Contents

1 Optimization and Generalization Properties of Deep Networks – Peter Bartlett

I wanted to talk this morning about the optimization properties of deep residual networks. Think of these as deep compositions of nonlinear vector-valued functions. What consider these classes of functions? The claims you here is that they are good ways of learning representations via depth. In computer vision, you see this – the early units in a network of these things computes simple functions of the image, like detecting edges, and later units are computing more useful things. It's a nice simple way of representing complex functions – depth seems to help here. There are results pointing to particular functions which can be computed exactly with a network of a certain depth, and if you want to approximate with a shallow network, you have to have an exponentially wider net. On the other hand, this nonlinear parametrization should make us pause and think about the optimization question. Nonlinear parameters should lead to a difficult optimization problem. That's what I want to talk about today: Some situations where we can avoid some of these difficulties.

I want to focus on residual nets throughout the talk. First we'll talk about the error surface, and then what we can say about simple optimization methods for some simple cases. What is a deep residual network? In a ReLU net there might be two stacked layers, and then put through some nonlinearity. In a deep residual network, there are these identity connections: Instead of computing a nonlinear function, you compute a nonlinear function plus the original input. In the usual network, 0 parameters give you the zero function, in residual net, it gives you the identity function. It also might help identify useful connections throughout the network.

Why are there sudden drops in the error? This is part of the dark art of training these networks. What the practioners do is set the step size, then make it smaller – when you adjust it, you get a sudden decrease. It's not just a straight stochastic descent, there's a particular schedule and step size. A few years ago, Moritz Hardt and Tengyu Ma had a nice paper looking at the linear case. If we have an invertible matrix, we can write it as a product of matrices. We need the determinant of our invertible matrix to be positive. We can then write

$$A = (I + A_m) \cdots (I + A_1)$$

where $\|A_i\| = O(1/m)$. The factors in this decomposition can be made closer and closer to identity. The O hides a notion of scale in the mapping. There's a rich family of linear maps we can represent as a composition of near-identity linear maps. The second result Hardt and Ma showed is that it implies some nice properties of the error surface in an estimation problem. For a linear Gaussian model $y = Ax + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$, consider choosing $A_1, \cdots, A_m$ to minimize $\mathbb{E}\left[\|(I + A_m) \cdots (I + A_1)\|^2\right]$. If $\|A_i\| < 1$, then every stationary point of the above condition is a global optimum – first order condition for optimality. No local optima or even saddle points.

The first part of the talk extends those two results to the nonlinear case. If we have a smooth invertible map h, then we can spread it out into a composition of near-identity functions. What do we mean by near-identity? Let the Lipschitz semi-norm be defined by $\|f(x) - f(y)\| \leq \|f\|_L \|x - y\|$. Then $\|h_i - Id\|_L = O(\frac{\log m}{m})$.

Theorem 1.1. *Consider a function $h : \mathbb{R}^d \to \mathbb{R}^d$ on bounded domain. Suppose it's differentiable, invertible, and smooth, this implies Lipschitz, also assume that the inverse is Lipschitz, and positive orientation (positive determinant). Then you can come up with m functions $h_i : \mathbb{R}^d \to \mathbb{R}^d$ satisfying $\|h - Id\|_L = O(\frac{\log m}{m})$ and $h_m \circ \cdots \circ h_1 = h$ on the domain. Here, the big O hides the domain radius squared term.*

Proof. We can construct explicitly these h_i, the whole argument is to construct them and show they satisfy this near-identity condition. Assume $h(0) = 0$ and $Dh(0) = Id$ (else shift and linearly transform). Then construct h_i so that $h_1(x) = h(a_1 x)/a_1$. Then we have a sequence of scalars $a_2, \cdots, a_m$: $h_2(h_1(x)) = h(a_2 x)/a_2$, and so on: $h_m(\cdots(h_1(x))\cdots) = \frac{h(a_m x)}{a_m}$. Pick $a_m = 1$ so that $h_m \circ \cdots \circ h_1 = h$, and ensure that a_1 is small enough. Ensure that a_i and a_{i+1} are close so that $a_i \approx Id$. □

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

The analogue of the second result is as follows:

Theorem 1.2. *For (X, Y) joint distribution, define*

$$Q(h) = (1/2)\mathbb{E}\left[\|h(X) - Y\|_2^2\right]$$

and define minimizer $h^*(x) = \mathbb{E}[Y|X = x]$. Consider a function $h = h_m \circ \cdots \circ h_1$ where $\|h_i - Id\|_L \leq \epsilon < 1$. Then you can show

$$\|D_{h_i}Q(h)\| \geq \frac{(1-\epsilon)^{m-1}}{\|h - h^*\|}(Q(h) - Q(h^*))$$

where $D_{h_i}Q$ is a Frechet derivative and $\|h\|$ is the induced norm. So if you find a stationary point in this sense, you must have $Q(h)$ is close to $Q(h^*)$. If the composition h_i is suboptimal and each function is near identity, then there is a downhill direction in function space: Functional gradient of Q with respect to h_i has a downhill direction. However this direction might be orthogonal to functions that can be computed with a fixed architecture. We should expect suboptimal stationary points in the ReLU or sigmoid parameter space.

The theorem does not say there are no local minima of a deep residual network of ReLUs or sigmoids with a fixed architecture, e.g. a parametrization – the theorem is all happening in function space. In terms of something you can measure it's not so clear.

Proof. The idea is we are thinking of functions at each layer as being close to the identity – that gives us information about the Lipschitz semi-norm of the difference, etc. When we compose things and take derivatives, we still get things which are close to the identity. We can think of the criterion as how far h is from h^* by using the projection theorem. Then we can apply the chain rule of Frechet derivatives. It is possible to choose a direction, and then you need to figure out the scale – we can argue things are not too small when all the h_i are close enough to the identity. □

Now, there are a bunch of things we'd like to be able to answer.

(a) What if the mapping h is not invertible? e.g., we would like to consider $h : \mathbb{R}^d \to \mathbb{R}$. If h can be extended to a bi-Lipschitz mapping to $\mathbb{R}^d$, it can be represented with flat functions at each layer. What if it cannot?

(b) Implications for optimization? Related to Polyak-Lojasiewicz function classes, proximal algorithms for these classes converge quickly.

(c) Regularized gradient methods for near-identity maps?

The second part of the talk is now skipping again to the easy case of linear networks, and looking at what we can say with simple first order optimization methods. We are thinking of $f_\Theta(x) = \Theta_L \cdots \Theta_1 x$. We take $(x, y) \sim P$, and consider minimizing $\ell(\Theta) = \frac{1}{2}\mathbb{E}\left[\|f(\Theta) - y\|^2\right]$.

Just assuming you have the population risk, we compute the exact gradient (ignoring practicalities of computing this). Then we'll make a few assumptions: $\mathbb{E}\left[xx^T\right] = I$ are convenient (we could whiten things to make that true). Secondly, our $y = \Phi x$ for some matrix Φ (true wlog because of the projection theorem). Think of Φ as the least squares map. Finally, let's write an L-fold combination of linear maps is just $f_\Theta(x) = \Theta_{1:L}x$. Gradient descent starts out with everything as identity, and then shifts

$$\Theta_i^{(t+1)} = \Theta_i^(t) - \eta(\Theta_{i+1:L})^T(\Theta_{1:L}^{(t)} - \Phi)(\Theta_{1:L}^{(t)} - \Phi)^T$$

Then we can get a linear convergence result. The proof idea is as follows: The gradient is big when the loss is big, and the Hessian is small for near identities – this tells you gradient descent will make good progress, and you don't stray too far from the identity as you proceed, as long as your step size is good. This demonstrates the gradient descent approach converges to the optimum – this is only if the identity is sufficiently good, which is a rather significant assumption.

If we're willing to assume the map is symmetric, we can get something pretty good.

Definition 1.3. Positive margin matrix.
A matrix A has margin $\gamma > 0$ if for all unit u, we have $u^T Au > \gamma$.

Theorem 1.4. *Suppose Φ is symmetric. Then everything works out nicely: If Φ has positive margin and the depth is $L \geq \log(\|\Phi\|_2/\gamma)$, and η is small enough, then after poly($L, \|\Phi\|_2/\gamma, 1/\eta$) $\log(d/\epsilon)$ iterations, gradient descent achieves loss less than ϵ.*

Proof. When we're working in symmetric case, if least squares map Φ is symmetric, then we just always stay in the same diagonalized space – all the dynamics decompose into diagonalizable dynamics and everything stays in the subspace – then eigenvalues stay positive, and everything will work out. □

If we move beyond the symmetric case, then suppose we have positivity condition for least squares map: Then after some number of iterations, we get near optimal – this is not just gradient descent.

Definition 1.5. Power projection algorithm.
Take a gradient step for each Θ_i. Project $\Theta_{1:L}$ onto the set of linear maps with margin γ – you want to keep the scales of these matrices the same. Set these things as the balanced factorization – write matrices with particular singular values as a product of matrices with the same singular values – use polar decomposition.

To summarize:

(a) Converges if $\ell(0)$ small (identity good)

(b) converges for positive symmetric map

(c) cannot converge for symmetric map with negative eigenvalue

(d) regularized gradient converges for positive map

(e) convergence is linear in all cases

(f) TODO: deep nonlinear residual networks? Can we do the same thing in function space?

2 Nonconvex Optimization Algorithms with Complexity Guarantees – Stephen Wright

I want to start off with some background and half-baked observations. The ML algorithms cheat sheet started appearing on my Twitter feed a few months ago. The idea is it's some sort of decision graph for deciding what algorithm I'll use. Many of these are optimization methods. Therefore, optimization is important in machine learning. Optimization formulations translate statistical principles (e.g., risk, likelihood, significance, generalizability) into measures and functions that can be attacked with an algorithm. The algorithms part is important – practical means to solve these problems, but black-box approach often doesn't work well – the context is important. Duality is valuable in several cases (e.g. kernel learning). Nonsmoothness appears often as a formulation tool to promote generalizability, but often in a highly structured way that can be exploited by algorithms. ML's influence on continuous optimization: ML has a different perspective on several issues:

(a) We (non-ML) didn't care about computational complexity and global convergence rates as much. Probably theoretical computer science influence – that causes us to take a look at a lot of algorithms and prove things about complexity.

(b) Fast local convergence rates are less interesting – asymptotic superlinear rates – much less impressed in ML world. I speculate because the domain of fast convergence is below the range of accuracy for empirical risk – that's not helpful for ML.

(c) Quite often, you prefer a cheap approximate solution compared to an expensive accurate solution.

Most algorithms are very old, ideas go back decades to the fifties even – these are being dusted off, taken out of the closet. Most are convex, but this trend continues in nonconvex formulations. Steepest descent + onise, trust-region, nonlinear conjugate gradient and L-BFGS, and Newton-conjugate gradient.

The word algorithm in ML is often used to denote the process that takes a training set and maps it to a predictor. You hope that you end up with good generalizability. I think that's how ML ppl tend to use the word algorithm. When you come use optimization, you'er breaking the algorithm into two steps: You write down optimization problem, and apply optimization algorithm – it's somehow decomposed. Thus, it's possible to get good generalizability which falls upon both the formulation and the algorithm – that's putting a lot of responsbility on the algorithm that wasn't there before. People gave us problems and asked us to solve it – nowadays, algorithms are pursuing a more nebulous agenda – they're not taking a given formulation. New questions arise: does small-batch SGD give results with better generalizability in neural nets? Says nothing about whether it does a better job of minimizing training error. Is the algorithm finding a "low-norm" solution generalizing better? These don't have much to do with minimizing objective – not exclusively at least.

A lot of stuff nowadays is nonconvex. Let's talk a bit about nonlinear least squares. This is where your objective is

$$\min_x f(x) := \frac{1}{2}\sum_{i=1}^{n} c_i(x)^2$$

We have

$$\nabla f(x) = \sum_{i=1}^{n} c_i(x)\nabla c_i(x)$$

and

$$\nabla^2 f(x) = \sum_{i=1}^{n} \nabla c_i(x)\nabla c_i(x)^T + \sum_{I=1}^{n} c_i(x)\nabla^2 c_i(x)$$

You can often get away with just the first term. This is a classical technique for nonlinear least squares. This is called Gauss-Newton. If the residuals are large, the model is wrong, so who cares about the slow convergence? You shouldn't be blindly solving the problem, ask questions about context.

Robust linear regression: Noise has a different structure from Gaussian noise; there are various possible regularizers which one can use. Tukey biweight, MCP, SCAD regularizers. These are sources of non-convexity.

In low-rank matrix completion, you solve a least squares problem which matches observations as much as possible. You can get a nonconvex problem by parametrizing $X = UV$, and doing a non-convex optimization property. Despite nonconvexity, near global minima can be found when A_j are incoherent. Use appropriate initialization, or observation that all local minima are near global. Saddle points are shown to have Hessians with negative curvature – "strict saddle" - this can be useful in algorithms. Now, consider deep nets. The objective you use is a summation of a loss function over the data, and SGD is the method of choice. There are several issues, namely to do with stepsize selection and the schedule, but also to do with batchsize. On their face these are engineering issues, and a lot of the work is engineering, but these affect generalizability too. Are local minima global? Of course not, but see Peter Bartlett's talk for some specific examples.

Now let's talk about methods we've been working on for smooth nonconvex optimization. We are seeking a second-order necessary point: zero gradient and PSD Hessian. Assume f is bounded below. We will look for approximate second order necessary points.

There are classical practical algorithms for large scale smooth nonconvex – Newton-CG, L-BFGS, Nonlinear CG. Limit points are stationary, and you can get fast local convergence to second order sufficient point. In 2006, Nesterov and Polyak spurred interest in getting complexity guarantees. Cubic regularization and trust-region methods have accrued interest. Going back to these methods and tweaking them for complexity bounds and asymptotic guarantees is now of itnerest. This has spurred interest in ML community – they've come up with adapting accelerated gradient, gradient descent with judiciously injected noise, approximate minimization of cubic approximation (Agarwal et al. paper).

I will show a method on two slides which give an approximate second order necessary point in $O(\max(\epsilon_g^{-2}, \epsilon_H^{-3}))$. If gradient norm is above ϵ_g, take a short steepest descent step. If it's below ϵ_H, then find direction such that it's an eigenvector of the min eigenvalue, and then take a specific length step in this direction. If you use first kind of step, use quadratic upper bound. Otherwise, use cubic upper bound. Regardless of which kind of step you take, you can quantify amount of decrease in terms of these tolerances. Then you can divide by amount of progress you make in every step, that gives you the complexity. To get to optimality, the lowest-complexity methods improve this dependence, but you need a much more elaborate algorithm. What about fundamental operation complexity? If you take the second kind of step, you need the decay corresponding to the most negative eigenvalue – use probabilistic Lanczozs method – a bunch of Hessian-vector product operations – then your operation complexity becomes $\max(\epsilon_g^{-2}, \epsilon_H^{-7/2})$. So you pay a price for using the second step. Some of the low complexity methods proposed recently are overly complex and not appealing in practice. We want to find an algorithm that is based on one of the classic algorithms for general smooth nonlinear unconstrained optimization. Our first attempt at this was a line-search method – steepest descent, negative curvature for $\nabla^2 f(x)$, Newton: $-\nabla^2 f(x)^{-1}\nabla f(x)$, slightly damped Newton: $-(\nabla^2 f(x) + 2\epsilon_H I)^{-1}\nabla f(x)$. Use randomized Lanczos to compute negative curvature, CG to compute Newton approximately. This gets you $\epsilon_H^{-7/4}$. We put on arXiv a few months ago a more stripped down procedure. We have a negative curvature direction, and an approximate slightly damped Newton – now we can use randomized version of CG to find negative curvature directions. We don't need Randomized Lanczos. We can also engineer things so that it stops after $O(\epsilon_H^{-1/2})$ steps. We don't need other tools, and end up with same sort of complexity: $\tilde{O}(e^{-7/4})$, and approximate stationarity is also guaranteed.

Three basic components in the algorithm:

(a) Modified CG – to detect negative curvature

(b) Minimum eigenvalue oracle – find direction of curvature less than $-1/2\epsilon$ for a given matrix, or certify that all eigenvalues are $> -\epsilon$.

(c) Damped Newton/ Curvature: Damped hessian is just hessian + 2ϵ.

The modifications for Modified conjugate gradient come in the termination step – we have to perform several nonstandard termination tests. What are the nonstandard termination tests. The first is if the residual drops below a small fraction of the original residual, we have found a sufficiently accurate solution. If damped Hessian $d^T \bar{H} d \leq \epsilon \|d\|^2$, declare negative curvature. Finally, the weird one is if the residual is at least a certain value – then residual is not decreasing as rapidly as it would be if the eigenvalue were lower bounded by ϵ. If this last case shows up, you can go back and look at previous conjugate gradient directions. So we can show that the number of CG iterations is at most approximation $\epsilon^{-1/2}$ – this is similar to Nesterov acceleration. But CG is historically preferred in practice. That's the conjugate gradient component of the method.

The next is the minimum eigenvalue oracle. Can be implemented with randomized Lanczos in $\tilde{O}(\epsilon^{-1/2})$ iterations. We showed you can use conjugate gradient instead, and you get same complexity. They both work with same Krylov space (this is the span of the things you've seen so far). Then you find the minimum eigenvector in that subspace. Conjugate gradient is finding minimum eigenvalue problem with an offset in the same Krylov subspace. They'll find different answers, but they'll both come up with negative curvature directions – this is much better because conjugate gradient can be implemented much more efficiently. We tried doing this on Tukey Biweight function. Nonlinear CG tended to do well in our experiments, and converge faster. We were somewhat similar to L-BFGS, and heavy ball somewhat slower. Sometimes Newton-CG has a better solution.

Future work:

(a) Line search strategy is important

(b) Can we enhance Newton-CG to have better numerical performance?

(c) Some unresolved issue regarding Nonlinear-CG and L-BFGS – not much in theory to back them up! Currently analysis of L-BFGS is "not too much worse than steepest descent" – doesn't say much more than that, but its performance is much better than steepest descent.

3 Recovering a Hidden Hamiltonian Cycle via Linear Programming – Yihong Wu

We observe a weighted undirected complete graph on n vertices with weighted adjacency matrix W. We have a latent Hamiltonian cycle C^*. $e \in C$ is drawn from distribution P, and $e \notin C$ is drawn from distribution Q. Goal is to recover C^* as closely as possible. For this talk suppose $Q = \mathcal{N}(0,1)$ and $P = \mathcal{N}(\mu, 1)$. $W = \mu \cdot$ adj matrix of C^* + noise. One application of this model is DNA scaffolding.

Algorithmically, spectral methods fail miserably: If you have no noise, then if you look at the second eigenvector of the cycle, the phase will give you the cycle length perfectly. However, the spectral gap of the cycle is too small. The deviation of the noise is too large in comparison, and in noisy settings, the method will fail. If you use thresholding, you'll get some sub-optimal (in the constant) threshold – this talk is about attaining the exact constant. Greedy merging improves the constant a bit. In this talk, we show that using certain LP relaxations will achieve the sharp threshold.

In general, the threshold is determined by the Renyi divergence: The LP works when $D_{1/2}(P\|Q) - \log n \to \infty$. This is optimal under mild assumptions – e.g. Chernoff bound is tight. Let's talk about convex relaxations of TSP. One can first write down an ILP. One such constraint is known as subtour elimination. The subtour LP is a relxation which drops the $\{0, 1\}$ constraint and changes it to $[0, 1]$. In the LP we will use, we will only use degree constraint and box constraint – This is called a Fractional 2-factor LP. This has been analyzed in the worst-case sense. You usually get some integrality gap for metric TSP. We proved the most relaxed version (compared to various SDP strategies) succeeds (therefore all the other more complicated things succeed too).

The proof approach at a high-level could have two possible approaches: (1) – construct a dual witness that certifies the ground truth w.h.p. (e.g. KKT conditions). This is successful in proving SDP relaxation in a bunch of ways. The limitation is that the construction is ad-hoc – it's like black magic, you have to guess what the dual is. You have to show dual-feasible w.h.p. This is not easy.

The other hand, there's the primal approach which is direct – show there's no other feasible solution other than the one you want to prove – not easy to do with SDP. But with LP, you know the solution will be at the extremal points of the feasible set. The feasible set is a polytope, so it will be a vertex of the polytope. That lets you execute the primal approach for this problem all the way up to the information theoretic threshold. The dual approach gives you KKT conditions (Farkas' lemma) – then you have to pick feasible choices of dual variables.

In the primal approach, you show there's nothing better: Look at polytope $\{X \in [0, 1]^{n \times n} : \sum_{j=1}^{n} X_{ij} = 2\}$. The proof heavily exploits properties of the extremal points of this polytope. Half-integrality is a key property.

4 GANs for Compressed Sensing and Adversarial Defense – Alex Dimakis

A deep generative model is a feedforward neural net from vector $z \in \mathbb{R}^k$ and produces $G(z) \in \mathbb{R}^n$, differentiable and Lipschitz. Assume $k << n$. This is a model for the source code (e.g. info theory) – we've never before had good models for sources. This is a differentiable compression mechanism. It can be trained to take Gaussian z and produce samples of complicated human faces. Training can be done using standard ML (VAEs) or adversarial training (GANs). I don't particularly care if you do one or another. I only care if you give me this blackbox beforehand, and it's differentiable. You can do linear interpolation in the latent space – does it look like it's smoothly changing, or does it jump? It smoothly interpolates. GANs produce good images. What can we do with them now?

I'm going to say you can solve compressive sensing with them, and you can also solve nonlinear inverse problems, as long as the observation process is differentiable.

Compressed sensing: usual spiel. When do you want to recover some unknown vector by observing linear measurements on its entries? Have to assume sparsity. But they can be sparse in a known basis – you can design it, or use a wavelet basis. Idea: Take sparsity out of compressed sensing. Sparsity is a decent model. Now we have data driven models. Replace sparsity with generative model. Then hopefully we can still solve. How do we do that?

Instead of assuming x is sparse, assume it comes from a pre-trained generative model.

Assume x^* is in the range of a good generative model. How do we recover $x^* = G(z^*)$ given noisy measurements? Replace sparsity with neural network; before we used Lasso, how do we recover now? Given target image x, how do we invert the GAN? E.g., find target z such that $G(z)$ is very close to x – just define loss $J(z) = \|G(z) - x\|$ looks as much as possible like the observation image. This function is differentiable with respect to z, so just do gradient descent on z. Let's take a variation that's a little more interesting. Now we have in-painting - observe only parts of image. There is no point to search for z so that it looks like the image of a person with a huge black square in the middle. Instead, dream of an image which agrees with my observations. So you essentially write this loss function: The measurement of the created image agrees with the measurement of what we look at – then do gradient descent. e.g. $J(z) = \|AG(z) - Ax\|$. We have a proof for the measurement process not creating a problem – there's absolutely no reason why gradient descent should work. But it works well. Another problem is super-resolution – you observe the blurring of the pixels. You assume you know the blurring operator. Do gradient descent in this space, you get this. It's comparable to SOTA for this problem. So the general algorithm is as follows: $z \to G(z) \to A \to y$. Then apply $\min_{y\in\mathbb{R}^k} \|y - AG(z)\|_2$ – here we just want the optimizer to solve this problem. This is nonconvex, and can be NP-hard, but still it works for reasons we don't understand. Our algorithm is to do gradient descent in z-space – this works because we can backprop. Even if it's a differentiable process, it is still exactly the same algorithm as long as you can backprop through it. Lasso beats us at some point – why do we flatten out with the deep net? After a few measurements, you've gotten the best possible representation that the GAN can produce. More measurements won't help you after a while. It has to do with the fact that the number of parameters in the GAN is too small. More nuance means more measurements. People have done follow-up work to combine Lasso and the generative model (Stephano Ermon). This is not something new – let's put structure beyond sparsity – model based compressed sensing, etc. What is different from well-known work in signal processing? Backprop – this model for sparsity is differentiable end-to-end. If you had a deep network which could find Hamiltonian cycles – maybe the neural network can go BELOW the information limit and then take over the world!

What can you prove now? Let $y = Ax^* + \eta$. If you solve $\hat{z} = \min_z \|y - AG(z)\|$. Theorem 1: If A is i.i.d. $N(0, 1/m)$ with $m = O(kd \log n)$. Then the reconstruction is close to optimal:

$$\|G(\hat{z}) - x^*\|_2 \leq C \min_z \|G(z) - x^*\|$$

We can get $C = 6$. Another more general version is that the number of measurements has to be $m = O(k \log L)$, where L is Lipschitz constant, which is exponential in the depth. I take the GAN off the shelf – if you want to sense MRI, I need an MRI-GAN. People are actually working on this – generative models are useful for conditioning appropriately. The key open problem is analyzing the solution of this problem.

Our algorithm works even for non-linear measurements: Now, A can be any non-linear operator because you can keep differentiating. We can't prove anything about it. Under some conditions, there are more interesting conditions that can be approved.

How is this proved? First show RIP, then Lasso will work. For a set of images (natural set S range of generative model), we'll show that random Gaussian measurement has S-REC w.h.p. for sufficient measurements. Then we'll show that the optimum recovers an optimum close to optimum z, if A has S-REC. The second theorem is easy, the first theorem is interesting – it uses Lipschitzness (e.g. REC style – that's easy). Don't say difference of two k sparse is $2k$-sparse – instead, say that the difference of two k-sparse vectors have their difference is far from null-space – this is the correct way to say things, now can replace vectors with "natural vectors". The proof is a covering argument: cover low dimensional sphere, use Lipschitz and to bound the covering number.

Now we talk a bit about adversarial examples in ML. Maximize "catness" of Costis, imperceivable tweak, can fool models. We moved from Costis to manifold of cats, but not staying with natural images. No longer are the examples natural images. If you have a GAN for the domain, project onto the range of generator before classifying – this could be a defense. This is like a de-noising process that projects image on natural manifold. It was also independently proposed. We broke it, and they broke it too – we can write a more complicated optimization problem – find two images which are close in pixel space – with gradient descent. So you can find adversarial examples on the manifold of natural images, then do gradient descent to add on the set. This is called the Robust Manifold Defense – you have to write the paper, then you have to write the blog post, then you have to tweet about the blog post. Why did you have to give a talk then? You have to tweet about the talk!

Approximately correct -blog.

To walk away with:

(a) Differentiable model is cool

(b) Can solve inverse problems with it

(c) Can defend with adversarial examples

Can also combine lasso and the method. Theory of compressed sensing extends to sets generated by generators.

The idea of differentiable compression seems general and interesting.

5 On the (Reduced) Complexity of Markov Decision Processes – Mengdi Wang

A Markov Decision Process is $M = (S, A, P, r, \gamma)$. I want to talk about RL in a very optimization focused way. Most of the one-player games can be cast as MDPs. We have a random walk over a large state space. Let's focus on discrete problem. Here we have a number of states and a number of actions to choose from for each state. We want to find a policy $\pi^* : S \to A$ to maximize total reward. We have a system specified by transition probabilities and also rewards. We want to maximize total reward.

Let's talk about complexity notions:

(a) People care about whether we can solve MDPs in runtime – strongly polynomial time. If we know everything about MDPs, can we compute optimal policy efficiently?

(b) Sample complexity: Given a simulator oracle taking a state-action pair and output a sample transition, the sample complexity of an algorithm is the number of sample transitions needed to find an ϵ-optimal policy.

(c) Regret of RL: Consider the act-in-the-world setting. The regret $R(T)$ is the difference between the maximal reward one can earn in T time steps and the actual reward earned by the algorithm.

Note that the model size of the MDP is given by $|S^2A|$ because we have state-action-state transition probabilities. Let's go through some recent results: First, stochastic primal-dual methods find approximate policies in sublinear time/ sample-complexit: $\tau^4\frac{SA}{\epsilon^2}$. This is actual run-time and sample complexity, where τ is related to mixing-time. That's not good enough because of the τ. We try to do better. In more recent work, given a simulator oracle, randomized value iteration with variance reduction gives an ϵ-optimal policy in near-optimal runtime and sample complexity $\frac{SA}{(1-\gamma)^3\epsilon^2}$ – this matches information theoretic lower bounds. The final result applies in the act-in-the-world setting: We get near optimal regret: $\sqrt{HSAT} + \text{poly}(H)SA$.

Another line of work: Trying to analyze runtime complexity lower bound for computing ϵ-optimal policy. IT is known MDP is polynomial solvable, POMDP is PSPACE hard. For any randomized algorithm, the runtime needed to compute ϵ-good policy withi probability at least 9/10 is $\Omega(|S|^2|A|)$. When MDP is given format of cumulative sums or binary trees, it gets reduced.

We are happy because we have upper and lower bounds for complexity. But there is something annoying here! S is too big! In dynamic programming, the issue is not which algorithm is slightly faster! It's about the state space is too big to deal with! So we don't want to do value iteration or primal dual; we should just do deep learning. It looks like curse of dimensionality is solved via deep learning. But theoretically, there is little we can say about why these methods work. Instead, we take deep learning as a motivation.

What does Deep RL do? It's doing some sort of transformation and dimension reduction. I think of this as a compression process. The observed states are compressed into something low-dimensional. I hope to understand this process. Now, about dimension reduction of complex systems: Suppose we are playing computer games. Suppose I watch a very good player – I want to make some sense out of the observations i make. There are many existing approaches – in terms of estimation, HMM is one way to do dimension reduction of time series. State space aggregation is another thing that's interesting. There are certain tensor models and special cases of MDPs which admit lower complexity algorithms. I want to consider a very basic problem: Just a Markov process. Let's say we just observe a time series, or a discrete valued time series. I don't know much, so let's assume it's Markov. In order to do any sort of dimension reduction, need to measure similarity between observed states. We could say two states are similar if they correspond to similar rewards/ value/

optimal actions. But they are not sufficient for dimensino reduction of Markov decision process. So I propose we consider a different notion of similarity. I say:

$$P(X_{t+1}|X_t = A) \approx P(X_{t+1}|X_t = B)$$

If they correspond to similar futures, we can apply similar controls and perhaps similar value functions. In other words, the rows of transition matrices are similar. This is a linear dependence in row-space of transition probability matrix. Is this notion stronger? It will imply all the previous thoughts, it is more general. Let's think about the low-rank-ness of the transition kernel: E.g. represent high-dimensional random walk with low-dimensional random walk– some low-rank decomposition structure of the matrix: $P = U\tilde{P}V^T$. State aggregation corresponds to a specific decomposition: Soft-state aggregation goes to non-negative decomposition. The compression is very simple. This is a pre-print at ICML 2018. We apply spectral methods to do decomposition of markov decision processes. We collect data and apply SVD truncation. This is like spectral clustering applied to Markov chains. The analysis is somewhat similar to matrix completion. We proved matching bounds. In general one can estimate the compressed space with good efficiency. Back to MDPs. Why do we need to find decomposition of Markov Chains. In DP or RL, people care about value functions. If you have a stationary Markov chain. Transitional reward is r. The average value function can be given by a limit $v = \lim_{T\to\infty} \frac{1}{T+1}(r + Pr + P^2r + \cdots)$. If the Markov chain is low-rank $P = \Phi\tilde{P}\Phi^T$, then value function and invariant state-ddistribution belong to subspace span$(|\Phi_1, \cdots, \Phi_4)$. We use spectral state compression as features for value function. We can write down Bellman equation for average reward in horizon MDP. Then

$$\bar{v}^* + v_s^* = \max_{a\in A}\left\{\sum_{s'\in S} P_a(s,s')v_{s'}^* + r_a(s)\right\}$$

Equivalently, it can be written as a Bellman saddle point problem. Nothing new yet – still high-dimensional. We have spectral state compression (we also have action features). Then we can have the value function belong to span of state features if matrix is truly low-rank. We'll use a linear model for that. We also need a dual variable, which is state-action distribution. This is a vector over state-action pairs. We use a bilinear model - features for states and actions: similar states in similar ways, simliar actions in similar way - goal: find low-dim weights. The optimal policy is just obtained by looking at stationary state-action distribution. After doing that, we can reduce the high-dimensional saddle point problem into a smaller one. This is a low dimensional saddle point problem over parameters for value functions and state-action distributions. In a recent paper, we propose Bilinear π learning. It's doing primal-dual updates on parameters for value (primal variables) – this is very siimlar to TD. All computation and memory is in low-dimensional vectors. Last slide: suppose that this Markov decision process is realizable using state-action features, which is true if the problem is exactly low-rank. Then the π-learning algorithm can learn using $\frac{r_S r_A}{(1-\gamma)^4\epsilon^2}$, where r_A, r_S are the ranks of state and action space. Without the realizability condition, there is a policy approxmation error.

6 Statistically and Computationally Efficient Tensor Completion – Ming Yuan

7 A Mean Field View of the Landscape of Two-Layer Neural Networks – Andrea Montanari

8 Optimal Hypothesis Testing for Stochastic Block Models with Growing Degrees – Zongming Ma

9 Statistical and Computational Guarantees of Mean Field Variational Inference for Community Detection – Harrison Zhou

10 Sparse and Low-Rank Tensor Estimation via Cubic Sketchings – Anru Zhang

11 Robustness Meets Algorithms – Ankur Moitra

12 Outranked: Exploiting Nonlinear Algebraic Structure in Matrix Recovery Problems – Robert Nowak

13 Learning the Learning Rate in Stochastic Gradient Descent – Rachel Ward

14 Rate-Optimal Meta-Learning – Alfred Hero

15 Geometry and Regularization in Nonconvex Statistical Estimation – Yuejie Chi

16 Understanding Generative Adversarial Networks – David Tse

17 The Power of Two Samples for Generative Adversarial Networks – Sewoong Oh

18 R-Local Minimum Analysis and Run-And-Inspect Method for Certain Nonconvex Optimization with