Autonomous Task Sequencing for Customized Curriculum Design in Reinforcement Learning

Sanmit Narvekar, Jivko Sinapov, and Peter Stone
Department of Computer Science
University of Texas at Austin
{sanmit, jsinapov, pstone} @cs.utexas.edu
Successes of Reinforcement Learning

Approaching or passing human level performance

BUT

Can take *millions* of episodes! People learn this **MUCH** faster
People Learn via Curricula

People are able to learn a lot of complex tasks very efficiently
Example: Quick Chess

- Quickly learn the fundamentals of chess
- 5 x 6 board
- Fewer pieces per type
- No castling
- No en-passant
Example: Quick Chess

University of Texas at Austin
Sanmit Narvekar
Quick Chess is a curriculum designed for people.

We want to do something similar automatically for autonomous agents.
Curriculum Learning

Task Creation
Presented at AAMAS ‘16

Sequencing

Transfer Learning
via Value Function Transfer

- Curriculum learning is a complex problem that ties task creation, sequencing, and transfer learning
Autonomous Task Sequencing
Sequencing as an MDP

- **State space** $S^c$: All policies $\pi_i$ an agent can represent
- **Action space** $A^c$: Different tasks $M_j$ an agent can train on
- **Transition function** $p^c(s^c, a^c)$: Learning task $a^c$ transforms an agent’s policy $s^c$
- **Reward function** $r^c(s^c, a^c)$: Cost in time steps to learn task $a^c$ given policy $s^c$
Sequencing as an MDP

A policy $\pi^C: S^C \rightarrow A^C$ on this curriculum MDP (CMDP) specifies which task to train on given learning agent policy $\pi_i$.

Learning full policy $\pi^C$ can be difficult!

- Taking an action requires solving a full task MDP
- Transitions are not deterministic
Sequencing as an MDP

• Instead, find one trace/execution in CMDP of $\pi^C$.

• **Main Idea**: Leverage fact that we know the target task and therefore what is relevant for the final state policy $\pi_f$ to guide selection of tasks.
Autonomous Sequencing

- Grid world domain

- Objectives
  - Navigate the world
  - Pick up keys
  - Unlock locks
  - Avoid pits
Autonomous Sequencing

• Recursive algorithm (6 steps)

• Each iteration adds a source task to the curriculum

• This in turn updates the policy

• Terminates when performance on target task greater than desired performance threshold
Step 1

• Assume learning budget $\beta$

• Attempt to **solve** target task directly in $\beta$ steps. Save samples

• Solvable?
  • Target task **easy to learn**
  • Started with policy that made it easy to learn. Done

• Goal: **incrementally** learn subtasks to **build a policy** that can learn the target task
Autonomous Sequencing

Step 2
- Could not solve target
- Create source tasks using methods from AAMAS ‘16.

Step 3
- Attempt to solve each source in $\beta$ steps
- Partition sources into solvable / unsolvable
Autonomous Sequencing

Step 4

- If solvable tasks exist, select the one that updates the policy the most on samples drawn from the target task

Assumption
- Source tasks that can be solved have policies that are relevant to the target task
- Don’t provide negative transfer
Autonomous Sequencing

Step 4 (cont.)

• Add source task to curriculum
• Return to Step 1

• (Re-evaluate on target task)
• Policy has changed, so we will get a new set of samples
• Samples biased towards agent’s current set of experiences
• This in turn guides selection of source tasks
Autonomous Sequencing

Step 5

- No sources solvable
- Sort tasks by sample relevance
  - Compare states experienced in target task with those in experienced in sources
- Recursively create sub-source tasks
- Return to Step 2 with the current source task as the target task
Autonomous Sequencing

**Step 6**

- **No sources usable** after exhausting the tree
- **Increase budget**, return to Step 1
- **Learning can be cached**, so agent can pick up where it left off

University of Texas at Austin
Sanmit Narvekar
Connection to CMDPs

- An **optimal path** in CMDP is one that reaches $\pi_f$ with least cost
- Selection in Step 4 picks tasks that update most towards $\pi_f$
- Learning budget **minimizes cost**
- Algorithm behaves **greedily** to balance updates and cost
Experimental Setup

• Grid world domain presented previously

Create multiple agents

• Multiple agents shows the algorithm is not dependent on implementation of RL agent
• Evaluate whether different agents benefit from individualized curricula
Experimental Setup

Agent Types

• Basic Agent
  • **State**: Sensors on 4 sides that measure distance to keys, locks, etc.
  • **Actions**: Move in 4 directions, pickup key, unlock lock

• Action-dependent Agent
  • State difference: *weights* on features are *shared* over 4 directions

• Rope Agent
  • Action difference: Like basic, but can use *rope action* to negate a pit
Basic Agent Results

![Graph showing the performance of different curriculums over game steps.]

- **No curriculum**
- **Basic curriculum**
- **Action dependent curriculum**
- **Rope curriculum**
Action-Dependent Agent Results

![Graph showing the return over game steps for different curriculum types.](image)

- **no curriculum**
- **action dependent curriculum**
- **basic curriculum**
- **rope curriculum**

---

University of Texas at Austin
Sanmit Narvekar

24
Rope Agent Results

The chart shows the performance of different curriculum strategies in a game. The x-axis represents the number of game steps, while the y-axis shows the return. Different lines represent different curriculum strategies:

- **no curriculum**
- **rope curriculum**
- **basic curriculum**
- **action dependent curriculum**
- **random curriculum**

As the game steps increase, the return values improve for all strategies, with the **rope curriculum** showing the most consistent and highest performance.
Summary

- Presented a novel formulation of curriculum generation as an MDP
- Proposed an algorithm to approximate a trace in this MDP
- Demonstrated method proposed can create curricula tailored to sensing and action capabilities of agents