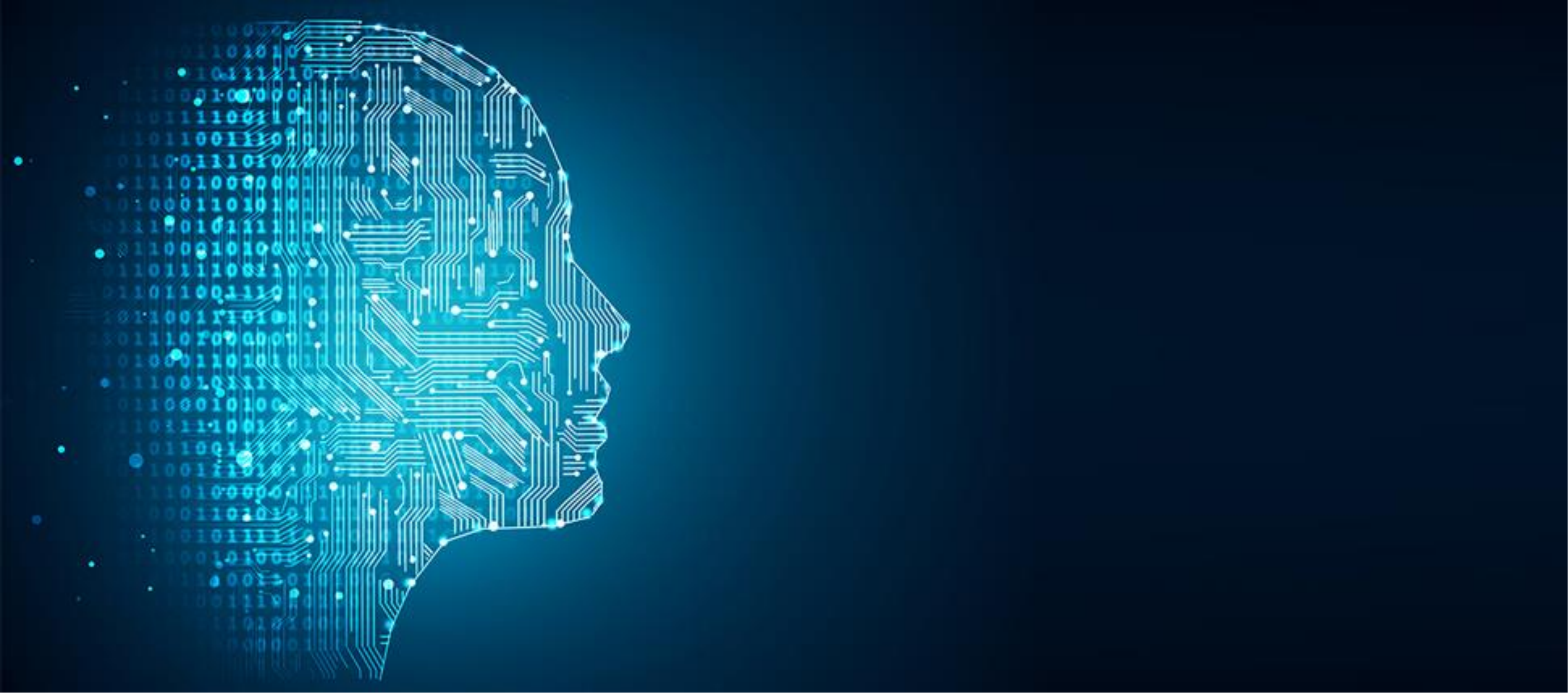


COMP SCI 1400
AI Technologies - Deep Learning Basics

Dr Kamal Mammadov

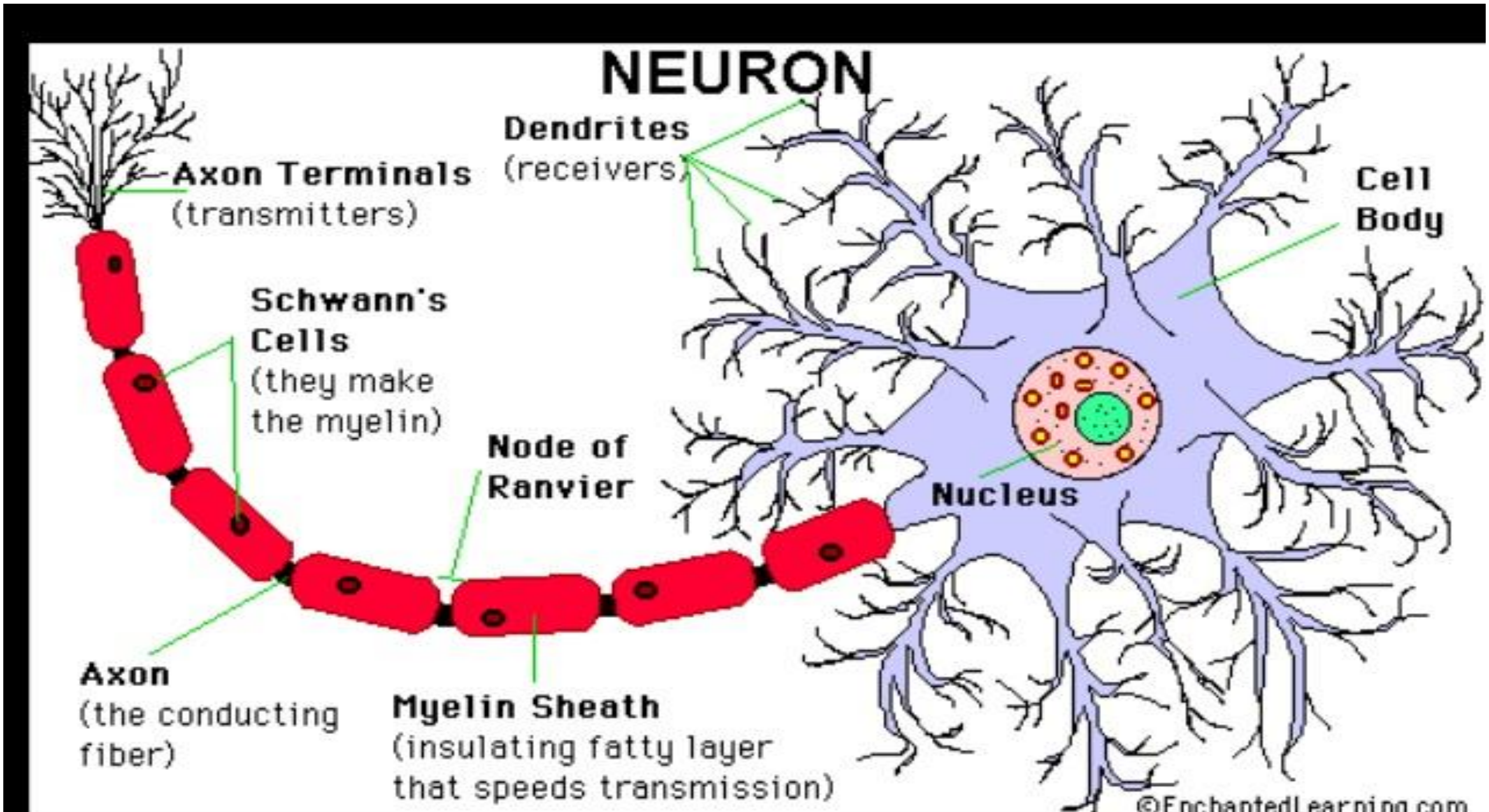
What can we do with AI techniques?

Deep Learning

Perceptron Learning Algorithm

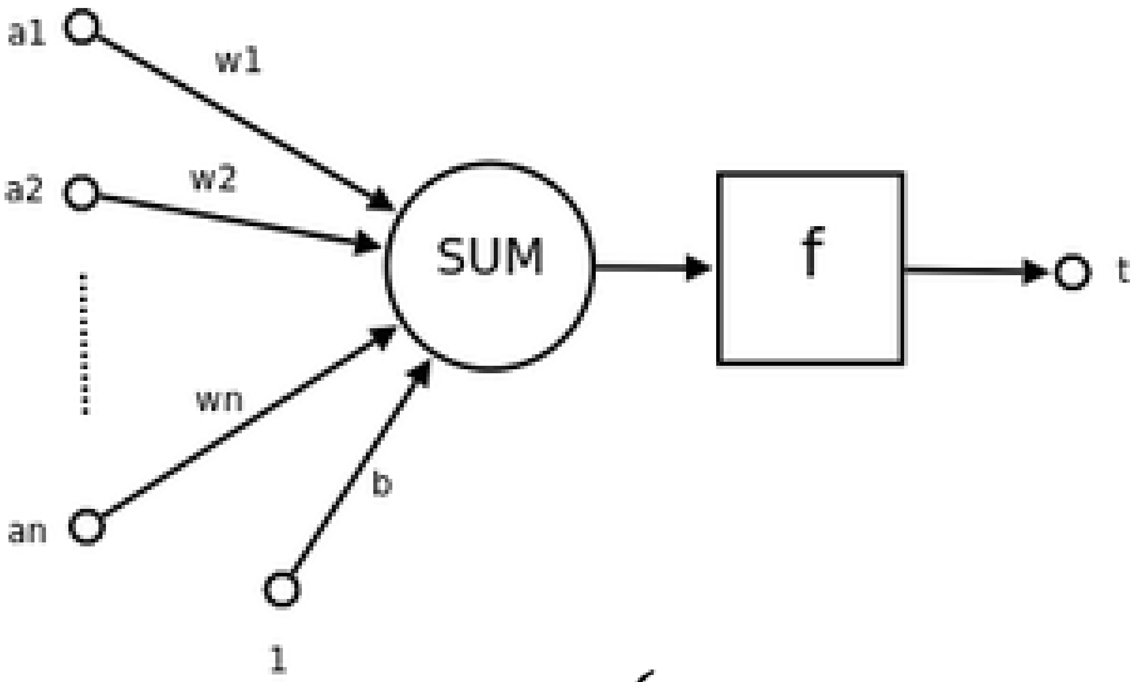
- Perceptron

Perceptron is a type of linear classifier



- Perceptron

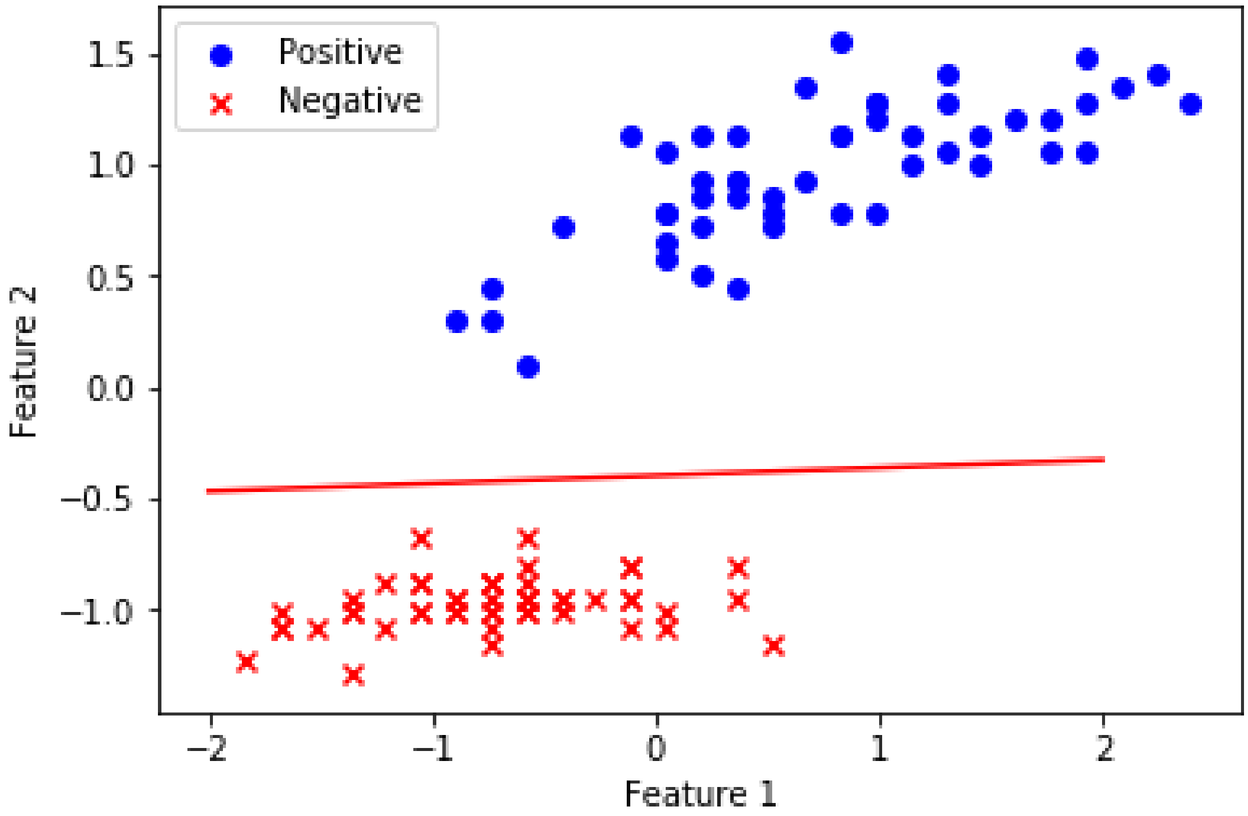
f is activation function



$$scores = \sum_{i}^{N} w_i x_i + b$$

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0, \\ 0 & \text{otherwise} \end{cases}$$

- Perceptron Learning Algorithm --- example



- ## Perceptron Learning Algorithm --- optimization

Step 0: randomly initialize weights

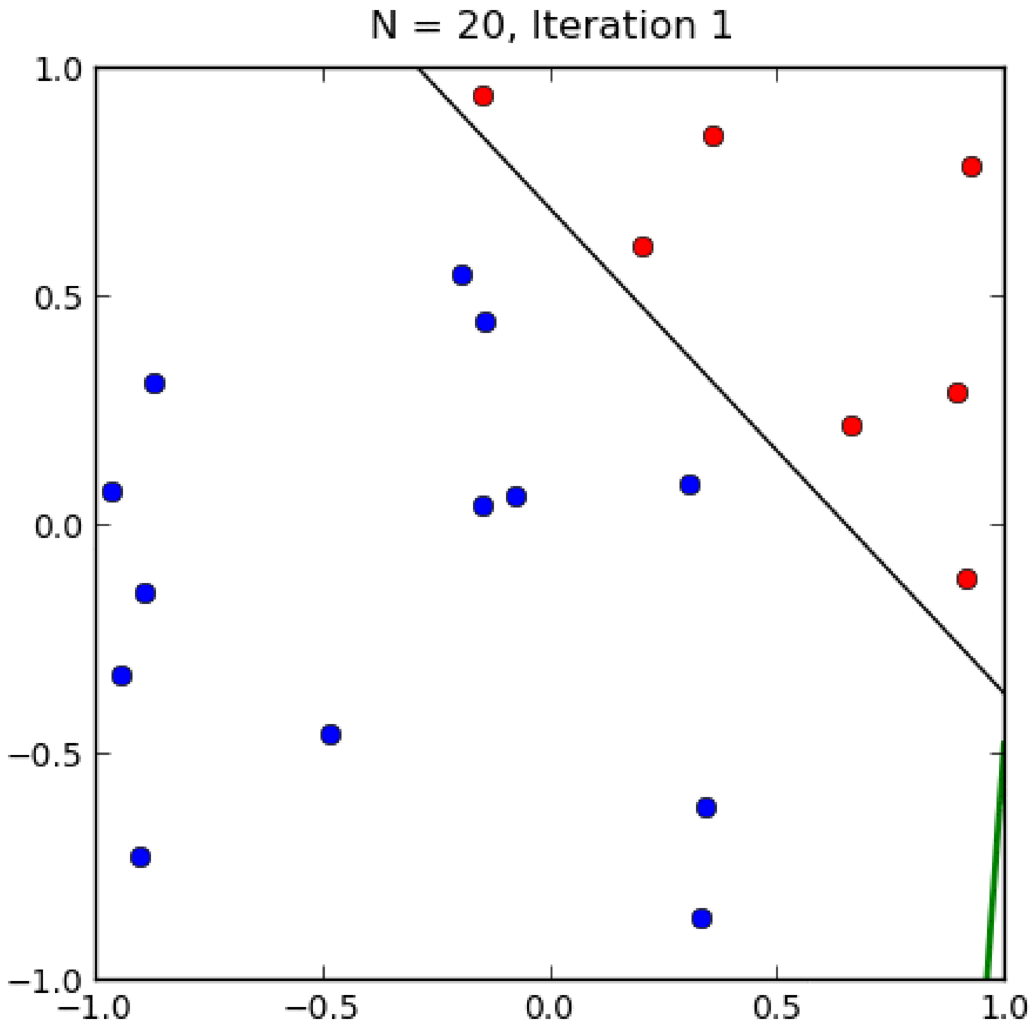
Step 1: calculate the actual output

$$\hat{y}_j^t = f(w_0^t x_{j,0} + w_1^t x_{j,1} + \cdots + w_n^t x_{j,n} + b)$$

Step 2: update the weights

$$w_i^{t+1} = w_i^t + r\left(y_j - \hat{y}_j^t\right)x_{j,i}$$

- Perceptron Learning Algorithm



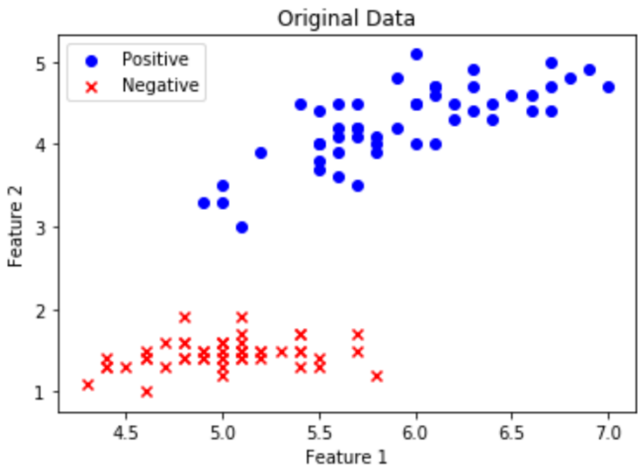
Programming Example

Input data

In [1]:
```
import numpy as np
import pandas as pd

data = pd.read_csv('./data1.csv', header=None)
# input samples, dim (100, 2)
X = data.iloc[:,:2].values
# output samples, dim (100, )
y = data.iloc[:,2].values
```

Data visualization

In [3]:
```
import matplotlib.pyplot as plt

plt.scatter(X[:50, 0], X[:50, 1], color='blue', marker='o', label='Positive')
plt.scatter(X[50:, 0], X[50:, 1], color='red', marker='x', label='Negative')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend(loc = 'upper left')
plt.title('Original Data')
plt.show()
```



PLA algorithm

Feature normalization

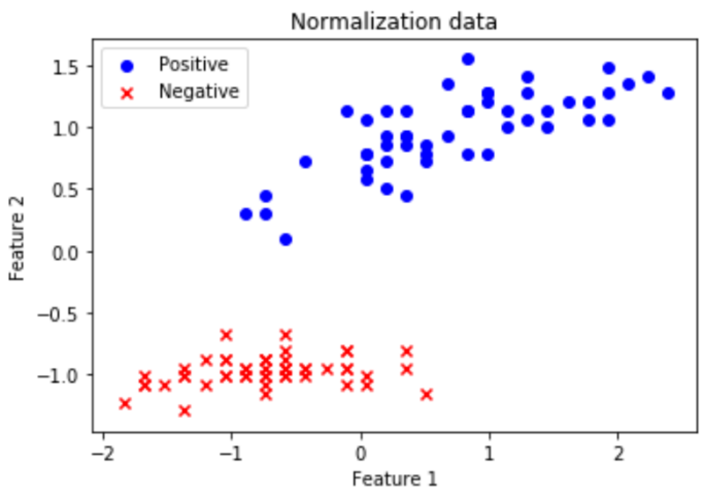
First, normalize the two features separately

$$X = \frac{X - \mu}{\sigma}$$

Among them, μ is the feature mean, and σ is the feature standard deviation.

In [4]:

```
# Mean
u = np.mean(X, axis=0)
# standard deviation
v = np.std(X, axis=0)

X = (X - u) / v

# draw
plt.scatter(X[:50, 0], X[:50, 1], color='blue', marker='o', label='Positive')
plt.scatter(X[50:, 0], X[50:, 1], color='red', marker='x', label='Negative')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend(loc = 'upper left')
plt.title('Normalization data')
plt.show()
```

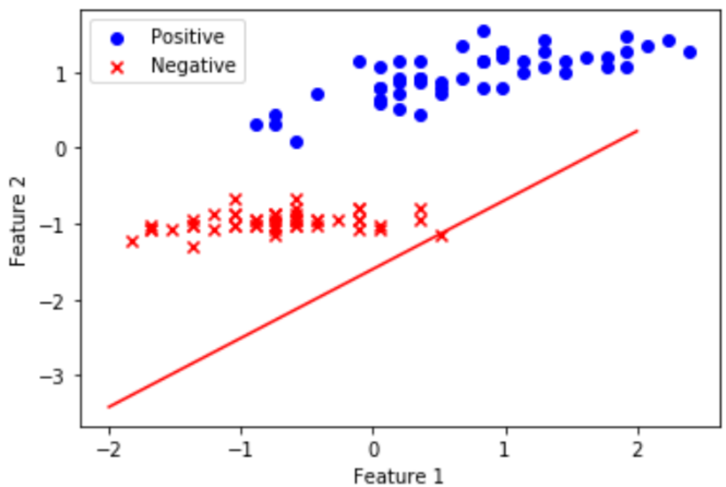


Classification Boundary init

In [5]:
```
# X + offset
X = np.hstack((np.ones((X.shape[0],1)), X))
# weight init
w = np.random.randn(3,1)
```

Display initial line position:

In [6]:
```
# First coordinate (x1, y1)
x1 = -2
y1 = -1 / w[2] * (w[0] * 1 + w[1] * x1)
# Second coordinate (x2, y2)
x2 = 2
y2 = -1 / w[2] * (w[0] * 1 + w[1] * x2)
# draw
plt.scatter(X[:50, 1], X[:50, 2], color='blue', marker='o', label='Positive')
plt.scatter(X[50:, 1], X[50:, 2], color='red', marker='x', label='Negative')
plt.plot([x1,x2], [y1,y2],'r')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend(loc = 'upper left')
plt.show()
```



Calculate scores, update weights

In [7]:
```
s = np.dot(X, w)
y_pred = np.ones_like(y)    # predict the output
loc_n = np.where(s < 0)[0]
y_pred[loc_n] = -1
```

Next, select one of the misclassified samples and use PLA to update the weight coefficient w.

In [8]:
```
# The first error point
t = np.where(y != y_pred)[0][0]
# update weights w
w += y[t] * X[t, :].reshape((3,1))
```

Iterative update training

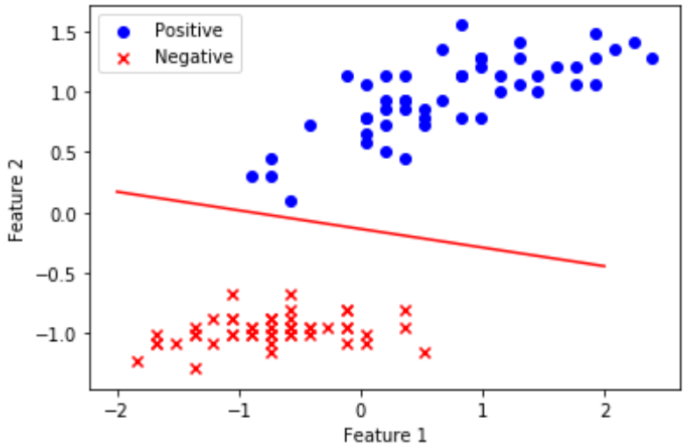
Updating the weight w is an iterative process. As long as there are misclassified samples, it will continue to update until all samples are classified correctly. (Note that the premise is that the positive and negative samples are completely separable)

In [9]:
```
for i in range(100):
    s = np.dot(X, w)
    y_pred = np.ones_like(y)
    loc_n = np.where(s < 0)[0]
    y_pred[loc_n] = -1
    num_fault = len(np.where(y != y_pred)[0])
    print('Update time %2d, error points: %2d' % (i, num_fault))
    if num_fault == 0:
        break
    else:
        t = np.where(y != y_pred)[0][0]
        w += y[t] * X[t, :].reshape((3,1))
```

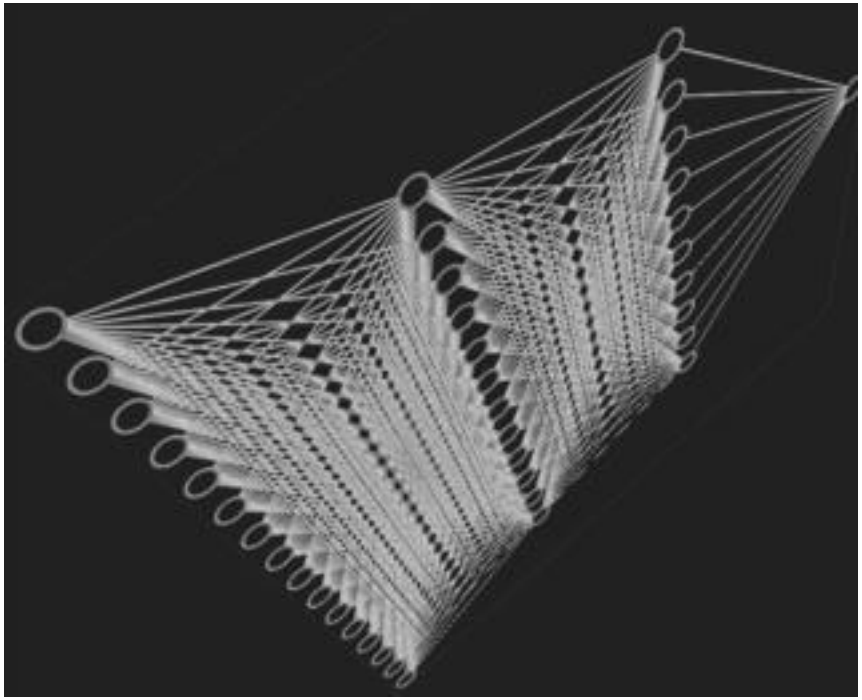
```
Update time  0, error points: 11
Update time  1, error points:  0
```

Draw result

In [10]:
```
# First coordinate (x1, y1)
x1 = -2
y1 = -1 / w[2] * (w[0] * 1 + w[1] * x1)
# Second coordinate (x2, y2)
x2 = 2
y2 = -1 / w[2] * (w[0] * 1 + w[1] * x2)
# draw
plt.scatter(X[:50, 1], X[:50, 2], color='blue', marker='o', label='Positive')
plt.scatter(X[50:, 1], X[50:, 2], color='red', marker='x', label='Negative')
plt.plot([x1,x2], [y1,y2],'r')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend(loc = 'upper left')
plt.show()
```

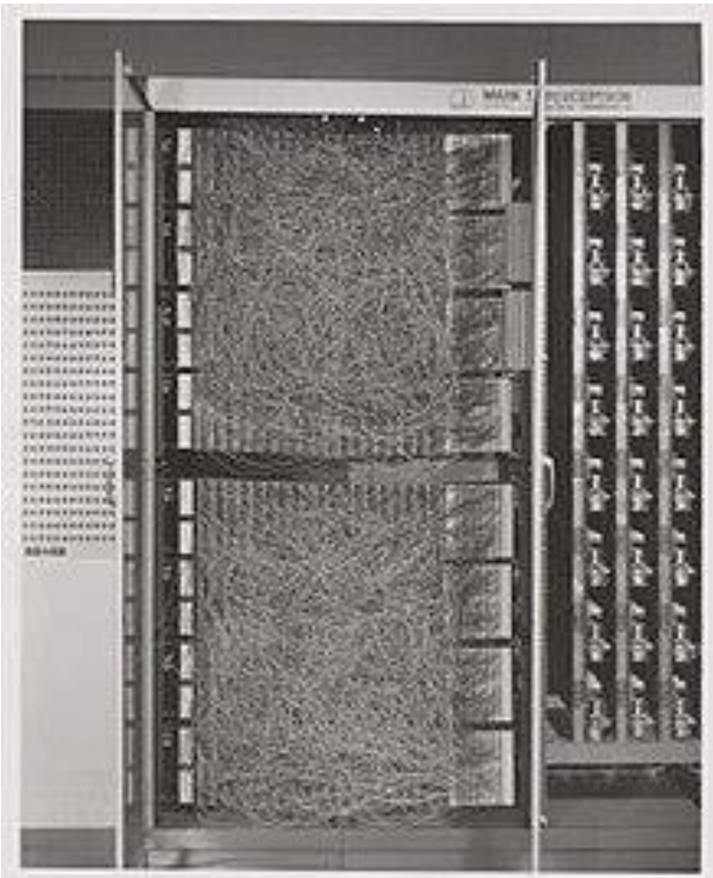


Perceptron – Multi-layer



https://en.wikipedia.org/wiki/Perceptron

Perceptron – Multi-layer



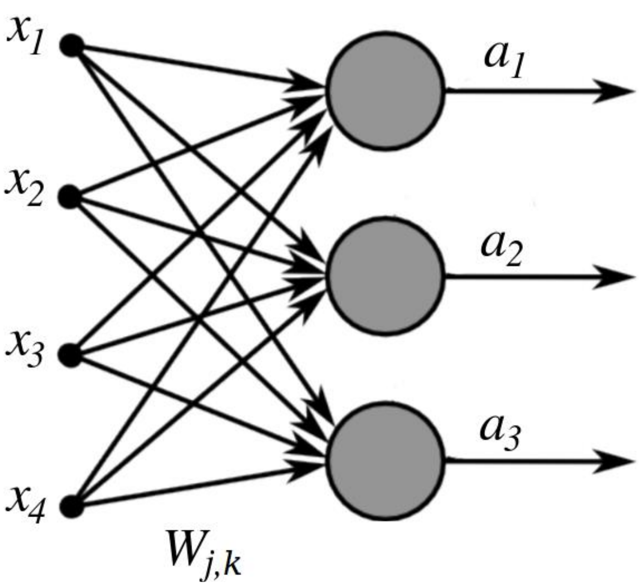
The perceptron algorithm was invented in 1958 at the [Cornell Aeronautical Laboratory] by [Frank Rosenblatt]

The perceptron **was intended to be a machine**, rather than a program, and while its first implementation was in software for the IBM 704.

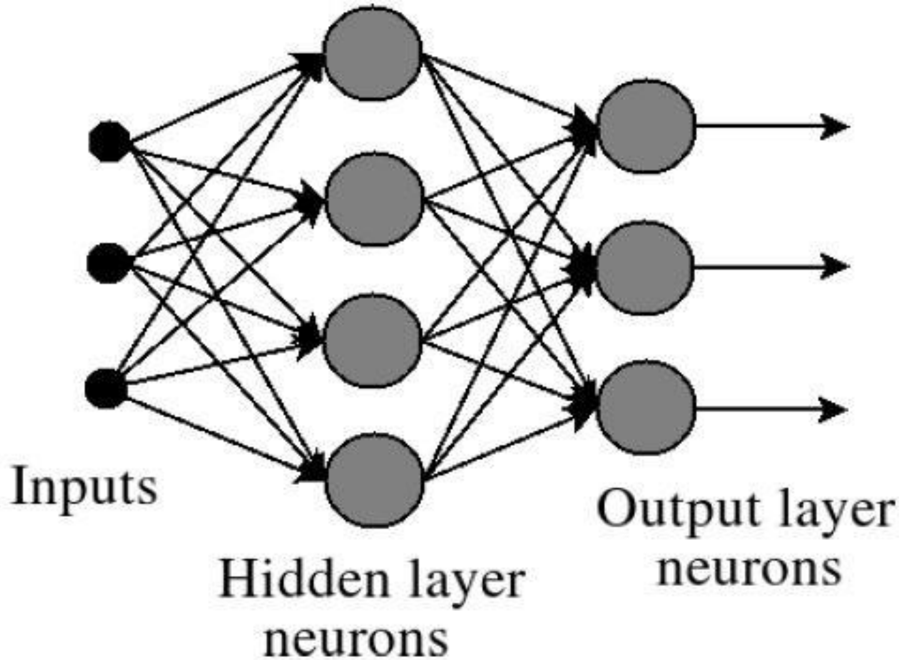
This machine was designed for **image recognition**: it had an array of 400 photocells, randomly connected to the "neurons".

kipedia.org/wiki/Perceptron

Multi-layer Perceptron (MLP) – Linear/Non-Linear



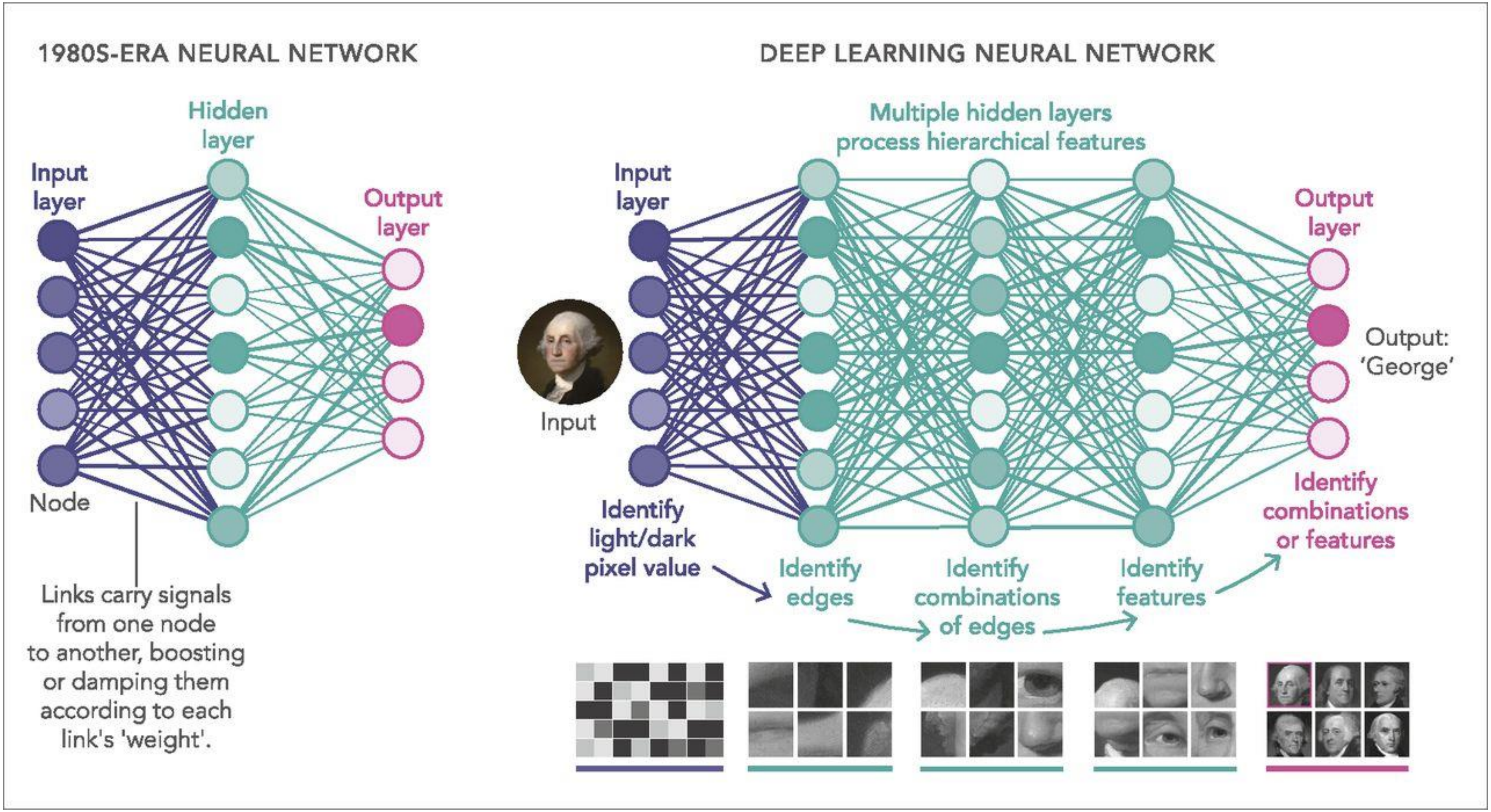
Perceptrons



MLP

MLPs are more expressive than Perceptrons since they can learn highly non-linear class boundaries.

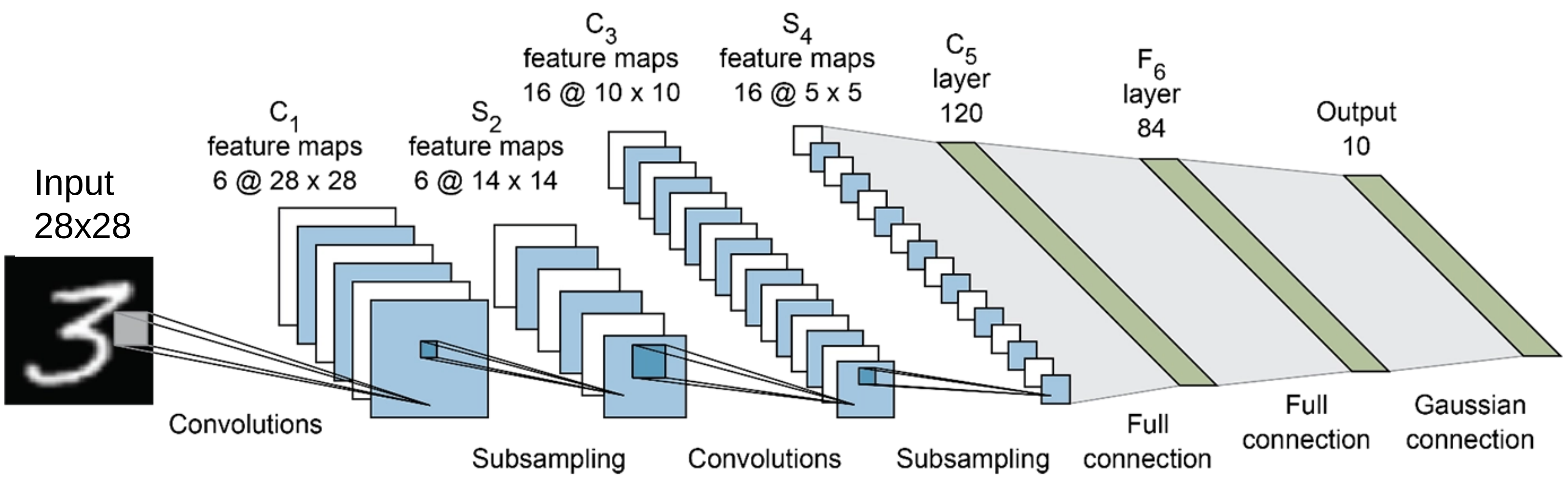
MLP vs Deep Neural Network



MLP is the most basic deep neural network

Convolutional neural network

Building Blocks of Deep CNNs

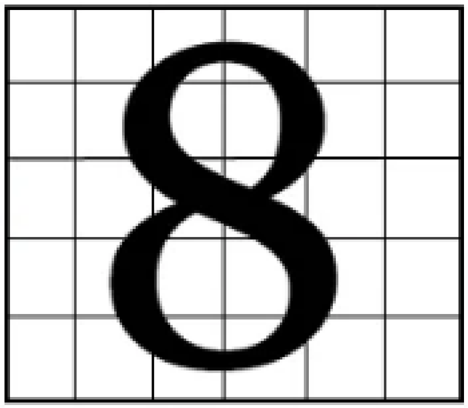


LeNet-5 1998, Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner

- Convolution layers
- Subsampling layers - max pooling, average pooling…
- Fully connected layers
- Activations - mostly Rectified Linear Units (ReLu) these days.

- ## CNN – image representation

Binary image

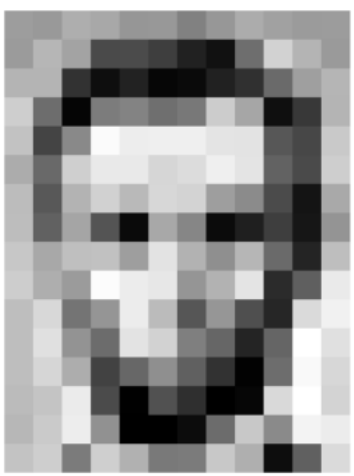


Represented in the form of an array

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

Digit 8 represented in the form of pixels of 0's and 1's

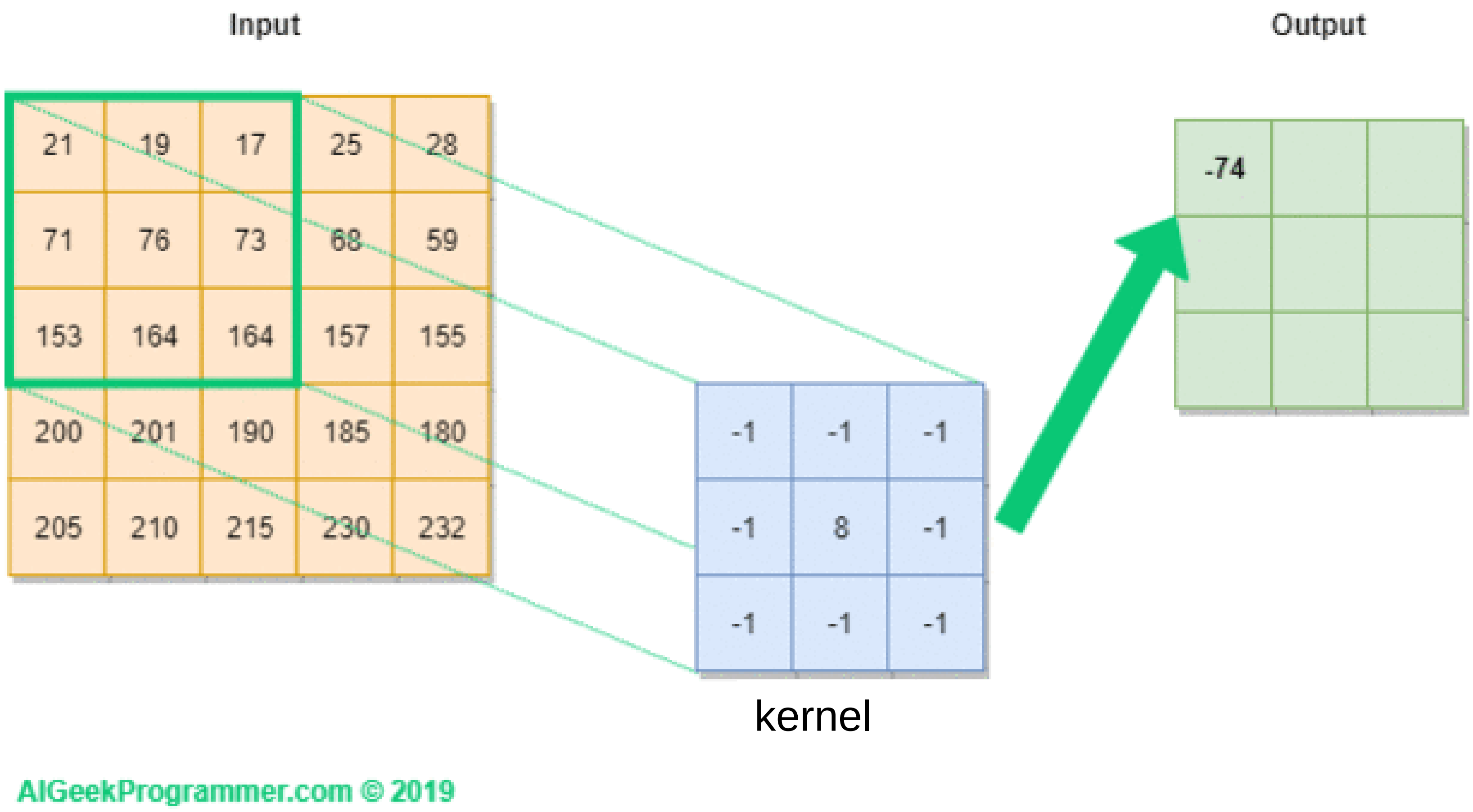
Color image



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	124	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

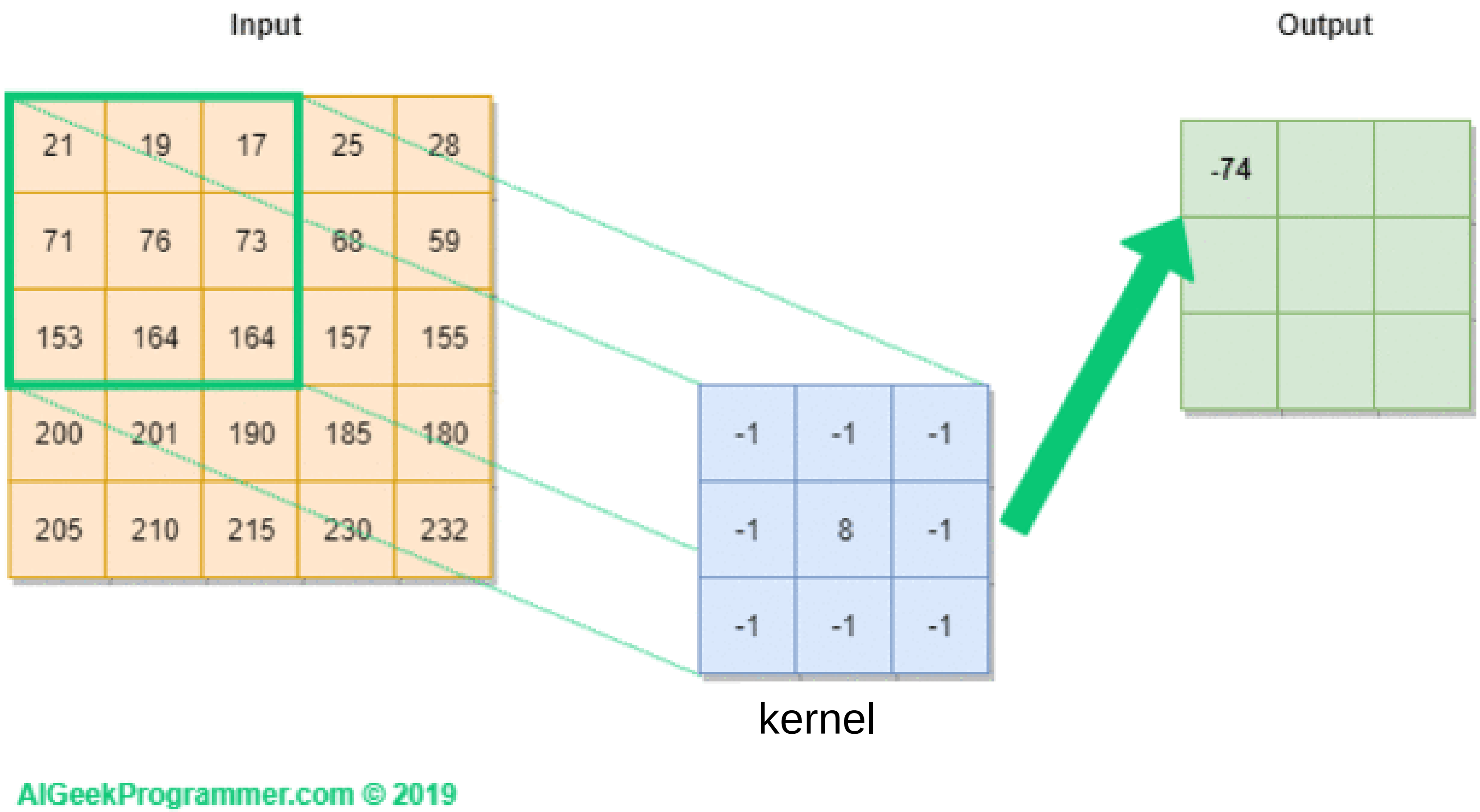
0~255

- # CNN - convolution



Single channel, one kernel

- ## CNN – output size

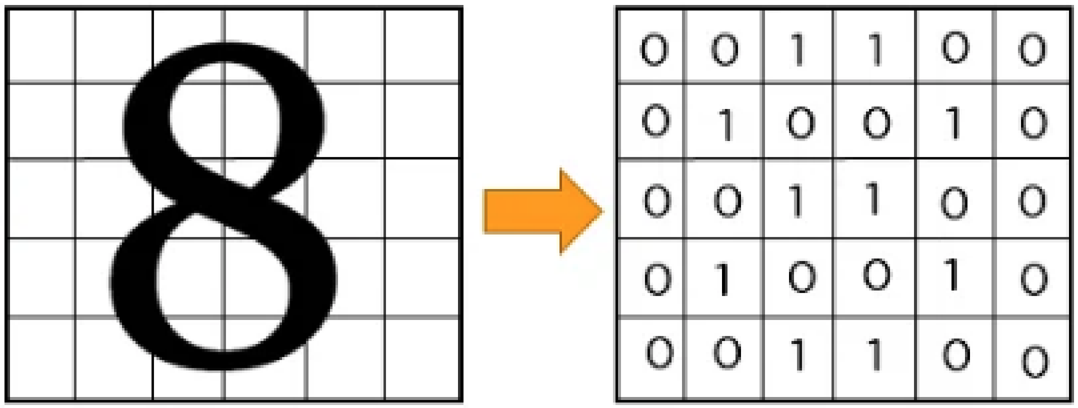


width of output: floor((width-kernel_size)/stride) + 1

height of output: floor((height-kernel_size)/stride) + 1

• CNN – color image

Binary image

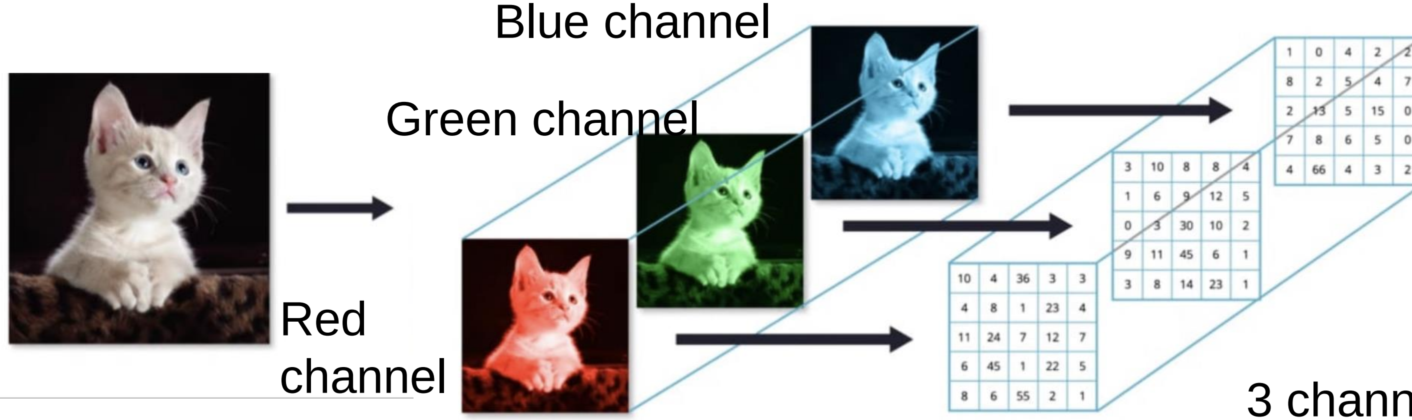


One channel

Represented in the form of an array

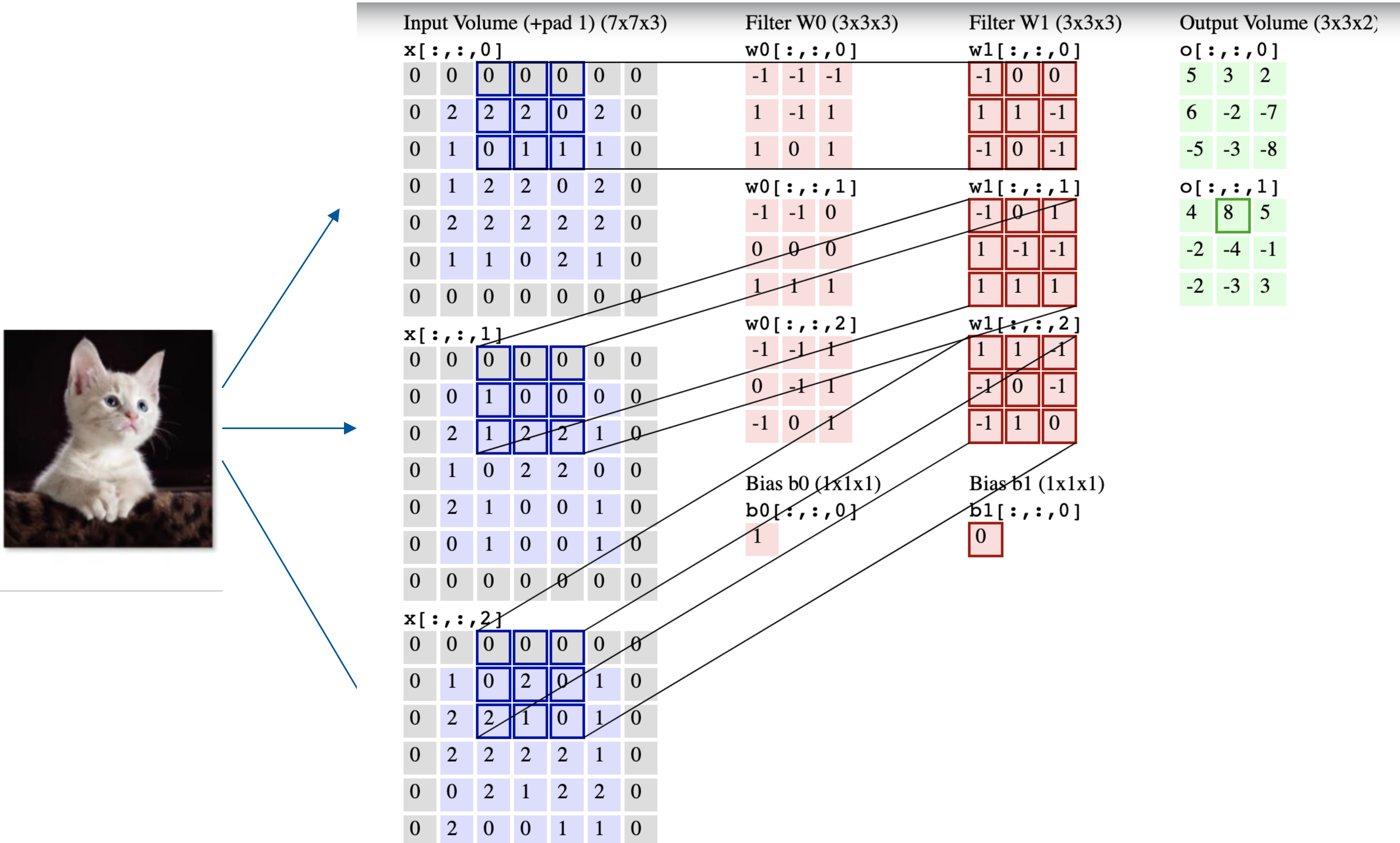
Digit 8 represented in the form of pixels of 0's and 1's

Color image

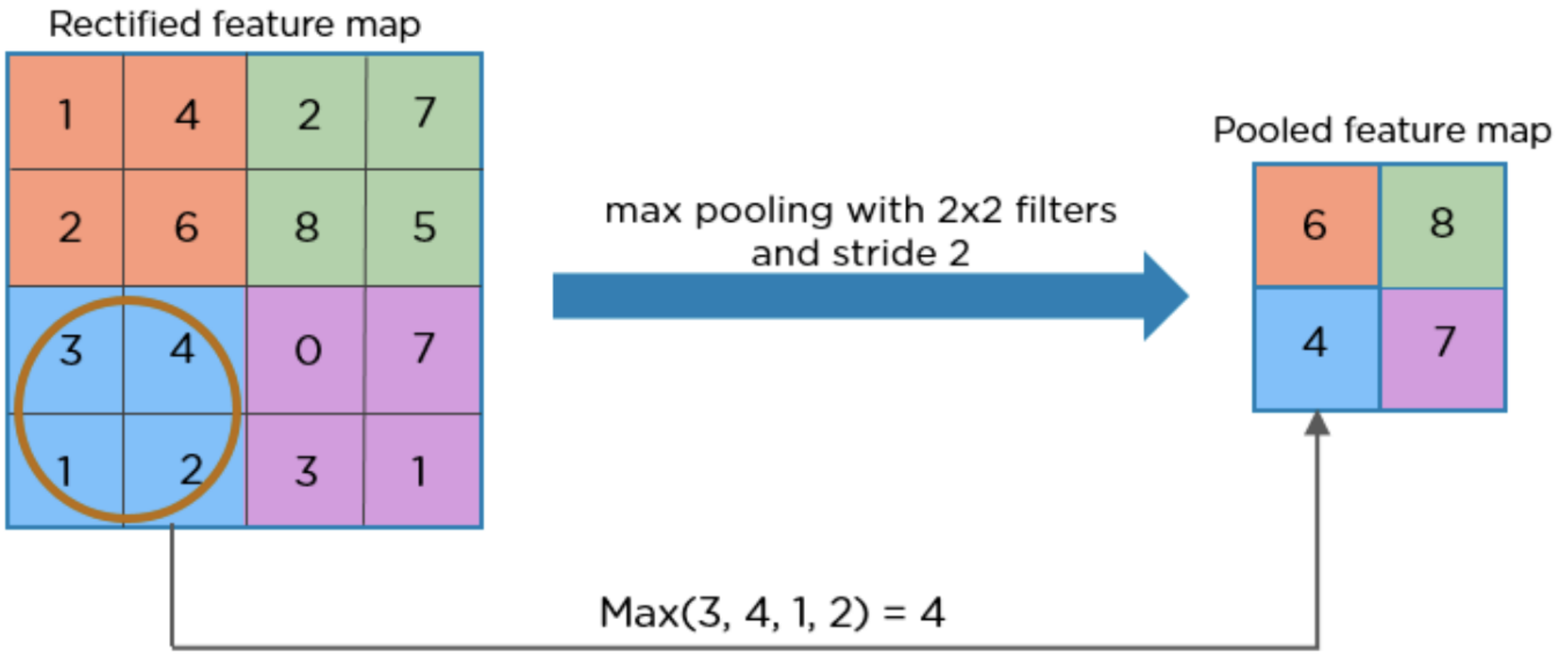


3 channels

CNN - convolution



- ## CNN - pooling



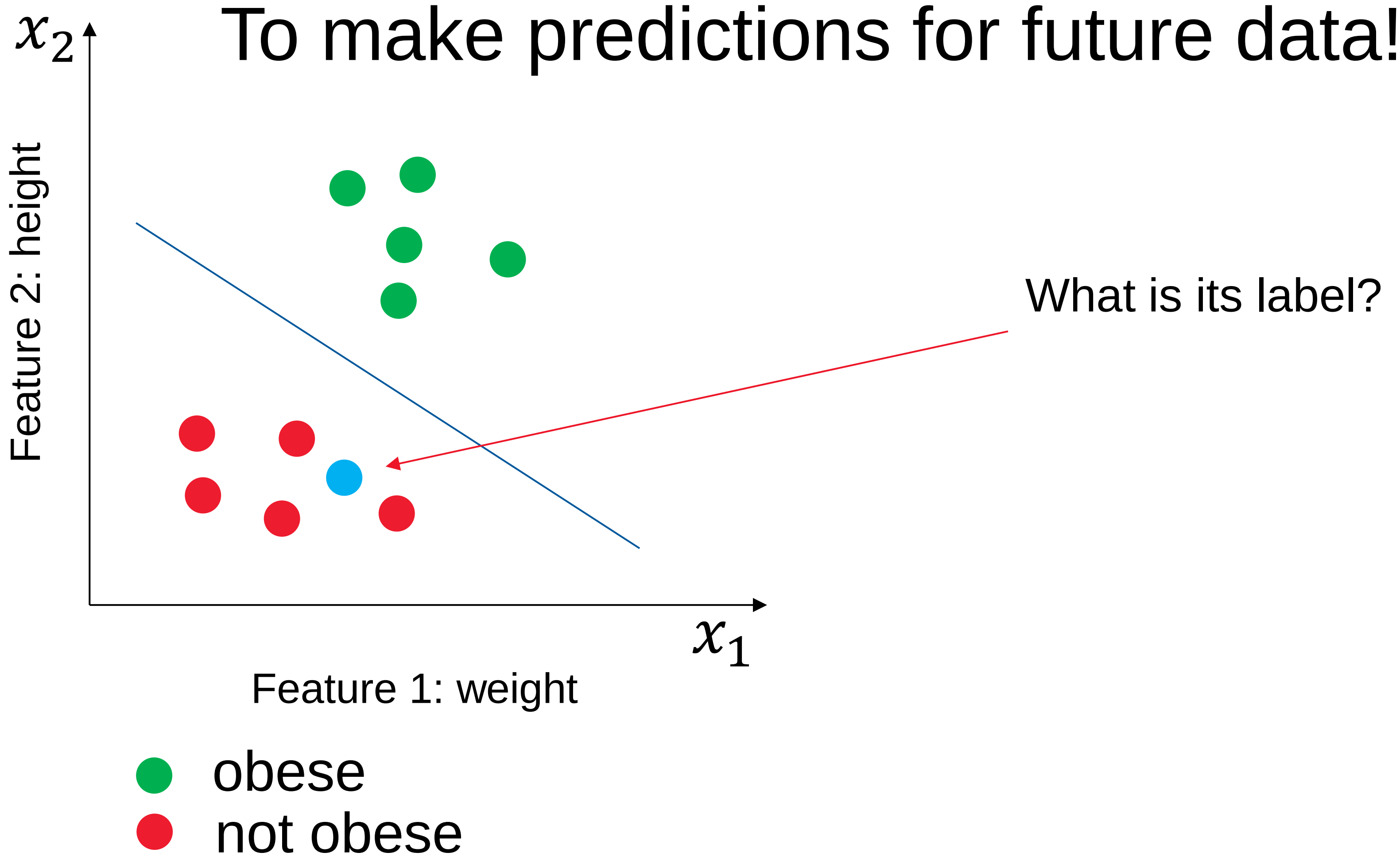
Rectified feature map
1
4
2
7
2
6
8
5
3
4
0
7
1
2
3
1
max pooling with 2x2 filters
and stride 2
Pooled feature map
6
8
4
7
Max(3, 4, 1, 2) = 4

Classifier Review

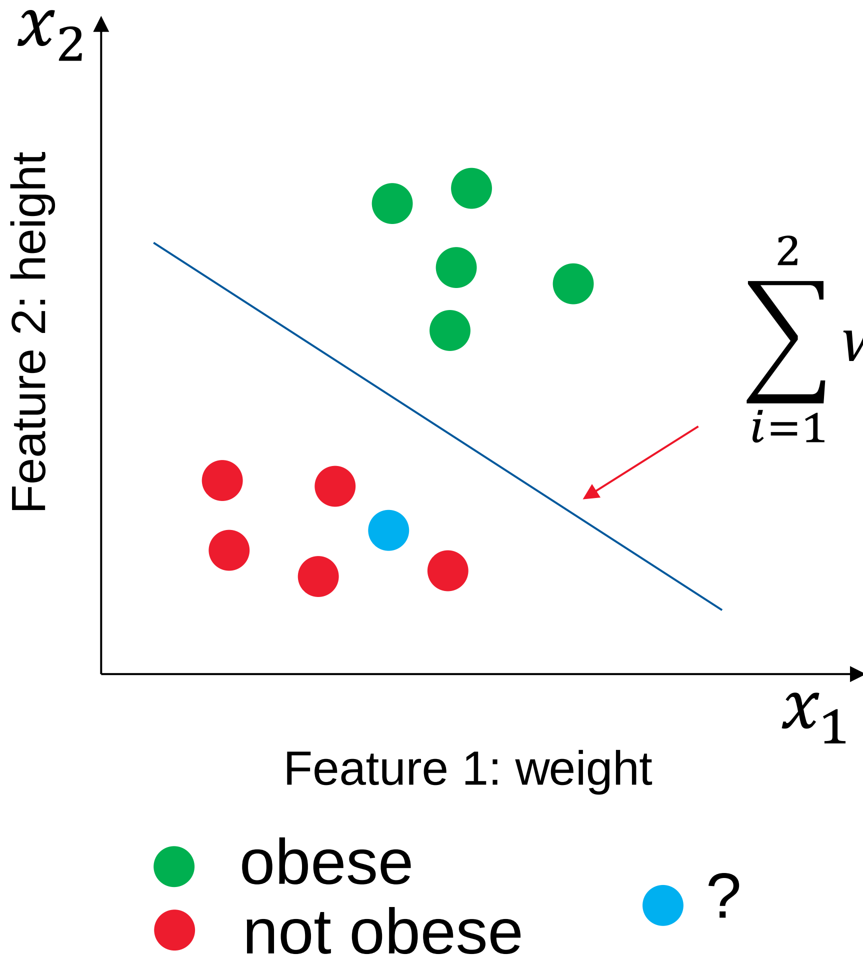
Why do we learn classification, regression or clustering?

Classifier Review

To make predictions for future data!



Classifier Review

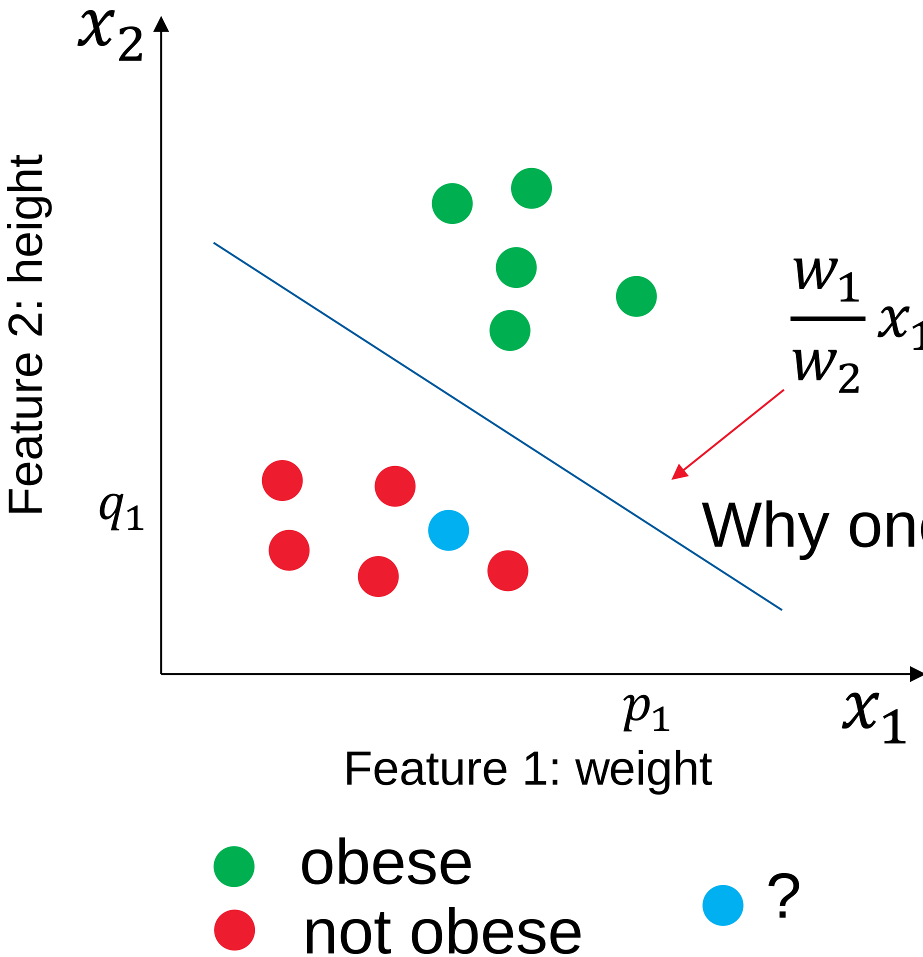


$$\sum_{i=1}^{2} w_i x_i + b = 0$$

$$w_1 x_1 + w_2 x_2 + b = 0$$

$$\frac{w_1}{w_2} x_1 + x_2 + \frac{b}{w_2} = 0$$

Classifier Review

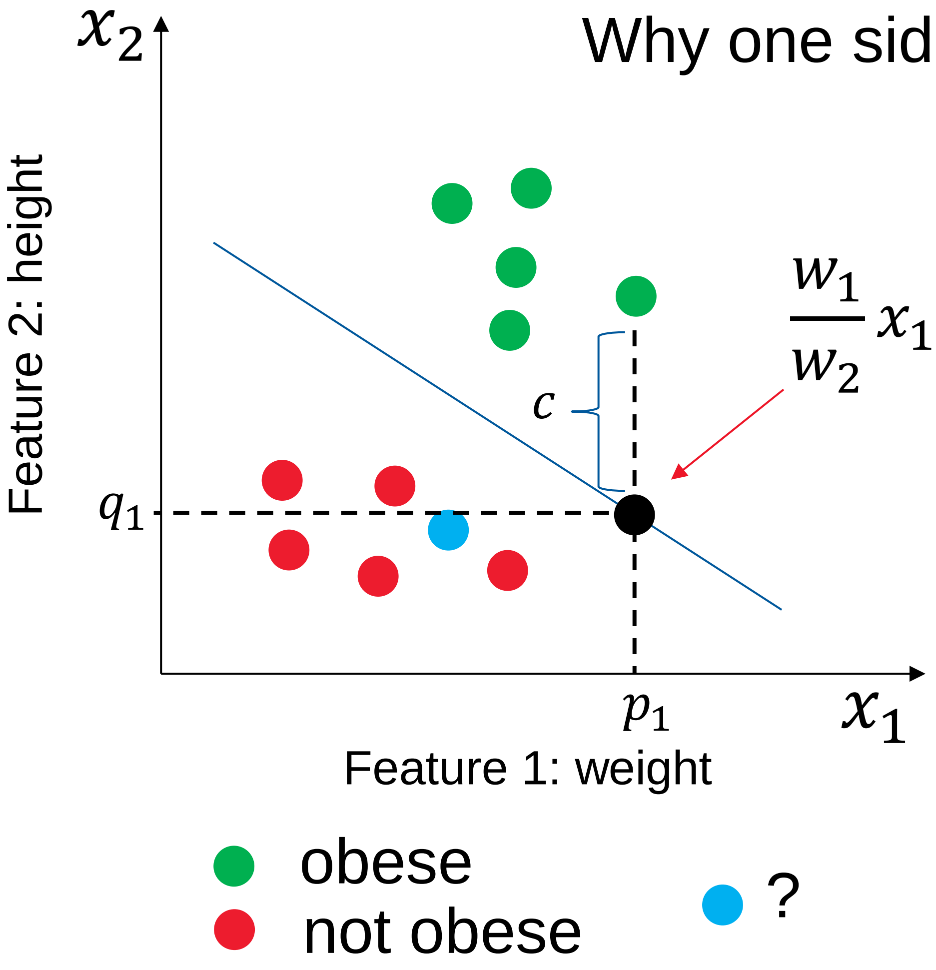


$$\frac{w_1}{w_2}x_1 + x_2 + \frac{b}{w_2} = 0$$

Why one side >0 and the other side <0?

Classifier Review

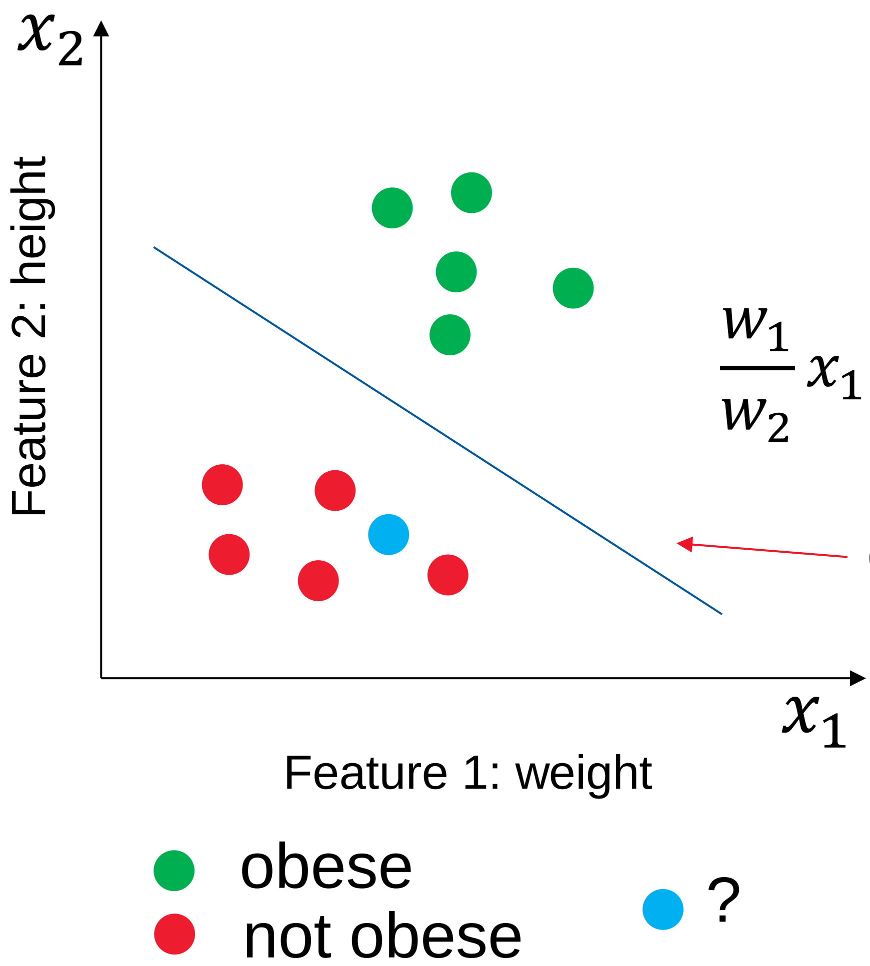
Why one side >0 and the other side <0?



$$\frac{w_1}{w_2}x_1 + x_2 + \frac{b}{w_2} = 0$$

Classifier Review

Techniques to compute classifier:
Threshold,
Max margin classifier
Soft margin classifier
Support Vector Machines



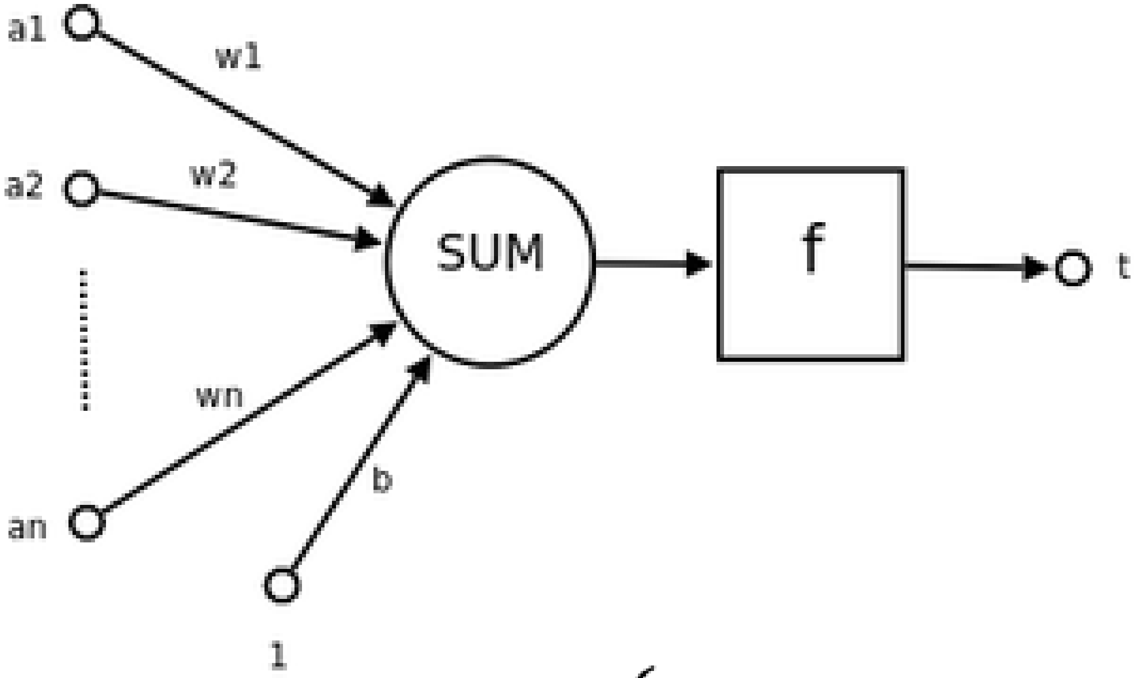
$$\frac{w_1}{w_2} x_1 + x_2 + \frac{b}{w_2} = 0$$

Classifier: $\hat{y} = \frac{w_1}{w_2} x_1 + x_2 + \frac{b}{w_2}$

$$\hat{y} = \sum_{i=1}^{2} w_i x_i + b$$

- Perceptron

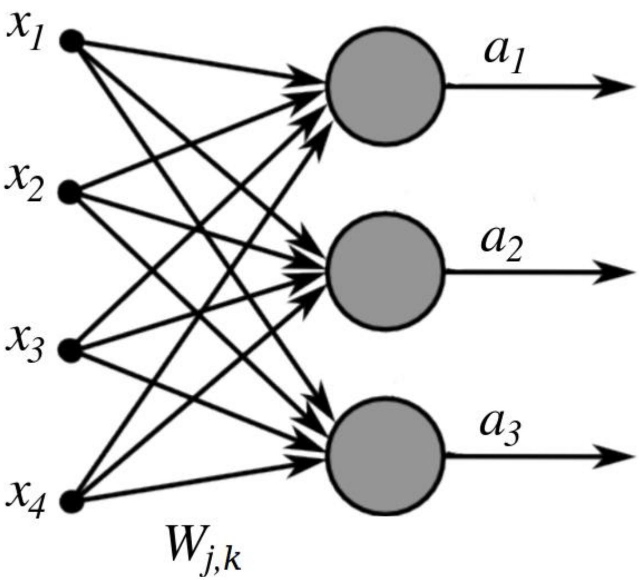
f is activation function



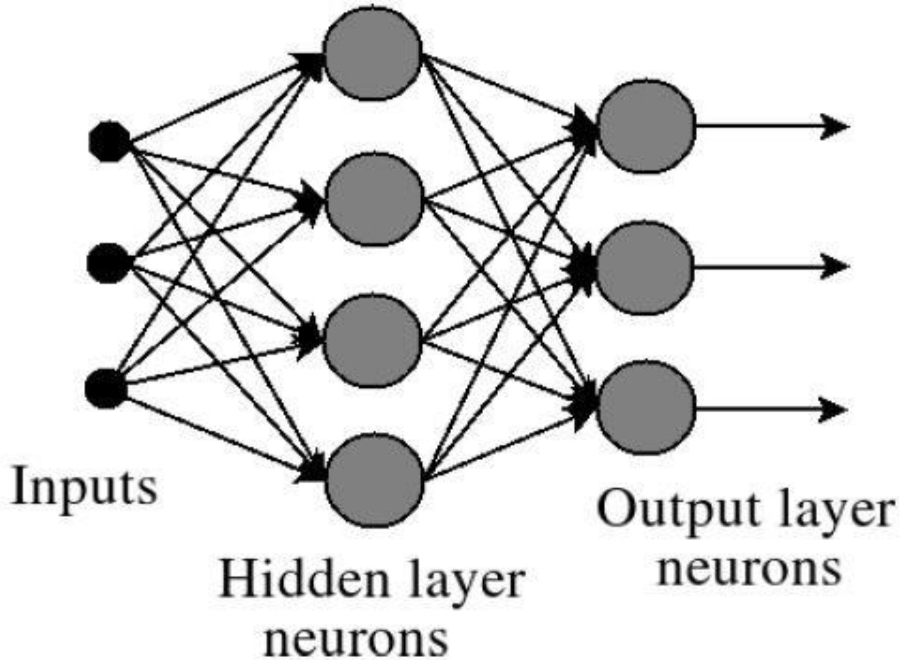
$$scores = \sum_{i}^{N} w_i x_i + b$$

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0, \\ 0 & \text{otherwise} \end{cases}$$

Multi-layer Perceptron (MLP) – Linear/Non-Linear



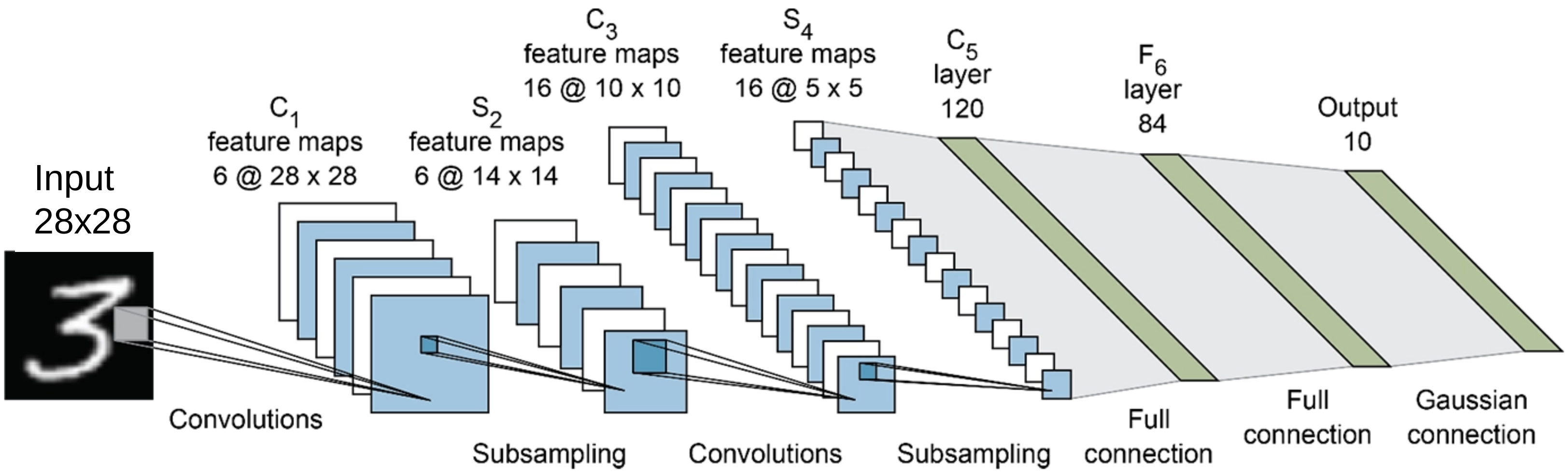
Perceptrons



MLP

MLPs are more expressive than Perceptrons since they can learn highly non-linear class boundaries.

Building Blocks of Deep CNNs



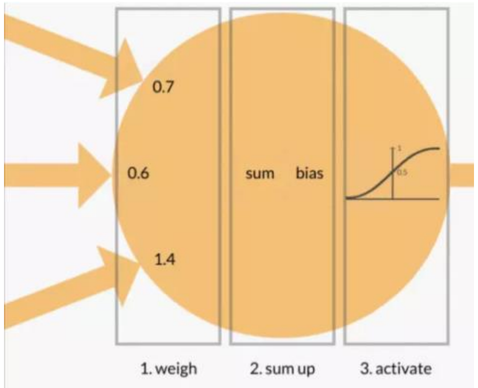
LeNet-5 1998, Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner

- Convolution layers
- Subsampling layers - max pooling, average pooling…
- Fully connected layers
- Activations - mostly Rectified Linear Units (ReLu) these days.

Activation

Activation Function: Sigmoid

- Non-linearity based on sigmoid.

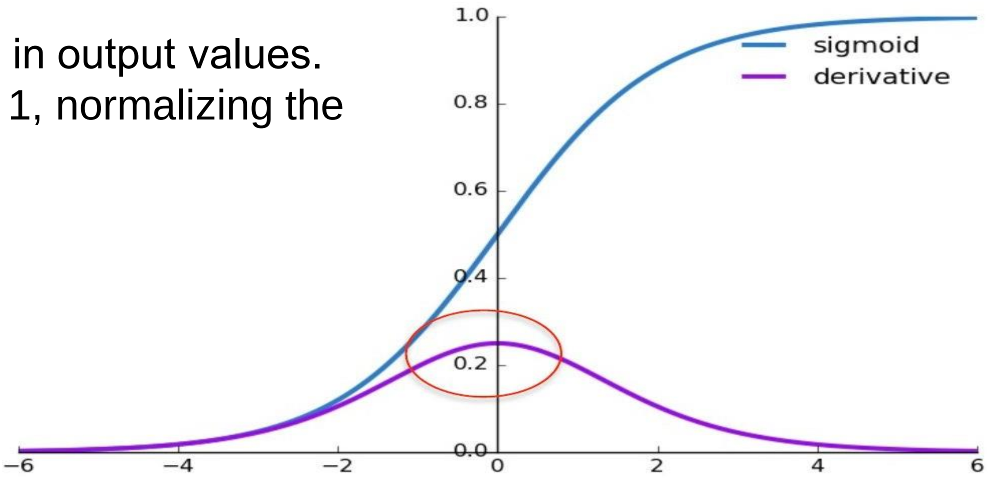


$$g_{sig}(in) = \frac{1}{1 + e^{-in}}$$

$$g'_{sig}(in) = \frac{1}{(1 + e^{-in})}\left(1 - \frac{1}{(1 + e^{-in})}\right)$$

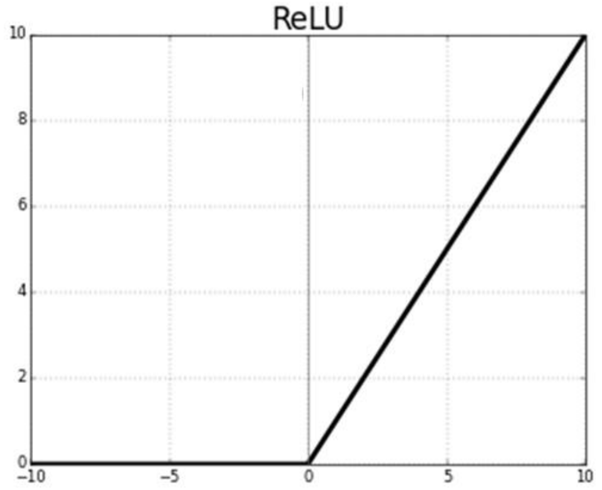
$$= g_{sig}(in)(1 - g_{sig}(in))$$

- Advantages
  - Smooth gradient, preventing “jumps” in output values.
  - Output values bound between 0 and 1, normalizing the output of each neuron.
- Disadvantages
  - Vanishing
  - Computationally expensive



Rectified Linear Units (ReLU)

- Maximum gradient magnitude is 1
- Still non-linear
- Gradient shape?



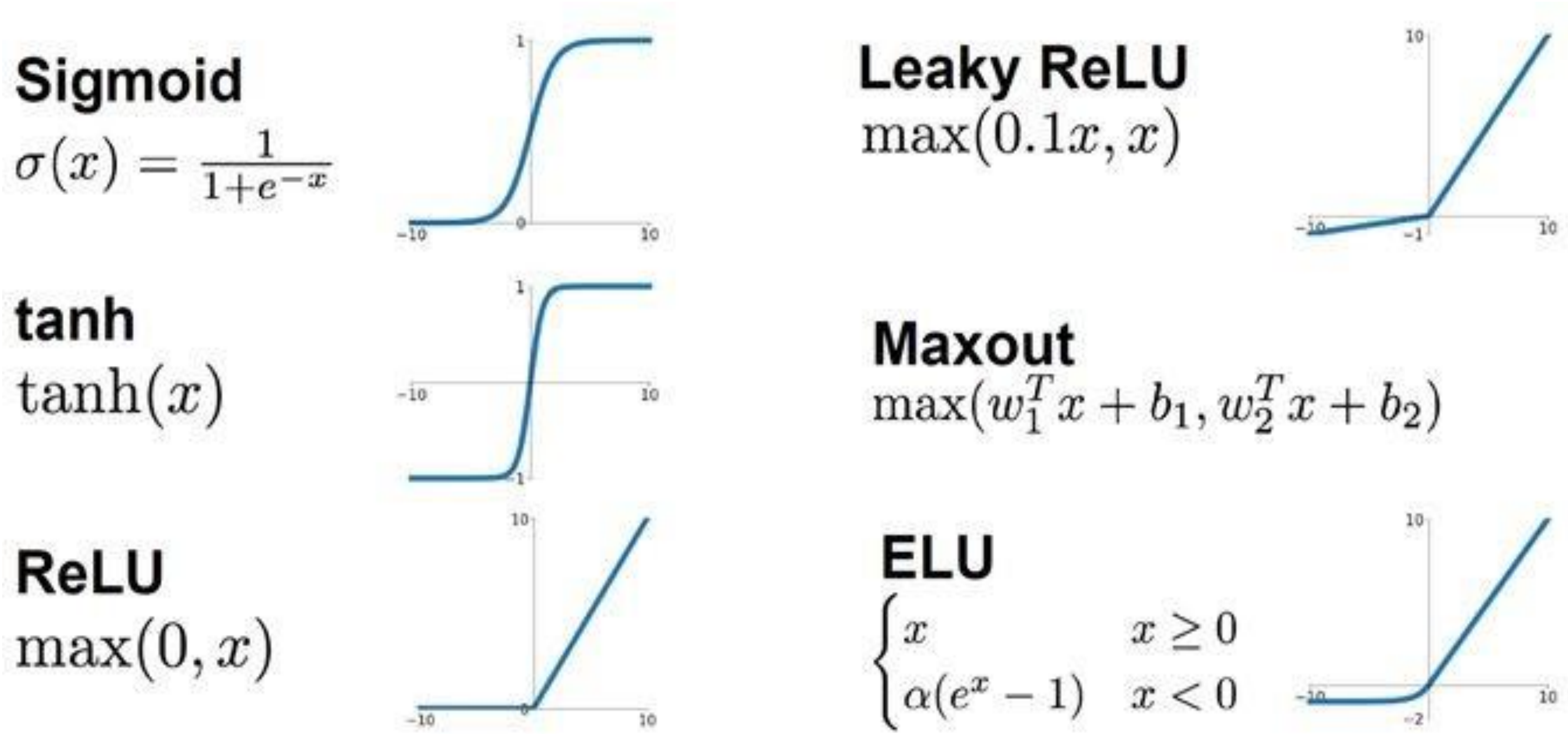
$$f(x) = \max(0, x).$$

$$f'(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}$$

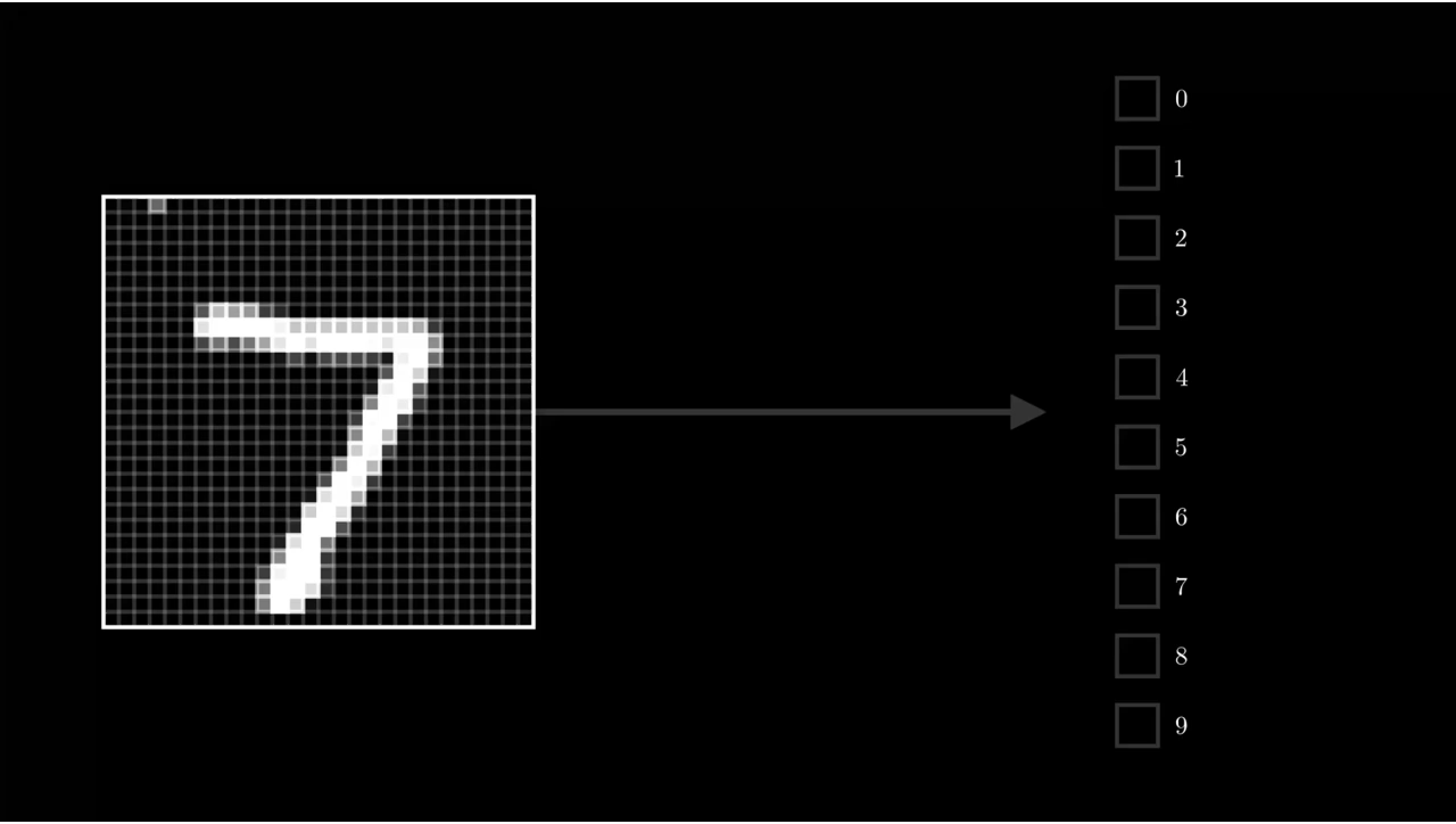
- Advantages
  - Computationally efficient—allows the network to converge very quickly
  - Non-linear—although it looks like a linear function, ReLU has a derivative function and allows for backpropagation

- Disadvantages
  - The Dying ReLU problem—when inputs approach zero, or are negative, the gradient of the function becomes zero, the network cannot perform backpropagation and cannot learn.

Rectified Linear Units (ReLU)

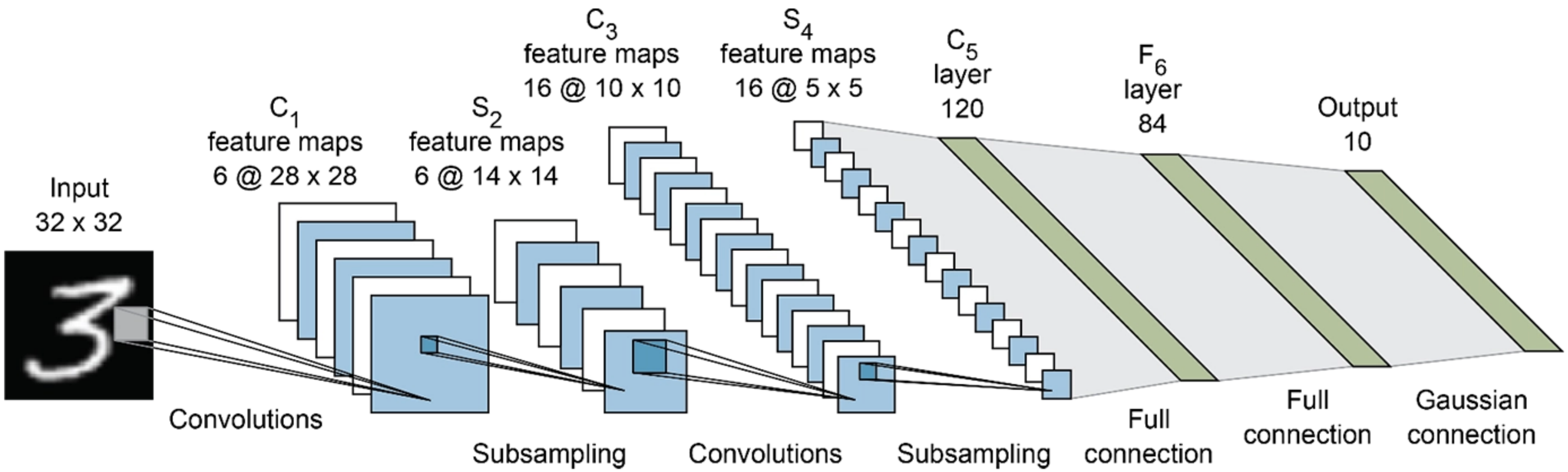


Put all these layers together



https://www.youtube.com/watch?v=jDe5BAsT2-Y

Coding



Implement LeNet-5 in PyTorch

LeNet code

Install PyTorch running environment

https://pytorch.org/get-started/previous-versions/

v1.11.0

You can choose other versions.

Conda

Use the command according to your computer

OSX

```
# conda
conda install pytorch==1.11.0 torchvision==0.12.0 torchaudio==0.11.0 -c pytorch
```

Linux and Windows

```
# CUDA 10.2
conda install pytorch==1.11.0 torchvision==0.12.0 torchaudio==0.11.0 cudatoolkit=10.2 -c pytorch

# CUDA 11.3
conda install pytorch==1.11.0 torchvision==0.12.0 torchaudio==0.11.0 cudatoolkit=11.3 -c pytorch

# CPU Only
conda install pytorch==1.11.0 torchvision==0.12.0 torchaudio==0.11.0 cpuonly -c pytorch
```

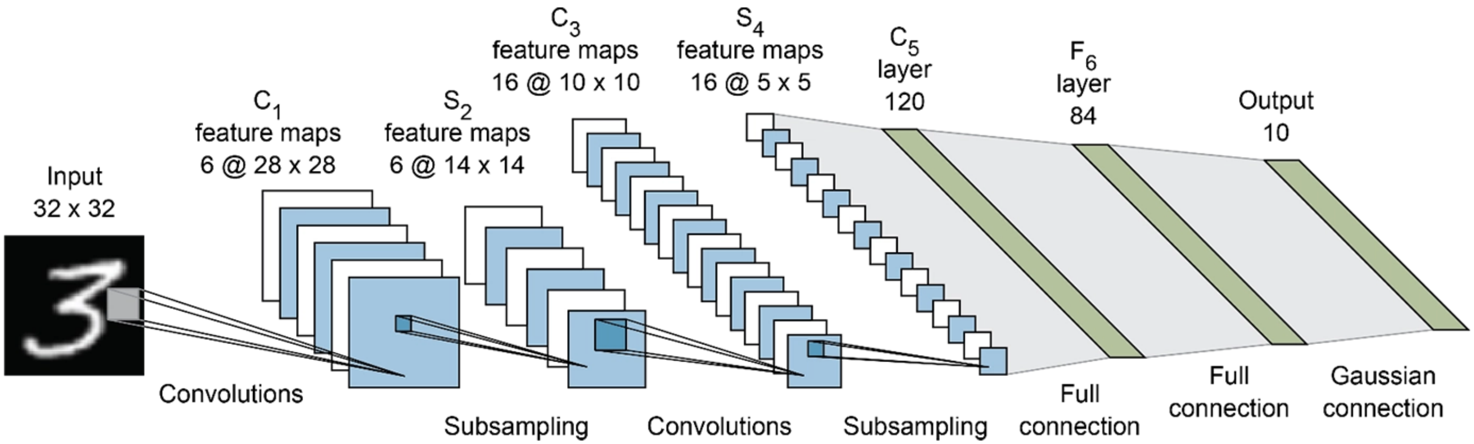
LeNet code

PYTORCH MXNET JAX TENSORFLOW

```
import torch
from torch import nn
from d2l import torch as d2l
```

https://d2l.ai/chapter_convolutional-neural-networks/lenet.html

LeNet co



PYTORCH MXNET JAX TENSORFLOW

```
def init_cnn(module):  #@save
    """Initialize weights for CNNs."""
    if type(module) == nn.Linear or type(module) == nn.Conv2d:
        nn.init.xavier_uniform_(module.weight)

class LeNet(d2l.Classifier):  #@save
    """The LeNet-5 model."""
    def __init__(self, lr=0.1, num_classes=10):
        super().__init__()
        self.save_hyperparameters()
        self.net = nn.Sequential(
            nn.LazyConv2d(6, kernel_size=5, padding=2), nn.Sigmoid(),
            nn.AvgPool2d(kernel_size=2, stride=2),
            nn.LazyConv2d(16, kernel_size=5), nn.Sigmoid(),
            nn.AvgPool2d(kernel_size=2, stride=2),
            nn.Flatten(),
            nn.LazyLinear(120), nn.Sigmoid(),
            nn.LazyLinear(84), nn.Sigmoid(),
            nn.LazyLinear(num_classes))
```

LeNet code

PYTORCH MXNET JAX TENSORFLOW

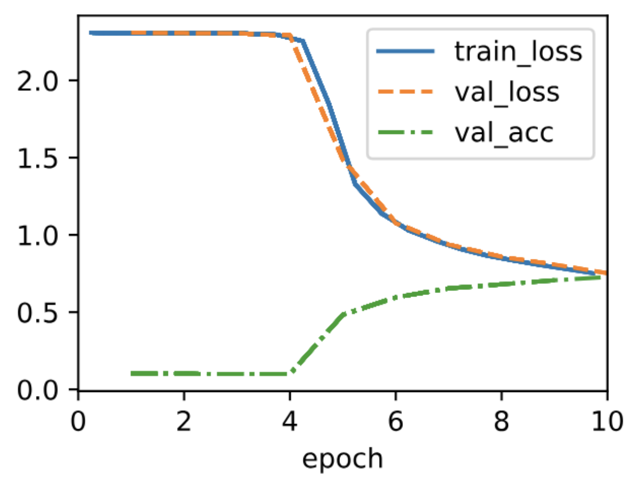
```python
@d2l.add_to_class(d2l.Classifier)  #@save
def layer_summary(self, X_shape):
    X = torch.randn(*X_shape)
    for layer in self.net:
        X = layer(X)
        print(layer.__class__.__name__, 'output shape:\t', X.shape)

model = LeNet()
model.layer_summary((1, 1, 28, 28))
```

```
Conv2d output shape:         torch.Size([1, 6, 28, 28])
Sigmoid output shape:        torch.Size([1, 6, 28, 28])
AvgPool2d output shape:      torch.Size([1, 6, 14, 14])
Conv2d output shape:         torch.Size([1, 16, 10, 10])
Sigmoid output shape:        torch.Size([1, 16, 10, 10])
AvgPool2d output shape:      torch.Size([1, 16, 5, 5])
Flatten output shape:        torch.Size([1, 400])
Linear output shape:         torch.Size([1, 120])
Sigmoid output shape:        torch.Size([1, 120])
Linear output shape:         torch.Size([1, 84])
Sigmoid output shape:        torch.Size([1, 84])
Linear output shape:         torch.Size([1, 10])
```

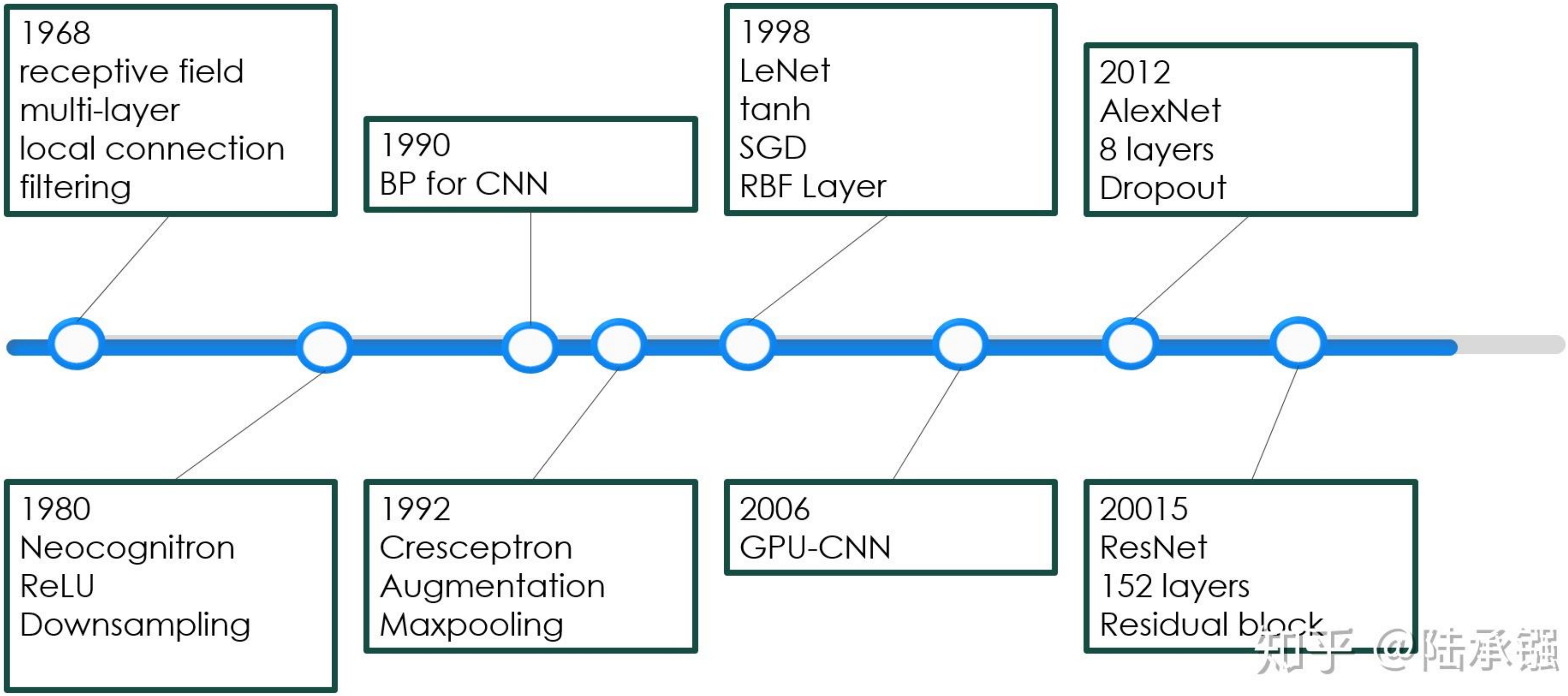
LeNet code

PYTORCH MXNET JAX TENSORFLOW

```
trainer = d2l.Trainer(max_epochs=10, num_gpus=1)
data = d2l.FashionMNIST(batch_size=128)
model = LeNet(lr=0.1)
model.apply_init([next(iter(data.get_dataloader(True)))[0]], init_cnn)
trainer.fit(model, data)
```

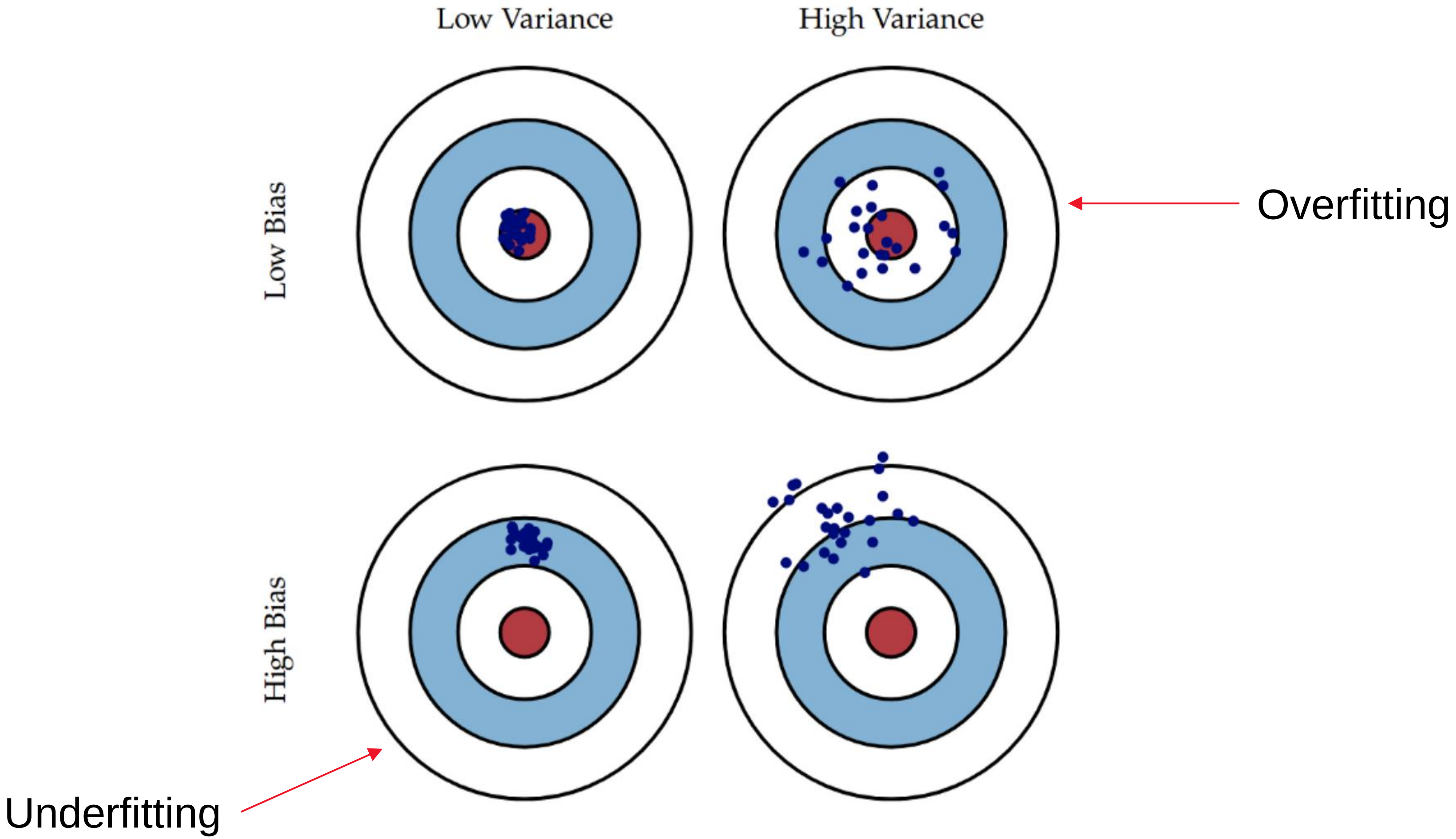


Deep learning



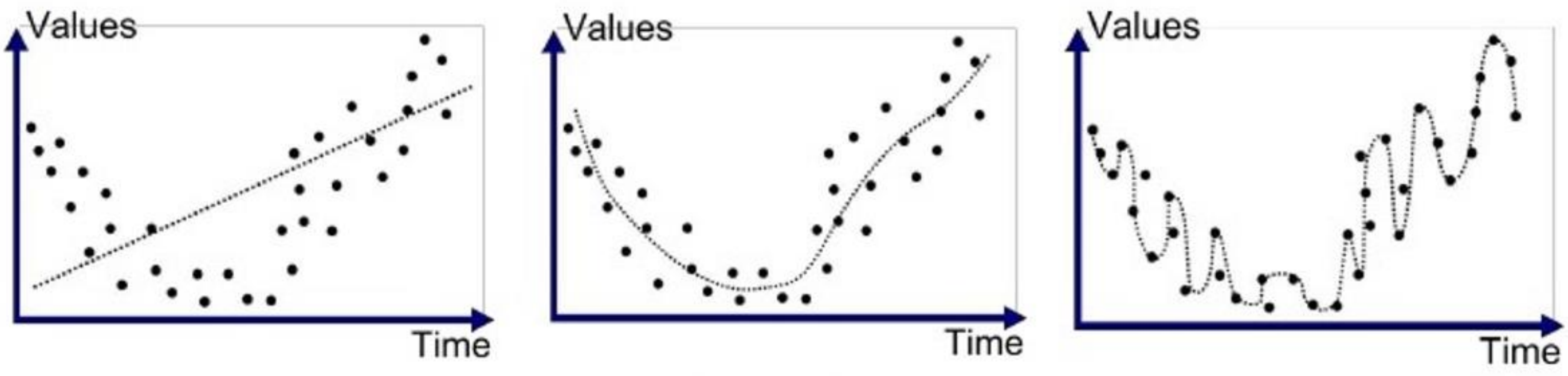
Bias & Variance

Bias & Variance



High bias: consistently make erroneous predictions
High Variance: high residual to the mean

Underfit & Good fit & Overfit

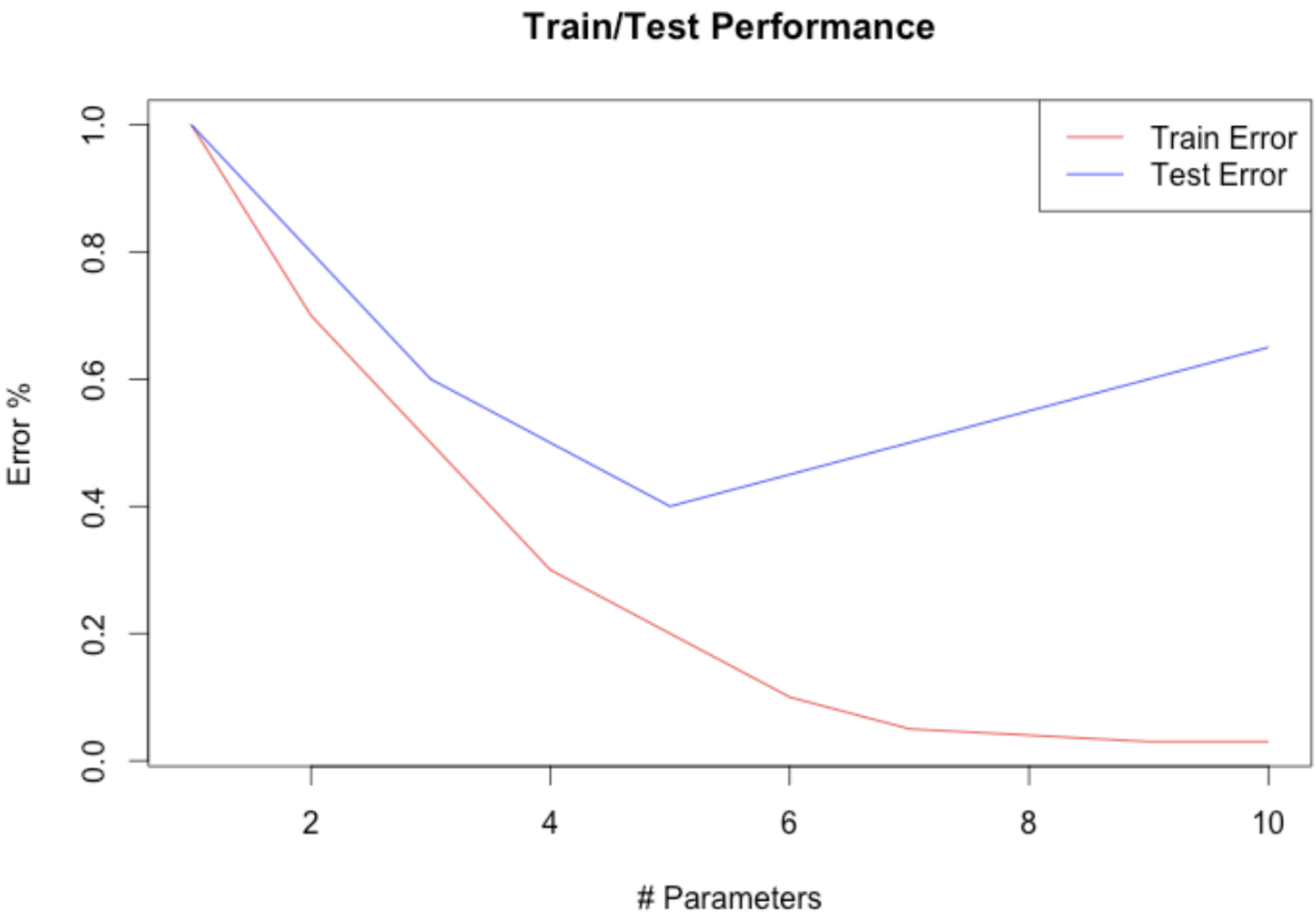


Underfitted

Good Fit/Robust

Overfitted

Bias & Variance

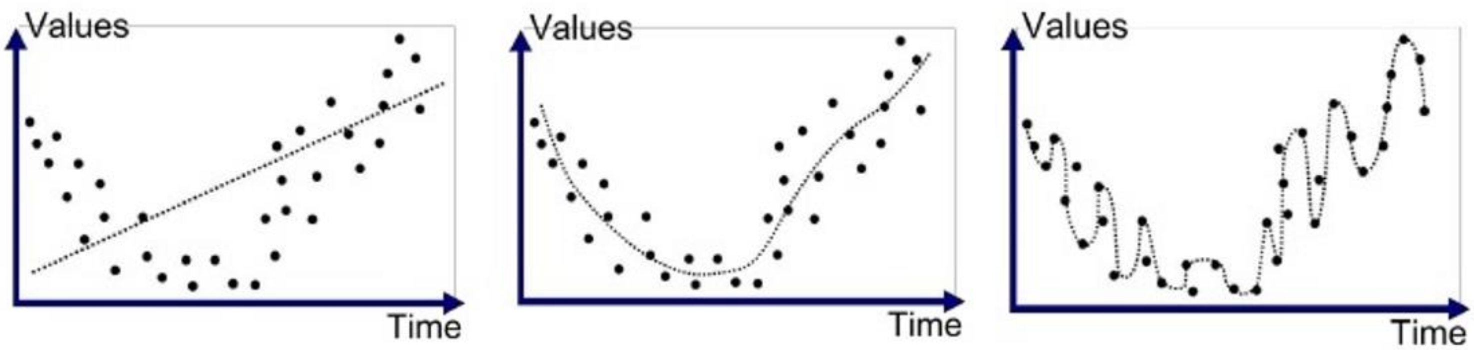


Regularization

Regularization

- Optimizing a loss function to learn parameters

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Fitting to data}} + \underbrace{\lambda R(W)}_{\text{Choose the simplest model}}$$



Underfitted Good Fit/Robust Overfitted

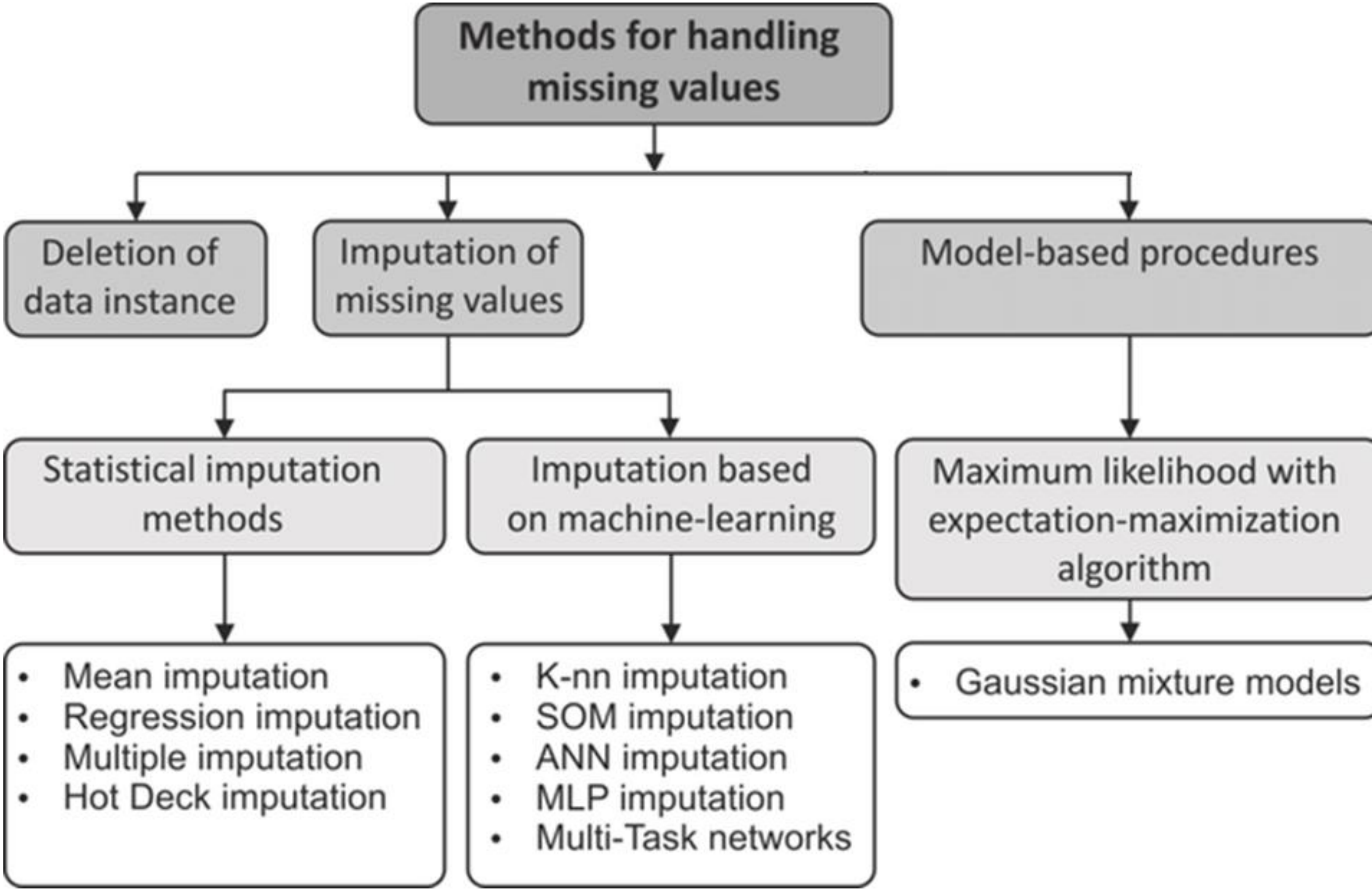
Regularization

- Commonly-used regularizers
  - L2-regularization (Lasso): $R_{L_2}(w) \triangleq ||W||_2^2$
  - L1-regularization (Ridge): $R_{L_1}(w) \triangleq \sum_{k=1}^{Q} ||W||_1$
  - Drop-out: it randomly selects some nodes and removes them along with all of their incoming and outgoing connections as shown below.
  - Early stopping: keep one part of the training set as the validation set. When we see that the performance on the validation set is getting worse, we immediately stop the training on the model. This is known as early stopping.

Handle Missing Value

Handle Missing Values

- When models cannot handle missing values:
  - Too many missing values and the dataset is big, then delete the instance/feature
  - Categorical data: transform NaN as new category; Replace by most frequent value; Replace using an algorithm like KNN using the neighbours; Predict the observation using a multiclass predictor, etc.
  - Continuous data: NaN as 0; mean/medium/mode; replace with value before or after; interpolation; regression.



Source: Jaroslav Bendl, 2016

Reference

- Hungyi Lee Tutorial

  http://speech.ee.ntu.edu.tw/~tlkagk/courses_ML20.html