ARTIFICIAL NEURAL NETWORKS

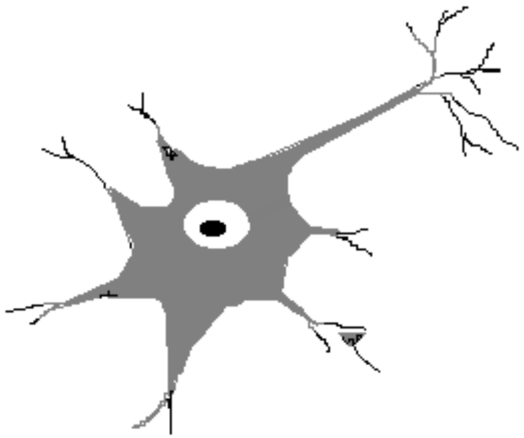
DEEP LEARNING

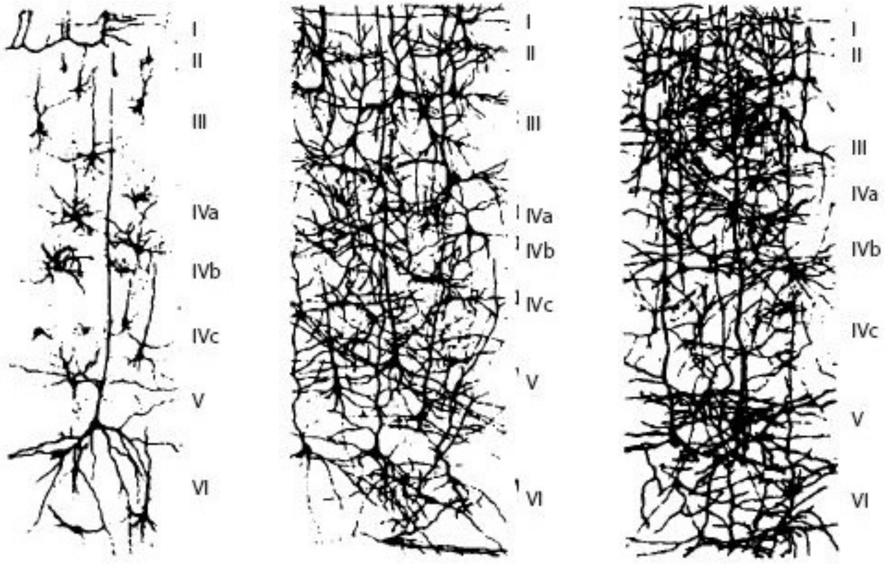
Key Developments

- Proper evaluation of machine learning methods
- Significant increase in amount of data
- Deeper and larger networks
- Faster training using GPUs

Motivation

- Simulate the biological neural system
- The brain consists of neurons linked together
- An artificial neural network (ANN) consists of nodes connected together by links





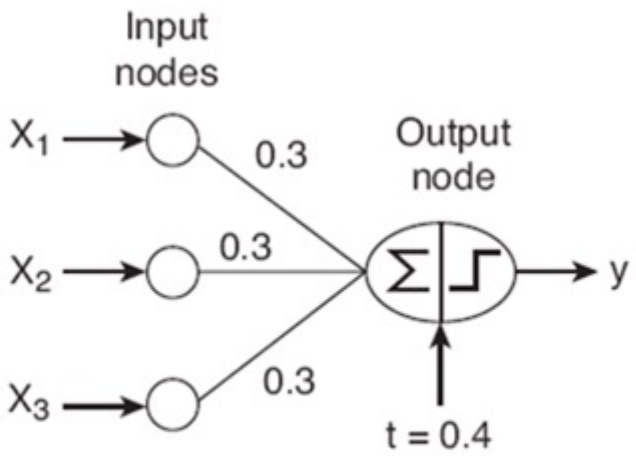
Perceptron

- Simplest form of ANN
- Binary classifier
- Consists of two types of nodes:
  - Input nodes: represent the input attributes
  - Output node: represents the model output
- Each input node is connected via a weighted link to the output node
- Training a perceptron models consists of adapting the link weights

Example

X_1	X_2	X_3	y
1	0	0	–1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	–1
0	1	0	–1
0	1	1	1
0	0	0	–1

(a) Data set.



(b) Perceptron.

t: bias factor

Sign function: activation

$$\hat{y} = \begin{cases} 1, & \text{if } 0.3x_1 + 0.3x_2 + 0.3x_3 - 0.4 > 0 \\ -1, & \text{if } 0.3x_1 + 0.3x_2 + 0.3x_3 - 0.4 < 0 \end{cases}$$

$$\begin{aligned} \hat{y} &= \mathrm{sign}(w_d x_d + w_{d-1} x_{d-1} + \cdots + w_1 x_1 - t) \\ &= \mathrm{sign}(w_d x_d + w_{d-1} x_{d-1} + \cdots + w_1 x_1 - w_0 x_0) = sign(w \cdot x) \end{aligned}$$

Perceptron Learning

- Initialize the weights to random values $(w_1, w_2, \ldots, w_m)$
- Keep updating the weights until the output is consistent with the class labels:
  - For each example (x_i, y_i) in the data set
    - Compute the predicted label $\hat{y}_i^{(k)}$
    - Adjust the weights: for each w_j:
      - Update $w_j^{(k+1)} = w_j^{(k)} + \lambda(y_i - \hat{y}_i^{(k)})x_{ij}$
- Repeat until training is done
  weights don't change

$w^{(k)}$: weight in the k^{th} iteration

λ: learning rate

x_{ij}: value of j^{th} attribute of i^{th} example x_i

Perceptron Learning

$$w_j^{(k+1)} = w_j^{(k)} + \lambda(y_i - \hat{y}_i^{(k)})x_{ij}$$

- If the prediction is correct:
  - $y - \hat{y} = 0$ so $w_j^{(k+1)} = w_j^{(k)}$ the weight does not change
- If the prediction is incorrect:
  - the weight is increased/decreased to compensate

If $y_i = +1$ (actual) and $\hat{y}_i = -1$ (predicted): $w_j^{(k+1)} = w_j^k + 2\lambda x_{ij}$
If $y_i = -1$ (actual) and $\hat{y}_i = +1$ (predicted): $w_j^{(k+1)} = w_j^k - 2\lambda x_{ij}$

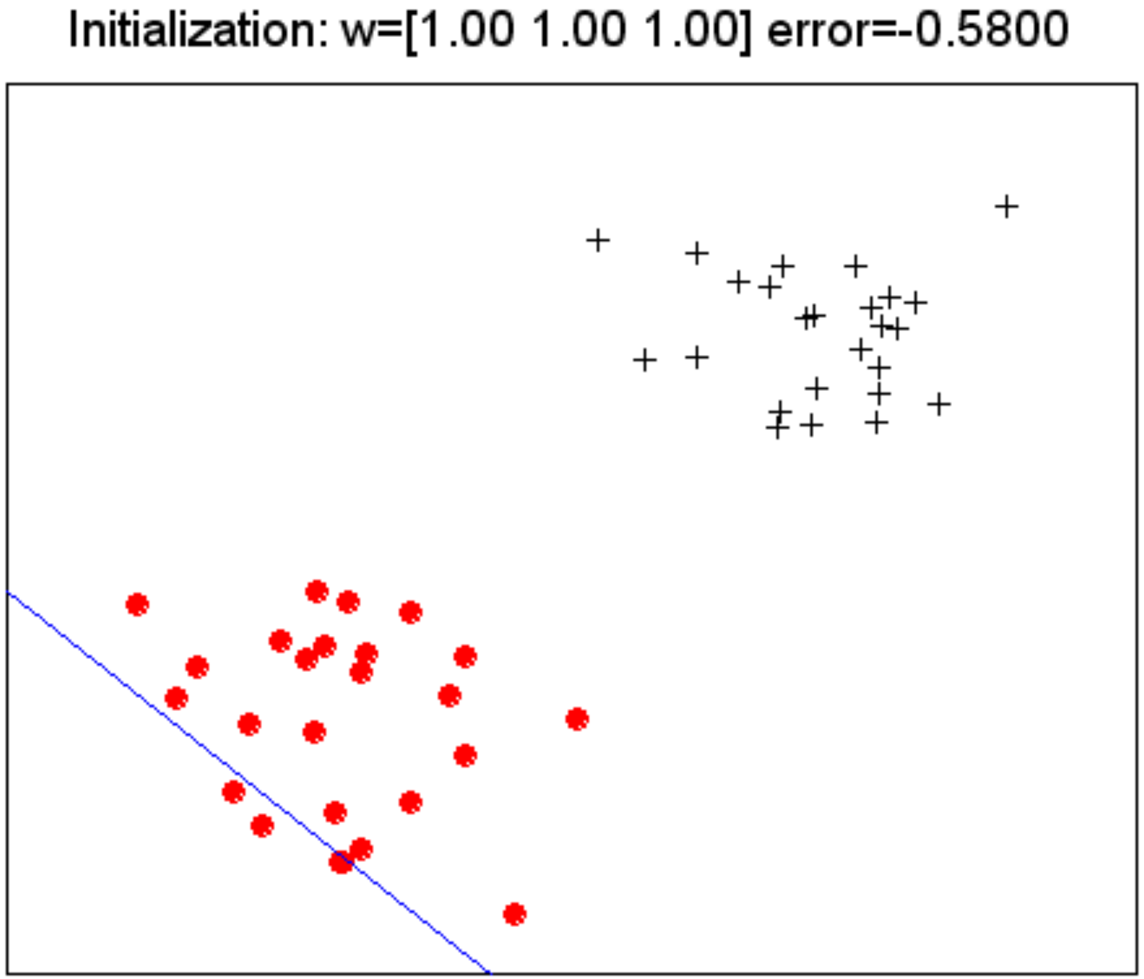
*The perceptron learning algorithm is based on **error correction** rather than gradient descent

Perceptron Learning

$$w_j^{(k+1)} = w_j^{(k)} + \lambda(y_i - \hat{y}_i^{(k)})x_{ij}$$

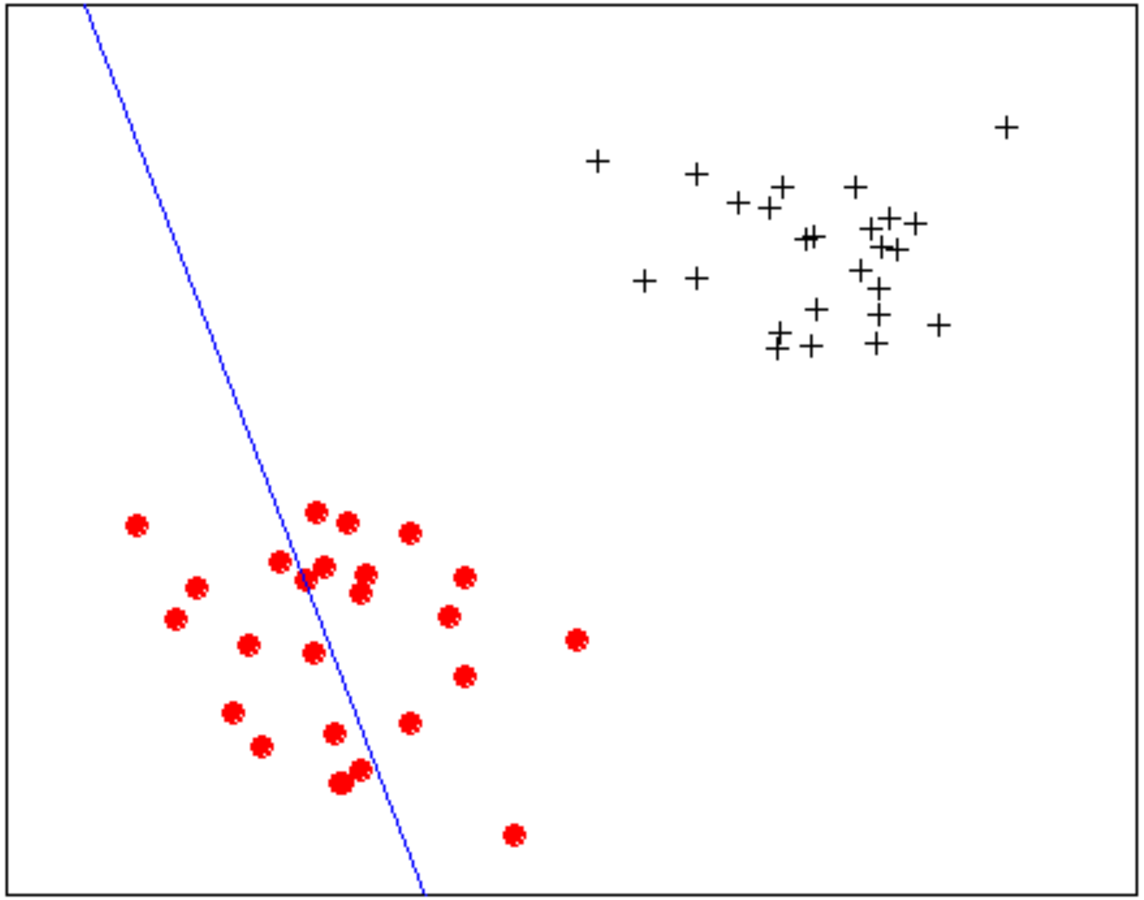
- The weight should not be changed drastically
- The learning rate ($\lambda \in [0,1]$) controls the amount of adjustment
- If λ is close to 1:
  - the new weight influenced by the adjustment amount
- If λ is close to 0:
  - the new weight influenced by the old weight

Example: Perceptron



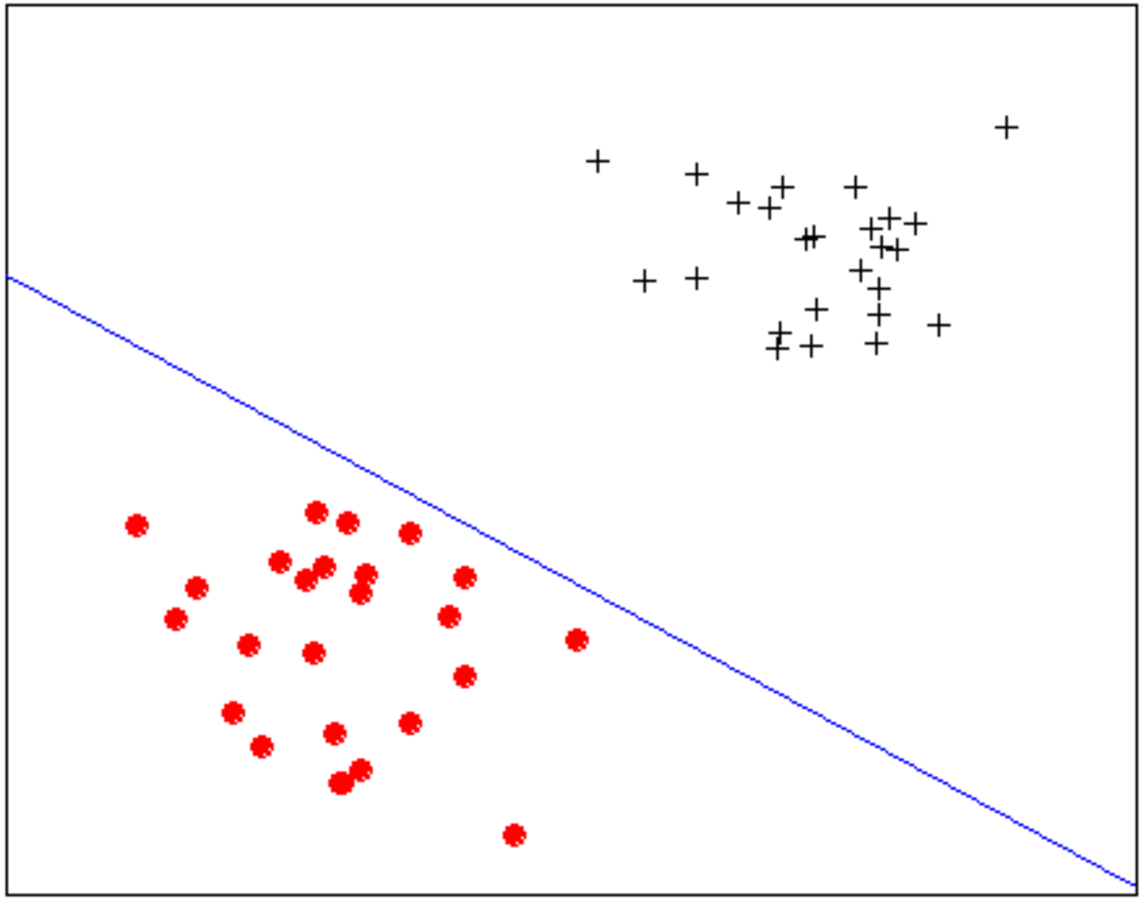
Example: Perceptron

After 1 data points: w=[0.00 1.31 0.39] error=-0.7400

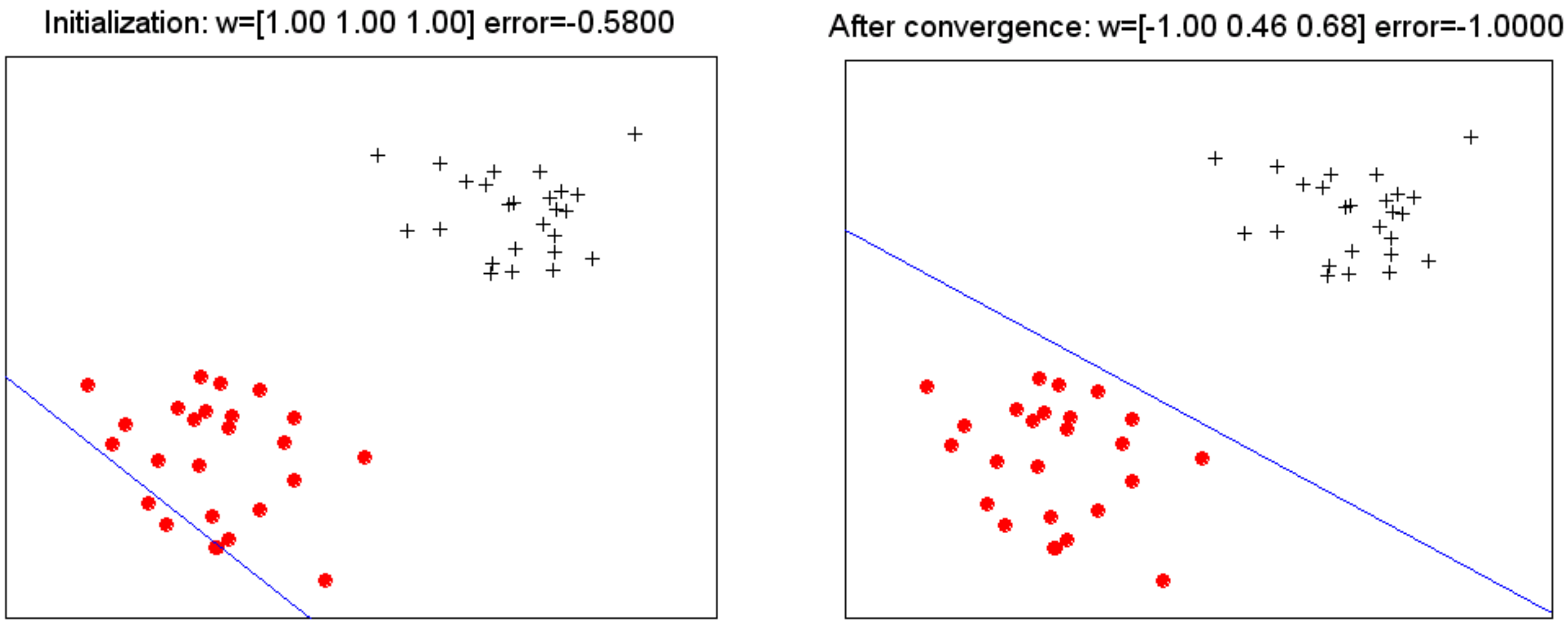


Example: Perceptron

After 6 data points: w=[-1.00 0.46 0.68] error=-1.0000

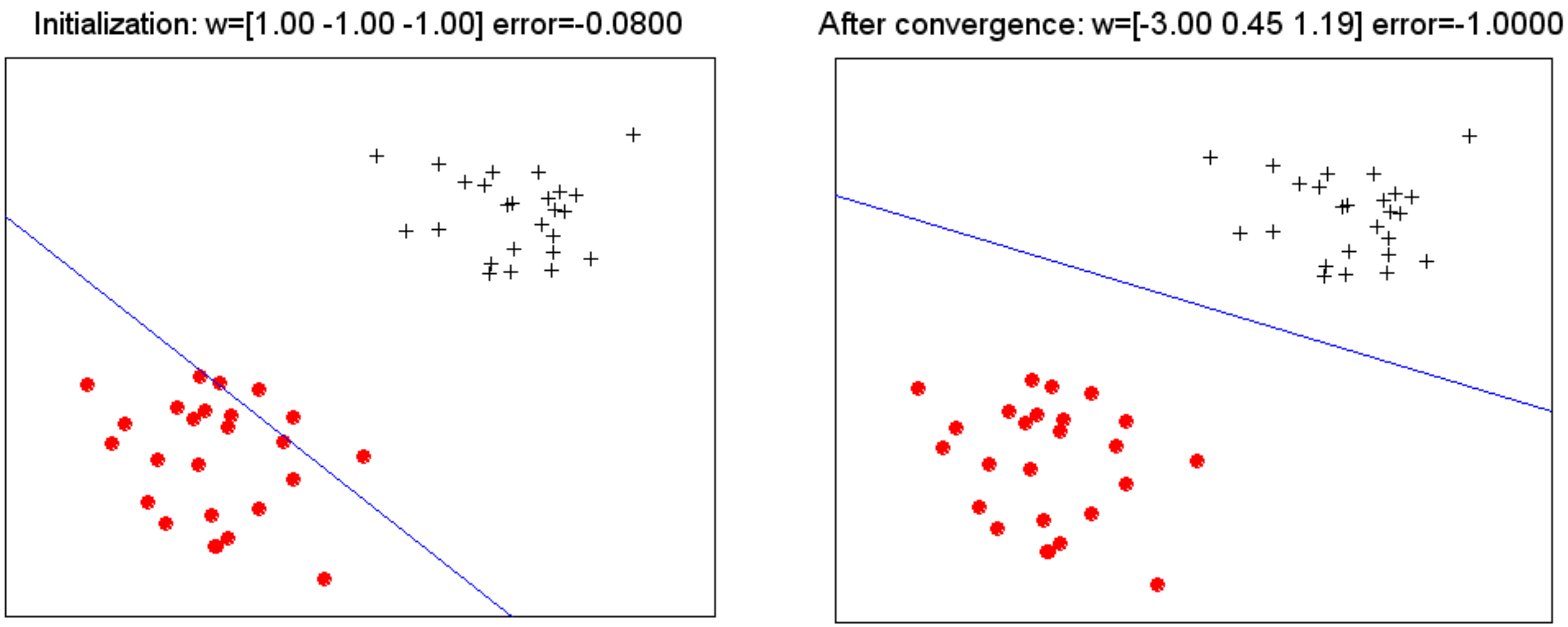


Problem with Perceptron



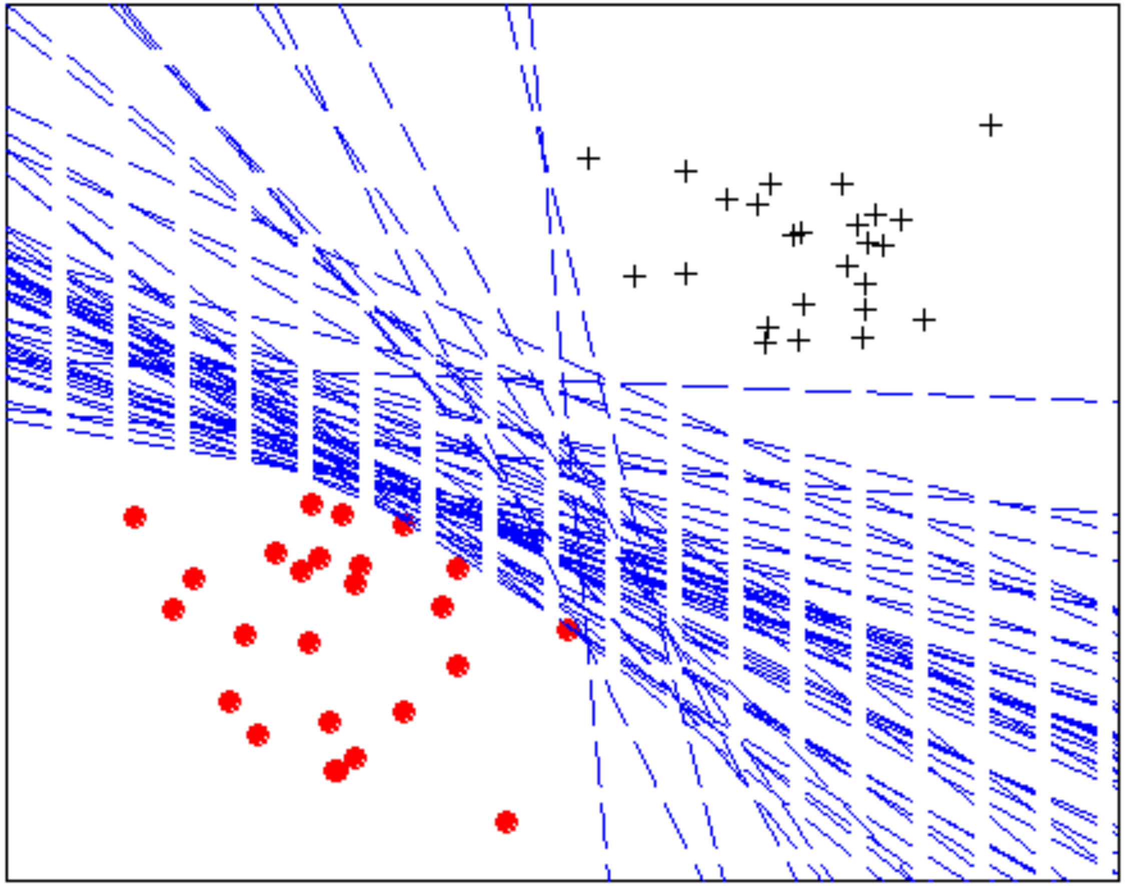
- One Possible Solution (for some initial ω)

Problem with Perceptron



- One Possible Solution (for some initial ω)

Problem with Perceptron



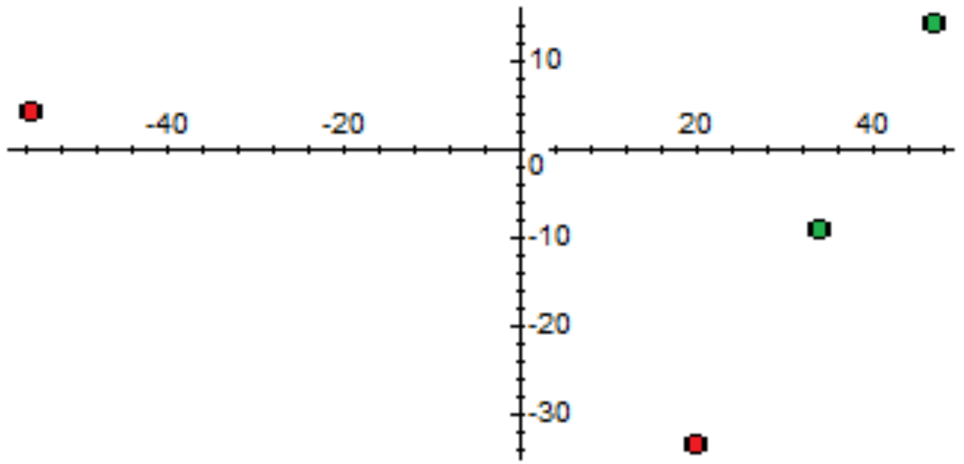
- Other possible solutions (depending on how ω is initialized)

Application: Stock Prediction

Symbol	%Change Aug-Sept	Returns Sept.	Returns Oct.	
ABC	34	-9	6	U
XYZ	-56	4	-11	D
PQ	20	-34	-8	D
ST	47	15	18	U

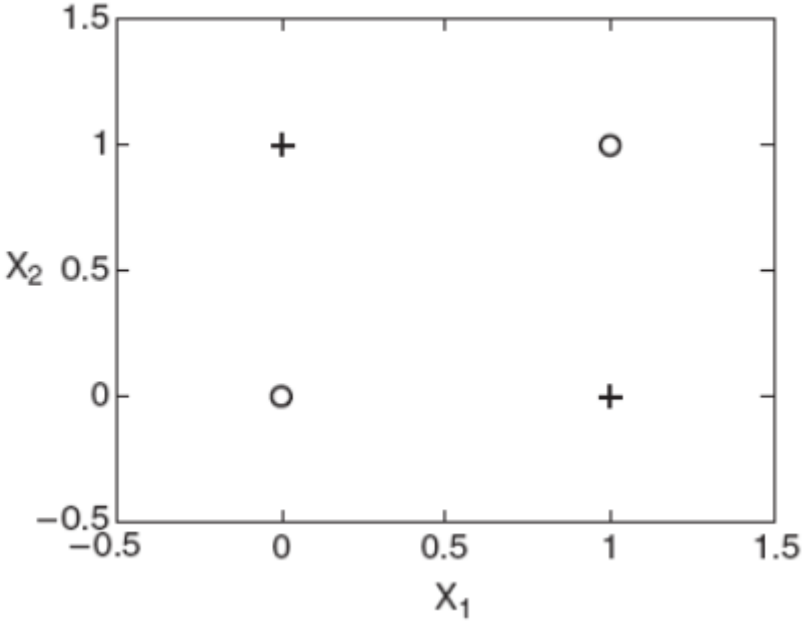
features to use

labels



Nonlinear Decision Boundary

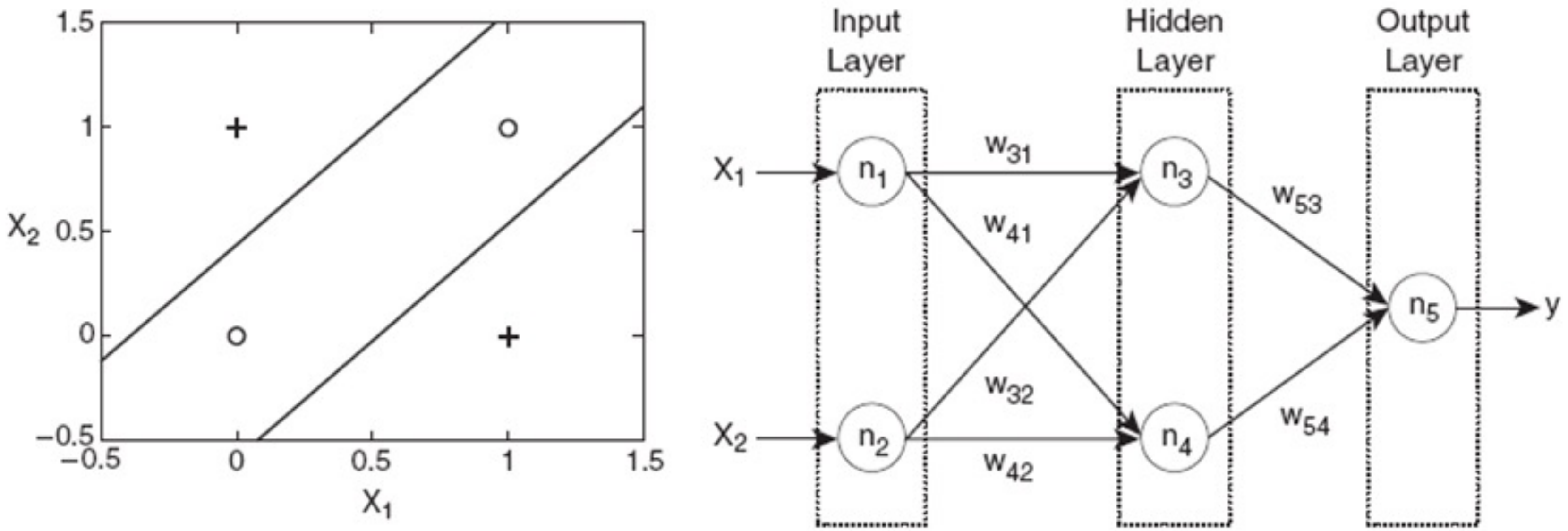
X_1	X_2	y
0	0	–1
1	0	1
0	1	1
1	1	–1



The learning algorithm is guaranteed to converge for linearly separable classification problems.
If the problem is not linearly separable, the algorithm may not converge

Exclusive OR (XOR) Example

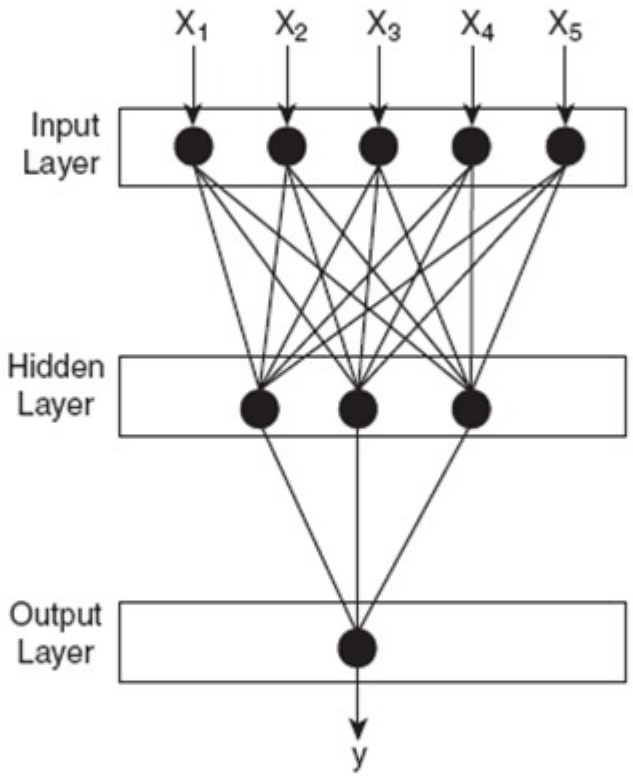
In the binary form:
- $0 \oplus 0 = 0$
- $0 \oplus 1 = 1$
- $1 \oplus 0 = 1$
- $1 \oplus 1 = 0$



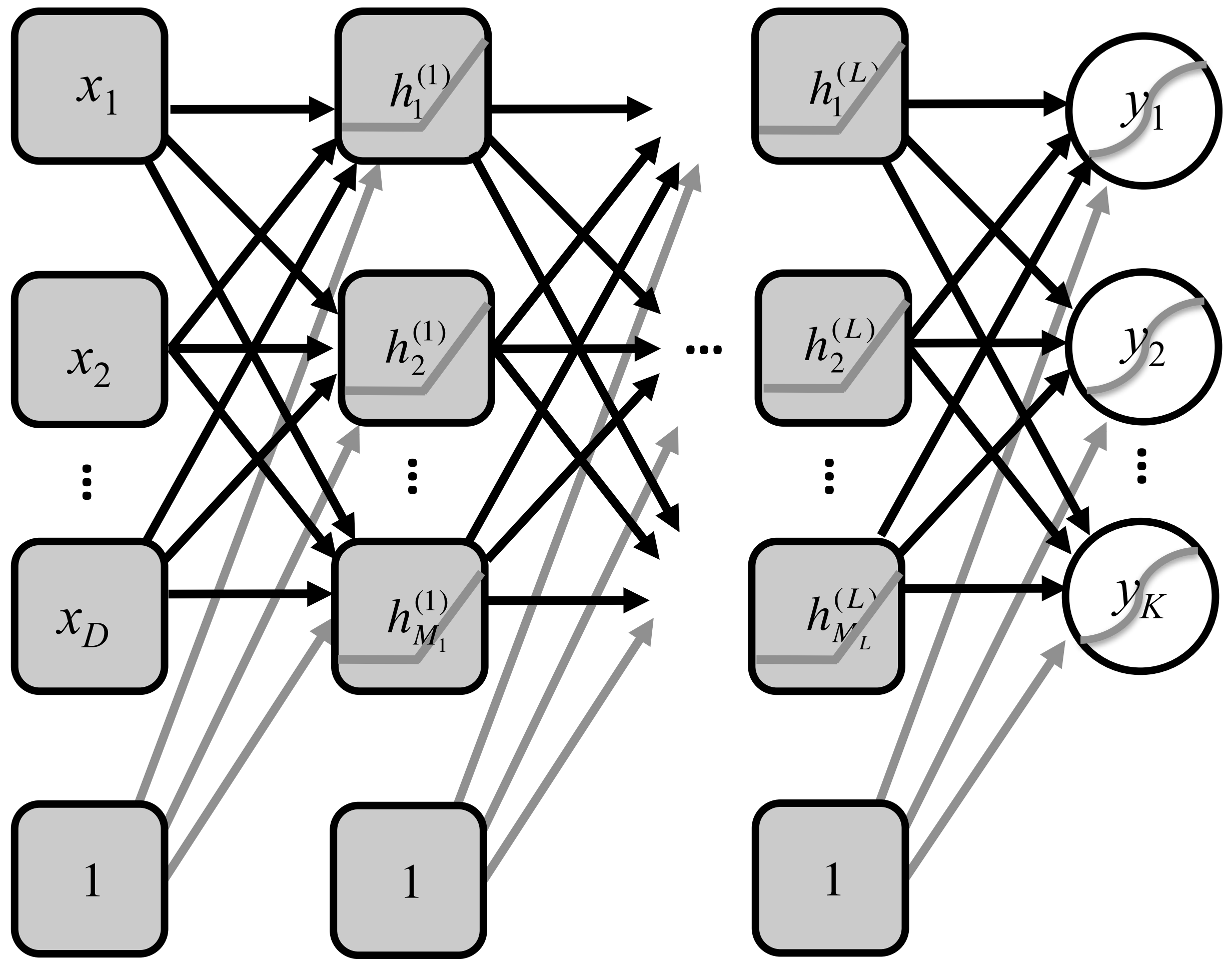
(a) Decision boundary. (b) Neural network topology.

Multilayer ANN

- The network contains several layers
- Intermediary layers: hidden layers
- Nodes in hidden layers: hidden nodes
- ***Feed Forward ANN***: nodes in one layer are connected to nodes in the next layer only
- ***Recurrent ANN***: nodes additionally connect to nodes in same layer or previous layers



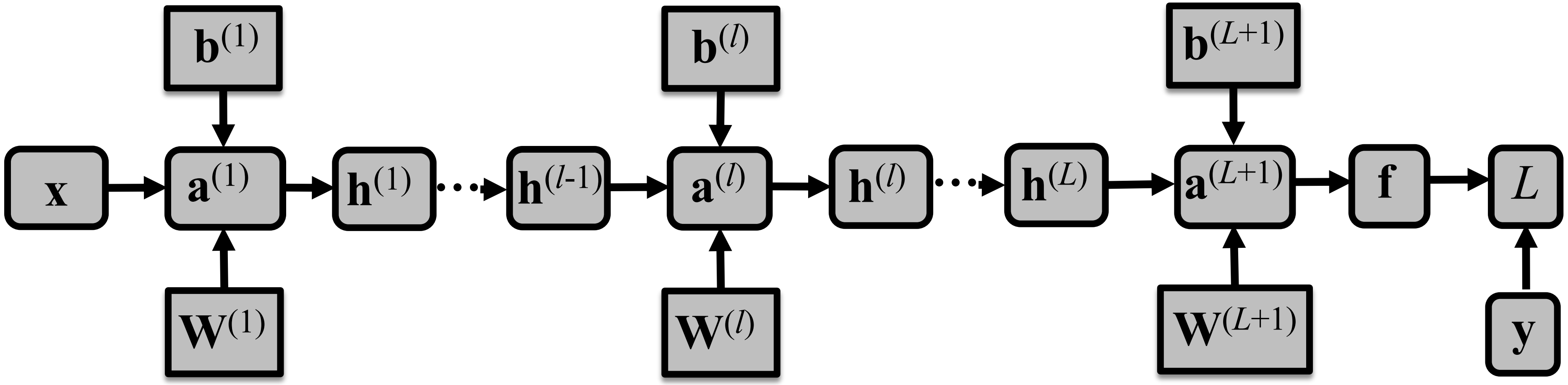
Feed Forward ANN



Feed Forward ANN

$$\mathbf{a}^{(1)} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)} \qquad \mathbf{a}^{(l)} = \mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)} \qquad \mathbf{a}^{(L+1)} = \mathbf{W}^{(L+1)}\mathbf{h}^{(L)} + \mathbf{b}^{(L+1)}$$

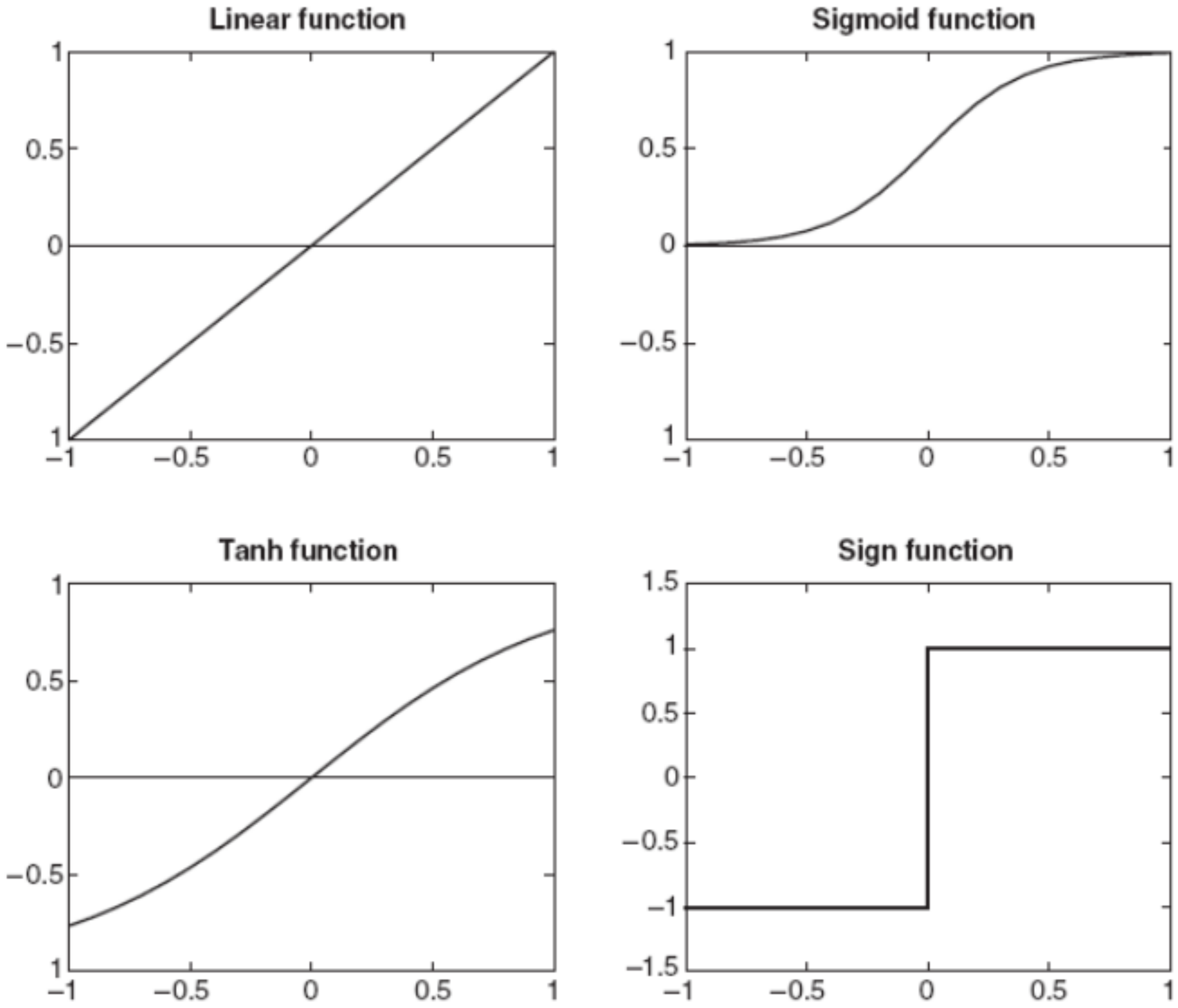
$$\mathbf{h}^{(1)} = \mathrm{act}(\mathbf{a}^{(1)}) \qquad \mathbf{h}^{(l)} = \mathrm{act}(\mathbf{a}^{(l)}) \qquad \mathbf{f} = \mathrm{out}(\mathbf{a}^{(L+1)})$$

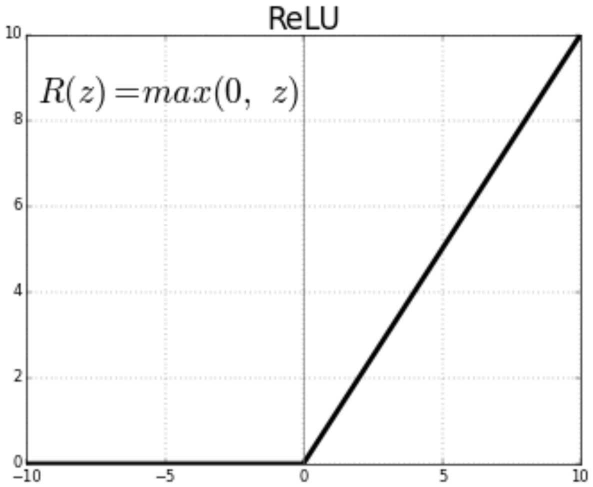


- Hidden nodes have 2 functions:
  - Pre-activation $a^{(i)}$
  - Activation h

Multilayer ANN

- Nodes may use **activation functions** other than the sign function





Model Learning

- Goal: find set of weights w that minimizes the error

$$E(w) = \frac{1}{2}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

- $\hat{y}_i$ is a function of w
- Output of ANN (y) is nonlinear => difficult to optimize
- Greedy algorithms:
  - Gradient descent efficient solution
  - Weight update formula dependent on algorithm

Design Issues

- Determine the structure of the network
  - Number of nodes in the input layer:

    one input node for each attribute

    transform categorical into binary: one input node per value
  - Number of nodes in the output layer

    1 node for a two class problem

    k nodes for a k-class problem
  - The network topology: number of hidden layers, hidden nodes, links
- Initialize the weights and bias parameters, usually at random
- Training example with missing values should be removed or estimated

Implementation – Type of Data Mining

What is the output variable?

- Real-valued (Regression)
  - MLPRegressor
  - Squared error

- Categorical (Classification)
  - MLPClassifier
  - Softmax

Loss Name, $L(f_i(\mathbf{x}_i;\theta),\mathbf{y}_i) =$
Squared error, $\sum_{k=1}^{K}(f_k(\mathbf{x})-y_k)^2$ Cross entropy, $-\sum_{k=1}^{K}[y_k\log f_k(\mathbf{x}) + (1-y_k)\log(1-f_k(\mathbf{x}))]$ Softmax, $-\sum_{k=1}^{K} y_k \log f_k(\mathbf{x})$

Implementation – Multi-layer Perceptron

class sklearn.neural_network. **MLPClassifier** (*hidden_layer_sizes=(100,), activation='relu', solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08*) [source]

Multi-layer Perceptron classifier.

This model optimizes the log-loss function using LBFGS or stochastic gradient descent.

New in version 0.18.

Parameters:

hidden_layer_sizes : tuple, length = n_layers - 2, default (100,)

The ith element represents the number of neurons in the ith hidden layer.

activation : {'identity', 'logistic', 'tanh', 'relu'}, default 'relu'

Activation function for the hidden layer.

- 'identity', no-op activation, useful to implement linear bottleneck, returns f(x) = x
- 'logistic', the logistic sigmoid function, returns f(x) = 1 / (1 + exp(-x)).
- 'tanh', the hyperbolic tan function, returns f(x) = tanh(x).
- 'relu', the rectified linear unit function, returns f(x) = max(0, x)

solver : {'lbfgs', 'sgd', 'adam'}, default 'adam'

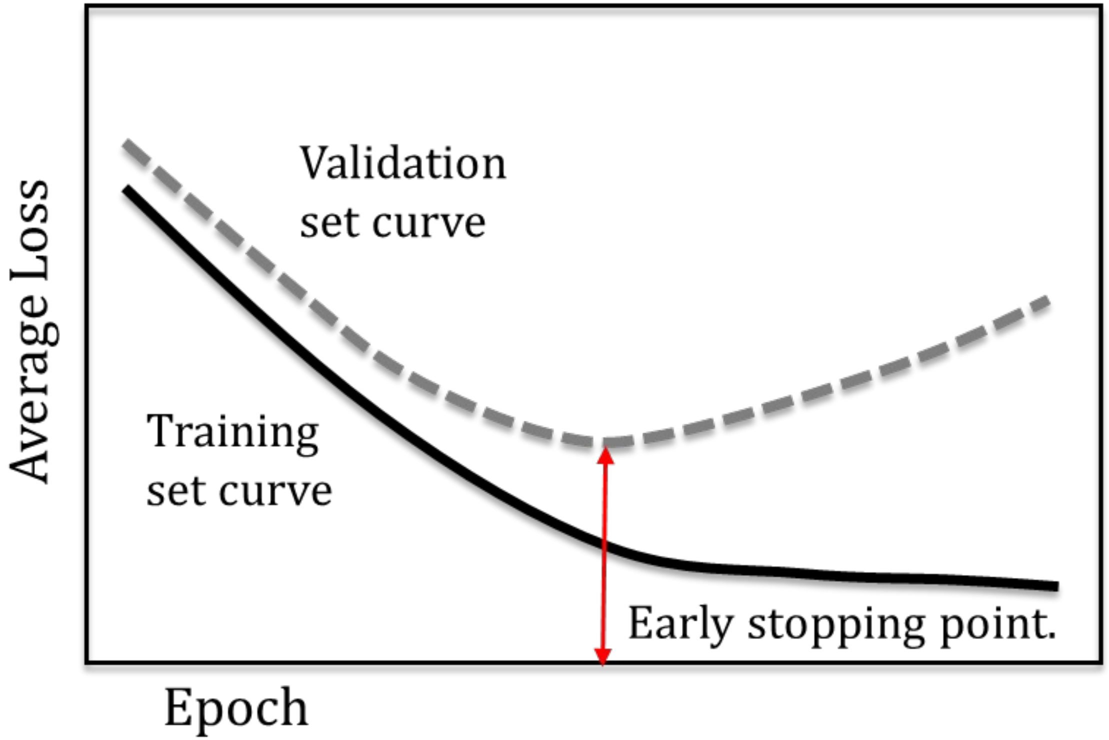
The solver for weight optimization.

- 'lbfgs' is an optimizer in the family of quasi-Newton methods.
- 'sgd' refers to stochastic gradient descent.
- 'adam' refers to a stochastic gradient-based optimizer proposed by Kingma, Diederik,

Parameters:

- hidden_layer_sizes
- activation
- max_iter
- early_stopping

Early Stopping



- In practice the curves above can be more noisy due to the use of stochastic gradient descent
- As such, it is common to keep the history of the validation set curve when looking for the minimum
  – even if it goes back up it might come back down

Characteristics of Neural Networks

- Multilayer neural networks with at least one hidden layer are universal approximators:
  - Can approximate any function
  - May suffer from overfitting
- Can handle redundant features
- Sensitive to noise
- Training is time consuming
- Classifying a test example is fast
- Hard to interpret

Deep Learning

1943

The first mathematical model mimicking human brain cells

1960

The first attempt on back propagation

1982

The first CNN

2000

Graphics Processing Units (GPUs)

2009

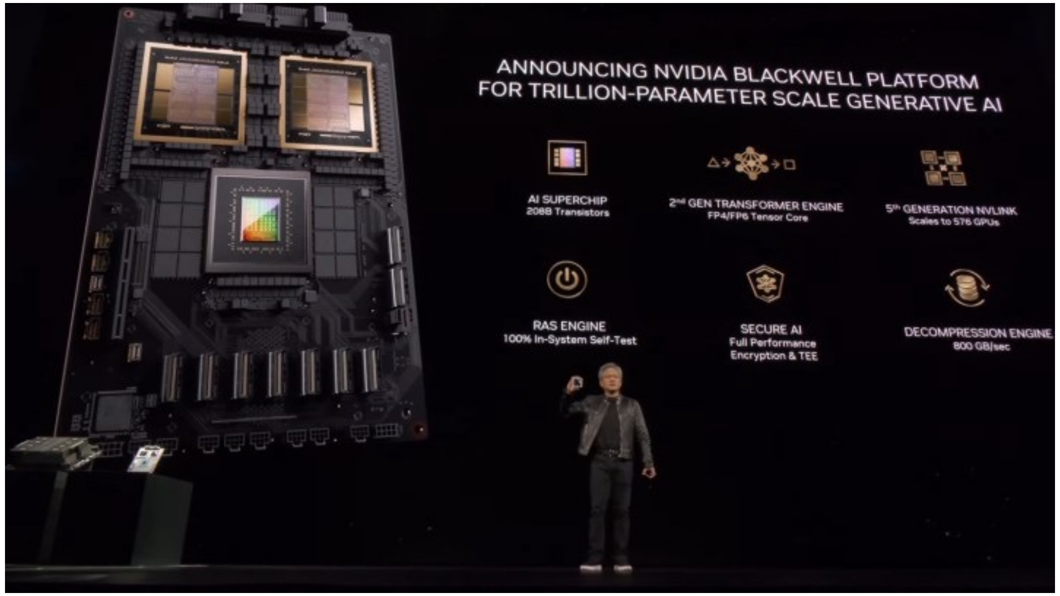
ImageNet,

a free database of more than 14 million labeled images

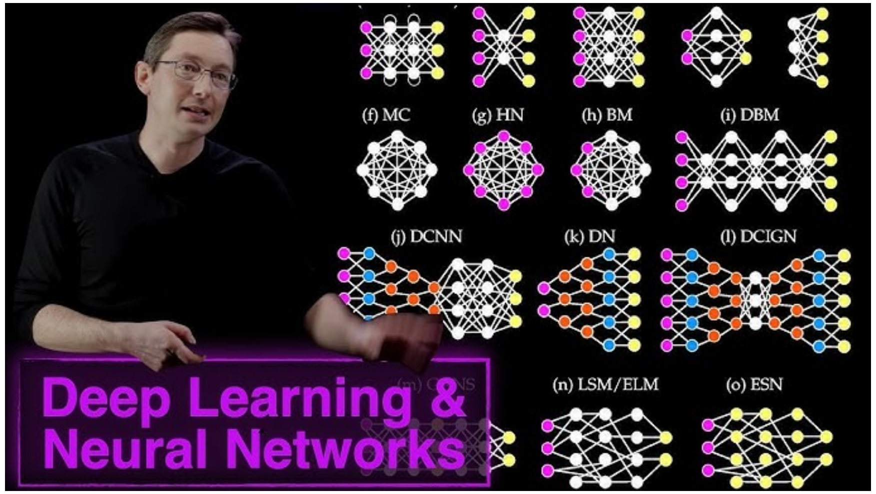
2012

The Era of Modern Deep Learning

Why Deep Learning Now?



GPUS



Algorithms

Big Data

Deep Learning Pros and Cons

✓ **Deep learning has led to revolutionary progresses in many applications**

Computer vision; natural language processing; autonomous driving; time-series forecasting; data mining

✗ **Low data efficiency**

Requires a tremendous amount of training data and their annotations

[Aggarwal,2018; Marcus, 2018]

✗ **Poor cross-dataset generalization**

The extracted patterns are data-specific, applying only to scenarios captured by training data

[Neyshabur, Behnam, et al, 2017; Kawaguchi, K., Kaelbling, L.P. and Bengio, Y., 2017]

✗ **Lack of Interpretability**

The extracted patterns represented as hidden features can not be well interpretated

[Zhang, Q.S. and Zhu, S.C., 2018; Chakraborty, Supriyo, et al, 2017]

Deep Learning = Learning Representations

- Traditional model of pattern recognition: fixed/hand-engineered features + trainable classifier

Head-crafted Feature Extractor

“Simple” Trainable Classifier

- End-to-end Learning/feature learning/deep learning: trainable features + trainable classifier

Trainable Feature Extractor

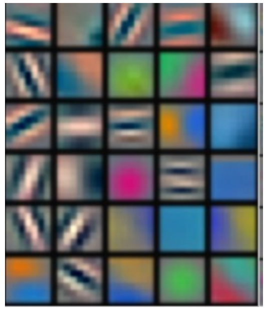
Trainable Classifier

Deep Learning = Learning Representations

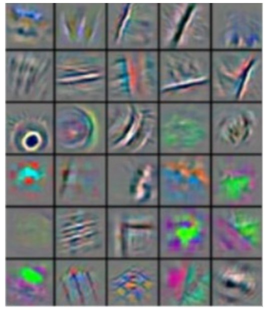
- Deep architecture: learn hierarchical representations

Trainable Feature Extractor → Trainable Classifier

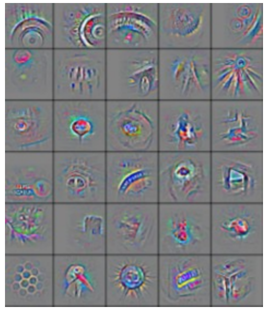
Low level feature



Mid level feature



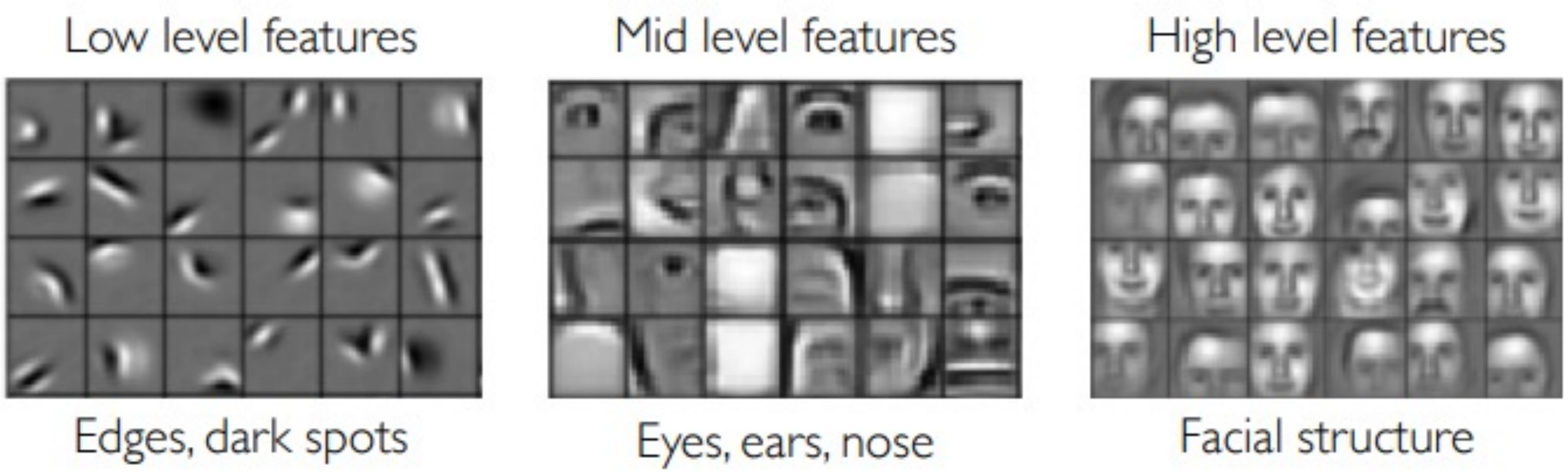
High level feature



More than one stage of non-linear feature extraction

Trainable Feature Hierarchies

- A hierarchy of trainable feature transforms
  - Each module transforms its input representation into a higher-level representation
  - High-level features are more global and more invariant
  - Low-level features are shared among categories



- Deep learning Goal: make all modules trainable and get them to learn appropriate representations

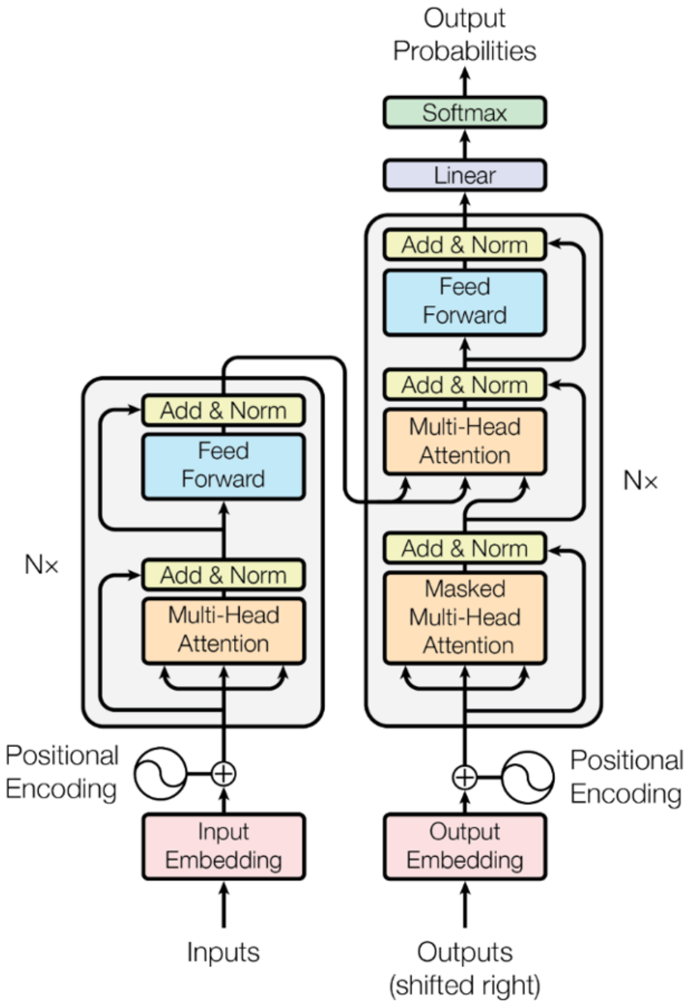
Deep Learning

- Algorithm/Architecture
  - MLP
  - Convolutional Neural Networks (CNN)
  - Recurrent Neural Networks (RNN)
  - Graph Neural Networks (GNN)
  - Attention and Transformers
- Training Strategy
  - Supervised
  - Unsupervised – generative learning
  - Reinforcement Learning

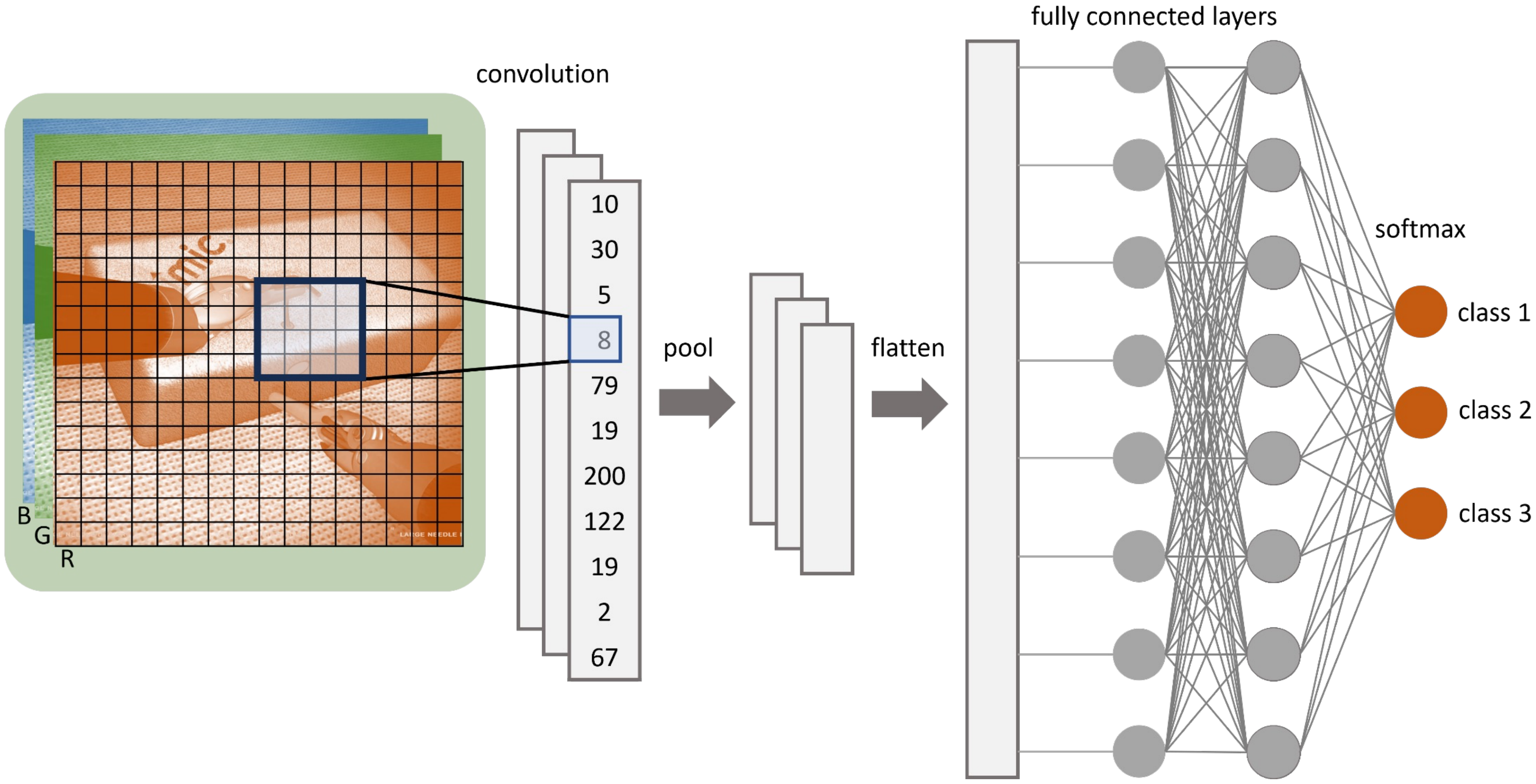
Algorithm/Architecture

- **Convolutional Neural Networks (CNN)**
  - Specialized for image processing tasks, where spatial hierarchies are important.
  - Examples: AlexNet/VGG19/ ResNet50
- **Recurrent Neural Networks (RNN)**
  - Best for sequential data such as time series or text. Uses feedback connections to retain memory of previous inputs.
  - Example: LSTM
- **Graph Neural Networks (GNN)**
  - Work directly with graph structures (e.g., social networks, molecular structures); Useful in tasks where relationships between elements are important, such as node classification or link prediction.
  - Example: GCN; GAT
- **Attention Mechanism**
  - Improve the performance of models that deal with sequential data; allows the model to focus on different parts of the input sequence when making predictions, rather than treating all inputs equally.
- **Transformer**
  - utilize attention to model relationships between all elements in a sequence simultaneously

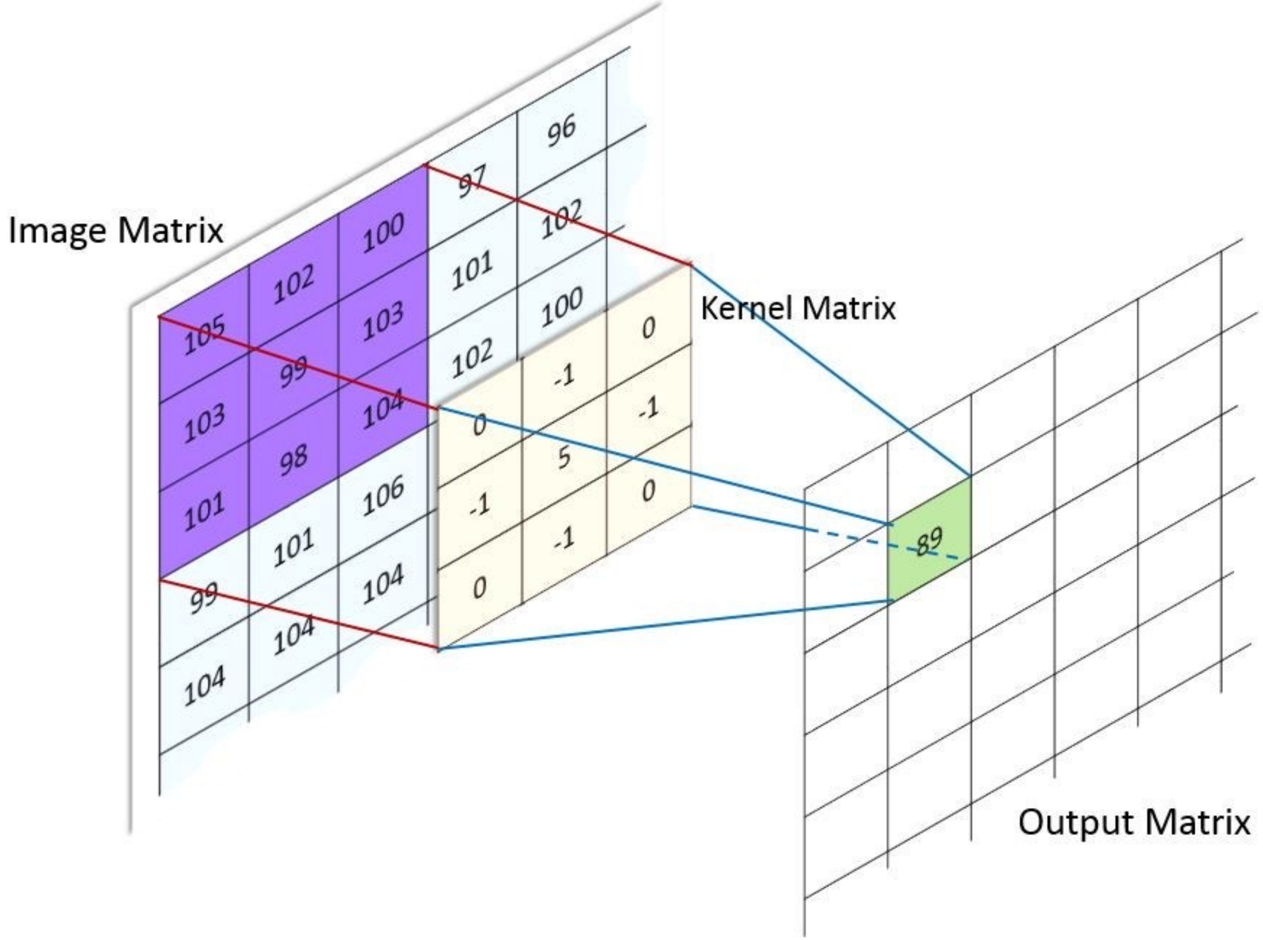
Transformers

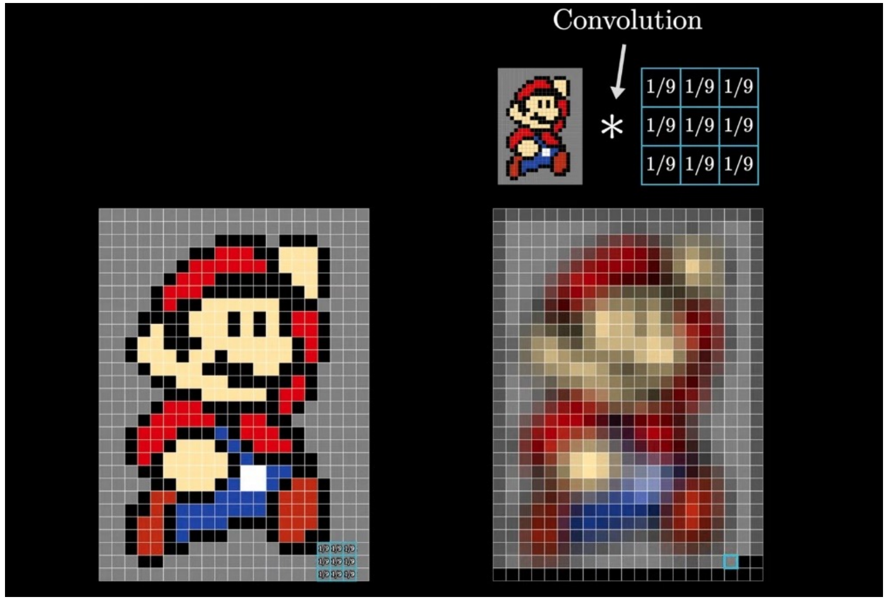


Convolutional Neural Networks (CNN)



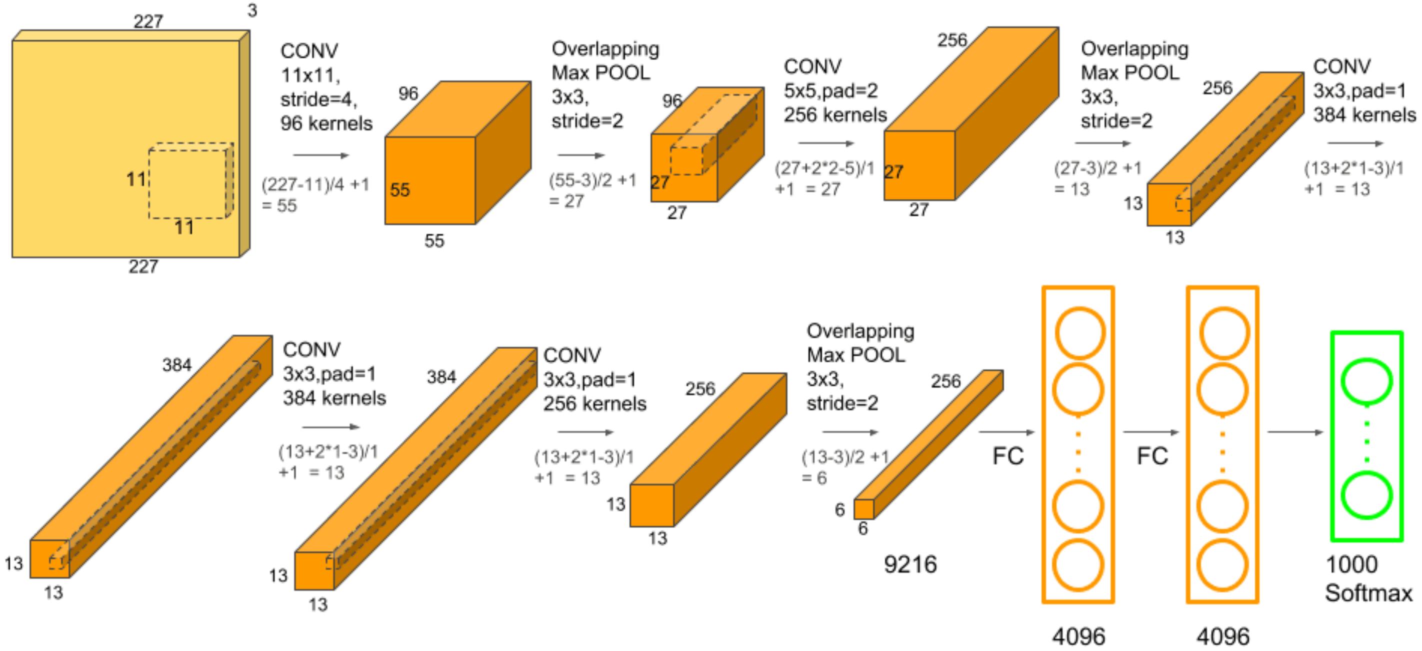
Convolutional Operation





Examples

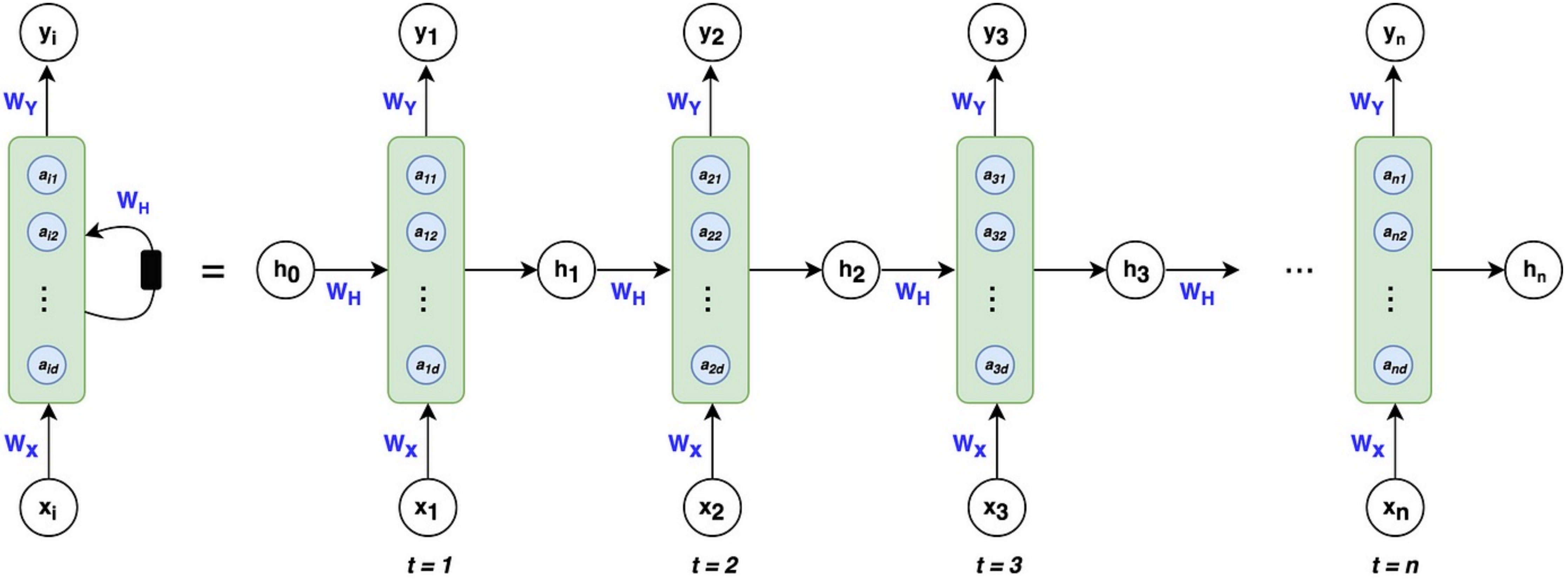
- Alexnet explanation and implementation in Tensorflow



- Deeper models:

Comparison					
Network	Year	Salient Feature	top5 accuracy	Parameters	FLOP
AlexNet	2012	Deeper	84.70%	62M	1.5B
VGGNet	2014	Fixed-size kernels	92.30%	138M	19.6B
Inception	2014	Wider - Parallel kernels	93.30%	6.4M	2B
ResNet-152	2015	Shortcut connections	95.51%	60.3M	11B

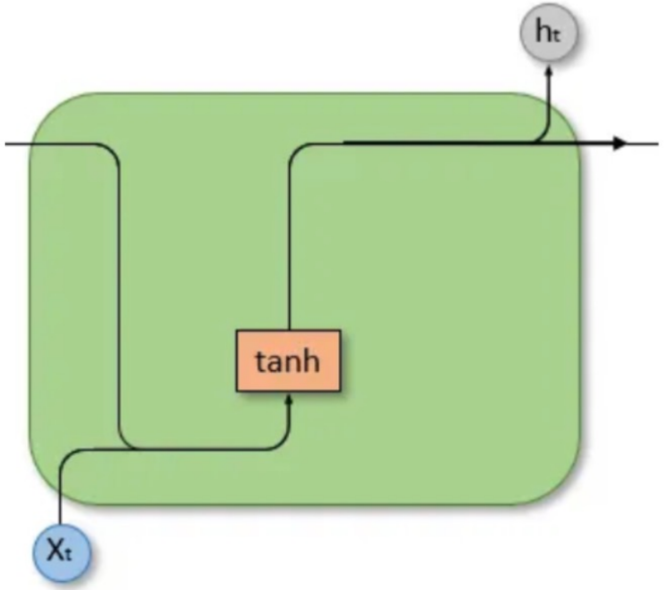
Recurrent Neural Network (RNN)



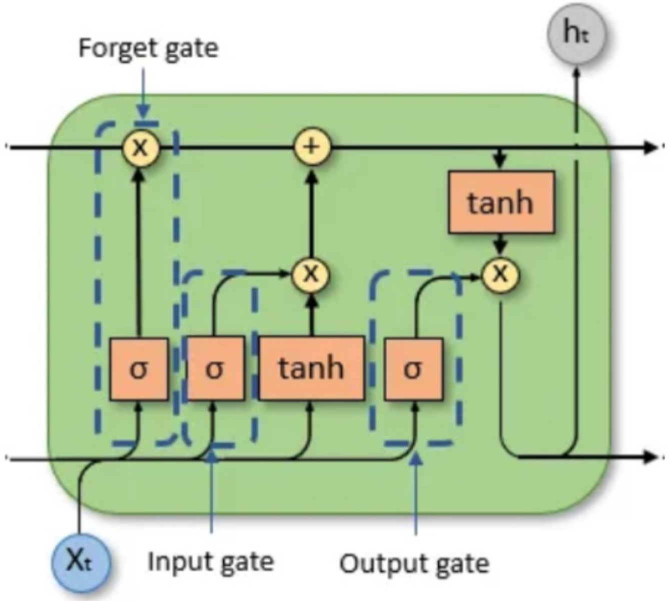
- Best for **sequential data** such as time series or text. Uses feedback connections to retain memory of previous inputs.

Comparisons

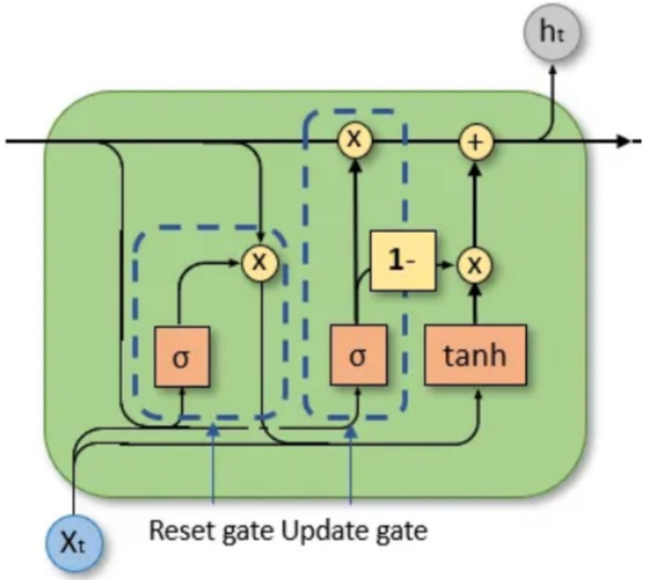
RNN



LSTM

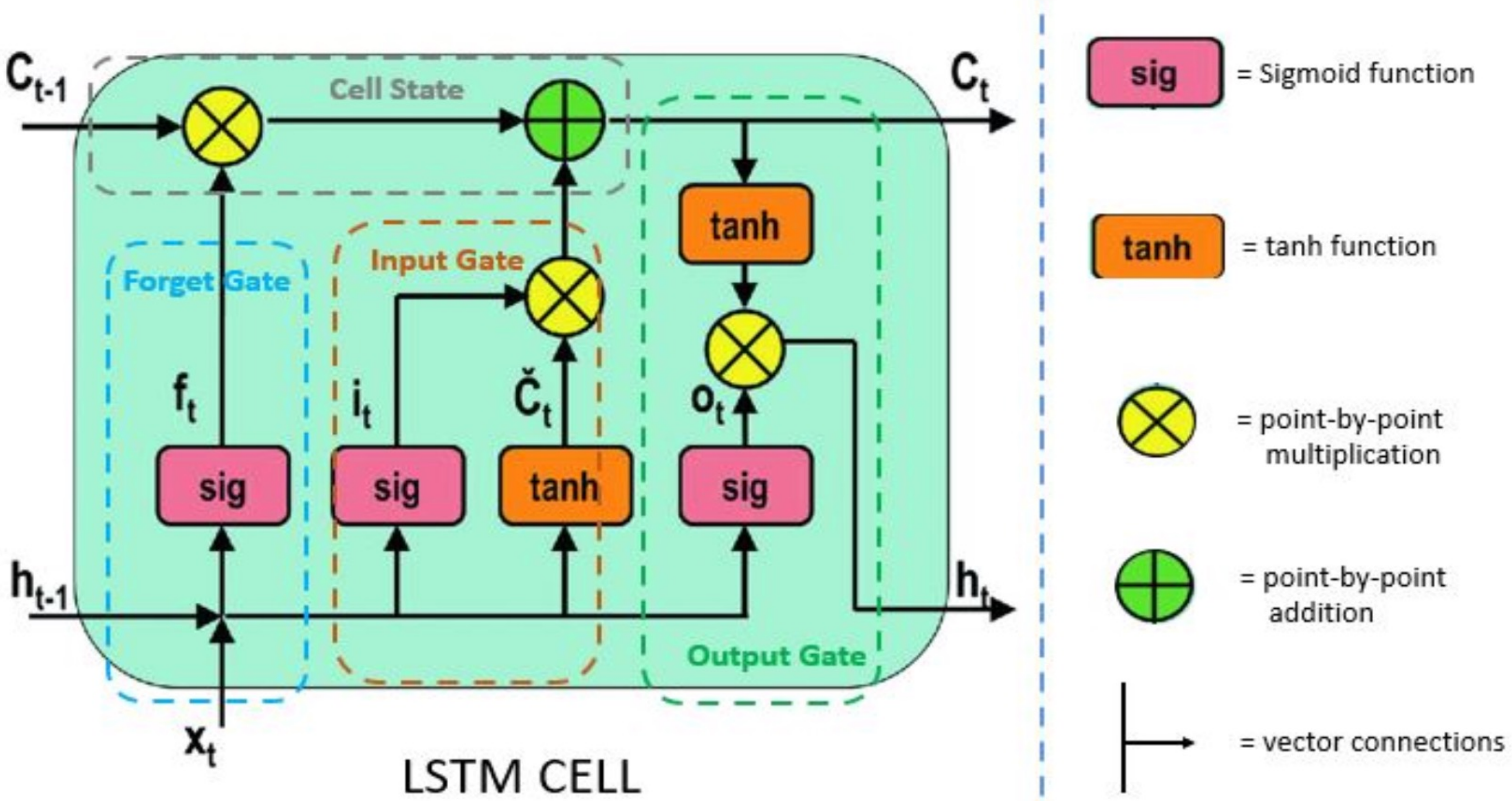


GRU

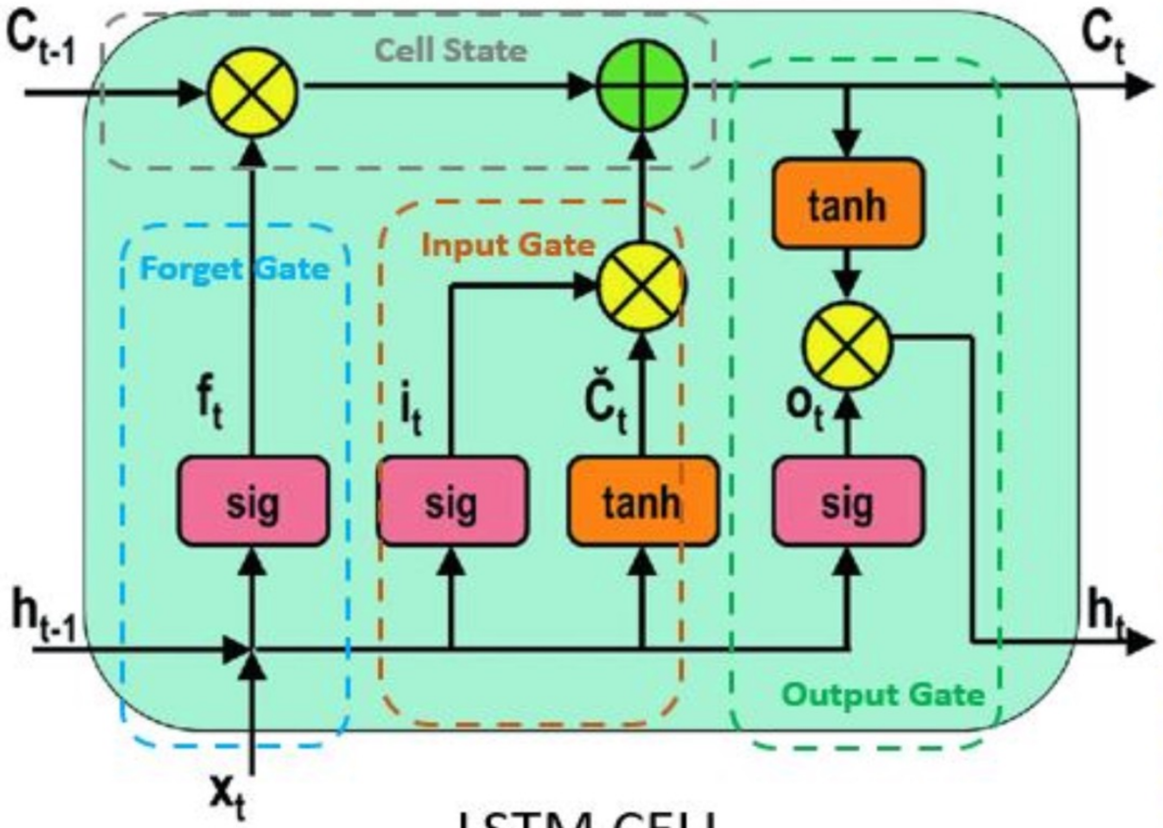


LSTM

- Long short-term memory network



LSTM



LSTM CELL

Forget Gate

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Input Gate

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Cell State

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Output Gate

$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$