Department of Computer Science & Applied Physics

B.Sc. Software Development
Artificial Intelligence Assignment 2025

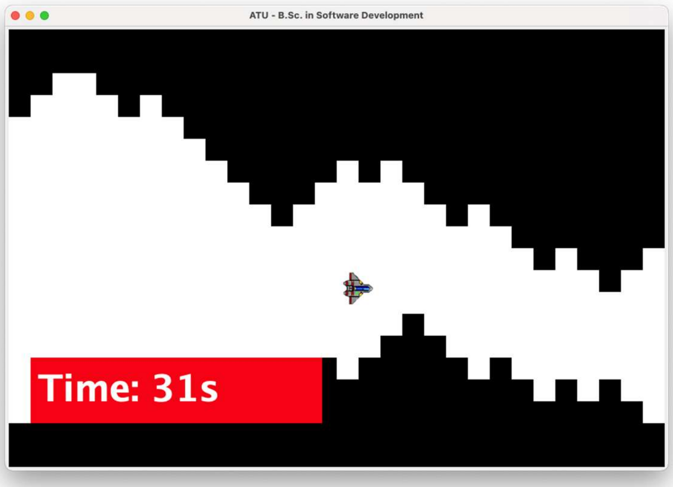
ASSIGNMENT DESCRIPTION & SCHEDULE

Teaching a Neural Network to Fly Autopilot

Note: This assignment will constitute 40% of the total marks for this module.

1. Overview

You have already seen how to use neural networks for classification and regression tasks, including the processing of images. We can also use neural networks in real-time control systems. Indeed, the ***Universal Approximation Theorem*** implies that a neural network with a single hidden layer of a sufficient number of neurons configured with a non-linear activation function can approximate any continuous function to an arbitrary level of accuracy. In this assignment, we are going to use a neural network as an auto-pilot controller for a plane in a game. The objective of the controller is to navigate the plane through an endlessly scrolling tunnel without touching its edges. As shown below, the plane is fixed on the *X*-axis but can move up or down on the *Y*-axis.

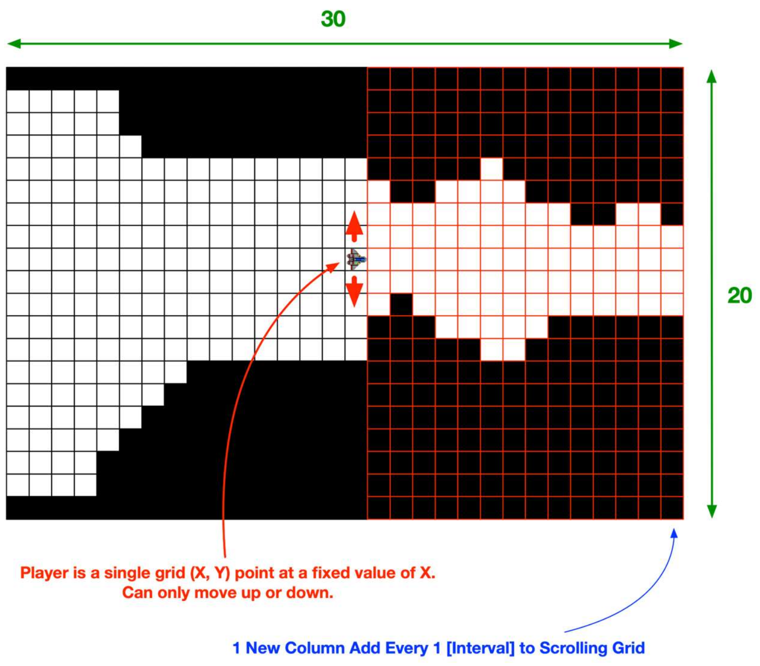


You are required to train a neural network to control the plane through the scrolling tunnel without any human intervention for as long as possible. A threshold of 30 seconds should be considered a very good result. A set of stubs for the actual game itself is available on Moodle, and you are not required to implement any game logic. Your task will be to extract training data from the game, either by manually controlling the plane yourself or using some other method, and then train the network to successfully control the plane on its own.

The neural network should be trained in a reasonable amount of time (< 1 minute) when the game starts up and have a fully documented topology presented in the README. You are free to use either the **SMILE**[1] or the **Encog 3.4** neural network APIs. Do not use any other 3rd party implementation. Any training and validation resources required by the neural network should be placed in the ***./resources/*** directory.

2. Minimum Requirements

The game is represented by a 30x20 grid of bytes, allowing a range of values between zero and 255 to be stored for each cell. In practice, however, it may be simpler to store just a zero or a one as the value in each cell. The column that the plane is located in is fixed, and the only movement possible is up or down on the Y-axis. In theory, the full 30x20 grid can be flat-mapped into a single-dimensional array. However, this arrangement includes all the cells behind the plane, which can probably be omitted. A better approach would be to only consider the *n* columns ahead of the player at any one time. Applying a horizon like this will reduce both the size of the vector needed to represent the game state and the amount of time it will take to process it. The code stubs include an example of how the plane may be auto-controlled using randomisation. The comments in the code stubs also show where and how the training data can be extracted.


30
20
Player is a single grid (X, Y) point at a fixed value of X.
Can only move up or down.
1 New Column Add Every 1 [Interval] to Scrolling Grid

The training data can be either saved to a file or generated on the fly and used to train the neural network while a player is controlling the plane. You have complete autonomy over the feature engineering, and you get to decide how you want to train the neural network. You should give careful consideration to the following:

Feature Engineering & Training Data

- The size of the vector to sample for training data.
- The frequency of the sampling.
- The label(s) to use for the training data.
- Any additional features/fields, e.g. metadata.

Network Topology

- The type of task (classification/regression)
- Number of nodes in the input layer.

[1] https://haifengl.github.io/; https://haifengl.github.io/api/java/smile/base/mlp/package-summary.html.

- Number of hidden layers and the number of nodes in each hidden layer.
- Number of nodes in the output layer.

Network Configuration

- Normalisation/Standardisation.
- Activation Functions.
- Training Hyperparameters, e.g. alpha, epochs, error.
- Saving/comparing network models.

Testing / Validation

- Time/number of movements
- Training error vs. time

You are free to asset-strip any online resources for functionality provided that you modify any code used and cite the source both in the README and inline as a code comment above the relevant section of the programme. You are not free to re-use whole components and will only be given marks for work that you have undertaken yourself. Please pay particular attention to how your application must be packaged and submitted in the scoring rubric provided. Marks will only be awarded for features described in the scoring rubric.

3. Deployment and Delivery

Please **read the following carefully** and pay particular attention to the files that you are required to submit and those that you should not include with your assignment:

- ***The project must be submitted by 11:59 PM on Friday, April 11th, 2025***. Before submitting the assignment, you should review and test it from a command prompt on ***a different computer*** to the one that you used to program the project.
- The project must be submitted as a Zip archive using the Moodle upload utility. You can find the area to upload the project under the *"Programming Assessment Submission (40%)"* link on Moodle. Only the contents of the submitted Zip will be considered. ***Do not add comments to the Moodle assignment upload form.***
- The name of the Zip archive should be <*id*>.zip, where <*id*> is your ATU student number.

Do NOT submit the assignment as an IDE project (e.g. Eclipse or VS Code projects) or submit any text files or other resources. If you do, you will lose marks. You will also lose marks if you hard-code any environmental variables in your project, like system paths and file names.

- The Zip archive should have the structure shown below.

Marks	Category
ai.jar	A Java archive containing your API and runner class with a ***main()*** method. For example, you can create the JAR file using the following command from inside the "bin" folder of the Eclipse project: **jar –cf ai.jar *** The application should be executable from a command line as follows: **java –cp ./ai.jar ie.atu.sw.Runner** You may assume that both the **SMILE** and **Encog** neural network APIs are already on the CLASSPATH.
src	A directory that contains the packaged **source code** for your application.

resources	A directory that contains any **saved models or training and validation resources** used by the neural network.
README.pdf	A PDF file detailing the **design and main features** of your application. Marks will only be given for features that are described in the README.

4. Marking Scheme

Marks for the project will be applied using the following criteria:

Element	**Marks**	**Description**
Efficacy Form	5	The self-efficacy form for the assignment was completed fully and submitted before April 11th.
Structure	5	*All-or-nothing.* Submitted on time and the packaging and deployment are both correct. The JAR file executes perfectly without any manual intervention when executed with the command **java –cp ./ai.jar ie.atu.sw.Runner.**
README	25	All **features and their design rationale** are fully documented. Your README should clearly explain your rationale for the following: • The design and extraction of training data. • The neural network topography and configuration. • Training hyperparameters. Your README must address each of the items above clearly and comprehensively.
Feature Engineering	35	Training data is well-designed and extracted efficiently from the game.
Integration with Game	10	The neural network is well-configured, working and fully integrated with the game.
Training Time	10	The neural network should be trainable in a short amount of time (< 1 minute) and capable of controlling the plane for 30 seconds.
Extras	10	Only **relevant extras** that have been fully documented in the README.

You should treat this assignment as a project specification. Any deviation from the requirements will result in some or all marks not being earned for a category. Each of the categories above will be scored using the following criteria:

Range	**Expectation**
0–39%	***Not delivering*** on basic expectations. The submission does not meet the minimum requirements.
40-60%	Meets ***basic*** expectations. The minimum requirements are met in whole or in part.
61–74%	Tending to exceed expectations. A ***high-quality*** assignment with additional functionality added.
75-89%	Exceeding expectations and ***very high-quality***. Any mark over 70% requires evidence of ***independent learning*** and cross-fertilisation of ideas, i.e. your project must implement a set of features using advanced concepts not covered in depth during the module.
90-100%	***Exemplary***. Your assignment can be used as a teaching aid with little or no editing.