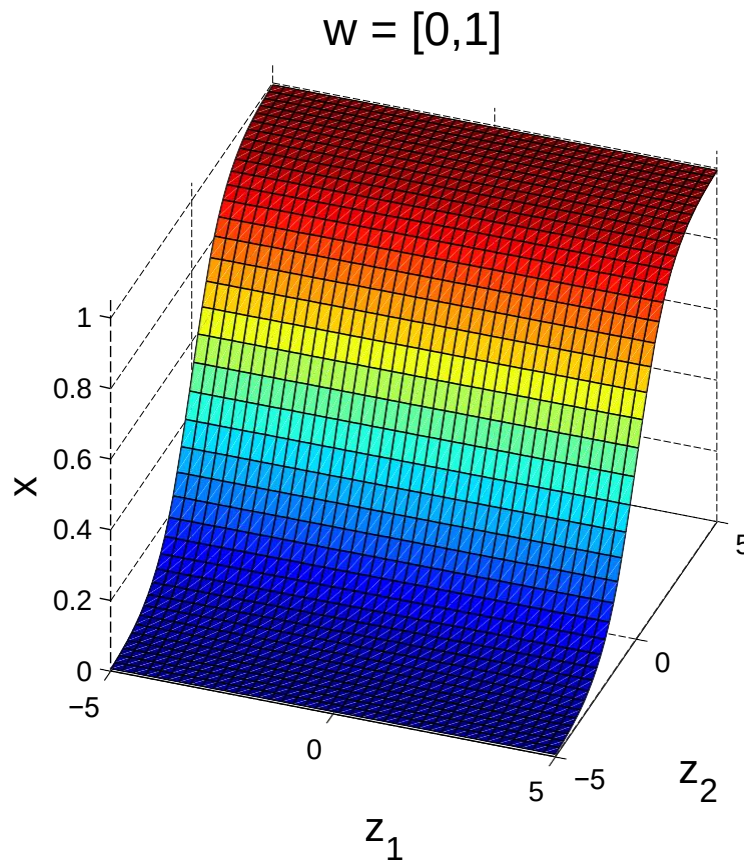


$$x(a) = \frac{1}{1 + \exp(-a)} \quad x \in (0, 1)$$

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

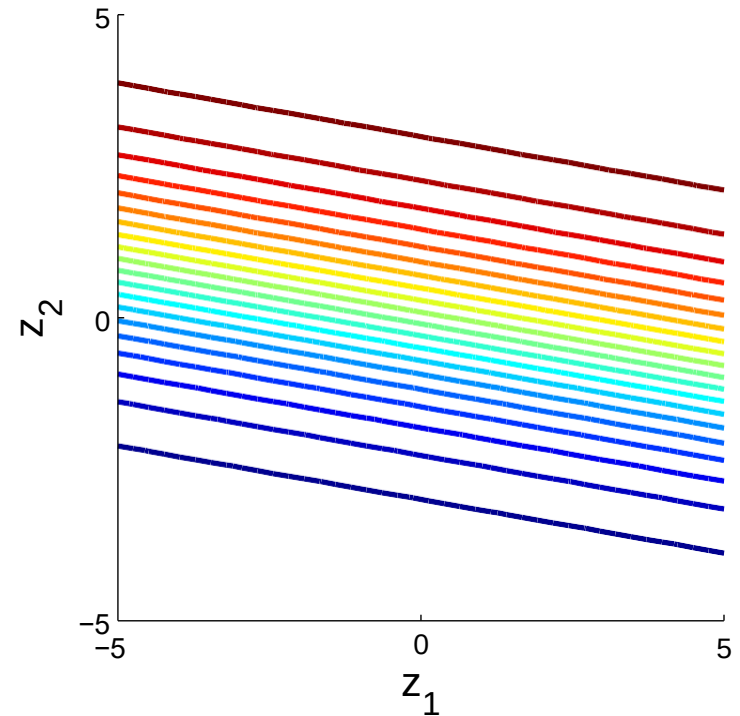
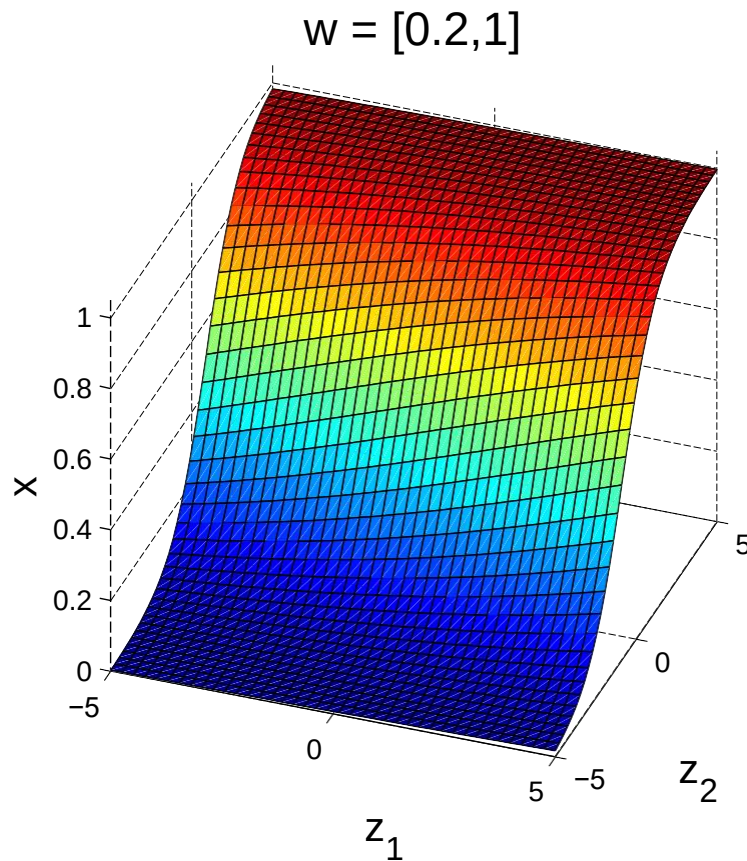


Input-Output Function of a Single Neuron



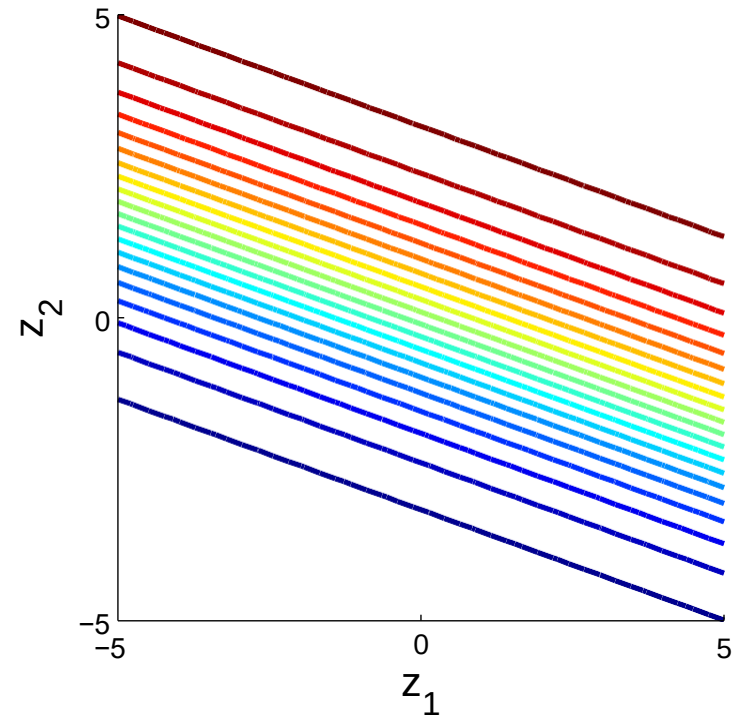
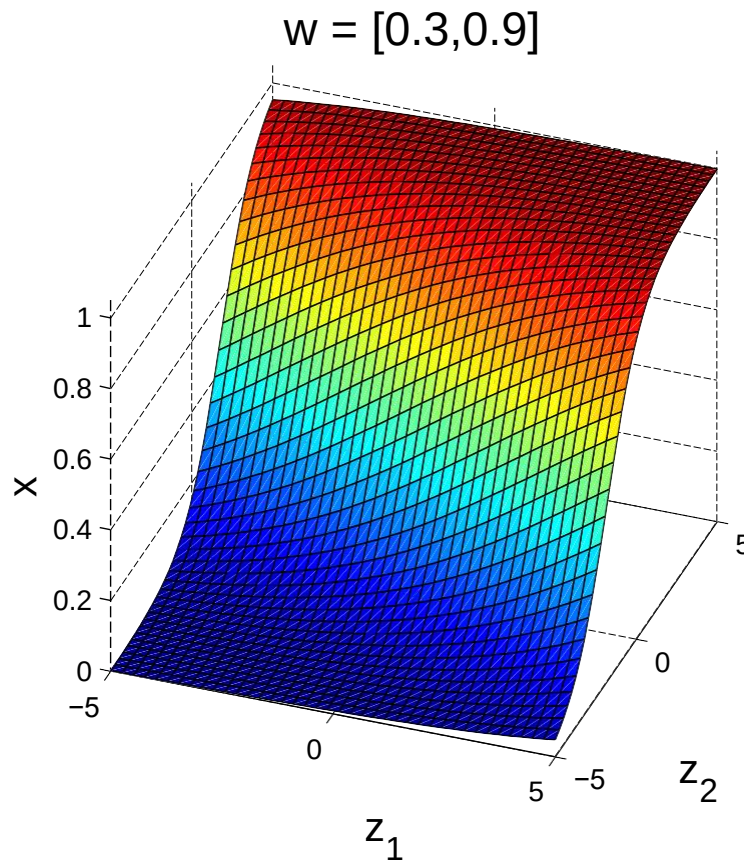
$$x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)}$$

Input-Output Function of a Single Neuron



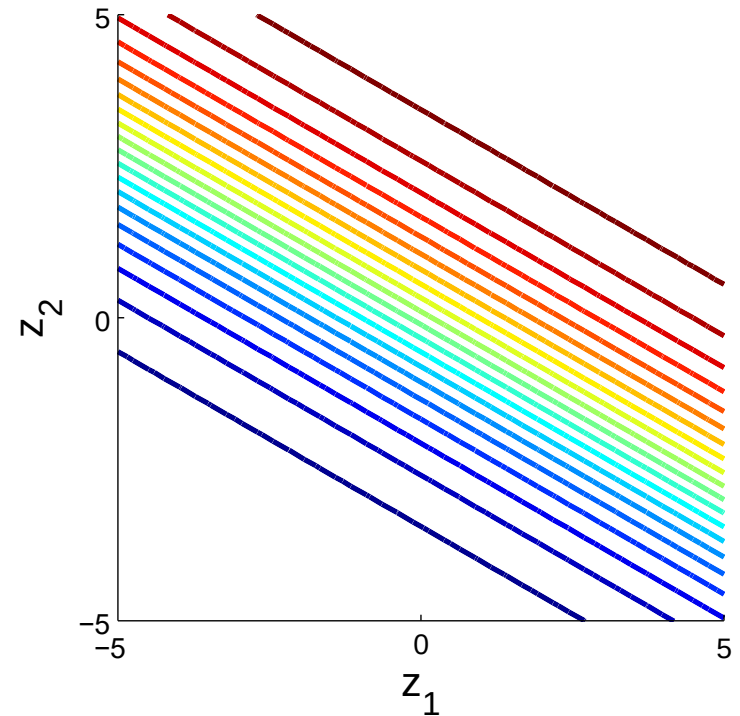
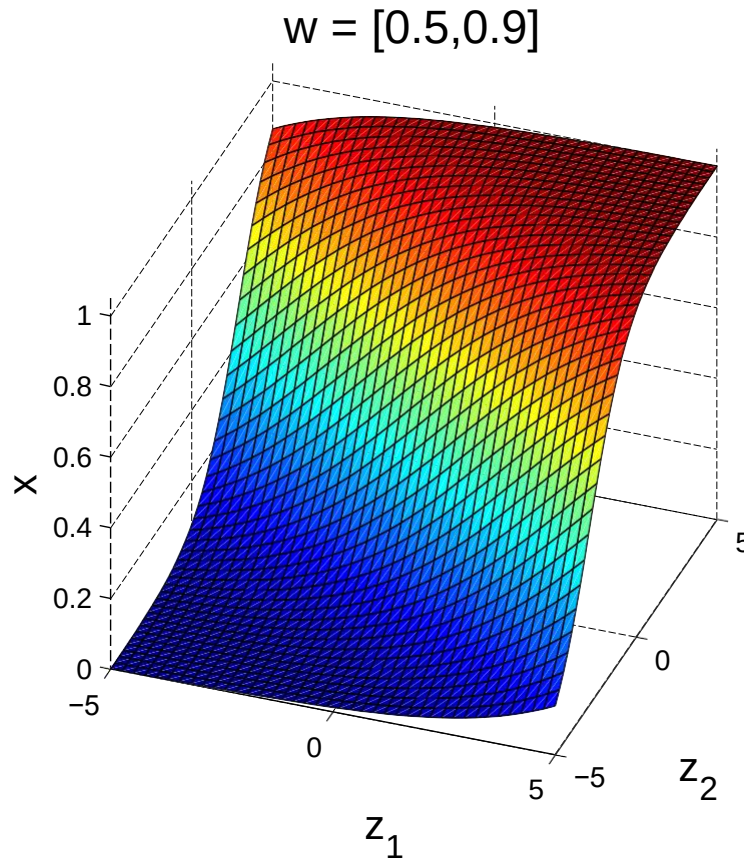
$$x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)}$$

Input-Output Function of a Single Neuron



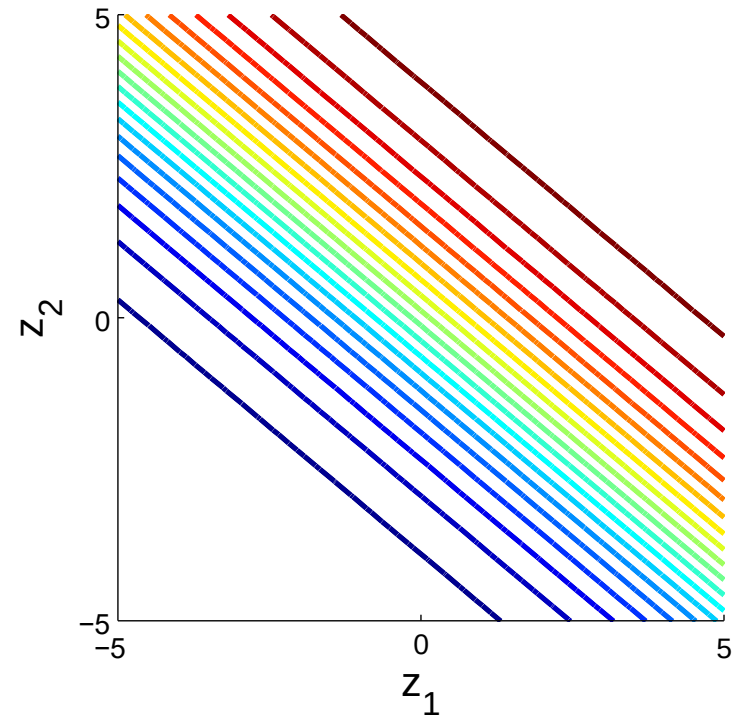
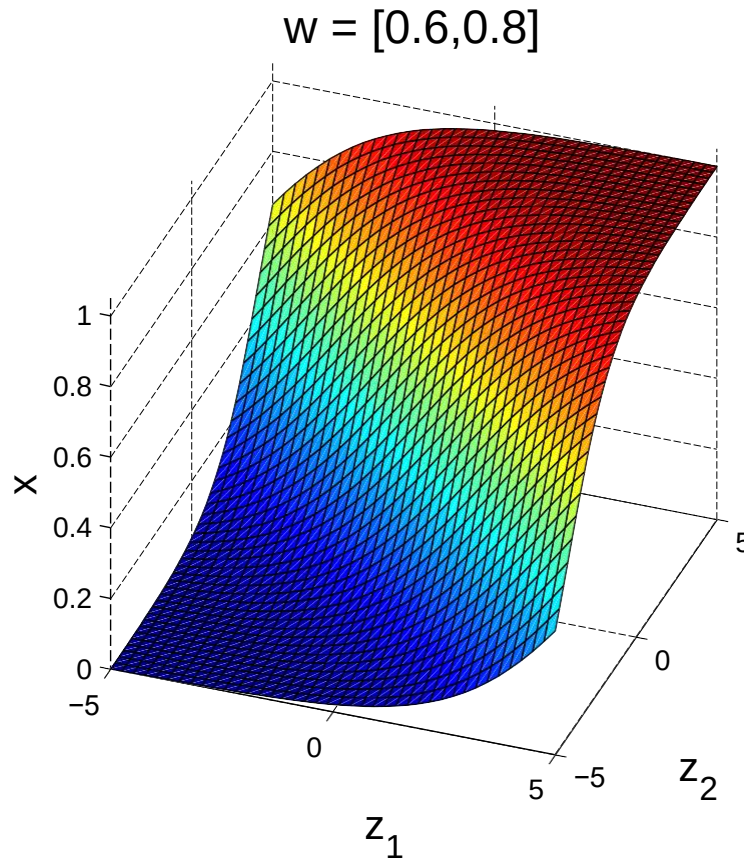
$$x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)}$$

Input-Output Function of a Single Neuron



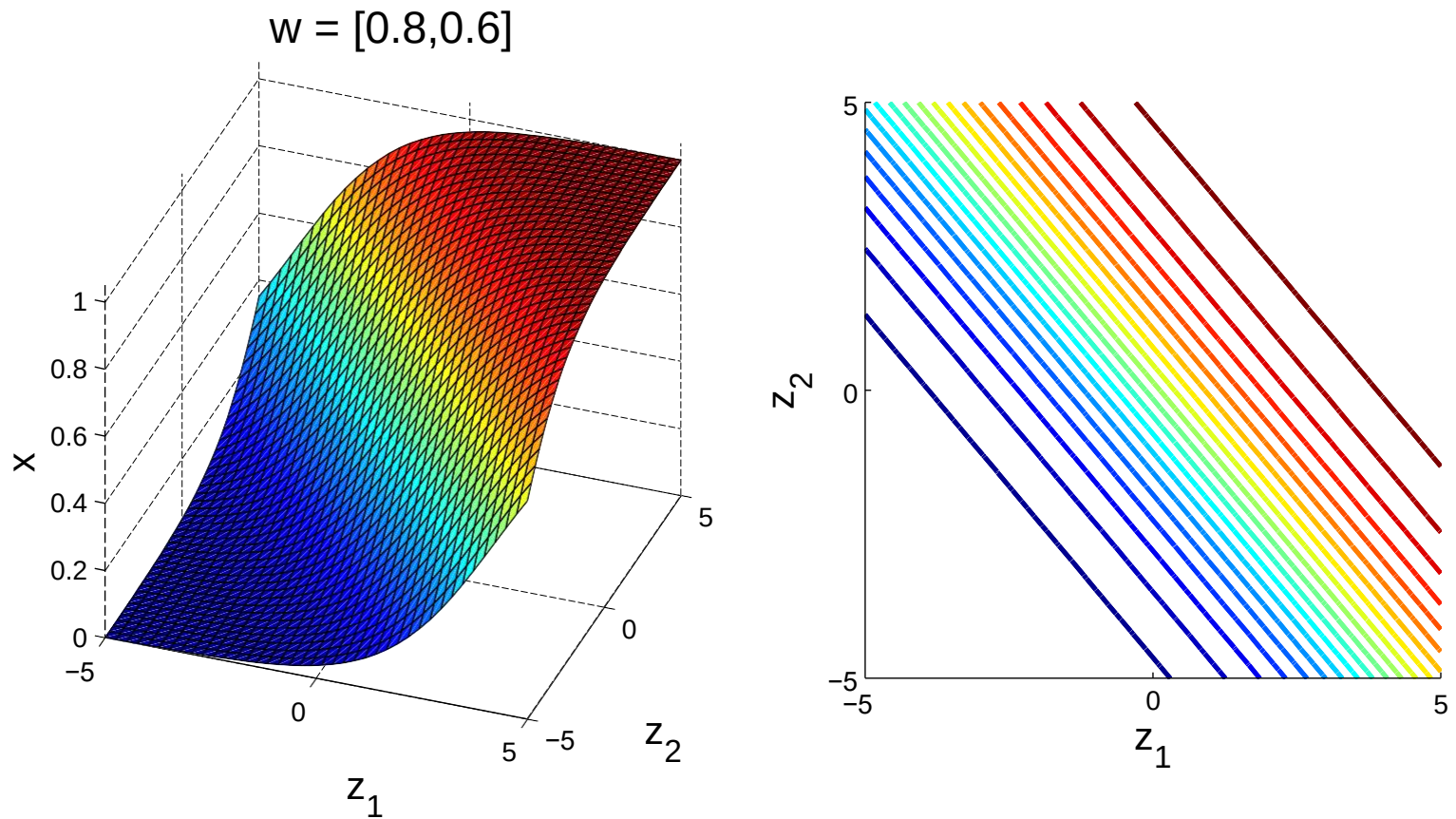
$$x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)}$$

Input-Output Function of a Single Neuron



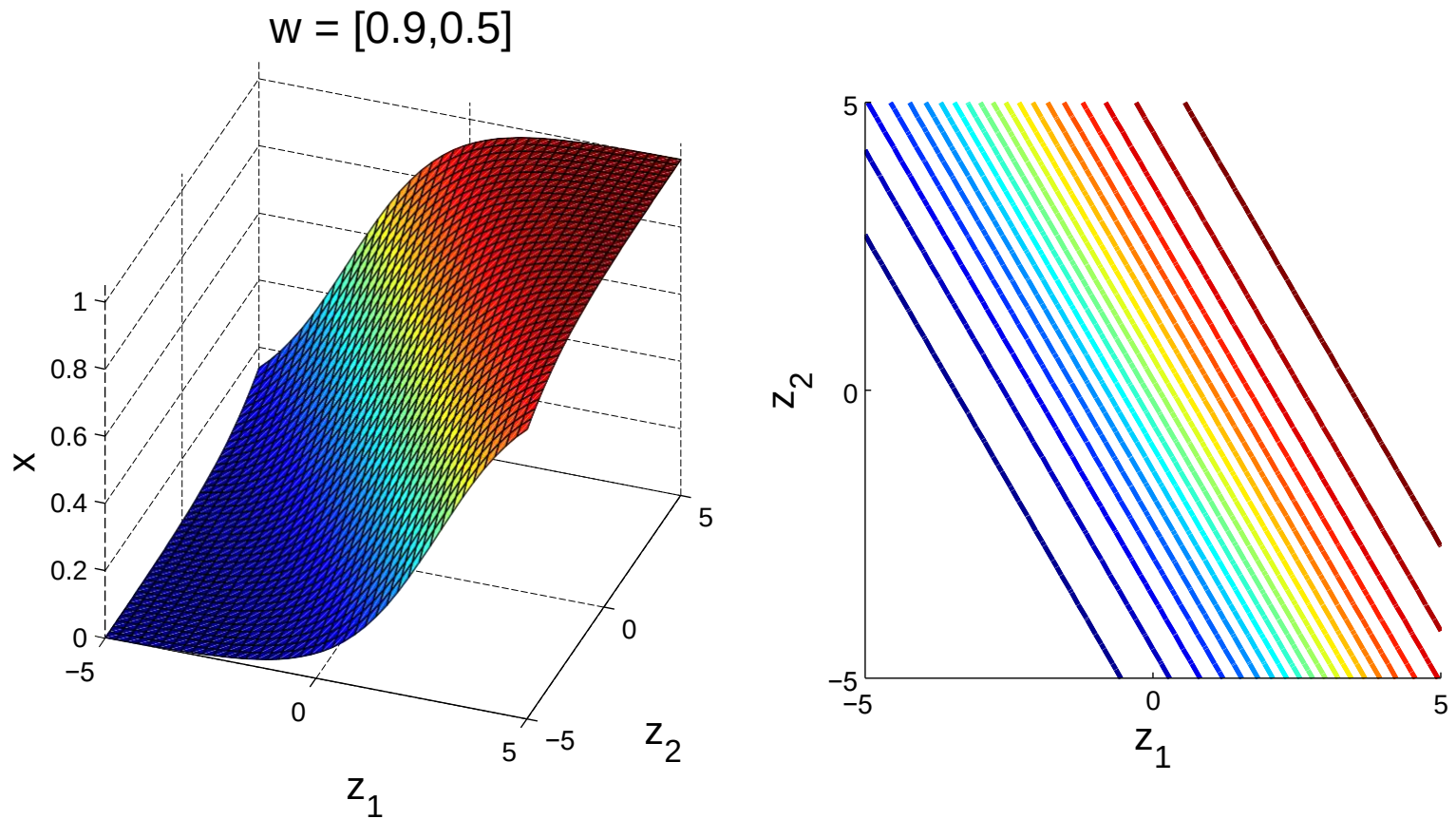
$$x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)}$$

Input-Output Function of a Single Neuron



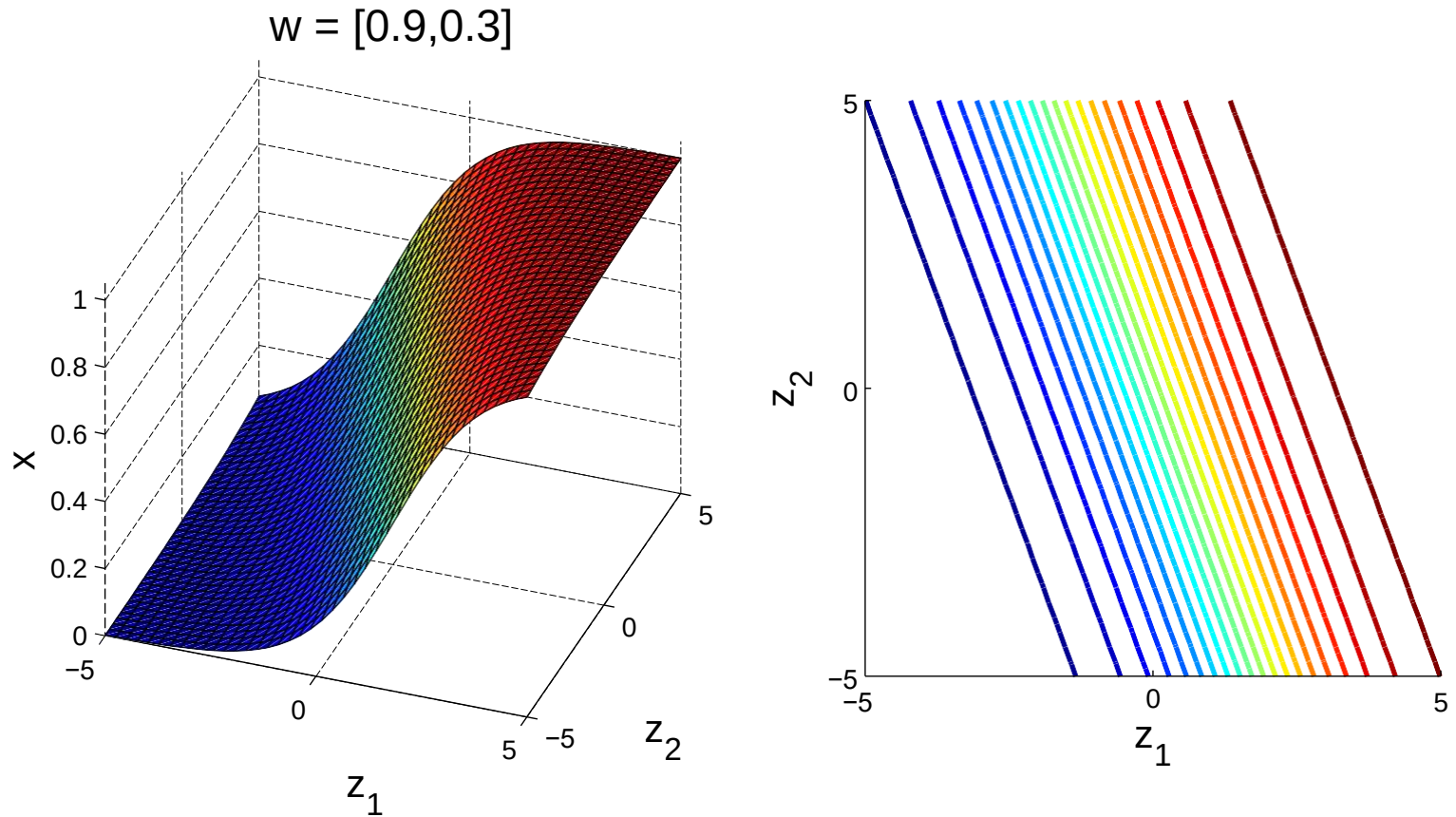
$$x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)}$$

Input-Output Function of a Single Neuron



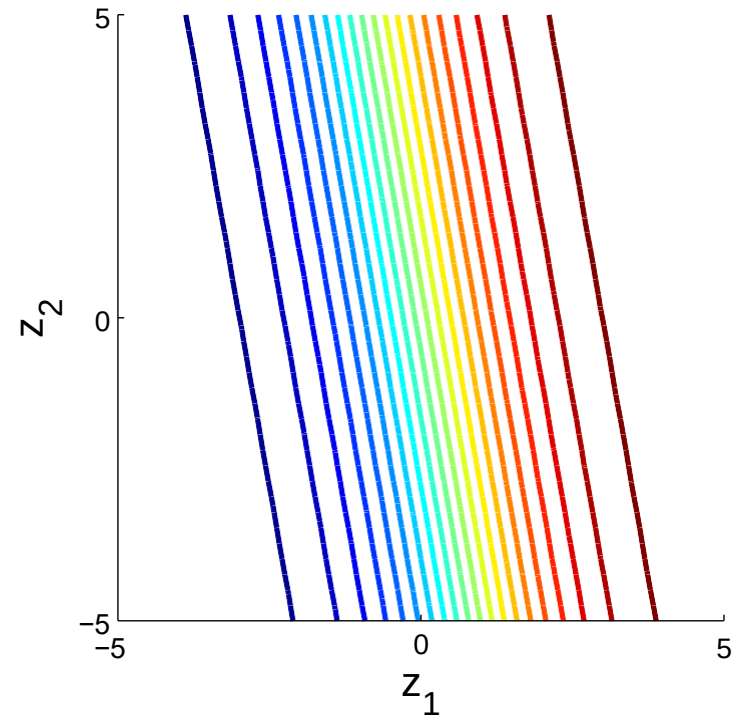
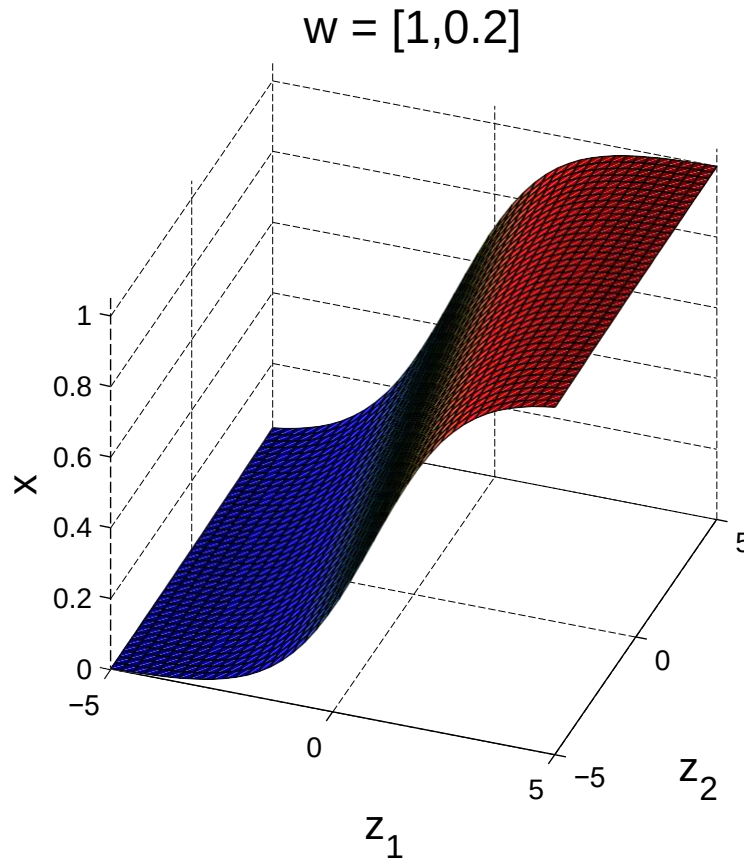
$$x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)}$$

Input-Output Function of a Single Neuron



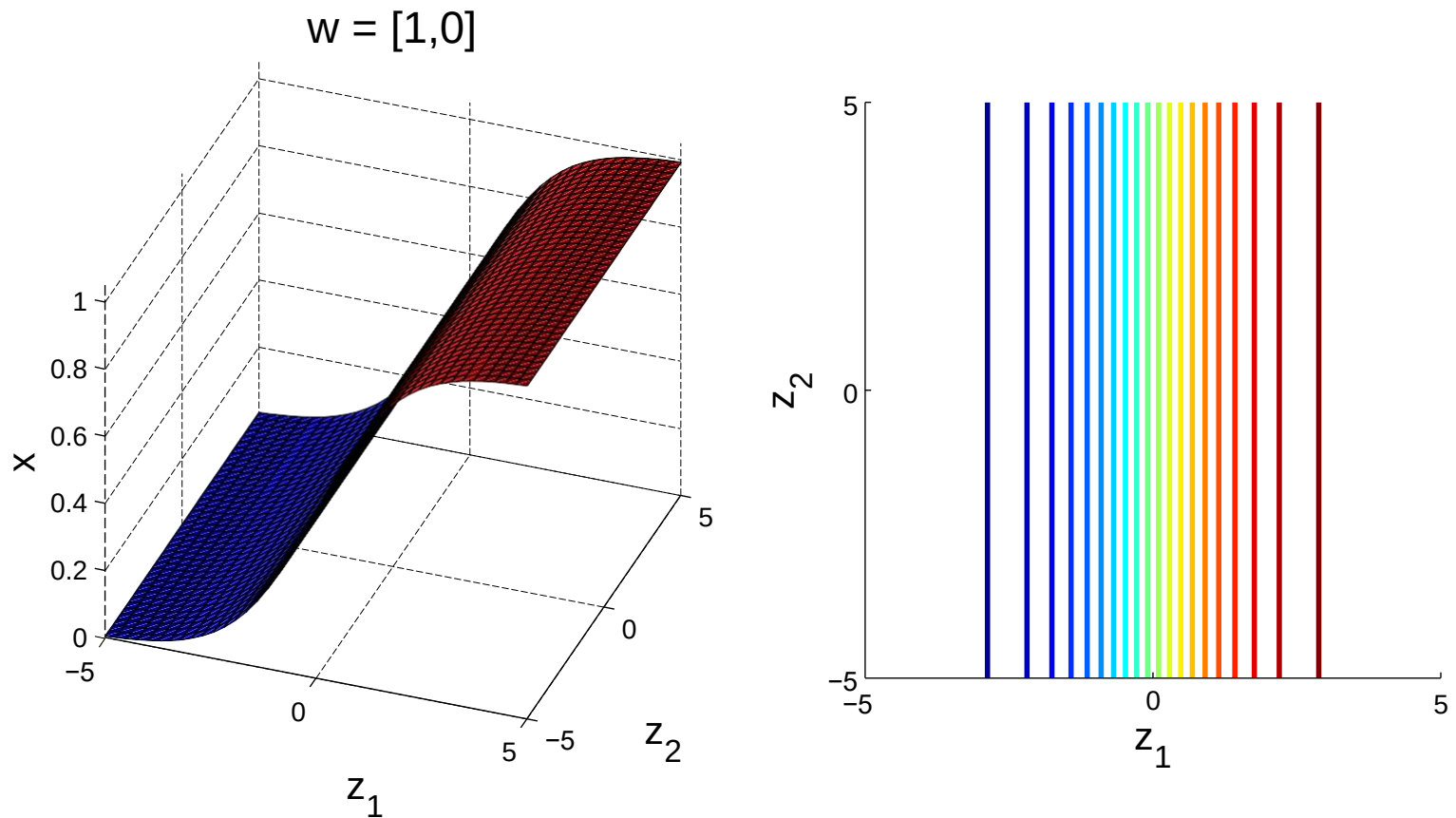
$$x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)}$$

Input-Output Function of a Single Neuron



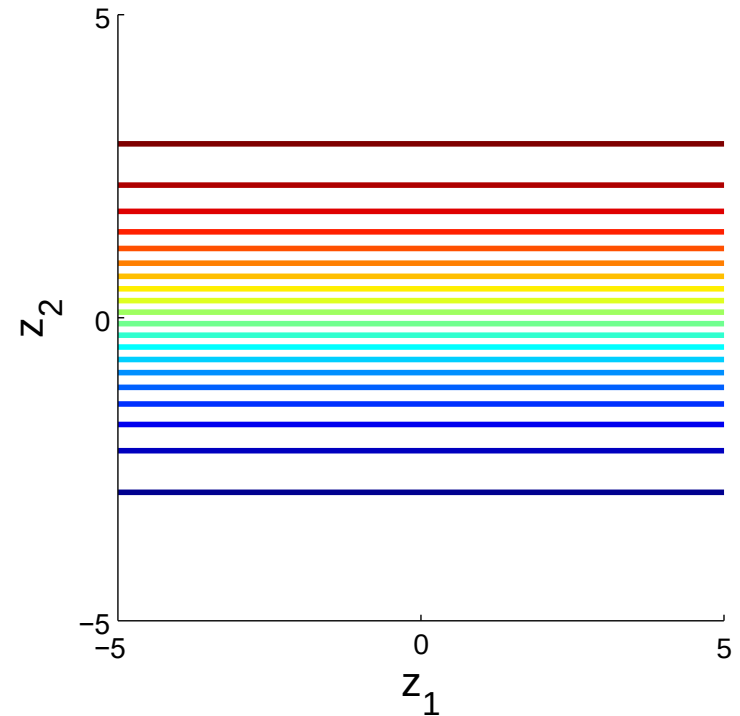
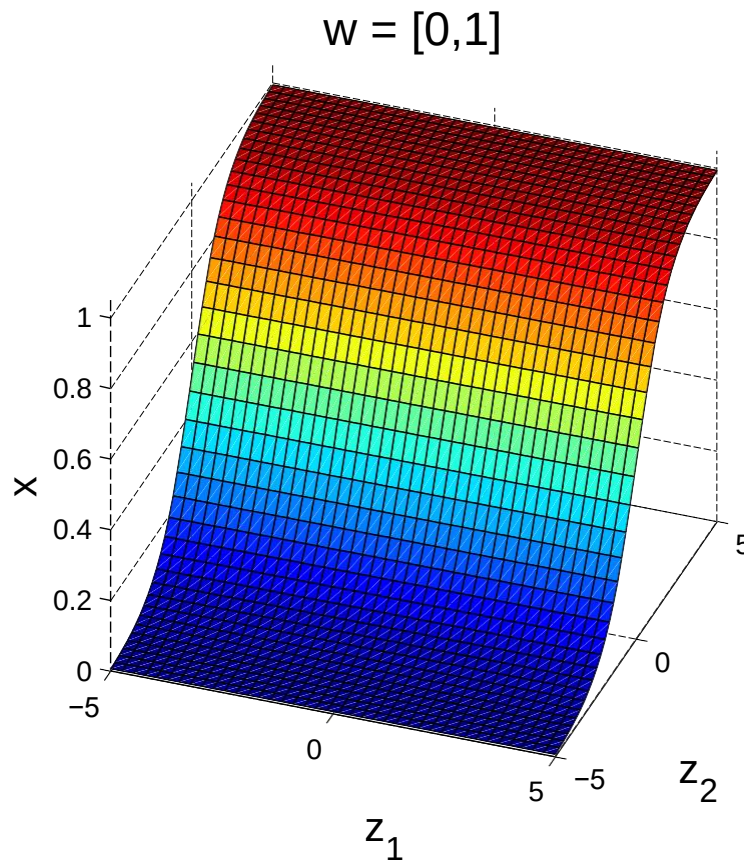
$$x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)}$$

Input-Output Function of a Single Neuron



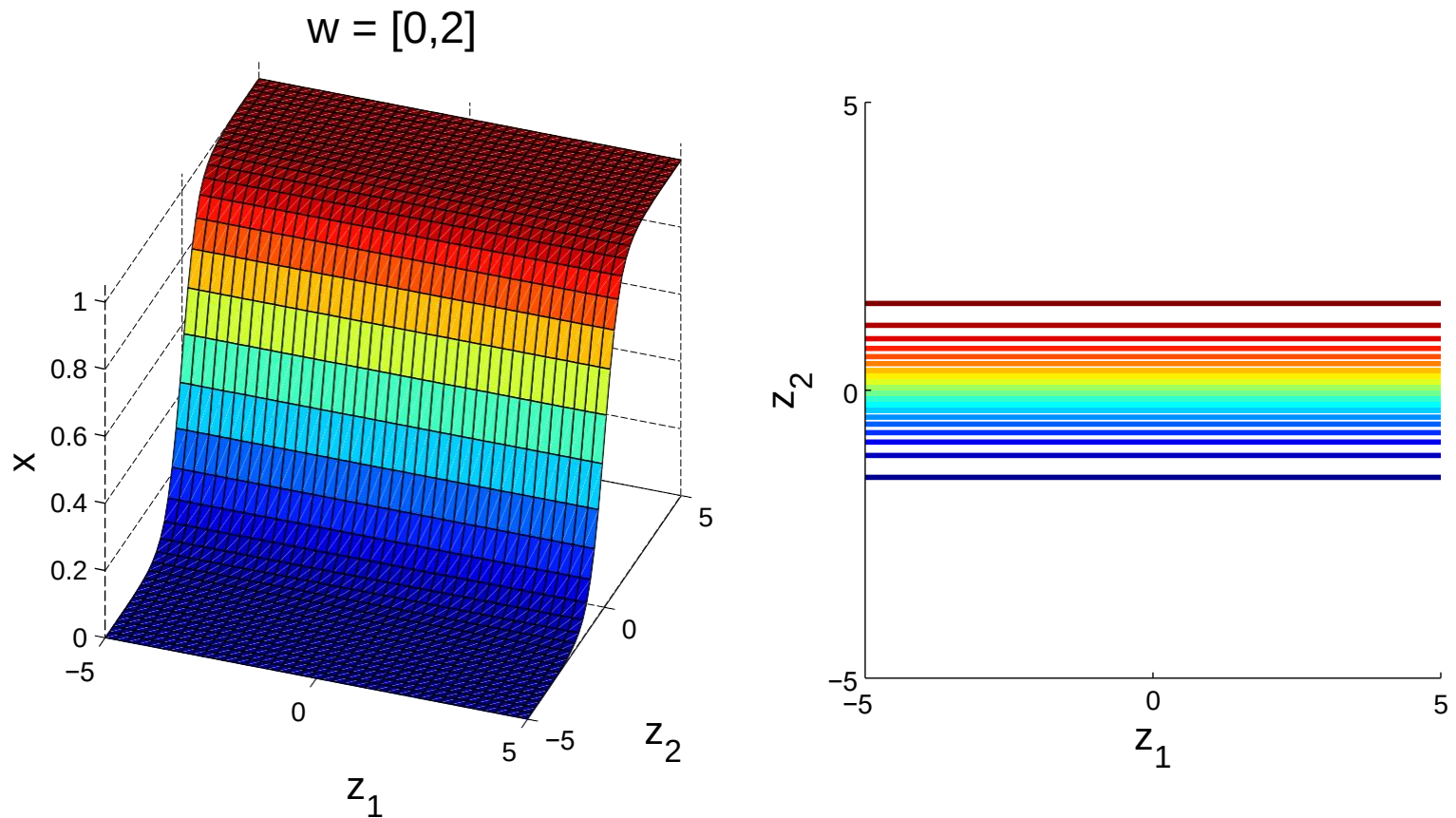
$$x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)}$$

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



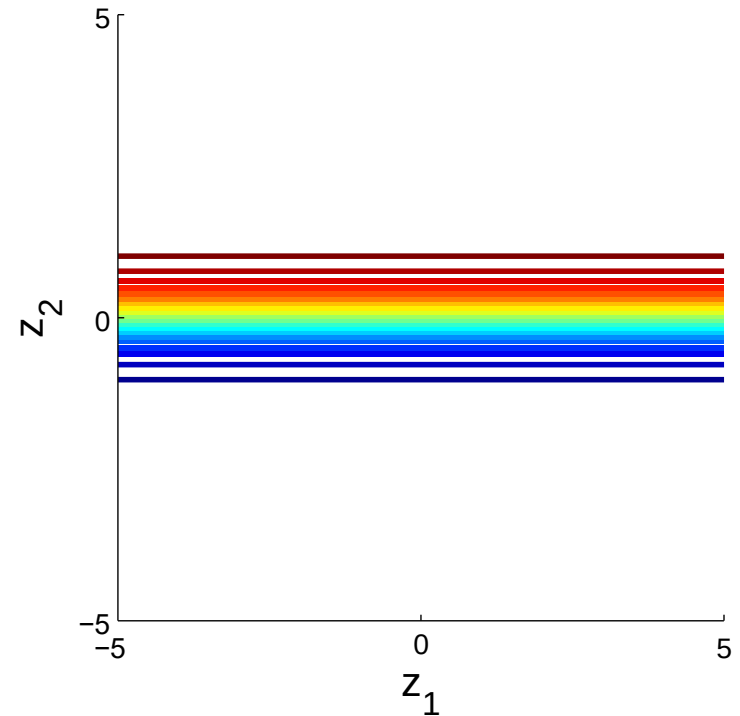
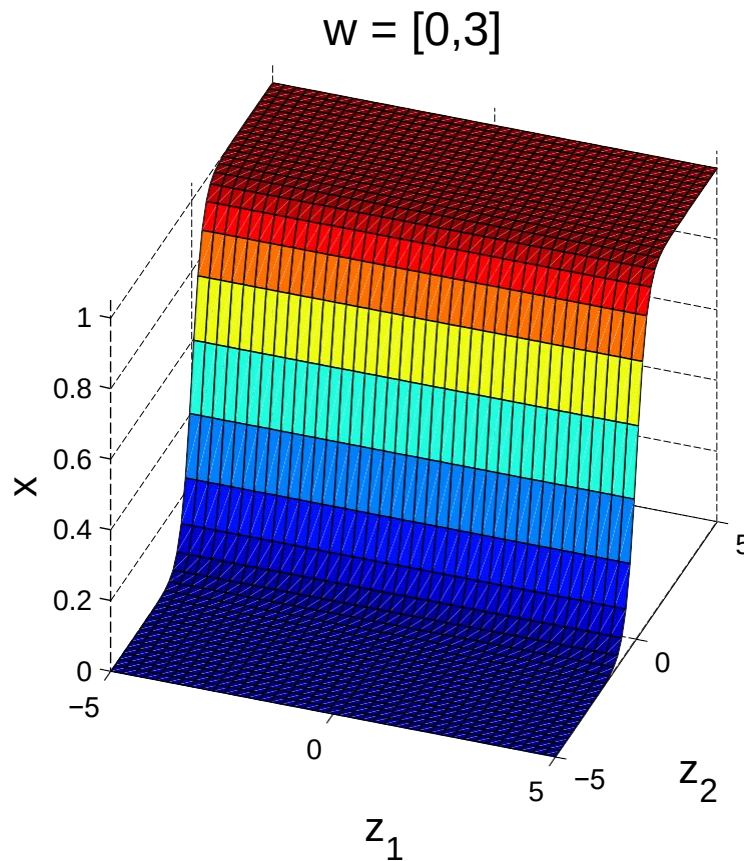
$$x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)}$$

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



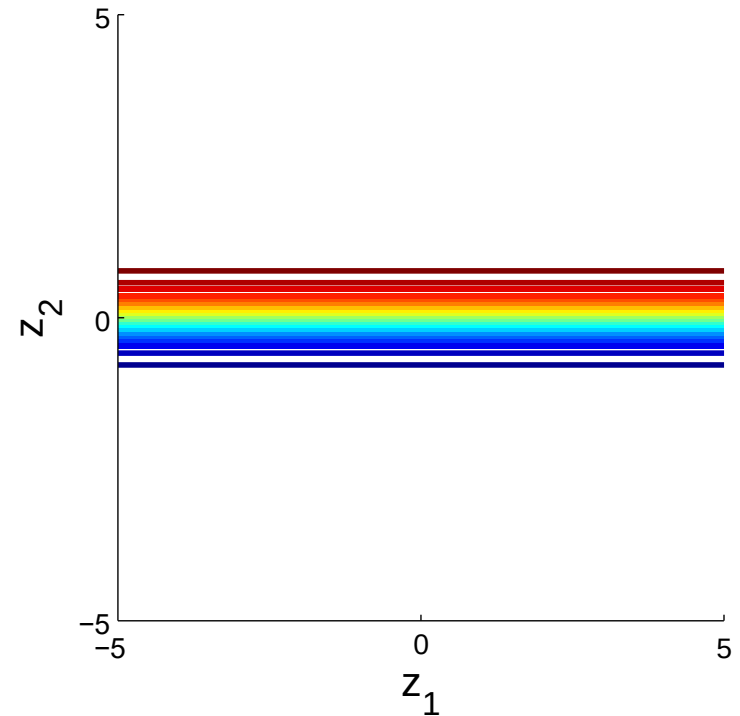
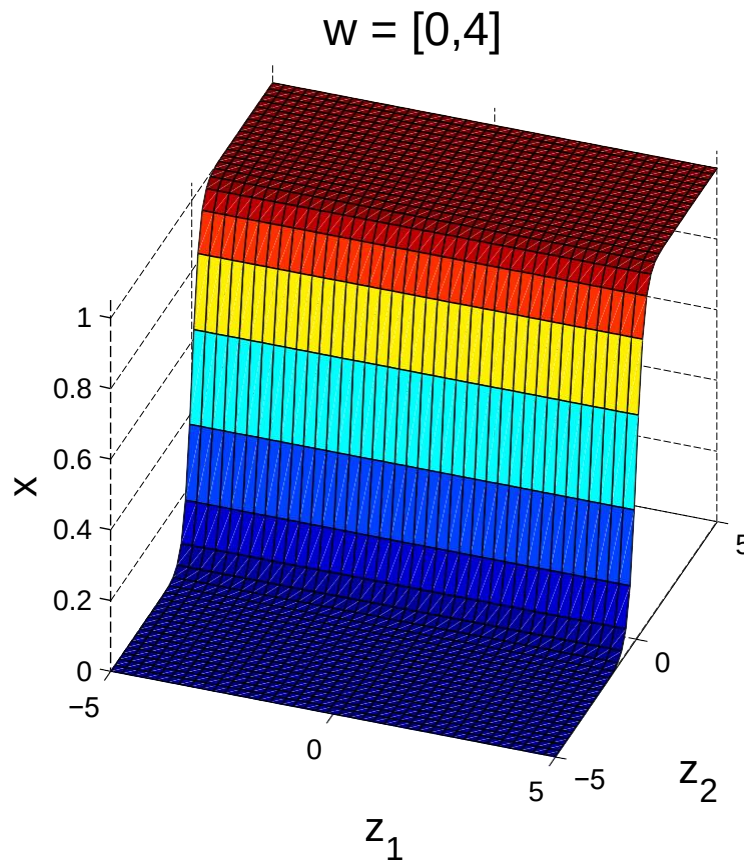
$$x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)}$$

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



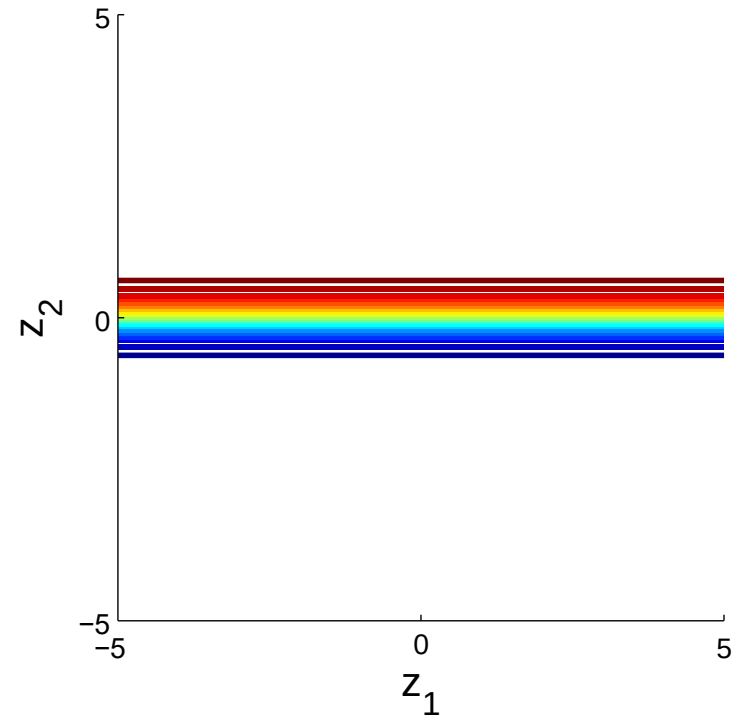
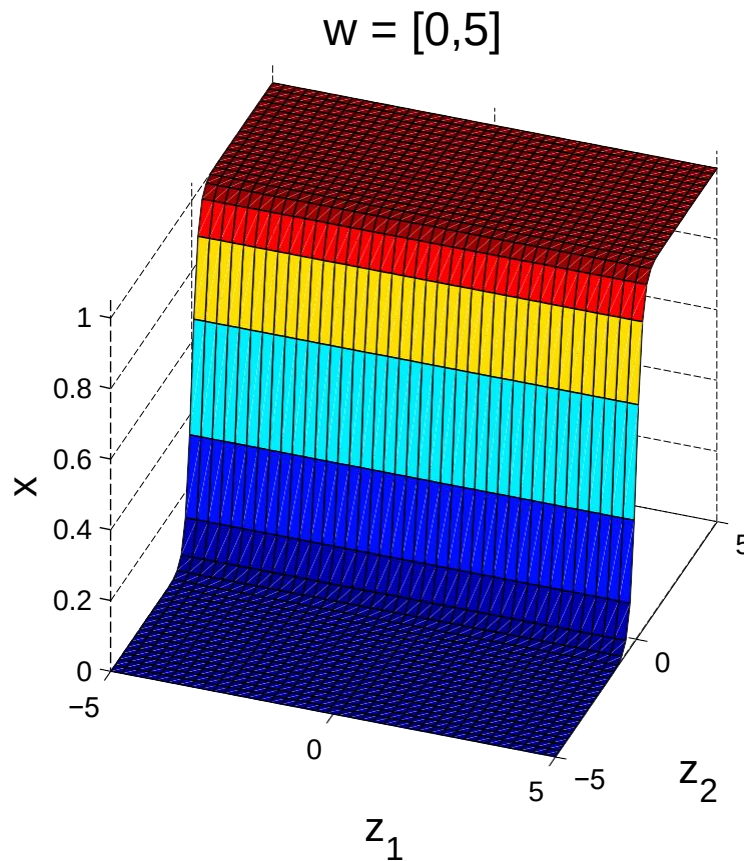
$$x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)}$$

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



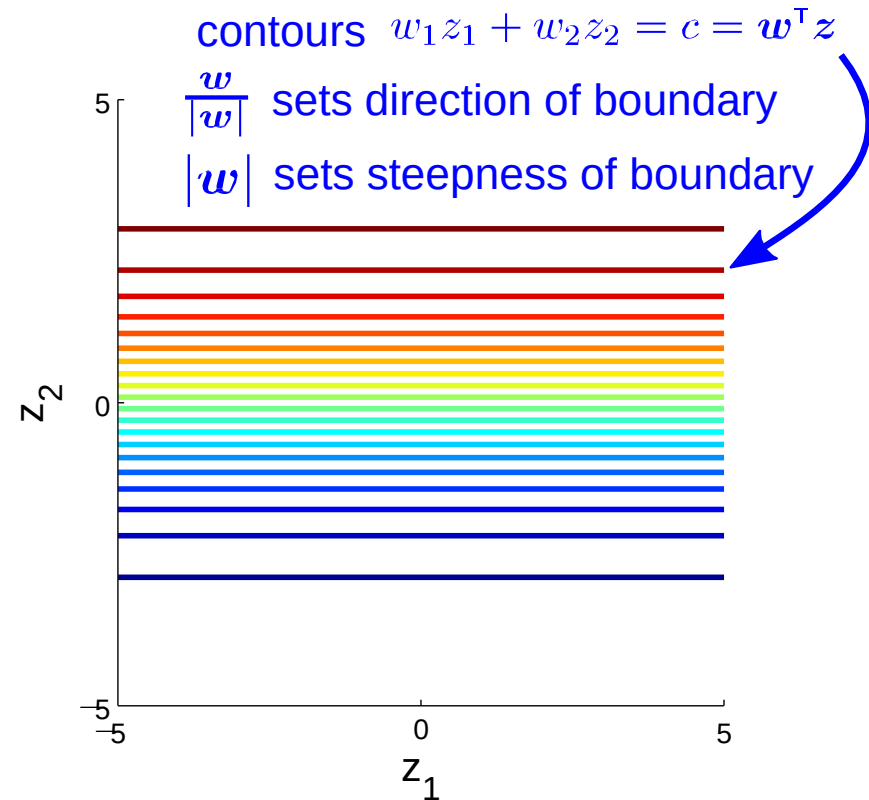
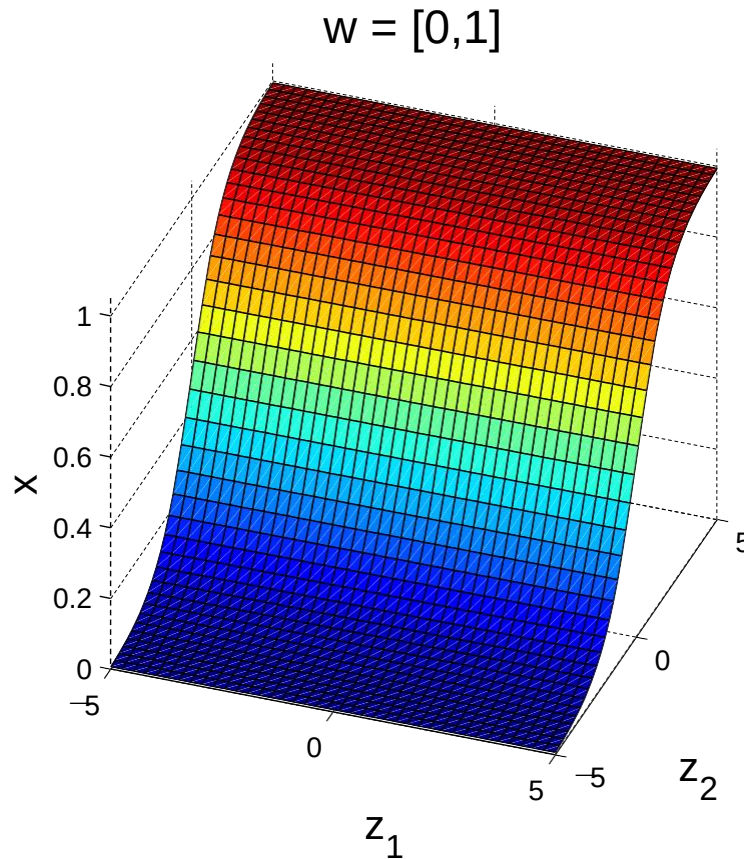
$$x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)}$$

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



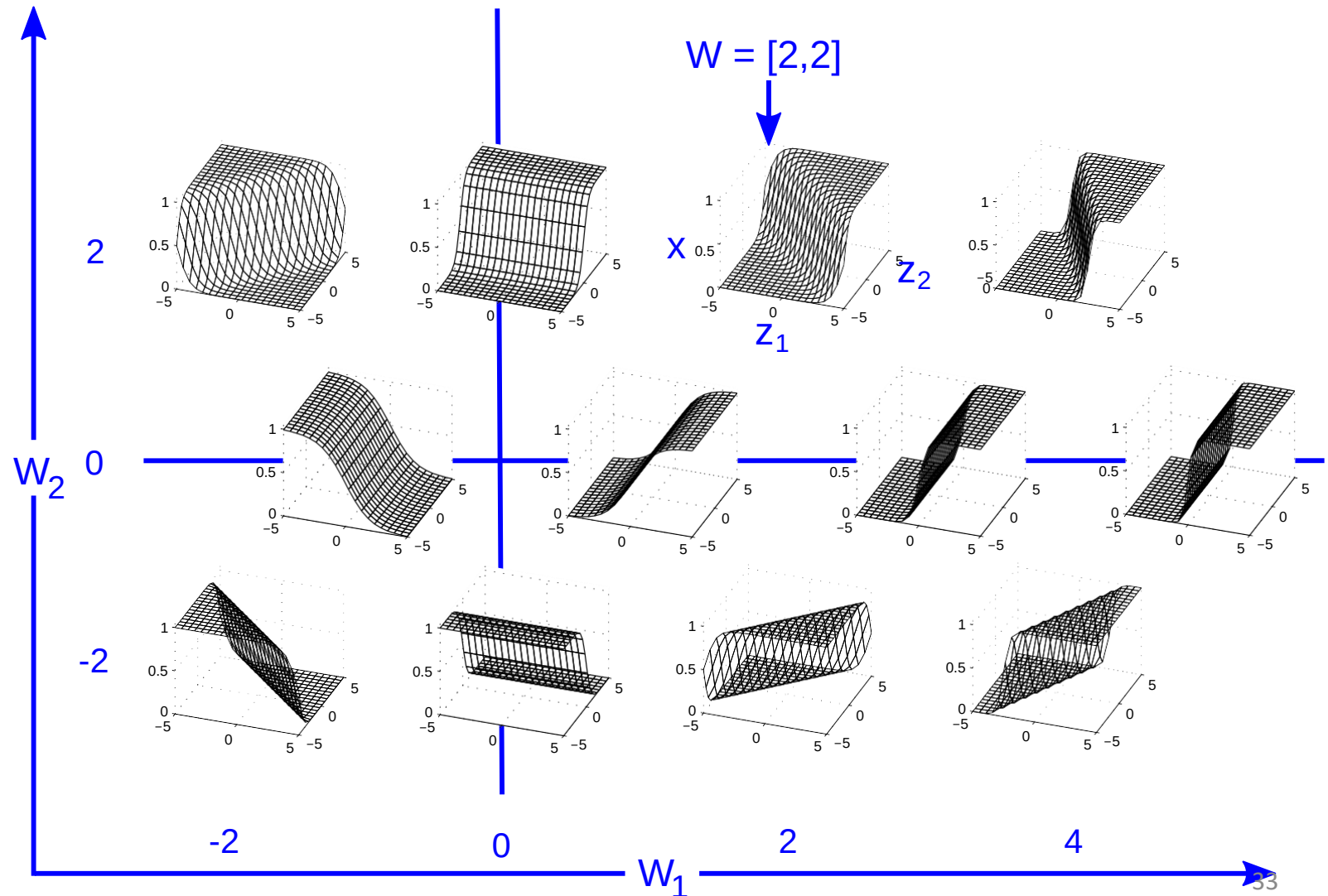
$$x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)}$$

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

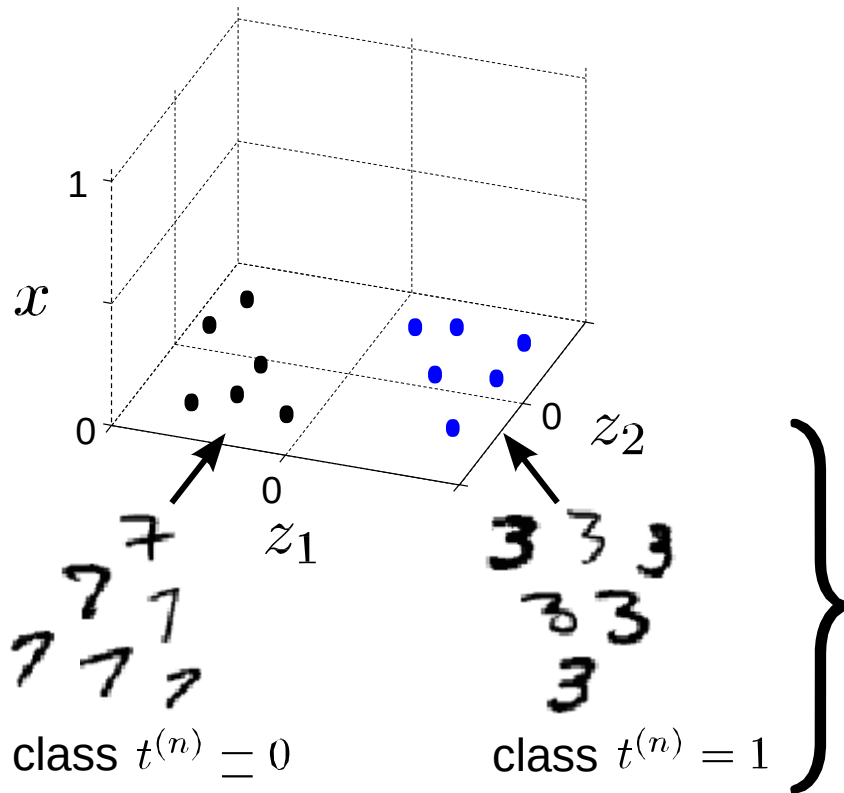


$$x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)}$$

Weight Space of a Single Neuron



Training a Single Neuron



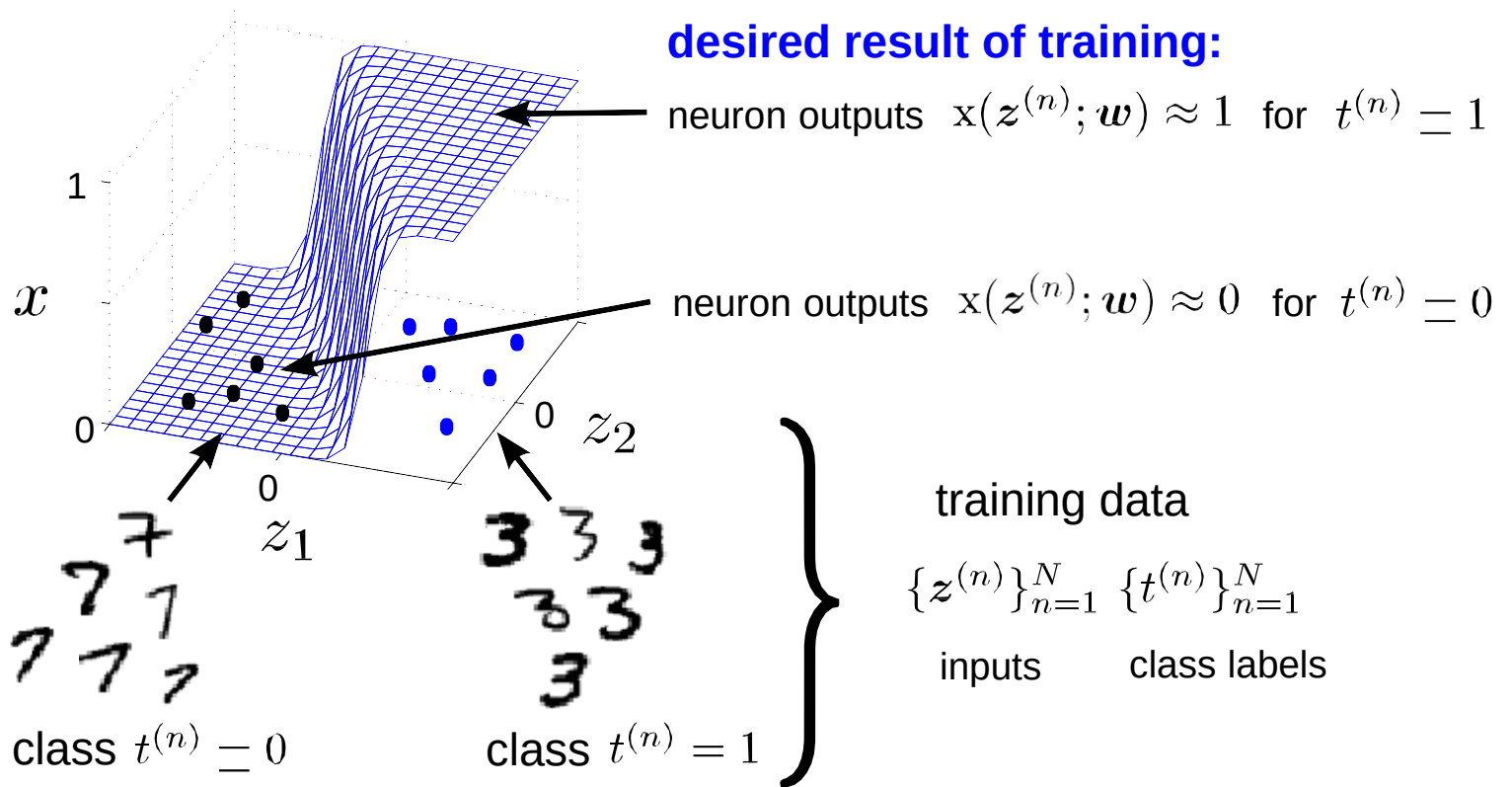
training data

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

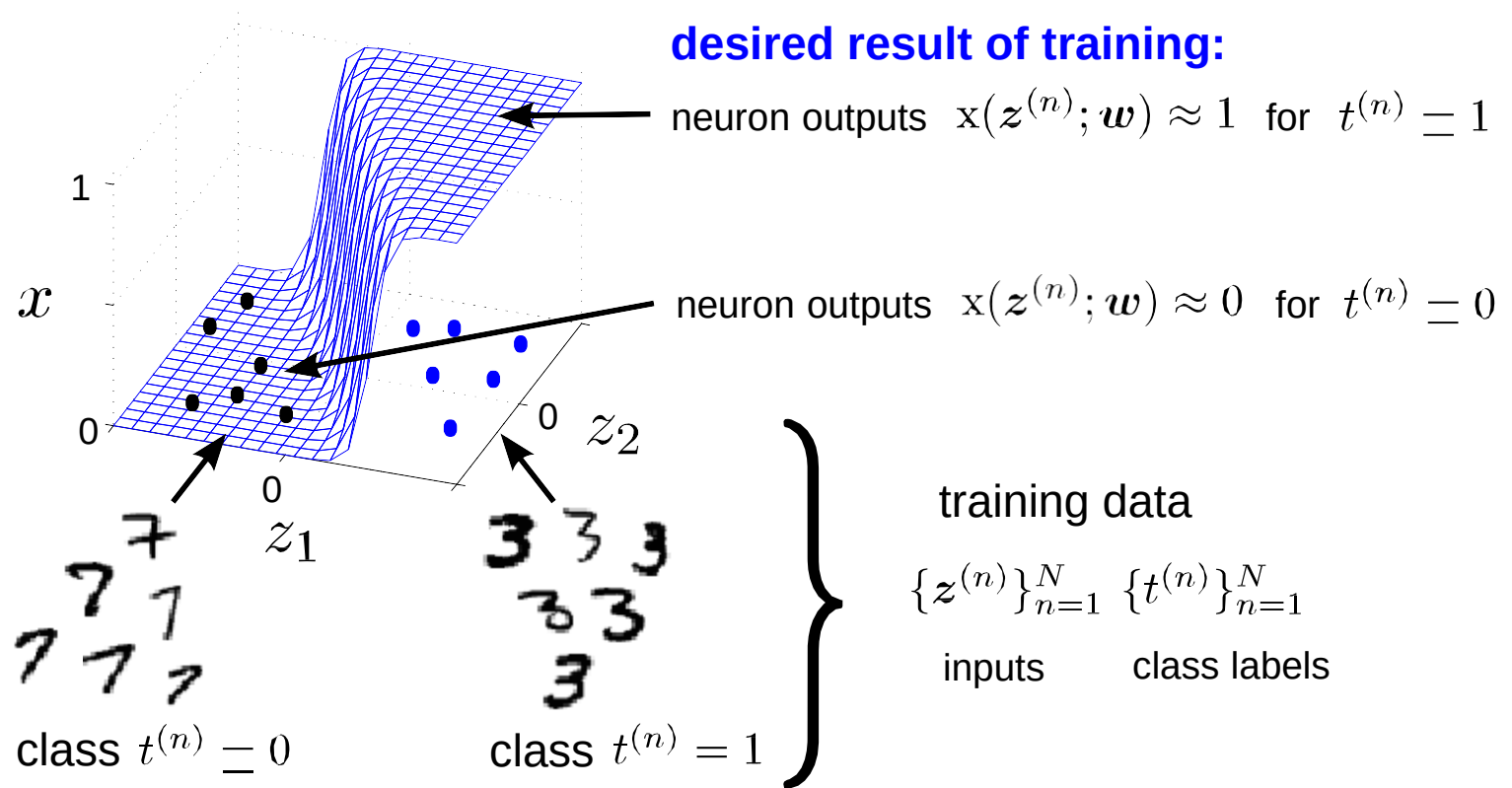
inputs

class labels

Training a Single Neuron



Training a Single Neuron

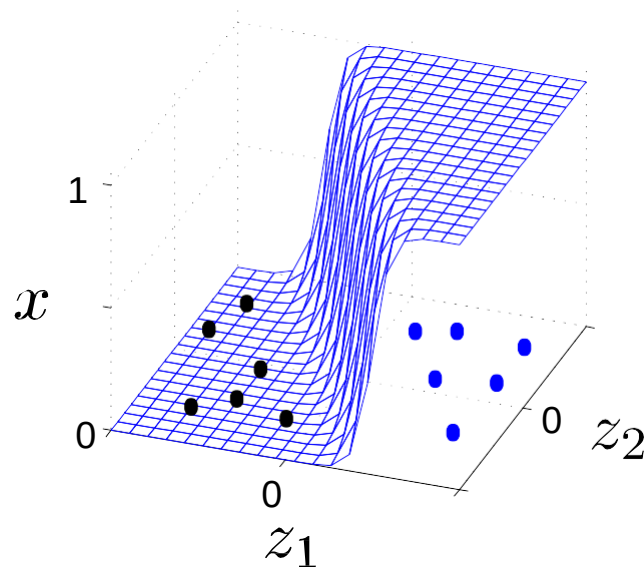


objective function:

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

surprise $-\log p(\text{outcome})$ when observing $t^{(n)}$ } encourages neuron output
 relative entropy between $x(z^{(n)}; \mathbf{w})$ and $t^{(n)}$ } to match training data 36

Training a Single Neuron



training data

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

inputs

class labels

objective function:

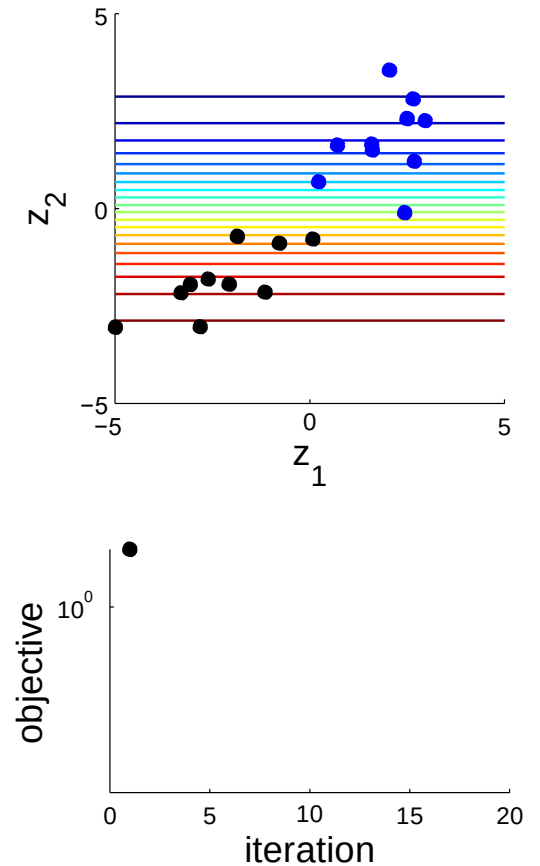
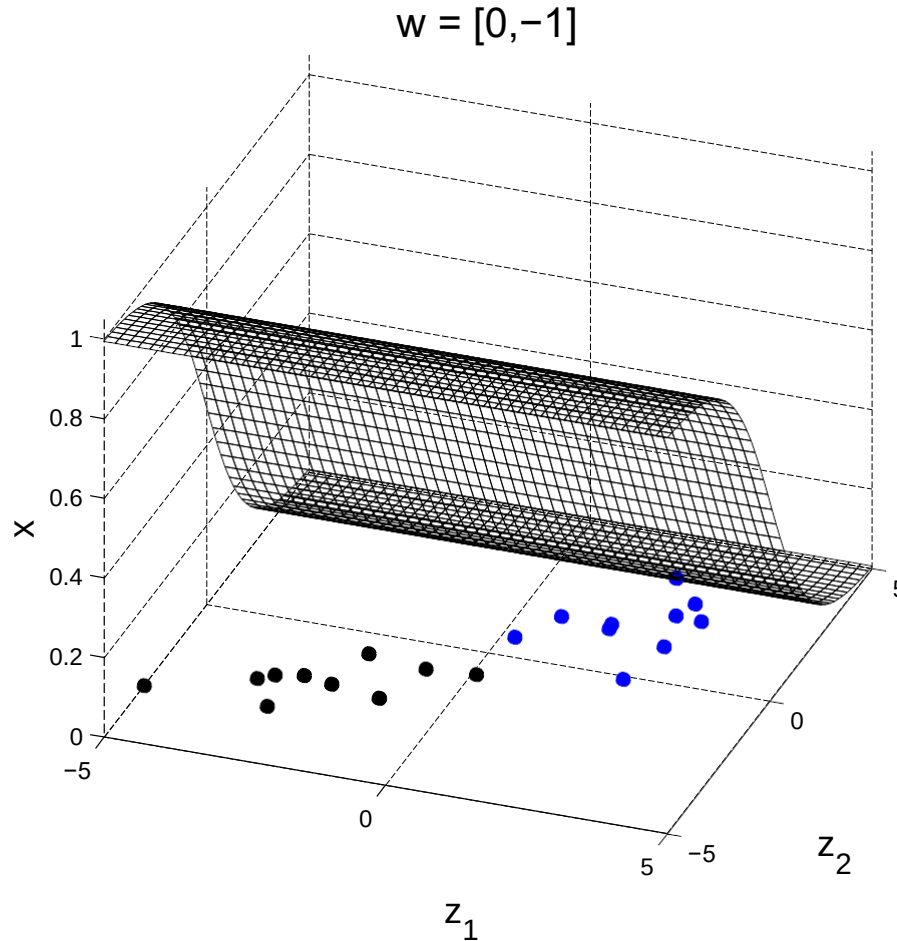
$$G(\mathbf{w}) = - \sum_n \left[t^{(n)} \log x(\mathbf{z}^{(n)}; \mathbf{w}) + (1 - t^{(n)}) \log (1 - x(\mathbf{z}^{(n)}; \mathbf{w})) \right] \geq 0$$

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

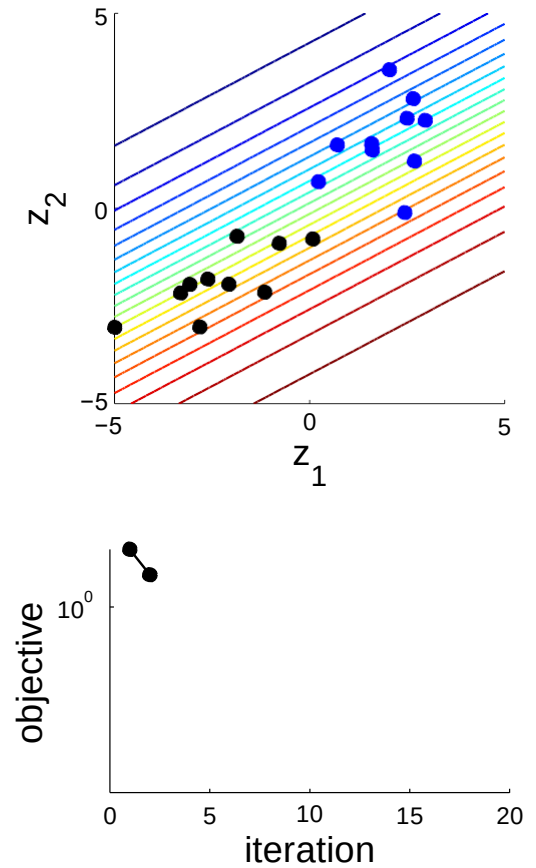
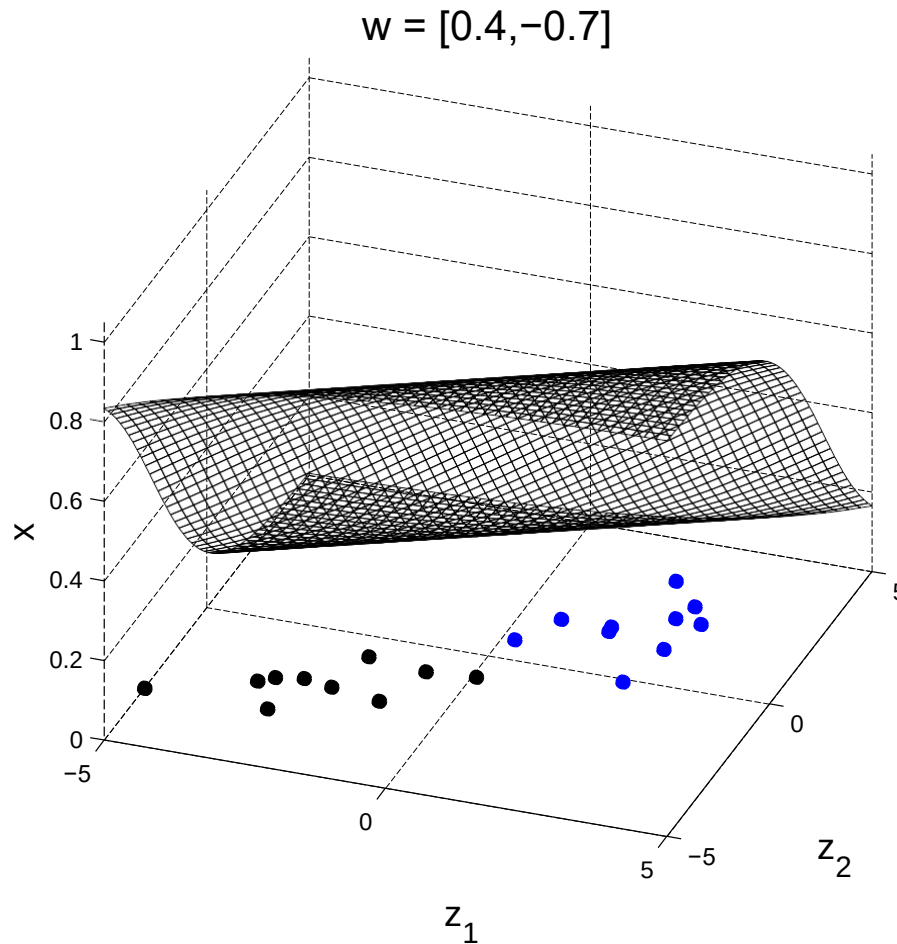
$$\frac{d}{d\mathbf{w}} G(\mathbf{w}) = \sum_n \frac{dG(\mathbf{w})}{dx^{(n)}} \frac{dx^{(n)}}{d\mathbf{w}} = - \sum_n (t^{(n)} - x^{(n)}) \mathbf{z}^{(n)} = \text{prediction error} \times \text{feature}$$

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{d}{d\mathbf{w}} G(\mathbf{w}) \quad \text{iteratively step down the objective (gradient points up hill)} \quad 37$$

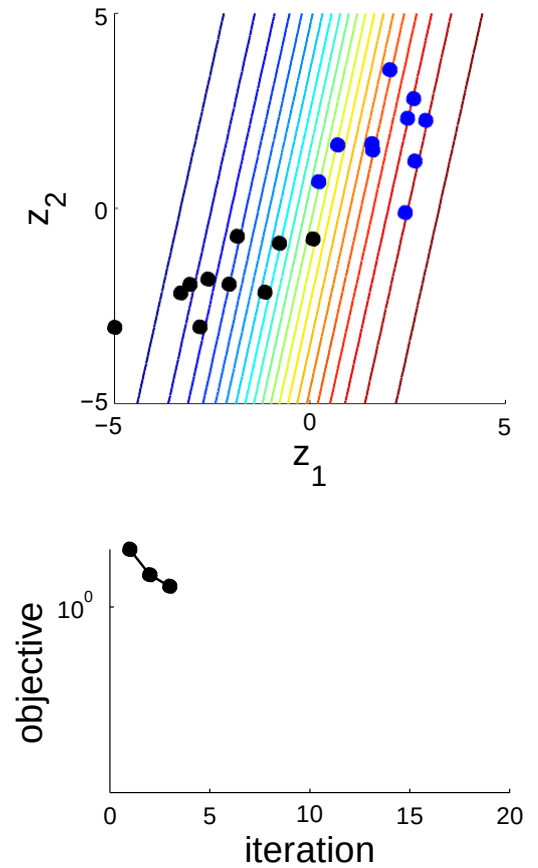
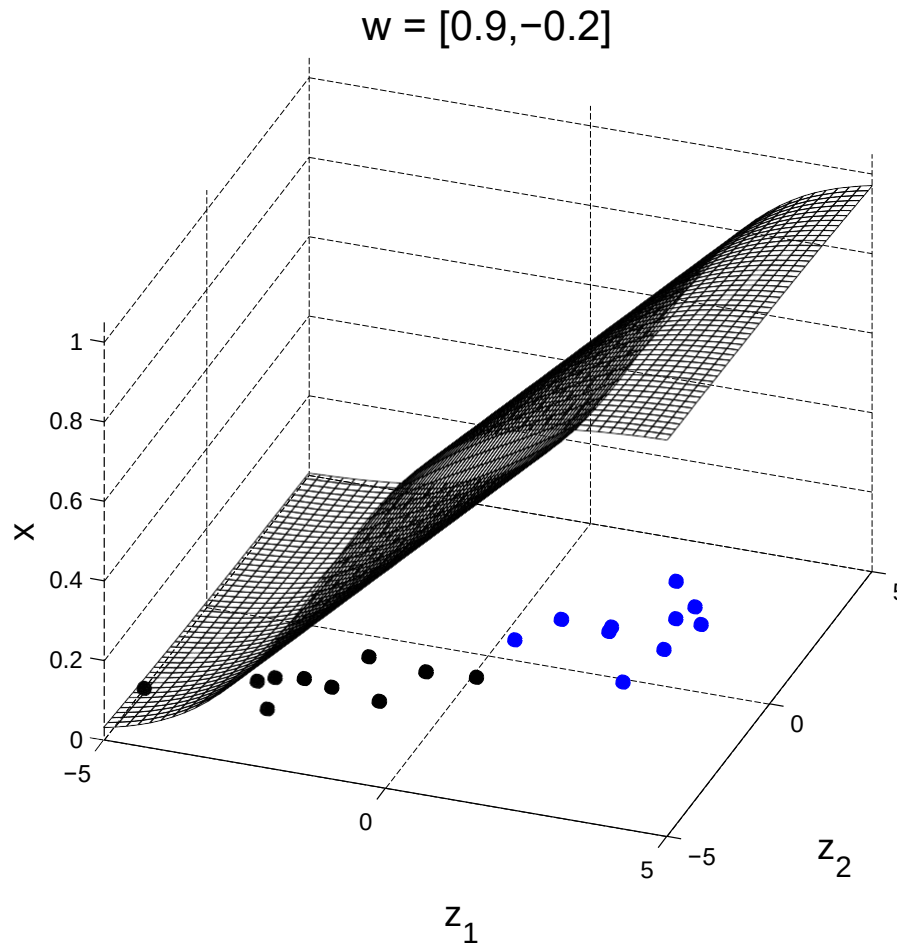
Training a Single Neuron



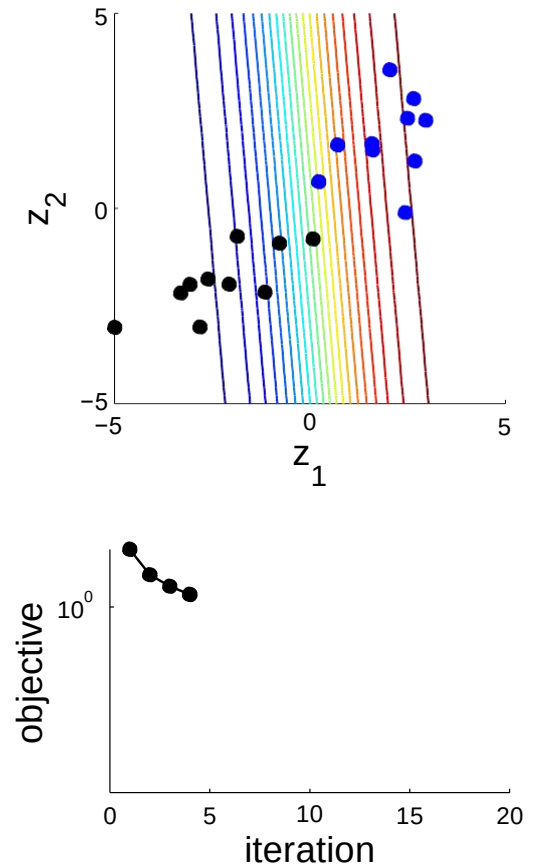
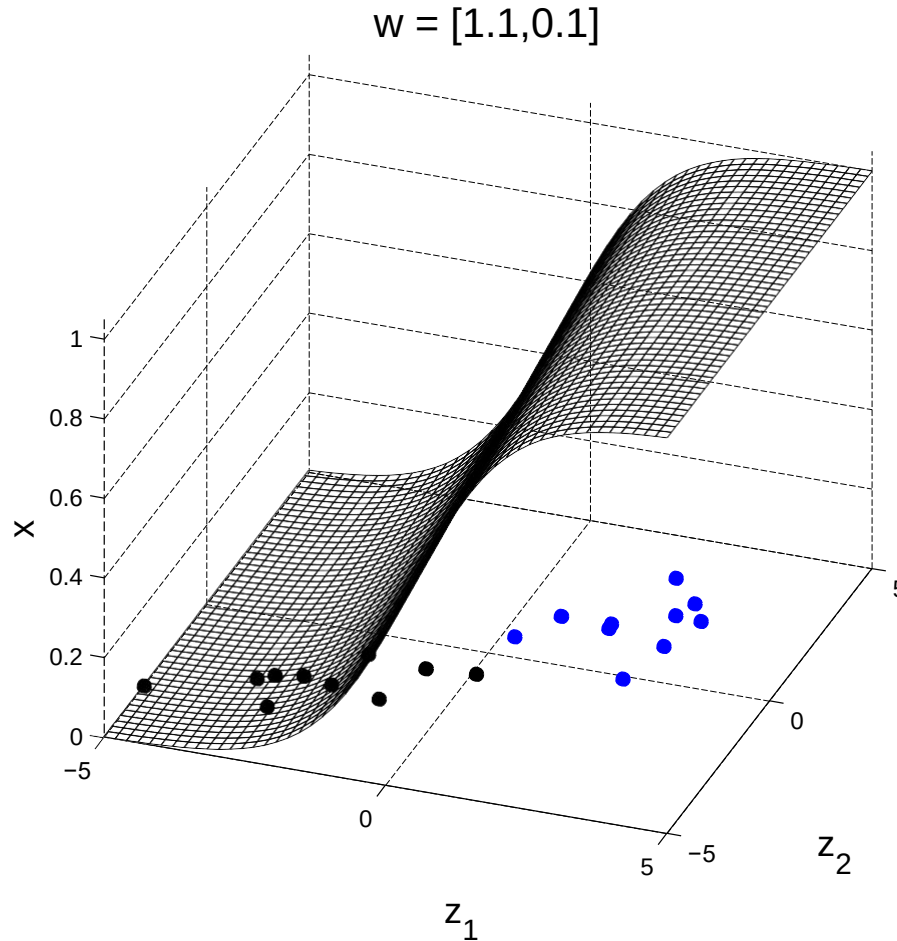
Training a Single Neuron



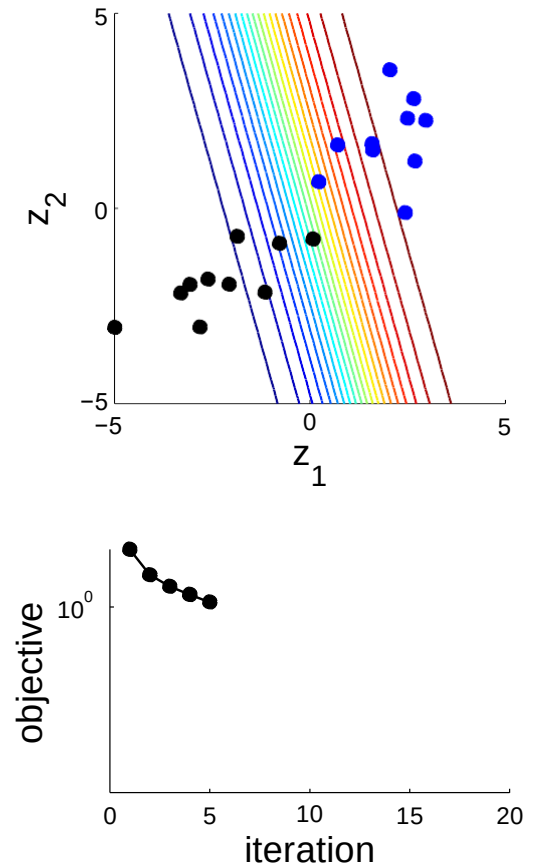
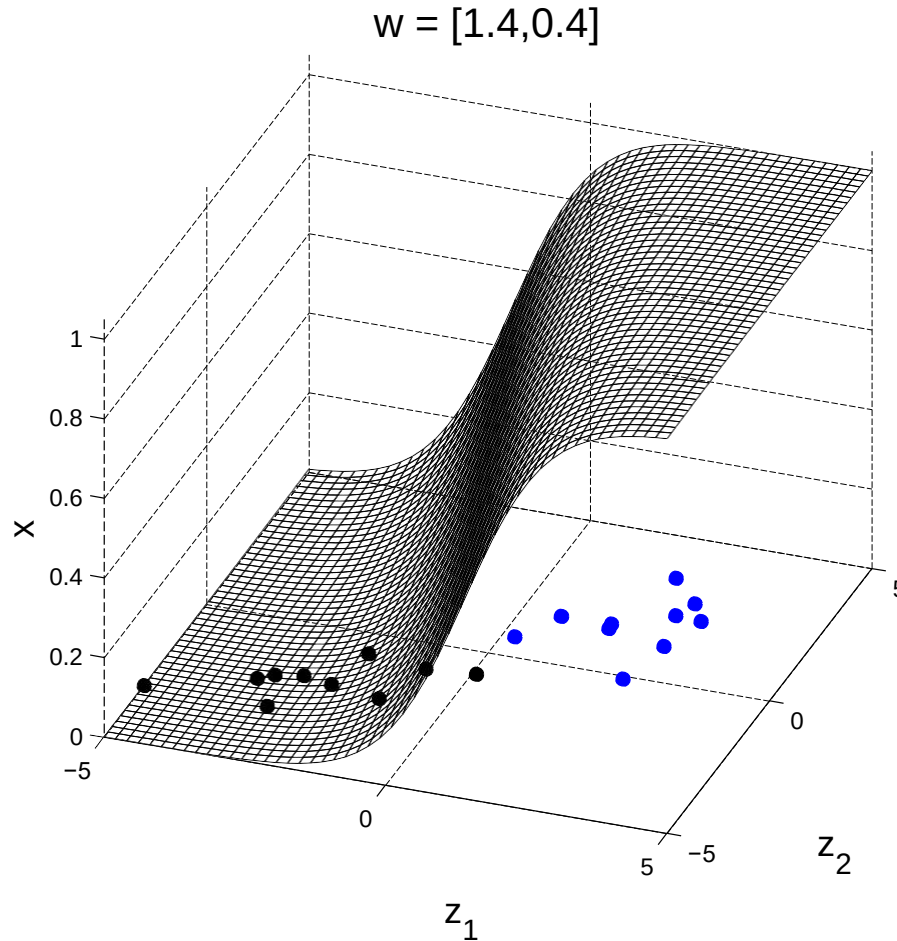
Training a Single Neuron



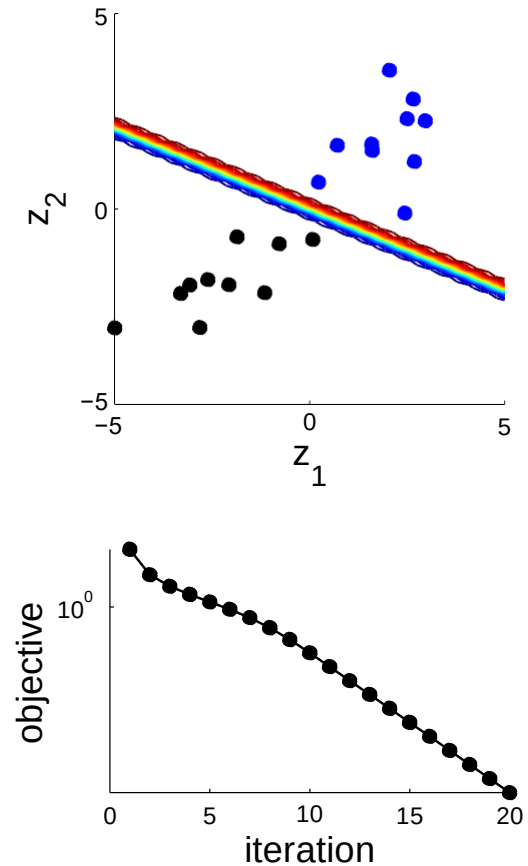
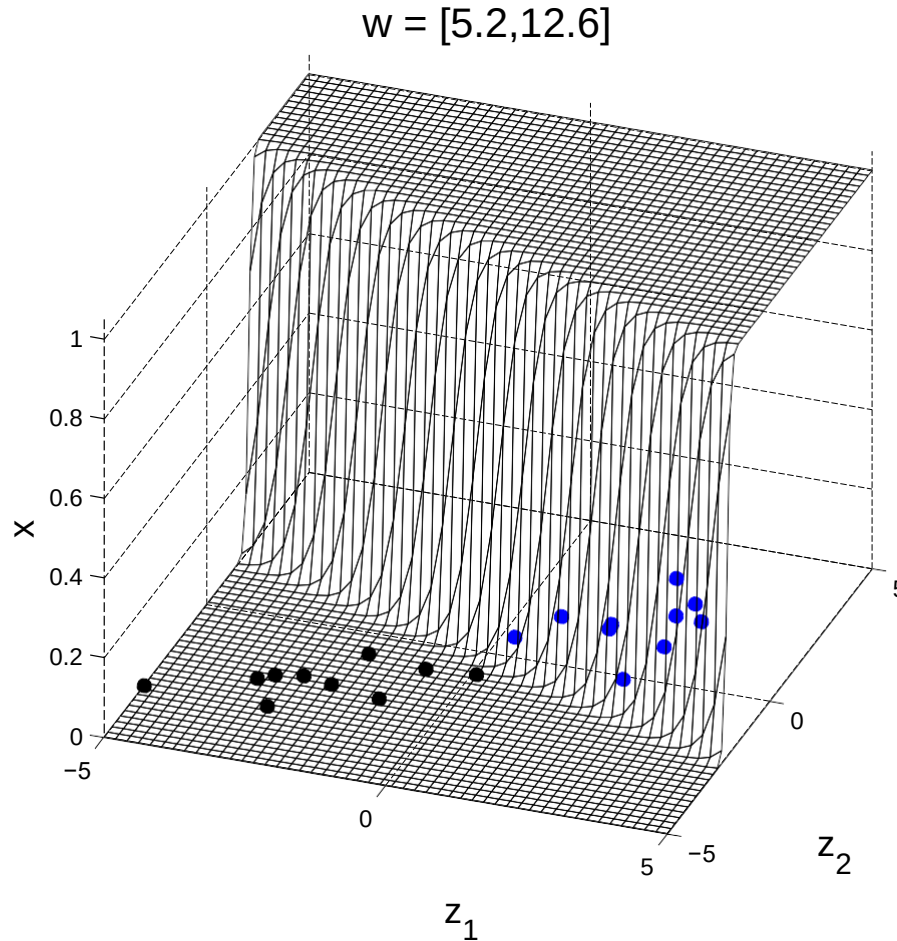
Training a Single Neuron



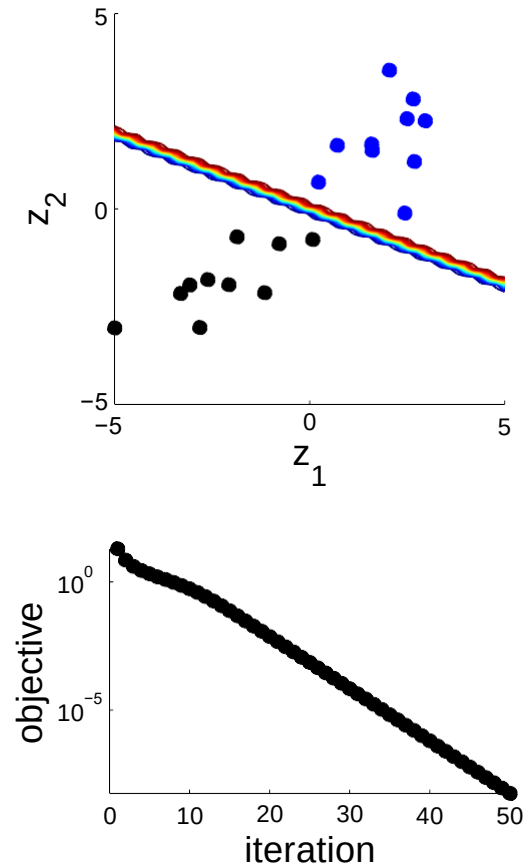
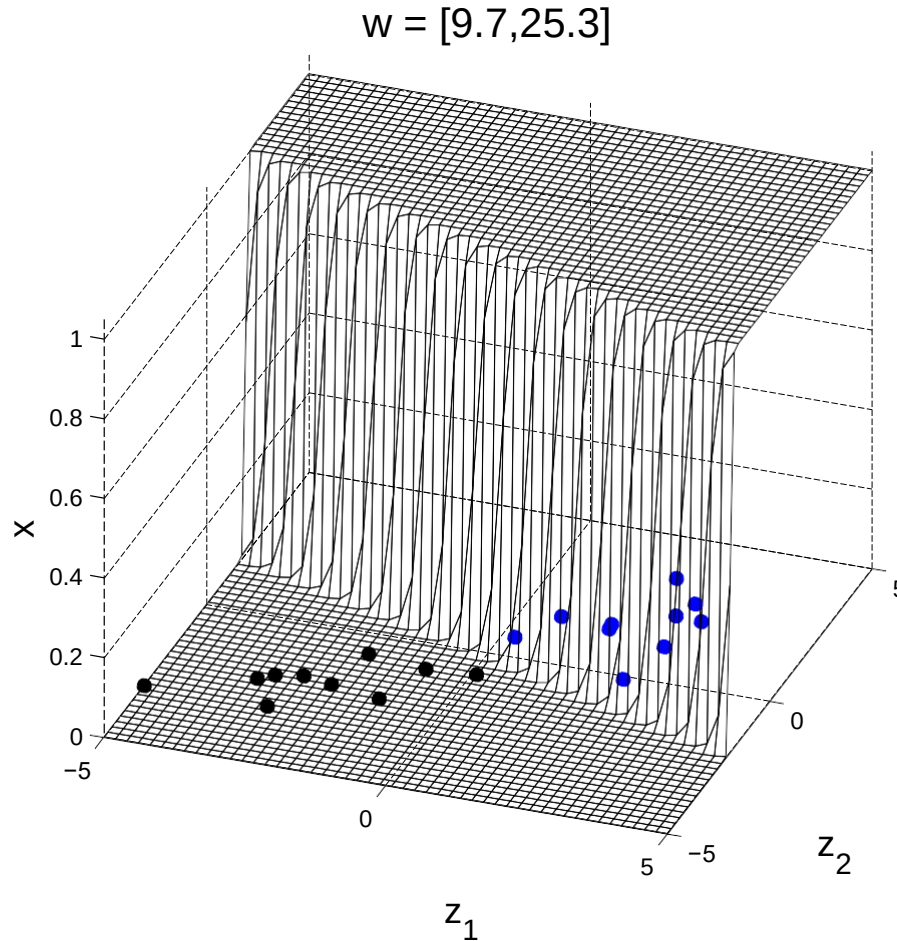
Training a Single Neuron



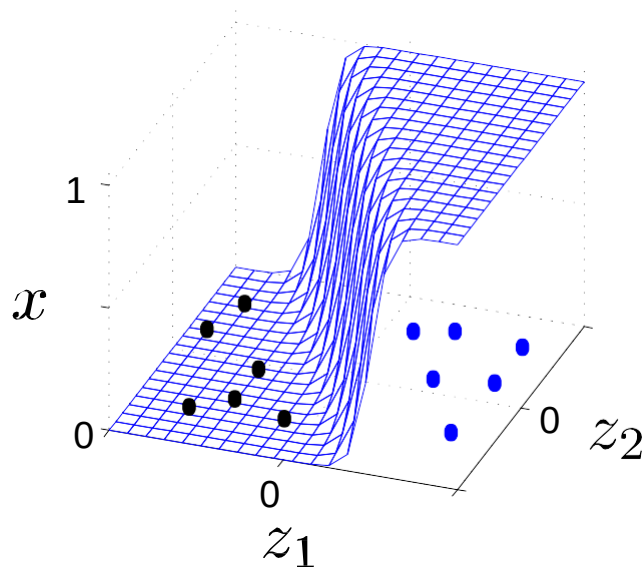
Training a Single Neuron



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:

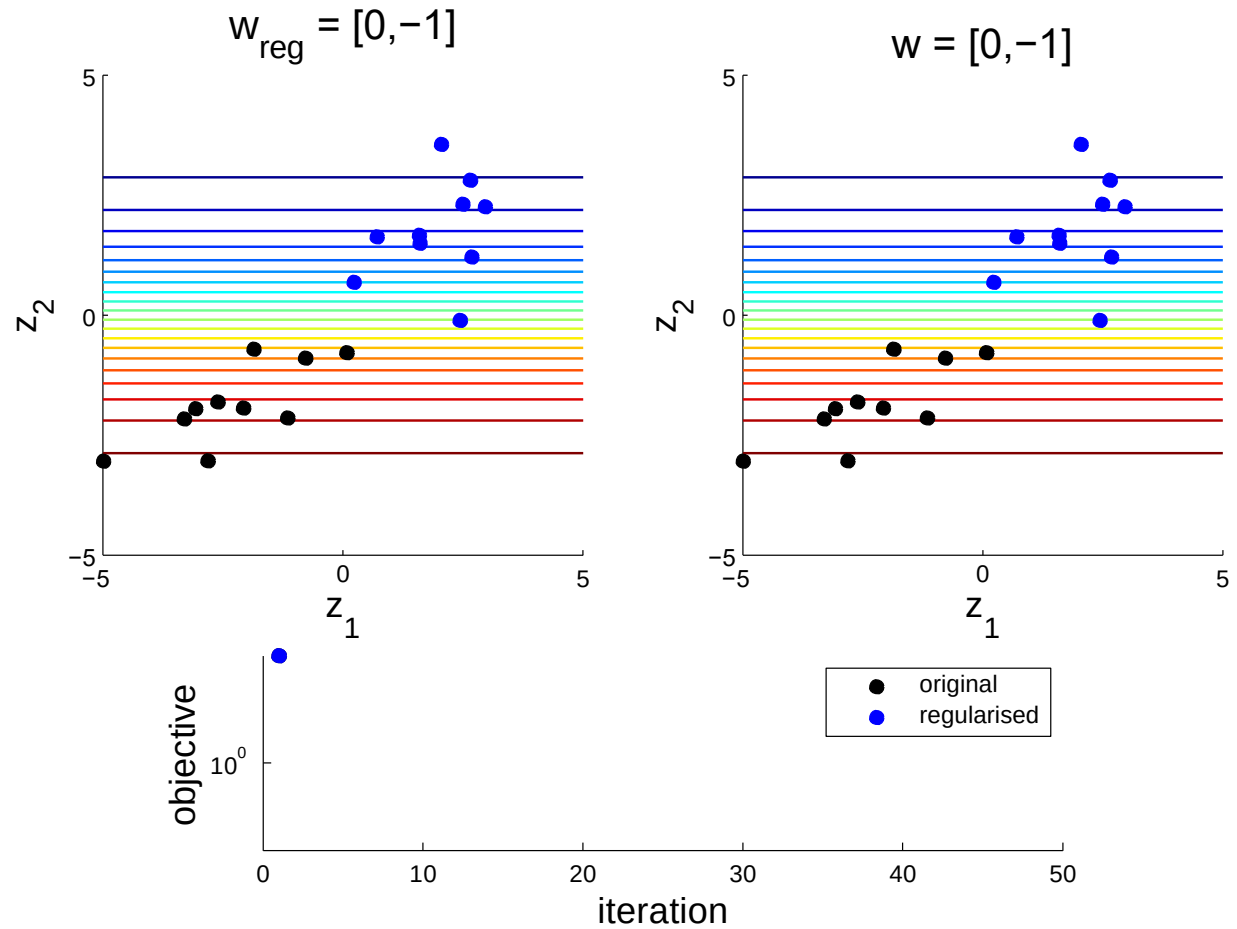
$$G(\mathbf{w}) = - \sum_n [t^{(n)} \log x(\mathbf{z}^{(n)}; \mathbf{w}) + (1 - t^{(n)}) \log (1 - x(\mathbf{z}^{(n)}; \mathbf{w}))]$$

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

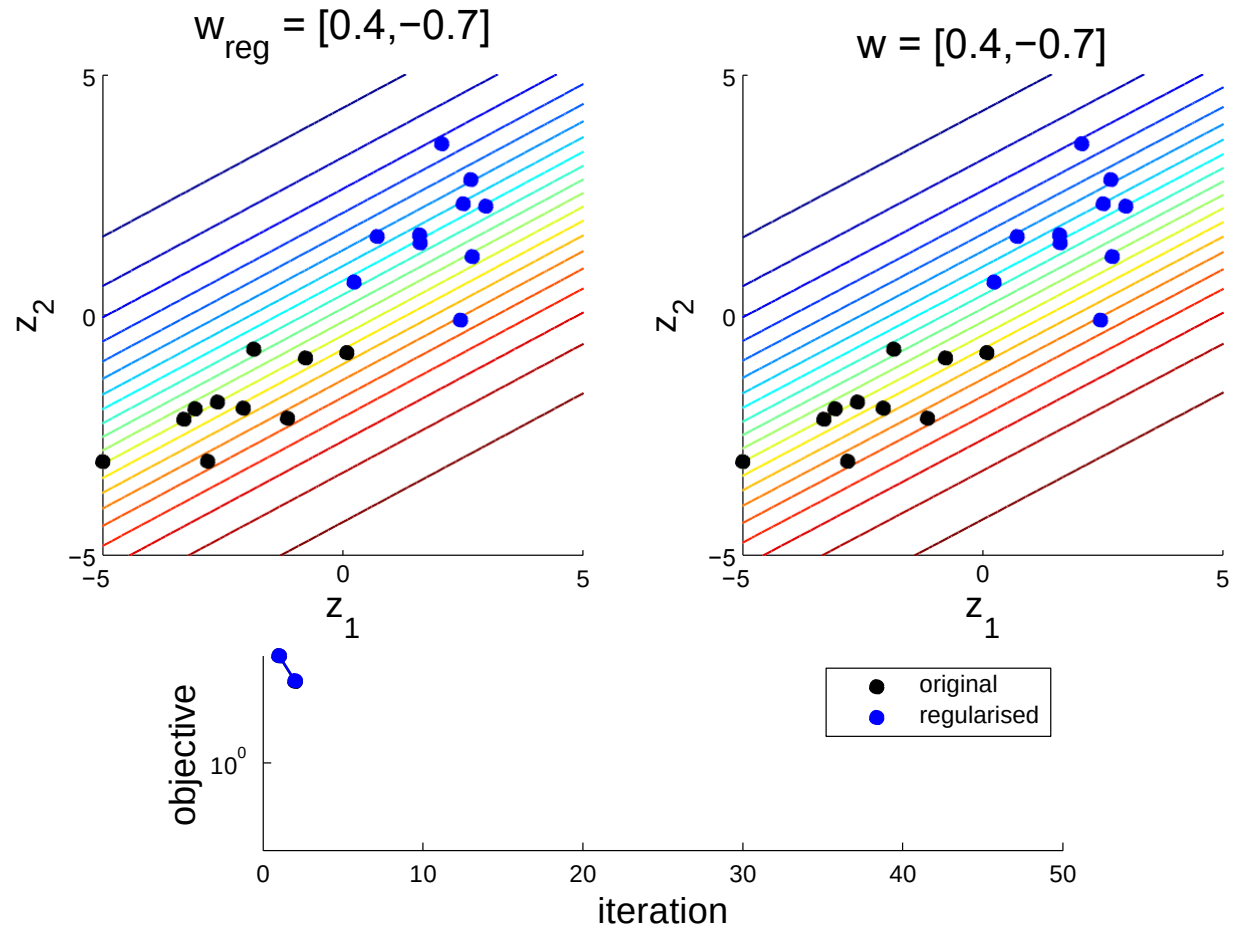
$$\mathbf{w}^* = \arg \min_{\mathbf{w}} M(\mathbf{w}) = \arg \min_{\mathbf{w}} [G(\mathbf{w}) + \alpha E(\mathbf{w})]$$

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

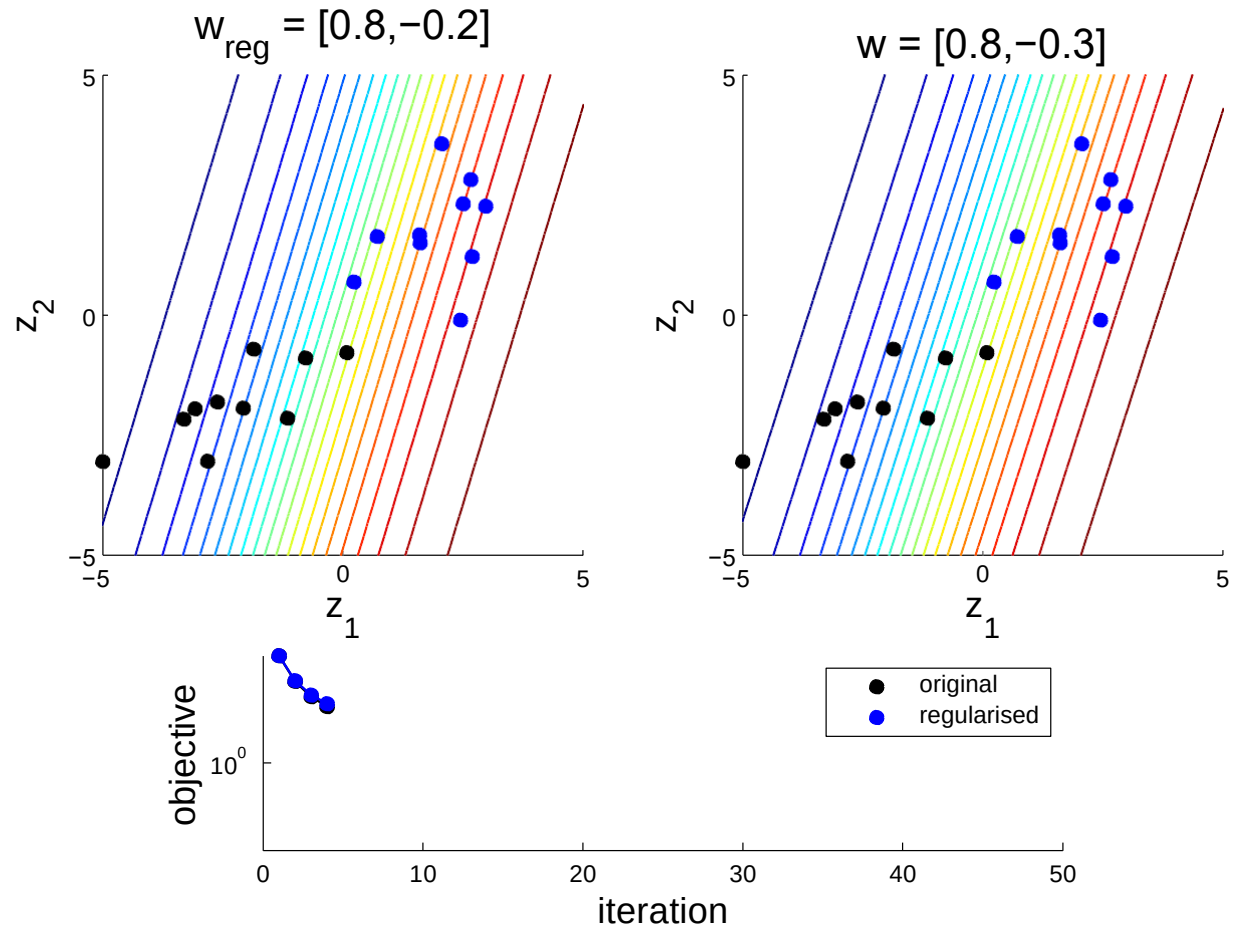
Training a Single Neuron (cont'd)



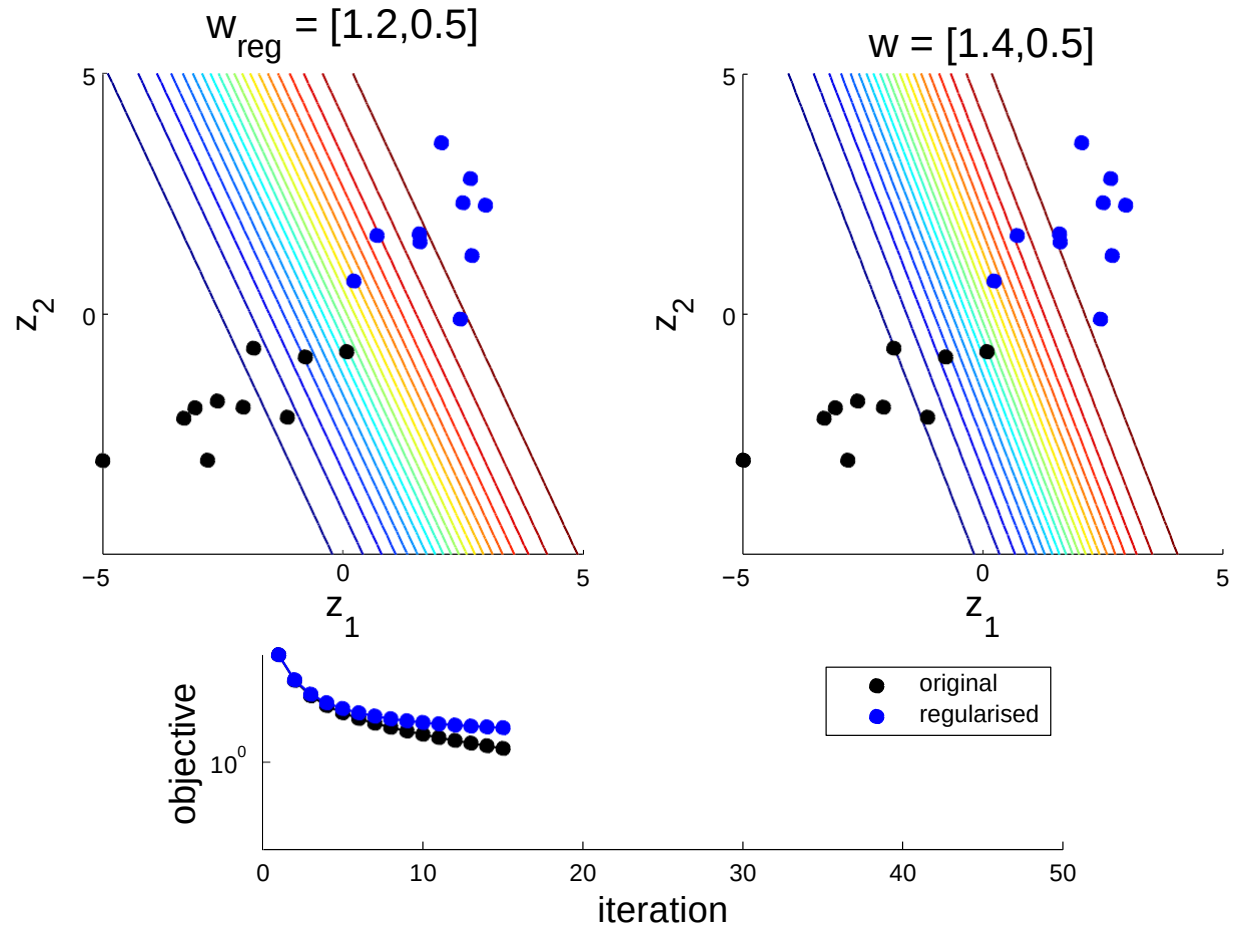
Training a Single Neuron (cont'd)



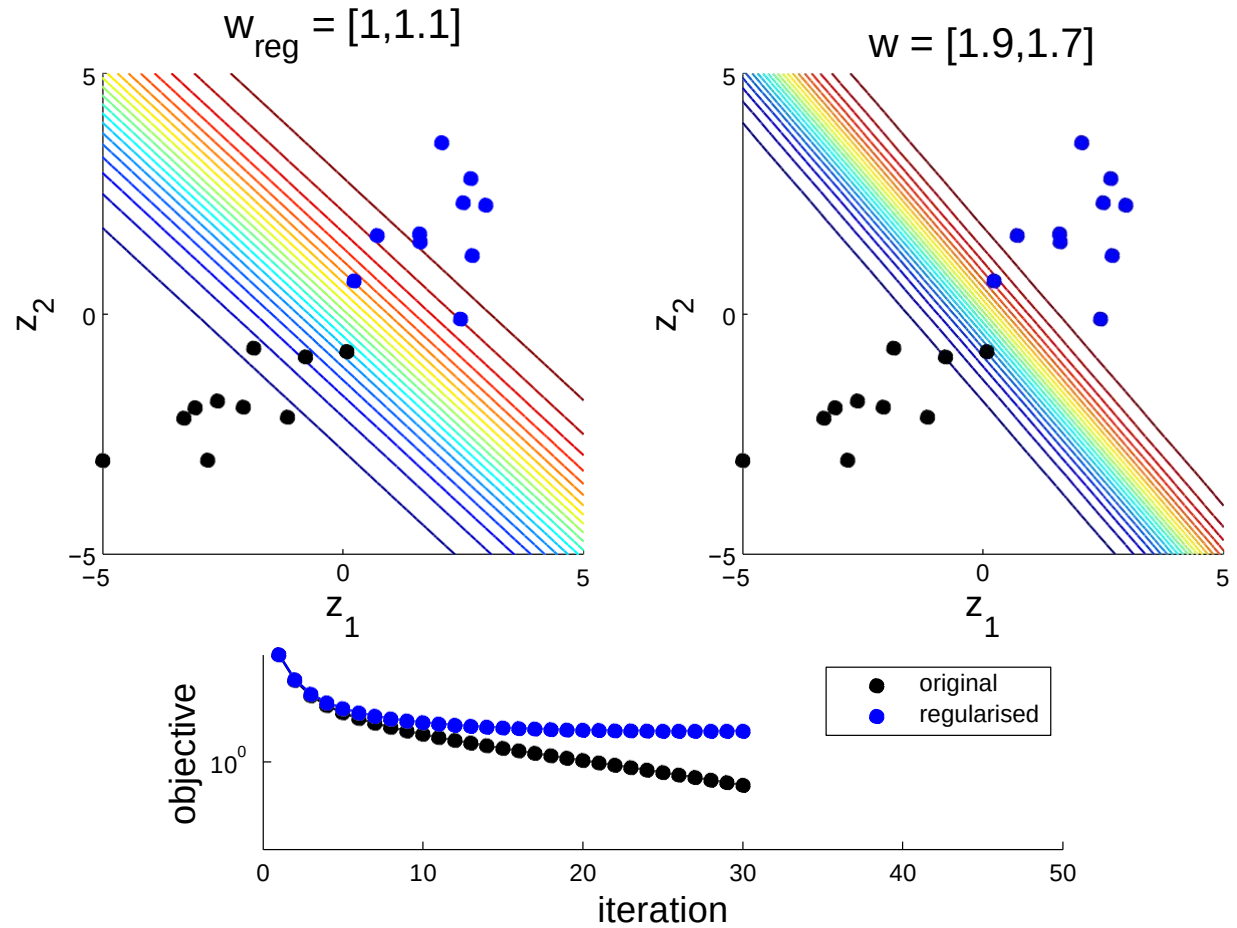
Training a Single Neuron (cont'd)



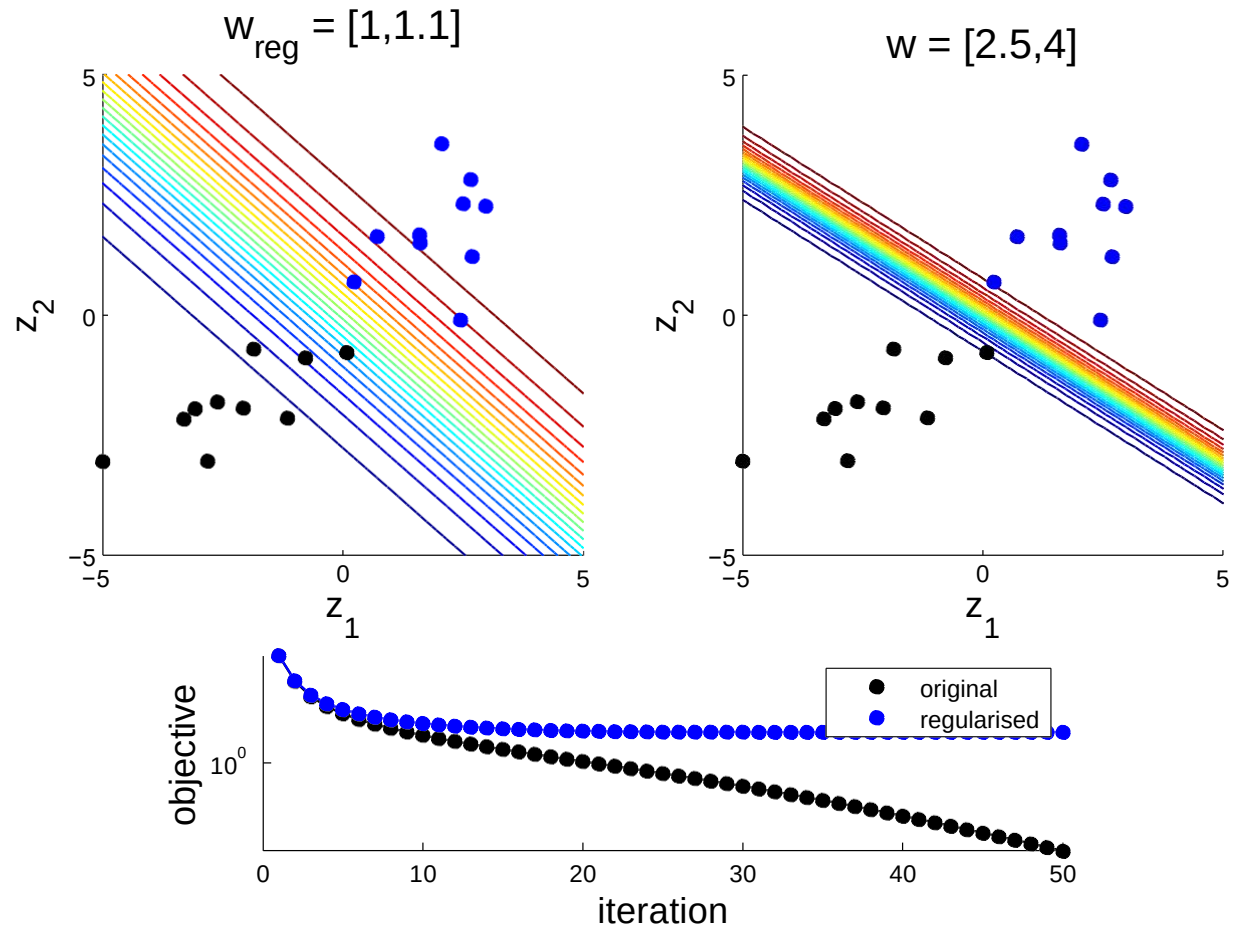
Training a Single Neuron (cont'd)



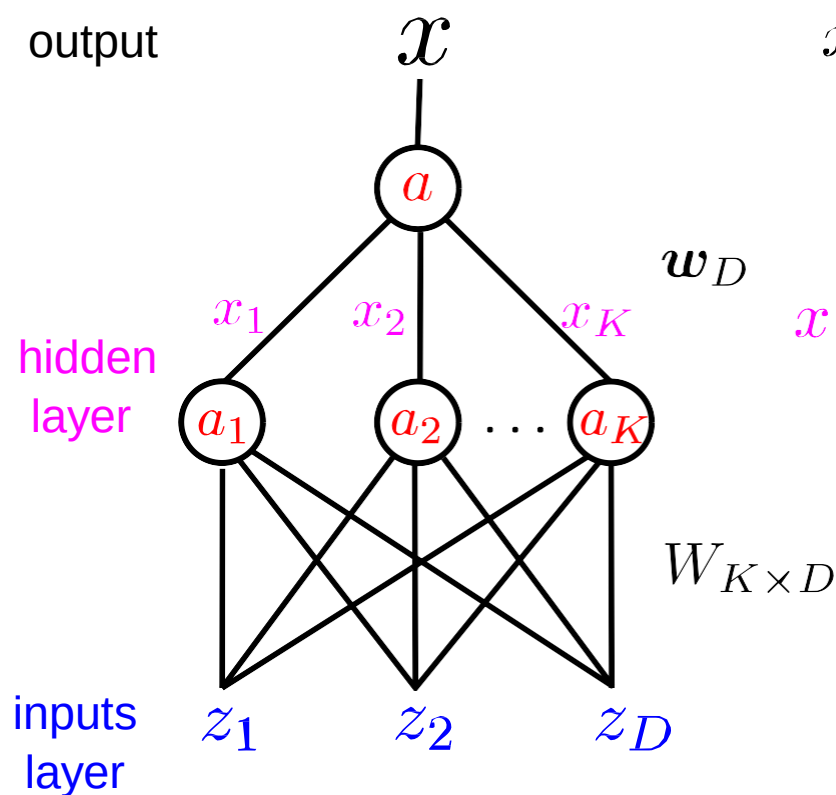
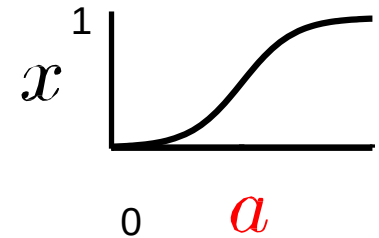
Training a Single Neuron (cont'd)



Training a Single Neuron (cont'd)



Single Hidden Layer Neural Networks



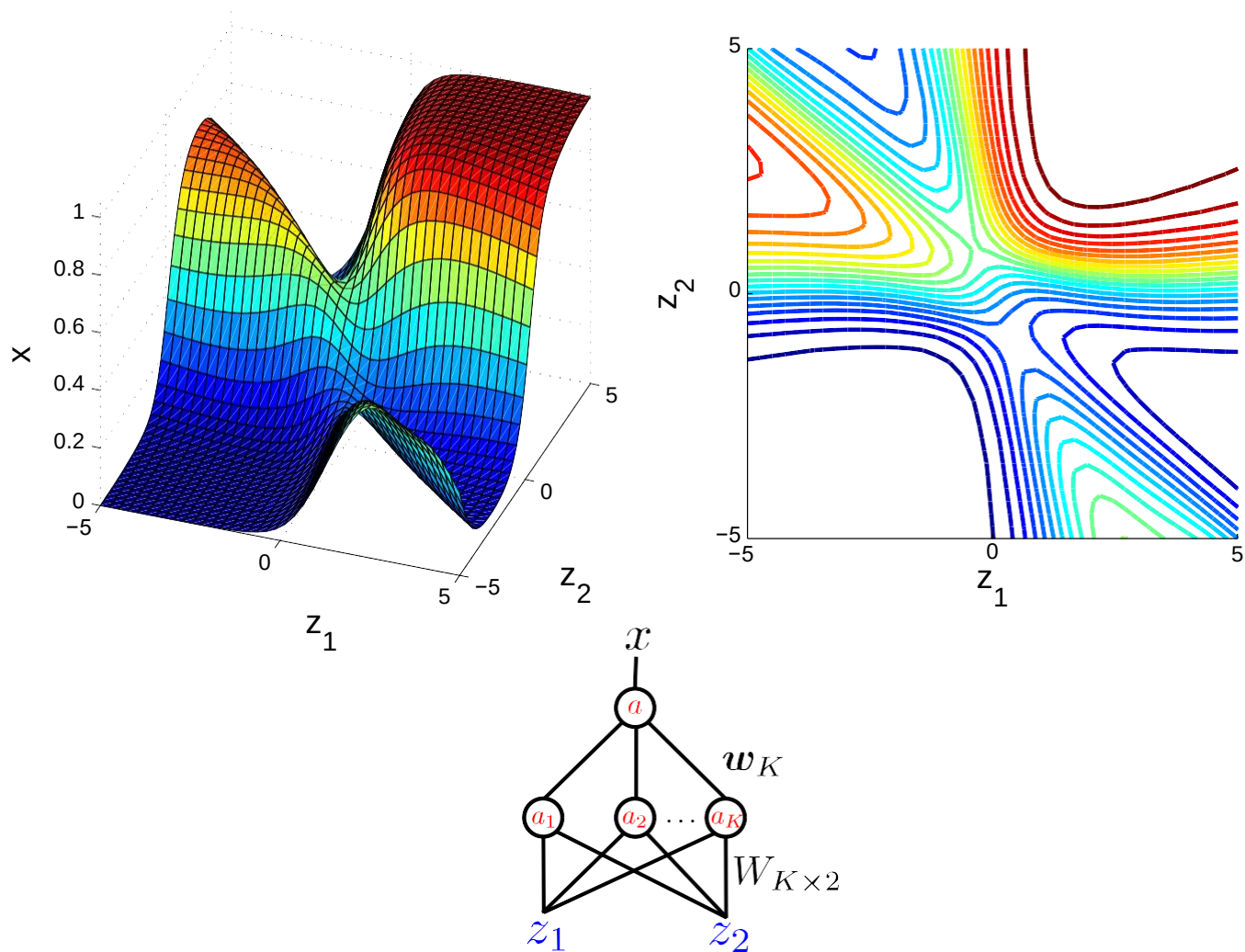
$$x(a) = \frac{1}{1 + \exp(-a)}$$

$$a = \sum_{k=1}^K w_k x_k$$

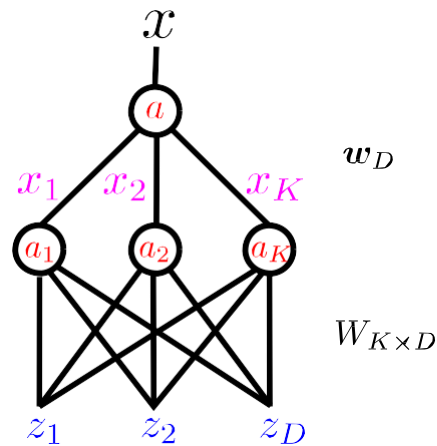
$$x(a_k) = \frac{1}{1 + \exp(-a_k)}$$

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

Sampling Random Neural Network Classifiers



Training a Neural Network with a Single Hidden Layer



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

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

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

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

objective function:

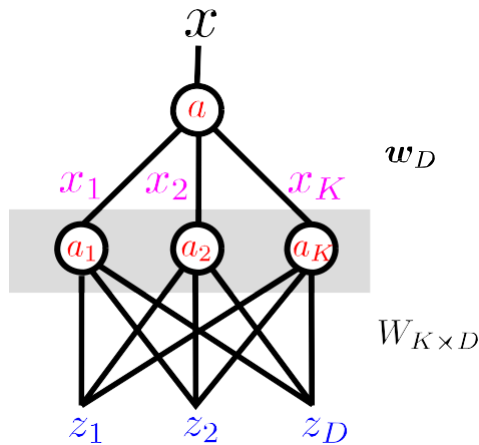
$$G(W, \mathbf{w}) = - \sum_n [t^{(n)} \log \mathbf{x}^{(n)} + (1 - t^{(n)}) \log (1 - \mathbf{x}^{(n)})] \text{ likelihood same as before}$$

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

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

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:

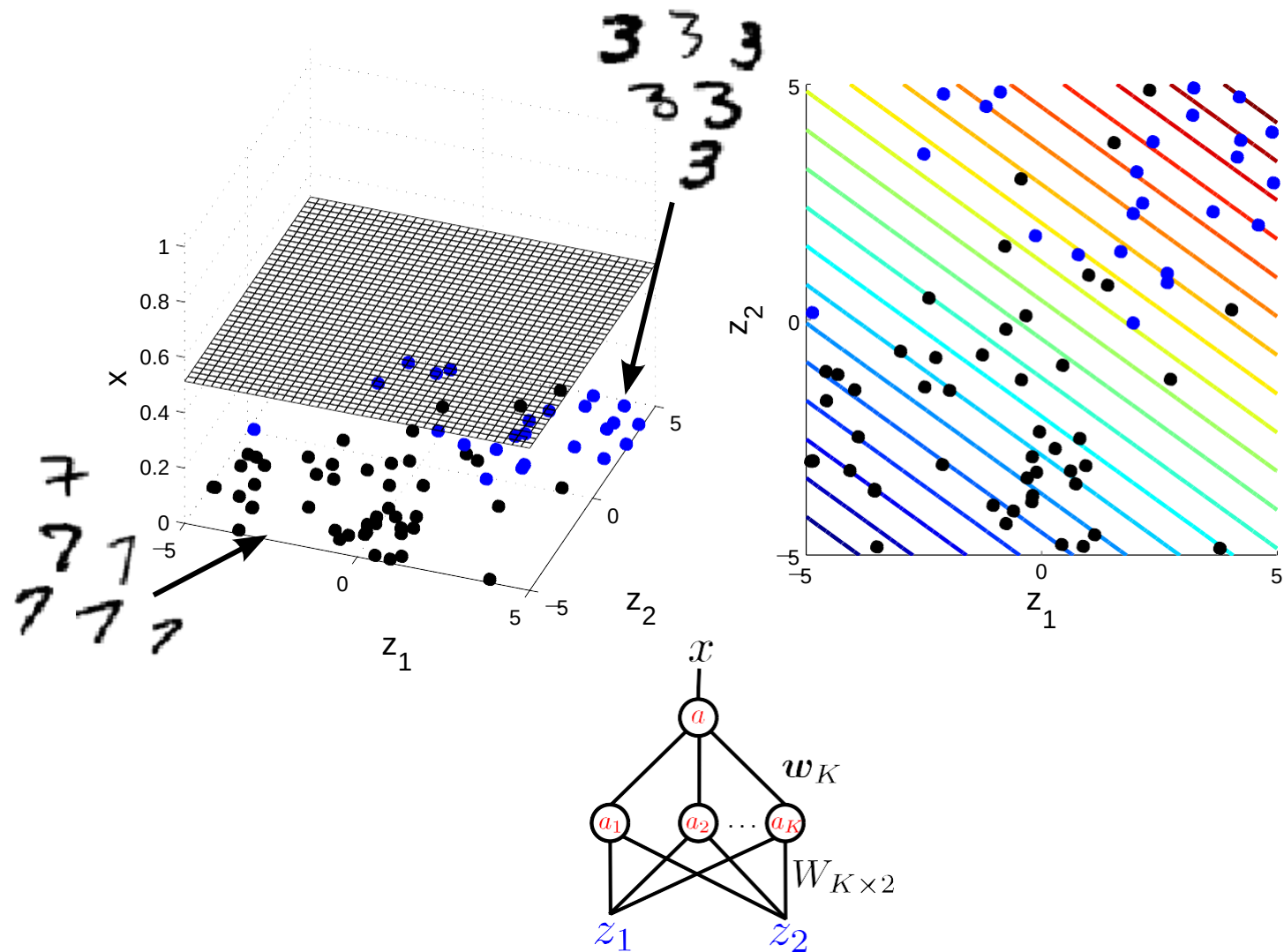
$$G(W, \mathbf{w}) = - \sum_n [t^{(n)} \log x^{(n)} + (1 - t^{(n)}) \log (1 - x^{(n)})] \text{ likelihood same as before}$$

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

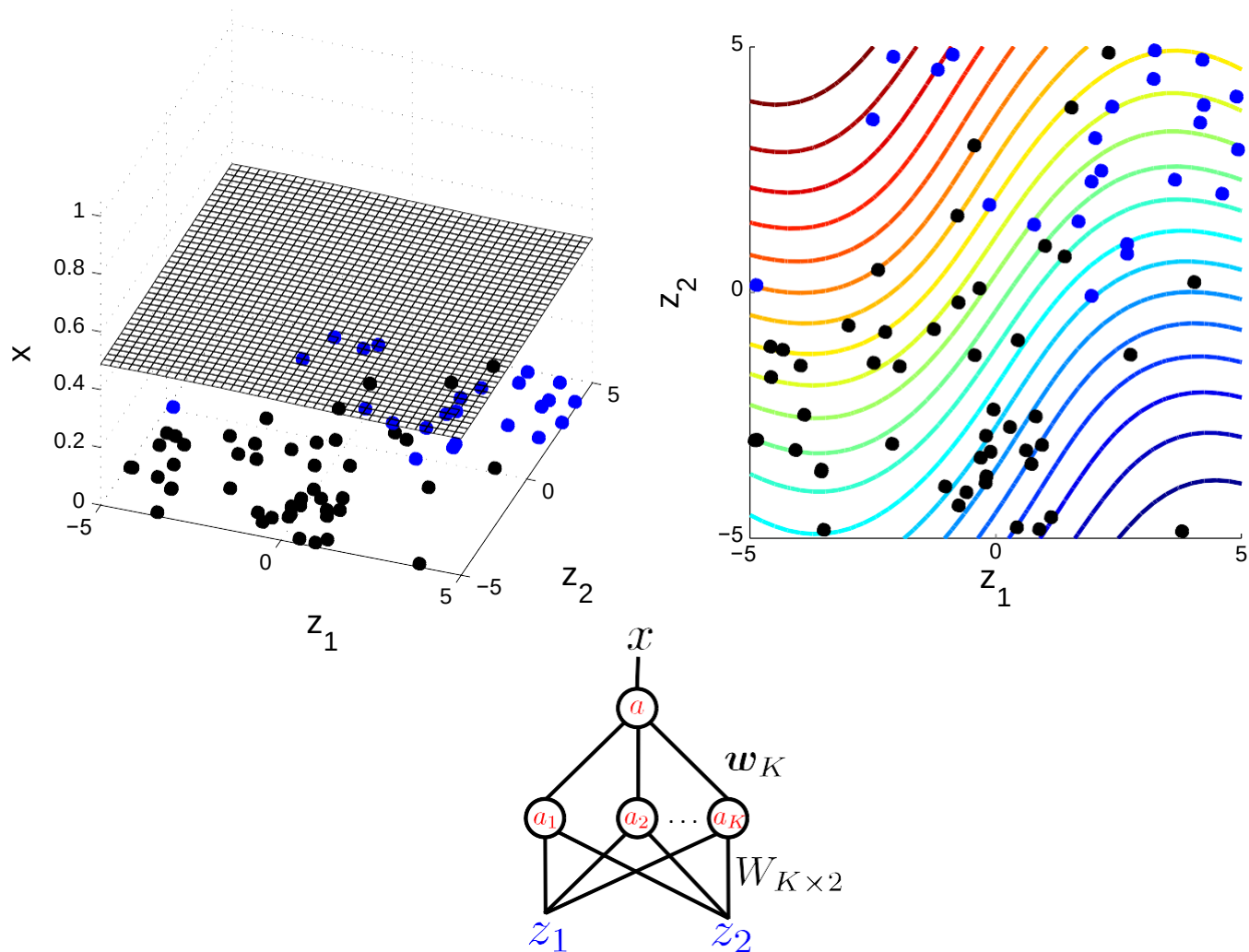
$$\{W, \mathbf{w}^*\} = \arg \min_{W, \mathbf{w}} M(W, \mathbf{w}) = \arg \min_{W, \mathbf{w}} [G(W, \mathbf{w}) + \alpha E(W, \mathbf{w})]$$

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

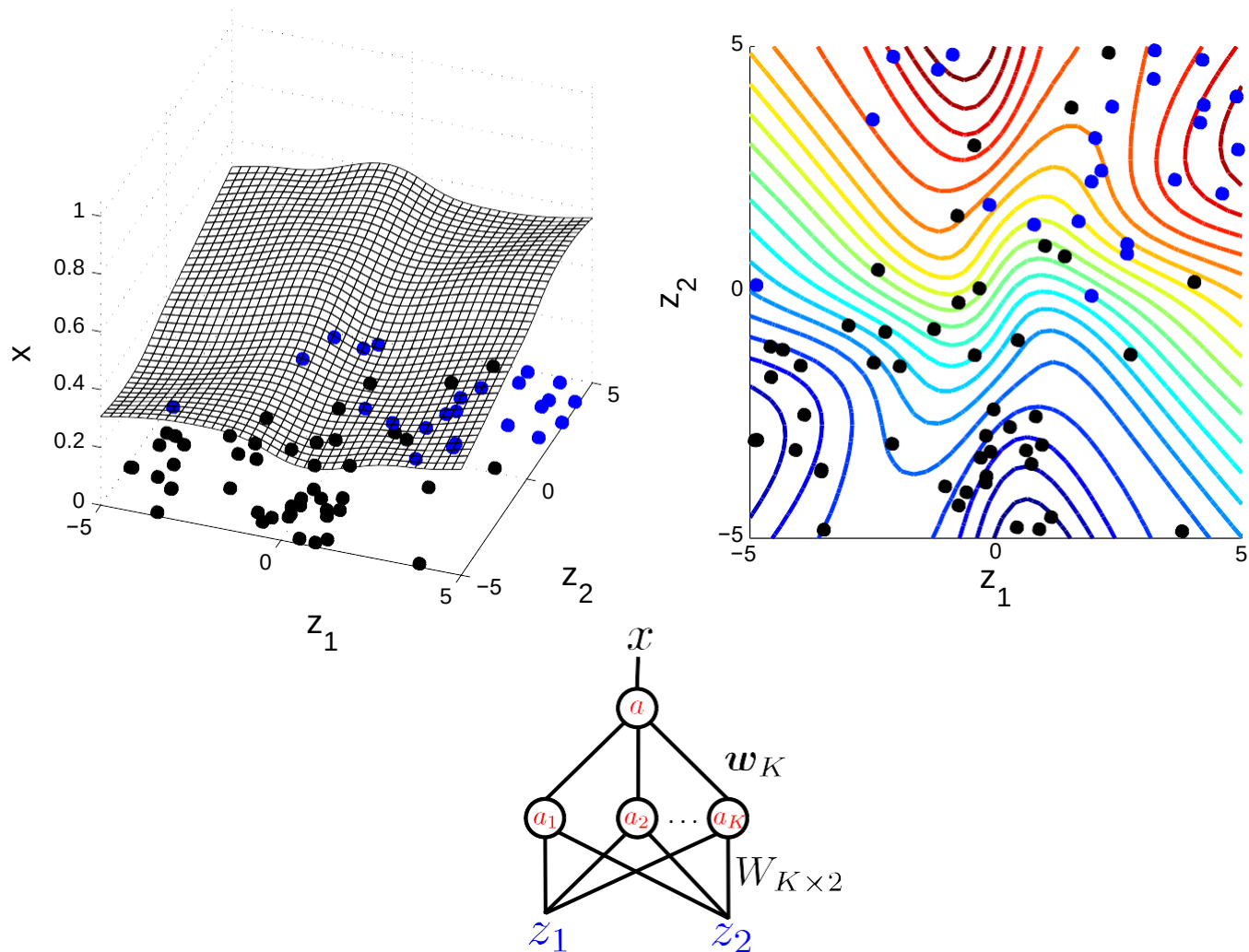
Training a Neural Network with a Single Hidden Layer



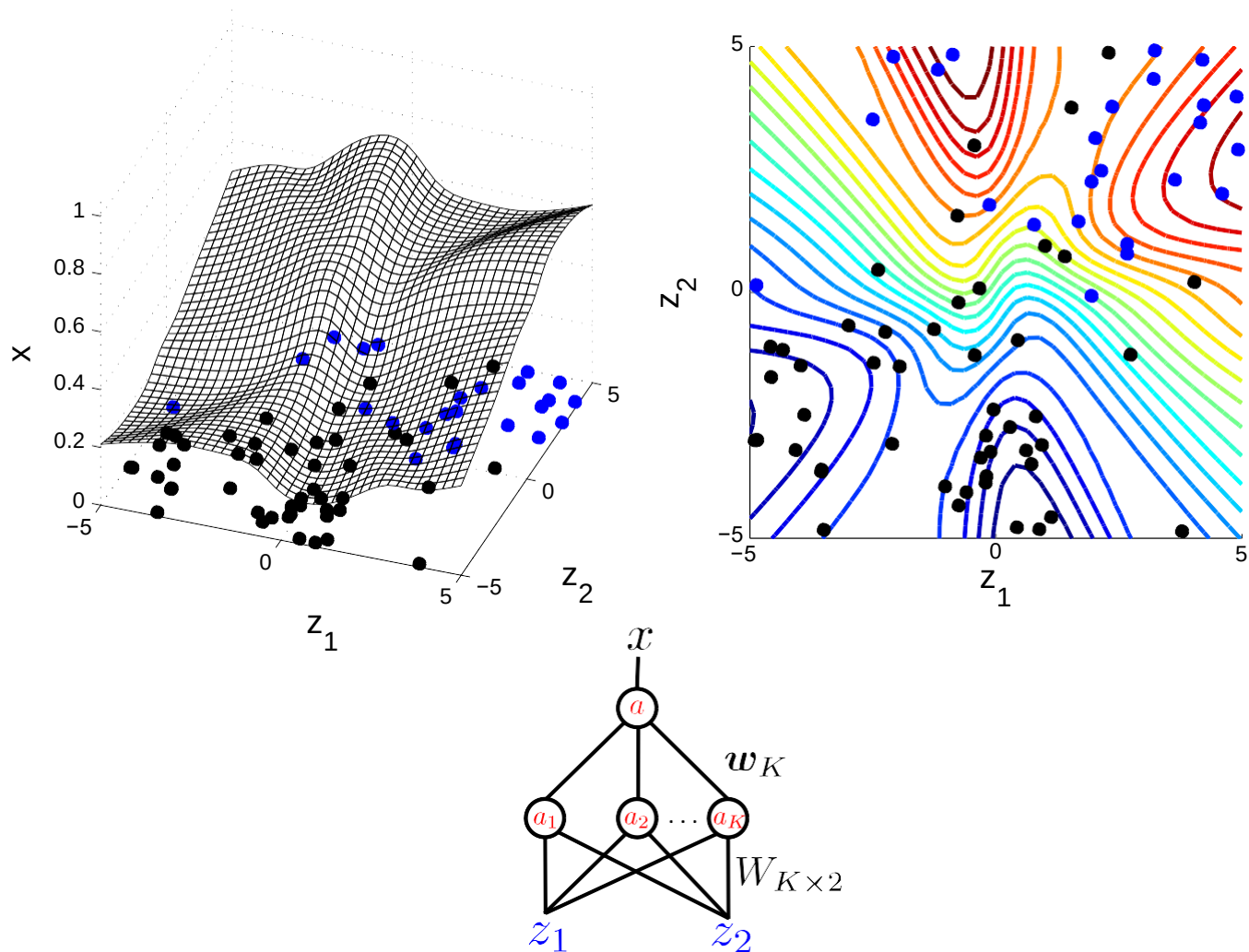
Training a Neural Network with a Single Hidden Layer



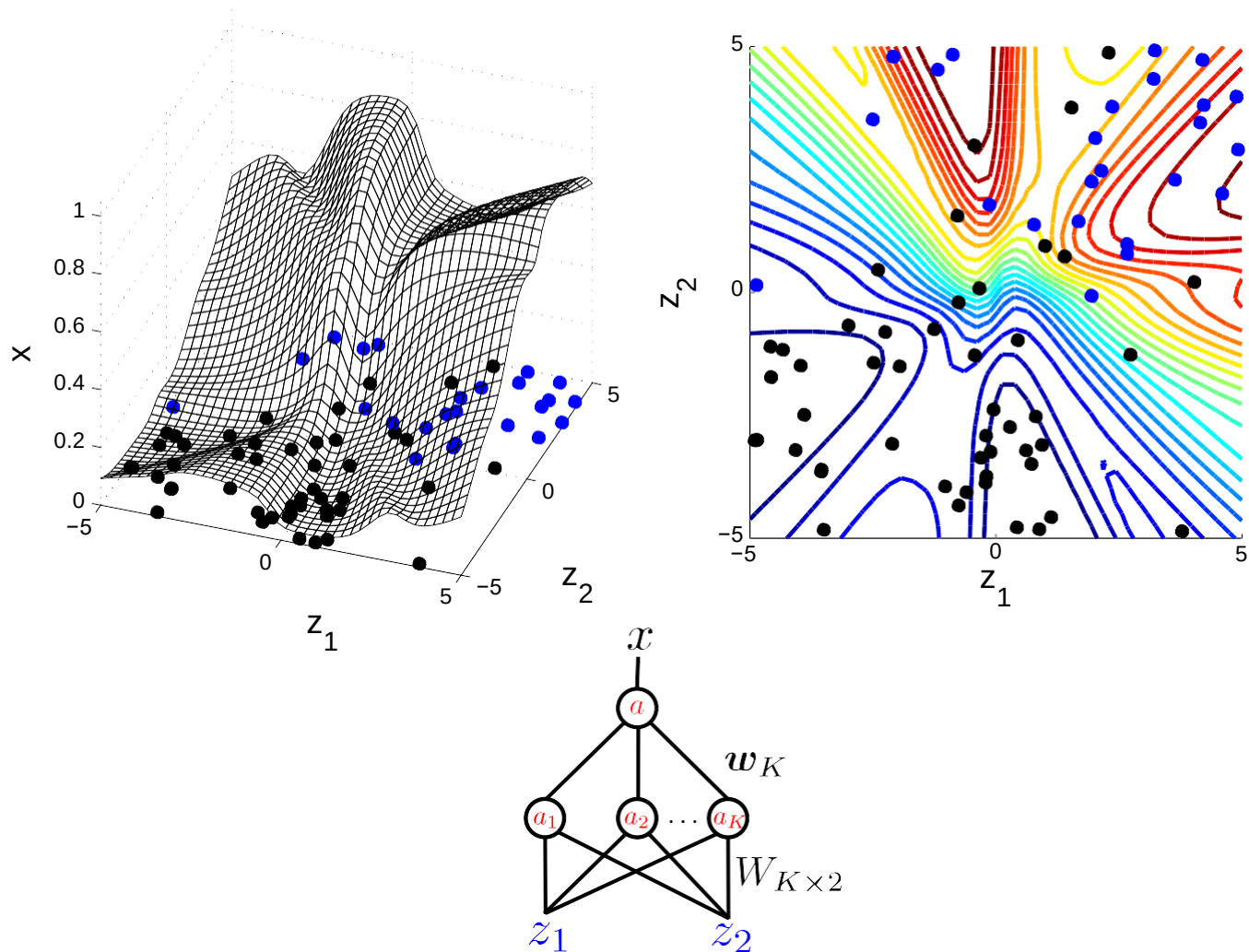
Training a Neural Network with a Single Hidden Layer



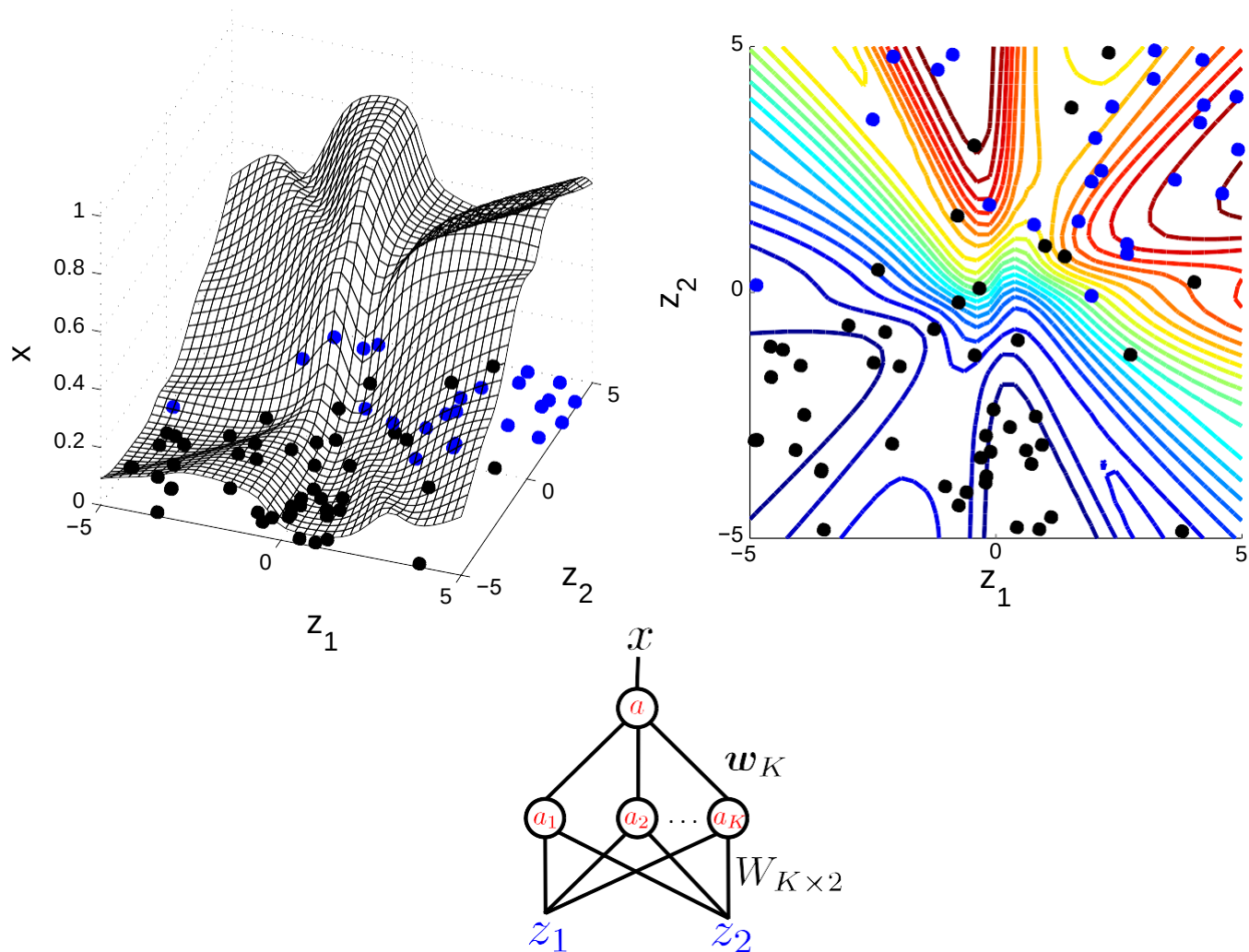
Training a Neural Network with a Single Hidden Layer



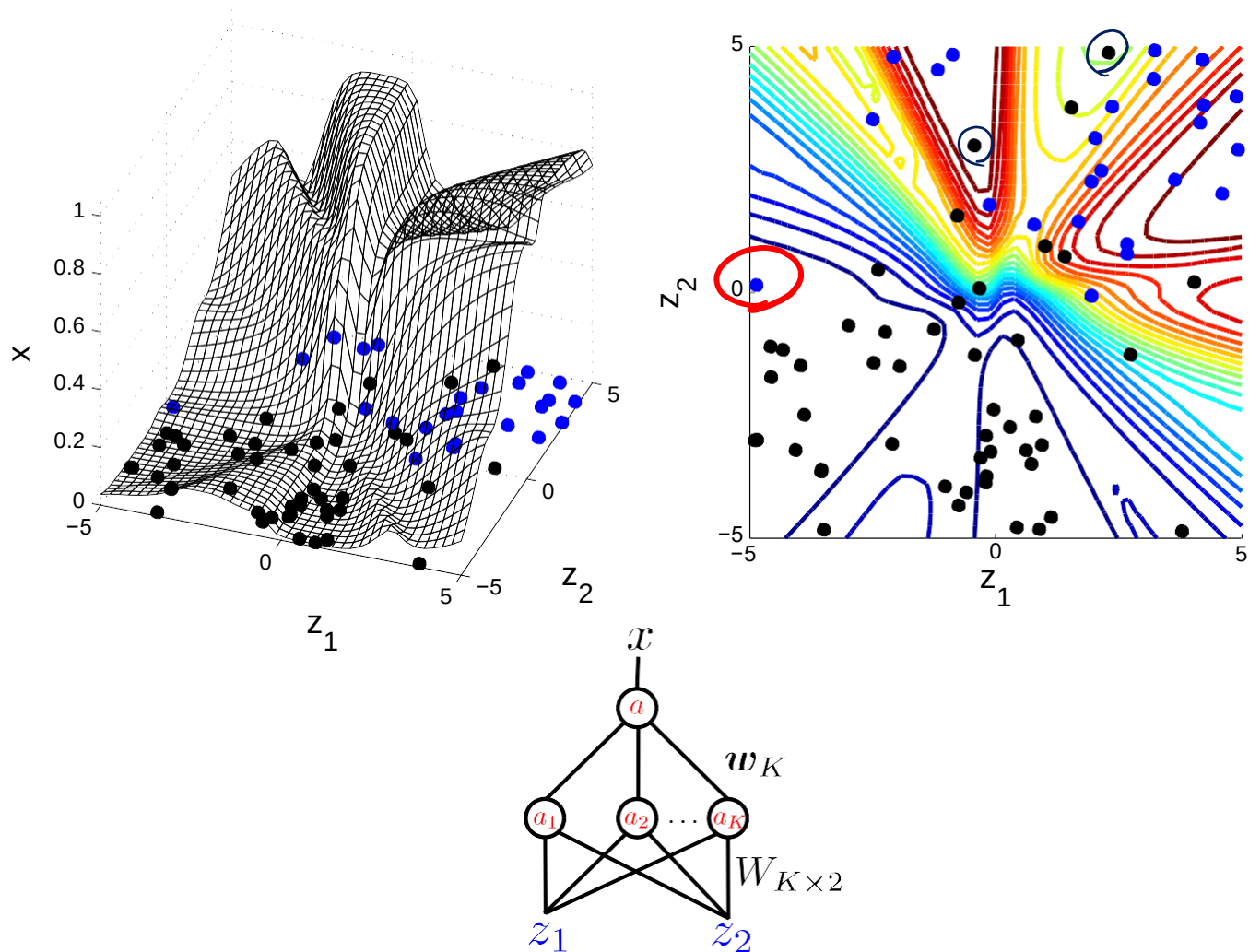
Training a Neural Network with a Single Hidden Layer



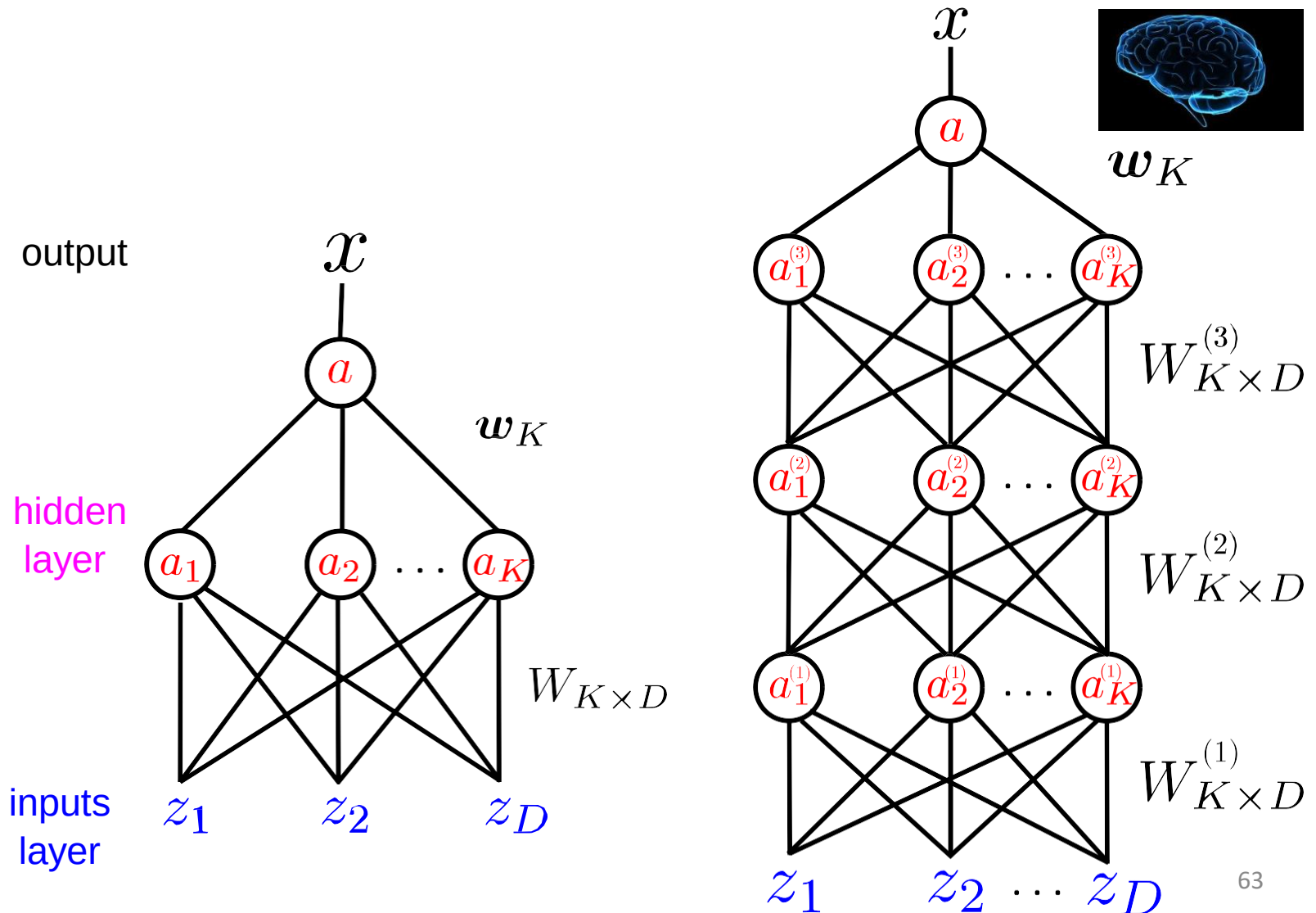
Training a Neural Network with a Single Hidden Layer



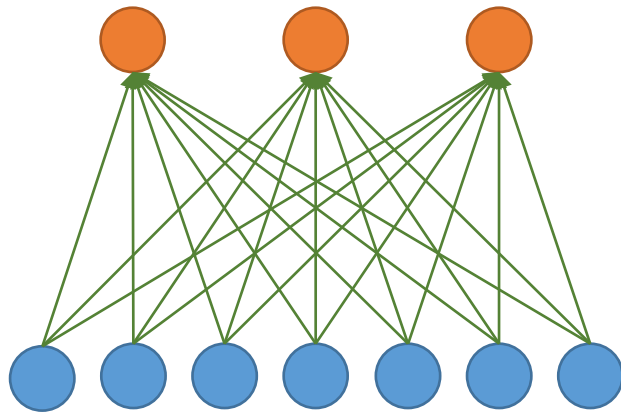
Training a Neural Network with a Single Hidden Layer



Hierarchical Models with Many Layers



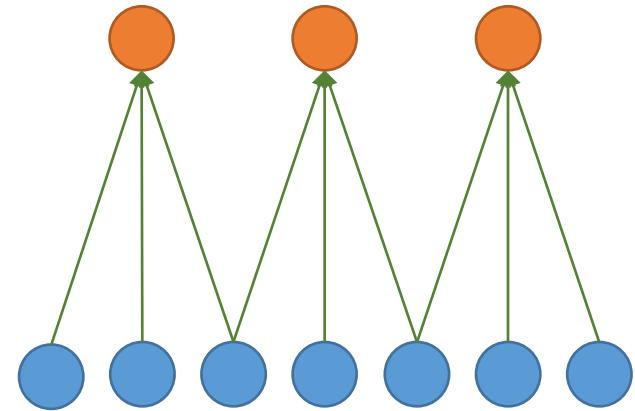
Convolutional Neural Networks (CNN): Local Connectivity



Hidden layer

Input layer

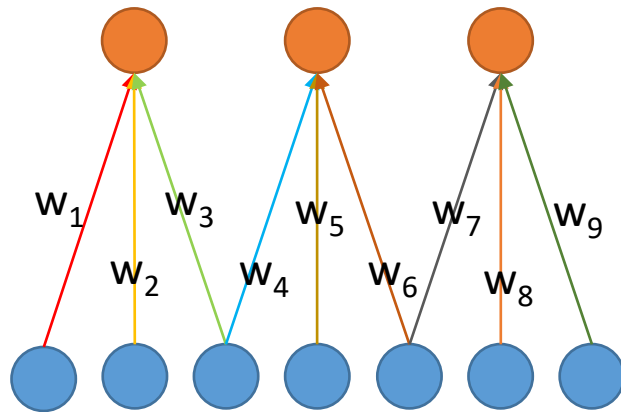
Global connectivity



Local connectivity

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

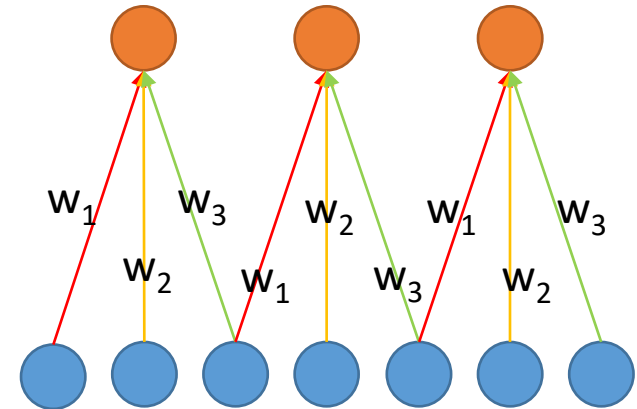
Convolutional Neural Networks (CNN): Weight Sharing



Hidden layer

Input layer

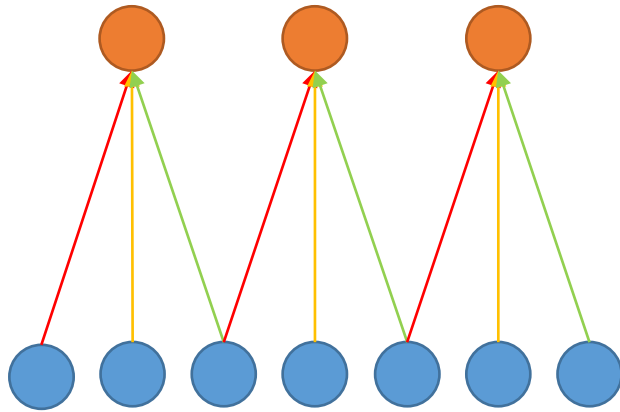
Without weight sharing



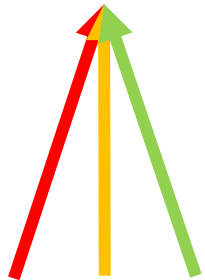
With weight sharing

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

CNN with Multiple Input Channels



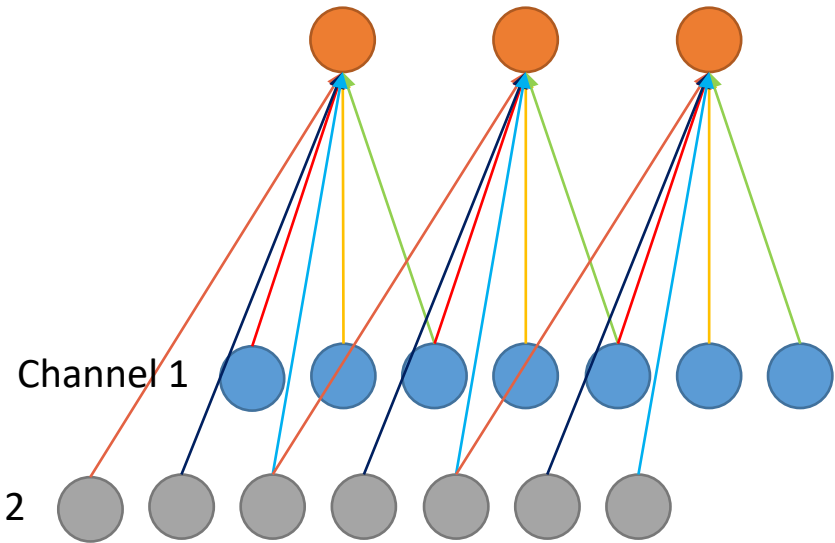
Single input channel



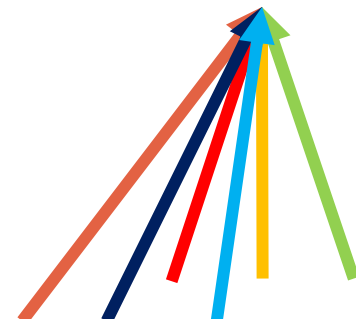
Filter weights

Hidden layer

Input layer

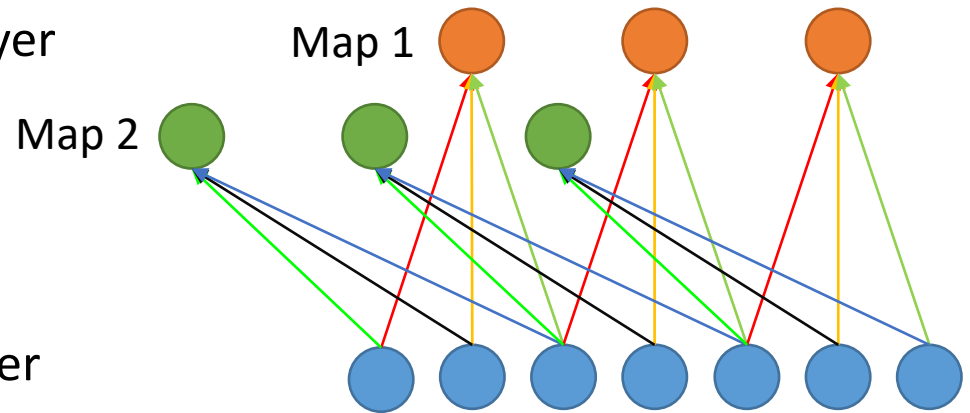
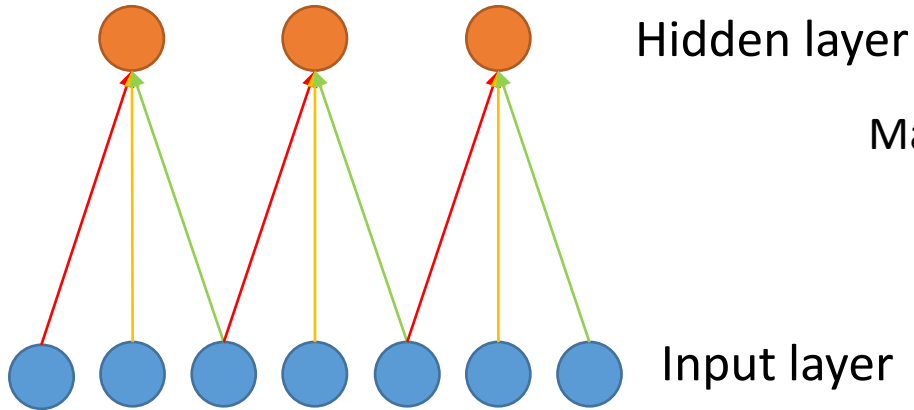


Multiple input channels

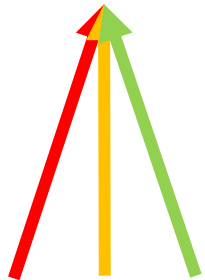


Filter weights

CNN with Multiple Output Maps

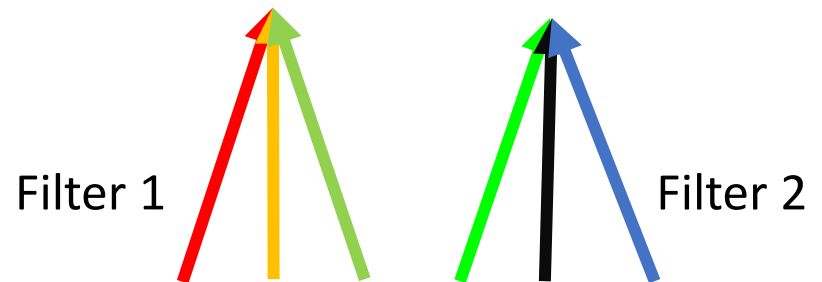


Single output map



Filter weights

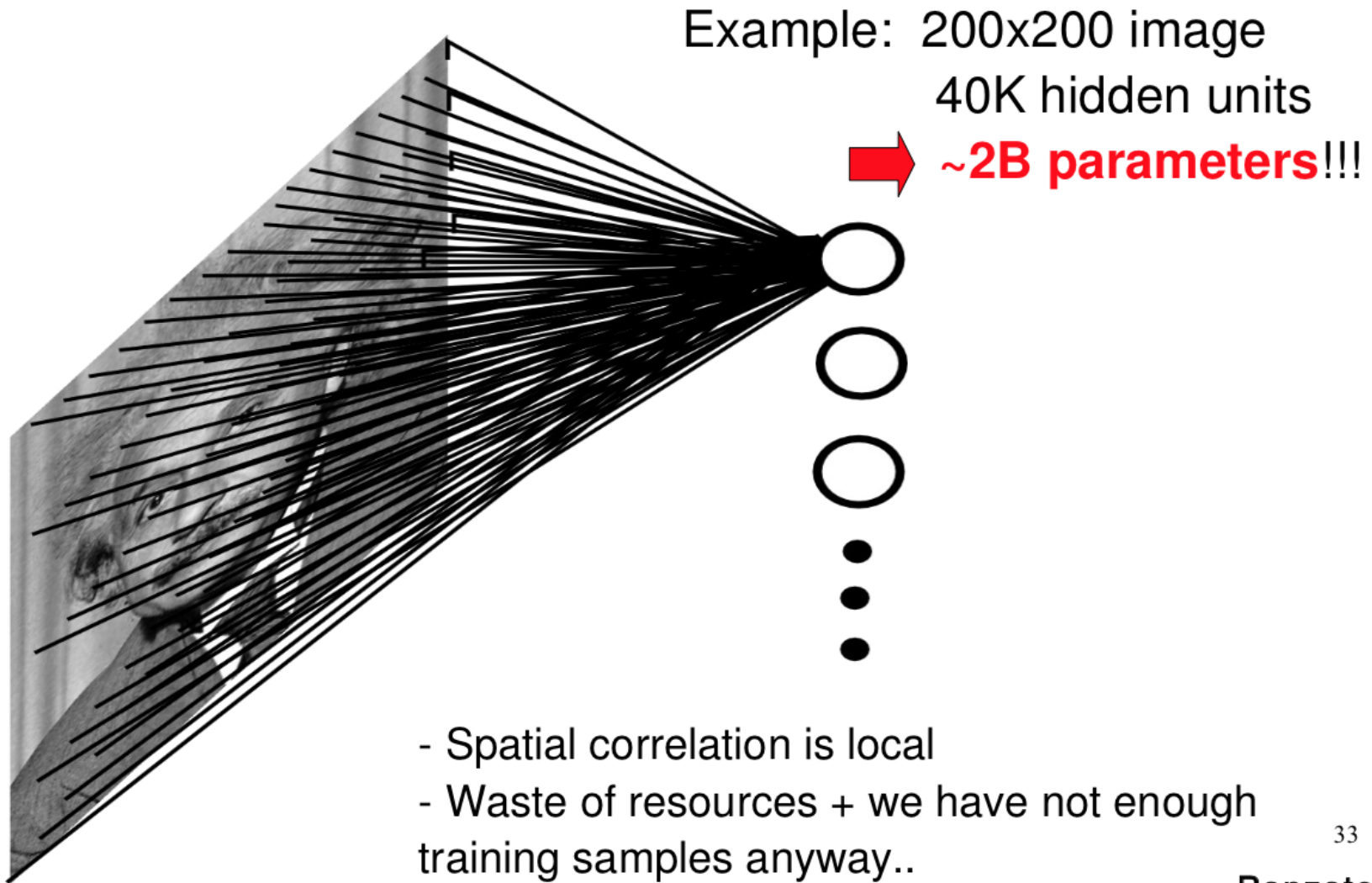
Multiple output maps



Filter weights

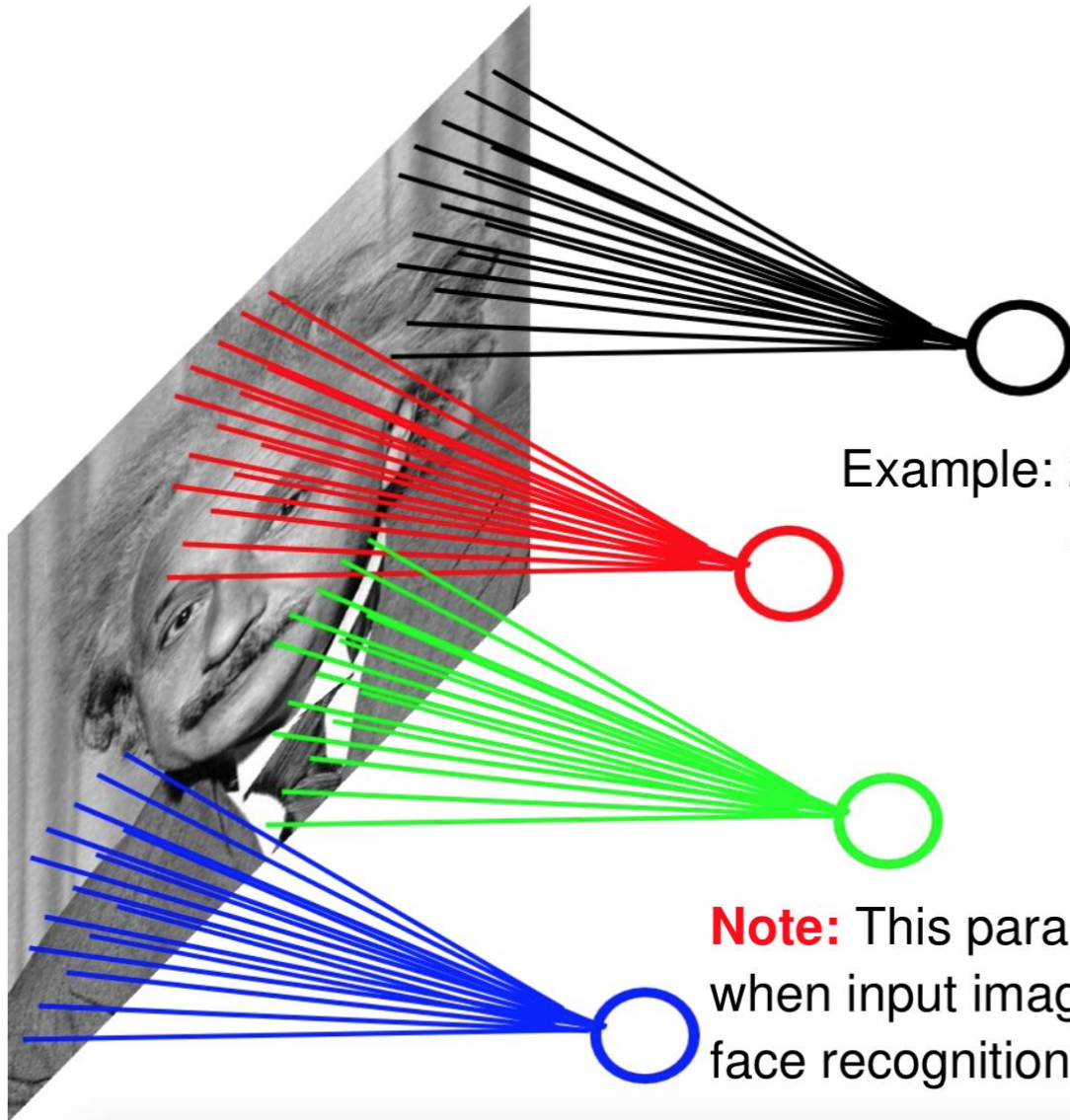
Generalized to 2D Cases:

Fully Connected Layer



Generalized to 2D Cases:

Locally Connected Layer



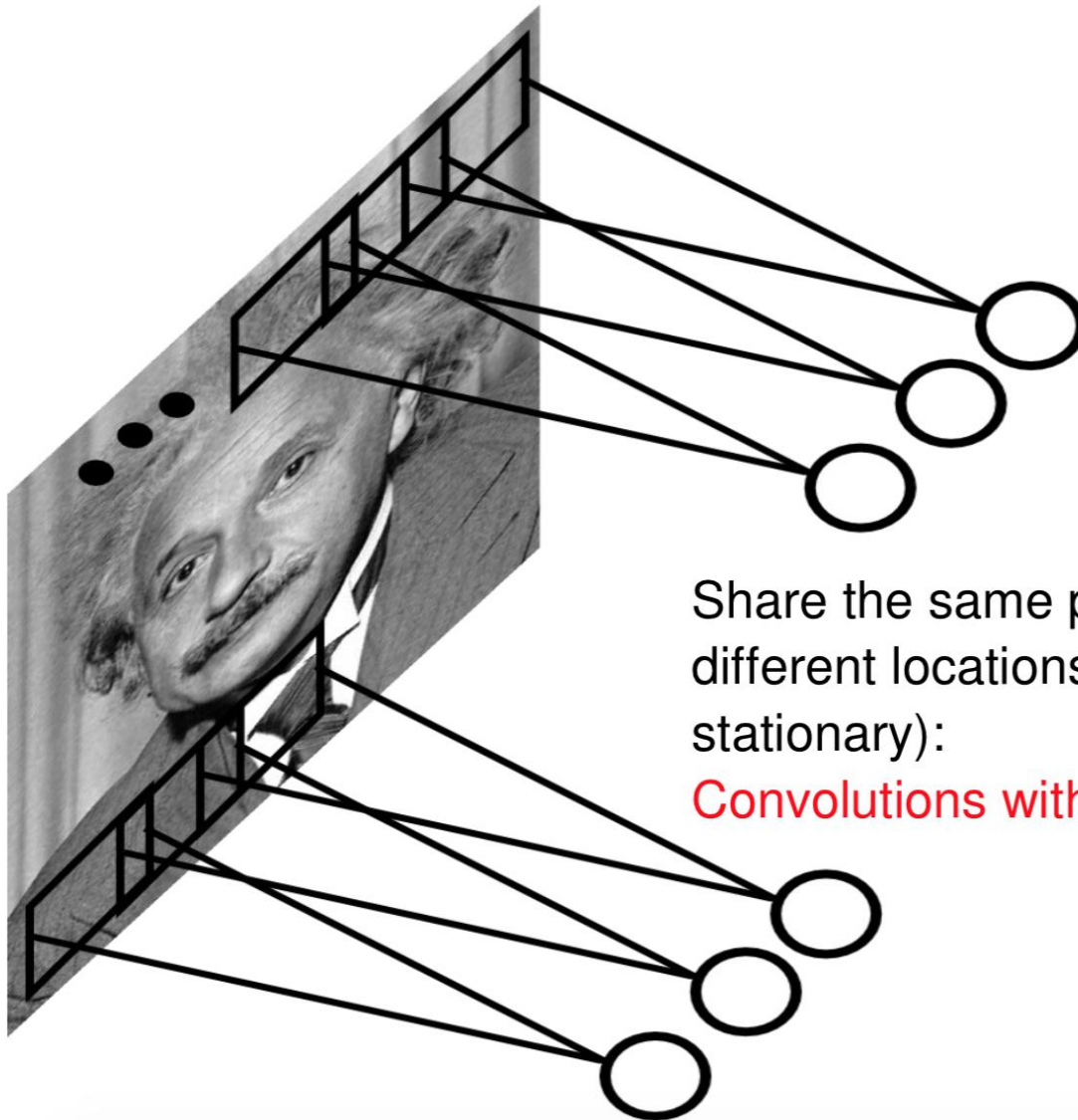
Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

Note: This parameterization is good when input image is registered (e.g., face recognition).

34

Generalized to 2D Cases:

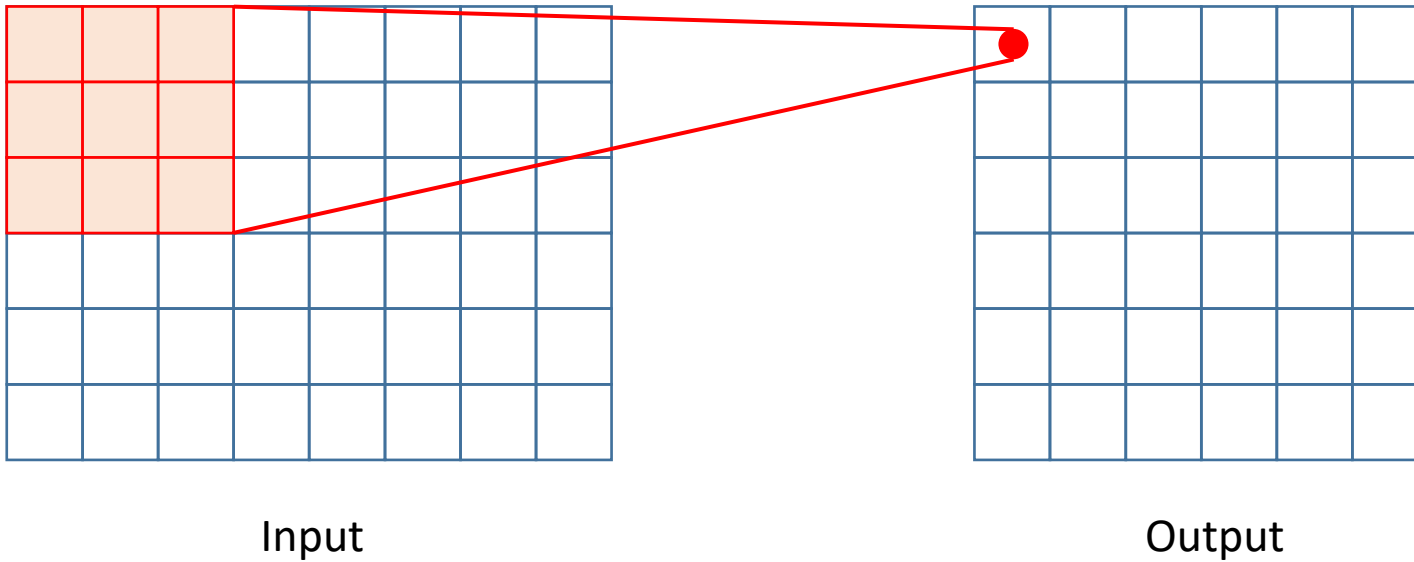
Convolutional Layer



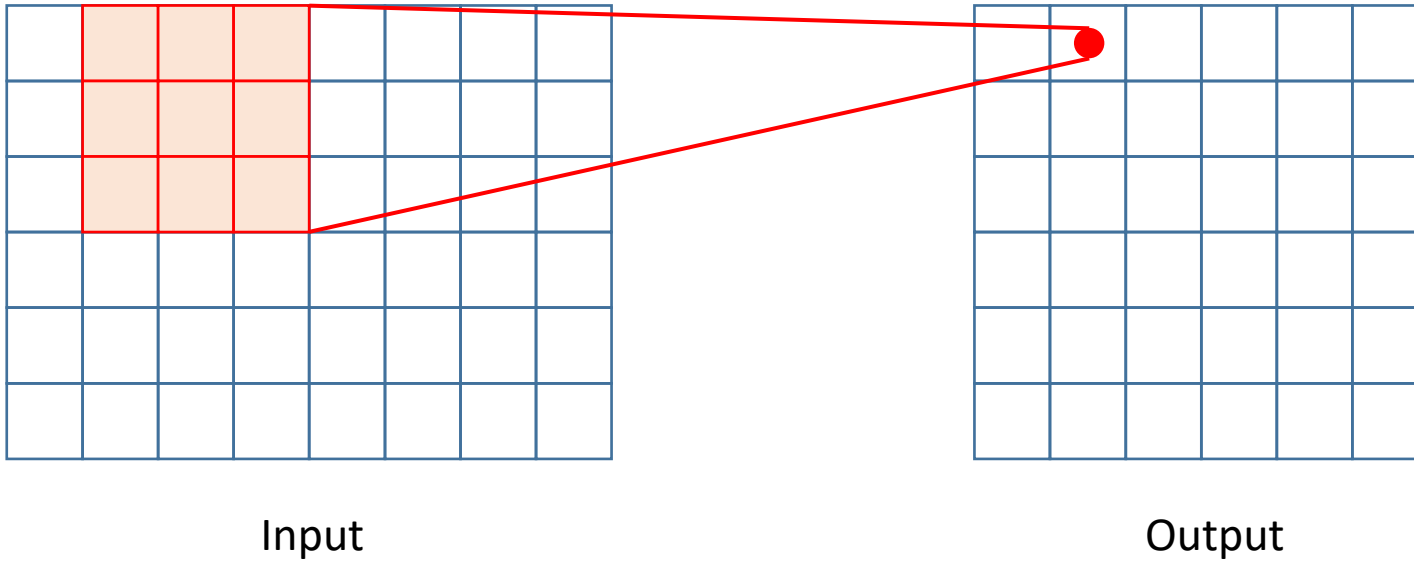
Share the same parameters across different locations (assuming input is stationary):

Convolutions with learned kernels

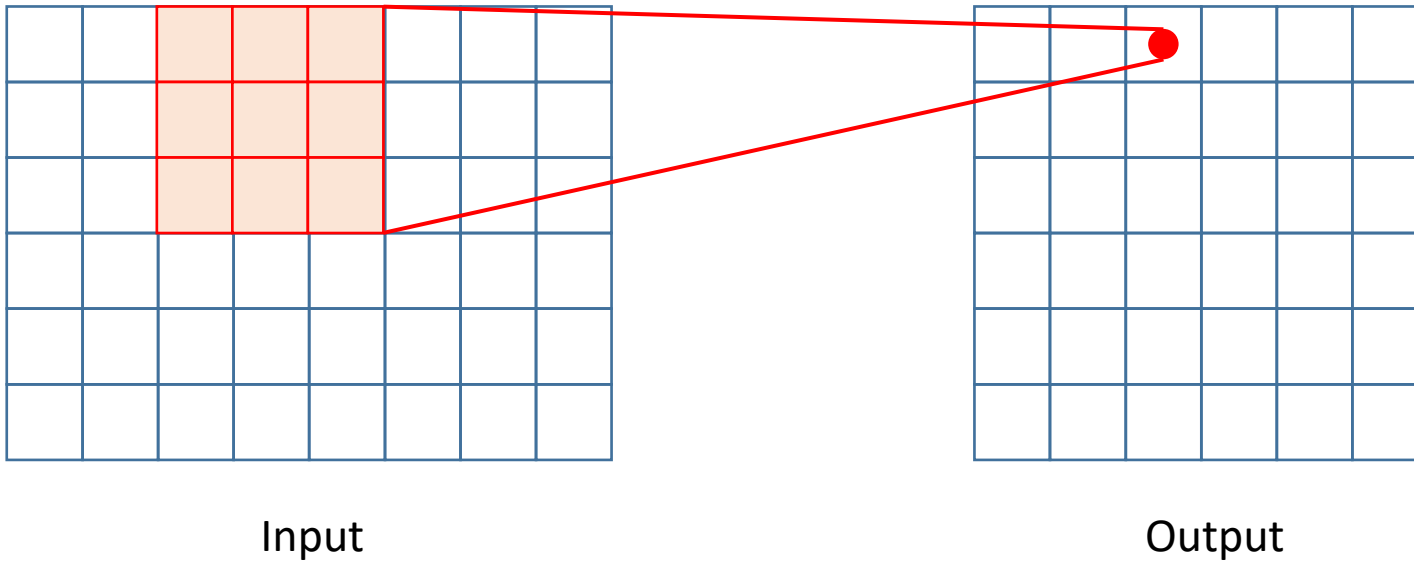
Convolutional Layer



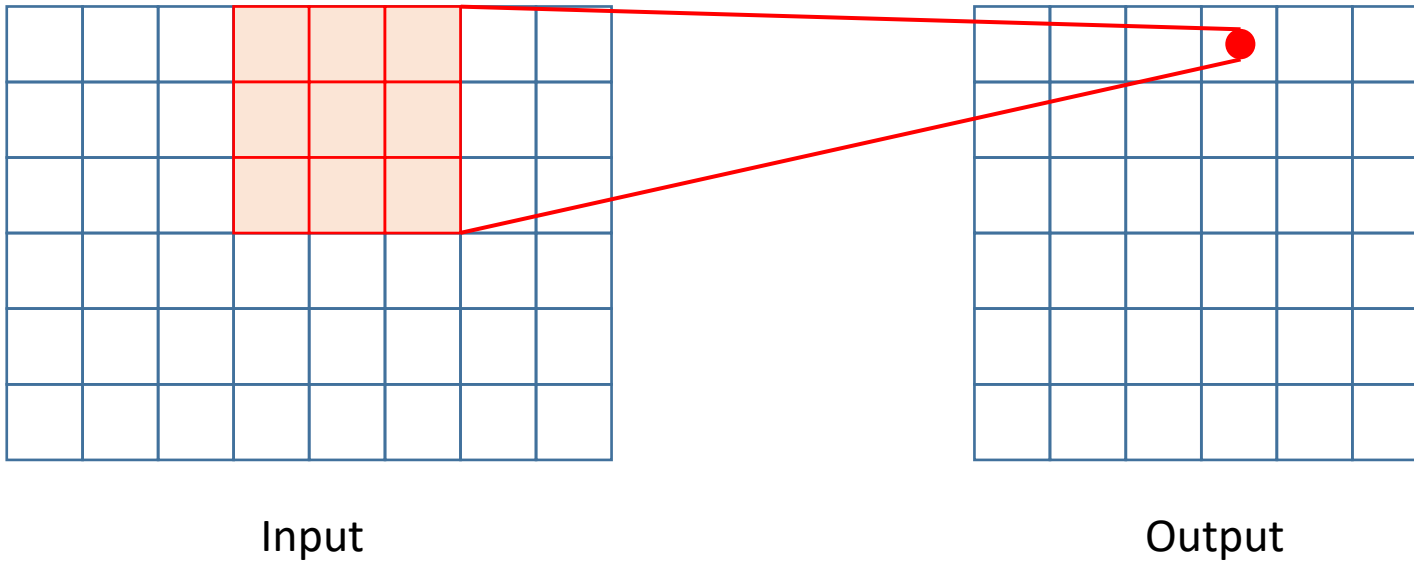
Convolutional Layer



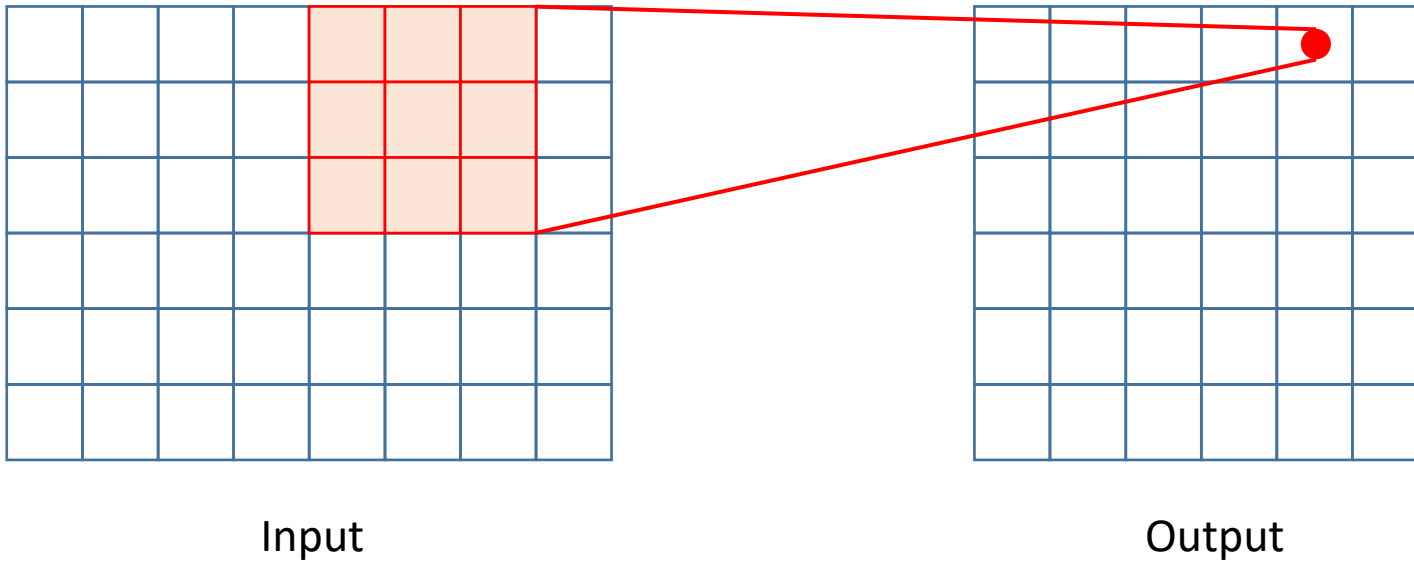
Convolutional Layer



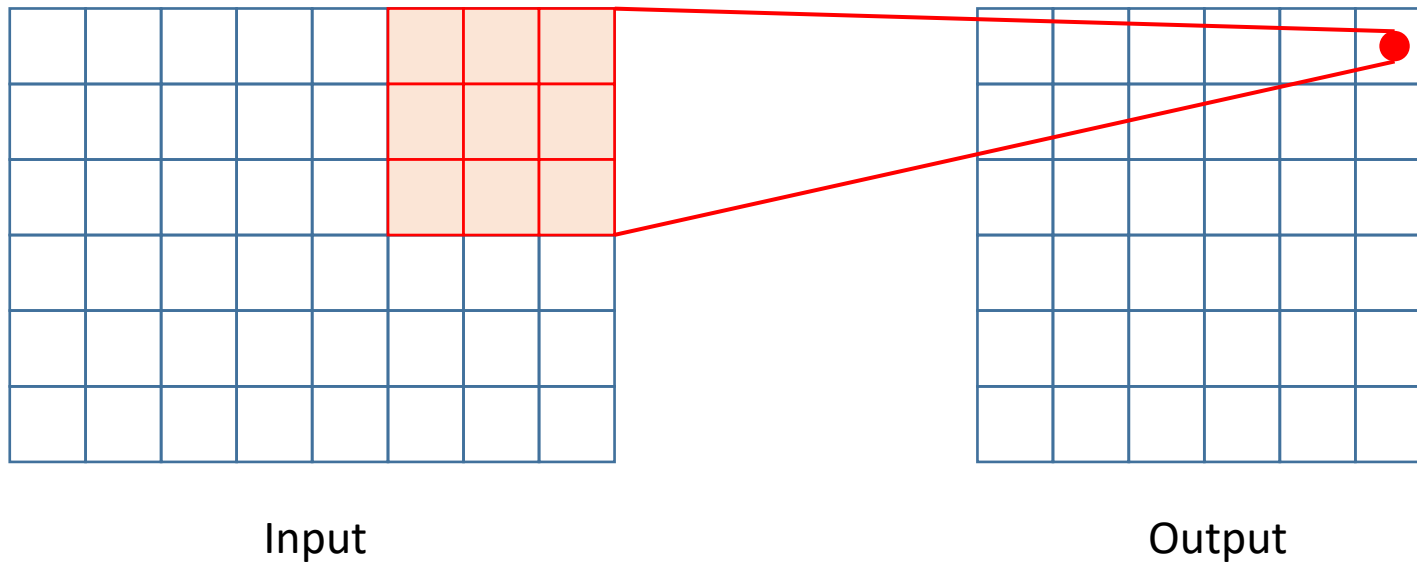
Convolutional Layer



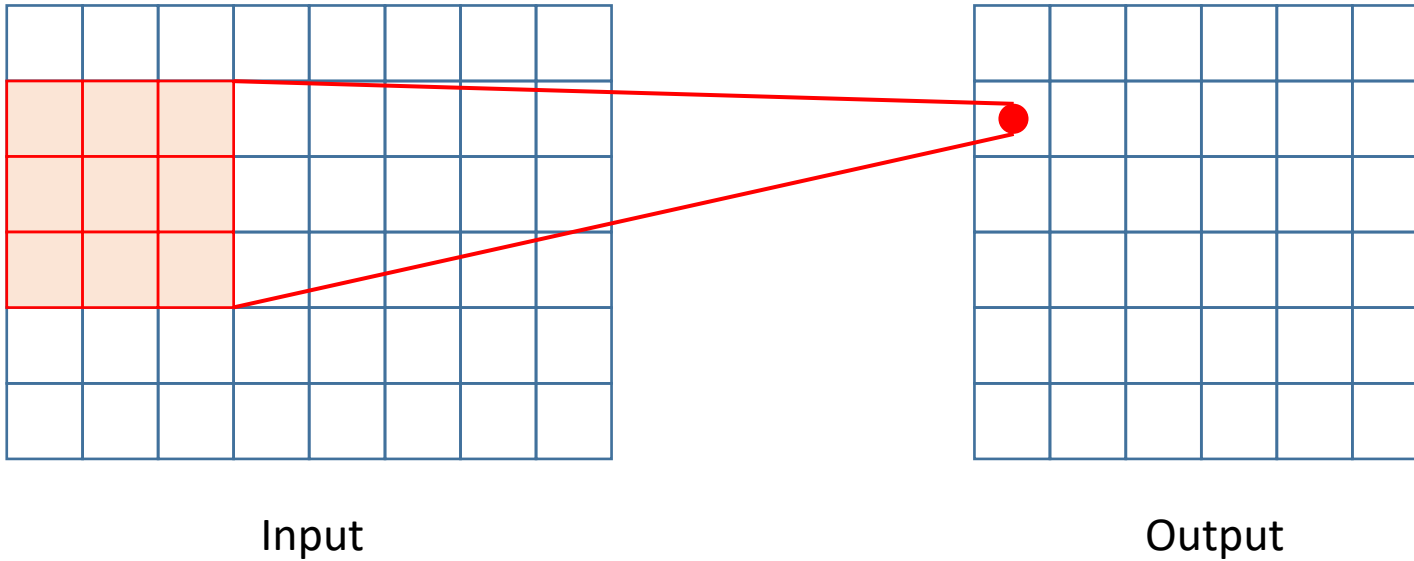
Convolutional Layer



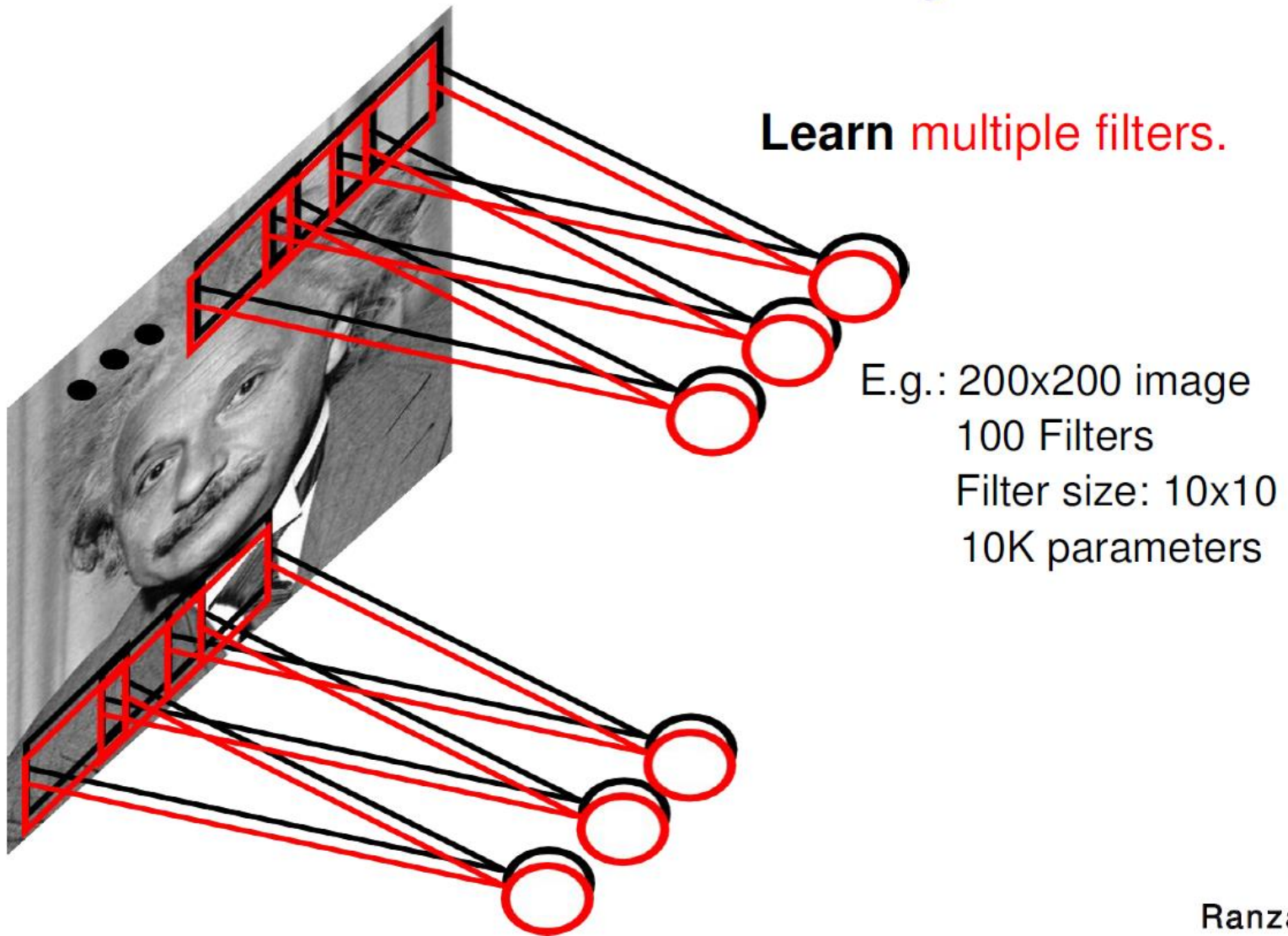
Convolutional Layer



Convolutional Layer

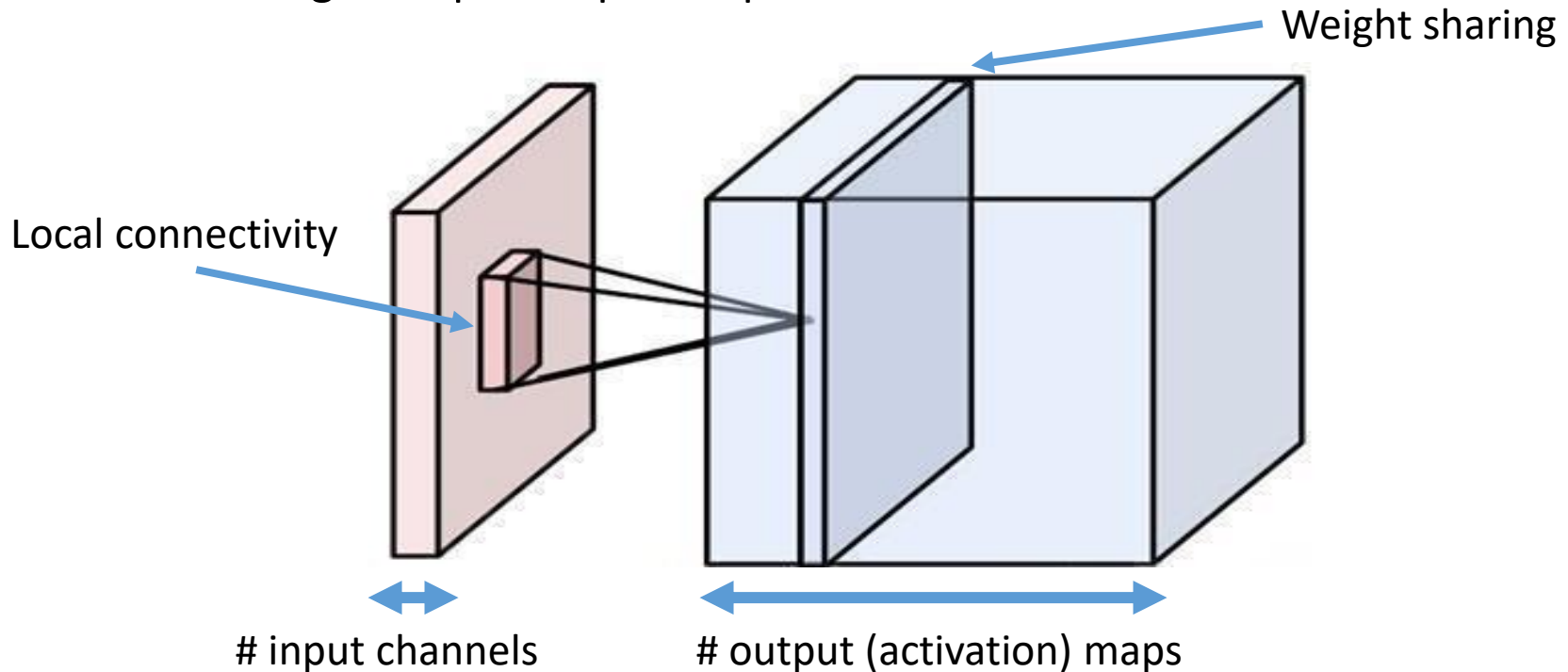


Convolutional Layer

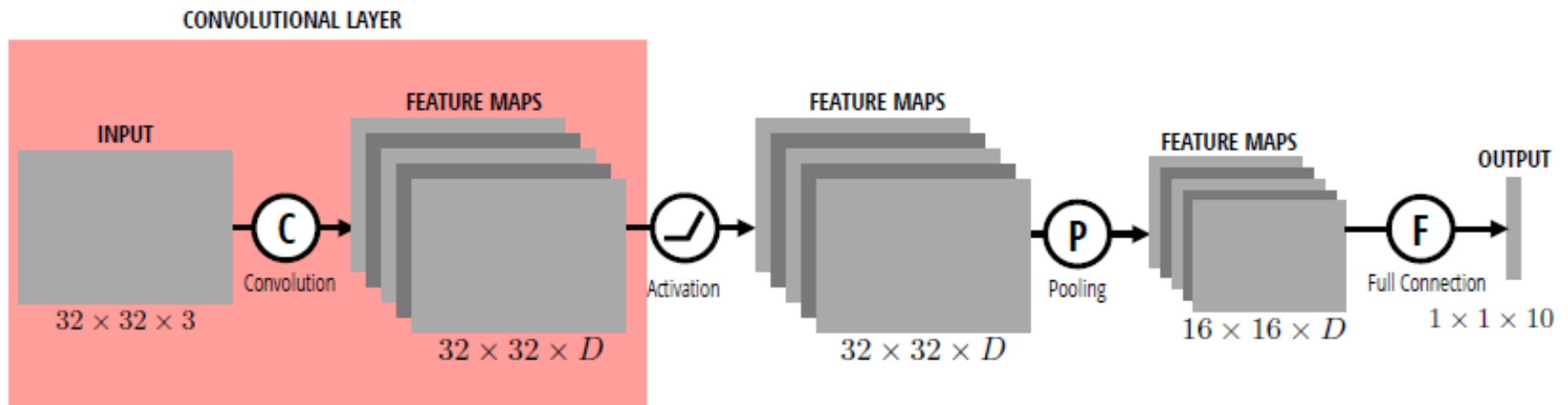


Putting them together → CNN

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

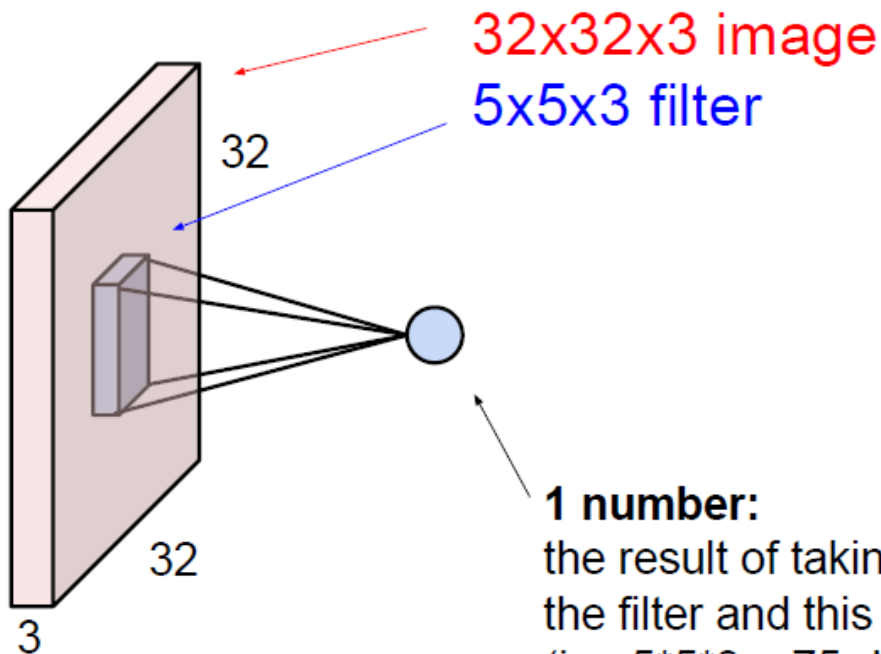


Convolution Layer in CNN



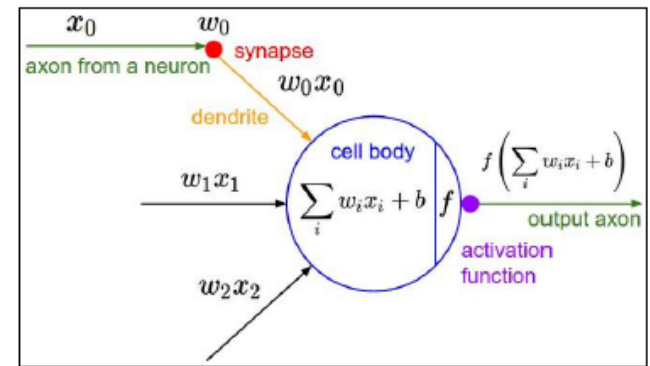
Putting them together (cont'd)

- The brain/neuron view of CONV layer



1 number:

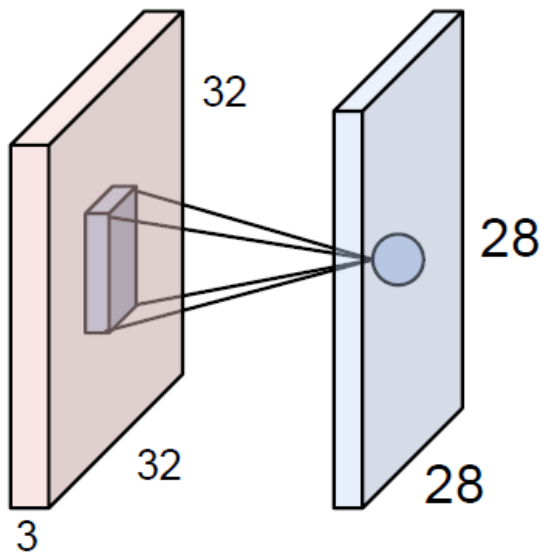
the result of taking a dot product between the filter and this part of the image (i.e. $5 \times 5 \times 3 = 75$ -dimensional dot product)



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

Putting them together (cont'd)

- The brain/neuron view of CONV layer



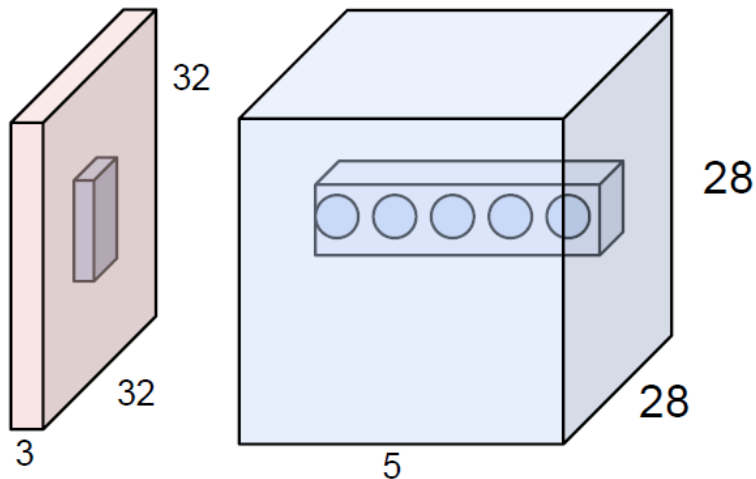
An activation map is a 28x28 sheet of neuron outputs:

1. Each is connected to a small region in the input
2. 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

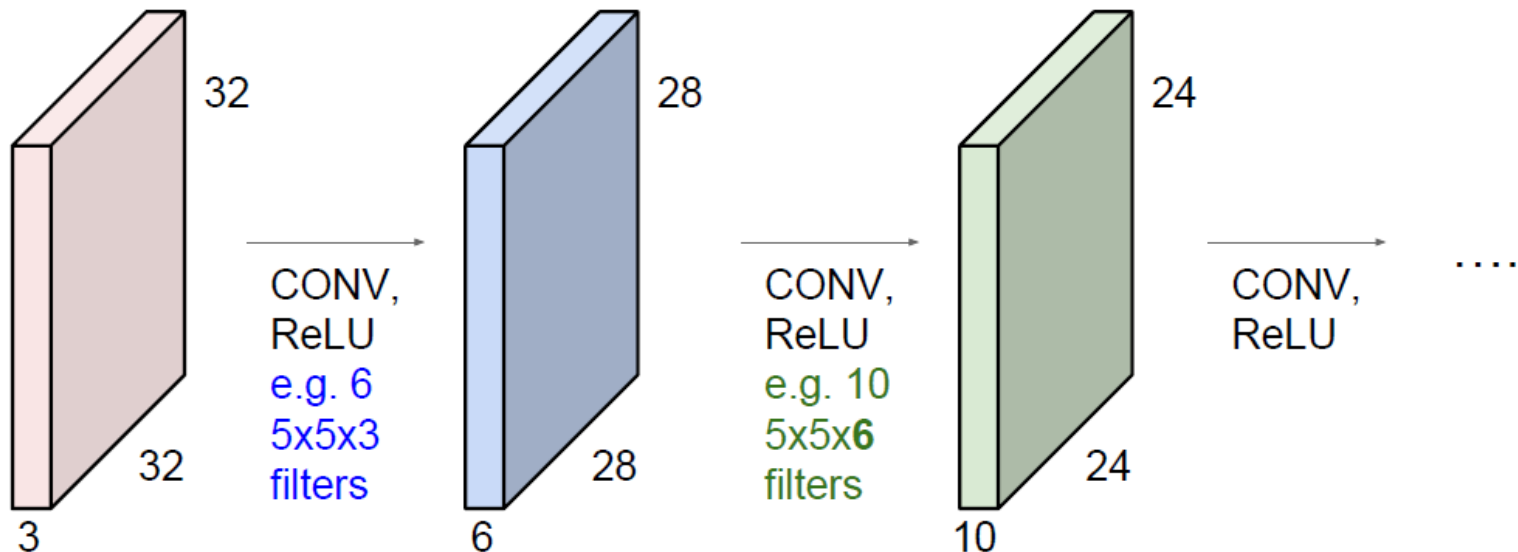


E.g. with 5 filters,
CONV layer consists of
neurons arranged in a 3D grid
(28x28x5)

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

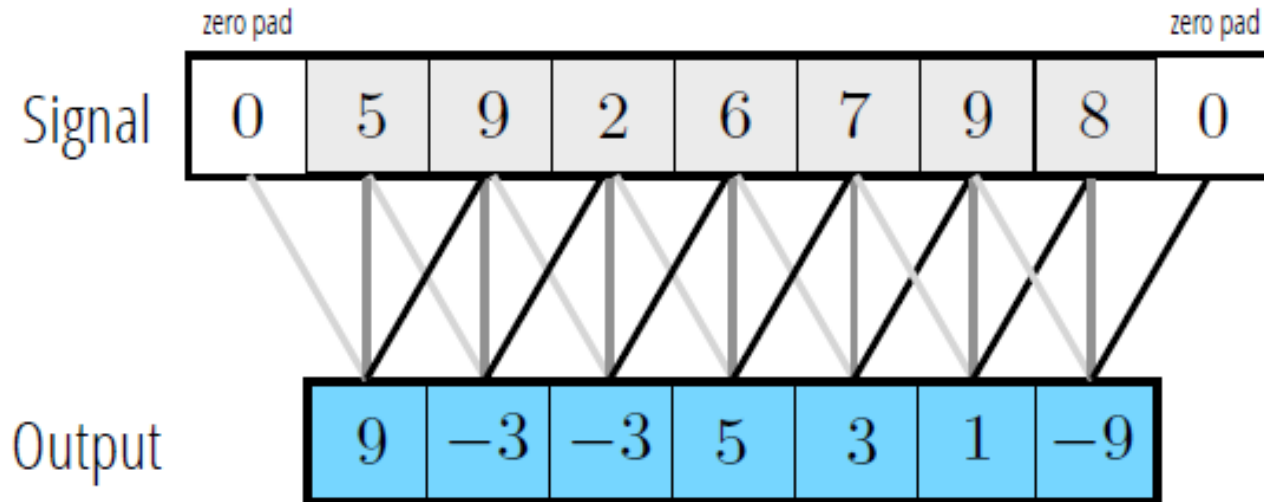
Putting them together (cont'd)

- Image input with 32 x 32 pixels convolved repeatedly with 5 x 5 x 3 filters shrinks volumes spatially (32 -> 28 -> 24 -> ...).



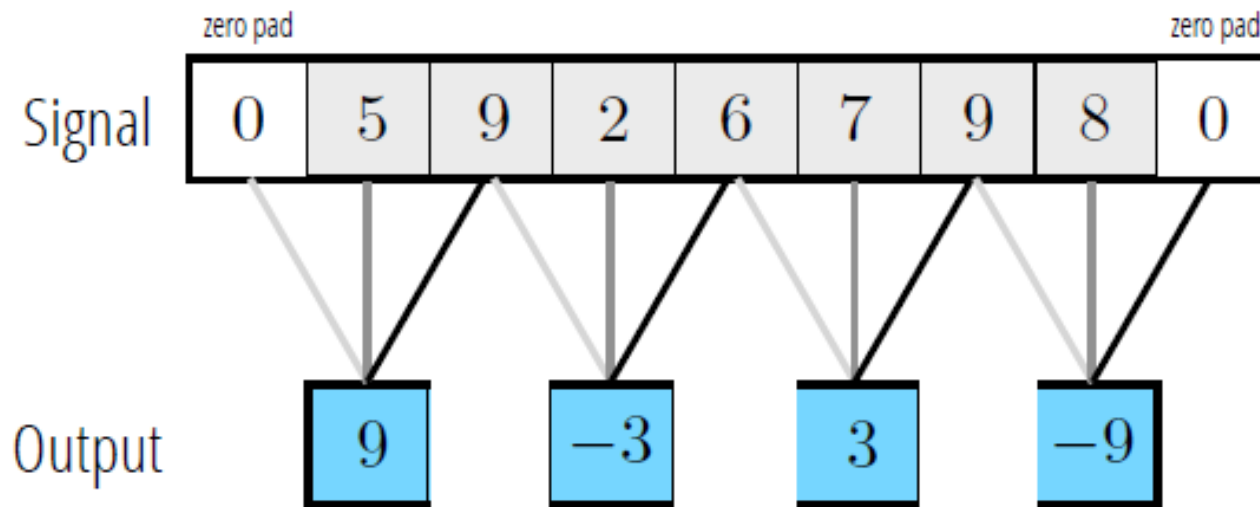
Variations of Convolution

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



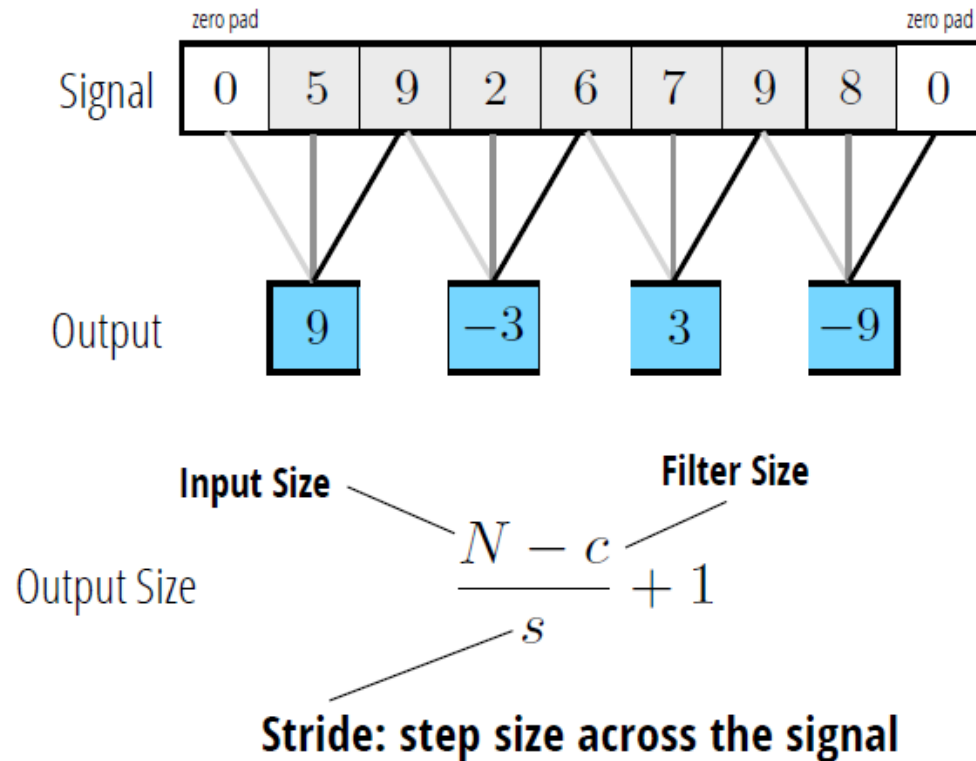
Variations of Convolution

- Stride
 - Step size across signals

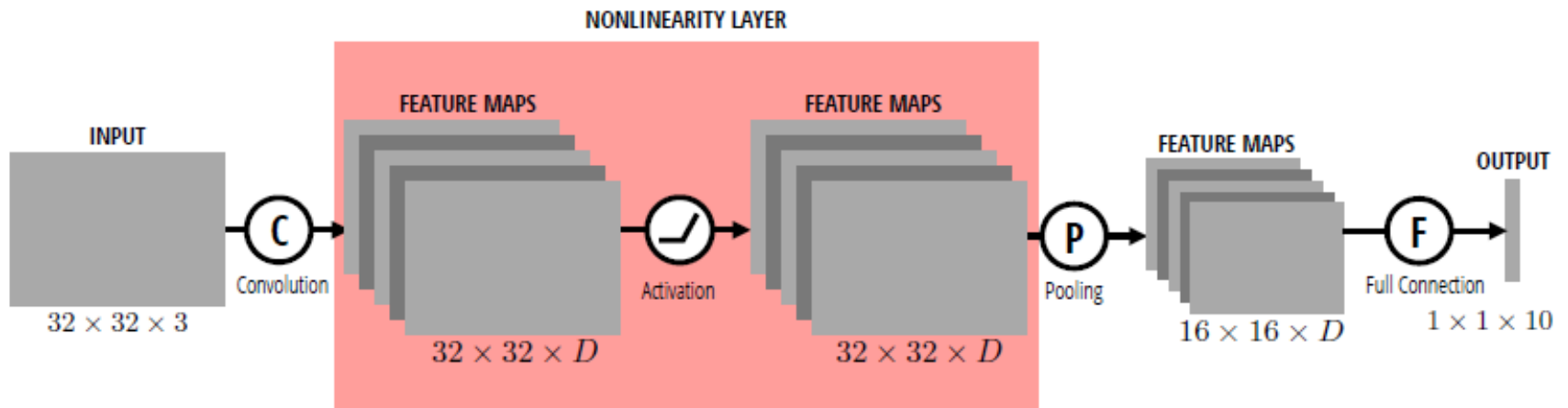


Variations of Convolution

- Stride
 - Step size across signals

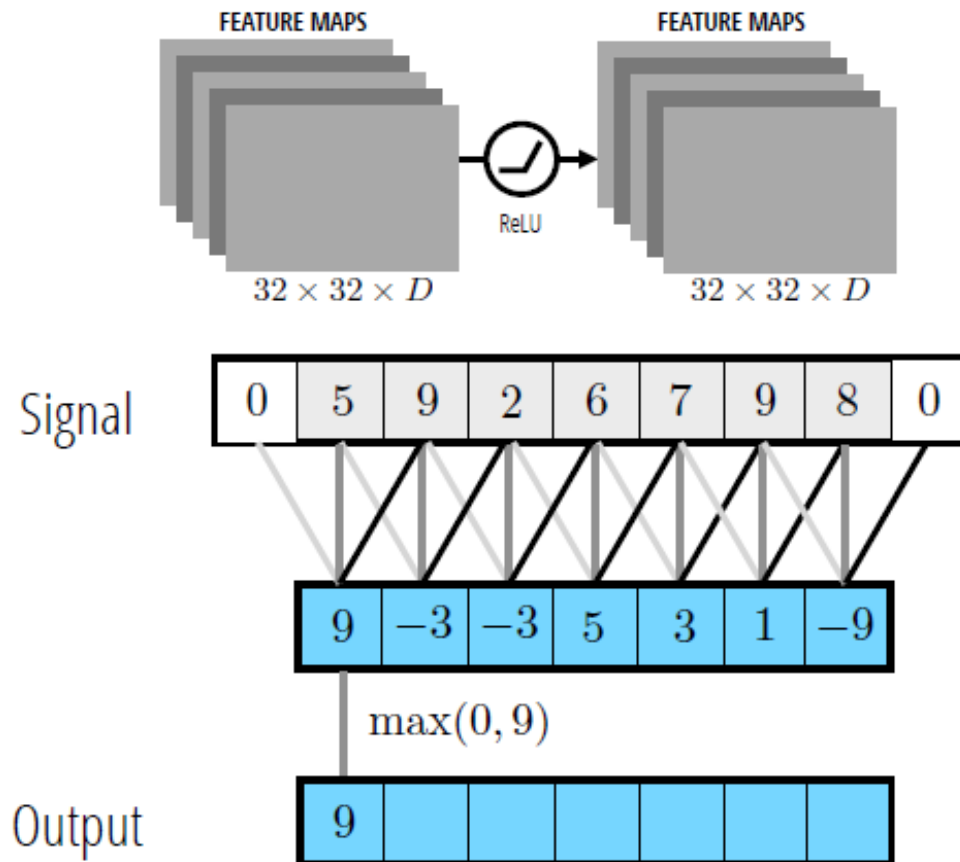


Nonlinearity Layer in CNN



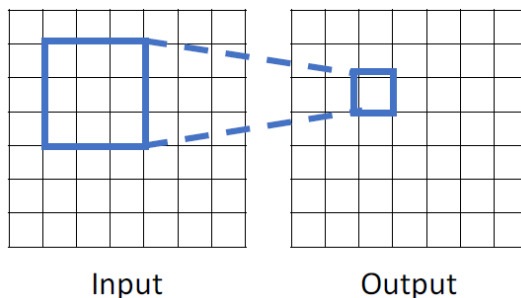
Nonlinearity Layer

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

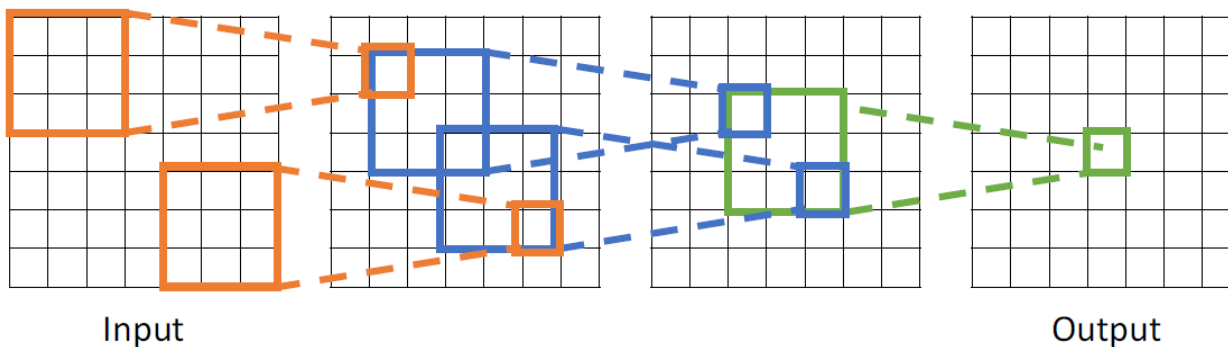


Receptive Field

- 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.

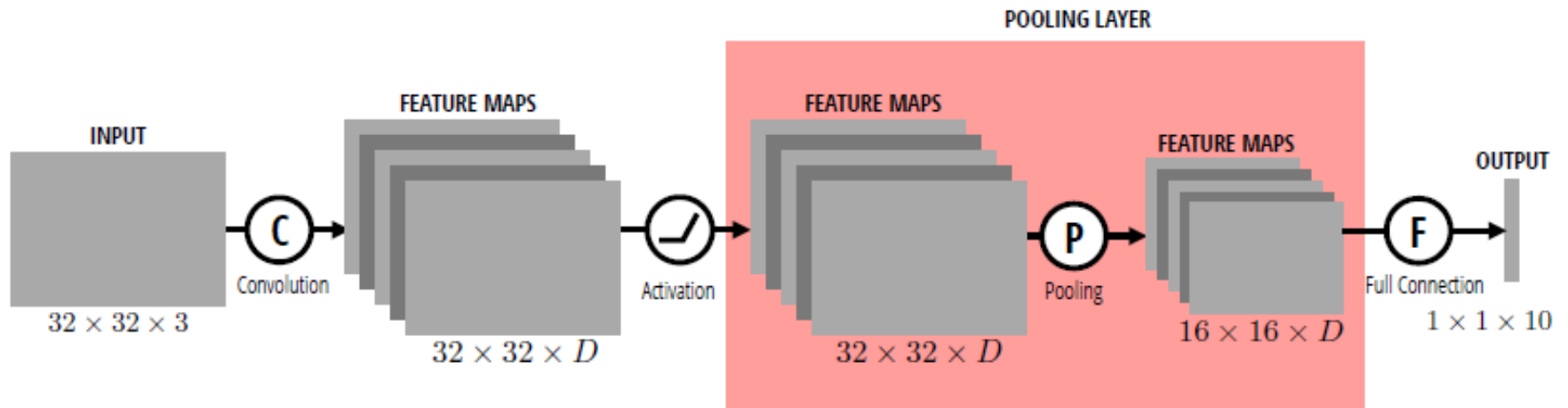


- Each successive convolution adds $n-1$ to the receptive field size. With a total of L layers, the receptive field size would be $1 + L * (n-1)$.



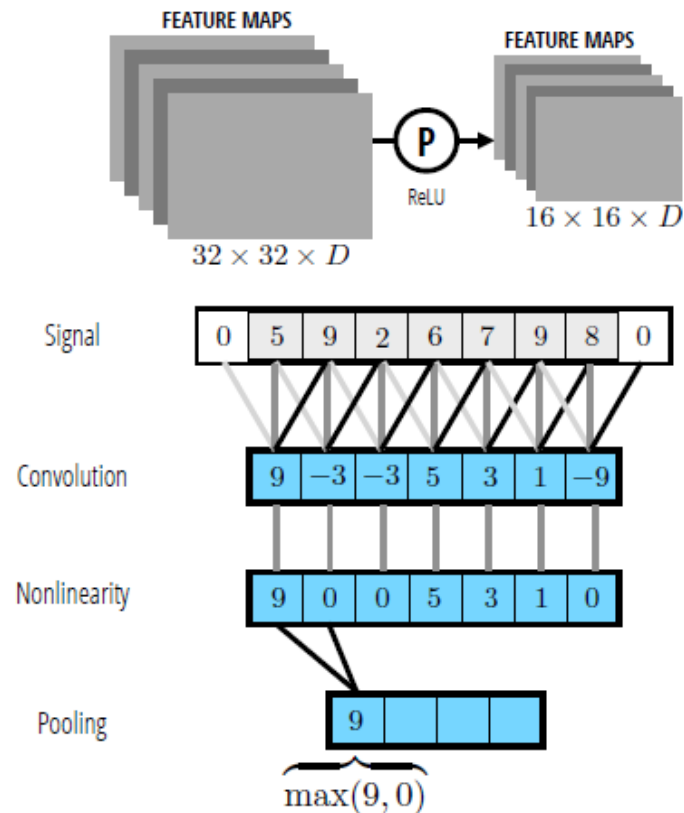
- Thus, for large images, we need many layers for each entry in output to “see” the entire input image. Possible solution → **downsample** the image/feature map (see **pooling layer** next)

Pooling Layer in CNN



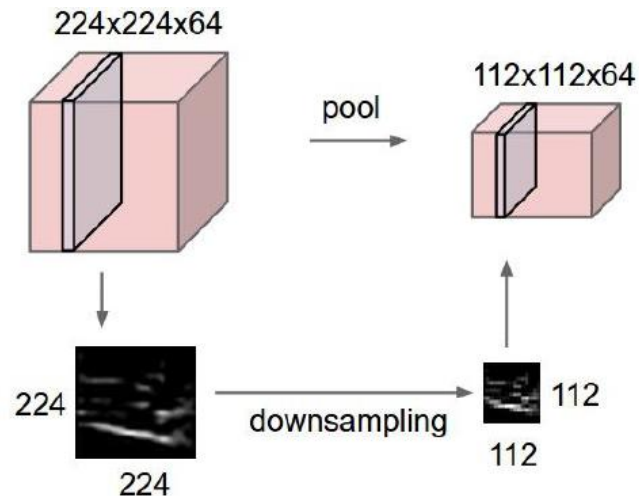
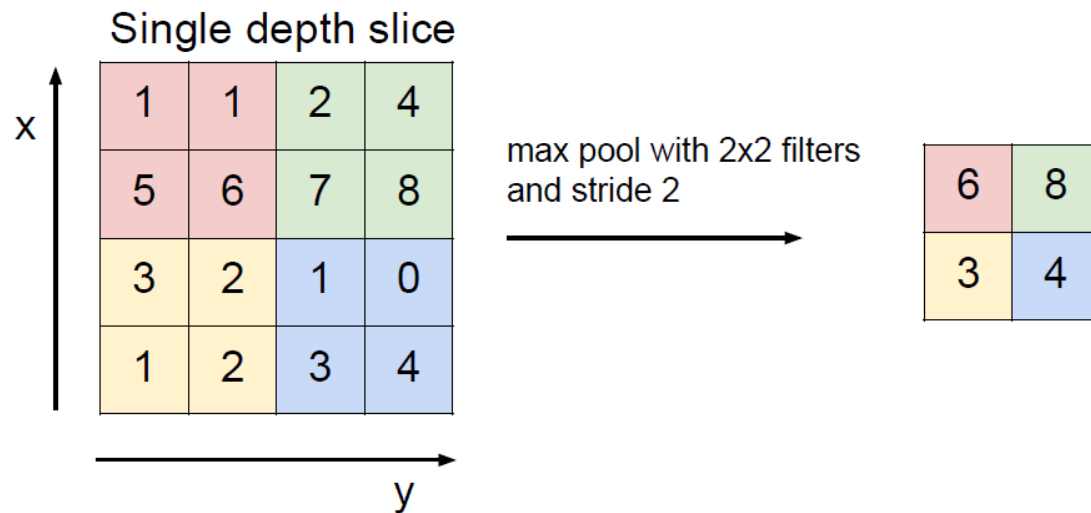
Pooling Layer

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

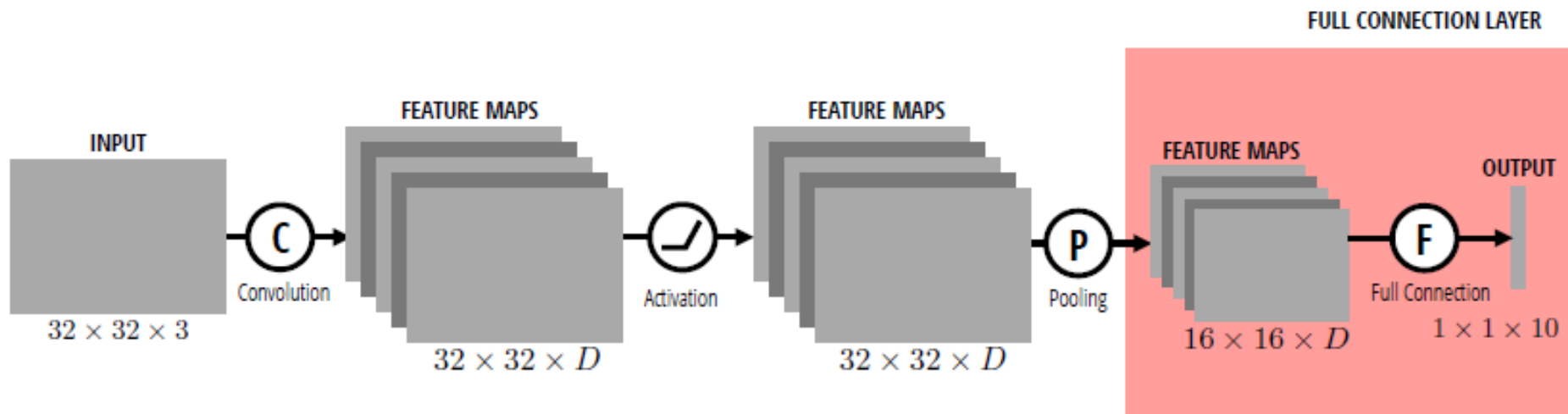


Pooling Layer for 2D Cases

- Reduces the spatial size and provides spatial invariance

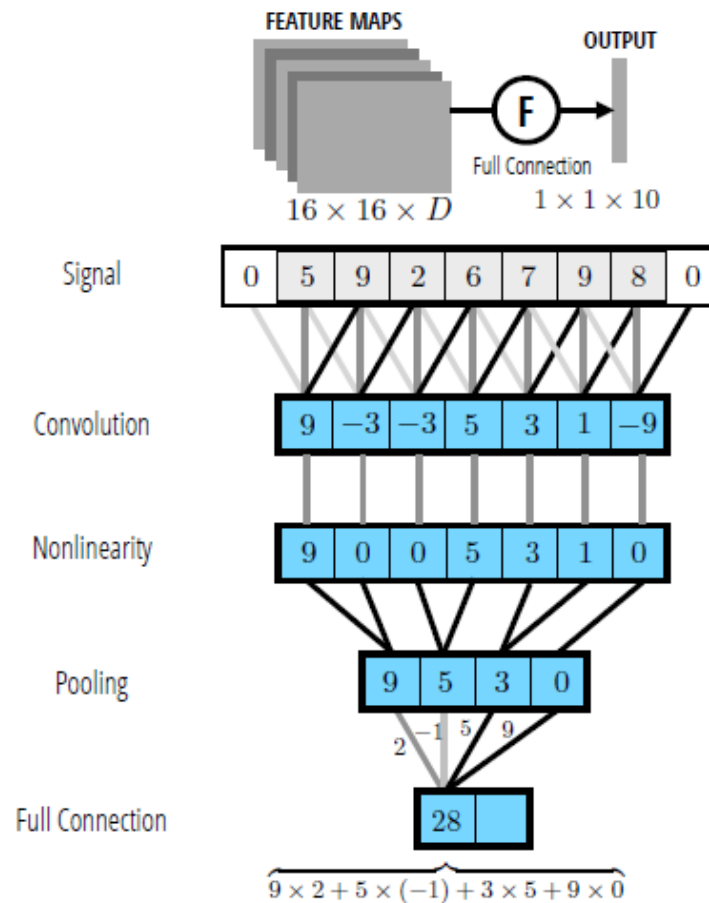


Fully Connected (FC) Layer in CNN



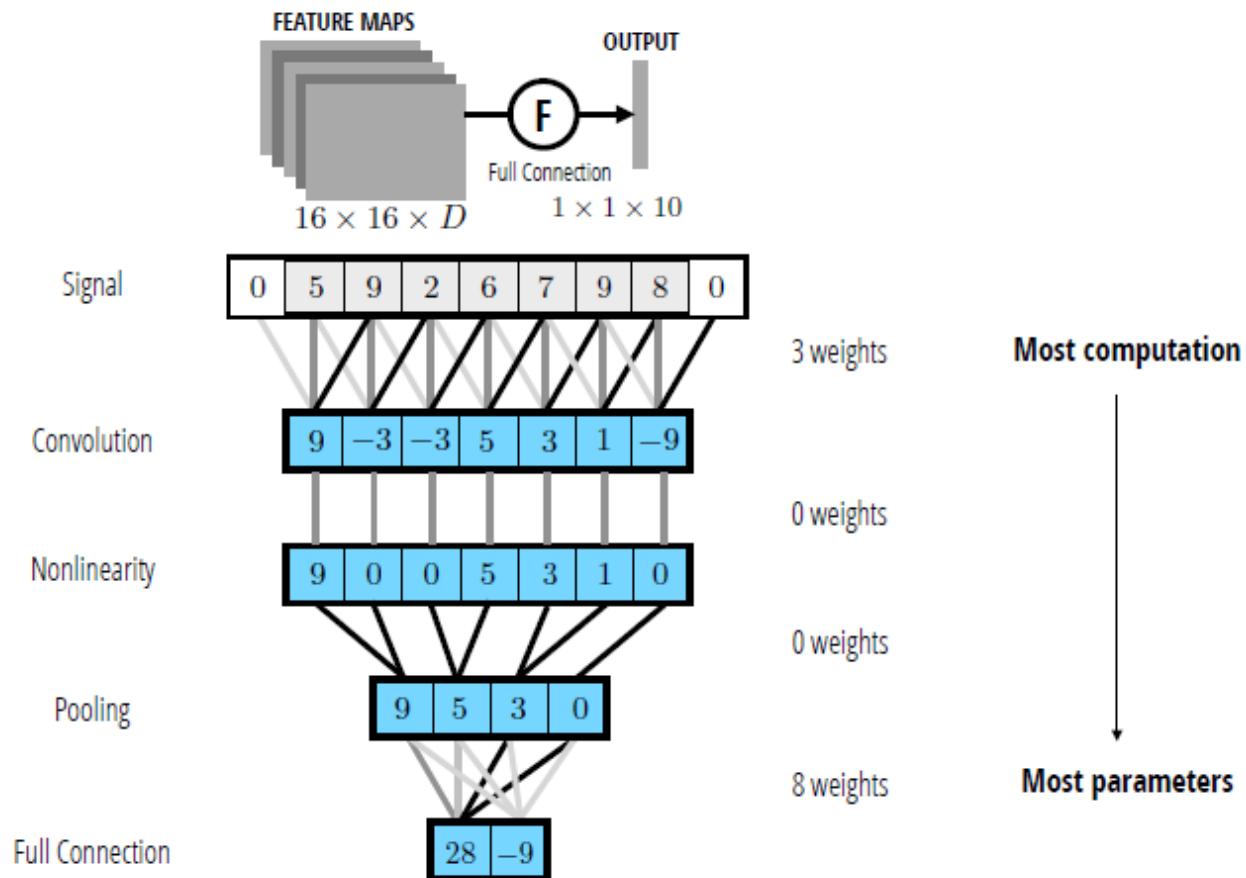
FC Layer

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

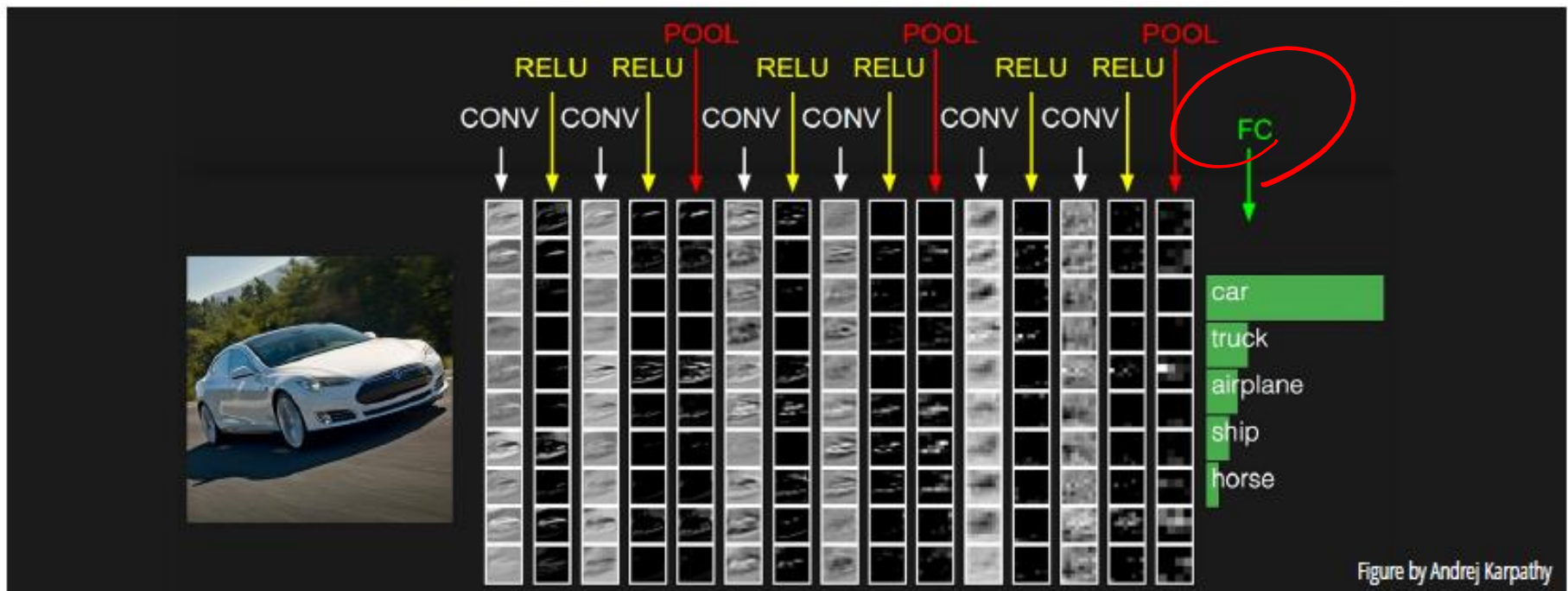
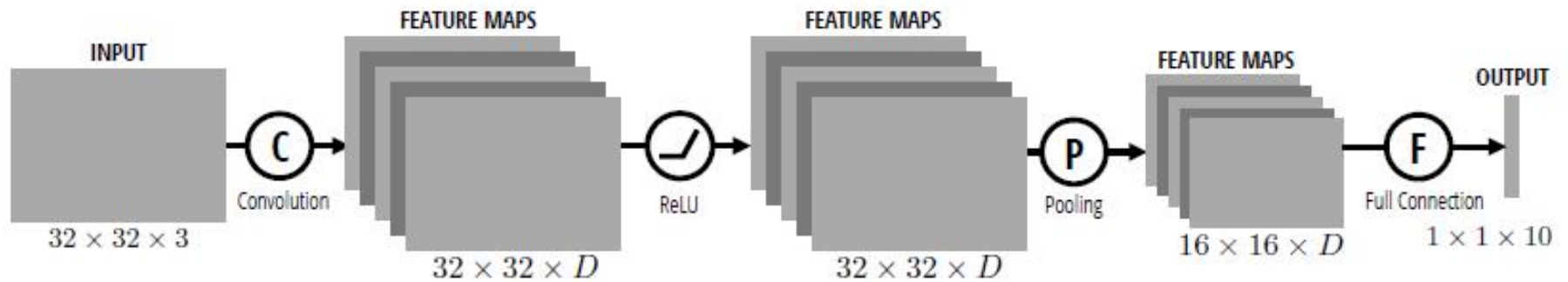


FC Layer

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

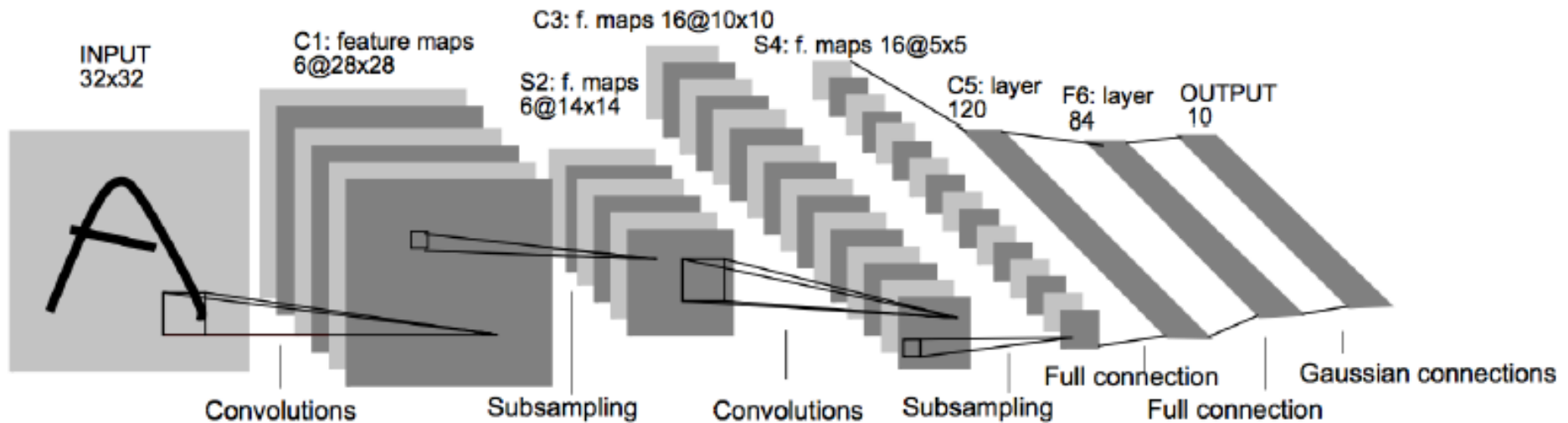


CNN

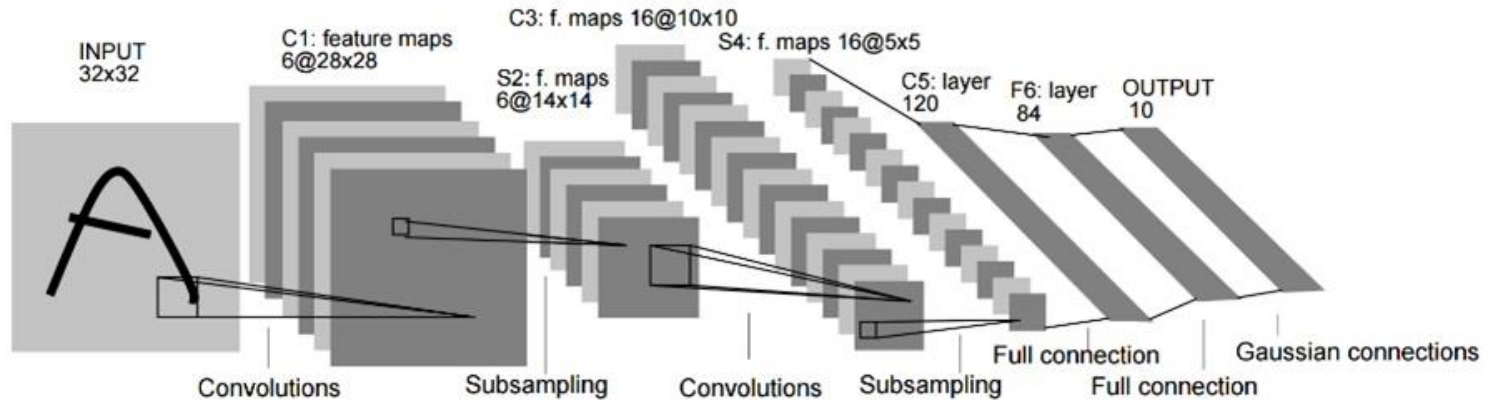


LeNet

- 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 2nd 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

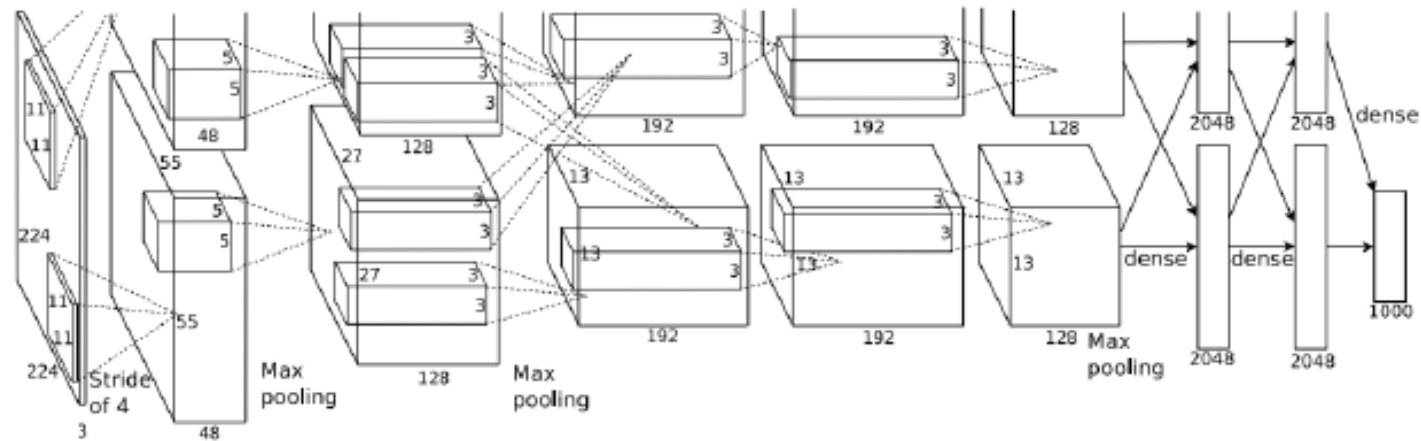
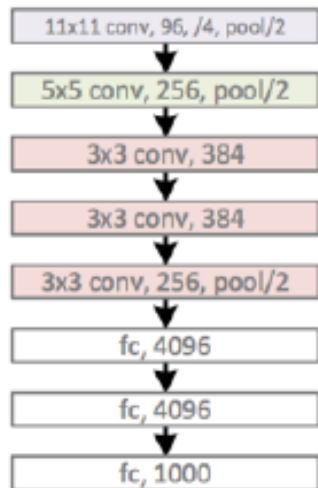
[13x13x256] **CONV5**: 256 3x3 filters at stride 1, pad 1

[6x6x256] **MAX POOL3**: 3x3 filters at stride 2

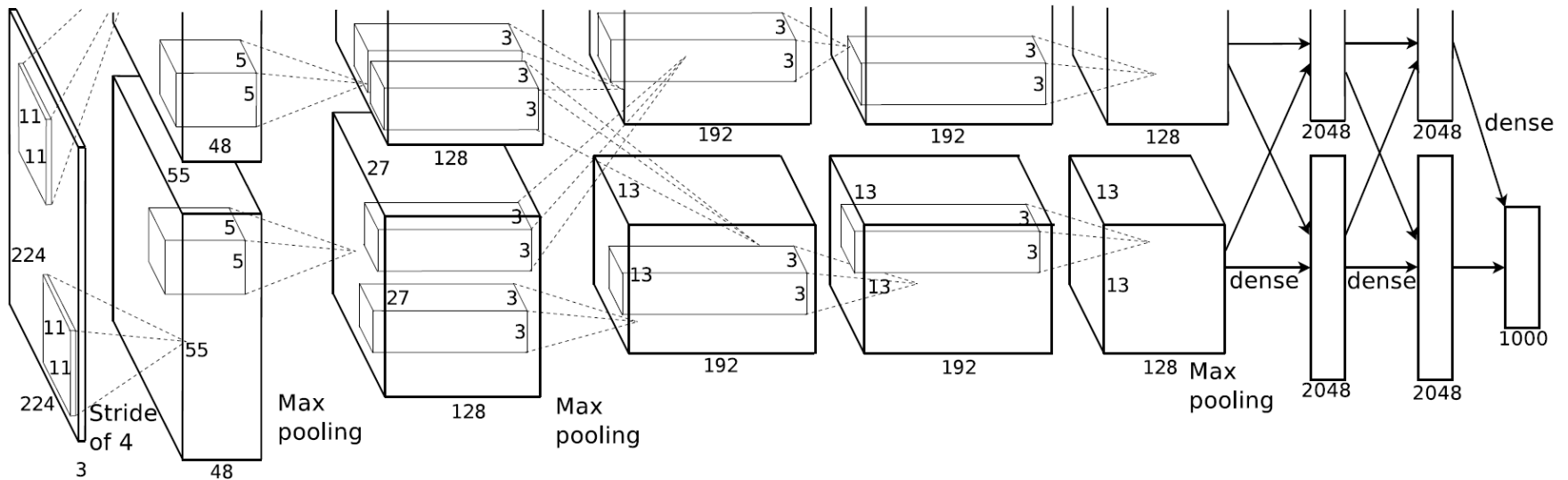
[4096] **FC6**: 4096 neurons

[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

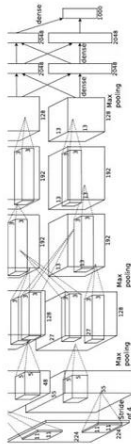
Removing feature & conv layers produces a **33% drop** in performance

Ultra Deep Network

http://cs231n.stanford.edu/slides/winter1516_lecture8.pdf

8 layers

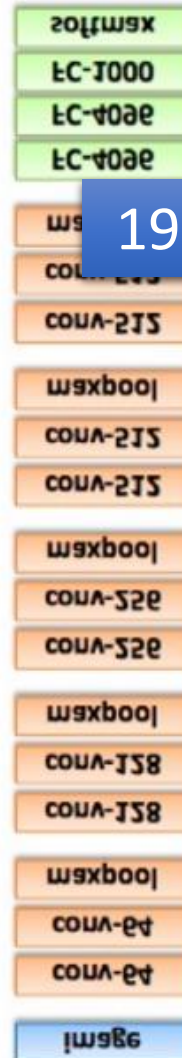
16.4%



AlexNet (2012)

19 layers

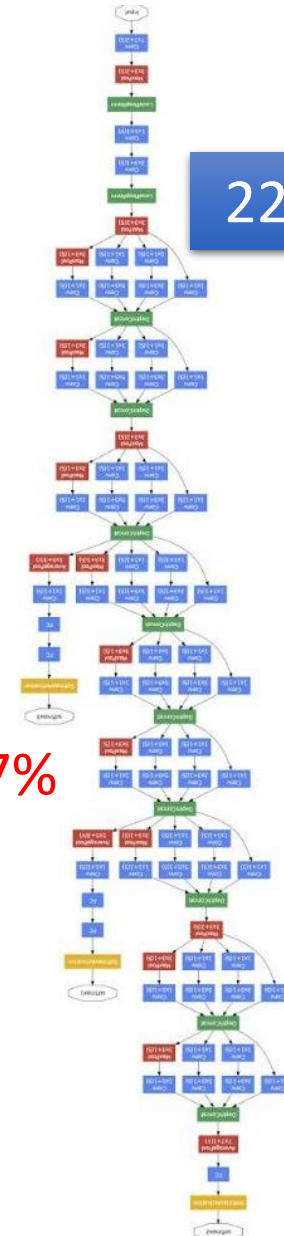
7.3%



VGG (2014)

22 layers

6.7%

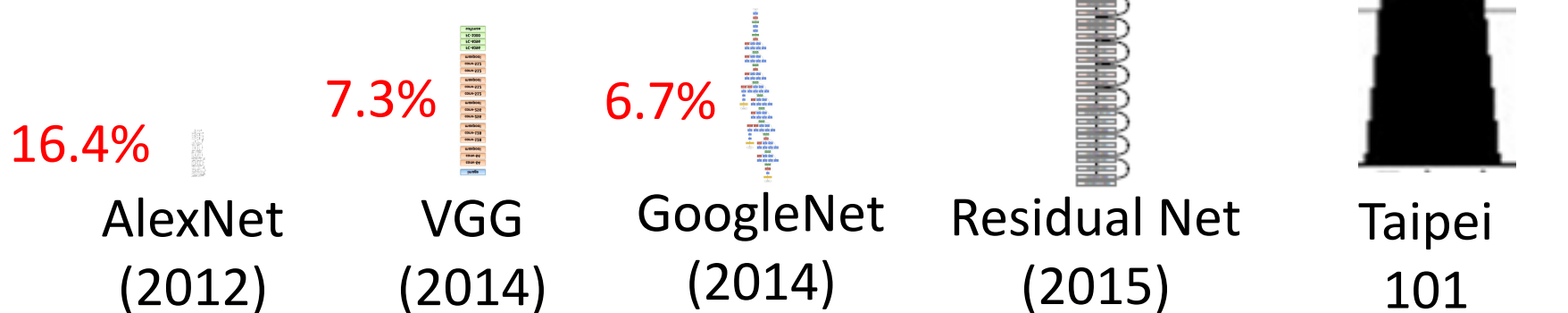


GoogleNet (2014)

Ultra Deep Network

[Prof. H.-Y. Lee]

This ultra deep network have special structure.



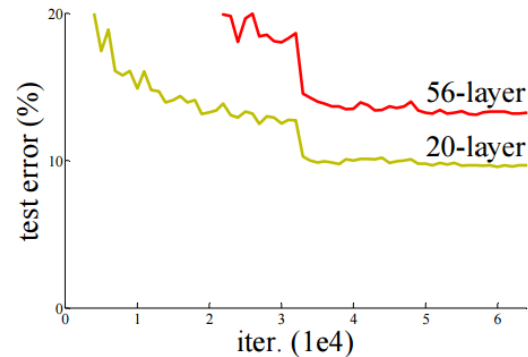
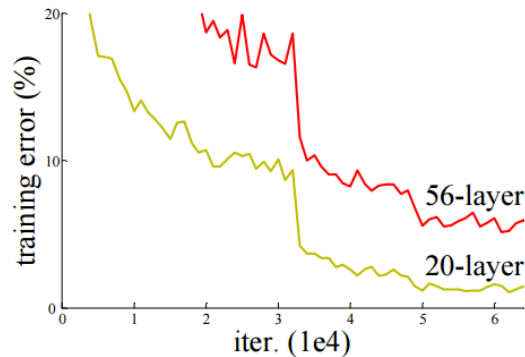
VGG (2014)

- 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)

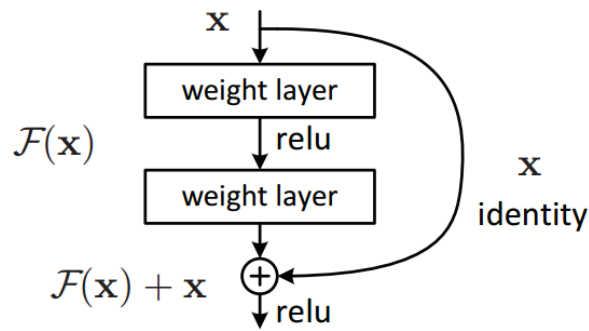


ResNet (2016)

- Can we just increase the #layer?



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

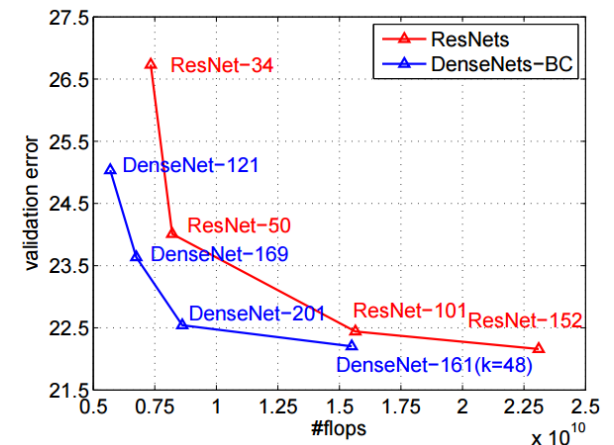
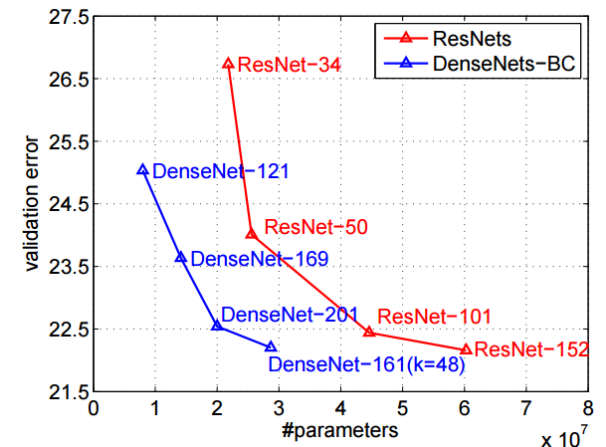
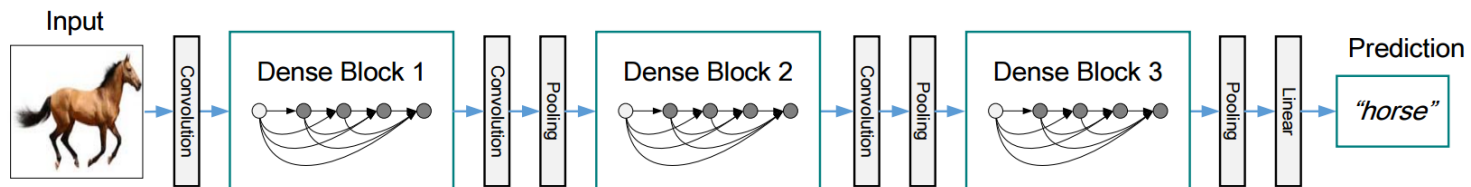
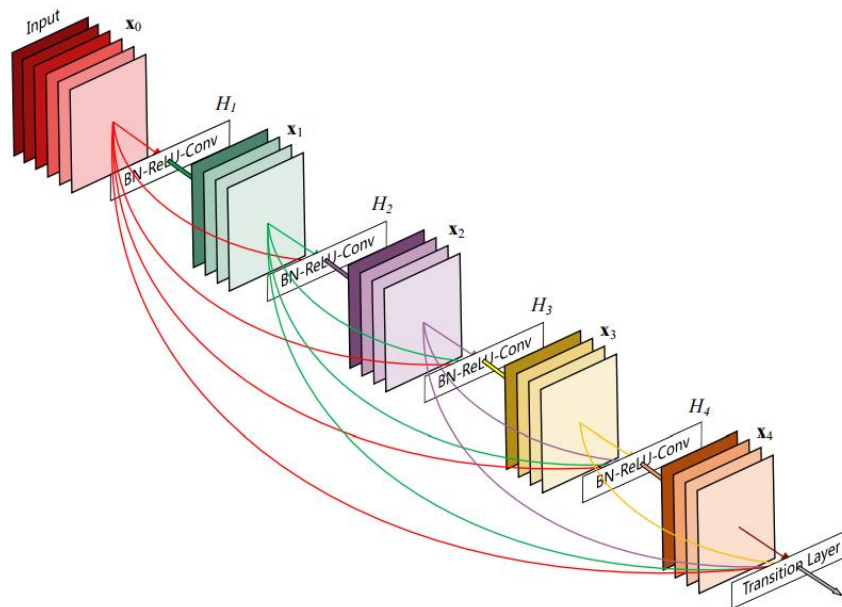


method	top-5 err. (test)
VGG [41] (ILSVRC'14)	7.32
GoogLeNet [44] (ILSVRC'14)	6.66
VGG [41] (v5)	6.8
PRReLU-net [13]	4.94
BN-inception [16]	4.82
ResNet (ILSVRC'15)	3.57

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

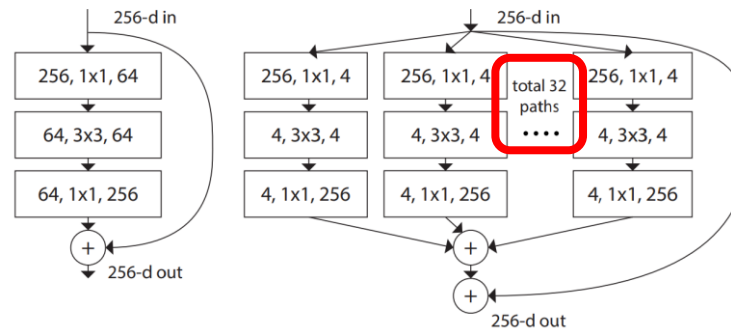
DenseNet (2017)

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



ResNeXT (2017)

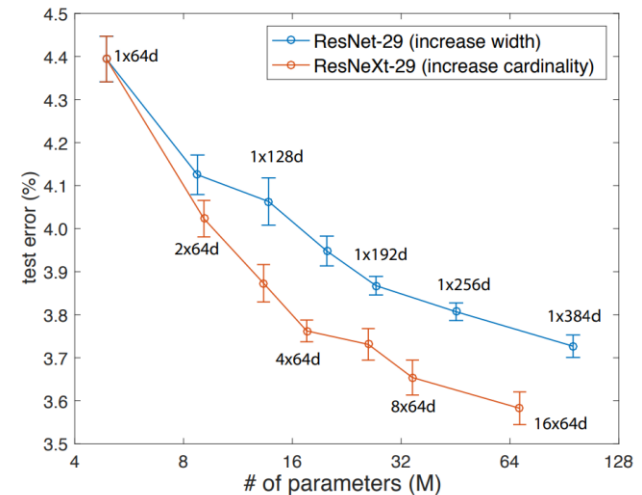
- Deeper and wider → better...what else?
 - Increase cardinality



ResNet block

ResNeXT block

	setting	top-1 error (%)
ResNet-50	1 × 64d	23.9
ResNeXt-50	2 × 40d	23.0
ResNeXt-50	4 × 24d	22.6
ResNeXt-50	8 × 14d	22.3
ResNeXt-50	32 × 4d	22.2
ResNet-101	1 × 64d	22.0
ResNeXt-101	2 × 40d	21.7
ResNeXt-101	4 × 24d	21.4
ResNeXt-101	8 × 14d	21.3
ResNeXt-101	32 × 4d	21.2



Various Deep Learning Models...

