Artificial Intelligence Fundamentals Laboratory

Piotr Łuczak PhD Eng.
piotr.luczak.1@p.lodz.pl

Winter 2024

1 Introduction

The aim of the laboratory is to familiarize the students with the basics of artificial intelligence. Three main foci of the laboratory course are artificial neural networks, search algorithms and fuzzy logic.

2 Laboratory dates and task deadlines

The laboratory will take place in weeks 1 - 15 of the winter semester from 08:15 till 10:00 (group 1) or from 10:15 till 12:00 (group 2) in room E109, Institute of Applied Computer Science, building A12.
Deadlines:

1. Week 3 (2024-10-18) → Data generator and visualizer
2. Week 6 (2024-11-15) → Single neuron
3. Week 9 (2024-12-06) → Shallow neural network
4. Week 12 (2025-01-10) → Search algorithms
5. Week 15 (2025-01-31) → Fuzzy control

The final grade will be the average of the grades from each of the assignments; a delay in handing in the task solution will decrease the grade for this task by 0.5. In order to pass the course, **all** assignments must be handed in and awarded a passing grade. Grading requirements are cumulative: in order to score a higher grade, all requirements for lower ones must also be satisfied.
Handing in an assignment involves presenting a running version of the code, answering questions related to the presented code and answering questions related to the material presented during the laboratories and lectures. Failing to answer the questions will prevent you from submitting assignment solutions during that laboratory session. Another attempt at handing in can be made during the next laboratory.
The **final deadline** for handing in all assignments is **2025-02-03** (start of the exam session), assignment solutions will not be accepted after this date.

3 Implementation requirements

- Task solutions must be implemented in Python[1] version 3.10 or higher.
- Use of existing implementations (libraries, code samples available online, etc.) of solutions to given problems is **prohibited unless otherwise specified by this manual**. Libraries may be used for ancillary code (plotting, matrix calculations, GUI, etc.), for these tasks matplotlib[2], NumPy[3] and Streamlit[4]/Dash[5]/QT[6] are suggested.
- All libraries used along with the exact version numbers must be specified in `requirements.txt`[7] or `environment.yml`[8]
- All libraries must be installable through pip or conda.
- Solutions to tasks 1 - 3 should be implemented as a single program with graphical user interface.
- Solution to tasks 4 and 5 should be implemented using the provided scripts.
- All solutions must be delivered as a single zip file, named **{ID}_{name}_{surname}.zip** at the end of the course, **failure to do so results in failing the subject.**

4 Task descriptions

4.1 Data generator and visualizer

Prepare a GUI that enables the generation and visualization of data samples from two classes. Samples should be two-dimensional, so that they can be plotted in x-y space. Each class should consist of one or more Gaussian modes[9] with their means and variances chosen randomly from some given interval (e.g. $\mu_x, \mu_y \in [-1..1]$).
The interface should allow for setting the desired number of modes per class and a desired number of samples per mode, as well as visualization of the generated samples on a two-dimensional plot. Class labels, which are either 0 or 1, should be indicated by colors.
The interface should be created using a proper UI library (QT, Streamlit, Dash (plotly), tkinter). **Do not use** the GUI functionality built into matplotlib.

4.1.1 Grading

Generation of samples with a single mode per class + visualization in GUI $\rightarrow$ 3
Addition of a user-provided number of samples per mode $\rightarrow$ 4
Addition of multiple modes $\rightarrow$ 5

[1] `https://www.python.org/`
[2] `https://matplotlib.org/`
[3] `https://numpy.org`
[4] `https://streamlit.io/`
[5] `https://dash.plotly.com/`
[6] `https://www.qt.io/qt-for-python`
[7] `https://pip.pypa.io/en/latest/user_guide/`
[8] `https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html`
[9] `https://en.wikipedia.org/wiki/Normal_distribution`

4.2 Single neuron

Implement an artificial neuron. The neuron should take samples generated by a code from the task 1 as its input and predict their class membership at its output. The neuron should be trainable through the formula presented during the lecture:

$$\Delta\vec{w_j} = \eta\varepsilon f'(s)\vec{x_j} = \eta(d-y)f'(\vec{w}^T\vec{x_j})\vec{x_j} \tag{1}$$

where:

- x_j is the j^{th} sample
- $f'(s)$ is the derivative of the activation function evaluated for the j^{th} sample
- $\vec{w}$ are the weights associated with inputs
- η is the learning rate
- d is the expected (true) class label
- y is a class label predicted by the neuron

Implemented neuron should allow for different activation functions for evaluation:

- Heaviside step function (perceptron)

$$H(s) = \begin{cases} 0 & s < 0 \\ 1 & s \geq 0 \end{cases}$$

- sigmoid (logistic function)

$$y = \frac{1}{1+e^{-\beta s}}$$

- sin
- tanh
- sign

$$sgn(s) = \begin{cases} -1 & s < 0 \\ 0 & s = 0 \\ 1 & s > 0 \end{cases}$$

- ReLu

$$ReLu(s) = \begin{cases} s & s > 0 \\ 0 & s \leq 0 \end{cases}$$

- leaky ReLu

$$lReLu(s) = \begin{cases} s & s > 0 \\ 0.01s & s \leq 0 \end{cases}$$

At least the Heaviside and logistic functions must be implemented. Training of the neuron should support the Heaviside (assume that the derivative is 1) and logistic function. Variable learning rate may be implemented but is not required.

$$\eta(n) = \eta_{min} + (\eta_{max} - \eta_{min})(1 + cos(\frac{n}{n_{max}}\pi)) \tag{2}$$

The GUI should present the decision boundary by setting two background colours for two half-planes.

4.2.1 Grading

Evaluation and training of a neuron with Heaviside and logistic activation functions + decision boundary visualization in GUI → 3
Addition of sin and tanh activation functions for evaluation and training OR introducing variable learning rate → 4
Addition of sign, ReLu and leaky ReLu activation functions for evaluation → 5

4.3 Shallow neural network

Implement a shallow (up to 5 layers) fully connected neural network. The network **interface** should be based on the one implemented in the task #2 and use the data generated by the task #1 as its input and yield the predicted class membership at its output. According to theory, any data classification problem should be solvable using a three-layered network, thus at the minimum the implemented solution should support the evaluation and training of such a network. The output should be presented in the form of two values describing the confidence that the network belongs to each class. The network should consist of an input layer (with two inputs but multiple neurons i.e. **more than two**), a two-neuron output layer, and at least one hidden layer, the remaining values of width and depth of the network should be configurable. The neurons should use the logistic activation function, though the implementation may be extended by adding more options. The network should be trained using the backpropagation formula:

$$\Delta \vec{w_j^k} = \eta \delta_k f'(s^k)\vec{x}_l = \eta \Sigma_i(\vec{d^i} - \vec{y^i}) f'(s^i) W_k^i f'(s^k)\vec{x}_l \tag{3}$$

As was the case in task 2, the GUI should present the decision boundary for the network through colouring the corresponding parts of the plot. The model should operate on a **dataset with multiple modes per class**.

4.3.1 Grading

Evaluation and training (backpropagation) of 3 layered fully connected neural network + decision boundary visualization in GUI → 3
Parametrizing number of layers / neurons per layer → 4
Addition of multiple activation functions OR training the samples in batches → 5

4.4 Search algorithms

Implement a simple route finder for the game of snake. Simple implementation of Snake using PyGame[10] has been provided on GitHub[11] The game has been modified to include an additional type of tiles that reduces the score and the length of the snake by 1 and as such should be avoided. These tiles should be considered when implementing the cost function. Each time the algorithm should generate a **full path** to the "fruit" block, which will then be executed in sequence. The search algorithm should automatically stop when no path to the "fruit" can be found. Visited nodes should be coloured blue using the functionality provided in the script. The code should allow for running each implemented algorithm separately.

4.4.1 Grading

Working solution implementing **both BFS and DFS** → 3
Solution that implements BFS, DFS and Dijkstra's algorithm → 4
Solution that implements BFS, DFS, Dijkstra's and A* algorithm → 5

[10] `https://www.pygame.org/`
[11] `https://github.com/PALuczak/ArtificialIntelligenceFundamentals/blob/master/Snake.py`

4.5 Fuzzy control

Implement a fuzzy controller capable of controlling a paddle in a classic Pong game[12] a suitable, simplified implementation has been provided on GitHub[13]. The controller should be implemented using scikit-fuzzy[14]. The controller will have access to two values: the distances between the center of the ball and the center of the paddle in X and Y dimension. The controller should be responsible for **choosing both the speed and the direction of the paddle**. The aim is to deflect the ball every time and not let the Opponent score. In comparison with the classic version, this implementation provides a different way of increasing the speed of the reflected ball. If the ball is deflected using the edgemost 25% of the paddle, its speed increases by 10%. As the speeds of the paddles are limited, at some point the provided naive Opponent will not be able to keep up.

4.5.1 Grading

Basic ball-following ruleset with consequents defined using Mamdami's method $\rightarrow$ 3
Basic ball-following ruleset with consequents defined using TSK method $\rightarrow$ 4
More complex ruleset that attempts to reflect the ball using edges of the paddle $\rightarrow$ 5

[12] `https://en.wikipedia.org/wiki/Pong`
[13] `https://github.com/PALuczak/ArtificialIntelligenceFundamentals/blob/master/Pong.py`
[14] `https://pythonhosted.org/scikit-fuzzy/`