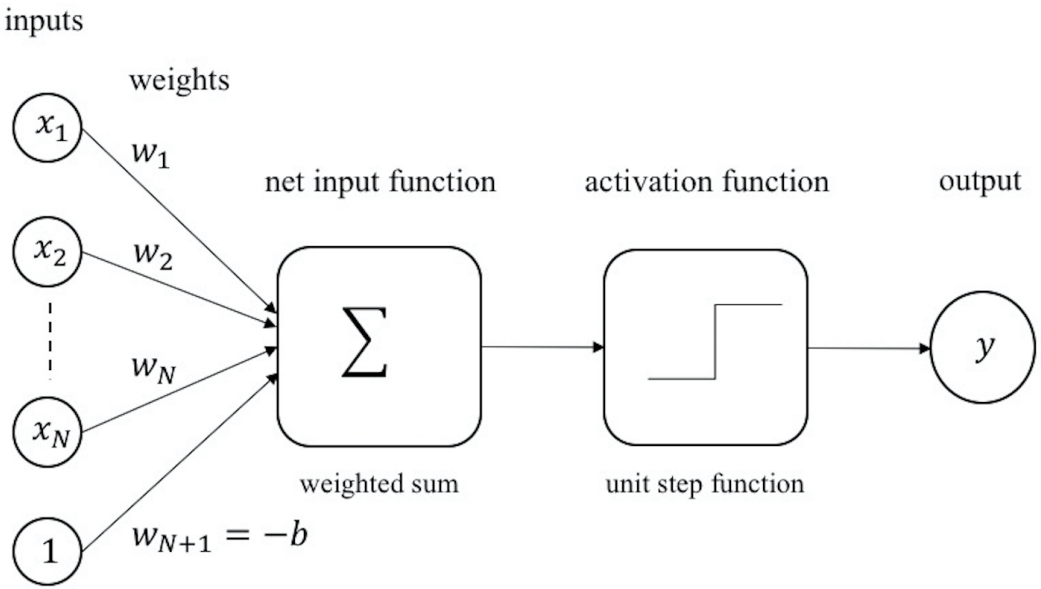
Introduction to Artificial Neural Network Theory and Applications

10. March, ACLS ZHAW Life Sciences

Perceptrons as Binary Classifiers



Given some input $\mathbf{x}$ of length N the perceptron outputs

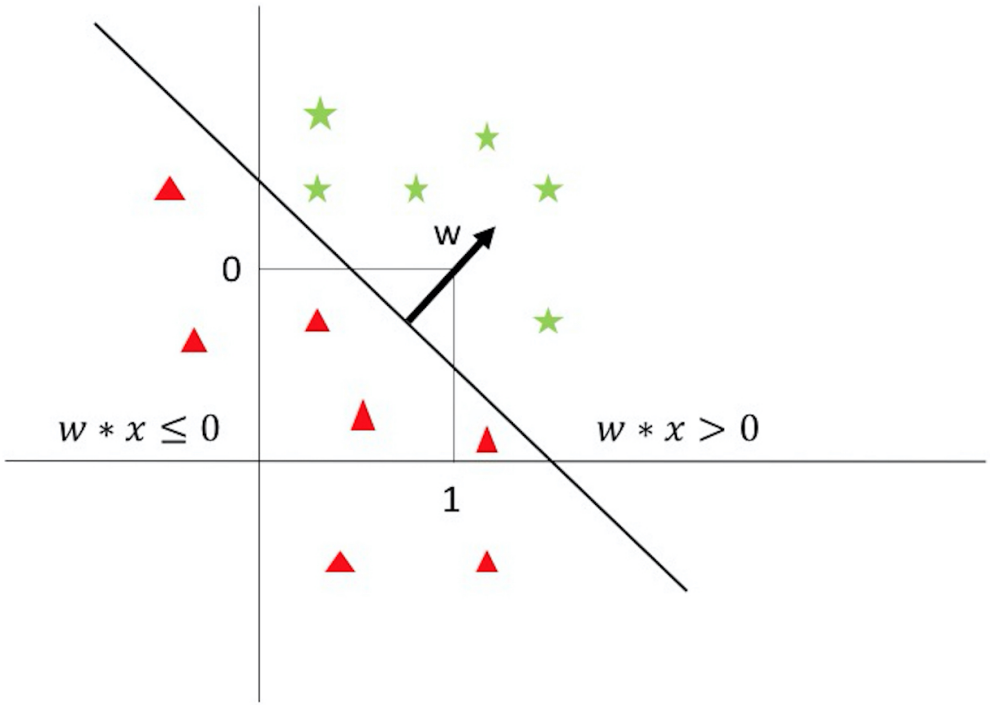
$$y = \sigma_{\mathbf{w}}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

where $\mathbf{w}$ is a vector of real-valued *weights* and b is the *threshold* or *bias*.
Usually, the bias b is converted: the input vector is extended to a $(N+1)$-dimensional vector $x_1,...,x_N,1$ and the weight vector to $w_1,...,w_N,w_{N+1}$ with $w_{N+1} = -b$.

Perceptrons as Binary Classifiers

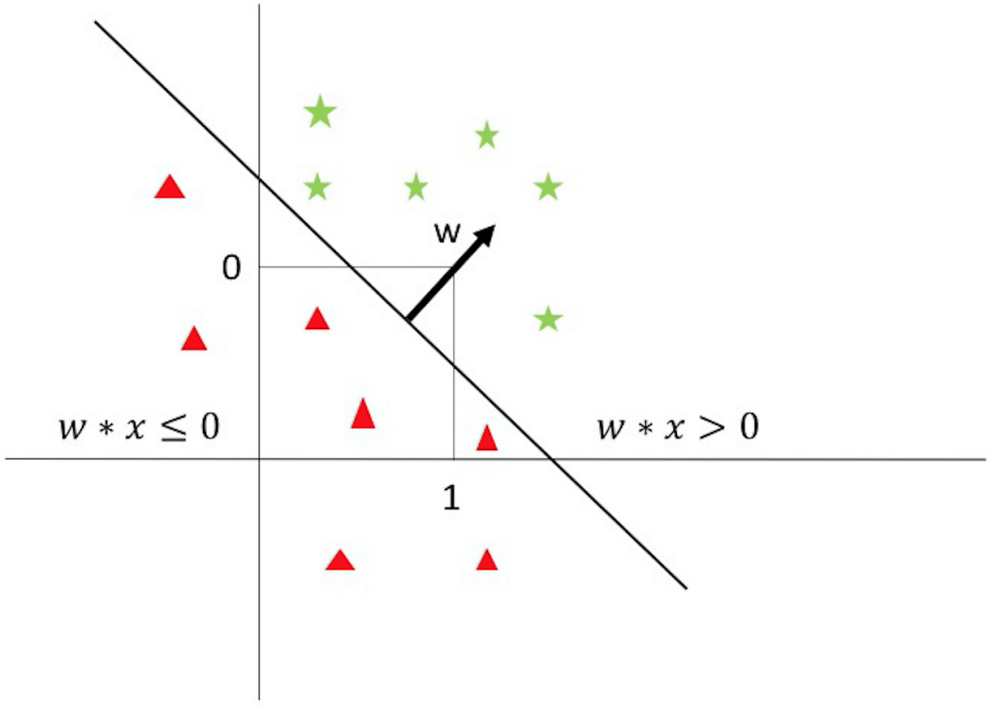
- Perceptron: historic artificial neural network (ANN) model invented in the late 1950s.
- In this course: basic example of ANN that allows to explain some of the main concepts in building ANN.
- Modified perceptrons: still of use in some applications, especially in so-called online learning systems.
- *Binary classifier*: it maps an input $\mathbf{x}$ (a real-valued vector) to a binary output value (i.e. to 0 or 1).
- Binary classification problems are ubiquitous.

Perceptrons as Binary Classifiers



The perceptron separates the data by a *decision boundary*: data points that satisfy $w \cdot x + b > 0$ lie on one side of the line, the other data points on the other side. The weight vector w determines the orientation of the line, the bias b shifts the line away from the origin, i.e., fixes the position of the line.

Perceptrons as Binary Classifiers



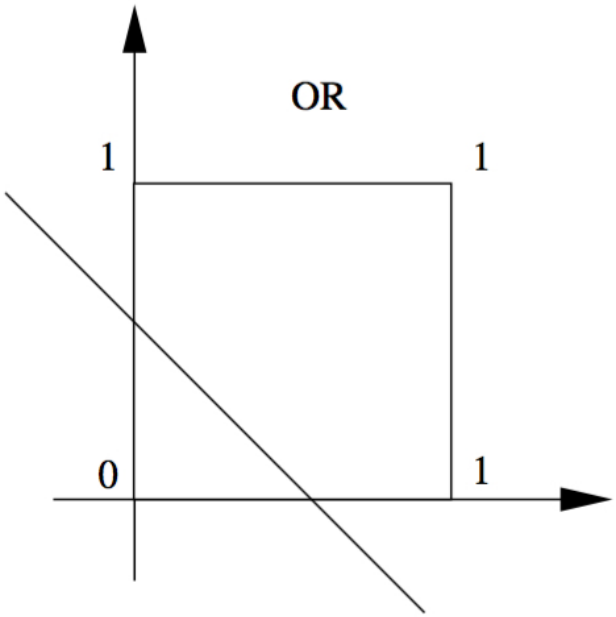
Task Convince yourself mathematically that in the case $N = 2$ the expression (**??**) effectively determines a straight line in the input space.

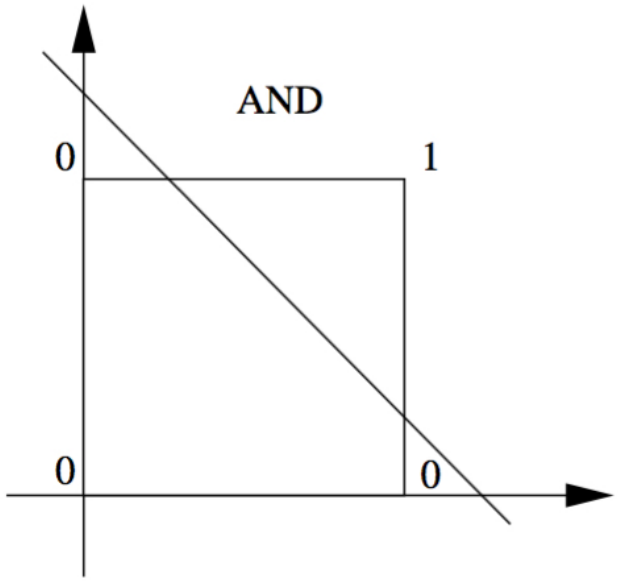
Perceptrons as Binary Classifiers

- Perceptron can only classify the data correctly if it is possible to separate the data points by a straight line. The problems for which this is possible are called *linearly separable*.
- Example: Boolean functions, i.e., the logical operators $f : \{0, 1\} \to \{0, 1\}$. Some Boolean operators can be computed by a perceptron, some cannot.

Perceptrons as Binary Classifiers

- Perceptron can only classify the data correctly if it is possible to separate the data points by a straight line. The problems for which this is possible are called *linearly separable*.
- Example: Boolean functions, i.e., the logical operators $f : \{0, 1\} \to \{0, 1\}$. Some Boolean operators can be computed by a perceptron, some cannot.





Perceptrons as Binary Classifiers

Task Compute by hand a perceptron that implements the logical operator AND or OR.

Perceptrons as Binary Classifiers

Task Display, both visually and mathematically, a Boolean operator that cannot be implemented by a perceptron.

Perceptrons as Binary Classifiers

Task Compute in Python a perceptron that implements the logical operator AND or OR.

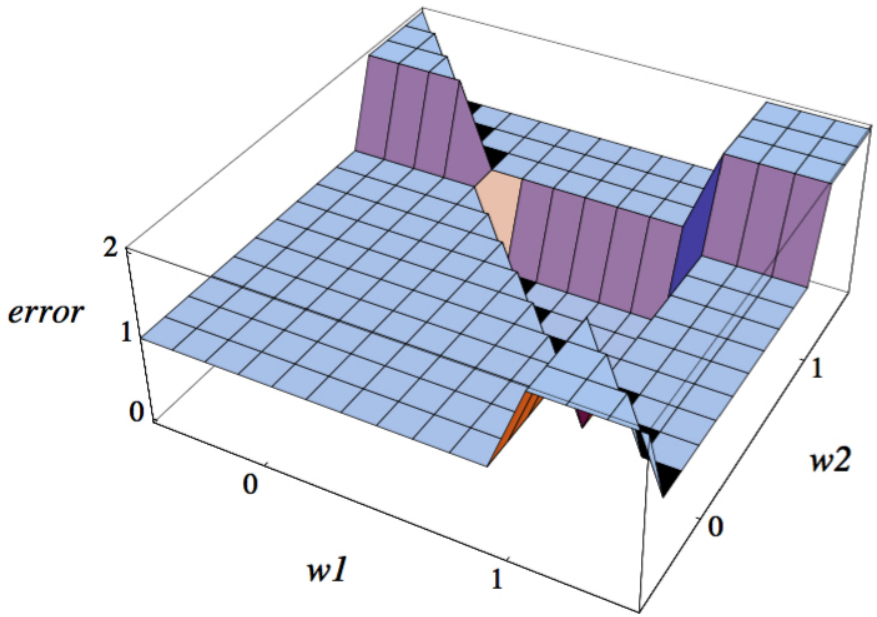
Perceptron Learning Rule

- Learning problem: classify the elements of a set into two classes (*binary classification problem*) by a perceptron.
- Find the weights and bias that separate set A from set B of input vectors.
- Mathematically: Find $\sigma_{\mathbf{w}}(\mathbf{x})$ such that $\sigma_{\mathbf{w}}(x) = 1$ for $x \in A$ and $\sigma_{\mathbf{w}}(\mathbf{x}) = 0$ for $\mathbf{x} \in B$.
- Error function: number of false classifications obtained using the weight vector $\mathbf{w}$

$$E(w) = \sum_{\mathbf{x} \in A} (1 - \sigma_{\mathbf{w}}(\mathbf{x})) + \sum_{\mathbf{x} \in B} \sigma_{\mathbf{w}}(\mathbf{x}). \tag{2}$$

- Error function must be minimised.

Perceptron Learning Rule



Example: error function for a perceptron with constant bias $b = 1$ that computes the binary function AND, with the error function plotted for all combinations of the two weights between -0.5 and 1.5. The solution region is the triangular area in the middle.

Perceptron Learning Rule

Find vector $\mathbf{w}$ capable of separating sets A and B, such that all vectors in A belong to the positive half-space ($\mathbf{w} \cdot \mathbf{x} > 0$) and all vectors in B to the negative half-space ($\mathbf{w} \cdot \mathbf{x} < 0$) of the linear separation.

1. An initial weight vector $\mathbf{w}_0$ is selected at time $t = 0$.
2. A vector $x \in A \cup B$ is selected.
3. If $\mathbf{x} \in A$ and $\mathbf{w} \cdot \mathbf{x} > 0$, select another vector $\mathbf{x}$.
   If $\mathbf{x} \in A$ and $\mathbf{w} \cdot \mathbf{x} \leq 0$, set $\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{x}$, $t = t + 1$ and select another vector $\mathbf{x}$.
   If $\mathbf{x} \in B$ and $\mathbf{w} \cdot \mathbf{x} < 0$, select another vector $\mathbf{x}$.
   If $\mathbf{x} \in B$ and $\mathbf{w} \cdot \mathbf{x} \geq 0$, set $\mathbf{w}_{t+1} = \mathbf{w}_t - \mathbf{x}$, $t = t + 1$ and select another vector $\mathbf{x}$.
4. If all vectors $\mathbf{x}$ are classified correctly stop. If not, define a stopping time ("epochs").

If the two sets A and B are linearly separable, the vector $\mathbf{w}$ is updated only a finite number of times, i.e., converges.

Perceptron Learning Rule

The perceptron learning rule in a more compact way: with t being the *target* the error is

$$e = t - y. \tag{3}$$

The weights in the extended version are updated as follows

$$\mathbf{w}^{t+1} = \mathbf{w}^{t} + e \cdot \mathbf{x}. \tag{4}$$

Perceptron Learning Rule

The perceptron learning rule in a more compact way: with t being the *target* the error is

$$e = t - y. \tag{3}$$

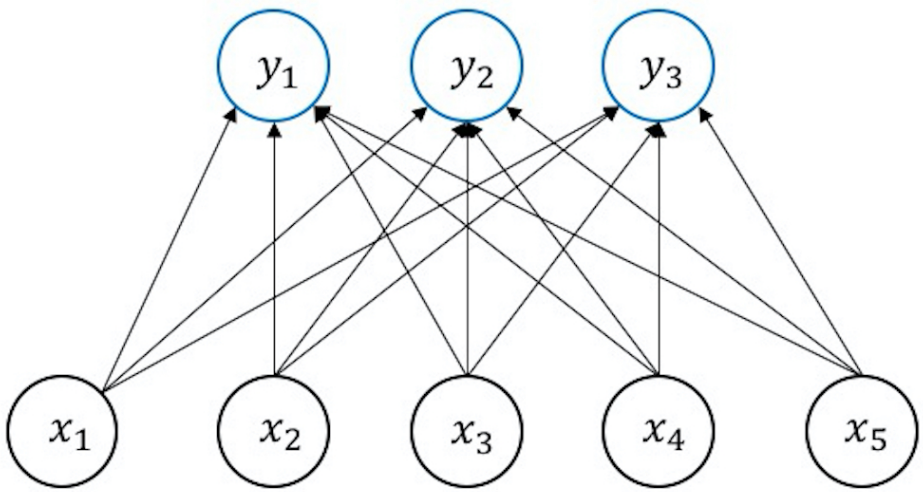
The weights in the extended version are updated as follows

$$\mathbf{w}^{t+1} = \mathbf{w}^{t} + e \cdot \mathbf{x}. \tag{4}$$

Task Convince yourself that (**??**) achieves the same as the scheme of the learning algorithm above.

Perceptron Learning Rule

Many extensions and generalisations to the perceptron exist.
A first extension is to allow for multiple outputs, leading to a *multi-perceptron* that outputs a vector $\mathbf{y}$.



Perceptron Learning Rule

The perceptron now involves a matrix $\mathbf{W}$ with elements w_{ij} and outputs (in the extended version)

$$\mathbf{y} = \sigma(\mathbf{W} \cdot \mathbf{x}) \tag{5}$$

with the activation function applied to each element of the vector $\mathbf{z} = \mathbf{W} \cdot \mathbf{x}$.

A multi-perceptron allows to assign to each input vector $\mathbf{x}$ a target vector $\mathbf{t}$, resulting in an error $\mathbf{e} = \mathbf{t} - \mathbf{y}$. Thereby multiclass classification problems can be solved. The learning rule generalises to

$$\mathbf{W}^{t+1} = \mathbf{W}^t + \mathbf{e}\mathbf{x}^T, \tag{6}$$

where $\mathbf{x}^T$ is the transpose of $\mathbf{x}$.

Perceptron Learning Rule

Task Implement the perceptron learning algorithm in Python. Let the algorithm learn the logical operators AND and OR. What happens in the case if the operator XOR should be learned?

Task Visualise the perceptron learning algorithm in Python for the OR, AND and XOR problem, i.e., draw the decision boundary at each learning step.

Task Implement the multi-perceptron in Python.

Activation Functions and Gradient Descent

Original perceptron:

$$y = \sigma(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

where $z = \mathbf{w} \cdot \mathbf{x} + b$ and the unit step function is used as the so-called *activation function*.

Modification: different activation function

$$y = \sigma(z) = z \tag{8}$$

Thus,

$$y = \mathbf{w} \cdot \mathbf{x} + b. \tag{9}$$

The Adaline, Widrow-Hoff or Delta Learning Rule

Assume that there are K examples (data points) of input vectors of size N and K target vectors of size M. We then want to minimise the cost function or performance measure

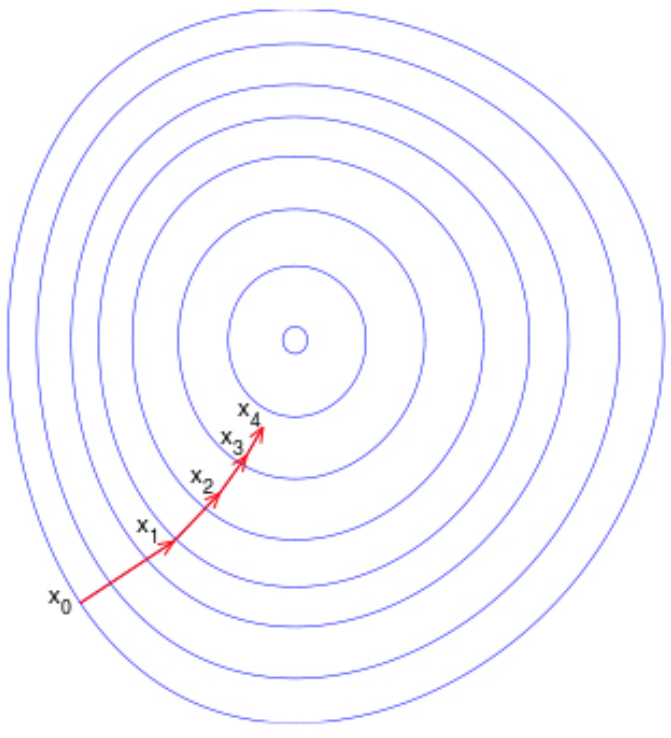
$$E(\mathbf{x}) = \frac{1}{2}\sum_{i,k}(t_{i,k} - y_{i,k})^2 = \frac{1}{2}\sum_{i,k}(t_{i,k} - \sum_{j} w_{ij}x_{j,k})^2. \tag{10}$$

In principle: solve for the weights $\mathbf{w}$ that minimise $E(\mathbf{w})$, i.e. solving $\nabla_{\mathbf{w}} E(\mathbf{w}) = 0$.

We want a learning rule: change the weights such that the error is reduced.

Change each weight w_{ij} by an amount Δw_{ij} proportional to the gradient of E, that is, $\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$.

The Adaline, Widrow-Hoff or Delta Learning Rule



The Adaline, Widrow-Hoff or Delta Learning Rule

Let us thus compute $\frac{\partial E}{\partial w_{ij}}$:

The Adaline, Widrow-Hoff or Delta Learning Rule

Let us thus compute $\frac{\partial E}{\partial w_{ij}}$:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \frac{1}{2} \sum_{i,k} (t_{i,k} - y_{i,k})^2 = \frac{1}{2} \sum_{k} \frac{\partial}{\partial w_{ij}} (t_{i,k} - y_{i,k})^2 \tag{11}$$

$$= \frac{1}{2} \sum_{k} 2(t_{i,k} - y_{i,k}) \frac{\partial}{\partial w_{ij}} (t_{i,k} - y_{i,k}) \tag{12}$$

$$= \sum_{k} (t_{i,k} - y_{i,k}) \frac{\partial}{\partial w_{ij}} (t_{i,k} - \sum_{j} w_{ij} x_{j,k}) \tag{13}$$

$$= \sum_{k} (t_{i,k} - y_{i,k})(-x_{j,k}). \tag{14}$$

The Adaline, Widrow-Hoff or Delta Learning Rule

- We have $\frac{\partial E}{\partial w_{ij}} = -\sum_k x_{j,k}(t_{i,k} - y_{i,k})$ and if set $\delta_{i,k} = t_{i,k} - y_{i,k}$ we have for the learning rule

$$w_{ij} = w_{ij} + \Delta w_{ij} = w_{ij} + \eta(\sum_k x_{j,k}\delta_{i,k}), \tag{15}$$

  where η is the so-called *learning rate*.
- Gradient descent: weights are incremented in the negative direction to the gradient, that is, $-(-\delta x_{j,k}) = +\delta x_{j,k}$. If $(t_{i,k} - y_{i,k})$ is classified as $0, 1, -1$, one gets the perceptron learning rule with $\pm x_{j,k}$.
- Although the learning rule thus looks identical to the perceptron rule, there are two main differences:
  - The output y is a real number and not a class label, i.e. 0 or 1, as in the perceptron learning rule.
  - The weight update is calculated based on all samples in the training set (sum over k in (**??**)), which is why this approach is also called *batch gradient descent*.

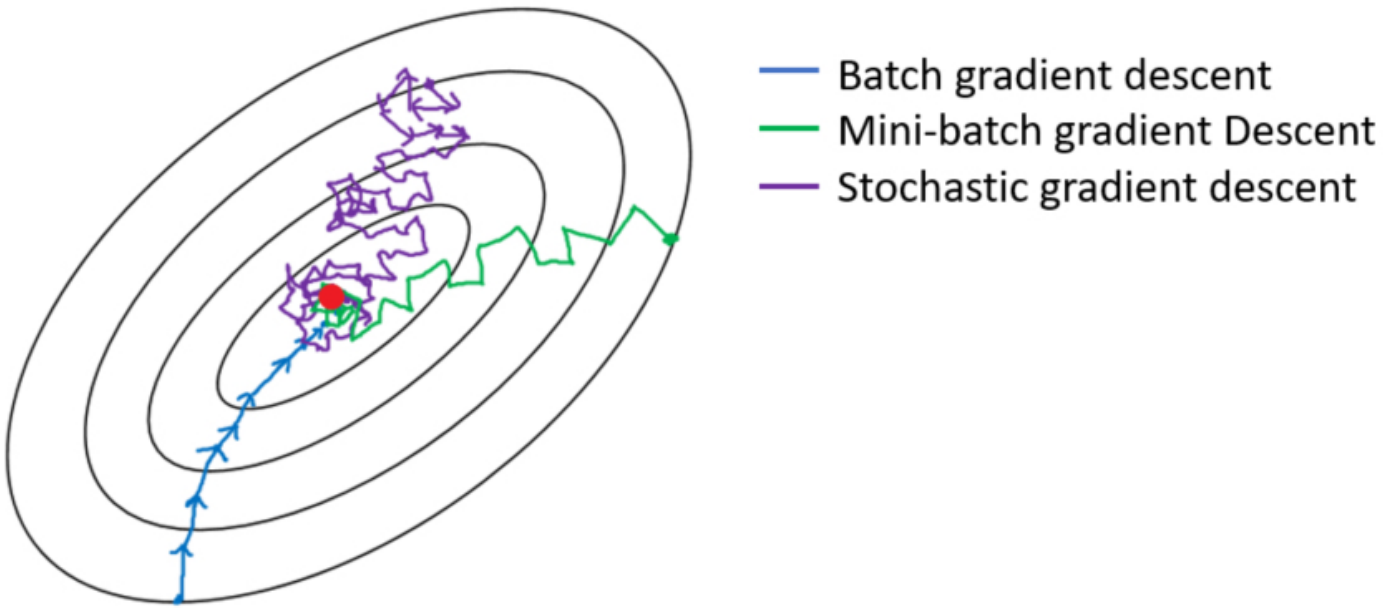
The Adaline, Widrow-Hoff or Delta Learning Rule

- Delta learning rule

$$w_{ij} = w_{ij} + \Delta w_{ij} = w_{ij} + \eta(\sum_{k} x_{j,k}\delta_{i,k}). \tag{16}$$

- Batch learning: cost function is minimised based on the complete training data set.
- Perceptron: *online learning*
- *Stochastic gradient descent* (sgd): approximates the minimisation of the cost function in steps.
- Advantages:
  - Often, sgd converges faster
  - computationally more efficient
  - online learning: classifier is immediately updated as new training data arrives, e.g., in web applications, and old training data can be discarded if storage is an issue.
- Large-scale machine learning systems: common practice to use *mini-batches*
- Standard sdg algorithm: sampling with replacement
  Sampling without replacement: each training sample is evaluated exactly once in every epoch

The Adaline, Widrow-Hoff or Delta Learning Rule



Activation Functions

- The delta learning rule can be derived for any linear differentiable output/activation function, whereas in the perceptron case one has the threshold output function.
- Computation of the gradient of the error function: we must guarantee the continuity and differentiability of the error function.
- Another popular activation function:
  Sigmoid function $\sigma : \mathbb{R} \to (0, 1)$ defined by

$$\sigma(z) = \frac{1}{1 + e^{-c \cdot z}}. \tag{17}$$

- Many other kinds of activation functions: any differentiable activation makes the error function also differentiable.

Tasks, Performance and Experience

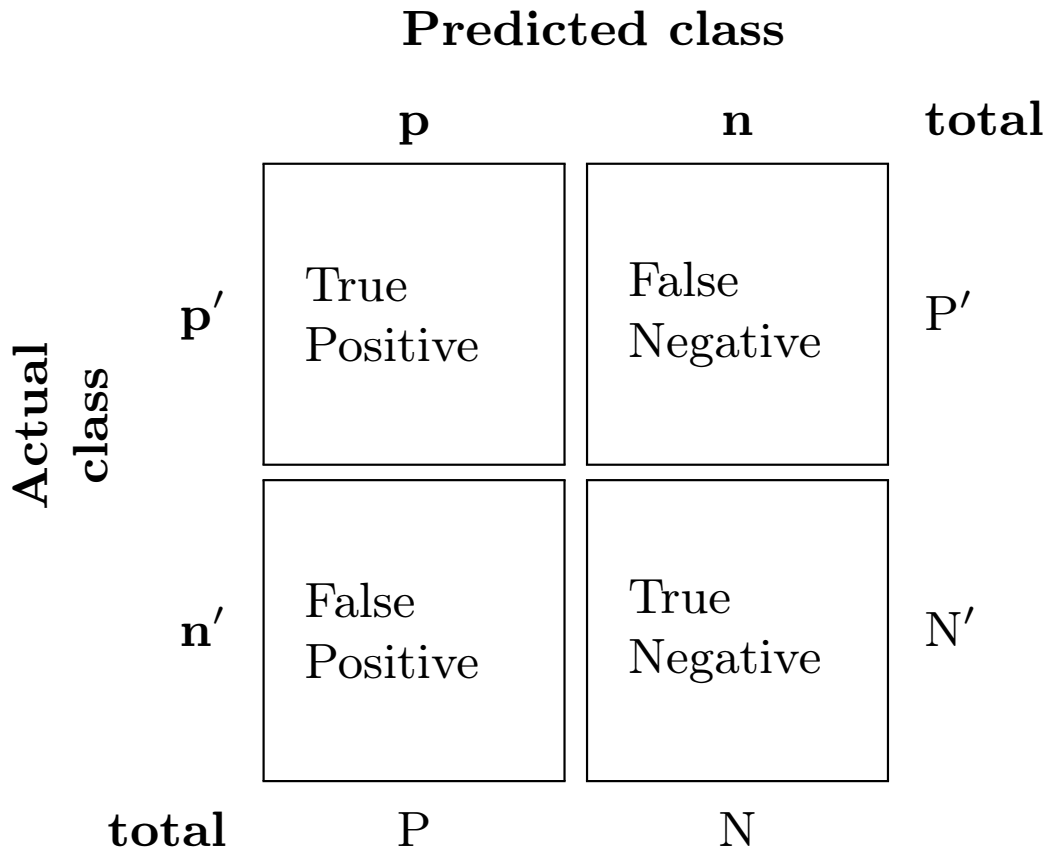
Read the section **Tasks, Performance and Experience** in the script.

Classification Tasks

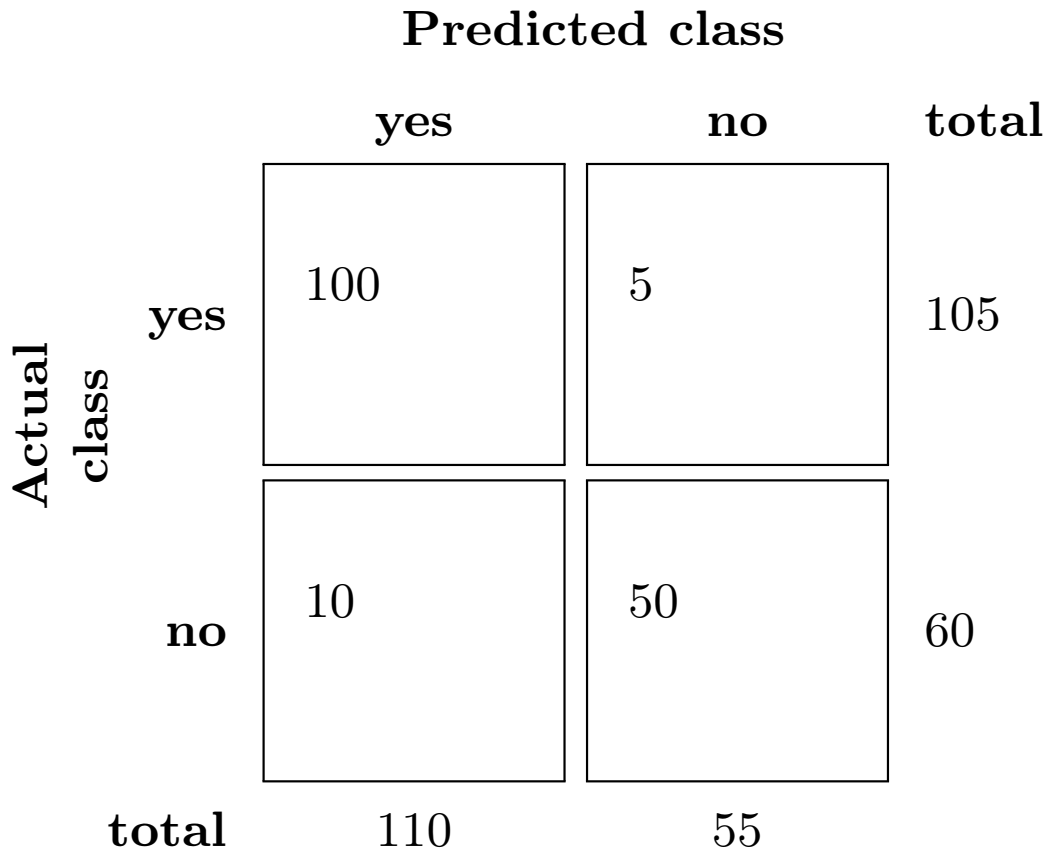
- Perceptron: binary classifier
- Classification tasks: binary classification, multiclass classification, multi-label classification
- How good is a classification? → *performance measures*

Performance Measures: Binary Classification

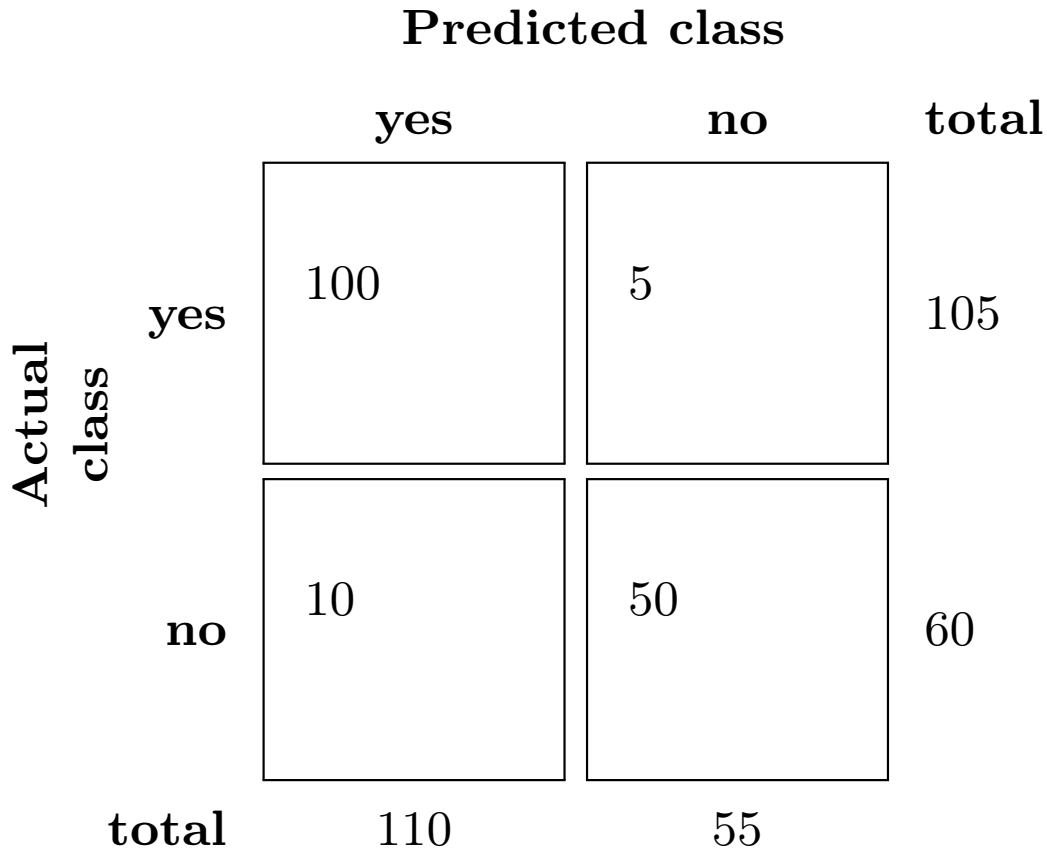
Contingency table



Performance Measures: Binary Classification



Performance Measures: Binary Classification



Example: a classifier predicted for 165 day whether is is dry the next day ("yes") or not ("no"). There was a total of 110 dry days predicted and 55 days with some precipitation. Actually, it was dry on 105 days and there was some rain on 60 days.

Performance Measures: Binary Classification

- **Accuracy**: the accuracy measures how often the classifier is correct, that is, $\frac{tp+tn}{N} = \frac{100+50}{165} = 0.91$.
- **Misclassification Rate**: the misclassification rate states how often the classifier was wrong, that is, $\frac{fp+fn}{N} = \frac{10+5}{165} = 0.09$.
- **Precision**: precision displays how often the classifier was correct in the cases it predicted a "yes", that is, $\frac{tp}{tp+fp} = \frac{100}{100+10} = 0.91$.
- **Recall** or **Sensitivity** or **True Positive Rate**: recall shows how often the classifier predicted "yes" when it actually was "yes", that is, $\frac{tp}{tp+fn} = \frac{100}{100+5} = 0.95$.
- **False Positive Rate**: the false positive rate shows how often the classifier predicted "yes" when it actually was "no", that is, $\frac{fp}{fp+tn} = \frac{10}{10+50} = 0.17$.
- **Specificity**: specificity tells you how often the classifier predicted "no" when it actually was "no", that is, $\frac{tn}{fp+tn} = \frac{50}{10+50} = 0.83$.
- **F-Score**: the F-score gives the harmonic mean of precision and recall, that is, $F_1 = 2 \cdot \frac{precision \cdot recall}{precision+recall}$.

Performance Measures: Multiclass Classification

Multiclass classification: classify and predict which elements of a given set belong to one of three or more classes.
Confusion matrix: For example, if in a classification problem one has three classes coded by $\{0, 1, 2\}$ and the true classes are $(2, 0, 2, 2, 0, 1)$ and the predicted classes are $(0, 0, 2, 2, 0, 2)$, the confusion matrix is

$$\mathbf{C} = \underbrace{\left.\begin{pmatrix} 2 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 2 \end{pmatrix}\right\}}_{\text{predicted}} \text{ actual}$$

Performance Measures: Multi-Label Classification

Multi-label classification: multiple classes or labels attached to an example.
For example, data on a patient in a hospital usually includes multiple labels such as blood pressure, temperature, etc.
Formal example: multi-label classification maps inputs $\mathbf{x}$ to binary vectors $\mathbf{y}$ (assigning a value of 0 or 1 for each element (label) in $\mathbf{y}$).

Performance Measures

Task Implement the binary classification performance measures and the multiclass classification performance measures in Python.

Task How could the multi-perceptron be used for a multiclass classification problem?

Task

Task Classify the iris data set with a modified multi-perceptron.
The iris dataset contains four measurements for 150 flowers representing three species of iris (Iris setosa, versicolor and virginica). The task is to predict the species given the measurement data.
Follow through these steps:

1. Load the iris dataset.
2. Visualize the iris dataset (scatterplots).
3. Split the dataset in training and test dataset.
4. Train a sigmoid multi-perceptron model.
5. Evaluate training performance.
6. Evaluate test performance.
7. How could the test performance be improved?
8. What are the problems with this model?