Predictive Hints in Optimistic Online Learning for Better Optimizers

Luca Mezger

Under the direction of Prof. Ashok Cutkosky and Qinzi Zhang
Department of Electrical & Computer Engineering
Boston University

Research Science Institute
September 7, 2024

Abstract

This research examines the integration of predictive hints in optimistic online learning to improve the efficiency of optimizers in machine learning applications. Utilizing the Online-to-Non-Convex Conversion (O2NC), we aim to improve the performance of optimizers, particularly for models with non-convex non-smooth loss functions. We analyze various techniques for generating predictive hints and evaluate their performance using a GPT-2 model. Our findings demonstrate that incorporating accurate future gradient predictions can drastically improve optimization outcomes, with some methods even surpassing the performance of the Adam optimizer. This highlights the potential of optimistic online learners in advancing optimization techniques for machine learning. Additionally, our work helps us to further understand the behaviour of optimizers.

Summary

Machine learning models often face challenges in efficiently processing data, especially when dealing with complex or large datasets. Traditional learning methods can be slow or ineffective, leading to suboptimal performance. We use a novel approach to improve these learning processes by incorporating predictions about the model's future behavior. By utilizing these predictions, we can potentially accelerate the learning process and enhance its accuracy. We tested various methods to generate these predictions using a large language model (GPT-2). Our results demonstrated that accurate predictions can improve the model's performance. This approach holds promise for improving the effectiveness of machine learning across various applications.

1 Introduction

The rapid growth of data availability and the increasing complexity of machine learning models highlight the need for more efficient optimization techniques. While not all fields have an abundance of data, machine learning tends to be applied more in data-rich areas. However, better optimization techniques are valuable for both small and large data sets. For small data sets, the primary challenge is overfitting, which happens when a model learns too much from the training data, including the noise. Overfitting leads to poor generalization on new data because the model captures random fluctuations rather than the underlying pattern. This overfitting can be mitigated by using better optimizers that require fewer gradient computations, thereby reducing the risk of learning noise. In contrast, for large data sets, the limitation is often computational power. Efficient optimization methods allow models to achieve lower loss values within the same computational budget by making the most out of the available gradient computations. Therefore, developing optimizers that shorten training durations and improve efficiency is essential for advancing machine learning models across various data environments.

Traditional methods like batch gradient descent and its variants can struggle to effectively train large-scale neural networks. While optimizers such as Adam (Kingma and Ba, 2014) have achieved practical success, a significant theoretical gap remains in understanding their behavior and performance. This gap can result in unexpected outcomes and inefficiencies in various machine learning applications (Reddi et al., 2019). Addressing this gap is essential for developing optimizers that are both theoretically robust and practically effective.

This paper uses a novel approach using the online-to-non-convex conversion (O2NC) proposed by Cutkosky et al. (2023). This method applies online learning principles to the optimization of neural networks, which allows for better theoretical analysis, especially of

non-convex, non-smooth objective functions, like loss functions of models incorporating ReLU or MaxPooling layers. By transforming complex optimization challenges into more manageable online learning tasks, variants of O2NC potentially enables better practical outcomes. The integration of O2NC with various online learning algorithms aims to achieve faster convergence and improved optimization performance. This research explores the optimistic approach, where future gradient predictions, known as hints, are utilized in the update step. We provide different methods for generating these prediction hints and evaluate their effectiveness on a GPT-2 model.

Our contributions include analyzing the applications of the O2NC framework for neural network optimization, with the use of optimistic gradient predictions in the optimization process, and evaluating the effectiveness of different prediction hint generation methods on a GPT-2 model.

The paper is organized as follows. Section 2 gives the required background, followed by Section 3 detailing optimism in online learning. Section 4 presents the results. Finally, Section 5 concludes the paper and discusses future research directions.

2 Preliminaries

2.1 Optimization in Machine Learning

Formally, stochastic optimization is the process of minimizing an expected objective function, $F(x) = \mathbb{E}_{z\sim D_z}[f(x, z)]$, by finding

$$x^* = \arg\min_{x\in\mathcal{X}} F(x)$$

where z represents random variables that follows the distribution D_z introducing uncertainty. In the machine learning context, f is called the loss function, x represents the

parameters of the model, and z is a data sample.

Optimizers are algorithms used to change the parameters of machine learning models, such as weights and biases, to reduce the losses (i.e., minimize objective function). The optimization algorithm plays the central role in training machine learning models. The most common optimizers include Stochastic Gradient Descent (SGD) and variants like SGD with Momentum, AdaGrad (Duchi et al., 2011), RMSProp (Hinton, 2012), and Adam (Kingma and Ba, 2014). We review some classical and popular optimizers in Appendix A.

SGD updates weights based solely on the gradient, SGDM adds a momentum term to accelerate convergence, and Adam combines adaptive learning rates with momentum. As shown in Figures 1 and 2, these optimizers have distinct behaviors in their convergence patterns and speed. It is evident that Adam outperforms SGDM, and SGDM is significantly better than SGD on training a GPT-2 model, as described in section 3.3.

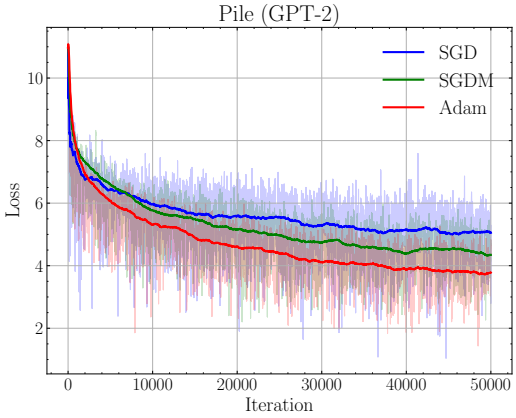

Figure 1: Cross Entropy Loss function (smoothed with time-weighted exponential moving average (EMA)) with $\eta = 0.1$ for SGD, $\eta = 1$ and $\beta = 0.9$ for SGDM, $\eta = 3 \times 10^{-4}$, $\beta_1 = 0.9$, and $\beta_2 = 0.999$ for Adam.

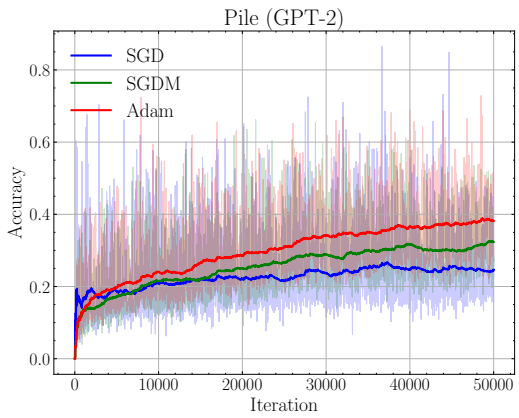

Figure 2: Accuracy (smoothed with time-weighted exponential moving average (EMA)2) with $\eta = 0.1$ for SGD, $\eta = 1$ and $\beta = 0.9$ for SGDM, $\eta = 3 \times 10^{-4}$, $\beta_1 = 0.9$, and $\beta_2 = 0.999$ for Adam.

2.2 Online Learning

Online learning is a framework for making sequential predictions under potentially adversarial conditions. In offline learning, actions can be based on the full training data set. In online learning, actions are based only on the information available up to the current time step. To gain some intuition consider a game in which, during each round t:

1. An adversary chooses a real number y_t in a given set.
2. The learner tries to guess the real number y_t by predicting $\hat{y}_t$.
3. The adversary reveals y_t, and the adversary picks an arbitrary loss which the learner incurs. For example, $\ell_t(\hat{y}_t) = (\hat{y}_t - y_t)^2$.

The learner's goal is to choose actions x_t such that the next loss $\ell_t(x_t)$ is as small as possible. We can formalize this as follows: The goal is to minimize regret $\text{Regret}_T(u)$, which is defined as the difference between the cumulative loss of the learner and the cumulative loss of the best-fixed decision in hindsight:

$$\text{Regret}_T(u) = \sum_{t=1}^{T} \ell_t(x_t) - \sum_{t=1}^{T} \ell_t(u)$$

where u represents the benchmark decision (i.e., the best decision in hindsight $u = \arg\min \sum_{t=1}^{T} \ell_t(u)$). Regret is used because it provides a way to measure the performance of the online learning algorithm relative to the best possible fixed strategy. It is particularly useful because, in an adversarial setting, it is impossible to guarantee minimization of the actual loss due to the unpredictable nature of the adversary. By focusing on minimizing regret, the learner ensures that its performance is nearly as good as the best-fixed action in hindsight, even under worst-case scenarios (Orabona, 2019). The concept of regret was first introduced by Hannan (1957) in the context of zero-sum repeated games.

Instead of using only one best decision in hindsight to calculate the regret, we can use multiple decisions, which is called dynamic regret. Dynamic regret is defined as the regret of an online learning algorithm when compared to a sequence of best decisions in hindsight rather than a single best decision. Formally, if $u_1, u_2, \ldots, u_T$ represents the sequence of decisions at each time step t, the dynamic regret DynamicRegret_T is given by:

$$\text{DynamicRegret}_T(u_1, \ldots, u_T) = \sum_{t=1}^{T} \ell_t(x_t) - \sum_{t=1}^{T} \ell_t(u_t)$$

The goal is to ensure that DynamicRegret_T grows sublinearly with T so that the average regret per round $\frac{\text{DynamicRegret}_T}{T}$ approaches zero as T increases. Achieving sublinear dynamic regret with vectors $u_t = \text{argmin}_x f_t(x)$ is often unattainable due to the requirement for perfect foresight and the excessively high benchmark these optimal decisions set. Therefore, we just assume $u_1, \ldots, u_T$ to be a more practical benchmark, not necessarily the best decisions in hindsight.

2.3 Minimizing Regret in Online Learning

In online learning, our goal is to minimize regret by using different algorithms. For example, in the number guessing game, a simple strategy called Follow-the-Leader (FTL) can be used, where the prediction for each round is the action that minimizes the total loss observed so far. Formally, the FTL prediction at round t is given by:

$$x_t = \arg\min_{x \in V} \sum_{i=1}^{t} \ell_i(x)$$

However, in the worst-case scenario, FTL performs very poorly. Hence, we cannot guarantee any regret bound. An example and a solution to the problem are given in Appendix B.

2.4 Online Learners in Machine Learning

Online learning can be used in machine learning as an incremental training approach where a model is updated iteratively with each new data point rather than processing the entire dataset simultaneously.

Formally, given a sequence of data points z_t for $t = 1, 2, \ldots$, the model parameters x_t are updated at each step t according to the rule:

$$x_{t+1} = \Pi_V(x_t - \eta_t \nabla \ell_t(x))$$

where η_t denotes the learning rate, $\nabla \ell_t(x)$ is the gradient of the loss function ℓ with respect to x_t, and Π_V denotes the projection onto the feasible set V (in future equations we will omit the projection Π_V). The feasible set V consists of all points x that satisfy the problem's constraints, such as $x \geq 0$ and $x \leq 10$ (Zinkevich, 2003). This fundamental algorithm in this framework is Online Gradient Descent (OGD), a counterpart to SGD, which updates the model parameters by taking a step in the direction of the negative gradient of the loss function.

A common extension to the OGD online learner is Online Mirror Descent (OMD), which we discuss in Appendix D.

The online learners described above are suited for applications where data arrives in a stream. These methods allow the model to adapt continuously to new data, making them well-suited for applications where data arrives sequentially, such as online recommendation systems, real-time financial predictions, and adaptive spam filtering (Cesa-Bianchi and Lugosi, 2006; Hazan et al., 2016; Orabona, 2019; Zinkevich, 2003).

2.5 Online-to-Non-Convex Conversion (O2NC)

Instead of using online learners to update the parameters directly, we will use a technique proposed by Cutkosky et al. (2023), called Online-to-Non-Convex Conversion (O2NC). O2NC transforms the minimization of a non-convex and non-smooth function into the problem of minimizing the dynamic regret. This approach allows online learning algorithms to handle non-convex optimization problems. In this context, an optimization algorithm updates a previous iterate x_{t-1} by moving in a direction Δ_t: $x_t = x_{t-1} + \Delta_t$. Instead of a standard update rule, an online learning algorithm determines Δ_t using linear losses $\ell_t(x) = \langle g_t, x \rangle$, where $g_t = \nabla f(x_t, z_t)$.

The key idea is to ensure that the first-order Taylor approximation $F(x_{t-1} + \Delta_t) - F(x_{t-1}) \approx \langle g_t, \Delta_t \rangle$ is accurate, without appeal to complex arguments involving higher-order derivatives. To do so, we define g_t as the gradient evaluated at a random point along the line segment connecting x_{t-1} and $x_{t-1} + \Delta_t$:

$$g_t \approx \nabla F(x_{t-1} + s_t \Delta_t)$$

where s_t is uniform on [0,1]. The fundamental theorem of calculus then ensures that $F(x_{t-1} + \Delta_t) - F(x_{t-1}) = \mathbb{E}[\langle g_t, \Delta_t \rangle]$

To make the most effective step, we would like to minimize the gap $F(x_t) - F(x_{t-1})$, which is equivalent to minimizing the inner product $\langle g_t, \Delta_t \rangle$. That is, we want Δ_t to be as close to the next negative gradient $-g_t$ as possible. Be aware that we do not know g_t prior to predicting Δ_t.

By defining a loss function $l_t(\Delta) = \langle g_t, \Delta \rangle$, we can rephrase this as minimizing the regret:

$$\text{Regret}_T(u) := \sum_{t=1}^{T} \langle g_t, \Delta_t - u \rangle$$

An example demonstrating how O2NC can be used to replicate the behavior of well-known optimizers, such as Adam, is provided in Appendix E.

3 Optimism in Online Learning

Optimism in online learning is a powerful concept that could improve the performance of learning algorithms by incorporating predictions or "hints" about future gradients. Different methods to compute the "hints" will be discussed in Section 3.5. This approach enables the algorithm to make more informed updates, thereby possibly improving key performance metrics such as the loss.

3.1 Optimistic Online Gradient Descent

A well-known application of optimism in online learning is the Optimistic Online Gradient Descent algorithm (often referred to as OGD; we will use OOGD to avoid any confusion with Online Gradient Descent). In OOGD, the update rule incorporates a hint h_t regarding the future gradient:

$$x_{t+1} = x_t - \eta_t(g_t + h_t - h_{t-1})$$

where η_t is the learning rate, and $g_t = \nabla \ell_t(x_t)$ represents the gradient of the loss function at x_t. The hint h_t serves as an estimate of the gradient at the next time step. If h_t is accurate in the sense that h_t approximates g_{t+1}, the regret bound can be significantly lower. Therefore, we might expect better performance in practice.

It's important to note that this update rule depends on how you interpret OOGD. From the Online Mirror Descent (OMD) perspective, the given formula is correct. However, from the Follow-the-Regularized-Leader (FTRL) perspective, the update can also be derived

from the FTRL update:

$$x_{t+1} = x_0 - \eta_t \left(\sum_{i=0}^{t} g_i + h_t \right)$$

By setting $h_0 = 0$, and considering the sum of the previous gradients and hints, this update is equivalent to:

$$x_{t+1} = x_t - \eta_t g_t + \eta_{t-1} h_{t-1} - \eta_t h_t$$

Both updates are correct; they simply reflect different perspectives (OMD vs. FTRL) on how to interpret the optimistic update in online learning.

In the ideal case where $h_t = g_{t+1}$, the update $x_{t+1} = x_t - \eta_t(g_t + g_{t+1})$ is highly informed, potentially leading to better performance. A generalized extension to OOGD, similar to OMD, is discussed in Appendix G (Orabona, 2019).

3.2 Theoretical Analysis of Optimistic Learners

To understand the performance of optimistic algorithms, we analyze the regret bounds. The upper bound for the regret quantifies the maximum excess loss an online algorithm can incur relative to the benchmark decision, considering all conceivable scenarios. This metric is essential for evaluating the algorithm's robustness. The regret bound of the OGD algorithm is given by:

$$R_T = \sum_{t=1}^{T} (\ell_t(x_t) - \ell_t(u)) \leq \frac{1}{2\eta} \|x_1 - u\|^2 + \frac{\eta}{2} \sum_{t=1}^{T} \|\nabla \ell_t(x_t)\|^2$$

A detailed mathematical derivation of the regret bound is provided in the Appendix H. The regret bound provides insight into how the choice of learning rate η, initial distance to the optimal weight $\|x_1 - u\|$, and the cumulative sum of squared gradients affect the

regret bound.

The regret bound for OOGD can be derived similar to the regret bound of OGD. The regret bound is given by:

$$R_T \leq \frac{1}{2\eta}\|x_1 - u\|^2 + \frac{\eta}{2}\sum_{t=1}^{T}\|\nabla\ell_t(x_t) - h_t\|^2$$

If the hint h_t approximates the next gradient $\nabla\ell_t(x_{t+1})$ accurately, the algorithm can achieve lower regret. Specifically, when $h_t \approx \nabla\ell_t(x_t)$, the term $\nabla\ell_t(x_t) - h_t$ becomes smaller. Therefore, the regret bound gets smaller.

Incorporating accurate hints about future gradients significantly reduces the regret bound and therefore could have better performance in practice.

3.2.1 Smaller Bound for $F(x_M)$

In Theorem 7 by Cutkosky et al. (2023), which provides an upper bound for the last loss:

$$E[F(x_M)] = F(x_0) + E\left[\sum_{n=1}^{M}\langle g_n, \Delta_n - u_n\rangle\right] + E\left[\sum_{n=1}^{M}\langle g_n, u_n\rangle\right],$$

the second term, $E\left[\sum_{n=1}^{M}\langle g_n, \Delta_n - u_n\rangle\right]$, represents the regret of the algorithm. That means a lower regret bound implies that this term is smaller, which directly translates to a lower bound for $F(x_M)$.

3.3 Evaluation Method

We employed a GPT-2 model to perform next token prediction (Radford et al., 2019) using the cross-entropy loss. The experiments were run using a setup where the model predicted the next word in a sequence of text. The training and evaluation were con-

Algorithm 1 Optimistic Follow-the-Regularized-Leader Algorithm (Orabona, 2019)

Require: A sequence of regularizers $\psi_1, \ldots, \psi_T : X \to \mathbb{R}$ (e.g., $\psi(x) = \frac{1}{2}\|x\|^2$), closed non-empty convex set $V \subseteq X \subseteq \mathbb{R}^d$

1: **for** $t = 1$ to T **do**
2: Predict next loss $\tilde{\ell}_t : V \to \mathbb{R}$
3: Output $x_t \in \arg\min_{x \in V} \psi_t(x) + \tilde{\ell}_t(x) + \sum_{i=1}^{t-1} \ell_i(x)$
4: Receive $\ell_t : V \to \mathbb{R}$ and pay $\ell_t(x_t)$
5: **end for**

ducted on the Pile dataset, which provided a comprehensive benchmark for testing the performance of various optimizers and hint generation methods. This setup allowed us to measure improvements in prediction accuracy and model efficiency when incorporating optimistic online learning techniques. The benchmark optimizer we used, unless specified differently, was Adam with $\eta = 0.0003$, $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

3.4 Proof of Concept for Optimistic Online Learning

To demonstrate the efficacy of optimism in online learning, we conducted a proof-of-concept experiment using an adapted version of the Optimistic Follow-the-Regularized-Leader (OFTRL) algorithm described in Algorithm 1 (Syrgkanis et al., 2015). We combined this algorithm with the discounting method discussed in Appendix E. We go into greater detail and different modifications to OFTRL in Appendix F. In our experiment, we employed a controlled "cheating" approach to validate that optimism can indeed enhance performance when accurate hints are provided. In order to get more accurate hints we use a second gradient evaluation at a future point.

In order to do this, a different update rule, as described in Figure 3, for the parameters was used. The initial state of the model parameters was saved in each iteration. The gradient was then computed at the current point x_t. Next, a step was taken using the normal FTRL algorithm without optimism (i.e., OFTRL with $h_t = 0$). After this, the gradient $\tilde{g}_t$ at the

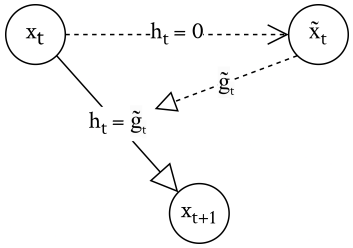

Figure 3: Where x_t is the initial point, $\tilde{x}_t$ is the point reached without optimism, and x_{t+1} is the updated point using the hint $\tilde{g}_t$.

new point was computed using a different batch or sample to ensure independence (i.e., reduce bias in a stochastic setting). The initial point was then reverted to, and a step was taken with the hint $h_t = \tilde{g}_t$. Be aware that $\tilde{g}_t \neq g_{t+1}$. This process was repeated for each iteration.

While this method is not practical for production use due to requiring twice as many gradient evaluations, it serves as a proof of concept. It demonstrates that if the hints closely approximate the future gradients, according to figures 4 and 5, improved results can be achieved.

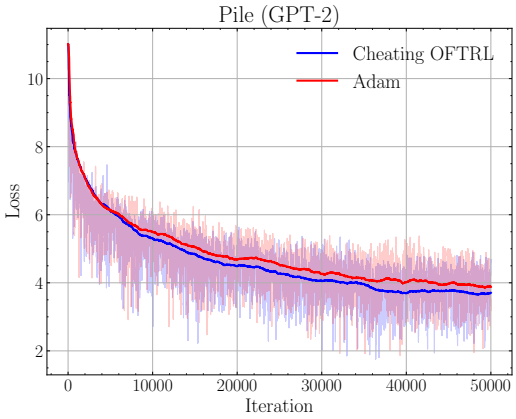

Figure 4: Loss function (smoothed with time-weighted EMA) with $\eta = 3 \times 10^{-4}$, $\beta_1 = 0.9$, and $\beta_2 = 0.999$ for both Adam and Cheating OFTRL

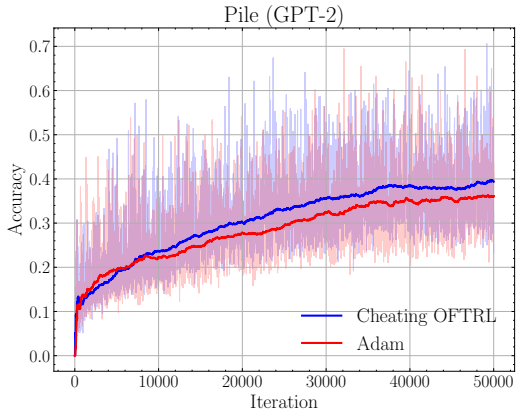

Figure 5: Accuracy (smoothed with time-weighted EMA) with $\eta = 3 \times 10^{-4}$, $\beta_1 = 0.9$, and $\beta_2 = 0.999$ for both Adam and Cheating OFTRL

3.5 Generating Hints

Hint prediction involves updating the value of a hint variable, h_{t+1}, using various methods that incorporate, for example, the past gradient g_t and the previous hints $h_1 \dots h_t$. These methods range from simple updates, like $h_{t+1} = g_t$, to approaches such as moving averages or momentum-based techniques. The goal is to set h_{t+1} as close to the next gradient g_{t+1} as possible. This can be achieved by balancing past and present information, often using a weighting parameter β. Different methods and their performance are proposed in Table 1. To ensure that these hints closely approximate the actual gradient, it is necessary to assume a certain level of smoothness on the loss function. If the gradient of the loss function changes too rapidly, the hints may mislead us, pointing us toward incorrect directions.

Formula	**Hyperparameter**	**EMA Loss**
$h_{t+1} = 0$ (Adam)	$\eta = 0.0003$	3.89
$h_{t+1} = g_t$	$\eta = 0.0003$	3.90
$h_{t+1} = \beta h_t + (1-\beta) g_t$	$\eta = 0.0003$, $\beta = 0.5$	3.96
$h_{t+1} = h_t + (1-\beta)(g_t - h_t)$	$\eta = 0.0003$, $\beta = 0.8$	3.94
$h_{t+1} = g_t + \beta(h_t - g_t)$	$\eta = 0.0003$, $\beta = 0.8$	3.98
$h_{t+1} = \beta h_t + \beta g_t$	$\eta = 0.0003$, $\beta = 0.5$	3.93
$h_{t+1} = \frac{t}{t+1} h_t + \frac{1}{t+1} g_t$	$\eta = 0.0003$	4.08
$h_{t+1} = \frac{\sqrt{h_t^2 + g_t^2}}{\sqrt{2}}$	$\eta = 0.0001$, $\beta = 0.9$	4.72
$h_{t+1} = \beta \Delta_t + (1-\beta) g_t$	$\eta = 0.0003$, $\beta = 0.8$	3.88

Table 1: Hint update methods, their formulas, hyperparameters, and time-weighted EMA of the train loss at the 50,000th iteration (same computational budget).

4 Results

The experiments conducted provided valuable insights into the effectiveness of different hint calculation methods in the context of optimistic online learning, as shown in Table

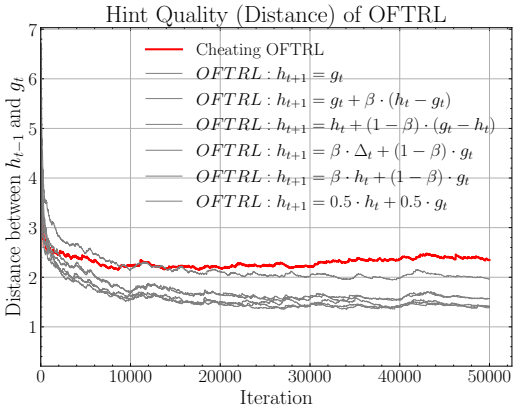

Figure 6: Distance $\|h_{t-1} - g_t\|$ of OFTRL according to table 1 (smoothed with time-weighted EMA).

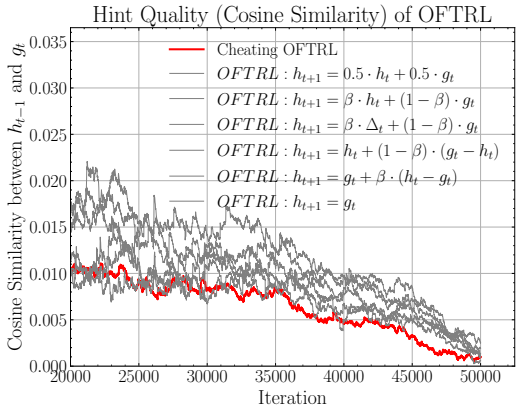

Figure 7: Cosine Similarity $\frac{h_{t-1} \cdot g_t}{\|h_{t-1}\| \|g_t\|}$ of OFTRL according to table 1 (smoothed with time-weighted EMA).

1. Some hints came close or even beat Adam by a marginal difference.. This shows that optimistic online learners have potential. Notably, the "cheating" version of OFTRL, which directly uses future gradients as hints, showed significantly better performance compared to methods where $h_t = 0$ or other hint calculation techniques. This superior performance is evident from the lower loss and higher accuracy metrics observed in Figures 4 and 5.

Despite the superior performance of the cheating version, the Euclidean distance between h_t and g_{t+1} was larger, and the cosine similarity was lower for this version. This indicates that while the cheating hints are effective, they are not necessarily close in direction or magnitude to the actual future gradients. Figures 6 and 7 illustrate these observations, showing higher distances and lower cosine similarities for the cheating method. These results seem very counterintuitive, but a reason can be found in the use of only one batch per iteration. Using a single batch for each update introduces significant stochasticity. Therefore, using a batch size of 1, $h_t = \tilde{g}_t$ does not accurately capture the real gradient g_{t+1}

To mitigate this, gradient accumulation, $g_{accum} = \frac{1}{N} \sum_{i=1}^{N} g_i$, where N is the number

of batches, was employed, increasing the number of batches per update step. Only g_{accum} was used in the optimization step. This approach reduced noise, resulting in more accurate hints and lower bias. The improvements are reflected in the lower Euclidean distance the last hint and the gradient, as shown in Figure 8. This shows us the dominance of the "cheating" OFTRL in predicting g_{t+1} accurately, especially when using larger batch sizes. The quality of gradient hints improves with increasing batch sizes, showing the distinction between smaller and larger batches in terms of reducing stochasticity. This is evidenced by Figures 9 and 10. The initial inversion of distances in Figure 9 is due to the higher absolute values of h_{t-1} and g_t. After 25,000 iterations, the distances reflect the correct relationship. Depending on the model, the use of bigger batch sizes could allow for big improvements in performance. Bigger batch sizes are especially useful because in practical applications batch sizes of 128 to 1024 are common.

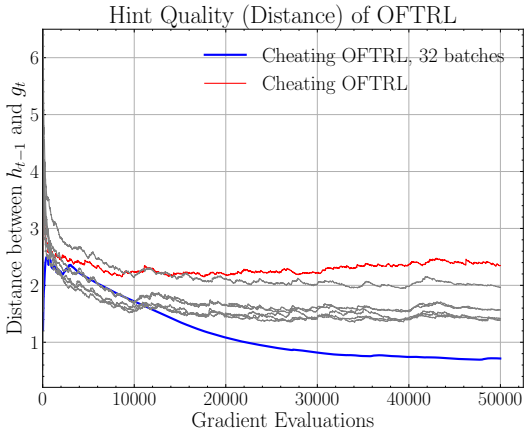

Figure 8: Distance $\|h_{t-1} - g_t\|$ of OFTRL. Each line represents a different hint calculation method, according to Table 1, using a single batch, except for the method that uses 32 batches (smoothed with time-weighted EMA).

In general, a positive correlation was observed between the Euclidean distance from h_t to g_{t+1} and the performance of each run. Runs with lower distances generally exhibited better performance, as depicted in Figures 11 and 12. This suggests that lower Euclidean

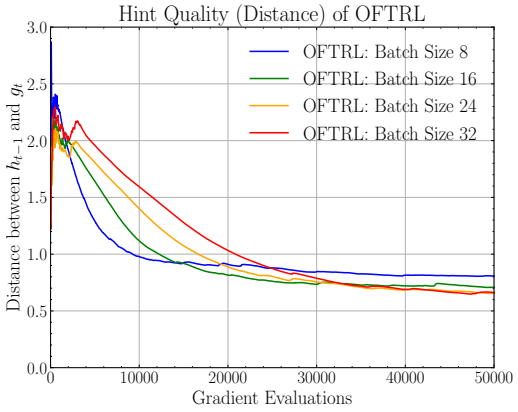

Figure 9: Distance $\|h_{t-1} - g_t\|$ of OFTRL.

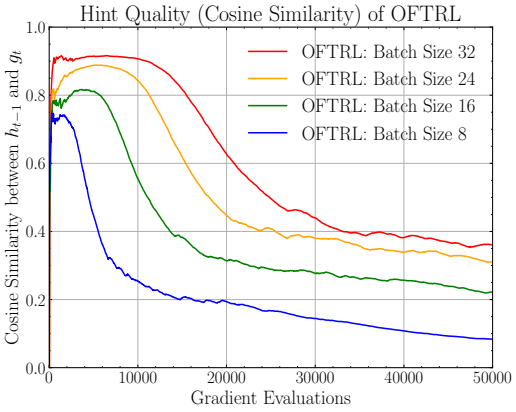

Figure 10: Cosine Similarity $\frac{h_{t-1} \cdot g_t}{\|h_{t-1}\| \|g_t\|}$ of

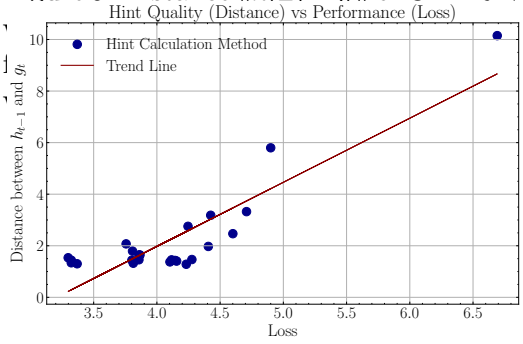

Figure 11: Distance $\|h_{t-1} - g_t\|$ of OFTRL compared to the loss (data points are the last value of the function smoothed with time-weighted EMA).

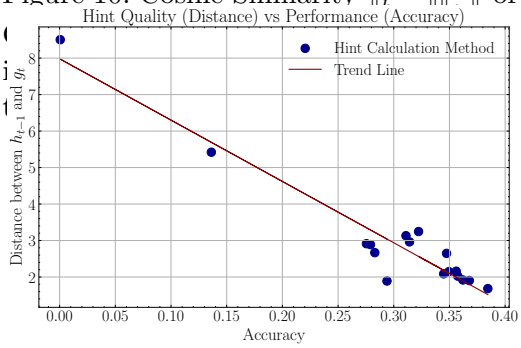

Figure 12: Distance $\|h_{t-1} - g_t\|$ of OFTRL compared to the accuracy (data points are the last value of the function smoothed with time-weighted EMA).

distance tends to be associated with better algorithm performance. However, the distance alone may not fully capture the hint quality, as a hint pointing in the correct direction but with a different magnitude might have a larger distance than a hint pointing in the opposite direction with a small magnitude.

To further analyze the hint quality, the cosine similarity can be used. In high-dimensional spaces, vectors tend to be orthogonal due to minimal projection. This phenomenon was observed in the hints, with most hints being nearly orthogonal to g_{t+1}. However, all hint

calculation methods had a positive cosine similarity, indicating that they still carry useful information about the direction of the future gradient, as shown in Figure 7. The cosine similarity provided a measure of the directional alignment between the hint and the actual future gradient.

5 Discussion

In conclusion, the results are very promising and highlight the potential of accurate future gradient predictions in online learning algorithms. We showed that the hint quality correlates with the performance of the optimistic algorithm. Some hints even improved the performance of the Adam optimizer. To draw definitive conclusions, further experiments on various datasets are necessary. The cheating version significantly improves performance metrics, underscoring the potential of this approach. However, practical methods for hint calculation need further refinement to achieve similar benefits without having to do two gradient evaluations per step. Future work could explore even more effective hint generation methods and their applicability across different machine learning tasks, such as computer vision or natural language processing. Additionally, the use of lower-precision numbers for hint calculations, such as float8, could offer a promising approach to reduce computational costs while maintaining performance. For instance, the cheating version achieves the same loss as Adam after roughly 36,000 iterations compared to 50,000, and the same accuracy after 33,000 iterations compared to 50,000. If the cheating version uses two gradient evaluations per iteration, it should achieve the same performance after only 25,000 iterations in order to have the same practicality as Adam. By reducing the computational cost with float8, we could use more iterations for the same computational budget, thereby improving performance. Furthermore, the use of larger batch sizes is a very promising approach to improve optimistic optimizers.

6 Key Takeaways

This research shows that using predictive hints in optimistic online learning can significantly improve the performance of machine learning optimizers. The technique can outperform traditional methods like Adam, making optimization faster and more effective. These advancements can enhance machine learning applications across various industries, including healthcare and finance. Supporting further development of these methods could lead to more efficient and robust AI solutions.

7 Acknowledgments

I would like to express my gratitude to my mentors, Professor Ashok Cutkosky and Qinzi Zhang, for their guidance. I am also thankful to my tutor, Shuvom Sadhuka for his advice. Additionally, I am grateful for my teaching assistants Victor Kolev, River Grace, Jenny Sendova, and Canaan He. I acknowledge the Massachusetts Institute of Technology, Boston University and the Center of Excellence in Education for making this program possible. My appreciation also goes to my fellow Rickoids of 2024. I am grateful to the FBK Bern, the René & Susanne Braginsky Stiftung, and the Fritz-Gerber-Stiftung for organizing and financing my participation in the Research Science Institute 2024. Lastly, I thank my parents and my brother for their continuous support throughout this program.

References

Kwangjun Ahn, Zhiyu Zhang, Yunbum Kook, and Yan Dai. Understanding adam optimizer via online learning of updates: Adam is ftrl in disguise. *arXiv preprint arXiv:2402.01567*, 2024.

Léon Bottou et al. Stochastic gradient learning in neural networks. *Proceedings of Neuro-Nımes*, 91(8):12, 1991.

Nicolo Cesa-Bianchi and Gábor Lugosi. *Prediction, learning, and games*. Cambridge university press, 2006.

Ashok Cutkosky, Harsh Mehta, and Francesco Orabona. Optimal stochastic non-smooth non-convex optimization through online-to-non-convex conversion. In *International Conference on Machine Learning*, pages 6643–6670. PMLR, 2023.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

James Hannan. Approximation to bayes risk in repeated play. *Contributions to the Theory of Games*, 3(2):97–139, 1957.

Elad Hazan et al. Introduction to online convex optimization. *Foundations and Trends® in Optimization*, 2(3-4):157–325, 2016.

Geoffrey Hinton. Lecture 6e rmsprop: Divide the gradient by a running average of its recent magnitude. Coursera, 2012. URL `https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`. Video.

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

Tengyu Ma, Maxwell Allman, and Faidra Monachou. Statistical learning theory, lecture 15. `https://web.stanford.edu/class/cs229t/scribe_notes/11_12_final.pdf`, 2018. CS229T/STATS231, Stanford University.

Francesco Orabona. A modern introduction to online learning. *arXiv preprint arXiv:1912.13213*, 2019.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019.

Vasilis Syrgkanis, Alekh Agarwal, Haipeng Luo, and Robert E Schapire. Fast convergence of regularized learning in games. *Advances in Neural Information Processing Systems*, 28, 2015.

Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the 20th international conference on machine learning (icml-03)*, pages 928–936, 2003.

A Existing Optimizers

Stochastic Gradient Descent (SGD): SGD updates model parameters by computing the gradient of the loss function with respect to the parameters for a single data point and then moving in the direction opposite to the gradient:

$$x_{t+1} = x_t - \eta_t \nabla \ell(x_t, z_t),$$

where x_t denotes the parameters or state at time t. The term x_{t+1} is the updated parameter or state at time $t+1$. The learning rate at time t, denoted as η_t, controls the size of the step taken in the direction of the negative gradient. The gradient of the loss function, $\nabla \ell$, is with respect to the parameters. The data point and its corresponding label at time t are represented as (z_t, y_t), where z_t is a single sample or a mini-batch and therefore introduces stochasticity, and y_t is the target value. We abreviate (z_t, y_t) to just z_t (Bottou et al., 1991).

SGD with Momentum (SGDM): SGDM is an extension of SGD that helps accelerate SGD in the relevant direction and dampens oscillations. It updates parameters using:

$$v_{t+1} = \gamma v_t + \eta_t \nabla \ell(x_t, z_t),$$

$$x_{t+1} = x_t - v_{t+1},$$

where v is the momentum and γ is the momentum coefficient.

Adam: Adam (Adaptive Moment Estimation) combines the advantages of two other extensions of SGD, Momentum and AdaGrad. It computes individual adaptive learning rates for different parameters:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla \ell(x_t),$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)(\nabla \ell(x_t))^2,$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t},$$

$$x_{t+1} = x_t - \eta_t \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon},$$

where β_1 and β_2 are the decay rates, and ϵ is a small constant to ensure numerical stability. Additionally $\hat{m}_t$ and $\hat{v}_t$ are debiased versions of m_t and v_t. This is necessary to correct the initialization bias. In the beginning, m_t and v_t are biased towards zero because they are initialized at zero and the moving averages use decay rates β_1 and β_2. $\hat{m}_t$ is an average of $\beta^{n-t} g_t$ for $t = 1, \ldots, n$ (Kingma and Ba, 2014).

B FTL Worst case Scenario and its Solution

Imagine a similar game to the one described in section 2.2. Let the feasible set $V = [-1, 1]$. Define $\ell_1(x) = \frac{1}{2}x$ and let ℓ_t for $t = 2, \ldots, T$ alternate between $-x$ and x. Thus,

$$\sum_{t=1}^{T} \ell_t(x) = \begin{cases} \frac{1}{2}x & \text{if } t \text{ is odd,} \\ -\frac{1}{2}x & \text{if } t \text{ is even.} \end{cases}$$

The FTL strategy will constantly switch between $x_t = -1$ and $x_t = 1$, making the incorrect decision at every iteration t. This demonstrates that the seemingly intuitive FTL approach fails in this scenario due to its instability (Hazan et al., 2016). We can improve this algorithm if we introduce a regularizer to reduce the instability. This algorithm is called Follow-The-Regularized-Leader (FTRL).

The FTRL algorithm modifies the decision rule of FTL by incorporating a regularization

function $\psi(x)$. At round $t+1$, the player selects x_{t+1} as follows:

$$x_{t+1} = \arg\min_{x \in V} \left(\ell_{1:t}(x) + \psi(x)\right)$$

where $\ell_{1:t}(x) = \sum_{i=1}^{t} \ell_i(x)$ represents the cumulative loss up to time t, and $\psi(x)$ is the regularization term. The simplest choice for $\psi(x)$ is $\frac{1}{2\eta}\|x\|^2$, with η being a positive regularization parameter. Consider the simplest example of a linear loss function $\ell_t(x) = g_t \cdot x$ with $g_t \in \mathbb{R}^n$. Using a quadratic regularizer $\psi(x) = \frac{1}{2\eta}\|x\|^2$, the update rule becomes:

$$x_{t+1} = \arg\min_{x \in \mathbb{R}^n} \left(g_{1:t} \cdot x + \frac{1}{2\eta}\|x\|^2\right)$$

where $g_{1:t} = \sum_{i=1}^{t} g_i(x)$. Solving the expression, we get:

$$x_{t+1} = -\eta g_{1:t}$$

This leads to the recursive update rule:

$$x_{t+1} = x_t - \eta g_t$$

which resembles gradient descent with a constant learning rate η. With this choice of the linear loss $\ell(x)$ and the regularizer $\psi(x)$ we can recover the simplest possible optimizer.

The regret of FTRL grows sublinearly $O(\sqrt{T})$ as shown in Appendix C. Compared to the regret of FTL that grows linearly $O(T)$, FTRL is better suited for online learning applications than FTL.

C Regret Bound for FTRL

The regret R is defined as the difference between the cumulative loss of the algorithm and the cumulative loss of the best fixed decision in hindsight:

$$R = \sum_{t=1}^{T} \ell_t(x_t) - \min_{x \in \Omega} \sum_{t=1}^{T} \ell_t(x)$$

C.1 FTRL Algorithm

The FTRL algorithm updates the weights at each iteration t as follows:

$$x_t = \arg\min_{x \in V} \left(\sum_{i=1}^{t-1} \ell_i(x) + \frac{1}{\eta} \phi(x) \right)$$

where $\phi(x)$ is a regularization function that is strongly convex, and η is a parameter.

C.2 Key Lemma

Suppose F is α-strongly convex, f is convex, and let $x = \arg\min_z F(z)$ and $x' = \arg\min_z G(z)$, where $G(x) = F(x) + f(x)$. Then:

$$0 \leq f(x) - f(x') \leq \frac{1}{\alpha} \|\nabla f(x)\|^2$$

C.3 Proof of the Key Lemma

Since F is α-strongly convex, by definition, we have:

$$F(x') \geq F(x) + \langle \nabla F(x), x' - x \rangle + \frac{\alpha}{2} \|x - x'\|^2$$

By the optimality condition of x:

$$\langle \nabla F(x), x' - x \rangle \geq 0$$

Therefore, we can simplify the strong convexity condition to:

$$F(x') \geq F(x) + \frac{\alpha}{2}\|x - x'\|^2 \tag{1}$$

Similarly, for G:

$$G(x') \geq G(x) + \langle \nabla G(x), x' - x \rangle + \frac{\alpha}{2}\|x - x'\|^2$$

By the optimality condition of x':

$$\langle \nabla G(x'), x - x' \rangle \geq 0$$

Thus, we obtain:

$$G(x') \leq G(x)$$

Since $G(x) = F(x) + f(x)$, we have:

$$G(x') = F(x') + f(x')$$

$$F(x') + f(x') \leq F(x) + f(x)$$

Rearranging this, we get:

$$f(x) - f(x') \geq F(x') - F(x) \tag{2}$$

From (1), we have:

$$F(x') \geq F(x) + \frac{\alpha}{2}\|x - x'\|^2$$

Substituting this into (2):

$$f(x) - f(x') \geq \frac{\alpha}{2}\|x - x'\|^2 \tag{3}$$

Since f is convex, using the first-order condition:

$$f(x) \leq f(x') + \langle \nabla f(x), x - x' \rangle$$

Taking the absolute value:

$$f(x) - f(x') \leq |\langle \nabla f(x), x - x' \rangle|$$

Applying the Cauchy-Schwarz inequality:

$$f(x) - f(x') \leq \|\nabla f(x)\| \cdot \|x - x'\|$$

From (3), we have:

$$\|x - x'\|^2 \leq \frac{2}{\alpha}(f(x) - f(x'))$$

Thus:

$$\|x - x'\| \leq \sqrt{\frac{2}{\alpha}(f(x) - f(x'))}$$

Substituting this back into the previous inequality:

$$f(x) - f(x') \leq \|\nabla f(x)\| \sqrt{\frac{2}{\alpha}(f(x) - f(x'))}$$

Let $a = f(x) - f(x')$. Then:

$$a \leq \|\nabla f(x)\| \sqrt{\frac{2a}{\alpha}}$$

$$a \leq \frac{2}{\alpha} \|\nabla f(x)\|^2$$

So we get:

$$f(x) - f(x') \leq \frac{1}{\alpha} \|\nabla f(x)\|^2$$

The lemma is thus proven:

$$0 \leq f(x) - f(x') \leq \frac{1}{\alpha} \|\nabla f(x)\|^2$$

C.4 Bounding the Regret

Using the key lemma:

$$\ell_t(x_t) - \ell_t(x_{t+1}) \leq \frac{1}{\alpha} \|\nabla \ell_t(x_t)\|^2$$

Summing over all t from 1 to T:

$$\sum_{t=1}^{T} (\ell_t(x_t) - \ell_t(x_{t+1})) \leq \frac{1}{\alpha} \sum_{t=1}^{T} \|\nabla \ell_t(x_t)\|^2$$

We can write the regret as:

$$R = \sum_{t=1}^{T} \ell_t(x_t) - \min_{x \in \Omega} \sum_{t=1}^{T} \ell_t(x)$$

Notice that:

$$\sum_{t=1}^{T} \ell_t(x_t) - \min_{x \in \Omega} \sum_{t=1}^{T} \ell_t(x) \leq \sum_{t=1}^{T} (\ell_t(x_t) - \ell_t(x_{t+1}))$$

Combining this with our previous bound, we get:

$$\sum_{t=1}^{T} \ell_t(x_t) - \min_{x \in \Omega} \sum_{t=1}^{T} \ell_t(x) \leq \frac{1}{\alpha} \sum_{t=1}^{T} \|\nabla \ell_t(x_t)\|^2$$

For the FTRL algorithm, $\alpha = \frac{1}{\eta}$ for the regularizer ϕ, thus:

$$R \leq \eta \sum_{t=1}^{T} \|\nabla \ell_t(x_t)\|^2$$

C.5 Final Regret Bound

If $\|\nabla \ell_t(x_t)\| \leq G$ for all t, then:

$$R \leq \frac{D}{\eta} + \eta \sum_{t=1}^{T} \|\nabla \ell_t(x_t)\|^2$$

Choosing $\eta = \sqrt{\frac{D}{TG^2}}$ gives:

$$R \leq 2G\sqrt{TD}$$

Thus, the regret of the FTRL algorithm is bounded by $R \leq 2G\sqrt{TD}$, where D is the range of the regularizer ϕ, and G is an upper bound on the gradient norms (Ma et al., 2018).

D Online Mirror Descent (OMD)

Online Mirror Descent (OMD) generalizes OGD by mapping the parameters into a dual space (to account for different geometric properties, see Figure 13) before updating them:

$$\theta_{t+1} = \nabla \psi(x_t) - \eta_t \nabla \ell(x_t, y_t),$$

$$x_{t+1} = \nabla\psi^*(\theta_{t+1}),$$

where θ_t are the dual space parameters, ψ is a convex function, and ψ^* is its conjugate. The simplest example for ψ would be the Euclidean mirror map, $\psi(x) = \frac{1}{2}\|x\|^2$, which reduces OMD to OGD, because $\nabla\psi(x) = x$ and $\nabla\psi^*(\theta) = \theta$ (Orabona, 2019).

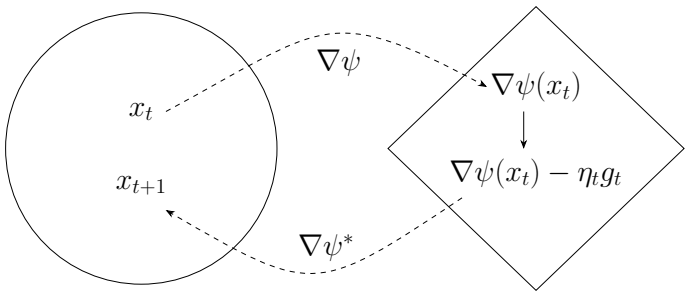

Figure 13: Visualization of the Duality Mapping used in OMD.

E FTRL functions like Adam

Utilizing the Online-to-Non-Convex (O2NC) conversion and the recursive notation, Ahn et al. (2024) observed that the Adam optimizer can be derived from the Follow-the-Regularized-Leader (FTRL) framework. In the FTRL setup, the update at each step is given by:

$$\Delta_t = -\eta_t \sum_{i=1}^{t} g_i$$

where η_t is the step size, we chose $\eta_t = \frac{\alpha}{\sqrt{\sum_{i=1}^{t} g_i^2}}$ to adapt to the gradient magnitudes. To weigh recent gradients more than past gradients and to align this with Adam, we introduce discount factors β_1 and β_2, creating a discounted-FTRL form:

$$\Delta_t = -\alpha \frac{\sum_{i=1}^{t} \beta_1^{t-i} g_i}{\sqrt{\sum_{i=1}^{t} \beta_2^{t-i} g_i^2}}$$

In combination with O2NC this formulation mirrors Adam's update rule, which employs exponentially weighted averages of past gradients and squared gradients to adjust the learning rate dynamically. With this method, we can derive the current benchmark optimizer, which is Adam, in an online learning context. This is confirmed by practical experiments. According to Figures 14 and 15 the performance of discounted-FTRL is equal to the performance of Adam in a non-online setting.

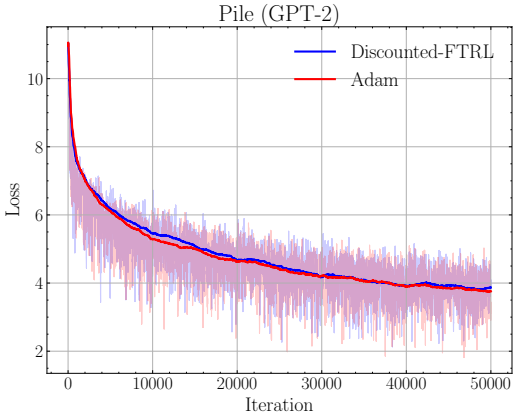

Figure 14: Loss function (smoothed with time-weighted EMA) with $\eta = 3 \times 10^{-4}$, $\beta_1 = 0.9$, and $\beta_2 = 0.999$ for both Adam and FTRL.

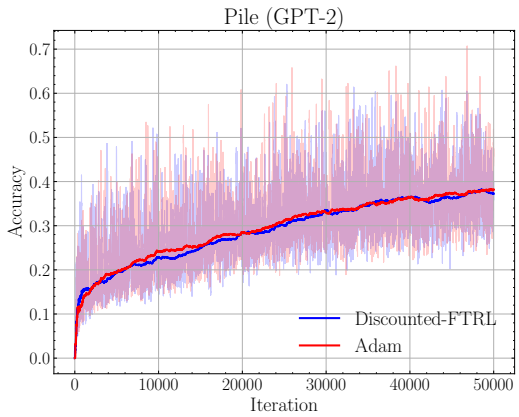

Figure 15: Accuracy (smoothed with time-weighted EMA) with $\eta = 3 \times 10^{-4}$, $\beta_1 = 0.9$, and $\beta_2 = 0.999$ for both Adam and FTRL.

F OFTRL

For all our main results using OFTRL, we used the following update rule:

$$\Delta_t = -\alpha \frac{\beta_1 \sum_{i=1}^{t} \beta_1^{t-i} g_i + (1-\beta_1) h_t}{\sqrt{\beta_2 \sum_{i=1}^{t} \beta_2^{t-i} g_i^2 + (1-\beta_2) h_t^2}}$$

In the case of calculating the intermediate "cheating" step, we used the following formula:

$$\Delta_t = -\alpha \frac{\sum_{i=1}^{t} \beta_1^{t-i} g_i}{\sqrt{\sum_{i=1}^{t} \beta_2^{t-i} g_i^2}}$$

Of course, for the actual implementation, we used a recursive update rule to save memory and computational power. Additionally, we added a bias correction term analogous to the bias correction term in the Adam optimizer:

$$\Delta_t = -\alpha \frac{\frac{\beta_1 \sum_{i=1}^{t} \beta_1^{t-i} g_i + (1-\beta_1) h_t}{1-\beta_1^t}}{\sqrt{\frac{\beta_2 \sum_{i=1}^{t} \beta_2^{t-i} g_i^2 + (1-\beta_2) h_t^2}{1-\beta_2^t}}}$$

This formula adds optimistic hints to the existing Adam optimizer derived in the O2NC framework. For every run, a cosine learning rate scheduler was used.

Alternative algorithms to the OFTRL algorithm described above have shown promise, particularly when modifying the denominator by using $\max((h_t - g_t)^2 - h_t^2, c)$ or $(h_t - g_t)^2$ instead of the traditional OFTRL update rule.

$$\Delta_t = -\alpha \frac{\sum_{i=1}^{t} \beta_1^{t-i} g_i}{\sqrt{\sum_{i=1}^{t} \beta_2^{t-i} g_i^2}}$$

We chose the modifications to take more aggressive steps, resulting in the new update rules:

$$\Delta_t = -\alpha \frac{\sum_{i=1}^{t} \beta_1^{t-i} g_i}{\sqrt{\sum_{i=1}^{t} \beta_2^{t-i} \max((h_t - g_t)^2 - h_t^2, c)}}$$

or

$$\Delta_t = -\alpha \frac{\sum_{i=1}^{t} \beta_1^{t-i} g_i}{\sqrt{\sum_{i=1}^{t} \beta_2^{t-i} (h_t - g_t)^2}}$$

These modifications resulted in performance similar to the standard discounted OFTRL, as shown by Figures 16 and 17. However, more investigation is necessary to fully understand the implications of these changes and to optimize their parameters for various machine learning applications. Further research could reveal additional benefits and potential drawbacks of these alternative approaches, leading to more robust optimization techniques.

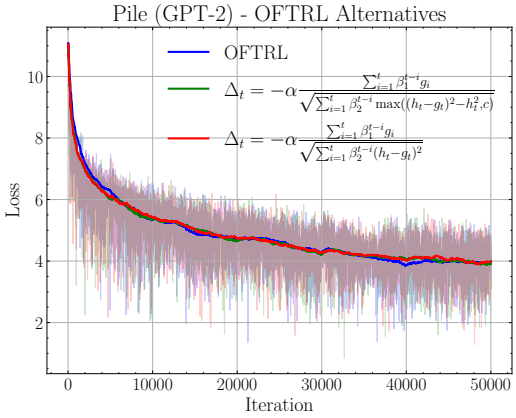

Figure 16: Loss function (smoothed with time-weighted EMA), with $h_{t+1} = \beta\Delta_t + (1-\beta)g_t$

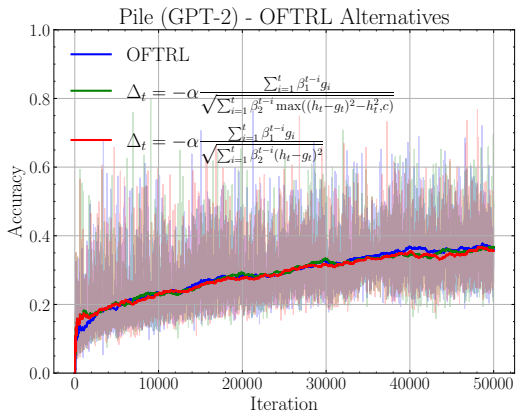

Figure 17: Accuracy (smoothed with time-weighted EMA) , with $h_{t+1} = \beta\Delta_t + (1-\beta)g_t$

G Optimistic Online Mirror Descent (OOMD)

Optimistic Online Mirror Descent (OOMD) is an extension of the traditional Online Mirror Descent (OMD) algorithm designed to leverage the additional information given through the hint:

$$x_{t+1} = \nabla\psi^* \left(\theta_t - \eta(\nabla\ell_t(x_t) + h_t - h_{t-1}\right),$$

where θ_t are the dual space parameters, ψ is a convex function, ψ^* is its conjugate, η is the learning rate, and h_t is the hint at time t. By accurately predicting the next gradient, OOMD can reduce the regret bound.

H Regret Bound for OGD

The regret R_T over T rounds is defined as:

$$R_T = \sum_{t=1}^{T} \ell_t(x_t) - \sum_{t=1}^{T} \ell_t(u)$$

where $\ell_t(x_t)$ is the loss at round t, x_t are the parameters at round t, and u is the optimal weight vector that minimizes the cumulative loss.

The weight update rule in OGD is given by:

$$x_{t+1} = x_t - \eta \nabla \ell_t(x_t)$$

where η is the learning rate and $\nabla \ell_t(x_t)$ is the gradient of the loss function at x_t.

To derive the regret bound, we start by leveraging the convexity of the loss function:

$$\ell_t(x_t) - \ell_t(u) \leq \nabla \ell_t(x_t) \cdot (x_t - u)$$

Summing over all iterations $t = 1$ to T:

$$R_T \leq \sum_{t=1}^{T} \nabla \ell_t(x_t) \cdot (x_t - u)$$

Next, we bound the inner product $\nabla \ell_t(x_t) \cdot (x_t - u)$ using the squared distance between x_t and u:

$$\|x_{t+1} - u\|^2 \leq \|x_t - \eta \nabla \ell_t(x_t) - u\|^2$$

Expanding the right-hand side, we get:

$$\|x_{t+1} - u\|^2 = \|x_t - u\|^2 - 2\eta \nabla \ell_t(x_t) \cdot (x_t - u) + \eta^2 \|\nabla \ell_t(x_t)\|^2$$

Rearranging the terms, we have:

$$2\eta\nabla\ell_t(x_t)\cdot(x_t-u)\leq\|x_t-u\|^2-\|x_{t+1}-u\|^2+\eta^2\|\nabla\ell_t(x_t)\|^2$$

Summing over all iterations $t=1$ to T:

$$2\eta\sum_{t=1}^{T}\nabla\ell_t(x_t)\cdot(x_t-u)=\sum_{t=1}^{T}\left(\|x_t-u\|^2-\|x_{t+1}-u\|^2\right)+\eta^2\sum_{t=1}^{T}\|\nabla\ell_t(x_t)\|^2$$

Notice that the sum on the right-hand side telescopes:

$$\sum_{t=1}^{T}\left(\|x_t-u\|^2-\|x_{t+1}-u\|^2\right)=\|x_1-u\|^2-\|x_{T+1}-u\|^2\leq\|x_1-u\|^2$$

Therefore,

$$2\eta\sum_{t=1}^{T}\nabla\ell_t(x_t)\cdot(x_t-u)\leq\|x_1-u\|^2+\eta^2\sum_{t=1}^{T}\|\nabla\ell_t(x_t)\|^2$$

Dividing both sides by 2η, we get:

$$\sum_{t=1}^{T}\nabla\ell_t(x_t)\cdot(x_t-u)\leq\frac{1}{2\eta}\|x_1-u\|^2+\frac{\eta}{2}\sum_{t=1}^{T}\|\nabla\ell_t(x_t)\|^2$$

By the convexity of ℓ_t, we replace $\nabla\ell_t(x_t)\cdot(x_t-u)$ with $\ell_t(x_t)-\ell_t(u)$:

$$\sum_{t=1}^{T}(\ell_t(x_t)-\ell_t(u))\leq\frac{1}{2\eta}\|x_1-u\|^2+\frac{\eta}{2}\sum_{t=1}^{T}\|\nabla\ell_t(x_t)\|^2$$

This completes the derivation of the regret bound for the OGD algorithm, providing insight into how the choice of learning rate η, initial distance to the optimal weight $\|x_1-u\|$, and the cumulative sum of squared gradients affect the overall performance.