ECE 8803: Online Decision Making in Machine Learning **Fall 2022**

Lecture 11: October 4

Lecturer: Guanghui Wang

11.1. Recap: Online convex optimization

Framework Last lecture, we introduced *online convex optimization* (OCO), which is a well established paradigm for modeling sequential decision making problems. OCO is performed in a sequence of consecutive rounds, where at iteration $t = 1, \ldots, T$, a learner firstly picks a decision $\boldsymbol{w}_t$ from a convex set $\mathcal{B} \in \mathbb{R}^d$; Simultaneously, an adversary reveals a convex loss function $\ell_t(\cdot) : \mathcal{B} \mapsto \mathbb{R}$. As a consequence, the learner suffers a loss $\ell_t(\boldsymbol{w}_t)$ and updates the decision. The goal is to minimize *regret*, which is defined as the difference between the cumulative loss of the learner with that of $\boldsymbol{w}^*$:

$$R_T = \sum_{t=1}^{T} \ell_t(\boldsymbol{w}_t) - \sum_{t=1}^{T} \ell_t(\boldsymbol{w}^*),$$

where $\boldsymbol{w}_* = \arg\min_{\boldsymbol{w} \in \mathcal{B}} \sum_{t=1}^{T} \ell_t(\boldsymbol{w})$ is the best decision in hindsight.

Assumptions For OCO, we made the following standard assumptions:

- Assumption 1 (Bounded decision set $\mathcal{B}$): We assume $\forall \boldsymbol{w} \in \mathcal{B}$, $\|\boldsymbol{w}\|_2 \leq D$;
- Assumption 2 (Bounded gradients): We assume $\forall \boldsymbol{w} \in \mathcal{B}$, $t \geq 1$, $\|\nabla \ell_t(\boldsymbol{w}_t)\|_2 \leq G$.

Here, $D > 0$ and $G > 0$ are some constants. Intuitively, Assumption 1 ensures that the decision set $\mathcal{B}$ is contained inside a d-dimensional ball, and Assumption 2 guarantees that ℓ_t does not change too quickly.

Algorithms To solve the OCO problem, one of the most natural algorithms is FTL, which, at each round t, picks the empirically best decision as $\boldsymbol{w}_{t+1}$:

$$\boldsymbol{w}_{t+1} = \arg\min_{\boldsymbol{w} \in \mathcal{B}} \sum_{i=1}^{t} \ell_i(\boldsymbol{w}).$$

However, as you may have seen in HW1, this naive algorithm can be highly *unstable* and will suffer linear regret under some counter examples (in HW1, the counter examples are constructed under the learning with expert advice problem. However, you can also find

similar counter examples in OCO where FTL will also suffer linear regret). To address this problem, we introduced the FTRL algorithm:

$$\boldsymbol{w}_{t+1} = \arg\min_{\boldsymbol{w}\in\mathcal{B}} \eta \sum_{i=1}^{t} \ell_i(\boldsymbol{w}) + R(\boldsymbol{w}),$$

where a *regularizer* $R : \mathcal{B} \mapsto \mathbb{R}$ is added to the argmin to *stabilize* the algorithm. We also put a parameter $\eta > 0$ in front of the cumulative loss to balance the two terms. We showed that, when we set $R(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2$ and $\eta = \frac{D}{G\sqrt{T}}$, the regret of FTRL is upper bounded by $O(DG\sqrt{T})$.

11.2. Online Gradient Descent

In today's lecture, we will discuss online gradient descent (OCO), which is another important algorithm for solving OCO problems. Last time (recall Section 10.5.1 of Lecture 10), we already briefly mentioned the general form of OGD, which can be seen as a special case of FTRL in the unconstrained case, i.e.,

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta\nabla\ell_t(\boldsymbol{w}_t). \tag{11.1}$$

However, the update in (11.1) does not work in the general OCO setting that is defined in Section 11.1, since in Assumption 1 we have assumed that the decision set $\mathcal{B}$ is *bounded*, while the $\boldsymbol{w}_{t+1}$ in (11.1) is not guaranteed to be inside $\mathcal{B}$. To address this issue, we have to introduce the following projection operator:

Definition 1 (Projection operator) *For any point $\boldsymbol{z} \in \mathbb{R}^d$ and convex set $\mathcal{B} \in \mathbb{R}^d$, define the projection operator $\Pi_{\mathcal{B}}(\boldsymbol{z}) = \arg\min_{\boldsymbol{z}'\in\mathcal{B}} \|\boldsymbol{z} - \boldsymbol{z}'\|_2$.*

Intuitively, the projection operator $\Pi_{\mathcal{B}}(\boldsymbol{z})$ takes any point $\boldsymbol{z} \in \mathbb{R}^d$ as input, and outputs a point in $\mathcal{B}$ which is closest to $\boldsymbol{z}$. With this new tool, we redefine the OGD algorithm as

$$\begin{cases} \boldsymbol{z}_t = \boldsymbol{w}_t - \eta\nabla\ell_t(\boldsymbol{w}_t) \\ \boldsymbol{w}_{t+1} = \Pi_{\mathcal{B}}(\boldsymbol{z}_t), \end{cases} \tag{11.2}$$

that is, in each round t, we firstly performs a gradient decent step to get an intermediate point $\boldsymbol{z}_t$ (the first line), and then project $\boldsymbol{z}_t$ back to our decision set $\mathcal{B}$ by using the projection operator to get $\boldsymbol{w}_{t+1}$ (the second line).

The following theorem shows that, with appropriately chosen step size η, OGD can also achieve an $O(DG\sqrt{T})$ bound, similar to that of FTRL. To be more specific, we can prove the following theorem.

Theorem 1 *Suppose Assumptions 1 and 2 hold, and let $\eta = \frac{D}{G\sqrt{T}}$. Then, the OGD algorithm defined in* (11.2) *enjoys the following regret bound:*

$$\sum_{t=1}^{T} \ell_t(\boldsymbol{w}_t) - \sum_{t=1}^{T} \ell_t(\boldsymbol{w}^*) \leq \frac{3DG\sqrt{T}}{2}.$$

In class, we went over the proof of this theorem step-by-step. The details can be found in the Appendix. Here in the main body we only discuss the basic idea of the proof.

Proof sketch Note that, the regret R_T is the cumulative sum of $\ell_t(\boldsymbol{w}_t) - \ell_t(\boldsymbol{w}^*)$ (the term $[\ell_t(\boldsymbol{w}_t) - \ell_t(\boldsymbol{w}^*)]$ is sometimes referred to as the *instantaneous regret of the algorithm at round* t). Our main idea is to firstly do a "one-step analyisis" to upper bound $\ell_t(\boldsymbol{w}_t) - \ell_t(\boldsymbol{w}^*)$, and then sum them together to get the the upper bound of R_T. Specifically, based on the convexity of $\ell_t(\boldsymbol{w})$, the property of the projection operator and the update rule in (11.2), we have

$$\forall t \geq 1, \qquad \ell_t(\boldsymbol{w}_t) - \ell(\boldsymbol{w}^*) \leq \frac{\eta G^2}{2} + \frac{\|\boldsymbol{w}_t - \boldsymbol{w}^*\|_2^2 - \|\boldsymbol{w}_{t+1} - \boldsymbol{w}^*\|_2^2}{2\eta},$$

and then the regret bound can be obtained by summing the above inequality over from 1 to T and substituting the configuration of η into the equation.

11.3. Adaptive OCO algorithms

We have learned that the two classical OCO algorithms (i.e., OGD and FTRL) can enjoy the $O(\sqrt{T})$ regret bound. A very natural and important follow-up questions is:

Is $O(\sqrt{T})$ the best we can do?

For the this question, The short answer is *Yes*, since we can actually construct the a *lower bound* for the any OCO algorithms:

Claim 1 (Informal) *For any OCO algorithms $\mathcal{A}$, we can find an OCO problem, on which the regret of $\mathcal{A}$ will be lower bounded by $C\sqrt{T}$, where $C > 0$ is some constant*[1]*.*

The above claim implies that it is *impossible* to design an OCO algorithm that can beat the $O(\sqrt{T})$ bound for *all* OCO problems, because we can always find at least one "bad" problem where the algorithm would suffer at least $C\sqrt{T}$ regret.

However, the lower bound above is too pessimistic, since it mainly focus on some extreme case that rarely happens. Although designing one algorithm that can beats $O(\sqrt{T})$ on *all* OCO problems is impossible, it is still possible to design algorithms that can ensure the $O(\sqrt{T})$ regret bound in general, while *automatically* achieving bounds tighter than $O(\sqrt{T})$ when the OCO problems are easy or structured. These kind of algorithms are called adaptive OCO algorithms, since they can adapt to the problems' intrinsic structure. In the following, we introduce several different adaptive OCO algorithms, which can get tighter bounds under different scenarios.

11.3.1 Adapting to gradient size in OCO

In the OCO setting, we may desire a different type of adaptation *to the scale of the losses*. In fact, a closer look at the regret bound for FTRL (or OGD) in OLO (or OCO) was given by

$$R_T \leq \eta \sum_{t=1}^{T} ||\boldsymbol{\ell}_t||_2^2 + \frac{D^2}{\eta}, \tag{11.3}$$

1. Note that this is just a very informal claim. In problem 2, HW3, you can construct the lower bound by yourself step-by-step.

where D is the upper bound on $||\boldsymbol{w}||_2$ for all decision vectors $\boldsymbol{w} \in \mathcal{B}$. We directly used that fact that $||\boldsymbol{\ell}_t||_2^2 \leq G^2$ to get an upper bound of $\eta G^2 T + \frac{D^2}{\eta}$. However, in reality, the loss vectors could be smaller in scale on some rounds than others. For example, stocks could fluctuate very little at the beginning of a prediction process and a lot later.

The natural question arises of whether we can adapt to such potential structure if it exists. If we knew all of the loss vectors beforehand, we could select $\eta := \frac{D}{\sqrt{\sum_{t=1}^T ||\boldsymbol{\ell}_t||_2^2}}$ to minimize the regret upper bound, and we would then get $R_T = \mathcal{O}(D\sqrt{\sum_{t=1}^T ||\boldsymbol{\ell}_t||_2^2})$. This is clearly a bound that adapts to the scale of the problem: smaller loss vectors will lead to a smaller regret. However, just like in the case of sequence prediction, this choice would require knowing the loss vectors beforehand — which will not be the case, as we are in an online learning setting. However, we can use the same trick as before and use the information we have available at every round. Concretely, we can set a time-varying, data-dependent step-size given by

$$\eta_t := \frac{D}{\sqrt{\sum_{s=1}^t ||\boldsymbol{\ell}_s||_2^2}},$$

and it again turns out to give exactly the regret bound that we want, i.e.

$$R_T = \mathcal{O}\left(D\sqrt{\sum_{t=1}^T ||\boldsymbol{\ell}_t||_2^2}\right). \tag{11.4}$$

In fact, the only difference between the "oracle" bound (if we knew the loss vectors beforehand) and the actually attainable bound above is in constant factors! Finally, note that, when the loss functions are smooth and non-negative, the regret bound of this algorithm can be improved to

$$R_T = \mathcal{O}\left(D\sqrt{\sum_{t=1}^T \ell_t(\boldsymbol{w}^*)}\right),$$

which is referred to as the "small-loss" bound. This kind of bound can be much more tighter when the cumulative loss of the best decision is small.

11.3.2 Adapting to sparse *data*: AdaGrad-type of algorithms

The final and most influential adaptive online learning algorithm that we will discuss is the AdaGrad algorithm. This algorithm was first proposed by Duchi et al. (2011) and has since seen extensive use in diverse ML applications across text, vision and speech.

The motivation for AdaGrad comes from dealing with *sparse* data in ML, in the sense that only a small number of features may influence the actual outcome and therefore the loss function. Concretely, suppose that $\boldsymbol{w}_t$ denote the parameters of our ML model and $\boldsymbol{\ell}_t$ denote the gradient of the loss incurred at round t by that model. The term $\boldsymbol{\ell}_t$ is the data-depedent term, in that it depends on the features of the data and In ML applications, it is usually the individual parameters of the model that will be bounded, i.e. we will have $w_{t,i} \leq D'$ for some value D'. Therefore, the ensuing constraint on the ML models will be $||\boldsymbol{w}_t||_2 \leq D := \sqrt{d}D'$,

where d here represents the number of parameters in the model. Plugging this constraint into the regret bounds that we have derived in class (i.e $R_T = \mathcal{O}(DG\sqrt{T})$) gives us an extra dependence on d: this is very undesirable in modern ML settings, which are very high-dimensional.

Can we reduce this dimension dependence? Yes. It turns out that there are several situations in ML in which the gradients of the loss functions, which we denoted by $\boldsymbol{\ell}_t$, may be *sparse* in the sense of having only a few non-zero entries. In the context of ML, this means that only a small number of features in the *data* may influence the actual prediction outcome and therefore the loss function $\ell_t(\cdot)$. Two concrete instantiations of this are provided below:

- In traditional text classification tasks, the input is given by 0/1 bigram features: for example, in the Reuters RCV-1 text dataset, the task is to classify news articles into one of four categories: "Economics', "Commerce", "Medical", and "Government". The number of bigram features used for classification is equal to 2 million; however, these feature vectors tend to be exceptionally sparse, typically containing fewer than 5000 non-zero features.

- In the days before deep learning, people used to solve computer vision tasks with hand-crafted features, including ImageNet! One such task involves an image-ranking task corresponding to each noun in ImageNet (which is associated with more than 500 images). For example, one could use the "visterms" features, which are $10,000$-dimensional but in reality have at most 100 non-zero components. This is another example of sparsity in feature space.

Intuitively, to perform correct classification we need to determine to what extent each feature influences the classification outcome. Accordingly, we would like to give frequently occurring features very low learning rates, and infrequently occurring features a higher learning rate: every time an infrequent feature is encountered, we as the learner should "take notice".

It turns out that the AdaGrad algorithm does exactly this. The main idea is to treat our high-dimensional problem (with such potential sparsity) as d separate OLO problems. The essential observation is that we can write the regret *relative* to a particular decision vector $\boldsymbol{w}$ as the decomposition of d "regrets" along each dimension, i.e.

$$\begin{aligned} R_T(\boldsymbol{w}) &:= \sum_{t=1}^{T} \langle \boldsymbol{\ell}_t, \boldsymbol{w}_t \rangle - \sum_{t=1}^{T} \langle \boldsymbol{\ell}_t, \boldsymbol{w} \rangle \\ &= \sum_{t=1}^{T} \sum_{i=1}^{d} l_{t,i} w_{t,i} - \sum_{t=1}^{T} \sum_{i=1}^{d} l_{t,i} w_i =: \sum_{i=1}^{d} R_{T,i}(\boldsymbol{w}) \end{aligned}$$

In other words, we decompose the regret into the sum of d separate, 1-dimensional OLO problems and try and solve them independently. This is the main idea of the most basic version of AdaGrad: in particular, it sets an *individual learning rate*

$$\eta_{t,i} = \frac{1}{\sqrt{\sum_{s=1}^{t} g_{s,i}^2}}$$

corresponding to each coordinate i. In the context of the ML examples above, you can think of the parameter $g_{t,i}$ as being related to the magnitude of feature i of the training example t. Since the learning rate is inversely proportional to these scalings, it is clear that we are giving infrequently occurring features higher learning rates.

AdaGrad is known to achieve the following regret rate:

$$R_T = \mathcal{O}\left(D' \cdot \sum_{i=1}^{d} \sqrt{\sum_{t=1}^{T} g_{t,i}^2}\right). \tag{11.5}$$

It is interesting and instructive to compare Equations (11.4) and (11.5) in an extreme sparse scenario: suppose, in particular, that only the gradients along direction 1 are non-zero (with value at most G) and all other gradients are 0. Then, Equation (11.4) can be verified to give a regret bound of $\mathcal{O}(D'G\sqrt{dT}$, while Equation (11.5) will only give a regret of $\mathcal{O}(D'G\sqrt{T})$, with no dependence on the dimension! Extending this logic further, one could see that AdaGrad would *adapt to the intrinsic sparsity level in the data*. This turns out to have substantial benefits in high-dimensional ML settings; see, e.g. the experimental results in Duchi et al. (2011); Dean et al. (2012)

Finally, we provide an alternative intuitive explanation of Adagrad. Lets take a close look of the step size for the i-th dimension:

$$\eta_{t,i} = \frac{1}{\sqrt{\sum_{s=1}^{t} g_{s,i}^2}} = \frac{1}{\sqrt{t\frac{1}{t}\sum_{s=1}^{t} g_{s,i}^2}} = \frac{1}{\sqrt{t}} \frac{1}{\sqrt{\frac{1}{t}\sum_{s=1}^{t} g_{s,i}^2}}.$$

In the right-hand-side of the equality, we observe that, the step size of Adagrad is $\frac{1}{\sqrt{t}}$ in general, and is adjusted by the *weighted average* of $g_{s,i}^2$. When the gradients of the i-th dimension are sparse, $\sqrt{\frac{1}{t}\sum_{s=1}^{t} g_{s,i}^2}$ will become very small (since there are many zeros), so the step size will be large, and vise versa. It means that, Adagrad can automatically assign a large step size for spares dimensions, so that it can catch the infrequent but valuable information therein. Meanwhile, it will keep a $1/\sqrt{t}$ step size for non-sparse dimensions, just let OGD.

Variants of Adagrad The AdaGrad algorithm also inspired more sophisticated adaptive algorithms. In the lecture, we briefly discussed three of them, including RMSprop, Adam and AMSgrad (We only need to have a look at the basic ideas of these popular methods for fun).

- RMSprop. RMSprop is a famous variant of Adagrad, and is proposed by Geoffrey Hinton in one of his lecture notes in 2012. The main idea is to use *exponential weighted average* to adjust the step size, instead of the uniform average, which is used in Adagrad. Specially, in RMSprop, the step size $\eta_{t,i}$ is

  $$\eta_{t,i} = \frac{1}{\sqrt{t}} \cdot \frac{1}{\sqrt{\sum_{s=1}^{t-1}(1-\beta)^{t-s} g_{s,i}^2 + \beta g_{t,i}^2}},$$

  where $0 < \beta \leq 1$ is a constant. We can observe that, the weight we assigned for $g_{s,i}$ decrease exponentially, so the gradients obtained in the rounds that are close to t

will play a more important role (since they are more related, they should have larger weights).

- Adam. Adam combines RMSprop with momentum. Momentum is the moving average of gradients: In round t, momentum is given by $\mathbf{v}_t = (1 - \beta_m)\mathbf{v}_{t-1} - \beta_m \nabla \ell_t(\boldsymbol{w}_t)$, where $0 < \beta_m < 1$ is a parameter. Adam uses the same step size as RMSprop, but does the descent step with $\mathbf{v}_t$, instead of $\nabla \ell_t(\boldsymbol{w}_t)$. See Images 2 and 3 of this blog to `https://ruder.io/optimizing-gradient-descent/index.html` for an illustration of why momentum is helpful.

- AMSgrad. In Adagrad, we can easily show that the step size is *non-increasing*. However, in Adam, the step size can decrease or *increase*. A recent work (Reddi et al., 2019) shows that, because of this reason, Adam may diverge in some OCO problems, which means that it can suffer linear regret. To avoid this issue, Reddi et al. (2019) proposed AMSgrad, which forces the step size of Adam to be non- decreasing. They showed that the new algorithm can ensure a regret bound similar to that of Adagrad. However, what is the best way to fix Adam is still an open question.

See Image 5 in the blog post: `https://ruder.io/optimizing-gradient-descent/index.html` for an illustration of the convergence of these algorithms in the case of the Beale function in 2 dimensions, which is given by $f(x, y) = (1.5 - x + xy)^2 + (2.25 - x + xy^2)^2 + (2.625 - x + xy^3)^2$ on the domain $[-4.5, 4.5] \times [-4.5, 4.5]$. This turns out to be convex in x, y (athough it is difficult to verify), but with very large gradients at the extremes as visualized in Figure 11.1. These are clearly settings where the magnitude of the gradient can vary from iteration to iteration, and setting an adaptive step size turns out to be extremely useful for faster convergence.

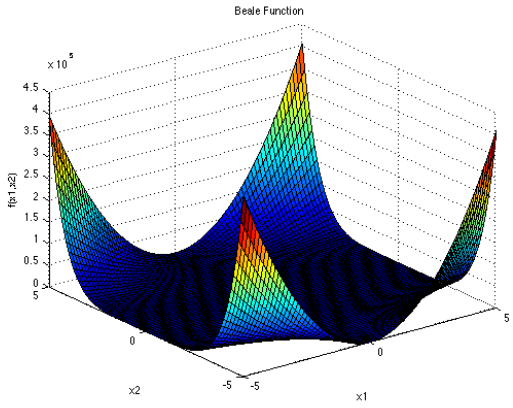

Figure 11.1: Surface plot of the Beale function, given by: $f(x, y) = (1.5 - x + xy)^2 + (2.25 - x + xy^2)^2 + (2.625 - x + xy^3)^2$.

11.4. Bibliographical notes

- Francesco Orabona's book draft is an excellent reference to learn more about adaptive online learning algorithms (and OCO in general). The brief treatment of AdaGrad that is provided here borrows from pages $26 - 29$ of this book. We did not cover proofs of why AdaGrad works or the full scope of guarantees on regret and optimization convergence: the full details are in the paper (Duchi et al., 2011) (which is a good, but advanced read). Related adaptive approaches were introduced by the concurrent work (McMahan and Streeter, 2010), but with a focus on what is called the "competitive ratio", a complementary notion of measuring performance to regret. While studying the competitive ratio of online algorithms is an extremely important topic, it is out of scope for this class.

- The examples that we have provided for the benefits of AdaGrad in training ML algorithms are directly from the original AdaGrad paper (Duchi et al., 2011). The use of AdaGrad in training deep neural networks was popularized by Dean et al. (2012). However, theoretical guarantees on its performance typically do not exist owing to the non-convexity of the problem of training a deep neural network. In practice, the tendency of AdaGrad to constantly shrink its learning rate is not always desired. Accordingly, several heuristic improvements over AdaGrad have been proposed, the most notable of which is Adam (Kingma and Ba, 2015). While Adam has seen tremendous practical impact and is the gold standard for training deep neural networks today, it is also very poorly understood and may not converge even in simple scenarios (Reddi et al., 2019). Informal blog posts such as this one: `https://ruder.io/optimizing-gradient-descent/index.html` provide a good window into how optimization algorithms are chosen in practice, and the intuition behind why they are used (but most of this intuition is not yet backed up by rigorous theory).

- There are several GitHub implementations of AdaGrad, e.g. `https://github.com/benbo/adagrad` that you may be interested in playing around with.

References

Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. Large scale distributed deep networks. *Advances in neural information processing systems*, 25:1223–1231, 2012.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *International Conference for Learning Representations*, 2015.

H Brendan McMahan and Matthew Streeter. Adaptive bound optimization for online convex optimization. *arXiv preprint arXiv:1002.4908*, 2010.

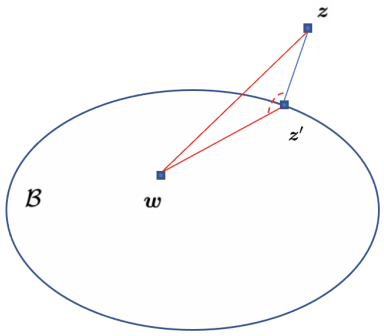

Figure 11.2: Figure for Lemma 2

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

Appendix A. Proof of Theorem 1

Before giving the proof, we first introduce a useful property of the projection operator.

Lemma 2 *Let $\boldsymbol{z}$ be any point in $\mathbb{R}^d$, $\mathcal{B}$ be a convex set, and $\boldsymbol{w}$ be a point in $\mathcal{B}$. Let $\boldsymbol{z}' = \Pi_{\mathcal{B}}[\boldsymbol{z}]$. Then we have*

$$\|\boldsymbol{z} - \boldsymbol{w}\|_2 \geq \|\boldsymbol{z}' - \boldsymbol{w}\|_2.$$

In class, we shortly discussed the proof of this lemma. Basally, we can draw a triangle (Figure 11.2), and prove lemma 2 by showing the angle $\angle \boldsymbol{w}\boldsymbol{z}'\boldsymbol{z} \geq 90°$. We can prove $\angle \boldsymbol{w}\boldsymbol{z}'\boldsymbol{z} \geq 90°$ by contradiction and using the definition of the projection operator.

Now we are ready to prove Theorem 1. We first do the following one-step analysis.

$$\begin{aligned}
\ell_t(\boldsymbol{w}_t) - \ell_t(\boldsymbol{w}^*) &\leq \langle \nabla \ell_t(\boldsymbol{w}_t), \boldsymbol{w}_t - \boldsymbol{w}^* \rangle \\
&= \frac{1}{\eta} \langle \boldsymbol{z}_t - \boldsymbol{w}_t, \boldsymbol{w}_t - \boldsymbol{w}^* \rangle \\
&\leq \frac{1}{2\eta} \left[\|\boldsymbol{z}_t - \boldsymbol{w}_t\|_2^2 + \|\boldsymbol{w}_t - \boldsymbol{w}^*\|_2^2 - \|\boldsymbol{z}_t - \boldsymbol{w}^*\|_2^2 \right] \\
&\leq \frac{1}{2\eta} \left[\|\boldsymbol{z}_t - \boldsymbol{w}_t\|_2^2 + \|\boldsymbol{w}_t - \boldsymbol{w}^*\|_2^2 - \|\boldsymbol{w}_{t+1} - \boldsymbol{w}^*\|_2^2 \right] \\
&= \frac{1}{2\eta} \left[\eta^2 \|\nabla \ell_t(\boldsymbol{w}_t)\|_2^2 + \|\boldsymbol{w}_t - \boldsymbol{w}^*\|_2^2 - \|\boldsymbol{w}_{t+1} - \boldsymbol{w}^*\|_2^2 \right] \\
&= \frac{\eta \|\nabla \ell_t(\boldsymbol{w}_t)\|_2^2}{2} + \frac{\|\boldsymbol{w}_t - \boldsymbol{w}^*\|_2^2 - \|\boldsymbol{w}_{t+1} - \boldsymbol{w}^*\|_2^2}{2\eta} \\
&= \frac{\eta G^2}{2} + \frac{\|\boldsymbol{w}_t - \boldsymbol{w}^*\|_2^2 - \|\boldsymbol{w}_{t+1} - \boldsymbol{w}^*\|_2^2}{2\eta}
\end{aligned} \tag{11.6}$$

Here, the first line is based on the first-order condition of convexity (recall the inequality under Figure 10.3 of lecture 10), the second line is because the update rule (the first line in

(11.2). Since $\boldsymbol{z}_t = \boldsymbol{w}_t - \eta \nabla \ell_t(\boldsymbol{w}_t)$, we have $\nabla \ell_t(\boldsymbol{w}_t) = \frac{1}{\eta}(\boldsymbol{z}_t - \boldsymbol{w}_t)$), the third line is due to the fact that $2ab = a^2 - b^2 - (a-b)^2$ (consider $\boldsymbol{z}_t - \boldsymbol{w}_t$ as a, $\boldsymbol{w}_t - \boldsymbol{w}^*$ as b), the forth line is an application of Lemma 2, the fifth line is based on the the update rule (the first line in (11.2). Since $\boldsymbol{z}_t = \boldsymbol{w}_t - \eta \nabla \ell_t(\boldsymbol{w}_t)$, we have $\nabla \ell_t(\boldsymbol{w}_t) = \frac{1}{\eta}(\boldsymbol{z}_t - \boldsymbol{w}_t)$). The last line is based on Assumption 2 (bounded gradients).

Sum it over from 1 to T, we have

$$\sum_{t=1}^{T} \ell_t(\boldsymbol{w}_t) - \sum_{t=1}^{T} \ell_t(\boldsymbol{w}^*) \leq \frac{\eta G^2 T}{2} + \sum_{t=1}^{T} \frac{\|\boldsymbol{w}_t - \boldsymbol{w}^*\|_2^2 - \|\boldsymbol{w}_{t+1} - \boldsymbol{w}^*\|_2^2}{2\eta}.$$

Note that the second term is a telescope sum. We can get

$$\sum_{t=1}^{T} \frac{\|\boldsymbol{w}_t - \boldsymbol{w}^*\|_2^2 - \|\boldsymbol{w}_{t+1} - \boldsymbol{w}^*\|_2^2}{2\eta} = \frac{\|\boldsymbol{w}_1 - \boldsymbol{w}^*\|_2^2 - \|\boldsymbol{w}_{T+1} - \boldsymbol{w}^*\|_2^2}{2\eta} \leq \frac{D^2}{\eta}.$$

Thus,

$$\sum_{t=1}^{T} \ell_t(\boldsymbol{w}_t) - \sum_{t=1}^{T} \ell_t(\boldsymbol{w}^*) \leq \frac{\eta G^2 T}{2} + \frac{D^2}{\eta},$$

and we can finish the proof by plugging in the value of η.