**CSCI699: Machine Learning Theory** Fall 2023

# Lecture 8: Memory-Sample Tradeoffs, Convex Optimization

*Instructor: Vatsal Sharan*

*These lecture notes are based on scribe notes by Ali Omrani, Navid Hashemi and Bhavya Vasudeva.*

## 1 Memory-Sample Tradeoffs

We'll now turn to a different kind of statistical-computational tradeoff. For most of the class, we have used the number of operations performed by the algorithm as the proxy for the algorithms running time and its computational efficiency. However, in many contemporary settings (including large-scale systems and devices on the edge), the memory or space usage of the algorithm can be crucial in determining efficiency. This is because growth in the available processing power has outpaced the growth in the available memory by many orders of magnitude (both due to Moore's law and specialized hardware such as GPUs), with the result that memory and data movements due to a shortage of memory are often the dominant performance and energy bottlenecks in modern learning systems. In addition, from a theoretical standpoint, memory is one of the most fundamental computational resources (e.g. for a Turing machine the number of steps that it runs for and its space usage are the two most fundamental metrics for efficiency).

Let us see a glimple of the role that memory plays in learning, in particular, if there are tradeoffs between the available memory and the number of samples needed for learning.

**Memory-Sample Tradeoffs for Parity Learning**

We'll understand this for the parity problem. The setting is as follows. The data comes in streaming fashion. We get datapoints one at a time, only get a single pass over your data stream. The goal is to learn the parity function:

$$\begin{aligned}
\mathcal{X}^d &= \{0,1\}^d \\
\mathcal{Y} &= \{0,1\} \\
\mathcal{C} &= \{w(x) = \langle w, x\rangle \mod 2 : w \in \{0,1\}^d\}.
\end{aligned}$$

The setup is as follows. There is some unknown $w^* \in \{0,1\}^d$ which we want to find. At every timestep we get a labelled example. We can store the example in memory if we like, or do some computation based on the example and store the result of the computation, but we don't get to see the example again i.e. we only get one pass over the datastream:

At $t = 1$

- Get $x_1 \sim Unif(\{0,1\}^d)$
- Get $b_1 = \langle x_1, w^*\rangle \mod 2$

At $t = 2$

- Get $x_2 \sim Unif(\{0,1\}^d)$
- Get $b_2 = \langle x_2, w^* \rangle mod\ 2$

and so on. Let us try to understand what algorithms might be possible here. The unknown vector $x^*$ is $d$-dimensional, so $\Omega(d)$ memory is necessary for even storing the solution, and hence for solving the problem. Also, since we're looking at a linear system (over GF(2)) in $d$-dimensions, $\Omega(d)$ examples are also necessary for learning $x^*$. So if we try to solve the problem with only $O(d)$ memory, could we still get the optimal $O(d)$ sample complexity? It does not seem easy, because with $O(d)$ memory we can only store a constant number of examples at a time, and maybe in that case we would need to see many more than $O(d)$ examples to solve the problem. Which brings us to the following question:

*For the parity learning problem, what is the tradeoff between the available memory and the number of samples needed for learning?*

We first consider two very natural algorithms for the problem.

**Algorithm 1**

Store $n = O(d)$ examples in memory and solve the linear system.

$$\begin{pmatrix} \text{---}x_1\text{---} \\ \text{---}x_2\text{---} \\ . \\ . \\ . \\ \text{---}x_n\text{---} \end{pmatrix} \begin{pmatrix} \\ \\ w^* \\ \\ \\ \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ . \\ . \\ . \\ b_n \end{pmatrix} \text{mod } 2$$

Since $w^*$ is d-dimensional with $n >> 100d$ examples, the system is full-rank w.h.p. Let us see the memory usage and sample complexity of this algorithm:

Samples $= n = O(d)$
Memory $= nd$ bits $= \Omega(d^2)$

So the algorithm is great in terms of its sample complexity, but uses a lot of memory. The following brute force search algorithm achieves the other extreme.

**Algorithm 2** Brute-Force Search

```
for every $w \in \{0,1\}^d$ do
    if $w$ is consistent over the next $o(d)$ examples we receive then
        return w
    end if
end for
```

Memory $= O(d)$
Samples $= d2^d = 2^{\Omega(d)}$

**Question:** What else is possible?

In a surprising result, Ran Raz showed that the above two algorithms are essentially all that is possible for the problem, and therefore we have almost a complete understanding of the algorithmic landscape of the problem.

**Theorem 1** ([1]). *Any algorithm for solving the above parity problem either requires $\Omega(d^2)$ memory, or at least $2^{\Omega(d)}$ samples.*

Interestingly, all our previous computational lower bounds were based on assumptions such as RP$\neq$ NP, but the above theorem is unconditional. It seems that understanding memory-sample tradeoffs is much more information-theoretic, and we can indeed show stark, unconditional lower bounds.

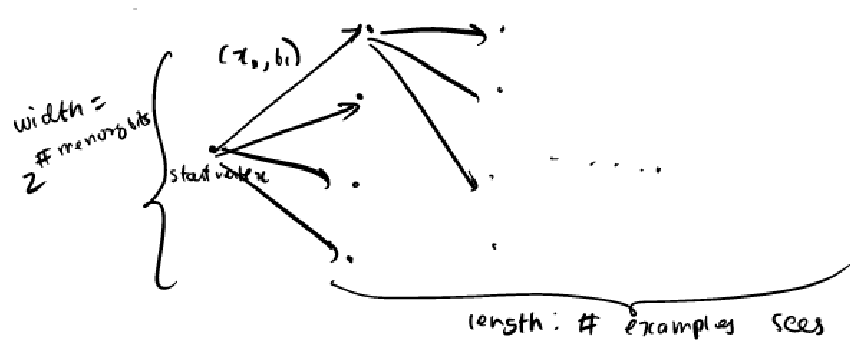


Figure 1: The proof is based on the idea of analyzing a branching program for the problem.

Subsequent work has extended the result to a much larger class of problems. For instance, the following result shows a memory lower bound based on the SQ dimension of the learning task.

**Theorem 2** ([2]). *Consider a hypothesis class $\mathcal{H}$ with SQ-dim$(\mathcal{H}) = s$. Then any algorithm for learning $\mathcal{H}$ requires $\Omega(\log^2 s)$ memory, or at least $s^{\Omega(1)}(poly(s))$ samples.*

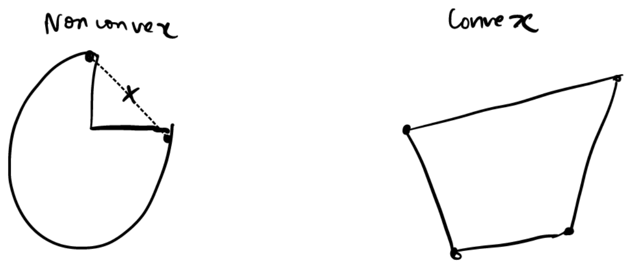
An upper bound which shows that classes with small SQ dimension can be learned with small memory is also known [3]. We saw earlier that large SQ dimension can make learning hard in the presence of noise (with a concept class based on parities being the only exception where learning is still possible despite the SQ dimension being large). The above theorem shows that learning with small memory is also hard with large SQ dimension. This is possibly not a complete coincidence, it appears that algorithmic problems which are easy can be solved with various constraints or deviations from ideal behavior (for example memory constraints or random noise), but problems which are hard become difficult to solve in the presence of such constraints or deviations. For example, though parities are PAC-learnable, they become hard to learn in the presence of noise, or under memory constraints. In constrast, "simpler" classes such as conjunctions remain easy to learn under these constraints. In this sence, hardness with respect to many different contraints (such as running time, memory, or even communication in distributed settings) appears to be quite related.

## 2 Convex Optimization

So far in this class, we started with how much data is needed for learning, then went on to the computational aspects, and saw algorithms for learning certain problems efficiently. We investigated the minimum resources required for learning—in terms of number of samples, running time, and memory. We have seen a few different frameworks for understanding these resources, such as VC dimension and the SQ model. In this lecture, we will explore a powerful algorithmic framework, *convex optimization.* Convex optimization forms the algorithmic backbone of machine learning.

We start with some basic properties:

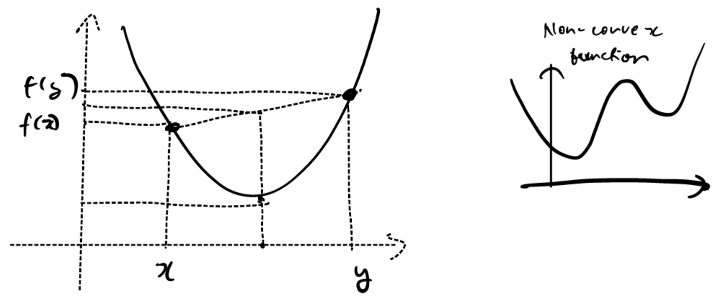
**Definition 3** (Convex set). *A set $\mathcal{C} \subseteq \mathbb{R}^d$ is convex if $x, y \in \mathcal{C} \implies tx + (1-t)y \in \mathcal{C},\ \forall 0 \le t \le 1$.*



Therefore a set is convex if the line joining any points on the set is also within the set. Convex functions can be defined similarly.

**Definition 4** (Convex function). *A function $f : \mathbb{R}^d \to \mathbb{R}$ is convex if domain($f$) $\subseteq \mathbb{R}^d$ is convex and*

$$f(tx + (1-t)y) \le tf(x) + (1-t)f(y) \quad \forall\ t \in [0,1] \text{ and } (x, y) \in domain(f).$$



Informally, the line joining any two points on the graph of the function must be above the function for the function to be convex. The following is an equivalent definition of convex functions, and says that the function must always be above the tangent at any point.

**Lemma 5.** *If $f$ is differentiable, then $f$ is convex if and only if Domain(f) is convex and:*

$$f(y) - f(x) \ge \langle y - x, \nabla f(x) \rangle$$
$$\text{(or equivalently, } f(y) - f(x) \le \langle y - x, \nabla f(y) \rangle).$$

**Proof.** *Exercise, use the definition of convexity.* ■

Convex functions are nice, since local information is sufficient to ensure optimality. The following corollary follows from the previous lemma.

**Corollary 6** (Local minimum implies global minimum)**.** *If $f$ is convex and differentiable the $\nabla f(x) = 0$ implies $x$ is a global minima of $f$.*

The following are some simple properties of convex functions. Together, they are sufficient to show convexity for most functions which are convex.

**Lemma 7** (Some Properties of convex functions)**.** *The following properties are true for convex functions,*

1. *If $f$ is twice differentiable, then $f$ is convex iff domain(f) is convex and $\forall\ x \in domain(f), \nabla^2 f(x) \succeq 0$, (where $A \succeq 0 \iff x^\top Ax \geq 0\ \forall x$).*
2. *If $f_i(x)$ is convex $\forall i \in [n]$, then $y(x) = \sum_{i=1}^{n} w_i f_i(x)$ where $w_i \geq 0\ \forall i$ is convex.*
3. *If $f_i(x)$ is convex $\forall\ i \in [n]$, then $g(x) = \max_{i \in [n]} f_i(x)$ is convex.*

## 3 Convex learning Problems

An optimization problem

$$\min_{x \in A} f(x)$$

is called a convex optimization problem if (1) $f(x)$ is convex (2) $A$ is convex. *Convex optimization problems can generally be solved efficiently.*

Recall the ERM problem of finding the ERM w.r.t some hypotheis class $\mathcal{H}$ on some training set $\mathcal{S} = (z_1, z_2, \cdots, z_n)$ where $z_i = (x_i, y_i)$,

$$\text{ERM}_{\mathcal{H}}(\mathcal{S}) = \arg\min_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \ell(h, z_i).$$

Let the hypothesis class $\mathcal{H}$ be parameterized by $w \in \mathbb{R}^d$. We will overload notation slightly, and assume that the domain of $w$ is also $\mathcal{H}$. Then we can write,

$$\text{ERM}_{\mathcal{H}}(\mathcal{S}) = \arg\min_{w \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \ell(w, z_i).$$

Then the following result follows since the average of convex functions is convex.

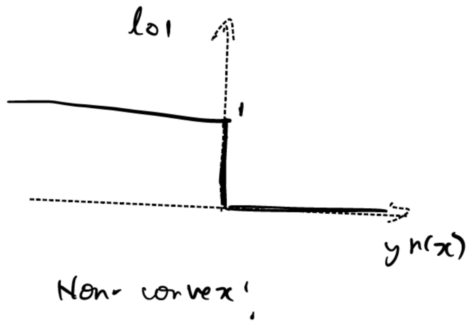
**Lemma 8.** *If $\ell$ is a convex loss (in terms of $w$), and $\mathcal{H}$ is convex, then $ERM_{\mathcal{H}}(\mathcal{S})$ is a convex optimization problem.*

A simple example is linear regression with the squared loss:

$$\begin{aligned}
&\text{Let } \mathcal{H} = \left\{ x \to \langle w, x\rangle, w \in \mathbb{R}^d \right\}, \\
&\ell(h,(x,y)) = (h(x) - y)^2, \\
&\ell(w,(x,y)) = (\langle w, x\rangle - y)^2, \\
&\mathcal{H} = \mathbb{R}^d \text{ (convex)}.
\end{aligned}$$

As an exercise, verify that $\ell(w,(x,y))$ is convex in terms of $w$. Therefore $\text{ERM}_{\mathcal{H}}(\mathcal{S})$ is a convex problem.
What happens when we consider classification, and the 0/1 loss?

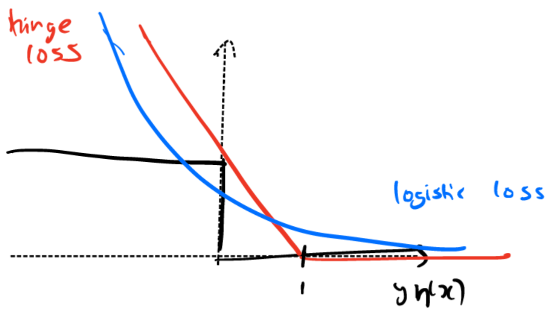
$$\ell_{01}(h,(x,y)) = \mathbb{1}(h(x) \neq y) = \mathbb{1}(yh(x) \leq 0).$$



Unfortunately, the 0-1 loss is non-convex. A common technique to handle a non-convex loss is to instead consider a *convex surrogate.*

## 4 Convex Surrogates

A convex surrogate is some loss function that we minimize instead of minimizing the 0-1 loss. The figure below shows some convex surrogates for the 0/1 loss.



$$\begin{aligned}
\text{Hinge loss: } &\ell(h,(x,y)) = \max(1 - yh(x), 0) \\
\text{Logistic loss: } &\ell(h,(x,y)) = \log(1 + e^{-yh(x)})
\end{aligned}$$

To find a good hypothesis from a training set, we can choose to minimize some convex surrogate instead of the 0-1 loss. However, for classification problems, the eventual goal is often to minimize the 0-1 loss, since it measures the classification errror. When is minimizing the convex surrogate effective in minimizing the original 0-1 loss?

Let $\mathcal{R}(h) = \mathbb{E}_{(x,y)\sim\mathcal{D}}[\mathbb{1}(yh(x) \leq 0)]$ be the expected 0/1 risk. Let $\phi : \mathbb{R} \to \mathbb{R}$ be some other function (used as a surrogate) and define $\mathcal{R}_\phi(h) = \mathbb{E}_{(x,y)\sim\mathcal{D}}[\phi(yh(x))]$ to be the *surrogate risk* (plugging in $\phi(yh(x))$ in place of $\mathbb{1}(yh(x) \leq 0)$ in $\mathcal{R}$).

Note that in this definition, functions which depend on $yh(x)$ can be used as a surrogate loss $\phi$. For e.g., the hinge loss and logistic loss fit this description.

Recall that $\mathcal{R}^* = \inf\limits_{f:\mathcal{X}\to\mathcal{Y}} \mathcal{R}(f)$ is the Bayes optimal risk.

Let $\mathcal{R}^*_\phi = \inf\limits_{f:\mathcal{X}\to\mathcal{Y}} \mathcal{R}_\phi(f)$ be the Bayes optimal $\phi$-risk.

**Definition 9.** *We say surrogate loss $\phi$ is classification calibrated if for any sequence of functions $f_i$ and every distribution $\mathcal{D}$ over $(x, y)$,*

$$\mathcal{R}_\phi(f_i) \to \mathcal{R}^*_\phi \implies \mathcal{R}(f_i) \to \mathcal{R}^*.$$

This says that if you can find a hypothesis which is Bayes optimal according to $\phi$-risk, then it will also be optimal according to the 0/1 loss, *i.e.* the surrogate is a good surrogate.

Note that this definition requires the surrogate risk to converge to the Bayes optimal surrogate risk, i.e. $\mathcal{R}_\phi(f_i) \to \mathcal{R}^*_\phi$ (Bayes optimal predictor for $\phi$-risk). Therefore it is only meaningful if we optimize our risk over some hypothesis class which includes the Bayes optimal predictor in terms of the $\phi$-risk.

The following theorem characterizes when a surrogate loss is classification calibrated.

**Theorem 10** ([4])**.** *Consider a surrogate loss $\phi(yh(x))$. If $\phi$ is convex, then it is classification calibrated if and only if $\phi'(0)$ exists and $\phi'(0) < 0$ (the derivatives are taken w.r.t. $yh(x)$).*

The following corollary is immediate from this theorem by taking derivatives.

**Corollary 11.** *Hinge loss and logistic loss are classification calibrated.*

Note that the squared loss $\ell(h; (x, y)) = (h(x) - y)^2$ doesn't fit the form of a function depending only on $yh(x)$, i.e. $\phi(yh(x)$. However, it is known that it is still classification calibrated.

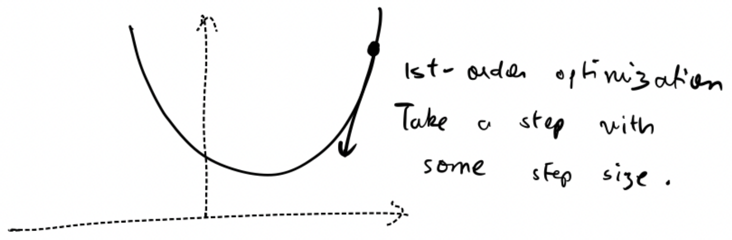
## 5 Gradient Descent (GD)

Let us go back to convex optimization problems, and look at algorithms for solving such problems. Consider an unconstrained optimization problem: $\min\limits_{x\in\mathbb{R}^d} f(x)$, where $f(x)$ is differentiable. Gradient descent is perhaps the most natural algorithm for solving the problem.

**Algorithm 3** Gradient Descent (GD)

```
Initialize $w_1$
for $t$=1,2,...,$T$ do
    $w_{t+1} \leftarrow w_t - \eta \nabla f(w_t)$
end for
```

Here, the output of the algorithm maybe the last iterate $w_T$, or the average iterate $\frac{1}{T}\sum_{t=1}^{T} w_t$. For any point, we are considering the first-order approximation of $f$ and taking a step in that direction with step size $\eta$. The figure below shows this for a 1-D problem.



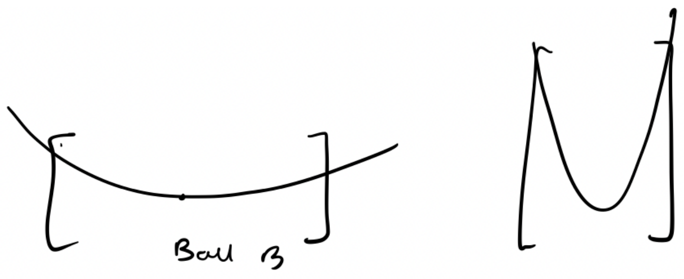
The following theorem proves the convergence of gradient descent for convex, Lipschitz problems.

**Theorem 12** (Convergence rate of GD). *Let $f$ be a convex, differentiable and $\rho$-Lipschitz function, i.e.*

$$|f(x) - f(y)| \le \rho\|x - y\| \;\forall\; x, y \in \mathbb{R}^d.$$

*Let $w^* = \underset{w\in\mathbb{R}^d}{\arg\min} f(w)$, $\|w^* - w_1\| \le B$, where $w_1$ is the initialization for GD. Suppose we run GD for $T$ steps with $\eta = \sqrt{\frac{B^2}{\rho^2 T}}$ and let $\bar{w} = \frac{1}{T}\sum_{t=1}^{T} w_t$. Then $\bar{w}$ satisfies $f(\bar{w}) - f(w^*) \le \frac{B\rho}{\sqrt{T}}$. Therefore if $T \ge \frac{B^2\rho^2}{\epsilon^2}$, then $f(\bar{w}) - f(w^*) \le \epsilon$.*

Note that the convergence rate depends on both $\rho$ and $B$. For a function which is flat, $\rho$ is small. When we initialize within a ball of radius $B$, the value on any point in this ball is not too different from the best value, so we don't need to take many steps to get small error, hence $T$ is small when $\rho$ is small. If the function is steep ($\rho$ is large) and we initialize in the same ball, we will need more steps to get small error, so $T$ would be large. These cases are shown in the figure below.



**Proof.** By using the definition of $\bar{w}$ and Jensen's,

$$\begin{aligned} f(\bar{w}) - f(w^*) &= f\left(\frac{1}{T}\sum_{t=1}^{T} w_t\right) - f(w^*) \\ &\leq \frac{1}{T}\sum_{t=1}^{T} f(w_t) - f(w^*) \\ &= \frac{1}{T}\sum_{t=1}^{T}(f(w_t) - f(w^*)). \end{aligned} \tag{1}$$

Because $f$ is convex, we have:

$$f(w_t) - f(w^*) \leq \langle w_t - w^*, \nabla f(w_t)\rangle. \tag{2}$$

Combining (1) and (2),

$$f(\bar{w}) - f(w^*) \leq \frac{1}{T}\sum_{t=1}^{T}\langle w_t - w^*, \nabla f(w_t)\rangle. \tag{3}$$

We do not need convexity for the rest of the proof. We will now upper bound $\sum_{t=1}^{T}\langle w_t - w^*, \nabla f(w_t)\rangle$.

**Lemma 13** (Iterative Update)**.** *Let $v_1, ..., v_T$ be an arbitrary sequence of vectors. Consider any algorithm with an update rule $w_{t+1} = w_t - \eta v_t$. Then,*

$$\sum_{t=1}^{T}\langle w_t - w^*, v_t\rangle \leq \frac{\|w^* - w_1\|^2}{2\eta} + \frac{\eta}{2}\sum_{t=1}^{T}\|v_t\|^2.$$

*If $\|w^* - w_1\| \leq B$, $\|v_t\| \leq \rho$, and $\eta = \sqrt{\frac{B^2}{\rho^2 T}}$,*

$$\sum_{t=1}^{T}\langle w_t - w^*, v_t\rangle \leq B\rho\sqrt{T}.$$

**Proof.** By completing the squares,

$$\begin{aligned} \langle w_t - w^*, v_t\rangle &= \frac{1}{\eta}\langle w_t - w^*, \eta v_t\rangle \\ &= \frac{1}{2\eta}(-\|w_t - w^* - \eta v_t\|^2 + \|w_t - w^*\|^2 + \eta^2\|v_t\|^2) \\ &= \frac{1}{2\eta}(-\|w_{t+1} - w^*\|^2 + \|w_t - w^*\|^2) + \frac{\eta}{2}\|v_t\|^2 \end{aligned}$$

By summing over all timesteps, we get a telescoping sum,

$$\begin{aligned}\sum_{t=1}^{T}\langle w_t - w^*, v_t\rangle &= \frac{1}{2\eta}\sum_{t=1}^{T}(-\|w_{t+1}-w^*\|^2 + \|w_t - w^*\|^2) + \sum_{t=1}^{T}\frac{\eta}{2}\|v_t\|^2 \\ &= \frac{1}{2\eta}(-\|w_{T+1}-w^*\|^2 + \|w_1 - w^*\|^2) + \frac{\eta}{2}\sum_{t=1}^{T}\|v_t\|^2 \qquad \text{(other terms cancel out due to telescopic sum)} \\ &\le \frac{1}{2\eta}\|w_1 - w^*\|^2 + \frac{\eta}{2}\sum_{t=1}^{T}\|v_t\|^2\end{aligned}$$

Now using the bounds on $\|w_1 - w^*\|^2$ and $\|v_t\|^2$,

$$\begin{aligned}\sum_{t=1}^{T}\langle w_t - w^*, v_t\rangle &\le \frac{B^2}{2\eta} + \frac{\eta}{2}T\rho^2 \\ \implies \frac{1}{T}\sum_{t=1}^{T}\langle w_t - w^*, v_t\rangle &\le \frac{B^2}{2\eta T} + \frac{\eta}{2}\rho^2.\end{aligned}$$

To minimize the bound, set both terms equal to get $\eta = \frac{B}{\rho\sqrt{T}}$, which gives

$$\frac{1}{T}\sum_{t=1}^{T}\langle w_t - w^*, v_t\rangle \le \frac{B\rho}{\sqrt{T}},$$

and completes the proof of the lemma. ■

As a final step, note that since $f$ is differentiable and $\rho$-Lipschitz, $\|\nabla f(x)\| \le \rho \; \forall \; x$. This completes the proof of the theorem. ■

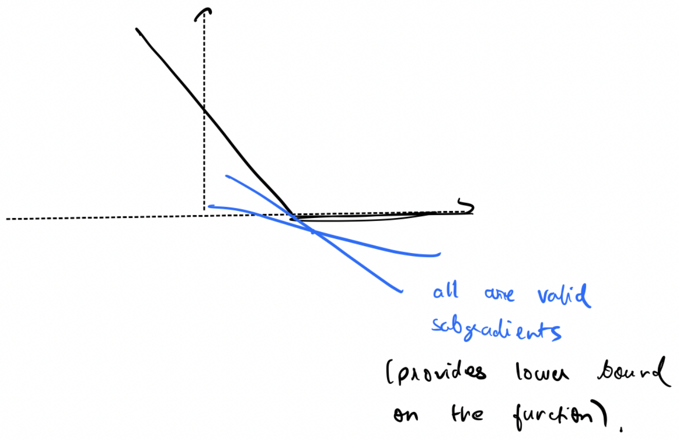
## 5.1 Variants of GD

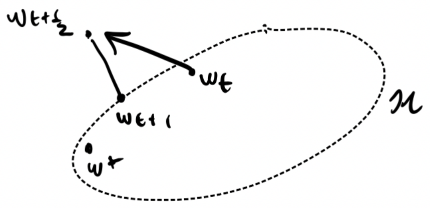
Various variations of gradient descent are known which work beyond the vanilla setup in the previous theorem.

• If $f$ is not differentiable, we can use *subgradient descent.* A subgradient $\partial f(x)$ is any vector which satisfies the first order definition of convexity, $f(y) - f(x) \le \langle y - x, \partial f(y)\rangle$.
For example, the hinge loss is not differentiable at 0 but there are many valid subgradients, as shown in the figure below.

• If we are doing constrained optimization, we can use *projected gradient descent (PGD).*
Suppose we are solving the following optimization problem: $\min_{x\in\mathcal{H}} f(x)$. Then PGD updates are given by:

$$\begin{aligned}w_{t+\frac{1}{2}} &= w_t - \eta\nabla f(w_t) \\ w_{t+1} &= \arg\min_{w\in\mathcal{H}}\|w - w_{t+\frac{1}{2}}\|.\end{aligned}$$


all are valid subgradients
(provides lower bound on the function).



The second step finds the projection of $w_{t+\frac{1}{2}}$ in the convex set $\mathcal{H}$ which is closest to it.

Projected GD is visualized in the figure above.

**Lemma 14.** *If $\mathcal{H}$ is convex, then for any $v \in \mathcal{H}$, $\|w_{t+1} - v\| \leq \|w_{t+\frac{1}{2}} - v\|$.*

**Proof.** Note that $v \in \mathcal{H}$ and projection finds a point $w_{t+1} \in \mathcal{H}$, closest to $w_{t+\frac{1}{2}}$. As $w_{t+\frac{1}{2}} \notin \mathcal{H}$ and $\mathcal{H}$ is convex, $w_{t+1}$ is closer to $v$. This Stack Overflow post has a more formal and geometric proof. ■

Using the above lemma, it is possible to show that PGD achieves the same convergence rate.

**Lemma 15.** *PGD has the same convergence rate as GD.*

The intuition for the proof is that the projection step never takes us farther away from the optimal point. Therefore, we can repeat the analysis for GD.

Note that the projection should be efficiently computable to do PGD. If $\mathcal{H}$ is a convex set, the projection can be computed efficiently.

## 5.2 Faster convergence under more assumptions

If we place more assumptions on the function than just convexity and Lipschitzness, then we can get faster convergence rates. One such assumption is *smoothness.*

**Definition 16** (Smoothness)**.** *We say a function $f$ is $\beta$-smooth if $\forall\ x, y \in$ domain$(f)$,*

$$\|\nabla f(x) - \nabla f(y)\| \leq \beta \|x - y\|.$$

Smoothness says that the gradient itself is a Lipschitz function, therefore the function cannot be too steep. This also implies that

$$f(y) \leq f(x) + \langle y - x, \nabla f(x) \rangle + \frac{\beta}{2}\|y - x\|^2.$$

This says that the function cannot change too quickly, and in praticular that the growth is upper bounded by some quadratic. Smoothness provides a faster convergence rate

**Theorem 17.** *Let $f$ be a convex function that is $\beta$-smooth. Then GD with step size $\eta = \frac{1}{\beta}$ satisfies*

$$f(w_T) - f(w^*) \leq \frac{B^2\beta}{2T}.$$

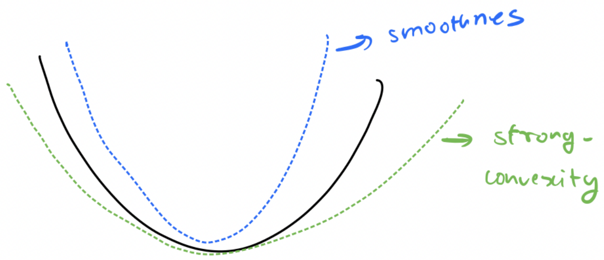
Therefore, GD finds a $\epsilon$-optimal point in $T \geq \frac{B^2\beta}{2\epsilon}$ iterations. Contrast this with the $O(1/\epsilon^2)$ rate that we got only with Lipschitzess. Also note that the convergence guarantee here is on the last iterate itself, and not the average iterate.

Another useful property is *strong convexity.*

**Definition 18** (Strong Convexity)**.** *A function $f$ is $\lambda$-strongly convex if its domain is convex and $\forall\ x, y$,*

$$f(y) \geq f(x) + \langle y - x, \nabla f(x) \rangle + \frac{\lambda}{2}\|y - x\|^2.$$

Strong convexity says that the function can be lower bounded by a quadratic. Smoothness and strong convexity give upper and lower bounds on $f$, respectively, as shown in the figure below. (This also implies that we must have $\beta > \lambda$.)



Together, strong convexity and smoothness yield an *exponentially* faster convergence rate.

**Theorem 19.** *Let $f$ be a convex function that is $\beta$-smooth and $\lambda$-strongly convex. Define condition number $\kappa = \frac{\beta}{\lambda} > 1$. Then GD with $\eta = \frac{2}{\beta + \lambda}$ satisfies*

$$f(w_T) - f(w^*) \leq \frac{B^2\beta}{2T} \exp\left(-T/\kappa\right).$$

In this case, $\epsilon$ error can be achieved in $T \geq \kappa \log\left(\frac{B^2\beta}{2\epsilon}\right)$ or $\mathcal{O}\left(\kappa \log\left(\frac{1}{\epsilon}\right)\right)$ iterations. This is known as a linear convergence rate in the literature, since the error reduces linearly with the number of iterations when we plot it on a log scale. We will discuss the smooth, strongly convex case more next time.

## 6 Further Reading

You can read the papers we mentioned to learn more about memory-sample tradeoffs for learning.

Our analysis of GD follows Chapter 14 in the book [5]. You can also read there for more details about subgradients and projected gradient descent. There are a number of good lectures notes online for surrogate losses if you want to learn more, such as this one.

## References

[1] Ran Raz. Fast learning requires good memory: A time-space lower bound for parity learning. *Journal of the ACM (JACM)*, 66(1):1–18, 2018.

[2] Sumegha Garg, Ran Raz, and Avishay Tal. Extractor-based time-space lower bounds for learning. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 990–1002, 2018.

[3] Jacob Steinhardt, Gregory Valiant, and Stefan Wager. Memory, communication, and statistical queries. In *Conference on Learning Theory*, pages 1490–1516. PMLR, 2016.

[4] Peter L Bartlett, Michael I Jordan, and Jon D McAuliffe. Convexity, classification, and risk bounds. *Journal of the American Statistical Association*, 101(473):138–156, 2006.

[5] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.