Contents

1 Introduction (Sanjoy Dasgupta)

I'll talk about 5 areas which need foundational work.

1. Teaching
2. Explanations and interpretations
3. Unsupervised++
4. Imitation
5. Semantic communication

These areas all have one feature in common: Cooperation between agents of different types, that don't know each other's insides (ex. machine and human).

1.1 Teaching

There is a spectrum of types of examples: adversarial (in online learning), random (in statistical learning), benign (in teaching). Does sample complexity dramatically improve? People have converged upon a particular model that is influential but also broken.

What is the minimum set of labeled examples needed to uniquely identify the target concept? It's a kind of description dimension. This is in relation to a specific concept class C.

Definition 1.1: Let C be a concept class and $h \in C$ be a target. $TD(h, C)$ is the smallest set of labeled instances for which h is the only consistent concept in C.

We can define $TD(C) = \max_h TD(h, C)$, or $\mathbb{E}_h TD(h, C)$.

This is geared towards finite concept classes.

This is broken because of the following problems.

1. It assumes the teacher knows the representation of the learner and the learner's concept class. Examples are tuned to the concept class.
2. The problem of selecting the teaching set can be NP-hard.
3. The predictions it gives for ideal teaching sets are ridiculous, ex. cat that looks like dog and dog that looks like cat. In practice people select examples that are far apart (the canonical cat/dog).

4. This only works for the realizable case.

What to do? What are better teaching models?

1. Who is teaching who? Human/machine teaching human/machine?
   (a) Human-human: education/cognitive science/childhood development
   (b) Human-machine: machine learning
   (c) Machine-human: intelligent tutoring
   (d) Machine-machine:
   cf. cotraining, two machines bootstrapping each other.
   GAN's teach each other to generate/discriminate (with opposite goals).
2. Avoid assuming the teacher knows the learner's represntation and concept class.
3. Interactivity: The machine could ask questions.

   Any semi-realistic model would be interactive.
4. Come up with models that gives far apart examples (Jerry Zhu).

   Ex. Let's say the learner does nearest neighbor. Suppose it is noisy nearest-neighbor. That pulls you apart.
5. Curriculum learning and self-paced learning strategies. Hierarchical learning. Simple things are learned first; then you add things.
6. Other kinds of tasks besides classification, ex. generative.
7. Restrict to an interesting domain like language.

The teacher does not have unbounded computation, but knows the concept and has a storehouse of examples gleaned from experience.

1.2 Explanations and interpretations

Ex. more than just saying you like a movie, say that you like a specific actor.

Ex. in computer vision In addition to giving a label (ex. zebra, antelope), give one-word explanations (stripes, antlers). Learn classifiers for these intermediate features as well. This taps into a potentially infinite latent space.

When feature space is high dimensional, this helps.

1. Models of explanation-based learning.

   What are the benefits of explanation-based learning.

2. Interpretable classifiers (transparency in ML).

   Output a hypothesis that scientists can understand.

   Accompany predictions with explanations. “Your loan was rejected because...”

   Decision trees used to do this automatically until people realized random forests do better.

   Use explanations to generate interpretable classifiers?

   Ex. sparse classifiers: give the support of which features you used.

1.3 Unsupervised learning++

Ex. topic modeling. Some are good, some are sliced/diced in various ways, some are garbage. Running the Gibbs sampler for longer doesn’t solve the problem. This is ripe for interactive feedback of some type.

Sometimes you just literally need feedback; we want to quantify how much feedback is needed.

What type of interaction is useful algorithmically and for the human?

It could be relationships between data points, constraints based on features, etc. A practitioner would choose the algorithm and the form of interaction.

(Q: how to avoid tricks such as: To transmit finite automaton, grammar, write down grammar or automaton. Make an arbitrary convention of how to translate examples into a grammar.)

1. Improving unsupervised learning with interaction

   (a) Modes of interaction

   (b) How much interaction?

   Normally use Euclidean distance. What you want to use if people’s subjective similarity scores?

2. Generalization theory for unsupervised learning

“Unsupervised learning++=Supervised learning–”: Unsupervised learning is talked about a lot as lossy compression. Here, you don’t have exactly the label you want, but have something that’s associated with what you want. This is unsupervised++ because one can imagine this built on top of unsupervised learning algorithms.

The only results I’m aware of are nonstatistical results: ex. for clustering points, query $n \ln n$ distances rather than $\binom{n}{2}$. There should not be any n here at all, just ε and the distribution.

1.4 Imitation learning (and teaching)

One or two decades we’ll be telling our domestic robots “this is how we like to make our coffee.”

Imitation learning seems a tractable case of reinforcement learning. Imitation implicitly assumes a sequence of actions.

It's not enough to explain why we did this; we have to explain things we don't want to do? Littman, NIPS had a formalization.

1.5 Semantic communication

1. One paper was by Juba, Sudan, Goldreich. Ex. You don't know the language. What protocol can you execute? The answer is disappointing: try everything.

2. Percy Liang: A computer is in charge of blocks. You want to move the blocks to a specific configuration by telling the computer what to do.

   Throw in 2 constraints: compositionality of language, pragmatics (different utterances probably mean different things).

   (Pragmatics is like dropout: it helps but is not crucial. There's other things going on, which NLP takes for granted but would be interesting for theorists: ex. loglinear model.)

1.6 Other topics

1. Language learning and generation

2. Crowdsourcing. Designing proper crowdsourcing experiments. How do we learn from weak teachers (Amazon Turkers) that make errors?

   See work by Nihar Shar, Kevin Jameson (Next, with Robert Nowak).

   `http://nextml.org/`

2 Two Paradigms of Semi-Supervised Active Clustering with Sample and Computational Complexity Bounds (Shai Ben-David, University of Waterloo)

I want to cover three things.

1. Introduction/preaching: what should clustering theory do?

2. Semi-supervised active clustering (joint work with Hassan Ashtiani and Shrinu Kushagra) [AKB16] `https://arxiv.org/abs/1606.02404`

3. Learning representations from clustering (Hassan) [AB15] `https://arxiv.org/abs/1506.05900`

2.1 Introduction

Different clustering algorithms yield very different outcomes. Contrast this with other computational tasks, like classification, for which different algorithms roughly give the same results.

Thus the choice of a clustering algorithm is very important. They have different objectives which cannot be satisfied simultaneously. Things we want may include:

1. Similar points are clustered together
2. Dissimilar points are in different clusters. (Already these may conflict—ex. a long chain of points)
3. Balanced cluster sizes
4. Stable under perturbation

Where in this simplex of properties do we want?

- Single linkage only cares about similar points.
- Max linkage (postpone merging far points) only cares about dissimilar points.
- k-means cares about balancing cluster sizes.

Different applications have different priorities. One application is record de-duplication: cluster together records that are the same. Here we would want max linkage: we want dissimilar records not to be identified. For viral spread, we want to group similar points together.

There is no universally optimal clustering. There is a need for domain-specific biases.

The first question is what algorithm to use; there is not enough research in this direction. It's not clear what tool is good for that. Most people use k-means without thinking. "Because everyone else is using this algorithm."

We can address the problem in two ways:

- I give similarity of pairs. Find the right clustering algorithm from that.
- Here is the tool I'm using, but I will play with the similarity.

We focus on the second approach. There are ways of asking for input from the user that is more intuitive than asking them to choose the objective—ex. ask them how to cluster a small sample of points.

2.2 Semi-supervised active clustering

See NIPS2016.

Here is the setup.

- We have unknown clustering $C_1, \ldots, C_K$ of (X, d) where d is known.

- The algorithm can interact with an oracle that knows $C_1, \ldots, C_K$.
- The type of interaction is queries of the form are x_1, x_2 in the same cluster or different clusters.

The algorithm computes, then decides which points to ask about next, etc.

If the user already knows, why do you need the clustering algorithm? There are so many data points; you don't want to ask n^2 queries. You need the algorithm to overcome the enormity of the data. There are specific applications for which this model is relevant, ex. data de-duplication. Here the number of clusters is large compared to the number of records.

The number of queries needed is logarithmic in the dataset.

I show the result that under some assumptions on the target clustering, there exists an algorithm that requires $O(k \ln n)$ queries and finds the target clustering in linear time $O(kn(\text{dim}))$, where dim is the dimension.

Without the queries the task is NP-hard. After log queries, the task collapses to linear time!

We assume that answers are noise-free.

We don't need to know k in advance. Hardness results kick in for k a function of n.

Trivially, kn queries always suffice. Find one representative in every cluster. For every new point, ask whether it's in the same cluster with every representative.

Here are the assumptions.

1. Center-based clustering. Every point belongs to the nearest center (Voronoi cell).
2. μ_i, the center of C_i, is $\mathbb{E}_{x \sim C_i} f(x)$, for some known f. This definitely holds for k-means.
3. Niceness (clusterability) assumption: γ-margin.

   Definition 2.1: $C_1, \ldots, C_k$ satisfies the γ-margin condition if $\forall i, \forall x \in C_i, \forall y \notin C_i$,
   $$d(x, \mu_i)(1 + \gamma) \leq d(y, \beta_i).$$

   Around every cluster there is an empty margin.

What do we know about k-means with such an assumption?

Theorem 2.2 (Hardness result)**.** *k-means is NP-hard under γ-margin condition as long as $\gamma \leq 0.84$. (Euclidean distance) In general, it is NP-hard for $\gamma \leq 1$.*

Here $k \sim n^{\varepsilon}$. The reduction is from set cover.

Although these conditions look strong, it is still hard to do k-means under these assumptions.

Positive result: For $\gamma > 2$, k-means without queries is feasible. Use single linkage and use dynamic programming to search over all prunings.

[1]

Usually when people find good clustering algorithms, it's with strong assumptions. The task becomes hard before the condition becomes natural.

The algorithm with queries:

[1] AWigderson: Could you find a clustering that uses $k \ln n$ centers greedily? Probably. Find too many clusters and use queries to prune?

1. Ask enough queries to get "many" (N) points in one cluster. Ask Nk queries.

   (Once you have representatives, you can compare, and we can ask which cluster you belong to.)

2. Pick a cluster with enough points and estimate its center. (This estimate is good by Chernoff.) The γ tells me how many points I need in my cluster to be able to tell whether points are in that cluster.

3. Binary search to find the cluster radius. Ask whether a point of some distance away is in the cluster. Need $\ln n$ queries.

4. Delete points in this cluster and repeat.

2.3 Learning representations from clustering

This is a different type of interaction. Hand the user a small subset $S \subseteq X$ and ask to get back their desired clustering of S.

What can I learn/generalize from this?

Based on this $S_1, \ldots, S_k$, how can we pick a suitable clustering tool for X? Note that maybe not all clusters are represented. (Otherwise, we can think of it as a classification problem.)

The idea is that what we want to learn is the metric. Instead of picking between different clustering algorithms, fix the clustering algorithm (regularized k-means) and learn the metric.

We fix a family of embeddings of X into some $\mathbb{R}^d$ (a family of kernels over X), F.

Now we can phrase the problem more precisely: the algorithmic task is to find $f \in F$ such that $A(f(X))|_S$ is close to $S_1, \ldots, S_k$. This is a well-defined problem; we can talk about sample and computational complexity.

This is not an ideal solution because it could be that the number of clusters could be much larger. The user can also give clusterings that are inconsistent when given a small vs. large set.

For sample complexity analysis of ERM algorithms in the model, we need a notion of distance between clusterings, and a notion of complexity of F.

The distance is

$$D((C_1, \ldots, C_k), (C_1', \ldots, C_k')) = \min_{\pi \in S_k} \frac{1}{|X|} \sum_{i=1}^{k} |C_i \triangle C'_{\pi(i)}|.$$

Now we can phrase as a PAC problem. What is the sample size $m_F^{UC}(\varepsilon, \delta)$ to get within ε with probability $1 - \delta$? This depends on the generalized pseudodimension of F. The pseudodimension is defined as follows. For F is a family of functions $X \to \mathbb{R}$,

$$p\dim(F) = \max\left\{n : \exists x_1, \ldots, x_n, b_1, \ldots, b_n, \forall \sigma \in \{0,1\}^n, \exists f \in F, \forall i \leq n, \mathbb{1}[f(x_i) \geq b_i] = \sigma_i\right\}.$$

It is the largest set which we can pseudo-shatter.

We have to generalize pseudodimension to vector-valued functions. The generalize is the maximum pseudodimension of all the projections.

In some common families of kernels, we can calculate the pseudodimension.

We don't have an efficient algorithm because to find the ERM we need to solve regularized k-means. How to use users' information to learn clustering on the whole dataset is interesting and this is only a partial answer.

There's little work on sample complexity of metric learning—how many pairs you need to get information about.

Would some condition on clustering make it easier? Fat-shattering dimension, etc.

3 Robust learning and inference (Yishay Mansour, TAU)

We consider the case that some of the attributes may be adversarially corrupted or missing. We limit the adversarial corruption to a finite set of modification rules, and we model it as a zero-sum game between an adversary, who selects a modification rule, and a predictor, who wants to accurately predict the state of nature. We consider a learning setting where the predictor receives a set of uncorrupted inputs and their classification. The predictor needs to select a hypothesis, from a known set of hypotheses, and is latter tested on inputs which the adversary might corrupt. We show how to utilize an ERM oracle to derive a near optimal predictor strategy, namely, picking a hypothesis that minimizes the error on the corrupted test inputs. We will also briefly mention the results for the inference model. In the inference setting the predictor has access to the joint uncorrupted distribution, and needs to build a predictor for adversarially corrupted inputs.

What do we mean by robustness? Different fields use it to refer to different things.

- Robust statistics: be immune to outliers.
- Robust optimization: be immune to small parameter perturbation (in some metric)
- Noise models in computational learning theory: Use a model of how data is generated, ex. flip labels with probability 0.1. But the algorithms can break if we flip with probability ≤ 0.1. Everything is calibrated to the specific noise level, 0.1.

  Ex. nearest neighbor
- Our model: "things are not what they seem."

Ex. Spam detection. Build many filters, and combine them into a spam filter. Unfortunately, the bad guys can adapt.

This is really a game between spammers and detectors. Spammers adjust content to the detectors. They can learn to fool a few detectors. Our goal is to classify spam correctly even if spammers adjust their messages.

Ex. Robust network failure detection.

- What happens when the detectors fail? We get from the detector something that looks reasonable but is incorrect.

- How to model failures? There are two ways, Bayesian vs. worst-case.

  A Bayesian needs to model the probability of failure and conditioning on the failure, what the distribution of outputs is.

  I'll consider worst-case.

The goal is to perform a good failure detection, in the sense of overcoming a k-point failure, under adversarial behavior.

I'll consider both missing and corrupted attributes.

1. Why do we have missing/corrupted data? Can we avoid it by requiring clean data?

   We used to think of ML as having correct inputs (we are choosing the inputs), and giving outputs.

   Applied ML doesn't have the methodology for choosing inputs.

2. Can we assume that it is iid? No, missing or corrupted data might depend on attribute value.

3. Do we have to clean the data? No, we should directly predict!

   Two paradigms: if you have missing data, fill in the missing data. If you clean the data you can use standard methods. There are multiple ways to clean, and cost could be high.

   I think the right thing to do is go directly to the next step, prediction.

Story: finance company wants to do data mining/machine learning. They find that 5% of the people were born Nov. 11, 1911. It's the only key you can press 6 strokes and get out of this field!

Ex. "Giants are more likely to be bilingual" because they entered their height in centimeters when asked for inches...

Consider some joint distribution D generating observable attributes x and labels y. An adversary corrupts x to produce z. We limit by the set of modification rules $\rho_i(y,x)$. The adversary can select the modification depending on x.

Notation:

- state of nature $y \in \{0,1\}$.
- signals $x \in X$
- distribution $D(y,x)$
- observed signals $z \in Z$.
- Modification rules $\rho_i(y,x) = z$, computed in polytime, m modification rules.

  For example, $\rho_i(y,x)$ flips signal i, $\rho_o(y,x)$ flips the odd signals.

The goal is given z, predict y.

The predictor, given z, predicts y, and in this way defines a policy $\pi(z)$.

The adversary selects ρ_i either

- statically: before x is selected.
- adaptively: after x is selected.

Model as a zero-sum game. Fixing policy π and modification rule ρ_i,

$$\mathrm{error}(\pi, \rho_i) = \mathbb{E}_{y,x}[\mathbb{P}[\pi(\rho_i(y,x)) \neq y]].$$

The optimal min-max error is

$$\mathrm{error}^* = \min_{\pi} \max_{\rho_i} \mathrm{error}(\pi, \rho_i).$$

Being optimal doesn't mean you're doing well: ex. the the adversary erases all inputs, and the predictor can't do anything. This means error* is very high.

The setting: Given a distribution D, known and computable, with set of possible corruptions:

For every D, there is an algorithm that given observable z, compute a prediction for y with probability of error error$^* + \varepsilon$, in time $\mathrm{poly}(n, m)$ in the static case, $\mathrm{poly}(n)\exp(m)$ (?) in the adaptive case.

We assume that in the training set examples are uncorrupted, in the test set examples are corrupted by adversary. Given hypothesis class H, given oracle for ERM in H, the goal is to select a mixture of hypotheses from H that minimizes the error.

Theorem 3.1.
1. *Given a sample S, we can efficiently compute a mixture which is ε-optimal on S.*
2. *For $|S| \geq \frac{\ln|H|/\delta}{\varepsilon^4}$, with probability $1-\delta$, get error(h) − obsError$(h) \leq \varepsilon$.*

References:

- Globerson and Roweis, nightmare at test time: robust learning by feature deletion
- Teo, Globerson, Roweis, Smola. Convex learning with invariances.

Where does it make a difference?

- Learning homogeneous hyperplanes $\mathrm{sign}(wx^T)$. Regular learning is $\min_w \mathbb{P}[(wx^T)y < 0]$. Large margin is $\min_w \mathbb{P}[(wx^T)y < \lambda]$. Corruption is setting one x_i to 0.

  Assume uniform distribution.

  A static adversary would zero the coordinate having largest weight. Success probability becomes $\min_w \mathbb{P}[(wx^T)y < \max_i |w_i|]$

  An adaptive adversary zeros out the coordinate having largest weight with correct sign. For $\mathrm{sign}(w_i x_i) = y$, get $\min_w \mathbb{P}[(wx^T)y < \max_i (x_i w_i)]$

Given ERM for class H, we'd like to learn a mixture of hypotheses $\Delta(H)$. Suppose that the number of corrupted inputs for x is $\leq m$. The error is

$$L(h) = \mathop{\mathbb{E}}_{x}[\max_{z \in \rho(x)} \ell(h(z), f(x))].$$

For any uncorrupted, consider the matrix M_x where $M_x(z,h) = \mathbb{1}(h(z) \neq y)$. (Here $z \in \rho(x), y = f(x)$.) The D selects the matrix.

The learner chooses $Q \in \Delta(H)$,

$$h_Q(z) = \sum_{h \in H} Q(h)h(z),$$

convex combination over h's. The learner doesn't know x, observes corrupted z, and generates a prediction. The adversary chooses $P_x \in \Delta(\rho(x))$, knows x, and generates $z \in \rho(x)$. They both play a mixed strategy. The goal of the learner is

$$\begin{aligned} Q^* &= \mathrm{argmin}_Q \mathop{\mathbb{E}}_{x}[\max_{P_x} P_x^T M_x Q] && (1) \\ \text{error}^* &= \min_Q \max_P \mathop{\mathbb{E}}_{x}[P_x^T M_x Q] && (2) \\ &= \max_P \min_Q \mathop{\mathbb{E}}_{x}[P_x^T M_x Q]. && (3) \end{aligned}$$

Note this doesn't take advantage of a mild adversary.

Think of $\min_Q \mathbb{E}_x[P_x^T M_x Q]$ as an ERM over the fixed distribution.

We would like the algorithm to choose few $h_i \in H$ to get good generalization.

Algorithm is based on regret minimization. Use a variant of exponential weights (Cesa-Bianchi-Mansour-Stoltz 2007). Maintain weights for $(z,(x,y))$ for each $(x,y) \in S$ and $z \in \rho(x)$. Output defines both Q and P. Expand sample by all possible corruptions.

Initialize weights $w_q(z,(x,y)) = 1$. Update weights as follows: given $h_t(\cdot)$, if $h_t(z) \neq y$, then $w_{t+1}(z,(x,y)) = (1+\eta)w_t(z,(x,y))$, else there is no change.

Normalize not for everything, but per x. Let P_t be the normalized w_t per x.

$$P_t(z,(x,y)) = \frac{w_t(z,(x,y))}{\sum_{z' \in \rho(x)} w_t(z',(x,y))}.$$

Given P_t use the ERM oracle to select h_t using $D^{P_t}(z,y)$:

$$\sum_{x: f(x)=y, z \in \rho(x)} P_x^t(z) D(x).$$

Use distribution to get next hypothesis to plug in.

Proof. The loss for $(z,(x,y))$ is

$$l_t(z,(x,y)) = l(h_t(z) \neq y) = M_x(z,h_t).$$

The cumulative loss of $(z,(x,y))$ is

$$L_T(z,(x,y)) = \sum_{t=1}^{T} l_t(z,(x,y)).$$

The algorithm loss is $L_T^{lin} = \sum_{t=1}^{T} P_t \cdot l_t$. Compare against the benchmark

$$L^* = \sum_{(x,y)\in S} \max_{z\in\rho(x)} L_T(z,(x,y)).$$

I finish learning, these are my hypothesis, what would you do? For each x choose the worst corruption.

The regret bound is

$$L^* - 2\sqrt{L^*|S|\ln m} \le L_T^{lin}.$$

The strategies are $P^* = \frac{1}{T}\sum_t P_t$, $Q^* = \frac{1}{T}\sum_t h_t$.

Theorem 3.2. *Fix uncorrupted S, $T \ge \frac{4|S|\ln m}{\varepsilon^2}$. P^* is ε-optimal for adversary and Q^* is ε-optimal for the learner.*

The proof is like for exponential weights.

$$W_{(x,y)}^t = \sum_{z\in\rho(x,y)} w_t(z,(x,y)) \ge (1+\eta)^{L^*(x,y)} \tag{4}$$

$$L^*(x,y) = \max_{z\in\rho(x)} L_T(z,(x,y)) \tag{5}$$

$$W^t = \prod_{(x,y)\in S} W_{(x,y)}^t \qquad \ge (1+\eta)^{L^*} \tag{6}$$

$$L^* - \eta L^* - |S|\ln m/\eta \le L_T^{lin}. \tag{7}$$

Multiplying the weights is the non-standard part. Take logs and do linear approximation.

Expected error given P, Q,

$$\begin{aligned}
R(P,Q) &= \sum_{x,y}\sum_{z}\sum_{h} P(z,(x,y))Q(h)I(h(z)\ne y) && (8)\\
\max_P \min_Q R(P,Q) &\ge \min_Q R(P^*,Q) && (9)\\
&\ge \frac{L_T^{lin}}{T} && (10)\\
&\ge \frac{L^*}{T} - \frac{2\sqrt{L^*|S|\ln m}}{T} && (11)\\
&\ge \max_P R(P,Q^*) - \frac{2\sqrt{L^*|S|\ln m}}{T} && (12)\\
&\ge \min_Q \max_P R(P,Q) - \frac{2\sqrt{L^*|S|\ln m}}{T}. && (13)
\end{aligned}$$

Using P^*, I'm getting as much as I can hope to get up to an additive factor. Q^* is ε-optimal for one side and P^* is ε-optimal for the other side.

To get generalization:

1. Let H^N be the class of averages of at most N $h_i \in H$. The first step is to limit N. It is enough to average $N_0 = O\left(\frac{1}{\varepsilon^2}\ln\left(\frac{1}{\delta}\right)\right)$. Use regular Chernoff bound.

2. Consider $h = \frac{1}{N}\sum_{i=1}^{N} h_i(z) \in H^N$. Bound the true and sample error by ε,

   $$\mathop{\mathbb{E}}_{(x,y)}[\max_{z\in\rho(x)} \ell(h(z), y)].$$

   This requires $|S| \geq \frac{1}{\varepsilon^4}\ln\left(\frac{|H|}{\delta}\right)$.

   The max in the $\mathbb{E}$ makes things more complicated. We bounded sample by $O\left(\frac{1}{\varepsilon^2}\right)$ and N_0 by $O\left(\frac{1}{\varepsilon^2}\right)$ so get $O\left(\frac{1}{\varepsilon^4}\right)$.

The final theorem: given hypothesis class H and ERM oracle for H, there is algorithm that with probability $1-\delta$ computes ε-optimal learning hypothesis in time $\mathrm{poly}(\left(\frac{1}{\varepsilon}\right), \left(\frac{1}{\delta}\right), \ln|H|)$.

Q: if adversary has finitely many choices, we can't model "choose error up to ε"—infinitely many choices?

A: First difficulty is showing the zero-sum game is well-defined.

This is far from robust statistics because I assume all inputs can be corrupted.

How to model robust statistics in this model: One modification rule is not modifying, the other is corrupt. Ex. put 98% on "don't modify".

4 Towards a Rigorous Approach to Approximate Computations in Huge-Scale Markovian Decision Processes (Csaba Szepesvari)

In this talk, without assuming any background beyond familiarity with basic probability and linear algebra, I will describe exciting new results on the computation of near-optimal policies in huge-scale Markovian decision processes (MDPs) via the so-called approximate linear programming methodology. In a nutshell, the new results provide meaningful upper bounds on the quality of the computed policies as a function of the linear basis functions chosen by the algorithm designer, while avoiding unrealistic assumptions which have effectively become the norm in prior results. As opposed to previous work that relied on worst-case reasoning over randomly sampled constraints, in this work the analysis is done using an operator-theoretic approach. I will finish by discussing the remaining main open problems, connections to alternative approaches, as well as potential extensions. Markov decision processes lie at the hear of many reinforcement learning algorithms and have found numerous applications across many engineering and scientific problems.

Based on joint work with Chandrashekar Lakshminarayanan and Shalabh Bhatnagar

For RL, one way to do things is: Let's learn a good model, run the planner, get a good policy. Does this work—do we have good planners?

Let

- S be state space
- A be action space

- P be transition probabilities (for every state and action, a distribution over next states, $P = (p_{sa} \in \Delta_n S), (s,a) \in S \times A)$.
- g be rewards, $g = (g_{sa} \in [0,1])$
- $\alpha \in (0,1)$ discount factor.

Classically, people think of S, A as finite. This is wrong: the number of states and actions in any reasonable space is exponential. You can't run algorithms even linear in the number of states and actions. We want sublinear scaling.

We want to maximize expected total reward

$$\mathbb{E}\left[\sum_{t=1}^{\infty} \alpha^t g_{S_t A_t}\right]$$

where $S_0, A_0, S_1, A_1, \ldots$ is the sequence of states and actions. [2]

Here we don't worry about exploration; we assume we can access the parameters.

A policy is a function $u : S \to A$: look at the history and decide what action to take. The value of the policy is the expected total reward under the policy

$$J_u(s) = \mathbb{E}_\mu\left[\sum_{t=0}^{\infty} \alpha^t g_{S_t,A_t} | s_0 = s\right] \tag{14}$$

$$\sup_u J_u(s) = J^*(s). \tag{15}$$

The optimal policy is the u^* such that $J_{u^*}(s) = J^*(s)$.

There always exists a policy that doesn't look at the past history, and just acts based on the current state, and is deterministic.

Theorem 4.1. *There $u^* : S \to A$ such that $J_{u^*} = J^*$.*

Assume you have an oracle MDP that given input s, a, outputs g_{sa}, p_{sa}. Assume also that given $J : S \to \mathbb{R}$, the oracle MDP gives $p_{sa}^T J$. (If you have sampling, you can estimate this. There's not that much difference between being able to sample and having this.)

In reality, maybe the regression/classification function is very complex. Instead, we ask: can we compete with a good function?

Make a hypothesis space for the optimal value function. Consider a feature function $\varphi : S \to \mathbb{R}^d$, and look for

$$J^*(s) \approx \varphi^T(s) r^*, \quad r^* \in \mathbb{R}^d.$$

[3]

Ex. Consider $S = [0,1]^n$, $1 < |A| < \infty$.

Assume p_{sa}, g_{sa} is L-Lipschitz in L_1.

[2] How strong do we believe in discounting?
We don't believe in it, we use it because it's convenient.
If you can do things with the discounted factor, you can probably do it without; this is the entry problem.

[3] Debate: Assume something about model vs. assume something about solution (what we do here).

1. Chow-Tsitsiklis, negative result: To get $J_u \geq J^* - \varepsilon$, in the worst case we need $\left(\frac{1}{\varepsilon}\right)^{\frac{n}{L}}$ queries.

2. J. Rust: "Randomization, breaking the curse of dimensionality."

   Polynomial-time computation up to ε in expectation. Rust's model is different.

   What's missing? Computational complexity depends on the largest value in the density function, typically exponentially large. Can I rejection sample by sampling uniformly? [4]

I want to compute policies!

This line of work goes back to the 70's.

Schweitzer-Sneider: mix linear programming with linear function approximation.

You can find an optimal policy by solving a LP: take $c \in (0, \infty)^s$ and solve

$$\min_{J \geq TJ} c^T J$$

where

$$(TJ)(s) = \min_a (g_{sa} + \alpha p_{sa}^T).$$

We know the optimal function is a fixed point $TJ^* = J^*$.

Define T_u by taking the action suggested by u: for $J : S \to \mathbb{R}$.

$$(T_u J)(s) = g_{su(s)} + \alpha p_{su(s)}^T J.$$

Let u be such that $T_u J = TJ$. The algorithm finds

$$\|J_u - J^*\|_\infty \leq \frac{\|J - J^*\|_\infty}{1 - \alpha}.$$

This is a multiplicative blowup of the error, a weak reassurance.

Let's focus on calculating some approximation of J^*.

We add an additional constraint

$$\min_{J \geq TJ, J = \Phi r} c^T J$$

where $r \in \mathbb{R}^k$, $k \ll |S|$.

We would like: if Φ is chosen well, solution of LP will not be far from best approximation of J^*.

The hypothesis space may be crappy, we learn relative to the hypothesis space.

Let $T_a J(s) = g_{sa} + \alpha p_{sa}^T$. (Always take action a.)

Is this even feasible? If $\mathbb{1} \in \operatorname{span} \Phi$, this is feasible. We can prove something stronger. Given $J \in \operatorname{span}(\Phi)$, if there exists $J' \in \operatorname{span}(\Phi)$, $J' \geq TJ'$ such that $\|J - J'\|_\infty \leq$

[4] Think of this as saying: MDP is a noisy system with no sinks, where you have no strong control. "If you have no control you can't be too stupid."

$\frac{(1+\alpha)\|J-J'\|_\infty}{1-\alpha}$, then there exists $r \in \mathbb{R}^k$, $J = \Phi r$, $\mathbb{1} \in \text{span}(\Phi)$, $\exists r_0 \in \mathbb{R}^k$, $\mathbb{1} = \Phi r_0$, then for $\lambda \in \mathbb{R}$, $\lambda \geq 0$,

$$J' = \Phi(r + \lambda r_0) \geq \max_a T_a(r + \lambda r_0) \tag{16}$$

$$T_a(r + \lambda r_0)(s) = g_{sa} + \alpha p_{sa}^T \phi(r + \lambda r_0) = (g_{sa} + \alpha p_{sa}^T \Phi r) + \alpha\lambda. \tag{17}$$

Find the smallest value of λ that satisfies this: how much you have to travel to meet the feasibility constraint.

$$\Phi r + \lambda 1 \geq T\Phi r + \alpha\lambda 1. \tag{18}$$

Is it tractable to minimize the objective? We group

$$c^T \Phi r = \left(\sum c(s)\varphi(s)\right)^T r.$$

One popular choice for φ is where every feature depends on a few state space variables. The constraint is $J(s) \geq g_{sa} + \alpha p_{sa}^T$.

Algorithm takes as input Φ, c, consoluts the oracle, has a way of calculating $c^T\Phi$, and outputs $\widehat{r}$.

Can we evaluate whether constraints are met?

$$(\Phi r)(s) \geq g_{sa} + \alpha(p_{sa}^T \Phi)r.$$

There are as many as $S \times A$, too many.

D, Van Roy: We can't solve LP, just sample. LP: If you know the k constraints, you can just choose them. Error scales polynomially in the right quantities. Blowup depends on misalignment between sampling measure and probability measure induced by the optimal policy.

If I'm in a huge state space, this mismatch will likely be huge.

Project/linearly combine constraints: RALP

$$\min_{W_a^T J \geq W_a^T T_a J, T_a J = \Phi r} c^T J.$$

Think of this as keeping just a few constraints. How does this error propagate to the result?

Theorem 4.2. *Let $\widehat{J}$ be the solution to the RALP, $c \in \Delta_1(S)$. Then*

$$\left\|J^* - \widehat{J}\right\|_{1,C} \leq \frac{C}{1-\alpha}\left(\inf_{r\in\mathbb{R}^d} \|J^* - \Phi r\|_\infty + \left\|\Gamma J^* - \widehat{\Gamma} J^*\right\|_\infty\right)$$

where

$$\Gamma J(s) = \min\left\{(\Phi r)(s) : \Phi r \geq T\Phi r, r \in \mathcal{N}\right\} \tag{19}$$

$$(\widehat{\Gamma} J)(s) = \min\left\{(\Phi r)(s) : W_a^T \Phi r \geq W_a^T T_a J, r \in \mathcal{N}\right\} \tag{20}$$

$$(\Gamma J^*)(s) = \min\left\{(\Phi r)(s) : \Phi r \geq J^*, r \in \mathcal{N}\right\} \tag{21}$$

$$(\widehat{\Gamma} J^*)(s) = \min\left\{(\Phi r)(s) : W^T \Phi r \geq W^T J^*\right\}. \tag{22}$$

(Once you drop some constraints, you can be unbounded, so we need $\mathcal{N}$.)

When you think about dropping the constraints, there are two things that are important.

Keep sufficiently many states, so for any new state you encounter, if you express as linear combination of features, coefficients should be small. This is like a subset selection problem: remaining ones can be expressed with linear combinations with small coefficients.

5 Sequential information maximization (Hamed Hassani)

Abstract: Optimal information gathering using uncertain observations is a central challenge in many disciplines of machine learning such as Bayesian experimental design, automated diagnosis, active learning, and decision making. A widely used method is to perform sequential observation selection, where the choice of the next observation depends on the history seen thus far. Despite the importance and widespread use in applications, little is known about the theoretical properties of sequential observation selection policies in the presence of noise. In particular, a long-open direction has been to analyse the persistent-noise setting that is arguably more relevant in practical applications. In this talk, we will present a new framework to capture the role of noise, and explain how it leads to the first rigorous analysis of the famous information-gain policy. We will then consider more general information gathering settings and provide new efficient algorithms, with provable guarantees, to deal with noisy observations.

Joint work with Yuxin Chen, Andreas Krause, and Amin Karbasi

5.1 General setup (Bayesian)

Let $Y \in \{y_1, \ldots, y_n\}$ the set of hypotheses, $X_1, \ldots, X_m$ the set of tests. The joint distribution is $P_{Y,X_1,\ldots,X_m}$.

Ex. sensors, recommendations, medical diagnoses (diseases, medical tests). A doctor wants to find out what diseases the patient has using tests. The first approach is the offline approach, select a susbet of tests and test. Based on all the results, decide what disease the patient case. In the adaptive setting, select one test; based on the result, choose the next test, and so on. Choose based on history. (Assume the tests don't affect the subject.)

Start with a uncertain prior. By performing a series of tests, we want to reduce uncertainty, have the posterior concentrated on one hypothesis.

Entropy measures uncertainty

$$H(Y) = -\sum_i p_{y_i} \ln p_{y_i}.$$

A uniform distribution has high entropy, a concentrated distribution has low entropy.

5.2 Information-gain heuristic

The information gain heuristic is a sequential greedy policy: at each step have a posterior $Y|$history with entropy $H(Y|\text{history})$. For each test X_i, the information that X_i provides is

$$H(Y|\text{history}) - H(Y|\text{history}, X_i) =: I(X_i; Y|\text{history}).$$

It is the difference in entropy before observing X_i and after observing X_i. The greedy policy is to pick the most informative test at each step.

The information-gain policy has been used since the 50's but there was no rigorous analysis.

The model is the conditionally independent Bayes model: conditioned on Y, the outcome of the tests $X_1, \ldots, X_m$ are independent. We know distributions $p_j(X_j = x_j|Y = y_i)$. People have analyzed this under the deterministic case and non-persistent noise case. (Mistakes are not systematic; you can make another query that's independent.)

One simple practical example if outcome of tests are noisy versions of the true outcome which is a deterministc function of Y.

A policy π is a decision tree; the next test is a function of the observations so far. We consider policies of length k. Initially there is some entropy $H(Y)$. At the end the entropy on average is $H(Y|\pi)$. We aim to maximize information gained by the policy,

$$I(\pi; Y) = H(Y) - H(Y|\pi).$$

We want

$$I_{\text{OPT}[k]} = \max_{\pi\in\Pi_k} I(\pi; Y)$$

the optimal amount of information after k steps. We compare with

$$I_{IG[l]} = I(\pi_{IG[l]}; Y)$$

the gain of the information-gain heuristic after l steps. How large should l be so that we can approximate $I_{\text{OPT}[k]}$?

Theorem 5.1. *To gain $(1-\alpha)I_{OPT[k]}$ it is sufficient to run IG for $O(k \cdot \frac{\ln n \ln\left(\frac{1}{\alpha}\right)}{S})$.*

Note in the offline noiseless setting this is a submodular optimization problem which is NP-hard.

Here S is the separability parameter,

$$S_i = \left(\min_{y,y':p_i(\cdot|y)\neq p_i(\cdot|y')} |p_i(\cdot|y) - p_i(\cdot|y')|_{TV}\right)^2, \tag{23}$$

$$S = \min_i S_i, \tag{24}$$

how well we can distinguish between any 2 hypotheses.

5.3 Analysis on conditionally independent model (proofs, insight)

Let G_l be the greedy algorithm after l steps. The gain is $I(G_l; Y)$. We want to compare with the optimal policy π_k^* which achieves $I(\pi_k^*; Y)$. We show

$$I(G_l; Y) \geq (1 - \exp(-cl/k))I(\pi_k^*; Y)$$

where $c = \frac{S}{\ln n}$.

To simplify the proof, assume each test has the following structure: There is a deterministic outcome D_i, and noise is added to this outcome—the bit is flipped with probability ε. Call the noise part N_i. Here $S_i = (1-2\varepsilon)^2$ (BSC). The observed outcome is X_i.

The proof has 2 main steps.

1. Relate 1-step greedy to OPT.

   Let $i^* = \text{argmin}_{i \in [m]} I(X_i; Y|\text{history})$.

   Lemma 5.2.

   $$\max_{i\in[m]} I(X_i; Y|history) \geq c\frac{\min_{\pi\in\Pi_k} I(\pi_k; Y|history)}{k}.$$

   LHS is the local gain; RHS is the global gain.

   Proof sketch of lemma: Relate the noiseless to noisy setting.

   (a) Consider the expected mass reduction by performing D_l, which is 1 with probability $1-p$. It is $2p(1-p)$.

   Pinsker's inequality gives $I(X_l; Y) \geq S_l p_l(1-p_l)$.

   (b) $I(\pi, N := \{N_i\}_{i\in M}; Y) \geq I(\pi; Y)$.

   (c) (Longest step) $I(\pi, N; Y) = k \ln n \underbrace{\max_{i\in[m]} s_i p_i(1-p_i)}_{\leq I(x_i;y)}$.

2. The amount of information that is gained at the ith level of greedy is

$$\begin{aligned}
I(G_i; Y) - I(G_{i-1}; Y) = I(G_i; Y|G_{i-1}) &= \mathbb{E}[I(X_{i^*}; Y)|G_{i-1} = g_{i-1}] && (25)\\
&\geq \frac{C}{k}\mathbb{E}[I(\pi_k^*; Y|G_{i-1} = g_{i-1})] && (26)\\
&= \frac{C}{k} I(\pi_k^*; Y|G_{i-1}) && (27)\\
&= \frac{C}{k}(H(Y|G_{i-1}) - H(Y|G_{i-1}, \pi_k^*)) && (28)\\
&\geq \frac{C}{k}(H(Y|G_{i-1}) - H(Y|\pi_k^*)) && (29)\\
&\geq \frac{C}{k}(H(Y) - H(Y|\pi_k^*) - (H(Y) - H(Y|G_{i-1}))) && (30)\\
I(G_i; Y) - I(G_{i-1}; Y) = I(G_i; Y|G_{i-1}) &\geq \frac{C}{k}[I(\pi_k^*; Y) - I(G_{i-1}; Y)] && (31)\\
\delta_i :&= I(\pi_k^*; Y) - I(G_i; Y) && (32)\\
\delta_i &\leq \left(1 - \frac{C}{k}\right)\delta_{i-1} && (33)\\
\implies \delta_l &\leq e^{-Cl/k}\delta_0. && (34)
\end{aligned}$$

5.4 Learning categories: new algorithms

Information gain heuristic could be arbitrarily bad. One such bad case is learning categories. We give new algorithms in this case.

Is the structural parameter S necessary? Yes. S can go to 0, making the bound arbitrarily bad.

Ex. find what category the hypothesis belongs to: type of disease, genre of movie. We want to learn Z depending on Y, which is much less complex than Y.

IG would choose the most informative test about Z. But S can become 0. Suppose hypothesis space has 4 hypotheses with $\frac{1}{4}$ probability. They are split into 2 groups, $Z =$green or red. Consider a test which cuts space into 2 parts, but doesn't inform you about Z. Then $I(X;Y) = 1$ but $I(X;Z) = 0$. So IG may fail arbitrarily badly compared to OPT.

We want to distinguish hypotheses in different categories. Transform hypothesis space, whose elements U are pairs of hypotheses y, y' with probability $p(y)p(y')$. We don't care about distinguishing y, y' in the same category. Merge those into one element. Learning Z is equivalent to learning U, $H(Z) = 0 \iff H(U) = 0$.

Pairwise information gain policy: At each step, choose the test that minimizes the uncertainty of U.

Ex. MovieLens-100k dataset. Objective: find category contains user's preference. Tests: given 2 movies, user picks one more similar to their preference.

6 Meta learning, black box optimization, and unsupervised learning (Ilya Sutskever, OpenAI)

The first part is on RL^2: fast reinforcement learning via slow reinforcement learning. The idea is to use a slow reinforcement learning algorithm to train a recurrent neural network policy that would, in effect, act as a fast reinforcement learning algorithm. We demonstrate the validity of this method by showing that it can learn to solve bandit problems and small tabular MDPs, and show that the method scales up reasonably well with today's RL techniques to maze navigation in 3d environments.

The second part is about Evolution Strategies, a derivative-free optimization method that has been rediscovered and explored by many research communities. We show that

1. it competitive with today's RL algorithms on standard RL benchmarks, and

2. it scales **extremely** well with number of workers.

This result is unexpected, because our models have hundreds of thousands of dimensions, and it wasn't obvious that Evolution Strategies would be successful on problems with so many dimensions.

I'll finish the talk with a brief overview of a few other research projects that are taking place in OpenAI.

6.1 Meta learning

RL^2: fast reinforcement learning via slow reinforcement learning.

How do humans do RL? Excellent data efficiency, prior knowledge.

What do we want? We want to solve lots of problems and become good at solving at problems as a result, learn to learn.

We can express prior experience as a distribution over environments (problems we've solved before). We've reduced meta-learning to supervised learning.

Given a distribution of environments, which RL algorithm does best?

Idea: Train an RNN policy to solve many environments.

Embed the RNN in different games and play.

RNN becomes policy and learning algorithm. At a high level it works because of supervised learning.

The RNN is the same in different environments. What changes are states of RNN.

You can go pretty far with stateless policy (state refers to state of RNN).

How does the slow RLer train the RNN?

You want to train a policy that solves a MDP, using policy gradients. $\pi_\theta(S)$ outputs a distribution over actions, $\pi_\theta(a|s)$. We have an MDP $P(s_t, r_t|s_{t-1}, a_{t-1})$. Use the MDP and policy together to generate sequence of states.

$$J(\theta) = \mathop{\mathbb{E}}_{s_1,r_1,a_1,s_2,\ldots}\left[\sum_t r_t\right].$$

There is a reasonably clean way to Monte Carlo estimate $\frac{\partial J(\theta)}{\partial \theta}$. Gradient descent takes you very far. You have to use some kind of algorithm.

Create composite MDP, distribution over environments. Reinterpret as learning on the fly.

A task is training point, and a training set is the set of training tasks.

The slow RL algorithm is trust region policy optimization. We want to train to become a fast RLer.

Take a multi-armed bandit problem. These have asymptotically optimal policies. For short horizons we should do better.

A bandit problem is determined by a vector of probabilities $p_1, \ldots, p_k$. On each episode, probabilities are randomly selected. RNN needs to know if the bandit returned a reward. (It remembers which does well.) n is the number of steps you take. Optimal strategy is only optimal asymptotically. When you only have 10 pulls you do better. If does better than Gittins for small horizons/number of arms, worse for large horizons/arms. (Gittins index is optimal for discounted case. The cost function is different.)

Now we have distribution over mazes. It can turn 3-D information into a representation of a map. This method is bottlenecked by fidelity of outer RL algorithm. Conceptually it has attributes of the right thing.

How is this limited? You have a distribution over tasks and want to do well on new tasks from the distribution—I think this is policy. Consider policy gradients; this algorithm does not make a distributional assumption. I'm not hiding a complicated distribution inside myself.

I think the truth is some kind of extrapolation between the two things. Ex. I'm talking to someone about topic, high level. Talking about details is painful. Pain is a sign the second algorithm is being activated.

Difference is orders of magnitude. We haven't done meta-learning on Atari. We need a much denser sample of games.

Is this how people learn how to play games?

We don't have a good way to benefit from planning in the general case.

There are various works on adapting the learning rate. Is this important? Use Adam. Learning rate is not dependent on tasks. Sample in iid manner. Give me a minibatch of tasks. Humans would learn 1 task well, then move onto a second, etc.

Do you observe differences in presenting order? In a learning system there are continuum of tasks. There are periods where you are in a very different environment.

Incremental learning: how to learn new without forgetting old. I think this is hard because there isn't any real problem that demands this. We don't do it because why do you need it? Once we have robots that have to learn to solve new tasks on the job, this will be central. But there is spam filtering, fake news detection—the environment adapts; you need to change in response. Practitioners prefer to retrain the whole thing because it's a predictable approach. You don't need it badly enough yet.

6.2 Black box optimization

Evolution strategies as a scalable alternative to reinforcement learning.

This is the simplest algorithm imaginable: add noise to parameters. If result improves, keep the change. Repeat.

Actually, sample a bunch of points and take a weighted average. In 2 dimensions it works just like gradient descent, but it's not obvious it works for high-dimensional problems.

For supervised learning it's always going to be worse, 1000 times slower on MNIST. There are advantages, in RL. If your reward signal is sparse, delayed... You want to learn policies that aren't very compact.

People have been using this for training low-dimensional policy all the time. You can train a model with 1000 parameters.

Evolution strategies is competitive with today's RL algorithms on standard benchmarks. It parallelizes really well. Have a bunch of workers. Each computes own noise vector, add to parameters. Then to communicate information. Normal distributed deep learning: all send gradients. With ES, just broadcast tiny scalars. All can agree in advance on random seeds for noise vectors. Communication is free here.

Variance decreases as you increase CPU cores. In distributed DL: at some point variance is low enough so this doesn't help. Here it always helps to reduce variance.

Cf. randomized finite differences.

Pick $v \sim N(0, I)$. This is an unbiased estimator of the gradient,

$$v\frac{F(x+\varepsilon v) - F(x)}{\varepsilon}.$$

This is only good for cases where you struggle with the real gradient. But in policy gradients there are things you can differentiate...

We don't have a clear example where this outperforms policy gradients. Policy gradients work better than we thought. Take away various things we thought were variance reduction techniques.

Policy gradients forces stochastic actions. Explore by iid noise in actions. Here explore by systematic change in policy. This may explore better.

What's the difference between this and other derivative-free optimization, particle swarm? Novelty is that old method works better than what most people thought.

Is this evolutionary strategies: sample neighborhood and go to point with best value. I think these formulations are identical.

$$J(\theta) = \mathop{\mathbb{E}}_{P(x|\theta)}[F(x)].$$

$P(x|\theta)$ is continuous. If $P(x|\theta) = N(\mu = \theta, \sigma = \varepsilon I)$, you get exactly $v\frac{F(x+\varepsilon v)-F(x)}{\varepsilon}$. (Also called 0-order optimization.)

Atari results: we can match one-day A3C on Atari with 1 hour of distributed implementation with 720 cores. We need 3x–10x more data; we don't need backward pass, twice as expensive as forward pass. In terms of actual compute it's not far off.

Better for exploratory, slower to learn overall? Is there a good interpolation?

One interpretation: policy gradients: take a different random action at each timesteps, decide. Learn random microdecision.

ES: Make 1 decision and stick with it the entire episode.

I think you want to make multiple random decisions, just not at micro-level.

Long horizons are hard for RL. ES is better at this than RL. It is relatively invariant to horizon length, action frequencies. Things like action frequencies have big effect on RL. You can have your agent make action every frame (1/6s), but this means it makes a lot of actions before it gets a reward. It's better in principle to force agent to stick with action.

ES may be more robust. Exploration at parameter level not at action level.

von Neumann architecture biases towards certain profiles of algorithms?

Dropout is basically the Bayesian method.

Speed of evolution strategies depends on the intrinsic dimensionality of the problem. If you run policy gradients in practice, as you make policies larger it gets harder to train.

Shouldn't derivative-free optimization become harder in higher dimensions?

Evolution strategies automatically discards the irrelevant dimensions, even when they live on a complicated subspace. We got a performance boost on going to bigger policies.

As long as there exist a subspace that determines the performance, optimization will be fast.

This works because it's a derivative method.

6.3 Unsupervised learning and the sentiment neuron

Unsupervised feature learning is the dream of pre-deep learning deep learning.

Train a big 4096 mLSTM (multiplicative LSTM) to predict next character in Amazon reviews. The result: model that learned excellent sentiment representation.

On Stanford sentiment analysis treebank.

Graph: Amount of supervised data needed to match other approaches. One neuron does most of the work!

Train classifiers on top of the state after the end of the sentence.

Trained on 40GB of text for a month on 4 GPU's.

Forcefully set neuron to positive, get positive sentiments. You can change one neuron to get completely different sentiment!

What are the conditions under which training gives the representation you want? Train your model to do one thing and it does well on a different thing.

There is some mysterious effect that would be useful to sort out.

Other cool stuff:

- more meta learning
- robotics
- lots of RL
- lots of unsupervised learning, generative models

Unsupervised learning: Shooting in dark: we don't have great theory/story for why this should work. Word2vec is a clear special case. In general you train a big model, why do you get a good representation? The fact that sometimes you do get that is mysterious.

If you try to model images, complicated, if your model doesn't do a good job of modeling room, floor, it won't care about faces.

I don't feel we can confidently predict whether an unsupervised feature learning algorithm can succeed.

Why are our brains so big? Classification models are small but need huge net for generative model?

Angry cats and happy dogs?

You can take word embeddings and do well; there's no mystery there.

Standard method: predict the next word, or missing words. Is there a different paradigm; predict something else you can get in an unsupervised way?

Make GAN's work for text.

It's possible the question is wrong. Put 2 baby cats on carousel; one cat can control the rod; the other cannot. The cat who doesn't get to move the rod doesn't learn to see. Action is important to learning. It creates natural place for causality.

7 Active classification with comparison queries (Shay Moran)

We study an extension of active learning in which the learning algorithm may ask the annotator to compare the distances of two examples from the boundary of their label-class. For example, in a recommendation system application (say for restaurants), the annotator may be asked whether she liked or disliked a specific restaurant (a label query); or which one of two restaurants did she like more (a comparison query).

We focus on the class of half spaces, and show that under natural assumptions, such as large margin or bounded bit-description of the input examples, it is possible to reveal all the labels of a sample of size n using approximately $O(\log n)$ queries. This implies an exponential improvement over classical active learning, where only label queries are allowed. We complement these results by showing that if any of these assumptions is removed then, in the worst case, $\Omega(n)$ queries are required.

Our results follow from a new general framework of active learning with additional queries. We identify a combinatorial dimension, called the *inference dimension*, that captures the query complexity when each additional query is determined by $O(1)$ examples (such as comparison queries, each of which is determined by the two compared examples). Our results for half spaces follow by bounding the inference dimension in the cases discussed above.

If time allows, we will discuss several directions for future research and some open problems.

Joint work with Daniel Kane, Shachar Lovett, and Jiapeng Zhang

7.1 Active learning

In active learning, get unlabeled data, ask for labels of a few data points, and predict as well as in supervised learning. When is this possible?

For example, for thresholds in 1D, do binary search.

Half-planes in 2-D: Given unlabeled points $x_1, \ldots, x_n \in \mathbb{R}^2$. Labels are $y_i = \text{sign}(\langle w, x\rangle - t)$. Any active algorithm requires $\Omega(n)$ queries. Ex. Take points in convex position; each can be separated from the rest by a line.

7.2 Active learning with additional (comparison) queries

We allow more interaction. What kind? It doesn't make sense to allow arbitrary additional queries. Asking arbitrary membership queries can be problematic (Lang, Baum).

A relative query is a query that holds relative information. “A vs. B?” “Is A more like B or C?”

Let $H = \{\text{sign}(f) : f \in F\}$ where F is a set of real functions, e.g. half-spaces and neural-nets. Label-query asks $\text{sign}(f(x_i))$ and comparison query asks $f(x_i) \geq f(x_j)$?

7.3 Learning half-spaces with comparison queries

Half-planes in 2D: This allows active learning with $O(\log n)$ queries.

1. Sample S of size 20.
2. Label S.
3. Find points in S closest to line.
4. Build cones
5. Label inputs in cones (must be correct)
6. Repeat on unlabeled points.

With probability $\geq \frac{1}{2}$ over samples, each iteration labels half of the remaining unlabeled points. Each iteration makes $O(1)$ queries, and so whp $O(\log n)$ are sufficient.

In $\mathbb{R}^3$ this fails. There are n points in $\mathbb{R}^3$ that require $\Omega(n)$ label/comparison queries for revealing all labels.

But we can do better if data has bounded bit complexity, $x_i \in \{0, \ldots, B\}^d$. Then all n labels can be revealed using $\widetilde{O}(d \ln B \ln n)$. Note $d \ln B$ is the size in bits of each example.

Theorem: Sample in $\mathbb{R}^d$ with margin γ. All n labels can be revealed using $\widetilde{O}(\ln(1/\gamma) \ln n)$ label and comparison queries.

Our result is actually with respect to $\frac{\min_i |f(x_i)|}{\max_i |f(x_i)|}$. This scales with f.

Can we get dimension independent bounds? No. There is a sample of n points in $\mathbb{R}^{n+1}$ with margin $\Omega(1)$ that requires $\Omega(n)$ label/comparison queries for revealing all labels.

7.4 Inference dimension

This is the combinatorial parameter we used to analyze.

Definition 7.1: The inference dimension of H is the minimal k such that every realizable sample of size k contains a point whose label can be inferred from the comparison and label queries on the remaining points.

Claim 7.2. *The inference dimension of thresholds is 3.*

If 3 points have the same labels, the midpoint can be inferred. If 3 points have different labels, an endpoint can be inferred.

Inference dimension of half-planes is at most 7 (in fact 5).

Must contain subsample of size 4 with same labels. The cone pointed at the nearest point is defined by 3 points. Any other point can be inferred.

Consider class of $H = \{\text{sign}(p) : \deg(p) \leq d\}$, where $p : \mathbb{R} \to \mathbb{R}$ is univariate polynomial. Open exercise: what is the inference dimension of H?

Theorem 7.3. *Let k denote the inference dimension of H. The labels of any realizable sample of size n can be revealed using $O(k \ln k \ln n)$ comparison and label queries.*

This can be extended to other types of additional queries.

Lower bound: There exists a realizable sample of size k that requires $\Omega(k)$ comparison/label queries.

Proof of upper bound:

1. Sample $4k$ points.

2. Query them. (Label and comparison)

3. Infer queries among unlabeled points.

4. Remove labeled points and repeat.

With probability at least half over sample, each iteration labels half of the remaining unlabeled points. Use boosting.

Inference dimension for bounded bit-complexity:

Theorem 7.4. *Consider (X, H), $X = \{0, \ldots, B\}^d$ and H the class of d-dimensional half-spaces. Inference dimension of (X, H) is $\widetilde{O}(d \ln B)$.*

Corollary 7.5. *There is a $2k$ linear decision tree for k-sum of depth $\widetilde{O}(n)$.*

Theorem 7.6. *Let H be class of d-dimension half-spaces with margin at least γ wrt X. The inference dimension of (X, H) is $\widetilde{O}(d \ln(1/\gamma))$.*

Future research:

- Relative queries
- Restricted membership queries. (Problem with just membership queries: what if data is on manifold, and query outside manifold?)
  - Use generative model $g : [0, 1]^k$ to generate meaningful membership queries.
- Streaming model
- Noisy comparisons
- Exact learning of threshold function on the boolean cube

Consider (X, H), $X = \{0, 1\}^d$, H is class of d-dimensional half-spaces. Inference dimension is $\widetilde{O}(d)$. Thus 2^d labels of every h can be revealed using $\widetilde{O}(d^2)$ label and comparison queries. This is info-theoretically tight: there are $2^{\Omega(d^2)}$ half-planes.

Question: can this be done efficiently? (The default algorithm is polynomial in space of data, 2^d.)

On boolean cube, we can find vertex with min value of linear function using linear programming. What about median vertex?

8 Deep submodular functions (Jeffrey Bilmes)

We start by covering how submodularity is useful in machine learning and data science (e.g., summarization, diversity modeling, tree-width unrestricted probabilistic models). We then show that while certain submodular functions (e.g., sums of concave composed with modular functions (SCMMs)) are useful due to their practicality, scalability, and versatility, they are limited in their ability to model higher level interaction. We thus define a new class of submodular functions, deep submodular functions (DSFs), and show that DSFs constitute a strictly larger class of submodular functions than SCMMs, but that they share all of the SCMM advantages. DSFs are a parametric family of submodular functions that share many of the properties of deep neural networks (DNNs), including many-layered hierarchical topologies, distributed representations, opportunities and strategies for training, and suitability to GPU-based matrix/vector computing. We show that for any integer $k > 0$, there are k-layer DSFs that cannot be represented by a k'-layer DSF for any $k' < k$. This implies that, like DNNs, there is a utility to depth, but unlike DNNs (which can be universally approximated by shallow networks), the family of DSFs strictly increase with depth. We show that DSFs, however, even with arbitrarily large k, do not comprise all submodular functions. For this talk, we will assume no prior background in submodularity and the talk will be self-contained. This is joint work with Wenruo Bai.

We define a class of functions useful in machine learning. These form a broader class within submodular functions.

Submodular functions are a vast family of functions. How to find the right function? Learn it. How to restrict the class of functions? We introduce a new class of submodular functions, retaining many aspects of more traditional classes, but which is broader. `http://www.arxiv.org/abs/1701.08939`.

Outline:

1. Submodular functions, ML, and data science
2. Feature based functions: sums of concave over modular (SCMMs)
3. Deep submodular functions
4. Matroid case
5. DSFs extend SCMMs
6. $DSF_k \supset DSF_{k-1}$
7. DSF's comprise not all submodular functions
8. Conclusion

You can think of DSF's analogous to neural networks. There exist 3-layer deep NN's can only be approximated by 2-layer deep NN with exponential number of units. Any continuous function can be approximated arbitrarily by 2-layer network. We have a strict $DSF_{k-1} \subset DSF_k$, even with arbitrary numbers of units. You lose a certain type of interaction.

Given a finite ground set of objects V, and a set function $f : 2^V \to \mathbb{R}$ (cf. pseudo-boolean function). For example $f(A)$ is the number of types of objects in A.

Let V be a finite set and $A \subseteq B \subseteq V$ be subets, and $v \notin B$, $f : 2^V \to \mathbb{R}$ be an information function.

- Non-negativity $f(A) \geq 0$ for all $A \subseteq V$.
- Monotonicity $f(B) \geq f(A)$: more items can only have more information (nice but not necessary).
- How much does A tell you about item v?

  $$\text{KnowledgeOf}(v|A) := 1 - \frac{f(A+v) - f(A)}{f(v)}.$$

  More data should tell you no less:

  $$\text{KnowledgeOf}(v|A) \leq \text{KnowledgeOf}(v|B).$$

  This is true iff f is submodular.

KnowledgeOf isn't negative if monotonicity holds.

Let A be indices. Take submatrix with those row/column indices, take determinant. This function is submodular and not monotone.

$f : 2^V$ is submodular if for any $A, B \subseteq V$,

$$f(A) + f(B) \geq f(A \cup B) + f(A \cap B) \tag{35}$$

$$\iff f(A \cup \{v\}) - f(A) \geq f(B \cup \{v\}) - f(B) \tag{36}$$

(submodular concave, diminishing returns formulation, respectively).

Ex. frienship is submodular: if you have 1000 friends, the 1001st friend is not so valuable. If you have 0 friends, the 1st friend is valuable.

Ex. Value in urn is diversity of balls in urn. This is set cover.

There are many more, for example, entropy, (McGill, 1954)

$$f(A) := H(X_A) = -\sum_{x_A} p(x_A) \ln p(x_A).$$

Further conditioning reduces entropy, $H(X_A|X_B) \geq H(X_A|X_B, X_C)$, equivalent to submodularity. This is distinct from subadditive $H(X_A) + H(X_B) \leq H(A \cup B)$.

(Ex. Submodular functions are useful for choosing a lesson plan for human learning. Sometimes you want more redundancy; sometimes you want diversity. cf. promise of MOOCs.)

Entropy requires additional inequalities, so area strict subset of nonnegative normalized submodular functions. Open: characterize what is the stuff outside the entropy functions. People keep finding more complicated entropy functions. Computer simulation found all these inequalities. What is the intersection of entropy and deep submodular functions?

Sums of weighted concave composed with additive functions, matroid rank, quantum entropy, log determinant, spectral functions applied to Hermitian matrix, certain Kolmogorov complexities...

We can define a notion of combinatorial independence

$$f(A \cup B) = f(A) + f(B).$$

Conditional independence given C,

$$f(A \cup B \cup C) + f(C) = f(A \cup C) + f(B \cup C).$$

Dependence

$$f(A|B) = f(A \cup B) - f(B) < f(A).$$

Conditional mutual info

$$I_f(A; B|C) = f(A \cup C) + f(B \cup C) - f(C)$$

Information amongst collection of sets

$$I(\{S_i\}) = \sum f(S_i) - f(\bigcup S_i)$$

Suppose defined on edges of some graph. Might constraint to be edge cover, vertex cover, have bounded cardinality (ex. summarization).

Say $x \in \{0,1\}^n$. You can define $P(x) = \frac{1}{Z}\exp(-f(x))$. Often underlying complexity is treewidth. But submodular functions can be defined that don't have small treewidth, but you can find MAP (though you can't find marginals).

Computer vision: Kolmogorov, 90's: look at Markov random field. When can MAP cut find solution? Precisely when function is submodular.

Any energy function can be written as a difference of submodular functions $\frac{1}{Z}\exp(g(x) - f(x))$. Sometimes you can get approximation guarantees.

Many applications to machine learning.

- Active learning (batch, online, adaptive): Choose more diverse batches
- Data summarization (document, image, video, music)
- Training data set reduction
- Test data set reduction (ex. one concept is over- and one is underrepresented)
- Genomic summarization (drug trials, ensure all groups represented)
- Feature selection (remove redundant features beforehand)
- Model selection (batch boosting)
- Data partitioning (allocate training data to different processors for parallel training)

More data is like no more data. This is something that also happens for nonparametric models. This happens even though you optimize complexity of model for amount of data. Each graph seems like concave function. If any way you permute data and it's concave, then it's a submodular function.

Let $V = [n]$ be finite ground set, $f : 2^V \to \mathbb{R}$ be set function. If for all permutations σ of V, for all $i \le j$,

$$f(\sigma_i|S_{i-1}) \ge f(\sigma_j|S_{j-1})$$

where $S_i\{\sigma_1, \ldots, \sigma_i\}$ then f is submodular.

Data is highly redundant!

You want to order data so accuracy goes up quickly and flattens out, so you can throw out the rest.

Cf. core-sets. Here we don't know the precise problem; we're trying to find a core-set on particular model of that data.

1. Whenever one is dealing with multi-pass ML systems, often the number of passes is large. Summarization methods are just used once. From the summary you can do model selection, etc.
2. Other algorithms for submodular functions, for exampe in the streaming setting...

How to find the right submodular function for your ML application?

- hand design the function f.
- learn f unconstrainedly.
- Learn f within c(1) onstrained degree family or (2) flexible constrained parametric family. We consider (2).

8.1 Feature based functions: sums of concave over modular (SCMMs)

A modular function is given by a vector,

$$m(A) = \sum_{a \in A} m(a).$$

It's normalized, $m(\phi) = 0$. Characterization:

$$m(A) + m(B) = m(A \cup B) + m(A \cap B).$$

Concave over non-negative modular $f(A) = \phi(m(A))$ is submodular.

If $f_1, \ldots, f_l$ are a set of submodular functions, then

$$f(A) = \sum_{i=1}^{l} \alpha_i f_i(A) + m_{\pm}(A)$$

is submodular. When $f_i(A) = \phi_i(m_i(A))$ is concave where ϕ_i, m_i are concave, modular, respectively, this is a SCMM.

What can you do with this class?

Feature functions: Use SCMM's readily.

Only requirement is that features be non-negative.

Ex. $m_u(v)$ counts number of times "New York City" appears in document. Ex. U is set of colors and $m_u(v)$ is number of pixels of color u.

For $X \subseteq V$, define $m_u(X) = \sum_{x \in X} m_u(x)$. With g nondecreasing concave, $\phi(m_u(X))$ grows subadditively.

$$\phi(m_u(A+v)) - \phi(m_u(A)) \geq \phi(m_u(B+v)) - \phi(m_u(B)).$$

Consider

$$f(X) = \sum_{u \in U} \alpha_u \phi_u(m_u(X)) + m_{\perp}(X).$$

Ex. feature counts (9,0,0), (3,3,3), (3,4,2). Adding diversity we get a boost.

When you consider a collection of sets of features, it's better for each set to be diverse in features.

Feature-based functions can be learnt on training set and tested on separate test set. Useful for streaming applications (evaluations do not require entire ground set).

Maybe you squares and triangles are more similar to each other than to circles.

$$F(A) = \sum_{u \text{ shape}} \sqrt{m_u(A)} = \sum_u \sqrt{\sum_{a \in A} \text{count}_u(a)}$$

doesn't allow higher-order interaction between shapes. Contribution of square is measured combinatorially independent of triangle. Put square root over!

$$\overline{F}(A) = \sqrt{\sum_{u \in \{\triangle, \square\}} \sqrt{m_u(A)}} + \sqrt{m_{\bigcirc}(A)}$$

8.2 Deep submodular functions

Add additional layer of nested concave functions

$$f(X) = \phi_{v^K}\left(\sum_{v^{K-1} \in V^{K-1}} w^{(K)}_{v^K}(v^{K-1}) \phi_{v^{K-1}}(\cdots)\right).$$

More generally we can define deep submodular functions recursively given a DAG.

$$\psi_V(x) = \phi_v\left(\sum_{u \in \text{parent}(v) \setminus V} w_{uv} \phi_u(x) + \langle m_v, x \rangle\right).$$

DSF is defined as $\psi_v(1_A) + m_\pm(A)$ where $m_\pm : V \to \mathbb{R}$ is arbitrary modular function.

Any deep neural net with nonnegative weights and concave function in nonnegative region is DSF.

The mother concave function is

$$\phi(x) = \min(x, 1).$$

DSF's may leverage work on DNNs (GPU-based training, distributed representations, toolkits, etc.).

Do DSF's constitute larger class? Where do family of DSF's lie in space of submodular functions?

Theorem:

1. family of DSFs is strictly larger than family of SCMMs.
2. family of k-layer DSFs strictly extends the family of $k-1$ layer DSFs for all $k \geq 0$. (0-layer is modular.)
3. Family of k-layer deep submodular functions over all k do not comprise all submodular functions.

Open:

1. Is there an approximation result?
2. If one restricts curvature, can $k-1$ layer approximate k layer functions? (cf. Lipschitzness. It's defined differently, seen and used as complexity parameter. Worst-case results come from fully curved functions. For less curved functions you get better approximation bounds.)

8.3 Matroid case

In matroid case: partition matroid rank. I show laminar is greater than partition matroid rank.

A matroid M is a set system $M = (V, I)$ where $\mathcal{I} = \{I_1, I_2, \dots\}$ where $I_i \subseteq V$ are called independent. Example: set of subsets of linearly independent vectors in a set of vectors.

$\phi \in \mathcal{I}$ and $\mathcal{I}$ is subclusive (any subset is in I), and all maximally independent sets have the same size. Given $A, B \in \mathcal{I}$ with $|A| < |B|$, there exists $b \in B \backslash A$ such that $A + b \in \mathcal{I}$. Rank of matroid is $r : 2^V \to \mathbb{Z}_+$, $r(A) = \max_{I \in \mathcal{I}} |I \cap A|$.

Partition matroids: partition $(V_1, \dots, V_l)$ of V is formed with capacities $k_1, \dots, k_l \in \mathbb{Z}_+$. Rank function is

$$r(X) = \sum_{i=1}^{l} \min(|X \cap V_i|, k_i)$$

a SCM.

Laminar matroid: For all $i \neq j$, either $F_i \cap F_j = \phi$ or $F_i \subseteq F_j$ or $F_j \subseteq F_i$. (Sets are nonintersecting or comparable.)

"Deep matroid rank function."

Each $F \in \mathcal{F}$ has associated capacity $k_F \in \mathbb{Z}_+$. Each laminar family corresponds to a tree.

The rank function takes the following form

$$r_F(A) = \min\left(\sum_{F' \in pa(F)} r_{F'}(A \cap F') + \left|A \backslash \bigcup_{F' \in pa(F)} F\right|, k_F\right).$$

(DAG is a tree.)

Lemma: Laminar matroids strictly generalize partition matroids.

Proof. Consider $\mathcal{F} = \{V, B\}$ where $k_V = 2$, $B \subset V$, $k_B = 1$, $|B| \geq 2$, $|V| \geq |B| + 2$, $r(X) = \min(\min(|X \cap B|, 1) + |X \backslash B|, 2)$.

Suppose

$$r_s(X) = \sum_i \min(|X \cap C_i|, k_i).$$

C's are disjoint. Choose $b_1 \in C_1, b_2 \in C$ for $i \neq j$. Then $r_s(\{b_1, b_2\}) = 2 \neq r(\{b_1, b_2\}) = 1$. Hence there is unique i, $B \subseteq C_i$. Then $k_i = 1$. Moreover $C_i = B$.

9 Diving into the shallows: a computational perspective on Large-Scale Shallow Learning (Misha Belkin)

Remarkable recent success of deep neural networks has not been easy to analyze theoretically. It has been particularly hard to disentangle relative significance of architecture and optimization in achieving accurate classification on large datasets. Rather than attacking this issue directly, it may be as useful to understand the limits of what is achievable with shallow learning. It turns out that the smoothness of kernels typically used in shallow learning presents an obstacle toward effective scaling to large datasets using iterative methods.

While shallow kernel methods have the theoretical capacity to approximate arbitrary functions, typical scalable algorithms such as GD/SGD drastically restrict the space of functions which can be obtained on a fixed computational budget. The issue is purely computational and persists even if infinite data are given.

In an attempt to address this issue we introduce EigenPro, a preconditioning technique based on direct (approximate) spectral modification. Despite this direct approach, EigenPro was able to match or beat a number of state of the art classification results in the kernel literature (which typically used large computational resources) with only a few hours of GPU time. I will continue with some speculation on the architecture/optimization dichotomy in deep and shallow networks. Finally, I will argue that optimization analysis concentrating on the optimality within a fixed computational budget may be more reflective of the realities of machine learning than the more traditional approaches based on rates/complexity of obtaining a given error.

Based on joint work with Siyuan Ma.

Classical shallow learning:

1. Carefully choose a space of functions H, e.g., a kernel
2. Carefully apply appropriate regularization.
3. Solve the (convex) optimization problem.

Modern deep learning:

1. Choose architecture (not that it matters too much).
2. User regularization (does not matter too much).
3. Run SGD for a while on some GPUs.
4. Obtain better results.

What changed?

For small problems, there is no problem that neural nets solve that cannot be solved with kernels, boosting, etc. We are in the modern big data era.

Empirical risk minimization:

$$\text{argmin}_{f \in H} \frac{1}{n} \sum_{i=1}^{n} L(f(x_i), y_i) + S(f)$$

where S is regularization. (I argue we should not use regularization; it's done for free by optimization.)

How does big data help?

1. Better averaging.
2. More coverage/better approximation.

The second point has a lot to do with the success of big data.

Small data allows richer optimization, big data allows richer function space. Our optimization algorithms are now restricted, however. For example, you can't use 2nd order optimization.

We want a large space H to get small empirical loss, and efficient optimization.

1. Nearest neighbor: gets 2×Bayes error.

   MNIST Bayes error needs to be $< 0.5\%$.

   But 1-NN MNIST Bayes error is $\approx 3\%$.

   We have a high-dimensional group action. If the different manifolds are close together, a point on a different manifold may be closer than a shift on the same manifold.

Consider something simpler: least squares (kernel) shallow networks. We'll explain and address one aspect of why shallow networks underperform on large data.

Computational reach: What kind of functions can be approximated on limited computational budget?

Expanding the reach: Learn a kernel to make computation (gradient descent) efficient. It's a preconditioning technique.

Optimization in ML is different from classical optimization:

- hard to know how much is desirable (optimization error is not generalization error).
- Some error is OK, avoid overfitting.
- Computationally limited.

Realistic goal: do best within fixed computational budget.

Shallow architecture is feature map $\phi : \mathbb{R}^d \to H$ followed by linear regression/classification

$$w^* = \text{argmin}_{w \in H} \frac{1}{n} \sum_i L(w \cdot \phi(x), y).$$

We may as well replace x by $\phi(x)$ in the data. Exact solution is

$$w^* = (X^T X)^{-1} X^T y$$

(can replace inverse by pseudoinverse). You can't do this for big data $n \approx d \approx 10^6$. We can't afford 10^{18} flops. But $O(n^2)$, $10^3 \cdot 10^{12}$ is probably OK.

We have to do gradient descent (Richardson iteration)

$$w \hookleftarrow w - \eta(X^T X w - X^T y).$$

What happens with this? The standard step size is $\frac{1}{\lambda_1}$. Write in eigenbasis of $X^T X$,

$$(w^* - w^{(t)}) \cdot e_i = \left(1 - \frac{\lambda_i}{\lambda_1}\right)^t (w^* \cdot e_i).$$

The computational reach after t iterations is

$$CR_t = \left\{ v : \sum \left(\left(1 - \frac{\lambda_i}{\lambda_1}\right)^{2t} v_i^2 \right) < t^2\varepsilon^2 \right\}.$$

When $\lambda_1 \gg \lambda_i$, only a small fraction of the parameter space is reachable after $t < \frac{\lambda_1}{\lambda_i}$ iterations. You need a lot of iterations to converge in that direction. When $\frac{\lambda_i}{\lambda_1} < \frac{1}{t} \ll 1$,

$$\left(1 - \frac{\lambda_i}{\lambda_1}\right)^{2t} \approx 1 - \frac{2t\lambda_i}{\lambda_1}.$$

We need $t = O\left(\frac{\lambda_1}{\lambda_i}\right)$ iterations to converge along each e_i, nothing to do with amount of data.

Think of t iterations of GD as spectral thresholding at $\lambda_i < \frac{\lambda_1}{t}$.

Consider kernel regression

$$w^* = \operatorname{argmin}_w \frac{1}{n} \sum_{i=1}^n \left(\sum_{j=1}^n w_i K(x_j, x_i) - y_i \right)^2.$$

We can fit any data, e.g., with Gaussian kernel. But Gaussian (smooth) kernels have exponential eigenvalue decay $t \approx \frac{\lambda_1(K)}{\lambda_i(K)} \approx e^{i^2/d}$.

This is good because gradient descent performs regularization (cf. early stopping), but it has too much regularization, effectively spectral thresholding at $i \approx \ln t$.

Adding regularization $\tau \|f\|^2$, condition number is now λ_1/t. Do we have better convergence?

Problem: bias. Regularization is useless.

Does all of this matter? Consider the Heaviside function. It's a nice function. With m points I can have $\frac{1}{m}$ error.

What can I get using my regularization? Write in Fourier basis,

$$\sum a_n \sin nx + b_n \cos nx, \quad b_n = 0, \quad a_n = \frac{1 + (-1)^n}{n}.$$

Basically $a_n \sim \frac{1}{n}$.

Take the heat kernel $e^{-t\Delta}$. (Δ is the Laplacian, second derivative.) In terms of eigenfunctions, this is $\sum e^{-t\lambda_n}(\sin nx + \cos nx)$, where $\lambda_n = \Theta(n^2)$. This is because $(\sin nx)'' = -n^2 \sin nx$. This is basically Gaussian kernel.

After n iterations, I get $\sqrt{\ln n}$ coordinates.

Even taking something with e.g. 3 derivatives, we only get polynomial decay, does not work with exponential decay. Things are worse with higher dimensions.

On real data, need 4000 iterations to fit 200 eigenvectors. When you take more eigenvectors, the more you take, the better result you have. On real data, you need a lot of epochs (not exponential, but large). Taking a smaller σ you get better results on training but large test error. Small σ is basically doing 1-NN.

Sharp kernels have small eigendecay; you overfit and don't get good results.

Preconditioning: $Kx = y \implies PKx = Py$.

EigenPro. Idea: split computational budget by

1. compute first k eigenvectors

2. Use GD after shrinking first k eigendirections (partial whitening).

Equivalent to preconditioned Richardson iteration,

$$w \hookleftarrow w - \eta P(X^T X w - X^T y),$$

$P = I - \sum_{i=1}^{k} \left(1 - \frac{\lambda_{k+1}}{\lambda_1}\right) e_i e_i^T$. Potential acceleration is $\eta^{-1} = \frac{\lambda_1}{\lambda_{k+1}}$.

Note we need to combine first and second order approximation (not just second order methods).

Small eigendirections with small eigenvalues have important information. You have to keep the information, but it's important to use second-order information.

Here's what the new kernel looks like

$$K(x,z) = \sum_{i=1}^{\infty} \lambda_i e_i(x) e_i(z) \tag{37}$$

$$K_{EiP}(x,z) = \sum_{i=1}^{k} \lambda_{k+1} e_i(x) e_i(z) + \sum_{i=1}^{k+1} \infty \lambda_i e_i(x) e_i(z). \tag{38}$$

Spectral modification.

Why EigenPro?

1. Compute first k of $X^T X$ eigenvectors to form P using Randomized SVD.

2. Approximate computation of P is OK as solution does not change.

3. Using preconditioned SGD, $w \hookleftarrow w - \eta P(X_m^T X_m w - X_m y)$.

This is cheap to compute; small overhead per iteration as P is low rank (+identity).

But we have to decrease η: eigenvectors of $X^T X$ and $X_m^T X_m$ do not align perfectly due to approximation. Choose $\eta = \sqrt{\frac{\lambda_{k_1}}{m}}$.

This really works. Experiment on Timit, Pegasos.

Conclusion: Deep is hard. Let's try to understand shallow. Computation is a limiting factor. Kernels can match NN at least for some problems. Build kernels for computational efficiency.

10 Greed is Good If We Add A Bit of Randomness (Amin Karbasi)

It is known that greedy methods perform well for maximizing monotone submodular functions. At the same time, such methods perform poorly in the face of non-monotonicity. In this talk, I show–arguably, surprisingly– that invoking the classical greedy algorithm $O(\sqrt{k})$-times leads to the (currently) fastest deterministic algorithm for maximizing a general submodular function subject to k-independent system constraints. This algorithm achieves $(1 + O(1/\sqrt{k}))k$ approximation using $O(nr\sqrt{k})$ function evaluations (here, n and r denote the size of the ground set and the maximum size of a feasible solution, respectively). I then show that by a careful random sampling procedure, we can run the greedy algorithm only once and obtain the (currently) fastest randomized algorithm for maximizing a submodular function subject to k-extendible system constraints (a subclass of k-independent system constrains). This randomized algorithm achieves $(k+3)$-approximation with only $O(nr/k)$ function evaluations. Finally, I derive an almost matching lower bound, and show that no polynomial time algorithm can have an approximation ratio smaller than $(k + 1/2 - \epsilon)$. No background in submodularity is needed. I will define all the important concepts during the talk. This is joint work with my student Chris Harshaw (Yale) and Moran Feldman (Open University of Israel).

Let $f : 2^N \to \mathbb{R}_+$. N is a ground set of size n. f is submodular if for all $X, Y \subseteq N$,

$$f(X) + f(Y) \geq f(X \cup Y) + f(X \cap Y).$$

Equivalently, $\forall X \subseteq Y \subseteq N$ and $e \notin Y$,

$$f(X \cup \{e\}) - f(X) \geq f(Y \cup \{e\}) - f(Y).$$

Definition has nothing to do with monotonicity.

Example of nonmonotone submodular function is cut graph, the number of outgoing edges. $f(\phi) = f(V) = 0$.

Ex. f measures representativeness. If you incorporate interaction between elements it's no longer monotone.

We want to maximize the submodular function subject to a constraint set I

$$\max_{X \subseteq I} f(X).$$

Say $(N; I)$ is an independent system if

$$B \in I, A \subseteq B \implies A \subseteq I.$$

Examples:

1. $\{X \subseteq N : |X| \leq k\}$ is an independent system.
2. A matroid is an independent system. (N, I) is a matroid if $A, B \in I$ and $|B| > |A|$, then there exists $e \in B$ such that $A \cup \{e\} \in I$.

   An example is sets of indices of linearly independent columns of a matrix.
3. Intersection of k matroids (not necessarily a matroid).
4. k-extendable system (N, I): for all $A \in I$ and extension $B \supseteq A$, if there exists e such that $A \cup \{e\} \in I$ then there exists Y of size at most k such that $(B \backslash Y) \cup \{e\} \in I$.

Inclusions:

$$\text{(cardinality)} \subseteq \text{(matroid)} \subseteq \text{(intersection of matroid)} \subseteq \text{(extendable)}$$

A matroid is a 1-extensible system. Can we solve the optimization problem when I is k-extendable?

Example of 2-extendable system that's not matroid: B-matching on graphs. Let G be graph, $b(v)$ be budget. Consider the set of matchings M such that any edge incident to any vertex v in the matching M is at most $b(v)$.

Results: An α-approximation is $f(S) \geq \frac{1}{\alpha} f\,(\text{opt})$. r is the max size of a feasible solution. (Runtime is number of query oracle calls made.)

Result	Approximation	Runtime
Gupte 10	$3k$	nrk
Last year	$2k$	nrk
2 weeks ago (determinstic) (repeated greedy)	$k + \Omega(\sqrt{k})$	$nr\sqrt{k}$
Randomized (sample greedy)	$k+3$	$\frac{nr}{k}$
Lower bound	$k + \frac{1}{2}$	

Lower bound: there is no poly algorithm with $F(S) \geq \frac{1}{k+\frac{1}{2}} F\,(\text{opt})$ unless $P = \mathsf{NP}$.

This basically closes the gap. The algorithms are extremely simple and fast.

For cardinality, non-monotone, $\max_{|X|\leq k} f(X)$,

Result	Approximation	
2 years ago	$\frac{1}{e}$	
1 year ago	$\frac{1}{e} + 0.04$	
Lower bound, 2011	0.38	
Lower bound, 2010	0.48	
Our result	$\left(\frac{1}{e}\right)^2$	linear (indep. of r)

Write $f(A|B) = f(A \cup B) - f(B)$.

Greedy algorithm for submodular D: Start with

$$N' \leftarrow N \tag{39}$$

$$S \leftarrow \phi. \tag{40}$$

While there exists $v \in N$ such that $D(v|S) > 0$, $S + \{v\} \in I$, let

$$U = \text{argmax}_v\, D(v|S) \tag{41}$$

$$S \leftarrow S \cup \{v\}. \tag{42}$$

This works very well when the function is monotone but otherwise it can get stuck.

Actually, this naive greedy algorithm is the one that achieves all these good running times *if* you run it correctly.

What do I mean?

Repeated greedy: for $i = 1$ to l,

- Let S_i be the solution of greedy where N_i is the ground set, I is independent set, and f is function.

- Let S_i' be the solution of unconstrained submodular maximization when S_i is the ground set,

  $$S_i' = \text{argmax}_{X \subseteq S_i} f(X)$$

  Note $f(S_i') \geq f(S_i)$.

- $$N_{i+1} := N \backslash S_i \tag{43}$$ $$T = \max\{S_1', \ldots\} \tag{44}$$

Theorem 10.1. *If we invoke greedy $\sqrt{k}$ times ($l = \sqrt{k}$), then greedy produces a solution T with approximation ratio*

$$k + (1 + \frac{\alpha}{2})\sqrt{k} + 3.$$

where α is the approximation guarantee of unconstrained problem.

We can get $\alpha = 3$.

The randomized algorithm is as follows. Let N' be a random set where each element of N is sampled indpendently with probability $\frac{1}{k+1}$. Run classical greedy once on N'. Then

$$\mathbb{E}[f(S)] \geq \frac{1}{k+3} f\left(\text{opt}\right).$$

All these algorithms also work in the monotone case and give the best guarantees there.

Proof sketch:

Lemma 10.2 (Gupta 10)**.** *Let S be the solution of classical greedy. Then $f(S) > \frac{1}{k+1} f(S \cup C)$ for any $C \in I$.*

Lemma 10.3. *For $1 \leq i \leq l$, $f(S_i) \geq \frac{1}{k+1} f(S_i \cup (OPT \cap N_i))$.*

Lemma 10.4.

$$f(S_i') \geq \frac{f(opt_i)}{\alpha} \geq \frac{f(S_i \cap opt)}{\alpha} \tag{45}$$

Lemma 10.5. *Let $g : e^N \to \mathbb{R}_+$ and $S \subseteq N$ be a random set such that each element appears with probability at most p (not necessarily independently). Then*

$$\mathbb{E}[f(S)] \geq (1 - p) f(\phi).$$

Let S to be randomly one of the S_i's.

Lemma 10.6.

$$\sum_{i=1}^{l} f(S_i \cup opt) \geq (l - 1) f\left(opt\right).$$

Let S be chosen one of $\{S_i\}_{i=1}^{d}$ randomly. Let $g(X) = f(X \cup \text{opt})$.

$$\frac{1}{e} \sum_{i=1}^{d} f(S_i \cup \text{opt}) = \mathbb{E}[f(S \cup \text{opt})] = \mathbb{E}[g(S)] = (1 - \frac{1}{d}) g(\phi) = (1 - \frac{1}{e}) f\left(\text{opt}\right)$$

Submodularity can also be defined with 3 sets: for all A, B, C,

$$f(A \cup (B \cap C)) + f(B \backslash C) \geq f(A \cup B).$$

(Proof: Consider $A \cap B \cap (C \cup C^c)$. Write f of this and use subadditivity.)

$$\begin{aligned}
OPT \backslash N_i &= N \cap OPT \cap N_i^c && (46)\\
&= OPT \cap (N \cap N_i^c) && (47)\\
&= OPT \cap (N \backslash N_i) && (48)\\
&= OPT \cap \left(\bigcup_{j=1}^{l-1} S_i \right) && (49)\\
&= \bigcup_{j=1}^{l-1} (OPT \cap S_j). && (50)
\end{aligned}$$

References

[AB15] Hassan Ashtiani and Shai Ben-David. “Representation learning for clustering: A statistical framework”. In: *arXiv preprint arXiv:1506.05900* (2015).

[AKB16] Hassan Ashtiani, Shrinu Kushagra, and Shai Ben-David. “Clustering with Same-Cluster Queries”. In: *Advances In Neural Information Processing Systems*. 2016, pp. 3216–3224.