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.

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

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

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

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

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

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

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

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.

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

(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.

3 Reinforcement learning, Emma Brunskill

In reinforcement learning, an agent interacts with the environment, and gets back a reward and next state which is uses to determine what the next action it takes will be. It learns to maximize expected reward.

How the world (stochastic environment) works is initially unknown.

Why care about RL?

- Learning to make good sequences of decisions under uncertainty is a critical part of autonomy and intelligence.
- There are many applications: robotics, consumer modeling, education (tutoring)...

Why is RL different from ML/AI planning? Standard AI planning assumes we know how the world works. We also think about sequences of actions, but because we don't know how the world works, there are more issues we have to tackle.

We have to deal with

1. generalization: The number of states could be enormous.
2. exploration: how to gather data.
3. delayed consequences: the decisions we make now could have consequences a lot later.

RL has been separate from the active learningn community, but I think there's a lot of overlap.

3.1 Standard RL setting

Most work has focused on Markov decision processes

Definition 3.1: A **Markov decision process (MDP)** has

- set of states S
- set of actions A
- stochastic transition/dynamics model $T(s, a, s')$, the probability of reaching s' after action a in state s
- reward model $R(s, a)$ (or $R(s)$ or $R(s, a, s')$)—random variable depending on state and action
- γ discount factor: how much to value immediate reward vs. future reward

A policy for a MDP is a function $\pi : S \to A$.

The γ-discounted reward is $R = \sum_{t=0}^{\infty} \gamma^t R_t$ where R_t is the reward at the next step. The Q-function $Q : S \times A \to \mathbb{R}$ for a policy π is

$$Q^\pi(x, a) = \mathbb{E}[R | s_0 = x, a_0 = a, \pi \text{ is followed after the first step}] \tag{43}$$
$$Q^*(x, a) = \max_\pi (Q^\pi(x, a)) \tag{44}$$

The Bellman equation is

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q^*(s', a').$$

The current state is sufficient to make the next decision.

MDPs may sound like a restricted model but you can always cheat by putting the whole history into the state space.

We can use the Bellman equation as an iterative algorithm.

Algorithm 3.2 (Value iteration)**:**

$$Q_0(s, a) = 0 \tag{45}$$
$$Q_{i+1}(s, a) = R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q_t(s', a') \tag{46}$$

The update is a contraction so it converges in the tabular case (where we keep track of every value $Q(s, a)$). We can recover the best policy from the Q-function by taking $\text{argmax}_a Q(s, a)$.

Algorithm 3.3 (Policy iteration)**:**

$$Q_0(s, a) = 0 \tag{47}$$
$$\pi_t(s) = \text{argmax}_a Q_t(s, a) \tag{48}$$
$$Q_{t+1}(s, a) = Q^{\pi_t}(s, a). \tag{49}$$

We can break all of RL into 3 different camps: value function, policy, model. There are algorithms combine these different approaches.

In model-based RL, use the experience to estimate model T and R, ex. with ML estimate of model parameters given observed (s_t, a_t, r_t, s_{t+1}) tuples. Use estimated models to estimate Q and Q to estimate π.

This uses data efficiently but is computationally expensive.

(Is this robust? If the models are close the Q-values are close. Do the errors compound? Yes.)

An alternative, more feasible approach is Q-learning, which is model free.

Algorithm 3.4 (Q-learning)**:** Initialize $Q(s, a)$ for all (s, a) pairs. On observing (s_t, a_t, r_t, s_{t+1}),

- Calculate TD-error

$$\delta(s_t, a_t) = r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t).$$

- Update
$$Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha\delta(s_t, a_t).$$

Here we're not picking the action.

This is computationally cheap. This only propagates experience one step, but this can be fixed by e.g. replay.

We can use all our information and recompute from scratch or just do local updates. Local updates are more computationally efficient but less sample efficient.

In policy search, directly search the policy space for $\text{argmax}_\pi V^\pi$. Parameterize policy and do stochastic gradient descent.

3.2 Exploration

We start off not knowing how the (stochastic) world works. Given some experience (data), we can estimate how the world works, or the Q-values or policies.

But the information that we've seen so far can be misleading, or we haven't seen everything. In particular, we may want to explore: try things that aren't seen to be optimal yet. Eventually, though, we want to exploit: gather high reward.

The agent is responsible for gathering data to learn how the world works.

Key ideas in exploration vs. exploitation algorithms are:

1. Act greedily mostly, sometimes randomly.
2. Be optimistic in the face of uncertainty.
3. Maintain a distribution over worlds.
   (a) Sample a world and act if the sample was the real world, or
   (b) Choose action reasoning about full distribution

Each gives a different algorithm.

3.3 Multi-arm bandit

Let's first cover a simpler model, the multi-arm bandit. There is no state.

Definition 3.5: The **multi-arm bandit** problem consists of the following.

- A is a set of arms
- R_a is unknown probability distribution of reward r if you pull arm a
- At each step t, the agent selects an arm a and gets $r_t \sim R_a$.
- The goal is to maximize cumulative reward over t pulls.

If we knew the best arm we could just choose that—but we don't.

Algorithm 3.6 (ε-greedy exploration)**:**

- Given t observations so far, estimate the mean reward for each action $\widetilde{\mu}_a$.
- W.p. $1-\varepsilon$ select $\operatorname{argmax}_a \widetilde{\mu}_a$.
- W.p. ε select random arm.

This only assumes rewards are bounded.

The second idea is optimism under uncertainty.

Algorithm 3.7 (UCB)**:** Estimate upper confidence bound on value (using concentration inequalities). Using Hoeffding,

$$\overline{\mu}_a = \widetilde{\mu}_a + \sqrt{\frac{2\ln t}{N_t(a)}}.$$

where $N_t(a)$ is the number of times a was pulled. Select $\operatorname{argmax}_a \overline{\mu}_a$.

(In the beginning pull everything once. You can't use this algorithm for huge action spaces.)

Either the arm you pull is reall the best arm and you get large reward, or you get information that allows you to revise your estimate.

We could be Bayesian: have a distribution over worlds. Maintain posterior distribution over reward distribution.

One approach to this is probability matching/Thompson sampling.

Choose arm with probability it is optimal

$$\pi(a|h_t) = \mathbb{P}(\mu_a > \mu_{a'}, \forall a' \neq a).$$

This may be hard to compute analytically, but we can estimate it by sampling.

Algorithm 3.8 (Thompson sampling)**:**

$$\pi(a|h_t) = \mathbb{P}(\mu_a > \mu_{a'}, \forall a' \neq a) = \mathop{\mathbb{E}}_{p(R|h_t)}[a - \operatorname{argmax}_a \mu_a].$$

?? Sample a reward distribution given posterior, compute the mean given the sampled R, and select the action with highest mean for that sample.

This works as long as it is cheap to represent the posterior and we can update it easily. Often use the conjugate distribution: if the reward is Bernoulli, use a beta distribution for the prior/posterior.

What is the Bayes-optimal algorithm, that gets the most reward achievable? Suppose we act for H steps. We want to select the arm to maximize cumulative expected reward over H steps given the prior. This directly reasons about the value of exploration and information. This is the best we can hope to do; it optimally balances exploration and exploitation.

This view transforms a learning problem into a planning (sequential decision-making) problem.

The hidden/latent state is static; it is parameters of the reward distribution. There is a posterior/belief state that is a probability over arm distribution parameters given the history.

Thus, an optimal policy for POMDP planning gives a Bayes-optimal solution for bandit learning. However, it is generally computationally intractable to do POMDP planning for continuous states exactly.

We can use sparse sampling of Monte Carlo Tree Search to approximately plan continuous-state MDP or POMDP. But the algorithms don't have provable guarantees.

For RL, the same ideas apply, except we compute Q-values rather than the total rewards.

For optimism in the face of uncertainty, compute upper confidence bound over Q values, and given state s, select $\text{argmax}_a\, Q(s,a)$. (UCRL algorithm)

To maintain a distribution over worlds, we have to approximate.

3.4 Evaluating an RL algorithm

Which should be pick?

When I say performance, I'm talking about the rewards, or the amount of data required to find the (near) optimal policy.

Other people have been focused on computational efficiency.

We can evaluate performance by empirical, convergence, asymptotic optimality, in the probability approximately correct model, regret, Bayes-optimality, and by comparing within a class of algorithms. Often algorithms with theoretical guarantees don't do so well empirically; we can tune constants of other a;gorithms to do better.

Convergence: do we converge to a single estimate of the Q-function? Depending on the domain, this is often nontrivial; Q-function estimate may end up oscillating.

Asymptotic optimality:

- Greedy in the limit of infinite exploration (decrease rate of exporation): Q-learning converges to Q^*. (Watkins and Dayan 1992)
- In the limit of infinite exploration model-based RL, we also converge to Q^* (Littman 1996)

This is unsatisfactory because it is an asymptotic result.

Definition 3.9: RL algorithm A is **PAC-MDP** if on all but N steps the algorithm's non-stationary policy A_t satisfies

$$V^{A_t}(s_t) \geq V^*(s_t) - \varepsilon$$

with probability $1-\delta$ where $N = \text{poly}(|S|, |A|, \frac{1}{\varepsilon}, \frac{1}{\delta}, \frac{1}{1-\gamma})$.

γ is the discount factor; think of $\frac{1}{1-\gamma}$ as the horizon.

A popular approach for establishing algorithm is PAC RL.

RL algorithm is greedy if it maintains a Q-value estimate Q_t and selects actions by $\text{argmax}_a\, Q_t(s,a)$.

Define a known MDP M_K related to the real MDP M: Given an input Q, K a set of known (s,a) pairs,

- for all $s, a \in K$, use the true transition and reward model M parameters,
- for all $s, a \notin K$, use self-loop and set reward to input $Q(s, a)$.

This is like optimism under uncertainty: imagine that places that are unknown have good reward.

Let $A(\varepsilon, \delta)$ be any greedfy learning RL algorithm such that with probability $\geq 1 - \delta$,

- accuracy $V_t^{\pi_t}(s_t) - V_{M_{K_t}}^{\pi_t}(s_t) \leq \varepsilon$
- Optimisim $Q(s, a) \geq Q^*(s, a) - \varepsilon$
- bounded learning complexity: total number of updates to Q estimates and visites to unknown (s, a) is bounded in terms of...

A popular approach for establishing algorithm is PAC RL: Divide horizon into epsidoes, during each episode either don't update Q¡ or update Q and visit unknown (s, a) pair, use threashold to set pair as known. Pig principle ensures tha we con only visit (s, a) a finite number of times until it becomes known.

How do we ensure...

- accuracy? Use the simulation lemma: if two MDPs are close in parameters in max norm, then thei value of a policy will be close in max norm.
- optimism? Combine simulation lemma and use opt estimate of rewards for all unknown (s, a) pairs.

PAC is good because it has finite sample complexity: we bound the total number of mistakes with high probability. It's bad because experimentally, we end up exploring much longer.

Early PAC bounds are like $N = \widetilde{O}\left(\frac{|S|^2|A|}{\varepsilon^3}\right)(1-\gamma)^6$. For $|S| = 10, |A| = 10, \varepsilon = 0.1, \gamma = 0.9$, we need $N = 10^{12}$ samples.

In the episodic case (act for H steps, and then reset) we have a bound tight up to $|S|$. The upper bound is $N = \widetilde{O}\left(\frac{|S|^2|A|H^3}{\varepsilon^2}\right)$ (δ hidden in log term) and the lower bound is $N = \widetilde{O}\left(\frac{|S||A|H^3}{\varepsilon^2}\right)$. There are analogous bounds for episodic, nonstationary dynamics. Key insights include a more refined definition of knownness depending on the probability of visiting (s, a) pairs (based on Tor, Hutter 2012), and bound $V^*\mathbb{P}(s'|s, a)$ instead of each separately.

In RL, samples are not iid, so we can't apply standard concentration.

Another approach is to think about regret. Total regret for an algorithm A is

$$\Delta(M, A, s, T) = \max_{\pi} \left[\mathbb{E}\left(\sum_{t=1}^{T} r_t\right)\right] - \sum_{t=1}^{T} r_t.$$

Note a subtlety: your policy and the optimal policy π may be evaluated along different sequences of states, unlike in PAC.

UCRL2 is a optimism under uncertainty algorithm. The diameter is the maximum over all state pairs of the expected number of timesteps to go from one state to another under a policy you choose.

The UCRL2 algorithm gives an upper bound and an expected regret upper bound

$$\Delta(M, UCRL2, s, T) \leq 34D|S|\sqrt{|A|T\ln\left(\frac{T}{\delta}\right)} \tag{50}$$

$$\mathbb{E}[\Delta(M, UCRL2, s, T)] \leq O\left(\frac{D^2|S||A|\ln T}{g}\right) \tag{51}$$

where $g = Q^*(s, a^*) - \max_{a' \neq a^*} Q^*(s, a')$ is the maximum gap betweeen the average reward of the best policy and best non-optimal policy for any state.

There is also an algorithm called REGAL (Bartlett).

We can also define Bayesian regret for RL. The PSRL expected regret (Bayesian with respect to distribution of MDPs) is $\mathbb{E}[\Delta(T, PSRL)] \leq O(\tau|S|\sqrt{|A|T\ln(|S||A|T)})$. This often does better than UCRL2. Note this is still linear in the state space.

Consider contextual bandits—close to the RL setting. At each time step get a context x that doesn't depend on our actions. The reward $r_t \sim R_{ax}$ depends on the action and context.

We want to do some generalization/sharing across our state or action space. Can we make different assumptions on R_{ax} to make this tractable? For example, it's linear.

We can reduce to supervised learning. We get computationally efficient (polylog in $|\Pi|$, size of set of policies),

$$\Delta(PolicyE, T) \leq O\left(\sqrt{|A|T\ln|\Pi|}\right).$$

Assume the class is expressive enough to have the optimal policy (but often it will have to be exponential, $|A|^{|X|}$).

3.5 Current and future work

3.5.1 Off policy policy evaluation

The past data is gathered using one or more behavior policies π_b, but we want to estimate the performance on a different π_a.

In an educational game: what level do we give the students at each time to keep them engaged and learning? We used off-policy evaluation to find a policy with 30% higher engagement. (Human selection did worse than random; in complex state spaces, human intuition can fail.)

The state space is a bunch of features of the gameplay. We don't assume it's Markov.

More details: we have historical data $R_j = \sum_i r_{ij}$, assume Markov. We want to

- Estimate MDP model and compute Q of π_e, or
- do model-free learning: just compute Q of π_e.

This gives a low variance estimator of policy perforamce, but it is biased and and inconsistent.

Use Importance Sampling: reweight distribution by probability of observing the history from the evaluation policy versus from the behavior policy. This is unbiased and strongly

consistent by is a high variance estimator.

$$\frac{1}{N}\sum_{j=1}^{N}\frac{\mathbb{P}(H_j|\pi_e)}{\mathbb{P}(H_j|\pi_b)}r_j$$

Can we get the best of two worlds? Dudik et al. 2011 did this for bandits, Jiang and Li 2015 did this for RL.

In MAGIC we (Thomas, Brunskill 2016) blend IS-based and model-based estimator to directly minimize the MSE.

$$MSE(x) = Bias\left(\sum_{j=1}^{\infty} x_j g^{(j)}(\pi_e|D)\right)^2 + \text{Var}\left(\sum_{j=0}^{\infty} x_j g^{(j)}(\pi_e|D)\right)$$

To compute the bias we need to know the true value. We do a conservative estimate of the value. We may overtrust the model sometimes. Empirically this leads to orders of magnitudes lower error.

Other variants:

$$\Delta Q(x,a) = \sum_{t\geq 0}\gamma^t\left(\prod_{s=1}^{t} c_s\right)\underbrace{\left(r_t + \gamma \mathop{\mathbb{E}}_{\pi} Q(x_{t+1},\cdot) - Q(x_t,a_t)\right)}_{\delta_t}.$$

- In IS the trace coefficient is $c_s = \frac{\pi(a_s|x_s)}{\mu(a_s|x_s)}$, which has high variance.
- In $Q^\pi(\lambda)$, discount large trajectories $c_s = \lambda$, but this is not safe (off-policy).
- In $TB(\lambda)$, take $c_s = \lambda\pi(a_s|x_s)$, but this not efficient.
- In Retrace(λ), take $c_s = \lambda \min\left(1, \frac{\pi(a_s|x_s)}{\mu(a_s|x_s)}\right)$. Idea is to cut high variance returns by terminating long sequencea of returns, dealing with the fact that this may throw away a lot of data. This is better but still struggles for long histories.

Long horizons, where we get the reward only at the end, is still a challenge. IS estimators have high variance. Retrace will cut (ignore long horizon reward), MAGIC will reduce to model-based estimator. An open problem is to get off-policy, unbiased, low variance estimators for long-horizon delayed reward problems.

3.5.2 Safety and risk sensitivity

We want confidence bounds on expected value learned from finite data, so we can guarantee the expected return of a new policy is better.

We want to compute more than expected return: distribution of returns, conditional value at risk.

People have focused mostly on batch learning from a past set of data, but we want safety during online reinforcement learning. See Moldovan and Abbeel 2012.

Deep RL scales up to large domains. DeepMind has results in Atari competitive with humans, and beat one of the world masters in Go. They use Monte Carlo Tree Search and use RL to estimate the value function.

Why has this happened over the last few years? Historically much of RL has used tabular representations, or linear combination of features (with limited expressive power, and requires feature selection), and dynamic Bayes nets which scales poorly. Fitted Value Iteration can use any regressor to represent value function, but many common function approximators are not non-expanders: the estimated Q-function may not converge. It can oscillate and fail to converge.

In practice, a neural network to represent the Q-value of policy can work really well. We have enormous data, NNs are a powerful function approximator, and various algorithmic modifications can improve stability. They have ≈ 13 layers deep.

Deep RL is exploding. Theory is very limited but it works well in practice.

The initial deep RL algorithms used simple ε-greedy exploration. In some games, exploration needs to be more targeted. This is also not sample efficient.

How to do generalization and exploration at the same time? Cluster state-action dynamics until evidence shows that it should be separate. This speeds up learning and still converges to optimal policy (Mandel, Liu, Brunskill, Popovis 2016). We can view this as posterior sampling over state representation as well as model parameters, and also relates to infinite POMDP's.

How to represent uncertainty with deep RL to support exploration? (Osband, Blundell, Pritzel, Van Roy 2016) The state representation to represent optimal policy may not be Markov. We might need a less compact representation to learn a policy vs. represent it. Aggregating states may lose Markovness.

Sample efficiency and representation: typically the formal sample efficiency scales as specifed state space, but we would like it to scale with the necessary state space. For factored MDP RL, we can get PAC RL to scale in the in-degree of necessary features, not max in-degree of all features. (Guo, Brunskill, in preparation.)

Open question: is this true (empirically) for deep RL?

Often only 1 frame is needed to represent the policy in Atari.

Other promising directions:

1. contextual MDPs (Krishnamurthy, Langford, Agarwal, Jiang, Schapire 2016a/b) (Best-in-class approach)

2. Temporal abstraction (actions, policies that take place over multiple timesteps). Ex. learning skill of going through a door; in-game transfer.

3. Human-in-the-loop RL (Mandoel, Liu, Brunskill, Popovic 2017). You can change the action space—humans create new actions to add into the system.

4 Interactive Learning of Classifiers and Other Structures

4.1 What is interactive learning?

Labeled data is expensive. Interactive learning hopes to reduce the amount of labeling necessary. Some examples:

1. Active learning: machine queries a few labels adaptively.
2. Explanation-based learning. In addition to lables the human gives an explanation in the form of relevant features.

   Ex. highlight words that are predictive of the label in document classification.

   ex. click on a part of the picture that explains the difference between the prediction and the actual label.

   what is the benefit these interactions give in addition to the labels? there's inherent ambiguity in the feedback
3. Interactions for unsupervised learning.

   For high dimensional datasets there are so many ways you can cluster it. There's no way an unsupervised algorithm knows which clustering you want.

   Show a random snapshot of the clustering of a few points; a human gives feedback, ex. which 2 points should be together that aren't.

   Machine has a clustering C of data X and wants feedback. Show human the restriction of C to $O(1)$ points from X.
4. Teaching: a human chooses maximally informative examples.

   In the other settings, a machine has to choose the informative examples.

Questions:

1. Efficient interaction algorithms: how much interaction is needed to learn?
2. Interaction versus computational complexity: situations where interaction circumvents computational hardness.
3. Modes of interaction: what kinds of interactions are easy and pleasant for human and provide reliable feedback?¿ Does it help to have a don't know option?
4. Communication gap between human and machine: How to bridge this?

4.2 Query learning of classifiers

Typical heuristics for active learning:

- Start with a pool of unlabeled data.
- Pick a few points at random and get their labels.
- Repeat: fix a classifier to the labels so far. Query the unlabelled point that is closest to the boundary, or most uncertain, or most likely to decrease overall uncertainty.

This seems sensible, how to analyze? The statistical learning model: there is an unknown, underlying distribution $\mathbb{P}$ on the (data, label) space, a hypothesis class H of candidate classifiers, and the target is h^* making fewest errors. Choose h_n after seeing n examples. We'd like $h_n \to h^*$ as rapidly as possible.

You get a distribution which looks less and less like the underlying distribution though: there is underlying bias going on.

Consider the data lying on a line. We're looking for a threshold classifier.

Even with infinitely many labels you could converge to the wrong thing, a classifier with greater error than the best achievable, which is not consistent.

The main problem is biased sampling: how can you ensure consistence? We have to be careful of misplaced confidence. We have to be aware of confidence, do some exploration.

Is there a generic fix to uncertainty-based heuristics?

What kind of benefits do we expect to get over random sampling?

Suppose the ground truth classifier is a threshold functions on the real line. In supervised learning, for error $\leq \varepsilon$, we need $\approx \frac{1}{\varepsilon}$ labeled points. In unsupervised learning, binary search just needs $\lg\left(\frac{1}{\varepsilon}\right)$ labels, giving an exponential improvement in label complexity. What about other hypothesis classes?

For supervised learning of a hypothesis class of VC dimension d, we need about $\frac{d}{\varepsilon}$ labeled points. Then there are $\leq \left(\frac{d}{\varepsilon}\right)^d$ ways to classify these using H (Sauer's lemma). If we ask queries that cut this space in half each time, then just $d \ln\left(\frac{d}{\varepsilon}\right)$ are needed. This is the dream situation. [1]

But halving queries might not exist: there are examples where each query could be highly biased. We can pick whatever comes closest (greedy approach). Many variants have been investigated. Query by committee: do something similar with Bayesian prior.

Three types of active learning:

1. Mellow active learning
2. Margin-based active learning
3.

[1] In teaching, you just need 2 examples: choose examples that are just to the left and right of the threshold.

4.2.1 Mellow active learning

Cohn, Atlas, Ladner 90's.

Separable data streams in.

Algorithm 4.1 (Mellow active learner)**:** Let H_1 be the hypothesis class. Repeat for $t = 1, 2, \ldots,$

- receive unlabeled $x_t \in X$
- If there is *any* disagreement within H about x_t's lable query label y_t and set $H_{t+1} = \{h \in H_t : h(x_t) = y_t\}$. Else $H_{t+1} = H_t$.

Only ask for labels for points in region of disagreement. This is the most conservative learner.

There is no need to explicitly maintain H.

This ends up with a label for everything: either the machine asked for the label, or was 100% sure and labeled it itself. There is no sampling bias.

The label complexity can be upper-bounded in terms of the VC dimension d of H and the disagreement coefficient θ which depends on H and on the distribution $\mathbb{P}$. To achieve misclassification error ε with constant probability, suffices to have number of labels

$$O\left(\theta d \ln\left(\frac{d}{\varepsilon}\right)\right).$$

The intuition: Let $\mathbb{P}$ be the underlying distribution on the input space X.

After t points are seen, H_t consists of classifiers with error at most $\approx \Delta_t := \frac{d}{t}$. Let $DIS(H_t) \subseteq X$ be the part of the input space on which there is disagreement within H_t Any point outside $DIS(H_t)$ is not queried.

The disagreement coefficient θ satisfies $\mathbb{P}(DIS(H_t)) \leq \theta \Delta_t$.

The expected number of queries is

$$\sum_{t=1}^{T} \mathbb{P}(DIS(H_t)) \leq \theta \sum_{t=1}^{T} = \theta \sum_{t=1}^{T} \frac{d}{t} \leq \Delta_t \approx \theta d \ln T.$$

Defining the disagreement coefficient requires us to get into the geometry of the hypothesis class. The induced pseudo-metric on hypotheses is

$$d(h, h') = \mathbb{P}[h(X) \neq h'(X)].$$

The ball is $B(h, r) = \{h' \in H : d(h, h') < \varepsilon\}$.

The disagreement region of anyu set of candidate hyp $V \subseteq H$ is $DIS(V) = \{x \in X : \exists h, h' \in V \text{ s.t. } h(x)$
Need only consider $V = B(h^*, r)$, h^* target hypothesis,

$$\theta = \sup_r \frac{\mathbb{P}[DIS(B(h^*, r))]}{r}.$$

For thresholds for $\mathbb{R}$, $\theta = 2$.

For linear separators, $\theta \leq \sqrt{d}$.

4.2.2 Margin-based active learning

(Balcan-Long)

Consider algorithms based on actual heuristics, like querying points lying close to the boundary. Query points that are within an ε margin of the boundary, and reduce ε over time.

Algorithm 4.2 (Margin-based active learning)**:** For example, say all $\|x\| = 1$, $t = 1, 2, \ldots$, let w_t be the classifier based on data so far. Randomly choose points with $|x \cdot w_t| \leq m_t$, and query their labels. $\{m_t\}$ is a schedule of margins that decreases to 0.

Later on, you need more examples because most of the examples are useless. Never get to the point where you need $\frac{d}{\varepsilon}$ examples to halve the error. Cut out the regions you're sure about. Operate in the region where the error rate is constant.

4.2.3 Active annotation

The schemes we covered before are fine-tuned to the classifier. If you decide to change the classifier, you have to get a different set of examples each time.

Input:

- finite set of data points $\{x_1, \ldots, x_n\}$ each of which has an associated label y_i that is initially missing.
- parameters $0 < \delta, \varepsilon < 1$.
- access to oracle that can supply any label y_i.

Output $\widehat{y}_i$ such that w.p. $\geq 1 - \delta$, at most ε fraction of these labels are incorrect

$$\sum_i \mathbb{1}(y_i \neq \widehat{y}_i) \leq \varepsilon n.$$

The goal is to minimize calls to the oracle.

For example, the input is a neighborhood graph G whose nodes are the data points x. Each node has an unknown label. The goal is to find the cut-edges in this graph that separate two labels.

Algorithm 4.3 (S^2 algorithm, Dasarthy-Nowak-Zhu)**:** Keep going until budget runs out.

- If there exist labeled nodes of opposite polarity that are connected in G, find the shortest path connecting nodes of opposite labels, and query its midpoint. Else Choose a random point and query.
- Remove any newly -revealed cut edges from the graph G.

Reduce amount of effort required by human to train a machine. Also applies to minimize experimental budget.

Often adaptive sampling leads to complicating complexity, including scalability, computation, fragility to modeling assumptions. This is why we haven't seen active learning used so much in practice.

Does interactivity always help? Not always, but it has promise in many applications.

Solution: identify specific ML models and apps that have demonstrated real-world successes. We go back to multi-armed bandits.

We focus on theoretical foundations of basic and linear bandit algorithms with human-machine interaction applications.

Why? The analysis and bounds are very tight (even in terms of constant factors), so we have theoretically-sound algorithms that are as aggressive as possible.

In the stochastic bandit problem, there are n arms, the reward distributions p_i are unknown. Each step select arm $i_t \in [n]$ and draw $x_{i_t,t} \sim p_{i_t}$ independently from past.

Examples: arms are ads, action is display an add, reward is clicks. Help scientists adapt select exper to det which genes involved in disease. Arms are genes/proteins.

Let $\mu_1 > \cdots \geq \mu_n$ be the expected rewards of each arm, $\Delta_i = \mu_1 - \mu_i$ the "gaps", $x_{it} \sim p_i$ indepedent random reward from arm i at tie t.

$$\widehat{\mu}_{i,t} = \frac{1}{t_i} \sum_{j=1}^{t_i} x_{ij}$$

Assume bounded, subgaussian rewards. Then for all i and all t, we get a confidence bound on μ_i.

The regret is

$$R_T = T \max_i \mu_i - \mathbb{E} \sum_{t=1}^{T} x_{i_t,t} = \sum_{i=1}^{n} \Delta_i \mathbb{E} T_i.$$

Sample arms to minimize $\mathbb{P}(\mu_{\widehat{i}} \neq \max_i \mu_i)$.

Eventually stop sampling suboptimal arms as long as there is some gap between the top and second best action.

Regret is $R_T = O(\sum_{i \geq 2} \frac{\ln T}{\Delta_i})$. Select top few.

For any i and all t w.p. $\geq 1 - \delta$,

$$\widehat{\mu}_{i,t} - 2\sqrt{\frac{\ln \ln \left(\frac{t}{\delta}\right)}{t}} \leq \mu_i \leq \widehat{\mu}_{i,t} + 2\sqrt{\frac{\ln \ln \left(\frac{t}{\delta}\right)}{t}}.$$

This is an adaptive algorithm. How does it compare to the best non-adaptive algorithm?

We have to sample every arm as many times as the second arm, $\sum_i T_i = \widetilde{O}(n\Delta_2^{-2})$. We could be a factor of n off.

Every week $n \approx 5000$ captions are submitted to the New Yorker. Crowdsource contest to volunteers who rate captions. The goal is to identify funniest caption.

Over time, focus on the better captions. Little gap between theory and practice. There is a 5x improvement in sample size.

There is a feature vector associated with each arm $x_i \in \mathbb{R}^d$. The reward model is $y_i = \langle x_i, \theta^* \rangle + \varepsilon_t$. Try to construct a confidence set so θ^* is always in that confidence set. Form the least-squares/ridge regression estimate of θ.

$$\widehat{\theta}_t \approx \theta^* + (x_t^T X_t + \lambda I)^{-1} X_t^T \varepsilon_t$$

Using martingale techniques get a confidence ellipsoid.

To select the arm to sample

$$x_{i_t} = \text{argmax}_{x_{1:n}} \left\langle x_i, \widehat{\theta}_t \right\rangle + \sqrt{\beta_t}\sqrt{x_t^T V x_t}...$$

The first term is the exploit term; the second is the explore term.

Interactive image search problem: User gives feedback on images from the Zappos50K datset (shoes). From deep learning, shoes are represented by $\mathbb{R}^{1000}$ feature vectors. Linear bandits retrieve 2-3 times as many relevant items.

Now we talk about systems and apps. It's difficult to even do an experiment! There are many practical challenges. Researchers are not easily able to test, theoreticians may be unaware of real-world challenges with interesting math solutions; practitioners are not able to apply interactive learning methods.

Two groups working on this are Microsoft's Multiworld Testing Decision Service, and NEXT at UW-Madison.

5 Deep learning for robotics, Sergey Levine

In my work, I design an algorithm and take it from the foundations to a real-life application. Robots include arms, mobile robots, drones...

Deep learning allowed end-to-end learning for computer vision.

5–10 years ago, the image pipeline was the following.

- Extract features, e.g. HOG
- Extract mid-level features (DPM)
- Apply a classifier, e.g. SVM.

Deep learning still does these things, but feature selection is automated. The kind of features we learn are somewhat generic and particular suited to the classification task.

How do we use this output to make a decision about what to do next? Note the decisions will affect what it sees next, so there is a loop (RL). There are two parts to robotics, perception and action. We can close this loop, the sensorimotor loop: train the entire model together.

Richard Dawkins "When a man throws a ball high in the air and catches it again. he behaves as if he had solved a set of differential equations in predicting the trajectory of the ball... at some subconscious level, something functionally equivalent to the mathematical calculations is going on."

McLeod and Dienes: But if we couple perception and control, we can shortcut the pipeline and do better.

(Ex. for pilots: Look for a spot of dust, and see if it's stationary with respect to the other airplane; if it is you are on a collision course.)

A person opens a door better than a robot now with less sensory information.

Standard robotic control involves observations, state estimation (vision), modeling and prediction, motion planning, low-level controller (PD), motor torques... We want to do this end-to-end.

Inputs from sensors are input into learned model, and it directly outputs the actuator commands. In reality there are many challenges that makes it more difficult than the $n+1$th application of deep learning. Often the supervision is indirect (though for self-driving cars you can have more direct supervision), and actions have consequences! Data is not iid, and actions will affect what you observe.

Why should we care? It allows us to make robots that do useful things in the real world, and understand intelligence. We can build other interactive systems like interactive personal assistants, smart power systems...

5.1 Formalisms

Let o be the observation and output be u. We want to learn the policy $\pi_\theta(u_t|o_t)$.

Let x be the state. In the probabilistic graphical model, the state obeys the Markov property. The observation in general does not.

5.2 Imitation learning

Consider o is the view from a camera on the windshield and u is the steering command. The simplest thing is to get a human to drive around to collect data, train with supervised learning to get $\pi_\theta(u_t|o_t)$.

Does this work? No. The intuitive explanation: Because of error, the expected trajectory will be different from the training trajectory, and once it is off the trajectory it can make bigger mistakes. Mistakes can accumulate quadratically.

Why did that self-driving car work? There are 3 cameras, not 1. Their network only processes 1 image at a time. There is a camera facing foward, labeled with steering command, and cameras facing left/right labeled with the steering command with small offset to right/left. This ensures stability: the algorithm can now cope with deviations.

Take the optimal controller, ask the controller to compensate for small perturbations to the states, and use those to augment the dataset. This tends to work better.

Can we make this work more often? Analyze this from a probabilistic standpoint. Why does the training diverge from the expected trajectory? Consider $p_{data}(o_t)$. $\pi_\theta(u_t|o_t)$ sees data from $p_{\pi_\theta}(o_t)$. The test distribution is different from the training distribution.

Instead of being clever about $p_{\pi_\theta}(o_t)$ be clever about $p_{data}(o_t)$.

Use DAgger: dataset aggregation.

Collect training data from $p_{\pi_\theta}(o_t)$ instead of $p_{data}(o_t)$, by running $\pi_\theta(u_t|o_t)$.

1. Train $\pi_\theta(u_t|o_t)$.
2. Run $\pi_\theta(u_t|o_t)$ to get dataset $D_\pi = \{o_1, \ldots, o_M\}$.
3. Ask human to label D_π with actions u_t.
4. Aggregate $D \leftarrow D \cup D_\pi$.

This will converge.

What are the problems?

We have ask a human for the labels. Can we get a computer to supply those labels.

Summary: Imitation learning is often insufficient by itself because of distribution mismatch. It sometimes works well with hacks (ex. left/right views from car).

Distribution mismatch does not seem to be the whole story. Imitation often works without Dagger. Can we think about how stability factors in?

Imitation is more than copying an action someone took. It can be more high-level—copying someone's intentions. When you infer intention and attempt to find a behavior that satisfies it, you are combining your observation of someone else and your own experience. Then you don't have to rely on the demonstration giving you all the data. This is a crucial ingredient of true imitation learning!

5.3 Imitation without a human

We want $\min_{u_{1:T}} \sum_{t=1}^{T} c(x_t, u_t)$ such that $x_t = f(x_{t-1}, u_{t-1})$.

A classical method is to use trajectory optimization. Keep substituting to get a single optimization problem, or use sequential quadratic programming, etc. You can differentiate through via backpropagation and optimize. In practice it helps to use a 2nd order method because f is applied many times.

We can write a probabilistic version, $x_{t+1} = f(x_t, u_t)$, $x_{t+1} \sim p(x_{t+1}|x_t, u_t)$ such as $N(f(x_t, u_t), \Sigma)$. We can build a simple stochastic policy such as $p(u_t|x_t) = N(K_t x_t + k_t, \Sigma_{u_t})$. There is an $\mathbb{E}$ in the optimization problem now.

Can we use something like this in the context of an algorithm like DAgger?

Another problem with DAgger: you also need to run π_θ. If you have a bad policy, results can be bad (car driving unsafely).

PLATO (policy learning with adaptive trajectory optimization):

1. Train $\pi_\theta(u_t|o_t)$ from human data.
2. Run $\widehat{\pi}(u_t|o_t)$ to get dataset $D_\pi = \{o_1, \ldots, o_M\}$.
3. Ask human to label D_π with actions u_t.
4. Aggregate $D \leftarrow D \cup D_\pi$.

Here

$$\widehat{\pi}(u_t|x_t) = \text{argmin}_{\widehat{\pi}} \sum_{t'=t}^{T} \underset{\widehat{\pi}}{\mathbb{E}}[c(x_{t'}, u_{t'})] + \lambda D_{KL}(\widehat{\pi}(u_t|x_t)||\pi_\theta(u_t|o_t)).$$

Replan at each step. Replanning is model predictive control (MPC). π_θ is control from images, while $\widehat{\pi}$ is control from state. We need some way to obtain the state at training time to solve this optimal control problem with respect to states. (This is a strong assumption.) We assume $p(x_{t+1}|x_t, u_t)$ is known but $p(o_t|x_t)$ is unknown. This is realistic for e.g. autonomous cars: physics are relatively simple; images are complicated.

Safety comes from: You're not ever running π_θ until it's fully trained.

There is also the rare events problem which this says nothing about.

Example: flying quadrucopter. If the policy is not very good (has high cost, ex. leads you towards an obstacle), the cost function becomes large.

This assumes knowing the true state. In the future we want to relax that assumption.

It would be nice to not require knowledge of dynamics. We can do trajectory optimization with unknown dynamics. We need derivatives $\frac{df}{dx_t}, \frac{df}{du_t}$ (linearization of system).

Fit a plane to data and hope it's close enough. There's a trick to make this work. If dynamics are $p(x_{t+1}|x_t, u_t) = N(f(x_t, u_t), \Sigma)$, fit (after each time step) $f(x_t, u_t) \approx A_t x_t + B_t u_t$, $A_t = \frac{df}{dx_t}, B_t = \frac{df}{du_t}$. we may go into region where linearization is wrong. We can add the constraint $D_{KL}(p(\tau), \widetilde{p}(\tau)) \leq \varepsilon$. We have a heuristic that adjusts ε.

How to combine with policy learning?

Guided policy search. $\min_\theta \mathbb{E}_{\pi_\theta}[c(\tau)]$ is equivalent to $\min_{\theta, p(\tau)} \mathbb{E}_p\, c(\tau)$ such that $\pi_\theta(u_t|o(x_t)) = p(u_t|x_t)$ for all t, x_t, u_t. This constraint is equivalent to $D_t(\pi_\theta, p) = 0$ for all t.

Write the Lagrangian and use dual gradient descent.

$$\mathcal{L}(\theta, p, \lambda) = \min_p \mathbb{E}[c(\tau)] + \sum_{t=1}^{T} \lambda_t D_t(\pi_\theta, p).$$

Optimize $\mathcal{L}$ wrt p, wrt θ, update λ with subgradient descent, and repeat.

Combine trajectory-centric RL with supervised learning, which tries to generalize.

The data is used to fit dynamics and used to train the NN.

For imitating optimal control, is there any difference from standard imitation learning? We can change the behavior of the "teacher" programmatically.

Can the policy help optimal control rather than the other way around?

5.4 Reinforcement learning

We consider model-free algorithms; directly optimize policies.

First we cover policy gradient, then make it practical with function approximators like deep NN.

We want

$$\min_\theta \underbrace{\mathop{\mathbb{E}}_{\tau \sim p_\theta(\tau)}[c(\tau)]}_{J(\theta)},$$

where

$$p_\theta(\tau) = p_\theta(x_1, u_1, \ldots, x_T, u_T) = p(x_1) \prod_{t=1}^{T} p(x_{t+1}|x, u)\pi_\theta(u_t|x_t).$$

Compute the gradient $\nabla_\theta J(\theta) = \int [\nabla_\theta p_\theta(\tau)] c(\tau)\, d\tau$. Use the trick $\nabla_\theta p_\theta(\tau) = p_\theta(\tau) \nabla_\theta \ln p_\theta(\tau)$. Get

$$\nabla_\theta \ln p_\theta(\tau) = \sum_{t=1}^{T} \nabla_\theta \ln \pi_\theta(u_t|x_t).$$

The dynamics go away! (Williams 1992)

For example, $\pi_\theta(u_t, x_t) = N(f_{NN}(x_t), \Sigma)$. In the REINFORCE algorithm, sample $\{r^i\}$ from $\pi_\theta(u_t|x_t)$, estimate

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\left(\sum_{t=1}^{T} \nabla_\theta \ln \pi_\theta(u_t|x_t)\right)\left(\sum_{t=1}^{T} c(x_t, u_t) - b\right)\right],$$

made gradient step.

Here b is a baseline. We use the identity

$$\mathbb{E}[\nabla \ln p(y) b] = \int p(y) \nabla \ln p(y) b = \int \nabla p(y) b = b \nabla \int p(y) = 0$$

b doesn't change the mean but reduces variance. A good, not optimal choice is $b = \mathbb{E}\left[\sum_t c(x_t, u_t)\right]$.

Using the average cost baseline, a good/bad policy becomes more/less likely.

This is often not good enough with large policy classes like NN, because of high variance. (Make b depend on state and even action.) There is poor conditioning (so use higher order methods, natural gradient), it is very hard to choose step size (trust region policy optimization, TRPO).

We can rewrite

$$\nabla_\theta J(\theta) = \sum_{t=1}^{T} \mathbb{E}\left[\nabla_\theta \ln \pi_\theta(u_t|x_t)\left(\sum_{t=1}^{T} c(x_t, u_t) - b\right)\right]$$

Using knowledge of causality (action at later times don't affect previous rewards) we get

$$\nabla_\theta J(\theta) = \sum_{t=1}^{T} \mathbb{E}\left[\nabla_\theta \ln \pi_\theta(u_t|x_t) \mathbb{E}\underbrace{\left(\sum_{t'=t}^{T} c(x_t, u_t) - b\right)}_{Q^{\pi_\theta}}\right]$$

Consider the toal remaining cost of executing $\pi_\theta(u_t|x_t)$.

We can fit a function approximator to Q. We want

$$\widehat{Q}^{\pi}_{\phi}(x_t, u_t) \approx \mathbb{E}\left[\sum_{t'=t}^{T} c(x_{t'}, u_{t'})\right],$$

fit by regression: This gives the actor-critic method.

Challenges include usual problems with policy gradient, bias/variance tradeoff. FA has lower variance but higher bias. We can combine Monte Carlo and function approximation: generalized advantage estimtion.

This can solve bipedal running.

There are also online actor-critic methods. Make one decision, add to duffer, sample minibatch from buffer, estimate $\nabla_\theta J(\theta)$, update θ.

Deep deterministic policy gradient (DDPG) is effective.

You can also just directly use the Q-algorithm to make decisions. Instead of estimating the Q-function of the current policy, you can take the best action (Q-learning). π could be $\propto \exp(-\widehat{Q}_\phi(x_t, u_t))$ or ε-greedy.

Q-learning for deep RL: make one decision with $u \sim \pi_\theta(u|x)$, add to buffer D. Sample minibatch from D to fit $\widehat{Q}_\phi$. Choose $\pi(u|x)$ based on $\widehat{Q}_\phi$, e.g. $\pi(u|x) \propto \exp(-\widehat{Q}_\phi(x,u))$.

For continuous Q-learning, you can choose a representation for your Q function that makes the minimization easier. Output a Q-function quadratic in the action.

$$\widehat{Q}_\phi(x,u) = \frac{1}{2}(u-\mu(x))^T P(x)(u-\mu(x)) + V(x).$$

Tradeoffs between imitation learning and RL:

Sample complexity: FA like NN tend to be data-hungry; this is a major challenge. Training on Atari (DQN) would take 100 hours if real-time; for the bipedal task it would be 50 hours (GAE). DDPG/NAF take 4-5 hours to learn basic manipulation.

Model based methods are more efficient: time-varying linear models take 3 minutes for real world manipulation, GPS with vision takes 30-40 minutes for real-world visuomotor policies.

What do we need to get the same degree of generalization? For supervised learning, we need computation, algorithms, data. For learning sensorimotor skills, we have computations, sort-of algorithms, but where do we get data?

We can scale up via parallel operating robots training together.

Emergent behavior for grasper: When the object is soft, it's more effective to just puncture it in the middle, if it's heavy, grasp in middle, if it's flat, estimate prob of success, until find one with good probability of success.

Can we learn the cost via visual features? ("learn what success means")

Challenges include sample complexity, safety, scalability, supervision.

Exploration is an important question. Part of why in door-opening, we used Q-learning is because it does better in exploration than model-fitting approach. Random exploration comes from using Boltzmann-style policy. A model-based approach depends on how good the model is. More explicit exploration can do a long way. One family that's effective are generative models. Use some measure of the surprise of model. Add exploration bonus to encourage visiting those more.

Adding noise can help: learn to correct mistakes.

What's a good place to do theory? In imitation learning, we want some set of assumption for which was can say something concrete. We can use that as guidance for data collection.

What is the ideal theorem you would like to see? In imitation learning, if your expret behaves according to some policy which satisfies some properties (stay in constrained region, etc.) then imitation learning has bounded loss, regret.

Imitation learning is a class of techniques. There are different properties it can have. DAgger requires more data.

Why are robots acting slowly? If you are using a control algorithm that assumes some model of system dynamics and it's not good, if you move slowly it will be less wrong.

Model-based, free seem different fundamentally. How to close the gap? One classical approach is the Dyna algorithm: fit a model of dynamics to the data, generate artificial rollouts from the model.