CS5011: Machine Learning

120010815

November 27, 2015

1 Introduction

The aim of this practical is to gain experience with machine learning and designing and using neural networks. The goal of this practical is to two systems based on neural networks, one of which is capable of scripted learning and the other capable of unscripted learning. Both of the neural nets are supposed to be tested on a Whack-a-Mole game in one dimension.

2 Part 1: Scripted Learning

2.1 Setting up

In order to make the network successful, it was important to find the correct settings and choose the ones that work best for the current purpose. The important settings are the amount of neurons per dimension, volume neuron width, the radial basis function, and training algorithm. After thorough testing it was concluded that 50 neurons per dimension provide a solid base for learning and despite making the training process slower, provide a much better result than a value suggested in different sources. The volume neuron width index was chosen to be 0.018.
There are four possible functions provided by Encog: Gaussian, Mexican Hat, Multiquadratic and Inverse multiquadratic. Four training methods provided by Encog were also tried: Resilient Propagation, Back Propagation, Quick Propagation, and Scaled Conjugate Gradient. After this, tests were run on all possible combinations of those settings and results were output to a file called *testresult.txt*. For each of the tests, time needed for one learning

iteration, overall training time, amount of perfectly predicted values and amount of values predicted for a mallet of size 2 were recorded. After the result has been analysed, it was decided that the best composition for the current purpose is a Gaussian Radial Basis Function trained with Resilient Propagation. It was not the only composition which produced 100 per cent of correct results, but it was also fast to learn.

2.2 State Transition Function

The Finite State Machine for the current implementation was chosen have two states. If a mallet is in an even position, then the state 0 is activated, otherwise the game is assumed to be in state 1. The agent predicts the next state, but the game still passes the correct state to the agent in order to make the environment of the agent to be equal to the environment of the game. If there was more time to work on this practical, another amount of states would be chosen for testing since the results for state output were not as good as the guesses for the mole position. One of the possible reasons for that it that values 0 and 1 are small comparing to the range of numbers between 0 and 100, and therefore, those numbers, together with the whole second output, did not "weight" as much as the rest of the net.

2.3 Output function

In the current implementation, the output depends on the current state and the clue. If the game is in state 1, then the mole will appear 5 spaces to the left from the clue. Otherwise it will appear 5 spaces to the right from the clue.

2.4 Topologies

Kohonen and Analog topologies were designed and tested for the current implementation. For the analog topology, the network had two output nodes, the first of them showing the position of the mole and the second one predicting the state of the game. For the Kohonen topology, the network had $100 + 2$ outputs. The first 100 outputs were for the mole position prediction. The outputs were supposed to ideally be between 0 and 1. The values for the state prediction were scaled and inverse-scaled, and the biggest value was chosen in a similar way.

2.5 Training Data and Iterations

Because the training data set was not very big, it made sense to put all the possible inputs and outputs into the training set. The network was also tested on a bigger FSM and only part of the possible inputs was used for training, the network then gave slightly worse, but overall similar results. Different amount of iterations was to be used for every function and training method, but overall it was clear that a number f iterations of about a 1000 was enough to train a network since after that the error rate converged. Thorough testing also proved it.

3 Part 2: Unscripted Learning

For this part of the practical, no traditional training was used, and the network had to be trained "on the go". Another class for created for this purpose for the simplicity of testing. The logic behind this implementation is as follows: An Radial Basis Function network is initialised for the Gaussian RBF. It is not being trained at the beginning, so all the weights have 0 value (although theoretically they could have any value). The agent receives all the data necessary for training: the usual clue and the current state, but also the correct answers to the tasks. The task of the agent at at time is to make the network give an answer as close to the correct as possible. For every input, a set of configurations is created. A configuration is an array of very small positive and negative values which are supposed to be applied to the current array of weight values in order to change the behaviour of the network. After thorough testing it was decided that a 100 configurations for every input is enough and produces result on the same level as the implementations with more configurations created per input. After the configurations is created, it is applied to the copy of the current setting of the network. Then the network with this new setting tries to predict the new position of the mole and a state, after which the result and the error for the latest output get recorded in a hash map. Then, the original copy of the weights is returned. Such experiments run for every configuration, after which the best configuration (that produces the smallest error) is chosen. Since a configuration is a set of small changes, we want to keep improving the network. If the error produced is smaller than the error of the original guess, it means that the configuration is pointing in the right direction of improvement. Therefore, it would be

wise to keep allying those changes until the error reduction rate converges. After that, the error is much lower and we can make another attempt to predict the output for the game. Overall, this method uses a technique similar to the Gradient Descent, except the progress of the changes is assesses with a function evaluating error rather than as a gradient, which makes the implementation easier to understand.

3.0.1 Problem of Over-fitting

Of course, if at the end of applying the changes the error still exists, the process could be repeated again with another set of configurations. However, this way of implementation creates a possible problem of over-fitting the problem, a problem which is very popular in machine learning. Probably, another design to use would collect past inputs and correct answers and try to learn from the past data, too. However, that would involve re-setting the network weights every time and, thus, would make the network re-learn everything with every input, which makes it less efficient and is this worse than the current implementation.

3.0.2 Environment function

The environment function chosen for this agent was

$$h(t) = t + 1 \tag{1}$$

since it grows with the time. The scoring systems evaluates the error as a sum of the two errors: error in predicting state and error in predicting the position of the mole. No other script is considered. Instead, the agent always tries to make a guess and then trains to produce a correct result.

4 Further Questions

4.1 Fault Tolerance

Every effort has been made to make the system as fault tolerant as possible. The possible "dangerous" places are the inputs from the game or inputs from the user. If an input from the game does not make sense, then the output will simply not correspond to what is considered to be the correct answer. The program is fully trained to deal with problems of human input.

4.2 Scripted vs Unscripted Learning

4.2.1 Advantages of Scripted Learning

It is quite clear why most computers and robots are scripted: the behaviour of such systems is much more predictable and much more restricted than the behaviour of the unscripted systems. In addition, a system with scripted learning mechanism does not require any sensors is it can be trained fully before starting work. A system with unscripted learning mechanism requires not only input, but also to keep a constant dialogue with the environment in order to receive feedback for its outputs in order to be able to learn further.

4.2.2 Advantages of Unscripted Learning

There is, however, a whole sector of tasks that systems trained with unscripted learning would be good at. Those tasks include environments which constantly change and thus require constant exchange of information between the robot and the environment and tasks where there is more than one correct answer. The second part of the practical shows on of those cases: the rule for the game changes with the time as it grows with the time, so the robot has to be able to adapt to the new environment and be able to train to it quickly.

4.2.3 Visual Representation of Different Underlying Mechanisms

After a thorough testing of both scripted and unscripted systems, it can be claimed that it is possible to visualise the differences between random,scripted and unscripted numerical output by looking at the patterns of the output and its relation to the correct results.
The random pattern is the easiest to notice since there will be no relation between any of the outputs and between each output and each correct result. In order to visualise the difference between scripted and unscripted learning, it is necessary to analyse the output results thoroughly. For example, the limits of the scripted systems will most often coincide with the limits of the possible correct results, whereas the limits of the unscripted agents will be more sparse. In general, unscripted agents are harder to learn, which means that the outputs will not be as close to the correct results, but they will still be in a close proximity to them, if the agent is successful. In addition, the performance of such an agent might change with the time, whereas the

performance of a scripted agent is likely to stay the same, given a consistent environment.

5 Design and Implementation

This section includes notes on the implementation of the system.
The agent re-trains if the results are not good enough, but continues the game instead of starting it from the beginning.
Neural Net for Part 1 in implemented in Java class NeuralNet. It produces both Kohonen and Analog results depending on the setting provided in the input. The results of the Kohonen topology are better than the results in the Analog topology.
The game result is being recorded in a Result object for further analysis. One of the properties of this object is "wrap error", which is a variable that count occasions when the agent was wrong around the edges of the Whack-a-Mole board. Such cases can be tricky for the agent since the program treats the board as a ring, s, for example, if a clue appeared at a slot 99, then, together with the current output function, the mole can appear at either 94 or 4, which is not easy to get trained for. However, in a lot of cases the agent does recognize those situations and treats them right.
The target tolerance is 2 for both parts. This means that as long as error is less than two, the answer will not be accepted as wrong. The program does, however, also keep track of the number of values guessed with no error.

6 Testing

6.1 Part 1: Scripted Learning

In the implementation, the user in invited to choose one of the three options. When testing, for every input and output the program displays the results of a turn, where the number in brackets is the correct answer and the number in front of the brackets is the output given by the agent. Game statistic is displayed at the end of every game. Option 1 runs 32 tests for networks with different settings and outputs it on the screen. An example of such a run is output in *"testresult.txt"* Selection 3 only trains one network and plays a game with it. The output is also shown on the screen.

```
Please enter 1 for part1 or 2 for part 2:
Warning: part 1 runs a series of 32 test on RBF nets
 with differet settings. If you only want to see the game
 for the best network, please select 3
3
Training Gaussian RESILIENTPROPAGATION true
Epoch #10 Error:0.011789377478382785
Epoch #20 Error:0.011054065131151872
Epoch #30 Error:0.010787141971193294
Epoch #40 Error:0.01062873169442086
Epoch #50 Error:0.010516774903419773
Epoch #60 Error:0.010441675828753695
Epoch #70 Error:0.010375875444352195
Epoch #80 Error:0.010321821584970036
Epoch #90 Error:0.0102778241845458
RBF method: Gaussian
Train method: RESILIENTPROPAGATION
Kohonen: true
Train time: 19789
One round time: 197
Correct count: 100
Acceptable Count: 100
Wrap Error: 0
Correct state count: 59
------------------------------------------------------------
```

This shows the progress of the training (100 iterations).

```
90. Clue: 54, mole: 49(49), state guess 1(1)
91. Clue: 84, mole: 79(79), state guess 0(1)
92. Clue: 68, mole: 63(63), state guess 1(1)
93. Clue: 68, mole: 63(63), state guess 1(1)
94. Clue: 29, mole: 24(24), state guess 1(0)
95. Clue: 11, mole: 16(16), state guess 1(0)
96. Clue: 60, mole: 65(65), state guess 0(1)
97. Clue: 14, mole: 9(9), state guess 0(1)
98. Clue: 53, mole: 48(48), state guess 1(0)
99. Clue: 84, mole: 89(89), state guess 0(1)
RBF method: Gaussian
Train method: RESILIENTPROPAGATION
Kohonen: true
Train time: 18693
One round time: 186
Correct count: 100
Acceptable Count: 100
Wrap Error: 0
Correct state count: 55
-----------------------------------------
```

In this representation, it shows an output for every turn and the game statistics in the end.

6.2 Part 2: Unscripted Learning

Selection 2 runs the unscripted agent on one game and outputs the result on the screen.

```
Please enter 1 for part1 or 2 for part 2:
Warning: part 1 runs a series of 32 test on RBF nets
 with differet settings. If you only want to see the game
 for the best network, please select 3
2
Used 9 cycles
0. Clue: 31, mole: 24(25), state guess 0(1)
Used 2 cycles
1. Clue: 25, mole: 18(18), state guess 0(0)
Used 29 cycles
2. Clue: 82, mole: 83(84), state guess -1(0)
Used 11 cycles
3. Clue: 21, mole: 22(22), state guess -1(0)
Used 9 cycles
4. Clue: 56, mole: 56(56), state guess 0(0)
Used 9 cycles
5. Clue: 83, mole: 82(82), state guess 0(0)
Used 7 cycles
6. Clue: 44, mole: 43(42), state guess 0(0)
Used 2 cycles
7. Clue: 73, mole: 68(70), state guess 0(0)
Used 4 cycles
8. Clue: 80, mole: 76(76), state guess 0(0)
95. Clue: 93, mole: 92(92), state guess -2(0)
Used 9 cycles
96. Clue: 15, mole: 22(23), state guess 0(1)
Used 12 cycles
97. Clue: 75, mole: 71(72), state guess -3(0)
Used 9 cycles
98. Clue: 1, mole: 7(7), state guess 0(1)
Used 2 cycles
99. Clue: 18, mole: 13(13), state guess -2(1)
DONE----------------------------------
Correct count: 41
Acceptable Count: 96
Wrap Error: 0
Correct state count: 16
------------------------------------------
```

7 Summary

Overall, the aims and the goals of the practical have been reached. A system consisting of a scripted and unscripted agent has been provided and experience with working in machine learning has been gained. All of the tasks of the assignment have been completed and are fully functional and tested.