Computer Vision:
from Recognition to Geometry

Lecture 6:
Convolution Neural Networks for Image Classification

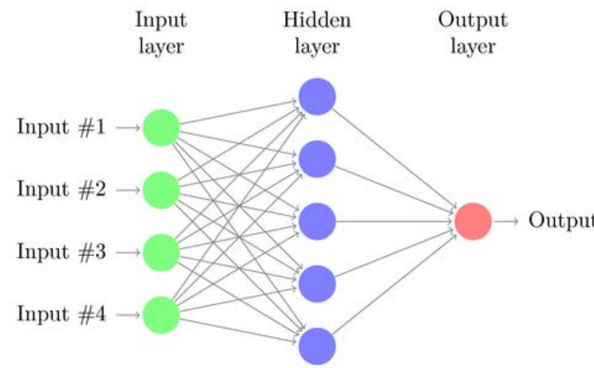
Yu-Chiang Frank Wang 王鈺強

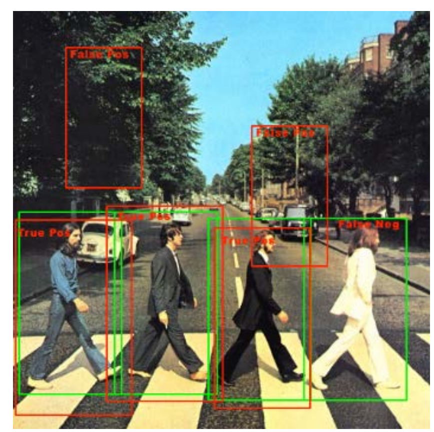
Dept. Electrical Engineering, National Taiwan University

2019/10/16

What’s to Be Covered Today…

- Intro to Neural Networks & CNN
    - Linear Classification
    - Neural Network for Machine Vision
    - Multi-Layer Perceptron
    - Convolutional Neural Networks
- Image Segmentation* (if time permits)
- Object Detection* (if time permits)

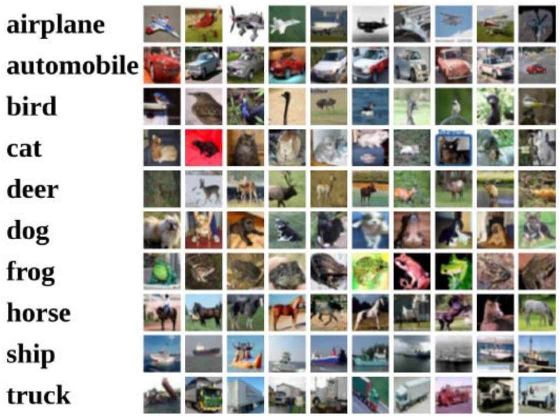


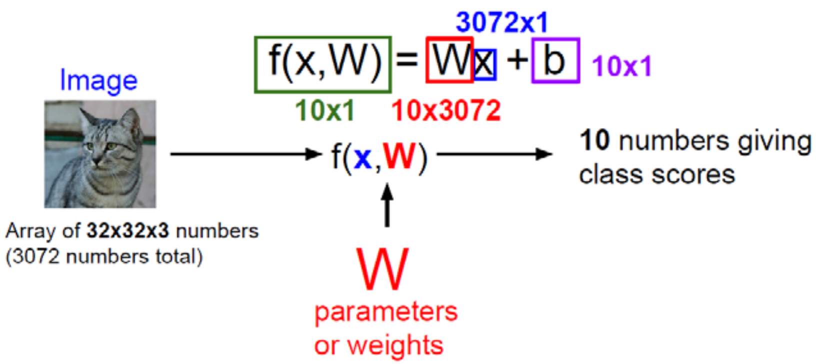


Many slides from Richard Turner, Fei-Fei Li, Yaser Sheikh, Simon Lucey, Kaiming He, and J.-B. Huang

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!)





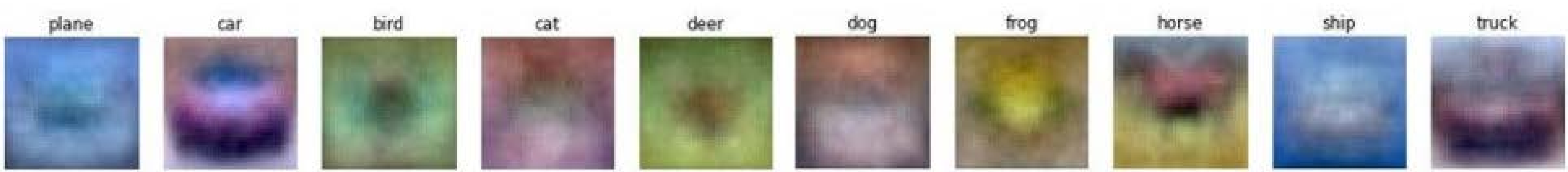


Image credit: Stanford CS231n

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.

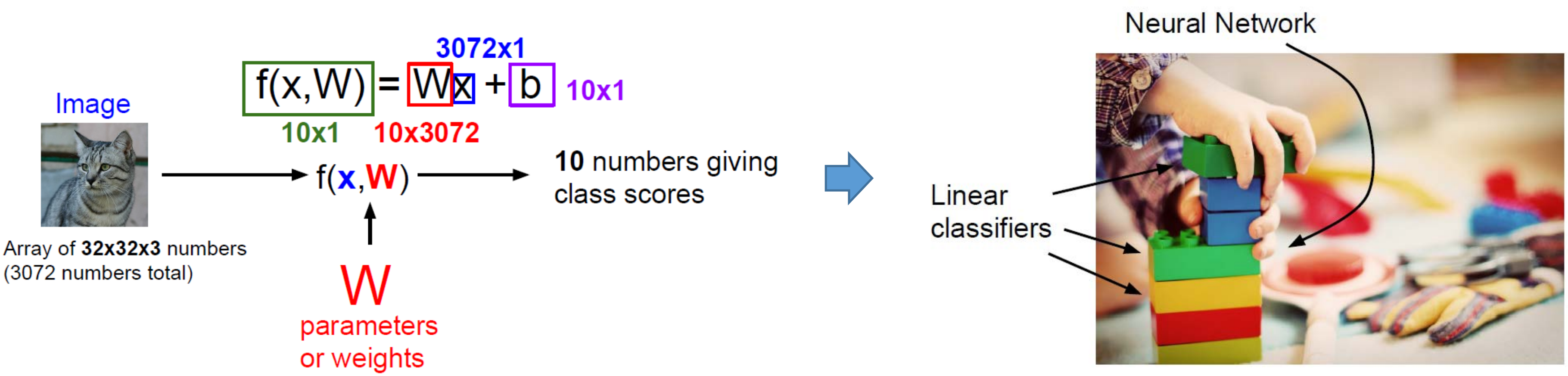
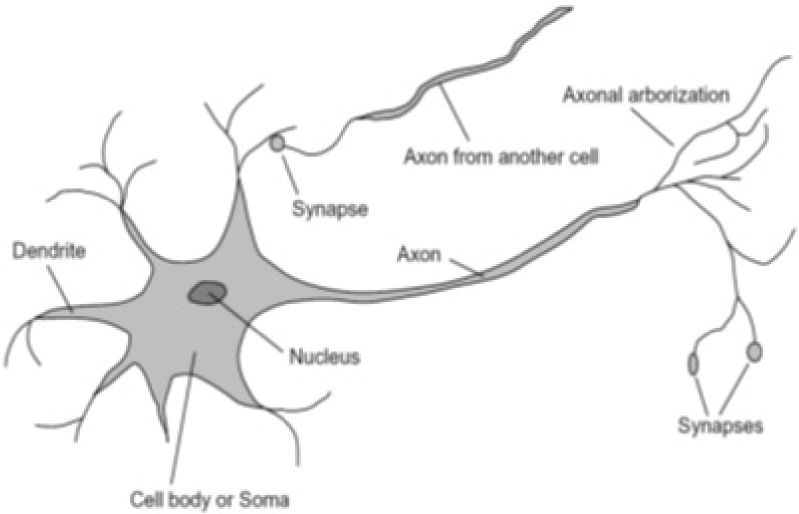
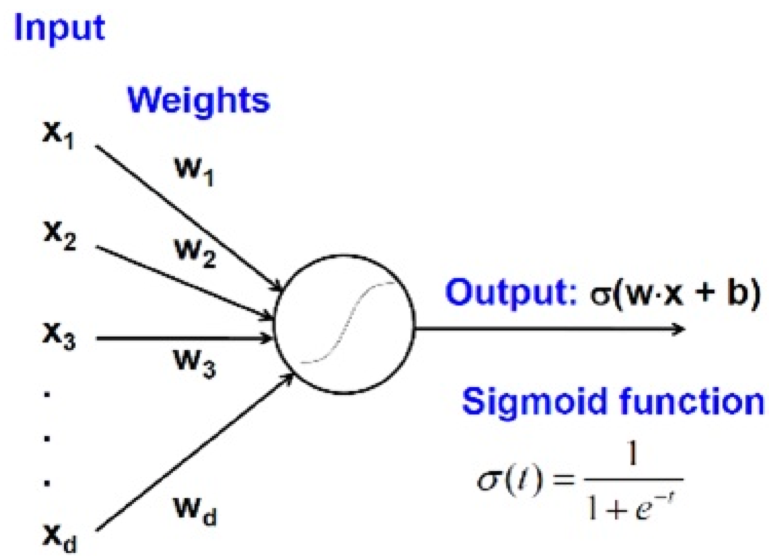


Image credit: Stanford CS231n

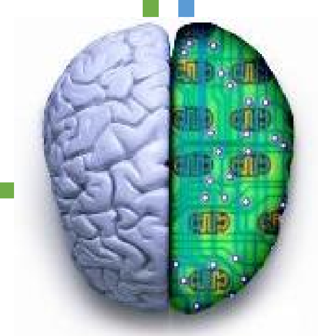
Biological neuron and Perceptrons



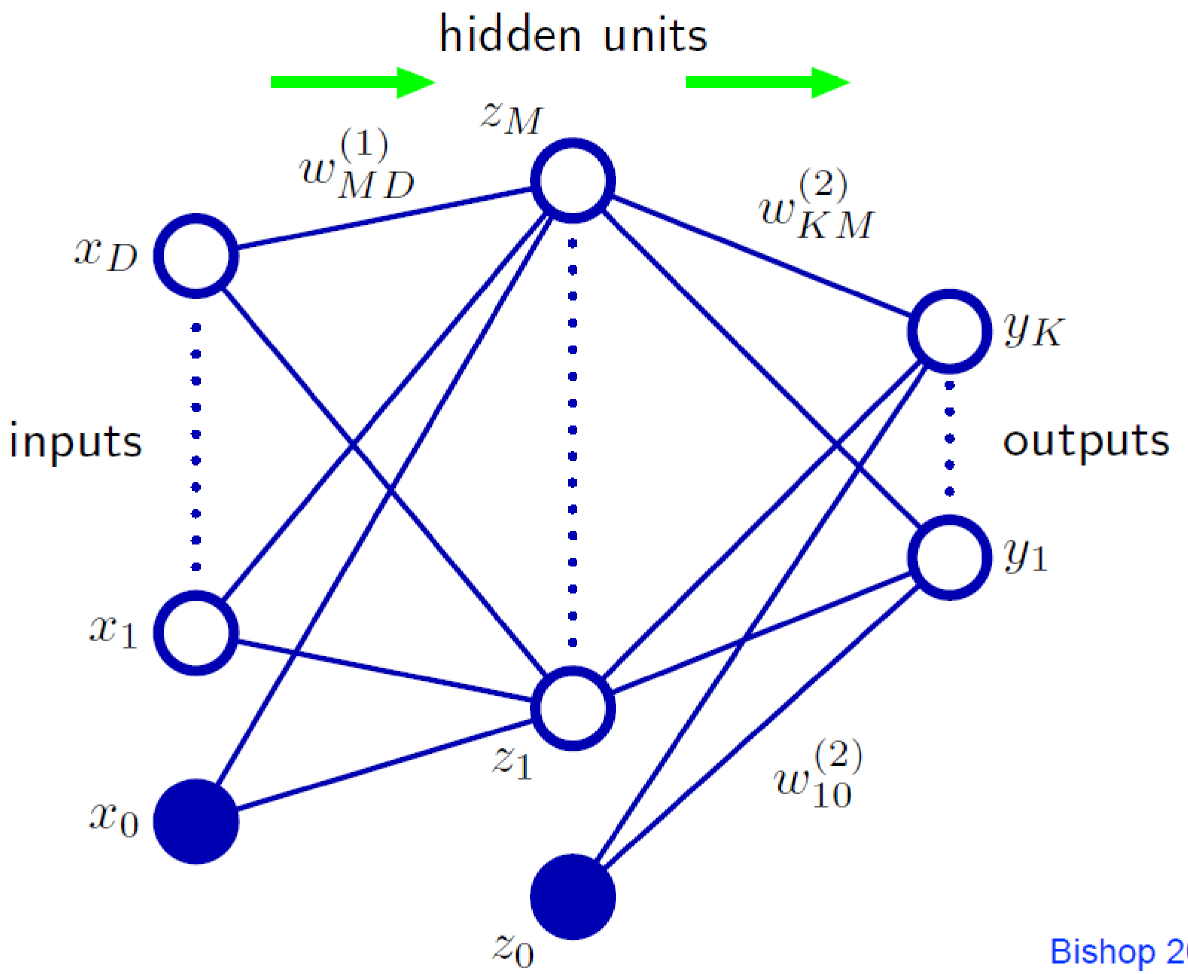
A biological neuron



An artificial neuron (Perceptron)
- a linear classifier



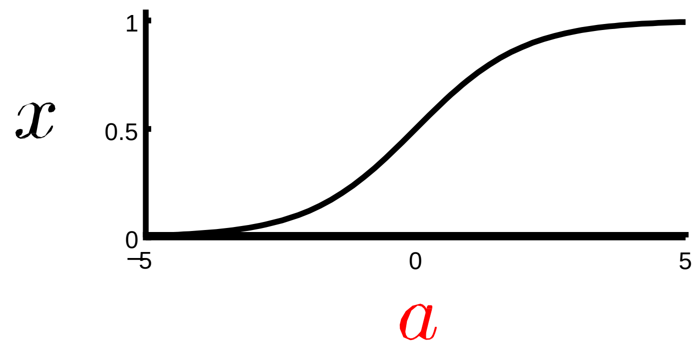
Multi-Layer Perceptron: A Nonlinear Classifier (cont'd)

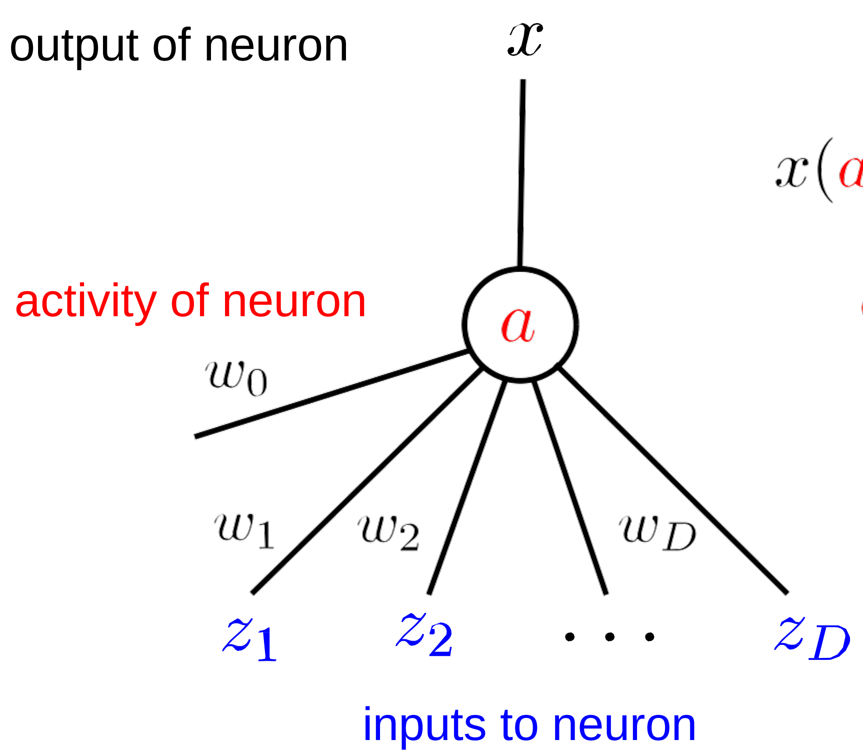


Bishop 2006

Let's Get a Closer Look...

- A single neuron

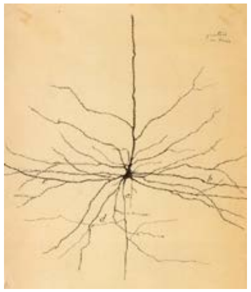




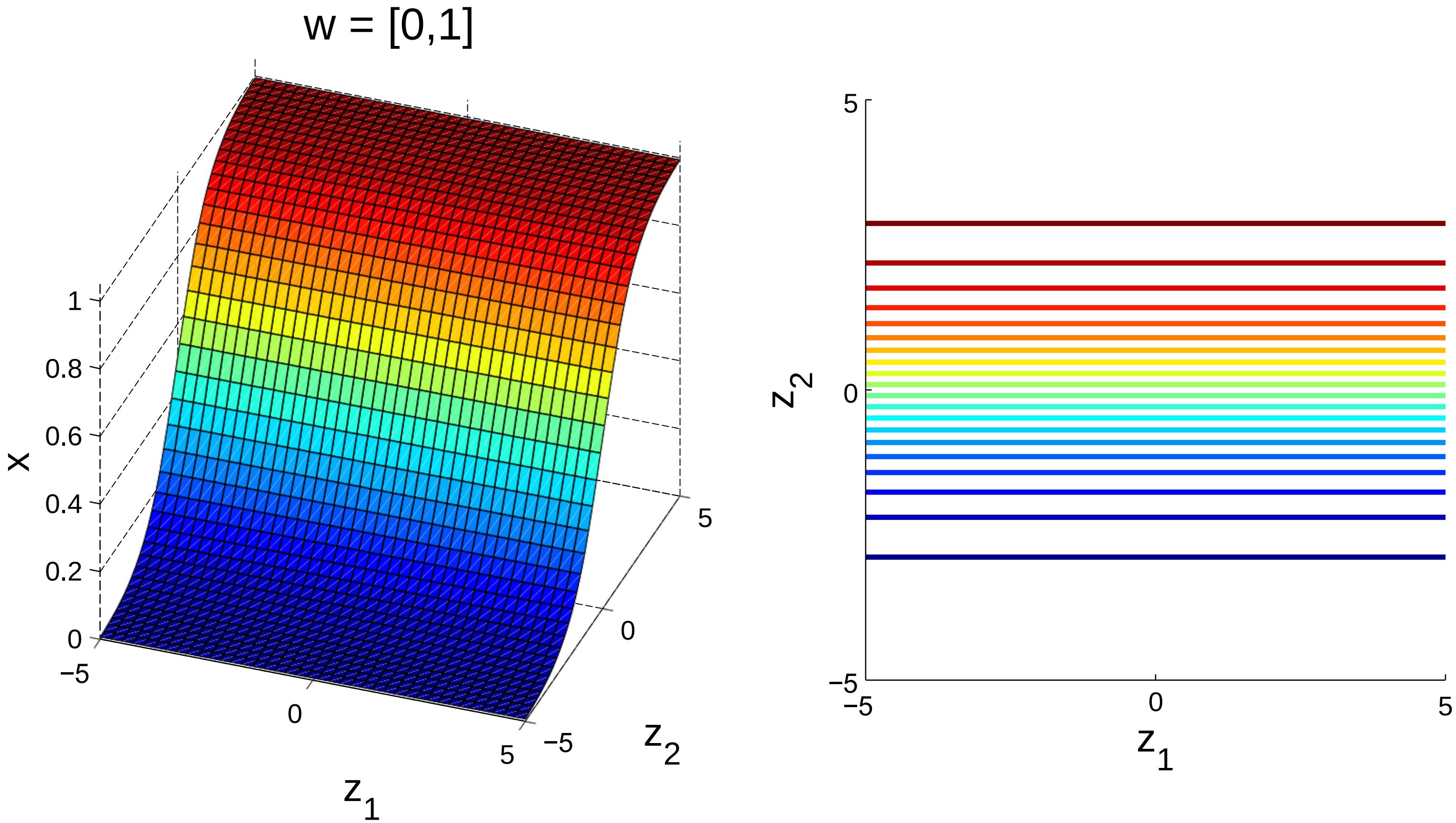
$$x(a) = \frac{1}{1+\exp(-a)} \qquad x \in (0,1)$$

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

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

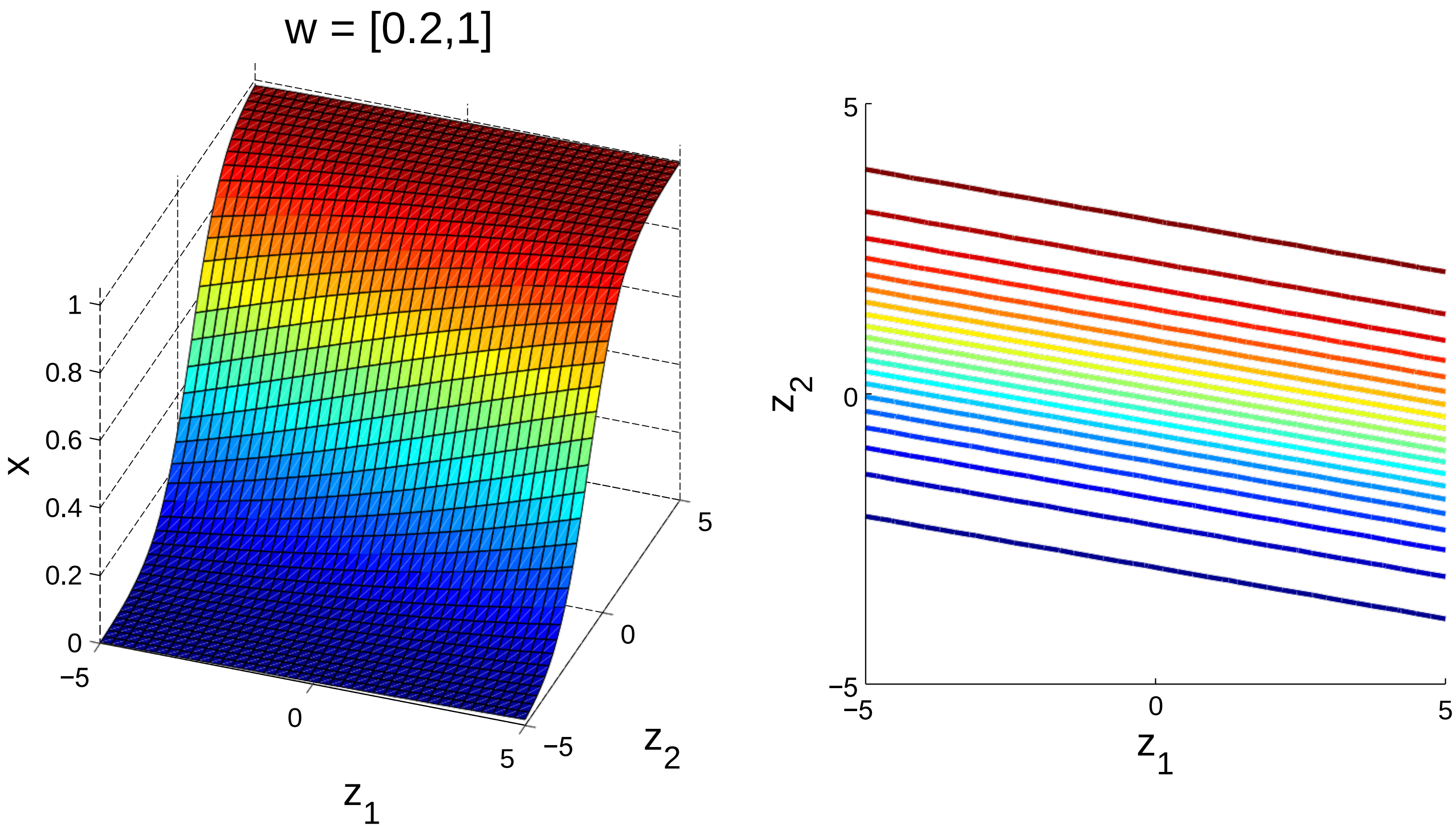


Input-Output Function of a Single Neuron



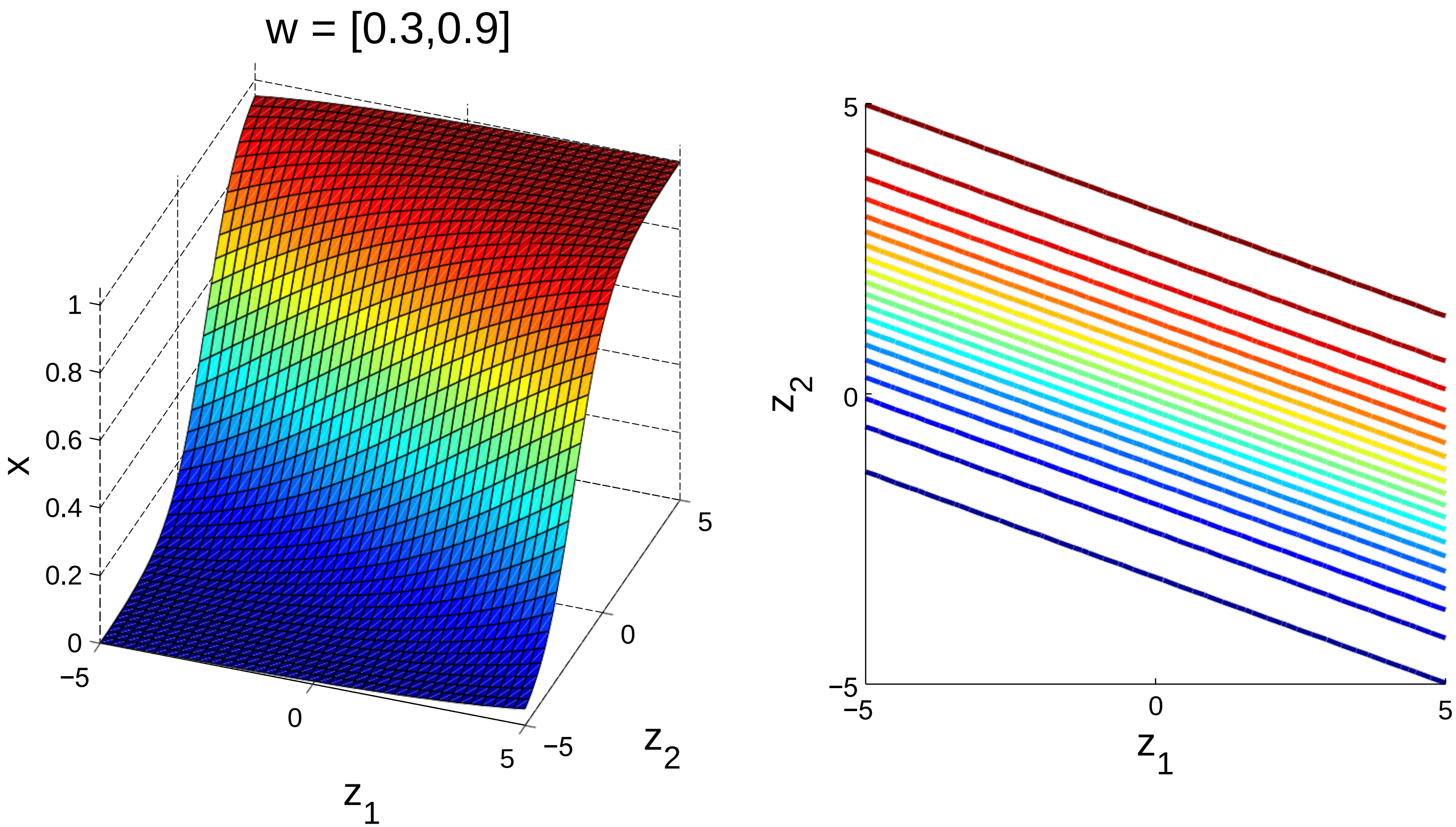
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

Input-Output Function of a Single Neuron



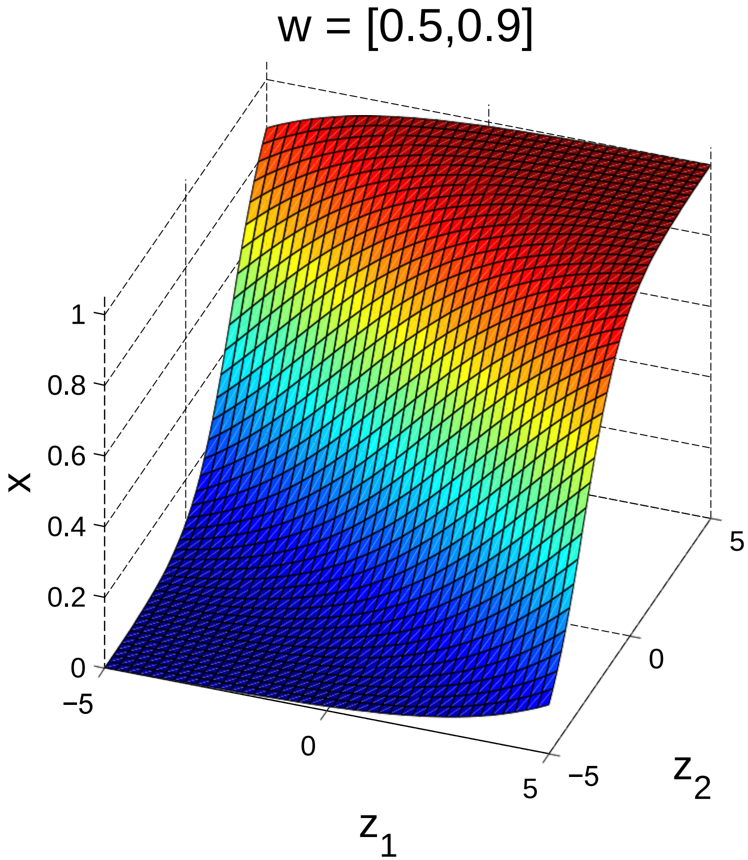
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

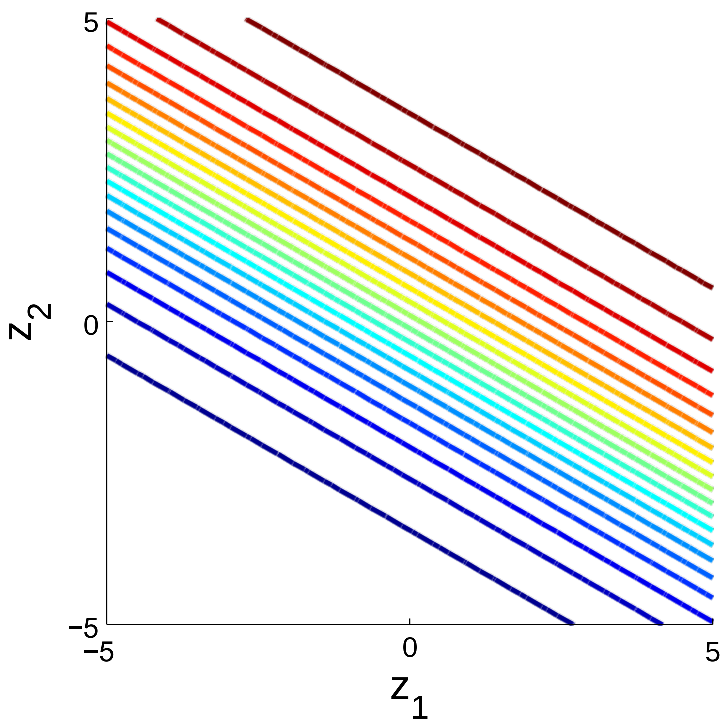
Input-Output Function of a Single Neuron



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

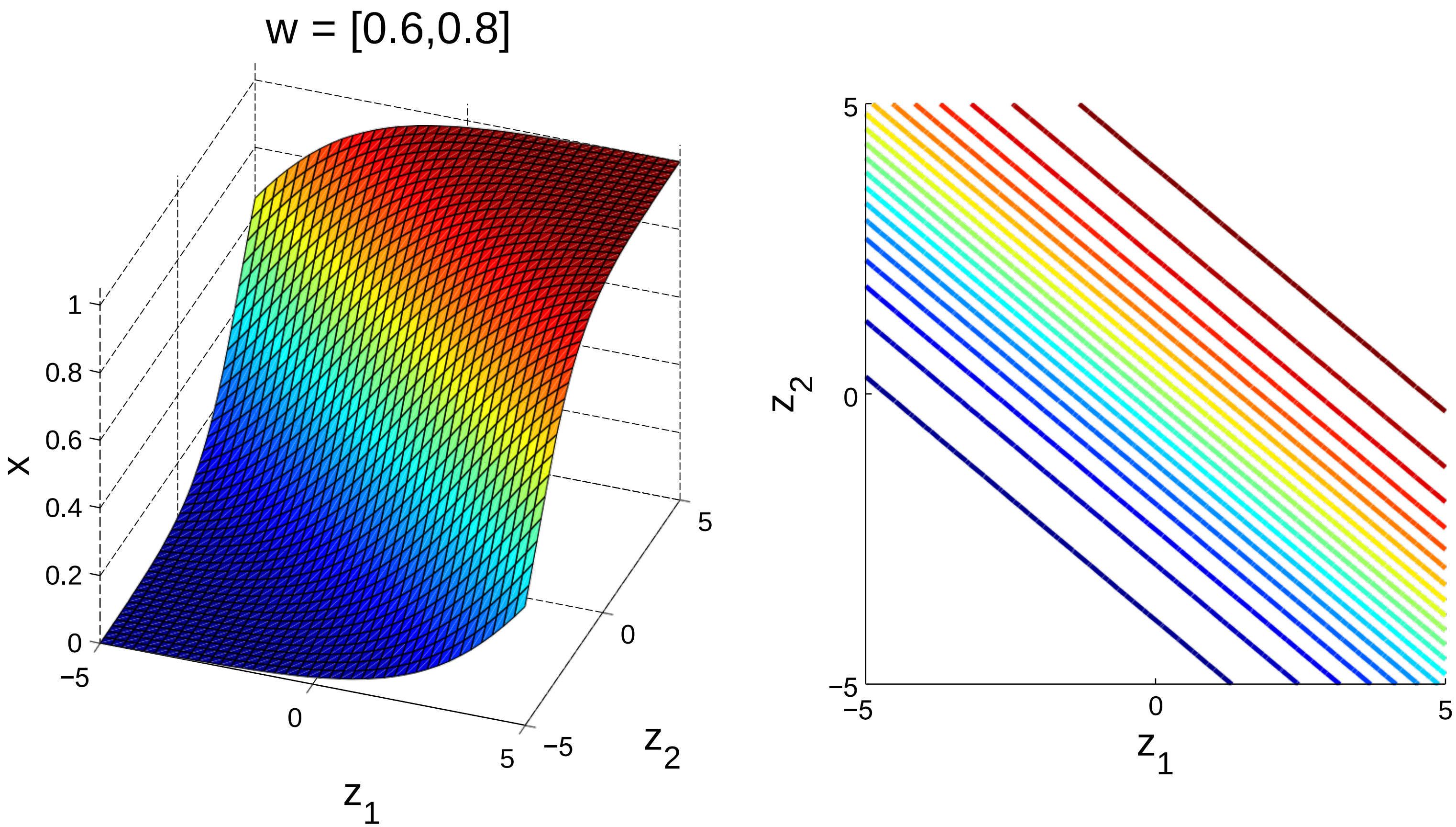
Input-Output Function of a Single Neuron





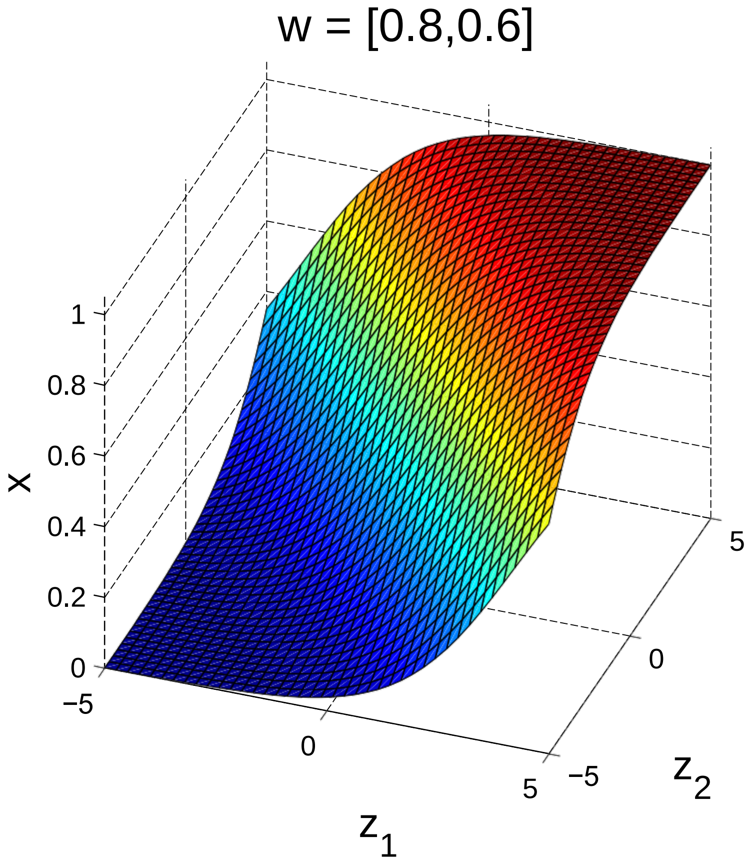
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

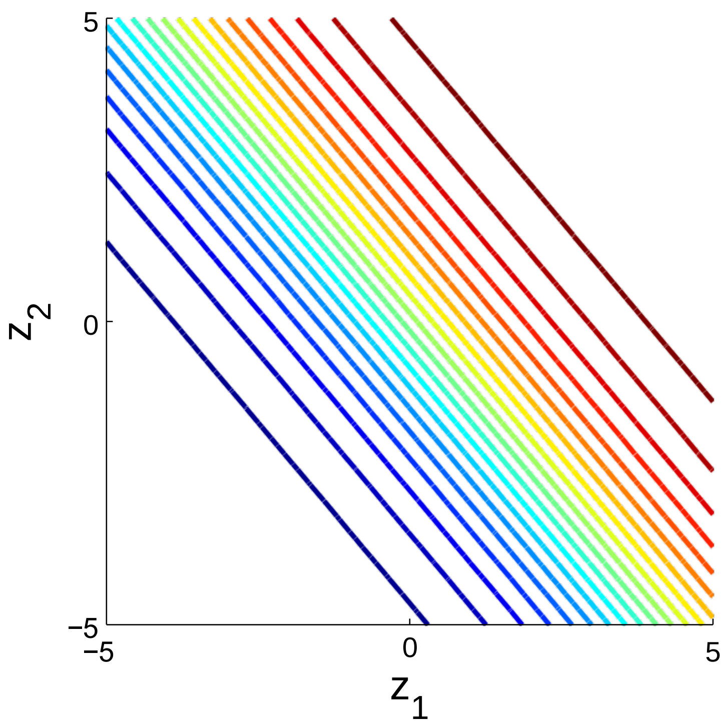
Input-Output Function of a Single Neuron



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

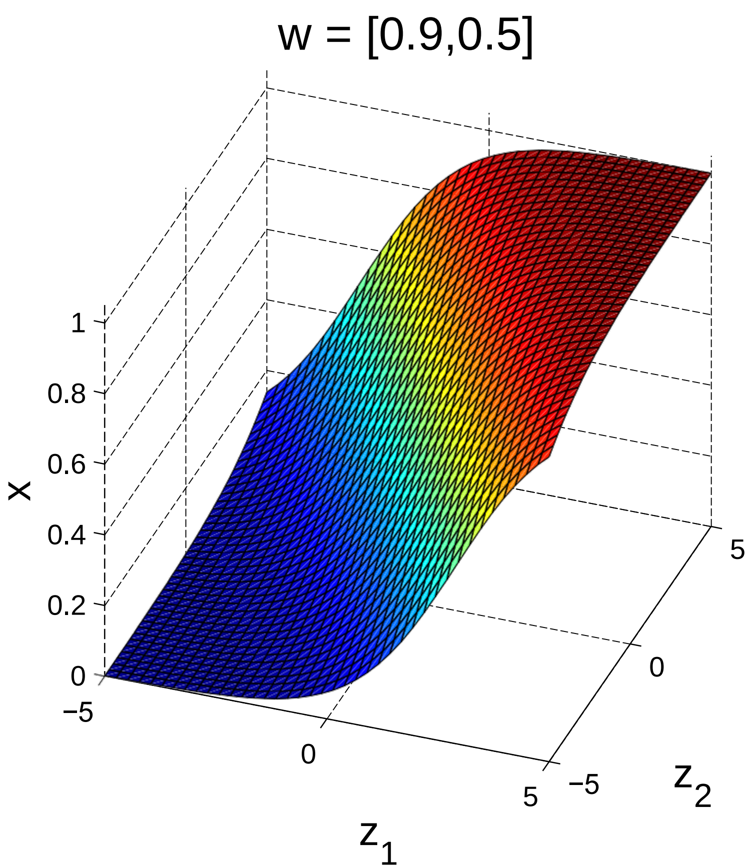
Input-Output Function of a Single Neuron

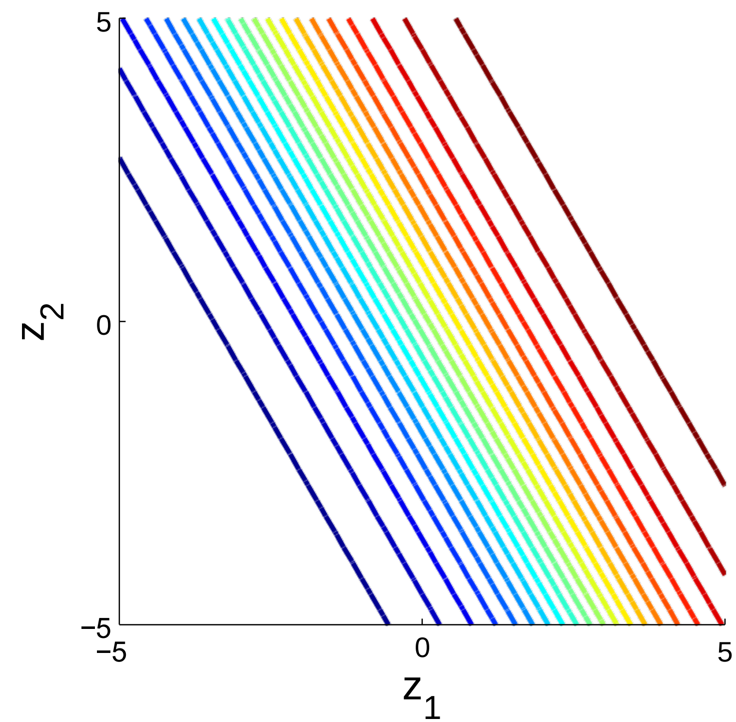




$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

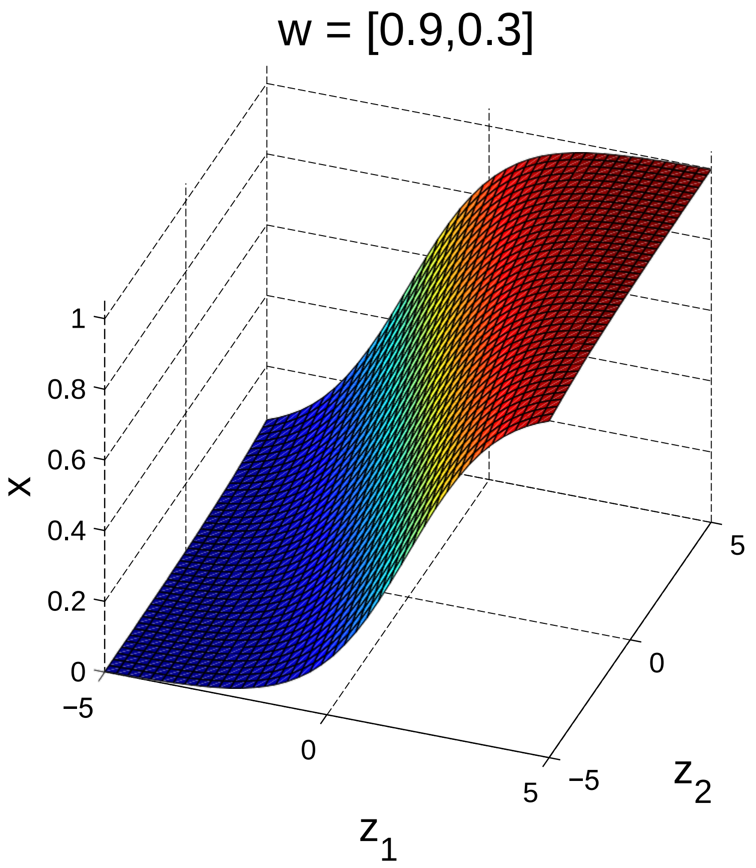
Input-Output Function of a Single Neuron

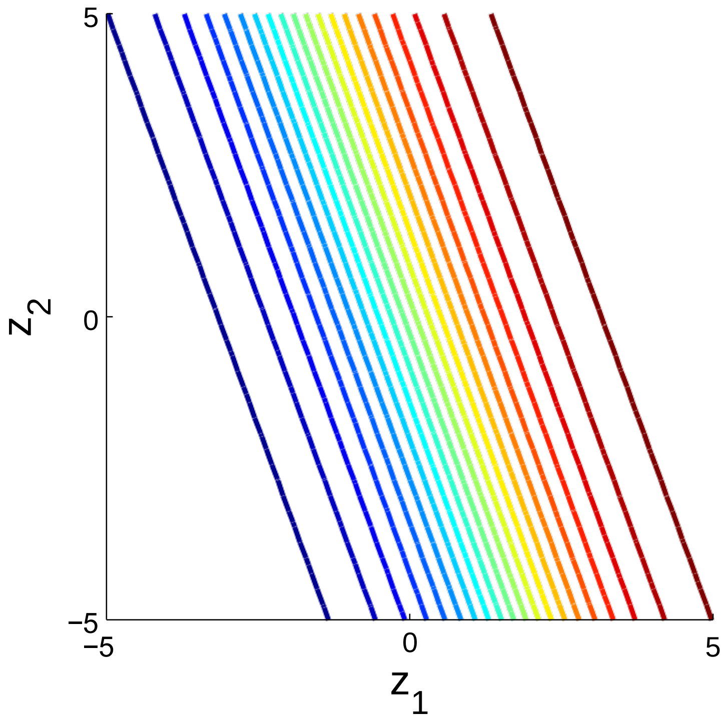




$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

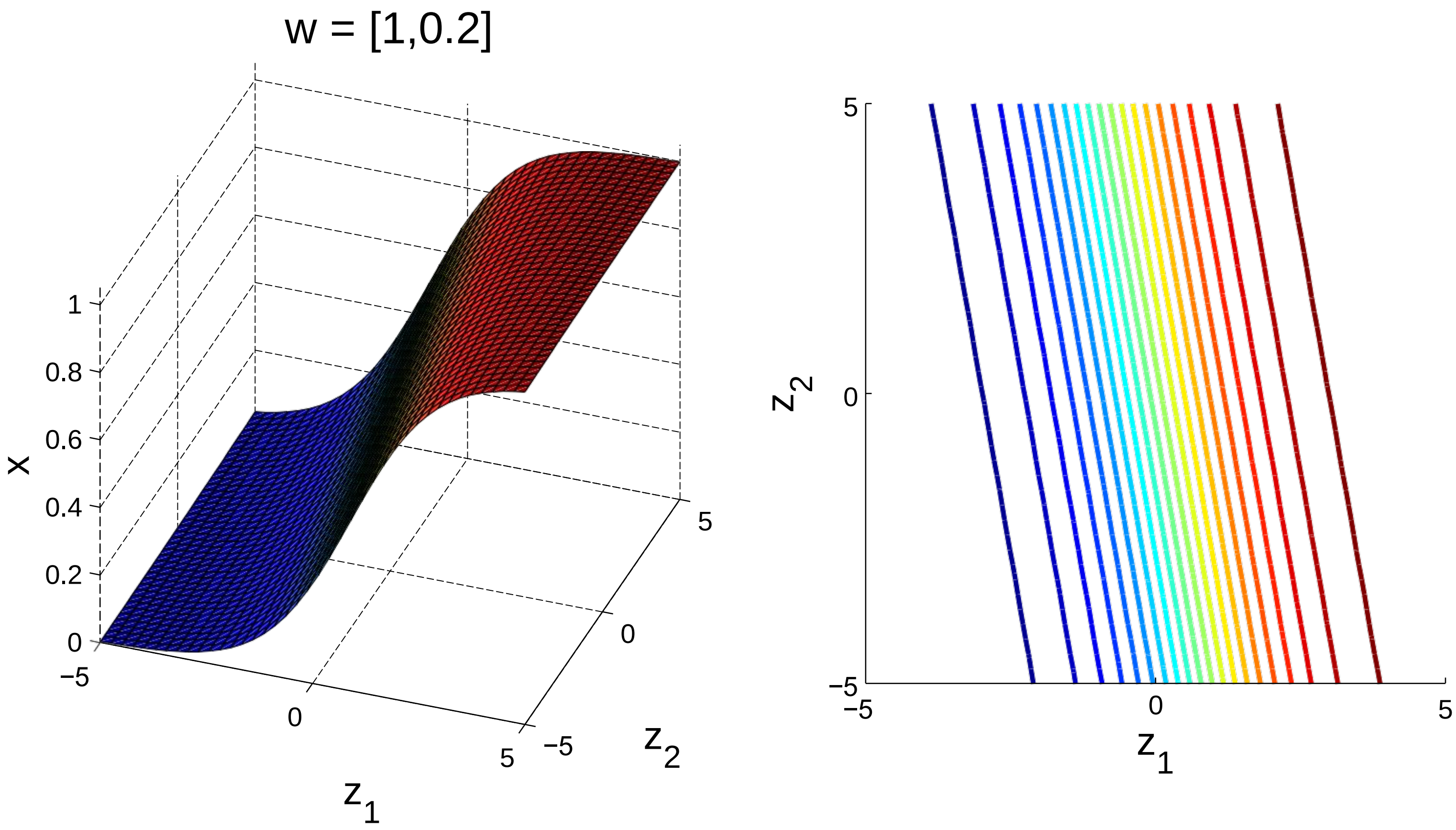
Input-Output Function of a Single Neuron





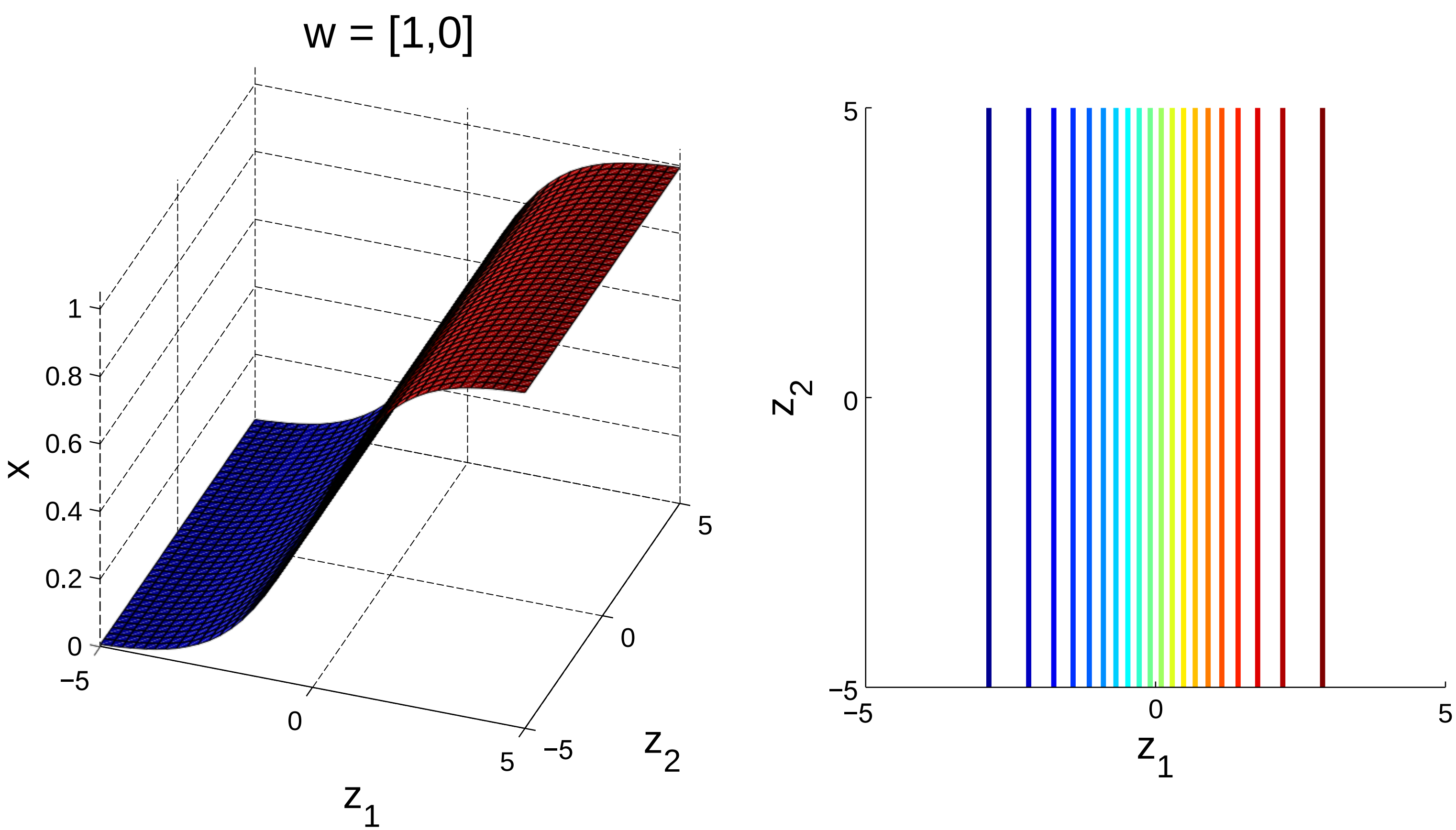
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

Input-Output Function of a Single Neuron



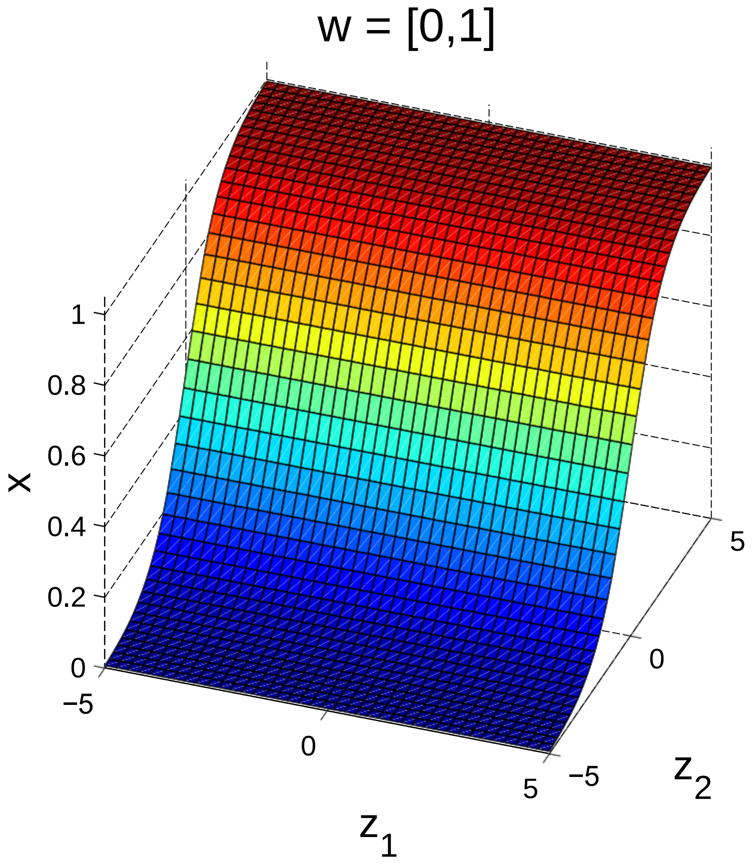
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

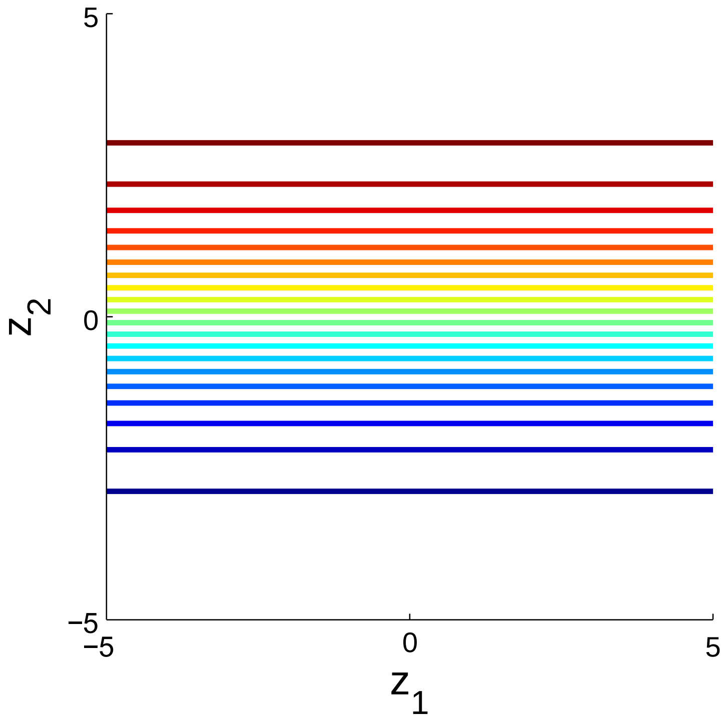
Input-Output Function of a Single Neuron



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

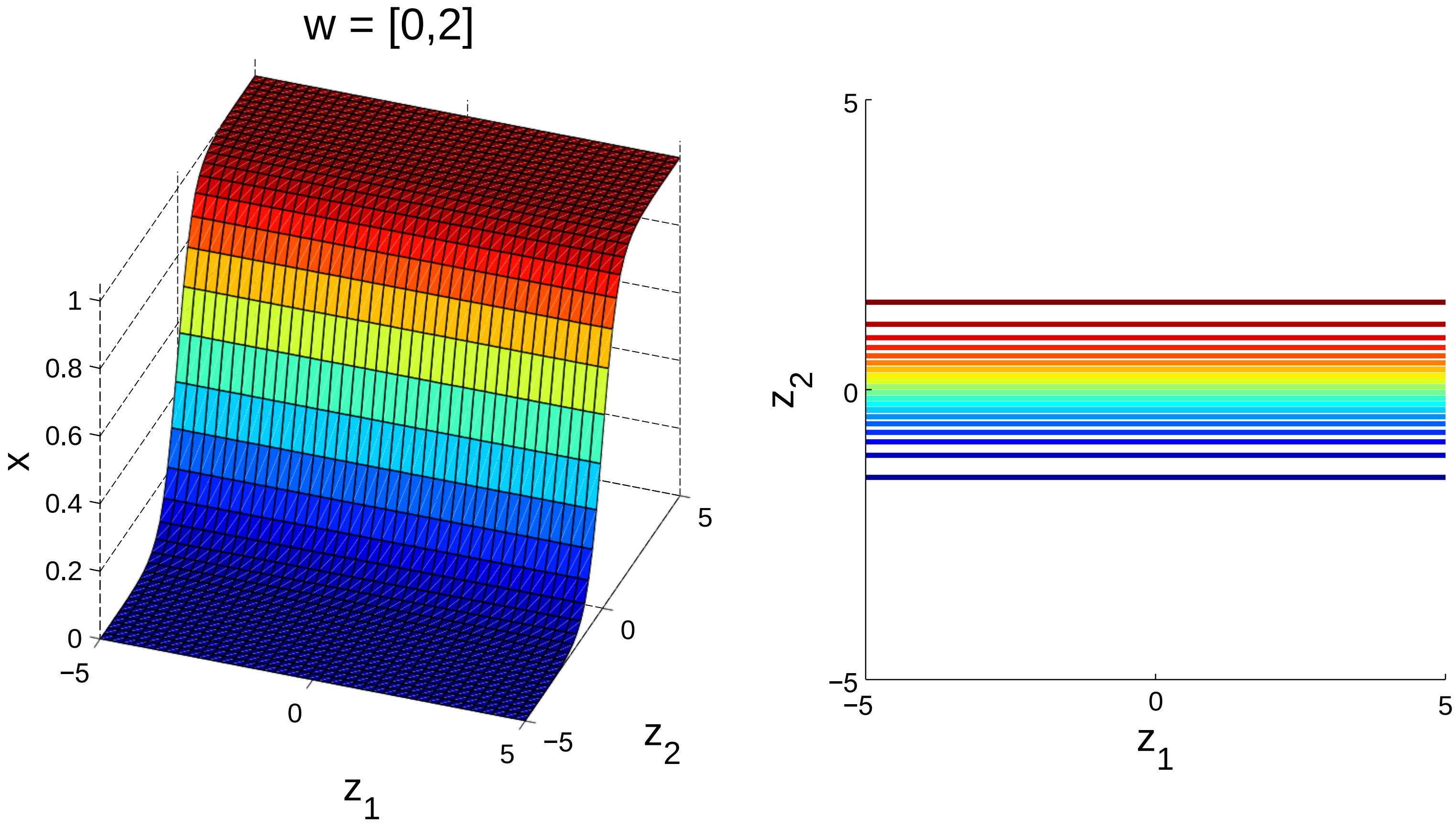
Input-Output Function of a Single Neuron (cont'd)





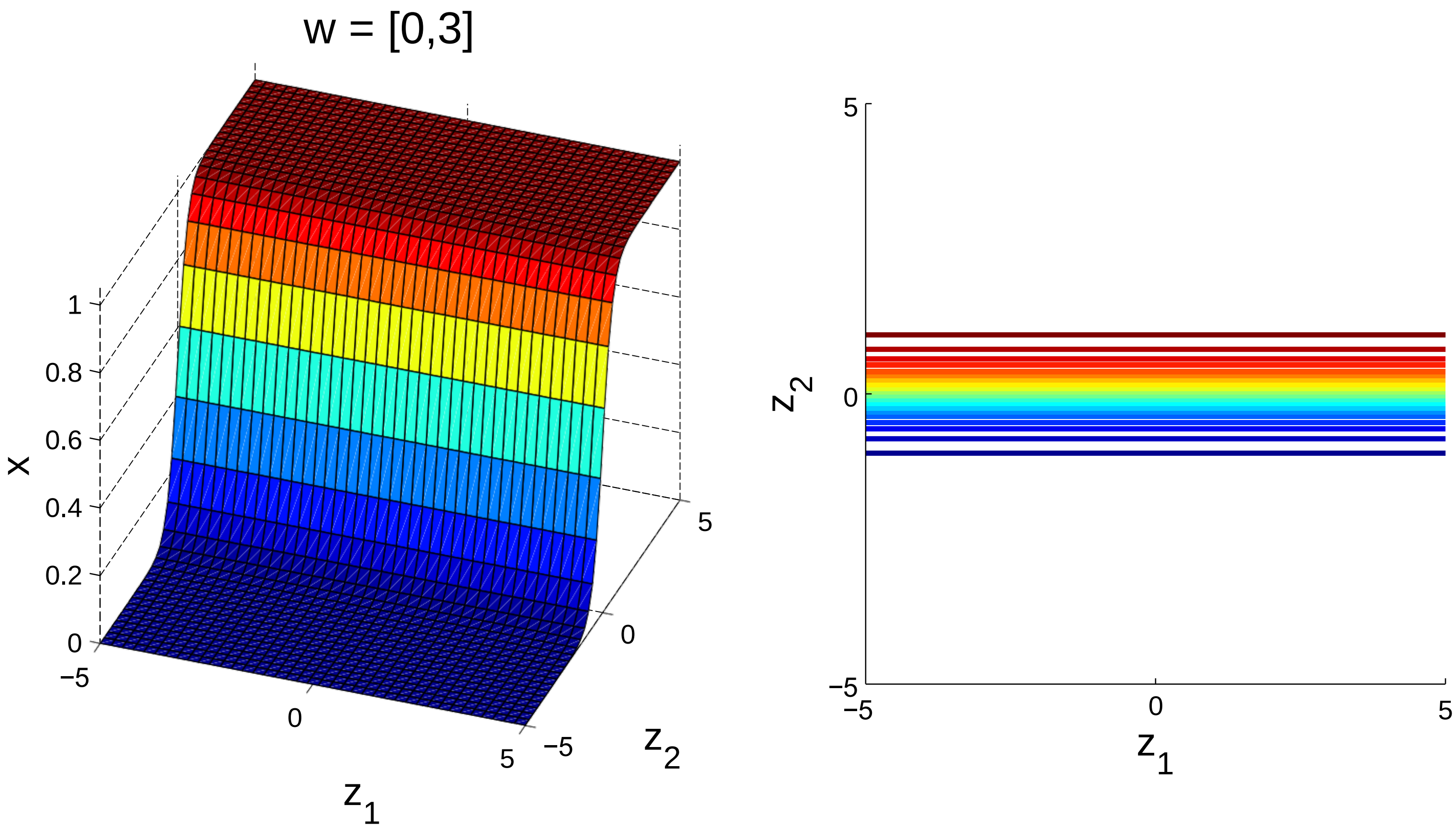
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

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



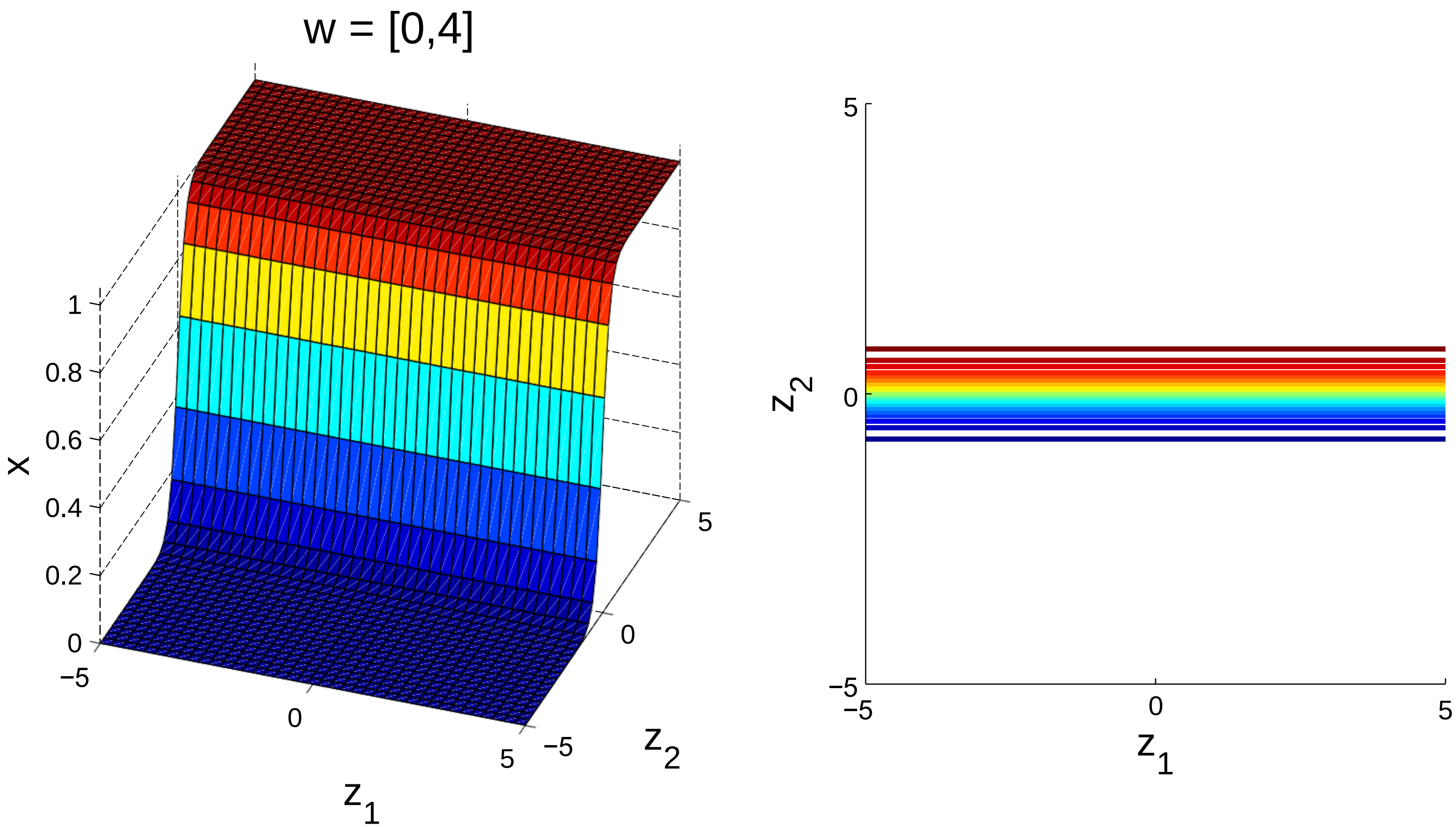
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

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



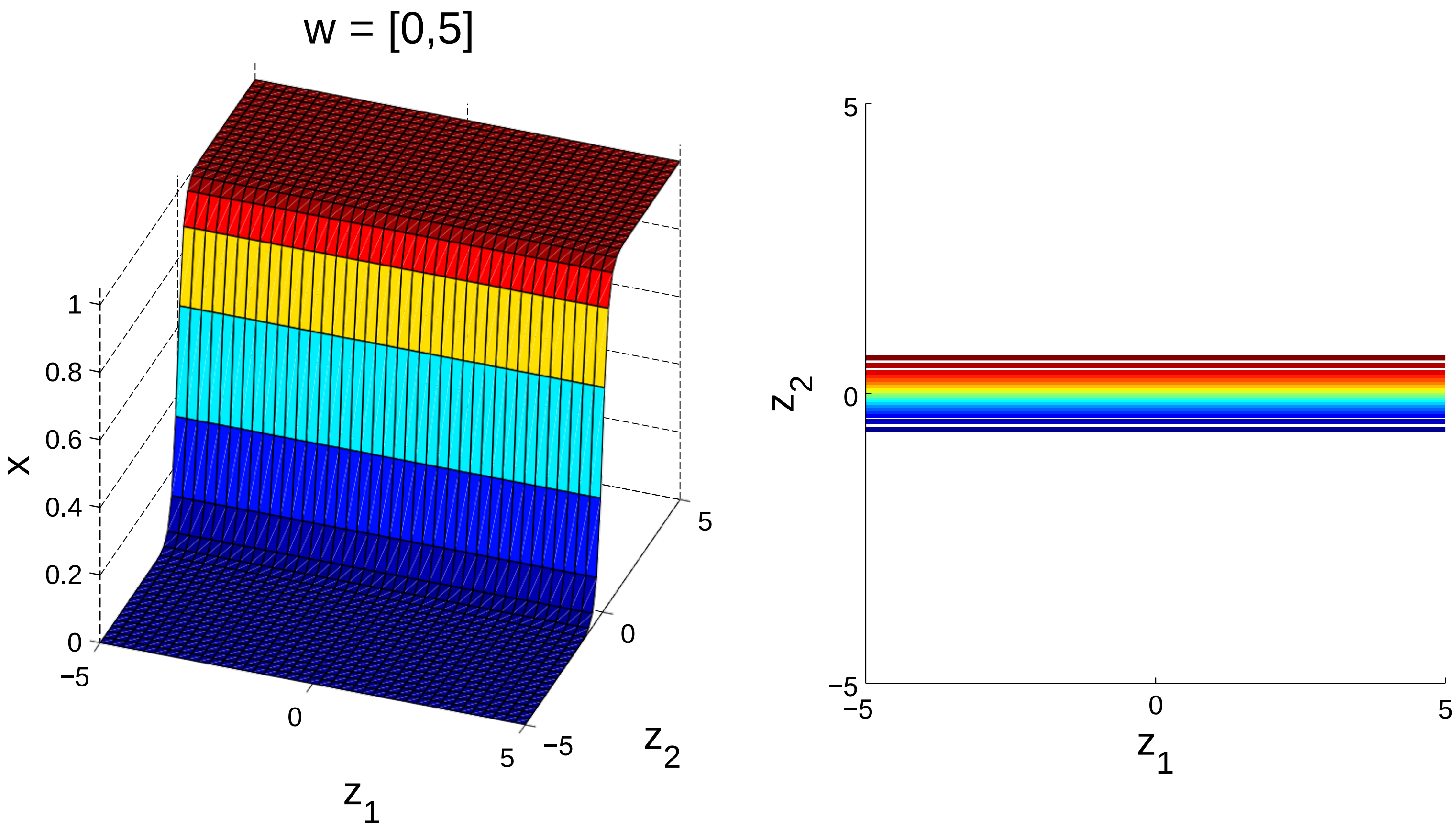
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

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



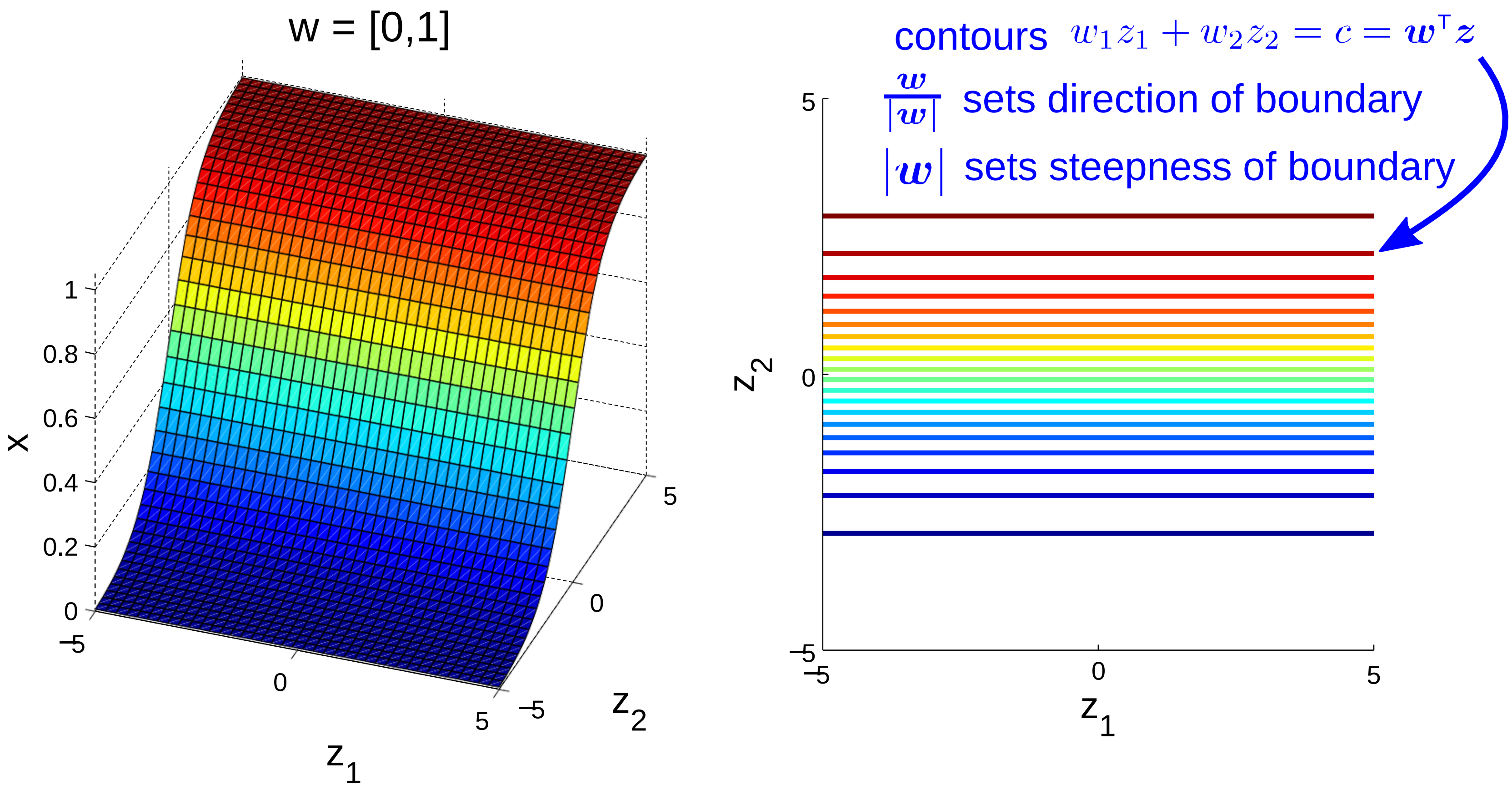
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

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



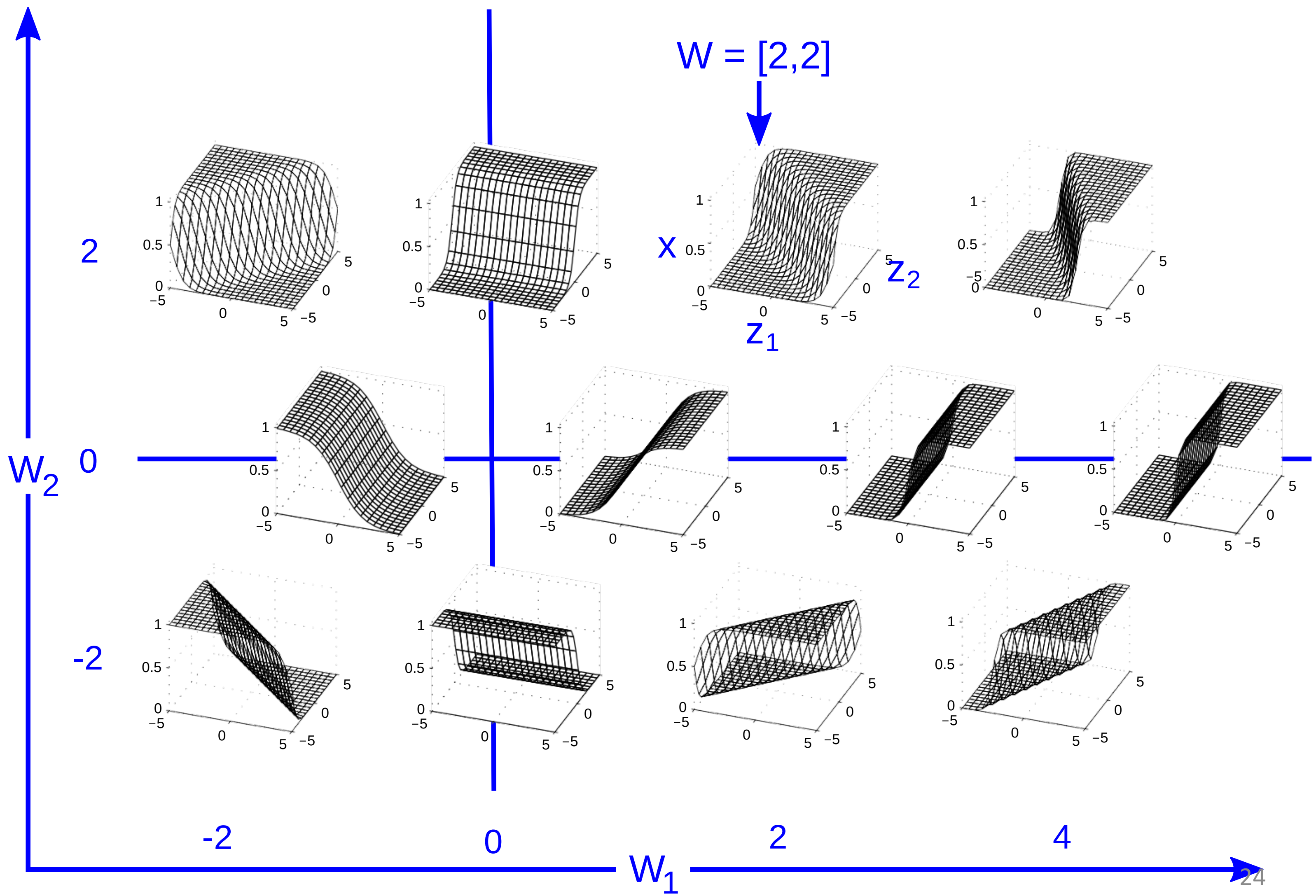
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

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

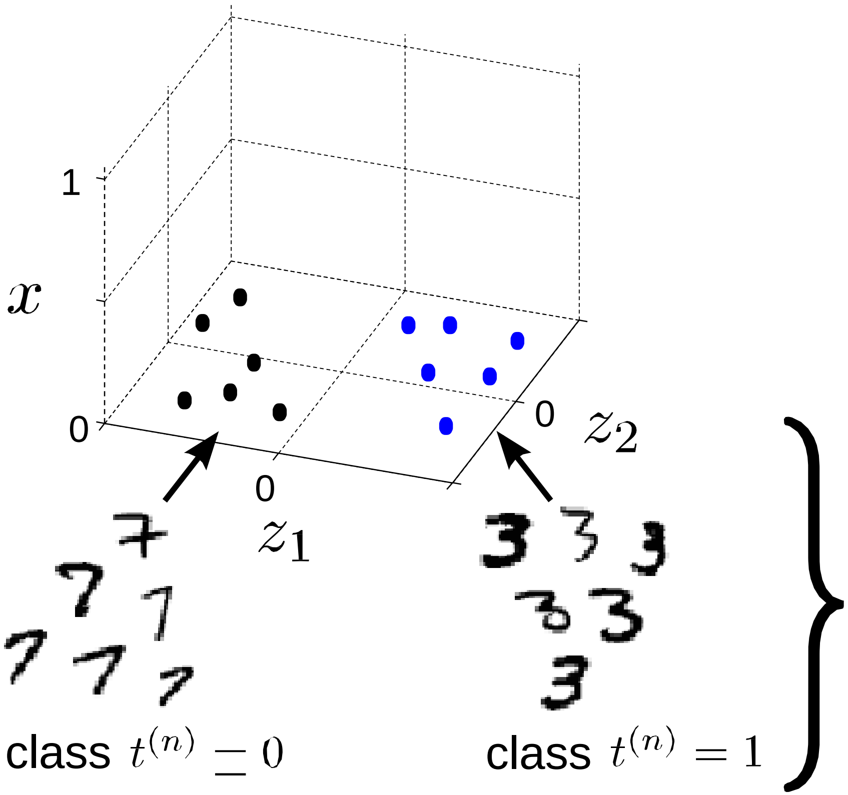


$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

Weight Space of a Single Neuron



Training a Single Neuron

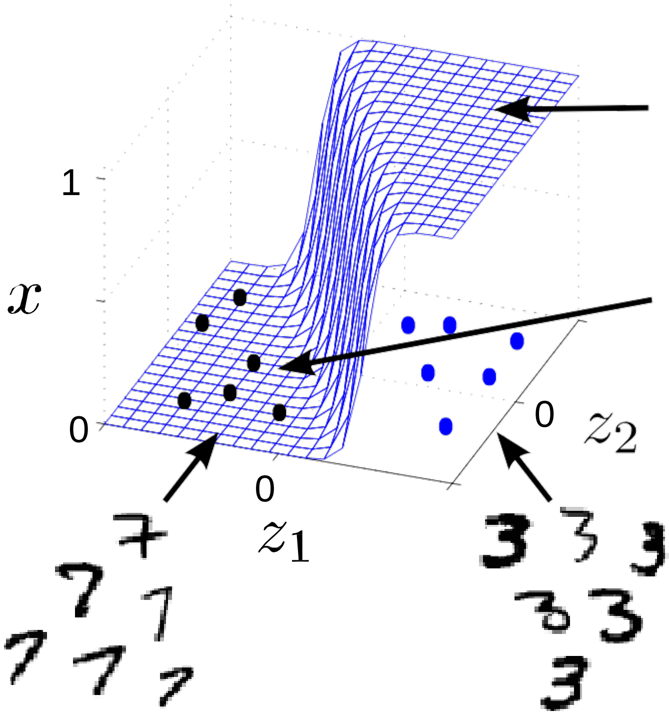


training data

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

inputs class labels

Training a Single Neuron



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

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

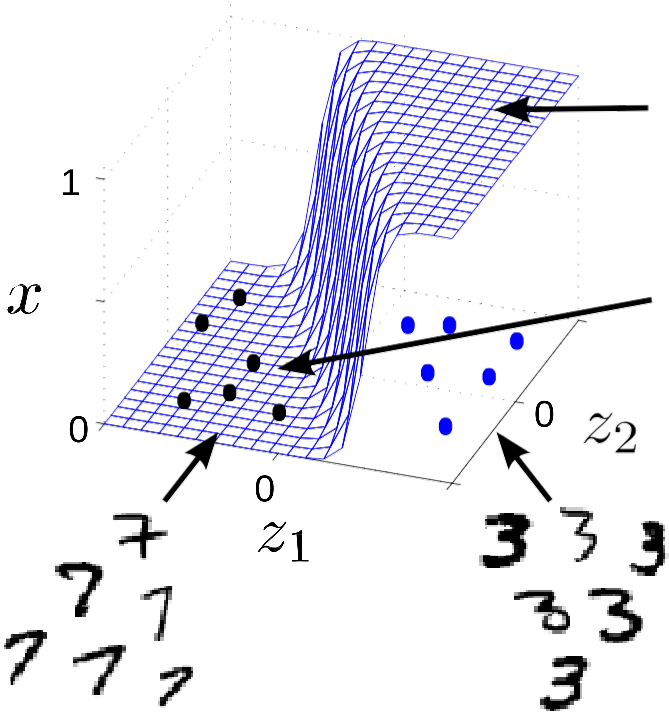
inputs class labels

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:

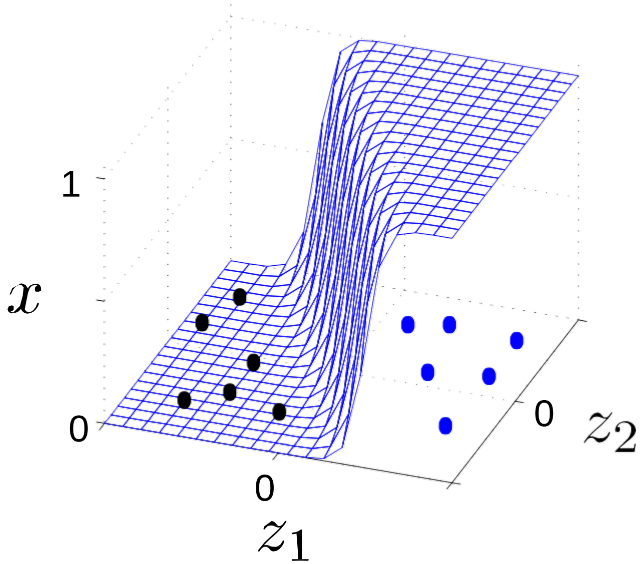
$$G(\boldsymbol{w}) = -\sum_n \left[t^{(n)} \log \mathrm{x}(\boldsymbol{z}^{(n)};\boldsymbol{w}) + (1 - t^n) \log\left(1 - \mathrm{x}(\boldsymbol{z}^{(n)};\boldsymbol{w})\right)\right] \quad \geq 0$$

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:

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

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

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

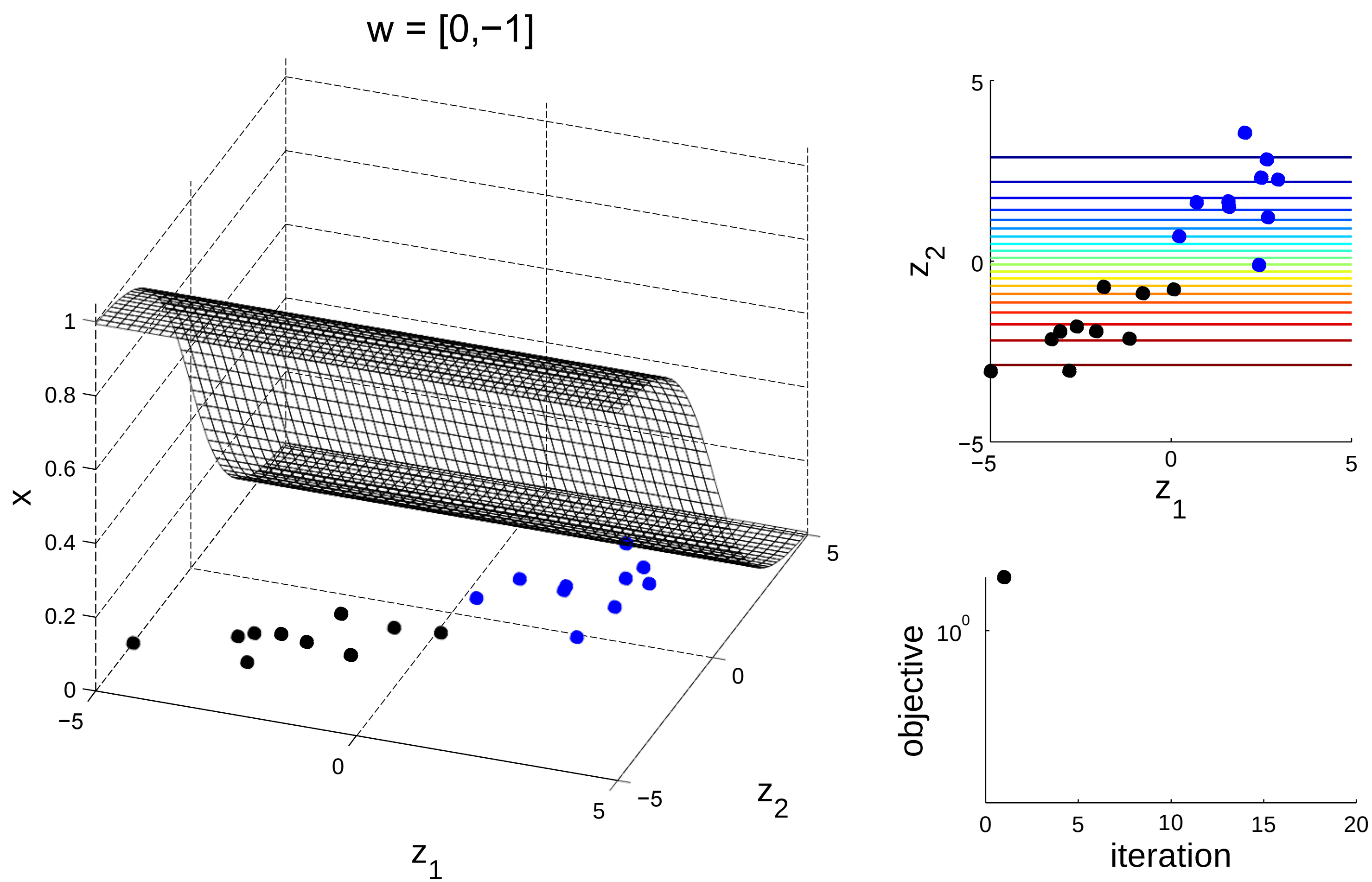
$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{w}}G(\boldsymbol{w}) = \sum_n \frac{\mathrm{d}G(\boldsymbol{w})}{\mathrm{d}x^{(n)}}\frac{\mathrm{d}x^{(n)}}{\mathrm{d}\boldsymbol{w}} = -\sum_n (t^{(n)} - x^{(n)})\boldsymbol{z}^{(n)}$$

= prediction error x feature

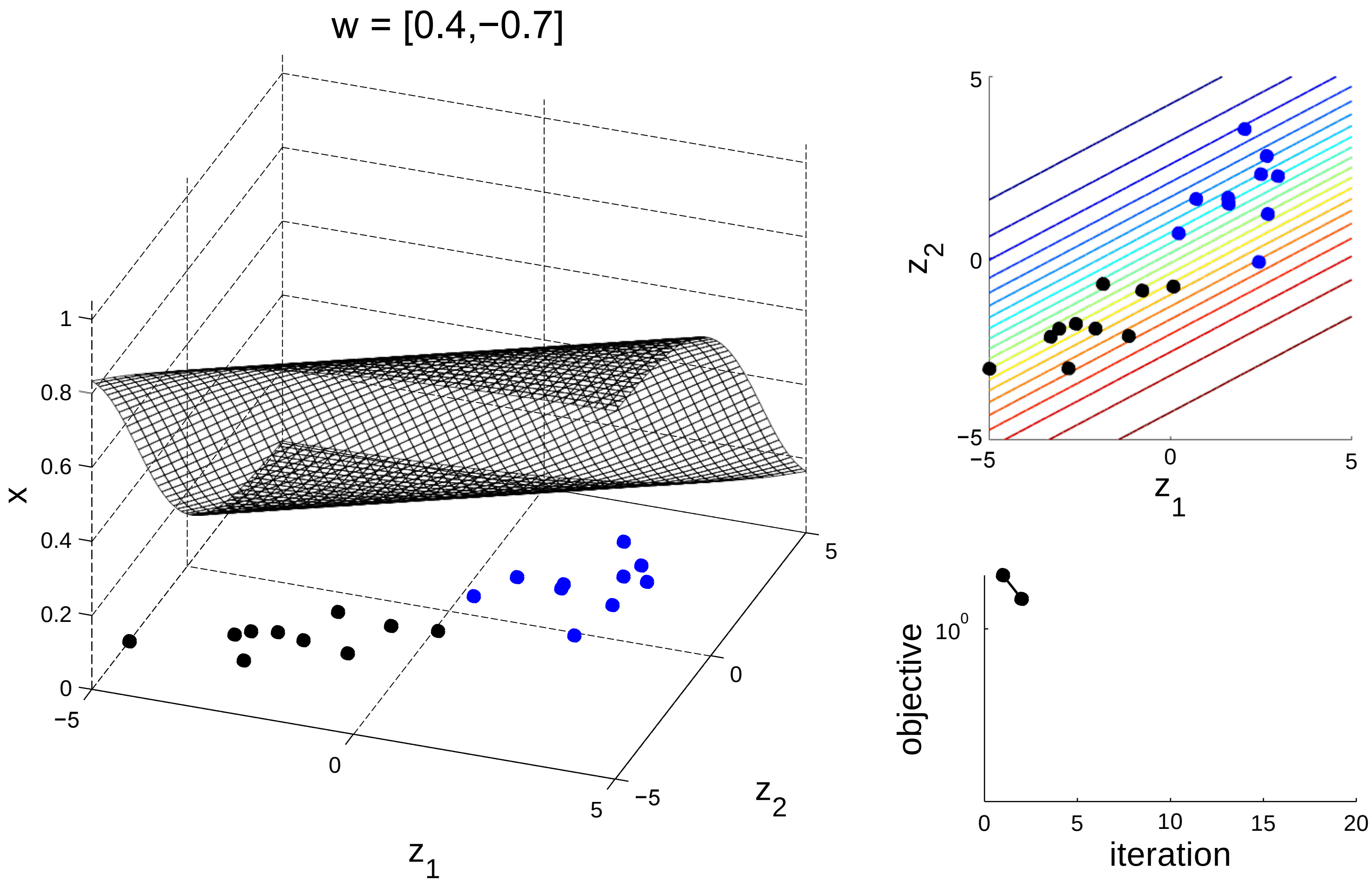
$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{w}}G(\boldsymbol{w})$$

iteratively step down the objective (gradient points up hill)

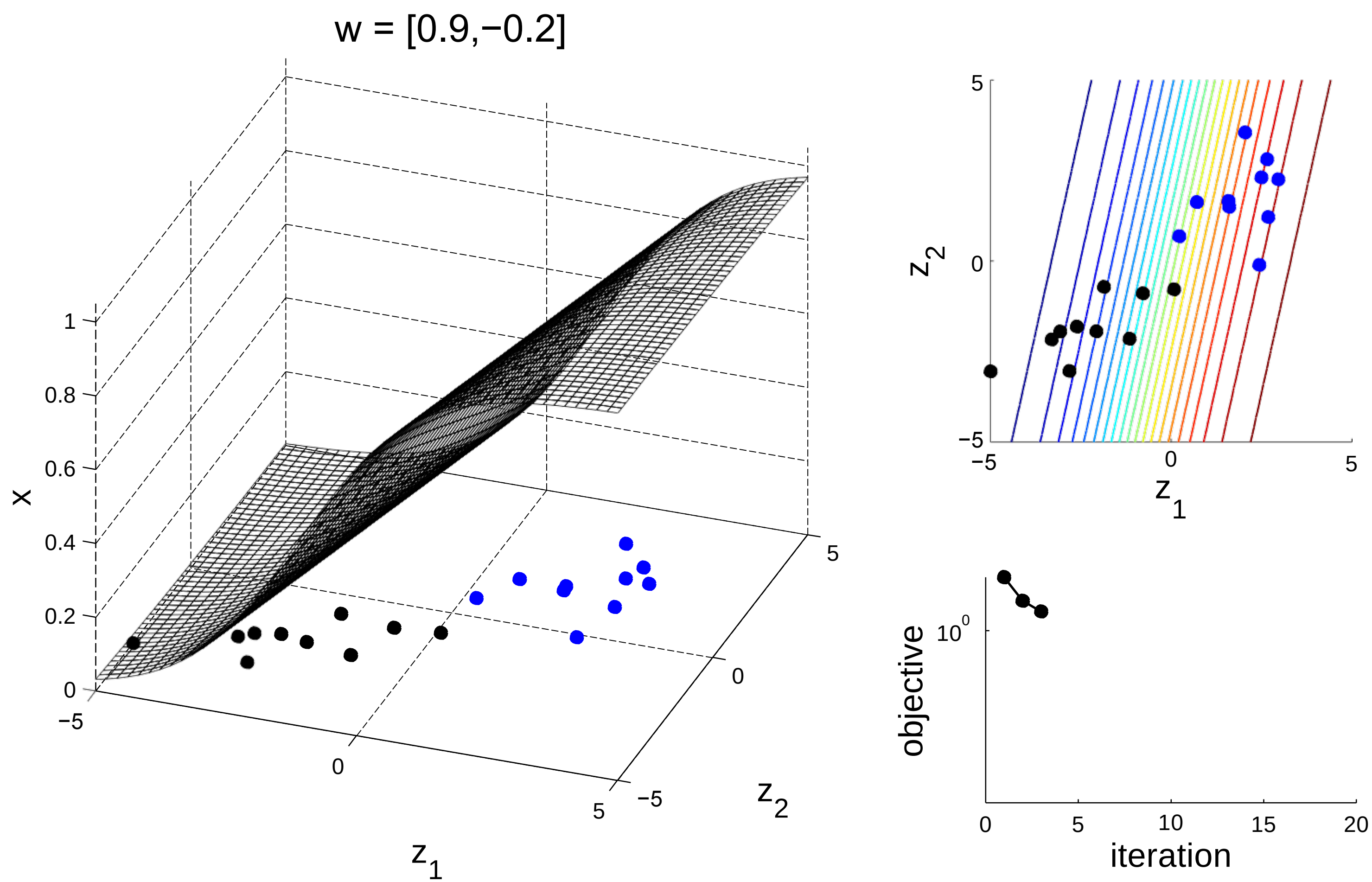
Training a Single Neuron



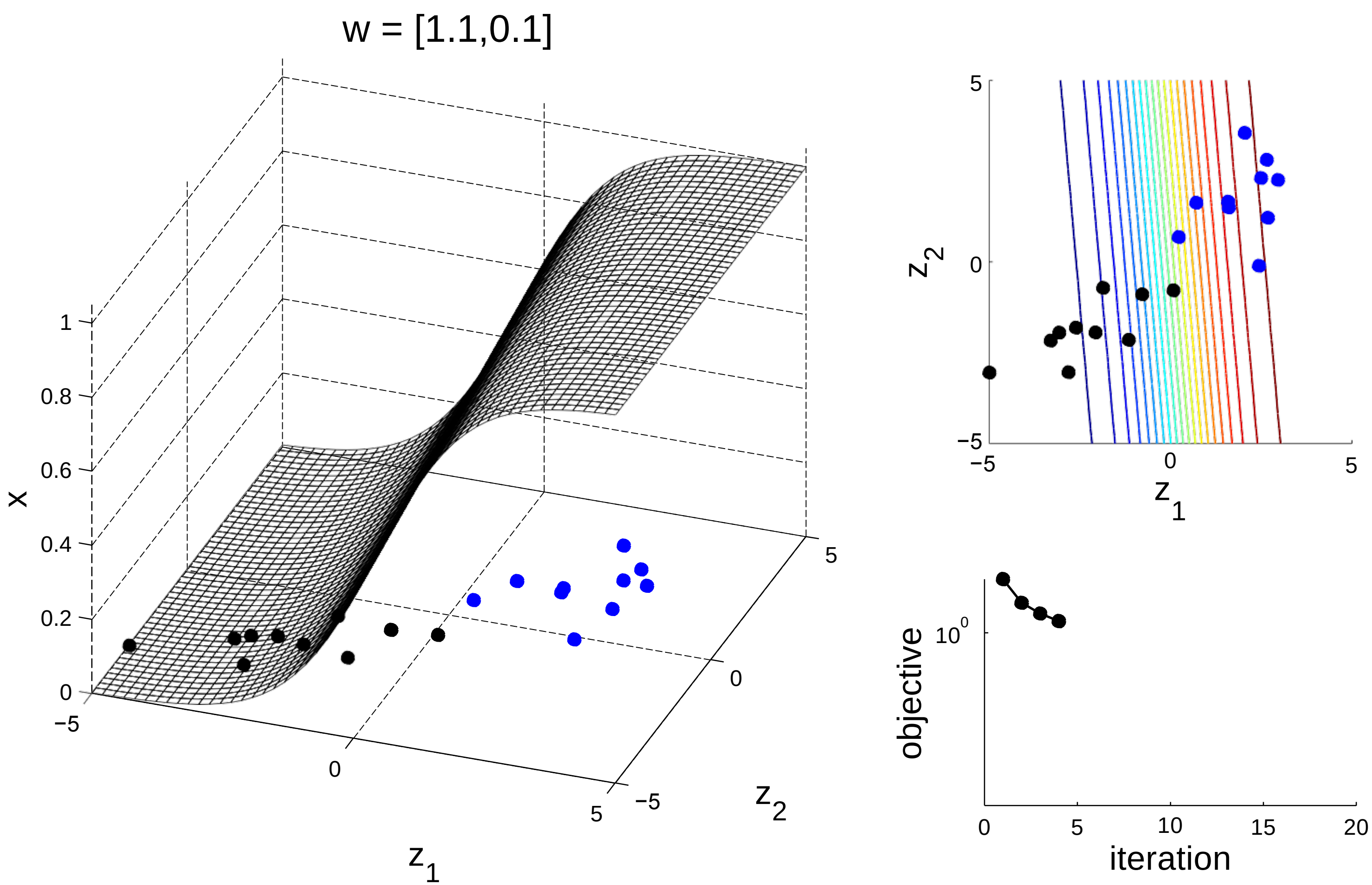
Training a Single Neuron



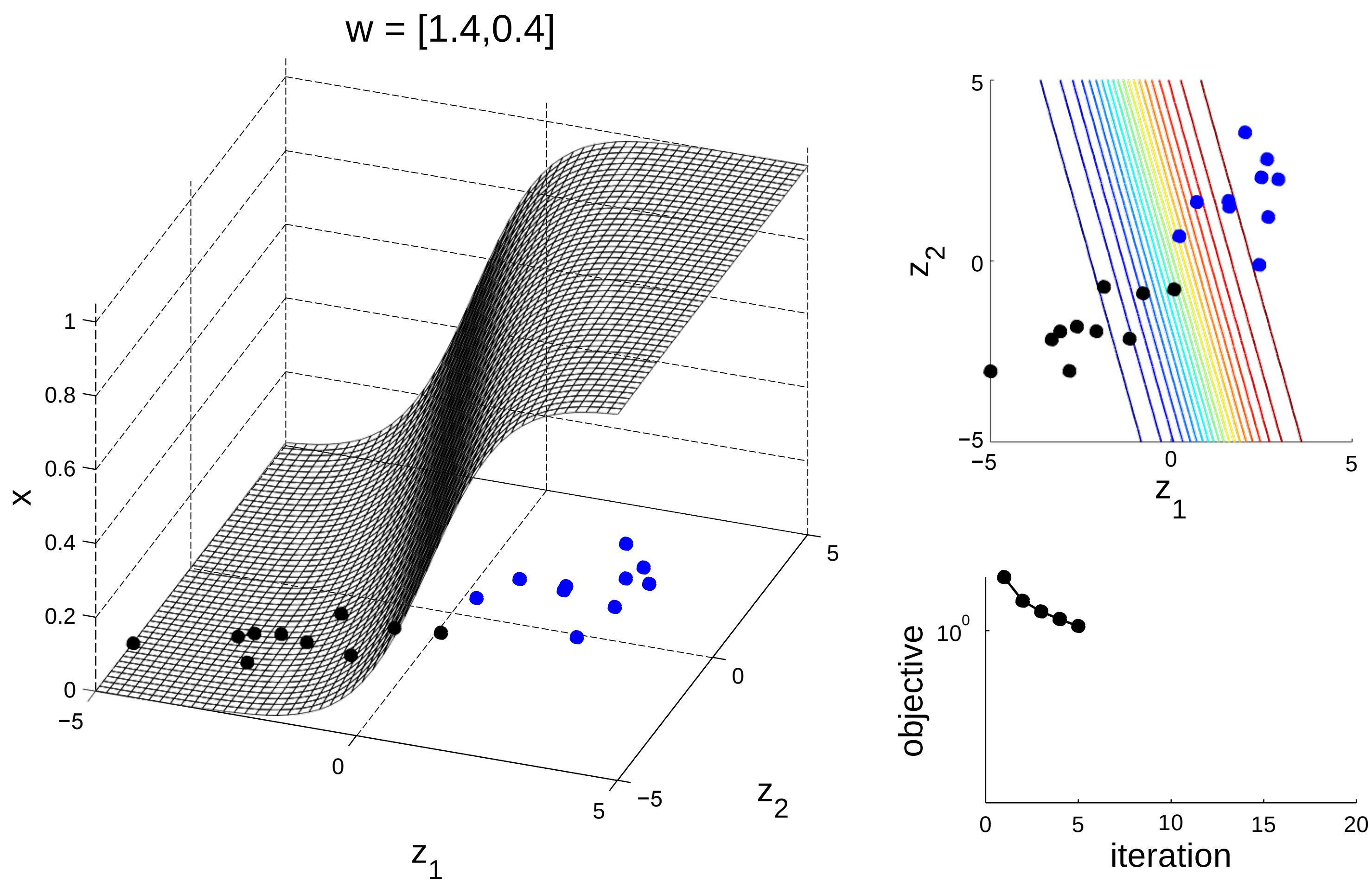
Training a Single Neuron



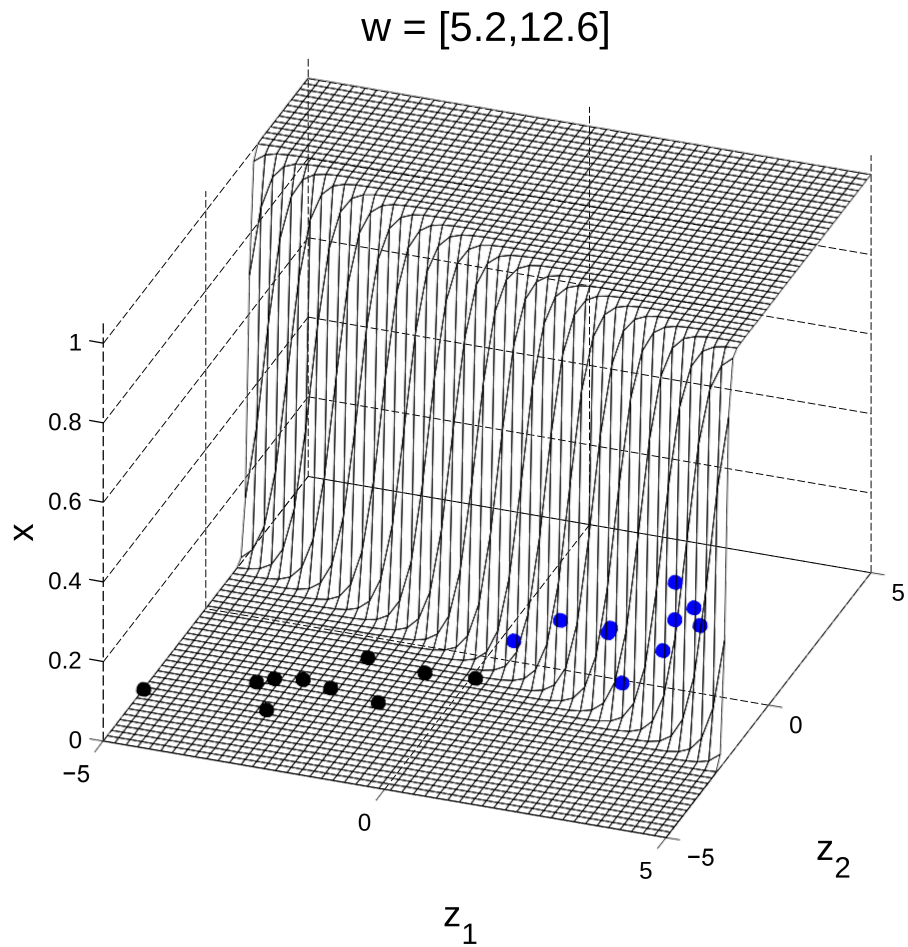
Training a Single Neuron

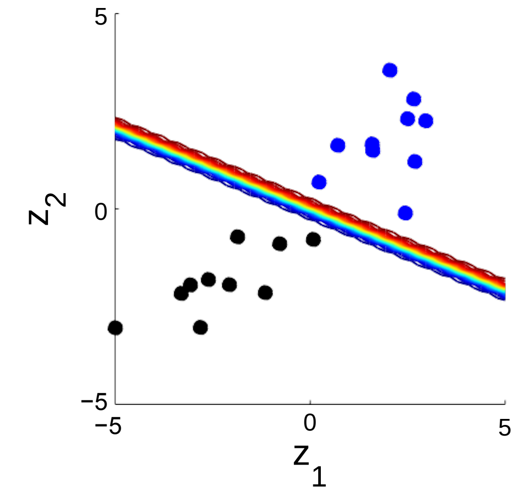


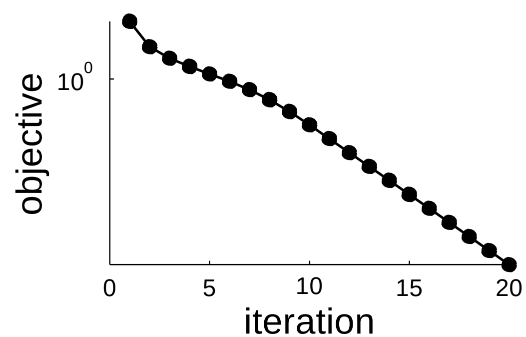
Training a Single Neuron



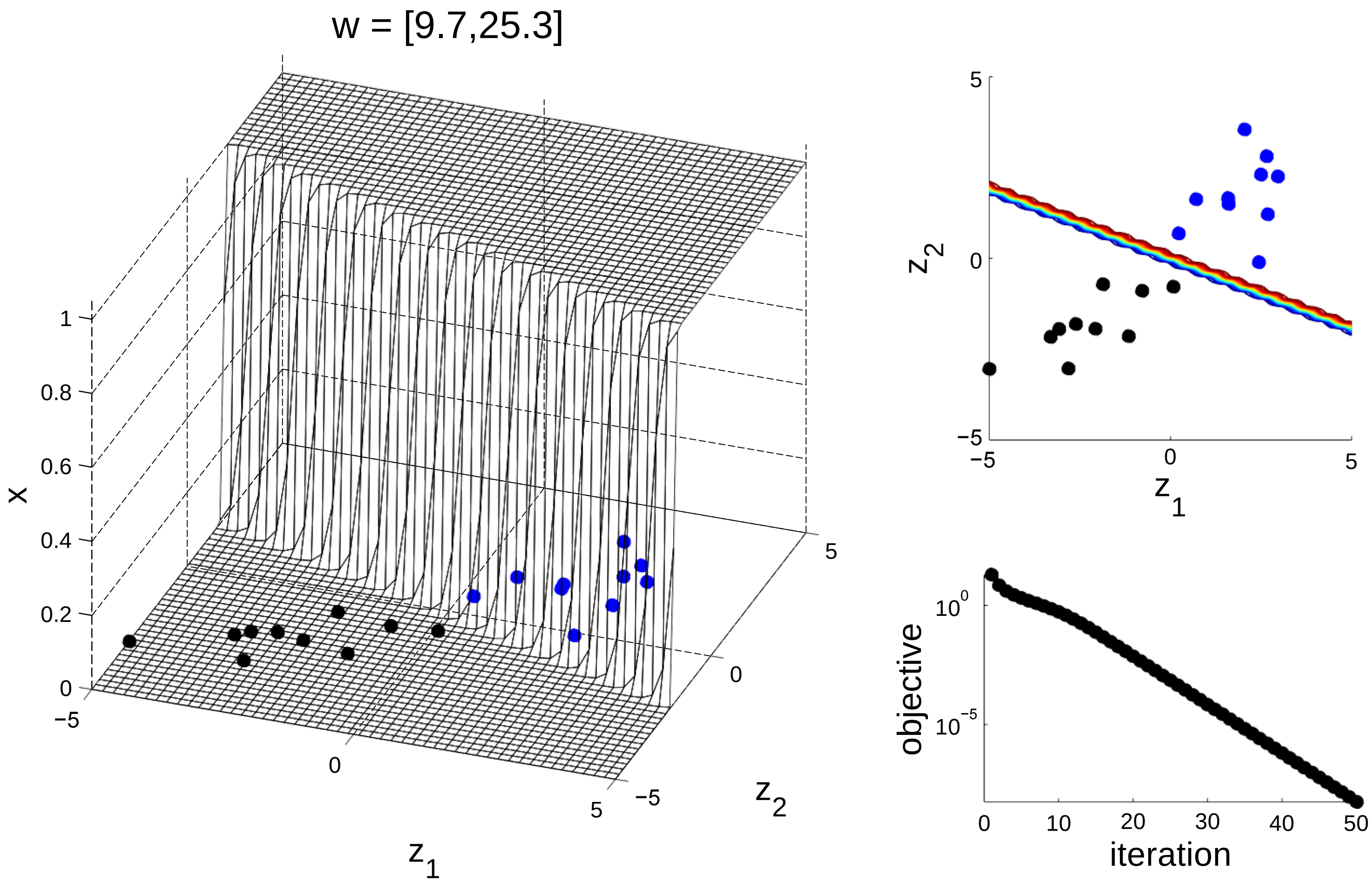
Training a Single Neuron



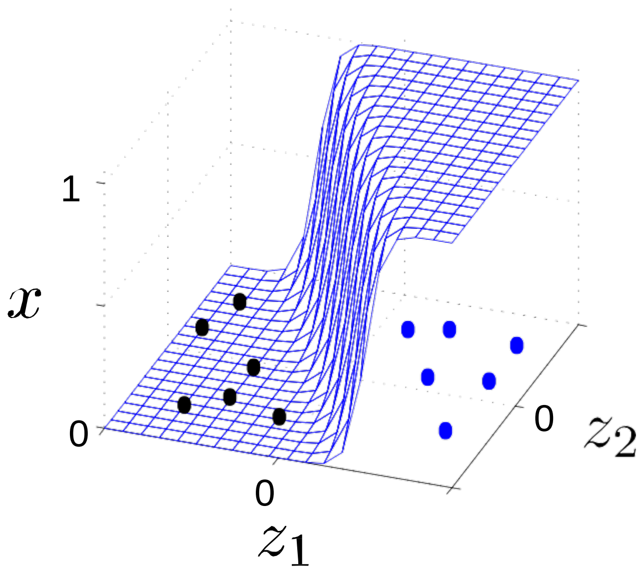




Training a Single Neuron



Overfitting and Weight Decay



training data

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

inputs class labels

objective function:

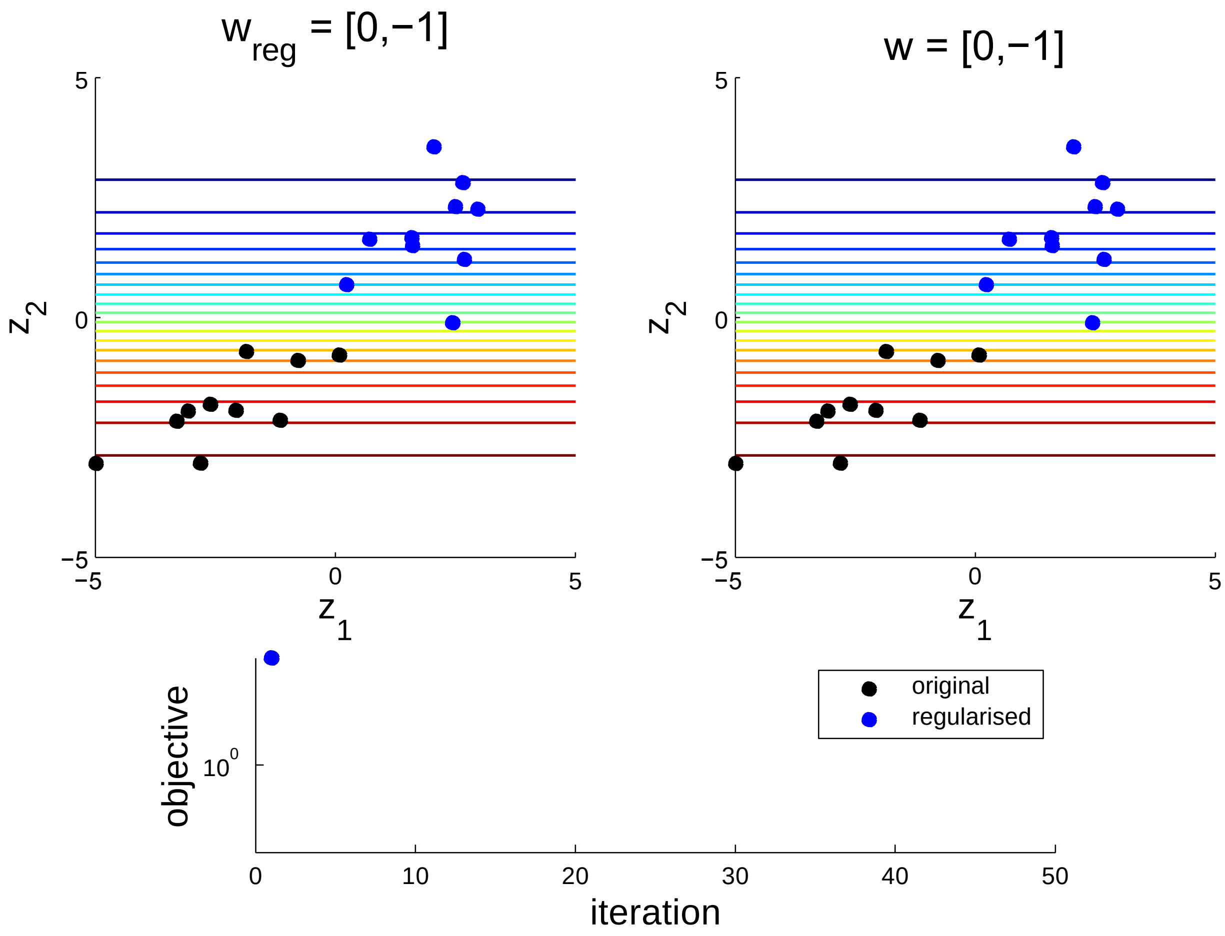
$$G(\boldsymbol{w}) = -\sum_n \left[t^{(n)} \log \mathrm{x}(\boldsymbol{z}^{(n)};\boldsymbol{w}) + (1-t^n) \log \left(1 - \mathrm{x}(\boldsymbol{z}^{(n)};\boldsymbol{w})\right)\right]$$

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

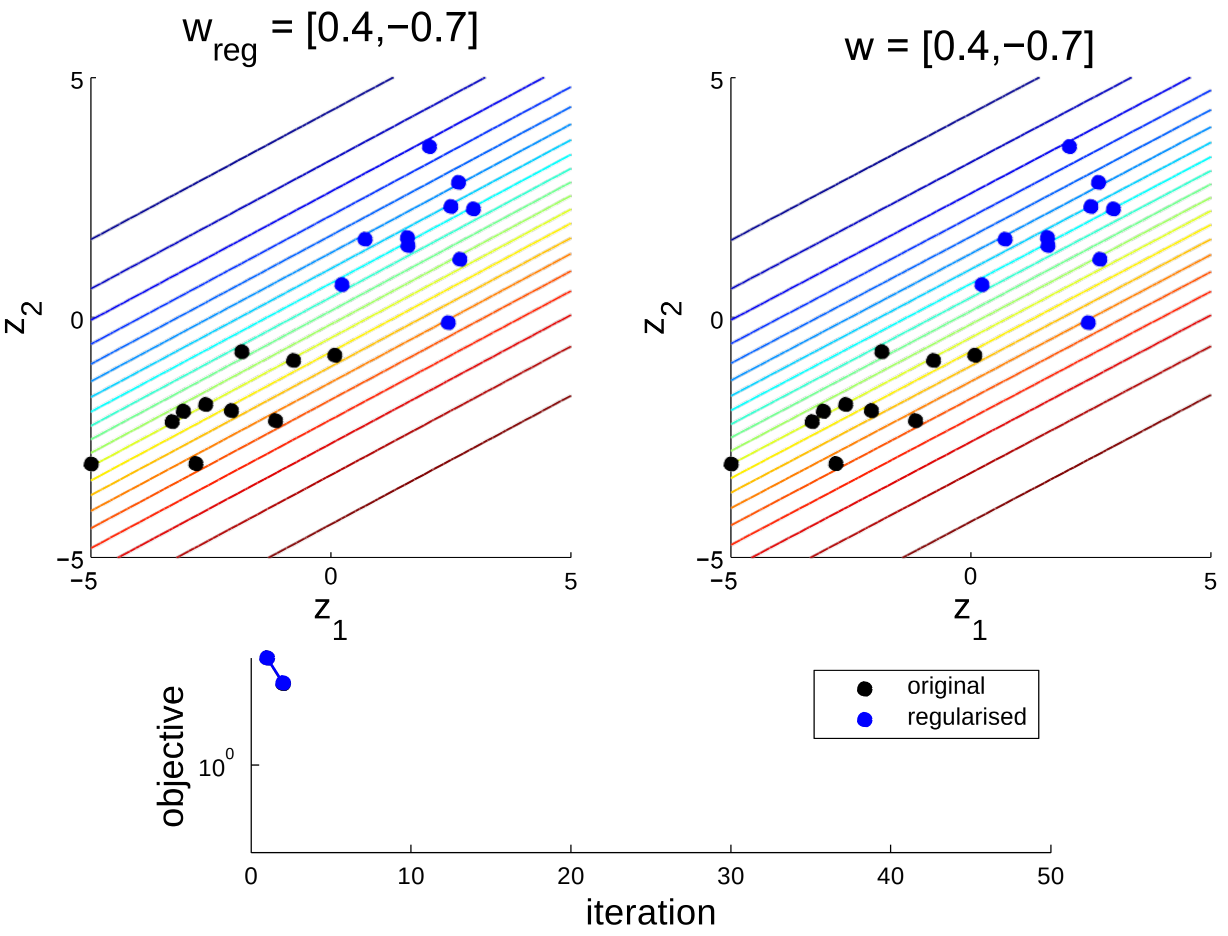
$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} M(\boldsymbol{w}) = \arg\min_{\boldsymbol{w}} \left[G(\boldsymbol{w}) + \alpha E(\boldsymbol{w})\right]$$

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

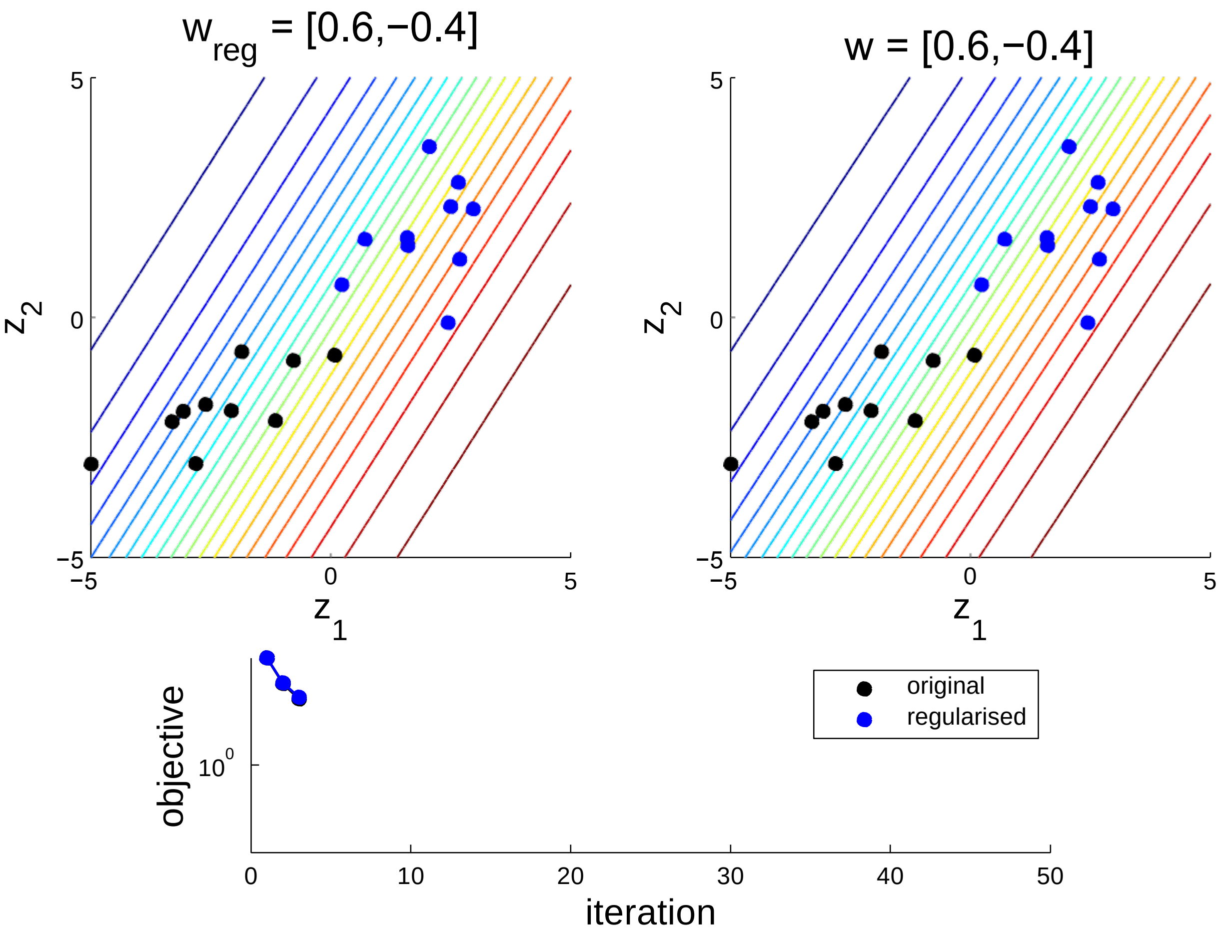
Training a Single Neuron (cont'd)



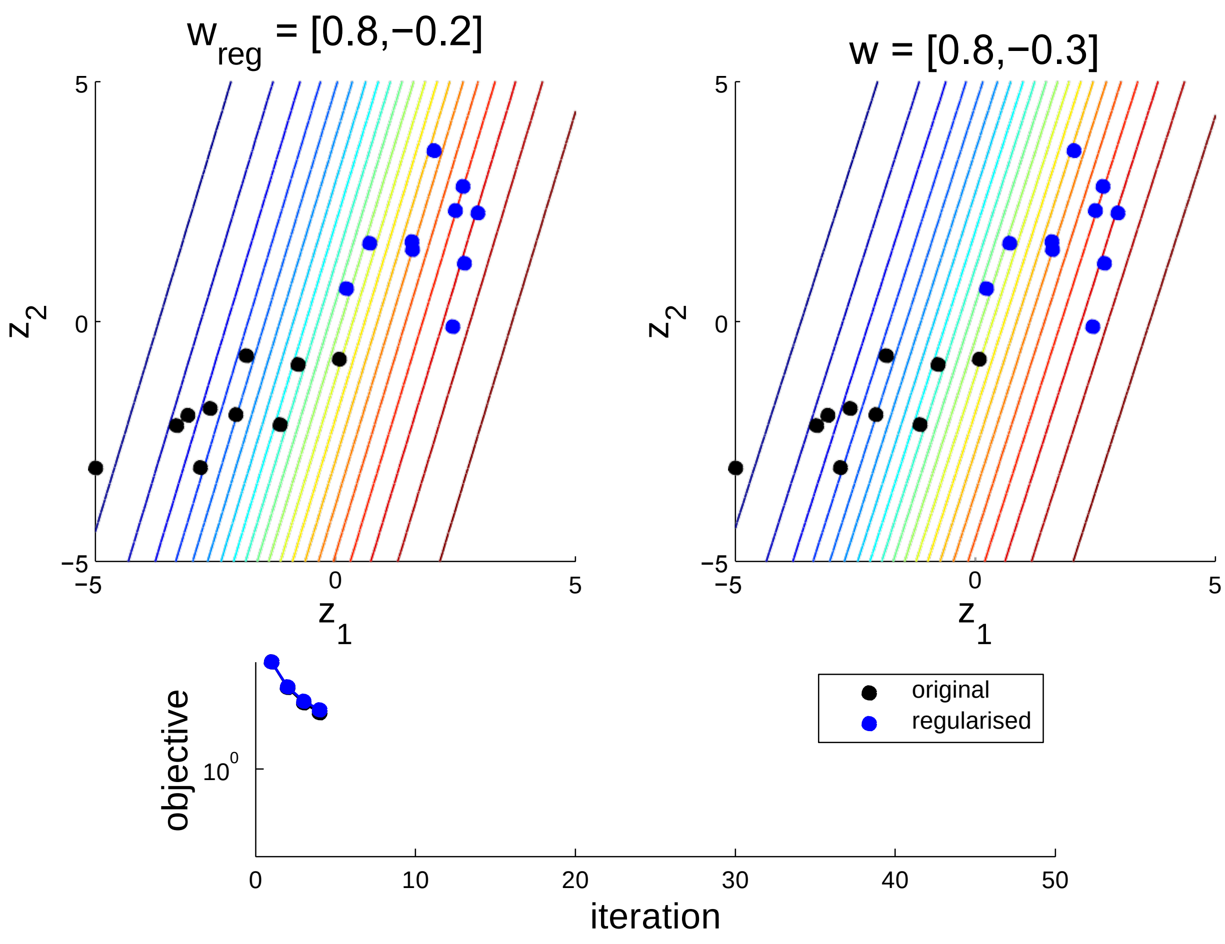
Training a Single Neuron (cont'd)



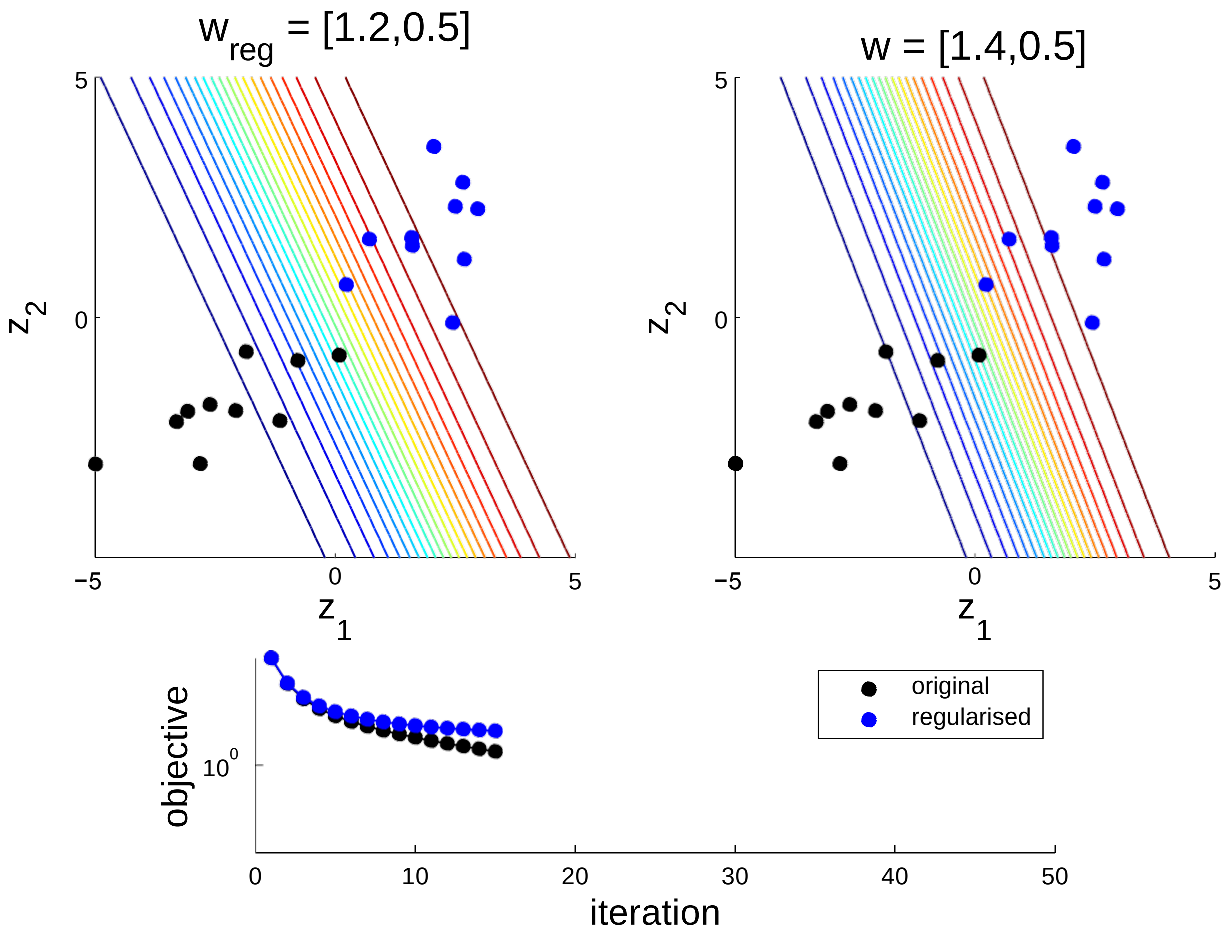
Training a Single Neuron (cont'd)



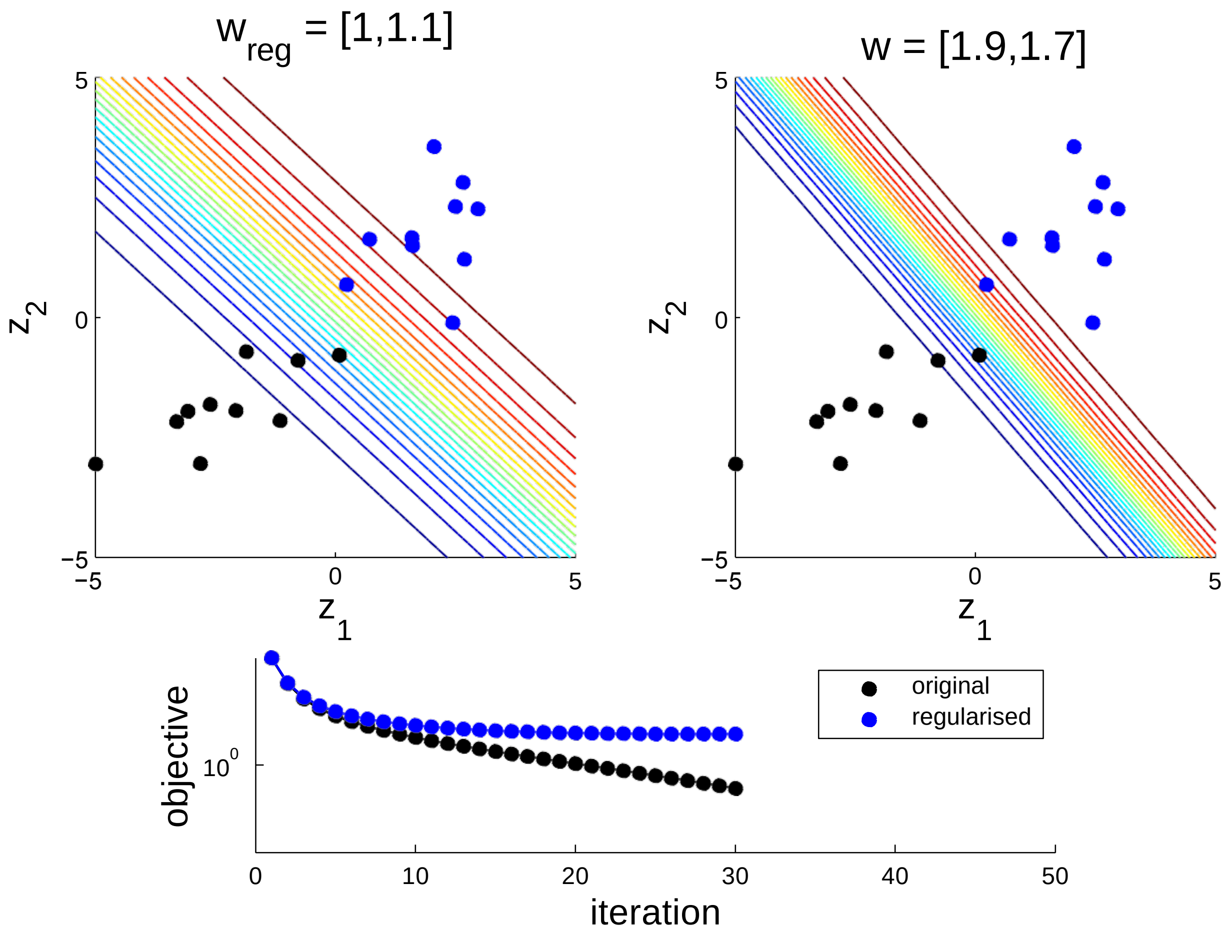
Training a Single Neuron (cont'd)



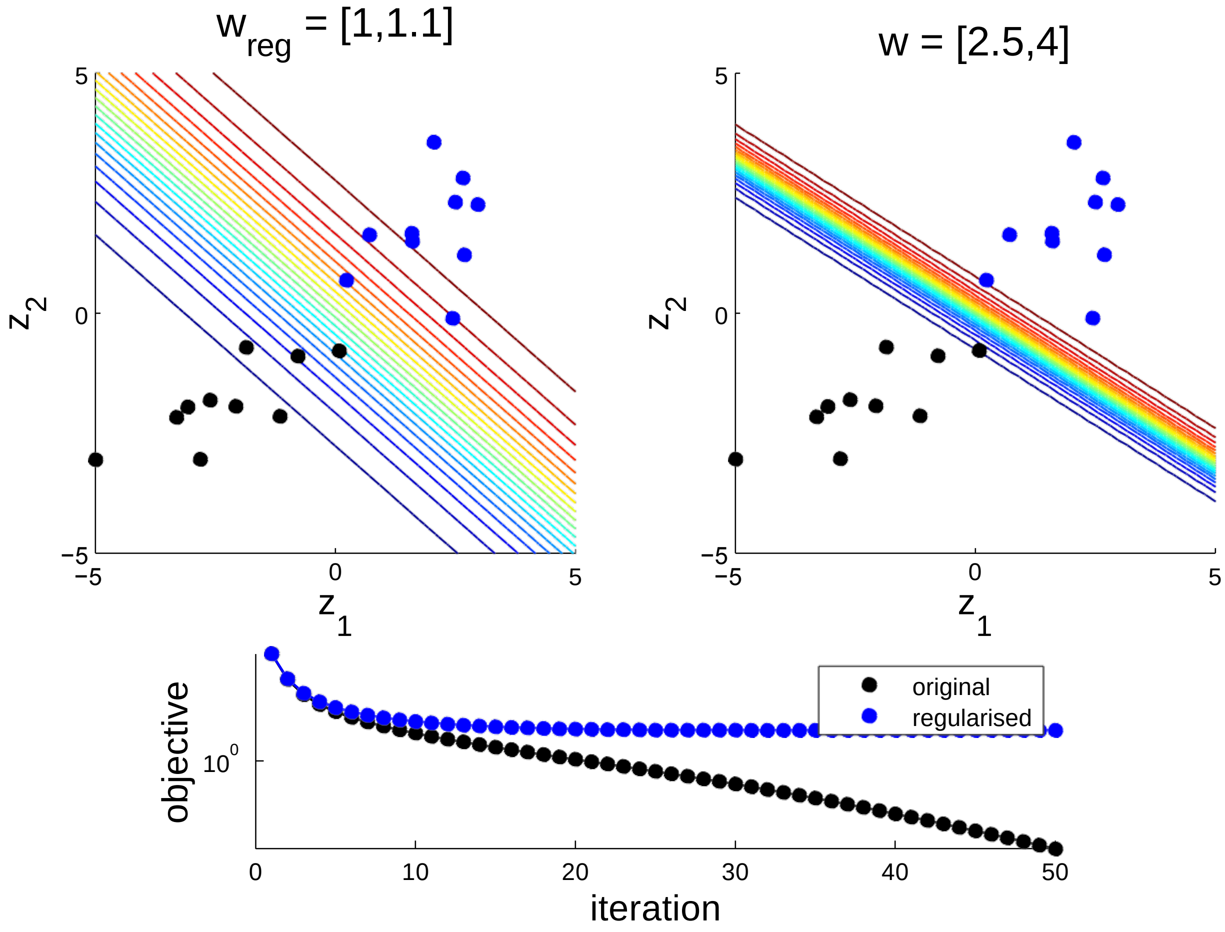
Training a Single Neuron (cont'd)



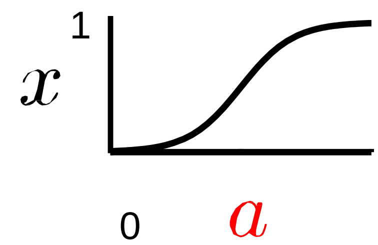
Training a Single Neuron (cont'd)

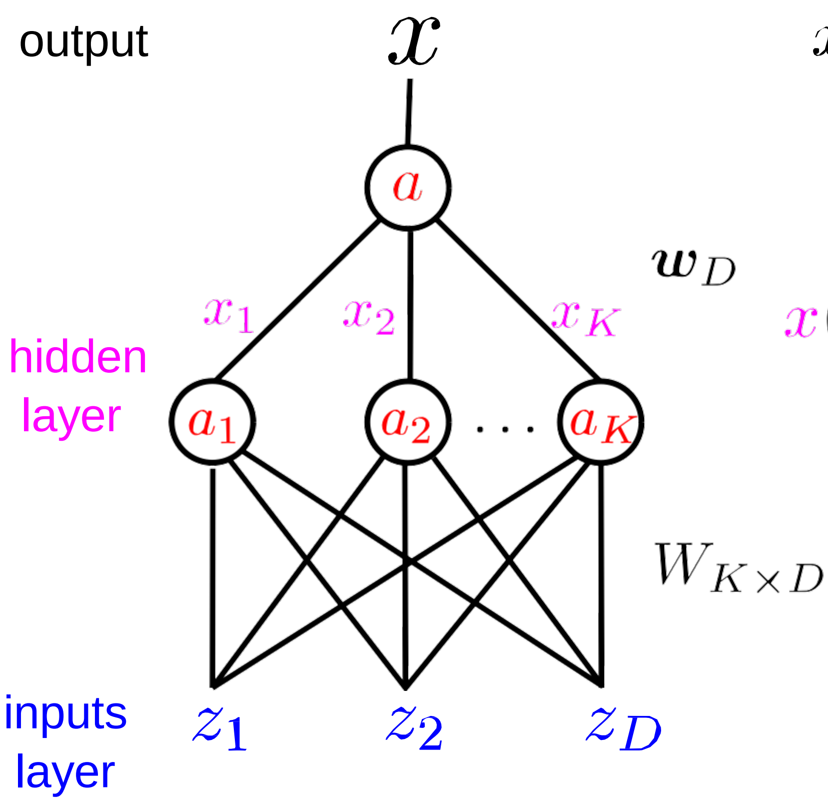


Training a Single Neuron (cont'd)



Single Hidden Layer Neural Networks





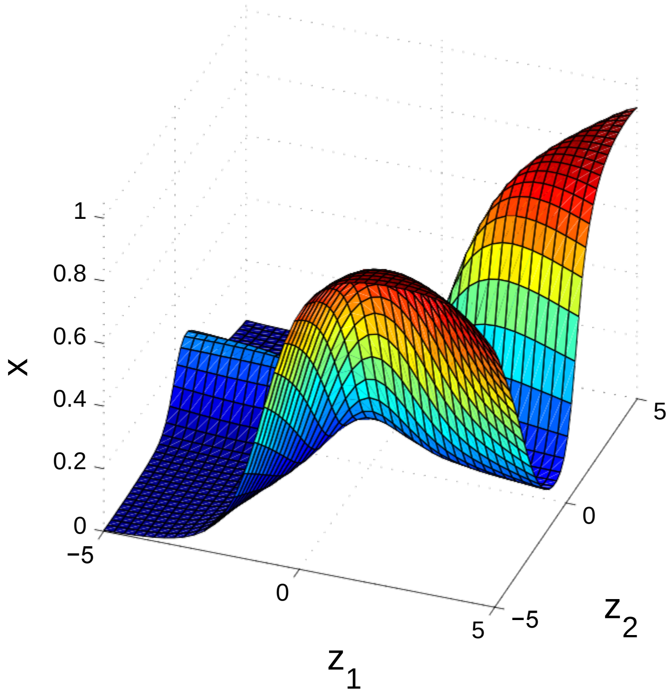
$$x(a) = \frac{1}{1+\exp(-a)}$$

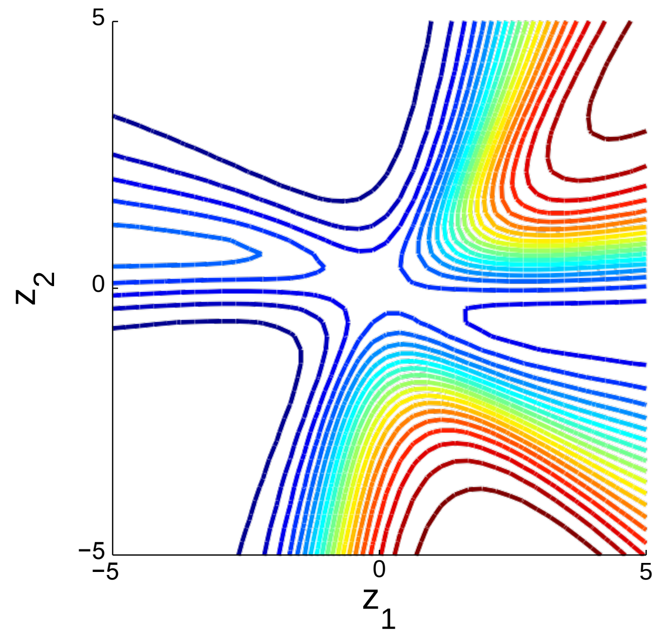
$$a = \sum_{k=1}^{K} w_k x_k$$

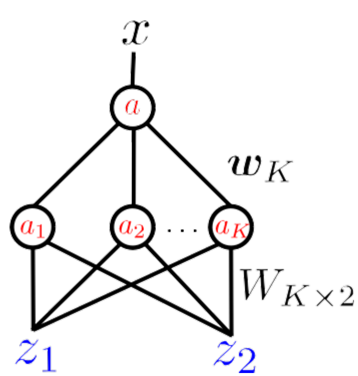
$$x(a_k) = \frac{1}{1+\exp(-a_k)}$$

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

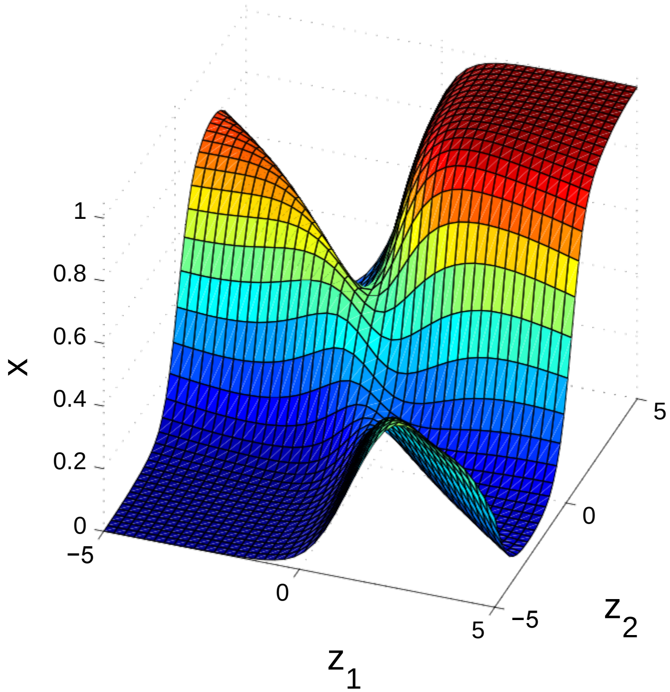
Sampling Random Neural Network Classifiers

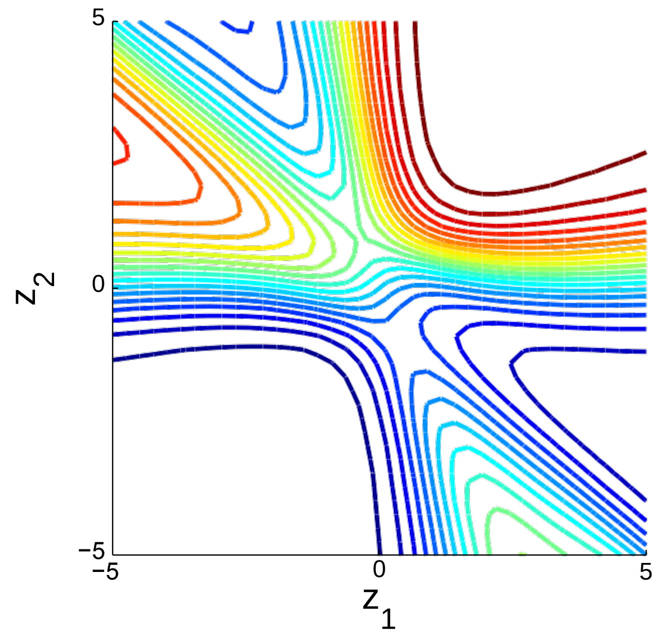


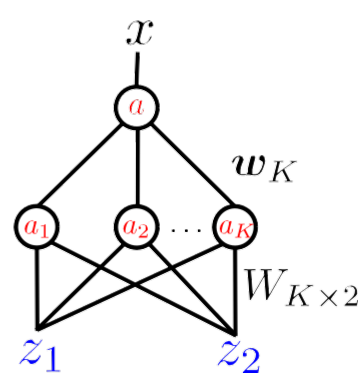




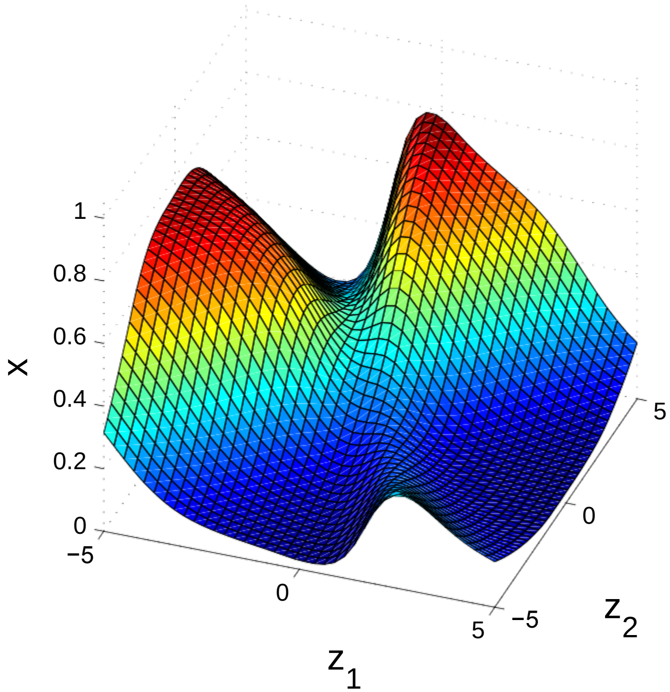
Sampling Random Neural Network Classifiers

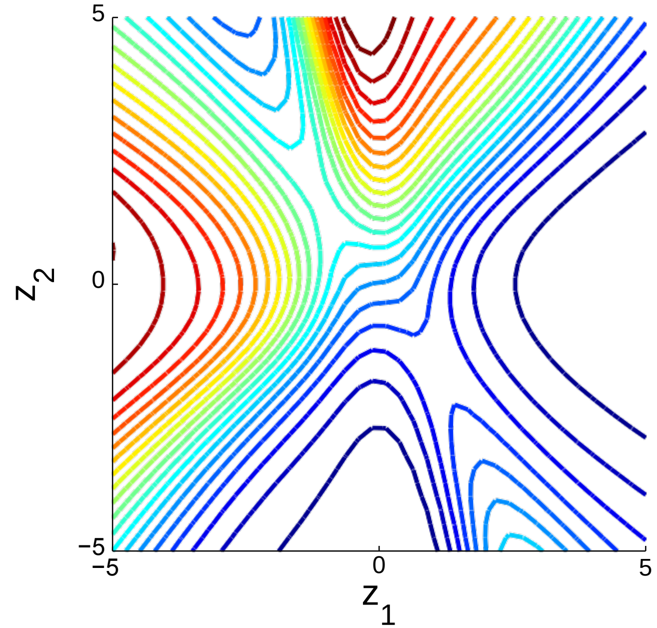


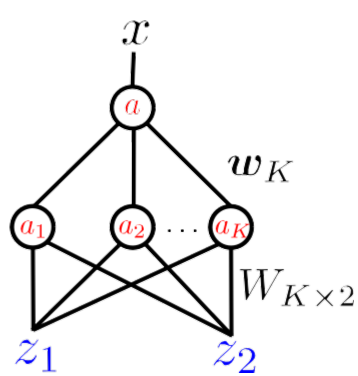




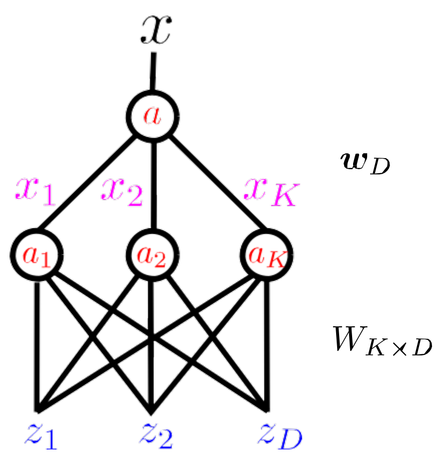
Sampling Random Neural Network Classifiers







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:

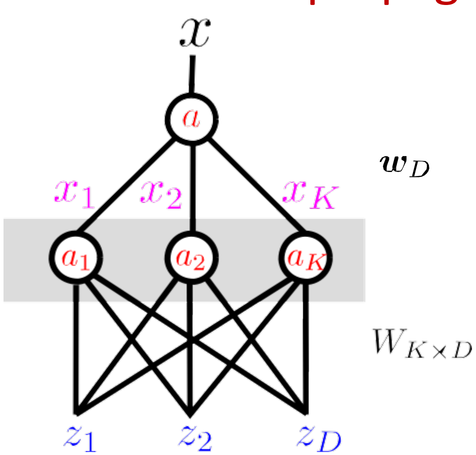
$$G(W, \boldsymbol{w}) = -\sum_n \left[t^{(n)} \log \mathrm{x}^{(n)} + (1 - t^n) \log \left(1 - \mathrm{x}^{(n)}\right)\right]$$ likelihood same as before

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

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

Training a Neural Network with a Single Hidden Layer

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



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

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

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

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

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

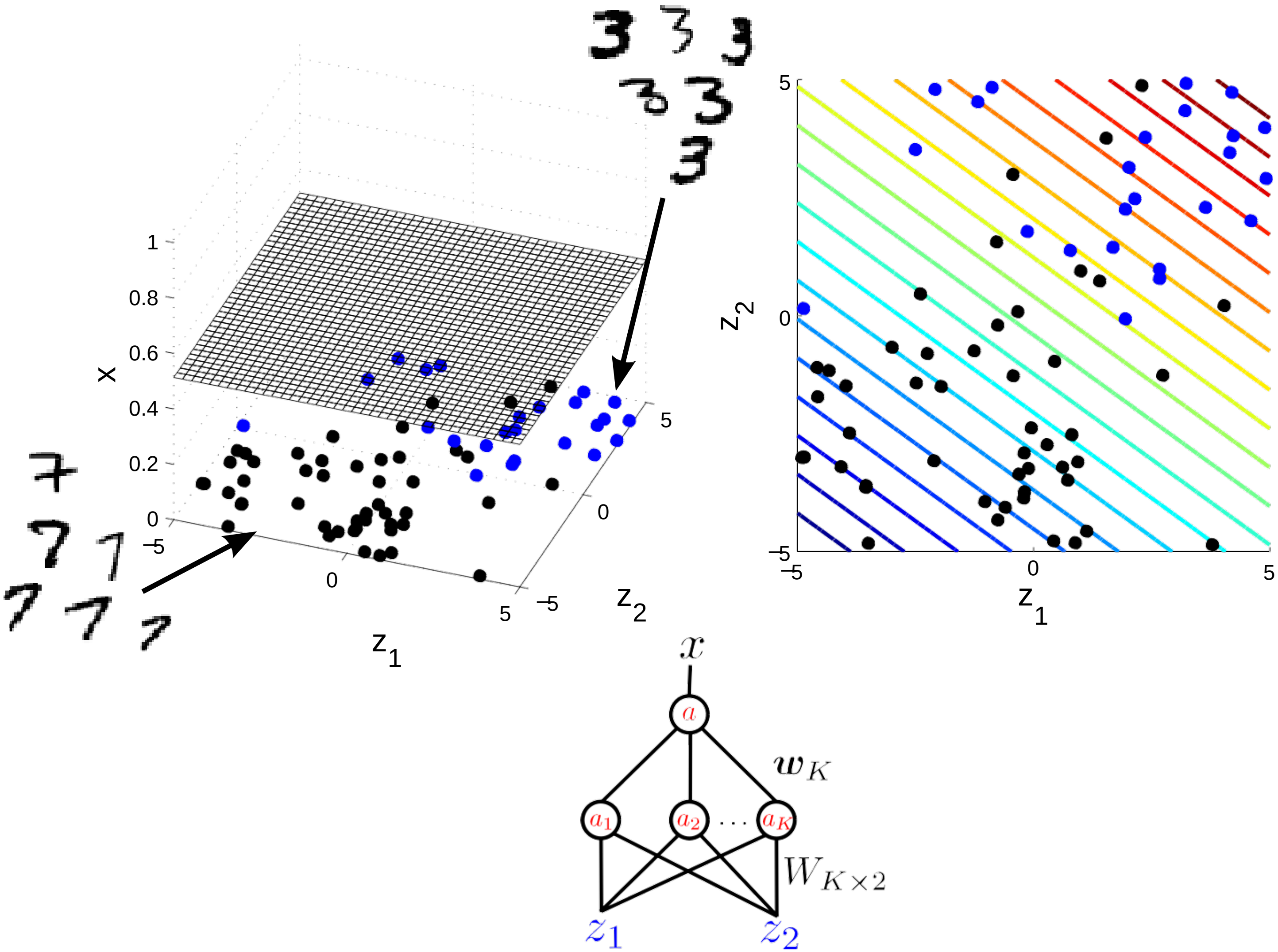
$$E(W, \boldsymbol{w}) = \tfrac{1}{2}\sum_i w_i^2 + \tfrac{1}{2}\sum_{ij} W_{ij}^2$$ regulariser discourages extreme weights

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

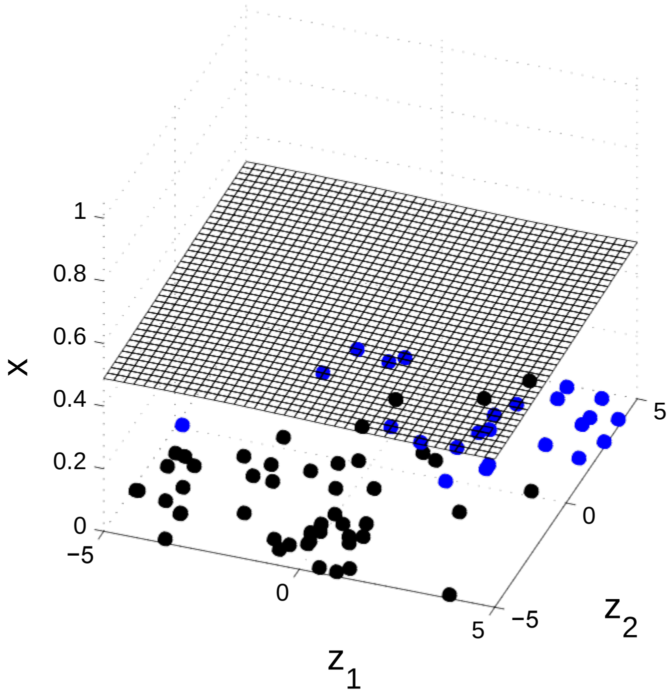
$$\frac{\mathrm{d}G(W,\boldsymbol{w})}{\mathrm{d}W_{ij}} = \sum_n \frac{\mathrm{d}G(W,\boldsymbol{w})}{\mathrm{d}x^{(n)}}\frac{\mathrm{d}x^{(n)}}{\mathrm{d}W_{ij}} = \sum_n \frac{\mathrm{d}G(W,\boldsymbol{w})}{\mathrm{d}x^{(n)}}\frac{\mathrm{d}x^{(n)}}{\mathrm{d}a^{(n)}}\frac{\mathrm{d}a^{(n)}}{\mathrm{d}W_{ij}}$$

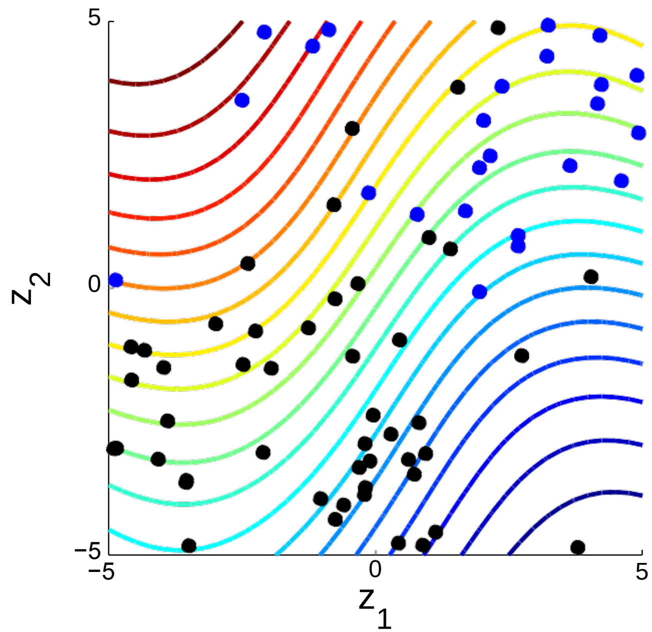
$$= \sum_n \frac{\mathrm{d}G(W,\boldsymbol{w})}{\mathrm{d}x^{(n)}}\frac{\mathrm{d}x^{(n)}}{\mathrm{d}a^{(n)}}\frac{\mathrm{d}a^{(n)}}{\mathrm{d}x_i^{(n)}}\frac{\mathrm{d}x_i^{(n)}}{\mathrm{d}W_{ij}} = \sum_n \frac{\mathrm{d}G(W,\boldsymbol{w})}{\mathrm{d}x^{(n)}}\frac{\mathrm{d}x^{(n)}}{\mathrm{d}a^{(n)}}\frac{\mathrm{d}a^{(n)}}{\mathrm{d}x_i^{(n)}}\frac{\mathrm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

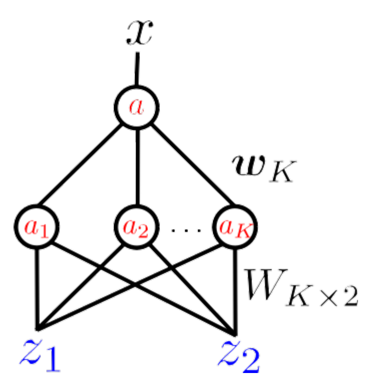
Training a Neural Network with a Single Hidden Layer



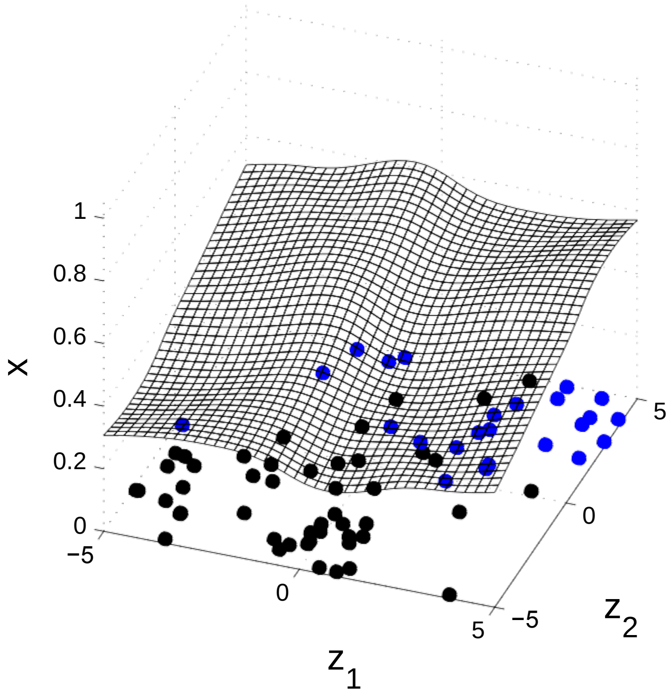
Training a Neural Network with a Single Hidden Layer

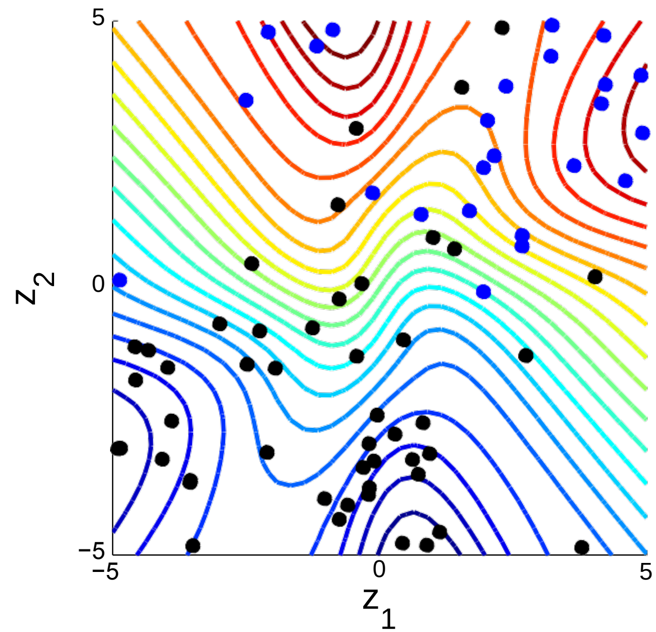


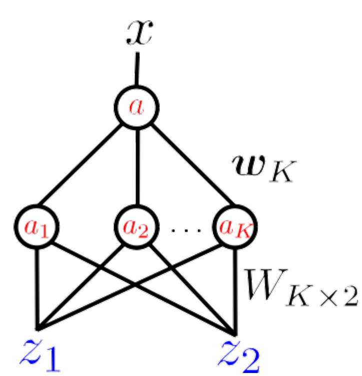




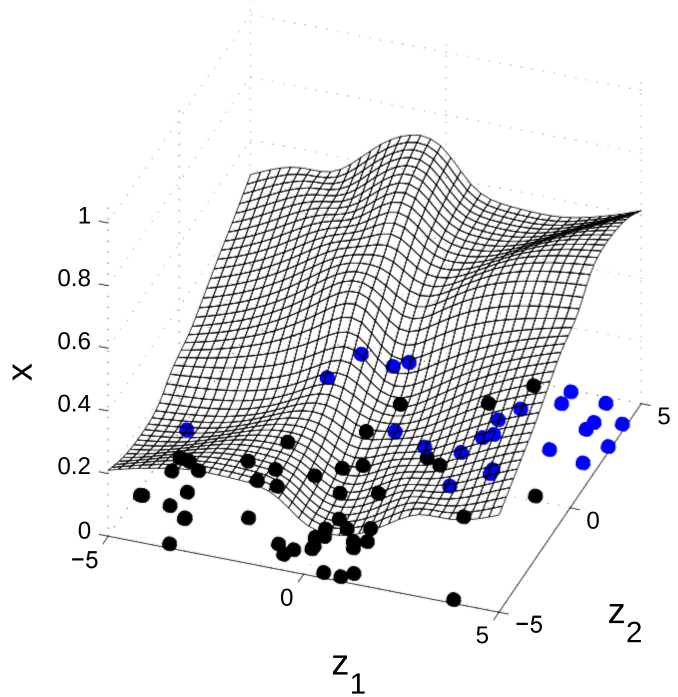
Training a Neural Network with a Single Hidden Layer

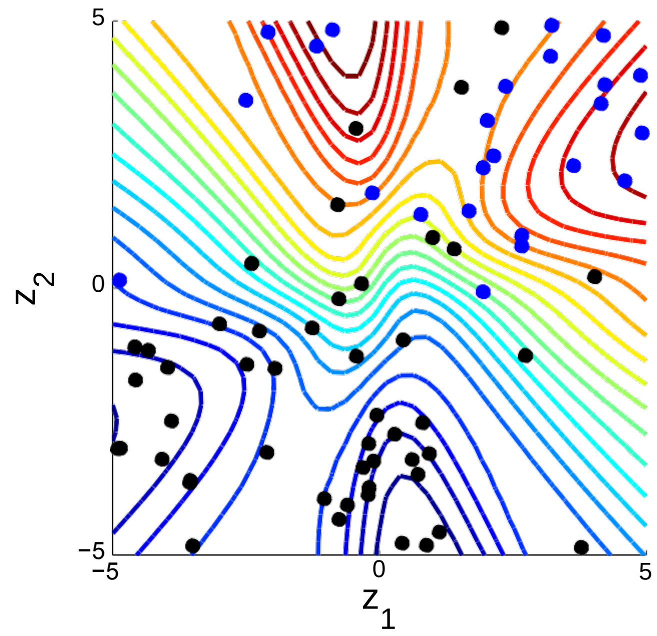


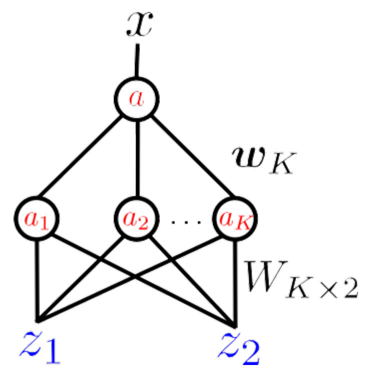




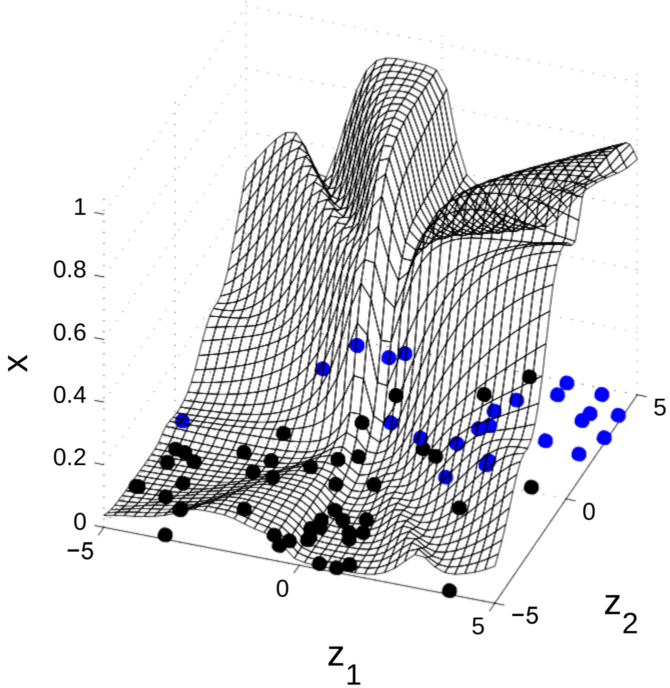
Training a Neural Network with a Single Hidden Layer

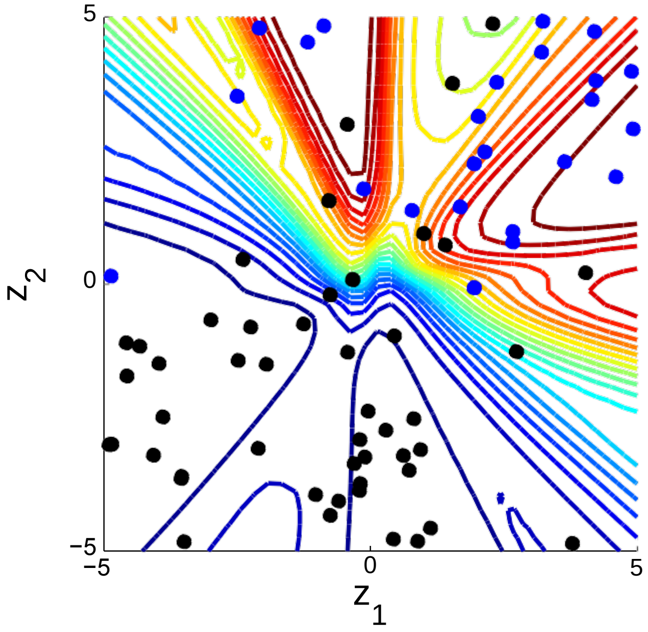


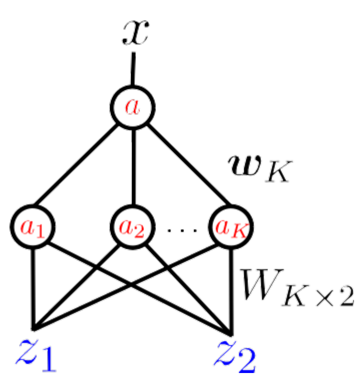




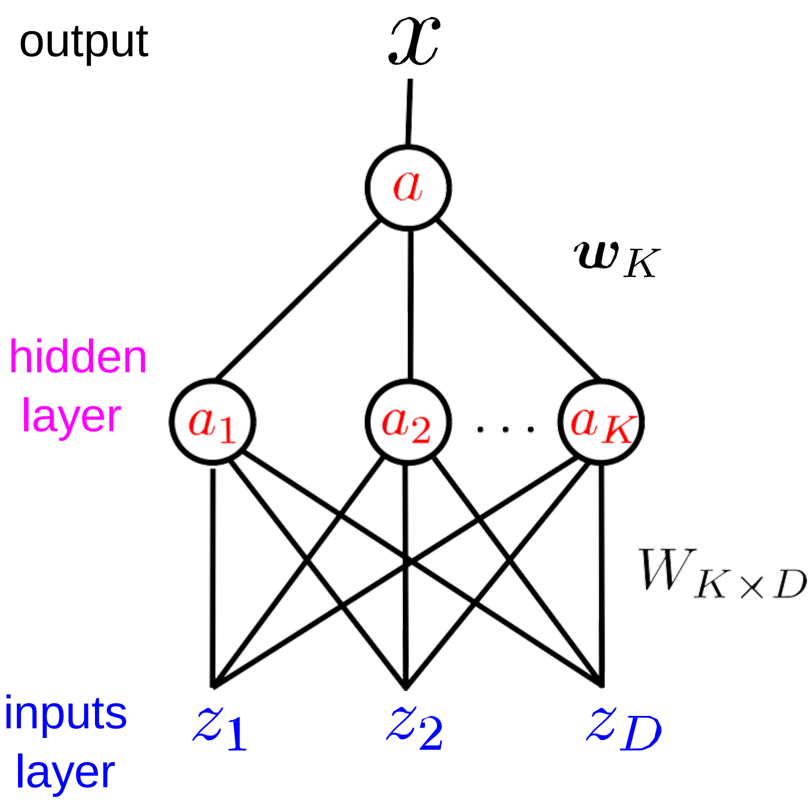
Training a Neural Network with a Single Hidden Layer

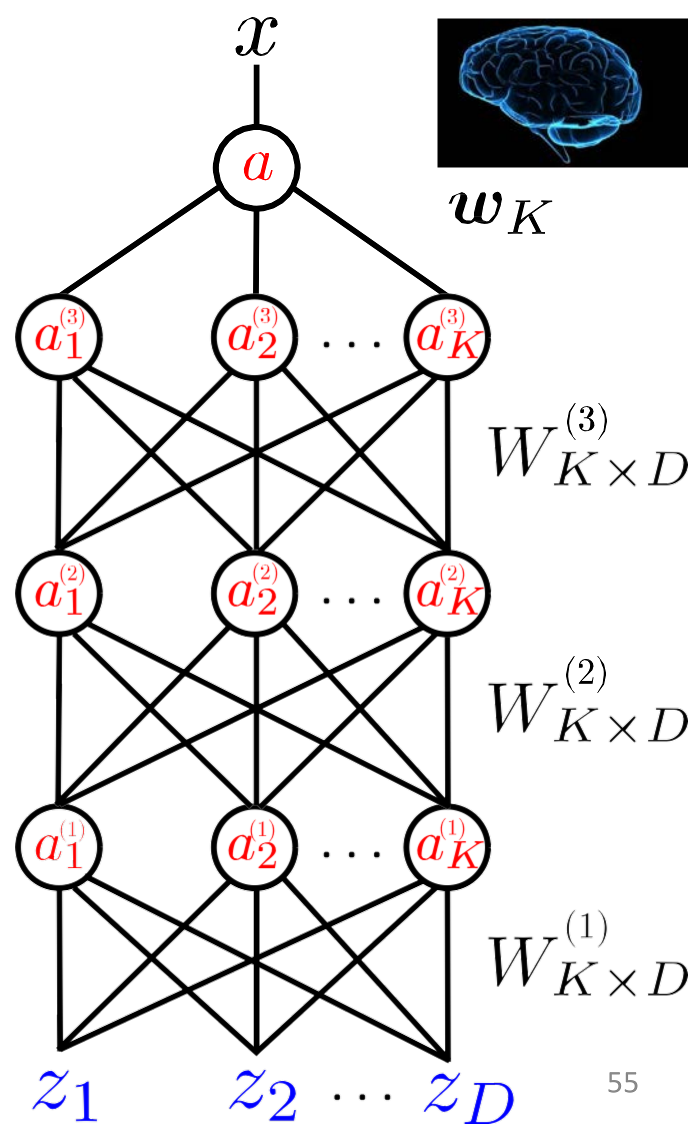






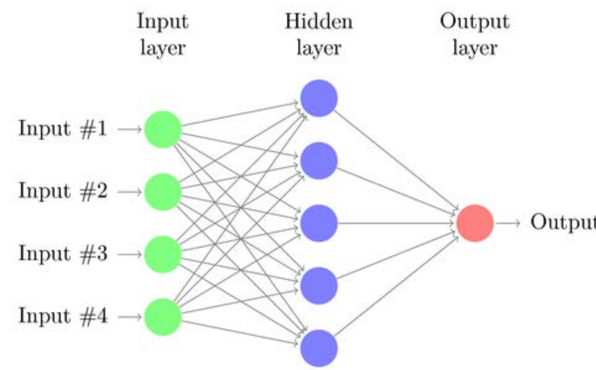
Hierarchical Models with Many Layers

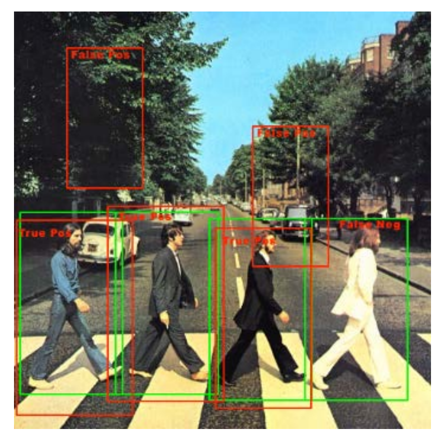




What’s to Be Covered Today…

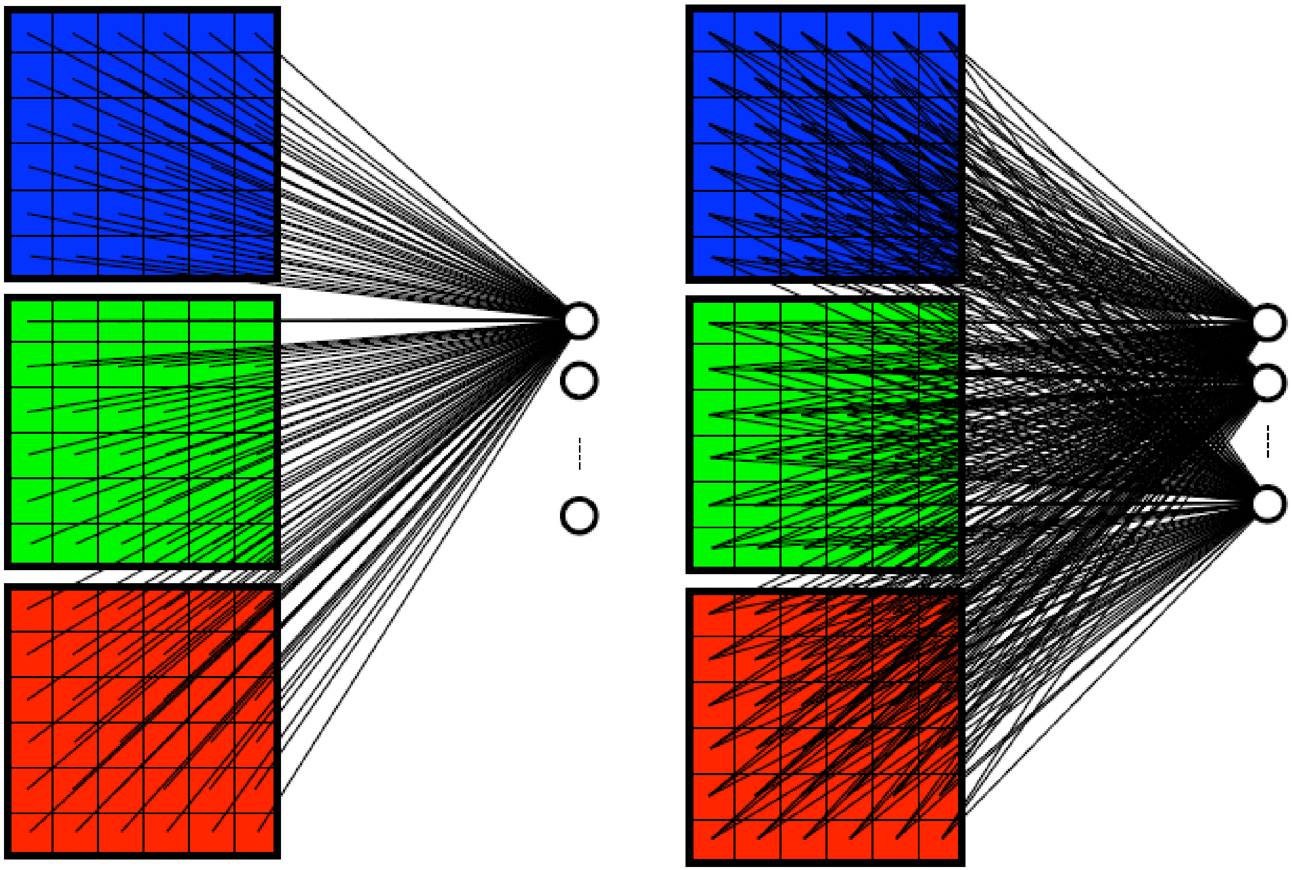
- Intro to Neural Networks & CNN
  - Linear Classification
  - Neural Network for Machine Vision
  - Multi-Layer Perceptron
  - Convolutional Neural Networks
- Image Segmentation* (if time permits)
- Object Detection* (if time permits)





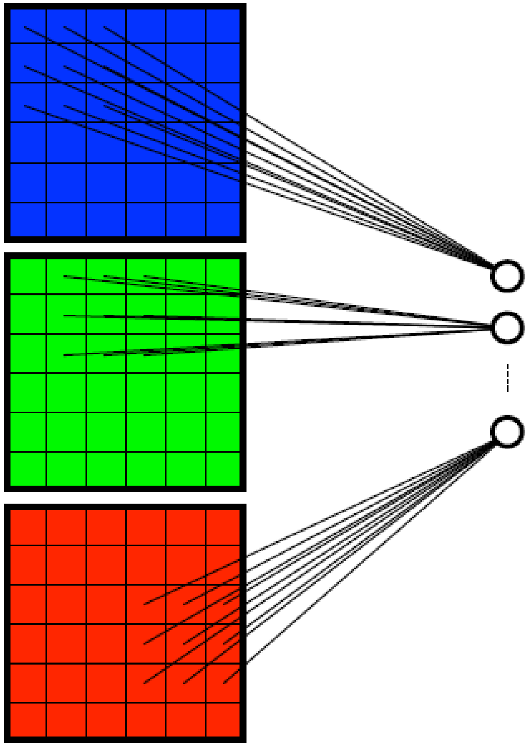
Convolutional Neural Networks

- How many weights for MLPs for images?



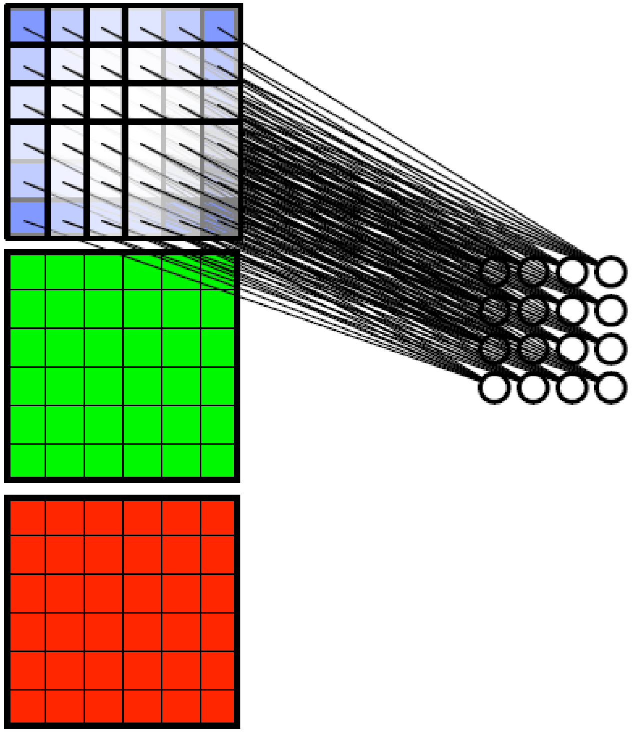
Convolutional Neural Networks

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

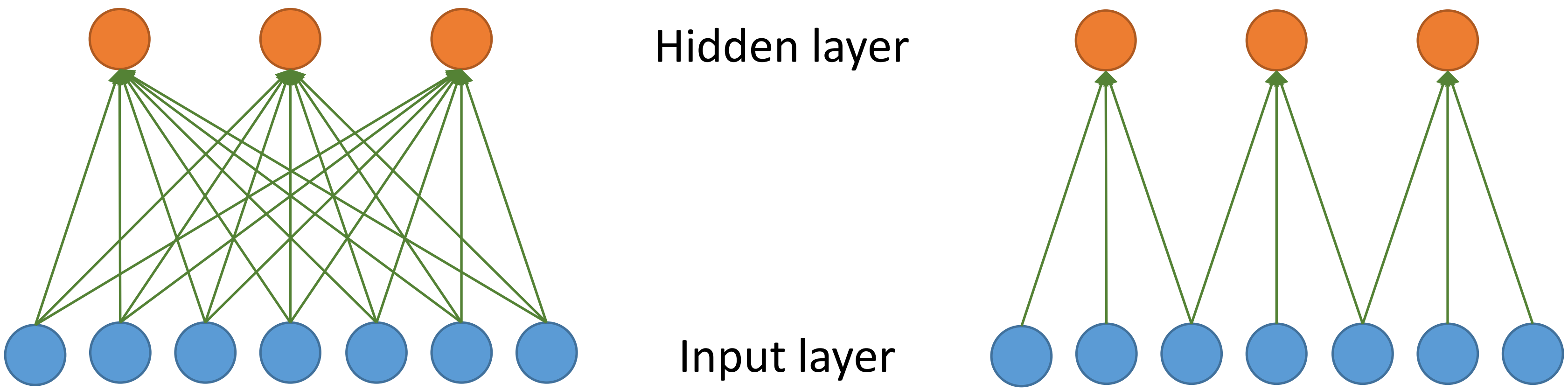


Convolutional Neural Networks

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



CNN: Local Connectivity

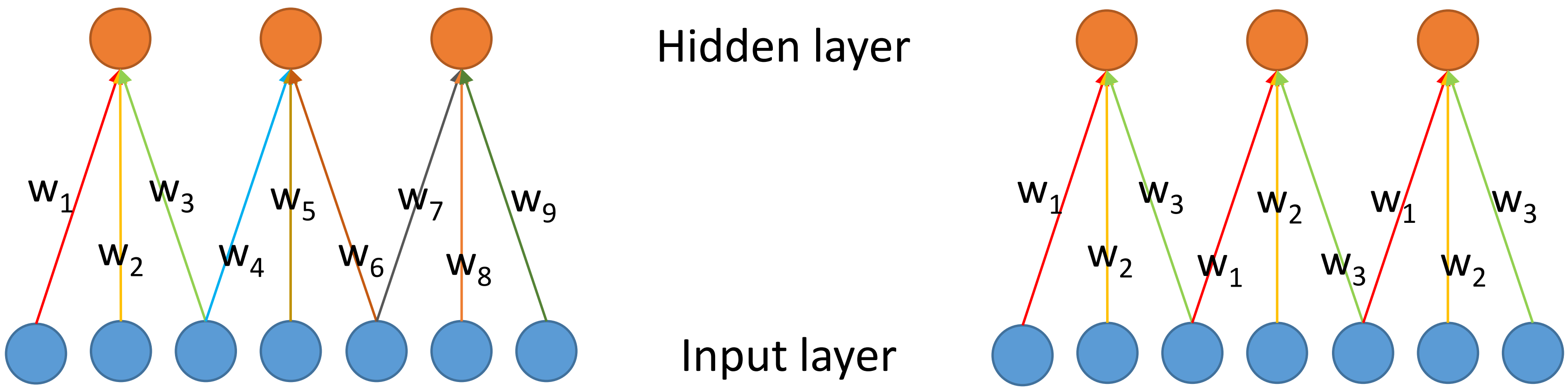


Global connectivity

Local connectivity

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

CNN: Weight Sharing

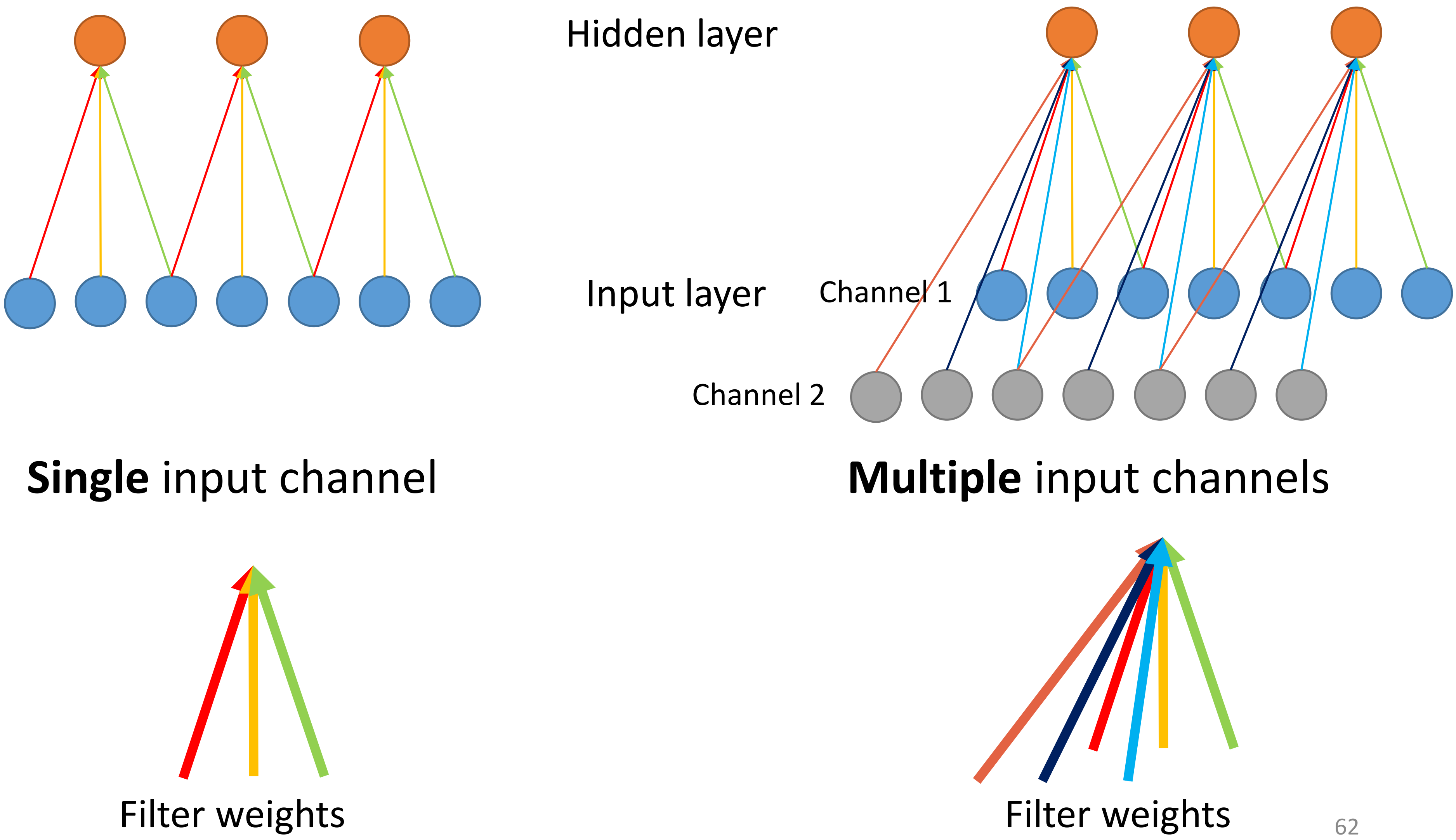


Without weight sharing

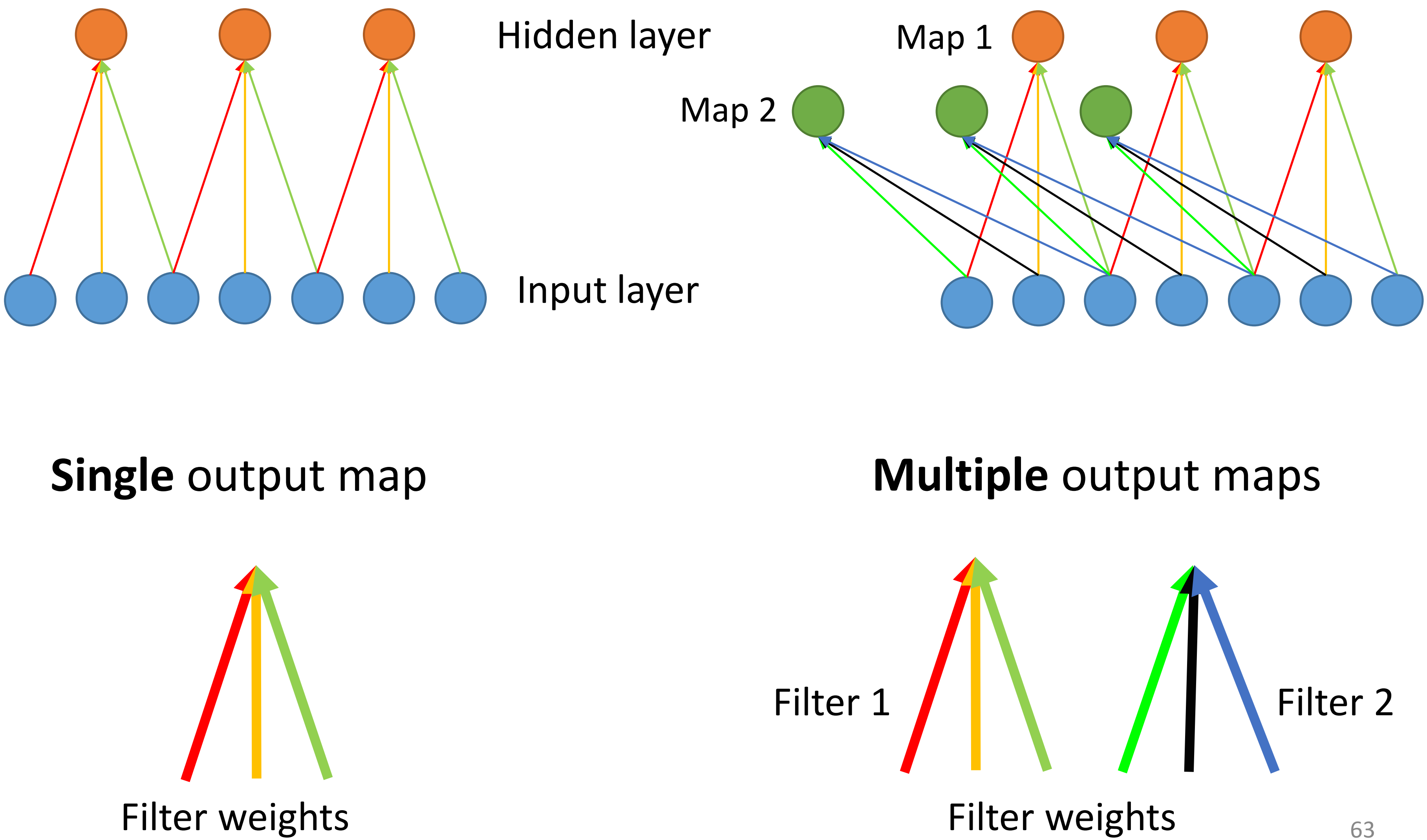
With weight sharing

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

CNN with Multiple Input Channels



CNN with Multiple Output Maps



Putting them together

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

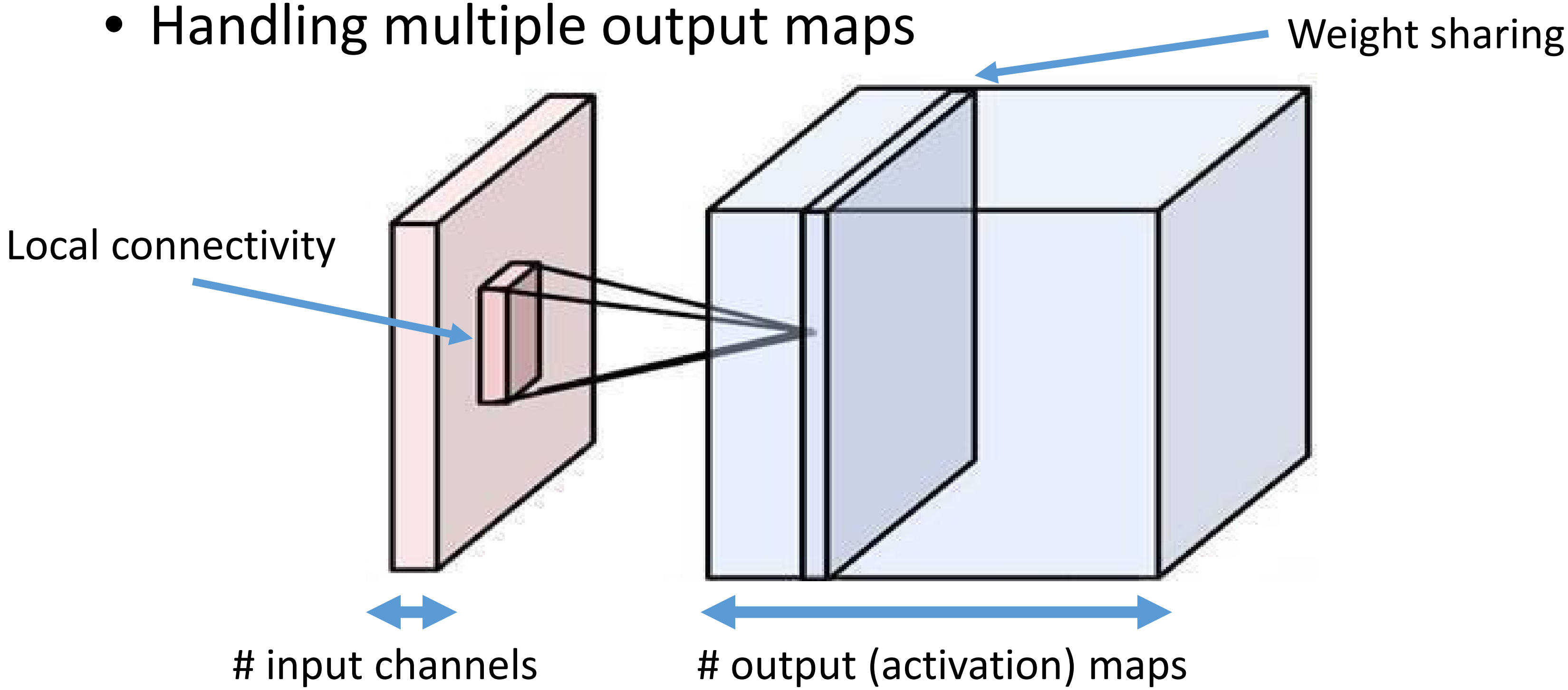
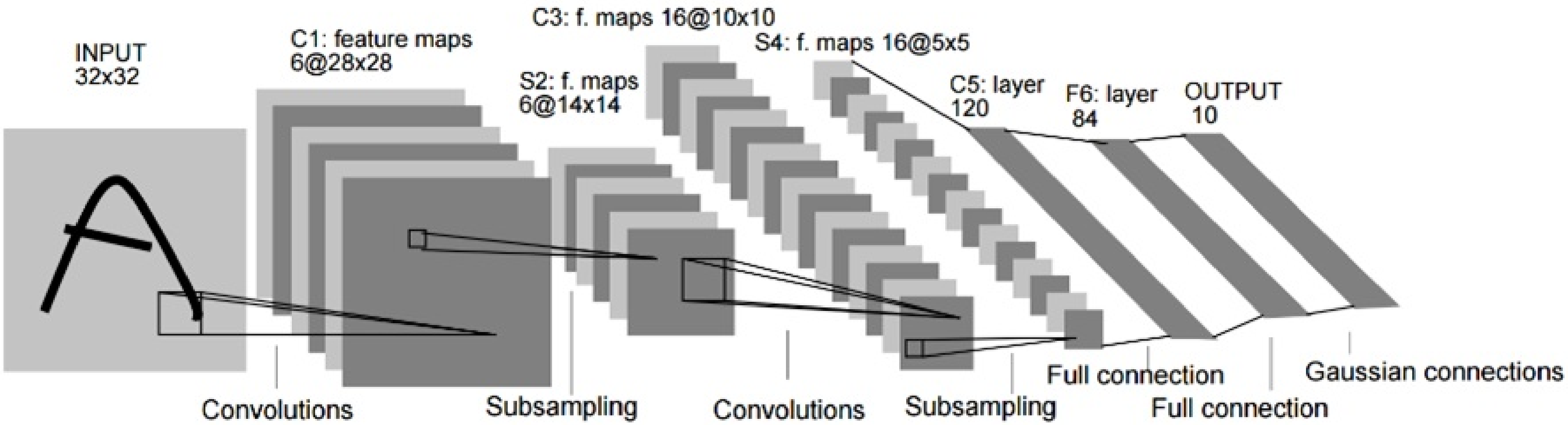
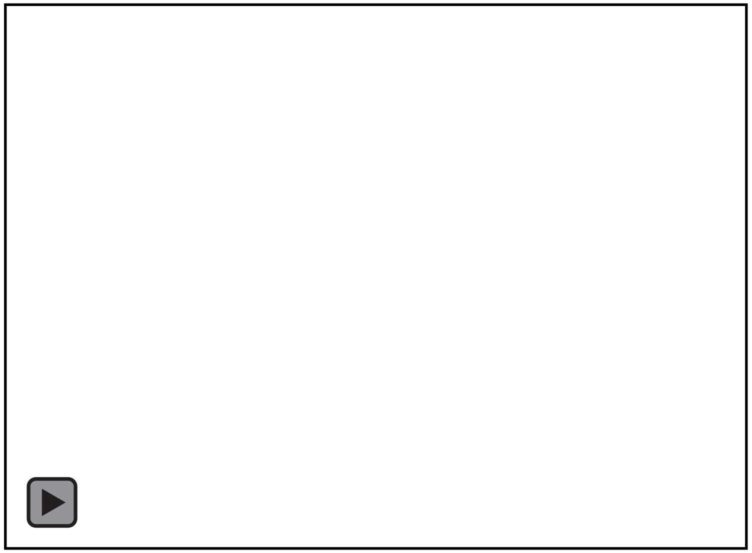


Image credit: A. Karpathy

LeNet [LeCun et al. 1998]

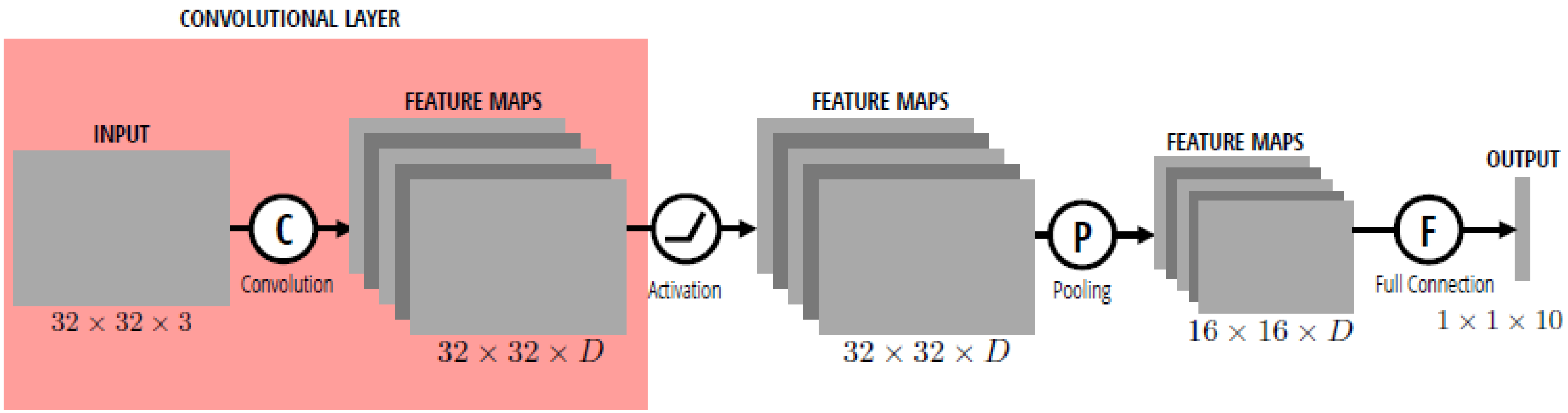


Gradient-based learning applied to document recognition
[LeCun, Bottou, Bengio, Haffner 1998]



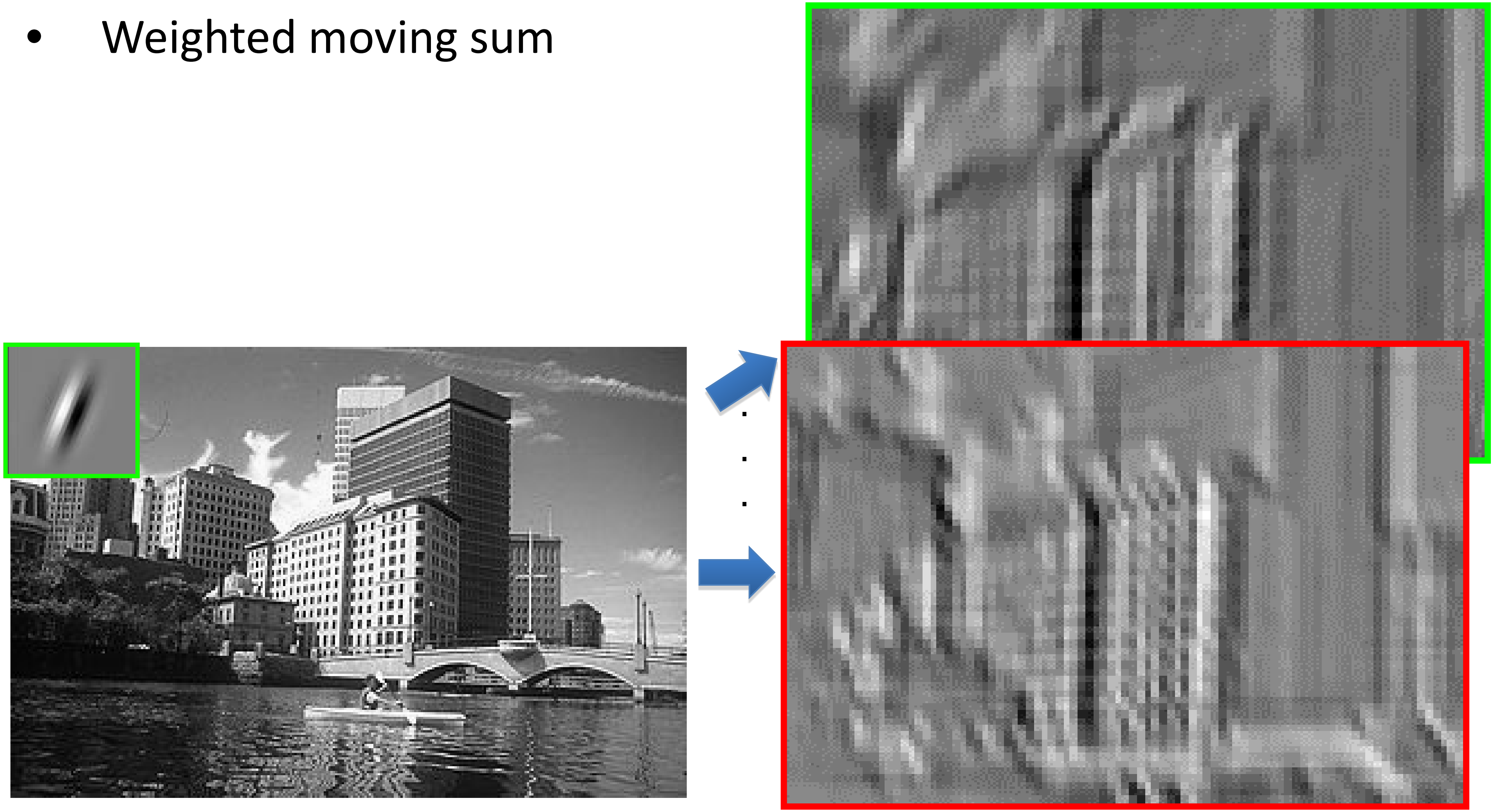
LeNet-1 from 1993

Convolution Layer in CNN



What is a Convolution?

- Weighted moving sum

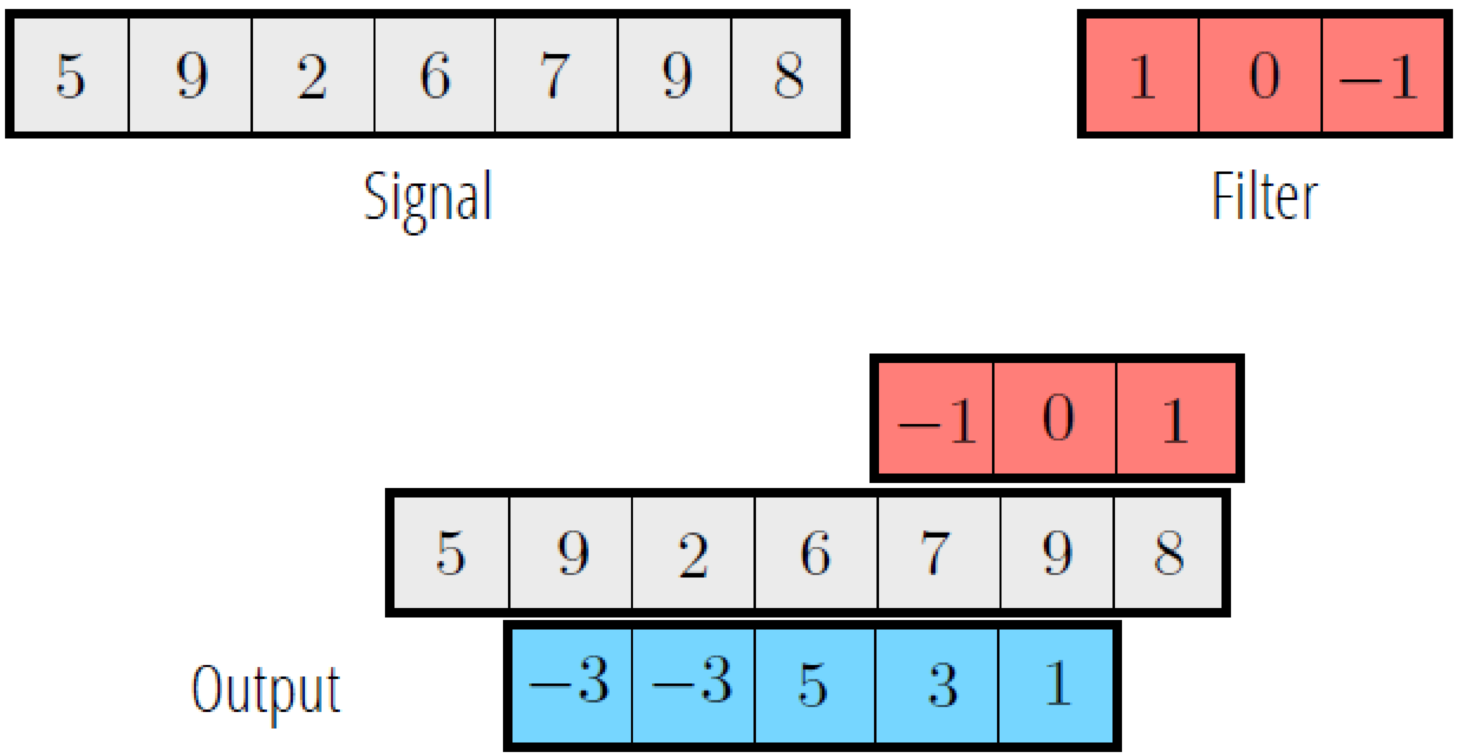


Input

Feature Activation Map

slide credit: S. Lazebnik

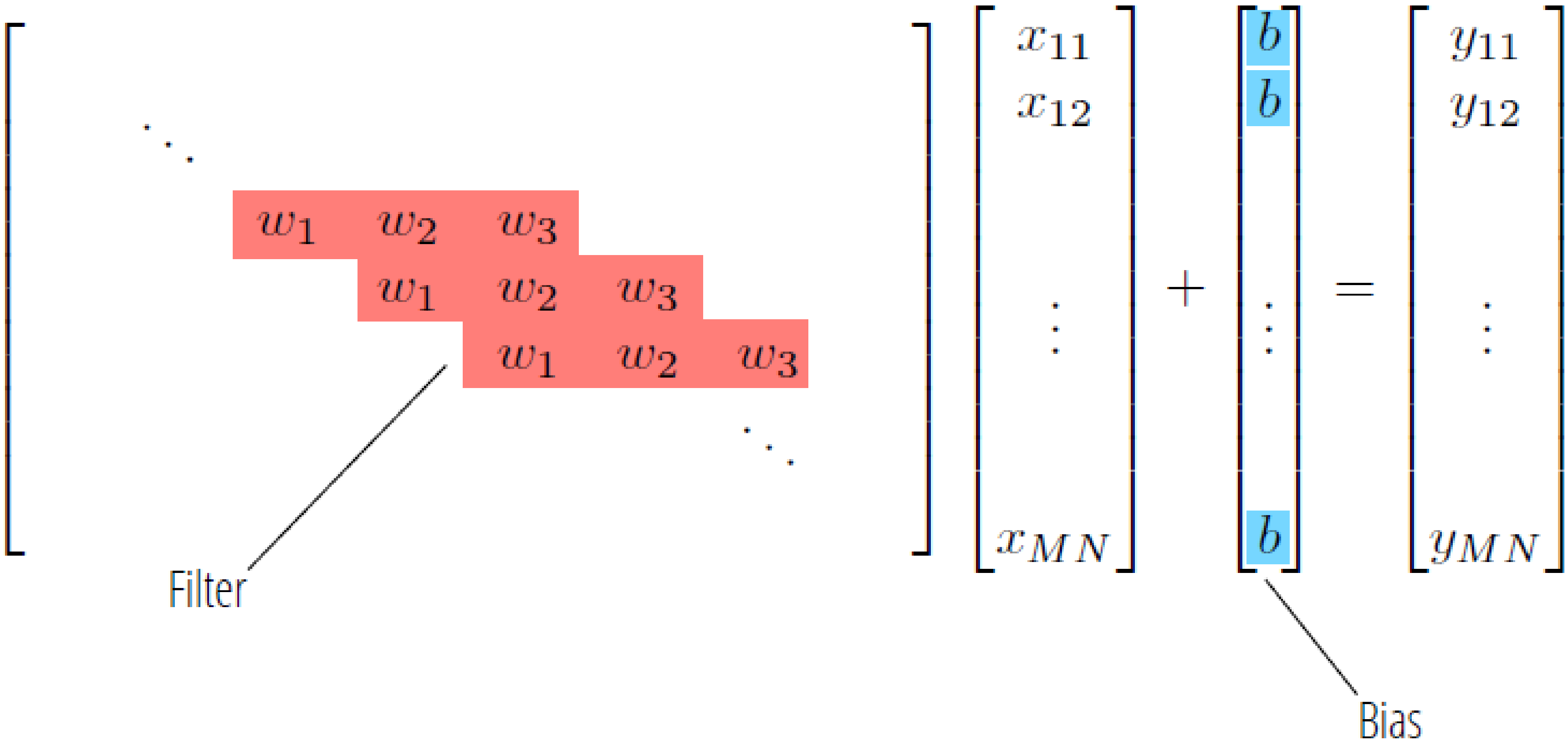
What is a Convolution?



Convolution is a local linear operator

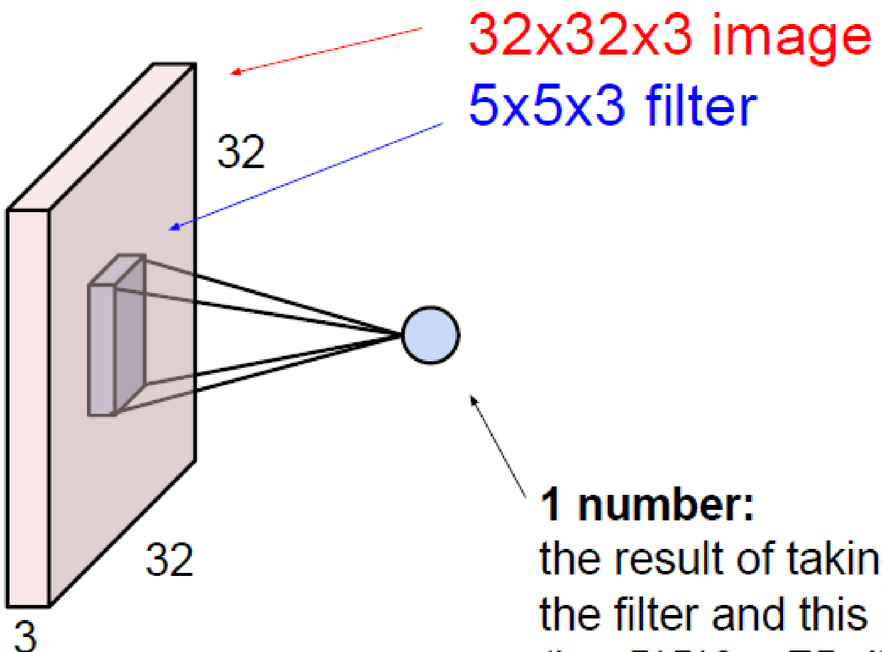
What is a Convolution?

- Toeplitz Matrix Form

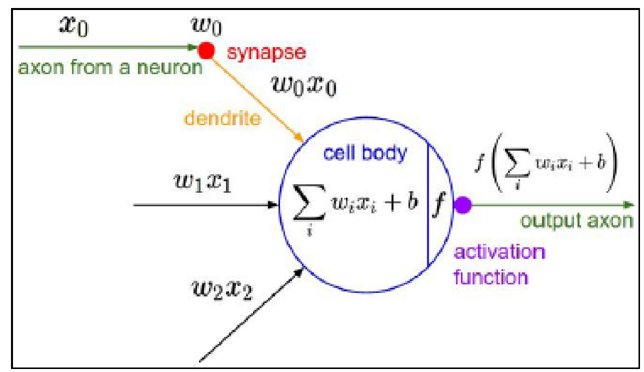


Putting them together (cont'd)

- The brain/neuron view of CONV layer



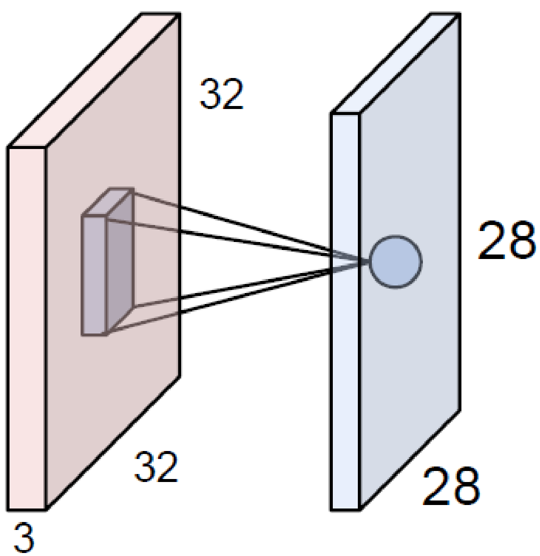
1 number:
the result of taking a dot product between
the filter and this part of the image
(i.e. 5*5*3 = 75-dimensional dot product)



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

Putting them together (cont'd)

- The brain/neuron view of CONV layer



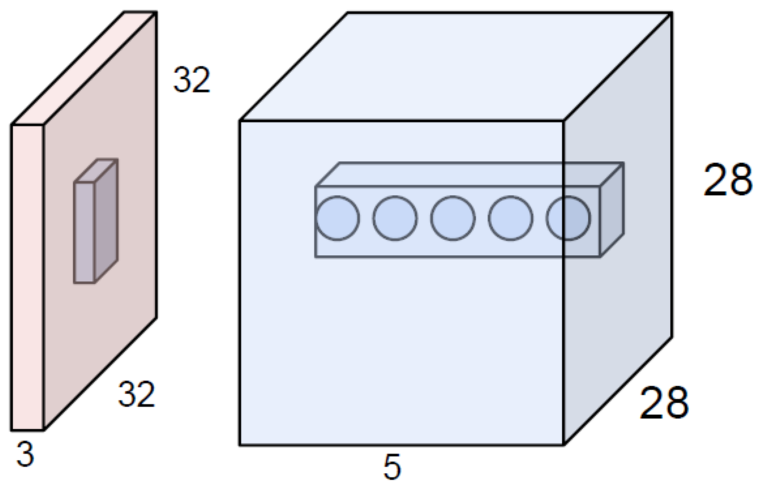
An activation map is a 28x28 sheet of neuron outputs:

1. Each is connected to a small region in the input
2. All of them share parameters

"5x5 filter" -> "5x5 receptive field for each neuron"

Putting them together (cont'd)

- The brain/neuron view of CONV layer

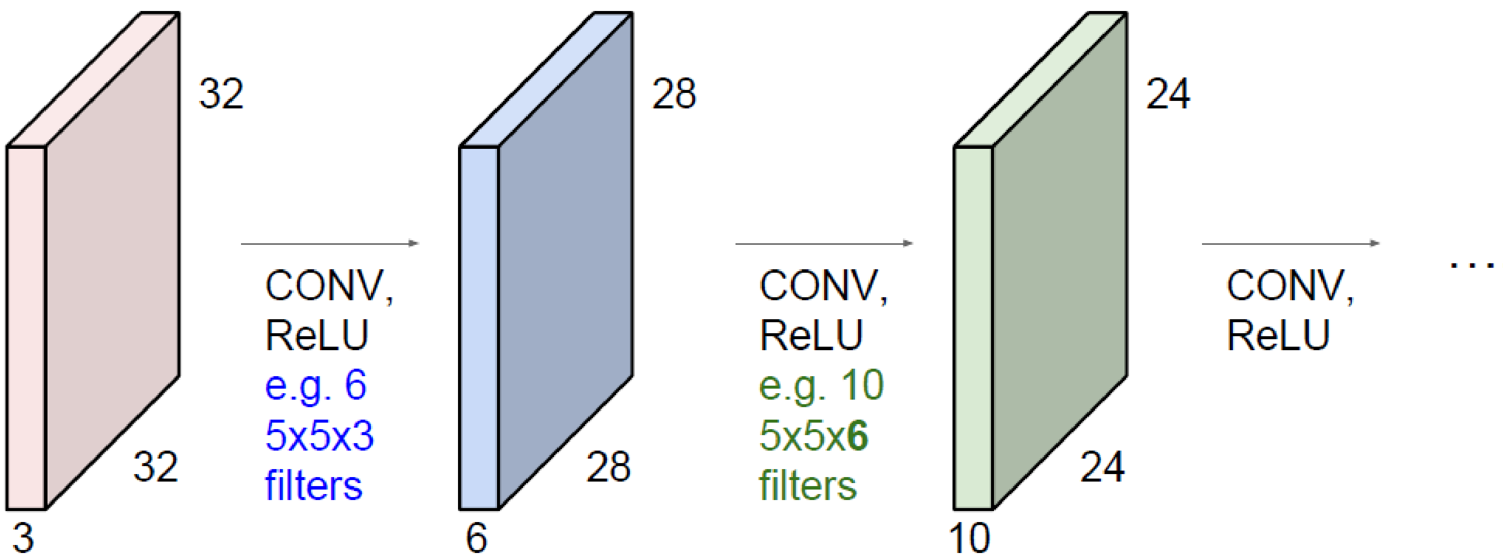


E.g. with 5 filters,
CONV layer consists of
neurons arranged in a 3D grid
(28x28x5)

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

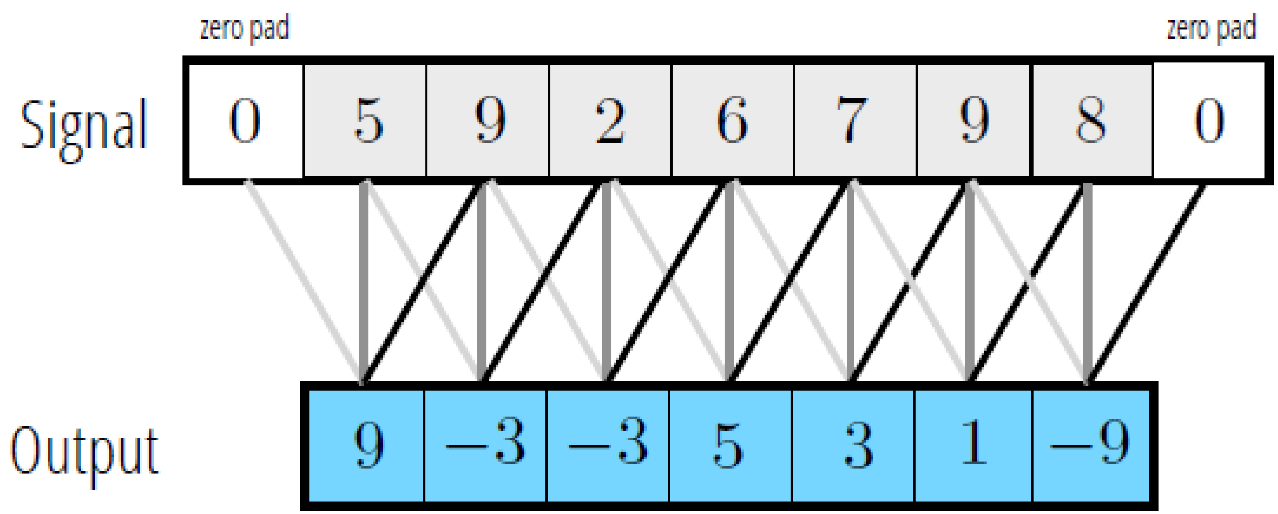
Putting them together (cont'd)

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



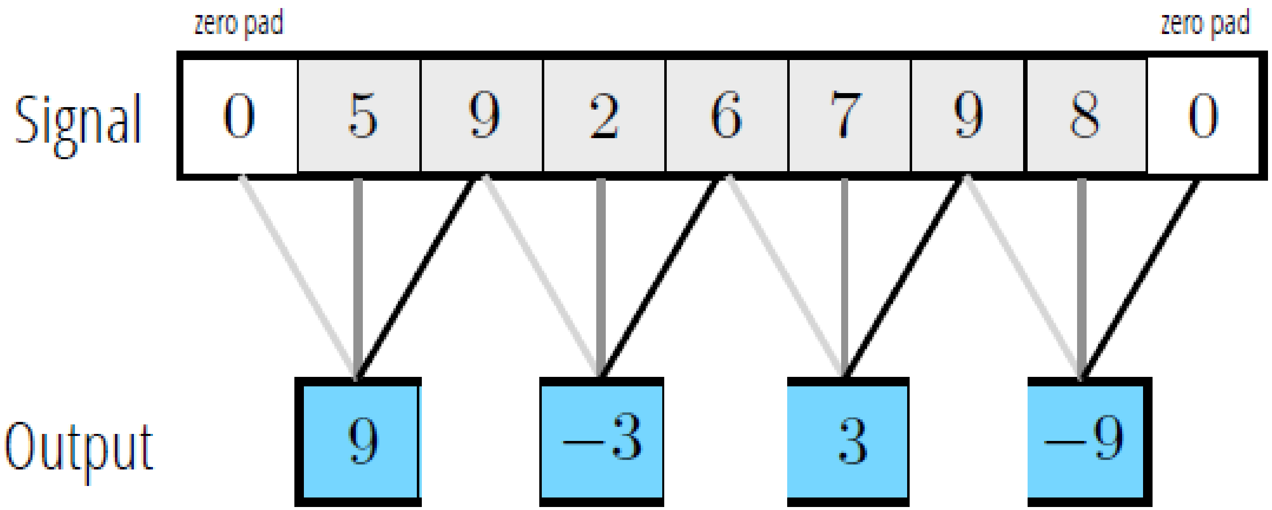
What is a Convolution?

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



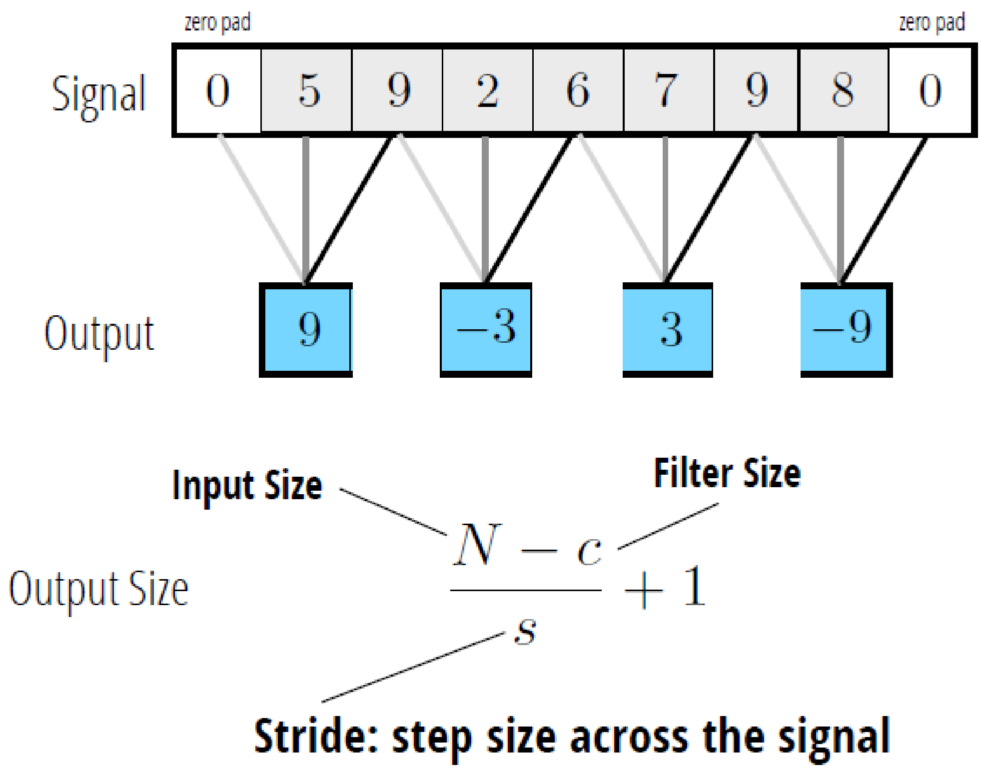
What is a Convolution?

- Stride
  - Step size across signals



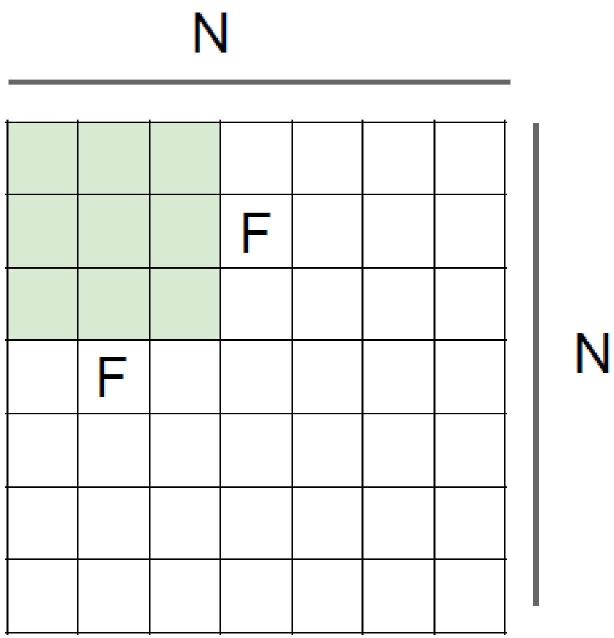
What is a Convolution?

- Stride
  - Step size across signals



What is a Convolution?

- Stride
  - Step size across signals

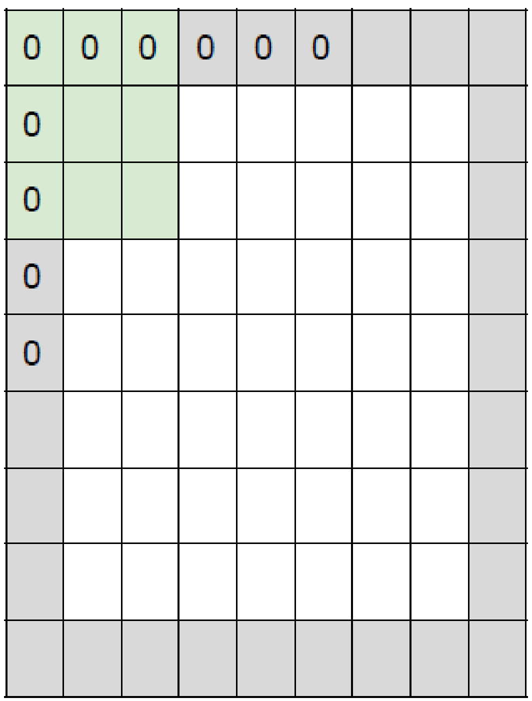


Output size:
(N - F) / stride + 1

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?

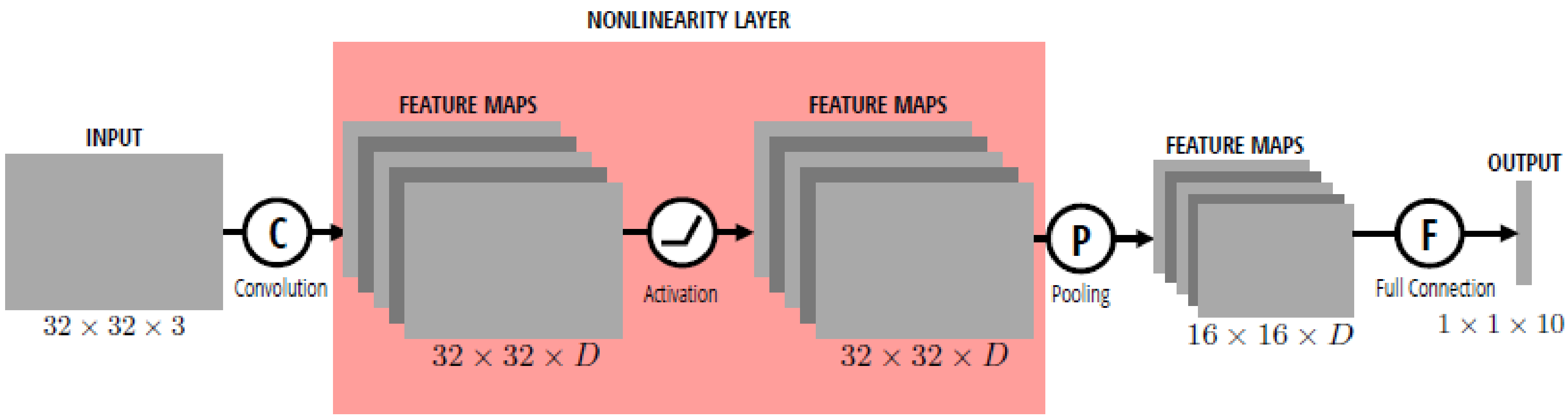
- Zero Padding + Stride



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

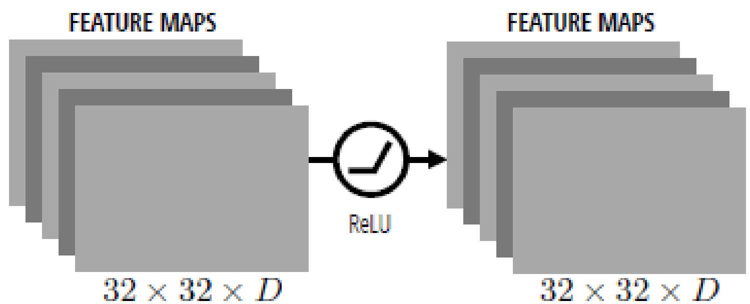
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

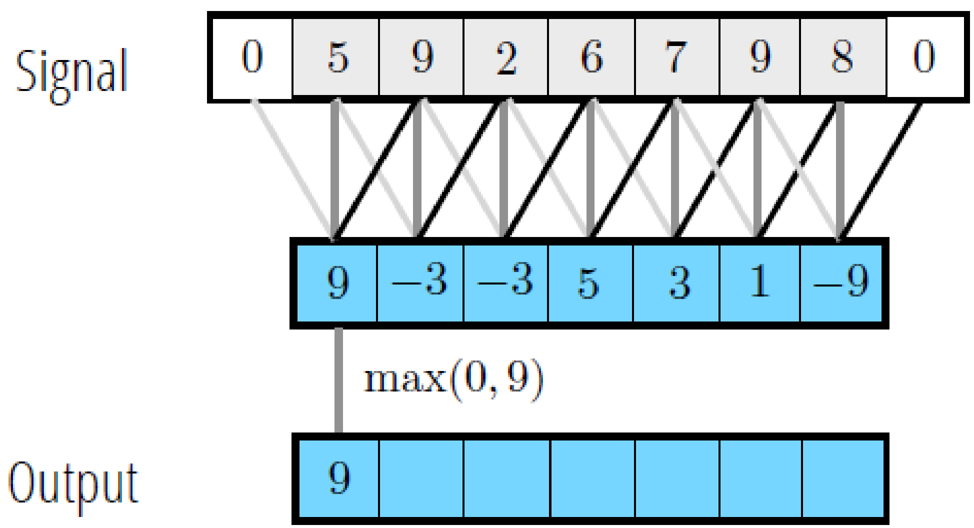
Nonlinearity Layer in CNN



Nonlinearity Layer

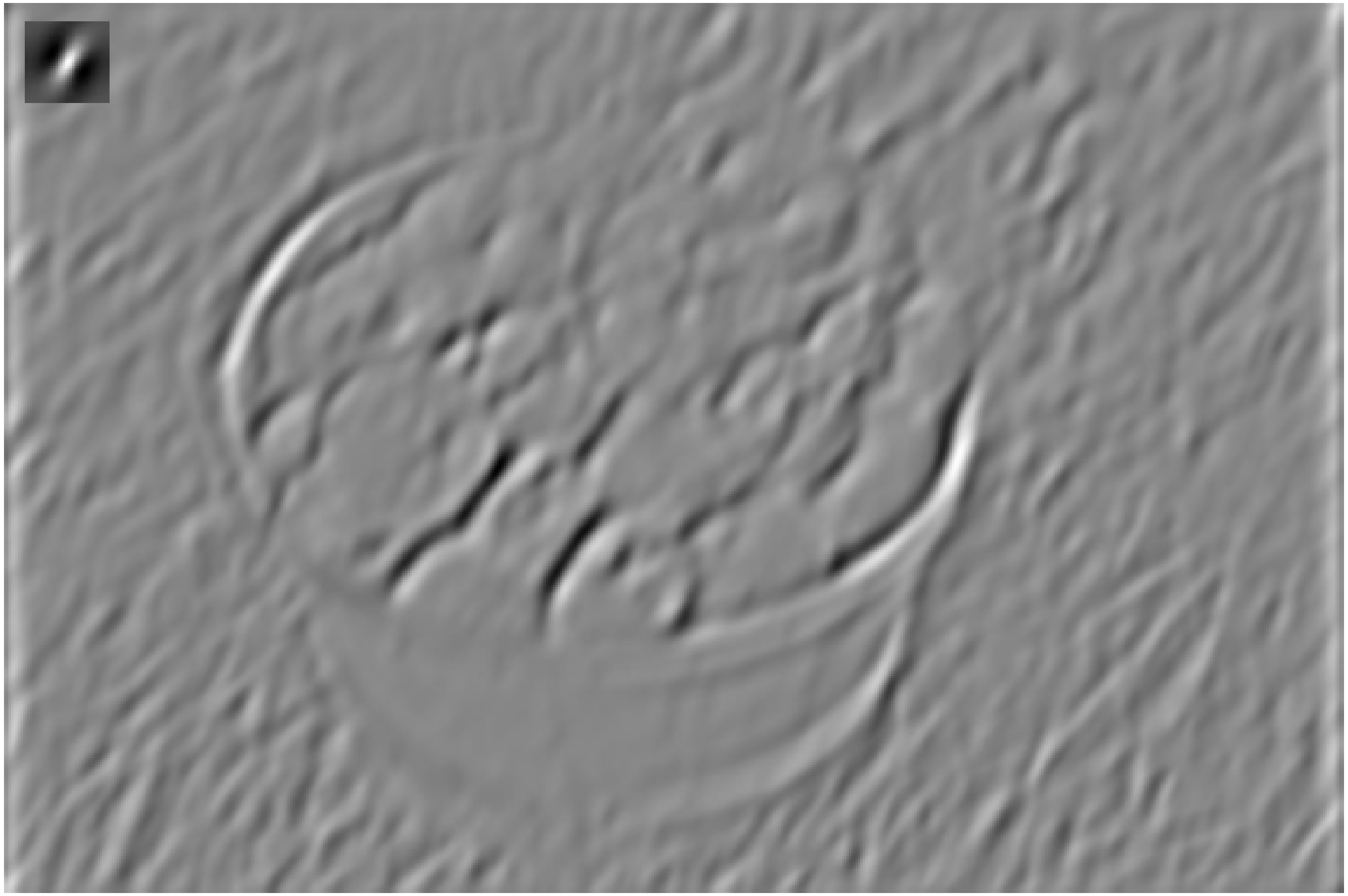
- E.g., ReLU (Rectified Linear Unit)
  - Pixel by pixel computation of max(0, x)





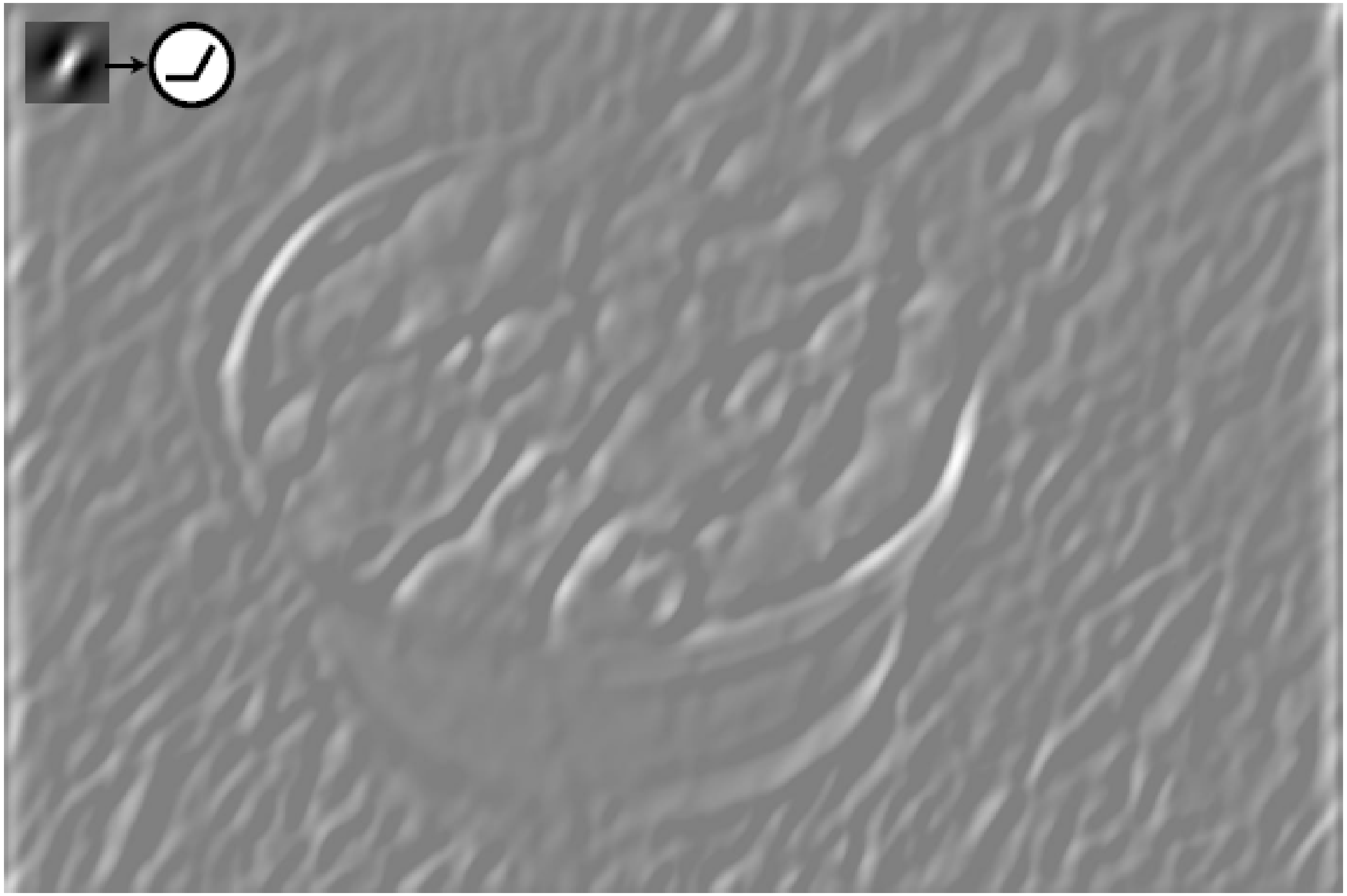
Nonlinearity Layer

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

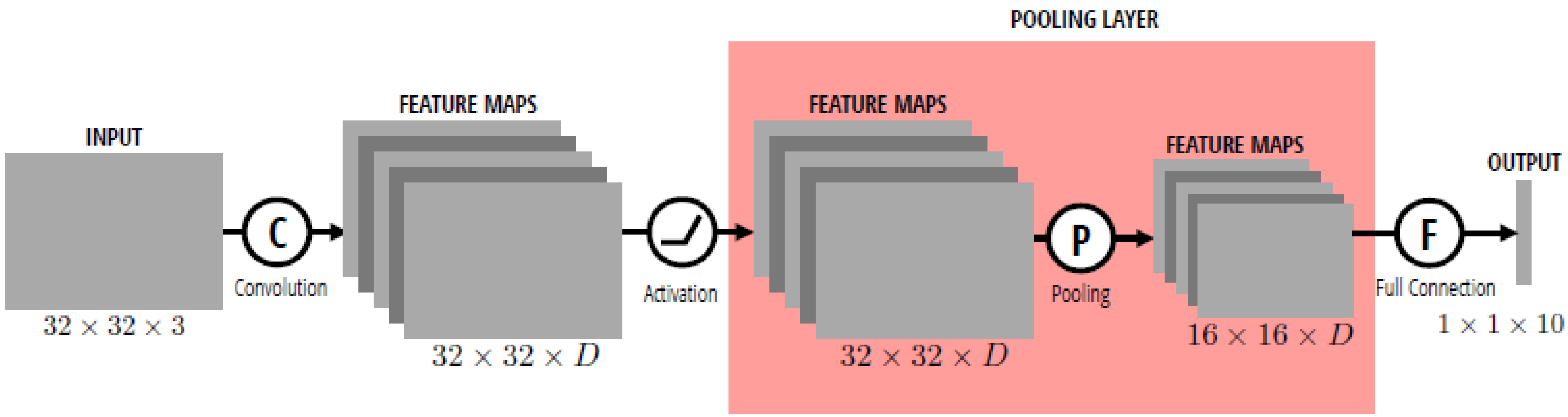


Nonlinearity Layer

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

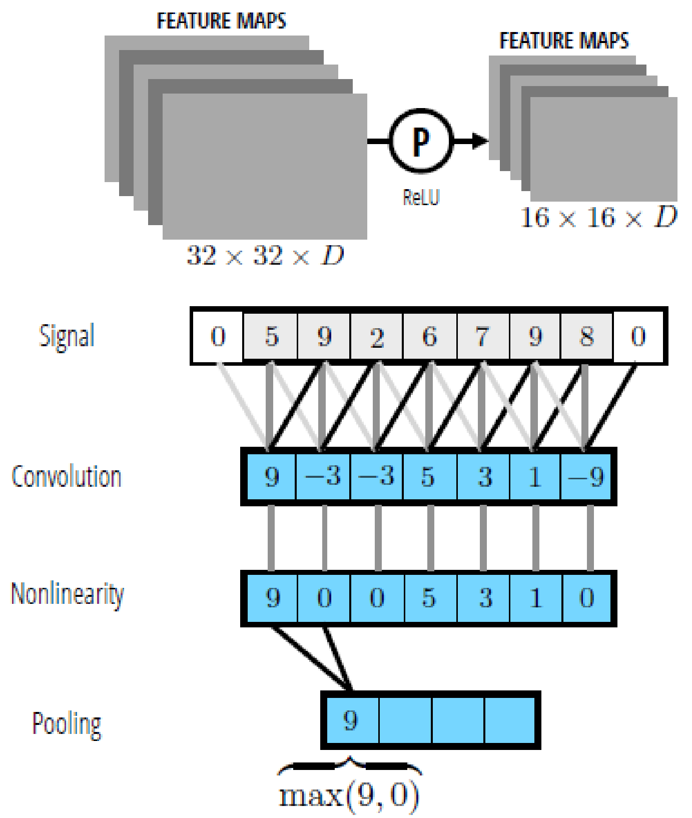


Pooling Layer in CNN



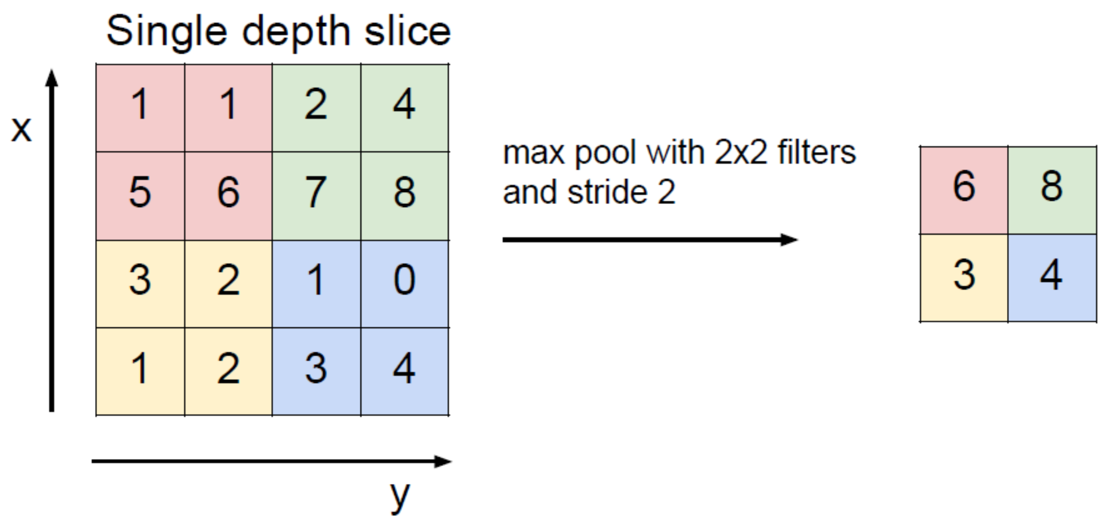
Pooling Layer

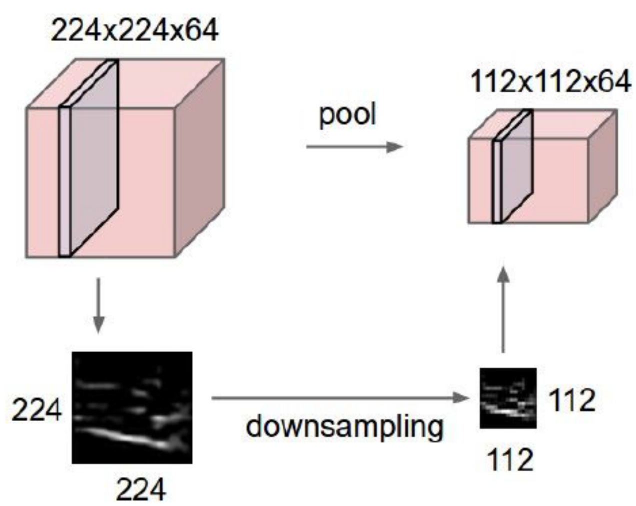
- Makes the representations smaller and more manageable
- Operates over each activation map independently
- E.g., Max Pooling



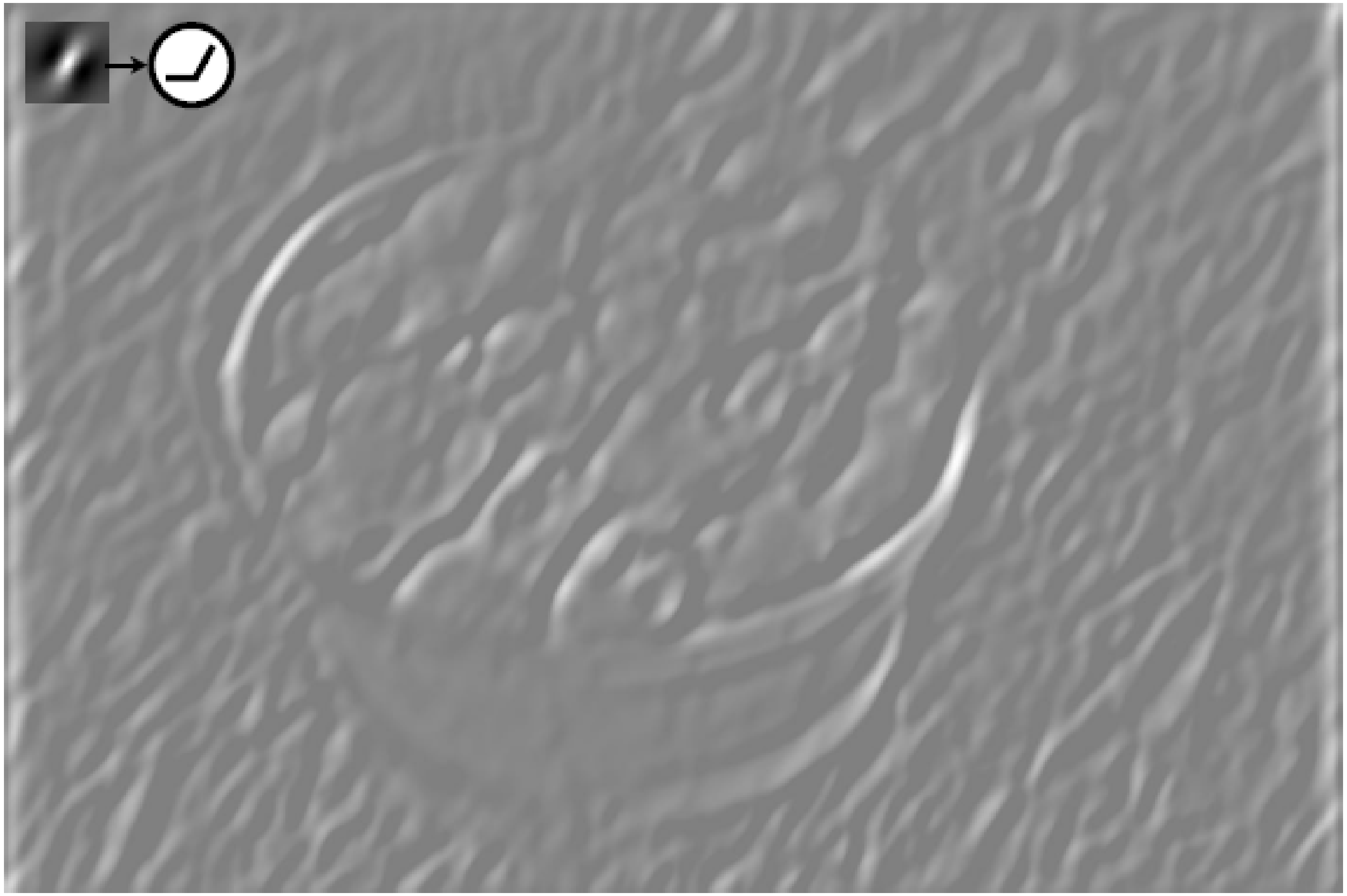
Pooling Layer

- Reduces the spatial size and provides spatial invariance

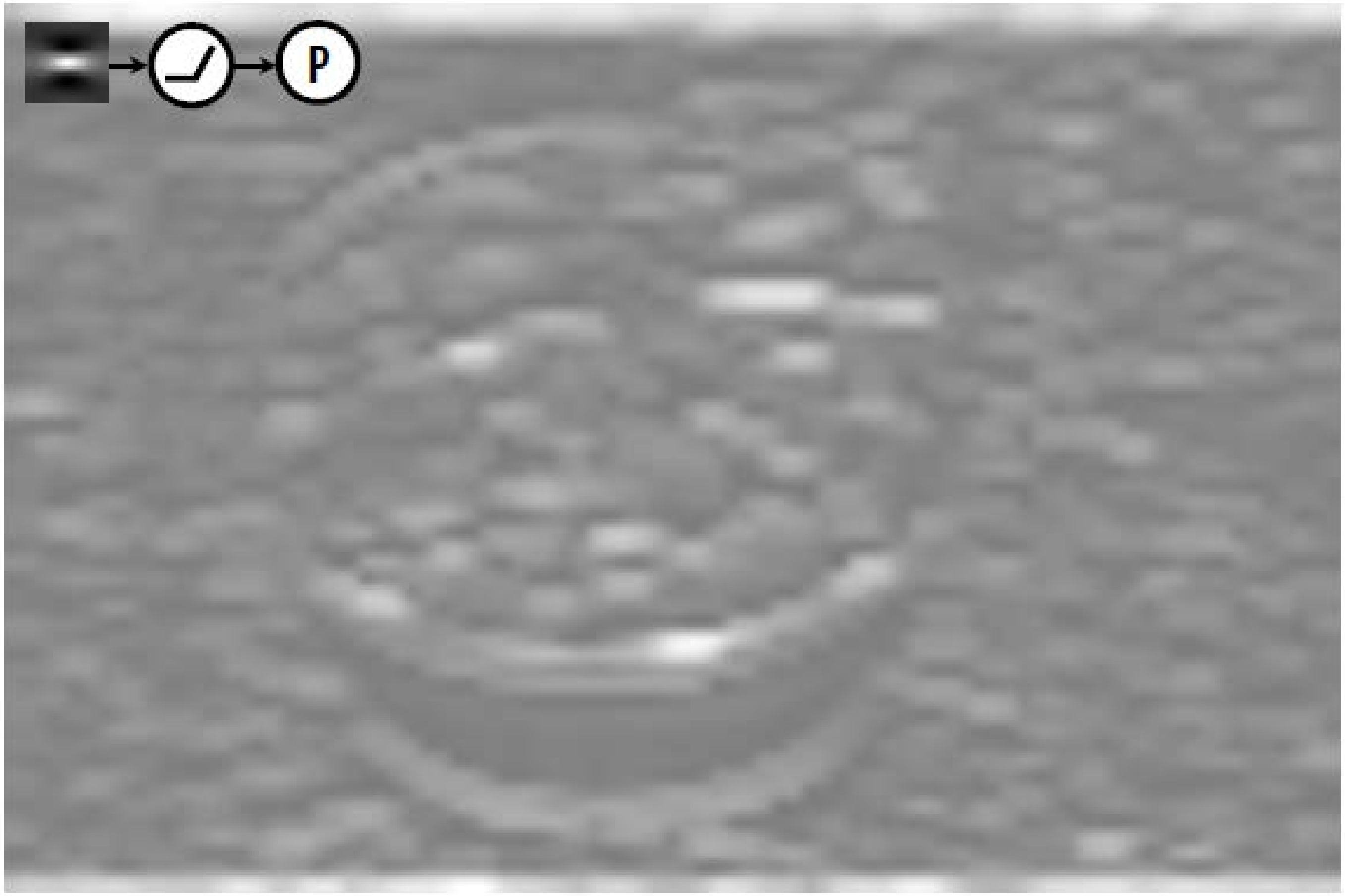




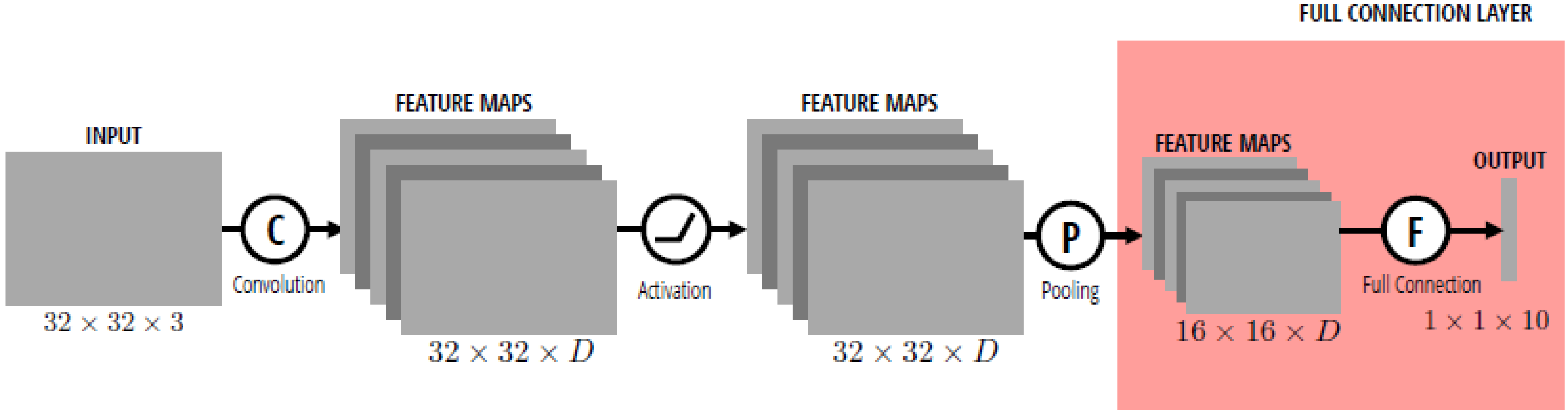
- Example
    - Nonlinearity by ReLU



- Example
  - Max pooling

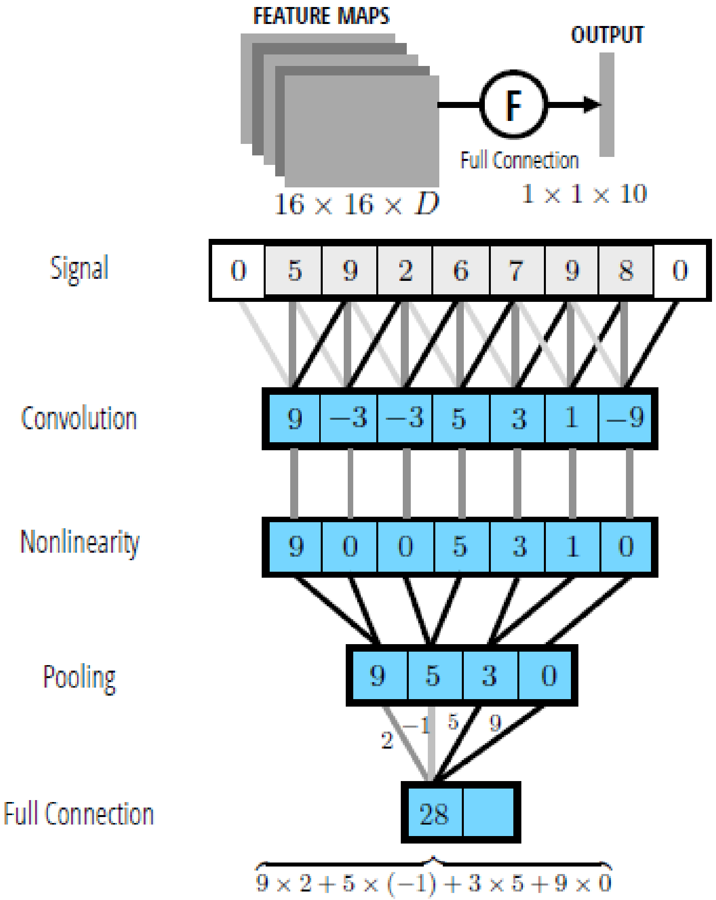


Fully Connected (FC) Layer in CNN



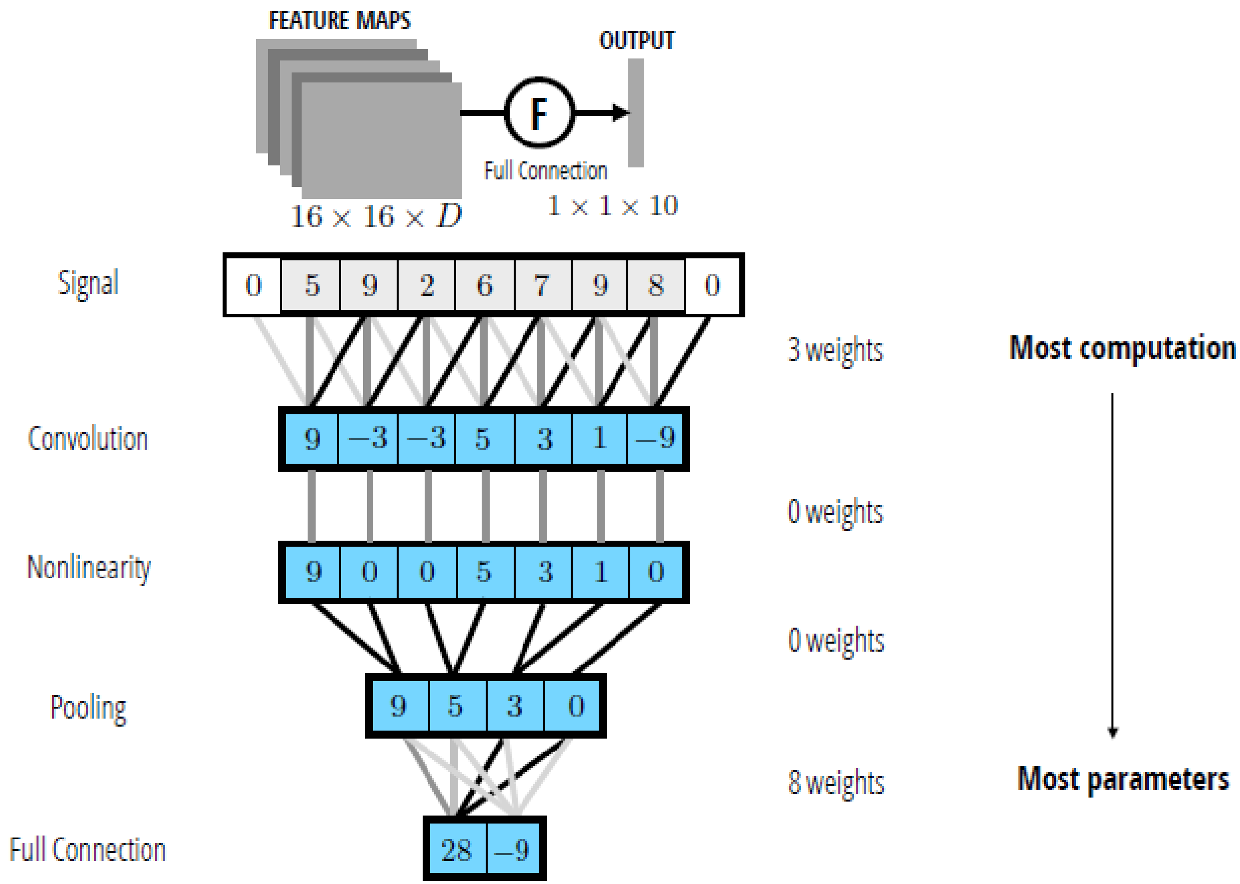
FC Layer

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

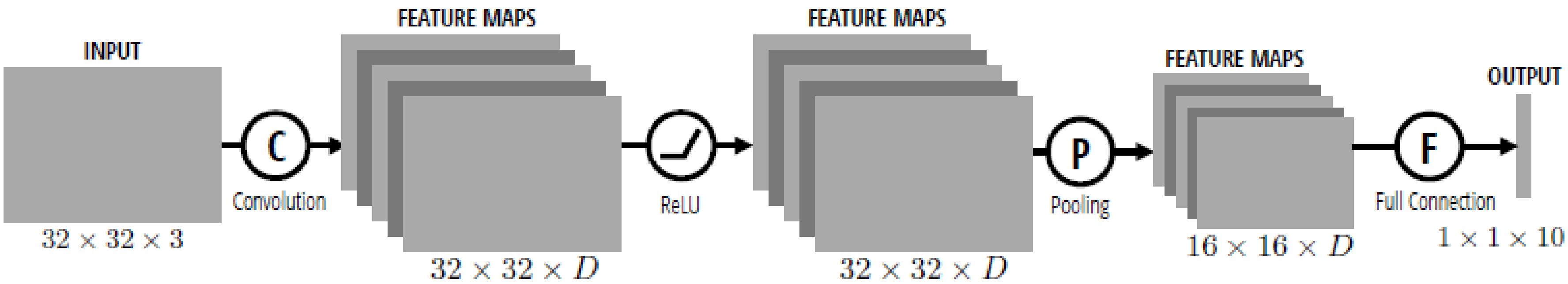


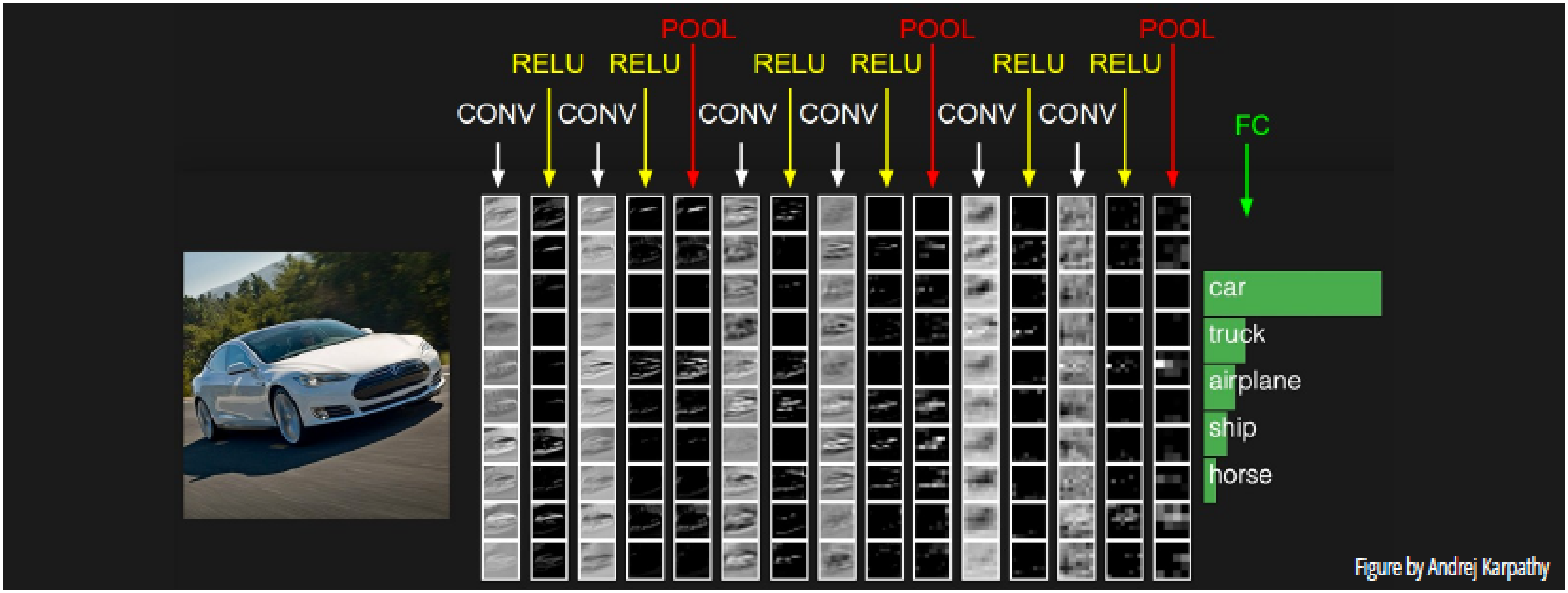
FC Layer

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



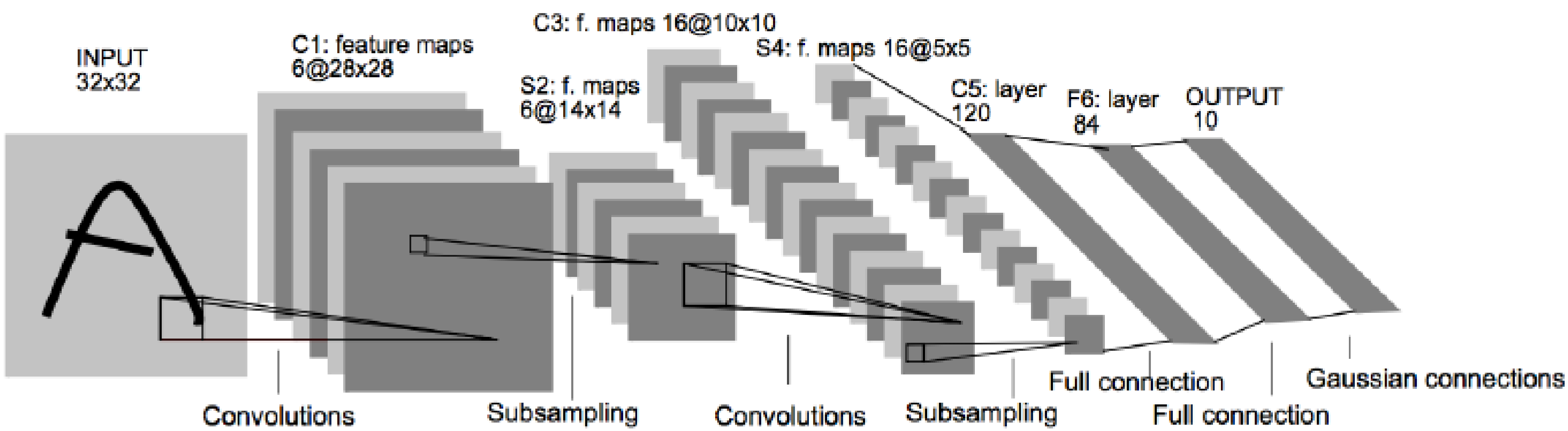
CNN


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


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

LeNet

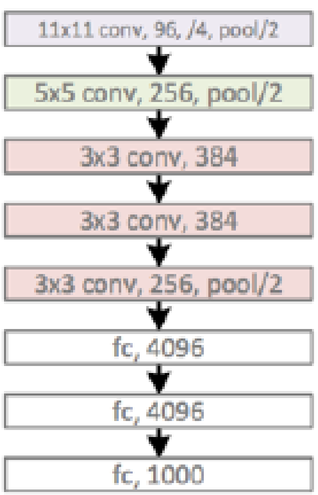
- Presented by Yann LeCun during the 1990s for reading digits
- Has the elements of modern architectures

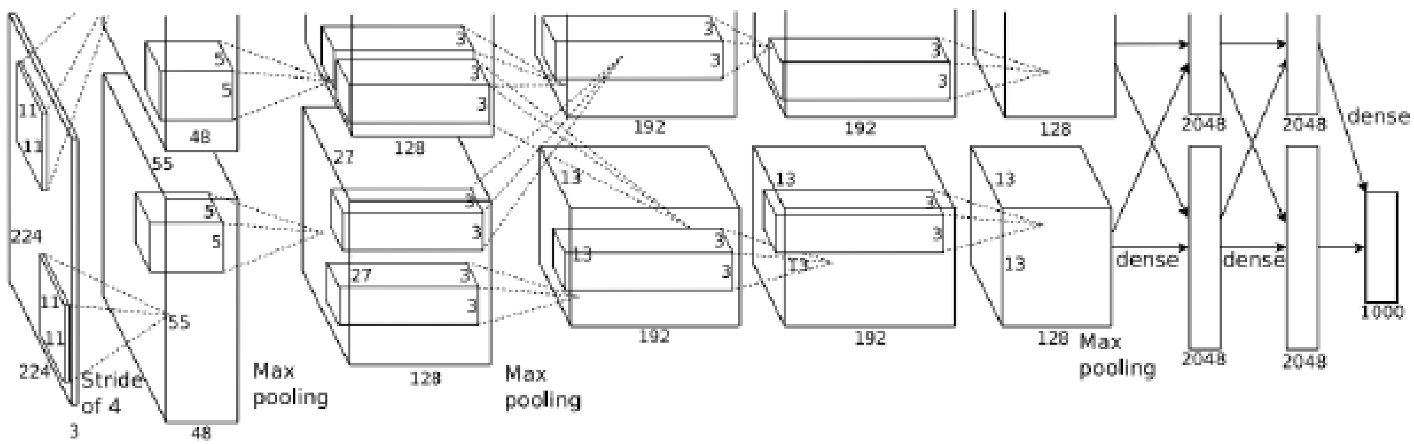


AlexNet [Krizhevsky et al., 2012]

- 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

AlexNet: 8 Layers with 18.2% top-5 error

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

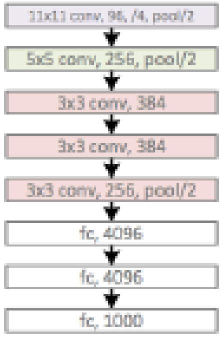
Removing Layer 6 and 7 reduces 50 million parameters, but only 5.7% drop in performance

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

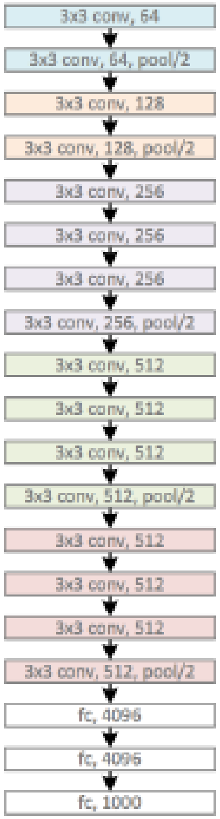
Removing feature & conv layers produces a **33% drop** in performance

CNN: A Revolution of Depth

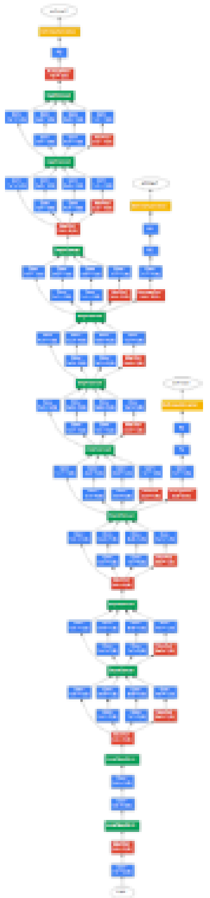
AlexNet, 8 layers
(ILSVRC 2012)



VGG, 19 layers
(ILSVRC 2014)

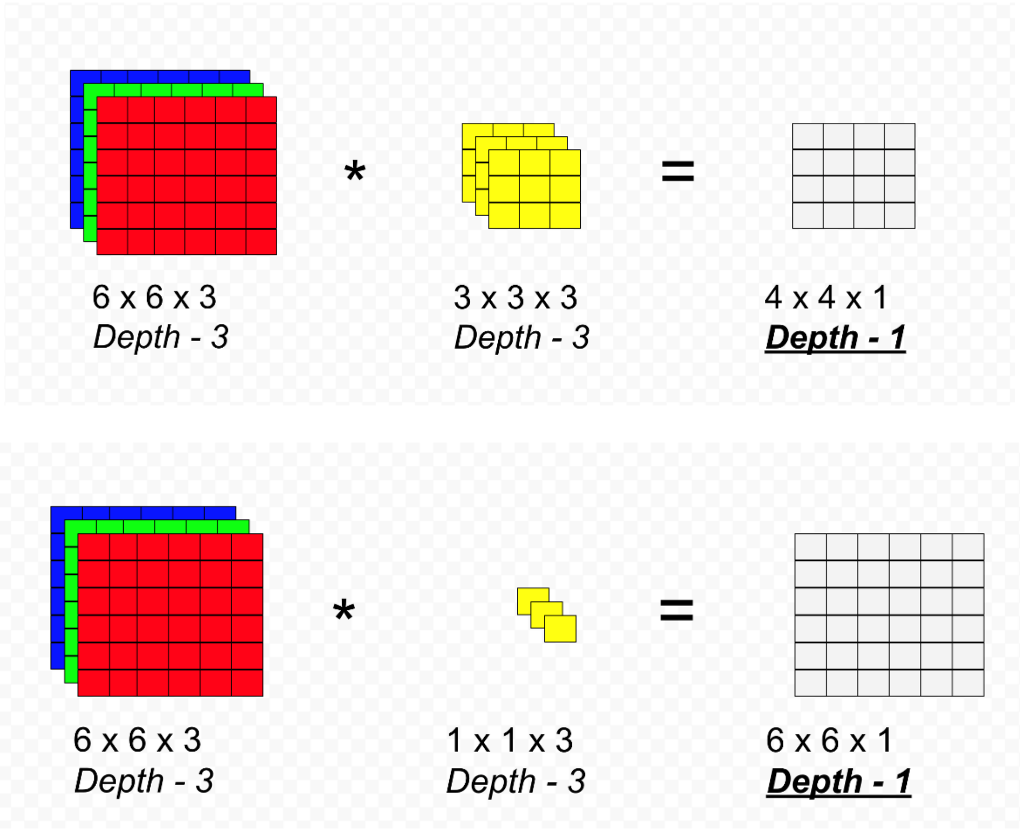


GoogleNet, 22 layers
(ILSVRC 2014)



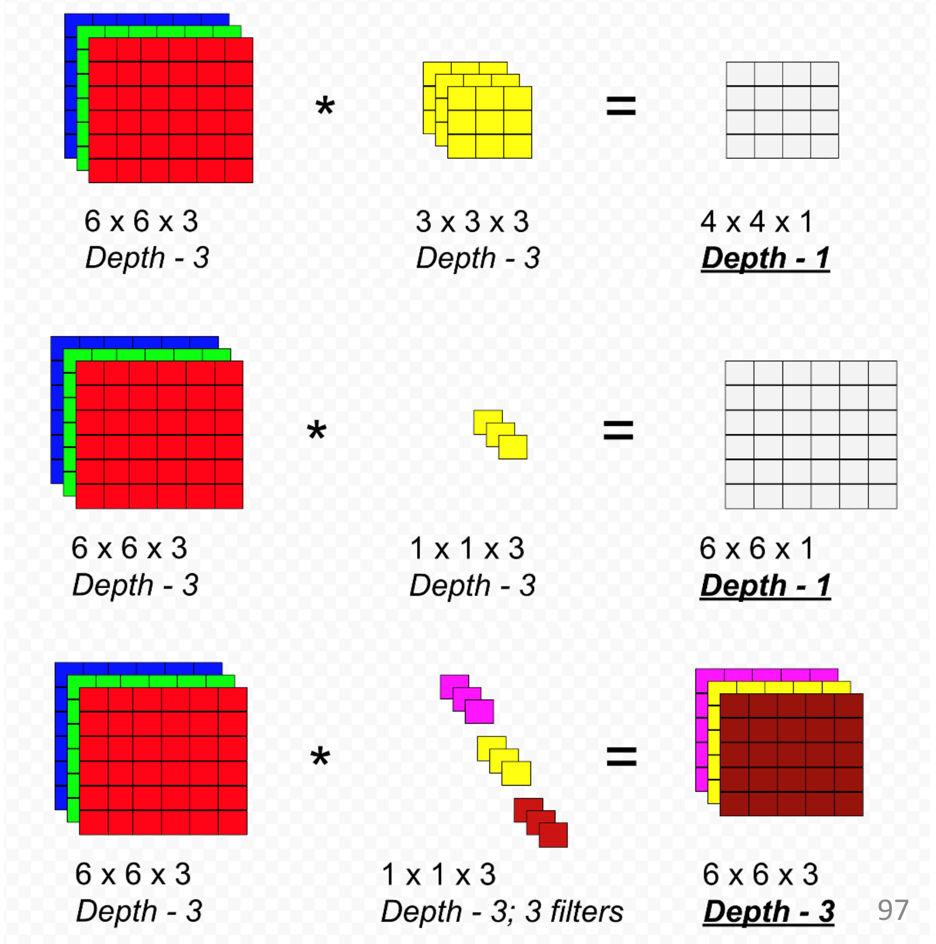
What is 1x1 Convolution?

- Doesn’t 1x1 convolution sound redundant?



What is 1x1 Convolution? (cont'd)

- Doesn't 1x1 convolution sound redundant?
- Simply speaking, it provides…
  - Dimension reduction
  - Additional nonlinearity



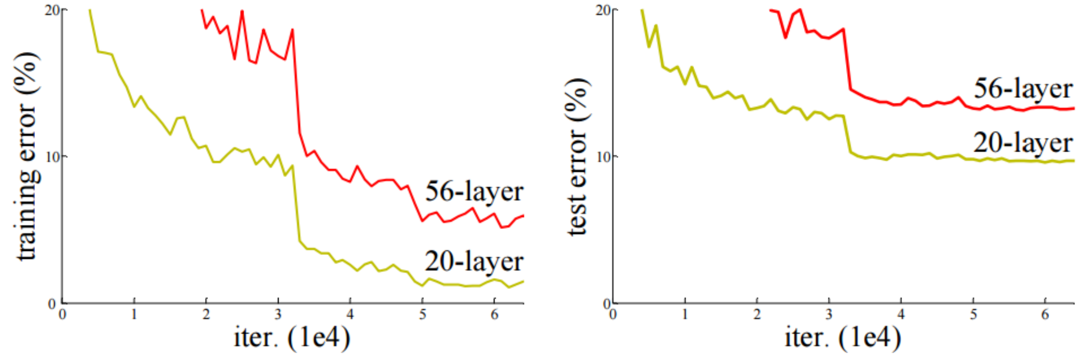
What is 1x1 Convolution? (cont'd)

- **Example 1**
  {28 x 28 x 192} convolved with 32 {5 x 5x 192} kernels into **{28 x 28 x 32}**
- (5 x 5 x 192) muls x (28 x 28) pixels x 32 kernels ~ 120M muls

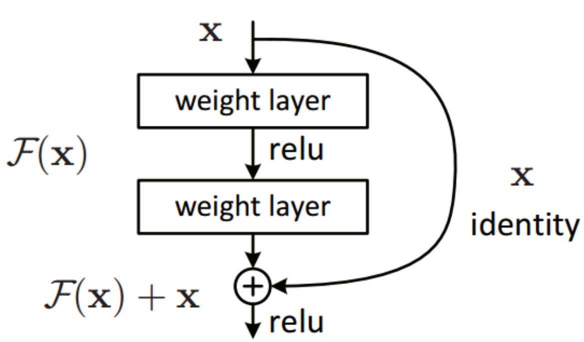
- **Example 2**
  {28 x 28 x 192} convolved with 16 {1 x 1x 192} kernels into
  {28 x 28 x 16}, followed by convolution with into 32 {5 x 5 x 16} kernels
  into **{28 x 28 x 32}**
- 192 mul x (28 x 28) pixels x 16 kernels ~ 2.4M
- (5 x 5 x 16) muls x (28 x 28) pixels x 32 kernels ~ 10M
- 12.4M vs. 120M

ResNet

- Can we just increase the #layer?



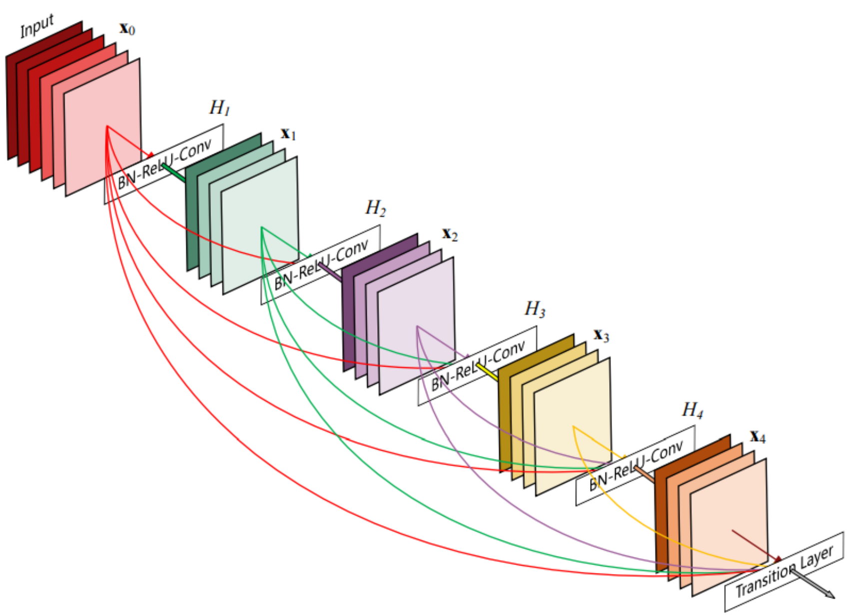
- How can we train very deep network?
  - Residual learning

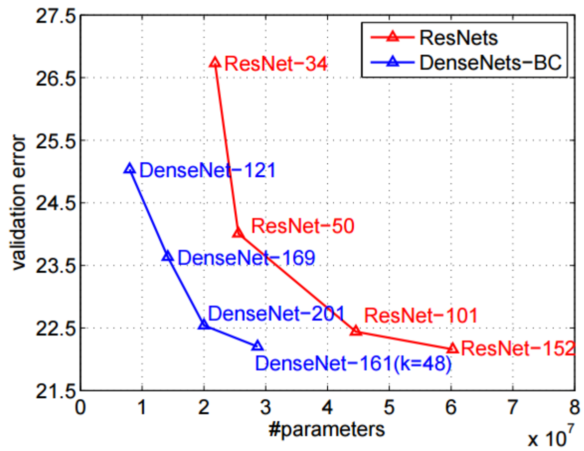


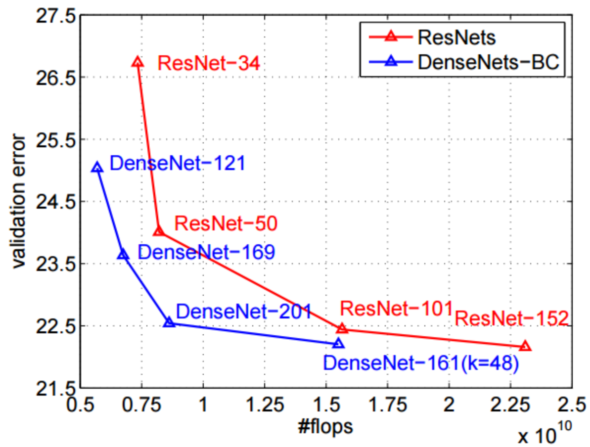
method	top-5 err. (**test**)
VGG [41] (ILSVRC'14)	7.32
GoogLeNet [44] (ILSVRC'14)	6.66
VGG [41] (v5)	6.8
PReLU-net [13]	4.94
BN-inception [16]	4.82
ResNet (ILSVRC'15)	**3.57**

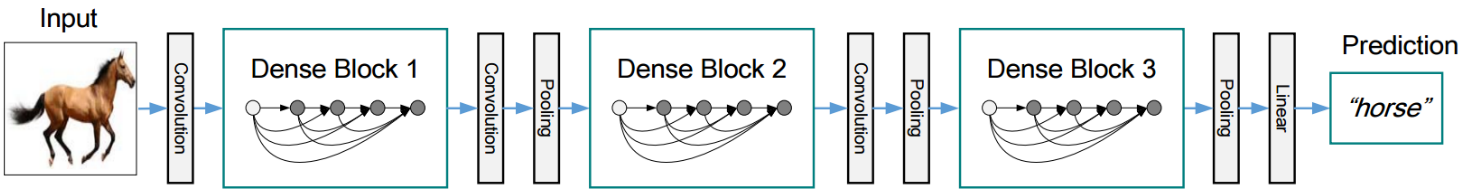
DenseNet

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



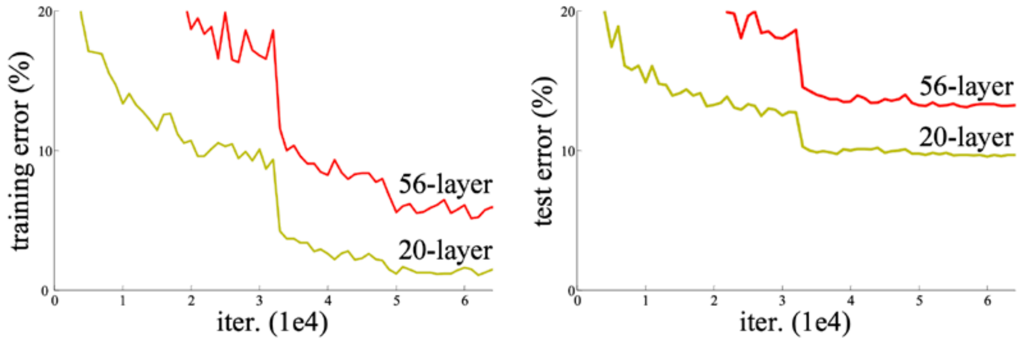




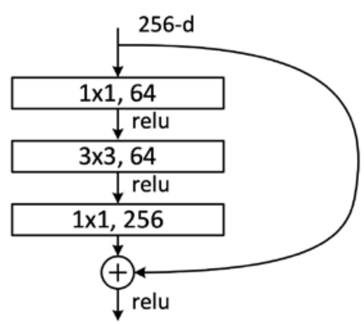


ResNet (cont'd)

- Can we just increase # of layers?



- How to train very deep networks?
  - 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**

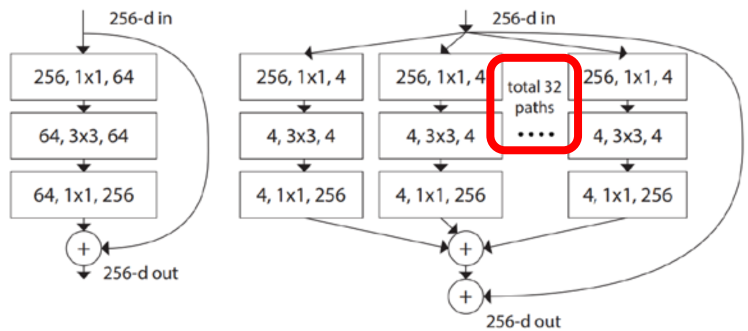
Non-Bottleneck (ResNet-18, 34)

Bottleneck (ResNet-50, 101, 152)

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

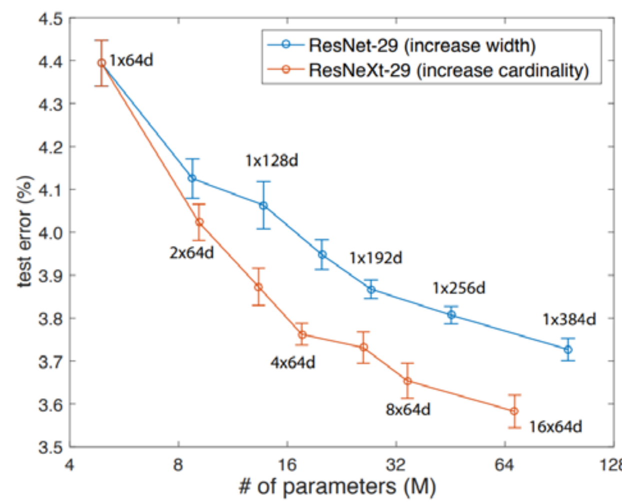
ResNeXT

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



ResNet block ResNeXt block

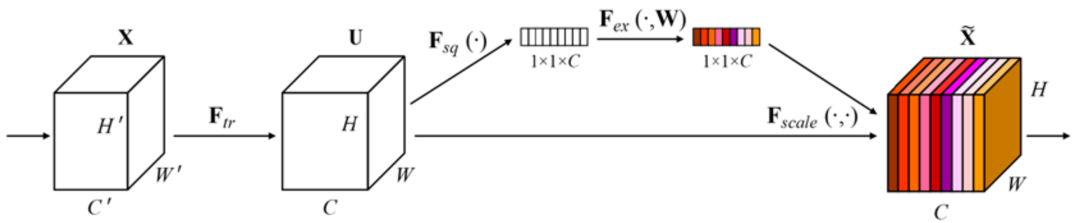
	setting	top-1 error (%)
ResNet-50	1 × 64d	23.9
ResNeXt-50	2 × 40d	23.0
ResNeXt-50	4 × 24d	22.6
ResNeXt-50	8 × 14d	22.3
ResNeXt-50	32 × 4d	**22.2**
ResNet-101	1 × 64d	22.0
ResNeXt-101	2 × 40d	21.7
ResNeXt-101	4 × 24d	21.4
ResNeXt-101	8 × 14d	21.3
ResNeXt-101	32 × 4d	**21.2**

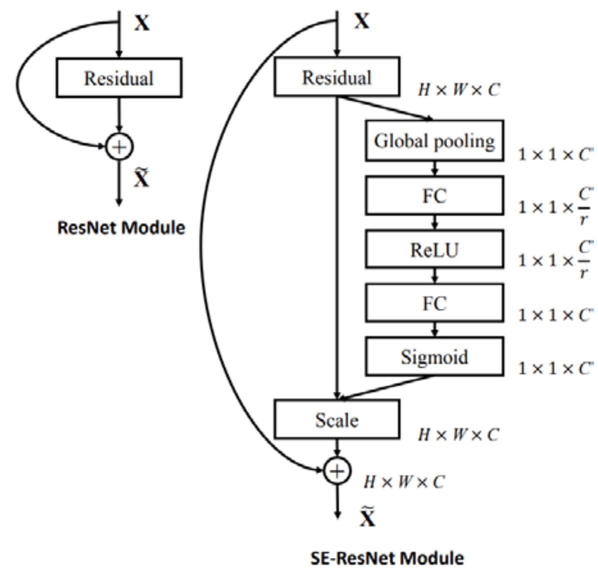


Xie, Saining, et al. "Aggregated residual transformations for deep neural networks." *CVPR,* 2017.

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]	$19.9^{\dagger}$	$4.9^{\dagger}$	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.

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, but little guidance on how to train CNNs.
- 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

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

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

$q = x + y$

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

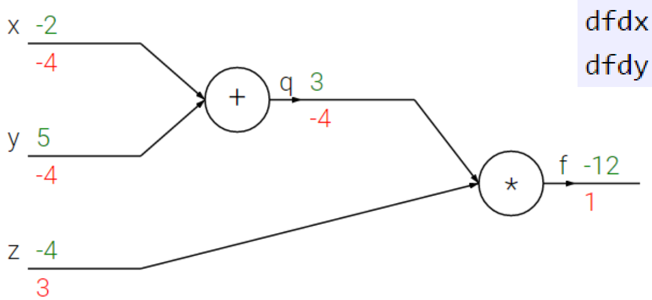
$f = qz$

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

Chain rule:

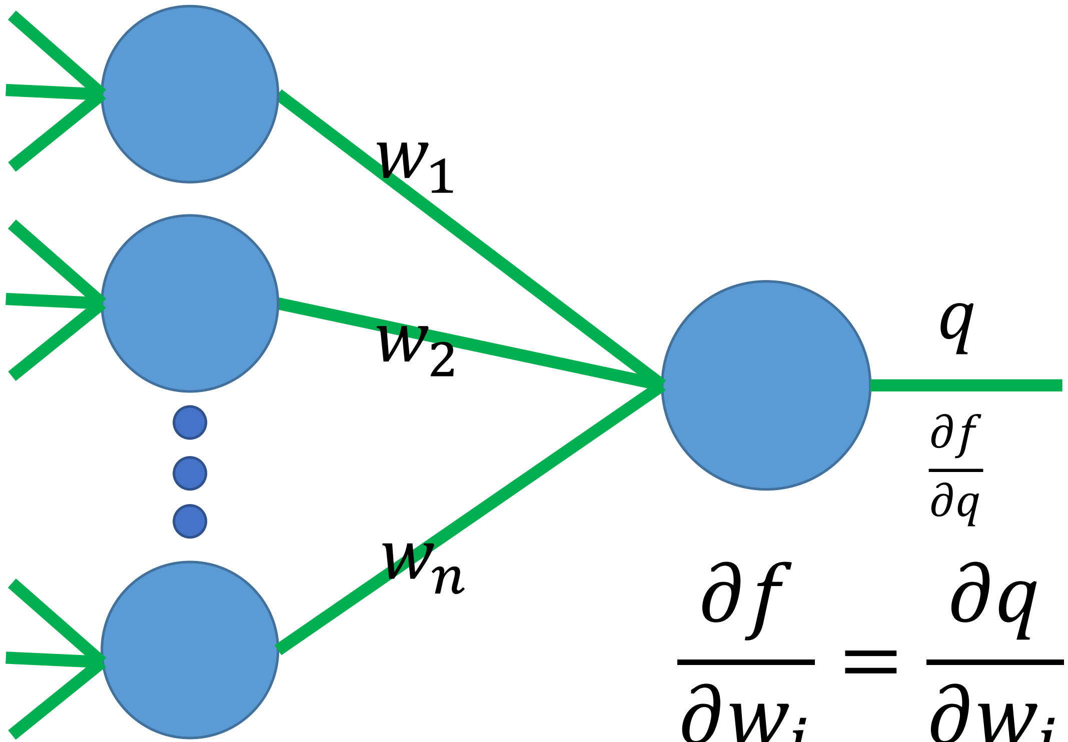
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

```
# 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)



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

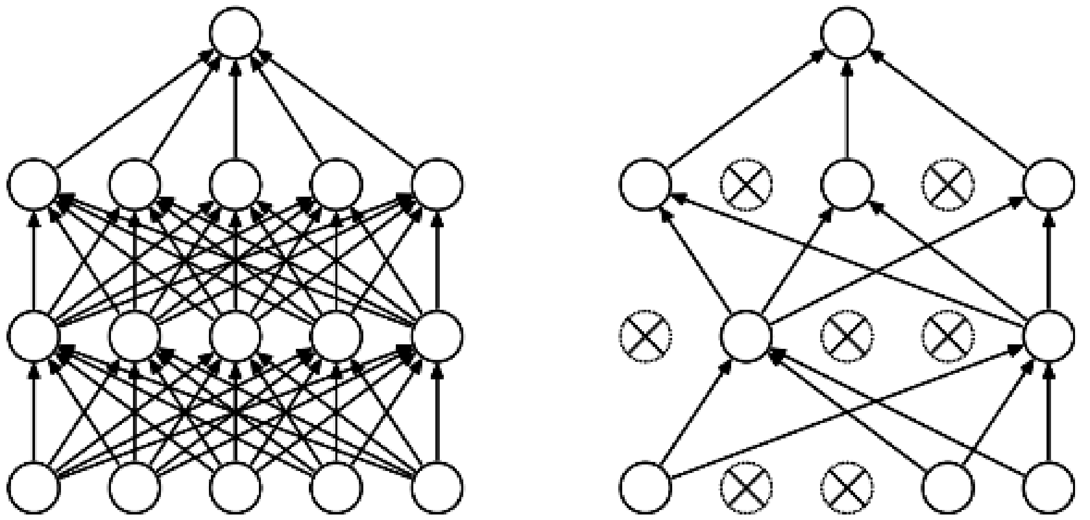
Local gradient

Gate gradient

Can be computed during forward pass

The gate receives this during backprop

Dropout



(a) Standard Neural Net (b) After applying dropout.

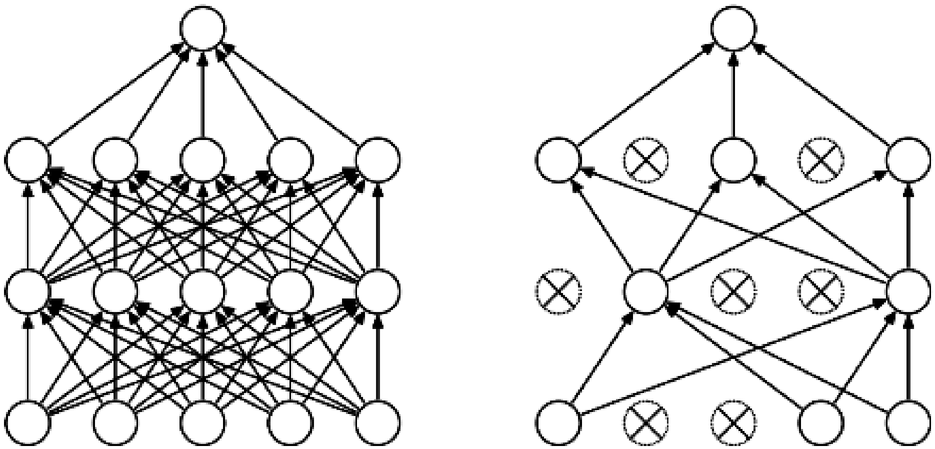
Intuition: successful **conspiracies**

- 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.

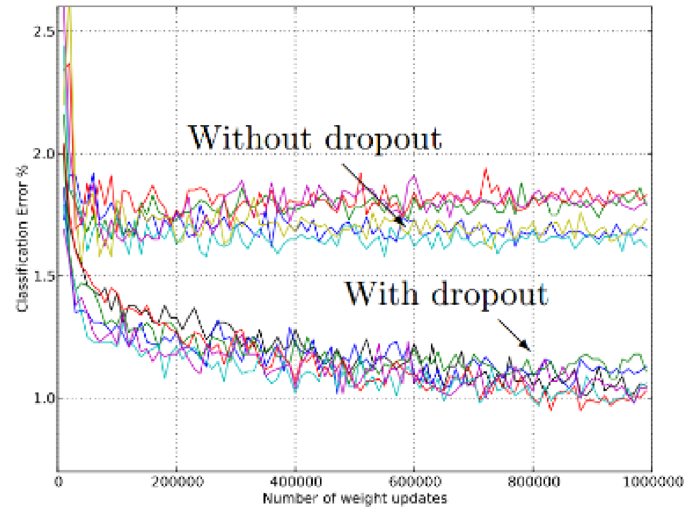
- Strategy B: plan 10 conspiracies each involving 5 people
  - Likely to succeed!

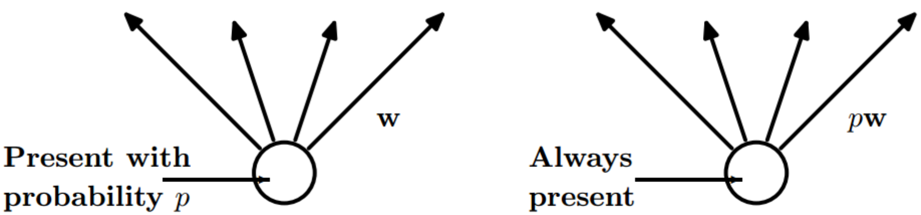
Dropout: A simple way to prevent neural networks from overfitting [Srivastava JMLR 2014]

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

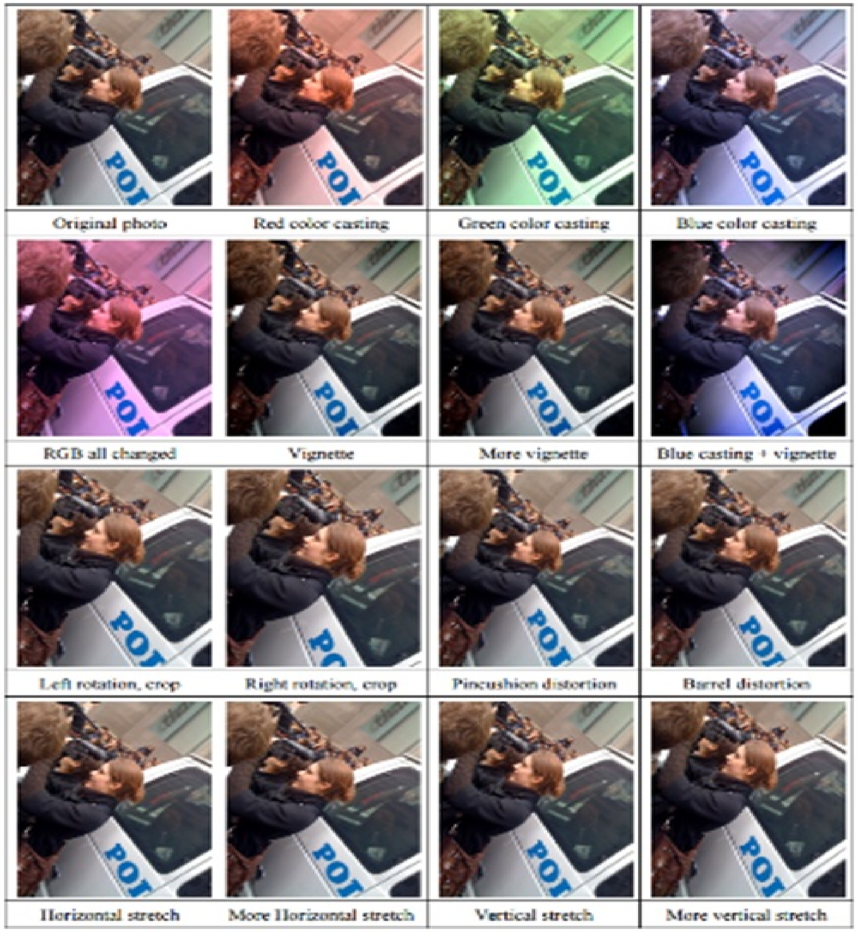
Model	Top-1 (val)	Top-5 (val)	Top-5 (test)
SVM on Fisher Vectors of Dense SIFT and Color Statistics	-	-	27.3
Avg of classifiers over FVs of SIFT, LBP, GIST and CSIFT	-	-	26.2
Conv Net + dropout (Krizhevsky et al., 2012)	40.7	18.2	-
Avg of 5 Conv Nets + dropout (Krizhevsky et al., 2012)	38.1	16.4	16.4

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

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

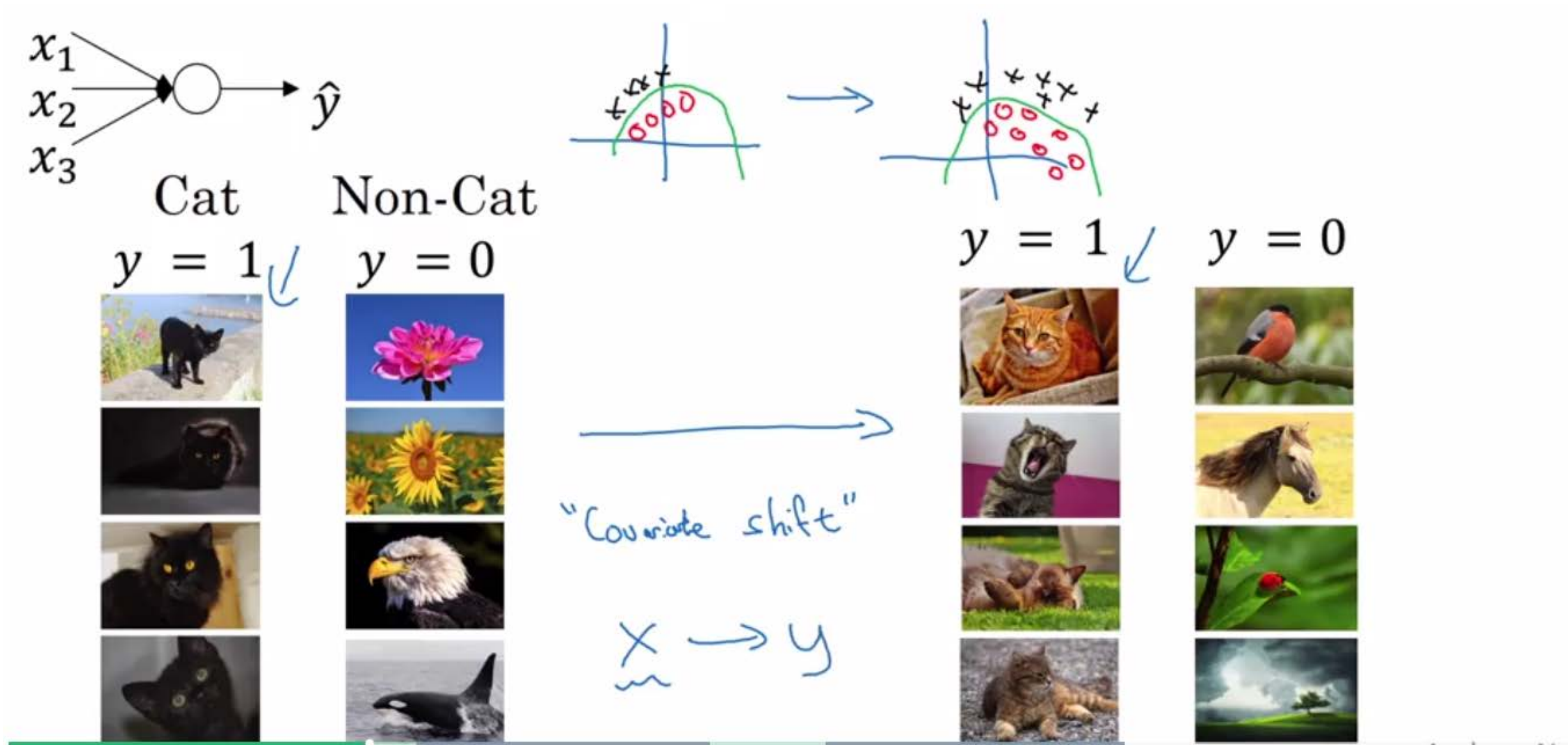
Data Augmentation (Jittering)

- 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

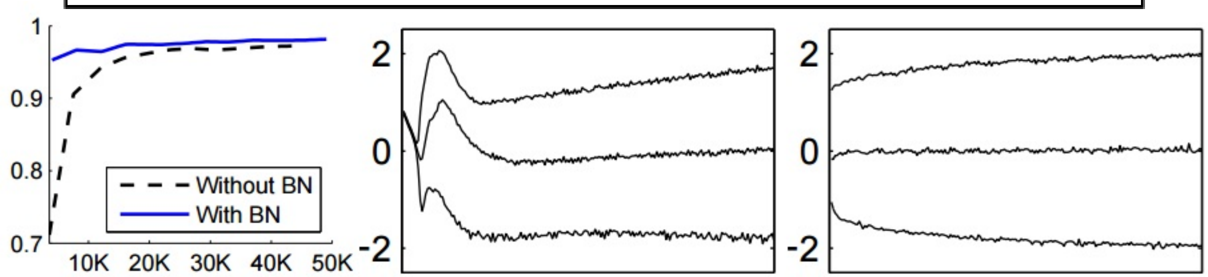
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

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

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

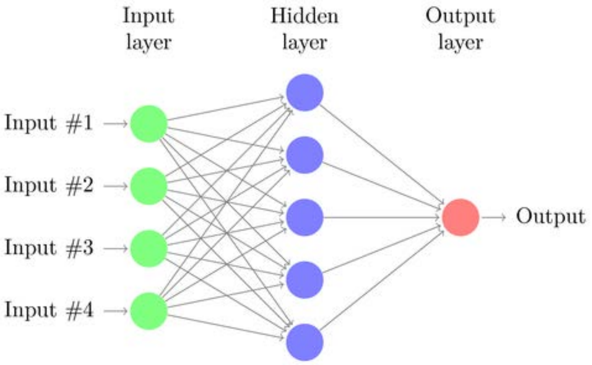


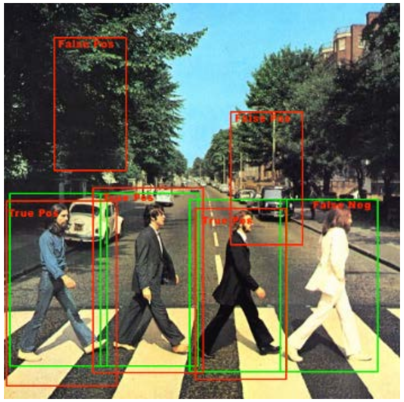
(a) (b) Without BN (c) With BN

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

What’s to Be Covered Today…

- Intro to Neural Networks & CNN
  - Linear Classification
  - Neural Network for Machine Vision
  - Multi-Layer Perceptron
  - Convolutional Neural Networks
- Image Segmentation
- Object Detection



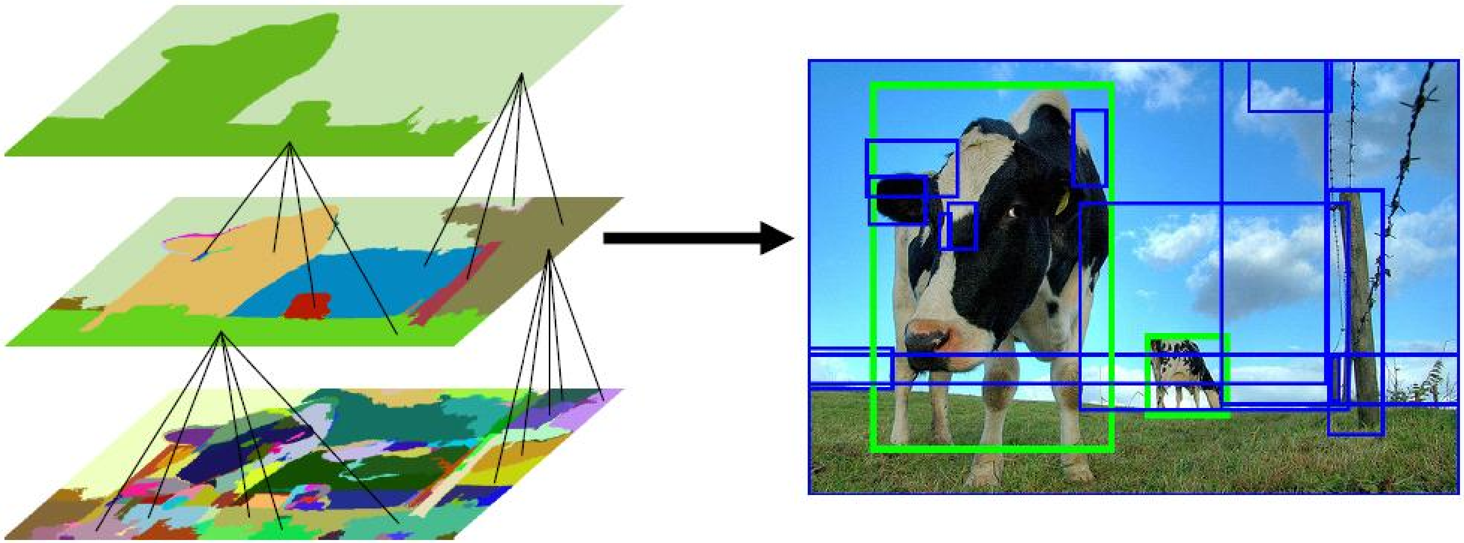


Many slides from Richard Turner, Fei-Fei Li, Yaser Sheikh, Simon Lucey, Kaiming He, and J.-B. Huang

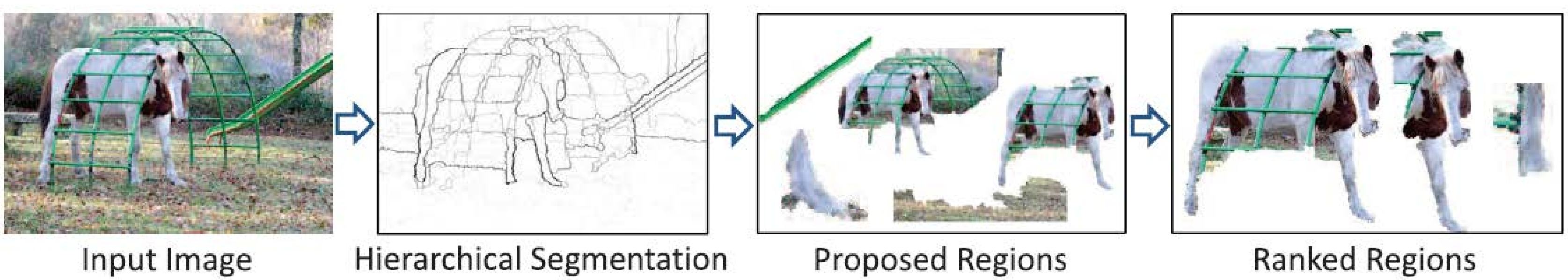
Image Segmentation

- Goal:
  Group pixels into meaningful or perceptually similar regions

Segmentation for Object Proposal



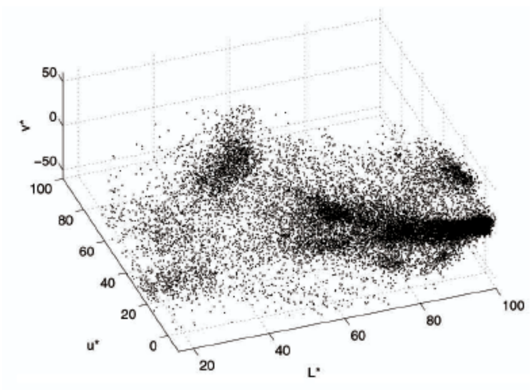
"Selective Search" [Sande, Uijlings et al. ICCV 2011, IJCV 2013]

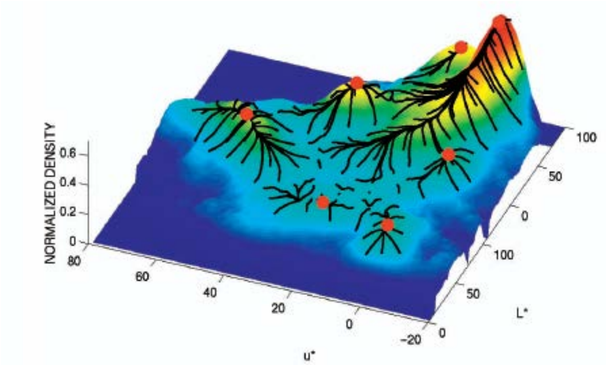


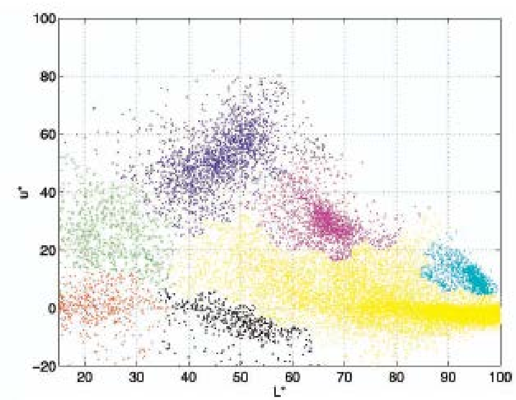
[Endres Hoiem ECCV 2010, IJCV 2014]

Segmentation via Clustering

- K-means clustering
- Mean-shift*
  - Find modes of the following non-parametric density



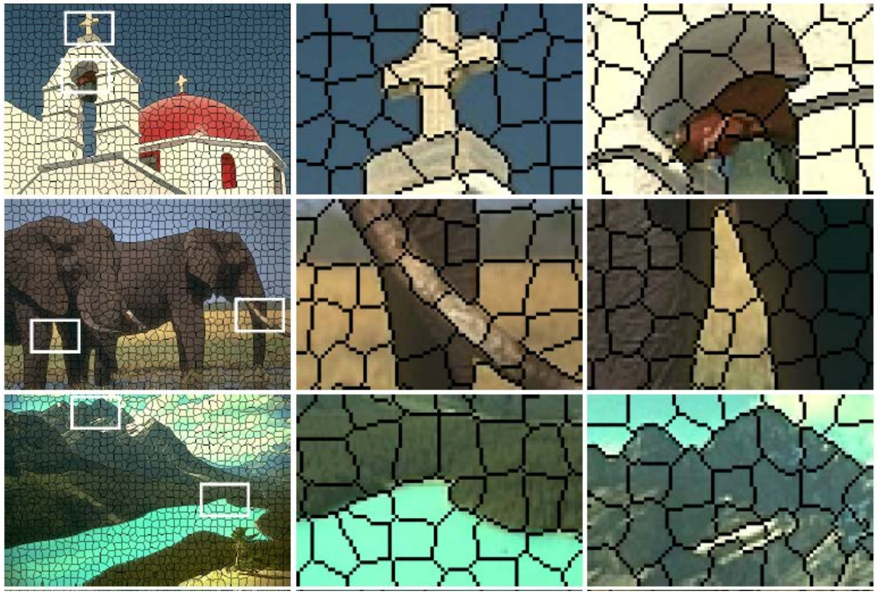




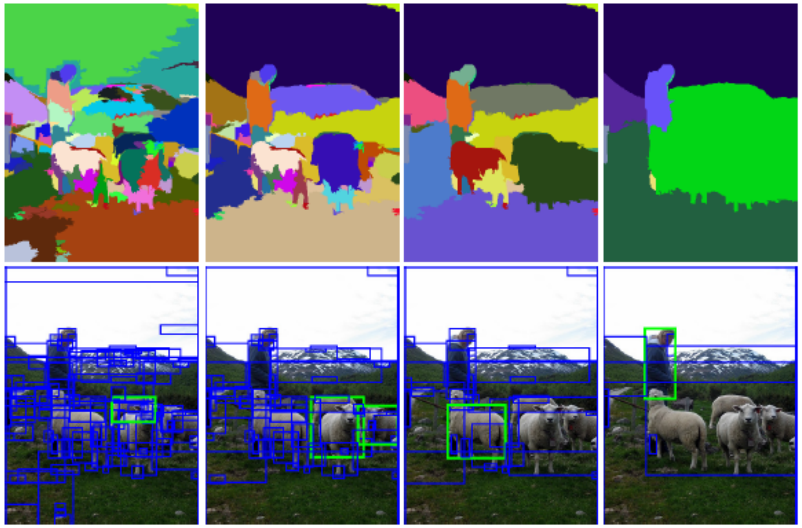
*D. Comaniciu and P. Meer, Mean Shift: A Robust Approach toward Feature Space Analysis, IEEE PAMI 2002.

Superpixels

- A simpler task of image segmentation
- Divide an image into a large number of regions, such that each region lies within object boundaries.
- Examples
  - Watershed
  - Felzenszwalb and Huttenlocher graph-based
  - Turbopixels
  - SLIC



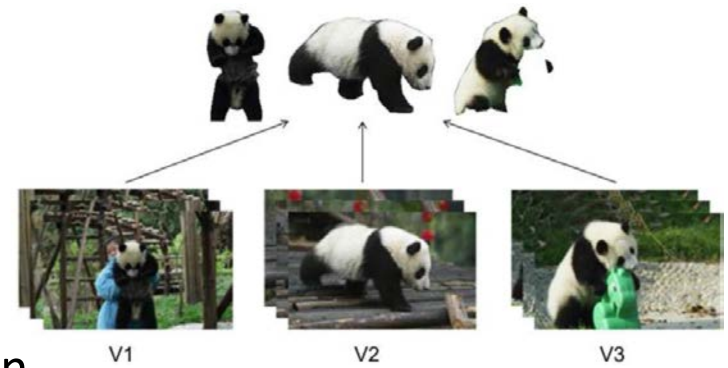
Multiple Segmentations



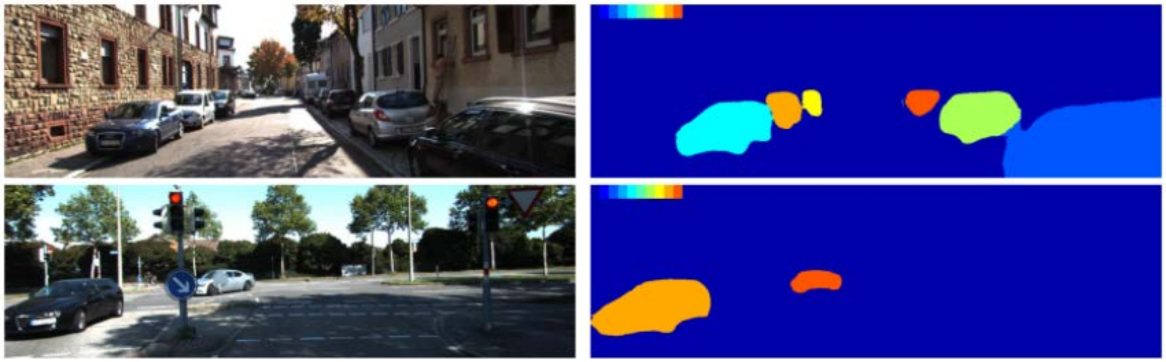
- Don't commit to one partitioning
- Hierarchical segmentation
  - Occlusion boundaries hierarchy:
    Hoiem et al. IJCV 2011 (uses trained classifier to merge)
  - Pb+watershed hierarchy: Arbeleaz et al. CVPR 2009
  - Selective search: FH + agglomerative clustering
  - Superpixel hierarchy
- Vary segmentation parameters
  - E.g., multiple graph-based segmentations or mean-shift segmentations
- Region proposals
  - Propose seed superpixel, try to segment out object that contains it
    (Endres Hoiem ECCV 2010, Carreira Sminchisescu CVPR 2010)

More Tasks in Segmentation

- Cosegmentation
  - Segmenting common objects from multiple images



- Instance Segmentation
  - Assign each pixel an object instance

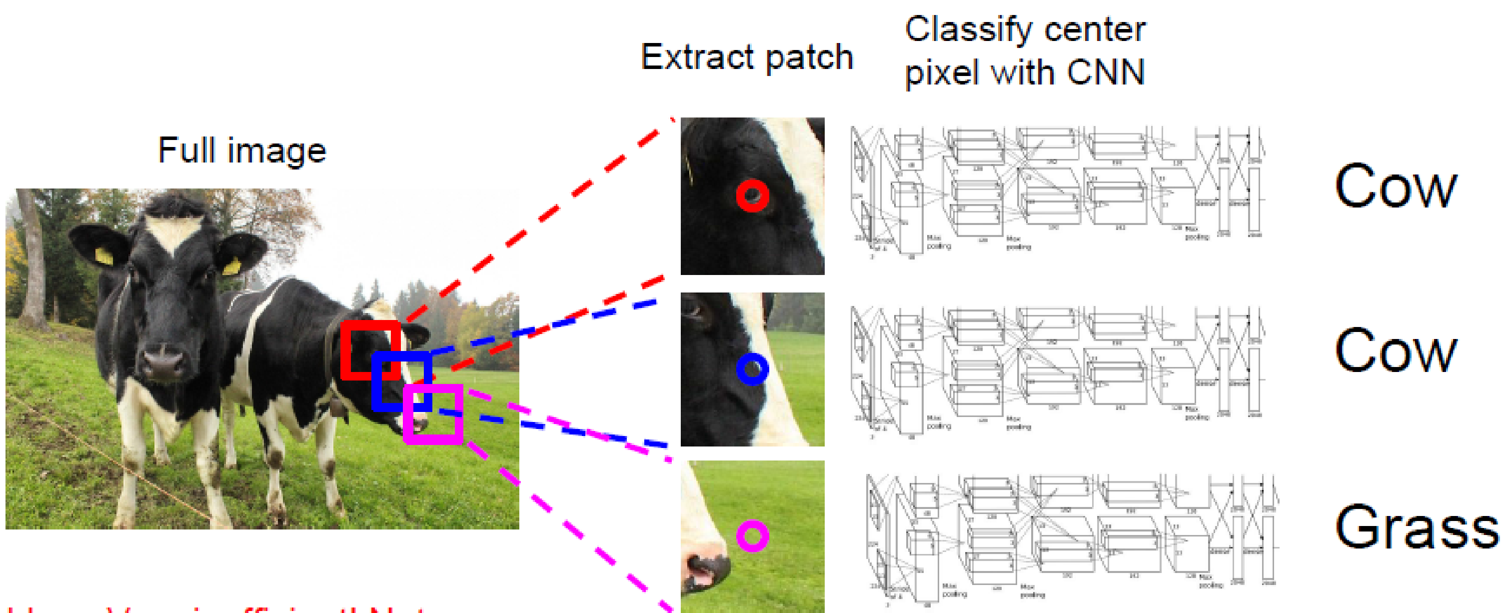


More Tasks in Segmentation

- Semantic Segmentation
  - Assign a class label to each pixel in the input image
  - Don’t differentiate instances, only care about pixels

Semantic Segmentation

- Sliding Window

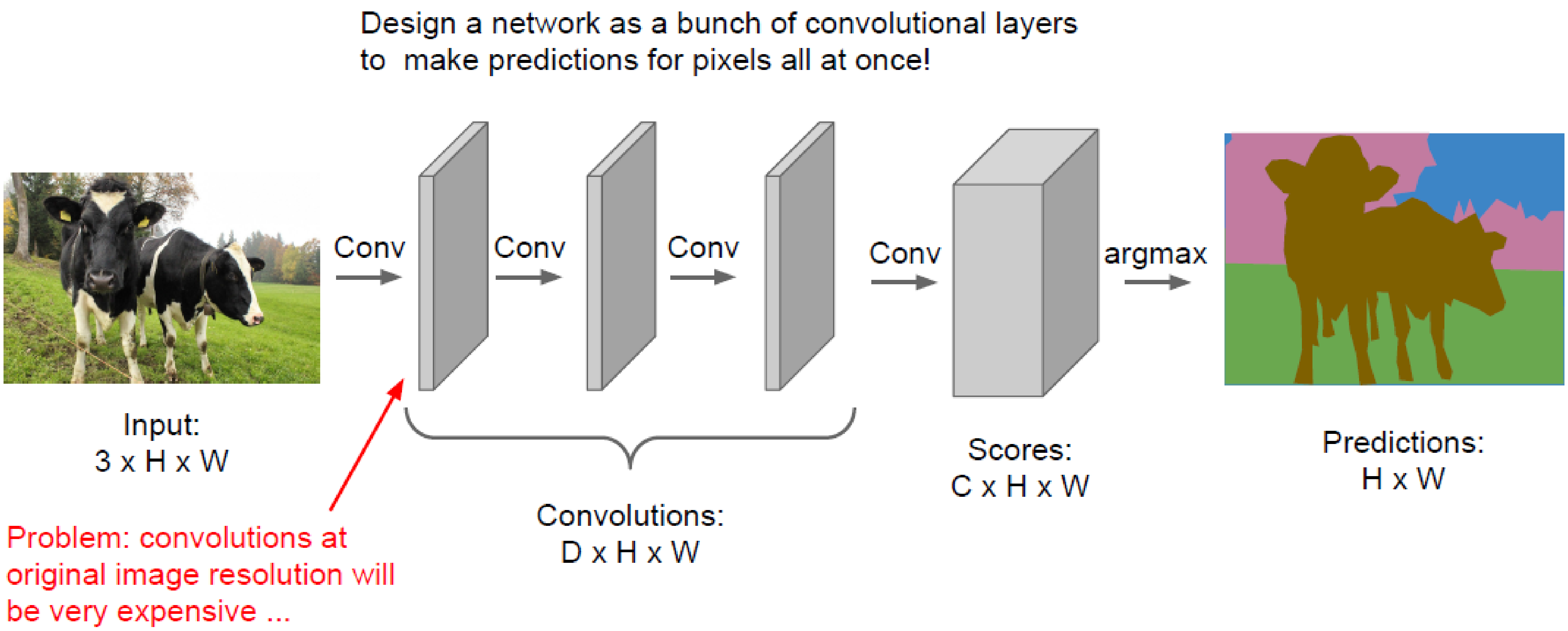


Problem: Very inefficient! Not reusing shared features between overlapping patches

Farabet et al, "Learning Hierarchical Features for Scene Labeling," TPAMI 2013
Pinheiro and Collobert, "Recurrent Convolutional Neural Networks for Scene Labeling", ICML 2014

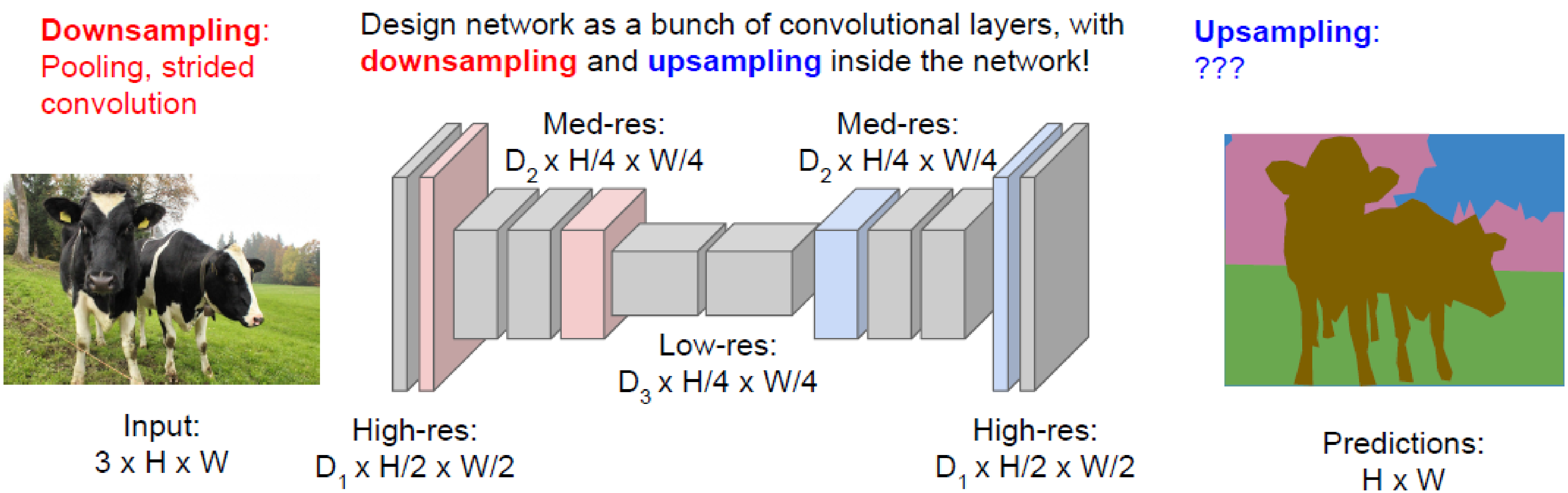
Semantic Segmentation

- Fully Convolutional Nets



Semantic Segmentation

- Fully Convolutional Nets (cont'd)



Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

In-Network Upsampling

- Unpooling

Nearest Neighbor

Input: 2 x 2

1	2
3	4

→

Output: 4 x 4

1	1	2	2
1	1	2	2
3	3	4	4
3	3	4	4

"Bed of Nails"

Input: 2 x 2

1	2
3	4

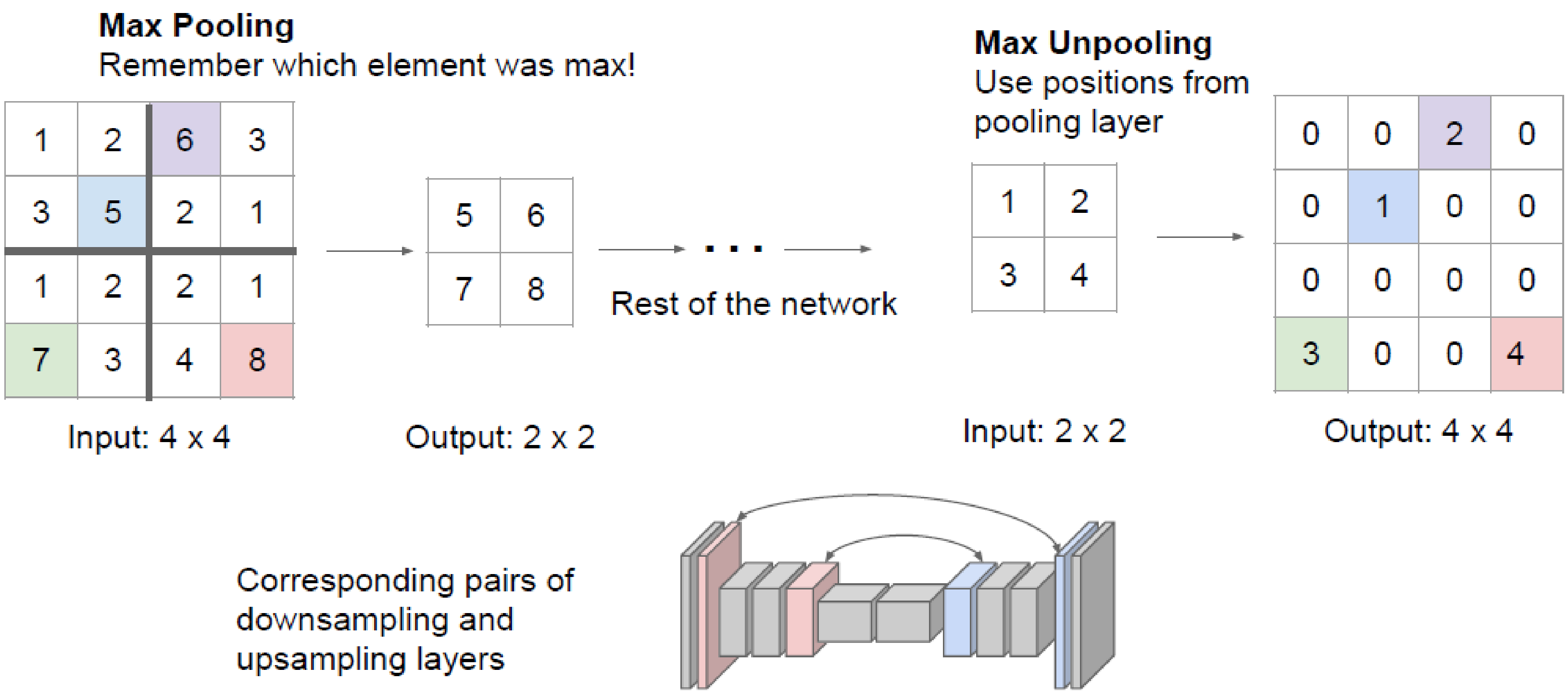
→

Output: 4 x 4

1	0	2	0
0	0	0	0
3	0	4	0
0	0	0	0

In-Network Upsampling

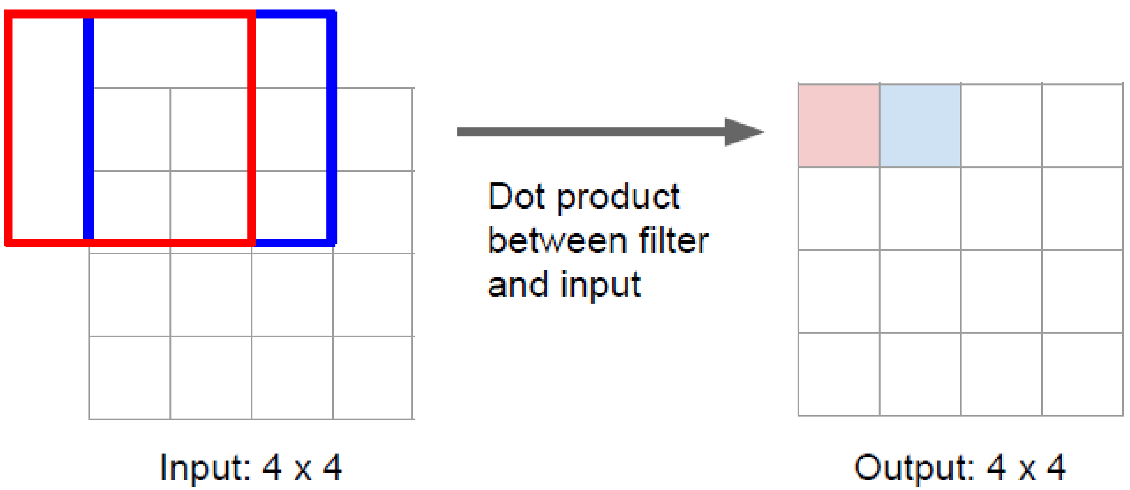
- Max Unpooling



In-Network Upsampling

- Learnable Upsampling: Transpose Convolution

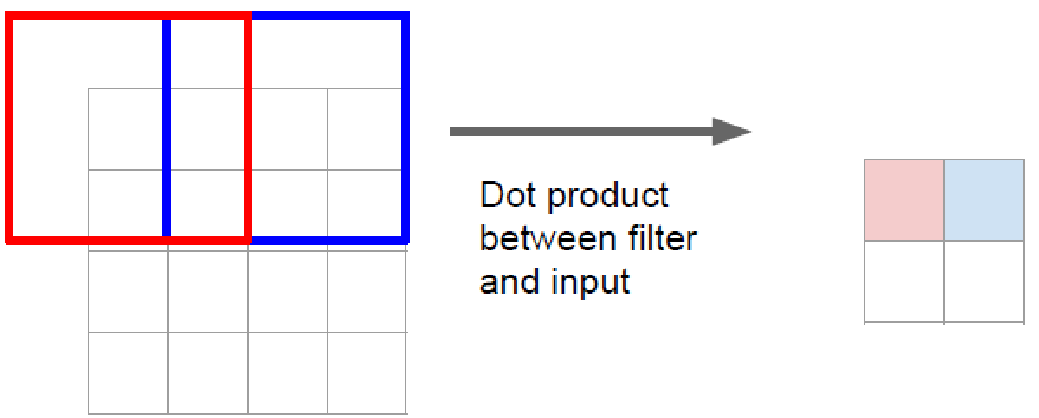
Recall: Normal 3 x 3 convolution, stride 1 pad 1



In-Network Upsampling

- Learnable Upsampling: Transpose Convolution

Recall: Normal 3 x 3 convolution, <u>stride 2</u> pad 1



Input: 4 x 4

Output: 2 x 2

Filter moves 2 pixels in the input for every one pixel in the output

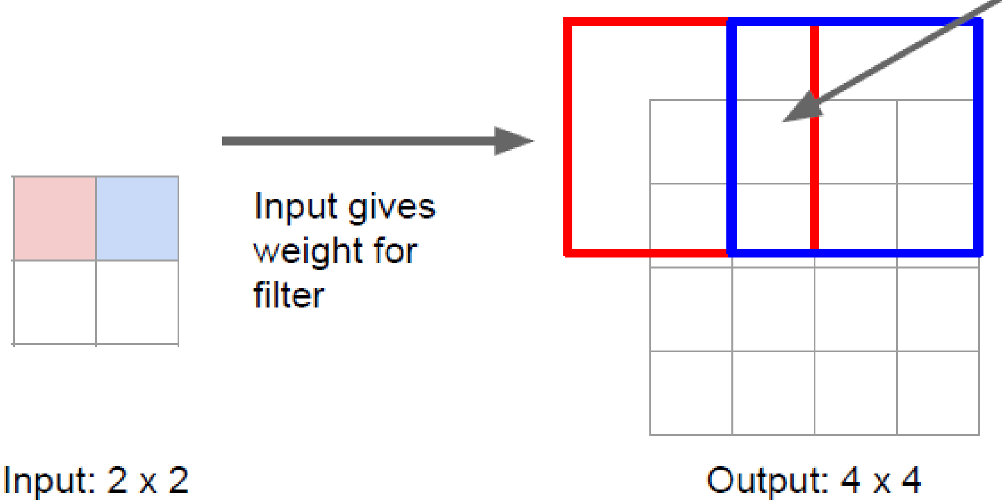
Stride gives ratio between movement in input and output

In-Network Upsampling

- Transpose Convolution

Other names:
-Deconvolution (bad)
-Upconvolution
-Fractionally strided convolution
-Backward strided convolution

3 x 3 **transpose** convolution, stride 2 pad 1

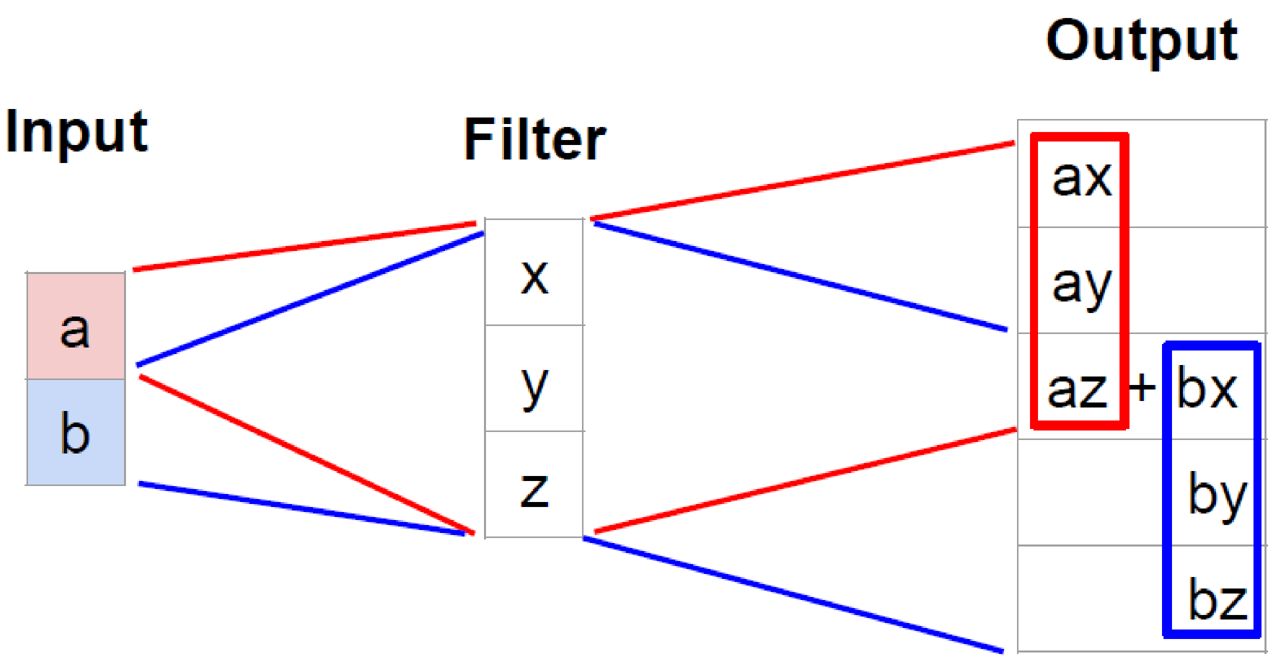


Filter moves 2 pixels in the <u>output</u> for every one pixel in the <u>input</u>

Stride gives ratio between movement in output and input

In-Network Upsampling

- Transpose Convolution
  - 1D example



Output contains copies of the filter weighted by the input, summing at where at overlaps in the output

Need to crop one pixel from output to make output exactly 2x input

In-Network Upsampling

- Transpose Convolution
  - Example as matrix multiplication

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & x & 0 & 0 & 0 \\ 0 & x & y & x & 0 & 0 \\ 0 & 0 & x & y & x & 0 \\ 0 & 0 & 0 & x & y & x \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ ax + by + cz \\ bx + cy + dz \\ cx + dy \end{bmatrix}$$

Example: 1D conv, kernel size=3, stride=1, padding=1

Convolution transpose multiplies by the transpose of the same matrix:

$$\vec{x} *^T \vec{a} = X^T\vec{a}$$

$$\begin{bmatrix} x & 0 & 0 & 0 \\ y & x & 0 & 0 \\ z & y & x & 0 \\ 0 & z & y & x \\ 0 & 0 & z & y \\ 0 & 0 & 0 & z \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} ax \\ ay + bx \\ az + by + cx \\ bz + cy + dx \\ cz + dy \\ dz \end{bmatrix}$$

When stride=1, convolution transpose is just a regular convolution (with different padding rules)

In-Network Upsampling

- Transpose Convolution
  - Example as matrix multiplication

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & 0 & x & y & z & 0 \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ bx + cy + dz \end{bmatrix}$$

Example: 1D conv, kernel size=3, <u>stride=2</u>, padding=1

Convolution transpose multiplies by the transpose of the same matrix:

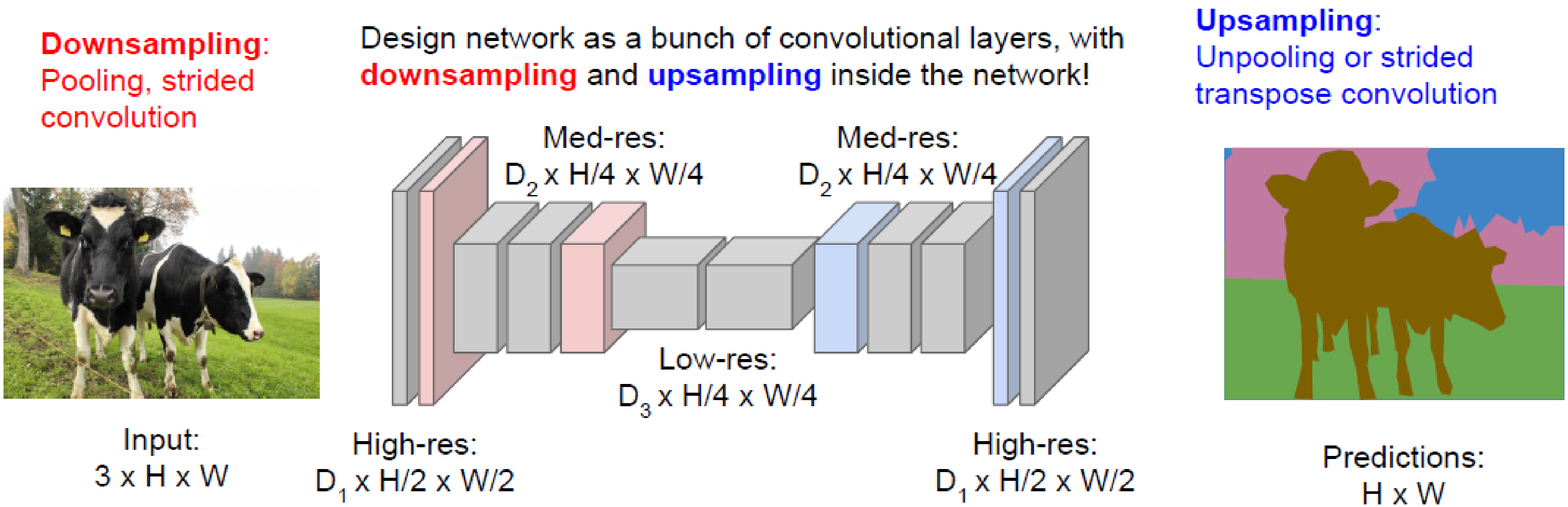
$$\vec{x} *^T \vec{a} = X^T\vec{a}$$

$$\begin{bmatrix} x & 0 \\ y & 0 \\ z & x \\ 0 & y \\ 0 & z \\ 0 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} ax \\ ay \\ az + bx \\ by \\ bz \\ 0 \end{bmatrix}$$

When stride>1, convolution transpose is no longer a normal convolution!

Fully Convolutional Networks (FCN)

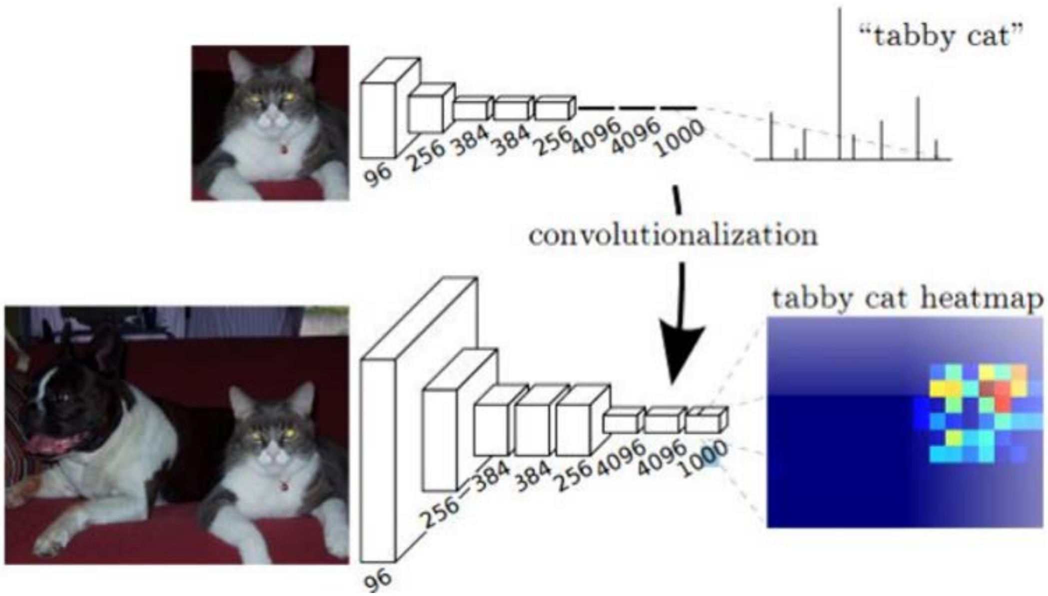
- Remarks
  - All layers are convolutional.
  - End-to-end training.



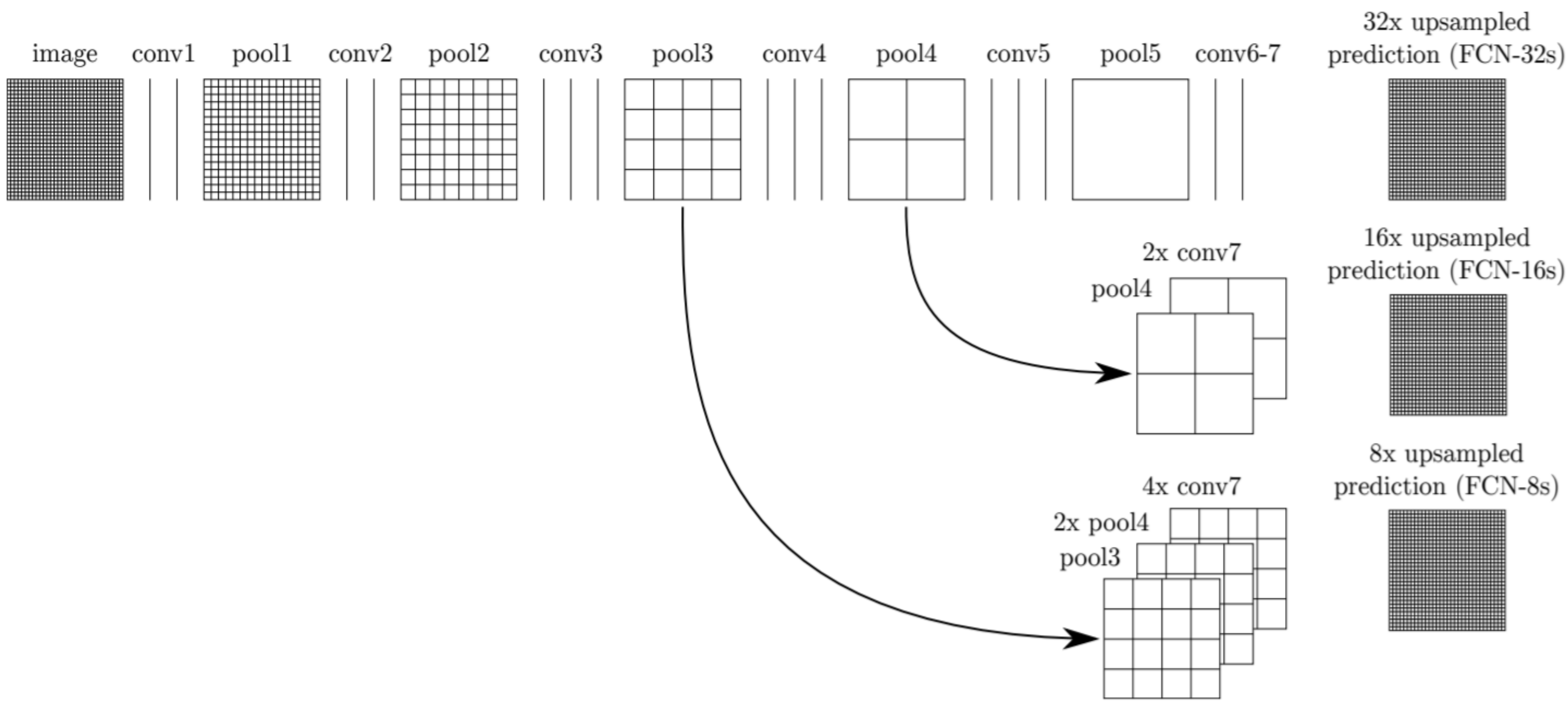
Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

Fully Convolutional Networks (FCN)

- More details
  - Adapt existing classification network to fully convolutional forms
  - Remove flatten layer and replace fully connected layers with conv layers
  - Use transpose convolution to upsample pixel-wise classification results



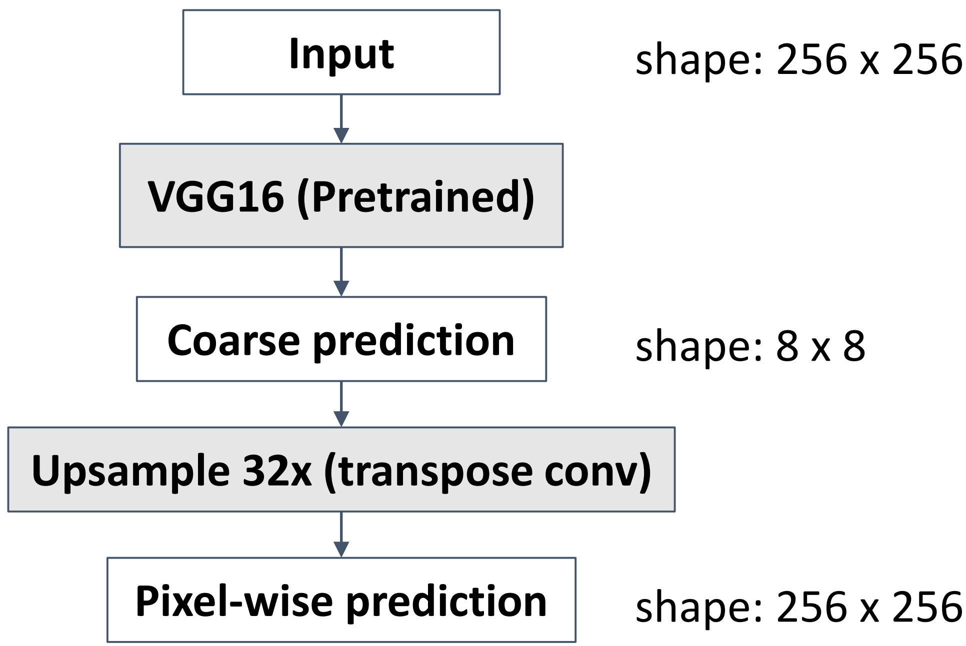
Fully Convolutional Networks (FCN)



Fully Convolutional Networks (FCN)

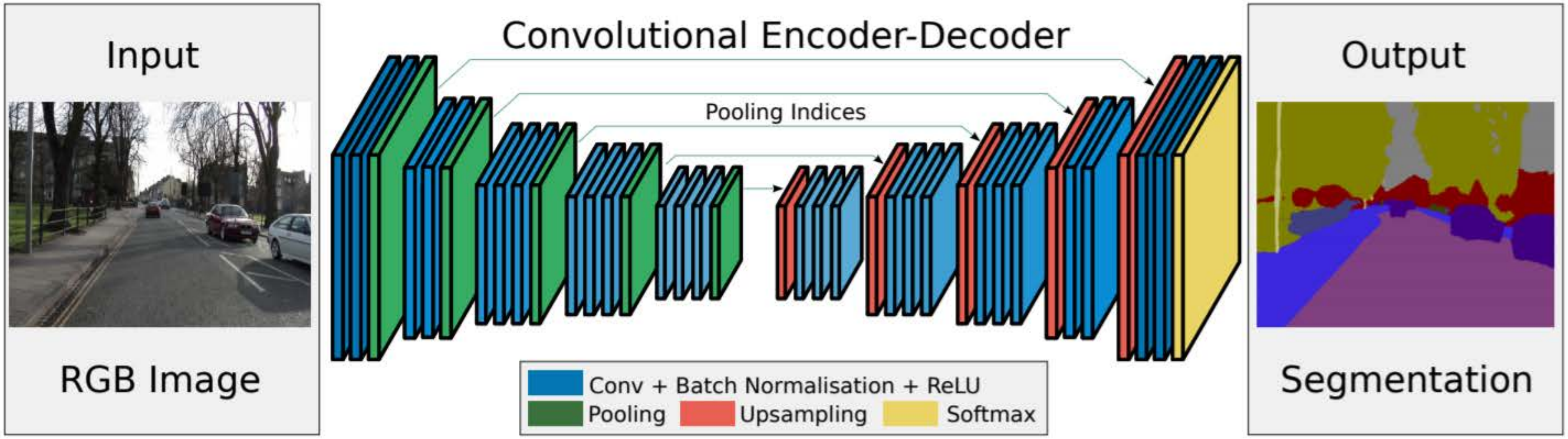
- Example
  - **VGG16-FCN32s**
  - Loss: pixel-wise cross-entropy

  i.e., compute cross-entropy between each pixel and its label, and average over all of them



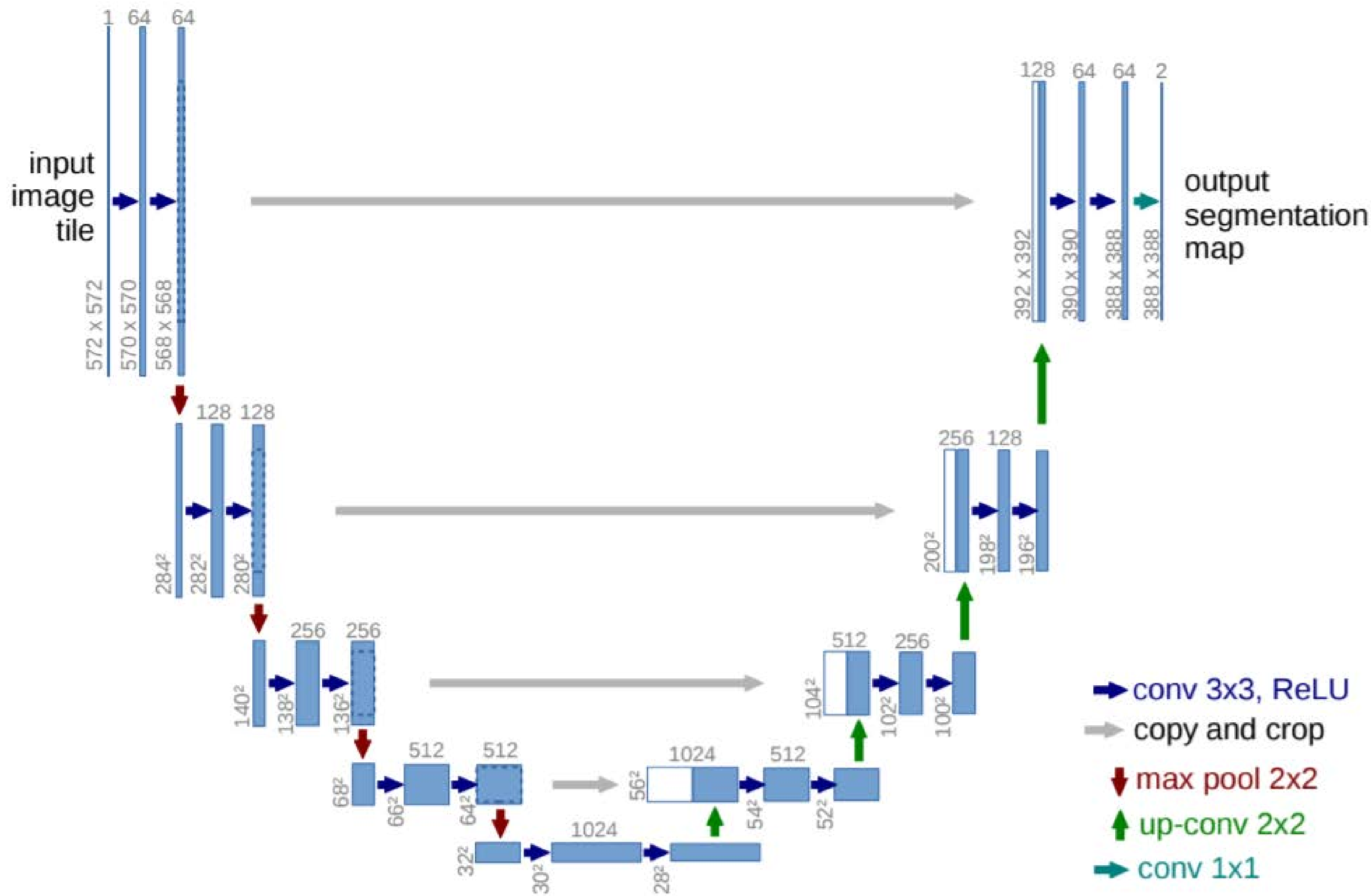
SegNet

- Efficient architecture (memory + computation time)
- Upsampling reusing max-unpooling indices
- Reasonable results without performance boosting addition
- Comparable to FCN



"SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation" [link]

U-Net



U-Net: Convolutional Networks for Biomedical Image Segmentation [link]