A Generalizing Agent in a Grid-world

Kai Bartlmae

December 22, 1995

1 Introduction

This paper describes the application of reinforcement learning to a grid-world problem. The agents task is to reach a goal while avoiding to hit into obstacles. The agent can sense only the nearest environment and the direction to the goal. The projects focused on three points. First, how it can be learned to solve the problem in a more general way, second, how a sophisticated function approximator like a basic neural net or a RBF-net can be used and third, how a noisy input-representation changes the performance of the agent.
In the experiment it can be seen that the agent learns the task even if the sensor-input is not correct all the time. The agent also generalizes to a new environment presented to it.

2 The Grid-World Problem

The task of the agent is to reach a goal in a grid-world while avoiding to hit into obstacles spread over this world. The agent has an egocentric view and is able to perform moves in four directions: Forward, backward, left and right. With each move the agent is turned into the direction of its last move (e.g. to go two steps left in a global view he has to perform the action left and forward). The agent can sense neighboring obstacles to the north, south, east and west. Further, the agent knows the general direction to get to the goal.

The obstacles are spread over the grid-world, so that the agent can always decrease the distance to the goal with a particular move.
The agent is dropped randomly into the world with direction up. Its task is now to reach the goal while avoiding obstacles.

Here a 11*11 grid-world with 27 obstacles was used to train the agent:

```
...........
.##..#..#..
.....#.....
##...#..###
..#.....#..
#....G.....
..#.....#..
....#...#..
..###....#.
..#...#....
.....###...
```

The obstacle are marked with a "#", free grids with a "." and the goal with a "G".

In order to test the performance of the policy the agent is tested on a second grid-world:

```
..........#
#####...#..
.......#...
..###.#...#
........#..
###..G....#
........#..
.#...#.####
..#..#.....
..#.###.#.#
....###....
```

3 The algorithm and network architecture

Here *Q-Learning* has been applied with *Temporal Difference Learning*. Different values of reinforcement were applied to each reward. The final values were 0.8 for reaching the goal, -0.01 for making a step and -0.2 for stepping into an obstacle or the border of the grid-world. The value for λ was finally set to 0.0. In order to make exploratory moves, the exploration rate was set to 1.0 and decreased by 0.001 after each trial until 0.05 is reached.

The state-representation consisted out of five values: Two representing the direction to the goal and four representing the sensor output. The directions first value was 1, when the goal was in front and -1 if in the rear. The second value was 1 when the direction to the goal was left and -1 if right from the agent. The sensors output represents the north, west, south, east sensors output. They are 1 if something is in the way and 0 if not.

The actions were coded in two values. The first was 1 for action forward and -1 for backward. Further the second value was 1 for action right and -1 for action left.
Therefore and different from other approaches only one RBF-Net was used to code the Q-Function. This was done to make full use of a possible generalization of the input-representation. So are actions, that a similar in its outcome similar coded.

To represent the Q-function a RBF-Net with 8 input-, 20 hidden and 1 output unit was used.

4 Results

In the experiments the agent was dropped randomly into the world. After 10 learning trails in the training-world the agent was tested: The agent was dropped on each single grid of the training world and its trail was observed, when following the greedy action. For all these test-trails the reward, the number of collisions and the trail-length was measured. If the agent was not able to find the goal, the trail was stopped. After the agent learned the task for the first time (counting from the 2000th trail) in the training-grid-world, its performance was measured in the test-grid-world.

In a first experiment, the agent was able to learn the task with 0 collisions in about 1100 trials. The behavior of the summed collisions over the test-trails can be seen in *figure 1*. The summed trail-length in *figure 2*. After 2000 trails the learned agent was tested on the reference world on each single grid. It showed that the agent was able to reach the goal on each trail with 0 collisions. This means, that the agent learned a policy optimal also for until now not seen states.

Interesting is that the agent learned the task very fast, but after many hundred trials, it was not able to find the goal for some trials, but relearned an optimal policy again. This can be seen in the spikes in *figure 2*.
This could be due to the generalization properties of the RBF Net. The net is not updated for the not chosen non-optimal state-action pairs, since the agent is almost greedy. But generalization still influences the Q-values.

In the second experiment, each sensors error was set to 5 percent. The performance was then calculated in an environment that had the sensor error in order to see, if the agent is able to find a good policy in this environment (*figure 3*). The performance was then also evaluated in an environment with a perfect

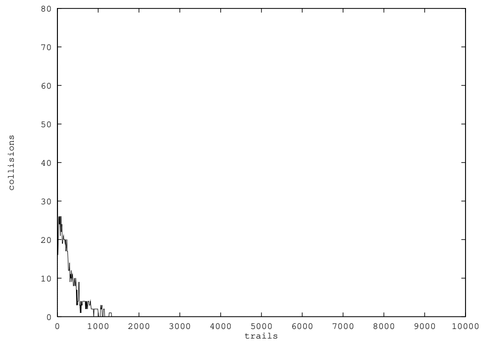

Figure 1: The sum of the collisions over all test-trails on the training-world with sensor error 0.0

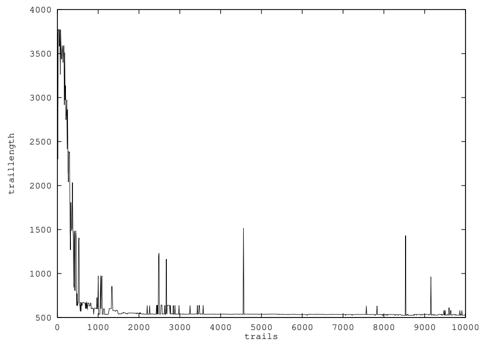

Figure 2: The sum of the traillength over all test-trails on the training-world with sensor error 0.0

sensor, in order to see if the policy is optimal in respect to collisions (*figure 4*). Here, the agent was able to learn the task in 1000 trails according the non-noisy test-trails, but the policy was unstable over the trails.
It can be seen that the agents finds a policy that is able to find the goal with no collisions.
Tested in the second world the agent made 3 collisions.

5 Conclusions

The project had three points to focus on:

First, how to solve a basic grid-world-problem with an agent more general, so that he is able to perform good in a different grid-world and is able to cope with

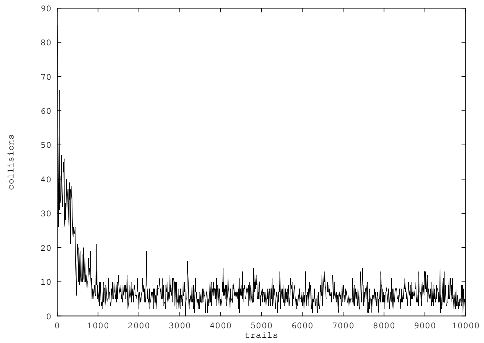

Figure 3: The sum of the collisions over all test-trails with sensor error 0.5 present during training and testing

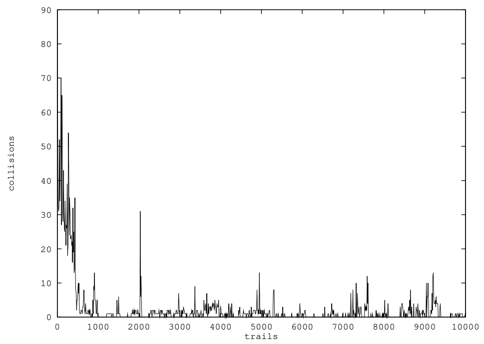

Figure 4: The sum of the collisions over all test-trails with sensor error 0.5 present only during training

different start- and goal-positions. So the agent should not learn an optimal path from a start to a goal, but rather the learning is lifted to learn how to find a path in general. Here it is shown that through an egocentric encoding and and locally generalising function- approximator this is possible. The agent can react even to unseen states and is able to act correctly.

Second, how a function approximator like a neural net or a RBF-net can be used. In the project an RBF-net was used rather than a sigmoid-unit output neural net. It was observed that the neural net had difficulties to converge. This could be due to its global generalization property in an online-learning setting. But even the local RBF-Net forgets an optimal policy and has to be re-learned. Approaches like DYNA, where non-optimal state-action pairs can be simulated, could help to remind the agent on bad actions.

Third, how noise changes the performance of the agent. It can be seen that the policy is still learned, but convergence is less stable than before.

It seems to be important to state the problem in a way like here, where a training-task, maybe a validation task and a testing-task are present, in order to see if the agent really learned what it was intended. This very is important if the task is to find an generalizing agent that behaves already close to optimal in a new environment and therefore has to adopt there only little.

6 References

All we did in class!