Deep Learning Basics

簡韶逸 Shao-Yi Chien
Department of Electrical Engineering
National Taiwan University

References and Slide Credits

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

Outline

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

History of Neural Network and Deep Learning

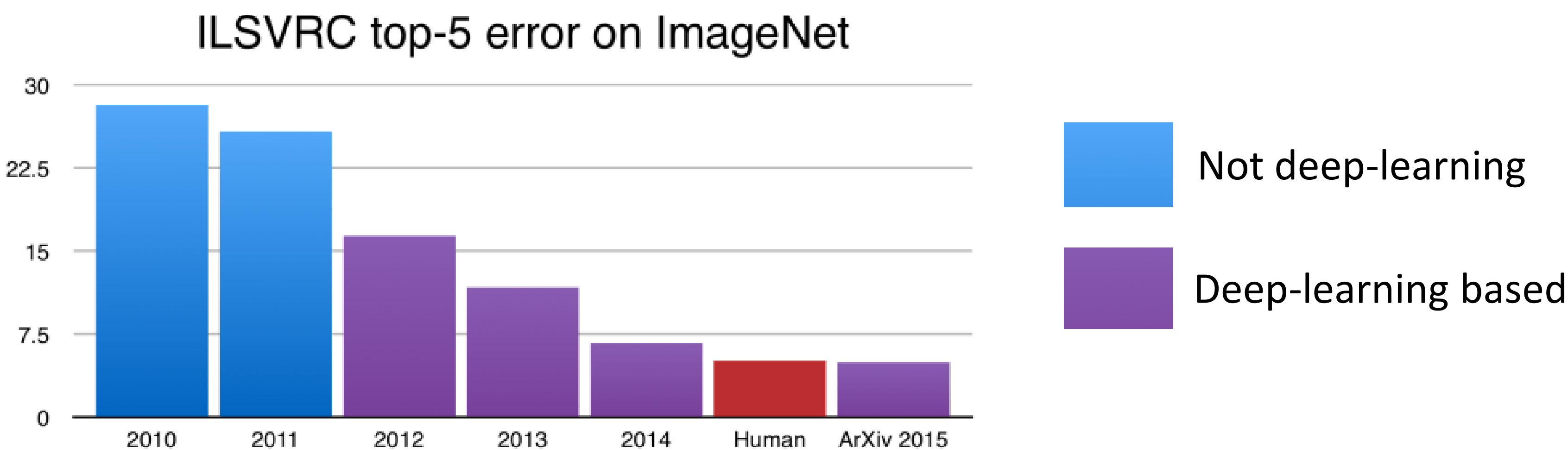
[Prof. Hung-Yi Lee]

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

Geoffrey Hinton

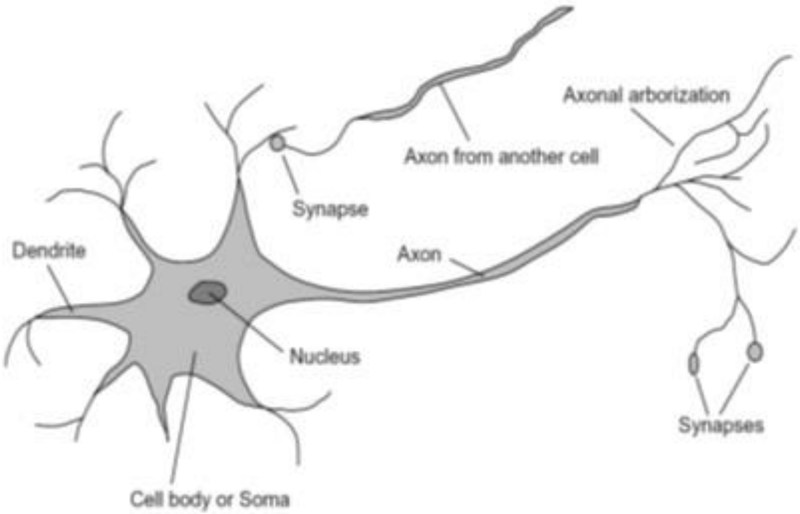
LeCun, Yann; Bengio, Yoshua; Hinton, Geoffrey, "Deep learning," *Nature*, 2015.

How Powerful?
Object Recognition

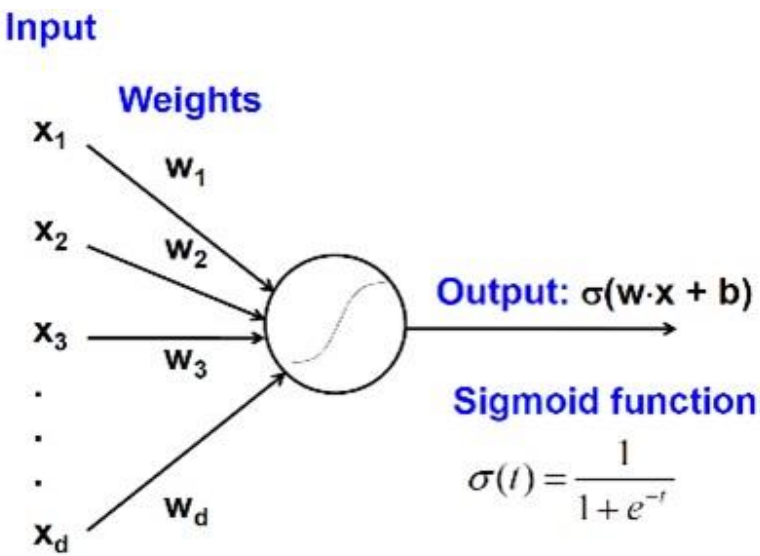


Source:
https://devblogs.nvidia.com/parallelforall/mocha-jl-deep-learning-julia/
https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/

Biological neuron and Perceptrons



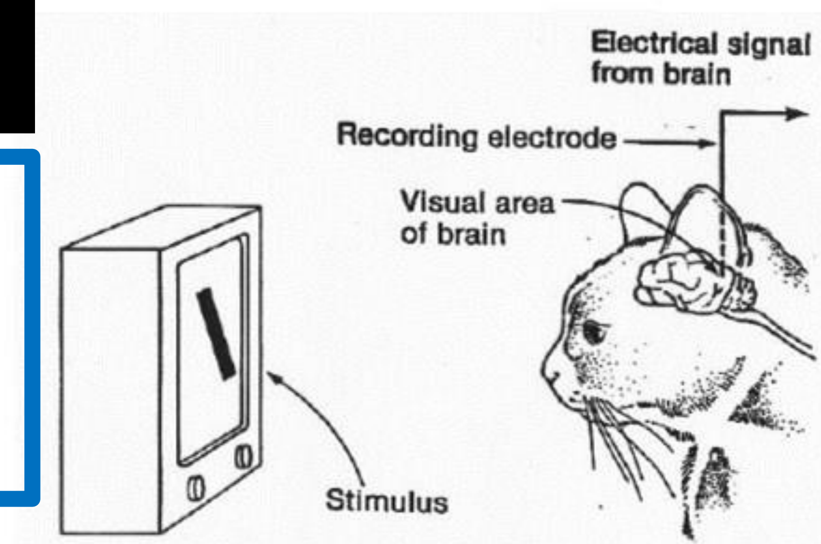
A biological neuron



An artificial neuron (Perceptron)
- a linear classifier

Simple, Complex and Hypercomplex cells

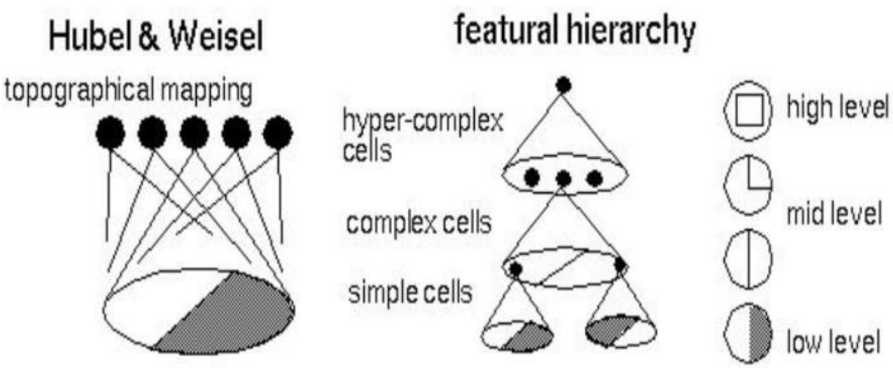
David H. Hubel and Torsten Wiesel



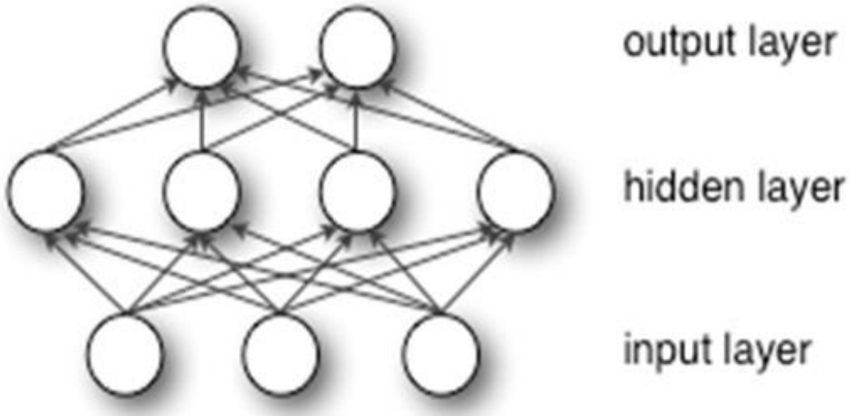
Suggested a **hierarchy** of **feature detectors** in the visual cortex, with higher level features responding to patterns of activation in lower level cells, and propagating activation upwards to still higher level cells.

David Hubel's Eye, Brain, and Vision

Hubel/Wiesel Architecture and Multi-layer Neural Network



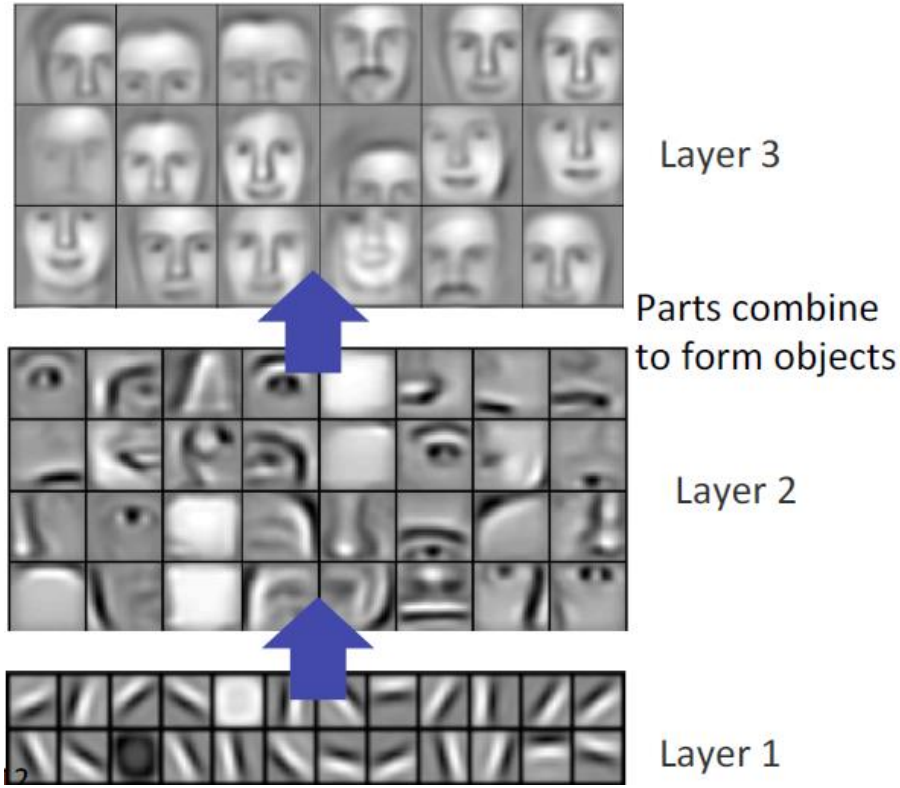
Hubel and Weisel's architecture

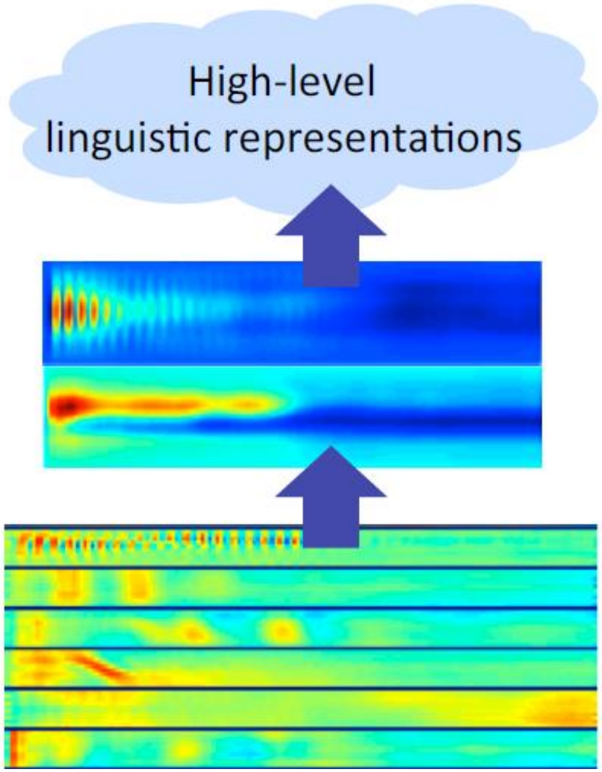


Multi-layer Neural Network
- A *non-linear* classifier

Hierarchical Representation Learning

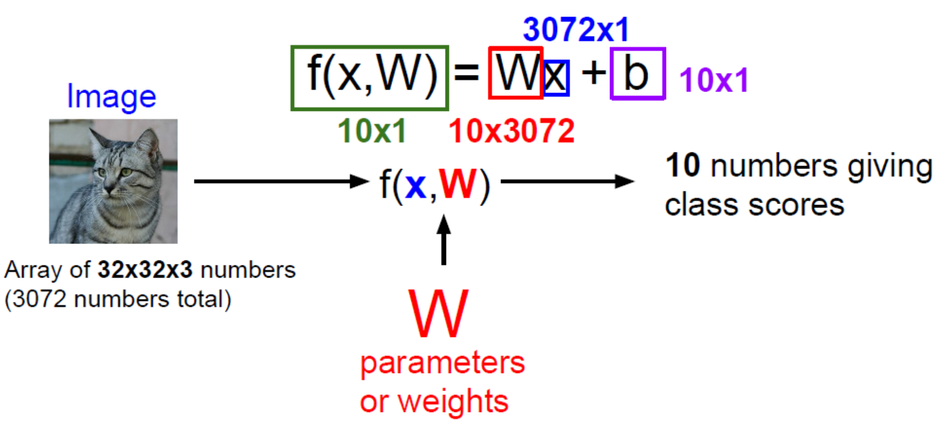
- Successive model layers learn deeper intermediate representations.





Recap: Linear Classification

- Linear Classifier
  - Let's take the input image as **x**, and the linear classifier as **W**.
    We need **y** = **Wx** + **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 **W** (plus a bias **b**),
    so that desirable outputs **y** = **Wx** + **b** can be expected.



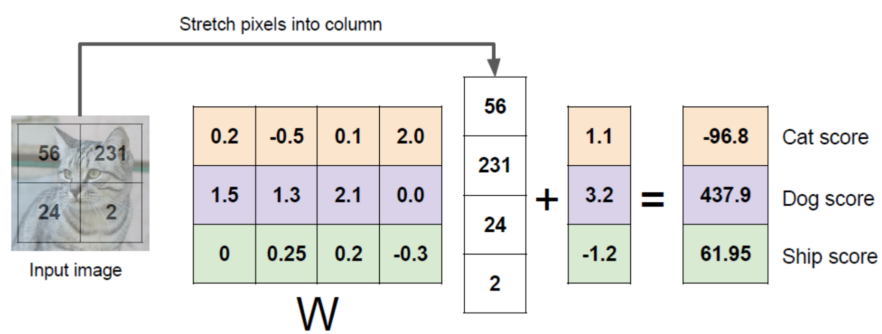
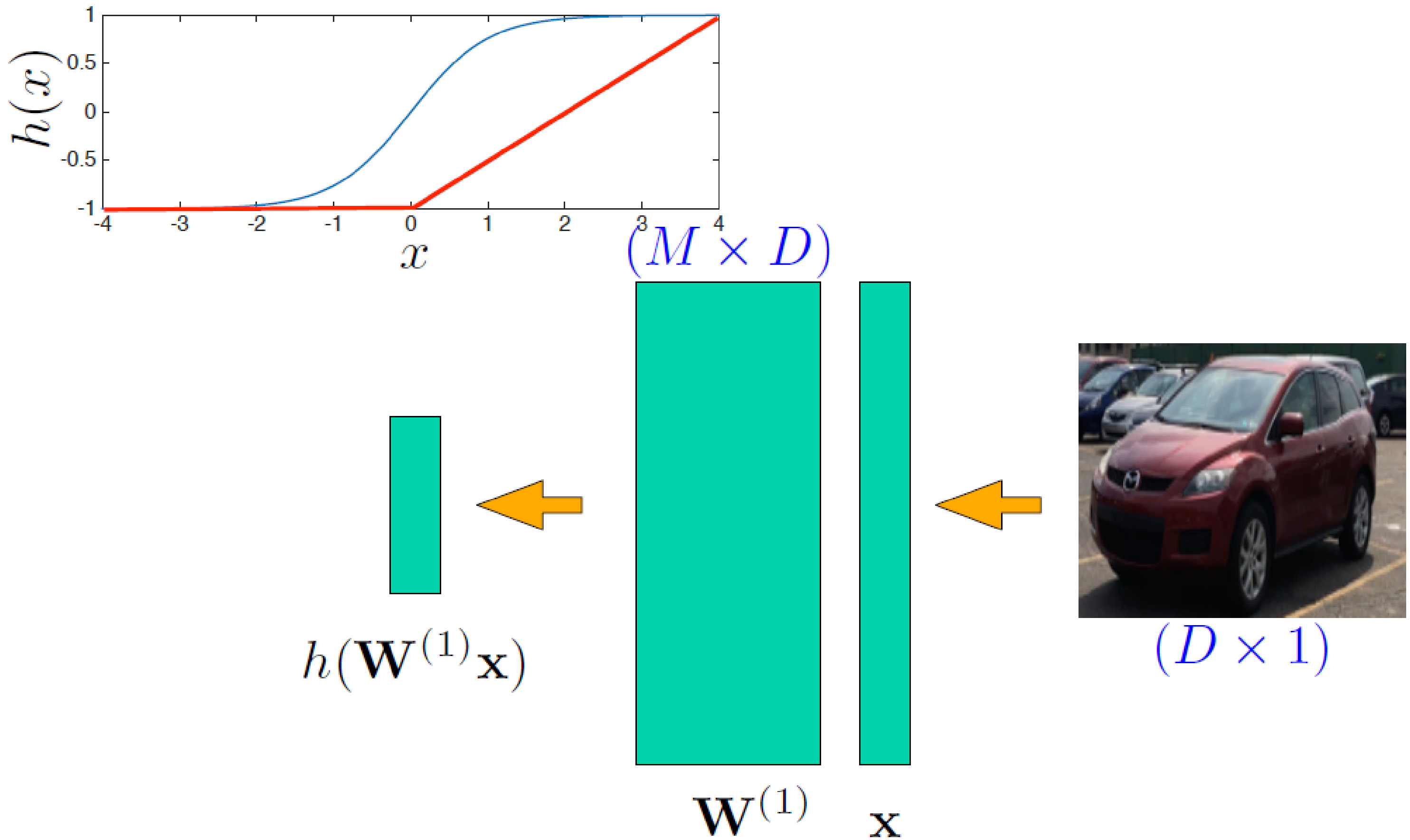
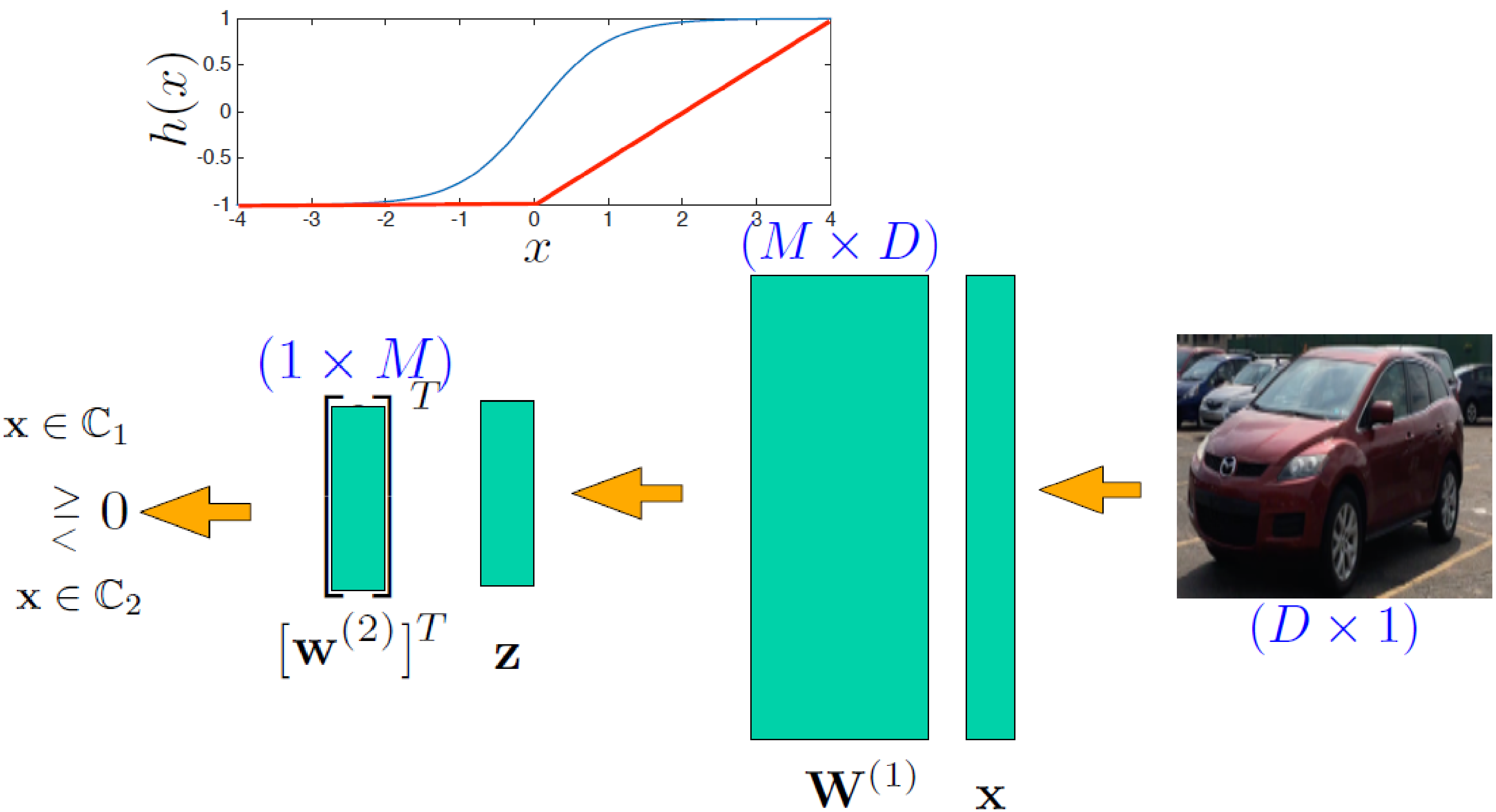


Image credit: Stanford CS231n

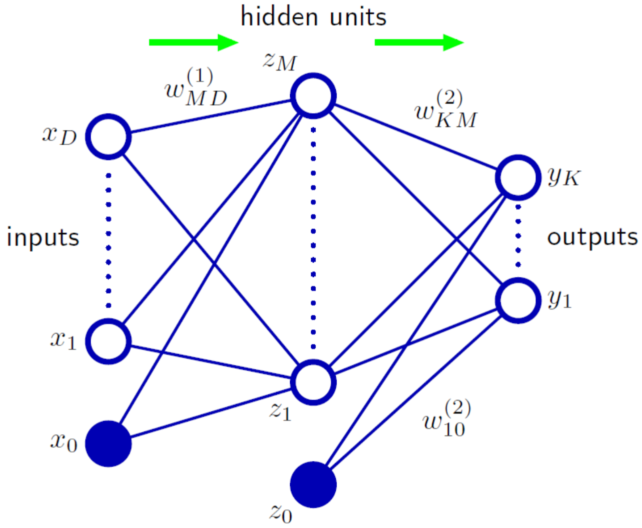
Multi-Layer Perceptron: A Nonlinear Classifier



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



Layer 1 in MLP



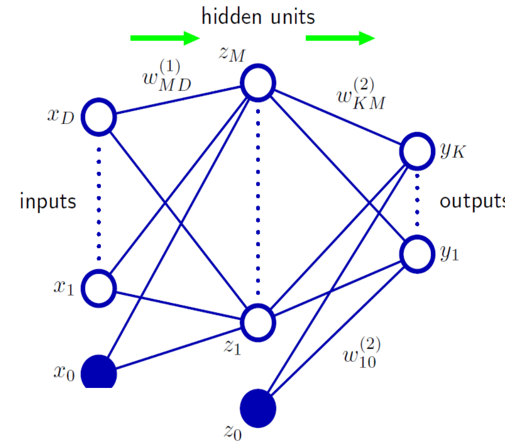
$$\mathbf{z} = \begin{bmatrix} z_1 \\ \vdots \\ z_M \end{bmatrix} \leftarrow \begin{bmatrix} h[\mathbf{x}^T \mathbf{w}_1^{(1)}] \\ \vdots \\ h[\mathbf{x}^T \mathbf{w}_M^{(1)}] \end{bmatrix}$$

$h() = \text{non-linear function}$

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

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

Layer 2 in MLP

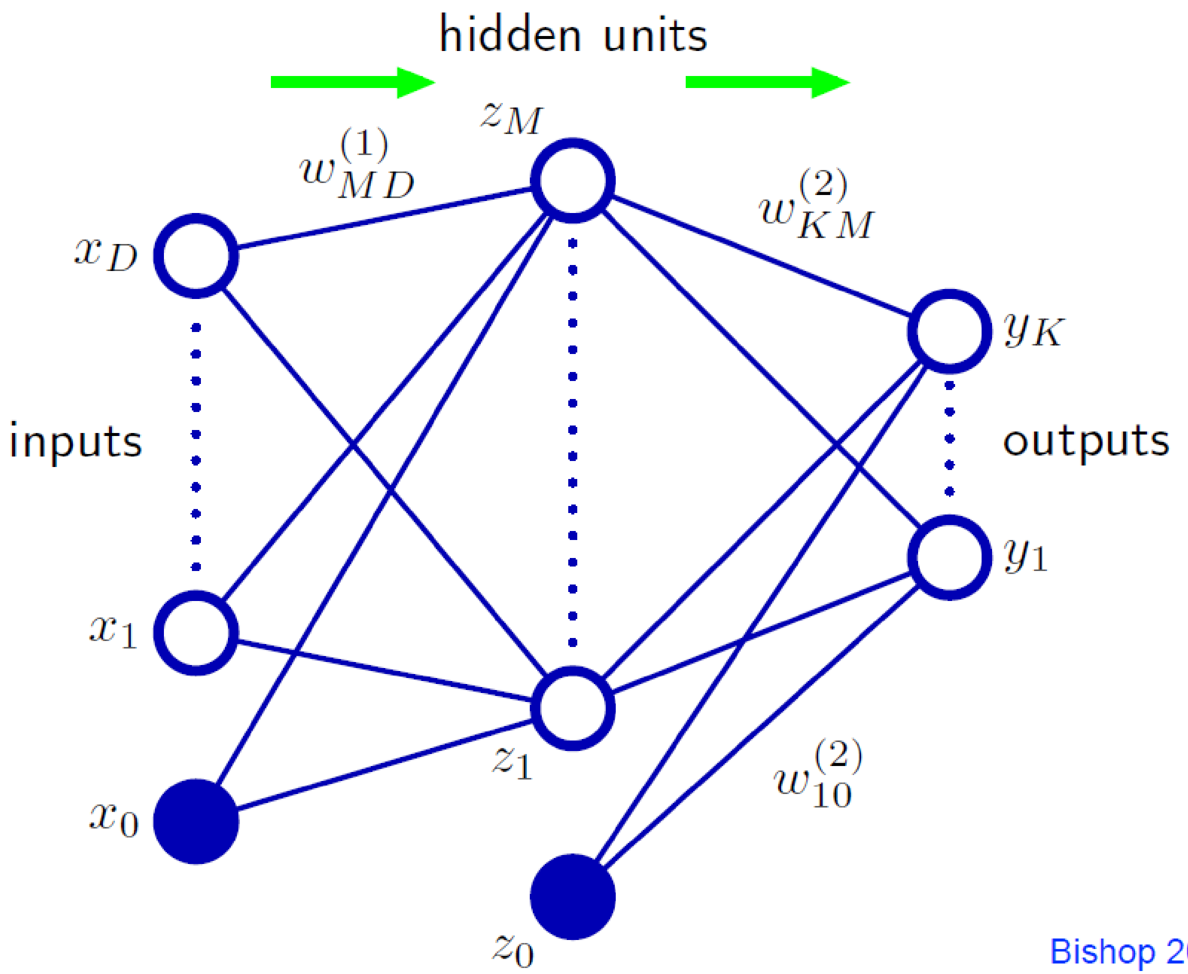


$\mathbf{x} \in \mathbb{R}^D$ $\qquad$ $\mathbf{z} \in \mathbb{R}^M$

$$\mathbf{z}^T \mathbf{w}^{(2)} \underset{\mathbf{z} \in \mathbb{C}_2}{\overset{\mathbf{z} \in \mathbb{C}_1}{\gtrless}} 0$$

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

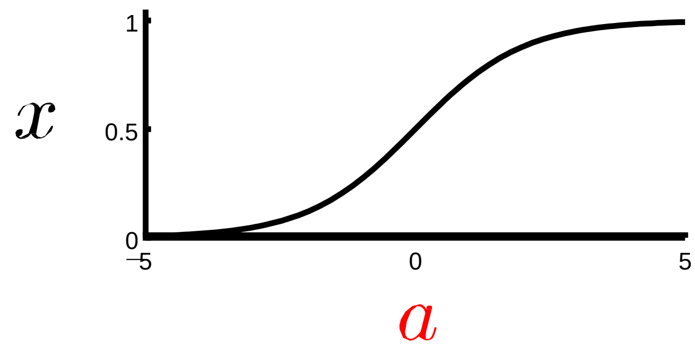
Multi-Layer Perceptron: A Nonlinear Classifier (cont'd)



Bishop 2006

Let's Get a Closer Look…

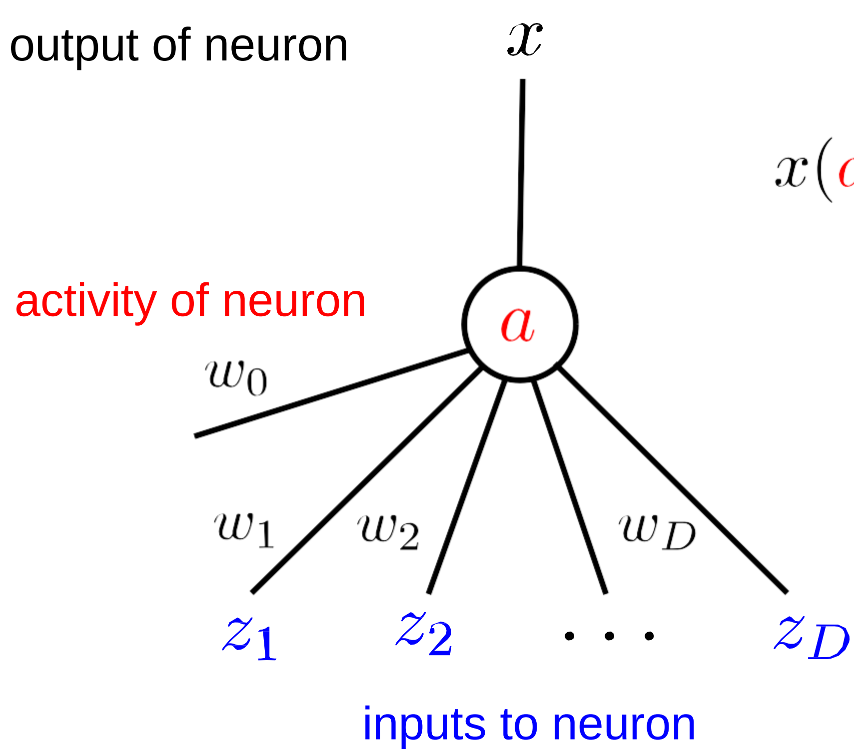
- A single neuron

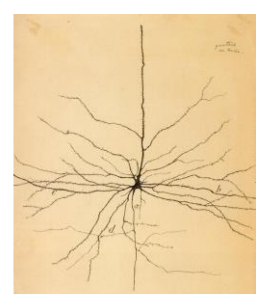


$$x(a) = \frac{1}{1+\exp(-a)} \qquad x \in (0,1)$$

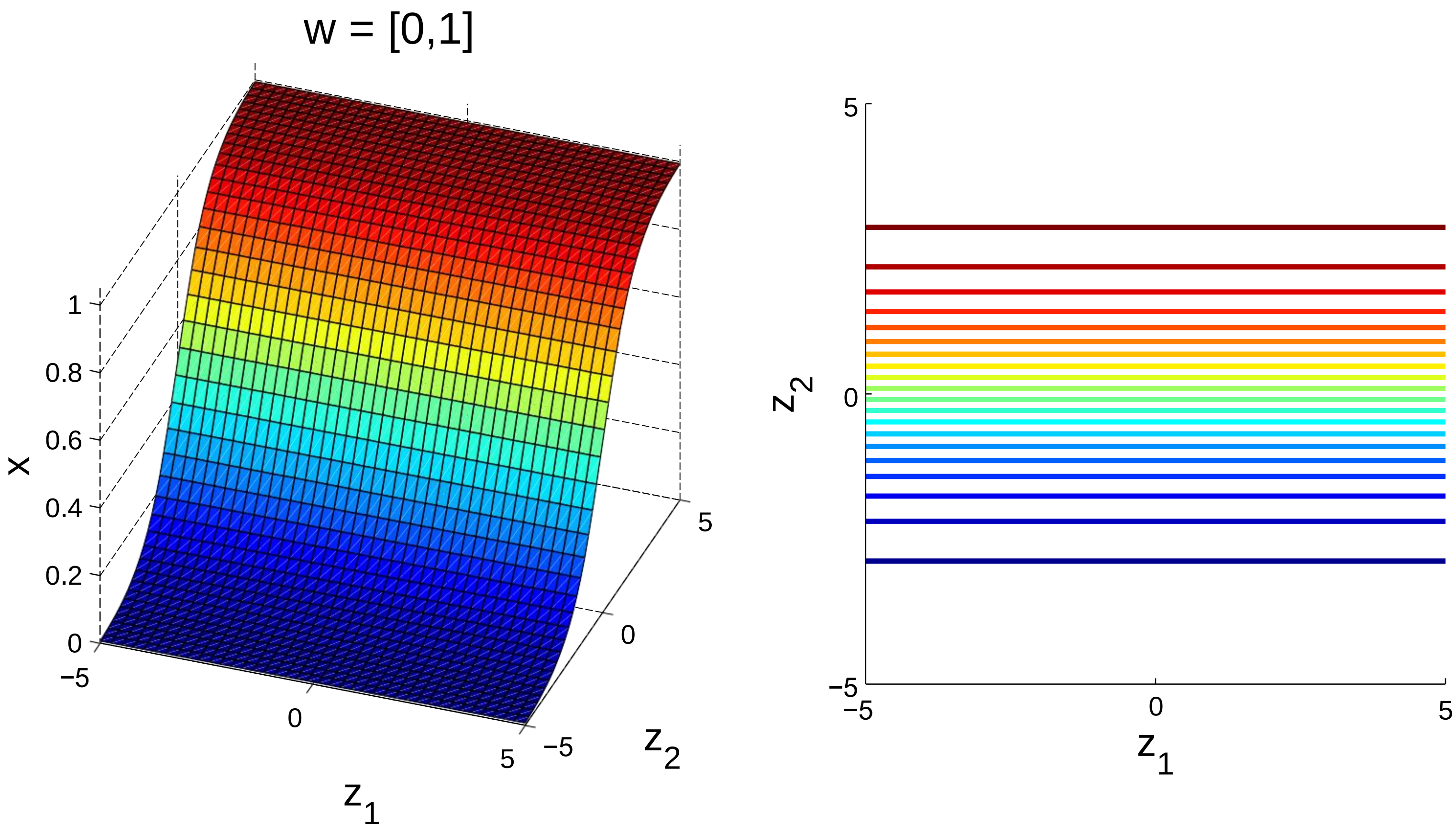
$$a = w_0 + \sum_{d=1}^{D} w_d z_d$$

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



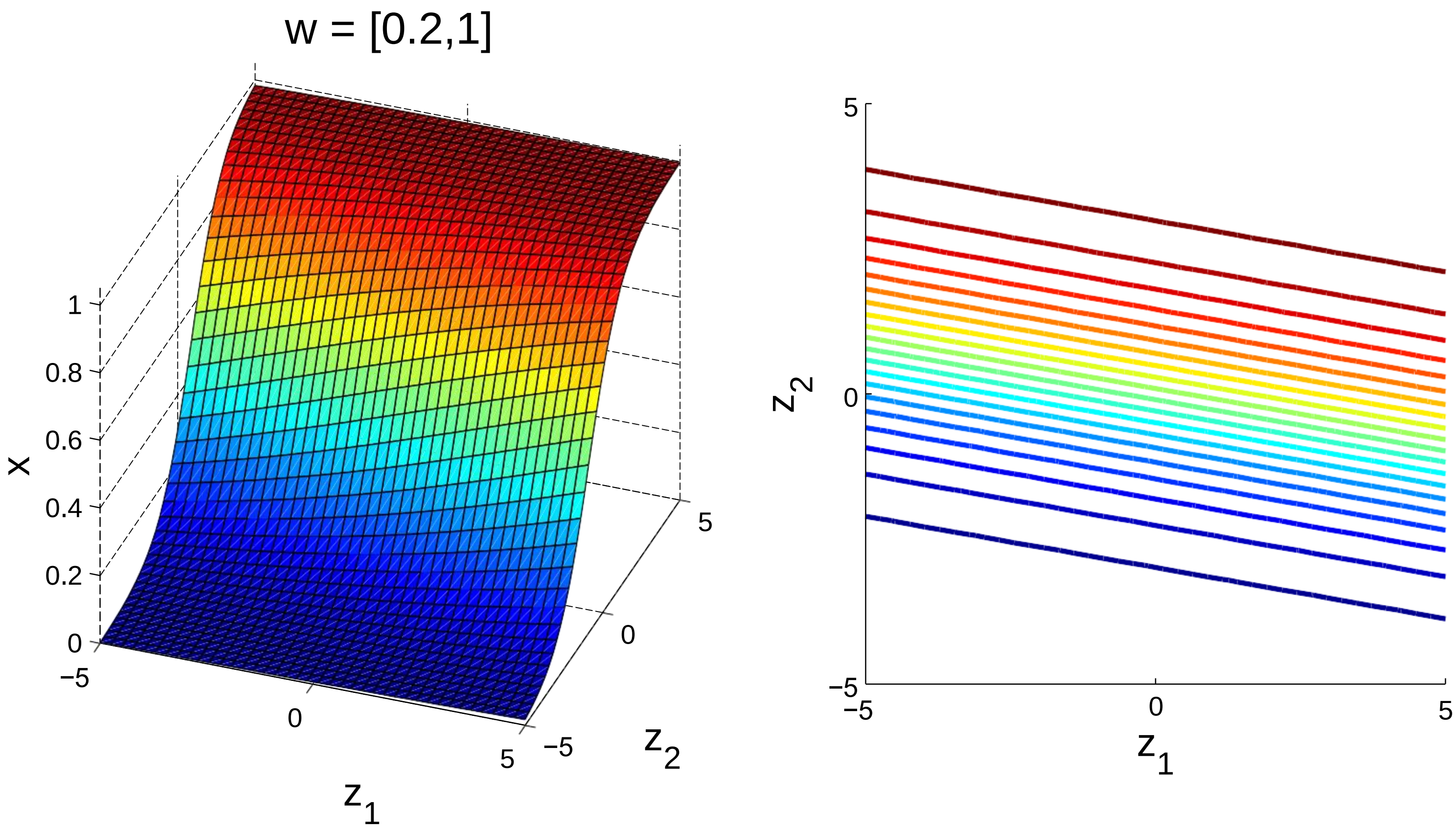


Input-Output Function of a Single Neuron



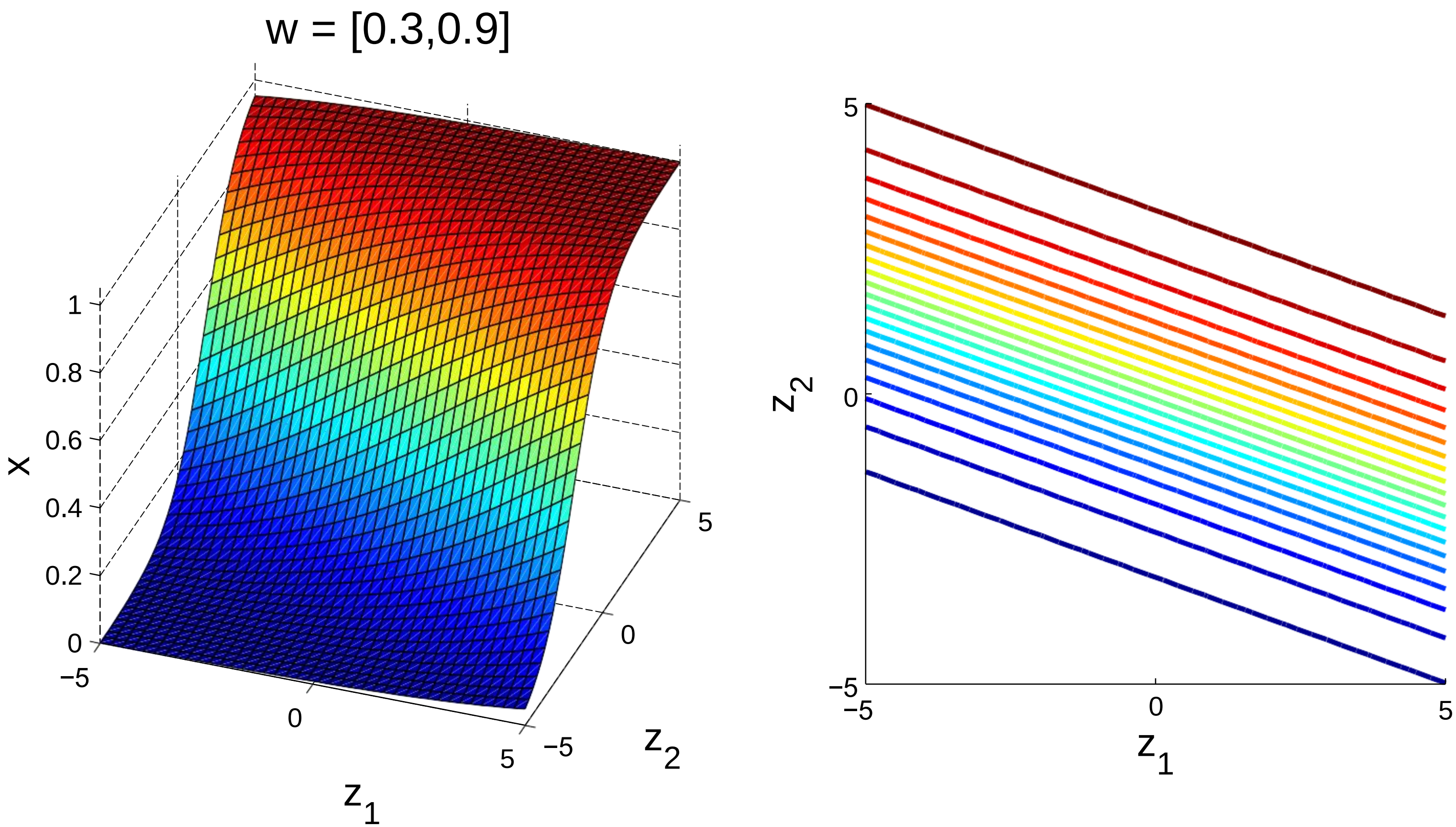
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

Input-Output Function of a Single Neuron



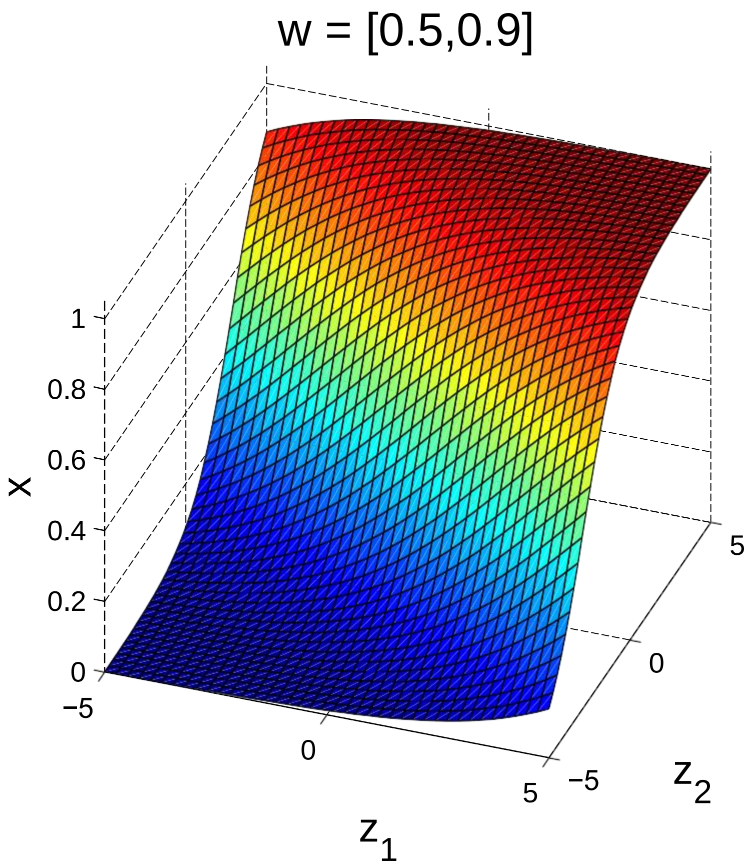
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

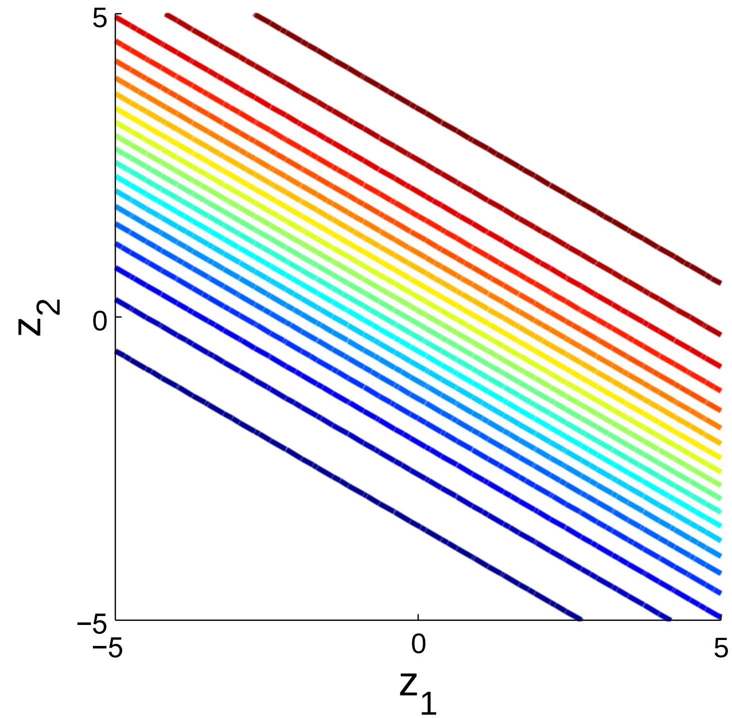
Input-Output Function of a Single Neuron



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

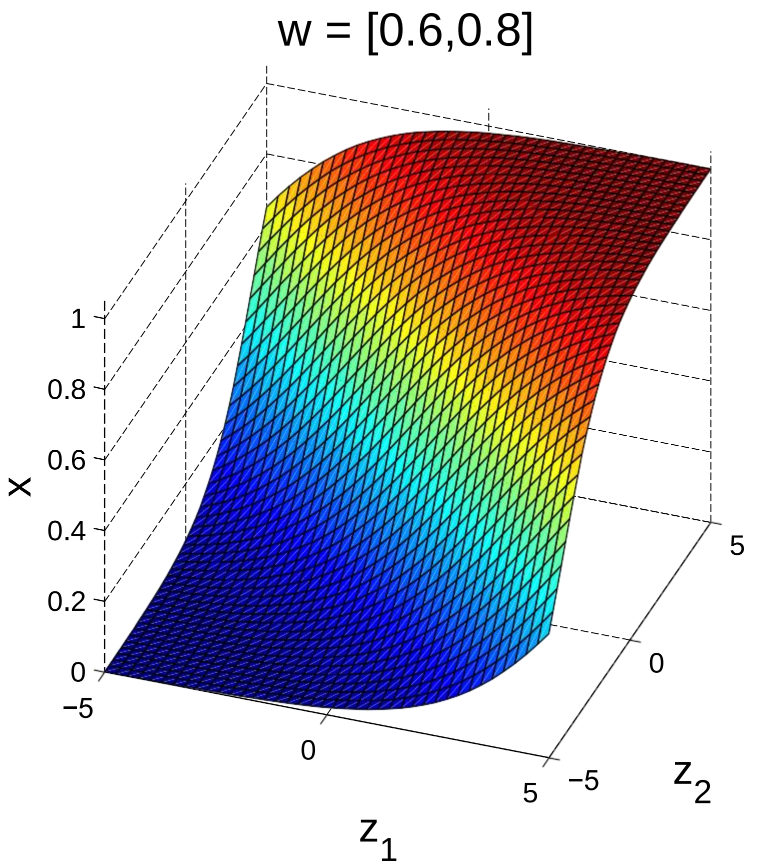
Input-Output Function of a Single Neuron

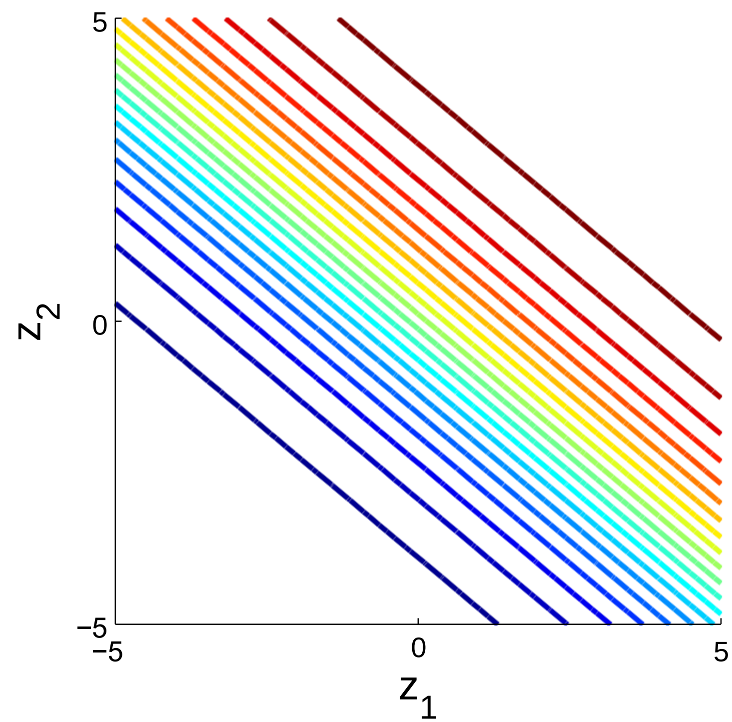




$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

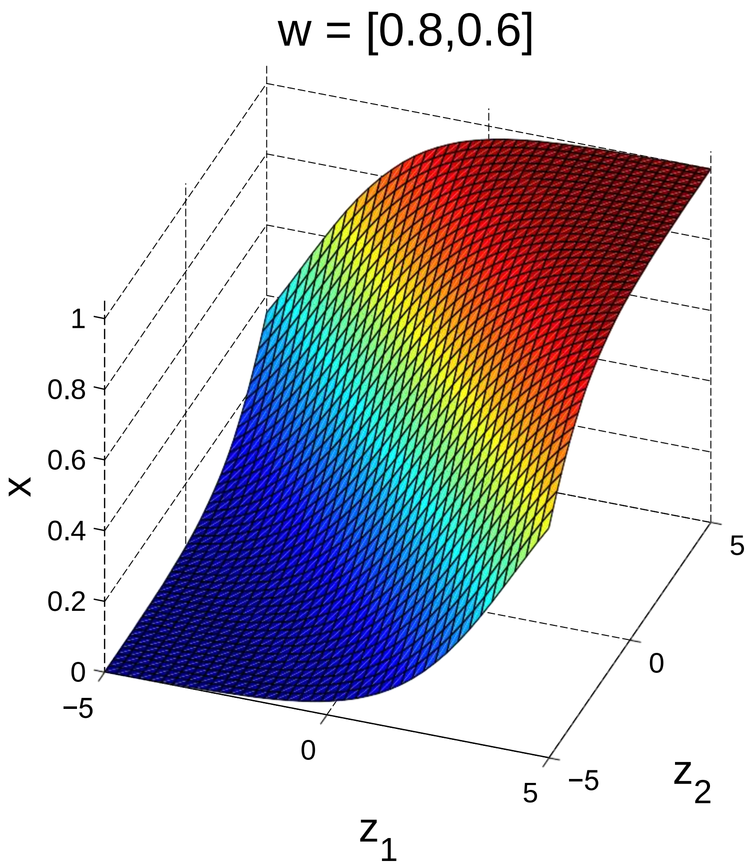
Input-Output Function of a Single Neuron

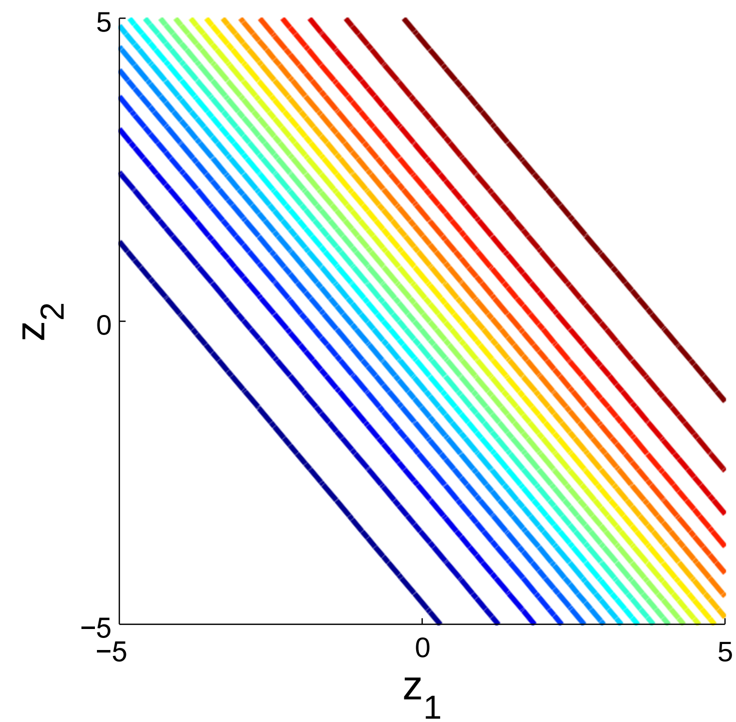




$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

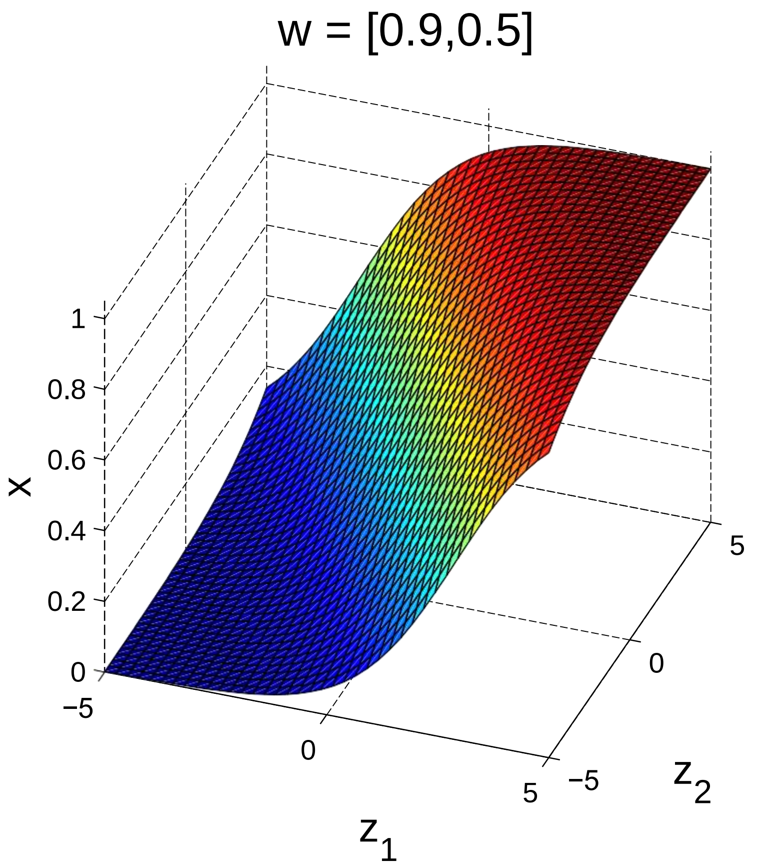
Input-Output Function of a Single Neuron

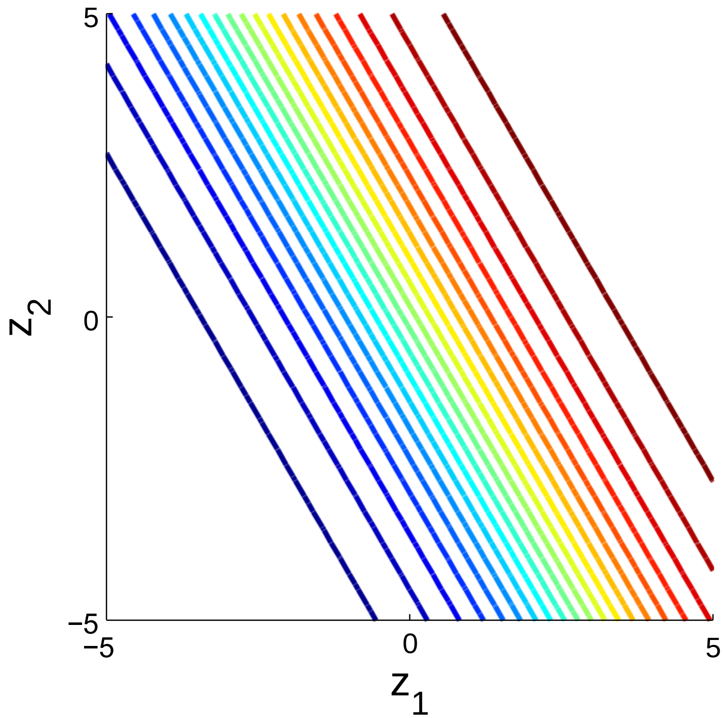




$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

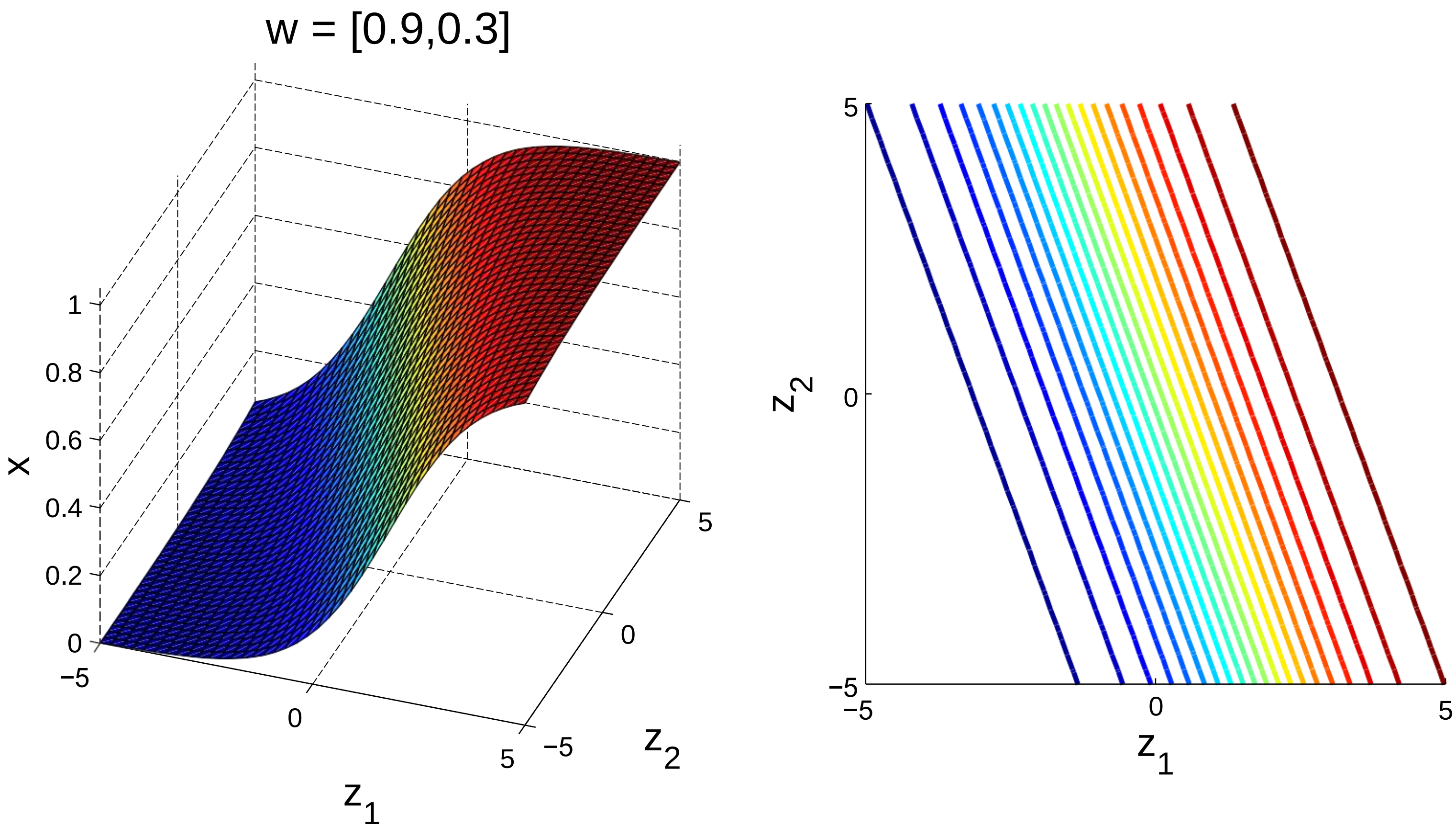
Input-Output Function of a Single Neuron





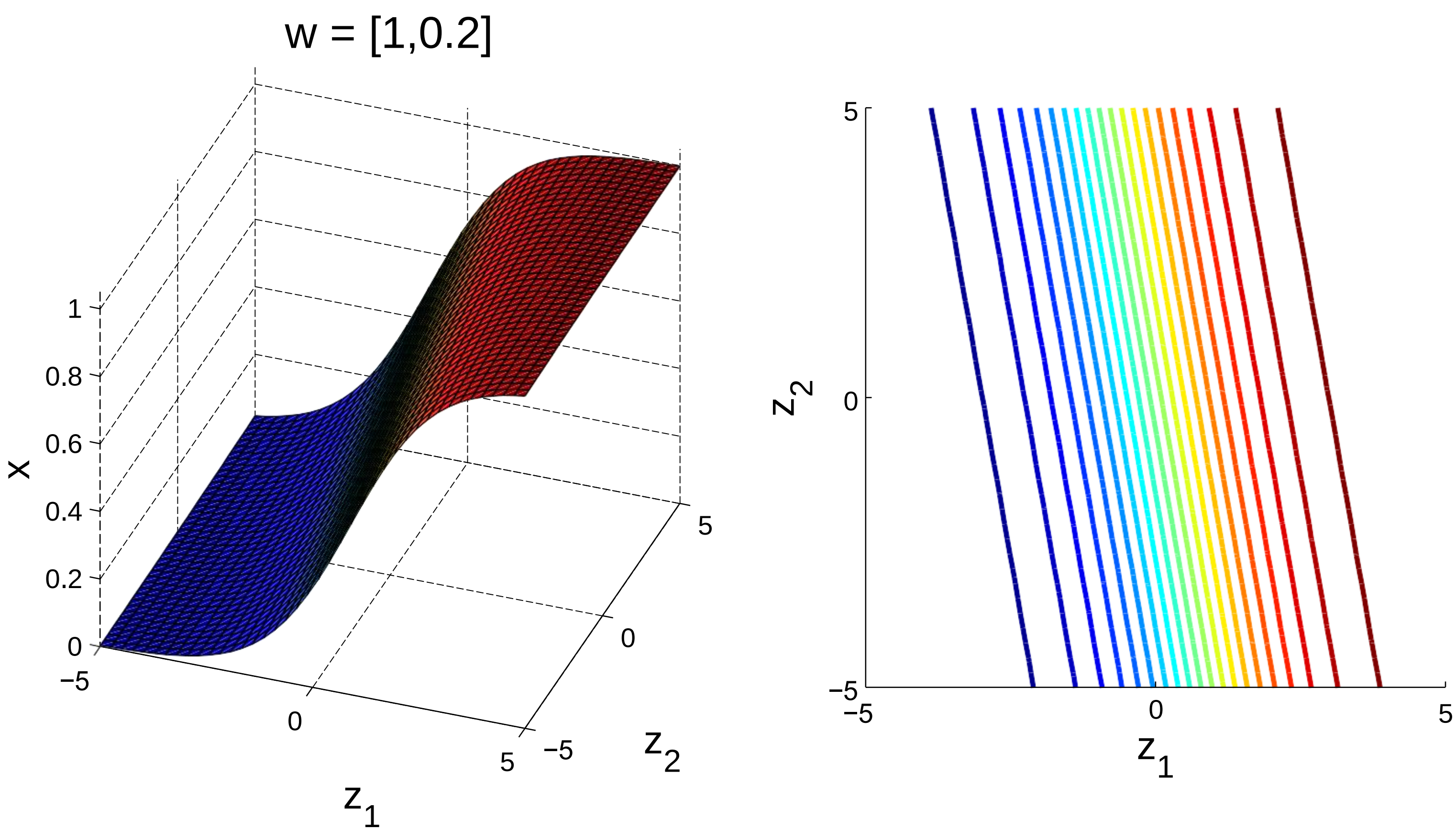
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

Input-Output Function of a Single Neuron



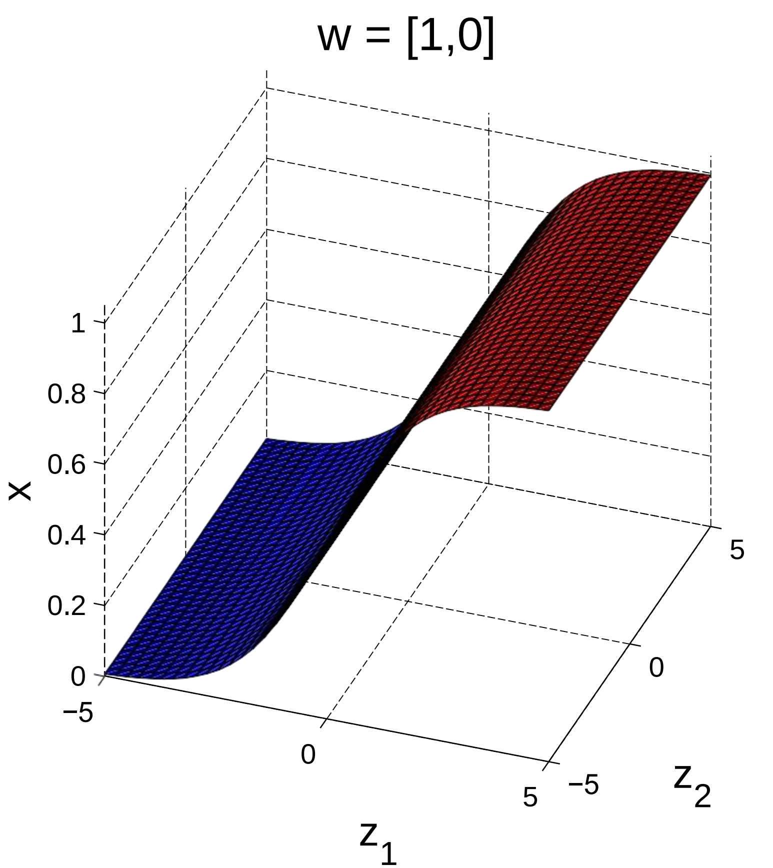
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

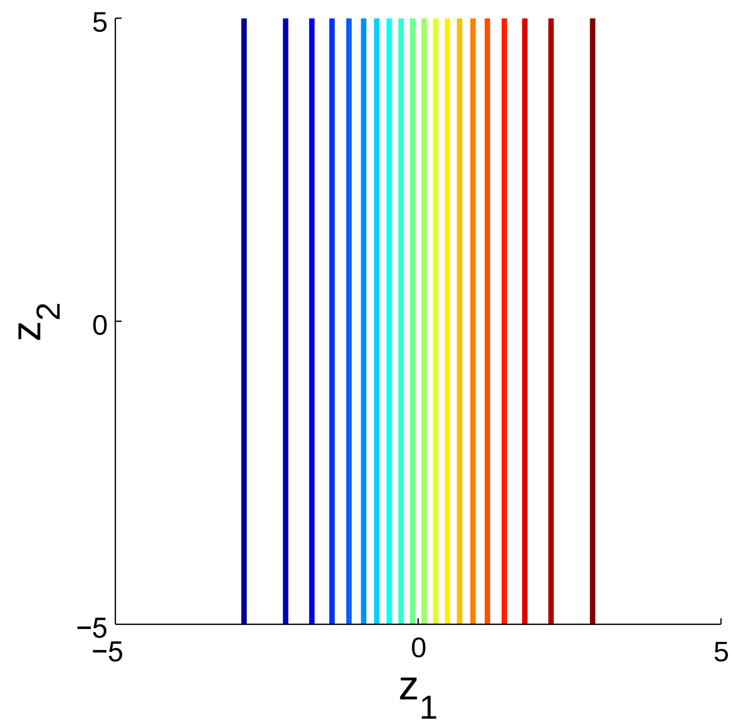
Input-Output Function of a Single Neuron



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

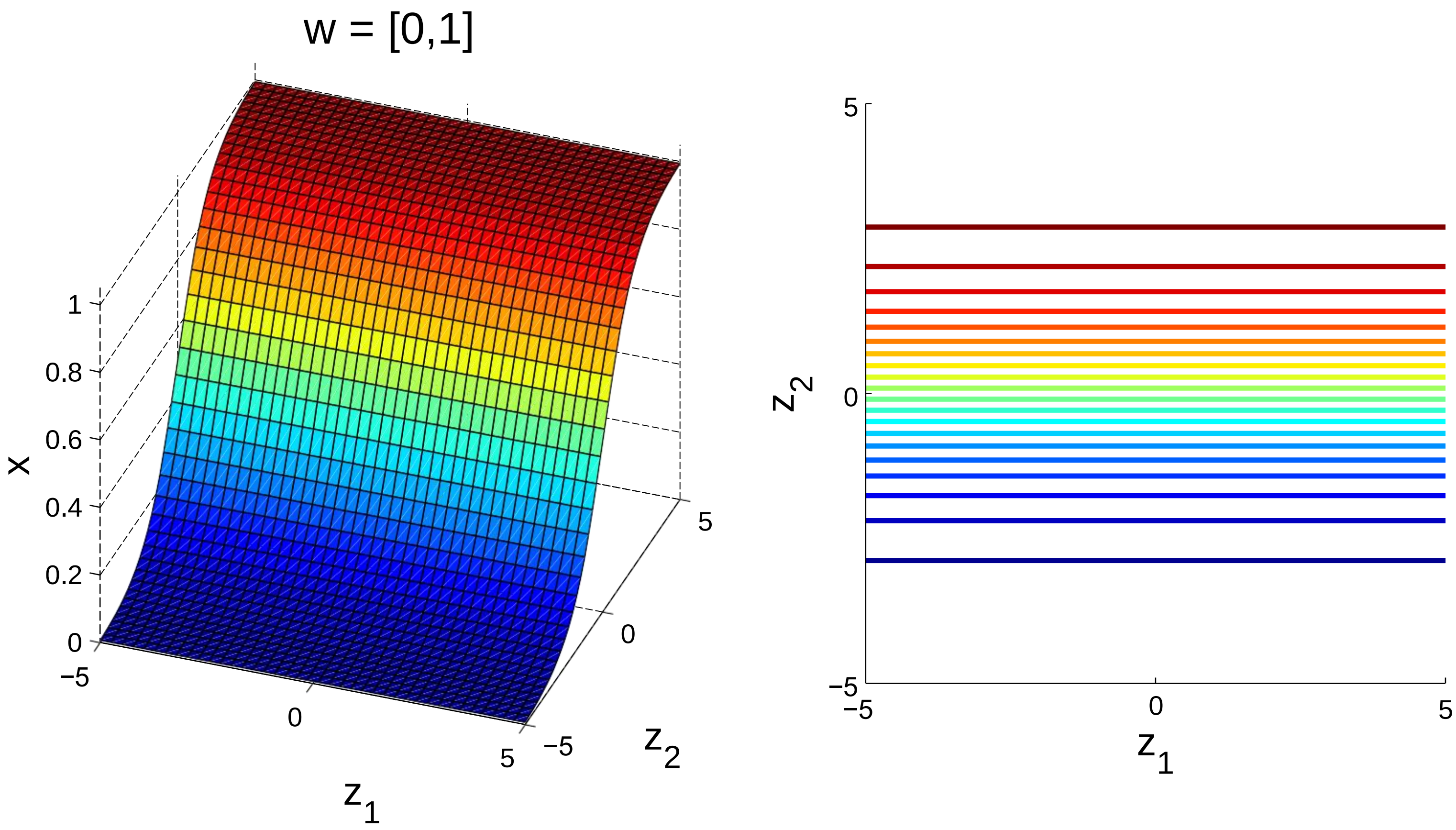
Input-Output Function of a Single Neuron





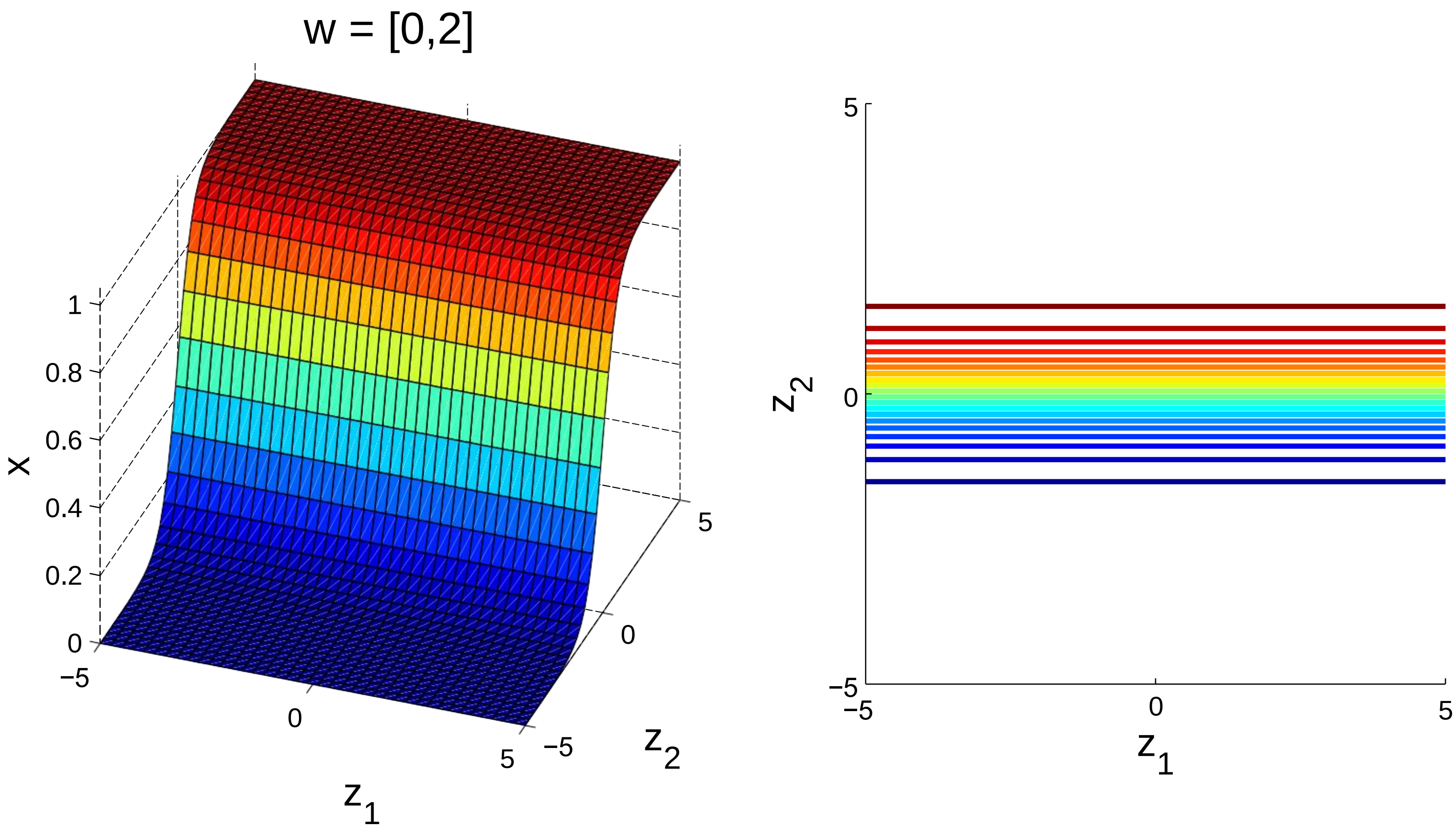
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

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



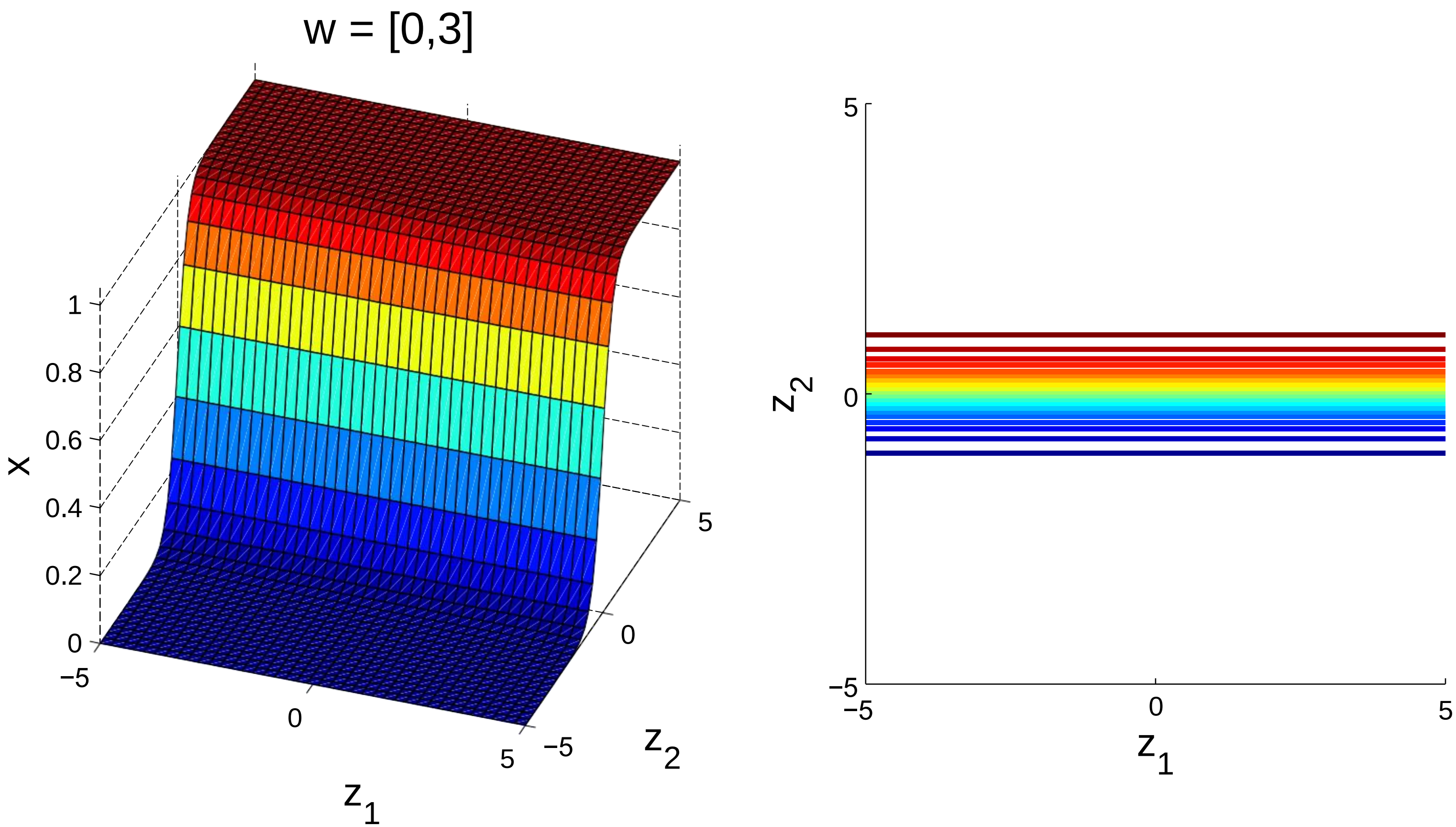
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

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



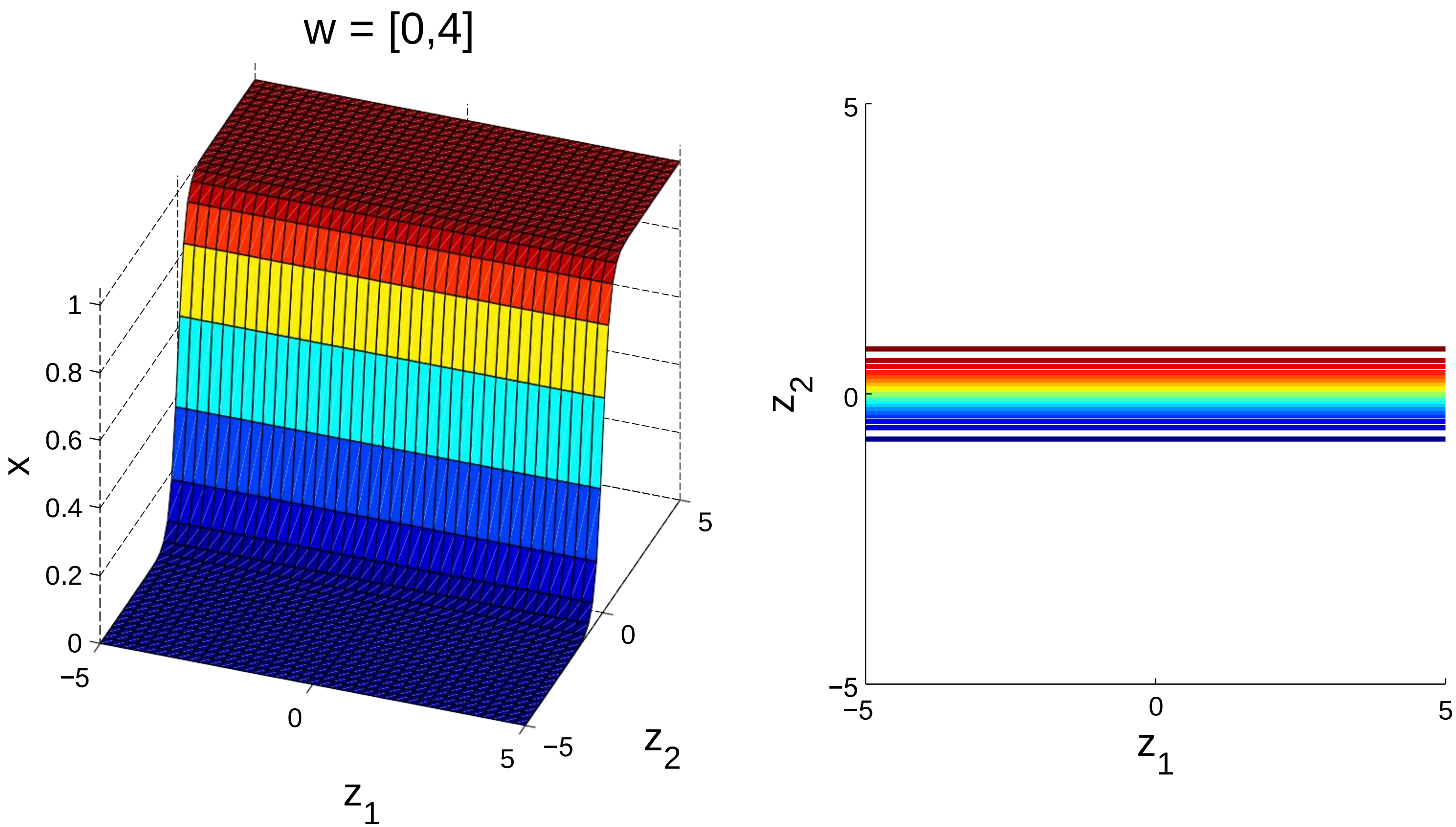
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

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



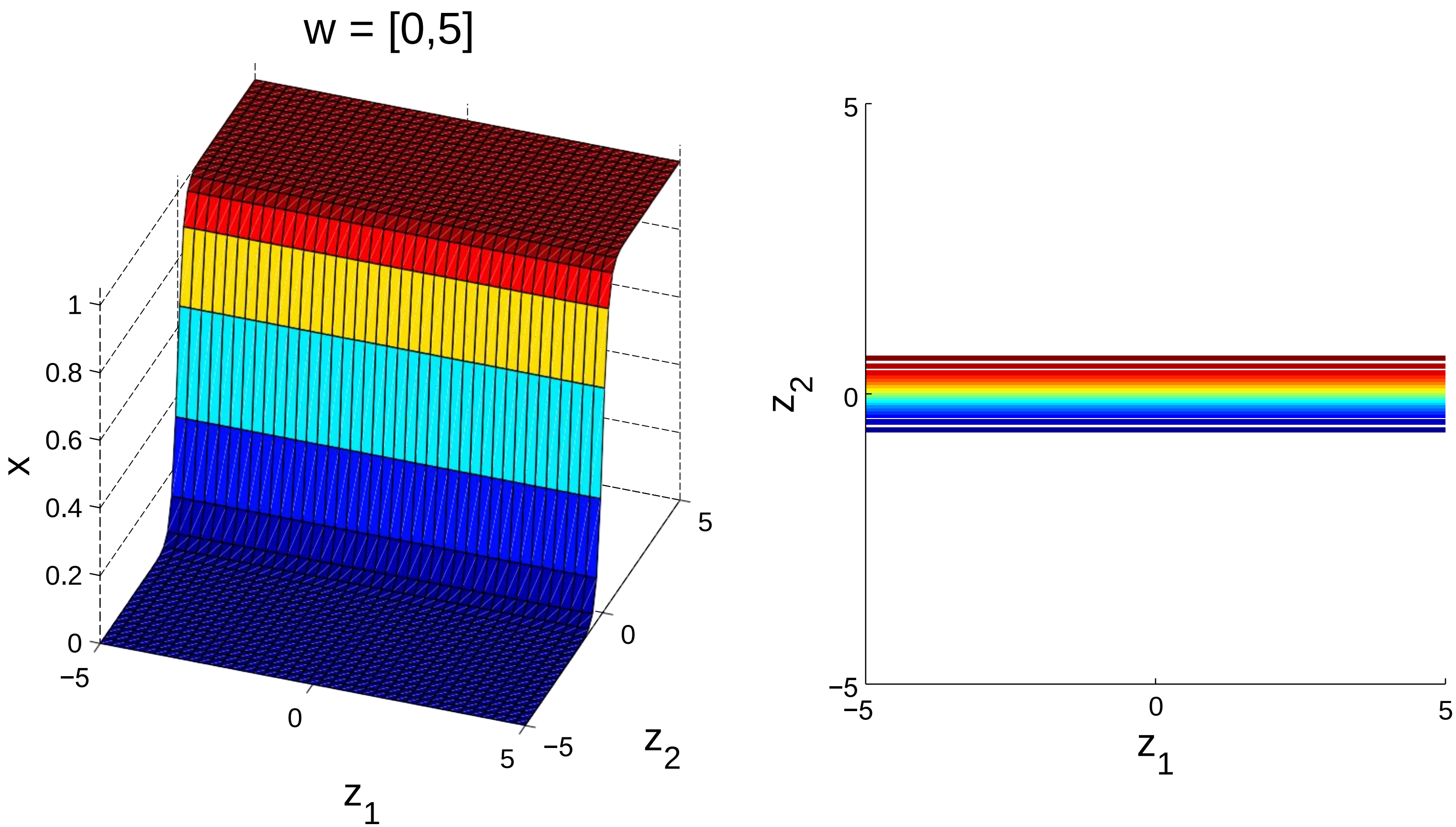
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

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



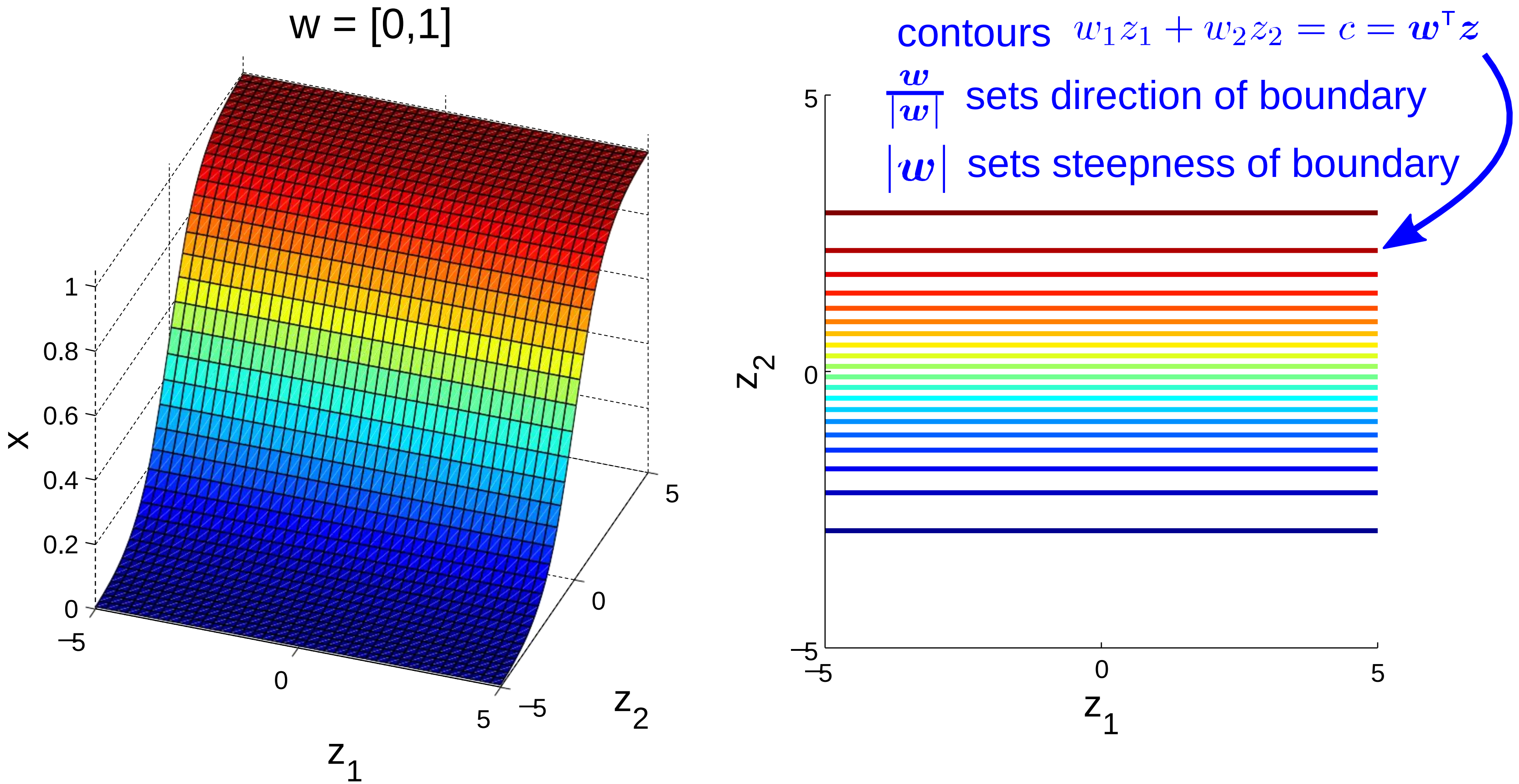
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

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



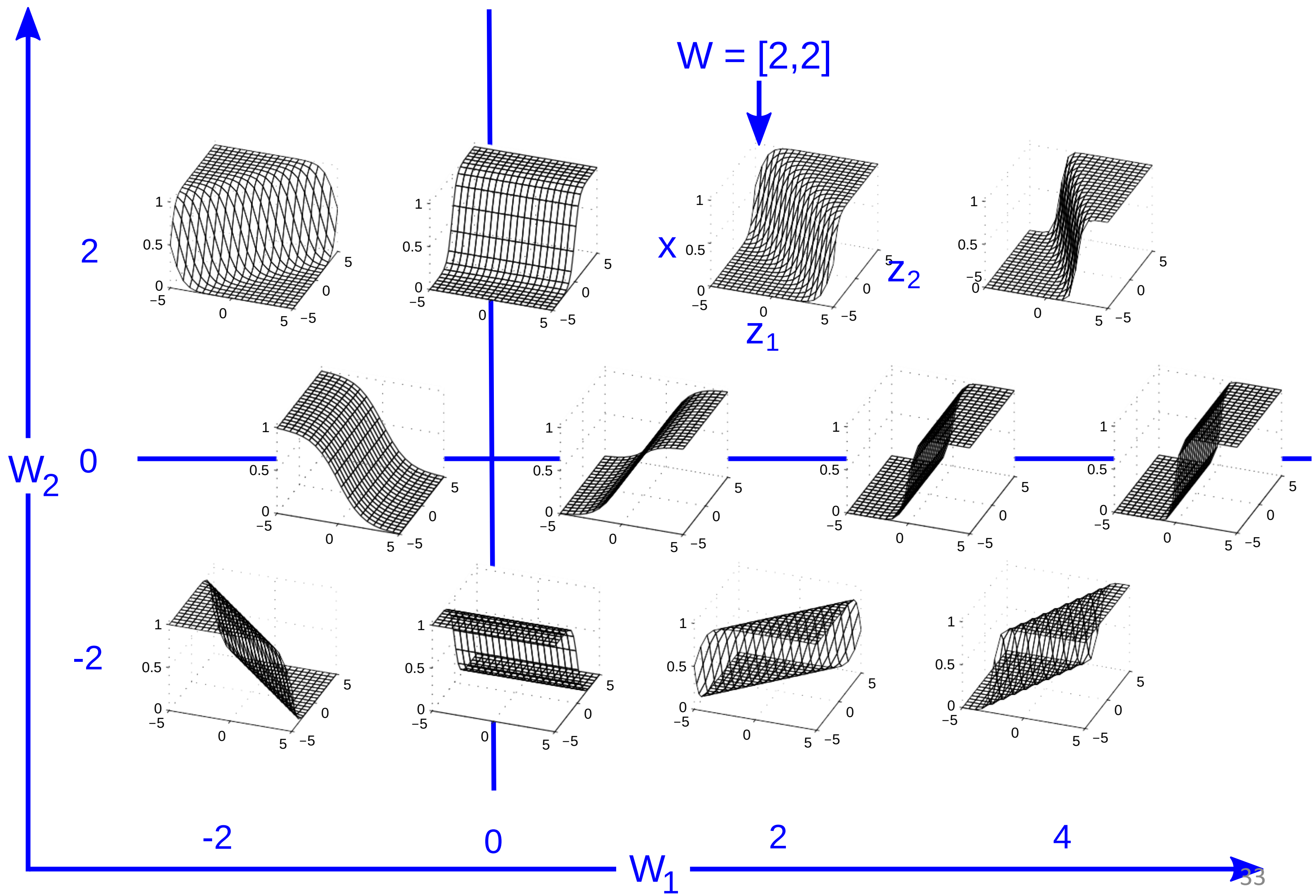
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

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

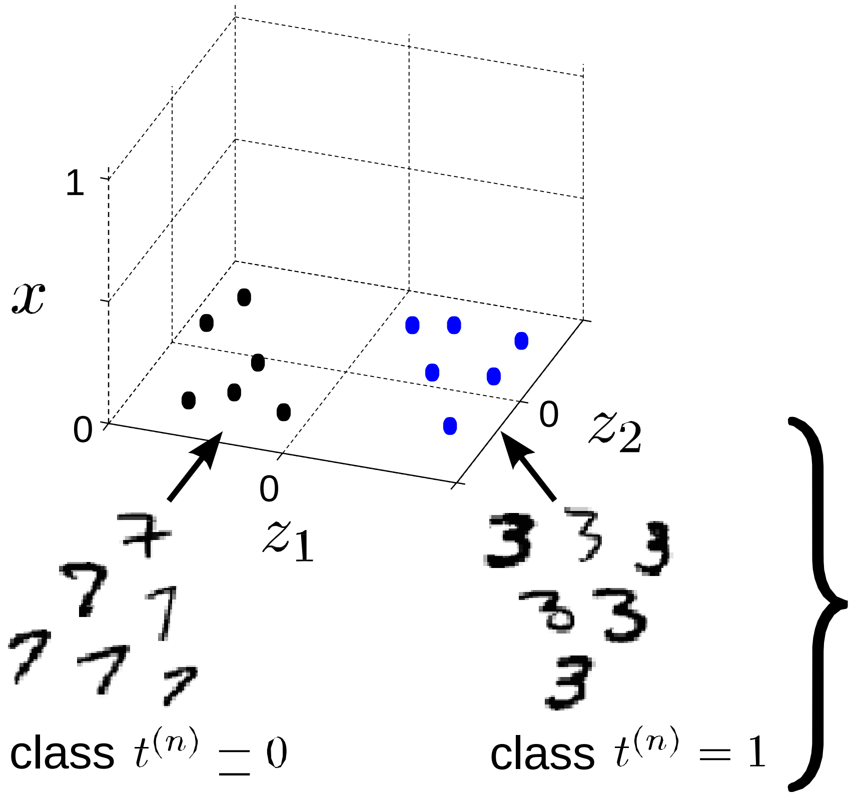


$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

Weight Space of a Single Neuron



Training a Single Neuron

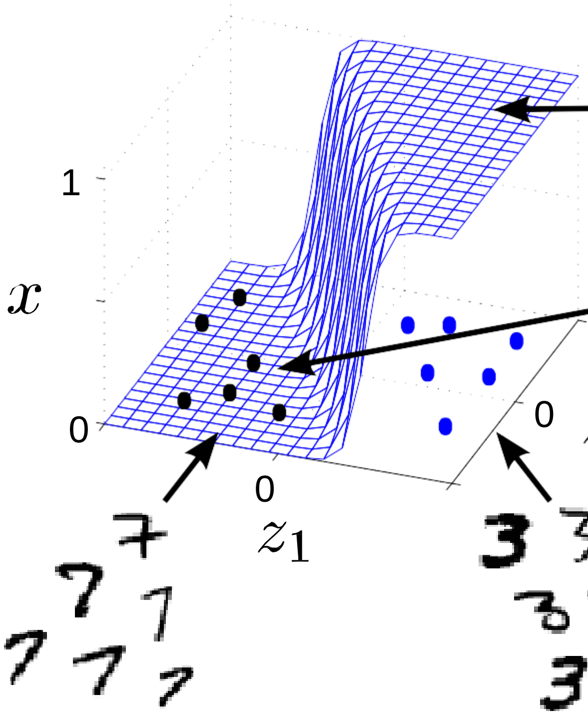


training data

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

inputs class labels

Training a Single Neuron



class $t^{(n)} = 0$

class $t^{(n)} = 1$

desired result of training:

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

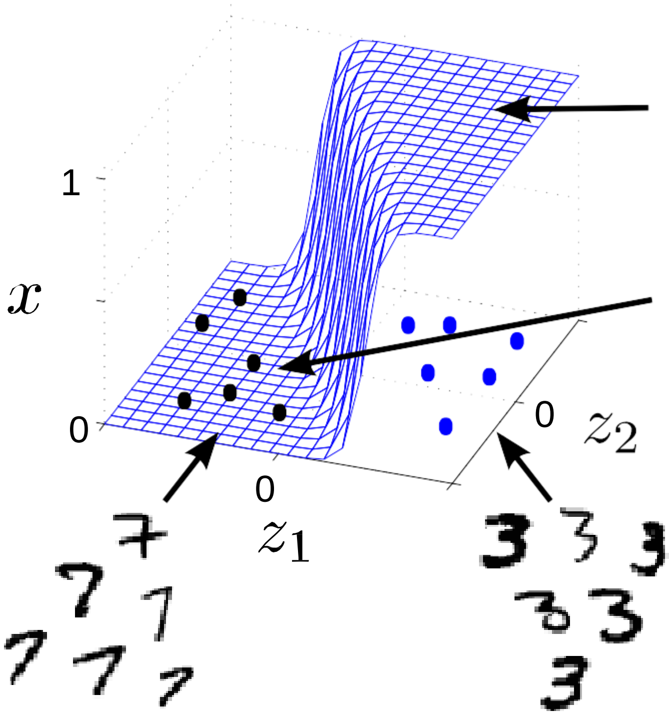
neuron outputs $\mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w}) \approx 0$ for $t^{(n)} = 0$

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 $t^{(n)} = 0$

class $t^{(n)} = 0$ class $t^{(n)} = 1$

training data

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

inputs class labels

objective function:

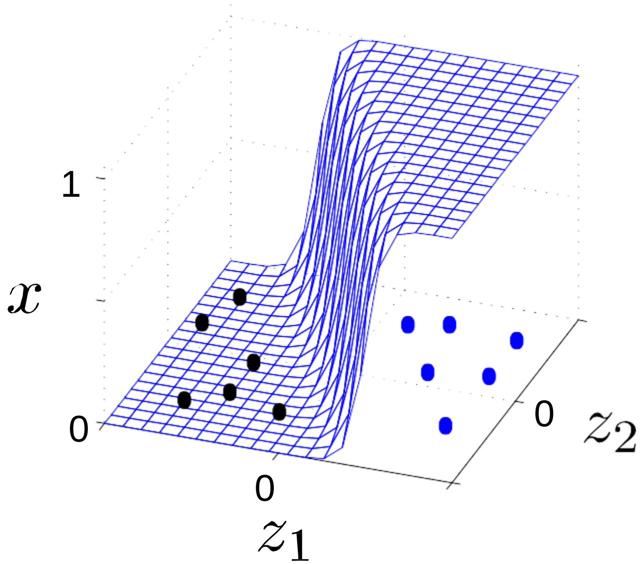
$$G(\boldsymbol{w}) = -\sum_n \left[t^{(n)} \log \mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w}) + (1 - t^{n}) \log \left(1 - \mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w})\right)\right] \geq 0$$

surprise $-\log p(\mathrm{outcome})$ 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

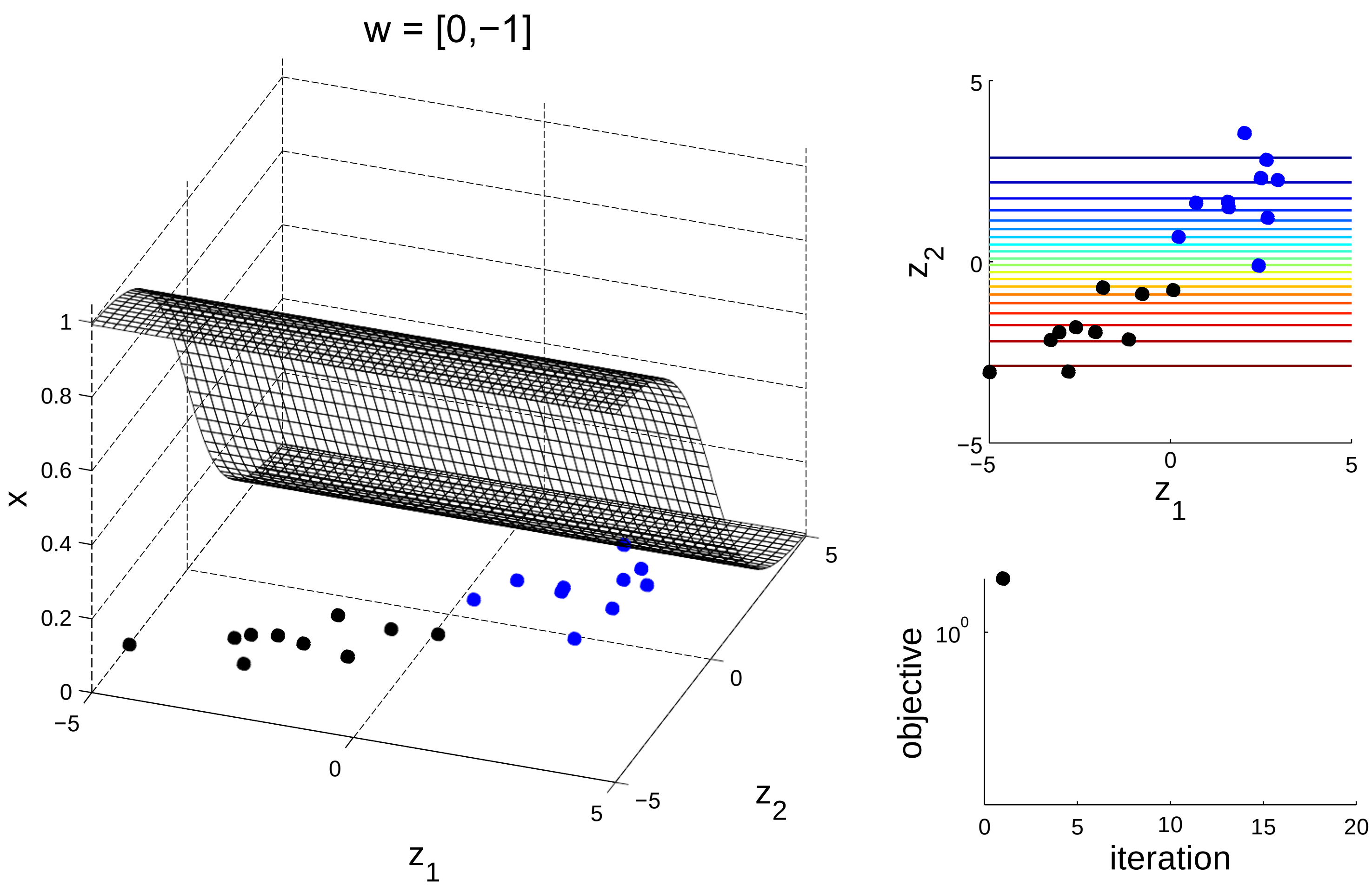
$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{w}} G(\boldsymbol{w}) = \sum_n \frac{\mathrm{d}G(\boldsymbol{w})}{\mathrm{d}x^{(n)}} \frac{\mathrm{d}x^{(n)}}{\mathrm{d}\boldsymbol{w}} = -\sum_n (t^{(n)} - x^{(n)}) \boldsymbol{z}^{(n)}$$

= prediction error x feature

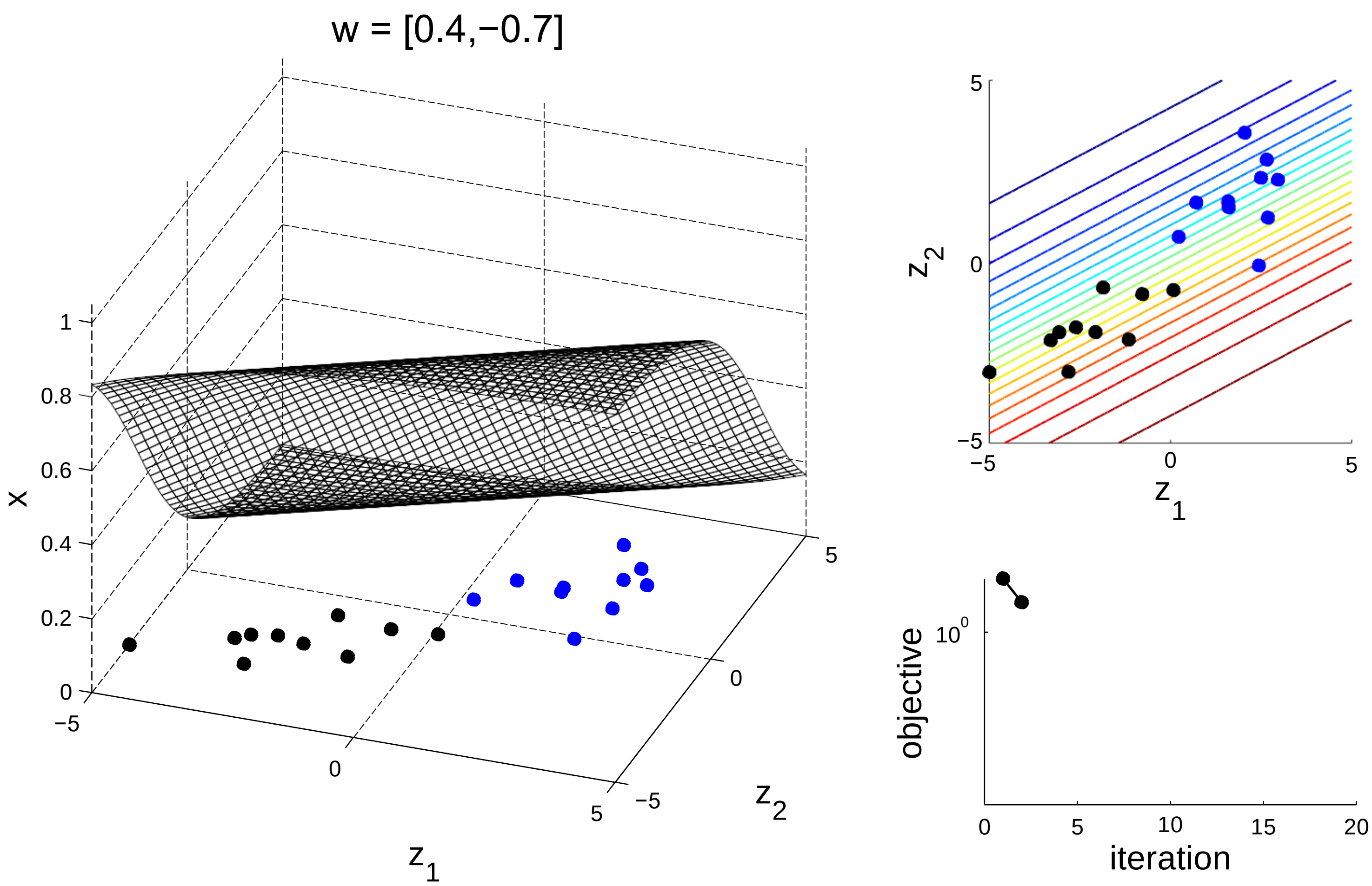
$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{w}} G(\boldsymbol{w})$$

iteratively step down the objective (gradient points up hill)

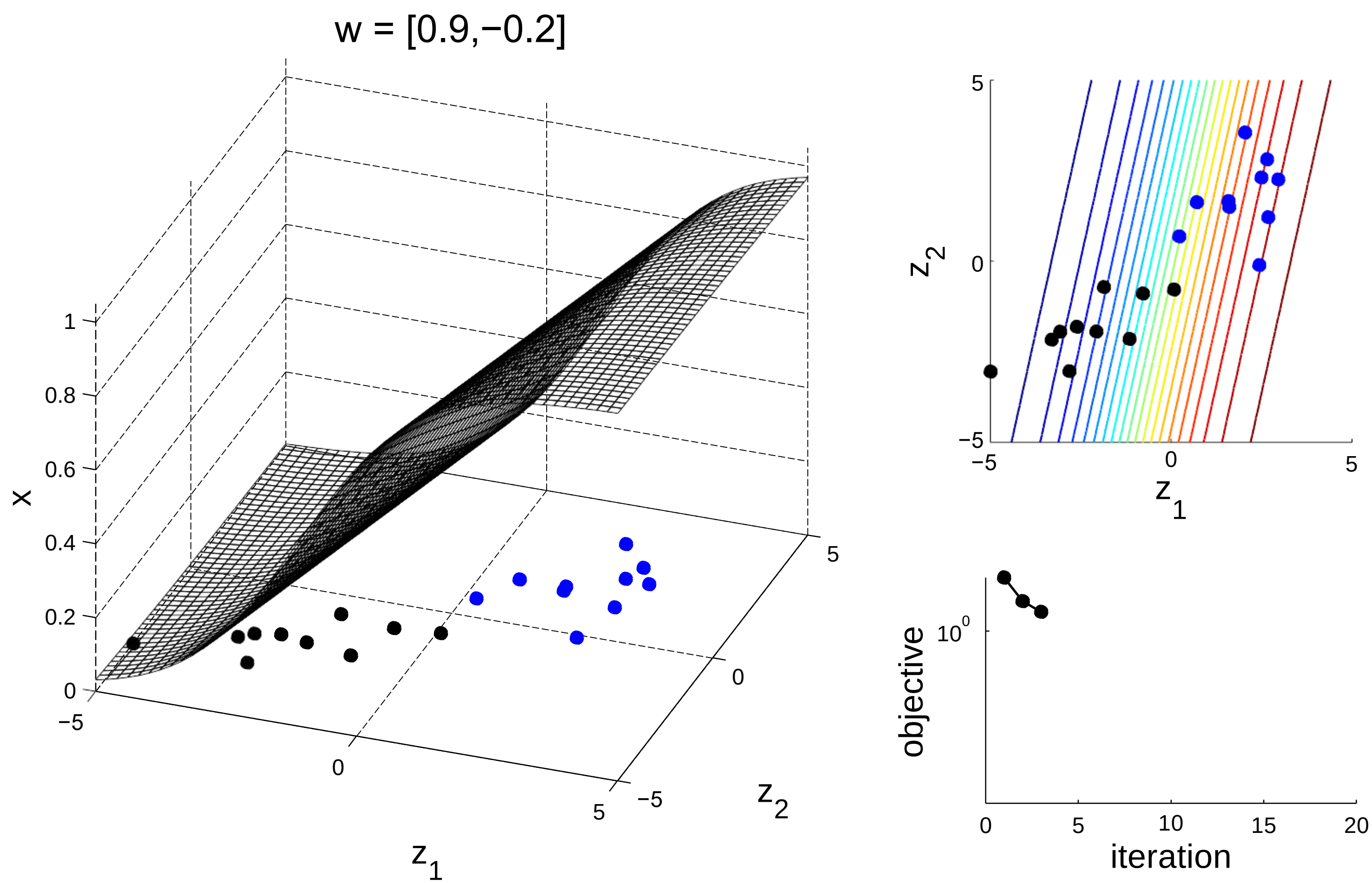
Training a Single Neuron



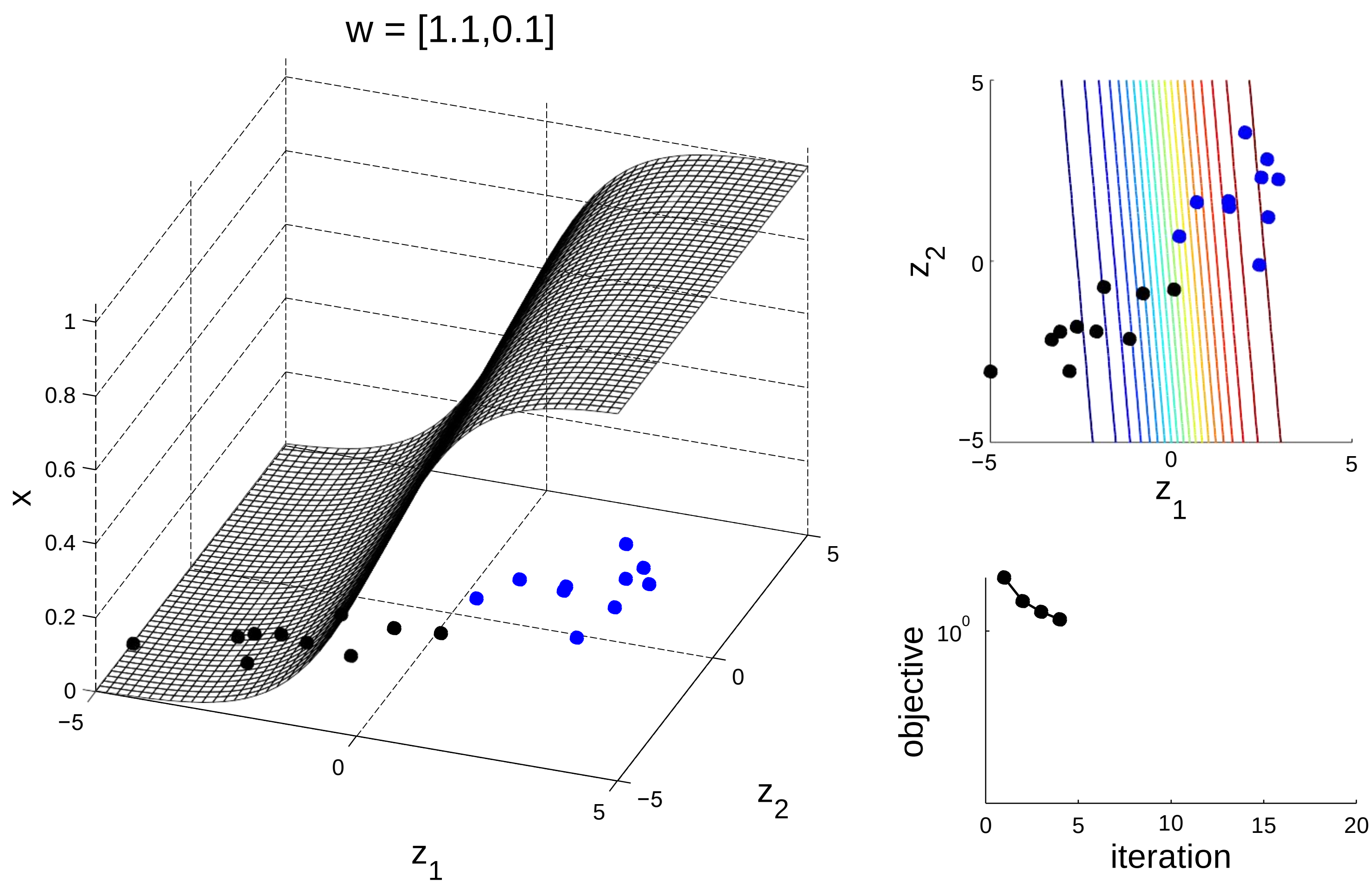
Training a Single Neuron



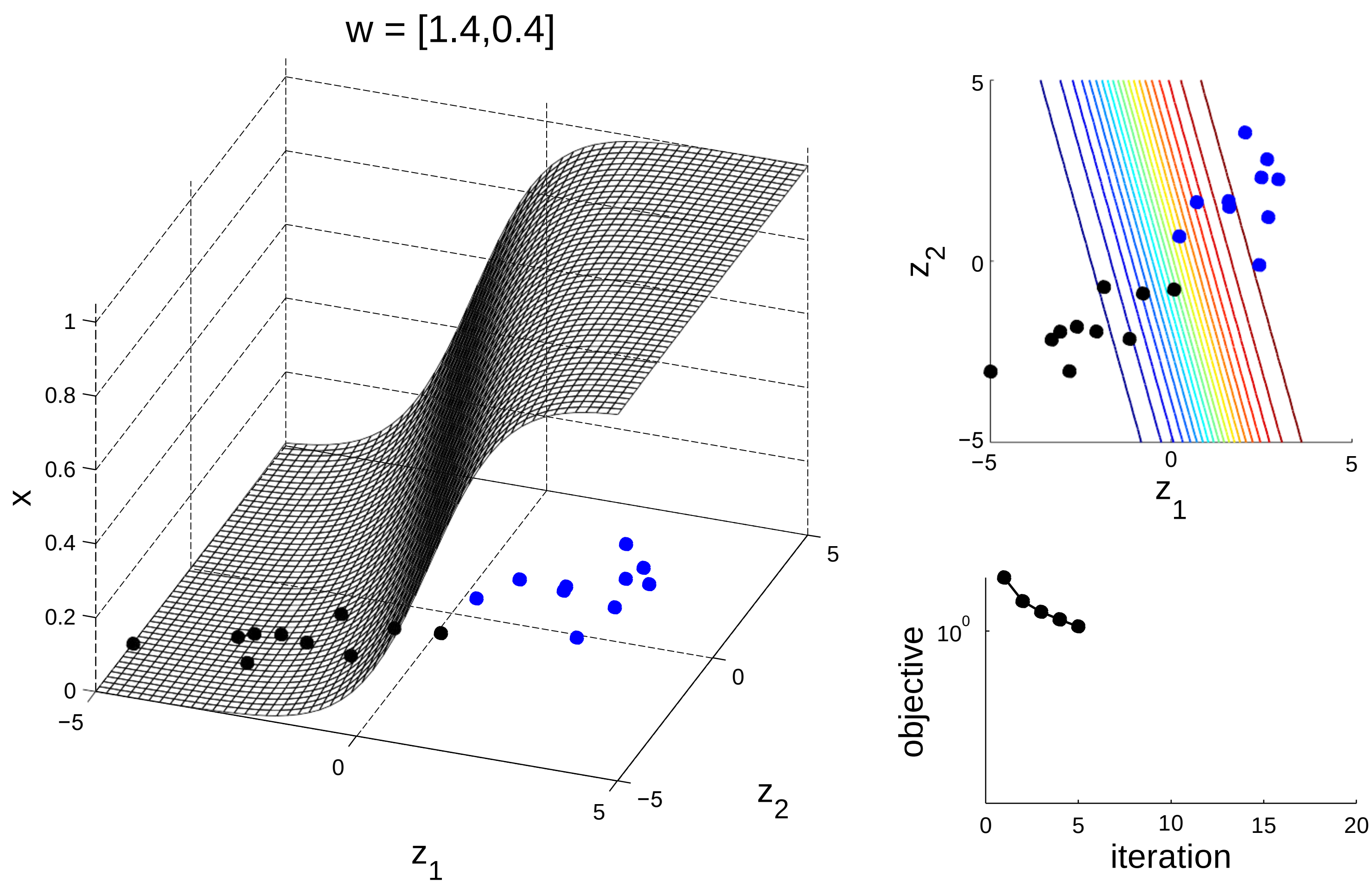
Training a Single Neuron



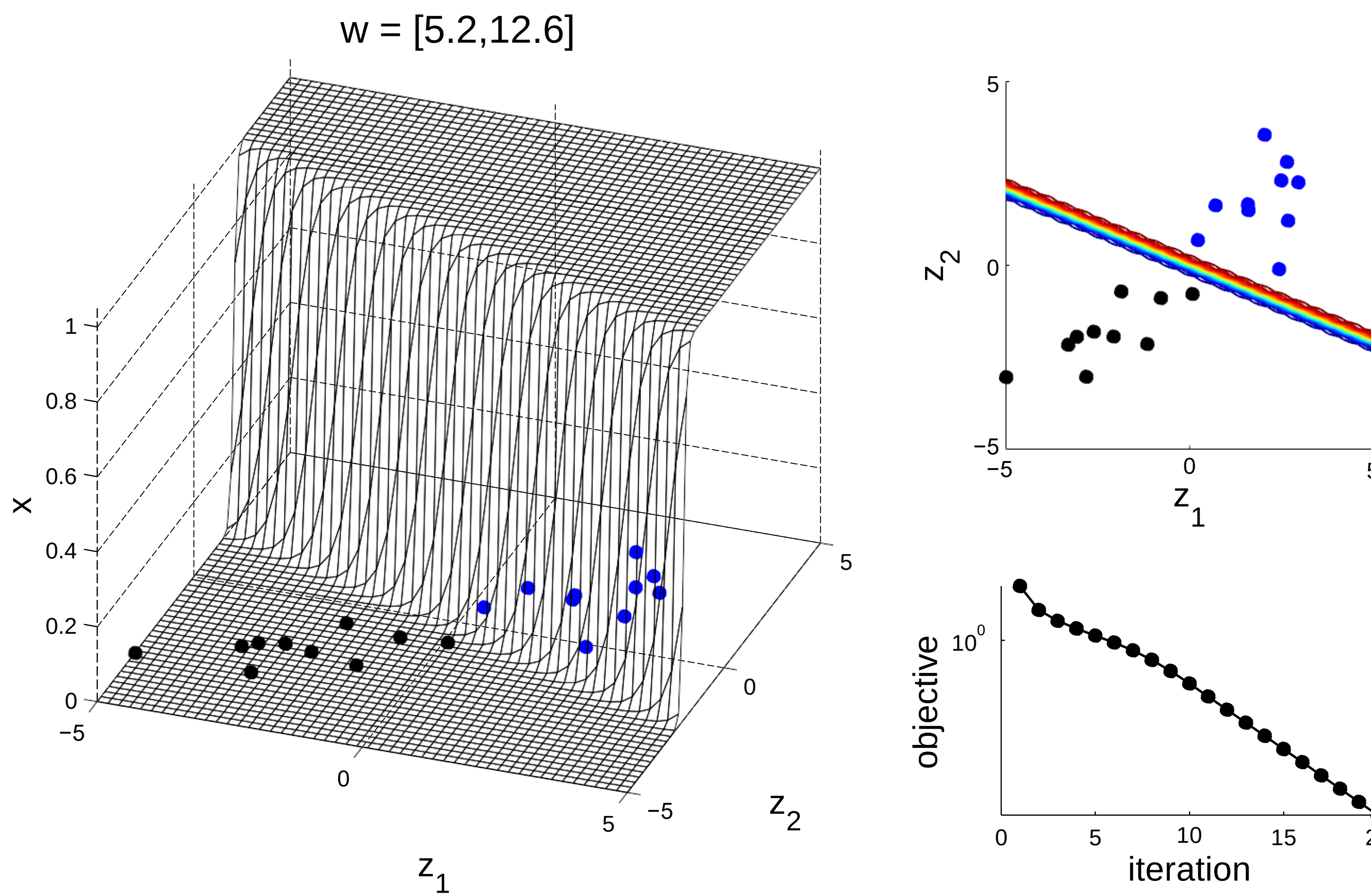
Training a Single Neuron



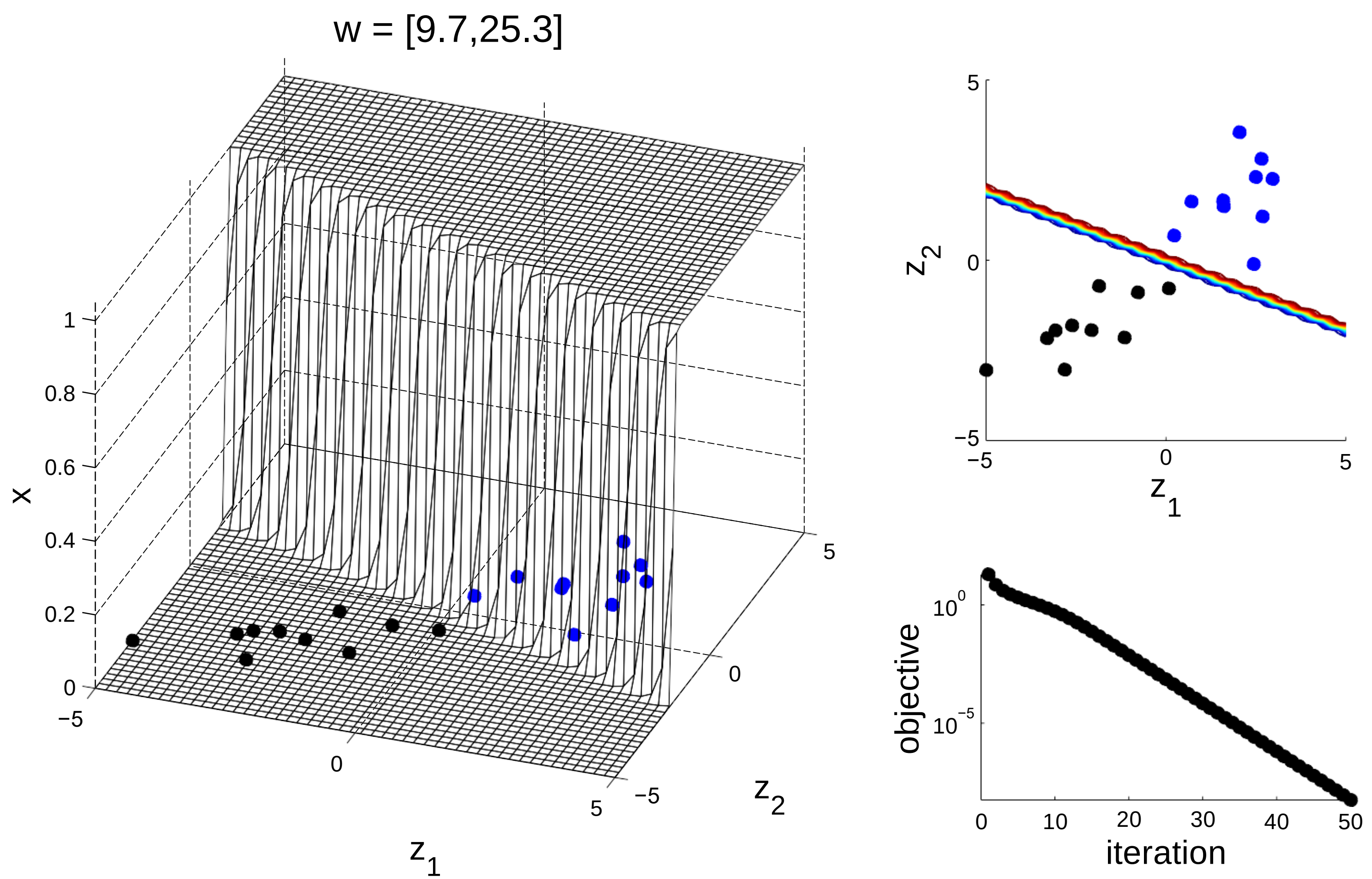
Training a Single Neuron



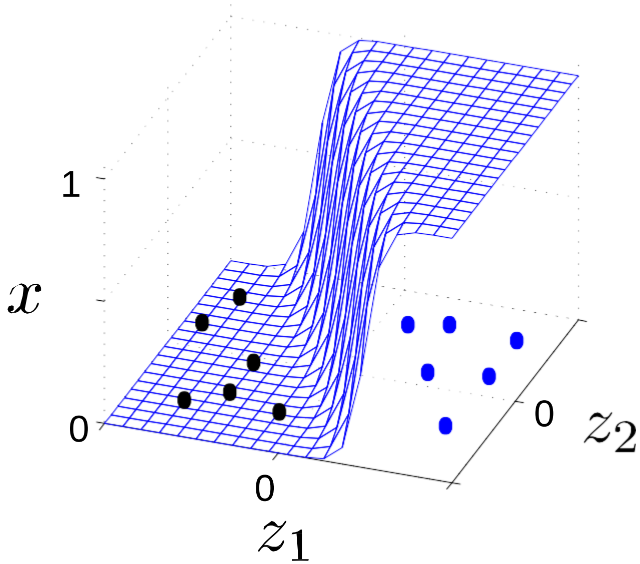
Training a Single Neuron



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:

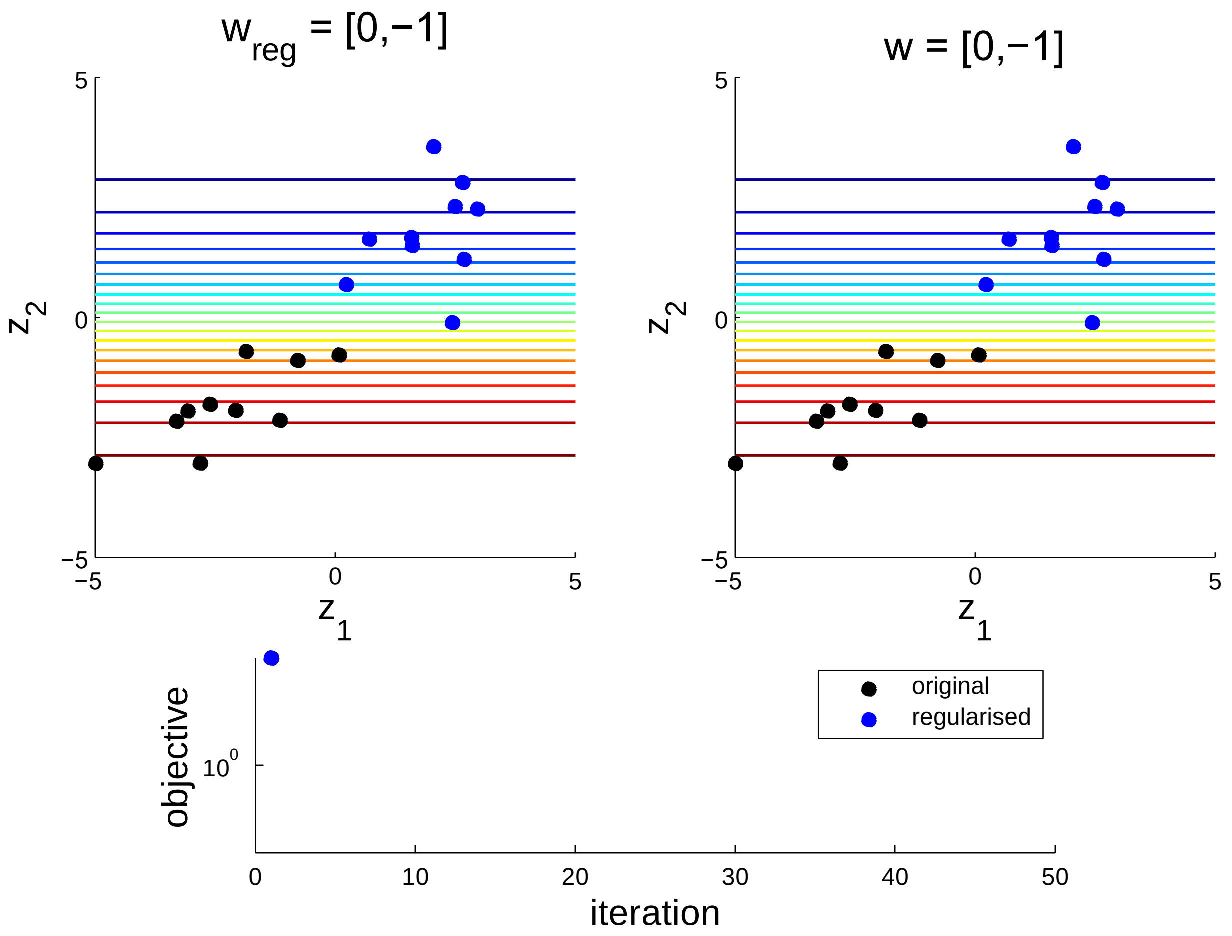
$$G(\boldsymbol{w}) = -\sum_n \left[t^{(n)} \log \mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w}) + (1 - t^n) \log \left(1 - \mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w})\right)\right]$$

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

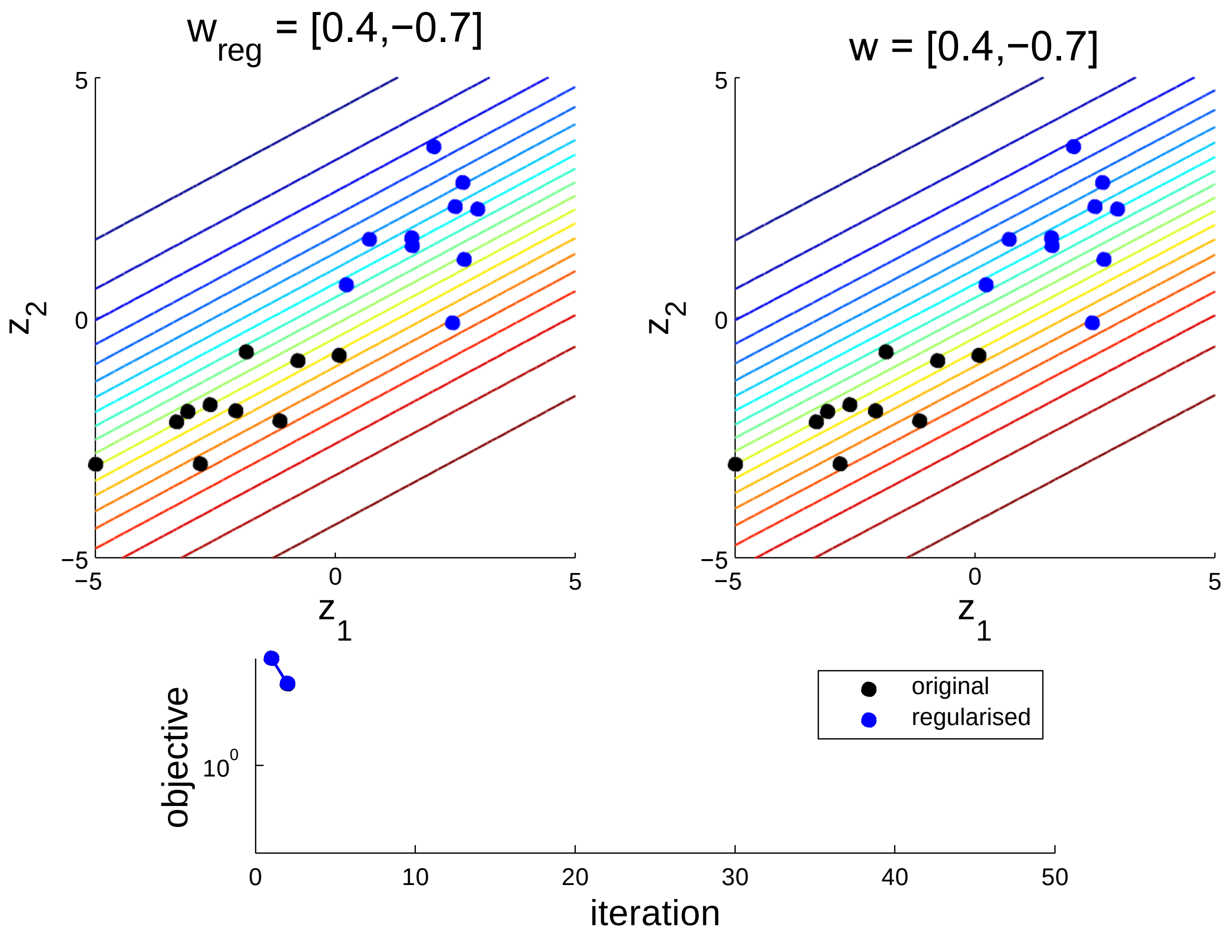
$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} M(\boldsymbol{w}) = \arg\min_{\boldsymbol{w}} \left[G(\boldsymbol{w}) + \alpha E(\boldsymbol{w})\right]$$

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

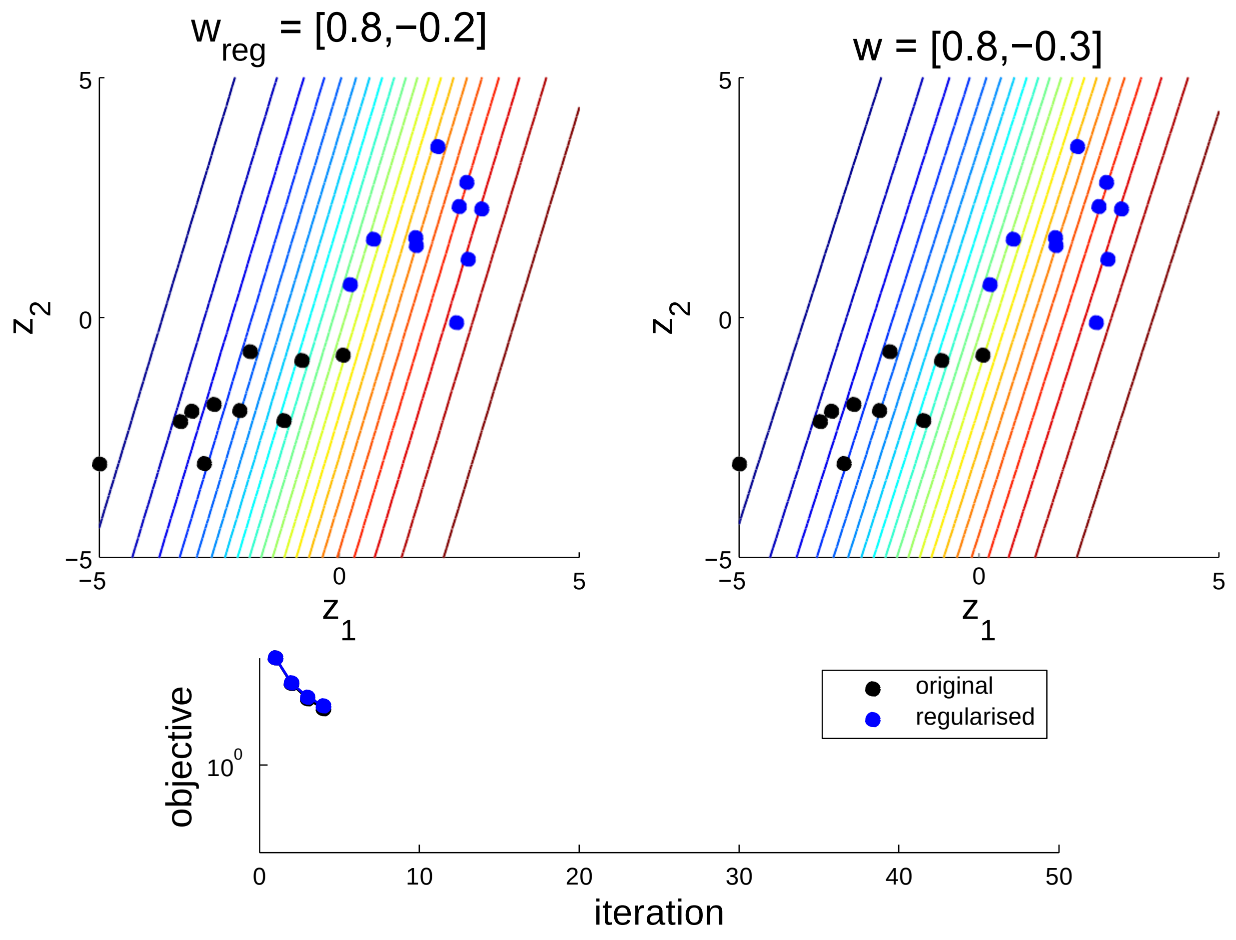
Training a Single Neuron (cont'd)



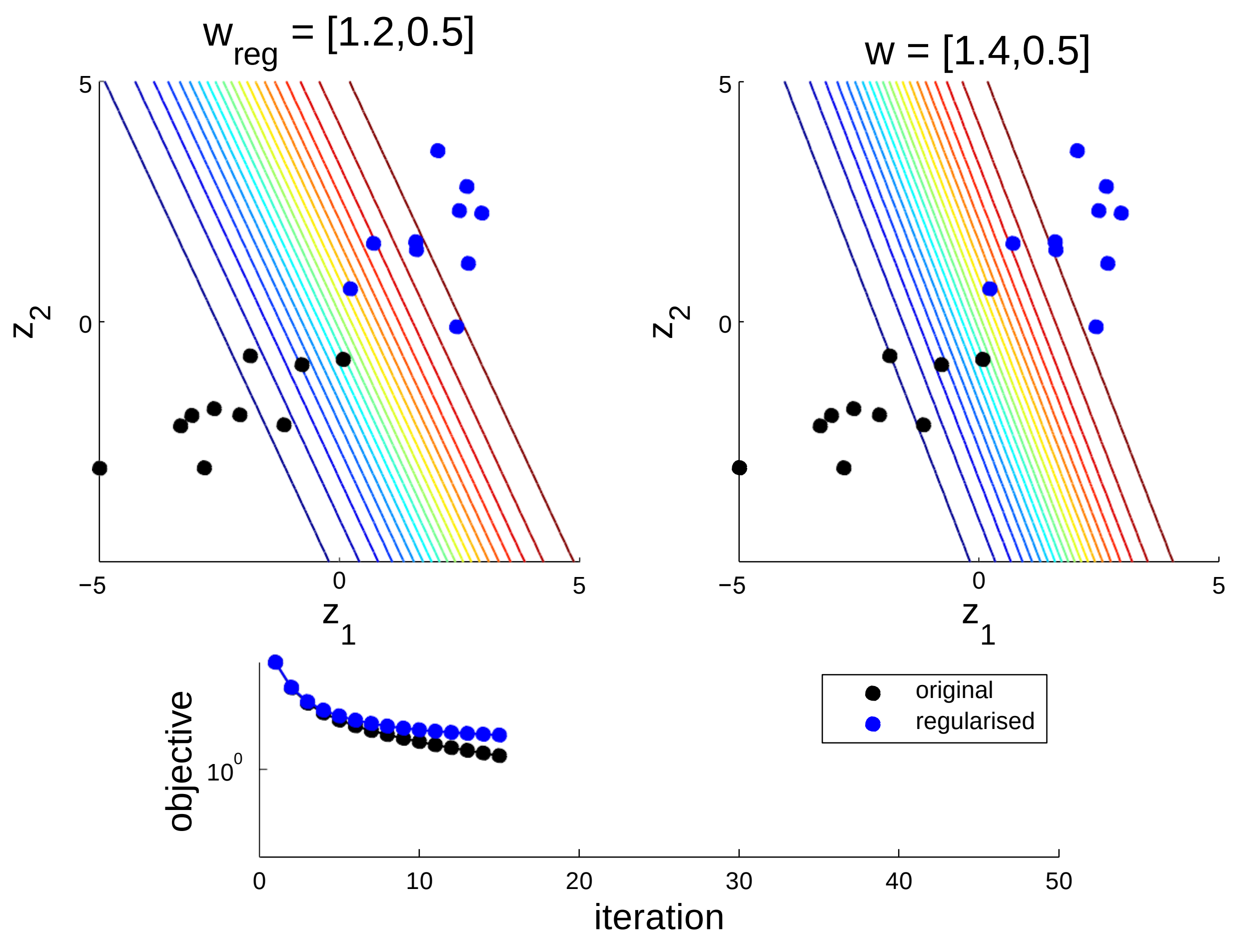
Training a Single Neuron (cont'd)



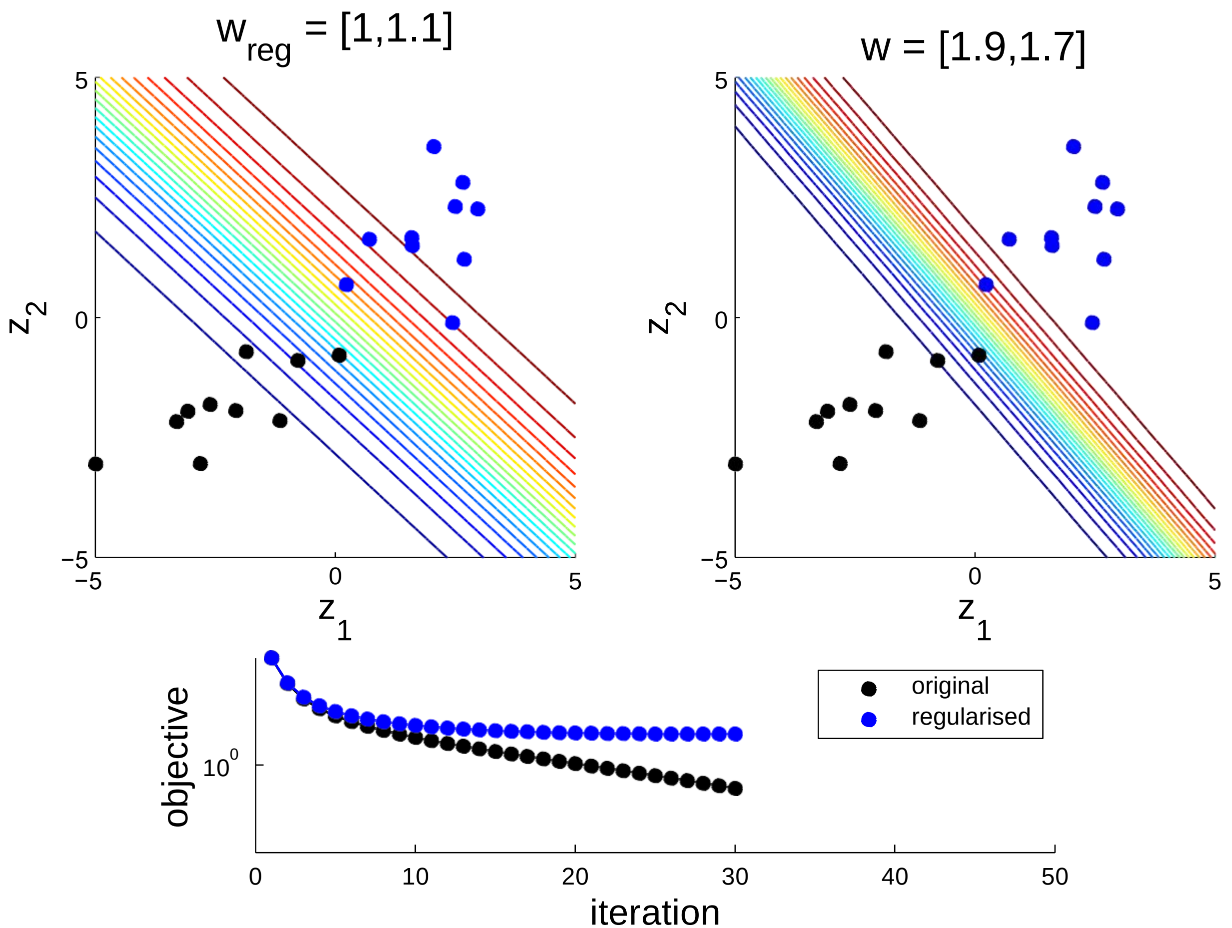
Training a Single Neuron (cont'd)



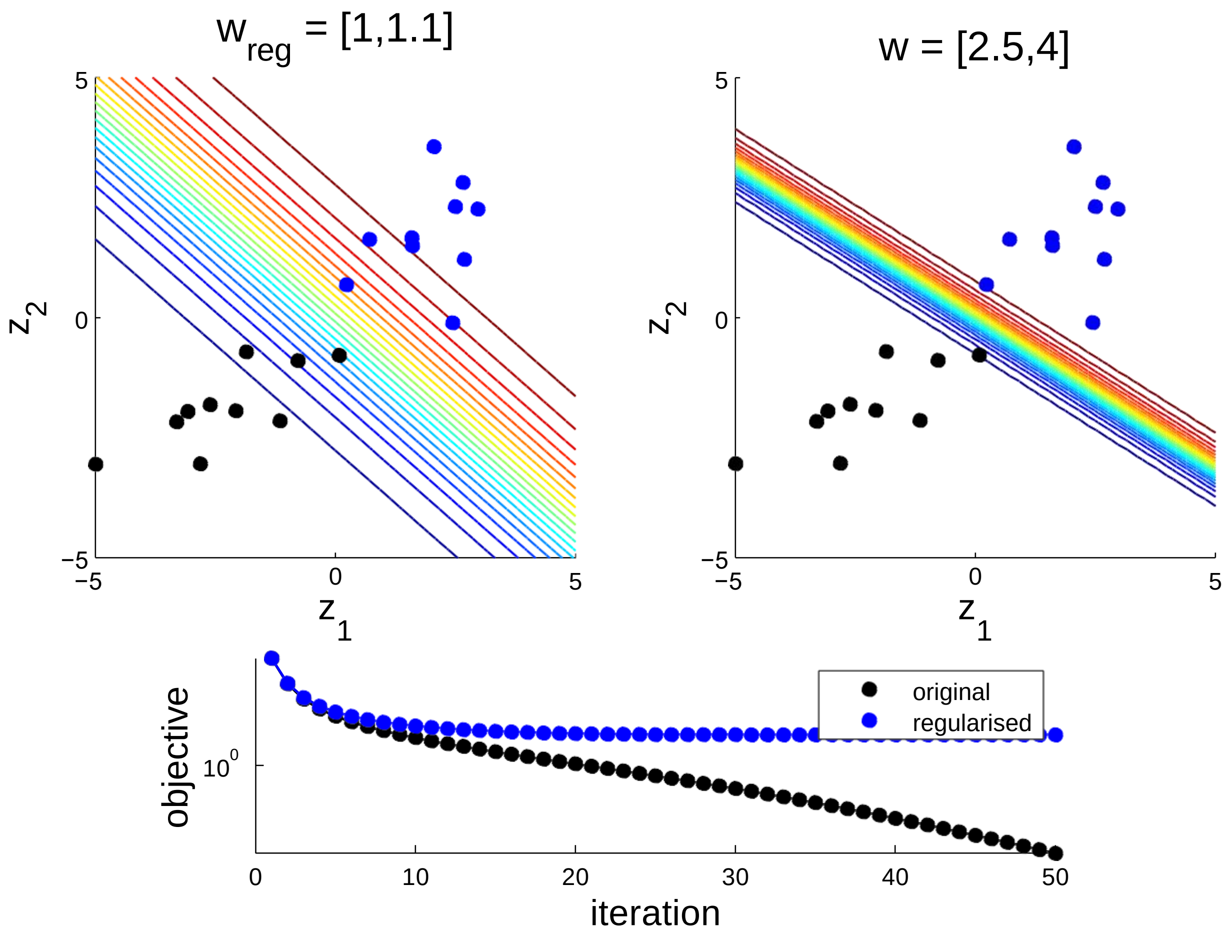
Training a Single Neuron (cont'd)



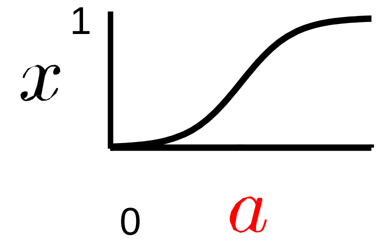
Training a Single Neuron (cont'd)

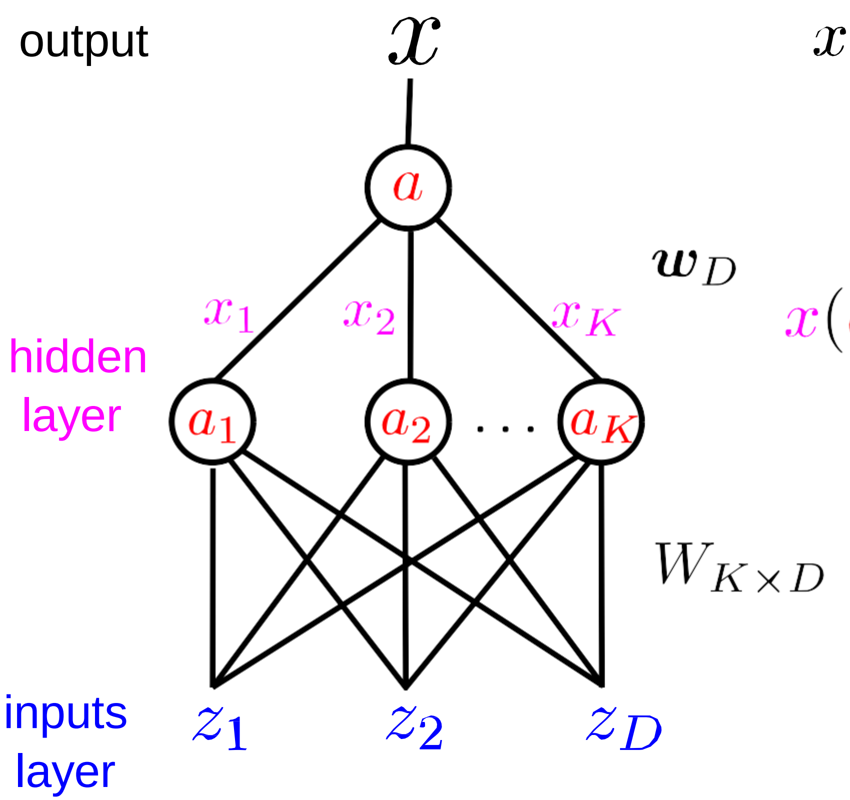


Training a Single Neuron (cont'd)



Single Hidden Layer Neural Networks





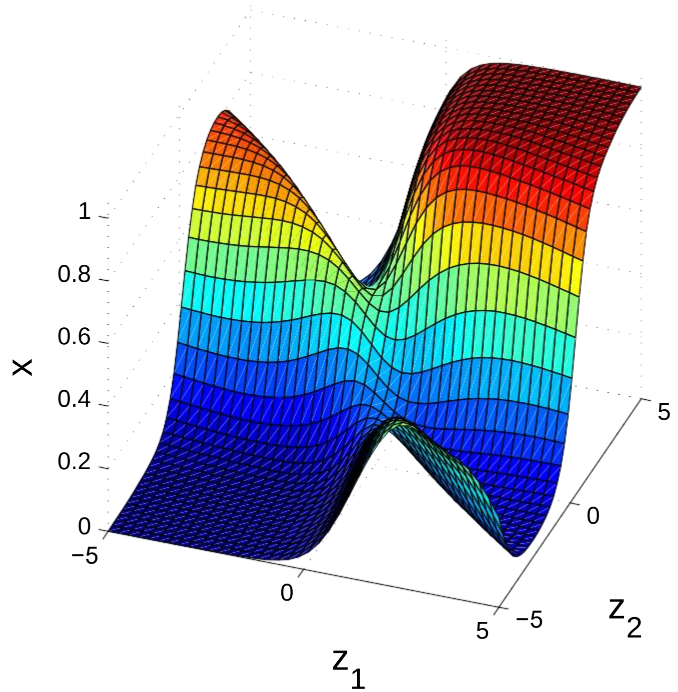
$$x(a) = \frac{1}{1+\exp(-a)}$$

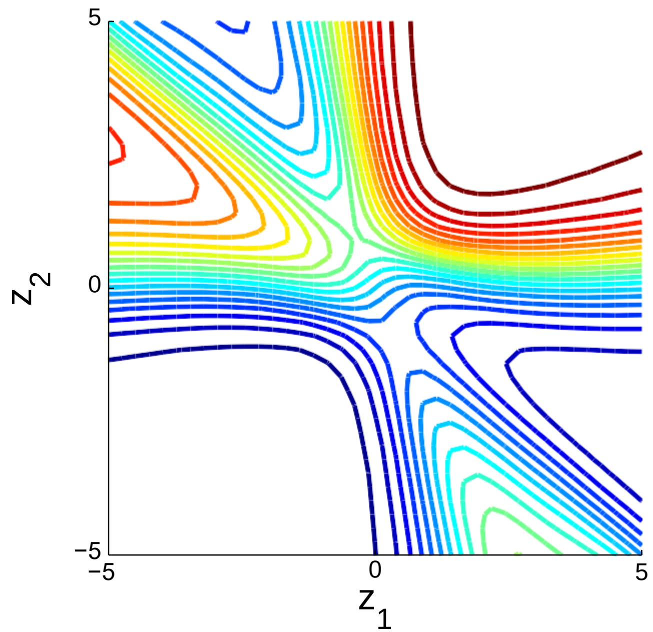
$$a = \sum_{k=1}^{K} w_k x_k$$

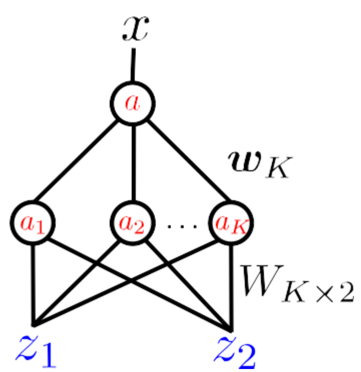
$$x(a_k) = \frac{1}{1+\exp(-a_k)}$$

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

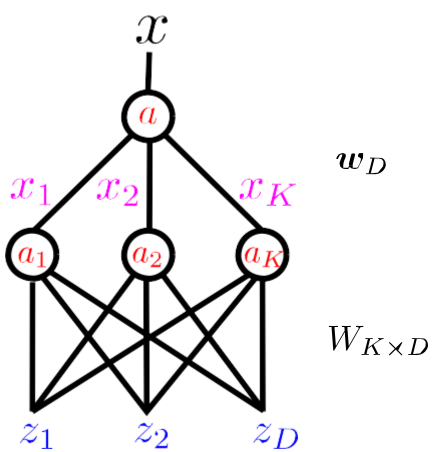
Sampling Random Neural Network Classifiers







Training a Neural Network with a Single Hidden Layer



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

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

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

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

objective function:

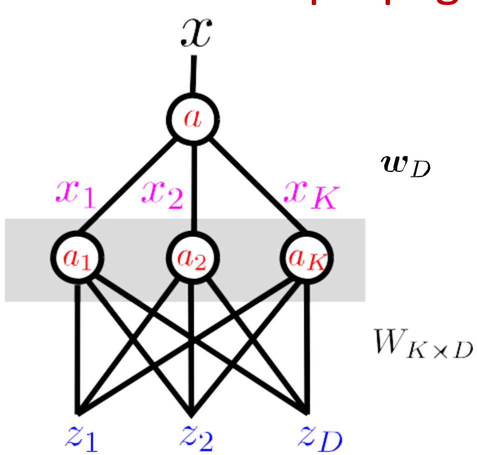
$$G(W, \boldsymbol{w}) = -\sum_n \left[t^{(n)} \log \mathrm{x}^{(n)} + (1 - t^n) \log\left(1 - \mathrm{x}^{(n)}\right)\right]$$ likelihood same as before

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

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

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

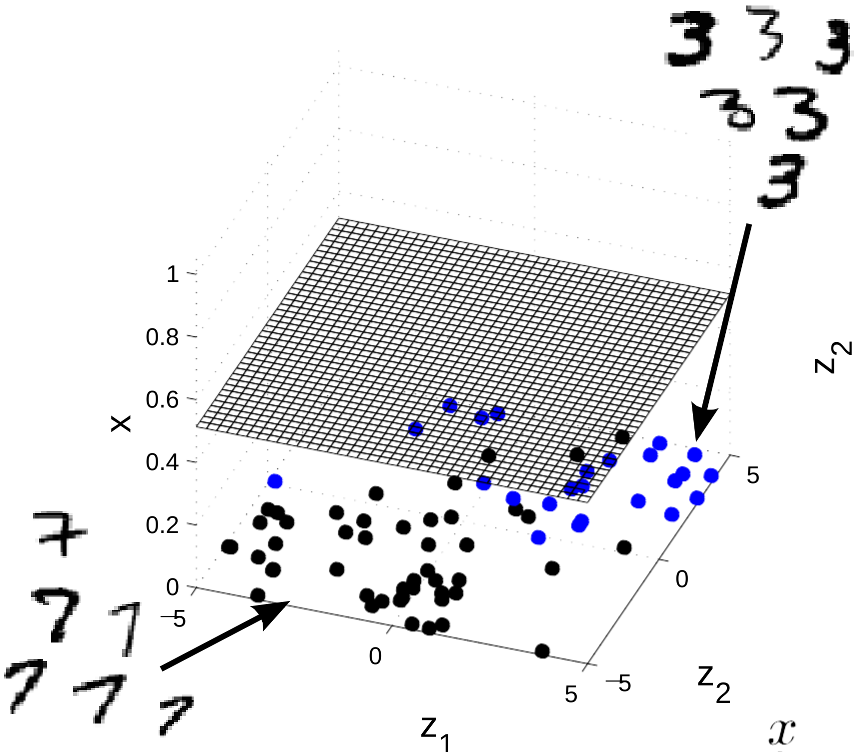
$$E(W, \boldsymbol{w}) = \frac{1}{2}\sum_i w_i^2 + \frac{1}{2}\sum_{ij} W_{ij}^2$$ regulariser discourages extreme weights

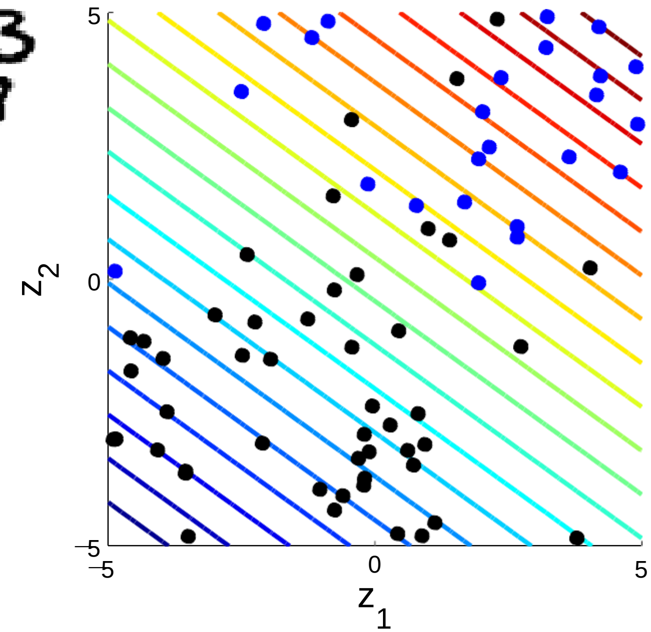
$$\{W, \boldsymbol{w}^*\} = \underset{W,\boldsymbol{w}}{\arg\min}\, M(W, \boldsymbol{w}) = \underset{W,\boldsymbol{w}}{\arg\min}\left[G(W, \boldsymbol{w}) + \alpha E(W, \boldsymbol{w})\right]$$

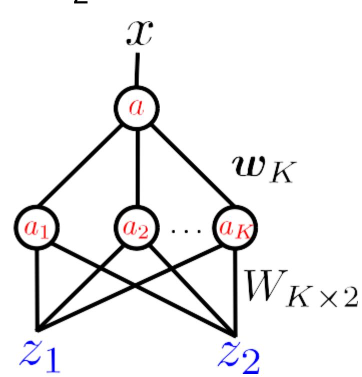
$$\frac{\mathrm{d}G(W,\boldsymbol{w})}{\mathrm{d}W_{ij}} = \sum_n \frac{\mathrm{d}G(W,\boldsymbol{w})}{\mathrm{d}x^{(n)}} \frac{\mathrm{d}x^{(n)}}{\mathrm{d}W_{ij}} = \sum_n \frac{\mathrm{d}G(W,\boldsymbol{w})}{\mathrm{d}x^{(n)}} \frac{\mathrm{d}x^{(n)}}{\mathrm{d}a^{(n)}} \frac{\mathrm{d}a^{(n)}}{\mathrm{d}W_{ij}}$$

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

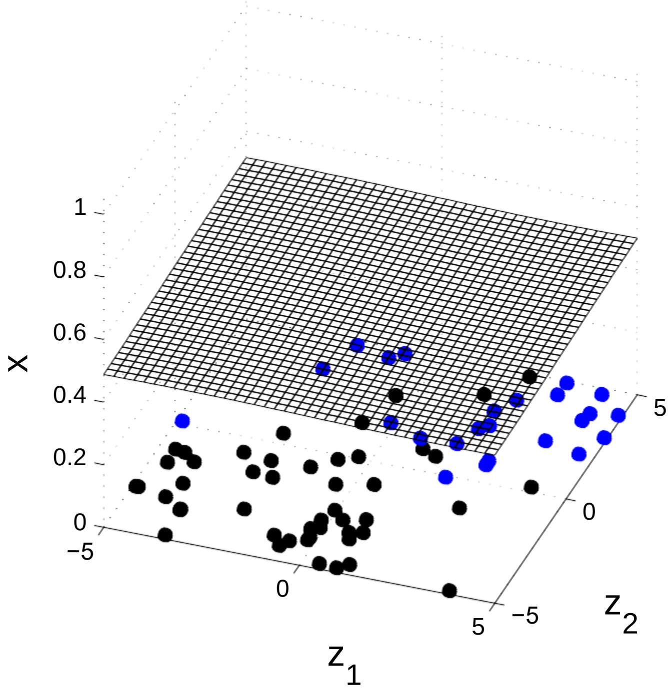
Training a Neural Network with a Single Hidden Layer

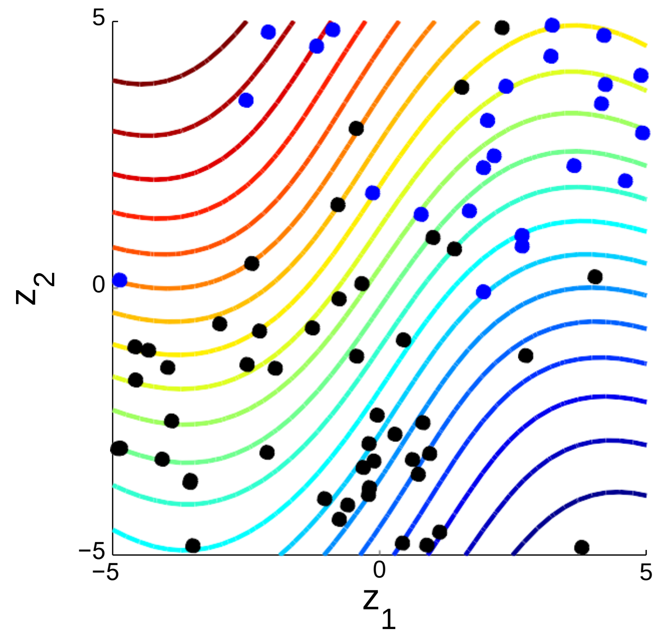


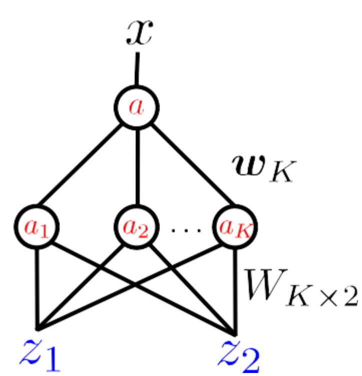




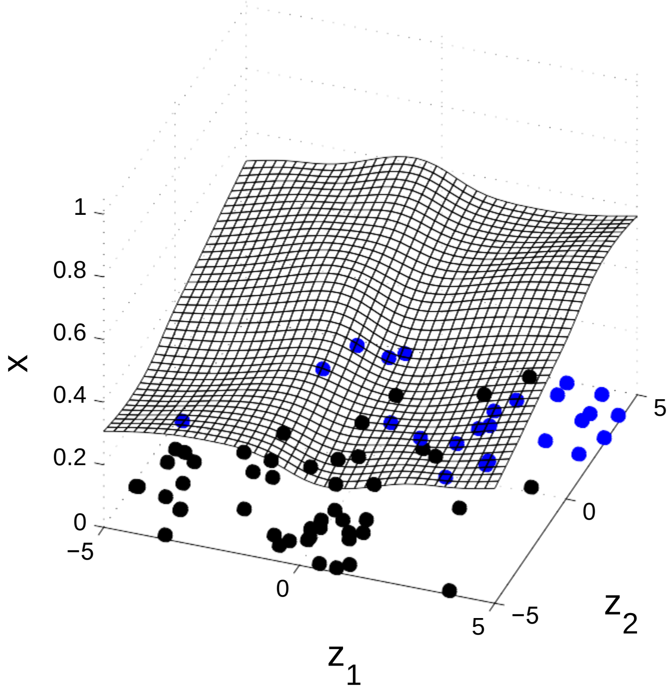
Training a Neural Network with a Single Hidden Layer

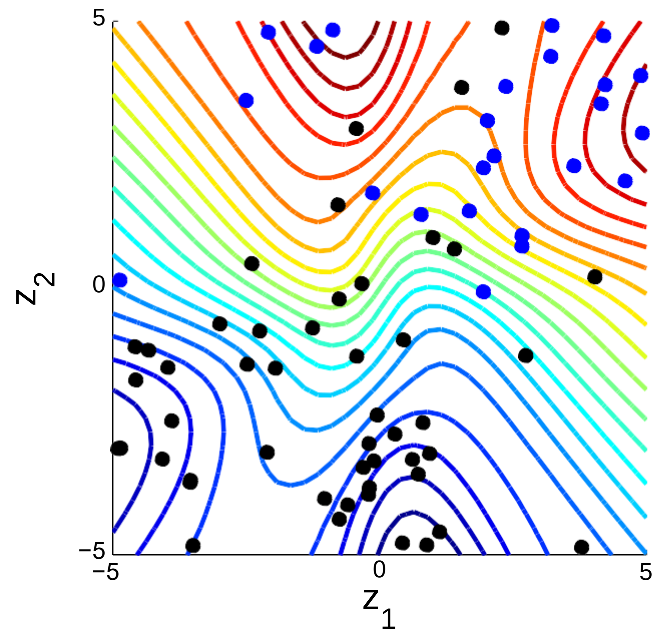


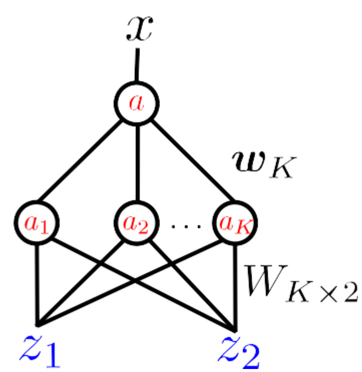




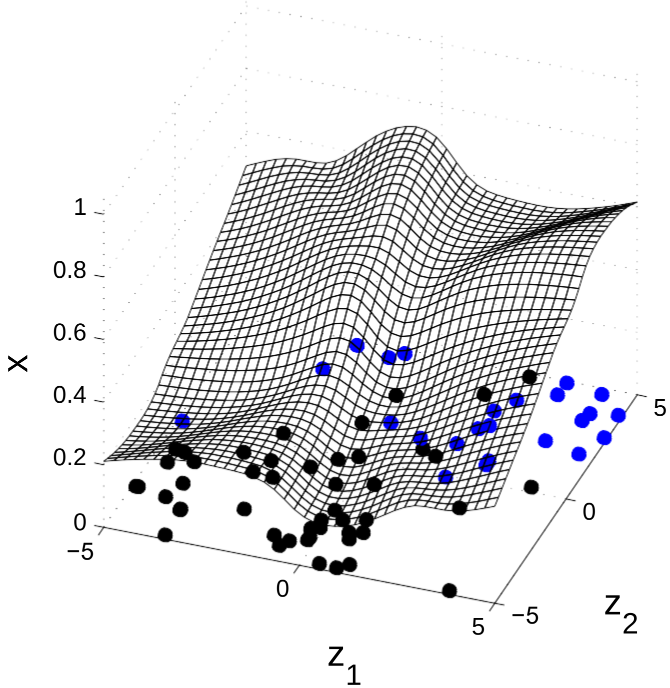
Training a Neural Network with a Single Hidden Layer

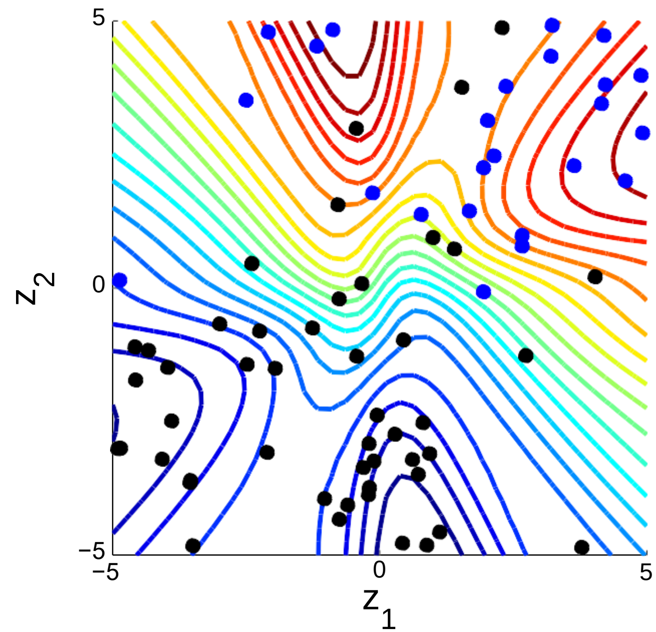


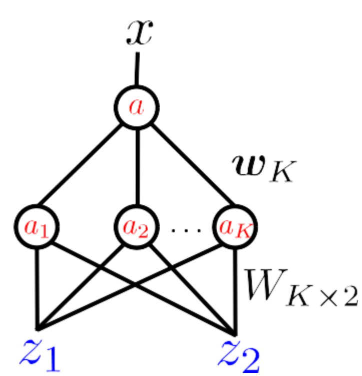




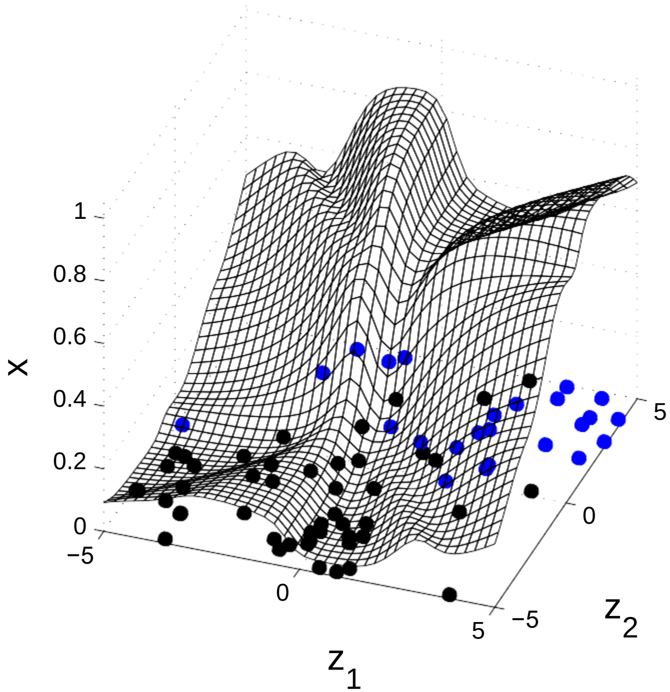
Training a Neural Network with a Single Hidden Layer

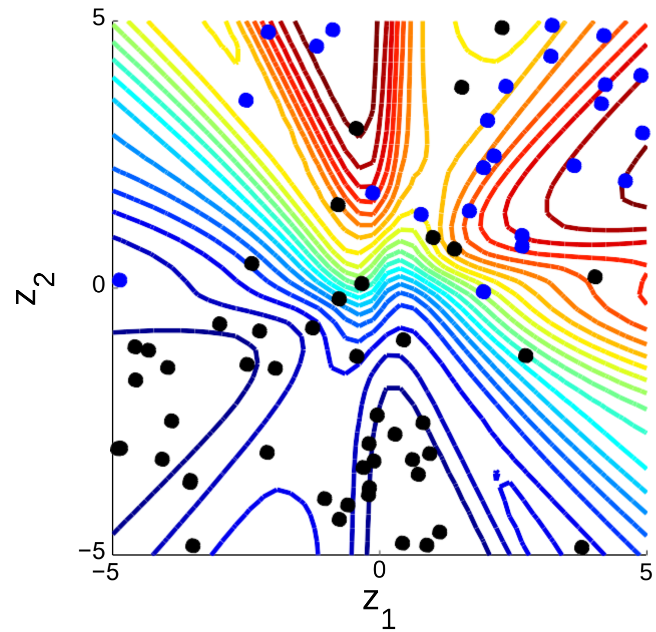


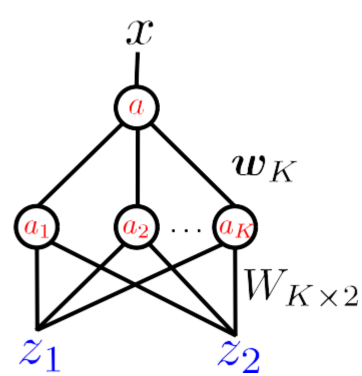




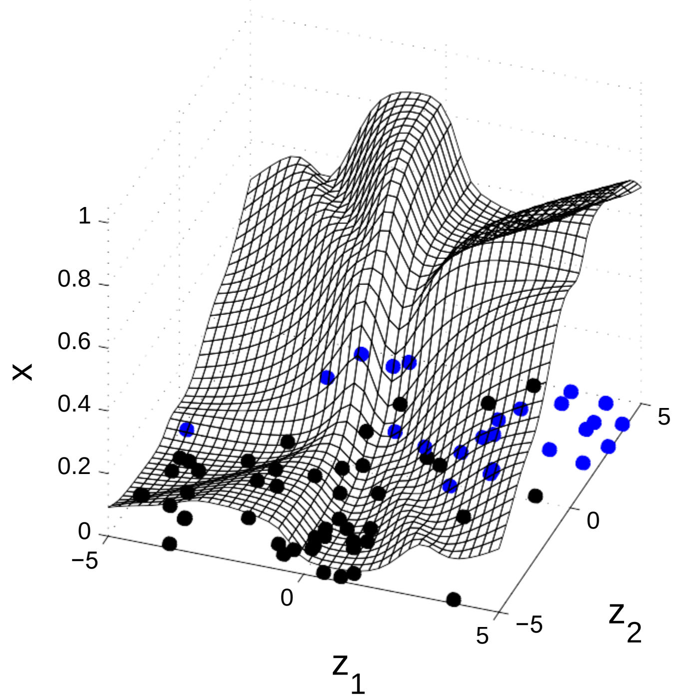
Training a Neural Network with a Single Hidden Layer

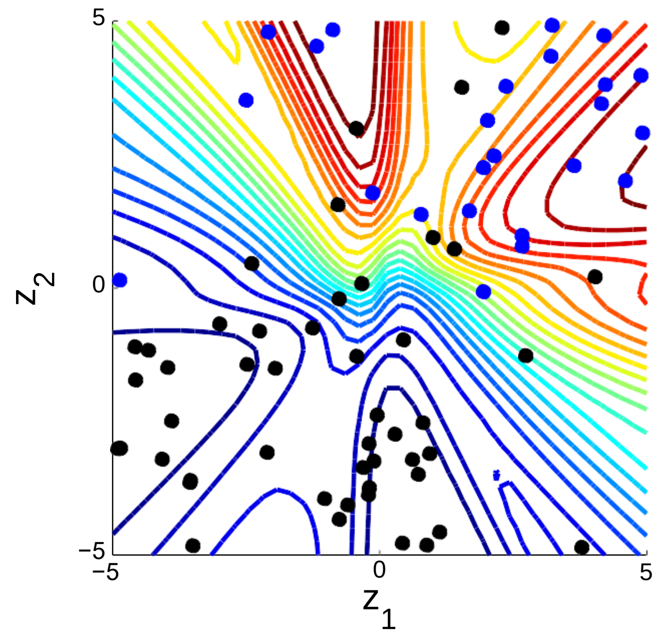


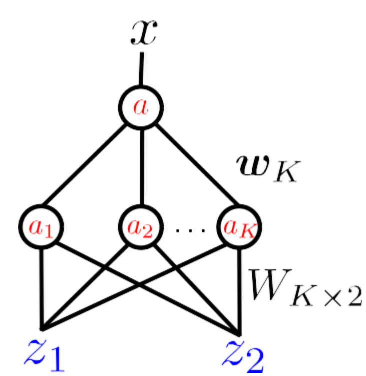




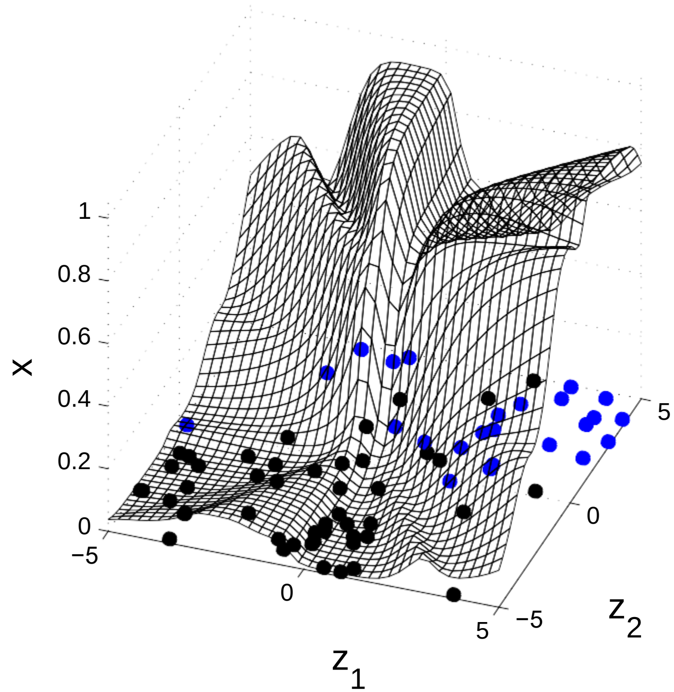
Training a Neural Network with a Single Hidden Layer

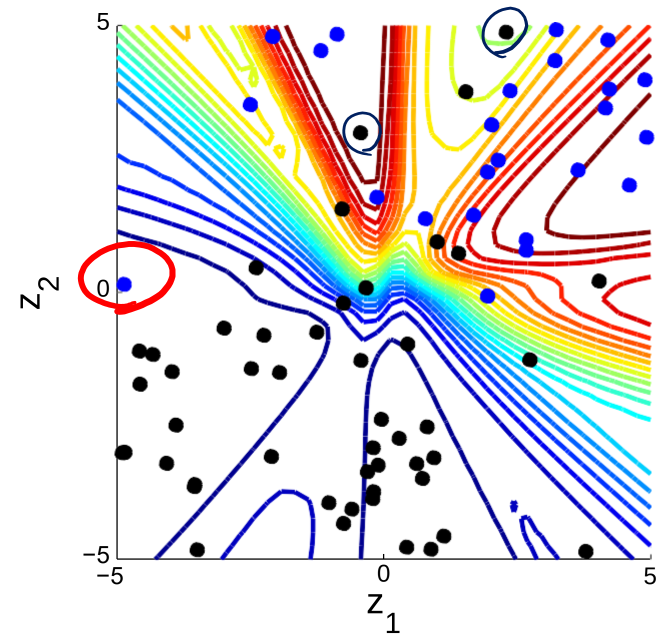


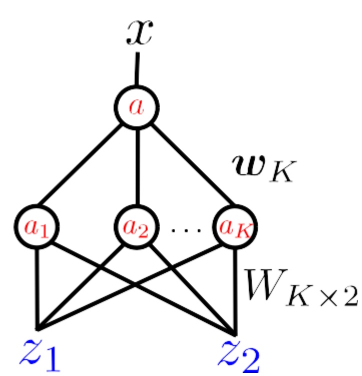




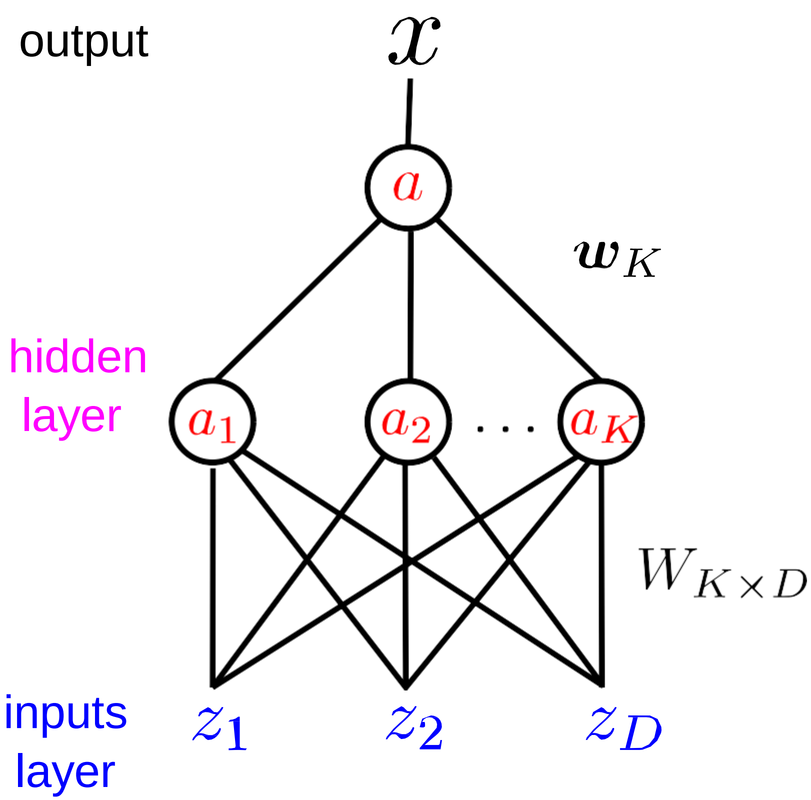
Training a Neural Network with a Single Hidden Layer

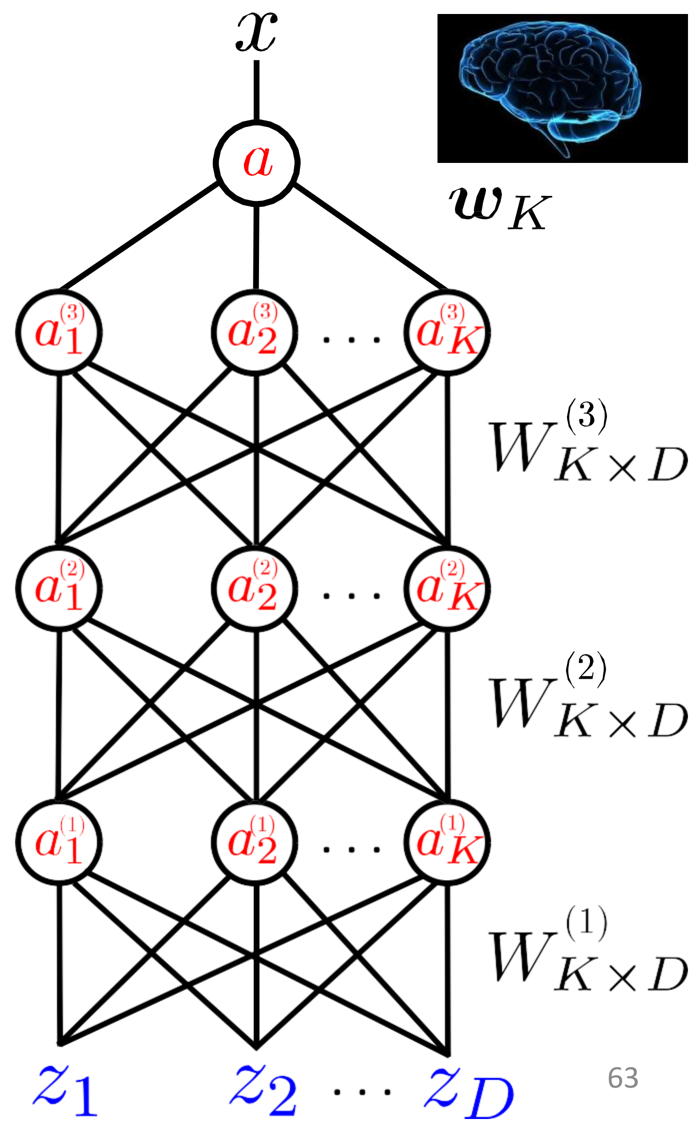




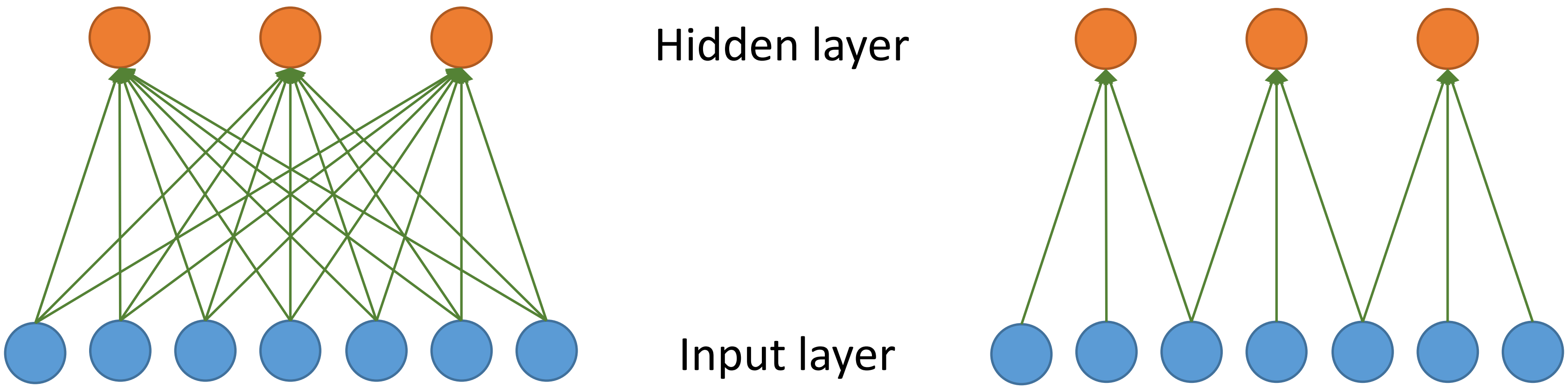


Hierarchical Models with Many Layers





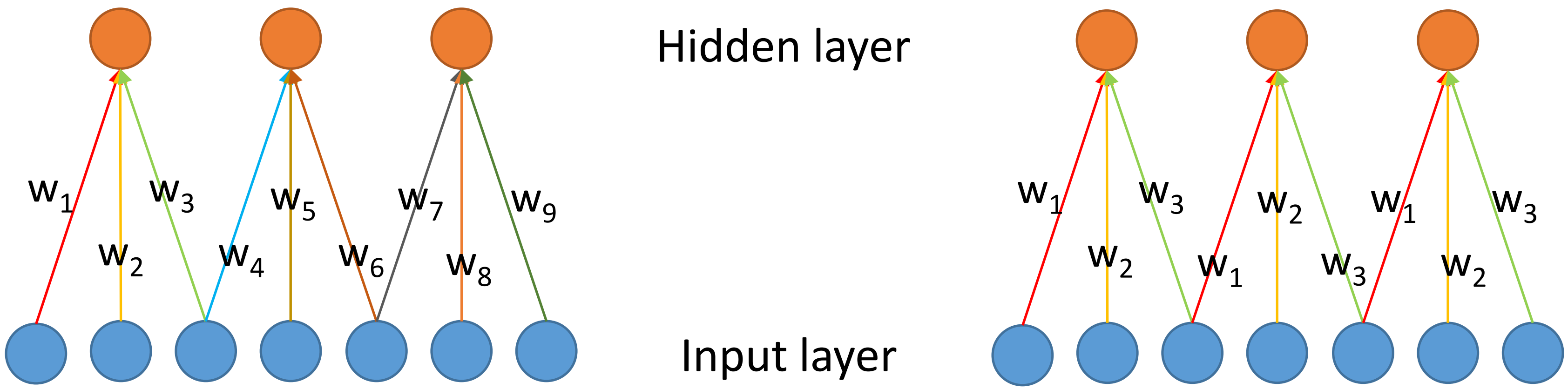
Convolutional Neural Networks (CNN): Local Connectivity



Global connectivity **Local** connectivity

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

Convolutional Neural Networks (CNN): Weight Sharing

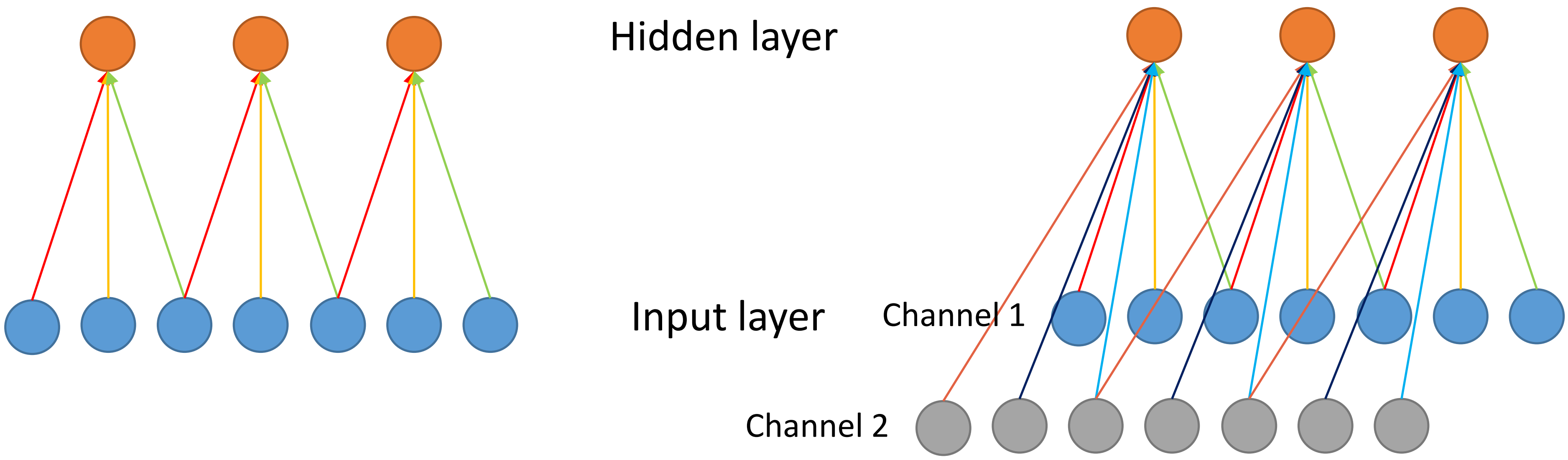


Without weight sharing

With weight sharing

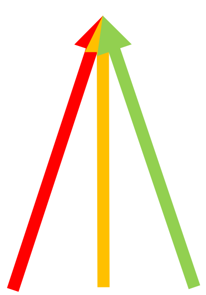
- # input units (neurons): 7
- # hidden units: 3
- Number of parameters
  - Without weight sharing: 9
  - With weight sharing : 3

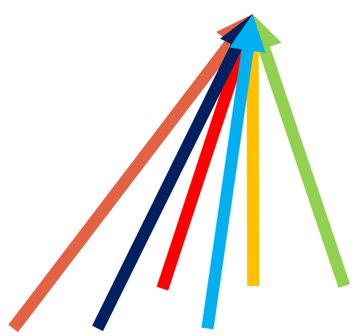
CNN with Multiple Input Channels



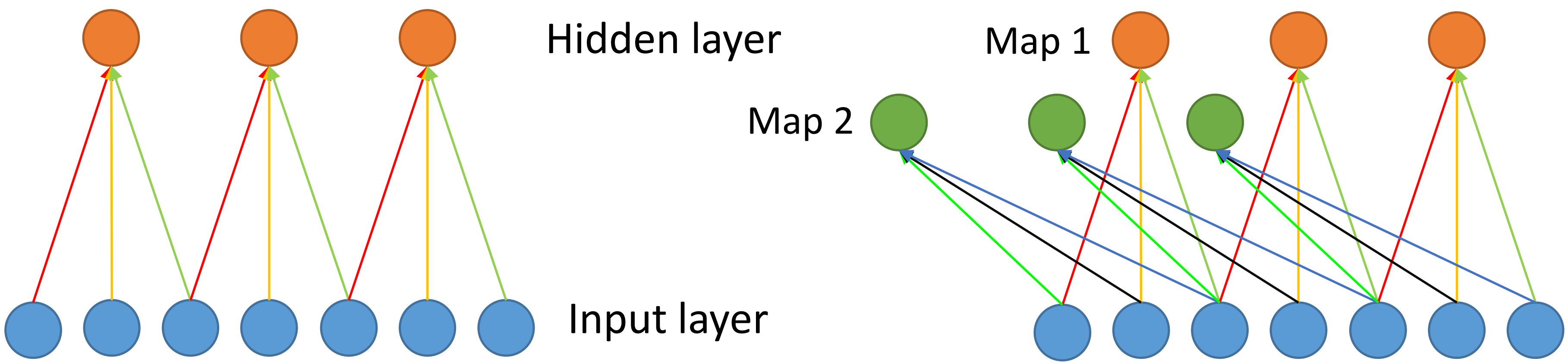
Single input channel

Multiple input channels

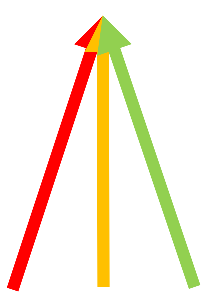




CNN with Multiple Output Maps

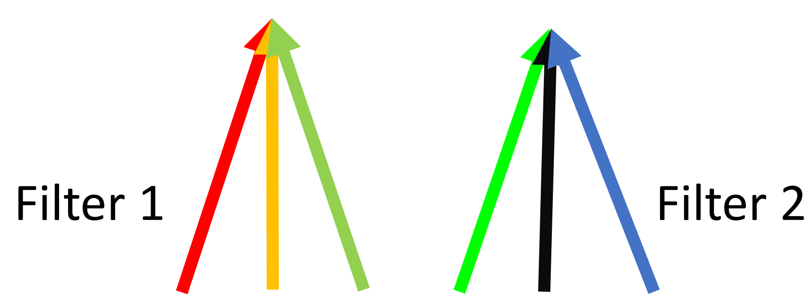


Single output map



Filter weights

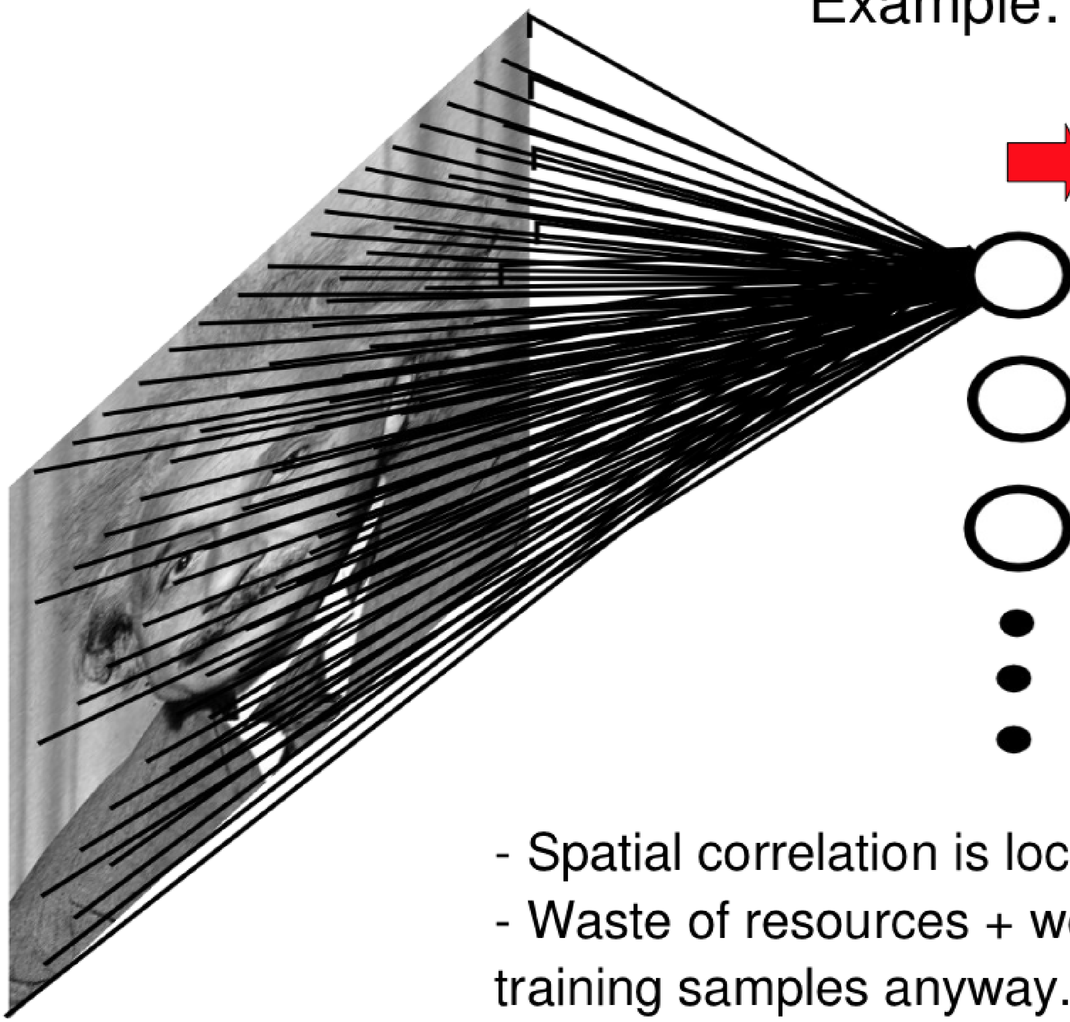
Multiple output maps



Filter weights

Generalized to 2D Cases:

Fully Connected Layer



Example: 200x200 image
40K hidden units
~2B parameters!!!

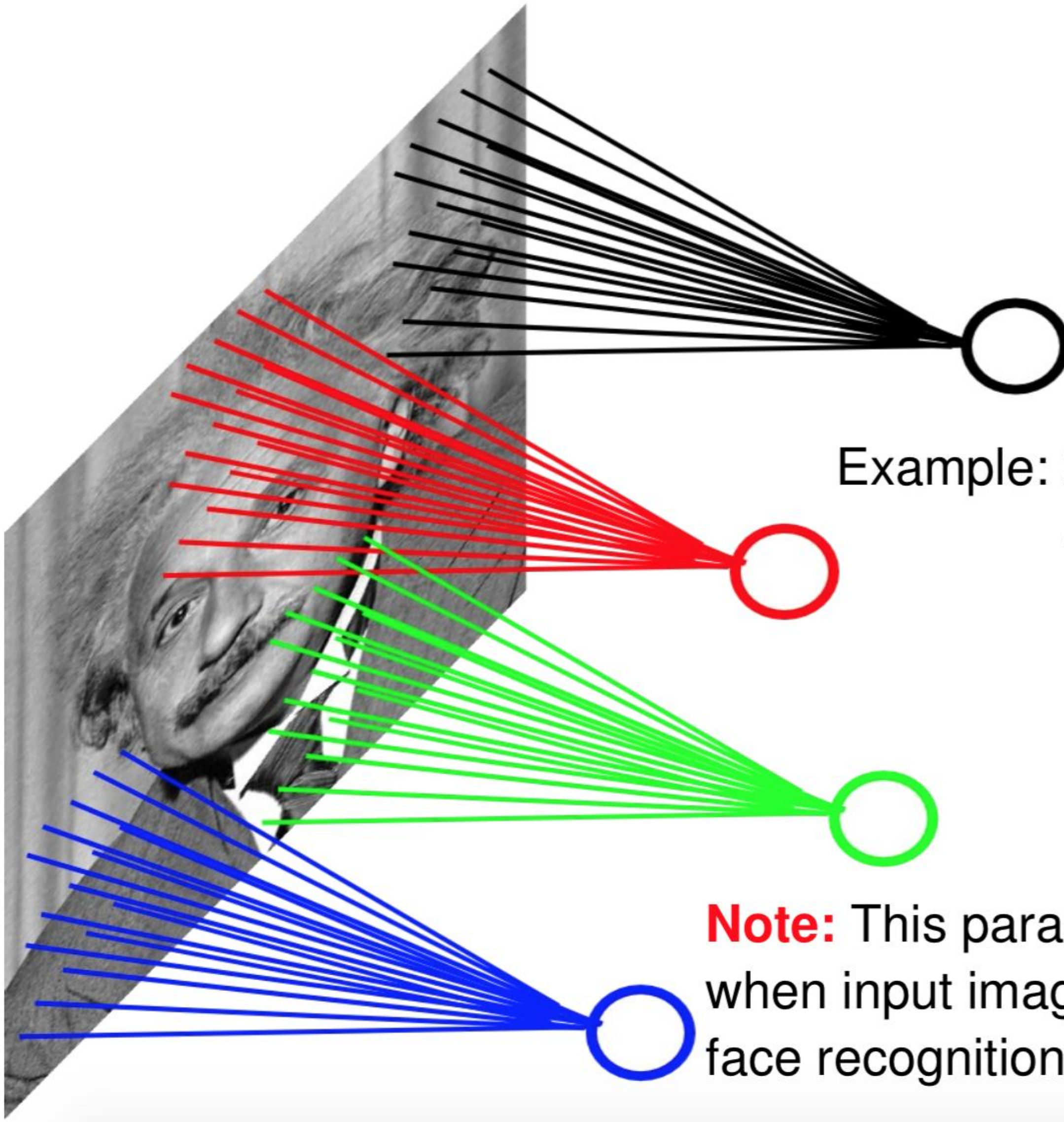
- Spatial correlation is local
- Waste of resources + we have not enough training samples anyway..

Ranzato

Ref: Marc'Aurelio Ranzato, Tutorial in CVPR2014

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).

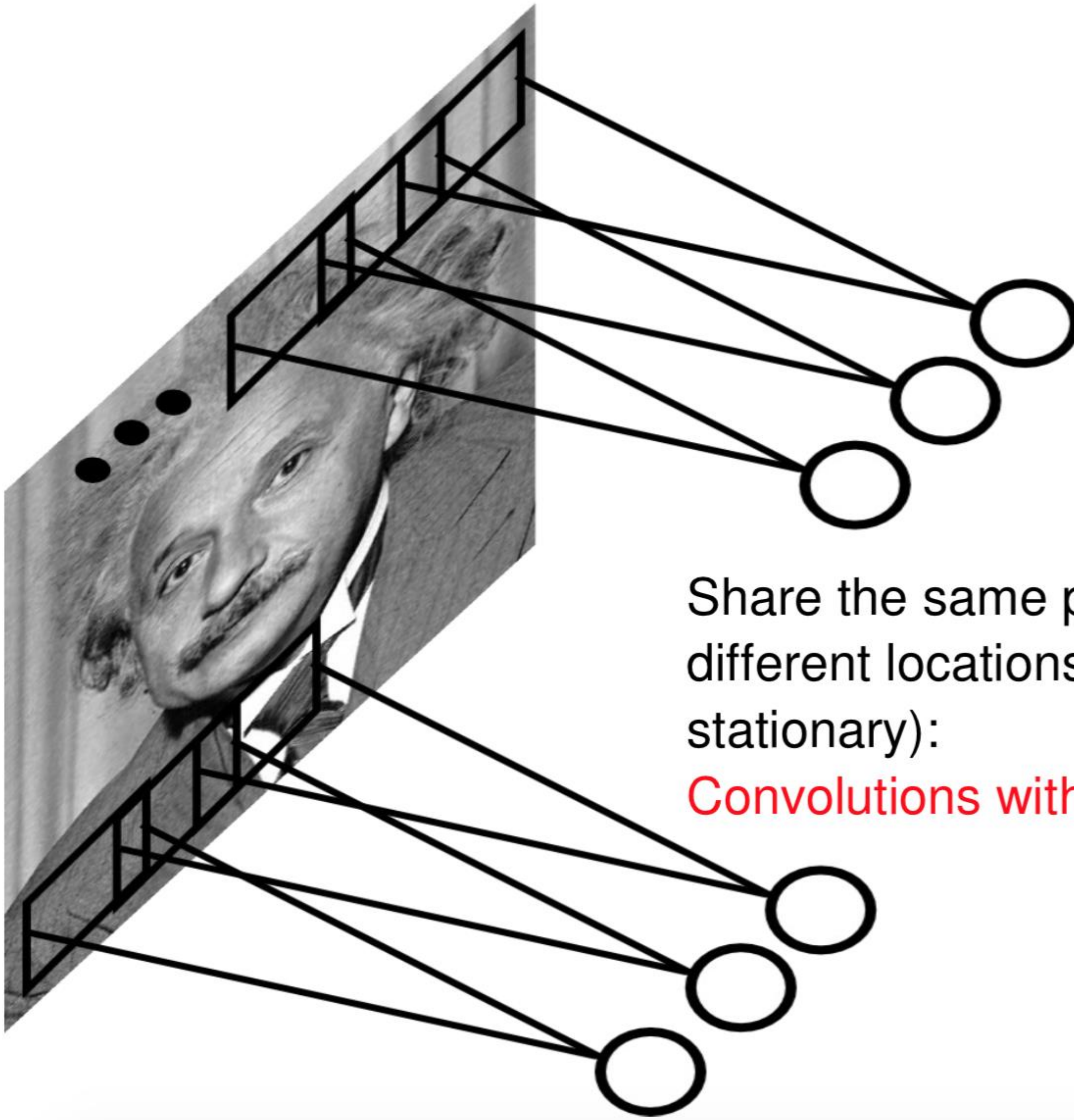
34

Ranzato

Ref: Marc'Aurelio Ranzato, Tutorial in CVPR2014

Generalized to 2D Cases:

Convolutional Layer



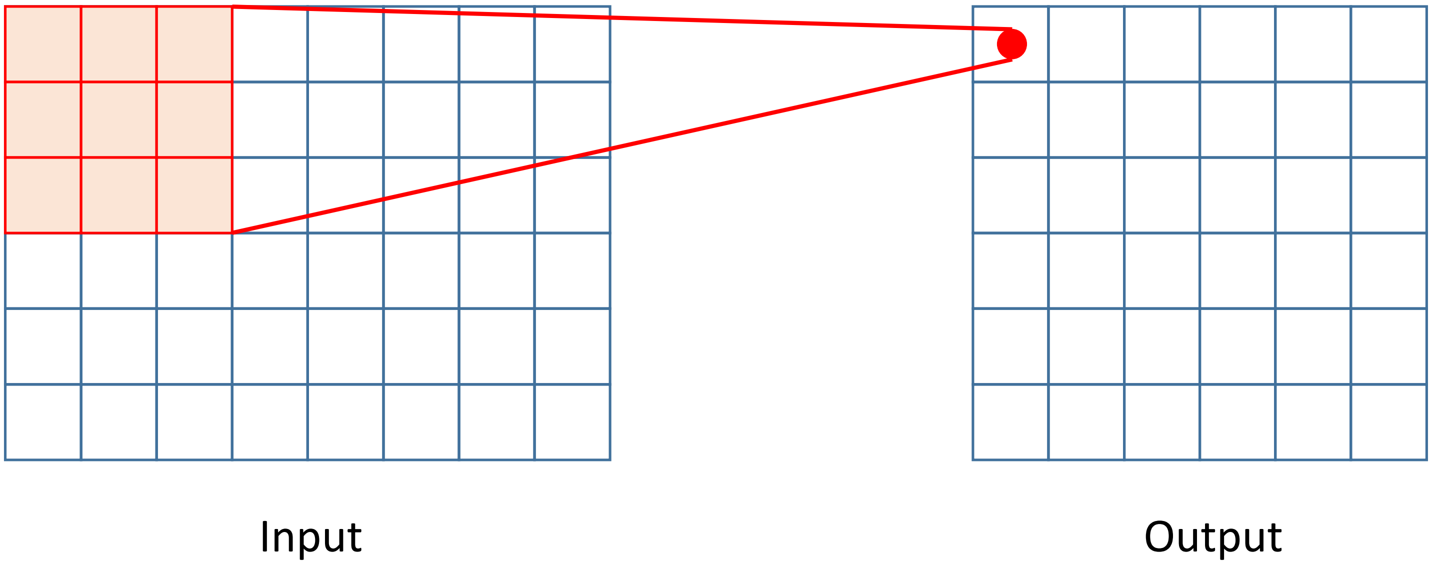
Share the same parameters across different locations (assuming input is stationary):
Convolutions with learned kernels

36

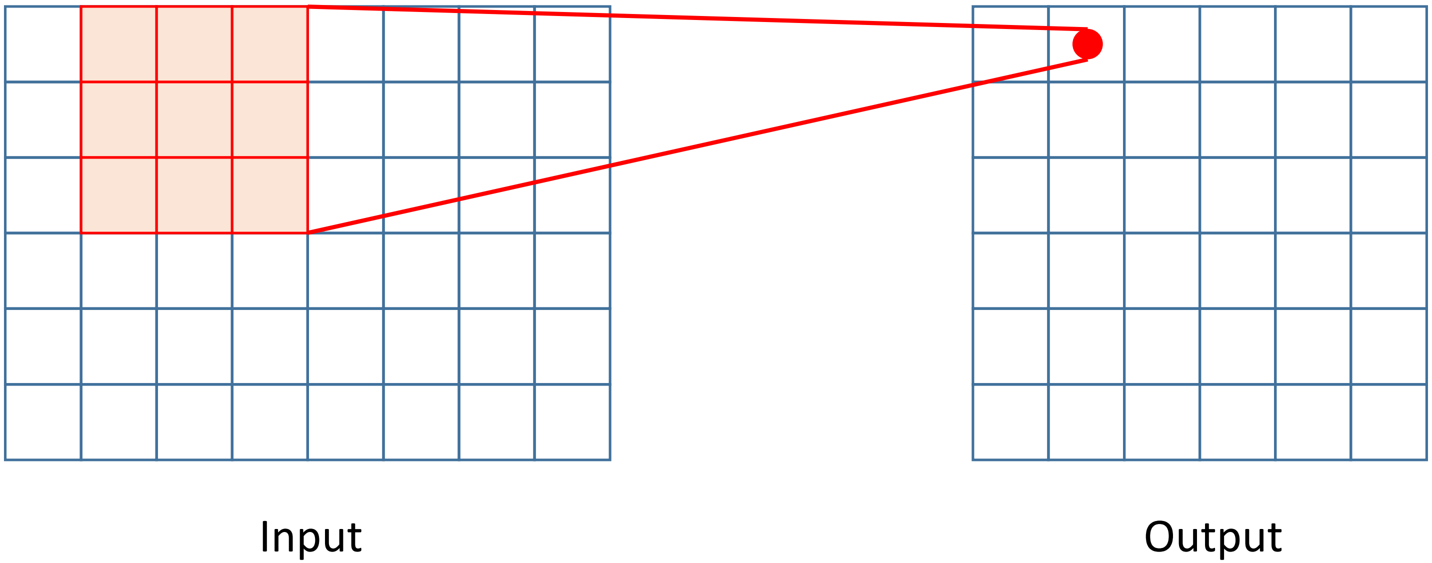
Ranzato

Ref: Marc'Aurelio Ranzato, Tutorial in CVPR2014

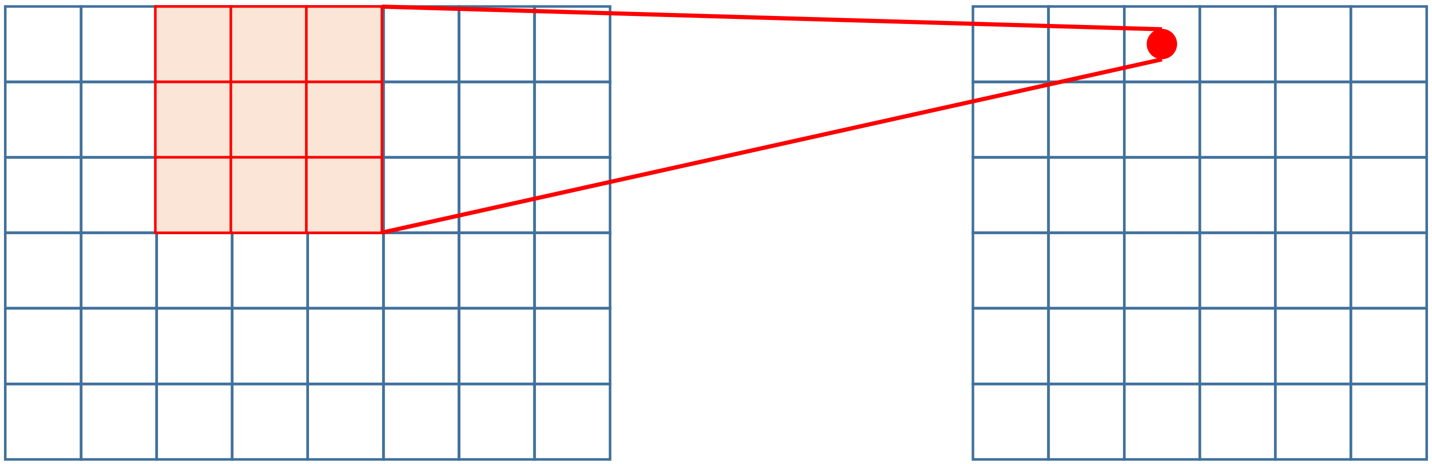
Convolutional Layer



Convolutional Layer



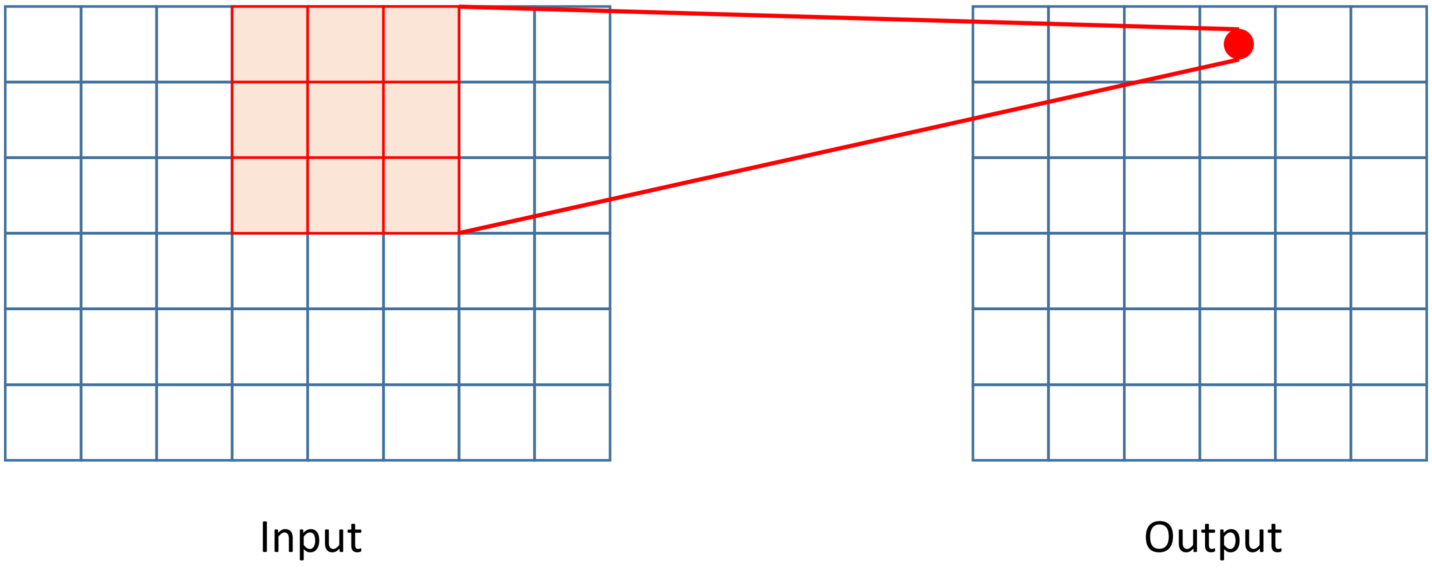
Convolutional Layer



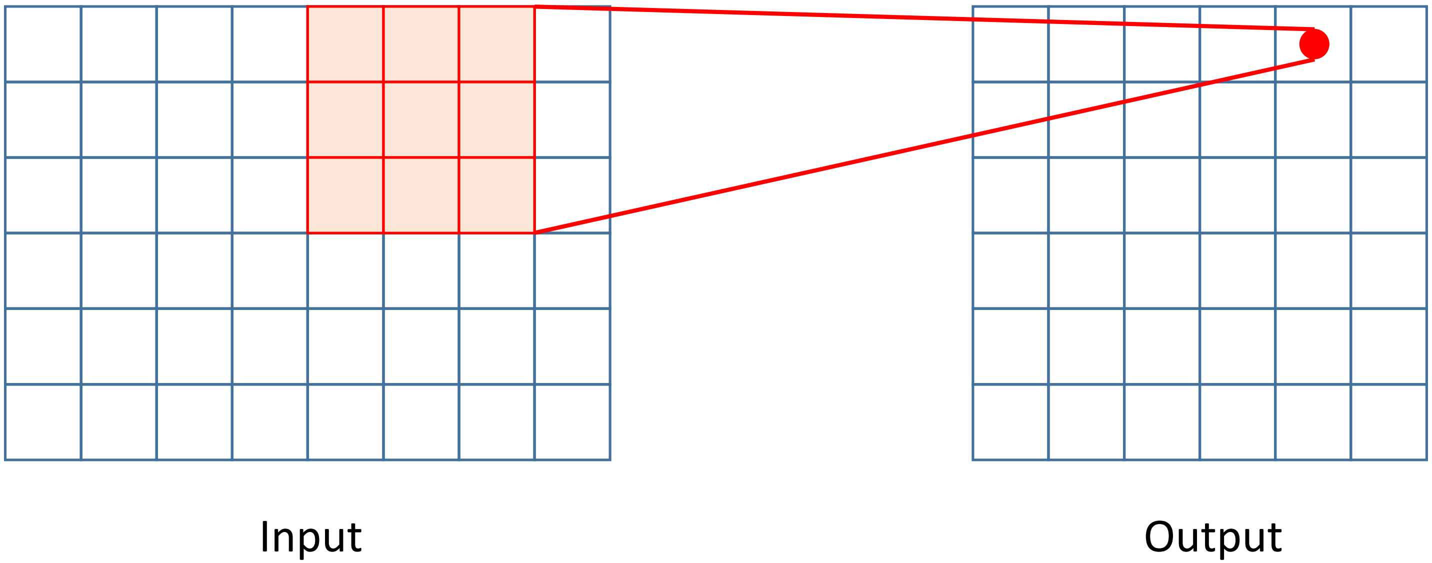
Input

Output

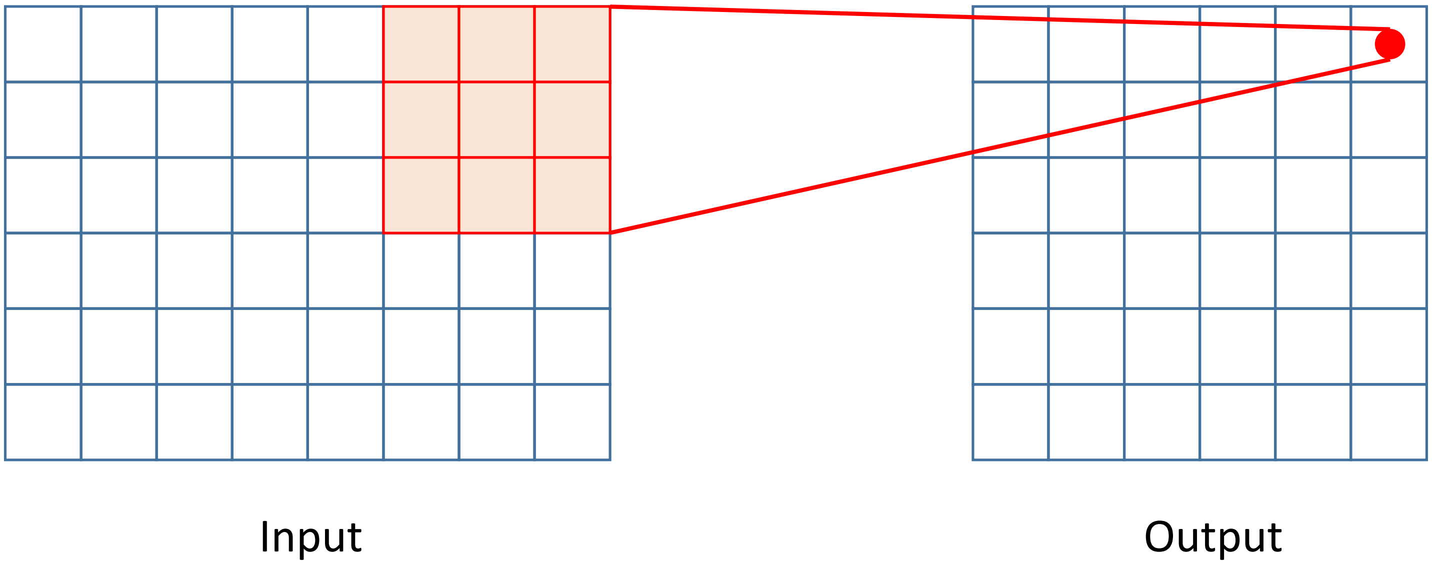
Convolutional Layer



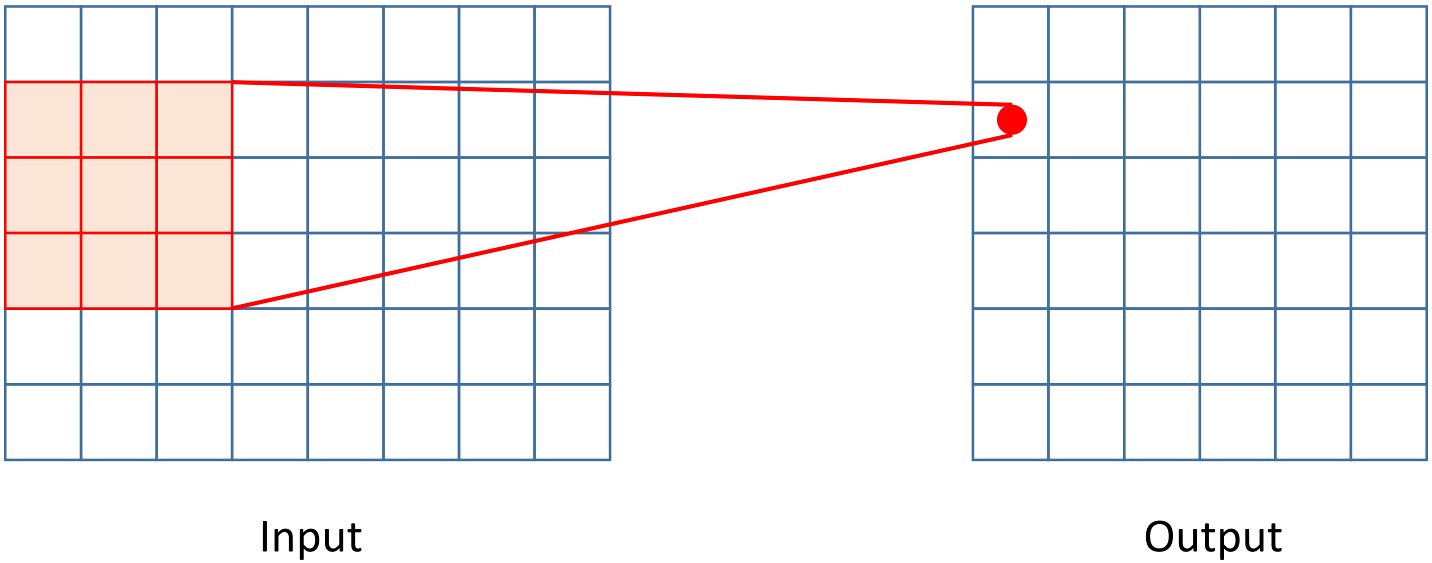
Convolutional Layer



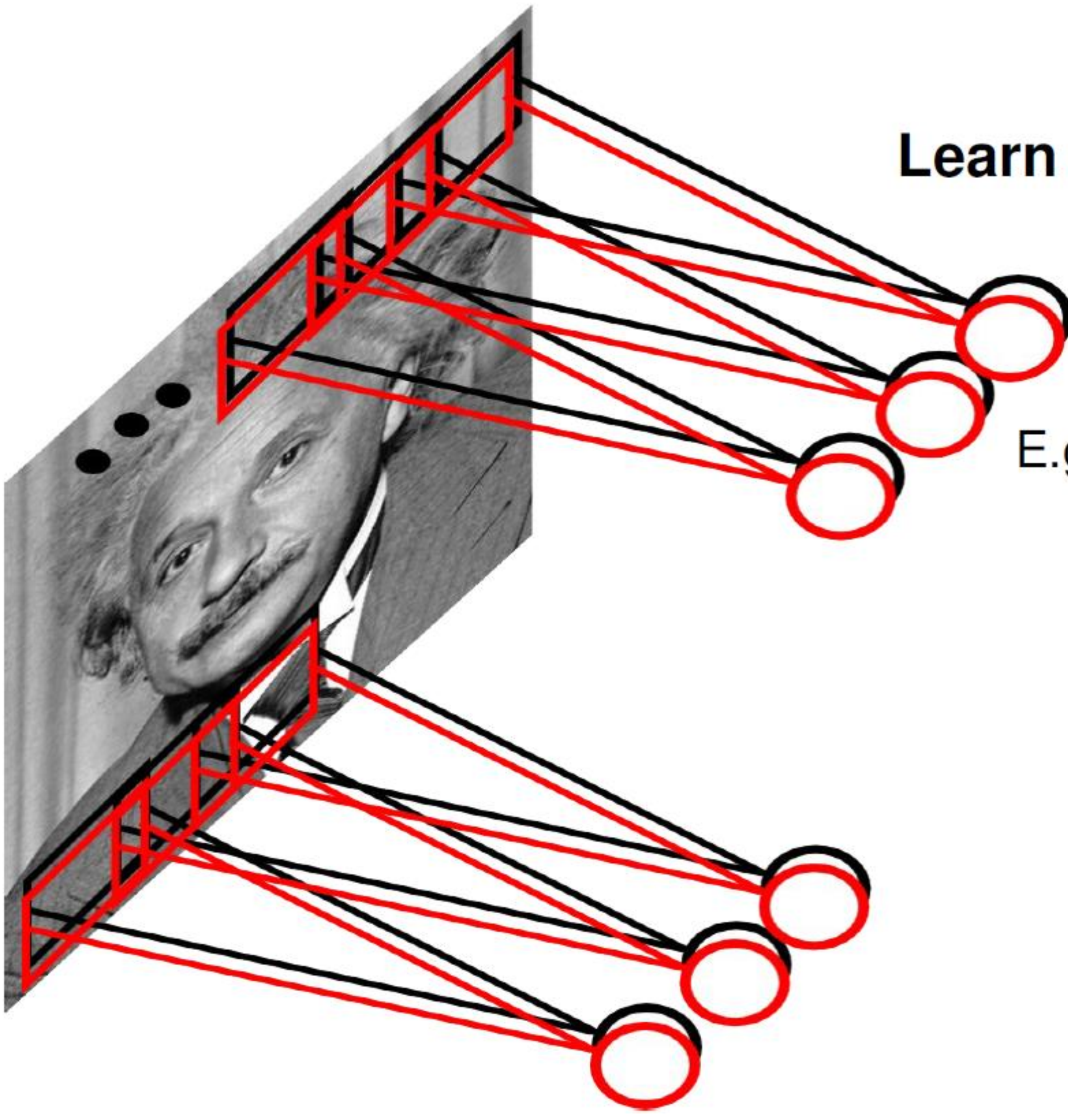
Convolutional Layer



Convolutional Layer



Convolutional Layer



Learn multiple filters.

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

61

Ranzato

Ref: Marc'Aurelio Ranzato, Tutorial in CVPR2014

Putting them together → CNN

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

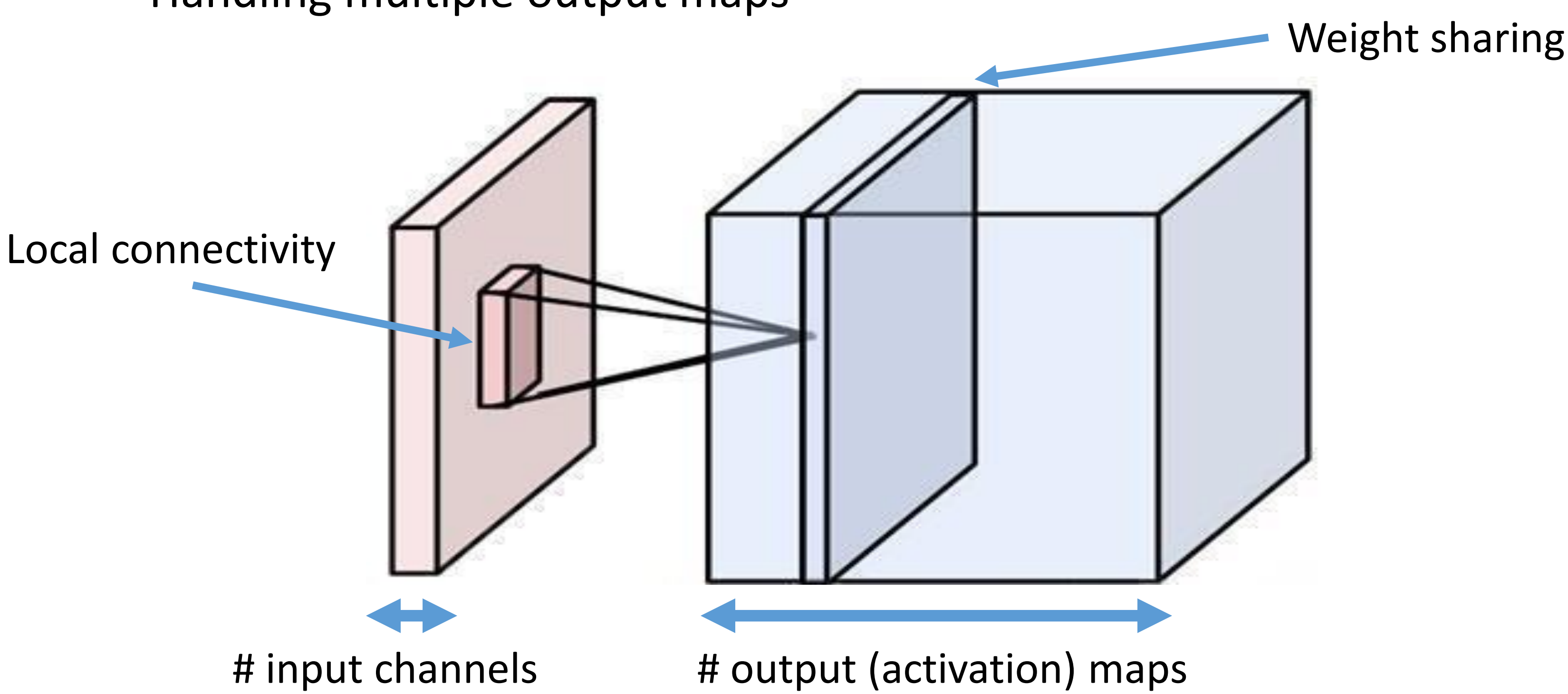
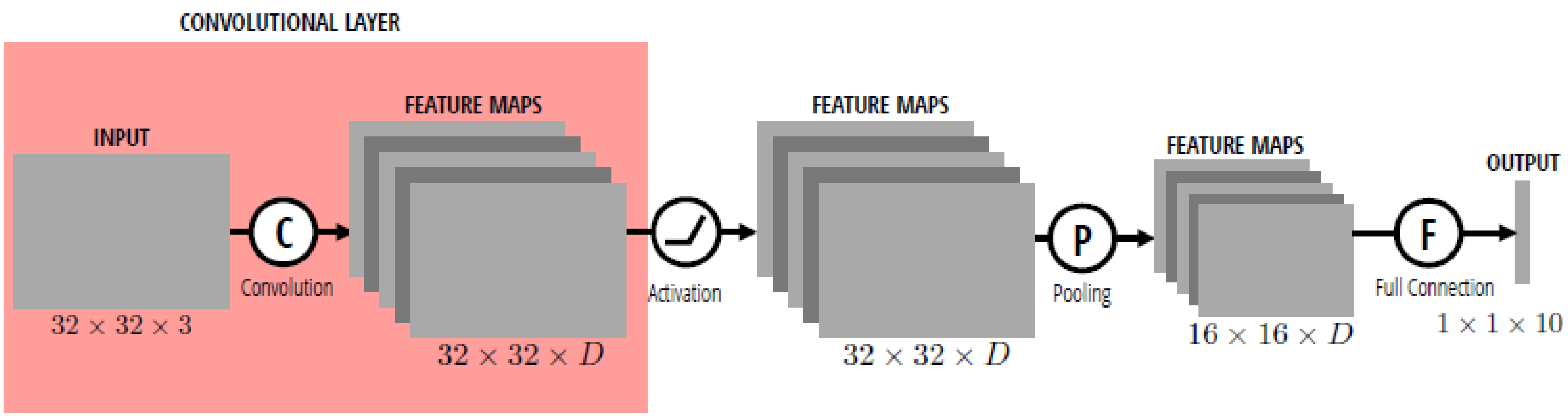


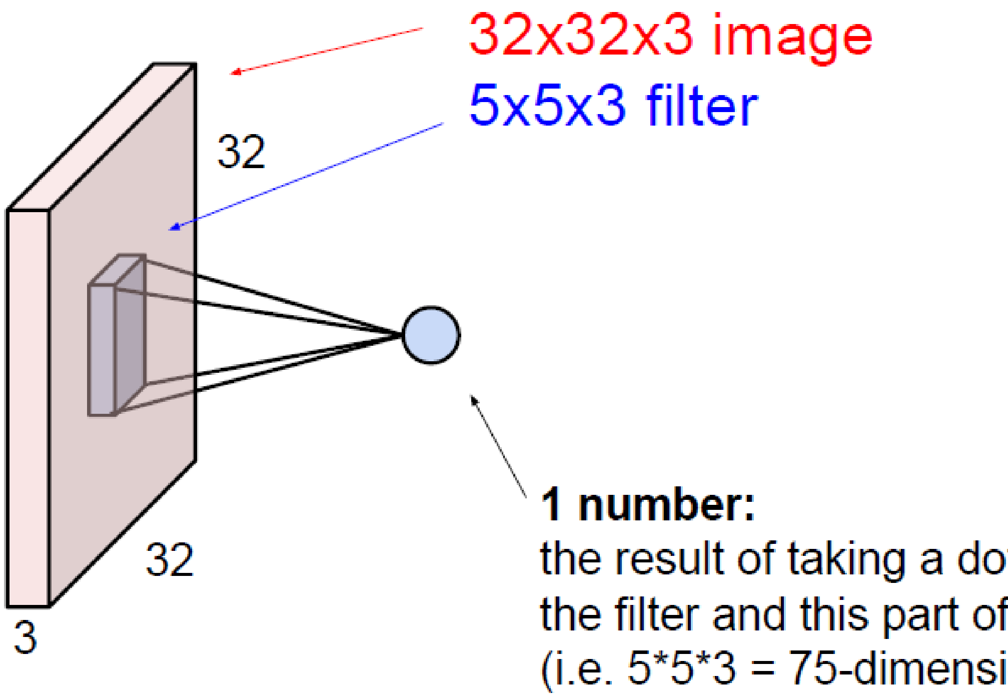
Image credit: A. Karpathy

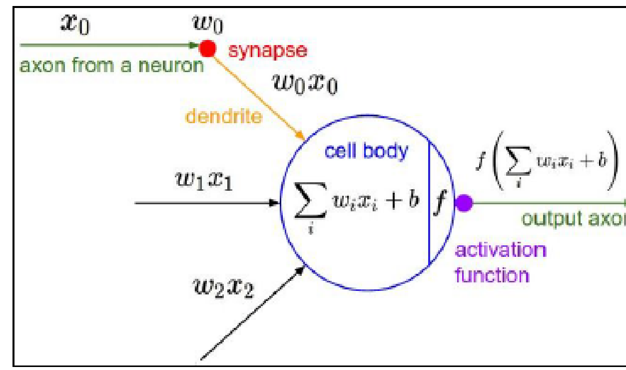
Convolution Layer in CNN



Putting them together (cont'd)

- The brain/neuron view of CONV layer

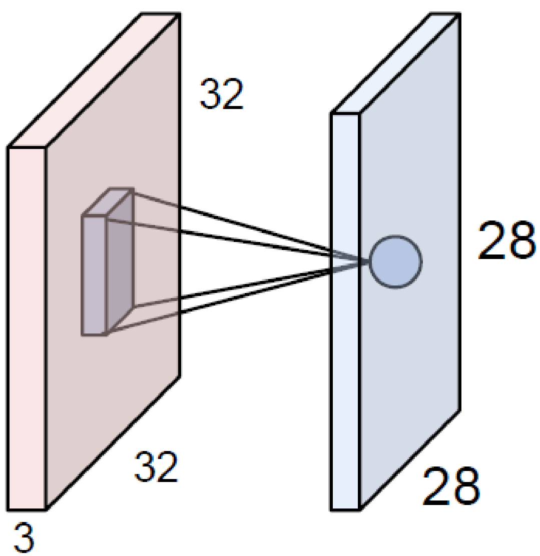




It's just a neuron with local connectivity...

Putting them together (cont'd)

- The brain/neuron view of CONV layer



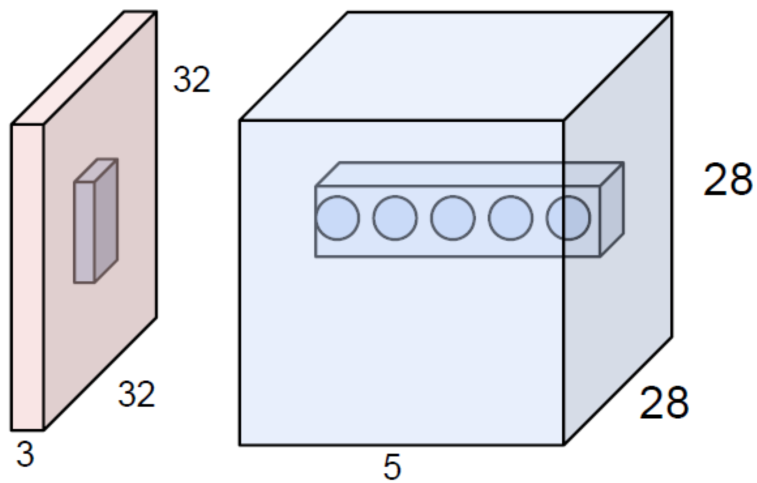
An activation map is a 28x28 sheet of neuron outputs:

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

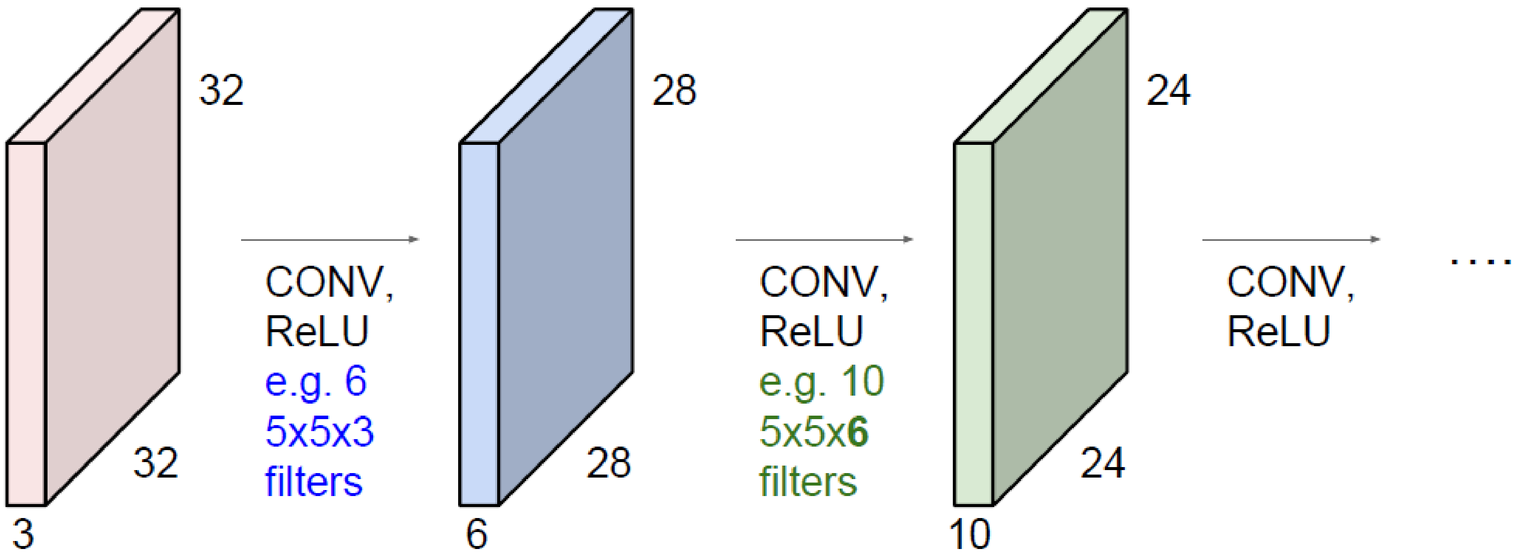


E.g. with 5 filters,
CONV layer consists of
neurons arranged in a 3D grid
(28x28x5)

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

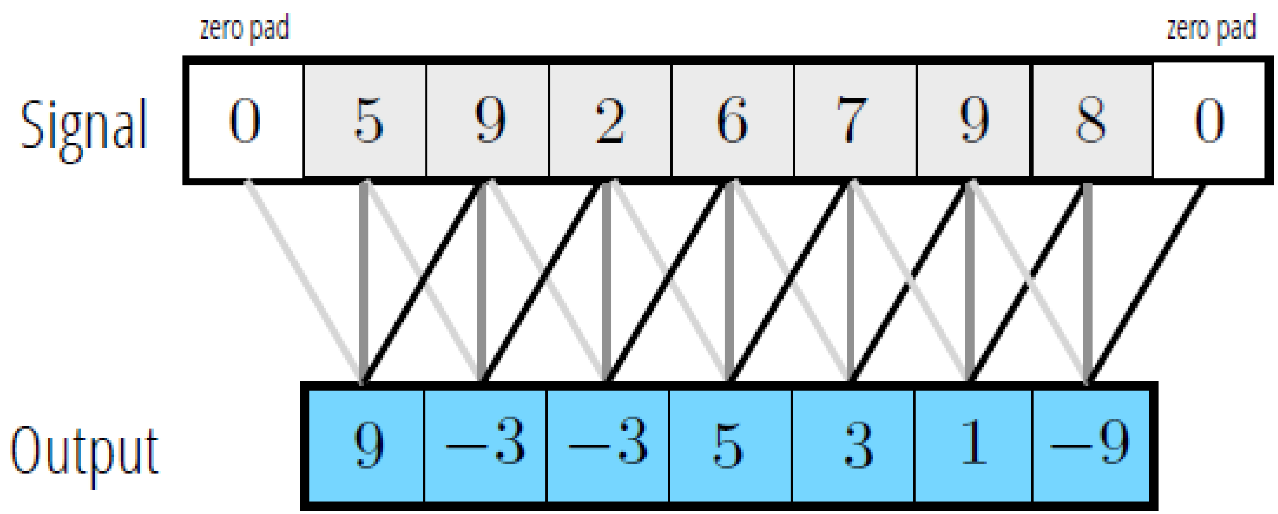
Putting them together (cont'd)

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



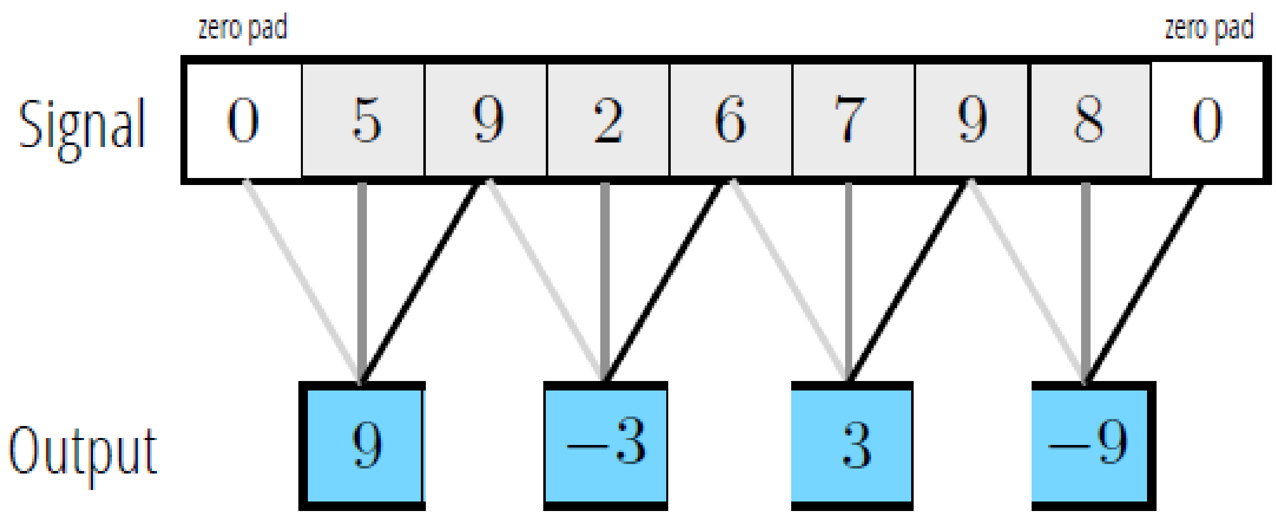
Variations of Convolution

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



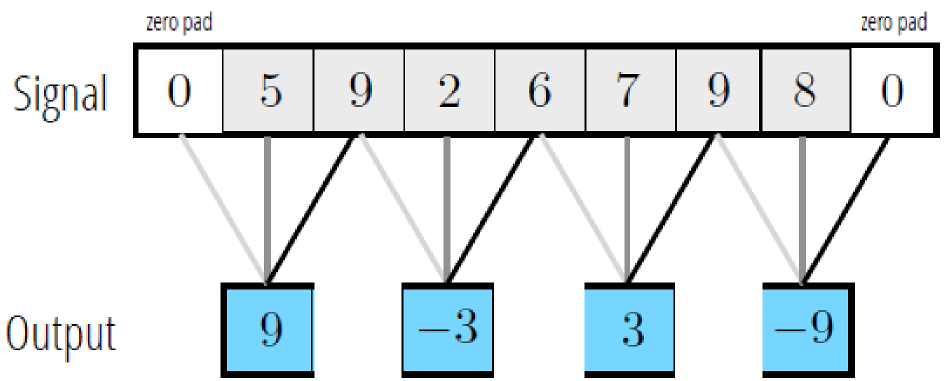
Variations of Convolution

- Stride
  - Step size across signals



Variations of Convolution

- Stride
  - Step size across signals



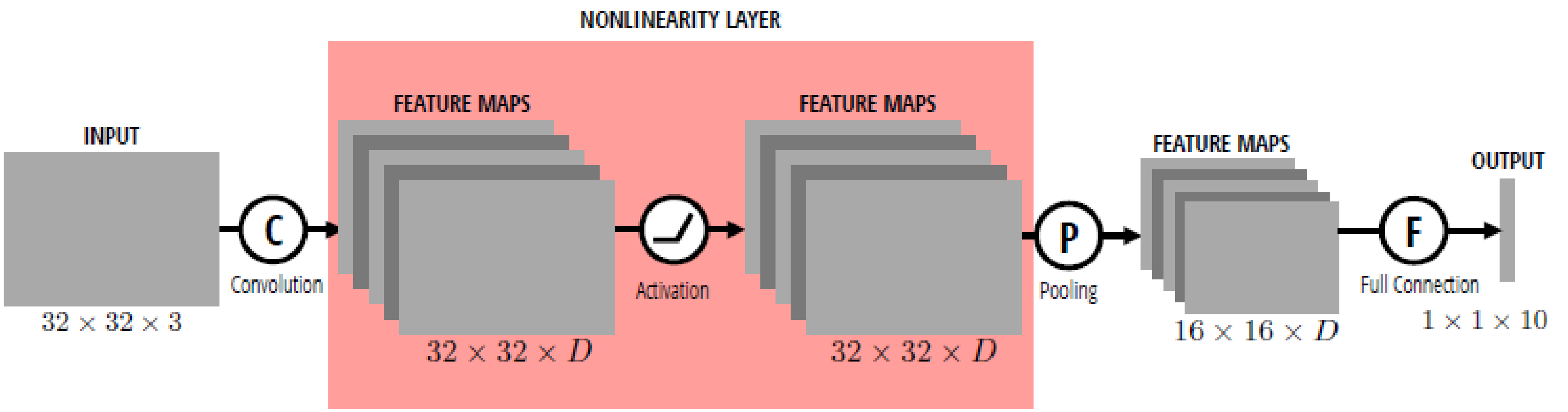
Input Size

Filter Size

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

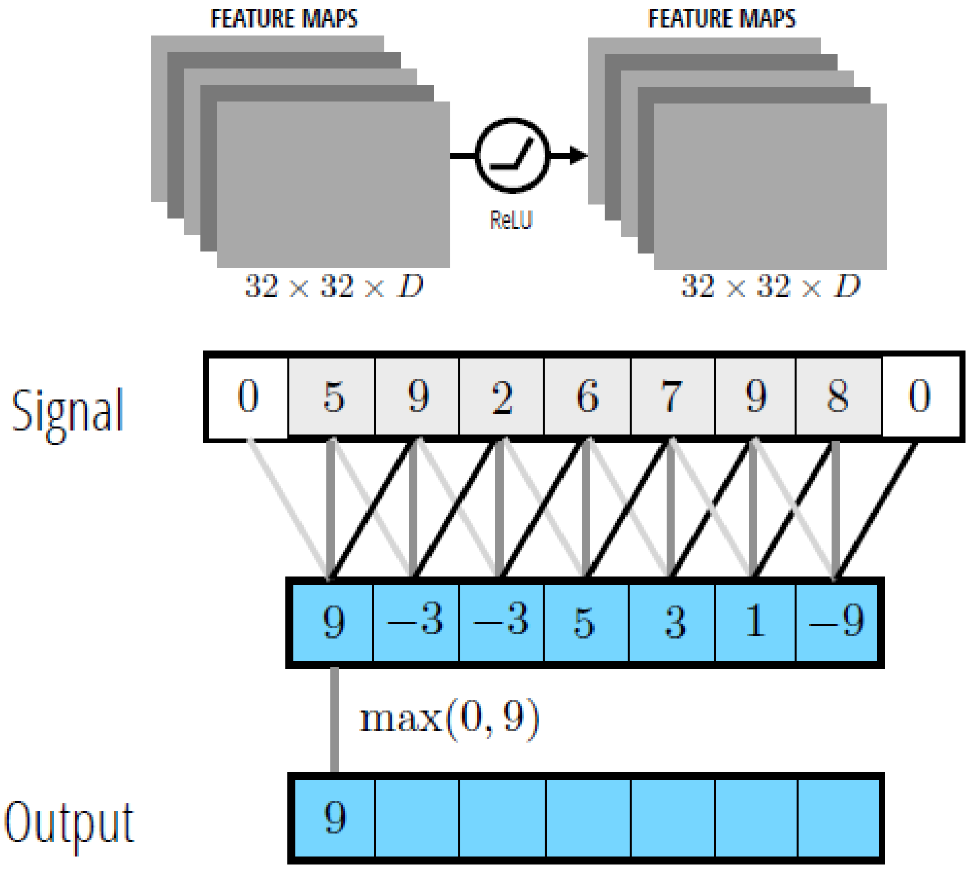
Stride: step size across the signal

Nonlinearity Layer in CNN



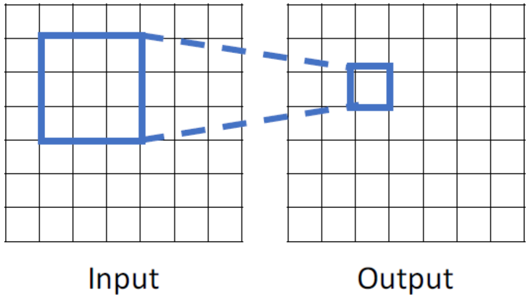
Nonlinearity Layer

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

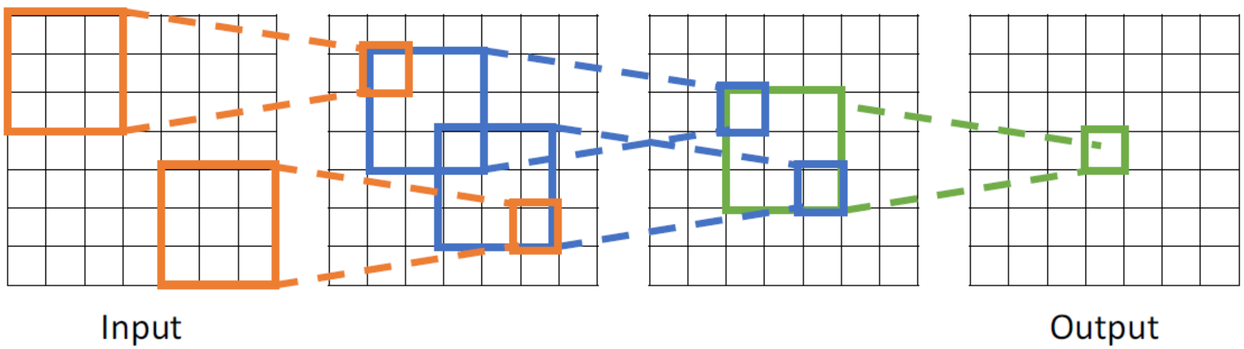


Receptive Field

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



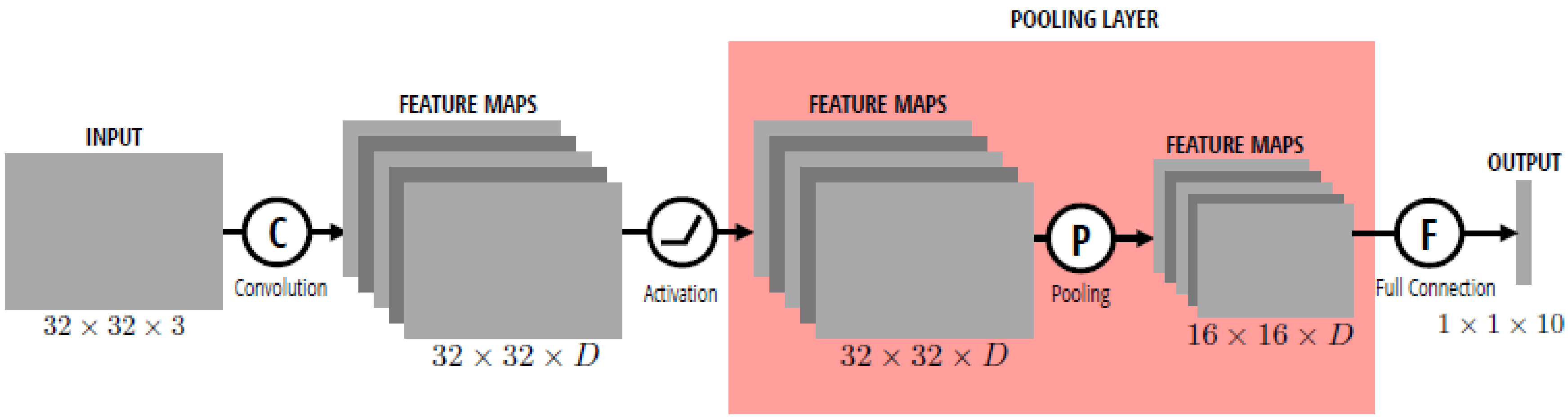
- Each successive convolution adds n-1 to the receptive field size.
  With a total of L layers, the receptive field size would be 1 + L * (n-1).



- 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)

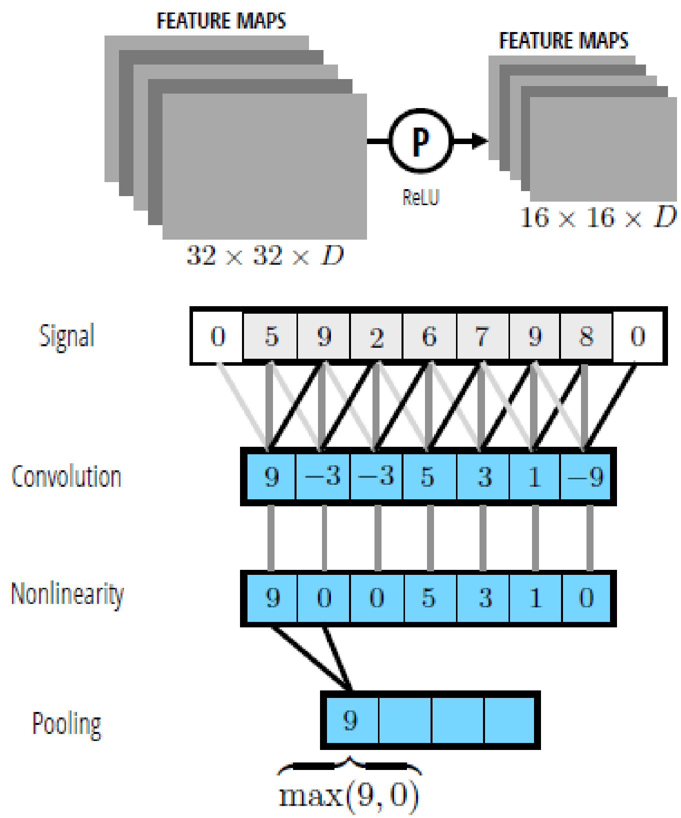
Slide credit: UMich EECS 498-007

Pooling Layer in CNN



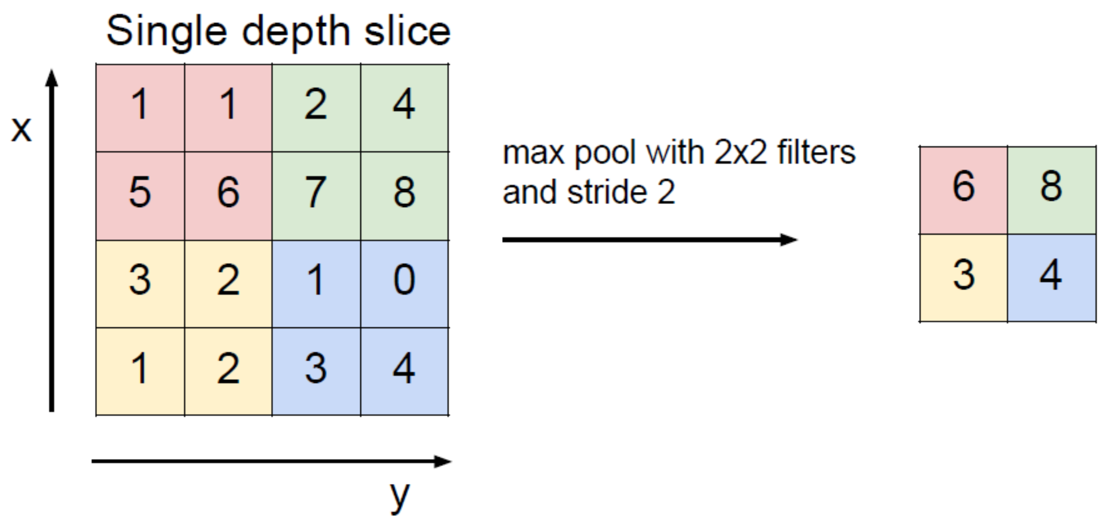
Pooling Layer

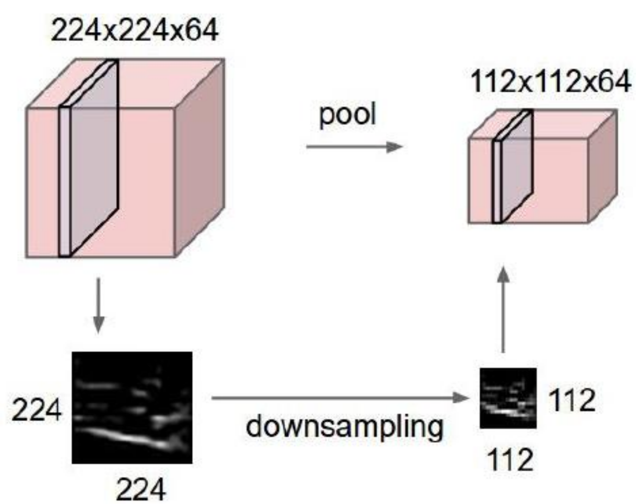
- Makes the representations smaller and more manageable
- Operates over each activation map independently
- E.g., Max Pooling



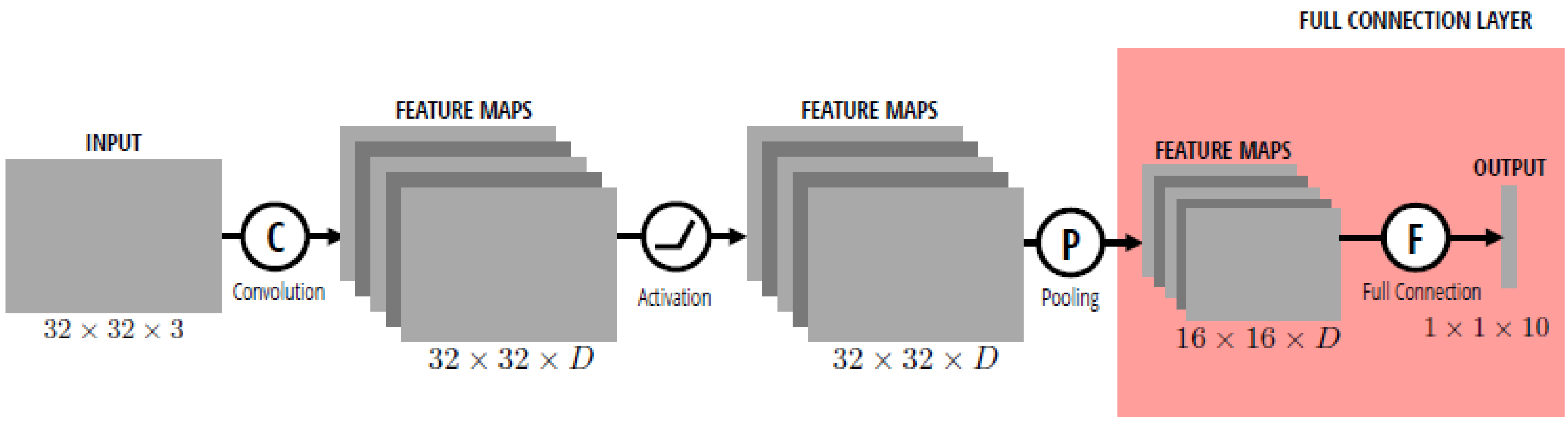
Pooling Layer for 2D Cases

- Reduces the spatial size and provides spatial invariance



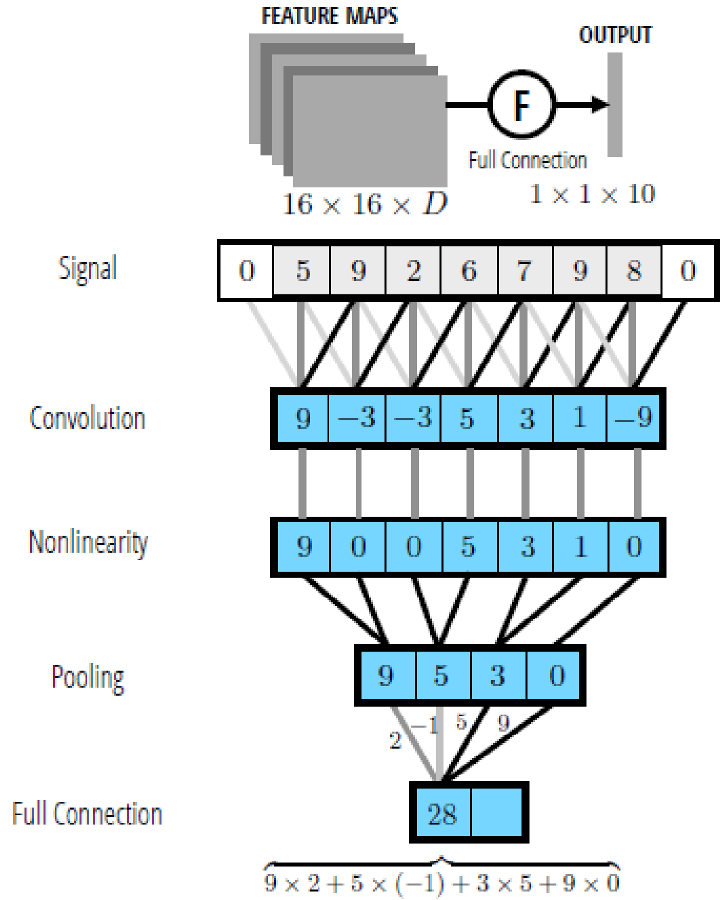


Fully Connected (FC) Layer in CNN



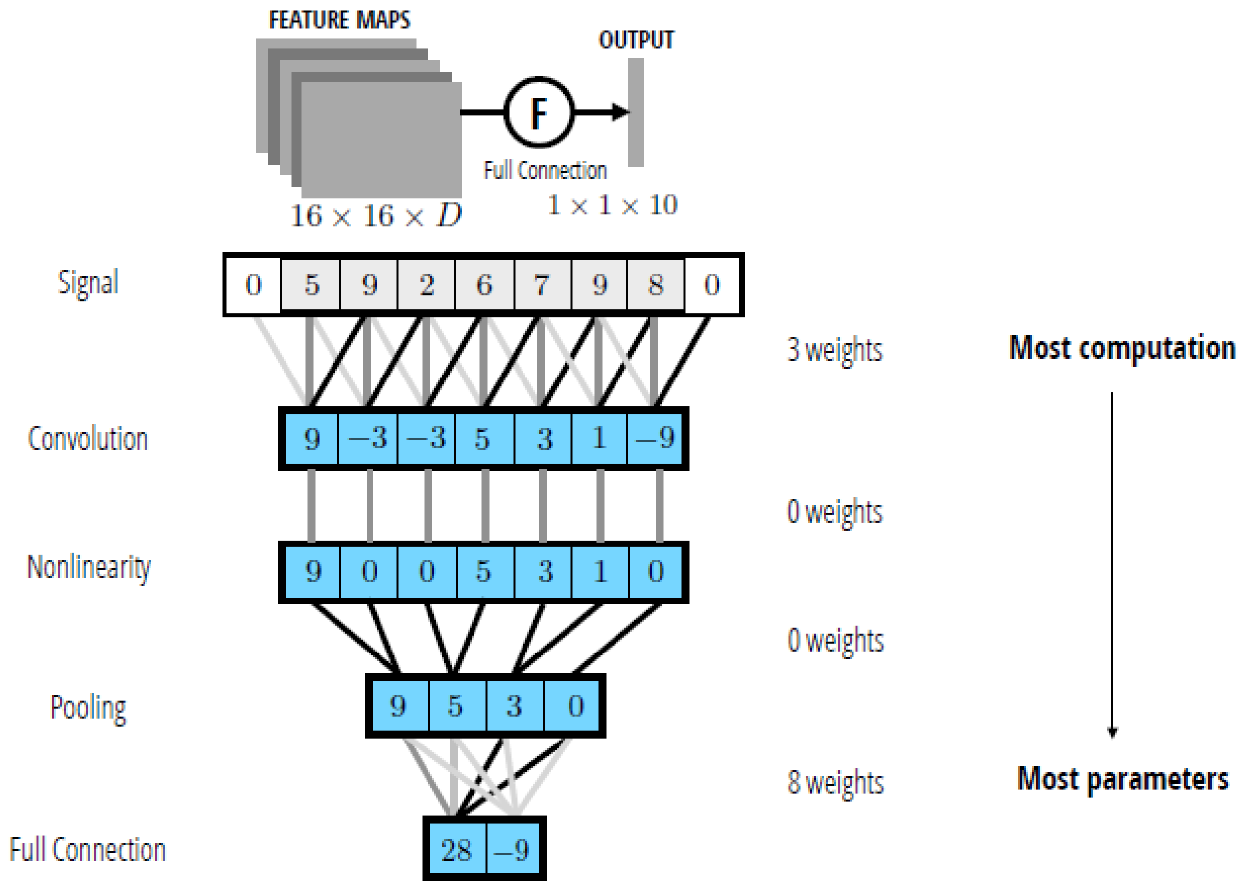
FC Layer

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

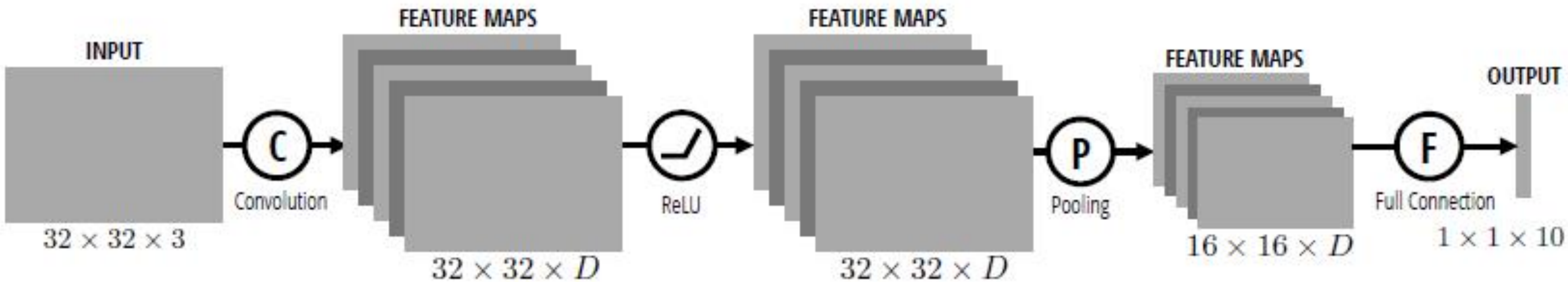


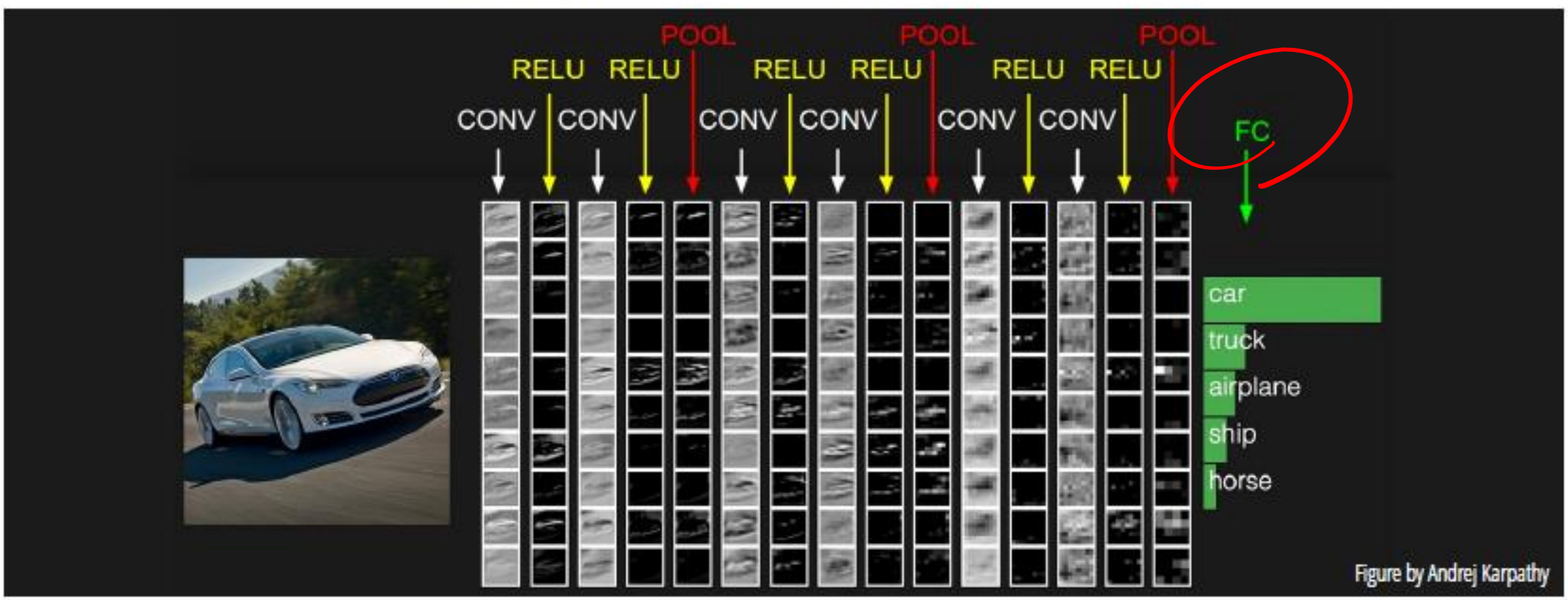
FC Layer

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



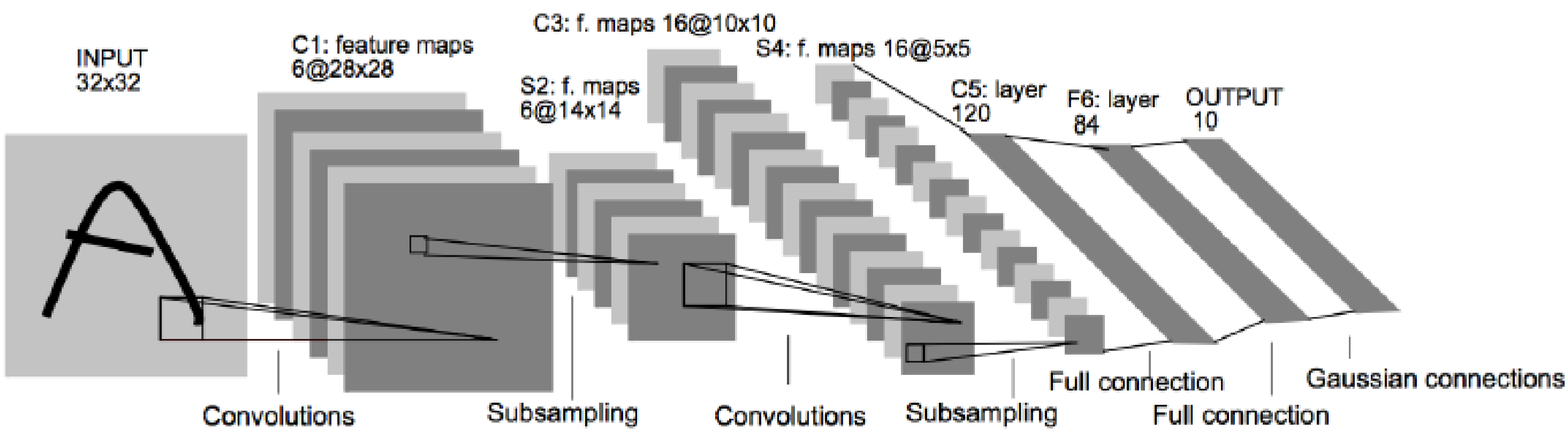
CNN


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
$1 \times 1 \times 10$

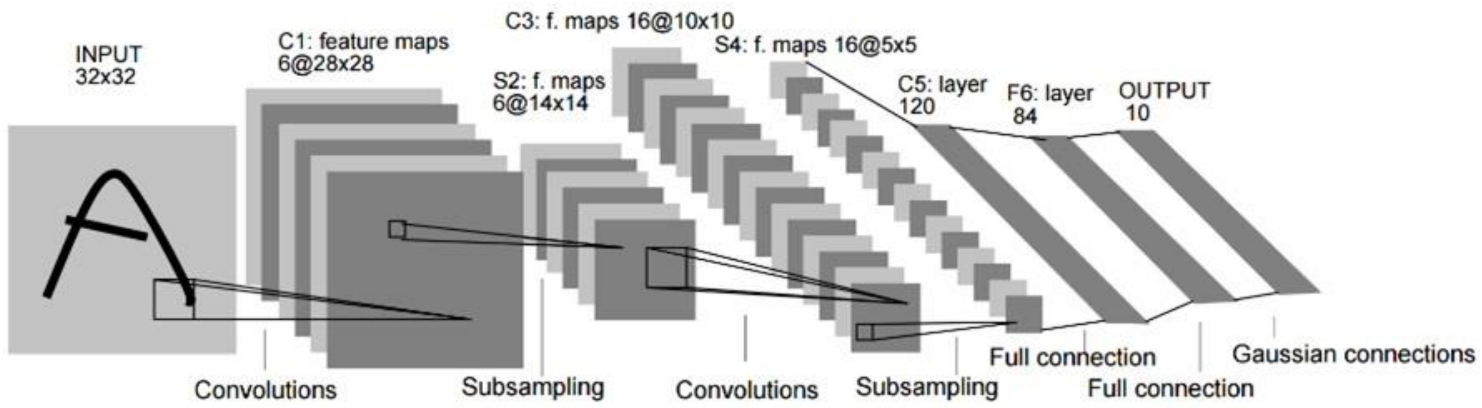

POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse
Figure by Andrej Karpathy

LeNet

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



LeNet [LeCun et al. 1998]



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

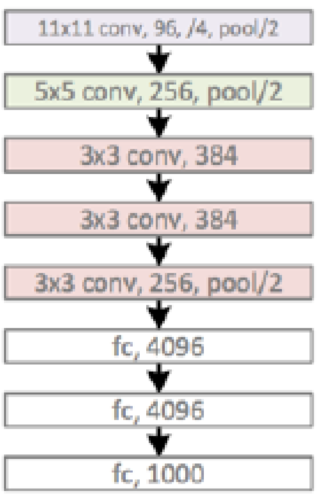
New Driving Forces

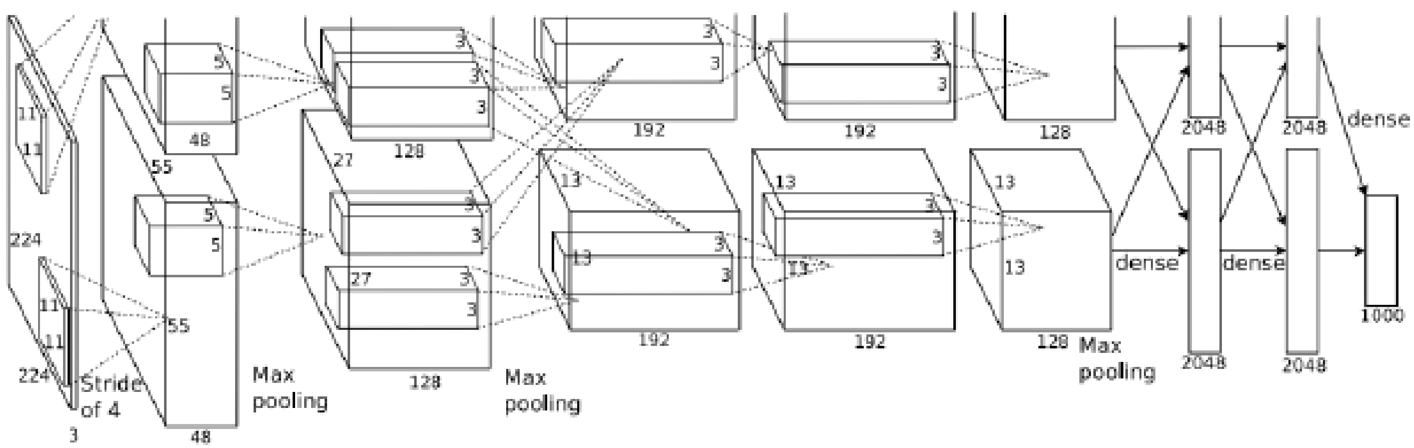
- CPU/GPU computing
  - Personal super computer
- Internet → big data → large datasets become available

AlexNet [Krizhevsky et al., 2012]

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

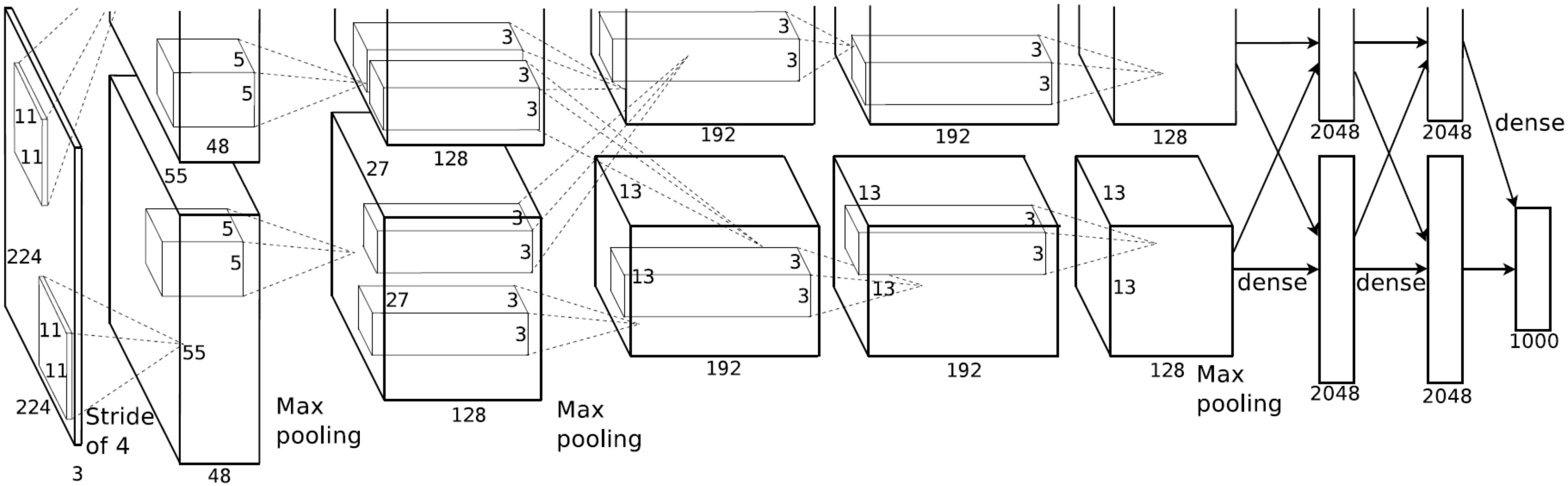
Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] 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)





Krizhevsky *et al.* “ImageNet classification with deep convolutional neural networks,” NIPS, 2012.

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)

Krizhevsky *et al.* “ImageNet classification with deep convolutional neural networks,” NIPS, 2012.

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

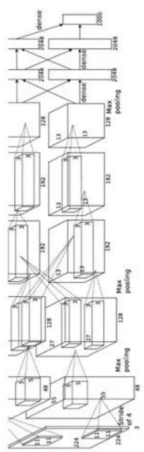
Removing feature & conv layers produces a **33% drop** in performance

Ultra Deep Network

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

8 layers

16.4%



AlexNet (2012)

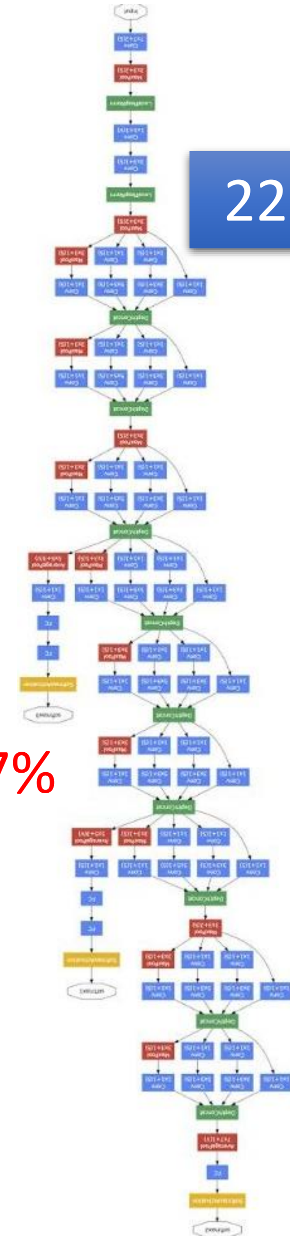
19 layers

7.3%

VGG (2014)

22 layers

6.7%



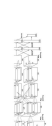
GoogleNet (2014)

Ultra Deep Network

[Prof. H.-Y. Lee]

This ultra deep network have special structure.

16.4%

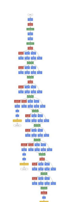


AlexNet (2012)

7.3%

VGG (2014)

6.7%



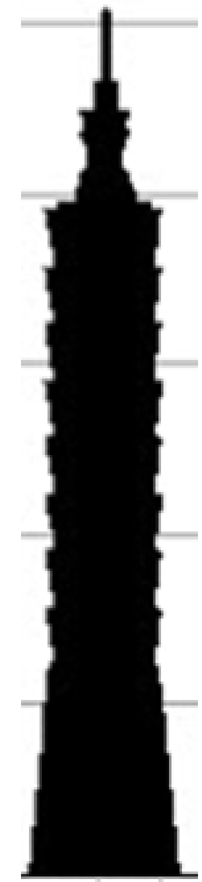
GoogleNet (2014)

152 layers

3.57%

Residual Net (2015)

101 layers



Taipei 101

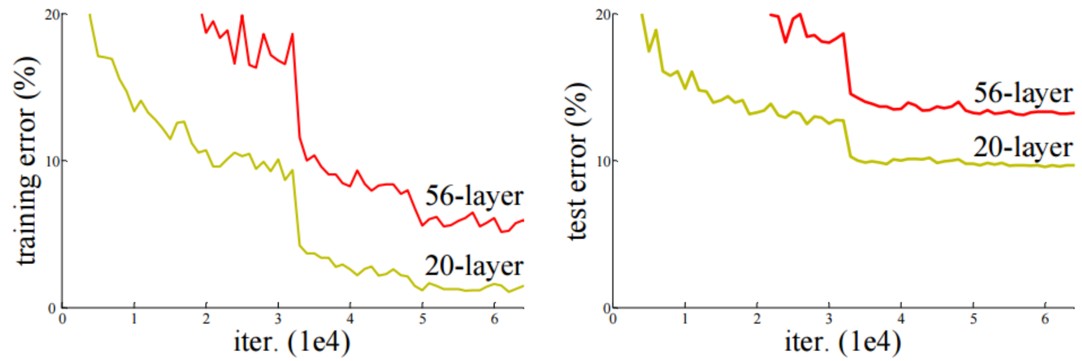
VGG (2014)

- Parameters:
  - Convolution: ~14M, 56MB
  - Fully connected: ~124M, 496MB
- Computation:
  - Convolution: 15.52G Floating MAC
  - Fully connected: 123.63M Floating MAC
  - Full-HD 30fps: 19.3TFLOPS(no overlap)

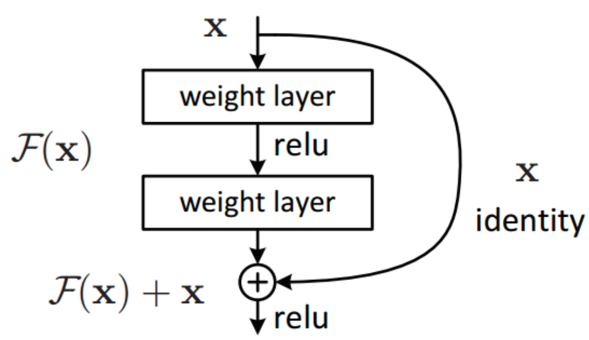
Simonyan and Zisserman, "Very Deep Convolutional Networks for Large-scale Image Recognition," arxiv :1409.1556v6, Sept. 2014

ResNet (2016)

- Can we just increase the #layer?



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

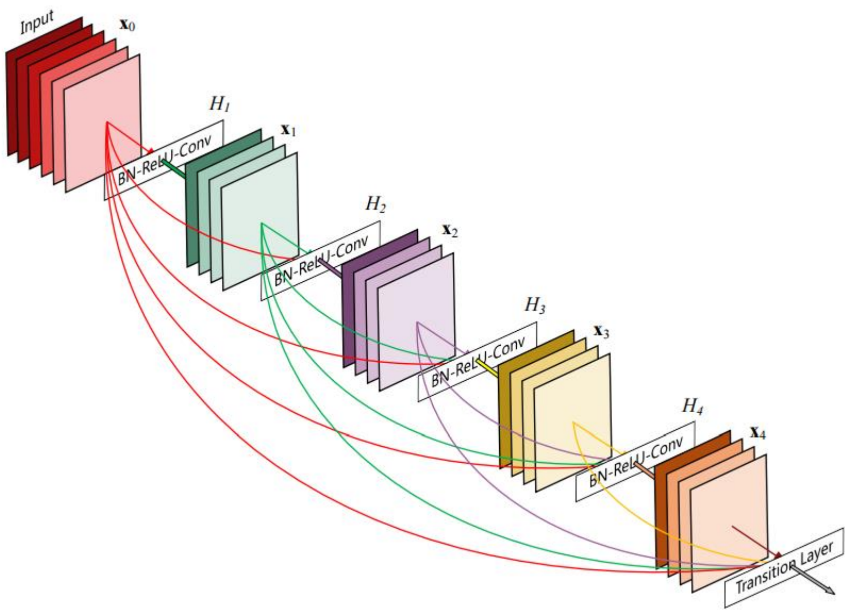


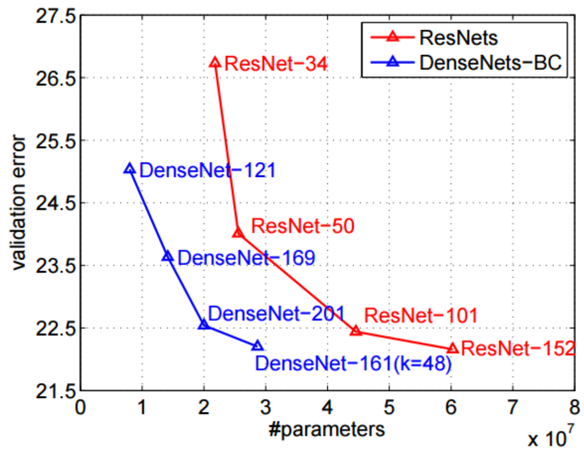
method	top-5 err. (**test**)
VGG [41] (ILSVRC'14)	7.32
GoogLeNet [44] (ILSVRC'14)	6.66
VGG [41] (v5)	6.8
PReLU-net [13]	4.94
BN-inception [16]	4.82
ResNet (ILSVRC'15)	**3.57**

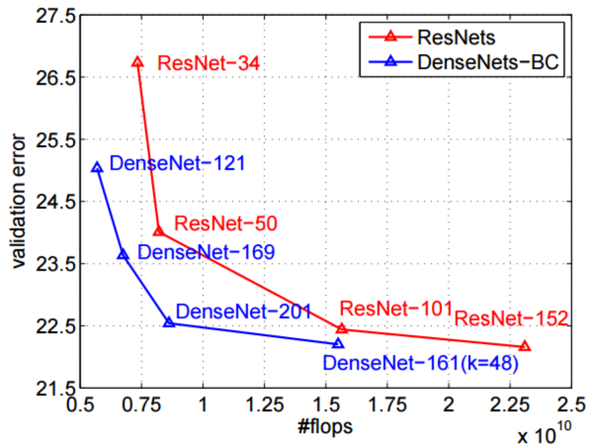
Ref: He, Kaiming, et al. "Deep residual learning for image recognition." *CVPR,* 2016.

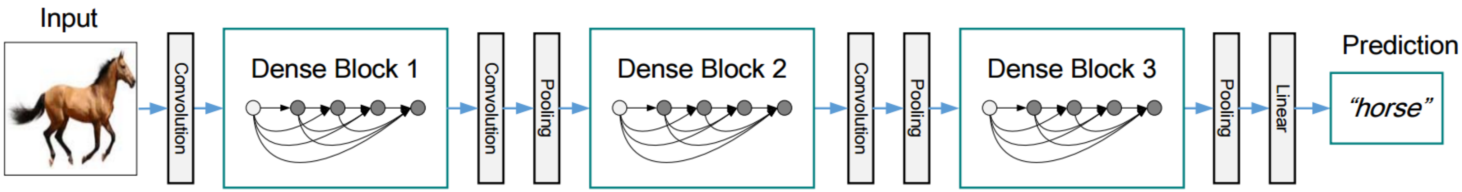
DenseNet (2017)

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



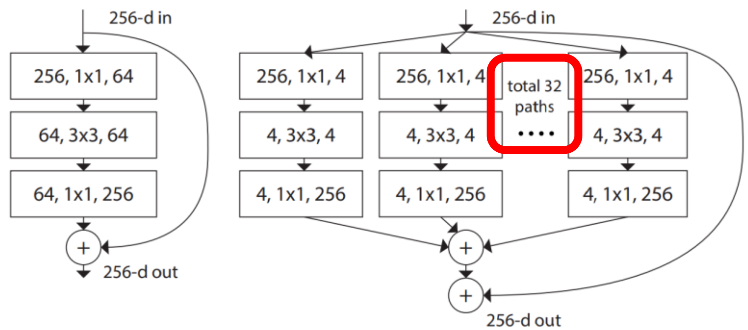






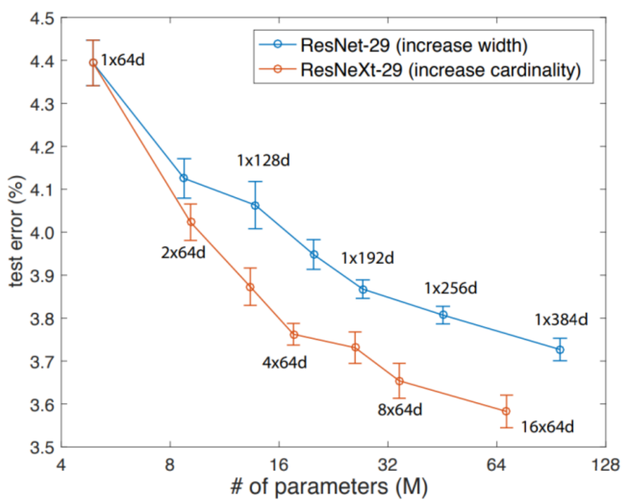
ResNeXT (2017)

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



ResNet block ResNeXt block

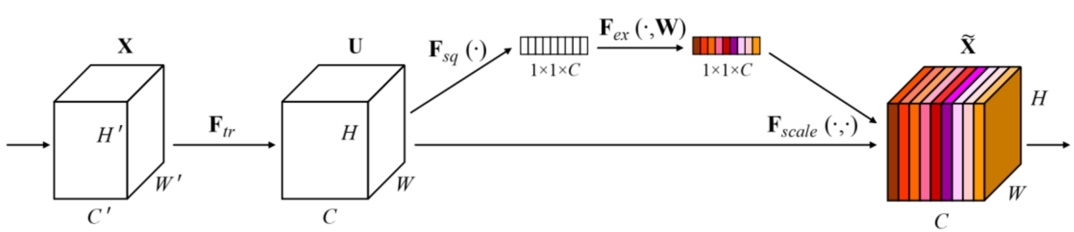
	setting	top-1 error (%)
ResNet-50	1 × 64d	23.9
ResNeXt-50	2 × 40d	23.0
ResNeXt-50	4 × 24d	22.6
ResNeXt-50	8 × 14d	22.3
ResNeXt-50	32 × 4d	**22.2**
ResNet-101	1 × 64d	22.0
ResNeXt-101	2 × 40d	21.7
ResNeXt-101	4 × 24d	21.4
ResNeXt-101	8 × 14d	21.3
ResNeXt-101	32 × 4d	**21.2**

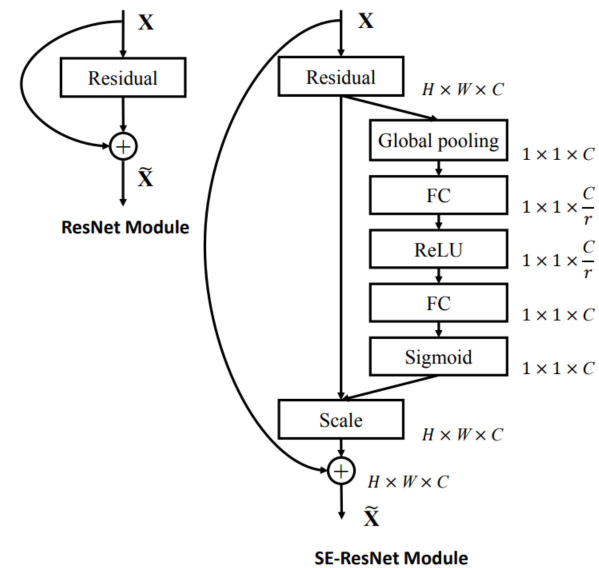


Xie, Saining, et al. "Aggregated residual transformations for deep neural networks." *CVPR,* 2017.

Squeeze-and-Excitation Net (SENet)

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

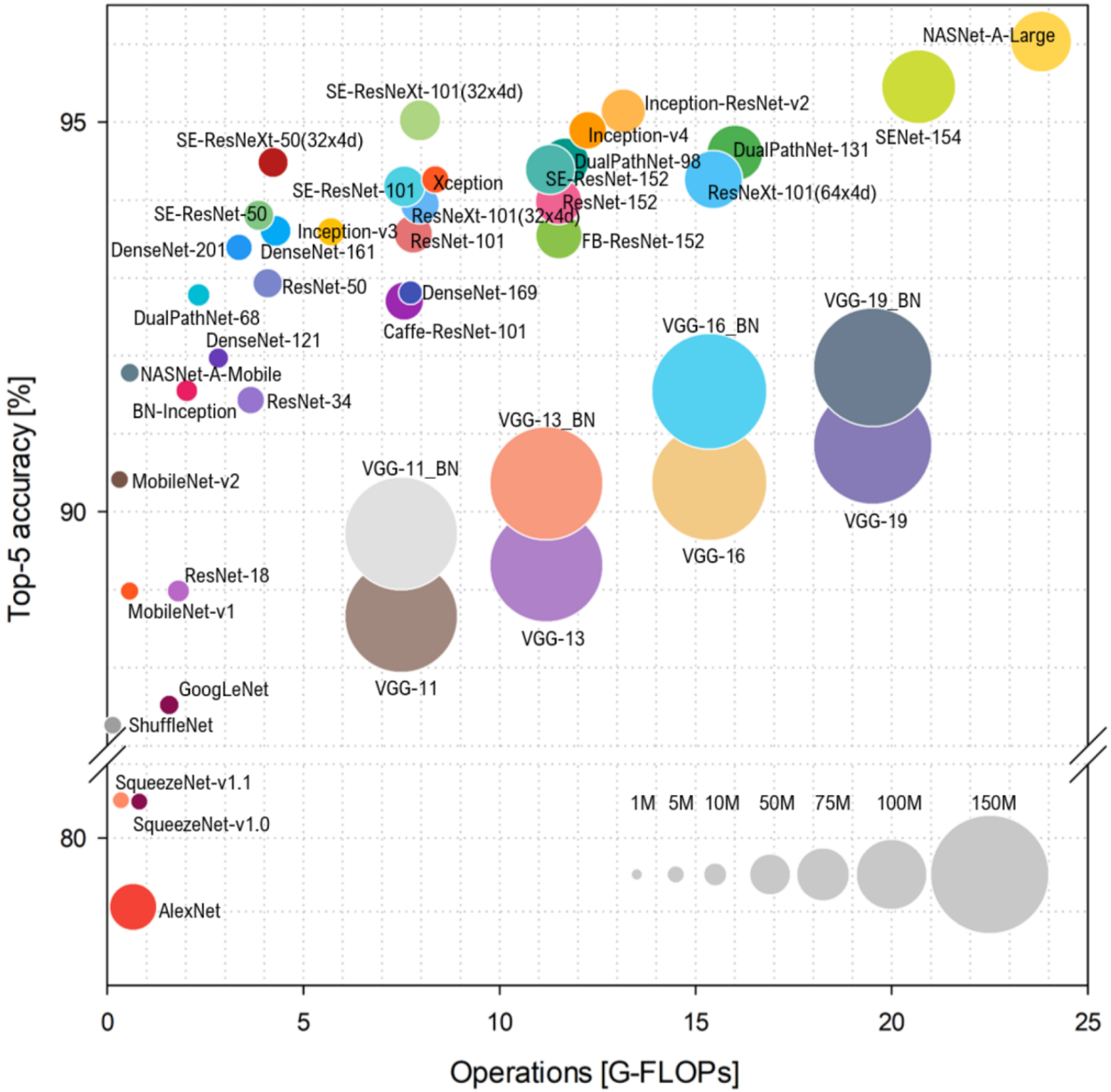




	original		re-implementation			SENet		
	top-1 err.	top-5 err.	top-1 err.	top-5 err.	GFLOPs	top-1 err.	top-5 err.	GFLOPs
ResNet-50 [13]	24.7	7.8	24.80	7.48	3.86	23.29$_{(1.51)}$	6.62$_{(0.86)}$	3.87
ResNet-101 [13]	23.6	7.1	23.17	6.52	7.58	22.38$_{(0.79)}$	6.07$_{(0.45)}$	7.60
ResNet-152 [13]	23.0	6.7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

Hu, Jie, Li Shen, and Gang Sun. "Squeeze-and-excitation networks." *CVPR,* 2018.

Various Deep Learning Models…



Ref: Bianco et al., "Benchmark Analysis of Representative Deep Neural Network Architectures," arXiv:1810.00736.