Learning to Navigate in Visual Environments

by

Peter Jin

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Kurt Keutzer, Chair
Professor Sergey Levine
Professor Ken Goldberg

Fall 2018

Learning to Navigate in Visual Environments

Abstract

Learning to Navigate in Visual Environments

by

Peter Jin

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Kurt Keutzer, Chair

Artificially intelligent agents with some degree of autonomy in the real world must learn to complete visual navigation tasks. In this dissertation, we consider the learning problem of visual navigation, as well as implementation issues facing agents that utilize learned visual perception systems. We begin by formulating visual navigation tasks in the setting of deep reinforcement learning under partial observation. Previous approaches to deep reinforcement learning do not adequately address partial observation while remaining sample-efficient. Our first contribution is a novel deep reinforcement learning algorithm, advantage-based regret minimization (ARM), which learns robust policies in visual navigation tasks in the presence of partial observability. Next, we are motivated by performance bottlenecks arising from large scale supervised learning for training visual perception systems. Previous distributed training approaches are affected by synchronization or communication bottlenecks which limit their scaling to multiple compute nodes. Our second contribution is a distributed training algorithm, gossiping SGD, which avoids both synchronization and centralized communication. Finally, we consider how to train deep convolutional neural networks when inputs and activation tensors have high spatial resolution and do not easily fit in GPU memory. Previous approaches to reducing memory usage of deep convnets involve trading off between computation and memory usage. Our third and final contribution is an implementation of spatially parallel convolutions, which partition activation tensors along the spatial axes between multiple GPUs, and achieve practically linear strong scaling.

To Karen, mom, and dad.

Contents

List of Figures

List of Tables

Acknowledgments

First, I'd like to thank my advisor Kurt Keutzer for providing a supportive research environment, and specifically for supporting my own various research endeavors over the years. I'm also thankful to have enjoyed close collaborations with Sergey Levine and Boris Ginsburg on work that appears in this dissertation. For the work on advantage-based regret minimization, Tambet Matiisen and Deepak Pathak helped with experiments, and Tuomas Haarnoja gave insightful feedback. The work on gossiping SGD traces back to a convex optimization class project with Qiaochu Yuan during the spring of 2016 (taught by Ben Recht), as well as to some inspiration from Forrest Iandola's work on scaling distributed SGD and some brainstorming during an ASPIRE winter retreat at Granlibakken. Our experiments on the Titan supercomputer at Oak Ridge were made possible by support from Ed F. D'Azevedo, Chris Fuson, and Lenny Oliker. Finally, I'm thankful to my family—my sister Karen and our parents—who have always been there for me.

I had the fortune of working in the company of many good colleagues in the lab during the end of the Par Lab and through the ASPIRE era: Michael Anderson, Khalid Ashraf, Yubei Chen, Amir Gholami, Katya Gonina, Forrest Iandola, Ali Jannesari, Kiseok Kwon, Patrick Li, Matt Moskewicz, David Sheffield, Bor-Yiing Su, Alberto Todeschini, Alvin Wan, Bichen Wu, Xiangyu Yue, and Sicheng Zhao. Kosta Ilov and Jon Kuroda were always very helpful and knowledgeable, as were Roxana Infante, Tami Chouteau, Ria Briggs, and Angie Abbatecola. I enjoyed enlightening discussions with numerous colleagues and friends at Berkeley, but I am especially grateful to Ma'ayan Bresler, Diya Das, David Fridovich-Keil, Rob Kenny, Babar Khan, Richard Li, Anting Shen, Haoran Tang, Josh Tobin, and Yang You.

I'd like to thank Joey Gonzalez, Ken Goldberg, and Sergey for serving on my quals committee, and to Sergey and Ken for serving on my dissertation committee. I'd like to especially thank Ujval Kapasi and Boris Ginsburg, who gave me the opportunity of interning at NVIDIA over the summer of 2017. I also had the pleasure of attending and competing in the 9th UEC Cup in Tokyo during March 2016, for which I'd like to thank Hiroshi Yamashita, and the organizers more generally, for their support and hospitality.

The work on advantage-based regret minimization (Chapter 2) was partially supported by Berkeley DeepDrive, ADEPT Lab industrial sponsors and affiliates Intel, Google, Siemens, and SK Hynix, DARPA Award Number HR0011-12-2-0016, and ASPIRE Lab industrial sponsors and affiliates Intel, Google, HPE, Huawei, LGE, Nokia, NVIDIA, Oracle, and Samsung. The work on gossiping SGD (Chapter 3) was partially supported by DARPA Award Number HR0011-12-2-0016 and by ASPIRE Lab industrial sponsors and affiliates Intel, Google, Hewlett-Packard, Huawei, LGE, NVIDIA, Oracle, and Samsung. This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. The work on spatially parallel convolutions (Chapter 4) was done in part using DGX systems during the author's internship at NVIDIA Corporation. Amazon also granted an AWS Educate credit.

Chapter 1

Introduction

For artificially intelligent systems to fully participate in the physical world, they must learn to do some basic tasks on their own. Arguably, among the most interesting basic tasks is *navigation*, which simply consists of moving toward a destination in the world. Why is navigation interesting? Although motion and navigation are not obviously requisite for AI, we reason that freedom of motion in the physical world contributes to an agent building richer internal representations of space than if the agent were stationary or restrained. Autonomy itself can also be broadly interpreted to encompass both the construction and the utilization of internal representations of an external environment [5]. So, the capability to navigate suggests to us a richness of spatial representation and entails autonomy, which would seem crucial for participation in the physical, spatial world.

Precisely, we are interested in having artificial agents learn to complete *visual navigation* tasks, during which the agent observes images of the world around it, and based on progress toward its objective (i.e. reach a goal position) the agent decides on and performs an action (e.g. move forward, turn left). We think of visual navigation as comprising a "visual perception" component and a "control and decision-making" component. State-of-the-art approaches to learned visual perception are based on the training of deep convolutional neural networks [53]. We also pose the learning of visual navigation tasks in the setting of deep reinforcement learning under partial observation: a visual navigation task is represented as a partially observable Markov decision process (POMDP), where states correspond to configurations of the world, observations correspond to sequences of images, and actions correspond to motion, and where policies or other function approximations are parameterized by deep convnets.

1.1 Contributions of This Thesis

In this dissertation, we examine two varieties of problems related to learning visual navigation tasks. The first problem class is concerned with how to solve the fundamental learning problem of visual navigation, which we posed above as deep reinforcement learning under partial observation. Previous deep reinforcement learning approaches based on policy gradients and Q-learning do not adequately address partial observation while remaining sample-efficient. We take inspiration from a family of methods based on counterfactual regret minimization which are proven to be robust under a form of partial observation [104], and as a result, we develop a novel deep reinforcement learning algorithm, advantage-based regret minimization (ARM). We find that ARM is able to learn strong policies compared to baseline methods in representative visual navigation tasks in the presence of partial observation perturbations. This work was previously published as [45].

The second problem class can be summarized as systems considerations associated with the implementation of learning. One problem in this class is that of scaling up supervised learning for perception to very large datasets by distributed training, while avoiding synchronization or communication bottlenecks that hamper existing synchronous and asynchronous methods. The building block we employ is the family of peer-to-peer gossip protocols, which, when combined with stochastic gradient descent (SGD), yields a novel gossiping SGD algorithm that is amenable to large scale distributed training for supervised learning. Gossiping SGD avoids both the communication bottleneck of comparable asynchronous methods [103] and the synchronization bottleneck of synchronous SGD, while still scaling to 16–32 nodes of a GPU supercomputing cluster for training a ResNet to classify ImageNet. This work was previously published as [43].

Another problem in the class of systems considerations concerns the training of deep convnets on high resolution inputs and activation tensors, which can surpass the limited memory capacity of GPUs. We develop a novel implementation of spatially parallel convolutions for multi-GPU systems, inspired by old ideas for distributing stencil codes in scientific computing. Interestingly, our implementation of spatially parallel convolutions on multiple GPUs often attains near-linear speedups, and occasionally attains superlinear speedups, compared to baseline single-GPU convolutions. This work was previously published as [44].

Chapter 2

Advantage-based Regret Minimization

2.1 Introduction

In this chapter, we focus on the problem of learning to complete *visual navigation* tasks: learning to autonomously move from one point to another in 3D environments with access only to first-person perspective images. We pose the problem of learning to visually navigate within the problem setting of reinforcement learning under partial observation. The difficulty of learning to navigate in visual environments largely arises from the effects of *partial observation.* Among these effects, three are especially relevant to reinforcement learning from images. First, observed images obtained in first-person perspective are projections of the full 3D state of the environment and are naturally partially observed. Second, images may contain substantial occlusion, whether due to the effects of 3D perspective or due to perturbation of the image acquisition process. Third, an agent actually observes a sequence or history of images, but practically only a finite memory of recent images may be stacked together into an "observation."

Many reinforcement learning problems of practical interest have the property of partial observability, where observations of state are generally non-Markovian. Practical deep reinforcement learning algorithms fall into two broad classes, neither of which satisfactorily deals with partial observability despite the prevalance of partial observations in the real world. One class of algorithms consists of value function-based methods such as deep Q-learning [65, 66], which are known to be highly sample efficient but generally assume a Markovian observation space. The other class of algorithms consists of Monte Carlo policy gradient methods such as trust region policy optimization [83], which do not need to assume Markovian observations but are less sample efficient than value function-based methods. Some policy gradient methods such as advantage actor-critic [64] introduce the Markov assumption through a critic or state-dependent baseline to improve sample efficiency.

There are two common workarounds for the problem of partial observation: (a) learning policies and value functions on finite length observation histories, and (b) learning recurrent policies and recurrent value functions. Finite length observation histories concatenate the most recent raw observations into a stack of observations, and are a simple but effective technique to approximate the appearance of full observability. Recurrent functions can potentially incorporate an infinite observation history, but they can be harder to optimize. When using either finite length observation histories or recurrent functions, the same value function-based methods and policy gradient methods are employed with their respective tradeoffs in partially observable domains.

We are interested in developing methods that combine the best of both value function-based methods and policy gradient methods for partial observable domains. That is, can we develop methods that are sample efficient but are also robust to partial observation spaces?

Our contribution is a new model-free deep reinforcement learning algorithm based on the principle of regret minimization which does not require access to a Markovian state. Our method learns a policy by estimating an advantage-like function which approximates a quantity called the counterfactual regret. Counterfactual regret is central to the family of counterfactual regret minimization (CFR) [104] algorithms for solving incomplete information games. Hence we call our algorithm "advantage-based regret minimization" (ARM).

We evaluate our approach on three partially observable visual reinforcement learning tasks: first-person 3D navigation in Doom and Minecraft [50, 46], avoiding partially observed objects in Doom, and playing Pong with partial observation perturbations [3]. In our experiments, we find that our method is more robust and offers substantial improvement over prior methods on partially observable tasks.

2.2 Related Work

Deep Q-learning and policy gradient methods

Deep reinforcement learning algorithms have been demonstrated to achieve excellent results on a range of complex tasks, including playing games [66, 72] and continuous control [83, 58, 56]. Prior deep reinforcement learning algorithms either learn state or state-action value functions [65], learn policies using policy gradients [83], or perform a combination of the two using actor-critic architectures [64].

Methods based on learning Q-functions can use replay buffers to include off-policy data, accelerating learning [58]. However, learning Q-functions with Bellman error minimization typically requires a Markovian state space. When learning from partial observations such as images, the inputs might not be Markovian. [65] introduced the concatenation of short observation sequences. Prior methods [36, 72, 64, 38] have proposed to mitigate this issue by learning recurrent critics and Q-functions, that depend on entire histories of observations. The recurrent deterministic policy gradient [38] for partially observable control uses a similar DDPG-style estimator as used in ARM (see Section 2.3).

Policy gradient methods typically do not need to assume a Markovian state, but tend to suffer from poor sample complexity, due to their inability to use off-policy data. However, all of these changes increase the size of the input space, increase variance, or make the optimization problem more complex. Our method instead learns cumulative advantage functions that depend only on the current observation, but can still handle non-Markovian problems.

Other related work

The form of our advantage function update resembles positive temporal difference methods [74, 35]. Additionally, our update rule for a modified cumulative Q-function resembles the average Q-function [1] used for variance reduction in Q-learning. In both cases, the theoretical foundations of our method are based on cumulative regret minimization, and the motivation is substantively different. Previous work by [78, 77] has connected regret minimization to reinforcement learning, imitation learning, and structured prediction, although not with counterfactual regret minimization. Regression regret matching [98] is based on a closely related idea, which is to directly approximate the regret with a linear regression model, however the use of a linear model is limited in representation compared to deep function approximation.

Finally, regret and value functions both typically take the form of expectations in reinforcement learning. An alternative view of value functions in RL is through the lens of value distributions [2]. A connection between regret and distributional RL could lead to very interesting future work.

2.3 Advantage-based Regret Minimization

In this section, we provide some background on counterfactual regret minimization, describe ARM in detail, and give some intuition for why ARM works.

Counterfactual Regret Minimization (CFR)

First, we give a reinforcement learning-centric exposition of counterfactual regret minimization (CFR) [104, 6].

The CFR model for partial observation is as follows. Let $\mathcal{S}$ be the state space. Let $\mathcal{I}$ be the space of *information sets*: an information set $I \in \mathcal{I}$ is a set of states $s \in I$, where $s \in \mathcal{S}$, such that only the information set I is directly observable, and the individual states contained in I are hidden. An information set I is therefore a kind of *aggregate state*.

In CFR, an extensive-form game is repeatedly played between N players, indexed by i. We consider an iterative learning setting, where at the t-th learning iteration, the i-th player follows a fixed policy π_t^i, choosing actions $a \in \mathcal{A}$ conditioned on information sets $I \in \mathcal{I}$

according to their policy $\pi_t^i(a|I)$. Let σ_t denotes the players' joint policy (their strategy profile). For any strategy profile σ, the i-th player has an expected value for playing the game, $J^i(\sigma)$. Let σ_t^{-i} represent the joint policy of all players except the i-th player; similarly, let the tuple (π_t^i, σ_t^{-i}) be the product of the i-th player's policy π_t^i and the other players' joint policy σ_t^{-i}.

The i-th player's *overall regret* after T learning iterations is defined:

$$(R_T^i)^{(\text{overall})} = \max_{\pi_*^i} \sum_{t=1}^{T} J^i(\pi_*^i, \sigma_t^{-i}) - J^i(\pi_t^i, \sigma_t^{-i}). \tag{2.1}$$

What is the interpretation of the overall regret? It is essentially how much better the i-th player could have done, had it always followed an optimal policy in hindsight instead of the actual sequence of policies π_t^i it executed. Intuitively, the difference $J^i(\pi_*^i, \sigma_t^{-i}) - J^i(\pi_t^i, \sigma_t)$ is the suboptimality of π_t^i compared to π_*^i, and the sum of the suboptimalities over learning iterations yields the area inside the learning curve. In other words, a smaller overall regret implies better sample efficiency.

Let $Q_{\sigma_t}^i(I, a)$ be the *counterfactual value* of the i-th player, where the i-th player is assumed to reach I and always chooses the action a in the aggregate state I, and otherwise follows the policy π_t^i, while all other players follow the strategy profile σ_t^{-i}. Similarly, let the expected counterfactual value be calculated as $V_{\sigma_t}^i(I) = \sum_{a \in \mathcal{A}} \pi_t^i(a|I) Q_{\sigma_t}^i(I, a)$. Let $(R_T^i)^{(\text{CF})}(I, a)$ be the *counterfactual regret* of the i-th player, which is the sum of the advantage-like quantities $Q_{\sigma_t}^i(I, a) - V_{\sigma_t}^i(I)$ after T learning iterations:

$$(R_T^i)^{(\text{CF})}(I, a) = \sum_{t=1}^{T} Q_{\sigma_t}^i(I, a) - V_{\sigma_t}^i(I). \tag{2.2}$$

Similarly, the *immediate counterfactual regret* can be obtained from the counterfactual regret by maximization over the player's action space:

$$(R_T^i)^{(\text{immCF})}(I) = \max_{a \in \mathcal{A}} \sum_{t=1}^{T} Q_{\sigma_t}^i(I, a) - V_{\sigma_t}^i(I). \tag{2.3}$$

The immediate counterfactual regret (Equation (2.3)) possesses a similar interpretation to the overall regret (Equation (2.1)), except that the immediate counterfactual regret and its constituent terms are additionally functions of the aggregate state I.

Suppose that one were to briefly consider each aggregate state I as a separate, independent subproblem. By naively treating the counterfactual regret $(R_T^i)^{(\text{CF})}(I, a)$ for each I as analogous to regret in an online learning setting, then at each learning iteration one may simply plug the counterfactual regret into a regret matching policy update [33]:

$$(\pi_{t+1}^i)^{\text{RM}}(a|I) = \frac{\max(0, (R_t^i)^{(\text{CF})}(I, a))}{\sum_{a' \in \mathcal{A}} \max(0, (R_t^i)^{(\text{CF})}(I, a'))}. \tag{2.4}$$

In the online learning setting, the regret matching policy achieves regret that increases with rate $O(\sqrt{T})$ in the number of iterations T [9, 26]; we can then say that the regret matching policy is *regret minimizing.*

It turns out that updating players' policies in the extensive game setting by iteratively minimizing the immediate counterfactual regrets according to Equation (2.4) will also minimize the overall regret $(R^i_T)^{(\text{overall})}$ with upper bound $O(|\mathcal{I}|\sqrt{|\mathcal{A}|T})$. The overall regret's $O(\sqrt{T})$ dependence on the number of iterations T is not impacted by the structure of the information set space $\mathcal{I}$, which is why CFR can be said to be robust to a certain kind of partial observability. This result forms the basis of the counterfactual regret minimization algorithm and was proved by [104].

Since we are interested in the application of CFR to reinforcement learning, we can write down "1-player" versions of the components above: the counterfactual value, reinterpreted as a *stationary* state-action value function $Q_{\pi|I\mapsto a}(I,a)$, where the action a is always chosen in the aggregate state I, and the policy π is otherwise followed [4]; the counterfactual regret, including its recursive definition:

$$\begin{aligned} R^{(\text{CF})}_T(I,a) &= \sum_{t=1}^{T} Q_{\pi_t|I\mapsto a}(I,a) - V_{\pi_t|I}(I) && (2.5)\\ &= R^{(\text{CF})}_{T-1}(I,a) + Q_{\pi_T|I\mapsto a}(I,a) - V_{\pi_T|I}(I) && (2.6) \end{aligned}$$

where $V_{\pi_t|I}(I) = \sum_{a\in\mathcal{A}} \pi_t(a|I) Q_{\pi_t|I\mapsto a}(I,a)$; and the regret matching policy update:

$$\pi^{\text{RM}}_{t+1}(a|I) = \frac{\max(0, R^{(\text{CF})}_t(I,a))}{\sum_{a'\in\mathcal{A}} \max(0, R^{(\text{CF})}_t(I,a'))}. \tag{2.7}$$

CFR+

CFR+ [89] consists of a modification to CFR, in which instead of calculating the full counterfactual regret as in Equation (2.6), instead the counterfactual regret is recursively positively clipped:

$$\begin{aligned} &R^{(\text{CF+})}_T(I,a) && (2.8)\\ &= [R^{(\text{CF+})}_{T-1}(I,a)]_+ + Q_{\pi_T|I\mapsto a}(I,a) - V_{\pi_T|I}(I) \end{aligned}$$

where $[x]_+ = \max(0,x)$ is the positive clipping operator. Comparing Equation (2.6) with Equation (2.8), the only difference in CFR+ is that the previous iteration's quantity is positively clipped in the recursion. This simple change turns out to yield a large practical improvement in the performance of the algorithm [6]. One intuition for why the positive clipping of CFR+ can improve upon CFR is that because $[R^{(\text{CF+})}_{T-1}]_+$ is nonnegative, it provides a kind of "optimism under uncertainty," adding a bonus to some transitions while ignoring others. The CFR+ update has also been shown to do better than CFR when the best action in hindsight changes frequently [90]. In the rest of this work we will solely build upon the CFR+ update.

From CFR and CFR+ to ARM

Recall that CFR and CFR+ model partial observation in extensive games with a space $\mathcal{I}$ of information sets or aggregate states. Because we are interested in general observation spaces $\mathcal{O}$ in partially observed Markov decision processes (MDPs), we naively map CFR and CFR+ onto MDPs and replace $\mathcal{I}$ with $\mathcal{O}$. It is known that Markov or stochastic games can be converted to extensive form [59, 81, 55], and an MDP is a 1-player Markov game. Because we are also interested in high dimensional observations such as images, we estimate a counterfactual regret function approximation [98] parameterized as a neural network.

The exact form of the counterfactual regret (either Equation (2.6) or Equation (2.8)), after translating aggregate states I to observations o, utilizes a stationary action-value function $Q_{\pi|o\mapsto a}(o,a)$. It is unclear how to learn a truly stationary action-value function, although [4] propose a family of consistent Bellman operators for learning locally stationary action-value functions. We posit that the approximation $Q_{\pi|o\mapsto a}(o,a) \approx Q_{\pi}(o,a)$, where $Q_{\pi}(o,a)$ is the usual action-value function, is acceptable when observations are rarely seen more than once in a typical trajectory.

The above series of approximations finally yields a learnable function using deep reinforcement learning; this function is a "clipped cumulative advantage function":

$$A_T^+(o,a) = \max(0, A_{T-1}^+(o,a)) + A_{\pi_T}(o,a) \tag{2.9}$$

where $A_\pi(o,a)$ is the usual advantage function. Advantage-based regret minimization (ARM) is then the resulting batch-mode deep RL algorithm that updates the policy to the regret matching distribution on the cumulative clipped advantage function:

$$\pi_{t+1}(a|o) = \frac{\max(0, A_t^+(o,a))}{\sum_{a'\in\mathcal{A}} \max(0, A_t^+(o,a'))}. \tag{2.10}$$

At the t-th batch iteration of ARM, a batch of data is collected by sampling trajectories using the current policy π_t, followed by two processing steps: (a) fit A_t^+ using Equation (2.9), then (b) set the next iteration's policy π_{t+1} using Equation (2.10).

Below, we will use the subscript k to refer to a timestep within a trajectory, while the subscript t refers to the batch iteration. To implement Equation (2.9) with deep function approximation, we define two value function approximations, $V_{\pi_t}(o_k;\theta_t)$ and $Q_t^+(o_k,a_k;\omega_t)$, as well as a target value function $V'(o_k;\varphi)$, where θ_t, ω_t, and φ are the learnable parameters. The cumulative clipped advantage function is represented as $A_t^+(o_k,a_k) = Q_t^+(o_k,a_k;\omega_t) - V_{\pi_t}(o_k;\theta_t)$. Within each sampling iteration, the value functions are fitted using stochastic gradient descent by sampling minibatches and performing gradient steps. In practice, we use

Adam to perform the optimization [51]. The state-value function $V_{\pi_t}(o_k;\theta_t)$ is fit using n-step returns with a moving target value function $V'(o_{k+n};\varphi)$, essentially using the estimator of the deep deterministic policy gradient (DDPG) [58]. In the same minibatch, $Q^+_t(o_k,a_k;\theta_t)$ is fit to a similar loss, but with an additional target reward bonus that incorporates the previous iteration's cumulative clipped advantage, $\max(0,A^+_{t-1}(o_k,a_k))$. The regression targets v_k and q^+_k are defined in terms of the n-step returns $g^n_k=\sum_{k'=k}^{k+n-1}\gamma^{k'-k}r_{k'}+\gamma^n V'(o_{k+n};\varphi)$:

$$v_k \triangleq g^n_k \tag{2.11}$$
$$q_k \triangleq r_k+\gamma g^{n-1}_{k+1} \tag{2.12}$$
$$\phi_k \triangleq Q^+_{t-1}(o_k,a_k;\omega_{t-1})-V_{\pi_{t-1}}(o_k;\theta_{t-1}) \tag{2.13}$$
$$q^+_k \triangleq \max(0,\phi_k)+q_k. \tag{2.14}$$

Altogether, each minibatch step of the optimization subproblem consists of the following three parameter updates:

$$\theta^{(\ell+1)}_t \leftarrow \theta^{(\ell)}_t-\frac{\alpha}{2}\nabla_{\theta^{(\ell)}_t}(V_{\pi_t}(o_k;\theta^{(\ell)}_t)-v_k)^2 \tag{2.15}$$
$$\omega^{(\ell+1)}_t \leftarrow \omega^{(\ell)}_t-\frac{\alpha}{2}\nabla_{\omega^{(\ell)}_t}(Q^+_t(o_k,a_k;\omega^{(\ell)}_t)-q^+_k)^2 \tag{2.16}$$
$$\varphi^{(\ell+1)} \leftarrow \varphi^{(\ell)}+\tau(\theta^{(\ell+1)}_t-\varphi^{(\ell)}). \tag{2.17}$$

The advantage-based regret minimization algorithm is summarized in Algorithm 1.

We note again that we use a biased value function estimator, whereas only the full returns are guaranteed to be unbiased in non-Markovian settings. In the high-dimensional domains we evaluated on, the most practical choice of deep advantage or value function approximation is based on biased but lower variance estimation, such as the n-step returns we use in practice [84, 29]. We also note that ARM is essentially CFR with different design choices to facilitate function approximation. CFR is proven to converge in domains with non-Markovian information set spaces when using tabular representations, suggesting that our use of biased advantage estimation does not fundamentally preclude the applicability or effectiveness of ARM on non-Markovian domains.

ARM vs. Existing Policy Gradient Methods

We note that the fundamental operation in CFR-like algorithms, including ARM, is exemplified by the counterfactual regret update in Equation (2.6) which superficially looks quite similar to a policy gradient-style update: the update is in the direction of a Q-value minus a baseline. However, we can take the analogy between ARM and policy gradient methods further and show that ARM represents an inherently different update compared to existing policy gradient methods.

Recent work has shown that policy gradient methods and Q-learning methods are connected via maximum entropy RL [71, 31, 67, 82]. One such perspective is from the soft policy

Algorithm 1 Advantage-based regret minimization (ARM).

```
initialize $\pi_1 \leftarrow$ uniform, $\theta_0, \omega_0 \leftarrow$ arbitrary
for $t$ in $1, \ldots$ do
    collect batch of trajectory data $\mathcal{D}_t \sim \pi_t$
    initialize $\theta_t \leftarrow \theta_{t-1}$, $\omega_t \leftarrow \omega_{t-1}$, $\varphi \leftarrow \theta_{t-1}$
    for $\ell$ in $0, \ldots$ do
        sample transitions:
        $(o_k, a_k, r_k, o_{k+1}) \sim \mathcal{D}_t$
        $\delta_k \leftarrow 1 - \mathbb{I}[o_k \text{ is terminal}]$
        calculate $n$-step returns:
        $g_k^n \leftarrow \sum_{k'=k}^{k+n-1} \gamma^{k'-k} r_{k'} + \gamma^n \delta_{k+n} V'(o_{k+n}; \varphi)$
        calculate target values:
        if $t = 1$ then
            $\phi_k \leftarrow 0$
        else
            $\phi_k \leftarrow Q_{t-1}^{+}(o_k, a_k; \omega_{t-1}) - V_{\pi_{t-1}}(o_k; \theta_{t-1})$
        end if
        $v(o_k) \leftarrow g_k^n$
        $q^{+}(o_k, a_k) \leftarrow \max(0, \phi_k) + r_k + \gamma g_{k+1}^{n-1}$
        update $\theta_t$ (Equation (2.15))
        update $\omega_t$ (Equation (2.16))
        update $\varphi$ (Equation (2.17))
    end for
    set $\pi_{t+1}(a|o) \propto \max(0, Q_t^{+}(o, a; \omega_t) - V_{\pi_t}(o; \theta_t))$
end for
```

iteration framework for batch-mode reinforcement learning [32], where at the t-th batch iteration the updated policy is obtained by minimizing the average KL-divergence between the parametric policy class Π and a target policy f_t. Below is the soft policy iteration update, where the subscript t refers to the batch iteration:

$$\begin{aligned}\pi_{t+1} &= \arg\min_{\pi \in \Pi} \mathbb{E}_{o \sim \rho_t}[D_{\mathrm{KL}}(\pi(\cdot|o) \| f_t(\cdot|o))] \\ &= \arg\min_{\pi \in \Pi} \mathbb{E}_{o \sim \rho_t, a \sim \pi(\cdot|o)}[\log(\pi(a|o)) - \log(f_t(a|o))]. \end{aligned} \tag{2.18}$$

Using the connection between policy gradient methods and Q-learning, we define the policy gradient target policy as a Boltzmann or softmax distribution on the entropy regularized advantage function $A^{\beta\text{-soft}}$:

$$f_t^{\mathrm{PG}}(a|o) \triangleq \frac{\exp(\beta A_t^{\beta\text{-soft}}(o, a))}{\sum_{a' \in \mathcal{A}} \exp(\beta A_t^{\beta\text{-soft}}(o, a'))}. \tag{2.19}$$

Now, parameterizing the policy π in terms of an explicit parameter θ, we obtain the expression for the existing policy gradient update, where $b(o)$ is a baseline function:

$$\begin{aligned}\Delta\theta^{\mathrm{PG}} \propto \mathbb{E}_{o\sim\rho_t,a\sim\pi(\cdot|o;\theta)}\Big[&\nabla_\theta \log(\pi(o|a;\theta))\cdot \\ &\big(-(1/\beta)\log(\pi(o|a;\theta)) \\ &\quad + A_t^{\beta\text{-soft}}(o,a) - b(o)\big)\Big].\end{aligned} \tag{2.20}$$

The non-entropy-regularized policy gradient arises in the limit $\beta \to \infty$, at which point $(1/\beta)$ vanishes.

Note that an alternative choice of target policy f_t will lead to a different kind of policy gradient update. A policy gradient algorithm based on ARM instead proposes a target policy based on the regret matching distribution:

$$f_t^{\mathrm{ARM}}(a|o) \triangleq \frac{\max(0, A_t^+(o,a))}{\sum_{a'\in\mathcal{A}}\max(0, A_t^+(o,a'))}. \tag{2.21}$$

Similarly, we can express the ARM-like policy gradient update, where again $b(o)$ is a baseline:

$$\begin{aligned}\Delta\theta^{\mathrm{ARM}} = \mathbb{E}_{o\sim\rho_t,a\sim\pi(\cdot|o;\theta)}\Big[&\nabla_\theta \log(\pi(o|a;\theta))\cdot \\ &\big(-\log(\pi(o|a;\theta)) \\ &\quad + \log(\max(0, A_t^+(o,a))) - b(o)\big)\Big].\end{aligned} \tag{2.22}$$

Comparing Equations (2.20) and (2.22), we see that the ARM-like policy gradient (Equation (2.22)) has a logarithmic dependence on the clipped advantage-like function $\max(0, A^+)$, whereas the existing policy gradient (Equation (2.20)) is only linearly dependent on the advantage function $A^{\beta\text{-soft}}$. This difference in logarithmic vs. linear dependence is responsible for a large part of the inherent distinction of ARM from existing policy gradient methods. In particular, the logarithmic dependence in an ARM-like update may be less sensitive to advantage overestimation compared to existing policy gradient methods, perhaps serving a similar purpose to the double Q-learning estimator [34] or consistent Bellman operators [4].

We also see that for the existing policy gradient (Equation (2.20)), the $-(1/\beta)\log(\pi(a|o;\theta))$ term, which arises from the policy entropy, is vanishing for large β. On the other hand, for the ARM-like policy gradient (Equation (2.22)), there is no similar vanishing effect on the equivalent policy entropy term, suggesting that ARM may perform a kind of entropy regularization by default.

In practice we cannot implement an ARM-like policy gradient exactly as in Equation (2.22), because the positive clipping $\max(0, A^+)$ can yield $\log(0)$. However we believe this is not an intrinsic obstacle and leave the question of how to implement an ARM-like policy gradient to future work.

Comparing ARM with Other Methods in Partially Observable Domains

The regret matching policy which is fundamental to ARM can be interpreted as a more nuanced form of exploration compared to the epsilon-greedy policy used with Q-learning. In partially observable domains, the optimal policy may generally be stochastic [42]. So ϵ-greedy policies which put a substantial probability mass on $\arg\max_a Q(o, a)$, e.g. by setting $\epsilon = 0.01$, can be suboptimal, especially compared to more general distributions on discrete actions, such as ARM's regret matching policy. The softmax policy typically learned by policy gradient methods is also quite general, but can still put too much probability mass on one action without compensation by an explicit entropy bonus as done in maximum entropy reinforcement learning.

Policy gradient methods have an overall regret bound of $R_T \leq B^2/\eta + \eta G^2 T$ derived from stochastic gradient descent, where B is an upper bound on the policy parameter ℓ^2-norm, G^2 is an upper bound on the second moments of the stochastic gradients, and η is the learning rate [20], Assuming an optimal learning rate $\eta = B/(G\sqrt{T})$, the policy gradient regret bound becomes $R_T \leq 2BG\sqrt{T}$. CFR has an overall regret bound of $R_T \leq \Delta|\mathcal{I}|\sqrt{|\mathcal{A}|T}$, where Δ is the positive range of returns, and $|\mathcal{I}|$ and $|\mathcal{A}|$ are cardinalities of the information set space and action space, respectively [104]. Both regret bounds have $O(\sqrt{T})$ dependence on T. As mentioned earlier, policy gradient methods can also be robust to partial observation, but unlike in the case of CFR, the dependence of the policy gradient regret bound on an optimal learning rate and on gradient variance may affect policy gradient convergence in the presence of estimation error. On the other hand, the constants in the CFR regret bound are constant properties of the environment.

For Q-learning per se, we are not aware of any known regret bound. Szepesvári proved that an upper bound on the convergence rate of Q-learning (specifically, the convergence rate of $\|Q_T(s, a) - Q^*(s, a)\|_\infty$), assuming a fixed exploration strategy, depends on an exploration condition number, which is the ratio of minimum to maximum state-action occupation frequencies [88], and which describes how "balanced" the exploration strategy is. If partial observability leads to imbalanced exploration due to confounding of states from perceptual aliasing [62], then Q-learning should be negatively affected in convergence and possibly in absolute performance.

2.4 Experiments

Because we hypothesize that ARM should perform well in partially observable domains, we conduct our experiments on visual tasks that naturally provide partial observations of state. Our evaluations use feedforward convnets with frame history inputs; our hyperparameters are listed in Section 2.5. We are interested in comparing ARM with other advantage-structured methods, primarily: (a) double deep Q-learning with dueling network streams [34, 97], which possesses an advantage-like parameterization of its Q-function and assumes Markovian observations; and (b) TRPO [83, 84], which estimates an empirical advantage using a baseline state-value function and can handle non-Markovian observations.

Learning First-person 3D Navigation

We first evaluate ARM on the task of learning first-person navigation in 3D maze-like tasks from two domains: ViZDoom [50] based on the game Doom, and Malmö [46] based on the game Minecraft. Doom and Minecraft both feature an egocentric viewpoint, 3D perspective, and rich visual textures. We expect that both domains exhibit a substantial degree of partial observability since only the immediate field-of-view of the environment is observable by the agent due to first-person perspective.

In Doom MyWayHome, the agent is randomly placed in one of several rooms connected in a maze-like arrangement, and the agent must reach an item that has a fixed visual appearance and is in a fixed location before time runs out. For Minecraft, we adopt the teacher-student curriculum learning task of [61], consisting of 5 consecutive "levels" that successively increase the difficulty of completing the simple task of reaching a gold block: the first level ("L1") consists of a single room; the intermediate levels ("L2"–"L4") consist of a corridor with lava-bridge and wall-gap obstacles; and the final level ("L5") consists of a 2×2 arrangement of rooms randomly separated by lava-bridge or wall-gap obstacles. Examples of the MyWayHome and the Minecraft levels are shown in Figure 2.1.

Our results on Doom and Minecraft are in Figures 2.2 and 2.3. Unlike previous evaluations which augmented raw pixel observations with extra information about the game state, e.g. elapsed time ticks or remaining health [50, 21], in our evaluation we forced all networks to learn using only visual input. Despite this restriction, ARM is still able to quickly learn policies with minimal tuning of hyperparameters and to reach close to the maximum achievable score in under 1 million simulator steps, which is quite sample efficient. On MyWayHome, we observed that ARM generally learned a well-performing policy more quickly than other methods. Additionally, we found that ARM is able to take advantage of an off-policy replay memory when learning on MyWayHome by storing the trajectories of previous sampling batches and applying an importance sampling correction to the n-step return estimator; please see Section 2.6.

We performed our Minecraft experiments using fixed curriculum learning schedules to evaluate the sample efficiency of different algorithms: the agent is initially placed in the first level ("L1"), and the agent is advanced to the next level whenever a preselected number of simulator steps have elapsed, until the agent reaches the last level ("L5"). We found that ARM and dueling double DQN both were able to learn on an aggressive "fast" schedule of only 62500 simulator steps between levels. TRPO required a "slow" schedule of 93750 simulator steps between levels to reliably learn. ARM was able to consistently learn a well performing policy on all of the levels, whereas double DQN learned more slowly on some of the intermediate levels. ARM also more consistently reached a high score on the final, most difficult level ("L5").

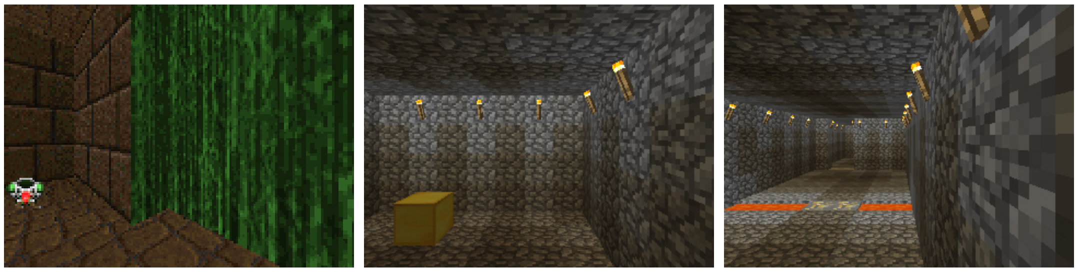

Figure 2.1: Screenshots from (left) Doom MyWayHome, (middle) Minecraft level 1, and (right) Minecraft level 4.

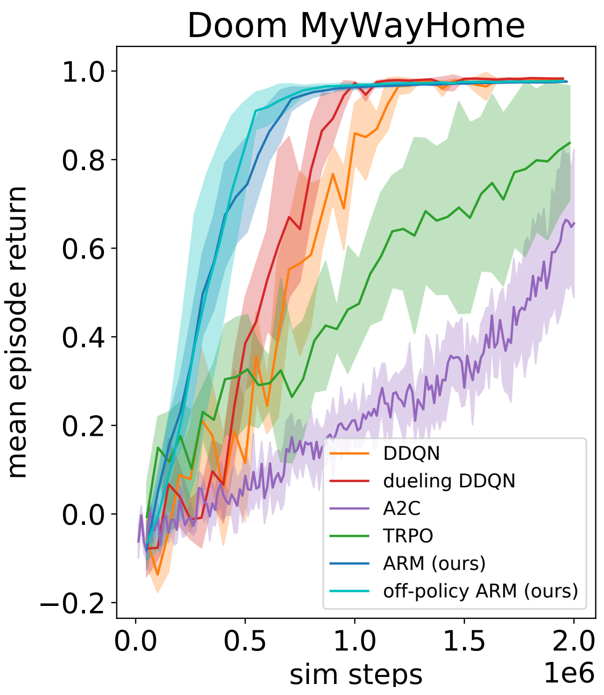

Figure 2.2: Evaluation on the Doom MyWayHome task.

Learning with Partially Observed Objects

Previously in Section 2.3, we argued that the convergence results for CFR suggest that reducing the size of the observation space could improve the convergence of CFR compared to methods based on policy gradients or Q-learning. We would expect that by controlling the

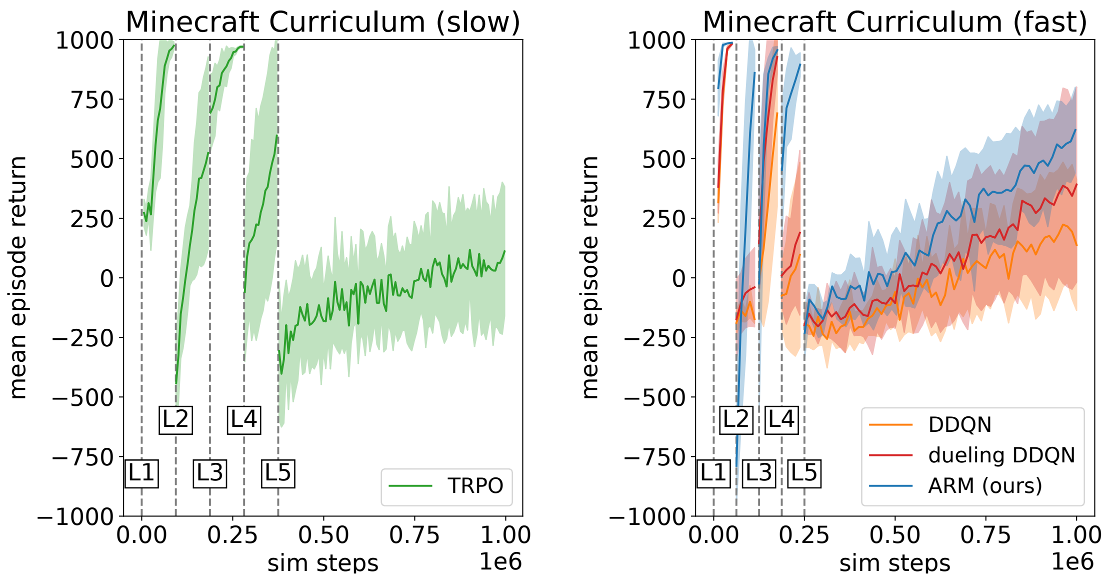

Figure 2.3: Evaluation on a Minecraft curriculum learning task. The simulator step counts at which each level begins are labeled and demarcated with dashed vertical lines.

degree of partial observability in a fixed task, one could expect the relative sample efficiency or performance of ARM to improve compared to other methods.

Whereas the navigation tasks in Section 2.4 only dealt with first-person motion in static 3D environments, tasks that add other objects which may move and are not always visible to the agent intuitively ought to be less observable. To test the effect of the degree of partial observability on fixed tasks with objects, we conduct experiments with at least two variants of each task: one experiment is based on the unmodified task, but the other experiments occlude or mask essential pixels in observed frames to raise the difficulty of identifying relevant objects.

We evaluate ARM and other methods on two tasks with objects: Doom Corridor via ViZDoom, and Atari Pong via the Arcade Learning Environment [3]. In Corridor, the agent is placed in a narrow 3D hallway and must avoid getting killed by any of several shooting monsters scattered along the route while trying to reach the end of the hallway. In Pong, the agent controls a paddle and must hit a moving ball past the computer opponent's paddle on the other side of the screen.

In our implementation of Corridor, which we call "Corridor+," we restrict the action space of the agent to only be able to turn left or right and to move to one side. Because the agent spawns facing the end of the hallway, the agent has to learn to first turn to its side to orient itself sideways before moving, while also trying to avoid getting hit from behind by monsters in close proximity. To induce partial observability in Corridor, we mask out a large square region in the center of the frame buffer. An example of this occlusion is shown in Figure 2.4. Because the only way for the agent to progress toward the end of the hallway is by moving sideways, the monsters that are closest to the agent, and hence the most dangerous, will appear near the center of the agent's field-of-view. So the center pixels are the most natural ones to occlude in Corridor+.

For Pong, we choose a rectangular area in the middle of the frame between the two players' paddles, then we set all pixels in the rectangular area to the background color. An illustration of the occlusion is shown in Figure 2.4. When the ball enters the occluded region, it completely disappears from view so its state cannot be reconstructed using a limited frame history. Intuitively, the agent needs to learn to anticipate the trajectory of the ball in the absence of visual cues of the ball's position and velocity.

It is also known that for Atari games in general, with only 4 observed frames as input, it is possible to predict hundreds of frames into the future on some games using only a feedforward dynamics model [73]. By default, all of our networks receive as input a frame history of length 4. This suggests that limiting the frame history length in Pong is another effective perturbation for reducing observability. When the frame history length is limited to one, then reconstructing the velocity of moving objects in Pong becomes more difficult [36].

Our results on Corridor+ are shown in Figure 2.5. Among all our experiments, it is on Corridor+ that TRPO performs the best compared to all other methods. One distinguishing feature of TRPO is its empirical full return estimator, which is unbiased on non-Markovian observations, but which can have greater variance than the n-step return estimator used in deep Q-learning and ARM. On Corridor+, there appears to be a benefit to using the unbiased full returns over the biased n-step returns, which speaks to the non-Markovian character of the Corridor+ task. It is also evident on Corridor+ that the performance of deep Q-learning suffers when the observability is reduced by occlusion, and that by comparison ARM is relatively unaffected, despite both methods using biased n-step returns with the same value of n ($n = 5$). This suggests that even when ARM is handicapped by the bias of its return estimator, ARM is intrinsically more robust to the non-Markovian observations that arise from partial observability. One possible direction to improve ARM is through alternative return estimators [84, 96].

Our results on Pong are shown in Figure 2.6. The convergence of ARM on all three variants of Pong suggests that ARM is not affected much by partial observation in this domain. As expected, when observability is reduced in the limited frame history and occlusion experiments, deep Q-learning performs worse and converges more slowly than ARM. TRPO is generally less sample efficient than either ARM or DQN, although like ARM it seems to be resilient to the partial observation perturbations.

2.5 Experimental Details

For each experiment, we performed 5 trials for each method (some TRPO experiments on Pong are for 3 trials).

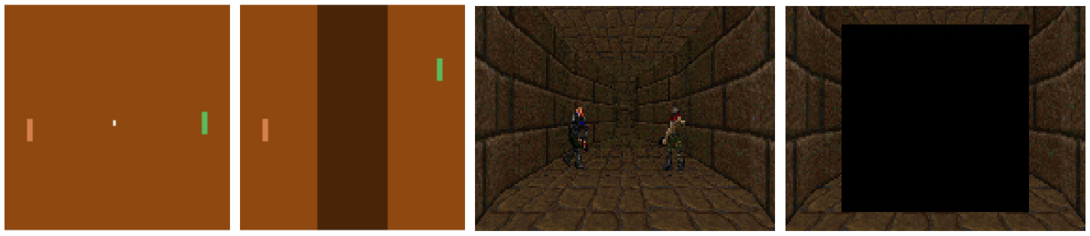

Figure 2.4: Screenshots from Atari Pong (left two) and Doom Corridor+ (right two), unmodified and with occlusion.

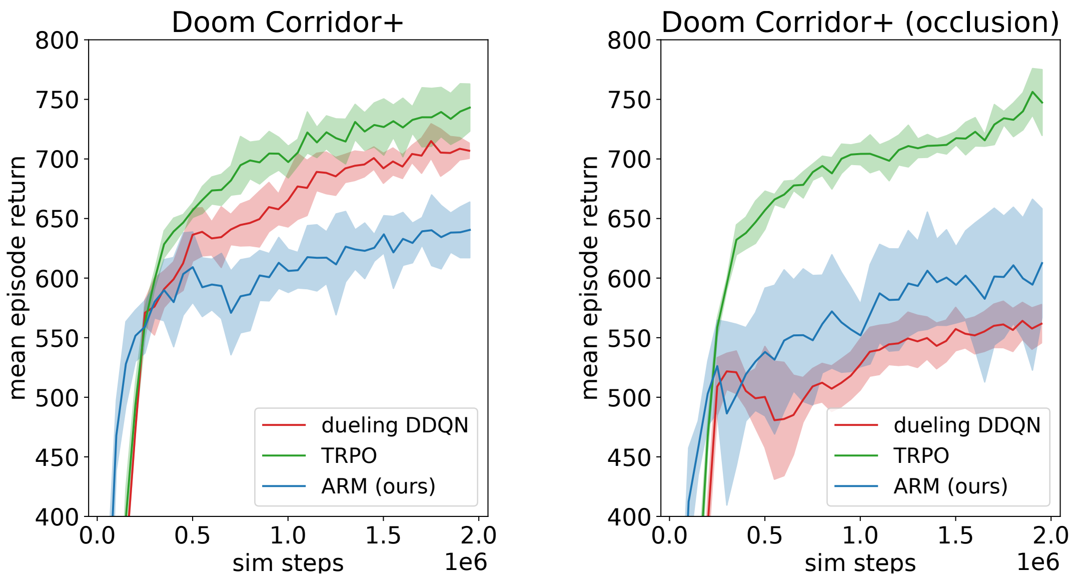

Figure 2.5: Results on Doom Corridor+: (left) unmodified, (right) with occlusion.

Hyperparameters

Our hyperparameter choices for each method are listed below. For all methods using Adam, we roughly tuned the learning rate to find the largest that consistently converged in unmodified variants of tasks.

ARM

Please see Tables 2.1 and 2.2 for hyperparameters used with ARM. Note that our choice of ARM hyperparameters yields an equivalent number of minibatch gradient steps per sample as used by deep Q-learning, i.e. 1 Adam minibatch gradient step per 4 simulator steps; c.f. Table 2.4 for the deep Q-learning hyperparameters. We kept hyperparameters (other than the learning rate and the number of steps n) constant across tasks.

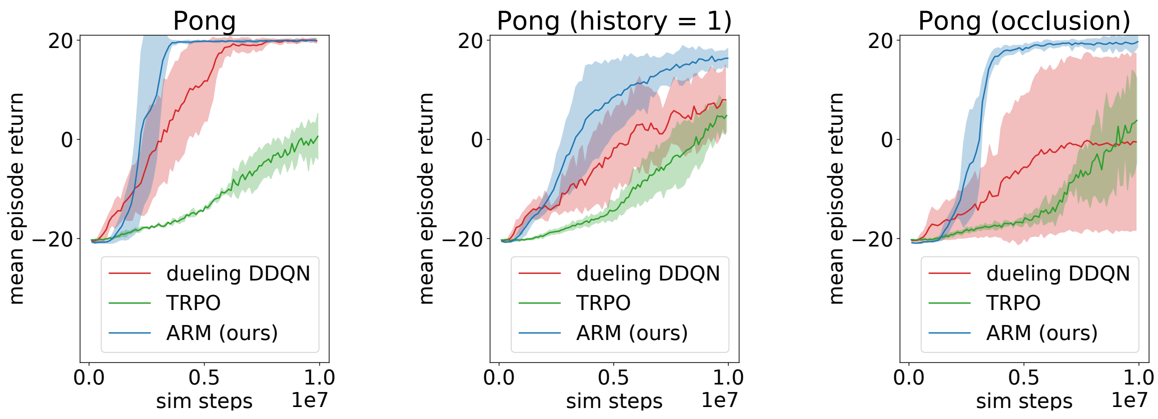

Figure 2.6: Results on Atari Pong: (left) unmodified, (middle) with limited frame history, (right) with occlusion.

Table 2.1: Hyperparameters for ARM.

Hyperparameter	Atari	Doom	Minecraft
Adam learning rate	$1e^{-4}$	$1e^{-5}$	$1e^{-5}$
Adam minibatch size	32	32	32
batch size	12500	12500	12500
gradient steps	3000	3000	3000
moving average (τ)	0.01	0.01	0.01
n-steps	1	5	5

A2C

Please see Table 2.3 for hyperparameters used with A2C. We found that increasing the number of steps n used to calculate the n-step returns was most important for getting A2C/A3C to converge on Doom MyWayHome.

DQN

Please see Table 2.4 for hyperparameters used with deep Q-learning. Dueling double DQN uses the tuned hyperparameters [34, 97]. In particular, we found that dueling double DQN generally performed better and was more stable when learning on Atari with the tuned learning rate $6.25 \times 10^{-5} \approx 6 \times 10^{-5}$ from [97], compared to the slightly larger learning rate of 1×10^{-4} used by ARM.

Table 2.2: Hyperparameters for off-policy ARM.

Hyperparameter	Doom
Adam learning rate	1e^{-5}
Adam minibatch size	32
batch size	1563
gradient steps	400
importance weight clip	1
moving average (τ)	0.01
n-steps	5
replay memory max	25000

Table 2.3: Hyperparameters for A2C.

Hyperparameter	Doom
Adam learning rate	1e^{-4}
Adam minibatch size	640
entropy bonus (β)	0.01
gradient clip	0.5
n-steps	40
num. workers	16

TRPO

Please see Table 2.5 for hyperparameters used with TRPO. We generally used the defaults, such as the KL step size of 0.01 which we found to be a good default. Decreasing the batch size improved sample efficiency on Doom and Minecraft without adversely affecting the performance of the learned policies.

Environment and Task Details

Our task-specific implementation details are described below.

Atari

For the occluded variant of Pong, we set the middle region of the 160 $\times$ 210 screen with x, y pixel coordinates $[55 \ldots 105), [34 \ldots 194)$ to the RGB color $(144, 72, 17)$. The image of occluded Pong in Figure 4 from the main text has a slightly darker occluded region for emphasis.

Table 2.4: Hyperparameters for dueling + double deep Q-learning.

Hyperparameter	Atari	Doom	Minecraft
Adam learning rate	$6e^{-5}$	$1e^{-5}$	$1e^{-5}$
Adam minibatch size	32	32	32
final exploration	0.01	0.01	0.01
gradient clip	10	—	—
n-steps	1	5	5
replay memory init	50000	50000	12500
replay memory max	10^6	240000	62500
sim steps/grad step	4	4	4
target update steps	30000	30000	12500

Table 2.5: Hyperparameters for TRPO.

Hyperparameter	Atari	Doom	Minecraft
batch size	100000	12500	6250
CG dampening	0.1	0.1	0.1
CG iterations	10	10	10
KL step size	0.01	0.01	0.01

We use the preprocessing and convolutional network model of Mnih et al. (2013). Specifically, we view every 4th emulator frame, convert the raw frames to grayscale, and perform downsampling to generate a single observed frame. The input observation of the convnet is a concatenation of the most recent frames (either 4 frames or 1 frame). The convnet consists of an 8×8 convolution with stride 4 and 16 filters followed by ReLU, a 4×4 convolution with stride 2 and 32 filters followed by ReLU, a linear map with 256 units followed by ReLU, and a linear map with $|\mathcal{A}|$ units where $|\mathcal{A}|$ is the action space cardinality ($|\mathcal{A}| = 6$ for Pong).

Doom

Our modified environment “Doom Corridor+” is very closely derived from the default “Doom Corridor” environment in ViZDoom. We primarily make two modifications: (a) first, we restrict the action space to the three keys $\{MoveRight, TurnLeft, TurnRight\}$, for a total of $2^3 = 8$ discrete actions; (b) second, we set the difficulty (“Doom skill”) to the maximum of 5.

For the occluded variant of Corridor+, we set the middle region of the 160×120 screen with x, y pixel coordinates $[30 \ldots 130), [10 \ldots 110)$ to black, i.e. $(0, 0, 0)$.

For Corridor+, we scaled rewards by a factor of 0.01. We did not scale rewards for MyWayHome.

The Doom screen was rendered at a resolution of 160×120 and downsized to 84×84. Only every 4th frame was rendered, and the input observation to the convnet is a concatenation of the last 4 rendered RGB frames for a total of 12 input channels. The convnet contains 3 convolutions with 32 filters each: the first is size 8×8 with stride 4, the second is size 4×4 with stride 2, and the third is size 3×3 with stride 1. The final convolution is followed by a linear map with 1024 units. A second linear map yields the output. Hidden activations are gated by ReLUs.

Minecraft

Our Minecraft tasks are based on the tasks introduced by [61], with a few differences. Instead of using a continuous action space, we used a discrete action space with 4 move and turn actions. To aid learning on the last level ("L5"), we removed the reward penalty upon episode timeout and we increased the timeout on "L5" from 45 seconds to 75 seconds due to the larger size of the environment. We scaled rewards for all levels by 0.001.

We use the same convolutional network architecture for Minecraft as we use for ViZDoom. The Minecraft screen was rendered at a resolution of 320×240 and downsized to 84×84. Only every 5th frame was rendered, and the input observation of the convnet is a concatenation of the last 4 rendered RGB frames for a total of 12 input channels.

2.6 Off-policy ARM via Importance Sampling

Our current approach to running ARM with off-policy data consists of applying an importance sampling correction directly to the n-step returns. Given the behavior policy μ under which the data was sampled, the current policy π_t under which we want to perform estimation, and an importance sampling weight clip c for variance reduction, the corrected n-step return we use is:

$$\begin{aligned} g_k^n(\mu \| \pi_t) = \sum_{k'=k}^{k+n-1} \gamma^{k'-k} \left(\prod_{\ell=k}^{k'} w_{\mu\|\pi_t}(a_\ell | o_\ell) \right) r_{k'} \\ + \gamma^n V'(o_{k+n}; \varphi) \end{aligned} \tag{2.23}$$

where the truncated importance weight $w_{\mu\|\pi_t}(a|o)$ is defined:

$$w_{\mu\|\pi_t}(a|o) = \min\left(c, \frac{\pi_t(a|o)}{\mu(a|o)} \right). \tag{2.24}$$

Note that the target value function $V'(o_{k+n}; \varphi)$ does not require an importance sampling correction because V' already approximates the on-policy value function $V_{\pi_t}(o_{k+n}; \theta_t)$. Our choice of $c = 1$ in our experiments was inspired by [96]. We found that $c = 1$ worked well but note other choices for c may also be reasonable.

When applying our importance sampling correction, we preserve all details of the ARM algorithm except for two aspects: the transition sampling strategy (a finite memory of previous batches are cached and uniformly sampled) and the regression targets for learning the value functions. Specifically, the regression targets v_k and q_k^+ (Equations (11)–(14) in the main text) are modified to the following:

$$v_k = g_k^n(\mu\|\pi_t) \tag{2.25}$$

$$q_k = (1 - w_{\mu\|\pi_t}(a_k|o_k))r_k + g_k^n(\mu\|\pi_t) \tag{2.26}$$

$$\phi_k = Q_{t-1}^+(o_k, a_k; \omega_{t-1}) - V_{\pi_{t-1}}(o_k; \theta_{t-1}) \tag{2.27}$$

$$q_k^+ = \max(0, \phi_k) + q_k. \tag{2.28}$$

2.7 Additional Experiments

Recurrence in Doom MyWayHome

We evaluated the effect of recurrent policy and value function estimation in the maze-like MyWayHome scenario of ViZDoom. For the recurrent policy and value function, we replaced the first fully connected operation with an LSTM featuring an equivalent number of hidden units (1024). We found that recurrence has a small positive effect on the convergence of A2C, but was much less significant than the choice of algorithm; compare Figure 2 in the main text with Figure 2.7 below.

Atari 2600 games

Although our primary interest is in partially observable reinforcement learning domains, we also want to check that ARM works in nearly fully observable and Markovian environments, such as Atari 2600 games. We consider two baselines: double deep Q-learning, and double deep fitted Q-iteration which is a batch counterpart to double DQN. We find that double deep Q-learning is a strong baseline for learning to play Atari games, although ARM still successfully learns interesting policies. One major benefit of Q-learning-based methods is the ability to utilize a large off-policy replay memory. Our results on a suite of Atari games are in Figure 2.8.

2.8 Discussion

In this chapter, we posed the problem of visual navigation in the setting of deep reinforcement learning under partial observation, and we presented a novel deep reinforcement learning

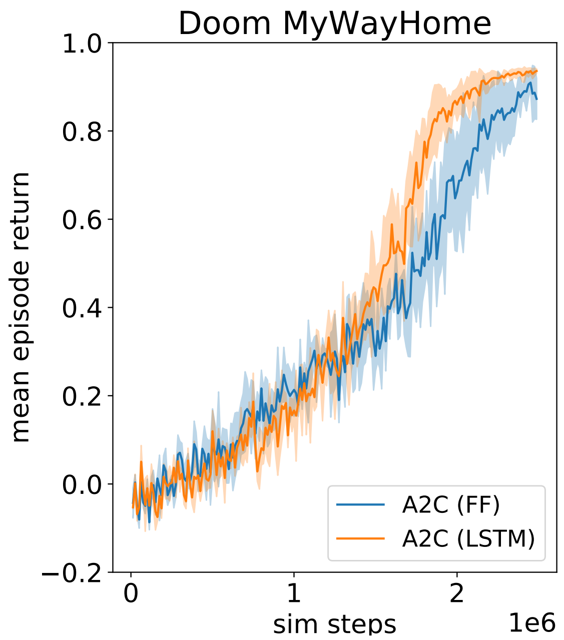

Figure 2.7: Comparing A2C with a feedforward convolutional network (blue) and a recurrent convolutional-LSTM network (orange) on the ViZDoom scenario MyWayHome.

algorithm based on counterfactual regret minimization. We call our method advantage-based regret minimization (ARM). Similarly to prior methods that learn state or state-action value functions, our method learns a cumulative clipped advantage function of observation and action. However, in contrast to these prior methods, ARM is well suited to partially observed or non-Markovian environments, making it an appealing choice in a number of difficult domains. When compared to baseline methods, including deep Q-learning and TRPO, on partially observable tasks such as first-person navigation in Doom and Minecraft and interacting with partially observed objects in Doom and Pong, ARM is robust to the degree of partial observability and can achieve substantially better sample efficiency and performance. This illustrates the value of ARM for partially observable problems.

Some baseline deep reinforcement learning algorithms parameterize exploration in their policies with hyperparameters. For example, maximum entropy RL methods such as soft actor-critic [32] control exploration with a Boltzmann temperature hyperparameter, while deep Q-learning with ϵ-greedy exploration [65] anneals the exploration rate ϵ. In contrast, ARM lacks an explicit hyperparameter that directly controls exploration in its learned policy, instead choosing the CFR+ defaults for updating the regret and the policy [89]. Alternative design choices one may consider in future work concern the regret and policy updates; some possibilities introduced since the publication of our work include laminar regret matching [23], as well as generalized discounting schemes for CFR [8].

One notable disadvantage of ARM is that, as currently formulated, it is not obviously

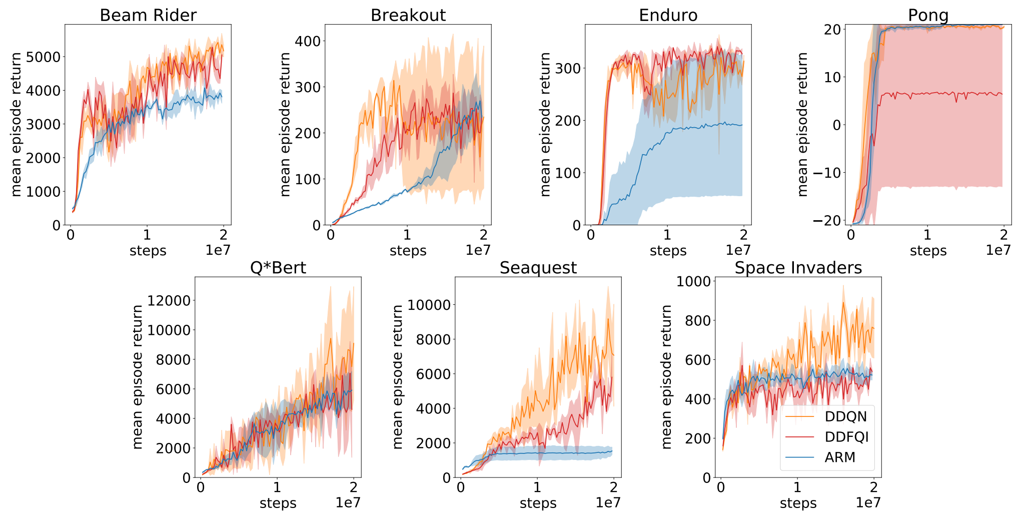

Figure 2.8: Comparing double deep Q-learning (orange), double deep fitted Q-iteration (red), and ARM (blue) on a suite of seven Atari games from the Arcade Learning Environment.

amenable to learning policies on continuous action spaces. In future work, we plan to explore adapting ARM to continuous action spaces and concomitant tasks.

Chapter 3

Gossiping SGD

3.1 Introduction

Another example of an agent that must navigate in a 3D environment is a self-driving car. Self-driving cars fuse data collected by a variety of sensors, including cameras and LIDAR, in order to build a model of their environment and perform their core task of navigation. In this chapter, we shift our attention from the reinforcement learning of visual navigation policies to the supervised learning of visual perception systems for self-driving cars.

In practice, the most common way to train perception systems in practice is through supervised learning on very large scale image datasets. As a result, reducing the training time becomes critical. For example, estimates of the total data gathered by a self-driving car start from at least 750 MB/s.[1] As another data point, a single 1080p 24-bit RGB video camera mounted on a self-driving car and streaming at 60 Hz yields 370 MB/s of uncompressed image data. With proper annotation or through an unsupervised learning scheme, all of this data can become useful for training the object detection system or grid-occupancy system of a self-driving car. The resulting training set can lead to weeks or more of training time on a single CPU/GPU system. Therefore, for such applications training time defines the most time consuming element of the workflow, and reduced training time is highly desirable.

To achieve significant reductions in training time, the training must be distributed across multiple CPUs/GPUs with the goal of strong scaling: as more nodes (i.e. compute servers) are thrown at the problem, the training time should ideally decrease proportionally. There are two primary approaches to distributed stochastic gradient descent (SGD) for training deep neural networks: (i) synchronous all-reduce SGD based on a fast all-reduce collective communication operation [40, 99, 16, 10], and (ii) asynchronous SGD using a parameter server [17, 13].

Both approaches (i) and (ii) have weaknesses at scale. Synchronous SGD is penalized by straggling processors, underutilizes compute resources, and is not robust in the face of failing

[1] `http://www.kurzweilai.net/googles-self-driving-car-gathers-nearly-1-gbsec`

processors or nodes. On the other hand, asynchronous approaches using parameter servers create a communication bottleneck and underutilize the available network resources, slowing convergence.

Individual researchers also have different numbers of nodes at their disposal, including compute and network resources. So determining the best approach for a given number of nodes, as well as the approach that scales to the most number of nodes, is of interest to practitioners with finite resources.

We are concerned with the following questions:

a. How fast do asynchronous and synchronous SGD algorithms converge at both the beginning of training (large step sizes) and at the end of training (small step sizes)?

b. How does the convergence of async. and sync. SGD vary with the number of nodes?

To compare the strengths and weaknesses of asynchronous and synchronous SGD algorithms, we train a modern ResNet convolutional network [37] on the ImageNet dataset [79] using various distributed SGD methods. We primarily compare synchronous all-reduce SGD, the recently proposed asynchronous elastic averaging SGD [103], as well as our own method, asynchronous gossiping SGD, based on an algorithm originally developed in a different problem setting [75]. Gossiping SGD is an asynchronous method that does not use a centralized parameter server, and in a sense, gossiping is a decentralized version of elastic averaging. We find that asynchronous SGD, including both elastic averaging and gossiping, exhibits the best scaling at larger step sizes and, perhaps counterintuitively in the case of gossiping, at smaller scales (up to around 32 distributed nodes). For smaller step sizes and at larger scales, all-reduce consistently converges to the most accurate solution faster than the asynchronous methods.

3.2 Related Work

In this section, we will describe the baseline synchronous and asynchronous SGD methods, as well as a recently proposed asynchronous method that is more scalable than its predecessor. We will use the following naming convention for SGD: θ are the parameters over which the objective is minimized, $\tilde{\theta}$ is the center parameter (if applicable), α is the step size, μ is the momentum, subscript i refers to the i-th node out of p total nodes, and subscript t refers to the t-th (minibatch) iteration. Additionally, b refers to the *per-node* minibatch size, and m refers to the *aggregate* minibatch size summed across all nodes.

Synchronous All-Reduce SGD

In traditional synchronous all-reduce SGD, there are two alternating phases proceeding in lock-step: (1) each node computes its local parameter gradients, and (2) all nodes collectively communicate to calculate the aggregate gradient, as if they all formed a large distributed minibatch. The second phase of exchanging gradients forms a barrier and is the communication-intensive phase, usually implemented by an eponymous all-reduce operation. The time complexity of an all-reduction can be decomposed into latency-bound and bandwidth-bound terms. Although the latency term scales with $O(\log(p))$, there are fast ring algorithms which have bandwidth term independent of p [91]. With modern networks capable of handling bandwidth on the order of 1–10 GB/s combined with neural network parameter sizes on the order of 10–100 MB, the communication of gradients or parameters between nodes across a network can be very fast. Instead, the communication overhead of all-reduce results from its use of a synchronization barrier, where all nodes must wait for all other nodes until the all-reduce is complete before proceeding to the next stochastic gradient iteration. This directly leads to a straggler effect where the slowest nodes will prevent the rest of the nodes from making progress. Examples of large-scale synchronous data parallel SGD for distributed deep learning are given in [40], [99], [16], and [10]. We provide pseudocode for synchronous data-parallel SGD in Algorithm 2.

Algorithm 2 Synchronous all-reduce SGD (with momentum).

```
initialize $\theta_{0,i} \leftarrow \theta_0$
for $t \in \{0, \dots, T\}$ do
    $\Delta\theta_{t,i} \leftarrow -\alpha_t \nabla f_i(\theta_{t,i}; X_{t,i}) + \mu \Delta\theta_{t-1}$
    $\Delta\theta_t \leftarrow$ all-reduce-average$(\Delta\theta_{t,i})$
    $\theta_{t+1,i} \leftarrow \theta_{t,i} + \Delta\theta_t$
end for
```

Despite the optimal logarithmic scaling with the number of nodes p, the cost of communication can become substantial and comparable to that of computation as one tries to scale to thousands of nodes. One solution is to attempt to overlap computation with communication: rather than strictly alternating the computation of local gradients with the communication of the global gradient, it is instead possible to delay the communication by one step, allowing both phases to occur simultaneously. Essentially, the communication of the global gradient is hidden, allowing all nodes to mostly fully utilize the computation resources. This approach is used in [85], which they call double buffering of minibatches. However, the aforementioned solution by [85] is still bulk-synchronous, which introduces a potentially significant synchronization cost at scale.

Large-scale synchronous SGD also implies large minibatch training. One effect of large minibatches is to reduce the noise of stochastic gradient estimates, which can detrimentally affect the convergence of SGD. However, since the publication of this work, new techniques based on learning rate rampup [27] and gradient normalization [100, 101] have made practical SGD with larger minibatch sizes, though concerns of communication and synchronization are still not obviated.

Asynchronous Parameter-Server SGD

A different approach to SGD consists of each node asynchronously performing its own gradient updates and occasionally synchronizing its parameters with a central parameter store. This form of asynchronous SGD was popularized by "Hogwild" SGD [76], which considered solving sparse problems on single machine shared memory systems. "Downpour" SGD [17] then generalized the approach to distributed SGD where nodes communicate their gradients with a central *parameter server*. The main weakness of the asynchronous parameter-server approach to SGD is that the workers communicate all-to-one with a central server, and the communication throughput is limited by the finite link reception bandwidth at the server. One approach for alleviating the communication bottleneck is introducing a delay between rounds of communication, but increasing the delay greatly decreases the rate of convergence [103]. Large scale asynchronous SGD for deep learning was pioneered in the Google DistBelief [17] and Microsoft Adam [13] systems. Large scale parameter server systems in the non-deep learning setting have also been demonstrated in [39] and [57]. We note that a second weakness of asynchronous methods compared to synchronous methods is that asynchronous gradient updates are naturally nondeterministic, hampering debugging and reproducibility, though we do not address this issue.

Algorithm 3 Asynchronous parameter-server ("Downpour") SGD.

```
# client code
initialize $\theta_{0,i} \leftarrow \theta_0$, $g \leftarrow 0$
for $t \in \{0, \dots, T\}$ do
  if $t > 0$ and $t \equiv 0 \mod \tau$ then
    send-client-gradient($g$)
    $\tilde{\theta} \leftarrow$ receive-server-param()
    $\theta_{t+1,i} \leftarrow \tilde{\theta}$
    $g \leftarrow 0$
  end if
  $g \leftarrow g + \nabla f_i(\theta_{t,i}; X_{t,i})$
  $\Delta\theta_{t,i} \leftarrow -\alpha_t \nabla f_i(\theta_{t,i}; X_{t,i})$
  $\theta_{t+1,i} \leftarrow \theta_{t,i} + \Delta\theta_{t,i}$
end for
```

```
# server code
initialize $\tilde{\theta}_0 \leftarrow \theta_0$
for $t \in \{0, \dots, T\}$ do
  $g \leftarrow$ receive-client-gradient()
  $\Delta\tilde{\theta}_t \leftarrow -\alpha_t g + \mu \Delta\tilde{\theta}_{t-1}$
  $\tilde{\theta}_{t+1} \leftarrow \tilde{\theta}_t + \Delta\tilde{\theta}_t$
  send-server-param($\tilde{\theta}_{t+1}$)
end for
```

Elastic Averaging SGD

Elastic averaging SGD [103] is a recently-proposed algorithm belonging to the family of asynchronous parameter-server methods which introduces a modification to the usual stochastic gradient objective to achieve faster convergence. Elastic averaging seeks to maximize the consensus between the center parameter $\tilde{\theta}$ and the local parameters θ_i in addition to the loss:

$$F_{\text{consensus}}(\theta_1, \ldots, \theta_p, \tilde{\theta}) = \sum_{i=1}^{p} \left[f(\theta_i; X_i) + \frac{\rho}{2} \|\theta_i - \tilde{\theta}\|^2 \right]. \tag{3.1}$$

The elastic averaging algorithm is given in Algorithm 4. The consensus objective of elastic averaging is closely related to the augmented Lagrangian of ADMM, and the gradient update derived from the consensus objective was shown to converge significantly faster than vanilla asynchronous parameter-server SGD [103]. However, as elastic averaging is a member of the family of asynchronous parameter-server approaches, it is still subject to a communication bottleneck between the central server and the client workers.

Because recent published results indicate that elastic averaging dominates previous asynchronous parameter-server methods [103], we will only consider elastic averaging from this point on.

Algorithm 4 Elastic averaging SGD. The hyperparameter β is the moving rate.

```
# client code
initialize $\theta_{0,i} \leftarrow \theta_0$
for $t \in \{0, \ldots, T\}$ do
  if $t > 0$ and $t \equiv 0 \mod \tau$ then
    $\tilde{\theta} \leftarrow$ receive-server-param()
    send-param-update($+\beta(\theta_{t,i} - \tilde{\theta})$)
    $\theta_{t,i} \leftarrow \theta_{t,i} - \beta(\theta_{t,i} - \tilde{\theta})$
  end if
  $\Delta\theta_{t,i} \leftarrow -\alpha_t \nabla f_i(\theta_{t,i}; X_{t,i}) + \mu \Delta\theta_{t-1,i}$
  $\theta_{t+1,i} \leftarrow \theta_{t,i} + \Delta\theta_{t,i}$
end for
```

```
# server code
initialize $\tilde{\theta}_0 \leftarrow \theta_0$
for $t \in \{0, \ldots, T\}$ do
  send-server-param($\tilde{\theta}_t$)
  $\Delta\theta \leftarrow$ receive-param-update()
  $\tilde{\theta}_{t+1} \leftarrow \tilde{\theta}_t + \Delta\theta$
end for
```

3.3 Gossiping SGD

Algorithm

In a nutshell, the synchronous all-reduce algorithm consists of two repeating phases: (1) calculation of the local gradients at each node, and (2) exact aggregation of the local gradients via all-reduce. To derive gossiping SGD, we would like to replace the synchronous all-reduce operation with a more asynchronous-friendly communication pattern. The fundamental building block we use is a *gossip aggregation* algorithm [49, 7], which combined with SGD leads to the gossiping SGD algorithm. Asynchronous gossiping SGD was introduced in [75] for the general case of a sparse communication graph between nodes (e.g. wireless sensor networks). The original problem setting of gossiping also typically involved synchronous rounds of communication, whereas we are most interested in asynchronous gossip.

We can also derive the mathematical formulation of the gossiping SGD update by conceptually linking gossiping to elastic averaging. We introduce a distributed version of the global consensus objective, in which the center parameter is replaced with the average of the local parameters:

$$F_{\text{dist-consensus}}(\theta_1, \dots, \theta_p) = \sum_{i=1}^{p} \left[f(\theta_i; X_i) + \frac{\rho}{2} \left\| \theta_i - \frac{1}{p} \sum_{j=1}^{p} \theta_j \right\|^2 \right]. \tag{3.2}$$

If we replace the distributed mean $\frac{1}{p}\sum_{j=1}^{p} \theta_{t,j}$ with the unbiased "one-node estimator" $\theta_{t,j_{t,i}}$, such that $j_{t,i} \sim \text{Uniform}(\{1, \dots, p\})$ and $\mathbb{E}[\theta_{t,j_{t,i}}] = \frac{1}{p}\sum_{j=1}^{p} \theta_{t,j}$, then we derive the gossiping SGD update:

$$\theta'_{t,i} = \theta_{t,i} - \alpha \nabla f(\theta_{t,i}; X_i) \tag{3.3}$$

$$\theta_{t+1,i} = \theta'_{t,i} - \beta(\theta'_{t,i} - \theta'_{j_{t,i},t}) \tag{3.4}$$

$$= (1 - \beta)\theta'_{t,i} + \beta\theta'_{j_{t,i},t}. \tag{3.5}$$

If $j_{t,i}$ is chosen uniformly as in the above "one-node estimator," then the algorithm is equivalent to "pull-gossip," i.e. each node pulls or receives θ_j from one and only one other random node per iteration. On the other hand, if we replace the "one-node estimator" with querying θ_j from multiple nodes, with the constraint that each j is represented only once per iteration, then the algorithm becomes "push-gossip," i.e. each node pushes or sends its own θ_i to one and only one other random node, while receiving from between zero and multiple other nodes. Push-gossiping SGD can be interpreted as an interleaving of a gradient step and a simplified push-sum gossip step [49]. Algorithms 5 and 6 describe pull-gossiping and push-gossiping SGD respectively.

Algorithm 5 Pull-gossiping SGD.

```
initialize $\theta_{0,i} \leftarrow \theta_0$
for $t \in \{0, \dots, T\}$ do
    if $t > 0$ and $t \equiv 0 \mod \tau$ then
        set $x_i \leftarrow \theta_{t,i}$
        choose a target $j$
        $\theta_{t,i} \leftarrow$ average of $x_i, x_j$
    end if
    $\Delta\theta_{t,i} \leftarrow -\alpha_t \nabla f_i(\theta_{t,i}; X_{t,i}) + \mu\Delta\theta_{t-1,i}$
    $\theta_{t+1,i} \leftarrow \theta_{t,i} + \Delta\theta_{t,i}$
end for
```

Algorithm 6 Push-gossiping SGD.

```
initialize $\theta_{0,i} \leftarrow \theta_0$
for $t \in \{0, \dots, T\}$ do
    if $t > 0$ and $t \equiv 0 \mod \tau$ then
        set $x_i \leftarrow \theta_{t,i}$
        choose a target $j$
        send $x_i$ to $i$ (ourselves) and to $j$
        $\theta_{t,i} \leftarrow$ average of received $x$'s
    end if
    $\Delta\theta_{t,i} \leftarrow -\alpha_t \nabla f_i(\theta_{t,i}; X_{t,i}) + \mu\Delta\theta_{t-1,i}$
    $\theta_{t+1,i} \leftarrow \theta_{t,i} + \Delta\theta_{t,i}$
end for
```

Analysis

Our analysis of gossiping SGD is based on the analyses in [69, 7, 75, 93, 103]. We assume that all processors are able to communicate with all other processors at each step. The main convergence result is the following:

Theorem 1. *Let f be a m-strongly convex function with L-Lipschitz gradients. Assume that we can sample gradients $g = \nabla f(\theta; X_i) + \xi_i$ with additive noise with zero mean $\mathbb{E}[\xi_i] = 0$ and bounded variance $\mathbb{E}[\xi_i^T \xi_i] \leq \sigma^2$. Then, running the asynchronous pull-gossip algorithm, with constant step size $0 < \alpha \leq \frac{2}{m+L}$, the expected sum of squares convergence of the local parameters to the optimal θ_* is bounded by*

$$\mathbb{E}[\|\theta_t - \theta_* \mathbf{1}\|^2] \leq \left(1 - \frac{2\alpha}{p} \frac{mL}{m+L}\right)^t \|\theta_0 - \theta_* \mathbf{1}\|^2 + p\alpha\sigma^2 \frac{m+L}{2mL} \tag{3.6}$$

Furthermore, with the additional assumption that the gradients are uniformly bounded as $\sup |\nabla f(\theta)| \leq C$, the expected sum of squares convergence of the local parameters to the mean $\bar{\theta}_t$ is bounded by

$$\mathbb{E}[\|\theta_t - \bar{\theta}_t \mathbf{1}\|^2] \leq \left(\lambda \left(1 - \frac{\alpha\, m}{p}\right)\right)^t \|\theta_0 - \bar{\theta}_0 \mathbf{1}\|^2 + \frac{\lambda \alpha^2 (C^2 + \sigma^2)}{1 - \lambda \left(1 - \frac{\alpha\, m}{p}\right)} \tag{3.7}$$

$$\textit{where } \lambda = 1 - \frac{2\beta(1-\beta)}{p} - \frac{2\beta^2}{p} \tag{3.8}$$

For the proofs, please see Section 3.5.

3.4 Experiments

Implementation

We implement the communication systems of gossiping SGD and other algorithms using Message Passing Interface (MPI) [94]. Because we wanted to run our code in cluster computing environments with Infiniband or more specialized interconnects, then targeting MPI was the easiest solution. We targeted our code to run on GPUs, using the Nvidia CUDA 7.0 driver and using the cuBLAS and cuDNNv4 [12] libraries for the core computational kernels.

For our experiments up to $p = 16$ nodes, we use a local cluster of 16 machines, each one consisting of an Nvidia Kepler K80 dual GPU, an 8-core Intel Haswell E5-1680v2 CPU, and a Mellanox ConnectX-3 FDR 4× Infiniband (56 Gb/s) NIC. We utilize only one GPU per K80.

For our larger scale experiments up to $p = 128$ nodes, we used a GPU supercomputer with over 10,000 total nodes [70]. Nodes consist of an Nvidia Kepler K20X GPU and an 8-core AMD Bulldozer Opteron 6274 CPU, and are connected by a Cray Gemini interconnect in a 3D torus configuration.

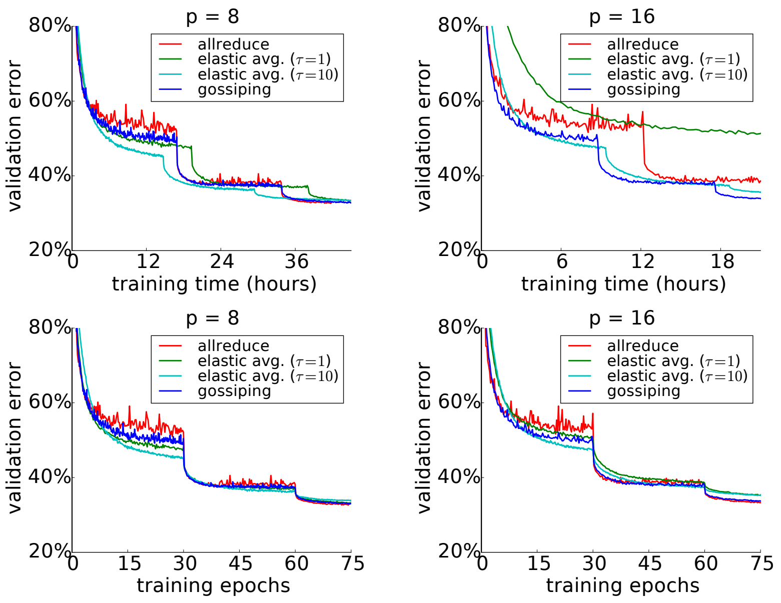

Figure 3.1: Center-crop validation loss and top-1 error on ImageNet over training wall-clock time and epochs. Shown are: (left) $p = 8$ nodes with *per-node* minibatch size $b = 32$, and (right) $p = 16$ nodes with *per-node* minibatch size $b = 16$.

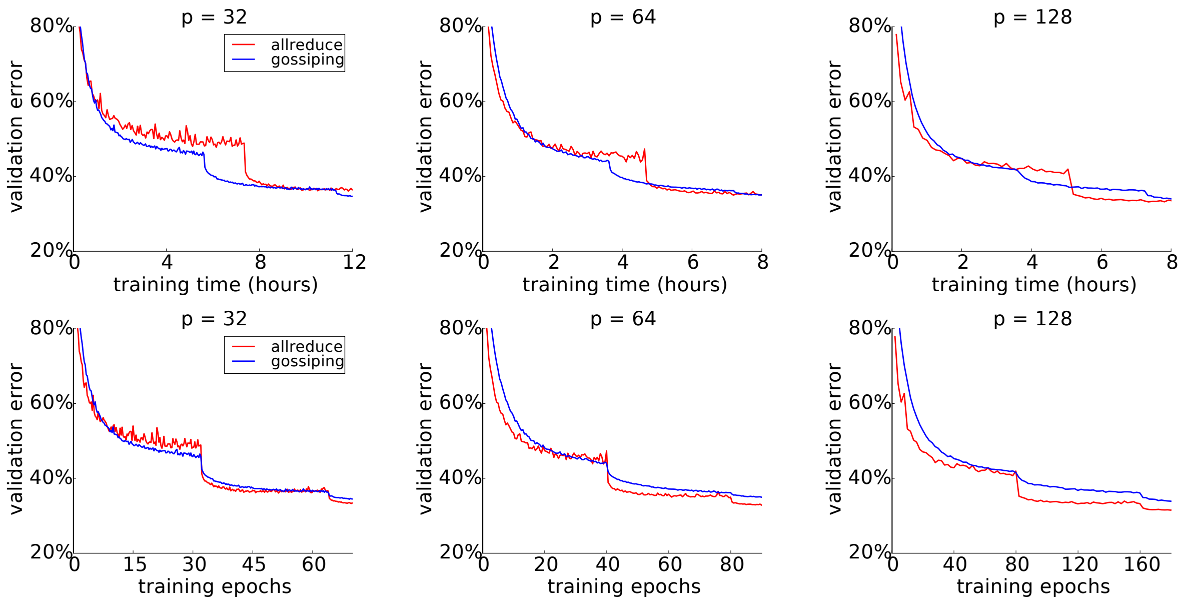

Figure 3.2: Center-crop validation loss and top-1 error on ImageNet over training wall-clock time and epochs, with different numbers of nodes p and *per-node* minibatch size $b = 16$. Shown are: (left) $p = 32$ nodes, (middle) $p = 64$ nodes, and (right) $p = 128$ nodes.

Methodology

We chose ResNets [37] for our neural network architecture; specifically, we trained ResNet-18, which is small enough to train rapidly for experimentation, but also possesses features relevant to modern networks, including depth, residual layers, and batch normalization [41]. We ran on the image classification problem of ImageNet consisting of 1.28 million training images and 50,000 validation images divided into 1000 classes [79]. Our data augmentation is as follows: we performed multi-scale training by scaling the shortest dimension of images to between 256 and 480 pixels [87], we took random 224×224 crops and horizontal flips, and we added pixelwise color noise [53]. We evaluate validation loss and top-1 error on center crops of the validation set images with the shortest dimension scaled to 256 pixels.

Unless otherwise noted, we initialized the step size to $\alpha = 0.1$, then we annealed it twice by a factor of 0.1. For our experiments with aggregate minibatch size $m = pb = 256$, we annealed at exactly 150k and 300k iterations into training (See Section 3.4 for details on batch size settings). For our experiments with larger aggregate minibatch sizes, we decreased the number of iterations at which the step size was annealed. We used Nesterov [68] momentum of $\mu = 0.9$ and weight decay of $\lambda = 10^{-4}$. For elastic averaging, we set $\beta = 0.8/p$. For all-reduce and gossiping, we used a communication interval of $\tau = 1$, i.e. communication occurred every iteration. For gossiping, we used both $\tau = 1$ and $\tau = 10$ (the latter is recommended in [103]).

Results

Our first set of experiments compare all-reduce, elastic averaging, and push-gossiping at $p = 8$ and $p = 16$ with an aggregate minibatch size $m = pb = 256$. The results are in Figure 3.1. For $p = 8$, elastic averaging with a communication delay $\tau = 10$ completes each iteration more quickly the other methods. Interestingly, all-reduce has practically no synchronization overhead on the system at $p = 8$ and is as fast as gossiping. All methods converge to roughly the same minimum loss value. For $p = 16$, gossiping converges faster than elastic averaging with $\tau = 10$, and both come ahead of all-reduce. Additionally, elastic averaging with both $\tau = 1$ and $\tau = 10$ has trouble converging to the same validation loss as the other methods once the step size has been annealed to a small value ($\alpha = 0.001$ in this case).

We also perform larger scale experiments at $p = 32$ nodes, $p = 64$ nodes, and $p = 128$ nodes in the GPU supercomputing environment. In this environment, elastic averaging did not perform well so we do not show those results here; pull-gossiping also performed better than push-gossiping, so we only show results for pull-gossiping. The results are in Figure 3.2. At this scale, we begin to see the scaling advantage of synchronous all-reduce SGD. One iteration of gossiping SGD is still faster than one iteration of all-reduce SGD, and gossiping works quickly at the initial step size. However, after the step size has been annealed, each iteration of gossiping SGD leads to less progress towards convergence than in synchronous SGD.

We note that the training time of SGD can be thought of as the product:

$$\text{(wall-clock time per iteration)} \times \text{(number of iterations)}. \tag{3.9}$$

One observation we made consistent with [10] was the following: letting synchronous all-reduce SGD run for many epochs, it will typically converge to a lower optimal validation loss (or higher validation accuracy) than either elastic averaging or gossiping SGD. We found that letting all-reduce SGD run for over 1 million iterations with a minibatch size of 256 led to a peak top-1 validation accuracy of 68.7% (31.3% top-1 error). However, elastic averaging often had trouble breaking 67% accuracy using $p = 16$ nodes, as did gossiping using more than $p = 32$ nodes. In other words, at larger scales the asynchronous methods require more iterations to convergence despite lower wall-clock time per iteration.

3.5 Proofs

The analysis below loosely follow the arguments presented in, and use a combination of techniques appearing from [17,16,10,18,9].

For ease of exposition and notation, we focus our attention on the case of univariate strongly convex function f with Lipschitz gradients. (Since the sum of squares errors are additive in vector components, the arguments below generalize to the case of multivariate functions.) We assume that the gradients of the function f can be sampled by all processors independently up to additive noise with zero mean and bounded variance. (Gaussian noise satisfy these assumptions, for example.) We also assume a fully connected network where all processors are able to communicate with all other processors. Additional assumptions are introduced below as needed.

We denote by $\theta_t = \left[\theta_{t,1}, \cdots, \theta_{t,p}\right]^T$ the vector containing all local parameter values at time step t, we denote by θ_* the optimal value of the objective function f, and we denote by $\bar{\theta}_t \equiv \frac{1}{p}\sum_{i=1}^{p} \theta_{t,i}$ the spatial average of the local parameters taken at time step t.

We derive bounds for the following two quantities:

- $\mathbb{E}[\|\theta_t - \theta_*\mathbf{1}\|^2]$, the squared sum of the local parameters' deviation from the optimum θ_*
- $\mathbb{E}[\|\theta_t - \bar{\theta}_t\mathbf{1}\|^2]$, the squared sum of the local parameters' deviation from the mean $\bar{\theta}_t = \frac{1}{p}\sum_{i=1}^{p} \theta_{t,i}$

where the expectation is taken with respect to both the "pull" parameter choice and the gradient noise term. In the literature [18], the latter is usually referred to as "agent agreement" or "agent consensus".

Synchronous pull-gossip algorithm

We begin by analyzing the synchronous version of the pull-gossip algorithm described in Algorithm 3. For each processor i, let j_i denote the processor, chosen uniformly randomly from $\{1, \cdots, p\}$ from which processor i "pulls" parameter values. The update for each $\theta_{t,i}$ is given by

$$\theta_{t+1,i} = \frac{1}{2}\left(\theta_{t,i} + \theta_{t,j_i}\right) - \alpha_t \left(\nabla f\left(\frac{1}{2}\left(\theta_{t,i} + \theta_{t,j_i}\right); X_{t,i}\right) + \xi_{t,i}\right) \tag{3.10}$$

We prove the following rate of convergence for the synchronous pull-gossip algorithm.

Theorem 2. *Let f be a m-strongly convex function with L-Lipschitz gradients. Assume that we can sample gradients $g = \nabla f(\theta; X_i) + \xi_i$ with additive noise with zero mean $\mathbb{E}[\xi_i] = 0$ and bounded variance $\mathbb{E}[\xi_i^T \xi_i] \leq \sigma^2$. Then, running the synchronous pull-gossip algorithm as outlined above with step size $0 < \alpha \leq \frac{2}{m+L}$, the expected sum of squares convergence of the local parameters to the optimal θ_* is bounded by*

$$\mathbb{E}[\|\theta_t - \theta_* \mathbf{1}\|^2] \leq \left(1 - 2\alpha \frac{mL}{m+L}\right)^t \|\theta_0 - \theta_* \mathbf{1}\|^2 + p\alpha\sigma^2 \frac{m+L}{2mL} \tag{3.11}$$

Remark 1. *Note that this bound is characteristic of SGD bounds, where the iterates converge to within some ball around the optimal solution. It can be shown by induction that a decreasing step size schedule of $\alpha_t \sim \mathcal{O}(1/pt)$ can be used to achieve a convergence rate of $\mathcal{O}(1/t)$.*

Proof. For notational simplicity, we denote $\theta_{t,i,j_i} \equiv \frac{1}{2}\left(\theta_{t,i} + \theta_{t,j_i}\right)$, and drop the $X_{t,i}$ in the gradient term. We tackle the first quantity by conditioning on the previous parameter values and expanding as

$$\mathbb{E}[\|\theta_{t+1} - \theta_* \mathbf{1}\|^2 \mid \theta_t] = \mathbb{E}\left[\sum_{i=1}^{p} \left(\theta_{t,i,j_i} - \theta_*\right)^T \left(\theta_{t,i,j_i} - \theta_*\right) \,\middle|\, \theta_t\right] \tag{3.12}$$

$$- 2\alpha_t \mathbb{E}\left[\sum_{i=1}^{p} \nabla f(\theta_{t,i,j_i})^T (\theta_{t,i,j_i} - \theta_*) \,\middle|\, \theta_t\right] \tag{3.13}$$

$$- 2\alpha_t \mathbb{E}\left[\sum_{i=1}^{p} \xi_{t,i}^T \left(\theta_{t,i,j_i} - \theta_*\right) \,\middle|\, \theta_t\right] \tag{3.14}$$

$$+ \alpha_t^2 \left[\sum_{i=1}^{p} \left(\nabla f(\theta_{t,i,j_i}) + \xi_{t,i}\right)^T \left(\nabla f(\theta_{t,i,j_i}) + \xi_{t,i}\right) \,\middle|\, \theta_t\right] \tag{3.15}$$

Recalling that strongly convex functions satisfy [17], $\forall x, z$,

$$\begin{aligned}
&(\nabla f(x) - \nabla f(z))^T (x - z) && (3.16)\\
&\geq \frac{mL}{m+L}(x-z)^T(x-z) + \frac{1}{m+L}(\nabla f(x) - \nabla f(z))^T (\nabla f(x) - \nabla f(z))
\end{aligned}$$

we can use this inequality, with $x = \theta_{t,i,j_i}$ and $z = \theta_*$ to bound the term in (3.13):

$$\begin{aligned}
&-2\alpha_t \mathbb{E}\left[\sum_{i=1}^{p} \nabla f(\theta_{t,i,j_i})^T (\theta_{t,i,j_i} - \theta_*) \,\middle|\, \theta_t\right] && (3.17)\\
&\leq -\frac{2\alpha_t mL}{m+L}\mathbb{E}\left[\sum_{i=1}^{p}(\theta_{t,i,j_i} - \theta_*)^T(\theta_{t,i,j_i} - \theta_*) \,\middle|\, \theta_t\right] - \frac{2\alpha_t}{m+L}\mathbb{E}\left[\sum_{i=1}^{p} \nabla f(\theta_{t,i,j_i})^T \nabla f(\theta_{t,i,j_i}) \,\middle|\, \theta_t\right]
\end{aligned}$$

Using (3.18), and regrouping terms in (3.12)-(3.15), we obtain

$$\begin{aligned}
\mathbb{E}[\|\theta_{t+1} - \theta_* \mathbf{1}\|^2 \,|\, \theta_t] \leq &\left(1 - 2\alpha_t \frac{mL}{m+L}\right)\mathbb{E}\left[\sum_{i=1}^{p}(\theta_{t,i,j_i} - \theta_*)^T(\theta_{t,i,j_i} - \theta_*) \,\middle|\, \theta_t\right] && (3.18)\\
&+\left(\alpha_t^2 - 2\alpha_t \frac{1}{m+L}\right)\mathbb{E}\left[\sum_{i=1}^{p} \nabla f(\theta_{t,i,j_i})^T \nabla f(\theta_{t,i,j_i}) \,\middle|\, \theta_t\right] && (3.19)\\
&+\alpha_t^2 \mathbb{E}\left[\sum_{i=1}^{p} \xi_{t,i}^T \xi_{t,i} \,\middle|\, \theta_t\right] && (3.20)
\end{aligned}$$

In the above expression, we have dropped the terms linear in $\xi_{t,i}$, using the assumption that these noise terms vanish in expectation. In addition, if the step size parameter α is chosen sufficiently small, $0 < \alpha_t \leq \frac{2}{m+L}$, then the second term in (3.19) can also be dropped. The expression we must contend with is

$$\mathbb{E}[\|\theta_{t+1} - \theta_* \mathbf{1}\|^2 \,|\, \theta_t] \leq \left(1 - 2\alpha_t \frac{mL}{m+L}\right)\mathbb{E}\left[\sum_{i=1}^{p}(\theta_{t,i,j_i} - \theta_*)^T(\theta_{t,i,j_i} - \theta_*) \,\middle|\, \theta_t\right] + \alpha_t^2 p\sigma^2 \tag{3.21}$$

Using the definition of θ_{t,i,j_i}, we can verify that the following matrix relation holds.

$$\begin{bmatrix} \theta_{t,1,j_1} - \theta_* \\ \vdots \\ \theta_{t,i,j_i} - \theta_* \\ \vdots \\ \theta_{t,p,j_p} - \theta_* \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & & & \frac{1}{2} & \\ & \ddots & & & \\ & \frac{1}{2} & \frac{1}{2} & & \\ & & & \ddots & \\ \frac{1}{2} & & & & \frac{1}{2} \end{bmatrix} \begin{bmatrix} \theta_{t,1} - \theta_* \\ \vdots \\ \theta_{t,i} - \theta_* \\ \vdots \\ \theta_{t,p} - \theta_* \end{bmatrix} \equiv M_I(\theta_t - \theta_* \mathbf{1}) \tag{3.22}$$

where the random matrix M_I depends on the random index set $I = \{j_i\}_{i=1}^p$ of uniformly randomly drawn indices. M_I has two entries in each row, one on the diagonal, and one appearing in the j_ith column, and is a right stochastic matrix but need not be doubly stochastic.

We can express this matrix as

$$M_I = \frac{1}{2}\sum_{i=1}^{p} e_i(e_i + e_{j_i})^T$$

and compute its second moment

$$\begin{aligned}
\mathbb{E}[M_I^T M_I] &= \frac{1}{4}\,\mathbb{E}\left[\left(I + \sum_{i=1}^{p} e_i e_{j_i}^T\right)^T \left(I + \sum_{i=1}^{p} e_i e_{j_i}^T\right)\right] && (3.23)\\
&= \frac{1}{2}\left(I + \frac{1}{p}\mathbf{1}\mathbf{1}^T\right) && (3.24)\\
&= Q\begin{bmatrix} 1 & & & \\ & \frac{1}{2} & & \\ & & \ddots & \\ & & & \frac{1}{2}\end{bmatrix} Q^T && (3.25)
\end{aligned}$$

where in the last line, the orthogonal diagonalization reveals that the eigenvalues of this matrix are bounded by $\frac{1}{2} \le \lambda_i \le 1$.

Using (3.25), we can further simply (3.21) to

$$\begin{aligned}
\mathbb{E}[\|\theta_{t+1} - \theta_*\mathbf{1}\|^2 \,|\, \theta_t] &\le \left(1 - 2\alpha_t \frac{mL}{m+L}\right)(\theta_t - \theta_*\mathbf{1})^T \mathbb{E}\left[M_I^T M_I\right](\theta_t - \theta_*\mathbf{1}) + \alpha_t^2 p\sigma^2 && (3.26)\\
&\le \left(1 - 2\alpha_t \frac{mL}{m+L}\right)\|\theta_t - \theta_*\mathbf{1}\|^2 + \alpha_t^2 p\sigma^2 && (3.27)
\end{aligned}$$

Assuming a constant step size $\alpha_t \equiv \alpha$, the above recursion can be unrolled to derive the bound

$$\mathbb{E}[\|\theta_t - \theta_*\mathbf{1}\|^2] \le \left(1 - 2\alpha \frac{mL}{m+L}\right)^t \|\theta_0 - \theta_*\mathbf{1}\|^2 + p\alpha\sigma^2 \frac{m+L}{2mL} \qquad (3.28)$$

We note that the above bound is characteristic of SGD bounds, where the iterates converge to a ball around the optimal solution, whose radius now depends on the number of processors p, in addition to the step size α and the variance of the gradient noise σ^2.

□

Asynchronous pull-gossip algorithm

We provide similar analysis for the asynchronous version of the pull-gossip algorithm. As is frequently done in the literature, we model the time steps as the ticking of local clocks governed by Poisson processes. More precisely, we assume that each processor has a clock which ticks with a rate 1 Poisson process. A master clock which ticks whenever a local processor clock ticks is then governed by a rate p Poisson process, and a time step in the algorithm is defined as whenever the master clock ticks. Since each master clock tick corresponds to the tick of some local clock on processor i, this in turn marks the time step at which processor i "pulls" the parameter values from the uniformly randomly chosen processor j_i. Modeling the time steps by Poisson processes provide nice theoretical properties, i.e. the inter-tick time intervals are i.i.d. exponential variables of rate p, and the local clock i which causes each master clock tick is i.i.d. drawn from $\{1, \cdots, p\}$, to name a few. For an in depth analysis of this model, and results that relate the master clock ticks to absolute time, please see [16].

The main variant implemented is the pull-gossip with fresh parameters in subsection 6.2.2. The update at each time step is given by

$$\begin{cases} \theta_{t+1,i} = (1-\beta)\left(\theta_{t,i} - \alpha_t\left(\nabla f(\theta_{t,i}) + \xi_{t,i}\right)\right) + \beta\theta_{t,j_i} & \\ \theta_{t+1,k} = \theta_{t,k} & \text{, for } \ k \neq i \end{cases}$$

where $\xi_{t,i}$ is the gradient noise

Note in the implementation we use $\beta = 1/2$, however in the analysis we retain the β parameter for generality.

We have the following convergence result.

Theorem 3. *Let f be a m-strongly convex function with L-Lipschitz gradients. Assume that we can sample gradients $g = \nabla f(\theta; X_i) + \xi_i$ with additive noise with zero mean $\mathbb{E}[\xi_i] = 0$ and bounded variance $\mathbb{E}[\xi_i^T \xi_i] \leq \sigma^2$. Then, running the asynchronous pull-gossip algorithm with the time model as described, with constant step size $0 < \alpha \leq \frac{2}{m+L}$, the expected sum of squares convergence of the local parameters to the optimal θ_* is bounded by*

$$\mathbb{E}[\|\theta_t - \theta_* \mathbf{1}\|^2] \leq \left(1 - \frac{2\alpha}{p}\frac{mL}{m+L}\right)^t \|\theta_0 - \theta_*\mathbf{1}\|^2 + p\alpha\sigma^2\frac{m+L}{2mL} \tag{3.29}$$

Furthermore, with the additional assumption that the gradients are uniformly bounded as $\sup|\nabla f(\theta)| \leq C$*, the expected sum of squares convergence of the local parameters to the mean*

$\bar{\theta}_t$ *is bounded by*

$$\mathbb{E}[\|\theta_t - \bar{\theta}_t \mathbf{1}\|^2] \leq \left(\lambda\left(1 - \frac{\alpha\, m}{p}\right)\right)^t \|\theta_0 - \bar{\theta}_0 \mathbf{1}\|^2 + \frac{\lambda \alpha^2 (C^2 + \sigma^2)}{1 - \lambda\left(1 - \frac{\alpha\, m}{p}\right)} \tag{3.30}$$

$$\textit{where } \lambda = 1 - \frac{2\beta(1-\beta)}{p} - \frac{2\beta^2}{p} \tag{3.31}$$

Remark 2. *Note again that this bound is characteristic of SGD bounds, with the additional dependence on p, the number of processors.*

Proof. For simplicity of notation, we denote $g_{t,i} \equiv \nabla f(\theta_{t,i}) + \xi_{t,i}$. We can write the asynchronous iteration step in matrix form as

$$\begin{bmatrix} \theta_{t+1,1} - \theta_* \\ \vdots \\ \theta_{t+1,i} - \theta_* \\ \vdots \\ \theta_{t+1,j_i} - \theta_* \\ \vdots \\ \theta_{t+1,p} - \theta_* \end{bmatrix} = \begin{bmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1-\beta & \beta & & \\ & & & \ddots & & \\ & & & 1 & & \\ & & & & \ddots & \\ & & & & & 1 \end{bmatrix} \left(\begin{bmatrix} \theta_{t,1} - \theta_* \\ \vdots \\ \theta_{t,i} - \theta_* \\ \vdots \\ \theta_{t,j_i} - \theta_* \\ \vdots \\ \theta_{t,p} - \theta_* \end{bmatrix} - \alpha_t \begin{bmatrix} 0 \\ \vdots \\ g_{t,i} \\ \vdots \\ 0 \\ \vdots \\ 0 \end{bmatrix} \right) \tag{3.32}$$

$$\equiv D_{i,j_i}\left((\theta_t - \theta_*) - \alpha_t g_t\right) \tag{3.33}$$

The random matrix D_{i,j_i} depends on the indices i and j_i, both of which are uniformly randomly drawn from $\{1, \cdots, p\}$. For notational convenience we will drop the subscripts, but we keep in mind that the expectation below is taken with respect to the randomly chosen indices. We can express the matrix as

$$D = I + \beta e_i (ej_i - e_i)^T \tag{3.34}$$

and compute its second moment as

$$\mathbb{E}[D^T D] = \left(1 - \frac{2\beta(1-\beta)}{p}\right)\mathbf{I} + \frac{2\beta(1-\beta)}{p^2}\mathbf{1}\mathbf{1}^T \tag{3.35}$$

$$= Q \begin{bmatrix} 1 & & & \\ & 1 - \frac{2\beta(1-\beta)}{p} & & \\ & & \ddots & \\ & & & 1 - \frac{2\beta(1-\beta)}{p} \end{bmatrix} Q^T \tag{3.36}$$

The orthogonal diagonalization reveals that the eigenvalues of this matrix are bounded by $1 - \frac{2\beta(1-\beta)}{p} \leq \lambda_i \leq 1$.

Using (3.36), we can expand and bound the expected sum of squares deviation by

$$\begin{aligned}
&\mathbb{E}[\|\theta_{t+1} - \theta_*\mathbf{1}\|^2|\theta_t] \\
&= \mathbb{E}\left[((\theta_t - \theta_*\mathbf{1}) - \alpha_t g_t)^T D^T D((\theta_t - \theta_*\mathbf{1}) - \alpha_t g_t)\right] && (3.37)\\
&\leq \mathbb{E}\left[\|(\theta_t - \theta_*\mathbf{1}) - \alpha_t g_t\|^2\right] && (3.38)\\
&= \|\theta_t - \theta_*\mathbf{1}\|^2 - \frac{2\alpha_t}{p}\sum_{i=1}^p \nabla f(\theta_{t,i})^T(\theta_{t,i} - \theta_*) + \frac{\alpha_t^2}{p}\sum_{i=1}^p \nabla f(\theta_{t,i})^T \nabla f(\theta_{t,i}) + \frac{\alpha_t^2}{p}\sum_{i=1}^p \xi_{t,i}^T \xi_{t,i} && (3.39)
\end{aligned}$$

where in the last line, we have dropped terms that are linear in $\xi_{t,i}$ by using the zero mean assumption. Making use of the strong convexity inequality (3.17), we can bound the second term in the above sum by

$$-\nabla f(\theta_{t,i})^T(\theta_{t,i} - \theta_*) \leq -\frac{mL}{m+L}(\theta_{t,i} - \theta_*)^T(\theta_{t,i} - \theta_*) - \frac{1}{m+L}\nabla f(\theta_{t,i})^T \nabla f(\theta_{t,i})$$

and rearrange the terms in (3.39) to derive

$$\begin{aligned}
\mathbb{E}[\|\theta_{t+1} - \theta_*\mathbf{1}\|^2|\theta_t] \leq &\left(1 - \frac{2\alpha_t}{p}\frac{mL}{m+L}\right)\|\theta_t - \theta_*\mathbf{1}\|^2 && (3.40)\\
&+ \left(\frac{\alpha_t^2}{p} - \frac{2\alpha_t}{p}\frac{1}{m+L}\right)\sum_{i=1}^p \nabla f(\theta_{t,i})^T \nabla f(\theta_{t,i}) && (3.41)\\
&+ \alpha_t^2\sigma^2 && (3.42)
\end{aligned}$$

The term in (3.41) can be dropped if α_t is chosen sufficiently small, with the now familiar requirement that $0 < \alpha_t < \frac{2}{m+L}$. Assuming this is the case, we have

$$\mathbb{E}[\|\theta_{t+1} - \theta_*\mathbf{1}\|^2|\theta_t] \leq \left(1 - \frac{2\alpha_t}{p}\frac{mL}{m+L}\right)\|\theta_t - \theta_*\mathbf{1}\|^2 + \alpha_t^2\sigma^2 \qquad (3.43)$$

Assuming a constant step size $\alpha_t \equiv \alpha$, we can use the law of iterated expectations to unroll the recursion, giving

$$\mathbb{E}[\|\theta_t - \theta_*\mathbf{1}\|^2] \leq \left(1 - \frac{2\alpha}{p}\frac{mL}{m+L}\right)^t \|\theta_0 - \theta_*\mathbf{1}\|^2 + p\alpha\sigma^2\frac{m+L}{2mL} \qquad (3.44)$$

Note this is the same convergence rate guarantee as stochastic gradient descent with an extra factor of p, albeit the coordination amongst processors required to adjust the step size is unrealistic in practice.

Next, we prove the bound on the processors' local parameters' convergence to each other / to the mean, $\mathbb{E}[\|\theta_t - \bar{\theta}_t \mathbf{1}\|^2]$.

First, we write the spatial parameter average in vector form as

$$\bar{\theta}_t = \frac{1}{p}\mathbf{1}^T \theta_t \tag{3.45}$$

With D as previously defined in (3.33), we can write

$$\bar{\theta}_{t+1}\mathbf{1} = \frac{1}{p}\mathbf{1}^T \theta_{t+1}\mathbf{1} = \frac{1}{p}\mathbf{1}^T D(\theta_t - \alpha g_t)\mathbf{1} \tag{3.46}$$

and so the term we wish to bound can be expanded as

$$\|\theta_{t+1} - \bar{\theta}_{t+1}\mathbf{1}\|^2 = (\theta_t - \alpha_t g_t)^T \left(D^T D - \frac{1}{p} D^T \mathbf{1}\mathbf{1}^T D \right) (\theta_t - \alpha_t g_t) \tag{3.47}$$

In addition to the diagonalization we previous calculated in (3.36) (reproduced below for convenience), we calculate some other useful diagonalizations.

$$\mathbb{E}[D^T D] = \left(1 - \frac{2\beta(1-\beta)}{p}\right)\mathbf{I} + \frac{2\beta(1-\beta)}{p^2}\mathbf{1}\mathbf{1}^T \tag{3.48}$$

$$= Q \begin{bmatrix} 1 & & & \\ & 1 - \frac{2\beta(1-\beta)}{p} & & \\ & & \ddots & \\ & & & 1 - \frac{2\beta(1-\beta)}{p} \end{bmatrix} Q^T \tag{3.49}$$

$$\mathbb{E}[D^T \mathbf{1}\mathbf{1}^T D] = \frac{2\beta^2}{p}\mathbf{I} + \left(1 - \frac{2\beta^2}{p^2}\right)\mathbf{1}\mathbf{1}^T \tag{3.50}$$

$$= Q \begin{bmatrix} p & & & \\ & \frac{2\beta^2}{p} & & \\ & & \ddots & \\ & & & \frac{2\beta^2}{p} \end{bmatrix} Q^T \tag{3.51}$$

$$\mathbb{E}\left[D^T D - \frac{1}{p}D^T \mathbf{1}\mathbf{1}^T D\right] = \left(1 - \frac{2\beta(1-\beta)}{p} - \frac{2\beta^2}{p^2}\right)\mathbf{I} + \left(\frac{2\beta(1-\beta)}{p^2} - \frac{1}{p}\left(1 - \frac{2\beta^2}{p^2}\right)\right)\mathbf{1}\mathbf{1}^T \tag{3.52}$$

$$= Q \begin{bmatrix} 0 & & & \\ & 1 - \frac{2\beta(1-\beta)}{p} - \frac{2\beta^2}{p^2} & & \\ & & \ddots & \\ & & & 1 - \frac{2\beta(1-\beta)}{p} - \frac{2\beta^2}{p^2} \end{bmatrix} Q^T \tag{3.53}$$

We note that all three matrices are diagonalized by the same orthogonal matrix Q. Furthermore, the first eigenvector, corresponding to the eigenvalues 1, p, 0 respectively, is $q_1 = \frac{1}{\sqrt{p}}\mathbf{1}$.

Continuing from (3.47), we take the expectation and use (3.53) to further bound the expression. In the computation below we use $\lambda \equiv 1 - \frac{2\beta(1-\beta)}{p} - \frac{2\beta^2}{p}$ as a notational shorthand.

$$\begin{aligned}
&\mathbb{E}[\|\theta_{t+1} - \bar{\theta}_{t+1}\mathbf{1}\|^2|\theta_t] \\
&= \mathbb{E}\left[(\theta_t - \alpha_t g_t)^T \left(D^T D - \frac{1}{p}D^T\mathbf{1}\mathbf{1}^T D\right)(\theta_t - \alpha_t g_t)\right] && (3.54)\\
&\le \mathbb{E}\left[\lambda\|(\theta_t - \bar{\theta}_t\mathbf{1}) - \alpha_t g_t\|^2\right] && (3.55)\\
&= \lambda\left(\|\theta_t - \bar{\theta}_t\mathbf{1}\|^2 - \frac{2\alpha_t}{p}\sum_{i=1}^p \nabla f(\theta_{t,i})^T(\theta_{t,i} - \bar{\theta}_t) + \frac{\alpha_t^2}{p}\sum_{i=1}^p \nabla f(\theta_{t,i})^T\nabla f(\theta_{t,i}) + \frac{\alpha_t^2}{p}\sum_{i=1}^p \xi_{t,i}^T\xi_{t,i}\right) && (3.56)\\
&\le \lambda\left(\|\theta_t - \bar{\theta}_t\mathbf{1}\|^2 + \frac{2\alpha_t}{p}\sum_{i=1}^p \left(f(\bar{\theta}_t) - f(\theta_{t,i})\right) - \frac{\alpha_t m}{p}\sum_{i=1}^p(\theta_{i,t} - \bar{\theta}_t)^2 + \frac{\alpha_t^2}{p}\sum_{i=1}^p(C^2 + \sigma^2)\right) && (3.57)\\
&\le \lambda\left(1 - \frac{\alpha_t m}{p}\right)\|\theta_t - \bar{\theta}_t\mathbf{1}\|^2 + \lambda\alpha_t^2(C^2 + \sigma^2) && (3.58)
\end{aligned}$$

In (3.56) and (3.57), we have used the definition of strong convexity and convexity respectively.

Finally, taking constant step size $\alpha_t \equiv \alpha$, and using the law of iterated expectations to unroll the recursion, we have

$$\mathbb{E}[\|\theta_t - \bar{\theta}_t\mathbf{1}\|^2] \le \left(\lambda\left(1 - \frac{\alpha m}{p}\right)\right)^t \|\theta_0 - \bar{\theta}_0\mathbf{1}\|^2 + \frac{\lambda\alpha^2(C^2 + \sigma^2)}{1 - \lambda\left(1 - \frac{\alpha m}{p}\right)} \tag{3.59}$$

When the step size α is small and the number of processors p is large, the quantity $\lambda\left(1 - \frac{\alpha m}{p}\right) = \left(1 - \frac{2\beta(1-\beta)}{p} - \frac{2\beta^2}{p}\right)\left(1 - \frac{\alpha m}{p}\right)$ is well approximated by $1 - \frac{2\beta(1-\beta)}{p} - \frac{2\beta^2}{p} - \frac{\alpha m}{p}$, which makes clear the dependence of the rate on the parameter p. □

3.6 Implementation Details

Gossip variants

Gossip with stale parameters (gradient step and gossip at the same time)

Consider a one-step distributed consensus gradient update:

$$\theta_{i,t+1} = \theta_{i,t} - \alpha \nabla f(\theta_{i,t}; X_i) - \beta \left(\theta_{i,t} - \frac{1}{p} \sum_{j=1}^{p} \theta_{j,t} \right). \tag{3.60}$$

If we replace the distributed mean $\frac{1}{p}\sum_{j=1}^{p} \theta_{j,t}$ with an unbiased one-sample estimator $\theta_{j_{i,t},t}$, such that $j_{i,t} \sim \text{Uniform}(\{1,\ldots,p\})$ and $\mathbb{E}[\theta_{j_{i,t},t}] = \frac{1}{p}\sum_{j=1}^{p} \theta_{j,t}$, then we derive the gossiping SGD update:

$$\begin{aligned} \theta_{i,t+1} &= \theta_{i,t} - \alpha \nabla f(\theta_{i,t}; X_i) - \beta(\theta_{i,t} - \theta_{j_{i,t},t}) && (3.61) \\ &= (1-\beta)\theta_{i,t} + \beta\theta_{j_{i,t},t} - \alpha \nabla f(\theta_{i,t}; X_i). && (3.62) \end{aligned}$$

Gossip with fresh parameters (gradient step before gossip)

Consider a two-step distributed consensus gradient update:

$$\begin{aligned} \theta'_{i,t} &= \theta_{i,t} - \alpha \nabla f(\theta_{i,t}; X_i) && (3.63) \\ \theta_{i,t+1} &= \theta'_{i,t} - \beta \left(\theta'_{i,t} - \frac{1}{p} \sum_{j=1}^{p} \theta'_{j,t} \right). && (3.64) \end{aligned}$$

If we replace the distributed mean $\frac{1}{p}\sum_{j=1}^{p} \theta'_{j,t}$ with an unbiased one-sample estimator $\theta'_{j_{i,t},t}$, such that $j_{i,t} \sim \text{Uniform}(\{1,\ldots,p\})$ and $\mathbb{E}[\theta'_{j_{i,t},t}] = \frac{1}{p}\sum_{j=1}^{p} \theta'_{j,t}$, then we derive the gossiping SGD update:

$$\begin{aligned} \theta'_{i,t} &= \theta_{i,t} - \alpha \nabla f(\theta_{i,t}; X_i) && (3.65) \\ \theta_{i,t+1} &= \theta'_{i,t} - \beta(\theta'_{i,t} - \theta'_{j_{i,t},t}) && (3.66) \\ &= (1-\beta)\theta'_{i,t} + \beta\theta'_{j_{i,t},t}. && (3.67) \end{aligned}$$

ImageNet data augmentation

We found that multi-scale training could be a significant performance bottleneck due to the computational overhead of resizing images, even when using multiple threads and asynchronous data loading. To remedy this, we used fast CUDA implementations of linear and cubic interpolation filters to perform image scaling during training on the GPU. We also preprocessed ImageNet images such that their largest dimension was no larger than the maximum scale (in our case, 480 pixels).

ResNet implementation

We implemented ResNet-18 using stacked residual convolutional layers with 1×1 projection shortcuts. We used the convolution and batch normalization kernels from cuDNNv4. The highest ImageNet validation set accuracy (center crop, top-1) our implementation of ResNets achieved was about 68.7% with the aforementioned multi-scale data augmentation; we note that researchers at Facebook independently reproduced ResNet with a more sophisticated data augmentation scheme and achieved 69.6% accuracy using the same evaluation methodology on their version of ResNet-18.[2]

3.7 Discussion

In this chapter, we proposed a distributed training method, gossiping SGD, for the large scale supervised learning of visual perception. Unlike previous approaches to distributing SGD for large scale training, gossiping SGD avoids synchronization and has no central parameter server bottleneck. In our experiments, we found that gossiping SGD scales well up to around 32 nodes, but above 32 nodes synchronous SGD is able to converge to better accuracy.

Revisiting the questions we asked in the beginning:

a. *How fast do asynchronous and synchronous SGD algorithms converge at both the beginning of training (large step sizes) and at the end of training (large step sizes)?*

   Up to around 32 nodes, asynchronous SGD can converge to a given accuracy level in fewer hours than all-reduce synchronous SGD when the step size is large. When the step size is small (roughly 0.001 or less), gossiping can converge faster than elastic averaging, but all-reduce synchronous SGD converges most consistently.

b. *How does the convergence behavior of asynchronous and synchronous distributed SGD vary with the number of nodes?*

   Both elastic averaging and gossiping seem to converge faster than synchronous all-reduce synchronous SGD with fewer nodes (up to 16–32 nodes). With more nodes (up to a scale of 100 nodes), all-reduce synchronous SGD can consistently converge to a high-accuracy solution, whereas asynchronous methods seem to plateau at lower accuracy. In particular, the fact that gossiping SGD does not scale as well as does synchronous SGD with more nodes suggests that the asynchrony and the pattern of communication, rather than the amount of communication (both methods have low amounts of communication), are responsible for the difference in convergence.

[2] `https://github.com/facebook/fb.resnet.torch`

Chapter 4

Spatially Parallel Convolutions

4.1 Introduction

One important subtask of visual navigation consists of identifying objects and obstacles in images, for example using methods for object detection or semantic segmentation. Distant or small objects may not be easily perceptible in low resolution images, potentially compromising navigation safety in important applications like self-driving cars. To ground ourselves with some intuition about image resolution in the real world, in Table 4.1 we tabulate the size in pixels of a small 10 cm $\times$ 10 cm object as a function of image resolution and the actual distance of the object from the camera, using camera and lens parameters compatible with those of the KITTI benchmark dataset [24]: namely, we use a focal length of 5 mm and a horizontal sensor size of about 6.4 mm.[1] For example, in a 720p image (1280 $\times$ 720 px), the 10 cm $\times$ 10 cm object at a distance of 100 m from the camera will appear as a single pixel in the image. As the image resolution is increased to that of a UHD image (3840 $\times$ 2160 px), the same object at a distance of 100 m will take up almost 10 pixels, which gives an object detection or semantic segmentation algorithm a bit more input data to work with. In Figure 4.1 we further illustrate the difference in the range of pixels available to the different resolutions (evaluated in Table 4.1) by comparing a bounding box around a pedestrian in a very high resolution image (3840 $\times$ 2160 px) and in a lower resolution image (1392 $\times$ 512 px). Correctly perceiving these kinds of distant or small objects is crucial for enabling safe visual navigation in the real world, necessitating high resolution images and the preservation of high resolution information within a convnet. In this chapter, we describe some implementation considerations for training deep convolutional neural networks with high resolution tensors.

Deep convolutional neural networks (convnets), which form the basis of state-of-the-art models in computer vision, are often trained on GPUs with limited memory capacity; typical

[1] Further details about the sensor setup for KITTI can be found at: `http://www.cvlibs.net/datasets/kitti/setup.php`. The focal length is chosen from the range 4 mm–8 mm for the Edmund Optics NT59-917 lens. The sensor size is for the Point Grey Flea 2 FL2-14S3C-C camera: `https://www.ptgrey.com/flea2-14-mp-color-firewire-1394b-sony-icx267-camera`.

Table 4.1: Pixel footprint of a 10 cm $\times$ 10 cm object varying with distance and image resolution, based on the camera and lens parameters of the KITTI dataset [24].

Distance	KITTI 1392 × 512 px	720p 1280 × 720 px	1080p 1920 × 1080 px	UHD 3840 × 2160 px
10 m	118	100	225	900
20 m	30	25	56	225
50 m	5	4	9	36
100 m	1	1	2	9

high-end GPUs have only 12 GB–16 GB of DRAM or HBM. Limited GPU memory presents an obstacle to training high resolution convnets on semantic segmentation and other tasks [48, 95]. Current frameworks implement data or model parallelism to split tensors onto multiple GPUs [40, 52, 18, 19]. However, data and model parallelism may be insufficient. Other approaches reduce working memory size but increase computation time: these include simply executing convnets on overlapping slices of the input, as well as sophisticated checkpointing techniques [11, 28]. Increasing the amount of downsampling within a convnet does reduce memory usage, but the addition of downsampling is also a destructive change to the model and its learning behavior that reduces the high resolution information we were originally motivated to preserve.

Our contribution is an implementation of spatially parallel convolutions, which uses a technique based on "halo regions" or "ghost zones" originating from stencil codes in scientific computing. Unlike approaches that trade off between memory and computation, spatially parallel convolutions have low overhead in additional computation and communication, while also enabling scaling beyond the limits of data and model parallelism. Spatially parallel convolutions are also a systems-level approach that does not change the model, as is the case when simply more downsampling is applied. Furthermore, unlike previous implementations of "halo regions" or "ghost zones" on GPUs [63, 14], our implementation specifically targets modern multi-GPU systems with high bandwidth inter-GPU interconnects.

In our experiments, we demonstrate that on a modern multi-GPU system, our implementation of spatially parallel convolutions exhibits excellent multi-GPU scaling in computation time and memory usage when applied on tensors with large spatial dimensions. In a few cases, spatial parallelism yields surprising but explainable superlinear speedups. Spatially parallel convolutions are additionally adjoinable: it is easy to backpropagate through spatially parallel convolutions for gradient-based optimization methods such as SGD.

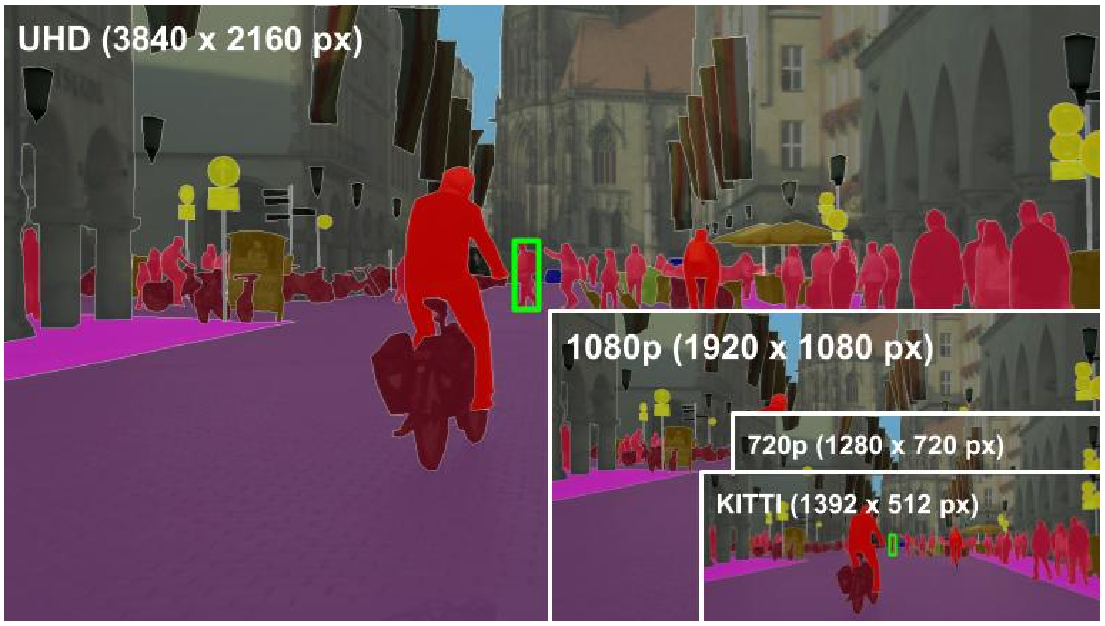

Figure 4.1: Comparing the number of pixels in progressively higher image resolutions with a semantic segmentation mask from the Cityscapes dataset [15]. Overlaid is the same faux bounding box (green) for a pedestrian on both the KITTI- and UHD-resolution images: in the lower resolution (KITTI) image, the bounding box occupies only around 2500–3000 pixels, whereas in the higher resolution (UHD) image, the bounding box occupies around 20000 pixels.

4.2 Related Work

Trading off between memory usage and (re-)computation

One family of approaches to reducing the memory usage of convnets exploits a well known tradeoff between storing the intermediate results of a computation in memory and recomputing the intermediate results. A simple technique from this family of approaches is as follows. For a convnet, one can statically compute the total receptive field in the input corresponding to any "pixel" in the output. When the convnet consists of only spatially local operations, a slice of pixels in the output can be computed using a corresponding slice of pixels in the input that accounts for the total receptive field of the convnet. Using this simple technique, a large output of a convnet that cannot fit in memory can instead be split into non-overlapping slices of pixels, where the computation of each of the non-overlapping output slices can fit in memory. However, the corresponding input slices are overlapping, and often substantially so due to the total receptive field of the convnet, resulting in a signficant amount of duplicate computation. A more sophisticated technique that trades off between memory usage and computation is gradient checkpointing [11, 28], which selectively recomputes some intermediate activation tensors but stores others to bound the amount of computation that must be performed while still reducing peak memory usage. Gradient checkpointing can still substantially increase the amount of required computation, as it may need to recompute whole operations such as an entire convolution. Our work on spatially parallel convolutions on multiple GPUs only adds a small amount of computation and communication overhead, which, in Section 4.4 below, we experimentally show does not add a substantial overhead, and sometimes can even yield a superlinear speedup compared to the baseline single-GPU convolution.

Downsampling

As convnets typically already contain downsampling operations, it is also possible to simply increase the amount of downsampling, yielding a network that contains lower resolution activation tensors which in turn consume less memory than the original network. However, increasing the amount of downsampling also fundamentally changes the nature of the model defined by the convnet and its learning behavior, possibly reducing peak accuracy and changing the optimal hyperparameters for training. Additionally, more downsampling further reduces the availability of high resolution information throughout the convnet, whereas we motivated earlier the need for preserving high resolution information in computer vision, e.g. to detect small objects in images. The selective addition of downsampling, such as in the combination of a low resolution context stream with a high resolution fovea stream [47], may resolve some negative effects on learning but can be domain-specific in its application. In contrast, spatially parallel convolutions are a purely systems-level approach to reducing memory usage that leave the model unchanged, only modifying the dataflow of the convnet without altering the final result of the computation.

Data and model parallelism

In what is commonly referred to as data parallelism for convnets, a minibatch of activations is distributed into multiple local batches, and each local batch is assigned to one processing elements (e.g. one GPU). For example, a minibatch of size 256 can be distributed onto 8 GPUs, where each GPU processes a local batch of size 32. However, a local batch size of one cannot be further distributed using data parallelism alone, and even with a batch size of one, a large enough convnet applied to a high resolution input image will not fit in GPU memory. Our approach of spatially parallel convolutions can be understood as distributing tensors along the spatial axes, sidestepping limits to data parallelism along only the batch size axis.

Alternatively, it is possible to apply model parallelism to convnets, which generically means distributing the parameters between GPUs. For convnets, model parallelism can also be casted as distributing the channels of activations onto the individual GPUs. When the input and output activations of a single convolution operation possess channels that are distributed on different GPUs, a copy must be performed, requiring an amount of communication up to the total size of the activation tensor. In other words, the communication overhead of model parallelism can be substantial for convnets. Furthermore, naive implementations of model parallelism serialize the computation among GPUs so that only one GPU is performing a convolution at a given time; this somewhat decreases the communication burden, which is reduced to a series of hand-offs of activation tensors between GPUs, but at the significant expense of no parallelization of computation between multiple GPUs. Our spatially parallel convolutions have low communication overhead and effectively parallelize computation between multiple GPUs.

4.3 Spatially Parallel Convolutions

In this section, we describe how spatially parallel convolutions work. The basic idea behind spatially parallel convolutions is as follows. In the field of stencil codes which also deals with spatial data, a tensor is spatially distributed into partitions, each of which lies on a single processing element. From the point of view of any single partition, the spatially adjacent partitions possess remote boundary data that must be communicated before performing local computation; this boundary data is called the "halo region." The implementation technique used by stencil codes is to slightly pad each partition with a "ghost zone" buffer that receives a copy of the halo region from adjacent partitions. In Figure 4.2 below, we illustrate a tensor partition along the spatial axes and the halo regions corresponding to a spatially parallel 3×3 convolution. The combined halo region copying and ghost zone padding leads to a small communication and memory overhead while preserving the total number of arithmetic operations. This is the basic approach we follow when implementing spatially parallel convolutions; we note that the same idea can be implemented for other spatially local operations, e.g. pooling.

In the rest of the section, we more precisely define the forward and backward passes of spatially parallel convolutions. We use the following notation: $P =$ number of GPUs, $N =$ batch size, $C =$ channels, $H =$ height, $W =$ width, $K =$ conv kernel size, $D =$ dilation rate, and $R =$ halo region size. For simplicity, our exposition assumes that tensors are in $NCHW$-layout, i.e. the leading dimension is the width axis. Additionally we assume each partition is a horizontal stripe of the whole tensor.

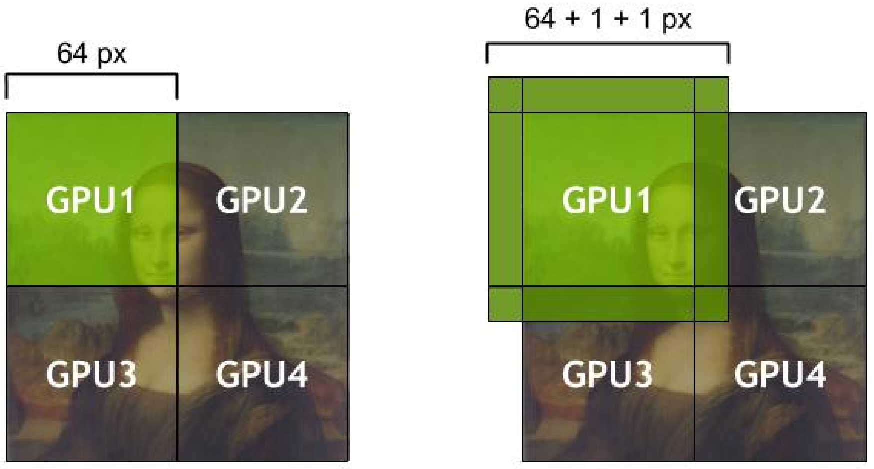

Figure 4.2: An illustration of a 128×128 tensor partitioned along both spatial axes for a spatially parallel 3×3 convolution: (left) without halo regions and (right) with halo regions highlighted.

Forward pass

First, we define the forward pass of spatially parallel convolutions. Let w be the convolution kernel, let x be the (unpartitioned) input tensor, and let y be the (unpartitioned) output tensor. Let x_p represent a partition of the tensor x, where the partitions are indexed by $p = 1$ to P and have height H/P. In vectorized notation, the forward pass is a halo region *exchange* and consists of the following operations, with loop index $i = 0$ to $R - 1$:

$$x_p[:, :, H/P + i, :] \leftarrow x_{p+1}[:, :, i, :] \tag{4.1}$$
$$x_p[:, :, -1 - i, :] \leftarrow x_{p-1}[:, :, H/P - 1 - i, :] \tag{4.2}$$
$$y_p \leftarrow \mathrm{Conv}(w, x_p). \tag{4.3}$$

Above, out-of-bounds array indices lie within the halo region.

Backward pass

Next, we define the backward pass of spatially parallel convolutions. Let $\Delta x \triangleq \nabla_x f$ be the gradient of the tensor x with respect to a function f, and let Δx_p represent a partition of Δx. The backward pass is the "transpose" of the forward pass, taking the form of a halo region *reduction* and consisting of the following, again with loop index $i = 0$ to $R - 1$:

$$\Delta w_p \leftarrow \text{ConvBwdKernel}(x_p, \Delta y_p) \tag{4.4}$$

$$\Delta w \leftarrow \sum_{p=1}^{P} \Delta w_p \tag{4.5}$$

$$\Delta x_p \leftarrow \text{TransposeConv}(w, \Delta y_p) \tag{4.6}$$

$$\Delta x_p[:,:,i,:] \leftarrow \Delta x_p[:,:,i,:] + \Delta x_{p-1}[:,:,H/P + i,:] \tag{4.7}$$

$$\Delta x_p[:,:,H/P - 1 - i,:] \leftarrow \Delta x_p[:,:,H/P - 1 - i,:] + \Delta x_{p+1}[:,:,-1 - i,:]. \tag{4.8}$$

The sum into Δw is done by an all-reduce operation, which can be executed asynchronously.

Implementation

Our preliminary implementation is essentially bulk-synchronous, consisting of a communication phase, during which halo region data are exchanged between adjacent GPUs, and a computation phase, during which convolution routines are executed independently on each GPU; a synchronization point separates the phases. A more efficient approach would be to fuse the halo region communication with the convolution routine, taking advantage of a low latency interconnect such as NVLink; we leave this for future work.

4.4 Experiments

We confirmed the correctness of our spatially parallel convolution implementation by training spatially parallel ResNet architectures on ImageNet [37, 79]. For example, on a spatially parallel version of ResNet-18 trained using up to 4 GPUs, we attained top-1, single-crop ILSVRC2012 validation accuracy of 69.5%, which is close to the peak accuracy of 69.57% attained by a baseline version of ResNet-18 trained with similar data augmentation but slight variations in network architecture.[2]

To evaluate our implementation of spatially parallel convolutions, our microbenchmarks compared two approaches for a given problem size: (a) computing a convolution when the entire problem size fits on a single GPU; and (b) computing a spatially parallel convolution when the problem size is partitioned across multiple GPUs. Our evaluation platform was a Tesla V100 4xGPU system with NVLink 2.0 interconnect via the Amazon EC2 P3.8x GPU instance type. We used convolution routines from cuDNN 7 with CUDA 9.0. We exclusively ran single-precision floating point operations. We also restricted the maximum workspace size to 4 GB.

The results of our microbenchmarks are summarized in Tables 4.2 and 4.3. The numbers suggest that for large problem sizes ($H, W \gtrsim 256$) spatially parallel convolutions on multiple

[2] `https://github.com/facebook/fb.resnet.torch`

GPUs can achieve roughly linear speedup up to 4 GPUs compared to convolutions on a single GPU. Dilated convolutions are harder to parallelize than regular convolutions, but on larger problems ($H, W \gtrsim 512$) spatially parallel dilated convolutions also scale well to 4 GPUs.

In a few cases (especially the results for the backward pass), we actually observe *superlinear* speedups, which is surprising but explainable. For instance, during the backward pass of a spatially parallel 3×3 convolution on 4 GPUs with problem size $H, W = 512$ (which has an empirical speedup of 4.7×; c.f. Table 4.2), each individual GPU executes a single-GPU convolution on the padded subproblem of size $H_{\text{sub}} = 130$, $W_{\text{sub}} = 514$. The superlinear speedup then arises in large part when a 512×512 convolution on a single GPU takes more than 4× the wall-clock time of the 130×514 convolution also on a single GPU. We treat the implementation details of the single-GPU convolution routines as a "black box," but we note that a 512×512 convolution is quite a large problem size. One possibility then is that there is simply further room for improvement in tuning the convolution routine for larger problem sizes.

Table 4.2: Spatially parallel convolution (fp32): $N = 32$, $C = 64$, $K = 3$, $D = 1$, $R = 1$. For each benchmark, we show both average wall-clock time (in ms) over 1000 trials and speedup over the single-GPU case, as well as the per-GPU memory usage (in MB) and fraction of the memory used compared to the single-GPU case.

Problem size	GPUs	Fwd. wall-clock	Bwd. wall-clock	Memory per GPU
$H = W = 128$	1 GPU	2.56 ms (1.0×)	6.63 ms (1.0×)	256 MB (1.00×)
	2 GPUs	1.52 ms (1.7×)	3.50 ms (1.9×)	134 MB (0.52×)
	4 GPUs	1.23 ms (2.1×)	2.33 ms (2.8×)	69 MB (0.27×)
$H = W = 256$	1 GPU	10.02 ms (1.0×)	26.81 ms (1.0×)	1024 MB (1.00×)
	2 GPUs	5.34 ms (1.9×)	11.79 ms (2.3×)	524 MB (0.51×)
	4 GPUs	3.11 ms (3.2×)	6.96 ms (3.9×)	266 MB (0.26×)
$H = W = 512$	1 GPU	45.15 ms (1.0×)	126.11 ms (1.0×)	4096 MB (1.00×)
	2 GPUs	20.18 ms (2.2×)	60.15 ms (2.1×)	2072 MB (0.50×)
	4 GPUs	10.65 ms (4.2×)	26.76 ms (4.7×)	1044 MB (0.25×)

4.5 Discussion

High resolutions are critical to the safe operation of visual navigation systems. However, it is difficult to train high resolution convnets due to limited GPU memory capacity, and existing approaches to reducing memory usage when training convnets either add significant computational overhead or are otherwise insufficient. Our contribution in this chapter is an implementation of spatially parallel convolutions for efficiently training high resolution

Table 4.3: Spatially parallel dilated convolution (fp32): $N = 32$, $C = 64$, $K = 3$, $D = 3$, $R = 3$.

Problem size	GPUs	Fwd. wall-clock	Bwd. wall-clock	Memory per GPU
$H = W = 128$	1 GPU	3.92 ms (1.0×)	11.10 ms (1.0×)	256 MB (1.00×)
	2 GPUs	2.69 ms (1.5×)	6.49 ms (1.7×)	147 MB (0.57×)
	4 GPUs	2.44 ms (1.6×)	5.55 ms (2.0×)	80 MB (0.31×)
$H = W = 256$	1 GPU	15.39 ms (1.0×)	43.99 ms (1.0×)	1024 MB (1.00×)
	2 GPUs	8.43 ms (1.8×)	22.66 ms (1.9×)	549 MB (0.54×)
	4 GPUs	5.33 ms (2.9×)	12.71 ms (3.5×)	287 MB (0.28×)
$H = W = 512$	1 GPU	61.26 ms (1.0×)	192.11 ms (1.0×)	4096 MB (1.00×)
	2 GPUs	31.68 ms (1.9×)	95.42 ms (2.0×)	2120 MB (0.52×)
	4 GPUs	17.51 ms (3.5×)	46.72 ms (4.1×)	1085 MB (0.26×)

convnets on multiple GPUs, thereby partitioning memory usage between GPUs while also parallelizing the computation with low communication overhead. We evaluated the performance of our implementation of spatially parallel convolutions on a multi-GPU system and found excellent scaling in computation time and memory usage.

Spatial parallelism can be composed with other complementary approaches for parallelism and memory reduction for convolutional networks. Spatially parallel convolutions are one extreme for partitioning or splitting tensors into more general distributed tensor representations [86]. Futhermore, the combination of spatial parallelism with both data and model parallelism is a design point, for which an optimum exists that minimizes the sum cost of computation and communication [25]. Gradient checkpointing trades off between working memory size and recomputation of intermediate values in a computation graph [11, 28]; its combination with spatial parallelism further reduces per-GPU memory usage, while retaining favorable scaling on multiple GPUs.

An open source version of our preliminary implementation is available at:
`https://github.com/peterhj/arraydiff_cuda`.

Chapter 5

Conclusion

This dissertation examined learning and systems challenges related to visual navigation tasks. First, we approached visual navigation tasks within the setting of deep reinforcement learning under partial observation. Previous deep reinforcement learning approaches based on policy gradients and Q-learning do not adequately address partial observation while remaining sample-efficient. We described a novel deep reinforcement learning algorithm, advantage-based regret minimization (ARM), which is able to learn robust policies in visual navigation tasks under partial observation perturbations. Second, we considered issues related to the large scale training of deep convolutional neural networks for visual perception. Large datasets require distributed training, but previous synchronous and asynchronous methods for distributed training are slowed down by synchronization and communication bottlenecks. We presented an algorithm, gossiping SGD, that can scale well on clusters without hitting those bottlenecks. Finally, the training of convnets on high-resolution images is limited by GPU memory capacity. We then showed that spatially parallel convolutions can partition memory usage, while simultaneously scaling well to multiple GPUs.

5.1 Future Directions

Counterfactual regret algorithms in continuous action spaces. In our work on advantage-based regret minimization (ARM), we exclusively evaluated partially observed environments with discrete action spaces. However, continuous action spaces are a prominent feature of real-world domains such as robotics. How can one extend ARM and the family of related counterfactual regret minimization algorithms to continuous action spaces?

One obstacle to a continuous variant of ARM could be summarized as a single technical issue: regret matching-style policies involve a normalization by the sum of counterfactual regrets over a discrete action space, which doesn't apply to continuous action spaces. To our knowledge, the only known work on addressing continuous action spaces in CFR-style algorithms involves action space discretization [54]. Because ARM involves regret approximation which is analogous to value function approximation, some promising approaches may be found in how existing value function methods for deep reinforcement learning have been adapted to continuous action spaces, e.g. continuous-action representations of the action-value function [30] or sampling actions from learned implicit distributions [31].

Simulating visual environments. One aspect we only briefly addressed concerns the nature of the external environment where learning occurs. For our work on ARM, we posed the navigation problem as model-free reinforcement learning, in which the environment consists of a black-box simulation taking actions as its inputs and returning visual observations and reward signals as its outputs. In our experiments, we were fortunately able to utilize high quality environments based on the 3D video games Doom (via ViZDoom [50]) and Minecraft (via Malmö [46]). On the other hand, our work on gossiping SGD and spatially parallel convolutions was motivated in large part by the scaling challenges that face current and future real-world deployments of visual perception systems, e.g. those in self-driving cars. Some promising preliminary results on bridging the gap between simulated visual environments and the real world include the transfer of policies trained on virtual scenes to the real world [60, 92, 80], renderable representations of visual scenes with partial observability effects [22], and simulation of alternative visual modalities, e.g. LIDAR [102].

Finally, although we described systems issues in learning from the perspective of implementing neural nets, we did not discuss implementation considerations of the other important component, the simulator. As the rate of learning in simulation can also be limited by a slow simulator, then future systems advances in the implementation of simulations can also directly transfer to more iterations of learning per wall-clock time interval, potentially yielding better performing models (in terms of the task-specific metric) within the same training time.

Bibliography

[1] Oron Anschel, Nir Baram, and Nahum Shimkin. “Averaged-DQN: Variance Reduction and Stabilization for Deep Reinforcement Learning”. In: *Proceedings of the 34th International Conference on Machine Learning*. 2017, pp. 176–185.

[2] Marc G. Bellemare, Will Dabney, and Rémi Munos. “A Distributional Perspective on Reinforcement Learning”. In: *Proceedings of the 34th International Conference on Machine Learning*. 2017.

[3] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. “The Arcade Learning Environment: An evaluation platform for general agents”. In: *Journal of Artificial Intelligence Research* 47 (2013), pp. 253–279.

[4] Mark G. Bellemare, Georg Ostrovski, Arthur Guez, Philip S. Thomas, and Rémi Munos. “Increasing the Action Gap: New Operators for Reinforcement Learning”. In: *AAAI Conference on Artificial Intelligence*. 2016.

[5] M. Botvinick, D.G.T. Barrett, P. Battaglia, N. de Freitas, D. Kumaran, J. Z Leibo, T. Lillicrap, J. Modayil, S. Mohamed, N.C. Rabinowitz, D.J. Rezende, A. Santoro, T. Schaul, C. Summerfield, G. Wayne, T. Weber, D. Wierstra, S. Legg, and D. Hassabis. “Building Machines that Learn and Think for Themselves: Commentary on Lake et al., Behavioral and Brain Sciences, 2017”. *arXiv preprint arXiv:1711.08378*. 2017.

[6] Michael Bowling, Neil Burch, Michael Johanson, and Oskari Tammelin. “Heads-up limit hold’em poker is solved”. In: *Science* 347.6218 (2015), pp. 145–149.

[7] Stephen Boyd, Arpita Ghosh, Balaji Prabhakar, and Devavrat Shah. “Randomized Gossip Algorithms”. In: *IEEE/ACM Transactions on Networking* 14.SI (2006), pp. 2508–2530.

[8] Noam Brown and Tuomas Sandholm. “Solving Imperfect-Information Games via Discounted Regret Minimization”. *arXiv preprint arXiv:1809.04040*. 2018.

[9] Nicolò Cesa-Bianchi and Gábor Lugosi. “Potential-Based Algorithms in On-Line Prediction and Game Theory”. In: *Machine Learning* 51.3 (2003), pp. 239–261.

[10] Jianmin Chen, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. “Revisiting Distributed Synchronous SGD”. *arXiv preprint arXiv:1604.00981*. 2016.

[11] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. "Training Deep Nets with Sublinear Memory Cost". *arXiv preprint arXiv:1604.06174*. 2016.

[12] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhammer. "cuDNN: Efficient Primitives for Deep Learning". *arXiv preprint arXiv:1410.0759*. 2014.

[13] Trishul Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. "Project Adam: Building an Efficient and Scalable Deep Learning Training System". In: *11th USENIX Symposium on Operating Systems Design and Implementation*. 2014, pp. 571–582.

[14] Adam Coates, Brody Huval, Tao Wang, David J. Wu, Andrew Y. Ng, and Bryan Catanzaro. "Deep learning with COTS HPC Systems". In: *Proceedings of the 30th International Conference on Machine Learning*. 2013.

[15] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. "The Cityscapes Dataset for Semantic Urban Scene Understanding". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016.

[16] Dipankar Das, Sasikanth Avancha, Dheevatsa Mudigere, Karthikeyan Vaidynathan, Srinivas Sridharan, Dhiraj Kalamkar, Bharat Kaul, and Pradeep Dubey. "Distributed Deep Learning Using Synchronous Stochastic Gradient Descent". *arXiv preprint arXiv:1602.06709*. 2016.

[17] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V. Le, and Andrew Y. Ng. "Large Scale Distributed Deep Networks". In: *Advances in Neural Information Processing Systems 25*. 2012, pp. 1223–1231.

[18] Tim Dettmers. "How to Parallelize Deep Learning on GPUs Part 1/2: Data Parallelism". 2014. URL: `http://timdettmers.com/2014/10/09/deep-learning-data-parallelism/`.

[19] Tim Dettmers. "How to Parallelize Deep Learning on GPUs Part 2/2: Model Parallelism". 2014. URL: `http://timdettmers.com/2014/11/09/model-parallelism-deep-learning/`.

[20] Travis Dick. "Policy Gradient Reinforcement Learning Without Regret". MA thesis. University of Alberta, 2015.

[21] Alexey Dosovitskiy and Vladlen Koltun. "Learning to Act by Predicting the Future". In: *International Conference on Learning Representations*. 2017.

[22] SM Ali Eslami, Danilo Jimenez Rezende, Frederic Besse, Fabio Viola, Ari S Morcos, Marta Garnelo, Avraham Ruderman, Andrei A Rusu, Ivo Danihelka, Karol Gregor, David P Reichert, Lars Buesing, Theophane Weber, Oriol Vinyals, Dan Rosenbaum, Neil Rabinowitz, Helen King, Chloe Hillier, Matt Botvinick, Daan Wierstra, Koray Kavukcuoglu, and Demis Hassabis. "Neural scene representation and rendering". In: *Science* 360.6394 (2018), pp. 1204–1210.

[23] Gabriele Farina, Christian Kroer, and Tuomas Sandholm. "Online Convex Optimization for Sequential Decision Processes and Extensive-Form Games". *arXiv preprint arXiv:1809.03075*. 2018.

[24] Andreas Geiger, Philip Lenz, and Raquel Urtasun. "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite". In: *2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012.

[25] Amir Gholami, Ariful Azad, Peter Jin, Kurt Keutzer, and Aydin Buluc. "Integrated Model, Batch and Domain Parallelism in Training Neural Networks". *arXiv preprint arXiv:1712.04432*. 2018.

[26] Geoffrey J. Gordon. "No-regret Algorithms for Online Convex Programs". In: *Advances in Neural Information Processing Systems*. 2007.

[27] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour". *arXiv preprint arXiv:1706.02677*. 2017.

[28] Andrūnas Gruslys, Rémi Munos, Ivo Danihelka, Marc Lanctot, and Alex Graves. "Memory-Efficient Backpropagation Through Time". In: *Advances in Neural Information Processing Systems*. 2016.

[29] Shixiang Gu, Timothy Lillicrap, Zoubin Ghahramani, Richard E. Turner, Bernhard Schölkopf, and Sergey Levine. "Interpolated Policy Gradient: Merging On-Policy and Off-Policy Gradient Estimation for Deep Reinforcement Learning". *arXiv preprint arXiv:1706.00387*. 2017.

[30] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. "Continuous Deep Q-Learning with Model-based Acceleration". In: *Proceedings of the 33rd International Conference on Machine Learning*. 2016.

[31] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. "Reinforcement Learning with Deep Energy-Based Policies". In: *International Conference on Machine Learning*. 2017.

[32] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *Proceedings of the 35th International Conference on Machine Learning*. 2018.

[33] Sergiu Hart and Andreu Mas-Colell. "A Simple Adaptive Procedure Leading to Correlated Equilibrium". In: *Econometrica* 68.5 (2000), pp. 1127–1150.

[34] Hado van Hasselt, Arthur Guez, and David Silver. “Deep Reinforcement Learning and Double Q-Learning”. In: *AAAI Conference on Artificial Intelligence*. 2016.

[35] Hado van Hasselt and Marco A. Wiering. “Reinforcement Learning in Continuous Action Spaces”. In: *Proceedings of the 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning*. 2007, pp. 272–279.

[36] Matthew Hausknecht and Peter Stone. “Deep Recurrent Q-Learning for Partially Observable MDPs”. *arXiv preprint arXiv:1507.06527*. 2017.

[37] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep Residual Learning for Image Recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016.

[38] Nicolas Heess, Jonathan J. Hunt, Timothy P. Lillicrap, and David Silver. “Memory-based control with recurrent neural networks”. *arXiv preprint arXiv:1512.04455*. 2015.

[39] Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B. Gibbons, Garth A. Gibson, Greg Ganger, and Eric P. Xing. “More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server”. In: *Advances in Neural Information Processing Systems 26*. 2013, pp. 1223–1231.

[40] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, and Kurt Keutzer. “Firecaffe: near-linear acceleration of deep neural network training on compute clusters”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016.

[41] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *Proceedings of the 32nd International Conference on Machine Learning*. 2015, pp. 448–456.

[42] Tommi Jaakkola, Satinder P. Singh, and Michael I. Jordan. “Reinforcement Learning Algorithm for Partially Observable Markov Decision Problems”. In: *Advances in Neural Information Processing Systems*. 1994.

[43] Peter H. Jin, Qiaochu Yuan, Forrest Iandola, and Kurt Keutzer. “How to scale distributed deep learning?” In: *NIPS 2016 ML Systems Workshop*. 2016.

[44] Peter Jin, Boris Ginsburg, and Kurt Keutzer. “Spatially Parallel Convolutions”. In: *ICLR 2018 Workshop Track*. 2018.

[45] Peter Jin, Kurt Keutzer, and Sergey Levine. “Regret Minimization for Partially Observable Deep Reinforcement Learning”. In: *Proceedings of the 35th International Conference on Machine Learning*. 2018.

[46] Matthew Johnson, Katja Hofmann, Tim Hutton, and David Bignell. “The Malmo Platform for Artificial Intelligence Experimentation”. In: *Proceedings of the 25th International Joint Conference on Artificial Intelligence*. 2016.

[47] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. “Large-scale Video Classification with Convolutional Neural Networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2014.

[48] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. “Progressive Growing of GANs for Improved Quality, Stability, and Variation”. *arXiv preprint arXiv:1710.10196.* 2017.

[49] David Kempe, Alin Dobra, and Johannes Gehrke. “Gossip-Based Computation of Aggregate Information”. In: *Proceedings of the 44th Annual IEEE Conference on Foundations of Computer Science.* 2003, pp. 482–491.

[50] Michal Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. “ViZDoom: A Doom-based AI Research Platform for Visual Reinforcement Learning”. *arXiv preprint arXiv:1605.02097.* 2016.

[51] Diederik P. Kingma and Jimmy Lei Ba. “Adam: A Method for Stochastic Optimization”. In: *International Conference on Learning Representations.* 2015.

[52] Alex Krizhevsky. “One weird trick for parallelizing convolutional neural networks”. *arXiv preprint arXiv:1404.5997.* 2014.

[53] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems 25.* 2012, pp. 1097–1105.

[54] Christian Kroer and Tuomas Sandholm. “Discretization of Continuous Action Spaces in Extensive-Form Games”. In: *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems.* 2015.

[55] Christian Kroer and Tuomas Sandholm. “Extensive-Form Game Abstraction With Bounds”. In: *Proceedings of the 15th ACM Conference on Economics and Computation.* 2014.

[56] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. “End-to-end training of deep visuomotor policies”. In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 1334–1373.

[57] Mu Li, David G. Andersen, Alex J. Smola, and Kai Yu. “Communication Efficient Distributed Machine Learning with the Parameter Server”. In: *Advances in Neural Information Processing Systems 27.* 2014, pp. 19–27.

[58] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. “Continuous control with deep reinforcement learning”. In: *International Conference on Learning Representations.* 2016.

[59] Michael L. Littman. “Markov games as a framework for multi-agent reinforcement learning”. In: *Proceedings of the 11th International Conference on Machine Learning.* 1994, pp. 157–163.

[60] Jeffrey Mahler, Jacky Liang, Sherdil Niyaz, Michael Laskey, Richard Doan, Xinyu Liu, Juan Aparicio Ojea, and Ken Goldberg. “Dex-Net 2.0: Deep Learning to Plan Robust Grasps with Synthetic Point Clouds and Analytic Grasp Metrics”. In: *Proceedings of Robotics: Science and Systems*. 2017.

[61] Tambet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. “Teacher-Student Curriculum Learning”. *arXiv preprint arXiv:1707.00183*. 2017.

[62] Andrew Kachites McCallum. “Efficient Exploration in Reinforcement Learning with Hidden State”. In: *AAAI Fall Symposium on Model-directed Autonomous Systems.* 1997.

[63] Paulius Micikevicius. “3D Finite Difference Computation on GPUs using CUDA”. In: *Proceedings of the 2nd Workshop on General Purpose Processing on Graphics Processing Units*. 2009, pp. 79–84.

[64] 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.* 2016, pp. 1928–1937.

[65] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. “Playing Atari with Deep Reinforcement Learning”. *arXiv preprint arXiv:1312.5602*. 2013.

[66] 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”. In: *Nature* 518.7540 (2015), pp. 529–533.

[67] Ofir Nachum, Mohammad Norouzi, Kelvin Xu, and Dale Schuurmans. “Bridging the Gap Between Value and Policy Based Reinforcement Learning”. In: *Advances in Neural Information Processing Systems*. 2017.

[68] Yuri Nesterov. “A Method of Solving a Convex Programming Problem with Convergence Rate O(1/k2)”. In: *Soviet Mathematics Doklady* (1983).

[69] Yurii Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course.* Vol. 87. 2013.

[70] Oak Ridge Leadership Computing Facility (OLCF). “Introducing Titan: Advancing the Era of Accelerated Computing”. `http://www.olcf.ornl.gov/titan`. 2015.

[71] Brendan O’Donoghue, Rémi Munos, Koray Kavukcuoglu, and Volodymyr Mnih. “Combining policy gradient and Q-learning”. *arXiv preprint arXiv:1611.01626*. 2017.

[72] Junhyuk Oh, Valliappa Chockalingam, Satinder Singh, and Honglak Lee. “Control of Memory, Active Perception, and Action in Minecraft”. In: *Proceedings of The 33rd International Conference on Machine Learning*. 2016, pp. 2790–2799.

[73] Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L Lewis, and Satinder Singh. “Action-Conditional Video Prediction using Deep Networks in Atari Games”. In: *Advances in Neural Information Processing Systems*. 2015.

[74] Xue Bin Peng, Glen Berseth, and Michiel van de Penne. “Terrain-Adaptive Locomotion Skills Using Deep Reinforcement Learning”. In: *ACM Transactions on Graphics* 35.4 (2016), p. 81.

[75] S. Sundhar Ram, A. Nedić, and V. V. Veeravalli. “Asynchronous Gossip Algorithms for Stochastic Optimization”. In: *Proceedings of the 48th IEEE Conference on Decision and Control*. 2009, pp. 3581–3856.

[76] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. “Hogwild: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent”. In: *Advances in Neural Information Processing Systems 24*. 2011, pp. 693–701.

[77] Stéphane Ross and J. Andrew Bagnell. “Reinforcement and Imitation Learning via Interactive No-Regret Learning”. *arXiv preprint arXiv:1406.5979*. 2014.

[78] Stéphane Ross, Geoffrey J. Gordon, and J. Andrew 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*. 2011, pp. 627–635.

[79] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Fei-Fei Li. “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision* 115.3 (2015), pp. 211–252.

[80] Fereshteh Sadeghi, Alexander Toshev, Eric Jang, and Sergey Levine. “Sim2Real View Invariant Visual Servoing by Recurrent Control”. *arXiv preprint arXiv:1712.07642*. 2017.

[81] Tuomas Sandholm and Satinder Singh. “Lossy Stochastic Game Abstraction with Bounds”. In: *Proceedings of the 13th ACM Conference on Economics and Computation*. 2012.

[82] John Schulman, Xi Chen, and Pieter Abbeel. “Equivalence Between Policy Gradients and Soft Q-Learning”. *arXiv preprint arXiv:1704.06440*. 2017.

[83] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. “Trust region policy optimization”. In: *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*. 2015, pp. 1889–1897.

[84] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. "High-Dimensional Continuous Control Using Generalized Advantage Estimation". *arXiv preprint arXiv:1506.02438*. 2016.

[85] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. "On Parallelizability of Stochastic Gradient Descent for Speech DNNs". In: *Proceedings of the 2014 IEEE Conference on Acoustics, Speech, and Signal Processing*. 2014, pp. 235–239.

[86] Noam Shazeer, Youlong Cheng, Niki Parmar, Dustin Tran, Ashish Vaswani, Penporn Koanantakool, Peter Hawkins, HyoukJoong Lee, Mingsheng Hong, Cliff Young, Ryan Sepassi, and Blake Hechtman. "Mesh-TensorFlow: Deep Learning for Supercomputers". In: *Advances in Neural Information Processing Systems*. 2018.

[87] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". *arXiv preprint arXiv:1409.1556*. 2014.

[88] Csaba Szepesvári. "The Asymptotic Convergence-Rate of Q-learning". In: *Advances in Neural Information Processing Systems*. 1998.

[89] Oskari Tammelin. "Solving Large Imperfect Information Games Using CFR+". *arXiv preprint arXiv:1407.5042*. 2014.

[90] Oskari Tammelin, Neil Burch, Michael Johanson, and Michael Bowling. "Solving Heads-up Limit Texas Hold'em". In: *Proceedings of the 24th International Joint Conference on Artificial Intelligence*. 2015.

[91] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. "Optimization of Collective Communication Operations in MPICH". In: *International Journal of High Performance Computing Applications* 19.1 (2005), pp. 49–66.

[92] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. "Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017.

[93] Behrouz Touri, A. Nedić, and S. Sundhar Ram. "Asynchronous Stochastic Convex Optimization for Random Networks: Error Bounds". In: *Information Theory and Applications Workshop (ITA) 2010*. 2010, pp. 1–10.

[94] David W. Walker and Jack J. Dongarra. "MPI: A Standard Message Passing Interface". In: *Supercomputer* 12 (1996), pp. 56–68.

[95] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. "High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs". *arXiv preprint arXiv:1711.11585*. 2017.

[96] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. "Sample Efficient Actor-Critic with Experience Replay". *arXiv preprint arXiv:1611.01224*. 2017.

[97] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. "Dueling Network Architectures for Deep Reinforcement Learning". In: *Proceedings of the 33rd International Conference on Machine Learning*. 2016, pp. 1995–2003.

[98] Kevin Waugh, Dustin Morrill, J. Andrew Bagnell, and Michael Bowling. "Solving Games with Functional Regret Estimation". In: *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*. Supplementary material in *arXiv preprint arXiv:1411.7974*. 2015.

[99] Ren Wu, Shengen Yan, Yi Shan, Qingqing Dang, and Gang Sun. "Deep Image: Scaling up Image Recognition". *arXiv preprint arXiv:1501.02876*. 2015.

[100] Yang You, Igor Gitman, and Boris Ginsburg. "Large Batch Training of Convolutional Networks". *arXiv preprint arXiv:1708.03888*. 2017.

[101] Yang You, Zhao Zhang, Cho-Jui Hsieh, James Demmel, and Kurt Keutzer. "ImageNet Training in Minutes". In: *Proceedings of the 47th International Conference on Parallel Processing*. 2018.

[102] Xiangyu Yue, Bichen Wu, Sanjit A. Seshia, Kurt Keutzer, and Alberto L. Sangiovanni-Vincentelli. "A LiDAR Point Cloud Generator: from a Virtual World to Autonomous Driving". *arXiv preprint arXiv:1804.00103*. 2018.

[103] Sixin Zhang, Anna E. Choromanska, and Yann LeCun. "Deep learning with Elastic Averaging SGD". In: *Advances in Neural Information Processing Systems 28*. 2015, pp. 685–693.

[104] Martin Zinkevich, Michael Johanson, Michael H. Bowling, and Carmelo Piccione. "Regret Minimization in Games with Incomplete Information". In: *Advances in Neural Information Processing Systems*. 2007.