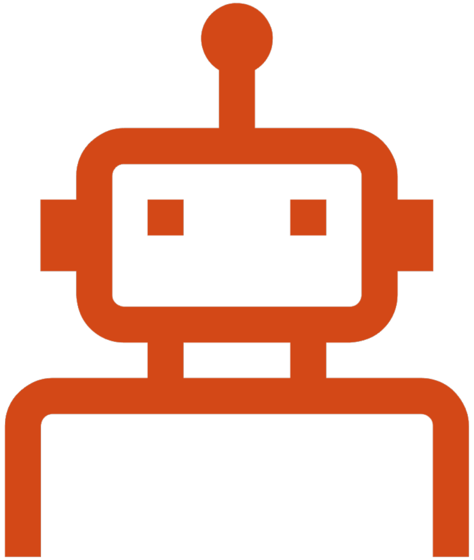
MACHINE LEARNING-BASED METHODS FOR CLASSIFICATION

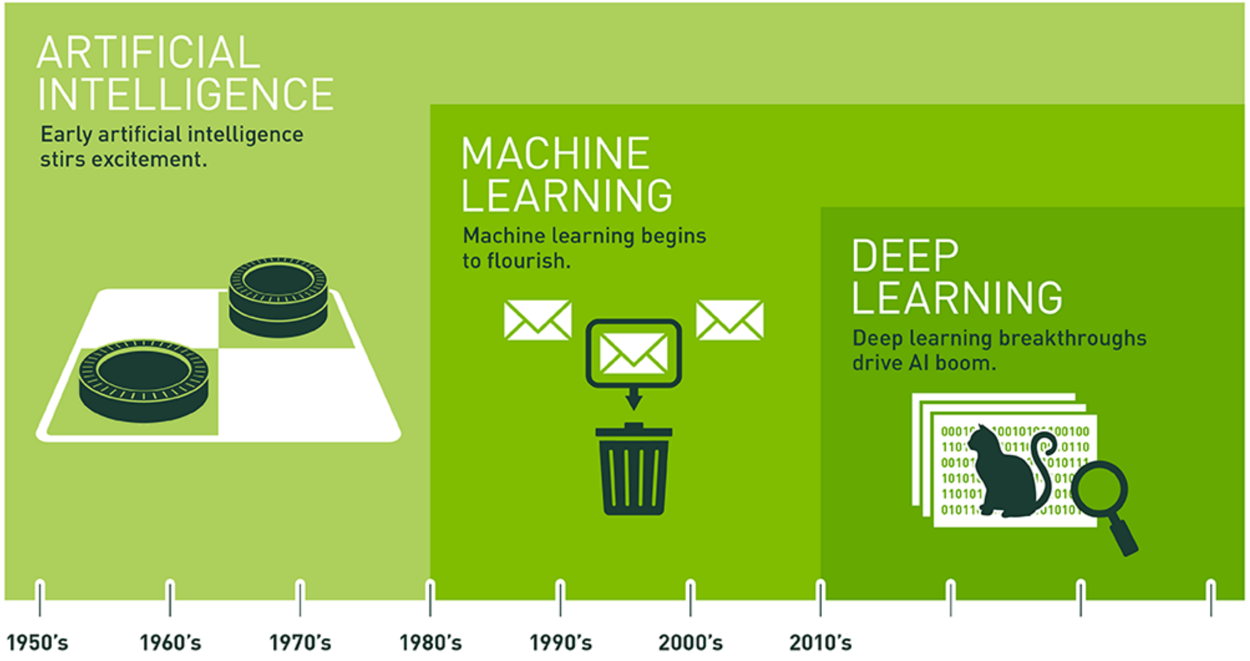
By Linh T. Hoang
Aizu, September 2020

CONTENTS

- Formulation of Machine learning (ML)
- Classification of ML methods
  - Based on learning style
  - Based on function
- Instance-based ML methods
  - K-nearest Neighbor (KNN) classifier
  - Learning Vector Quantization (LVQ) classifier
  - Numerical results on Iris flower dataset
- Neural networks
  - Multi-layer Perceptron (MLP)
    - Learning of MLP : backpropagation
  - Numerical results
- Conclusions



FORMULATION OF MACHINE LEARNING



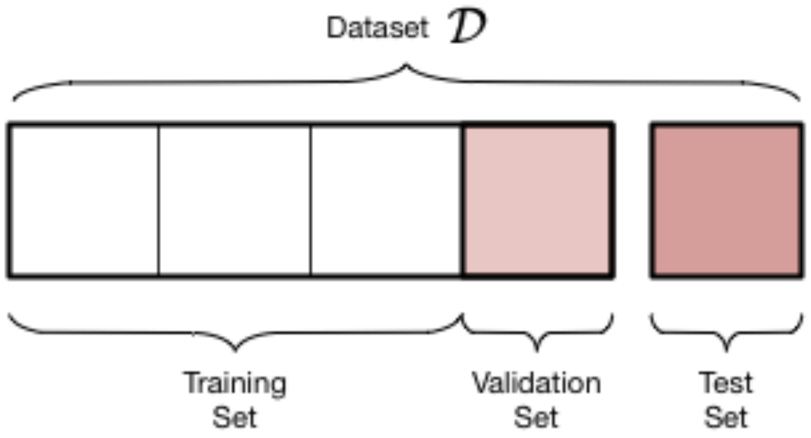
Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.

Source: https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/

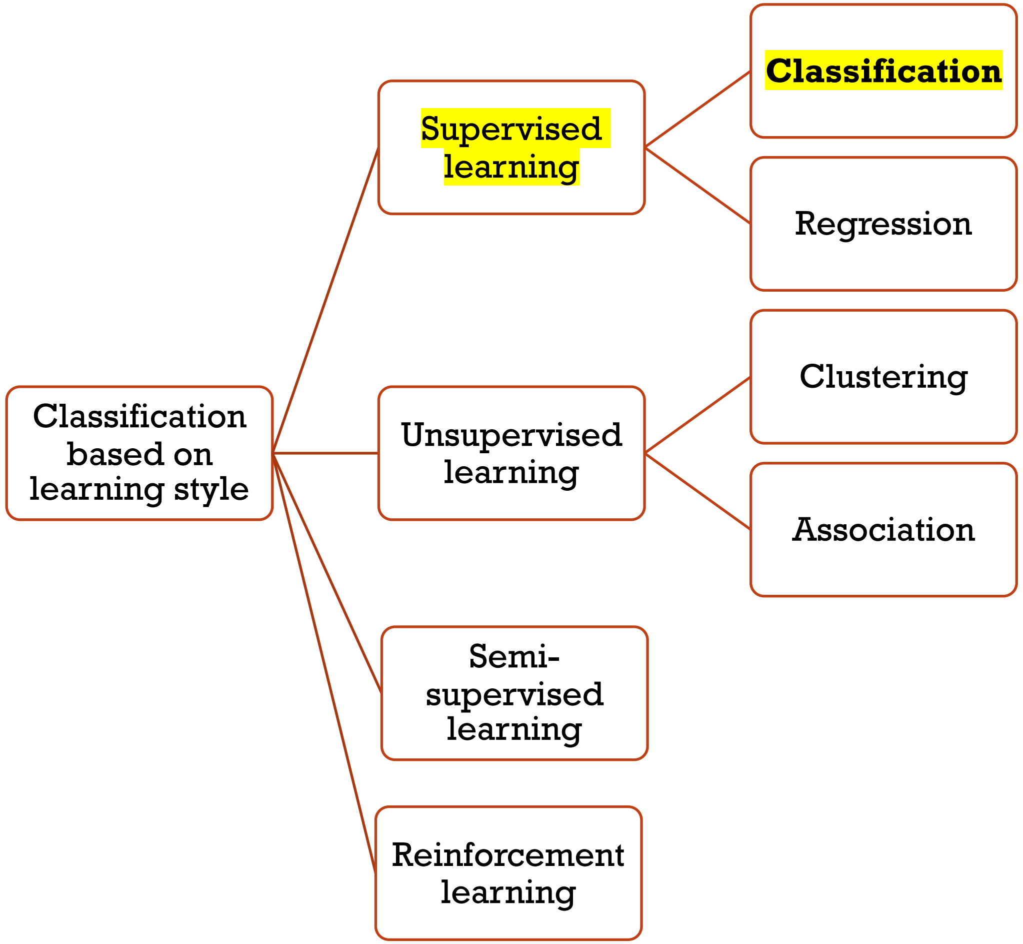
- **Machine learning** : *the subfield of computer science that "gives computers the ability to* ***learn without being explicitly programmed****"* – Wikipedia
- **Deep learning** (aka **deep structured learning**) : *a part of the broader family of machine learning methods* ***based on artificial neural networks*** – Wikipedia

FORMULATION OF MACHINE LEARNING (CONT.)

- Observation : the input of a model, **x** (in bold)
  - **x** = (x1, x2, … xn) : a feature vector
  - xi : a feature, i =1, 2, … n
- Label : the outcome of a model, y
  - y : can be a scalar (real numbers/integers) or a vector
- Model : a function (or a hypotheis), f(**x**) = y
- Parameters and hyper-parameters
  - **x** = (x1, x2)
  - f(**x**) = ax1^2 + bx2 + c
  - Parameters : (a, b, c)
  - Hyper-parameter :
    the degree of the polynomial f(**x**), i.e. 2
- Learning : the process of finding a model f(**x**) that can predict the labels (y) of <u>unseen observations</u> (**x**) in the test set <u>correctly in most cases.</u>



CLASSIFICATION OF ML METHODS (1/2)



Supervised : predict label(s) of a new input datapoint based on pairs of (input, label) in the training set.

- **Classification** : # of labels is finite.
  Eg: given a human face, detect whether he/she is a man/woman
- **Regression** : labels are continuous.
  Eg: given a human face, detect his/her age

Unsupervised : input datapoints are given without labels.

- Clustering : (eg) catergorize customers based on their purchasing behaviors.
- Association : (eg) recommendation system (if someone likes "Spider man" -> likely he/she also likes "Batman")

Semi-supervised : only a proportion of training datapoints are with labels.

Reinforcement : (target) decide which action should be taken based on particular situations to maximize the cumulative reward.
Eg: how to play Mario game to get the highest score

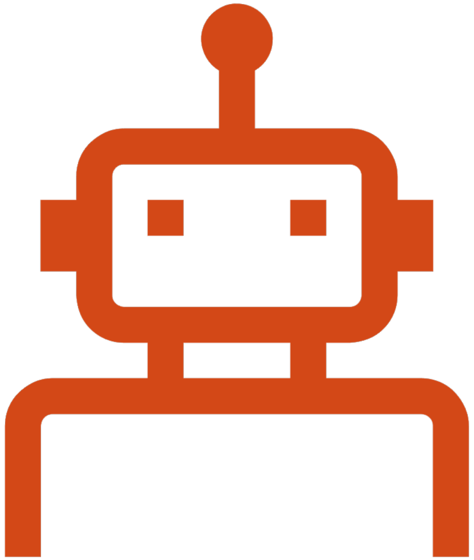
CLASSIFICATION OF ML METHODS (2/2)

Classification based on function

Regression	Linear regression, Logistic regression, …
Classification	Linear classification, Support vector machine (SVM), Kernel SVM, …
Instance-based algorithm	K-Nearest Neighbor (KNN), Learning Vector Quantization (LVQ)
Bayesian algorithm	Naiive Bayes, Gaussian Naiive Bayes, …
Clustering	K-means clusterring, K-medians, …
Neural network	Perceptron, Multi-layer perceptron (MLP), Convolutional Neural Network (CNN)
Dimensionality reduction	Principle Component Analysis (PCA), Linear Discriminant Analysis (LDA)
Ensemble algorithm	Bagging (**B**ootstrap **Agg**regat**ing**), AdaBoost, Random forest,…
…	

CONTENTS

- Formulation of Machine learning (ML)
- Classification of ML methods
  - Based on learning style
  - Based on function
- Instance-based ML methods
  - K-nearest Neighbor (KNN) classifier
  - Learning Vector Quantization (LVQ) classifier
  - Numerical results on Iris flower dataset
- Neural networks
  - Multi-layer Perceptron (MLP)
    - Learning of MLP : backpropagation
  - Numerical results
- Conclusions



K-NEAREST NEIGHBOR CLASSIFIER (KNN)

K-nearest neighbor classifier (KNN):

- If k=3 (solid line circle): the green dot is assigned to the red triangles
- If k=5 (dashed line circle): the green dot is assigned to the blue squares

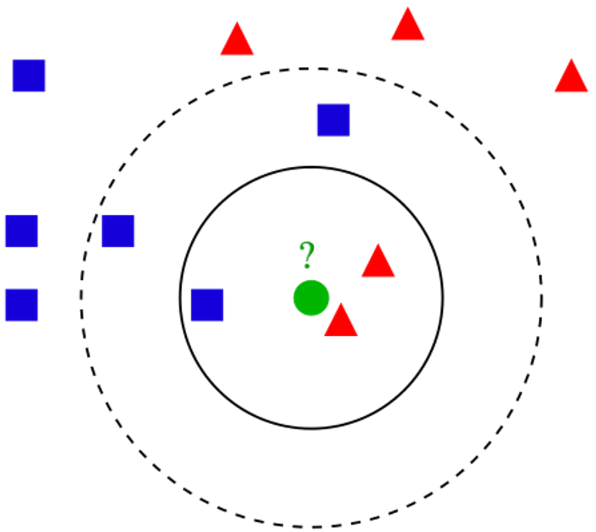
For k =1 : Nearest neighbor classifier

label (x) = label (p) if

$$p = \arg\min_{q \in \Omega} ||x - q||_2$$

x: a new datapoint
p: a datapoint in the training set (Ω)
||.|| : Euclidean distance (2-norm)

$$\|\mathbf{x}-\mathbf{q}\| = \sqrt{\sum_{j=1}^{n}(x_j - q_j)^2}$$



Example of *k*-NN classification from Wikipedia

- Does not require training process
- But **requires long time for testing** (since the entire training set is used to make predictions)

To reduce the computational cost
→ use representative(s) for each class.

LEARNING VECTOR QUANTIZATION (LVQ)

LVQ : an algorithm to find the representatives

Pseudo-code

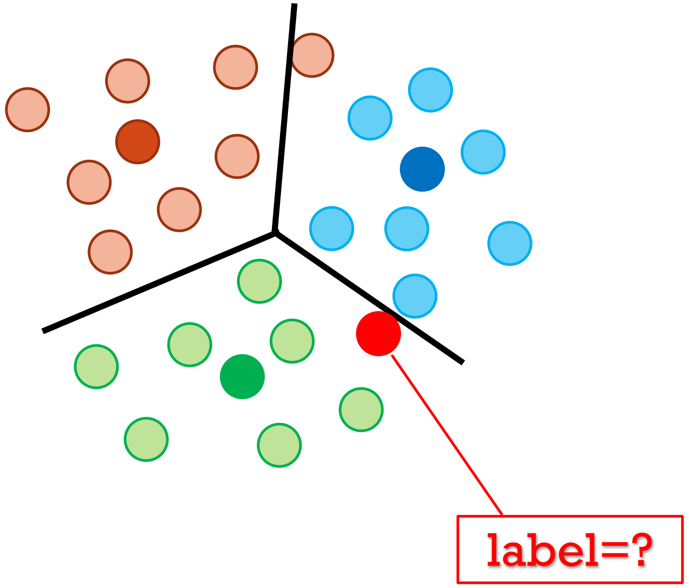
```
# Initialize the prototype set
>> Randomly create n_prototype prototypes
# Train the prototype set
>> Choose n_epoch          # no. of training cycles
>> Choose lrate_init       # initial learning rate
>> For i_epoch runs from 1 to n_epoch :
       lrate = lrate_init * (1-i_epoch/n_epoch)
       Initiate sum_err = 0
       For each datapoint x in the training set :
           Find the nearest neighbor p of x from the prototype set
           sum_err += alpha*||x-p||^2
           If label(x) == label(p):
               Set p += lrate * (x-p)
           Elseif label(x) != label(p):
               Set p -= lrate * (x-p)
               Re-adjust p so that p in an appropriate range (if
               needed)
>> Use 1NN classifier on the prototype set to label new data
   points in the testing set
```

lrate ↘ gradually

p : the nearest neighbor of **x**

Pull **p** closer to **x**

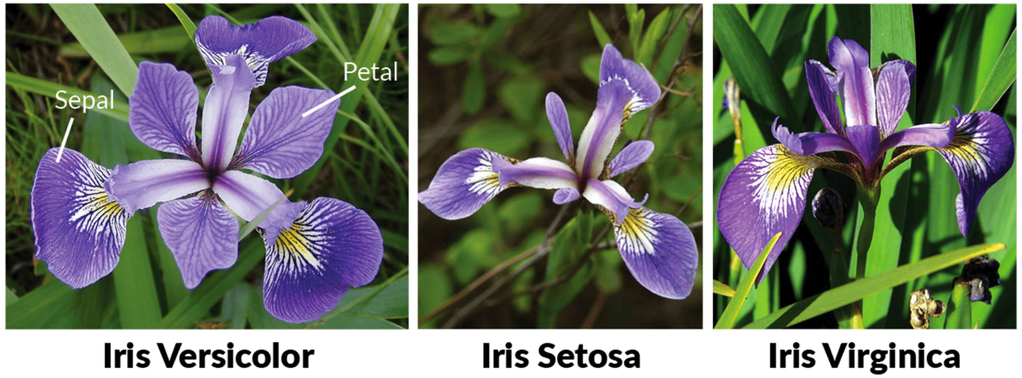
Push **p** away from **x**



Using NNC on the prototype set instead of on the training set

NUMERICAL RESULTS / IRIS DATASET

Iris flower dataset
(from UCI Machine learning repository)



# classes	3
# datapoints	150
# attributes	4 (sepal + petal length and width)

	1NN	LVQ
Avg. accuracy	95.25%	92.87%
Acc. variance	3.02%	10.47%
Avg. train time	--	378.25 (ms)
Avg. test time	34.623 (ms)	8.74 (ms)

Note:
- Train size = 75, test size = 75 (randomly split in each run)
- Averaged over 100 runs
- For LVQ:
  - prototypes = 15 (3 classes)
  - epochs = 30
  - lrate_init = 0.5

NUMERICAL RESULTS / IRIS DATASET (CONT.)

LVQ-based classifier on Iris dataset

Increasing # of prototypes for each class :

- Improves the average accuracy (Fig. 1)
- Improves the stability (Fig. 2)
- At the cost of lengthening both training and testing time (Fig. 3), since more prototypes are used

# of prototypes : a critical hyper-parameter

- should be selected at the **trade-off between (accuracy + stability) and (training + testing time)**.

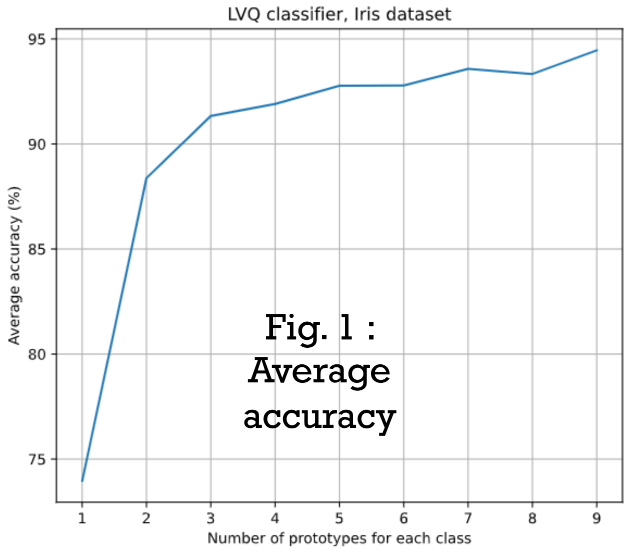


Fig. 1 : Average accuracy

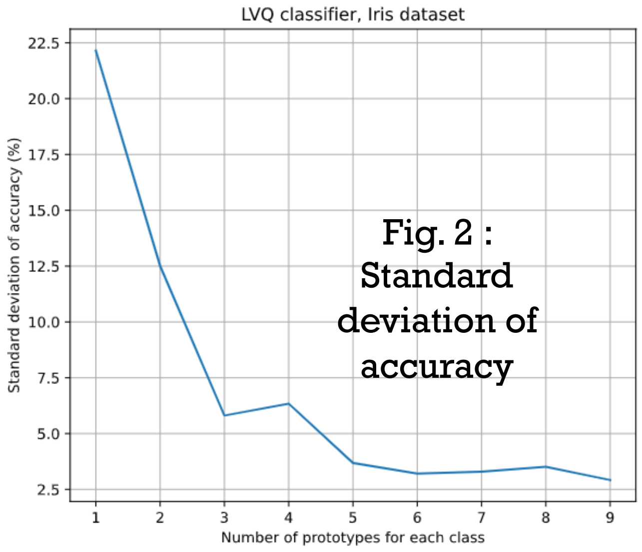


Fig. 2 : Standard deviation of accuracy

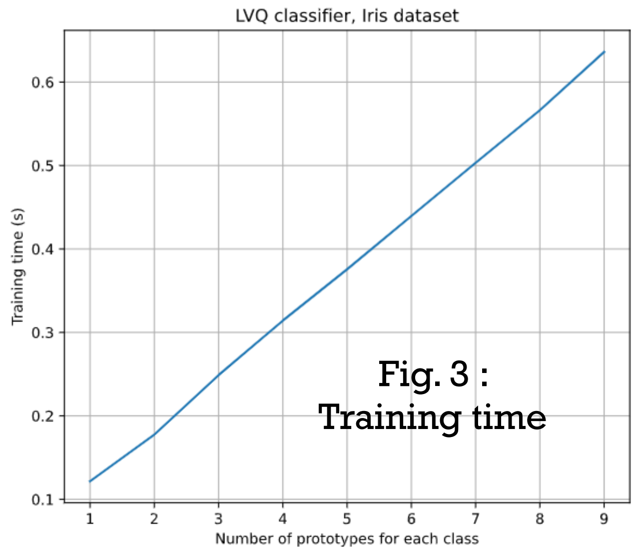


Fig. 3 : Training time

DISCRIMINANT FUNCTIONS

1 Define the representatives for a 2-class problem :

$$\mathbf{r}^+ = \frac{1}{|\Omega^+|}\sum_{\mathbf{p}\in\Omega^+}\mathbf{p}, \quad \mathbf{r}^- = \frac{1}{|\Omega^-|}\sum_{\mathbf{q}\in\Omega^-}\mathbf{q},$$

r^+, r^- : representatives
Omega$^+$: set of positive training data
Omega$^-$: set of negative training data

2a Using representatives directly for recognition :

$$\text{Label}(\mathbf{x}) = \begin{cases} +1 & \text{if } \|\mathbf{x}-\mathbf{r}^+\| < \|\mathbf{x}-\mathbf{r}^-\| \\ -1 & \text{if } \|\mathbf{x}-\mathbf{r}^-\| < \|\mathbf{x}-\mathbf{r}^+\| \end{cases}$$

(equivalent)

2b Using the discriminant function :

$$\text{Label}(\mathbf{x}) = \begin{cases} +1 & \text{if } g^+(\mathbf{x}) > g^-(\mathbf{x}) \\ -1 & \text{if } g^+(\mathbf{x}) < g^-(\mathbf{x}) \end{cases}$$

$g^+(.), g^-(.)$: discriminant functions

$$g^+(\mathbf{x}) = \sum_{j=1}^{n} x_j r_j^+ - \frac{1}{2}\sum_{j=1}^{n}(r_j^+)^2;$$

$$g^-(\mathbf{x}) = \sum_{j=1}^{n} x_j r_j^- - \frac{1}{2}\sum_{j=1}^{n}(r_j^-)^2$$

To solve a multi-class problem :

Given x, label(x) = i^* if :
$i^* = \arg\max_i g_i(\boldsymbol{x})$

LINEAR DECISION BOUNDARY

Solving a 2-class problem requires only one discriminant function :

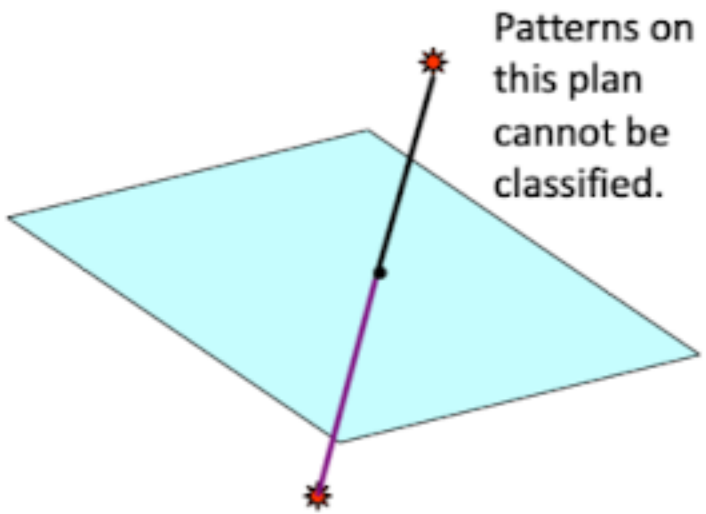
$$g(\mathbf{x}) = g^+(\mathbf{x}) - g^-(\mathbf{x}) = \sum_{j=1}^{n} w_j x_j - \theta$$

$$g^+(\mathbf{x}) = \sum_{j=1}^{n} x_j r_j^+ - \frac{1}{2}\sum_{j=1}^{n}(r_j^+)^2;$$

$$g^-(\mathbf{x}) = \sum_{j=1}^{n} x_j r_j^- - \frac{1}{2}\sum_{j=1}^{n}(r_j^-)^2$$

$$w_i = r_i^+ - r_i^-;$$

$$\theta = \frac{1}{2}\sum_{i=1}^{n}[(r_i^+)^2 - (r_i^-)^2]$$

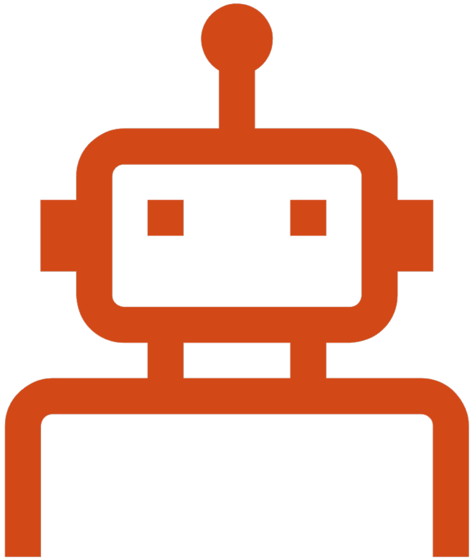


$$H : \sum_{i=1}^{n} w_i x_i - \theta = 0$$

The **hyper-plan** defined by g(**x**) forms the **decision broundary**.

CONTENTS

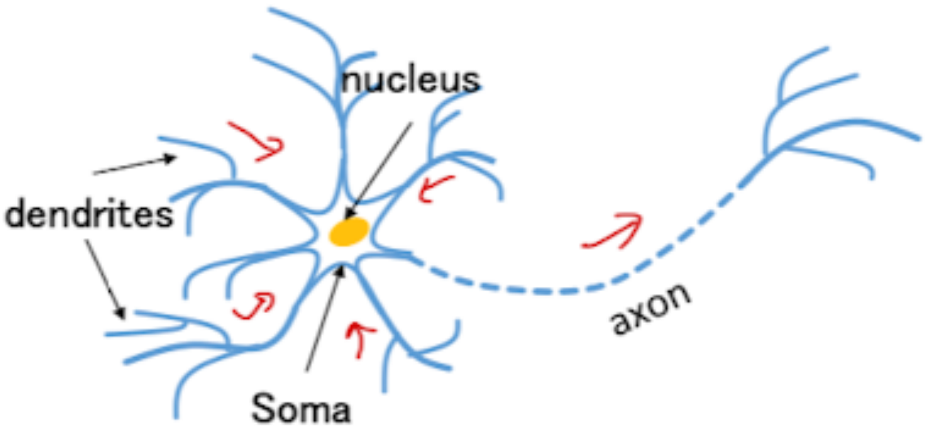
- Formulation of Machine learning (ML)
- Classification of ML methods
  - Based on learning style
  - Based on function
- Instance-based ML methods
  - K-nearest Neighbor (KNN) classifier
  - Learning Vector Quantization (LVQ) classifier
  - Numerical results on Iris flower dataset
- Neural networks
  - Multi-layer Perceptron (MLP)
    - Learning of MLP : backpropagation
  - Numerical results
- Conclusions



FROM HUMAN BRAINS TO NEURAL NETWORKS

Human brain :

- The CPU that controls the whole body
- A huge and complex network with approximately **10^11** (100B) neurons and **10^4** connections for each neuron

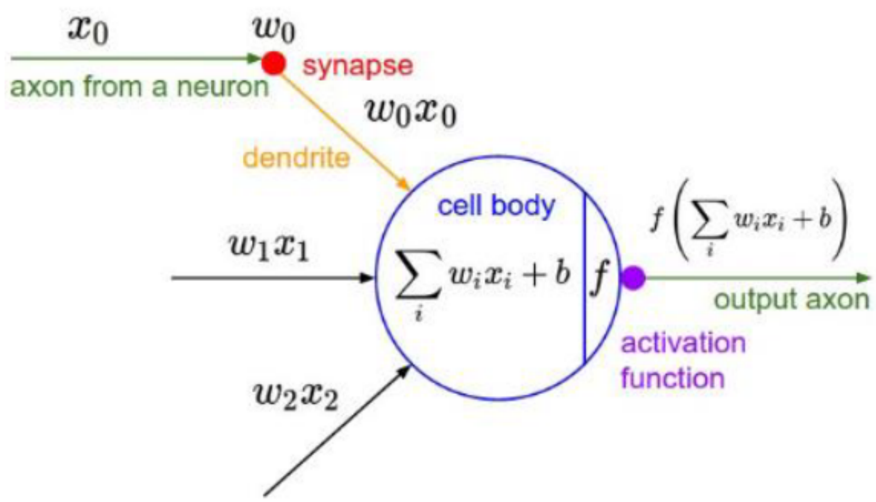


Structure of a neuron

Mathematical model :

$$y = \ g(\mathrm{u}) = f\left(\sum_{\mathrm{i}=1}^{\mathrm{n}} w_i x_i + b\right) = f(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x} + b)$$

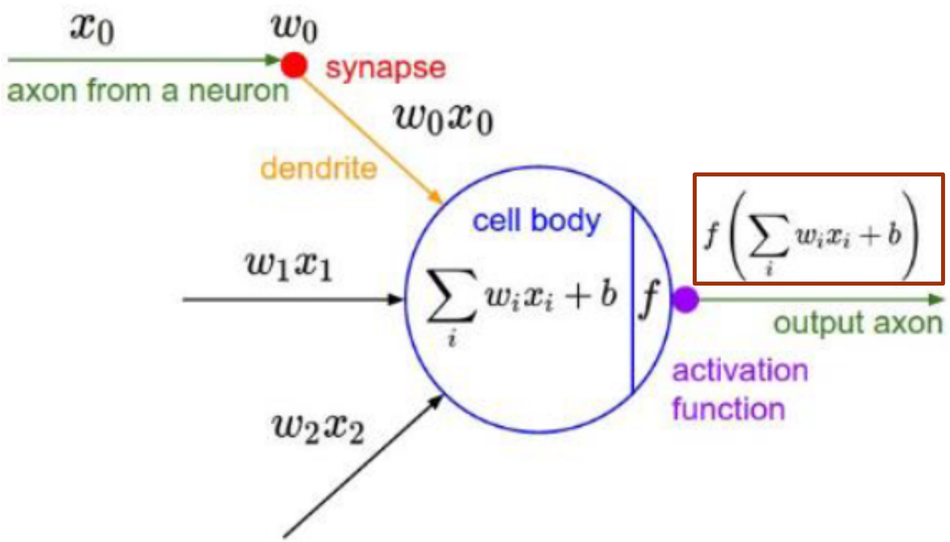
$\boldsymbol{x}$: input vector; $\boldsymbol{w}$: weight vector;
b : bias (threshold); y : output;
$f(.)$: activation function



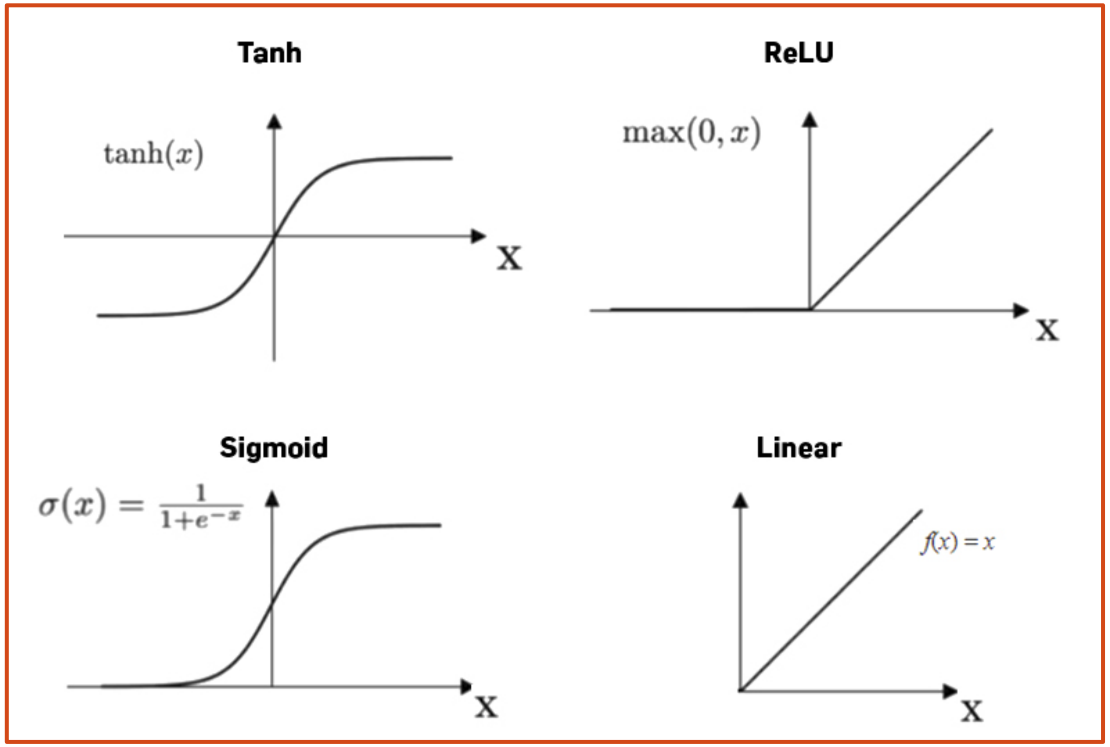
A neuron is modeled as a multi-input single-output system

Image credit : "CS231n Convolutional Neural Networks"

FROM HUMAN BRAINS TO NEURAL NETWORKS (CONT.)



Mathematical model of a neuron

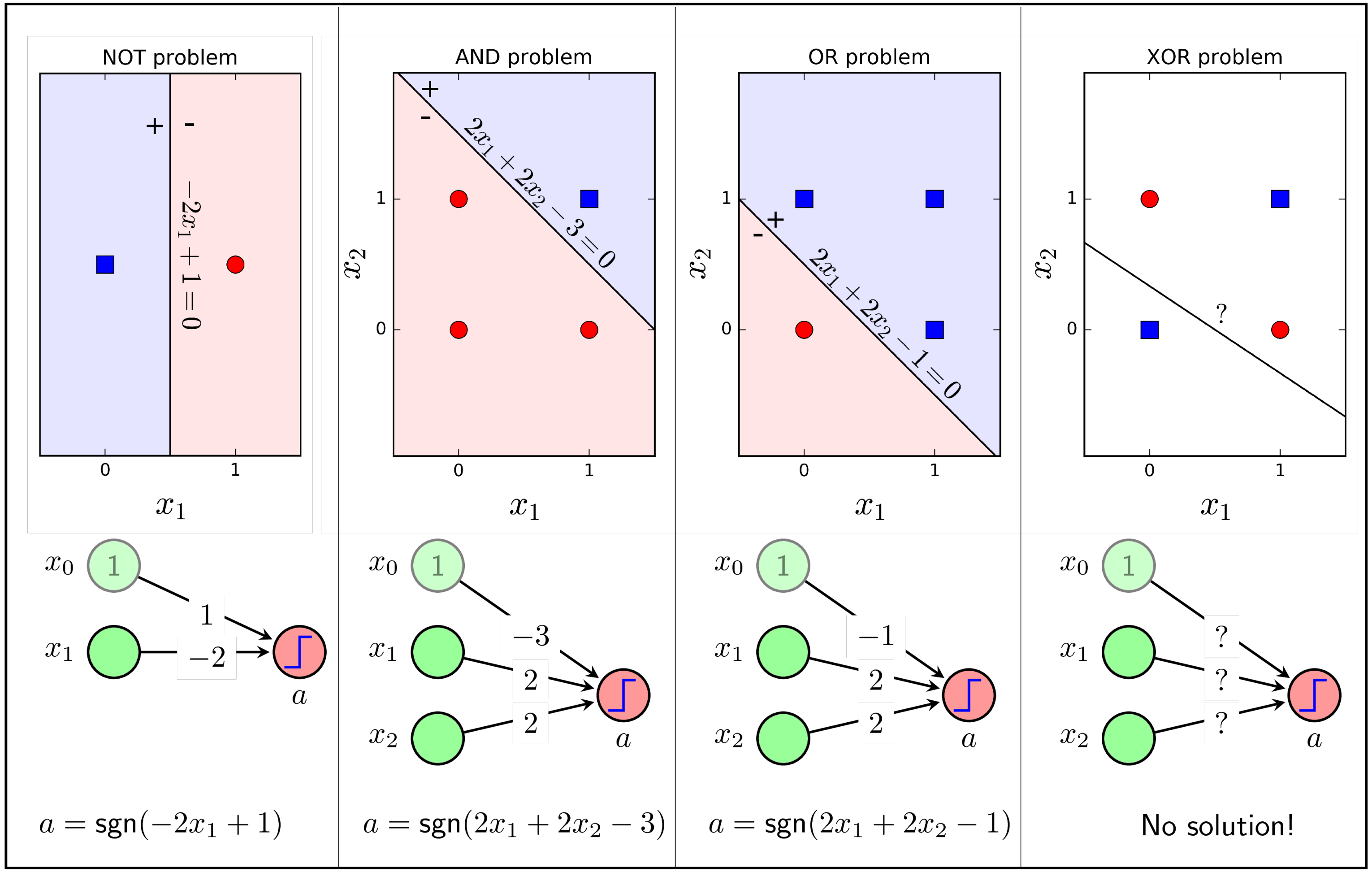


Activation functions

FROM HUMAN BRAINS TO NEURAL NETWORKS (CONT.)

One neuron has one linear decision boundary.

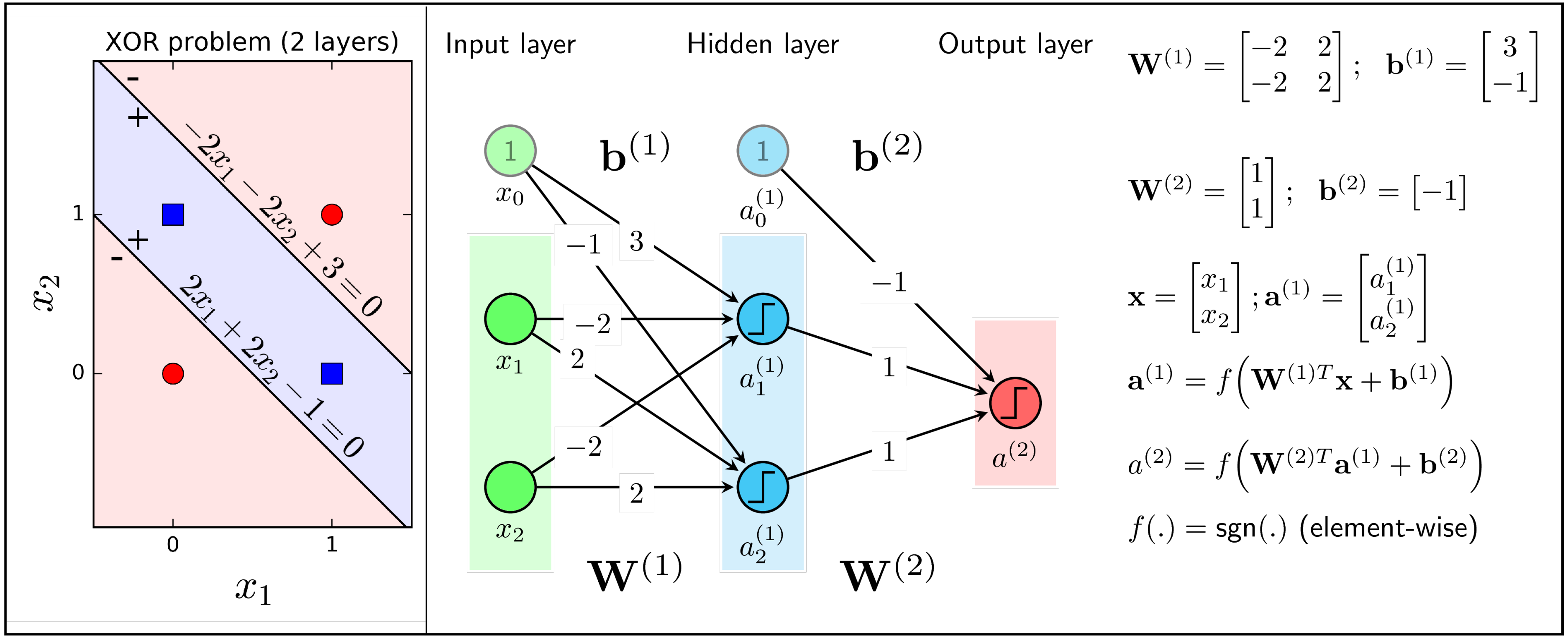
OR, AND, and OR problems: linearly seperatable.



Using **perceptrons** to model the operation of logics NOT, AND, and OR.

Image source : Machine learning cơ bản by Vũ Hữu Tiệp

FROM HUMAN BRAINS TO NEURAL NETWORKS (CONT.)



Using a **multi-layer perceptron** for the XOR problem.

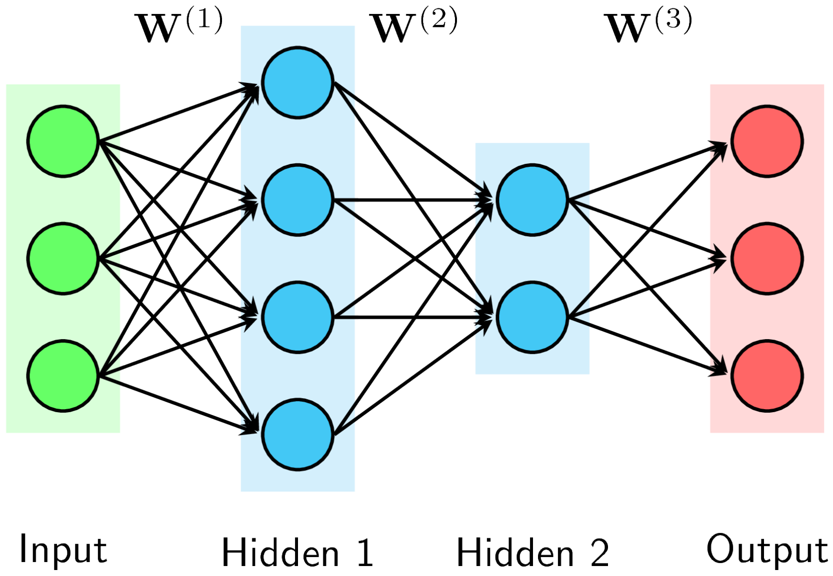
"learning" in ML

How to find the weights and biases for a MLP automatically?

(for image classification, #parameters is up to hundreds of millisions to biliions)

Image source : Machine learning cơ bản by Vũ Hữu Tiệp

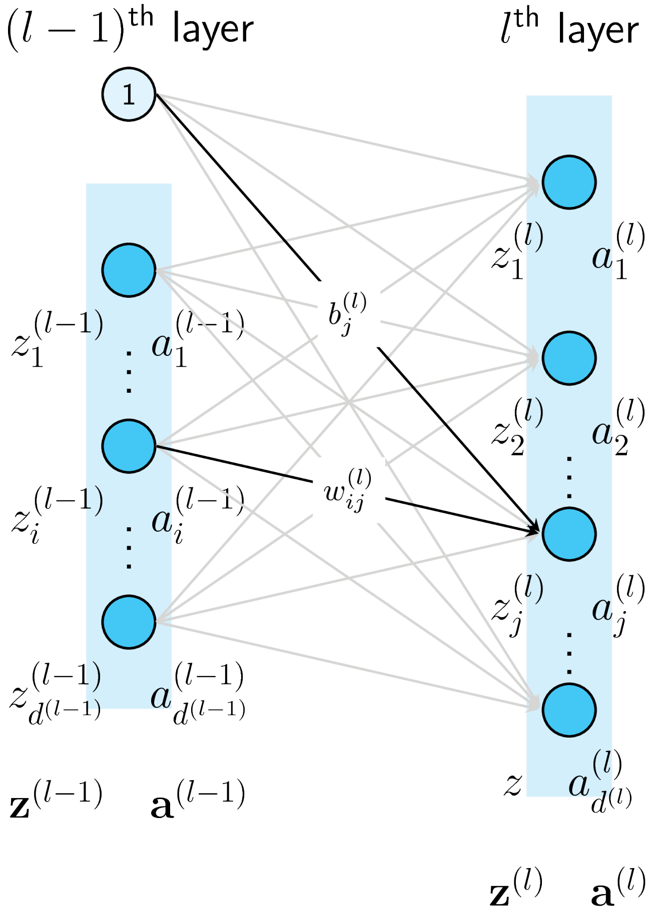
MULTI-LAYER PERCEPTRON (MLP) : DEFINITION



Multilayer perceptron : the most popular neural network

- 1 input layer
- 1 output layer
- Several (or many) hidden layers

Note : a perceptron dose not have hidden layers.



$$\mathbf{W}^{(l)} \in \mathbb{R}^{d^{(l-1)} \times d^{(l)}}$$

$$\mathbf{b}^{(l)} \in \mathbb{R}^{d^{(l)} \times 1}$$

$$z_j^{(l)} = \mathbf{w}_j^{(l)T} \mathbf{a}^{(l-1)} + b_j^{(l)}$$

$$\mathbf{z}^{(l)} = \mathbf{W}^{(l)T} \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$$

$$\mathbf{a}^{(l)} = f(\mathbf{z}^{(l)})$$

$$a_i^{(l)} = f(\mathbf{w}_i^{(l)T} \mathbf{a}^{(l-1)} + b_i^{(l)})$$

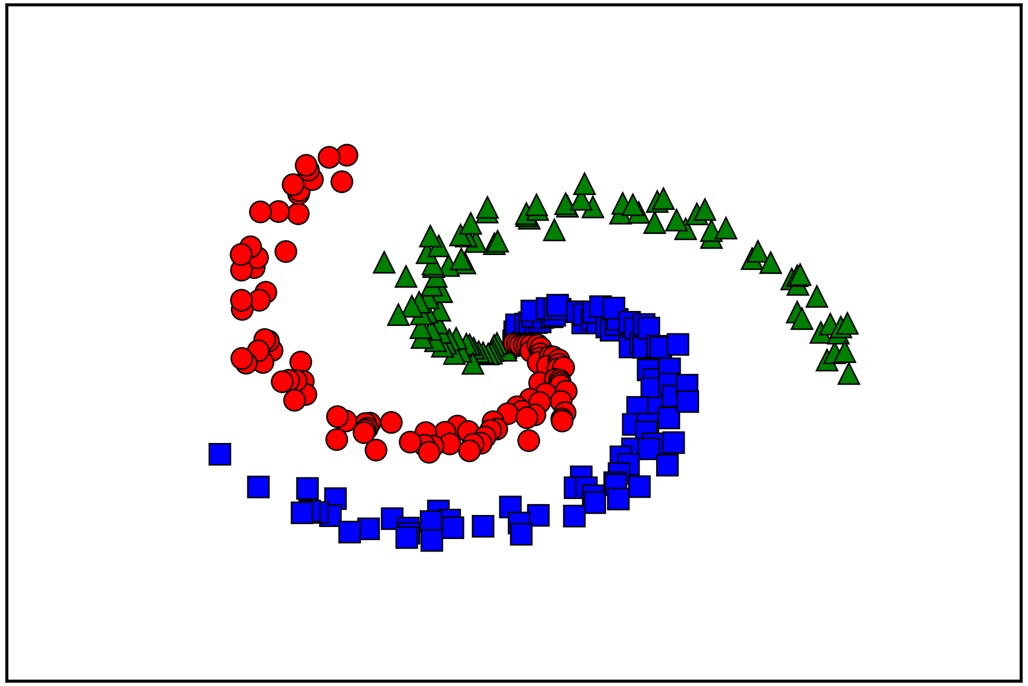
$$\mathbf{a}^{(l)} = f(\mathbf{W}^{(l)T} \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)})$$

Activation functions:

- non-linear
- element-wise
- ReLU : often used

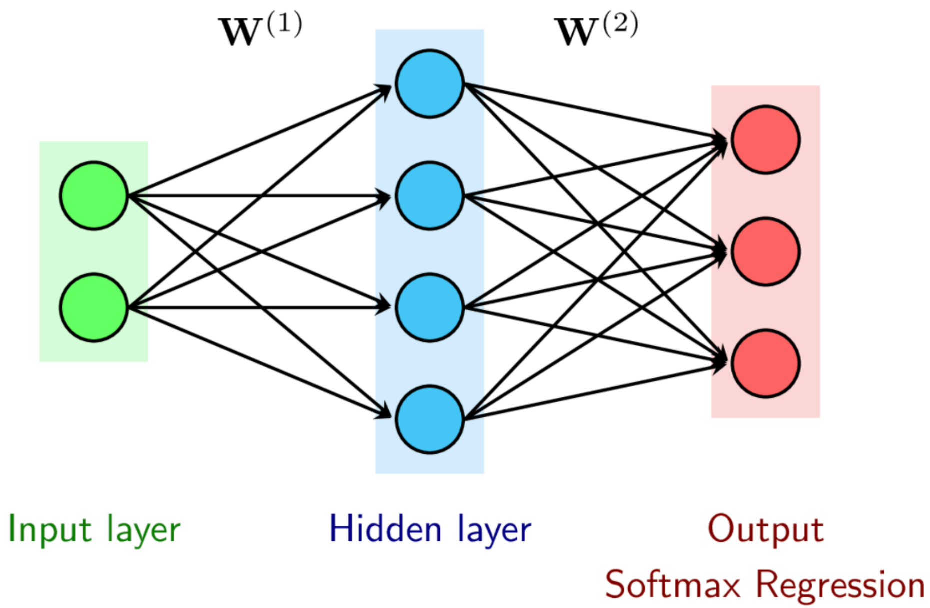
Image source : Machine learning cơ bản by Vũ Hữu Tiệp

MLP: SCENARIO



Scenario: using an MLP to classify this dataset (not linearly-seperable).

classes: C = 3
attributes: 2 (x and y)
datapoints: N= 300 (100 for each class)

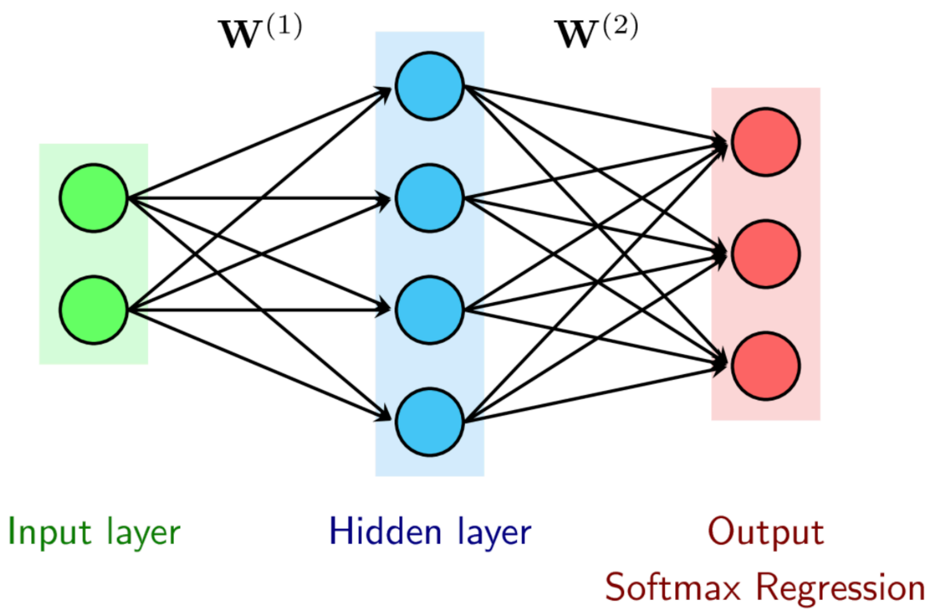


2-layer neural network

Optimizer : Batch Gradient Descent

Via: Machine learning cơ bản by Vũ Hữu Tiệp

MLP: FEEDFORWARD AND LOSS FUNCTION



Optimizer : Batch Gradient Descent

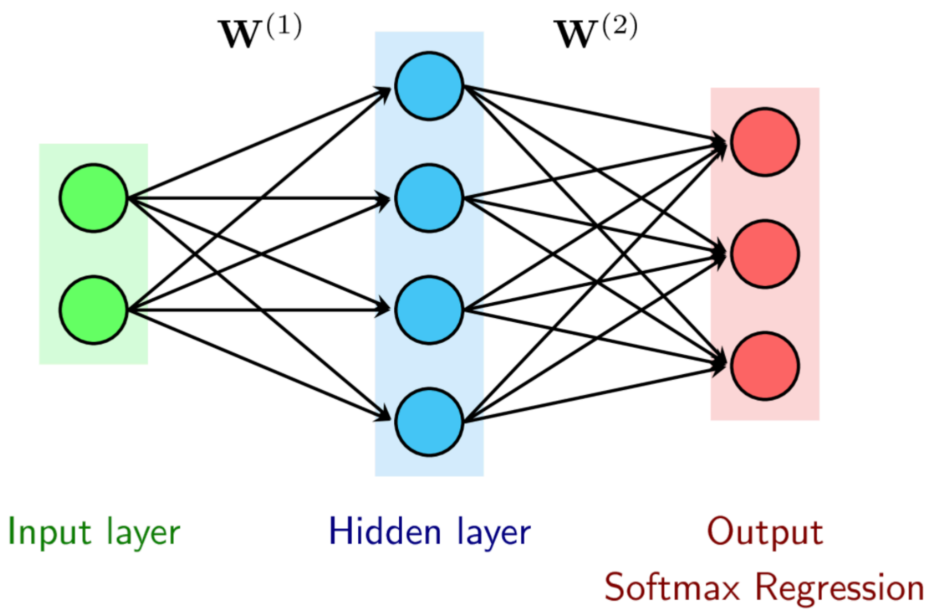
Feedforward
(predict outputs for given inputs)

$$
\begin{aligned}
\mathbf{Z}^{(1)} &= \mathbf{W}^{(1)T}\mathbf{X} \\
\mathbf{A}^{(1)} &= \max(\mathbf{Z}^{(1)}, \mathbf{0}) \\
\mathbf{Z}^{(2)} &= \mathbf{W}^{(2)T}\mathbf{A}^{(1)} \\
\hat{\mathbf{Y}} = \mathbf{A}^{(2)} &= \text{softmax}(\mathbf{Z}^{(2)})
\end{aligned}
$$

Loss function (cross-entropy):

$$J \triangleq J(\mathbf{W}, \mathbf{b}; \mathbf{X}, \mathbf{Y}) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ji}\log(\hat{y}_{ji})$$

MLP: BACK-PROPAGATION (GRADIENT DESCENT)



Optimizer : Batch Gradient Descent

Backpropagation:

$$\mathbf{E}^{(2)} = \frac{\partial J}{\partial \mathbf{Z}^{(2)}} = \frac{1}{N}(\hat{\mathbf{Y}} - \mathbf{Y})$$

$$\frac{\partial J}{\partial \mathbf{W}^{(2)}} = \mathbf{A}^{(1)}\mathbf{E}^{(2)T}$$

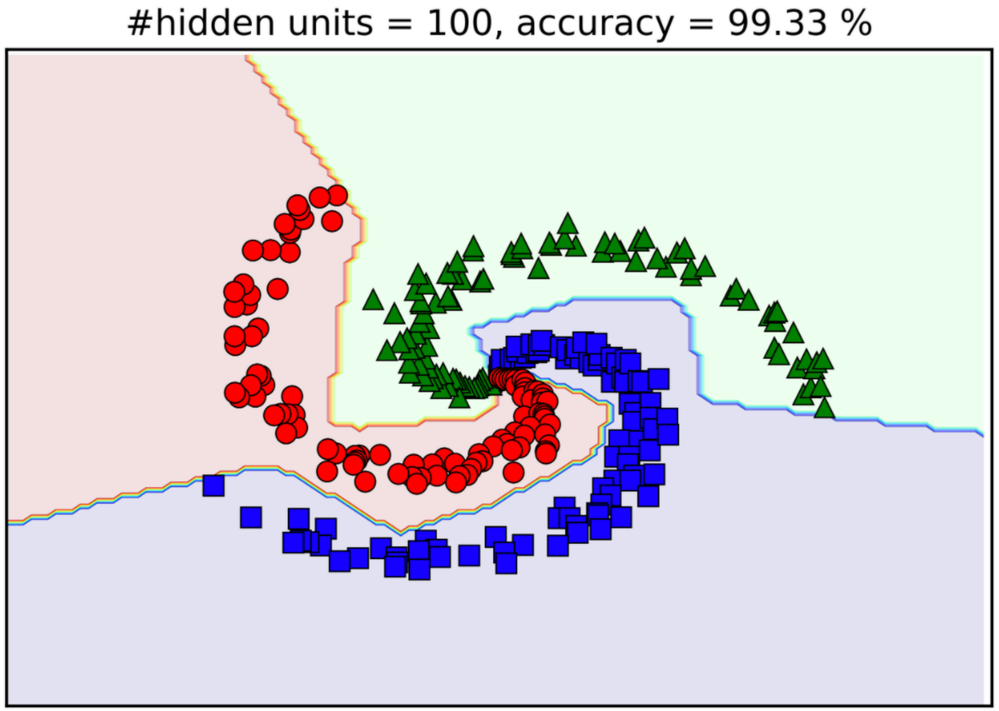
$$\frac{\partial J}{\partial \mathbf{b}^{(2)}} = \sum_{n=1}^{N} \mathbf{e}_n^{(2)}$$

$$\mathbf{E}^{(1)} = \left(\mathbf{W}^{(2)}\mathbf{E}^{(2)}\right) \odot f'(\mathbf{Z}^{(1)})$$

$$\frac{\partial J}{\partial \mathbf{W}^{(1)}} = \mathbf{A}^{(0)}\mathbf{E}^{(1)T} = \mathbf{X}\mathbf{E}^{(1)T}$$

$$\frac{\partial J}{\partial \mathbf{b}^{(1)}} = \sum_{n=1}^{N} \mathbf{e}_n^{(1)}$$

NUMERICAL RESULTS



Hình 9: Kết quả khi sử dụng 1 hidden layer với 100 units.

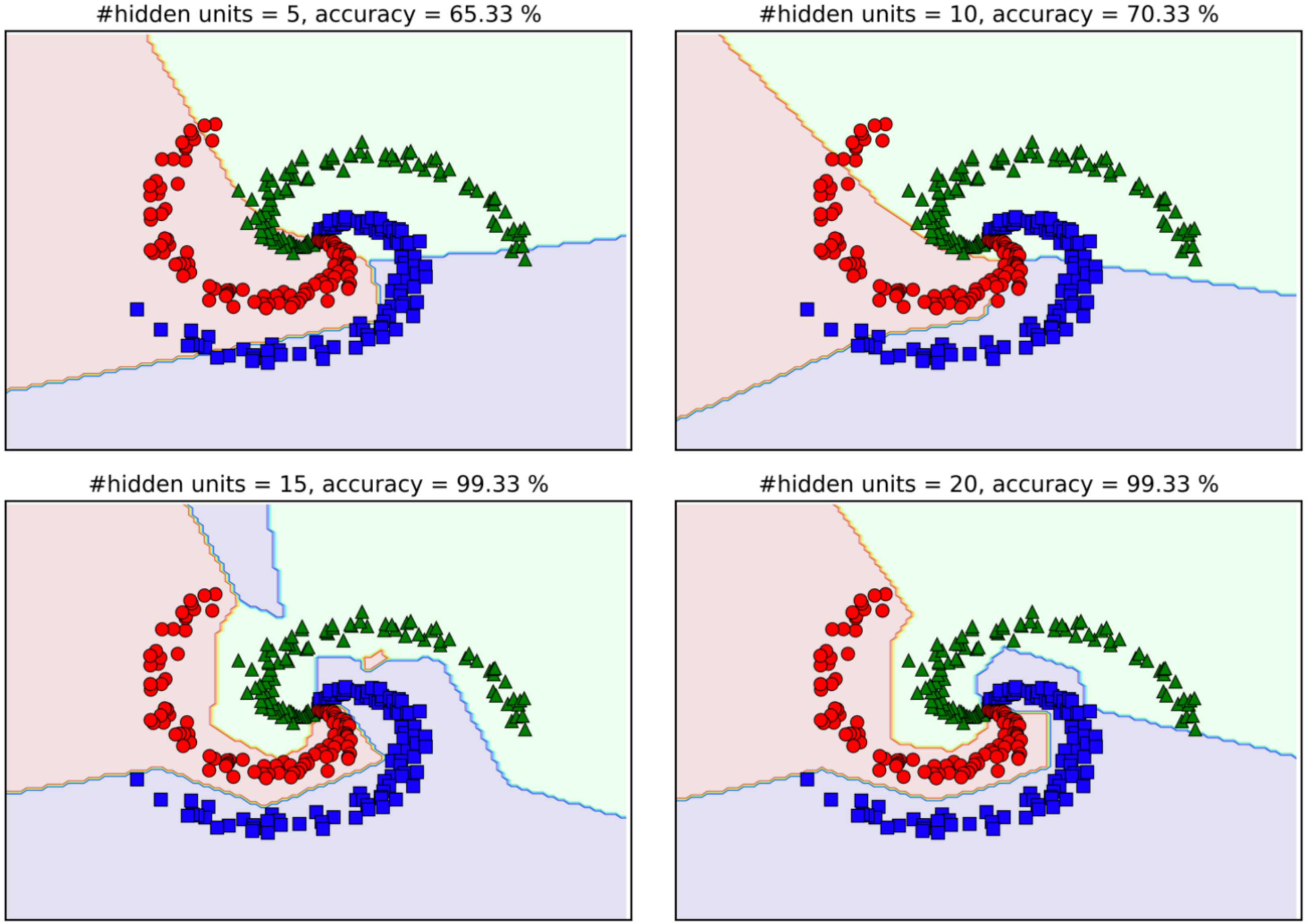
iterations = 10,000
learning_rate = 1

Accuracy = 99.33%
(only 2 points are missclassified)

By adding only one more hidden layer, we can build up non-linear boundaries for calssification.

Via: Machine learning cơ bản by Vũ Hữu Tiệp

NUMERICAL RESULTS



Hình 10: Kết quả với số lượng units trong hidden layer là khác nhau.

Increasing # of hidden units improves the accuracy.

CONCLUSIONS

Neural networks:

- [3] proved that: a NN (with appropriate # of layers and activation funtions) can approximate any continuous function given any error rate epsilon>0.
- # of layers, # of hidden units and activation functions: critical hyper-parameters.
- Increasing # of hidden units:
  - may produce better accuracy
  - but requires longer time for training+testing
  - and may result in overfitting problem (does well on training set but does not generalize well on testing set).

Machine Learning: a very big field with a wide range of applications.

REFERENCES

This slide refers to and uses various images obtained from:

1. "CS231n: Convolutional Neural Networks" for Visual Recognition by Prof. Fei-Fei Li from Stanford. http://cs231n.stanford.edu/ (accessed Aug. 22, 2020).
2. T. Vu, "Machine learning cơ bản," *Tiep Vu's blog*, Jul. 17, 2017. https://machinelearningcoban.com/.
3. G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Math. Control Signal Systems*, vol. 2, no. 4, pp. 303–314, Dec. 1989, doi: 10.1007/BF02551274.

- Thank you for your attention
- Q&A