Deeply AggreVaTeD: Differentiable Imitation Learning for Sequential Prediction

Wen Sun [1] **Arun Venkatraman** [1] **Geoffrey J. Gordon** [2] **Byron Boots** [3] **J. Andrew Bagnell** [1]

Abstract

Recently, researchers have demonstrated state-of-the-art performance on sequential prediction problems using deep neural networks and Reinforcement Learning (RL). For some of these problems, oracles that can demonstrate good performance may be available during training, but are not used by plain RL methods. To take advantage of this extra information, we propose *AggreVaTeD*, an extension of the Imitation Learning (IL) approach of Ross & Bagnell (2014). AggreVaTeD allows us to use expressive differentiable policy representations such as deep networks, while leveraging training-time oracles to achieve faster and more accurate solutions with less training data. Specifically, we present two gradient procedures that can learn neural network policies for several problems, including a sequential prediction task and several high-dimensional robotics control problems. We also provide a comprehensive theoretical study of IL that demonstrates that we can expect up to exponentially-lower sample complexity for learning with AggreVaTeD than with plain RL algorithms. Our results and theory indicate that IL (and AggreVaTeD in particular) can be a more effective strategy for sequential prediction than plain RL.

1. Introduction

A fundamental challenge in artificial intelligence, robotics, and language processing is sequential prediction: to reason, plan, and make a sequence of predictions or decisions to minimize accumulated cost, achieve a long-term goal, or optimize for a loss acquired only after many predictions.

Although conventional supervised learning of deep models has been pivotal in advancing performance in sequential prediction problems, researchers are beginning to utilize Reinforcement Learning (RL) methods to achieve even higher performance (Ranzato et al., 2015; Bahdanau et al., 2016; Li et al., 2016). In sequential prediction tasks, future predictions often depend on the history of previous predictions; thus, a poor prediction early in the sequence can lead to high loss (cost) for future predictions. Viewing the predictor as a *policy* π, deep RL algorithms are able to reason about the future accumulated cost in sequential prediction problems. These approaches have dramatically advanced the state-of-the-art on a number of problems including high-dimensional robotics control tasks and video and board games (Schulman et al., 2015; Silver et al., 2016).

In contrast with general reinforcement learning methods, *imitation learning* and related sequential prediction algorithms such as SEARN (Daumé III et al., 2009), DaD (Venkatraman et al., 2015), AggreVaTe (Ross & Bagnell, 2014), and LOLS (Chang et al., 2015b) reduce the sequential prediction problems to supervised learning by leveraging a (near) optimal *cost-to-go oracle* that can be queried for the next (near)-best prediction at any point during training. Specifically, these methods assume access to an oracle that provides an optimal or near-optimal action and the future accumulated loss Q^*, the so-called cost-to-go. For robotics control problems, this oracle may be a human expert guiding the robot during the training phase (Abbeel & Ng, 2004) or the policy from an optimal MDP solver (Ross et al., 2011; Kahn et al., 2016; Choudhury et al., 2017) that is either too slow to use at test time or leverages information unavailable at test time. For sequential prediction problems, an oracle can be constructed by optimization (e.g., beam search) or by a clairvoyant greedy algorithm (Daumé III et al., 2009; Ross et al., 2013; Rhinehart et al., 2015; Chang et al., 2015a) that, given the training data's ground truth, is near-optimal on the task-specific performance metric (e.g., cumulative reward, IoU, Unlabeled Attachment Score, BLEU).

[1]Robotics Institute, Carnegie Mellon University, USA [2]Machine Learning Department, Carnegie Mellon University, USA [3]College of Computing, Georgia Institute of Technology, USA. Correspondence to: Wen Sun <wensun@cs.cmu.edu>.

Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017. Copyright 2017 by the author(s).

Expert, demonstrator, and oracle are used interchangeably.

We stress that the oracle is only required to be available during training. Therefore, the goal of IL is to learn a policy $\hat{\pi}$ with the help of the oracle (π^*, Q^*) during the training session, such that $\hat{\pi}$ achieves similar or better performance at test time when the oracle is unavailable. In contrast to IL, reinforcement learning methods often initialize with a random policy π_0 or cost-to-go estimate Q_0 that may be far from optimal. The optimal policy (or cost-to-go) must be found by exploring, often with random actions.

A classic family of IL methods is to collect data from running the demonstrator or oracle and train a regressor or classifier via supervised learning. These methods (Abbeel & Ng, 2004; Syed et al., 2008; Ratliff et al., 2006; Ziebart et al., 2008; Finn et al., 2016; Ho & Ermon, 2016) learn either a policy $\hat{\pi}^*$ or $\hat{Q}^*$ from a fixed-size dataset pre-collected from the oracle. Unfortunately, these methods exhibit a pernicious problem: they require the training and test data to be sampled from the same distribution, despite the fact they explicitly change the sample policy during training. As a result, policies learned by these methods can fail spectacularly (Ross & Bagnell, 2010). *Interactive* approaches to IL such as SEARN (Daumé III et al., 2009), DAgger (Ross et al., 2011), and AggreVaTe (Ross & Bagnell, 2014) interleave learning and testing to overcome the data mismatch issue and, as a result, work well in practical applications. Furthermore, these interactive approaches can provide strong theoretical guarantees between training time loss and test time performance through a reduction to no-regret online learning.

In this work, we introduce *AggreVaTeD*, a *differentiable* version of AggreVaTe (Aggregate Values to Imitate (Ross & Bagnell, 2014)) which allows us to train policies with efficient gradient update procedures. AggreVaTeD extends and scales interactive IL for use in sequential prediction and challenging continuous robot control tasks. We provide two gradient update procedures: a regular gradient update developed from Online Gradient Descent (OGD) (Zinkevich, 2003) and a natural gradient update (Kakade, 2002; Bagnell & Schneider, 2003), which is closely related to Weighted Majority (WM) (Littlestone & Warmuth, 1994), a popular no-regret algorithm that enjoys an almost dimension-free property (Bubeck et al., 2015).

AggreVaTeD leverages the oracle to learn rich polices that can be represented by complicated non-linear function approximators. Our experiments with deep neural networks on various robotics control simulators and on a dependency parsing sequential prediction task show that AggreVaTeD can achieve expert-level performance and even *super-expert* performance when the oracle is sub-optimal, a result rarely achieved by non-interactive IL approaches. The differentiable nature of AggreVaTeD additionally allows us to employ Recurrent Neural Network policies, e.g., Long Short-Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997), to handle partially observable settings (e.g., observe only partial robot state). Empirical results demonstrate that by leveraging an oracle, IL can learn much faster than RL.

In addition to providing a set of practical algorithms, we develop a comprehensive theoretical study of IL on discrete MDPs. We construct an MDP that demonstrates exponentially better sample efficiency for IL than any RL algorithm. For general discrete MDPs, we provide a regret upper bound for AggreVaTeD with WM, which shows IL can learn dramatically faster than RL. We provide a regret lower bound for any IL algorithm, which demonstrates that AggreVaTeD with WM is near-optimal.

To summarize the contributions of this work: (1) AggreVaTeD allows us to handle continuous action spaces and employ recurrent neural network policies for Partially Observable Markov Decision Processes (POMDPs); (2) understanding IL from a perspective that is related to policy gradient allows us to leverage advances from the well-studied RL policy gradient literature (e.g., gradient variance reduction techniques, efficient natural gradient computation); (3) we provide a new sample complexity study of IL and compare to RL, showing that we can expect up to exponentially lower sample complexity. Our experimental and theoretical results support the proposition:

> Imitation Learning is a more effective strategy than Reinforcement Learning for sequential prediction with near-optimal cost-to-go oracles.

2. Preliminaries

A Markov Decision Process consists of a set of states, actions (that come from a policy), cost (loss), and a model that transitions states given actions. Interestingly, most sequential prediction problems can be framed in terms of MDPs (Daumé III et al., 2009). The actions are the learner's (e.g., RNN's) predictions. The state is then the result of all the predictions made so far (e.g., the dependency tree constructed so far or the words translated so far). The cumulative cost is the performance metric such as (negative) UAS, received at the end (horizon) or after the final prediction. For robotics control problems, the robot's configuration is the state, the controls (e.g., joint torques) are the actions, and the cost is related to achieving a task (e.g., distance walked).

Formally, a finite-horizon Markov Decision Process (MDP) is defined as $(\mathcal{S}, \mathcal{A}, P, C, \rho_0, H)$. Here, $\mathcal{S}$ is a set of S states and $\mathcal{A}$ is a set of A actions; at time step t, P_t is the transition dynamics such that for any $s_t \in \mathcal{S}, s_{t+1} \in \mathcal{S}, a_t \in \mathcal{A}$,

i.e., the regret bound depends on poly-log of the dimension of parameter space.

$P_t(s_{t+1}|s_t, a_t)$ is the probability of transitioning to state s_{t+1} from state s_t by taking action a_t at step t; C is the cost distribution such that a cost c_t at step t is sampled from $C_t(\cdot|s_t, a_t)$. Finally, we denote $\bar{c}_t(s_t, a_t)$ as the expected cost, ρ_0 as the initial distribution of states, and $H \in \mathbb{N}^+$ as the finite horizon (max length) of the MDP.

We define a stochastic policy π such that for any state $s \in \mathcal{S}$, $\pi(\cdot|s) \in \Delta(A)$, where $\Delta(A)$ is a A-dimensional simplex, conditioned on state s. $\pi(a|s) \in [0, 1]$ outputs the probability of taking action a at state s. The distribution of trajectories $\tau = (s_1, a_1, \ldots, a_{H-1}, s_H)$ is determined by π and the MDP, and is defined as

$$\rho_\pi(\tau) = \rho_0(s_1) \prod_{t=2}^{H} \pi(a_{t-1}|s_{t-1}) P_{t-1}(s_t|s_{t-1}, a_{t-1}).$$

The distribution of the states at time step t, induced by running the policy π until t, is defined $\forall s_t$:

$$d_t^\pi(s_t) = \sum_{\{s_i, a_i\}_{i \leq t-1}} \rho_0(s_1) \prod_{i=1}^{t-1} \pi(a_i|s_i) P_i(s_{i+1}|s_i, a_i).$$

Note that the summation above can be replaced by an integral if the state or action space is continuous. The average state distribution $\bar{d}^\pi(s) = \sum_{t=1}^{H} d_t^\pi(s)/H$.

The expected average cost of a policy π can be defined with respect to ρ_π or $\{d_t^\pi\}$:

$$\mu(\pi) = \mathop{\mathbb{E}}_{\tau \sim \rho_\pi} \left[\sum_{t=1}^{H} \bar{c}_t(s_t, a_t) \right] = \sum_{t=1}^{H} \mathop{\mathbb{E}}_{s \sim d_t^\pi(s), a \sim \pi(a|s)} [\bar{c}_t(s, a)].$$

We define the state-action value $Q_t^\pi(s, a)$ (i.e., cost-to-go) for policy π at time step t as:

$$Q_t^\pi(s_t, a_t) \quad = \bar{c}_t(s_t, a_t) + \mathop{\mathbb{E}}_{s \sim P_t(\cdot|s_t, a_t), a \sim \pi(\cdot|s)} Q_{t+1}^\pi(s, a),$$

where the expectation is taken over the randomness of the policy π and the MDP.

We define π^* as the expert policy (e.g., human demonstrators, search algorithms equipped with ground-truth) and $Q_t^*(s, a)$ as the expert's cost-to-go oracle. We emphasize that π^* may not be optimal, i.e., $\pi^* \notin \arg\min_\pi \mu(\pi)$. Throughout the paper, we assume $Q_t^*(s, a)$ is known or can be estimated without bias (e.g., by rolling out π^*: starting from state s, applying action a, and then following π^* for $H - t$ steps).

When π is represented by a function approximator, we use the notation π_θ to represent the policy parametrized by $\theta \in \mathbb{R}^d$: $\pi(\cdot|s; \theta)$. In this work we specifically consider optimizing policies in which the parameter dimension d may be large. We also consider the partially observable setting in our experiments, where the policy $\pi(\cdot|o_1, a_1, ..., o_t; \theta)$ is defined over the whole history of observations and actions (o_t is generated from the hidden state s_t). We use both LSTM and Gated Recurrent Unit (GRU) (Chung et al., 2014) based policies where the RNN's hidden states provide a compressed feature of the history. To our best knowledge, this is the first time RNNs are employed in an IL framework to handle partially observable environments.

3. Differentiable Imitation Learning

Policy based imitation learning aims to learn a policy $\hat{\pi}$ that approaches the performance of the expert π^* at test time when π^* is no longer available. In order to learn rich policies such as LSTMs or deep networks (Schulman et al., 2015), we derive a method related to *policy gradients* for imitation learning and sequential prediction. To do this, we leverage the reduction of IL and sequential prediction to online learning as shown in (Ross & Bagnell, 2014) to learn policies represented by expressive differentiable function approximators.

The fundamental idea in Ross & Bagnell (2014) is to use a no-regret online learner to update policies using the following loss function at each episode n:

$$\ell_n(\pi) = \frac{1}{H} \sum_{t=1}^{H} \mathop{\mathbb{E}}_{s_t \sim d_t^{\pi_n}} \Big[\mathop{\mathbb{E}}_{a \sim \pi(\cdot|s_t)} [Q_t^*(s_t, a)] \Big]. \quad (1)$$

The loss function intuitively encourages the learner to find a policy that minimize the expert's cost-to-go *under the state distribution resulting from the current learned policy* π_n. Specifically, Ross & Bagnell (2014) suggest an algorithm named *AggreVaTe* (Aggregate Values to Imitate) that uses Follow-the-Leader (FTL) (Shalev-Shwartz et al., 2012) to update policies: $\pi_{n+1} = \arg\min_{\pi \in \Pi} \sum_{i=1}^{n} \ell_n(\pi)$, where Π is a pre-defined convex policy set. When $\ell_n(\pi)$ is strongly convex with respect to π and $\pi^* \in \Pi$, after N iterations AggreVaTe with FTL can find a policy $\hat{\pi}$ with:

$$\mu(\hat{\pi}) \leq \mu(\pi^*) - \epsilon_N + O(\ln(N)/N), \quad (2)$$

where $\epsilon_N = [\sum_{n=1}^{N} \ell_n(\pi^*) - \min_\pi \sum_{n=1}^{N} \ell_n(\pi)]/N$. Note that $\epsilon_N \geq 0$ and the above inequality indicates that $\hat{\pi}$ can outperform π^* when π^* is not (locally) optimal (i.e., $\epsilon_n > 0$). Our experimental results support this observation.

A simple implementation of AggreVaTe that aggregates the values (as the name suggests) will require an exact solution to a batch optimization procedure in each episode. When π is represented by large, non-linear function approximators, the $\arg\min$ procedure generally takes more and more computation time as n increases. Hence an efficient incremental update procedure is necessary for the method to scale.

To derive an incremental update procedure, we can take one of two routes. The first route, suggested already by (Ross & Bagnell, 2014), is to update our policy with an incremental no-regret algorithm such as weighted majority (Littlestone & Warmuth, 1994), instead of with a batch algorithm like FTRL. Unfortunately, for rich policy classes such as deep networks, no-regret learning algorithms may not be available (e.g., a deep network policy is non-convex with respect to its parameters). So instead we propose a novel second route: we directly differentiate Eq. 1, yielding an update related to policy gradient methods. We work out the details below, including a novel update rule for IL based on natural gradients.

Interestingly, the two routes described above yield almost identical algorithms if our policy class is simple enough: e.g., for a tabular policy, AggreVaTe with weighted majority yields the natural gradient version of AggreVaTeD described below. And, the two routes yield complementary theoretical guarantees: the first route yields a regret bound for simple-enough policy classes, while the second route yields convergence to a local optimum for extremely flexible policy classes.

3.1. Online Gradient Descent

For discrete actions, the gradient of $\ell_n(\pi_\theta)$ (Eq. 1) with respect to the parameters θ of the policy is

$$\nabla_\theta \ell_n(\theta) = \frac{1}{H}\sum_{t=1}^{H} \mathbb{E}_{s_t\sim d_t^{\pi_{\theta_n}}} \sum_a \nabla_\theta \pi(a|s_t;\theta) Q_t^*(s_t,a). \tag{3}$$

For continuous action spaces, we cannot simply replace the summation by integration since in practice it is hard to evaluate $Q_t^*(s,a)$ for infinitely many a, so, instead, we use importance weighting to re-formulate ℓ_n (Eq. 1) as

$$\begin{aligned}\ell_n(\pi_\theta) &= \frac{1}{H}\sum_{t=1}^{H} \mathbb{E}_{s\sim d_t^{\pi_{\theta_n}}, a\sim\pi(\cdot|s;\theta_n)} \frac{\pi(a|s;\theta)}{\pi(a|s;\theta_n)} Q_t^*(s,a)\\ &= \frac{1}{H}\mathbb{E}_{\tau\sim\rho_{\pi_{\theta_n}}} \sum_{t=1}^{H} \frac{\pi(a_t|s_t;\theta)}{\pi(a_t|s_t;\theta_n)} Q_t^*(s_t,a_t). \end{aligned}\tag{4}$$

See Appendix B for the derivation of the above equation. With this reformulation, the gradient with respect to θ is

$$\begin{aligned}\nabla_\theta \ell_n(\theta) &= \frac{1}{H}\mathbb{E}_{\tau\sim\rho_{\pi_{\theta_n}}} \sum_{t=1}^{H} \frac{\nabla_\theta\pi(a_t|s_t;\theta)}{\pi(a_t|s_t;\theta_n)} Q_t^*(s_t,a_t)\\ &= \frac{1}{H}\mathbb{E}_{\tau\sim\rho_{\pi_{\theta_n}}} \sum_{t=1}^{H} \nabla_\theta \ln(\pi(a_t|s_t;\theta_n)) Q_t^*(s_t,a_t). \end{aligned}\tag{5}$$

The above gradient computation enables a very efficient update procedure with online gradient descent: $\theta_{n+1} = \theta_n - \eta_n \nabla_\theta \ell_n(\theta)|_{\theta=\theta_n}$, where η_n is the learning rate.

3.2. Policy Updates with Natural Gradient Descent

We derive a natural gradient update procedure for imitation learning inspired by the success of natural gradient descent in RL (Kakade, 2002; Bagnell & Schneider, 2003; Schulman et al., 2015). Following (Bagnell & Schneider, 2003), we define the Fisher information matrix $I(\theta_n)$ using trajectory likelihood:

$$I(\theta_n) = \frac{1}{H^2}\mathbb{E}_{\tau\sim\rho_{\pi_{\theta_n}}} \nabla_{\theta_n}\log(\rho_{\pi_{\theta_n}}(\tau))\nabla_{\theta_n}\log(\rho_{\pi_{\theta_n}}(\tau))^T, \tag{6}$$

where $\nabla_\theta \log(\rho_{\pi_\tau}(\tau))$ is the gradient of the log likelihood of the trajectory τ which can be computed as $\sum_{t=1}^{H}\nabla_\theta\log(\pi_\theta(a_t|s_t))$. Note that this representation is equivalent to the original Fisher information matrix proposed by (Kakade, 2002). Now, we can use Fisher information matrix together with the IL gradient derived in the previous section (Eq. 53) to compute the natural gradient as $I(\theta_n)^{-1}\nabla_\theta\ell_n(\theta)|_{\theta=\theta_n}$, which yields a natural gradient update: $\theta_{n+1} = \theta_n - \mu_n I(\theta_n)^{-1}\nabla_\theta\ell_n(\theta)|_{\theta=\theta_n}$.

Interesting, as we mentioned before, when the given MDP is discrete and the policy class is in a tabular representation, AggreVaTe with Weighted Majority (Littlestone & Warmuth, 1994) yields an extremely similar update procedure as AggreVaTeD with natural gradient. Due to space limitation, we defer the detailed similarity between AggreVaTe with Weighted Majority and AggreVaTeD with natural gradient to Appendix A. As Weighted Majority can speed up online learning (i.e., almost dimension free (Bubeck et al., 2015)) and AggreVaTe with Weighted Majority enjoys strong theoretical guarantees on the performance of the learned policy (Ross & Bagnell, 2014), this similarity provides an intuitive explanation why we can expect AggreVaTeD with natural gradient to speed up IL and learn a high quality policy.

4. Sample-Based Practical Algorithms

In the previous section, we derived a regular gradient update procedure and a natural gradient update procedure for IL. Note that all of the computations of gradients and Fisher information matrices assumed it was possible to exactly compute expectations including $\mathbb{E}_{s\sim d^\pi}$ and $\mathbb{E}_{a\sim\pi(a|s)}$. In this section, we provide practical algorithms where we approximate the gradients and Fisher information matrices using finite samples collected during policy execution.

4.1. Gradient Estimation and Variance Reduction

We consider an episodic framework where given a policy π_n at episode n, we roll out π_n K times to collect K trajectories $\{\tau_i^n\}$, for $i\in[K]$, $\tau_i^n = \{s_1^{i,n}, a_1^{i,n}, ...\}$. For gradient $\nabla_\theta\ell_n(\theta)|_{\theta=\theta_n}$ we can compute an unbiased estimate using $\{\tau_i^n\}_{i\in[K]}$:

$$\tilde{\nabla}_{\theta_n} = \frac{1}{HK}\sum_{i=1}^{K}\sum_{t=1}^{H}\sum_{a}\nabla_{\theta_n}\pi_{\theta_n}(a|s_t^{i,n})Q_t^*(s_t^{i,n},a), \tag{7}$$

$$\tilde{\nabla}_{\theta_n} = \frac{1}{HK}\sum_{i=1}^{K}\sum_{t=1}^{H}\nabla_{\theta_n}\ln(\pi_{\theta_n}(a_t^{i,n}|s_t^{i,n}))Q_t^*(s_t^{i,n},a_t^{i,n}). \tag{8}$$

for discrete and continuous setting respectively.

When we can compute $V_t^*(s)$, we can replace $Q_t^*(s_t^{i,n},a)$ by the state-action advantage function $A_t^*(s_t^{i,n},a) = Q_t^*(s_t^{i,n},a) - V_t^*(s_t^{i,n})$, which leads to the following unbiased and variance-reduced gradient estimation for continuous action setting (Greensmith et al., 2004):

$$\tilde{\nabla}_{\theta_n} = \frac{1}{HK}\sum_{i=1}^{K}\sum_{t=1}^{H}\nabla_{\theta_n}\ln(\pi_{\theta_n}(a_t^{i,n}|s_t^{i,n}))A_t^*(s_t^{i,n},a_t^{i,n}), \tag{9}$$

In fact, we can use any baselines to reduce the variance by replacing $Q_t^*(s_t,a_t)$ by $Q_t^*(s_t,a_t) - b(s_t)$, where $b_(s_t) : \mathcal{S} \to \mathbb{R}$ is a action-independent function. Ideally $b(s_t)$ should be some function approximator that approximates $V^*(s_t)$. In our experiments, we test linear function approximator $b(s) = w^T s$, which is online learned using π^*'s roll-out data.

The Fisher information matrix (Eq. 19) is approximated as:

$$\tilde{I}(\theta_n) = \frac{1}{H^2K}\sum_{i=1}^{K}\nabla_{\theta_n}\log(\rho_{\pi_{\theta_n}}(\tau_i))\nabla_{\theta_n}\log(\rho_{\pi_{\theta_n}}(\tau_i))^T$$
$$= S_n S_n^T, \tag{10}$$

where, for notation simplicity, we denote S_n as a $d\times K$ matrix where the i's th column is $\nabla_{\theta_n}\log(\rho_{\pi_{\theta_n}}(\tau_i))/(H\sqrt{K})$. Namely the Fisher information matrix is represented by a sum of K rank-one matrices. For large policies represented by neural networks, $K \ll d$, and hence $\tilde{I}(\theta_n)$ a low rank matrix. One can find the descent direction δ_{θ_n} by solving the linear system $S_n S_n^T \delta_{\theta_n} = \tilde{\nabla}_{\theta_n}$ for δ_{θ_n} using Conjugate Gradient (CG) with a fixed number of iterations, which is equivalent to solving the above linear systems using Partial Least Squares (Phatak & de Hoog, 2002). This approach is used in TRPO (Schulman et al., 2015). The difference is that our representation of the Fisher matrix is in the form of $S_n S_n^T$ and in CG we never need to explicitly compute or store $S_n S_n^T$ which requires d^2 space and time. Instead, we only compute and store S_n ($O(Kd)$) and the total computational time is still $O(K^2 d)$. The learning-rate for natural gradient descent can be chosen as $\eta_n = \sqrt{\delta_{KL}/(\tilde{\nabla}_{\theta_n}^T \delta_{\theta_n})}$, such that $KL(\rho_{\pi_{\theta_{n+1}}(\tau)}\|\rho_{\pi_{\theta_n}(\tau)}) \approx \delta_{KL} \in \mathbb{R}^+$

Algorithm 1 AggreVaTeD (Differentiable AggreVaTe)

```
1: Input: The given MDP and expert $\pi^*$. Learning rate $\{\eta_n\}$. Schedule rate $\{\alpha_i\}$, $\alpha_n \to 0, n \to \infty$.
2: Initialize policy $\pi_{\theta_1}$ (either random or supervised learning).
3: for n = 1 to N do
4:     Mixing policies: $\hat{\pi}_n = \alpha_n\pi^* + (1-\alpha_n)\pi_{\theta_n}$.
5:     Starting from $\rho_0$, roll out by executing $\hat{\pi}_n$ on the given MDP to generate $K$ trajectories $\{\tau_i^n\}$.
6:     Using $Q^*$ and $\{\tau_i^n\}_i$, compute the descent direction $\delta_{\theta_n}$ (Eq. 7, Eq. 8, Eq. 9, or CG).
7:     Update: $\theta_{n+1} = \theta_n - \eta_n\delta_{\theta_n}$.
8: end for
9: Return: the best hypothesis $\hat{\pi} \in \{\pi_n\}_n$ on validation.
```

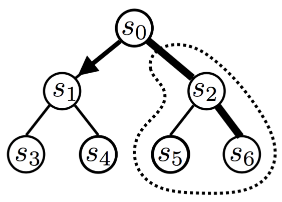


Figure 1. The binary tree structure MDP $\tilde{\mathcal{M}}$.

4.2. Differentiable Imitation Learning: AggreVaTeD

Summarizing the above discussion, we present the differentiable imitation learning framework *AggreVaTeD*, in Alg. 1. At every iteration n, the roll out policy $\hat{\pi}_n$ is a mix of the expert policy π^* and the current policy π_{θ_n}, with mixing rate α ($\alpha_n \to 0, n \to \infty$): at every step, with probability α $\hat{\pi}_n$ picks π^* and picks π_{θ_n} otherwise. This mixing strategy with the decay rate was first introduced in (Ross et al., 2011) for IL, and later on was used in sequence prediction (Bengio et al., 2015). In Line 6, one can either choose Eq. 8 or the corresponding variance reduced estimation Eq. 9 to perform regular gradient descent, and choose CG to perform natural gradient descent. AggreVaTeD is extremely simple: we do not need to perform any data aggregation (i.e., we do not need to store all $\{\tau_i\}_i$ from all previous iterations); the computational complexity of each policy update scales in $O(d)$.

When we use non-linear function approximators to represent the polices, the analysis of AggreVaTe from (Ross & Bagnell, 2014) will not hold, since the loss function $\ell_n(\theta)$ is not convex with respect to parameters θ. Nevertheless, as we will show in experiments, in practice AggreVaTeD is still able to learn a policy that is competitive with, and sometimes superior to, the oracle's performance.

5. Quantify the Gap: An Analysis of IL vs RL

How much faster can IL learn a good policy than RL? In this section we quantify the gap on discrete MDPs when IL can (1) query for an optimal Q^* or (2) query for a noisy but unbiased estimate of Q^*. To measure the speed of learning, we look at the *cumulative* regret of the entire learning process, defined as $R_N = \sum_{n=1}^{N}(\mu(\pi_n) - \mu(\pi^*))$. A smaller regret rate indicates faster learning. Throughout this section, we assume the expert π^* is optimal. We consider finite-horizon, episodic IL and RL algorithms.

5.1. Exponential Gap

We consider an MDP $\mathcal{M}$ shown in Fig. 1 which is a depth-K binary tree-structure with $S = 2^K - 1$ states and two actions a_l, a_r: go-left and go-right. The transition is deterministic and the initial state s_0 (root) is fixed. The cost for each non-leaf state is zero; the cost for each leaf is i.i.d sampled from a given distribution (possibly different distributions per leaf). Below we show that for $\mathcal{M}$, IL can be exponentially more sample efficient than RL.

Theorem 5.1. *For $\mathcal{M}$, the regret R_N of any finite-horizon, episodic RL algorithm is at least:*

$$\mathbb{E}[R_N] \geq \Omega(\sqrt{SN}). \tag{11}$$

The expectation is with respect to random generation of cost and internal randomness of the algorithm. However, for the same MDP $\mathcal{M}$, with the access to Q^*, we show IL can learn exponentially faster:

Theorem 5.2. *For the MDP $\mathcal{M}$, AggreVaTe with FTL can achieve the following regret bound:*

$$R_N \leq O(\ln(S)). \tag{12}$$

Fig. 1 illustrates the intuition behind the theorem. Assume during the first episode, the initial policy π_1 picks the rightmost trajectory (bold black) to explore. We query from the cost-to-go oracle Q^* at s_0 for a_l and a_r, and learn that $Q^*(s_0, a_l) < Q^*(s_0, a_r)$. This immediately tells us that the optimal policy will go left (black arrow) at s_0. Hence the algorithm does not have to explore the right sub-tree (dotted circle).

Next we consider a more difficult setting where one can only query for a noisy but unbiased estimate of Q^* (e.g., by rolling out π^* finite number of times). The above halving argument will not apply since deterministically eliminating nodes based on noisy estimates might permanently remove good trajectories. However, IL can still achieve a poly-log regret with respect to S, even in the noisy setting:

Theorem 5.3. *With only access to unbiased estimate of Q^*, for the MDP $\mathcal{M}$, AggreVaTeD with WM can achieve the following regret with probability at least $1-\delta$:*

$$R_N \leq O\Big(\ln(S)(\sqrt{\ln(S)N} + \sqrt{\ln(2/\delta)N})\Big). \tag{13}$$

The detailed proofs of the above three theorems can be found in Appendix E,F,G respectively. In summary, for MDP $\mathcal{M}$, IL is is exponentially faster than RL.

5.2. Polynomial Gap and Near-Optimality

We next quantify the gap in general discrete MDPs and also show that AggreVaTeD is near-optimal. We consider the harder case where we can only access an unbiased estimate of Q_t^*, for any t and state-action pair. The policy π is represented as a set of probability vectors $\pi^{s,t} \in \Delta(A)$, for all $s \in \mathcal{S}$ and $t \in [H]$: $\pi = \{\pi^{s,t}\}_{s\in\mathcal{S}, t\in[H]}$.

Theorem 5.4. *With access to unbiased estimates of Q_t^*, AggreVaTeD with WM achieves the regret upper bound:*

$$R_N \leq O\big(HQ^e_{\max}\sqrt{S\ln(A)N}\big). \tag{14}$$

Here $Q^e_{\max}$ is the maximum cost-to-go of the expert. The total regret shown in Eq. 14 allows us to compare IL algorithms to RL algorithms. For example, the Upper Confidence Bound (UCB) based, near-optimal optimistic RL algorithms from (Jaksch et al., 2010), specifically designed for efficient exploration, admit regret $\tilde{O}(HS\sqrt{HAN})$, leading to a gap of approximately $\sqrt{HAS}$ compared to the regret bound of imitation learning shown in Eq. 14.

We also provide a lower bound on R_N for the $H = 1$ case which shows the dependencies on N, A, S are tight:

Theorem 5.5. *There exists an MDP (H=1) such that, with only access to unbiased estimates of Q^*, any finite-horizon episodic imitation learning algorithm must have:*

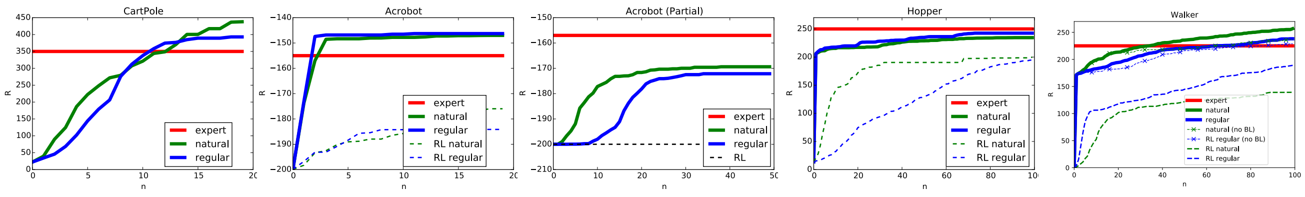
$$\mathbb{E}[R_N] \geq \Omega(\sqrt{S\ln(A)N}). \tag{15}$$

The proofs of the above two theorems regarding general MDPs can be found in Appendix H,I. In summary for discrete MDPs, one can expect at least a polynomial gap and a possible exponential gap between IL and RL.

6. Experiments

We evaluate our algorithms on robotics simulations from OpenAI Gym (Brockman et al., 2016) and on Handwritten Algebra Dependency Parsing (Duyck & Gordon, 2015). We report reward instead of cost, since OpenAI Gym by default uses reward and dependency parsing aims to maximize UAS score. As our approach only promises there

Here we assume $Q^e_{\max}$ is a constant compared to H. If $Q^e_{\max} = \Theta(H)$, then the expert is no better than a random policy of which the cost-to-go is around $\Theta(H)$.



(a) Cartpole (b) Acrobot (c) Acrobot (POMDP) (d) Hopper (e) Walker

Figure 2. Performance (cumulative reward R on y-axis) versus number of episodes (n on x-axis) of AggreVaTeD (blue and green), experts (red), and RL algorithms (dotted) on different robotics simulators.

exists a policy among all of the learned polices that can perform as well as the expert, we report the performance of the best policy so far: $\max\{\mu(\pi_1), ..., \mu(\pi_i)\}$. For regular gradient descent, we use ADAM (Kingma & Ba, 2014) which is a first-order no-regret algorithm, and for natural gradient, we use CG to compute the descent direction. For RL we use REINFORCE (Williams, 1992) and Truncated Natural Policy Gradient (TNPG) (Duan et al., 2016).

6.1. Robotics Simulations

We consider CartPole Balancing, Acrobot Swing-up, Hopper and Walker. For generating an expert, similar to previous work (Ho & Ermon, 2016), we used a Deep Q-Network (DQN) to generate Q^* for CartPole and Acrobot (e.g., to simulate the settings where Q^* is available), while using the publicly available TRPO implementation to generate π^* for Hopper and Walker to simulate the settings where one has to estimate Q^* by Monte-Carlo roll outs π^*.

Discrete Action Setting We use a one-layer (16 hidden units) neural network with ReLu activation functions to represent the policy π for the Cart-pole and Acrobot benchmarks. The value function Q^* is obtained from the DQN (Mnih et al., 2015) and represented by a multi-layer fully connected neural network. The policy π_{θ_1} is initialized with common ReLu neural network initialization techniques. For the scheduling rate $\{\alpha_i\}$, we set all $\alpha_i = 0$: namely we did not roll-in using the expert's actions during training. We set the number of roll outs $K = 50$ and horizon $H = 500$ for CartPole and $H = 200$ for Acrobot.

Fig. 2a and 2b shows the performance averaged over 10 random trials of AggreVaTeD with regular gradient descent and natural gradient descent. Note that AggreVaTeD outperforms the experts' performance significantly: Natural gradient surpasses the expert by 5.8% in Acrobot and **25**% in Cart-pole. Also, for Acrobot swing-up, at horizon $H = 200$, with high probability a randomly initialized neural network policy won't be able to collect any reward signals. Hence the improvement rates of REINFORCE and TNPG are slow. In fact, we observed that for a short horizon such as $H = 200$, REINFORCE and Truncated Natural Gradient often even fail to improve the policy at all (failed 6 times among 10 trials). On the contrary, AggreVaTeD does not suffer from the delayed reward signal issue, since the expert will collect reward signals much faster than a randomly initialized policy.

Fig. 2c shows the performance of AggreVaTeD with an LSTM policy (32 hidden states) in a partially observed setting where the expert has access to full states but the learner has access to partial observations (link positions). RL algorithms did not achieve any improvement while AggreVaTeD still achieved 92% of the expert's performance. In Appendix K, we provide extra experiments on partial observable CartPole with GRU-based policies, where we demonstrate that even in partial observable setting, AggreVaTeD can learn RNN polices that outperform experts.

Continuous Action Setting We test our approaches on two robotics simulators with continuous actions: (1) the 2-d Walker and (2) the Hopper from the MuJoCo physics simulator. Following the neural network settings described in Schulman et al. (2015), the expert policy π^* is obtained from TRPO with one hidden layer (64 hidden states), which is the same structure that we use to represent our policies π_θ. We set $K = 50$ and $H = 100$. We initialize π_{θ_1} by collecting K expert demonstrations and then maximize the likelihood of these demonstrations (i.e., supervised learning). We use a linear baseline $b(s) = w^T s$ for RL and IL.

Fig. 2e and 2d show the performance averaged over 5 random trials. Note that AggreVaTeD outperforms the expert in the Walker by 13.7% while achieving 97% of the expert's performance in the Hopper problem. After 100 iterations, we see that by leveraging the help from experts, AggreVaTeD can achieve much faster improvement rate than the corresponding RL algorithms (though eventually we can expect RL to catch up). In Walker, we also tested AggreVaTeD without linear baseline, which still outperforms the expert but performed slightly worse than AggreVaTeD with baseline as expected.

6.2. Dependency Parsing on Handwritten Algebra

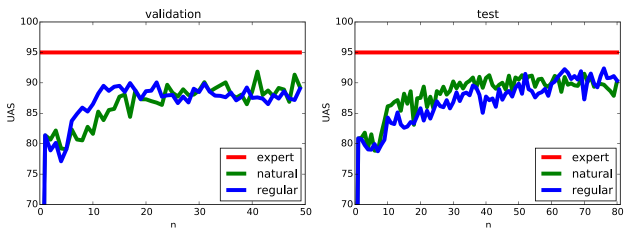
We consider a sequential prediction problem: transition-based dependency parsing for handwritten algebra with raw image data (Duyck & Gordon, 2015). The parsing task

Arc-Eager	AggreVaTeD (LSTMs)	AggreVaTeD (NN)	SL-RL (LSTMs)	SL-RL(NN)	RL (LSTMs)	RL (NN)	DAgger	SL (LSTMs)	SL (NN)	Random
Regular	**0.924**±0.10	0.851±0.10	0.826± 0.09	0.386±0.1	0.256±0.07	0.227±0.06	0.832±0.02	0.813±0.1	0.325±0.2	∼0.150
Natural	0.915±0.10	0.800±0.10	0.824±0.10	0.345±0.1	0.237±0.07	0.241±0.07				

Table 1. Performance (UAS) of different approaches on handwritten algebra dependency parsing. *SL* stands for supervised learning using expert's samples: maximizing the likelihood of expert's actions under the sequences generated by expert itself. *SL-RL* means RL with initialization using SL. *Random* stands for the initial performances of random policies (LSTMs and NN). The performance of DAgger with Kernel SVM is from (Duyck & Gordon, 2015).

for algebra is similar to the classic dependency parsing for natural language (Chang et al., 2015a) where the problem is modelled in the IL setting and the state-of-the-art is achieved by AggreVaTe with FTRL (using Data Aggregation). The additional challenge here is that the inputs are handwritten algebra symbols in raw images. We directly learn to predict parse trees from low level image features (Histogram of Gradient features (HoG)). During training, the expert is constructed using the ground-truth dependencies in training data. The full state s during parsing consists of three data structures: Stack, Buffer and Arcs, which store raw images of the algebraic symbols. Since the sizes of stack, buffer and arcs change during parsing, a common approach is to featurize the state s by taking the features of the latest three symbols from stack, buffer and arcs (e.g., (Chang et al., 2015a)). Hence the problem falls into the *partially observable* setting, where the feature o is extracted from state s and only contains partial information about s. The dataset consists of 400 sets of handwritten algebra equations. We use 80% for training, 10% for validation, and 10% for testing. We include an example of handwritten algebra equations and its dependency tree in Appendix J. Note that different from robotics simulators where at every episode one can get fresh data from the simulators, the dataset is fixed and sample efficiency is critical.

The RNN policy follows the design from (Sutskever et al., 2014). It consists of two LSTMs. Given a sequence of algebra symbols τ, the first LSTM processes one symbol at a time and at the end outputs its hidden states and memory (i.e., a summary of τ). The second LSTM initializes its own hidden states and memory using the outputs of the first LSTM. At every parsing step t, the second LSTM takes the current partial observation o_t (o_t consists of features of the most recent item from stack, buffer and arcs) as input, and uses its internal hidden state and memory to compute the action distribution $\pi(\cdot|o_1, ..., o_t, \tau)$ conditioned on history. We also tested reactive policies constructed as fully connected ReLu neural networks (NN) (one-layer with 1000 hidden states) that directly maps from observation o_t to action a, where o_t uses the most three recent items. We use variance reduced gradient estimations, which give better performance in practice. The performance is summarised in Table 1. Due to the partial observability of the problem, AggreVaTeD with a LSTM policy achieves significantly better UAS scores compared to the NN reactive policy and DAgger with a Kernelized SVM (Duyck & Gordon, 2015). Also AggreVaTeD with a LSTM policy achieves 97% of optimal expert's performance. Fig. 3 shows the improvement rate of regular gradient and natural gradient on both validation set and test set. Overall we observe that both methods have similar performance. Natural gradient achieves a better UAS score in validation and converges slightly faster on the test set but also achieves a lower UAS score on test set.



(a) Validation (b) Test

Figure 3. UAS (y-axis) versus number of iterations (n on x-axis) of AggreVaTeD with LSTM policy (blue and green), experts (red) on validation set and test set for Arc-Eager Parsing.

7. Conclusion

We introduced AggreVaTeD, a differentiable imitation learning algorithm which trains neural network policies for sequential prediction tasks such as continuous robot control and dependency parsing on raw image data. We showed that in theory and in practice IL can learn much faster than RL with access to optimal cost-to-go oracles. The IL learned policies were able to achieve expert and sometimes super-expert levels of performance in both fully observable and partially observable settings. The theoretical and experimental results suggest that IL is significantly more effective than RL for sequential prediction with near optimal cost-to-go oracles.

Acknowledgement

This research was supported in part by ONR 36060-1b-1141268.

References

Abbeel, Pieter and Ng, Andrew Y. Apprenticeship learning via inverse reinforcement learning. In *ICML*, pp. 1. ACM, 2004.

Bagnell, J Andrew and Schneider, Jeff. Covariant policy search.

IJCAI, 2003.

Bahdanau, Dzmitry, Brakel, Philemon, Xu, Kelvin, Goyal, Anirudh, Lowe, Ryan, Pineau, Joelle, Courville, Aaron, and Bengio, Yoshua. An actor-critic algorithm for sequence prediction. *arXiv preprint arXiv:1607.07086*, 2016.

Bengio, Samy, Vinyals, Oriol, Jaitly, Navdeep, and Shazeer, Noam. Scheduled sampling for sequence prediction with recurrent neural networks. In *NIPS*, 2015.

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

Bubeck, Sébastien, Cesa-Bianchi, Nicolo, et al. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends® in Machine Learning*, 2012.

Bubeck, Sébastien et al. Convex optimization: Algorithms and complexity. *Foundations and Trends® in Machine Learning*, 2015.

Chang, Kai-Wei, He, He, Daumé III, Hal, and Langford, John. Learning to search for dependencies. *arXiv preprint arXiv:1503.05615*, 2015a.

Chang, Kai-wei, Krishnamurthy, Akshay, Agarwal, Alekh, Daume, Hal, and Langford, John. Learning to search better than your teacher. In *ICML*, 2015b.

Choudhury, Sanjiban, Kapoor, Ashish, Ranade, Gireeja, Scherer, Sebastian, and Dey, Debadeepta. Adaptive information gathering via imitation learning. *RSS*, 2017.

Chung, Junyoung, Gulcehre, Caglar, Cho, KyungHyun, and Bengio, Yoshua. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Daumé III, Hal, Langford, John, and Marcu, Daniel. Search-based structured prediction. *Machine learning*, 2009.

Duan, Yan, Chen, Xi, Houthooft, Rein, Schulman, John, and Abbeel, Pieter. Benchmarking deep reinforcement learning for continuous control. In *ICML*, 2016.

Duyck, James A and Gordon, Geoffrey J. Predicting structure in handwritten algebra data from low level features. *Data Analysis Project Report, MLD, CMU*, 2015.

Finn, Chelsea, Levine, Sergey, and Abbeel, Pieter. Guided cost learning: Deep inverse optimal control via policy optimization. In *ICML*, 2016.

Greensmith, Evan, Bartlett, Peter L, and Baxter, Jonathan. Variance reduction techniques for gradient estimates in reinforcement learning. *JMLR*, 2004.

Ho, Jonathan and Ermon, Stefano. Generative adversarial imitation learning. In *NIPS*, 2016.

Hochreiter, Sepp and Schmidhuber, Jürgen. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Jaksch, Thomas, Ortner, Ronald, and Auer, Peter. Near-optimal regret bounds for reinforcement learning. *JMLR*, 2010.

Kahn, Gregory, Zhang, Tianhao, Levine, Sergey, and Abbeel, Pieter. Plato: Policy learning using adaptive trajectory optimization. *arXiv preprint arXiv:1603.00622*, 2016.

Kakade, Sham. A natural policy gradient. *NIPS*, 2002.

Kakade, Sham and Langford, John. Approximately optimal approximate reinforcement learning. In *ICML*, 2002.

Kingma, Diederik and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Li, Jiwei, Monroe, Will, Ritter, Alan, Galley, Michel, Gao, Jianfeng, and Jurafsky, Dan. Deep reinforcement learning for dialogue generation. *arXiv preprint arXiv:1606.01541*, 2016.

Littlestone, Nick and Warmuth, Manfred K. The weighted majority algorithm. *Information and computation*, 108(2):212–261, 1994.

Mnih, Volodymyr et al. Human-level control through deep reinforcement learning. *Nature*, 2015.

Phatak, Aloke and de Hoog, Frank. Exploiting the connection between pls, lanczos methods and conjugate gradients: alternative proofs of some properties of pls. *Journal of Chemometrics*, 2002.

Ranzato, Marc'Aurelio, Chopra, Sumit, Auli, Michael, and Zaremba, Wojciech. Sequence level training with recurrent neural networks. *ICLR 2016*, 2015.

Ratliff, Nathan D, Bagnell, J Andrew, and Zinkevich, Martin A. Maximum margin planning. In *ICML*, 2006.

Rhinehart, Nicholas, Zhou, Jiaji, Hebert, Martial, and Bagnell, J Andrew. Visual chunking: A list prediction framework for region-based object detection. In *ICRA*. IEEE, 2015.

Ross, Stéphane and Bagnell, J. Andrew. Efficient reductions for imitation learning. In *AISTATS*, pp. 661–668, 2010.

Ross, Stephane and Bagnell, J Andrew. Reinforcement and imitation learning via interactive no-regret learning. *arXiv preprint arXiv:1406.5979*, 2014.

Ross, Stéphane, Gordon, Geoffrey J, and Bagnell, J.Andrew. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, 2011.

Ross, Stephane, Zhou, Jiaji, Yue, Yisong, Dey, Debadeepta, and Bagnell, Drew. Learning policies for contextual submodular prediction. In *ICML*, 2013.

Schulman, John, Levine, Sergey, Abbeel, Pieter, Jordan, Michael I, and Moritz, Philipp. Trust region policy optimization. In *ICML*, pp. 1889–1897, 2015.

Shalev-Shwartz, Shai et al. Online learning and online convex optimization. *Foundations and Trends® in Machine Learning*, 2012.

Silver, David et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 2016.

Sutskever, Ilya, Vinyals, Oriol, and Le, Quoc V. Sequence to sequence learning with neural networks. In *NIPS*, 2014.

Syed, Umar, Bowling, Michael, and Schapire, Robert E. Apprenticeship learning using linear programming. In *ICML*, 2008.

Venkatraman, Arun, Hebert, Martial, and Bagnell, J Andrew. Improving multi-step prediction of learned time series models. *AAAI*, 2015.

Williams, Ronald J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 1992.

Ziebart, Brian D, Maas, Andrew L, Bagnell, J Andrew, and Dey, Anind K. Maximum entropy inverse reinforcement learning. In *AAAI*, 2008.

Zinkevich, Martin. Online Convex Programming and Generalized Infinitesimal Gradient Ascent. In *ICML*, 2003.