Contents

1 Optimization for Machine Learning, Elad Hazan

`http://www.cs.princeton.edu/~ehazan/tutorial/SimonsTutorial.htm`

The paradigm is as follows: we have a machine we want to train on inputs, like images to classify. The distribution is over vectors in $\mathbb{R}^n$, and the output is a label $b = f_\theta(a)$ where θ are the parameters. The behavior of the function is set by the parameters.

We care about training the machine *efficiently* and so that it *generalizes*.

We will cover

1. Learning as mathematical optimization
2. Regularization
3. Gradient descent++ (Frank-Wolfe, acceleration, variance reduction...)

1.1 Math optimization

The input is a function $f : K \to \mathbb{R}$ for $K \subseteq \mathbb{R}^d$. The output is a minimizer $x \in K$ such that $f(x) \leq f(y)$ for all $y \in K$.

How can we access f? We assume we can access values and derivatives. We don't have to work in the oracle model; sometimes we know the function (e.g., a polynomial). Even in the non-oracle model, the problem can easily be NP-hard.

Learning is the same as optimization over data (a.k.a. empirical risk minimization) of the function

$$\text{argmin}_{x \in \mathbb{R}^d} \frac{1}{m} \sum_{i=1}^{m} \ell_i(x, a_i, b_i) + R(x)$$

where m is the number of examples, (a, b) are pairs of (feature, label), d is the dimension, and x is the parameter. Training means fitting the parameters of the model.

For example, in linear classification, we try to find a hyperplane which separates points of one class from points of another class. Given a sample $S = \{(a_1, b_1), \ldots, (a_m, b_m)\}$ where $b_i \in \{\pm 1\}$, find a hyperplane (WLOG through the origin) minimizing the number of mistakes:

$$\text{argmin}_{\|x\| \leq 1} \left| \left\{ i : \text{sign}(x^T a_i) \neq b_i \right\} \right|.$$

We can put it in the form we had before

$$\text{argmin}_{\|x\| \leq 1} \frac{1}{m} \sum_{i=1}^{m} \ell(x, a, b), \quad \ell(x, a_i, b_i) = \begin{cases} 1, & x^T a \neq b \\ 0, & x^T a = b. \end{cases}$$

This is an example of how to convert a learning problem to an optimization problem. This simple problem is already NP-hard.

Why? The sum of sign functions can be any piecewise constant function (with finitely many pieces), which can be complicated.

Is there a local property that ensures global optimality? Yes, convexity.

Definition 1.1: A continuous function $f : \mathbb{R}^d \to \mathbb{R}$ is **convex** iff

$$f\left(\frac{1}{2}(x + y)\right) \leq \frac{1}{2} f(x) + \frac{1}{2} f(y),$$

or equivalently (for differentiable functions),

$$f(y) \geq f(x) + \nabla f(x)^T (y - x).$$

Informally the function looks like a smiley.

Similarly we can define convex sets.

Definition 1.2: A closed set K is **convex** iff $\frac{1}{2}(x + y) \in K$ whenever $x, y \in K$.

To make the linear (or kernel) classification problem tractable, we consider convex relaxations such as the following loss functions $\ell(x, a_i, b_i)$.

1. Ridge/linear regression $(x^T a_i - b_i)^2$
2. SVM (the most popular method a decade ago) $\max\{0, 1 - b_i x^T a_i\}$.
3. Logistic $\ln(1 + e^{-b_i x^T a_i})$.

We have cast learning as mathematical optimization and argued convexity is algorithmically important. Next we cover algorithms.

1.1.1 Gradient descent

The most naive method is gradient descent. It is a local algorithm where we move in the direction of steepest descent.

Algorithm 1.3 (Gradient descent)**:**

$$-[\nabla f(x)]_i := -\frac{\partial}{\partial x_i} f(x) \tag{1}$$

$$y_{t+1} \leftarrow x_t - \eta \nabla f(x_t) \tag{2}$$

$$x_{t+1} = \text{argmin}_{x \in K} |y_{t+1} - x|. \tag{3}$$

Note the second step is necessary if we are optimizing over a subset K of $\mathbb{R}^n$; we have to project back to the set.

Theorem 1.4. *For step size* $\eta = \frac{D}{G\sqrt{T}}$

$$f\left(\frac{1}{T}\sum_t x_t\right) \leq \min_{x^* \in K} f(x^*) + \frac{DG}{\sqrt{T}}$$

where G is upper bound on norm of gradients and D is the diameter of the constraint set:

$$\forall t, \quad \|\nabla f(x_t)\| \leq G \tag{4}$$

$$\forall x, y \in K, \quad \|x - y\| \leq D. \tag{5}$$

Proof. We make 2 observations.

$$|x^* - y_{t+1}|^2 = |x^* - x_t|^2 - 2\eta \nabla f(x_t)(x_t - x^*) + \eta^2 |\nabla f(x_t)|^2 \tag{6}$$

$$|x^* - x_{t+1}|^2 \leq |x^* - y_{t+1}|^2 \tag{7}$$

The second follows from the Pythagorean theorem because the angle made at x_{t+1} is obtuse.

Combining these results and telescoping.

$$|x^* - x_{t+1}|^2 \leq |x^* - x_t|^2 - 2\eta \nabla f(x_t)(x_t - x^*) + G^2 \tag{8}$$

$$f\left(\frac{1}{T}\sum_t x_t\right) - f(x^*) \leq \frac{1}{T}\sum_t \left[f\left(\sum_t x_t\right) - f(x^*)\right] \qquad \text{convexity} \tag{9}$$

$$\leq \frac{1}{T}\sum_t \nabla f(x_t)(x_t - x^*) \tag{10}$$

$$\leq \frac{1}{T}\sum_t \frac{1}{2\eta}(|x^* - x_{t+1}|^2 - |x^* - x_t|^2) + \frac{\eta}{2}G^2 \tag{11}$$

$$\leq \frac{1}{T2\eta}D^2 + \frac{\eta}{2}G^2 \leq \frac{DG}{\sqrt{T}} \tag{12}$$

with our choice of η. □

Note we showed that the average of the points converges. Under more conditions we can show the last point converges. There are examples in the stochastic setting where the average but not the last point converges.

Thus, to get ε-approximate solution, apply GD $O\left(\frac{1}{\varepsilon^2}\right)$ times.

1.1.2 Online/stochastic gradient descent and ERM

This is not suited to what we are looking for. For ERM problems we want

$$\text{argmin}_{x\in\mathbb{R}^d} \frac{1}{m}\sum_{i=1}^{m} \ell_i(x, a_i, b_i) + R(x)$$

The gradient depends on *all* data, which is inefficient, and we haven't considered generalization.

We consider simultaneous optimization and generalization. We have to recall that our data came from a distribution. We use the statistical (PAC) learning model.

- Nature chooses iid from a distribution D over $A \times B = \{(a, b)\}$.
- The learner chooses a hypothesis $h \in H$.
- There is a loss function ℓ, such as $\ell(h, (a, b)) = (h(a) - b)^2$.
- The error is $err(h) = \mathbb{E}_{a,b\sim D}[\ell(h, (a, b))]$.

Definition 1.5: We say that the hypothesis class H of functions $X \to Y$ if **learnable** if for all $\varepsilon, \delta > 0$ there exists an algorithm such that after seeing m examples, for $m = \text{poly}(\delta, \varepsilon, \dim(H))$, it finds h such that w.p. $1 - \delta$,

$$err(h) \le \min_{h^*\in H} err(h^*) + \varepsilon.$$

A more powerful setting is online learning in games.

1. Player picks $h_t \in H$.
2. Adversary chooses $(a_t, b_t) \in A$.
3. Loss function is ℓ.

There is no distribution. The aim is to minimize

$$\frac{1}{T}\left[\sum_t \ell(h_t, (a_t, b_t)) - \min_{h^*\in\mathcal{H}} \sum_t \ell(h^*, (a_t, b_t))\right].$$

We say that H is **learnable** in this setting if this quantity (the regret) approaches 0 as $T \to \infty$. This is not a priori clear that it can be done. Note your algorithm is allowed to change, and you compare to a fixed h^*. Vanishing regret in this setting implies generalization in the PAC setting; it is strictly more general.

From this point onwards, we consider $f_t(x) = \ell(x, a_t, b_t)$, the loss for one example.

Can we minimize regret efficiently?

There is a natural analogue of gradient descent, online gradient descent. The only difference is that we take a step with respect to the gradient of the *current* (possibly adversarially chosen!) function f_t.

Algorithm 1.6 (Online/stochastic gradient descent)**:**

$$y_{t+1} = x_t - \eta \nabla f_t(x_t) \tag{13}$$
$$x_{t+1} = \text{argmin}_{x \in K} |y_{t+1} - x|. \tag{14}$$

This may look strange at first glance: we take a step using f_t even though we may never see this function again!

(Notes: We assume we can always see the gradient. There are extensions that work even if you only see the loss, the bandit optimization problem.) .

Theorem 1.7 (Zinkevich)**.** *The regret of online gradient descent is*

$$\sum_t f_t(x_t) - \sum_t f_t(x^*) = O(\sqrt{T}).$$

The proof is similar to the previous proof. The only difference is that the gradient function is different at each step.

Proof. We make the same 2 observations.

$$|x^* - y_{t+1}|^2 = |x^* - x_t|^2 - 2\eta \underbrace{\nabla f_t(x_t)}_{:=\nabla_t}(x_t - x^*) + \eta^2 |\nabla f_t(x_t)|^2 \tag{15}$$
$$|x^* - x_{t+1}|^2 \leq |x^* - y_{t+1}|^2 \tag{16}$$

Combining these results and telescoping,

$$|x^* - x_{t+1}|^2 \leq |x^* - x_t|^2 - 2\eta \nabla_t (x_t - x^*) + \eta^2 \|\nabla_t\|^2 \tag{17}$$
$$f\left(\sum_t x_t\right) - f(x^*) \leq \sum_t \nabla_t (x_t - x^*) \tag{18}$$
$$\leq \sum_t \frac{1}{2\eta}(|x^* - x_{t+1}|^2 - |x^* - x_t|^2) + \frac{\eta}{2} \sum_t \|\nabla_t\|^2 \tag{19}$$
$$\leq \frac{1}{\eta} |x_1 - x^*|^2 + \eta T G \leq DG\sqrt{T} \tag{20}$$

□

This is tight. For the lower bound, take $K = [-1, 1]$, $f_1(x) = x$, $f_2(x) = -x$. The expected loss is 0, and the regret compared to either $-1, 1$ is on the order of the variance.

$$\mathbb{E}|\#1's - \#-1's| = \Omega(\sqrt{T}).$$

This gives rise to the most important problem in optimization for ML, stochastic gradient descent. The learning problem is

$$\text{argmin}_{x \in \mathbb{R}^d} F(x)$$

where

$$F(x) = \mathop{\mathbb{E}}_{(a_i, b_i)} [\ell(x, a_i, b_i)].$$

Use online gradient descent where at each step we take a random example $f_t(x) = \ell_i(x, a_i, b_i)$.

We have proved that

$$\frac{1}{T}\sum_t \nabla_t^T x_t \le \min_{x^* \in K} \frac{1}{T}\sum_t \nabla_t^T x^* + \frac{DG}{\sqrt{T}}.$$

Taking expectation we achieve the same bound as in gradient descent. If m is the number of examples, we've moved from $O\left(\frac{d}{\varepsilon^2}\right)$ rather than $O\left(\frac{md}{\varepsilon^2}\right)$ steps for ε generalization error with slight degradation because we only get a result in expectation.

$$\mathbb{E}\left[F\left(\frac{1}{T}\sum_t x_t\right) - \min_{x^* \in K} F(x^*)\right] \le \mathbb{E}\left(\frac{1}{T}\sum_t \nabla_t^T (x_t - x^*)\right) \le \frac{DG}{\sqrt{T}}.$$

(This is easier than the perceptron algorithm which requires finding a misclassified point at each step.)

1.2 Regularization

What is regularization and why do it? Statistical learning theory (Occam's razor) says that the number of examples needed to learn a hypothesis depends on the "dimension" which depending on the setting, could be

- VC dimension
- fat-shattering dimension
- Rademacher width
- margin/norm of linear/kernel classifier.

An expressing hypothesis class can overfit.

In PAC theory, regularization reduces complexity of the hypothessis.

In the regret minimization framework, regularization helps stability.

(Regularization also helps for the purpose of optimization—adding a convex function can distort the function to become convex. We don't consider this here. Note this may hurt generalization.)

We are trying to minimize regret (loss compared to best in hindsight). The most natural is to take

$$x_t = \text{argmin}_{x \in K} \sum_{i=1}^{t-1} f_i(x).$$

This doesn't work: consider the $\pm x$ example, when the adversary always chooses the opposite. This is called fictitious play in economics. This fails because of instability, the loss function can vary wildly each step.

This modification provably works (Kalai-Vempala 2005):

$$x_t' = \text{argmin}_{x \in K} \sum_{i=1}^{t} f_i(x) = x_{t+1}.$$

If $x_t \approx x_{t+1}$ we get a regret bound. However, the instability $|x_t - x_{t+1}|$ can be large.

We alter this further: pick the best point in hindsight, but add a term to make it stable.

Algorithm 1.8 (Follow The Regularized Leader)**:**

$$x_t = \text{argmin}_{x \in K} \sum_{i=1}^{t-1} \nabla_t^T x + \frac{1}{\eta} R(x)$$

where $R(x)$ is a strongly convex function.

Adding R ensures stability.

Theorem 1.9. *FTRL achieves regret*

$$\nabla_t^T (x_t - x_{t+1}) = O(\eta).$$

In economics this is called smooth fictitious play. There is a "center of mass" effect that pulls you towards a solution.

What to choose for R? The most obvious is $R(x) = \frac{1}{2} \|x\|^2$. For linear cost functions we recover gradient descent (π_K is projection to K)

$$x_t = \text{argmin}_{x \in K} \sum_{i=1}^{t-1} \nabla f_i(x)^T x + \frac{1}{\eta} R(x) = \pi_K(-\eta \sum_{i=1}^{t-1} \nabla f_i(x_i)).$$

There are many interesting algorithms you can recover by this paradigm, for example multiplicative weights. Take $f_t(x) = c_t^T x$ where c_t is vector of losses, $R(x) = \sum_i x_i \ln x_i$ the negative entropy. Then

$$x_t = \exp(-\eta \sum_{i=1}^{t-1} c_i)/Z_t$$

where Z_t is a normalization constant.

1.3 Gradient descent++

We cover AdaGrad, variance reduction, ande acceleration/momentum.

1.3.1 AdaGrad

What regularization should we choose? If we have a generalized linear model, then OGD update is inefficient if we have sparse data because it treats all coordinates the same way. Sparse data is common. Adaptive regularization copes with sparse data.

Which regularization to pick? The idea of AdaGrad is to treat choosing the R as a learning problem itself. Consider the family of regularizations

$$R(x) = \|x\|_A^2, \quad A \succeq 0, \quad \text{Tr}(A) = d.$$

This is finding the best regret in hindsight for a matrix optimization problem.

Algorithm 1.10 (AdaGrad)**:**

$$G_t = \text{diag}\left(\sum_{i=1}^{t} \nabla f_i(x) \nabla f_i(x_i)^T\right) \tag{21}$$

$$y_{t+1} = x_t - \eta G_t^{-\frac{1}{2}} \nabla f_t(x_t) \tag{22}$$

$$x_{t+1} = \text{argmin}_{x \in K} (y_{t+1} - x)^T G_t (y_{t+1} - x). \tag{23}$$

Theorem 1.11. *AdaGrad gives regret bound*

$$O\left(\sum_i \sqrt{\sum_t \nabla_{t,i}^2}\right).$$

This regret bound can be $\sqrt{d}$ better than SGD. The $\frac{1}{\sqrt{T}}$ in SGD was tight, but the constants can be improved by AdaGrad.

1.3.2 Variance reduction

Variance reduction uses special ERM structure and is very effective for smooth and convex functions.

Acceleration/momentum works for smooth convex functions only; it is used in general purpose optimization since the 80's.

Definition 1.12: If $0 \prec \alpha I \preceq \nabla^2 f(x) \preceq \beta I$, then the condition number if $\gamma = \frac{\beta}{\alpha}$. We say that f is α-strongly convex if the first inequality holds, β-smooth if the second inequality holds.

A convex function always satisfies this with some $\alpha \geq 0$. We can also talk about smoothness for non-convex functions:

$$-\beta I \preceq \nabla^2 f(x) \preceq \beta I.$$

Why do we care? Well-conditioned functions exhibit faster optimization; they can be optimized in polynomial time. Gradient descent is not polytime per se. Polytime means a bound logarithmical in approximation.

Smoothness is important in second-order methods

The loss function in ridge and logistic regression are strongly convex and smooth.

For smooth functions, a gradient step causes decrease in function value proportional to the value of the gradient. Taking $\eta = \frac{1}{2\beta}$,

$$f(x_{t+1}) - f(x_t) \leq -\nabla_t (x_{t+1} - x_t) + \beta |x_t - x_{t+1}|^2 \tag{24}$$

$$= -(\eta + \beta \eta^2) |\nabla_t|^2 = -\frac{1}{4\beta} |\nabla_t|^2. \tag{25}$$

Lemma 1.13 (Gradient descent lemma)**.** *For β-smooth functions, $f(x_{t+1}) - f(x_t) \leq -\frac{1}{4\beta} |\nabla_t|$.*

For M-bounded functions, what happens when we take many steps?

$$-2M \leq f(x_T) - f(x_1) \leq \sum_t [f(x_{t+1}) - f(x_t)] \leq -\frac{1}{4\beta} \sum_i |\nabla_i|^2.$$

There exists t for which

$$|\nabla_t|^2 \leq \frac{8M\beta}{T}.$$

1. Note we didn't use convexity. This is pretty much the only thing we can say for nonconvex functions.
2. Note for convex functions, we get a quadratic improvement: for $T = \Omega\left(\frac{1}{\varepsilon}\right)$ we get $|\nabla_t|^2 \leq \varepsilon$.

For nonconvex optimization, we can't hope for global optimality, but we can hope for local optimality (gradient vanishes).

This is nice but not practical for ML. We're taking full gradient steps; we need to go over the entire data set.

Let's look at the stochastic version. Take a step in direction $\widetilde{\nabla}_t$ where $\mathbb{E}\widetilde{\nabla}_t = \nabla_t$.

$$\begin{aligned}
\mathbb{E}[f(x_{t+1}) - f(x_t)] &\leq \mathbb{E}[-\nabla_t(x_{t+1} - x_t) + \beta|x_t - x_{t+1}|^2] && (26)\\
&\leq \mathbb{E}[-\widetilde{\nabla}_t \cdot \eta\nabla_t + \beta|\widetilde{\nabla}_t|^2] && (27)\\
&= -\eta\nabla_t^2 + \eta^2\beta\mathbb{E}|\widetilde{\nabla}_t|^2 && (28)\\
&= -\eta\nabla_t^2 + \eta^2\beta(\nabla_t^2 + \mathrm{Var}(\widetilde{\nabla}_t)). && (29)
\end{aligned}$$

Theorem 1.14. *For gradient descent for β-smooth, M-bounded functions, for $T = O\left(\frac{M\beta}{\varepsilon^2}\right)$, there exists $t \leq T$, $|\nabla_t|^2 \leq \varepsilon$.*

In practice, we take minibatches: take 10 or 100 examples instead of 1. This decreases variance, and is amenable to computer architecture.

In theory, tune step size according to these parameters. In practice, take a logarithmic scale, and test those step sizes.

Consider a hybrid model: sometimes compute the full gradient and sometimes take only the gradient of 1 example. This interpolates between GD and SGD.

Estimator combines both to create a random variable with lower variance.

Algorithm 1.15 (SVRG)**:**

$$x_{t+1} = x_t - \eta[\widetilde{\nabla}f(x_t) - \widetilde{\nabla}f(x_0) + \nabla f(x_0)].$$

Every so often, compute the full gradient and restart at new x_0.

Theorem 1.16 (Schmidt, LeRoux, Bach 12; Johnson, Zhang 13, Mahdavi, Zhang, Jin 13)**.** *Variance reduction for γ-well-conditioned functions produces an ε-approximate solution in*

$$O\left((m+\gamma)d\ln\left(\frac{1}{\varepsilon}\right)\right)$$

γ should be interpreted in $\frac{1}{\varepsilon}$ beacuse a naturally occuring function is not strongly convex (ex. hinge loss), but we can artificially add one with γ behaving like $\frac{1}{\varepsilon}$—this reduces the problem of optimizing a general convex function to optimizing a well-conditioned function.

There is a decoupling of m and $\frac{1}{\varepsilon}$ as compared to SGD.

1.3.3 Acceleration/momentum

This is a breakthrough by Nesterov, 1983. For certain optimization problems this gives the optimal number of steps. In practice it helps but not by much. Combining everything gives in theory the best known running time for first order methods,

$$O\left((m+\sqrt{\gamma m})d\ln\left(\frac{1}{\varepsilon}\right)\right).$$

This is tight (Woodworth, Srebro, 2015).

SVRG is in practice very effective for convex optimization.

Now we move from first-order to second order methods.

1.3.4 Second-order methods

Gradient descient moves in direction of steepest descent. It doesn't take into account the curvature of the function. One way to correct is to use local curvature; normalize the gradient according to the local norm.

Take a second-order approximation and optimize that. Think of GD as taking first-order Taylor approximation.

Algorithm 1.17 (Newton method)**:**

$$x_{t+1} = x_t - \eta[\nabla^2 f(x)]^{-1}\nabla f(x).$$

This is solving linear equations corresponding to the Taylor approximation of the function.

For non-convex function this can move to ∞. The solution is to solve a quadratic approximation in a local area (the trust region).

This is not used in practice because inversion takes d^3 time per iteration, but recently there have been advances.

To try to speed up the Newton direction computation:

- Spielman-Teng 2004: solve diagonally dominant systems of equations in linear time.
- Approximate by low-rank matrices and invert by Sherman-Morrison, etc. This is still d^2 time.
- Stochastic Newton (Linear-time second-order stochastic algorithm, LiSSA): Let $\widetilde{n} = [\nabla^2 f(x)]^{-1}\nabla f(x)$ be the Newton direction.

  Use the structure of the ML problem. For simplicity, suppose the loss is rank-1.

  $$\text{argmin}_x \mathop{\mathbb{E}}_i[\ell(x^T a_i, b_i) + \frac{1}{2}|x|^2].$$

  This is an unbiased estimator of the Hessian,

  $$\widetilde{\nabla^2} = a_i a_i^T \cdot \ell'(x^T a_i, b_i) + l, \quad i \sim U[1,\dots,m].$$

  Clearly $E[\widetilde{\nabla^2}] = \nabla^2 f$, but $\mathbb{E}[\widetilde{\nabla^2}^{-1}] \neq (\nabla^2 f)^{-1}$. It's not clear how to get an unbiased estimator of the Newton direction!

  We circumvent the Hessian estimation. 3 steps:

1. Represent Hessian inverse as infinite series $\nabla^{-2} = \sum_{i=0}^{\infty}(l - \nabla^2)^i$.
2. Sample from the infinite series (Hessian-gradient product), once:

$$[\nabla^2 f]^{-1}\nabla f = \mathbb{E}_{i\sim\mathbb{N}}(1 - \nabla^2 f)^i \nabla f \frac{1}{\mathbb{P}(i)}$$

   (The distribution depends on the condition number, ex. take the uniform distribution up to the condition number.)
3. Estimate the Hessian power by taking examples,

$$\mathop{\mathbb{E}}_{i\in\mathbb{N},k\sim[i]}\left[\prod_{k=1}^{i}(1 - \nabla^2 f_k)\nabla f \frac{1}{\mathbb{P}(i)}\right].$$

   This only uses vector-vector products.

We get unbiased estimate in linear time.

Algorithm 1.18 (LiSSA)**:** Use estimator $\widetilde{\nabla^{-2} f}\nabla f$ as above, where we compute full (or large batch) gradient ∇f. Move in the direction $\widetilde{\nabla^{-2} f}\nabla f$.

Theorem 1.19 (Agarwal, Bullins, Hazan 15)**.** *The running time is*

$$O\left(dm\ln\left(\frac{1}{\varepsilon}\right) + \sqrt{\gamma}d d\ln\left(\frac{1}{\varepsilon}\right)\right).$$

This is faster than first-order methods and seems to be tight (Arjevani, Shamir 16).

Can you do variance reduction with the Hessian estimator? This is an open question people are working on.

LiSSA works better BFGS methods.

For ML we cannot tolerate anything with running time superlinear.

1.3.5 Optimization with constraints

We talk about constrained optimization. One example is matrix completion. The (i, j) entry in the table is the rating of user i for movie j; we want to complete missing entries. Assume the true matrix is low-rank. The convex relaxation is bounded trace.

The trace norm is sum of singular values. The projection step is difficult, cubic time. Optimizing a linear function over this set is easy though.

Thus we do not want to project. To solve $\min_{x\in K} f(x)$, f smooth, convex, assuming linear optimization over K is easy, use

Algorithm 1.20 (Franke-Wolfe)**:**

$$v_t = \operatorname{argmin}_{x\in K} \nabla f(x_t)^T x \tag{30}$$

$$x_{t+1} = x_t + \eta_t(v_t - x_t). \tag{31}$$

This is same spirit as GD but different. This has been used a lot in stochastic optimization.

2 Submodularity and ML: Theory and Applications, Stefanie Jegelka and Andreas Krause

Consider a ground set V; let $F : 2^V \to \mathbb{R}$ be a function on the power set. Assume $F(\phi) = 0$ and we have a black-box oracle to evaluate F.

What does this have to do with ML?

- V are variables to observe, $F(S)$ is information obtained from observing them
- V is the seed nodes in a network, $F(S)$ is the spread of information
- V is collection of images, sentences, etc. $F(S)$ is a measure of representation (ex. how well they describe/summarize the data). This includes dictionary learning, matrix approximation, object detection...

In these examples, we want $\max_S F(S)$, where F could be coverage, spread, diversity, etc.

We could also want to do the opposite, maximize coherence, smoothness, $\min_S F(S)$.

- V are data points and $F(S)$ is coherence/separation.
- V is pixels in an image and $F(S)$ is coherence/matching (for picking out an object from an image)
- V are coordinates (variables) and $F(S)$ is coherence.

Many functions can be represented as optimizing a set function. We need some additional structure that makes this doable.

Convex functions

- occur in many models, and are often the only nontrivial property that can be stated in general.
- is preserved under many operations and transformations
- have sufficient structure for theory
- allow efficient minimization.

In the discrete world, submodular set-functions share the above four properties.

We'll define submodularity, and talk about minimization, maximization, and advanced topics.

2.1 What is submodularity?

Submodularity is defined by marginal/diminishing gains.

Definition 2.1: $F : 2^V \to \mathbb{R}$ is **submodular** if for all $A \subseteq B$ and $s \notin B$,

$$F(A \cup s) - F(A) \geq F(B \cup s) - F(B).$$

For example, the more sensors I have, the less information I gain from adding another one. Here we view F as a utility function; we can also view consider F as a cost function, in which it represents economies of scale. Ex. The more you buy, the less an extra item costs.

Another way to represent the submodular property is by the union-intersection property: for all $S, T \subseteq V$,

$$F(S) + F(T) \geq F(S \cup T) + F(S \cap T).$$

Where does this submodularity property actually come up?

1. A modular function is one such that $F(S) = \sum_{e \in S} w(e)$ for some weight function w on V. Here, for $e \notin A$,
$$F(A \cup e) - F(A) = w(e).$$
F is both submodular and supermodular.

2. Coverage function: For example, for V all possible sensor locations, F is the area covered by all sensors,
$$F(S) = \left| \bigcup_{v \in S} \text{area}(S) \right|.$$
Networks coverage is a stochastic version of coverage.

3. Mutual information: There is a random variable (ex. temperature) X_i at all locations. Observations at adjacent locations are not independent. We can observe Y_i, which are the variables X_i plus noise. Select some of these observations. How much do I reduce uncertainty about latent variables Y by observing some of the X's?

   The objective is the difference between the uncertainty about Y before sensing and the uncertainty about Y after sensing, which is the mutual information.

$$F(A) = H(Y) - H(Y|X_A) = I(Y; X_A).$$

4. Entropy: Consider rv's $X_1, \ldots, X_n$, $F(S) = H(X_S)$ the joint entropy of variables indexed by S. Entropy only shrinks if you condition on more variables,

$$H(A \cup e) - H(A) = H(X_e|X_A) \leq H(X_e|X_B) = H(B \cup e) - H(B)$$

   if $A \subseteq B$.

   If $X_i, i \in S$ are statistically independent, H is modular/linear on S. Submodular allows some degree of dependence.

5. Linear independence: V is a set of column vectors and $F(S)$ = rank of the matrix formed by columns in S.

6. Graph cuts $F(S) = \sum_{u \in S, v \notin S} w_{uv}$. The minimum cut problem tries to minimize this.

   Consider the cut function on the graph of 2 nodes u, v with an edge between them. Consider the union-intersection property. The only nontrivial inequality is the inequality

   $$F(\{u\}) + F(\{v\}) \geq F(\{u, v\}) + F(\phi).$$

   This is true because $2w_{u,v} \geq 0$.

   For an arbitrary graph we get a sum of functions like this, so graph cut is submodular.

7. Distributions can be log-submodular or log-supermodular, sub/supermodular function that is exponentiated.

   (a) A log-supermodular distribution satisfies $P(S) \propto \exp(-F(S))$. Equivalently,

   $$P(S)P(T) \leq P(S \cup T)P(S \cap T).$$

   This means positive associations are allowed. For example, ferromagnetic Ising model/conditional random field.

   An application is image segmentation. A set of pixels are more likely to be an object if they are positively correlated. Adjacent functions are more likely to take the same label than different labels; there is a penalty when there is a difference.

   What is the benefit of submodular functions here? Finding the mode of a supermodular distribution corresponds to minimizing a submodular function. We can approximate partition functions.

   (b) Log-submodular distributions have

   $$P(S)P(T) \geq P(S \cup T)P(S \cap T).$$

   Examples are determinantal point processes and volume sampling $P(S) \propto \text{Vol}(\{v_i\}_{i \in S})$, which prefers more linearly independent vectors. A determinantal point process has $P(S) \propto \det(L_S)$, the submatrix formed by rows and columns with indices in S.

Submodular functions arise in graph, game, matroid, information theory, stochastic processes, machine learning, information theory, and electrical networks.

Does submodularity correspond to discrete convexity or concavity? A bit of both.

- They are like convex functions because you can make it into a convex function (convex relaxation) and optimize that . There is a duality theory.

- On the other hand, diminishing returns corresponds to shrinking derivatives which corresponds to a concave function.

  For example, consider $F(S) = g(|S|)$. This is submodular iff g is concave. Taking this further, I could take $g(\sum_{i \in S} w_i) = g(\sum_i w_i x_i)$ where x is the indicator for S.

Stacking submodular functions we get a deep submodular function. It looks like a deep net! The function $F(x)$ defined by

$$z_l^1 = g_l^1(\sum_i w_{l,i}^1 x_i) \tag{32}$$

$$z_l^k = g_l^k(\sum_j w_{l,j}^k z_j^{k-1}) \tag{33}$$

$$F(S) = \sum_l z_l^K. \tag{34}$$

is submodular if the g_l^k are concave and increasing and weights are nonnegative. (Unlike in a neural net we are not optimizing over the w's but over the x's.)

More generally we can define submodular functions on lattices.

Definition 2.2: A **submodular function** on a lattice satisfies

$$f(x) + f(y) \geq f(x \vee y) + f(x \wedge y).$$

On a lattice, diminishing returns is stronger than submodularity.

Many optimization results generalize to this setting.

Let's look at computational problems.

2.2 Submodular minimization

How can we find $\min_{S \subseteq V} F(S)$? We use a relaxation,

$$\min_{x \in \{0,1\}^n} F(x) \to \min_{x \in [0,1]^n} f(x).$$

How do we do this? What function f interpolates F?

One natural thing to do is to define f as the expectation of a rv. How do do this in a way such that f has nice properties? Do threshold rounding.

Definition 2.3: The **Lovász extension** of F is

$$f(x) := \mathop{\mathbb{E}}_{\theta \sim x}[F(S_\theta)].$$

where $\theta \in [0,1]$ is sampled uniformly, and $S_\theta = \{e : x_e \geq \theta\}$.

For example, for $x = [0.5, 0.8]$,

$$\mathbb{P}(\{a,b\}) = 0.5 \tag{35}$$

$$\mathbb{P}(\{b\}) = 0.3 \tag{36}$$

$$\mathbb{P}(\phi) = 0.2, \tag{37}$$

then $f(x) = 0.5F(\{a,b\}) + 0.3F(\{b\})$.

(???) Two examples:

1. $F(S) = \max\{|S|, 1\}$. Then $f(x) = 0.8 \max_i x_i = \|x\|_\infty$. This is the l^∞ norm of the vector.
2. For $F(S) = \text{cut}(S)$, $f(x) = |x_a - x_b|$. This is the total variation function.

Theorem 2.4. *The Lovász extension is convex iff F is submodular.*

We prove that if F is submodular, than the Lovász extension is convex.

Proof sketch. If F is submodular, we can write it as a pointwise max of convex functions.

$$f(x) = \max_{y \in B_F} y^T x.$$

Here B_F is the base polytope.

Definition 2.5: The **submodular polyhedron** is

$$P_F := \left\{ y \in \mathbb{R}^n : \sum_{a \in A} y_a \leq F(A) \text{ for all } A \subseteq V \right\}.$$

The **base polytope** is

$$B_F = \left\{ y \in B_F : \sum_{a \in V} y_a = F(V) \right\}.$$

Note that there are an exponential number of inequalities defining P_F. □

Examples are

- probability simplex
- spanning tree polytope (convex hull of spanning tree indicator variables)
- permutahedron (convex hull of permutation matrices)

How can we do linear optimization over the base polytope? There are exponentially many constraints one for each subset.

But these polytopes are so nice that greedy algorithm works (Edmonds 1971).

Algorithm 2.6 (Greedy algorithm for linear optimization over base polytope)**:**

1. Sort cost vector $x_{\pi(1)} \geq x_{\pi(2)} \geq \cdots$.
2. This gives sets $S_i = \{\pi(1), \ldots, \pi(i)\}$.
3. Set $y_{\pi(i)} = F(S_i) - F(S_{i-1})$ (marginal gains).

We can do this optimization in $n \ln n$ time, the time to sort.

This implies we can compute Lovasz extension and subgradients of Lovasz extension.

A piecewise linear function is not differentiable at corner points, but there are many linear functions that lower bounds and touch at the point of discontinuity. These slopes are the subgradients. We can do gradient descent with subgradients instead. We have to be more careful with step sizes but it works.

Now we put things together. How do we go back to the discrete solution? (The solution could be fractional.)

Algorithm 2.7 (Submodular optimization)**:**

1. Relax to a convex optimization problem. Solve it using e.g. the ellipsoid algorithm.
2. The relaxation is exact. Pick elements with positive coordinates $S^* = \{e : x_e^* > 0\}$.

This solves submodular minimization in polynomial time.

There are different optimization algorithms we can apply.

- Ellipsoid method
- Subgradient method
- minimum-norm point/Fujishige-Wolfe algorithm

Another approach is combinatorial methods. Ex. Min-cut has a polytime combinatorial algorithm. Solve the dual of the minimization problem and use network flow algorithms.

The minimum-norm point/Fujishige-Wolfe algorithm uses a different relaxation,

$$\min_x f(x) + \frac{1}{2} \|x\|^2 .$$

where f is the Lovász extension. This solves a parametric series of problems

$$\min_{S \subseteq V} F(S) + \alpha |S|$$

for all α, in particular $\alpha = 1$. Thresholding at α gives the optimal solution x^* at α.

The dual problem is the minimum norm point of the base polytope

$$\min_{y \in B_F} \|y\|^2 .$$

It suffices to solve this.

Computing whether we are inside or outside the polytope, and hence projecting to it, is difficult. Instead, rely on the fact that we can do efficient linear optimization. Use the Frank-Wolfe algorithm.

At each step, find the best point along the segment joining the current point to the point that solves a linear program,

$$s^t \in \text{argmax}_{s \in B_f} \left\langle -\nabla g(y^t), s \right\rangle .$$

The min-norm point algorithm converges the fastest.

There are applications to structured sparsity, decomposition and parallel algorithms, variational inference...

For sparsity: Relax the subset selection problem using the Lovász extension. This is like going from l^0 to l^1.

2.3 Submodular maximization

This exploits concavity-like properties.

Now we want $\max F(S)$, often subject to some constraints.

Many such problems are motivated by optimal information gathering, like telling robots where to go to collect measurements, choosing a subset of variables to do experiments on,... Here $F(S)$ represents information.

Another application is data summarization. Select a subset of images that are a representative sample. This could be for exploratory data analysis. To train deep learning model on a large data set, what if we could just select a representative subset and train it on that subset.

Definition 2.8: A set function f is monotone if whenever $S \subseteq T$ then $F(S) \leq F(T)$.

A non-example is the graph-cut function.

Unconstrained maximization of monotone submodular functions is trivial: take the entire set. It makes sense to consider constraints like cardinality $|S| \leq k$.

This is NP-hard so we look for approximation algorithms. The simplest algorithm is greedy.

Algorithm 2.9 (Greedy algorithm)**:** Let $S_0 = \phi$. For $i = 0, \ldots, k-1$,

$$e^* = \text{argmax}_{e \in V \setminus S_i} F(S_i \cup \{e\}) \tag{38}$$

$$S_{i+1} = S_i \cup \{e^*\}. \tag{39}$$

This looks like a discrete analogue of gradient descent.

In practice the greedy algorithm does close to optimal. One can prove the following.

Theorem 2.10 (Nemhauser, Wolsey, Fisher 78)**.** *Let F be (nonnegative) monotone submodular, S_k be the solution of the greedy algorithm. Then*

$$F(S_k) \geq \left(1 - \frac{1}{e}\right) F(S^*).$$

In general, no polytime algorithm can do better.

Proof. Consider $F(S_l)$ as a function of l. (Side note: this is concave from submodularity.) Look at the gaps at each step of the algorithm $\Delta_i = OPT_k - F(S_i)$. The key lemma is the rate equation.

Lemma 2.11 (Rate equation)**.**

$$\max_e F(S_i \cup \{e\}) - F(S_i) \geq \frac{1}{k}\Delta_i.$$

This relates the marginal gain at the ith step to the gap.
This implies $\Delta_{i+1} \leq \left(1 - \frac{1}{k}\right)\Delta_i$. Recursively,

$$\Delta_l \leq \left(1 - \frac{1}{k}\right)^l OPT_k.$$

and so

$$F(S_l) \geq (1 - e^{-l/k})OPT_k.$$

□

This is tight: You can match this with cover functions.

2.3.1 Greedy++ algorithms

Now we talk about "greedy++" algorithms.

1. What if I have more complex constraints?
2. Greedy algorithms take time $O(nk)$. What if n, k are large?
3. What if the function is not monotone?

2.3.2 Complex constraints

A more complex constraint is a budget,

$$\sum_{e \in S} c(e) \leq B.$$

For example, it's more expensive to place sensors at certain locations, we have to summarize in 140 characters...
We can

1. run the greedy algorithm, or
2. run a modified greedy algorithm using the benefit-cost ratio.

$$e^* = \text{argmax}_e \frac{F(S_i \cup \{e\}) - F(S_i)}{c(e)}.$$

We can construct instances where either does arbitrarily badly, but if we pick the better of the two outputs, we can always get an approximation factor of $1 - \frac{1}{\sqrt{e}}$.
There are algorithms that are more general and achieve $1 - \frac{1}{e}$.
We relax from a discrete to a continuous function.

$$\max_{S \in I} F(S) \to \max_{x \in \text{conv}(I)} f_M(x).$$

We will round to get back the discrete solution.

The first attempt is to use the Lovász extension: But it is convex, and we can't maximize it.

Instead, use the multilinear extension: sample an item e with probability x_e (each is Bernoulli random variable)

$$f_M(x) = \mathop{\mathbb{E}}_{S \sim x}[F(S)] \tag{40}$$

This is neither convex nor concave (it is convex in some directions, and concave in others).

Unlike the Lovász extension, the multilinear extension can in general only be approximated by sampling so is slower unless you have special structure.

Use the continuous greedy algorithm on f_M.

The feasible set is some polytope. Do a Frank-Wolfe-like algorithm.

Algorithm 2.12 (Continuous greedy algorithm)**:** Start at 0. Look at the gradient of the multilinear extension at that point. Solve the linear program in the direction of the gradient. Increment by a step side in that direction.

$$v_{t+1} = \text{argmax}_{x \in P}\, x^T g_t \tag{41}$$

$$x_{t+1} = x_t + \gamma v_{t+1}. \tag{42}$$

Take $\gamma = \frac{1}{T}$.

(N.B. it is v_{t+1}, not $v_{t+1} - x_t$: move along the segment parallel to the segment joining the origin to v_{t+1}.)

This requires the polytope to be a downward closed polytope: for $x \in P$, $[0, x_1] \times \cdots \times [0, x_n] \subseteq P$.

The continuous greedy algorithm always exploits concave directions.

This also works for the more general class of monotone continuous DR-submodular functions, which could be nonconvex functions!

We need to do rounding (omitted).

2.3.3 Faster algorithms

How can we leverage parallelism? A natural idea is to parallelize the greedy selection. This requires communication after every element has been selected.

An idea is to partition the dataset, find the solution for each, and then merge the solutions together, and run greedy again. Each could pick $\frac{k}{m}$. This doesn't work well.

A simple modification: each selects a feasible solution of size k, merge to get a solution of size km, and then run the greedy algorithm on that set. Without more assumptions, this doesn't do well, giving $\frac{1}{\min\{\sqrt{k}, m\}}$ approximation.

Instead consider the best among the $m + 1$ sets: the final greedy solution and one of the sets chosen in the first step. Then if the partition is random, in expectation we get a $\frac{1}{2}\left(1 - \frac{1}{e}\right)$ approximation.

There is also work on accelerating the sequential algorithm, filtering/streaming/multi-stage algorithms, and distributed algorithms.

2.3.4 Non-monotone maximization

Non-monotone maximization is generally inapproximable unless F is nonnegative—it is NP-hard even to know the sign of the optimal solution.

For unconstrained maximization, the double greedy algorithm gives the optimal $\frac{1}{2}$ approximation.

For constrained maximization: for cardinality constraints, use the randomized greedy algorithm.

2.4 Advanced topics

2.4.1 Semigradient methods

We want to optimize $F(S)$ with some constraints. In some cases, this is very hard for submodular F, but easy or well approximable for modular $F(S) = \sum_{a \in S} w_a$.

Examples: solving modular optimization problems over trees, matchings, cuts, paths is tractable. Replacing by submodular functions, the resulting problem is difficult. So we approximate the submodular function by a modular function.

Start with initial guess S, and repeat: approximate F by modular function F, and optimize that.

Similar to convex functions, submodular functions have subdifferentials (Fujishige). A subdifferential at X satisfies

$$m(S) \leq F(S) \text{ for all } S \subseteq V \text{ and } m(X) = F(X).$$

They also have superdifferentials (Iyer, Jegelka, Bilmes).

The semigradients (sub/superdifferentials) have a polyhedral structure.

You can recover results for SFMax (submodular function maximization) if you choose the subgradients suitably. However, this generalization allows us to handle more complex constraints. Guarantees depend on the **curvature** κ, which measures how far the function is from modular:

$$\kappa = 1 - \min_{j \in V} \frac{F(V) - F(V \backslash \{j\})}{F(\{j\})}.$$

(Compare the smallest possible gain from adding j, with the value of j alone.) Guarantees are on the order of $\frac{1}{1-\kappa}$.

There are nice applications in computer vision: segmentation, informative path planning.

Can we do inference in the resulting distributions $P(S) = \frac{1}{Z} \exp(\pm F(S))$? (For log-sub/supermodular distributions the signs are $+/-$, respectively.) The key challenge is to compute the normalizing constant $Z = \sum_S \exp(\pm F(S))$. This is #P-hard.

The gradient perspective is useful here. The idea is variational inference. Elements from the sub/superdifferentials bound F,

$$x(A) \leq F(A) \leq y(A).$$

Compute the partition function for the upper and lower bounds,

$$\sum_{A \subseteq V} \exp(x(A)) \leq \sum_{A \subseteq V} \exp(F(A)) \leq \sum_{A \subseteq V} \exp(y(A))$$

with the inequalities reversed if we replace F by $-F$, y by $-y$. The upper and lower bounds break into products. We can optimize over these upper and lower bounds by exploiting structure of the sub/superdifferentials.

An application is semantic segmentation. Solving submodular optimization finds the mode, the MAP. We can also compute the marginal entropy for each pixel.

2.4.2 Interactive optimization

We generalize subset selection problems to adaptive problems. Suppose we are a vet treating a puppie. Depending on the heart rate, we might take a ECG, where we might take a blood sample or take a fMRI, etc.

The setting: Pick an element, observe something about that element, and then depending on that element, decide which next one to pick. Is there a notion of submodularity for sequential decision tasks?

Given

- items $V = [n]$
- random variables X_i, $i \in V$ taking values in O.
- objective $f : 2^V \times O^V \to \mathbb{R}$.

We want a policy π that maps observation x_A to next item. The value is

$$F(\pi) = \sum_{x_V} P(x_V) f(\pi(x_V), x_V).$$

The policy could take exponential space to write, so we can restrict to e.g. trees of size k. Are there sufficient conditions for greedy algorithm to work? Generalize the notion of marginal gain, the conditional expected benefit of adding item s,

$$\Delta(s|x_A) = \mathbb{E}\left[f(A \cup \{s\}, x_V) - f(A, x_V)|x_A\right].$$

What's the natural greedy algorithm? The adaptive greedy policy.

Definition 2.13: The value function satisfies **adaptive monotonicity** and **adaptive submodularity** if

$$\Delta(s|x_A) \geq 0$$

and

$$\Delta(s|x_A) \geq \Delta(s|x_B) \text{ for } x_A \preceq x_B,$$

respectively.

Submodularity says it's always better to take an action earlier than later.

We also get $1 - \frac{1}{e}$ approximation.

People also try to use submodular optimization to solve non-submodular problems, like applying convex methods to nonconvex problems.