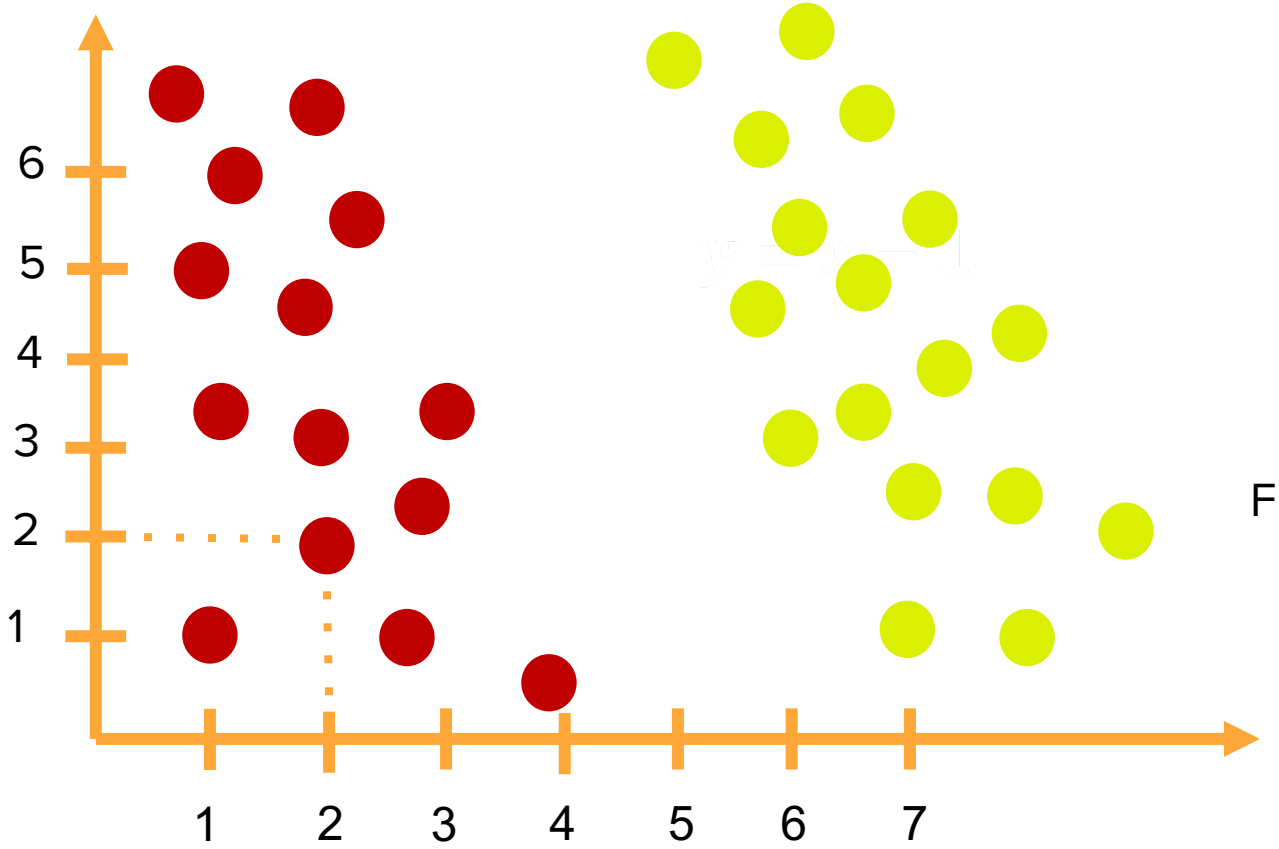
Introduction to Artificial Neural Networks - DNN

Content

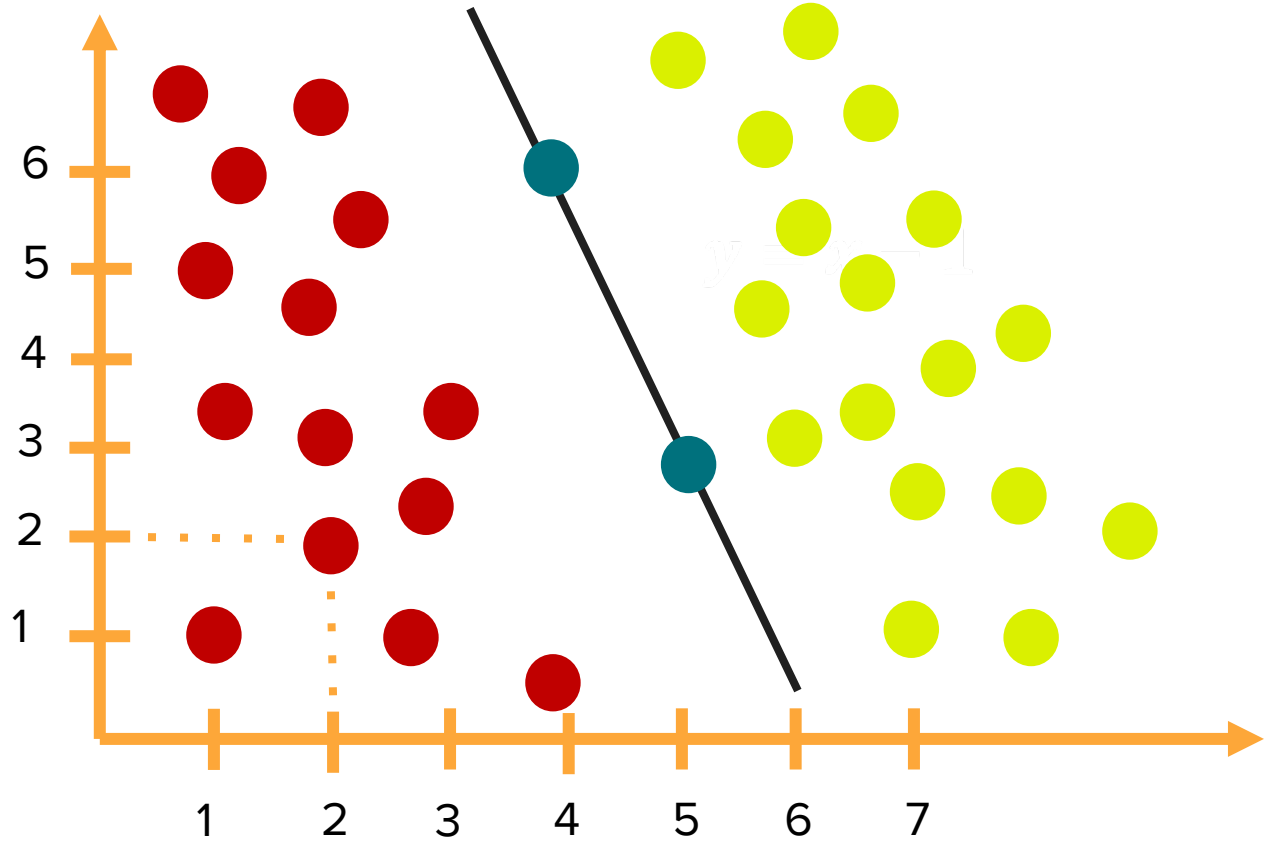
- Introduction to DNN
- Feed forward and Backpropagation
- Activation Functions
- Loss Functions
- Training and Evaluation
- Gradient Descent Categories
- Implementation using Tensorflow.

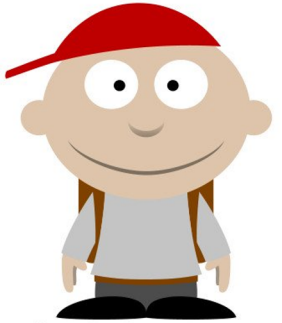
Introduction to DNN



Find a line to separate red and yellow points.

Introduction to DNN

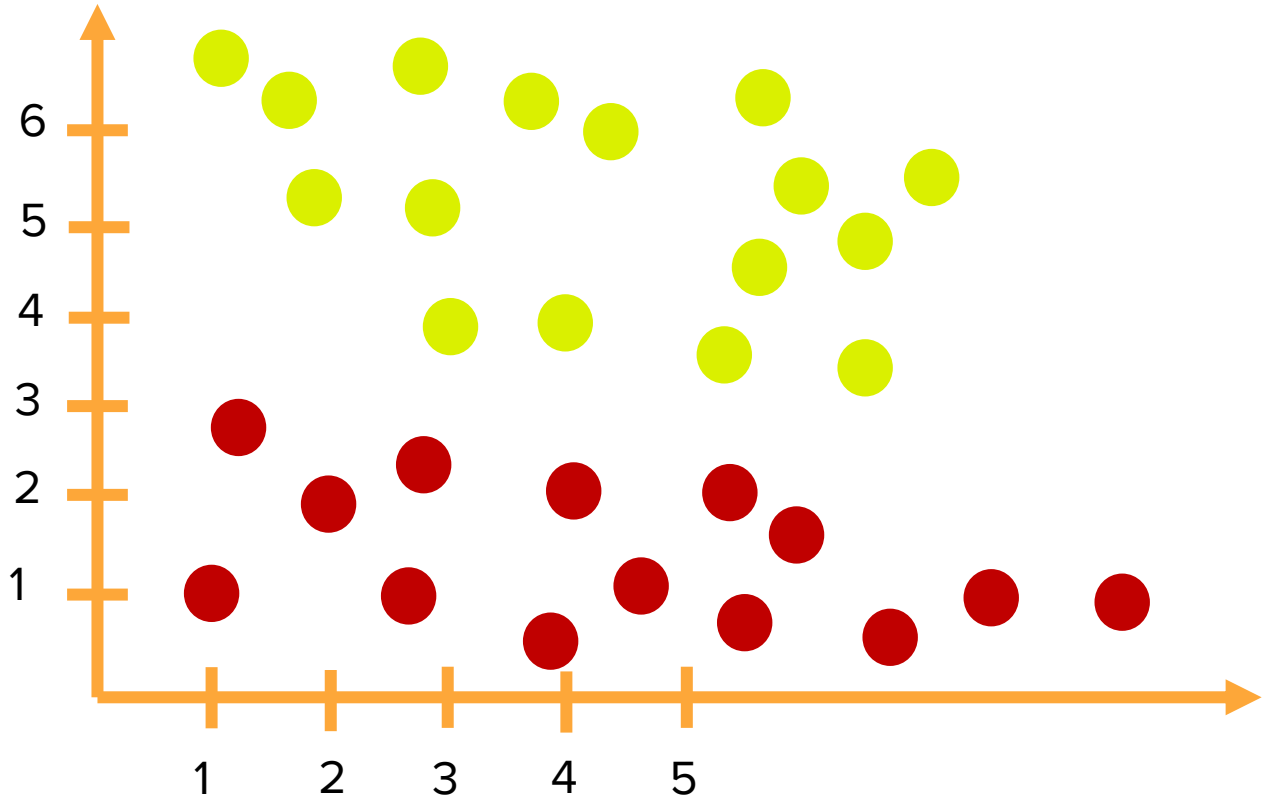




Hmm…

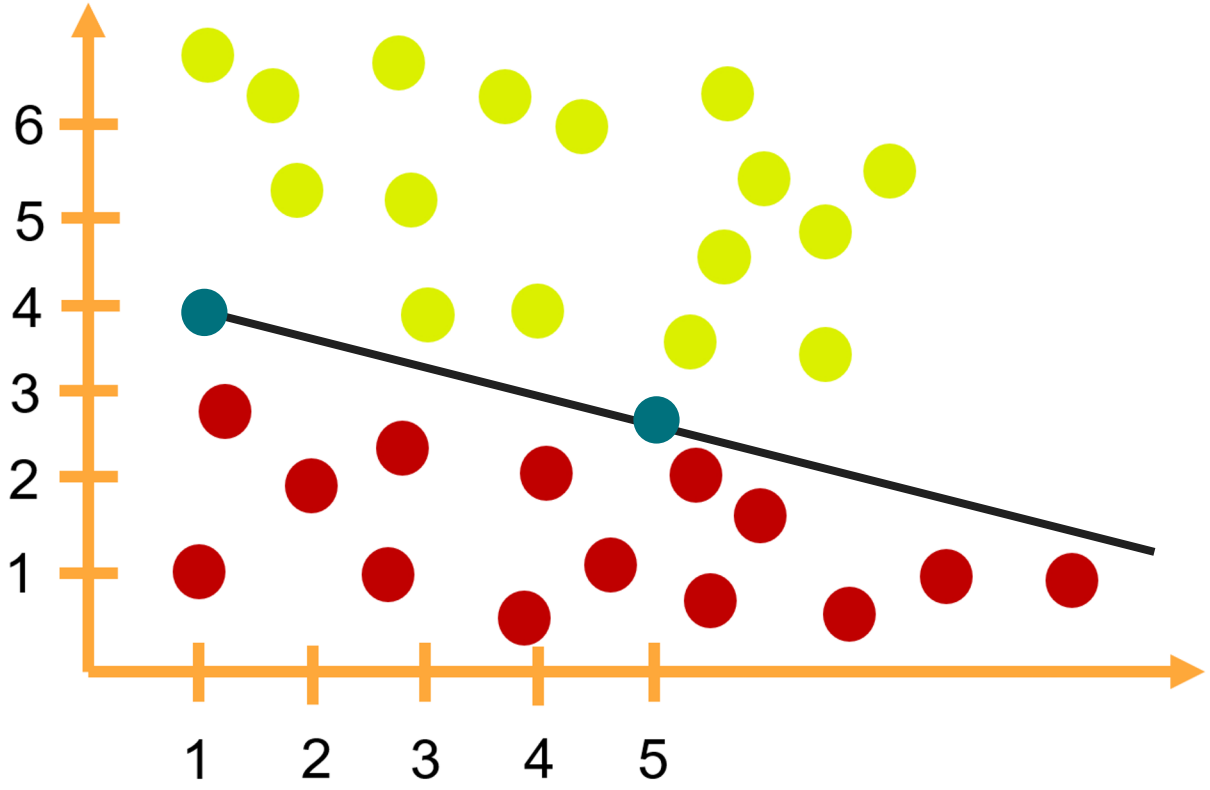
$y = -3x + 18$

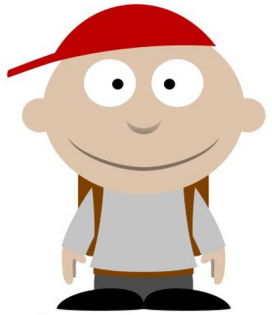
Introduction to DNN



Next

Introduction to DNN

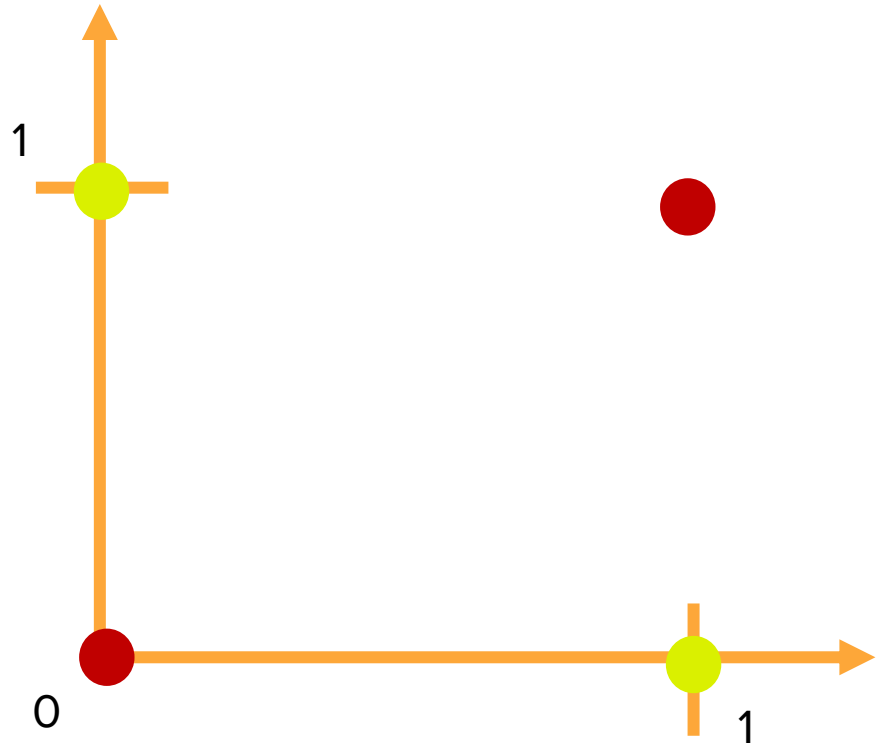




Hmm…

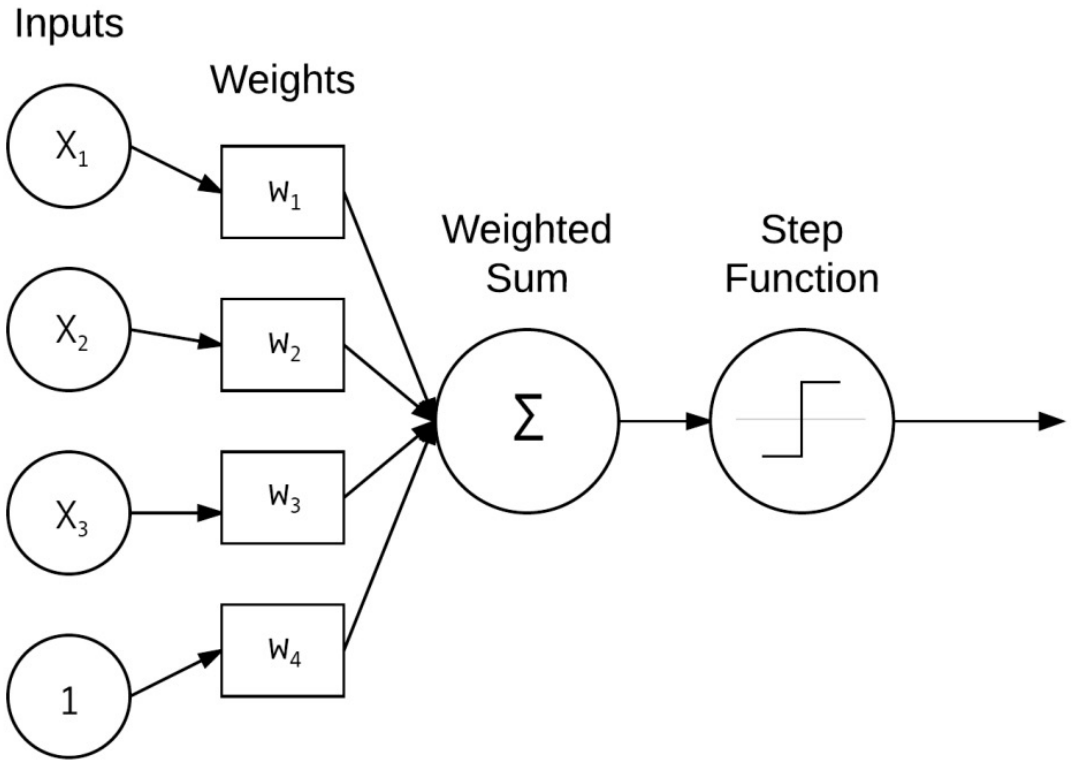
$$y = -\frac{1}{4}x + \frac{17}{4}$$

Introduction to DNN



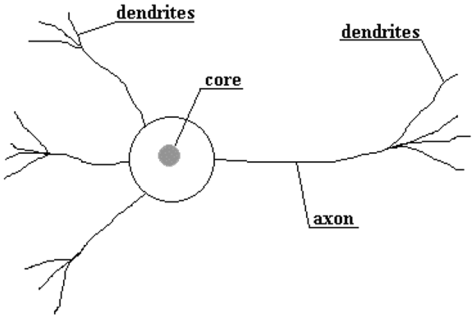
Difficult with only one line!

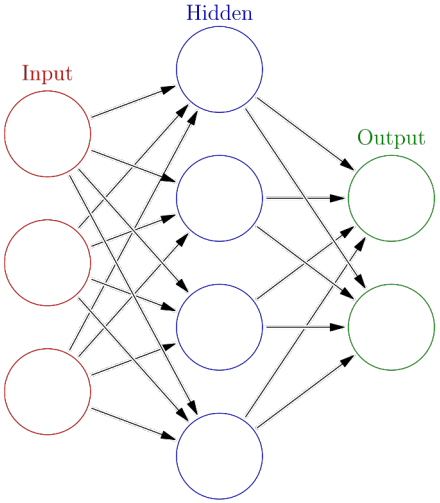
Introduction to DNN - Perceptron



Neural Network

- A kind of computational model as a set of simple computational units (neurons and their connections).
- Alias names
  - Feed forward networks
  - Artificial neural networks
- Architecture origins from human brain.
- Commonly applied in image classification, text classification, etc.





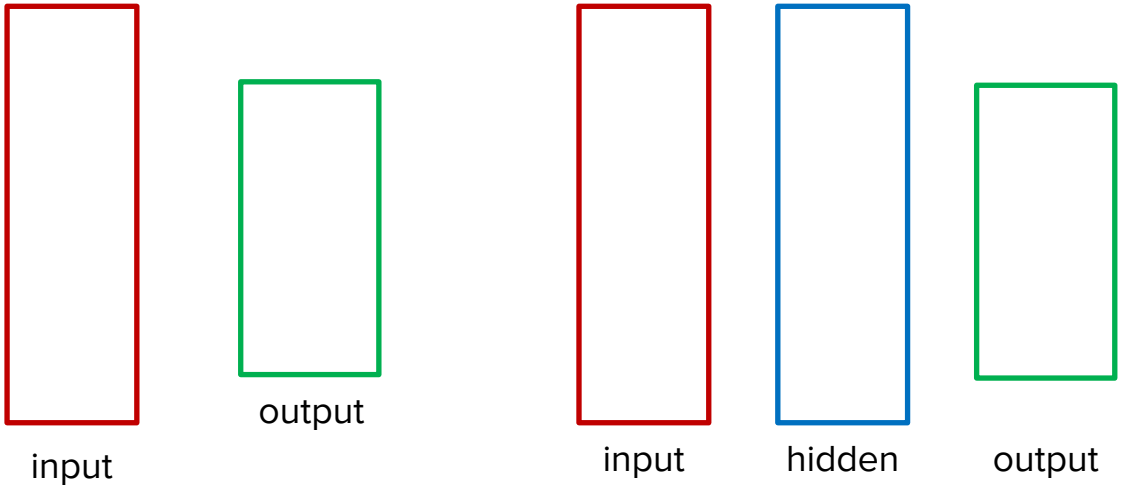
Source: Wikipedia

Architecture

- Concepts:
  - Layer
  - Neuron
  - Weight
  - Bias
- Characteristics:
  - Fully connected
  - Number of hidden layers
  - Number of neurons in hidden layers

Layer

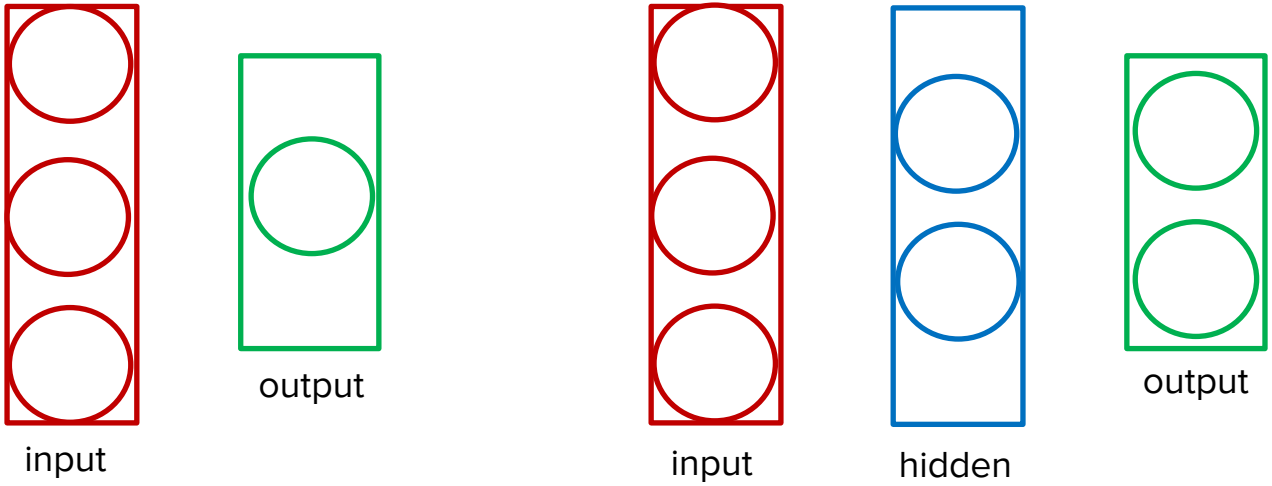
- Layer ➔ a set of neuron consisting of computational results
- Three kinds of layer:
  - Input layer
  - Hidden layer (optional)
  - Output layer



Source: Wikipedia

Neuron

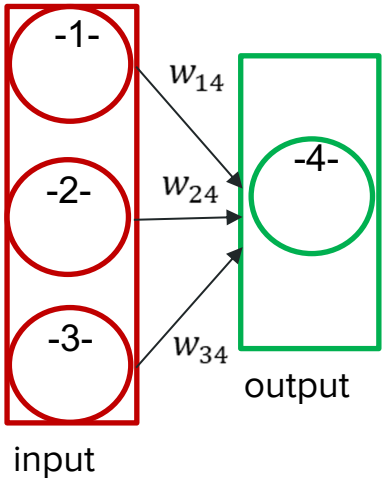
- Neuron ➜ a unit consisting of computational result

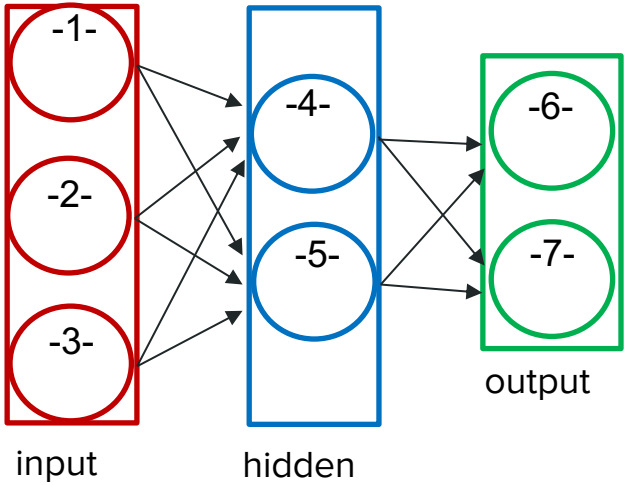


Source: Wikipedia

Weight

- Weight is the value on the connection between to neurons.

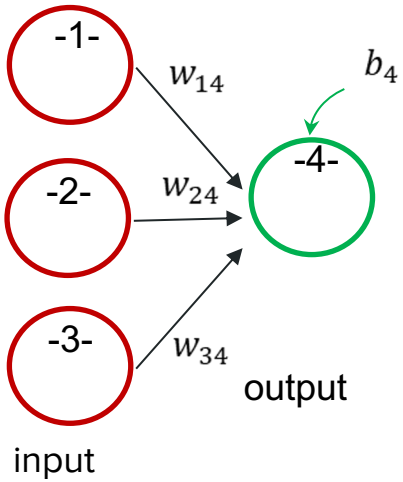


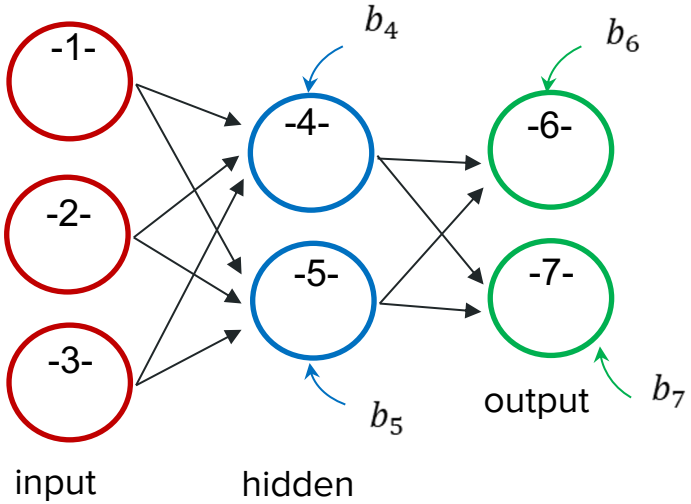


Source: Wikipedia

Bias

- Bias is applied for every neuron, excluding output ones.
- Bias helps the model to be more “flexible”





Source: Wikipedia

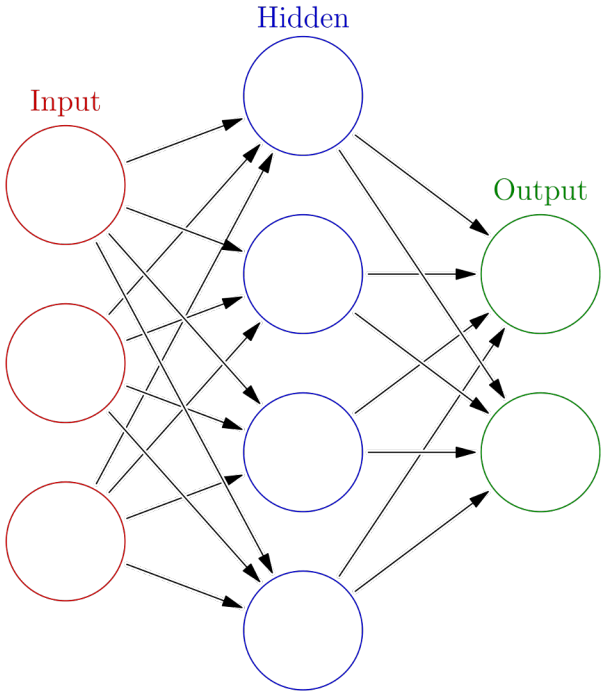
Architecture

- Concepts:
  - Layer
  - Neuron
  - Weight
  - Bias

  } **parameters**
- Characteristics:
  - Fully connected
  - Number of hidden layers
  - Number of neurons in hidden layers

Characteristics

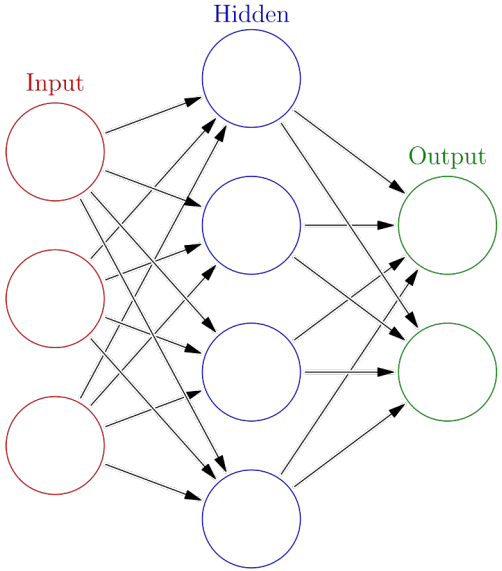
- Fully connected: each neuron in layer k is connected to all neurons in layer k+1



Source: Wikipedia

Characteristics

- Number of neurons in layers and number of layers depend on the problem.
- The more number of hidden layers and number of neurons in layers, the more complicated the neural network is.



Source: Wikipedia

Architecture

- Concepts:
  - Layer
  - Neuron
  - Weight
  - Bias
- Characteristics:
  - Fully connected
  - Number of hidden layers
  - Number of neurons in hidden layers

} **hyper-parameters**

Feed forward

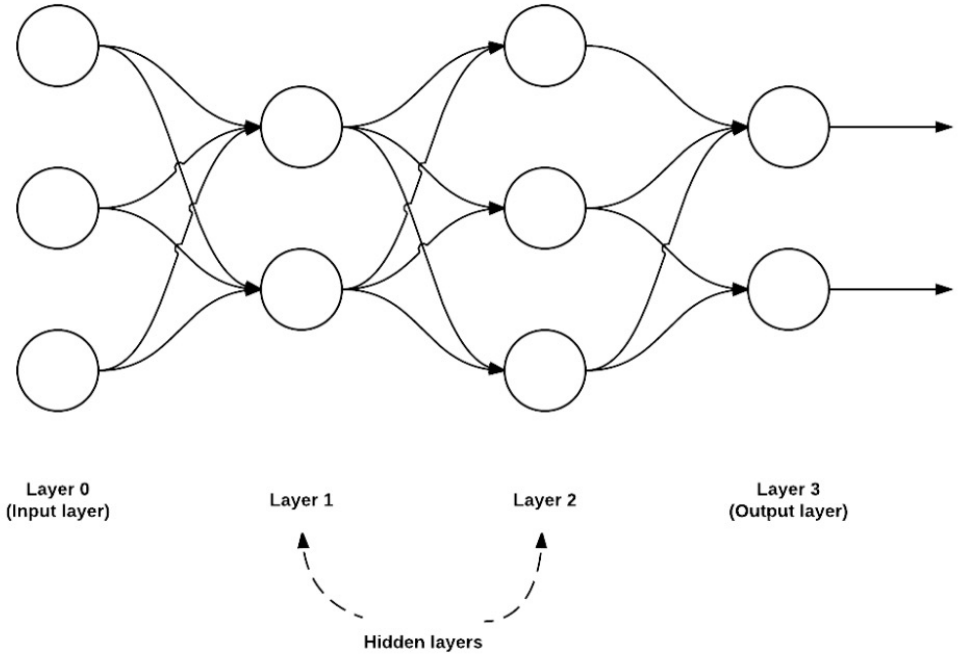
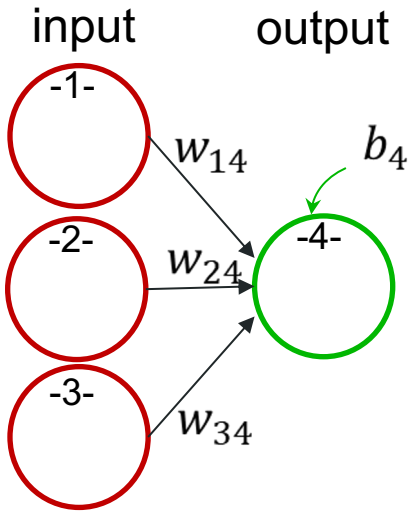


Figure 10.5: An example of a feedforward neural network with 3 input nodes, a hidden layer with 2 nodes a second hidden layer with 3 nodes, and a final output layer with 2 nodes.

Feed forward

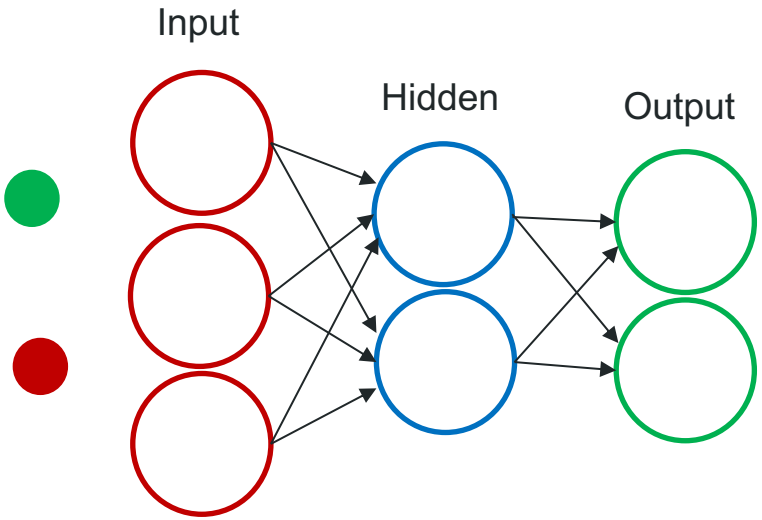


$$w_{14} = 0.1$$
$$w_{24} = 0.6 \quad b_4 = 0.8$$
$$w_{34} = 0.5$$

$$x_1 = 5$$
$$x_2 = 3$$
$$x_3 = 2$$

$$\hat{y}_4 = x_1 \times w_{14} + x_2 \times w_{24} + x_3 \times w_{34} + b_4$$
$$= 5 \times 0.1 + 3 \times 0.6 + 2 \times 0.5 + 0.8$$
$$= 0.5 + 1.8 + 1 + 0.8$$
$$= \mathbf{4.1}$$

Activation Functions



Activation function

$$\hat{y} = \left(\sum x.w\right) + bias$$

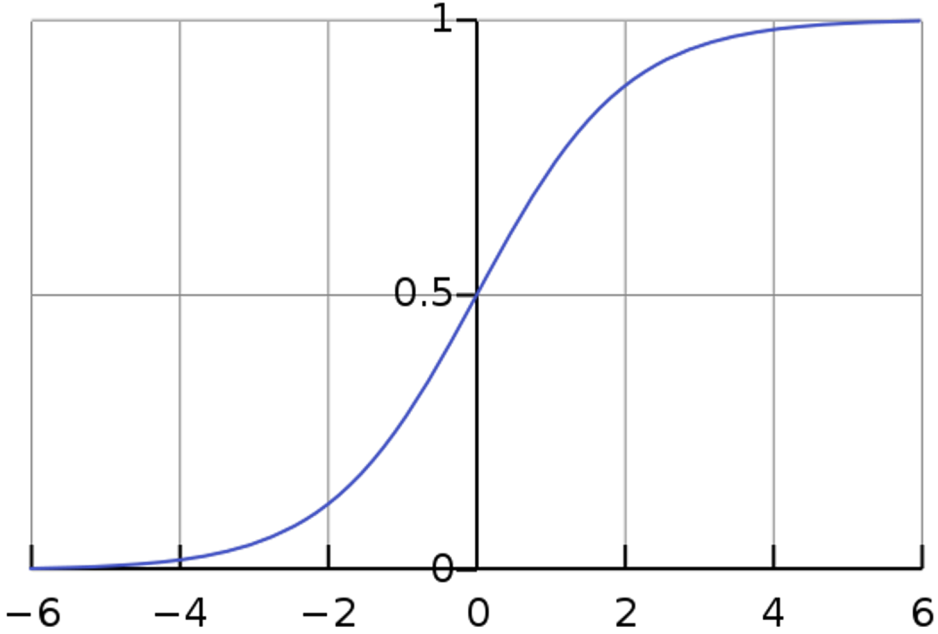
$\rightarrow \hat{y} \in (-\infty; +\infty)$

Which neuron should be “activated”?

Neural network is linear for any number of hidden layers

Activation Function

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



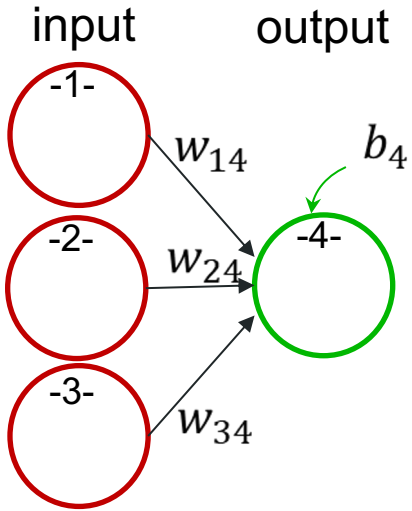
Activation Function

- Output of each neuron is handled by the activation function before being passed to the next neuron.

$$\hat{y} = \left(\sum x.w\right) + bias$$

$$\hat{y} = \text{sigmoid}\left(\left(\sum x.w\right) + bias\right)$$

Feed forward



$$x_1 = 5$$
$$x_2 = 3$$
$$x_3 = 2$$

$$w_{14} = 0.1$$
$$w_{24} = 0.6 \quad b_4 = 0.8$$
$$w_{34} = 0.5$$

tmp ~~$\hat{y}_4$~~ $= x_1 \times w_{14} + x_2 \times w_{24} + x_3 \times w_{34} + b_4$

$= 5 \times 0.1 + 3 \times 0.6 + 2 \times 0.5 + 0.8$

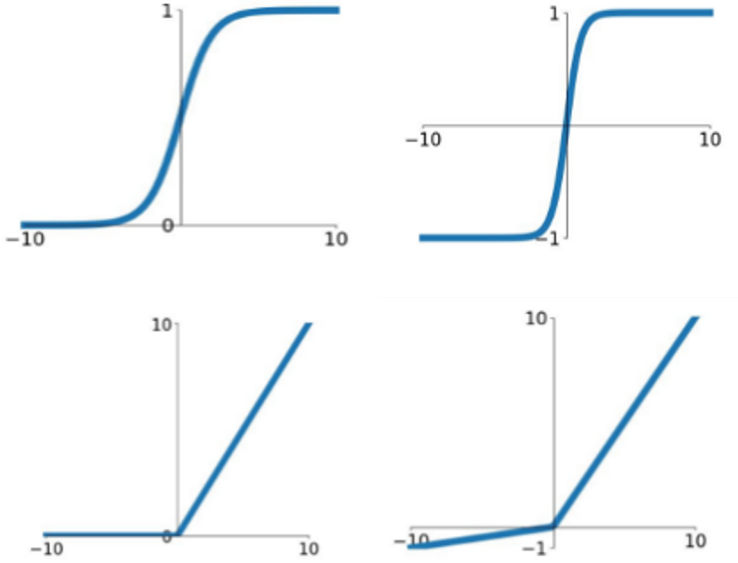
$= 0.5 + 1.8 + 1 + 0.8$

$= 4.1$

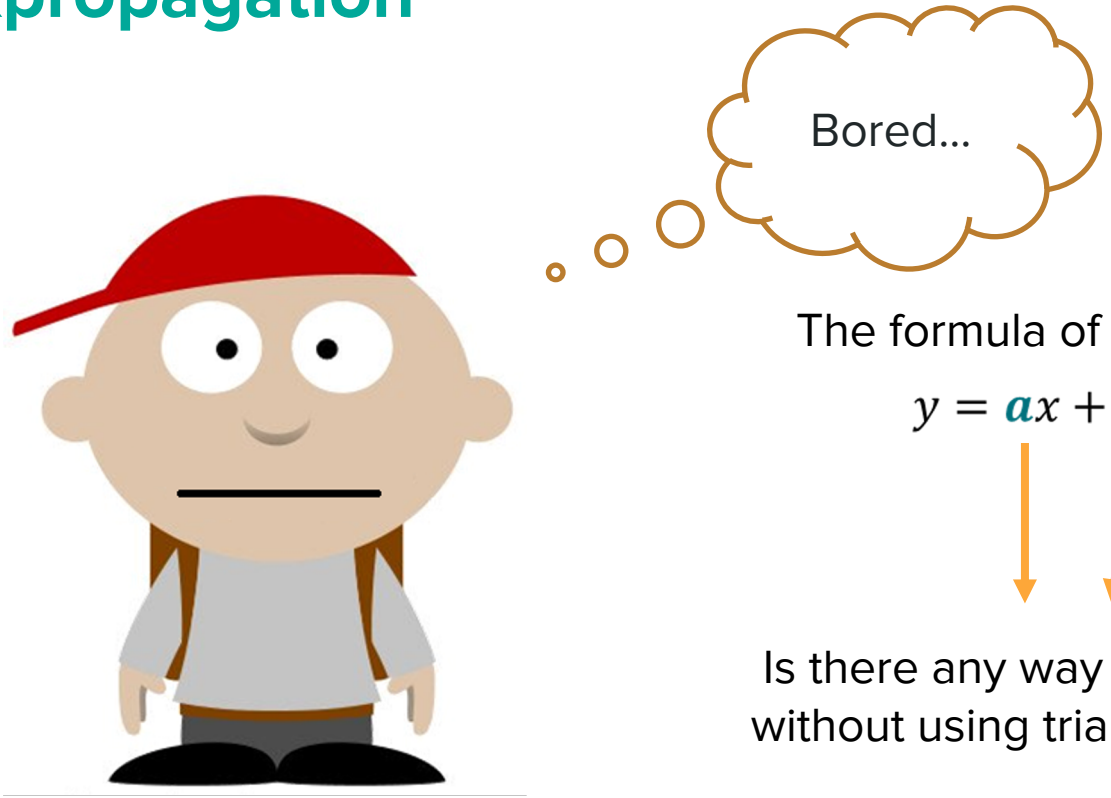
$\hat{y}_4 = \text{sigmoid}(\text{ tmp }) = \frac{1}{1+e^{4.1}} = 0.0163$

Activation Functions

- Common activation functions:
  - Sigmoid
  - Tanh
  - ReLU
  - LeakyReLU



Backpropagation



The formula of the line

$$y = ax + b$$

Is there any way to find a, b without using trial and error?

Backpropagation

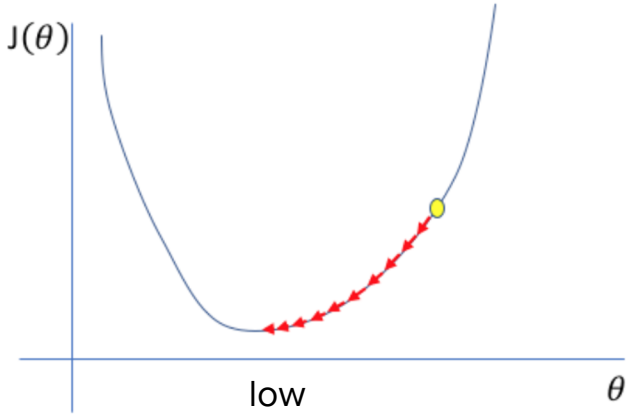
- Backpropagation is the approach of algorithms that helps the neural network to learn/update weights after feed forward steps.
- There is a variety of optimizers/algorithms
  - Gradient descent
  - AdaGrad
  - RMSProp
  - Adam

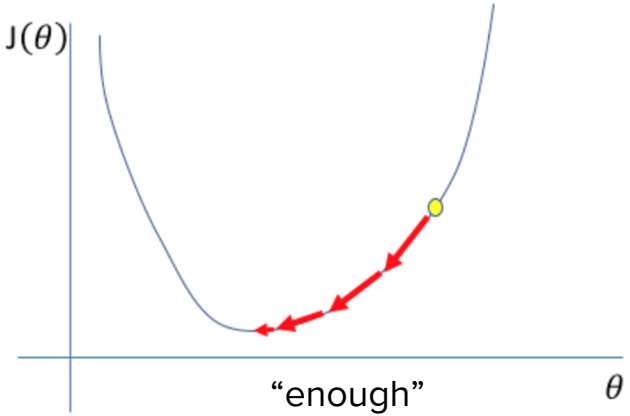
Backpropagation

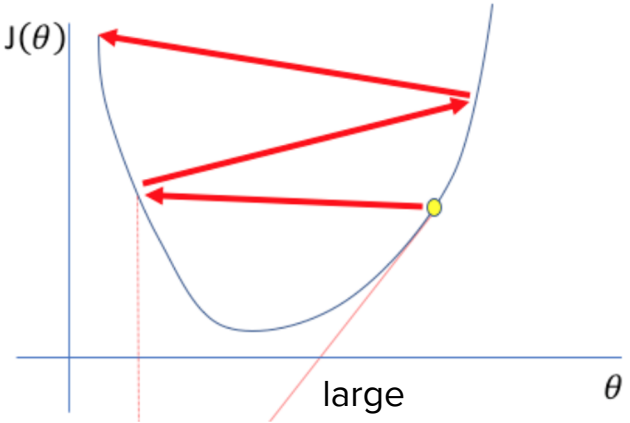
- Gradient-based approach
  - Calculate the gradient of the loss function regarding to weights.
  - Using the data set to update parameters iteratively while the loss function still reduces (not converge).
- Procedure
  - Design a network architecture
  - Iteratively
    - Feed a sample to the network to get output, calculate loss
    - Update weights using gradient of the loss function
    - Stop iteration when loss value stops reducing
  - Save network weights for usage

Hyper parameters

- Number of neurons in hidden layers
  - Low: converge faster ➜ simple problem, simple data
  - Large: converge slower, easier to get overfitting if data set is small ➜ complicated problem, large data set
- Learning rate ➜ “step size” to update weights







Loss Functions

- Mean Square Error

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

- Cross-entropy

$$H(\mathbf{p}, \mathbf{q}) = -\sum_{i=1}^{C} p_i \log q_i$$

Training and Evaluation

- Training phase: iteratively feed samples to the network to update weights
- Evaluation phase:
  - Feed samples to the network to calculate outputs, loss value, accuracy, etc.
  - weights are “frozen” (not being updated).
- Training samples === Evaluation samples???
  - Data set is divided into
    - Training set ➜ update weights to optimize loss function
    - Test set ➜ evaluate quality of the trained network

Training and Evaluation

- Iteratively feed samples to network to update weights

  ➜ how many epochs/steps?

  ➜ when to stop?

- Approach
  - Split a part of training set to be validation set
  - Track loss value in validation set during training phase
  - Terminate training when loss stops reducing

    ➜ Early Stopping technique

Training and Evaluation

- Data set
  - Training set
  - Validation set
  - Test set
- Loss value, accuracy, etc. go with the corresponding subset.

Convolutional Neural Networks

Contents

Contents

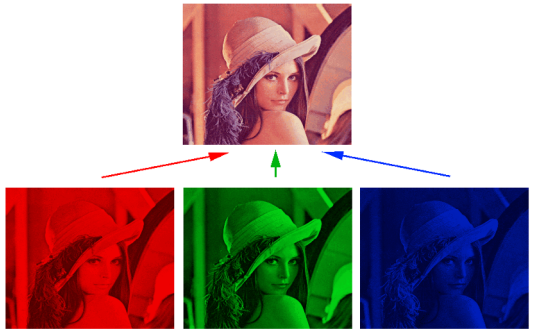
Introduction - Typical image types

Grayscale image

0	120	3	0	6
100	0	43	2	128
2	1	255	0	1
34	0	4	5	7
1	1	56	0	1

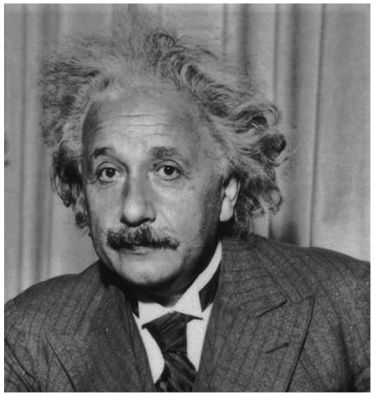
height

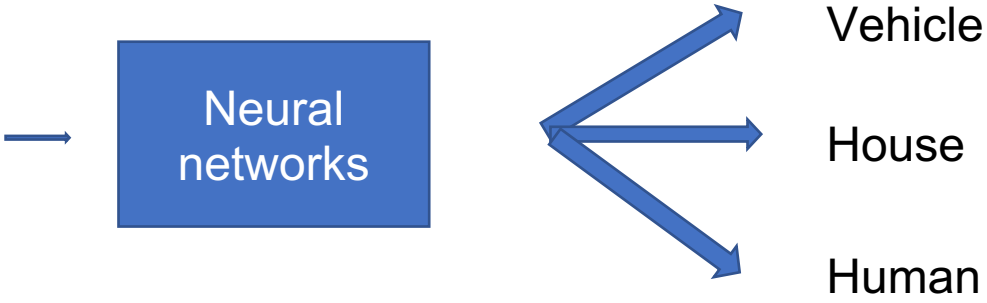
RGB image



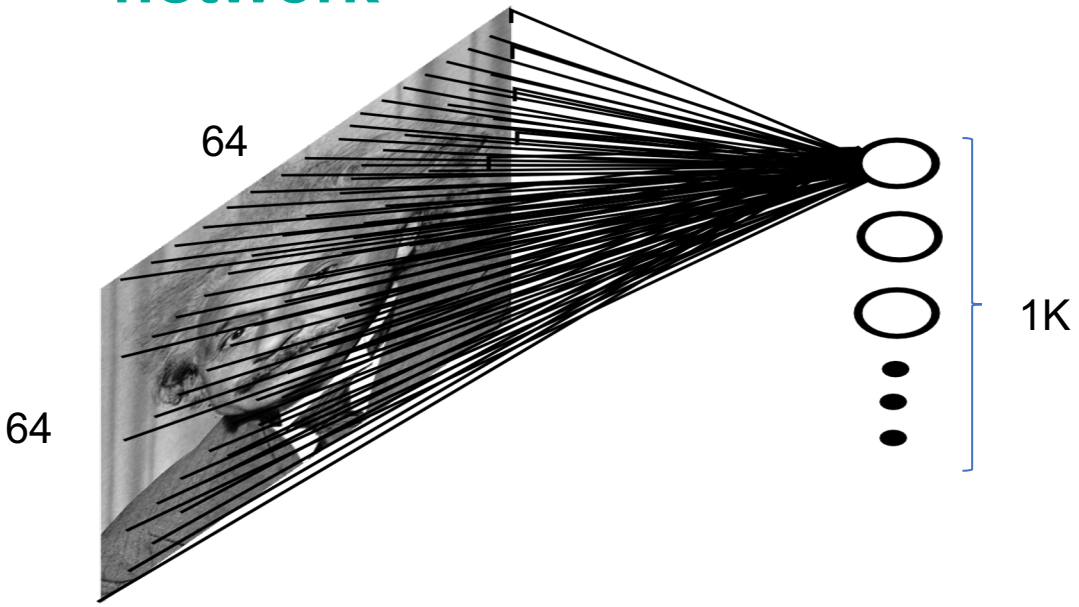
depth

Introduction - image classification





Introduction - Fully connected neural network



Each hidden node connects to all pixels

The no. parameters of this layer :

64*64*3*1000 = 1.2B

Too much parameters!!!

Introduction - Convolution operator in image processing

- Edge detection

$*$

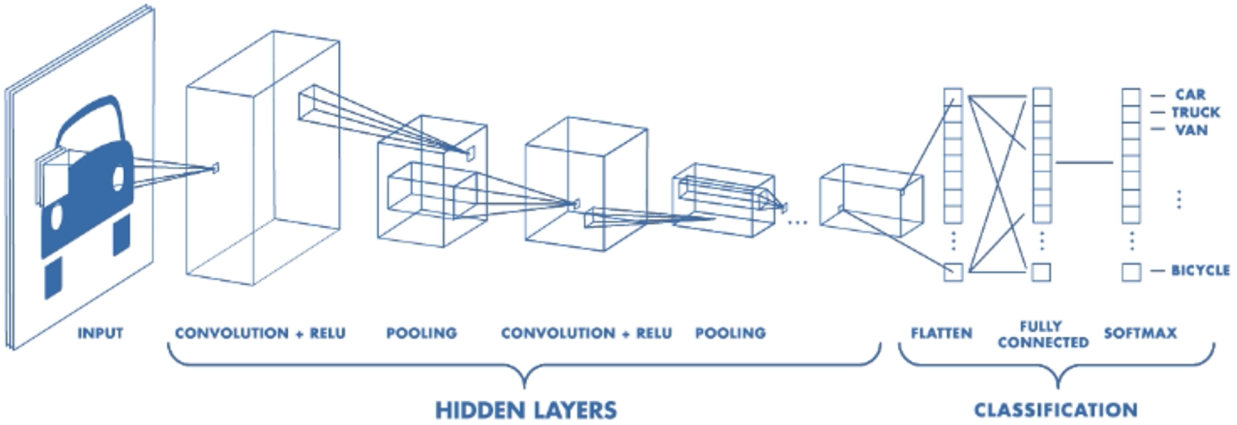
1	0	-1
2	0	-2
1	0	-1

Sobel filter

$=$

Contents

CNN - Typical architecture



CNN - Typical architecture

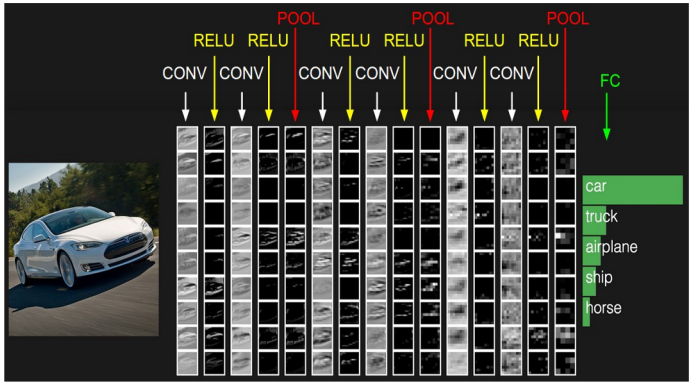
- Convolutional layer
- Activation layer (ReLU)

Repeat k times

- Pooling layer (optional)

Repeat n times

- Fully connected layer



k = 2, n = 3

CNN - Typical architecture

- Convolutional layer
- Activation layer (ReLU)
- Pooling layer (optional)

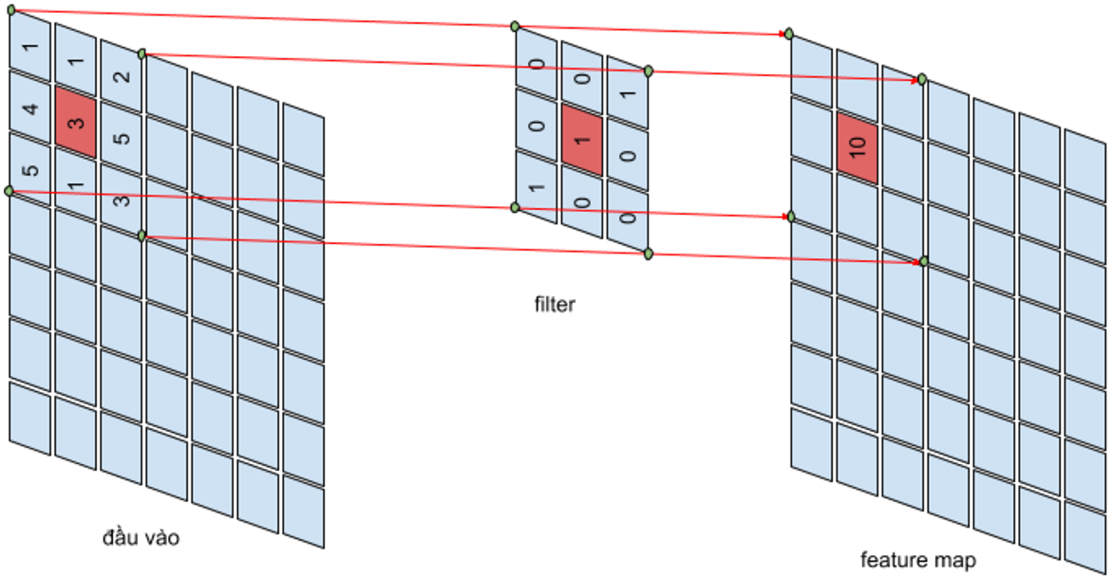
Repeat k times

Repeat n times

- Fully connected layer

Convolutional layer - 2D convolution

- Convolution at a location in the image



Convolutional layer - Stride

- Stride (S) is sliding step size of filter

5x5 image

2	0	3	0	1
1	0	4	2	1
3	1	1	0	1
1	0	4	5	0
0	1	2	0	1

*

3x3 filter

0	1	0
0	1	0
0	1	0

=

3x3 feature map

1	8	2
1	9	7
2	7	5

S = 1

=

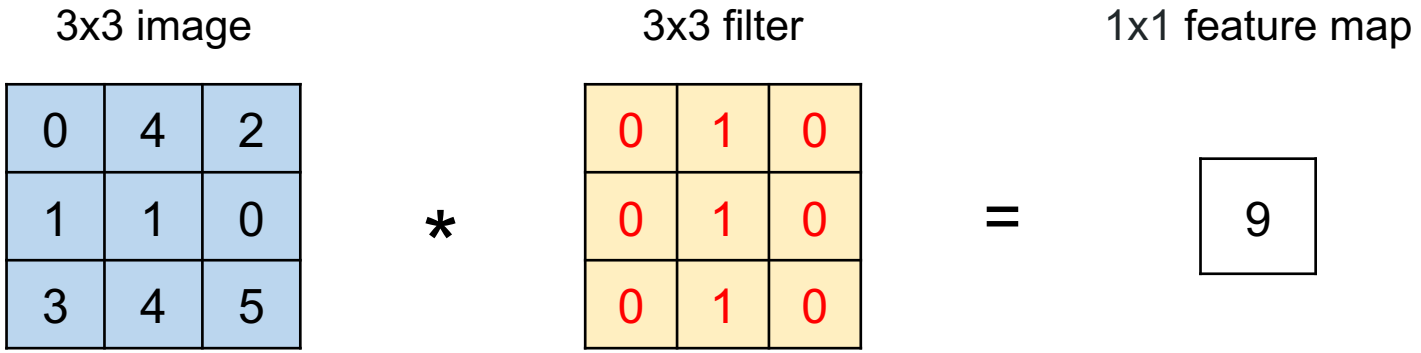
2x2 feature map

1	2
2	5

S = 2

Convolutional layer - Padding

- When using convolution, information at borders is lost ...



… and the image size decreases rapidly → Padding

Convolutional layer - Padding

- Adding zeros - 0 to borders of the image

3x3 image after padding

0	0	0	0	0
0	0	4	2	0
0	1	1	0	0
0	3	4	5	0
0	0	0	0	0

*

3x3 filter

0	1	0
0	1	0
0	1	0

=

3x3 feature map

1	5	2
4	**9**	**7**
4	**5**	**6**

An $W \times H$ image, $F \times F$ filter, stride of S, padding of P
Feature map of $W' \times H'$:

$$\mathrm{W}' = \left\lfloor \frac{\mathrm{W}+2\mathrm{P}-\mathrm{F}}{\mathrm{S}} \right\rfloor + 1, \mathrm{H}' = \left\lfloor \frac{\mathrm{H}+2\mathrm{P}-\mathrm{F}}{\mathrm{S}} \right\rfloor + 1$$

CNN - Typical architecture

- Convolutional layer
- Activation layer (ReLU)
- Pooling layer (optional)

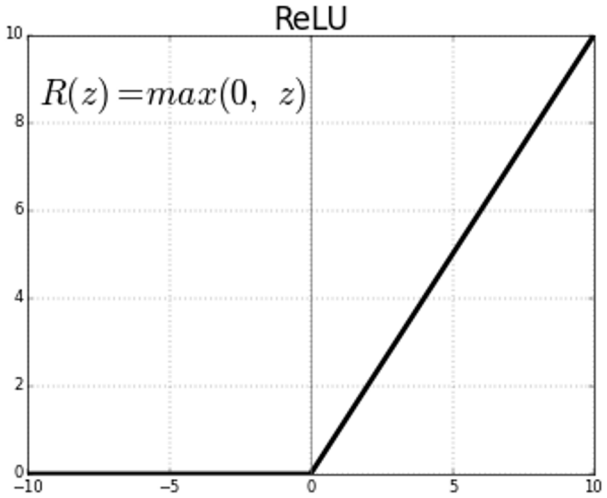
Repeat k times

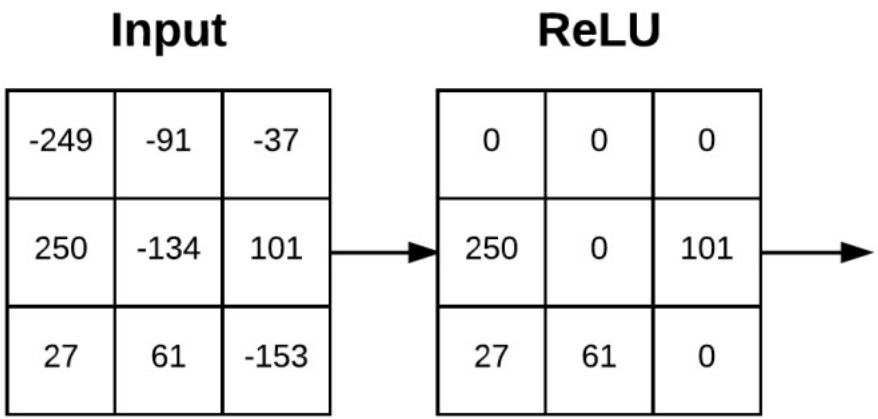
Repeat n times

- Fully connected layer

Activation layer - ReLU function

- In CNN, ReLU layer often goes after the convolutional layer.
- Beside ReLU, we can use other activation layers, such as sigmoid or tanh.





CNN - Typical architecture

- Convolutional layer
- Activation layer (ReLU)

Repeat k times

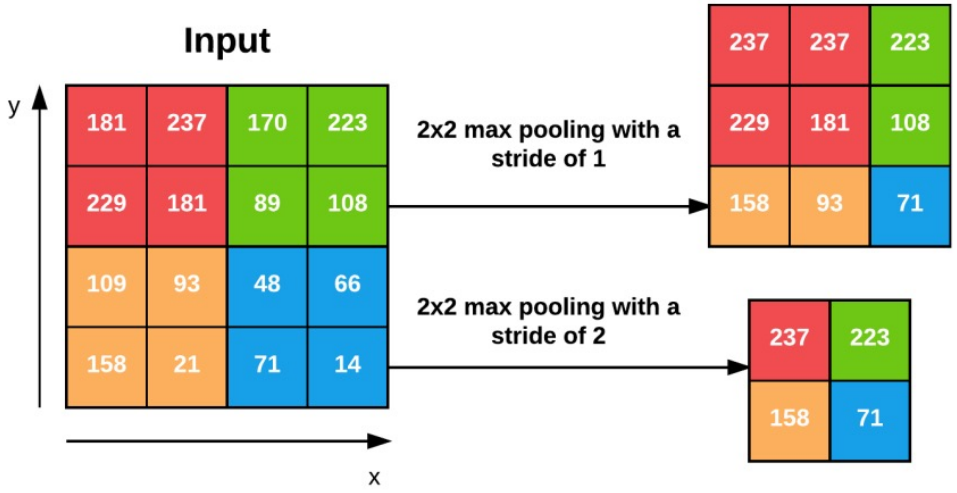
- Pooling layer (optional)

Repeat n times

- Fully connected layer

Pooling layer - Max pooling, average pooling

- Stride is often as the same as filter size.
- Filter is applied independently along the depth dimension.
- To reduce feature map size, the absolute position of feature is no longer important.



Input After pooling

CNN - Typical architecture

- Convolutional layer
- Activation layer (ReLU)
- Pooling layer (optional)

Repeat k times

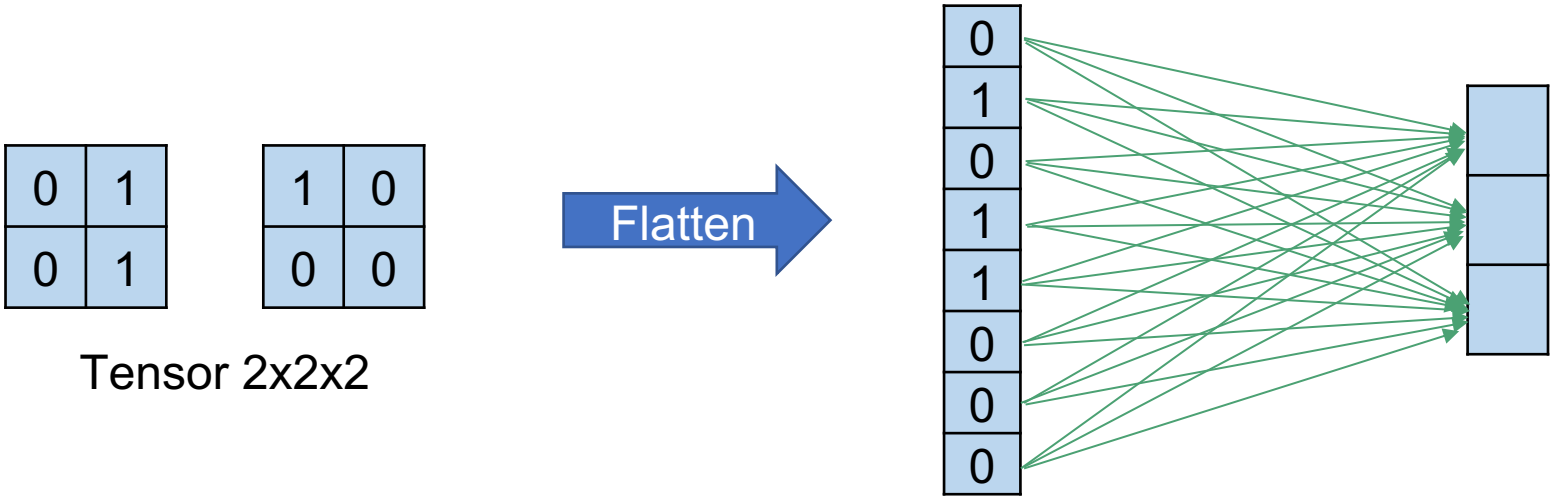
Repeat n times

- Fully connected layer

Fully Connected layer - Flattening

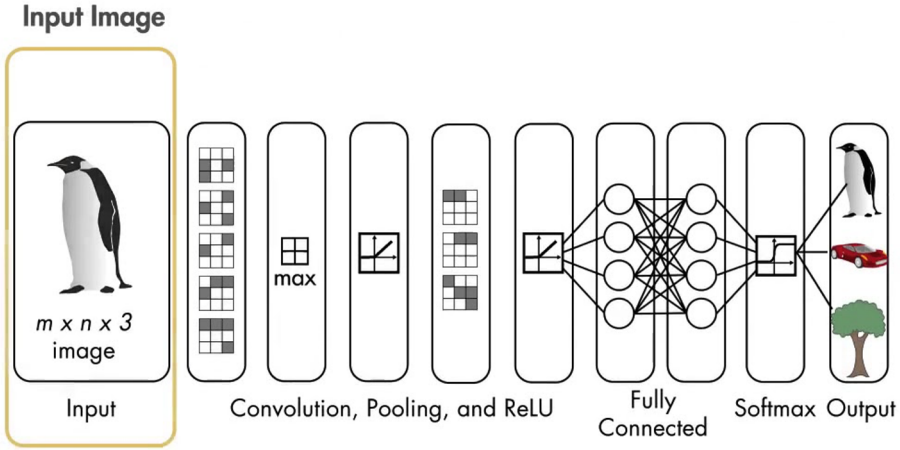
- Flattening: transform 3D tensor to 1D tensor to make inputs for fully connected layer.

  For example:



Fully Connected layer - Exercise

- Design the CNN structure for 3-category classification problem with and image input of 78x78x3

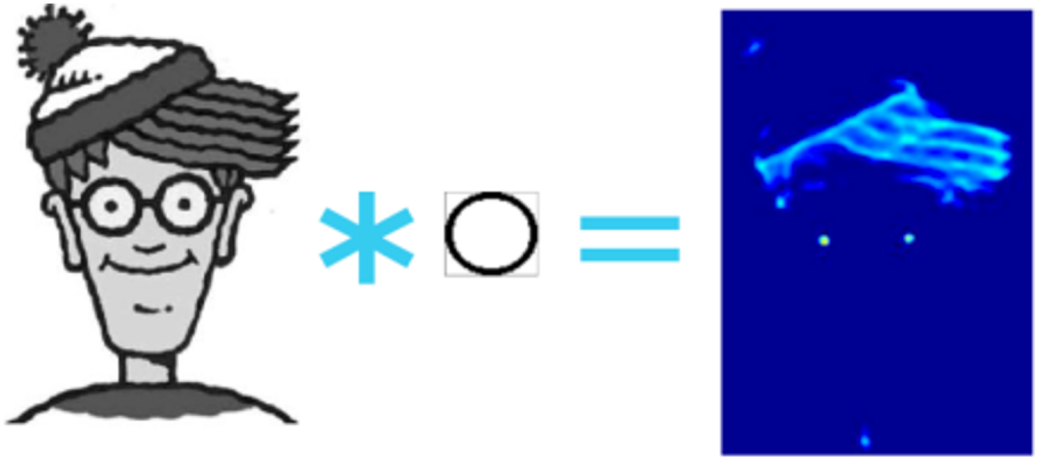


Popular
Vision architectures

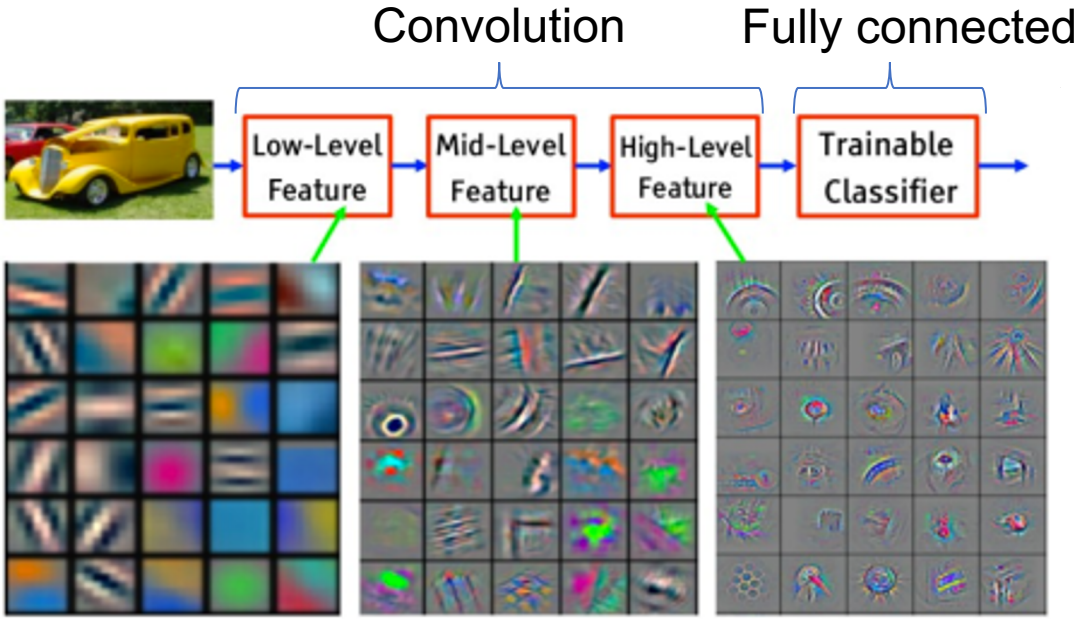
Contents

1. Review CNN
2. Popular Vision architectures
   a) The evolution of CNN
   b) LeNet-5
   c) AlexNet
   d) VGGNet
   e) ResNet
3. Transfer Learning

1 Review CNN

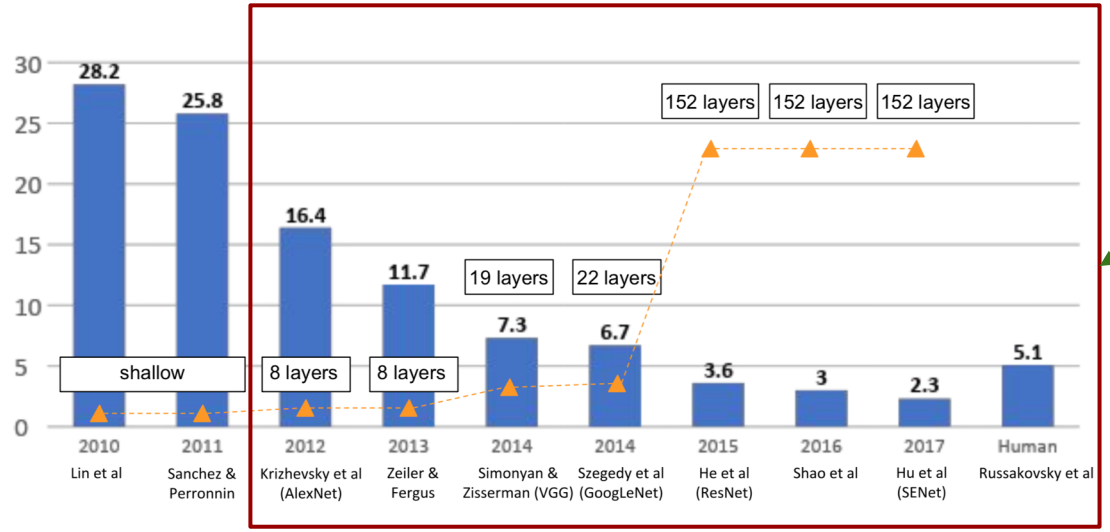


1 Review CNN



Learning Hierarchical Representations

2 The evolution of CNN



Deep CNN

CNN milestones

LeNet (1993)

IMAGENET

ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

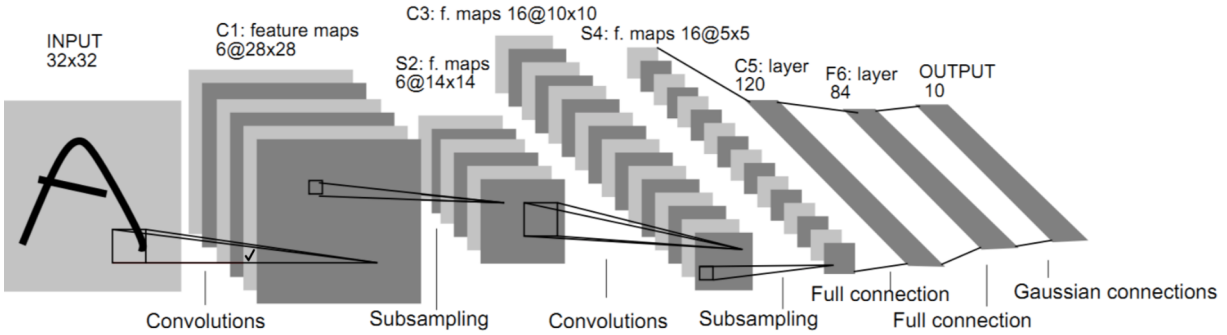
> 14M images
> 20K classes

The annual competition from 2010:
+ Classification
+ Detection

2 LeNet-5

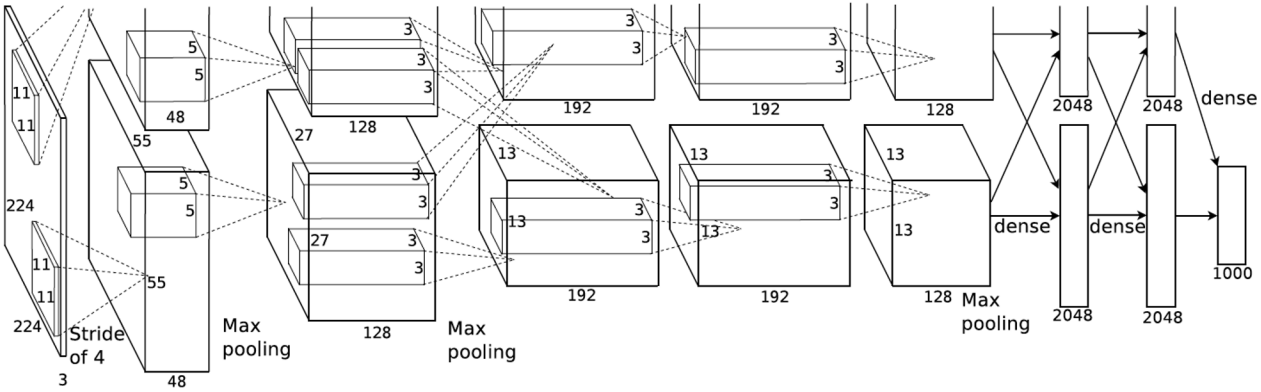
The first and standard convolution neural network:

- Convolutions
- Subsampling (average pooling)
- Sigmoid or tanh nonlinearity
- Fully-connected outputs



LeCun, Yann, et al. *"Gradient-based learning applied to document recognition."* Proceedings of the IEEE, 1998.

2 AlexNet – ILSVRC 2012 Winner



Similar network to LeNet but:

- Max pooling, ReLU nonlinearity.
- More data and bigger model.
- GPU implementation (50x faster)
- Dropout regularization

Alex, Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. *"Imagenet classification with deep convolutional networks."* NIPS 2012

2 VGGNet – ILSVRC 2014 1st Runner Up

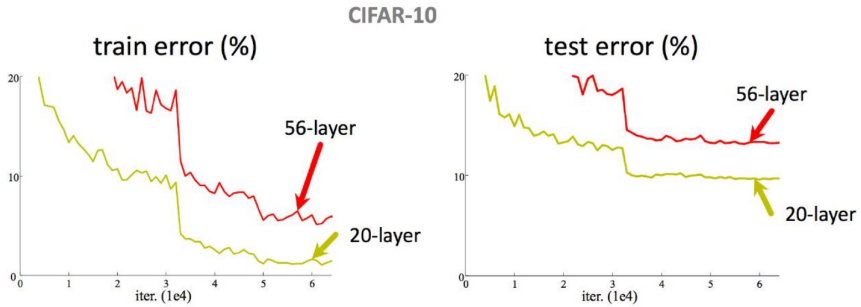
- Sequence of deeper networks trained progressively
- Large receptive fields replaced by successive layers of 3x3 convs (with ReLU between)
  - Reduce no. params (3 filter 3x3 vs 1 filter 7x7)
- Modularized design - Can easily plug a new module:
  VGG11 -> VGG13 -> VGG16 -> VGG19
- The large no. parameters: VGG16 - 138M

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

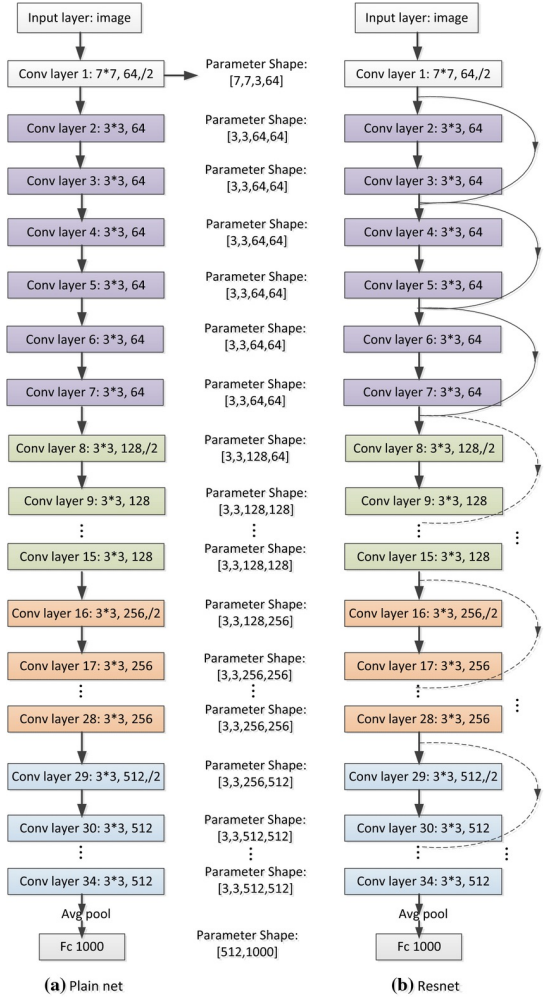
3 Resnet



- Plain nets: stacking of 3x3 conv layers
- 56-layers got higher training and test error

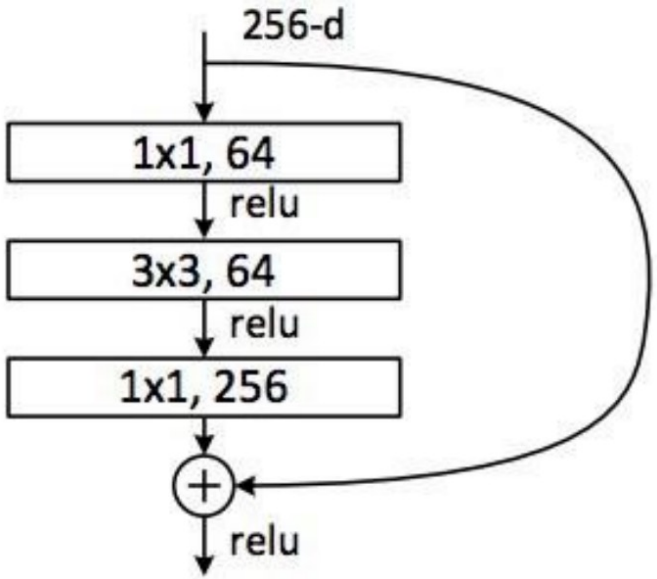
Problem of going deeper?

- Vanishing Gradient
  - Most neurons will die during back propagating the weights.



Szegedy, Christian, et al. *"Rethinking the inception architecture for computer vision."* CVPR 2016.

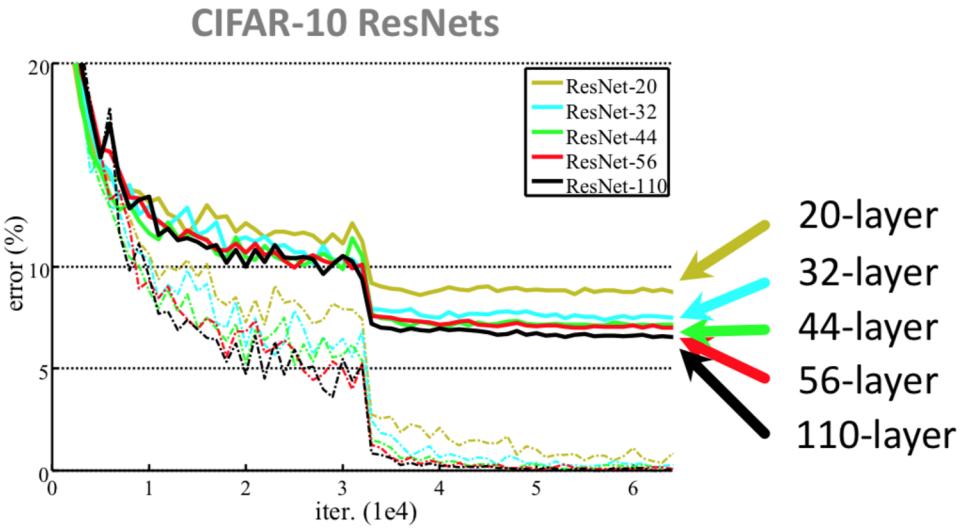
3 Resnet



- # parameters
  - 256 x 64 + 64 x 3 x 3x 64 + 64 x 256 = ~70K
- # parameters just using 3 x 3 x 256 x 256 conv layer = ~600K

Szegedy, Christian, et al. *"Rethinking the inception architecture for computer vision."* CVPR 2016.

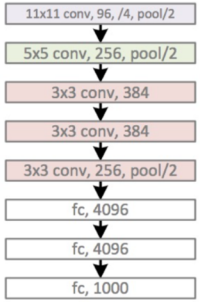
2 ResNet - ILSVRC 2015 Winner



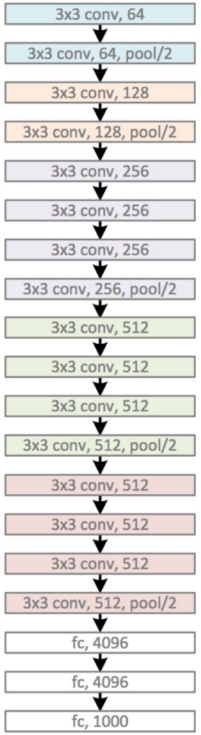
He, Kaiming, et al. *"Deep residual learning for image recognition."* CVPR 2016.

2 Networks so far ...

AlexNet, 8 layers
(ILSVRC 2012)



VGG, 19 layers
(ILSVRC 2014)



GoogleNet, 22 layers
(ILSVRC 2014)

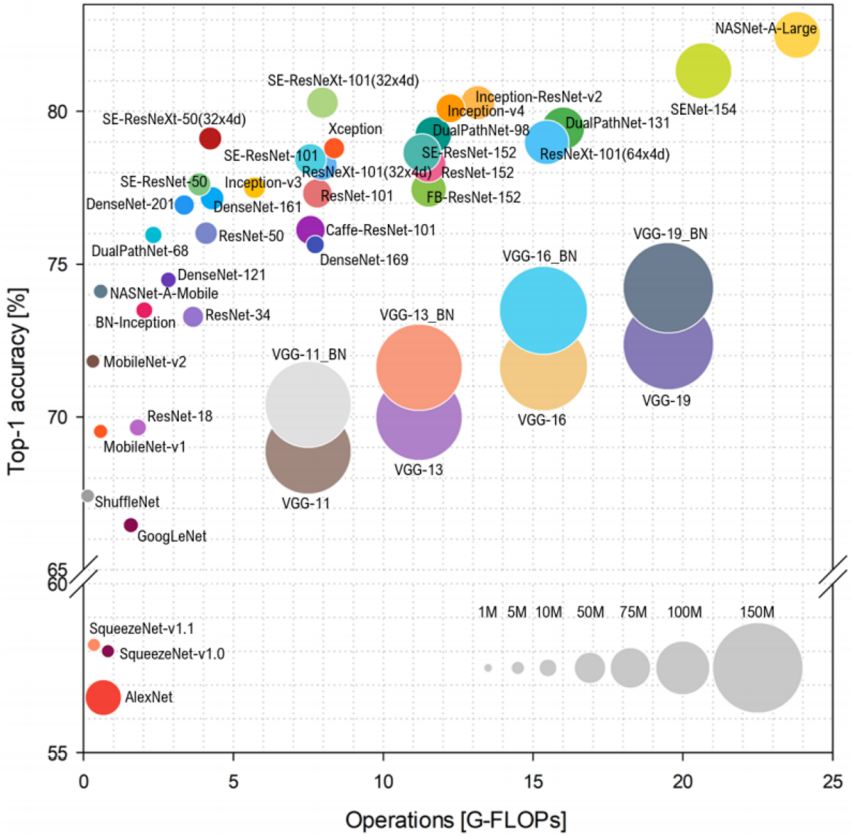
2 Networks so far ...

AlexNet, 8 layers
(ILSVRC 2012)

VGG, 19 layers
(ILSVRC 2014)

ResNet, 152 layers
(ILSVRC 2015)

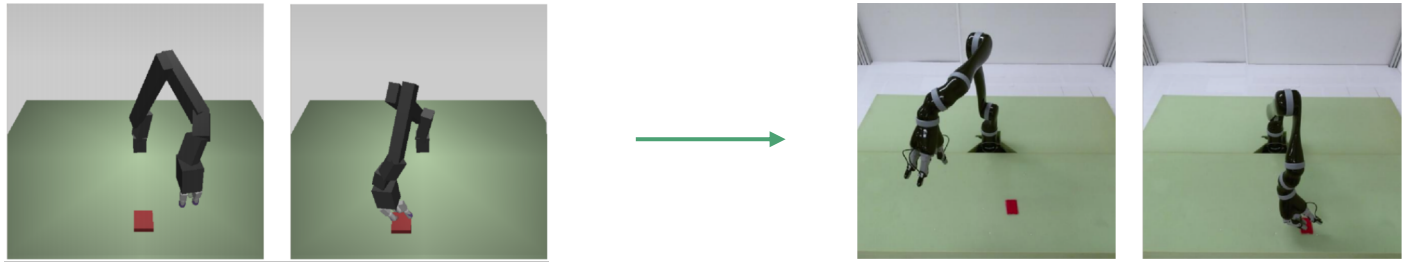
2 CNN Summarization



Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
ResNeXt50	96 MB	0.777	0.938	25,097,128	-
ResNeXt101	170 MB	0.787	0.943	44,315,560	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	0.825	0.960	88,949,818	-

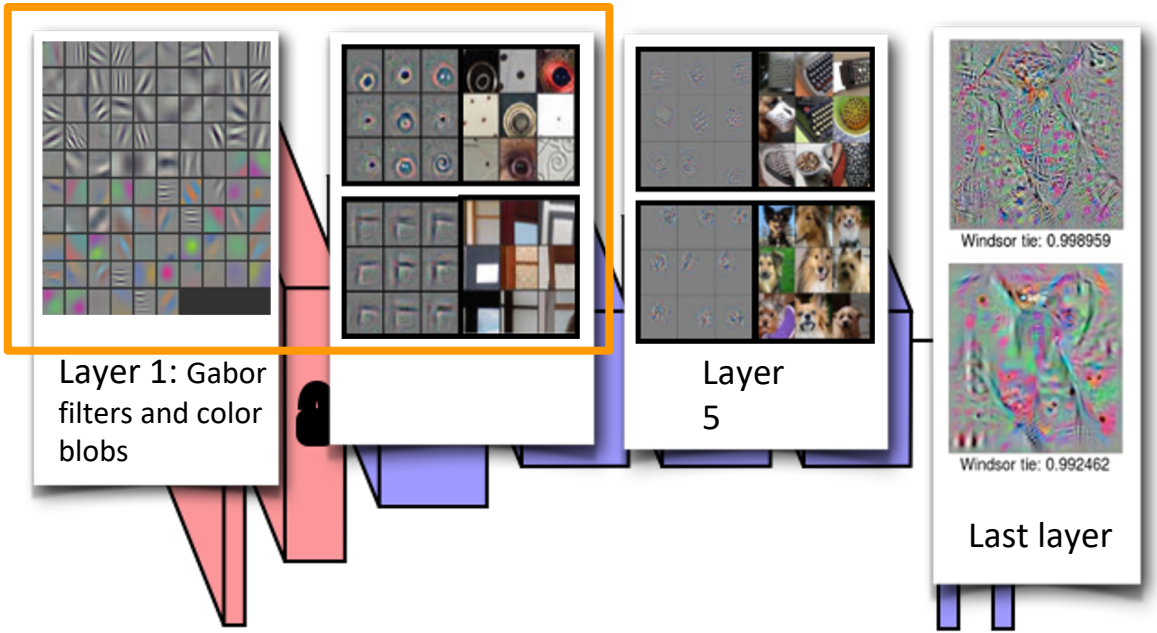
3 Transfer Learning

In practice, engineers never design the network manually and train it from scratch on a new dataset. They often use popular pretrained models that was trained on a huge dataset and partially train the network on the target dataset. This method is called **Transfer Learning**



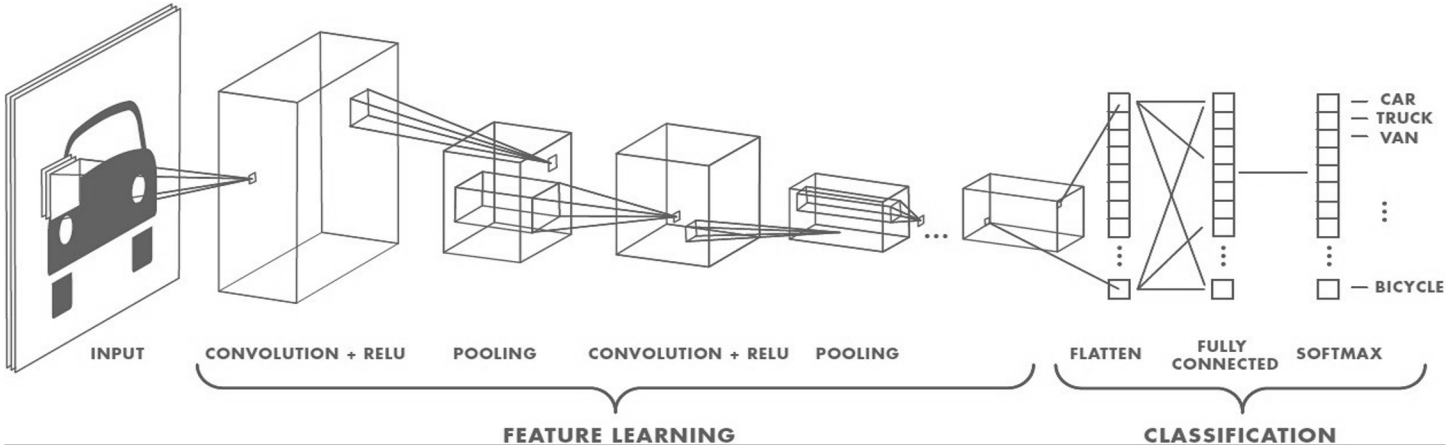
3 Transfer Learning

Representation features of the model learned on the large dataset is often universal, especially in the first layers



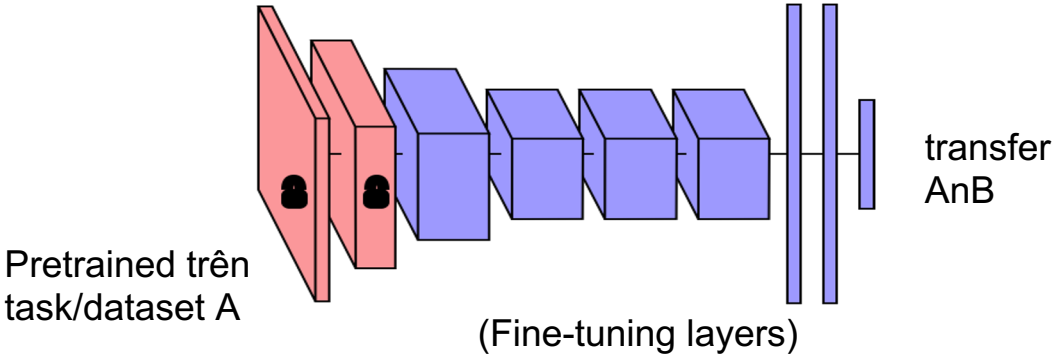
3 Transfer learning techniques

- **Keep the feature learning:** Freeze all parameters of feature learning in the pretrained model, train the classification part only.



3 Transfer learning techniques

- **Train partially the feature learning:** Train/ finetuning a part of the pretrained model. On the big dataset, this technique helps learning features of the target dataset.



3 Transfer learning techniques

There are several factors affecting the choosing retrained layers: **The similarity between two tasks** (dataset domain, …), **the dataset size**, …

Hypothesis: When retraining a layer, if the performance drops on the pretrained dataset, the layer is specific for that dataset. Otherwise, the features is general, we don't need to retrained that layer.

THANK YOU