Computer Vision: from Recognition to Geometry

Lecture 5: Deep Learning Basics

http://media.ee.ntu.edu.tw/courses/cv/18F/

FB: NTUEE Computer Vision Fall 2018

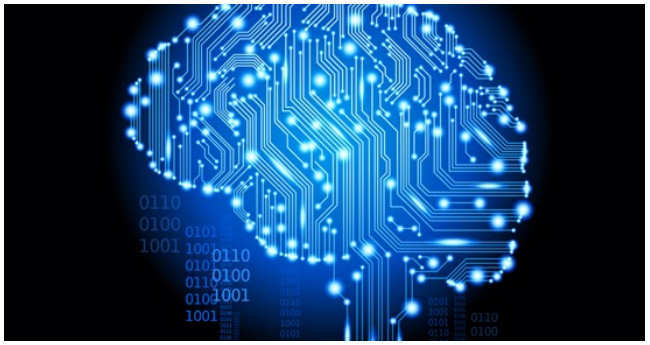
Yu-Chiang Frank Wang 王鈺強, Associate Professor

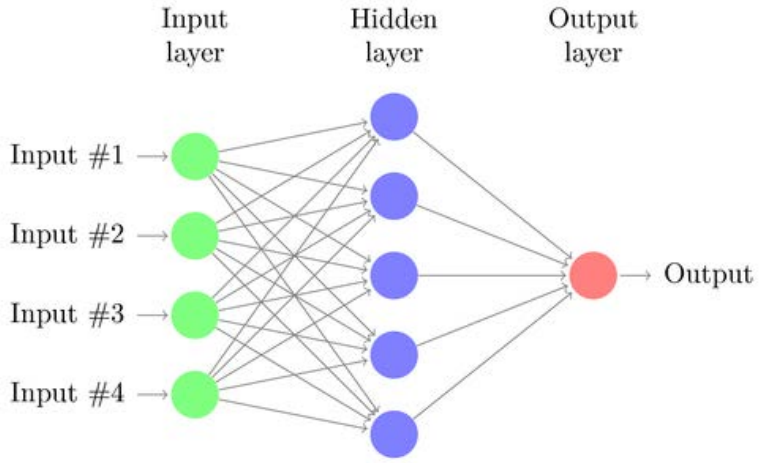
Dept. Electrical Engineering, National Taiwan University

2018/10/17

What's to Be Covered Today…

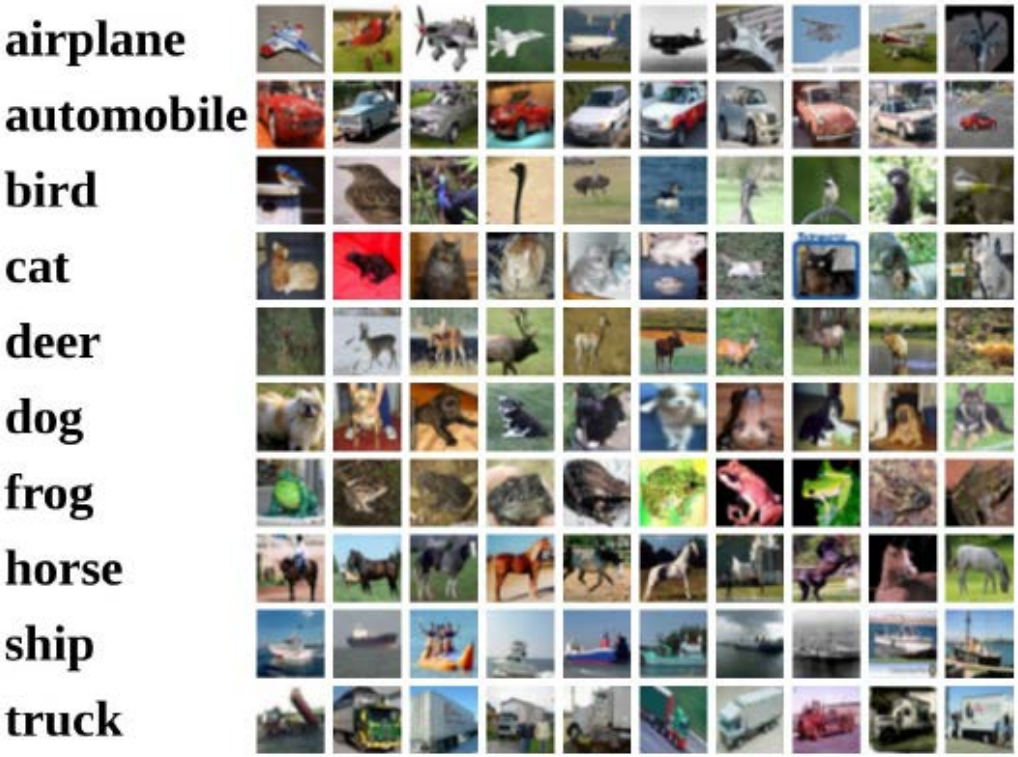
- Supervised Learning: Linear Classification
- Deep Learning Basics
  - Neural Network for Machine Vision
  - Multi-Layer Perceptron
  - Convolutional Neural Networks





Linear Classification

- Linear Classifier
  - Can be viewed as a parametric approach. Why?
  - Assuming that we need to recognize 10 object categories of interest
  - E.g., CIFAR10 with 50K training & 10K test images of 10 categories.
    And, each image is of size 32 x 32 x 3 pixels.



Linear Classification (cont'd)

- Linear Classifier
  - Can be viewed as a parametric approach. Why?
  - Assuming that we need to recognize 10 object categories of interest (e.g., CIFAR10).
  - Let's take the input image as **x**, and the linear classifier as **W**. We hope to see that **y** = **Wx** + **b** as a 10-dimensional output indicating the score for each class.

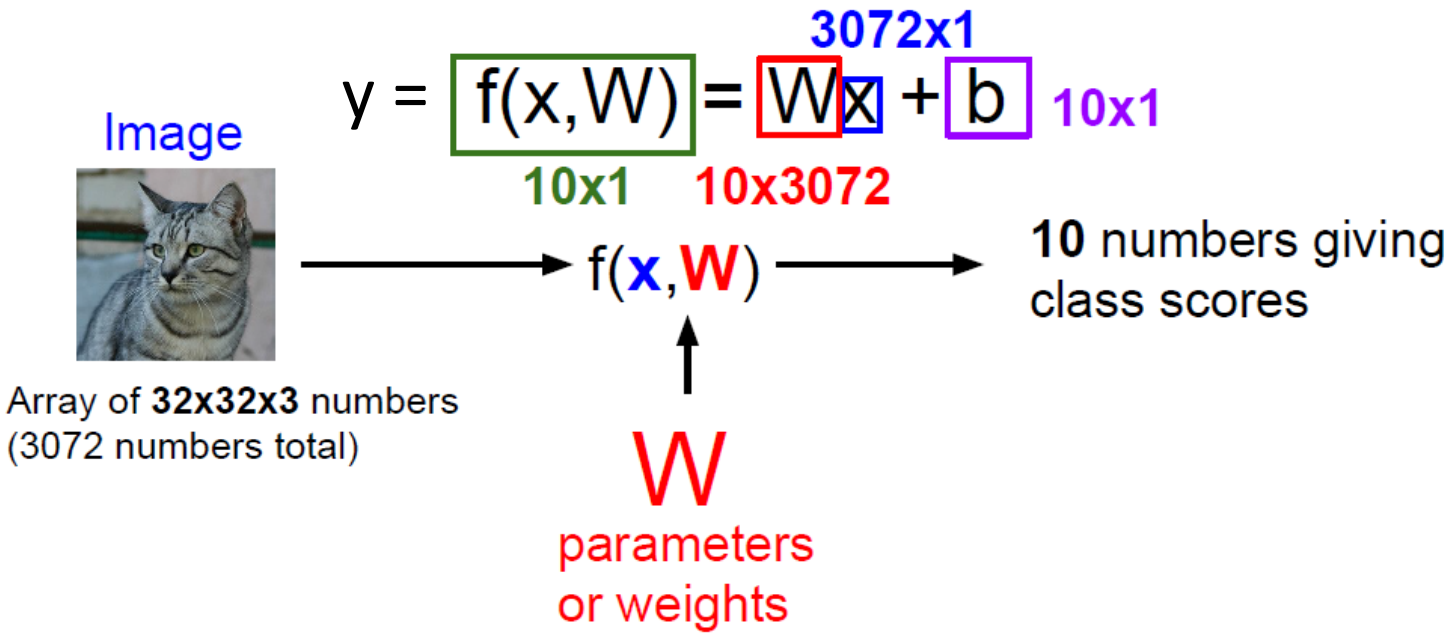


Image credit: Stanford CS231n

Linear Classification (cont'd)

- Linear Classifier
  - Can be viewed as a parametric approach. Why?
  - Assuming that we need to recognize 10 object categories of interest (e.g., CIFAR10).
  - Let's take the input image as **x**, and the linear classifier as **W**. We hope to see that **y** = **Wx** + **b** as a 10-dimensional output indicating the score for each class.
  - Take an image with 2 x 2 pixels & 3 classes of interest as example:
    we need to learn linear transformation/classifer **W** and bias **b**,
    so that desirable outputs **y** = **Wx** + **b** can be expected.

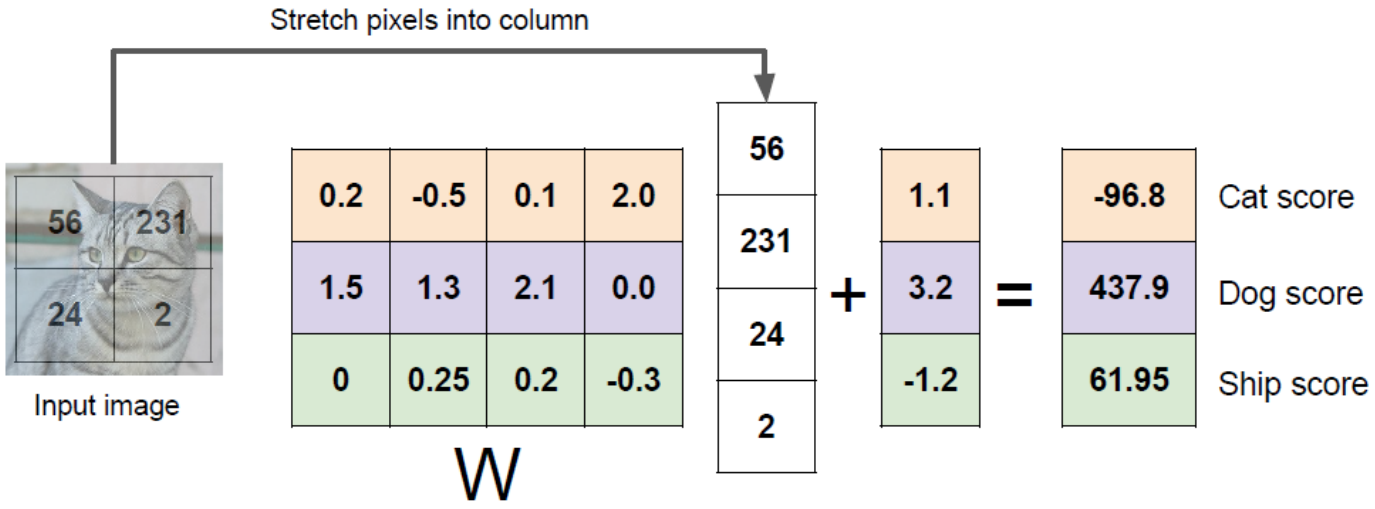
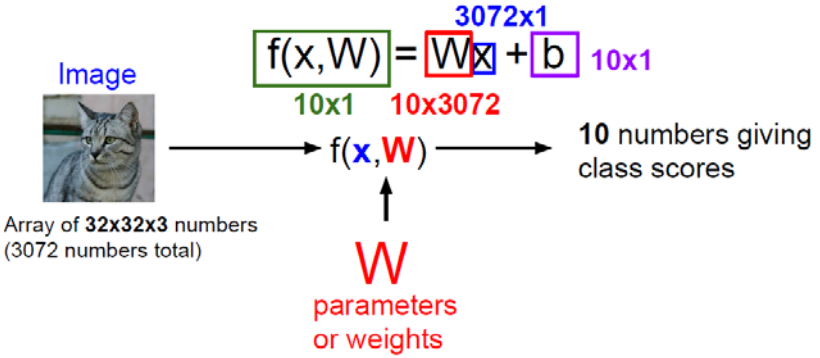
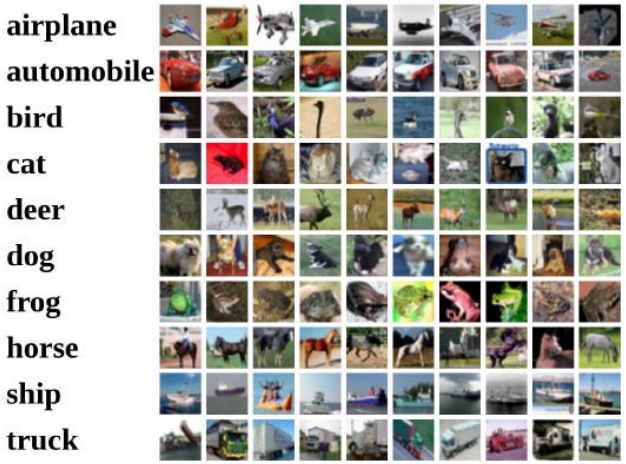


Image credit: Stanford CS231n

Some Remarks



- Interpreting **y** = **Wx** + **b**
  - What can we say about the learned **W**?
  - The weights in **W** are trained by observing training data **X** and their ground truth **Y**.
  - Each column in **W** can be viewed as an exemplar of the corresponding class.
  - Thus, **Wx** basically performs inner product (or correlation) between the input **x** and the exemplar of each class. (Signal & Systems!)



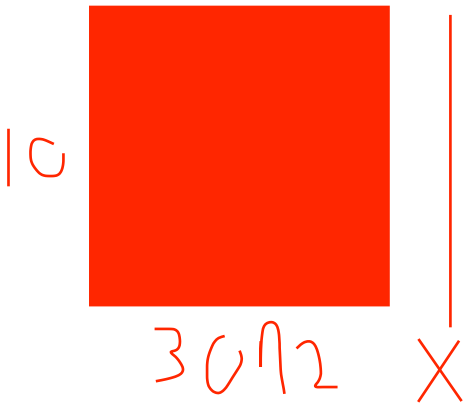
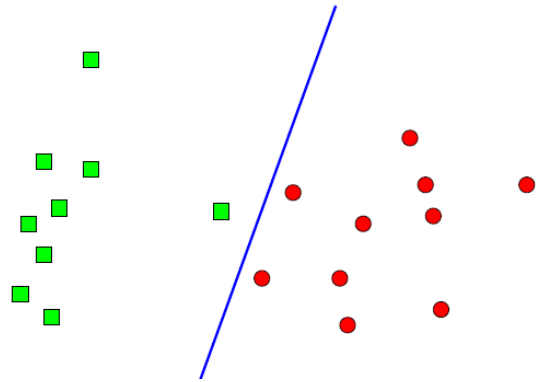


Image credit: Stanford CS231n

From Binary to Multi-Class Classification



e.g., Support Vector Machine (SVM)

二元分類器

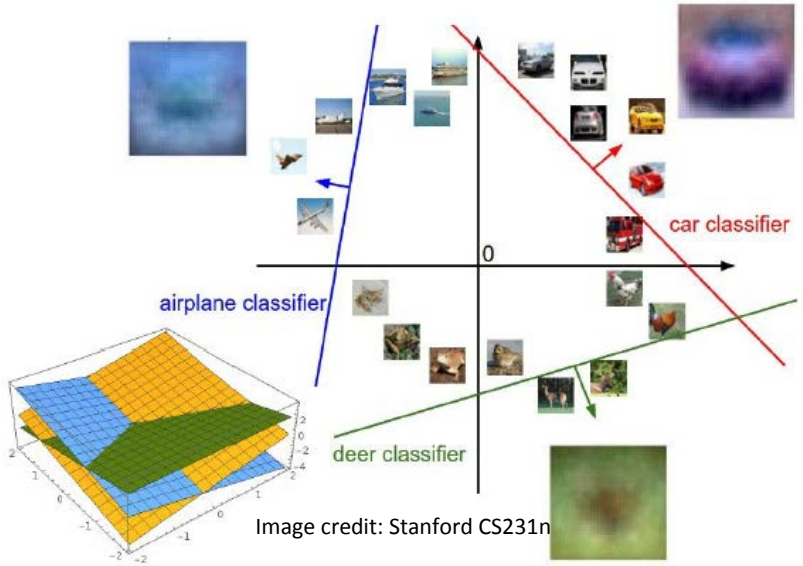


Image credit: Stanford CS231n

- How to extend binary classifiers for solving multi-class classification?
  - 1-vs.-1 (1-against-1) vs. 1-vs.-all (1-against-rest)

兩兩類別之間就要train一個分類器
1 - 2
2 - 3
1- 3
投票

每次專心在一個類別
1 - 2&3
2 - 1&3
3 - 1&2
贏者全拿

k classes
左 - C(k,2)個
右 - k個
未必誰好誰壞

Linear Classification

- Remarks
  - Starting points for many multi-class or complex/nonlinear classifier
  - How to determine a proper loss function for matching **y** and **Wx+b**, and thus how to learn the model **W** (including the bias **b**), are the keys to the learning of an effective classification model.

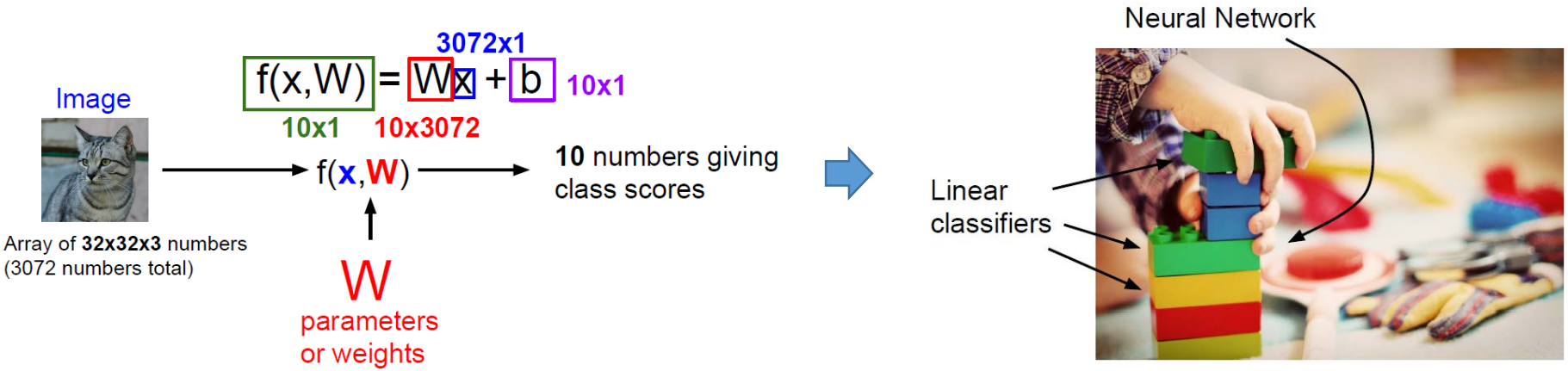


Image credit: Stanford CS231n

Supervised Learning for Visual Classification

- General framework

Image Data

pre-processing

+

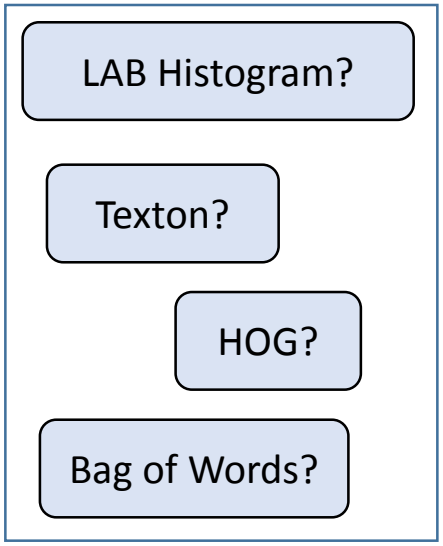
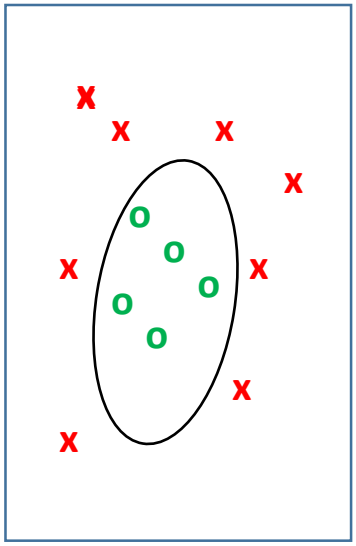


Image Features

+

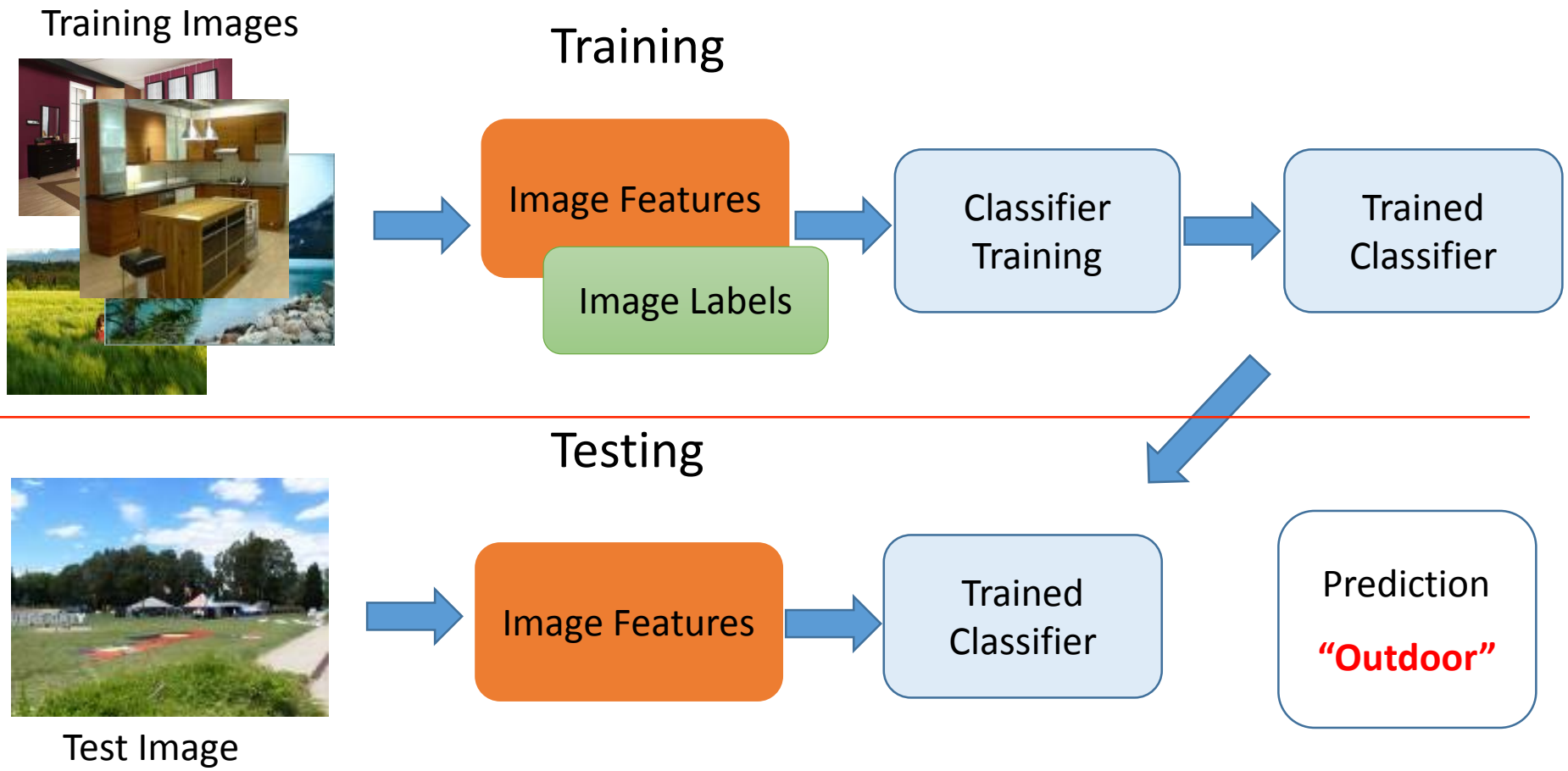


Classifier

= Category label

Supervised Learning for Visual Classification

- Training vs. Testing Phases



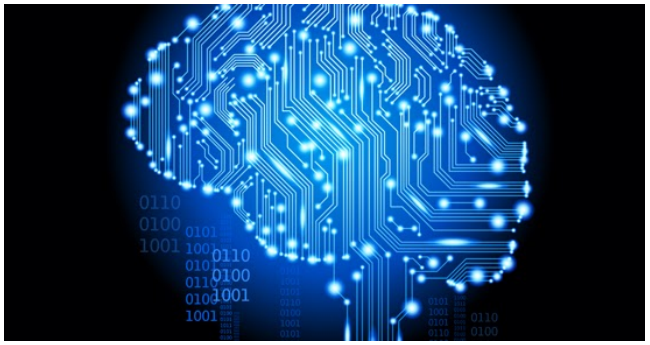
What Are the Right Features?

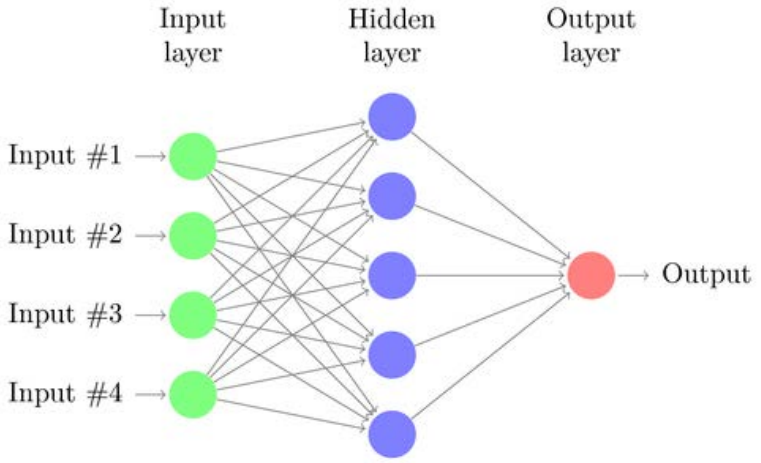
- Depending on the task of interest!
- Possible choices
  - Object: shape
    - Local shape info, shading, shadows, texture
  - Scene : geometric layout
    - linear perspective, gradients, line segments
  - Material properties: albedo, feel, hardness
    - Color, texture
  - Action: motion
    - Optical flow, tracked points

hand-craft feature

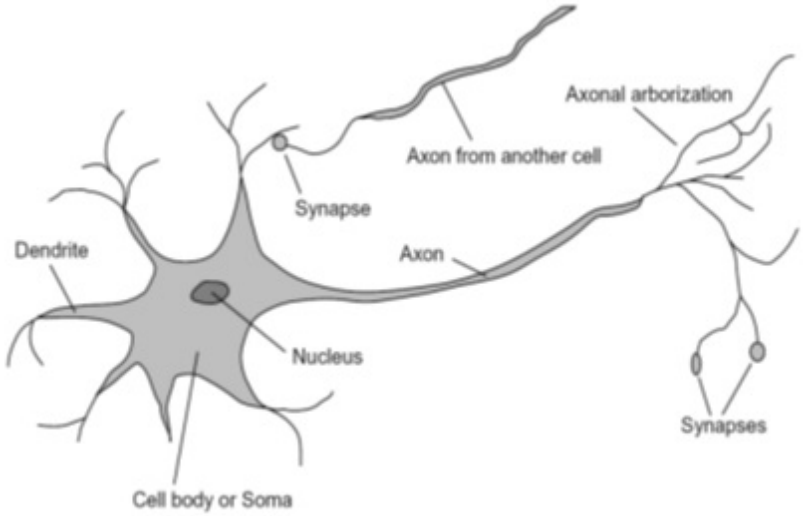
What's to Be Covered Today…

- Supervised Learning: Linear Classification
- Deep Learning Basics
  - Neural Network for Machine Vision
  - Multi-Layer Perceptron
  - Convolutional Neural Networks

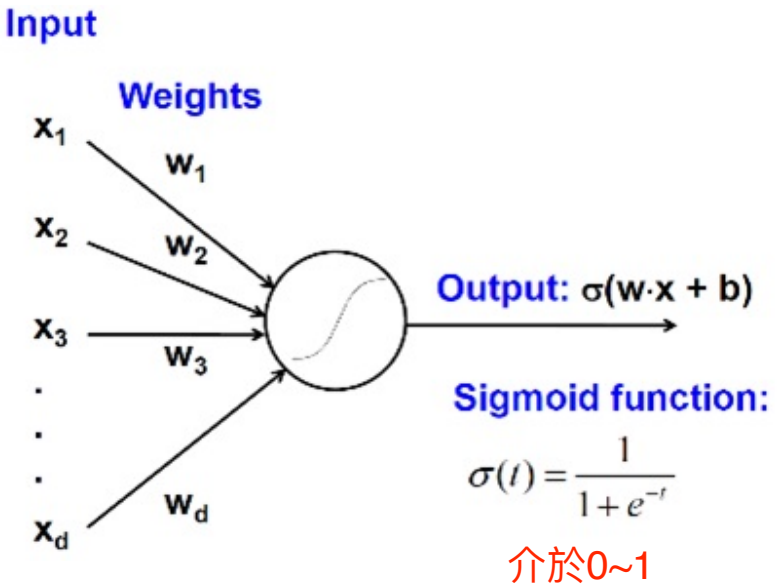




Biological neuron and Perceptrons



A biological neuron

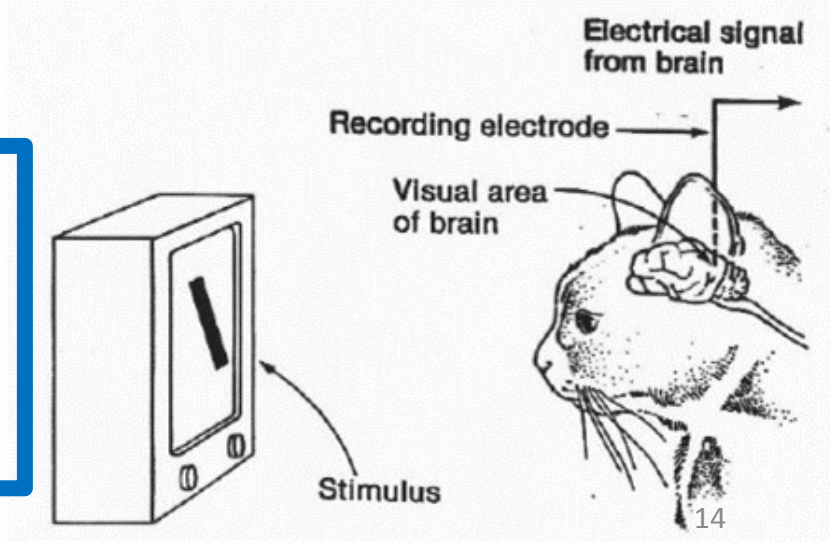


An artificial neuron (Perceptron)
- a linear classifier

Simple, Complex and Hypercomplex cells

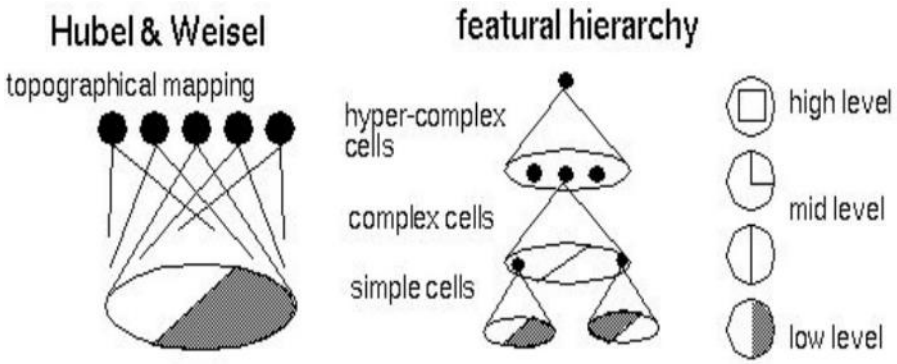
David H. Hubel and Torsten Wiesel

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.

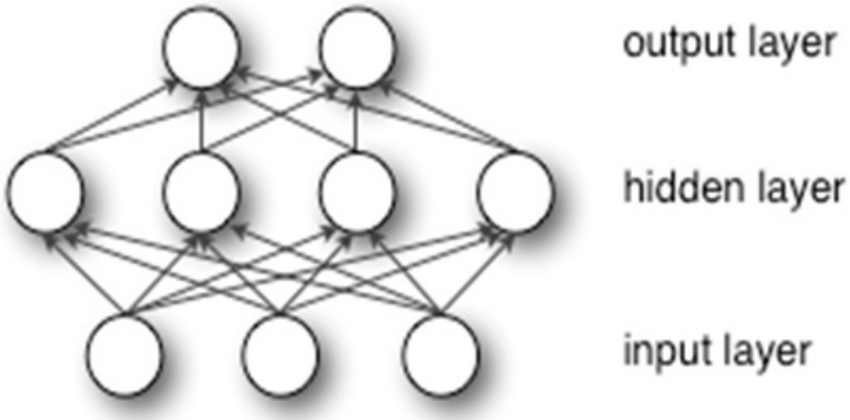


David Hubel's Eye, Brain, and Vision

Hubel/Wiesel Architecture and Multi-layer Neural Network



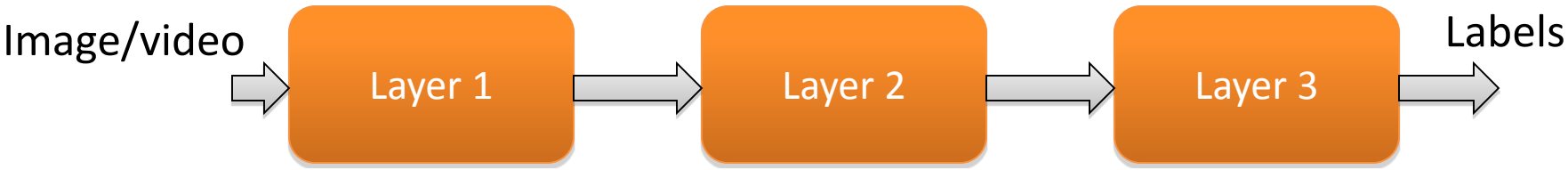
Hubel and Weisel's architecture



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

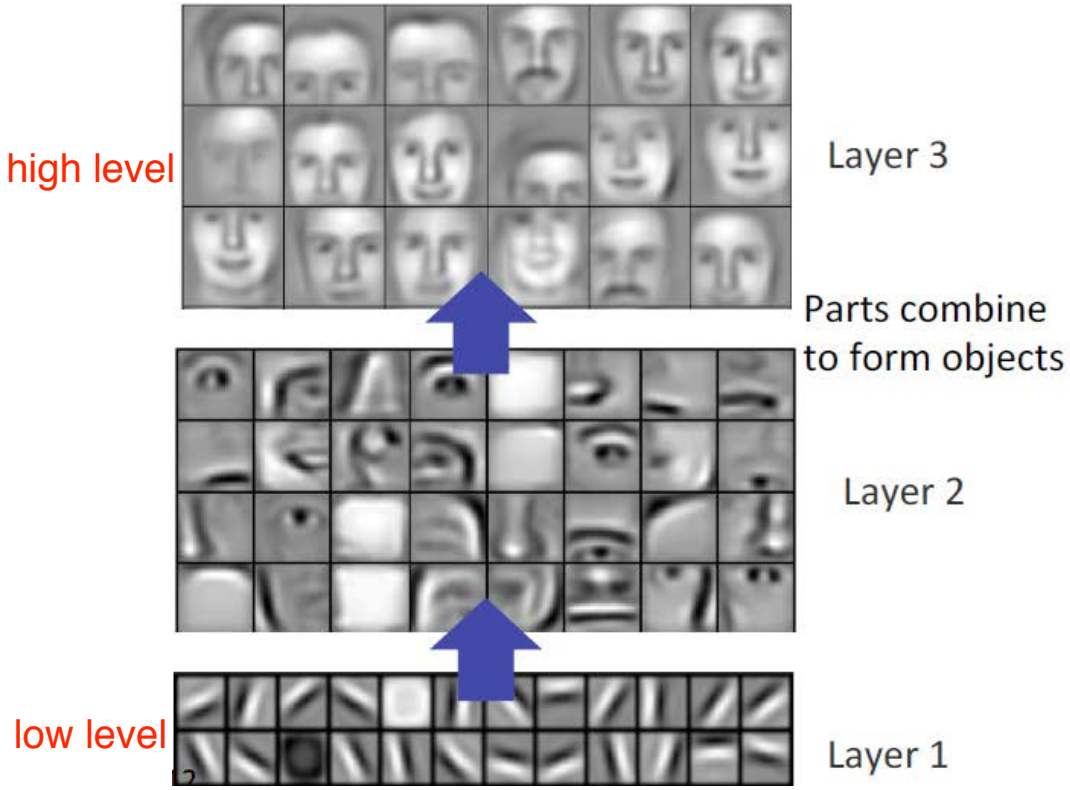
Learning Hierarchy of Feature Extractors

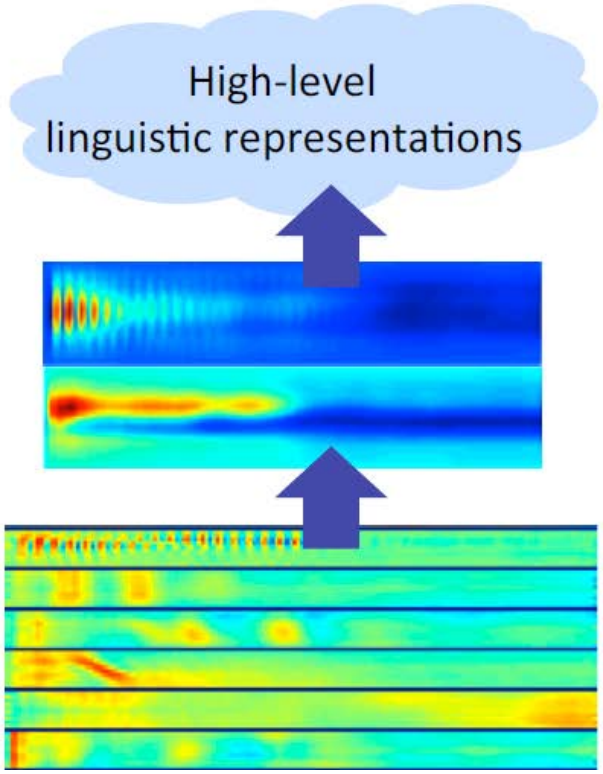
- Each layer of hierarchy extracts features from outputs of the previous layer(s).
- All the way from pixels to classifier
- Layers have the (nearly) same structure



Hierarchical Learning

- Successive model layers learn deeper intermediate representations.





Revisit of Linear Classification

- Linear Classifier
  - Can be viewed as a parametric approach. Why?
  - Assuming that we need to recognize 10 object categories of interest (e.g., CIFAR10).
  - Let's take the input image as **x**, and the linear classifier as **W**. We hope to see that **y** = **Wx** + **b** as a 10-dimensional output indicating the score for each class.

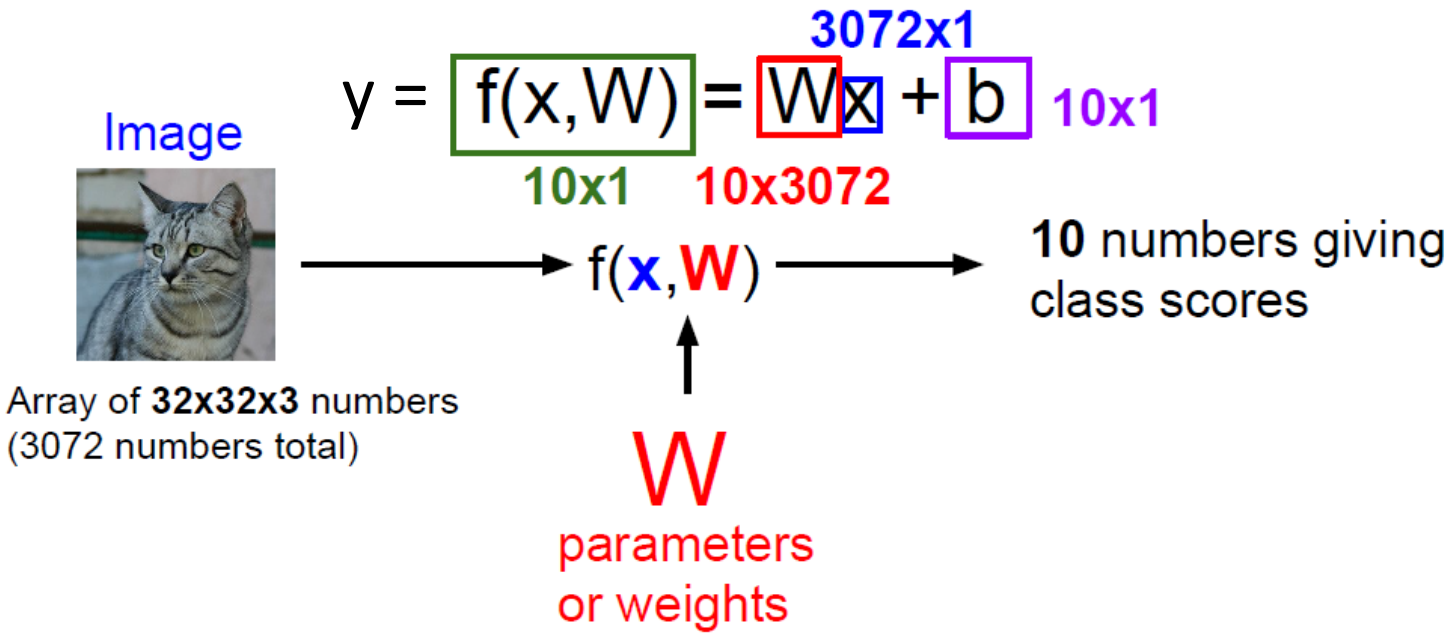
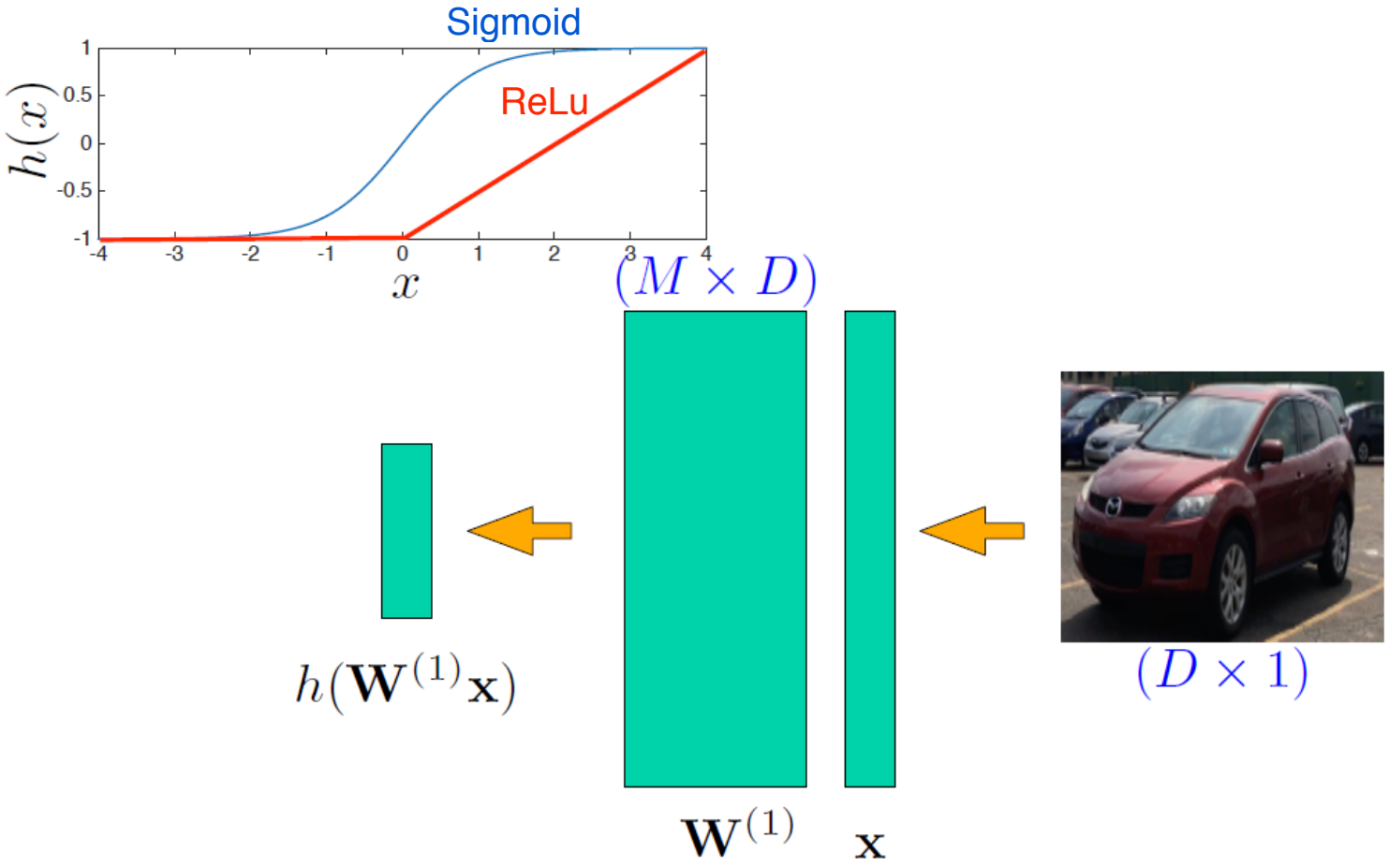
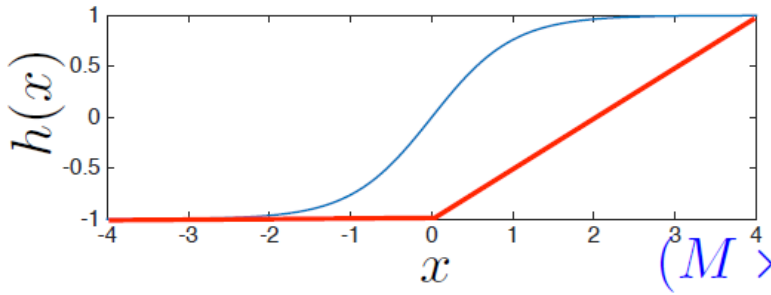


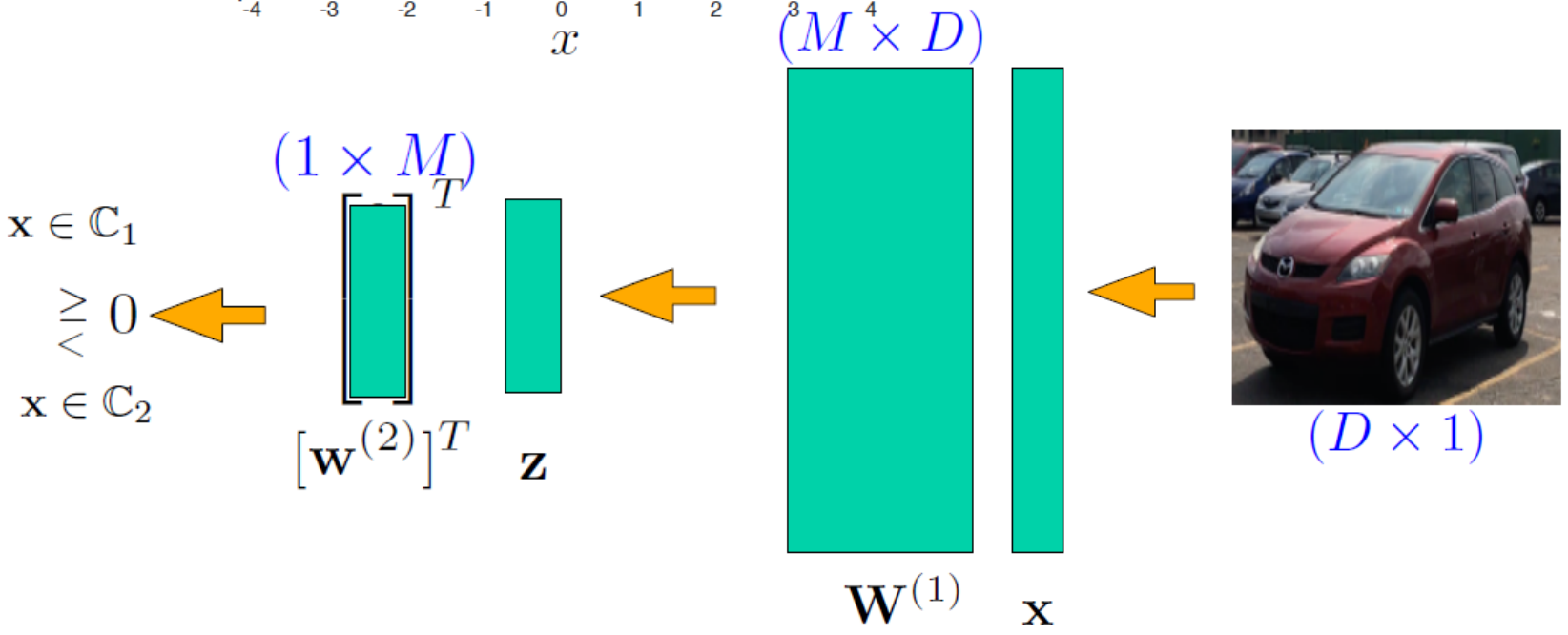
Image credit: Stanford CS231n

Multi-Layer Perceptron: A Nonlinear Classifier

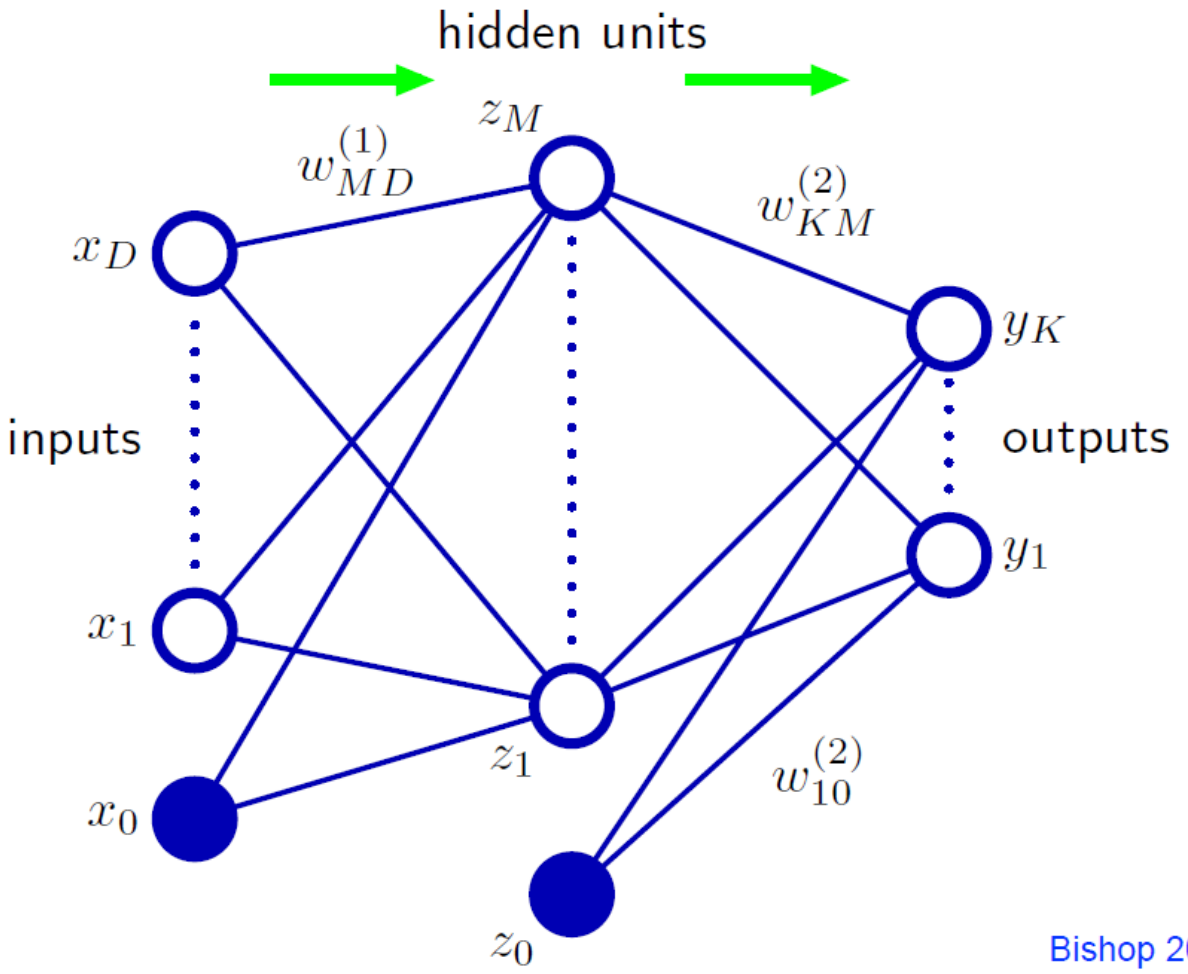


Multi-Layer Perceptron: A Nonlinear Classifier (cont'd)



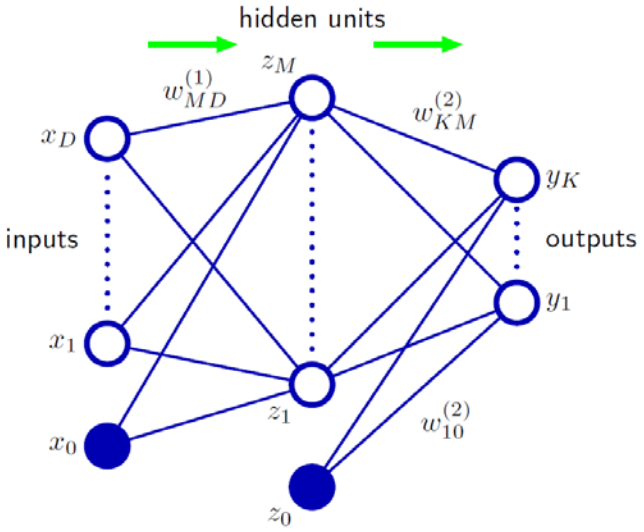


Multi-Layer Perceptron: A Nonlinear Classifier (cont'd)



Bishop 2006

Layer 1 in MLP



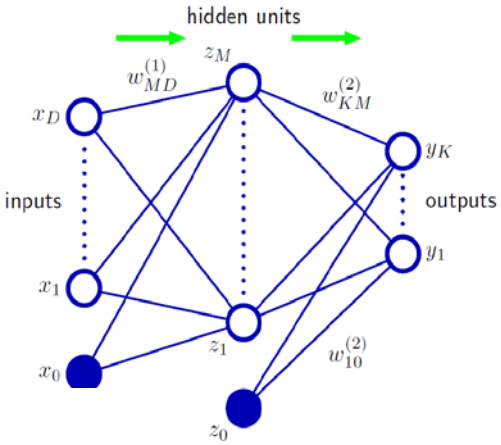
$$\mathbf{z} = \begin{bmatrix} z_1 \\ \vdots \\ z_M \end{bmatrix} \leftarrow \begin{bmatrix} h[\mathbf{x}^T \mathbf{w}_1^{(1)}] \\ \vdots \\ h[\mathbf{x}^T \mathbf{w}_M^{(1)}] \end{bmatrix}$$

$h()$ = non-linear function

$[\mathbf{w}_1^{(1)}, \ldots, \mathbf{w}_M^{(1)}]$ = 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

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

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

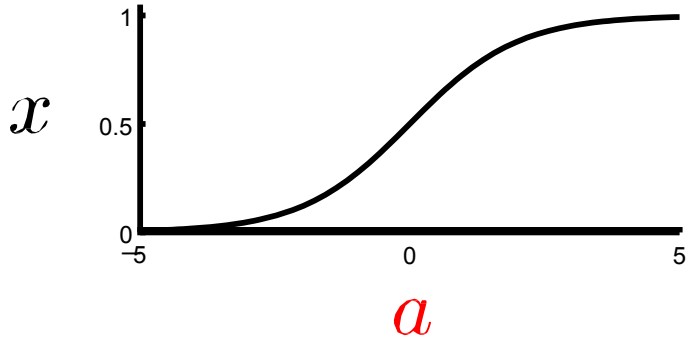
$$\mathbf{z}^T \mathbf{w}^{(2)} \gtrless 0$$

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

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

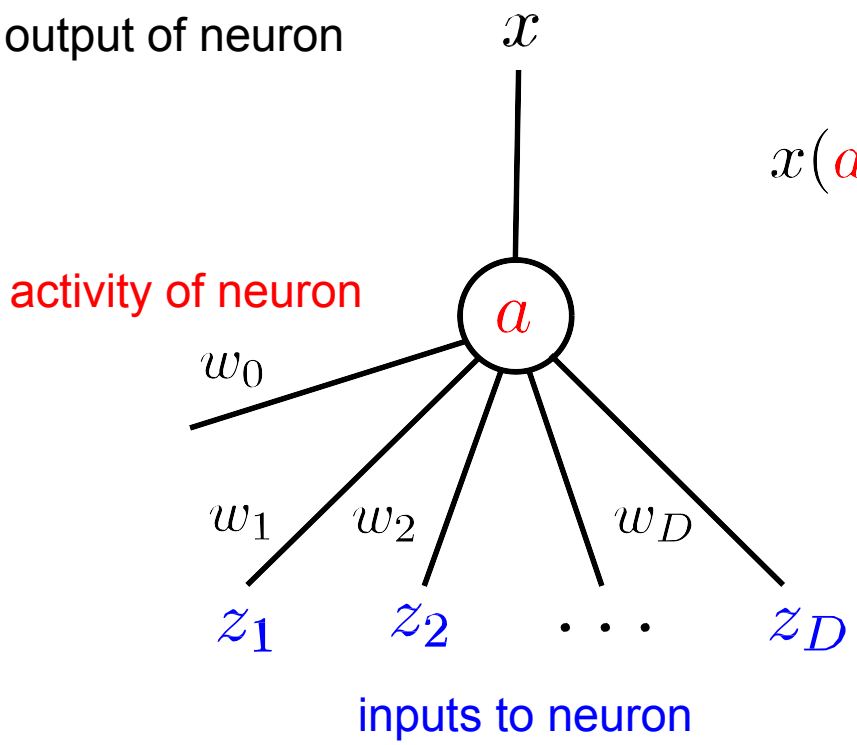
Let's Get a Closer Look…

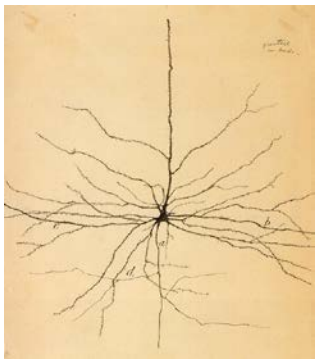
- A single neuron



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

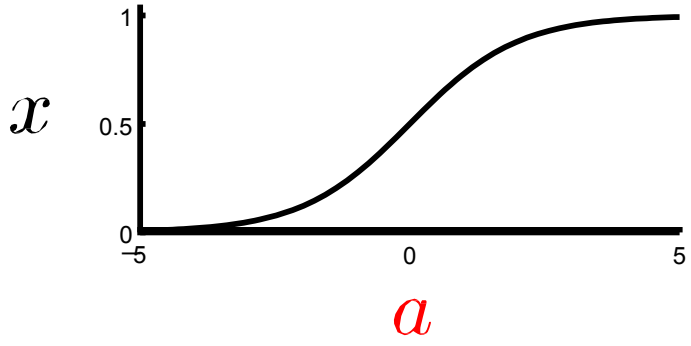
$$a = w_0 + \sum_{d=1}^{D} w_d z_d$$





Let's Get a Closer Look…

- A single neuron



output of neuron x

activity of neuron a

w_0

$1 = z_0$

bias term

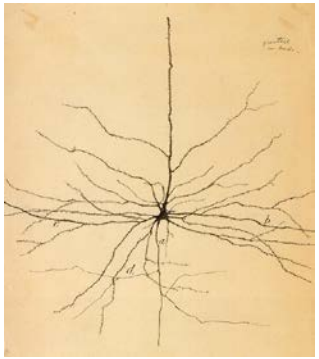
w_1 w_2 w_D

z_1 z_2 $\cdots$ z_D

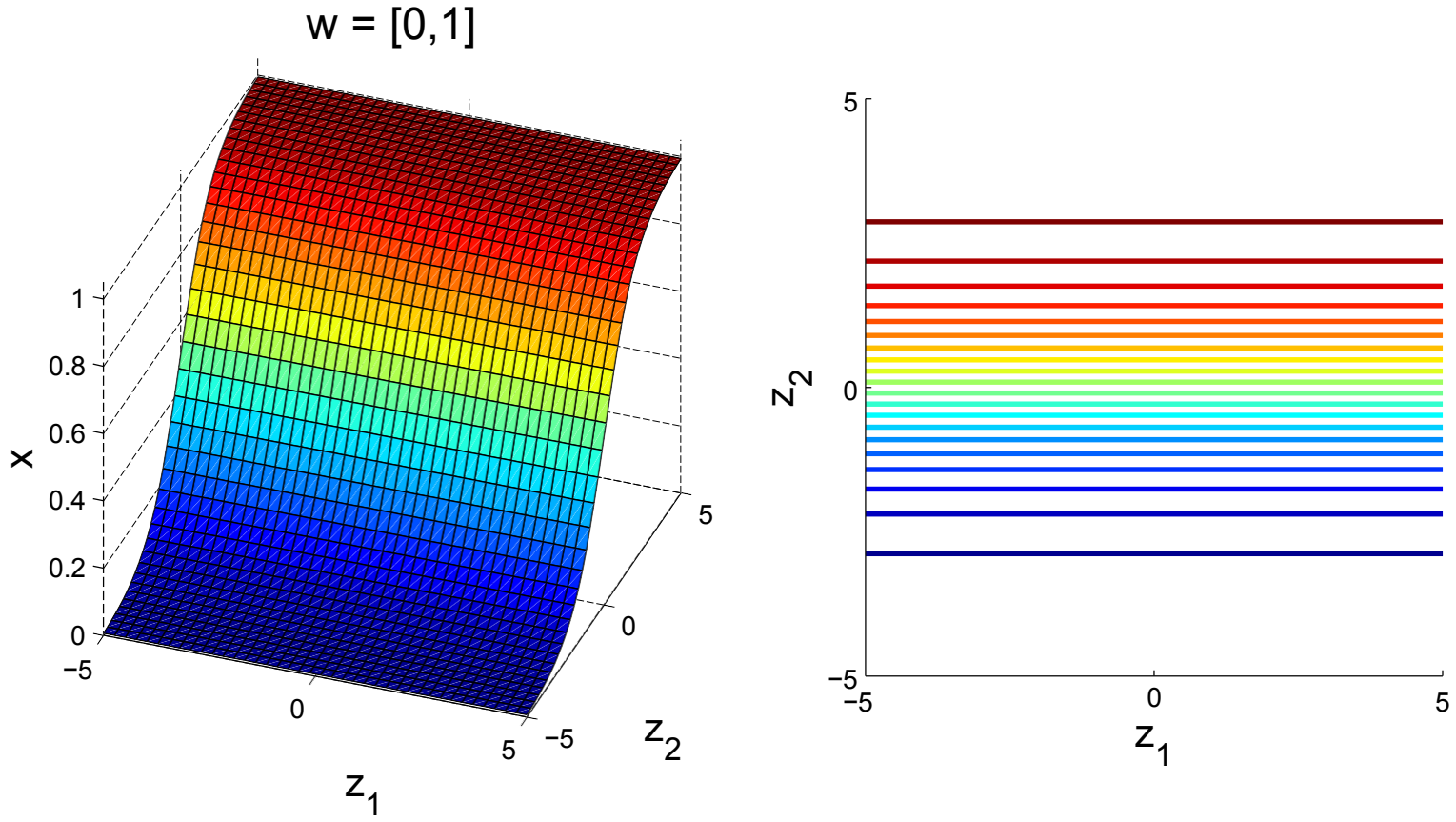
inputs to neuron

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

$$a = \sum_{d=0}^{D} w_d z_d$$

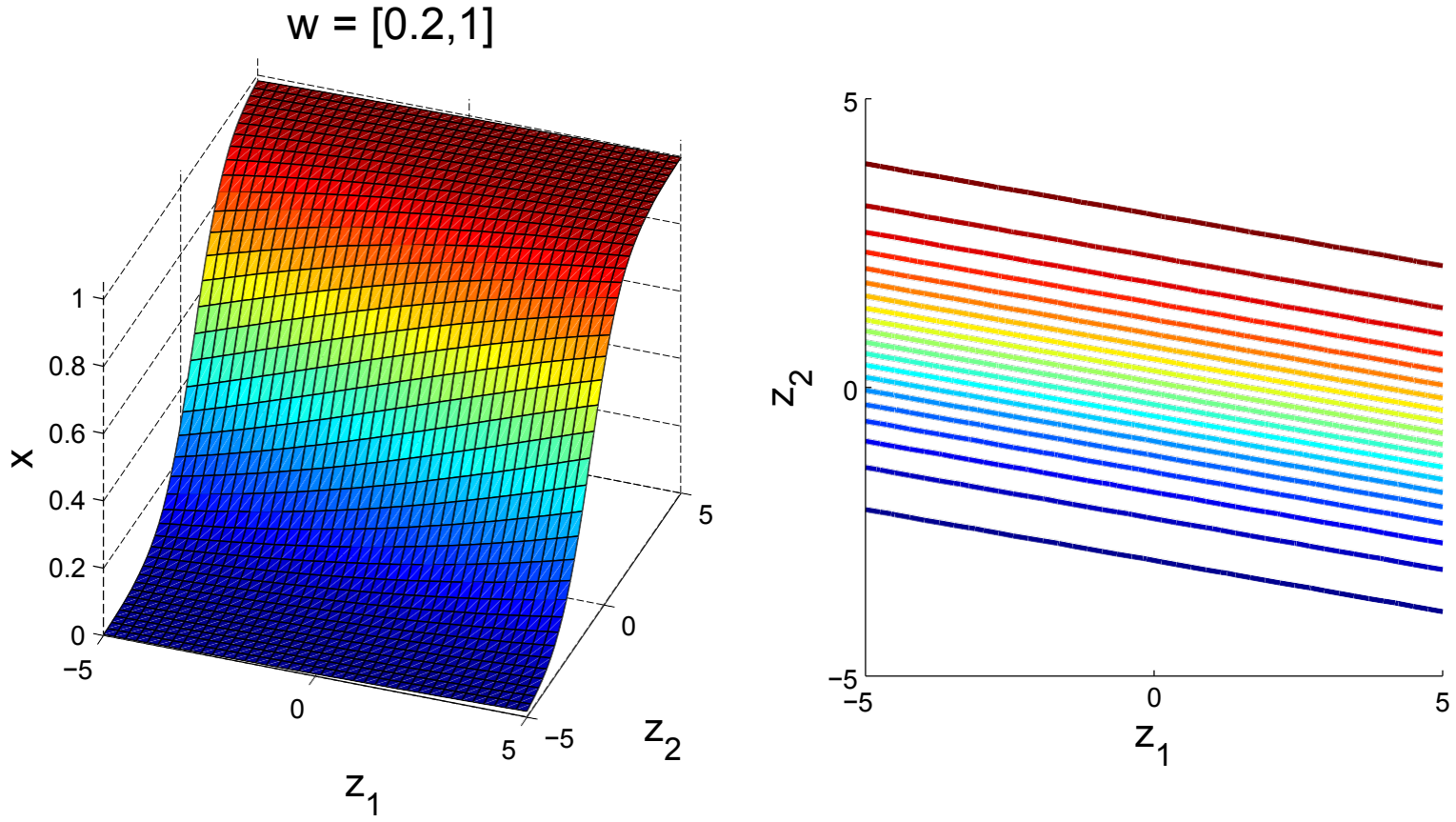


Input-Output Function of a Single Neuron



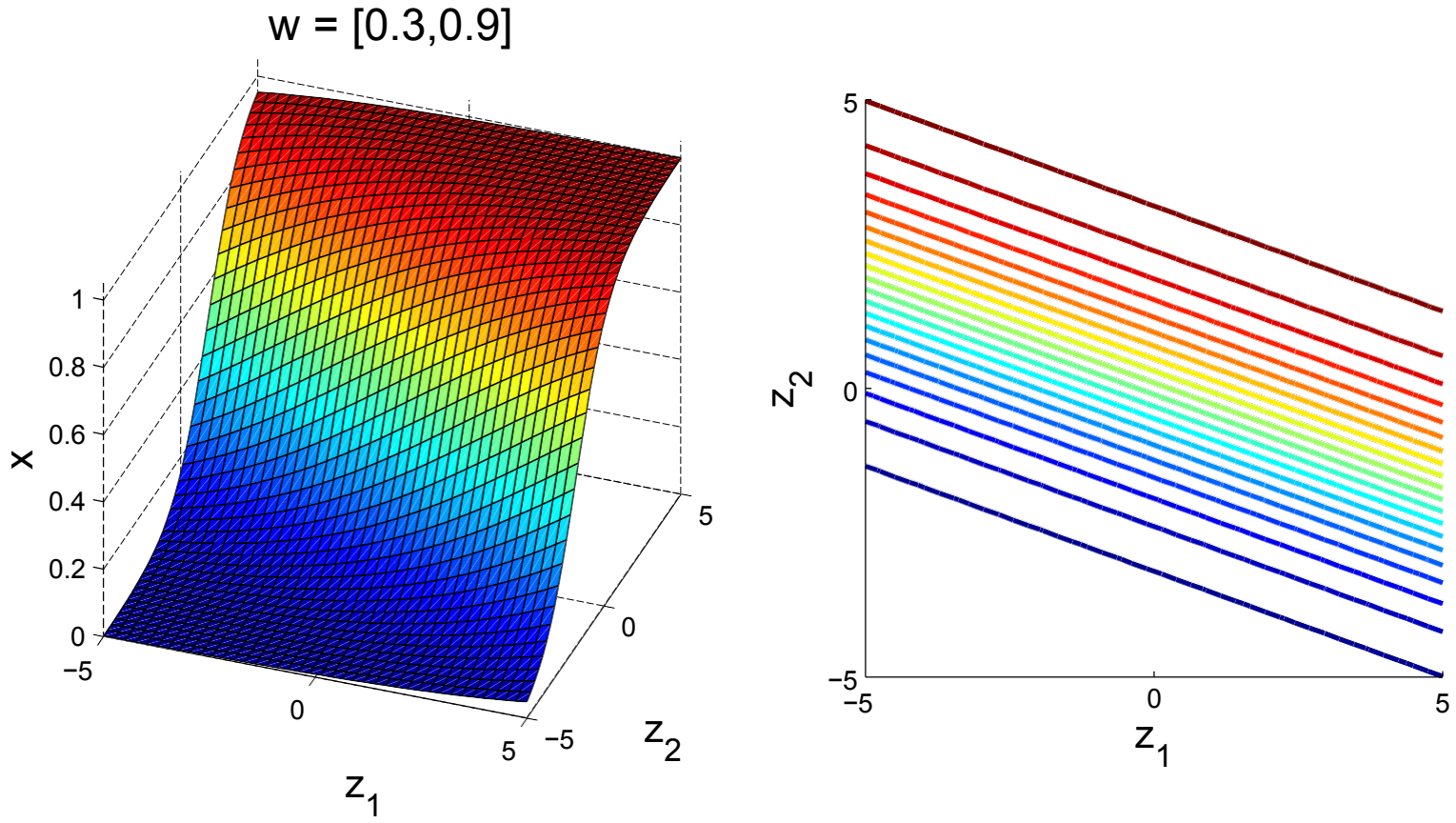
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

Input-Output Function of a Single Neuron



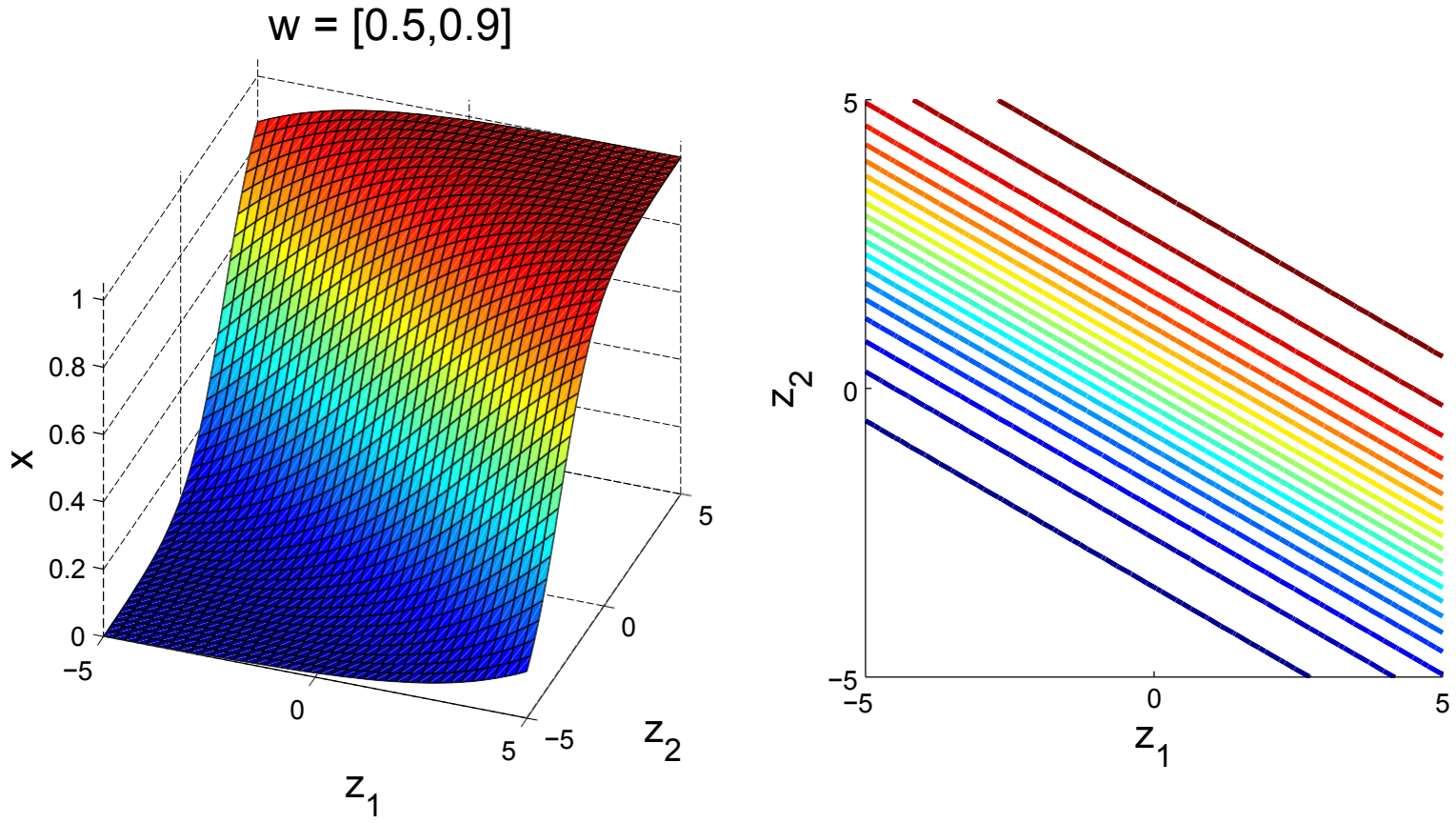
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

Input-Output Function of a Single Neuron



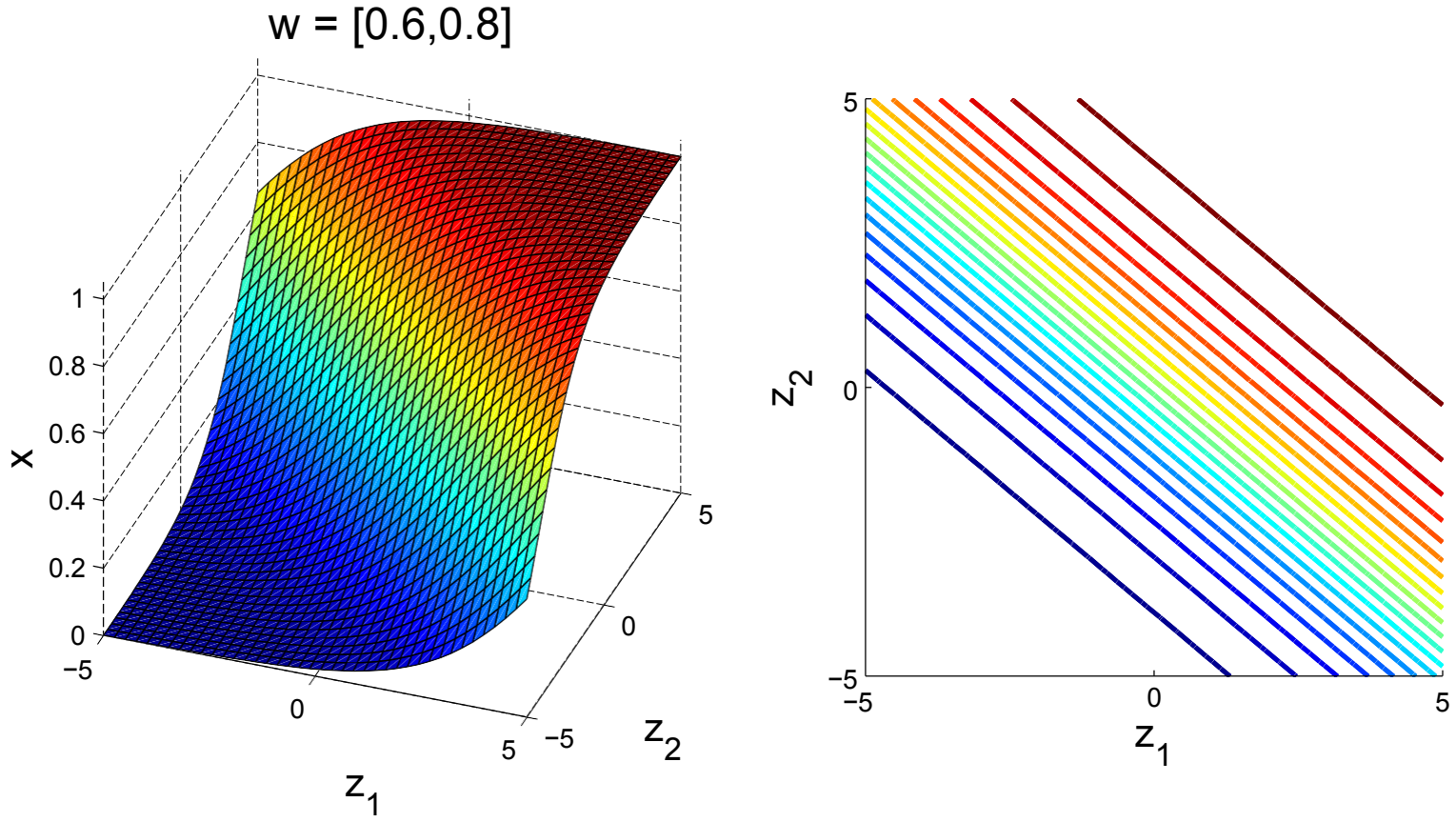
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

Input-Output Function of a Single Neuron



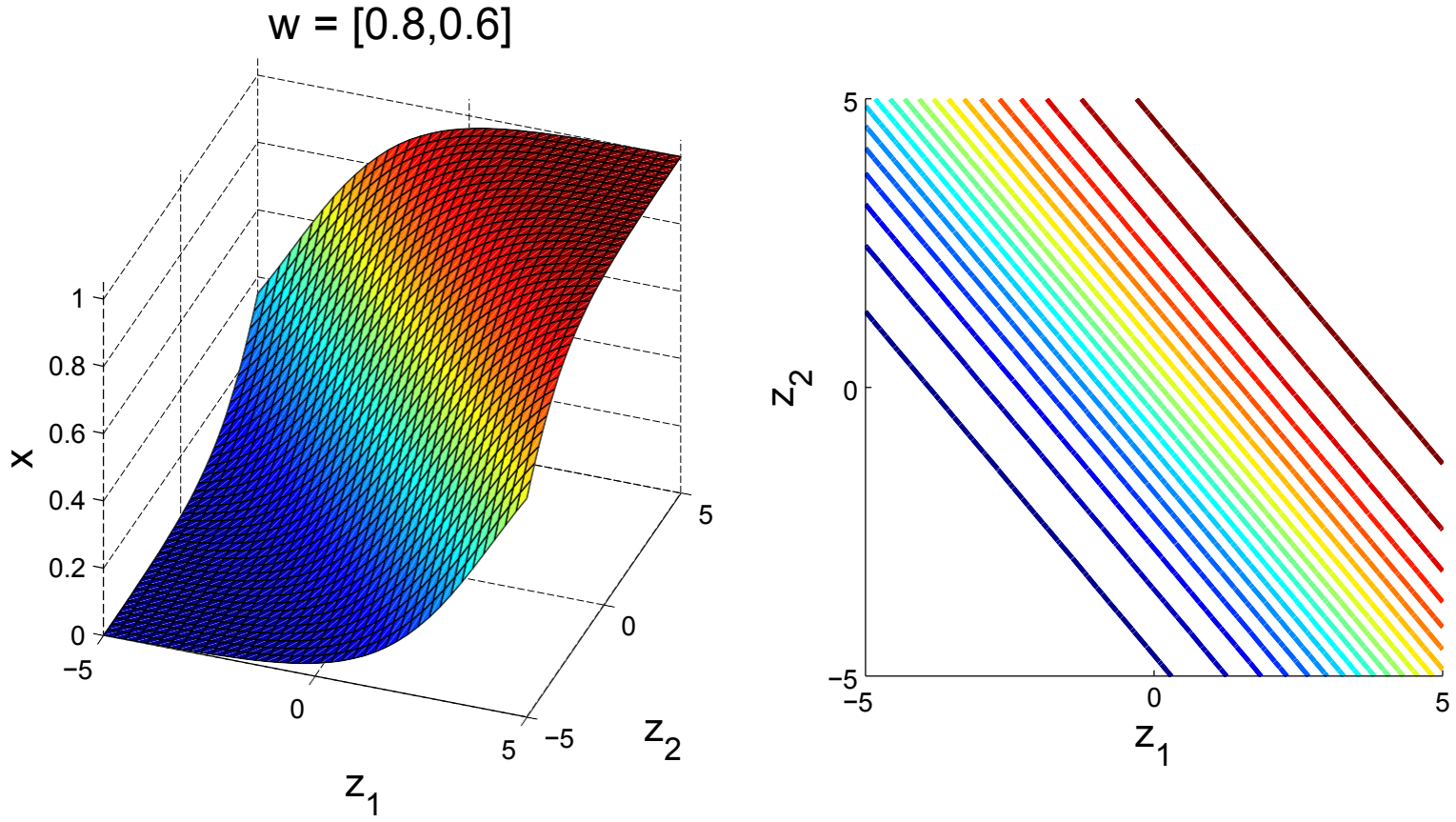
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

Input-Output Function of a Single Neuron



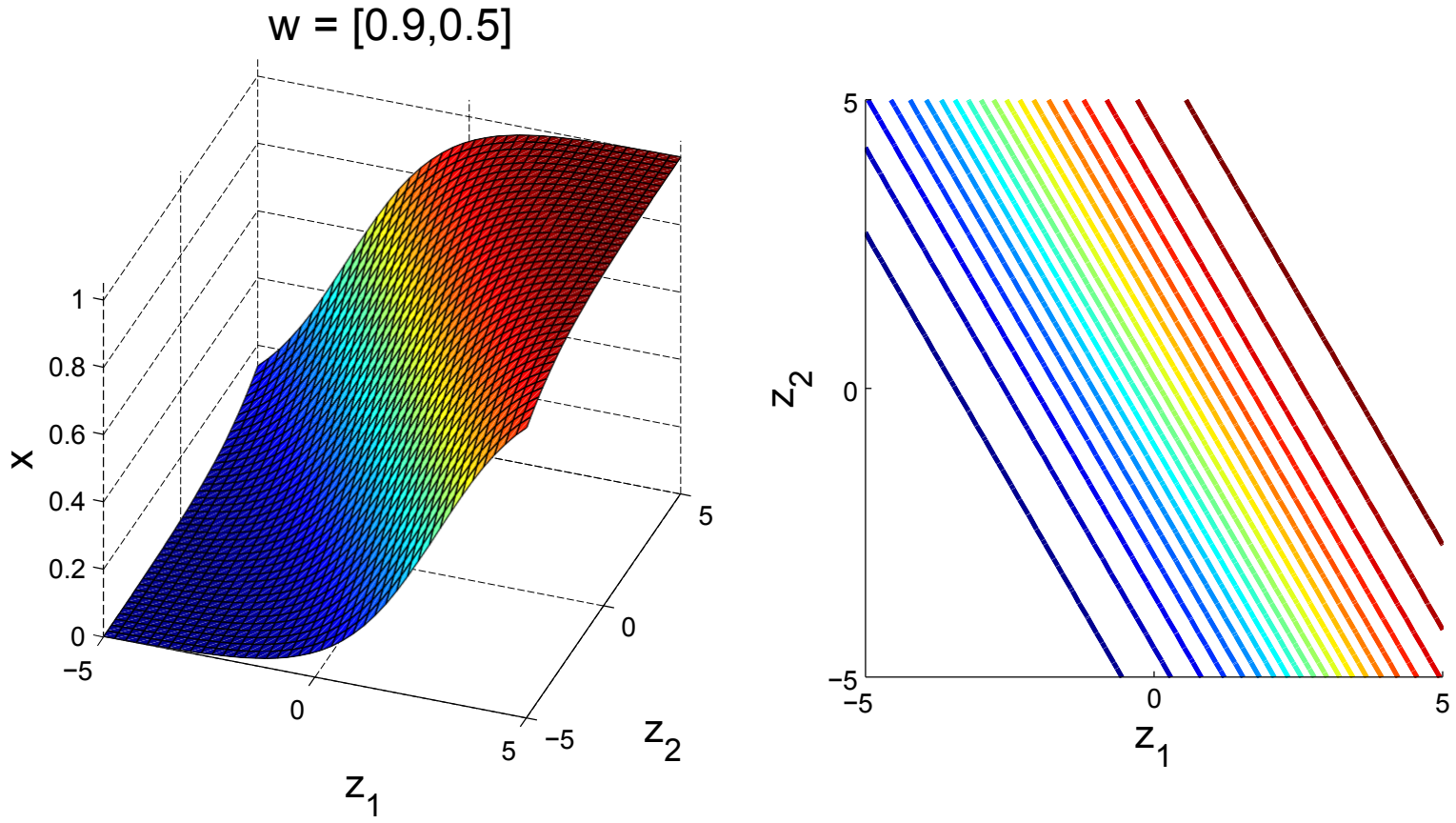
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

Input-Output Function of a Single Neuron



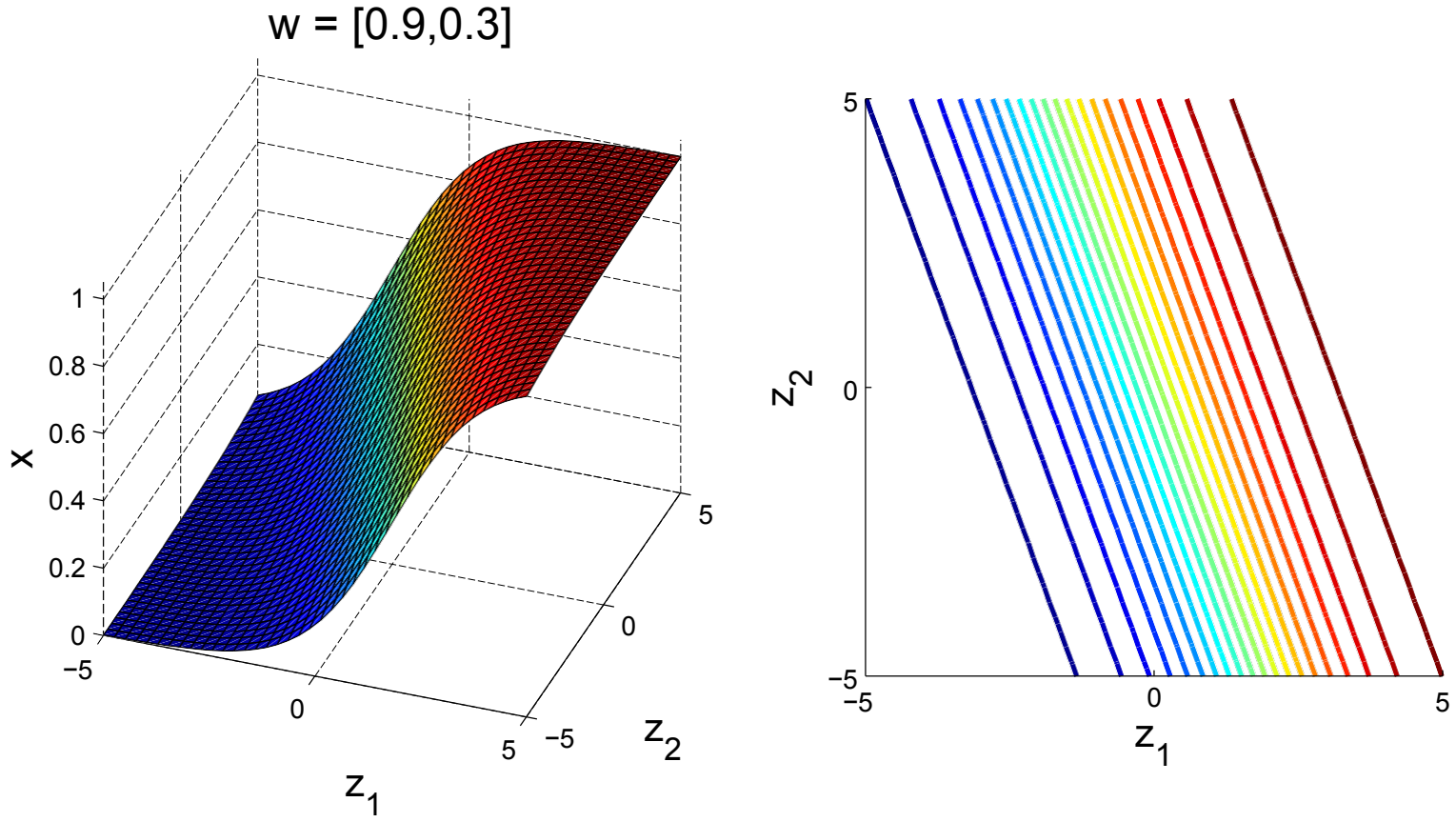
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

Input-Output Function of a Single Neuron



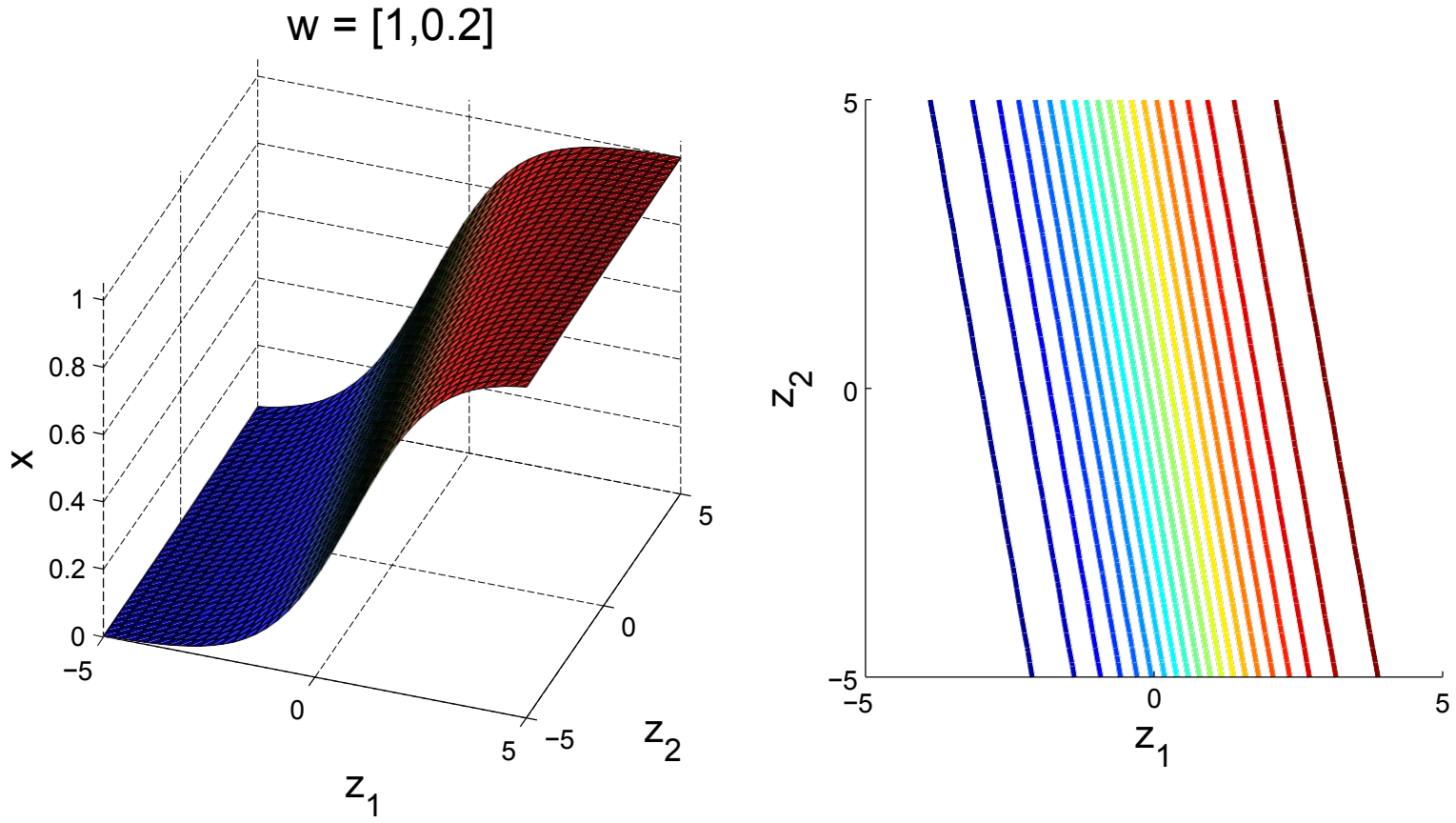
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

Input-Output Function of a Single Neuron



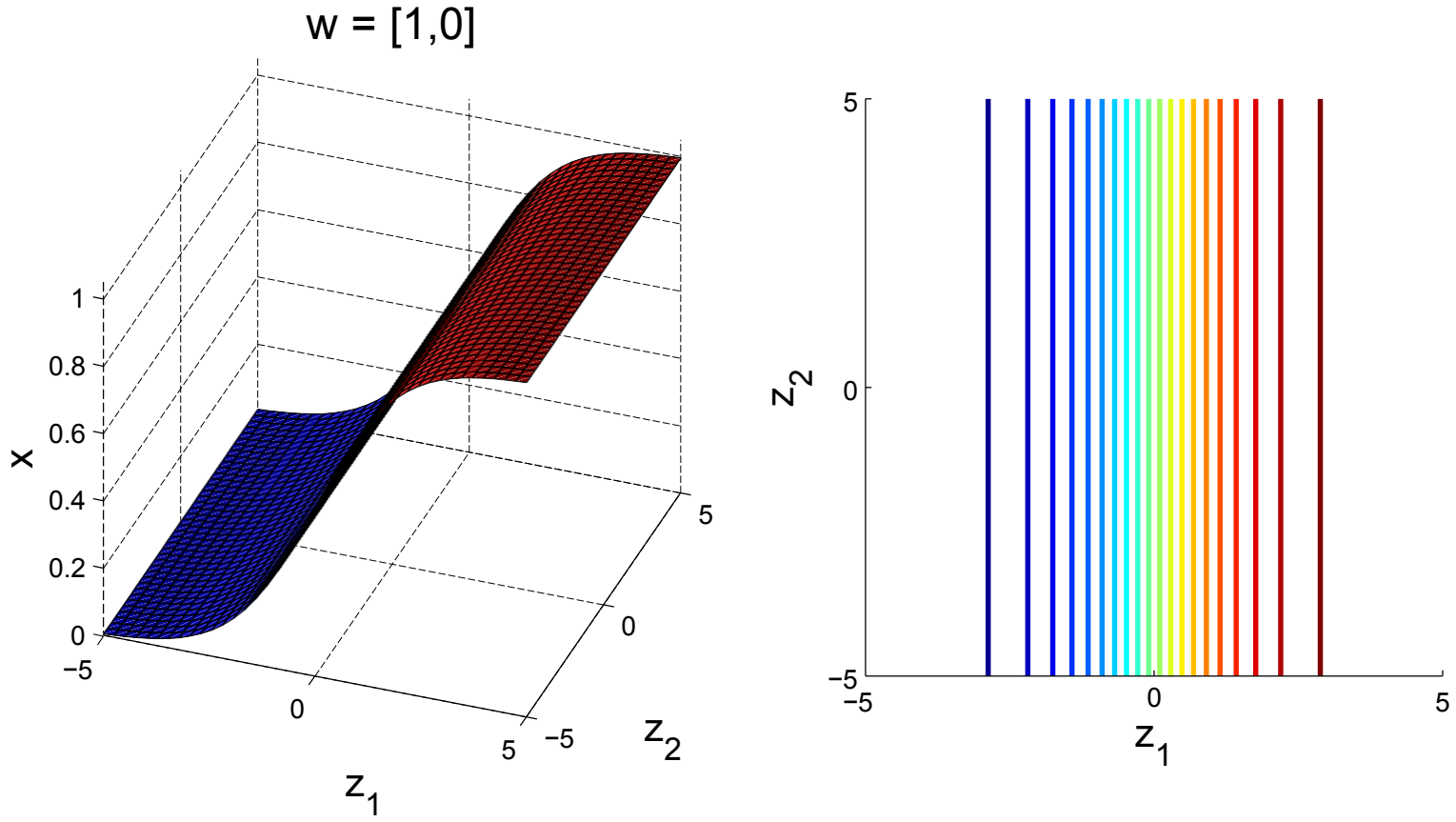
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

Input-Output Function of a Single Neuron



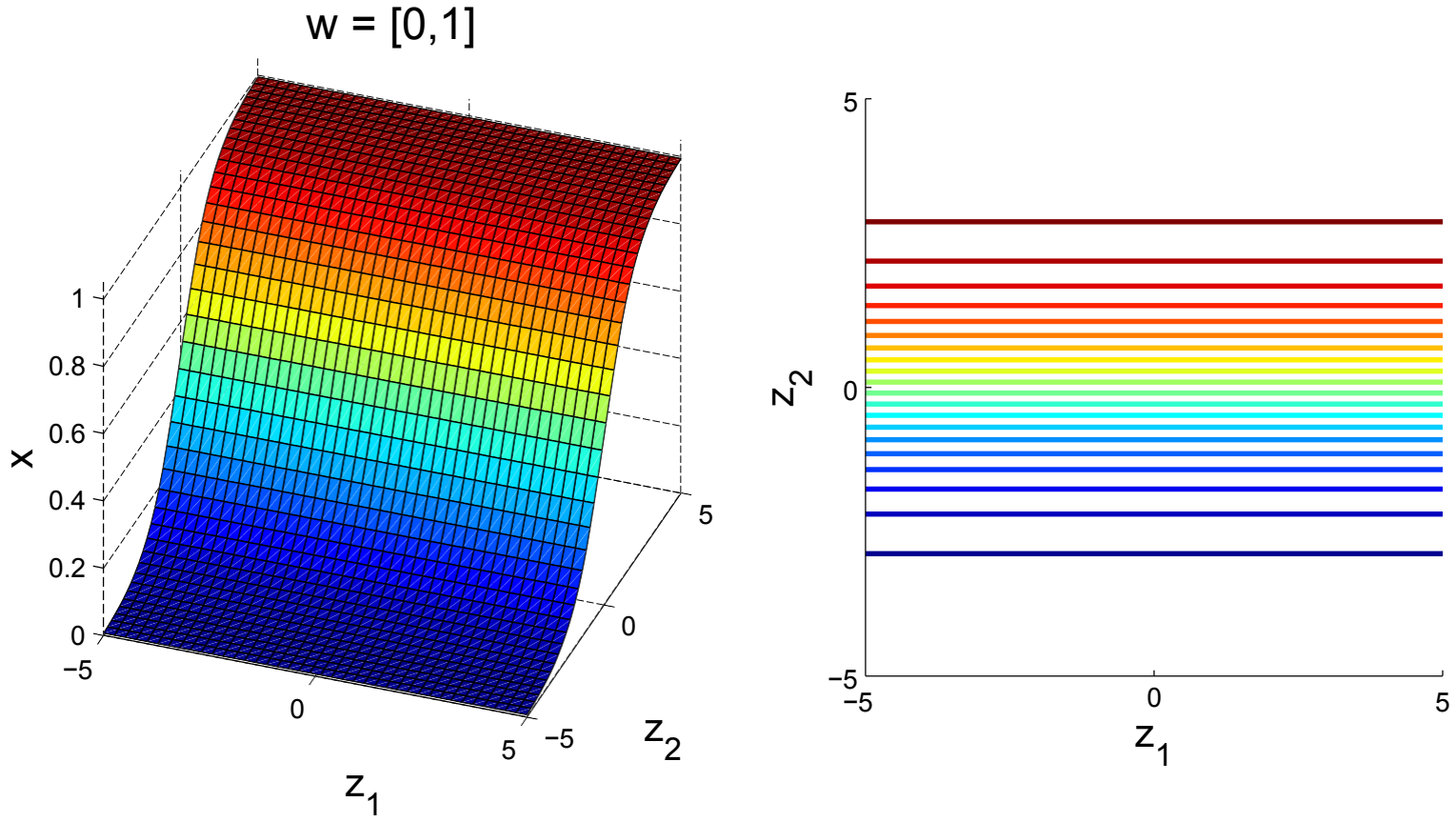
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

Input-Output Function of a Single Neuron



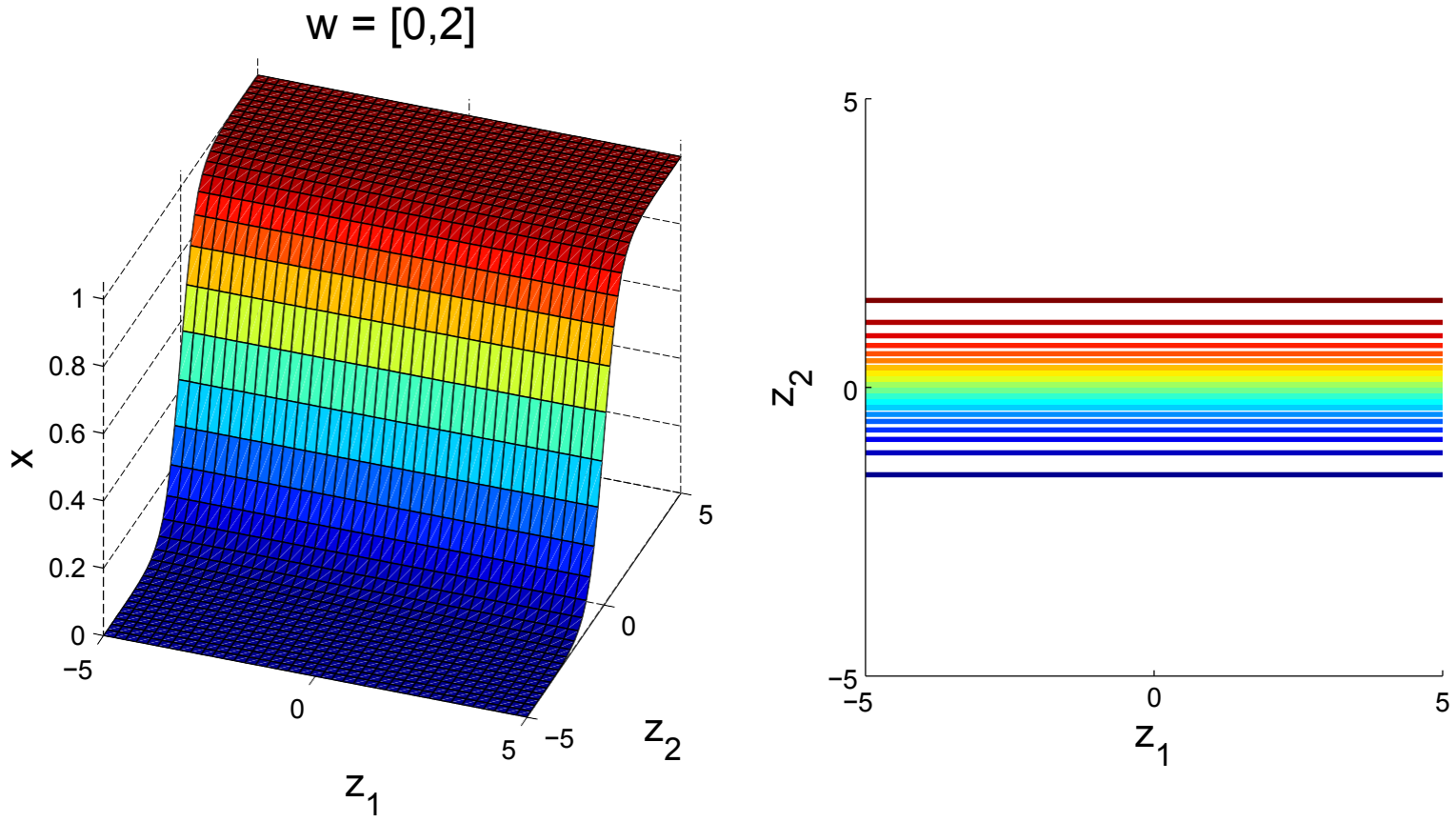
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

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



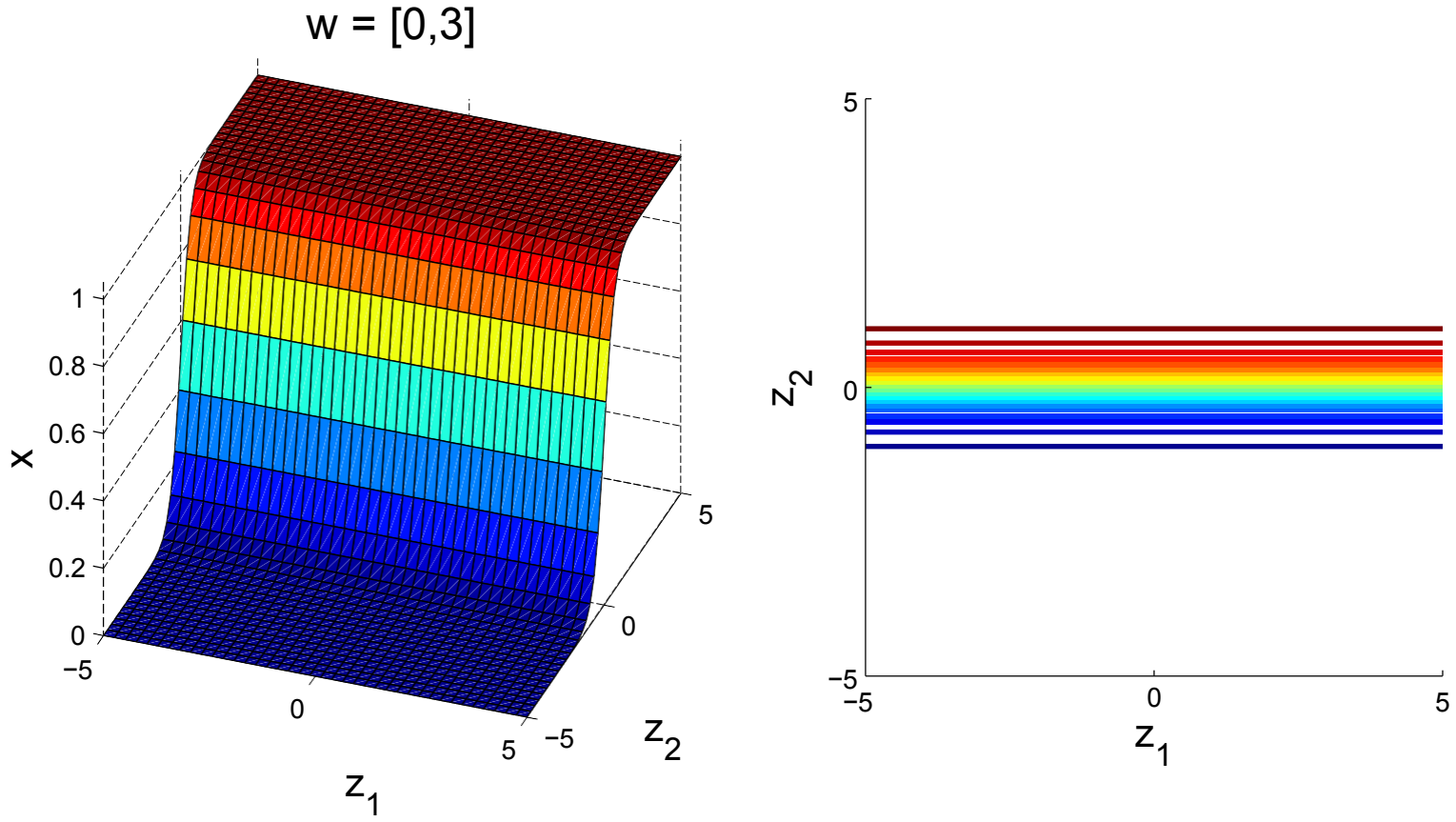
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

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



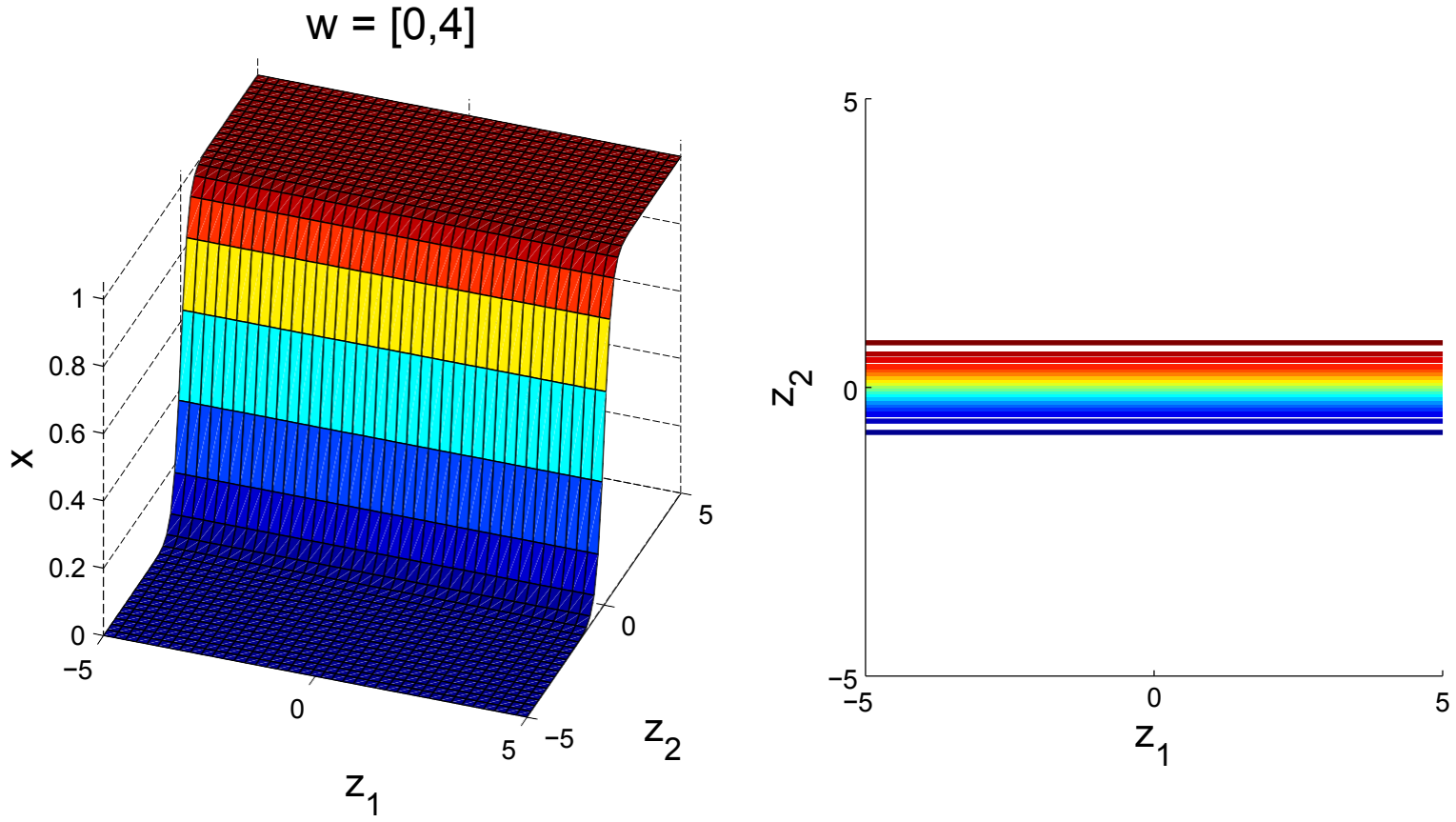
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

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



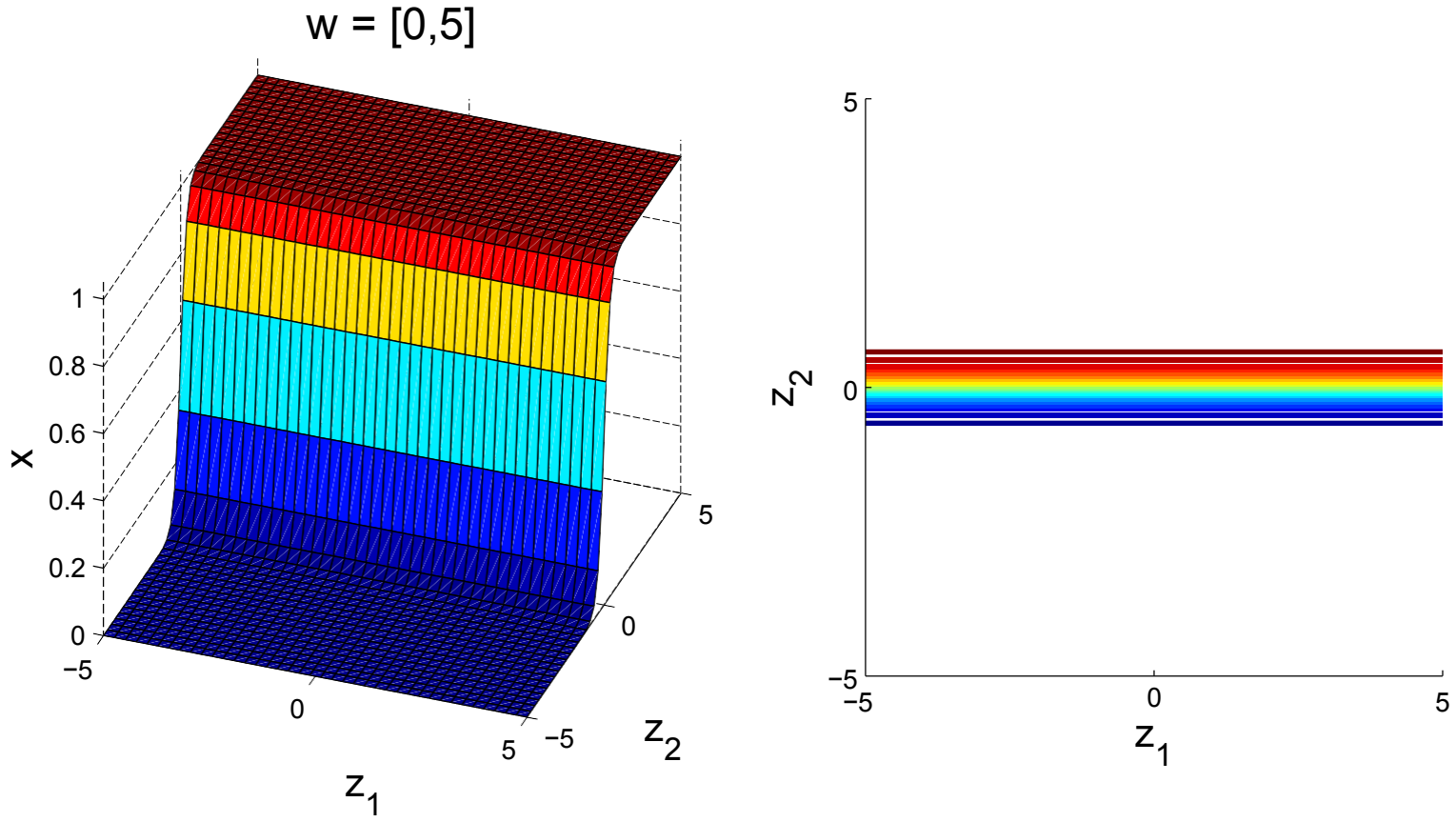
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

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



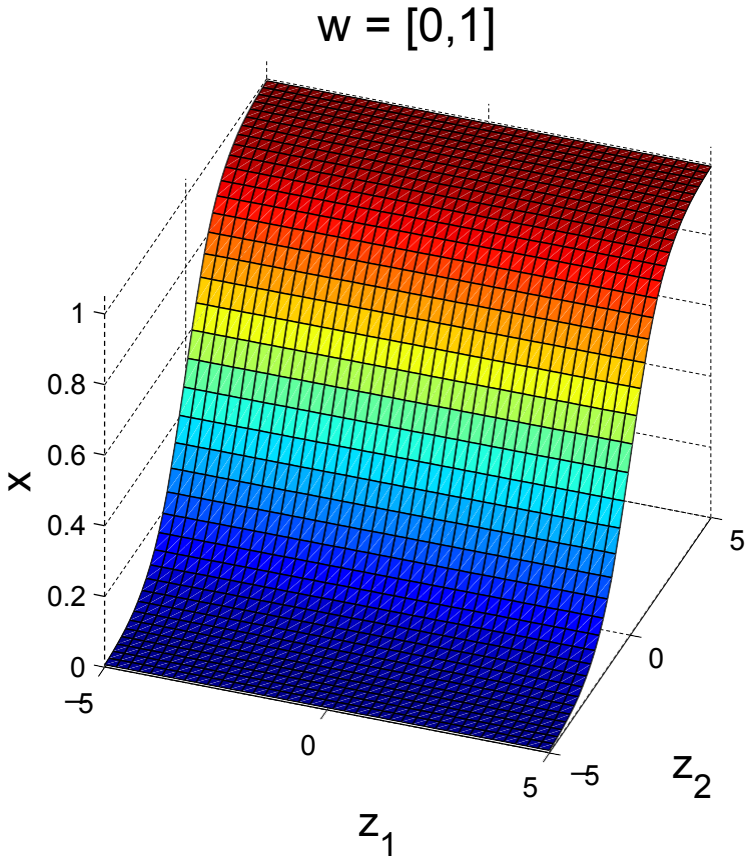
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

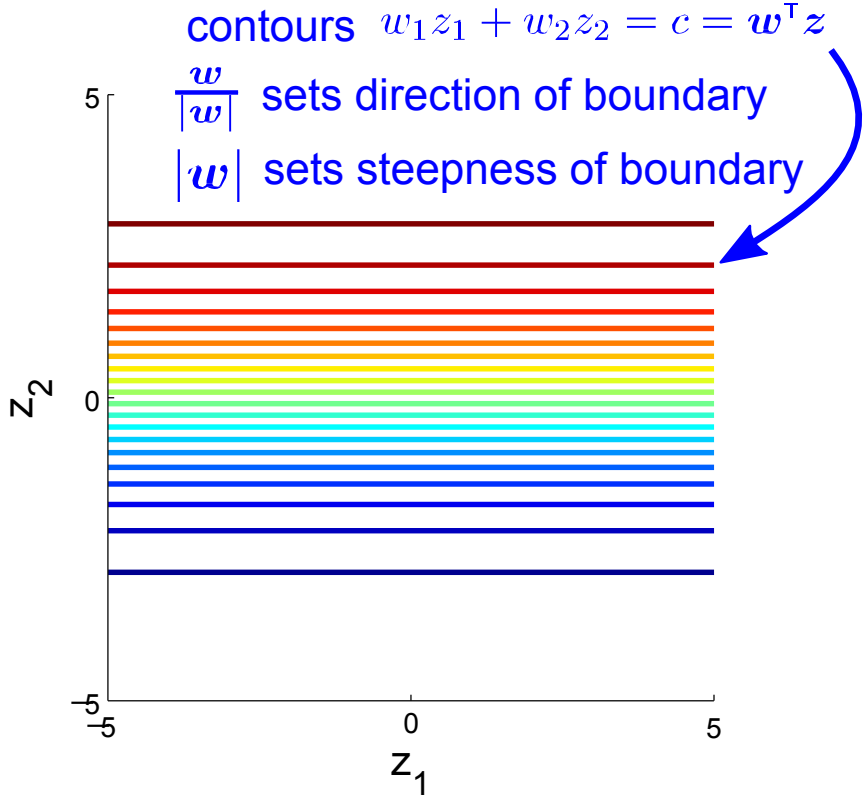
Input-Output Function of a Single Neuron (cont'd)



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

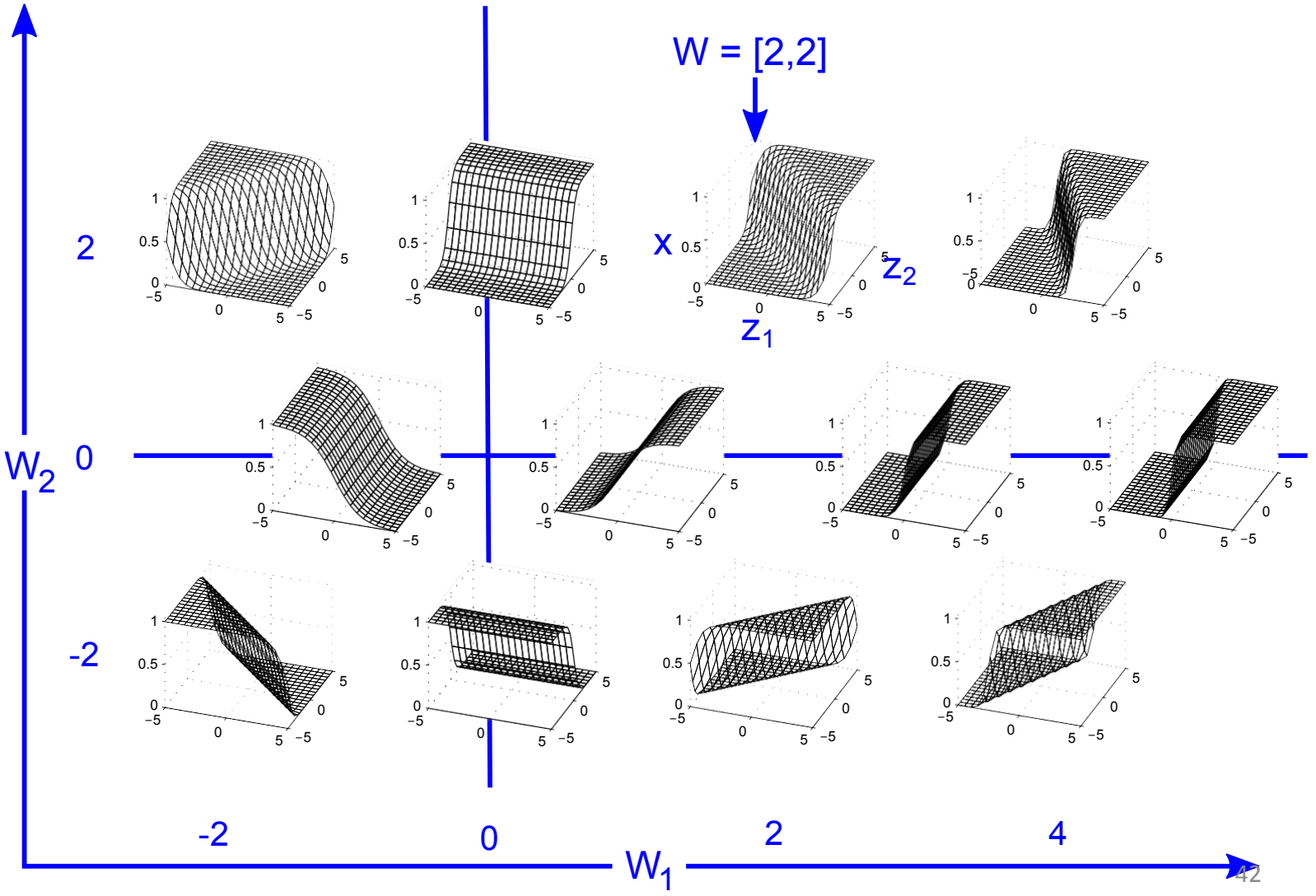
Input-Output Function of a Single Neuron (cont'd)



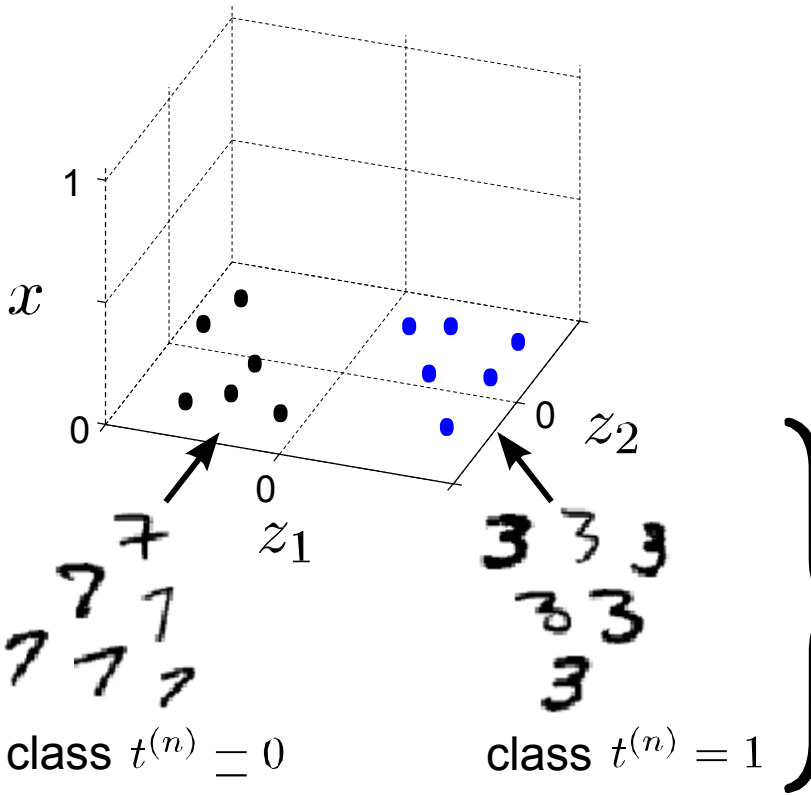


$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

Weight Space of a Single Neuron



Training a Single Neuron

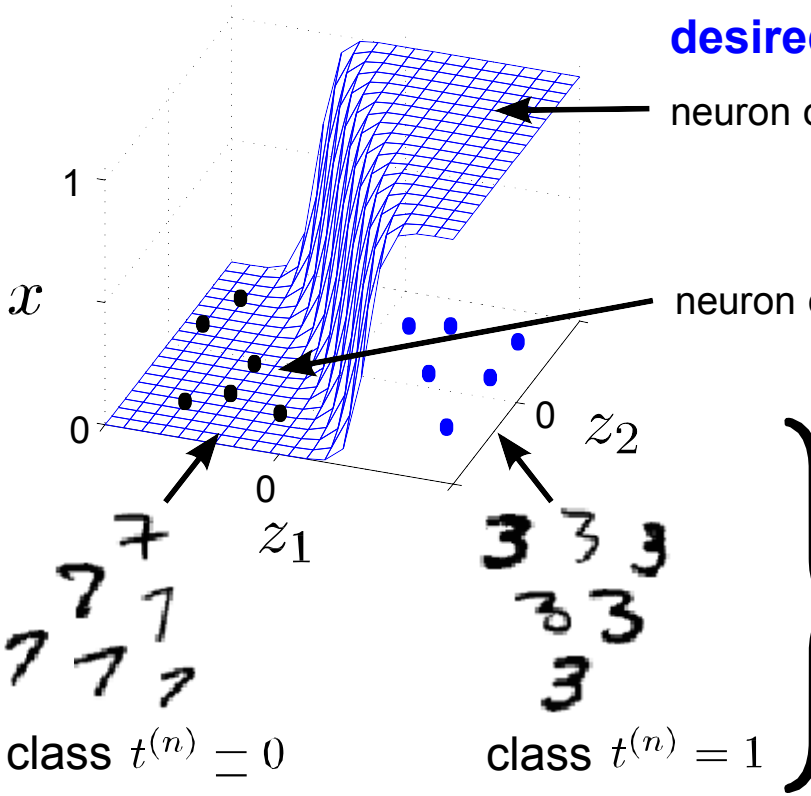


training data

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

inputs class labels

Training a Single Neuron



desired result of training:

neuron outputs $\mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w}) \approx 1$ for $t^{(n)} = 1$

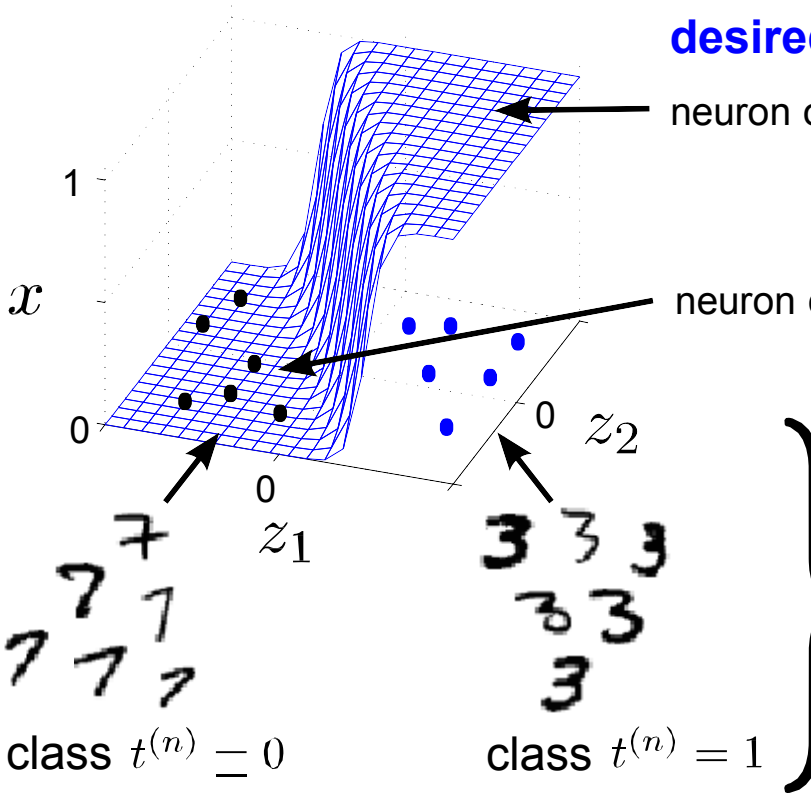
neuron outputs $\mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w}) \approx 0$ for $t^{(n)} = 0$

training data

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

inputs class labels

Training a Single Neuron



desired result of training:

neuron outputs $\mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w}) \approx 1$ for $t^{(n)} = 1$

neuron outputs $\mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w}) \approx 0$ for $t^{(n)} = 0$

training data

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

inputs class labels

objective function:

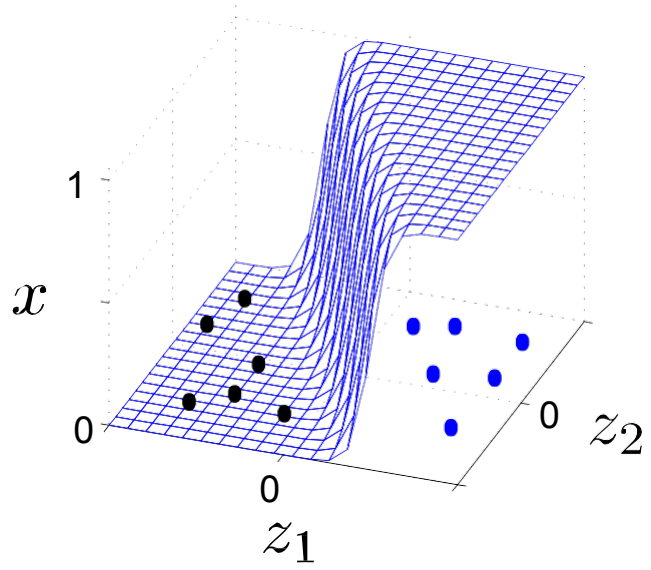
$$G(\boldsymbol{w}) = -\sum_n \left[t^{(n)} \log \mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w}) + (1 - t^n) \log \left(1 - \mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w})\right)\right] \geq 0$$

surprise $-\log p(\text{outcome})$ when observing $t^{(n)}$

relative entropy between $\mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w})$ and $t^{(n)}$

} encourages neuron output to match training data

Training a Single Neuron



training data

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

inputs class labels

objective function:

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

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} G(\boldsymbol{w})$$

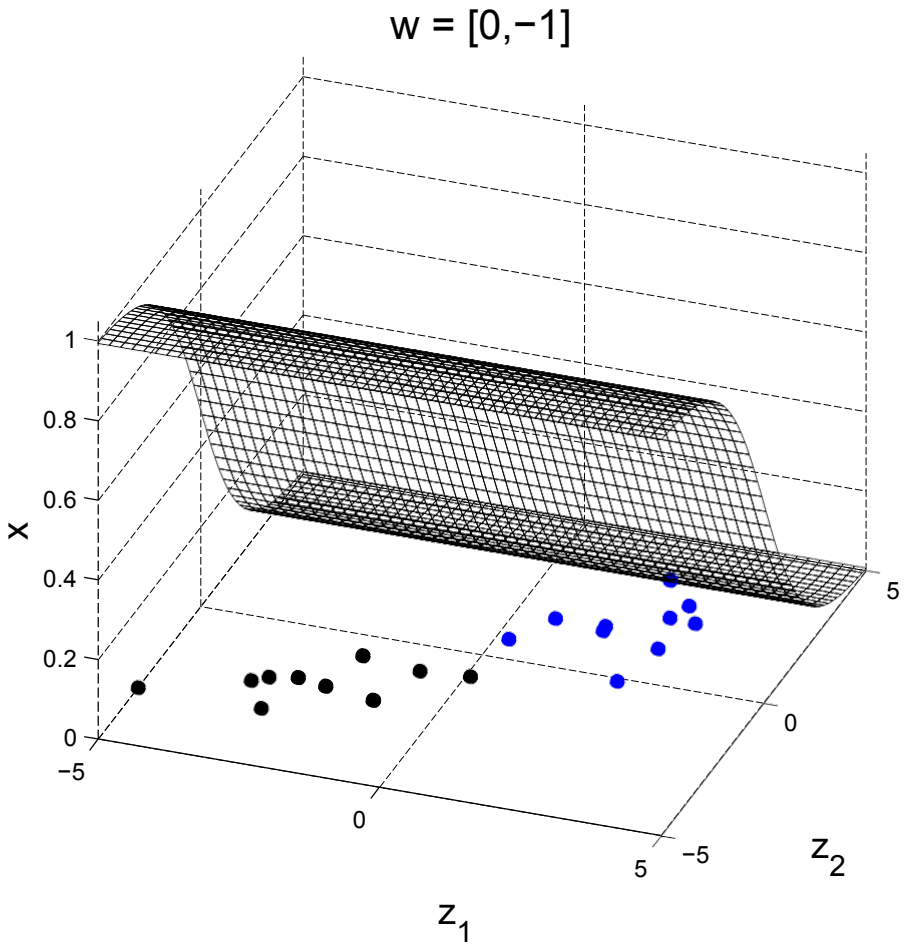
choose the weights that minimise the network's surprise about the training data

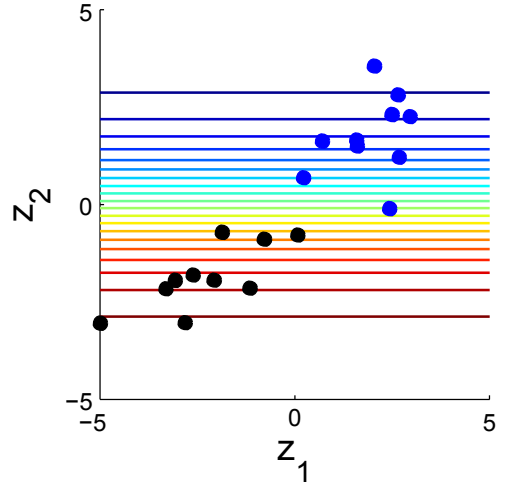
$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{w}} G(\boldsymbol{w}) = \sum_n \frac{\mathrm{d}G(\boldsymbol{w})}{\mathrm{d}x^{(n)}} \frac{\mathrm{d}x^{(n)}}{\mathrm{d}\boldsymbol{w}} = -\sum_n (t^{(n)} - x^{(n)}) \boldsymbol{z}^{(n)}$$ = prediction error x feature

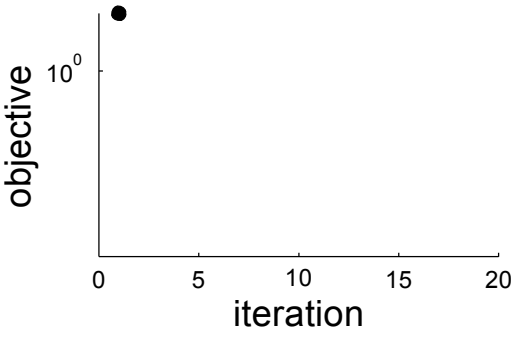
$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{w}} G(\boldsymbol{w})$$

iteratively step down the objective (gradient points up hill)

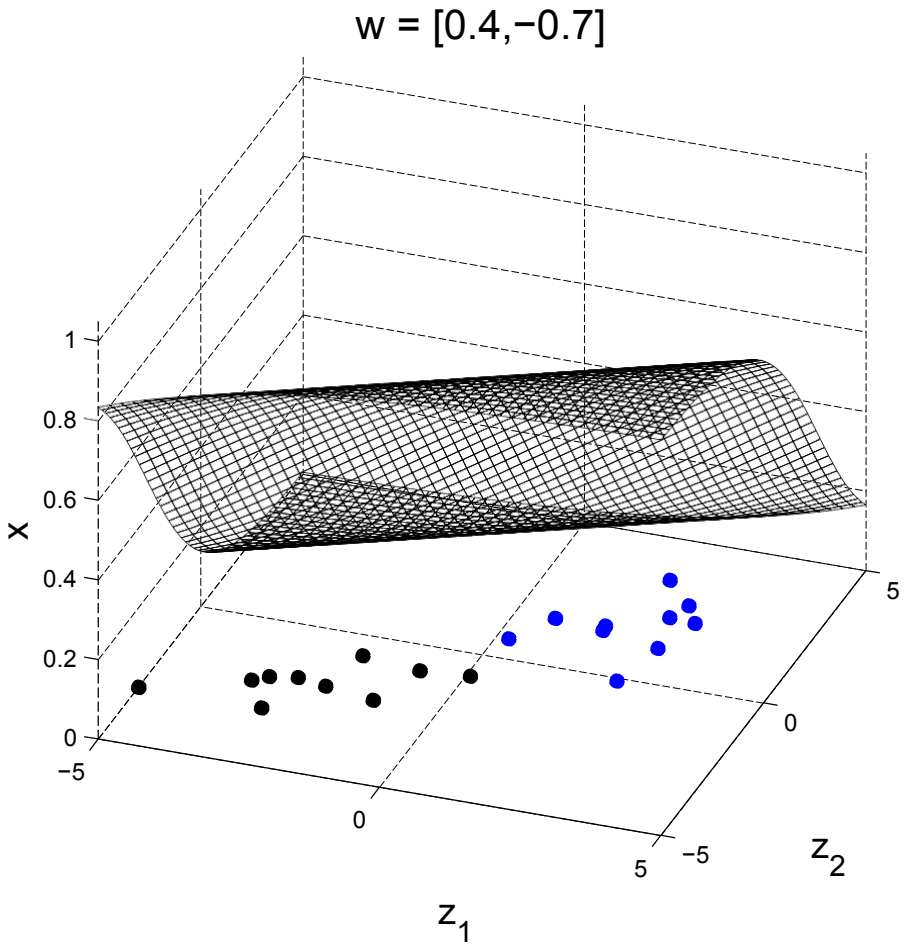
Training a Single Neuron

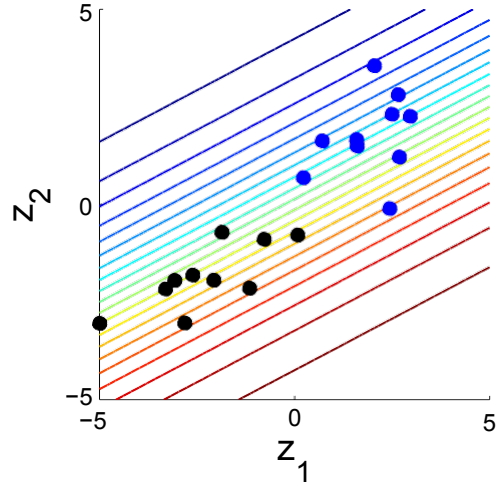


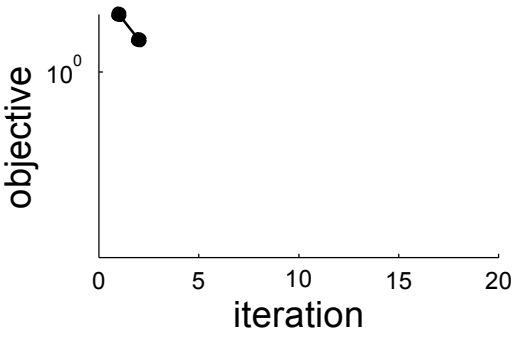




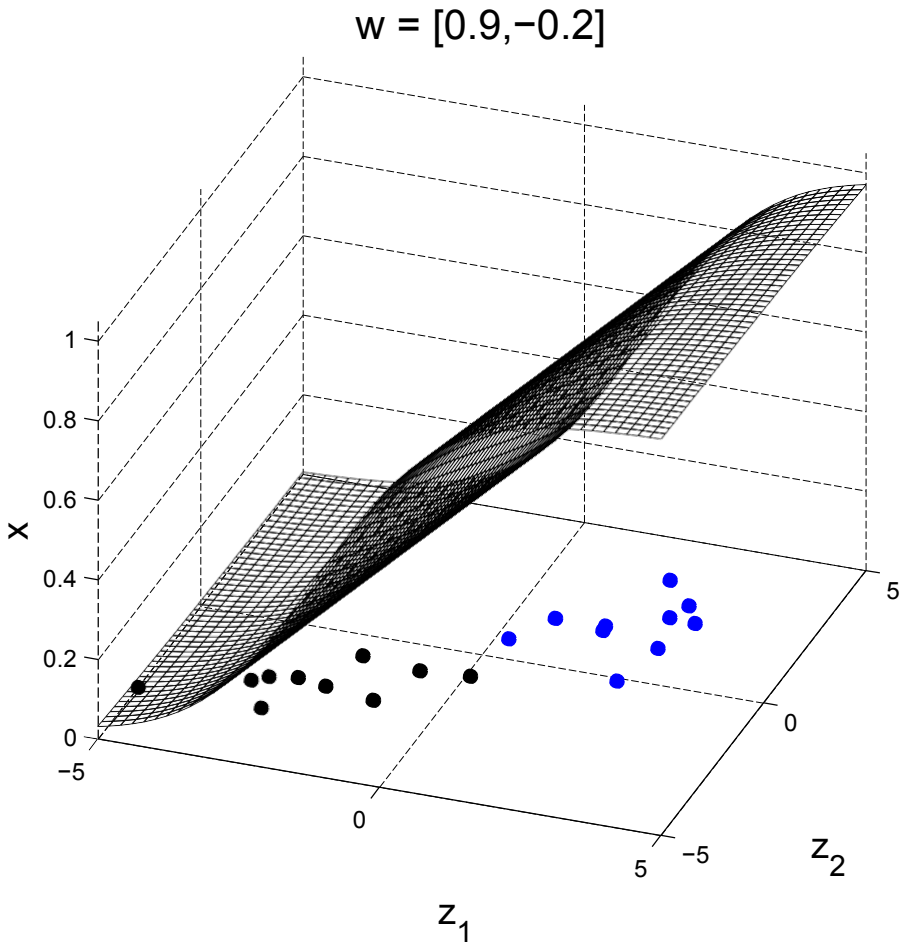
Training a Single Neuron

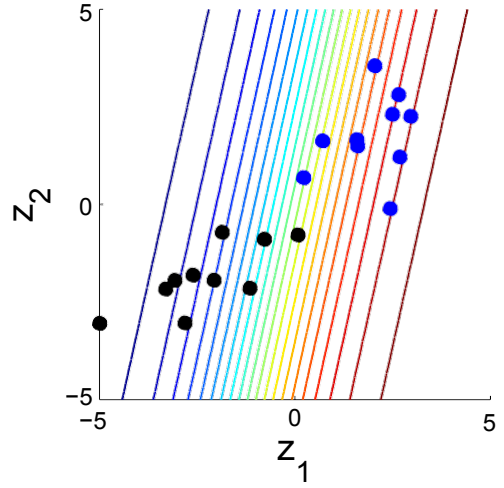


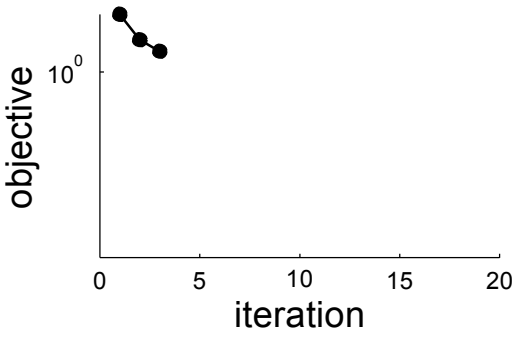




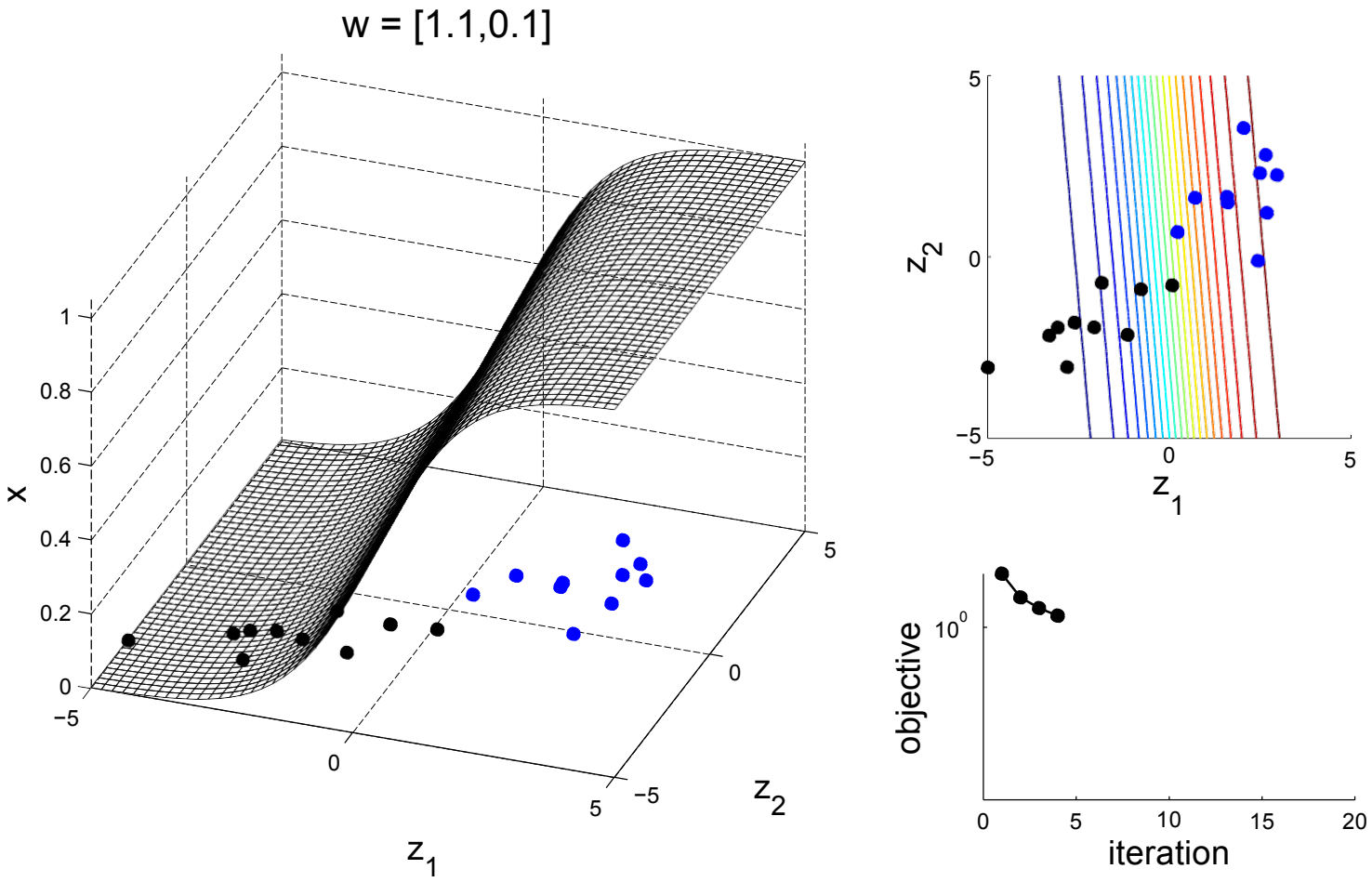
Training a Single Neuron



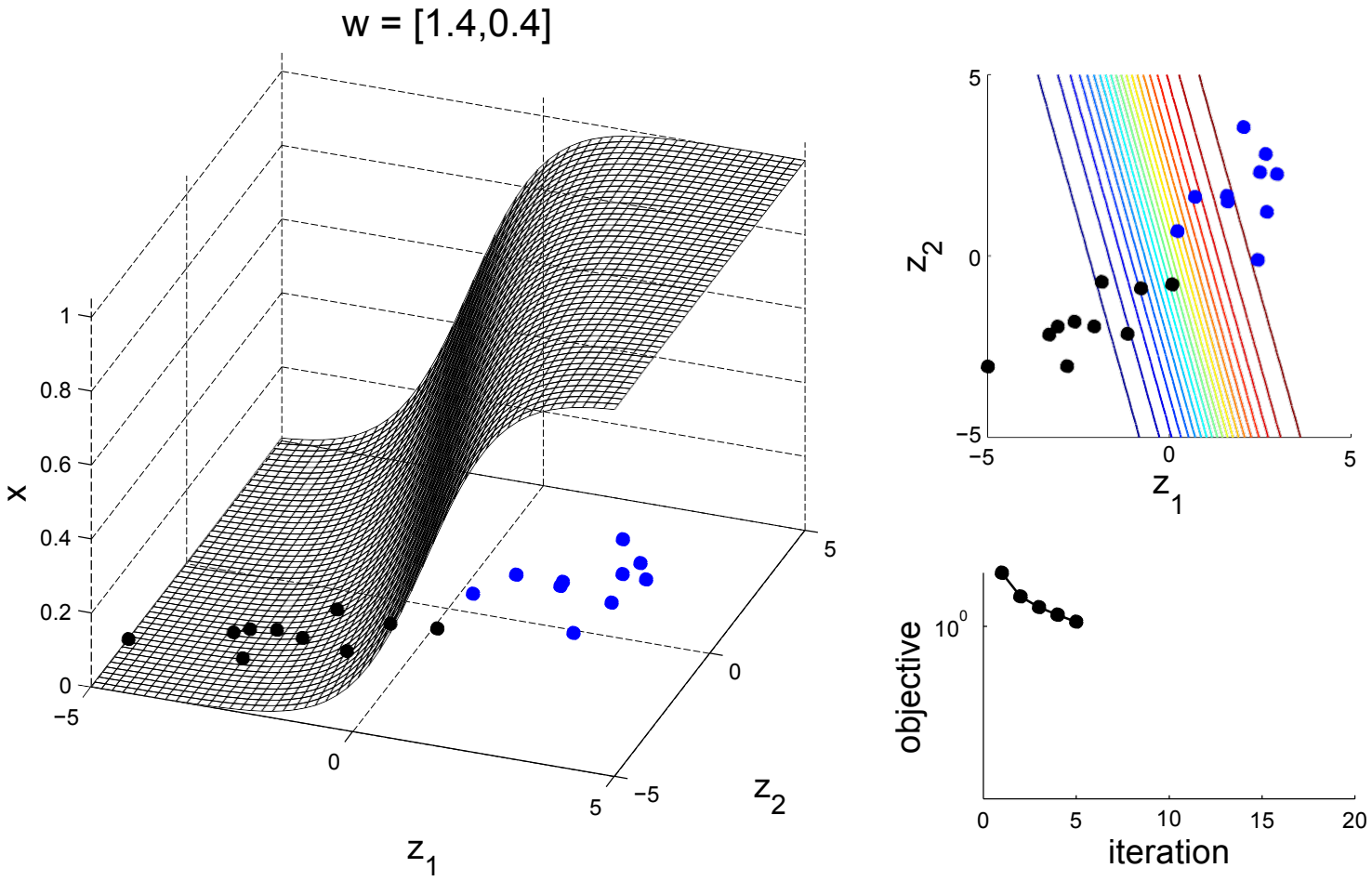




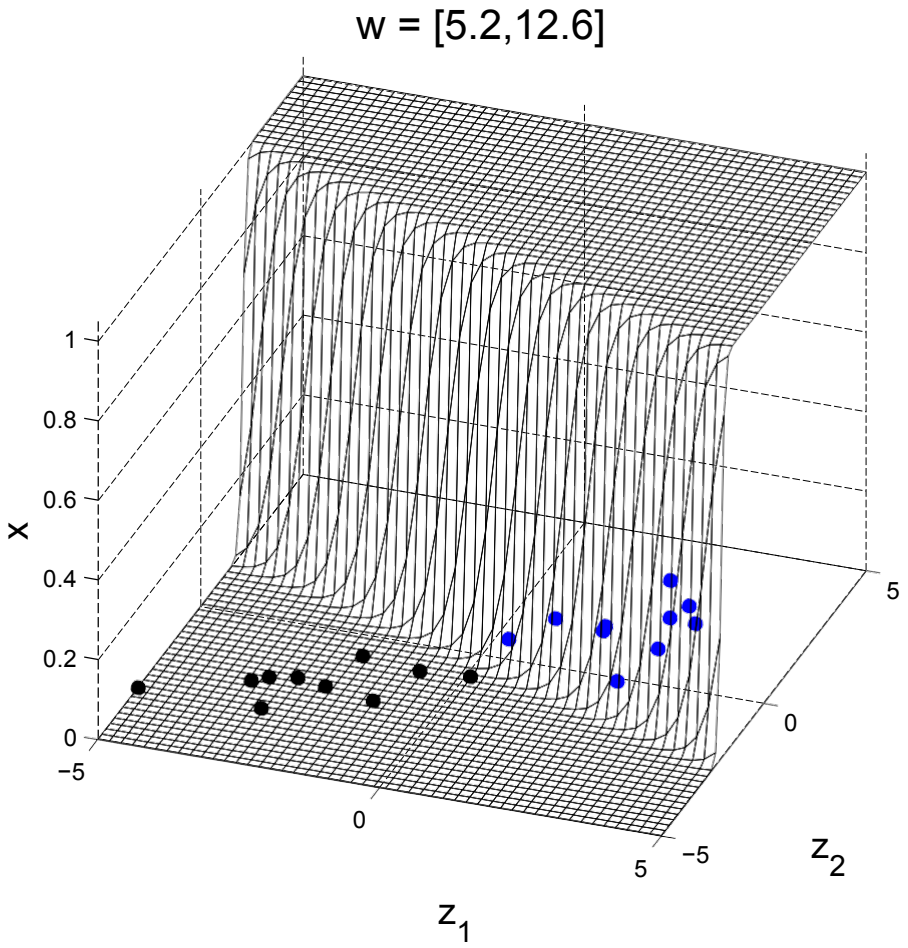
Training a Single Neuron

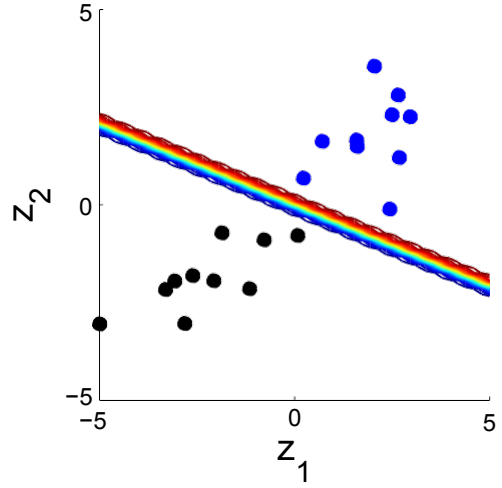


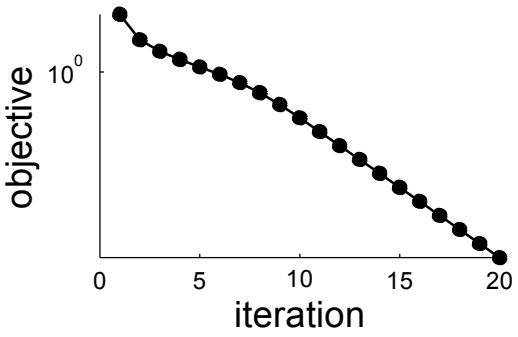
Training a Single Neuron



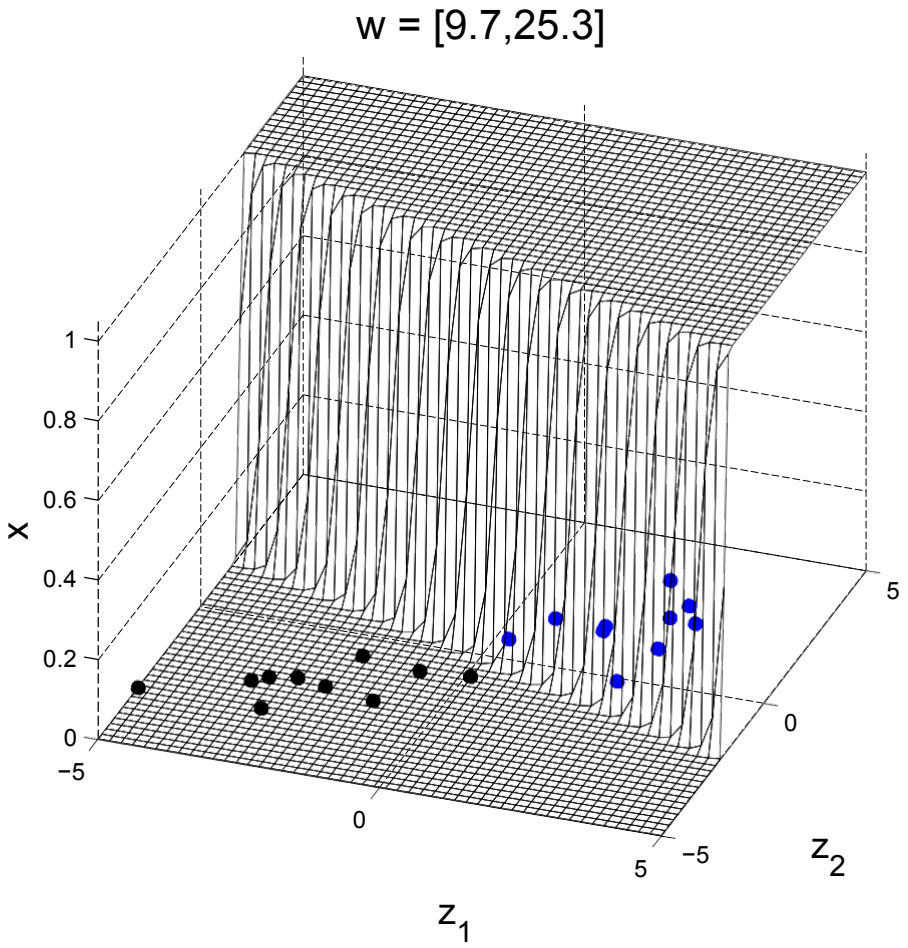
Training a Single Neuron

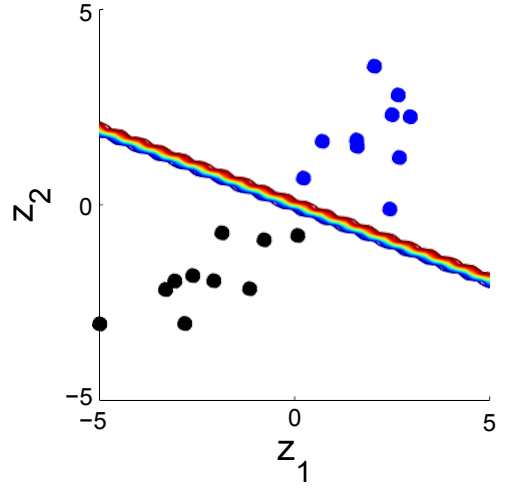


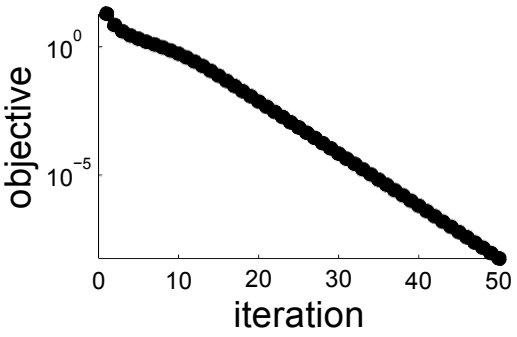




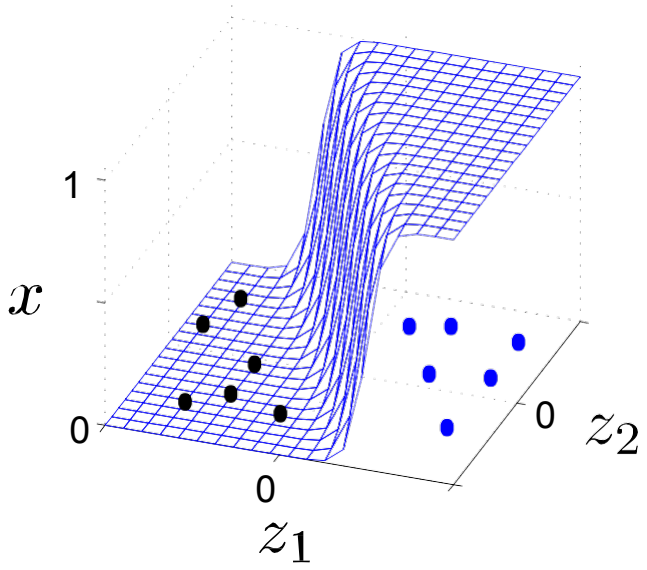
Training a Single Neuron







Overfitting and Weight Decay



training data

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

inputs class labels

objective function:

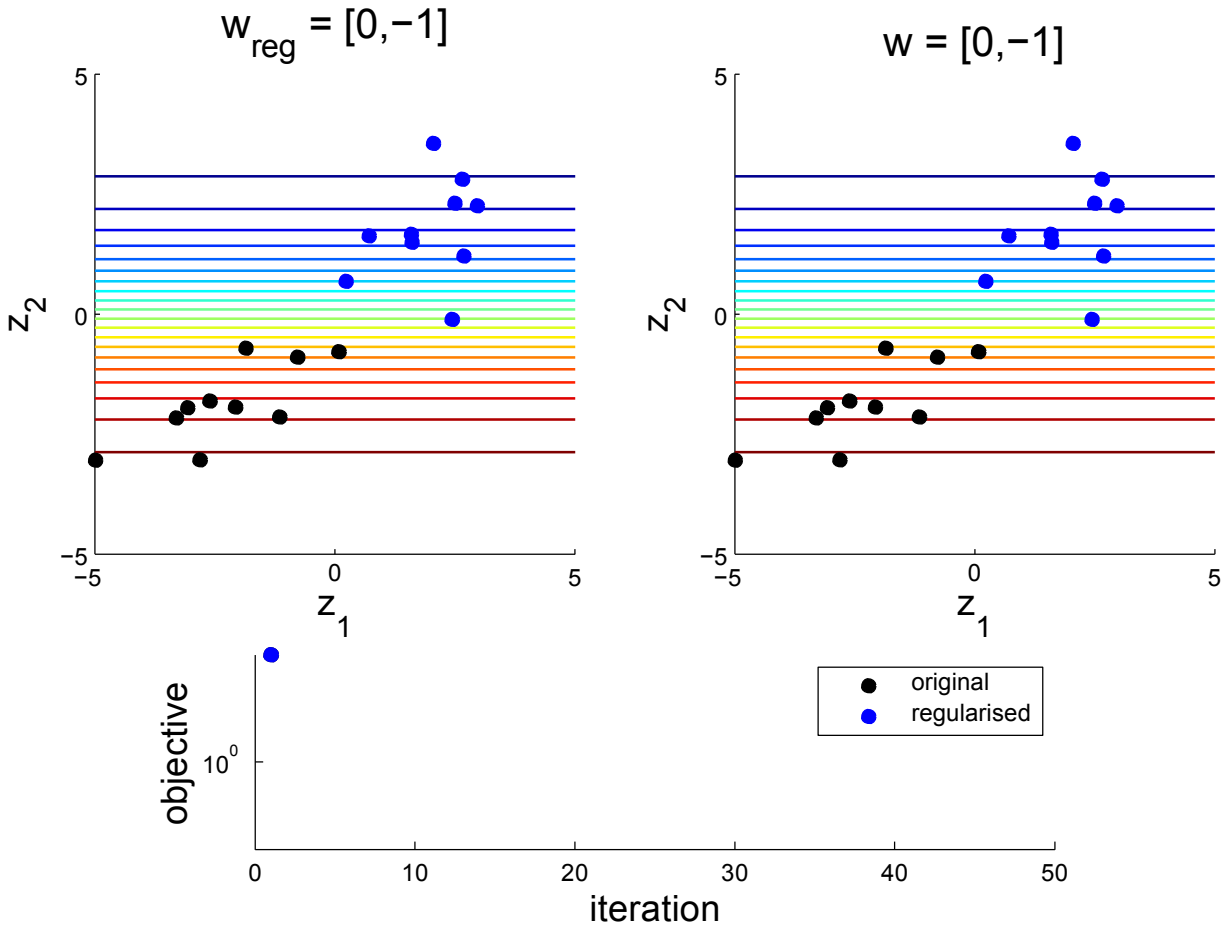
$$G(\boldsymbol{w}) = -\sum_n \left[t^{(n)} \log \mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w}) + (1 - t^n) \log \left(1 - \mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w})\right)\right]$$

$$E(\boldsymbol{w}) = \tfrac{1}{2} \sum_i w_i^2$$ regulariser discourages the network using extreme weights

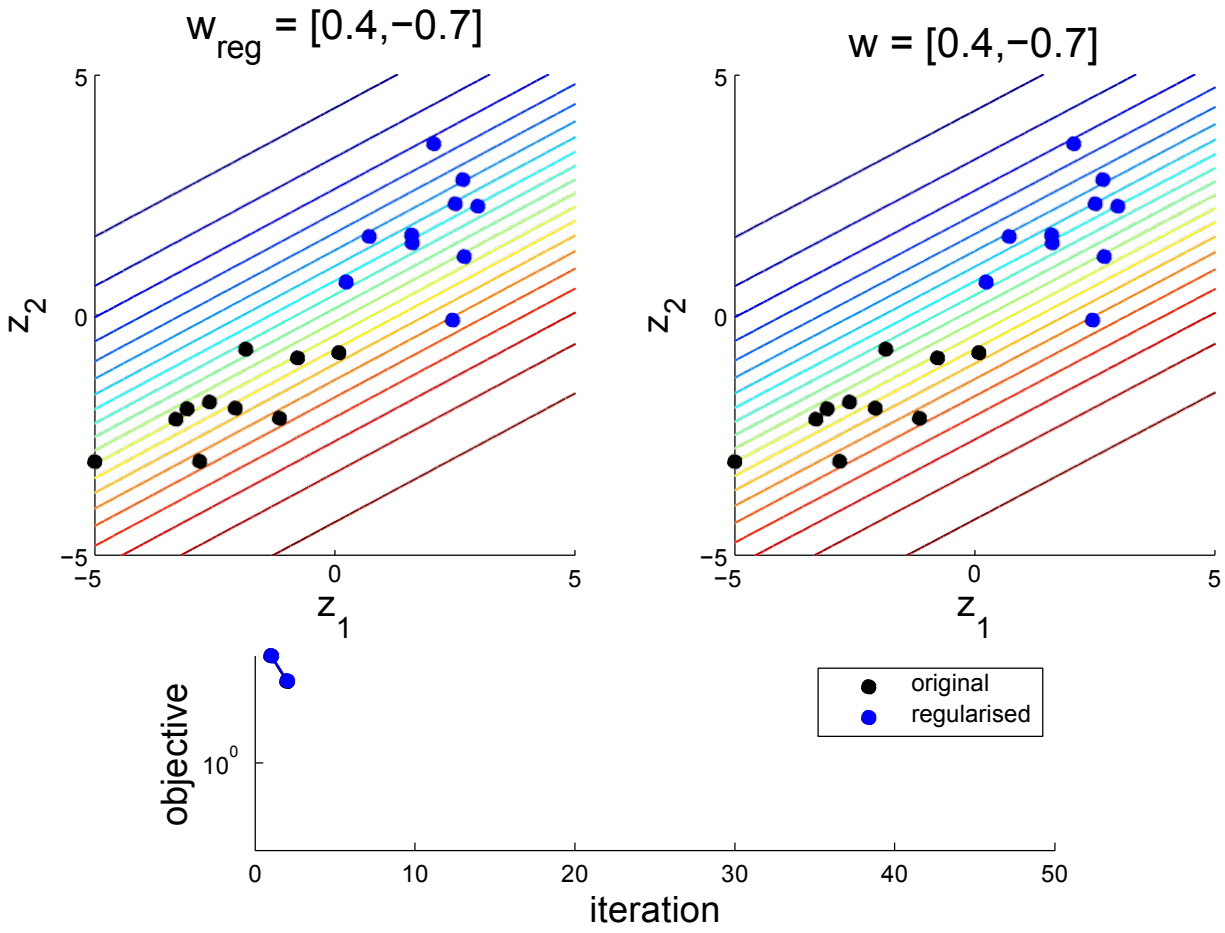
$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} M(\boldsymbol{w}) = \arg\min_{\boldsymbol{w}} \left[G(\boldsymbol{w}) + \alpha E(\boldsymbol{w})\right]$$

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

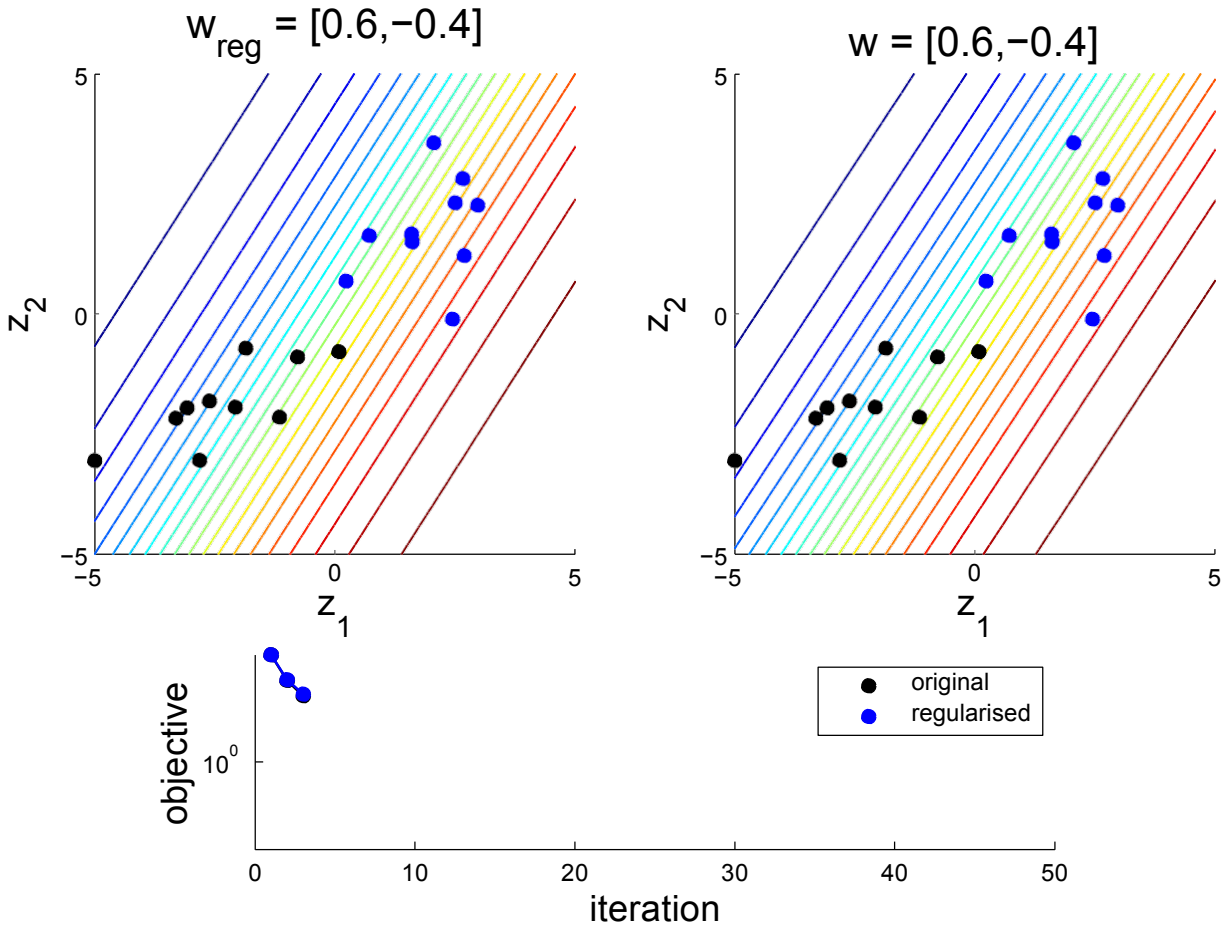
Training a Single Neuron (cont'd)



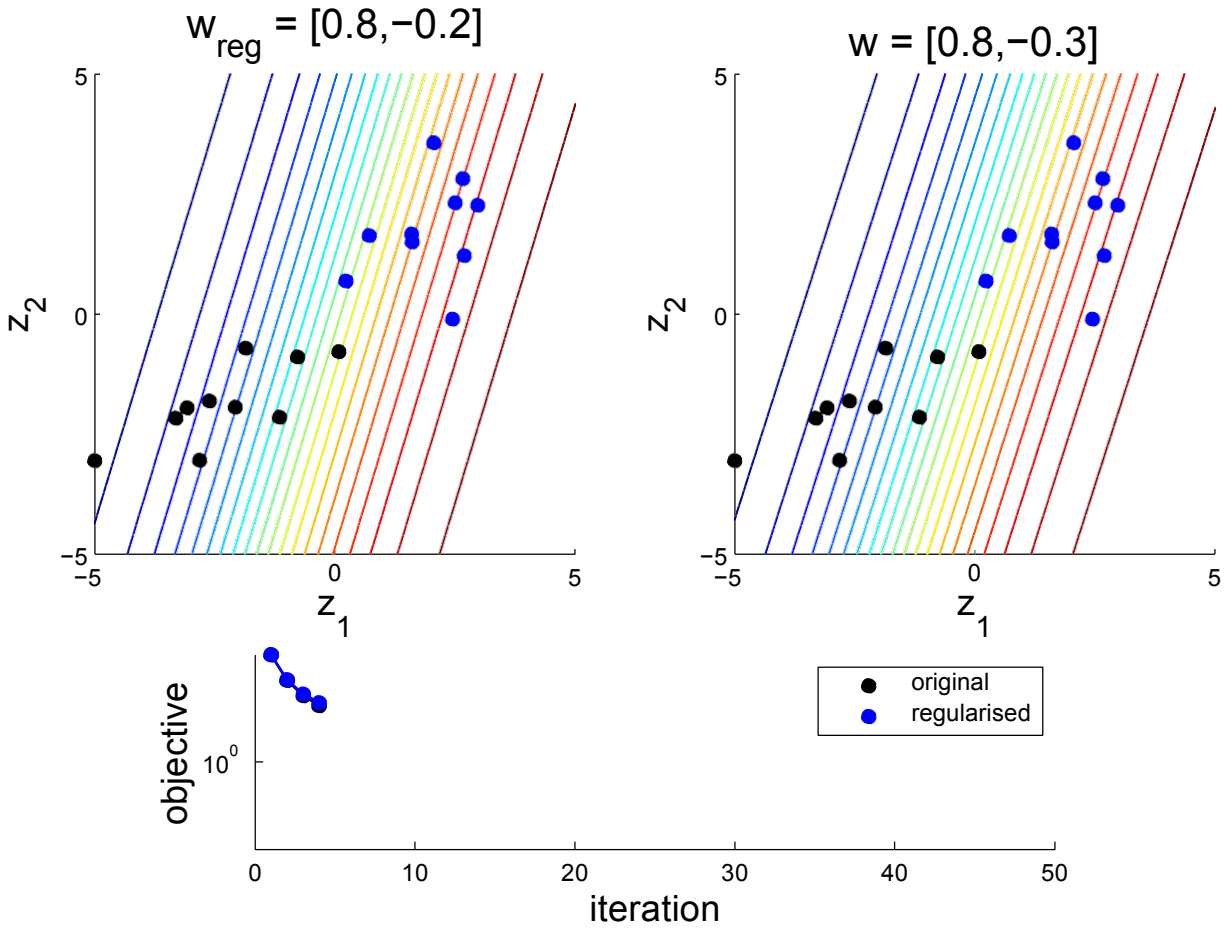
Training a Single Neuron (cont'd)



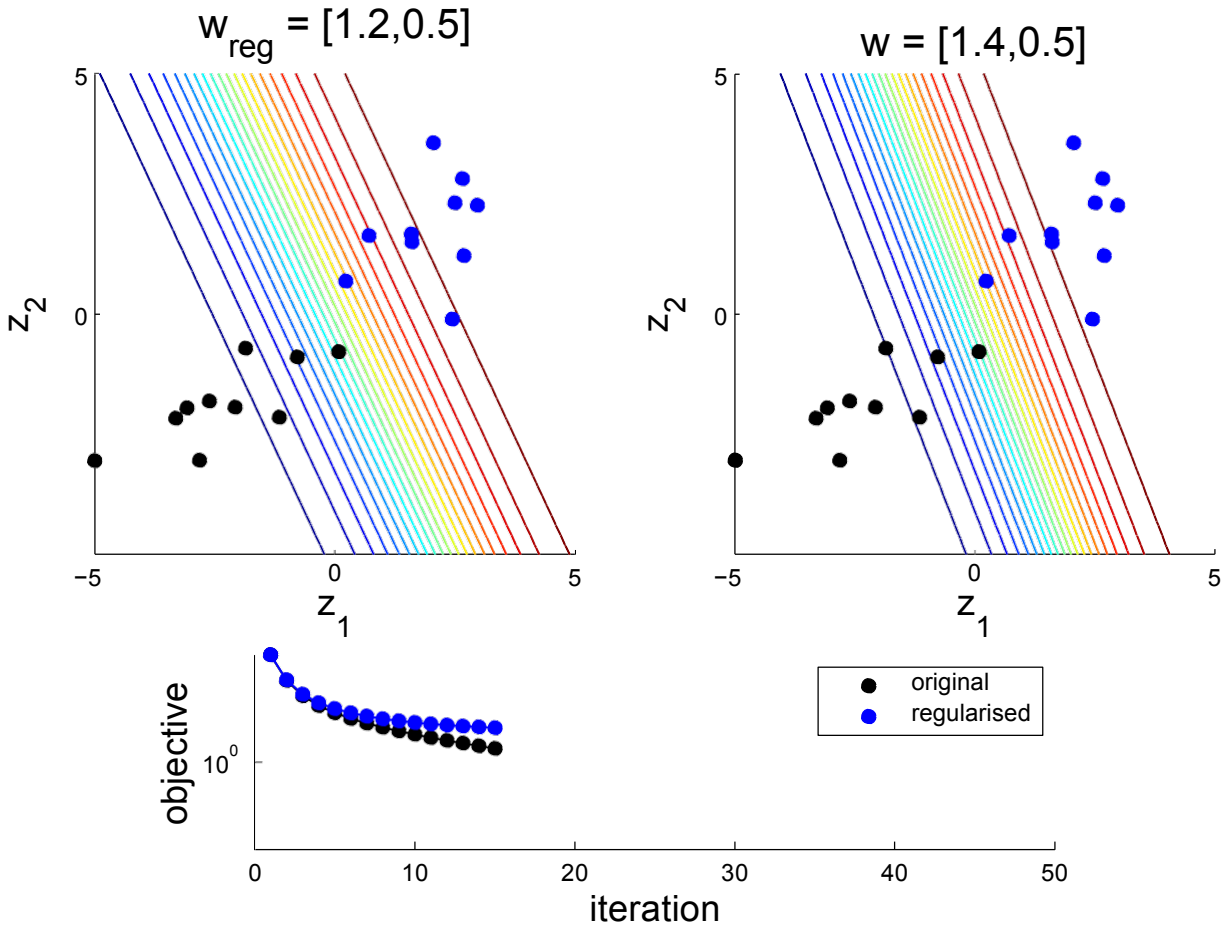
Training a Single Neuron (cont'd)



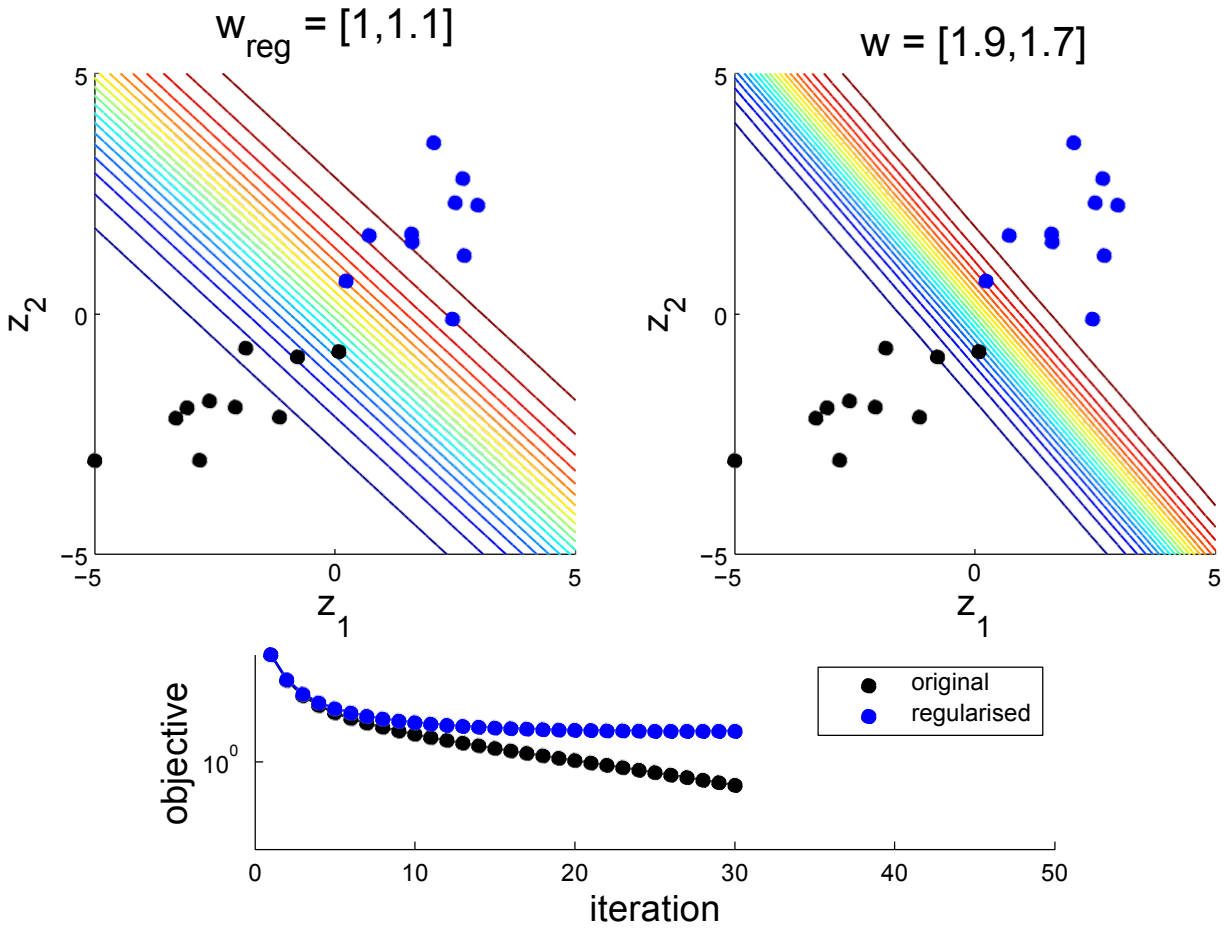
Training a Single Neuron (cont'd)



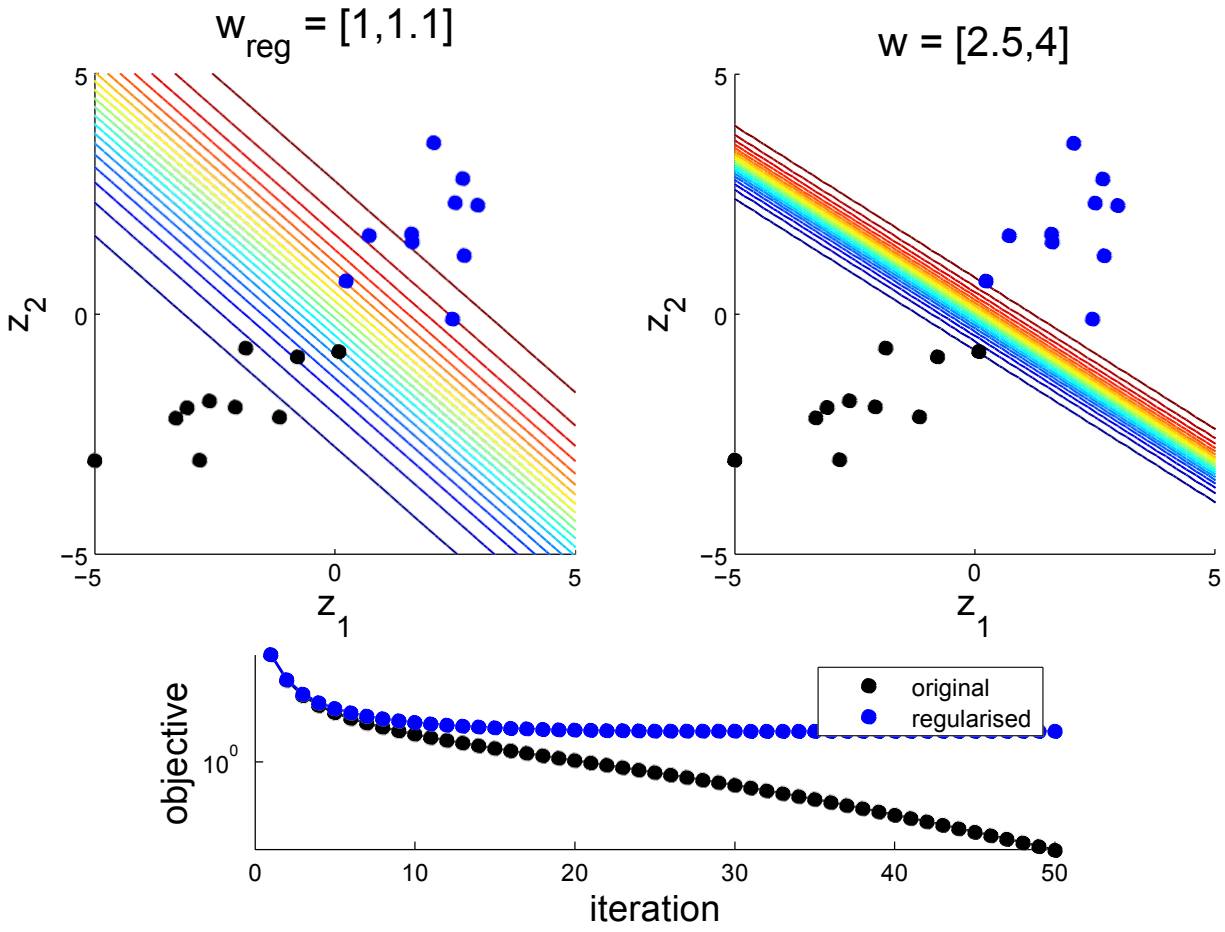
Training a Single Neuron (cont'd)



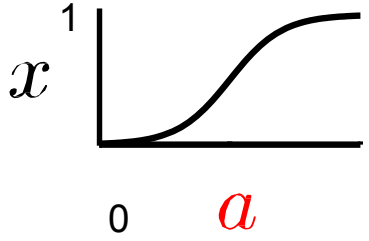
Training a Single Neuron (cont'd)

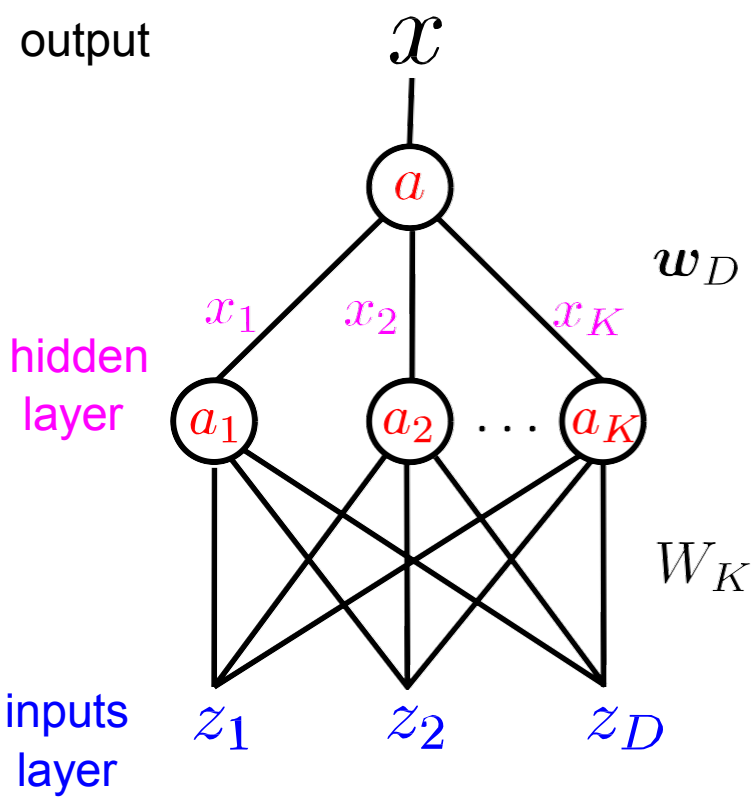


Training a Single Neuron (cont'd)



Single Hidden Layer Neural Networks





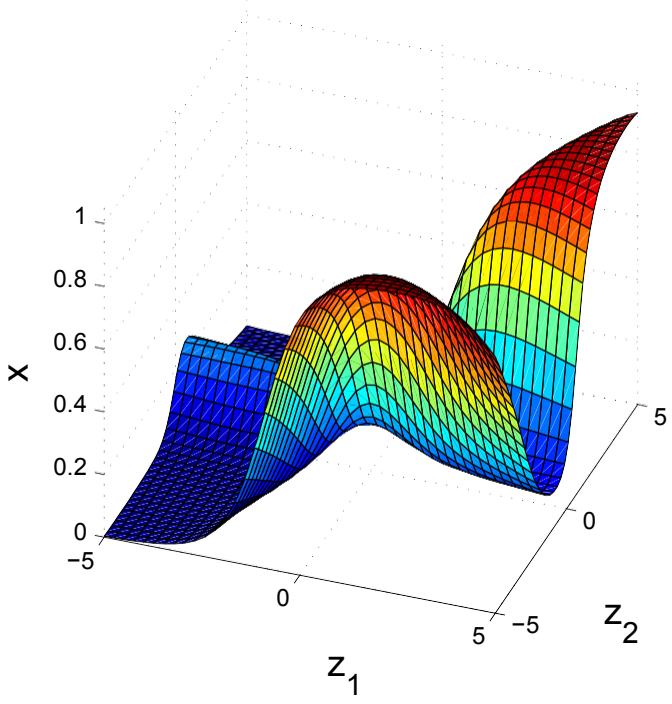
$$x(a) = \frac{1}{1+\exp(-a)}$$

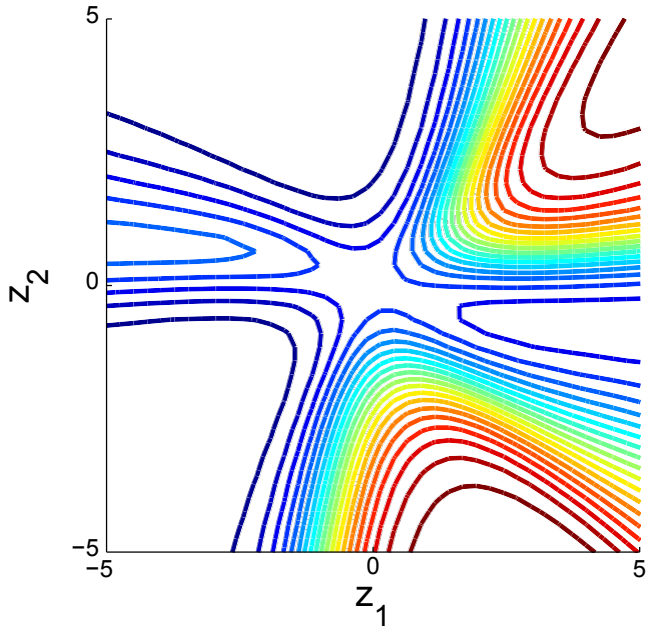
$$a = \sum_{k=1}^{K} w_k x_k$$

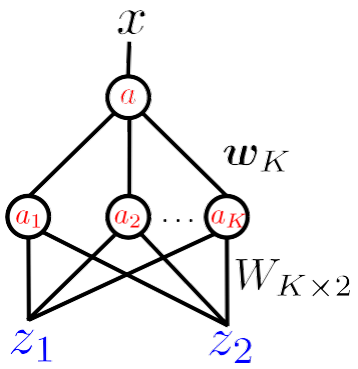
$$x(a_k) = \frac{1}{1+\exp(-a_k)}$$

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

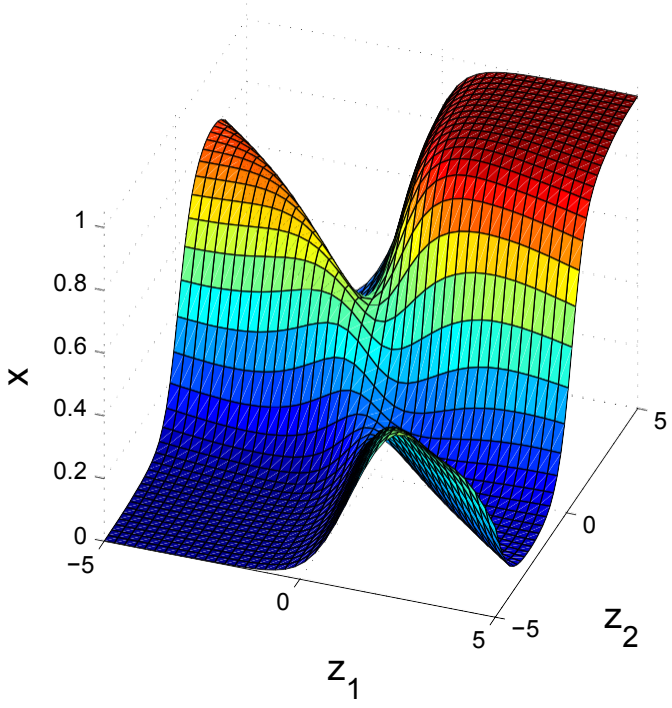
Sampling Random Neural Network Classifiers

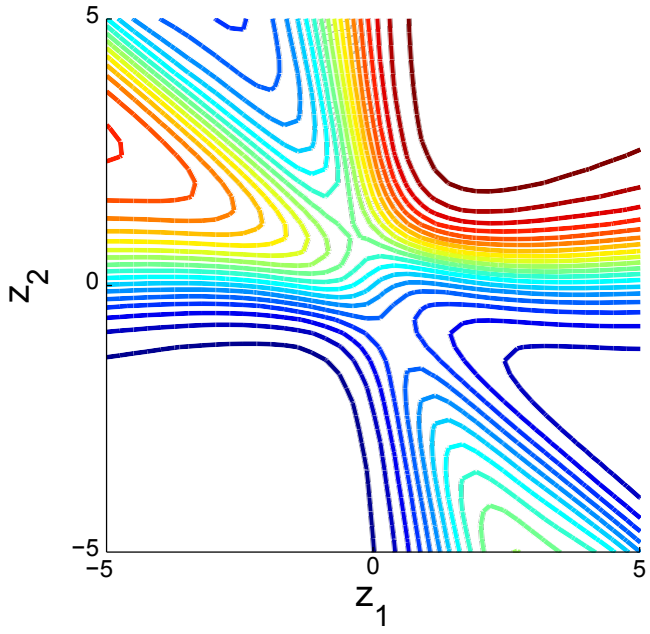


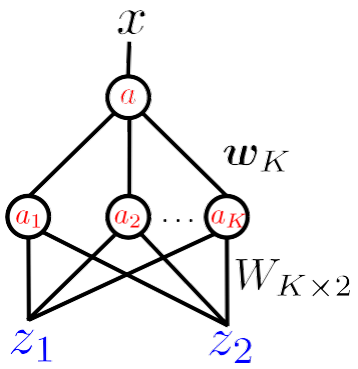




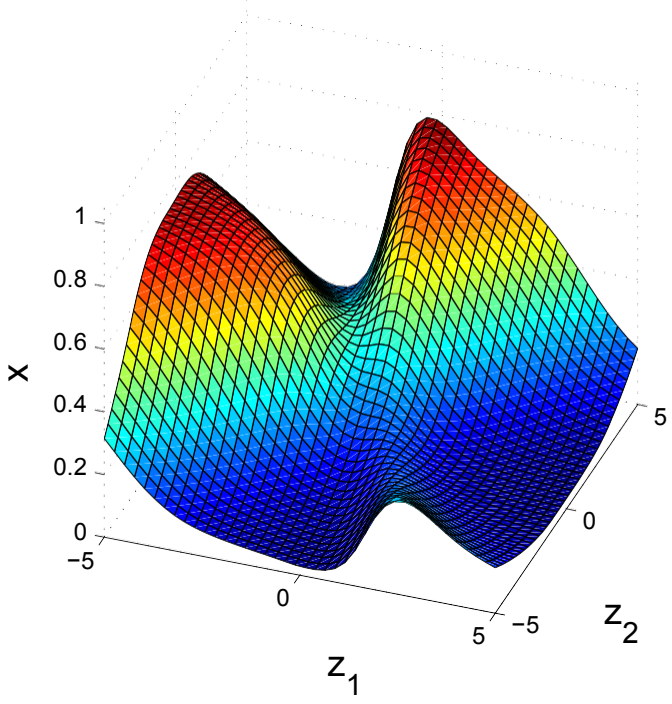
Sampling Random Neural Network Classifiers

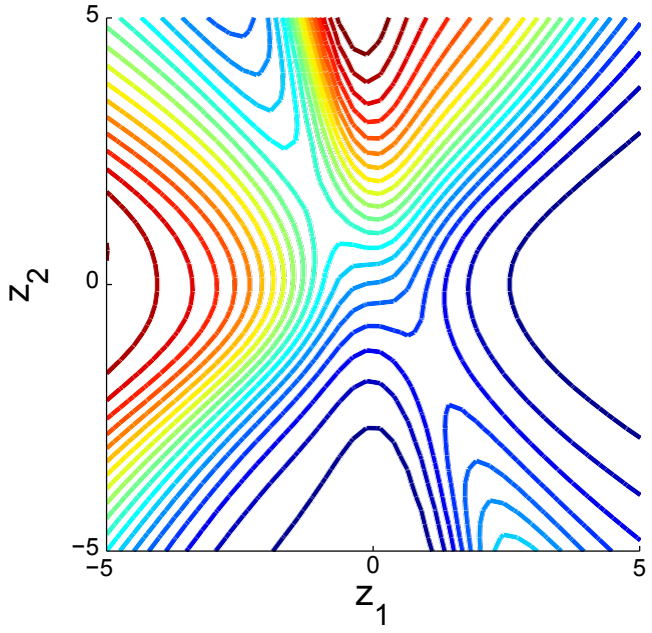


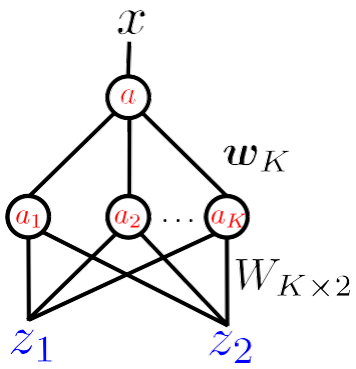




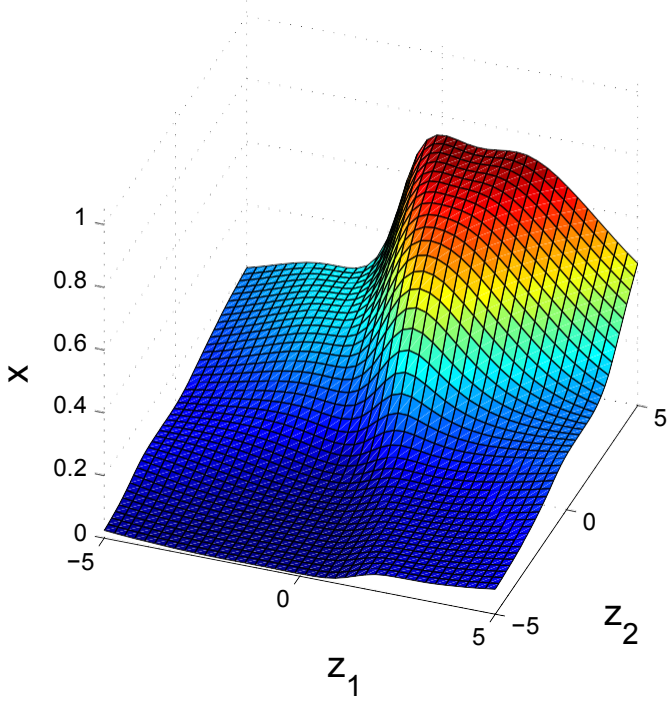
Sampling Random Neural Network Classifiers

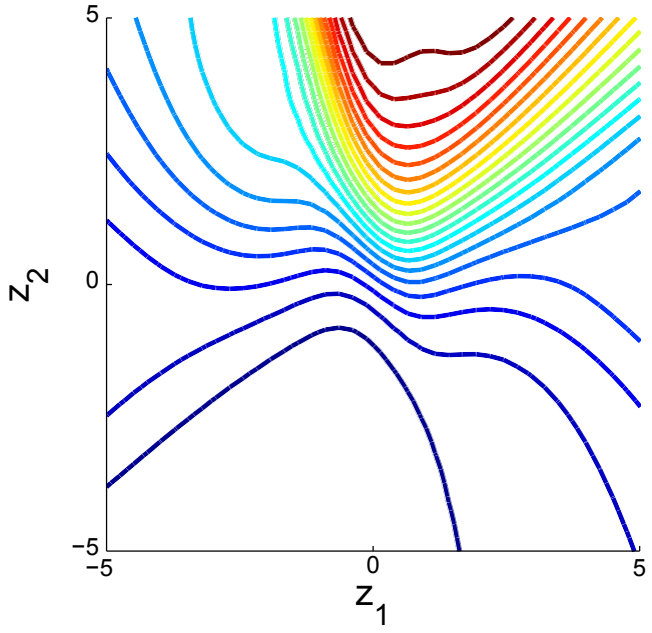


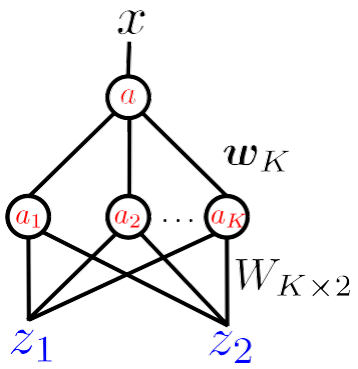




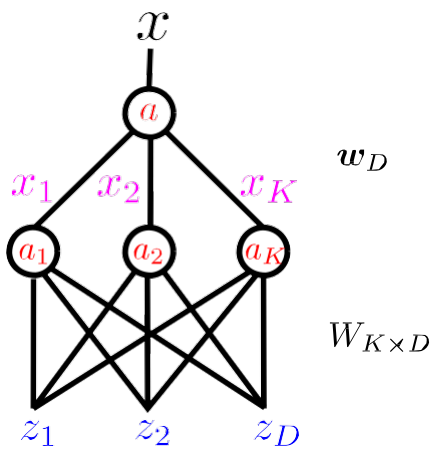
Sampling Random Neural Network Classifiers







Training a Neural Network with a Single Hidden Layer



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

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

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

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

objective function:

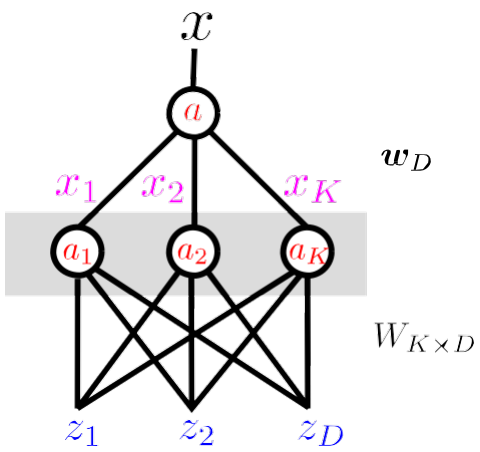
$$G(W, \boldsymbol{w}) = -\sum_n \left[t^{(n)} \log \mathrm{x}^{(n)} + (1 - t^n) \log \left(1 - \mathrm{x}^{(n)}\right)\right]$$ likelihood same as before

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

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

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.



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

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

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

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

objective function:

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

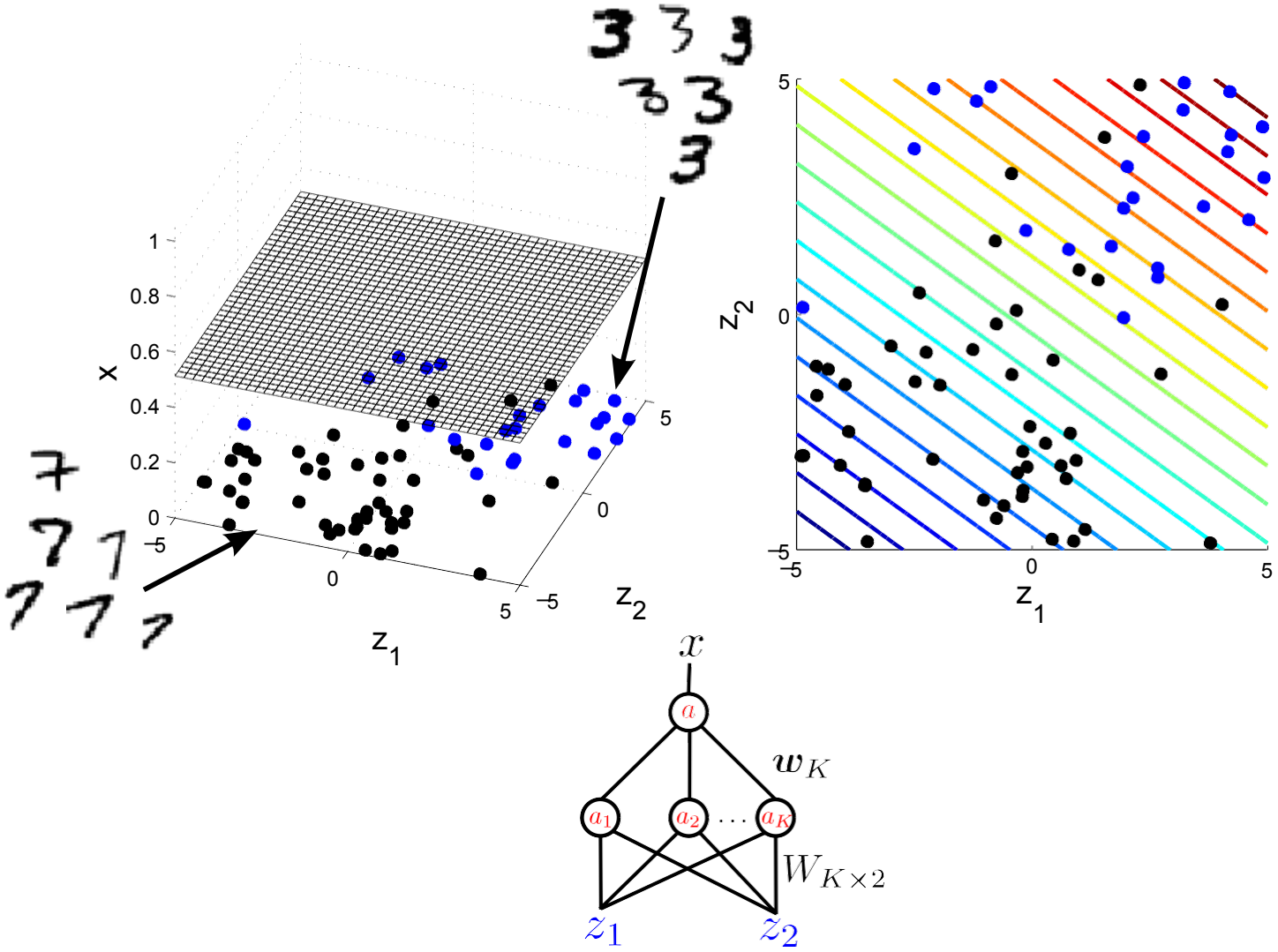
$$E(W, \boldsymbol{w}) = \tfrac{1}{2} \sum_i w_i^2 + \tfrac{1}{2} \sum_{ij} W_{ij}^2$$ regulariser discourages extreme weights

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

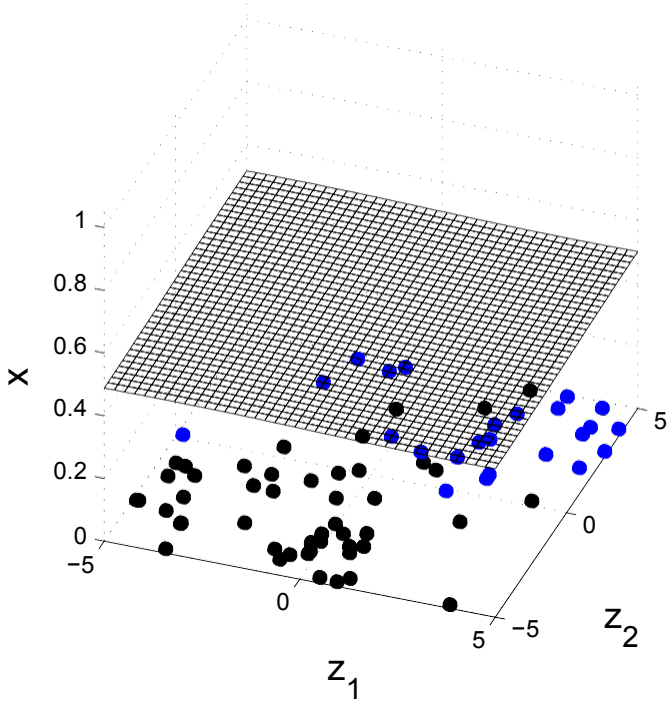
$$\frac{\mathrm{d}G(W, \boldsymbol{w})}{\mathrm{d}W_{ij}} = \sum_n \frac{\mathrm{d}G(W, \boldsymbol{w})}{\mathrm{d}x^{(n)}} \frac{\mathrm{d}x^{(n)}}{\mathrm{d}W_{ij}} = \sum_n \frac{\mathrm{d}G(W, \boldsymbol{w})}{\mathrm{d}x^{(n)}} \frac{\mathrm{d}a^{(n)}}{\mathrm{d}W_{ij}}$$

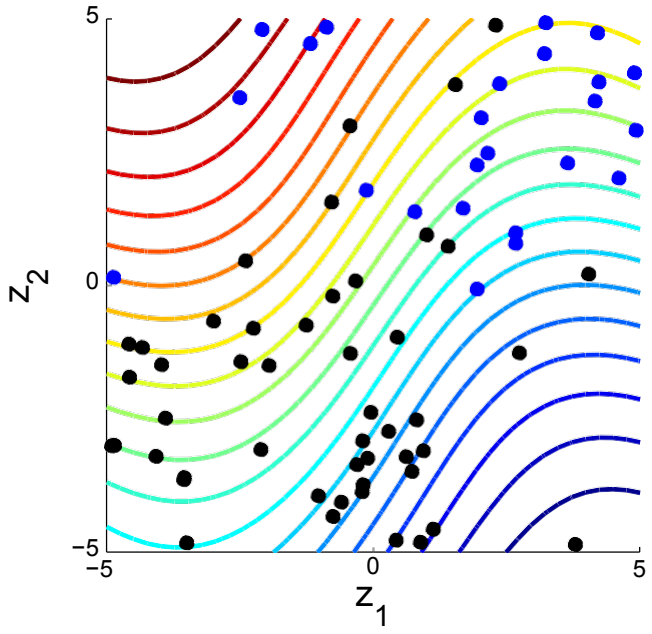
$$= \sum_n \frac{\mathrm{d}G(W, \boldsymbol{w})}{\mathrm{d}x^{(n)}} \frac{\mathrm{d}x^{(n)}}{\mathrm{d}a^{(n)}} \frac{\mathrm{d}a^{(n)}}{\mathrm{d}x_i^{(n)}} \frac{\mathrm{d}x_i^{(n)}}{\mathrm{d}W_{ij}} = \sum_n \frac{\mathrm{d}G(W, \boldsymbol{w})}{\mathrm{d}x^{(n)}} \frac{\mathrm{d}x^{(n)}}{\mathrm{d}a^{(n)}} \frac{\mathrm{d}a^{(n)}}{\mathrm{d}x_i^{(n)}} \frac{\mathrm{d}a_i^{(n)}}{\mathrm{d}W_{ij}}$$

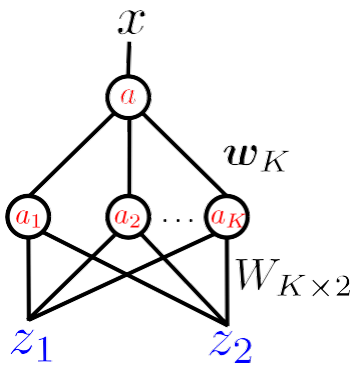
Training a Neural Network with a Single Hidden Layer



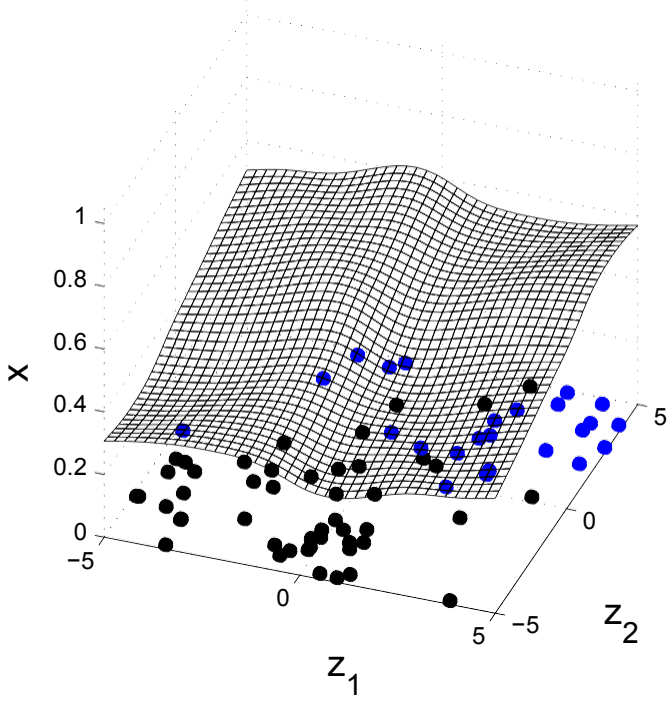
Training a Neural Network with a Single Hidden Layer

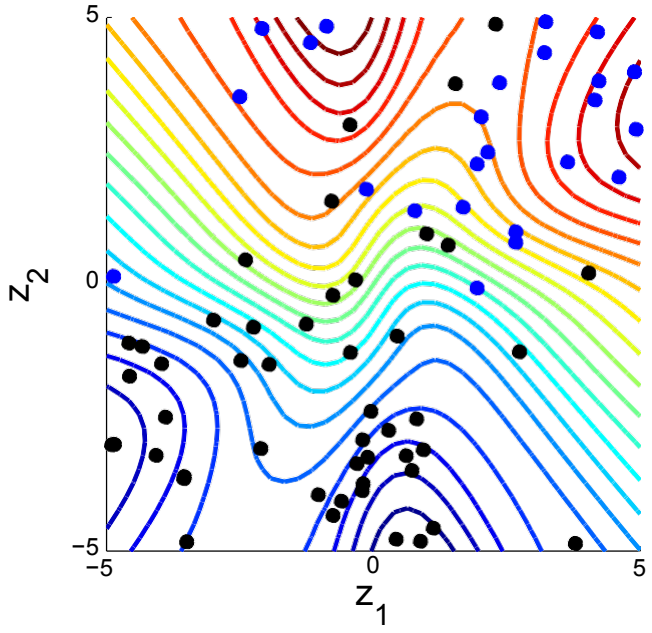


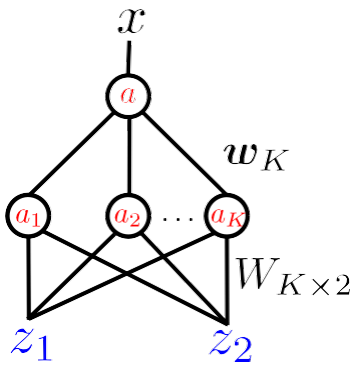




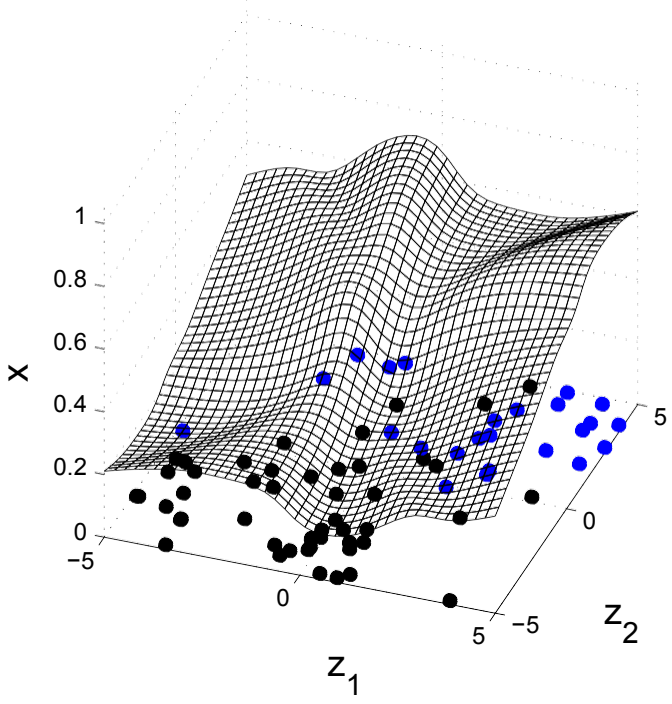
Training a Neural Network with a Single Hidden Layer

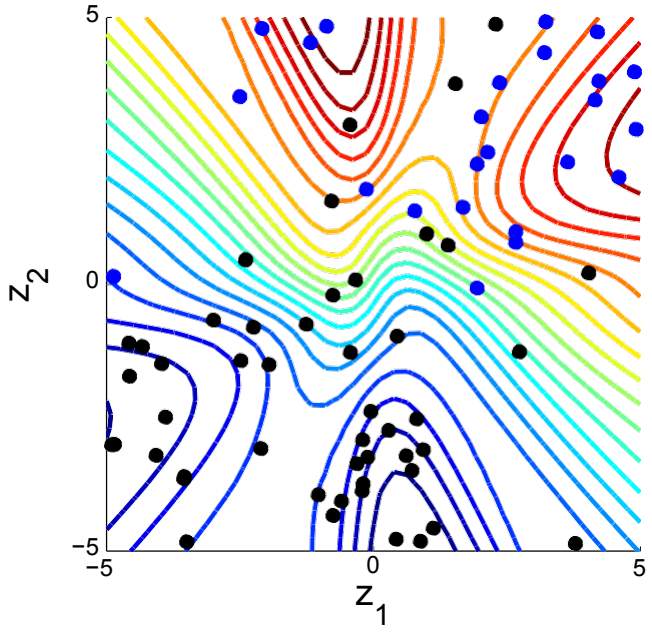


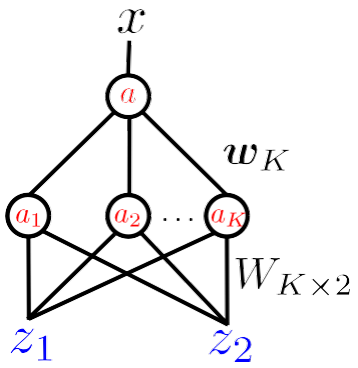




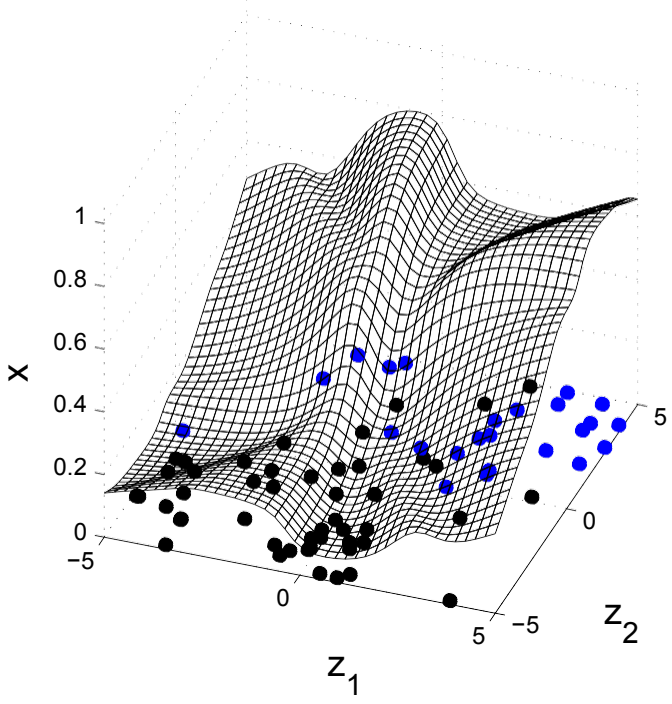
Training a Neural Network with a Single Hidden Layer

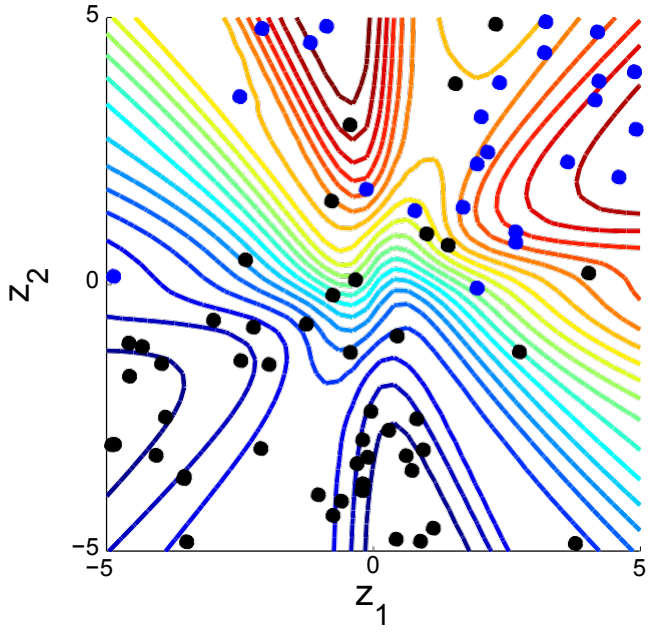


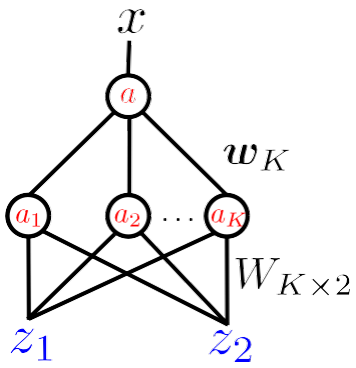




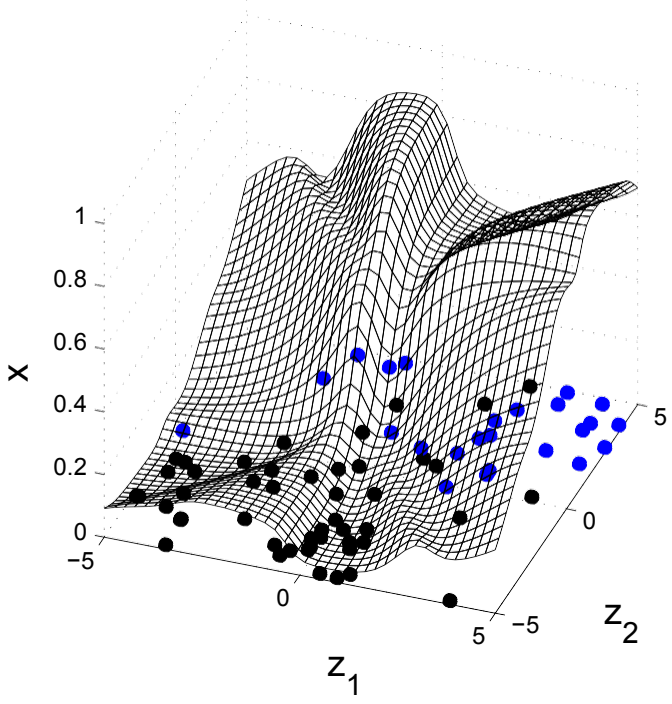
Training a Neural Network with a Single Hidden Layer

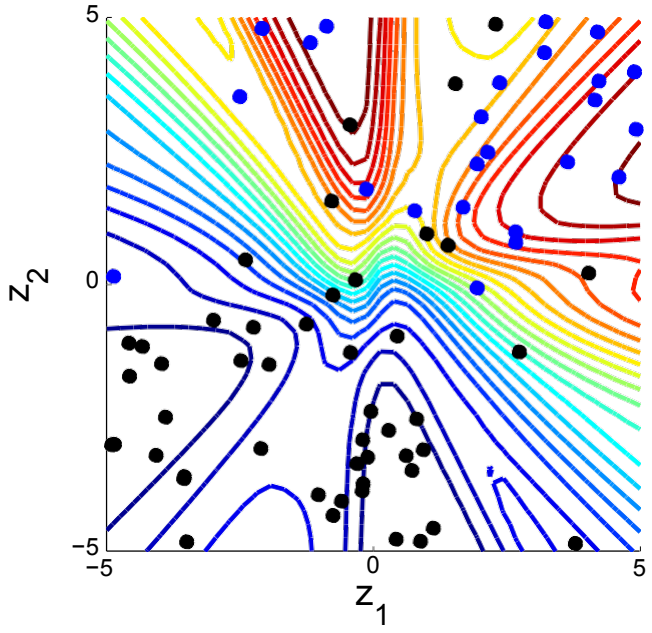


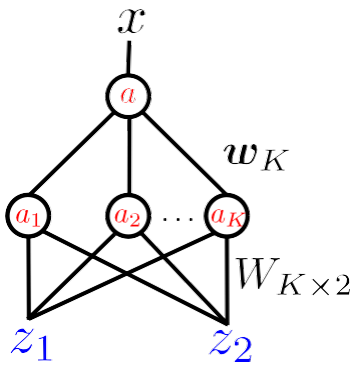




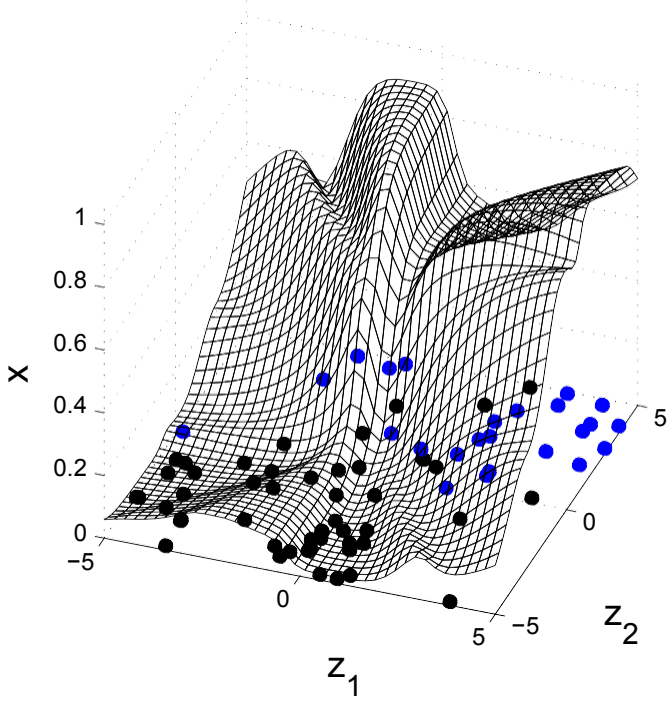
Training a Neural Network with a Single Hidden Layer

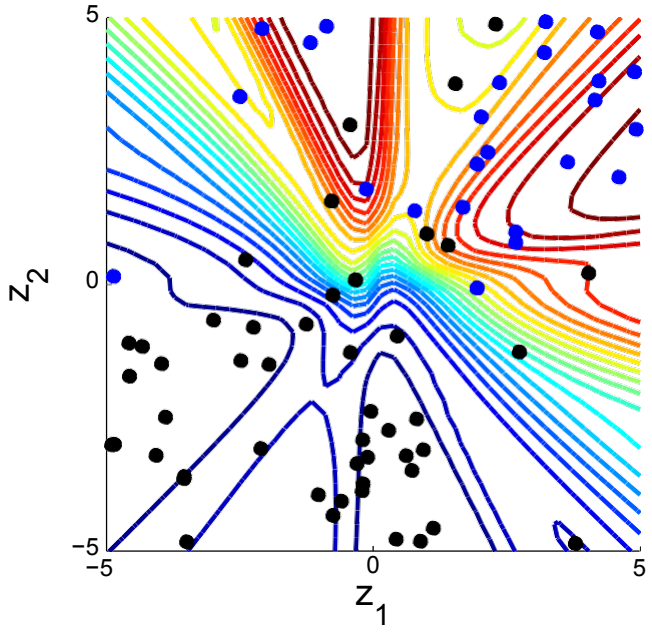


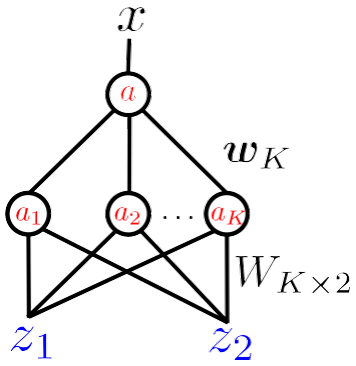




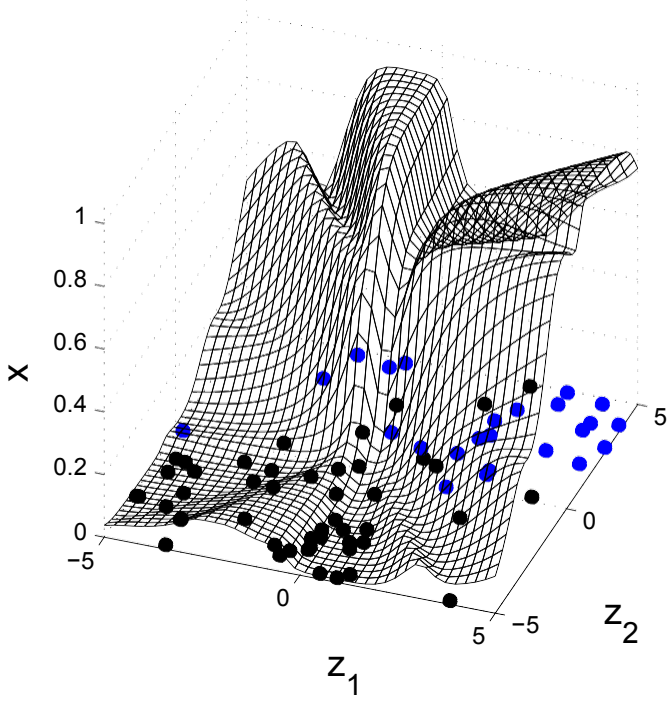
Training a Neural Network with a Single Hidden Layer

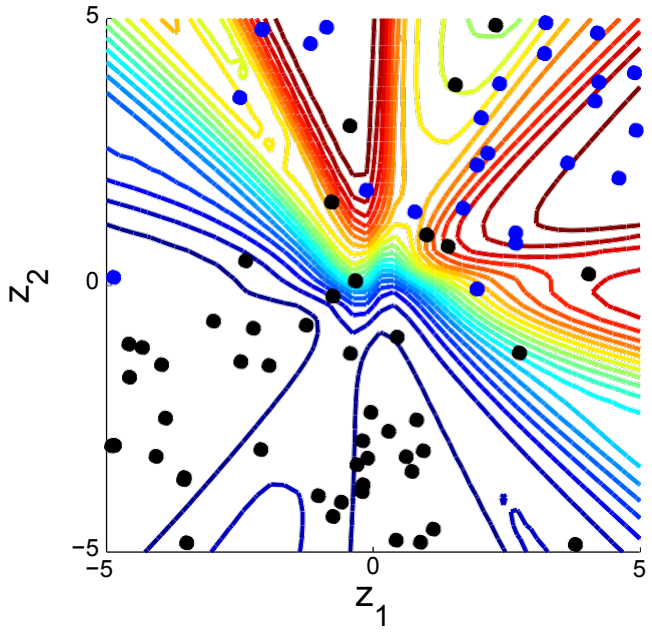


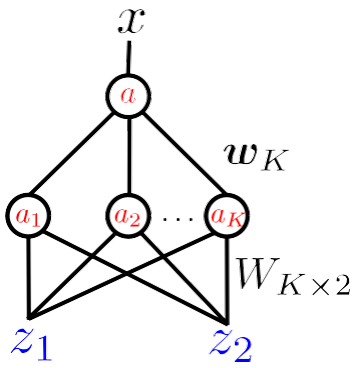




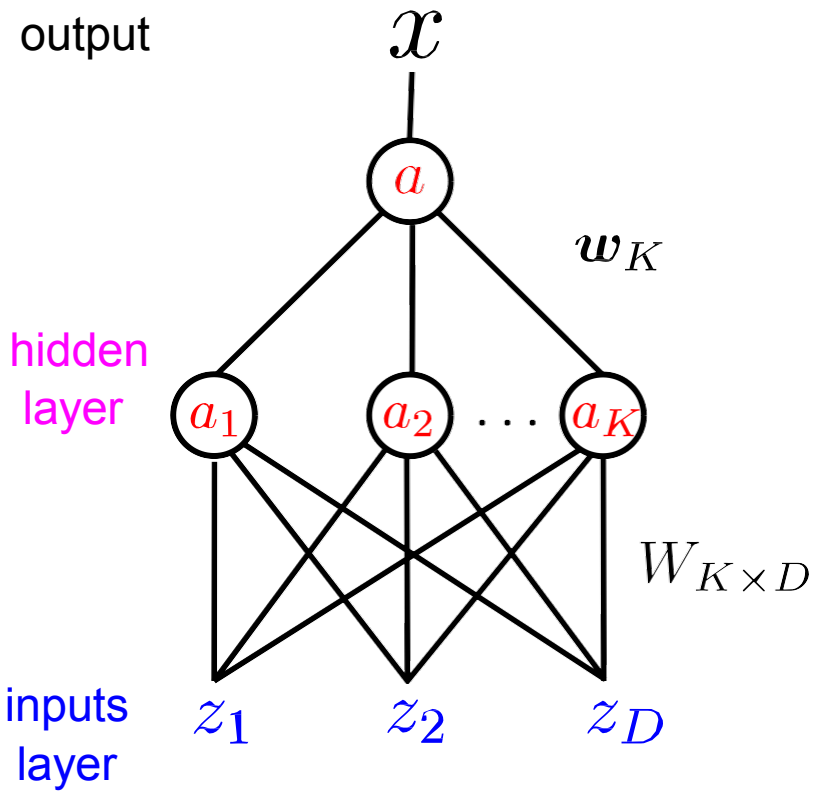
Training a Neural Network with a Single Hidden Layer

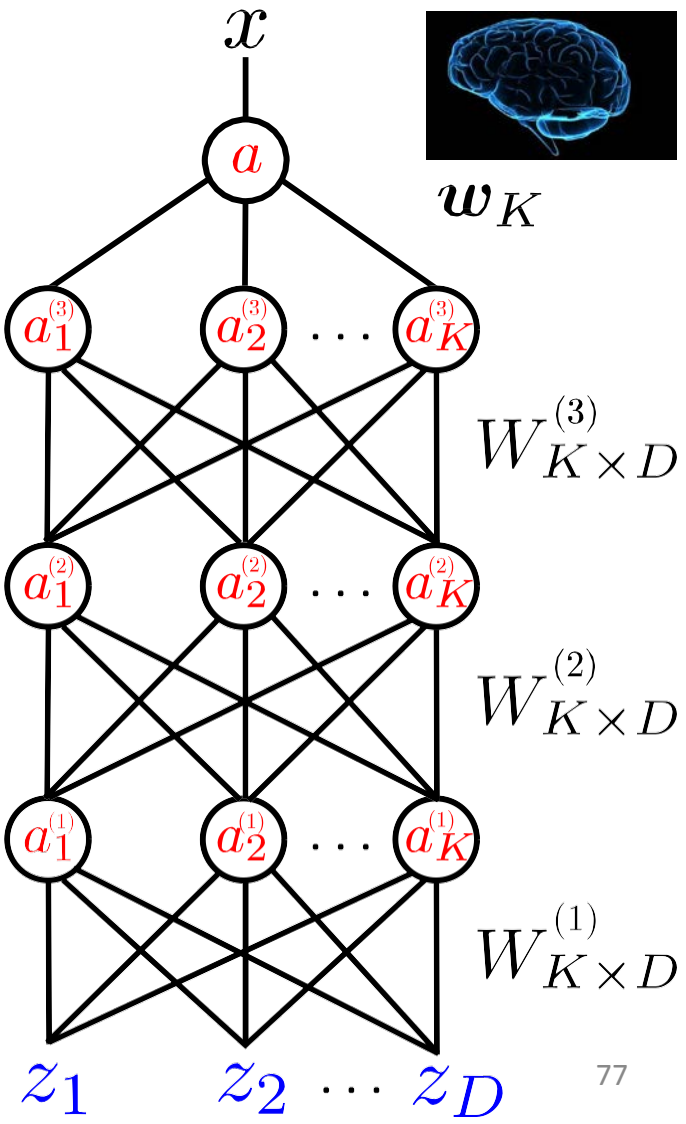






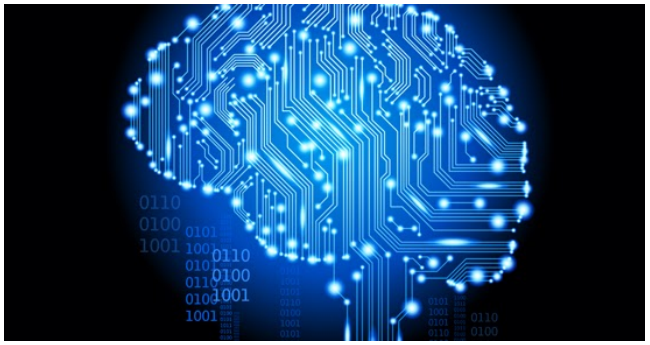
Hierarchical Models with Many Layers

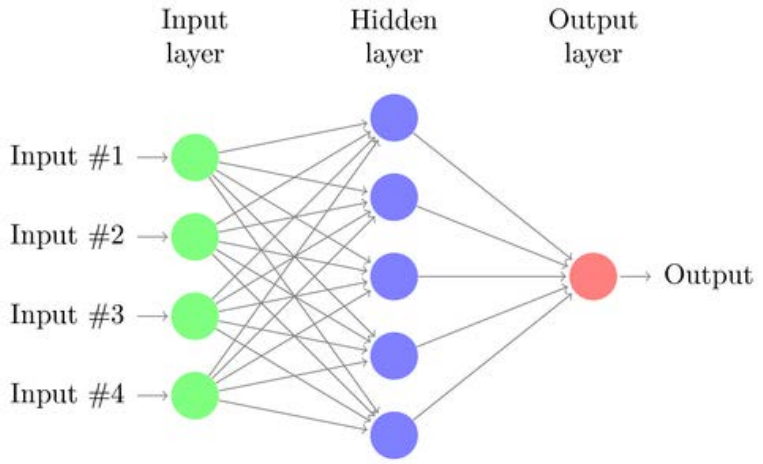




What's to Be Covered Today…

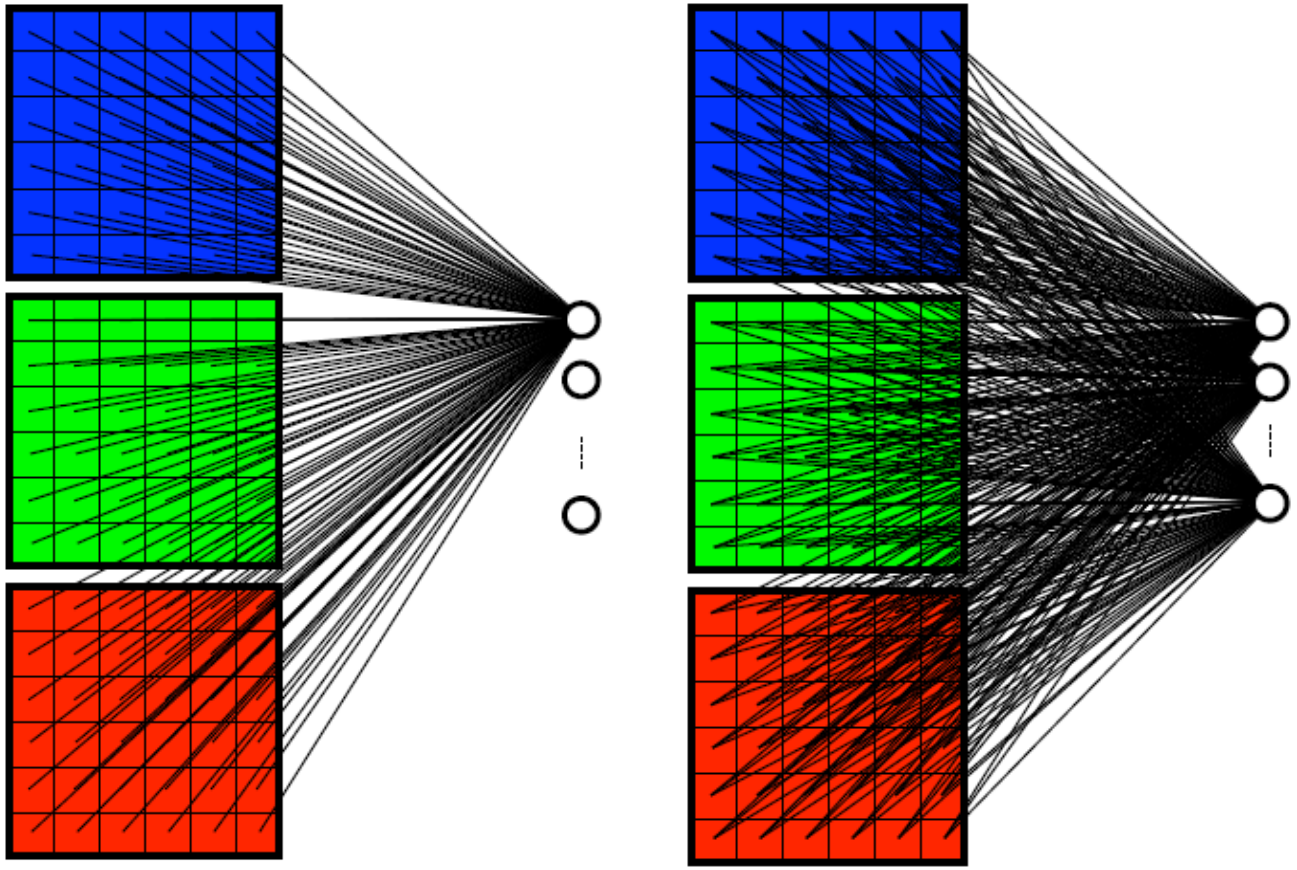
- Supervised Learning: Linear Classification
- Deep Learning Basics
  - Neural Network for Machine Vision
  - Multi-Layer Perceptron
  - Convolutional Neural Networks





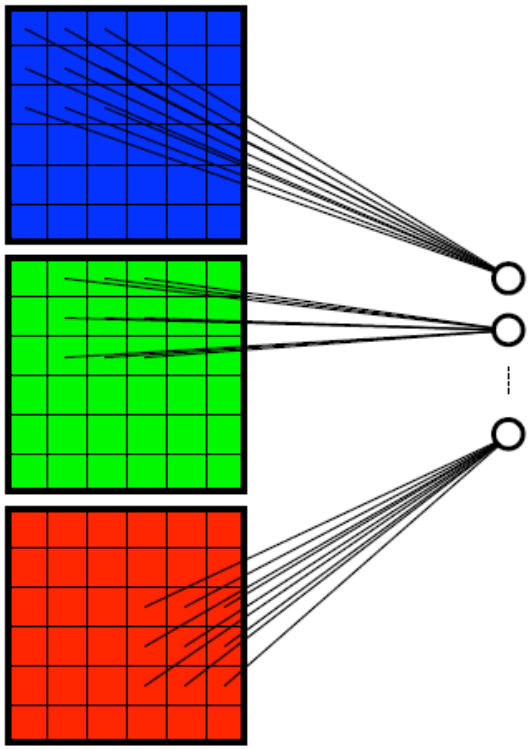
Convolutional Neural Networks

- How many weights for MLPs for images?



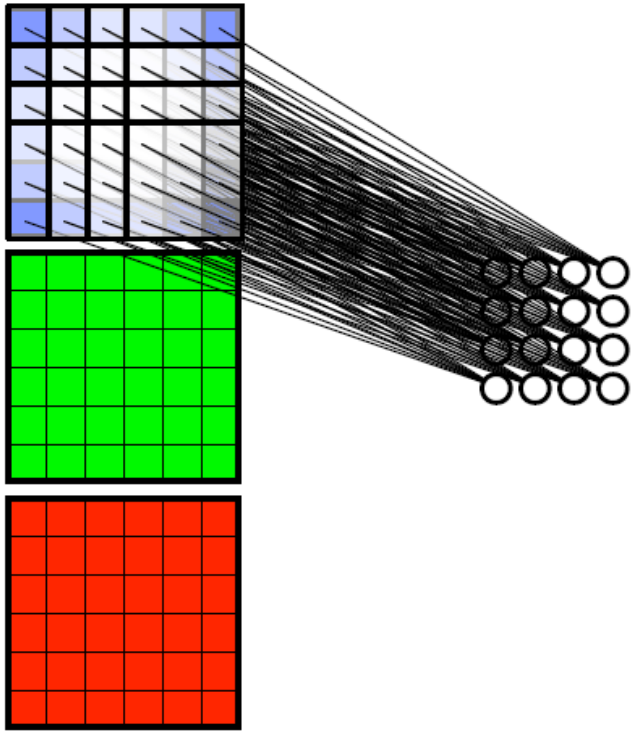
Convolutional Neural Networks

- Property I of CNN: Local Connectivity
  - Each neuron takes info only from a neighborhood of pixels.

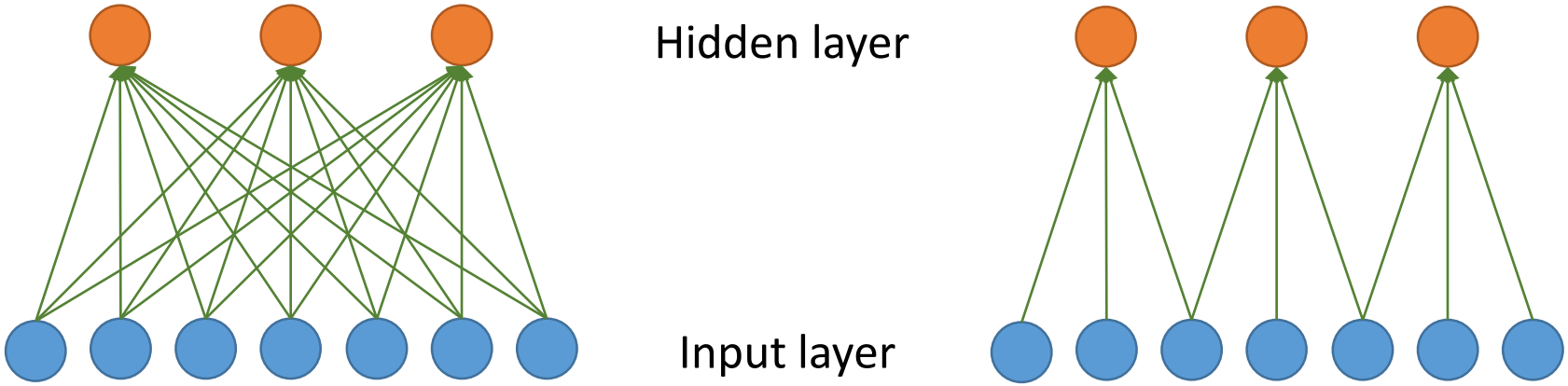


Convolutional Neural Networks

- Property II of CNN: Weight Sharing
  - Neurons connecting all neighborhoods have identical weights.



CNN: Local Connectivity

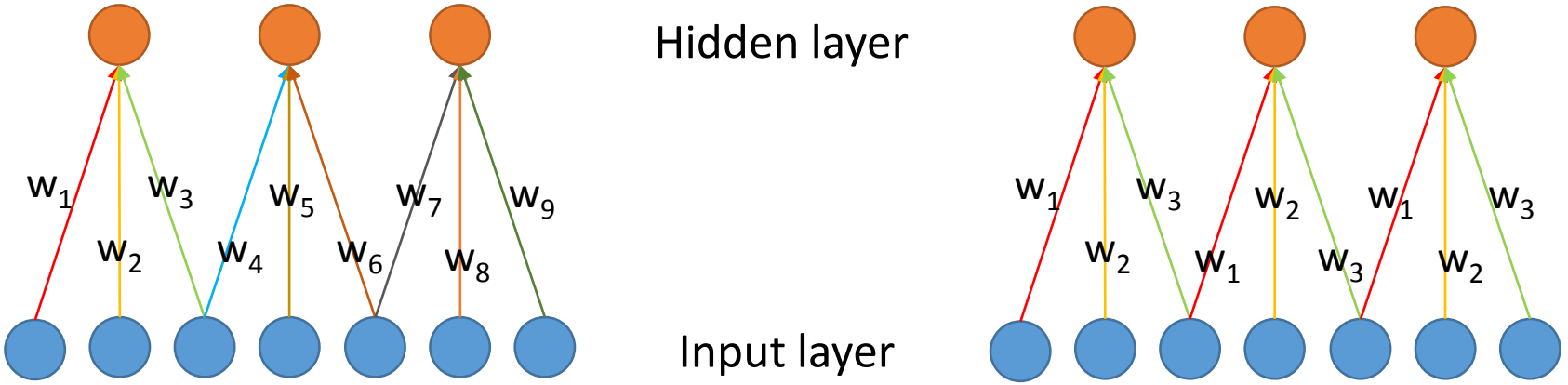


Global connectivity

Local connectivity

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

CNN: Weight Sharing

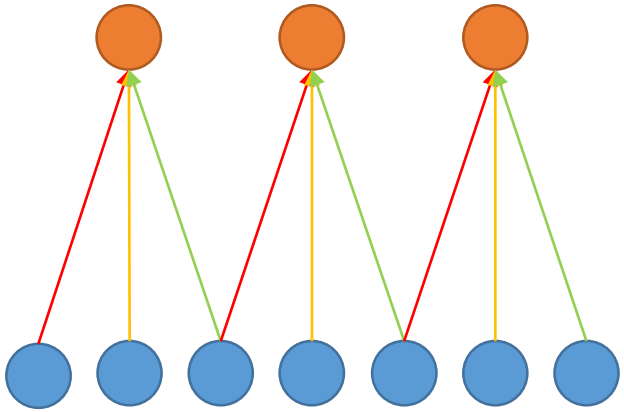


Without weight sharing

With weight sharing

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

CNN with Multiple Input Channels

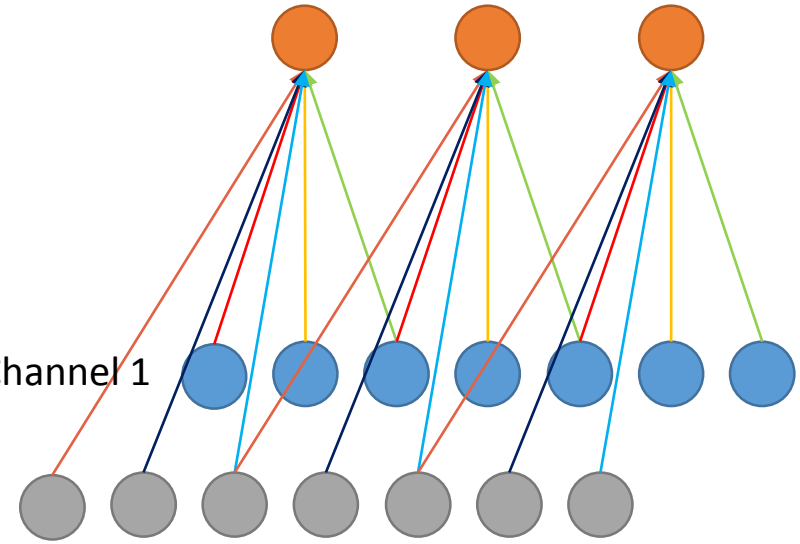


Hidden layer

Input layer

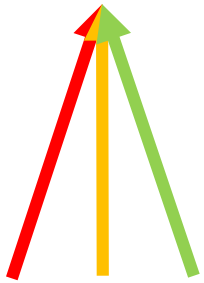
Channel 1

Channel 2

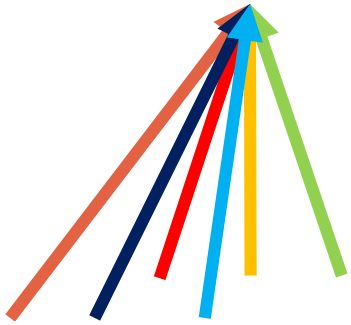


Single input channel

Multiple input channels

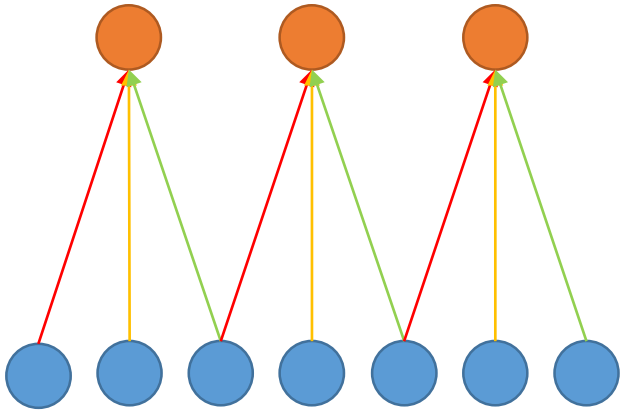


Filter weights



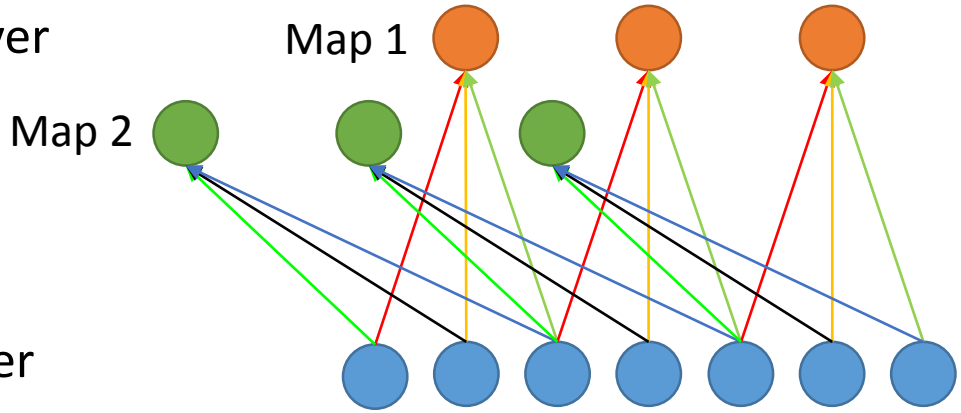
Filter weights

CNN with Multiple Output Maps

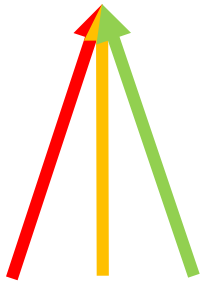


Hidden layer

Input layer

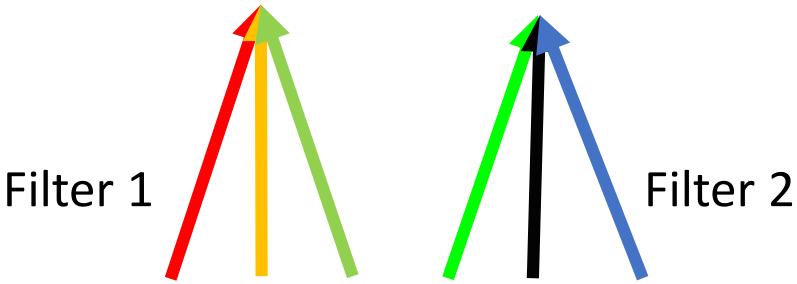


Single output map



Filter weights

Multiple output maps



Filter weights

Putting them together

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

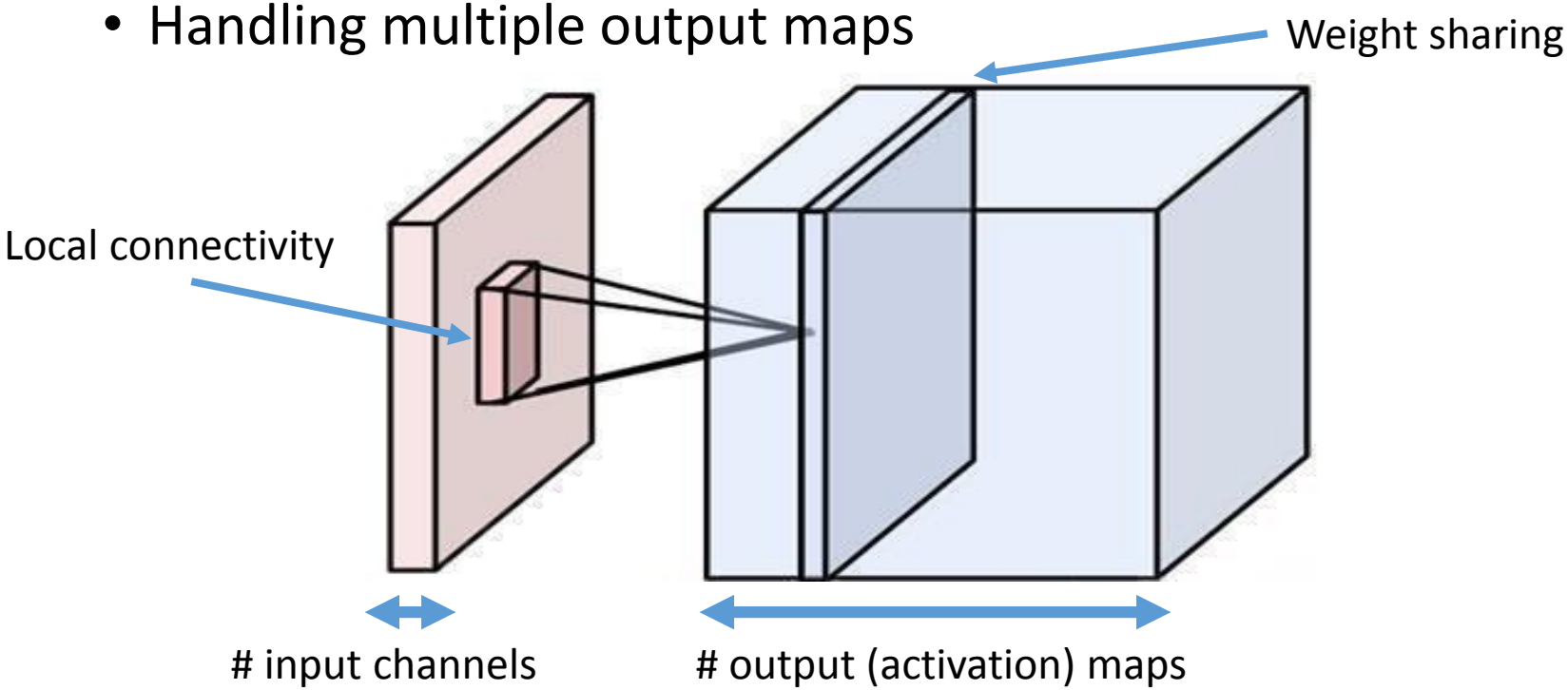
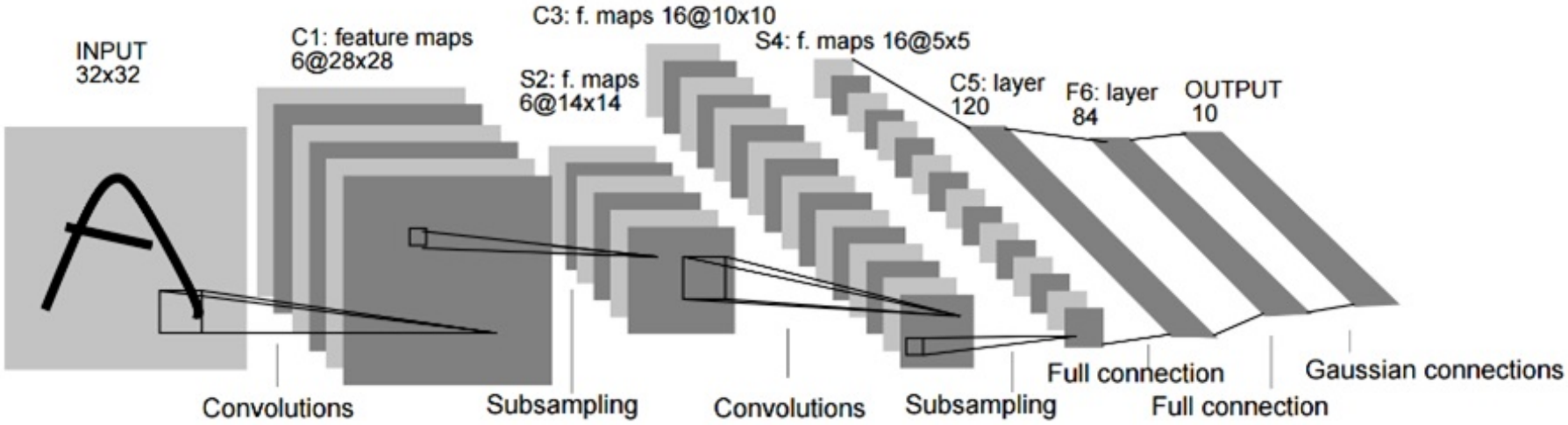


Image credit: A. Karpathy

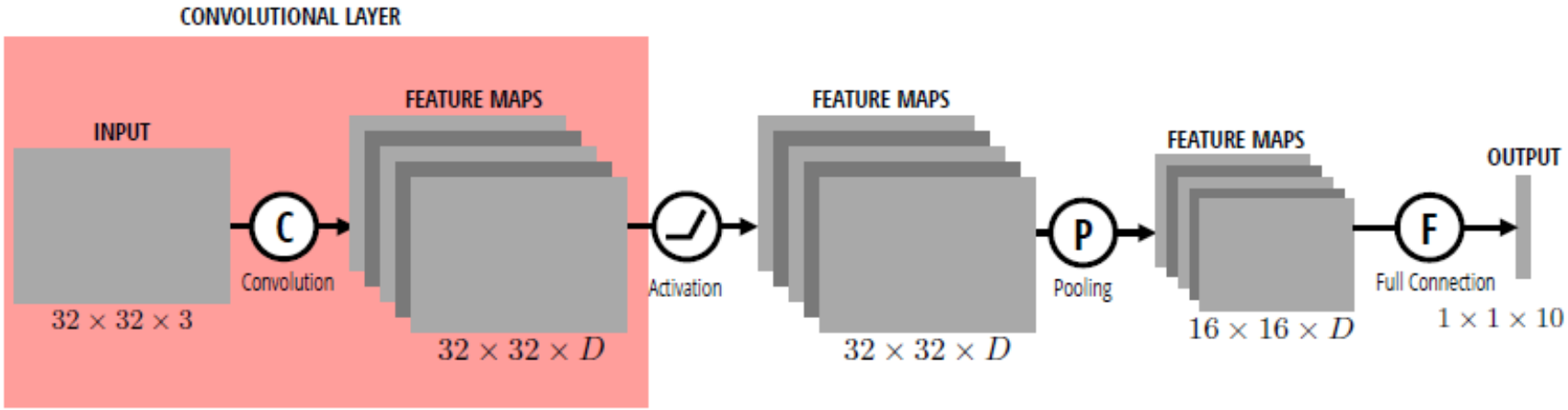
LeNet [LeCun et al. 1998]



Gradient-based learning applied to document recognition [LeCun, Bottou, Bengio, Haffner 1998]

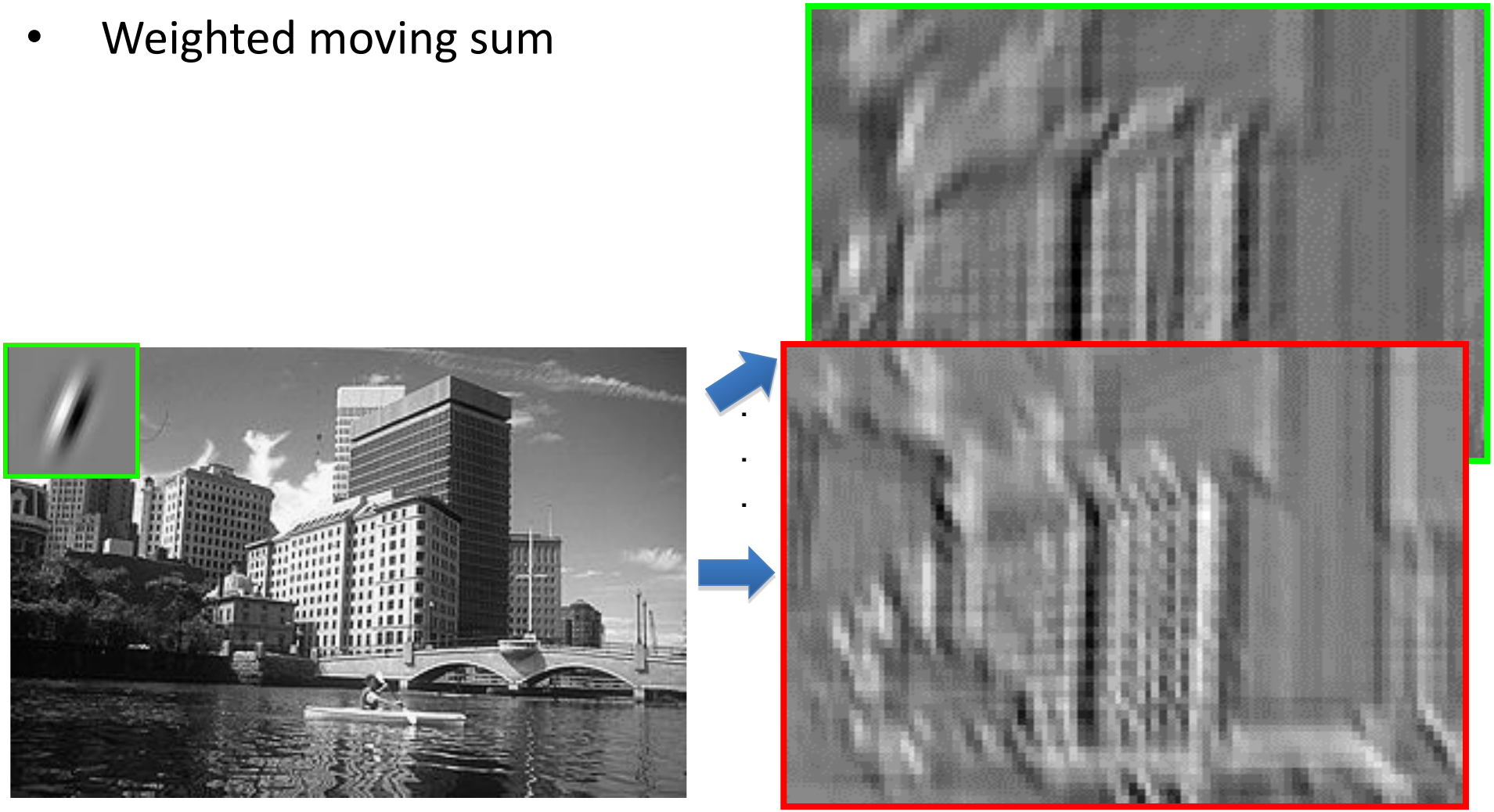
LeNet-1 from 1993

Convolution Layer in CNN



What is a Convolution?

- Weighted moving sum

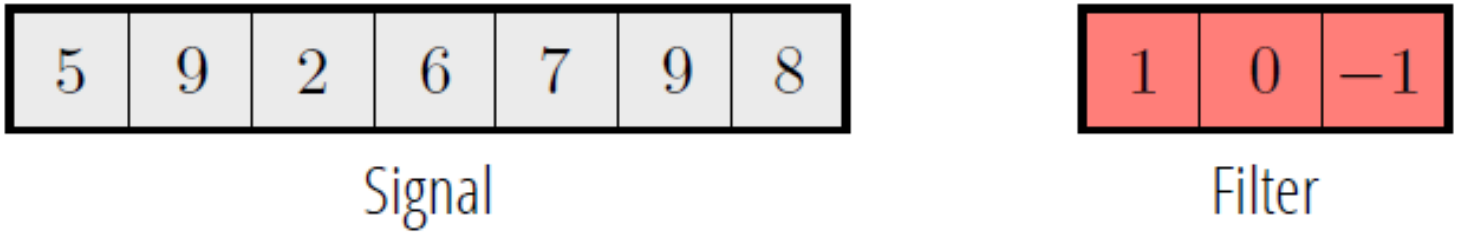


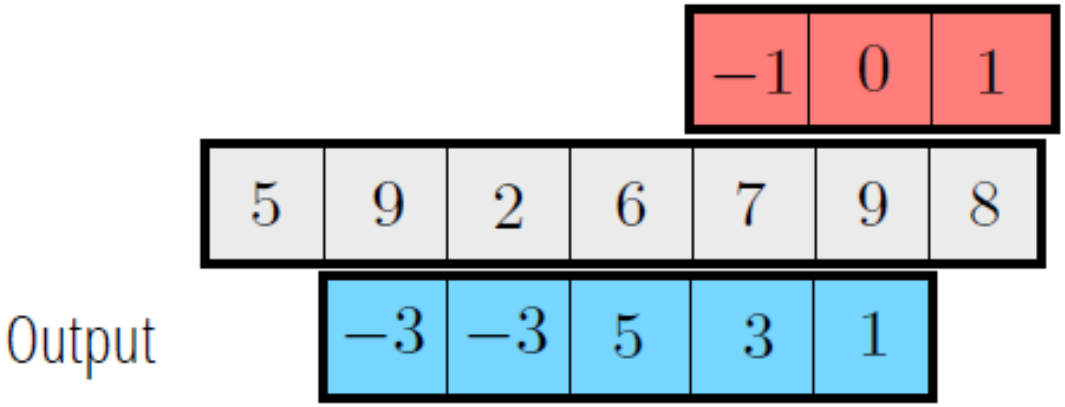
Input

Feature Activation Map

slide credit: S. Lazebnik

What is a Convolution?

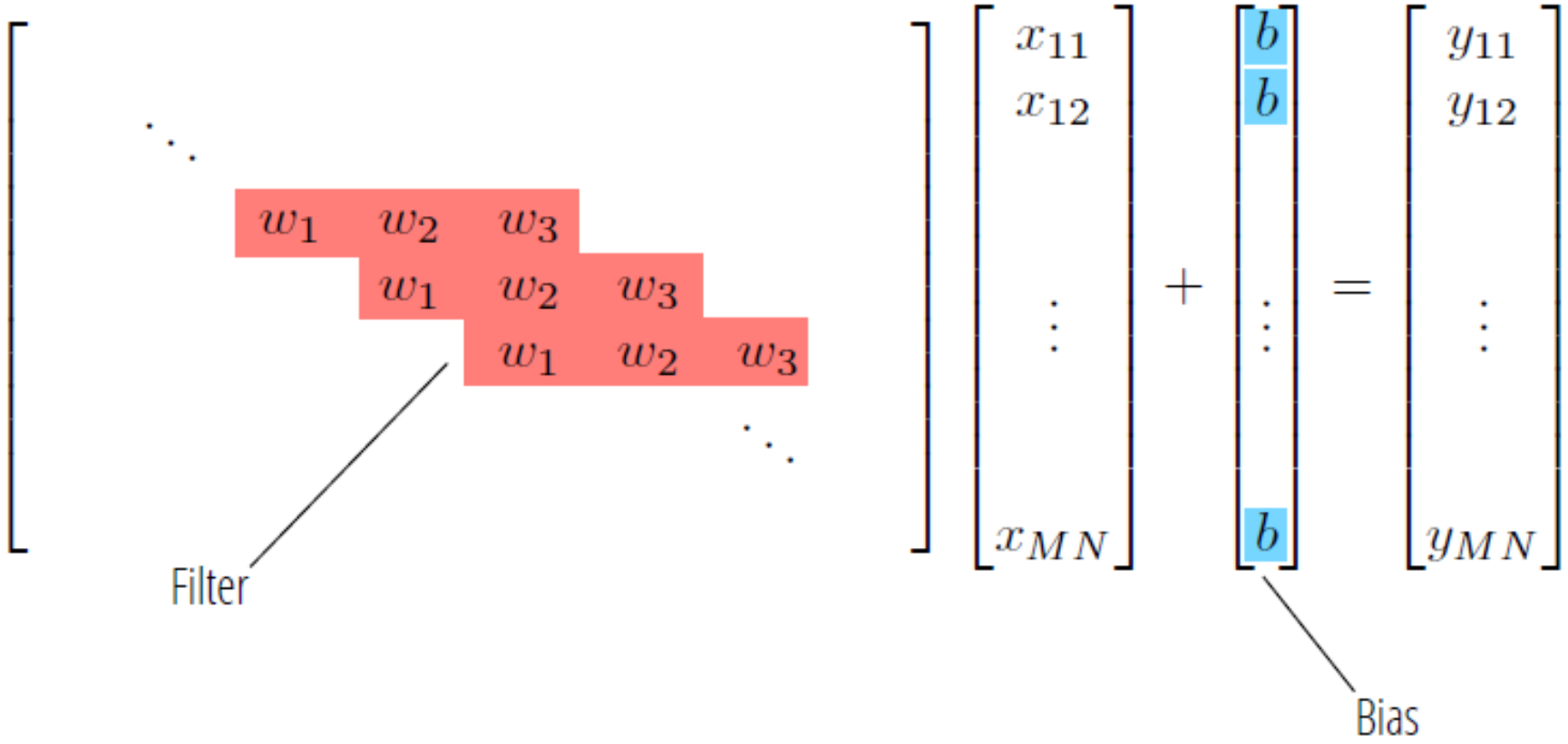




Convolution is a local linear operator

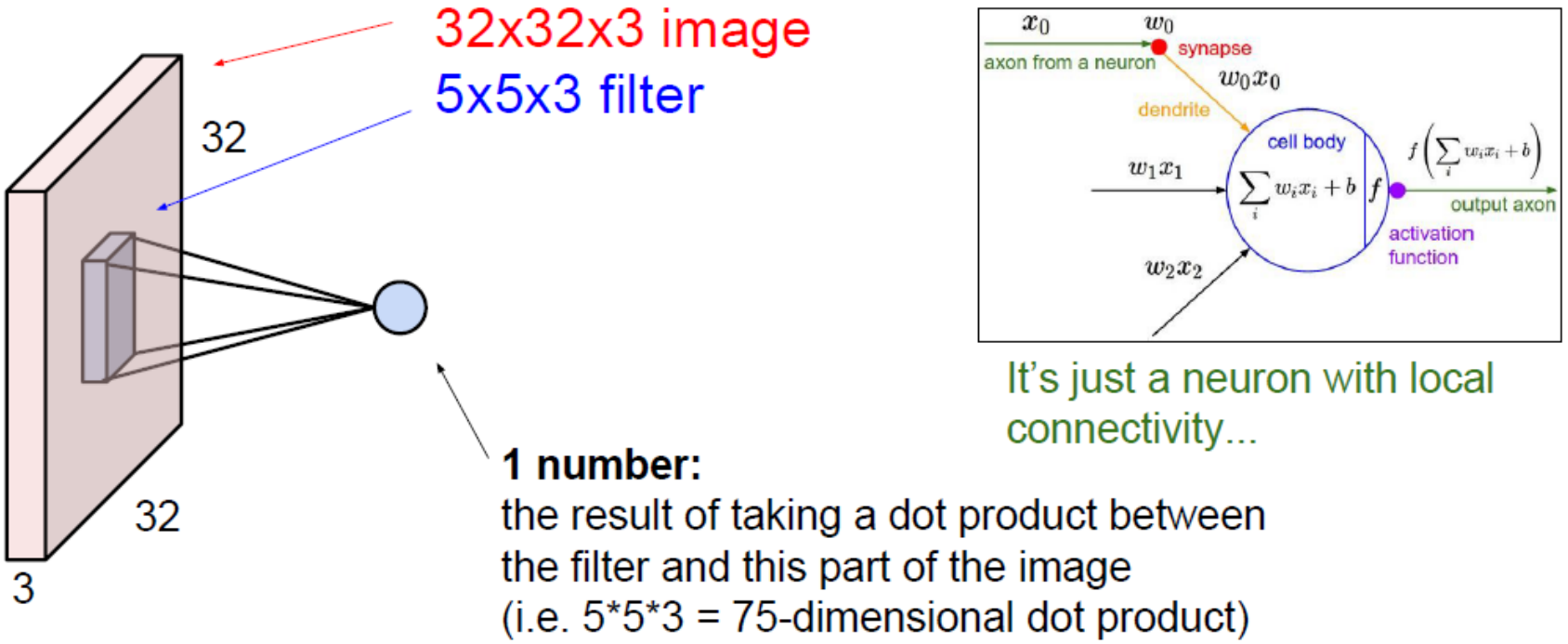
What is a Convolution?

- Toeplitz Matrix Form



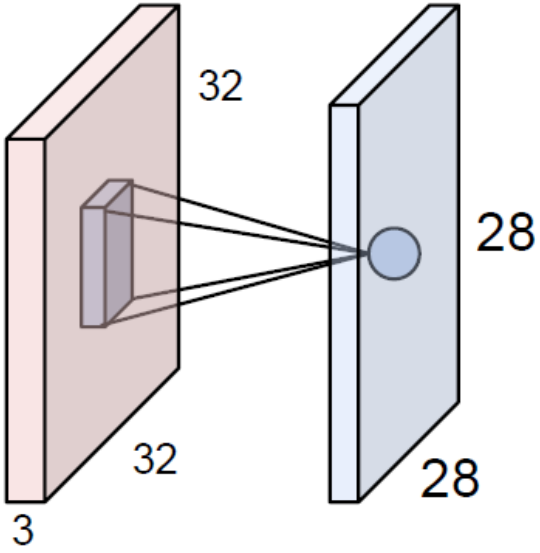
Putting them together (cont'd)

- The brain/neuron view of CONV layer



Putting them together (cont'd)

- The brain/neuron view of CONV layer



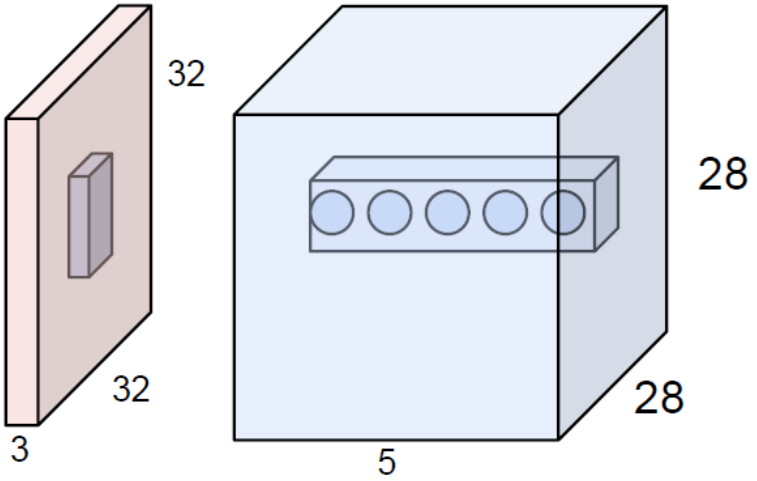
An activation map is a 28x28 sheet of neuron outputs:

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

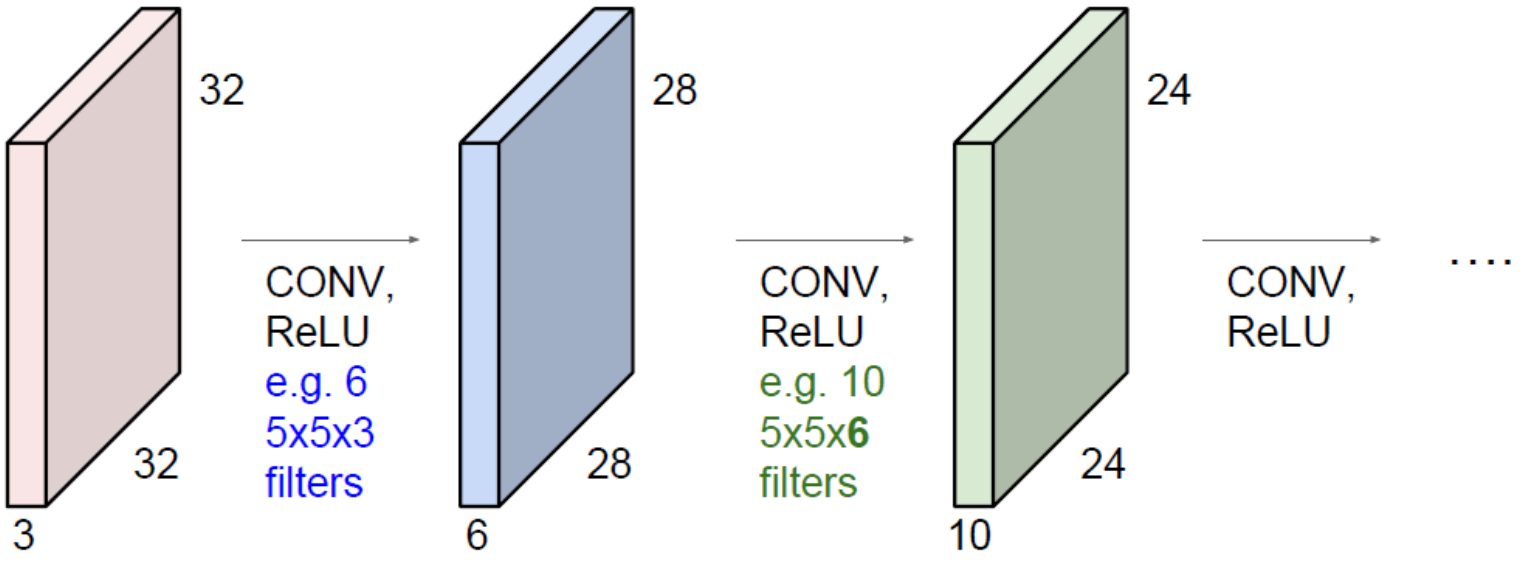


E.g. with 5 filters,
CONV layer consists of
neurons arranged in a 3D grid
(28x28x5)

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

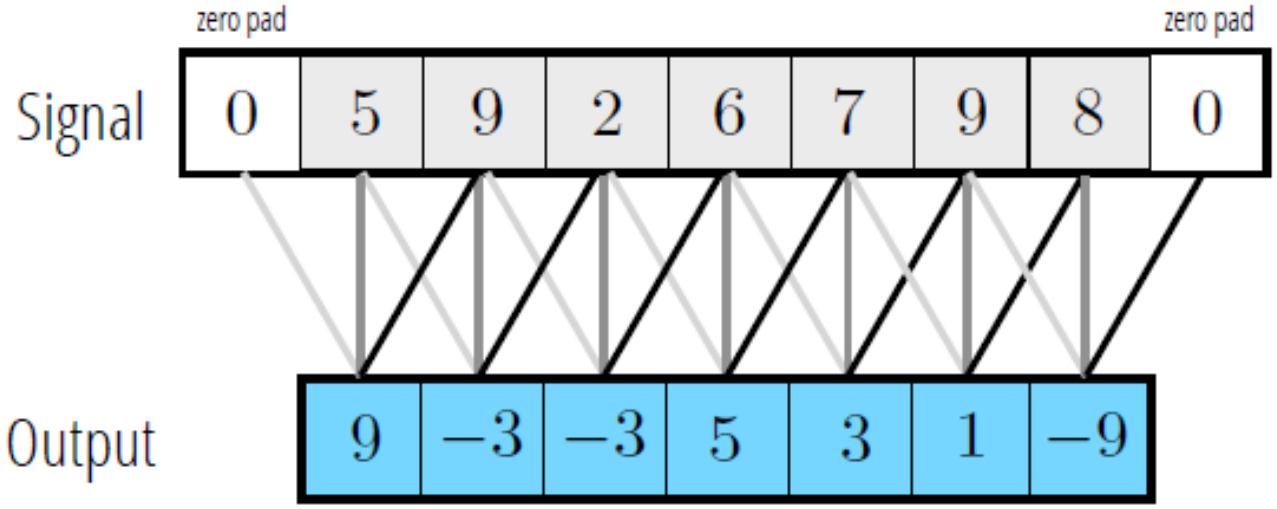
Putting them together (cont'd)

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



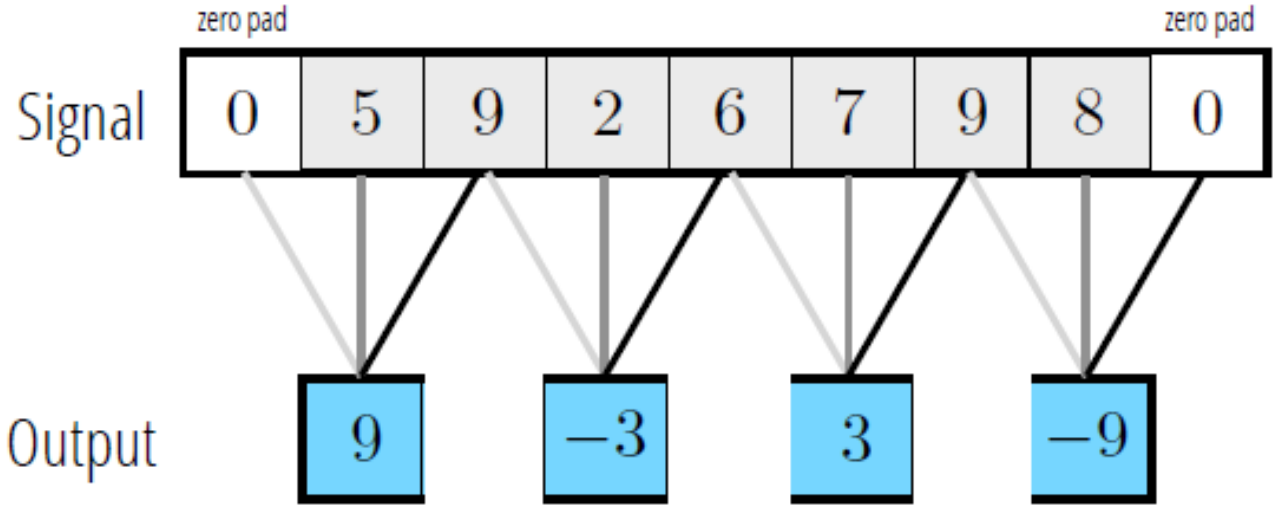
What is a Convolution?

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



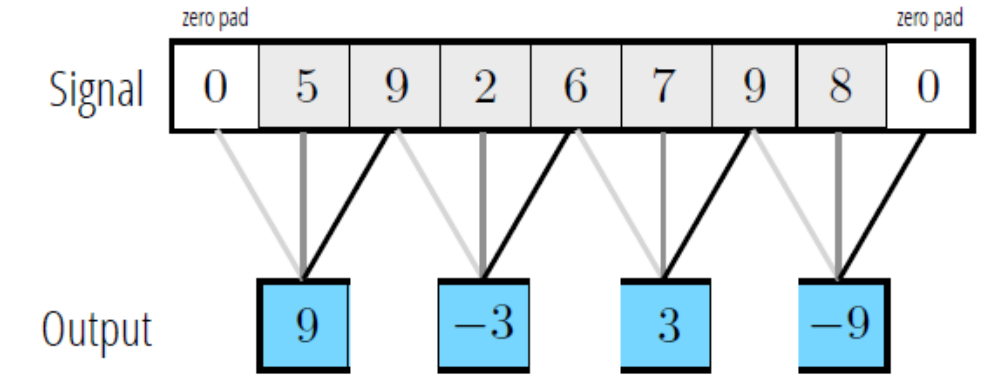
What is a Convolution?

- Stride
  - Step size across signals



What is a Convolution?

- Stride
  - Step size across signals



Input Size

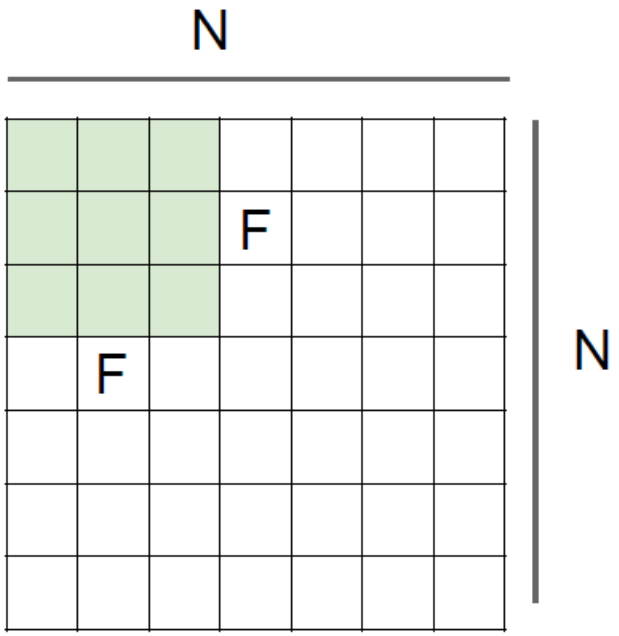
Filter Size

$$\text{Output Size} = \frac{N - c}{s} + 1$$

Stride: step size across the signal

What is a Convolution?

- Stride
  - Step size across signals

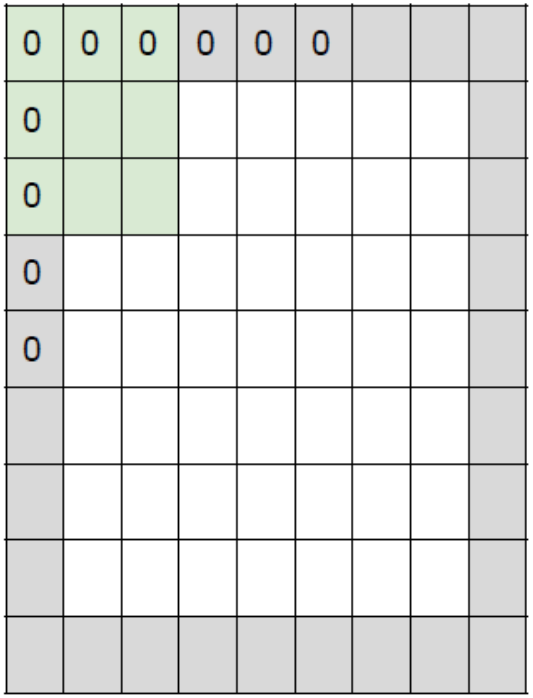


Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

What is a Convolution?

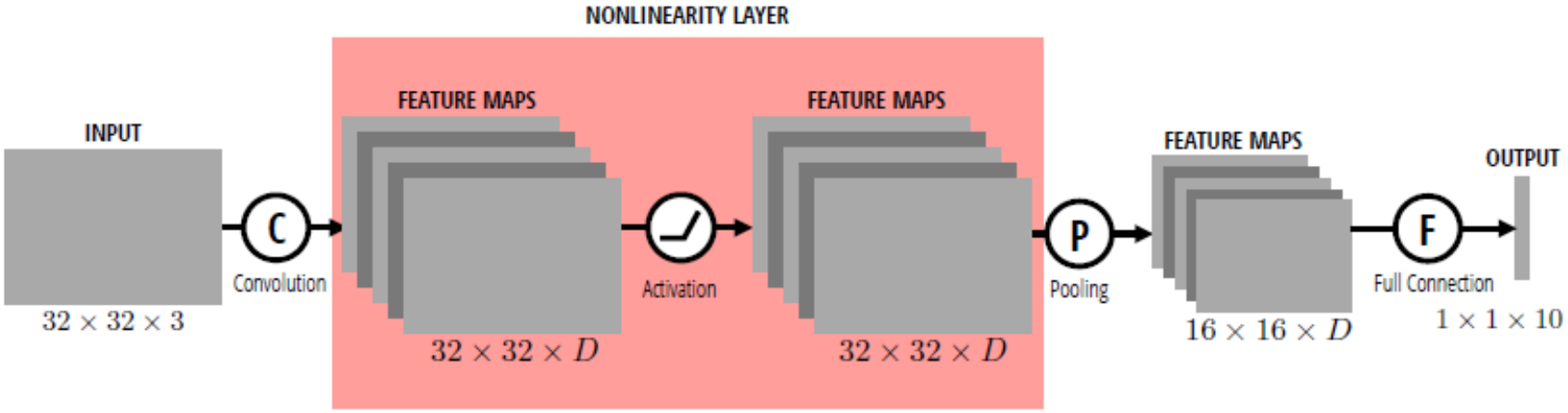
- Zero Padding + Stride



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

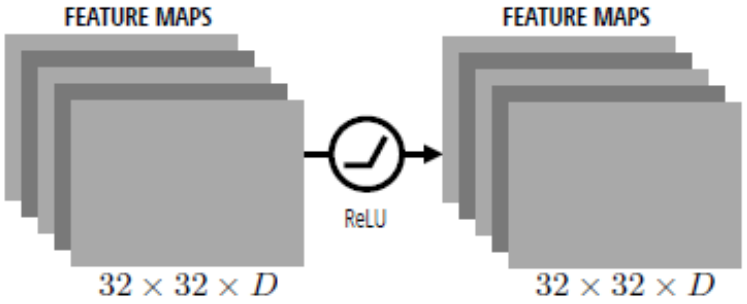
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

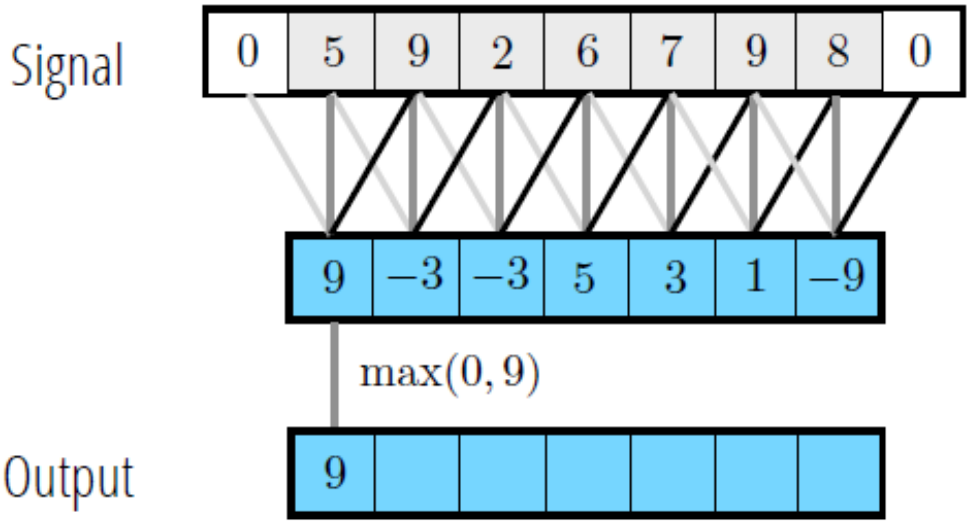
Nonlinearity Layer in CNN



Nonlinearity Layer

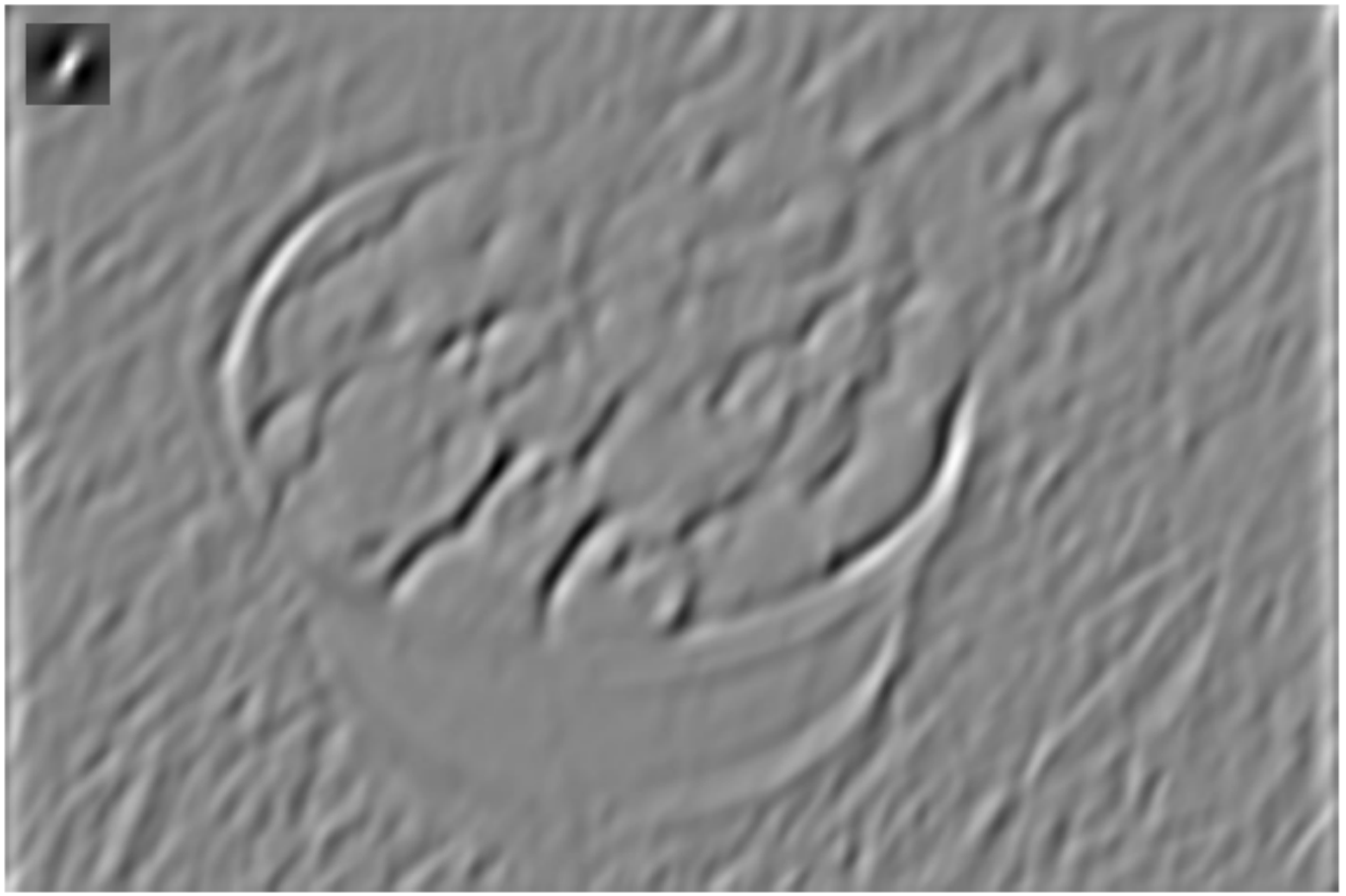
- E.g., ReLU (Rectified Linear Unit)
  - Pixel by pixel computation of max(0, x)





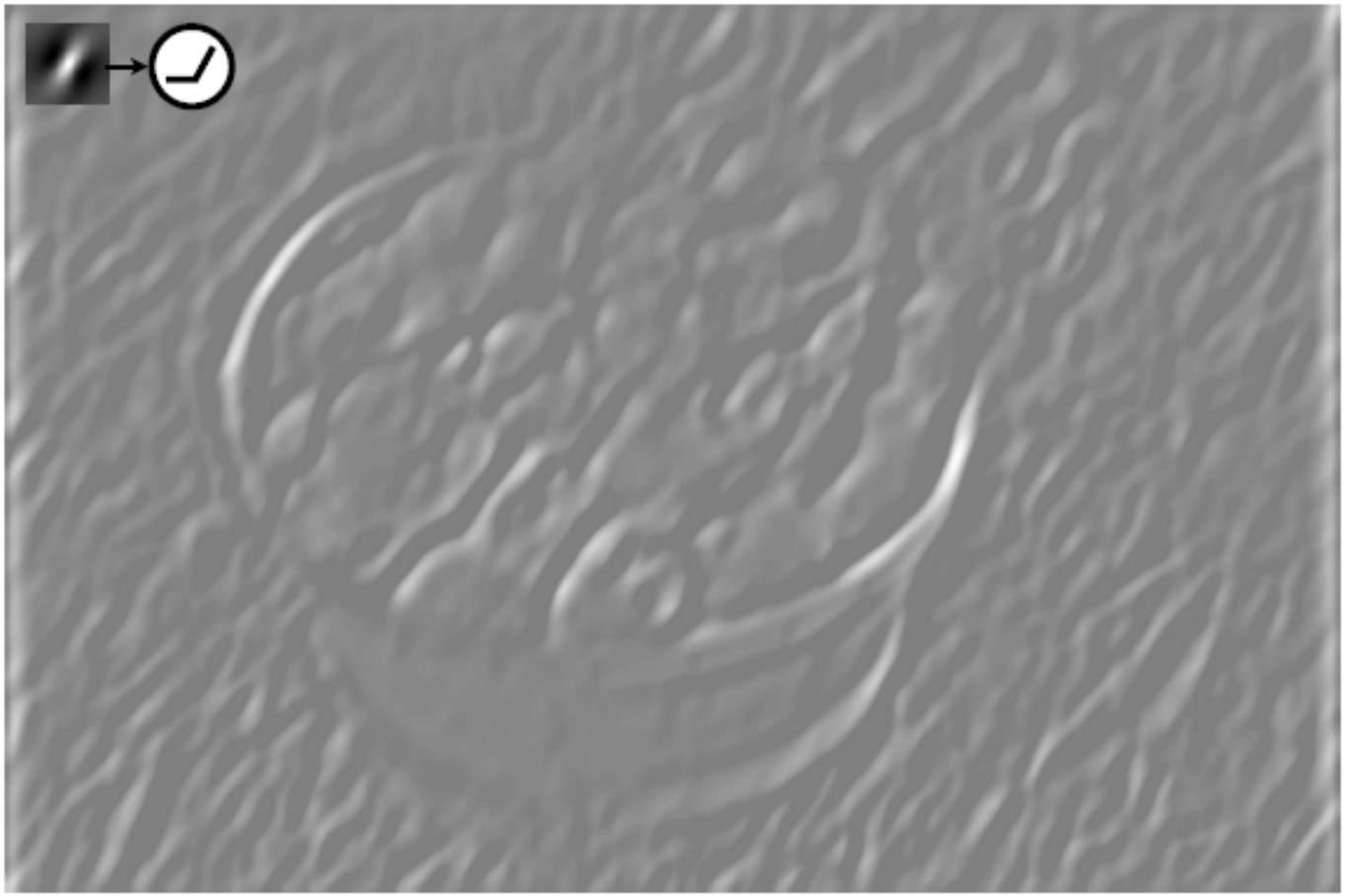
Nonlinearity Layer

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

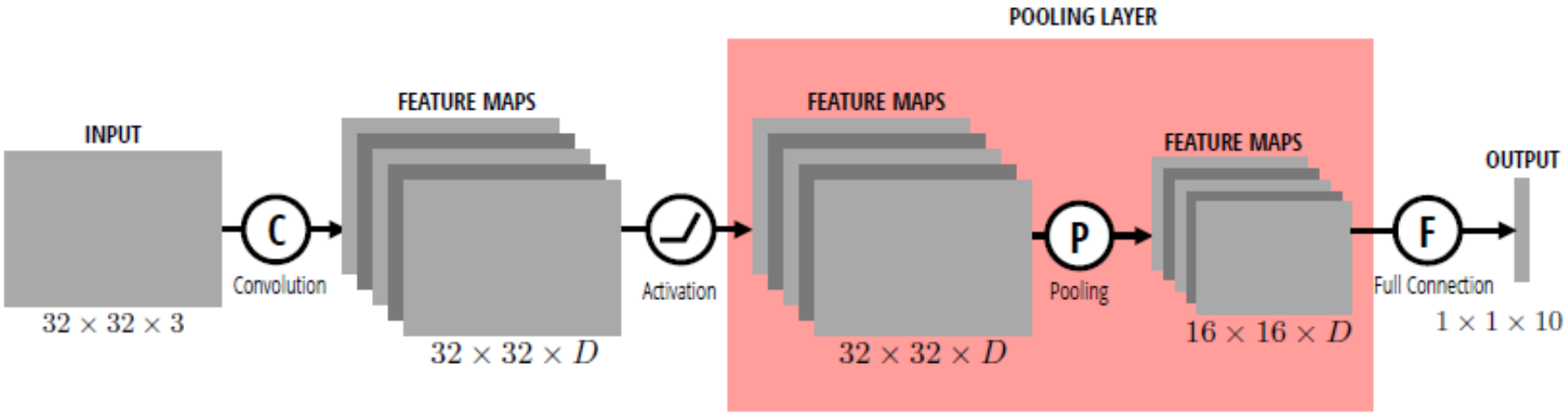


Nonlinearity Layer

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

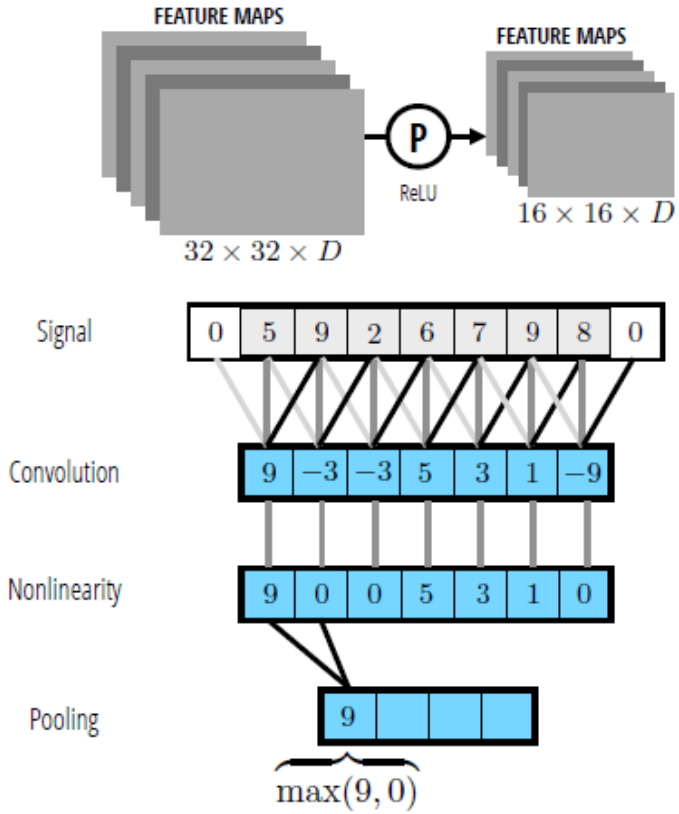


Pooling Layer in CNN



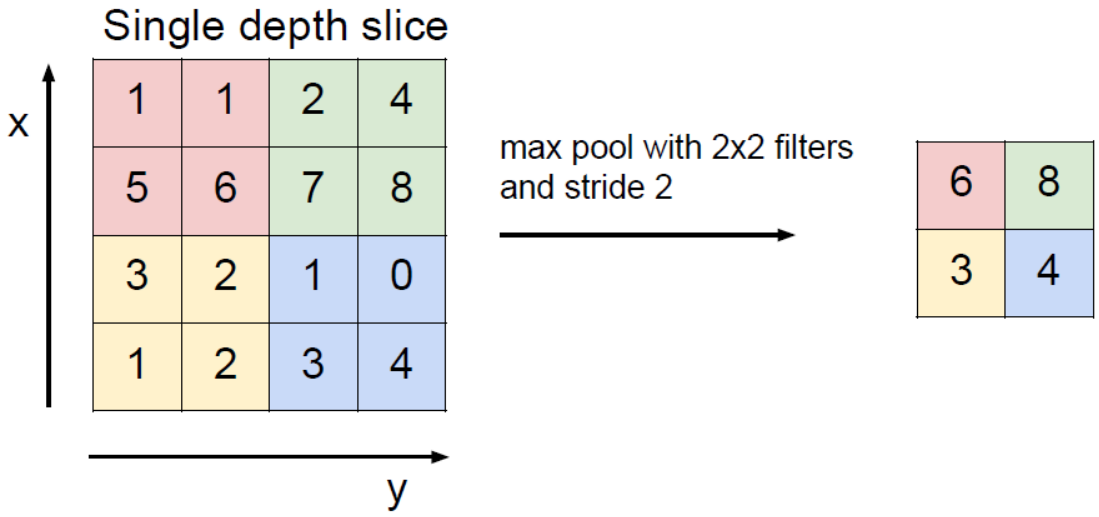
Pooling Layer

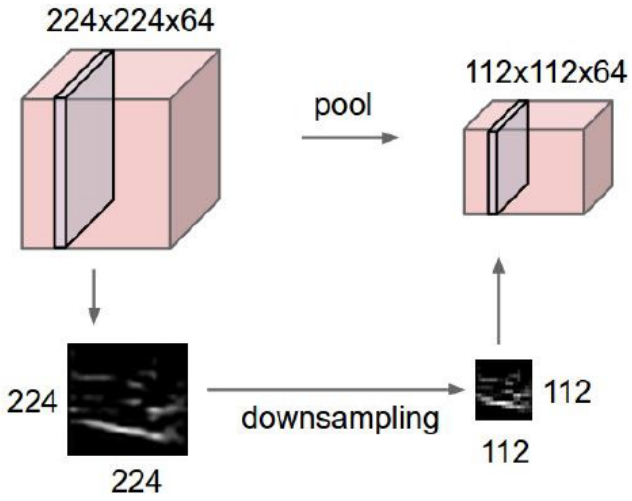
- Makes the representations smaller and more manageable
- Operates over each activation map independently
- E.g., Max Pooling



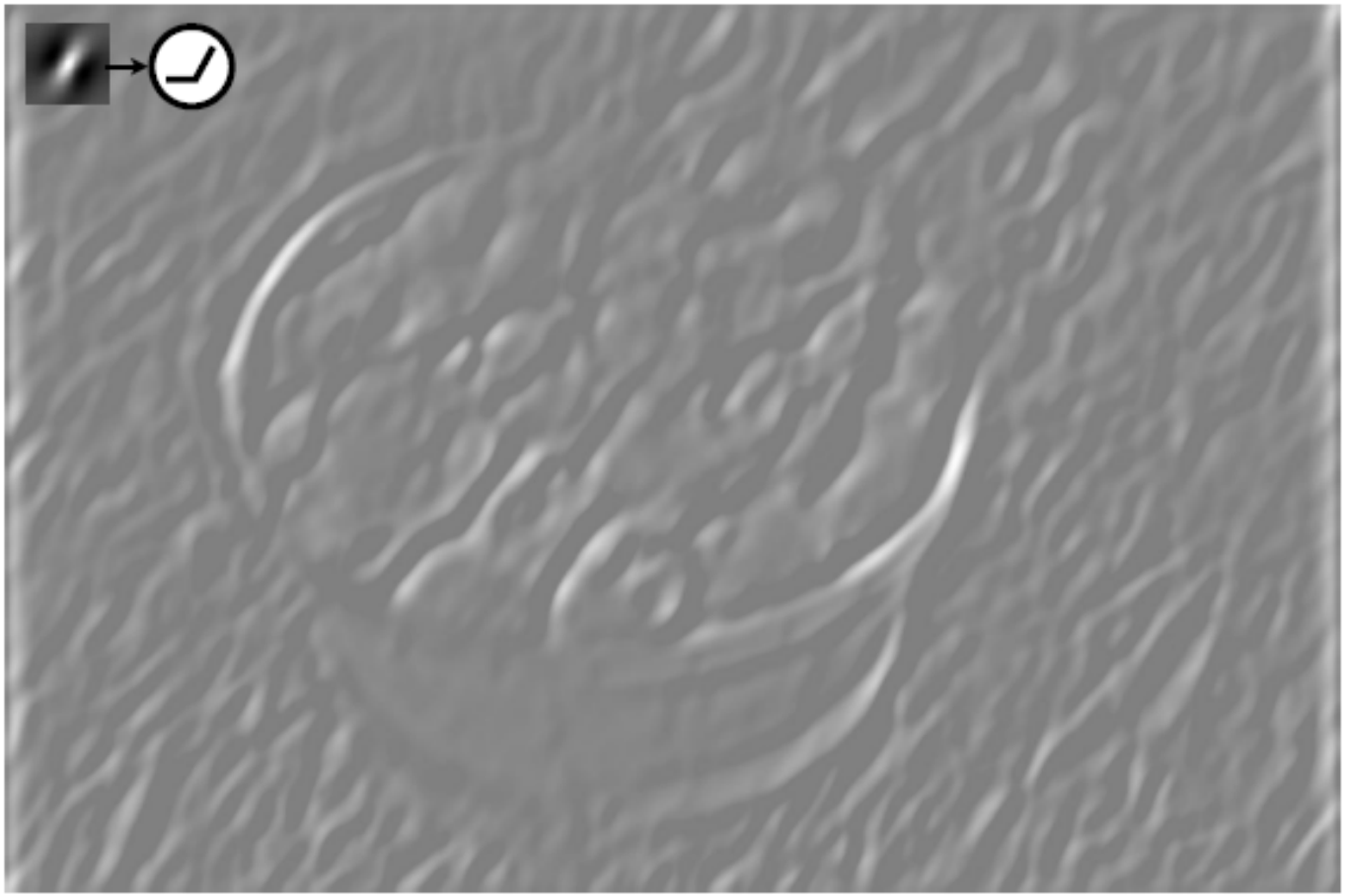
Pooling Layer

- Reduces the spatial size and provides spatial invariance

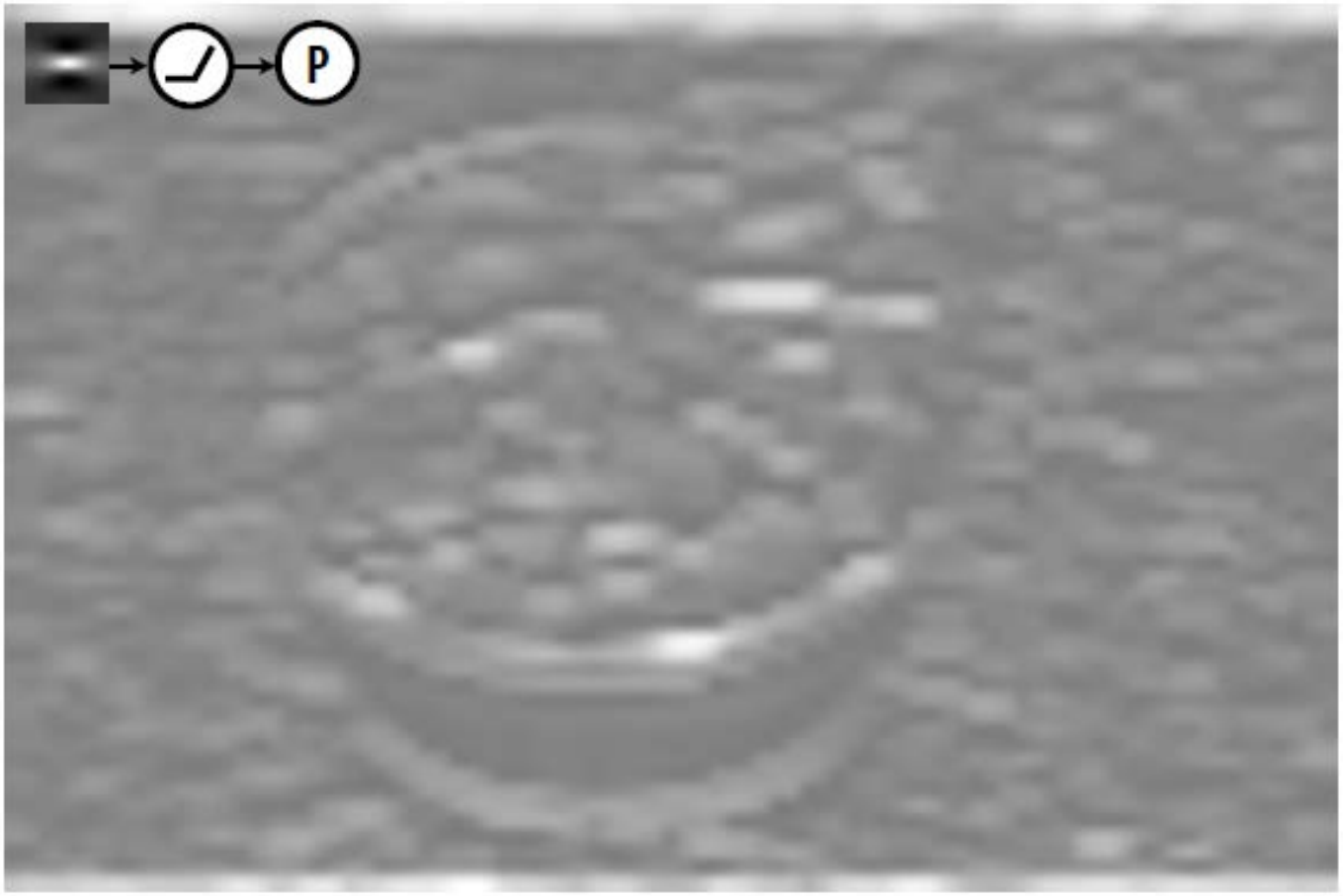




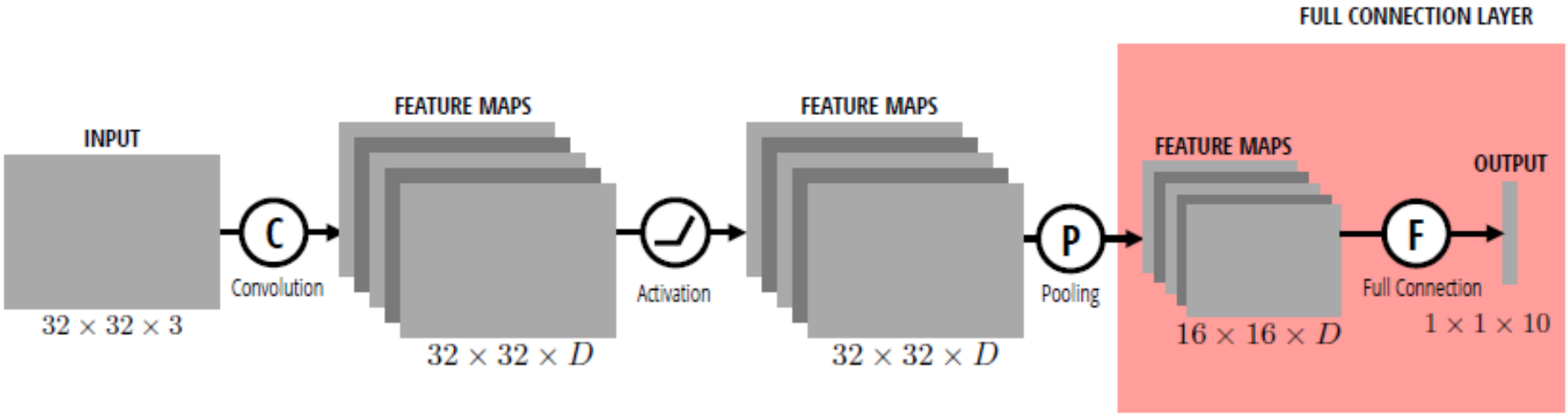
- Example
  - Nonlinearity by ReLU



- Example
  - Max pooling

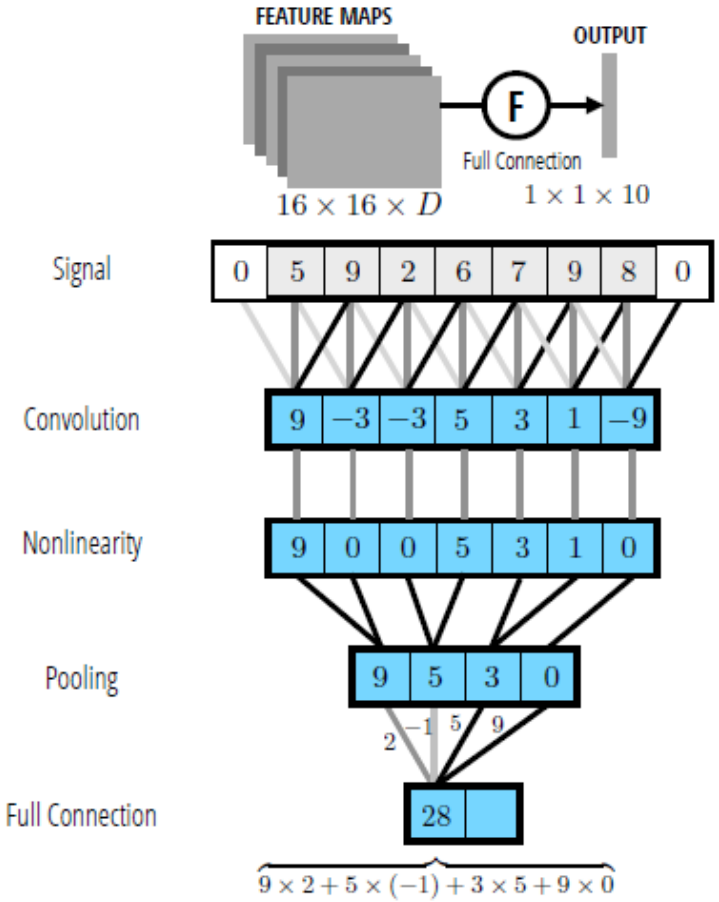


Fully Connected (FC) Layer in CNN



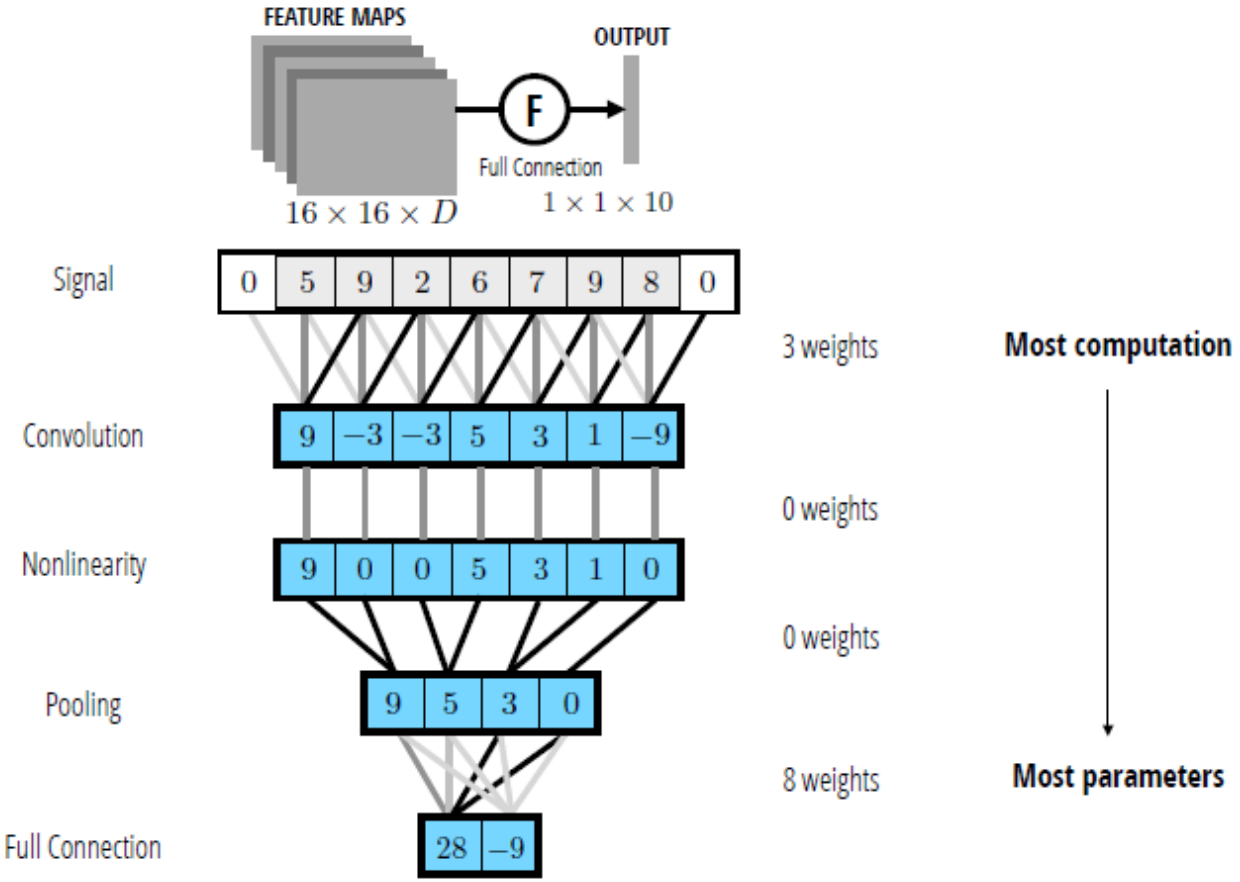
FC Layer

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

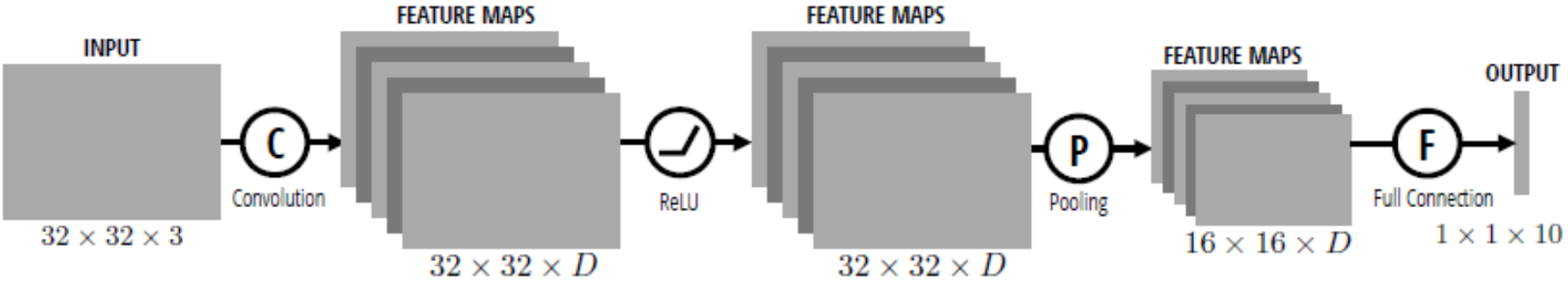


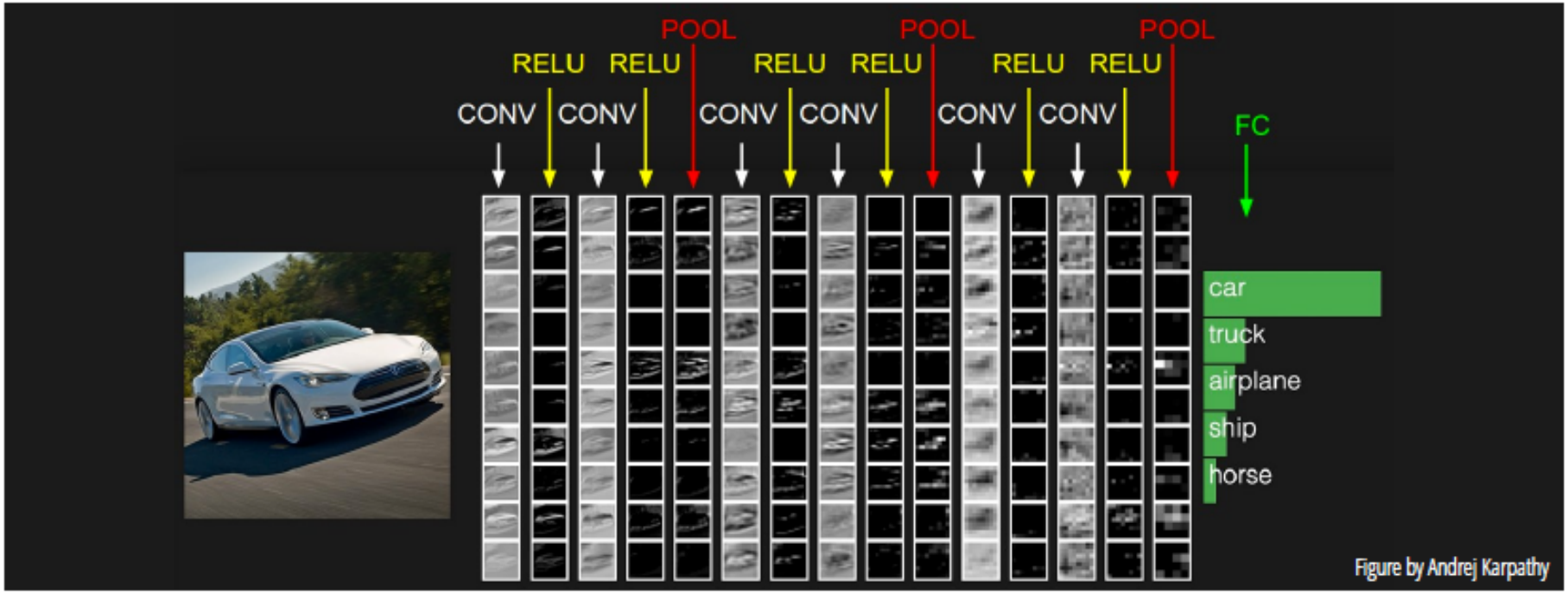
FC Layer

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

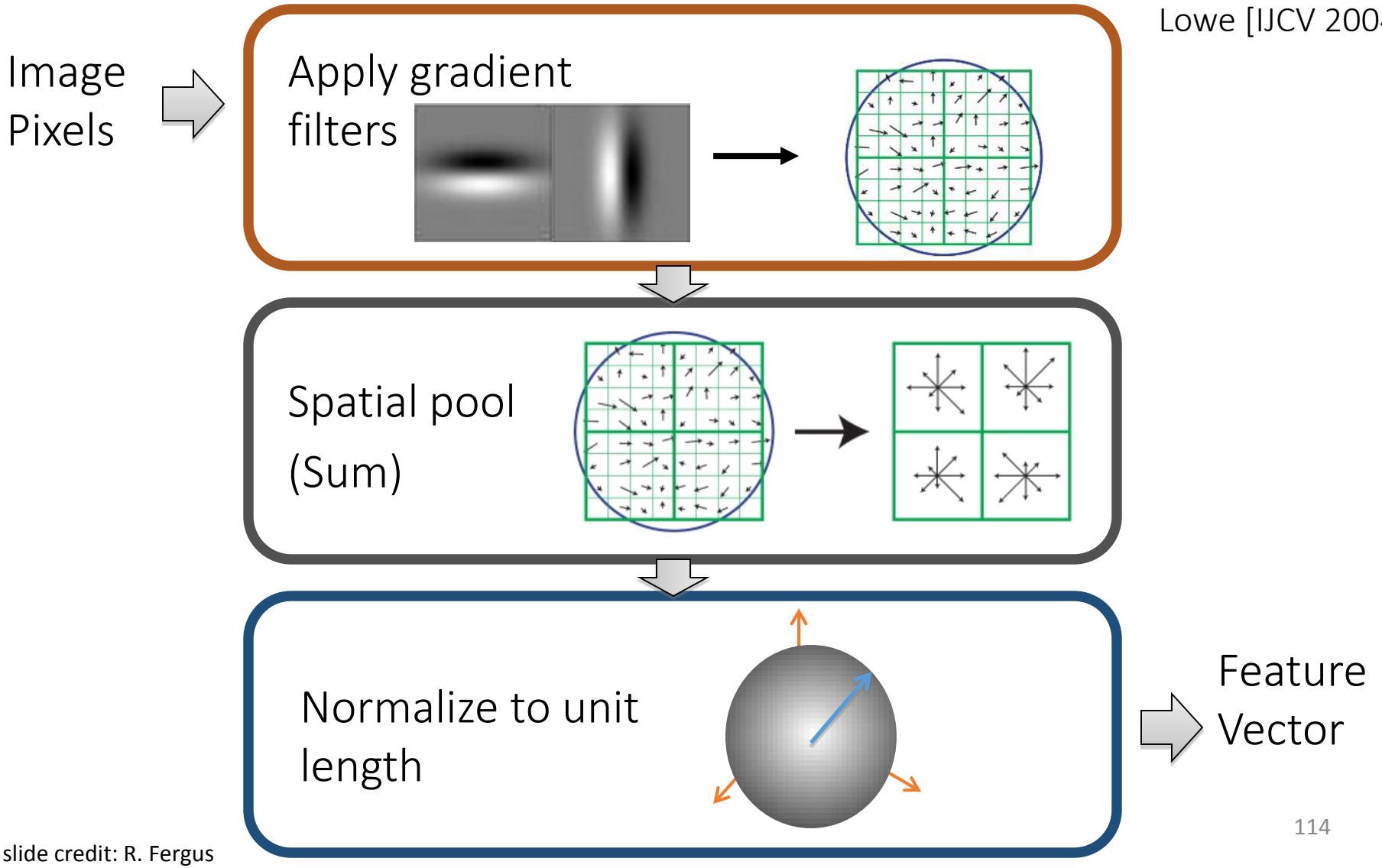


CNN


INPUT
$32 \times 32 \times 3$
C
Convolution
FEATURE MAPS
$32 \times 32 \times D$
ReLU
FEATURE MAPS
$32 \times 32 \times D$
P
Pooling
FEATURE MAPS
$16 \times 16 \times D$
F
Full Connection
OUTPUT
$1 \times 1 \times 10$


CONV
RELU
CONV
POOL
CONV
RELU
CONV
POOL
CONV
RELU
CONV
POOL
FC
car
truck
airplane
ship
horse
Figure by Andrej Karpathy

Recall: BoW for Classification

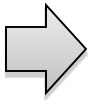


Lowe [IJCV 2004]

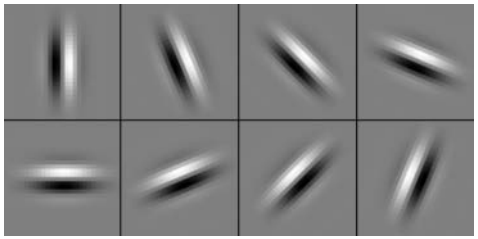
slide credit: R. Fergus

Lowe [IJCV 2004]

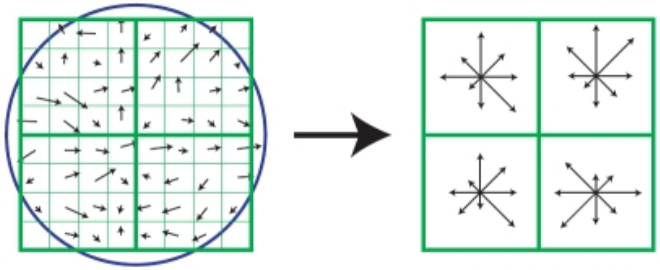
Image Pixels



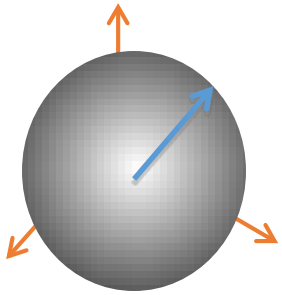
Apply oriented filters



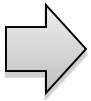
Spatial pool (Sum)



Normalize to unit length



Feature Vector

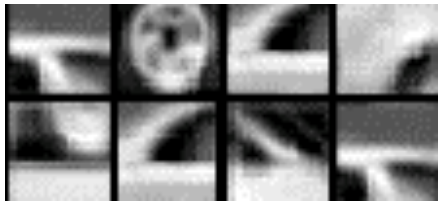


slide credit: R. Fergus

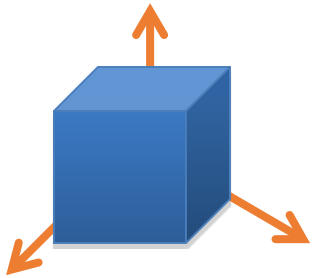
Spatial Pyramid Matching

SIFT Features

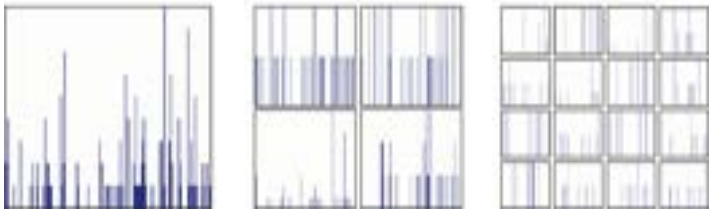
Filter with Visual Words



Max



Multi-scale spatial pool (Sum)

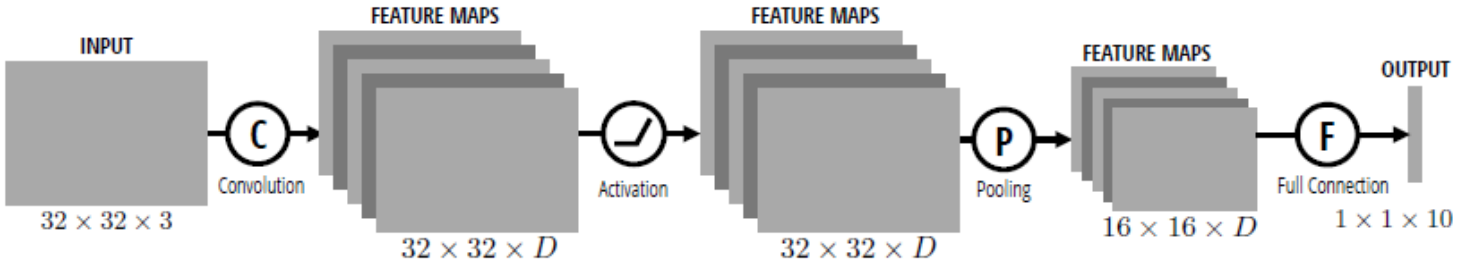


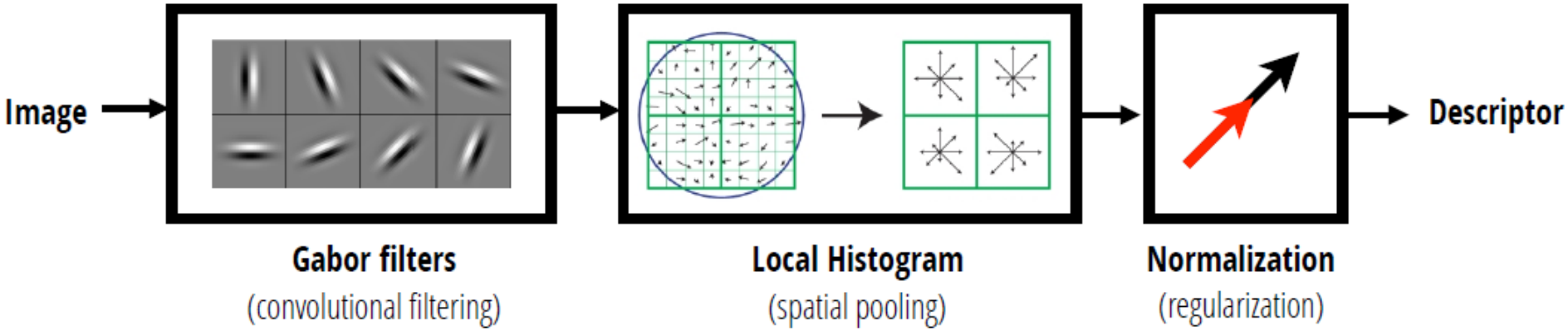
Classifier

Lazebnik, Schmid, Ponce [CVPR 2006]

slide credit: R. Fergus

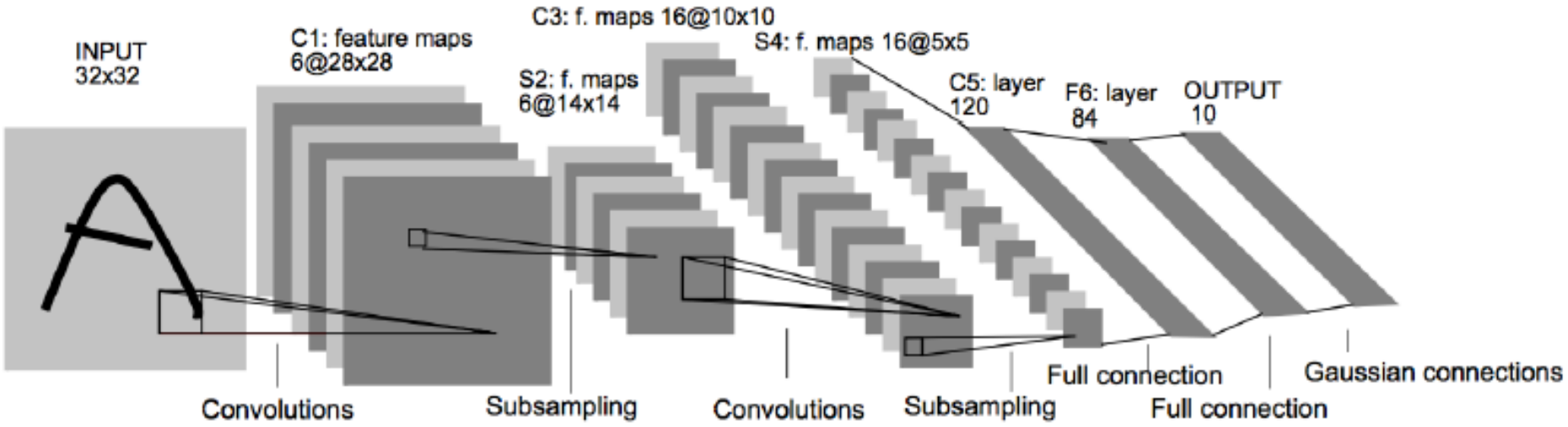
CNN vs. BoW Pipeline





LeNet

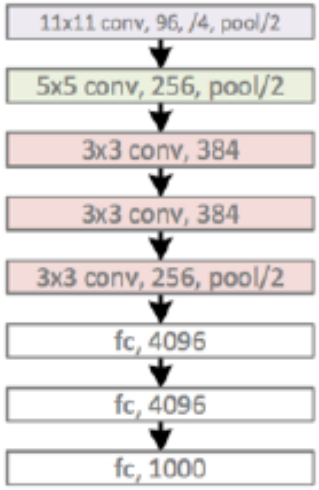
- Presented by Yann LeCun during the 1990s for reading digits
- Has the elements of modern architectures

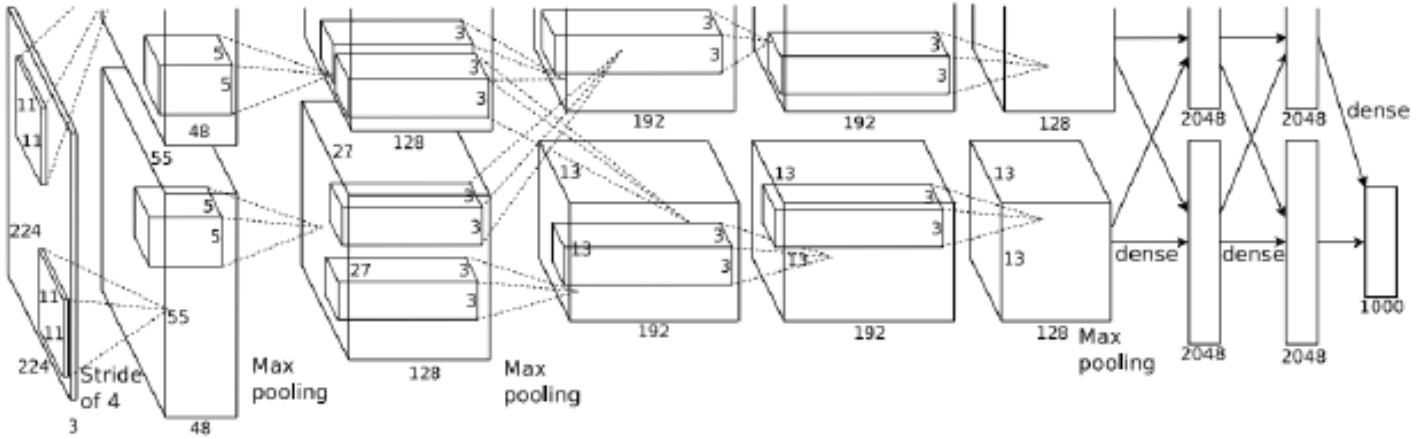


AlexNet [Krizhevsky et al., 2012]

- 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)





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!

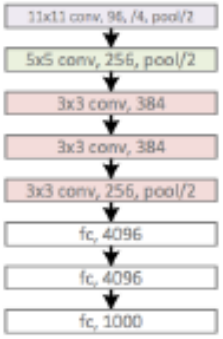
Removing Layer 6 and 7 reduces 50 million parameters, but only 5.7% drop in performance

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

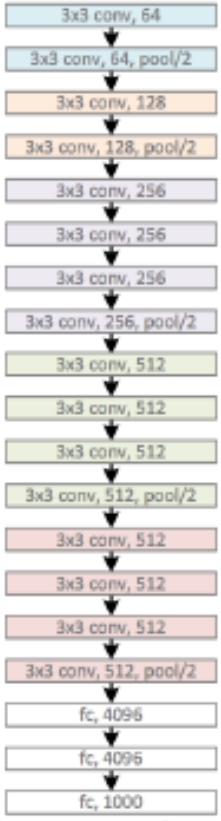
Removing feature & conv layers produces a **33% drop** in performance

CNN: A Revolution of Depth

AlexNet, 8 layers
(ILSVRC 2012)



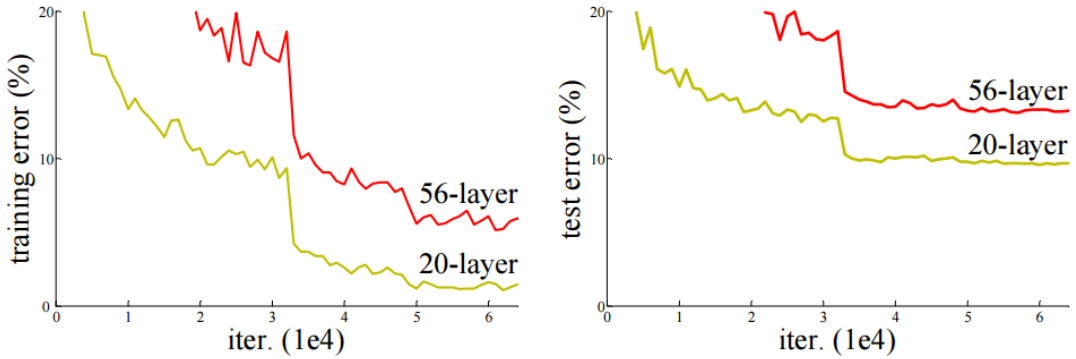
VGG, 19 layers
(ILSVRC 2014)



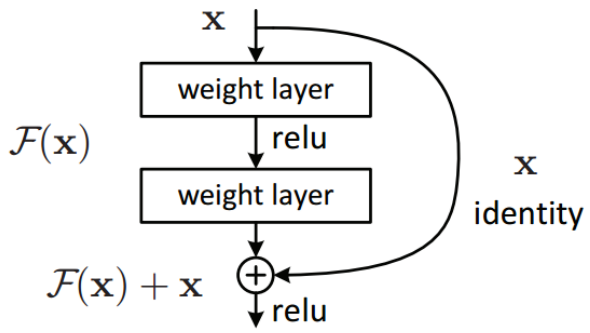
GoogleNet, 22 layers
(ILSVRC 2014)

ResNet

- Can we just increase the #layer?



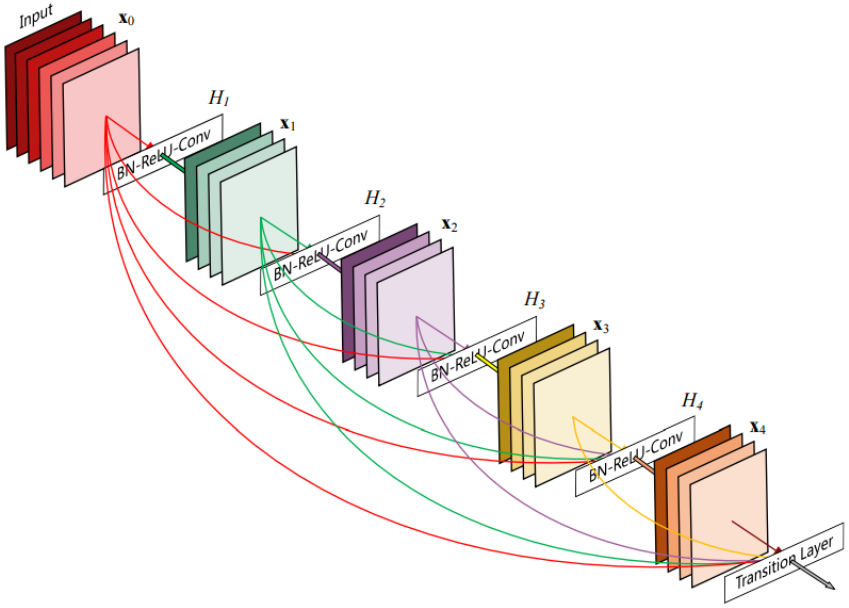
- How can we train very deep network?
  - Residual learning

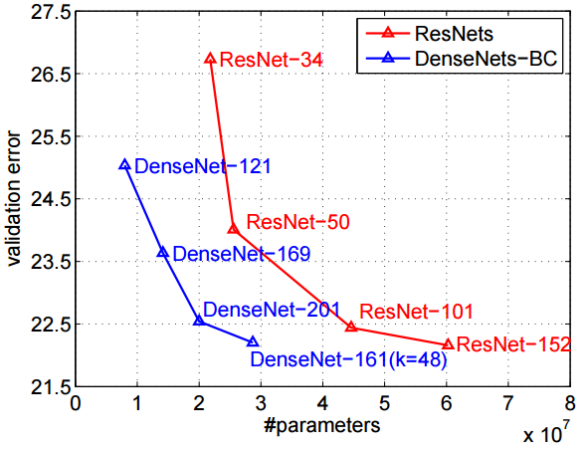


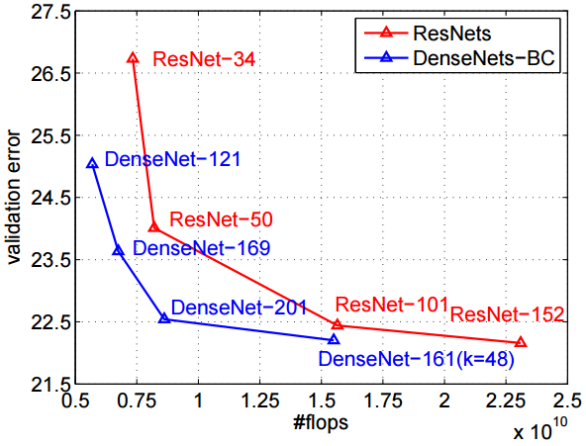
method	top-5 err. (**test**)
VGG [41] (ILSVRC'14)	7.32
GoogLeNet [44] (ILSVRC'14)	6.66
VGG [41] (v5)	6.8
PReLU-net [13]	4.94
BN-inception [16]	4.82
ResNet (ILSVRC'15)	**3.57**

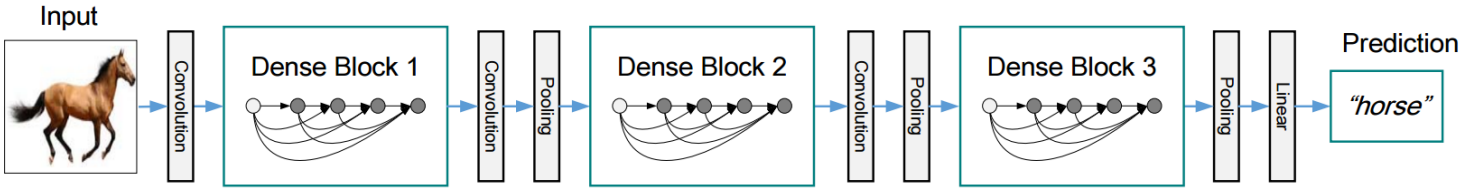
DenseNet

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

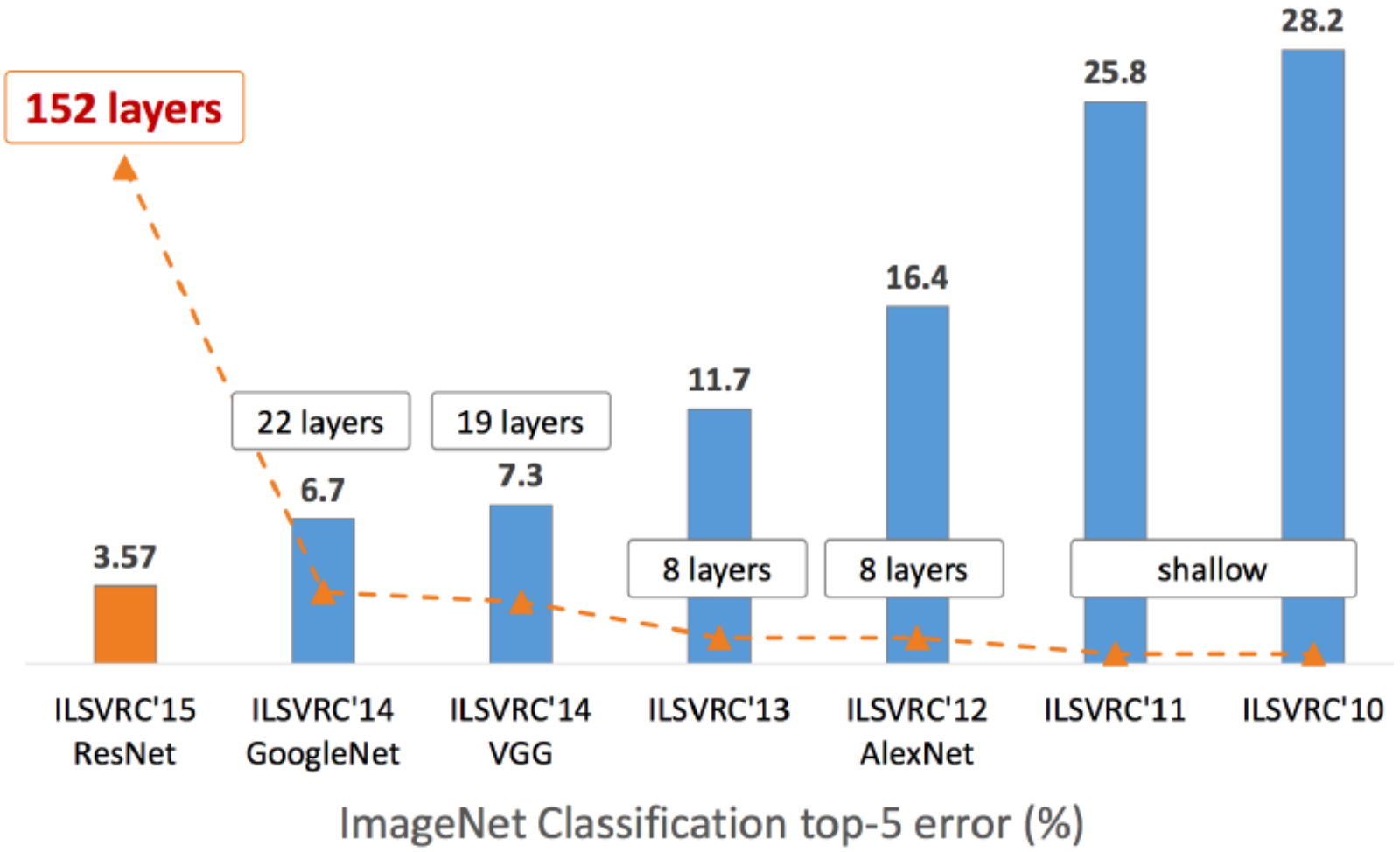








CNN: A Revolution of Depth



Remarks

- CNN:
  - Reduce the number of parameters
  - Reduce the memory requirements
  - Make computation independent of the size of the image
- Neuroscience provides strong inspiration on the NN design.
- Neuroscience provides little guidance on how to train CNNs.
- Only a few structures discussed: convolution, nonlinearity, pooling

Training Convolutional Neural Networks

- Backpropagation +
  stochastic gradient descent with momentum
  - Neural Networks: Tricks of the Trade
- Dropout
- Data augmentation
- Batch normalization

Training CNN with gradient descent

- A CNN as composition of functions

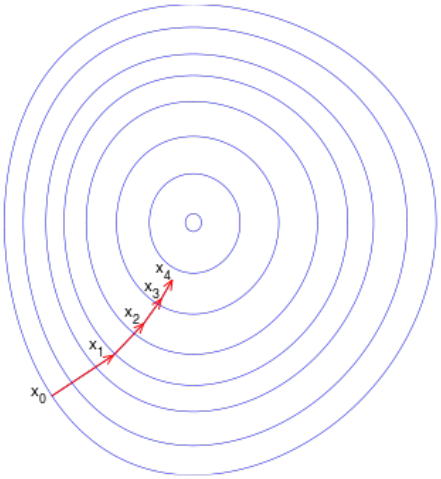
$$f_{\boldsymbol{w}}(\boldsymbol{x}) = f_L(\dots\ (f_2(f_1(\boldsymbol{x}; \boldsymbol{w}_1); \boldsymbol{w}_2)\ \dots; \boldsymbol{w}_L)$$

- Parameters

$$\boldsymbol{w} = (\boldsymbol{w_1}, \boldsymbol{w_2}, \dots \boldsymbol{w_L})$$

- Empirical loss function

$$L(\boldsymbol{w}) = \frac{1}{n}\sum_i l(z_i, f_{\boldsymbol{w}}(\boldsymbol{x_i}))$$



- Gradient descent

$$\boldsymbol{w^{t+1}} = \boldsymbol{w^t} - \eta_t \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{w}}(\boldsymbol{w^t})$$

New weight

Old weight

Learning rate

Gradient

An Illustrative Example

$$f(x, y) = xy, \qquad \frac{\partial f}{\partial x} = y, \frac{\partial f}{\partial y} = x$$

Example: $x = 4, y = -3 \Rightarrow f(x, y) = -12$

Partial derivatives

$$\frac{\partial f}{\partial x} = -3, \qquad \frac{\partial f}{\partial y} = 4$$

Gradient

$$\nabla f = [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}]$$

Example credit: Andrej Karpathy

$$f(x, y, z) = (x + y)z = qz$$

$q = x + y$

$\frac{\partial q}{\partial x} = 1, \qquad \frac{\partial q}{\partial y} = 1$

$f = qz$

$\frac{\partial f}{\partial q} = z, \qquad \frac{\partial f}{\partial z} = q$

Goal: compute the gradient

$$\nabla f = [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}]$$

Example credit: Andrej Karpathy

$$f(x, y, z) = (x + y)z = qz$$

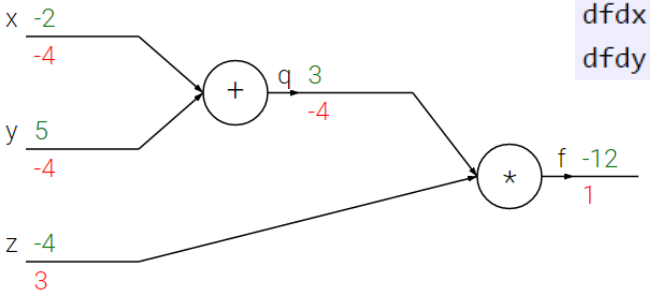
$$q = x + y$$
$$\frac{\partial q}{\partial x} = 1, \qquad \frac{\partial q}{\partial y} = 1$$

$$f = qz$$
$$\frac{\partial f}{\partial q} = z, \qquad \frac{\partial f}{\partial z} = q$$

Chain rule:

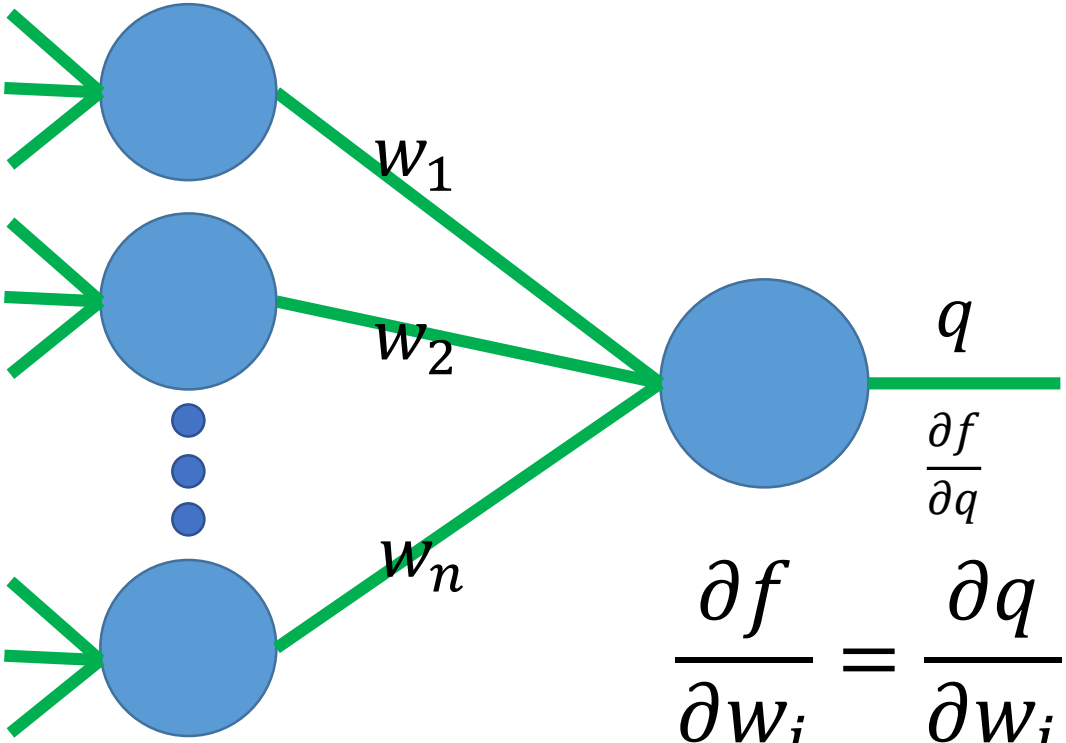
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

```
# set some inputs
x = -2; y = 5; z = -4

# perform the forward pass
q = x + y # q becomes 3
f = q * z # f becomes -12

# perform the backward pass (backpropagation) in reverse order:
# first backprop through f = q * z
dfdz = q # df/dz = q, so gradient on z becomes 3
dfdq = z # df/dq = z, so gradient on q becomes -4
# now backprop through q = x + y
dfdx = 1.0 * dfdq # dq/dx = 1. And the multiplication here is the chain rule!
dfdy = 1.0 * dfdq # dq/dy = 1
```



Example credit: Andrej Karpathy

Backpropagation (recursive chain rule)



$$\frac{\partial f}{\partial w_i} = \frac{\partial q}{\partial w_i}\frac{\partial f}{\partial q}$$

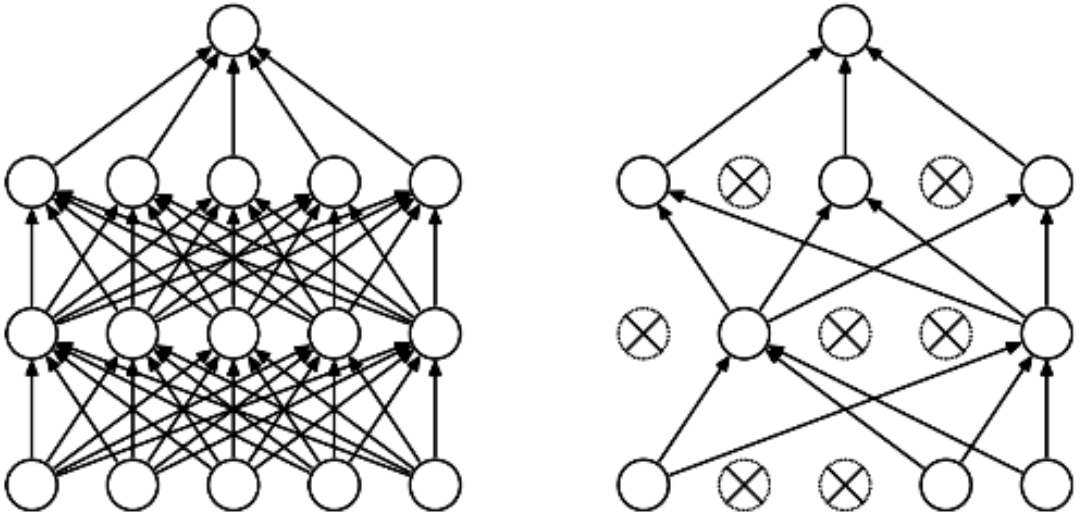
Local gradient

Gate gradient

Can be computed during forward pass

The gate receives this during backprop

Dropout



(a) Standard Neural Net

(b) After applying dropout.

Intuition: successful conspiracies

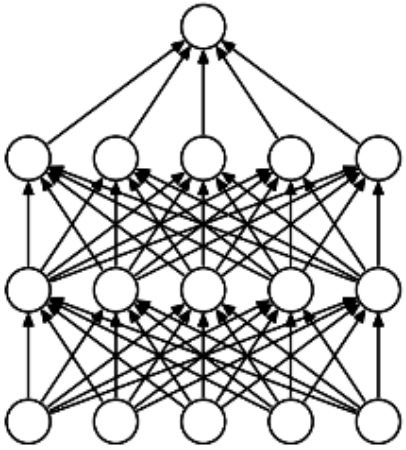
- 50 people planning a conspiracy

- Strategy A: plan a big conspiracy involving 50 people
  - Likely to fail. 50 people need to play their parts correctly.

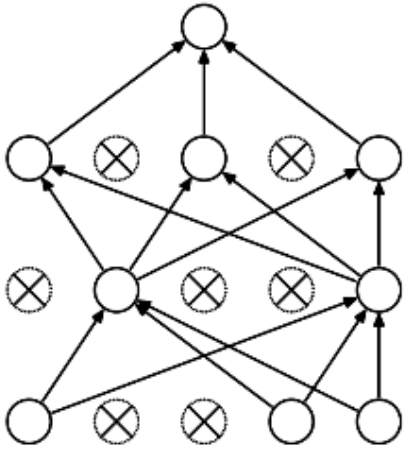
- Strategy B: plan 10 conspiracies each involving 5 people
  - Likely to succeed!

Dropout: A simple way to prevent neural networks from overfitting [Srivastava JMLR 2014]

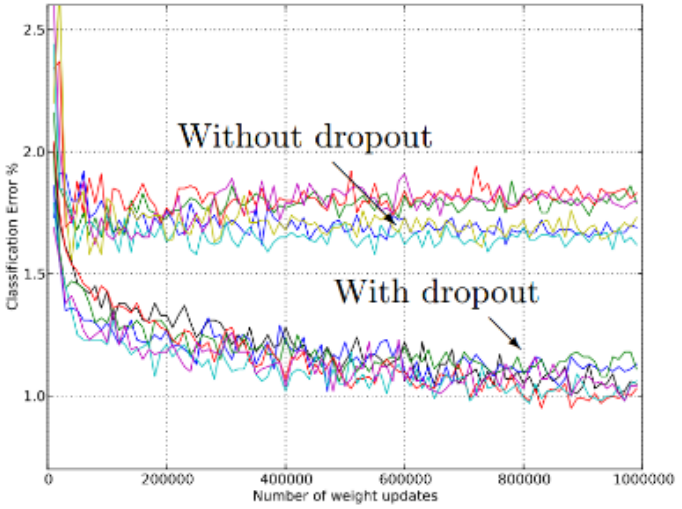
Dropout

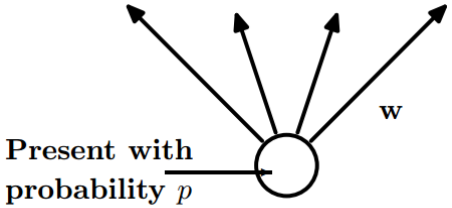


(a) Standard Neural Net

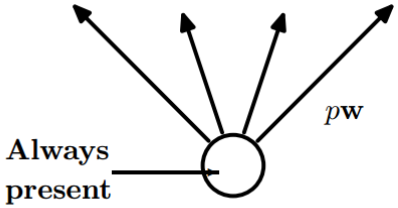


(b) After applying dropout.





(a) At training time



(b) At test time

Main Idea: approximately combining exponentially many different neural network architectures efficiently

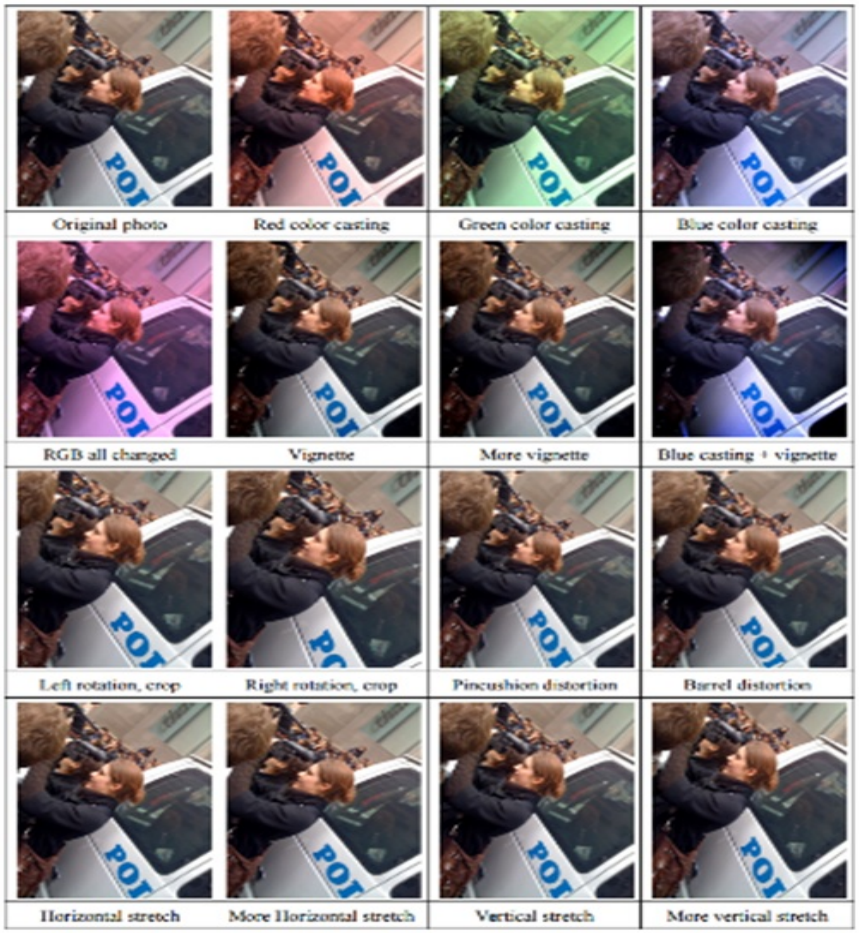
Model	Top-1 (val)	Top-5 (val)	Top-5 (test)
SVM on Fisher Vectors of Dense SIFT and Color Statistics	-	-	27.3
Avg of classifiers over FVs of SIFT, LBP, GIST and CSIFT	-	-	26.2
Conv Net + dropout (Krizhevsky et al., 2012)	40.7	18.2	-
Avg of 5 Conv Nets + dropout (Krizhevsky et al., 2012)	38.1	16.4	16.4

Table 6: Results on the ILSVRC-2012 validation/test set.

Dropout: A simple way to prevent neural networks from overfitting [Srivastava JMLR 2014]

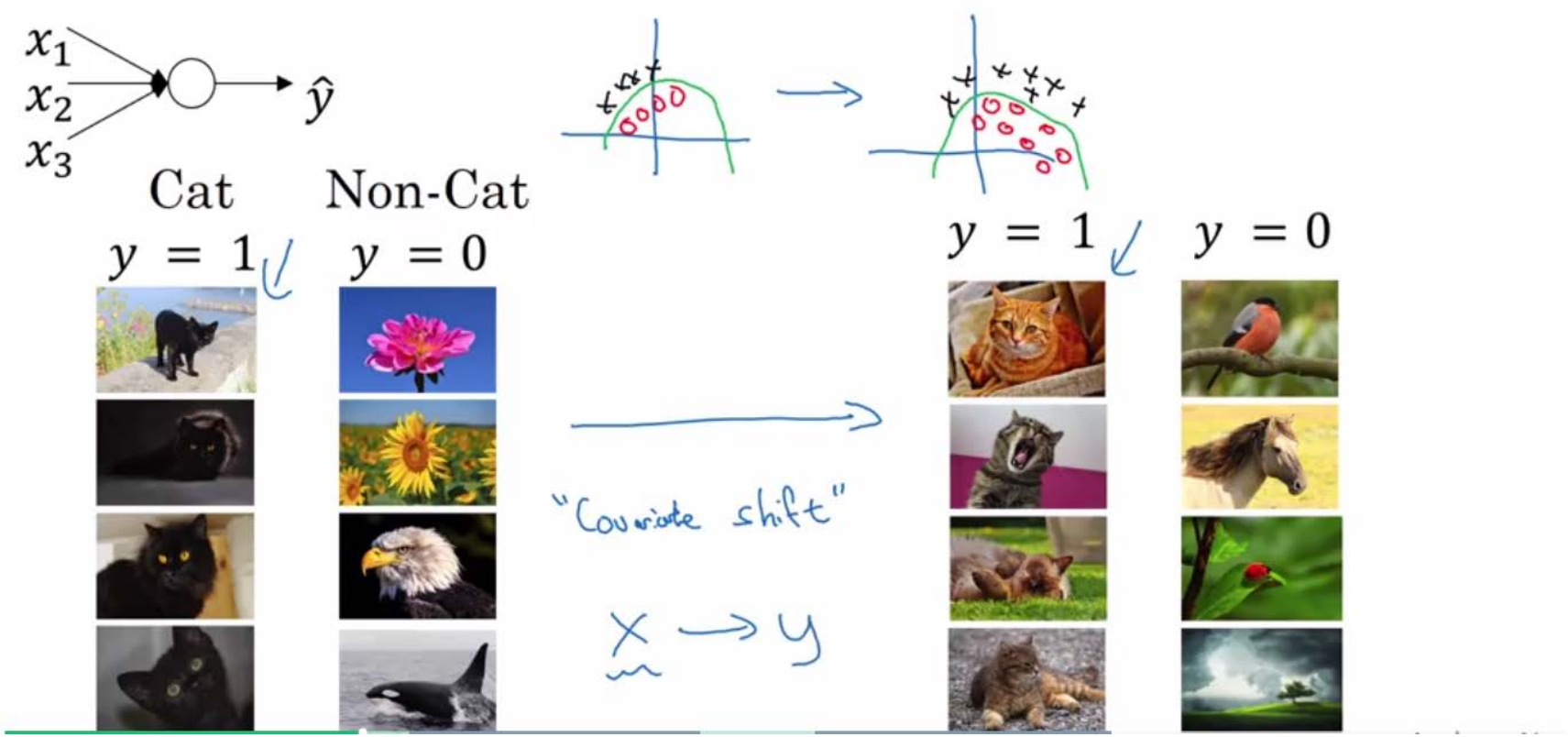
Data Augmentation (Jittering)

- Create *virtual* training samples
  - Horizontal flip
  - Random crop
  - Color casting
  - Geometric distortion



Deep Image [Wu et al. 2015]

Batch Normalization



Credit: Andrew Ng

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

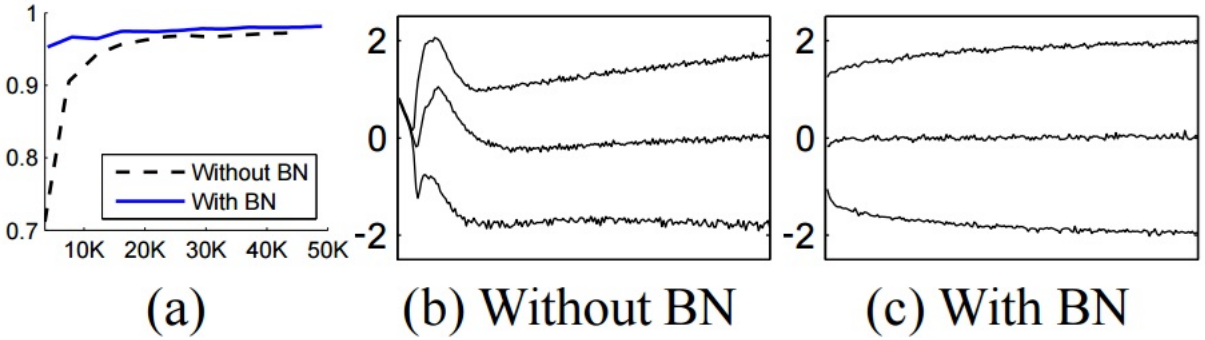
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$



(a) (b) Without BN (c) With BN

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift [Ioffe and Szegedy 2015]

Resources

- http://deeplearning.net/
  - Hub to many other deep learning resources
- https://github.com/ChristosChristofidis/awesome-deep-learning
  - A resource collection deep learning
- https://github.com/kjw0612/awesome-deep-vision
  - A resource collection deep learning for computer vision
- http://cs231n.stanford.edu/syllabus.html
  - Nice course on CNN for visual recognition
- http://deeplearning.ai
  - Lots of online course videos by Andrew Ng

What We've Learned Today…

- Supervised Learning: Linear Classification
- Deep Learning Basics
  - Neural Network for Machine Vision
  - Multi-Layer Perceptron
  - Convolutional Neural Networks