Neural Networks: Introduction to Network Training

Andrzej Kordecki

Neural Networks (ML.ANK385 and ML.EM05): Lecture 04
Division of Theory of Machines and Robots
Institute of Aeronautics and Applied Mechanics
Faculty of Power and Aeronautical Engineering
Warsaw University of Technology

Table of Contents

Loss function

Training a neural network

Designing and training a neural network is not much different from training any other machine learning model. Nearly all neural networks learning algorithms can be described by combination of:

- specification of a dataset,
- loss function,
- optimization procedure,
- model/structure of network.

These statements are supported in many tasks of both supervised and unsupervised learning.

Loss function

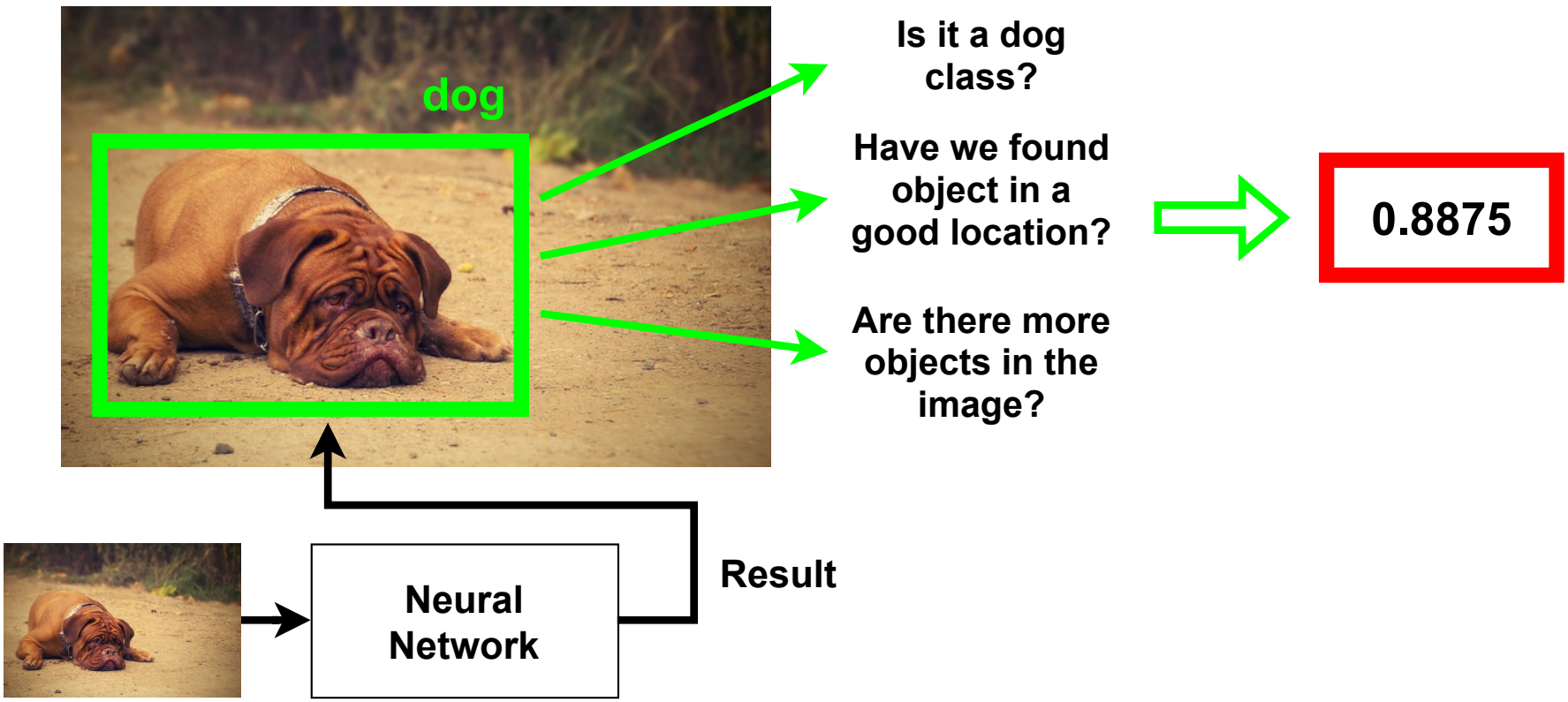
The cost function represents goals of our neural network.

The cost function (loss function, profit function, error function, objective function):

- It is a function used to evaluate a candidate solution. It allows us to judge if we have found a to our problem solution.
- It is a function that all the various aspects of a possibly complex system down to a single number (scalar value).
- We use it for neural network parameter estimation

Loss function

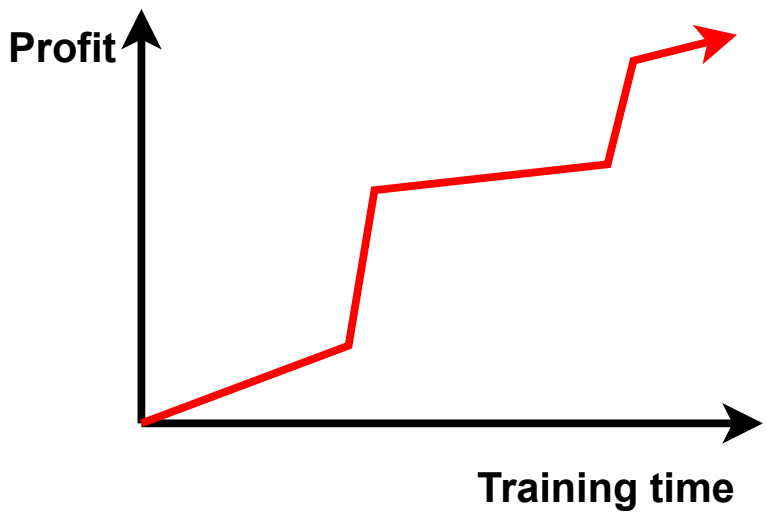
Example of object detection in image.

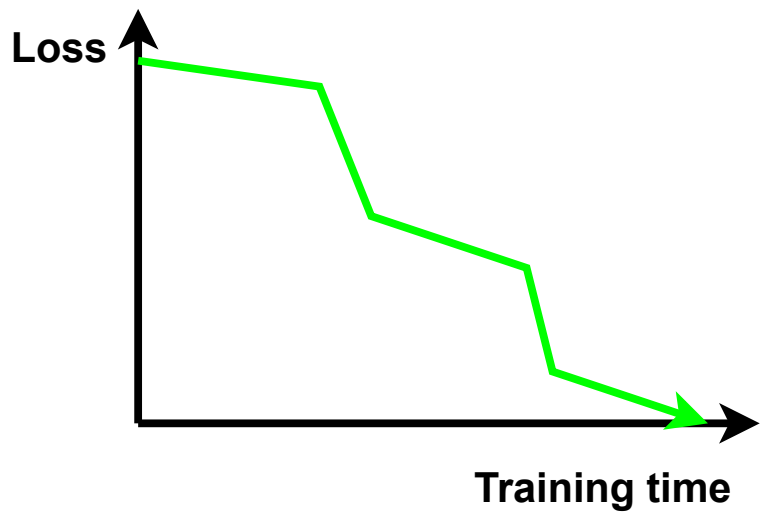


Loss function

The optimization problem seeks to:

- minimize value of loss function,
- maximize value of profit function.



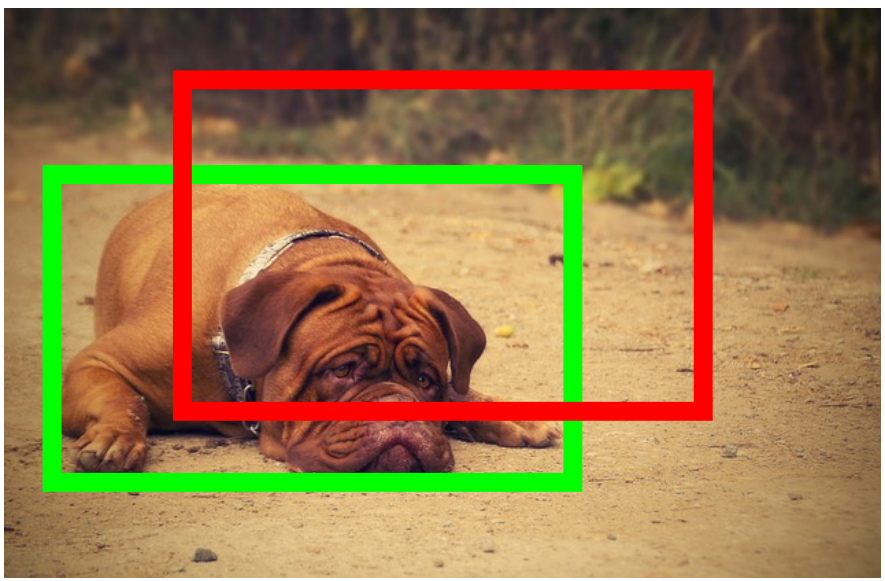


In neural networks, we seek to minimize the error.

Loss function

The loss function:

- Quantifies the amount by which the prediction values deviates from the actual values (error value),
- If we choose a poor error function and obtain unsatisfactory results, the fault is ours for badly specifying the goal of the search.



Object coordinates (x, y, dx, dy) deviates from the actual values:

difference = 0.1

or

difference = 10

Loss function

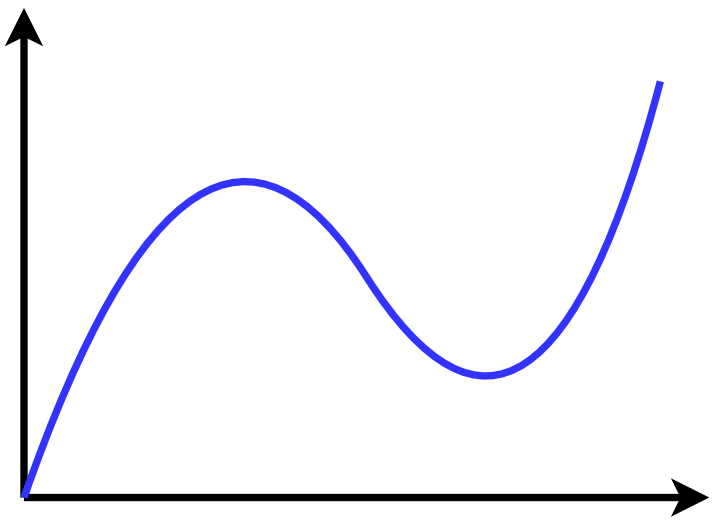
Loss function characteristics:

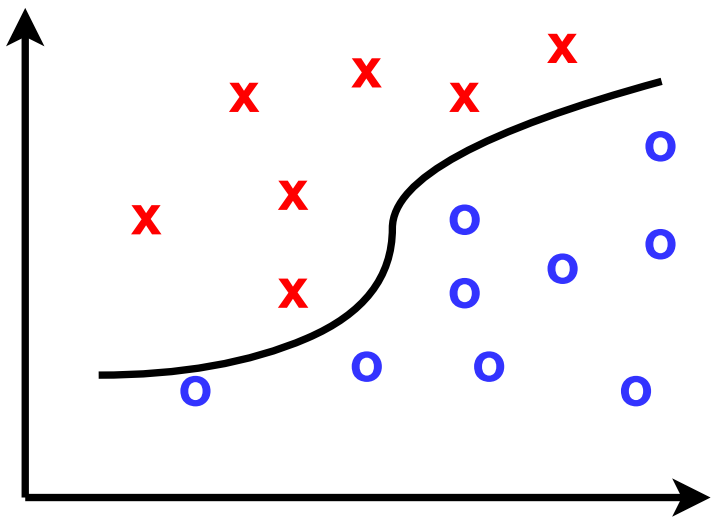
- There is no universal loss function that works in all kind of neural network tasks.
- The choice of loss function is directly related to the activation function used in the output layer of your neural network.
- We can replace any components of loss function mostly independently from the others networks settings - we can obtain a very wide variety of loss function.

Loss function

Loss functions in a supervised learning problem can be mainly classified into two categories:

- Regression losses,
- Classification losses.





Regression loss function

Regression loss function may take the form of function dependent on difference between desired output y and predicted NN output $\hat{y}$:

$$L = \ell(y - \hat{y})$$

The residuals (errors):

$$r = y - \hat{y} \rightarrow L = \ell(r)$$

Regression loss function

Examples of regression loss functions:

- Mean Square Error (MSE, L2 Loss):

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

- Mean Absolute Error (MAE, L1 Loss):

$$L = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

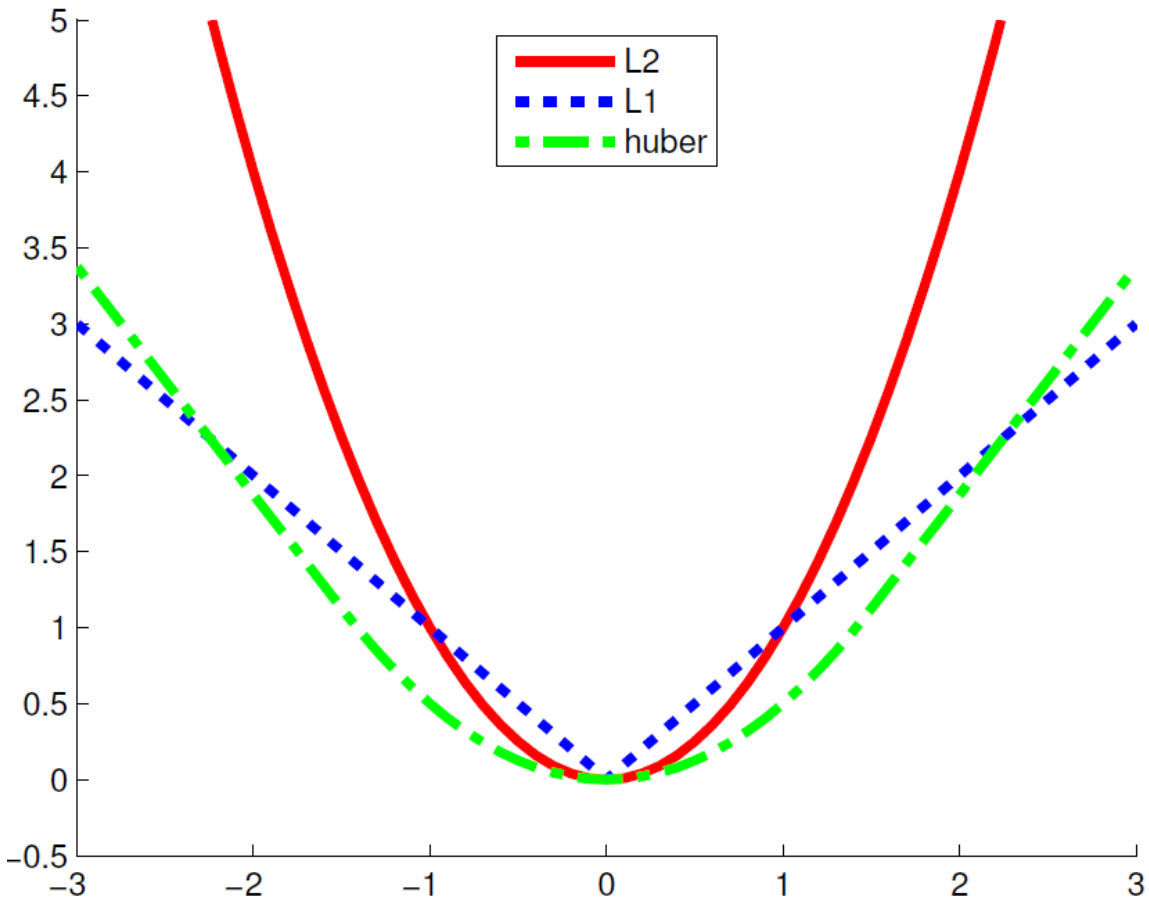
- Huber Loss:

$$L = \begin{cases} 0.5(y - \hat{y})^2 & \text{if } |y - \hat{y}| \leq c \\ c|y - \hat{y}| - 0.5c^2 & \text{otherwise} \end{cases}$$

Do we have to divide by N?

Regression loss function

The influence of error on the loss function.



Regression loss function

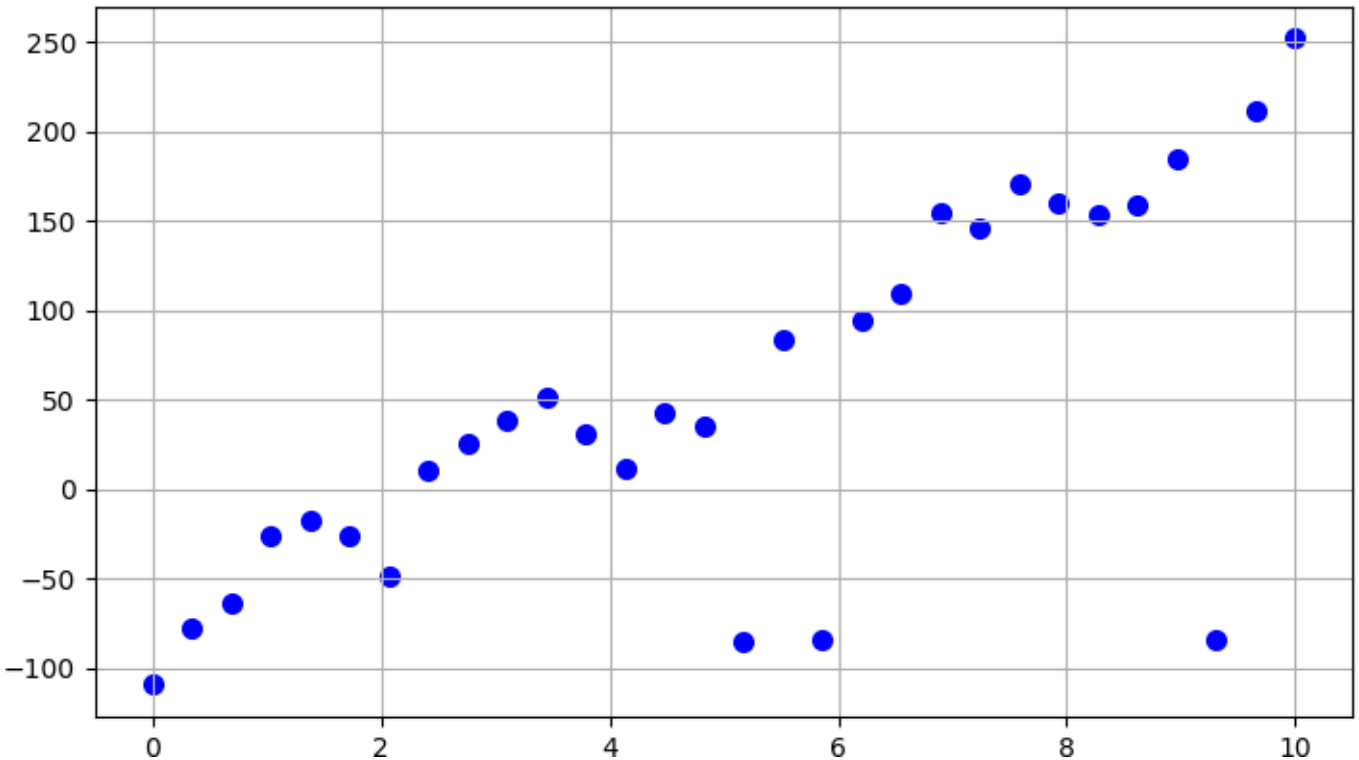
Example: Find the relation between water and its price?

water $[m^3]$ (x)	price [pln] (y)
19	70
38	152
44	164
77	276
99	292
...	...

Where should we start?

Regression loss function

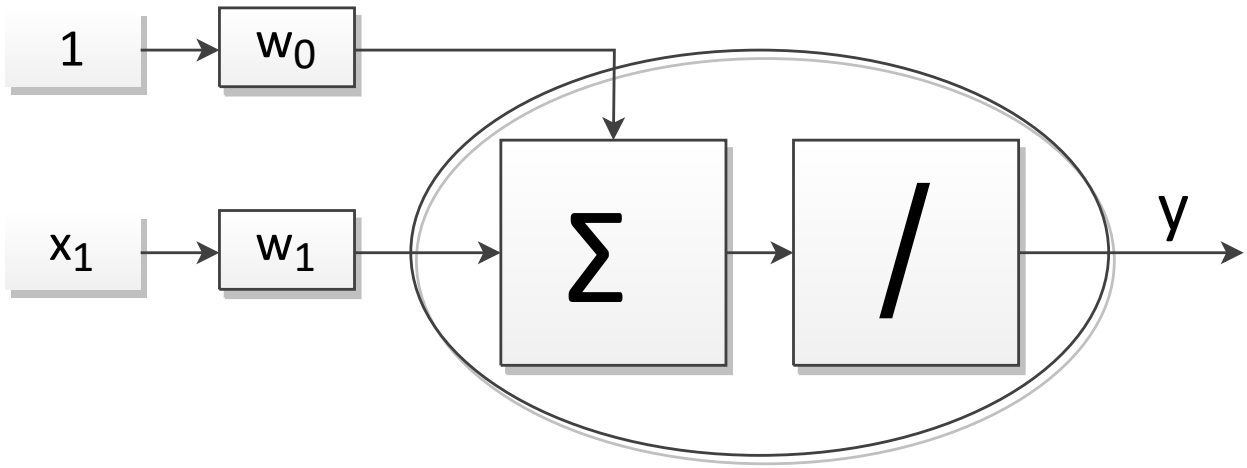
If possible we should make data visualization.



What neural network structure should we use?

Regression loss function

Neaural netwrok struture:



Output value of one neuron with linear activation function:

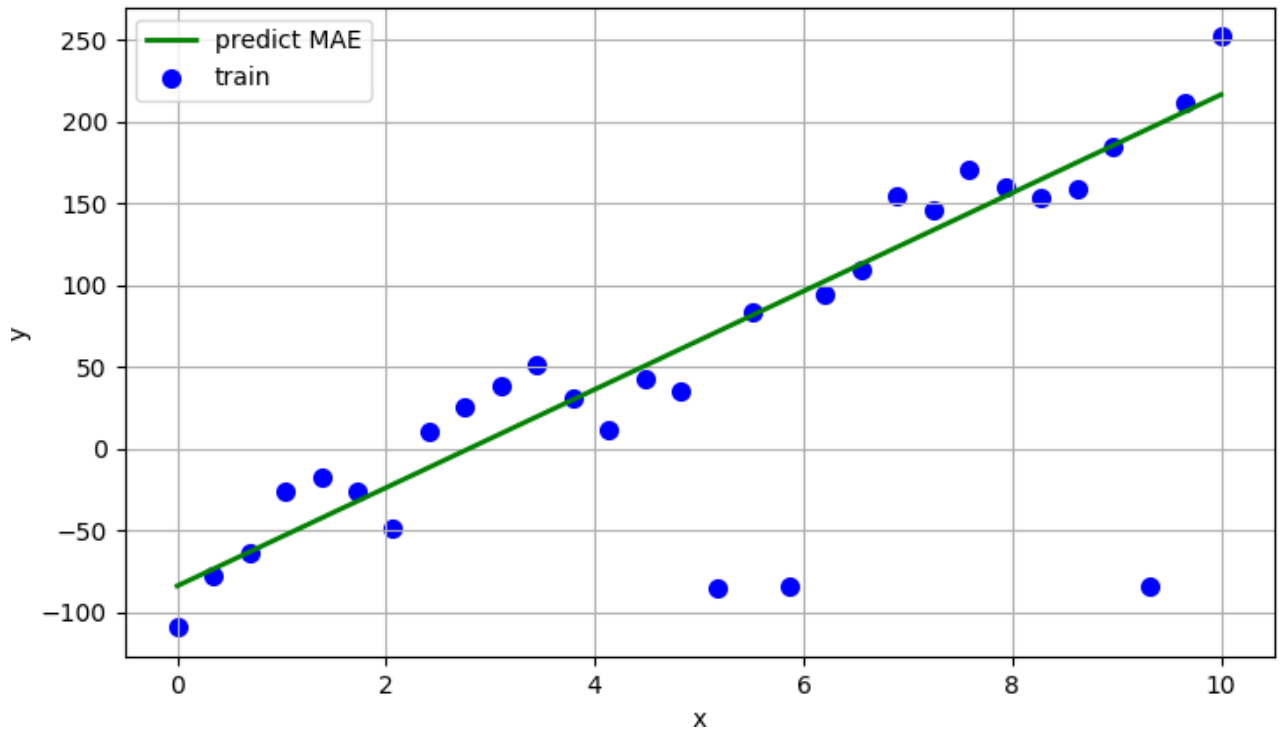
$$y = w_1 x_2 + w_0$$

Regression loss function

The hypothesis:

$h(w, x) = w_1 x_1 + w_0$

$\min_{w_0, w_1} \frac{1}{n} \sum_{k=1}^{n} |y - h(a, w)|$



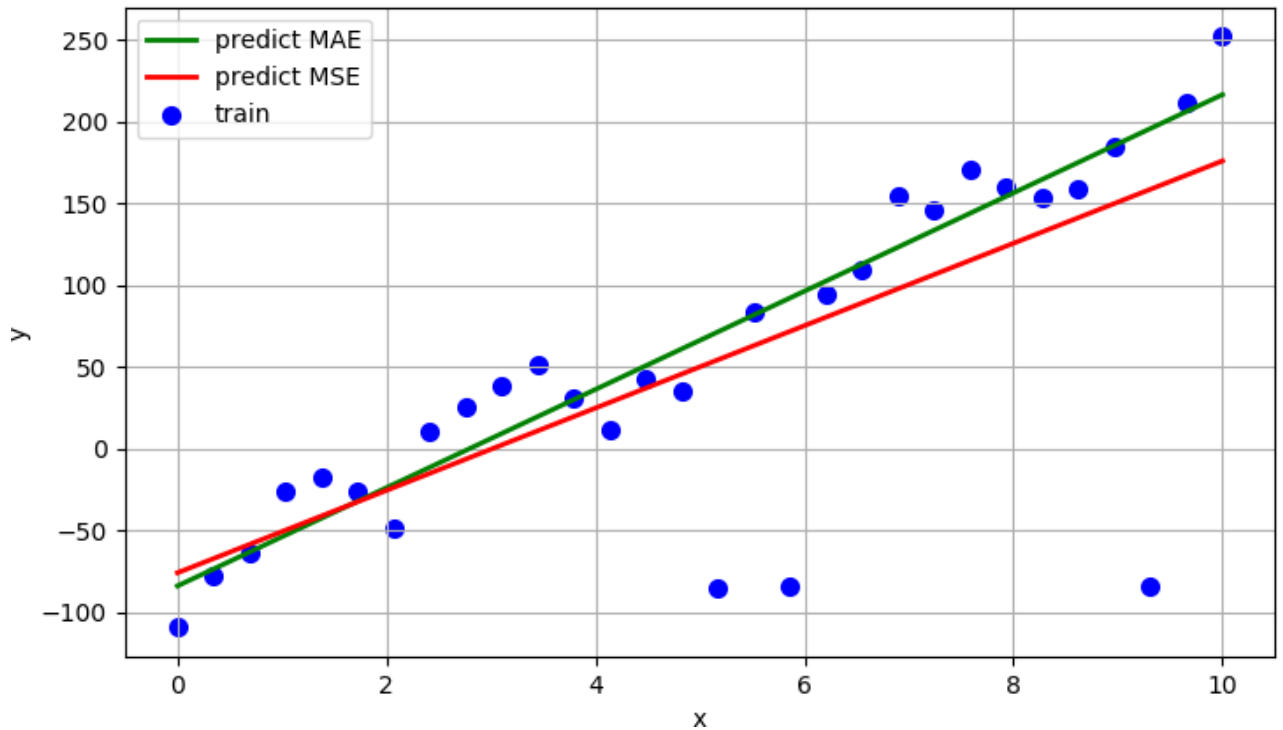
The goal: Choose w_0 and w_1 so that $h(w, x)$ is close to y.

Regression loss function

The hypothesis:

$h(w, x) = w_1 x_1 + w_0$

$\min_{w_0, w_1} \frac{1}{n} \sum_{k=1}^{n} [y - h(a, w)]^2$



The model with MSE loss give more weight to outliers than a model with MAE loss.

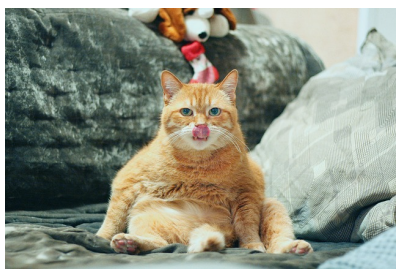
Classification loss function

Loss functions for classification:

- Output of classification problem is class score $L = f(x)$ for the input x.
- Magnitude of the score usually represents the "confidence" of our prediction.
- Target variable y, is a binary variable, 1 for true and -1 (0 or $f(x)$ - depends on loss function) for false ,

Dog?

0.88



Cat?

0.78

horse?

0.99

Classification loss function

- Given training data (x, y),
- Set of decisions is $t = \{-1, 1\}$, i.e. the set of classes,
- Loss function can be defined as delta function - zero-one loss:

$$L = H(f(x) \neq y)$$

i.e. $yf(x) > 0$ (checking signs) for a correct classification:

$$\begin{cases} f(x_i) \geq 0 \rightarrow & y_i = 1 \\ f(x_i) < 0 \rightarrow & y_i = -1 \end{cases}$$

H - Heaviside step function.

Classification loss function

Examples of binary loss functions for classification:

- Square loss ($t = \{-1, 1\}$)

$$L = \frac{1}{N}\sum_{i=1}^{N}(1 - tf(x_i))^2$$

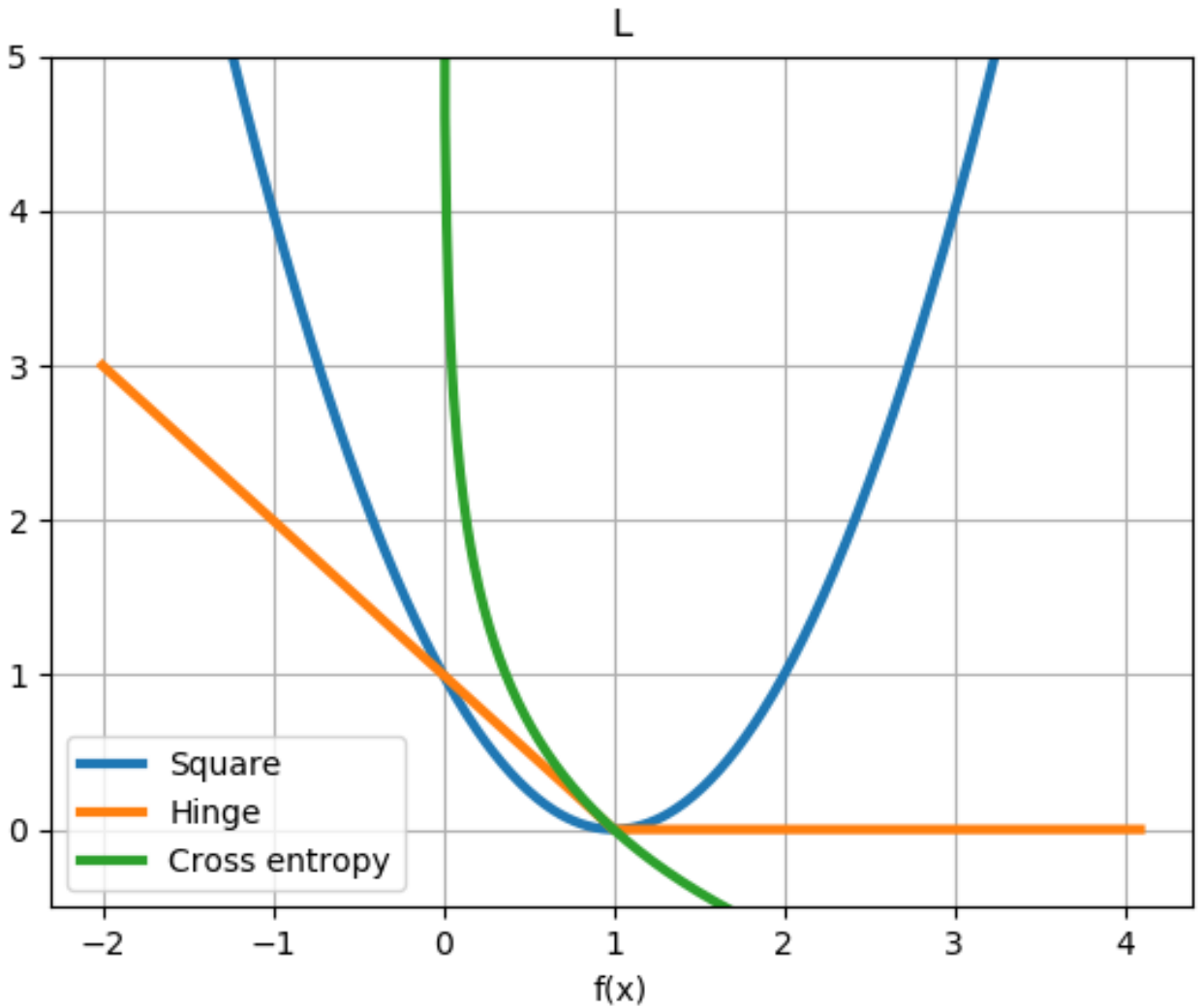
- Hinge loss ($t = \{-1, 1\}$)

$$L = \frac{1}{N}\sum_{i=1}^{N}\max(0, 1 - tf(x_i))$$

- Binary cross entropy loss ($t = \{0, 1\}$)

$$L = -\frac{1}{N}\sum_{i=1}^{N}[t_i \log(f(x_i)) + (1 - t_i)\log(1 - f(x_i))]$$

Classification loss function

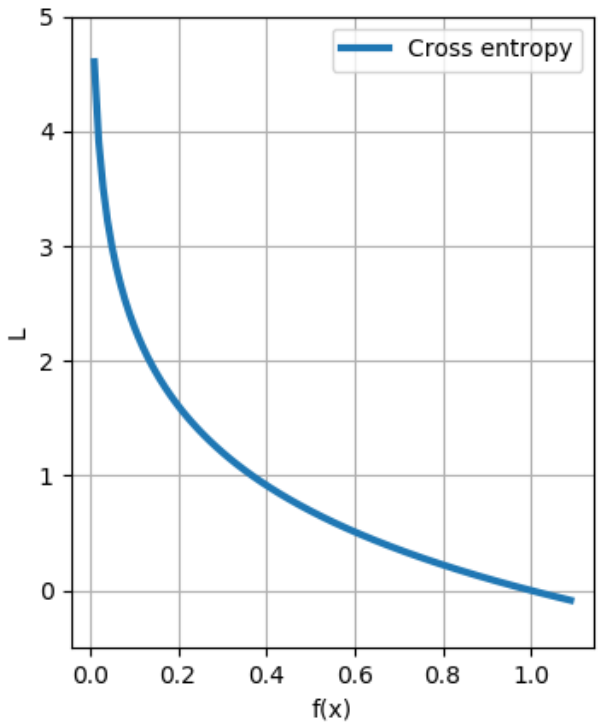


Classification loss function

Cross entropy loss function for multiple classes is one of the most used loss functions:

$$L = -\frac{1}{N}\sum_{k=1}^{N}\sum_{i=1}^{C} t_{k,i}\log(f(x_{k,i}))$$

where C is the number of classes and $f(x_{k,i}) \in < 0, 1 >$ is the predicted probability of the sample belonging to class t.

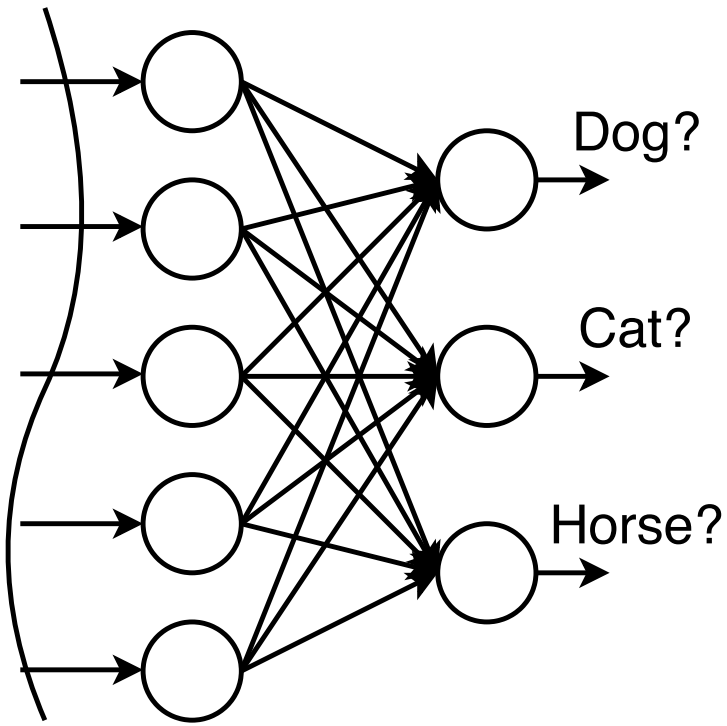


Cross entropy indicates the distance between what the output distribution and the original distribution.

Classification loss function

The soft-max layer is often used with cross-entropy loss function for 1-of-N output encoding. The function rescales the outputs, so that the activations sum to 1 and all of them lie between 0 and 1:

$$y_k = \frac{\exp(v_k)}{\sum_k \exp(v_k)}$$



Classification loss function

Multiclass Support Vector Machine loss for multiple classes:

$$L_i = \sum_{j \neq y_i} \max(0, 1 + f(x_j) - f(x_{y_i}))$$

- f is a scoring function (network output),
- sums over all incorrect classes ($j \neq y_i$),
- score vector will be positive when the class is predicted to be present

Classification loss function

Example of classification problem - network output:



plane	**-0.4**	-3.6	1
car	1.5	**2.7**	-2.1
ship	3	0.8	**-2**

Classification loss function

Calculation:

- Plane training example

$$L_1 = max(0, (1.5-(-0.4)+1)+max(0, (3-(-0.4)+1) = 7.3$$

Result: High loss - very wrong prediction

- Car training example

$$L_1 = max(0, (-3.6-2.7+1)+max(0, (0.8-2.7+1) = 0$$

Result: Zero loss - correct prediction

- Plane training ship

$$L_1 = max(0, (1-(-2)+1)+max(0, (-2.1-(-2)+1) = 5$$

Result: High loss - wrong prediction

Neural network evaluation

Neural network evaluation

Results quality measures:

- The criteria of neural network training quality depend on desired result: monotonicity, complexness, repetition, dimensionality, relative approximation simplicity, inconsistency, class separability, compactness and more.
- The basic measure of quality is loss function, but this is just one measure of obtained results.
- Depending on the goal of neural network we can divide quality measures more related with: approximation or classification.

What is an information for ANN?

Discrete variable X:

- $X : \Omega \rightarrow R$
- n random events that are countable,
- $w_i \in \Omega$ - result of a random event, i = 1,...,n,
- $x_i \in R$ - the real value assigned to the event,
- each w_i has probability assigned $p_i = P(w_i)$, $\sum_{i=1}^{n} p_i = 1$

Distribution measures:

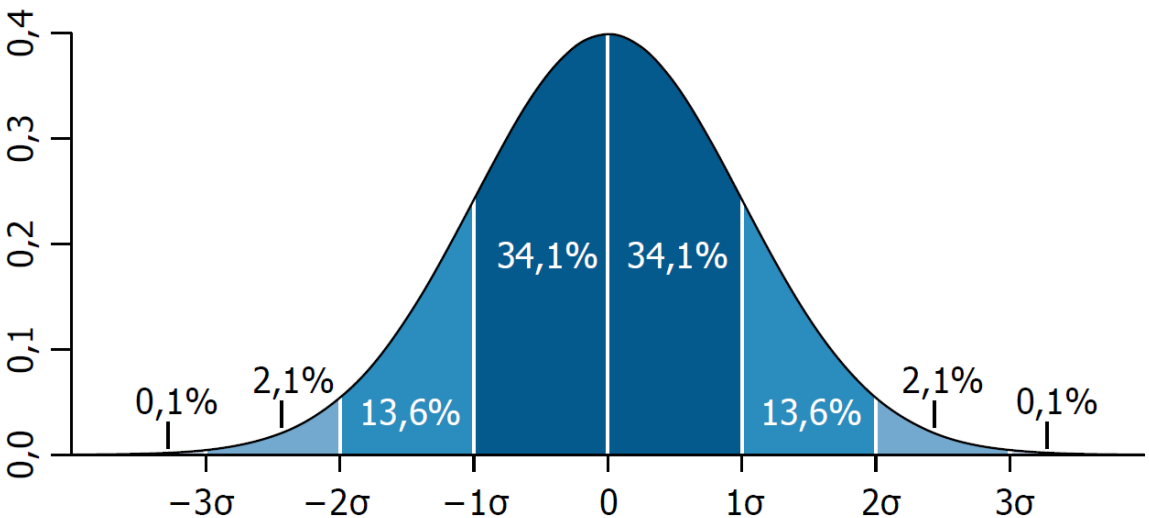
- Expected value: $\mu = E(x) = \sum_{i=1}^{n} x_i p_i$
- Standard deviation: $\sigma = \sqrt{var(X)} = \sqrt{E[(x-\mu)^2]} = \sqrt{E(x^2) - E^2(x)} = \sqrt{\sum_{i=1}^{n}(x_i^2 p_i) - \mu^2}$

In practice, we often cannot determine values μ and σ.

Gaussian distribution

The probability density function (PDF) of Gaussian distribution is given by:

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(x-\mu)^2}$$



We are looking for small representation of data sufficient to reliable reconstruct of properties of whole population.

Discrete variable X

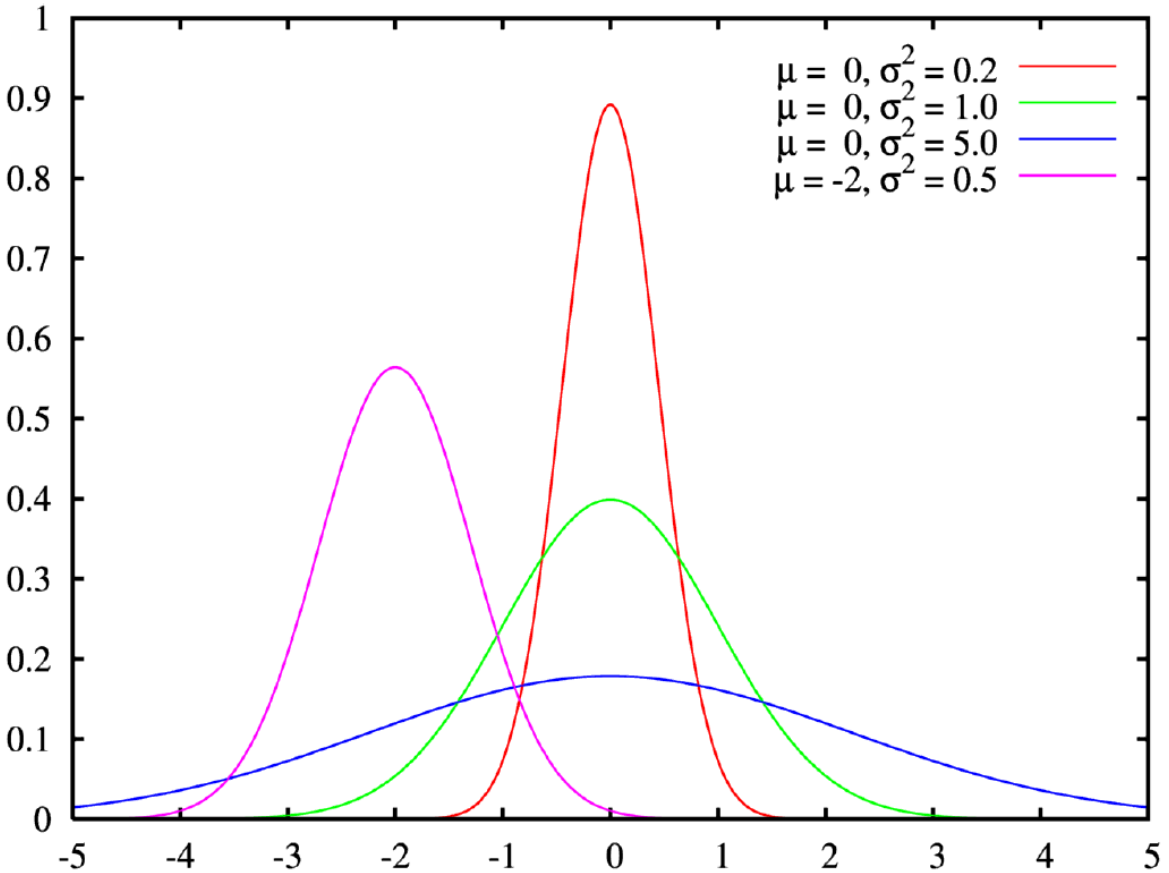
Expected value:

- Coin toss:
  $n = 2,\ w_1 = \text{head},\ w_2 = \text{tail},\ x_1 = 0;\ x_2 = 1$
  $p_1 = p_2 = \frac{1}{2}$
- Dice roll: $n = 6,\ w_i$ – throw of i dots, $x_i = i,\ p_i = \frac{1}{6}$

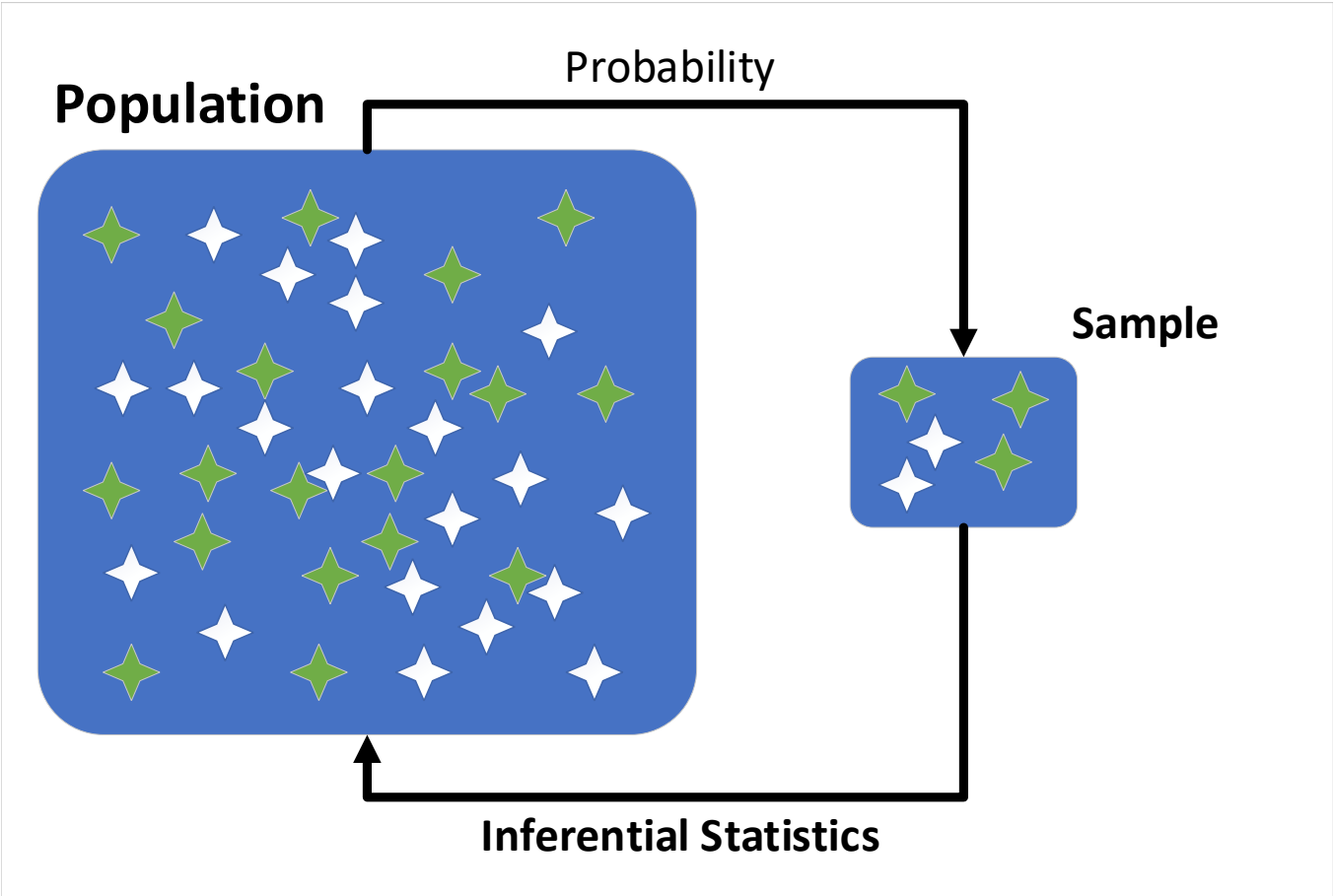
Distribution:

- Coin toss:
  - for $E = \{0, 1\} \rightarrow \mu = \frac{1}{2}, \sigma = \frac{1}{2}$
  - for $E = \{-1, 1\} \rightarrow \mu = 0, \sigma = 1$
- Dice roll: $\mu = 3\frac{1}{2}, \sigma = \sqrt{\frac{105}{36}} \approx 1.7$

Standard deviation



Estimator



Estimator

- Estimator: Statistic whose calculated value is used to estimate a population parameter, θ
- Estimate: A particular realization of an estimator, $\hat{\theta}$

Characteristics of estimator:

- For a given sample x, the "error" of the estimator:

$$e(x) = \hat{\theta}(x) - \theta$$

  As the sample size increases $\hat{\theta}$ gets closer to θ

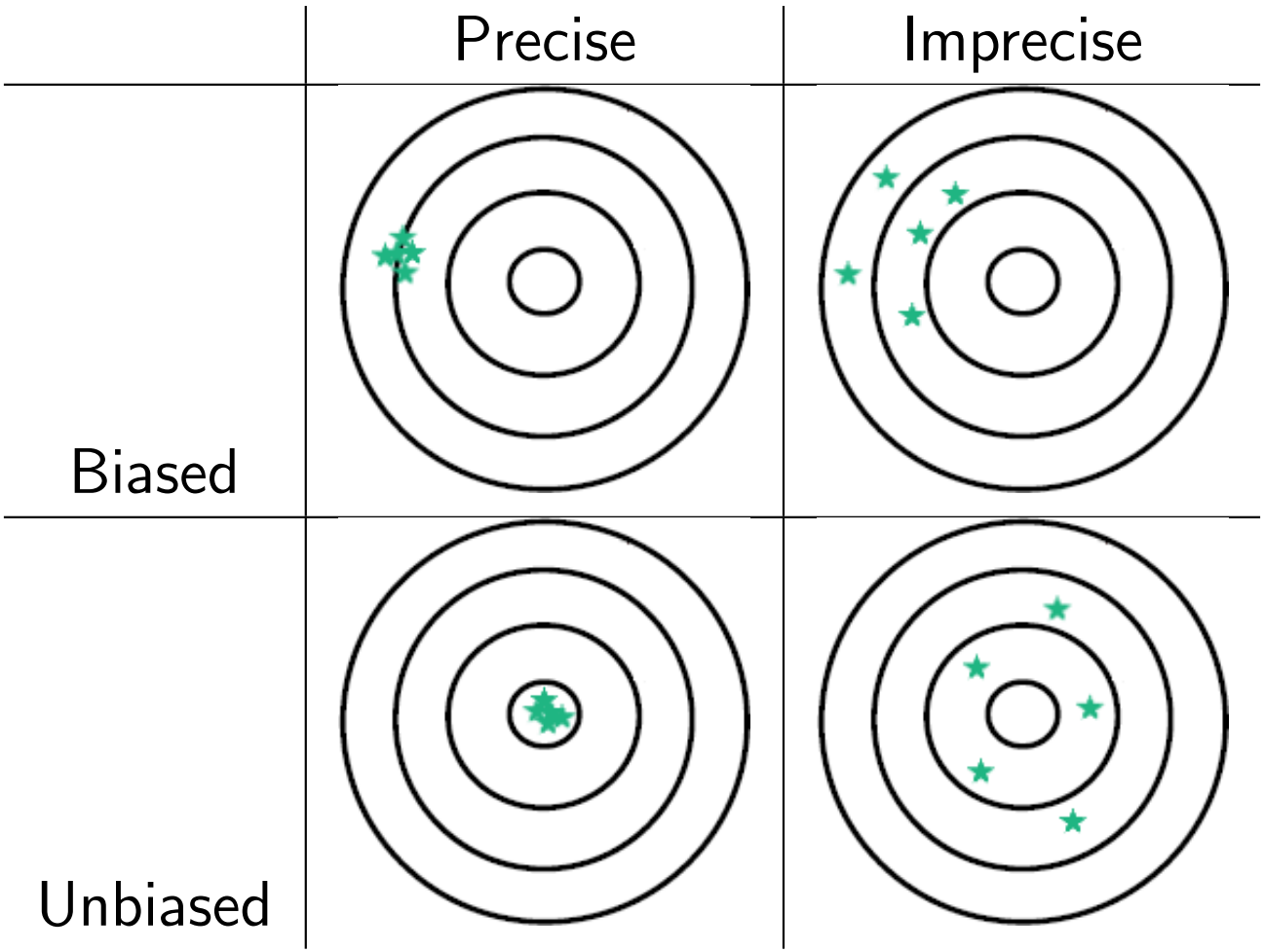
$$lim_{n \to \infty} P(|e(x)| > \epsilon) = 0$$

- The bias of $\hat{\theta}$ is defined as:

$$B(\hat{\theta}) = E(\hat{\theta}) - \theta$$

- Sampling distribution should have a small standard error.

Estimator



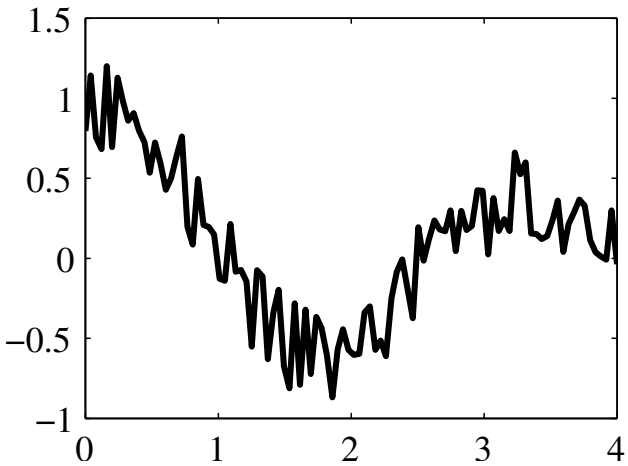
Estimator

There is no single estimator for the standard deviation with all these properties:

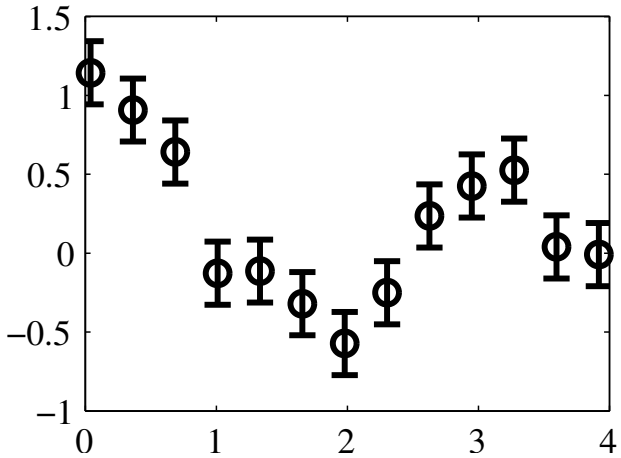
- Estimation of the expected value:
  $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$
- Estimation of the standard deviation:
  - Biased estimator (maximum likelihood estimate):
    $s = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2} = \sqrt{\bar{x^2} - \bar{x}^2}$
  - Unbiased estimator for the variance:
    $s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2} = \sqrt{\frac{n}{n-1}(\bar{x^2} - \bar{x}^2)}$

Conclusion

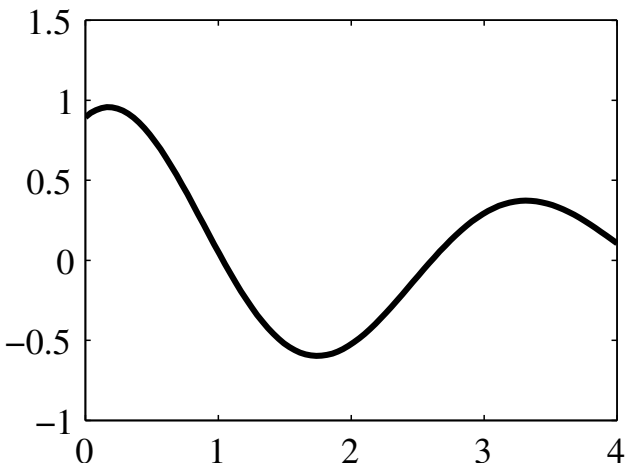
1. Reality



2. Measurement



3. Regression



4. Summary

- Estimation of the expectancy.
- Network generalizes "knowledge".
- How to evaluate this with many inputs?

Classification - Confusion matrix

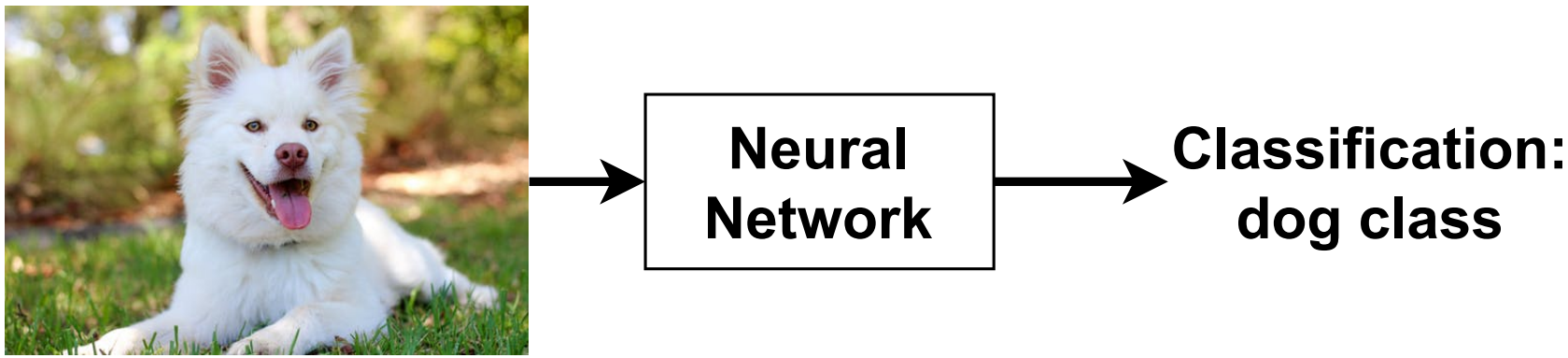
A confusion matrix (table of confusion) allows to assess the quality of results of classification problem:

- It gives you insight not only into the errors being made by your classifier but more importantly the types of errors that are being made,
- matrix is usually defined for binary classification (e.g. Yes or No, Positive or Negative), but is mainly used to evaluate the results of multi-classes,
- Table layout that allows clear visualization of the classification results,

Confusion matrix

Possible classification results of binary classification:

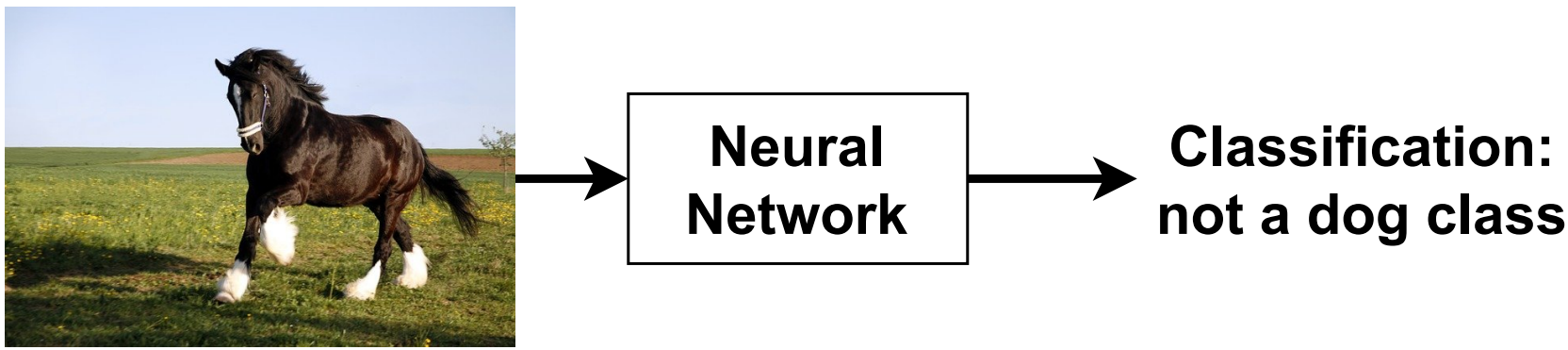
- True Positive (**TP**) - Observation is positive, and is predicted to be positive, e.g. the image shows a dog and the image is classified to dog class,



Confusion matrix

Possible classification results of binary classification:

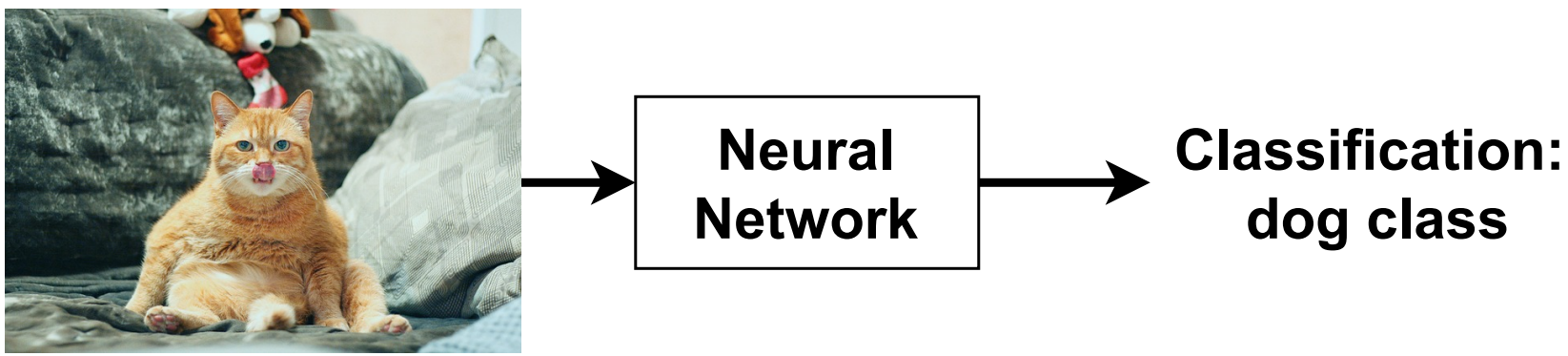
- True Negative (**TN**) - Observation is negative, and is predicted to be negative, e.g. the image shows a horse and the image is classified as not belonging to the dog class.



Confusion matrix

Possible classification results of binary classification:

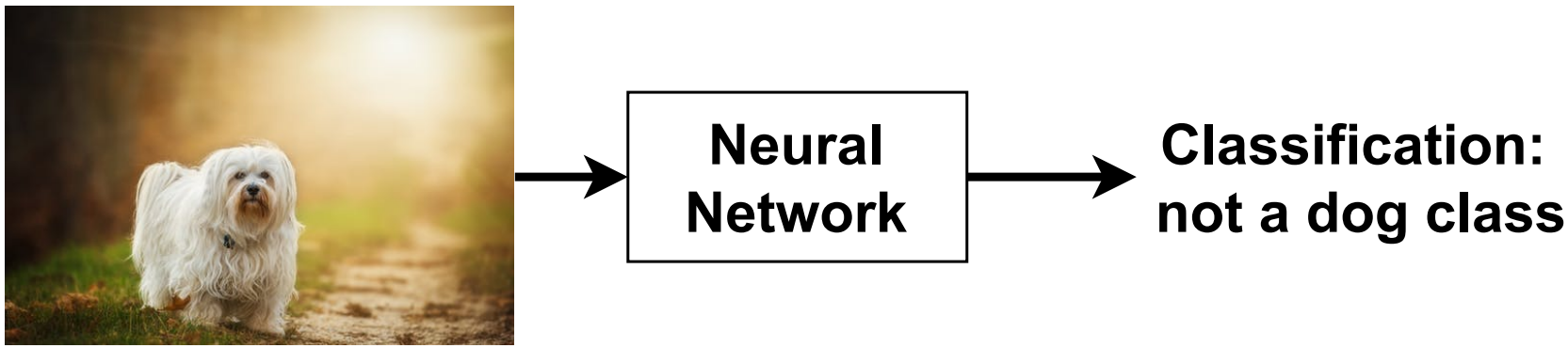
- False Positive (**FP**) - Observation is negative, but is predicted positive (type 1 error), e.g. the image shows a cat and the image is classified into a dog class,



Confusion matrix

Possible classification results of binary classification:

- False Negative (**FN**) - Observation is positive, but is predicted negative (type 2 error), e.g. the image shows a dog and the image is classified as not belonging to the dog class.



Confusion matrix

Confusion matrix in binary classification

		Predicted class	
		Positive	Negative
Actual class	Positive	TP True Positive	FN False Negative
	Negative	FP False Positive	TN True Negative

Confusion matrix

Selected classification measures:

- Accuracy:

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}$$

TP	FN
FP	TN

- Misclassification Rate (Error Rate):

$$ERR = \frac{FP + FN}{TP + TN + FP + FN}$$

TP	FN
FP	TN

The values range of classification measures is $< 0, 1 >$.

Confusion matrix

Selected classification measures:

- Recall (Sensitivity, True Positive Rate):

$$TPR = \frac{TP}{TP + FN}$$

TP	FN
FP	TN

- Precision:

$$PPV = \frac{TP}{TP + FP}$$

TP	FN
FP	TN

Confusion matrix

Selected classification measures:

- Specificity (Selectivity, True negative rate):

$$TNR = \frac{TN}{TN + FP}$$

TP	FN
FP	TN

- Negative predictive value:

$$NPV = \frac{TN}{TN + FN}$$

TP	FN
FP	TN

Confusion matrix

Selected classification measures:

- F1 (F-score, F-measure):

$$F_1 = 2\frac{Recall * Precision}{Recall + Precision} = \frac{2TP}{2TP + FN + FP}$$

F-score helps to measure both Recall and Precision at the same time. It uses harmonic mean in place of arithmetic mean by punishing the extreme values more.

Confusion matrix

Example: The results of the classification of 64 images into 3 classes: Cats, Dogs and Horses.

- 20 images of Dogs were classified into classes: 12 Dog, 6 Cat and 2 Horse.
- 26 images of Cats were classified into classes: 4 Dog, 22 Cat and 0 Horse.
- 18 images of Horses were classified into classes: 2 Dog, 1 Cat and 15 Horse.

Evaluate obtained results with use of confusion matrix.

Confusion matrix

Confusion matrix for 3 classes

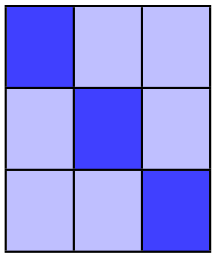
		Predicted class		
		Dog	Cat	Horse
Actual class	Dog	12	6	2
	Cat	4	22	0
	Horse	2	1	15

Confusion matrix

Total classification measures:

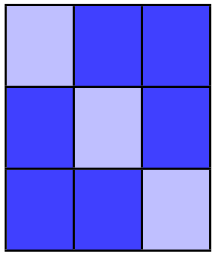
- Accuracy:

$$ACC = \frac{12 + 22 + 15}{64} = 76,6\%$$



- Misclassification Rate:

$$ERR = \frac{6 + 2 + 2 + 2 + 1}{64} = 23,4\%$$



Confusion matrix

Confusion matrix for 3 classes

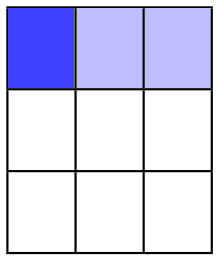
		Predicted class		
		Dog	Cat	Horse
Actual class	Dog	12	6	2
	Cat	4	22	0
	Horse	2	1	15

Confusion matrix

Classification measures for dog class:

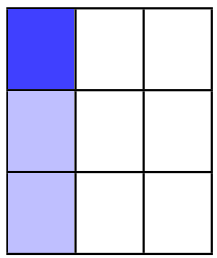
- Recall:

$$TPR = \frac{12}{12 + 6 + 2} = 60\%$$



- Precision:

$$PPV = \frac{12}{12 + 4 + 2} = 66,7\%$$



- $F_1 = 2 * \frac{0.6 * 0.667}{0.6 + 0.667} = 0,6352$

Introduction to Network Training

Network training

Training of neural network is a procedure for determining appropriate connection weights between neurons:

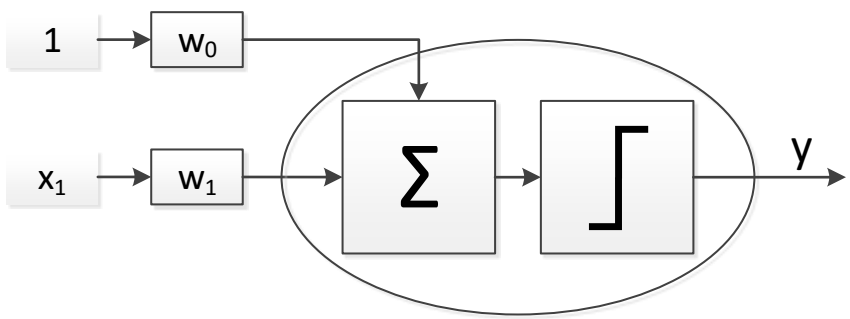
- N measurements y_j $j = 1, ..., N$ for inputs x_j.
- Minimize "distance" between data and model with use of assumed loss functions.
- Achieving "zero" for an loss function is generally not desirable (if possible?).

Perceptron Learning Rule

The classification Perceptron network use decision boundaries in shape of hyperplanes. The learning in this case is a process of shifting around the hyperplanes until each training pattern is classified correctly, but this shifting can be split up into a number of small steps (iterations).

$$g(v) = f(\sum_{i=1}^{n} w_i x_i)$$

$$f(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0 \end{cases}$$



Perceptron Learning Rule

Our goal is to find the w vector that can perfectly classify positive inputs and negative inputs in our data. Therefore we can assume following loss function :

$$|y - g|$$

where: y is target output and g is actual output. If the network weights at time k are $w_i(k)$, then the shifting process of moving weights by $\Delta w_i(k)$ for one sample:

$$w_i(k+1) = w_i(k) + \Delta w_i(t)$$

We initialize w with some random values.

Perceptron Learning Rule

We can determine learning algorithm by analysis Perceptron results

If	Effect	w	$y - g$
$y = g$	do nothing	$w_i = w_i$	0
$g = 1$,	$\sum w_i x_i$ too large	w_i depending on x_i	-1
$y = 0$	if $x_i = 1$	$w_i^{k+1} = w_i^k - \mu x_i^k$	
	if $x_i = 0$	$w_i^{k+1} =$ doesn't matter	
	change $w \rightarrow$	$w_i^{k+1} = w_i^k - \mu x_i^k$	
$g = 0$	$\sum w_{ij} x_i$ too small	$w_{i,j}$ depending on x_i	1
$y = 1$	if $x_i = 1$	$w_i^{k+1} = w_i^k + \mu x_i^k$	
	if $x_i = 0$	$w_i^{k+1} =$ doesn't matter	
	change $w \rightarrow$	$w_i^{k+1} = w_i^k + \mu x_i^k$	

Perceptron Learning Rule

The weight update can be written in the form:

$$w_i^{k+1} = w_i^k + (y - g)\mu x_i^k$$

- This weight update equation is called the Perceptron Learning Rule. The positive parameter μ is called the learning rate or step size. It determines how smoothly we shift the decision boundaries.
- If a problem is linearly separable than in a finite number of iterations we will find correct solution.

Network training

Training parameters:

- Weight changes need to be applied repeatedly – for each weight w in the network, and for each training pattern in the training set,
- Epoch - one pass through all the weights for the whole training set,
- Stop - all the network outputs match the targets for all the training patterns $\rightarrow$ w or the results of the neural network are not improving.

Network training

Two training algorithms

- On-line training.
  - Adaptive.
  - Model updated with successive examples.
  - Not all examples have to be available.
- Off-line traning.
  - All elements of the training set are used at once.
  - So all them have to be available.

Typical scheme

- Off-line training on available data.
- Model update by on-line learning.

Learning for Linear Models

Example: Linear regression function $x^T w$

- $g(x, w) = x^T w$ – hypothesis (for specific w).
- y – vector of output (measurement).
- x – vector of inputs (measurement).

Objective function:

$$L(w) = \min \sum_{j=1}^{n} [y(x_j) - g(x_j, w)]^2$$

Learning for Linear Models

Observation matrix:

$$\Psi = \begin{bmatrix} x^1 \\ x^2 \\ \vdots \\ x^N \end{bmatrix}_{N \times q} \Rightarrow g = \Psi w$$

Objective function:

$$J(w) = \min \sum_{k=1}^{n} [y(x_j) - g(x_j, w)]^2 =$$
$$||y - \Psi w||^2 = (y - \Psi w)^T (y - \Psi w)$$

Learning for Linear Models

Linear least squares (LLS):

- Objective function: $(y - \Psi w)^T (y - \Psi w)$
- Gradient of the objective function:

$$\frac{\partial J}{\partial w} = -2(y - \Psi w)^T \Psi = 0$$
$$\Psi^T \Psi w = \Psi^T y$$

- Parameter estimation

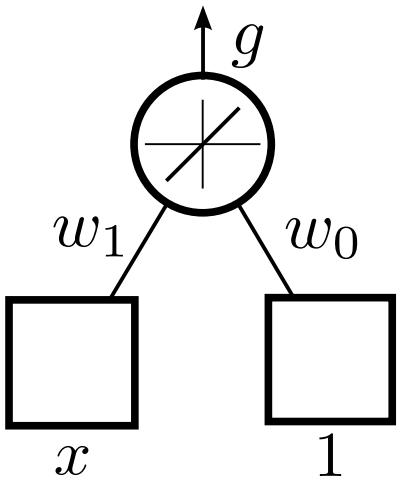
$$w = (\Psi^T \Psi)^{-1} \Psi^T y$$

- Residues

$$r = y - \Psi w \Rightarrow \epsilon = r^T r$$

Example

Linear network



Input data:

$$x^1 = [1, 2] \\ x^2 = [1, 3] \\ x^3 = [1, 5] \quad , y = \begin{bmatrix} 3 \\ 4 \\ 6 \end{bmatrix}$$

Weights (estimation):

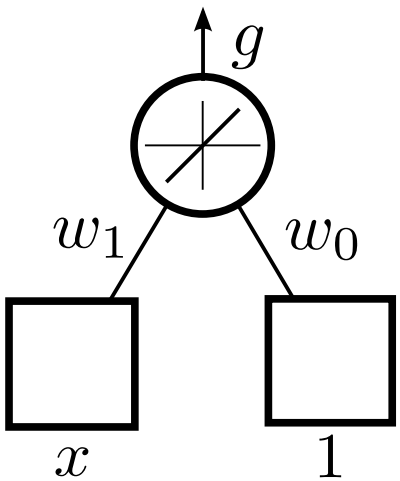
$$w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} =$$

$$= \left(\begin{bmatrix} 1 & 1 & 1 \\ 2 & 3 & 5 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 1 & 3 \\ 1 & 5 \end{bmatrix} \right)^{-1} \cdot$$

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 3 & 5 \end{bmatrix} \begin{bmatrix} 3 \\ 4 \\ 6 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Example

Linear network (noise) - small change in input data



Weights (estimation):

$$w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} 0.9312 \\ 1.0204 \end{bmatrix}$$

For x = 10, g = 11.1355

Input data:

$$x^1 = [1, 2.1] \quad x^2 = [1, 3] \quad x^3 = [1, 5] \quad , y = \begin{bmatrix} 3 \\ 4.1 \\ 6 \end{bmatrix}$$

$$g = \begin{bmatrix} 3.07 \\ 3.99 \\ 6.03 \end{bmatrix} , \epsilon = 0.02$$

The least-mean-square algorithm

Partial objective function

$$L(w) = \min \sum_{j=1}^{n} [y_j - g(x_j, w)]^2$$

Least Mean Square iterative change of weights:

$$w_{k+1} = w_k + \mu[g_{k+1} - (x_{k+1})^T w_k]x_{k+1}$$

μ - assumed constant learning rate.

Example

Initial data:

$$x_1 = [1, 2] \\ x_2 = [1, 3] \\ x_3 = [1, 5] \quad , y = \begin{bmatrix} 3 \\ 4 \\ 6 \end{bmatrix} , w_0 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} , \mu = 0.1$$

Iteration 1

$$w_1 = w_0 + \mu^k [g_1 - (x_1)^T w_0] x_1 =$$

$$\begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} + 0.1 \left(3 - [1, 2] \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \right) \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} + 0.1(3 - 1.5) \begin{bmatrix} 1 \\ 2 \end{bmatrix} =$$

$$\begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} + 0.15 \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 0.65 \\ 0.8 \end{bmatrix}$$

Example

Iteration 2

$$w_2 = w_1 + \mu[g_2 - (x_2)^T w_1]x_2 =$$

$$\begin{bmatrix}0.65\\0.8\end{bmatrix} + 0.1\left(4 - [1,3]\begin{bmatrix}0.65\\0.8\end{bmatrix}\right)\begin{bmatrix}1\\3\end{bmatrix} =$$

$$\begin{bmatrix}0.65\\0.8\end{bmatrix} + 0.095\begin{bmatrix}1\\3\end{bmatrix} = \begin{bmatrix}0.745\\1.085\end{bmatrix}$$

Iteration 3

$$w_3 = w_2 + \mu[g_3 - (x_3)^T w_2]x_3 = \begin{bmatrix}0.728\\1.000\end{bmatrix}$$

- Where to get the data for further iteration?
- How many iterations?

Example

Results after many epochs:

$$w_5 = \begin{bmatrix} 0.763 \\ 1.079 \end{bmatrix}$$

$$w_{10} = \begin{bmatrix} 0.789 \\ 1.047 \end{bmatrix}$$

$$w_{300} = \begin{bmatrix} 0.9998 \\ 1.0000 \end{bmatrix}$$

For 300 iteration error is $1.67e^{-4}$.

Questions