UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Faculty of Engineering, Built Environment and Information Technology

EAI 320

Intelligent Systems

Practical 4 Guide

7 March 2019

Concept: Prof. Warren P. du Plessis
Guide written by: Llewellyn Strydom

I. General Instructions

This section contains general instructions which are applicable to all reports.

- The prescribed LaTeX format should be used, and the generated portable document format (PDF) file should be submitted.
- The commented Python source code should also be submitted.
- Reports must be submitted via the EAI 320 ClickUP page. Do not email reports to the lecturer or Assistant Lecturer (AL) as emailed reports will be considered not to have been submitted.
- The names of submitted files must have the format provided below.
  `eai320_prac_{practical number}_{student number}.{extension}`
- No late assignments will be accepted. No excuses for late submission will be accepted.
- Each student must do their own work. Academic dishonesty is unacceptable and cases will be reported to the university Legal Office for suspension.
- The report must include the standard cover-page declaration for individual assignments provided in the General Study Guide of the Department of Electrical, Electronic and Computer Engineering. The declaration is not included in the LaTeX template automatically and needs to be explicitly uncommented to indicate agreement.
- All information from other sources must be clearly identified and referenced.
- Text and code may not be included as images, and reports that do so will be considered not to have been submitted. Axis labels, legends, and other information normally included in figures may form part of a figure image.
- Computer code should not be included in the report as it is submitted separately.
- The code provided to students should also not be included in submissions.
- Any attempt to interfere with the operation of the framework used (e.g. to modify the scores of agents) will be regarded as academic dishonestly.

A. *Submission*

Reports should only be submitted via the

- `Practicals` → `Practical 4` → `Practical 4 Report`
- `Practicals` → `Practical 4` → `Practical 4 Code Task 2`
- `Practicals` → `Practical 4` → `Practical 4 Code Task 3`

links on the EAI 320 ClickUP page.

Only assignments submitted in via ClickUP will be accepted. Do not email reports to the lecturer or AL as emailed reports will be ignored.

Late reports will not be accepted! Accepting late reports is extremely unfair on those students who submit their work timeously because their tardy colleagues are effectively given additional time to complete the same work. Students are advised to submit the day before the deadline to avoid inevitable problems with ClickUP, internet connections, unsynchronised clocks, load shedding, hard-drive failure, computer theft, etc.. Students who choose to submit close to the deadline accept the risk associated with their actions, and no excuses for late submissions will be accepted.

Students will be allowed to submit updated copies of their reports until the deadline, so there will be no excuse for submitting late. Rather be marked on an incomplete early version of your report than fail to submit anything.

B. Academic Dishonesty

Academic dishonesty is completely unacceptable. Students should thus familiarise themselves with the University of Pretoria's rules on academic dishonesty summarised in the study guide and the university's rules. Students found guilty of academic dishonesty will be reported to the Legal Office of the University of Pretoria for suspension.

Students are required to include the standard cover page for individual assignments provided in the General Study Guide of the Department of Electrical, Electronic and Computer Engineering as part of their reports. This standard cover page includes a statement that the student submitting the report is aware of the fact that academic dishonesty is unacceptable and a statement that the submitted work is the work of that student. Failure to include this cover page will mean that the report will be considered not to have been submitted.

While students are encouraged to work together to better understand the work, each student is required to independently write their own code and report. No part of any student's work may be the same as any part of another student's work.

Students should clearly indicate material from other sources and provide complete references to those sources. Examples of commonly-used sources include the textbook [1] and this document [2]. Note that this does not mean that students may reuse code and/or information found in books, on the internet, or in other sources as students are required to complete the tasks themselves.[1]

II. Scenario

Artificial neural networks (ANNs) refer to a framework of machine learning algorithms that are inspired by the biological neural networks that constitute animal brains. ANNs learn to perform tasks and make decisions by considering examples, and are generally programmed without any task-specific rules. ANNs can approximate functions that represent continuous, discrete or categorical data, as long as the data are appropriately encoded. This feature means that neural nets have a wide range of applicability [3].

This assignment will require students to implement an ANN with backpropagation. The trained ANN will be used to propose objects for a rock-paper-scissors (RPS) agent to play during a match. The same RPS framework that was used for the previous assignment will be used again [4].

The ANN will take as input the last 2 moves from the game, and should return a single proposed object. This approach is similar to that of the genetic algorithm (GA) from the previous practical [5]. The inputs and outputs of the ANN can be represented in a number of ways, and it is up to the student to decide how they would do this. Figure 1 represents an example of an ANN that uses a 1-of-K (also one-hot) encoding scheme [6] for the inputs and the outputs. Remember that although the network in Figure 1 only considers the last move, the ANN that will be implemented for this practical will consider the last two moves.

[1]The objective of all academic assignments is fundamentally that students learn by completing the assignments. Merely reusing code and/or information found elsewhere defeats this objective because a key part of the learning process is performing the tasks oneself.

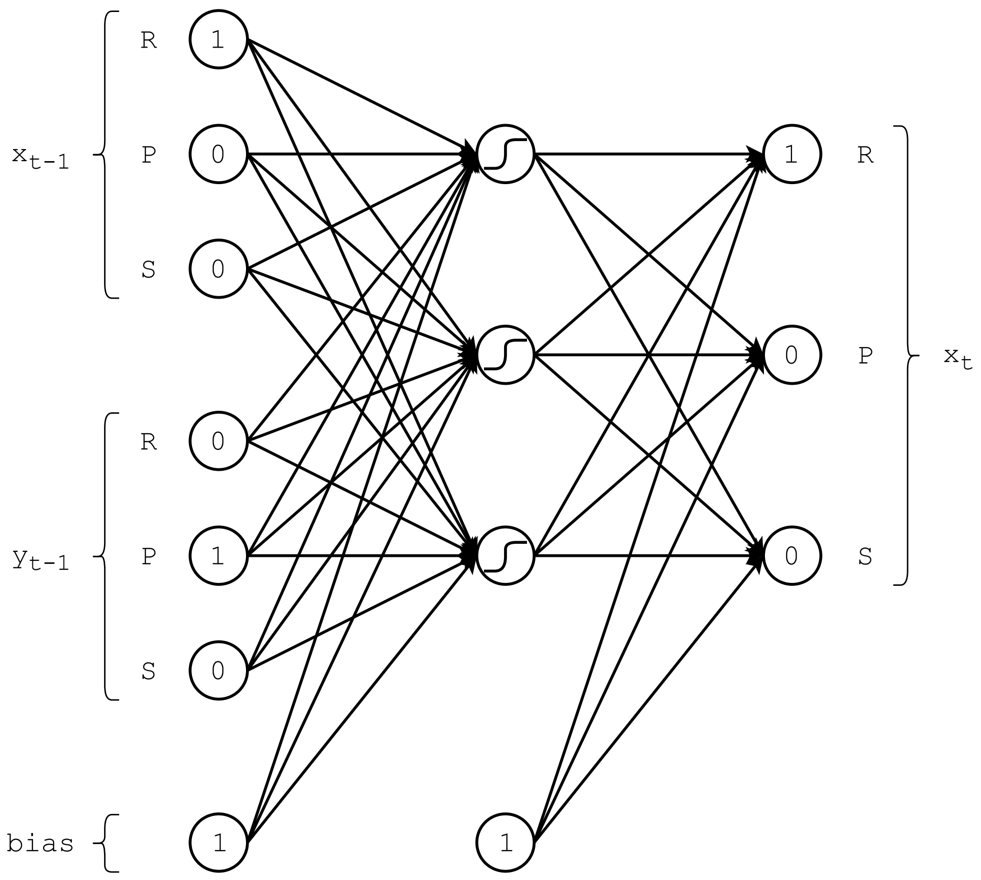

Fig. 1: An example of an ANN with one hidden layer that takes the previous move as input and outputs a proposed object. Note that the bias neurons are represented as neurons with a fixed value of 1, which makes training easier than calculating separate gradients for the weights and the biases.

III. Instructions

A. *Task 1*

Implement the backpropagation algorithm, for ANNs, as described in the prescribed textbook [1], as a Python function. The function should perform backpropagation on a three (or more) layered network with D inputs, H hidden neurons and C outputs. Both the input layer and the hidden layer(s) should contain a bias neuron(s).

The function should take the following input arguments.

- The input data as a $D \times N$ array, where D is the number of input features and N is the number of data samples. (Note that D excludes the bias neuron).
- The target values as a $C \times N$ array, where C is the number of output neurons.
- The initial values of the weights from the input layer to the hidden layer as a $(D+1)\times H$ array. H is the number of neurons in the hidden layer, excluding the bias neuron.
- The initial values of the weights from a hidden layer to the next hidden layer as an $(H+1)\times H$ array, if there is more than one hidden layer.
- The initial values of the weights from the hidden layer to the output layer as a $(H+1)\times C$ array.
- The learning rate, η.
- The terminating conditions in the form of number of epochs and error convergence.

The function should return the following parameters.

- The updated input-to-hidden layer weight array
- All updated hidden-to-hidden layer weight arrays, if there is more than one hidden layer.
- The updated hidden-to-output layer weight array.
- A vector of loss values for each epoch (see discussion below.)

In each epoch (training iteration), the function should compute the L_2 loss

$$E = \frac{1}{2}\sum_k (t_k - z_k)^2$$

where t_k and z_k refer to the target and return values of a single output neuron k. The values of this loss function should be stored in a vector with a length equal to that of the total number of epochs. A graph of the L_2 loss versus the number of iterations should be included in your technical report.

The error convergence criterion can be calculated as the absolute value of the difference between the current error and the previous error. The algorithm should terminate when this reaches a specified threshold.

Any activation function that is differentiable over its entire domain may be used for each of the neurons.[2] The activation function and its derivative should be implemented in separate functions.

1) Suggestions: The inputs and outputs to the ANN can be represented in a number of ways. An intuitive approach is to represent “R,” “P,” and “S” using a 1-of-K (also one-hot) encoding scheme as illustrated in Fig. 1. Another approach is to simply represent “R,” “P,” and “S” by a -1, 0, and 1, and other possibilities no doubt exist. You can implement a function that converts a history of two moves to the input format that you have chosen to use during training and inference.

NumPy arrays are convenient for storing the weights of your array, as they allow easily and efficient matrix operations. Use this to vectorise your code.[3]

This assignment requires you to include at least one hidden layer, but you are allowed to experiment with more layers.

B. Task 2

Train an ANN on the data that is provided in `data.csv` using the backpropagation algorithm that you have implemented. This file is the same file that was used for the previous practical and as such, the data are represented in the same format [5].

The student is expected to experiment with different hyperparameters, and all students should ensure that they answer the questions that are listed below as part of their discussion.

- How does the size (and number) of the hidden layers affect the performance of the ANN?
- How are the input data represented?
- How are the output data represented?
- How does the learning rate η affect the training process?
- How did the loss increase/decrease during training?

After training your network, you should use the network that you have trained to implement a fully-functional RPS agent. Please note that this agent must already be trained and that it should not implement backpropagation. The agent must interface seamlessly with `rpsrunner.py` and should thus follow the same format as agents from previous practicals.

[2]While backpropagation may work if there are points at which the activation function is not differentiable, problems may be encountered.

[3]Vectorisation is the process of replacing loops by operations which inherently perform multiple operations. Examples of vectorisation are to use vector and matrix operations instead of looping through a large number of values.

1) Suggestions: While testing and debugging, it is a good idea to save the weights of your network to a file, and then initialise your ANN at the start of each match by reading the values from the file. This will also allow you to train your network from a specific checkpoint, rather than having to train from scratch every time. However, for the final submission you will have to hard-code the values, as the lecturer and the marker will not have access to the file containing the weights.

C. Task 3

Adapt the agent from Task 2 to learn online while playing a match of RPS. You will do this by performing backpropagation after every n moves, where n can be any number between 1 and 100. The goal of this agent is to learn how to beat its present opponent, so the training should only consider the data from the current match and not any previous training data. Initialise your network using the same weights that you used for Task 2, or else your network will produce very bad results in the first few rounds.

In order to ensure that you have written efficient code and to allow for easy testing, you must ensure that this agent can play 100 matches of 10 000 rounds each against a simple bot (like `only_rock.py` or `only_scissors.py`) in less than 10 minutes.

1) Suggestions: The learning rate η will have a large influence on the performance of the agent, and thus you should choose it carefully by experimenting with different values.

If the agent does not meet the timing requirements, the value of n should be increased until the agent meets the requirements.

IV. Submission Requirements

Marks will primarily be awarded on the basis of the motivations provided. Merely obtaining the correct results is not nearly as important as describing how the results were obtained.

A. Report Content

Each student is required to submit a report providing the information listed below. Failure to comply with these instructions will lead to marks being deduced.

- Provide an introduction to the assignment.
- Explain how you have implemented a ANN with backpropagation.
- Provide the results that were obtained.
- Provide a discussion of the results. Make sure to discuss your ANN implementation and how different parameters of the algorithm effect its performance.
- Use the prescribed LaTeX format.
- The submitted computer code must **not** be included in the report, otherwise TurnItIn will flag a 100% similarity.
- All the material necessary to produce a formal document (e.g. an abstract, introduction, and conclusion). These parts of a document are not included in practical guides to shorten them.

B. Code Instructions

Each student is required to submit code conforming to the requirements listed below. A mark of zero will be awarded if the submitted code file does not run, produces errors, or does not follow the instructions.

- All code must be commented to a point where the implementation of the underlying algorithm can be determined.
- The submission must be a single file.
- Submitted code will be evaluated using the command

```
rpsrunner.py -m 100 -r 10000 agent.py only_rock.py,only_paper.py,
          only_scissors.py,beat_previous.py,beat_common.py.
```

- All print statements and any other functions that were used for unit testing etc. must be removed before submission.
- All submissions must be written using Python 2.7 as a result of the fact that `rpsrunner.py` is written in Python 2.7.
- The code submitted must be the final implementation of Task 1.
- TurnItIn will not accept code with the file extension `.py` or `.csv`, so the file extensions should be changed to `.txt` before submission.

References

[1] S. J. Russell and P. Norvig, *Artificial intelligence: A modern approach*, 3rd ed. Prentice Hall, 2010.

[2] L. Strydom, *EAI 320 – Practical 4 Guide*, University of Pretoria, 8 Mar. 2019.

[3] S.-C. Wang, *Interdisciplinary Computing in Java Programming: Artificial Neural Networks*, 2nd ed. TRIUMF, 2003.

[4] B. Knoll. (2011, 6 Feb.) Rock paper scissors programming competition. [Online]. Available: http://www.rpscontest.com/

[5] L. Strydom, *EAI 320 – Practical 3 Guide*, University of Pretoria, 21 Feb. 2019.

[6] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

Appendix A
Abbreviations

AL	Assistant Lecturer
ANN	artificial neural network
GA	genetic algorithm
PDF	portable document format
RPS	rock-paper-scissors