Exercise 3: CNN for maze

Furkan Bozkurt, Theresa Eimer

The network we used to solve the task consisted of four Convolutional layers with a MaxPool after every two layers. In the first two convolutional layers, we used 32 2x2 filters with padding to the same size and a pool size of 2. The following convolutional layers used 64 4x4 filters. All of them were ReLu activated and there was a 25% Dropout after each pooling layer. Afterwards there was a dense layer with 512 units, also activated by ReLu. The output layer used was a Softmax layer.
Training was done by using SGD (Nesterov accelerated) with a learning rate of 0.01, a batch size of 32 and a momentum of 0.9 for 20 episodes. In almost all cases the agent found the target very fast and took a short path (if not always the absolute shortest), only in a few cases it didn't find the way and started to oscillate.

Experiment 1 - Adding more history

Increasing the history doesn't change the performance of the network with respect to finding the target a lot as it's already pretty good. However, it shortens the agent's path further (e.g. eliminates small detours) and it now almost identical to the given A* path.

Experiment 2 - Changing the target location

If we change the target more than one or two steps the agent can't find the target at all. If we only move it one step though, the agent can reach it in most cases. That's, of course, because we train the agent on the specific path to the target from which it won't deviate much if at all.

Experiment 3 - Changing the map

Changing the map can make a huge difference or none at all, depending on if the change blocks the A* path the agent wants to take. In that case, it usually doesn't generalize enough to find a new way to the target.

Methods for generalization

We tried less training, less complex networks (less hidden units) and also less history, but in all cases the general performance wasn't good and the generalization didn't improve as we thought. The idea behind this approach was to use the underfitted model because it's worse at following the exact A*

path, so in deviating from it, it might get around new obstacles on the map. We also used more complex networks with multiple fully connected layers, but here we had similar experiences. It seemed that either the network was too well trained on the exact path, meaning it wasn't flexible in finding the target at all, or by not knowing the path too well, it just couldn't get to the target in a lot of cases.
We think the best way to build a generalizing network would be to use Reinforcement Learning, as there we don't just learn the path but an adaptable method to find the target.