Optimistic Simulated Exploration as an Incentive for Real Exploration

Ivo DANIHELKA[1]

[1]Dept. of Cybernetics, Czech Technical University, Karlovo náměstí 13, 121 35 Prague 2, Czech Republic

ivo.danihelka@agents.felk.cvut.cz

Abstract. *Many reinforcement learning exploration techniques are overly optimistic and try to explore every state. Such exploration is impossible in environments with the unlimited number of states. I propose to use simulated exploration with an optimistic model to discover promising paths for real exploration. This reduces the needs for the real exploration.*

Keywords

Reinforcement learning, model-based, environment prediction, exploration.

1. Introduction

In reinforcement learning[8] an agent collects rewards in an environment. The environment is not known in advance. The agent has to explore it to learn where to go.

A reward could be received when taking an action in a state. The agent aims to maximize her long-term reward in the environment. She should not miss any state with an important reward or a shorter path to it.

There are many existing exploration techniques[10] that are optimistic in the face of uncertainty. Their optimism assumes that a greater reward will be obtained when taking an unknown action.

The problem is how to do exploration in environments with the unlimited number of states. In these environments, it is not possible to try every action in every possible state.

I study a new way to do exploration in environments with the unlimited number of states. I use simulated exploration as an incentive for real exploration. The simulated exploration proposes promising paths to explore. I describe how to use this kind of exploration in section 2.

My experiments in section 3 demonstrate how simulated exploration reduced the needed amount of real exploration. I also discuss when it is possible.

Many works touched related problems. They inspired me and I discuss them in section 4.
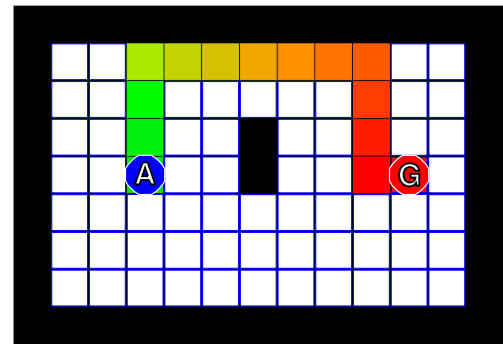
1.1. An Example of the Problem

Fig. 1. A grid world environment with agent *A*, goal state *G* and a suboptimal path from the start to the goal.

There is an example of a reinforcement learning task in Figure 1. The environment is a grid world. The agent has to find a path from the start to the goal. This environment gives a reward just when the goal is reached. The task is then repeated.

A state is represented as a tile on the grid and the agent could choose from four different actions: *North*, *East*, *South*, *West*. But the agent does not know their meanings.

The agent does not know that the environment is a 2D grid world with Manhattan distance metric. She sees it just as a graph and has yet to explore its structure.

A *value* of a state represents the summed expected reward that the agent expects to see if she continues to follow her policy form that state. Q-learning[5] is typically used to learn the values of state-action transitions.

In an environment with the limited number of states, We could use exploration that is optimistic about rewards from unknown actions. This exploration would try to explore every action in every state to see if there is a greater reward. Such wide exploration is impossible if the environment is without boundaries.

2. The Idea

The idea is to use simulated exploration to discover promising paths for real exploration. This reduces the needs for real exploration. An approximate model of the environment is used to simulate the exploration.

2.1. Approximate Models

An approximate model is an approximation of the real environment. It predicts how the environment will behave when executing an action in a state. The model predicts the next state of the environment and what reward will be obtained for the action.

An approximate model could be optimistic or pessimistic. An optimistic model assume a greater reward or better transitions than it is possible in the real environment. For example, an optimistic model could assume no barriers on the path.

It is better to use optimistic models than the pessimistic ones[8]. Using an optimistic model will lead to discovery of a more accurate model of the environment or to discovery of better paths. A pessimistic model would assume that no better path exists and would miss it.

The optimistic model should be as accurate as possible to prevent too many mistakes and corrections. An overly optimistic model would assume that every state transition will lead to a state with the highest reward. A model could aim to be more accurate at the risk to become pessimistic in some states. That is a risk we accept when we don't want to explore all states in unlimited state space.

For example, a model aiming to be more accurate could assume that already tried actions will have similar effects in new states. This does not need to be true when a wall is hit, but that would be corrected if the model is optimistic. An example of one such model is given in section 3.

These less optimistic models may not need to work as an incentive for exploration on their own. A greedy agent sees no reason to try new actions when they don't lead directly to states with a higher value. A simulated exploration is needed to discover the promising paths.

2.2. Simulated Exploration

When given an approximate model, the simulated exploration will try to find promising paths under that model. It tries different actions in the model and explores where they lead. The exploration is not real, it is simulated without taking such actions in the real environment.

To work as an incentive for real exploration, values of environment states continue to be updated during the simula-

tion as if it were a real experience. When an unexplored state leads to a state with a high value, the value of the unexplored state is also increased.

The simulation is done as any other planning method. It is executed to do a few simulated explorations and then it is interleaved by real acting. The following code shows a body of a typical agent. The simulated exploration would be inside the `self.planner.plan()` method, possibly with some other planning.

```python
def agent_step(self, s, r=None):
  """Does a single step and
  returns a selected action.
  The agent sees the current state 's' and
  the reward 'r' obtained for the last action.
  """
  if r is not None:
    self.learner.learn(
        self.last_s, self.last_a, r, s)

  if rl.is_terminal(s):
    return None

  self.planner.plan(s)

  a = self.policy.select_action(s)
  self.last_s = s
  self.last_a = a
  return self.last_a
```

Code 1. Body of a typical agent expressed in Python.

The simulated exploration could discover a promising path that leads to an existing or predicted reward. The promising path would be visible for the agent as a set of states that got a high value. Some of these states could still be unexplored in the real environment.

I will describe two ways how to spread the simulated exploration: *trajectory sampling* and *prioritized sweeping*. But any method that will simulate experience within a given approximate model could be used.

Trajectory Sampling

The simulated exploration could follow a trajectory generated by an exploration policy. The trajectory could start from any state. For example, it could start from the agent's real current state. That restricts the simulated state space to the states near the agent.

Code 2 shows an example of simulated exploration along a trajectory. The maximal depth of the sampled trajectory is limited to limit the amount of computation done inside one planning step.

```python
def plan_along_trajectory(self, s):
  """Simulates exploration of a trajectory
  from the given state.
  """
  path = []
  while len(path) <= self.max_depth:
```

```
  if rl.is_terminal(s):
    break

  a = self.exploration_policy.select_action(s)
  next_s_probs, r = self.model.predict(s, a)
  if len(next_s_probs) == 0:
    break

  path.append((s, a))
  next_s = choose_probable(next_s_probs)
  if next_s == s:
    break
  s = next_s

for s, a in reversed(path):
  self.updater.estimate_q(s, a)
```

Code 2. Simulating exploration by trajectory sampling.

The simulated exploration policy could be completely different from the policy used for acting in the real environment. The simulated exploration policy is just used to sample states from the state space. The approximate model is used to estimate values of the sampled states.

Prioritized Sweeping

Prioritized sweeping[11] schedules updates of state-action values when one of their children changed its value. A priority queue is typically used for that. The amount of change in a child serves as a priority to process its parents.

Prioritized sweeping could be used to simulate exploration on its own or in combination with other methods. When used on its own, it is needed to have an approximate model that is able to return parents of a state. Related unexplored states have to be also returned as possible parents.

The depth of sweeping of unexplored states is limited by setting a minimal considered priority. The states where the amount of change is below this threshold are not swept. This will work unless the approximate model repeatedly predicts a reward in new unexplored states.

When the model does not give information about parents, it is still possible to use prioritized sweeping in combination with another simulated exploration. It is just needed to correct the remembered parents when the optimistic model is corrected. Otherwise a wrong parent would continue to be updated by optimistic values.

Remembered parents could be easily corrected when the used model produces a distribution of all possible next states. These next states are all possible children of a parent and its old children could be discarded.

3. Experiments

The experiments test how simulated exploration reduces the amount of needed real exploration. It is assumed that a suitable optimistic model of the environment could be used.

3.1. The Task

In my experiments, I used the 3277-state grid world mentioned inside the *Reinforcement Learning: A Survey*[5]. Figure 2 shows the used environment. The environment is fully deterministic.

I specified an initial suboptimal path inside the grid world. The suboptimal path gives a hint where the reward is in the unlimited state space. This allows to use the simulated exploration without optimistic assumptions about the reward.
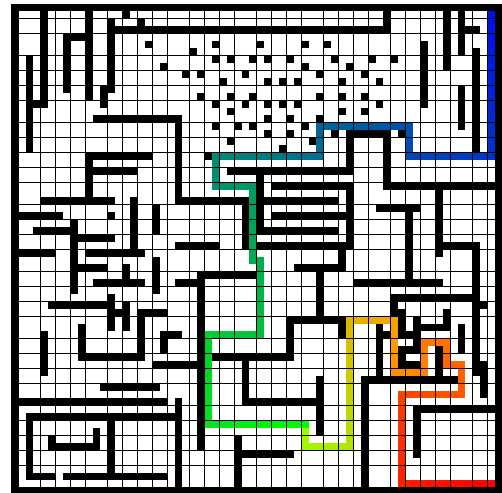
Fig. 2. A grid world used for experiments. It contains an initial suboptimal path from the start to the goal.

The start position of the agent is fixed to the top-right corner of the grid world. The aim to reduce the amount of exploration is most relevant when the start position is fixed. In that case, the state space used for acting is smaller than the whole available state space.

3.2. The Used Model

The used approximate model was composed as a combination two models: model based on observations and a model based on recent action effects.

The **model based on observations** is a classical model that records frequencies of different outcomes for every state-action transition. The frequencies are used to estimate

the probabilities of the outcomes. This model does not predict anything about yet unexplored states.

The **model based on recent action effects** records the last seen effects of each action. It then predicts that a seen action will have the same effects in a new state. It also tries to be optimistic and ignores when an action does not have any effects on the state. That could happen when a wall is hit. So it does not remember any wall. The following code shows a code of such a model.

```
class RecentEffectBasedModel:
  def __init__(self, actions, initial_r):
    self.actions = actions
    self.initial_r = initial_r
    self.increments = [None] * len(actions)

  def predict(self, s, a):
    """Predicts probabilities of next states
    and the returned reward.
    """
    a_index = self.actions.index(a)
    increment = self.increments[a_index]
    if increment is None:
      return [], 0

    return [(s + increment, 1.0)], self.initial_r

  def learn(self, s, a, r, next_s):
    """Remembers state changes from experience.
    It remembers the last seen increment
    for the given action.
    """
    if rl.is_terminal(next_s):
      return

    # Walls are not learned by this model.
    # It is kept optimistic for the grid world.
    increment = next_s - s
    if increment != 0:
      a_index = self.actions.index(a)
      self.increments[a_index] = increment
```

Code 3. A model that tries to predict state changes.

The above shown code assumes that a state is represented by a single numeric variable. But an analogous approach could be used when the state is represented by a vector of multiple variables[3].

The used initial suboptimal path should use every action at least once. That allows the model based on action effects to predict unexplored states in all directions. An alternative would be to do an additional small amount of exploration at the beginning of the task.

The two used models provide different predictions about state transitions and rewards. The model based on observations predicts just already seen transitions and rewards. The model based on recent action effects tries to predict future state transitions and assumes no extra reward from them. The prediction based on recent action effects is wrong when a wall is hit, so it is considered to be less accurate.

These two models are combined together to compose a final approximate model. The final model uses the most

accurate prediction available. It asks the models for a prediction starting from the most accurate model. When the asked model knows nothing about the given transition, a less accurate model is asked. The code of the combined model follows.

```
class CombinedModel:
  def __init__(self, models):
    """Accepts a given list of models
    that have decreasing accuracy.
    """
    self.models = models

  def predict(self, s, a):
    """Returns the most accurate
    available prediction
    for the given transition.
    """
    for model in self.models:
      next_s_probs, r = model.predict(s, a)
      if len(next_s_probs) > 0:
        return next_s_probs, r
    return [], 0

  def learn(self, s, a, r, next_s):
    for model in self.models:
      model.learn(s, a, r, next_s)
```

Code 4. A model that combines multiple approximate models.

3.3. Used Algorithms

I compare exploration with optimistic initial values with two simulated explorations: trajectory sampling and prioritized sweeping of unexplored states.

The exploration with **optimistic initial values** assumes value 1.0 inside every state-action transition. This strategy is incentive enough to find an optimal policy in a deterministic environment.

The initial state-action values for the other exploration algorithms were set to 10^{-32}. Such a low value does not serve as an incentive for exploration, but it remains possible to decrease it. It is important to be able to decrease the value of tried state-action transitions. It prevents the agent to get stuck.

The simulated **trajectory sampling** was tested with three different maximal depths: 3, 6 and 12. The used simulated exploration policy selects a random action for every step. The sampled trajectory always starts from the current agent's state. The simulated exploration tries 10 trajectories per each real step.

The prioritized **sweeping of unexplored** states uses parents supplied by the approximate model of the environment. Our approximate model consists of multiple models, so it has to combine the parents predicted by the different models. I do this by letting the least accurate model predict some parents. The more accurate models are then used to prune the list of possible parents. The second least accurate

model then predicts some other parents. The algorithm is depicted by Code 5.

```
def get_parents(self, s):
  """Returns possible parents of the given state.
  """
  parents = set()
  for i, model in reversed(list(
        enumerate(self.models))):
    subparents = model.get_parents(s)
    for parent in subparents:
      if is_possible_transition(parent, s,
          self.models[:i]):
        parents.add(parent)

  return parents
```

Code 5. A method to return combined parents from multiple models. The used models have decreasing accuracy.

All mentioned explorations were aided with prioritized sweeping of explored states. I implemented the improved prioritized sweeping algorithm of Wiering and Schmidhuber[11].

3.4. Results

I measured the amount of exploration done under the different exploration algorithms. The exploration is finished when the shortest path is found and used for subsequent episodes. The initial path was 217 steps long. The shortest path was 203 steps long and all used algorithms were able to find it.

The numbers of explored states and state-actions are given in table 1.

	Steps	States	State-actions
Optimistic initial values	19,673	3,201	12,797
Trajectory sampling (3)	2,830	523	902
Trajectory sampling (6)	4,967	932	1,864
Trajectory sampling (12)	5,291	1,695	3,668
Sweeping of unexplored	6,372	1,580	3,536

Tab. 1. The performance of different explorations. The columns report the number of steps until finding the final policy and the numbers of explored states and state-actions. The trajectory sampling used maximal depths 3, 6 and 12.

It is interesting to note the relation between the number of explored states and the number of explored state-actions. The exploration equipped with simulated exploration tried on average about two actions in every visited state. On the other hand, the exploration with optimistic initial values tried all four actions in almost every visited state.

3.5. Discussion

All measured simulated explorations had lower amount of exploration than the overly optimistic exploration with optimistic initial values.

The trajectory sampling required the lowest amount of exploration to find the shortest path. But it does not guarantee that it will find the shortest path. The maximal depth of the sampled trajectory limits the space where to search for promising paths. That limits the amount of exploration done, but it also allows to miss an optimal path.

The prioritized sweeping of unexplored states performed very well. It required to explore 3.6 times less state-actions than the exploration with optimistic initial values. And the sweeping of unexplored states guarantees to find any possible promising path that exists under the used approximate model.

The lower amount of real exploration was possible thanks to simulated exploration in an approximate optimistic model. These methods are not suitable for environments where it is impossible to learn an optimistic model of unexplored states of the environments. These methods risk that they will miss the optimal path when a pessimistic model is used.

The used approximate model does not predict any unexplored reward. It was needed to start with an initial suboptimal path to show some reward. More sophisticated approximate models could also try to predict the reward.

4. Related Work

The using of an observed model for planning was pioneered by the Dyna architecture by Sutton[7]. The Dyna planning continues to update seen transitions with changes in the estimated state-action value. Its extensions provide prioritized sweeping[11] and a usage with linear function approximation to represent the environment[9].

The idea to use optimistic models came from the book *Reinforcement Learning: An Introduction*[8]. It discusses how optimistic models do not miss a promising path. It also proposes to use trajectory sampling on large tasks.

Many works touched the problem of exploration in large spaces. Smart and Kaelbling[6] reduced the amount of exploration by using initial knowledge. They supplied the agent with example trajectories. During these trajectories, the agent was driven by a human operator or by a piece of code. The example trajectories do not needed to be optimal. They just have to give hints where the reward is.

Envelope methods[4] limit the amount of planning by restricting the state space of a known environment. They do not need to explore, because the environment is fully known, but they still needed to limit the number of states to consider.

Apprenticeship learning[1] aims to prevent destructive exploration. A teacher first demonstrates the task. That demonstration serves to learn an approximate model of environment. An optimal policy is learned off-line in the model and tested later in the real environment. The new experience serves to improve the model and the cycle continues. The apprenticeship learning is not intended to propose new promising paths, so unexplored states are not considered in the model.

An extension to the apprenticeship learning uses the learned approximate model to find a policy improvement direction[2]. It could be viewed as a search for a promising direction where to steer the policy parameters. The needed amount of steering is then tested in the real environment.

5. Conclusions

I proposed a new way how to do exploration in unlimited state spaces. It uses a simulated exploration and an optimistic model of the environment.

It remains to be discovered how to learn optimistic models for a wide range of environments. They should provide optimistic predictions about unexplored states, but their optimism should be as low as possible.

My experiments show the possibility to reduce the real exploration when given an optimistic model.

Acknowledgements

References

[1] ABBEEL, P., NG, A.Y. Exploration and apprenticeship learning in reinforcement learning. *Proceedings of the 22nd International Conference on Machine Learning* 2005, p. 1 - 8.

[2] ABBEEL, P., QUIGLEY, M., NG, A. Y. Using inaccurate models in reinforcement learning. *Proceedings of the 23rd International Conference on Machine Learning*, 2006, p. 1 - 8.

[3] BOUTILIER, C., DEAN, T., HANKS, S. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 1999, vol. 11, p. 1 - 94.

[4] DEAN, T., KAELBLING, L. P., KIRMAN, J., NICHOLSON, A. Planning under time constraints in stochastic domains. *Artificial Intelligence*, 1995, vol. 76, no. 1 - 2, p. 35 - 74.

[5] KAELBLING, L. P., LITTMAN, M. L., MOORE, A. W. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 1996, vol. 4, p. 237 - 285.

[6] SMART, W. D., KAELBLING, L. P. Practical reinforcement learning in continuous spaces. *Proceedings of the Seventeenth International Conference on Machine Learning*, 2000, p. 903 - 910.

[7] SUTTON, R. S. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. *Proceedings of the Seventh International Conference on Machine Learning*, 1990, p. 216 - 224.

[8] SUTTON, R. S., BARTO, A. G. *Reinforcement learning: An introduction*, MIT Press, 1998.

[9] SUTTON, R. S., SZEPESVARI, C., GERAMIFARD, A., BOWLING, M. Dyna-style planning with linear function approximation and prioritized sweeping. *Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence*, 2008, p. 528 - 536.

[10] SZITA, I., LŐRINCZ, A. The many faces of optimism: a unifying approach. *Proceedings of the 25th International Conference on Machine Learning*, 2008, p. 1048 - 1055.

[11] WIERING, M. A., SCHMIDHUBER, J. Efficient model-based exploration. *Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, 1998, p. 223 - 228.

About Authors...

Ivo DANIHELKA is a PhD student at the Department of Cybernetics at CTU in Prague. Previously, he was a software engineer working on billing systems, TV over the Internet, games, data mining and multiple web sites. He still likes to read well written source code.