Learning

- Learning agents
- Inductive learning
  - Different Learning Scenarios
  - Evaluation
- Neural Networks
  - Perceptrons
  - Multilayer Perceptrons
  - Deep Learning
- Reinforcement Learning
  - Temporal Differences
  - Q-Learning
  - SARSA

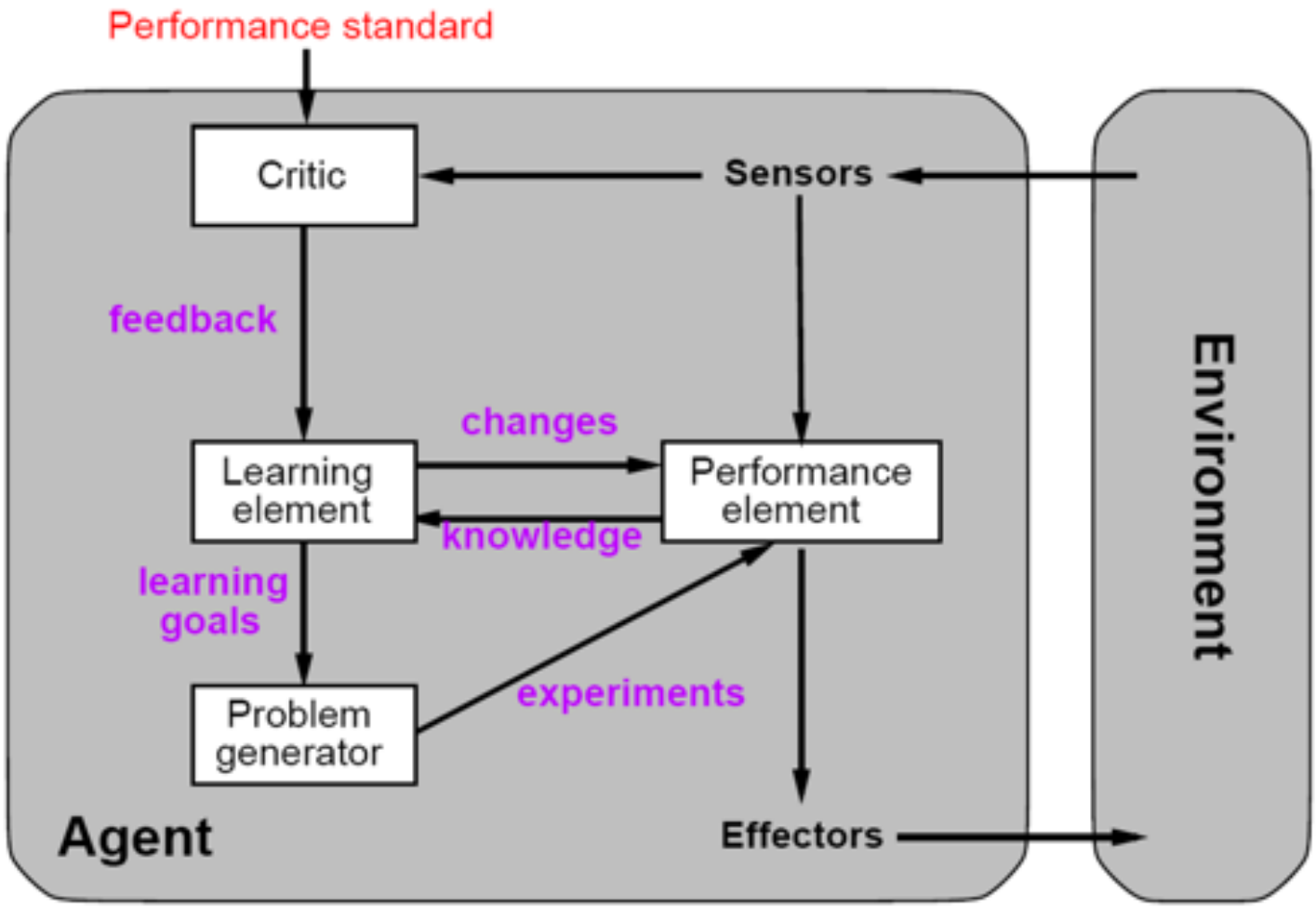
Material from
Russell & Norvig,
chapters 18.1,
18.2, 20.5 and 21

Slides based on Slides
by Russell/Norvig,
Ronald Williams,
Torsten Reil,
Alyosha Efros,
Fei-Fei li,
Viktoriia Sharmanska,
Geoff Hinton,
and many more

Learning

- Learning is essential for unknown environments,
  - i.e., when designer lacks omniscience
- Learning is useful as a system construction method,
  - i.e., expose the agent to reality rather than trying to write it down
- Learning modifies the agent's decision mechanisms to improve performance

Learning Agents



Learning Element

- Design of a learning element is affected by
  - Which components of the performance element are to be learned
  - What feedback is available to learn these components
  - What representation is used for the components

- Type of feedback:
  - Supervised learning:
    - correct answers for each example
  - Unsupervised learning:
    - correct answers not given
  - Reinforcement learning:
    - occasional rewards for good actions

Different Learning Scenarios

Supervised Learning

- A teacher provides the value for the target function for all training examples (labeled examples)
- concept learning, classification, regression

Reinforcement Learning

- The teacher only provides feedback but not example values

Semi-supervised Learning

- Only a subset of the training examples are labeled

Unsupervised Learning

- There is no information except the training examples
- clustering, subgroup discovery, association rule discovery

Inductive Learning

Simplest form: learn a function from examples

- f is the (unknown) target function
- An example is a pair $(x, f(x))$
- Problem: find a hypothesis h
  - given a training set of examples
  - such that $h \approx f$
  - on *all* examples
    - i.e. the hypothesis must generalize from the training examples
- This is a **highly simplified** model of real learning:
  - Ignores prior knowledge
  - Assumes examples are given

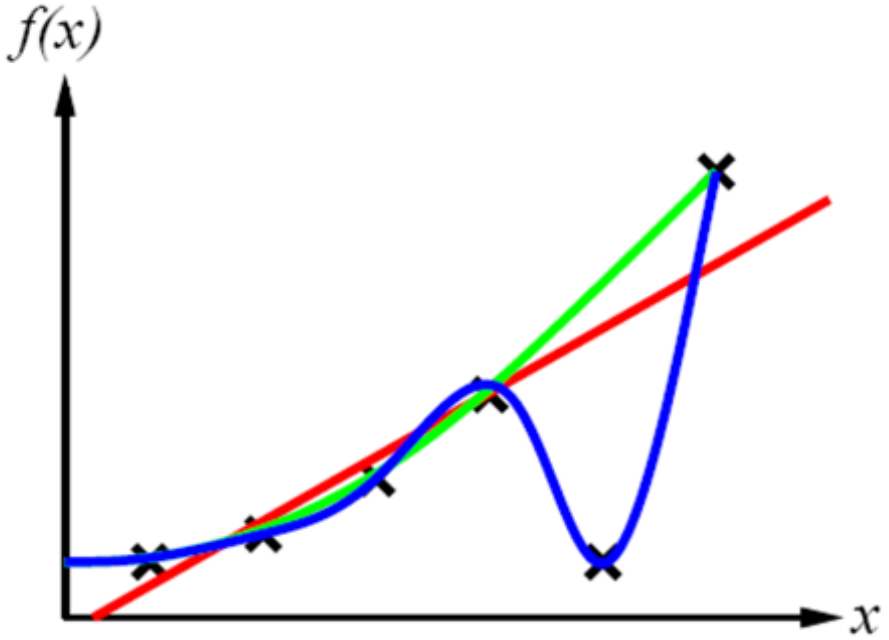
Inductive Learning Method

Construct/adjust h to agree with f on training set

- h is consistent if it agrees with f on all examples

Example:

- curve fitting



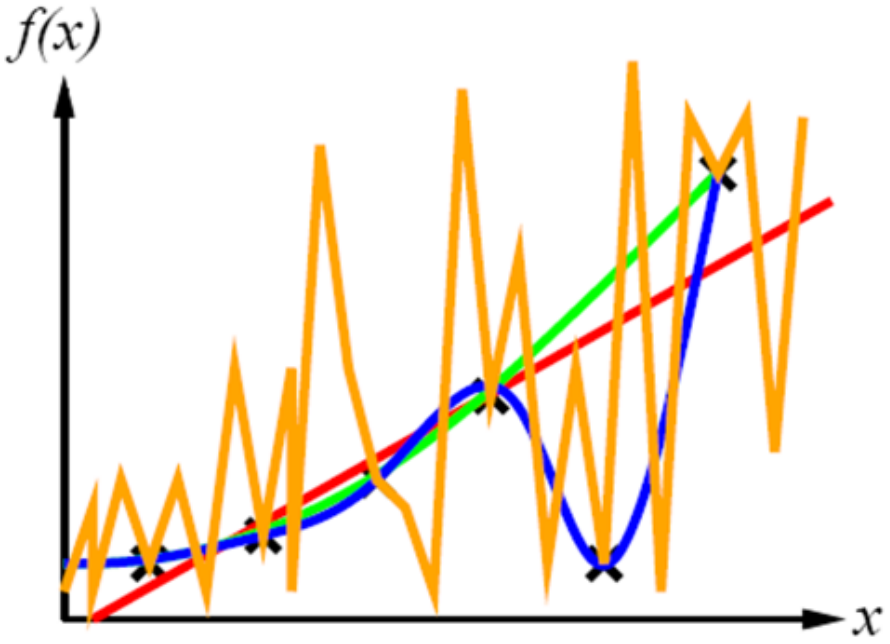
Inductive Learning Method

Construct/adjust h to agree with f on training set

- h is consistent if it agrees with f on all examples

Example:

- curve fitting

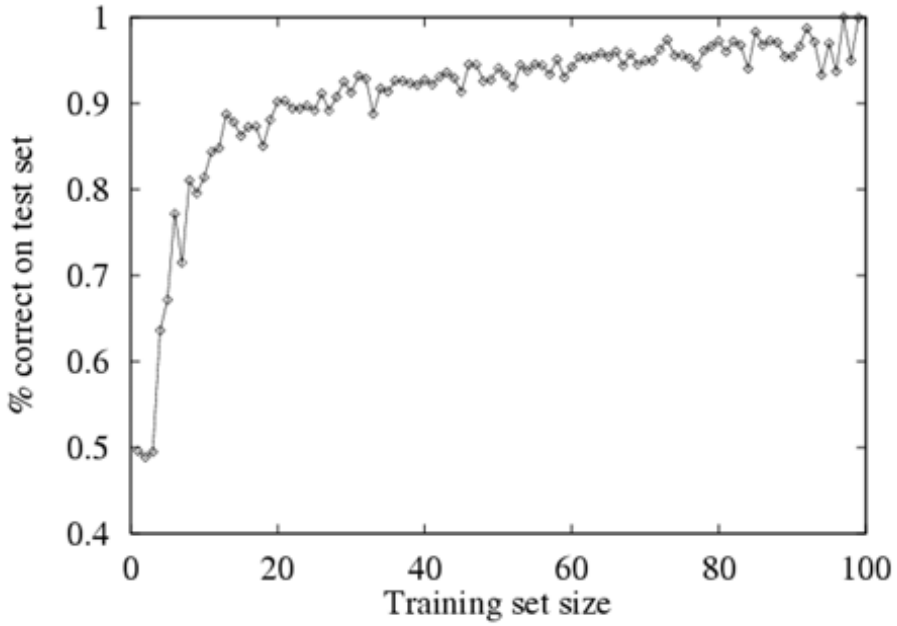


- Ockham's Razor
  - The best explanation is the simplest explanation that fits the data
- Overfitting Avoidance
  - maximize a combination of consistency and simplicity

Performance Measurement

- How do we know that $h \approx f$?
  - Use theorems of computational/statistical learning theory
  - Or try h on a new test set of examples where f is known
    (use same distribution over example space as training set)

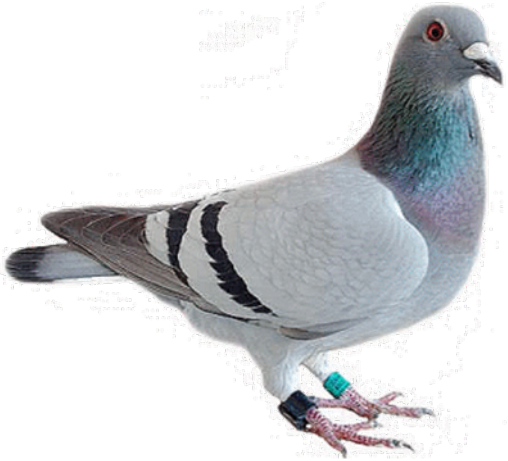
Learning curve = % correct on test set over training set size

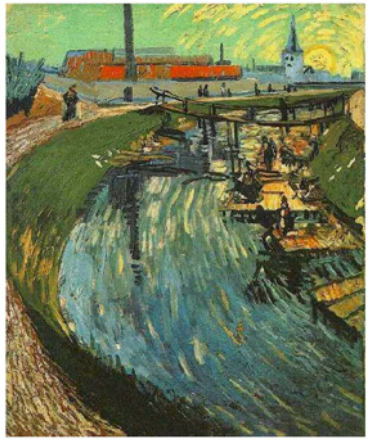


Pigeons as Art Experts

Famous experiment (Watanabe *et al.* 1995, 2001)

- Pigeon in Skinner box
- Present paintings of two different artists (e.g. Chagall / Van Gogh)
- Reward for pecking when presented a particular artist





Results

- Pigeons were able to discriminate between Van Gogh and Chagall with 95% accuracy
  - when presented with pictures they had been trained on
- Discrimination still 85% successful for previously unseen paintings of the artists

→ Pigeons do not simply memorise the pictures
  - They can extract and recognise patterns (the ‘style’)
  - They generalise from the already seen to make predictions

- This is what neural networks (biological and artificial) are good at (unlike conventional computer)

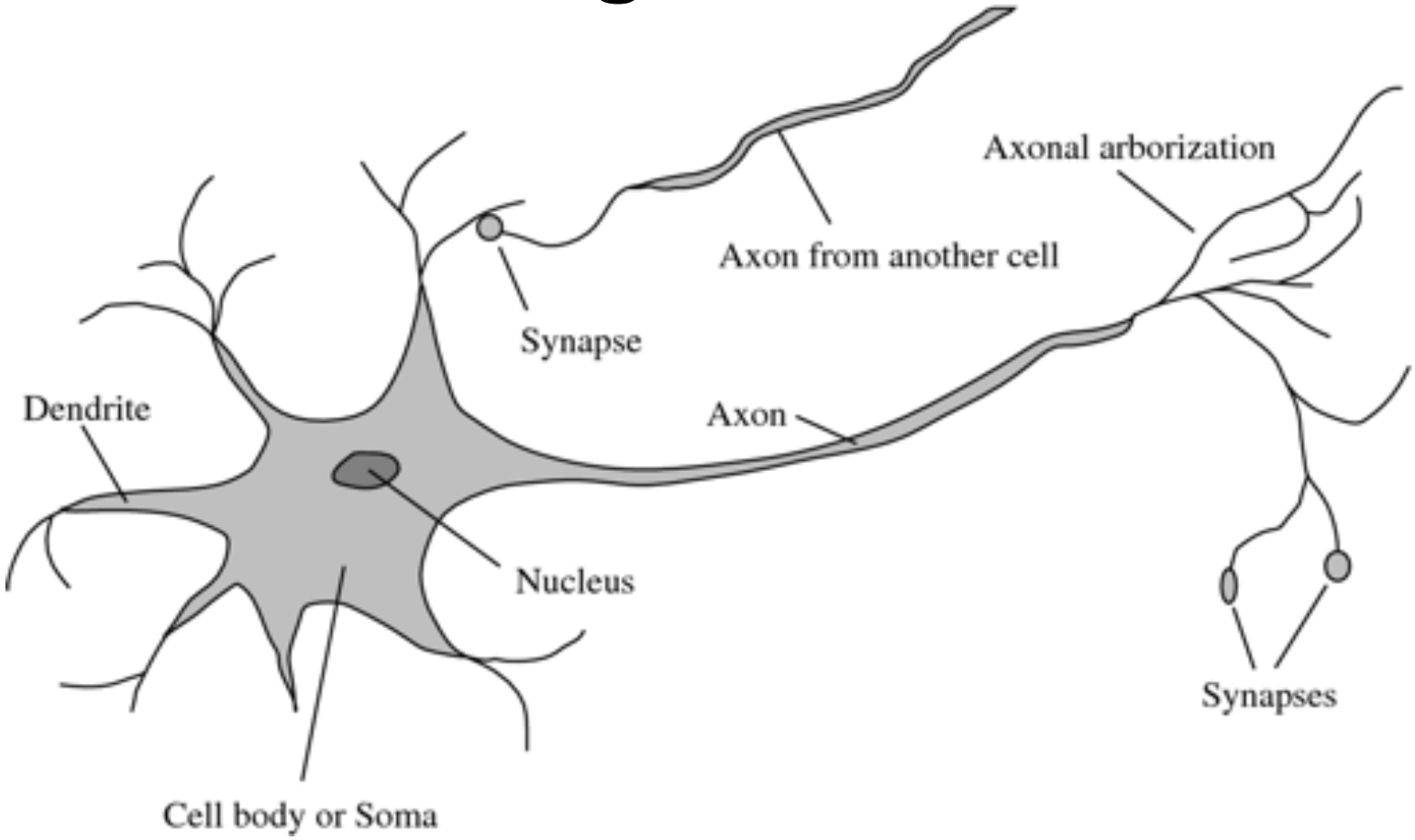
What are Neural Networks?

- **(Highly simplified)** models of the brain and nervous system
- Highly parallel
  - Process information much more like the brain than a serial computer
- Learning

- Very simple principles
- Very complex behaviours

- Applications
  - As powerful problem solvers
  - As (approx.) biological models

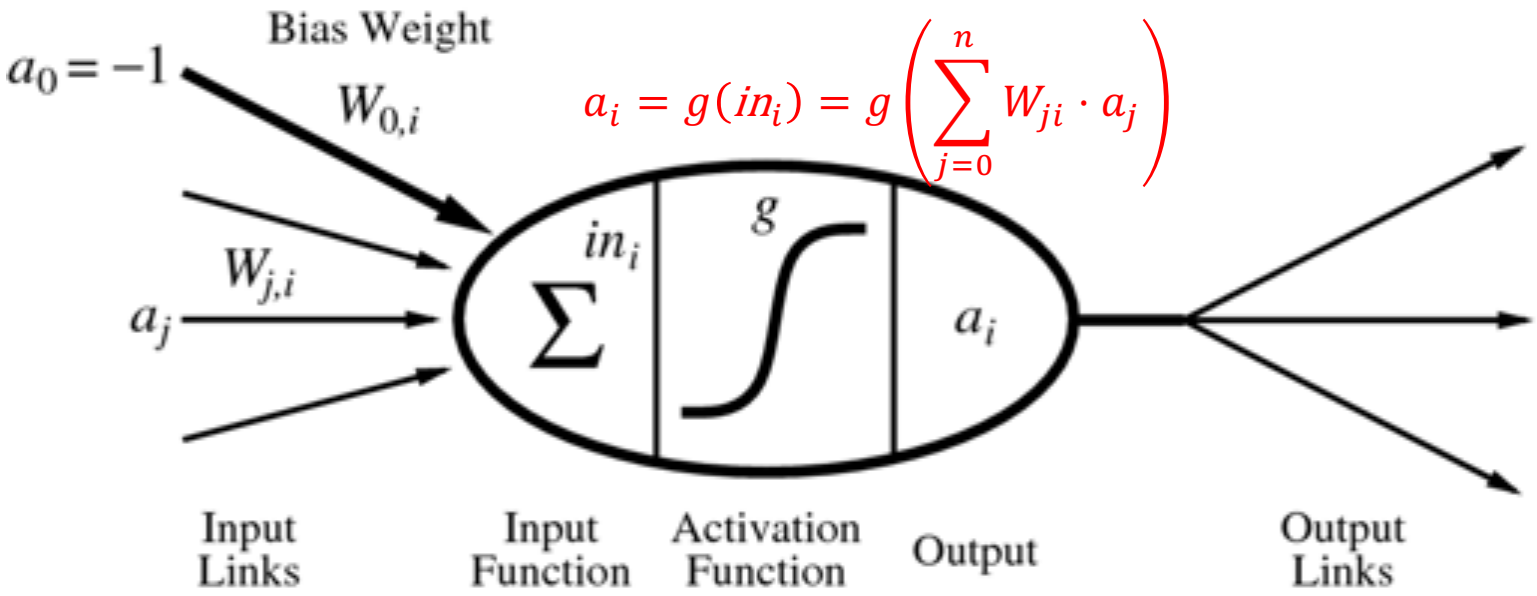
A Biological Neuron



- Neurons are connected to each other via synapses
- If a neuron is activated, it spreads its activation to all connected neurons

An Artificial Neuron

(McCulloch-Pitts,1943)



- Neurons correspond to nodes or units
- A link from unit j to unit i propagates activation a_j from j to i
- The weight $W_{j,i}$ of the link determines the strength and sign of the connection
- The total input activation is the sum of the input activations
- The output activation is determined by the activiation function g

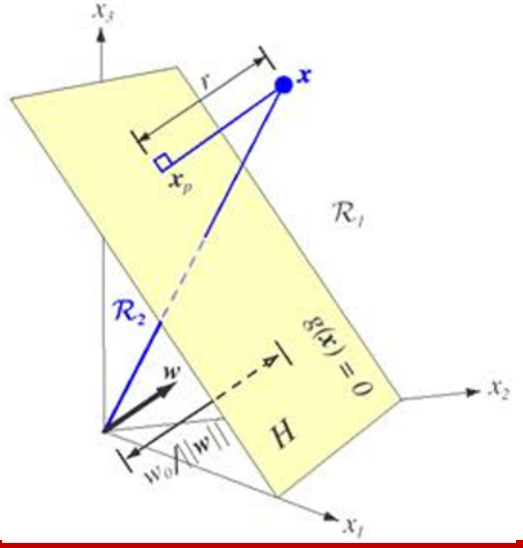
Perceptron

(Rosenblatt 1957, 1960)

- A single node
  - connecting n input signals a_j with one output signal a
  - typically signals are -1 or $+1$
- Activation function
  - A simple threshold function:

$$a = \begin{cases} -1 & if \sum_{j=0}^{n} W_j \cdot a_j \leq 0 \\ 1 & if \sum_{j=0}^{n} W_j \cdot a_j > 0 \end{cases}$$

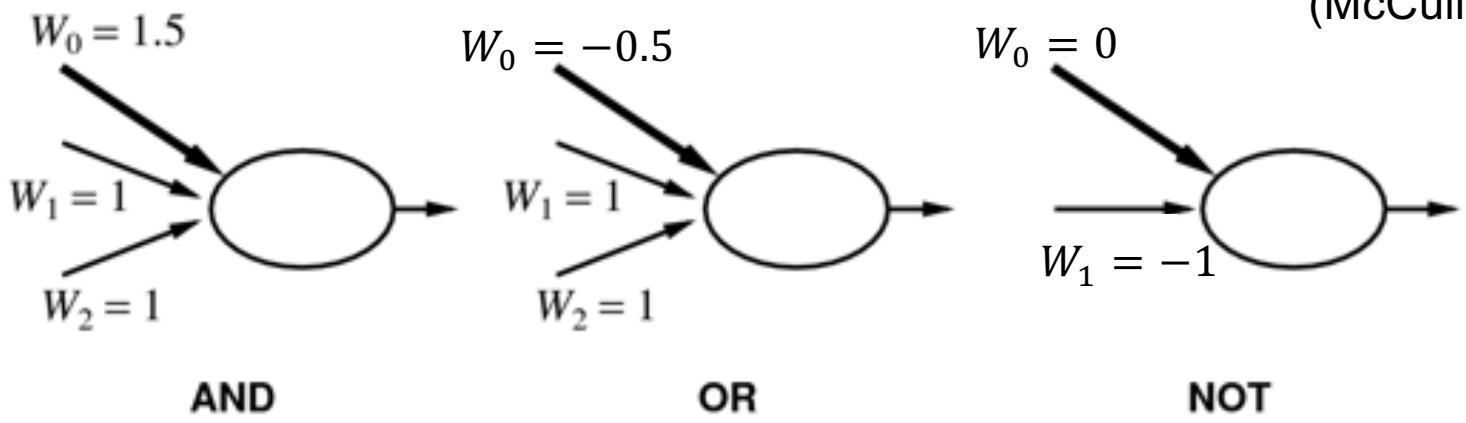
- Thus it implements a linear separator
  - i.e., a hyperplane that divides n-dimensional space into a region with output -1 and a region with output 1



Perceptrons and Boolean Functions

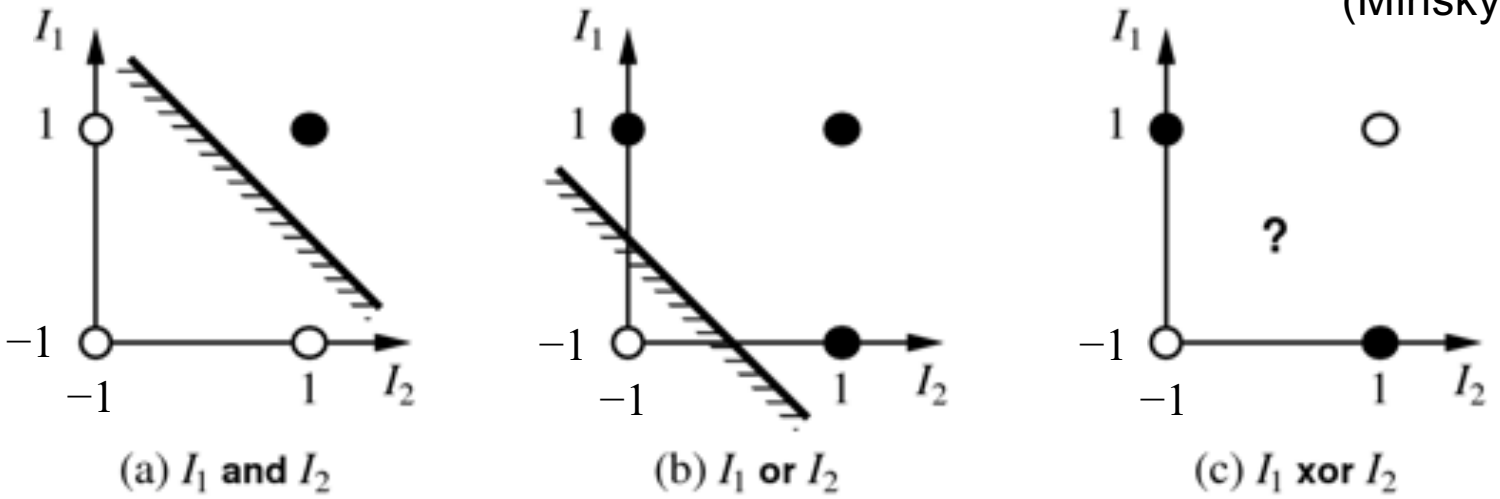
- a Perceptron can implement all elementary logical functions

(McCullogh & Pitts, 1943)



- more complex functions like XOR cannot be modeled

(Minsky & Papert, 1969)



no linear separation possible

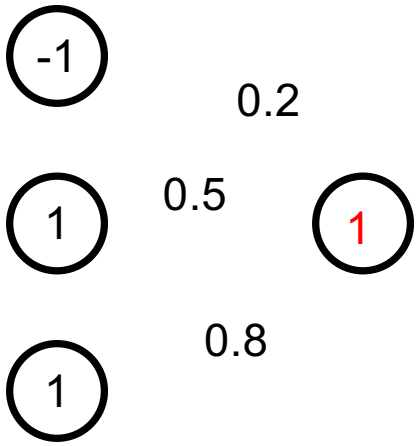
Perceptron Learning

- Perceptron Learning Rule for Supervised Learning

$$W_j \leftarrow W_j + \alpha \cdot \big(f(\boldsymbol{x}) - h(\boldsymbol{x})\big) \cdot x_j$$

learning rate error

- Example:



Computation of output signal $h(x)$

$$\text{in}(x) = -1 \cdot 0.2 + 1 \cdot 0.5 + 1 \cdot 0.8 = 1.1$$

$$h(x) = 1 \, because \, in(x) > 0 \, (activation \, function)$$

Assume target value $f(x) = -1$ (and $\alpha = 0.5$)

$$W_0 \leftarrow 0.2 + 0.5 \cdot (-1 - 1) \cdot -1 = 0.2 + 1 = 1.2$$

$$W_1 \leftarrow 0.5 + 0.5 \cdot (-1 - 1) \cdot 1 = 0.5 - 1 = -0.5$$

$$W_2 \leftarrow 0.8 + 0.5 \cdot (-1 - 1) \cdot 1 = 0.8 - 1 = -0.2$$

Measuring the Error of a Network

- The error for one training example $\mathbf{x}$ can be measured by the squared error
  - the squared difference of the output value $h(\mathbf{x})$ and the desired target value $f(\mathbf{x})$

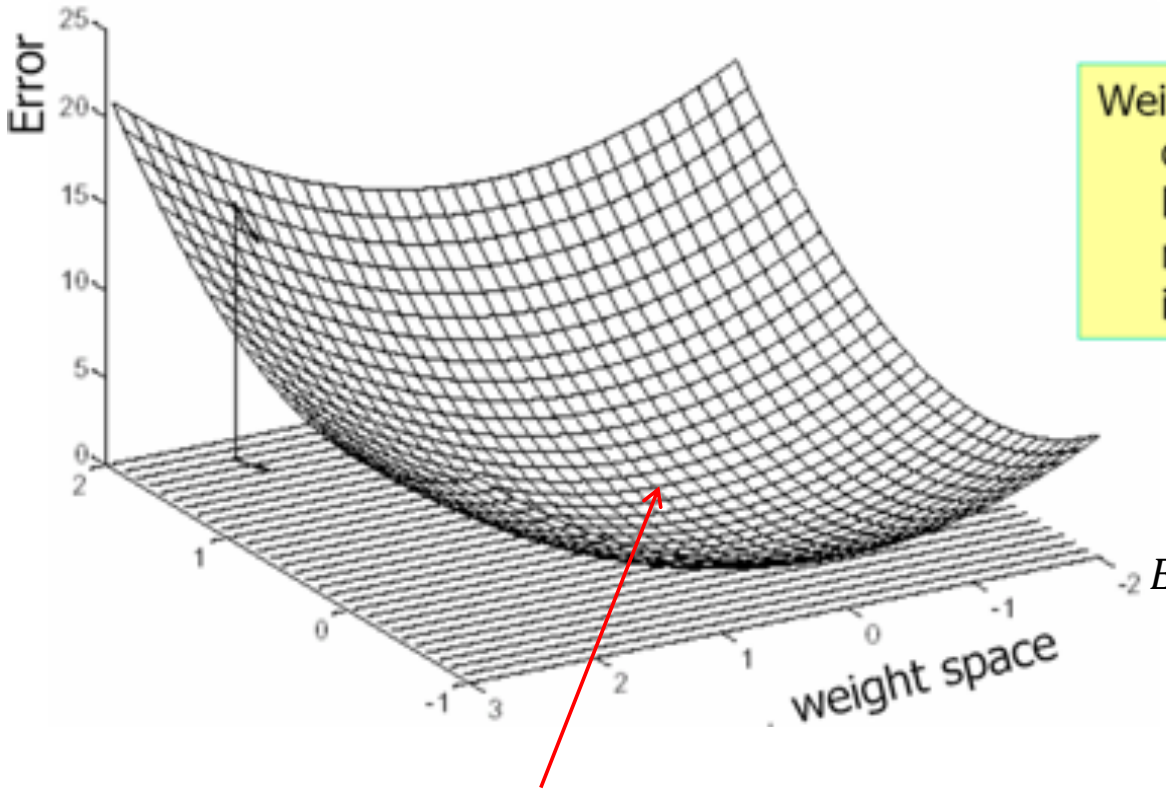
- $$E(\boldsymbol{x}) = \frac{1}{2} Err^2 = \frac{1}{2}\left(f(\boldsymbol{x}) - h(\boldsymbol{x})\right)^2 = \frac{1}{2}\left(f(\boldsymbol{x}) - g\left(\sum_{j=0}^{n} W_j \cdot x_j\right)\right)^2$$

- For evaluating the performance of a network, we can try the network on a set of datapoints and average the value
  (= sum of squared errors)

$$E(Network) = \sum_{i=1}^{N} E(\boldsymbol{x_i})$$

Error Landscape

- The error function for one training example may be considered as a function in a multi-dimensional weight space



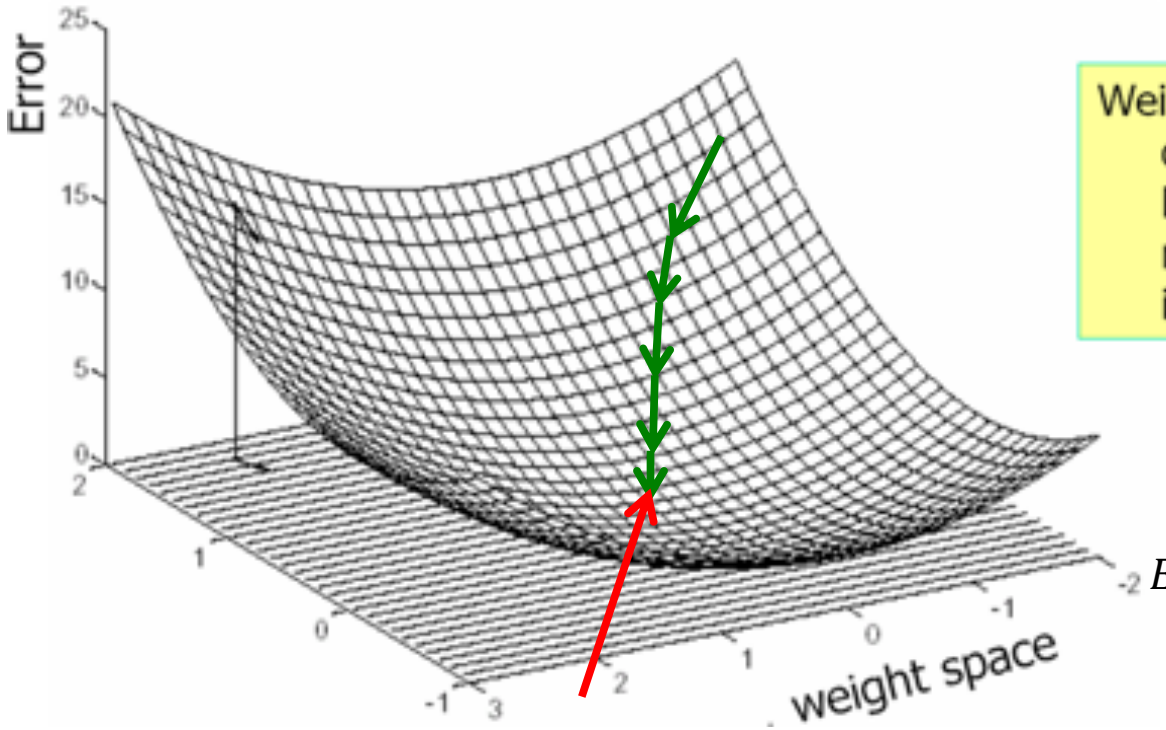
Weight space is N-dimensional, where N is the total number of weights in the network

$$E(W) = \frac{1}{2}\left(f(\boldsymbol{x}) - g\left(\sum_{j=0}^{n} W_j \cdot x_j\right)\right)^2$$

- The best weight setting for one example is where the error measure for this example is minimal

Error Minimization via Gradient Descent

- In order to find the point with the minimal error:
  - go downhill in the direction where it is steepest



Weight space is N-dimensional, where N is the total number of weights in the network

$$E(W) = \frac{1}{2}\left(f(\boldsymbol{x}) - g\left(\sum_{j=0}^{n} W_j \cdot x_j\right)\right)^2$$

  - … but make small steps, or you might shoot over the target

Error Minimization

- It is easy to derive a perceptron training algorithm that minimizes the squared error

$$E = \frac{1}{2}Err^2 = \frac{1}{2}\big(f(\boldsymbol{x}) - h(\boldsymbol{x})\big)^2 = \frac{1}{2}\left(f(\boldsymbol{x}) - g\left(\sum_{j=0}^{n} W_j \cdot x_j\right)\right)^2$$

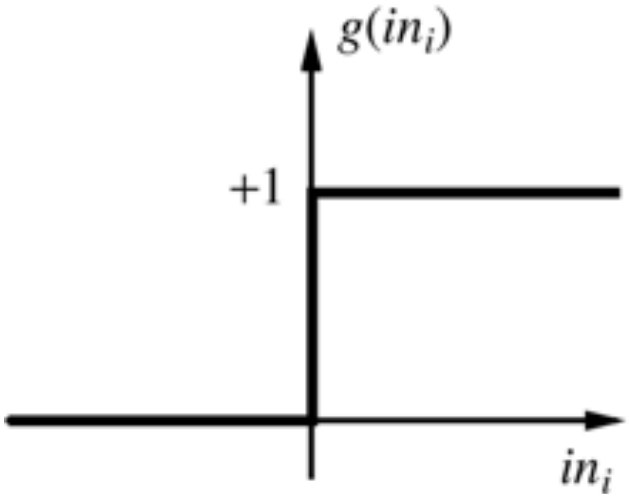
- Change weights into the direction of the steepest descent of the error function

$$\frac{\partial E}{\partial W_j} = Err \cdot \frac{\partial Err}{\partial W_j} = Err \cdot \frac{\partial}{\partial W_j}\left(f(\boldsymbol{x}) - g\left(\sum_{k=0}^{n} W_k \cdot x_k\right)\right) = -Err \cdot g'(in) \cdot x_j$$

  - To compute this, we need a continuous and differentiable activation function g!
- Weight update with learning rate α: $W_j \leftarrow W_j + \alpha \cdot Err \cdot g'(in) \cdot x_j$
  - positive error → increase network output
    - increase weights of nodes with positive input
    - decrease weights of nodes with negative input

Threshold Activation Function

- The regular threshold activation function is problematic
  - $g'(x) = 0$, therefore $\frac{\partial E}{\partial W_{j,i}} = -Err \cdot g'(in_i) \cdot x_j = 0$
  - $\rightarrow$ no weight changes!

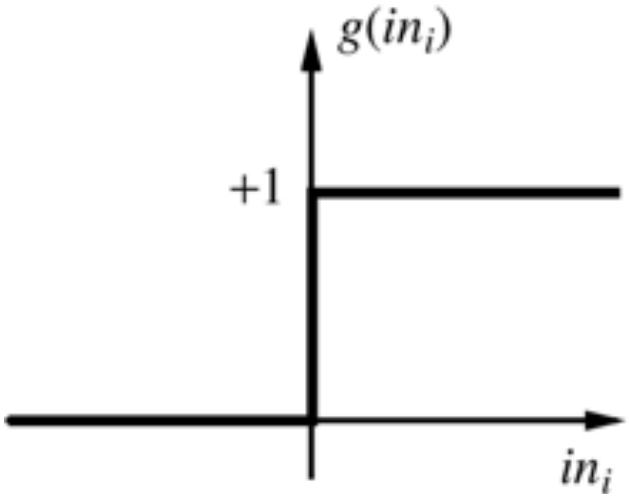


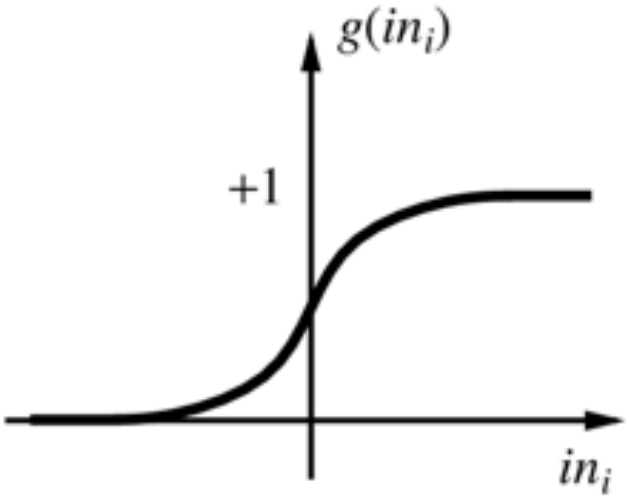
$$g(x) = \begin{cases} 0 & if x \leq 0 \\ 1 & if x > 0 \end{cases}$$

$$g'(x) = 0$$

Sigmoid Activation Function

- A commonly used activation function is the sigmoid function
  - similar to the threshold function
  - easy to differentiate
  - non-linear





$$g(x) = \begin{cases} 0 & if x \leq 0 \\ 1 & if x > 0 \end{cases}$$

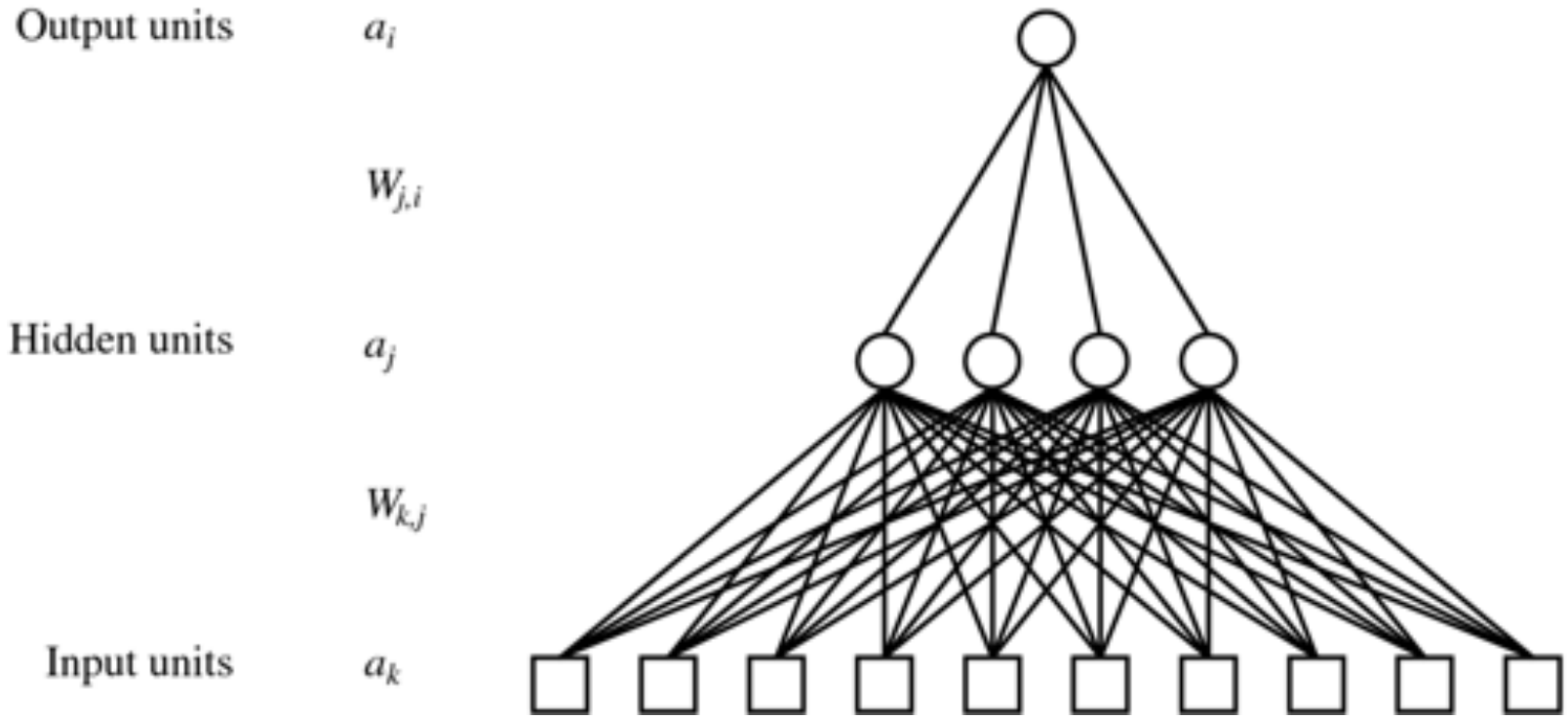
$$g'(x) = 0$$

$$g(x) = \frac{1}{1 + e^{-x}}$$

$$g'(x) = g(x)(1 - g(x))$$

Multilayer Perceptrons

- Perceptrons may have multiple output nodes
  - may be viewed as multiple parallel perceptrons
- The output nodes may be combined with another perceptron
  - which may also have multiple output nodes
- The size of this hidden layer is determined manually

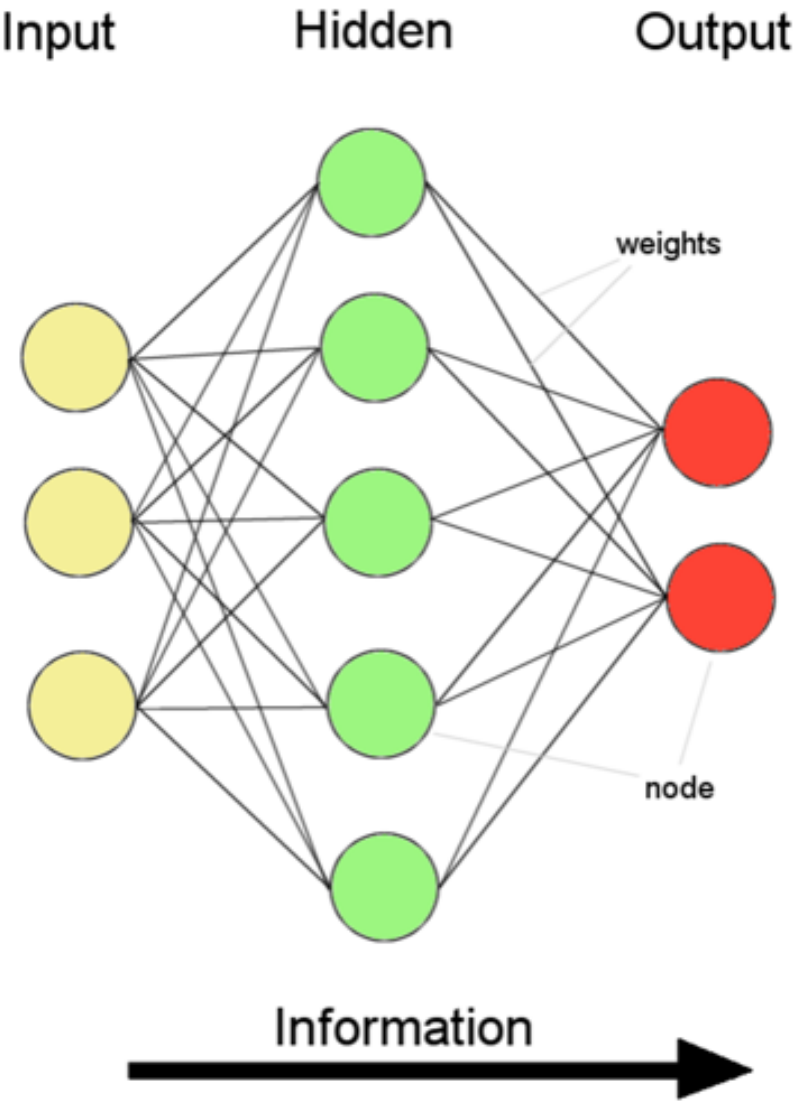


Multilayer Perceptrons

- Information flow is unidirectional
- Data is presented to *Input layer*
- Passed on to *Hidden Layer*
- Passed on to *Output layer*

- Information is distributed

- Information processing is parallel



Expressiveness of MLPs

- Every continuous function can be modeled with three layers
  - i.e., with one hidden layer
- Every function can be modeled with four layers
  - i.e., with two hidden layers

Backpropagation Learning

- The output nodes are trained like a normal perceptron

$$W_{ji} \leftarrow W_{ji} + \alpha \cdot Err_i \cdot g'(in_i) \cdot x_j = W_{ji} + \alpha \cdot \Delta_i \cdot x_j$$

  - Δ_i is the error term of output node i times the derivation of its inputs

- the error term Δ_i of the output layers is propagated back to the hidden layer

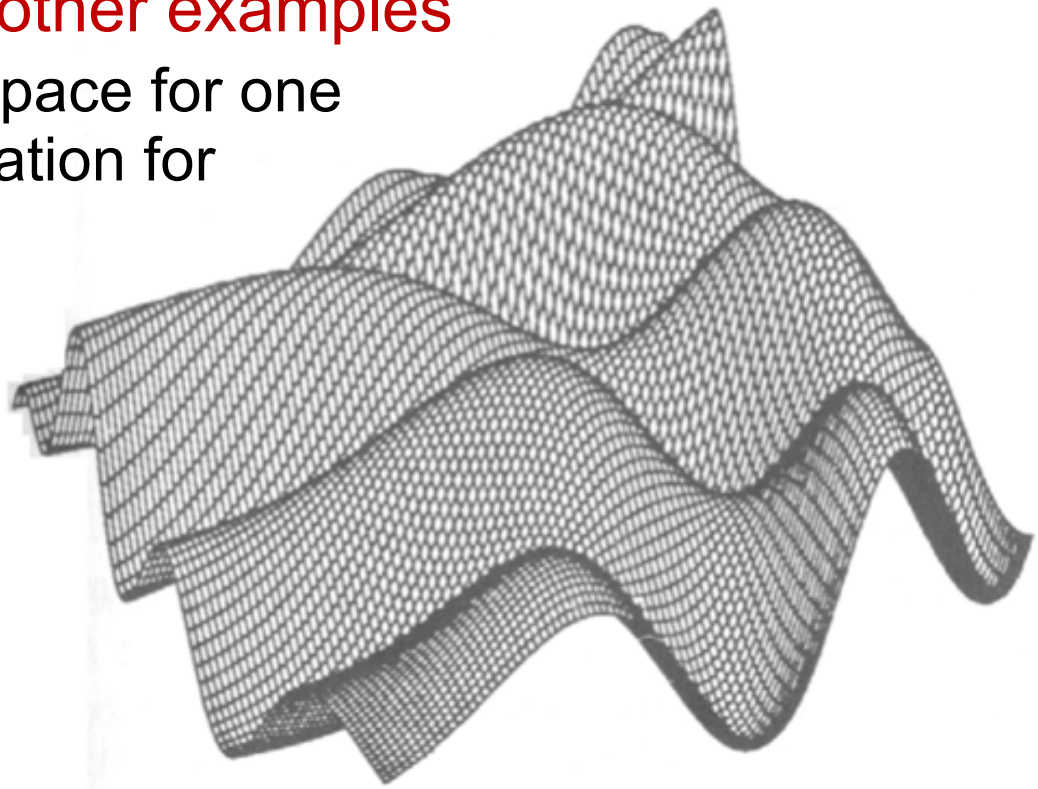
$$\Delta_j = (\sum_i W_{ji} \cdot \Delta_i) \cdot g'(in_j) \qquad W_{kj} \leftarrow W_{kj} + \alpha \cdot \Delta_j \cdot x_k$$

  - the training signal of hidden layer node j is the weighted sum of the errors of the output nodes

- Thus the information provided by the gradient flows backwards through the network

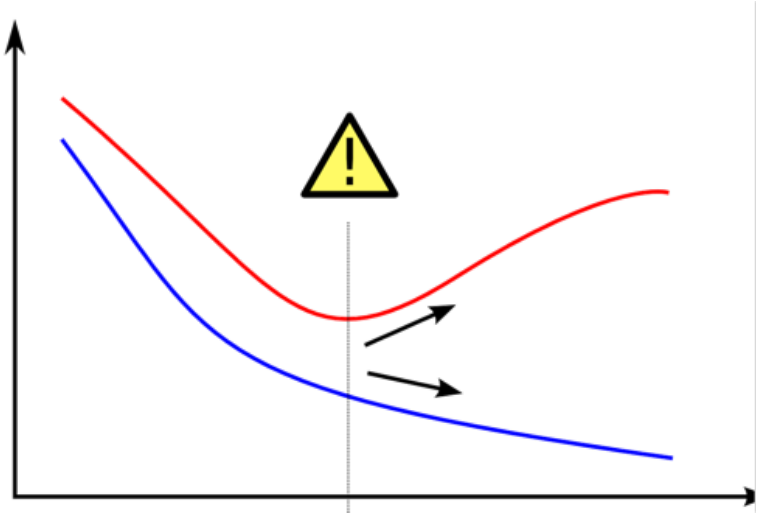
Minimizing the Network Error

- The error landscape for the entire network may be thought of as the sum of the error functions of all examples
  - will yield many local minima → hard to find global minimum
- Minimizing the error for one training example may destroy what has been learned for other examples
  - a good location in weight space for one example may be a bad location for another examples
- Training procedure:
  - try all examples in turn
  - make small adjustments for each example
  - repeat until convergence
- One Epoch = One iteration through all examples



Overfitting

- Training Set Error continues to decrease with increasing number of training examples / number of epochs
  - an epoch is a complete pass through all training examples
- Test Set Error will start to increase because of overfitting



- Simple training protocol:
  - keep a separate validation set to watch the performance
    - validation set is different from training and test sets!
  - stop training if error on validation set gets down

Deep Learning

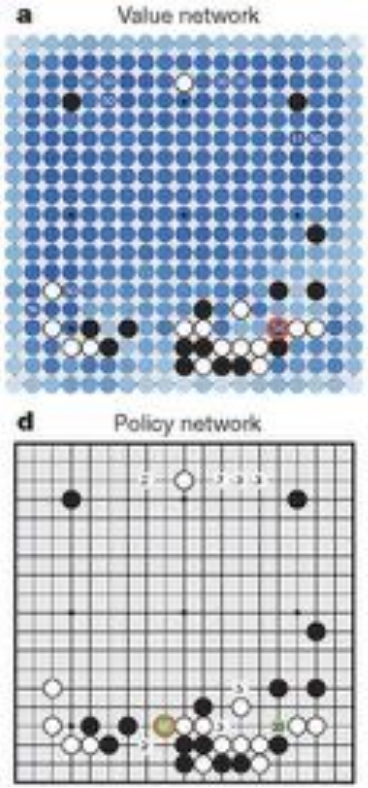
- In the last years, great success has been observed with training „deep“ neural networks
  - Deep networks are networks with multiple layers

- Key ingredients:
  - A lot of training data are needed and available (big data)
  - Fast processing and a few new tricks made fast training for big data possible

Watch NATURE video at https://www.youtube.com/watch?v=g-dKXOlsf98

DeepMind's AlphaGo

DeepMind's AlphaGo



Deep policy network is trained to produce probability map of promising moves. The deep value network is used to prune the (mcmc) search tree

Goal of Deep Architectures

High-level semenatical representations

To this aim most approaches use (stacked) neural networks

Edges, local shapes, object parts

Low level representation

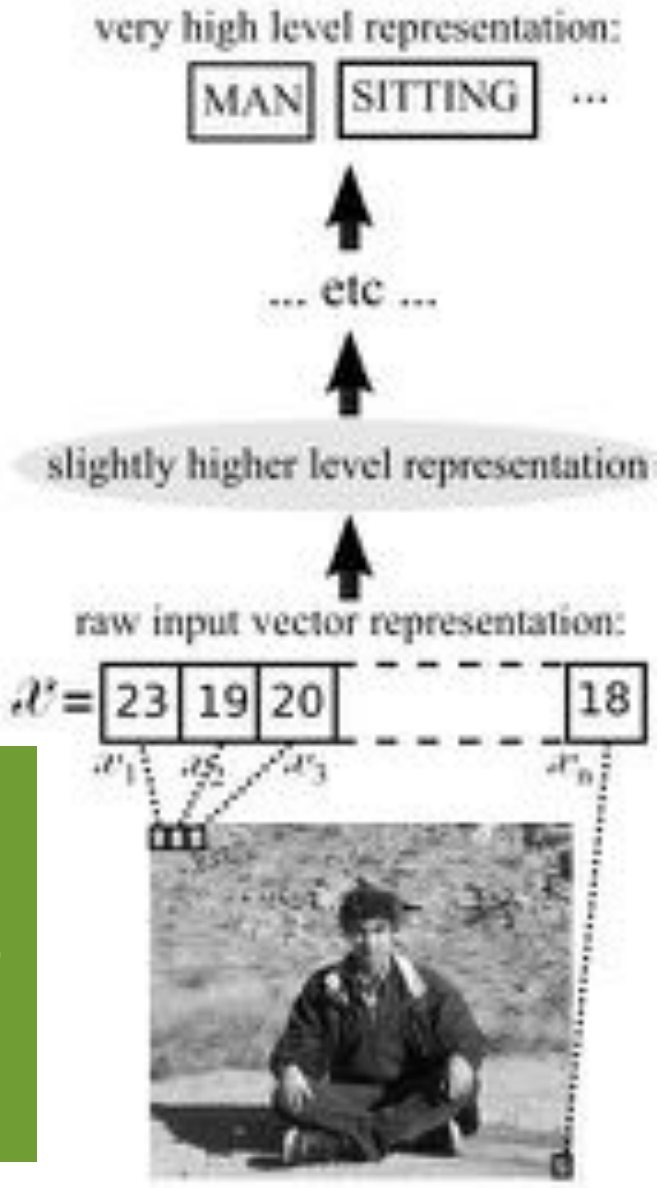


Figure is from Yoshua Bengio

Deep learning methods aim at

- learning feature hierarchies
- where features from higher levels of the hierarchy are formed by lower level features.

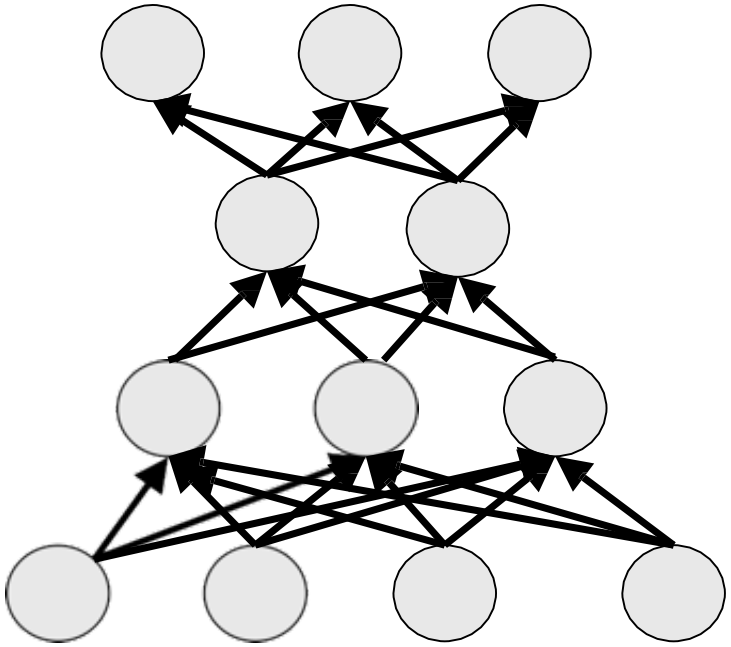
Deep Architectures

Deep architectures are composed of multiple levels of non-linear operations, such as neural nets with many hidden layers.

Output layer

Hidden layers

Input layer



Examples of non-linear activations:

$$\tanh(x)$$

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

$$\max(0, x)$$

In practice, NN with multiple hidden layers work better than with a single hidden layer.

(Deep) Convolutional Networks CNNs

Compared to standard neural networks with similarly-sized layers,

- CNNs have much fewer connections and parameters
- and so they are easier to train
- and typically have more than five layers (a number of layers which makes fully-connected neural networks almost impossible to train properly when initialized randomly)

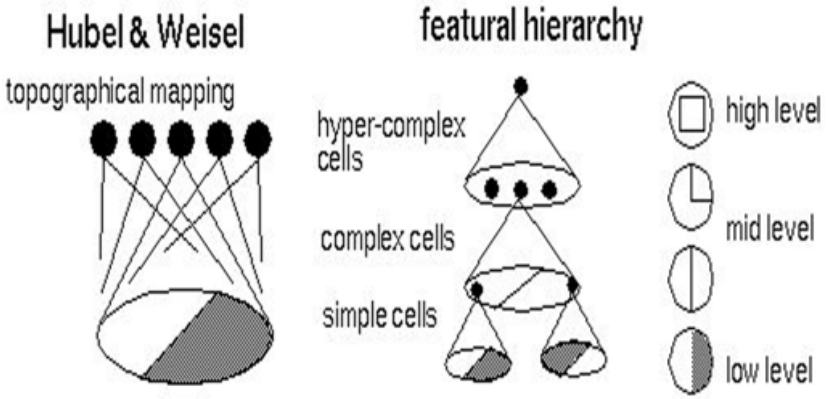
LeNet, 1998 LeCun Y, Bottou L, Bengio Y, Haffner P: Gradient-Based Learning Applied to Document Recognition, Proceedings of the IEEE

AlexNet, 2012 Krizhevsky A, Sutskever I, Hinton G: ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012

Cortical Receptive Fields

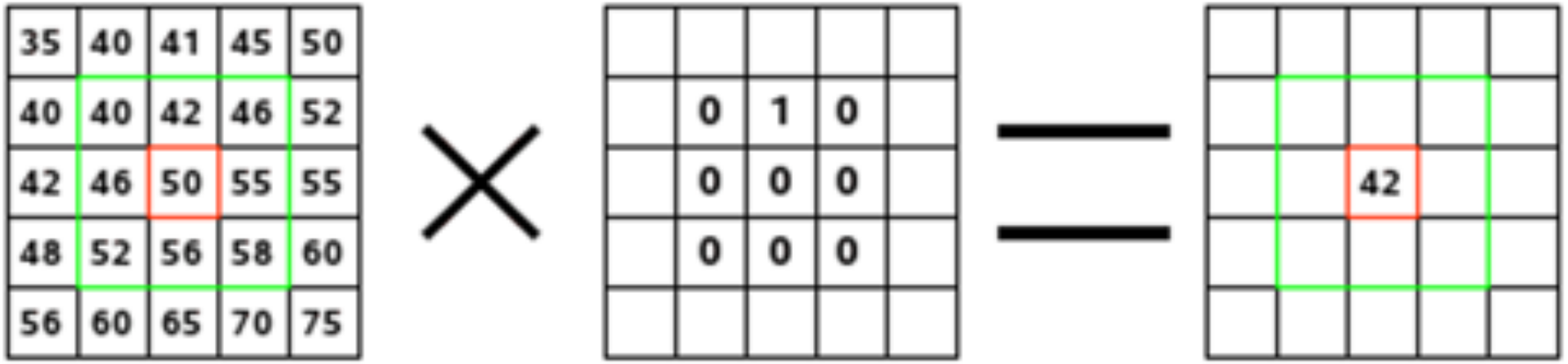
David Hubel & Thorston Wiesel, Nobel Prize 1981

https://www.youtube.com/watch?v=IOHayh06LJ4



Convolutional Neural Networks

- Convolution:
  - for each pixel of an image, a new feature is computed using a weighted combination of its nxn neighborhood



5x5 image

3x3 convolution runs over all possible 3x3 subimages of picture

resulting image only one pixel shown

Convolution - Blur

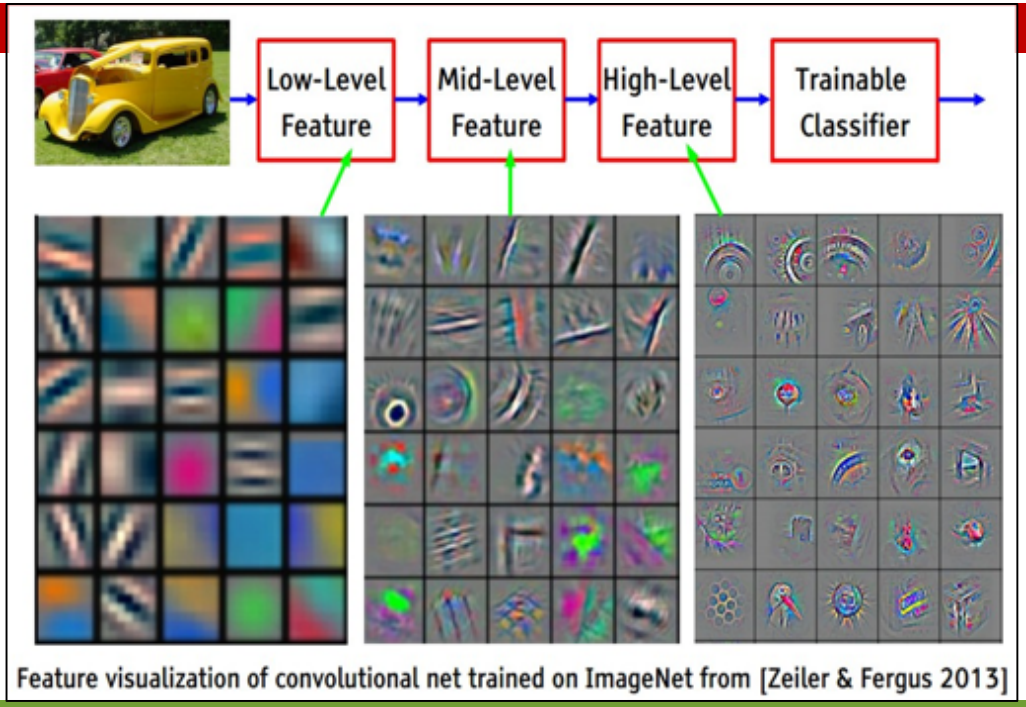
0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0

Convolution - Edge detection

	0	1	0	
	1	-4	1	
	0	1	0	

Outputs of Convolution

Outputs of Convolution



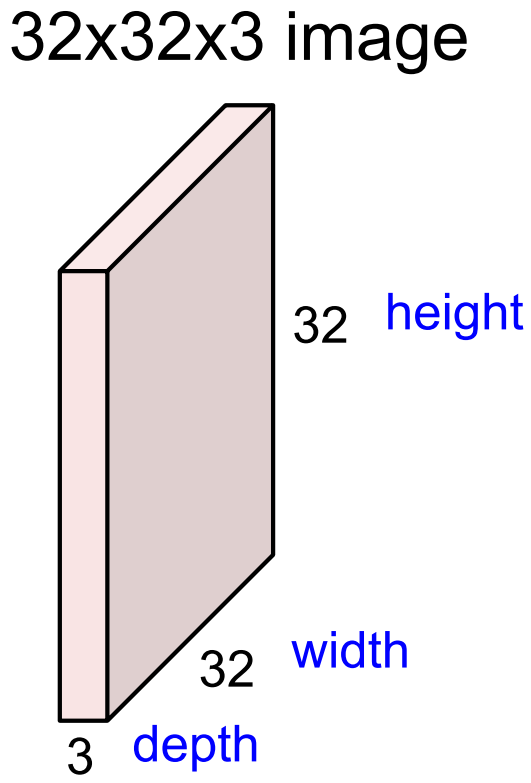
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Deep Convolutional Networks

The first breakthrough of DCNs was on image classification

- Convolutional layer
- Non-linear activation function ReLU
- Max pooling layer
- Fully connected layer

Convolutional layer

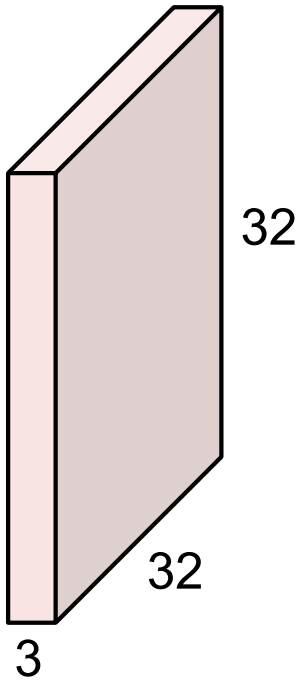


Filter try to detect local patterns such as color, edges, …

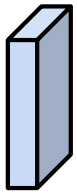
Convolutional layer

Filters always extend the full depth of the input volume

32x32x3 image



5x5x3 filter

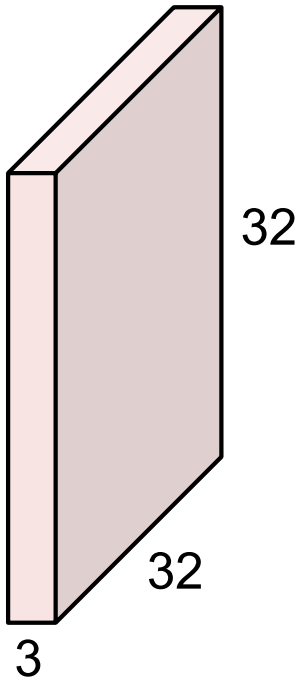


Filter try to detect local patterns such as color, edges, …

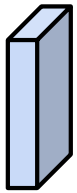
Convolutional layer

Filters always extend the full depth of the input volume

32x32x3 image



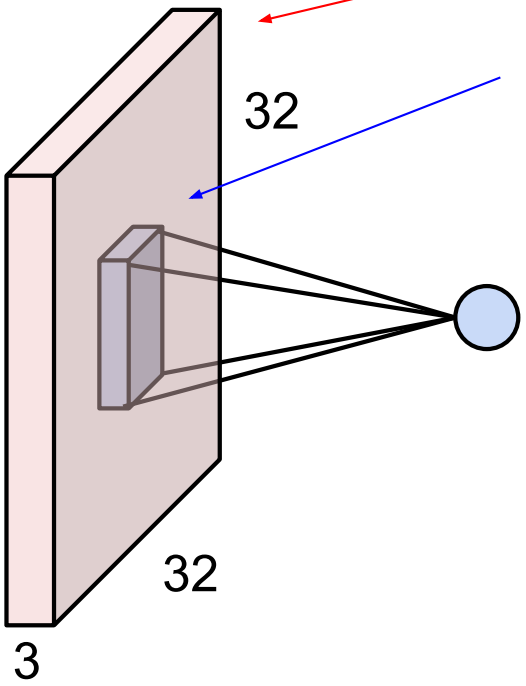
5x5x3 filter



Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Filter try to detect local patterns such as color, edges, …

Convolutional layer



32x32x3 image

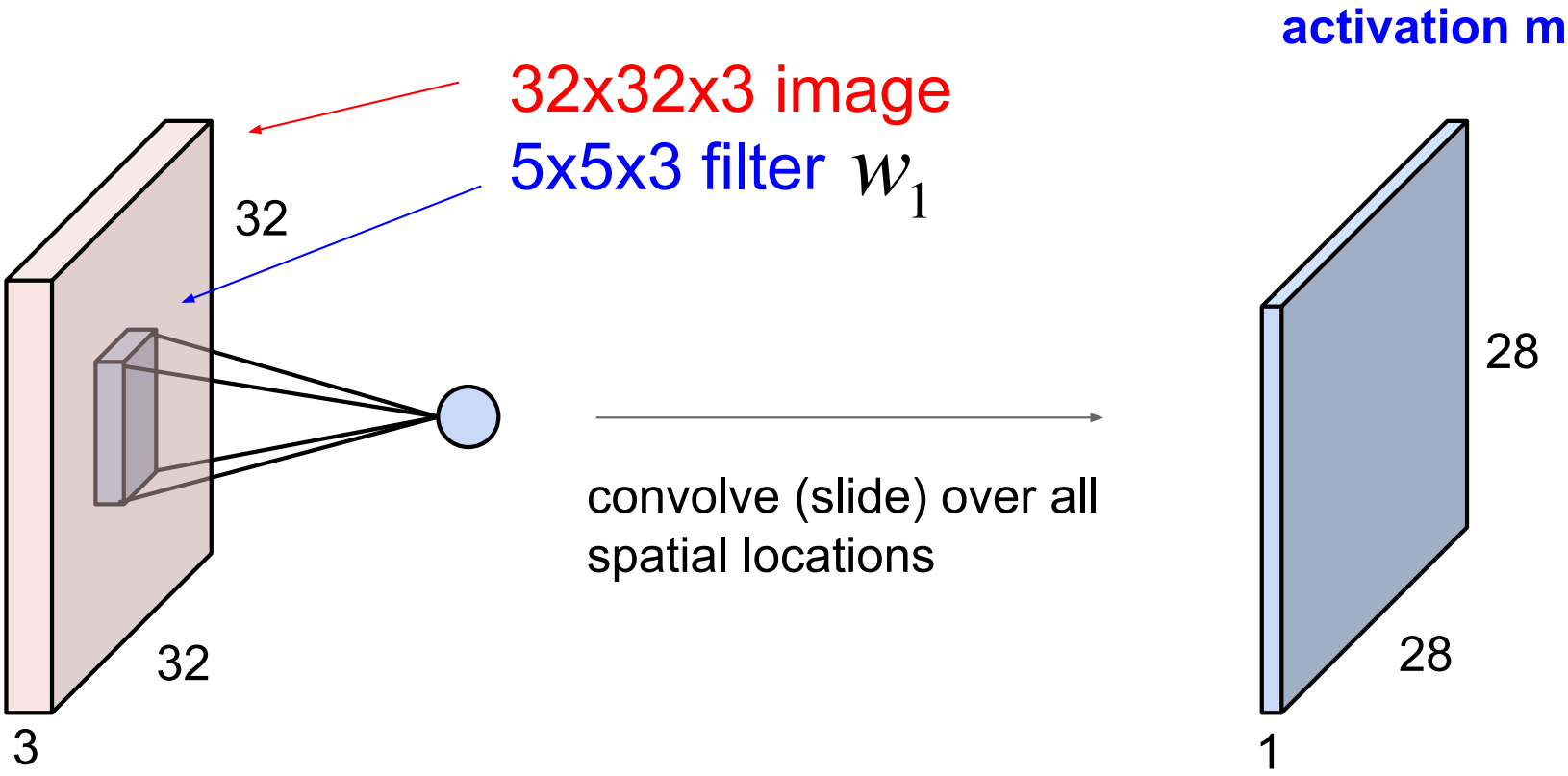
5x5x3 filter w

1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image x

$w^T x$ (in general, $w^T x$ + bias)

Filter try to detect local patterns such as color, edges, …

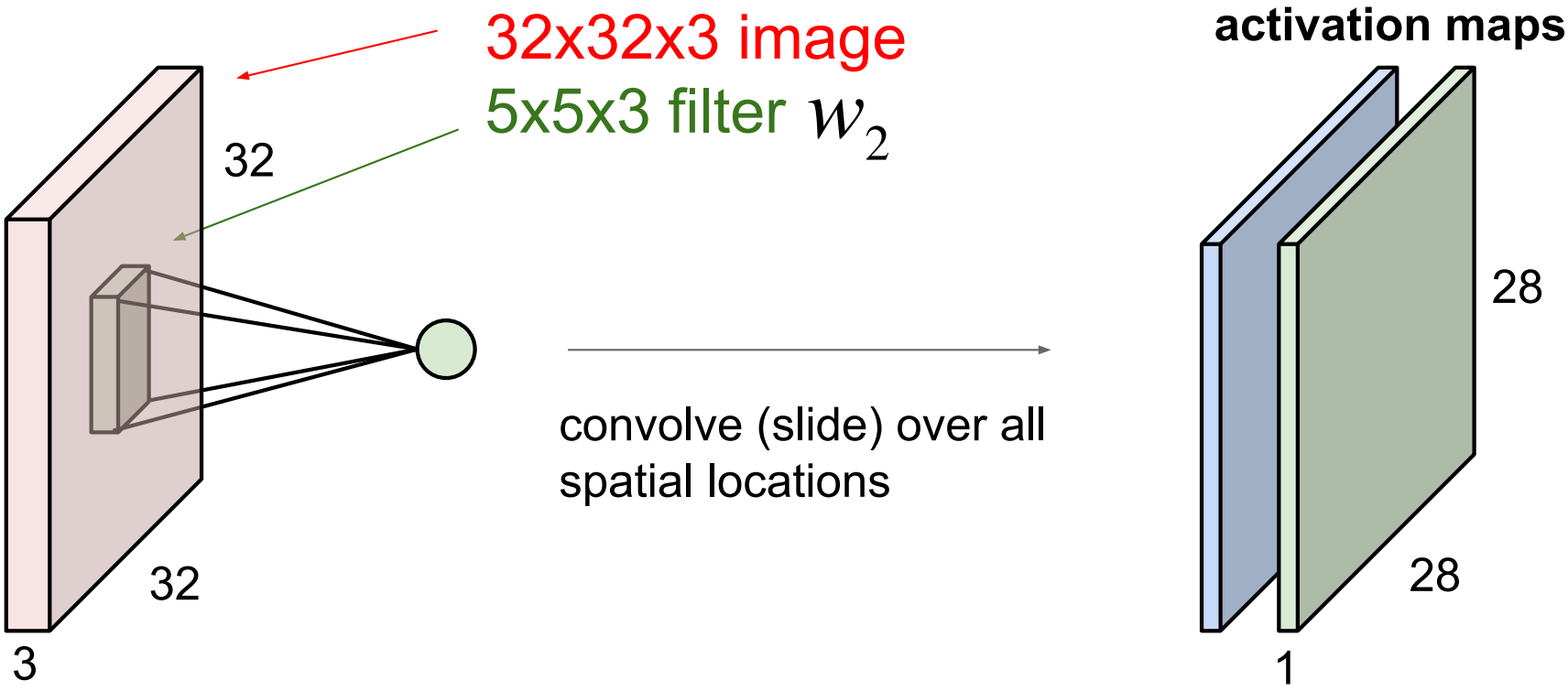
Convolutional layer



Filter try to detect local patterns such as color, edges, …

Convolutional layer

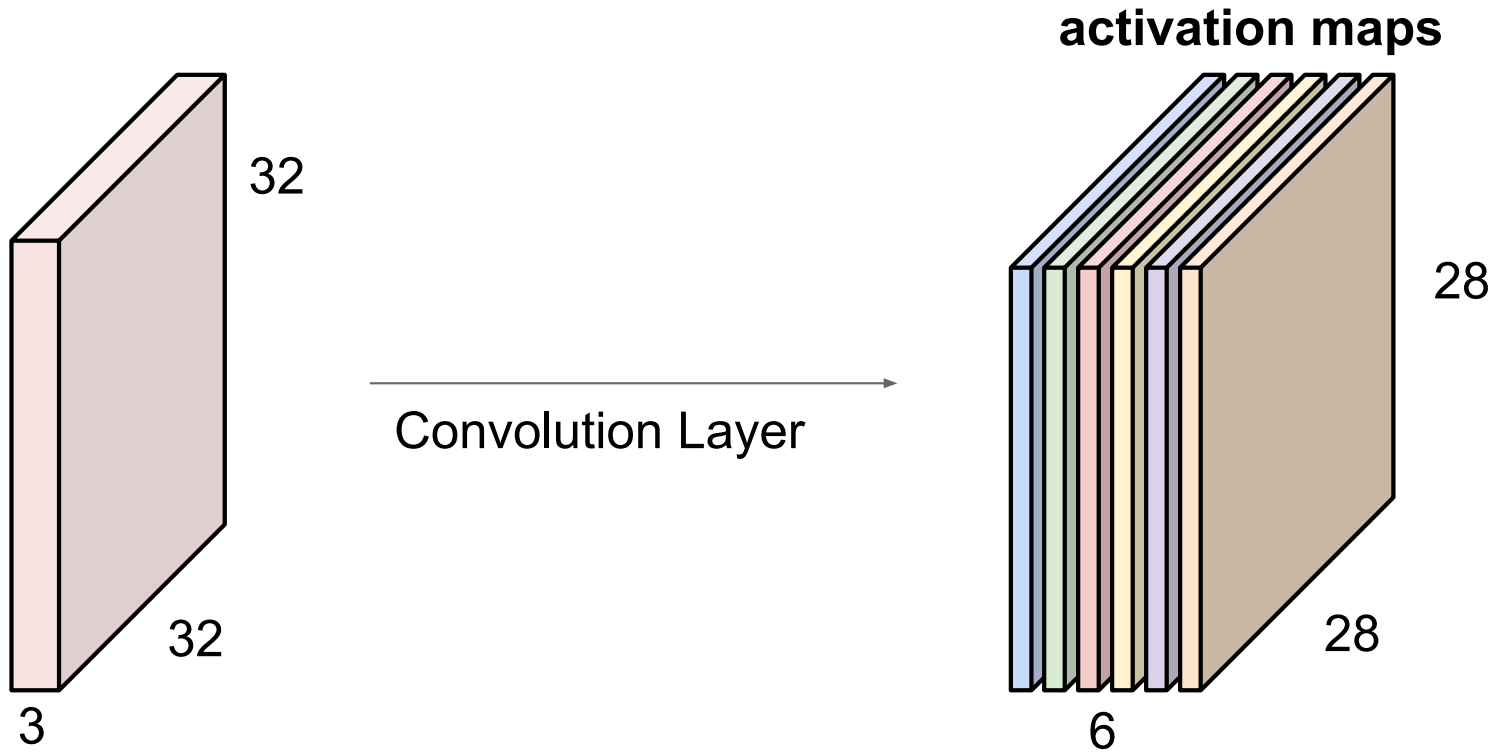
consider a second, green filter



Filter try to detect local patterns such as color, edges, …

Convolutional layer

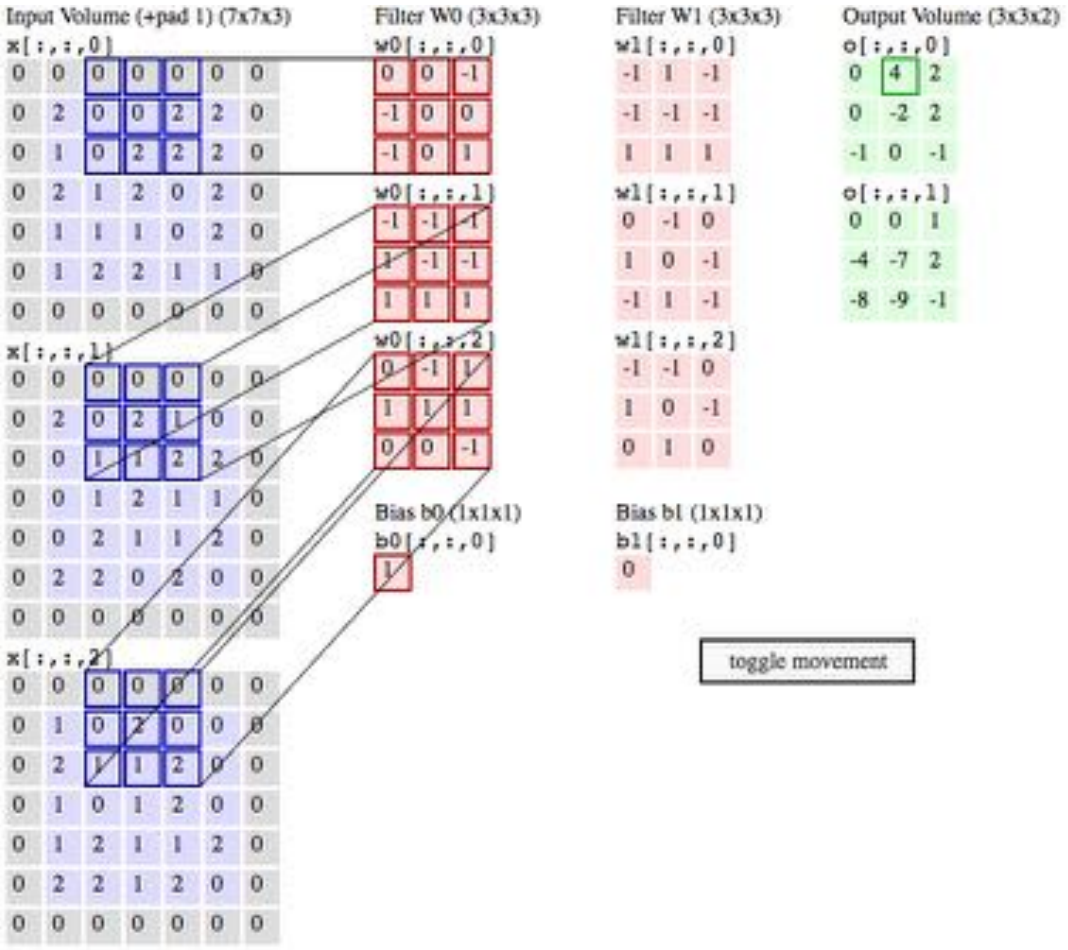
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a "new image" of size 28x28x6!

Filter try to detect local patterns such as color, edges, …

See this in action

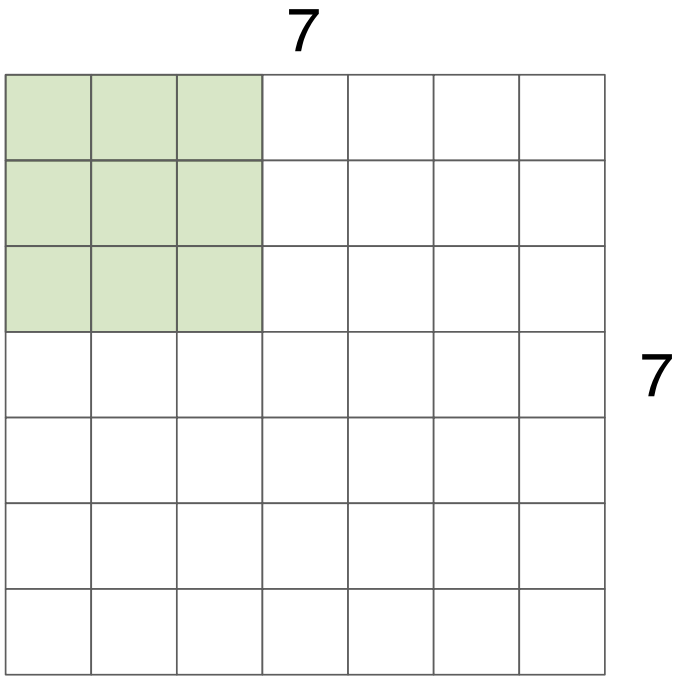


http://cs231n.github.io/assets/conv-demo/index.html

What is the complexity of doing these local detections?

Spatial dimensions

A closer look at spatial dimensions

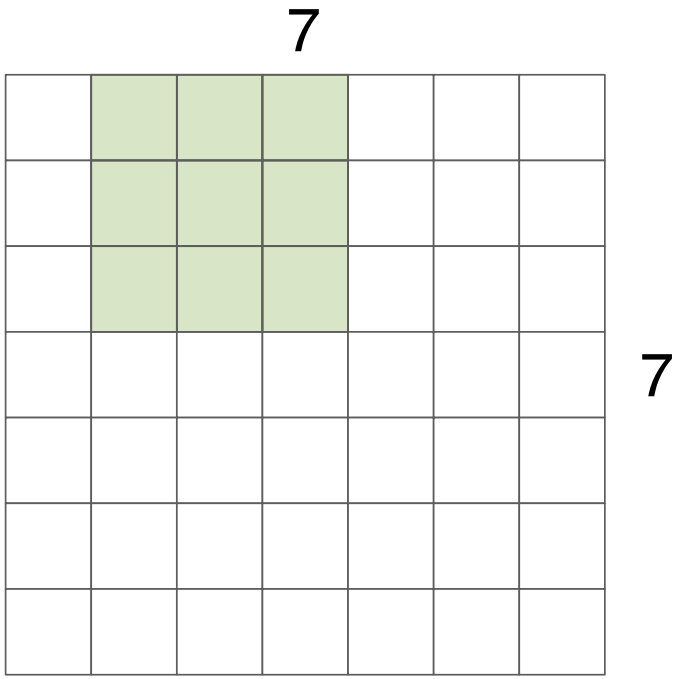


7x7x1 image

3x3x1 filter w

Spatial dimensions

A closer look at spatial dimensions



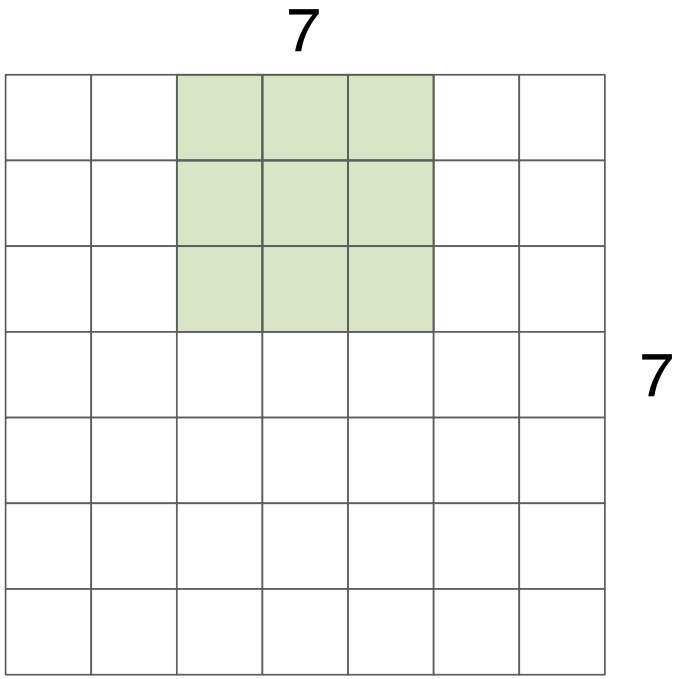
7x7x1 image

3x3x1 filter w

Slide over all locations using **stride 1** horizontally and vertically, S=1

Spatial dimensions

A closer look at spatial dimensions



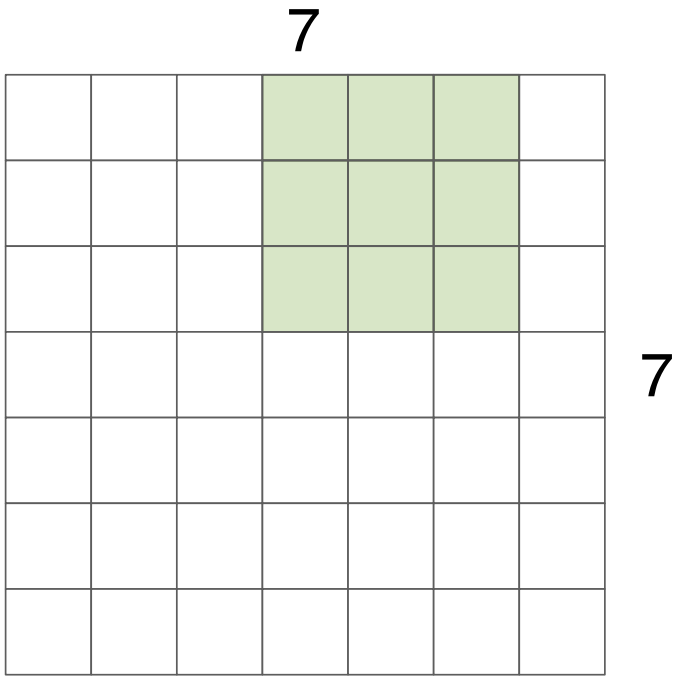
7x7x1 image

3x3x1 filter w

Slide over all locations using **stride 1** horizontally and vertically, S=1

Spatial dimensions

A closer look at spatial dimensions



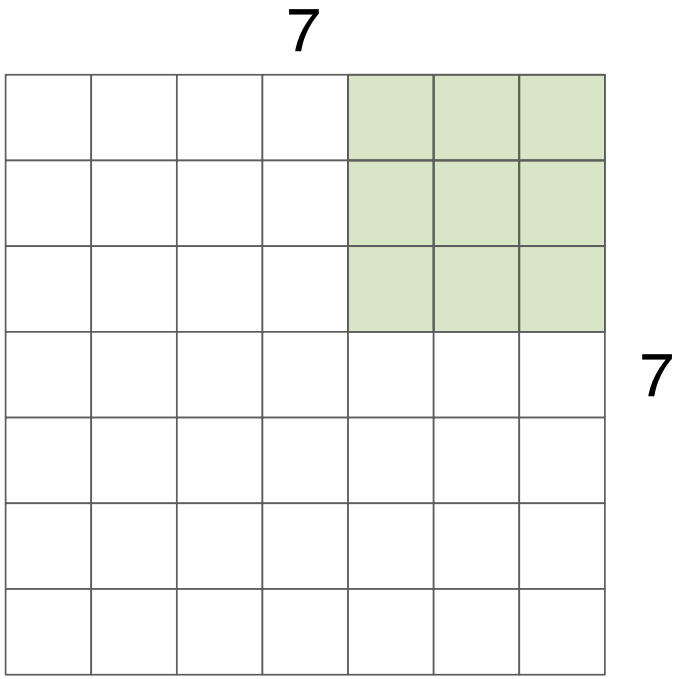
7x7x1 image

3x3x1 filter w

Slide over all locations using **stride 1** horizontally and vertically, S=1

Spatial dimensions

A closer look at spatial dimensions



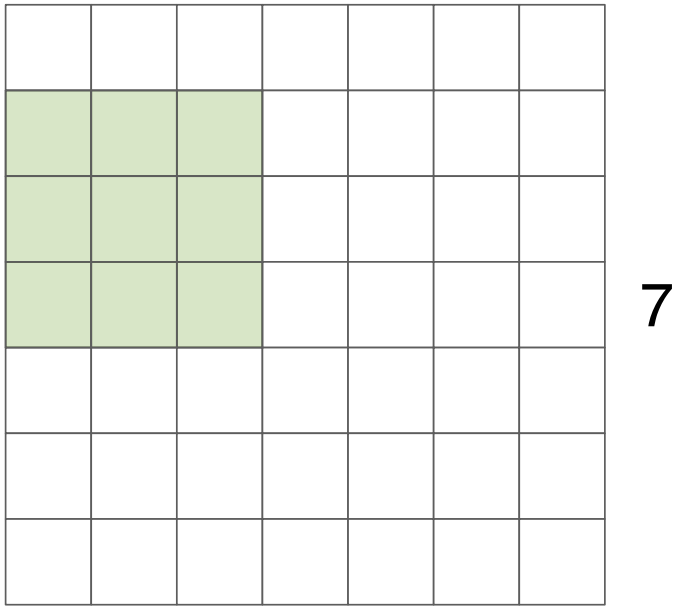
7x7x1 image

3x3x1 filter w

Slide over all locations using **stride 1** horizontally and vertically, S=1

Spatial dimensions

A closer look at spatial dimensions



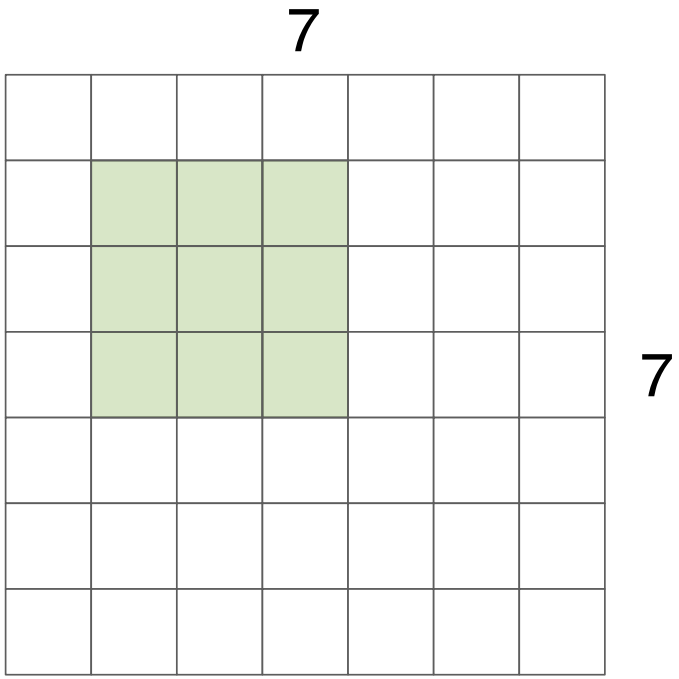
7x7x1 image

3x3x1 filter w

Slide over all locations using **stride 1** horizontally and vertically, S=1

Spatial dimensions

A closer look at spatial dimensions



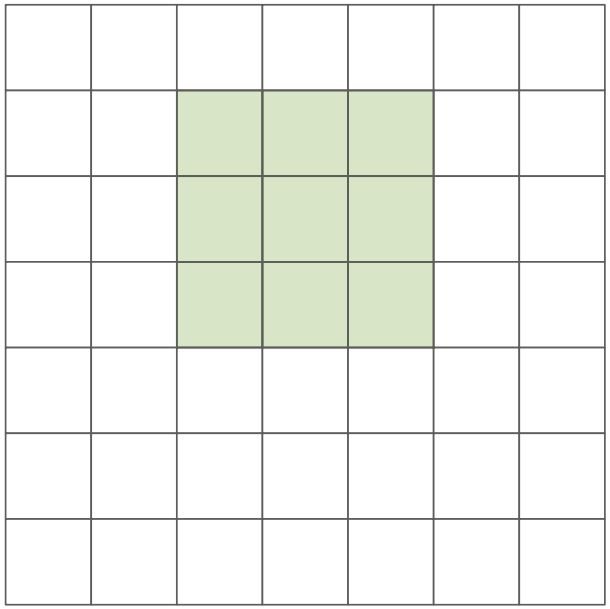
7x7x1 image

3x3x1 filter w

Slide over all locations using **stride 1** horizontally and vertically, S=1

Spatial dimensions

A closer look at spatial dimensions



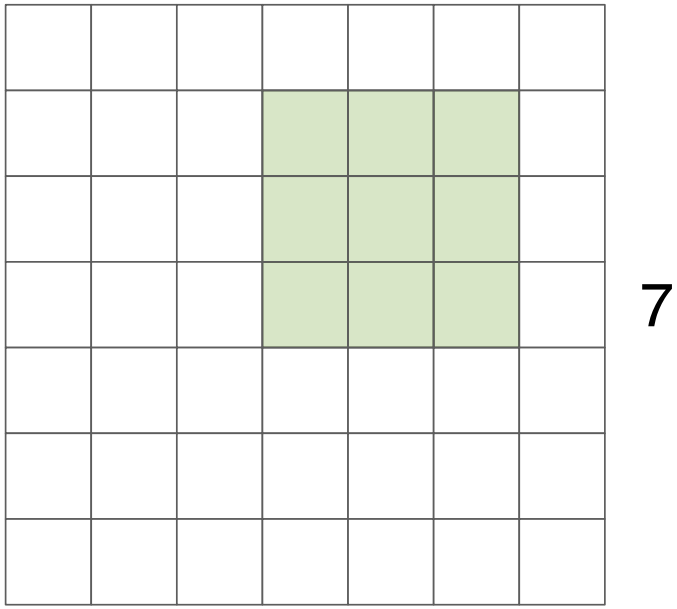
7x7x1 image

3x3x1 filter w

Slide over all locations using **stride 1** horizontally and vertically, S=1

Spatial dimensions

A closer look at spatial dimensions



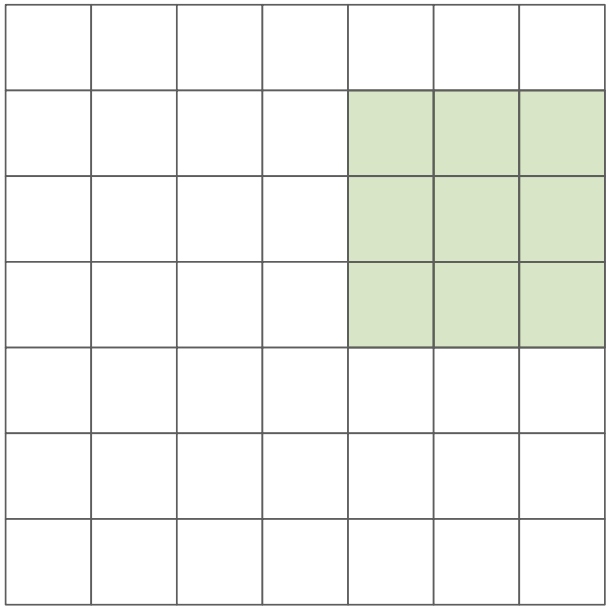
7x7x1 image

3x3x1 filter w

Slide over all locations using **stride 1** horizontally and vertically, S=1

Spatial dimensions

A closer look at spatial dimensions



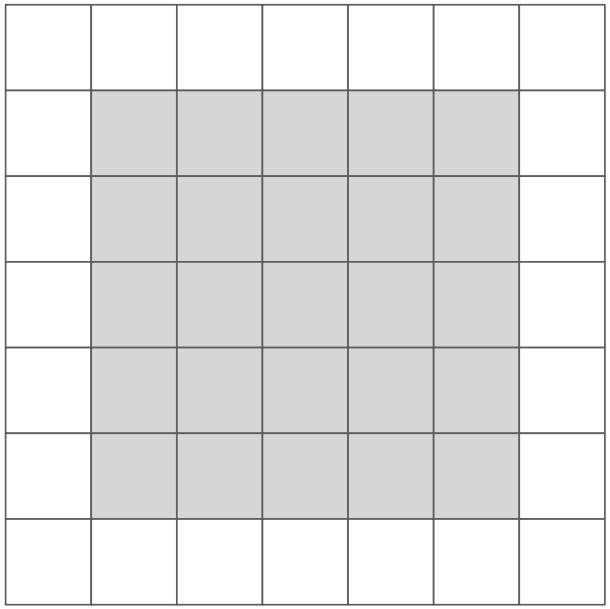
7x7x1 image

3x3x1 filter w

and so on …

Spatial dimensions

A closer look at spatial dimensions



7x7x1 image

3x3x1 filter w

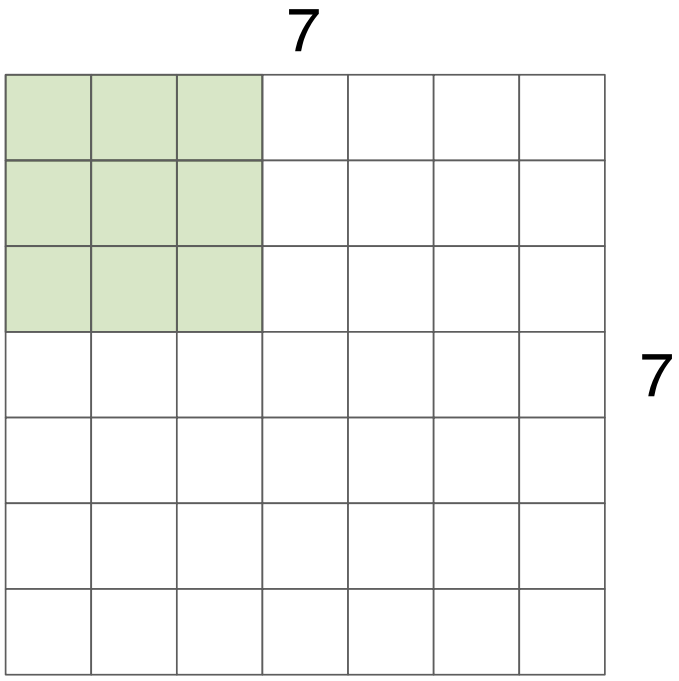
stride S=1

$\Rightarrow$ **5x5 output**

activation map

Spatial dimensions

A closer look at spatial dimensions



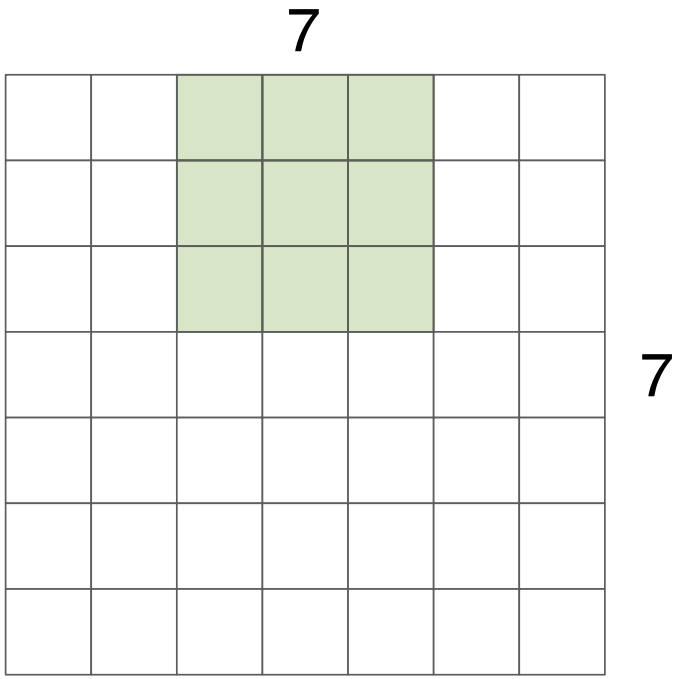
7x7x1 image

3x3x1 filter w

Slide over all locations **using stride 2** horizontally and vertically, S=2

Spatial dimensions

A closer look at spatial dimensions



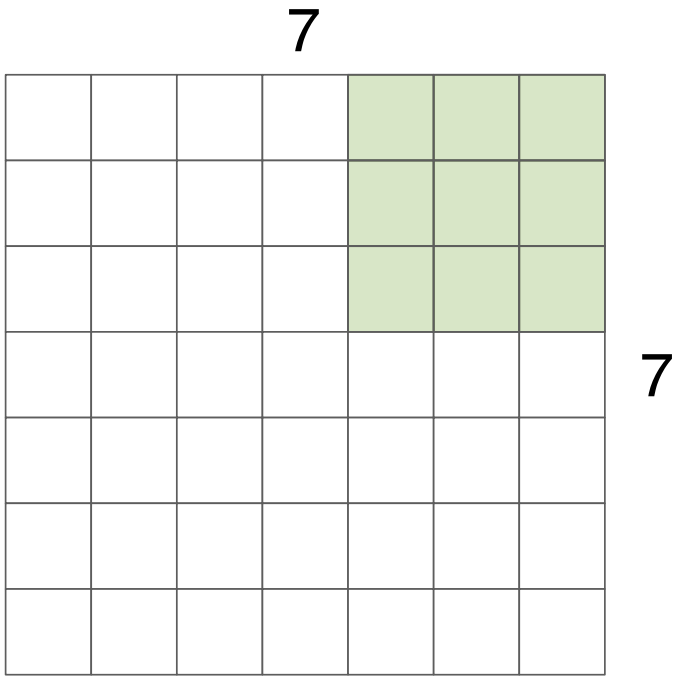
7x7x1 image

3x3x1 filter w

Slide over all locations **using stride 2** horizontally and vertically, S=2

Spatial dimensions

A closer look at spatial dimensions



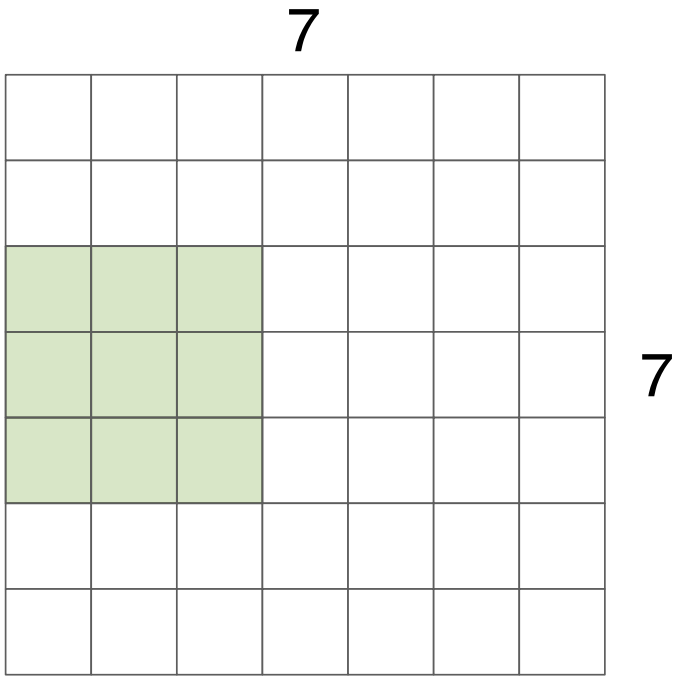
7x7x1 image

3x3x1 filter w

Slide over all locations **using stride 2** horizontally and vertically, S=2

Spatial dimensions

A closer look at spatial dimensions



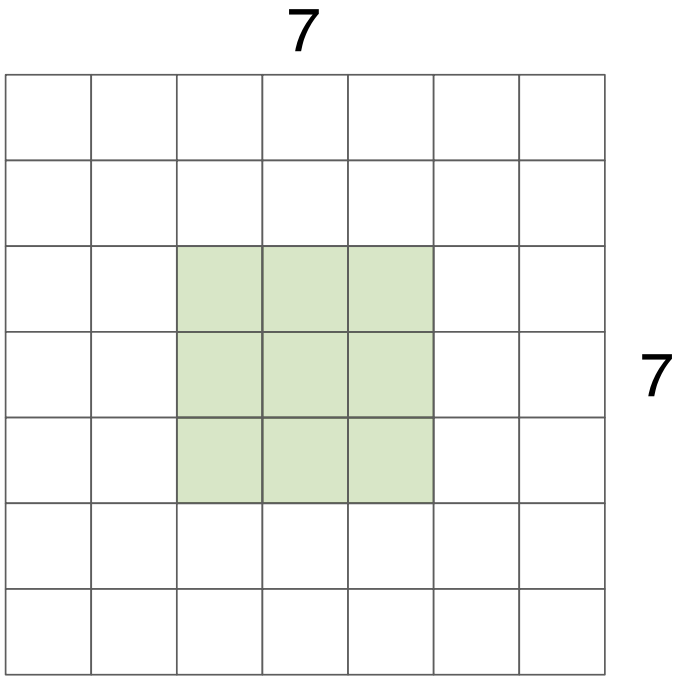
7x7x1 image

3x3x1 filter w

Slide over all locations **using stride 2** horizontally and vertically, S=2

Spatial dimensions

A closer look at spatial dimensions



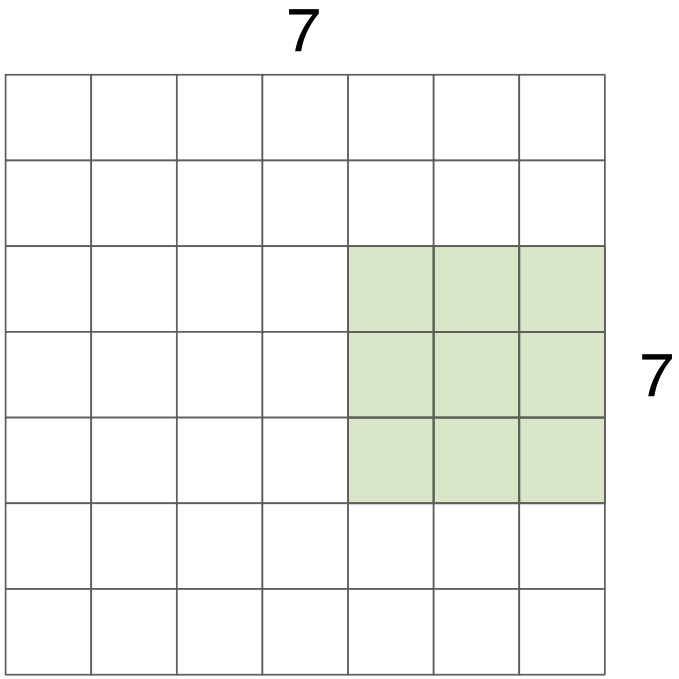
7x7x1 image

3x3x1 filter w

Slide over all locations **using stride 2** horizontally and vertically, S=2

Spatial dimensions

A closer look at spatial dimensions



7x7x1 image

3x3x1 filter w

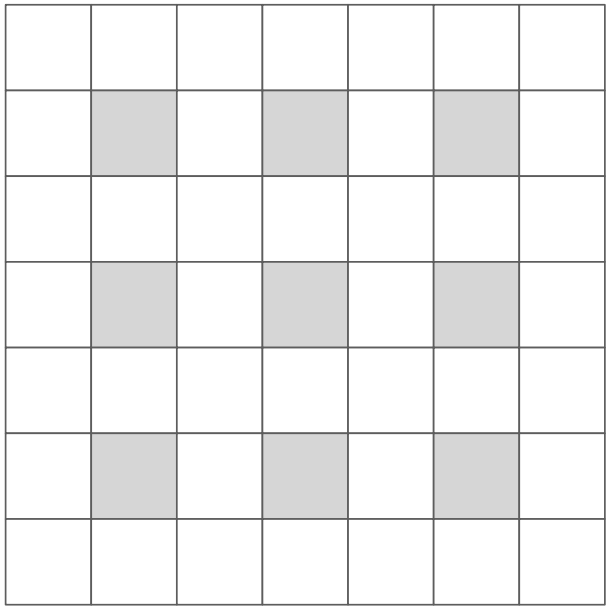
Slide over all locations **using stride 2** horizontally and vertically, S=2

...

=> ? output

Spatial dimensions

A closer look at spatial dimensions



7x7x1 image

3x3x1 filter w

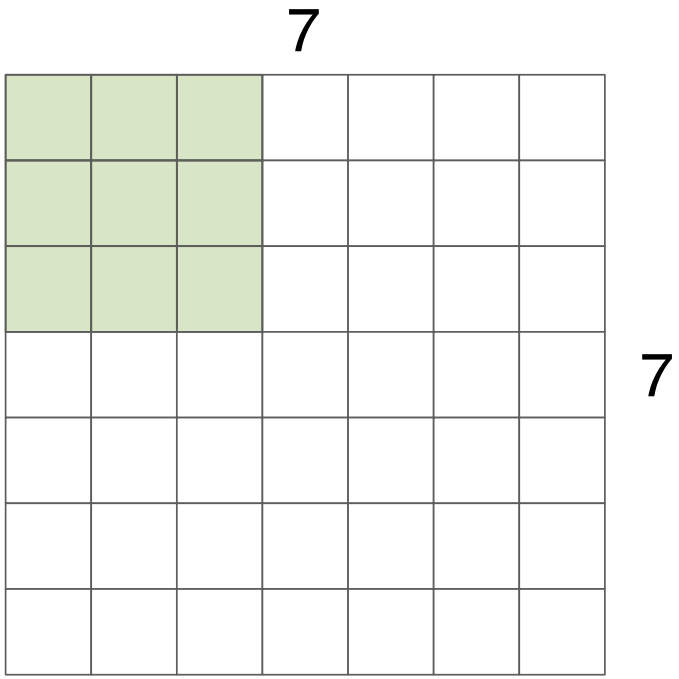
stride S=2

$\Rightarrow$ **3x3 output**

activation map

Spatial dimensions

A closer look at spatial dimensions



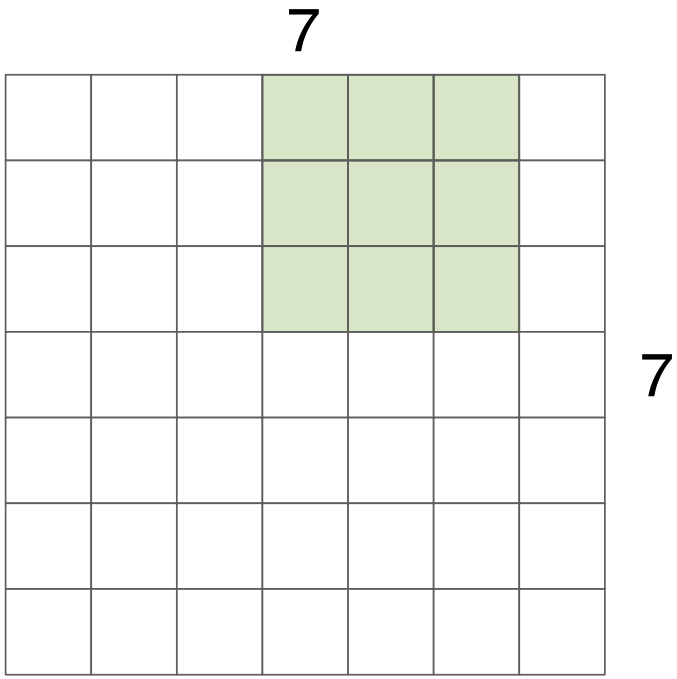
7x7x1 image

3x3x1 filter w

stride S=3

Spatial dimensions

A closer look at spatial dimensions



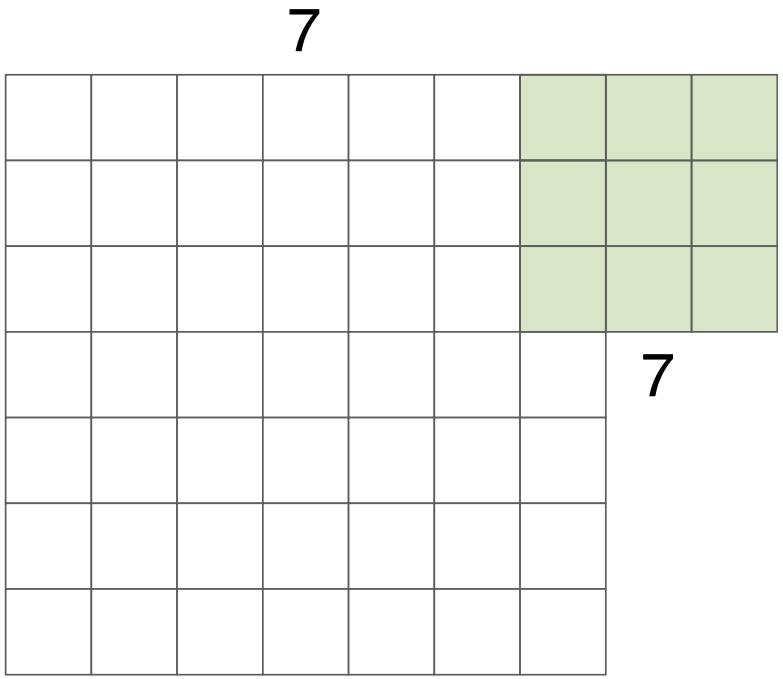
7x7x1 image

3x3x1 filter w

stride S=3

Spatial dimensions

A closer look at spatial dimensions



7x7x1 image

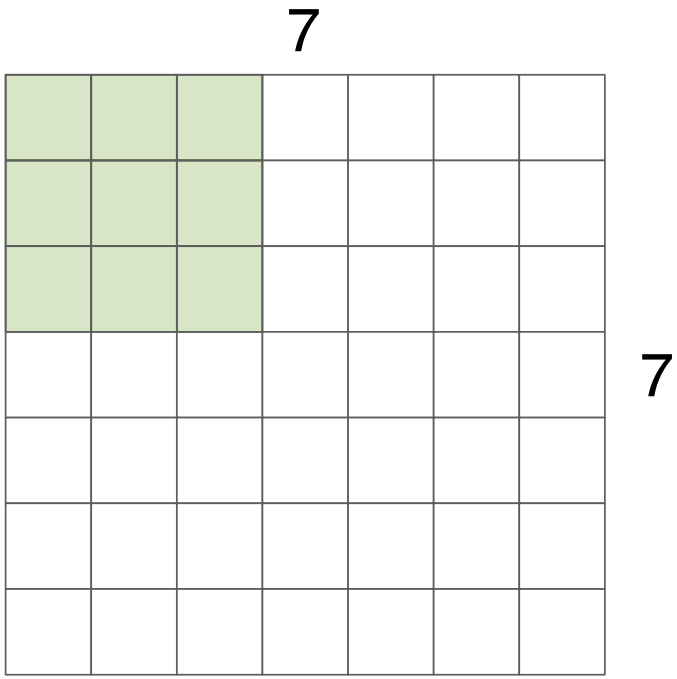
3x3x1 filter w

stride S=3

doesn't fit!
cannot apply 3x3x1 filter on 7x7x1 image with stride 3

Spatial dimensions

A closer look at spatial dimensions



7x7x1 image

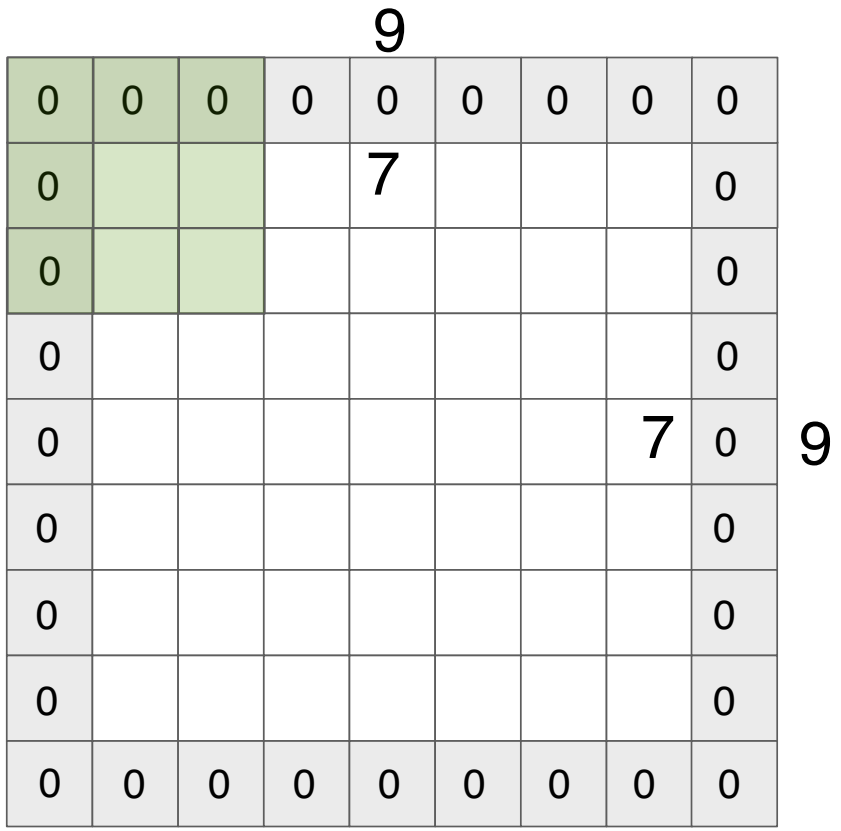
3x3x1 filter w

stride S=3

?

Spatial dimensions

Add zero padding around the border



0	0	0	0	0	0	0	0	0
0				7				0
0								0
0								0
0							7	0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

7x7x1 image

3x3x1 filter w

stride S=3

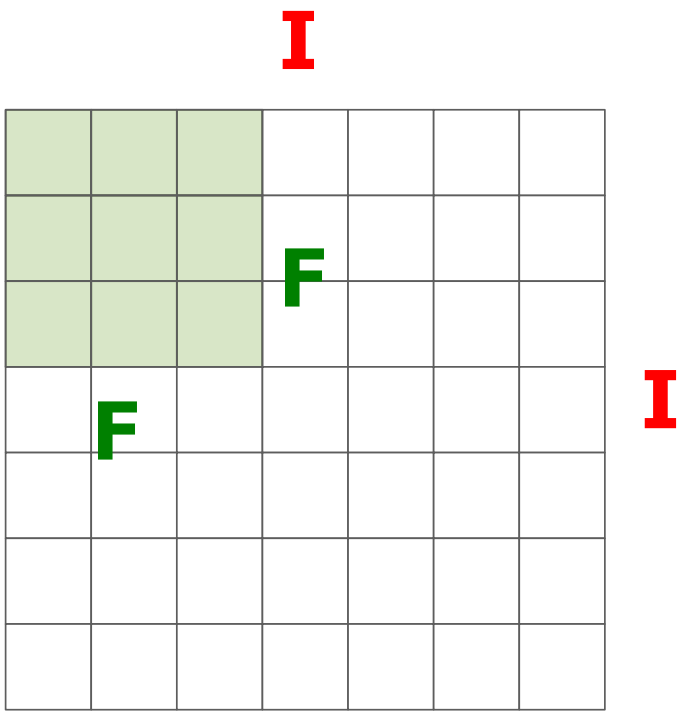
padding = 1

$\Rightarrow$ **3x3 output**

activation map

Spatial dimensions

Spatial dimension of the output $\frac{I - F + 2P}{S} + 1$



Ix**I**x**d** input

Fx**F**x**d** filter w

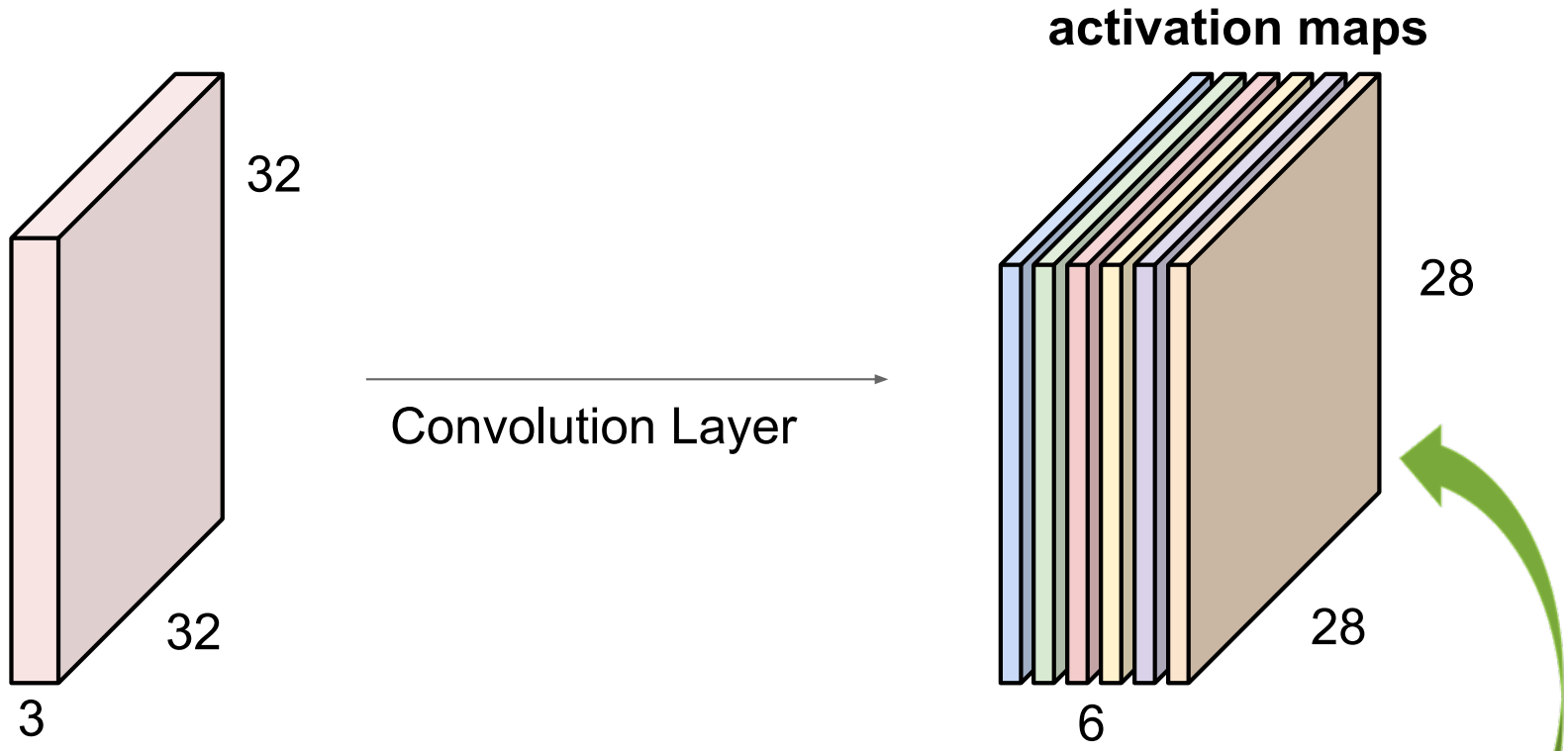
stride **S**

padding **P**

If width $\mathbf{I}_{\text{width}}$ and height $\mathbf{I}_{\text{height}}$ of the input differ, this formula is applied independently for $\mathbf{I}_{\text{width}}$ and $\mathbf{I}_{\text{height}}$.

Back to convolutional layer

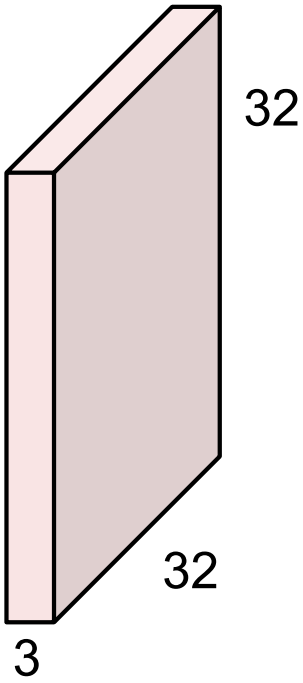
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a "new image" of size 28x28x6!

Spatial dimension: $\frac{32-5+2\cdot 0}{1}+1=28$

Ask yourself



Convolution Layer

Output volume size

?

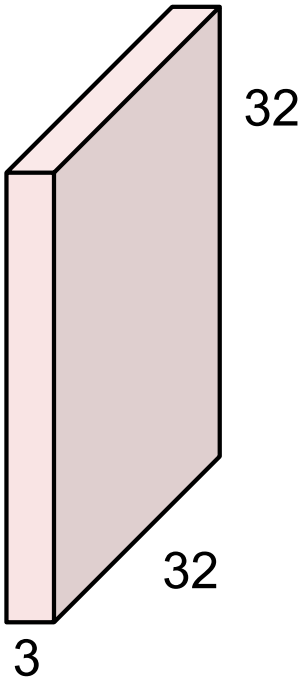
32x32x3 image

10 filters 5x5x3

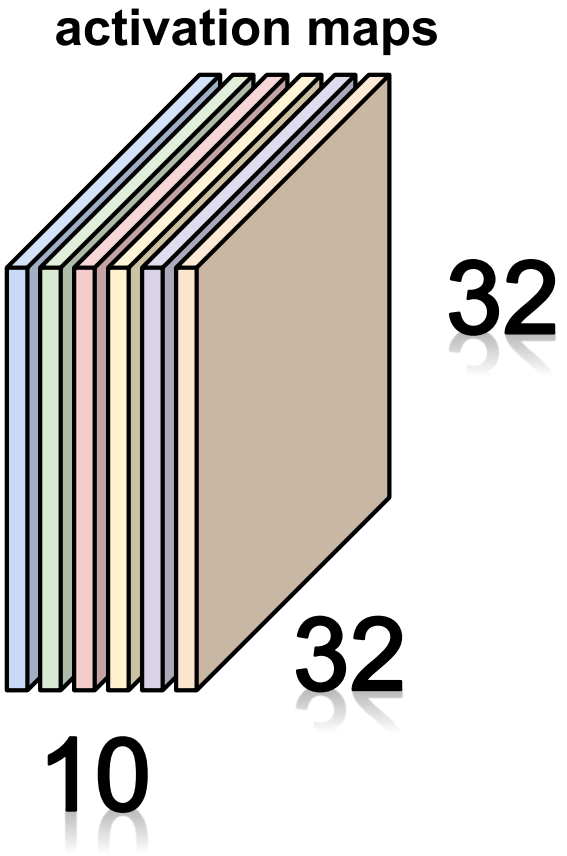
stride S=1

padding P=2

Ask yourself



Convolution Layer



32x32x3 image

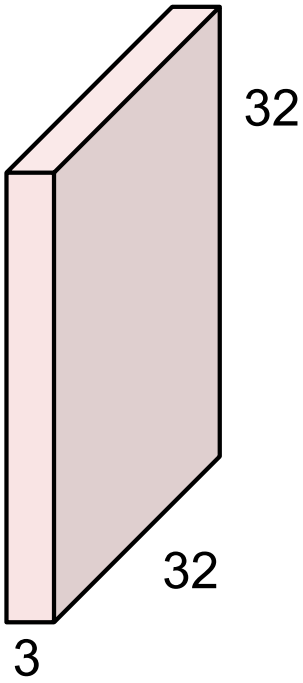
10 filters 5x5x3

stride S=1

padding P=2

$$\frac{32-5+2\cdot 2}{1}+1=32$$

Quiz time



Convolution Layer

32x32x3 image

10 filters 5x5x3

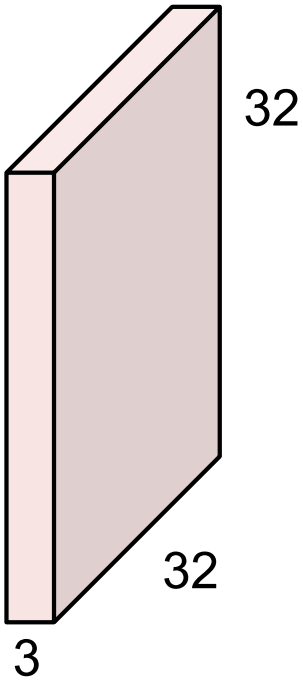
stride S=1

padding P=2

Number of **parameters** in this layer?

?

Quiz time



Convolution Layer

32x32x3 image

10 filters 5x5x3

stride S=1

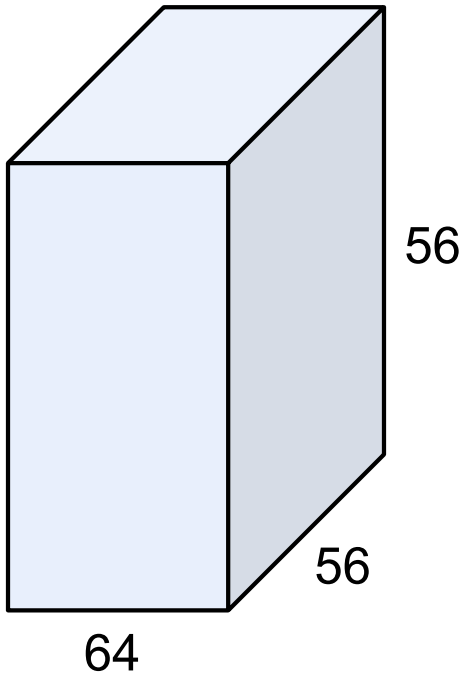
padding P=2

Number of **parameters** in this layer?

Each filter has 5x5x3=75 parameters

=>75x10 = **750**

Quiz time



1x1 CONV
with 32 filters

(each filter has size 1x1x64, and performs a 64-dimensional dot product)

Can we do convolution with 1x1xdepth filter

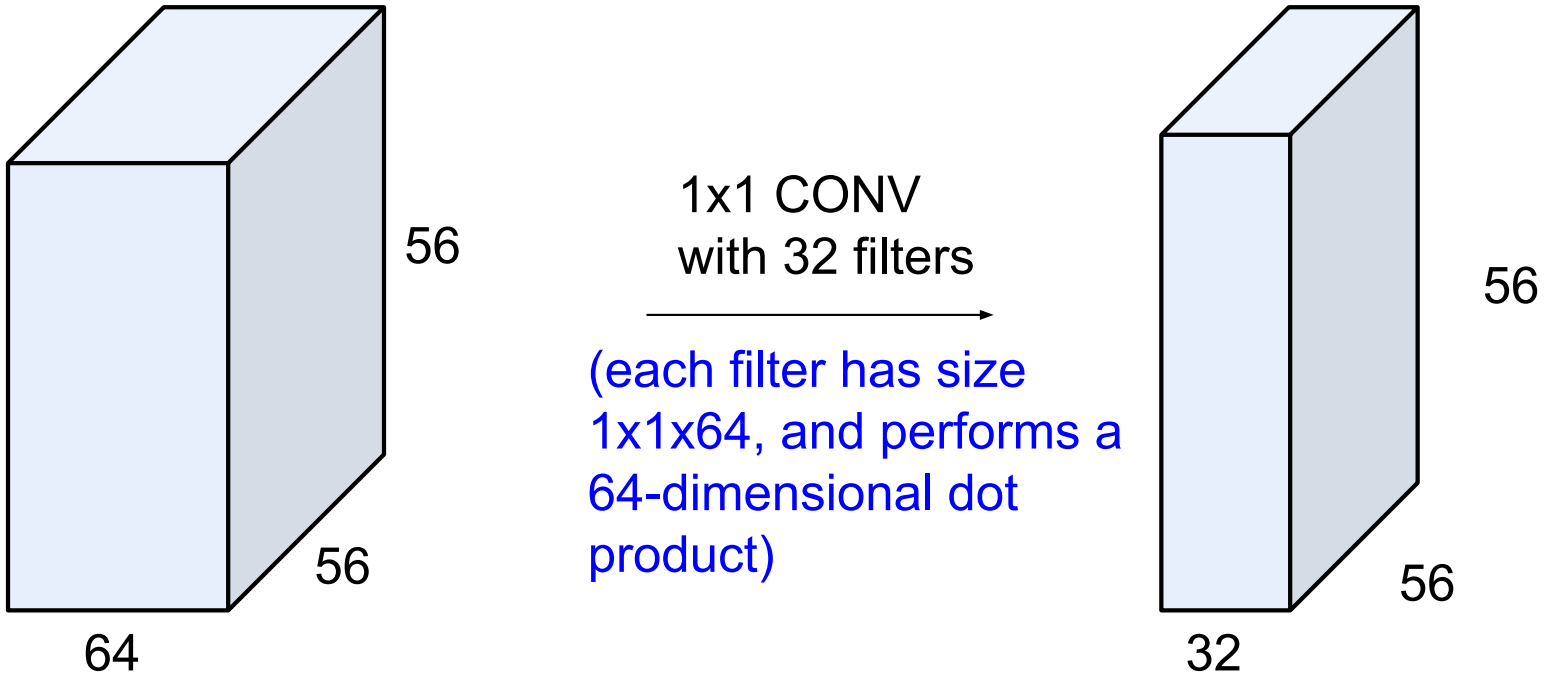
?

56x56x64 image

32 filters 1x1x64

S=1, P=0

Quiz time



Inexpensive convolution

Using 5x5x64 filters would result in 1600-dimensional dot product

Convolutional layer: summary

- Accepts an input of size **I**x**I**x**d**
- Requires four specifications:
  - Number of filters **K**
  - Filter size **F**x**F**x**d**
  - The stride S
  - Padding P
- Outputs a volume of size **O**x**O**x**K**, where O = $\frac{I - F + 2P}{S} + 1$

Often in practice:
K is power of 2, e.g. 32, 64, 128
F = 3, S=1, P=1
F = 5, S=1, P=2
F = 5, S=2, P is set accordingly
F = 1, S=1, P=0

- In the output volume, the i-th activation map is the result of a convolution of the i-th filter over the input with a stride S and padding P.

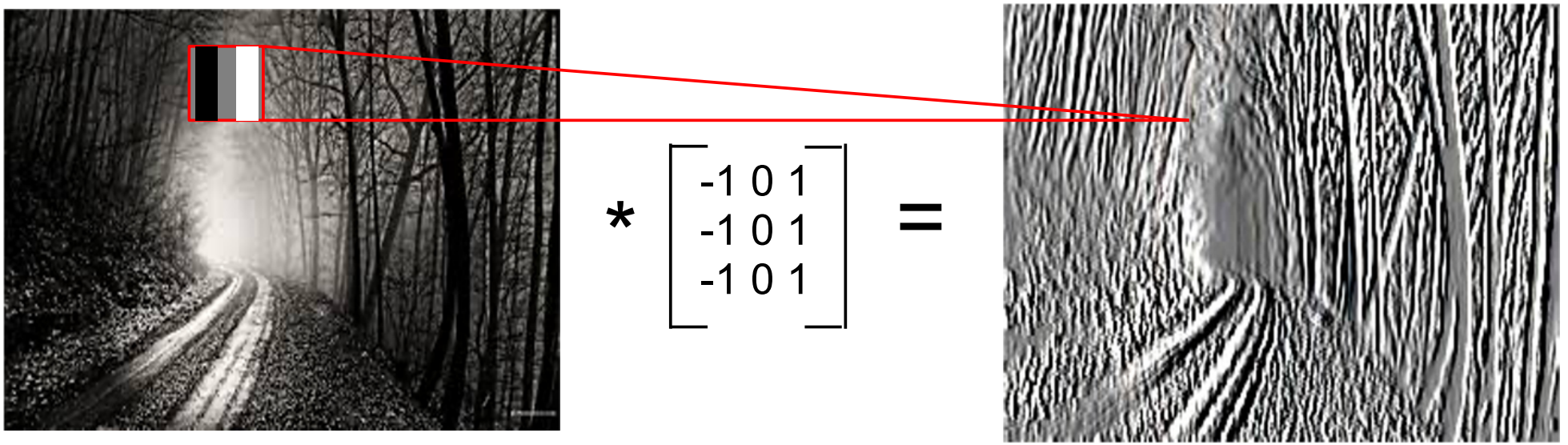
- **Local connectivity and parameter sharing**:
- each convolutional layer has (**F**x**F**x**d**)x**K** weight parameters to be learned (the fully connected layer would have **I**x**I**x**d**x**O**x**O**x**K** par.)

Why is it called convolutional layer?

We call the layer convolutional because it is related to convolution of two signals:

$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

elementwise multiplication and sum of a filter and the signal (image)

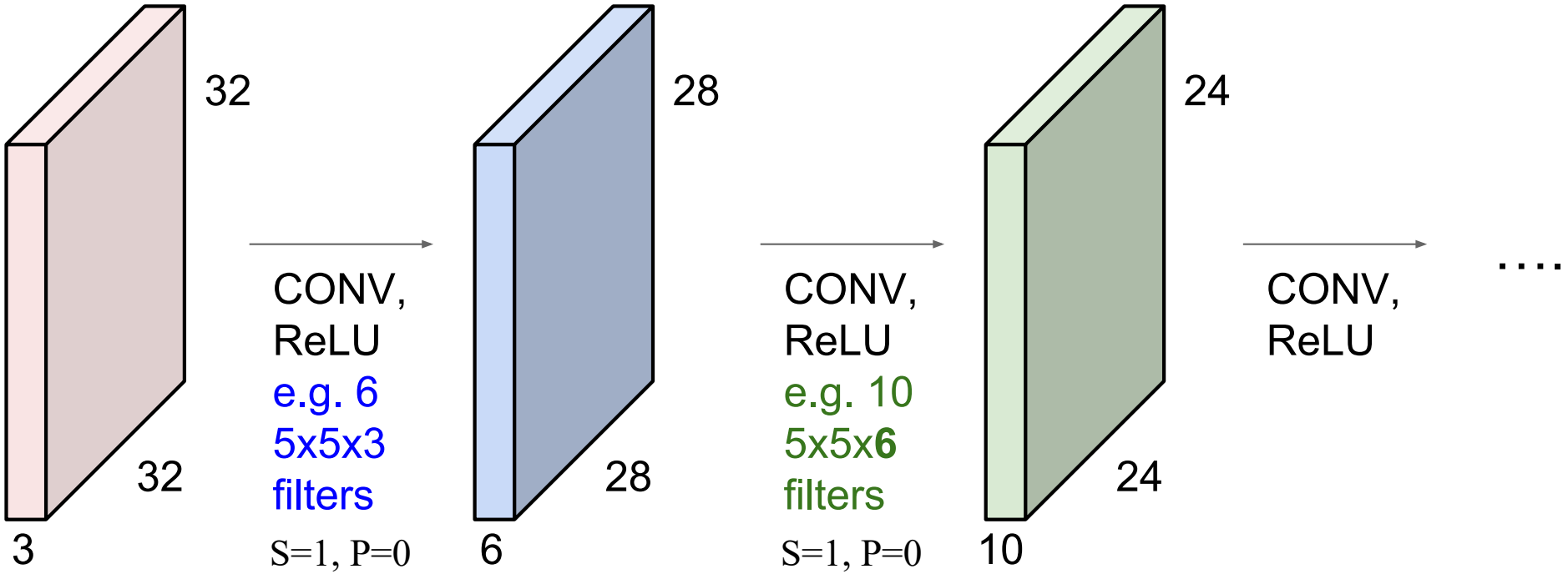


Deep Convolutional Networks

- Convolutional layer
- Non-linear activation function ReLU
- Max pooling layer
- Fully connected layer

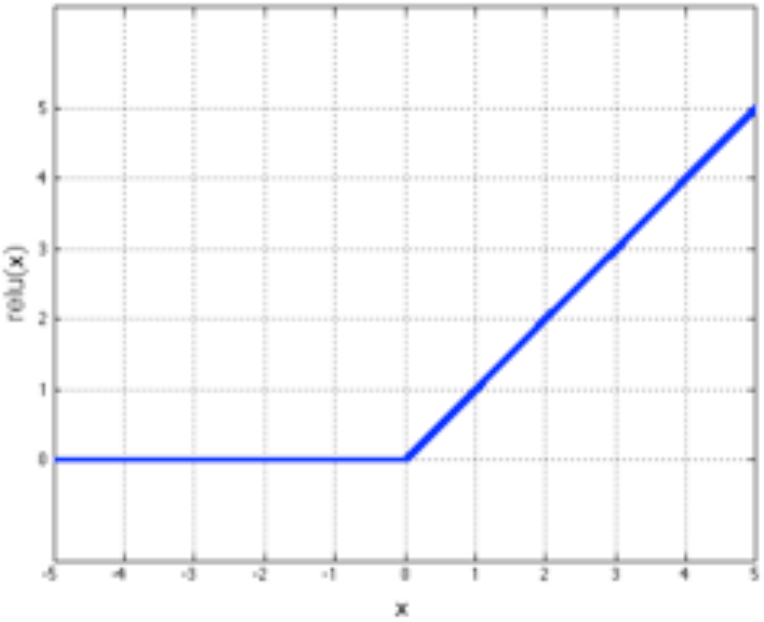
Where is ReLU?

Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



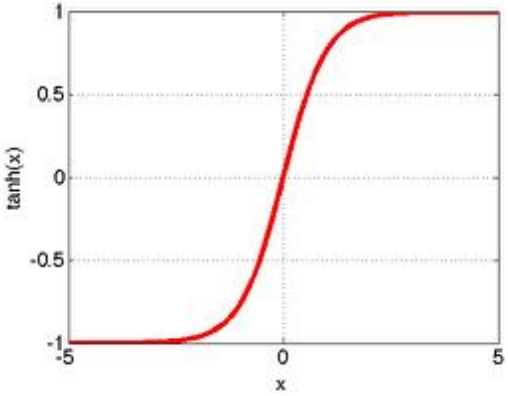
Rectified Linear Unit, ReLU

- Non-linear activation function are applied per-element
- Rectified linear unit (ReLU):
  - max(0,x)
  - makes learning faster (in practice x6)
  - avoids saturation issues (unlike sigmoid, tanh)
  - simplifies training with backpropagation
  - preferred option (works well)

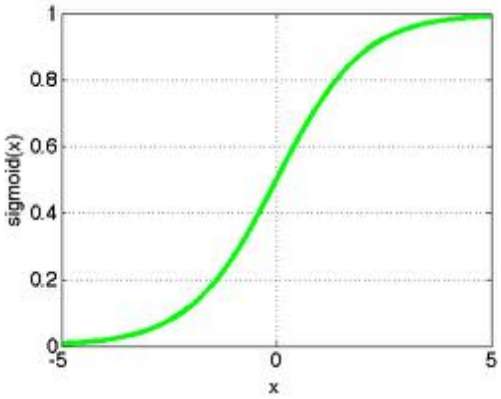


Other examples:

tanh(x)



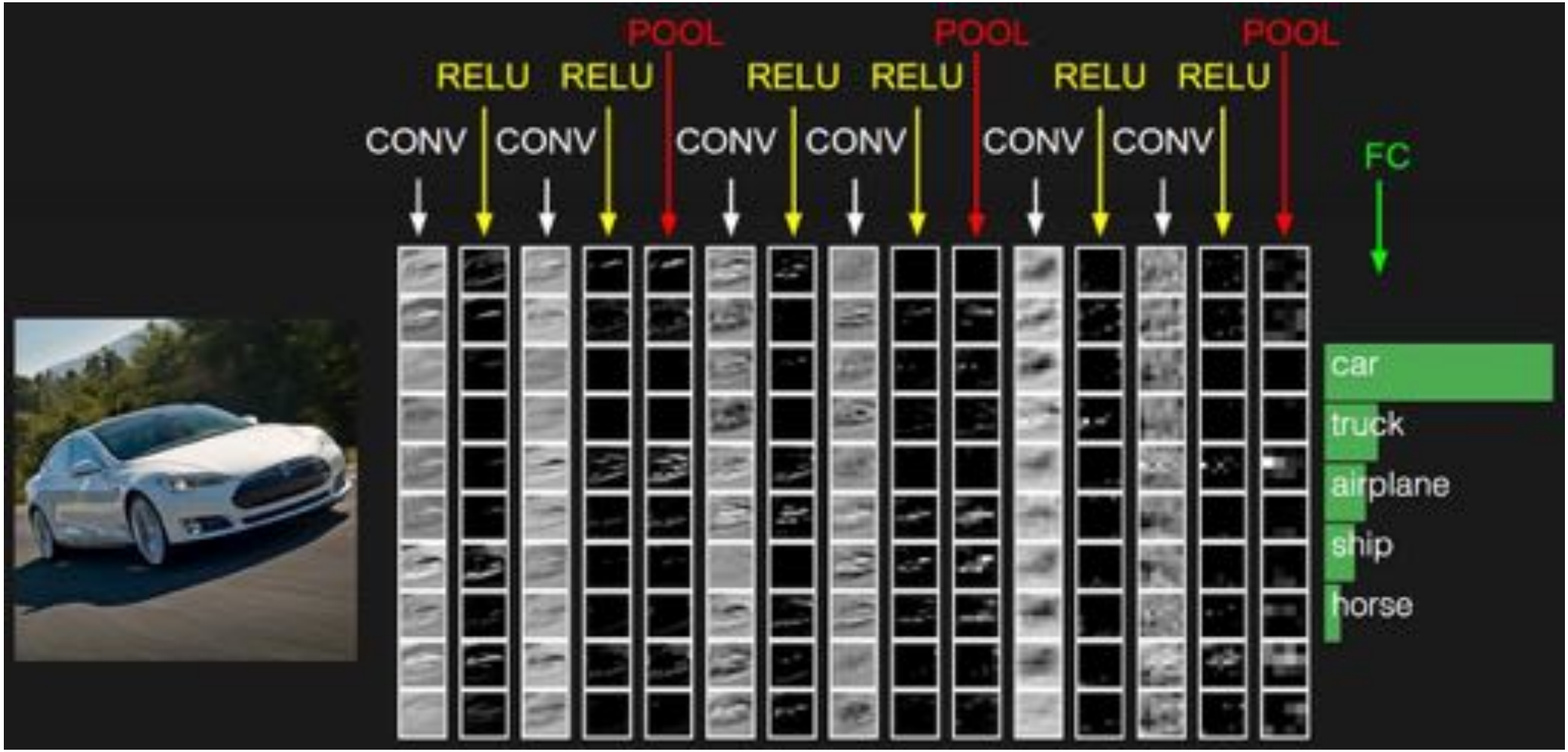
sigmoid(x)=$(1+e^{-x})^{-1}$



Deep Convolutional Networks

- Convolutional layer
- Non-linear activation function ReLU
- Max pooling layer
- Fully connected layer

Where is pooling?

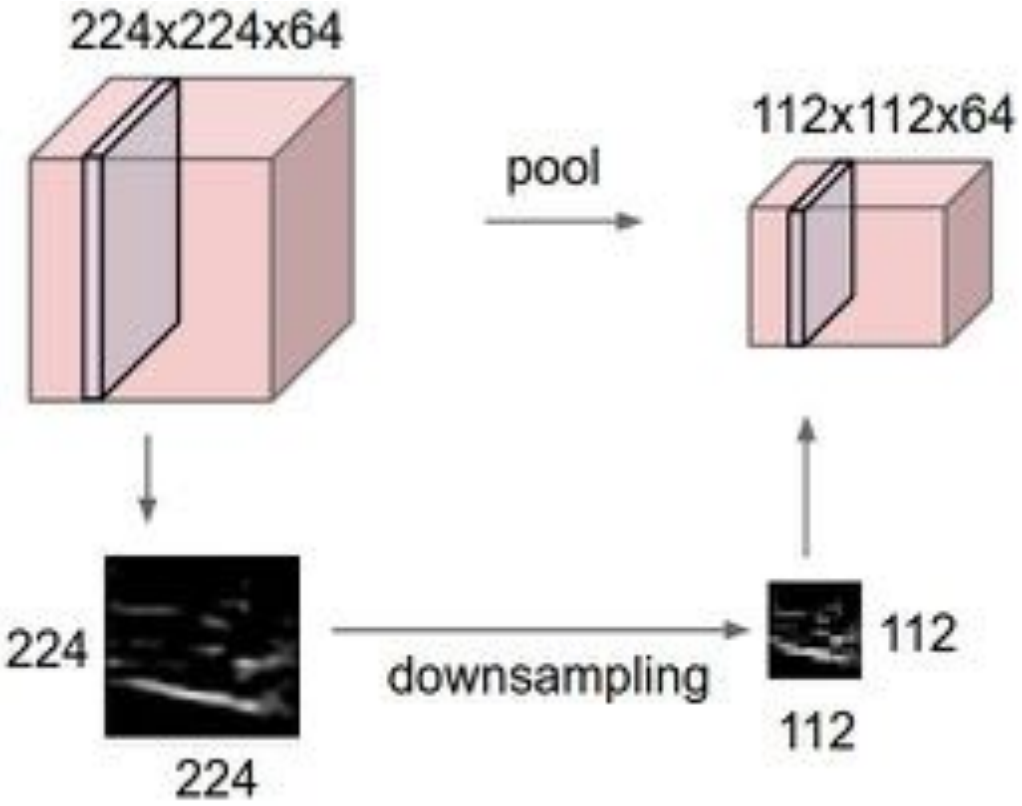


Two more layers to go: pooling and fully connected layers ☺

Spatial pooling

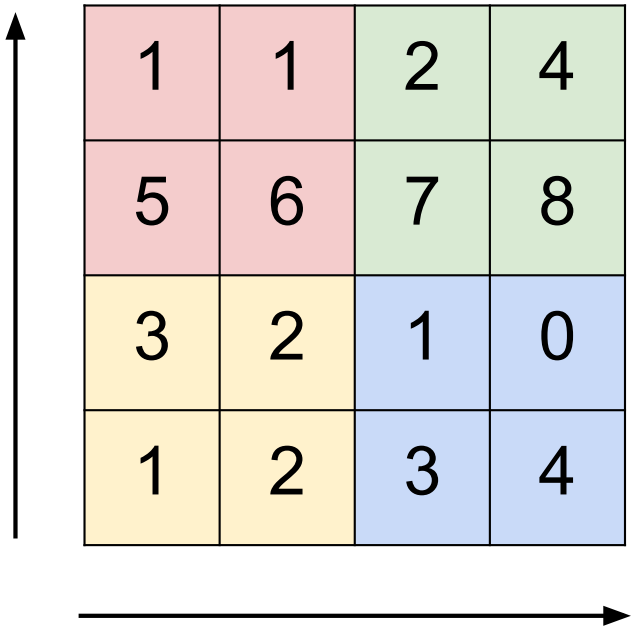
Pooling layer

- Makes the representations smaller (downsampling)
- Operates over each activation map independently
- Role: invariance to small transformation

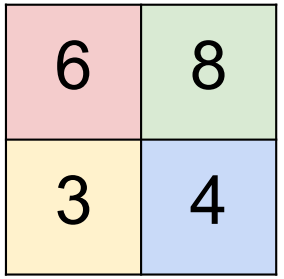


Max pooling

Single activation map



max pool with 2x2 filters and stride 2



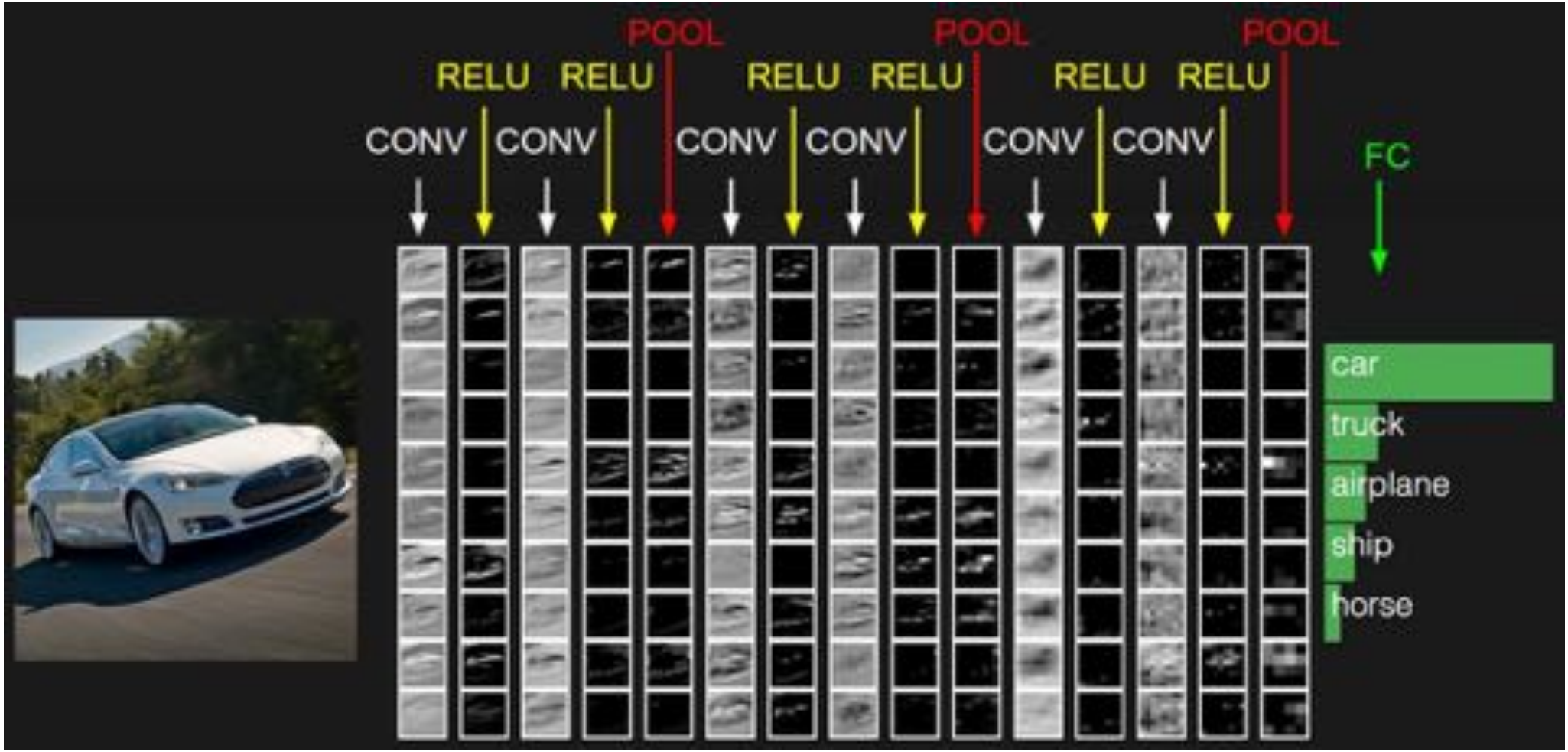
Alternatives:

- sum pooling
- overlapping pooling

Deep Convolutional Networks

- ☑ Convolutional layer
- ☑ Non-linear activation function ReLU
- ☑ Max pooling layer
- ❑ Fully connected layer

Where is a fully connected layer?



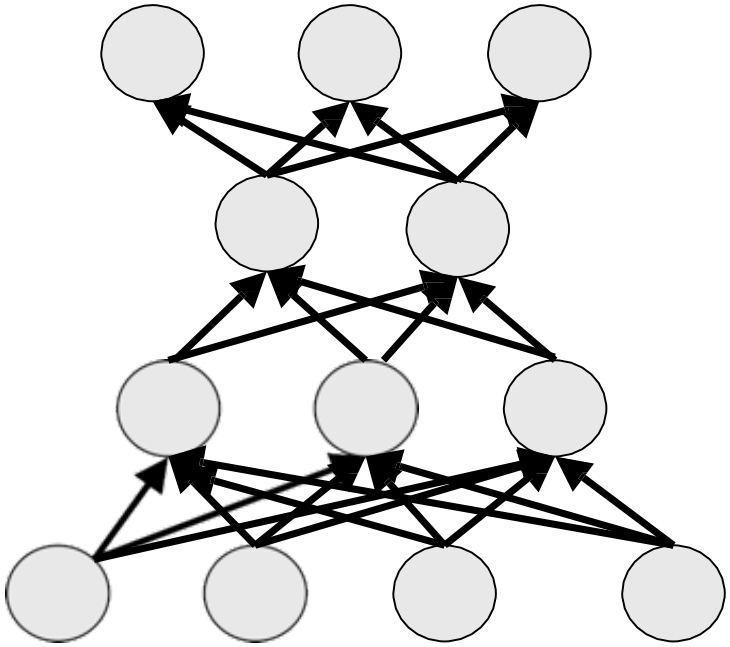
Fully connected layer

Contains neurons that connect to the entire input volume, as in ordinary Neural Networks:

Output layer

Hidden layer

Hidden layer

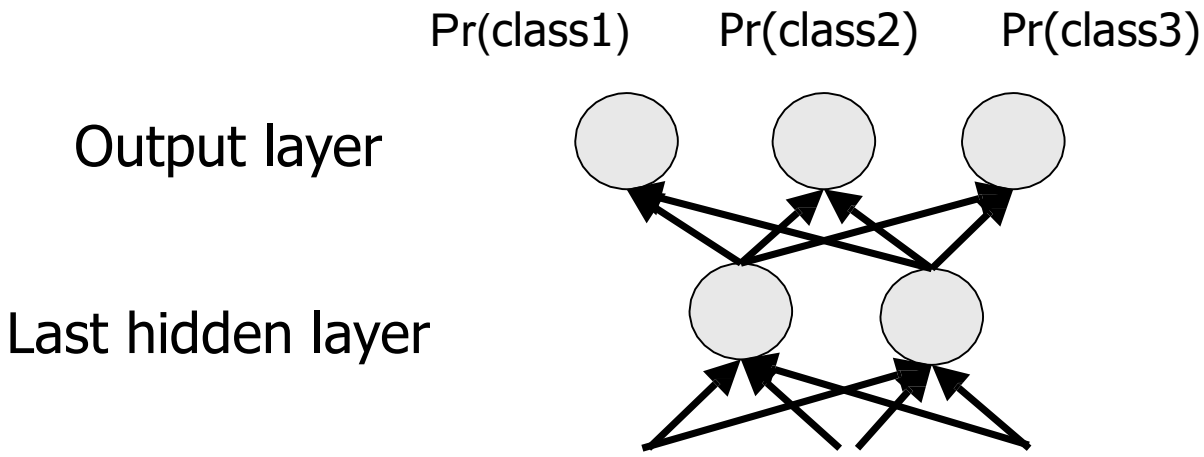


neurons between two adjacent layers are fully pairwise connected, but neurons within a single layer share no connections

Output layer

In classification:

- the output layer is fully connected with number of neurons equal to number of classes
- followed by softmax non-linear activation



Running CNNs demo

To see this in action, check

http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

ImageNet Large Scale Visual Recognition Challenge (ILSVRC-2010)

- 1K Kategorien
- 1.2M Trainingsbilder (~1000 pro Katgeorie)
- 50,000 Validierungsbilder
- 150,000 Testbilder

Bildklassifikation

- Faltungsschichten
- Nichtlineare Aktivierungsfunktion ReLU
- Max pooling Schicht
- Vollständig verbundene Schicht

For Training use (Mini-batch) Stochastic gradient descent (SGD)

Initialize the parameters

Run over all training examples (several times):

- **Draw randomly zufällig** a (small batch of) data point(s)
- Propagation the activations **forward** from the input to the output layer and compute the classification error, e.g.,

$$E = \frac{1}{2}(y_{predicted} - y_{true})^2$$

- Propagation the error **backwards**, i.e., compute the gradient of the error function w.r.t. to the parameters via backpropgation
- **Update** the parameters by going into the direction of the gradient, **SGD:**

$$w^{t+1} = w^t - \alpha \cdot \frac{dE}{dw}(w^t)$$

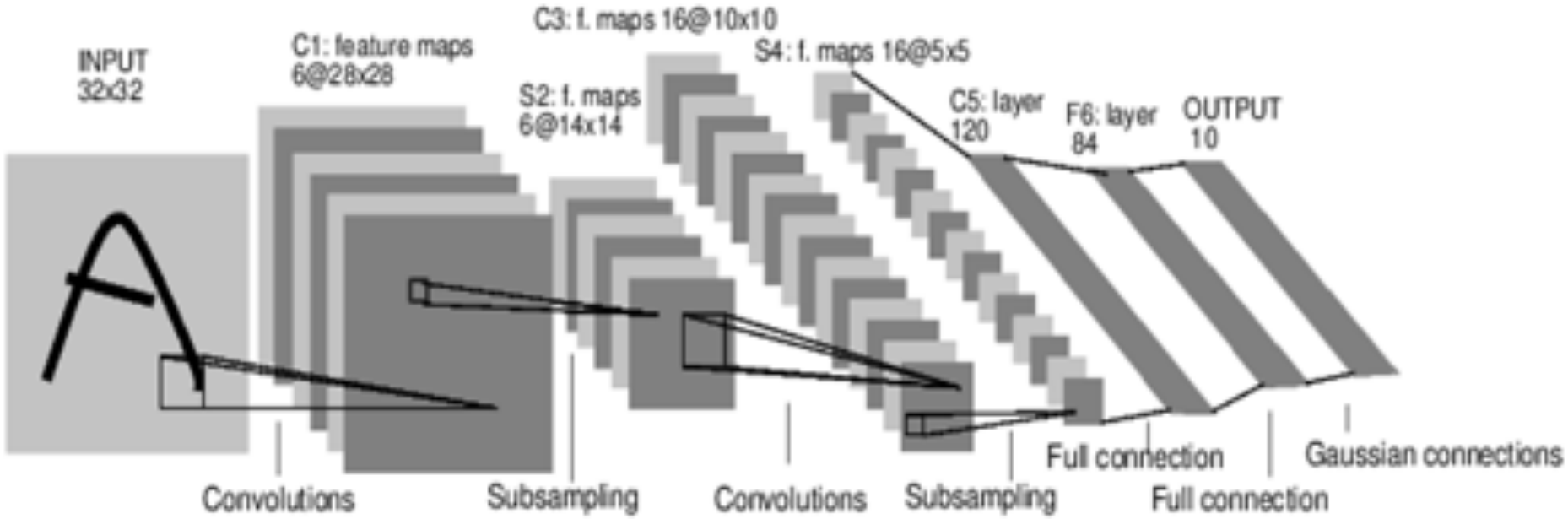
A lot of tricks and other things not touched upon here

Learning DNNs is a kind of art!

Yann Lecun : ”DNNs require an interplay between intuitive insights, theoretical modeling, practical implementations, empirical studies, and scientific analyses”

Example: LeNet-5

[LeCun et al., 1998]

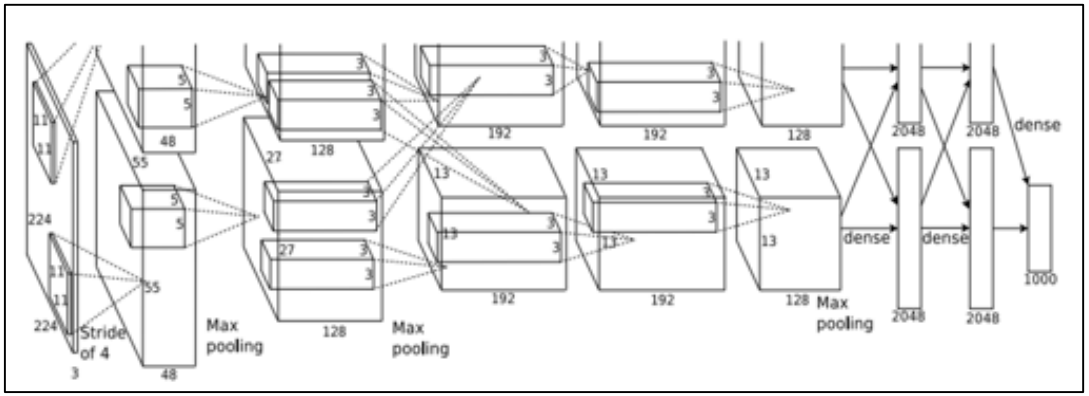


Conv Filter with 5x5 and stride 1
2x2 Pooling Layers with stride 2
Tanh non-linearity

[CONV-POOL-CONV-POOL-CONV-FC]

Example: AlexNet

[Krizhevsky et al., 2012]

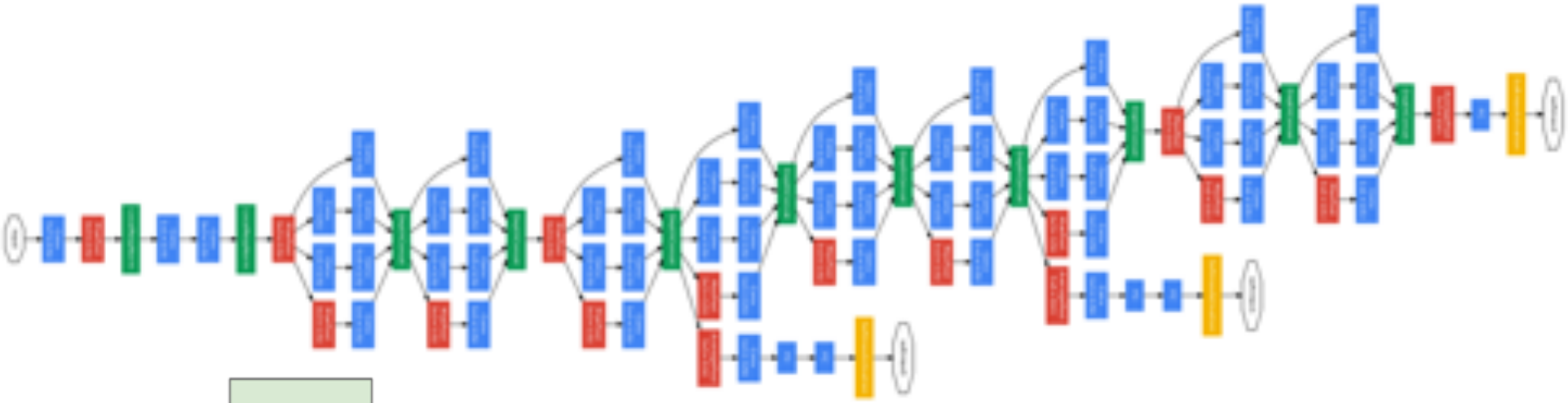


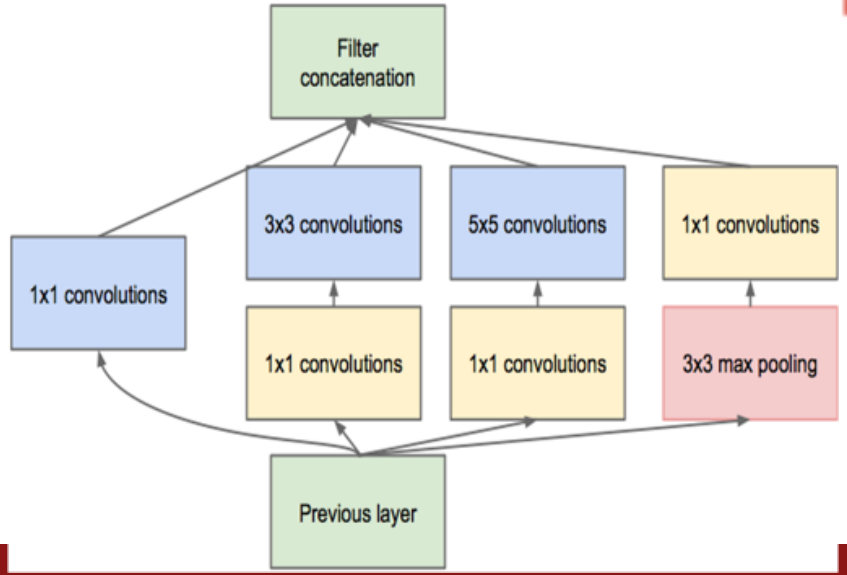
Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256]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

Input: 227x227x3 images

Example: GoogLeNet

[Szegedy et al., 2014]

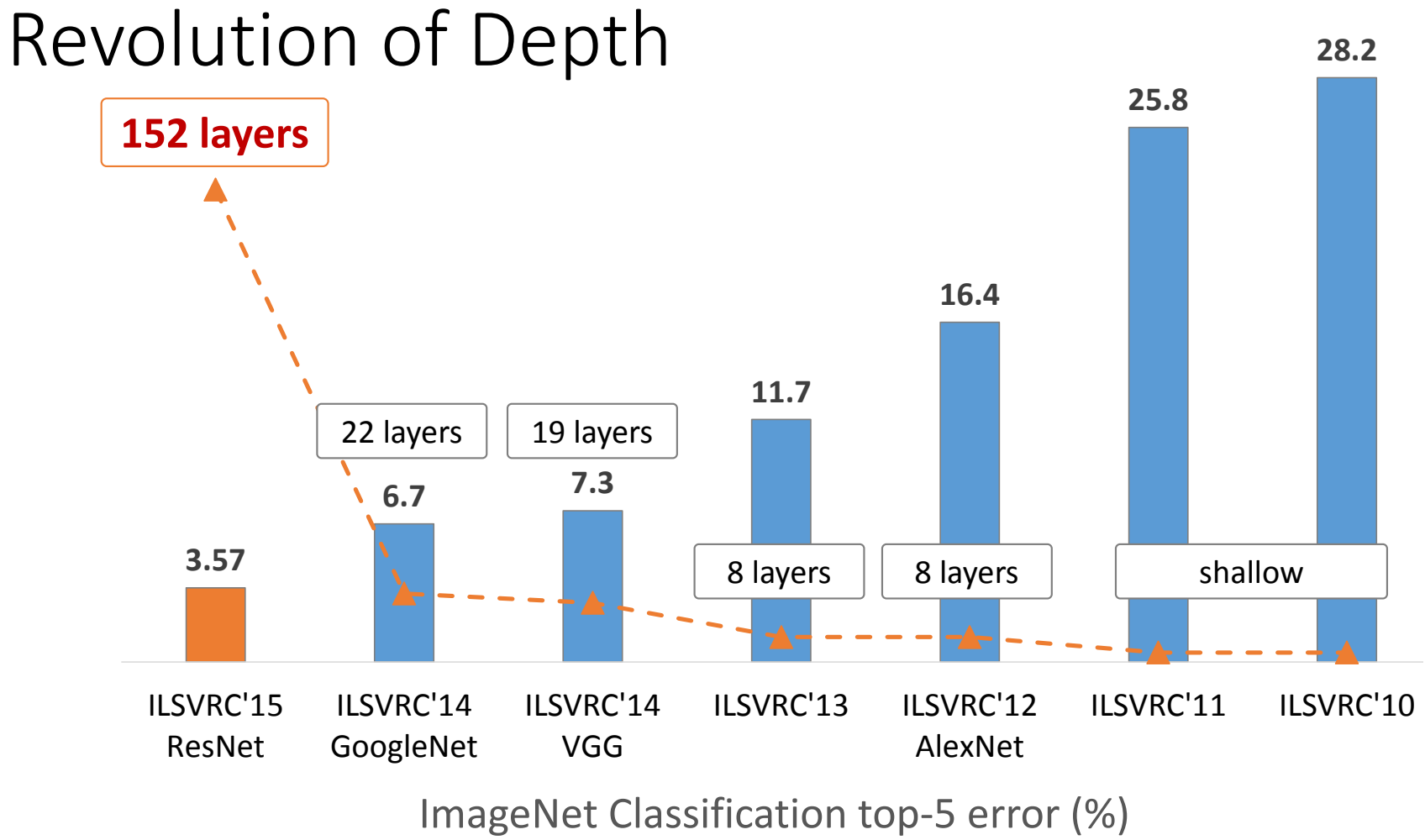




Inception Modul

ILSVRC 2014 Gewinner (6.7% top 5 error)

The Deep Learning Revolution



Examples for Deep Learning Frameworks

Deep Learning with PyTorch

What have we learnt?

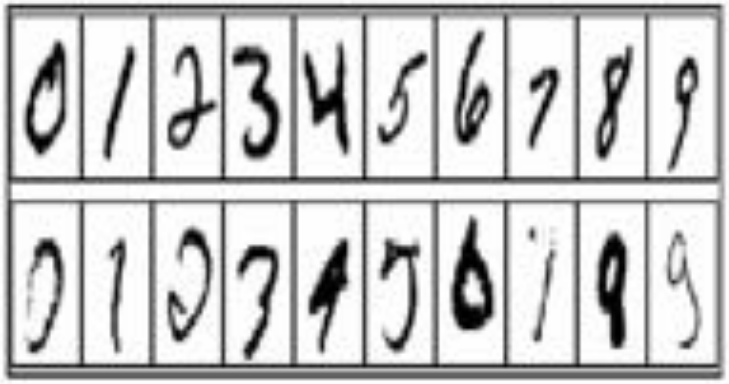
- Deep Neural Networks aim at learning feature hierachies
- We have understood the structure of convolutional neural networks, one of the central DNN architectures

  Convolutional layer, ReLU,Max pooling layer, fully connected layer

  as well as how to compute spatial dimensions and the number of parameters
- DNNs are rather large but result in state-of-the-art performance on many tasks

Wide Variety of Applications

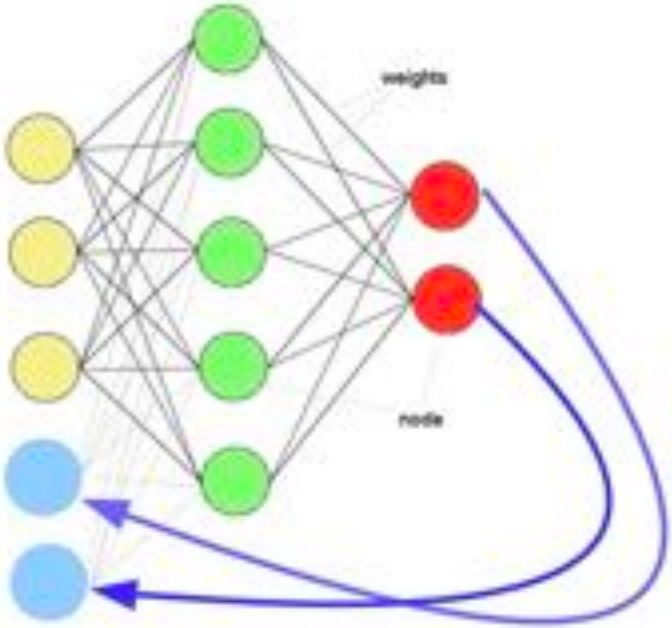
- Speech Recognition
- Autonomous Driving
- Handwritten Digit Recognition
- Credit Approval
- Backgammon
- etc.



- Good for problems where the final output depends on combinations of many input features
  - rule learning is better when only a few features are relevant
- Bad if explicit representations of the learned concept are needed
  - takes some effort to interpret the concepts that form in the hidden layers

Recurrent Neural Networks

- Recurrent Neural Networks (RNN)
  - allow to process sequential data
  - by feeding back the output of the network into the next input
- Long-Short Term Memory (LSTM)
  - add „forgetting“ to RNNs
  - good for mapping sequential input data into sequential output data
    - e.g., text to text, or time series to time series



- Deep Learning often allows „end-to-end learning“
  - e.g., learn a network that does the complete translation of text in one language into another language
  - previously, learning often concentrated on individual components (e.g. word sense disambiguation)

Self-Supvervised Learning and Transformer