Introduction to Deep Learning

Outline

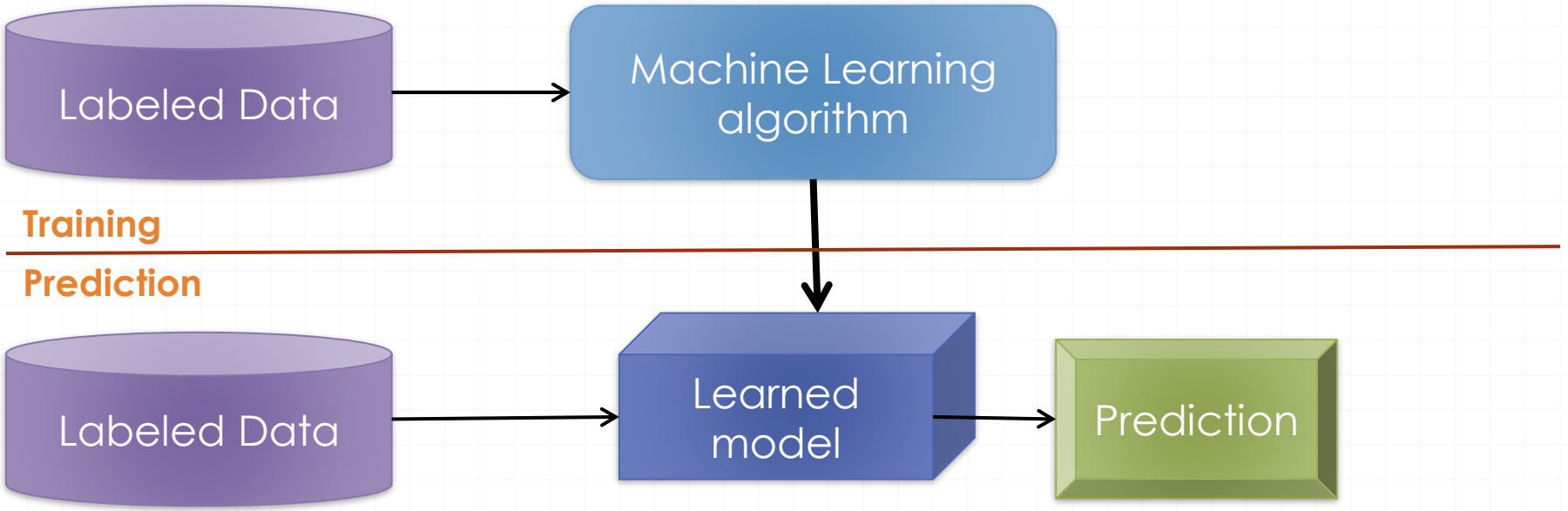
- Machine Learning basics
- Introduction to Deep Learning
  - what is Deep Learning
  - why is it useful
- Main components/hyper-parameters:
  - activation functions
  - optimizers, cost functions and training
  - regularization methods
  - tuning
  - classification vs. regression tasks
- DNN basic architectures:
  - convolutional
  - recurrent
  - attention mechanism
- Application example: Relation Extraction

Most material from CS224 NLP with DL course at Stanford

~~Backpropagation~~
~~GANs & Adversarial training~~
~~Bayesian Deep Learning~~
~~Generative models~~
~~Unsupervised / Pretraining~~

Machine Learning Basics

Machine learning is a field of computer science that gives computers the ability to **learn without being explicitly programmed**



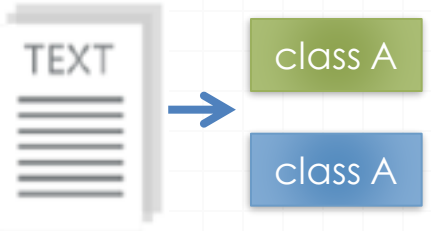
Methods that can learn from and make predictions on data

Types of Learning

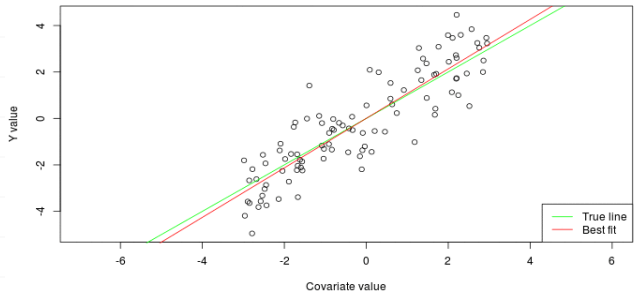
Supervised: Learning with a **labeled training** set
Example: email *classification* with already labeled emails

Unsupervised: Discover **patterns** in **unlabeled** data
Example: *cluster* similar documents based on text

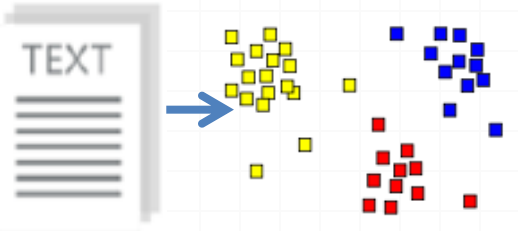
Reinforcement learning: learn to **act** based on **feedback/reward**
Example: learn to play Go, reward: *win or lose*



Classification



Regression



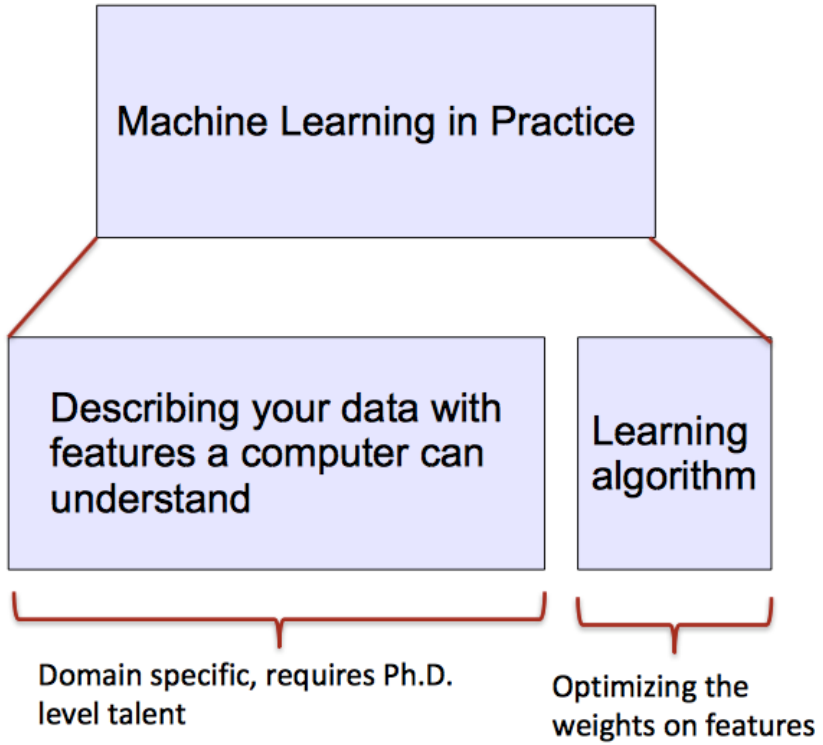
Clustering

Anomaly Detection
Sequence labeling
…

http://mbjoseph.github.io/2013/11/27/measure.html

ML vs. Deep Learning

Most machine learning methods work well because of **human-designed representations** and **input features**
ML becomes just **optimizing weights** to best make a final prediction

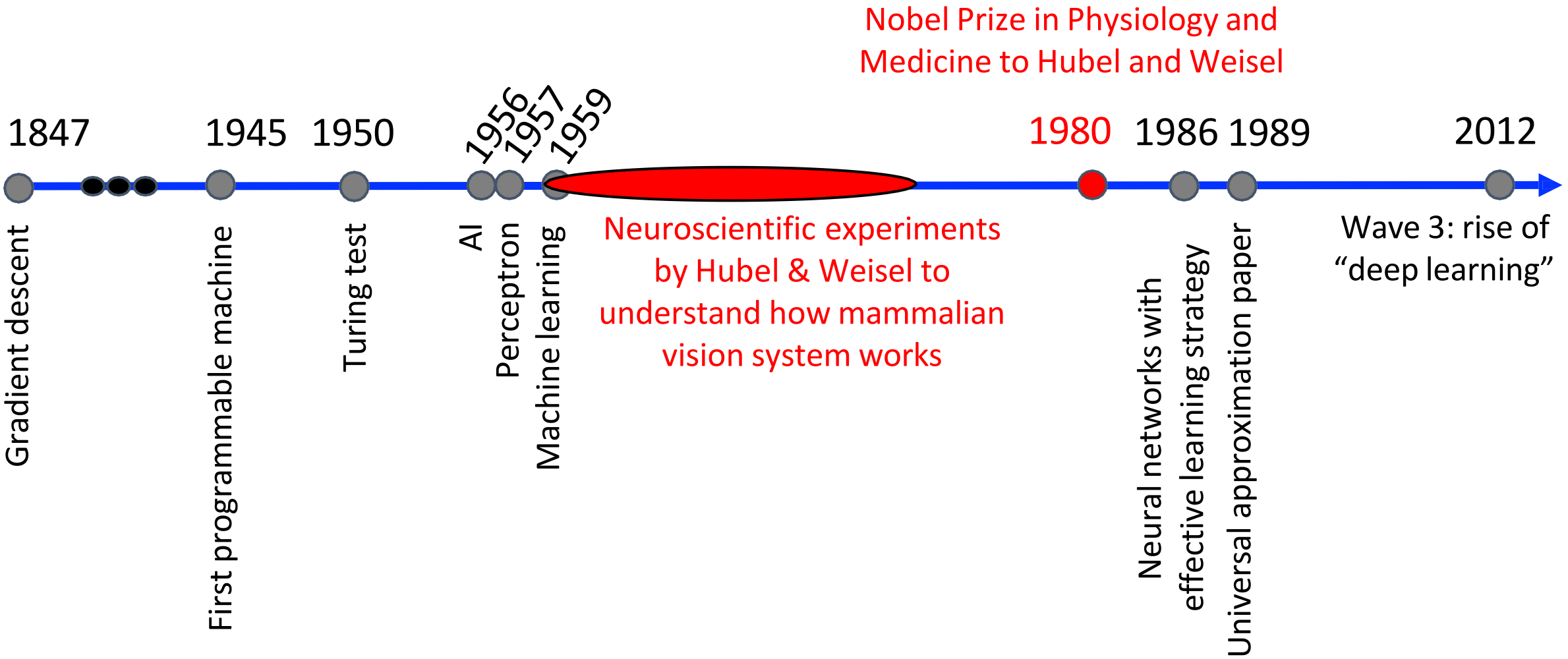


Feature	NER
Current Word	✓
Previous Word	✓
Next Word	✓
Current Word Character n-gram	all
Current POS Tag	✓
Surrounding POS Tag Sequence	✓
Current Word Shape	✓
Surrounding Word Shape Sequence	✓
Presence of Word in Left Window	size 4
Presence of Word in Right Window	size 4

Today's Topics

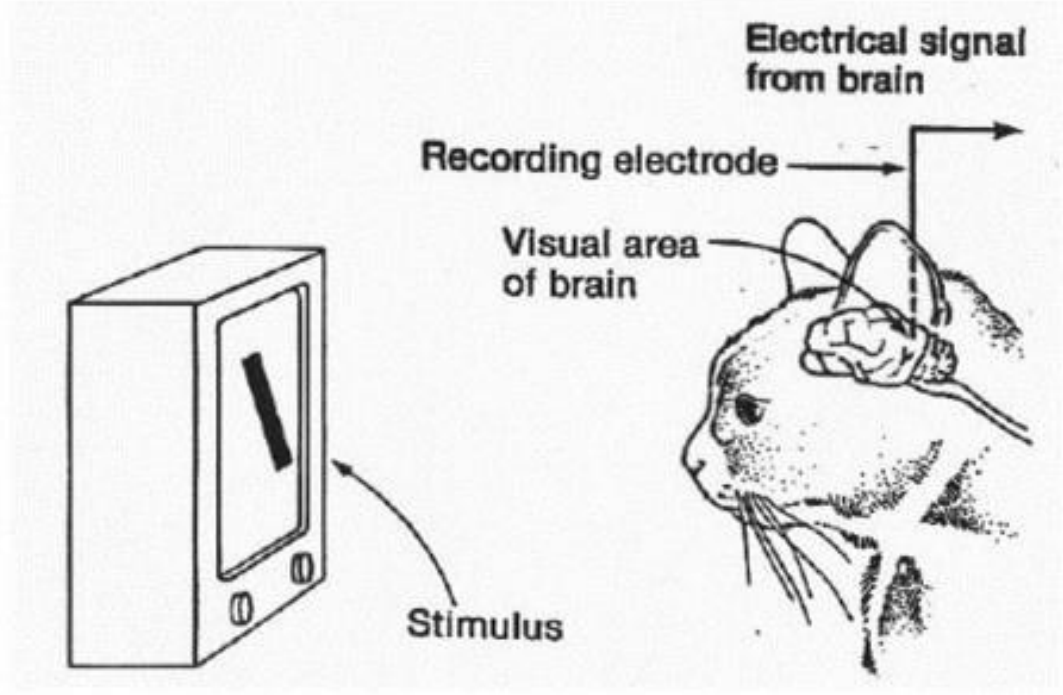
- Neural Networks for Spatial Data
- History of Convolutional Neural Networks (CNNs)
- CNNs – Convolutional Layers
- CNNs – Pooling Layers
- Programming tutorial

Historical Context: Inspiration



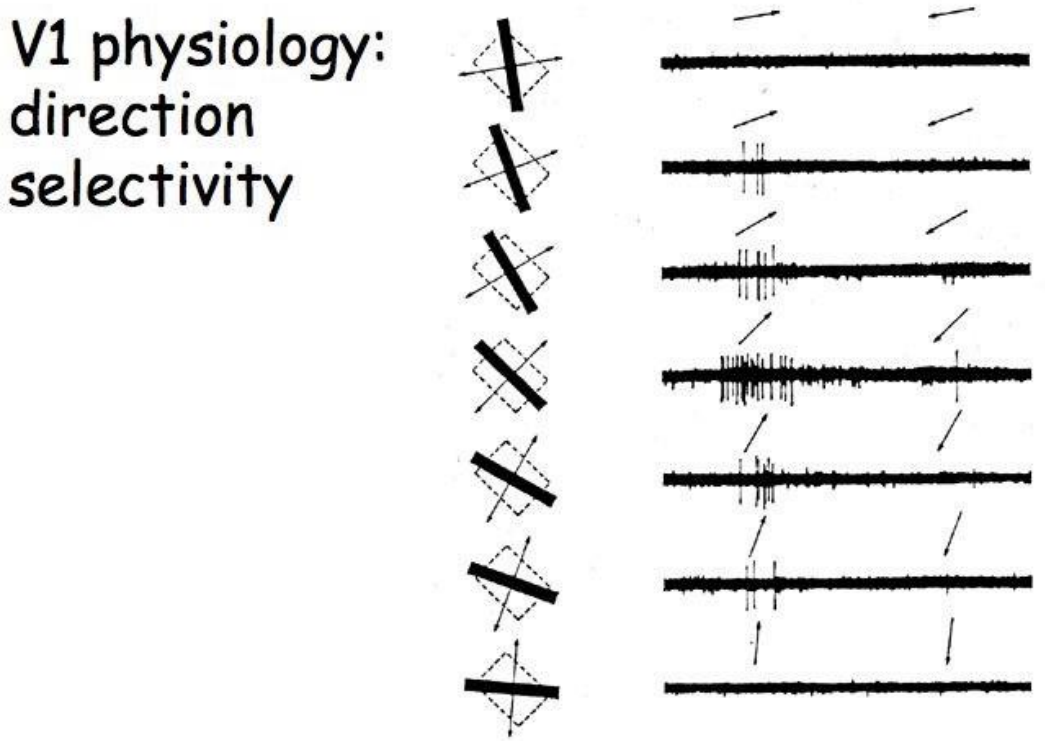
Motivation: How Vision System Works

Experiment Set-up:



https://www.esantus.com/blog/2019/1/31/convolutional-neural-networks-a-quick-guide-for-newbies

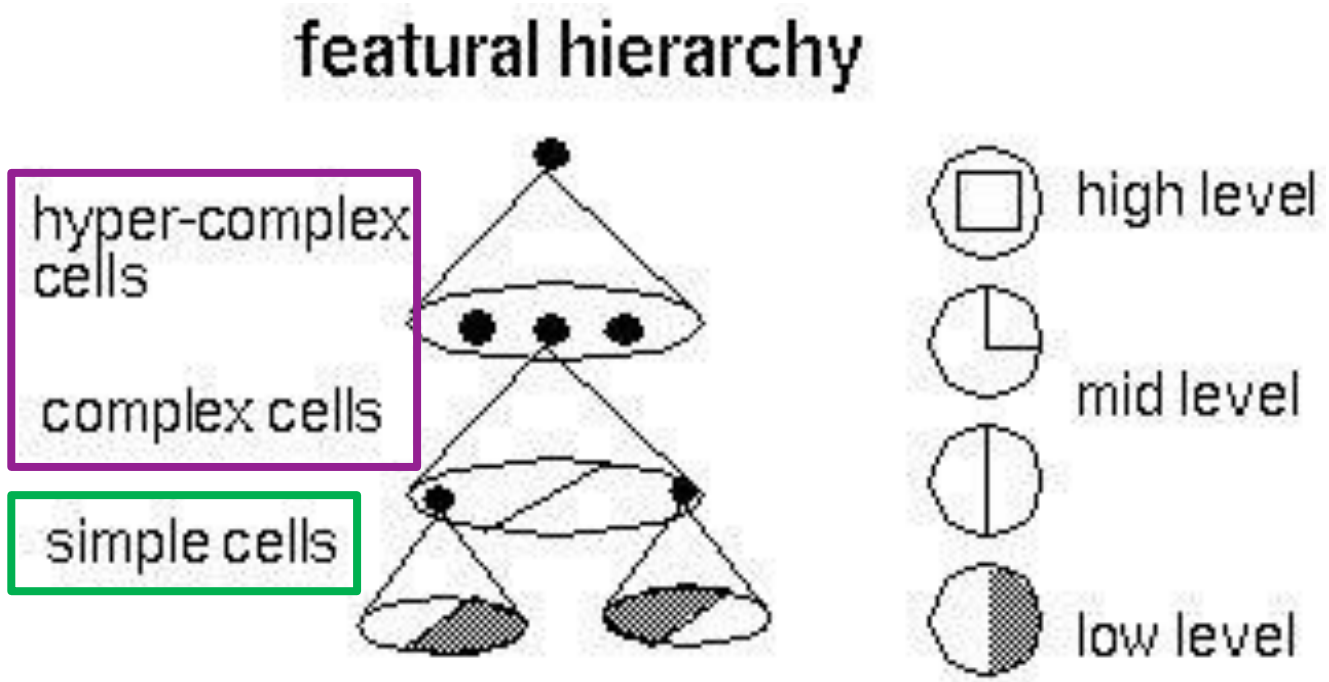
Key Finding: initial neurons responded strongly only when light was shown in certain orientations



https://www.cns.nyu.edu/~david/courses/perception/lecturenotes/V1/lgn-V1.html

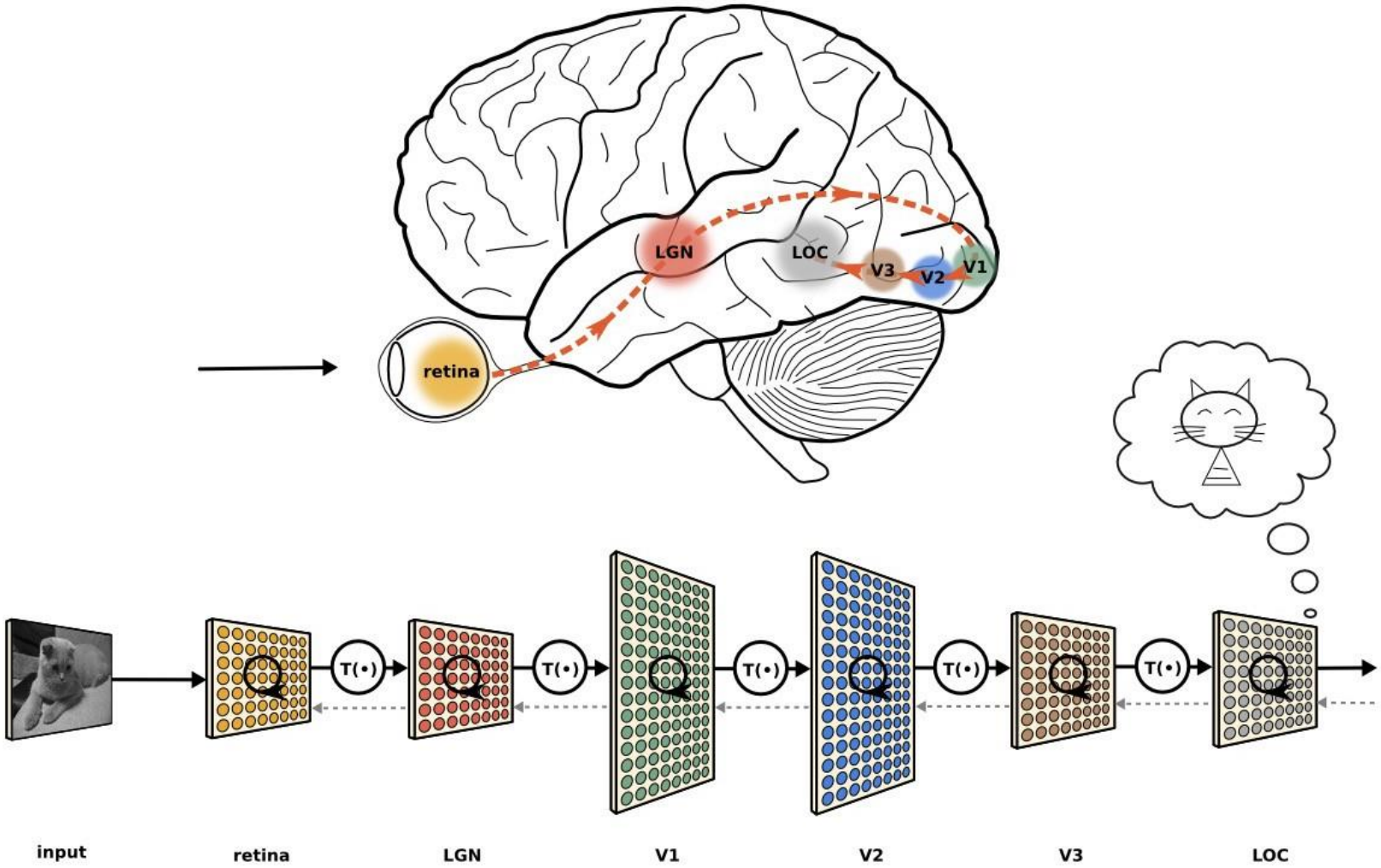
Motivation: How Vision System Works

Key Idea: cells are organized as a hierarchy of feature detectors, with higher level features responding to patterns of activation in lower level cells



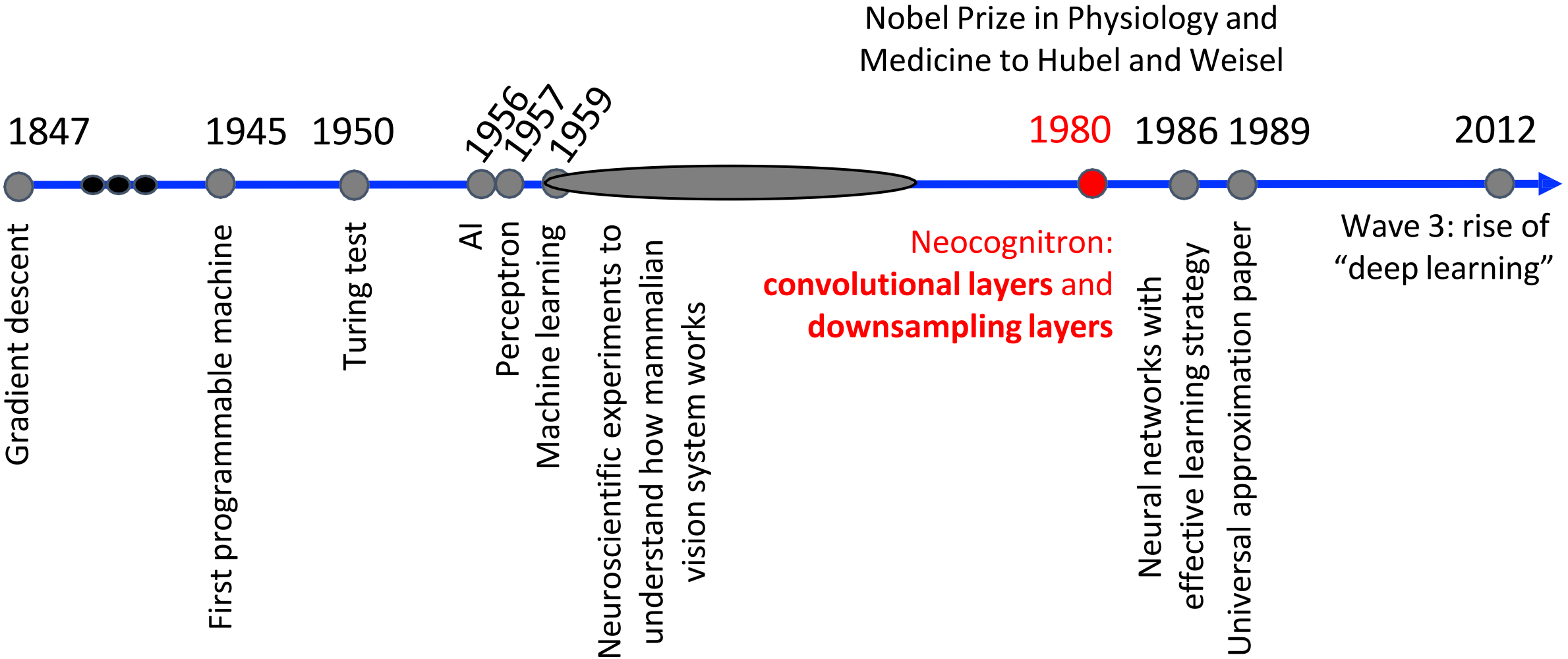
Source: https://bruceoutdoors.files.wordpress.com/2017/08/hubel.jpg

Motivation: How Vision System Works



https://neuwritesd.files.wordpress.com/2015/10/visual_stream_small.png

Historical Context: Key Ingredients



Neocognitron: Key Ingredients

http://personalpage.flsi.or.jp/fukushima/index-e.html

"In this paper, we discuss how to synthesize a neural network model in order to endow it an ability of pattern recognition like a human being… the network acquires a similar structure to the hierarchy model of the visual nervous system proposed by Hubel and Wiesel."

- Fukushima, Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position. Biological Cybernetics, 1980.

Neocognitron: Key Ingredients

Cascade of simple and complex cells:

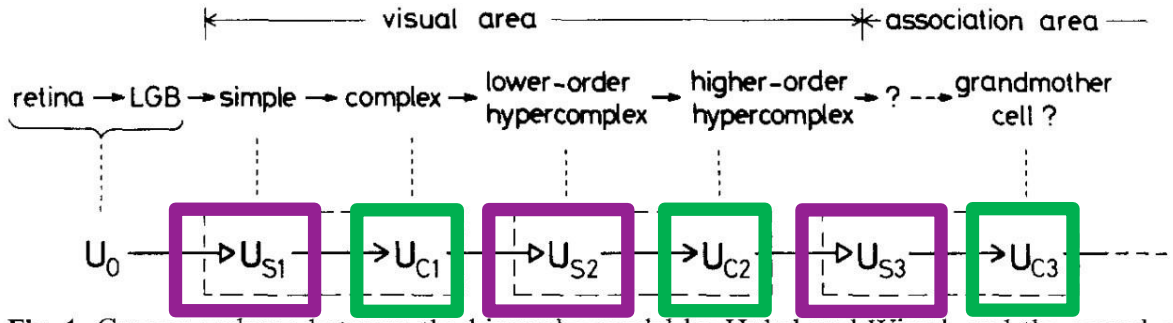


Fig. 1. Correspondence between the hierarchy model by Hubel and Wiesel, and the neural network of the neocognitron

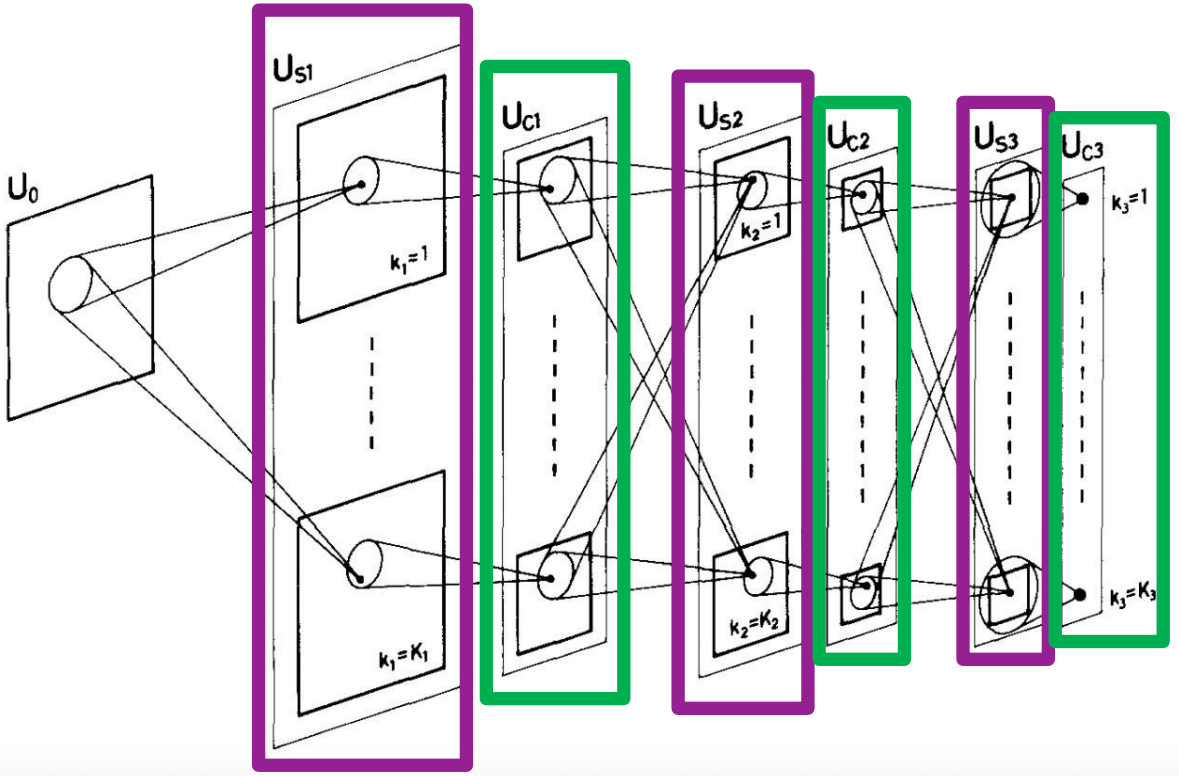
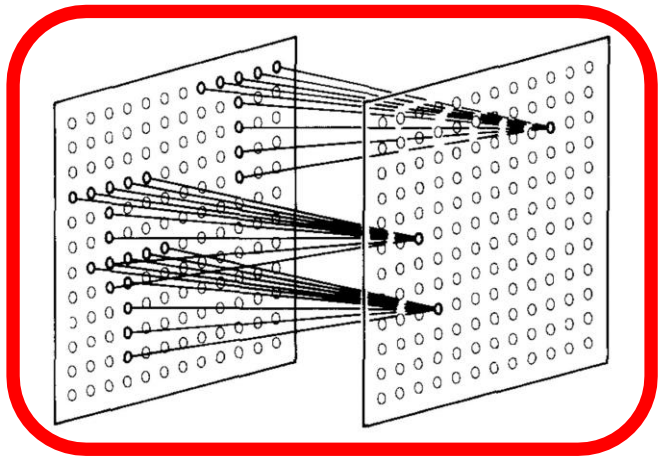


Fig. 2. Schematic diagram illustrating the interconnections between layers in the neocognitron

Fukushima, 1980.

Neocognitron: Key Ingredients

Simple cells extract local features using a sliding filter:



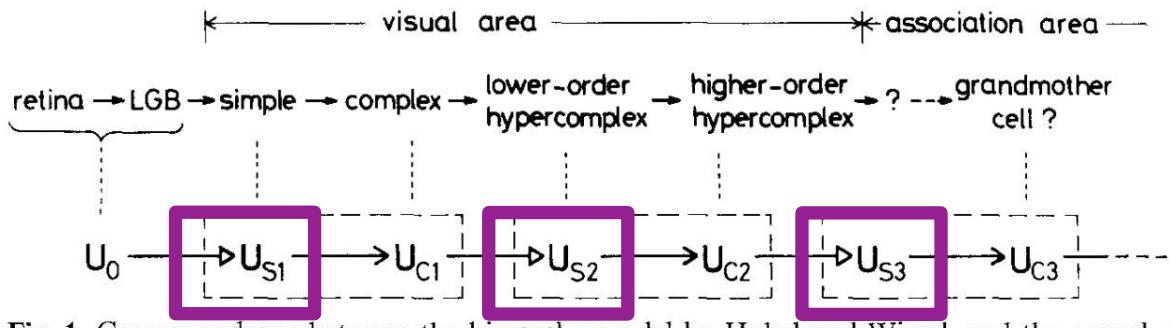


Fig. 1. Correspondence between the hierarchy model by Hubel and Wiesel, and the neural network of the neocognitron

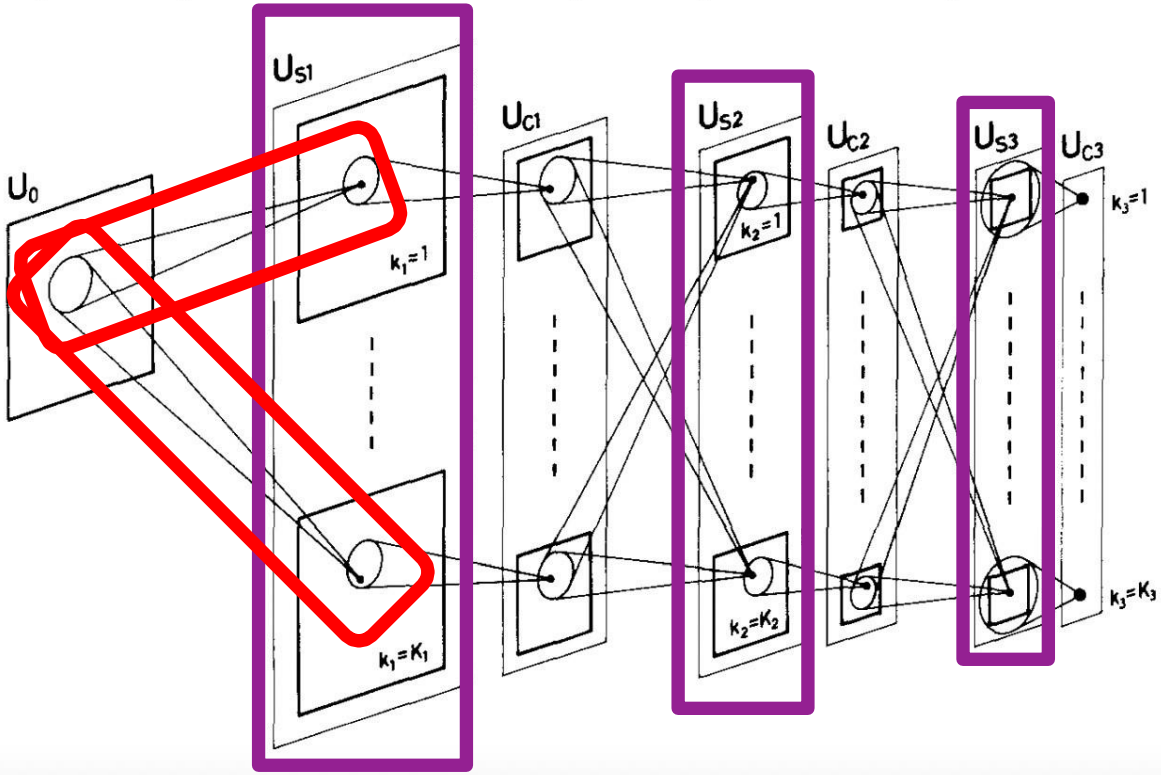


Fig. 2. Schematic diagram illustrating the interconnections between layers in the neocognitron

Fukushima, 1980.

Neocognitron: Key Ingredients

Complex cells fire when any part of the local region is the desired pattern

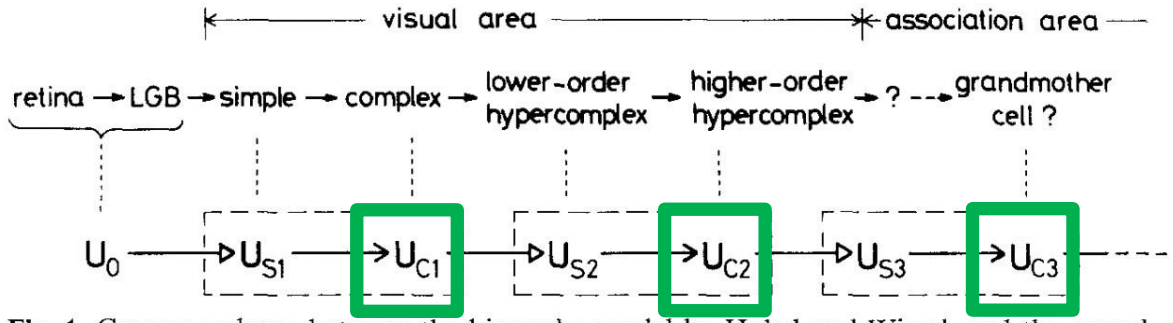


Fig. 1. Correspondence between the hierarchy model by Hubel and Wiesel, and the neural network of the neocognitron

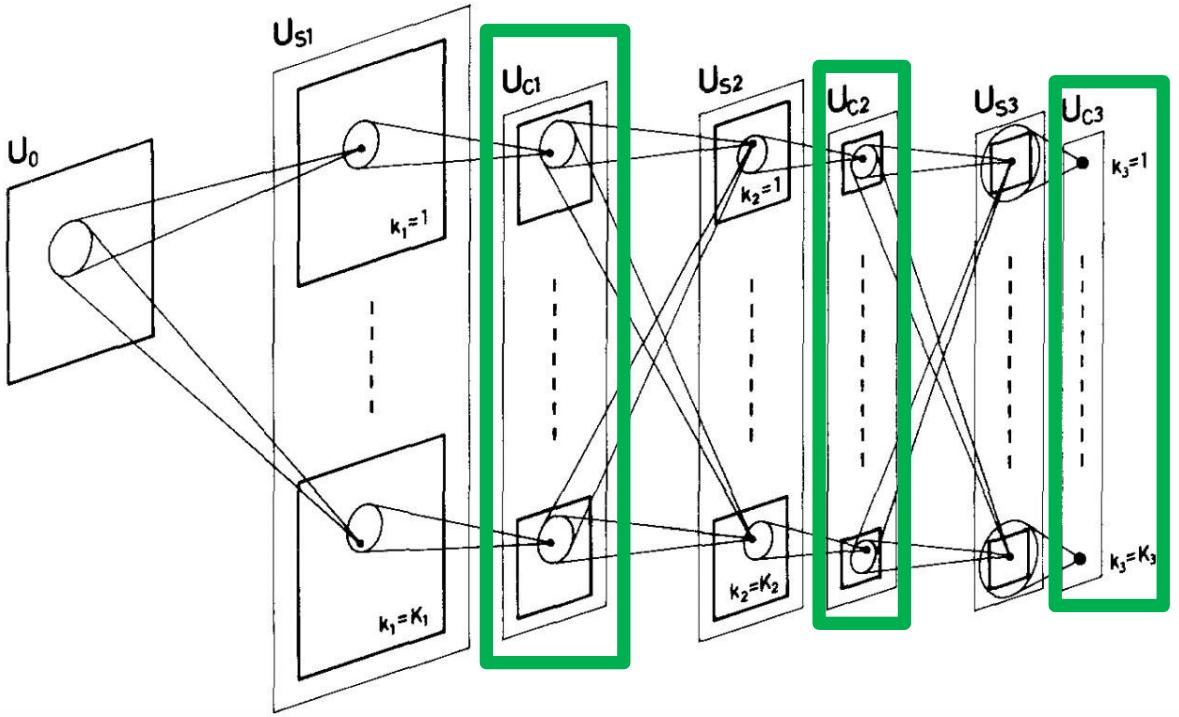


Fig. 2. Schematic diagram illustrating the interconnections between layers in the neocognitron

Fukushima, 1980.

Neocognitron: Key Ingredients

1. Convolutional layers

→ modifiable synapses

→ unmodifiable synapses

2. Pooling Layers

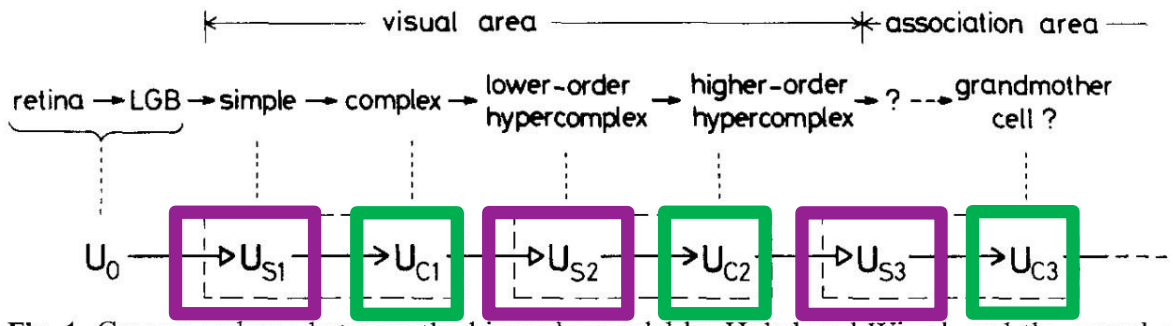


Fig. 1. Correspondence between the hierarchy model by Hubel and Wiesel, and the neural network of the neocognitron

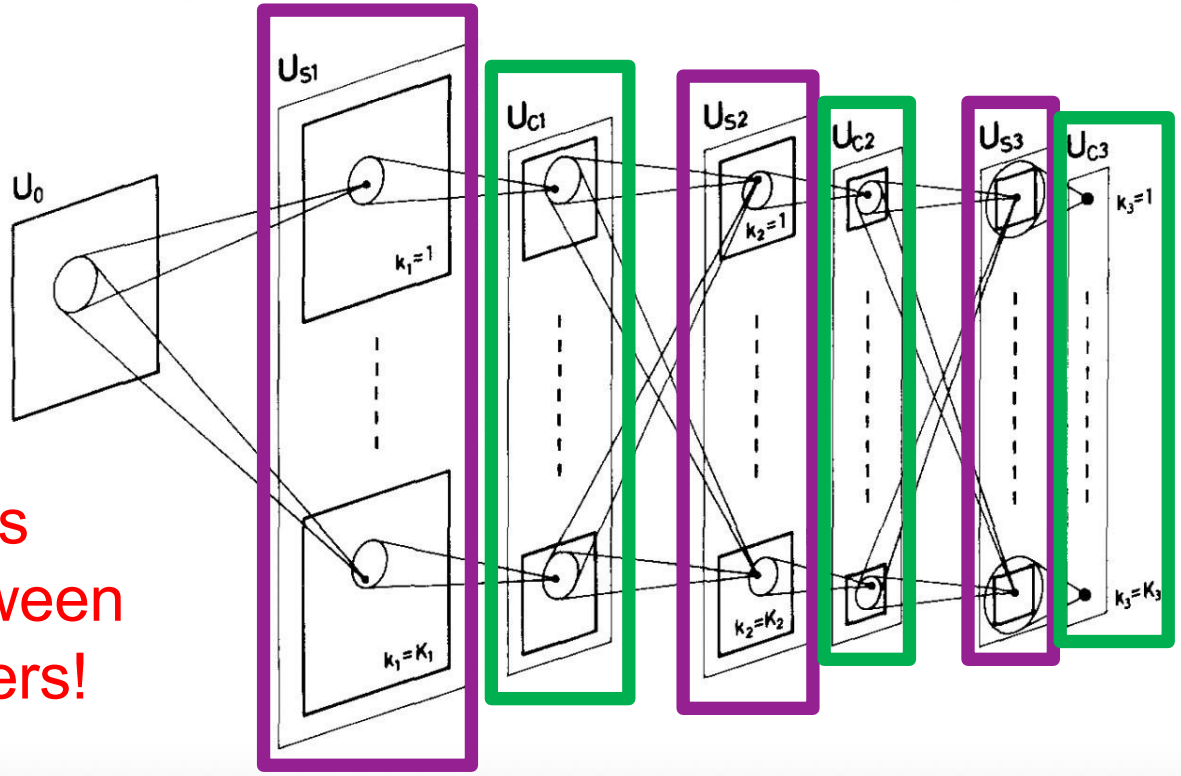


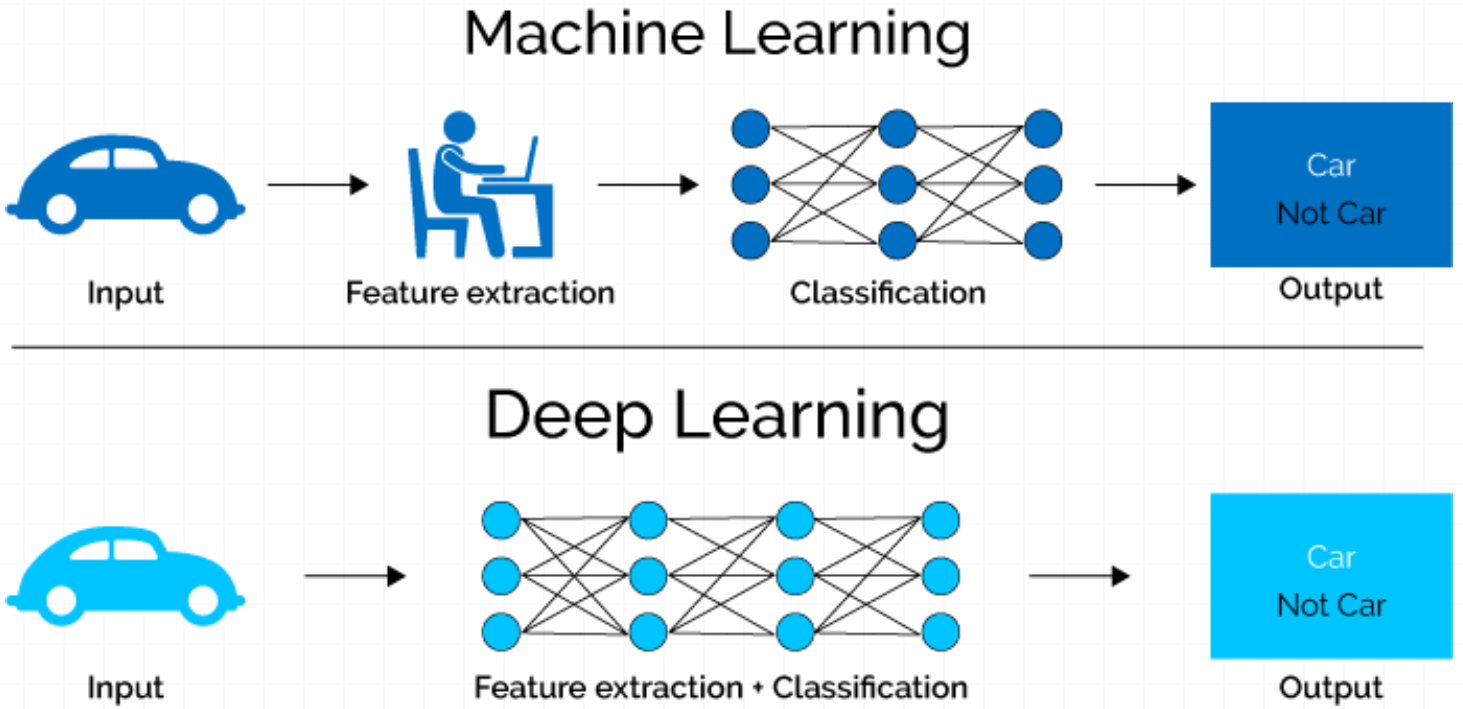
Fig. 2. Schematic diagram illustrating the interconnections between layers in the neocognitron

Note: modern networks similarly alternate between these two types of layers!

Fukushima, 1980.

What is Deep Learning (DL) ?

A machine learning subfield of learning **representations** of data. Exceptional effective at **learning patterns**.
Deep learning algorithms attempt to learn (multiple levels of) representation by using a **hierarchy of multiple layers**
If you provide the system **tons of information**, it begins to understand it and respond in useful ways.



https://www.xenonstack.com/blog/static/public/uploads/media/machine-learning-vs-deep-learning.png

So, 1. **what exactly is deep learning** ?

And, 2. **why is it generally better** than other methods on image, speech and certain other types of data?

So, 1. **what exactly is deep learning** ?

And, 2. **why is it generally better** than other methods on image, speech and certain other types of data?

The short answers

1. **'Deep Learning' means** using a **neural network** with **several layers of nodes** between input and output

2. **the series of layers between input & output do feature identification and processing in a series of stages, just as our brains seem to.**

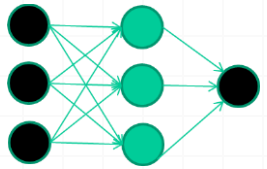
hmmm… OK, but:

3. **multilayer neural networks have been around for 25 years. What's actually new?**

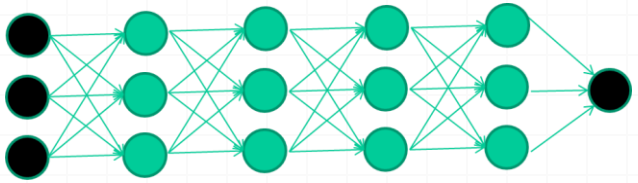
hmmm… OK, but:

3. **multilayer neural networks have been around for 25 years. What's actually new?**

we have always had good algorithms for learning the weights in networks with 1 hidden layer



but these algorithms are not good at learning the weights for networks with more hidden layers

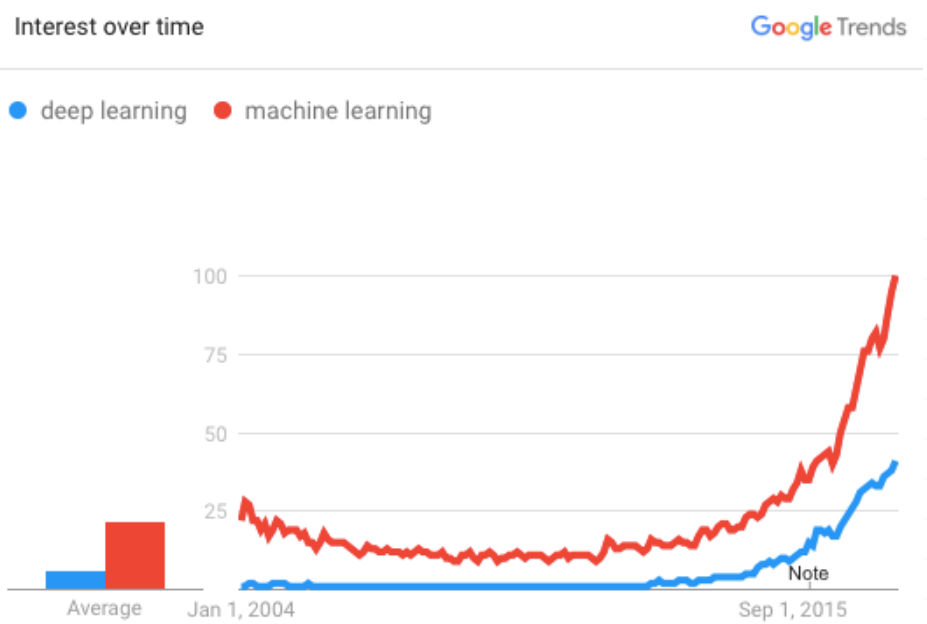


what's new is: <u>algorithms for training many-later networks</u>

Why is DL useful?

- Manually designed features are often **over-specified**, **incomplete** and take a **long time to design** and validate
- Learned Features are **easy to adapt**, **fast** to learn
- Deep learning provides a very **flexible**, (almost?) **universal**, learnable framework for representing world, visual and linguistic information.
- Can learn both unsupervised and supervised
- Effective **end-to-end** joint system learning
- Utilize large amounts of training data

In ~2010 DL started outperforming other ML techniques
first in speech and vision, then NLP



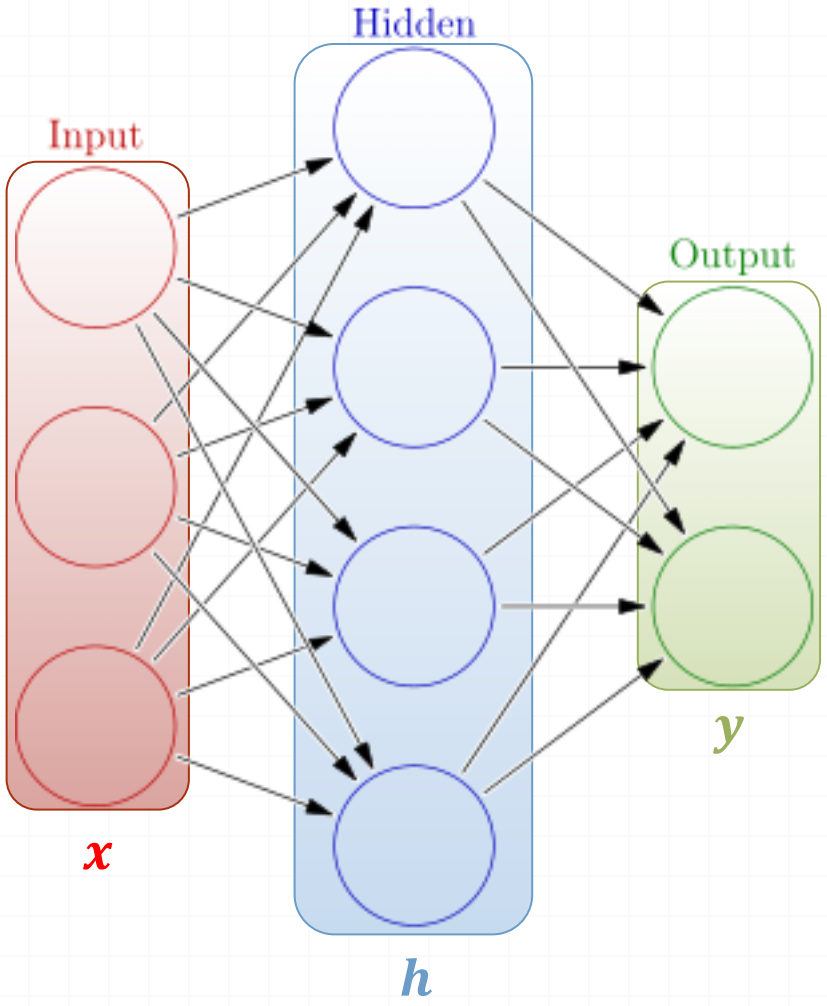
Key components of DL

- Neural Networks,
- Layers,
- Activation Functions,
- Loss Functions,
- Optimizers

Key components of CNNs

- Convolutional Layers (Filters, Strides, Padding)
- Pooling Layers (Max Pooling, Average Pooling)
- Fully Connected Layers

Neural Network Intro



Weights

$$h = \sigma(W_1 x + b_1)$$

$$y = \sigma(W_2 h + b_2)$$

Activation functions

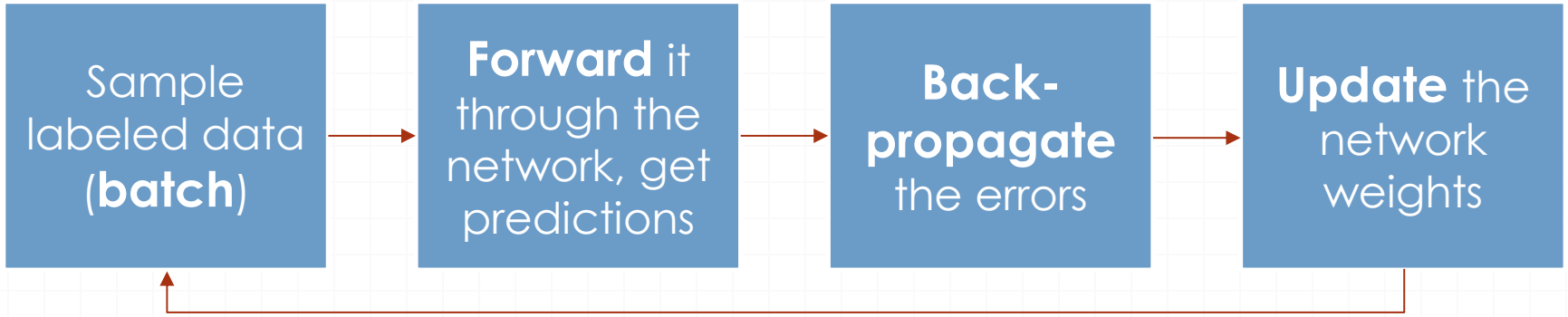
How do we train?

4 + 2 = 6 neurons (not counting inputs)
[3 x 4] + [4 x 2] = 20 weights
4 + 2 = 6 biases

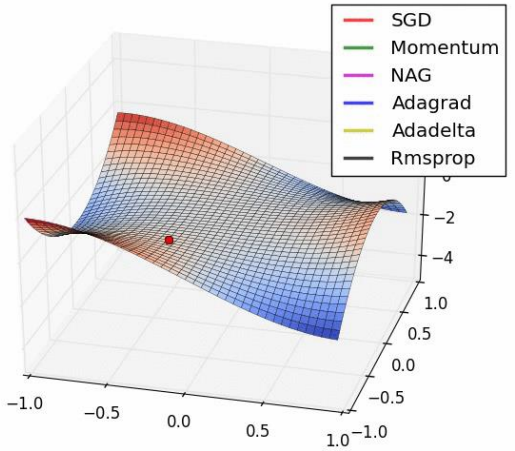
26 learnable **parameters**

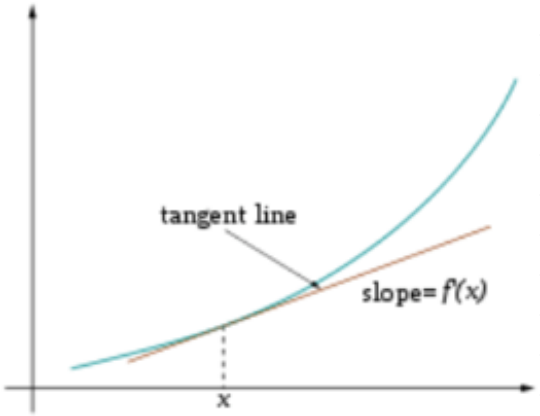
Demo

Training



Optimize (min. or max.) **objective/cost function $J(\theta)$**
Generate **error signal** that measures difference between predictions and target values



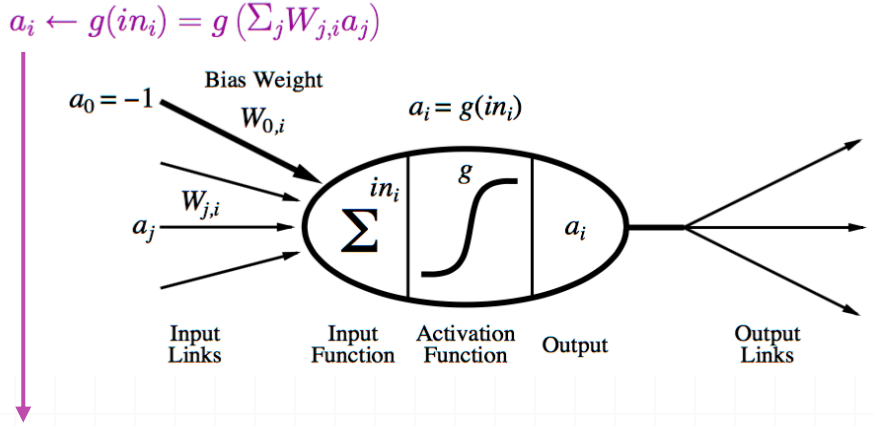


Use error signal to change the **weights** and get more accurate predictions
Subtracting a fraction of the **gradient** moves you towards the **(local) minimum of the cost function**

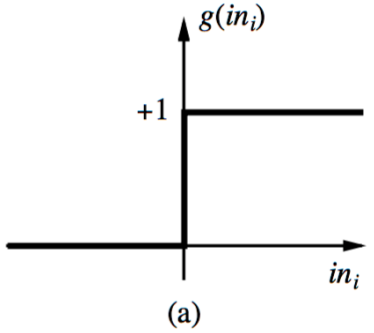
https://medium.com/@ramrajchandradevan/the-evolution-of-gradient-descend-optimization-algorithm-4106a6702d39

Neural Network: A Neuron

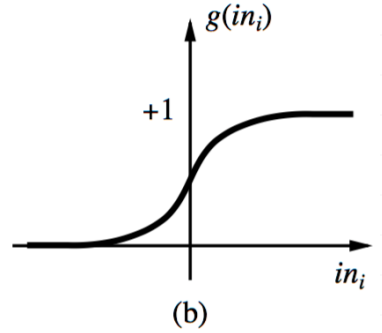
- A neuron is a computational unit in the neural network that exchanges messages with each other



Possible activation functions:

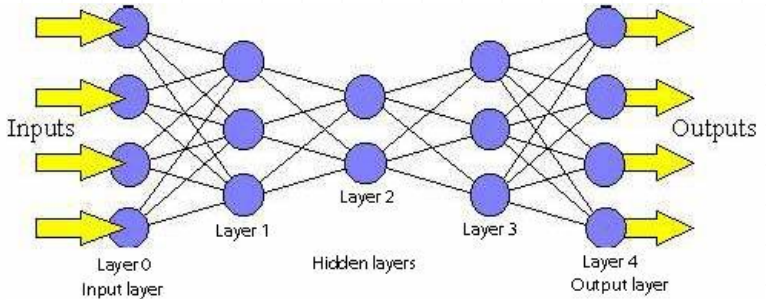


Step function/threshold function



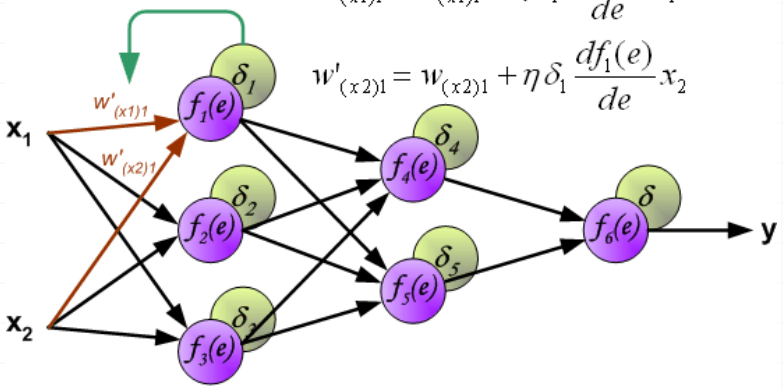
Sigmoid function (a.k.a, logistic function)

Feed forward/Backpropagation Neural Network



$$w'_{(x1)1} = w_{(x1)1} + \eta\delta_1 \frac{df_1(e)}{de} x_1$$

$$w'_{(x2)1} = w_{(x2)1} + \eta\delta_1 \frac{df_1(e)}{de} x_2$$



Feed forward algorithm:

- Activate the neurons from the bottom to the top.

Backpropagation:

- Randomly initialize the parameters
- Calculate total error at the top, $f_6(e)$
- Then calculate contributions to error, δ_n, at each step going backwards.

Limitations of Neural Networks

Random initialization + **densely connected networks** lead to:

- High cost
  - Each neuron in ANN can be considered as logistic regression.
  - Training entire ANN is to train all interconnected logistic regressions.
- Difficult to train as the number of hidden layers increases
  - Recall that logistic regression is trained by gradient descent.
  - In backpropagation, gradient is progressively getting more dilute. That is, below top layers, the correction signal δ_n is minimal.
- Stuck in local optima
  - The objective function of the neural network is usually not convex.
  - The random initialization does not guarantee starting from the proximity of global optima.
- Solution:
  - Deep Learning/Learning multiple levels of representation

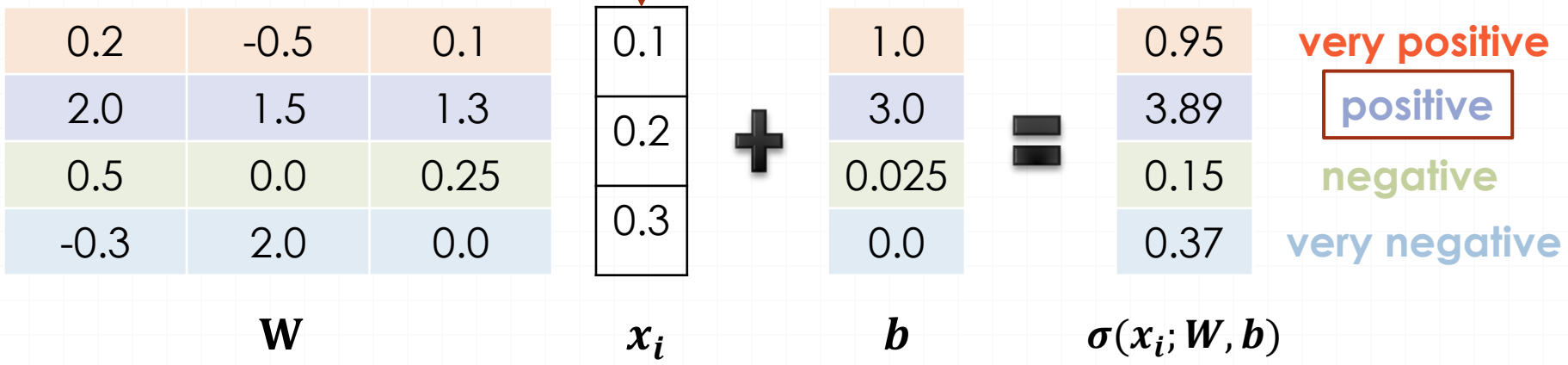
One forward pass

Text (input) representation

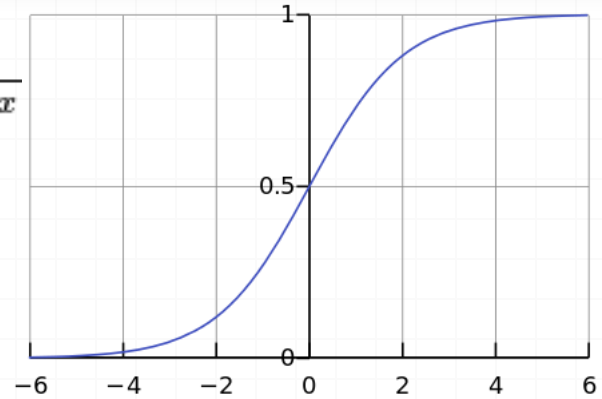
TFIDF

Word embeddings

....



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



-0.06

W1

-2.5

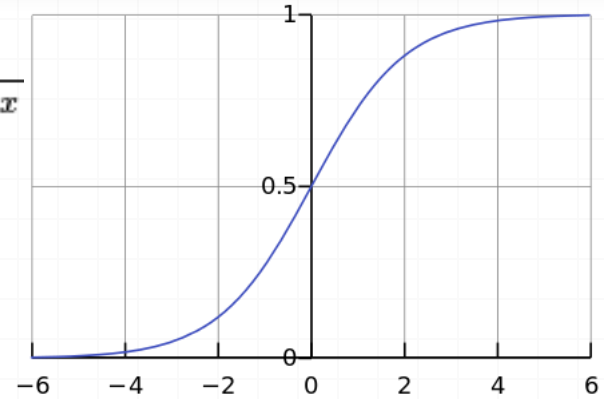
W2

W3

1.4

f(x)

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



-0.06

2.7

-2.5

-8.6

0.002

1.4

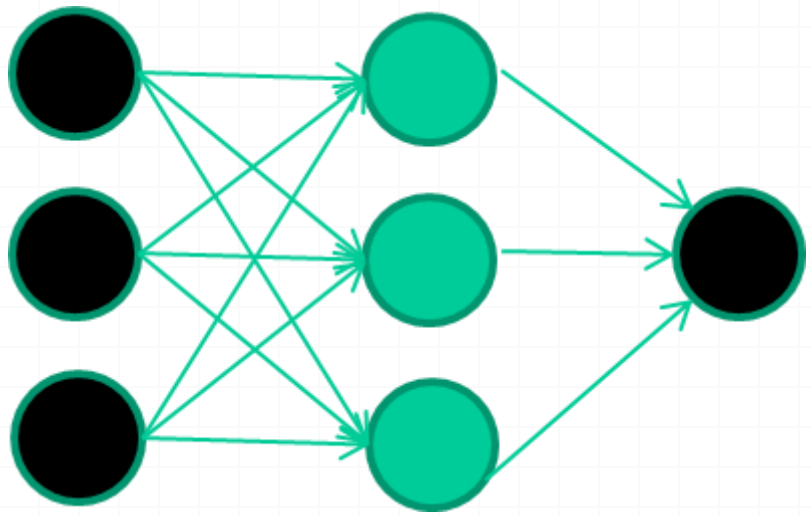
f(x)

$x =$ -0.06×2.7 + 2.5×8.6 + 1.4×0.002 = 21.34

A dataset

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

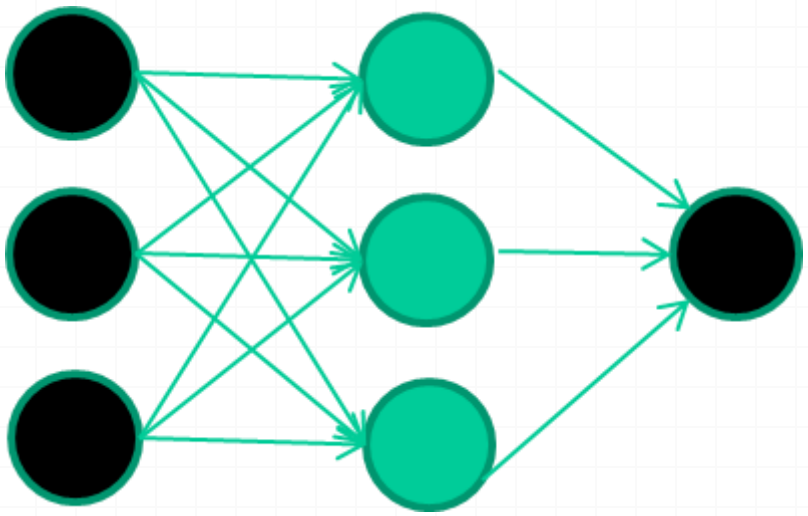
etc …



Training the neural network

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …

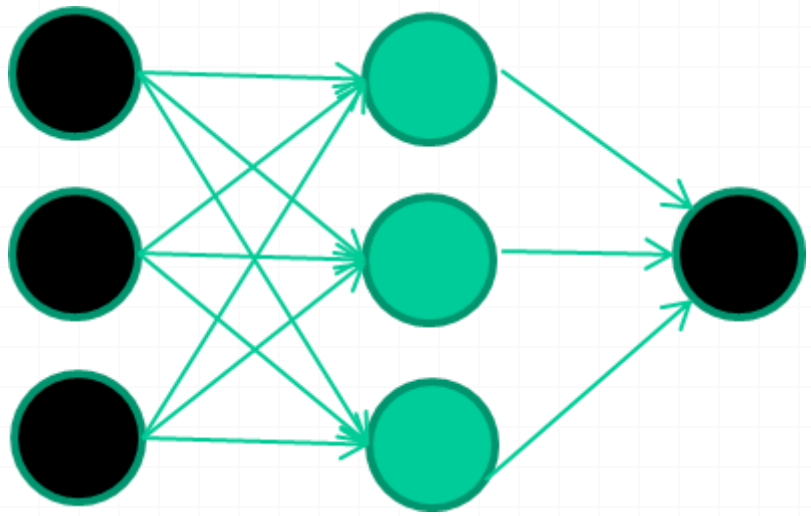


Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …

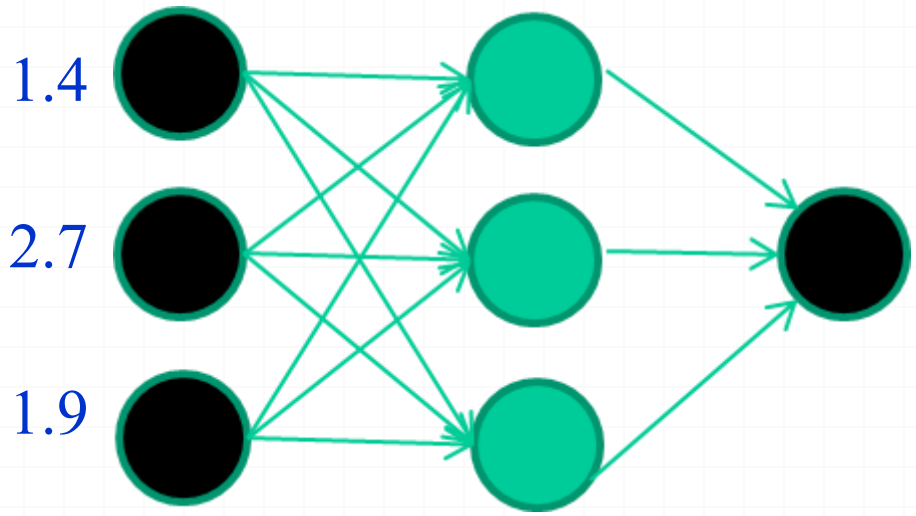
Initialise with random weights



Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

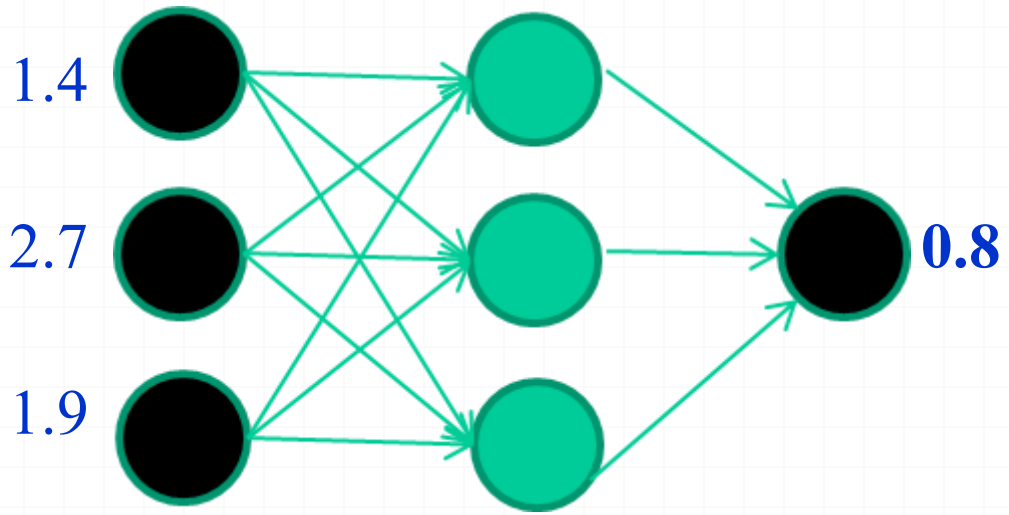
Present a training pattern



Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

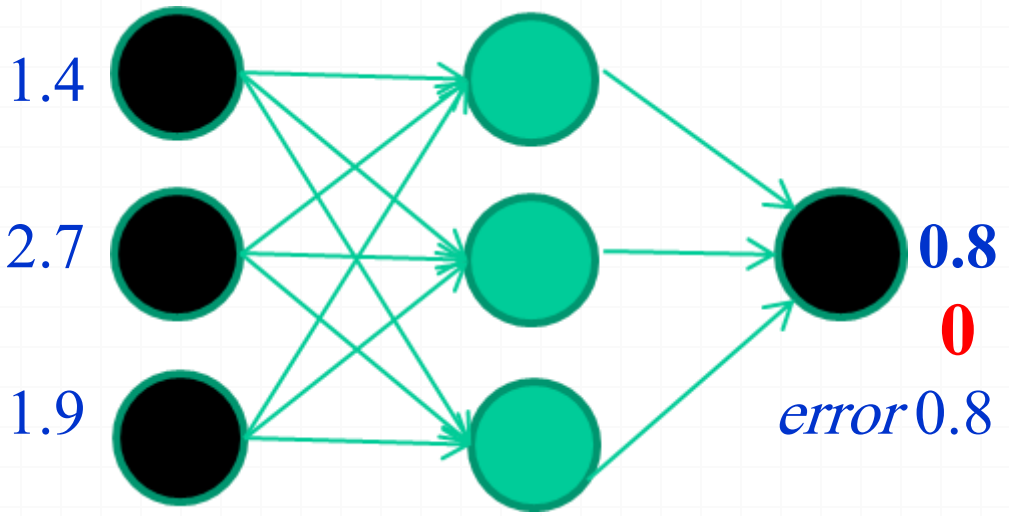
Feed it through to get output



Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

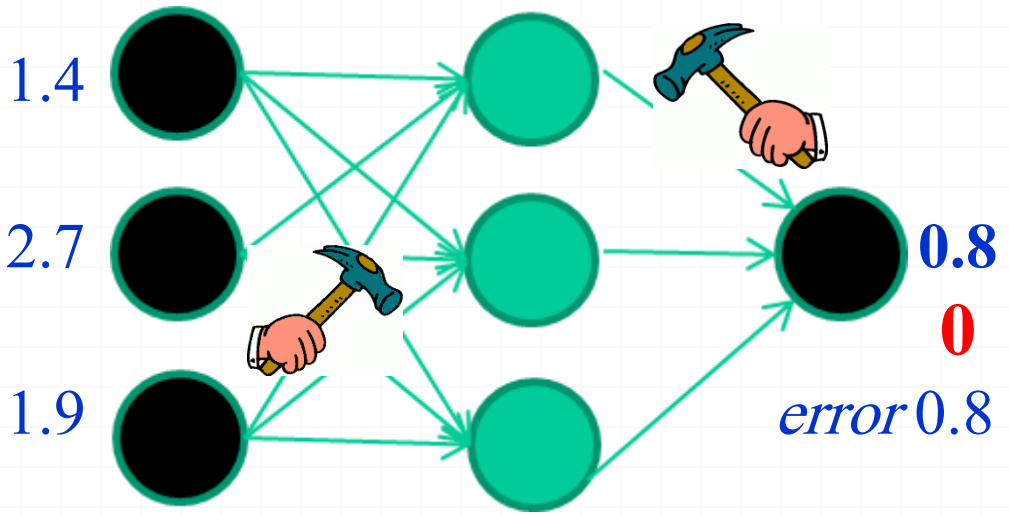
Compare with target output



Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Adjust weights based on error

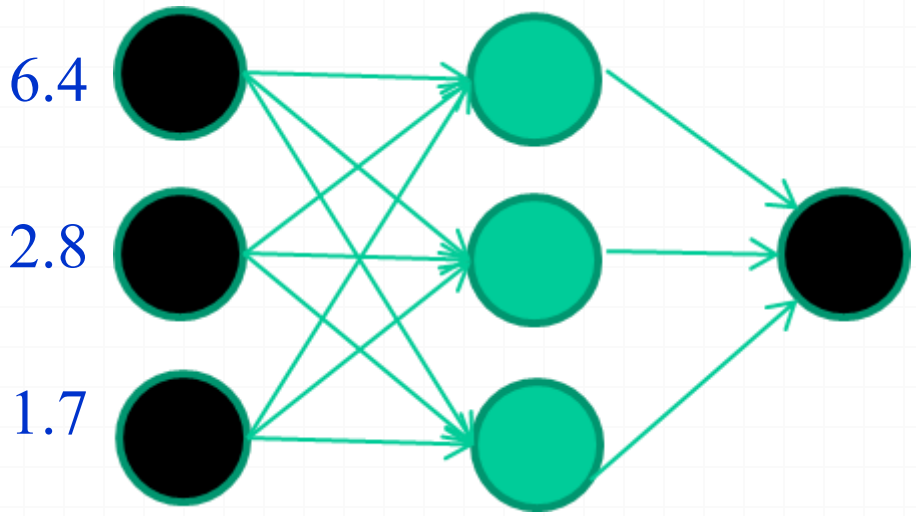


Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …

Present a training pattern

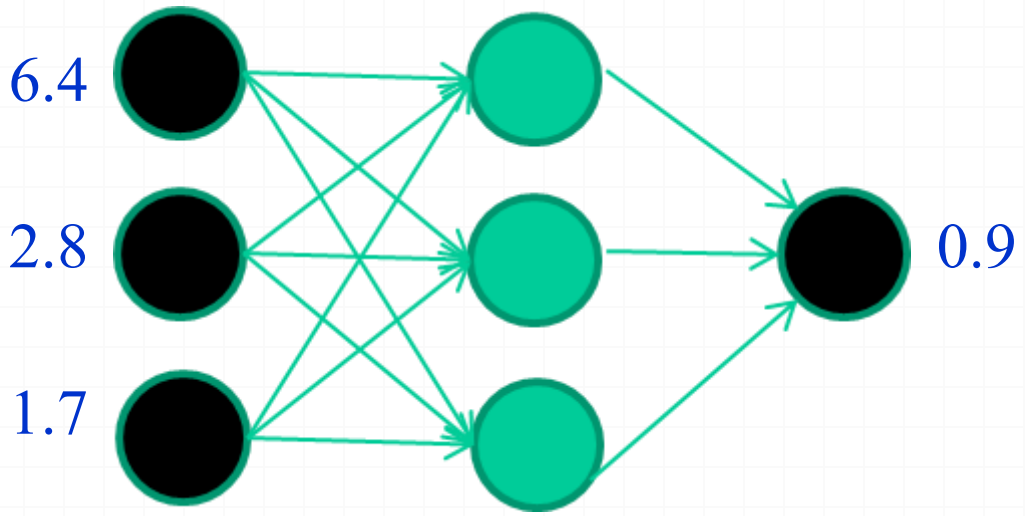


Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …

Feed it through to get output

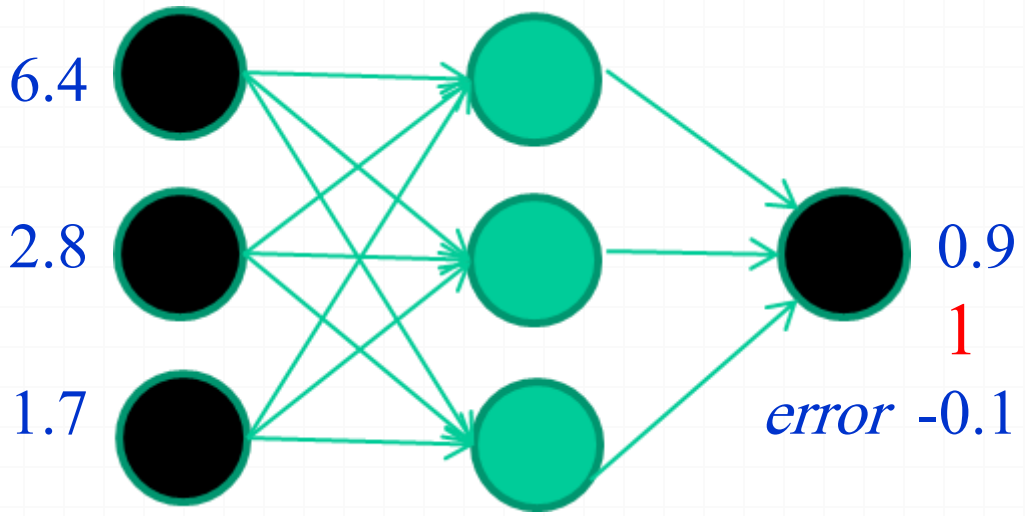


Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …

Compare with target output

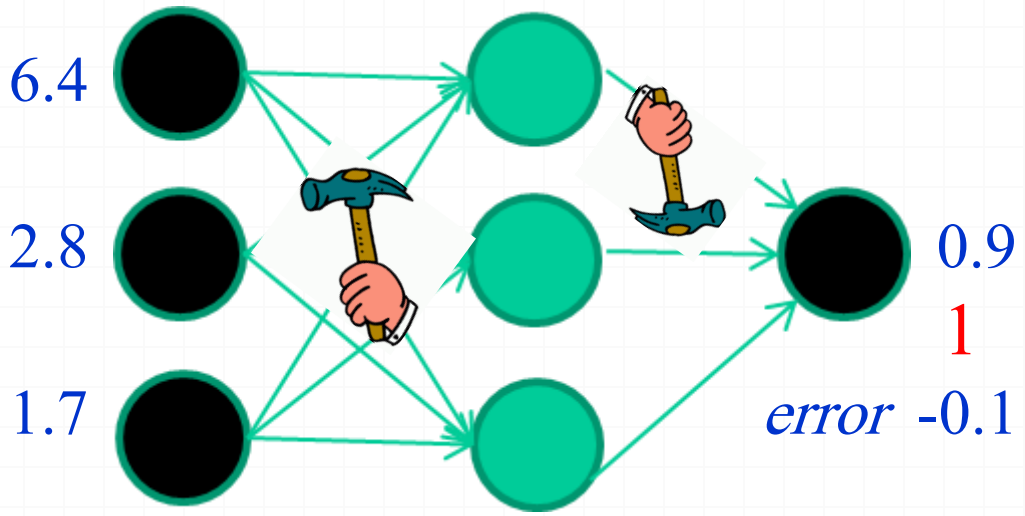


Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …

Adjust weights based on error

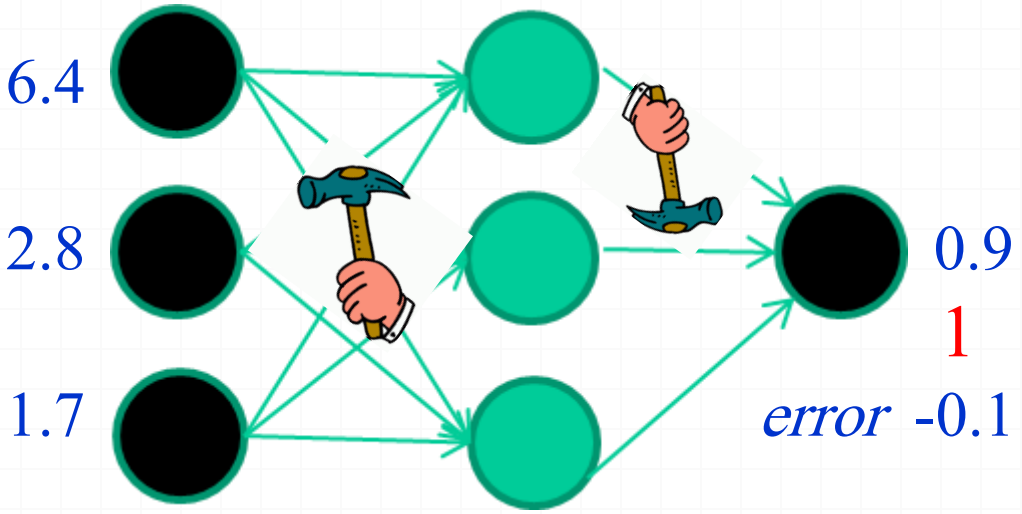


Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …

And so on ….

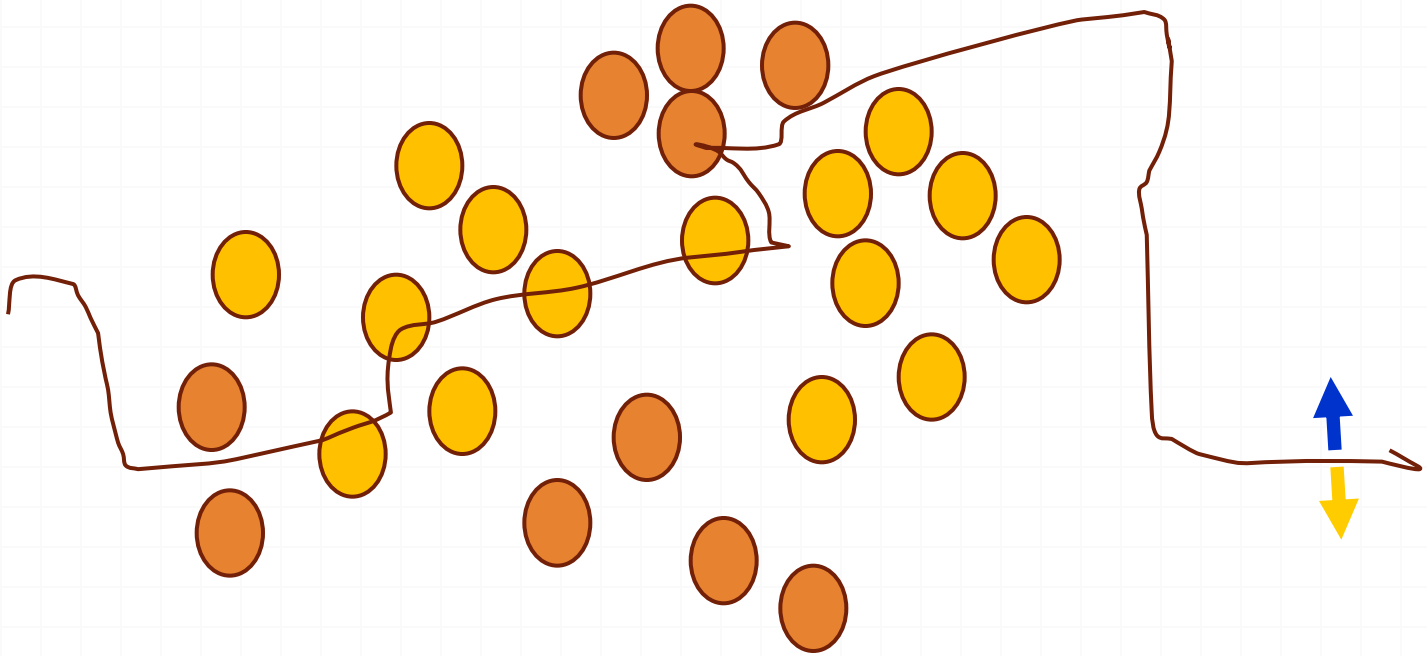


Repeat this thousands, maybe millions of times – each time taking a random training instance, and making slight weight adjustments

Algorithms for weight adjustment are designed to make changes that will reduce the error

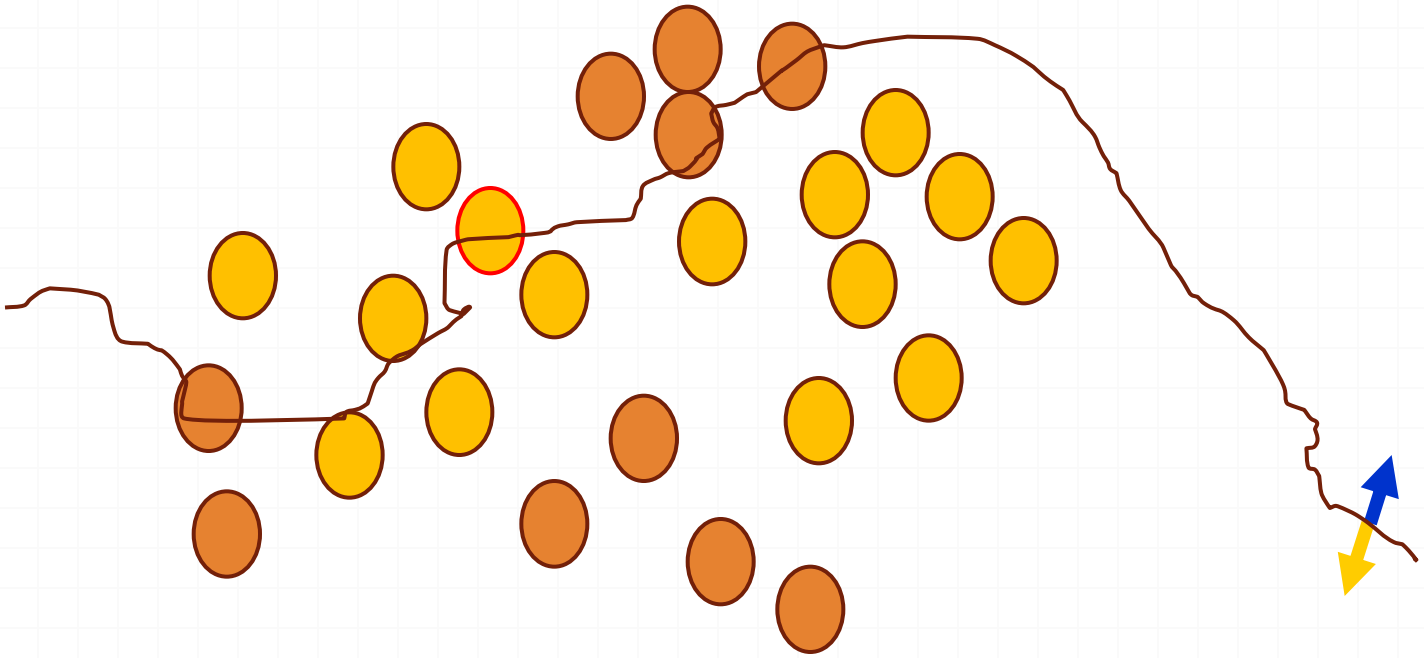
The decision boundary perspective…

Initial random weights



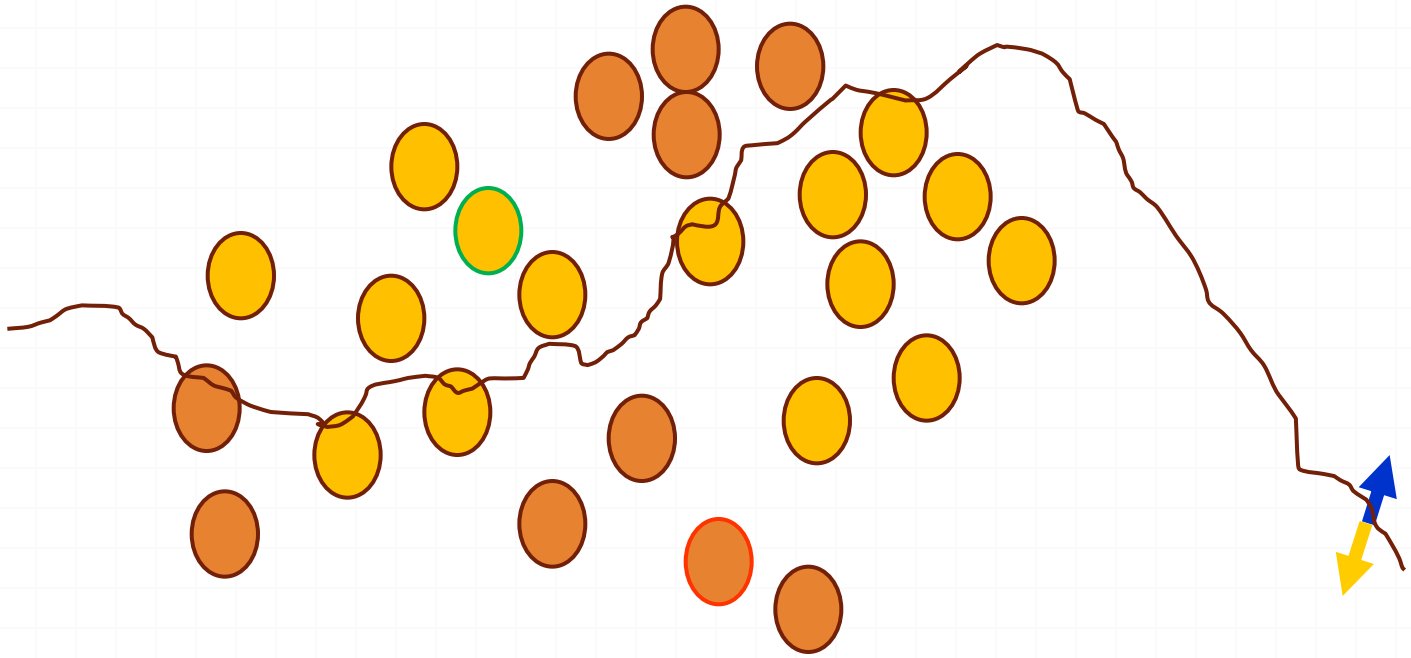
The decision boundary perspective…

Present a training instance / adjust the weights



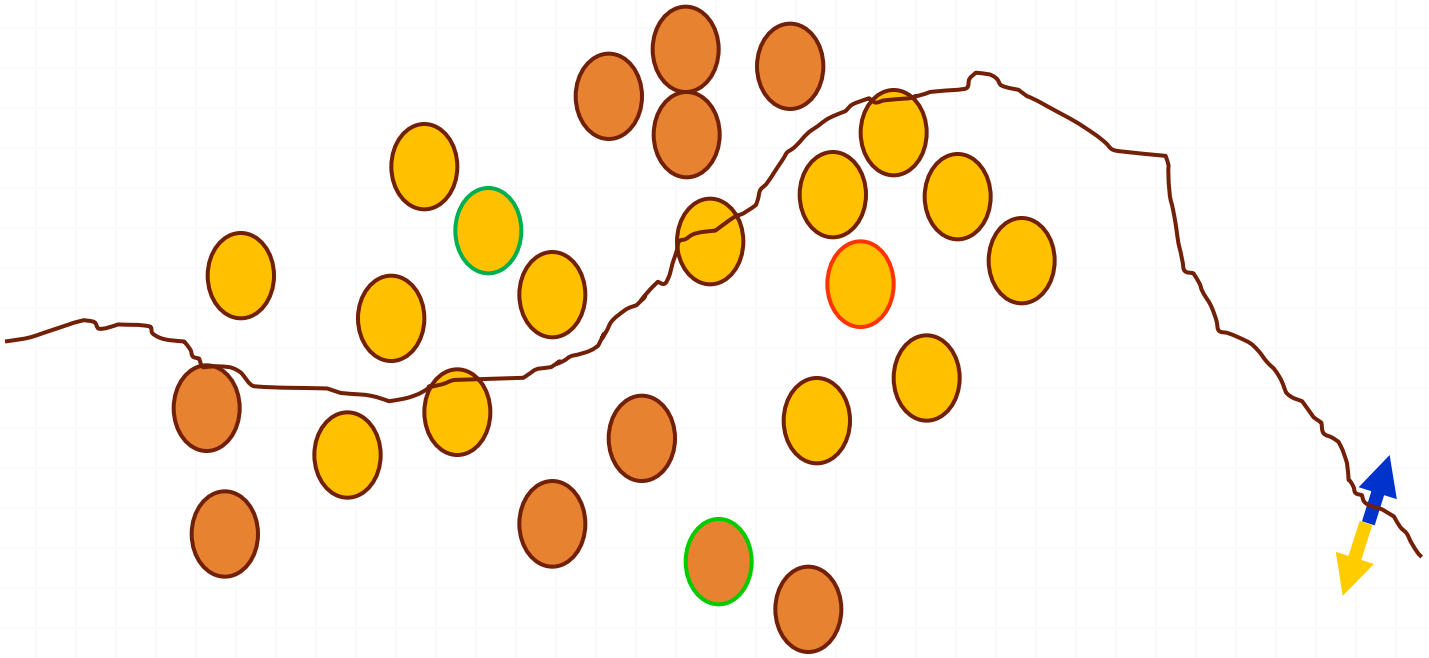
The decision boundary perspective…

Present a training instance / adjust the weights



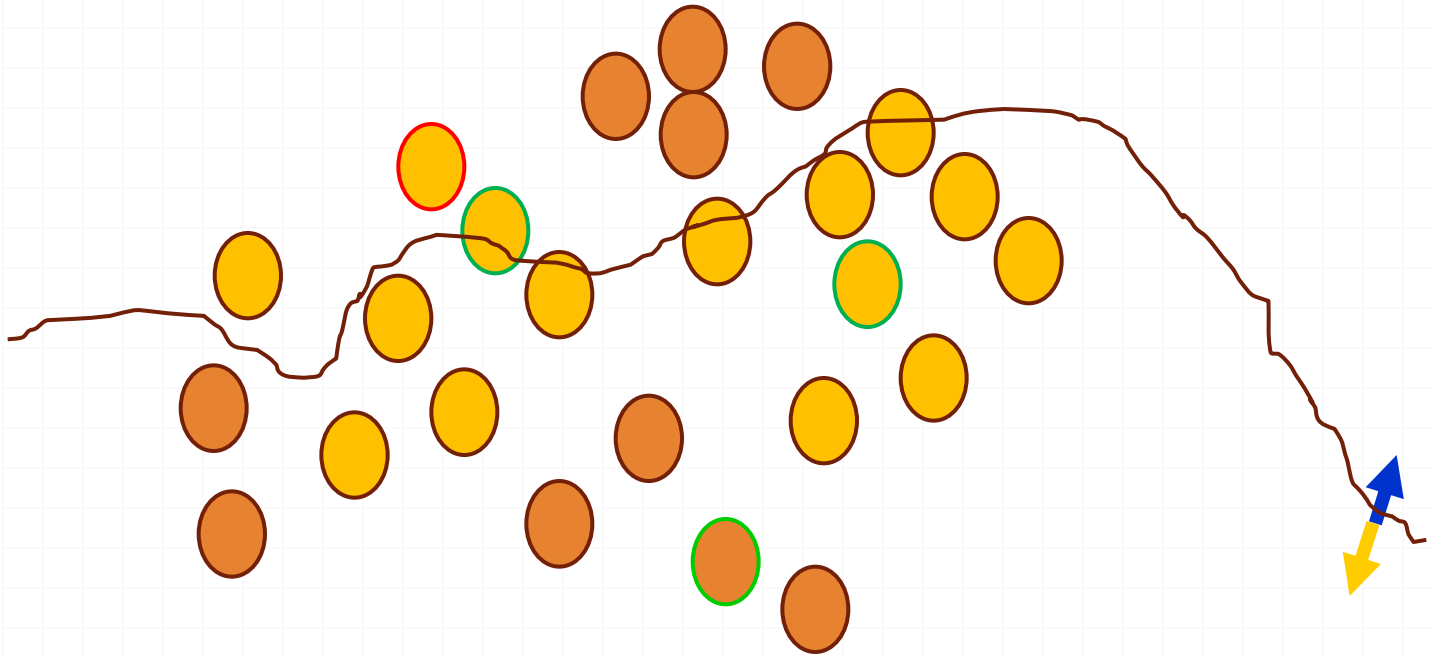
The decision boundary perspective…

Present a training instance / adjust the weights



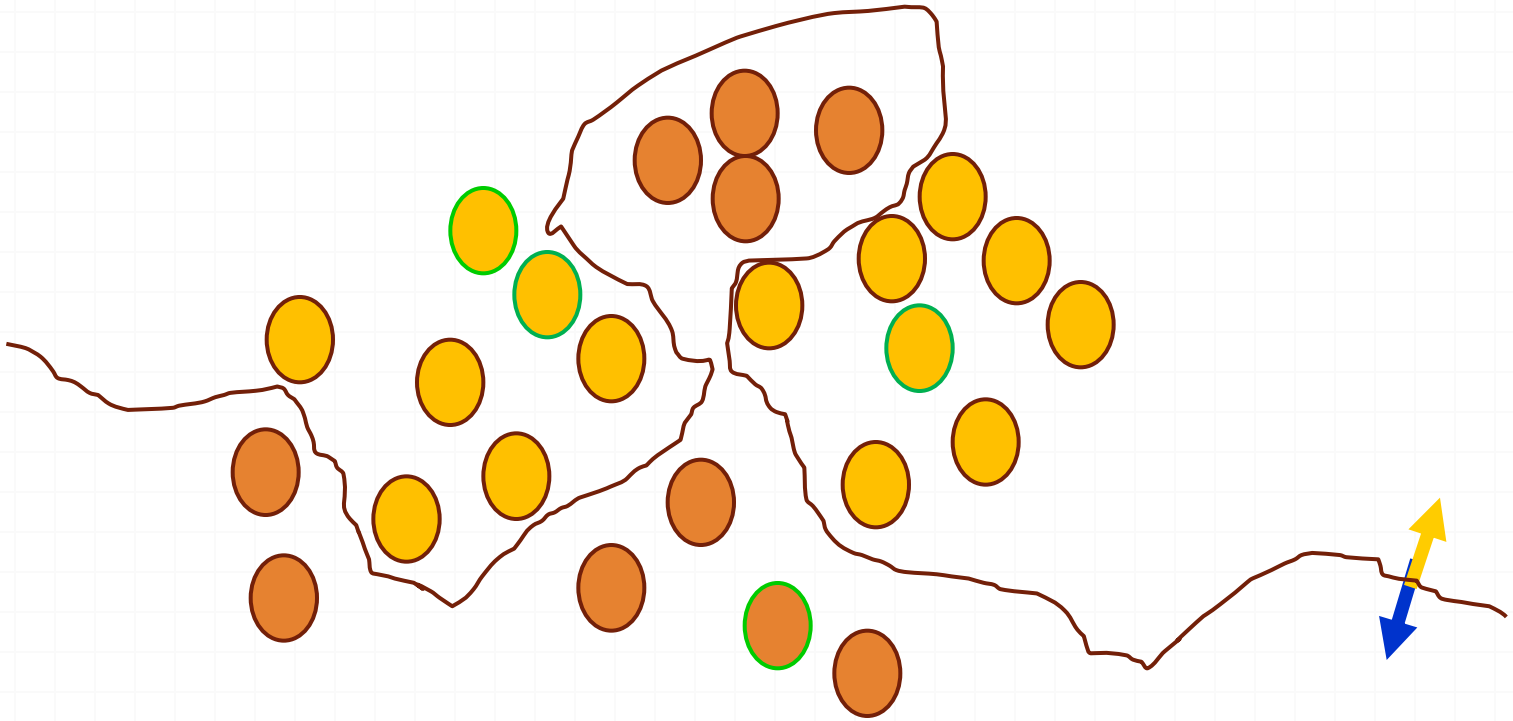
The decision boundary perspective…

Present a training instance / adjust the weights



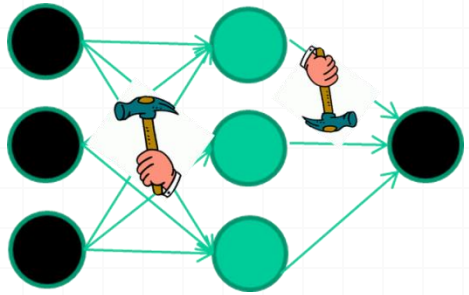
The decision boundary perspective…

Eventually ….



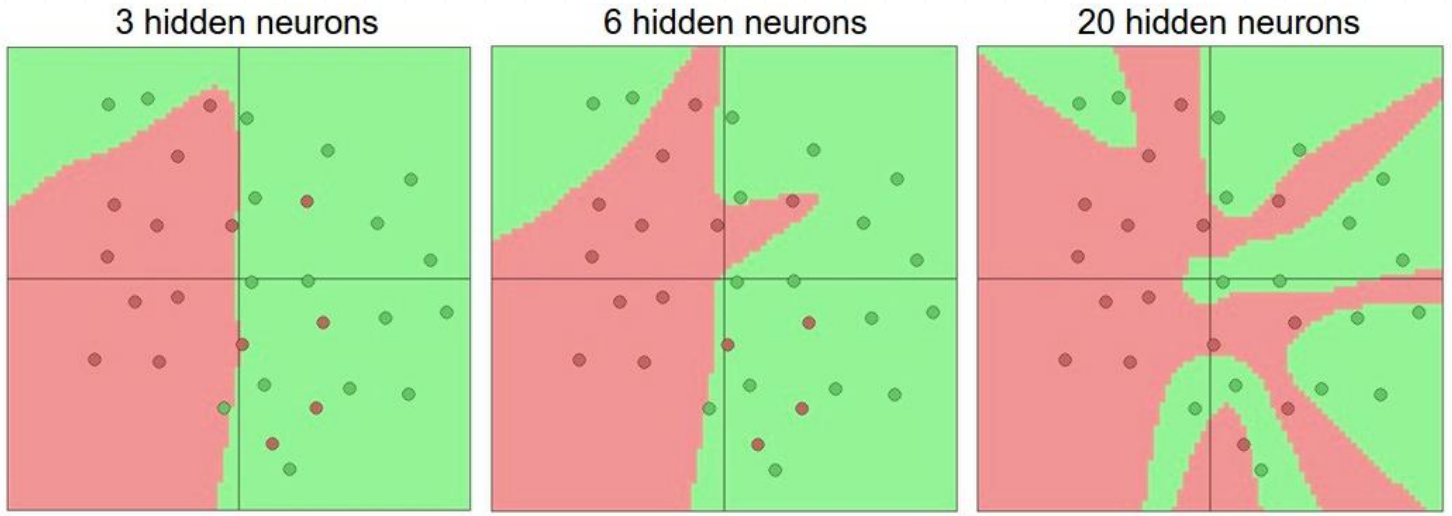
The point I am trying to make

- weight-learning algorithms for NNs are dumb
- they work by making thousands and thousands of tiny adjustments, each making the network do better at the most recent pattern, but perhaps a little worse on many others
- but, by dumb luck, eventually this tends to be good enough to learn effective classifiers for many real applications



Activation functions

Non-linearities needed to learn complex (non-linear) representations of data, otherwise the NN would be just a linear function $W_1 W_2 x = Wx$

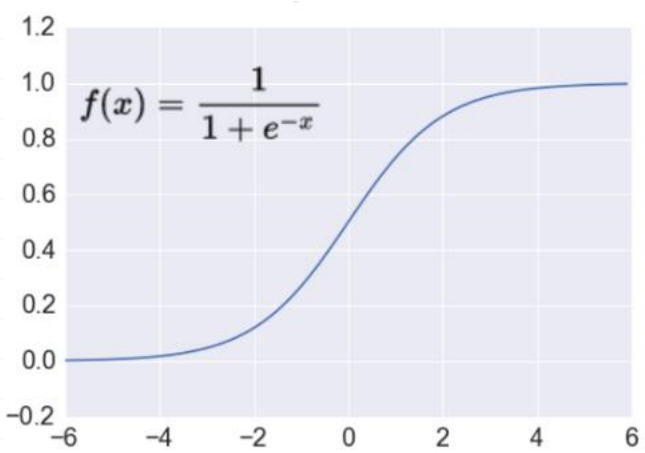


http://cs231n.github.io/assets/nn1/layer_sizes.jpeg

More layers and neurons can approximate more complex functions

Full list: https://en.wikipedia.org/wiki/Activation_function

Activation: Sigmoid



http://adilmoujahid.com/images/activation.png

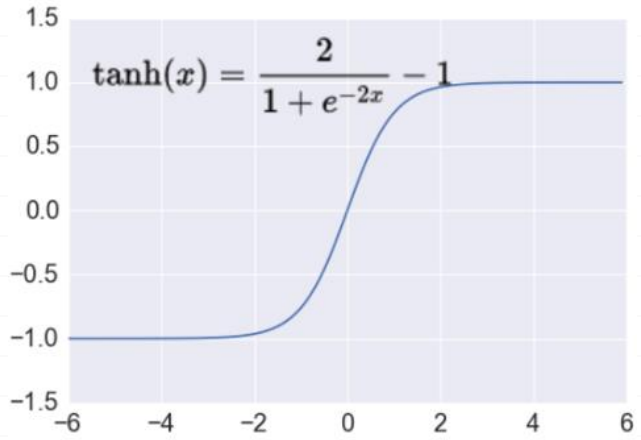
Takes a real-valued number and "squashes" it into range between 0 and 1.

$$R^n \rightarrow [0,1]$$

\+ Nice interpretation as the **firing rate** of a neuron
- 0 = not firing at all
- 1 = fully firing

\- Sigmoid neurons **saturate** and **kill gradients**, thus NN will barely learn
- when the neuron's activation are 0 or 1 (saturate)
  - 😟 gradient at these regions almost zero
  - 😟 almost no signal will flow to its weights
  - 😟 if initial weights are too large then most neurons would saturate

Activation: Tanh



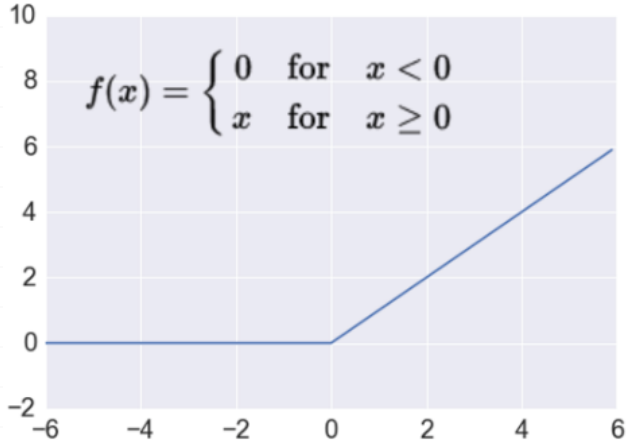
http://adilmoujahid.com/images/activation.png

Takes a real-valued number and "squashes" it into range between -1 and 1.

$$R^n \rightarrow [-1,1]$$

- Like sigmoid, tanh neurons **saturate**
- Unlike sigmoid, output is **zero-centered**
- Tanh is a **scaled sigmoid**: $\tanh(x) = 2sigm(2x) - 1$

Activation: ReLU



http://adilmoujahid.com/images/activation.png

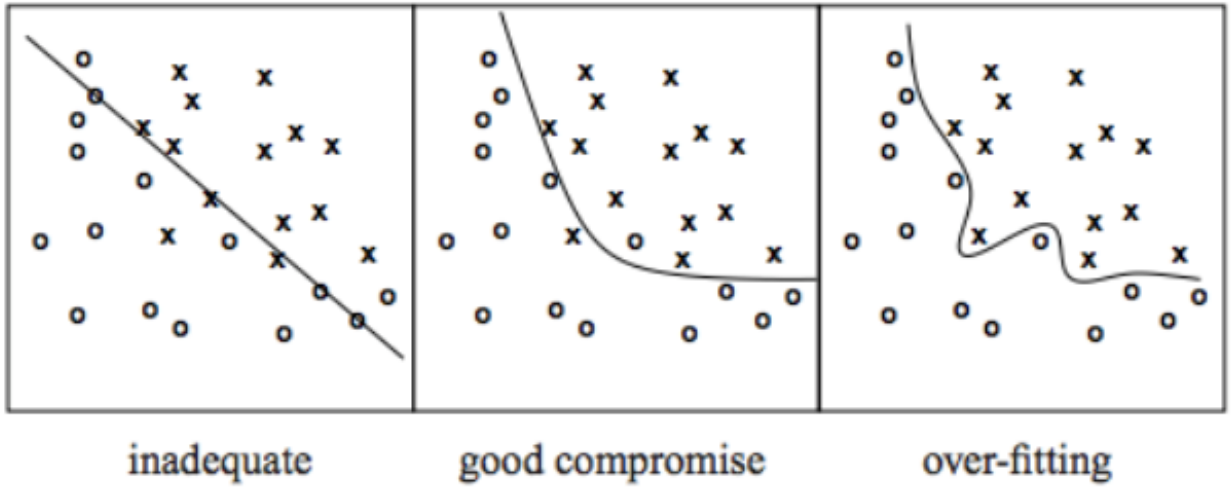
Takes a real-valued number and thresholds it at zero $f(x) = \max(0, x)$

$$R^n \rightarrow R^n_+$$

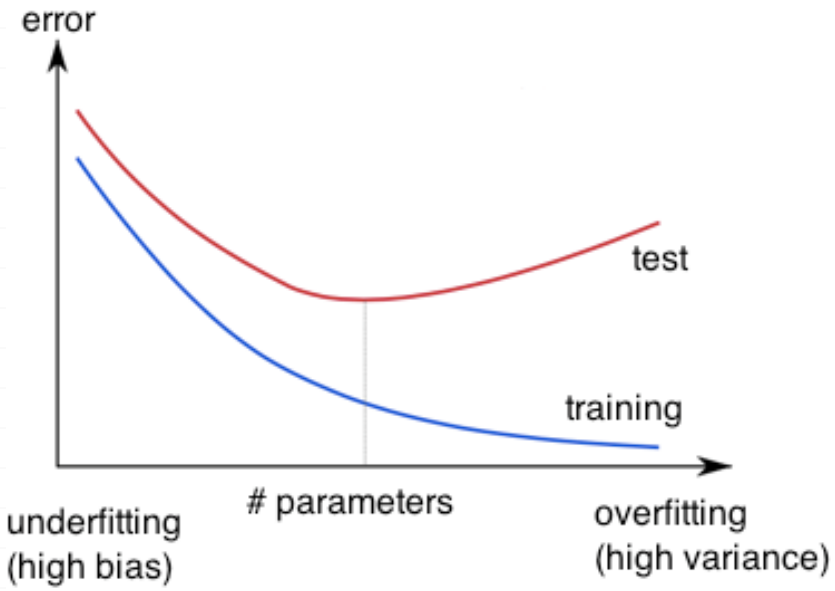
Most Deep Networks use ReLU nowadays

- Trains much **faster**
  - accelerates convergence of (Stochastic gradient descent) SGD
  - due to linear, non-saturating form
- Less expensive operations
  - compared to sigmoid/tanh (exponentials etc.)
  - implemented by simply thresholding a matrix at zero
- More **expressive**
- Prevents the **gradient vanishing problem**

Overfitting



http://wiki.bethanycrane.com/overfitting-of-data



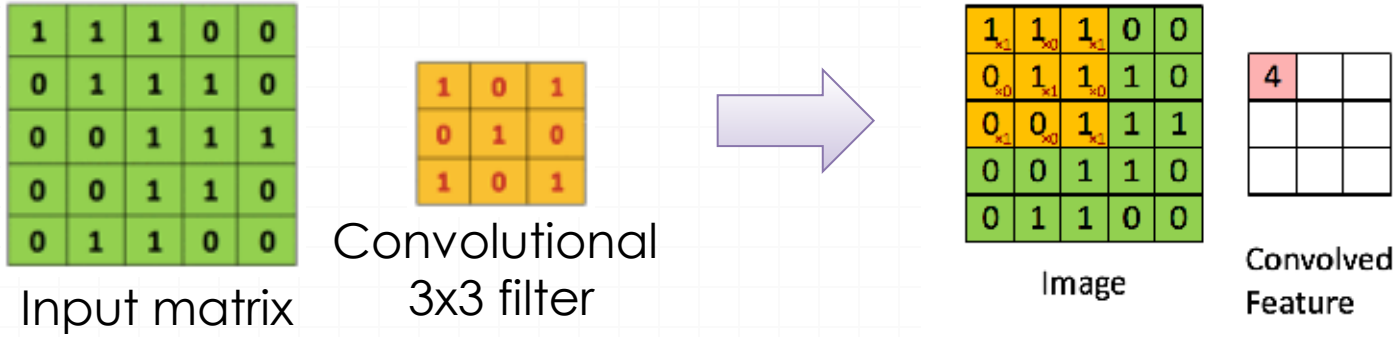
https://www.neuraldesigner.com/images/learning/selection_error.svg

Learned hypothesis may **fit** the training data very well, even outliers (**noise**) but fail to **generalize** to new examples (test data)

Convolutional Neural Networks (CNNs)

Main CNN idea for text:
Compute vectors for n-grams and group them afterwards

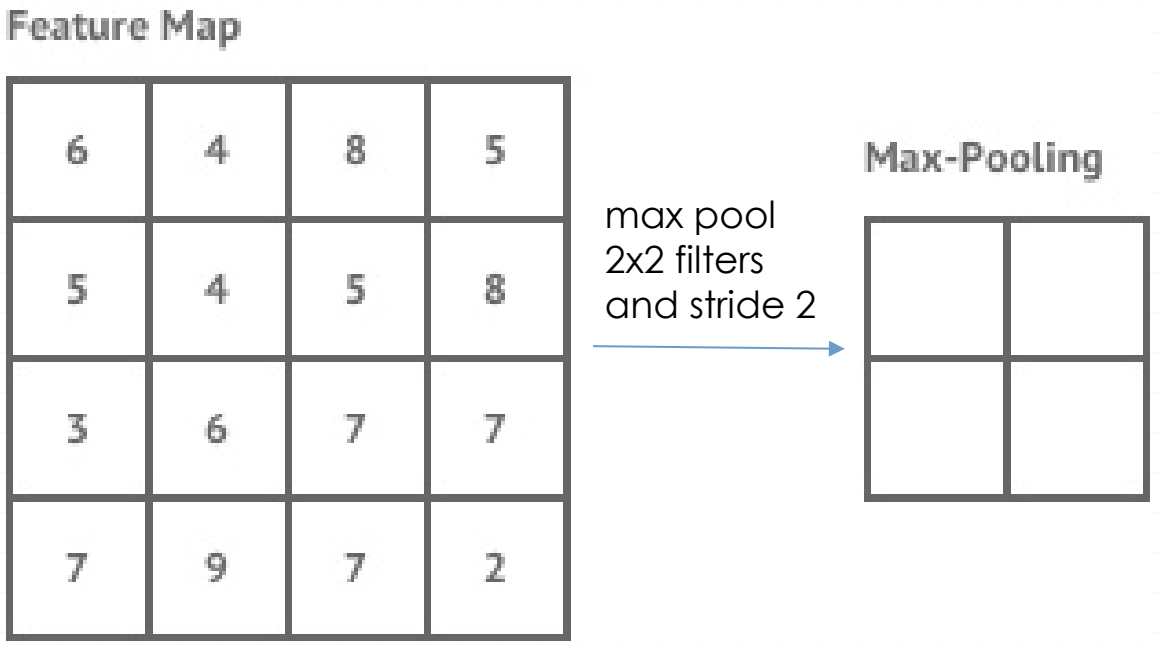
Example: "this takes too long" compute vectors for:
This takes, takes too, too long, this takes too, takes too long, this takes too long



http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

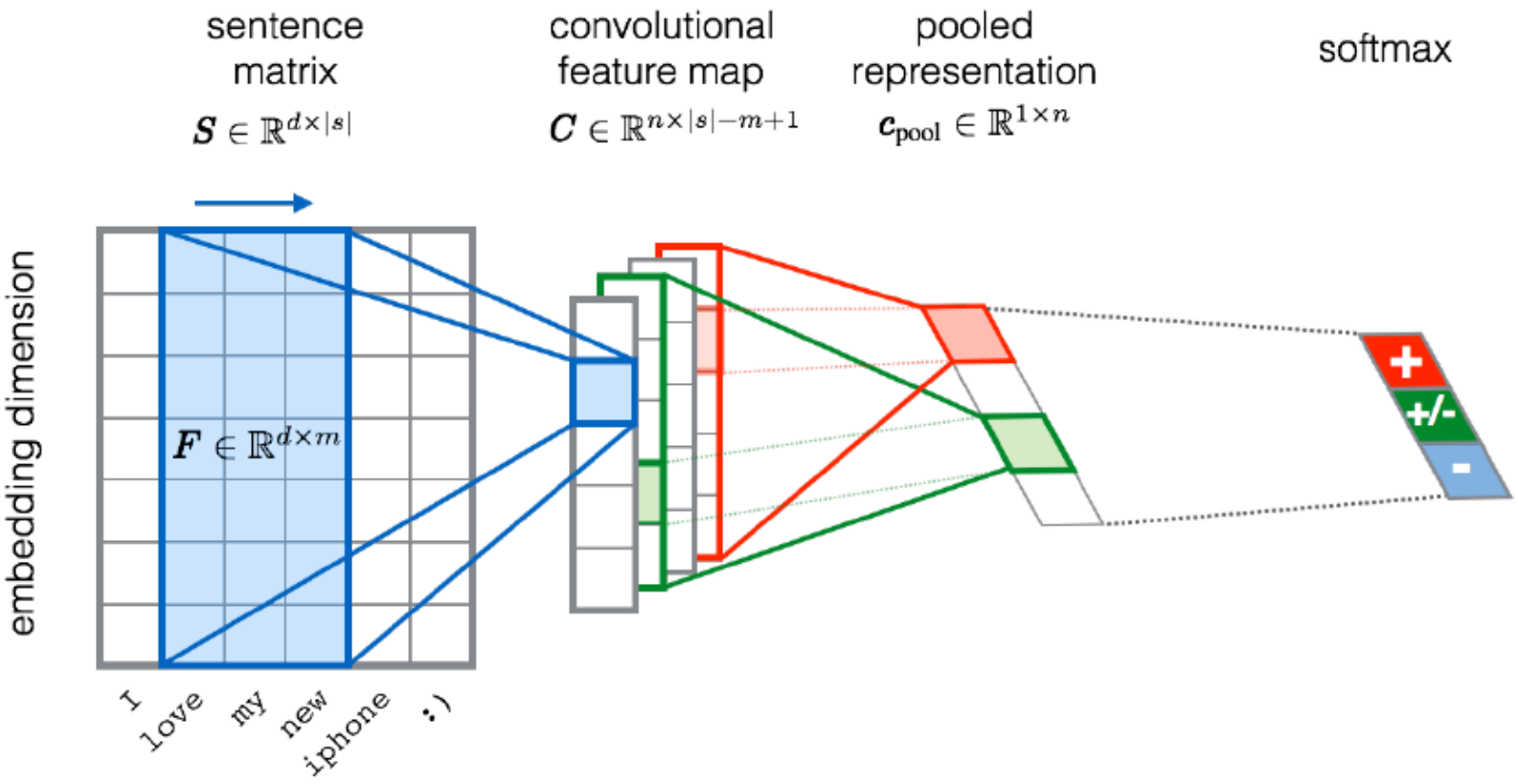
Convolutional Neural Networks (CNNs)

Main CNN idea for text:
Compute vectors for n-grams and **group them afterwards**



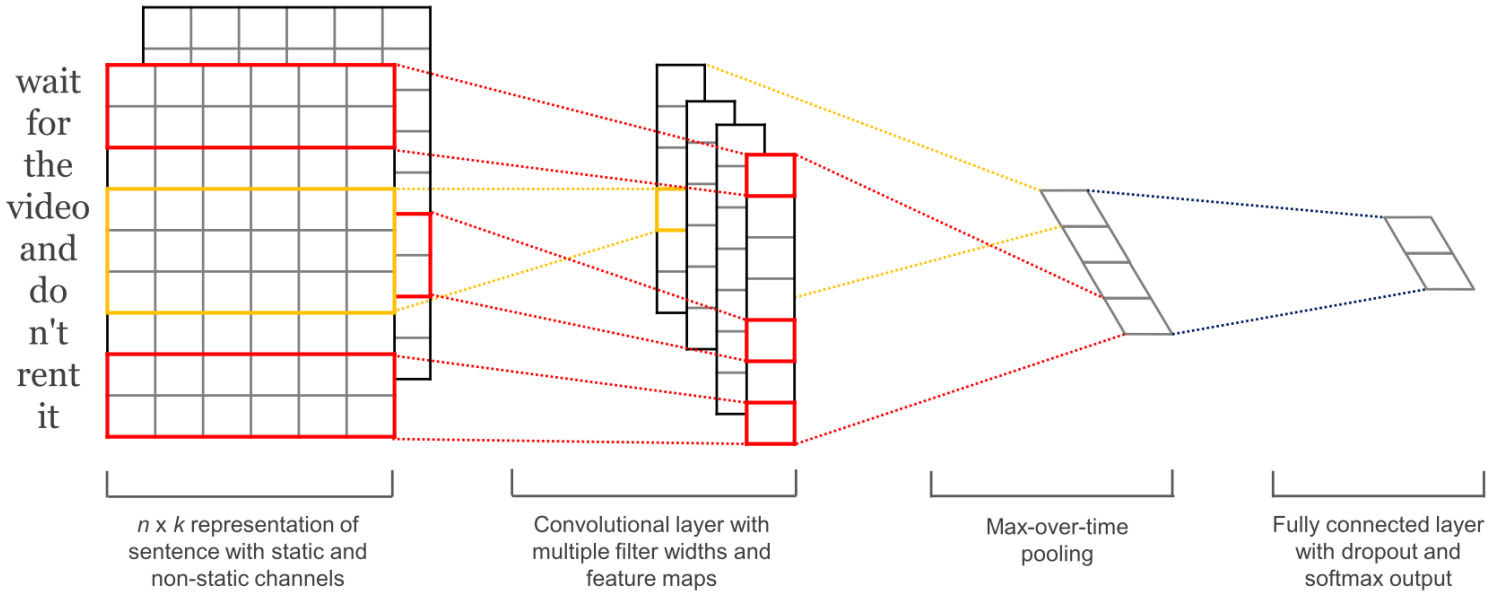
https://shafeentejani.github.io/assets/images/pooling.gif

CNN for text classification



Severyn, Aliaksei, and Alessandro Moschitti. "UNITN: Training Deep Convolutional Neural Network for Twitter Sentiment Classification." *SemEval@ NAACL-HLT*. 2015.

CNN with multiple filters

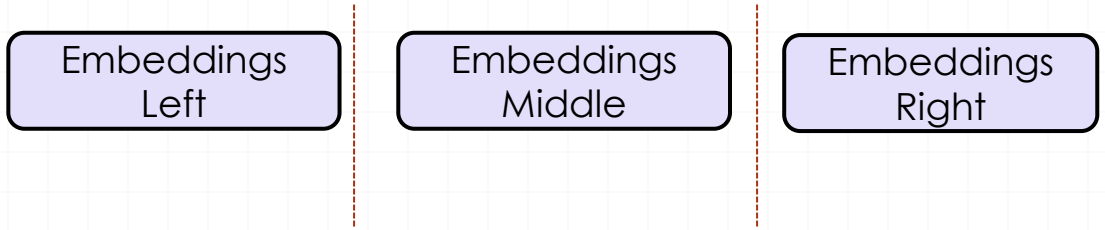


Kim, Y. "Convolutional Neural Networks for Sentence Classification", EMNLP (2014)

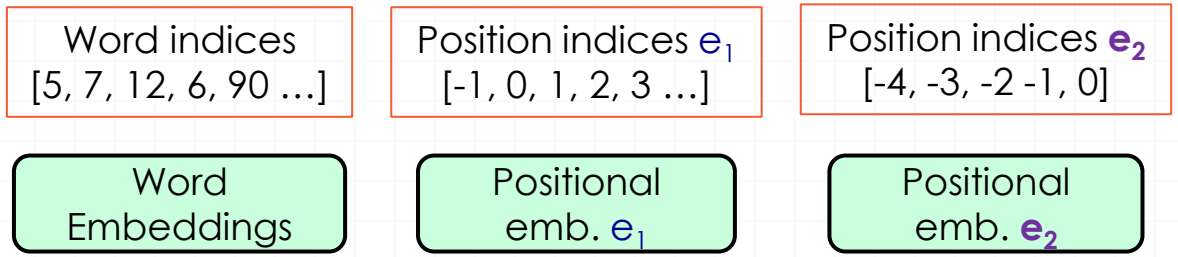
sliding over 3, 4 or 5 words at a time

Features / Input representation

1) context-wise split of the sentence

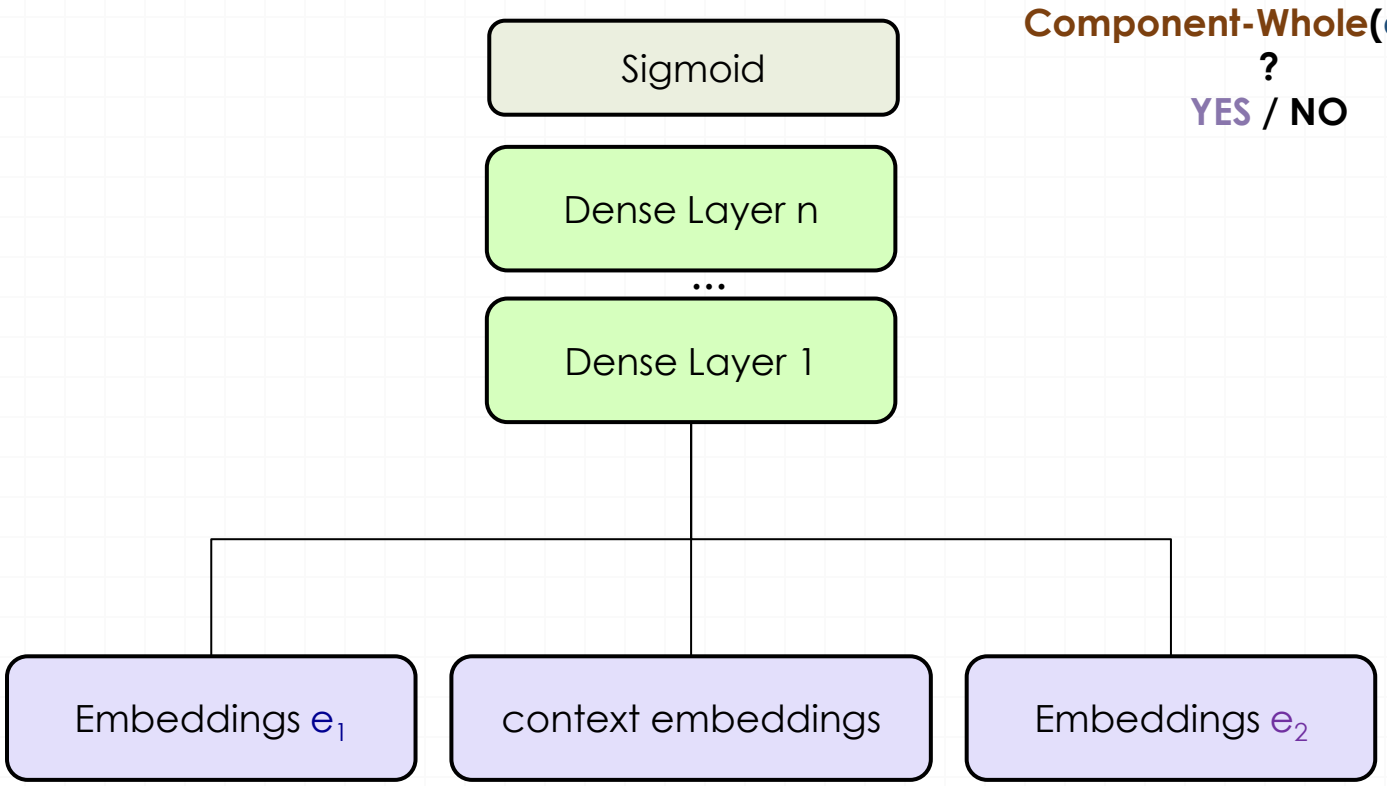


2) word sequences concatenated with positional features



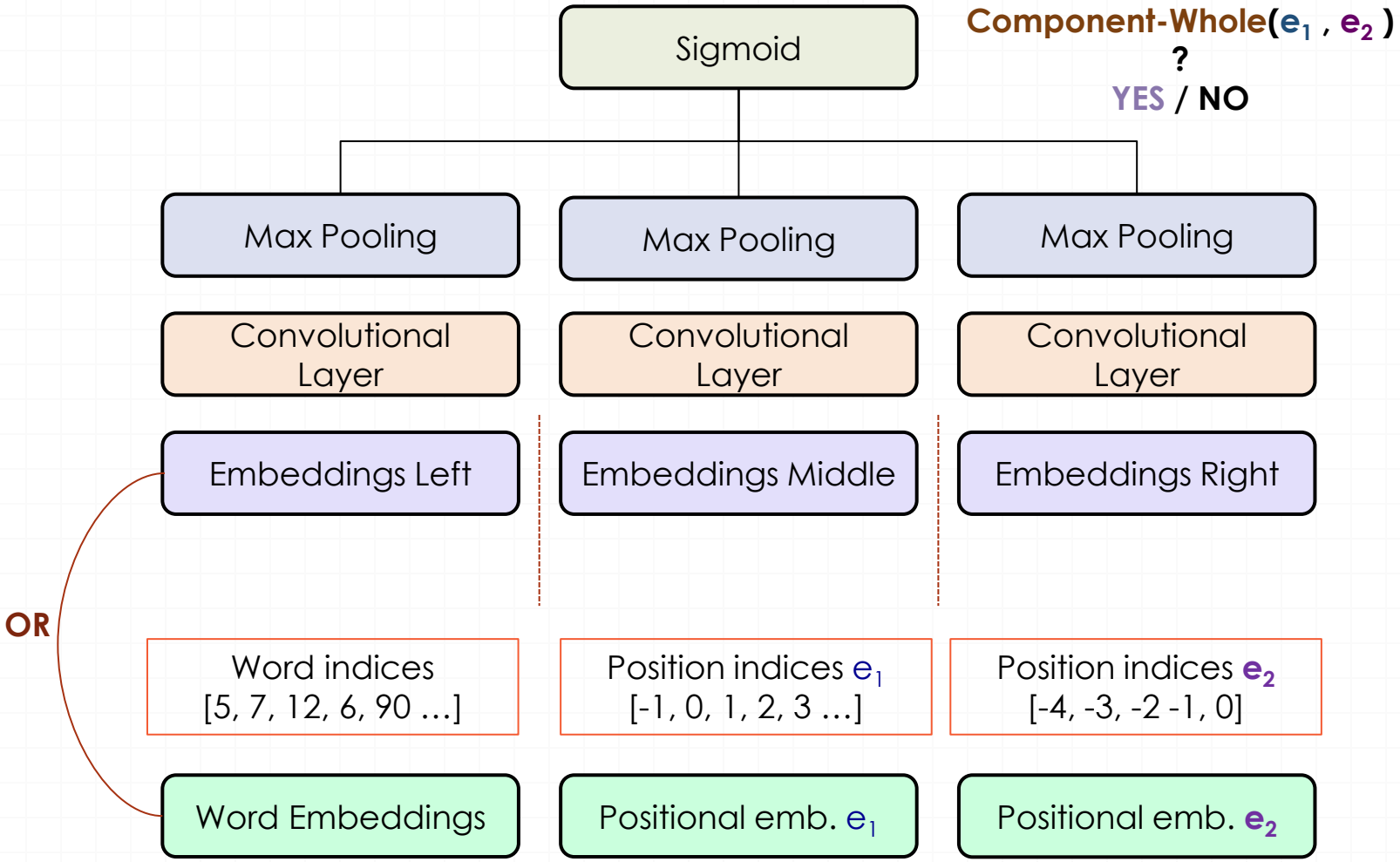
3) concatenating embeddings of two entities with average of word embeddings for rest of the words

Models: MLP



Simple fully-connected multi-layer perceptron

Models: CNN

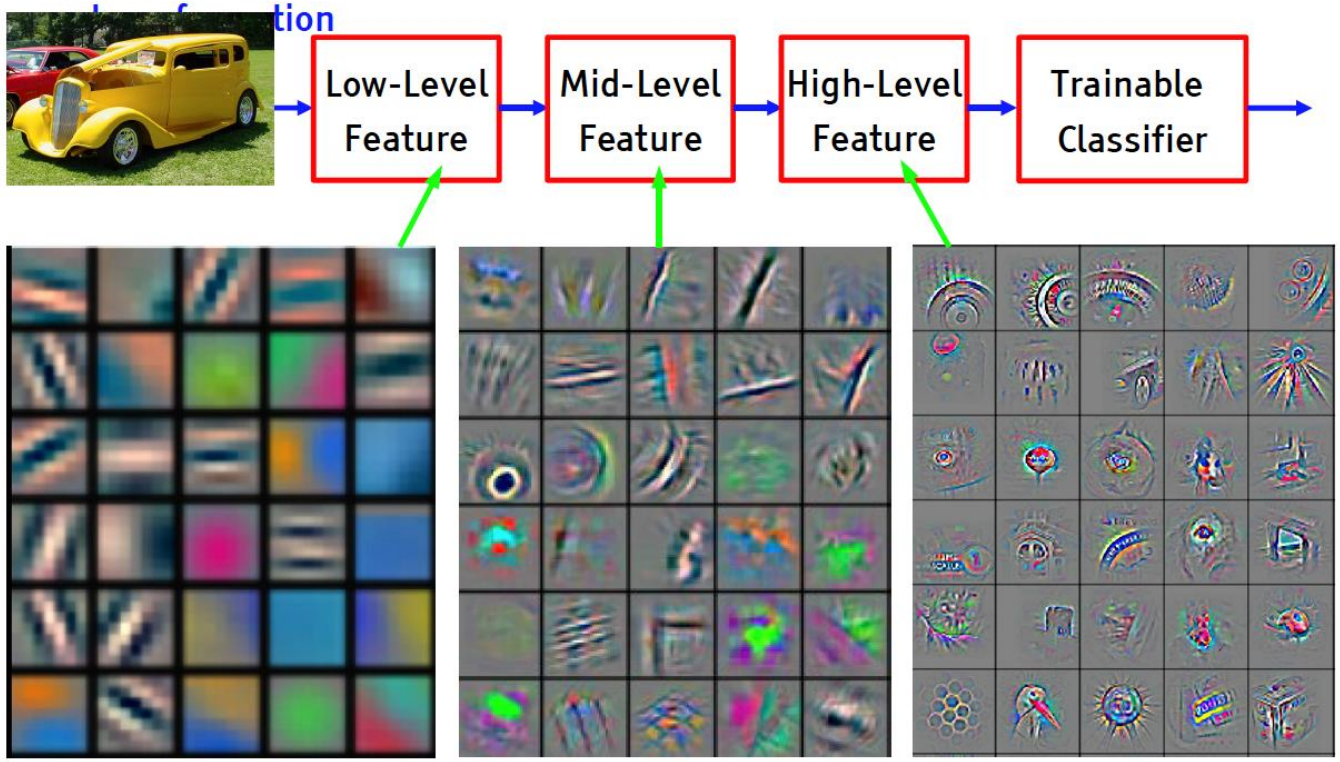


Zeng, D.et al. "Relation classication via convolutional deep neural network".COLING (2014)

Deep Learning Today

- Advancement in speech recognition in the last 2 years
  - A few long-standing performance records were broken with deep learning methods
  - Microsoft and Google have both deployed DL-based speech recognition systems in their products
- Advancement in Computer Vision
  - Feature engineering is the bread-and-butter of a large portion of the CV community, which creates some resistance to feature learning
  - But the record holders on ImageNet and Semantic Segmentation are convolutional nets
- Advancement in Natural Language Processing
  - Fine-grained sentiment analysis, syntactic parsing
  - Language model, machine translation, question answering

Deep Learning = Learning Hierarchical Representations



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

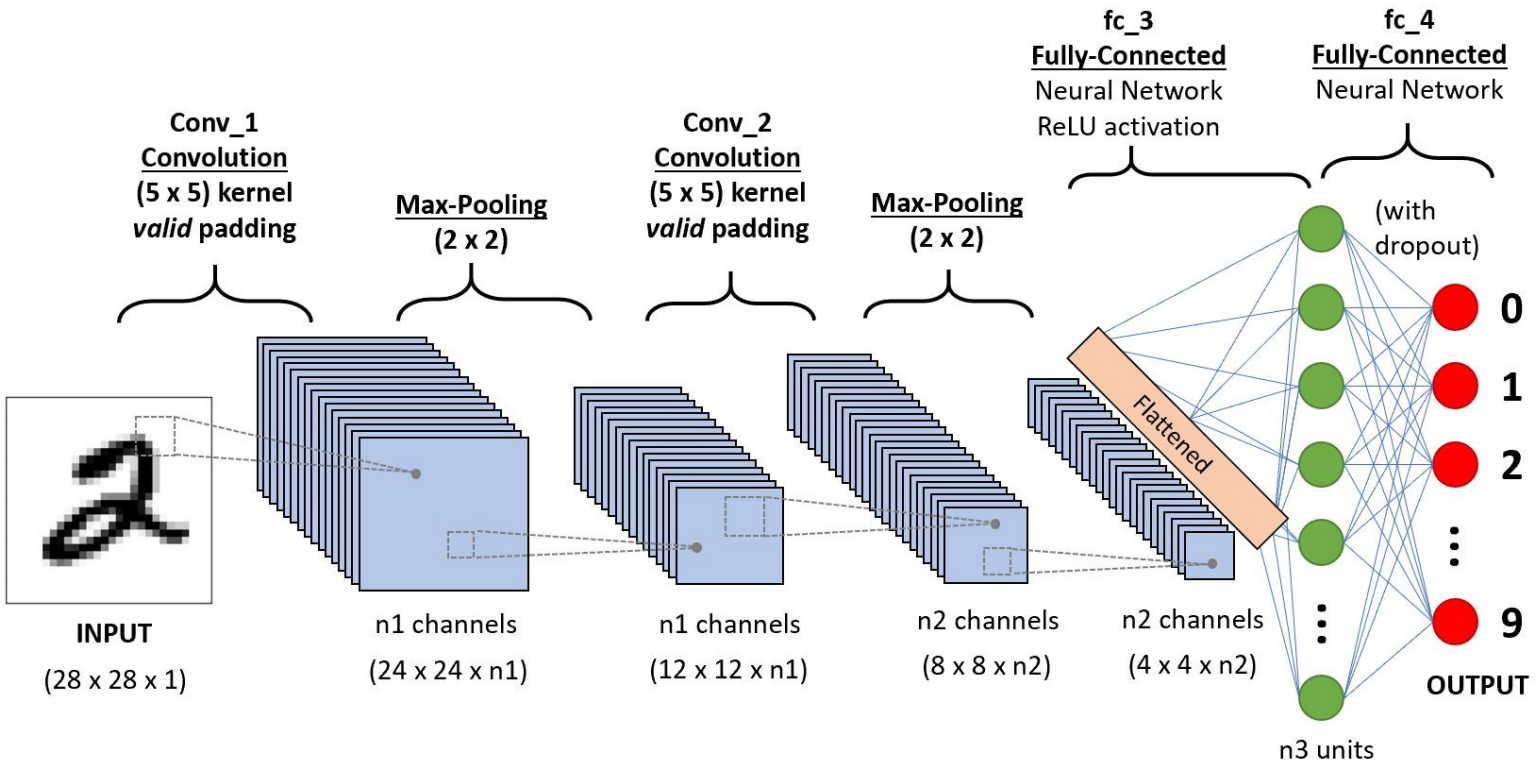
Convolutional Neural Network (CNN)

- Convolutional Neural Networks are inspired by mammalian visual cortex.
  - The visual cortex contains a complex arrangement of cells, which are sensitive to small sub-regions of the visual field, called a receptive field. These cells act as local filters over the input space and are well-suited to exploit the strong spatially local correlation present in natural images.
  - Two basic cell types:
    - Simple cells respond maximally to specific edge-like patterns within their receptive field.
    - Complex cells have larger receptive fields and are locally invariant to the exact position of the pattern.

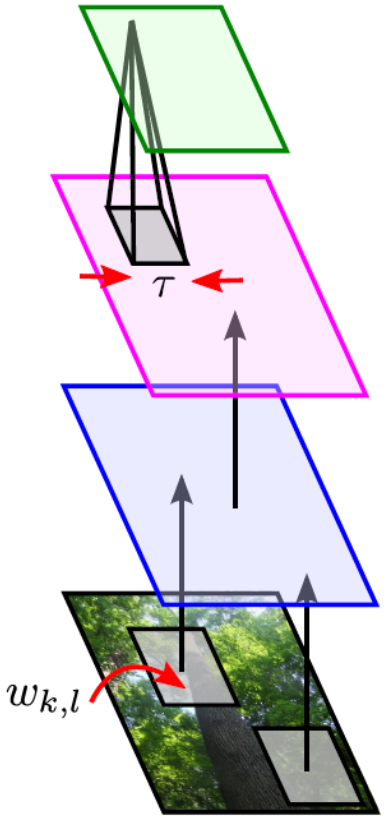
CNN Architecture

- Intuition: Neural network with specialized connectivity structure,
  - Stacking multiple layers of feature extractors
  - Low-level layers extract local features.
  - High-level layers extract learn global patterns.
- A CNN is a list of layers that transform the input data into an output class/prediction.
- There are a few distinct types of layers:
  - Convolutional layer
  - Non-linear layer
  - Pooling layer

CNN Example


Conv_1
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
Conv_2
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
fc_3
Fully-Connected
Neural Network
ReLU activation
fc_4
Fully-Connected
Neural Network
(with dropout)
Flattened
INPUT
(28 x 28 x 1)
n1 channels
(24 x 24 x n1)
n1 channels
(12 x 12 x n1)
n2 channels
(8 x 8 x n2)
n2 channels
(4 x 4 x n2)
n3 units
0
1
2
9
OUTPUT

Building-blocks for CNN's



$$x_{i,j} = \max_{|k|<\tau,|l|<\tau} y_{i-k,j-l}$$

mean or subsample also used

pooling stage

Feature maps of a larger region are combined.

$$y_{i,j} = f(a_{i,j})$$

e.g. $f(a) = [a]_+$

$f(a) = \text{sigmoid}(a)$

non-linear stage

Feature maps are trained with neurons.

$$a_{i,j} = \sum_{k,l} w_{k,l} z_{i-k,j-l}$$

Shared weights

convolutional stage

Each sub-region yields a feature map, representing its feature.

$z_{i,j}$

Images are segmented into sub-regions.

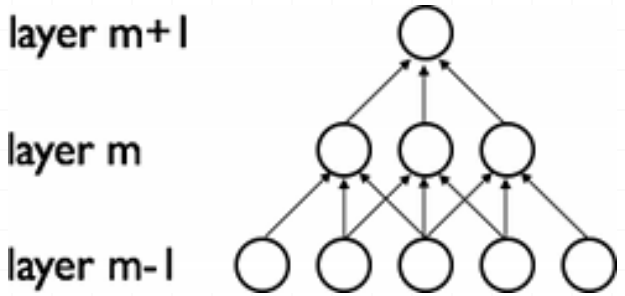
input image

CNN Architecture: Convolutional Layer

- The core layer of CNNs
- The convolutional layer consists of a set of filters.
  - Each filter covers a spatially small portion of the input data.
- Each filter is convolved across the dimensions of the input data, producing a multidimensional feature map.
  - As we convolve the filter, we are computing the dot product between the parameters of the filter and the input.
- Intuition: the network will learn filters that activate when they see some specific type of feature at some spatial position in the input.
- The key architectural characteristics of the convolutional layer is local connectivity and shared weights.

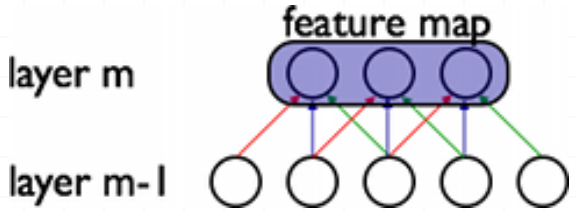
CNN Convolutional Layer: Local Connectivity

- Neurons in layer **m** are only connected to 3 adjacent neurons in the **m-1** layer.
- Neurons in layer **m+1** have a similar connectivity with the layer below.
- Each neuron is unresponsive to variations outside of its *receptive field* with respect to the input.
  - Receptive field: small neuron collections which process portions of the input data
- The architecture thus ensures that the learnt feature extractors produce the strongest response to a spatially local input pattern.



CNN Convolutional Layer Shared Weights

- We show 3 hidden neurons belonging to the same feature map (the layer right above the input layer).
- Weights of the same color are shared—constrained to be identical.
- Gradient descent can still be used to learn such shared parameters, with only a small change to the original algorithm.
- The gradient of a shared weight is simply the sum of the gradients of the parameters being shared.
- Replicating neurons in this way allows for features to be detected regardless of their position in the input.
- Additionally, weight sharing increases learning efficiency by greatly reducing the number of free parameters being learnt.



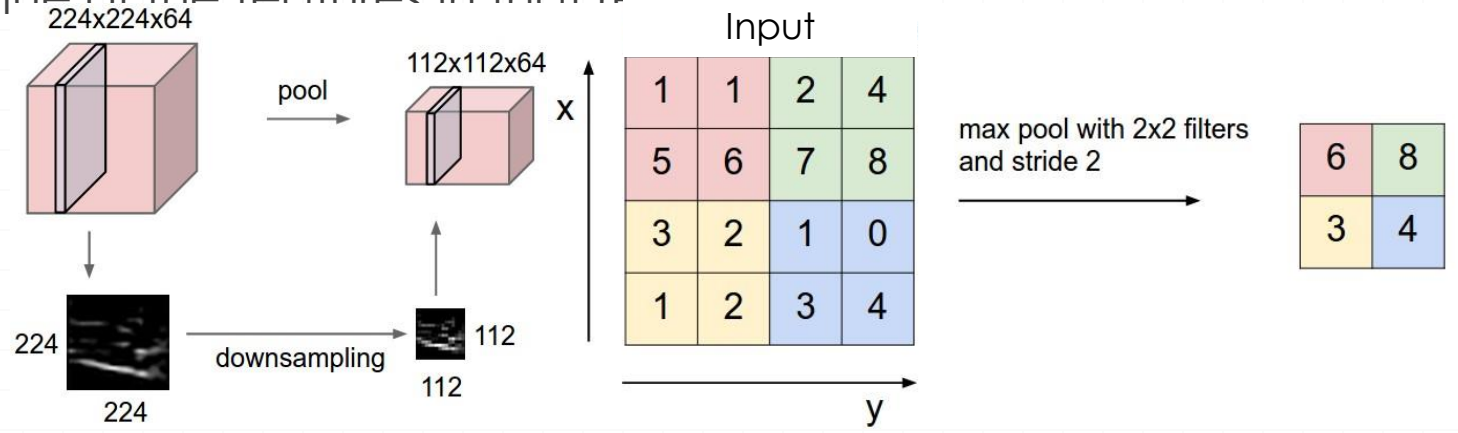
CNN Architecture: Non-linear Layer

- Intuition: Increase the nonlinearity of the entire architecture without affecting the receptive fields of the convolution layer

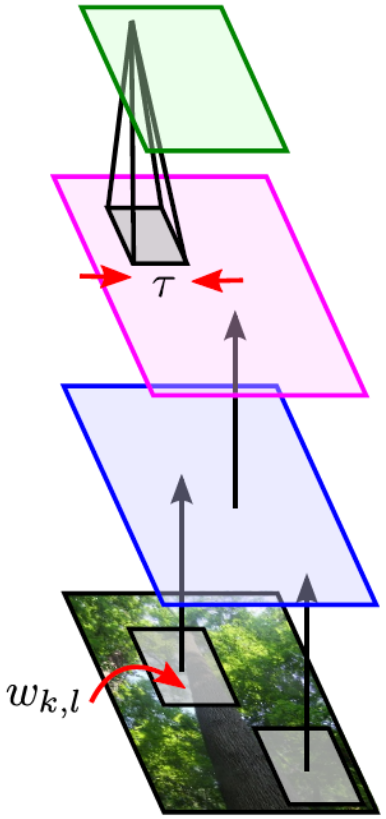
- A layer of neurons that applies the non-linear activation function, such as,
  - $f(x) = \max(0, x)$
  - $f(x) = \tanh x$
  - $f(x) = |\tanh x|$
  - $f(x) = (1 + e^{-x})^{-1}$

CNN Architecture: Pooling Layer

- Intuition: to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting
- Pooling partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum value of the features in that region



Building-blocks for CNN's



$$x_{i,j} = \max_{|k|<\tau,|l|<\tau} y_{i-k,j-l}$$

mean or subsample also used

pooling stage

Feature maps of a larger region are combined.

$$y_{i,j} = f(a_{i,j})$$

e.g. $f(a) = [a]_+$

$f(a) = \text{sigmoid}(a)$

non-linear stage

Feature maps are trained with neurons.

$$a_{i,j} = \sum_{k,l} w_{k,l} z_{i-k,j-l}$$

Shared weights

convolutional stage

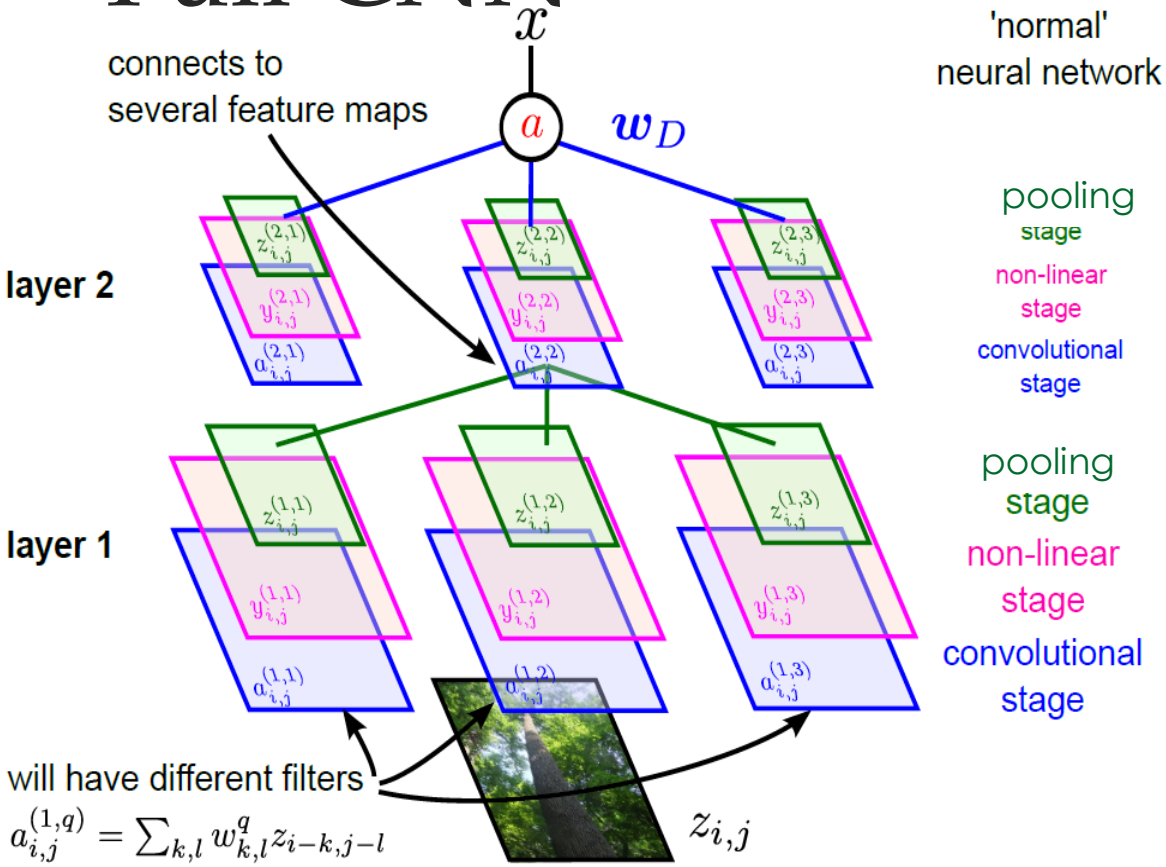
Each sub-region yields a feature map, representing its feature.

$z_{i,j}$

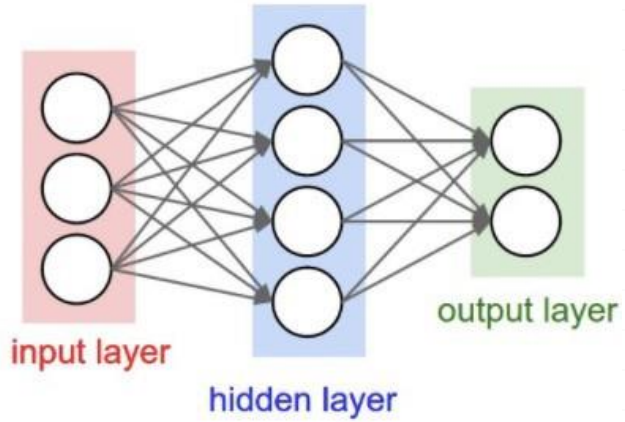
Images are segmented into sub-regions.

input image

Full CNN


x
connects to
several feature maps
a
w_D
'normal'
neural network
layer 2
layer 1
pooling
stage
non-linear
stage
convolutional
stage
pooling
stage
non-linear
stage
convolutional
stage
will have different filters
$a_{i,j}^{(1,q)} = \sum_{k,l} w_{k,l}^{q} z_{i-k,j-l}$
$z_{i,j}$

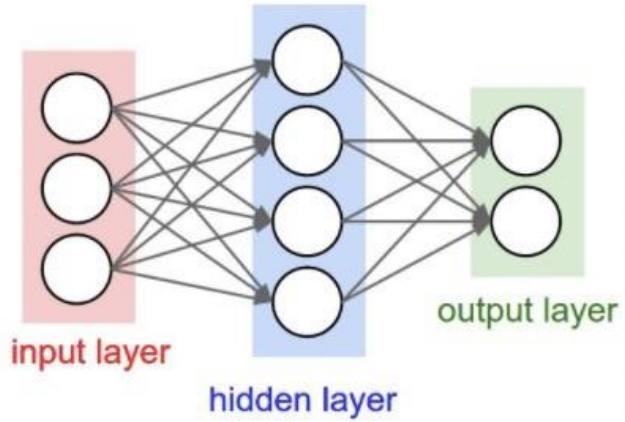
Motivation: Fully-Connected Layers Are Limited



Each node provides input to each node in the next layer

- Assume 3-layer model with 100 nodes, 100 nodes, and then 2 nodes
  - e.g., how many weights are in a 640x480 grayscale image?
    - 640x480x100 + 100x100 + 100x2 = 30,730,200
  - e.g., how many weights are in a 3.1 Megapixel grayscale image (2048X1536)?
    - 2048x1536x100 + 100x100 + 100x2 = 314,583,000

Motivation: Fully-Connected Layers Are Limited



Issue: many model parameters in fully connected networks

- Assume 3-layer model with 100 nodes, 100 nodes, and then 2 nodes
  - e.g., how many weights are in a 640x480 grayscale image?
    - 640x480x100 + 100x100 + 100x2 = 30,730,200
  - e.g., how many weights are in a 3.1 Megapixel grayscale image (2048X1536)?
    - 2048x1536x100 + 100x100 + 100x2 = 314,583,000

Motivation: Fully-Connected Layers Are Limited

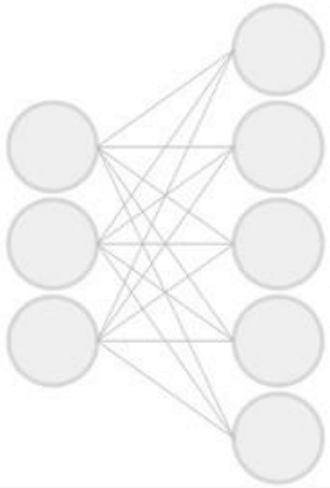
Many model parameters…

- increases chance of overfitting
- requires more training data
- increases memory/storage requirements

- Assume 3-layer model with 100 nodes, 100 nodes, and then 2 nodes
  - e.g., how many weights are in a 640x480 grayscale image?
    - 640x480x100 + 100x100 + 100x2 = 30,730,200
  - e.g., how many weights are in a 3.1 Megapixel grayscale image (2048X1536)?
    - 2048x1536x100 + 100x100 + 100x2 = 314,583,000

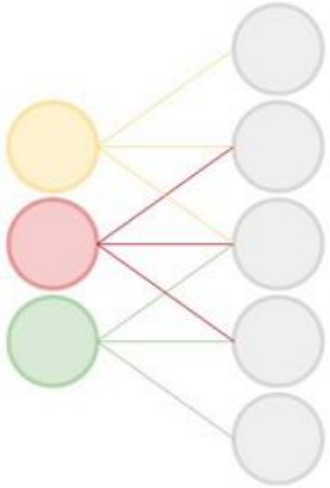
Key Ingredient 1: Convolutional Layers

Fully-connected:



Rather than have each node provide input to each node in the next layer…

Convolutional:

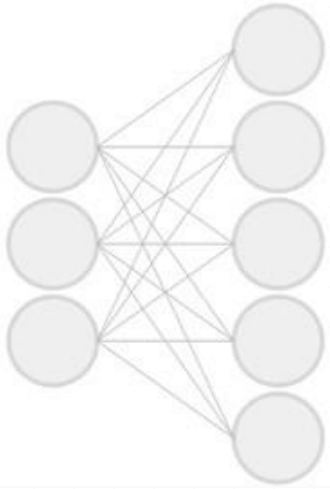


each node receives input only from a small neighborhood in previous layer (and there is parameter sharing)

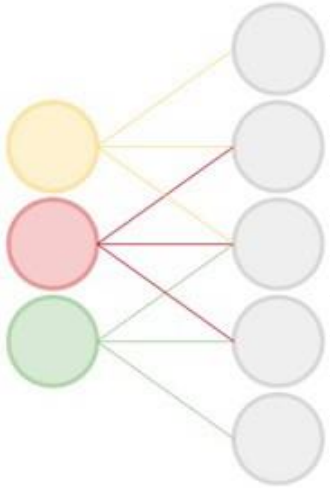
Figure Source: https://qph.fs.quoracdn.net/main-qimg-2e1f0071ca9878f7719ed0ea8aeb386d

Fully-Connected vs Convolutional Layers

Fully-connected:



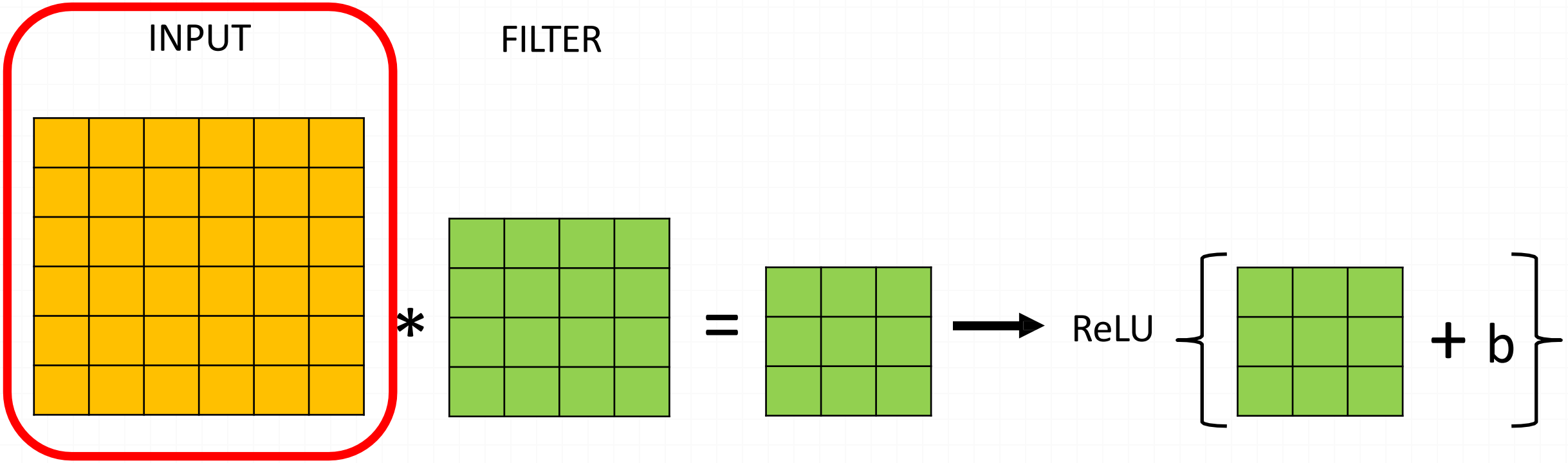
Convolutional:



Convolutional layers dramatically reduce number of model parameters!

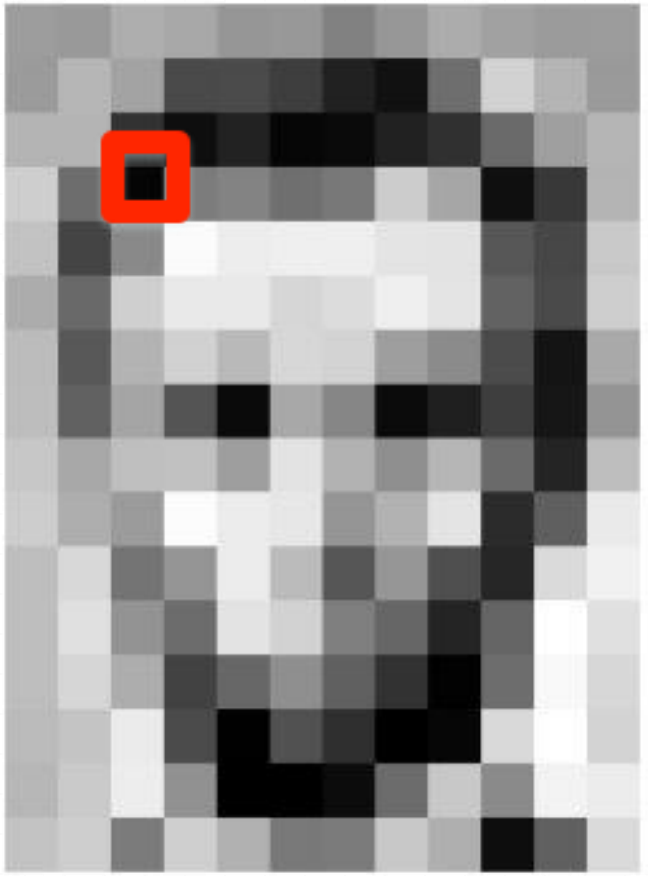
Figure Source: https://qph.fs.quoracdn.net/main-qimg-2e1f0071ca9878f7719ed0ea8aeb386d

Key Ingredient 1: Convolutional Layers



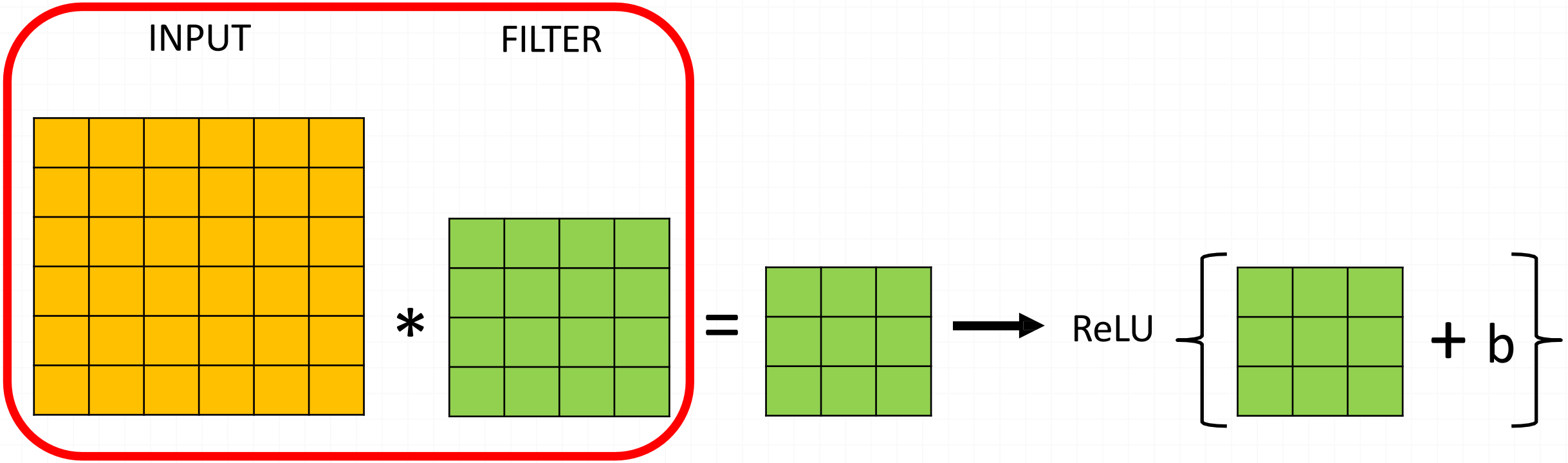
Recall: Image Representation (8-bit Grayscale)

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	24	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218



https://ai.stanford.edu/~syyeung/cvweb/tutorial1.html

Key Ingredient 1: Convolutional Layers



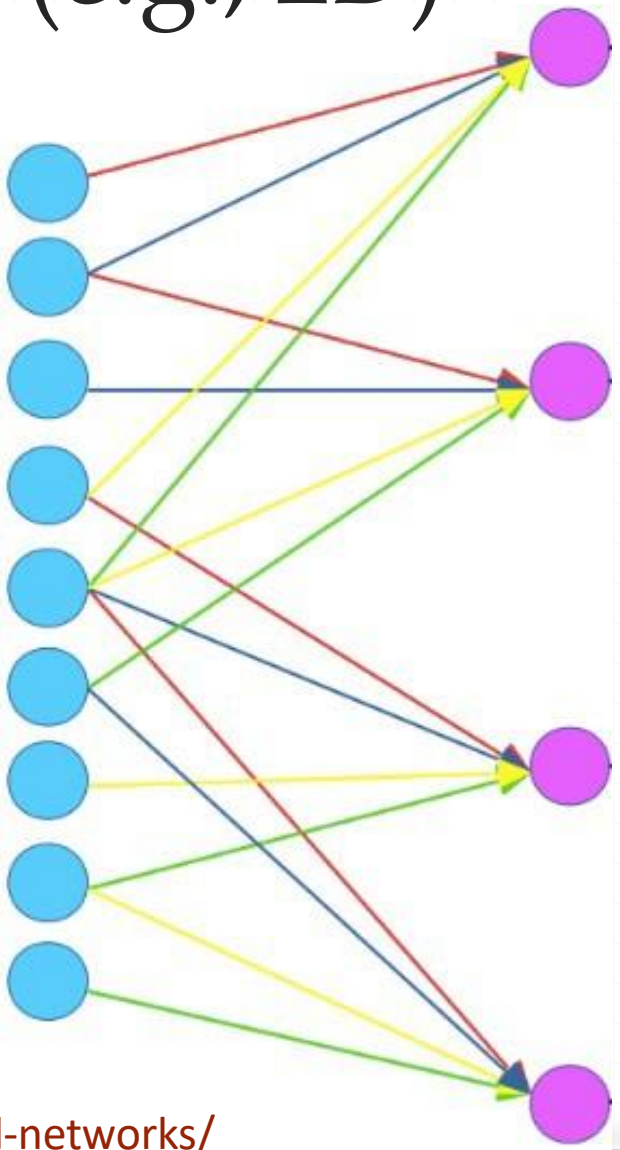
Convolution: Applies Linear Filter (e.g., 2D)

Input

Filter
(aka – Kernel)

Feature
Map

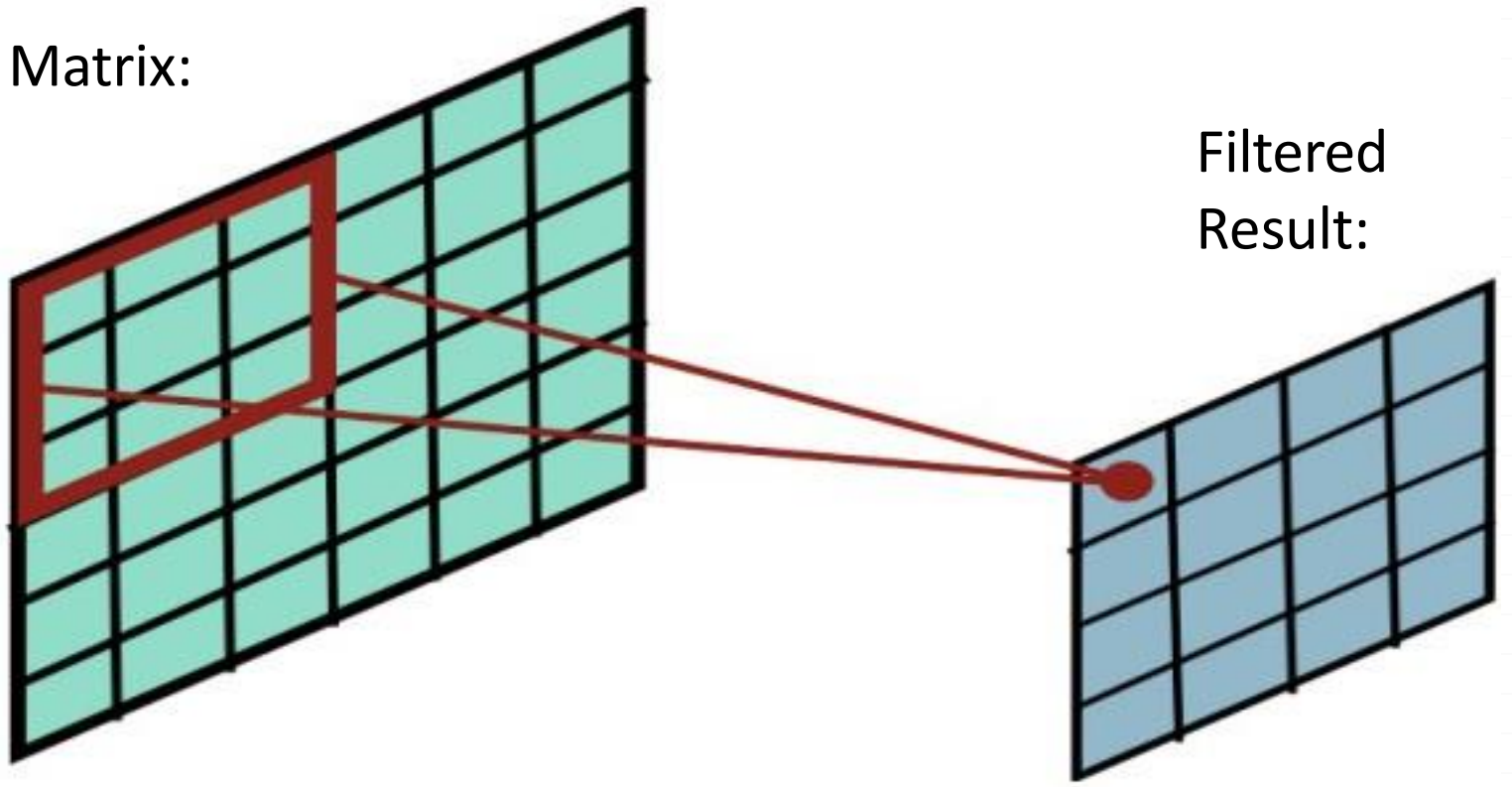
Way to Interpret
Neural Network



- Compute a function of local neighborhood for each location in matrix
- A filter specifies the function for how to combine neighbors' values

https://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/

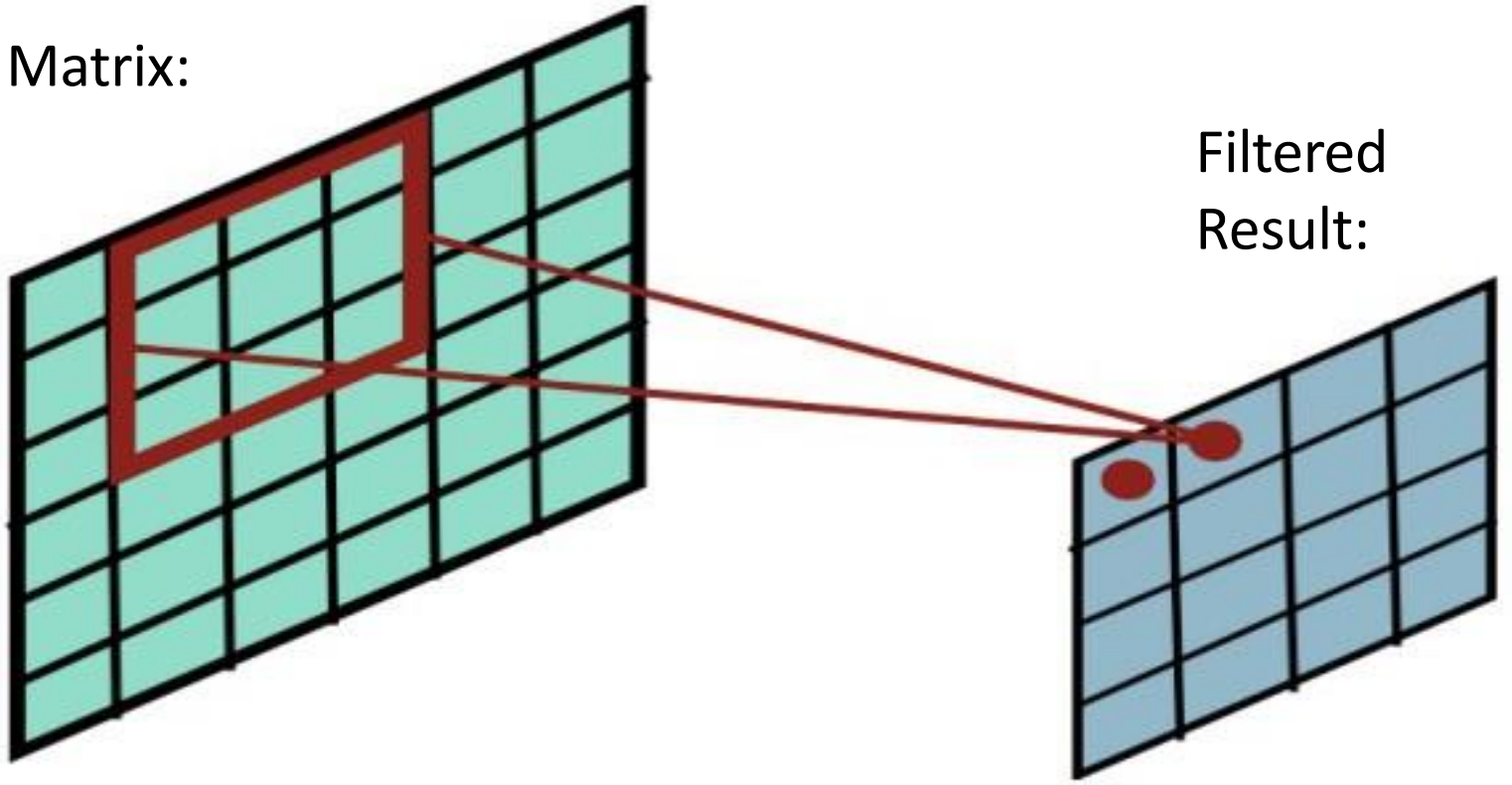
2D Filtering



Slides filter over the matrix and computes dot products

https://people.eecs.berkeley.edu/~jrs/189/lec/cnn.pdf

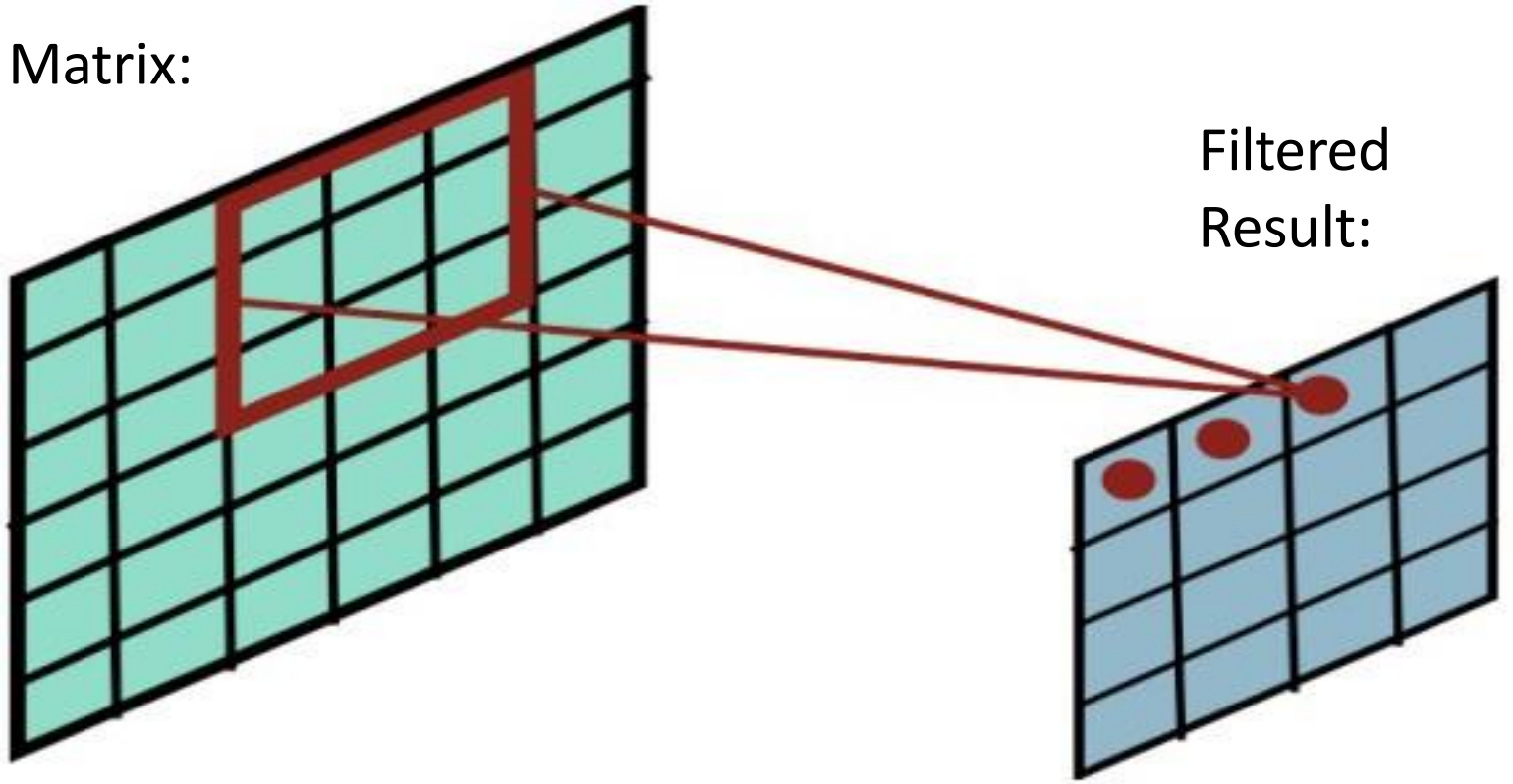
2D Filtering



Slides filter over the matrix and computes dot products

https://people.eecs.berkeley.edu/~jrs/189/lec/cnn.pdf

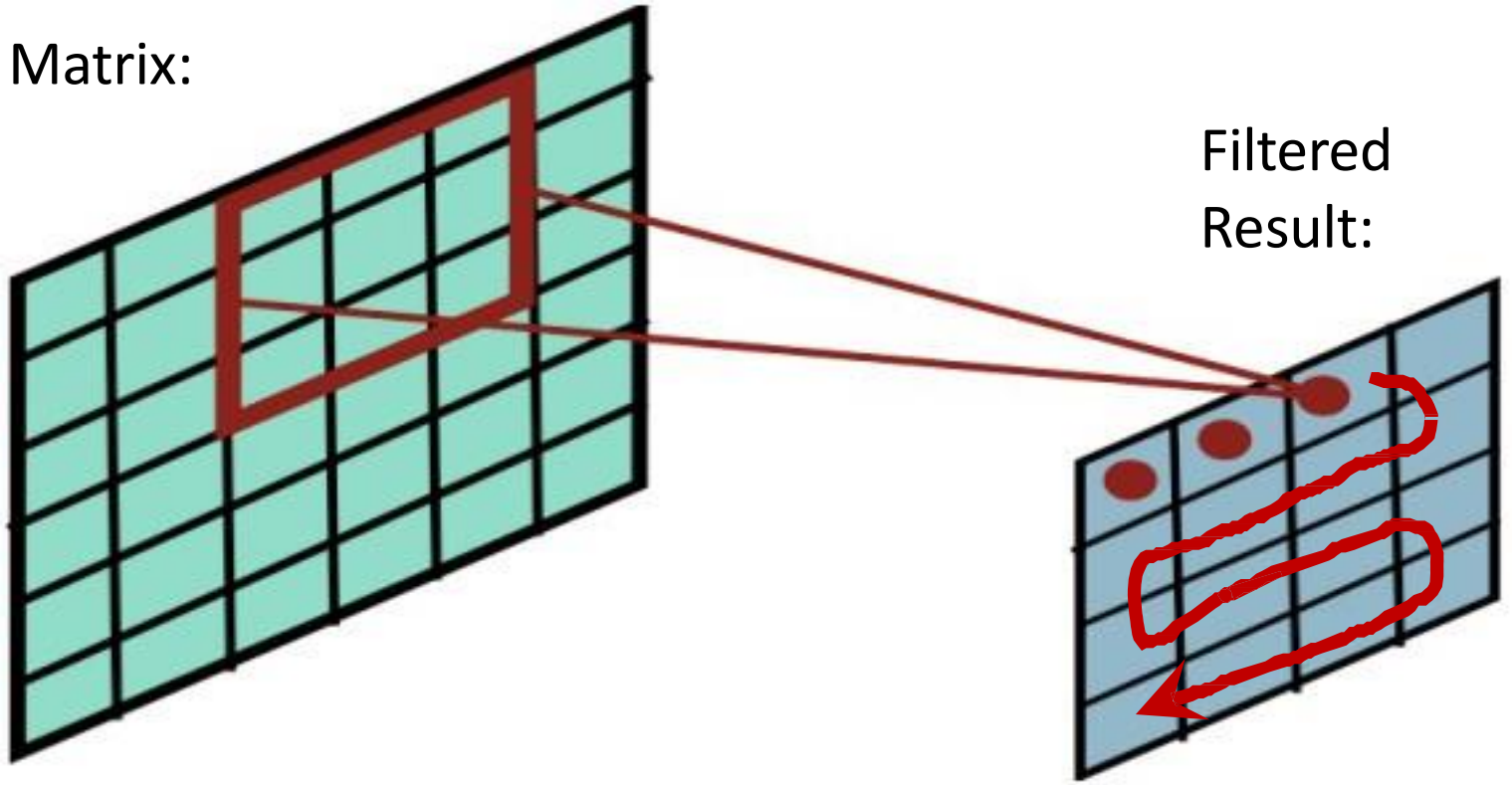
2D Filtering



Slides filter over the matrix and computes dot products

https://people.eecs.berkeley.edu/~jrs/189/lec/cnn.pdf

2D Filtering



Slides filter over the matrix and computes dot products

https://people.eecs.berkeley.edu/~jrs/189/lec/cnn.pdf

2D Filtering: Toy Example

Input

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

Filter

1	0	1
0	1	0
1	0	1

Feature Map

?	?	?
?	?	?
?	?	?

Dot Product = 1*1 + 1*0 + 1*1 + 0*0 + 1*1 + 1*0 + 0*1 + 0*1 + 0*0 + 0*0 + 1*1

Dot Product = 4

2D Filtering: Toy Example

Input

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

Filter

1	0	1
0	1	0
1	0	1

Feature Map

4	?	?
?	?	?
?	?	?

2D Filtering: Toy Example

Input

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

Filter

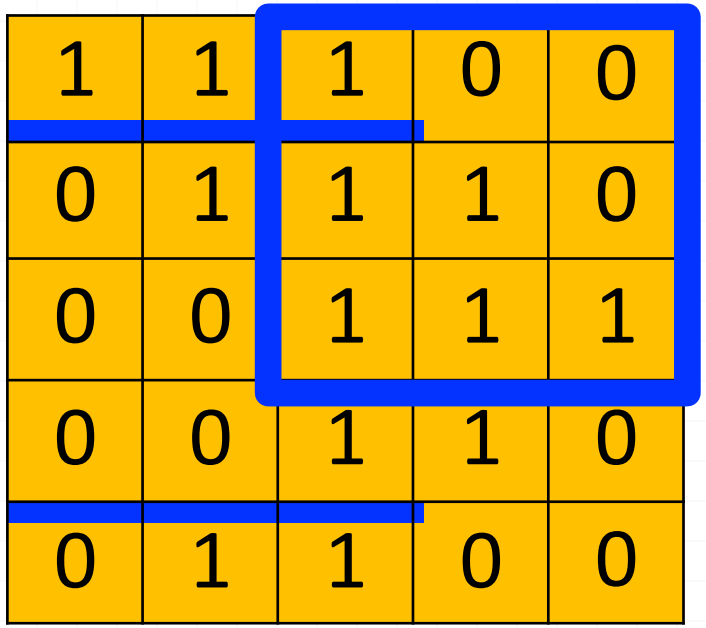
1	0	1
0	1	0
1	0	1

Feature Map

4	3	?
?	?	?
?	?	?

2D Filtering: Toy Example

Input



Filter

1	0	1
0	1	0
1	0	1

Feature Map

4	3	4
?	?	?
?	?	?

2D Filtering: Toy Example

Input

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

Filter

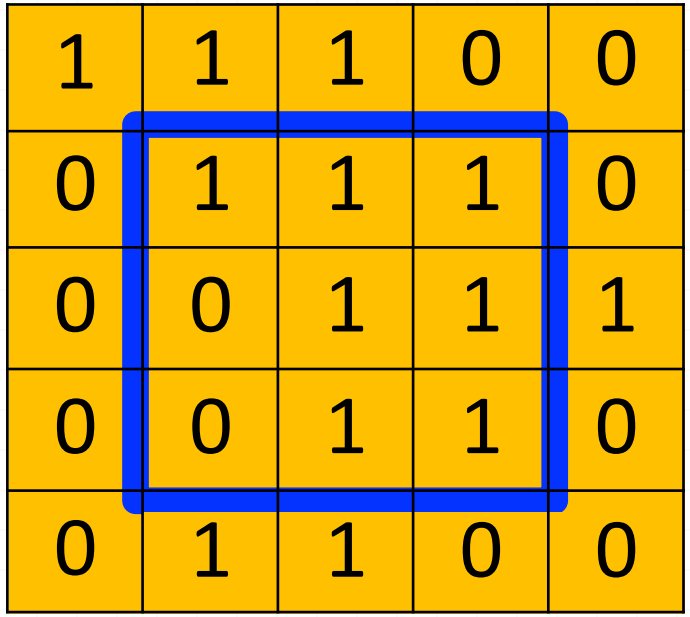
1	0	1
0	1	0
1	0	1

Feature Map

4	3	4
2	?	?
?	?	?

2D Filtering: Toy Example

Input



Filter

1	0	1
0	1	0
1	0	1

Feature Map

4	3	4
2	4	?
?	?	?

2D Filtering: Toy Example

Input

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

Filter

1	0	1
0	1	0
1	0	1

Feature Map

4	3	4
2	4	3
?	?	?

2D Filtering: Toy Example

Input

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

Filter

1	0	1
0	1	0
1	0	1

Feature Map

4	3	4
2	4	3
2	?	?

2D Filtering: Toy Example

Input

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

Filter

1	0	1
0	1	0
1	0	1

Feature Map

4	3	4
2	4	3
2	3	?

2D Filtering: Toy Example

Input

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

Filter

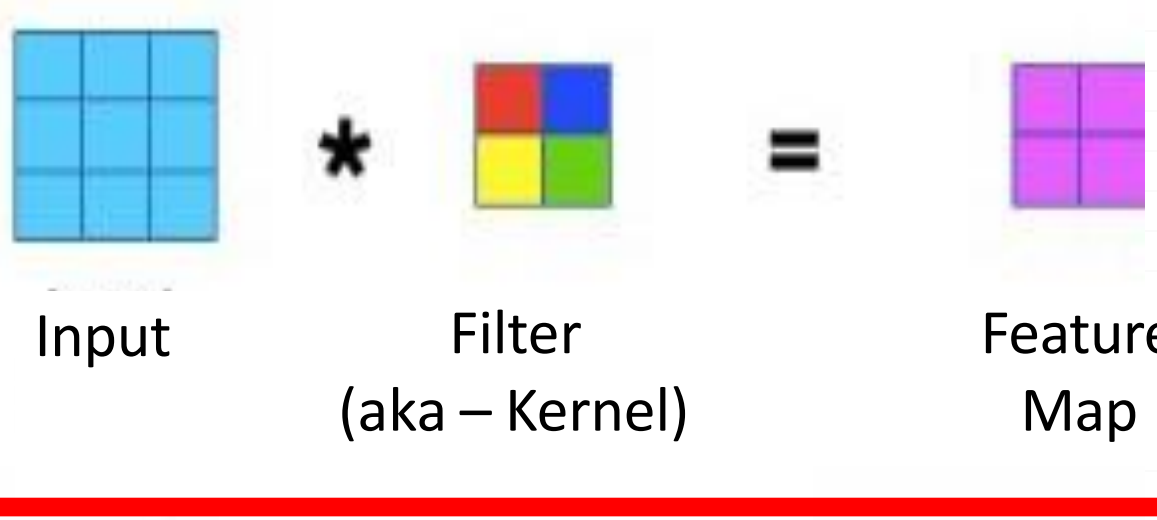
1	0	1
0	1	0
1	0	1

Feature Map

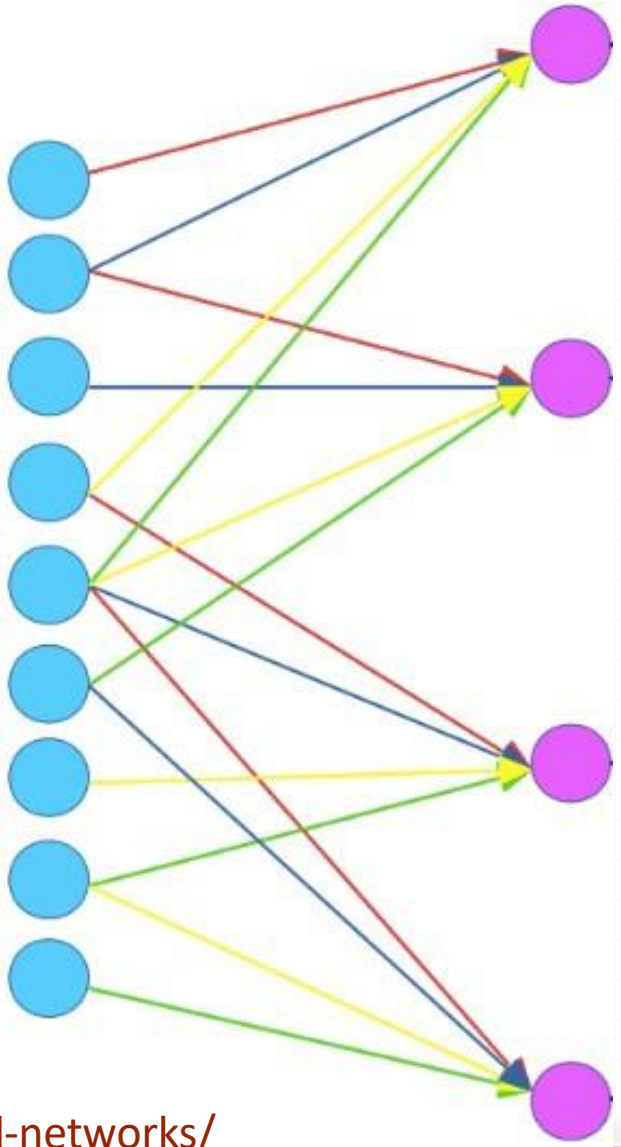
4	3	4
2	4	3
2	3	4

Convolutional Layer

- Many neural network libraries use “convolution” interchangeably with “cross correlation”; for mathematicians, these are technically different
- Examples in these slides show the “cross-correlation” function

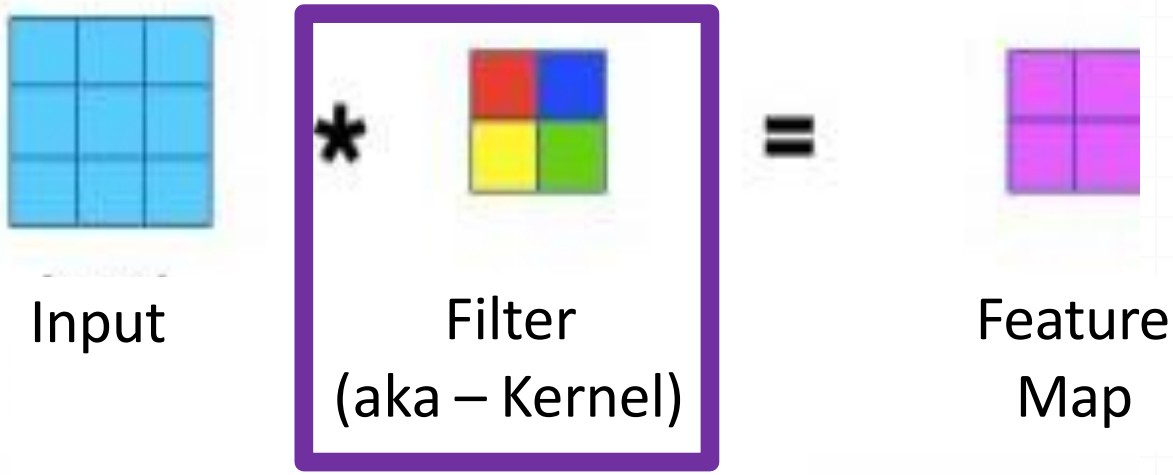


Way to Interpret
Neural Network

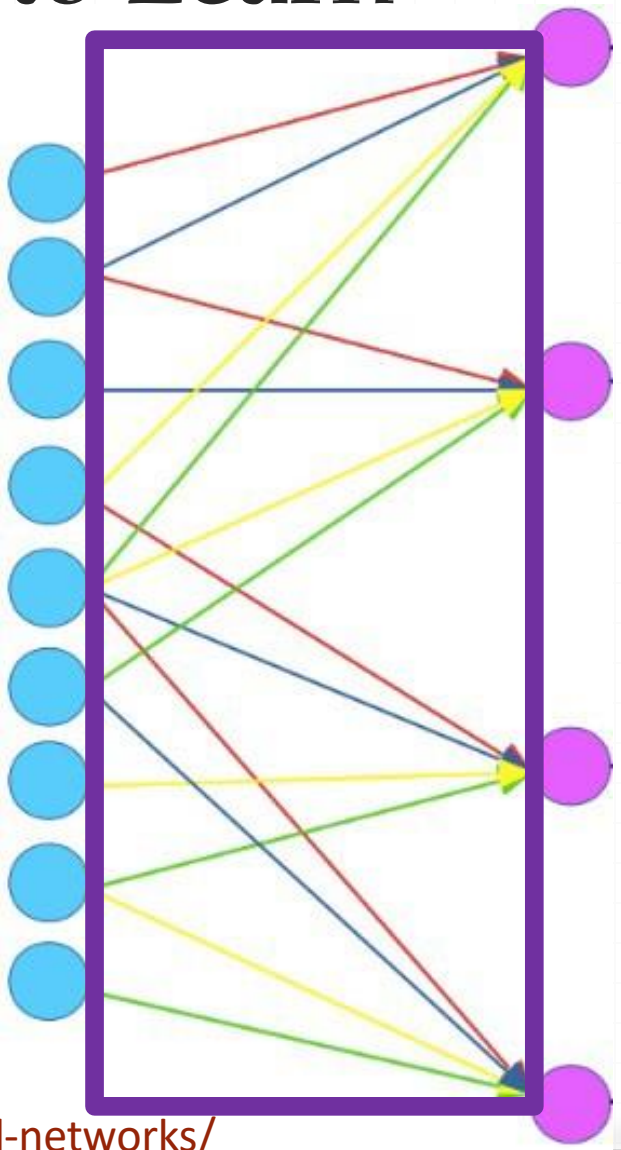


https://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/

Convolutional Layer: Parameters to Learn



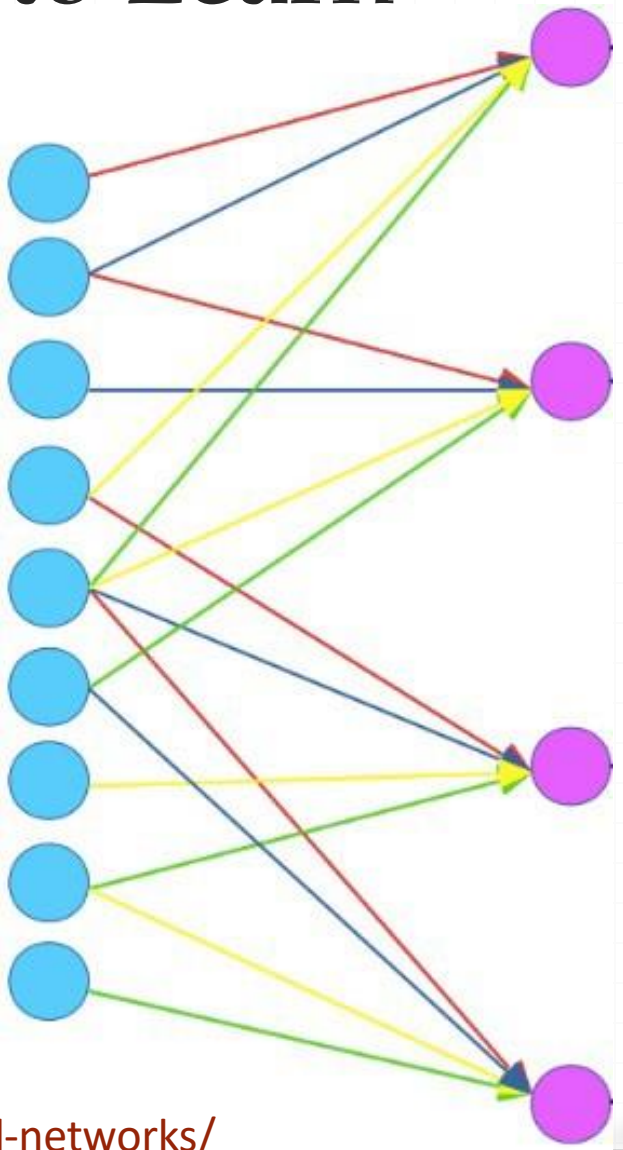
Way to Interpret
Neural Network



https://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/

Convolutional Layer: Parameters to Learn

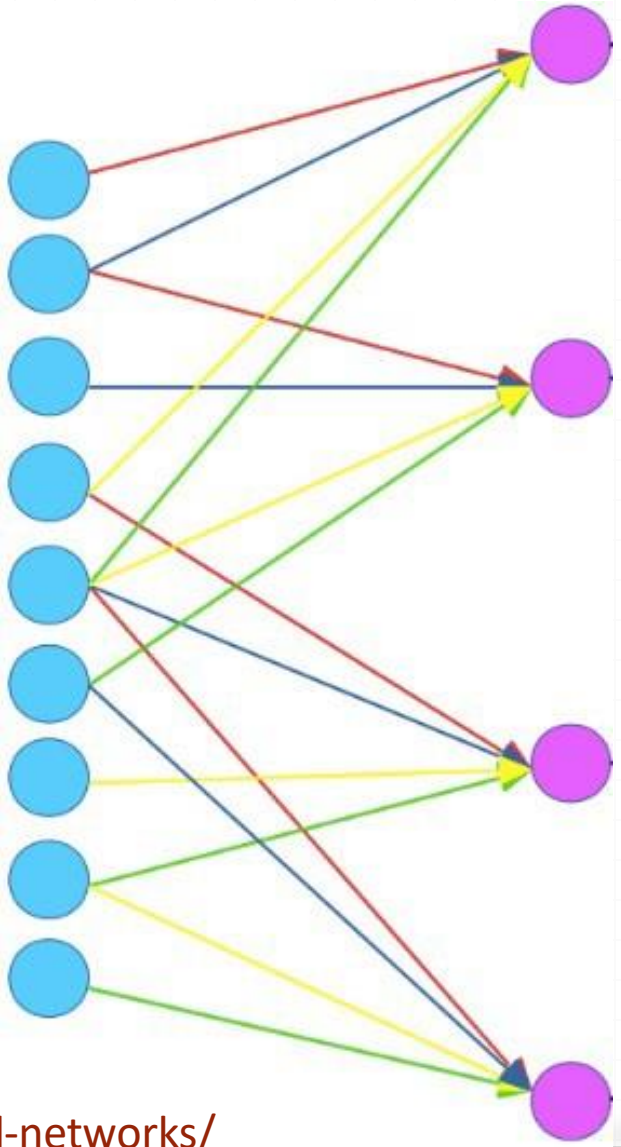
- For shown example, how many weights must be learned?
  - 4 (red, blue, yellow, and green values)
- If we instead used a fully connected layer, how many weights would need to be learned?
  - 36 (9 turquoise nodes x 4 magenta nodes)



https://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/

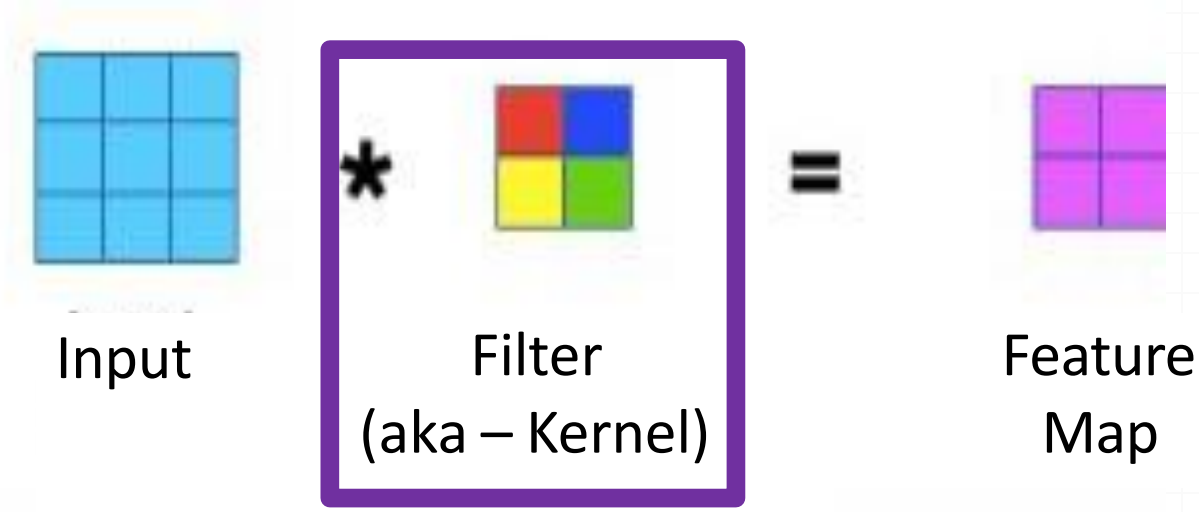
Convolutional Layer: Parameters to Learn

Neocognitron hard-coded filter values...
filter values are learned for CNNs

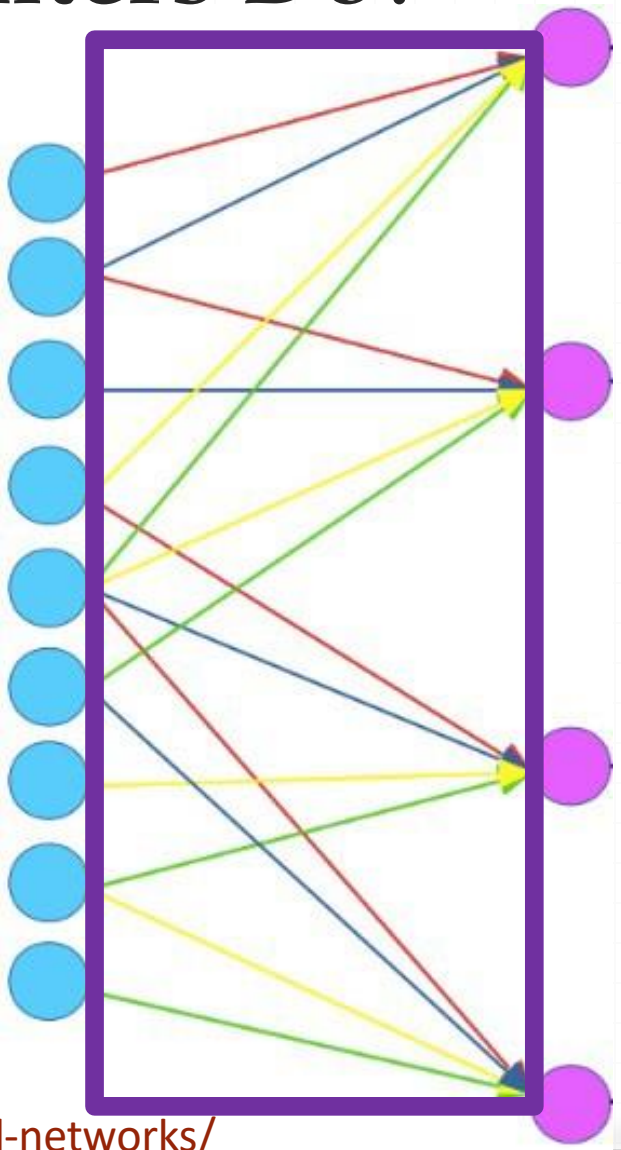


https://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/

Convolutional Layer: What Can Filters Do?



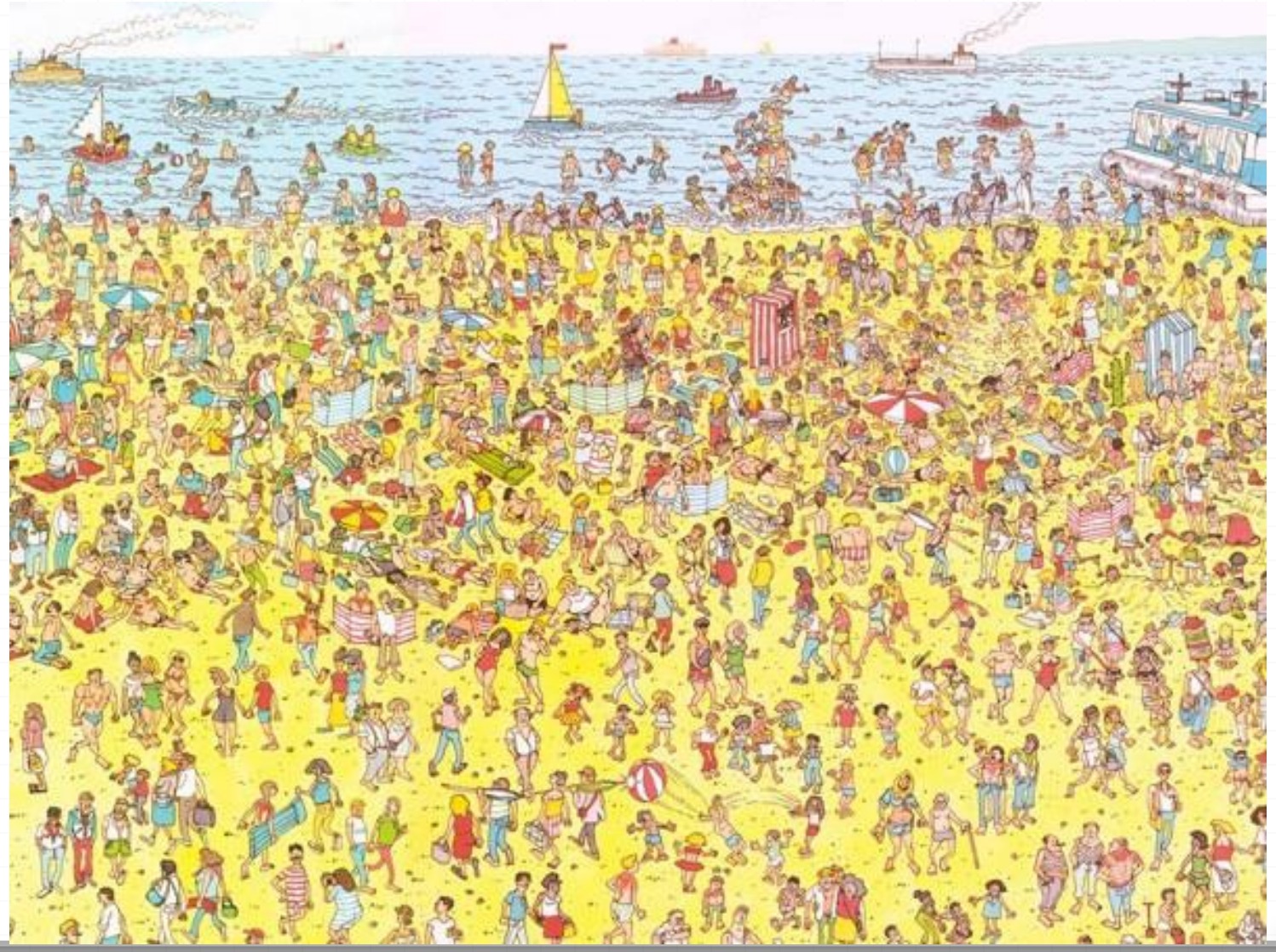
Way to Interpret
Neural Network



https://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/

Convolutional Layer: What Can Filters Do?

Filter



Convolutional Layer: What Can Filters Do?

- e.g.,

Filter

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Visualization of Filter

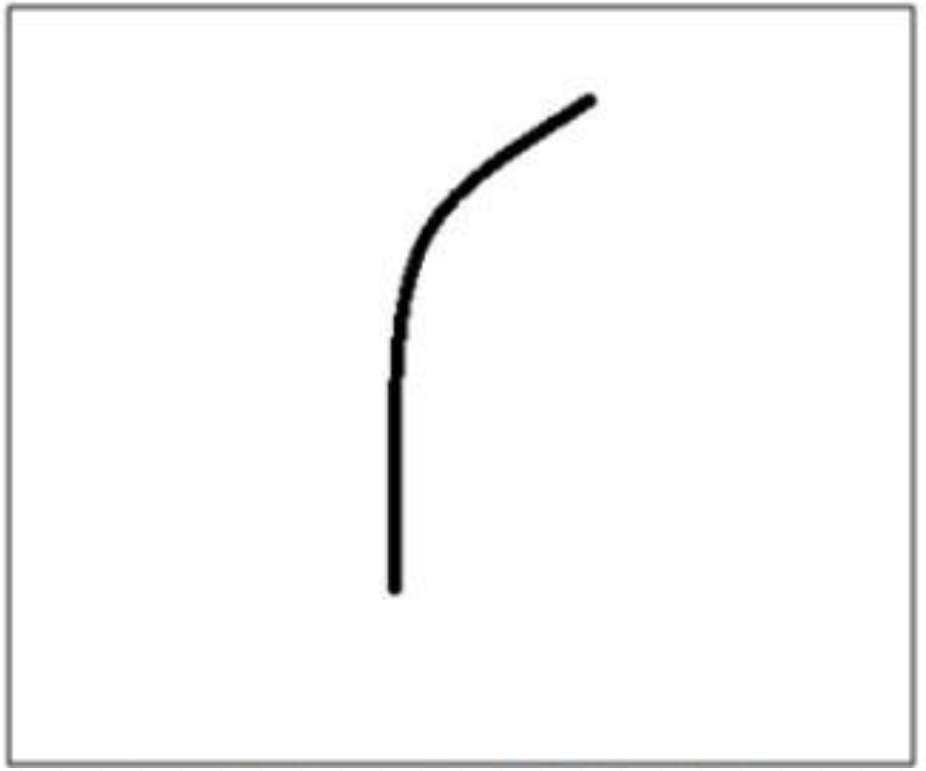
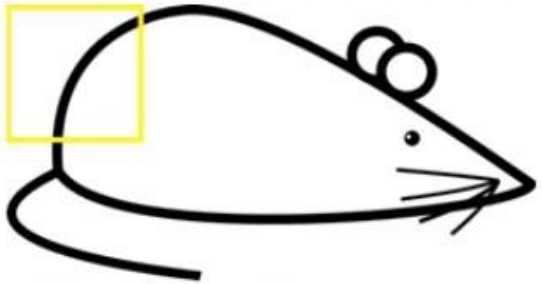


Image Credit: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

Convolutional Layer: What Can Filters Do?

- e.g.,

Filter Overlaid on Image



Image

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

*

Filter

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Weighted Sum = ?

Weighted Sum = (50x30) + (20x30) + (50x30) + (50x3) + (50x30)

Weighted Sum = 6600 **(Large Number!!)**

Image Credit: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

Convolutional Layer: What Can Filters Do?

- e.g.,

Filter Overlaid on Image

Image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

*

Filter

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Weighted Sum = ?

Weighted Sum = 0 **(Small Number!!)**

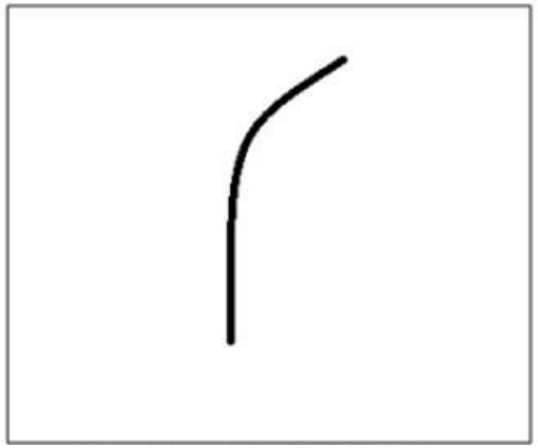
Image Credit: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

Convolutional Layer: What Can Filters Do?

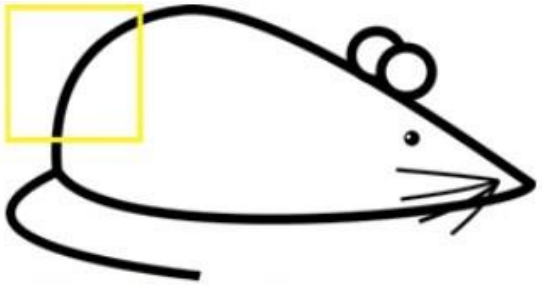
This Filter is a Curve Detector!

- e.g.,

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0



Filter Overlaid on Image (Big Response!)



Filter Overlaid on Image (Small Response!)

Image Credit: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

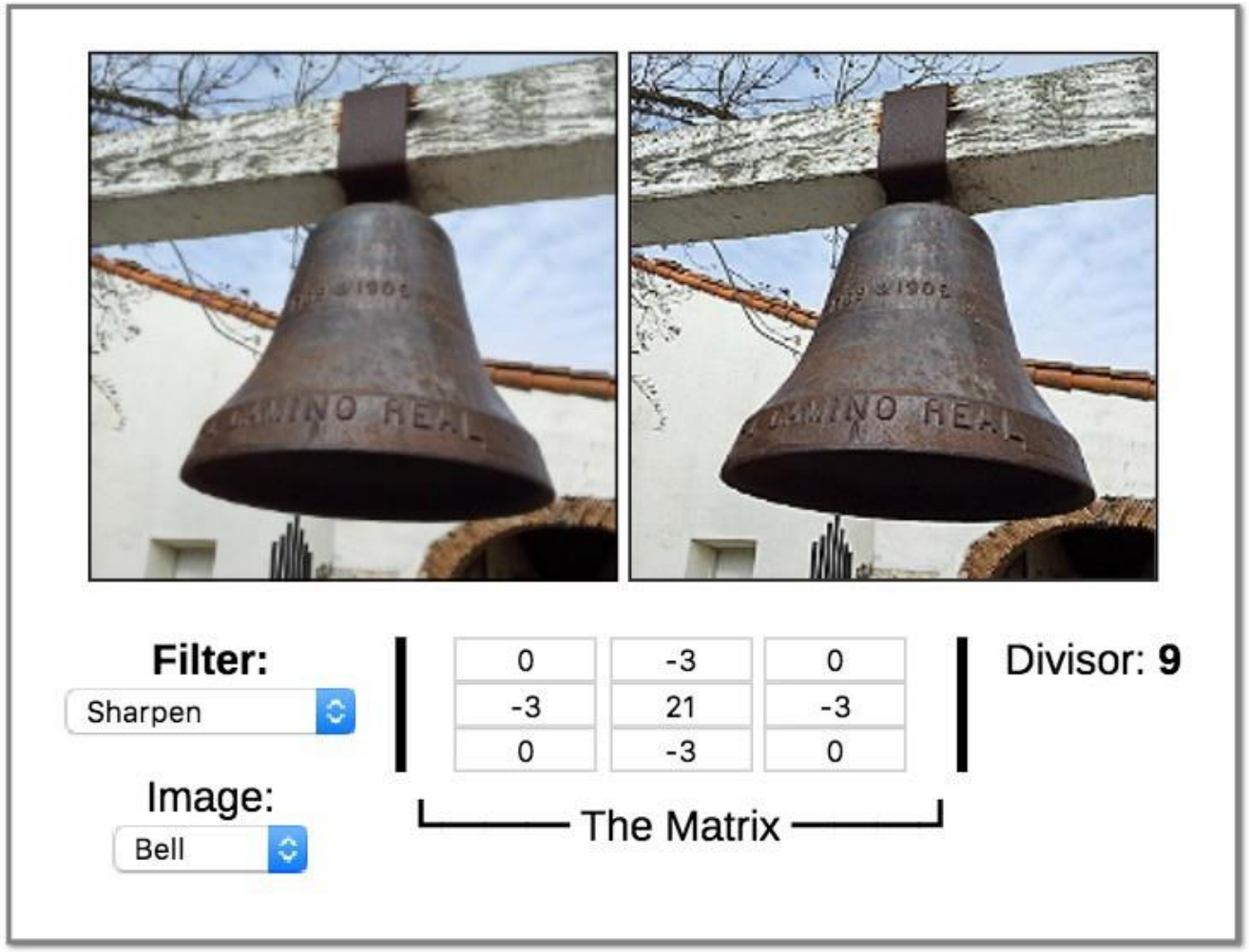
Convolutional Layer: What Can Filters Do?

Feature Map

Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

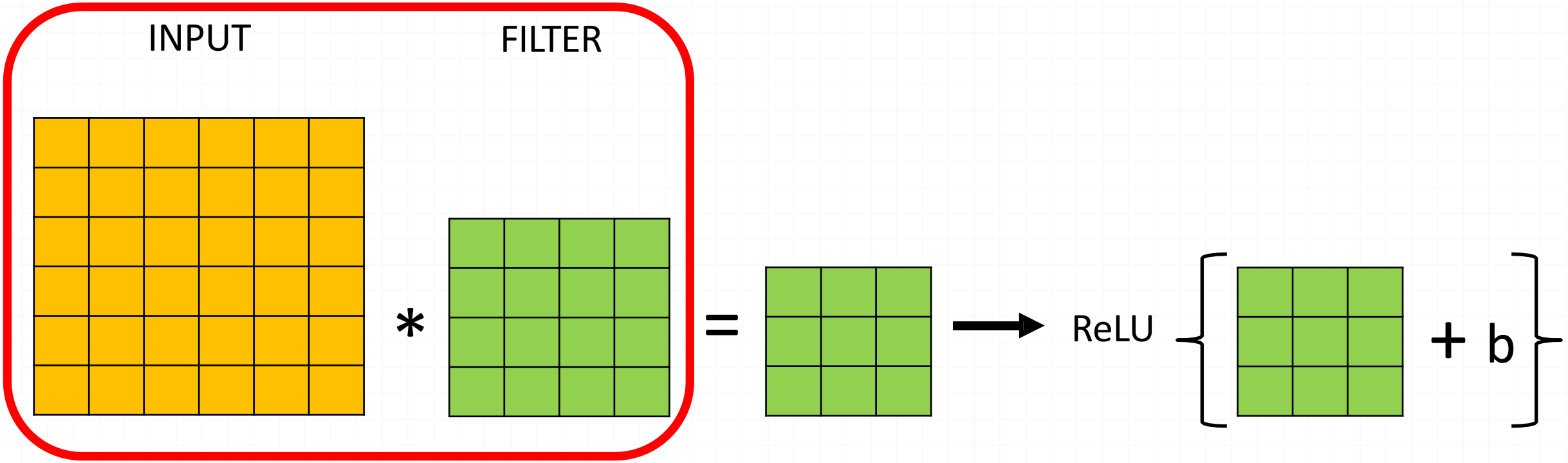
https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

Convolutional Layer: What Can Filters Do?



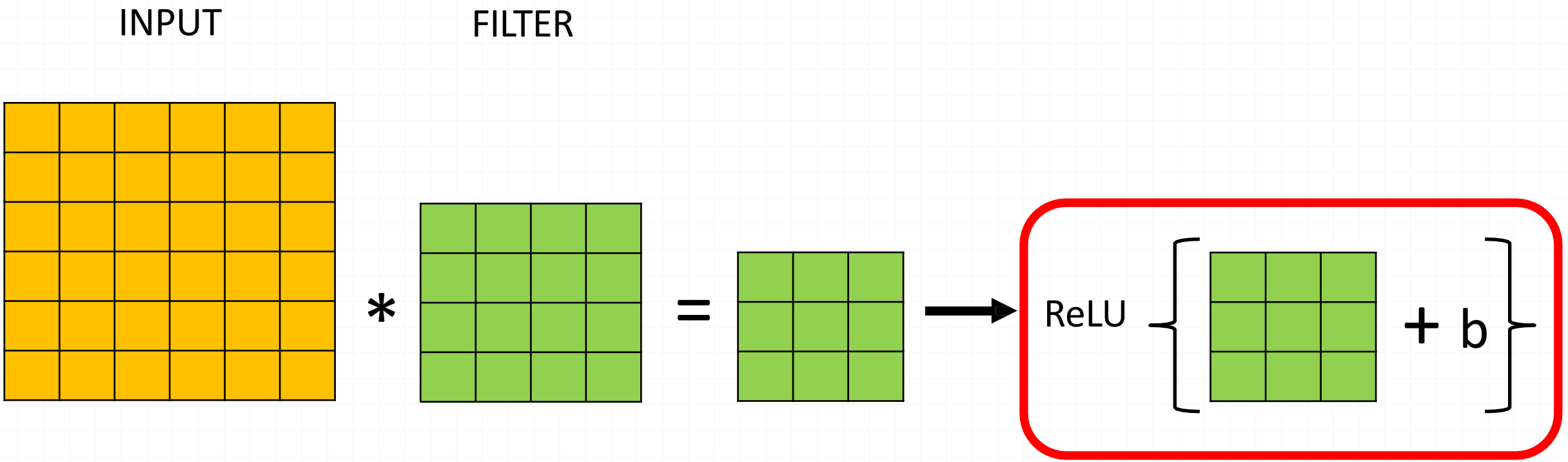
Demo: http://beej.us/blog/data/convolution-image-processing/

Key Ingredient 1: Convolutional Layers



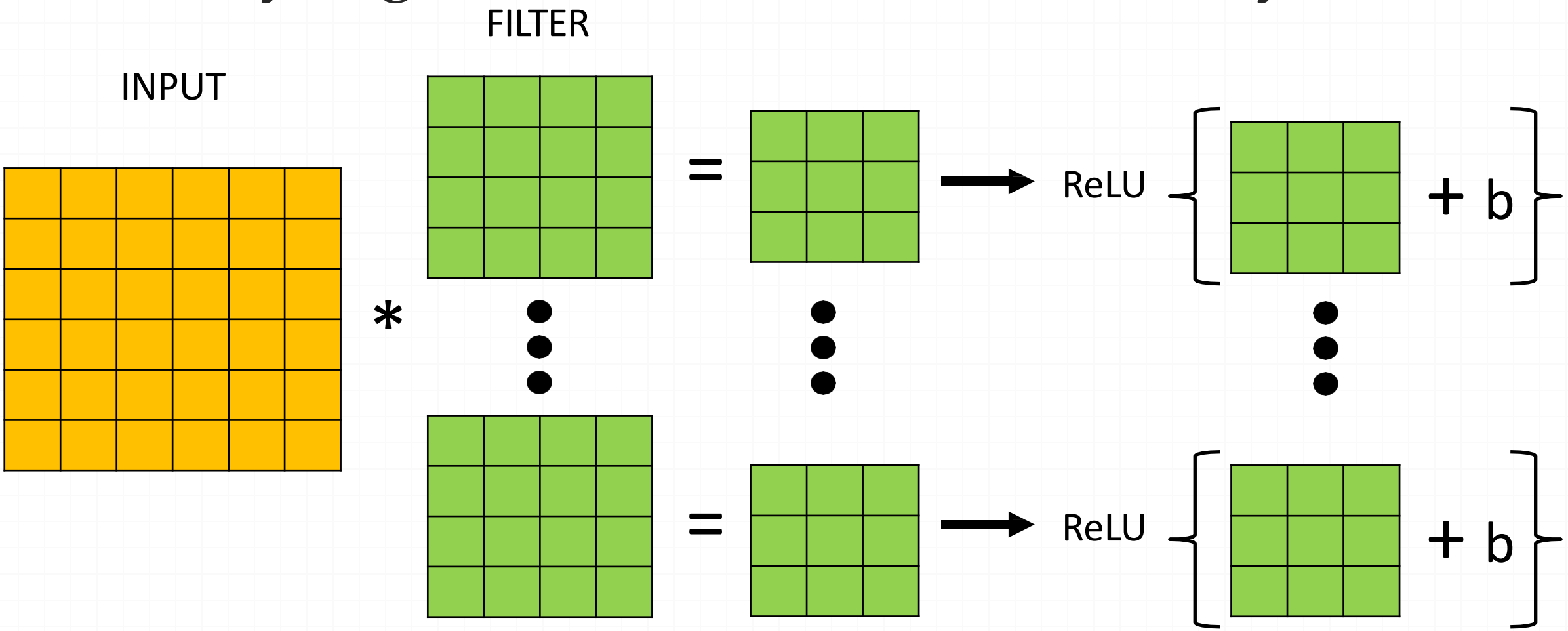
Can choose filters of any size to support feature learning!

Key Ingredient 1: Convolutional Layers



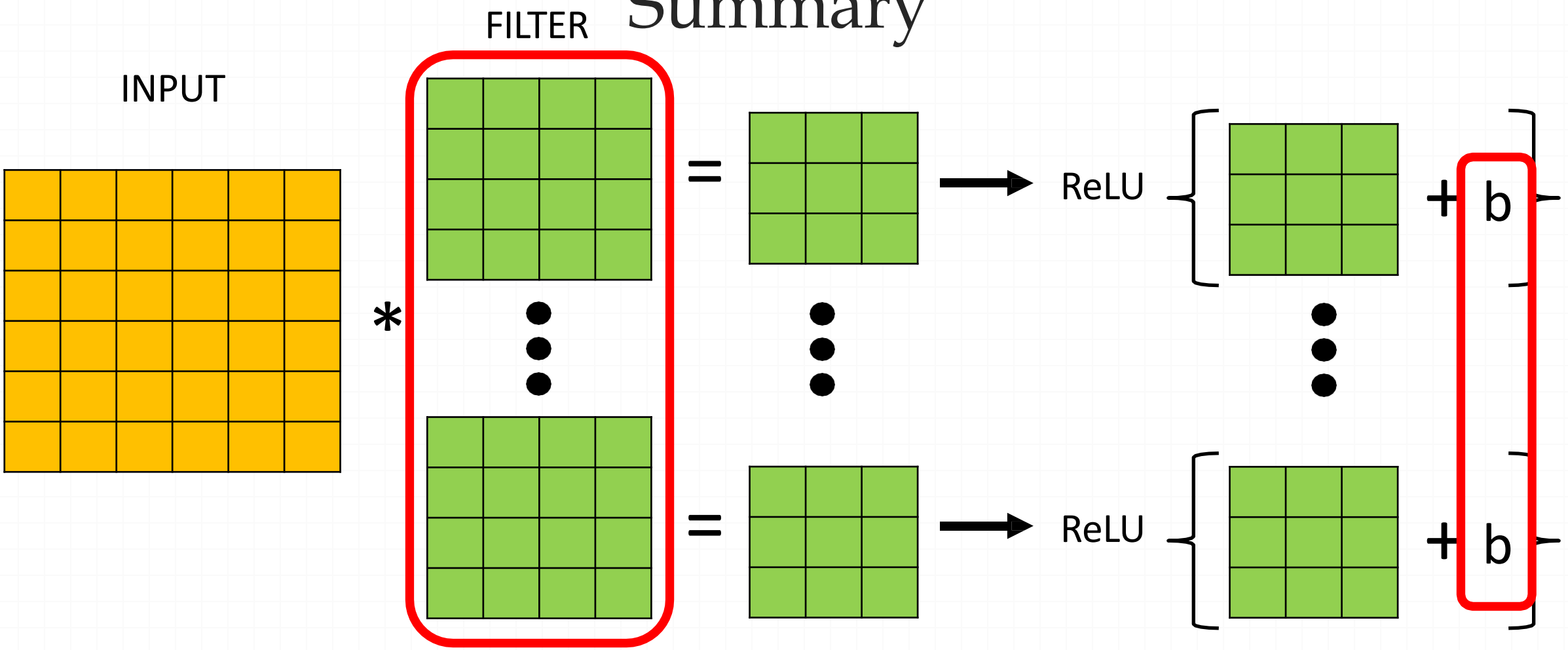
Filtered results are passed, with a bias term, through an activation function to create **activation/feature maps**

Key Ingredient 1: Convolutional Layers



Can have multiple filters (with a unique bias parameter per filter)

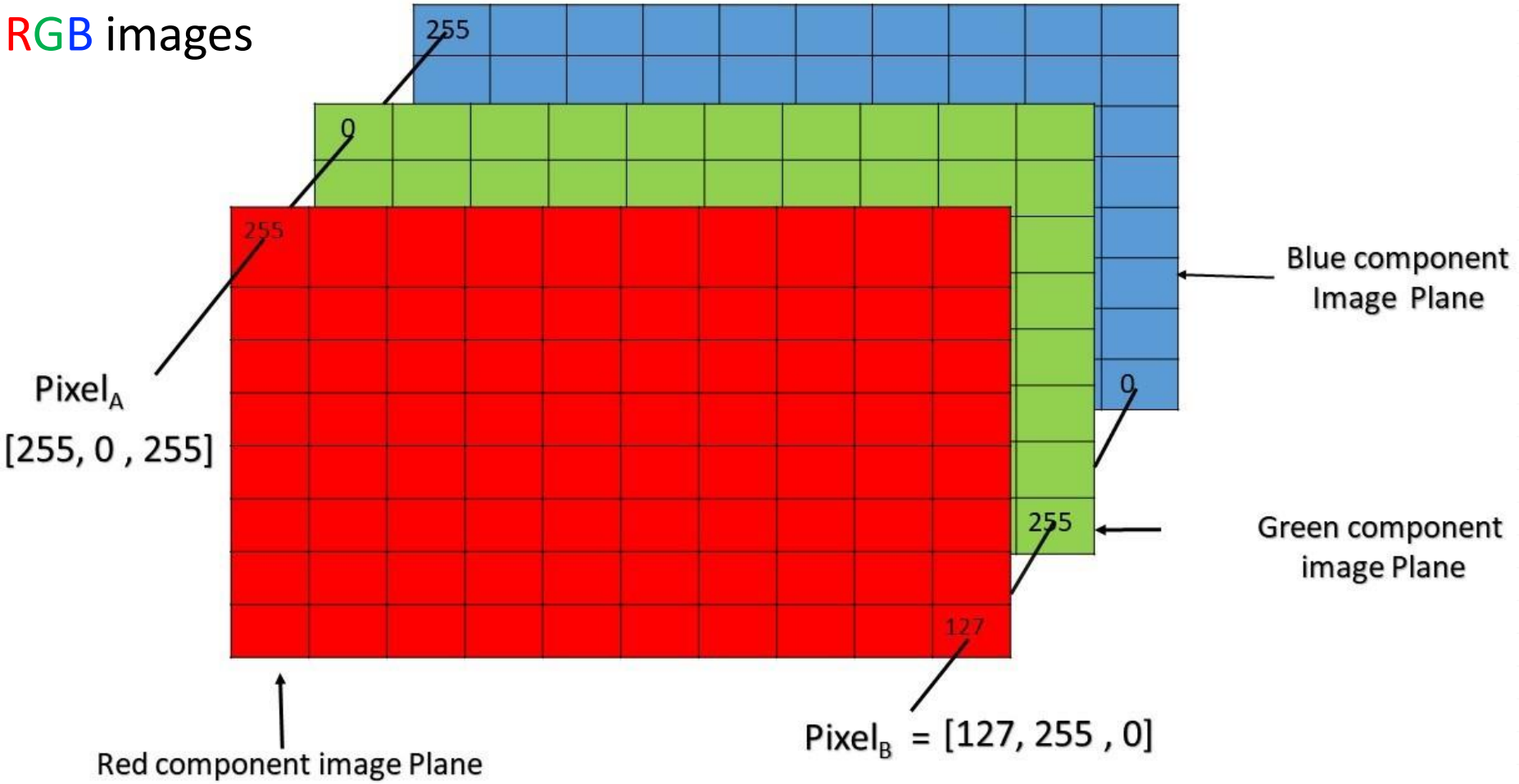
Key Ingredient 1: Convolutional Layer Summary



Neural networks learn values for all filters and biases in all layers

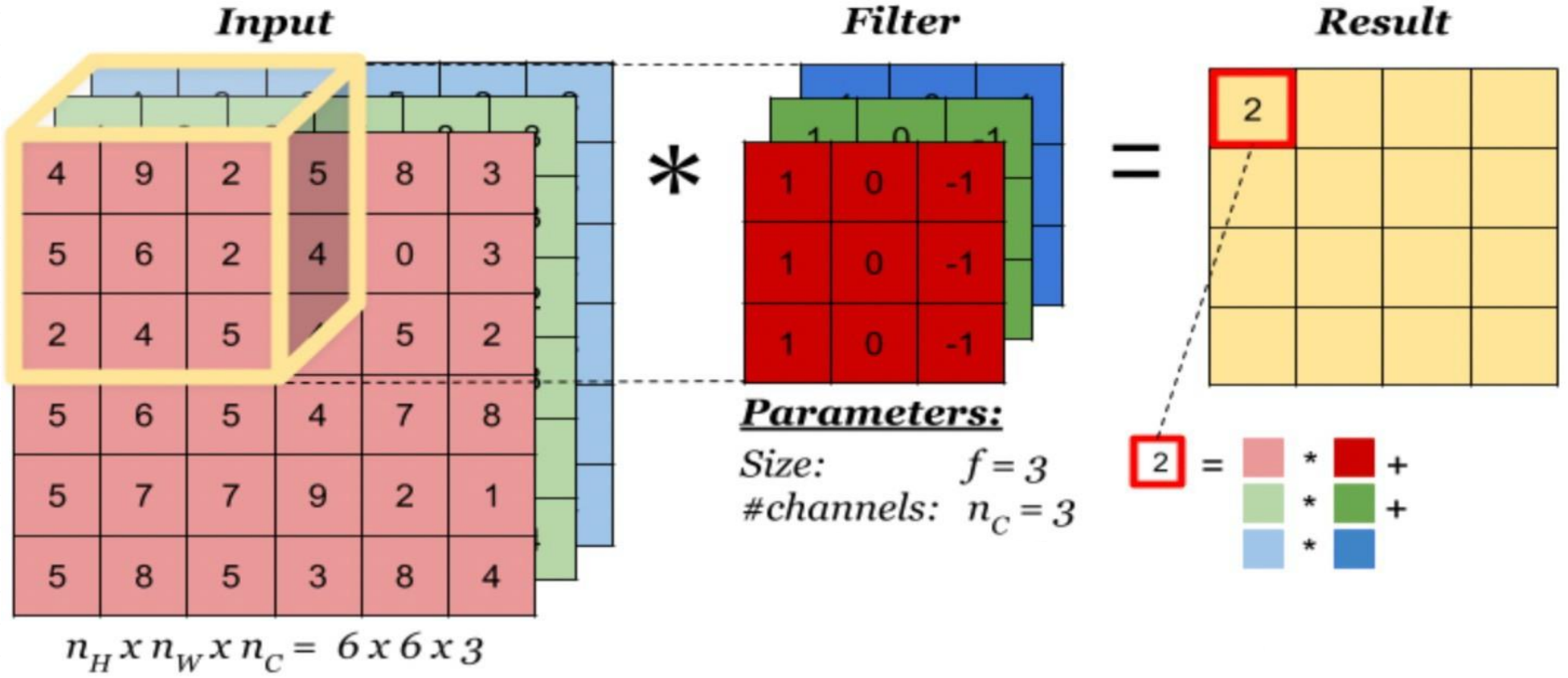
How Filters Are Applied to Multi-Channel Inputs

e.g., RGB images



https://www.geeksforgeeks.org/matlab-rgb-image-representation/

How Filters Are Applied to Multi-Channel Inputs

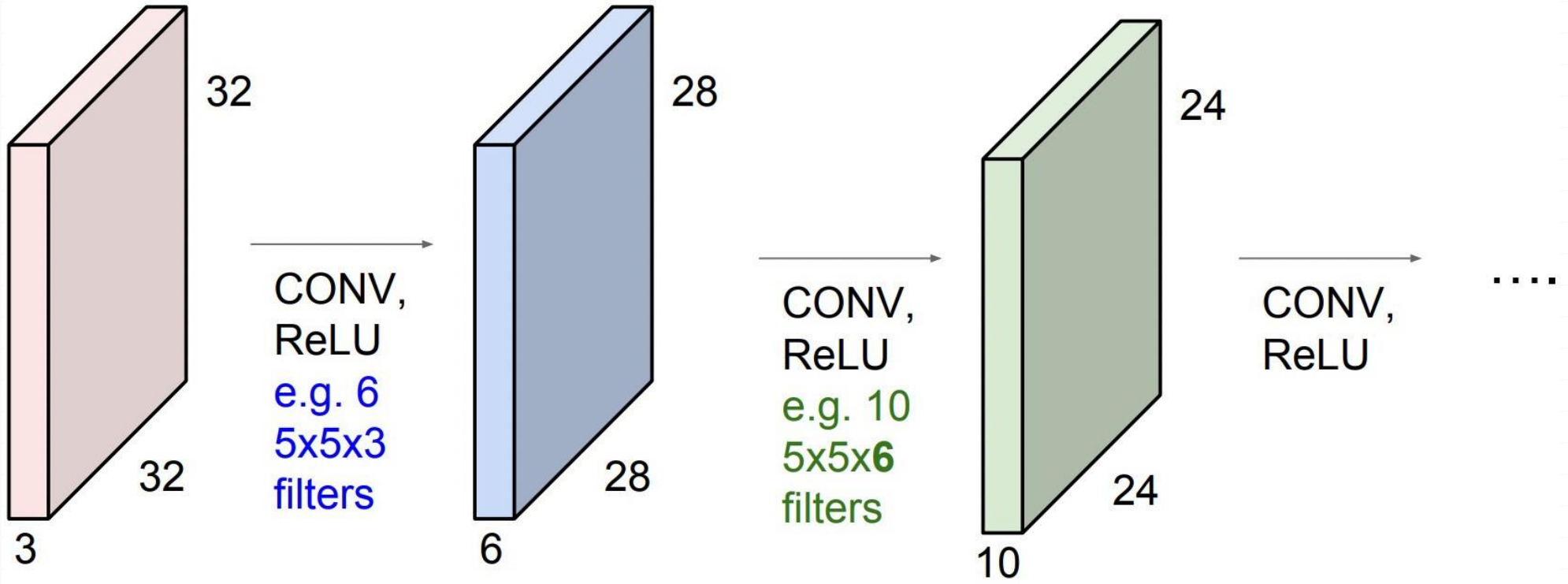


Number of channels in a filter matches that of the input

https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/

Convolutional Layers Stacked

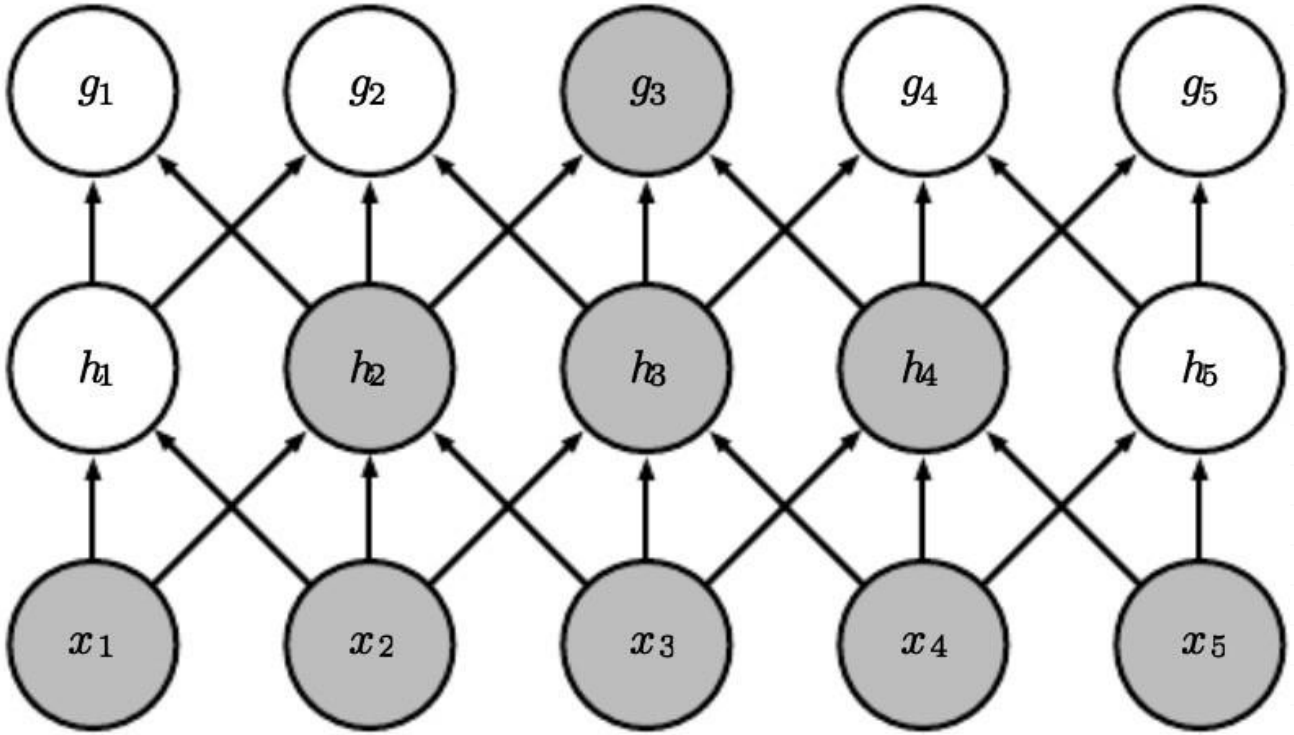
Can then stack a sequence of convolution layers; e.g.,



http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture05.pdf

Convolutional Layers Stacked

Can then stack a sequence of convolution layers, which leads to identifying patterns in increasingly **larger regions of the input (e.g., pixel) space:**

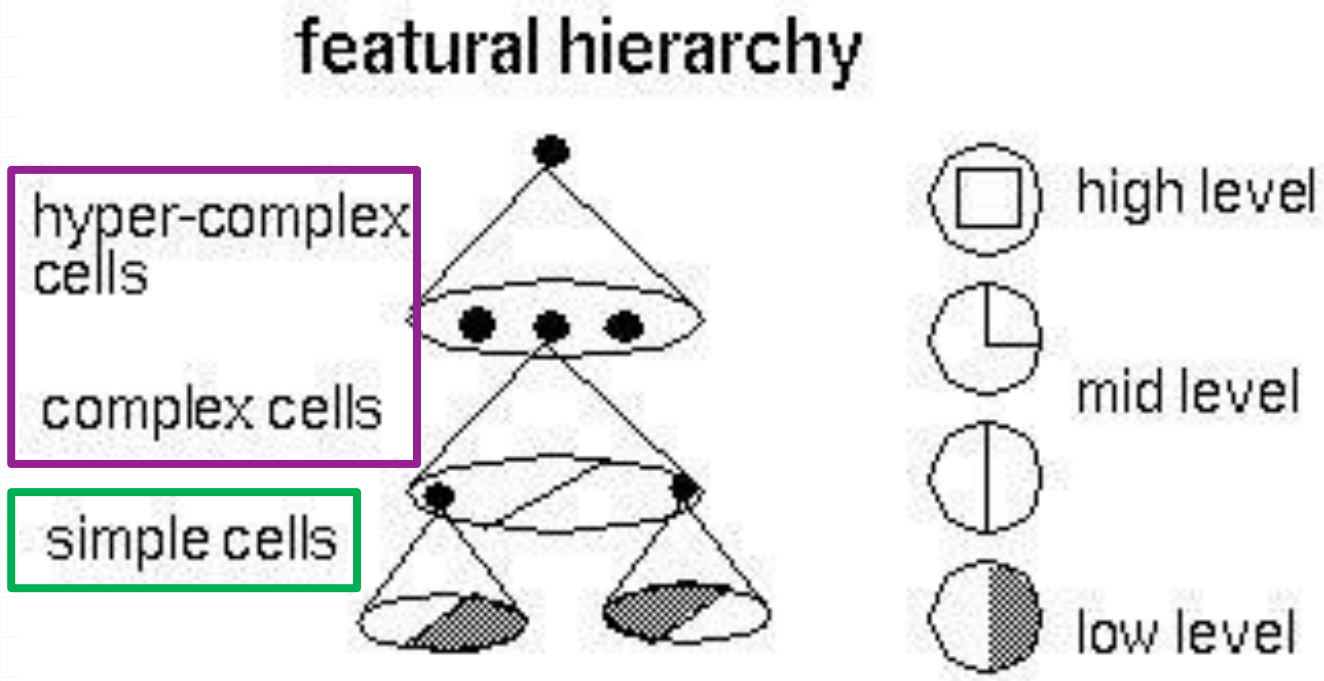


https://www.deeplearningbook.org/contents/convnets.html

Convolutional Layers Stacked

Can then stack a sequence of convolution layers, which leads to identifying patterns in increasingly **larger regions of the input (e.g., pixel) space** and **mimicking vision system**:

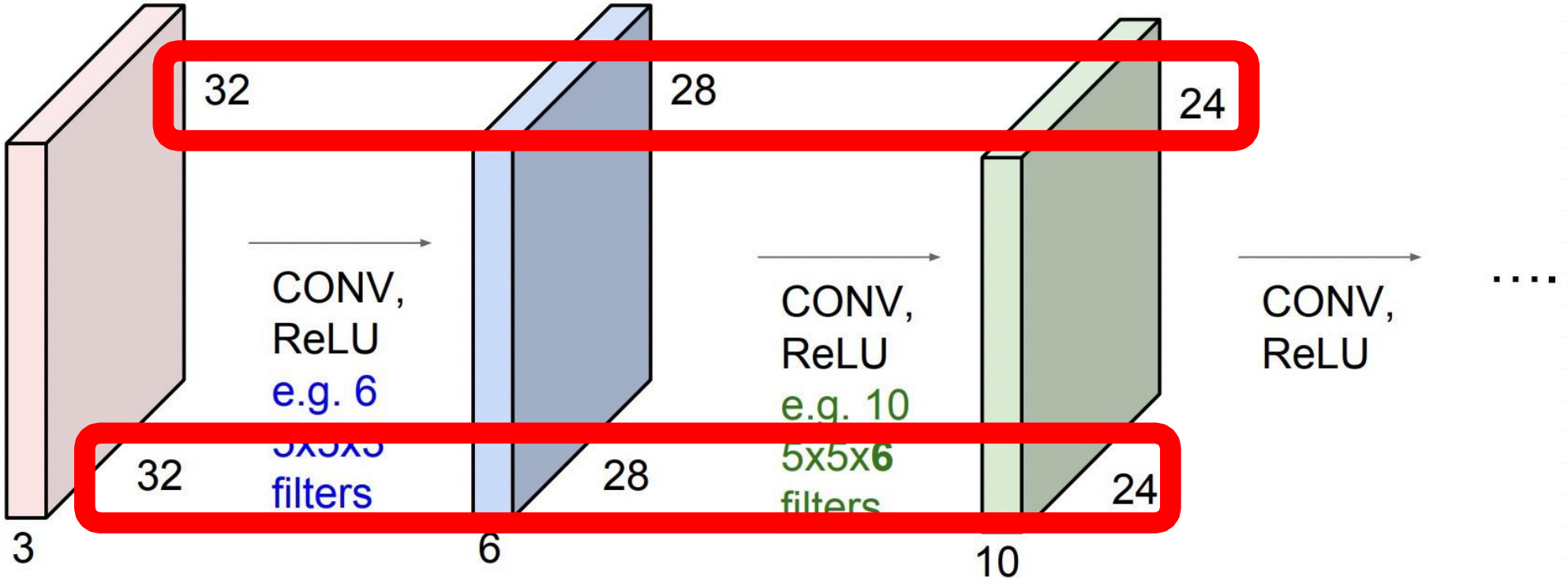
Higher level features are constructed by combining lower level features



Source: https://bruceoutdoors.files.wordpress.com/2017/08/hubel.jpg

Problem #1: Input Shrinks

Why do the dimensions shrink with each convolutional layer?



Information is lost around boundary of the input!

http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture05.pdf

Solution: Control Output Size with Padding

- **Padding**: add values at the boundaries

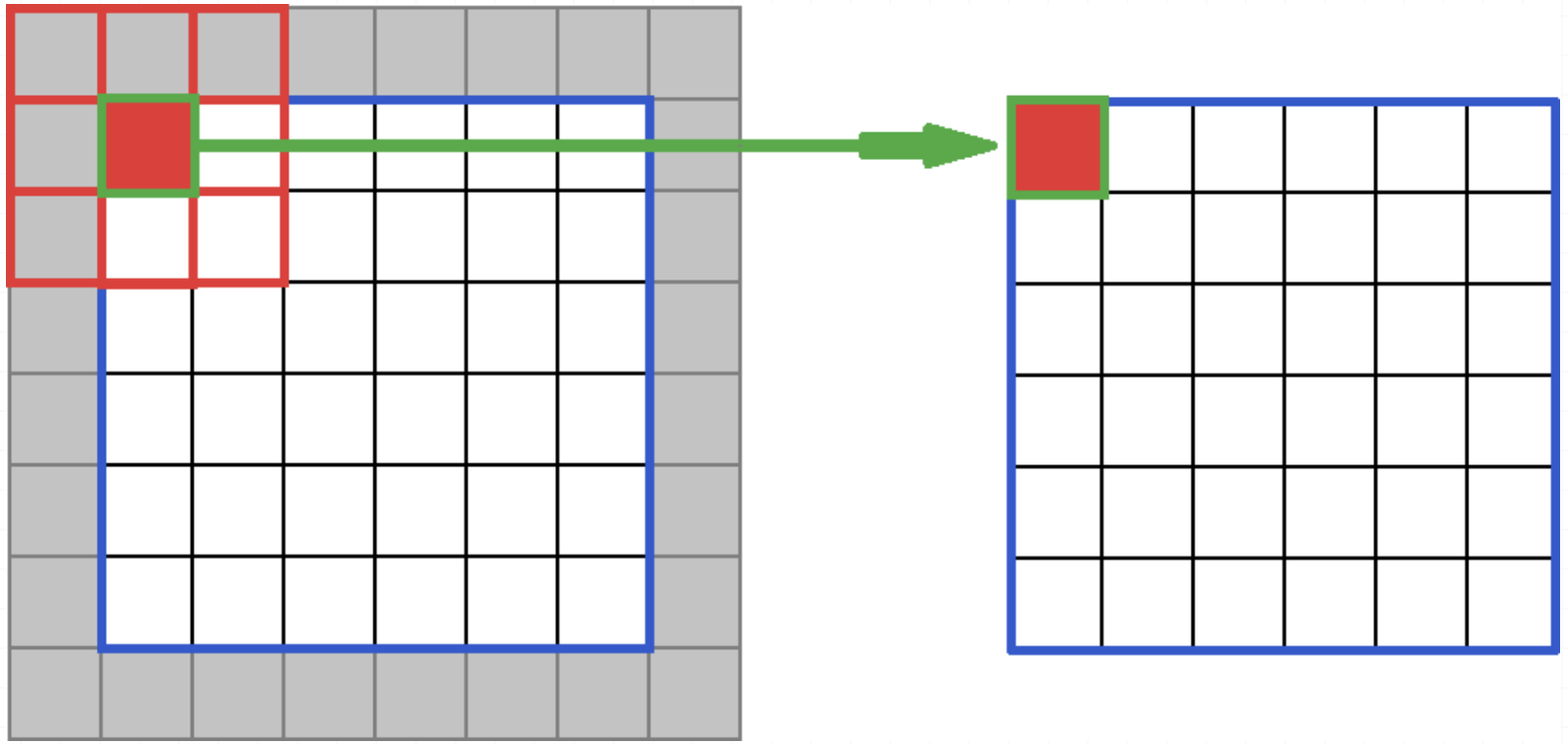
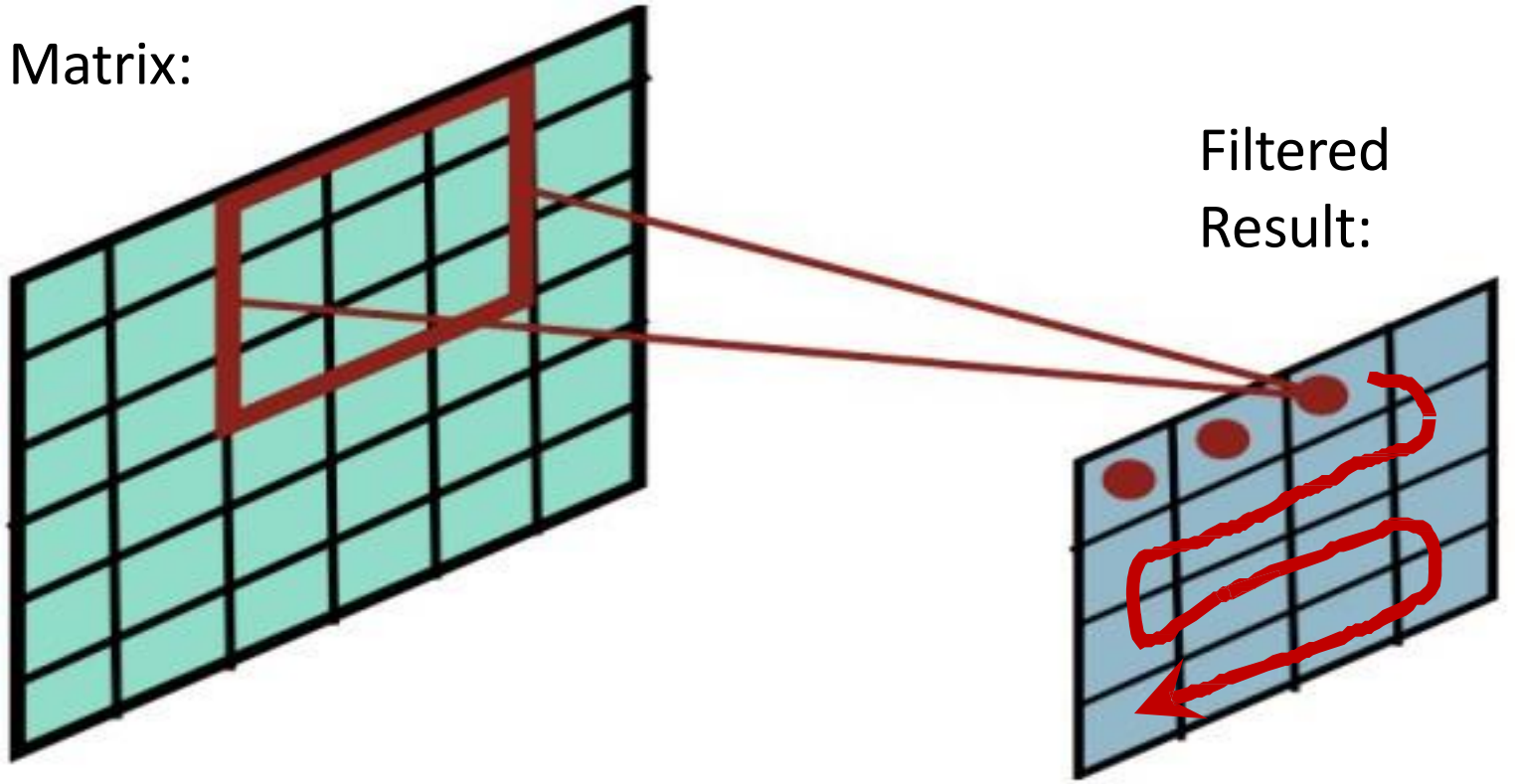


Image Credit: https://software.intel.com/en-us/node/586159

Problem #2: Computation Expensive



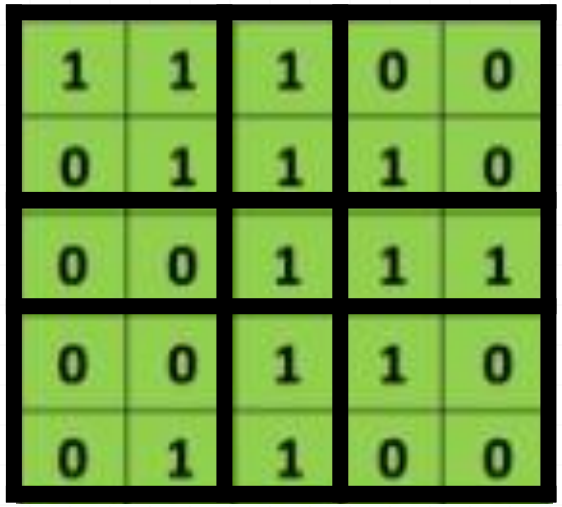
Many computations to slide filter over every point in the matrix and compute dot products

https://people.eecs.berkeley.edu/~jrs/189/lec/cnn.pdf

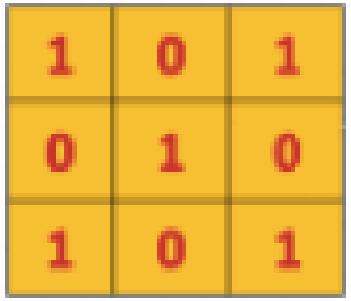
Idea: Reduce Computations with Stride

- **Stride**: how many steps taken spatially before applying a filter
  - e.g., 2x2

Image



Filter



Feature Map

4	4
2	4

http://deeplearning.net/software/theano/tutorial/conv_arithmetic.html

Convolutional Layers: Parameters vs Hyperparameters

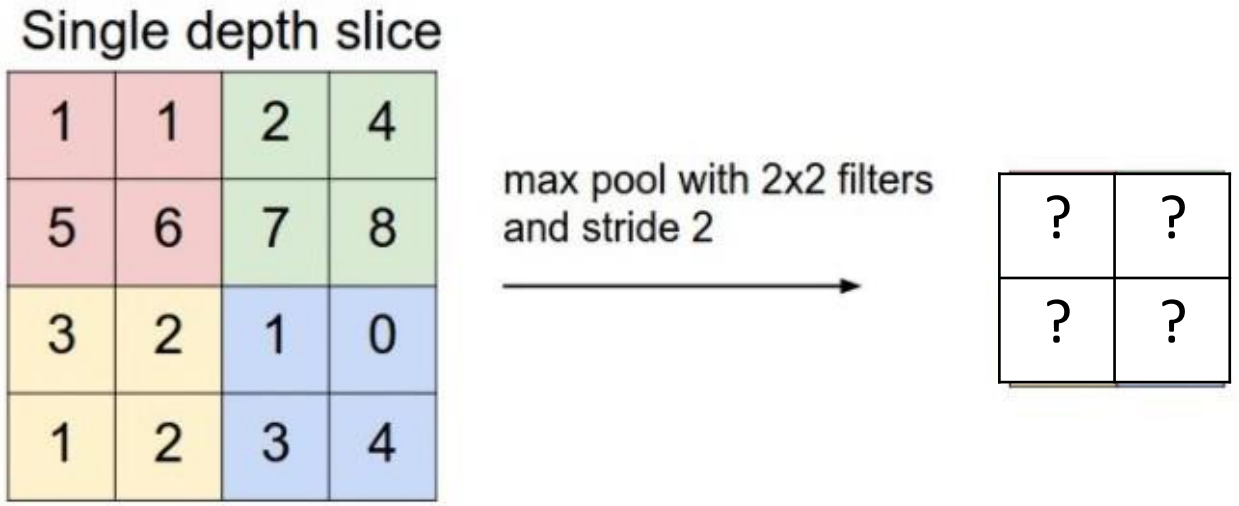
- Parameters
  - Weights
  - Biases
- Hyperparameters:
  - Number of filters, including height and width of each
  - Padding type
  - Strides

Today's Topics

- Neural Networks for Spatial Data
- History of Convolutional Neural Networks (CNNs)
- CNNs – Convolutional Layers
- CNNs – Pooling Layers
- Programming Tutorial

Pooling Layer: Summarizes Neighborhood

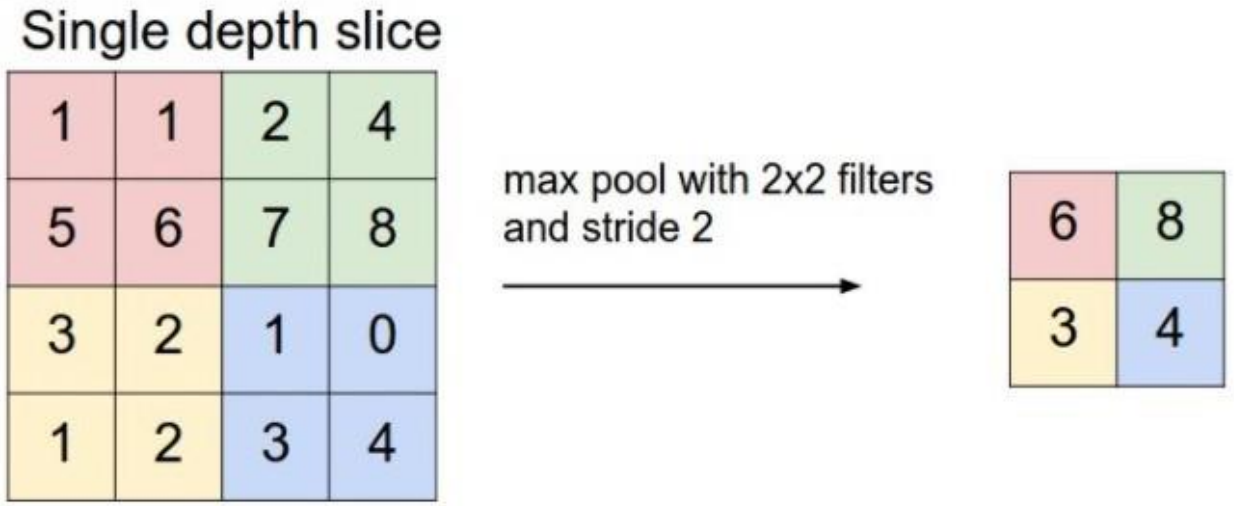
- **Max-pooling***:* partitions input into a set of non-overlapping rectangles and outputs the maximum value for each chunk



http://cs231n.github.io/convolutional-networks/#pool

Pooling Layer: Summarizes Neighborhood

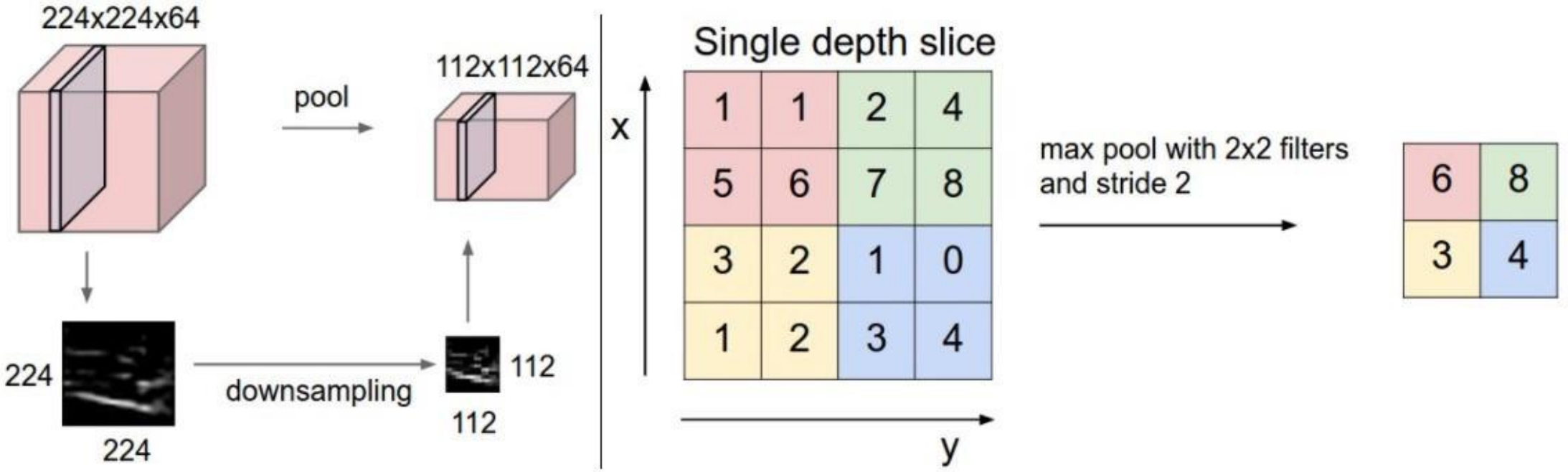
- **Max-pooling***:* partitions input into a set of non-overlapping rectangles and outputs the maximum value for each chunk



http://cs231n.github.io/convolutional-networks/#pool

Pooling Layer: Summarizes Neighborhood

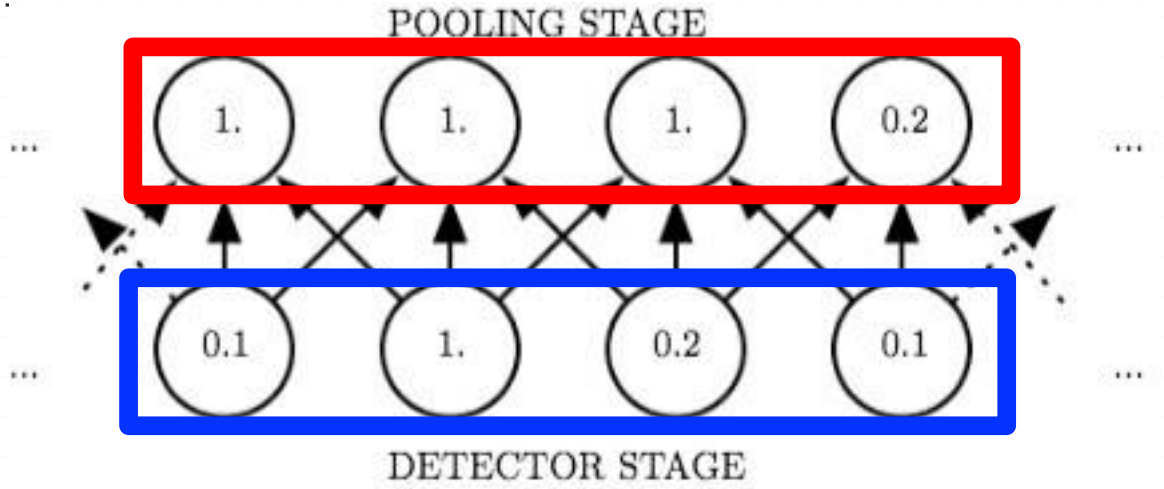
- **Max-pooling***:* partitions input into a set of non-overlapping rectangles and outputs the maximum value for each chunk



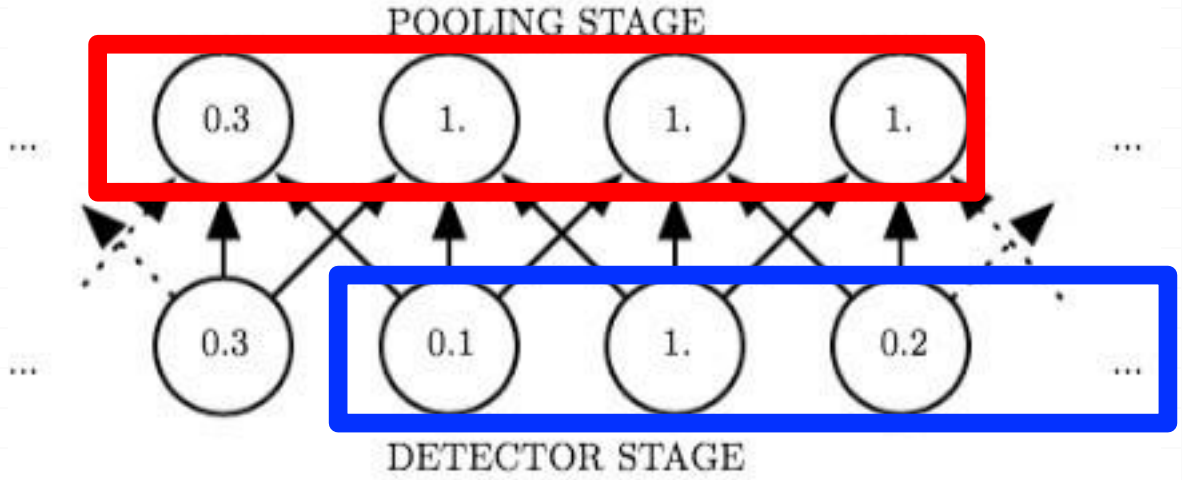
http://cs231n.github.io/convolutional-networks/#pool

Pooling

- Resilient to small translations



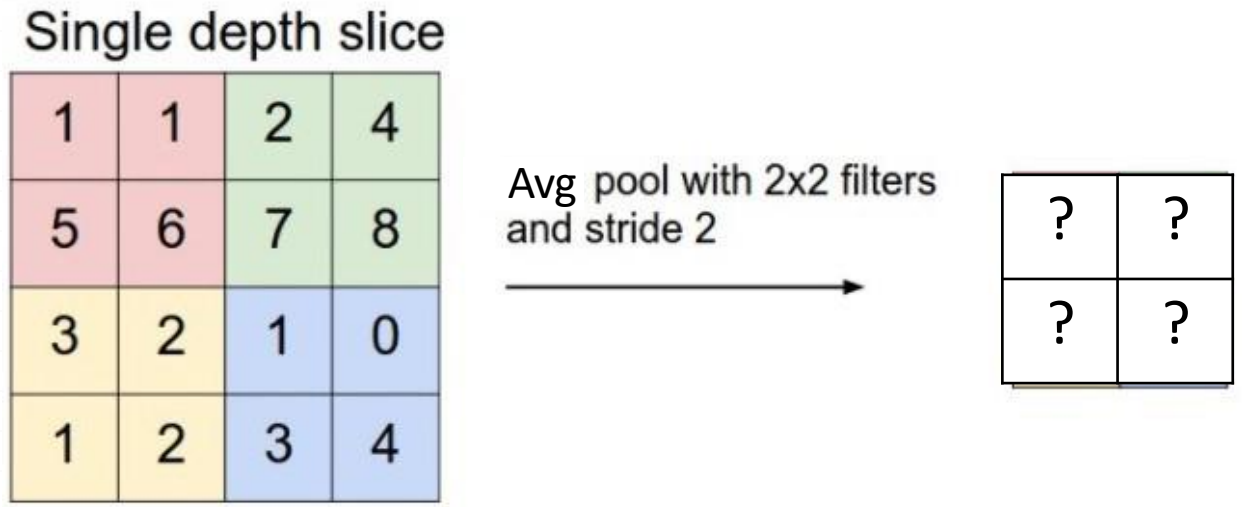
- e.g.,
  - Input: all values change (shift right)
  - Output: only half the values change



https://www.deeplearningbook.org/contents/convnets.html

Pooling Layer: Summarizes Neighborhood

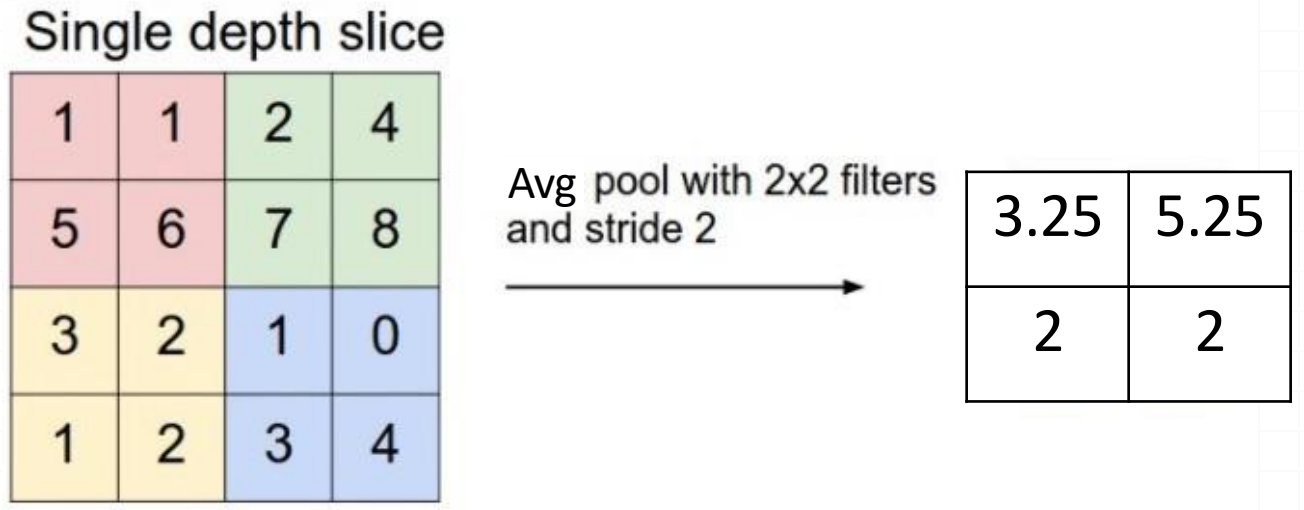
- **Max-pooling***:* partitions input into a set of non-overlapping rectangles and outputs the maximum value for each chunk
- **Average-pooling***:* partitions input into a set of non-overlapping rectangles and outputs the average value for each chunk



http://cs231n.github.io/convolutional-networks/#pool

Pooling Layer: Summarizes Neighborhood

- **Max-pooling***:* partitions input into a set of non-overlapping rectangles and outputs the maximum value for each chunk
- **Average-pooling***:* partitions input into a set of non-overlapping rectangles and outputs the average value for each chunk



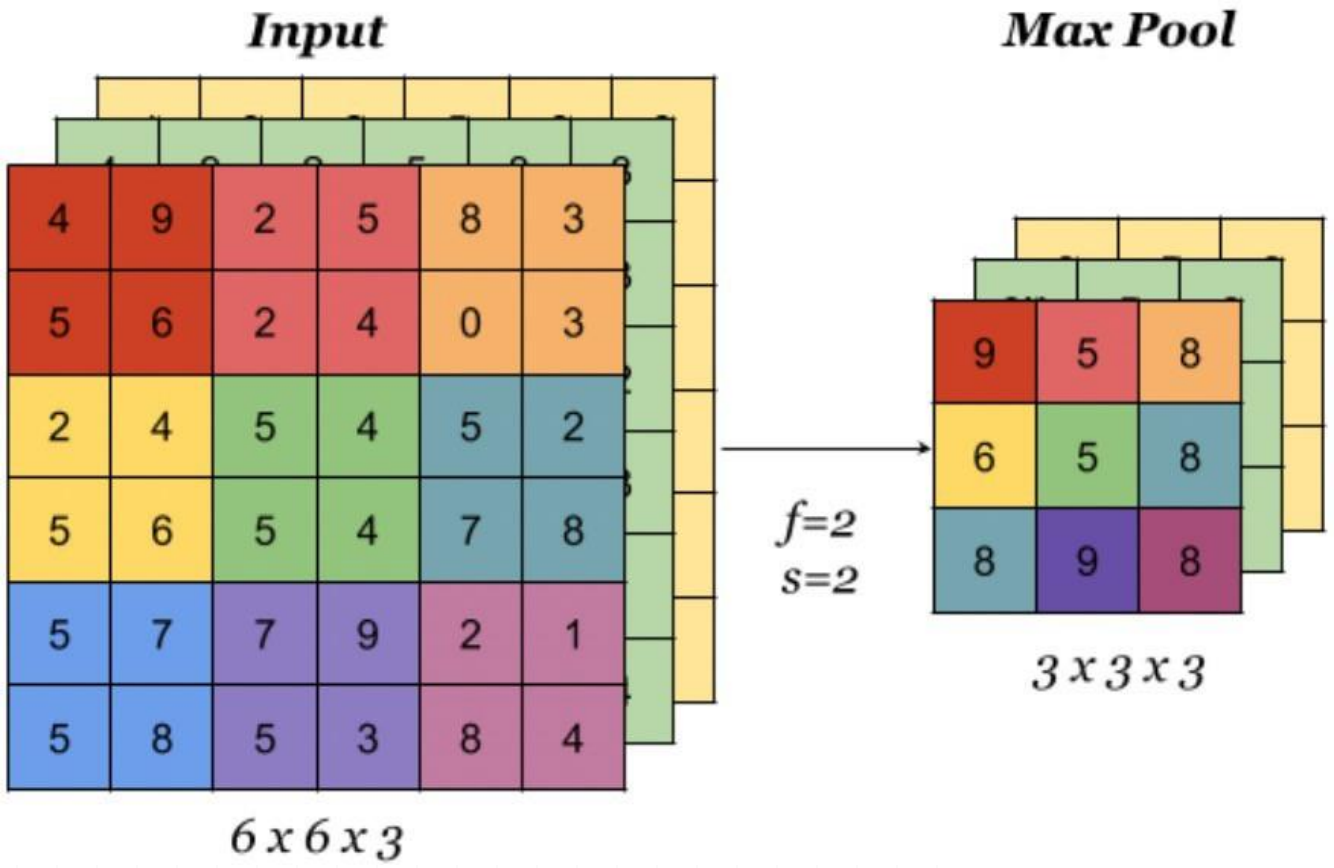
http://cs231n.github.io/convolutional-networks/#pool

Pooling Layer: Summarizes Neighborhood

- **Max-pooling***:* partitions input into a set of non-overlapping rectangles and outputs the maximum value for each chunk
- **Average-pooling***:* partitions input into a set of non-overlapping rectangles and outputs the average value for each chunk

- And many more pooling options
  - E.g., listed here https://pytorch.org/docs/stable/ [illegible]tml#pooling-layers

Pooling for Multi-Channel Input



Pooling is applied to each input channel separately

https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/

Pooling Layer: Benefits

- Builds in invariance to translations of the input
- Reduces memory requirements
- Reduces computational requirements