Contents

Workshop website (with videos): `https://simons.berkeley.edu/workshops/schedule/3751`

1 Variational Inference: Foundations and Innovations (David Blei, Columbia University)

One of the core problems of modern statistics and machine learning is to approximate difficult-to-compute probability distributions. This problem is especially important in probabilistic modeling, which frames all inference about unknown quantities as a calculation about a conditional distribution. In this tutorial I review and discuss variational inference (VI), a method a that approximates probability distributions through optimization. VI has been used in myriad applications in machine learning and tends to be faster than more traditional methods, such as Markov chain Monte Carlo sampling.

This tutorial aims to provide both an introduction to VI, a modern view of the field, and an overview of the role that probabilistic inference plays in many of the central areas of machine learning. First, I will provide a review of variational inference. Second, I describe some of the pivotal tools for VI that have been developed in the last few years, tools like Monte Carlo gradient estimation, black box variational inference, stochastic variational inference, and variational autoencoders. Finally, I discuss some of the unsolved problems in VI and point to promising research directions.

Examples: overlapping community structure in US patents, topics in New York Times 1.8 million articles, population genetics, neuroscience analysis of 220 million fMRI measurements, 1.7 million taxi trajectories.

The probabilistic pipeline:

- Take knowledge and question; make assumption
- Take data and discover patterns
- Predict and explore

Notes:

- Customized data analysis is important to many fields.
- Pipeline separates assumptions, computation, application
- Eases collaborative solutions to statistics problems.

Probabilistic models:

- Probabilistic model is joint distribution $p(z, x)$ of hidden variables z and observed variables x.
- Inference about unknowns is through posterior, $p(z|x) = \frac{p(z,x)}{p(x)}$.
- $p(x)$ is often intractable to compute, so do approximate inference.

VI turns inference into optimization. Posit a variational family of distributions over latent variables $q(z; v)$. Fit variational parameters v to be close in KL.

Example: find means of mixture of gaussians. KL getting smaller is the same as evidence lower bound getting larger.

VI and stochastic optimization. Stochastic optimization makes VI better:

- stochastic VI scales to massive data.
- black box VI generalizes to wide class of models.
- reparametrization enables expressive families and amortized inference.

1.1 Mean-field VI

Mean-field inference casts Bayesian computation as optimization, SVI scales to massive data.

Motivation: topic models discover hidden thematic structure in large collection of documents. Each topic is distribution over words, each document is distribution over corpus-wide topics. Each word is drawn from one of those topics.

LDA as graphical model:

$$\alpha \to \theta_d \to z_{d,n} \to \underbrace{w_{d,n}}_{\text{observation}} \leftarrow \beta_k \leftarrow \eta$$

Plate N contains $z_{d,n}, w_{d,n}$; D contains θ_d and D, K contains β_k. Plate means repeated.

- α proportions parameter
- θ_d per-document proportions.
- $z_{d,n}$ per word topic assignment
- $w_{d,n}$ observed word
- β_k topics.

Posterior of latent variables is $p(\beta, \theta, z|w)$. We can't calculate the denominator, $\int_\beta \int_\theta \sum_z p(\beta, \theta, z, w)$.

Roadmap:

- Define generic class of conditionally conjugate models.
- Derive classical mean-field VI (from statistical physics).
- Derive stochastic VI, which scales to massive data.

Consider model

$$p(\beta, z, x) = p(\beta) \prod_{i=1}^{n} p(z_i, x_i|\beta).$$

Complete conditional is conditional of latent variable given observations and other latent variables. Assume each complete conditional is in the exponential family

$$p(z_i|\beta, x_i) = h(z_i) \exp(\eta_l(b, x_i)^T z_i - a(\eta_l(\beta, x_i))) \quad (1)$$
$$p(\beta|z, x) = h(\beta) \exp(\eta_g(z, x)^T \beta - a(\eta_g(z, x))) \quad (2)$$

Global parameter comes from conjugacy

$$\eta_g(z, x) = \alpha + \sum_{i=1}^{n} t(z_i, x_i).$$

Open: recognize all the bad properties KL divergence has, and finding alternative divergences or alternative optimization algorithms to alleviate those bad properties.

You will hardly see the KL divergence, you see the evidence lower bound (ELBO),

$$\mathcal{L}(v) = \mathbb{E}_q[\ln p(\beta, z, x)] - \mathbb{E}_q[\ln q(\beta, z; v)].$$

KL is intractable, VI optimizes the evidence lower bound instead.

- It is a lower bound on $\ln p(x)$.
- First term prefers q to place mass on MAP estimate.
- Second term encourages q to be diffuse.
- Caveat: ELBO is not convex.

We need to specify form of $q(\beta, z)$. The mean0field family is fully factorized,

$$q(\beta, z; \lambda, \phi) = q(\beta; \lambda) \prod_{i=1}^{n} q(z_i; \phi_i).$$

Tweak each marginal separately.

$$\mathcal{L}(\lambda, \phi) = \mathbb{E}_q[\ln p(\beta, z, x)] - \mathbb{E}_q[\ln q(\beta, z)].$$

Traditional VI uses coordinate ascent. Iteratively update each parameter, holding others fixed. Cf. Gibbs sampling.

Repeat: for each data point i,

$$\phi_i \leftarrow \mathbb{E}_\lambda[\eta_l(\beta, x_i)].$$

Set global parameter

$$\lambda \leftarrow \alpha + \sum_{i=1}^{n} \mathbb{E}_{\phi_i}[t(Z_i, x_i)].$$

Repeat until ELBO converged. Classical VI does some local computation for each data point.

1.2 Stochastic VI

Given massive dataset, subsample data, infer local structure, and update global structure. This gives us faster estimates.

Stochastic optimization: Replace gradient with noisy estimates.

$$v_{t+1} = v_t + \rho_t \widehat{\nabla}_v \mathcal{L}(v_t)$$

ρ_t follow Robbins-Munro condition.

The natural gradient of the ELBO is

$$\nabla_\lambda^{\text{nat}} \mathcal{L}(\lambda) = (\alpha + \sum_{i=1}^{n} \mathbb{E}_{\phi_i^*}[t(Z_i, x_i)]) - \lambda$$

Construct noisy natural gradient and use it. This is faster and better.

Time series model are not technically in this class.

1.3 Black box variational inference

Approximate inference can be difficult to derive, especially for models that are not conditionally conjugate, like discrete choice models, Bayesian generalized linear models...

Easily use VI with any model, no exponential family requirements. Cf. Metropolis Hastings (MCMC) of VI.

Sample from q or related distribution; form noisy gradients without model-specific comptuation.

If you dream of model on the beach without any constraints, you probably get a non-conjugate model. (Nonlinear time series, deep latent Gaussian, models with attention, deep exponential families...)

Simplest example is Bayesian logistic regression. Data are pairs (x_i, y_i), x_i is covariate, $y_i \in \{0, 1\}$ is binary label, z are regression coefficients. $p(y|x) = \frac{1}{1+e^{-xz}}$.

Goal: approximate posterior coefficient $p(z|x, y)$. Variational family $q(z; v)$ is normal, $v = (\mu, \sigma^2)$.

ELBO: we can't analytically take the expectation.

Options: derive model-specific bound, use other approximations that require model-specific analysis. Neither is good.

Key idea: write gradient as expectation, giving score gradient

$$\nabla_v \mathcal{L} = \mathop{\mathbb{E}}_{q(z;v)}[\nabla_v \ln q(z; v)(\ln p(x, z) - \ln q(z; v))].$$

a.k.a. likelihood ratio or REINFORCE gradient.

Construct noisy unbiased gradients with MC,

$$\widehat{\nabla}_v = \frac{1}{S} \sum_{s=1}^{S} \nabla_v \ln q(z_s; v)(\ln p(x, z_s) - \ln q(z_s; v)), \quad z_s \sim q(z; v)$$

This doesn't work out-of-the-box. We must control variance of gradient. Rao-Blackwellization, control variates, importance sampling... Use adaptive learning rates (ex. RMSProp). We can subsample data, do SVI.

We used this to fit deep exponential families. Adaptation of deep representation to probabilistic models. Before, have to use sigmoid belief networks. Now we can do this more broadly.

We used this for topographic factor analysis with fMRI.

We can take VI and use it in probabilistic programming.

Write model down as program that generates data. Instead of compiling, compile as something that takes observations and generates inference. It requires methods of deriving inference algorithms. Prob programming compilers. Black-box methods enable this!

There are 2 ideas left on the table:

- reparametrization gradient: write latent variable as transformed variable. Express gradient of ELBO as expectation

$$\nabla \mathcal{L} = \mathop{\mathbb{E}}_{s(\varepsilon)}[\nabla_z[\ln p(x, z) - \ln q(z; v)]\nabla_v t(\varepsilon, v)]$$

- Amortization: Variational parameters are a function of input $v(x)$.

Ex. variational autoencoder. Model: neural network plus Gaussian noise, $x = f(z) + \varepsilon$. Variational parameters are deep neural net.

See Edward, probabilistic programming library.

Open problems:

- Theory: MCMC widely analyzed, VI less explored.
- Optimization
- Alternative divergences
- Better approximations

2 Representational and Optimization Properties of Deep Residual Networks (Peter Bartlett, UC Berkeley)

Deep residual networks have been widely adopted for computer vision applications because they exhibit fast training, even for very deep networks. The key difference between these networks and other feedforward networks is that a residual network with near-zero parameters computes a near-identity function, rather than a near-zero function. We consider mappings of this kind: compositions of near-identity functions. Rather than fixing a finite parameterization, such as fixed-size ReLU or sigmoid layers, we allow arbitrary Lipschitz deviations from the identity map; these might be Lipschitz-constrained sigmoid layers of unbounded width, for instance. We investigate representational and optimization properties of these near-identity compositions. In particular, we show that a smooth bi-Lipschitz function can be represented exactly as a composition of functions that are Lipschitz-close to the identity, and greater depth allows a smaller Lipschitz constant. We also consider the optimization problem that arises from regression with a composition of near-identity nonlinear maps. For functional gradients with respect to these nonlinear maps, we show that any critical point of a convex objective in the near-identity region must be a global minimizer.

Joint work with Steve Evans and Phil Long.

View deep nets as deep compositions of nonlinear functions

$$h = h_m \circ \cdots \circ h_1,$$

where m is large. For example, $h_i(x) = \sigma(W_i x)$, where $\sigma(v)_i = \frac{1}{1+\exp(-v_i)}$, or $h_i(x) = r(W_i x)$, $r(v)_i = \max\{0, v_i\}$.

Why deep?

- Representation learning: Depth provides an effective way of representating useful features.

- Rich non-parametric family: Depth provides parsimonious representations. Nonlinear parametrizations provide better rates of approximation. Some functions require much more complexity for a shallow representation.

But optimization is worse as the depth increases.
I'll talk about deep residual networks.
ImageNet progress: There were 2 big jumps from

1. from shallow to deep (8-layer) nets (AlexNet)
2. and to ResNets (152 layers).

A residual network has a shortcut connection,

$$H(x) = F(x) + x.$$

Advantages:

- with zero weights, the network computes the identity.
- identity connections provide useful feedback throughout the network.

What's behind the success of residual networks?
Intuition: linear functions.

1. Every invertible A with $\det(A) > 0$ can be written as
$$A = (I + A_m) \cdots (I + A_1)$$
where $\|A_i\| = O\left(\frac{1}{m}\right)$. (Hardt, Ma, 2016.)
2. For a linear Gaussian model, $y = Ax + \varepsilon$, $\varepsilon \sim N(0, \sigma^2 I)$, consider choosing $A_1, \ldots, A_m$ to minimize quadratic loss
$$\mathbb{E} \left\| (I + A_m) \cdots (I + A_1)x - y \right\|^2.$$
If $\|A_i\| < 1$, every stationary point of the quadratic loss is a global optimum.

We show these statements are true in the nonlinear case.

Theorem. *Computation of smooth invertible map h can be spread throughout a deep network $h_m \circ \cdots \circ h_1 = h$ so that all layers compute near-identity functions*

$$\left\| h_i - \mathrm{Id} \right\|_L = O\left(\frac{\ln m}{m}\right).$$

The Lipschitz seminorm:

$$\|f(x) - f(y)\| \leq \|f\|_L \|x - y\|.$$

Think of h_i as near-identity maps, ex.

$$h_i(x) = x + A\sigma(Bx).$$

As the network gets deeper, the functions $x \mapsto A\sigma(Bx)$ can get flatter.

Theorem 2.1. *Consider $h : \mathbb{R}^d \to \mathbb{R}^d$ on bounded $X \subset \mathbb{R}^d$. Suppose it is*

1. *differentiable*
2. *invertible*
3. *smooth: for some $\alpha > 0$, $\forall x, y, u$, $\|Dh(y) - Dh(x)\| \le \alpha \|y - x\|$. (Here $\|Dh(y)\|$ is the induced norm $\|f\| := \sup\left\{\frac{\|f(x)\|}{\|x\|} : \|x\| > 0\right\}$.*
4. *Lipschitz inverse: for some $M > 0$, $\|h^{-1}\|_L \le M$.*
5. *Positive orientation: for some x_0, $\det(Dh(x_0)) > 0$.*

Then there are m functions $\mathbb{R}^d \to \mathbb{R}^d$,

$$\|h_i - \mathrm{Id}\|_L = O\left(\frac{\ln m}{m}\right)$$

and $h_m \circ \cdots h_1 = h$.

When you're close to the identity, you have an analogous behavior of the error surface that all local optimum are global.

Key ideas:

1. Assume $h(0) = 0$, $Dh(0) = \mathrm{Id}$ (else shift and linearly transform).
2. Construct h_i so that

$$h_1(x) = \frac{h(a_1 x)}{a_1} \tag{3}$$

$$h_2(h_1(x)) = \frac{h(a_2 x)}{a_n}. \tag{4}$$

3. Pick $a_m = 1$ so that $h_m \circ \cdots \circ h_1 = h$.
4. Ensure A_1 is small so $h_1 \approx Dh(0) = \mathrm{Id}$.
5. Ensure a_i, a_{i+1} are close so that $h_i \approx \mathrm{Id}$.
6. Show $\|h_i - \mathrm{Id}\|_L$ are small on small and large scales.

Error landscape:

Theorem 2.2. *For (X, Y) with arbitrary joint distribution, let*

$$Q(h) = \frac{1}{2}\mathbb{E}\,\|h(X) - Y\|_2^2$$

and define the minimizer $h^*(x) = \mathbb{E}[Y|X = x]$. (Ex. if (X, Y) is uniform, Q is empirical risk, h^* is ERM.)

Consider $h = h_m \circ \cdots \circ h_1$, $\|h_i - \mathrm{Id}\| \le \varepsilon < 1$.

Then for all i,

$$\|D_{h_i} Q(h)\| \ge \frac{(1-\varepsilon)^{m-1}}{\|h - h^*\|}(Q(h) - Q(h^*)).$$

($D_{h_i} Q$ is Fréchet derivative, $\|h\|$ is induced norm.)

If composition h is sub-optimal and each h_i is near identity, theorem says there is a downhill direction in function space.

The theorem does not say there are no local minima of deep residual network of ReLUs or sigmoids with a fixed architecture. The direction in function space can be orthogonal to functions that can be computed with the fixed architecture.

We should expect suboptimal stationary points in the ReLU or sigmoid parameter space, but they arise solely within a layer.

Proof idea: If $\|f - \mathrm{Id}\|_L \leq \alpha < 1$, then

1. f is invertible
2. $\|f\|_L \leq 1 + \alpha$ and $\|f^{-1}\|_L \leq \frac{1}{1-\alpha}$.
3. For $F(g) = f \circ g$, $\|h\| = \|h\|_L$.

1. Projection theorem implies $Q(h) = \frac{1}{2}\mathbb{E}\,\|h(X) - h^*(X)\|_2^2 + \text{(constant)}$.
2. Then
$$D_{h_i}Q(h) = \mathbb{E}[(h(X) - h^*(X)) \cdot ev_X \circ D_{h_i}h].$$
3. It is possible to choose a direction Δ such that $\|\Delta\| = 1$ and $D_{h_i}Q(h)(\Delta) = c\mathbb{E}\,\|h(X) - h^*(X)\|_2^2$.
4. Because h_js are near identities, get the inequality.

Questions:

- What if the mapping is not invertible?

  If h can be extended to bi-Lipschitz mapping to $\mathbb{R}^d$, it can be represented with flat functions at each layer. (Project to 1-D at the end.)

  What if it cannot be extended?
- Implications for optimization? Related to Polyak-Lojasiewicz function classes. Proximal algorithms for these classes converge quickly to stationary points.
- Do stochastic gradient methods produce near-identities?

3 Composing Graphical Models with Neural Networks for Structured Representations and Fast Inference (David Duvenaud, University of Toronto)

How to combine the complementary strengths of probabilistic graphical models and neural networks? We compose latent graphical models with neural network observation likelihoods. For inference, we use recognition networks to produce local evidence potentials, then combine them using efficient message-passing algorithms. All components are trained simultaneously with a single stochastic variational inference objective. We use this framework to automatically segment and categorize mouse behavior from raw depth video.

Practical problem: coding mouse behavior.The inner loop of this experience means that a grad student needs to watch the mouse for hours. There huge variability. Lots of scientific pipelines involve these tasks. There are lots of "behavior phonemes", which could be on very short scales. An initial model is to have clustering.

Simplest thing: run mixture Gaussians on raw video data.

Read data doesn't form nice Gaussian clusters. Real data lies on low-dimensional manifolds in high-dimensional spaces. Fitting a mixture of Gaussians will tile the space with lots of Gaussians. It has good properties as a density estimators, but we're worried about inductive bias and have lost the interpretability of a mixture of Gaussians.

Imagine putting a NN in there: warp the data by a NN. Get the best of both worlds: flexible density model and interpretable high-level structure.

20-year explosion at NIPS: here's a interesting model, work out inference for it.

Pros of probabilistic graphical models:

- structured representations
- priors and uncertainty
- data and computational efficiency

Cons:

- Rigid assumptions may not fit
- Feature engineering
- Top-down inference. (MCMC: top-down explanation, move in latent space of explanations.)

Deep learning cons:

- Once you fit a model you get uninterpretable goo.
- It's difficult to parameterize.
- Can require lots of data.

Pros:

- flexible
- feature learning
- recognition networks (learn to do inference, take burden off inference schemes). This is like bottom-up inference

Neural nets and graphical models have complementary benefits.

Make PGM's great again!

Modeling idea: graphical models for latent variables, neural nets for observations.

This is already what we do in supervised learning: Layers of NN up to last layer is transformation into space where logistic regression can handle the data. NN is a bridge from non-interpretable version of data, to space where structured assumptions do hold.

How to build a model to model mouse behavior?

There is a discrete latent variable that encodes the behavior. Add continuous variable representing physical state in low-dimensional state. The behavioral state determines the dynamics of this.

We don't need supervised learning! By jointly learning, we can fit the data and find an explanation.

Connect to frames of video by neural net.

$$y_t|x_t, \gamma \sim N(\mu(x_t;\gamma), \Sigma(x_t;\gamma)).$$

Anyone can write down a fancy model; what's cool is doing inference.

How would we do inference if we had nice conjugate observations, ex. Kalman filter. Cast in SVI framework,

$$q(\theta)q(x) \approx p(\theta, x|y) \tag{5}$$
$$\mathcal{L}(\eta_\theta, \eta_x) = \mathbb{E}_{q(\theta)q(x)}\left[\ln \frac{p(\theta, x, y)}{q(\theta)q(x)}\right]. \tag{6}$$

We can get an unbiased estimate of the gradient with a single data point.

Steps:

1. Compute evidence potentials
2. Run fast message passing
3. Compute natural gradient

$p(x|\theta)$ linear dynamical system, $p(y|x,\gamma)$ NN decoder, $p(\theta)$ conjugate prior, $p(\gamma)$ generic.

The ELBO has $\ln \frac{p(\theta,\gamma,x)p(y|x,\gamma)}{q(\theta)q(\gamma)q(x)}$. The term $p(y|x,\gamma)$ is a problem. Construct a conjugate variant of it

$$\hat{L} = \mathbb{E}\left[\ln \frac{p(\theta,\gamma,x)\exp(\psi(x;y,\phi))}{q(\theta)q(\gamma)q(x)}\right]$$

If observations were conjugate this would be the right ELBO.

1. Apply recognition network.
2. Run fast PGM algorithms. Combine local evidence from every frame to coherent story.
3. Sample, compute flat gradients.
4. Compute natural gradient.

Natural gradient SVI: expensive for general observation. Pros: optimal local factor, exploits conjugate graph structure, arbitrary inference queries, natural gradients.

Variational autoencoders (brute force): Fast for general observations. Cons: suboptimal local factor, ϕ does all local inference (neural network has to learn Bayes's rule), can only do limited inference queries (those we've trained it to answer), no natural gradients

Structured VAE's have all benefits: fast for general observations, optimal given conjugate evidence, exploits conjugate graph structure, arbitrary inference queries, natural gradients on η_θ.

This recipe is general, it plugs into many different scenarios. We have deep versions of everything probabilistic done in NIPS, in principle.

Jointly learn gaussian mixture in latent space, and mapping to latent space.

Ex. Learn linear latent variable dynamical system, bouncing ball. Give 4-D latent space. Arrange data so that it can be modeled by dynamics!

How to set number of clusters? Marginal likelihood turns off clusters you don't need. Make a huge model; it gets rid of what it doesn't need.

For a NN, if we change what we condition on we would have to retrain the NN. A RNN could address some of the problems, but probably wouldn't train as fast.

Transform to egocentric view of mouse. Model has discrete and continuous part, and generative model.

We can generate and see if it looks like real data. Take a 2-D slice: scrunched up vs. long, standing up vs. laying down.

For the interpretable structure part: look at real data that most actives one of the states. One state is "rearing up". This is a sensible behavioral phoneme. Another is "fall from rear", "grooming".

Limitations and future work:

1. Capacity:
    - How expressive is latent linear structure? (word embeddings, analogical reasoning in image models)
    - SVAE can use nonlinear latent structure.
2. Complexity:
    - PGMs get complicated.
    - SVAE keeps complexity modular.
3. Future work
    - model-bsaed RL
    - automatic structured search
    - semi-supervised applications. Ex. Group by having human label 1 hour of video.

4 Scaling Up Bayesian Inference for Big and Complex Data (David Dunson, Duke University)

For scientific applications involving large and complex data, it is useful to develop probability models and inference methods but issues arise in terms of computing speed and stability. In such settings accurate uncertainty quantification is critical and common fast approximations often do a poor job of UQ. In this talk, I discuss recent developments in scaling up sampling algorithms to big problems - considering broad classes of approximate and parallelizable MCMC algorithms with corresponding theory guarantees. A particular focus will be on data augmentation approaches. A variety of applications will be provided as motivation.

Usually we focus on optimization, but we care about uncertainty quantification because we're doing scientific inference. We can't do deep learning and point optimization because uncertainty is everything. A black box for pattern classification, prediction is not enough. We want to learn the structure. For example, data is brain structure in humans and we want to relate it to autism, genes and relation to disease, etc.

Machine has been taken over by deep learning because everyone is focused on industry-type problem where you have huge sample sizes. Deep learning methods are not designed for the setting where the data is small and the dimensionality is large.

Let the likelihood be $L(y|\theta)$ where y is data and θ is parameter. The posterior is

$$\pi(\theta|y) = \frac{\pi(\theta)L(y|\theta)}{\int \pi(\theta)L(y|\theta)\, d\theta}$$

This is hard to approximate, usually doesn't have closed form. Often we don't know how well they do. They can do well in industry applications, not necessarily in scientific applications. There's MCMC. We can bypass calculating the integral, define some sampling that converges to the right distribution, do inferences based on samples. Some problems arise in big and high-dimensional data,

- computational bottleneck. (Calculate matrices, invert matrices, etc.) Get per-iteration slowdowns.
- Mixing rate problems: In an ideal world we generate independent samples, but in reality they are autocorrelated. You have to collect a lot of samples to get the same MC error.

Study properties of MCMC algorithms and try to address these problems to make it competitive with variational inference. Three directions:

1. Embarrassingly parallel MCMC algorithms. (EPMCMC)
2. Approximate MCMC. Until recently people have been religiously defining a Markov chain with kernel converging to exactly the desired distribution. You can get orders of magnitudes speedup if you relax this. (Annals of statistics paper.) Comp-minimax: broad class of MCMC algorithms, ε error probability in kernel, as ε larger, can evaluate faster. If you have a certain computational budget, I can use this.

3. Data augmentation (for discrete and unbalanced data). It might be hard to define a MC for the posterior, so introduce latent variable z, $\pi(\theta, z|y)$. This looks more complicated but maybe you made all the operations easier. This is like EM for sampling.

   We have strong results on this.

4.1 EP MCMC

$$L(y|\theta) = \prod_{i=1}^{n} L(y_i|\theta).$$

As the sample size increases, it becomes harder to do MCMC efficiently. What can we do? We would like to put data on different machines. Let K is the number of machines/processors, and $y[j]$ is a shard/chunk of data on machine j. MCMC hasn't been so good on GPU because of communication costs.

The jth subset posterior distribution is

$$\pi(\theta|y[j]) = \frac{\pi(\theta)L(y[j]|\theta)^k}{d}$$

We have a bunch of noisy posterior distributions. Maybe we can take some appropriate notion of mean or median to give an estimate of the true posterior distribution. Think of a Banach space where each point is a probability measure (posterior), and we get close to the true distribution by taking some notion of mean.

Use the (WASP) Wasserstein barycenter posterior. Minimize sum of squares of Wasserstein-2 distance to point. If each was atomic, get $\frac{1}{T}\sum_{t=1}^{T} \delta_{\theta[j]}(t)$.

In general, solve using a sparse linear program; this is fast. Throw data on different machines. Run with likelihood raised to power. We get

$$\widehat{\pi}(\theta|y) = \frac{1}{T}\sum_{t=1}^{T} \pi_t \delta_{\theta_t}.$$

(From a frequentist perspective we can study the properties of this. True posterior generates around that point. Get nice bounds between the estimate and the exact posterior depending on k, m.)

We're really interested in functionals of the parameters $f_j(\theta)$. (Ex. probability of a disease.) Almost always you're interested in 1-dimensional functionals. The Wasserstein barycenter in the 1-dimensional case can be calculated analytically. Our algorithm only has to do the following: run MCMC in parallel, calculate percentile of interest. Feed back to central machine and average. Build theoretically strong approximation to true posterior. On each machine run your chunk of data.

4.2 aMCMC

We want to sample from $\pi(\theta|y)$.

We have a transition kernel K, obey MCMC rules so it converges the right thing. Design K's that are good, solve general problems, etc.

K_t could be faster to approximate. Use stochastic approximation.

Assumptions

1. exact MCMC mixes well.
2. K_θ is close to K.

This converges to $\pi_t(\theta|y)$ which is close to $\pi(\theta|y)$. But what are we actually calculating? $f_j(\theta)$. We want the Monte Carlo error to be small, decreasing in clock time as fast as possible. There are 2 parts to the error.

1. asymptotic bias
2. pure MC error

For a given computational budget and loss function there is an optimal choice of ε. Study several classes of algorithms people use and get practical insights into use.

We looked at...

- logistic regression with stochastic approximation. This doesn't work very well because the ε is big unless the subset sample size is huge, when we don't get much speedup. This was a surprising results. In practice, this minibatch things don't work very well.
- adaptive Gaussian approximation
- Gaussian process models: for small ε you can get huge computational savings

4.3 Data augmentation MCMC

Existing algorithms: Albert-Cho, Polya-Gamma.

But if n is huge,...

The mixing rate for these algorithms as n gets large for unbalanced data goes to 0, $\sqrt{(\text{sample size})}$ or faster.

We were modified by computational advertising: n is enormous. Data is very imbalanced. Most of the data is 0.

Even in a simple problem, samples are very autocorrelated.

Deep thing going on theoretically: The reason these algorithms fail is there's a mismatch. What if the rate of concentration of $\pi(\theta|y)$ is not the same as $\pi(\theta|z,y)$? If there's a big mismatch, mixing becomes slow. We calibrate the data augmentation algorithm to adjust for the mismatch.

5 Unbiased Estimation of the Spectral Properties of Large Implicit Matrices (Ryan Adams, Harvard University)

Many empirical questions about machine learning systems are best answered through the eigenvalues of associated matrices. For example, the Hessian of the loss function for a large deep network gives us insight into the difficulty of optimization and sensitivity to parameters. The spectra of adjacency and Laplacian matrices of large graphs helps us understand the global structure between vertices. Unfortunately, in the most interesting situations, these matrices are too large to be explicitly instantiated, to say nothing of diagonalizingthem directly; rather they are implicit matrices, which can only be interrogatedvia matrix-vector products. In this work I will discuss how several differentrandomized estimation tricks can be assembled to construct unbiased estimatorsof the spectra of large implicit matrices via generalized traces.

It's important to think of ourselves as doing science. How do we do better empirical science in deep learning? One perspective I got from David is the idea of building microscopes. In fields such as biology, you are motivated to build a system to analyze the system better. It's helpful to frame the problems we ask in the terms of building microscopes.

We care about spectra of large matrices (ex. Hessians) for a variety of purposes. For neural nets, we want to reason about how hard optimization problem is. We can study things small scales. How do we examine large systems we see in practice? This is harder, we can't just diagonalize a 10 billion by 10 billion matrix. We want to tackle these problems: sensitivity to parameter, whether we are at local minima, etc. We care about spectra of graphs, conductance, number of triangle,s communities.

This arises too in many-body systems. Computational chemistry: density functionals, involve reasoning about large systems where eigenvalues have physical interpretations

I'll talk about two tricks from the 1980s. Get unbiased estimates for things you want to know about big matrices, where you have implicit access to the matrix. In deep learning you can't even get a matrix-vector product, you get an estimate of it (minibatch estimate).

We have a vector A, all we can do is pick x and get an unbiased estimate of Ax.

Trace estimate: if we can generalize traces, we can do a lot of estimates. We look at

$$\ln f(A),$$

for example, log determinant. We think our functions as functions of the eigenvalues. Apply function to eigenvalues and sum them.

5.1 Trick 1

For $x \sim N(0, I)$,

$$\mathbb{E}_x[x^T Ax] = \mathrm{Tr}(A).$$

Proof:

$$\begin{aligned}
\mathbb{E}[(x^T Ax)] &= \mathbb{E}[\mathrm{Tr}(x^T Ax)] && (7)\\
&= \mathbb{E}[\mathrm{Tr}(xx^T A)] && (8)\\
&= \mathrm{Tr}(\mathbb{E}[xx^T A]) && (9)\\
&= \mathrm{Tr}(\mathbb{E}[xx^T]A) && (10)\\
&= \mathrm{Tr}(A). && (11)
\end{aligned}$$

This works for any random variable such that $\mathbb{E}[xx^T] = I$. This is generally atrributed to Hutchinson in 1989. It has a blessing of dimensionality. The bigger the A, the faster it converges due to Central Limit Theorem.

For Gaussians, you can also prove this by diagonalizing with an orthogonal matrix; the rotation of a Gaussian is Gaussian.

An easy example is to estimate the logdet.

5.2 Trick 2: Poisson estimator

It solves a problem that when you learn undergrad statistics, seems impossible.

We have a quantity y and a random estimator $\widehat{y}$ that is unbiased,

$$\mathbb{E}[\widehat{Y}] = Y.$$

But we're interested not in Y but $Z = \exp(Y)$. In general, $\exp(\widehat{Y})$ is not the right estimator for Z. For analytic functions you can fix it up and make it an unbiased estimator.

Draw an integer $K \sim \text{Poisson}(\lambda)$. Make a random product of a bunch of Y's, and let

$$\widehat{Z} = \exp(\lambda) \prod_{k=1}^{K} \widehat{Y}_k.$$

This is unbiased because

$$\widehat{E}\widehat{Z} = \sum_{j=0}^{\infty} \frac{\lambda^j}{j!} e^{-\lambda} \prod_{k=1}^{j} \frac{\widehat{Y}_k}{\lambda} \tag{12}$$

$$= \sum_j \frac{\lambda^j}{j!} \frac{Y^j}{\lambda^j} = \sum_{j=0}^{\infty} \frac{Y^j}{j!} \tag{13}$$

An expectation of a product of independent random variables is the product of expectations. Ex. we get easy estimates of log-likelihoods; use this to get a Markov chain.

This is just randomized function approximation by importance sampling. Let's generalize. It can be obnoxious for the reasom importance sampling is obnoxious.

Suppose we have $f(x) = \sum_{j=0}^{\infty} a_j x^j$. Put our Monte Carlo had on and importance sample this. Form

$$\widehat{f}(x) = \frac{a_j x^j}{\pi_j}.$$

Variance can be a disaster if we choose π_j poorly with respect to a_j. Log is obnoxious because of slow decay, we can choose zeta distribution with comparable tails.

When I first saw this, this seemed like magic.

(Is there a simple condition? We want the ratio to have an upper bound. If you have lighter tails than target, could get giant weights and giant varianaces.)

Holomorphic operator calculus: do this except for matrices. Generalize analytic function to matrices. Write out power matrices in the same way, do matrix powers.

Diagonalizing $A = U\Lambda U^T$, $A^k = U\Lambda^k U^T$. Write $f(X) = \sum_{j=0}^{\infty} a_j X^j$.

Combine the two tricks. Get estimates of Y's from minibatches, we can still push this through.

Gluing and generalizing these tricks, we can estimate in embarrassingly parallel way computing things in giant neural networks.

Actual algorithm: given f and power series (note $f(X)$ only converges if operator norm $\|X\|_2$ is within radius of convergence), draw x from proposal distribution, hit it j times, compute inner product with original x, weight with a_j and weight with proposal probability.

Write down a Chebyshev approximation for indicator function, then we can count the number of eigenvalues in an interval.

We need to hit random x with matrix a bunch of times. Actually we're doing all the work of computing intermediate terms up to some point. We can do something better. We're doing a random truncation, correct with the probability we would choose that truncation. This is familiar to the numerics community.

$\widehat{f}(x) = \sum_{j=0}^{k} \frac{a_j}{1-\sum_{l=0}^{j} \pi_l}$. (Count all the number of times you hit j on the way to the truncation point.) This is not always better. I don't know when exactly it is better.

I hope this will help us to the science of asking questions about 10 billion parameter models. Graphs, big quantum systems.

We can compute the trace by 10 billion orthogonal vectors. The hope is this gets smaller error faster. Ex. there's a spike what we won't see.

Ex. how bad is your saddle point. There's a lot of folk wisdom about local minima, saddle points, etc. We want to check this. Ex. number of negative eigenvalues. Approximate indicator function. Penalty is Gibbs phenomenon, wiggles at the edges.

6 Stochastic Gradient MCMC for Independent and Dependent Data Sources (Emily Fox, University of Washington)

Many recent Markov chain Monte Carlo (MCMC) samplers leverage continuous dynamics to define a transition kernel that rapidly explores a target distribution. In tandem, a focus has been on devising scalable variants that use stochastic gradients in the dynamic simulations. However, such stochastic gradient MCMC (SG-MCMC) samplers have lagged behind their full-data counterparts in terms of the complexity of dynamics considered since proving convergence in the presence of the stochastic gradient noise is nontrivial.

In this talk, we first present a general recipe for constructing SG-MCMC samplers that translates the task of finding a valid sampler into one of choosing two matrices. Importantly, any continuous Markov process sampling from the target distribution can be written in our framework. We then turn our attention to MCMC techniques based on jump processes, and propose an algorithm akin to Metropolis Hastings—with the same ease of implementation—but allowing for irreversible dynamics. Finally, we describe how SG-MCMC algorithms can be applied to applications involving dependent data, where the challenge arises from the need to break the dependencies when considering minibatches of observations. We propose an algorithm that harnesses the inherent memory decay of the process and provably leads to the correct stationary distribution.

We demonstrate our methods on a number of large-scale applications, including a streaming Wikipedia LDA analysis and segmentation of a lengthy genome sequence and ion channel recording.

I look at time series applications.

- An example (with Zillow) is model a local housing index: look at value in a local region and see how it changes over time. We want to share information between locations.
- Neuroimaging datasets. Discover functional connectivity. Do automatic parsing of recording. Recordings are long, up to 2 weeks long.

We want to handle complex posteriors in large-data settings. Develop scalable Bayesian approaches.

We focus specifically on MCMC.

Posteriors can be

- Complex: There may be multiple modes or strong correlation across dimensions.
- Large-data: Issue is scalability.

Classical approach is MCMC via jump processes. Standard method is Metropolis Hastings. Propose θ' from kernel depending on past value θ. Accept or rejct. Often this inefficiently explores posterior.

There's been interest in continuous dynamics based samplers, a.k.a, gradient MCMC.

One popular example is Hamiltonian Monte Carlo (HMC). The target posterior of θ is

$$\pi(\theta) := p(\theta|D) \propto \exp(-U(\theta)).$$

Add auxiliary momentum variable r. These 2 terms define an energy landscape that we explore.

$$\pi(\theta, r) \propto \exp\left(-U(\theta) - \frac{1}{2}r^T M^{-1} r\right).$$

Use Hamiltonian dynamics to collect samples on fixed continuous-time interval. Walk around a given level set.

Write down Hamiltonian dynamics.

$$d\theta = M^{-1} r\, dt \tag{14}$$
$$dr = -\nabla U(\theta)\, dt. \tag{15}$$

Analogy: hockey puck on frictionless ice surface. The momentum carries over hill. They leave target distribution invariant.

We'll simulate jointly θ and r. To get ergodicity, HMC resamples momentum. Jump between level sets and walk around. If you just want samples of θ, just discard samples of r.

HMC is one set of dynamics. We want $\pi(\theta)$ invariant under these dynamics. Simulated samples are desired posterior samples. Which have property of correct stationary distribution? HMC, Langevin dynamics, Riemann manifold LD/HMC. A natural question is: is there a general recipe for constructing a dynamic that lives in this space?

Yi Ma: it is possible.

$$\pi(z) := p(z|D) \propto \exp(-H(z)), \quad z = \{\theta, r\}.$$

Define a SDE, with PSD S, skew-symmetric Q, total energy H, such that $\pi(z)$ is invariant.

$$dz = -[D + Q]\nabla H \, dt + \Gamma(z) \, dt + \sqrt{2D} dW(t).$$

If you use this SDE, for any D, Q, you get a good sampler. HMC, LD lie in this space. Any valid sampler has a D and Q in our framework. You capture the space of all valid samplers.

For a practical algorithm, consider ε-discretization. We can correct bias.

There is a gradient computation. For each each step you have to make a pass through the entire dataset.

To scale grad-MCMC, replace true gradient with stochastic gradient.

$$\nabla \widetilde{H}(z) = -\frac{N}{n} \sum_{j \in S} \nabla \ln p(x_j|z) - \nabla \ln p(z).$$

For minibatches sampled uniformly at random from data, $\mathbb{E}[\nabla \widetilde{H}(z)] = \nabla H(z)$. We assume $\nabla \widetilde{H}(z) = \nabla H(z) + [N(0, V(\theta)), 0]^T$.

Does this work? There's hope that it would.

Welling and Teh proposed SGLD. SGD with momentum gives significantly faster convergence. What happens to momentum-based SGLD?

Random wind blows the puck. Flighty puck: this changes the distribution. Play on gravel street: add friction.

$$d\theta = M^{-1} r \, dt \tag{16}$$

$$dr = \nabla U(\theta) \, dt - \frac{1}{2}\varepsilon V(\theta) M^{-1} r \, dt + N(0, \varepsilon V(\theta) dt). \tag{17}$$

This is SGHMC which has SGLD, SGD as special cases.

Need step size going to 0 to get bias to 0. We allow for small bias and use finite ε.

But for more complicated dynamics what's the right correction? Go back to general recipe

$$z_{t+1} = z_t - \varepsilon_t[(D + Q)\nabla H + \Gamma] + N(0, 2\varepsilon_t D(z_t)).$$

Modified: replace ∇H by estimate and $2\varepsilon_t D(z_t)$ by $2D(z_t - \varepsilon \widehat{B}_t)$.

SGRiemannHMC. Naive approach has wrong stationary distribution. Take D, Q of SGHMC and make state dependent. Apply to online LDA. There's improved performance on the the Riemann version (it helps in exploring). Top performer is SGRHMC?

Divide and conquer parallel MCMC approaches. Divide data on different machines, run MCMC, form subposteriors and merge. Examples:

- consensus Monte Carlo
- Kernel density estimator per subposterior.

One target distribution is multivariate t. Momentum helps explore tails.

KDE scales poorly with dimensionality; other methods are robust to dimensionality.

Minibatch methods meet dependent data: In time series, we have dependent data. All stochastic gradient methods assume you have independent observations to form minibatches. Consider a HMM with global parameter θ (transition probabilities, observation parameters). Batch learning for HMMs: use current θ to form local state beliefs. Propagate information forwards and them backgrounds. Combine to form smooth local state beliefs. Use to update belief about global parameters.

Cost is $O(K^2T)$ per global update. This is inefficient for long time series.

One challenge is sequencing human chromatin segmentation. Here $T = 250$ million.

Can we use minibatch learning for HMMs?

Look at subsequence, do local forward and backward pass. Use local state beliefs to update global parameters. This issue is that information is getting blocked: we're not properly propagating information.

Do we expect x_t to influence $x_{t+1000000}$? There's memory decay in this process. In most applications, probably not.

Buffering: do forward and backward on a larger subchain. In practice, you only need limited buffer. Complexity is $O(K^2L_{buffer})$ per iterations. Similar idea as splash BP (parallelizing BP). Buffer size depends on θ.

Minibatch learning for HMMs via stochastic variational inference (SVI-HMM).

Previous analysis used dynamic Bayesian network. Because of complexity of inference, ad to break into chunks. We used a vanilla HMM with Gaussian emissions and apply SVI procedure. We finish in < 1 hour compared to days.

SG-MCMC is not so straightforward:

- assumes continuous parameter space. Typical HMM MCMC algorithms iterate on sampling latent discrete-valued state sequence.
- Minibatch examples correlated.

SVI-HMM learns approximation to posterior, assumes conjugate priors, uses heuristic buffer length estimation. We want to avoid this.

Marginalize latent state sequence. Everything gets connected when you do that, but observations are more correlated locally.

$$p(y|\theta) = 1^T P(y_T)A \cdots P(y_1)A\pi_0.$$

Rewrite in terms of specific subsequence. Group together terms corresponding to $y_{\tau-L:\tau+L}$ as $P(y_{\tau,L})$,

$$q^T_{\tau+L+1}P(y_{\tau,L})\pi_{\tau-L-1}$$

$U_{\theta_i} = -(\ln p(y|\theta))_{\theta_i} - (\ln p(\theta))_{\theta_i}$. Rewrite $(\ln p(y|\theta))_{\theta_i}$ as summing over disjoint subsequences. q, π calculations involving touching nearly all T observations. Instead take stochastic gradient approach.

$$-\frac{T}{L}\frac{q^T_{\tau+L+1}P_{\theta_i}(y_{\tau,L})\pi_{\tau-L-1}}{q^T_{\tau+L+1}P(y_{\tau,L})\pi_{\tau-L-1}}$$

How much buffering needed? Estimate Lyapunov exponent of underlying dynamical system.

Enforce minimum gap ν. Appeal to CLT, plug into SG-MCMC theory.

Ion channel analysis, segmentation. Previously studied using Bayesian nonparametrics on 2000-10000 observations (due to complexity). Our dataset uses 209,634 observations and is much faster.

Q/A:

Variance of SG noise. Use control variates to reduce.

Compared to a minibatch of size 100, with 10 minibatches of size 10, in practice you can get more efficiency because you're exploring different parts of what's going on.

Detailed balance sufficient but not necessary. (Ex. irreversible.)

7 On Gradient-Based Optimization: Accelerated, Distributed, Asynchronous and Stochastic (Michael Jordan, UC Berkeley)

Many new theoretical challenges have arisen in the area of gradient-based optimization for large-scale statistical data analysis, driven by the needs of applications and the opportunities provided by new hardware and software platforms. I discuss several recent results in this area, including: (1) a new framework for understanding Nesterov acceleration, obtained by taking a continuous-time, Lagrangian/Hamiltonian perspective, (2) a general theory of asynchronous optimization in multi-processor systems, (3) a computationally-efficient approach to variance reduction, and (4) a primal-dual methodology for gradient-based optimization that targets communication bottlenecks in distributed systems.

[1]

There's a lot of theoretical challenges in modern optimization and large-scale data analysis: a need to exploit parallelism, stochasticity, tolerating asynchrony.

Our group is working on optimization.

Look for lower bounds in lower bounds and statistics—there's not that many. Optimization has many oracle lower bounds.

Outline:

- Continuous time: Variational, Hamiltonian and symplectic perspectives on Nesterov acceleration. Almost all our theory work has been done in discrete time. A lot of phenomenon you don't see in discrete time.
- Avoiding saddle points, efficiently.
- Stochastically-controlled stochastic gradient
- Ray, a next-generation platform for ML workloads. Stat/systems is a good collaboration.

[1] I went to China and gave this talk to 3000 people. At the end, "What do you think about deep learning?" It's important to be empirical. In my career I want to turn over something more mathematical to the next generation. Now the field is very empirical again. For the young people in the room: there's a whole world of theory.

7.1 Variational, Hamiltonian and Symplectic perspectives on acceleration

Talk about Nesterov acceleration in a different way. Setting: unconstrained convex optimization $\min_{x\in\mathbb{R}^d} f(x)$. Classical vs. gradient descent get convergence rate $O\left(\frac{1}{k}\right), O\left(\frac{1}{k^2}\right)$.

$$\begin{aligned} \text{Classical: } x_{k+1} &= x_k - \beta\nabla f(x_k) && (18)\\ \text{Accelerated: } y_{k+1} &= x_k - \beta\nabla f(x_k) && (19)\\ x_{k+1} &= (1-\lambda_k)y_{k+1} + \lambda_k y_k. && (20)\end{aligned}$$

The lower bound if you only look at a constrained gradient oracle is $O\left(\frac{1}{k^2}\right)$, which surpised people who though classical gradient descent was optimal.

This looks like an implicit step, the leapfrog integrator. Most people in optimization don't think about integration very much. If you're a Bayesian frequentist, it's nice to know these can be linked. (Bayesians care about integration more.)

Two classes of methods:

1. Gradient: Gradient descent, mirror descent, cubic regularizaed Newton's method
2. Accelerated: Nesterov's, accelerated mirror descent, accelerated cubic-regularized Newton's method. (You might go back up: Take the curves pretty quick, a little sloshing is needed.)

Look at the algebra, learn Nesterov's style.

Gradient descent is discretization of gradient flow

$$\dot{X}_t = -\nabla f(X_t).$$

Su, Boyd, Candes 2014: Continuous time limit of accelerated gradient descent is a second-order ODE

$$\ddot{X}_t + \frac{3}{t} + \nabla f(X_t) = 0.$$

Study from point of view of applied ODE research. Bessel equations, etc. Why this differential equation?

What's the best, optimal way to optimize? If you write down a way to optimize in continuous/discrete time, it has to be for some reason.

Our work: general variational approach to acceleration, a systematic discretization methodology. How do you discretize differential questions in physics? You have to discretize the right way to do real science.

What does variational mean? Turn into optimization problem. Ex. eigenvalue. Write down a Rayleigh coefficient and optimize.

Define the Bregman Lagrangian.

$$\mathcal{L}(x,\widehat{x},t) = e^{\gamma_t+\alpha_t}(D_h(x+e^{-\alpha_t}\widehat{x},t) - e^{\beta_t}f(x)).$$

Is there a generative principle for differential equations? Yes, write something that I integrate, optimize that integral. It turned into the main tool for physics.

Lagrangian is function of paths. First term is Bregman divergence.

Here

$$D_h(y,x) = h(y) - h(x) - \langle \nabla h(x), y - x\rangle$$

is Bregman divergence. Bregman divergence takes where you are plus a little velocity, $D_h(x+e^{-\alpha_t}\widehat{x}, t)$, like an energy term. h is the convex distance-generating function. In Euclidean setitng, simplifies to damped Lagrangian,

$$e^{\gamma_t - \alpha_t}\left(\frac{1}{2}\|\dot{x}\|^2 - e^{\beta_t} f(x)\right).$$

Ideal scaling conditions reduce to 1 degree of freedom.

Variational problem over curves

$$\min_X \int \mathcal{L}(X_t, \dot{X}_t, t)dt.$$

Optimal curve is characterized by Euler-Lagrange equation,

$$\frac{d}{dt}\left[\frac{\partial \mathcal{L}}{\partial \dot{x}}(X_t, \dot{X}_t, t)\right] = \frac{\partial \mathcal{L}}{\partial x}(X_t, \dot{X}_t, t).$$

Master ODE:

$$\ddot{X}_t + (e^{\alpha_t} + \dot{\alpha}_t)\dot{X}_t + e^{2\alpha_t+\beta_t}[\nabla^2 h(X_t + e^{-\alpha_t \dot{X}_t}]^{-1}\nabla f(X_t) = 0$$

You choose h fitted to the problem, I can choose a different geometry. You can write a whole bunch of h's and try them out.

Theorem 7.1. *Under ideal scaling, the EL equation has convergence $f(X_t) - f(x^*) \le O(e^{-\beta_t})$.*

Proof: exhibit Lyapunov function for dynamics.

This suggests exponentially fast algorithms, what gives? Let's explore what happens when we set $\beta_t = p\ln t$ to get $O\left(\frac{1}{t^p}\right)$ convergence rate.

$$\alpha_t = \ln p - \ln t \tag{21}$$
$$\beta_t = p\ln t + \ln C \tag{22}$$
$$\gamma_t = p\ln t \tag{23}$$
$$\ddot{X}_t + \frac{p+1}{t}\dot{X}_t + Cp^2t^{p-2}[\nabla^2 h(X_t + \frac{t}{p}\dot{X}_t)]^{-1}\nabla f(X_t) = 0 \tag{24}$$

In continuous time, the notion of rate is not deep, I changed the clock; I can go as fast as I want. The fact that it's not a mystery in continuous time is what helps me. When you go into discrete time, something will break.

Now look at differential equations for different values of p to see what math structure emerges. $p = 2$ is accelerated gradient descent. $p = 3$ is accelerated cubic-regularized

Newton's method. Covariance: As p changes, the path is not changing, the rate of moving changes. The same is not true of gradient flow. There's something important about the path; the rate you move along it is not so important. Acceleration is changing the speed.

How do we discretize to get back to discrete time, stay stable and get the fast rate? Write EL as system of first-order equations. Naive discretization is not stable! Runge-Katta, stiff differential equation methods also don't work.

It became a conundrum, how do we discretize? We took Nesterov's equation and reverse-engineered the discretization. It's stable and achieves the fast rate. But the better way to think about this is to remember symplectic integration. In practice people do symplectic integration. Ex. I know energy is conserved; I'm on a cycle. If I try to discretize, it won't stay on the cycle. Is there a discretization that exactly conserves energy momentum? The physics would still be there.

Jacobi and Hamilton figured this out, conserve energy and momentum. Go to phase space. Look at Fenchel conjugate of velocity, momentum. Look at local Jacobian, preserving volume. Preserve exterior forms. Local stiffness property. This is a factor of 10 faster and more stable. Hamilton-Jacobi-Bellman equations, at root of RL.

Nesterov blends gradient and momentum.

$$x_{k+1} = \frac{p}{k+p} z_k + \frac{k}{k+p} y_k \tag{25}$$

$$z_k = \operatorname{argmin}_z \left[Cpk^{(p-1)} \langle \nabla f(y_k), z \rangle + \frac{1}{\varepsilon} D_h(z, z_{k-1}) \right]. \tag{26}$$

Interlace step from Euler discretization from promixal step $y_k = \operatorname{argmin}_y \left[f_{p-1}(y; x_k) + \frac{1}{2\varepsilon p} \|y - x_k\|^p \right] \leftarrow O\left(\frac{1}{\varepsilon k^{p-1}}\right)$.

If $\nabla^{p-2} f$ is $\frac{1}{\varepsilon}$ Lipschitz and h is uniformly convex of order p, then $f(y_k) - f(x^*) \le O\left(\frac{1}{\varepsilon k^p}\right)$. To get $O\left(\frac{1}{T^3}\right)$, we use f_{p-1} which is 2nd order Taylor approximation to f. Oracle gives access to higher-order derivatives (Hessian).

Underlying Nesterov in continuous time is polynomial Euler-Lagrange equation.

Far from the optimum, gradient flow is too slow. Use momentum. But once you get into neighborhood, momentum hurts you.

Can a mixture of flows do better? Just adding together, get an algorithm with same worst-case rate but in practice goes faster.

Move from a Lagragian to Hamiltonian formulation, to get more robust solvers. Do Fenchel/Legendre transform,

$$L(q, v, t) \to H(q, p, t, \mathcal{E}) \tag{27}$$

$$\left(\frac{dq}{dt}, \frac{dv}{dt}\right) \to \left(\frac{dq}{d\tau}, \frac{dp}{d\tau}, \frac{dt}{d\tau}, \frac{d\mathcal{E}}{d\tau}\right). \tag{28}$$

Geometric functional analysis, Harer et al. Throw it into machinery and get faster algorithms.

Symplectic-mix captured spirit of Nesterov: fast progress early on, and still dive into optimum later. I thought Nesterov is just symplectic integration, but it's convex combination which is not dynamics, some rotation. We're still trying to find a principled way of thinking about it.

Nonconvex, stochastic optimization? Get SDEs. Is there an optimal way to diffuse, some Fokker-Planck equation? Hybrid/Hamiltonian Monte Carlo: integrate with symplectic integrator like leapfrog integrator. It's the best algorithm for Monte Carlo. Integration is behind optimization and Bayesian inference.

7.2 Avoiding saddlepoints efficiently

Efficient means good dimension dependence.

Originally when I was running neural nets, loss comes down fast and plateaus. Other problems: tensor completion: there's a ring of saddle points around the optimum. Number of iterations for gradient descent is dimension-free.

Analyze perturbed gradient descent (PGD): For $t = 0, 1, \ldots,$, do

1. If perturbation condition holds then $x_t \leftarrow x_t + \xi_t$, $\xi_t \sim B_0(r)$.
2. Gradient step.

To analyze, add more conditions: ρ-Hessian Lipschitz. For l-gradient Lipschitz and ρ-Hessian Lipschitz, PGD converges to ε-second-order stationary point. Dimension dependence is polylog, $O(\ln^4(d)/\varepsilon^2)$.

There is a little differential geometry in the analysis: look at stuck region around saddle point. If you are in the stuck region, a perturbation will get you out; if you are out, flow will take you out.

When gradient is small, add noise and wait a while before adding it again.

7.3 Stochastically-controlled stochastic gradient

SVRG has favoriable rates in terms of ε but has a n factor. Compute full gradient, use it as control to get variance down. Computational cost is too high because you have to take full gradient. What if you take a small subset to use as reference gradient?

Stochastically controlled SG: General convex $\frac{1}{\varepsilon^2} \wedge \frac{n}{\varepsilon}$, strongly convex $\left(\frac{1}{\varepsilon} \wedge n + \kappa\right) \ln\left(\frac{1}{\varepsilon}\right)$.

Algorithm: randomly pick I with size B, generate $N \sim Geo(\gamma)$, for $k = 1 : N$ do SGD with reference gradient.

SCSG fixes $B_j = B(\varepsilon)$, other version increases it.

8 Sampling Polytopes: From Euclid to Riemann (Yin Tat Lee, Microsoft Research and University of Washington)

We introduce the Geodesic Walk for sampling Riemannian manifolds and apply it to the problem of generating uniform random points from the interior of a convex polytope in n-dimensional Euclidean space. The walk is a simulation of a stochastic differential equation defined using a convex barrier function; each step is the solution of an ordinary differential equation. It improves the time complexity of sampling polytopes and is the first walk to achieve sub-quadratic complexity. Part of the talk will be spent introducing relevant aspects of manifolds, stochastic processes, efficient solution of differential equations, and how this walk overcomes the bottlenecks of random walks in Euclidean space. No background in string theory (or Riemannian geometry) will be assumed. Joint work with Santosh Vempala.

My dream: tell the complexity of a convex problem by looking at the formula.

This is hard. Ex. given a graph with m edges and n vertices, min st cut, $\min_{f(s)=0,f(t)=1}\sum_{x\sim y}|f(x)-f(y)|$. You can smooth this problem, do Nesterov gradient descent, etc., but our final goal is to solve the original problem; think about how much error you incur. If you true to use AGD, you get $m^{1.5}$. If you write as LP and use Laplacian solver you get $m^{1.5}$. Using packing LP, you get $m^{1.5}$. In the past few years, I've been trying to understand given the problem, what oracle setting and what algorithm to use? M14 gets $O(m^{\frac{10}{7}})$; LS14 get $O(m\sqrt{n})$.

Min cost flow problem $\min_{B^Tf=d,0\le f\le 1} c^Tf$.

Submodular minimization $\min_{S\in 2^{[n]}} f(S)$ where f satisfies diminishing returns $f(S+u)-f(S)\le f(T+e)-f(T)$, $T\subset S, e\notin S$. f can be extended to convex function on $[0,1]^n$, subgradient can be computed in n^2 time, can be solved in $\widetilde{O}(n^3)$.

Book reference: Geometric algorithms and combinatorial optimization

Algorithmic convex geo: given convex set K, we have operations

1. membership: check $x\in K$ $(l_K(x))$
2. separation: assert $x\in K$, or find hyperplane separating $(\partial l_K(x))$
3. width, $\min_{x\in K} c^Tx$. $(l_K^*(c))$
4. Optimize $\operatorname{argmin}_{x\in K} c^Tx$ $(\partial l_K^*(c))$
5. sample $g(x)\mathbb{1}_K$, g logconcave $(\approx e^{-g}1_K)$
6. integrate g: compute $\int_K g(x)\,dx$, g logconcave. $(\int_K g(x)\,dx)$

They are all equivalent in polytime.

Why those operations? Define dual

$$f^*(c) = \operatorname{argmax}_x c^Tx - f(x).$$

Let $l_K = \infty\mathbb{1}_{K^c}$. All the oracles can be written in terms of $l_K, l_K^*, 1_K$; see above. We are getting tight poly equivalence between membership, width, separation, optimization. Convex optimization is relation between separation and optimization. The poly equivalence is classically by ellipsoid method. Traditionally ellipsoid method is viewed as impractical; now we have efficient method. We'll focus on membership to sampling.

Sampling: given convex set K, sample point from uniform distribution on K.

Generalized problem: given logconcave distribution f, output point according to f. Useful for optimization, integration/counting, learning, rounding.

This is the best way to minimize convex function with noisy value oracle, and only way to compute volume of convex set.

I'll focus on the problem of sampling from a convex set K.

Application: convex bandit. For each round $t=1:T$,

- adversary selects convex loss function ℓ_t
- Choose (possibly randomly) x_t from unit ball in n dimensions based on past obsevations
- Receives loss/observations $\ell_t(x_t) \in [0,1]$. Nothing else about ℓ_t is revealed.

Measure performance by regret

$$R_T = \sum_{t=1}^{T} \ell_t(x_t) - \min_{x \in K} \sum_{t=1}^{T} \ell_t(x).$$

There is a good fixed action but we only learn one point each iteraction, adversary can give confusing information. Gold standard is $O(\sqrt{T})$. After a decade of research we have $R_T = n^{10.5}\sqrt{T}$.

Two ways to think about sampling problem.

1. Oracle setting: membership oracle answers Yes/No to "$x \in K$".

   Start with a ball $x + rB \subseteq K \subseteq x + \text{poly}(n)rB$.

2. Explicit: given description of K, ex. polytope.

Oracle setting: hit and run gives $O(n^3)$. Conjuectured lower bound is $\Omega(n^2)$. Conjectured optimal is ball work. At x, pick random y from $x + \delta B_n$. If $y \in K$, go to y, else sample again.

This walk may get trapped on one side if set is not convex. To measure, see if there is bottleneck. Define isoperimetric constant

$$\phi_K = \min_S \frac{Area(\partial S)}{\min(\text{Vol}(S), \text{Vol}(S^c))}.$$

Theorem 8.1. *Given a random point in K, we can generate another in $O\left(\frac{n}{\delta^2 \phi_K^2} \ln\left(\frac{1}{\varepsilon}\right)\right)$ iterations of Ball Walk.*

Isoperimetric constant of convex set: not affine invariant and can be arbitrarily small. Ex. $1 \times L$ rectangle. Normalize K such that $\text{Cov}(K) = I$. K is isotropic if mean 0 and $\text{Cov}(K) = I$.

Theorem 8.2. *If $\delta < \frac{0.001}{\sqrt{n}}$ ball walk stays inside set with constant probability.*

Theorem 8.3. *Given a random point in iso K, generate another in $O(\frac{n^2}{\phi_K^2} \ln\left(\frac{1}{\varepsilon}\right))$. ($\varepsilon$ is total variation between resulting distribution and uniform distribution.)*

To make body isotropic we can sample the body to compute covariance. "Chicken and egg".

Conjecture 8.4 (Kannan-Lovász-Simonovits conjecture)**.** *For any isotropic convex K, $\phi_K = \Omega(1)$.*

If true, Ball Walk takes $O(n^2)$ iterations for isotropic K.

Related conjecture

Conjecture 8.5 (Slicing conjecture)**.** *Any unit volume convex set K has slice with volume $\Omega(1)$.*

Conjecture 8.6 (Thin-shell conjecture)**.** *For isotropic convex K, $\mathbb{E}[(\|x\| - \sqrt{n})^2] = O(1)$.*

If you try to solve min s-t cut, min cut max flow, where does $\sqrt{n}$ come from: Any symmetric convex set can be approximated by ellipsoid inside, such that scaling by $\sqrt{n}$ it contains the set. Any convex set is approximated $1 + \frac{1}{\sqrt{n}}$ factor. This is probabilistic statement we don't know how to use.

Conjecture 8.7 (Generalized Levy concentration)**.** *For logconcave p¡ 1-Lipschitz f,*

$$\mathbb{P}(|f(x) - \mathbb{E}f| > t) = \exp(-\Omega(t))$$

All of these conjectures are easy to prove for ellipsoids; it asks if all convex sets looks like ellipsoids.

Main result: Lovasz-Simonovits93, $\phi = \Omega(1)n^{-\frac{1}{2}}$.

What if we cut the body by sphere only? $\sigma_K := \sqrt{\frac{n}{\mathrm{Var}(\|X\|^2)}} \geq \phi_K$. This is still difficult. Eldan 2012: $\phi = \widetilde{\Omega}(1)\sigma = \widetilde{\Omega}(1)n^{-0.333}$, LVempala 2016, $\phi = \Omega(1)n^{-.25}$. We have $O(n^{2.5})$ mixing for ball walk.

Do you know better way to bound mixing time of ball walk?

Sampling for polytopes. Input: polytope with m constraints and n variables. Sample from uniform on K.

1. Dikin walk, KN09: mn iterations, $mn^{1.38}$ time/iteration (cost of matrix inversion)
2. LV: ball walk $n^{2.5}$, mn time/iterations
3. LV16: $mn^{0.75}$, $mn^{1.38}$. (To sample from ellipse, solve a linear system.)

First sub-quadratic algorithm! For previous algorithms, hypercube was hardest. Here, hypercube takes less time, but we can't show it's the worst case.

How does nature mix particles? Brownian motion works for sampling on $\mathbb{R}^n$. However convex set has boundary. Options:

1. Reflection
2. Remove boundary by blowing up. Construct manifold which projects to convex set. This requires explicit polytopes.

Stretch points near boundary to ∞. If you do a random walk you will drift to ∞. The distortion makes the hard constraint become soft.

Use Riemannian manifolds. n-dimensional manifold is n-dimensional surface. At every point there is a tangent space consisting of all directions along the manifold. For any point there is an inner product defined locally. Inner product in T_pM depends on p. Useful for defining length of curve $c : [0,1] \to M$,

$$L(c) = \int_0^1 \left\| \frac{d}{dt} c(t) \right\|_{c(t)} dt.$$

Distance is infimum over all paths.

Generalized ball walk: at x, pick random y from D_x where $D_x = \{y : d(x,y) \le 1\}$.

Hessian manifold is subset of $\mathbb{R}^n$ with inner product defined by $\langle u, v \rangle_p = u^T \nabla^2 \phi(p) v$.

For polytope $\{\forall i, a_i^T x \ge b_i\}$ use log barrier function $\phi(x) = \sum_{i=1}^m \ln\left(\frac{1}{s_i(x)}\right)$. $s_i(x) == a_i^T x - b_i$ is distance from x to constraint i. p blows up when x is close to boundary, walk is slower when close to boundary.

This converges to the boundary (∞).

Make modifications. Note if $p(x \to y) = p(y \to x)$ then stationary distribution is uniform. To make Markov chain symmetric, use

$$\widetilde{p}(x \to y) = \min(p(x \to y), p(y \to x)), \text{ if } x \ne y. \tag{29}$$

Dikin walk: at x, pick $y \in D_x$. If $x \in D_y$, accept with probability $\min(1, \frac{\mathrm{Vol}(D_x)}{\mathrm{Vol}(D_y)})$.

Dikin ellipsoid is contained in K. Why not pick next step from blown-up Dikin ellipsoid, blow up by $\sim \sqrt{\frac{n}{\ln m}}$. Problem: success probability exponentially small. Hypercube is worst-case for previous algorithms.

Look at Dikin walk, possible y's you go to is not symmetric around x because you reject close to boundary. This is like drift back to the center. Taking step size to 0, Dikin walk becomes SDE,

$$dx_t = \mu(x_t)\, dt + \sigma(x_t)\, dW_t,$$

$\sigma(x_t) = \phi''(x_t)^{-\frac{1}{2}}$, $\mu(x_t)$ is drift towards center. Drift is from Fokker-Planck equation. Make stationary distribution constant, New walk: $x_{t+h} = x_t + h \cdot \mu(x_t) + \sigma(x_t) W$ with $W \sim N(0, hI)$. This doesn't make sense—goes outside manifold. Exponential map: find shortest path start with x in direction; traces geodesic on manfold.

$$x_{t+h} = \exp_{x_t}(h/2 \cdot \mu(x_t) + W)$$

This has discretization error, do metropolis filter. Filter is complicated.

Key lemma 1: provable long geodesic. For any geodeisc, if direction is random, length $\widetilde{O}(n^{\frac{1}{4}})$. Faster algorithm for max-flow.

Key lemma 2: massive cancellation.

Q's:

- We have no background on ODE/SDE, RG. Improve running time?
- How to avoid filtering?
- Is there a way to tell a walk mixed or not? Even if we cannot prove KLS, algorithm can still stop early.

- Higher order method in SDE useful in MCMC?
- Other suggestions/heuristics?

HMC makes acceptance probability higher.

9 Understanding Generalization in Adaptive Data Analysis (Vitaly Feldman, IBM Almaden)

I will describe recent work on algorithms for ensuring generalization when random samples are reused to perform multiple analyses adaptively. I will also discuss connections to the problem of understanding generalization of algorithms for stochastic convex optimization and some challenging open problems.

(with Dwork, Hardt, Pitassi, Reingold, Roth 14, 15) New results (F, Steinke 17)

We have a learning problem described by distribution P over domain X. Input is dataset $S = (x_1, \dots, x_n) \sim P^n$. Do analysis A, get model $f = A(S)$. What is $\mathbb{E}_{x\sim P}[loss(f, x)]$.

In practice data analysis is adaptive, depending on previous analyes of same dataset. Starting from the second analysis, the iid assumption no longer holds. If we had unlimited data, we could do conservative data splitting, split into k chunks, run each on separate chunk. In practice, data is never unlimited; can we do better than this conservative approach?

"Thou shalt not test hypotheses suggested by data." With exception from some limited cases, there are no good solutions. It's often ignored, causing spurious conclusions. "Quiet scandal of statistics." (Leo Breiman, 1992)

ML then was young. Split into training and testing. Training used to run learning algorithm, testing used to measure expected loss. If test set used only once, test error of f is $\approx \mathbb{E}_{x\sim P}[loss(f, x)]$. But test set was used multiple times.

Since then the area has grown. Split 3-ways, training, validation, and testing. This mitigates overfitting to the test-set but is still unsatisfying. It doesn't solve all the problem; with a lot of tuning we could overfit to validation set. The problems which are solved are solved by the same conservative data-splitting.

Find a way to mitigate dependencies. Goal: given S compute v_i's close to running A_i on fresh samples. Each analysis is a query. Design algorithm for answering adaptively-chosen queries.

Queries are

$$A_i(S) = \frac{1}{n} \sum_{x\in S} \phi_i(x), \quad \phi_i : X \to [0, 1].$$

Example: $\phi_i = Loss(f, x)$. $|v_i - \mathbb{E}_{x\sim P}[\phi_i(x)]| \le \tau$ w.p. $1 - \beta$.

Can use to measure correlations, moments, accuracy/loss.

Given k non-adaptive query functions $\phi_1, \dots, \phi_k$ and n iid samples from P estimate $\mathbb{E}_{x\sim P}[\phi_i(x)]$. Use empirical mean $\mathbb{E}_S[\phi_i] = \frac{1}{n} \sum_{x\in S} \phi_i(x)$.

What if we use $\mathbb{E}_S[\phi_i]$? There are sequences of queries where the number of samples required is $n \ge \frac{k}{\tau^2}$, the kind that would often arise in boosting, etc. This is not much better than data splitting $n = O\left(\frac{k \ln k}{t^2}\right)$.

There exists an algorithm that can answer k adaptively chosen SQ's with accuracy τ for

$$n = \widetilde{O}\left(\frac{\sqrt{k}}{\tau^{2.5}}\right)$$

to estimate within τ. Data splitting gives $\widetilde{O}\left(\frac{k}{\tau^2}\right)$.

BNSSSU analyze better to get $n = \widetilde{O}\left(\frac{\sqrt{k}}{\tau^2}\right)$, and generalize to low-sensitivity analyses $|A_i(S) - A_i(S')| \le \frac{1}{n}$ when S, S' differ in single element.

Analyses rely on differential privacy. If you look at output distribution on adjacent datasets, they will be multiplicative close except δ, $\mathbb{P}_M[M(S) \in Z] \le e^{\varepsilon}\mathbb{P}_M[M(S') \in Z] + \delta$.

Differential privacy implies generalization.

Differential privacy is stability. It implies strongly uniform replace-one stability and generalization in expectation.

DP composes adaptively: composition of k ε-DP algorithms: for every $\delta > 0$, is $(\varepsilon\sqrt{k \ln\left(\frac{1}{\delta}\right)}, \delta)$-DP.

Take any DP algorithm, it will work in this adaptive setting. Answering low-sensitivity queries ($\max_{S,S'} |A(S) - A(S')| \le \frac{1}{n}$) is the most basic algorithm in DP. Answer query A with $A(S) + \zeta$, ex. $\zeta \sim N(0, \tau)$.

Can we go beyond low-sensitivity queries? Error scales with worst-case sensitivity, $\Delta\sqrt{n}k^{\frac{1}{4}}$ where Δ is worst-case sensitivity, $\max_{A,S,S'} A(S) - A(S')$.

We show there exists an algorithm that for any adaptively chosen sequence $A_1, \dots, A_k : X^t \to \mathbb{R}$ given $n = \widetilde{O}(\sqrt{k}t)$ iid samples from P outputs $v_1, \dots, v_k$ such that whp for all i,

$$|\mathbb{E}_{S\sim P^t}[A_i(S)] - v_i| \le 2\sigma_i$$

where $\sigma_i = \sqrt{\mathrm{Var}_{S\sim P^t}[A_i(S)]}$.

Get error that scales with $\frac{\mathrm{Var}_{S\sim Pt}[A_i(S)]}{\sqrt{n}}k^{\frac{1}{4}}$.

Stable median algorithm: split into chunks m of size t, $n = tm$. Find an approximate median with DP relative to U, value greater than bottom $\frac{1}{3}$ and smaller than top $\frac{1}{3}$ in U.

Use a simpler algorithm based on exponential mechanism: output $v \in T$ with probability $\propto e^{-\varepsilon\left||\{u \le u : u \in Y\}| - \frac{m}{2}\right|}$. Uses $O\left(\frac{\ln r}{\varepsilon}\right)$. Get stability and confidence amplification for price of one log factor.

Key result: differential privacy approximately preserves quantiles.

Let M be DP algorithm that on input $U \in Y^m$ outputs $\phi : Y \to \mathbb{R}$, $v \in \mathbb{R}$. Then whp over $U \sim D^m$, $(\phi, v) = M(U)$,

$$\mathbb{P}_{y\sim D}[v \le \phi(y)] \approx \mathbb{P}_{y \mathrm{im} U}[v \le \phi(y)].$$

If v is within $\left(\frac{1}{3}, \frac{2}{3}\right)$ empirical quantiles, then v within $\left(\frac{1}{4}, \frac{3}{4}\right)$ quantiles, mean $\pm 2\sigma$. If ϕ is well-concentrated on D then easy to prove high-probability bounds.

Limitations: (Hardt, Ullman 14, Steinke, Ullman 15). Any algorithm for answering k adaptively chosen SQ's with accuracy τ requires $n = \Omega(\sqrt{k}/\tau)$ samples. (In high dimension or under crypto assumptions)

Verification of responses to queries: $n = O(\sqrt{c}\ln k)$ where c is number of queries that failed verification. Get data splitting if overfitting, reusable holdout, maintain public leaderboard in competition.

Open problems:

- Queries depend only on previous answers: Does there exist a SQ analyst whose queries require more than $O(\ln k)$ samples to answer withing 0.1 accuracy/confidence?
- Fixed "natural" analyst/learning algorithm. When does adaptivity require more samples?

  Ex. gradient descent for stochastic convex optimization.

Let $K = B_2^d(1) = \{x : \|x\|_2 \le 1\}$. Let $F = \{f \text{ convex} : \forall x \in K \, \|\nabla f(x)\|_2 \le 1\}$ class of convex 1-Lipschitz functions.

We don't know how many samples we need for this problem to know that the solution generalizes. Sample machinery of uniform convergence gives that $O\left(\frac{d}{\varepsilon^2}\right)$ examples suffices and is necessary (F16). But SGD solves using $O\left(\frac{1}{\varepsilon^2}\right)$ samples.

Empirical gradient is $\nabla f_S(x_t) = \frac{1}{n}\sum \nabla f_i(x_t)$. On fresh samples, $\|\nabla f_S(x_t) - \nabla f_P(x_t)\|_2 \le \frac{1}{\sqrt{n}}$.

Overall, get $\frac{d}{\varepsilon^2}$ statistical queries with accuracy ε in $\frac{1}{\varepsilon^2}$ adaptive rounds. How many samples do you need to answer these adaptive queries? Sample splitting gives $O\left(\frac{\ln d}{\varepsilon^4}\right)$, DP gives $\widetilde{O}\left(\frac{\sqrt{d}}{\varepsilon^3}\right)$, both incomparable and incomparable to uniform convergence bounds.

(Q: Address conjecture that SGD is form of regularizer? The issue here is reuse.)

Uniform convergence is not needed?

Further directions:

- Going beyond tools from DP: other notions of stability for outcomes, max/mutual information.
- Generalization beyond uniform convergence.
- Using in practice.
- Are some types of adaptivity more benign than others?

10 Gradient Descent Learns Linear Dynamical Systems (Moritz Hardt, Google)

We prove that gradient descent efficiently converges to the global optimizer of the maximum likelihood objective of an unknown linear time-invariant dynamical system from a sequence of noisy observations generated by the system. Even though objective function is non-convex, we provide polynomial running time and sample complexity bounds under strong but natural assumptions. Linear systems identification has been studied for many decades, yet, to the best of our knowledge, these are the first polynomial guarantees for the problem we consider.

Joint work with Tengyu Ma and Ben Recht.

Does SGD learn a linear dynamical system? Yes!

We have a distribution over sequences $\{(x_t, y_t)\}_{t\in[T]}$. Population risk

$$f(\widehat{\theta}) = \mathbb{E}_{\{x_t\},\{\xi_t\}}\left[\frac{1}{T}\sum_{t=1}^{T}\|\widehat{y}_t - y_t\|^2\right]$$

where $\widehat{\theta} = (\widehat{A}, \widehat{B}, \widehat{C}, \widehat{D})$ is the parameters of learning system.

$\widehat{y}_t$ is the prediction of the learned system from the correct initial state.

Population risk is non-convex. Quasi-convexity is sufficient condition for convergence of SGD,

$$\langle \nabla f(\theta), \theta - \theta^*\rangle \geq \tau(f(\theta) - f(\theta^*)),$$

"gradient correlated with descent direction".

Under what conditions is risk quasi-convex? We need terminology from control theory. Controllable canonical form is $\begin{pmatrix} 0 & I \\ -a_n & -[a_{n-1}, \ldots, a_1]\end{pmatrix}$, $B = [0, \ldots, 0, 1]^T$, shift with linear combination at bottom. There are only n relevant parameters, and we don't need to worry about B. The bottom row tells exactly what the characteristic polynomial is, $z^n + a_1 z^{n-1} + \cdots + a_n$. Stability (inputs don't blow up) means $\rho(A) < 1$.

Polynomial roots condition (Pac-Man condition): consider $q(z) = 1 + a_1 z + \cdots + a_n z^n = z^n p_a(z^{-1})$. We require $\Re(q(z)) > |\Im(q(z))|$ for all $|z| = 1$.

This ensures stability of system (Rouché), convexity of constraint region, and quasi-convexity of population risk.

Assume Pac-Man, pairwise independent inputs with mean 0 and variance 1, iid noise with mean 0 and variance σ^2. Projected stochastic gradient descent with N samples of length T learns unknown system of order n with $f(\widehat{\theta}) \precsim \sqrt{\frac{n^5+\sigma^2 n^3}{TN}}$.

Overparametrization helps. Learn order n system with $n' > n$ parameters. Replace Pac-Man with weak separation of roots condition. Note for large enough n' problem turns into linear regression.

Proof outline:

1. Relate objective function to transfer function of system.

2. Use properties of transfer function to show quasi-convexity.

First step: Unroll objective function using definition.

$$f(\widehat{\theta}) \approx \left\|D - \widehat{D}\right\|^2 + \sum_{k=0}^{\infty}(\widehat{C}\widehat{A}^k B - CA^k B)^2.$$

Define idealized risk

$$g(\widehat{A}, \widehat{C}) = \sum_{k=0}^{\infty}(\widehat{C}\widehat{A}^k B - CA^k B)^2.$$

Transfer function is $G(z) = \sum_{t=1}^{\infty} z^{-t} C A^{t-1} B$. Evaluate at a point on complex unit circle. Get nice curves which correspond to useful values of θ.

Pass from time to frequency domain. If $\rho(\widehat{A}) < 1$,

$$g(\widehat{A}, \widehat{C}) = \int_0^{2\pi} |\widehat{G}(e^{i\theta}) - G(e^{i\theta})|^2 \, d\theta.$$

Proof: $G(e^{i\theta})$ is Fourier transform of $\{CA^{t-1}B\}$. Parseval's identity.

How does this help establish quasi-convexity?

Lemma: $G(z) = C(zI - A)^{-1}B = \frac{c_1+c_2z+\cdots+c_nz^{n-1}}{z^n+a_1z^{n-1}+\cdots+a_n}$.

In which domain is $(G(z) - \widehat{G}(z))^2$ quasi-convex? This reduces to the case $f(\widehat{u}, \widehat{v}) = \left(\frac{\widehat{u}}{\widehat{v}} - \frac{u}{v}\right)^2$. It's quasiconvex over $\left\{(\widehat{u}, \widehat{v}) : \frac{\widehat{v}}{v} > 0\right\}$.

Idealized risk is quasi-convex under PacMan condition. Śoderstŕom, Brief paper (1973). Some properties of the output error method (1982).

- SGD solves SysID under PacMan condition. What condition is sufficient and necessary?
- Overspecification helps a lot, eventually becomes linear regression. (When?)
- Right dependence on N, T. What about dependence on n?
- Build theory for sequence learning. What can we learn from control theory?

11 Your Neural Network Can't Learn Mine (John Wilmes, Georgia Institute of Technology)

We study the question of what functions can be learned by neural network training algorithms, and prove surprisingly general lower bounds in the realizable case, where the function to be learned can be represented exactly as a neural network. Previous lower bounds apply only for special activation functions or unnatural input distributions, or for highly restricted classes of algorithms. We prove lower bounds for learning a family of functions realizable as small neural networks with a single hidden layer, with any activation function from a broad family (including sigmoid and ReLU) over any "nice" input distribution (e.g., Gaussian or uniform). Our results apply to any statistical query algorithm, including all known neural network training algorithms, regardless of the model architecture, loss function, or optimization routine.

Joint work with Le Song, Santosh Vempala, and Bo Xie.

"I know it works well in practice, but what about in theory?"

We give lower bounds for realizable case. (Everything is close to realizable.) There are classical lower bounds:

- NP-hard to train neural net with 3 threshold neurons (Blum-Rivest 1993)
- Complexity/crypto assumptions: can't efficiently learn small depth-2 networks even improperly (Klivas-Sherstov 2006; Daniely-Linial-Shalev-Schwarts, 2014)

Distributions that show these are difficult are unnatural.

Positive results: learn LDS, single layer ReLU with special structure (Brutzkus-Globerson 2017), can learn some NNs via non-NN algorithms (JSA16, ZLJ15, ABGM12).

Can we get any formal guarantees for a family with 1 hidden layer? Choose your favorite network architecture, activation units, loss function, gradient descent variant, regularization scheme.

Theorem 11.1. *If using only black box functions of input, need batch size $\Omega(n)$ or $2^{\Omega(n)}$ time.*

Shamir (2016) gives exponential lower bounds against vanilla SGD with mean-square loss, regardless of batch size. The functions are nonrealizable but construction is similar in spirit.

Specific parameters are not particularly important.

Lower bound applies to algorithms of following form: estimate $v = \mathbb{E}_{(x,y)\sim D}(h(W, x, y))$ where W are current weights of NN. (x, y) is labeled example of input distribution D and H arbitrary $[0, 1]$-valued function. Use v to update W.

Statistical query algorithms (Kearns 1993) interacts with input distribution via one of the oracles

1. STAT(τ): query $f : X \to [0, 1]$, response is $|\mathbb{E}_D f - v| \le \tau$.
2. VSTAT: $|\mathbb{E}_D f - v| \le \max\left\{\frac{1}{t}\sqrt{\frac{(\mathbb{E}_D f)(1-\mathbb{E}_D f)}{t}}\right\}$. (Seldman-Grigorescu-RVX12)
3. 1-BIT: Query is $f : X \to \{0, 1\}$. Response is $f(x)$ for $x \sim D$. (Ke Yang, 2001)

(Is the lower bound of $\Omega(n)$ batch size damning?)

Sigmoid function of sharpness s is $\frac{e^{sx}}{1+e^{sx}}$.

Theorem 11.2. *There exist functions $f : \mathbb{R}^n \to \mathbb{R}$,realized as NNs with single sigmoid layer such that any statistical algorithm learning f over log-concave product distribution needs d queries to STAT(τ) or VSTAT($1/\tau^2$) or $\frac{1}{\tau^2}$ queries to 1-BIT where $\tau = O\left(\frac{1}{s\sqrt{n}}\right)$ and $d = 2^{\Omega(n)}$.*

(SGD could be more powerful though—give unbiased estimator. Lower bound is against adversarial oracle)

Hard family of functions: $F_\sigma : \mathbb{R} \to \mathbb{R}$ affine combination of σ-units, almost periodic on $[-\widetilde{O}(n), \widetilde{O}(n)]$, period $\frac{1}{s}$.

For all $S \subseteq [n]$ with $|S| = \frac{n}{2}$, take $f_S(x) = F_\sigma\left(\sum_{i\in S} x_i\right)$.

Classical tool is statistical dimension: $SDA(C, D, \overline{\gamma}) = d$ if d is max such that for all $C' \subseteq C$, if $|C'| \ge |C|/d$, then average correlation is $\rho_D(C') \le \overline{\gamma}$.

$:\mathbb{R} \to \mathbb{R}_{\ge 0}$ is ε-mollifier if $\mathrm{Supp}(\chi) \subseteq [-\varepsilon, \varepsilon]$ and $\|\chi\|_1 = 1$. Write $\chi_y(x) = \chi(x - y)$.

It's enough that on average my functions have small correlation even if I just look at functions on some neighborhood.

Claim: over logconcave product distribution D, for all $y \in \mathbb{R}$, covariance is $\mathrm{Cov}_D(\chi_y \circ f_S, \chi_y \circ f_T) = O(\mathbb{E}_D(\chi_y f_S)^2/(s^2 n))$.

...

Open: what can neural nets learn? What if weight matrix is generic? Experiments suggest difficulties. Smaller sensitivity bounds for concepts realized with greater depth?

12 Machine Learning for Healthcare Data (Katherine Heller, Duke University)

We will review multiple ways of acquiring healthcare data and the machine learning methods which are currently being used for analysis. These include: 1) Disease trends and other predictions from electronic health record (EHR) data 2) Mobile apps and devices for improving the granularity of recorded health data and 3) The combination of mobile app and social network information. Current work in these areas will be presented and the future of machine learning contributions to the field will be discussed.

12.1 Electronic health record

We make gaussian process-based models for chronic kidney diease and sepsis.

eGFR is a measure of kidney function.

An age 47 patient has untreated diabetes and high blood pressure, normal kidney function; at age 51 patient goes to ER for kidney failure.

The first few years is a missed opportunity to prevent or delay kidney failure.

$< 10\%$ with moderate CKD (chronic kidney disease) and $< 50\%$ with severe CKD are even aware of illness.

We want to create ML methods to predict what will happen to someone in the future, ex. whether they are likely to have kidney failure.

We want to look at a bunch of lab values and predict whether someone is at risk. Schulam and Saria, 2015: Use a Gaussian process model. There are population effects, latent subpopulation curve, individual long-term deviations, and individual transient deviations (GP).

We want to correlate trajectories from different labs. We look at eGFR and 5+ other measurements. Use data earlier in time to predict future lab values. For most lab values we do better by introducing correlation.

CKD is correlated with heart disease and diabetes.

Goal: jointly model risk of future loss of kidney function and cardiac events. Use a conditional independent joint likelihood model. Use a Poisson process model with conditional likelihood on events.

There's a lot of red tape and bureaucracy in getting these into actual healthcare. We use with someone in user interface.

We haven't looked at how treatment affects . How does presenting this info to providers make a difference?

Sepsis (infection contracted, usually in hospital, often through some line put in, serious).

People use Early Warning Scores. Get points for being above/below certain thresholds. What's currently being used is threshold-y. Can we move away from that into a more probabilistic setting and do better? The answer is almost always yes.

We introduce a temporal component. Fall back on gaussian process regression with 0 mean and certain kernel and noise parameters.

At the end you want a classification system. Take the output from expected trajectory and feed it into classifier. Minimize expected loss wrt gaussian process parameters. Use conjugate gradient (Lanczos).

Time series are variable in length. Use RNN classifier.

Use to flag people who are at risk for sepsis.

The current system if very interpretable (ex. blood pressure ≤ 90 is bad).

12.2 National surgery quality improvement program

15/100 surgical procedures result in complications.

We use NSQIP (national surgery quality improvement program) data.

Our goal is to take the data and use it to improve outcomes for patients.

First thing we tried is sparse logistic regression model. Look at complications, AUC (there's a fair amount of signal—you can predict risk for these complications). We're trying to do a better than a surgeon assessing by sight.

There are some things we can't do much about, ex. esophageal cancer. Some things we can intervene in, ex. nutritional deficiency.

App gives risks of complications.

Clustering of procedural codes: now is done by hand. We're interested in using a (hierarchical) clustering methodology to find procedural codes that go together. This leads to modest improvement in 7/8 outcomes.

Transfer learning: Patients at Duke probably not in national database. Transfer between national and local data. Downweight national data, upweight local data. Normal factor analysis.

Hierarchical infinite factor model improves our ability to make predictions. One downside is that it uses standard sampling methods, up to 100,000 people now. We're looking at speeding this up using stochastic gradient, Nose-Hoover Thermostats Sampling.

Everything I talked about was post-operative. My goal is to create a prediction model that goes from the first to last visit, in real time during surgery. We're working on pre-operative end now. There's more leeway for intervention now. Separate people depending on whether people are at low or high risk (if low risk, do pscreen by phone; if high risk, refer to clinic).

12.3 Infectious disease

Model the spread of infection (influenza) in student dorms. How to use new sensors (cell phones, mobile apps), what challenges they bring.

I was loosely involved in several studies, in MIT and UMichigan. Track how sick people are and who they interact with. Can we predict someone's likelihood of getting sick with their symptoms?

We use a graph-coupled HMM model. Have a HMM associated with every person in a social network. Look at people interacting over time. Whether are sick/healthy in the future depends on whether they interacted with people who are sick/healthy. Inference in coupled HMM is very hard. (Can we predict if we have 10% of links missing?)

Did I get infected by someone in social network, or someone outside social network? We compare to standard epidemiology models. When someone becomes sick, they stop filling out their surveys, and you get empty boxes. Our algorithm can infer than that they are sick.

University of Michigan: Individual students have covariates (sleeping, washing hands, etc.). Incorporate into model.

For inference, do Gibbs-EM. There's a lot of use of stochastic variational Bayes.

Look at multiple sclerosis patients, but no data is being collected. It's a complicated disease. How can we monitor it more continuously? We developed an app with embedded consent, disease history survey, daily symptom and medication survey, and 5 activities. There's a pairing with fitness tracker. We try to make it low work. If nothing changed, they can just say nothing changed and repopulate. They can see what symptoms and task performance has been like.

Initial analyses: develop sparse logistic regression for likelihood of each symptom experience, hierarchical layer based on GP for time series, discover hidden subpopulations, evaluate efficiency of intervention.

QA: Share with doctors. Are doctors sharing with patients? Ex. tell them they're at high risk. You should evaluate whether you want the surgery. We are just starting to introduce this to clinicians.

13 Machine Learning Combinatorial Optimization Algorithms (Dorit Hochbaum, UC Berkeley)

We present a model for clustering which combines two criteria: Given a collection of objects with pairwise similarity measure, the problem is to find a cluster that is as dissimilar as possible from the complement, while having as much similarity as possible within the cluster. The two objectives are combined either as a ratio or with linear weights. The ratio problem, and its linear weighted version, are solved by a combinatorial algorithm within the complexity of a single minimum s,t-cut algorithm. This problem (HNC) is closely related to the NP-hard problem of normalized cut that is often used in image segmentation and for which heuristic solutions are generated with the eigenvector technique (spectral method).

HNC has been used, as a supervised or unsupervised machine learning technique. In an extensive comparative study HNC was compared to leading machine learning techniques on benchmark datasets. The study indicated that HNC and other similarity-based algorithms provide robust performance and improved accuracy as compared to other techniques. Furthermore, the technique of "sparse-computation" enables the scalability of HNC and similarity-based algorithms to large scale data sets.

Let $G = (V, E, (w_{ij})_{\{i,j\}\in E})$ be a weighted undirected graph. The capacity of (A, B) subsets of vertices in a graph is

$$C(A, B) = \sum_{\{i,j\}\in E, i\in A, j\in B} w_{ij}.$$

The degree volume is

$$d(A) = \sum_{i\in A} d_i.$$

Here's an intuitive clustering criterion that combines 2 objectives: interesting similarity within the clusters, dissimilar from complement.

HNC1: NP-hardness based on reduction from partition that says it's weakly NP-hard, (cf. Cheeger) $\min_{S\subseteq V} \frac{C(S,\overline{S})}{d(S)} + \frac{C(S,\overline{S})}{d(\overline{S})}$. Sharon 2007:

$$\min_{S\subseteq V} \frac{C(S,\overline{S})}{C(S,S)}.$$

Eigenvector algorithms are intractable for large images. Algorithm is hierarchical coalescing of pixels. Is the 2nd problem the same as the first?

HNC is polytime solvable by rewriting as monotone integer programming. Each inequality: can contain 2 variables with opposite sign coefficients, z_{ij} appears in 1 inequality only with coefficient 1. In this case, the variables are all binary. It is the complexity of a cut on a certain associated graph. Have 1 node for each possible value.

(? Min ratio problem, nonmonotone submodular divided by supermodular function)

Minimize $\frac{\sum w_{ij} z_{ij}}{\sum w'_{ij} z_{ij}}$, $x_i - x_j \le z_{ij}$, $x_j - x_i \le z_{ji}$, $y_{ij} \le x_i$, $y_{ij} \le x_j$, x_j, z_{ij}, y_{ij} binary.

Minimizing $\frac{C(S,\overline{S})}{C(S,S)}$ is equivalent to minimizing $\frac{C(S,\overline{S})}{d(S)}$.

Lemma:

$$\min_{S\subseteq V} \frac{C(S,\overline{S})}{q(S)} + \frac{C(S,\overline{S})}{q(\overline{S})} = \min_{y^TQ1=0, y_i\in\{-b,1\}} \frac{y^T(D-W)y}{y^TQy} = \min_{y^TQ1=0, y_i\in\{-b,1\}} \frac{y^TLy}{y^TQy}.$$

The spectral continuous relaxation removes $y_i \in \{-b, 1\}$; the combinatorial relaxation removes $y^TQ1 = 0$.

We solve the problem for any b. You can describe the solution compactly.

$$\alpha(b) = \min_{y\in\{-b,1\}} \frac{y^T(D-W)y}{y^TQy}$$

General technique problems: find whether $\min_{x\in F} \frac{f(x)}{g(x)} < \lambda$. There is a general procedure for constructing a graph associated to a monotone integer program. Capacity of cut $S \cup \{s\}, T \cup \{t\}$ in the graph is constant plus Rayleigh ratio. Now there is a process of simplifying the graph. The cut problem can be solved for all problems. To do this I need to simplify the graph. Scale arc weights.

If you have graph with capacities depending on single parameter, source-adjacent/sink-adjacent monotone, then you can find all break points (?). Cut problem is parametric: capacities are linear in parameter on one side and independent on the other. There are at most n breakpoints.

HNC1 does better than eigenvector. (Ex. eigenvector result splits sky in half, NC' does better.) Better: Use entropy for node weights instead.

HNC used in data mining: can be used in supervised or unsupervised way. Examples: denoising spectra of nuclear detectors, ranking drug effectiveness. Neurofinder, SHA2017

We test 2 variants of SNC. Neural nets don't do well. SNC achieves best, most robust performance across data sets. Pairwise similarities are good to use. Unlike in image segmentation, anything can be neighbor of anything else. Number of pairwise similarities grows quadratically. cf. Sparsification, but we don't have the matrix. The idea is to detect where the relevant similarities are.

Approximate PCA, based on ConstantTimeSVD (Drineas, Kannan, Mahoney 2006). Subdivide dimension into k pieces.

14 A Cost Function for Similarity-Based Hierarchical Clustering (Sanjoy Dasgupta, UC San Diego)

The development of algorithms for hierarchical clustering has been hampered by a shortage of precise objective functions. To help address this situation, we introduce a simple cost function on hierarchies over a set of points, given pairwise similarities between those points. We show that this criterion behaves sensibly in canonical instances and that it admits a top-down construction procedure with a provably good approximation ratio.

We show, moreover, that this procedure lends itself naturally to an interactive setting in which the user is repeatedly shown snapshots of the hierarchy and makes corrections to these.

Hierarchical clustering is a useful tool for exploratory data analysis. It captures structure at all scales, and we have well-established algorithms like average linkage.

It's not completely clear what these algorithms are doing. Individual steps seem innocuous (merging things together) and produce a tree, but there's no objective function that it's optimizing. There's no way to compare performance, impose constraints, etc.

These algrotihms are defined procedurally, rather than in terms of optimization problems.

Input: similarity function on $X = \{x_1, \ldots, x_n\}$. Represent as undirected graph with weights w_{ij}. Idea for cost function: charge for edges that are cut.

But in hierarchical clustering, you eventually cut all the edges. Charge more the higher up an edge is cut. Multiply by number of data points remaining.

$$C(T) = \sum_{i,j} w_{ij} \# \left(\text{descendants of lowerest common ancestor of } i, j\right).$$

There's nothing related to depth here, but the algorithm will produce trees with depth $O(\ln n)$.

Sanity check:

- There is always an optimal tree that is binary.
- If the similarity graph is disconnected, the top split must cut no edges.
- Three canonical examples:
  1. Caterpillar (line) graph: separating off one by one incurs cost $n+(n-1)+\cdots+1 = \Omega(n^2)$; the balanced tree gets cost $O(n \ln n)$.
  2. Complete graph: all trees have the same cost.

     Charge the number of nodes that was left when the edge is broken. Upper bound is $n\binom{n}{2}$.

     Bound the cost of the tree by revealing it one node at a time. Let ϕ_j be upper bound on what's left after j splits uncovered. Initial upper bound is $\phi_0 = n^3$. Say jth split divides set of size m into sizes $k, m-k$. Then upper bound changes by

     $$\phi_j - \phi_{j-1} = m^3 - (k^3 + (m-k)^3) = 3mk(m-k).$$

     Cost of split is $c_j = mk(m-k) = \frac{\phi_j - \phi_{j-1}}{3}$. Total cost of tree is $c_1 + \cdots + c_{n-1} = \frac{\phi_0 - \phi_{n-1}}{3} = \frac{n^3 - n}{3}$.

3. Planted partition model: Correcting clustering in expectation.

It is NP-hard to minimize the cost function. We have to look at heuristics. Here's a natural heuristic. Do spectral bipartitioning, make that the top split, and recurse.

Sparsest cut is NP-hard but we have lots of α- approximation algorithms: spectral, LP, SDP relaxation. Then overall you get $\alpha \ln n$ approximation. Recursive spectral (or other types of) partitioning has been propsed in other contexts.

Would this be extendable to case when nodes have costs? Absolutely.

Approximation guarantee.

Lemma 14.1. *For any binary tree T on V there is $(\frac{1}{3}, \frac{2}{3})$-balanced partition A, B of V with $\frac{w(A,B)}{|A||B|} < \frac{27}{4|V|^3}C(T)$.*

Let B_i be the smaller set broken off at step i, A_i the larger. Pick smallest k with $|B_1| + \cdots |B_k| \geq \frac{n}{3}$. Let $A = A_k$, $B = \bigcup_{i=1}^k B_i$. Then $C(T) \geq \frac{2n}{3} w(A, B)$.

A more general cost function is

$$\sum_{i,j} f(w_{ij} \# \,(\text{descendants of lowerest common ancestor of } i, j)).$$

Further developments:

- Charikar-Chatziafratis: If an α-approximation to sparsest cut is used, overall approximation ratio is still just $O(\alpha)$.
- Cohen-Addad-Kanade-Mathieu: Sometimes we want recovery guarantee. Suppose similarity graph came from ultrametric (ex. phylogenetic tree): Similarities are non-increasing function of ultrametric distances. Then we can recover the ultrametric tree.

Ex. clustering with attributes. Average linkage also did well. What's the point then? We put all this effort into finding a cost function... There was a point.

1. Average linkage is quadratic, not efficient on big data.

   This is top-down, so you can look for quick-and-dirty ways of doing sparse cut in linear time.

2. Then you can do constrained hierarchical clustering.

Suppose you want to be interpretable: each split should be on a single feature.

Interactive hierarchical clustering. There are a lot of ways to cluster animals. (For similarity for animal dataset, we just used dot products.) There's no way an unsupervised algorithm can figure out what you want, without interaction.

Start an interactive procedure. On each round, pick 10 animals. Show hierarchy restricted to those 10 animals. Ask expert: does this hierarch look good? Ex. ({tiger, collie}, gorilla). Tiger and collie should be in group not containing gorilla. Minimize the same loss function with this constraint. Algorithmically this is quite easy to do.

Cost function remains the same, but now there are hard constraints. How to incorporate constraints? It's hard to incorporate into a bottom-up scheme, easier in a top-down schemes. (Another way is to learn similarity function?) Algorithm to produce tree is only approximation algorithm. What's the most efficient way to get to the best tree? Correct final result, or go back through algorithm?

There are many cases where we already have cost functions, where it makes sense to come up with another. Often we come up with one by positing a probabilistic model; cost function is just the negative log-likelihood. Assumptions can be hard to swallow. Log-likelihood cost function can be a huge mess, not something we want to deal with algorithmically, just do EM or local search. There's somthing to be gained by trying to find something similar that captures similar things that we can deal with algorithmically.

15 Simons Institute Open Lecture: "Does Computational Complexity Restrict Artificial Intelligence (AI) and Machine Learning? (Sanjeev Arora, Princeton University)

Can machines think? Philosophy and science have long explored this question. Throughout the 20th century, attempts were made to link this question to the latest discoveries—Goedel's theorem, Quantum Mechanics, undecidability, computational complexity, cryptography etc. Starting in the 1980s, a long body of work led to the conclusion that many interesting approaches—even modest ones—towards achieving AI were computationally intractable, meaning NP-hard or similar. One could interpret this body of work as a "complexity argument against AI."

But in recent years, empirical discoveries have undermined this argument, as computational tasks hitherto considered intractable turn out to be easily solvable on very large-scale instances. Deep learning is perhaps the most famous example.

This talk revisits the above-mentioned complexity argument against AI and explains why it may not be an obstacle in reality. We survey methods used in recent years to design provably efficient (polynomial-time) algorithms for a host of intractable machine learning problems under realistic assumptions on the input. Some of these can be seen as algorithms to extract semantics or meaning out of data.

"The law of clickbait says that if there is a question mark at the end, the answer is no."

Let's start with the oldest question in CS: Can machines think? Ada Lovelace formulated the term computer. "They don't do anything new, only do what we program them to."

Turing formulates the Imitation Game (Turing Test) in "Can machines think?" He includes 9 frequent objections. One, that addresses Ada's, is that you can program a computer to learn from experience. One objection is ESP.

We distinguish between:

- Weak AI: Machine's input/output behavior is human-like (as in Turing test).
- Strong AI: Machine actually has a "mind," with human-like thought processes.

There's a lot of controversy about strong AI, less so recently.

15.1 History: Controvery over strong AI

People thought it was impossible.

Simulation argument for AI: "In principle one could just simulate the human brain." (10^{40} atoms, etc.)

Science fiction writers think of questions like: You can take a core dump, would you have 2 copies?

This is impractical even many decades into the future; we don't understand well how the brain works.

We don't even have working model of C. elegans brain (connectome understood by White et al 1986; 302 neurons and 5000 synapses).

The simulation idea seems OK but is impractical.

Searle attemped to show impossibility of strong AI. He flipped the simulation argument: Any computer that passes the Turing test in Chinese is simulatable by a group of English-speaking humans. But no one in the room understands Chinese.

To me this is fallacious (it's just saying a computer is made of gates). But in the non-major class I was teaching, students found it convincing.

Message passing among large groups can be quite powerful, like the "intelligence" of ant colony, social hierarchies, agriculture, war, etc.

The "room" would contain 10^{10} and would be the size of Connecticut.

There is accumulating evidence that consciousness is an emergent phenomenon.

There is the claimed impossibility of strong AI via Goedel's Theorem (Lucas 1960's, Penrose 1980's). Suppose you start off by thinking: I don't believe in AI. How do I refute it? There aren't so many mathematical ways of refuting something vague and not well-defined. Latch on whatever has happened in math in recent decades—Goedel's Theorem. The construction of Goedel's sentences, humans think it's true, but it's not provable in, say, Peano arithmetic. The computer can only recognize provable things, so cannot recognize its truth. Humans have intelligence that can't be implemented in computers. Quantum gravity in the brain plays an important role? The book caused a firestorm. You can spend a few evenings spending the back and forth on this.

At best this argument says that the computer can make mistakes which a well-trained mathematician might not. It's not clear what that refutes.

15.2 History: computational complexity (P vs. NP)

Computational complexity was invented to prove limitations on computers, the next step beyond Turing and Goedel's undecideability. Goal: characterize computation time needed to solve a problem.

We have no way to prove that the computer takes $\gg n \ln n$ time to solve any reasonable problem. We prove hardness modulo established conjectures.

P is the class of problems where a solution can be found in poly time; NP is the class of problems with solutions which can be checked in polynomial time. Ex. you can't search through a very large tree in poly time.

NP complete means every NP problem is reducible to instance of this in n^ε time. They are the hardest problems of NP: if they have a solution, you can solve any other NP problem.

P=NP is often phrased as "Can brilliance/creativity be automated?" It takes brilliance to come up with a solution, no brilliance to check it. So P=NP means if you can check it, you can come up with a solution.

If P≠NP then NP-complete problems cannot be solved in polytime.

Complexity obstacle to AI: Very simple learning problems, even involving fairly low-level perception, turn out to be NP-complete. (Kirousis, Papadimitriou 85).

It is important to realize that NP-hardness only concerns difficulty of worst instance, so only relevant if we desire an algorithm that works for every input. Cf. Goedel's Theorem.

Work in last couple of decades suggests that computing even fairly weak approximations is also NP-complete for many problems (uses PCP theorems, Unique Games Conjecture, etc.).

Suppose you have an algorithm that gets within 10% of all traveling salesman problems (TSP). Then I can leverage it so solve any NP problem. Convert boolean satisfiability ot TSP problem. Use approximation algorithm to solve satisfiability exactly.

If you believe that boolean satisfiability is hard, then 10% approximation for TSP is hard. The approximation ratio that is hard is the boundary of what known algorithms can do, so current algorithms are the best algorithms.

This was a punch in the gut: I was hoping that some of these problems were easy.

15.3 Survey of machine learning approaches and why computational complexity seems to present a hurdle

This has always been our line of thought, not the philosophical problems.

ML went through phases, no phases really go away.

Logic-based approach rose out of logic work. "Programs with common sense," McCarthy 1959. It sketched the logic-based approach: have a database of facts and system for logically deducing new facts from these. People add commonsense axioms to a database. The hope is that once it has enough, it is complete enough, and becomes intelligent. This continues. Many programming environments were developed for symbolically expressing and solveing problems.

Practical difficulty: tedious to code millions of facts.

Deduction in even simple logic systems is NP-complete.

A Bayesian net (Pearl 88) is a probabilistic graphical model of data. Roughly speaking, this is the probabilistic analogue of logical reasoning.

Deduction: compute probablity of certain variable being 1 conditioned on the known values of some other variables. This problem is NP-hard.

This seems a relevant way of thinking about the world. Progress was made with heuristic algorithms.

Kernel SVMs are an example of learning from data. Given examples of birds and animals who are not birds, give classifier separating birds and not birds. Algorithm runs in polynomial time via convex programming.

It was a reaction to NP-completeness. We will have provable algorithms in polytime! Map to feature space and come up with classifier on that space. In general you can allow it to make error; there is a nice statistical learning theory that says when you can do this. This works well on simple problems, not so much on complex problems.

Anyone who's read any newspaper in the last 5 years know that deep learning is the current approach. It was originally inspired by neurons in the brain.

You have simple computational units (the popular one is the ReLU gate), $RELU(\sum_i w_i x_i + \theta)$. Parameters are trained using backpropagation. That classical algorithm applied on large datasets and modern computers works well. There's been an explosion of work. Neural nets get better-than-human (97%) accuracy on image classification using 100+ layers and millions of edges. (Pictures collected from Internet with labels. It's better than any individual human. Aggregated over the wisdom of crowds you get better accuracy.)

Complexity theory says that learning even a shallow net with 1 hidden layer is intractable, Klivans-Sherstov 10.

Computational complexity's emerging lesson: for most natural problems, the natural phrasing is computationally intractable (NP-hard). This need not mean the problem is intractable in practice. It may even be solvable in industrial scales.

Is NP-hardness an obstacle for understanding reality?

Suppose we want to understand a text corpus. This falls under the problem of "topic modeling." When people prove this is NP-hard, that means there are NP-hard instances. But this doesn't say anything about the original problems. You can define a smaller bounding box containing the original problem that is polytime.

The bounding box is a human construct, depending on your advisor, your coffee that morning, etc.

I want to talk about how to draw a smaller bounding box.

15.4 Efforts to understand one particular neural net method

People solve language processing tasks with recurrent deep nets. Ex. Google is rolling out much improved translation.

How does a deep net understand language? "Understanding" is captured in parameters of deep net. How do you train them? Train to correctly predict next word given previous k words. This is a black-box algorithm.

If I give you the first half of the sentence and ask you to predict the second half, you need to understand some grammar, common sense, etc.

A simple neural net for language understanding is $\mathbb{P}(w_t) = f_\theta(w_{t-1}, \dots, w_{t-k})$,

$$\mathbb{P}[w|w_1, \dots, w_5] \propto \exp\left[v_w \cdot \left(\frac{1}{5}\sum_i v_{w_i}\right)\right].$$

NN gets positive feedback if correct, negative if incorrect.

Word vectors can be used to solve analogies like man:woman::king:queen by completing the parallelogram.

Why does our algorithm find such vectors? What property of language does this reflect?

You can do theory with it. It's possible to create representations of sentence/paragraphs, e.g. skip-thought (Kiros 16). The tiger rules this jungle, a lion hunts in a forest—similar sentences even though they don't share words.

One popular question about deep nets: optimization problem is nonconvex, unclear why local improvement (backprop) finds good solutions.

More important question (IMO): What is the structure of data that deep nets are capturing and can they be captured by simpler algorithms?

We have a generative model for language. Words are things in your head which associate with vectors, $\mathbb{P}(w|c_t) \propto \exp(v_w \cdot c_t)$.

1. Model validates many things in NLP, like PMI, $PMI(w, w') = v_w \cdot v_{w'}/d \pm O(\varepsilon)$.
2. word2vec and GloVe methods are approximate inference for our model.
3. Relations correspond to lines.
4. With slight modificiation, yield simple and understadndable sentence embeddings competitive with NN.

15.5 Why computational complexity may not be a hurdle going forward

Proposed agenda for theoretical ML:

- look beyond NP-hardness. Look beyond usual convex formulations. (matrix factorization, tensor decompositions, LVM)
- Beware "minimum description length" aesthetic; likely leads to hard problems. Fewest assumptions does not give best theorem.
- Combine math and modeling.

AI is elephant in the room. There are many communities looking at AI, we are like blind men with our own takes. I'm sure theory and math will play a big role in this endeavor.

16 Embedding as a tool for algorithm design (Le Song, Georgia Institute of Technology)

Many big data analytics problems are intrinsically complex and hard, making the design of effective and scalable algorithms very challenging. Domain experts need to perform extensive research, and experiment with many trial-and-errors, in order to craft approximation or heuristic schemes that meet the dual goals of effectiveness and scalability. Very often, restricted assumptions about the data, which are likely to be violated in real world, are made in order for the algorithms to work and obtain performance guarantees. Furthermore, previous algorithm design paradigms seldom systematically exploit a common trait of real-world problems: instances of the same type of problem are solved repeatedly on a regular basis, differing only in their data. Is there a better way to design effective and scalable algorithms for big data analytics?

I will introduce a framework for addressing this challenge based on the idea of embedding algorithm steps into nonlinear spaces, and learn these embedded algorithms from problem instances via either direct supervision or reinforcement learning. In contrast to traditional algorithm design where every steps in an algorithm is prescribed by experts, the embedding design will delegate some difficult algorithm choices to nonlinear learning models so as to avoid either large memory requirement, restricted assumptions on the data, or limited design space exploration. I will illustrate the benefit of this new design framework using large scale real world data, including a materials discovery problem, a recommendation problem over dynamic information networks, and a problem of learning combinatorial algorithms over graphs. The learned algorithms can reduce memory usage and runtime by orders of magnitude, and sometimes result in drastic improvement in predictive performance.

I think of algorithm design as more than just algorithms, model plus algorithms. Sometimes when you appropriately design the model, the algorithm is better.

Categorize ML algorithms in 2 classes:

1. function approximations (neural nets, kernel methods; use optimization to estimate parameters,

2. graphical model (model more detailed information between variables, parameterize with potential, learn with graphical model inference procedures).

Function approximation is more scalable. Graphical models are less scalable.

We want to use lots of prior knowledge (reduce amount of data we need) and still be scalable. We want to combine the two approaches.

Try to learn the inference algorithm directly.

Steps: Identify structure, define graphical model, embed inference algorithm, link embedding to target.

Examples:

1. Prediction for structured data: is drug/materials effective or ineffective? Is information cascade (in social media) viral/non-viral? Is natural language sentence positive/negative? Are code graphs benign or malicious?

   Design kernels for structures and work from that point. Use bag of structures. Have a dictionary of small structures like trees of depth 1, 2.

   You need 1 billion parameters to get state-of-the-art performance, you need to get to level 3, 4, 5, 6.

   You can do something smarter to extract feature. One is based on graph isomorphism trick.

   Compress your representation as you go. First hash the type into fixed dimension vector. (Weisfeiler-Lehman). Has for each node, combine with hashes of neighbors.

After each node has a fixed vector; aggregate. You can fix a dimension for the entire structure. Hash is manually designed. You still need 100 million parameters.

Embedding reduces model size by 1000 times.

2. Dynamic processes over networks. People buying items is not static. You have the ratings and timestamp information. Typically when you model this, you use matrix factorization and discard temporal information. Each row of U is one user, each column of V is one item, UV makes the prediction.

   A simple algorithm is alternating least squares. Solve optimization for u_i by aggregating information from interacting items v_j the user has interacted with, and vice versa.

   With embedding, reduce prediction error for return time by threefold.

3. Combinatorial optimization over graphs.

   Social networks are nonstatic. Look at minimum vertex cover which has 2-approximation algorithm. Pick a small number of nodes such that for each edge at least one node has been selected. Look at degree information. Look at edges and sum degrees at 2 ends. Select edges with highest total degree. This is a manually designed rule. With embedding we can learn something way better, with approximation ratio close to 1.

   Train using a network generator. Update according to performance. This learns algorithm parameters for particular distribution of graphs.

Key observation and fundamental question: Algorithm is structured composition of manually designed operations. How to embed this in a space?

Thinking in graphical model terms is good idea.

Ex. for a chemical molecule, nodes are atoms, edges are bonds. Introduce a latent variable model with latent variable for each node.

Describe using pairwise Markov random field. Joint likelihood

$$\mathbb{P}(H|X) \propto \prod_{i\in V} \psi_v(H_i, X_i|\theta_v) \prod_{ij\in E} \psi_e(H_i, H_j|\theta_e).$$

How to aggregate info across graph? Getting posterior is aggregating information across graph structure. If graph is not a tree, it's a difficult problem. Compute q_i in iterative fashion. Initialize each node with posterior distribution, at each step aggregate information. Take node potential and edge potential to neighbors, and aggregate. Dependency structure:

$$\mathcal{T}(X_i, \{q_j(H_j)\}_{j\in N(i)}).$$

You propagate information according to the graph structure.

Chicken-and-egg problem: To learn parameter we have to run the algorithm.

Reparametrize updates. We vector space embedding for representing distributions. Map density to vector space by $\mathbb{E}_q[\phi(H)]$. If you are able to pick flexible, rich, nonlinear mapping, this is good. This is a nonparametric way of describing density. If you have operations which apply to the original, you can find another operation which does the same on the embedding.

$$\mathcal{T}(q(H)) = \widetilde{\mathcal{T}}(\mu_H).$$

Ex. $\phi(H) = (H, H^2, H^3, \ldots)^T$.

Combine embedding ideas with graphical models. Key operation is sum-product rule (with Bayes rule). We have spectral HMM, kernel belief propagaion, etc.

Let's embed mean-field and other inference algorithms. The algorithm is: initialize μ_i for all i, and iterate T times

$$\mu_i \leftarrow \widetilde{\mathcal{T}}(X_i, \{\mu_j\}_{j\in N(i)})$$

Aggregate according to nonlinear mapping $\widetilde{\mathcal{T}}$—parametrize it and learn it.

Directly parametrize nonlinear mapping:

$$\mu_i \leftarrow \sigma\left(W_1 X_i + W_2 \sum_{j\in N(i)} \alpha_i(\mu_j)\mu_j\right).$$

W_1, W_2 will be learned.

Paradigm shift: An embedded algorithm is structured composition of nonlinear operations/functions.

A graphical model inference algorithm is an instance of an embedded inference algorithm, which is a subset of generic function approximator (which doesn't consider structure).

Benefit of thinking in graphical model sense is that there's many different ways of aggregating information. Parametrize messages. Have 2 densities associated with each edge. Initialize messages and update it.

Ex. embed belief propagation: Initialize μ_{ij}, and iterate T times: $\mu_{ij} \leftarrow \widetilde{\mathcal{T}}(X_i, \{\mu_{li}\}_{l\in N(i)\setminus j})$ for all i, j. Aggregate $\mu_i = \widetilde{\mathcal{F}}(\{\mu_{li}\}_{l\in N(i)})$ for all i. Learn using supervised learning, generative models, RL, etc.

New tools for algorithm design: graphical models (incorporate prior info, reason about structure inference algorithm), function approxiamtion (representation ability, statistcal complexity, generalization ability).

Scenarios:

1. Prediction for structured data: Learn efficiency (PCE) for molecular structure.

   Embed mean field and belief propagation. Run regression,

   $$V^T\left(\sum_{j\in V_i} \alpha_j(\mu_j)\mu_j\right) = f\,(\text{molecule}).$$

   Weisfeiler-Lehman level 6 needs 1000 million parameters, this needs 0.1 million parameters to get same performance.

2. Dynamic process over networks. LVM evolving over time. Create a new node each interaction. Graph structure grows. Map to latent variable model.

   Every interaction, make updates using previous features. This corresponds to filtering algorithm (predict in future). Have unrolled interaction graph, embed filtering. Ex. predict time of interaction.

What is learned by these embedding algorithms (when they do better than matrix factorization)? Consider the temporal knowledge graph completion problem. New events add to the graph.

Example: "Enemy's friend is an enemy." Cairo threatens Manchester, Manchester provides aid to Croatia. Predict that Cairo predicts Croatia.

"Friends' friend is a friend, common enemy strengthens the tie." Columbia reduces relation with Belgium, Belgium fights Ottawa, Columbias has trade coop with New Delhi, New Delhi has diplomatic coop with Ottawa. Predict Colombia has material coop with Ottawa.

3. Combinatorial optimization over graph.

   Reward is -1 for each node picked. State is current selected nodes. Use RL (Q-learning) to pick nodes.

   Run CPLEX, use as proxy for global optimum. It produces results one by one. Produce something almost as good, but that runs much faster.

   Learned algorithm balances between degree of picked node and fragmentation of the graph. In the end the algorithm has potential to pick a node that is adjacent to many remaining nodes.

4. Program learning: See digits, recognize and perform addition, carry, repeat. There's only weak supervision.

17 Robust Estimation of Mean and Covariance (Anup Rao, Georgia Institute of Technology)

We consider the fundamental statistical problem of robustly estimating the mean and covariance of a distribution from i.i.d. samples in n-dimensional space, i.e. in the presence of arbitrary (malicious) noise, with no assumption on the noise. Classical solutions (Tukey point, geometric median) are either intractable to compute efficiently or produce estimates whose error scales as a power of the dimension. In this talk, we present efficient and practical algorithms to compute estimates for the mean, covariance and operator norm of the covariance matrix, for a wide class of distributions. We show that the estimate of the algorithm is higher than the information-theoretic lower bound by a factor of at most the square-root of the logarithm of the dimension. This gives polynomial-time solutions to some basic questions studied in robust statistics. One of the applications is agnostic SVD.

Suppose we are given samples from an unknown distribution in come class—ex. 1-D Gaussian. We can accurately estimate mean and variance. Fisher: MLE are optimal in a certain sense. Tukey: what about errors in the model? This gave rise to robust statistics.

How do estimators behave in a neighborhood around the model?

Suppose we have gaussian samples with noise. We need to constrain noise. Let an adversary add η fraction of points. (Huber)

1. iid samples $x_1, \ldots, x_{(1-\eta)m} \sim D$.

2. adversary adds ηm points after seeing the samples.

We want to minimize $\|\widehat{\mu}-\mu\|_2$ and $\left\|\widehat{\Sigma}-\Sigma\right\|_2$.

PCA: $2k$ points can completely destroy top k components.

Does empirical mean and variance work? No. A single corrupted sample can arbitrarily corrupt the estimates. But the median and median absolute deviation do work. This is optimal.

$$MAD = \text{median}[|X_i - \text{ median}(X)|].$$

Several approaches have been suggested in high dimensions, ex. generalization of median. All known estimators are hard to compute or lose poly factors in the dimension.

- Tukey median: error guarantee, but NP-hard
- Geometric median: bad error guarantee, good running time.

Results (FOCS 2016). We need assumptions on the distribution, bounded 4th moments:

$$\mathop{\mathbb{E}}_{D}((x-\mu)^T v)^4 \le C_4(\mathop{\mathbb{E}}_{D}((x-\mu)^T v)^2)^2.$$

(Ex. subgaussian, subexponential).

Theorem 17.1. *1. (Known covariance) With* $m = O\left(\frac{(n\ln n + n\ln\left(\frac{1}{\eta}\right))\ln n}{\eta^2}\right)$ *samples , can find* $\|\widehat{\mu}-\mu\|_2 = O(\eta^{\frac{3}{4}}\sigma\sqrt{\ln n})$*, lower bound* $\Omega(\eta^{\frac{3}{4}}\sigma)$.

2. (Unknown covariance) With $m = O\left(\frac{(n\ln n + n\ln\left(\frac{1}{\eta}\right))\ln n}{\eta^{4/3}}\right)$ *samples , can find* $\|\widehat{\mu}-\mu\|_2 = O(\eta^{\frac{1}{2}}\sigma\sqrt{\ln n})$.

For gaussian we can improve $\frac{3}{4}$ to 1.

For covariance estimation, apply mean estimation to xx^T. Since we're applying to 2nd order, we need bounded 8th moments and 8-wise independence.

For the rest of the talk, I assume gaussian with known covariance. For bounded 4th moments, even median doesn't work. Fix: find the smallest interval that contains $1-\eta$ fraction of the points and compute sample mean. This is the main switch one has to make to make this work for general (nongaussian) distributions.

Multiple extensions of median: coordinate-wise median, geometric median $\text{argmin}_y \sum_i \|x_i - y\|_2$. Both have $\|\widehat{\mu}-\mu\| = \Omega(\eta\sigma\sqrt{n})$ error in worst case. (Ex. for coordinate-wise, sum errors from coordinates.)

Bad case for medians is to have η fraction concentrated far away.

Let's consider easy case in high-dimensions. Suppose we are given mean shift direction, the direction from true mean to sample mean. Then this is an easy problem; reduce to 1-D problem. Project all points onto $n-1$ dimension perpendicular space. There is no mean shift in this space; compute sample mean as estimator. Remaining is 1-D problem.

It's sufficient to know direction of mean shift vector. We can't take the top principal component.

Algorithm alternates between two steps.

1. Weak outlier-removal step
   (a) If dimension is 1, return median.
   (b) Remove all points in S at distance $> C\sigma\sqrt{n}$ from coordinatewise median.
2. Project to top $\frac{n}{2}$ principal components and recurse.
   (a) Identify good subspace (high-dimensional subspace where coincide with true mean). Let V be span of top $n/2$ eigenvectors of covar matrix, W span of bottom.
   (b) Recursively compute mean of $\text{Proj}_V(S)$.
   (c) Append mean of W, $mean(\text{Proj}_W(S))$.

Projection of mean shift vector onto bottom $n/2$ space is small.

Covariance matrix of $S = \Sigma_S$ is

$$\Sigma_S = (1-\eta)\sigma^2 I + \underbrace{\eta\Sigma_N + \frac{1-\eta}{\eta}\delta_{\text{shift}}\delta_{\text{shift}}^T}_{A}.$$

Outlier removal ensures noise can increase $n/2$'th eigenvalue by small amount. From outlier removal, $\text{Tr}(A) = O(\eta\sigma^2 n)$. Weyl's inequality from robust statistics says

$$\lambda_{\frac{n}{2}}(\Sigma_S) \le (1-\eta)\sigma^2 + \lambda_{n/2}(A) \le (1+O(\eta))\sigma^2.$$

$\text{Tr}(\Sigma_N) = O(\sigma^2 n)$, so $\text{Tr}(\eta\Sigma_N) = O(\eta\sigma^2 n)$.

Second step: once we have control of $n/2$th eigenvalue, we know projection onto bottom $n/2$ eigenspace is small. Rayleigh quotient is small because we have upper bound on eigenvalue.

$$\|P_W\delta_{\text{shift}}\|^2 = O(\eta^2\sigma^2).$$

Error independent of dimension. Get $\sqrt{\ln n}$ because we recurse $\ln n$ times.

$$\|\widehat{\mu} - \mu\|^2 = O(\eta^2\sigma^2 \ln n)$$

It's a simple algorithm to code.

For covariance estimation, we need to compute huge matrices d^2, and covariances of those $d^2 \times d^2$. It would be very good to get rid of this!

There is a case where we can get rid of this. If the problem were to estimate the 2-norm of the covariance, we don't need to construct the $d^2 \times d^2$ matrix. Compute sample covar matrix and top principal component. We can estimate variance in this direction. If they are close, we are done. If they differ by lot, threshold by 2 hyperplanes and throw out outliers. This converges in $O(n)$ iterations. This constructs only $d \times d$ covariance matrices.

Summary: Algorithm for robust estimation of mean and covariance, works for wide class of distributions, needs only linear number of samples for mean estimation, simple and practical.

Open:

- algorithm matching lower bounds for general distribution?
- better time complexity for covariance estimation in operator norm?
- robust estimation in other problems, like Gaussian mixture models?

Algorithm also computes the Tukey median approximately.

18 Computational Efficiency and Robust Statistics (Ilias Diakonikolas, University of Southern California)

We consider the following basic problem: Given corrupted samples from a high-dimensional Gaussian, can we efficiently learn its parameters? This is the prototypical question in robust statistics, a field that took shape in the 1960's with the pioneering works of Tukey and Huber. Unfortunately, all known robust estimators are hard to compute in high dimensions. This prompts the following question: Can we reconcile robustness and computational efficiency in high-dimensional learning?

In this talk, I will describe the first efficient algorithms for robustly learning a high-dimensional Gaussian that are able to tolerate a constant fraction of corruptions. More broadly, I will present a set of algorithmic techniques that yield efficient robust estimators for high-dimensional models under fairly general conditions.

(Joint work with Kamath, Kane, Li, Moitra, Stewart)

Can we design algorithms that are provably robust and computationally efficient?

MLE is the wrong thing to do here.

Robust statistics has developed independent of other areas.

Robust 1-D Gaussian learning: Find a 1-D estimate with TV distance between true and estimated $\leq \varepsilon$.

What happens in higher dimensions?

Suppose we have some family of high-dimensional distribution. Get n samples from $F \in \mathcal{F}$, adversary inspects samples and changes arbitrarily ε-fraction of them. Difference from Huber's model: Adversary can both add and subtract samples.

Two special cases are unknown mean and covariance.

Computationally efficient estimators can only handle $\varepsilon \leq \frac{1}{\sqrt{d}}$.

There is algorithm: given $\varepsilon > 0$ and ε-corrupted set of N samples from d-dimensional Gaussian $N(\mu, \Sigma)$, the algorithm runs in poly$(N, d, \frac{1}{\varepsilon})$ time and finds $\widehat{\mu}, \Sigma$,

$$d_{TV}(N(\mu,\Sigma), N(\widehat{\mu},\widehat{\Sigma})) = O(\varepsilon \ln\left(\frac{1}{\varepsilon}\right)) + \widetilde{O}\left(\frac{d}{\sqrt{N}}\right).$$

Corollary: for $N = \widetilde{O}\left(\frac{d^2}{\varepsilon^2}\right)$, get poly $\left(\frac{d}{\varepsilon}\right)$ time with TV distance $O(\varepsilon \ln\left(\frac{1}{\varepsilon}\right))$.

General recipe:

1. Find appropriate parameter distance.

   Basic fact: $d_T V(N(0,\mu), N(0,\widehat{\mu})) \leq \frac{\|\mu-\widehat{\mu}\|_2}{2}$.

   Corollary: suffices to output $\widehat{\mu}$, $\|\mu-\widehat{\mu}\|_2 = \widetilde{O}(\varepsilon)$.

2. Detect when naive estimator has been compromised.

   In high-dimensions, being an outlier is a global, not local property.

   We need a certificate for why the true mean is far from the empirical mean.

   Lemma: given N ε-corrupted samples from $N(\mu, I)$ and $N \geq 10(d + \ln\left(\frac{1}{\delta}\right))/\varepsilon^2$ then for $\widehat{\mu}, \widehat{\Sigma}$, with probability $\geq 1-\delta$,

   $$\|\mu-\widehat{\mu}\|_2 \geq C\varepsilon\sqrt{\ln\left(\frac{1}{\varepsilon}\right)},$$

   $\left\|\widehat{\Sigma} - E\right\|_2 \geq C'\varepsilon \ln\left(\frac{1}{\varepsilon}\right)$.

   Adversary needs to mess up the 2nd moments to mess up the 1st moments.

   Use contrapositive: calculate empirical mean, find approximately largest eigenvalue. Outliers have special behavior in direction of largest eigenvector.

   If you project empirical data in this direction (1-D projection), you get outliers. We will find a threshold and remove all points above it.

3. Find good paramters, or make progress. Filtering: efficient outlier detection and removal.

   Filtering: Suppose $\left\|\widehat{\Sigma} - I\right\|_2 \geq C'\varepsilon \ln\left(\frac{1}{\varepsilon}\right)$. Project in direction of v, find T s.t. $\mathbb{P}_{x\sim_u S}[|v \cdot x - \text{median}(v\cdot x)| > T] \geq 3e^{-T^2/2}$. Throw away all points such that $|v\cdot x - \text{median}(v \cdot x)| > T$. Iterate on new dataset.

   A significant fraction of bad points cannot satisfy inequality with $\leq$. Then by definition, the variance in that direction is small.

   Fraction of bad points above threshold is more than 2 times the fractions of good points. Every iteration remove at least ε/d bad points. We will need at most d iterations.

I have removed a fraction ε fraction of good points. If I remove ε, I cannot change the empirical mean by too much.

Running time is $\widetilde{O}(Nd^2)$.

Generality of robust mean estimation. Focus of initial version was on specific distribution families (Gaussian, discrete product distributions). Algorithm works under weaker concentration assumptions with appropriate guarantees. We have concentration implied by moment assumptions. Under 2nd moment assumptions, get $O(\sqrt{\varepsilon})$. Under 4th moment assumption, get $O(\varepsilon^{\frac{3}{4}})$. Under subgaussian, $O(\varepsilon\sqrt{\ln\left(\frac{1}{\varepsilon}\right)})$. Sample complexity is near-optimal for all these cases.

Covariance is significantly more technical. Recall highest eigenvalue tells us something about empirical first moments. Here, look at empirical fourth moments.

Using the general recipe:

1. Mahalanobis distance.
2. Let $X_i \sim N(0, \Sigma)$. Fourth moment tensor of X_i id determined by Σ, $M = \mathbb{E}[(X_i \otimes X_i)(X_i \otimes X_i)^T]$, $M = 2\Sigma^{\otimes 2} + (\Sigma^{\phi})(\Sigma^{\phi})^T$.

   Transform data and look for large eigenvalues. $Y_i := (\widehat{\Sigma})^{-\frac{1}{2}} X_i$. If $\widehat{\Sigma}$ were true covariance, $Y_i \sim N(0, I)$ for inliers, in which case $\frac{1}{N}\sum_{i=1}^{N}(Y_i \otimes Y_i)(Y_i \otimes Y_i)^T - 2I - (I')(I')^T$ would have small eigenvalues.

   (Instead of using linear form, use an appropriate degree 2 polynomial.)

   An adversary needs to mess up 4th moments to tamper with 2nd moments.

Assemble algorithm: Use doubling trick $X_i - X_i' \sim N(0, 2\Sigma)$.
Use restricted problems to detect outliers.
All previous estimators have performance that deteriorates as dimension. New algorithms have dimension-independent errors.
Subsequent work/future directions:

- Optimal guarantees for Gaussian? Can we remove the logs? Yes for additive model (Huber's model), no (allowing adding and subtracting) in general; there is a statistical query barrier.
- More general prob models/settings?

19 System and Algorithm Co-Design, Theory and Practice, for Distributed Machine Learning (Eric Xing, Carnegie Mellon University)

I'll focus on presenting open problems we face in designing efficient depolyable ML algorithms.

Outside people see machine learning as a black box where you throw in data and get results out. We are viewing computers the same way. In many theorems you don't see coefficients relating to the machine's behavior.

Our startup aims to give ML solutions to big data users. The kind of solutions we have now is hand-crafted, one-time solution. It's a big barrier making ML not so popular in industry.

I developed a good algorithm for social network embedding, supports 1 million users in 6 minutes. But then we wanted to bring it to facebook. We want to run code on 100 million users. If using 1000 Hadoop machines, should support 100 million users in 0.6 minutes. But it doesn't finish in more than a week!

The system is not ideal. How to ground solutions on real computational infrastructure?

Setup

$$\operatorname{argmax}_\theta \mathcal{L}(\{x_i, y_i\}_{i=1}^{N}; \theta) + \Omega(\theta).$$

This is solved by an iterative step. This is under severe challenge: number of parameters is huge, dataset is large. Doing this for an indefinite number of iterations is challenging. You need to solve on many machines. How to get them to behave as a single machine is hard.

How do we parallelize a program? One way to make it equivalent to a sequential program is synchronization. This assumes zero-cost sync, fault recovery, but this is in practice an expensive step.

Either do synchronization and pay huge cost in latency and communication overhead, or break equivalence.

This brings us to a renewed effort to analyze efficiency in a practical algorithm in the field. Moder deep learning algorithm forces you to move to multi-core machines. Throughput grows linearly with number of machines. But number of iterations it take to converge increases: if you do a lousy job in synchronization, quality of each iteration degrades.

To achieve this it is not enough to design a smarter algorithm or reengineer the system. By knowing details in systems, the algorithms can be redesigned to be optimized. This contrasts with traditional algorithmic and systems approach, which is transaction-centric, guaranteed by atomic correctness at each step.

Now people create a specialized tool for one algorithm; we want generic solutions.

1. How to distribute?

   When you divide data and model to different machines, tools to make sure things are not going wrong are different.

   First: do load-balancing. Ex. do large regression problem in 1 billion dimensions. (There are regularizations so this is practical.) How to solve using a cluster?

   Divide big vector into chunks to put on different machines. This is not trainable, between different dimensions there is possible correlation. We need to check $x_1x_2^T$ first.

   Hope that a random partition helps. But this is a weak guarantee: in practice you are likely to run into strong coupling problems in high dimensions.

   System co-design: dynamically schedule task by examining dependencies among dimensions.

   Correlations between dimensions is itself a big problem. How to heuristically or systematically solve that?

   In high-dimensional regression, people observe that not all dimensions are equally important; a small fraction of coefficients are changing significantly. Focus on scheduling those first.

   There have been heuristic and formal ideas to do priority scheduling and block scheduling, detect coupling and bring them on the same machine.

   This is structure-aware parallelization. The system part is to make this automatic.

   Scheduling makes convergence faster. This type of speedup enjoys a convergence guarantee with rate proportional to number of machines. This requires dividing data nicely.

2. How to bridge computation and communication?

   You still need to communicate, even if not so often.

   In data-parallel problem, run proximal gradient algorithm. Should I wait for every machine to give info or allow some asynchrony?

   People use bulk synchronous parallel bridging model: computation and communcation occur during disjoint times. Bridging is difficult and expensively especially when iterating.

   How to avoid it?

   Hogwild! Idea: folklore is right: ignore communication: let all machines do their job, let them come back asynchronously. Most high-dimensional problems are sparse; same dimensions not touched on different cores, don't conflict. But it doesn't work so well.

   Parallelism vs. distribution: latency is larger in distributed setting.

   Quality of solution depend on mean and variance of latency.

   Amount of asynchrony each machine can allow: stale synchronous p. bridging model.

   Empirically the algorithm doesn't converge.

3. What to communicate?

   Trade off computing and communication.

   This is a strange question. We communicate the intermediate results, right?

   Many models are matrix models. Matrices are often big, ex. have many billions of parameters. It will choke your ethernet network.

   The full update is often not used in a distributed environment. Is this the only thing we can do?

   The form of the update often has low-rank structure, especially when you are squeezing down the number of samples. Outer product of data-dependent and parameter dependent vector.

   What if we communicate pre-updates instead of updates? This is the idea of "sufficient vectors."

   There is difficulty in establishing equivalence. Full update follows master-slave architecture, easily distributed and aggregated.

   But here, aggregating loses the low-rank structure. You have to transmit updates one at a time.

   But often every update is using small amount of data. Stochastic gradient algorithm can benefit from this.

   Storage is elastic.

   Across different models SFB (sufficient factor broadcast) is faster.

4. How to communicate?

> ML people are used to the server-worker (master-slave) architecture: Hadoop, Spark. A centralized place aggregates messages. Unbounded slaves do partial work. Communicate work to be done. Workers need latest possible version of parameters.
>
> Instead, only send updates, but they are as big as full updates if you don't do reduction. Switch away from master-slave to P2P architecture. Pre-updates cannot be aggregated before they are communicated.
>
> Tricks reduce from quadratic to linear complexity in terms of size of network, constant factor of neighborhood of broadcast. Partial broadcast: not to pass a message only one time. Iteratively keep sending to neighbors. There is effect of information relay even if you just broadcast information partially.
>
> There is strong improvement over popular frameworks. We scale well to the size of the cluster; there is a close-to-linear power.
>
> We can establish consistency results. You converge to the solution that is no different from if you sent out the full matrix to each node.

Interesting behavior: if you look at deep neural net, size of parameters is not uniform. Treat them differently: hybrid communication.

We provide a new low-level communication layer for models built in Caffe and Tensorflow. We do much better than their communication protocol.

Measure curves over convergence time, not throughput.

The goal of the CMU lab and my startup Petuum is to take complicated ML stack and make it more standardized.

What is the bottleneck at the end? It's everything. First is the computation. Storing in disk IO is inefficient. Once you are in distributed environment, communication and load balancing is bottleneck. Fault tolerance and security is another bottleneck. Some people resort to hardware innovation. Multicore GPU are hot in research communities or rich companies with deep pockets.

We don't rewrite tensorflow code. They have 2-tier structure. You see operators, program-state graphs. Another tier you don't see is how they implement their operators. That's highly dependent. We rewrote some of that part, replace with better distributed protocol.

Is this portable to different clusters? Typically as long as your cluster has NPI, standard distribution environment, our solution will work. Modularize solution stack. There is a layer you can take out and focus on.

20 The Polytope Learning Problem (Eric Xing, Carnegie Mellon University)

I will talk about the polytope learning problem: Given random samples from a polytope in the n-dimensional space with poly(n) vertices, can we efficiently find an approximation to the polytope? (Information theoretically, poly(n) many samples suffice.) I will discuss some approaches and related conjectures for this problem, and also its relation to other problems in learning theory.

More general problem: learning convex bodies. Let $K \in \mathbb{R}^d$ be unknown convex body. Input is uniformly random samples $x_1, x_2, \ldots \in K$. Problem is to reconstruct K.

Output is a set K' that is close to K, $|U_K - U_{K'}|_1 \leq \varepsilon$. K' is given by a poly(d) time oracle (improper learning).

How many samples/time is needed?

PAC-learning of convex bodies is hard because VC-dimension of convex bodies is unbounded.

We are only given positive samples.

In constant dimensions it can be done efficiently using tight-fitting polytopes, required time scales as $\left(\frac{1}{\varepsilon}\right)^{\Omega(d)}$. High dimensions have too much room.

Interesting things can happen in high-dimensions with randomized algorithms With membership queries, volume computation in poly(d) time, Dyer-Frieze-Kannan 1989.

How about learning convex bodies with membership oracle?

$2^{\Omega\left(\sqrt{\frac{d}{\varepsilon}}\right)}$ samples are needed to learn convex bodies in $\mathbb{R}^d$ from random samples and membership queries.

Idea: Construct a large family of convex bodies in $\mathbb{R}^d$ such that any two bodies are far from each other. Polynomially many random samples and membership queries from any such body insufficient to determine which one it is.

Hard family construction of KOS08: take a sphere and take intersections of exponentially many half-spaces tangent to it.

Another lower-bound construction (GR09). Start with cross-polytope in $\mathbb{R}^d$, conv($\{\pm e_i\}$). This has 2^d facets and $2d$ vertices. To each facet attach pyramid of fixed height. TO know whether a pyramid is attached to facet, we must see point in that pyramid. But total volume of pyramid is $O\left(\frac{1}{d}\right)$ of whole body.

Easy fix: take products of $\sqrt{d}$ cross-polytopes in $\mathbb{R}^{\sqrt{d}}$ with randomly attached pyramids. This satisfies the 2 conditions.

Lovász: Can volume be estimated from poly many random samples?

Eldan09: Exponentially many samples are needed to approximate volume. (Only samples, no oracle.)

Comvex bodies in all lower bounds have complex boundaries: exponentially many facets

Restrict convex bodies: polytopes with m facets (intersection of m halfspaces). VC-dimension is $O(md \ln m)$. PAC learning: poly$_{\delta,\varepsilon}(md)$ samples suffice. Don't know how to efficiently PAC-learn intersetions of 2 half-spaces. There are many computational hardness results.

For fixed distribution: Gaussian: $d^{\frac{\ln m}{\varepsilon^2}} \wedge d\left(\frac{m}{\varepsilon}\right)^m$ time complexity.

We need extra restriction that intersectino is bounded $m \geq d+1$.

Restate problem: given $P \subset \mathbb{R}^d$ with at most m facets, poly(m, d) uniformly random samples from P, efficiently reconstruct P.

If allow membership oracle, can PAC-learn in poly time.

Approaches

1. Polytope learning problem has sample complexity poly$_{\delta,\varepsilon}(md)$.

Min volume polytope Q with at most m facets and enclosing samples is good reconstruction (folklore).

For P_1, P_2 with at most m facets, VC-dimension of $P_1 \backslash P_2$ is $O(dm \ln m)$. By VC uniform convergence theorem, $\frac{1}{2}|U_P - U_Q|_1 = \frac{|P\backslash Q|}{|P|}$ is small.

It is NP-hard to find smallest volume enclosing simplex. But we get a structured distribution; we can use various heuristics but haven't succeeded. We don't even know how to learn cubes using this approach, but they can be learned using another.

2. Method of moments.

   Subproblem: Given samples from a parallelopiped $X = AS + b$, $S \in_U [-1,1]^d$ uniform, $A \in \mathbb{R}^{d\times d}$ nonsingular, $b \in \mathbb{R}^d$, reconstruct parallelopiped.

   This reduces to learning cubes. Assume A is rotation matrix and $b = 0$.

   The class of cubes can be learned efficiently and properly (Delfosse-Loubaton95, Frieze-Jerrun-Kannan96). The fourth directional moment of the cube is

$$F(v) = \mathbb{E}_X(v \cdot X)^4.$$

   If input cube is axis-aligned with unit sides, $F(v) = c_1 - c_2(v_1^4 + \cdots + c_n^4)$. Local min for $\|v\|_2 = 1$ are exactly facets. Use projected gradient descent to find facets.

Learning simplices: FJK96 asked about learning polytopes, in particular simplices. Simplices can be learned efficiently (Anderson-Goyal-Rademacher13). Fastest algorithm for learning intersections of $d+1$ half-spaces under Gaussian distribution required $d^{\ln d}$ time.

Two proofs for learning simplices:

- Third directional moment: more involved than cubes
- Reduce to independent component analysis.

$X = AS$ where $X, S \in \mathbb{R}^d$ are random variables, A is fixed unknown nonsingular $A \in \mathbb{R}^{d\times d}$. $(S_1, \ldots, S_d)$ are independent non-Gaussian components with unknown distribution. FJK, DL results apply to ICA.

Standard $d-1$ dimensional simplex is

$$\Delta_{n-1} = \left\{ x \in \mathbb{R}^d : \sum_i x_i = 1, x_i \geq 0 \right\}.$$

If X is uniform on Δ_{n-1}, $r \geq 0$ has density $\propto x^{d-1}e^{-x}$, then X has identical independent nonnegative components with density $\propto e^{-x}$.

Tensor decomposition methods are a different but related way to utilize moment info, recently popular for latent variable models.

Learning simplices is a special case of LDA, learnable by tensor decomposition.

Can we use method of moments to learn polytopes? We can learn cubes, simplices, cross-polytopes. For general polytopes symmetry is lost. It is hard to extract facet information from moments without symmetry.

Exact reconstruction for polytopes with m vertices if given first $O(md)$ moments in $d+1$ random directions (Gravin-Lasserre,-Pasechnik-Robins12). Doesn't help with polytope learning: estimating kth moment seems to require $2^{\Omega(k)}$ samples. Can we learn with moments of constant order?

Estimate third directional moment in direction $v \in \mathbb{R}^d$ is $\mathbb{E}[(v, X)^3] = \sum_{i,j,k} v_i v_j v_k \mathbb{E}[X_i X_j X_k]$.

Q: does 100th moment tensor suffice to determine polytopes with at most $10d$ facets information-theoretically? Does $M_{100}(P) = M_{100}(Q)$ imply $P = Q$? Answer is no.

d-gons can't be determined for large d by counting argument; lift by making prisms.

Degeneracy issue is common in applications of method of moments. For Gaussian mixture models (Kakade-Hsu12) require means be in general position. If Gaussian are on line, sample requirement can be exponential in number of Gaussians (Moitra-Valiant10).

Question: Moments of order 100 suffice to determine polytopes with at most $10d$ facets if normals are in general position?

3. Radon transform: widely used technique in computerized tomography. Send X-ray in any direction you want, and check how much the signal is attentuated, get length of $L \cap K$. Radon transform of K map lines to distance of intersection with K.

   Radon inversion formula allows reconstruction of K from $R_K(L)$ for all L. This works in higher dimensions too, with $d-1$ dimensional hyperplanes.

   In $\mathbb{R}^3$: $R_K(P)$ is the are of intersection of K with plane P. $(R^* R_K)(x)$ is the integral of $R_K(P)$ over all planes with $x \in P$. Then

$$1_K(x) = -\frac{1}{8\pi^2}\Delta(R^* R_K)(x).$$

   (Similar formulas in higher dimensions.)j

   $R_K(H)/\mathrm{vol}(H)$ is approximately the fraction of sample points in thin band around H divided by width of band.

   Two hurdles: stability of inversion: sensitive to errors. Computation of inversion involves integral over unit sphere in $\mathbb{R}^n$ and multiple differentiation.

21 Computational Challenges and the Future of ML Panel

Panelists: Maryam Fazel (University of Washington), Yoav Freund (UC San Diego), Michael Jordan (UC Berkeley), Richard Karp (UC Berkeley), and Marina Meila (University of Washington)

What are one or two important computational challenges? What might the future bring?

YF: Communication is at least as important as computation. When we think about distributed algorithms—the only way to work with very large datsets—what is the least amount we can communicate and still achieve our task. Usually we think of it as something in the background, but if you think about the barrier model (work in parallel, send back to master), communication scales with number of workers, if you have too many workers, time to communicate documents. I know this from experiments on my Spark cluster. Average load on CPU's is 5%. Something else is blocking it.

MF: Background in optimization. From the POV of optimization, we understand convex very well, but people now talk about nonconvex. Saying nonconvex is not saying anything. We need to say how we're moving away from convexity. On certain problems with strict assumptions we can say a lot, ex. local optima are global, can reach from certain starting point. What if I don't have those assumptions? You might have lots of local minima, but they may be good. You may not want global minima if it is very deep. This also relates to what you do with model afterwards (generalization).

Setting the step size is not obvious anymore. Stopping criteria is not obvious.

What classes of nonconvex optimization problems can we say something specific?

RK: A few impressions of an outsider to the area. The usual model of supervised learning is general and simple. It's the problem of inferring a function from values at sample points. In complete form you specify the training data, its representation, space of possible hypothesis (possibly with a probability distribution), action based on function value, optimization algorithm.

Deep learning seems to have decisive advantages over regression, SVM, Bayesian networks. There have been successes in translation, self-driving cars, etc. Some places of concern: very large datasets are needed. Not clear what function is being inferred—different experts may disagree on training data. Choosing most probable function in a deep net is nonconvex. Another issue is that of comprehensibility: a deep net with huge number of neurons, layers, when it's succeeded, what has it learned, how can you get your mind around what it's doing? If you can't compactly certify, you're weak against adversarial attacks. Safety: If you don't have specification of what it's doing, how do you know it won't go wild and mess up your nuclear reactor, etc.? Rigidity: people learn concepts over time with many environments and examples. Here the format is frozen. It's a static model without a ot of autonomy or plasticity. It doesn't correspond to how humans or biological systems learn. Call from proposals from DARPA: next gen ML tech that can learn from new situations, learning on the fly. We need flexibility and adaptability.

MM: Computational challenges seem to always come with statistical challenges. If the data is so big and divided on so many computers, can we believe that it's iid?

This is not just a curse but a blessing: accumulating evidence that easy data implies easy computation and vice versa. Data fits algorithm (ex. separating hyperplane). When data fits task, learning rates are fast. It doesn't seem to be a time to think of easy data when there is hard data as well.

There are 2 tasks: prediction and explanation. In prediction, a value is satisfactory. I see that prediction from certain POV is solved but explanation is much further behind. A related question you might ask: why do we need explanation?

A big challenge from statistics and computational POV is validation.

Where is ML most successful, pervasive, where is it not? YES: speech recognition, image captioning, self-driving cars, translation, playing chess. Results easy to validate. (Middle: finance, health) Neuroscience, biology, chemistry, astronomy, physics: Hard to validate. ML generates hypothesis fast, but when it's expensive to give labeled data, then ML is not doing such a good job. How to both generate hypotheses and validate them. Do the human job of testing hypotheses and explanations. I think of this as computational challenge because it is harder to apply models, algorithms for smaller data.

We can't expect classic model selection and validation to work. We have to find methods that tell whether predictions are trustworthy.

MJ: I don't think it's so much a computation issue. I think deep neural nets are not hard to optimize. Folk wisdom: from every point there's a downhill path to optimal. I think it's all statistics. We've borrowed math of statistics, but we still don't think like statisticians. There's lots of problems with data, like how is data sampled? You may find out too late. Error is debt, chaos. We're not ready for the future world, with confidence interval on our predictions; we have black boxes.

In the future, doctor will take your genome, put through ML systems, using all previous data. "Neural netoutputs 0.62, greater than 0.6. You need a liver operation." What's the confidence interval? Build a system that quantifies uncertainty. What about when all decisions are related?

This is inference and decision-making. You have to look at where did data come from in the world? Don't just borrow the techniques. Also differentiate between correlation and causation. In real life, a policy planner doesn't know what variables to change. You have to reason and do causal inference. Control requires causal inference. Worry about stability in time-critical, death-critical systems. We're way unprepared. How to talk about the work without all the hype. How to not take on projects not like AlphaGo which causes a storm, people talking about robots taking over the world which have nothing to do with ML. Take on projects having to do with water, health, times, don't make it into NYT but make world a better place, not glorious wave of computer scientists making superhumans. I don't recognize this.

MT: What role should theory have in role of empirical ML?

MJ: Most cool problems are theory problems. Communication is a theory problem. How to get causes? Look at the formal structure, see that you've pieced things apart.

MT: The empirical wave is not paying attention to theory, they are content running SGD for everything, increasing size of nets, adding band-aids...

YF: A lot of the hype is driven by companies like google. Companies are treating it as a playground. It is not what they sell, what they depend on. They will vary something here and there. A lot of it has to do with expectations. One of acronyms I like least is AI. We have no idea what intelligence is. I like IA: intelligence amplification: the idea that we use computers to enhance our ability to do dredge work for us, at the end it's our decision. It's hard to excite young people about it, but it's a valid thing. Ex. How to estimate mean with a large fraction of outliers?

Now everything is AI. People are not changing what they did but are clling it AI.

MF: Justify heuristics in practice Why do they work? What is it about the class of problems, then we can improve heuristics and apply more robustly.

MM: Before we wanted to understand other data sources. NN is new data source. Understand this new process we have created.

RK: When you start dealing with safety-critical applications, controlling robots, health systems, driving a car you better have firm theoretical guarantees to know that system won't go wild.

LR: Is there need from theory to come up with new definitions?

MF: In optimization we need to formulate properties better. Properties of functions, problem instances? What do we mean by finding solution—what solutions are of interest?

YF: Advice on coming up with new models: work on applications. This has been my approach: work with people on sciences applications. Some problems are easy, just understand regression. Quickly you get into problems, how do I analyze data so huge, which examples to train on. In many cases you cannot think of ML as independent entitty. You have process in lab involving decision-making, finding micro-crystal in droplets. You want to make decision-making process. But the more important thing: how do we change process going on in lab, many small decisions in lab, to give them benefit. Go back to interactive learning.

RK: Smoothed analysis, might possibly be adapted to validate supervised learning schemes. Even though may have errors, correct almost everywhere. Take any datapoint, perturb slightly, whp will get modified datapoint where solution will be correct.

Q: Great danger if we use these models and don't understand, like in healthcare, sentencing (racially biased). Putting out these models we don't fully understand, others are using them without understanding. What's our obligations in creating these models to ensure they are used appropriately.

YF: Be very careful about data selection process, labeling process, evaluation process. You and potential users should agree on. When you go forward, use those processes, so statement at the end of the day that it works is justified statistically.

MM: This is a greater research question. Online learning can have something to say about it. How to continue exploring to estimate error while it is in use?

Q: In terms of progress made in field. In other fields Physicists, impact of bomb. Impact of gene editing. This community might be at edge of star-turn. Who has responsibilities of consequences.

MJ: They could track forward consequences. Admit it's a research problem. Fake news epidemic is shocking and real. Mark Zuckerberg: it's not our problem. No. We have to think it through. We have to partner with lawyers, sociologists, take responsibility, not just be computer scientists. Don't say, consequences be damned.

There's no imminent danger, but let's be clear about uncertainty—we don't get that from people talking about ML. We have young people overexcited already. People think AI is solved and everywhere, and just want to cash in. Take machine translation. That problem's "nearly solved". From copying what humans can do, it's not solved but useful, you can build companies of it. But I know I didn't take a trillion sentences and built a neural net. Building a neural net isn't understanding language. We're not doing intelligent things. Translations, dialogue is not rotation. It should be called statistics, but then became ML, AI. It should be IA.

MM: It's an interesting paradox, large nets trained on Wikipedia, it seems there's a model of large companies using a small number of humans' free work to make profits. Where is this going?

RK: Biologists set up guidelines, which genetic manipulations are too dangerous to perform. It's hard to establish that in this field because of diversity of applications. When used to control cars, physical systems, etc. it might be necessary to take field to agree on standards of safety.

JW: No inteligence going on, not the point. Technology could affect society, like how manufacturing affected workers. People rioting. Uber exploiting drivers so they can use them to train self-driving cars, these people who are bedrock of profits will be redundant, and drivers not developing own skills. Confidence intervals in statistics. Suppose you understand that. That doesn't affect social acceptance of algorithm. You can blame your doctor from diagnosing you incorrectly. When you have algorithms makind that decision... Is there any way of formulating this from a math perspective? Beyond ability to discuss mathematically?

YF: I give an example, where you can do that. Biopsies are huge, 20000 times 60000. A pathologists look at in half minute and decide whether there is cancer, they use heuristics. We can use system to point them in places to look. In a lot of cases.

In many decisions which have to do with analyzing medical records, people are doing it now as best as they can, computer can do it better, point out the phenomena that human can agree, that's the right phenomena. In an IA approach, humans are in control. You continue to do it, but I make your work more efficient, accurate. There are less alarmist ways to think about this things that can produce useful products for the world.

Facebook is a great thing, but then it also gives bad actors new opportunities to exploit people. It's a complicated decision: should we disallow, regulate? These are not technical questions.

MJ: This needs a big dialogue beyond our room. The speed has changed. In industrial revolution people had 30 years. A lot of jobs are lost really quickly. Truck drivers can go into call centers. In five years dialogue systems are still crappy, but they can replace call centers. I'm not as alarmist in the long run because people have adapted. Massage therapists make a lot of money now, 100 years ago they did not exist. Brand new jobs will be created. Sitting in a piece of metal hurtling on a road is weird for people a long time ago.

JB: Science fiction is different from social unrest.

MJ: Be slower, more social, less greedy, I don't know how we'll do that.

SV: This wave of ML successes is exciting. Has it revealed to you anything new about nature of computation?

JB: Computational complexity hasn't delivered something interesting for this field yet: average-case, specialized to certain cases. Optimization has lower-bound oracles which match upper-bound rates. Computation emerged in an era of logic. Poly-exp barrier. We're not in that world, we need new models, average-case doesn't do justice.

RK: CC has focused on worst-case analysis, which makes it irrelevant, NP-hardness doesn't matter in practical situations. Nothing has replaced worst-case analysis. Average-case models are too simplistic to be useful in applications. There's a challenge to find more optimistic models, not overly trivial. This is open question not just for computational learning theory but combo optimization.

MJ: Nemirovskii-Yudin was good. But now there's more problems.

DD: The practice of the community has been to come up with regularity conditions. Deep learning: throw things against the wall and it sticks, even if in difficult complexity class. We can make fewer assumptions about the world, ex. translation invariance. Is there any hope for theory about data itself across different domains? "Theory of data"?

MJ: Theory of group actions, equivariant. Put constraints on estimator.

RK: d-dimensional Euclidean. What about, ex. language?

MM: Good old Bayes nets. It's hard to do inference. What progress has been made in other fields? Look at success in SAT solvers, solve NP-hard problems, and build models based on other fields of computation.

Q: In what way has ML been excessive? Are deep nets too complex a model? Come up with more reasonable models?

RK: Deep nets used successfully don't have much explanatory power. We have no idea what they're doing, what they're representing, whether they can be simplified without losing effectiveness?

MM: I see it as problem for future to measure how much overparametrization. In stats it's known that to predict well is not to understand the distribution of the model. To predict well you have to overparametrize unless you have sufficiently large sample. Interesting paradox. One cannot explain and predict with the same model.

YF: One way to think is in terms of scaling. Trivial way to learn is nearest neighbors, store all data and it works well. The problem is that the size of the representation is the size of the data. For neural nets, I'm convinced it learns if the number of parameters you need stops increasing as the amount of data increases. It should not need to increase. It's a hard question to do tractably. Can you find a low-dimensional representation, something that does not increase, size of model does not increase linearly with size of data?

SA: A learned neural net can be sparsified dramatically.

YF: How many nonzero parameters do you need?

SA: You can make it binary.

YF: It's a question about scaling.

SD: Many of us are in academia, we have this common experience, there are hordes of students who just want to learn about deep learning. As responsible scholars we make sure they learn the fundamentals, all the aspects of ML before we arrive at that lecture about backprop. That's good but I can't help feeling in this context, there's other stuff we also need to teach: societal impacts, etc., not part of standard ML class. Any thoughts about what those things should be, what form it should take?

MJ: Causal inference, we don't teach that. For medical domain, that's what it's all about. Does drug have causal effect? You can do nearest-neighbor. It's not a gradient descent. You can't give both treatment and control. If you have a twin... if you do NN you can find a doppelganger, they got treatment, you got control. We can teach that to freshman, and we're not making them excited about that. They want to classify cats in images. You don't have imagination to think about that other class of problems. Get people oriented into these sci deep unsolved problems.

Teach AB test. We designed data science class in Berkeley. In industry, on first day, learn to do AB test. Give users 2 versions of website. Most people in ML don't teach that!

There's the permutation test: aggregate 2 columns, permute randomly, break into 2 new columns. Is original in tail or middle?

Ex. social makeup of juries.

Not just be in AI subworld. We're in social world, etc. I wish journals would go back to using ML instead of AI.

MJ: Cultural divide, statistics vs. ML.

JB: You had a great paper on algorithmic weakening...

MJ: CS, statistics was originally 1 field that happened to get broken apart. Combine the pessimism of statisticians, optimism of computer scientists, blend, and work together.

22 Computationally Tractable and Near Optimal Design of Experiments (Aarti Singh, Carnegie Mellon University)

Classical experimental design problems in statistics tend to have combinatorial solutions. We consider computationally tractable methods for the experimental design problem, where k out of n design points of dimension p are selected so that certain optimality criteria are approximately satisfied. We prove a constant approximation ratio under a very weak condition that $k > 2p$, and a $(1 + \varepsilon)$ relative approximation ratio under slightly stronger conditions in which k is still a linear function of p. Numerical results on both synthetic and real-world design problems verify the practical effectiveness of the proposed algorithm.

The higher-level picture: we have big data but small labels. The bottleneck is there are lots of images but someone has to sit down and label them. Sometimes labels come from expensive experiments. Experimental design is non-feedback-driven (choosing which labels to collect at the beginning); active learning is feedback-driven.

There's a lot of need for statistical grounding for label efficient methods. How can we make experimental design more tractable?

Big data, small labels:

1. profits of buildings, energy usage
2. healthcare records, patient satisfaction
3. natural images as visual stimuli, brain response (which images to show people in a scanner)
4. experimental parameters' settings, material properties

Given a large pool of data with label budget, how to subsample data points for labeling?

We look at experimental design for linear regression. The stochastic model is $y = X\beta + \varepsilon$, $\varepsilon \sim N_n(0, \sigma^2 I_n)$. We are in the large sample setting $n > p$.

We want to estimate the true regression coefficient vector β, and predict labels $Z\beta$ of potentially different data $Z \in \mathbb{R}^{m\times p}$.

What if we can only collect k labels where $p \leq k \leq n$. This is a subset selection problem.

If we have all the labels, we can find the least squares solution,

$$\widehat{\beta} = (X^T X)^{-1} X^T y.$$

Estimation and prediction error are

$$\mathbb{E}[\left\|\widehat{\beta} - \beta\right\|] = \sigma^2 \operatorname{Tr}((X^T X)^{-1})/m \tag{30}$$

$$\mathbb{E}\left[\frac{\left\|Z\widehat{\beta} - Z\beta\right\|_2^2}{m}\right] = \sigma^2 \operatorname{Tr}((Z^T Z)(X^T X)^{-1})/m \tag{31}$$

We care about minimiax optimality

$$\inf_{A \in \mathcal{A}(k)} \sup_{\beta} \mathbb{E}[R(\widehat{\beta}, \beta)].$$

An algorithm is sampling strategy plus estimator. $R = \frac{\left\|Z\beta - Z\widehat{\beta}\right\|_2^2}{m}$.

Experimental design in classical statistics: For estimation, A-optimality (average) is

$$\min_{|S| \le k} \operatorname{tr}((X_S^T X_S)^{-1})$$

For estimation, V-optimality (variance) is

$$\min_{|S| \le k} \sigma^2 \operatorname{tr}[(Z^T Z)(X_S^T X_S)^{-1}].$$

Enumerating over subsets takes n^k time.

Avron-Boutsidis13: polytime deterministic algorithms that select $|S| = k$,

$$\underbrace{\left\|X_S^+\right\|_F^2}_{\operatorname{tr}(((X_S)^T X_S)^{-1})} \le \frac{n - p + 1}{k - p + 1} \underbrace{\left\|X^+\right\|_F^2}_{\operatorname{tr}((X^T X)^{-1})}.$$

Greedily remove rows. This outputs $|S| = k, k \ge 2p$ such that difference is $O(\frac{n}{k}\sigma^2 \operatorname{Tr}((X^T X)^{-1})$. This compares against the full solution, not the best k-subset.

What if we analyze this under a stochastic model? Sample rows according to leverage scores $h_{ii} = x_i^T (X^T X)^{-1} x_i$, reweight, and solve. Leverage score sampling can be worse than uniform sampling for estimating regression coefficients in stochastic model.

We make a convex relaxation

$$f_{opt} = \min_{\pi, \|\pi\|_1 \le k, 0 \le \pi_i \le 1} \operatorname{tr}[(X^T \operatorname{diag}(\pi) X)^{-1}] \tag{32}$$

$$g_{opt} = \min_{\pi, \|\pi\|_1 \le k, 0 \le \pi_i \le 1} \frac{1}{m} \operatorname{tr}[(Z^T Z)(X^T \operatorname{diag}(\pi) X)^{-1}]. \tag{33}$$

Then sample according to the distribution given by π. Convex relaxation lower bounds the minimax error.

Convex program can be cast as SDPs, but still only scales to 200-300.

Sampling:

1. Without replacement:

   Soft budget: $|S| = O_p(k)$, select row $i \sim Ber(\pi^*)$ for each row.

   Hard budget $|S| \le k$: Pick row using $Ber(\pi_i^*)$ until $|S| = k$.

2. Sample with replacement:

   Soft budget: Sample row $\sim \pi^*$. Repeat $\left\lceil \frac{p}{k x_i^T \Sigma^{*-1} x_i} \right\rceil$ times, $\Sigma^* = X^T \operatorname{diag}(\pi^*) X$.

   Hard budge: Stop when $|S| = k$.

If $\frac{p \ln k}{k} = O(\varepsilon^2)$ and (some condition on covariance condition number requirement), then get $1+\varepsilon$-optimal estimate to the combinatorially optimal solutions for f_{opt}, g_{opt}.

Key proof idea: we need to show

$$\operatorname{Tr}((X_S^T X_S)^{-1}) \leq (1+\varepsilon) \operatorname{tr}((X^T \operatorname{diag}(\pi^*) X)^{-1}).$$

Look at eigenvalues. Show that subsampled covariance is good spectral approximation of optimal covariance, cf. Spielman-Srivastava 11. We only need one-sided unweighted sparsifier for a weighted graph.

Can we get relative $1+\varepsilon$-error approximation with hard budget $|S| = k$?

Essentially we are doing a L^1 relaxation. $\operatorname{Supp}(\pi^*)$ is $k + p^2$ in theory, but $k + p$ in simulations.

Idea: Instead of sampling data points, according to π^*, start with support of π^* and greedily remove some. We get this to work but with $|S| = k = \Omega\left(\frac{p^2}{\varepsilon}\right)$ instead of $p \ln k / k = O(1)$.

With Allen-Zhu, get $1+\varepsilon$-approximation, can handle other criteria (A, V, D, E, T, G), $|S| = k = \Omega\left(\frac{p}{\varepsilon^2}\right)$. Pushing down to $|S| = k > 2p$, we get $O(1)$ approximation.

Faster implementation via projective gradient descent. While objective hasn't decreased enough, do projective gradient descent step.

Projection of N-dimensional vector π onto intersection of ℓ_1, ℓ_∞ balls:

$$x^* = \operatorname{argmin}_x \|x - \pi\|_2^2 \text{ s.t. } \|x\|_1 \leq c_1, \quad \|x\|_\infty \leq c_2.$$

(The is one of the most rediscovered subproblem solutions...)

Real-data experiment: material synthesis of thin films of TiO_2 at low temperatures using microwaves (ex. for solar cells). Optimize input settings to get best coverage. Predict material coverage using few experimental settings. You can get close with few experiments. Compare leverage score vs. greedy: greedy chooses more diverse settings.

Wind speed prediction: Greedy and fedorov do the best. Sample improves over leverage score but requires slightly larger budget.

Minimum eigenvalue as regret minimization:

$$\lambda_{\min}(X_S^T X_S) \geq \frac{1}{1+\varepsilon} \lambda_{\min}(X^T \operatorname{diag}(\pi^*) X) \tag{34}$$

$$\text{whiten } \widetilde{x}_t = (X^T \operatorname{diag}(\pi^*) X)^{-\frac{1}{2}}. \tag{35}$$

Minimize regret

$$R(\{A_t\}_{t=1}^k) = \sum_{t=1}^k \langle F_t, A_t \rangle - \inf_{U \succ 0, \operatorname{tr}(U)=1} \sum_{t=1}^k \langle F_t, U \rangle \tag{36}$$

where $F_t = \widetilde{x}_{i_t} \widetilde{x}_{i_t}^T$. Use: follow the regularized leader/mirror descent.

Extension: Generalized linear model $y = g(X\beta) + \varepsilon$. For MLE,

$$\mathbb{E}[\left\|\widehat{\beta}_n - \beta_0\right\|^2] = (1 + o(1))\,\mathrm{tr}(I(X, \beta_0)^{-1}).$$

Future directions:

- Relative error approximation with budget linear in p. Can we obtain relative $1 + \varepsilon$ error with $|S| \sim k > p/\varepsilon$?
- Fast converging algorithm for convex relaxation.
- Experimental design for bit budget.
- High-dimensional setting: what if $n \ll p$, with sparsity or other prior assumption on regression coefficients, $\|\beta\|_q^q \leq R_q$.

Ex. MRI sparse in wavelet basis but acquired in Fourier basis. $y = FW\beta + \beta$. Which frequencies to acquire?

Proposed approach: used half budget to learn support. Use convex solution on support of β.