Contents

1 Review of Previous Class 1

2 An Algorithm for Bandit Convex Optimization 1
   2.1 Gradient Descent Without a Gradient . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
      2.1.1 Estimating the Gradient . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
      2.1.2 Applying the Gradient Estimator . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1

3 Boosting 3
   3.1 Is Weak Learning Equivalent to Strong Learning? . . . . . . . . . . . . . . . . . . . . . . . . . . 3
   3.2 AdaBoost . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
   3.3 What Happens In Practice? . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

1 Review of Previous Class

We previously talked about the Multi-Armed Bandit problem, and saw an algorithm for doing it - it was separate exploration, exploitation.

We also talked about the EXP3 algorithm, which has $O(\sqrt{Tn\log(n)})$ for $n$-bandits. The reason it gets the tight bound is because it explores and it exploits simultaneously. We also discussed online routing with bandit convex optimization.

Our talk today will focus on the following talks.

1. First, we will finish up talking about BCO.

2. Then, we will begin a discussion about boosting.

2 An Algorithm for Bandit Convex Optimization

2.1 Gradient Descent Without a Gradient

This algorithm is due to Flaxman, Kalai, and McMahan. This is an ingenious algorithm for generally solving BCO. I hope you will be surprised that something can be done in this difficult setting. Let me start by describing the idea. The core of the idea is that we are able to construct a gradient for a convex function even though we cannot see the gradient.

2.1.1 Estimating the Gradient

Choose $x_t \in \mathcal{K}$, and we see only one loss $f_t(x_t) \in \mathbb{R}$. For one dimension, let us recall the definition of $f'(x)$: It is $\lim_{\delta \to 0} \frac{f(x+\delta) - f(x-\delta)}{2\delta} = f'_\delta(x)$. Consider the following random variable $\tilde{V} = f(x + \delta)^{\frac{1}{2}}$ with probability $\frac{1}{2}$, and $f(x - \delta)^{\frac{1}{2}}$ with probability $\frac{1}{2}$. 

1
Then we have \(\mathbb{E}[\nabla] = f'_\delta(x) \sim f'(x)\). So now we want to generalize this to multiple dimensions. For \(f: \mathbb{R}^d \to \mathbb{R}\), we have \(\mathbb{E}_{\mathbf{v} \in S_2}[f(x + \delta \mathbf{v}) \cdot \mathbf{v}] = \frac{d}{\delta} \nabla f_\delta(x)\), where \(f_\delta(x) = \int_{u \in B} f(x + \delta u)du\). This equality follows from Stokes’ Theorem.

We can verify that \(\forall x |f_\delta(x) - f(x)| \leq \delta \cdot G\), where \(G\) is either the Lipschitz constant or an upper bound on \(\|\nabla f\|\).

Note that convexity is a property that is closed under taking averages. Therefore, \(f_\delta(x)\) is a convex function.

### 2.1.2 Applying the Gradient Estimator

Let us say we will use OGD, and wherever we need the gradient, replace it with our estimator. We have say

\[
y \leftarrow y_t - \eta \tilde{\nabla} f_t(y_t)
\]

The only issue may be if the sphere we are using to estimate the gradient overlaps with the boundary of the convex set. In this case, we may need to move the sphere a bit so that this does not occur.

**Definition 2.1. FKM algorithm.**

The algorithm is as follows. For \(t = 1, \ldots, T\):

1. Pick \(u \sim S\) uniformly at random.
2. \(x_t = y_t + u \cdot \delta\), play \(x_t\).
3. \(\tilde{\nabla}_t = \frac{d}{\delta} f(x_t) \cdot u\)
4. \(y_{t+1} \leftarrow \prod_{K_{\delta}} [y_t - \eta \tilde{\nabla}_t]\)

where \(K_\delta\) corresponds to the set of points shifted so that the ball lies within the set. We have explicitly \(K_\delta = \{y|y = (1 - \delta) \cdot x + \delta x_0 | x \in K\}\), where \(x_0\) is the center of \(K\) (it is actually true for any point that has some constant distance from the boundary). This is called the Minkowski cutoff.

Now we give the main theorem.

**Theorem 2.2.** \(\text{Regret}(FKM) = \sum_t f_t(x_t) - \min_{x^* \in K} \sum f_t(x^*) = \mathcal{O}\left( \frac{T^q}{d}\right)\) for some \(q, \delta\).

**Proof.** Denote \(x^*_\delta = \prod_{K_\delta} [x^*]\). Observe that \(f(x^*_\delta) - f(x^*) \leq G \cdot \delta\) for any \(G\)-Lipschitz \(f\). Also assume that \(K \subseteq B\).

Now let us analyze this algorithm.

\[
\mathbb{E}\left[\sum_t (f_t(x_t) - f_t(x^*))\right] \leq \mathbb{E}\left[\sum_t (f_t(x_t) - f_t(x^*_\delta))\right] + T\delta G
\]

\[
\leq \mathbb{E}\left[\sum_t f_t(y_t) - f_t(x^*_\delta)\right] + 2T\delta G \text{ because } d(x_t, y_t) \text{ is at most } \delta
\]

\[
\leq \mathbb{E}\left[\sum_t f_t,\delta(y_t) - f_t,\delta(x^*_\delta)\right] + 4GT\delta \text{ because you cannot be further from the farthest point}
\]

\[
\leq \left[\frac{D}{\eta} + \eta \sum |\tilde{\nabla}_t|^2\right] + 4GT\delta
\]

\[
\leq \frac{D}{\eta} + \eta \cdot T \frac{d^2}{\delta^2} + 4GT\delta = \mathcal{O}\left( \frac{T^q}{d}\right)
\]

by picking \(\delta = \frac{1}{T^q}\) and \(\eta = \frac{1}{T^{q/2}}\)

(1)
This algorithm is still the best algorithm known for BCO in general. The lower bound is $\sqrt{T}$, so this is not tight and no tight bounds are known. This may be a good (but hard) question to consider for the final project.

The question I ask in the homework is: can you use this to do online routing? You have to model online routing as BCO over the flow polytope. It starts off at some distribution of paths, we take a ball, where you sample around it, and so on. A very convoluted way of thinking about the problem, but gives you a really strong result. Furthermore this is efficient, nothing is exponential in the number of paths. The complexity is also polynomial in $d$ - this is an efficient model, we get around the problem of representing discretely.

**Remark 2.3.** BLO (Bandit Linear Optimization) is a special case, with regret $O(\sqrt{T})$. Where are we loose then? Well, we are scaling down by $\delta$ every iteration in order to fit a ball. If we replace balls by ellipsoid, we don’t lose this. We could also add self-concordant barrier functions from optimization.

Another place we lose is when we move from $x$ to $y$. In the linear case, there is no loss: in the linear case, if you average over the ball, $f_\delta = f$ (you don’t have an inequality). We could also add a regularization function. If you choose regularization smartly (corresponding to the ellipsoids). There is a way to choose the regularization to make the $\|\tilde{\nabla}t\|^2$ to be constant for the correct choice of norm - in effect, what we get is exploration and low variance for free.

**Remark 2.4.** For general BCO, we still have upper bounds that are $O(T^{3/4})$, and lower bound $\sqrt{T}$. That’s it for BCO, and I must say this is a very active area in online learning, many recent papers and so on. It’s one of the more difficult areas of online learning.

Here we are concluding the Bandit model, and in fact the online model in general. We will move on to Boosting.

### 3 Boosting

This is a relatively big step outside the learning we have seen so far. We had that online learning contains statistical learning - boosting contains statistical learning, but not online learning (though there is something called online boosting).

The main idea of boosting is very elegant and very intuitive. It is foundational work in machine learning. It was invented by Rob Schapire in his PhD thesis. He answered the question of whether boosting was possible, and gave an efficient algorithm (though it was not very practical). Later at Bell Labs, he with another person invented AdaBoost, which is both efficient and practical.

You can think of the following problem. Say you want to classify emails, you’re trying to come up with a filter. The model we started off talking about was taking a linear combination of words and picking a linear classification model. Boosting takes a different approach. It talks about taking ‘rules of thumb’ (a classifier that does a little better than average). For instance, if ’Viagra’ is in the email, reject it. Otherwise, flip a coin. This is called a weak learner. From these, can we construct a strong learning algorithm in the sense of statistical learning? The answer turns out to be yes.

Let us define the underlying concept of a rule of thumb in the context of learning. It weakens the definition of statistical learning in the following way:

**Definition 3.1.** Weak learnability. We have a hypothesis class $\mathcal{H}$ with respect to $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$, where $h \in \mathcal{H}$ is a function $h : \mathcal{X} \rightarrow \mathcal{Y}$. We sample $S$ from $\mathcal{D}$. If $\exists$ an algorithm such that seeing large enough $m(\gamma, \delta, \text{VC-dim}(\mathcal{H}))$ samples, for any $\delta, \gamma$, with probability $1 - \delta$, err$(\tilde{h}) \leq \frac{1}{2} - \gamma$, where $\tilde{h}$ is the hypothesis returned by the algorithm.

How does this differ from statistical learnability? There is only one difference: the error was less than $\epsilon$ instead of $\frac{1}{2} - \gamma$, and the number of samples was dependent on $\epsilon$. This case is known as strong learnability.

We can consider the set of all problems that are weakly learnable. Every strong learner is clearly weak as well, but does the direction go the other way? In other words, is weak learning equivalent to strong learning?
### 3.1 Is Weak Learning Equivalent to Strong Learning?

Assume there exists \( h \in \mathcal{H} \), where \( err(h) = 0 \) (i.e. realizability), because it is much easier. There is agnostic boosting too, however.

**Remark 3.2.** I should mention that weak learning is a lot easier to come up with than strong learning, and it is attractive for that reason. The algorithm that will eventually come out will not be equivalent to ERM.

What I will do will now be a bit non-standard. I will prove boosting using the techniques from online learning that we have learned so far. This is not the way Rob Schapire does it, but this will be a bit simpler to write on the board. The main idea is as follows: Use the weak learner to learn new things about the distribution, but not very well. We want to boost this to epsilon. We will change the distribution to focus on the hardest examples, to write on the board.

**Theorem 3.3.** **Reduction: Strong learning \( \rightarrow \) Weak learning.** Given \( \mathcal{H}, \mathcal{X}, \mathcal{Y}, \mathcal{D} \).

Assume \( \exists A^{WL}(\mathcal{D}, \delta) \) that returns \( h \in \mathcal{H} \) such that \( err_p(h) \leq \frac{1}{2} - \gamma \) with probability \( 1 - \delta \).

**Proof.** We will use regret minimization over the simplex as a black box.

Input: \( \mathcal{H}, \delta, \epsilon \) and a \( \gamma \)-weak learner \( A^{WL} \). Output: \( \tilde{h} \notin \mathcal{H} \), but \( \tilde{h} : \mathcal{X} \rightarrow \mathcal{Y} \) such that \( err_p(\tilde{h}) \leq \epsilon \).

Let \( S \) be samples from \( \mathcal{D} \), with \( |S| = m \). We will choose \( m \) later. Let \( p_1 \) be uniform on \( |S| = m \) examples. For \( t = 1, \cdots, T \):

1. Find \( h_t \leftarrow A^{WL}(p_t, \frac{\delta}{T}) \)
2. Define \( f_t \) over \( \Delta_m \): \( f_t(p) = r_t^p \), where \( r_t(i) = 1 \) if \( h_t(x_i) \neq y_i \), and 0 otherwise.
3. \( p_{t+1} \leftarrow \) update \( p_t \) according to algorithm \( B \), which is an OCO algorithm for \( \Delta_m \).
4. Return \( \tilde{h}(x) = \text{sgn} \left( \sum_{t=1}^T h_t(x) \right) \) (basically, take the majority vote).

**Theorem 3.4.** \( err_S(\tilde{h}) = 0 \) with probability \( 1 - \delta \).

**Proof.** \( err_{S,p}(h_t) = \sum_{i=1}^m p(i) \mathbf{1}_{h_t(x_i) \neq y_i} \).

Then \( r_t^p p_t = 1 - err_{S,p}(h_t) \).

\[
err_{S,p}(h_t) \leq \frac{1}{2} - \gamma \text{ with probability } 1 - \frac{\delta}{T}, \text{ by choice of } h_t. \text{ Then } P \{ r_t^p p_t \geq \frac{1}{2} + \gamma \} \geq 1 - \frac{\delta}{T}.
\]

Therefore, \( P \{ \frac{1}{T} \sum_i f_t(p_t) > \frac{1}{2} + \gamma \} \geq 1 - \delta \).

Let us look at \( p^* \), the uniform distribution over misclassified examples of \( \tilde{h} \). We will use regret with respect to the uniform distribution for our online learning algorithm.

At least half of the examples have to make a mistake if we get it wrong, therefore, \( \sum_t r_t^p p^* \leq \frac{T}{2} \).

Therefore the number of correct examples is at most \( \frac{T}{2} \). Now let us combine the two observations.

With probability \( 1 - \frac{\delta}{T}, \frac{1}{2} + \gamma \leq \frac{1}{T} \sum_i f_t(p_t) \leq \frac{1}{T} \sum_i f_t(p^*) + \frac{\text{Regret}}{T} \leq \frac{1}{2} + \frac{\text{Regret}}{T} \rightarrow \frac{1}{2} \) as \( T \rightarrow \infty \), so we get a contradiction (\( \frac{\text{Regret}}{T} < \gamma \)).

We assumed we had a uniform distribution over misclassified examples. The conclusion is that the set of misclassified examples is empty - therefore, there are no misclassified examples. So we have \( T \geq \frac{\text{Regret}}{\gamma} \sim \sqrt{T \log(m)} \), so \( T \sim \frac{\log(m)}{\gamma} \). So the number of iterations is dependent on how good the weak learner is. The better it is, the fewer the number of iterations.

\[ \square \]
The theorem says that after \( T = \frac{\log(m)}{\gamma^2} \) iterations, \( err_S(\tilde{h}) = 0 \). Does this imply that \( err_D(\tilde{h}) \leq \epsilon \)? This is not immediate, because \( \tilde{h} \) is not in \( \mathcal{H} \) for the fundamental theorem of learning theory to apply. The answer is the following:

If \( \text{VC-dim}(\mathcal{H}) = d \), and \( \tilde{h} \in \tilde{\mathcal{H}}_T \), \( \text{VC-dim}(\tilde{\mathcal{H}}_T) \leq T \cdot d \), where \( \tilde{\mathcal{H}}_T \) is the set of hypotheses that are sign functions from combinations of \( h \in \mathcal{H} \). Then we have that the complexity is \( \frac{T \cdot d}{\sqrt{m}} \log(\frac{1}{\delta}) \sim \frac{\log(m)}{\sqrt{m}} \frac{\log(\frac{1}{\delta})}{\sqrt{m}} \), and we are roughly ok for showing statistical learning.

This is a fairly easy proof of boosting, however the algorithm it produces is not the best.

## 3.2 AdaBoost

The AdaBoost algorithm is a special version of this scheme. The only thing we need to worry about in the general template is the algorithm \( B \). Suppose we say \( p_{t+1}(i) = \frac{p_t(i) \cdot e^{-\eta_r t(i)}}{\sum p_{t+1}(i)} \) - this is the multiplicative learning rule. We typically take \( \frac{1}{\eta} = \sqrt{\frac{\log(m)}{\gamma^2}} \). For AdaBoost, we take \( \eta_t = \frac{\gamma_t^2}{\sqrt{\log(m)}} \), where \( \gamma_t \) is the edge of weak-learner \( h_t \). This \( \gamma_t \) can change drastically from iteration to iteration, and you can take advantage of it. In theory it doesn’t help. AdaBoost assumes you know it - but you can compute it - you can go over all examples and see where you make an error. You have to do it anyways, since you have to compute the \( r_t \).

## 3.3 What Happens In Practice?

We have many algorithms that are not proved, but we know they are weak learners. Boosting is helpful in this case. Decision-tree algorithms are one approach that boosting uses. The hypothesis class of a decision tree is very useful. The main problem is computational - constructing a decision tree is a hard problem. Even though it is hard, there are all sorts of constructions of good decision trees for data. You can think of your data being vectors \( x \in \{0, 1\}^d \), \( y = \{0, 1\} \). The nodes of a decision tree will be certain data points (Is height greater than 6 feet?), and the leaves of the tree will be 0 or 1. How do you find the best tree? A tree could in principle be exponentially large. These trees can be extremely large, and have all sorts of criteria - for instance, which one partitions the data the best way. This is called the Information Game. I am not going to say any rigorous things about this, because there are no rigorous things to be said. No one knows. It is a great heuristic, but no one knows. There is a system called CAT45 tree.

People have used this for decades, but it didn’t work - of course not, we have no bounds. But when we apply boosting, it works beautifully. You combine a bunch of trees into a committee of trees, and you know if the individual trees are better than random (of course they are, anything is better than random) - you get a strong result. This is state of the art in many problems to do with text, including search. When people look at search and text classification - there are two leading things: SVM, SVM with kernels (we sort of talked about it in the online learning). The other thing people do is boosting with trees.