

# CPEN 502 Assignment-b: Reinforcement Learning (Look Up Table)

Ali Asgari Khoshouyeh (Student #24868739)

December 15, 2021

## Team Members

We are a team of three sharing the same code base.

- Christina Sun
- Husna Kalim
- Ali Asgari Khoushouyeh

It is noteworthy to mention that close to the extended deadline we realized that our code is orders of magnitude slower on my teammates' machines. So we sharing the plot data too.

*(4) The use of a neural network to replace the look-up table and approximate the  $Q$ -function has some disadvantages and advantages.*

*a) There are 3 options for the architecture of your neural network. Describe and draw all three options and state which you selected and why. (3pts)*

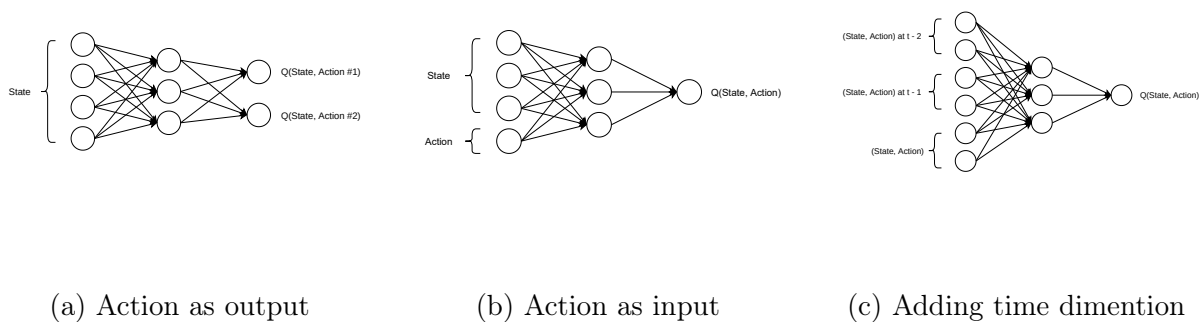


Figure 1: Different architectures considered for the neural network as the function approximator for the  $Q$  values of different game actions.

Fig. 1 shows the different architectures we considered for our neural network to be plugged in instead of the lookup table. The first two architectures shown in Fig. 1a and Fig. 1b are the ones discussed in the class. The first one codes the  $Q$  value for different possible actions at each step as different neurons of the output. In the second architecture the action is coded as some input neurons and the  $Q$  value corresponding the the encoded action as the input at a certain state is coded as the single output neuron.

The novel architecture that we used is the one shown in Fig. 1c which is essentially same as the second architecture, but, it also receives the state-actions of the two past time-steps as input. This is following our intuition used in the second part of the assignment where we included the past two state-actions in the lookup table keys to capture the delay of rewards as it may take some time for a bullet to hit the enemy and get reflected in the reward policy.

*b) Show (as a graph) the results of training your neural network using the contents of the LUT from Part 2. Your answer should describe how you found the hyper-parameters which worked best for you (i.e. momentum, learning rate, number of hidden neurons). Provide graphs to backup your selection process. Compute the RMS error for your best results. (5 pts)*

Learning Rate	Momentum	Hidden Neurons	RMSE
1.00E-04	0	5	9.666091389
1.00E-04	0	15	9.660078042
1.00E-04	0	30	9.628909289
0.001	0	5	7.632740399
0.001	0	15	6.943443362
0.001	0	30	6.711401519
0.01	0	5	7.985192317
0.01	0	15	7.018144655
0.01	0	30	6.530952131
1.00E-04	0.5	5	9.148578254
1.00E-04	0.5	15	8.993952348
1.00E-04	0.5	30	9.051175005
0.001	0.5	5	7.297405961
0.001	0.5	15	<b>6.335407802</b>
0.001	0.5	30	<b>6.289893926</b>
0.01	0.5	5	7.919711744
0.01	0.5	15	7.75168296
0.01	0.5	30	8.222388904
1.00E-04	0.9	5	7.713839065
1.00E-04	0.9	15	6.790881731
1.00E-04	0.9	30	6.754064308
0.001	0.9	5	7.772037228
0.001	0.9	15	7.099569645
0.001	0.9	30	6.728784713
0.01	0.9	5	11.50926595
0.01	0.9	15	13.85358874
0.01	0.9	30	18.89859967

Table 1: RMSE after 100 epochs under different hyper-parameter values for training the neural network on the static lookup table data.

We performed a grid search on different possible values of number of hidden neurons, momentum and learning rate. As the Q values in our lookup table ranged from  $-1.2$  to several hundreds, we removed the activation of the last layer, allowing the last fully connected layer to act as a regression solver. Please note that we will keep a bipolar activation when we start training with live data, as the limited range of the output prevents the Q values to go beyond the range.

The results are shown in Table 1. As the difference between 15 hidden neurons and 30 hidden neurons (shown bold in the table) is negligible, we keep 15 neurons as it is much

faster.

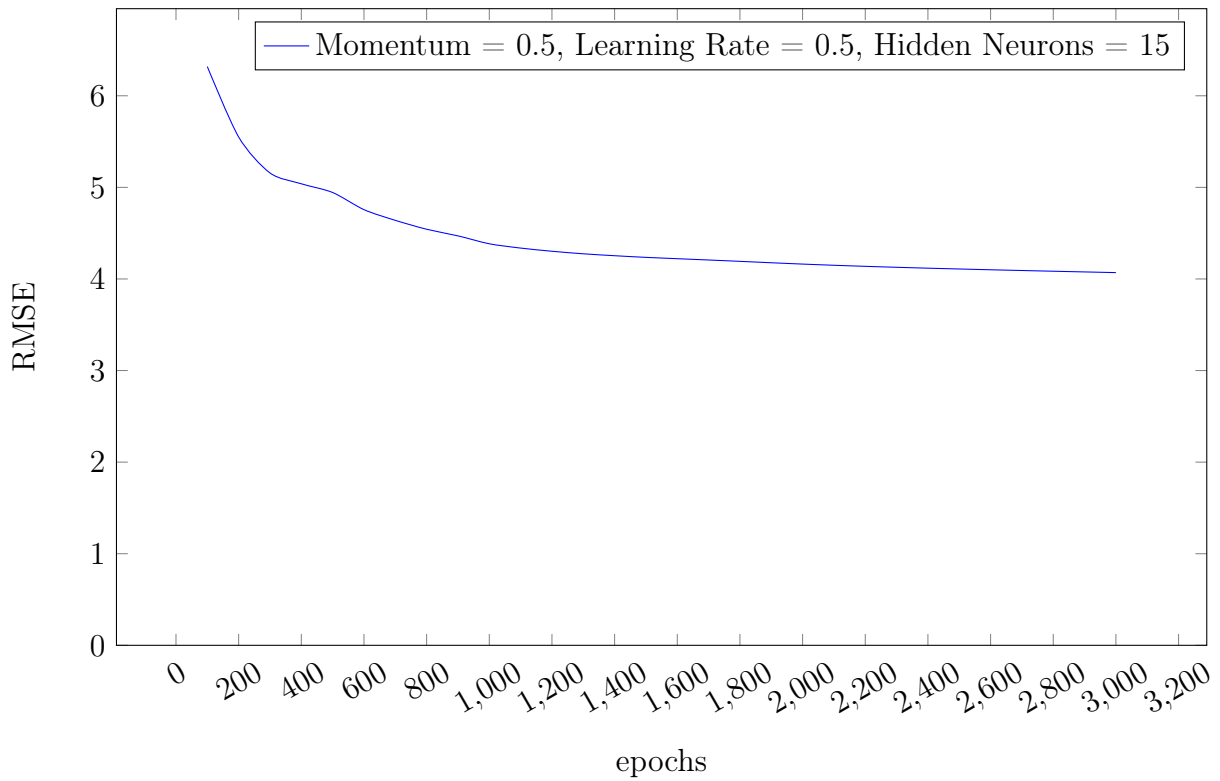


Figure 2: The convergence of the select hyper-parameters over static lookup table data.

We show the convergence of the selected settings in Fig. 2. It shows that our neural network is able to reduce the error fitting on the lookup table static data by decreasing the error from more than 6 to around 4.

*c) Comment on why theoretically a neural network (or any other approach to  $Q$ -function approximation) would not necessarily need the same level of state space reduction as a look up table. (2 pts)*

As an example, we are using the distance to the enemy as one of the dimensions of the states. When using a lookup table, it matters to decrease the number of possible values to few, so the number of entries in the lookup table do not blow up and the get revisited often. But as the neural network treats the input as a real number and does not do exact-match like the lookup table, close values also let the states be somehow recalled by the neural network. Thus, using the large number of possible values for the distant to enemy is still tractable by the neural network. That said, it is still important to limit the range of that value to something close to the other inputs so the weight initialization would fit this dimension like the other dimensions.

*(5) Hopefully you were able to train your robot to find at least one movement pattern that results in defeat of your chosen enemy tank, most of the time.*

*a) Identify two metrics and use them to measure the performance of your robot with online training. I.e. during battle. Describe how the results were obtained, particularly with regard to exploration? Your answer should provide graphs to support your results. (5 pts)*

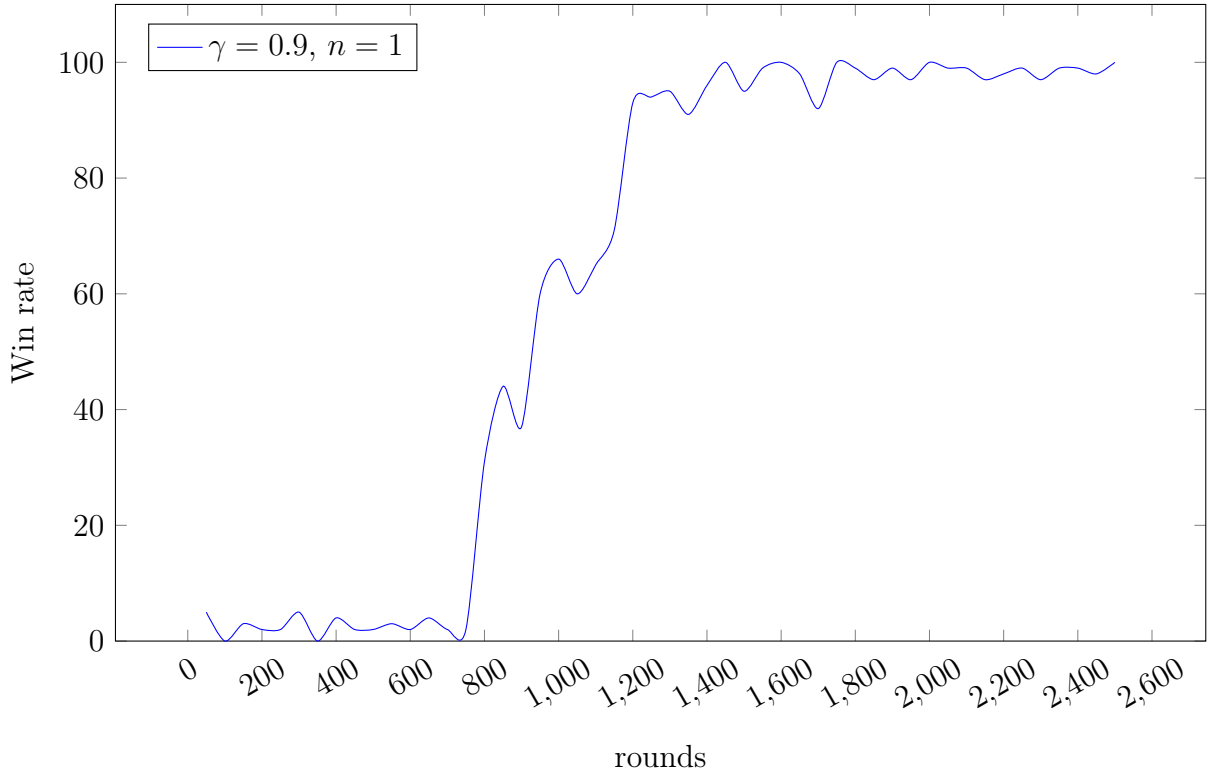


Figure 3: Win rate with regard to number of training rounds.

We measure the following metrics:

1. **Win rate:** The number of rounds won among 100 rounds.
2. **Average enemy energy:** The average enemy energy at the end of rounds for the course of 100 test rounds. Lower values show a better agent in causing damage to the enemy. This also will reflect our policy which only rewards on decrease in enemy energy.

We interleave battles of training with a robot with  $\epsilon = 0.8$  and test battles of 100 rounds with a robot with  $\epsilon$  set to 0.05. We only report the metrics for the test robot (i.e.  $\epsilon$  set to 0.05).

We realized that the learning process is a lot quicker if we remove the enemy distance from the input dimensions. This makes sense about our opponent which is the Corners robot. The Corners robot navigates to one corner and then tries to hit us. The best

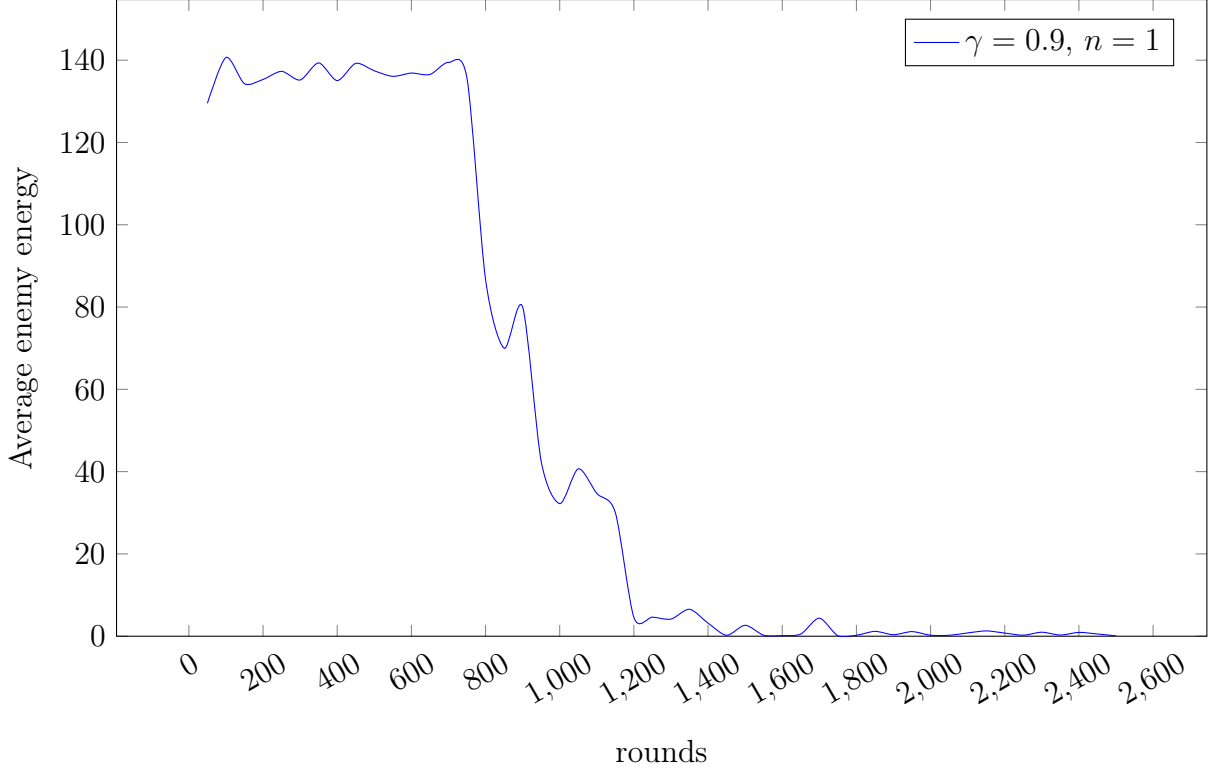


Figure 4: Average enemy energy with regard to number of training rounds.

strategy that our robot comes up with is to fire so often, and this strategy is independent of the distance to enemy. Thus, we are reporting the results with that simplification hereafter.

Fig. 3 shows that how the win rate for our robot increases after rounds of training. We observe a somehow abrupt increase in the win rate. My hypothesis explaining the reason this happens is that with the neural network the Q values change gradually towards the optimal values. The optimal actions are competing the other actions with such moving Q values. Probably, because of the fact that weights are shared for different state-actions, the optimal actions exceed the Q value of the other actions and win the competition in a short number of rounds. That said, they were shifting towards the competing actions since the beginning and exceed the Q value of other actions only after 1200 rounds.

As for the second metric, Fig. 4 shows that the average energy of the enemy decays as our agent learns more.

*b) The discount factor  $\gamma$  can be used to modify influence of future reward. Measure the performance of your robot for different values of  $\gamma$  and plot your results. Would you expect higher or lower values to be better and why? (3 pts)*

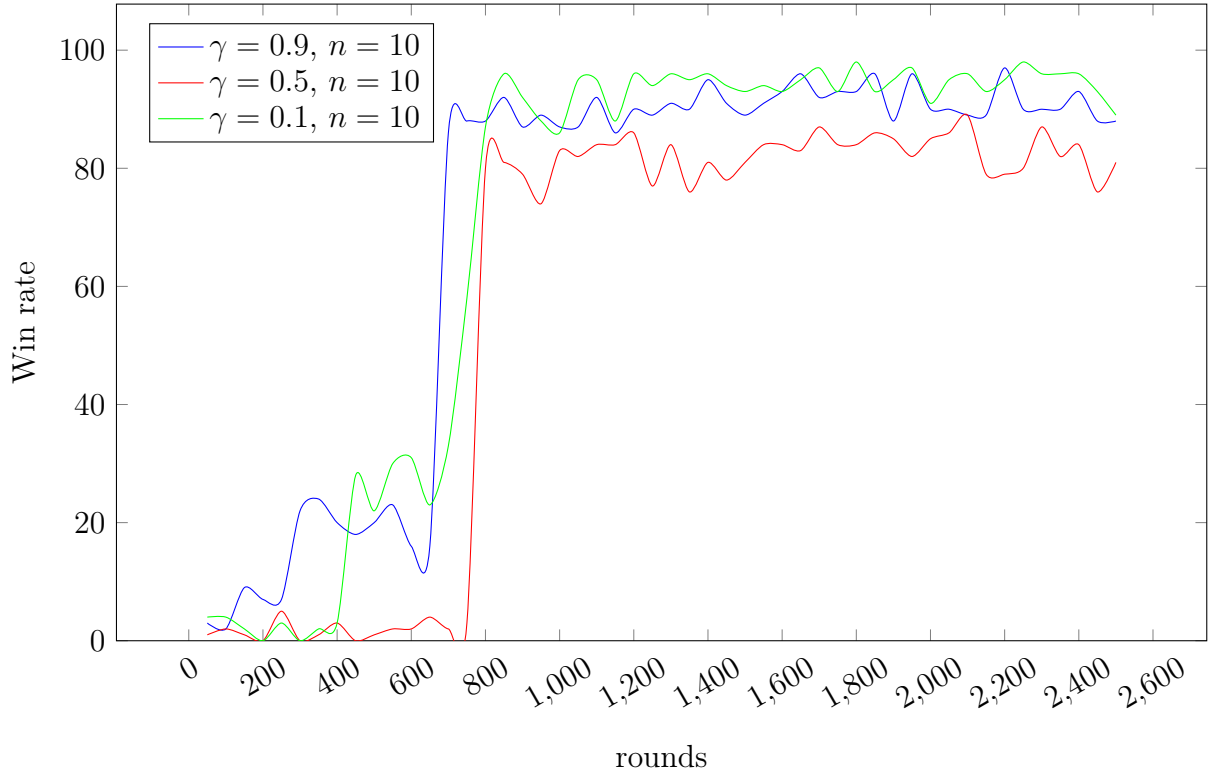


Figure 5: Win rate with regard to number of training rounds, under different  $\gamma$ .

Fig. 5 shows the convergence of win rate for different gamma values. As can be observed the trend is similar for  $\gamma = 0.1$  and  $\gamma = 0.9$ , but, the training seems to be more complex and converging at a less win rate for  $\gamma = 0.5$ . My hypothesis to explain this is that high gamma values will lead to quicker convergence in the Q-learning side, meaning rewards propagate quicker to the earlier stages. With the low gamma values the Q values used to update the neural network change slower, so the neural network has more time to adapt. But the intermediate gamma value (0.5) has none of the advantages. It is neither slow while providing updates to the neural network, nor needing less steps in the convergence of Q values, thus it leads to a less ultimate effectiveness of training.

Intuitively, there are two dynamical systems interacting with each other and trying to converge here, one is the neural network, the other is the Q-learning values. These two are chained together. High gamma values help the Q-learning and low gamma values help the neural network. But the intermediate value does not provide any of the advantages.



*c) Theory question: With a look-up table, the TD learning algorithm is proven to converge – i.e. will arrive at a stable set of Q-values for all visited states. This is not so when the Q-function is approximated. Explain this convergence in terms of the Bellman equation and also why when using approximation, convergence is no longer guaranteed. (3 pts)*

If you unroll the Q-value based on the Q-learning formula, assuming the reward is some stochastic random variable, we can define the Q-value as the following expected:

$$V(s) = E[r_0 + \gamma V(s_1) | s_0 = s]$$

so  $r_0 + \gamma V(s_1)$  is an unbiased estimate for the random variable  $V(s)$  <sup>1</sup>. This means initializing  $V$  with arbitrary values for states and then sampling the consequent states of the game ( $s$  and  $s'$ ), and updating the values based on the following rule with a positive learning rate ( $\alpha$ ) will converge to the unbiased estimates:

$$V(s) \leftarrow V(s) + \alpha(r + \gamma V(s') - V(s))$$

Unfortunately, with using a function approximation this update rule is not done precisely, instead the left-hand-side shifts closer to the right-hand-side, while implicitly corrupting values for other states, thus this update rule which is guaranteed to converge on the unbiased estimates is not performed and the guarantee does not hold.

---

<sup>1</sup>[https://en.wikipedia.org/wiki/Temporal\\_difference\\_learning#Mathematical\\_formulation](https://en.wikipedia.org/wiki/Temporal_difference_learning#Mathematical_formulation)

d) When using a neural network for supervised learning, performance of training is typically measured by computing a total error over the training set. When using the NN for online learning of the  $Q$ -function in robocode this is not possible since there is no *a-priori* training set to work with. Suggest how you might monitor learning performance of the neural net now. (3 pts)

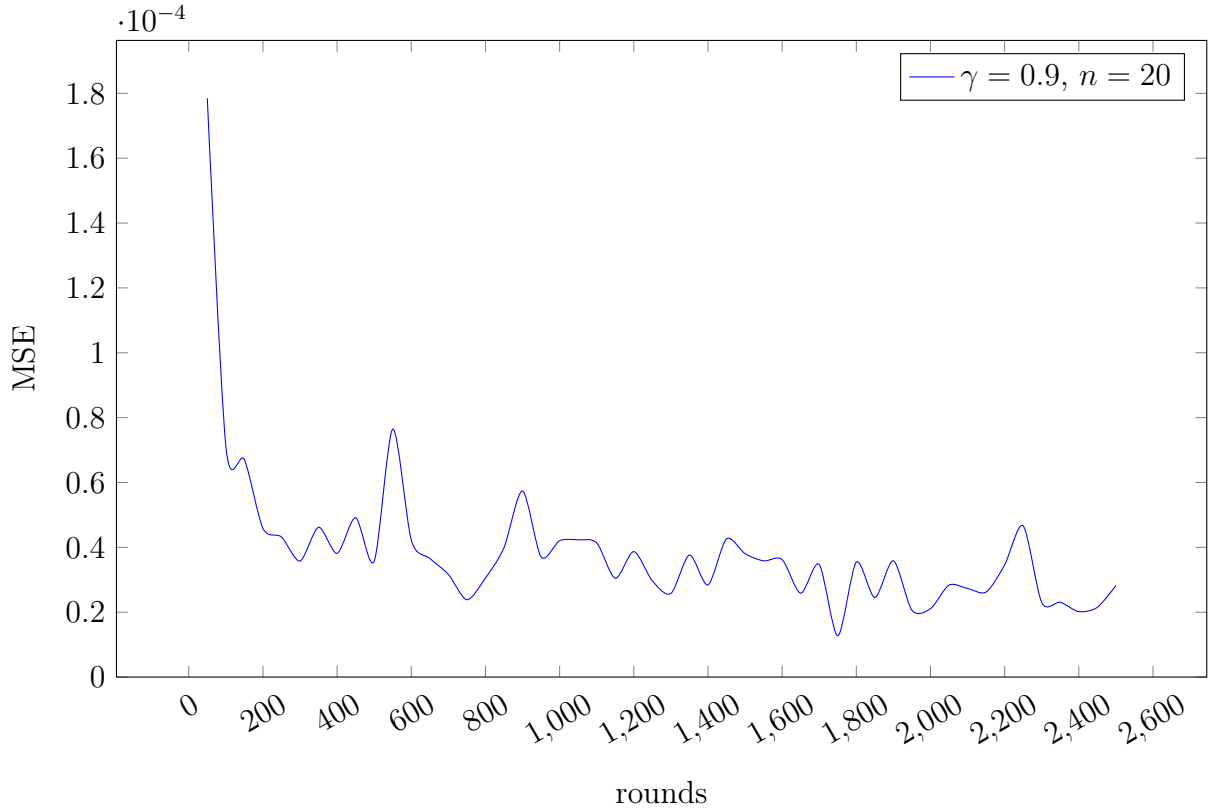


Figure 6: MSE over 20 subsequent states, with regard to rounds of training.

Inspired by the idea of replay memory, we use the loss over a local window of data points. Fig. 6 shows how the MSE as a metric for the performance of the neural network decreases after rounds of training. With the use of replay memory we are able to calculate this loss over 20 subsequent data points.

e) At each time step, the neural net in your robot performs a back propagation using a single training vector provided by the RL agent. Modify your code so that it keeps an array of the last say  $n$  training vectors and at each time step performs  $n$  back propagations. Using graphs compare the performance of your robot for different values of  $n$ . (4 pts)

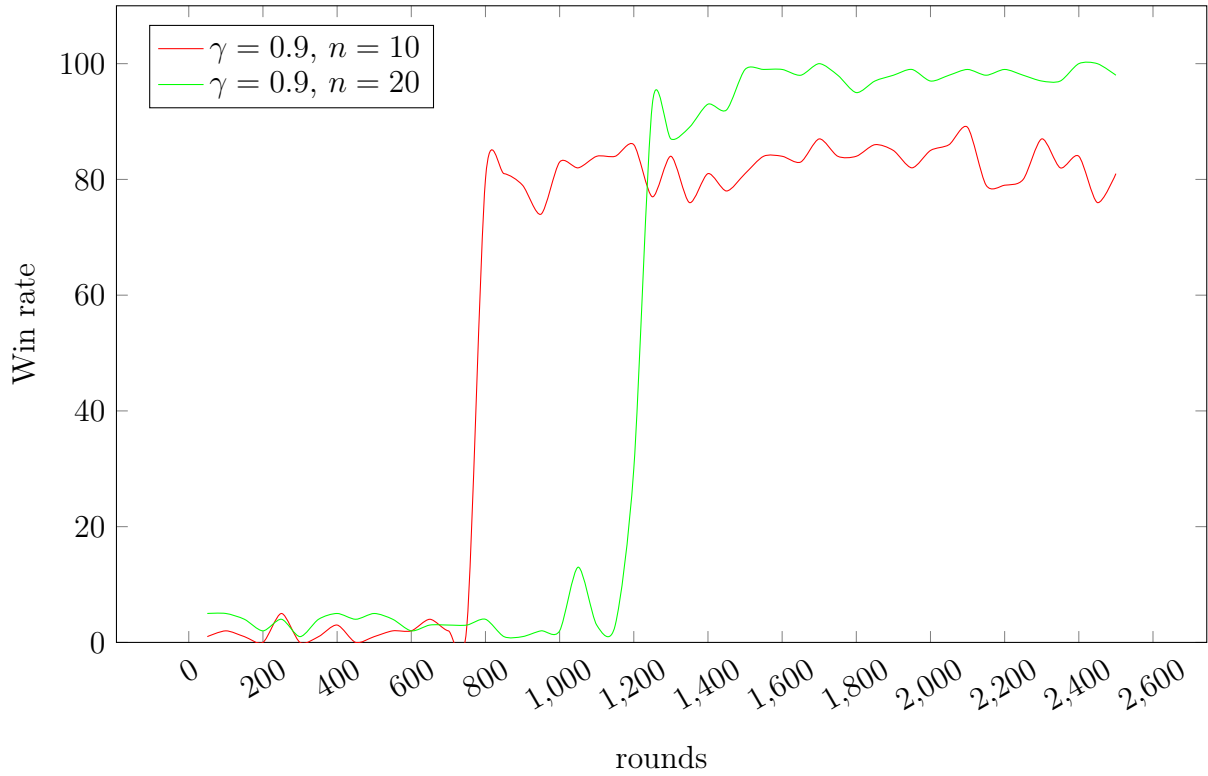


Figure 7: Win rate with regard to number of training rounds, under different  $\gamma$ .

Fig. 7 shows the different convergence behaviour with regard to different values for the window size of the replay memory. We observe that high value for the window size leads to a better win rate eventually. My hypothesis to explain this is that a larger local window allows the neural network to skip local optima and hopefully take better strategies.

## *(6) Overall Conclusions*

*a) This question is open-ended and offers you an opportunity to reflect on what you have learned overall through this project. For example, what insights are you able to offer with regard to the practical issues surrounding the application of RL & BP to your problem? E.g. What could you do to improve the performance of your robot? How would you suggest convergence problems be addressed? What advice would you give when applying RL with neural network based function approximation to other practical applications? (4 pts)*

Here are some insights we came up with:

1. More hidden neurons make the neural network to be able to learn more complex patterns. We were able to achieve less numerical error over our static lookup table data with more number of hidden neurons.
2. In a game where there are delays for the rewards to get effective (e.g. for firing to result in change in enemy energy) one trick to turn the environment back to a Markov environment is to treat series of subsequent states as some macro states which are path independent and actually form a Markov chain.
3. A lot of the time it is possible that large learning rates result in lack of convergence of the neural network. The search for a working and yet efficient learning rate has a rule of thumb of searching with different orders of magnitude and it is not required to search with linear steps.
4. To approach a complex implementation such as Q-learning with a deep neural network, it is always better to first break it into sub-problems. We have done this several times during the project. To test our Q-learning code we used a simpler game with the goal of navigating the robot the top right corner of the game only. To debug our neural network we started with a single layer wide neural network which was quite imitating a lookup table. To check if our neural network is capable of learning the game patterns we were instructed to first train the neural network over the static data from a lookup table.

*b) Theory question: Imagine a closed-loop control system for automatically delivering anesthetic to a patient under going surgery. You intend to train the controller using the approach used in this project. Discuss any concerns with this and identify one potential variation that could alleviate those concerns. (3 pts)*

The essential problem here is that there is no grant for killing the patient because of the RL algorithm performing trial and errors.

To fix this, I would suggest training the control system on a pseudo patient which only measures the dosage of anaesthetic over time.

To further make things safe, after the training is done we can try applying formal verification to ensure no harmful state is met while the execution of the trained agent.

We can also put safeguards in the controller, such as maximum limits of the dosage so that it does not pass the safe limits.

# Appendices

## A Source Codes

```
1 package autograd;
2
3 import jdk.jshell.spi.ExecutionControl;
4
5 public class Addition extends Operator {
6     @Override
7     public double evaluate(IVariable[] operands) {
8         double result = 0.;
9         for (IVariable operand :
10             operands) {
11             result += operand.evaluate();
12         }
13         return result;
14     }
15
16     @Override
17     public void backwards(IVariable[] operands, IVariable[] sources, double
18         gradient) throws ExecutionControl.NotImplementedException {
19         for (IVariable o :
20             operands) {
21             o.backward(sources, gradient);
22         }
23     }
24 }
```

Listing 1: autograd/Addition.java

```
1 package autograd;
2
3 import jdk.jshell.spi.ExecutionControl;
4
5 public class Exponentiation extends Operator {
6     @Override
7     public double evaluate(IVariable[] operands) {
8         if (operands.length != 2) {
9             throw new IllegalArgumentException("Exponentiation accepts 2
10                 arguments.");
11         }
12         return Math.pow(operands[0].evaluate(), operands[1].evaluate());
13     }
14
15     @Override
16     public void backwards(IVariable[] operands, IVariable[] sources, double
17         gradient) throws ExecutionControl.NotImplementedException {
18         IVariable baseVariable = operands[0];
19         var baseValue = baseVariable.evaluate();
20         IVariable exponentVariable = operands[1];
21         var exponentValue = exponentVariable.evaluate();
22         if (exponentVariable.getParameters().length > 1) {
23             throw new ExecutionControl.NotImplementedException("Back
24                 propagation to the exponent is not implemented.");
25         }
26     }
27 }
```

```

23         var gradientToPropagate = Math.pow(gradient * baseValue *
24             exponentValue, exponentValue - 1);
25         baseVariable.backward(sources, gradientToPropagate);
26     }

```

Listing 2: autograd/Exponentiation.java

```

1 package autograd;
2
3 public interface IInitializer {
4     double next();
5 }

```

Listing 3: autograd/IInitializer.java

```

1 package autograd;
2
3
4 import jdk.jshell.spi.ExecutionControl;
5
6 public interface IOperator {
7     IVariable apply(IVariable... operands);
8
9     double evaluate(IVariable[] operands);
10
11     void backwards(IVariable[] operands, IVariable[] sources, double
12         gradient) throws ExecutionControl.NotImplementedException;

```

Listing 4: autograd/IOperator.java

```

1 package autograd;
2
3 import jdk.jshell.spi.ExecutionControl;
4
5 public interface IVariable {
6     double evaluate();
7
8     void backward(IVariable[] sources, double gradient) throws
9         ExecutionControl.NotImplementedException;
10
11     Parameter[] getParameters();

```

Listing 5: autograd/IVariable.java

```

1 package autograd;
2
3 import jdk.jshell.spi.ExecutionControl;
4
5 public class Multiplication extends Operator {
6
7     @Override
8     public double evaluate(IVariable[] operands) {
9         double result = 1.;
10         for (IVariable operand :
11             operands) {

```

```

12         result *= operand.evaluate();
13     }
14     return result;
15 }
16
17 @Override
18 public void backwards(IVariable[] operands, IVariable[] sources, double
    gradient) throws ExecutionControl.NotImplementedException {
19     validateOperands(operands);
20     var multiplier = operands[0];
21     var multiplicand = operands[1];
22     var multiplierValue = multiplier.evaluate();
23     var multiplicandValue = multiplicand.evaluate();
24     multiplier.backward(sources, gradient * multiplicandValue);
25     multiplicand.backward(sources, gradient * multiplierValue);
26 }
27 }

```

Listing 6: autograd/Multiplication.java

```

1 package autograd;
2
3 import jdk.jshell.spi.ExecutionControl;
4
5 public class Negation extends Operator {
6
7     public Negation() {
8         this.numberOfOperands = 1;
9     }
10
11     @Override
12     public double evaluate(IVariable[] operands) {
13         validateOperands(operands);
14         return -operands[0].evaluate();
15     }
16
17     @Override
18     public void backwards(IVariable[] operands, IVariable[] sources, double
        gradient) throws ExecutionControl.NotImplementedException {
19         operands[0].backward(sources, -gradient);
20     }
21 }

```

Listing 7: autograd/Negation.java

```

1 package autograd;
2
3 import jdk.jshell.spi.ExecutionControl;
4
5 import java.util.Arrays;
6 import java.util.HashSet;
7
8 public class Operation implements IVariable {
9     private final IOperator operator;
10    private final IVariable[] operands;
11
12    public Operation(IOperator operator, IVariable... operands) {
13        this.operator = operator;

```



```

14         this.operands = operands;
15     }
16
17     @Override
18     public double evaluate() {
19         return operator.evaluate(operands);
20     }
21
22     @Override
23     public void backward(IVariable[] sources, double gradient) throws
        ExecutionControl.NotImplementedException {
24         operator.backwards(operands, sources, gradient);
25     }
26
27
28     @Override
29     public Parameter[] getParameters() {
30         HashSet<Parameter> result = new HashSet<>();
31         for (IVariable o :
32             this.operands) {
33             result.addAll(Arrays.asList(o.getParameters()));
34         }
35         return result.toArray(new Parameter[0]);
36     }
37
38     public IVariable[] getOperands() {
39         return operands;
40     }
41 }

```

Listing 8: autograd/Operation.java

```

1 package autograd;
2
3 public abstract class Operator implements IOperator {
4     protected Integer numberOfOperands;
5
6     public Operator() {
7         this.numberOfOperands = null;
8     }
9
10    @Override
11    public IVariable apply(IVariable... operands) {
12        return new Operation(this, operands);
13    }
14
15    protected void validateOperands(IVariable[] operands) {
16        if (this.numberOfOperands == null) {
17            return;
18        }
19        if (operands.length != this.numberOfOperands) {
20            throw new IllegalArgumentException(String.format("%s accepts
21                only one operand.", this.getClass().getName()));
22        }
23    }

```

Listing 9: autograd/Operator.java

```

1 package autograd;
2
3 import java.io.Serializable;
4 import java.util.Arrays;
5
6 public class Parameter implements IVariable, Serializable {
7     private double value;
8     private double gradient;
9     private boolean trainable;
10    private int layer;
11    private final int parameterId;
12    static transient int parameterCounter = 0;
13
14    public Parameter() {
15        parameterId = parameterCounter++;
16    }
17
18    public Parameter(double value) {
19        this.value = value;
20        trainable = true;
21        parameterId = parameterCounter++;
22    }
23
24    public Parameter(double value, boolean trainable) {
25        this.value = value;
26        this.trainable = trainable;
27        parameterId = parameterCounter++;
28    }
29
30    public static IVariable[] createTensor(double[] desired) {
31        var result = new Parameter[desired.length];
32        for (int i = 0; i < result.length; i++) {
33            result[i] = new Parameter(desired[i]);
34        }
35        return result;
36    }
37
38    @Override
39    public double evaluate() {
40        return value;
41    }
42
43    @Override
44    public void backward(IVariable[] sources, double gradient) {
45        if (Arrays.stream(sources).anyMatch(x -> x == this)) {
46            setGradient(gradient + getGradient());
47        }
48    }
49
50    @Override
51    public Parameter[] getParameters() {
52        return new Parameter[]{ this };
53    }
54
55    public double getValue() {
56        return this.value;
57    }

```

```

58
59     public void setValue(double value) {
60         this.value = value;
61     }
62
63     public double getGradient() {
64         return gradient;
65     }
66
67     private void setGradient(double gradient) {
68         this.gradient = gradient;
69     }
70
71     public boolean isTrainable() {
72         return this.trainable;
73     }
74
75     public void zeroGradient() {
76         this.setGradient(0);
77     }
78
79     public int getLayer() {
80         return layer;
81     }
82
83     public void setLayer(int layer) {
84         this.layer = layer;
85     }
86
87     public int getParameterId() {
88         return parameterId;
89     }
90 }

```

Listing 10: autograd/Parameter.java

```

1 package autograd;
2
3 import jdk.jshell.spi.ExecutionControl;
4
5 public class ReLU extends Operator {
6
7     public ReLU() {
8         this.numberOfOperands = 1;
9     }
10
11     @Override
12     public double evaluate(IVariable[] operands) {
13         if (operands.length != 1) {
14             throw new IllegalArgumentException("Sigmoid operator only
15                 accepts one operand");
16         }
17         double result = Math.max(0., operands[0].evaluate());
18         // System.out.println("ReLU " + result);
19         return result;
20     }
21
22     @Override

```

```

22     public void backwards(IVariable[] operands, IVariable[] sources, double
        gradient) throws ExecutionControl.NotImplementedException {
23         validateOperands(operands);
24         var x = operands[0];
25         var y = evaluate(operands);
26         //      System.out.println(gradient);
27         if (y > 0) {
28             //      System.out.println("Gradient " + y + " " + gradient);
29             x.backward(sources, gradient);
30         } else {
31             //      System.out.println("Gradient 0");
32             x.backward(sources, 0.);
33         }
34     }
35 }

```

Listing 11: autograd/ReLU.java

```

1 package autograd;
2
3 import jdk.jshell.spi.ExecutionControl;
4
5 public class Sigmoid extends Operator {
6
7     public Sigmoid() {
8         this.numberOfOperands = 1;
9     }
10
11     @Override
12     public double evaluate(IVariable[] operands) {
13         if (operands.length != 1) {
14             throw new IllegalArgumentException("Sigmoid operator only
                accepts one operand");
15         }
16         return 1. / (1 + Math.exp(-operands[0].evaluate()));
17     }
18
19     @Override
20     public void backwards(IVariable[] operands, IVariable[] sources, double
        gradient) throws ExecutionControl.NotImplementedException {
21         validateOperands(operands);
22         var x = operands[0];
23         var y = evaluate(operands);
24         x.backward(sources, gradient * y * (1 - y));
25     }
26 }

```

Listing 12: autograd/Sigmoid.java

```

1 package autograd;
2
3 import java.util.Random;
4
5 public class UniformInitializer implements IInitializer {
6
7     double a;
8     double b;
9     Random random;

```

```

10
11 public UniformInitializer(double a, double b) {
12     this.a = a;
13     this.b = b;
14     this.random = new Random();
15 }
16
17 @Override
18 public double next() {
19     return random.nextDouble() * (b - a) + a;
20 }
21 }

```

Listing 13: autograd/UniformInitializer.java

```

1 package dataset;
2
3 public class BinaryToBipolarWrapper implements IDataset {
4
5     IDataset binaryDataSet;
6
7     public BinaryToBipolarWrapper(IDataset binaryDataSet) {
8         this.binaryDataSet = binaryDataSet;
9     }
10
11     @Override
12     public DataPoint next() {
13         DataPoint result = binaryDataSet.next();
14         if (result == null) return null;
15         double[] x = result.getX().clone();
16         double[] y = result.getY().clone();
17         for (int i = 0; i < x.length; i++) {
18             x[i] = 2 * x[i] - 1;
19         }
20         for (int i = 0; i < y.length; i++) {
21             y[i] = 2 * y[i] - 1;
22         }
23         return new DataPoint(x, y);
24     }
25
26     @Override
27     public void reset() {
28         binaryDataSet.reset();
29     }
30
31     @Override
32     public DataPoint onlyReadNext() {
33         return binaryDataSet.onlyReadNext();
34     }
35 }

```

Listing 14: dataset/BinaryToBipolarWrapper.java

```

1 package dataset;
2
3 public class DataPointDataSet implements IDataset {
4     private final DataPoint dataPoint;
5     private boolean endOfDataSet = false;

```

```

6
7 public DataPointDataSet(DataPoint dataPoint) {
8     this.dataPoint = dataPoint;
9 }
10
11
12 @Override
13 public DataPoint next() {
14     if (endOfDataSet) {
15         return null;
16     } else {
17         endOfDataSet = true;
18         return dataPoint;
19     }
20 }
21
22 @Override
23 public void reset() {
24     endOfDataSet = false;
25 }
26
27 @Override
28 public DataPoint onlyReadNext() {
29     if (endOfDataSet) {
30         return null;
31     } else {
32         return dataPoint;
33     }
34 }
35 }

```

Listing 15: dataset/DataPointDataSet.java

```

1 package dataset;
2
3 public class DataPoint {
4     private final double[] x;
5     private final double[] y;
6
7     public DataPoint(double[] x, double[] y) {
8         this.x = x;
9         this.y = y;
10    }
11
12    public double[] getY() {
13        return y;
14    }
15
16    public double[] getX() {
17        return x;
18    }
19 }

```

Listing 16: dataset/DataPoint.java

```

1 package dataset;
2
3 public interface IDataset {

```

```

4      DataPoint next();
5
6      void reset();
7
8      DataPoint onlyReadNext();
9  }

```

Listing 17: dataset/IDataset.java

```

1  package dataset;
2
3  import fa.LUT;
4  import representation.IRepresentable;
5
6  import javax.naming.InsufficientResourcesException;
7  import javax.xml.crypto.Data;
8  import java.util.ArrayList;
9  import java.util.HashMap;
10
11 // TODO Christina and Husna
12 public class LookupTableDataSet implements IDataset {
13
14     ArrayList<DataPoint> points = new ArrayList<>();
15     int index = 0;
16
17     public LookupTableDataSet(LUT lut) {
18         HashMap map = lut.getHashMap();
19         for (Object key :
20             map.keySet()) {
21             IRepresentable representable = (IRepresentable) key;
22             points.add(new DataPoint(
23                 representable.toVector(),
24                 lut.eval(representable)));
25         }
26     }
27
28
29
30     @Override
31     public DataPoint next() {
32         if (index < points.size())
33             return points.get(index++);
34         return null;
35     }
36
37     @Override
38     public void reset() {
39         index = 0;
40     }
41
42     @Override
43     public DataPoint onlyReadNext() {
44         if (index < points.size())
45             return points.get(index);
46         return null;
47     }
48 }

```

Listing 18: dataset/LookupTableDataSet.java

```

1 package dataset;
2
3 import representation.IRepresentable;
4
5 import java.awt.geom.Line2D;
6 import java.io.Serializable;
7 import java.lang.reflect.Array;
8 import java.util.ArrayList;
9 import java.util.LinkedList;
10
11 public class RobotDataSet implements IDataset, Serializable {
12     LinkedList<double[]> x = new LinkedList<>();
13     LinkedList<double[]> y = new LinkedList<>();
14     int index = 0;
15     int offset = 0;
16     int windowSize;
17     public RobotDataSet(int windowSize) {
18         this.windowSize = windowSize;
19     }
20
21     public void addPattern(double[] input, double[] output) {
22         x.add(input);
23         y.add(output);
24         if (x.size() > windowSize) {
25             offset = 0;
26             x.removeFirst();
27             y.removeFirst();
28         }
29     }
30
31     @Override
32     public DataPoint next() {
33         if (index < x.size()) {
34             var result = new DataPoint(x.get(index), y.get(index));
35             index++;
36             return result;
37         }
38         return null;
39     }
40
41     @Override
42     public void reset() {
43         index = offset;
44     }
45
46     @Override
47     public DataPoint onlyReadNext() {
48         if (index < x.size()) {
49             return new DataPoint(x.get(index), y.get(index));
50         }
51         return null;
52     }
53
54     public int getSize() {
55         if (x.size() < windowSize) return x.size();
56     }
57

```



```

58     return windowSize;
59 }
60
61 public LinkedList<double[]> getX() {
62     return x;
63 }
64
65 public LinkedList<double[]> getY() {
66     return y;
67 }
68 }

```

Listing 19: dataset/RobotDataSet.java

```

1 package dataset;
2
3 public class XORBinaryDataSet implements IDataset {
4
5     protected double[][] x;
6     protected double[] y;
7     private int index;
8
9     public XORBinaryDataSet() {
10         index = 0;
11         x = new double[][] {
12             {0., 0.},
13             {0., 1.},
14             {1., 0.},
15             {1., 1.},
16         };
17         y = new double[] {
18             0.,
19             1.,
20             1.,
21             0.,
22         };
23     }
24
25     @Override
26     public DataPoint next() {
27         if (index < x.length) {
28             var result = new DataPoint(x[index], new double[] {y[index]});
29             index++;
30             return result;
31         }
32         return null;
33     }
34
35     @Override
36     public void reset() {
37         index = 0;
38     }
39
40     @Override
41     public DataPoint onlyReadNext() {
42         if (index < x.length) {
43             return new DataPoint(x[index], new double[] {y[index]});
44         }

```

```

45         return null;    }
46     }

```

Listing 20: dataset/XORBinaryDataSet.java

```

1 package fa;
2
3 import representation.IRepresentable;
4
5 import java.io.FileNotFoundException;
6 import java.io.IOException;
7
8 public interface IFunctionApproximation {
9     void train(IRepresentable input, double[] output);
10    double[] eval(IRepresentable input);
11
12    void save() throws IOException;
13
14    void load() throws IOException, ClassNotFoundException;
15
16    int getSize();
17 }

```

Listing 21: fa/IFunctionApproximation.java

```

1 package fa;
2
3 import representation.IRepresentable;
4
5 import java.io.*;
6 import java.util.HashMap;
7
8 public class LUT implements IFunctionApproximation {
9
10    private final String filePath;
11    //    int distance_level = 3;
12    //    int robot_energy_level = 3;
13    //    int enemy_energy_level = 3;
14    //    int position_level = 48;
15    private HashMap StateMap = new HashMap();
16    boolean readOnly;
17
18
19    public LUT(String filePath, boolean readOnly) {
20        this.filePath = filePath;
21        this.readOnly = readOnly;
22    }
23
24    public void save(File argFile) {
25
26    }
27
28
29    public void load(String argFileName) throws IOException {
30
31    }
32
33    //LEFT HERE: finish the two methods below for implementation tomorrow
34    morning.

```

```

34 // for each unique state vector, generate its key and match it with a
    state object that contains 5 actions
35
36 // public void initialiseLUT() {
37 //     for (int distance = 0; distance < distance_level; distance++) {
38 //         for (int robot_energy = 0; robot_energy < robot_energy_level;
    robot_energy++) {
39 //             for (int enemy_energy = 0; enemy_energy <
    enemy_energy_level; enemy_energy++) {
40 //                 for (int position = 0; position < position_level;
    position++) {
41 //                     //double[] state_vector = {distance, robot_energy,
    enemy_energy, position};
42 //                     //double key = indexFor(state_vector);
43 //                     State newState = new State(distance, robot_energy
    , enemy_energy, position);
44 //                     // Q values are automatically set to 0 by default
45 //                     newState.addAll(); // add all actions for each
    state?
46
47 //                     StateMap.put(newState, newState);
48 //
49 //                 }
50 //             }
51 //         }
52 //     }
53 // }
54
55
56
57 public double train(double[] X, double argValue) {
58
59
60
61
62     return 0;
63 }
64
65 @Override
66 public void train(IRepresentable input, double[] output) {
67     if (readOnly) {return;}
68     double[] repr = input.toVector();
69     System.out.print("train " + input + " to " + output[0]);
70     System.out.println();
71     this.StateMap.put(input, output);
72 }
73
74 @Override
75 public double[] eval(IRepresentable input) {
76     return (double[]) StateMap.getDefault(input, new double[]{ 0 });
77 }
78
79 @Override
80 public void save() throws IOException {
81     if (readOnly) return;
82     new ObjectOutputStream(new FileOutputStream(this.filePath)).
        writeObject(StateMap);
83 }

```

```

84
85  @Override
86  public void load() throws IOException, ClassNotFoundException {
87      this.StateMap = (HashMap) new ObjectInputStream(new FileInputStream
88          (this.filePath)).readObject();
89  }
90
91  public int getSize() {
92      return this.StateMap.size();
93  }
94
95  public HashMap getHashMap() {
96      return this.StateMap;
97  }

```

Listing 22: fa/LUT.java

```

1  package fa;
2
3  import autograd.IInitializer;
4  import autograd.Parameter;
5  import autograd.UniformInitializer;
6  import dataset.DataPoint;
7  import dataset.IDataset;
8  import dataset.RobotDataSet;
9  import jdk.jshell.spi.ExecutionControl;
10 import nn.*;
11 import optimization.GradientDescent;
12 import optimization.ILoss;
13 import optimization.IOptimizer;
14 import representation.IRepresentable;
15
16 import java.io.*;
17
18 // TODO Christina and Husna
19 public class NN implements IFunctionApproximation {
20     Model model = Factory.createNeuralNetwork(
21         new int[]{12, 15, 1},
22         new BipolarSigmoid(),
23         new UniformInitializer(-1, 1),
24         // new UniformInitializer(0, 0),
25         true,
26         true
27     );
28     IOptimizer optimizer = new GradientDescent(0.0001, 0.9);
29     ILayer activation;
30     IInitializer initializer;
31     int windowSize;
32     RobotDataSet dataSet;
33     ILoss loss = new MeanSquaredError(model.getOutput());
34     int epochs = 1;
35     double lossLimit = 0.000001;
36     IFitCallback collector = new ConvergenceCollector();
37
38     private final String filePath;
39     boolean readOnly;
40

```

```

41 public NN(String filePath, boolean readOnly) {
42     // construct the neural network
43     this.filePath = filePath;
44     this.readOnly = readOnly;
45     this.windowSize = 1;
46 }
47
48 public NN(String filePath, boolean readOnly, int windowSize) {
49     // construct the neural network
50     this.filePath = filePath;
51     this.readOnly = readOnly;
52     this.windowSize = windowSize;
53 }
54 @Override
55 public void train(IRepresentable input, double[] output) {
56     if (readOnly) {return;}
57     System.out.print("train " + input + " to " + output[0]);
58     System.out.println();
59     // construct a single datapoint dataset out of the data point
60     dataSet.addPattern(toInternalRepresentation(input.toVector()),
61         output);
62     System.out.println("SIZE: " + dataSet.getSize());
63     // call fit on the neural network
64     try {
65         dataSet.reset();
66         DataPoint next = dataSet.next();
67         System.out.println("Desired: " + next.getY()[0]);
68         System.out.println("Before fit: " + model.evaluate(next.getX())
69             [0]);
70         var totalLoss = model.fit(dataSet, optimizer, loss, epochs,
71             lossLimit);
72         System.out.println("After fit: " + model.evaluate(next.getX())
73             [0]);
74         System.out.println("LOSS: " + Math.sqrt(totalLoss / dataSet.
75             getSize()));
76     } catch (ExecutionControl.NotImplementedException e) {
77         e.printStackTrace();
78     }
79     System.out.println("So many parameters " + model.
80         getTrainableParameters().length);
81     for (var p :
82         model.getTrainableParameters()) {
83         System.out.print(p.getValue() + " ");
84     }
85     System.out.println();
86 }
87
88 @Override
89 public double[] eval(IRepresentable input) {
90     // feed the input to the neural network and return the outcome as
91     // the q value
92     double[] input_vector = input.toVector();
93     return model.evaluate(toInternalRepresentation(input_vector));
94 }
95
96 private double[] toInternalRepresentation(double[] rawPattern) {
97     return rawPattern;
98 }
99 // double[] sixtyFourPattern = new double[64];

```

```

92 //         int sixtyFourIndex = 0;
93 //         for (int i = 0; i < 3; i++) {
94 //             for (int j = 0; j < 4; j++) {
95 //                 sixtyFourIndex += Math.pow(4, i) * j * rawPattern[i * 4 +
j];
96 //             }
97 //         }
98 //         for (int i = 0; i < 64; i++) {
99 //             if (sixtyFourIndex == i) {
100 //                 sixtyFourPattern[i] = 1;
101 //             } else {
102 //                 sixtyFourPattern[i] = 0;
103 //             }
104 //         }
105 //         return sixtyFourPattern;
106     }
107
108     @Override
109     public void save() throws IOException {
110         // save all the parameters of the neural network, the weights
111         if (readOnly) return;
112         ObjectOutputStream stream = new ObjectOutputStream(new
FileOutputStream(this.filePath));
113         stream.writeObject(dataSet);
114         for (Parameter param :
model.getTrainableParameters()) {
115             stream.writeObject(param);
116         }
117         // save the entire model, or just the weights?
118         // need to record the different versions of weight along the way?
119         stream.close();
120     }
121
122
123     @Override
124     public void load() throws IOException, ClassNotFoundException {
125         // load the weights of the neural network from the filePath
126
127         //         ObjectInputStream modelStream = new ObjectInputStream(new
FileInputStream(this.getClass().getClassLoader().getResource("TrainedNN.
obj").getPath()));
128         //         Parameter[] parameters = (Parameter[]) (modelStream.readObject());
129         //         Parameter[] trainables = model.getTrainableParameters();
130         //         for (int i = 0; i < parameters.length; i++) {
131         //             trainables[i].setValue(parameters[i].getValue());
132         //         }
133         //         modelStream.close();
134
135         dataSet = new RobotDataSet(windowSize);
136         ObjectInputStream stream = new ObjectInputStream(new
FileInputStream(this.filePath));
137         dataSet = (RobotDataSet) stream.readObject();
138         for (var param: model.getTrainableParameters()) {
139             param.setValue(((Parameter) stream.readObject()).getValue());
140         }
141         stream.close();
142     }
143
144     @Override

```

```

145     public int getSize() {
146         return dataSet.getSize();
147     }
148
149     public double getLoss() throws ExecutionControl.NotImplementedException
150     {
151         IDataset wrappedDataset = new IDataset() {
152             int index = 0;
153
154             @Override
155             public DataPoint next() {
156                 if (NN.this.dataSet.getX().size() <= index) {
157                     return null;
158                 }
159                 DataPoint dataPoint = new DataPoint(NN.this.dataSet.getX().
160                     get(index), NN.this.dataSet.getY().get(index));
161                 index++;
162                 return dataPoint;
163             }
164
165             @Override
166             public void reset() {
167                 index = 0;
168             }
169
170             @Override
171             public DataPoint onlyReadNext() {
172                 return null;
173             }
174         };
175         return model.fit(dataSet, new GradientDescent(0, 0), loss, 1, 0) /
176             this.dataSet.getSize();
177     }
178
179     public RobotDataSet getDataSet() {
180         return dataSet;
181     }
182 }

```

Listing 23: fa/NN.java

```

1 package fa;
2
3 import representation.IRepresentable;
4
5 import java.io.IOException;
6
7 public class>NNLUT implements IFunctionApproximation {
8
9    >NN nn = new>NN("NNTNinetyRobot.NN", false);
10   >LUT lut = new>LUT("NNTNinetyRobot.LUT", false);
11    public>NNLUT() {
12
13    }
14
15    @Override
16    public void train(IRepresentable input, double[] output) {
17        nn.train(input, output);

```

```

18         lut.train(input, output);
19     }
20
21     @Override
22     public double[] eval(IRepresentable input) {
23         double first = nn.eval(input)[0];
24         double second = lut.eval(input)[0];
25         System.out.println("NN " + first + " LUT " + second);
26         return new double[] {second};
27     }
28
29     @Override
30     public void save() throws IOException {
31         nn.save();
32         lut.save();
33     }
34
35     @Override
36     public void load() throws IOException, ClassNotFoundException {
37         nn.load();
38         lut.load();
39     }
40
41     @Override
42     public int getSize() {
43         return 0;
44     }
45 }

```

Listing 24: fa/NNLUT.java

```

1 package nn;
2
3 import autograd.IVariable;
4 import autograd.Parameter;
5
6 public class BipolarSigmoid implements ILayer {
7
8     Parameter scale;
9
10    public BipolarSigmoid(double scale) {
11        this.scale = new Parameter(scale, false);
12    }
13
14    public BipolarSigmoid() {
15        this.scale = new Parameter(1., false);
16    }
17
18    @Override
19    public IVariable[] apply(IVariable[] input) {
20        var sigmoid = new autograd.Sigmoid();
21        var scalar = new Parameter(2, false);
22        var constant = new Parameter(-1, false);
23        var addition = new autograd.Addition();
24        var multiplication = new autograd.Multiplication();
25        var result = new IVariable[input.length];
26        for (int i = 0; i < input.length; i++) {
27            result[i] = multiplication.apply(

```



```

28         addition.apply(
29             multiplication.apply(
30                 scalar,
31                 sigmoid.apply(input[i])),
32             constant
33         ),
34         this.scale
35     );
36 }
37 return result;
38 }
39 }

```

Listing 25: nn/BipolarSigmoid.java

```

1 package nn;
2
3 import java.util.ArrayList;
4
5 public class ConvergenceCollector implements IFitCallback {
6     ArrayList<Double> loss;
7
8     public ConvergenceCollector() {
9         this.loss = new ArrayList<>();
10    }
11
12    @Override
13    public void collect(int epoch, double loss) {
14        this.loss.add(loss);
15    }
16
17    public int getEpochs() {
18        return loss.size();
19    }
20
21    @Override
22    public String toString() {
23        StringBuilder sb = new StringBuilder();
24        for (int i = 0; i < loss.size(); i++) {
25            sb.append(+i + " " + loss.get(i) + "\n");
26        }
27        return sb.toString();
28    }
29 }

```

Listing 26: nn/ConvergenceCollector.java

```

1 package nn;
2
3 import autograd.IInitializer;
4 import autograd.IVariable;
5 import autograd.Parameter;
6 import autograd.UniformInitializer;
7
8 public class Factory {
9     public static Model createNeuralNetwork(int[] sizes, ILayer activation,
10        IInitializer initializer) {
11         return createNeuralNetwork(sizes, activation, initializer, true,
12            true);
13     }
14 }

```

```

11     }
12     public static Model createNeuralNetwork(int[] sizes, ILayer activation,
13         IInitializer initializer, boolean lastActivation) {
14         return createNeuralNetwork(sizes, activation, initializer,
15             lastActivation, true);
16     }
17
18     public static Model createNeuralNetwork(int[] sizes, ILayer activation,
19         IInitializer initializer, boolean lastActivation, boolean useBiases
20     ) {
21         if (sizes.length < 2) {
22             throw new IllegalArgumentException("Sizes must at least contain
23                 2 integers for the first and the second layer.");
24         }
25         var inputs = new Parameter[sizes[0]];
26         for (int i = 0; i < inputs.length; i++) {
27             inputs[i] = new Parameter(initializer.next());
28         }
29         IVariable[] lastLayerOutput = inputs;
30         for (int i = 1; i < sizes.length; i++) {
31             lastLayerOutput = new Linear(sizes[i - 1], sizes[i],
32                 initializer, useBiases).apply(lastLayerOutput);
33             if (i < sizes.length - 1 || lastActivation) {
34                 lastLayerOutput = activation.apply(lastLayerOutput);
35             }
36         }
37         return new Model(inputs, lastLayerOutput);
38     }
39
40     public static Model createNeuralNetwork(int[] sizes, ILayer activation,
41         boolean lastActivation) {
42         return createNeuralNetwork(sizes, activation, new
43             UniformInitializer(-0.5, 0.5), lastActivation, true);
44     }
45
46     public static Model createNeuralNetwork(int[] sizes, ILayer activation)
47     {
48         return createNeuralNetwork(sizes, activation, true);
49     }
50 }

```

Listing 27: nn/Factory.java

```

1 package nn;
2
3 public interface IFitCallback {
4     void collect(int epoch, double loss);
5 }

```

Listing 28: nn/IFitCallback.java

```

1 package nn;
2
3 import autograd.IVariable;
4
5 public interface ILayer {
6     IVariable[] apply(IVariable[] input);
7 }

```

Listing 29: nn/ILayer.java

```

1 package nn;
2
3 import autograd.*;
4
5 public class Linear implements ILayer {
6     private final IVariable[][] weight;
7     private final IVariable[] bias;
8
9     public Linear(int inFeatures, int outFeatures, IInitializer initializer
10         , boolean useBiases) {
11         this.weight = new Parameter[outFeatures][inFeatures];
12         this.bias = new Parameter[outFeatures];
13         for (int i = 0; i < outFeatures; i++) {
14             for (int j = 0; j < inFeatures; j++) {
15                 this.weight[i][j] = new Parameter(initializer.next());
16             }
17             if (useBiases)
18                 this.bias[i] = new Parameter(initializer.next());
19             else
20                 this.bias[i] = new Parameter(0., false);
21         }
22     }
23
24     @Override
25     public IVariable[] apply(IVariable[] input) {
26         var result = new IVariable[this.weight.length];
27         for (int i = 0; i < this.weight.length; i++) {
28             int inputSize = this.weight[i].length;
29             IVariable[] muls = new IVariable[inputSize + 1];
30             for (int j = 0; j < inputSize; j++) {
31                 muls[j] = new Multiplication().apply(this.weight[i][j],
32                     input[j]);
33             }
34             muls[inputSize] = this.bias[i];
35             result[i] = new Addition().apply(muls);
36         }
37         return result;
38     }
39
40     private int getWidth() {
41         return this.weight.length;
42     }
43 }

```

Listing 30: nn/Linear.java

```

1 package nn;
2
3 import autograd.*;
4 import jdk.jshell.spi.ExecutionControl;
5 import optimization.ILoss;
6
7 public class MeanSquaredError implements IVariable, ILoss {
8

```

```

9  private final IVariable operation;
10 private final Parameter[] desired;
11
12 public MeanSquaredError(IVariable[] output) {
13     var negation = new Negation();
14     var addition = new Addition();
15     var multiplication = new Multiplication();
16     var exponentiation = new Exponentiation();
17     Parameter two = new Parameter(2, false);
18     Parameter half = new Parameter(0.5, false);
19     int length = output.length;
20     desired = new Parameter[output.length];
21     var summationTerms = new IVariable[length];
22     for (int i = 0; i < length; i++) {
23         desired[i] = new Parameter();
24         summationTerms[i] = exponentiation.apply(
25             addition.apply(output[i], negation.apply(desired[i])),
26             two
27         );
28     }
29     this.operation = multiplication.apply(addition.apply(summationTerms
30         ), half);
31 }
32
33 @Override
34 public double evaluate() {
35     return operation.evaluate();
36 }
37
38 @Override
39 public void backward(IVariable[] sources, double gradient) throws
40     ExecutionControl.NotImplementedException {
41     operation.backward(sources, gradient);
42 }
43
44 @Override
45 public Parameter[] getParameters() {
46     return this.operation.getParameters();
47 }
48
49 @Override
50 public void setDesired(double[] desired) {
51     for (int i = 0; i < this.desired.length; i++) {
52         this.desired[i].setValue(desired[i]);
53     }
54 }
55 }

```

Listing 31: nn/MeanSquaredError.java

```

1 package nn;
2
3 import autograd.IVariable;
4 import autograd.Operation;
5 import autograd.Parameter;
6 import dataset.DataPoint;
7 import dataset.IDataset;
8 import jdk.jshell.spi.ExecutionControl;

```

```

9 import optimization.ILoss;
10 import optimization.IOptimizer;
11
12 import javax.swing.*;
13 import javax.xml.crypto.Data;
14 import java.io.Serializable;
15 import java.util.*;
16 import java.util.function.IntFunction;
17 import java.util.stream.Collectors;
18
19 public class Model {
20     private final Parameter[] input;
21     private final IVariable[] output;
22
23     public Model(Parameter[] input, IVariable[] output) {
24         this.input = input;
25         this.output = output;
26     }
27
28
29     public double[] evaluate(double[] input) {
30         var result = new double[output.length];
31         for (int i = 0; i < input.length; i++) {
32             this.input[i].setValue(input[i]);
33         }
34         for (int i = 0; i < output.length; i++) {
35             result[i] = output[i].evaluate();
36         }
37         return result;
38     }
39
40     public Parameter[] getParameters() {
41         HashSet<Parameter> result = new HashSet<>();
42         for (IVariable o :
43             this.output) {
44             result.addAll(Arrays.asList(o.getParameters()));
45         }
46         return result.toArray(new Parameter[0]);
47     }
48
49     public Parameter[] getTrainableParameters() {
50         var results = new HashSet<Parameter>();
51         for (Parameter p :
52             getParameters()) {
53             if (p.isTrainable()) {
54                 results.add(p);
55             }
56         }
57         for (Parameter p : input) {
58             results.remove(p);
59         }
60
61         return results.stream().sorted(Comparator.comparing(Parameter::
62             getParameterId)).toArray(Parameter[]::new);
63     }
64
65     public IVariable[] getOutput() {
66         return output;
67     }

```

```

66     }
67
68     public double fit(IDataSet dataSet, IOptimizer optimizer, ILoss loss,
69         int epochs, double lossLimit) throws ExecutionControl.
70         NotImplementedException {
71         return fit(dataSet, optimizer, loss, epochs, lossLimit, (epoch, 1)
72             -> {
73             });
74     }
75
76     public double fit(IDataSet dataSet, IOptimizer optimizer, ILoss loss,
77         int epochs, double lossLimit, IFitCallback callback) throws
78         ExecutionControl.NotImplementedException {
79         return fit(dataSet, optimizer, loss, epochs, lossLimit, callback,
80             true);
81     }
82
83     public double fit(IDataSet dataSet, IOptimizer optimizer, ILoss loss,
84         int epochs, double lossLimit, IFitCallback callback, boolean online)
85         throws ExecutionControl.NotImplementedException {
86         var parameters = getTrainableParameters();
87         Map<Integer, List<Parameter>> layeredParameters = layerParameters(
88             parameters);
89         if (epochs < 1) {
90             throw new IllegalArgumentException("At least one epochs
91                 required.");
92         }
93         double totalLoss = 0;
94         for (int i = 0; i < epochs; i++) {
95             totalLoss = 0;
96             dataSet.reset();
97             DataPoint dataPoint;
98             while ((dataPoint = dataSet.next()) != null) {
99                 setInput(dataPoint.getX());
100                 loss.setDesired(dataPoint.getY());
101                 totalLoss += loss.evaluate();
102                 if (online) {
103                     for (Integer j : layeredParameters.keySet().stream().
104                         sorted().collect(Collectors.toList())) {
105                         Parameter[] layerParameters = layeredParameters.get
106                             (j).toArray(new Parameter[0]);
107                         loss.backward(layerParameters, 1.);
108                         optimizer.update(layerParameters);
109                     }
110                 } else {
111                     loss.backward(parameters, 1.);
112                 }
113             }
114             callback.collect(i, totalLoss);
115             if (totalLoss < lossLimit) {
116                 break;
117             }
118             if (!online) {
119                 optimizer.update(parameters);
120             }
121         }
122         return totalLoss;
123     }
124 }

```

```

112 private Map<Integer, List<Parameter>> layerParameters(Parameter[]
113 parameters) {
114     setLayers(getOutput(), 0);
115     return Arrays.stream(parameters).collect(Collectors.groupingBy(
116         Parameter::getLayer));
117 }
118 private void setLayers(IVariable[] outputs, int layer) {
119     if (outputs.length == 0) return;
120     HashSet<IVariable> nextOutput = new HashSet<>();
121     for (IVariable i : outputs) {
122         if (i instanceof Parameter) {
123             ((Parameter) i).setLayer(layer);
124         }
125         if (i instanceof Operation) {
126             nextOutput.addAll(Arrays.asList(((Operation) i).getOperands
127                 ()));
128         }
129     }
130     setLayers(nextOutput.toArray(new IVariable[0]), layer + 1);
131 }
132 private void setInput(double[] x) {
133     for (int i = 0; i < input.length; i++) {
134         input[i].setValue(x[i]);
135     }
136 }
137 }

```

Listing 32: nn/Model.java

```

1 package nn;
2
3 import autograd.IVariable;
4
5 public class ReLU implements ILayer {
6
7     @Override
8     public IVariable[] apply(IVariable[] input) {
9         var operator = new autograd.ReLU();
10        var result = new IVariable[input.length];
11        for (int i = 0; i < input.length; i++) {
12            result[i] = operator.apply(input[i]);
13        }
14        return result;
15    }
16 }

```

Listing 33: nn/ReLU.java

```

1 package nn;
2
3 import autograd.IVariable;
4
5 public class Sigmoid implements ILayer {
6
7     @Override

```

```

8      public IVariable[] apply(IVariable[] input) {
9          var operator = new autograd.Sigmoid();
10         var result = new IVariable[input.length];
11         for (int i = 0; i < input.length; i++) {
12             result[i] = operator.apply(input[i]);
13         }
14         return result;
15     }
16 }

```

Listing 34: nn/Sigmoid.java

```

1 package optimization;
2
3 import autograd.Parameter;
4
5 import java.util.HashMap;
6
7 public class GradientDescent implements IOptimizer {
8
9     private final HashMap<Parameter, Double> lastDelta;
10    private final double learningRate;
11    private final double momentum;
12
13    public GradientDescent(double learningRate, double momentum) {
14        this.lastDelta = new HashMap<>();
15        this.learningRate = learningRate;
16        this.momentum = momentum;
17    }
18
19    @Override
20    public void update(Parameter[] parameters) {
21        for (Parameter p :
22             parameters) {
23            double delta = -p.getGradient() * learningRate + momentum *
24                           lastDelta.getOrDefault(p, 0.);
25            // System.out.println(p.getValue() + " " + delta);
26            p.setValue(p.getValue() + delta);
27            p.zeroGradient();
28            lastDelta.put(p, delta);
29        }
30    }
31 }

```

Listing 35: optimization/GradientDescent.java

```

1 package optimization;
2
3 import autograd.IVariable;
4
5 public interface ILoss extends IVariable {
6     void setDesired(double[] desired);
7 }

```

Listing 36: optimization/ILoss.java

```

1 package optimization;
2

```



```

3 import autograd.Parameter;
4
5 public interface IOptimizer {
6     void update(Parameter[] parameters);
7 }

```

Listing 37: optimization/IOptimizer.java

```

1 package policy;
2
3 import representation.IState;
4 import representation.States;
5
6 public class EnergyReward implements IPolicy {
7     @Override
8     public double getReward(IState run, IState last) {
9         // should we give rewards on one state or on the change of last two
10         // states?
11         States states = (States) run;
12         States lastStates = (States) last;
13         // return states.getMyEnergy() - states.getEnemyEnergy();
14         return (lastStates.getEnemyEnergy() - states.getEnemyEnergy()) /
15             100.;
16     }
17 }

```

Listing 38: policy/EnergyReward.java

```

1 package policy;
2
3 import representation.IState;
4 import representation.States;
5
6 public class EnergyRewardTerminal implements IPolicy {
7     @Override
8     public double getReward(IState run, IState last) {
9         // should we give rewards on one state or on the change of last two
10        // states?
11        States states = (States) run;
12        States lastStates = (States) last;
13        // return states.getMyEnergy() - states.getEnemyEnergy();
14
15        if (states.getEnemyEnergy() == 0)
16            return 1;
17        if (states.getMyEnergy() == 0)
18            return -1;
19        return 0;
20    }
21 }

```

Listing 39: policy/EnergyRewardTerminal.java

```

1 package policy;
2
3 import representation.Coordinates;
4 import representation.IState;
5

```

```

6 public class GoTopRight implements IPolicy {
7     @Override
8     public double getReward(IState run, IState dummy) {
9         Coordinates coordinates = (Coordinates) run;
10        var x = coordinates.getX();
11        var y = coordinates.getY();
12        if (x == 7 && y == 5) {
13            return 1;
14        }
15        return 0;
16    }
17 }

```

Listing 40: policy/GoTopRight.java

```

1 package policy;
2
3 import representation.IState;
4
5 public interface IPolicy {
6     double getReward(IState currentState, IState lastState);
7 }

```

Listing 41: policy/IPolicy.java

```

1 package representation;
2
3 public class Action {
4
5     enum ActionName {FIRE,RIGHT,LEFT, AHEAD, BACK};
6     // want an instance variable "action"
7     ActionName action;
8     double QValue = 0;
9
10    public Action (String name) {
11
12        switch (name) {
13            case "fire":
14                action = ActionName.FIRE;
15                break;
16            case "right":
17                action = ActionName.RIGHT;
18                break;
19            case "left":
20                action = ActionName.LEFT;
21                break;
22            case "ahead":
23                action = ActionName.AHEAD;
24                break;
25            case "back":
26                action = ActionName.BACK;
27                break;
28        }
29    }
30    public static void main(String[] args) {
31        Action action = new Action("right");
32        System.out.println(action.action);
33    }

```

34  
35  
36 }

Listing 42: representation/Action.java

```
1 package representation;
2
3 import java.io.Serializable;
4
5 public class Concatenation implements IRepresentable, Serializable {
6     private IRepresentable first;
7     private IRepresentable second;
8
9     public Concatenation(IRepresentable state, IRepresentable action) {
10         this.first = state;
11         this.second = action;
12     }
13     @Override
14     public double[] toVector() {
15         double[] stateVector = first.toVector();
16         double[] actionVector = second.toVector();
17         double[] result = new double[stateVector.length + actionVector.
18             length];
19         System.arraycopy(stateVector, 0, result, 0, stateVector.length);
20         System.arraycopy(actionVector, 0, result, stateVector.length,
21             actionVector.length);
22         return result;
23     }
24
25     @Override
26     public int hashCode() {
27         return first.hashCode() + second.hashCode();
28     }
29
30     @Override
31     public boolean equals(Object obj) {
32         if (!(obj instanceof IRepresentable)) return false;
33         var testRepr = ((IRepresentable) obj).toVector();
34         double[] result = toVector();
35         if (testRepr.length != result.length) return false;
36         for (int i = 0; i < result.length; i++) {
37             if (result[i] != testRepr[i]) return false;
38         }
39         return true;
40     }
41
42     @Override
43     public String toString() {
44         return "Concatenation{" +
45             "first=" + first +
46             ", second=" + second +
47             '}';
48     }
49 }
```

Listing 43: representation/Concatenation.java

```

1 package representation;
2
3 public class ConcatenationRepresentation implements
4     IStateActionRepresentation {
5
6
7     @Override
8     public IRepresentable represent(IState state, IAction action) {
9         return new Concatenation(state, action);
10    }
11 }

```

Listing 44: representation/ConcatenationRepresentation.java

```

1 package representation;
2
3 import java.io.Serializable;
4 import java.util.Objects;
5
6 public class Coordinates implements IState, Serializable {
7     private int x;
8     private int y;
9     private int heading;
10
11     public Coordinates(int x, int y, int heading) {
12         setX(x);
13         setY(y);
14         setHeading(heading);
15     }
16
17     public void setX(int x) {
18         this.x = x;
19     }
20
21     public void setY(int y) {
22         this.y = y;
23     }
24
25     public void setHeading(int bearing) {
26         this.heading = bearing;
27     }
28
29     @Override
30     public IState clone() {
31         return new Coordinates(this.x, this.y, this.heading);
32     }
33
34     public int getX() {
35         return this.x;
36     }
37
38     @Override
39     public double[] toVector() {
40         return new double[] {x, y, heading};
41     }
42
43     @Override

```

```

44 public boolean equals(Object o) {
45     if (this == o) return true;
46     if (o == null || getClass() != o.getClass()) return false;
47     Coordinates that = (Coordinates) o;
48     return x == that.x && y == that.y && heading == that.heading;
49 }
50
51 @Override
52 public int hashCode() {
53     return Objects.hash(x, y, heading);
54 }
55
56 public int getY() {
57     return y;
58 }
59
60 @Override
61 public String toString() {
62     return "Coordinates{" +
63         "x=" + x +
64         ", y=" + y +
65         ", heading=" + heading +
66         '}';
67 }
68 }

```

Listing 45: representation/Coordinates.java

```

1 package representation;
2
3 import robocode.Event;
4 import robocode.ScannedRobotEvent;
5 import robocode.StatusEvent;
6
7 public class CoordinatesRepresentation implements IStateRepresentation {
8     @Override
9     public IState represent(IState state, Event event) {
10         if (state == null) {
11             state = new Coordinates(0, 0, 0);
12         }
13         Coordinates coordinates = (Coordinates) state.clone();
14         if (event instanceof StatusEvent) {
15             StatusEvent statusEvent = (StatusEvent) event;
16             coordinates.setX((int) (statusEvent.getStatus().getX() / 100));
17             coordinates.setY((int) (statusEvent.getStatus().getY() / 100));
18             coordinates.setHeading((int) ((statusEvent.getStatus().
19                 getHeading() + 45) / 90));
20         }
21         return coordinates;
22     }
23 }

```

Listing 46: representation/CoordinatesRepresentation.java

```

1 package representation;
2
3 public interface IAction extends IRepresentable {
4

```

```
5 }
```

Listing 47: representation/IAction.java

```
1 package representation;
2
3 import robocode.Robot;
4
5 public interface IActionRepresentation extends IRepresentation {
6     void takeAction(Robot robot, IAction action);
7
8     IAction [] getActions();
9 }
```

Listing 48: representation/IActionRepresentation.java

```
1 package representation;
2
3 import java.io.Serializable;
4
5 public interface IRepresentable extends Serializable {
6     double [] toVector();
7 }
```

Listing 49: representation/IRepresentable.java

```
1 package representation;
2
3 import robocode.Robot;
4 import robocode.Event;
5
6 public interface IRepresentation {
7 }
```

Listing 50: representation/IRepresentation.java

```
1 package representation;
2
3 public interface IStateActionRepresentation {
4     IRepresentable represent(IState state, IAction action);
5 }
```

Listing 51: representation/IStateActionRepresentation.java

```
1 package representation;
2
3 public interface IState extends IRepresentable {
4     public IState clone();
5 }
```

Listing 52: representation/IState.java

```
1 package representation;
2
3 import robocode.Event;
4
5 public interface IStateRepresentation extends IRepresentation {
6     /**
```

```

7      * Evolves the robot state given the last state and the event
8      * @param state the previous state of the robot
9      * @param event the robot event containing changes to the state
10     * @return returns a new state expressing the changed state
11     */
12     IState represent(IState state, Event event);
13 }

```

Listing 53: representation/IStateRepresentation.java

```

1 package representation;
2
3 import java.io.Serializable;
4 import java.util.Objects;
5
6 public class Move implements IAction, Serializable {
7     @Override
8     public double[] toVector() {
9         double value = 0;
10        switch (actionType) {
11            case AHEAD:
12                value = 1;
13                break;
14            case TURN_LEFT:
15                value = 2;
16                break;
17            case TURN_RIGHT:
18                value = 3;
19                break;
20            default:
21                assert false;
22        }
23        return new double[] { value };
24    }
25
26    @Override
27    public String toString() {
28        return "Move{" +
29            "actionType=" + actionType +
30            '}';
31    }
32
33    public enum ActionType {
34        TURN_RIGHT,
35        TURN_LEFT,
36        AHEAD,
37    }
38
39    ActionType actionType;
40
41    public Move(ActionType actionType) {
42        this.actionType = actionType;
43    }
44
45    public ActionType getActionType() {return actionType;}
46
47    @Override
48    public boolean equals(Object o) {

```

```

49         if (this == o) return true;
50         if (o == null || getClass() != o.getClass()) return false;
51         Move move = (Move) o;
52         return actionType == move.actionType;
53     }
54
55     @Override
56     public int hashCode() {
57         return Objects.hash(actionType);
58     }
59 }

```

Listing 54: representation/Move.java

```

1 package representation;
2
3 import robocode.Robot;
4
5 public class MoveRepresentation implements IActionRepresentation {
6     @Override
7     public void takeAction(Robot qLearningRobot, IAction action) {
8         if (action == null) return;
9         if (!(action instanceof Move)) {
10             throw new IllegalArgumentException("Move representation can
11                 only take move actions.");
12         }
13         Move move = (Move) action;
14         System.out.println("CASTED");
15         if (move.getActionType() == Move.ActionType.TURN_LEFT) {
16             qLearningRobot.turnLeft(90);
17         } else if (move.getActionType() == Move.ActionType.TURN_RIGHT) {
18             qLearningRobot.turnRight(90);
19         } else if (move.getActionType() == Move.ActionType.AHEAD) {
20             qLearningRobot.ahead(100);
21         }
22     }
23
24     @Override
25     public IAction[] getActions() {
26         return new IAction[] {
27             new Move(Move.ActionType.AHEAD),
28             new Move(Move.ActionType.TURN_LEFT),
29             new Move(Move.ActionType.TURN_RIGHT),
30         };
31     }
32 }

```

Listing 55: representation/MoveRepresentation.java

```

1 package representation;
2
3 public class Representation {
4
5 }

```

Listing 56: representation/Representation.java

```

1 package representation;

```



```

2
3 import java.util.ArrayList;
4 import java.util.List;
5 import java.util.Objects;
6
7 public class State {
8
9     // The states
10
11     // the relative distance to the enemy (<200, <300, >=300)
12     // our energy (<30, >=30, >100)
13     // the enemy's energy (<30, >=30, >100)
14     // x, y position of our own (step by 100)
15
16     // The actions
17
18     // fire(1)
19     // turn right (90)
20     // turn left (90)
21     // go ahead (100)
22     // go back (100)
23     // do nothing
24
25     private int distance;
26     private int energy;
27     private int enemyEnergy;
28     private int x;
29     private int y;
30     private int enemyBearing;
31
32     List<Action> actions = new ArrayList<Action>();
33
34     public void add(Action a) {
35         actions.add(a);
36     }
37
38     public void addAll() {
39         actions.add(new Action("fire"));
40         actions.add(new Action("right"));
41         actions.add(new Action("left"));
42         actions.add(new Action("ahead"));
43         actions.add(new Action("back"));
44     }
45
46     @Override
47     public boolean equals(Object o) {
48         if (this == o) return true;
49         if (o == null || getClass() != o.getClass()) return false;
50         State state = (State) o;
51         return distance == state.distance && energy == state.energy &&
52             enemyEnergy == state.enemyEnergy && x == state.x;
53     }
54
55     @Override
56     public int hashCode() {
57         return Objects.hash(distance, energy, enemyEnergy, x);
58     }

```

59  
60  
61  
62 }

Listing 57: representation/State.java

```
1 package representation;  
2 import robocode.*;  
3  
4 public class StateRep implements IStateRepresentation {  
5     public StateRep() {}  
6     @Override  
7     // represent method will be called under two circumstances: either from  
8     // onStatus or from onScannedRobot.  
9     // the event passed in can be of either StatusEvent or  
10    ScannedRobotEvent  
11    public IState represent(IState state, Event event) {  
12        //passed states are the last states, all null at first turn  
13        if (state == null) {  
14            state = new States(0, 0, 0, 0, 0, 0, 0, 0);  
15        }  
16        States states = (States) state.clone(); //cast State to States  
17  
18        if (event instanceof ScannedRobotEvent) {  
19            ScannedRobotEvent scannedEvent = (ScannedRobotEvent) event;  
20            states.setDistance((int) (scannedEvent.getDistance()));  
21            states.setEnemyEnergy((int) scannedEvent.getEnergy());  
22            states.setBearing((int) scannedEvent.getBearing());  
23        }  
24        if (event instanceof StatusEvent) {  
25            StatusEvent statusEvent = (StatusEvent) event;  
26            states.setX((int) statusEvent.getStatus().getX());  
27            states.setY((int) statusEvent.getStatus().getY());  
28            states.setHeading((int) statusEvent.getStatus().getHeading());  
29            states.setMyEnergy((int) statusEvent.getStatus().getEnergy());  
30        }  
31        if (event instanceof WinEvent) {  
32            states.setEnemyEnergy(0);  
33        }  
34        if (event instanceof DeathEvent) {  
35            states.setMyEnergy(0);  
36        }  
37        return states;  
38    }  
39 }
```

Listing 58: representation/StateRep.java

```
1 package representation;  
2  
3 import java.util.Arrays;  
4  
5 public class States implements IState{  
6  
7     private int distance;  
8     private int x;
```

```

9     private int y;
10    private int heading;
11    private int bearing;
12
13    public void setBearing(int bearing) {
14        this.bearing = bearing;
15    }
16
17    public int getBearing() {
18        return bearing;
19    }
20
21    public enum energy {LOW, MEDIUM, HIGH};
22    private int myEnergy;
23    private int enemyEnergy;
24
25    public States(int distance, int x, int y, int heading, int myEnergy,
26        int enemyEnergy, int bearing) {
27        setDistance(distance);
28        setX(x);
29        setY(y);
30        setHeading(heading);
31        setMyEnergy(myEnergy);
32        setEnemyEnergy(enemyEnergy);
33        setBearing(bearing);
34    }
35
36    public void setDistance(int distance) {
37        this.distance = distance;
38    }
39
40    public void setX(int x) {
41        this.x = x;
42    }
43
44    public void setY(int y) {this.y = y;}
45    public void setHeading(int heading) {
46        this.heading = heading;
47    }
48
49    public void setMyEnergy(int myEnergy) {
50        this.myEnergy = myEnergy;
51    }
52
53    public void setEnemyEnergy(int enemyEnergy) {
54        this.enemyEnergy = enemyEnergy;
55    }
56
57    @Override
58    public IState clone() {
59        return new States(this.distance, this.x, this.y, this.heading, this
60            .myEnergy, this.enemyEnergy, this.bearing);
61    }
62
63    @Override
64    public double[] toVector() {
65        return new double[]{
66            this.x / 200,
67            this.y / 200,
68            (this.heading + 45) / 90,
69            (this.bearing + 45) / 90,
70            this.myEnergy / 40,

```

```

65 //          this.enemyEnergy / 40,
66 //          this.distance / 200,
67     };
68 }
69
70 public int getDistance() {
71     return distance;
72 }
73
74 public int getX() {
75     return x;
76 }
77
78 public int getY() {
79     return y;
80 }
81
82 public int getHeading() {
83     return heading;
84 }
85
86 public int getMyEnergy() {
87     return myEnergy;
88 }
89
90 public int getEnemyEnergy() {
91     return enemyEnergy;
92 }
93
94 @Override
95 public boolean equals(Object o) {
96     if (this == o) return true;
97     if (o == null || getClass() != o.getClass()) return false;
98     States states = (States) o;
99     double[] mine = toVector();
100    double[] theirs = states.toVector();
101    for (int i = 0; i < mine.length; i++) {
102        if (mine[i] != theirs[i]) {
103            return false;
104        }
105    }
106    return true;
107 }
108
109 @Override
110 public int hashCode() {
111     return Arrays.hashCode(toVector());
112 }
113
114 @Override
115 public String toString() {
116     return "States{" +
117         "distance=" + distance +
118         ", x=" + x +
119         ", y=" + y +
120         ", heading=" + heading +
121         ", bearing=" + bearing +
122         ", myEnergy=" + myEnergy +

```

```

123         ", enemyEnergy=" + enemyEnergy +
124         '}', '};
125     }
126 }

```

Listing 59: representation/States.java

```

1 package representation;
2
3 import java.io.Serializable;
4 import java.util.Objects;
5
6 public class TNinetyAction implements IAction, Serializable {
7     @Override
8     public double[] toVector() {
9         var value = new double[] {0, 0, 0, 0};
10        switch (actionType) {
11            case AHEAD:
12                value = new double[] {1, 0, 0, 0};
13                break;
14            case TURN_LEFT:
15                value = new double[] {0, 1, 0, 0};
16                break;
17            case TURN_RIGHT:
18                value = new double[] {0, 0, 1, 0};
19                break;
20            case FIRE:
21                value = new double[] {0, 0, 0, 1};
22                break;
23            default:
24                assert false;
25        }
26        return value;
27    }
28
29    @Override
30    public String toString() {
31        return "Move{" +
32            "actionType=" + actionType +
33            '}', '};
34    }
35
36    public enum ActionType {
37        TURN_RIGHT,
38        TURN_LEFT,
39        AHEAD,
40        FIRE,
41        RANDOMLY_MOVE,
42    }
43
44    ActionType actionType;
45
46    public TNinetyAction(ActionType actionType) {
47        this.actionType = actionType;
48    }
49
50    public ActionType getActionType() {return actionType;}
51

```

```

52  @Override
53  public boolean equals(Object o) {
54      if (this == o) return true;
55      if (o == null || getClass() != o.getClass()) return false;
56      TNinetyAction that = (TNinetyAction) o;
57      return actionType == that.actionType;
58  }
59
60  @Override
61  public int hashCode() {
62      return Objects.hash(actionType);
63  }
64  }

```

Listing 60: representation/TNinetyAction.java

```

1  package representation;
2
3  import robocode.Robot;
4
5  import java.util.Random;
6
7  public class TNinetyActionRepresentation implements IActionRepresentation {
8      @Override
9      public void takeAction(Robot qLearningRobot, IAction action) {
10         if (action == null) {
11             System.out.println("Action is null");
12             return;
13         }
14         if (!(action instanceof TNinetyAction)) {
15             throw new IllegalArgumentException("TNinety representation can
16                 only take TNinety actions.");
17         }
18         TNinetyAction move = (TNinetyAction) action;
19         if (move.getActionType() == TNinetyAction.ActionType.TURN_LEFT) {
20             qLearningRobot.turnLeft(90);
21         } else if (move.getActionType() == TNinetyAction.ActionType.TURN_RIGHT) {
22             qLearningRobot.turnRight(90);
23         } else if (move.getActionType() == TNinetyAction.ActionType.AHEAD) {
24             qLearningRobot.ahead(100);
25         } else if (move.getActionType() == TNinetyAction.ActionType.FIRE) {
26             qLearningRobot.fire(18);
27         } else if (move.getActionType() == TNinetyAction.ActionType.RANDOMLY_MOVE) {
28             MoveRepresentation moveRepresentation = new MoveRepresentation();
29             System.out.println("TAKING RANDOM");
30             moveRepresentation.takeAction(qLearningRobot,
31                 moveRepresentation.getActions()[new Random().nextInt(3)]);
32         }
33     }
34
35     @Override
36     public IAction[] getActions() {
37         return new IAction[] {
38             new TNinetyAction(TNinetyAction.ActionType.AHEAD),

```

```

37         new TNinetyAction(TNinetyAction.ActionType.TURN_LEFT),
38         new TNinetyAction(TNinetyAction.ActionType.TURN_RIGHT),
39         new TNinetyAction(TNinetyAction.ActionType.FIRE),
40 //         new TNinetyAction(TNinetyAction.ActionType.RANDOMLY_MOVE),
41     };
42 }
43 }

```

Listing 61: representation/TNinetyActionRepresentation.java

```

1 package rl;
2
3 import fa.IFunctionApproximation;
4 import fa.LUT;
5 import policy.IPolicy;
6 import representation.*;
7
8 public interface ILearning {
9     IAction takeStep(IState lastState, IAction lastAction, IState
10         currentState);
11
12     IStateRepresentation getStateRepresentation();
13     IActionRepresentation getActionRepresentation();
14
15     IPolicy getPolicy();
16
17     IFunctionApproximation getFunctionApproximation();
18 }

```

Listing 62: rl/ILearning.java

```

1 package rl;
2
3 import fa.IFunctionApproximation;
4 import org.jetbrains.annotations.NotNull;
5 import policy.IPolicy;
6 import representation.*;
7
8 import java.util.ArrayList;
9 import java.util.Random;
10
11 public class QLearning implements ILearning {
12
13     private double epsilon;
14     private final double alpha;
15     private final double gamma;
16     private final Random random;
17     private IStateRepresentation stateRepresentation;
18     private IActionRepresentation actionRepresentation;
19     private IPolicy policy;
20     private IFunctionApproximation functionApproximation;
21     private IStateActionRepresentation stateActionRepresentation;
22     private int depth;
23     private boolean onlineLearning = false;
24     private ArrayList<IRepresentable> history = new ArrayList<>();
25     private ConcatenationRepresentation concatenation = new
26         ConcatenationRepresentation();

```

```

27 public QLearning(IStateRepresentation stateRepresentation ,
28                 IActionRepresentation actionRepresentation ,
29                 IStateActionRepresentation stateActionRepresentation ,
30                 IPolicy policy , IFunctionApproximation
31                 functionApproximation , double epsilon , double alpha
32                 , double gamma) {
33     this.stateActionRepresentation = stateActionRepresentation;
34     this.stateRepresentation = stateRepresentation;
35     this.actionRepresentation = actionRepresentation;
36     this.policy = policy;
37     this.epsilon = epsilon;
38     this.alpha = alpha;
39     this.gamma = gamma;
40     this.random = new Random();
41     this.functionApproximation = functionApproximation;
42     this.depth = 1;
43     this.onlineLearning = false;
44 }
45
46 public QLearning(IStateRepresentation stateRepresentation ,
47                 IActionRepresentation actionRepresentation ,
48                 IStateActionRepresentation stateActionRepresentation ,
49                 IPolicy policy , IFunctionApproximation
50                 functionApproximation ,
51                 double epsilon , double alpha , double gamma, int depth ,
52                 boolean onlineLearning) {     this.epsilon =
53                 epsilon;
54     this.alpha = alpha;
55     this.gamma = gamma;
56     this.stateRepresentation = stateRepresentation;
57     this.actionRepresentation = actionRepresentation;
58     this.policy = policy;
59     this.functionApproximation = functionApproximation;
60     this.stateActionRepresentation = stateActionRepresentation;
61     this.depth = depth;
62     this.random = new Random();
63     this.onlineLearning = onlineLearning;
64 }
65
66 @Override
67 public IAction takeStep(IState lastState , IAction lastAction , IState
68     currentState) {
69     if (lastState == null || lastAction == null || currentState == null
70     ) {
71         return explore();
72     }
73     System.out.println("Current state " + currentState);
74     IRepresentable oldSA = stateActionRepresentation.represent(
75         lastState ,
76         lastAction
77     );
78     history.add(oldSA);
79     while (history.size() > depth) {
80         history.remove(0);
81     }
82     if (history.size() < depth) return explore();
83     IAction bestAction = exploit(currentState);

```



```

78     IAction backupAction;
79     IAction toTakeAction;
80     double randomDouble = this.random.nextDouble();
81     boolean explored = false;
82     if (randomDouble < this.epsilon) {
83         IAction action = explore();
84         backupAction = action;
85         toTakeAction = action;
86         explored = true;
87     } else {
88         backupAction = bestAction;
89         toTakeAction = bestAction;
90     }
91     if (!onlineLearning) {
92         backupAction = bestAction;
93     }
94
95     for (int i = 0; i < history.size() - 1; i++) {
96         oldSA = new Concatenation(oldSA, history.get(i));
97     }
98     IRepresentable currentAction = stateActionRepresentation.represent(
99         currentState, backupAction);
100    for (int i = 1; i < history.size(); i++) {
101        currentAction = new Concatenation(currentAction, history.get(i)
102        );
103    }
104
105    if (explored) {
106        logAction(currentAction, "explored");
107    } else {
108        logAction(currentAction, "exploited");
109    }
110
111    double oldQ = functionApproximation.eval(oldSA)[0];
112    double r = policy.getReward(currentState, lastState); // why would
113    evaluate Rewards for last state?
114
115    double currentQ = functionApproximation.eval(currentAction)[0];
116    System.out.println("train " + oldQ + " = " + oldQ + " + " + alpha +
117        " (" + r + " + " + gamma + " * " + currentQ + " - " + oldQ + "
118        )");
119    System.out.println("train " + oldSA + " " + backupAction);
120    functionApproximation.train(
121        oldSA,
122        new double[] {
123            oldQ + alpha * (r + gamma * currentQ - oldQ)
124        }
125    );
126    return toTakeAction;
127 }
128
129 private IAction exploit(IState currentState) {
130     IAction bestAction = actionRepresentation.getActions()[0];
131     IRepresentable stateAction = stateActionRepresentation.represent(
132         currentState, bestAction);
133     for (int i = 1; i < history.size(); i++) {
134         stateAction = new Concatenation(stateAction, history.get(i));

```

```

130     }
131     double bestQ = functionApproximation.eval(stateAction)[0];
132
133     for (IAction action: actionRepresentation.getActions()) {
134         stateAction = stateActionRepresentation.represent(
135             currentState, action
136         );
137         for (int i = 1; i < history.size(); i++) {
138             stateAction = new Concatenation(stateAction, history.get(i)
139                 );
140         }
141         double q = functionApproximation.eval(
142             stateAction)[0];
143         System.out.print(" " + q + " " + action);
144         System.out.println();
145         if (q > bestQ) {
146             bestAction = action;
147             bestQ = q;
148         }
149     }
150     return bestAction;
151 }
152
153 private void logAction(IRepresentable bestAction, String hint) {
154     if (bestAction == null) return;
155     System.out.print(bestAction);
156     System.out.print(" " + hint + " ");
157     System.out.println(functionApproximation.eval(bestAction)[0]);
158 }
159
160 private IAction explore() {
161     IAction[] actions = actionRepresentation.getActions();
162     IAction action = actions[getRandom().nextInt(actions.length)];
163     return action;
164 }
165
166 @NotNull
167 private Random getRandom() {
168     return this.random;
169 }
170
171 @Override
172 public IStateRepresentation getStateRepresentation() {
173     return stateRepresentation;
174 }
175
176 @Override
177 public IActionRepresentation getActionRepresentation() {
178     return actionRepresentation;
179 }
180
181 @Override
182 public IPolicy getPolicy() {
183     return policy;
184 }
185
186 @Override
187 public IFunctionApproximation getFunctionApproximation() {

```

```

187         return functionApproximation;
188     }
189
190     public void setEpsilon(double epsilon) {
191         this.epsilon = epsilon;
192     }
193
194     public double getEpsilon() {
195         return this.epsilon;
196     }
197 }

```

Listing 63: rl/QLearning.java

```

1 package rl;
2
3 import fa.IFunctionApproximation;
4 import org.jetbrains.annotations.NotNull;
5 import policy.IPolicy;
6 import representation.*;
7
8 import java.util.Random;
9
10 public class SARSA Learning implements I Learning {
11
12     private double epsilon;
13     private final double alpha;
14     private final double gamma;
15     private final Random random;
16     private IStateRepresentation stateRepresentation;
17     private IActionRepresentation actionRepresentation;
18     private IPolicy policy;
19     private IFunctionApproximation functionApproximation;
20     private IStateActionRepresentation stateActionRepresentation;
21
22     public SARSA Learning(IStateRepresentation stateRepresentation,
23                         IActionRepresentation actionRepresentation,
24                         IStateActionRepresentation
25                             stateActionRepresentation,
26                         IPolicy policy, IFunctionApproximation
27                             functionApproximation, double epsilon, double
28                             alpha, double gamma) {
29         this.stateActionRepresentation = stateActionRepresentation;
30         this.stateRepresentation = stateRepresentation;
31         this.actionRepresentation = actionRepresentation;
32         this.policy = policy;
33         this.epsilon = epsilon;
34         this.alpha = alpha;
35         this.gamma = gamma;
36         this.random = new Random();
37         this.functionApproximation = functionApproximation;
38     }
39
40     @Override
41     public IAction takeStep(IState lastState, IAction lastAction, IState
42         currentState) {
43         if (lastState == null || lastAction == null || currentState == null
44             ) {

```

```

40         return explore();}
41     System.out.println("Current state " + currentState);
42     IRepresentable oldSA = stateActionRepresentation.represent(
43         lastState,
44         lastAction);
45     double oldQ = functionApproximation.eval(oldSA)[0];
46     double r = policy.getReward(currentState, lastState); // why would
47         evaluate Rewards for last state?
48     double newQ;
49
50     if (this.random.nextDouble() < this.epsilon) {
51         IAction action = explore();
52         IRepresentable exploreSA = stateActionRepresentation.represent(
53             currentState,
54             action);
55         newQ = functionApproximation.eval(exploreSA)[0];
56         System.out.println("train " + oldQ + " = " + oldQ + " + " +
57             alpha + " (" + r + " + " + gamma + " * " + newQ + " - " +
58             oldQ + ")");
59         System.out.println("train " + oldSA + " " + action);
60         functionApproximation.train(
61             oldSA,
62             new double[] {
63                 oldQ + alpha * (r + gamma * newQ - oldQ)
64             });
65         logAction(currentState, action, "explored");
66         return action;
67     } else {
68         IAction bestAction = exploit(currentState);
69         IRepresentable currentBest = stateActionRepresentation.
70             represent(
71                 currentState,
72                 bestAction);
73         newQ = functionApproximation.eval(currentBest)[0];
74         System.out.println("train " + oldQ + " = " + oldQ + " + " +
75             alpha + " (" + r + " + " + gamma + " * " + newQ + " - " +
76             oldQ + ")");
77         System.out.println("train " + oldSA + " " + bestAction);
78         functionApproximation.train(
79             oldSA,
80             new double[] {
81                 oldQ + alpha * (r + gamma * newQ - oldQ)
82             });
83         logAction(currentState, bestAction, "exploited");
84         return bestAction;
85     }
86 }
87
88 private IAction exploit(IState currentState) {
89     double bestQ = 0;
90     IAction bestAction = actionRepresentation.getActions()[0];
91     for (IAction action: actionRepresentation.getActions()) {
92         double q = functionApproximation.eval(
93             stateActionRepresentation.represent(
94                 currentState, action
95             ))[0];
96         System.out.print(" " + q + " " + action);
97         System.out.println();
98     }
99 }

```

```

92         if (q > bestQ) {
93             bestAction = action;
94             bestQ = q;
95         }
96     }
97     return bestAction;
98 }
99
100 private void logAction(IState currentState, IAction bestAction, String
    hint) {
101     if (bestAction == null) return;
102     System.out.print(bestAction);
103     System.out.print(" " + hint + " ");
104     System.out.println(functionApproximation.eval(
105         stateActionRepresentation.represent(
106             currentState, bestAction
107         ))[0]);
108 }
109
110 private IAction explore() {
111     IAction[] actions = actionRepresentation.getActions();
112     IAction action = actions[getRandom().nextInt(actions.length)];
113     return action;
114 }
115
116 @NotNull
117 private Random getRandom() {
118     return this.random;
119 }
120
121 @Override
122 public IStateRepresentation getStateRepresentation() {
123     return stateRepresentation;
124 }
125
126 @Override
127 public IActionRepresentation getActionRepresentation() {
128     return actionRepresentation;
129 }
130
131 @Override
132 public IPolicy getPolicy() {
133     return policy;
134 }
135
136 @Override
137 public IFunctionApproximation getFunctionApproximation() {
138     return functionApproximation;
139 }
140
141 public void setEpsilon(double epsilon) {
142     this.epsilon = epsilon;
143 }
144
145 public double getEpsilon() {
146     return this.epsilon;
147 }
148 }

```

---

Listing 64: rl/SARSA Learning.java

```
1  /*
2  * Copyright (c) 2001–2021 Mathew A. Nelson and Robocode contributors
3  * All rights reserved. This program and the accompanying materials
4  * are made available under the terms of the Eclipse Public License v1.0
5  * which accompanies this distribution, and is available at
6  * https://robocode.sourceforge.io/license/epl-v10.html
7  */
8  package robot;
9
10
11 import robocode.DeathEvent;
12 import robocode.Robot;
13 import robocode.ScannedRobotEvent;
14 import static robocode.util.Utils.normalRelativeAngleDegrees;
15
16 import java.awt.*;
17
18
19 /**
20 * Corners – a sample robot by Mathew Nelson.
21 * <p>
22 * This robot moves to a corner, then swings the gun back and forth.
23 * If it dies, it tries a new corner in the next round.
24 *
25 * @author Mathew A. Nelson (original)
26 * @author Flemming N. Larsen (contributor)
27 */
28 public class Corners extends Robot {
29     int others; // Number of other robots in the game
30     static int corner = 0; // Which corner we are currently using
31     // static so that it keeps it between rounds.
32     boolean stopWhenSeeRobot = false; // See goCorner()
33
34     /**
35      * run: Corners' main run function.
36      */
37     public void run() {
38         // Set colors
39         setBodyColor(Color.red);
40         setGunColor(Color.black);
41         setRadarColor(Color.yellow);
42         setBulletColor(Color.green);
43         setScanColor(Color.green);
44
45         // Save # of other bots
46         others = getOthers();
47
48         // Move to a corner
49         goCorner();
50
51         // Initialize gun turn speed to 3
52         int gunIncrement = 3;
53
54         // Spin gun back and forth
55         while (true) {
```

```

56         for (int i = 0; i < 30; i++) {
57             turnGunLeft(gunIncrement);
58         }
59         gunIncrement *= -1;
60     }
61 }
62
63 /**
64  * goCorner: A very inefficient way to get to a corner. Can you do
65  * better?
66  */
67 public void goCorner() {
68     // We don't want to stop when we're just turning...
69     stopWhenSeeRobot = false;
70     // turn to face the wall to the "right" of our desired corner.
71     turnRight(normalRelativeAngleDegrees(corner - getHeading()));
72     // Ok, now we don't want to crash into any robot in our way...
73     stopWhenSeeRobot = true;
74     // Move to that wall
75     ahead(5000);
76     // Turn to face the corner
77     turnLeft(90);
78     // Move to the corner
79     ahead(5000);
80     // Turn gun to starting point
81     turnGunLeft(90);
82 }
83
84 /**
85  * onScannedRobot: Stop and fire!
86  */
87 public void onScannedRobot(ScannedRobotEvent e) {
88     // Should we stop, or just fire?
89     if (stopWhenSeeRobot) {
90         // Stop everything! You can safely call stop multiple times.
91         stop();
92         // Call our custom firing method
93         smartFire(e.getDistance());
94         // Look for another robot.
95         // NOTE: If you call scan() inside onScannedRobot, and it sees
96         // a robot,
97         // the game will interrupt the event handler and start it over
98         scan();
99         // We won't get here if we saw another robot.
100        // Okay, we didn't see another robot... start moving or turning
101        again.
102        resume();
103    } else {
104        smartFire(e.getDistance());
105    }
106 }
107
108 /**
109  * smartFire: Custom fire method that determines firepower based on
110  * distance.
111  *
112  * @param robotDistance the distance to the robot to fire at
113  */

```

```

110 public void smartFire(double robotDistance) {
111     if (robotDistance > 200 || getEnergy() < 15) {
112         fire(1);
113     } else if (robotDistance > 50) {
114         fire(2);
115     } else {
116         fire(3);
117     }
118 }
119
120 /**
121  * onDeath: We died. Decide whether to try a different corner next
122  * game.
123  */
124 public void onDeath(DeathEvent e) {
125     // Well, others should never be 0, but better safe than sorry.
126     if (others == 0) {
127         return;
128     }
129     // If 75% of the robots are still alive when we die, we'll switch
130     // corners.
131     if ((others - getOthers()) / (double) others < .75) {
132         corner += 90;
133         if (corner == 270) {
134             corner = -90;
135         }
136         out.println("I died and did poorly... switching corner to " +
137             corner);
138     } else {
139         out.println("I died but did well. I will still use corner " +
140             corner);
141     }
142 }

```

Listing 65: robot/Corners.java

```

1 package robot;
2
3 import fa.IFunctionApproximation;
4 import fa.LUT;
5 import policy.EnergyReward;
6 import policy.IPolicy;
7 import representation.*;
8 import rl.ILearning;
9 import rl.QLearning;
10 import robocode.ScannedRobotEvent;
11
12 public class LUTTNinetyRobot05 extends QLearningRobot {
13     public LUTTNinetyRobot05() {
14         super(createLearning());
15     }
16
17
18     public static ILearning createLearning() {
19         IActionRepresentation actionRepresentation = new
20             TNinetyActionRepresentation();

```



```

20     IStateRepresentation stateRepresentation = new StateRep();
21     IFunctionApproximation functionApproximation = new LUT("
22         LUTTNinetyRobot.obj", false);
23     IPolicy policy = new EnergyReward();
24     return new QLearning(
25         stateRepresentation,
26         actionRepresentation,
27         new ConcatenationRepresentation(),
28         policy,
29         functionApproximation,
30         0.5, 0.1, 0.9, 3, false);
31 }
32
33 @Override
34 public void onScannedRobot(ScannedRobotEvent event) {
35     super.onScannedRobot(event);
36     System.out.println("HELLLLOOOOOO");
37 }

```

Listing 66: robot/LUTTNinetyRobot05.java

```

1 package robot;
2
3 import fa.IFunctionApproximation;
4 import fa.LUT;
5 import policy.EnergyReward;
6 import policy.IPolicy;
7 import representation.*;
8 import rl.ILearning;
9 import rl.QLearning;
10 import robocode.ScannedRobotEvent;
11
12 public class LUTTNinetyRobot0 extends QLearningRobot {
13     public LUTTNinetyRobot0() {
14         super(createLearning());
15     }
16
17     public static ILearning createLearning() {
18         IActionRepresentation actionRepresentation = new
19             TNinetyActionRepresentation();
20         IStateRepresentation stateRepresentation = new StateRep();
21         IFunctionApproximation functionApproximation = new LUT("
22             LUTTNinetyRobot.obj", false);
23         IPolicy policy = new EnergyReward();
24         return new QLearning(
25             stateRepresentation,
26             actionRepresentation,
27             new ConcatenationRepresentation(),
28             policy,
29             functionApproximation,
30             0.0, 0.1, 0.9, 3, false);
31     }
32
33 @Override
34 public void onScannedRobot(ScannedRobotEvent event) {
35     super.onScannedRobot(event);

```

```

35     System.out.println("HELLOOOOOOO");
36 }
37 }

```

Listing 67: robot/LUTTNinetyRobot0.java

```

1 package robot;
2
3 import fa.IFunctionApproximation;
4 import fa.LUT;
5 import policy.EnergyReward;
6 import policy.IPolicy;
7 import representation.*;
8 import rl.ILearning;
9 import rl.QLearning;
10 import robocode.ScannedRobotEvent;
11
12 public class LUTTNinetyRobotConfident extends QLearningRobot {
13     public LUTTNinetyRobotConfident() {
14         super(createLearning());
15     }
16
17
18     public static ILearning createLearning() {
19         IActionRepresentation actionRepresentation = new
20             TNinetyActionRepresentation();
21         IStateRepresentation stateRepresentation = new StateRep();
22         IFunctionApproximation functionApproximation = new LUT("
23             LUTTNinetyRobot.obj", true);
24         IPolicy policy = new EnergyReward();
25         return new QLearning(
26             stateRepresentation,
27             actionRepresentation,
28             new ConcatenationRepresentation(),
29             policy,
30             functionApproximation,
31             0.05, 0.1, 0.8, 3, false);
32     }
33
34     @Override
35     public void onScannedRobot(ScannedRobotEvent event) {
36         super.onScannedRobot(event);
37         System.out.println("HELLOOOOOOO");
38     }
39 }

```

Listing 68: robot/LUTTNinetyRobotConfident.java

```

1 package robot;
2
3 import fa.IFunctionApproximation;
4 import fa.LUT;
5 import policy.EnergyReward;
6 import policy.GoTopRight;
7 import policy.IPolicy;
8 import representation.*;
9 import rl.ILearning;
10 import rl.QLearning;

```

```

11 import robocode.ScannedRobotEvent;
12
13 public class LUTTNinetyRobot extends QLearningRobot {
14     public LUTTNinetyRobot() {
15         super(createLearning());
16     }
17
18
19     public static ILearning createLearning() {
20         IActionRepresentation actionRepresentation = new
21             TNinetyActionRepresentation();
22         IStateRepresentation stateRepresentation = new StateRep();
23         IFunctionApproximation functionApproximation = new LUT("
24             LUTTNinetyRobot.obj", false);
25         IPolicy policy = new EnergyReward();
26         return new QLearning(
27             stateRepresentation,
28             actionRepresentation,
29             new ConcatenationRepresentation(),
30             policy,
31             functionApproximation,
32             0.8, 0.1, 0.9, 3, false);
33     }
34
35     @Override
36     public void onScannedRobot(ScannedRobotEvent event) {
37         super.onScannedRobot(event);
38         System.out.println("HELLLLOOOOOOO");
39     }
40 }

```

Listing 69: robot/LUTTNinetyRobot.java

```

1 package robot;
2
3 import fa.IFunctionApproximation;
4 import fa.LUT;
5 import policy.EnergyReward;
6 import policy.IPolicy;
7 import representation.*;
8 import rl.ILearning;
9 import rl.QLearning;
10 import robocode.ScannedRobotEvent;
11
12 public class LUTTNinetyRobotOnline extends QLearningRobot {
13     public LUTTNinetyRobotOnline() {
14         super(createLearning());
15     }
16
17
18     public static ILearning createLearning() {
19         IActionRepresentation actionRepresentation = new
20             TNinetyActionRepresentation();
21         IStateRepresentation stateRepresentation = new StateRep();
22         IFunctionApproximation functionApproximation = new LUT("
23             LUTTNinetyRobot.obj", false);
24         IPolicy policy = new EnergyReward();
25         return new QLearning(

```

```

24         stateRepresentation ,
25         actionRepresentation ,
26         new ConcatenationRepresentation() ,
27         policy ,
28         functionApproximation ,
29         0.8, 0.1, 0.9, 3, true);
30     }
31
32     @Override
33     public void onScannedRobot(ScannedRobotEvent event) {
34         super.onScannedRobot(event);
35         System.out.println("HELLLLLLOOOOOO");
36     }
37 }

```

Listing 70: robot/LUTTNinetyRobotOnline.java

```

1 package robot;
2
3 import fa.IFunctionApproximation;
4 import fa.LUT;
5 import policy.EnergyReward;
6 import policy.EnergyRewardTerminal;
7 import policy.IPolicy;
8 import representation.*;
9 import rl.ILearning;
10 import rl.QLearning;
11 import robocode.ScannedRobotEvent;
12
13 public class LUTTNinetyRobotTerminal extends QLearningRobot {
14     public LUTTNinetyRobotTerminal() {
15         super(createLearning());
16     }
17
18
19     public static ILearning createLearning() {
20         IActionRepresentation actionRepresentation = new
21             TNinetyActionRepresentation();
22         IStateRepresentation stateRepresentation = new StateRep();
23         IFunctionApproximation functionApproximation = new LUT("
24             LUTTNinetyRobot.obj", false);
25         IPolicy policy = new EnergyRewardTerminal();
26         return new QLearning(
27             stateRepresentation ,
28             actionRepresentation ,
29             new ConcatenationRepresentation() ,
30             policy ,
31             functionApproximation ,
32             0.8, 0.1, 0.9, 3, false);
33     }
34
35     @Override
36     public void onScannedRobot(ScannedRobotEvent event) {
37         super.onScannedRobot(event);
38         System.out.println("HELLLLLLOOOOOO");
39     }
40 }

```

Listing 71: robot/LUTTNinetyRobotTerminal.java

```

1 package robot;
2
3 /**
4  * DISCLAIMER: the code below has been auto-generated by Robocode
5  *   http://robocode.sourceforge.net/
6  */
7
8 import robocode.*;
9
10 // API help : http://robocode.sourceforge.net/docs/robocode/robocode/Robot.html
11
12 /**
13  * MyFirstRobot - a robot by (your name here)
14  */
15 public class MyFirstRobot extends AdvancedRobot
16 {
17     /**
18      * run: MyFirstRobot's default behavior
19      */
20     public void run() {
21         // Initialization of the robot should be put here
22
23         // After trying out your robot, try uncommenting the import at the
24         // top,
25         // and the next line:
26
27         // setColors(Color.red,Color.blue,Color.green); // body,gun,radar
28
29         // Robot main loop
30         while(true) {
31             // Replace the next 4 lines with any behavior you would like
32             ahead(100);
33             turnGunRight(360);
34             back(100);
35             turnGunRight(360);
36         }
37
38     /**
39      * onScannedRobot: What to do when you see another robot
40      */
41     public void onScannedRobot(ScannedRobotEvent e) {
42         // Replace the next line with any behavior you would like
43         fire(1);
44     }
45
46     /**
47      * onHitByBullet: What to do when you're hit by a bullet
48      */
49     public void onHitByBullet(HitByBulletEvent e) {
50         // Replace the next line with any behavior you would like
51         back(10);
52     }
53
54     /**
55      * onHitWall: What to do when you hit a wall

```

```

56     */
57     public void onHitWall(HitWallEvent e) {
58         // Replace the next line with any behavior you would like
59         back(20);
60     }
61 }

```

Listing 72: robot/MyFirstRobot.java

```

1 package robot;
2
3 import fa.IFunctionApproximation;
4 import fa.NN;
5 import policy.EnergyReward;
6 import policy.IPolicy;
7 import representation.*;
8 import rl.ILearning;
9 import rl.QLearning;
10
11 public class NNTNinetyRobotConfident extends QLearningRobot {
12     public NNTNinetyRobotConfident() {
13         super(createLearning());
14     }
15
16
17     public static ILearning createLearning() {
18         IActionRepresentation actionRepresentation = new
19             TNinetyActionRepresentation();
20         IStateRepresentation stateRepresentation = new StateRep();
21         IFunctionApproximation functionApproximation = new NN("
22             NNTNinetyRobot.obj", true);
23         IPolicy policy = new EnergyReward();
24         return new QLearning(
25             stateRepresentation,
26             actionRepresentation,
27             new ConcatenationRepresentation(),
28             policy,
29             functionApproximation,
30             0.05, 0.1, 0.9, 3, false);
31     }
32 }

```

Listing 73: robot/NNTNinetyRobotConfident.java

```

1 package robot;
2
3 import fa.IFunctionApproximation;
4 import fa.LUT;
5 import fa.NN;
6 import fa.NNLUT;
7 import policy.EnergyReward;
8 import policy.IPolicy;
9 import representation.*;
10 import rl.ILearning;
11 import rl.QLearning;
12 import robocode.Robot;
13

```

```

14 public class NNTNinetyRobot extends QLearningRobot {
15     public NNTNinetyRobot() {
16         super(createLearning());
17     }
18
19
20     public static ILearning createLearning() {
21         IActionRepresentation actionRepresentation = new
22             TNinetyActionRepresentation();
23         IStateRepresentation stateRepresentation = new StateRep();
24         IFunctionApproximation functionApproximation = new NN("
25             NNTNinetyRobot.obj", false, 20);
26         // IFunctionApproximation functionApproximation = new NNLUT();
27         // IFunctionApproximation functionApproximation = new LUT("
28             NNTNinetyRobot.obj", false);
29         IPolicy policy = new EnergyReward();
30         return new QLearning(
31             stateRepresentation,
32             actionRepresentation,
33             new ConcatenationRepresentation(),
34             policy,
35             functionApproximation,
36             0.8, 0.1, 0.9, 3, false);
37     }
38 }

```

Listing 74: robot/NNTNinetyRobot.java

```

1 package robot;
2
3 import representation.IAction;
4 import representation.IState;
5 import rl.ILearning;
6 import robocode.*;
7
8 import java.io.IOException;
9
10 public class QLearningRobot extends Robot {
11     private ILearning learning;
12     private IState state;
13     private IState lastState;
14     private IAction lastAction;
15     private long lastTurn;
16     private int turn = 0;
17     private StatusEvent lastStatusEvent;
18
19     @Override
20     public void run() {
21         super.run();
22         // setAdjustGunForRobotTurn(true);
23         // setAdjustRadarForGunTurn(true);
24         while (true) {
25             // Replace the next 4 lines with any behavior you would like
26             // ahead(100);
27             // turnGunRight(360);
28             // back(100);
29             // turnRadarLeft(360);

```

```

30         turnGunRight(360);
31         turn ++;
32     }
33 }
34
35 public int getTurn() {
36     return turn;
37 }
38
39 public QLearningRobot(ILearning learning) {
40     this.learning = learning;
41     try {
42         this.learning.getFunctionApproximation().load();
43     } catch (IOException e) {
44         e.printStackTrace();
45     } catch (ClassNotFoundException e) {
46         e.printStackTrace();
47     }
48 }
49
50 public ILearning getLearning() {
51     return learning;
52 }
53
54
55 private IState getLastState() {
56     return lastState;
57 }
58
59 public IState getState() {
60     return this.state;
61 }
62
63 @Override
64 public void onScannedRobot(ScannedRobotEvent event) {
65     super.onScannedRobot(event);
66     processEvent(event);
67 }
68
69 private void processEvent(Event event) {
70     if (event instanceof ScannedRobotEvent && this.getTurn() > this.
71         lastTurn || event instanceof WinEvent || event instanceof
72         DeathEvent) {
73         IState newState = learning.getStateRepresentation().represent(
74             getState(), lastStatusEvent);
75         newState = learning.getStateRepresentation().represent(newState
76             , event);
77         //set current state
78         setState(newState);
79         IAction action = learning.takeStep(getLastState(),
80             getLastAction(), getState());
81         //take new action
82         this.lastTurn = this.getTurn();
83         System.out.println("Turn " + this.lastTurn);
84         takeAction(action);
85     }
86 }

```



```

83  @Override
84  public void onWin(WinEvent event) {
85      super.onWin(event);
86      processEvent(event);
87  }
88
89  @Override
90  public void onDeath(DeathEvent event) {
91      super.onDeath(event);
92      processEvent(event);
93  }
94
95  private IAction getLastAction() {
96      return lastAction;
97  }
98
99  private void takeAction(IAction action) {
100     learning.getActionRepresentation().takeAction(this, action);
101     if (action != null) lastAction = action;
102 }
103
104 public void setState(IState state) {
105     this.lastState = this.state;
106     this.state = state;
107 }
108
109 @Override
110 public void onRoundEnded(RoundEndedEvent event) {
111     try {
112         learning.getFunctionApproximation().save();
113     } catch (IOException e) {
114         e.printStackTrace();
115     }
116 }
117
118 @Override
119 public void onStatus(StatusEvent e) {
120     lastStatusEvent = e;
121 }
122 }

```

Listing 75: robot/QLearningRobot.java

```

1  package robot;
2
3  import robocode.Robot;
4  import robocode.StatusEvent;
5
6  public class TopLeftCornerRobot extends Robot {
7      @Override
8      public void run() {
9          super.run();
10         ahead(100);
11     }
12
13     @Override
14     public void onStatus(StatusEvent e) {
15         super.onStatus(e);

```

```

16     var yDistance = getBattleFieldHeight() - getY();
17     var xDistance = getX();
18     var angle = Math.atan(yDistance / xDistance);
19     angle = 270 + Math.toDegrees(angle);
20     System.out.printf("%f, %f, %f, %f %n", getY(), getX(), getHeading()
21         , angle);
22     double absHeadingDiff = Math.abs(getHeading() - angle);
23     if (getHeading() < angle) {
24         if (absHeadingDiff > 180) {
25             turnLeft(360 - absHeadingDiff);
26         } else {
27             turnRight(absHeadingDiff);
28         }
29     } else {
30         if (absHeadingDiff > 180) {
31             turnRight(360 - absHeadingDiff);
32         } else {
33             turnLeft(absHeadingDiff);
34         }
35     }
36     if (absHeadingDiff < 0.01) {
37         ahead(100);
38     }
39 }

```

Listing 76: robot/TopLeftCornerRobot.java

```

1  /*
2   * Copyright (c) 2001–2021 Mathew A. Nelson and Robocode contributors
3   * All rights reserved. This program and the accompanying materials
4   * are made available under the terms of the Eclipse Public License v1.0
5   * which accompanies this distribution, and is available at
6   * https://robocode.sourceforge.io/license/epl-v10.html
7   */
8  package robot;
9
10
11  import robocode.HitRobotEvent;
12  import robocode.Robot;
13  import robocode.ScannedRobotEvent;
14  import robocode.WinEvent;
15  import static robocode.util.Utils.normalRelativeAngleDegrees;
16
17  import java.awt.*;
18
19
20  /**
21   * Tracker – a sample robot by Mathew Nelson.
22   * <p>
23   * Locks onto a robot, moves close, fires when close.
24   *
25   * @author Mathew A. Nelson (original)
26   * @author Flemming N. Larsen (contributor)
27   */
28  public class Tracker extends Robot {
29      int count = 0; // Keeps track of how long we've
30      // been searching for our target

```

```

31 double gunTurnAmt; // How much to turn our gun when searching
32 String trackName; // Name of the robot we're currently tracking
33
34 /**
35  * run: Tracker's main run function
36  */
37 public void run() {
38     // Set colors
39     setBodyColor(new Color(128, 128, 50));
40     setGunColor(new Color(50, 50, 20));
41     setRadarColor(new Color(200, 200, 70));
42     setScanColor(Color.white);
43     setBulletColor(Color.blue);
44
45     // Prepare gun
46     trackName = null; // Initialize to not tracking anyone
47     setAdjustGunForRobotTurn(true); // Keep the gun still when we turn
48     gunTurnAmt = 10; // Initialize gunTurn to 10
49
50     // Loop forever
51     while (true) {
52         // turn the Gun (looks for enemy)
53         turnGunRight(gunTurnAmt);
54         // Keep track of how long we've been looking
55         count++;
56         // If we've haven't seen our target for 2 turns, look left
57         if (count > 2) {
58             gunTurnAmt = -10;
59         }
60         // If we still haven't seen our target for 5 turns, look right
61         if (count > 5) {
62             gunTurnAmt = 10;
63         }
64         // If we *still* haven't seen our target after 10 turns, find
65         // another target
66         if (count > 11) {
67             trackName = null;
68         }
69     }
70
71     /**
72     * onScannedRobot: Here's the good stuff
73     */
74     public void onScannedRobot(ScannedRobotEvent e) {
75
76         // If we have a target, and this isn't it, return immediately
77         // so we can get more ScannedRobotEvents.
78         if (trackName != null && !e.getName().equals(trackName)) {
79             return;
80         }
81
82         // If we don't have a target, well, now we do!
83         if (trackName == null) {
84             trackName = e.getName();
85             out.println("Tracking " + trackName);
86         }
87         // This is our target. Reset count (see the run method)

```

```

88     count = 0;
89     // If our target is too far away, turn and move toward it.
90     if (e.getDistance() > 150) {
91         gunTurnAmt = normalRelativeAngleDegrees(e.getBearing() + (
92             getHeading() - getRadarHeading()));
93
94         turnGunRight(gunTurnAmt); // Try changing these to
95             setTurnGunRight,
96         turnRight(e.getBearing()); // and see how much Tracker improves
97             ...
98         // (you'll have to make Tracker an AdvancedRobot)
99         ahead(e.getDistance() - 140);
100         return;
101     }
102
103     // Our target is close.
104     gunTurnAmt = normalRelativeAngleDegrees(e.getBearing() + (
105         getHeading() - getRadarHeading()));
106     turnGunRight(gunTurnAmt);
107     fire(3);
108
109     // Our target is too close! Back up.
110     if (e.getDistance() < 100) {
111         if (e.getBearing() > -90 && e.getBearing() <= 90) {
112             back(40);
113         } else {
114             ahead(40);
115         }
116     }
117     scan();
118 }
119
120 /**
121  * onHitRobot: Set him as our new target
122  */
123 public void onHitRobot(HitRobotEvent e) {
124     // Only print if he's not already our target.
125     if (trackName != null && !trackName.equals(e.getName())) {
126         out.println("Tracking " + e.getName() + " due to collision");
127     }
128     // Set the target
129     trackName = e.getName();
130     // Back up a bit.
131     // Note: We won't get scan events while we're doing this!
132     // An AdvancedRobot might use setBack(); execute();
133     gunTurnAmt = normalRelativeAngleDegrees(e.getBearing() + (
134         getHeading() - getRadarHeading()));
135     turnGunRight(gunTurnAmt);
136     fire(3);
137     back(50);
138 }
139
140 /**
141  * onWin: Do a victory dance
142  */
143 public void onWin(WinEvent e) {
144     for (int i = 0; i < 50; i++) {
145         turnRight(30);

```

```

141         turnLeft(30);
142     }
143 }
144 }

```

Listing 77: robot/Tracker.java

```

1 package robot;
2
3 import fa.IFunctionApproximation;
4 import fa.LUT;
5 import policy.EnergyReward;
6 import policy.GoTopRight;
7 import policy.IPolicy;
8 import representation.*;
9 import rl.ILearning;
10 import rl.QLearning;
11
12 public class TrivialLUTRobotConfident extends QLearningRobot {
13     public TrivialLUTRobotConfident() {
14         super(createLearning());
15     }
16
17
18     public static ILearning createLearning() {
19         IActionRepresentation actionRepresentation = new MoveRepresentation
20             ();
21         IStateRepresentation stateRepresentation = new
22             CoordinatesRepresentation();
23         IFunctionApproximation functionApproximation = new LUT("
24             TrivialLUTRobot.obj", true);
25         IPolicy policy = new GoTopRight();
26         return new QLearning(
27             stateRepresentation,
28             actionRepresentation,
29             new ConcatenationRepresentation(),
30             policy,
31             functionApproximation,
32             0.05, 0.1, 0.8);
33     }
34 }

```

Listing 78: robot/TrivialLUTRobotConfident.java

```

1 package robot;
2
3 import fa.IFunctionApproximation;
4 import fa.LUT;
5 import policy.EnergyReward;
6 import policy.GoTopRight;
7 import policy.IPolicy;
8 import representation.*;
9 import rl.ILearning;
10 import rl.QLearning;
11
12 public class TrivialLUTRobot extends QLearningRobot {
13     public TrivialLUTRobot() {
14         super(createLearning());
15     }
16 }

```

```

15     }
16
17
18     public static ILearning createLearning() {
19         IActionRepresentation actionRepresentation = new MoveRepresentation
20             ();
21         IStateRepresentation stateRepresentation = new
22             CoordinatesRepresentation();
23         IFunctionApproximation functionApproximation = new LUT("
24             TrivialLUTRobot.obj", false);
25         IPolicy policy = new GoTopRight();
26         return new QLearning(
27             stateRepresentation,
28             actionRepresentation,
29             new ConcatenationRepresentation(),
30             policy,
31             functionApproximation,
32             0.8, 0.1, 0.8);
33     }
34 }

```

Listing 79: robot/TrivialLUTRobot.java

```

1 package autograd;
2
3
4 import org.junit.Assert;
5 import org.junit.Test;
6
7 public class VariableTest {
8
9     @Test
10     public void testAddition() {
11         Assert.assertEquals(new Addition().apply(new Parameter(12), new
12             Parameter(2)).evaluate(), 14., 0.);
13     }
14
15     @Test
16     public void testVariableEvaluation() {
17         Assert.assertEquals(new Parameter(250).evaluate(), 250., 0);
18     }
19 }

```

Listing 80: autograd/VariableTest.java

```

1 package fa;
2
3 import org.junit.Assert;
4 import org.junit.Ignore;
5 import org.junit.Test;
6 import representation.IRepresentable;
7
8 public class TestFunctionApproximation {
9     @Ignore
10     @Test
11     public void TestLUT() {
12         LUT lut = new LUT(null, true);

```

```

13     IRepresentable desiredState = new IRepresentable() {
14         @Override
15         public double[] toVector() {
16             return new double[0];
17         }
18     };
19     IRepresentable otherState = new IRepresentable() {
20         @Override
21         public double[] toVector() {
22             return new double[0];
23         }
24     };
25     double[] desiredResponse = new double[] {10};
26     double[] otherDesiredResponse = new double[] {5};
27     lut.train(desiredState, desiredResponse);
28     Assert.assertEquals(desiredResponse, lut.eval(desiredState),
29         0.);
30     lut.train(otherState, otherDesiredResponse);
31     Assert.assertEquals(otherDesiredResponse, lut.eval(otherState),
32         0.);
33     lut.train(desiredState, otherDesiredResponse);
34     Assert.assertEquals(otherDesiredResponse, lut.eval(
        desiredState), 0.);
    }
}

```

Listing 81: fa/TestFunctionApproximation.java

```

1 package nn;
2
3 import autograd.Parameter;
4 import jdk.jshell.spi.ExecutionControl;
5 import org.junit.Assert;
6 import org.junit.Test;
7
8 import java.util.Arrays;
9
10 public class NeuralNetworkTest {
11
12     @Test
13     public void testNeuralNetworkFactory() {
14         var model = Factory.createNeuralNetwork(new int[] {2, 4, 1}, new
15             Sigmoid());
16         var result = model.evaluate(new double[] {0, 0});
17         Assert.assertEquals(result.length, 1);
18     }
19
20     @Test
21     public void testNeuralNetworkGradient() throws ExecutionControl.
22         NotImplementedException {
23         var model = Factory.createNeuralNetwork(new int[] {2, 4, 1}, new
24             Sigmoid());
25         Parameter[] parameters = model.getTrainableParameters();
26         for (Parameter parameter :
27             parameters) {
28             parameter.setValue(1);
29         }
30         var result = model.evaluate(new double[] {1, 0});
31     }
32 }

```

```

28 double[] desired = new double[] { 1 };
29 Assert.assertEquals(result.length, 1);
30 double delta = 1e-5;
31 double expected = 0.9892621636390686; // obtained by pytorch
32 Assert.assertEquals(result[0], expected, delta);
33 var loss = new MeanSquaredError(model.getOutput());
34 loss.setDesired(desired);
35 loss.backward(parameters, 1);
36 var gradients = new double[parameters.length];
37 for (int i = 0; i < parameters.length; i++) {
38     gradients[i] = parameters[i].getGradient();
39 }
40
41 Assert.assertArrayEquals(Arrays.stream(new double[] { // calculated
42     by pytorch
43         -0.000114063048386015, -0.00010046639363281429,
44         -0.00010046639363281429, -0.00010046639363281429,
45         -0.00010046639363281429, -1.197589335788507e-05,
46         -1.197589335788507e-05, -1.197589335788507e-05,
47         -1.197589335788507e-05, -1.197589335788507e-05,
48         -1.197589335788507e-05, -1.197589335788507e-05,
49         -1.197589335788507e-05, -0.0, -0.0, -0.0, -0.0
50     }).sorted().toArray(), Arrays.stream(gradients).sorted().toArray(),
51     delta);
52 }
53
54 @Test
55 public void testNeuralNetworkGradientBipolar() throws ExecutionControl.
56     NotImplementedException {
57     var model = Factory.createNeuralNetwork(new int[] { 2, 4, 1 }, new
58         BipolarSigmoid());
59     Parameter[] parameters = model.getTrainableParameters();
60     for (Parameter parameter :
61         parameters) {
62         parameter.setValue(1);
63     }
64     var result = model.evaluate(new double[] { 1, -1 });
65     double[] desired = new double[] { 1 };
66     Assert.assertEquals(result.length, 1);
67     double delta = 1e-5;
68     double expected = 0.8904789686203003; // obtained by pytorch
69     Assert.assertEquals(expected, result[0], delta);
70     var loss = new MeanSquaredError(model.getOutput());
71     loss.setDesired(desired);
72     loss.backward(parameters, 1);
73     var gradients = new double[parameters.length];
74     for (int i = 0; i < parameters.length; i++) {
75         gradients[i] = parameters[i].getGradient();
76     }
77
78     Assert.assertArrayEquals(Arrays.stream(new double[] { // calculated
79         by pytorch
80             -0.011338012292981148, -0.005239490419626236,
81             -0.005239490419626236, -0.005239490419626236,
82             -0.005239490419626236, -0.004458376672118902,
83             -0.004458376672118902, -0.004458376672118902,
84             -0.004458376672118902, -0.004458376672118902,
85             -0.004458376672118902, -0.004458376672118902,
86             -0.004458376672118902, -0.004458376672118902,
87             -0.004458376672118902, -0.004458376672118902,
88             -0.004458376672118902, -0.004458376672118902,
89             -0.004458376672118902, -0.004458376672118902,
90             -0.004458376672118902, -0.004458376672118902,
91             -0.004458376672118902, -0.004458376672118902,
92             -0.004458376672118902, -0.004458376672118902,
93             -0.004458376672118902, -0.004458376672118902,
94             -0.004458376672118902, -0.004458376672118902,
95             -0.004458376672118902, -0.004458376672118902,
96             -0.004458376672118902, -0.004458376672118902,
97             -0.004458376672118902, -0.004458376672118902,
98             -0.004458376672118902, -0.004458376672118902,
99             -0.004458376672118902, -0.004458376672118902,
100             -0.004458376672118902, -0.004458376672118902,
101             -0.004458376672118902, -0.004458376672118902,
102             -0.004458376672118902, -0.004458376672118902,
103             -0.004458376672118902, -0.004458376672118902,
104             -0.004458376672118902, -0.004458376672118902,
105             -0.004458376672118902, -0.004458376672118902,
106             -0.004458376672118902, -0.004458376672118902,
107             -0.004458376672118902, -0.004458376672118902,
108             -0.004458376672118902, -0.004458376672118902,
109             -0.004458376672118902, -0.004458376672118902,
110             -0.004458376672118902, -0.004458376672118902,
111             -0.004458376672118902, -0.004458376672118902,
112             -0.004458376672118902, -0.004458376672118902,
113             -0.004458376672118902, -0.004458376672118902,
114             -0.004458376672118902, -0.004458376672118902,
115             -0.004458376672118902, -0.004458376672118902,
116             -0.004458376672118902, -0.004458376672118902,
117             -0.004458376672118902, -0.004458376672118902,
118             -0.004458376672118902, -0.004458376672118902,
119             -0.004458376672118902, -0.004458376672118902,
120             -0.004458376672118902, -0.004458376672118902,
121             -0.004458376672118902, -0.004458376672118902,
122             -0.004458376672118902, -0.004458376672118902,
123             -0.004458376672118902, -0.004458376672118902,
124             -0.004458376672118902, -0.004458376672118902,
125             -0.004458376672118902, -0.004458376672118902,
126             -0.004458376672118902, -0.004458376672118902,
127             -0.004458376672118902, -0.004458376672118902,
128             -0.004458376672118902, -0.004458376672118902,
129             -0.004458376672118902, -0.004458376672118902,
130             -0.004458376672118902, -0.004458376672118902,
131             -0.004458376672118902, -0.004458376672118902,
132             -0.004458376672118902, -0.004458376672118902,
133             -0.004458376672118902, -0.004458376672118902,
134             -0.004458376672118902, -0.004458376672118902,
135             -0.004458376672118902, -0.004458376672118902,
136             -0.004458376672118902, -0.004458376672118902,
137             -0.004458376672118902, -0.004458376672118902,
138             -0.004458376672118902, -0.004458376672118902,
139             -0.004458376672118902, -0.004458376672118902,
140             -0.004458376672118902, -0.004458376672118902,
141             -0.004458376672118902, -0.004458376672118902,
142             -0.004458376672118902, -0.004458376672118902,
143             -0.004458376672118902, -0.004458376672118902,
144             -0.004458376672118902, -0.004458376672118902,
145             -0.004458376672118902,
```



```

        -0.004458376672118902, 0.004458376672118902,
        0.004458376672118902, 0.004458376672118902,
        0.004458376672118902
70    }).sorted().toArray(), Arrays.stream(gradients).sorted().toArray(),
        delta);
71    }
72 }

```

Listing 82: nn/NeuralNetworkTest.java

```

1  package optimization;
2
3  import autograd.UniformInitializer;
4  import dataset.*;
5  import jdk.jshell.spi.ExecutionControl;
6  import nn.*;
7  import org.junit.Assert;
8  import org.junit.Ignore;
9  import org.junit.Test;
10
11 import java.io.FileWriter;
12 import java.io.IOException;
13 import java.util.ArrayList;
14 import java.util.Comparator;
15 import java.util.Optional;
16
17 public class GradientDescentTest {
18
19     private final static int trials = 300;
20
21     @Ignore
22     @Test
23     public void TestSimpleGD() throws ExecutionControl.
24         NotImplementedException {
25         var model = Factory.createNeuralNetwork(
26             new int[]{2, 4, 1},
27             new Sigmoid(),
28             new UniformInitializer(-0.5, 0.5)
29         );
30         var dataSet = new XORBinaryDataSet();
31         var optimizer = new GradientDescent(0.2, 0.);
32         var loss = new MeanSquaredError(model.getOutput());
33         double finalLoss = model.fit(dataSet, optimizer, loss, 40000, 0.05);
34         ;
35         Assert.assertTrue("Big loss " + finalLoss, finalLoss < 0.05);
36     }
37
38     @Ignore("Skipping slow convergence tests.")
39     @Test
40     public void TestConvergence() throws ExecutionControl.
41         NotImplementedException, IOException {
42         int diverged = 0;
43         ArrayList<ConvergenceCollector> stats = new ArrayList<>();
44         for (int i = 0; i < GradientDescentTest.trials; i++) {
45             var model = Factory.createNeuralNetwork(
46                 new int[]{2, 4, 1},
47                 new Sigmoid(),
48                 new UniformInitializer(-0.5, 0.5)

```

```

46         );
47         var dataSet = new XORBinaryDataSet();
48         var optimizer = new GradientDescent(0.2, 0.);
49         var loss = new MeanSquaredError(model.getOutput());
50         var collector = new ConvergenceCollector();
51         double finalLoss = model.fit(dataSet, optimizer, loss, 40000,
52             0.05, collector);
53         stats.add(collector);
54         if (finalLoss > 0.05) {
55             diverged += 1;
56         }
57     }
58     outputGraphData("a", stats);
59     Assert.assertTrue("Convergence with high probability busted!",
60         diverged < 6);
61 }
62
63 private void outputGraphData(String assignmentPart, ArrayList<
64     ConvergenceCollector> stats) throws IOException {
65     FileWriter of = new FileWriter("doc/" + assignmentPart + "_avg.tex"
66         );
67     double average = stats.stream().mapToInt(ConvergenceCollector::
68         getEpochs).average().getAsDouble();
69     of.write(String.valueOf(average));
70     of.close();
71
72     Optional<ConvergenceCollector> representative = stats.stream().min(
73         Comparator.comparingDouble(c -> Math.abs(c.getEpochs() - average
74             )));
75     of = new FileWriter("doc/" + assignmentPart + ".tex");
76     of.write(representative.get().toString());
77     of.close();
78 }
79
80 @Ignore("Skipping slow convergence tests.")
81 @Test
82 public void TestBipolarGD() throws ExecutionControl.
83     NotImplementedException, IOException {
84     int diverged = 0;
85     int trials = GradientDescentTest.trials;
86     ArrayList<ConvergenceCollector> stats = new ArrayList<>();
87     for (int i = 0; i < trials; i++) {
88         var model = Factory.createNeuralNetwork(
89             new int[]{2, 4, 1},
90             new BipolarSigmoid(),
91             new UniformInitializer(-0.5, 0.5)
92         );
93         var dataSet = new BinaryToBipolarWrapper(new XORBinaryDataSet())
94             );
95         var optimizer = new GradientDescent(0.2, 0.);
96         var loss = new MeanSquaredError(model.getOutput());
97         var collector = new ConvergenceCollector();
98         double finalLoss = model.fit(dataSet, optimizer, loss, 3500,
99             0.05, collector);
100         if (finalLoss > 0.05) {
101             diverged += 1;
102         }
103         stats.add(collector);
104     }

```

```

94     }
95     outputGraphData("b", stats);
96     Assert.assertTrue("Convergence with high probability busted! " +
97         diverged + " failure out of " + trials, diverged < 6);
98 }
99 @Ignore
100 @Test
101 public void TestBipolarMomentumGD() throws ExecutionControl.
102     NotImplementedException, IOException {
103     int diverged = 0;
104     int trials = GradientDescentTest.trials;
105     ArrayList<ConvergenceCollector> stats = new ArrayList<>();
106     for (int i = 0; i < trials; i++) {
107         var model = Factory.createNeuralNetwork(
108             new int[]{2, 4, 1},
109             new BipolarSigmoid(),
110             new UniformInitializer(-0.5, 0.5)
111         );
112         var dataSet = new BinaryToBipolarWrapper(new XORBinaryDataSet());
113         var optimizer = new GradientDescent(0.2, 0.9);
114         var loss = new MeanSquaredError(model.getOutput());
115         var collector = new ConvergenceCollector();
116         double finalLoss = model.fit(dataSet, optimizer, loss, 1000,
117             0.05, collector);
118         if (finalLoss > 0.05) {
119             diverged += 1;
120         }
121         stats.add(collector);
122     }
123     outputGraphData("c", stats);
124     Assert.assertTrue("Convergence with high probability busted! " +
125         diverged + " failure out of " + trials, diverged < 6);
126 }
127 @Ignore("ReLU not working on XOR or buggy")
128 @Test
129 public void TestBipolarMomentumGDReLU() throws ExecutionControl.
130     NotImplementedException, IOException {
131     int diverged = 0;
132     int trials = GradientDescentTest.trials;
133     ArrayList<ConvergenceCollector> stats = new ArrayList<>();
134     for (int i = 0; i < trials; i++) {
135         var model = Factory.createNeuralNetwork(
136             new int[]{2, 4, 1},
137             new ReLU(),
138             new UniformInitializer(-0.5, 0.5)
139         );
140         var dataSet = new BinaryToBipolarWrapper(new XORBinaryDataSet());
141         var optimizer = new GradientDescent(0.01, 0.9);
142         var loss = new MeanSquaredError(model.getOutput());
143         var collector = new ConvergenceCollector();
144         double finalLoss = model.fit(dataSet, optimizer, loss, 1000,
145             0.05, collector);

```

```

144         if (finalLoss > 0.05) {
145             diverged += 1;
146         }
147         stats.add(collector);
148     }
149     outputGraphData("c", stats);
150     Assert.assertTrue("Convergence with high probability busted! " +
151         diverged + " failure out of " + trials, diverged < 6);
152 }
153
154 @Test
155 public void TestLookupNeuralNet() throws ExecutionControl.
156     NotImplementedException, IOException {
157     var model = Factory.createNeuralNetwork(
158         new int[] {12, 1},
159         new BipolarSigmoid(),
160         new UniformInitializer(-0.5, 0.5),
161         false
162     );
163     double[] input = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1};
164     double desired = 2.1;
165     DataPointDataSet dataPointDataSet = new DataPointDataSet(new
166         DataPoint(input, new double[] {desired}));
167     double toleratedError = 0.01;
168     double lossLimit = Math.pow(toleratedError, 2) / 2;
169     model.fit(dataPointDataSet, new GradientDescent(0.5, 0.), new
170         MeanSquaredError(model.getOutput()), 1000, lossLimit);
171     Assert.assertEquals(model.evaluate(input)[0], desired,
172         toleratedError);
173 }

```

Listing 83: optimization/GradientDescentTest.java

```

1 package rl;
2
3 import org.junit.Ignore;
4 import policy.IPolicy;
5 import representation.*;
6 import org.junit.Assert;
7 import org.junit.Test;
8 import robocode.Robot;
9 import robocode.control.BattleSpecification;
10 import robocode.control.BattlefieldSpecification;
11 import robocode.control.RobocodeEngine;
12 import robocode.control.RobotSpecification;
13 import robocode.control.events.BattleAdaptor;
14 import robocode.control.events.RoundEndedEvent;
15 import robocode.control.events.TurnEndedEvent;
16 import robocode.control.snapshot.IRobotSnapshot;
17 import robot.TrivialLUTRobot;
18 import robot.TrivialLUTRobotConfident;
19
20 import java.io.File;
21 import java.util.ArrayList;
22
23 public class TestQLearning {

```

```

24
25 @Ignore("Focus on testing TNinety")
26 @Test
27 public void TestTrivialLUTRobot() {
28     new File("TrivialLUTRobot.obj").deleteOnExit();
29     Robot opponent = new TrivialLUTRobot();
30     ArrayList<IState> states = new ArrayList<>();
31     TrivialLUTRobotConfident robot = new TrivialLUTRobotConfident();
32     System.setProperty("NOSECURITY", "true");
33     RobocodeEngine.setLogMessagesEnabled(false);
34     RobocodeEngine engine = new RobocodeEngine(new java.io.File(System.
        getProperty("user.home") + "/robocode/"));
35     engine.addBattleListener(new BattleAdaptor() {
36         @Override
37         public void onTurnEnded(TurnEndedEvent event) {
38             super.onTurnEnded(event);
39             for (IRobotSnapshot robotSnapshot: event.getTurnSnapshot().
                getRobots()) {
40 //                 System.out.println(robotSnapshot.getShortName());
41                 if (robotSnapshot.getShortName().equals("
                    TrivialLUTRobotConfident*")) {
42                     states.add(new Coordinates(
43                         (int) (robotSnapshot.getX() / 100),
44                         (int) (robotSnapshot.getY() / 100), 0));
45                 }
46             }
47         }
48     });
49 //     engine.setVisible(true);
50     int numberOfRounds = 100;
51     BattlefieldSpecification battlefield = new BattlefieldSpecification
        (800, 600);
52     RobotSpecification[] selectedRobots = engine.getLocalRepository(
        robot.getClass().getCanonicalName() + "*, " + opponent.getClass()
        .getCanonicalName() + "*");
53     BattleSpecification battleSpec = new BattleSpecification(
        numberOfRounds, battlefield, selectedRobots);
54     engine.runBattle(battleSpec, true); // waits till the battle
        finishes
55     engine.close();
56
57     Assert.assertTrue("Seemingly battle didn't happen. No state is
        collected.", states.size() > 1);
58     IState lastRun = states.get(states.size() - 1);
59     IState firstRun = states.get(0);
60     IPolicy policy = robot.getLearning().getPolicy();
61 //     double initialReward = policy.getReward(firstRun);
62 //     double finalReward = policy.getReward(lastRun);
63 //     Assert.assertTrue(
64 //         String.format("Learning wasn't effective, initial reward
        %f, final reward %f", initialReward, finalReward),
65 //         initialReward <= finalReward);
66     }
67 }

```

Listing 84: rl/TestQLearning.java

```

1 package robot;

```

```

2
3 import autograd.UniformInitializer;
4 import dataset.*;
5 import fa.LUT;
6 import jdk.jshell.spi.ExecutionControl;
7 import nn.*;
8 import optimization.GradientDescent;
9 import org.junit.Assert;
10 import org.junit.Ignore;
11 import org.junit.Test;
12 import representation.Concatenation;
13 import representation.States;
14 import representation.TNinetyAction;
15
16 import java.io.FileOutputStream;
17 import java.io.IOException;
18 import java.io.ObjectOutputStream;
19 import java.util.ArrayList;
20
21 public class TestLUTNN {
22
23     @Ignore
24     @Test
25     public void GridSearch() throws ExecutionControl.
26         NotImplementedException, IOException, ClassNotFoundException {
27
28         LUT lut = new LUT(this.getClass().getClassLoader().getResource("
29             LUTTNinetyRobot.obj").getPath(), true);
30         lut.load();
31         System.out.println(lut.getSize());
32         Assert.assertTrue(false);
33         IDataset dataSet = new LookupTableDataSet(lut);
34
35         for (double momentum :
36             new double[]{0., 0.5, 0.9}) {
37             for (double lr :
38                 new double[] {1e-4, 1e-3, 1e-2}) {
39                 for (int hiddenNeurons :
40                     new int[]{5, 15, 30}) {
41                     var model = Factory.createNeuralNetwork(
42                         new int[]{15, hiddenNeurons, 1},
43                         new BipolarSigmoid(),
44                         new UniformInitializer(-0.05, 0.05),
45                         false
46                     );
47                     var optimizer = new GradientDescent(lr, momentum);
48
49                     var loss = new MeanSquaredError(model.getOutput());
50                     var collector = new ConvergenceCollector();
51                     double finalLoss = model.fit(dataSet, optimizer, loss,
52                         100, 0.00, collector, true);
53                     System.out.println(lr + " " + momentum + " " +
54                         hiddenNeurons + " " + finalLoss);
55
56                 }
57             }
58         }
59     }
60 }

```

```

56  @Ignore
57  @Test
58  public void BestGraph() throws ExecutionControl.NotImplementedException
    , IOException , ClassNotFoundException {
59
60      LUT lut = new LUT(this.getClass().getClassLoader().getResource("
        LUTTNinetyRobot.obj").getPath(), true);
61      lut.load();
62      IDataset dataSet = new LookupTableDataSet(lut);
63
64      var model = Factory.createNeuralNetwork(
65          new int[] {15, 15, 1},
66          new BipolarSigmoid(),
67          new UniformInitializer(-0.05, 0.05),
68          false
69      );
70      var optimizer = new GradientDescent(0.001, 0.5);
71
72      var loss = new MeanSquaredError(model.getOutput());
73      var collector = new ConvergenceCollector();
74      for (int i = 0; i < 30; i++) {
75          double finalLoss = model.fit(dataSet, optimizer, loss, 100,
            0.00, collector, true);
76          System.out.println((100 * i + 100) + " " + Math.sqrt(finalLoss
            / lut.getSize()));
77      }
78  }
79 }

```

Listing 85: robot/TestLUTNN.java

```

1  package robot;
2
3  import fa.IFunctionApproximation;
4  import fa.LUT;
5  import fa.NN;
6  import jdk.jshell.spi.ExecutionControl;
7  import org.junit.Ignore;
8  import org.junit.Test;
9  import representation.IState;
10 import robocode.Robot;
11 import robocode.control.BattleSpecification;
12 import robocode.control.BattlefieldSpecification;
13 import robocode.control.RobocodeEngine;
14 import robocode.control.RobotSpecification;
15 import robocode.control.events.BattleAdaptor;
16 import robocode.control.events.BattleCompletedEvent;
17 import robocode.control.events.RoundEndedEvent;
18 import robocode.control.events.TurnEndedEvent;
19 import robocode.control.snapshot.IRobotSnapshot;
20
21 import java.io.File;
22 import java.io.FileWriter;
23 import java.io.IOException;
24 import java.util.ArrayList;
25 import java.util.Objects;
26
27 public class TestTNinetyRobot {

```

```

28
29 // @Ignore
30 @Test
31 public void TestTrivialLUTRobot() {
32 //     testRobot(new LUTTNinetyRobot0(), 1, 100);
33 //     testRobot(new LUTTNinetyRobotOnline(), 1, 100);
34 //     testRobot(new LUTTNinetyRobotTerminal(), 500, 100);
35 //     testRobot(new LUTTNinetyRobot05(), 100, 100);
36 //     testRobot(new LUTTNinetyRobot(), 500, 2);
37 //     NN functionApproximation = (NN) (new NNTNinetyRobot()).
getLearning().getFunctionApproximation();
38 //     var dataSet = functionApproximation.getDataSet();
39 //     for (int i = 0; i < dataSet.getX().size(); i++) {
40 //         double[] x = dataSet.getX().get(i);
41 //         for (int j = 0; j < x.length; j++) {
42 //             System.out.print(x[j] + " ");
43 //         }
44 //         System.out.println(dataSet.getY().get(i)[0]);
45 //     }
46
47     new File("NNTNinetyRobot.obj").deleteOnExit();
48     new File("NNTNinetyRobot.LUT").deleteOnExit();
49     new File("NNTNinetyRobot.NN").deleteOnExit();
50     testRobot(new NNTNinetyRobotw1g9(), 50, 100);
51 //     testRobot(new NNTNinetyRobotw10g9(), 50, 100);
52 //     testRobot(new NNTNinetyRobotw10g5(), 50, 100);
53 //     testRobot(new NNTNinetyRobotw10g1(), 50, 100);
54 //     testRobot(new NNTNinetyRobotw20g9(), 50, 100);
55 }
56
57 private void testRobot(Robot trainRobot, int rounds, int battles) {
58
59     String outputFileName = "doc/" + trainRobot.getClass().getName() +
        ".tex";
60 //     new File(outputFileName).deleteOnExit();
61     ArrayList<IState> states = new ArrayList<>();
62     NNTNinetyRobotConfident testRobot = new NNTNinetyRobotConfident();
63     Corners opponent = new Corners();
64     System.setProperty("NOSECURITY", "true");
65     RobocodeEngine.setLogMessagesEnabled(false);
66     RobocodeEngine engine = new RobocodeEngine(new File(System.
        getProperty("user.home") + "/robocode/"));
67     engine.addBattleListener(new BattleAdaptor() {
68
69         double energy = 0;
70         double enemyEnergy = 0;
71         int battles = 0;
72         TurnEndedEvent turnEndedEvent;
73
74         @Override
75         public void onTurnEnded(TurnEndedEvent event) {
76             super.onTurnEnded(event);
77             turnEndedEvent = event;
78         }
79
80         @Override
81         public void onRoundEnded(RoundEndedEvent event) {
82             super.onRoundEnded(event);

```



```

83         IRobotSnapshot[] robots = turnEndedEvent.getTurnSnapshot().
            getRobots();
84         for (IRobotSnapshot robot :
85             robots) {
86             if (robot.getName().equals("robot.
NNTNinetyRobotConfident*")) {
87                 energy += robot.getEnergy();
88             } else {
89                 enemyEnergy += robot.getEnergy();
90             }
91         }
92     }
93
94     @Override
95     public void onBattleCompleted(BattleCompletedEvent event) {
96         super.onBattleCompleted(event);
97         boolean shouldPrint = false;
98         try {
99             FileWriter of = new FileWriter(outputFileName, true);
100             for (var result :
101                 event.getIndexedResults()) {
102                 if (Objects.equals(result.getTeamLeaderName(),
testRobot.getClass().getCanonicalName() + "*"))
103                 {
104                     battles += 1;
105                     of.write((battles * rounds) + " " + result.
getFirsts() + " " + (energy / 100) + " " + (
106                         enemyEnergy / 100) + " ");
107                     System.out.print(result.getFirsts() + " ");
108                     shouldPrint = true;
109                 }
110             }
111             if (shouldPrint) {
112                 NN functionApproximation = (NN) (new
NNTNinetyRobotConfident()).getLearning().
getFunctionApproximation();
113                 System.out.println(functionApproximation.getSize()
+ "\n");
114                 double loss = functionApproximation.getLoss();
115                 System.out.println(loss + "\n");
116                 of.write(loss + "\n");
117             }
118             of.close();
119         } catch (IOException | ExecutionControl.
NotImplementedException e) {
120             e.printStackTrace();
121         }
122         energy = 0;
123         enemyEnergy = 0;
124     }
125 }
126
127 for (int i = 0; i < battles; i++) {
128     int numberOfRounds = rounds;
129     BattlefieldSpecification battlefield = new
BattlefieldSpecification(800, 600);
130     Robot robot;
131     if (i % 2 == 0) {
132         engine.setVisible(false);

```

```

130         robot = trainRobot;
131     } else {
132         numberOfRounds = 100;
133         engine.setVisible(false);
134         robot = testRobot;
135     }
136     RobotSpecification[] selectedRobots = engine.getLocalRepository
        (robot.getClass().getCanonicalName() + "*, " + opponent.
        getClass().getCanonicalName() + "*");
137     BattleSpecification battleSpec = new BattleSpecification(
        numberOfRounds, battlefield, selectedRobots);
138     engine.runBattle(battleSpec, true); // waits till the battle
        finishes
139     }
140     engine.close();
141 }
142 }

```

Listing 86: robot/TestTNinetyRobot.java