Supervised Learning

Silèye Ba

Adjunct Lecturer, Neoma Business School, Paris
Machine Learning Scientist, L'Oreal, Paris

02.02.2023

Overview

Overview

Class evaluation

Group projects : deadline 31 of March 2023

- Choose a finance problem at `https://challengedata.ens.fr/`
- Constitute 3-persons groups : send group and problem
- Deliverables : data, working notebook and synthetic report

Personal projects

- Problem will be given 16th of March 2023
- Deadline to submit : deadline 31 of March 2023

Projects expectations

- Use what you learned during class : test and compare many models
- Evaluating your proposals properly : cross validation, use right metrics
- Being creative is always a plus : go beyong what you learned in class
- Code quality is part of the evaluation

Overview

Few reminders from last classes

Gradient of $f : \mathbb{R}^D \to \mathbb{R}$

If $\mathbf{x} = (x_1, x_2, ..., x_D)^t$, the gradient of $f(\mathbf{x}) \in \mathbb{R}$ is the vector constituted of it's partial derivative $\nabla_{\mathbf{x}} f(\mathbf{x}) = (\frac{\partial f}{\partial x_1}(\mathbf{x}), \frac{\partial f}{\partial x_2}(\mathbf{x}), ..., \frac{\partial f}{\partial x_D}(\mathbf{x}))^t$

Gradient descent

$f : \mathbb{R}^D \to \mathbb{R}$ a convex function, gradient descent is an algorithm to iteratively estimate the minimum $\hat{\mathbf{x}}$ of f according to the steps :

1. Select an initial value $\hat{\mathbf{x}}_0$, set $t = 0$
2. Update $\hat{\mathbf{x}}_{t+1} = \hat{\mathbf{x}}_t - \lambda \nabla_{\mathbf{x}} f(\hat{\mathbf{x}}_t)$ ($\lambda > 0$: descent step)
3. Update $t = t + 1$, and loop back to step 2 until convergence

Supervised learning : classification

What is classification

We are given data samples $(\mathbf{x}_n)$ with corresponding labels (y_n) where $y_n \in \{0, 1, ..., C-1\}$.

- C is called the number of classes
- Binary classification : $C = 2$
- Multi-class problem : $C > 2$
- Multi-label problem : non-mutually exclusive labels

Learning formalized as function approximation

- Hypothesis : assume $y = f_{\mathbf{w}}(\mathbf{x})$ for some unknown function $f_{\mathbf{w}}$ parametrized by $\mathbf{w}$
- Learning problem : estimate $\mathbf{w}$ given labeled dataset $(\mathbf{x}_n, y_n)$
- Make reliable predictions $y = f_{\mathbf{w}}(\mathbf{x})$ for new samples $\mathbf{x}$
  - Generalization : ability to reliably predict on new samples

Supervised learning : regression

What is regression

We are given data samples $(\mathbf{x}_n)$ with associated values $(\mathbf{y}_n)$ where $\mathbf{y}_n$ are continuous variable $(\in \mathbb{R}^d)$.

- Similar to classification problem but with continuous outputs

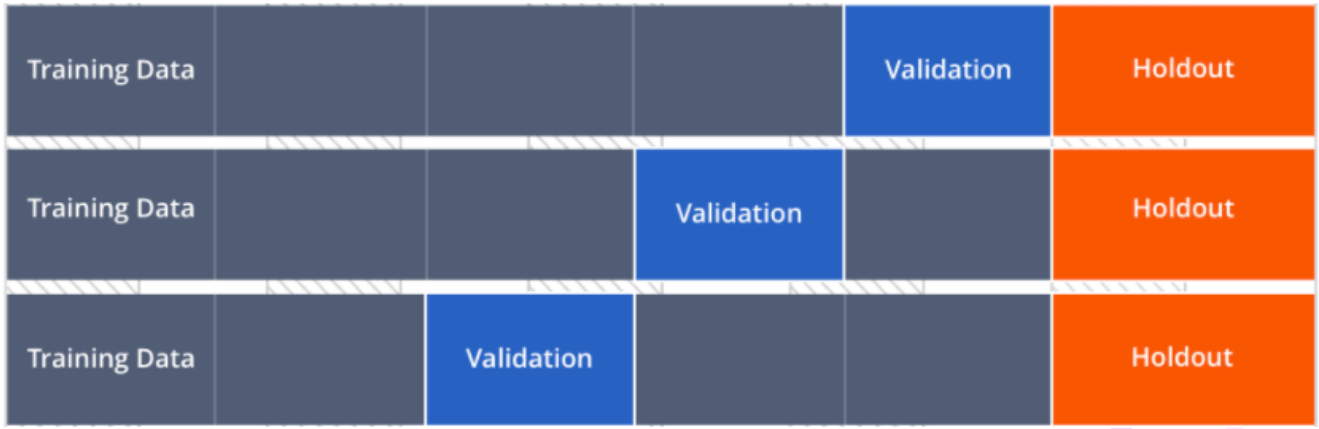
Learning formalized as function approximation

- Hypothesis : assume $\mathbf{y} = f_{\mathbf{w}}(\mathbf{x})$ for some unknown function $f_{\mathbf{w}}$ parametrized by $\mathbf{w}$
- Learning problem : estimate $\mathbf{w}$ given training dataset $(\mathbf{x}_n, \mathbf{y}_n)$
- Make reliable prediction $\mathbf{y} = f_{\mathbf{w}}(\mathbf{x})$ for new samples $\mathbf{x}$

Supervised learning evaluation : cross validation

- Assessing models generalization abilities :
  - How well models performs on samples outside the training set
- Evaluation protocol : cross-validation
  - Split available data into three subset : training, validation, test
  - Use training data to fit model parameters :
    - Many model parameters' configurations are considered
  - Use the validation data to select the best parameters
  - Use the test data set to assess the best model generalization performances

six-fold cross-validation



Overview

Overview

K-nearest neighbors (KNN) method for classification

If we are given :

- Dataset $(\mathbf{x}_n, y_n)$ of samples $\mathbf{x}_n \in \mathcal{X}$ with labels $y_n \in \mathcal{C} = \{0, ..., C-1\}$
- Distance ρ to compute proximity between elements of $\mathcal{X}$
- δ_c : Dirac delta function
- $\mathcal{N}_{\rho,K}(\mathbf{x})$: set of the K-nearest neighbors of samples $\mathbf{x}$

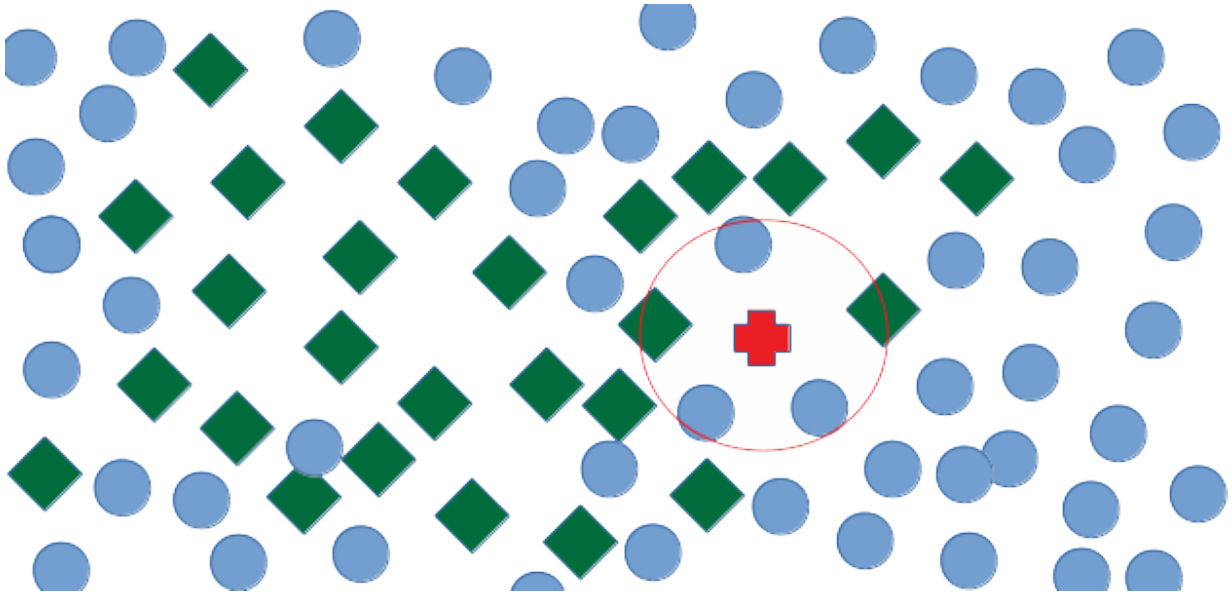
KNN classifier

K-nearest neighbors prediction for sample $\mathbf{x}$ is given by

$$f_{\rho,K}(\mathbf{x}) = \arg\max_{c \in \mathcal{C}} \sum_{\mathbf{x}_k \in \mathcal{N}_{\rho,K}(\mathbf{x})} \delta_c(y_k)$$

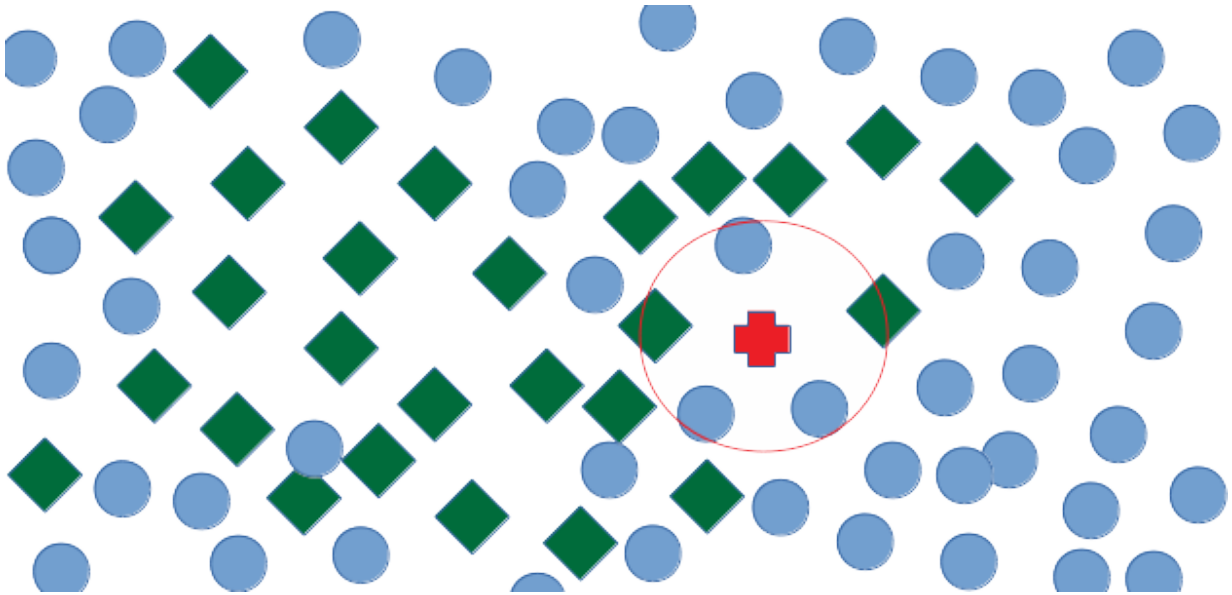
KNN for classification : simple example

- Data : blue dots, green diamonds
- Red cross is a new sample
- Question : what is the class of red cross in a 5-NN classification ?



KNN for classification : simple example

- Data : blue dots, green diamonds
- Red cross is a new sample
- Question : what is the class of red cross in a 5-NN classification ?



In a 5-NN classification, the class of the red cross is blue dot !

K-nearest neighbors (KNN) method for regression

Principle

- Dataset $(\mathbf{x}_n, y_n)$ of samples $\mathbf{x}_n \in \mathcal{X}$ with labels $y_n \in \mathbb{R}^D$
- Distance ρ to compute proximity between elements of $\mathcal{X}$
- Averaging kernel $\omega(\mathbf{x}, \mathbf{x}_{n_k})$
- K-nearest neighbors regression of sample $\mathbf{x}$ is given by

$$f_{\rho,K}(\mathbf{x}) = \sum_{\mathbf{x}_k \in \mathcal{N}_{\rho,K}(\mathbf{x})} \omega(\mathbf{x}, \mathbf{x}_k) y_k$$

Example of averaging kernels

- Simple averaging : $\omega(\mathbf{x}, \mathbf{x}_{n_k}) = \frac{1}{K}$
- Gaussian averaging : $\omega(\mathbf{x}, \mathbf{x}_{n_k}) = \frac{\exp -\lambda\rho(\mathbf{x},\mathbf{x}_{n_k})}{\sum_{l=1}^{K} \exp -\lambda\rho(\mathbf{x},\mathbf{x}_{n_l})}$

Remarks about K-nearest neighbors

Pro

- No parameters learning is required : model is fully specified when number of neighbors K and metric ρ are provided
- KNN allows to address problems with non-linear decision boundaries

Cons

- High computational cost : efficient search procedure are required to quickly find the K nearest neighbors
  - For every test samples $\mathbf{x}$, computing distance to all samples $\mathbf{x}_n$ is needed
  - Computationally expensive for very large datasets
- Large storage capacity : all the samples are stored in memory

Overview

Decision trees for classification

Definition

Given a dataset $(\mathbf{x}_n, y_n)$, decision trees for classification are models that are iteratively built by segmenting the input space of the samples $\mathbf{x}_n$ into **homogenenous** regions in term of sample classes y_n.The decision tree classifier is a function of the form :

$$f_\Omega(\mathbf{x}) = \sum_{l=1}^{L} c_l \mathbb{I}_{\Omega_l}(\mathbf{x})$$

where $\mathbb{I}_{\Omega_l}$ is the indicator function of the set Ω_l, and c_l is the majority class of samples $\mathbf{x}_n$ in region Ω_l.

Algorithm

1. Choose an attribute (component) from $\mathbf{x} = (x^1, x^2, ..., x^D)$
2. Compute the significance of an attribute x^d for splitting the data
3. Split data based on the best attribute
4. Loop back to step 1

Decision trees for classification : homogeneity measure

An homogeneity measure captures to which extent input space is segmented in regions containing samples of a single class.

Entropy as homogeneity measure

If in area Ω_l the occurence probability of samples of class c is p_{lc}, entropy inside Ω_l is :

$$E_l = -\sum_{c=1}^{C} p_{lc} \log p_{lc}$$

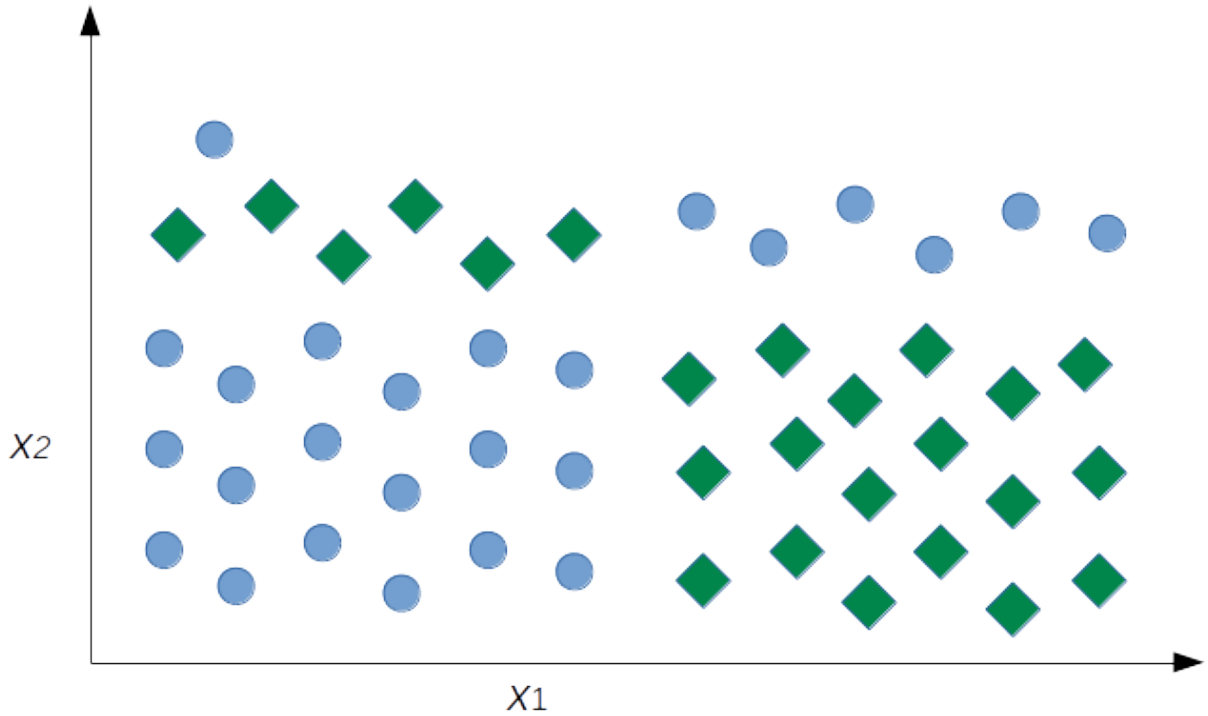
- When Ω_l contains only samples of one class : $E_l = 0$
- When Ω_l contains uniformly samples of all classes : $E_l = \log C$

Global homogeneity measure : average entropy

$$E = \frac{1}{L} \sum_{l=1}^{L} E_l$$

Solving a simple decision tree classification problem

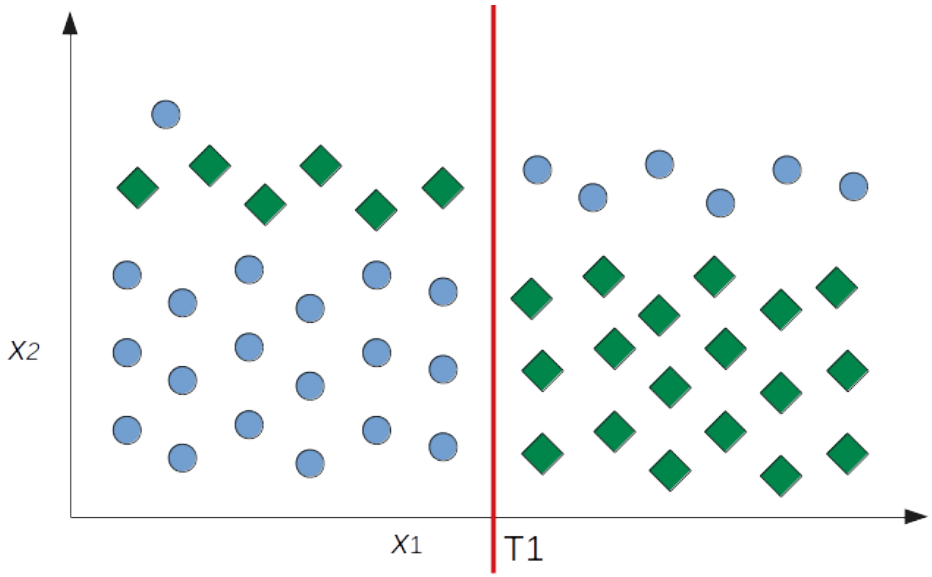
Build a decision tree to classify blue dots and green diamonds :
$\mathbf{x} = (x^1, x^2)$



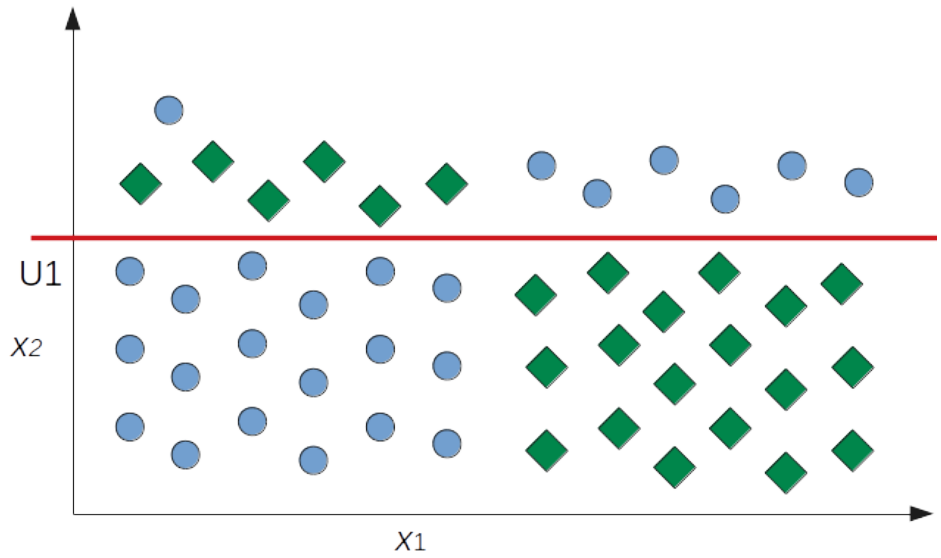
Simple classification problem : first split

- Proposed split based on attribute x^1 : $T1$
- Proposed split based on attribute x^2 : $U1$
- Which split is the best one ?

vertical split



horizontal split

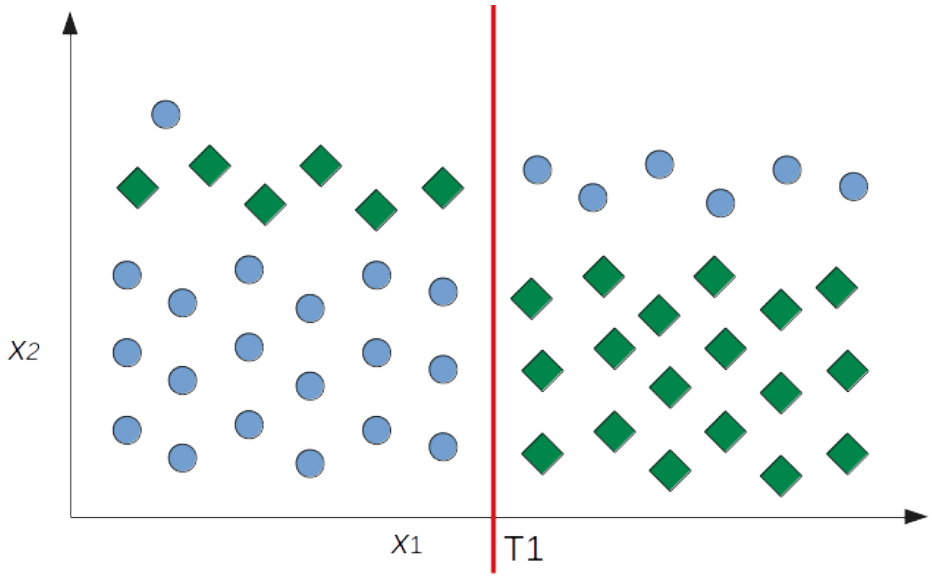


$T1$ is the best split base on the entropy measure.

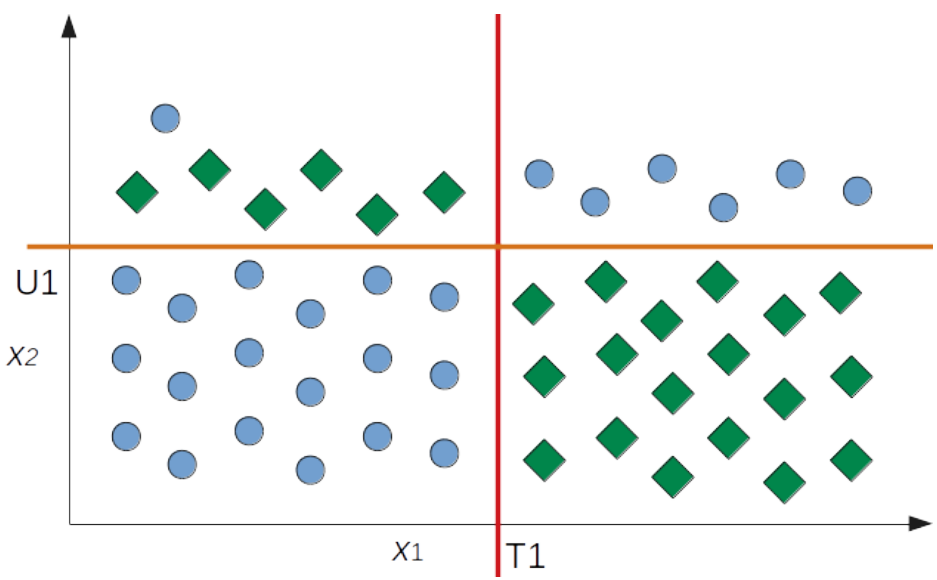
Simple classification problem : second split

- Second optimal split : split wrt to attribute x^2 : $U1$

first optimal split

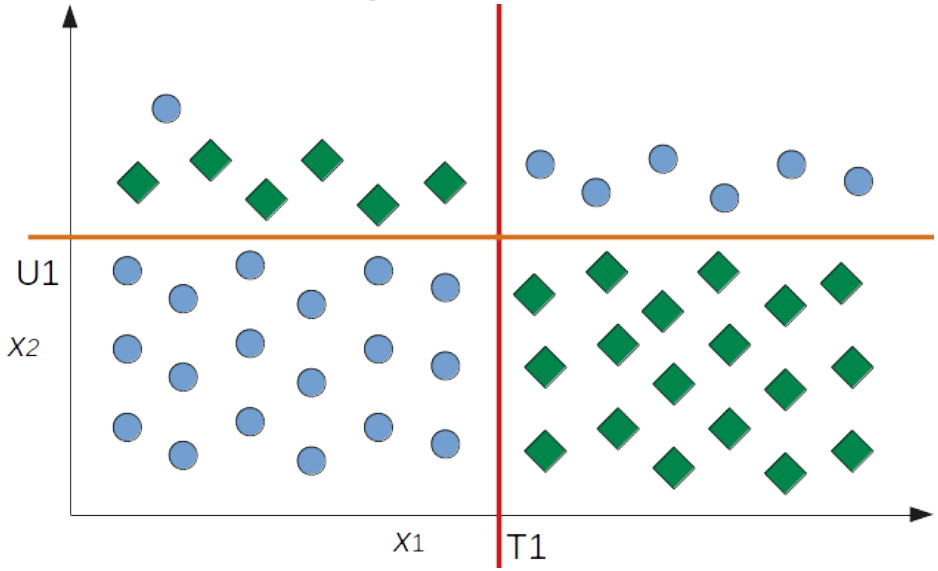


second optimal split

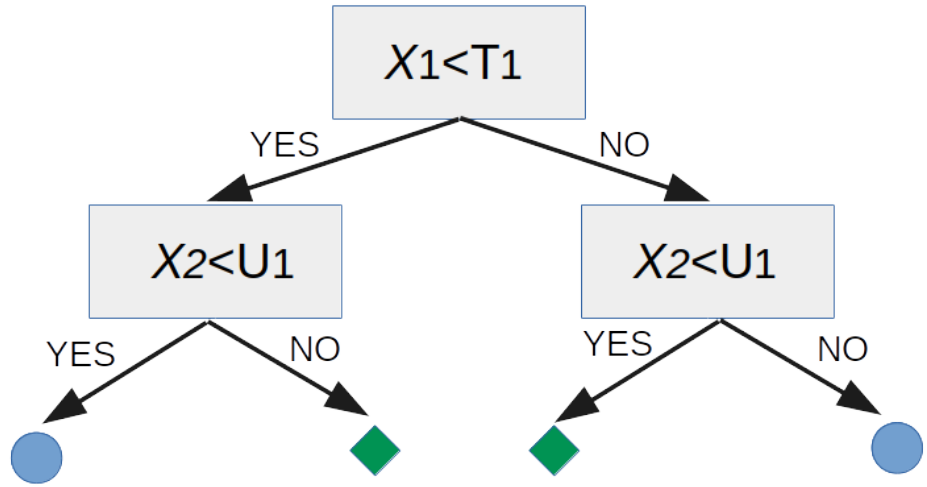


Simple classification problem : decision tree

input space segmentation



decision tree



Decision trees for regression

Definition

Given a dataset $(\mathbf{x}_n, y_n)$, decision trees for regression are models that are iteratively built by segmenting the input space of the samples $\mathbf{x}_n$ into regions Ω_l such that

$$\frac{1}{|\Omega_l|} \sum_{\mathbf{x}_n \in \Omega_l} \rho(y_n, y_{\Omega_l}(\mathbf{x})) < \epsilon.$$

The decision tree regressor is a function of the form :

$$f_\Omega(\mathbf{x}) = \sum_{l=1}^{L} y_{\Omega_l}(\mathbf{x}) \mathbb{I}_{\Omega_l}(\mathbf{x})$$

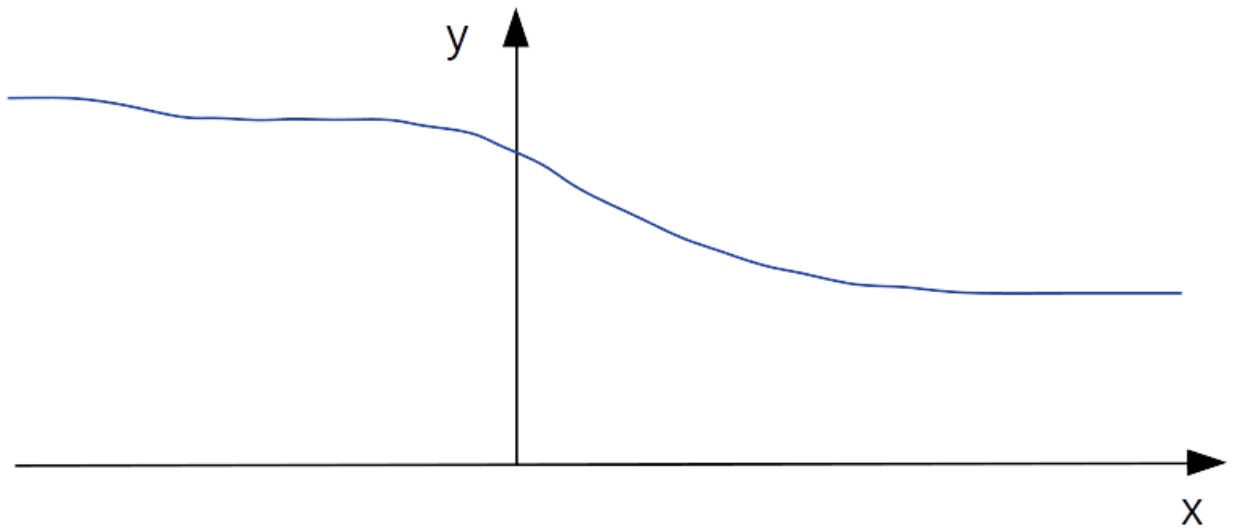
Algorithm

1. Choose an attribute (component) from $\mathbf{x} = (x_1, x_2, ...)$
2. Compute the significance of an attribute for splitting the data
3. Split data based on the best attribute
4. Loop back to step 1

Regression trees fitting example

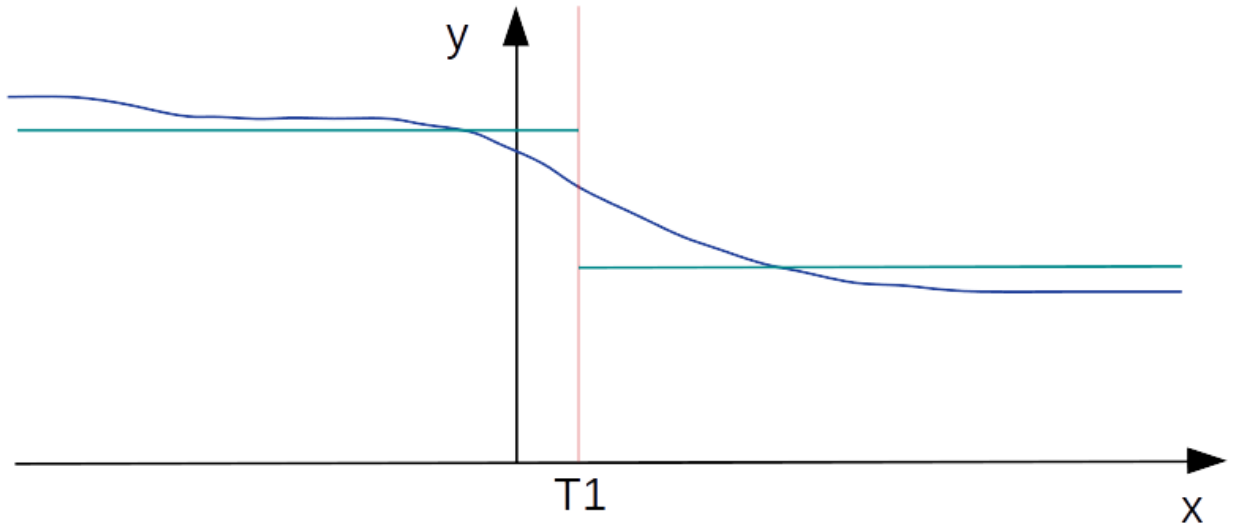
Fitting a regression tree to the function below when, for $\mathbf{x} \in \Omega_l$, considering as approximating function $y_{\Omega_l}(\mathbf{x})$ the average of the outputs y_n of samples $\mathbf{x}_n \in \Omega_l$:

$$y_{\Omega}(\mathbf{x}) = \frac{1}{|\Omega_l|} \sum_{\mathbf{x}_n \in \Omega_l} y_n$$



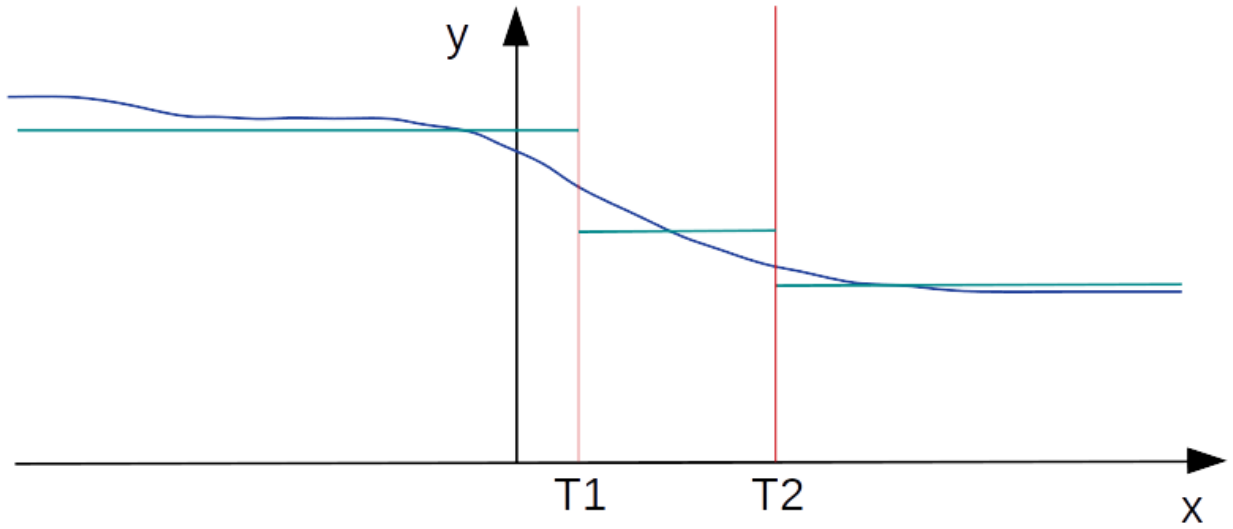
Regression trees fitting example : step 1

Input space is segmented into two regions : $x < T_1$ and $x \geq T_1$



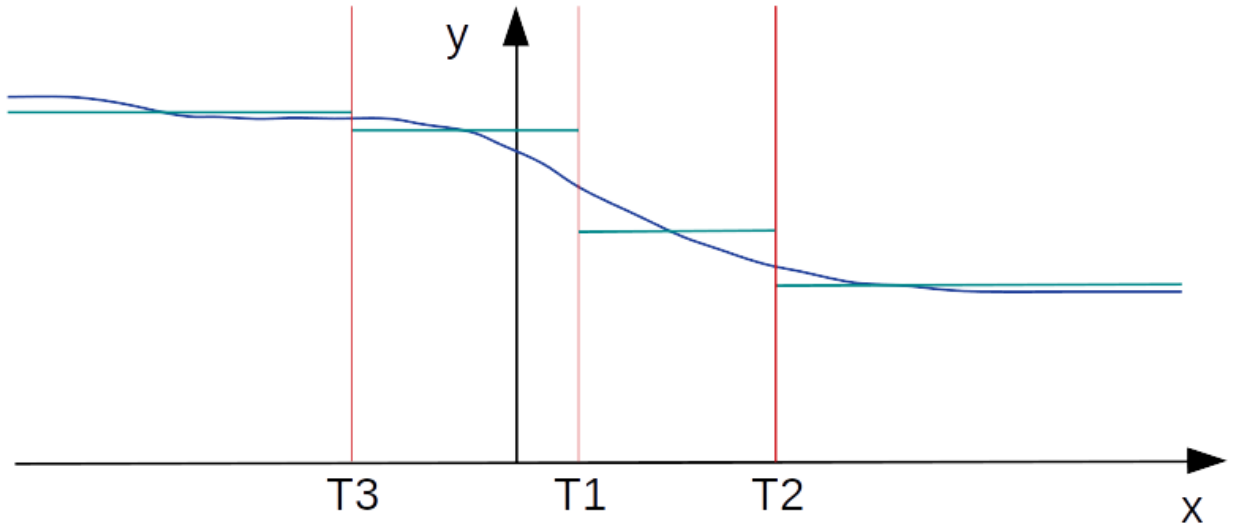
Regression trees fitting example : step 2

region $x \geq T_1$ is segmented into two regions : $T_1 \leq x < T_2$ and $x \geq T_2$



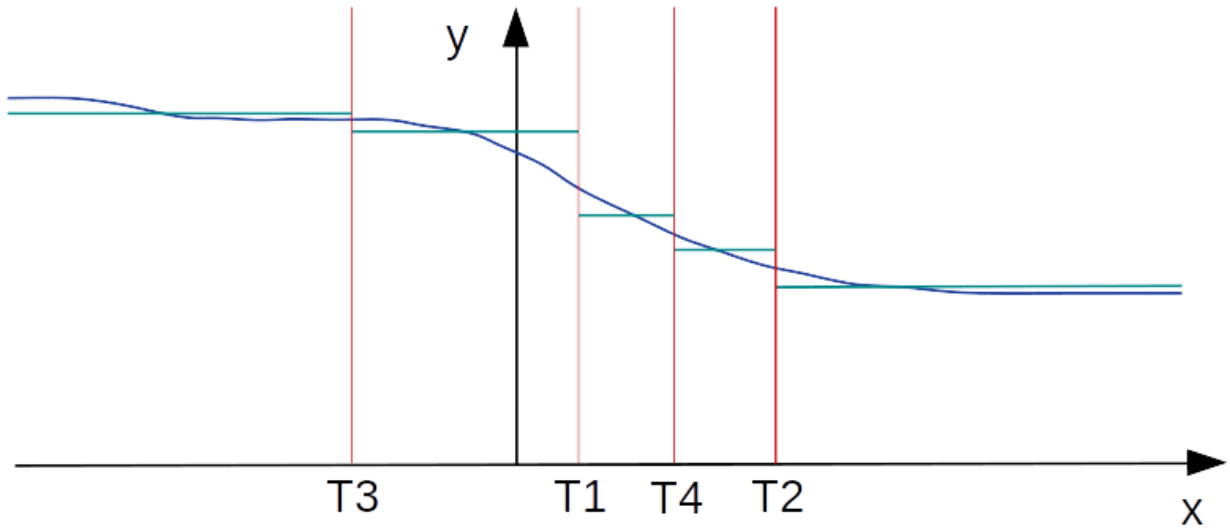
Regression trees fitting example : step 3

region $x < T_1$ is segmented into two regions : $T_3 \leq x < T_1$ and $x < T_3$



Regression trees fitting example : step 4

region $T_1 \leq x < T_2$ is segmented into two regions : $T_1 \leq x < T_4$ and $T_4 \leq x < T_2$



Segmentation stops because in all regions, approximation is *close enough* to the sample outputs.

Remarks about decision trees

Pro

- Decision trees address non-linear classification/regression problems
- Decisions trees are not black-boxes : interpretable decision reasons
  - Attribute importances be computed

Cons

- Greedy iterative splitting : input space is exhaustively searched
  - Solution : use random splits
- Horizontal-vertical splitting leads to deep trees to address obliques classification boundaries or functions
  - Solution : use oblique decision trees with linear regression defining leaves
- Overfitting tendencies :
  - Solution : pruning, random forests

Random forest : bootstrapping decision trees

Principle

Given a training dataset $\mathcal{D} = \{(\mathbf{x}_n, y_n), n = 1, ..., N\}$

- for $m = 1, ..., M$
  - Bootstrap a sub-dataset $\mathcal{D}_m$ from $\mathcal{D}$
  - Build a regression tree $f_\Omega^{(m)}(x)$
- Random forest : aggregation of the built regression trees

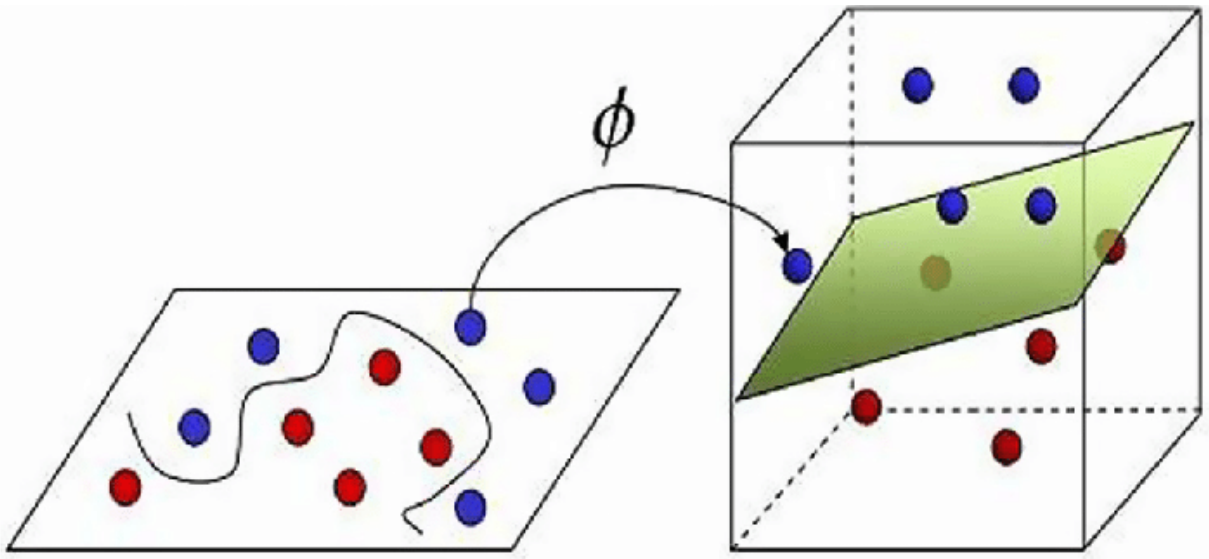
$$F(x) = \frac{1}{M} \sum_{m=1}^{M} f_\Omega^{(m)}(x)$$

- Resulting models robust to noise due to averaging procedure
- Random forest are interpretable models as are regression trees
- Models of a family known as bagging, boosting
  - Any model can be bagged : support vecto machines, neural net, etc
  - Challenge top performing models are usually bagged models

Overview

Kernel methods : motivations

Mapping inputs to a feature space where they can be explained linearly.



Kernel methods

Definition

Kernel methods are a class of models for pattern modelling in which linear models are turned into non-linear ones by applying the **kernel trick**, namely by non-linearly transforming input features $\mathbf{x}$ to features $\phi(\mathbf{x})$ through a kernel κ defined by the scalar product $\kappa(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^\top \phi(\mathbf{x}')$.

Kernel trick

In practise, only the kernel $\kappa(\mathbf{x}, \mathbf{x}')$ is used and not directly the feature map ϕ, because the map ϕ always occurs as the scalar product $\phi(\mathbf{x})^\top \phi(\mathbf{x}')$.

Famous kernel methods

- Kernel regression
- Support vector machines (SVM)
- Kernel principal component analysis (PCA)

Kernel trick regression

Problem statement

We are given a dataset $(\mathbf{x}_n, y_n)$ of input features $\mathbf{x}_n$ with corresponding labels y_n. We are also given features map $\phi(\mathbf{x})$ and a kernel κ defined by the scalar product

$$\kappa(\mathbf{x}_n, \mathbf{x}_m) = \phi(\mathbf{x}_n)^\top \phi(\mathbf{x}_m).$$

Our goal is to build a linear regression mapping features to outputs as :

$$y = \mathbf{w}^\top \phi(\mathbf{x})$$

Regularized training loss

$$L(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(\mathbf{w}^\top \phi(\mathbf{x}_n) - y_n)^2 + \frac{\lambda}{2}\mathbf{w}^\top \mathbf{w}$$

with gradient :

$$\nabla_{\mathbf{w}} L(\mathbf{w}) = \sum_{n=1}^{N}(\mathbf{w}^\top \phi(\mathbf{x}_n) - y_n)\phi(\mathbf{x}) + \lambda \mathbf{w}$$

Kernel trick regression

Vanishing gradient's loss gives

$$\mathbf{w} = \sum_{n=1}^{N} a_n \phi(\mathbf{x}_n) \text{ with } a_n = \frac{-1}{\lambda}(\mathbf{w}^\top \phi(\mathbf{x}_n) - y_n)$$

Replacing a_n in the training loss

$$L(\mathbf{a}) = \frac{1}{2}\mathbf{a}^\top K K \mathbf{a} - \mathbf{a}^\top K \mathbf{y} + \frac{1}{2}\mathbf{y}^\top \mathbf{y} + \frac{\lambda}{2}\mathbf{a}^\top K \mathbf{a}$$

with

$$K_{nm} = \phi(\mathbf{x}_n)^\top \phi(\mathbf{x}_m) = \kappa(\mathbf{x}_n, \mathbf{x}_m)$$

Solving for a_n gives $a = (K + \lambda I)^{-1}\mathbf{y}$ and finally the regression model is :

$$y(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x}) = \mathbf{k}(x)^\top (K + \lambda I)^{-1}\mathbf{y}$$

with $\mathbf{k}(x)$ the vector with component $\kappa(\mathbf{x}, \mathbf{x}_n)$

Examples of kernels

Name	Kernel $\kappa(\mathbf{x}, \mathbf{x}')$
Linear kernel	$(\mathbf{x}^\top \mathbf{x}')^2$
Polynomial kernel	$(\mathbf{x}^\top \mathbf{x}' + 1)^D$
Gaussian kernel	$\exp -\frac{1}{2\sigma^2}(\mathbf{x} - \mathbf{x}')^\top(\mathbf{x} - \mathbf{x}')$
Laplace kernel	$\exp -\frac{1}{\sigma}\sqrt{(\mathbf{x} - \mathbf{x}')^\top(\mathbf{x} - \mathbf{x}')}$

Kernel trick regression : remarks

Instance based learning

- New prediction is obtained as a linear combination of target values from the training set
- Model requires kernel computation over all samples at evaluation

$$y(\mathbf{x}) = \mathbf{k}(x)^\top (K + \lambda I)^{-1}\mathbf{y}$$
$$\mathbf{k}(x) = (\kappa(\mathbf{x}, \mathbf{x}_n))$$
$$K = (\kappa(\mathbf{x}_n, \mathbf{x}_m))$$

Kernel trick

- Solution only depends on kernel $\kappa(\mathbf{x}, \mathbf{x}')$ and not feature maps $\phi(\mathbf{x})$
- Support vector machine (SVM) is built following similar principle

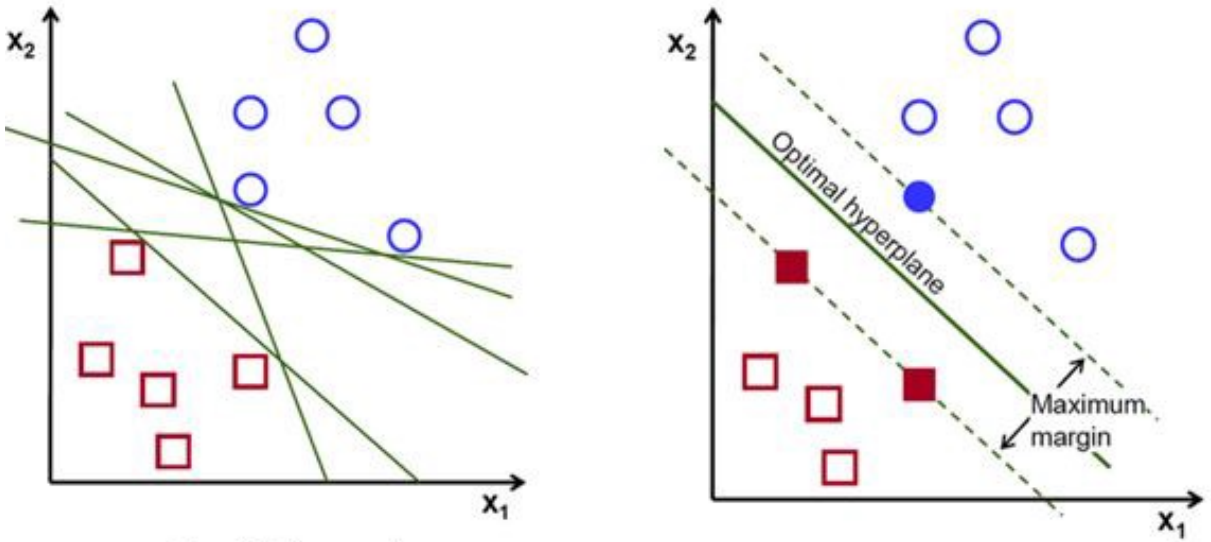
Support vector machines : maximum margin classifiers

Linearly separable case

Considering the two-class classification with the model in feature space $\phi(\mathbf{x})$ of the form :

$$y(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x}) + b$$

Denote by $t_n = +1$ if $y(\mathbf{x}) > 0$ and $t_n = -1$ if $y(\mathbf{x}) < 0$. Assuming samples are linearly separable many solutions can be found. Support vector machine looks for the unique maximal margin solution.



Posing SVM's max-margin classification problem

Distance between $\phi(\mathbf{x})$ and hyperplane $y(\mathbf{x}) = 0$

$$\frac{|y(\mathbf{x}_n)|}{\sqrt{\mathbf{w}^\top \mathbf{w}}} = \frac{t_n y(\mathbf{x}_n)}{\sqrt{\mathbf{w}^\top \mathbf{w}}} = \frac{t_n(\mathbf{w}^\top \phi(\mathbf{x}_n) + b)}{\sqrt{\mathbf{w}^\top \mathbf{w}}}$$

Max-margin problem

$$\arg\max_{\mathbf{w},b} \left(\frac{1}{\sqrt{\mathbf{w}^\top \mathbf{w}}} \min_n [t_n(\mathbf{w}^\top \phi(\mathbf{x}_n) + b)] \right)$$

- Problem is invariant to rescaling of $\mathbf{w}$, b
- The distance to the closest point can be set to $t_n(\mathbf{w}^\top \phi(\mathbf{x}_n) + b) = 1$
- Distance of any points to hyperplane verify $t_n(\mathbf{w}^\top \phi(\mathbf{x}_n) + b) \geq 1$

$$\begin{cases} \arg\max_{\mathbf{w},b} \frac{1}{\sqrt{\mathbf{w}^\top \mathbf{w}}} \\ t_n(\mathbf{w}^\top \phi(\mathbf{x}_n) + b) \geq 1, \forall n \end{cases} \iff \begin{cases} \arg\min_{\mathbf{w},b} \frac{1}{2} \mathbf{w}^\top \mathbf{w} \\ t_n(\mathbf{w}^\top \phi(\mathbf{x}_n) + b) \geq 1, \forall n \end{cases}$$

SVMs : solving max-margin classification problem

Loss function introducing Lagrange multipliers $\mathbf{a} = (a_1, a_2, ..., a_N)$

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}\mathbf{w}^\top \mathbf{w} - \sum_{n=1}^{N} a_n[t_n(\mathbf{w}^\top \phi(\mathbf{x}_n) + b) - 1]$$

$$\nabla_{\mathbf{w}} L = 0 \text{ implies } \mathbf{w} = \sum_{n=1}^{N} a_n t_n \phi(\mathbf{x}_n)$$

Solution replacing $\mathbf{w}$

$$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n \phi(\mathbf{x})^\top \phi(\mathbf{x}_n) + b = \sum_{n=1}^{N} a_n t_n \kappa(\mathbf{x}, \mathbf{x}_n) + b$$

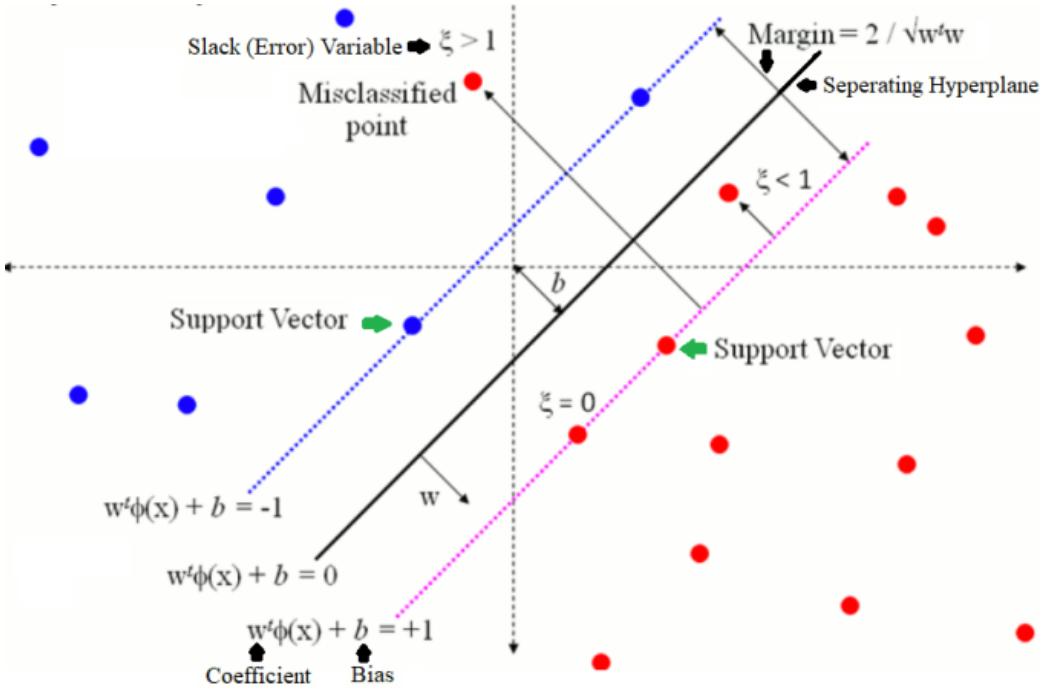
- Karush Kuhn Tucker (KKT) conditions : if $t_n \phi(\mathbf{x}_n) \neq 1$, $a_n = 0$
- Only points $\mathbf{x}_n$ lying on the margin (support vectors) count

Non linearly separable cases

Slack variables $\xi_1, \xi_2, ..., \xi_N$

$$\begin{cases} \arg\min_{\mathbf{w},b} \frac{1}{2}\mathbf{w}^\top \mathbf{w} + C\sum_{n=1}^{N} \xi_n \\ t_n(\mathbf{w}^\top \phi(\mathbf{x}_n) + b) \geq 1 - \xi_n, \forall n \end{cases}$$

- Slack variables allows points to be on the wrong side of the decision plane



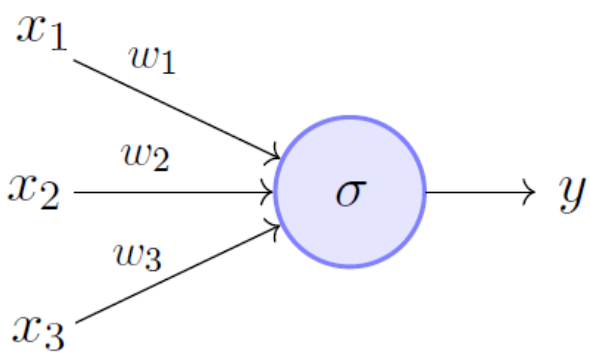
Overview

Artificial neurones

An artificial neurone implements the function

$$\begin{aligned}\sigma(\mathbf{x}) &= \frac{1}{1+e^{-(w_1x_1+w_2x_2+...+w_Dx_D-b)}} \\ &= \frac{1}{1+e^{-\mathbf{w}^t\mathbf{x}-b}}\end{aligned}$$

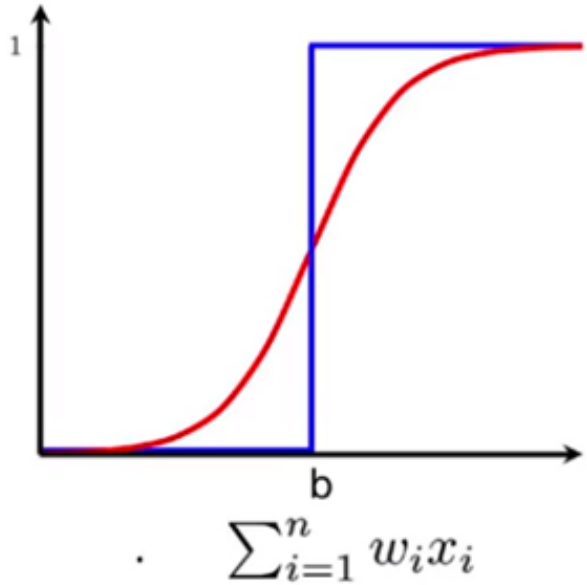
- $\mathbf{x} = (x_1, x_2, ..., x_D)^t$: input vector
- $\mathbf{w} = (w_1, w_2, ..., w_D)^t$: weight vector
- b : bias
- $\sigma(\mathbf{x})$: sigmoid activation function



Activitation of an artificial neurone

The articial neurone modulates its input using it activation function

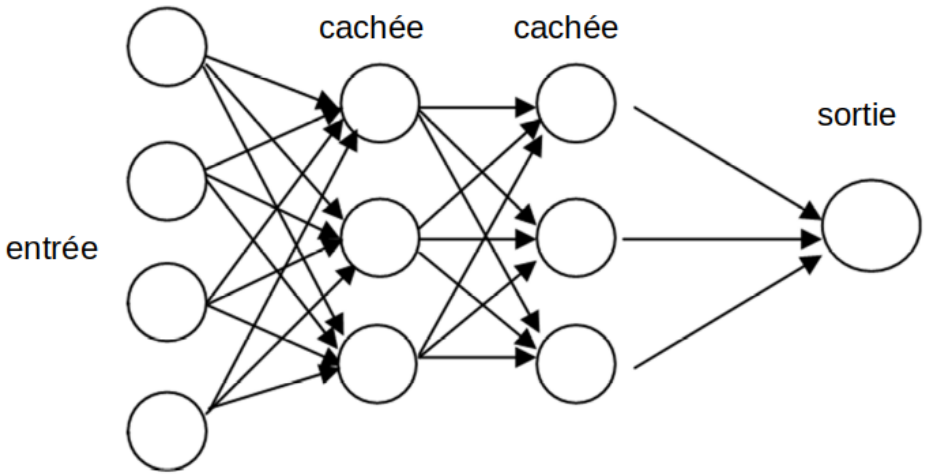
- b : is a threshold
- Si $\mathbf{w}^t\mathbf{x} > b$: neurone sends value 1.0
- Si $\mathbf{w}^t\mathbf{x} < 0$: neurone sends value 0



Neural network : multi layer perceptron (MLP)

A neural network is a system of connected artificial neurones

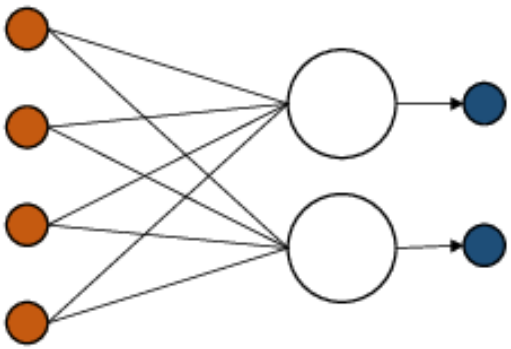
- Layers : neurones at the same level
- Types of layers : input, hidden, outputs
- Hidden layers : network internal layers



Types of neural networks

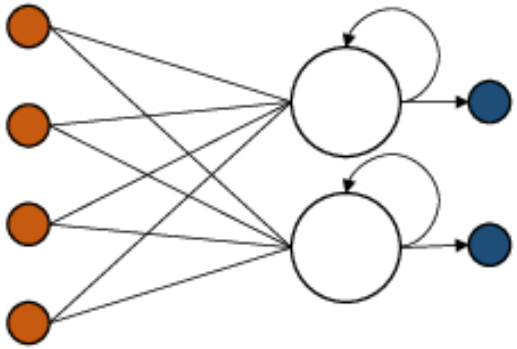
Feed-forward networks

- Without cycles : information flow from input to output layers
- Used for static prediction : image recognition, spam prediction



Recurrent networks

- Networks with cycles : temporal feedback loops
- Used for tasks requiering sequential information : machine translation, stock price prediction



Feed-forward networks : formally

Inputs : $\mathbf{x} = (x_1, ..., x_D)$

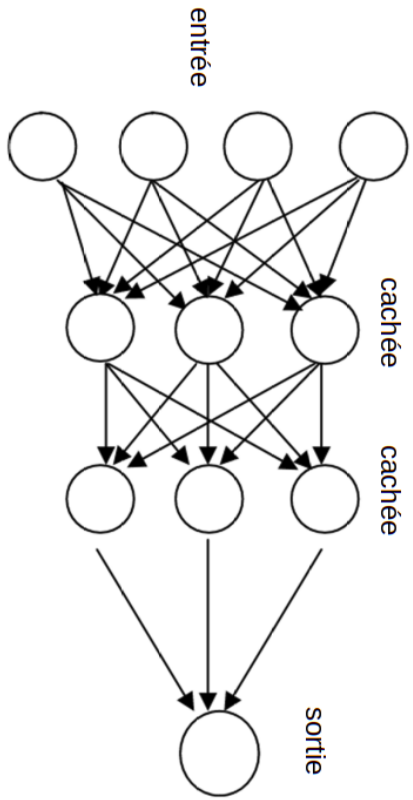
Hidden layer n°1 : $\mathbf{y}^{(1)} = \left(y_1^{(1)}, y_2^{(1)}, ..., y_{K_1}^{(1)}\right)$

$$y_k^{(1)} = \phi\left(\sum_{d=1}^{D} w_{1d} x_d + w_{10}\right)$$

Hidden layer n°l : $\mathbf{y}^{(l)} = \left(y_1^{(l)}, y_2^{(l)}, ..., y_{K_l}^{(l)}\right)$

$$y_k^{(l)} = \phi\left(\sum_{s=1}^{K_{l-1}} w_{ls} y_s^{(l-1)} + w_{l0}\right)$$

Output layer : $\mathbf{o} = \left(y_1^{(L)}, y_2^{(L)}, ..., y_{K_L}^{(L)}\right)$

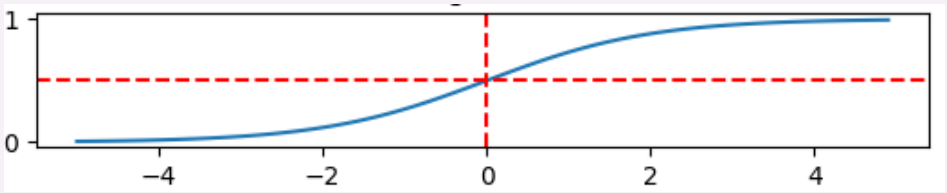


Activations functions $\phi(r)$

- Give to neural networks their nonlinear structures
- Without activation functions neural networks are linear regressions
- Previous layer information is modulated by activation function

Sigmoid function

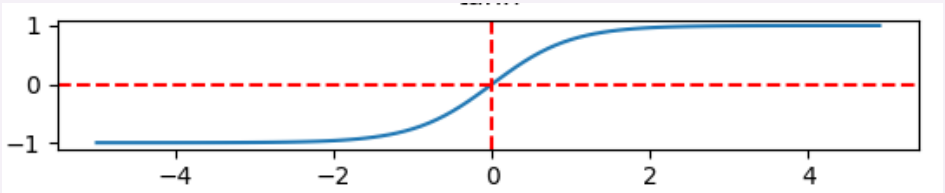
$$\phi(r) = \frac{1}{1 + e^{-r}}$$



Activation function : hyperbolic tangent

Hyperbolic tangent : tanh

$$\phi(\mathbf{r}) = \frac{e^{r} - e^{-r}}{e^{r} + e^{-r}}$$

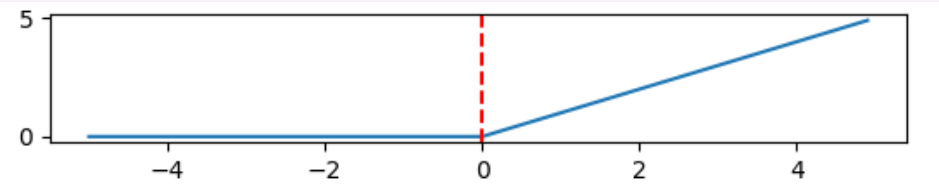


Activations to modulate networks response between $[-1, 1]$

Rectified linear activation unit

Rectified linear unit : relu

$$\phi(r) = \max(0, r)$$



Most used activation nowadays

Softmax activation function

Softmax unit

Given K real values r_1, r_2, ..., r_K

$$\phi(r_k) = \frac{e^{r_k}}{\sum_{l=1}^{K} e^{r_l}}$$

- Activation for a multi-class classifier outputs $K \geq 2$
- If $K = 2$ softmax function is equivalent to a sigmoid function

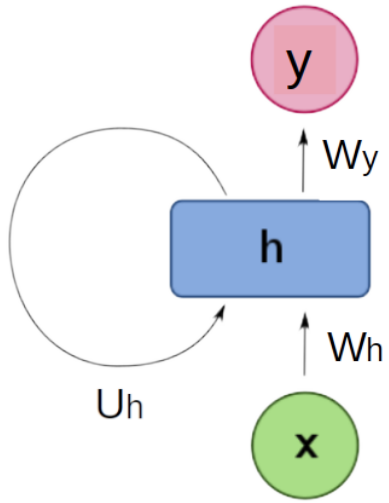
Recurrent neural networks

Reccurent unit

- $\mathbf{x}_t$: input vector
- $\mathbf{h}_t$: hidden variable
- $\mathbf{y}_t$: output vector
- W, U, b : weight and bias
- ϕ_h, ϕ_y : Activation functions

Reccurent unit equation

$$\begin{aligned} \mathbf{h}_t &= \phi_h(W_h \mathbf{x}_t + U_h \mathbf{h}_{t-1} + b_h) \\ \mathbf{y}_t &= \phi_o(W_y \mathbf{h}_t + b_y) \end{aligned}$$



Output types for neural networks

Binary classication

- Output activation : sigmoid fonction
- Defines probability of class 1

Multi-class classification K

- Output activation : softmax
- Defines probabilities for all K classes

Regression

- No output activation

Learning neural network parameters

Principles

- Supervised learning with a training datasets (x_n, y_n)
- Specification of a loss function allowing to compare desired outputs and network predictions
- Gradient descent on average loss over training samples

Feedforward networks

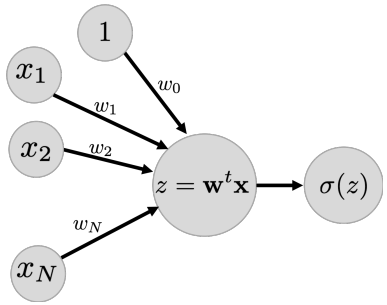
Error back propagation : retropropagate parameters updates from outputs to hidden layers to inputs

Reccurent networks

Temporal error back-propagation from final to initial time

Training a feedforward network without hidden layers

- Activation function : sigmoid $\sigma(u) = \frac{1}{1+e^{-u}}$
- Model output : $y(\mathbf{x}) = \sigma(\mathbf{w}^t\mathbf{x})$



Model training : estimation of parameters $\mathbf{w}$

- Loss function : cross-entropy :

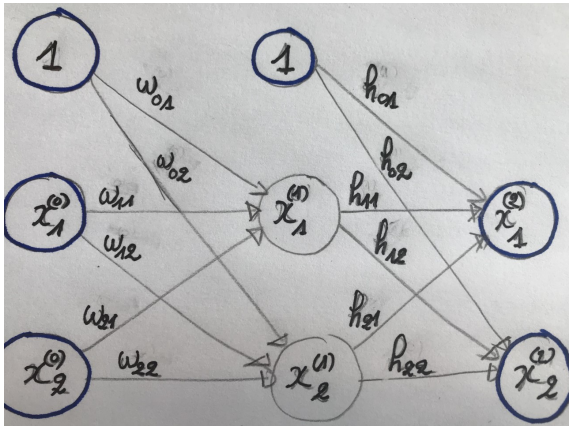
$$y_n \log \sigma(\mathbf{w}^t\mathbf{x_n}) + (1 - y_n) \log(1 - \sigma(\mathbf{w}^t\mathbf{x_n}))$$

- Average loss :

$$L(\mathbf{w}) = \frac{1}{N} \sum_{n}^{N} y_n \log \sigma(\mathbf{w}^t\mathbf{x_n}) + (1 - y_n) \log(1 - \sigma(\mathbf{w}^t\mathbf{x_n}))$$

- Find $\mathbf{w}$: using gradient descent $\mathbf{w}_t = \mathbf{w}_{t-1} - \alpha \nabla_{\mathbf{w}} L(\mathbf{w}_{t-1})$

Study of a feed-forward network with a single hidden layer



From inputs to hidden layers

$$x_1^{(1)} = \sigma\left(w_{01} + w_{11}x_1^{(0)} + w_{21}x_2^{(0)}\right) = \sigma(z_1^{(1)}) \text{ with } z_1^{(1)} = \mathbf{w}_{.1}^t\mathbf{x}^{(0)}$$

$$x_2^{(1)} = \sigma\left(w_{02} + w_{12}x_1^{(0)} + w_{22}x_2^{(0)}\right) = \sigma(z_2^{(1)}) \text{ with } z_2^{(1)} = \mathbf{w}_{.2}^t\mathbf{x}^{(0)}$$

From hidden to output layers

$$x_1^{(2)} = \sigma\left(h_{01} + h_{11}x_1^{(0)} + h_{21}x_2^{(0)}\right) = \sigma(z_1^{(2)}) \text{ with } z_1^{(2)} = \mathbf{h}_{.1}^t\mathbf{x}^{(1)}$$

$$x_2^{(2)} = \sigma\left(h_{02} + h_{12}x_1^{(0)} + h_{22}x_2^{(0)}\right) = \sigma(z_2^{(2)}) \text{ with } z_2^{(2)} = \mathbf{h}_{.2}^t\mathbf{x}^{(1)}$$

Network with a hidden layer : loss for a sample

Quadratic error

$$L(\mathbf{w}, \mathbf{h}) = \sum_{d=1}^{2} \frac{\left(y_d - x_d^{(2)}\right)^2}{2}$$

- Quadratic error between labels and network predictions
- Network parameters to optimize for $\mathbf{w}$ and $\mathbf{h}$

Training by gradient error back propagation 1/3

Compute gradient with respect to $\mathbf{h}_{.i}$

$$
\begin{aligned}
\nabla_{\mathbf{h}_{.i}} L(\mathbf{w}, \mathbf{h}) &= \nabla_{\mathbf{h}_{.i}} \sum_{d=1}^{2} \frac{\left(y_d - x_d^{(2)}\right)^2}{2} \\
&= \left(y_i - \sigma(z_i^{(2)})\right) \nabla_{\mathbf{h}_{.i}} \sigma\left(z_i^{(2)}\right) \\
&= \left(y_i - x_i^{(2)}\right) \sigma'\left(z_i^{(2)}\right) \nabla_{\mathbf{h}_{.i}} z_i^{(2)} \\
&= \left(y_i - x_i^{(2)}\right) \sigma'\left(z_i^{(2)}\right) \mathbf{x}^{(1)} \\
&= e_i^{(2)} \mathbf{x}^{(1)}
\end{aligned}
$$

where $e_i^{(2)} = \sigma'\left(z_i^{(2)}\right)\left(y_i - x_i^{(2)}\right)$

Training by gradient error back propagation 2/3

Compute gradient with respect to $\mathbf{w}_{.j}$

$$\begin{aligned}
\nabla_{\mathbf{w}_{.j}} L(\mathbf{w}, \mathbf{h}) &= \nabla_{\mathbf{w}_{.j}} \sum_{i=1}^{2} \frac{\left(y_i - x_i^{(2)}\right)^2}{2} \\
&= \sigma'\left(z_j^{(1)}\right) \sum_i h_{ji} e_i^{(2)} \mathbf{x}^{(0)} \\
&= e_j^{(1)} \mathbf{x}^{(0)}
\end{aligned}$$

where $e_j^{(1)} = \sigma'\left(z_j^{(1)}\right) \sum_i h_{ji} e_i^{(2)}$

Training by gradient error back propagation 3/3

Error back-propagation

1. Select sample $\mathbf{x}^{(0)}$ and output $\mathbf{y} = (y_i)$
2. Use $\mathbf{x}^{(0)}$ to compute network activations $z_j^{(1)}$, $x_j^{(1)}$, $z_i^{(2)}$, $x_i^{(2)}$
3. Retro-propagate errors from outputs to inputs :
   1. Compute $e_i^{(2)} = \sigma'\left(z_i^{(2)}\right)\left(y_i - x_i^{(2)}\right)$
   2. Compute $e_j^{(1)} = \sigma'\left(z_j^{(1)}\right)\sum_i h_{ji} e_i^{(2)}$
4. Compute gradient :
   - Compute $\nabla_{\mathbf{w}_{.j}} L(\mathbf{w}, \mathbf{h}) = e_j^{(1)} \mathbf{x}^{(0)}$
   - Compute $\nabla_{\mathbf{h}_{.i}} L(\mathbf{w}, \mathbf{h}) = e_i^{(2)} \mathbf{x}^{(1)}$
5. Update network parameters :
   - Update $\mathbf{h}_{.i} = \mathbf{h}_{.i} - \alpha \nabla_{\mathbf{h}_{.i}} L(\mathbf{w}, \mathbf{h})$
   - Update $\mathbf{w}_{.j} = \mathbf{w}_{.j} - \alpha \nabla_{\mathbf{w}_{.j}} L(\mathbf{w}, \mathbf{h})$
6. Loop back to step 1

Convolutional neural networks CNN : Yann Lecun

- Specially designed for image processing
- Network weights are convolved with image to be processed
- Top performing network between 2014 to 2022

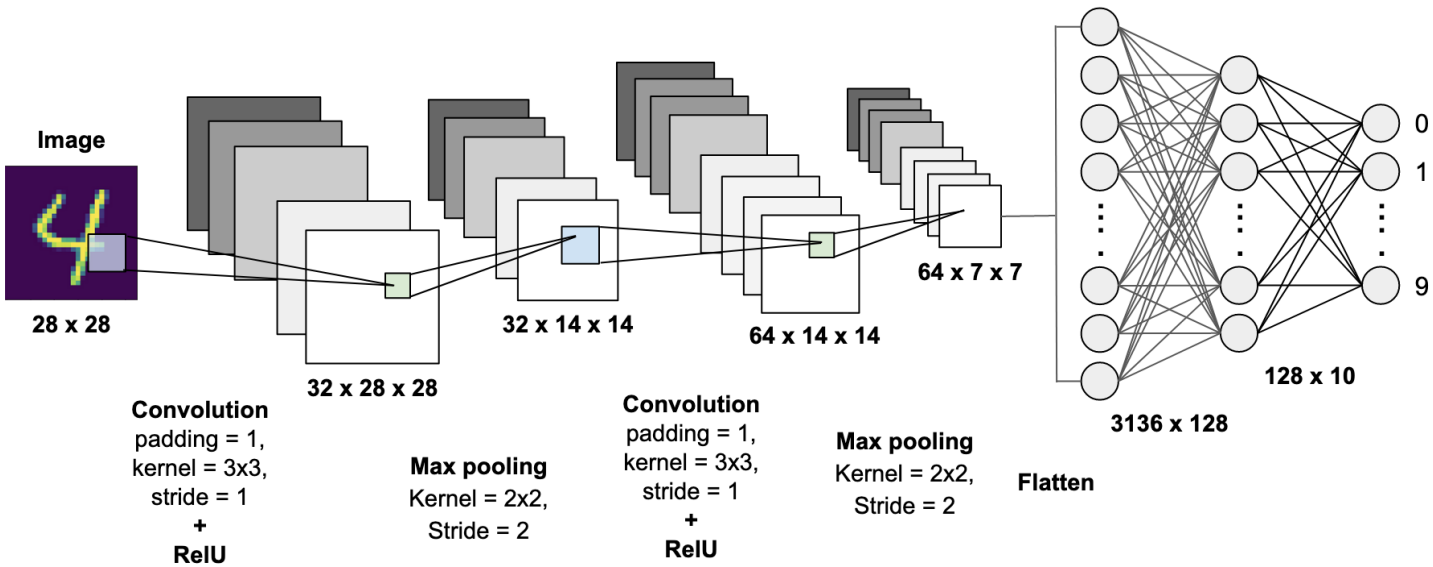
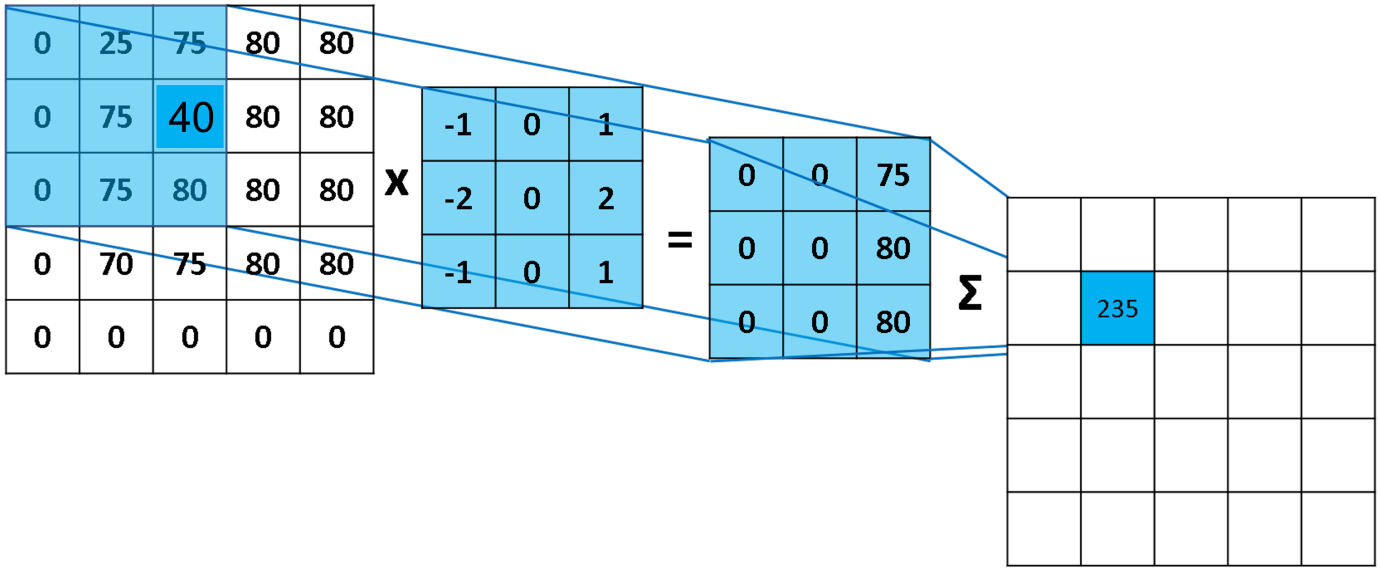
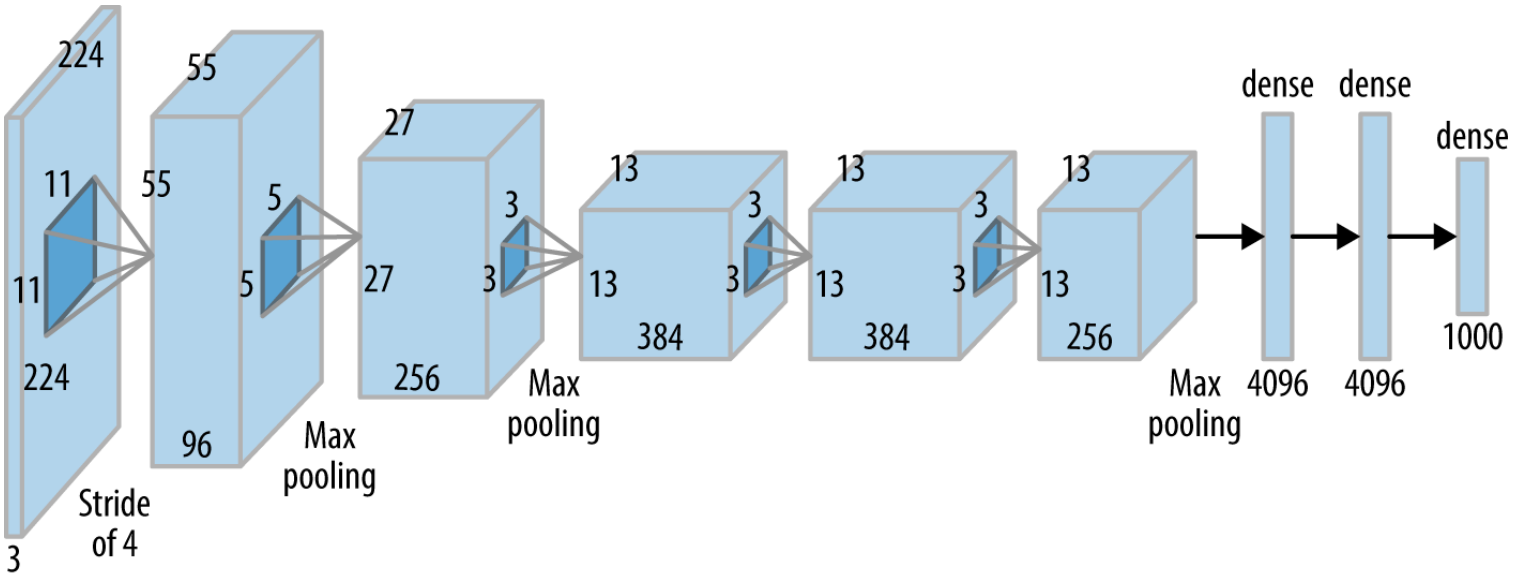


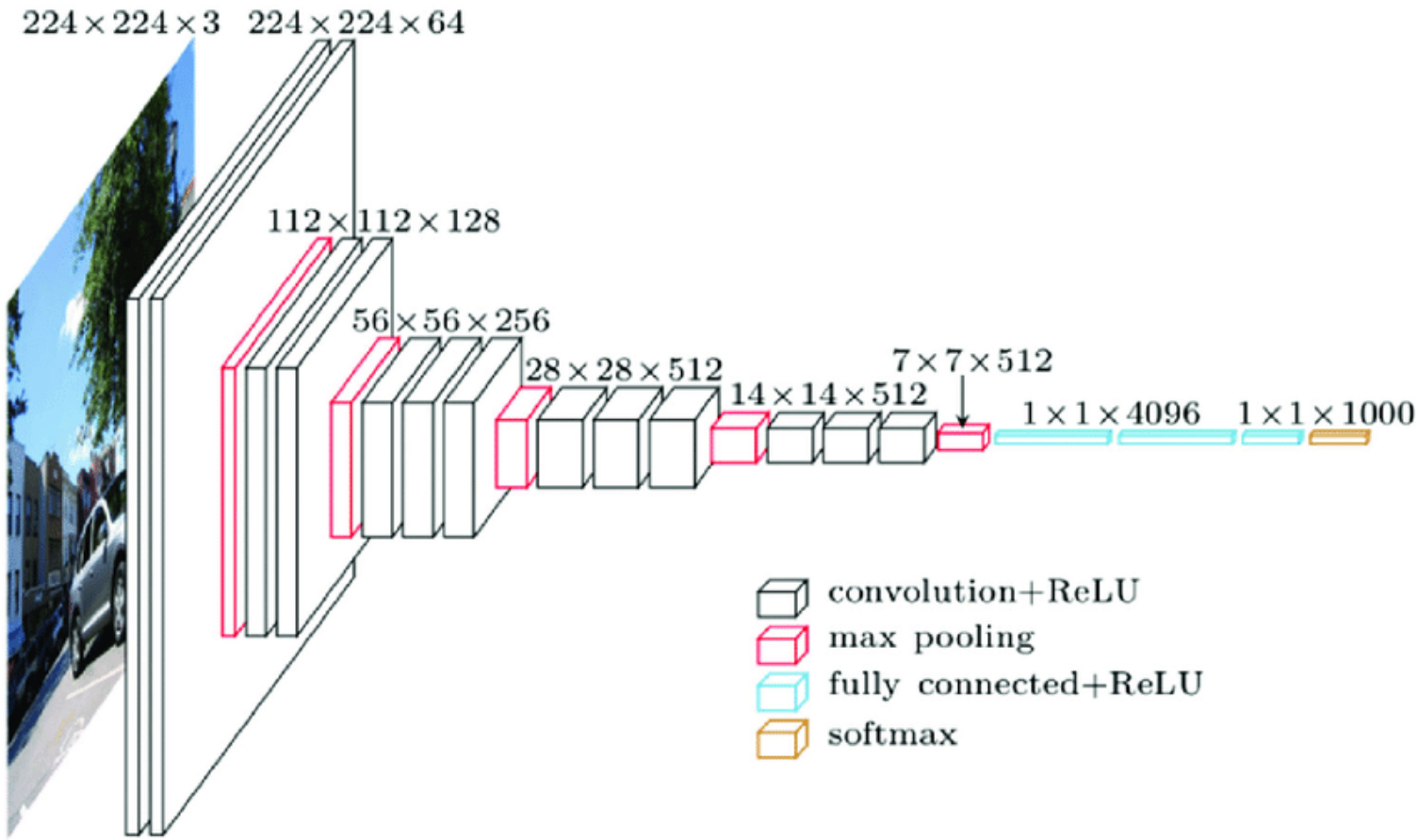
Image convolution by network weights



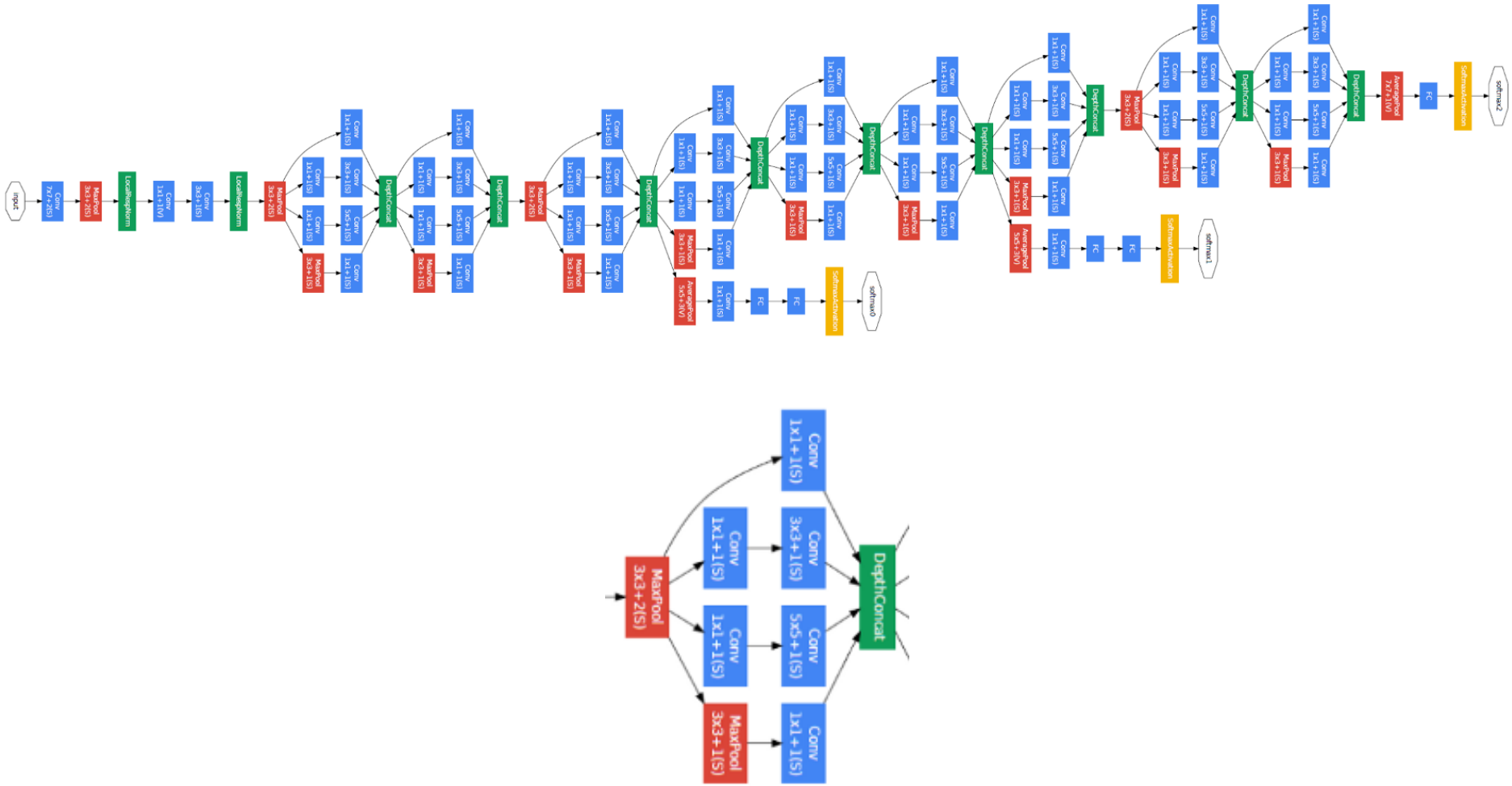
AlexNet : University Toronto Machine Learning Group



VGG-Net-16 : Oxford Visual Geometry Group

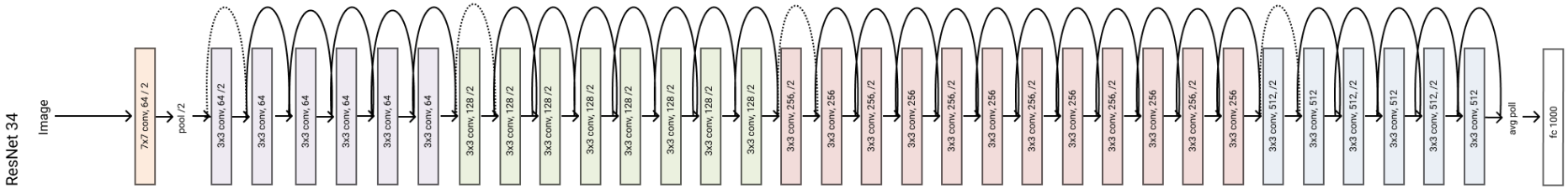


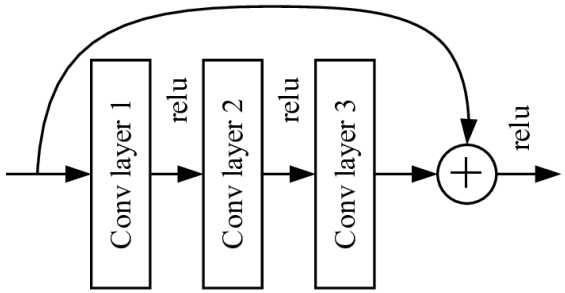
Inception Network : Google



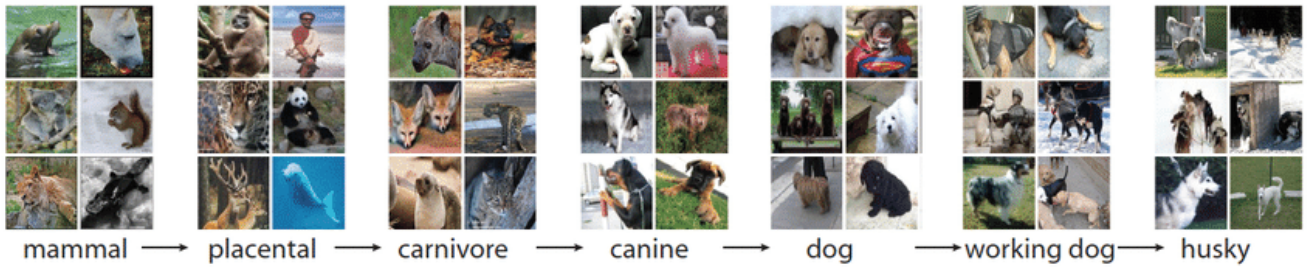
Residual networks (resnet) : MSR Asia

- Deepest networks
- Deepest networks can reached 150 hidden layers
- Specificity : skip connections
- Since 2015 resnet hold best image classification performances measured in ImageNet Dataset



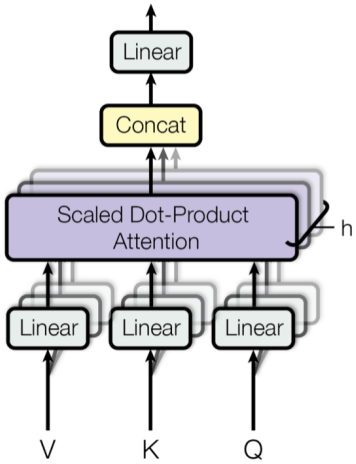


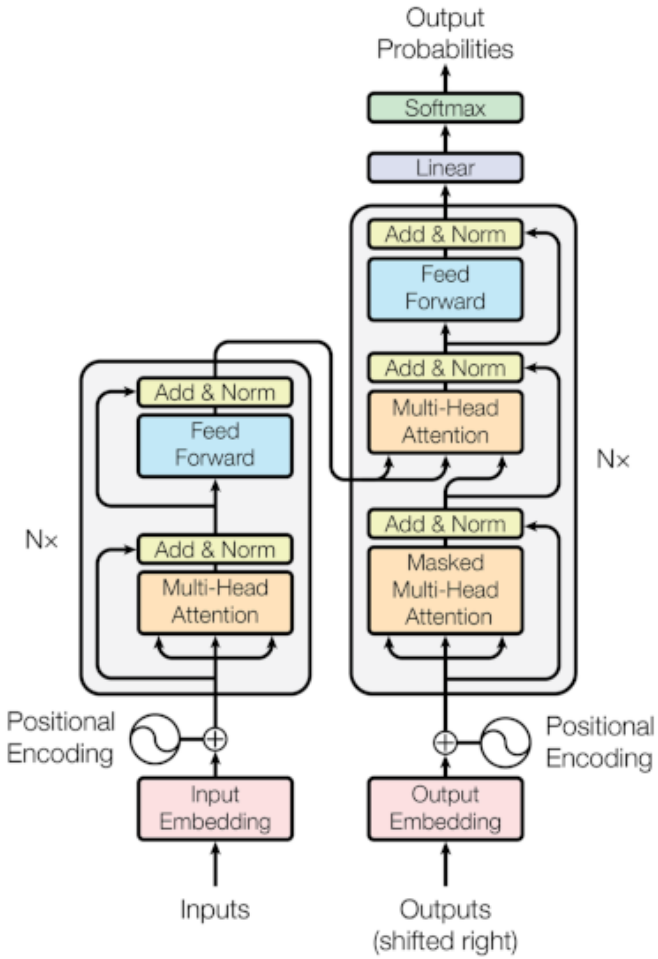
Challenge Imagenet



Transformers : Google Brain

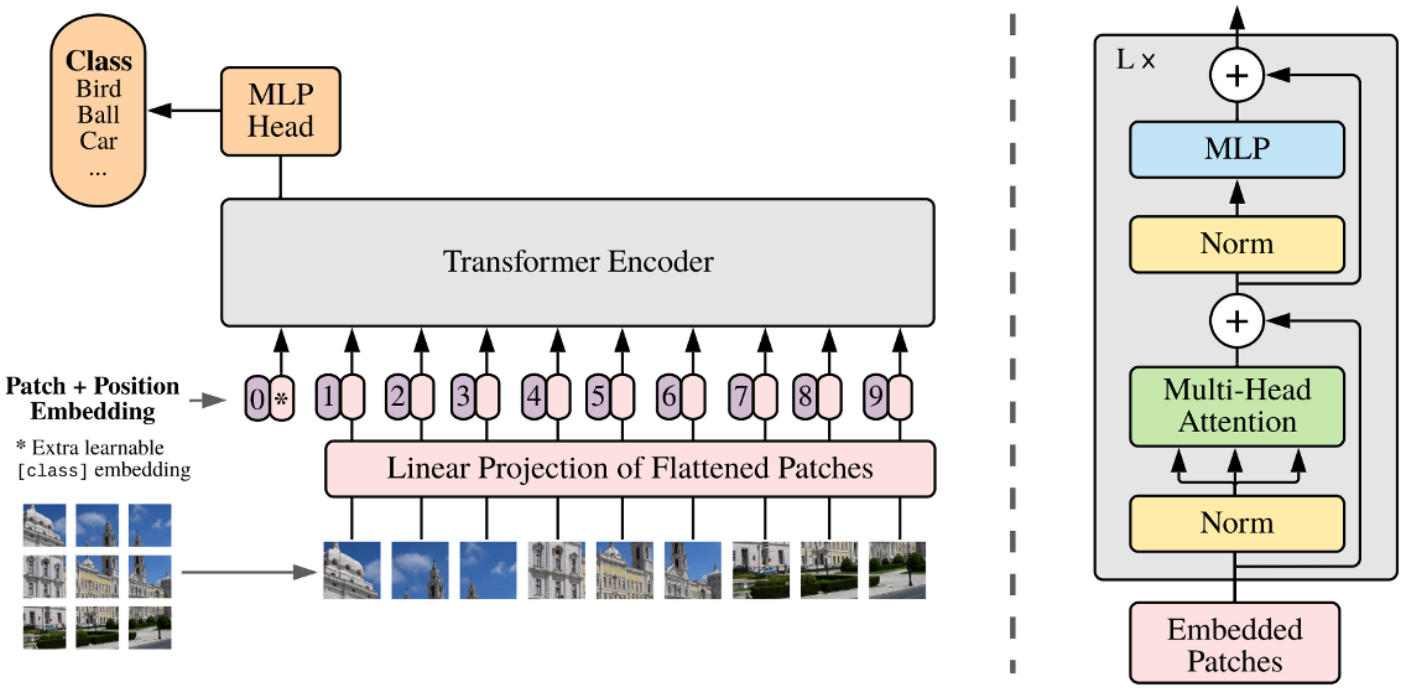
- Standard architecture for NLP :
  - BERT : Google Brain
  - GPT, Chat-GPT : Open AI
- Encoder-decoder
- Multi-head attention layer
  - Self-attention





Visual transformers : Facebook

- Input is an image
- Transformer encoder layer : based on multi-head attention
- Top performing model with CNN on computer visions tasks



Next class : lab session

- Professor Anindya Roy
- Supervised learning : stock price movements prediction