Curriculum Distillation to Teach Playing Atari

Chen Tang
John F. Canny

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2018-161
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2018/EECS-2018-161.html

December 1, 2018

Acknowledgement

I would first like to thank my advisor Professor John Canny for guiding and mentoring me through research. His patience and insightful ideas created an incredible research experience for me. I am incredibly grateful for the opportunity to work with him.

Thank you to all my friends who have helped me during my past five years at Berkeley. I am very grateful for all the memories we shared. Finally, I would like to thank my family for their continual support throughout my years at Berkeley.

Curriculum Distillation to Teach Playing Atari

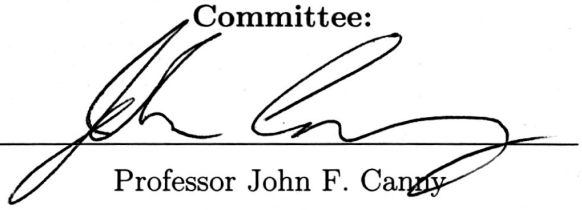
by Chen Tang

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:



Professor John F. Canny
Research Advisor

May 16, 2018

(Date)

****

Professor Sergey Levine
Second Reader

May 16, 2018

(Date)

Abstract

Curriculum Distillation to Teach Playing Atari

by

Chen Tang

Master of Science in Electrical Engineering and Computer Science

University of California, Berkeley

Professor John F. Canny, Chair

We propose a framework of curriculum distillation in the setting of deep reinforcement learning. By selecting samples in its training history, a machine teacher sends those samples to a learner to improve its learning progress. In this paper, we investigate the idea on how to select these samples to maximize learner's progress. One key idea is to apply the Zone of Proximal Development principle to guide the learner with samples slightly in advance of its current performance level. Another idea is to use the samples where teacher itself makes the biggest progress in its parameter space. To foster robust teaching and learning, we adapt such framework to distill curriculum from multiple teachers. We test such framework on a few Atari games. We show that those samples selected are both interpretable for humans, and are able to help machine learners converge faster in the training process.

Contents

Chapter 1

Introduction

Deep reinforcement learning has shown its great capacity in learning how to act in complex environments. We have seen successes in various domains, such as playing Atari Games [27] and AlphaGo Zero [37] beating human Go players. Reinforcement learning is a type of machine learning where agents learn from the data and experience.

In addition to learning, machine is also able to teach. Machine teaching is the inverse problem of machine learning [44]. It is the process that trained agents use their "knowledge" to help the next generations of learners. The goal is to make teacher more productive at helping learners learn [38]. The goal of machine teaching is not to train another models, but to generate the most effective examples that can easily transfer the knowledge to learners.

Learners can be either human learners or machine learners. There's a great amount of applications in machine teaching. When the learners are human learners, it is an application of personalized education. Having assumptions of learners' cognitive models, the machine teacher is able to optimize finding examples for each individual learner to master the knowledge with various backgrounds [18, 32]. When the learners are machine learners, it can be a case of distilling knowledge from multiple machine teachers [17]. Other applications include data poisoning attacks, which can also be modeled as machine teaching [24].

In the setting of reinforcement learning, those samples from teachers must be sequential. One research direction is curriculum learning [5] where teachers design a curriculum for the learner based on learner's recent states. Similar to human learning, teachers usually start from simple examples and gradually move to harder ones. If teacher knows the knowledge of student's full learning algorithms and parameters, it is model-based teaching process [45].

In this paper, we propose a framework of curriculum learning with partial knowledge of the learner. In this framework, we have well-trained agents, called teacher $\{T_1, \ldots, T_N\}$,

who as a whole send selected samples to a learner L throughout learner's training process. Teachers are only aware of the learning algorithm, or more specifically the loss function of the learner. The learner, at the same time, is aware of the teaching sending samples and needs to incorporate those samples in its training. Instead of using optimization techniques to find those samples, we distill such curriculum directly from teachers' own training histories. We rely on the intuition that samples which help teachers move closer to the optimal parameters also help the learner. Our method is also based on the human teaching methodology and cognitive models, the principle of *Zone of Proximal Development* (ZPD) [22]. ZPD principle says that learner makes the most learning progress if the teachers give examples slightly beyond the learner's current level. By applying such framework to teach playing Atari game, we use those selected samples to interpret the training process of deep reinforcement learning. In addition, we also show that the curriculum consisted of those samples can expedite learner's training progress.

In Chapter 2, we review background in reinforcement learning and cognitive models used in teaching. In Chapter 3, we present related work in machine teaching and deep reinforcement learning. In Chapter 4, we illustrate the framework of curriculum distillation under ZPD principle, and propose methods to find the most optimal samples for a teacher to send to a learner. In Chapter 5, we visualize those pedagogical samples and show the training curve under such curriculum generation framework in Atari games. Finally in Chapter 6, we discuss the potential future research problems.

Chapter 2

Background

2.1 Reinforcement Learning

Reinforcement learning is a popular way to solve a Markov Decision Process (MDP) [40]. A MDP is usually defined as a tuple $\langle S, A, R, T, \gamma \rangle$ where S is the state space, A is the action space, $R(s, a)$ is the reward function, $T(s, a, s')$ is the transition probability, and γ is the reward discount. When the agent takes an action $a \in A$ in the state $s \in S$, the environment generates a reward $R(s, a)$ and goes to the next state $s' \in S$ with probability $T(s, a, s') = \mathbb{P}(s'|s, a)$. Reinforcement learning is to find a policy $\pi : S \mapsto A$, a function mapping a state to an action. Here, we consider the action space A is discrete and a stochastic policy $\pi(a|s)$, where π maps to a probability distribution of A conditioning on s.

Consider under the policy π, agent has the trajectory $\{(s_1, a_1, r_1, s'_1), \ldots, (s_T, a_T, r_T, s'_T)\}$ where each tuple represents a transition in the trajectory. Agent receives a reward r_t at time-stamp t. It is common to assume that future rewards are discounted by a factor of γ per time-stamp. Therefore, the goal for reinforcement learning is to find the best policy π^* such that it maximizes the expected discounted future rewards throughout agent's lifetime:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=1}^{T} \gamma^{t-1} r_t \middle| \pi\right].$$

The function $Q : (S, A) \mapsto R$ maps a state-action pair to a real value called Q-value. $Q^\pi(s, a)$ denotes the expected discounted future reward received by an agent who takes action a_{t_0} in state s_{t_0} and continues to act according to policy π. The optimal Q-function is found to maximize such reward over policy π:

$$Q^*(s_{t_0}, a_{t_0}) = \max_{\pi} \mathbb{E}\left[\sum_{t=t_0}^{T} \gamma^{t-t_0} r_t \middle| s_{t_0}, a_{t_0}, \pi\right].$$

The optimal Q-function satisfies Bellman equation where

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathbb{P}(s'|s,a)} \left[R(s,a) + \gamma \max_{a'} Q^*(s',a') \right]. \tag{1}$$

Intuitively, it says that if the optimal Q-values are known one step ahead, the optimal policy is simply maximizing over Q-values with one step discount $R(s,a) + \gamma Q^*(s',a')$. One classic way to solve such problem is to use value iteration [4], which uses iterative updates and dynamic programming to solve for Equation 1. Those methods make Q-function converge to the optimal as the number of iterations goes to infinity.

2.2 Deep Q-Learning

A common way to model those mapping functions is to use function approximators. Q-function can be parameterized into a function $Q_\theta(s,a)$. Then it can be estimated using any parametric model such as least squares or neural networks. The problem becomes an optimization problem over the parameter space $\theta \in \mathbb{R}^d$. Deep Q-Network (DQN) [27] is using a neural network to optimize Q-values based on Bellman equation in Equation 1.

Given a state s, it uses $Q_\theta(s,a)$ to select the best action a. The environment gives a reward r and transits to the next state s'. Such process yields a *sample transition* $\langle s,a,r,s' \rangle$. The loss function for one sample transition $\langle s,a,r,s' \rangle$ is

$$L_\theta(s,a,r,s') = \left[\underbrace{Q_\theta(s,a)}_{\text{Estimated } Q\text{-value}} - \underbrace{\left(r + \gamma \max_{a'} Q_{\theta^-}(s',a') \right)}_{\text{Target } Q\text{-value}} \right]^2. \tag{2}$$

Different from supervised learning using deep neural networks, the target value in reinforcement learning is also affected by θ. Therefore in order to make stable update to the network, we use a target network θ^- to compute the target Q-values in Equation 2. θ^- is a snapshot of θ, held fix for a period of time when computing the target Q-values, and synchronized with θ every T_{sync} steps.

Another key idea in DQN is that it uses a replay buffer D to store all such transitions. The buffer is essentially a queue that follows First-In-First-Out (FIFO) principle. During the training time, the algorithm samples uniformly from the replay buffer to form a minibatch, and uses backpropagation algorithm [13] to minimize the loss of the minibatch according to Equation 2.

Algorithm 1 shows the complete algorithm. The original paper [27] shows huge success in applying deep Q-learning to play Atari games, reaching or exceeding human-level

performance on 29 of the 46 games.

Algorithm 1 Deep Q-Learning

initialize a replay buffer D, environment state s, and network parameter θ, θ^- where $\theta^- = \theta$
foreach steps $t = 1, 2, \ldots$ **do**
  Compute $Q_\theta(s_t, a)$ of the current state s_t and all actions $a \in A$.
  Select the best action $a_t = \arg\max_a Q_\theta(s_t, a)$.
  With probability ϵ, replace action a_t with a random action. ▷ ϵ-greedy
  Execute a_t in the environment which generates a reward r_t and next state s_{t+1}.
  Add the sample transition $\langle s_t, a_t, r_t, s_{t+1} \rangle$ in the replay buffer D.
  if the current episode is complete **then**
    Reset the environment to s_{t+1}.
  end if
  if $t \mod T_{play} = 0$ and $|D| \geq D_{start}$ **then**
    Sample a minibatch B uniformly from the replay buffer D.
    Compute target Q_{θ^-} values y_i, for each sample $\langle s_i, a_i, r_i, s'_i \rangle$ in the minibatch:

$$y_i = \begin{cases} r_i & s'_i \text{ is a terminal state} \\ r_i + \gamma \max_{a'} Q_{\theta^-}(s'_i, a') & s'_i \text{ is a non-terminal state} \end{cases}.$$

    Compute the average loss for the minibatch $L = \frac{1}{n} \sum_B \left(y_i - Q_\theta(s_i, a_i)\right)^2$.
    Perform a gradient descent step on θ.
  end if
  if $t \mod T_{sync} = 0$ **then**
    Update $\theta^- = \theta$.
  end if
end for

Following DQN, a series of follow-up papers [36, 31, 41, 42, 11, 2] algorithmically improve the training convergence speed. THe original DQN suffers from overestimation bias from the max operator in Equation 2. Double DQN (DDQN) [41] is the extension to overcome such issue. It decouples max selector and action selector and ends up with the new loss:

$$L_\theta^{DDQN}(s, a, r, s') = \left[Q_\theta(s, a) - \left(r + \gamma \underbrace{Q_{\theta^-}\left(s', \overbrace{\arg\max_{a'} Q_\theta(s', a')}^{\text{action selector using } \theta} \right)}_{Q\text{-value evaluation using } \theta^-} \right) \right]^2. \quad (3)$$

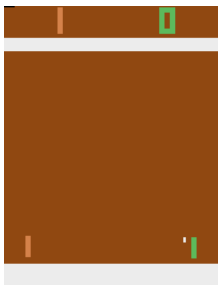
The paper [41] shows improvement in performance in most of the Atari games.

2.3 Atari 2600

The training and testing environment of this paper is the Arcade Learning Environment (ALE) [3] which can be used to play all Atari 2600 games. OpenAI provides a wrapper *gym* [6] on top of ALE. In order to make DQN perform the best, it is necessary to make the following necessary preprocessings [27] to the input frames:

- *Image Down-sampling.* The raw frame from Atari environment is a 210×160 colored pixel image, and we down-sample it to 84×84 gray-scale image.
- *Reward Clipping.* In order to have the same learning rate across all games, we clip all positive rewards to be 1 and all negative rewards to be -1.
- *Episodic Life.* In order to help faster convergence in value estimation, we set the end of each life in the game as the end of an episode. However, the game is only reset when an episode is truly over.
- *Frame Stacking.* We stack $k = 4$ last frames in the history together to generate input states.
- *Frame Skip.* In order to save computation, we repeat every selected action for $k = 4$ consecutive frames and only return the last frame to the agent.
- *Random Initialization.* When resetting the environment, we initialize the environment by taking $n \in \{0, \ldots, 30\}$ random actions in the environment.

After preprocessings, the state space has dimension $4 \times 84 \times 84$. In this paper, we will look closely at one Atari game, Pong, shown in Figure 1. Pong is usually the easiest game to play as the key to winning is very simple.



(a) Pong

Figure 1: Sample frames from an Atari game - Pong.

Pong is a "contact-seeking" game where the player needs to catch the "ball" (white dot) with the "paddle" (the green bar on the right of the screen). The left red "paddle" is controlled by the opponent and the player controls the green one. The player gets

1 point when the opponent misses a "ball" and -1 point when itself loses a "ball". The episode ends when either player catches 11 points. The total reward for each episode is between -21 and 21.

2.4 Zone of Proximal Development

The concept of Zone of Prxomial Development is a psychology concept built on human cognitive model, first proposed by Lev Vygotsky [22]. It states that when a child is following an adult's examples, it is able to gradually develop the skills without assistance. Those examples need to be in the Zone of Proximal Development (ZOP), which is the distance between a child's development level under self-learning and under education from adult's examples [8]. In other words, if teaching is within ZOP, it is beneficial for students to advance in their learning progress.

Vygotsky's ZOD principle inspired another theory called scaffolding, first proposed by Jerome Bruner [30]. It says that when the teacher guides student's learning, it should gives examples only as necessary and tapers off such guidance over the time. Although such frameworks were originally developed for children learning, they have been shown applicable in general human learning. Overall speaking, under such principles, teachers educate students with examples slightly beyond their current development level, and reduces such guidance throughout the education process.

Chapter 3

Related Work

3.1 Reinforcement Learning with Imitation

Under our curriculum learning framework, the learner takes samples from teachers during training. Thus one area of related work is imitation learning where the agent learns from demonstration data. The setup is similar to supervised learning, imposing a classification loss between the predicted action distribution and the true action in the demonstration data. However, one of the common problem is that the distribution of the demonstration data can be different from the distribution of the states that agents face. DAGGER [34] is one method to use data aggregation to mitigate such issue.

There are a few related work on integrating imitation learning with reinforcement learning. A paper from DeepMind [16] introduces using expert's demonstration data in Deep Q-learning training process. In addition to the Double DQN loss in Equation 3, it also introduces three auxiliary loss: an n-step loss, a regularization loss, and a classification margin loss for experts' samples. The authors argue that imposing such supervised loss pushes the Q-values of the actions picked by the expert above those of other actions.

In addition to such classification margin loss, there are a few other supervised loss in the literatures. Cross entropy loss in policy gradients methods is a very common one [29, 9]. Similarly in soft Q learning, a loss on the entropy of the action distribution can be used [12]. The authors claim that such method is able to learn from imperfect demonstration data.

Our paper builds on top of these existing work, incorporating imitation learning into reinforcement learning. We treat samples from teachers as demonstration data, and introduce similar auxiliary supervised loss function for the learner such that learner is able to learn from teachers' samples.

3.2 Interpretable Machine Learning

When the reinforcement learning agents are trained using a deep neural network, it becomes hard to interpret what agent has learned. For self-driving agents, visual attention can be used to interpret trained agents [19]. In the domain of Atari games, saliency maps are popular to understand why the agent makes certain decisions [15]. Another attempt is to use Semi-Aggregated Markov Decision Processes (SAMDP) to visualize clusters of the policy using t-SNE [23] on the neural network activations [43]. In the domain of machine teaching, there are some related work on generating interpretable teaching strategies, effective at teaching humans [25].

Our paper interprets machine learning from a slight different perspective. We find samples that directly help minimize the loss function. We use a gradient-based method, but instead of finding attentions in those samples, we are finding a set of samples that boosts learning.

3.3 Machine Teaching

In machine teaching, teacher can either select examples from its training history, or generate synthetic data. There are many papers use optimization to find a subset of samples for teaching purposes [28, 39]. However, those methods are commonly computational inefficient. There are some attempts to adopt a greedy approach to achieve similar theoretical guarantees [7]. Communication protocols [10] is another way to communicate between agents but requires training a recurrent deep Q-network and full exposure of all learners' models and gradients. Such communication protocols might not be interpretable.

Iterative machine teaching [21] is the closest work. It finds samples that balances the usefulness and difficulties of samples according to stochastic gradient descent (SGD) methods. The way the authors define the usefulness of the samples is similar to the *progress scores* we will define in Section 4.2. In their framework, the teacher must know learner's weight parameters, and learner's optimization method needs to be stochastic gradient descent. Nevertheless, the paper provides a strong theoretical foundation for the teaching dimension. Their follow-up paper [20] extends the earlier framework, disabling teacher to access student's model parameters. They introduce active learning framework to test student's performance. Our paper extends their original paper such that under multiple teacher framework and ZPD principles, the usefulness of the samples can be well approximated without knowing learner's weight parameters.

There are also some related papers incorporating machine teaching with reinforcement learning. A partially observable Markov decision process planning problem can be

modeled as the teaching problem [33]. There is also some work on building models to estimate animal's learning model using policy gradients for adaptive optimal training [1], similar to our *progress scores* in Section 4.2.

3.4 Curriculum Learning and Knowledge Distillation

Curriculum learning [5] specifies that when training a model, it is beneficial to start with a easy subtask and gradually increase the difficulty level of the following tasks. The paper shows that such learning process is a guided optimization for learners as it guides learners converging to a better minima in the model parameter space. From human experiment studies, curriculum learning is more consistent with human learning compared to teaching dimension models [18]. There are some related work on building a intrinsically motivated model to generate automated curriculum for neural network trainings [14].

Another related domain is knowledge distillation [17]. Distillation refers to the process of first training a model for many tasks $\{T_1, \ldots, T_n\}$ and then use it to learn task T_{n+1} quickly. However, knowledge distillation usually relies heavily on knowing the model parameters. In our framework, we do not have model assumptions or access to the trained model. Samples are the only communication protocol.

In summary, our paper has three major contributions:

1. We propose a framework of curriculum distillation such that teachers use their own histories to approximate learner's current state, send examples slightly in advance of learner's current level (ZPD), and generate a set of teaching samples according to how much they have helped teachers' training. The framework doesn't require joint optimization.

2. We present and visualize results from a machine-machine teaching process, in playing Atari games using deep neural networks.

Chapter 4

Approach

4.1 Single-Teacher Curriculum Distillation

Following the principle of Zone of Proximal Development in Section 2.4, teacher (machine) needs to (1) find out learner's current performance level and (2) generate samples in the Zone of Proximal Development.

Setup

When the teacher is trained, no matter using DQN, A3C [26] or Evolution strategies [35], we take snapshots of its neural network parameter θ_t every $T_{snapshot}$ steps. At the end of the training process, we have the history of how the parameter evolves throughout training: $\{\theta_1, \ldots, \theta_N\}$. We denote the last snapshot of parameters as $\theta^* \approx \theta_N$ since it is approximately close to the optimal parameters. In the meantime, teacher's trajectories of each episode are also saved.

Matching

Once we have the whole history of the teacher, we can match the learner's current performance level to one of the teacher's snapshots. One simplest way is to match according to the average episode reward. Episode rewards tend to be very noisy especially during the training time, in order to detect significant reward improvement, we average episode rewards in a sliding window of 100 episodes. Suppose learner's current average episode reward is R, we match it to one of the teacher's snapshots θ_i such that the average episode reward $R_i \approx R$. We make the assumption here that similar average episode rewards correspond to the similar performance level. By matching the average episode rewards, teacher is able to find where the current learner locates in teacher's training history.

Finding Samples

To generate samples in ZPD, teacher needs to educate using examples slightly beyond learner's current level. After matching learner's current performance to teacher's snapshot θ_i, we use samples around teacher's snapshot $\min(N, i + T_{lead})$. In other words, we look ahead T_{lead} snapshots to give such a slight advancement "into the future". Teacher loads $T_{episodes}$ episodes before and after snapshot $\min(N, i + T_{lead})$ and samples them as educating examples for the learner.

Hybrid Minibatch

When learner is trained, the minibatch not only contains samples from its own replay buffer, but also has samples from the teacher. Let δ be the ratio of teacher's samples in learner's minibatch. By the principle of scaffolding, δ should decay throughout the time, denoted as $\delta(t)$. Similar to [16], we apply a margin loss for teacher's sample transition $\langle s, a, r, s' \rangle$:

$$L_\theta^{margin}(s, a, r, s') = \max_{a' \in A} [Q_\theta(s, a') + l(a, a')] - Q_\theta(s, a),$$

where $l(a, a')$ is the same margin function defined in [16]. Along with the standard Double DQN loss, we define the loss for each minimatch as

$$L_\theta(B) = \frac{1}{|B|} \sum_B L_\theta^{DDQN}(s, a, r, s') + \lambda \frac{1}{|T|} \sum_T L_\theta^{margin}(s, a, r, s')$$

where the margin loss is only applied to teacher's samples T in the minibatch.

The algorithm for the teacher is in Algorithm 2 and an illustration of the procedure is in Figure 2.

Algorithm 2 Single-Teacher Curriculum Distillation

prepare snapshots of teacher's model parameters throughout training: $\{\theta_1, \ldots, \theta_N\}$
initialize a replay buffer D
while learner asks for samples at steps t **do**
  if $t \mod T_{update} = 0$ steps or D is empty **then**:
    $R \leftarrow$ Learner's current average episode rewards
    Find a snapshot θ_i such that its average episode rewards $R_i \approx R$.
    Load the snapshot $\theta_{\min(N, i+T_{lead})}$ and find the episode number E_i of the snapshot.
    Load all sample transitions from episode $E_i - T_{episodes}$ to $E_i + T_{episodes}$ into D.
  end if
  Sample $|B| \cdot \delta(t)$ transitions from D.
end while

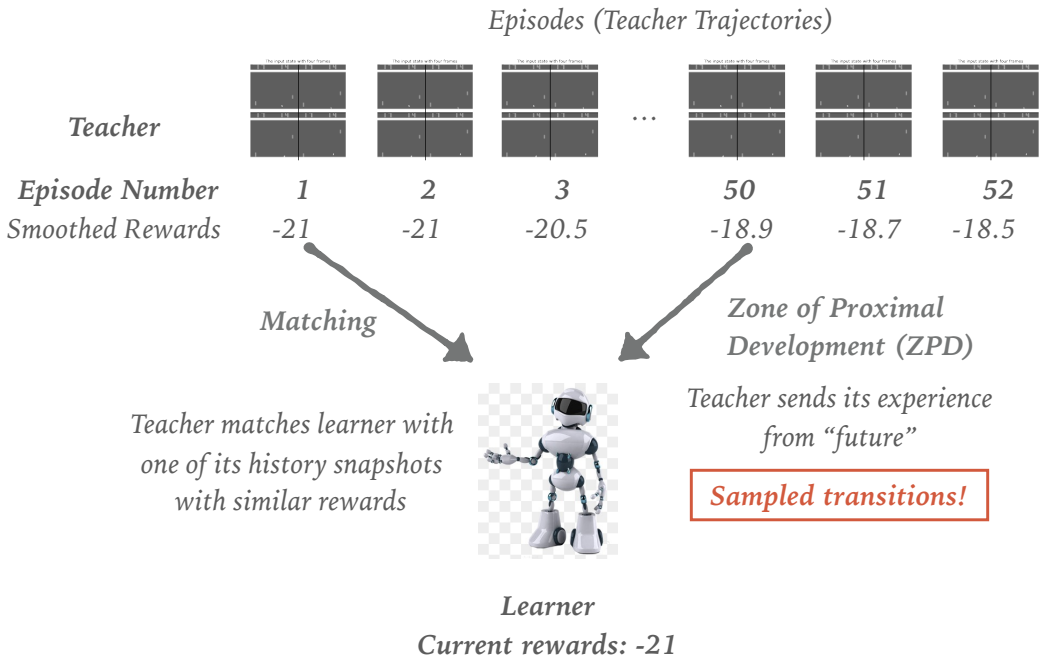


Figure 2: A diagram of the single-teacher curriculum distillation. When the machine teacher is trained, we take snapshots of its model and transitions in its trajectories. When a learner comes and asks for samples, the teacher matches learner's current rewards to one of its snapshot and send samples from a later snapshot following the ZPD principle.

4.2 Progress Scores for Better Sampling

Algorithm 2 uses a uniform distribution to sample from teacher's replay buffer D. A natural followup question is: Are we able to sample more effectively from teacher's history so that it performs better than a uniform distribution. In this section, we propose a measure called *progress scores*.

Intuitively, the progress score for a specific sample measures how helpful it is to make the network parameters converge to the optimal. Suppose teacher has a sequence of snapshots $\{\theta_1, \theta_2, \ldots, \theta_N\}$ with a total of N snapshots. If the model is properly trained and converges, we can assume that the last snapshot is closer to the optimal parameter vector $\theta_N \approx \theta^*$. Therefore the vector $\theta^* - \theta_i$ measure the optimal direction in the parameter space at snapshot θ_i. To measure how helpful a specific sample is during its training process, we can measure the inner product between the sample's gradient direction at snapshot θ_i and the optimal parameter direction $\theta^* - \theta_i$. The progress score

for a sample s at snapshot θ_i can be written as

$$\Gamma(s) = \frac{(\theta^* - \theta_i)^T(-\nabla_{\theta_i} L_{\theta_i}(s))}{\|\theta^* - \theta_i\|_2} = \frac{(\theta_i - \theta^*)^T \nabla_{\theta_i} L_{\theta_i}(s)}{\|\theta^* - \theta_i\|_2} \tag{4}$$

where $\nabla_{\theta_i} L_{\theta_i}(s)$ is the gradient vector of the loss for sample s at snapshot θ_i. Geometrically, progress score is the length of the gradient vector projected on the optimal parameter direction. If the score is positive, it means the projected gradient direction is the same as the optimal direction. If negative, it implies a reverse direction to the optimal one. Figure 3 illustrates such idea in a simple diagram.

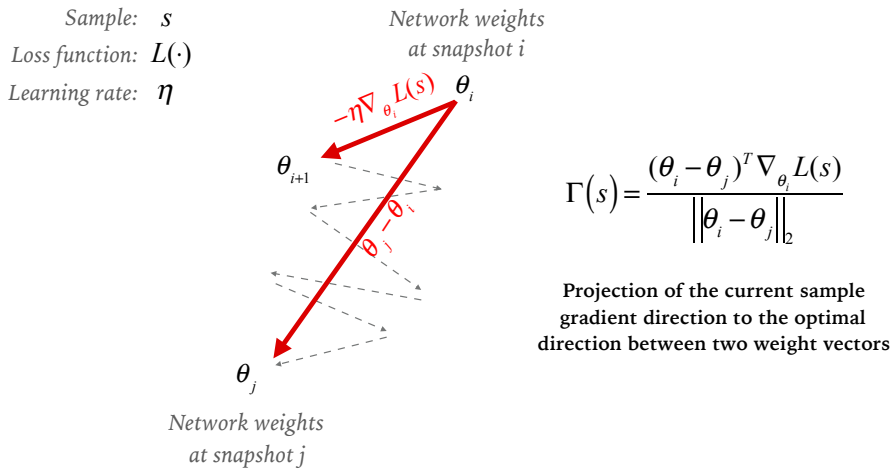


Figure 3: An illustration of progress scores in the parameter space. Consider two snapshots of parameters θ_i, θ_j where $j > i$. A sample s with loss $L(s)$ is evaluated at snapshot θ_i, which has gradient direction of $-\nabla_{\theta_i} L(s)$. We define $Gamma(s)$ as the projection of $-\nabla_{\theta_i} L(s)$ onto $\theta_j - \theta_i$. Intuitively, it represents how much the gradient direction of current sample s under parameter θ_i aligns with the snapshot gradient direction $\theta_j - \theta_i$. In our definition of progress scores, we set $\theta_j = \theta^*$ which $\theta^* - \theta_i$ represents the global parameter direction.

Driven by the nature of first-order optimization methods, we propose to sample teacher's selected examples according to progress scores. Consider for each sample in teacher's replay D, we can compute the progress scores in terms of the current snapshot parameter θ_i, denoting $\Gamma_i(s)$ as a vector of the progress scores. Note that $\Gamma_i(s)$ can be either positive or negative. To convert all $\Gamma_i(s)$ into a distribution for sampling, we propose two methods:

1. *Set negative* $\Gamma_i(s)$ *to be* ϵ. It essentially only considers samples with gradients in the direction of the global parameter direction.

2. *Signal learner to negate gradient direction for samples with negative* $\Gamma_i(s)$. To consider all samples, teacher needs to send additional flags, stating whether $\Gamma_i(s)$ is positive or not. For samples with negative $\Gamma_i(s)$, learner needs to apply the weight -1 when doing the optimization. Sampling is then based on $|\Gamma_i(s)|$.

Both methods should perform well. In our implementation, we use the first method. The most important reason is that by negating some sample's gradient direction, it needs additional tuning on learning rate schedules. Once all progress scores are converted to positive, we can make them into a probability distribution similar to the implementation in prioritized replay [36]:

$$p_i(s) = \frac{\Gamma_i(s)^\alpha}{\sum_j \Gamma_j(s)^\alpha}$$

where α is used to control how much prioritization to use. Moreover, since we sample according to progress scores, we can correct the sampling bias by using importance-sampling (IS) weights. When learner is trained, it applies an additional correction weight for each sample from teacher:

$$w_i(s) = (|D| \cdot p_i(s))^{-\beta}$$

where $|D|$ is the size of the teacher's replay and β is controlling how much correction to use.

4.3 Computing Progress Scores

One critical observation is that computing the numerator in the progress score equation is not easy. It requires per-sample gradients. A naive approach is to forward pass one sample at a time so the backward gradients can be used for that specific sample. However, such approach is very computational expensive as it needs to perform backpropagation for each sample. We propose a faster way to compute the numerator using minibatches instead of individual samples.

Use a very simple linear layer, without bias, as an example. Suppose two snapshots of the parameter are θ_i and θ_j where $j > i$. Let the input to the layer be x and output be y. For snapshot θ_i, we have

$$y = x\theta_i.$$

Once we have the loss $L(x;\theta_i)$, the goal is to compute

$$(\theta_i - \theta_j)^T \frac{\partial L(x;\theta_i)}{\partial \theta_i} \tag{5}$$

Chain rules gives

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial x} = \theta_i^T \frac{\partial L}{\partial y}, \quad \frac{\partial L}{\partial \theta_i} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial \theta_i} = x^T \frac{\partial L}{\partial y}.$$

We observe that to make the form of Equation 5, we need a dummy tensor $x_0 = 0$. By adding $x_0\theta_j$ with $x\theta_i$, we preserve the same y and $L(x)$ value and is able to accumulate the gradient for θ_j:

$$y = x\theta_i = x\theta_i + x_0\theta_j,$$
$$\frac{\partial L}{\partial x_0} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial x_0} = \theta_j^T \frac{\partial L}{\partial y},$$
$$\frac{\partial L}{\partial x} - \frac{\partial L}{\partial x_0} = (\theta_i - \theta_j)^T \frac{\partial L}{\partial y}$$

Therefore, we can compute the goal in Equation 5:

$$(\theta_i - \theta_j)^T \frac{\partial L(x)}{\partial \theta_i} = x\left[\frac{\partial L(x)}{\partial x} - \frac{\partial L(x)}{\partial x_0}\right] \tag{6}$$

The diagram in Figure 4 illustrates the gradient backpropagation for this simple linear layer.

We can observe that values needed to compute Equation 5 never get aggregated for the minibatch. Therefore, for a minibatch X, our target values can be simply rewritten as

$$(\theta_i - \theta_j)^T \frac{\partial L(X)}{\partial \theta_i} = \text{diag}\left(X\left[\frac{\partial L}{\partial X} - \frac{\partial L}{\partial X_0}\right]\right). \tag{7}$$

In addition, such computation can be easily generalized to linear layers with bias and convolution layers.

We design a network that can be easily adapted to compute Equation 7. We can flatten the parameter vector θ for each layer individually. The network is split into upper network and lower network. Upper network is parametrized by θ_i and lower network is parametrized by θ_j. For each layer, the upper network does the regular forward computation using X. The lower network is only used to accumulate gradients for θ_j, therefore, only $X_0 = 0$ feeds into the lower network. After the computation for each layer, we sum the output together from two networks. Based on this design, the sum should be exactly the same as solely forwarding X in the network θ_i. For each layer k after backpropagation, we only need $\frac{\partial L}{\partial X^{(k)}}, \frac{\partial L}{\partial X_0^{(k)}}, X^{(k)}$ to compute Equation 7. A design

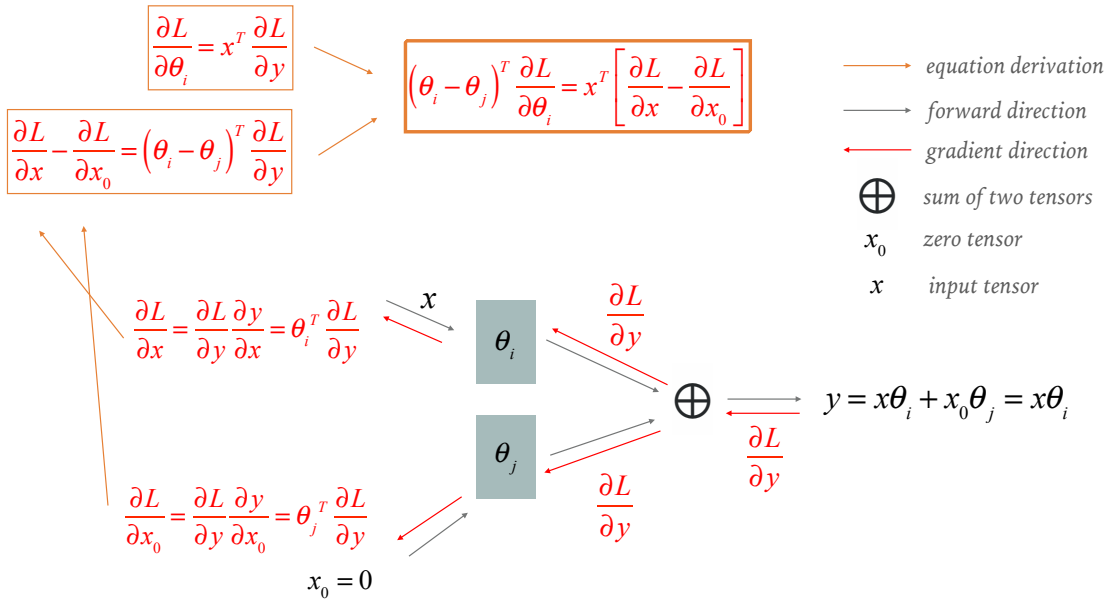


Figure 4: An illustration of how progress scores can be computed in a linear layer.

of such network is illustrates in Figure 5.

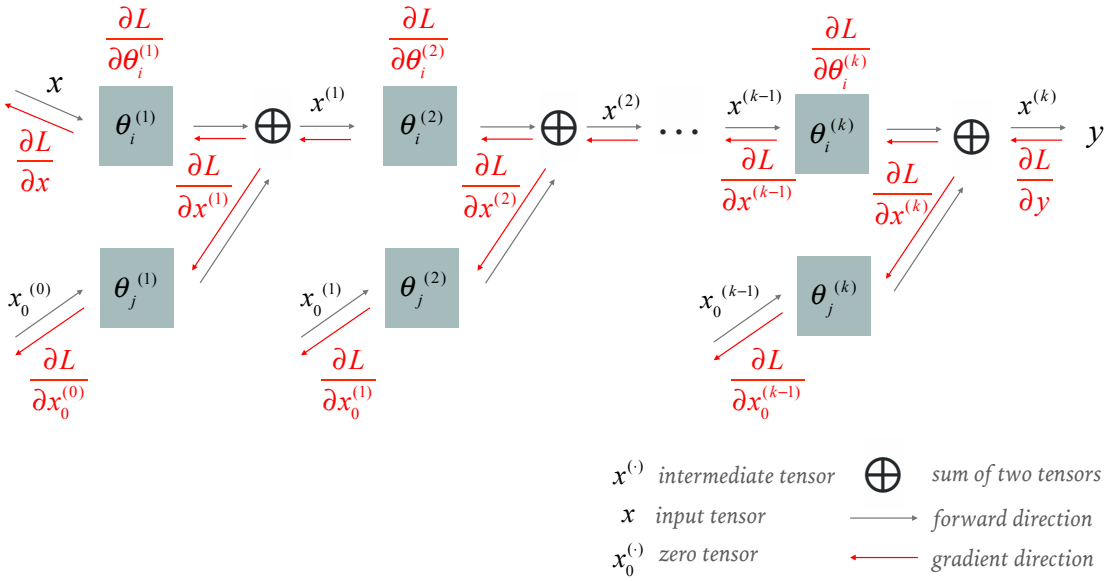


Figure 5: Network to compute progress scores. The upper layer is parameterized by θ_i and fed the real data X while the lower layer is parameterized by θ_j and fed the dummy zero tensor $X_0 = 0$. $X^{(k)}$ is an intermediate variable in the forward pass, the sum of output from upper layer and lower layer.

4.4 Multiple-Teacher Curriculum Distillation

Section 4.1 discusses curriculum distillation following one single teacher. The framework can be generalized to distillation from a pool of teachers. One teacher framework tends to be a bit noisy when the teacher approximates learner's performance level. When multiple teachers participate, their approximation of learner's performance can be averaged hence reduces variance. In addition, averaging the progress scores among multiple teachers also reduces variance.

Such framework extends the framework in Section 4.1 by having a set of teachers $\{\tau_1, \ldots, \tau_p\}$. Each teacher τ_j performs matching based on learner's average episode reward, and contributes sample transitions based on ZDP principles to a shared replay buffer D^*. After each teacher has put samples in the buffer, it also needs to compute the progress scores for all samples that other teacher contributes. The final progress scores are the average over all teachers' computation. Figure 6 and Algorithm 3 illustrates such process.

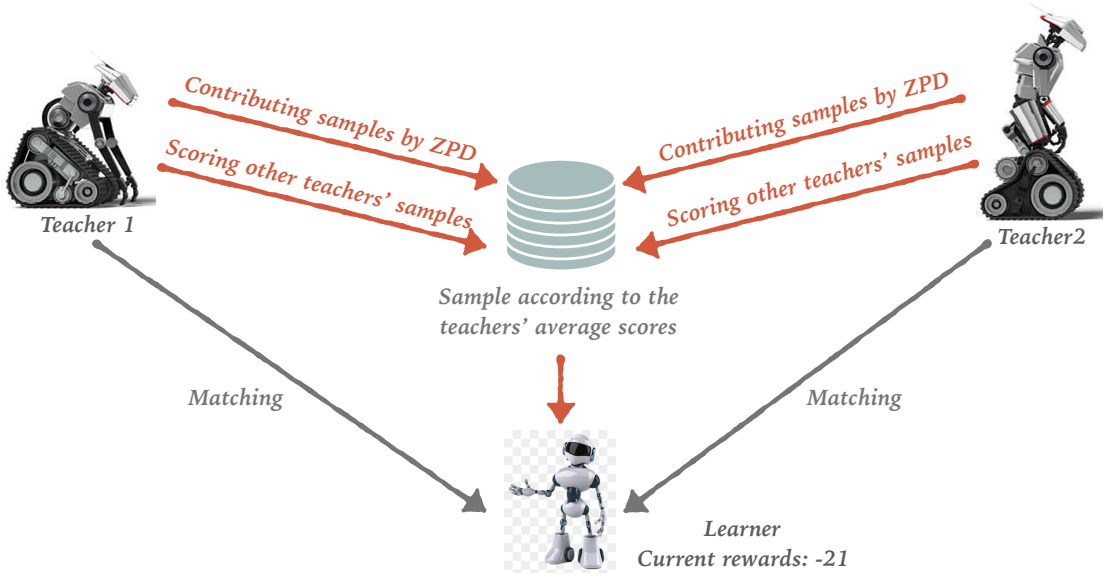


Figure 6: A diagram of multiple-teacher curriculum distillation. Individual teacher does the matching, contributes samples by ZPD, and scores all other teachers' samples. The final progress scores are the average over all teachers' scores.

We also argue the stability of such method. We slightly modify the framework above by allowing each teacher to simulate those sampled transitions. Suppose teacher j matches learner to its snapshot $\theta_i^{(j)}$. We can use snapshot $\theta_i^{(j)}$ as a surrogate to simulate learner's performance with some samples. Suppose we sample a minibatch according

Algorithm 3 Multiple-Teacher Curriculum Distillation

initialize a shared replay buffer D^*
while learner asks for samples at steps t **do**
  if $t \mod T_{update} = 0$ steps or D^* is empty **then**:
    $R \leftarrow$ Learner's current average episode rewards
    foreach teacher τ_j **do**
      Find a snapshot $\theta_i^{(j)}$ such that its average episode rewards $R_i^{(j)} \approx R$.
      Load snapshot $\theta^{(j)}_{\min(N, i+T_{lead})}$ and find its episode number $E_i^{(j)}$.
      Load all transitions from episode $E_i^{(j)} - T_{episodes}$ to $E_i^{(j)} + T_{episodes}$ into D^*.
    end for
    foreach teacher τ_j **do**
      Compute progress scores (Equation 4) for samples from all other teachers.
    end for
    Average progress scores from all teachers.
  end if
  Sample $|B| \cdot \delta(t)$ transitions from D^* according to progress scores.
end while

to the averaged progress scores to simulate learner. For each teacher, it uses the same loss function as learner and performs gradient updates on snapshot $\theta_i^{(j)}$ to $\hat{\theta_i^{(j)}}$. Then, we can recompute progress scores using the simulated parameter $\hat{\theta_i^{(j)}}$ and repeat this process multiple iterations.

The whole idea of this simulation process is that we want to make samples selected by teachers more independent. Figure 7 illustrates this deduplicate process. Comparing the probability distribution from averaged progress scores before and after the deduplicate process, we see minimal changes in the perplexity of the distribution. It shows the stability of the progress score measures.

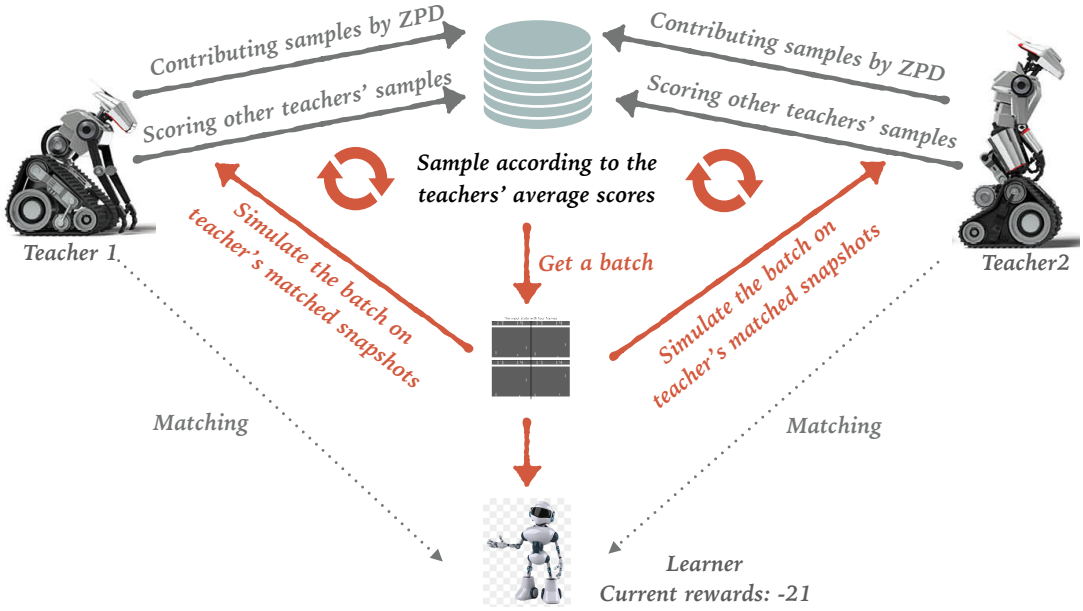


Figure 7: Multiple teacher framework with deduplicate process. After scoring all samples, each teacher sample a minibatch accordingly and update the effect on its matched parameter. Each teacher then rescores the samples using the updated parameter. The process continues for a few iterations.

Chapter 5

Experiments

5.1 Setup

For reproducibility purposes, we list all the hyperparameters we used in game Pong in Table 1. Figure 8 shows the training curve for a standard Double-Q agent with 18.0 points as the target score. During the training process, we take snapshots of the network weights every 40k training steps.

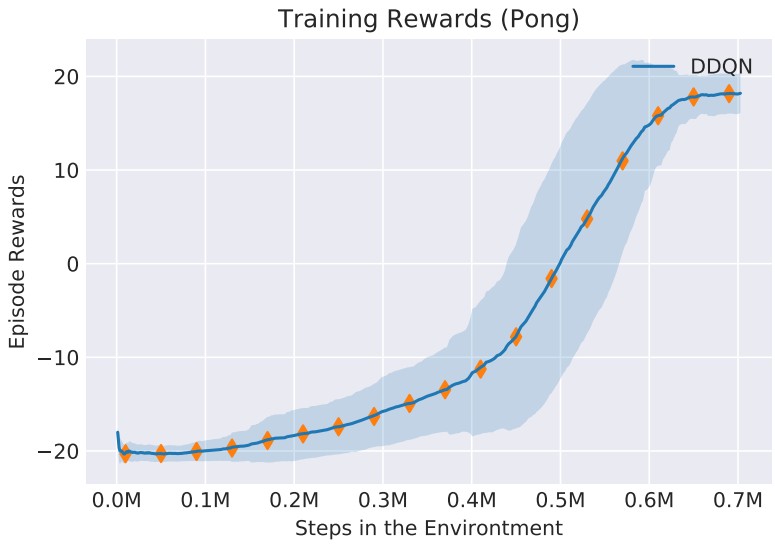


Figure 8: A sample training reward curve for Pong. The curve is smoothed across the most 100 training episodes with the error band denoting the standard errors of the rewards. Throughout this trajectory, it has 358 episodes in total along with 18 snapshots marked in the figure.

Symbol	Description	Values
Stopping rule	Stopping criteria during training	Either the last 100 episodes achieve an average rewards above 18.0 or it hits a total of 1,000,000 training steps
T_{play}	Number of steps to step in the training environment in between training steps Q-network	4
T_{sync}	Number of training steps to update target Q-network	1,000
$\|B\|$	Training minibatch size	128
$\|T\|$	Number of teacher's samples in the training minibatch size	Depend on minibatch blending rule
$\|D\|$	Replay buffer size	100,000
D_{start}	Minimum size of the samples in the replay buffer for training	10,000
ϵ	Controlling the probability of selecting a random action	Anneal from 1 to 0.2 in the first 100,000 frames and then to 0.02 until the end
η	Learning rate for Adam	Anneal from 2.5e-4 to 1e-4 in the first 100,000 frames and then stay at 1e-4 until the end
$T_{snapshot}$	Number of steps to take a snapshot of the model parameters during training teacher models	40,000
Minibatch blending rule	The composition of teacher's samples in the learner's training minibatch	Anneal from 0.5 to 0.2 in the first 100,000 frames and then to 0.02 until the end
α	Softmax exponents to convert progress scores into probability	1
β	Exponents to correct importance sampling	Anneal from 0 to 1
λ	Scaling factor controling the supervised loss	5e-2
$l(a, a')$	Margin function in the classification loss	0.8 for $a \neq a'$
T_{lead}	Number of snapshots to lead in advance of the matching snapshot	$\{1, 2, 4, 8\}$
$T_{episodes}$	Number of episodes to read around the selected snapshots	5
T_{update}	Number of training steps to update snapshot matching process	10,000
$\|\tau\|$	Number of teachers	5

Table 1: Hyperparameters for Atari game Pong

5.2 ZPD Experiments

The first experiment is to examine the ZPD principles. We repeat the same single-teacher curriculum distillation procedure with various T_{lead} in Algorithm 2. In Figure 9, we show its effect on accelerating learner's training. All experiments are averaged over 12 different random seeds. We can see that the optimal number of leading snapshots is around 2, which is around 80k steps or 40 episodes ahead. With fewer or more snapshots, the guidance produces more variance, resulting in some of the experiment getting extremely low scores.

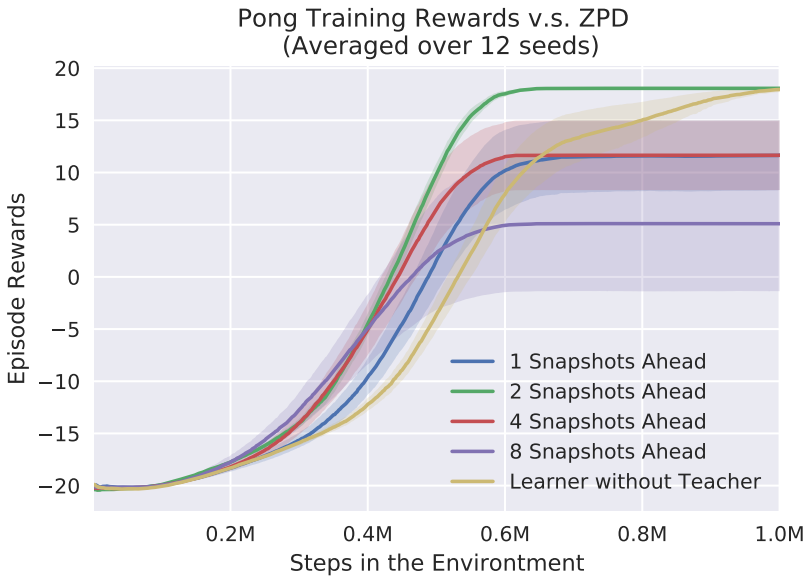
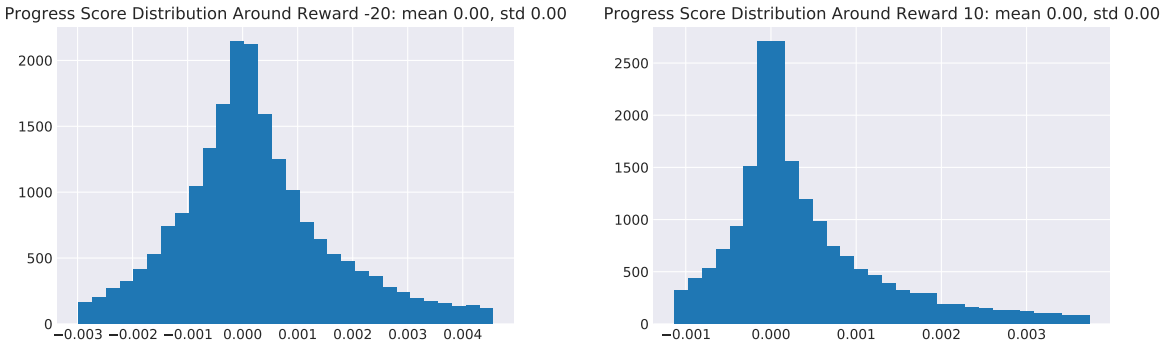


Figure 9: Pong's training rewards with different degrees of leading snapshots. 2 snapshots seem to be in the Zone of Proximal Development.

5.3 Progress Scores

In this section, we visualize the progress scores defined in Chapter 4.2. In Figure 10, we compare the histogram of the progress scores at two snapshots, one with averaged rewards around -20 and the other around -10. The average progress score is a very small positive number around zero as globally samples make progress towards the optimal parameters. The most important thing to notice between two histograms is that the more progress scores are getting closer to 0 when learning progresses. Intuitively, in the later stage of learning, fewer and fewer samples can actually help the agent achieve higher rewards.

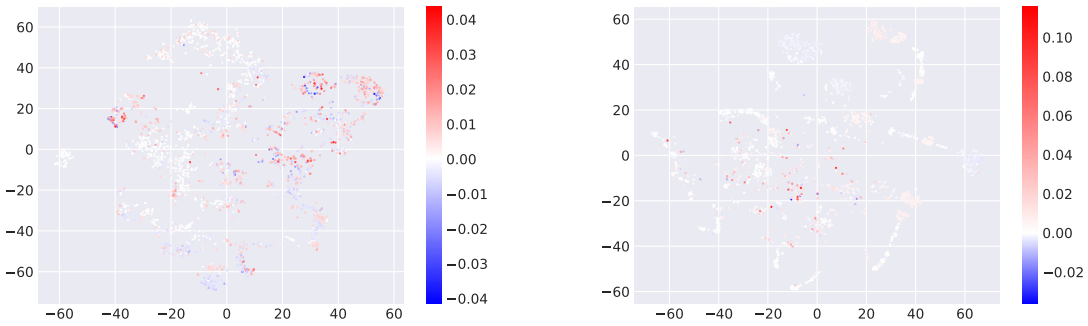
Another way to visualize such progress scores is to color the progress scores on a t-SNE projection of states. Similar to [43], we use the teacher's final snapshot model to



(a) Rewards: -20 **(b)** Rewards: 1-

Figure 10: Histograms of the progress scores at snapshot with averaged rewards -20 and 10.

get the activations for each state. Then we use PCA to reduce the dimension to 50, and project them to 2 dimension using t-SNE. We also color each sample using the progress scores computed at their corresponding snapshots. In Figure 11, we showed the t-SNE plot with progress scores at snapshots with rewards -20 and 10. It can be seen that states with positive progress scores have their clusters. Similarly for those states with negative progress scores. The clustering effect becomes a bit weaker in the later stage, likely due to progress scores are skewed to 0. But in both cases, we can see that those states with around 0 progress scores usually have their own unique clusters.



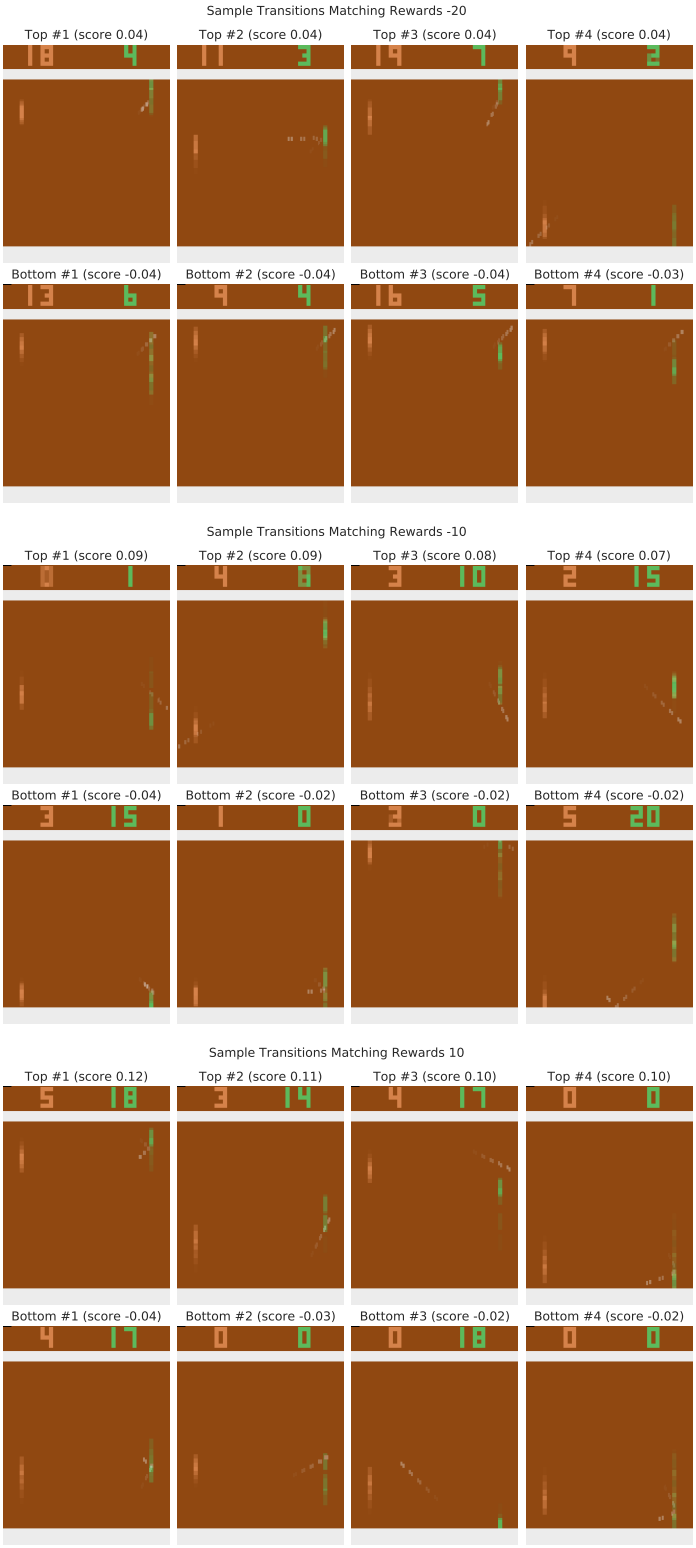
(a) Rewards: -20 **(b)** Rewards: 10

Figure 11: t-SNE projection of state activations colored by progress scores at snapshots with averaged rewards -20 and 10.

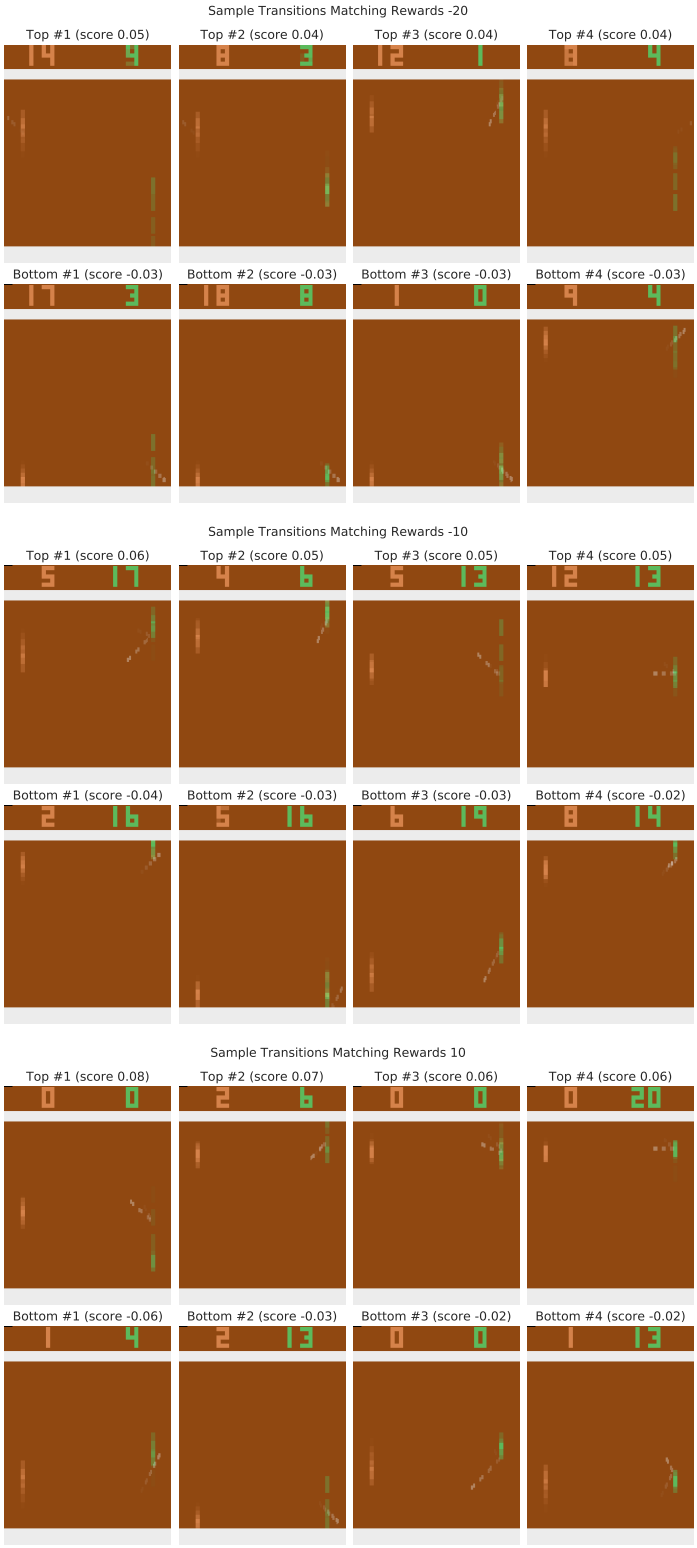
5.4 Samples with Extreme Progress Scores

In Figure 12, we demonstrate a few samples selected by such framework at snapshots with rewards around -20, -10 and 10. The idea of DQN is to make the neural network prediction satisfy Bellman equation. Then by dynamic programming, it propagates Q-values to earlier states. Therefore, useful samples should be those critical to correct the loss in predicting the Q-values.

From the figures, we can see that states that help agent learn are the states with non-zero rewards. For snapshot with averaged rewards around -20, states with most positive scores are those getting one point, and those with most negative scores are those losing one point. The panel on the left is generated by one teacher and the one on the right is averaged over 5 different teachers. The samples are more consistently interpretable compared to those generated by only one sample.



(a) Scored by one random teacher



(b) Scored by 5 random teachers

Figure 12: Sampled transitions sampled according to progress scores for snapshot with rewards around -20, -10 and 10. The left panel shows the most extreme samples scored by one teacher. The right panel shows ones rated by 5 teachers.

5.5 Learning Based on Progress Scores

Finally, we demonstrate the performance of sampling according to the averaged progress scores across multiple teachers. In Figure 13, we use ZPD-based machine teaching framework with 1 snapshot look ahead. We compare sampling with a uniform distribution with sampling according to the averaged progress scores rated by 5 teachers. For each experiment, all five teachers are randomly selected. We run 12 different random seeds and average the result together. It can be seen that such sampling accelerates learner's training even further.

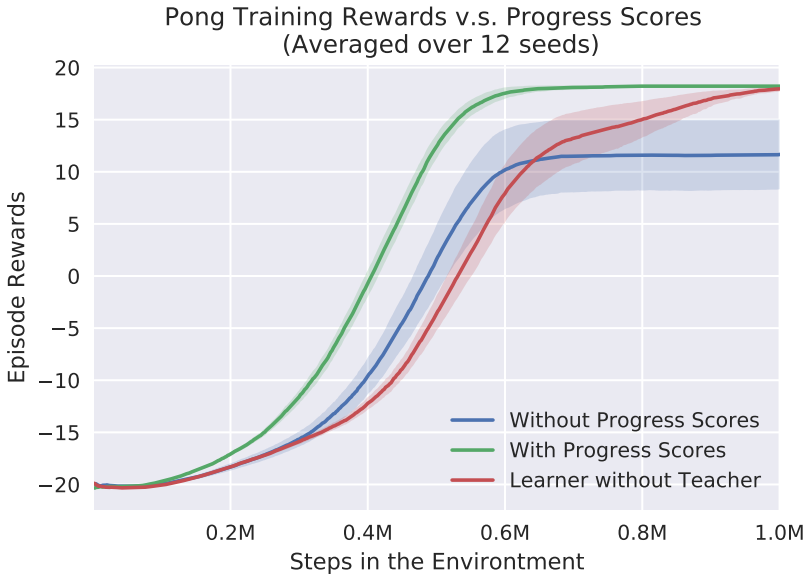


Figure 13: Pong's training rewards with ZPD-based machine teaching. The experiment uses 1 snapshot ahead of current learner's performance and compares the performance between uniform distribution and sampling according to progress scores by 5 teachers. It can be seen that using multiple teachers' averaged progress score accelerates the learning.

Chapter 6

Discussions

In this paper, we discuss a framework to distill curriculum from well-trained DQN agents. By following ZPD principle, we demonstrate that good samples in the curriculum must be slight beyond learner's current performance level. In addition, good samples should be the ones that help teachers themselves the most during the training. Therefore, in the setting of machine teaching and curriculum learning, we are able to approximately find the best samples for a learner, based on the learner's averaged rewards. There are some uncertainty and noise for individual teacher so that we propose the methods to average across multiple teachers for robustness. In the end, we visualize those samples selected at different stages when training an agent playing Pong. When a new machine agent is trained using such curriculum, we see great improvement in its training convergence time.

There are a few very interesting ideas to follow this work. The primary one is to generalize this framework to other learning settings. For example, distill knowledge when training a image classifier or a generative model. In the setting of reinforcement learning, it will be helpful to use the interpretability of such framework to understand other training algorithms such as policy gradients method and evolution strategies.

Another idea is to find the connection between the curriculum generated under such framework and flash card algorithm. We can observe from those samples with extreme progress scores in Pong that some of them are very similar across different snapshots. It will be interesting to model those as a recalling process.

Hierarchical reinforcement learning is to divide learning tasks into individual tasks. It will be interesting as well to generate subtask curriculum or a Semi Markov Decision Process based on the distilled curriculum we proposed.

References

[1] Ji Hyun Bak, Jung Yoon Choi, Athena Akrami, Ilana Witten, and Jonathan W Pillow. Adaptive optimal training of animal behavior. In D D Lee, M Sugiyama, U V Luxburg, I Guyon, and R Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 1947–1955. Curran Associates, Inc., 2016.

[2] Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. *arXiv preprint arXiv:1707.06887*, 2017.

[3] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[4] Richard Bellman. A markovian decision process. *Journal of Mathematics and Mechanics*, pages 679–684, 1957.

[5] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM, 2009.

[6] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[7] Yuxin Chen, Oisin Mac Aodha, Shihan Su, Pietro Perona, and Yisong Yue. Near-optimal machine teaching via explanatory teaching sets. In *International Conference on Artificial Intelligence and Statistics*, pages 1970–1978, 2018.

[8] Edwin S Ellis and Lou Anne Worthington. Research synthesis on effective teaching principles and the design of quality tools for educators. technical report no. 5. 1994.

[9] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*, 2018.

[10] Jakob Foerster, Ioannis Alexandros Assael, Nando de Freitas, and Shimon Whiteson. Learning to Communicate with Deep Multi-Agent Reinforcement Learning. In D D Lee, M Sugiyama, U V Luxburg, I Guyon, and R Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2137–2145. Curran Associates, Inc., 2016.

[11] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, et al. Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*, 2017.

[12] Yang Gao, Ji Lin, Fisher Yu, Sergey Levine, Trevor Darrell, et al. Reinforcement learning from imperfect demonstrations. *arXiv preprint arXiv:1802.05313*, 2018.

[13] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[14] Alex Graves, Marc G Bellemare, Jacob Menick, Remi Munos, and Koray Kavukcuoglu. Automated curriculum learning for neural networks. *arXiv preprint arXiv:1704.03003*, 2017.

[15] Sam Greydanus, Anurag Koul, Jonathan Dodge, and Alan Fern. Visualizing and understanding atari agents. *arXiv preprint arXiv:1711.00138*, 2017.

[16] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Gabriel Dulac-Arnold, et al. Deep q-learning from demonstrations. *arXiv preprint arXiv:1704.03732*, 2017.

[17] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[18] Faisal Khan, Bilge Mutlu, and Xiaojin Zhu. How do humans teach: On curriculum learning and teaching dimension. In *Advances in Neural Information Processing Systems*, pages 1449–1457, 2011.

[19] Jinkyu Kim and John Canny. Interpretable learning for self-driving cars by visualizing causal attention. *arXiv preprint arXiv:1703.10631*, 2017.

[20] Weiyang Liu, Bo Dai, Xingguo Li, James M Rehg, and Le Song. Towards black-box iterative machine teaching. *arXiv preprint arXiv:1710.07742*, 2017.

[21] Weiyang Liu, Bo Dai, James M Rehg, and Le Song. Iterative machine teaching. *arXiv preprint arXiv:1705.10470*, 2017.

[22] P. Lloyd and C. Fernyhough. *Lev Vygotsky: Critical Assessments*. Number v. 3 in Critical Assessments. Routledge, 1999.

[23] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[24] Shike Mei and Xiaojin Zhu. Using machine teaching to identify optimal training-set attacks on machine learners. 2015.

[25] Smitha Milli, Pieter Abbeel, and Igor Mordatch. Interpretable and pedagogical examples. *arXiv preprint arXiv:1711.00694*, 2017.

[26] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

[27] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[28] Igor Mordatch and Pieter Abbeel. Emergence of grounded compositional language in multi-agent populations. *arXiv preprint arXiv:1703.04908*, 2017.

[29] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. *arXiv preprint arXiv:1709.10089*, 2017.

[30] David R Olson. *Jerome Bruner: The cognitive revolution in educational theory*. Bloomsbury Publishing, 2014.

[31] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. In *Advances in neural information processing systems*, pages 4026–4034, 2016.

[32] Kaustubh R Patil, Xiaojin Zhu, Łukasz Kopeć, and Bradley C Love. Optimal teaching for limited-capacity human learners. In *Advances in neural information processing systems*, pages 2465–2473, 2014.

[33] Anna N Rafferty, Emma Brunskill, Thomas L Griffiths, and Patrick Shafto. Faster teaching via pomdp planning. *Cognitive science*, 40(6):1290–1332, 2016.

[34] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011.

[35] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

[36] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[37] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[38] Patrice Y Simard, Saleema Amershi, David M Chickering, Alicia Edelman Pelton, Soroush Ghorashi, Christopher Meek, Gonzalo Ramos, Jina Suh, Johan Verwey, Mo Wang, et al. Machine teaching: A new paradigm for building machine learning systems. *arXiv preprint arXiv:1707.06742*, 2017.

[39] Sainbayar Sukhbaatar, Rob Fergus, et al. Learning multiagent communication with backpropagation. In *Advances in Neural Information Processing Systems*, pages 2244–2252, 2016.

[40] Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction. 2011.

[41] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, volume 16, pages 2094–2100, 2016.

[42] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

[43] Tom Zahavy, Nir Ben-Zrihem, and Shie Mannor. Graying the black box: Understanding dqns. In *International Conference on Machine Learning*, pages 1899–1908, 2016.

[44] Xiaojin Zhu. Machine teaching: An inverse problem to machine learning and an approach toward optimal education. In *AAAI*, pages 4083–4087, 2015.

[45] Xiaojin Zhu, Adish Singla, Sandra Zilles, and Anna N Rafferty. An overview of machine teaching. *arXiv preprint arXiv:1801.05927*, 2018.