Neural Networks for Twist Puzzles (old)

We consider several options for neural networks for twisty puzzles:

1. $NN: S \to A$: approximating the strategy $\pi(s) \in A$
2. $NN: S \to \mathbb{R}$: approximating the state-value function $V(s) \in \mathbb{R}$
3. $NN: S\times A \to \mathbb{R}$: approximating the action-value function $Q(s,a) \in \mathbb{R}$

Option 1: $S \to A$

For bandaged puzzles the possible actions can depend on the current state s. Therefor option (1) would be difficult to use for this generalisation. This option also leads to issues when we haven't explored every possible action in a given state.

Option 2: $S \to \mathbb{R}$

wrong: The network in option (2) knows nothing about possible actions but only evaluates states. However to get training data for this network we would need to know how good a state is. But using Q-Learning we only learn, how good a given action is in a given state. To accurately calculate the state value we would again need to know the values of all possible actions in that state. This is the same problem as in option (1).

correction: To learn state-values, one can use a very similar method to Q-Learning. Having only S as the input space instead of $S\times A$ obviously reduces the number of possible inputs dramatically.

In Q-Learning there are many possible state-action pairs (s_n, a_n) where s' (the result of applying action a_n to state s_n) is the same for all pairs. However all of these pairs can have different values, even though they lead to the same state s'. Intuitively the action that lead to the state s' should not influence it's value in a deterministic environment.

Using V-Learning solves exactly this problem. The value of an action is only determined by the resulting state. This reduces input space size and allows the algorithm to update the value of each state more frequently, improving learning speed.

However this also has drawbacks. In order to evaluate an action a in state s, we need to calculate the resulting state s' of applying a to s. Only after that can we evaluate $V(s')$ as the expected future reward.

In Conclusion we can trade memory requirements and learning speed for slower evaluation. In practice it seems like learning is still faster than using Q-Learning.

Option 3: $S\times A \to \mathbb{R}$

Finally we are left with option (3), approximating the Q-function. Since Q-Learning yields Q-values, we can generate training data very easily. So this is the option we choose.

Neural network architecture

Now that we decided on an approach we can set the number of layers and neurons in those layers.

We want the state and the action as an input. Since it can be difficult for the network to seperate up to 30 discrete actions from just one input neuron, we use one input neuron for each action. That way each action is represented as a vector with one 1 and the rest 0s.

So we get an input size of `len(state) + len(action_space)`.

As an output we only want one number and therefor only need a single Neuron.

Reading several discussions online showed that it is recommended to use more than two hidden layers. Training networks with two or more layers efficiently is supposed to be very difficult and time consuming. Therefor we will try to start with just one hidden layer.

The size of this hidden layer is recommended to be at most twice as large as the input layer.

Experimenting with different network architectures is very important though as it is very difficult to predict how good a given architecture is.

Issues with the training data

When looking at our generated Q-table we noticed that the Q-values are extremely close together. This makes sense as during training many explored states never lead to a solved state and are therefor never associated with a reward.

So there are many states where the only reward they ever got was the penalty for making moves. Those small penalties multiplied with a learning rate and discount factor result in very small changes to the Q-values.

---

training data visualisation

To be able to watch the learning process and evaluate a given network architecture it is useful to plot the loss over time for different architectures.