Contents

1 Representation and unsupervised learning: Introduction (Sanjeev Arora)

I'll describe ways in which the current frameworks are not up to the task.

In statistical learning theory, we observe examples $x_1, \ldots, x_n$ with corresponding labels $y_1, \ldots, y_n$. ERM is finding

$$\text{argmin}_C \sum_{i=1}^{n} L_C(x_i, y_i) + R(C)$$

where L_C is the loss function. If $x_1, \ldots, x_n$ are iid distributed according to D and C comes from a class of "low complexity" (Rademacher, etc.) then the test loss is approximately the training loss.

This framework doesn't explain many things about deep nets. The following are mysteries.

1. Generalization in deep nets. See Moritz, Recht [Zha+16].[1] Take image data with *random* labels. Training a neural net with sufficient capacity (M nodes) achieves 0 training error. The neural net can fit even random labels.

   But neural nets trained with backprop still generalizes.

2. Unrelated classes seem to help. ImageNet has 1000 classes, each with 1000 examples. Knowing additional classes helps training for a specific class.

3. Transfer learning. Train a deep net on images. The top 2 layers of the deep net has great features for many other visual task.

[1] `https://arxiv.org/abs/1611.03530`

4. Zero-shot learning: BM Lake, R Salakhutdinov, JB Tenenbaum [LST15].[2] Train on data set with letters from 50 languages. Learn a new character. The program can learn well with one or two examples.

   When you see a character, what does your mind do to remember it? Remember it as distinct strokes.

   In theory papers we ignore the distribution. The statistical learning theorem works for every distribution. But the classifier is very related to the distribution.

   Regularization can incorporate a prior: this is one way to relate with Bayesian approaches.

5. Domain adaptation: If the distributions are very different there are no guarantees. But the distributions can be different: we can train on ImageNet and use for X-ray classification.

   The classifier is very related to the distribution. You can't think of choosing them independently. Clustering, classification with margin are examples. We want a mroe sophisticated language.

   In text, we work with bag-of-words models.

Representation learning is finding a map from a data space (raw pixels, etc.) to a representation space, $x \mapsto h$. This could be a many-to-one map.

We need a language to talk about such maps.

We can use simple models like cluster models, mixtures of Gaussians.

The Bayesian view has been to describe representation learning as distribution learning.The task of learning h is finding the MLE for what generated x. Their notion of representation learning is distribution learning: minimize KL divergence. Why should KL divergence lead to good learning?

We have a loglinear topic model with dynamics which comes up with a vector for each word. How to do this for fMRI data, or small corpora (100 documents, too small to do topic model)? Once you've understood how to understand meanings using Wikipedia, you can do a simple domain adaptation.

We had fMRI data (50k dimensional vector every time step) of people watching a movie (Sherlock). The semantic content of the movie was replaced by human annotation every 5 seconds, 2000 in total. Now we correlate.

Evaluation: Presented a 50k vector, given 5 annotations, choose the correct one.

If we want to classify according to any linear classifier; then we have to preserve all the bits. The human has a semantic model to generate the text; there is a mapping from the semantic representation to the text; we try to find the reverse map.

Some discussions:

- Example: Clustering should take into account the task: how they are going to be used. Representations can be profoundly affect by what you want to do. In clustering there is the concrete task of matching the clustering algorithm with the task. Coming up with tools about how to pick the clustering algorithm is an important problem.

[2]`https://staff.fnwi.uva.nl/t.e.j.mensink/zsl2016/zslpubs/lake15science.pdf`

- Early stopping in NN gives better generalization. This is more general than NN. Which algorithms have this property? You can also do early stopping with gradient descent with kernel methods. In principle you can overfit but you do not. (Note: The surrogate might never reach 0, but the classification error can be 0.)
- Can we phrase GANs as a problem rather than an algorithm?
- Neural networks act a little like our brain. This is fascinating. If you look at details in the brain, it's different. Are we functionally capturing what's happening? Once you have a good implemention, you can steal any brain. Ex. Train a neural net to repeat blood flow and use it as a feature.
- Human-in-the-loop: Practitioners have a problem where they want to achieve human accuracy. If they fail to get training error to 0, make network bigger. Look at the generalization error on the validation set. If that's too high, add regularization. Play with network structure, get more data, etc.

  What algorithms would work well with human-in-the-loop?

  For nonconvex problem, often human creativity is required to get good initialization. To prove anything nontrivial, initialization plays a role.

  See Curves dataset.

Some general topics:

- Nonconvex optimization
- Generalization vs. optimization for nonconvex models.
- Representation power of depth (measure other than number of nodes)
- Models for representations (GANs)
- Differentiable computing. (Real numbers, logic)

Layers are like function calls; recurrent nets are loops. There are primitive operations which you can compose: loop operator, multiply, divide. Kernel methods cannot represent these efficiently. You get power by reusing computation.

Many things you say about neural networks you can say about polynomials. Why don't people use polynomials in ML? There are certain polys you can write as a neural network.

How to interpret "Turing-complete" in the context of NN? Have a read-write memory. It can read things it wrote last time, write things for next time.

2 Apocryphal Results, Recent Results, and Open Problems in Neural Network Representations, (Matus Telgarsky, UIUC)

[3]

[3]We use "Avi" notation:

- Representation? If you use a certain class to represent your predictor, what are the benefits of that choice?
- Apocryphal/subspaces: Ancient results: subspaces can fit every function. Many people can state these results but are not comfortable with the proofs. Ex. Every continuous function can be approximated with polynomials. This can be used as a lemma to prove that a neural net with two layers can fit continuous functions.
- k to k^2 via Δ: There's always a benefit to adding more layers. We will see this by constructing a strange, pathological function. You can use this function to compute many polynomials, parity, etc.
- Applications of Δ
- Algorithmic interlude: I'll comment about algorithmic question: how to determine whether you should add a $k+1$ layer. Even for polynomials this is hard.
- k to $k+1$: Initially I thought this was the kind of technical problem that wastes the time of many people, but I don't even know the hard function looks like—my guesses have failed.
- Rational functions: are also approximable.
- Recurrent networks: there's a trivial construction of a recurrent network that cannot be computed by a deep net, because you can have loops.

  Eduardo Sontag and Siegelmann: Turing-complete model. Empirical work has missed this.
- Generalization (?)
- Two-layer NN

[4]

2.1 Representation

How can we represent our function class?

For example there are 3 ways to represent polynomials.

10 is a small constant.
100 is big-Oh.
1000 is big-Oh with scary constant.

[4] Keep in mind:

1. View theory as leading the way, not just analyzing existing algorithms.
2. We shouldn't lock ourselves in rooms.

1. By degree
$$\bigcup_{r\geq 1}\left\{x\mapsto \sum_{|\alpha|\leq r} c_\alpha x^\alpha : c_\alpha \in \mathbb{R}^{\binom{r+d}{r}}\right\}.$$

2. By kernel
$$\bigcup_{r\geq 1}\bigcup_{n\geq 1}\left\{x\mapsto \sum_{i=1}^n c_i k_r(x,x_i) : (x_i)_{i=1}^n \subseteq \mathbb{R}^{nd}, c\in\mathbb{R}^d\right\}$$
where $k_r(x,z)$ is the inner product of $[1,x_1,\ldots,x_1^2,x_1x_2,\ldots,x_n^r]$ with $[1,z_1,\ldots]$.

3. Sum-product network.

There are different tradeoffs in ease of representability, generalization, etc.

What is the set of low-complexity polys? A sparse function in one representation can be dense in another.

Things we want to prove:

1. Equivalence: One representation can be approximated by another.

2. Inequivalence: There are functions in one class that cannot be approximated by another. Depending on prior, one class really is better.

Compare and contrast programming languages and ML languages.

- In PL, we want human readable/writeable. In ML we want machine learnable human interpretable? (But neural nets are not human interpretable...)
- In PL we have (debugging) tools to diagnose errors. In ML we don't have this, but generalization theory.
- In PL we have Turing completeness. In ML we haven't looked at this. (This isn't just about function approximation—Turing-complete machine can take variable length input and time.)

My conjecture on why NN are successful is because of representation: the things they represent are what we see in nature.

Challenge: Come up with a function class better than neural networks in some aspect that you can quantify.

2.2 Apocryphal/subspaces

Theorem 2.1. *Let RECT* $= \{x\mapsto \mathbb{1}[x\in\prod_{i=1}^d [a_i,b_j]]\}$. *Then*
$$\sup_{f:[0,1]^d\to\mathbb{R}, f\ \textit{continuous}}\ \inf_{g\in\operatorname{span}(RECT)} \|f-g\|_1 = 0.$$

This implies "boosted decision trees" also suffice. (An indicator of a box is an intersection of halfspaces.) The size is $\left(\frac{L}{\varepsilon}\right)^d$. (Good luck doing anything with this.) (Information-theoretically this is the bound; pack bits into a grid.)

Instead of $g \in \text{span}(RECT)$,

- Can take g to be 3-layer neural nets. Such g can compute functions in RECT.
- For g polynomials, two proofs:
    1. use Bernstein polynomials. Control the L^∞ norm: break into 2 cases, close to point and far from point.

       Weierstrass's original proof: convolve with a Gaussian, look at Taylor approximation. $x \mapsto \mathbb{E}_{\tau\sim N(0,\sigma^2)} f(x+\tau)$ is analytic; it has Taylor expansion.

  "Discrete-time vs. continuous-time diffusion."

There is a proof for 2-layer NN that reduces to polynomial approximation. See

- Hornik-Stinchcombe-White [HSW89]
- Cybenko [Cyb89]
- Funahashi [Fun89]

Reverse engineering the size estimate, I get $\left(\frac{L}{\varepsilon}\right)^{2d}$.

I don't think this is a neural net theorem; this is a theorem about a linear subspace $x \mapsto \sum_i c_i \sigma(a_i^T x + b_i)$.

2.3 Separation: k to k^2 via Δ

Theorem 2.2 ([Tel16]).
- *For all $k \geq 1$,*
- *there exists $f : [0,1] \to [0,1]$, f has $10k^2$ layers, 2 nodes per layer, and*
- *for all $g : \mathbb{R} \to \mathbb{R}$, with $\leq k$ layers, $\leq 2^k$ nodes,*

$$\int_{[0,1]} |f(x) - g(x)|^2 \, dx \geq \frac{1}{100}.$$

Why care about number of nodes? All the complexity measures in terms of generalization scale with number of nodes. VC dimension scales as number of layers times parameters.

This works for many classes of functions; we prove it with ReLU. The fact that I'm using the uniform measure is amazing. The ReLU allows me to use it. If I make this statement for the sigmoid, I only know how to prove it with a special hand-crafted measure, and I approximate sigmoids with ReLUs.

If we fix the complexity in the correct matter, each of {shallow networks, deep networks} represents functions that can't be represented by the other.

Open: Find a function easy to write down as a shallow network and is hard to represent by a deep net. You can express it with d blowup in size; the claim is that factor is necessary. I have a candidate function I don't know how to show; it seems to be a coding theory problem. [5]

Proof. Step 1: Find the hard function.

Observation: In a linear representation, the number of bumps is about the number of basis functions.

Define the triangle function

$$\Delta(x) = \begin{cases} 2x, & x \in [0, \frac{1}{2}] \\ 2(1-x), & x \in [\frac{1}{2}, 1] \\ 0, & \text{otherwise.} \end{cases} = \sigma(2\sigma(x) - 4\sigma(x - \frac{1}{2}))$$

where $\sigma(x) = \max\{0, x\}$ is ReLU.

This function has incredible properties in composition: It copies functions, the second copy in reverse. If a function has k bumps, then composing with Δ gives $2k$ bumps.

Thus $\Delta^{(k)}$ has 2^{k-1} bumps.

Take $f = \Delta^{(10k^2)}$.

Step 2: Upper bound the number of bumps for shallow networks.

Define a s-affine function as a piecewise linear function with s pieces.

The sum of s-affine and t-affine function is $s + t - 1$ affine. The composition of s and t-affine functions is at most st affine, and this is tight.

Adding functions slowly increases complexity; composing quickly increases complexity.

Thus a neural network computes $\leq (\text{nodes})^{(\text{layers})}$ affine pieces.

The shallow network can have at most $(2^k)^k$ affine pieces, compared to 2^{10k^2}.

Step 3: L_1 bound.

This is a counting argument. Break the shallow function into pieces based on when they cross $y = \frac{1}{2}$. On average $\Delta^{(10k^2)}$ has 10 times as many oscillations as the shallow function. In any such piece, about half of the triangles are on the wrong side. □

This proof is robust: I can prove separations about polynomials, etc.

2.4 Algorithmic open questions

- Representations reachable by SGD or some other efficient method.

  Rewrite the $k \to k^2$ theorem in terms of a a function that is learnable efficiently by a net with k^2 layers such that the same algorithm for fewer layers means you need exponentially many nodes.

- Polytime algorithm to see if you want to add layers.

  This is a pain. Consider a poly $x_1x_2x_3$, correlation with any other monomial is 0.

  You need structural assumptions on the network to make such a theorem go through.

  Can you make a boosting algorithm on layers? Residual networks are like boosting.

[5] "It's what not what you teach, it's what they learn."

- Polytime test to remove layers.
  - In practice, obtain $((x_i, y_i))_{i=1}^n$.
  - Fit f_0 of high complexity.
  - Now fit f_1 of lower complexity $((x_i, f_0(x_i)))_{i=1}^n$. Claim: this gets lower (generalization?) error.
  - Now fit $((x_i, f_1(x_i)))_{i=1}^n$, etc.

  [6]

  People have prescribed sizes of networks they try.

2.5 k to $k+1$

Find a function in $k+1$ layers which if you try to approximate with k layers you need exponentially many nodes.

Why should this be true? Evidence:

- Eldan-Shamir 2016, $k = 2$. [ES15]
- Boolean circuits: known forever.
- An average-case depth hierarchy theorem for boolean circuits, Servedio et al. 2016 [RST15]

I need a candidate hard function.

Ideas:

1. Another open problem: characterize many hard functions.
2. Kolmogorov 1957. (Superposition theorem[7]) Consider (almost) bijections $[0,1]^d \to [0,1]$—space filling curve. Neural nets can build bijections.

   We need multivariate functions here.

This is much more approachable: Characterize many hard functions.

Given a function (e.g. the sawtooth function), we can write $+1, -1$ when it is above/below the midline.

Define a mapping from sequences to lines. Given $\{-1,1\}^d$, consider the function that is a sequence of triangles going above $(+1)$ or below (-1) the $\frac{1}{2}$ line. A NN **certifies** this sequence if it's equal to this graph.

What is the set of all sequences achievable with $10k^2$ nodes?

- $\Lambda^{(10k^2)}$ certifies the sequence $\underbrace{-1, +1, \ldots}_{2^{10k^2-1}}$.

[6]S Arora: This can be used in a semisupervised setting. Use humans to label a small set; use the trained network f_0 to label more images.

[7]`https://en.wikipedia.org/wiki/Kolmogorov%E2%80%93Arnold_representation_theorem`

- Any sequence of length $\leq k$.
- Can only do exponentially few sequences of length $\geq 100k^4$.

"Find a graphical grammar with what you can do with neural networks." (Rong Ge)

When you go to high dimensions, this problem becomes nasty: how do you even carve up the space and define the language? If I could answer this, I think it would give enough intuition to solve k vs. $k+1$.

2.6 Recurrent networks

- Computational model by Siegelmann-Sontag.

  RNN's are Turing-complete, so there exists $O(1)$ RNN's that is not approximated by any deep net.

  The RNN gets an input, and gives output and whether it's thinking (whether it's done), and passes state. It can churn as long as it wants—it determines when to stop thinking and get the next input.

  To encode a Turing machine, f is a state transition function for the TM; the state contains the tape, head, etc. (you can pack it into a real number)

  This has loops so it has crazy power. How do they work in practice?

  Here is a function that cannot be approximated by a deep net of any size: $f(x,k) = \Delta^{\lg(k)}(x)$. The RNN computes $k/2$ (the "output" bit) and asks if $k \approx 1$ (the "done" bit). RNN's can do loops of unbounded length.

This model of computation is absurdly powerful.

This theorem is trivial; the intelligence part is the model.

Another open question: generalization for RNN's. Can you remove the $\surd$ dependence on length?

I'm most interested in the $k \to k+1$ problem and recurrent nets (learning algorithms, etc.).

3 What Non-Convex Functions Can We Optimize? (Rong Ge)

Many machine learning problems require optimizing a non-convex objective. In this talk we identify a class of non-convex functions where all local minima are also globally optimal. For such functions, stochastic gradient descent efficiently converges to the global optimum . Several interesting problems are known to have this property, and we will in particular show matrix completion has no spurious local minimum.

Based on joing works with Furong Huang, Jason Lee, Chi Jin, Tengyu Ma, Yang Yuan

Two results:

1. Gradient descent with strict saddles

2. Matrix completion: all local optima are global

The title is an open problem, a problem I'm hoping to understand better.
Why nonconvex? Many ML problems are nonconvex, such as

- find the best clustering
- learn the best neural networks
- find communities in social networks.

Convex relaxations sometimes work, but are too slow. In some problems (ex. neural networks) we don't have alternatives to nonconvex optimization.

In theory nonconvex problems are NP-hard in worst case. In practice, SGD and tuning works okay; it finds solutions with reasonable quality.

Why? In practice we are not in worst-case scenario; it is tractable for real-life instances. What properties of real-life instances can we use?

What do we know in convex optimization? A convex function satisfies

$$f\left(\frac{x+y}{2}\right) \le \frac{f(x)+f(y)}{2}.$$

When strongly convex, there is a unique global optimum solution. Find by gradient descent (stochastic, accelerated) or second-order methods—Newton's algorithm (trust region, cubic regularization). They are guaranteed to find the global minimum. The geometry of convexity gives algorithms.

In nonconvex optimization, we hope that we can do something similar. Find clean geometric properties for nonconvex functions that allow efficient solutions.

Relaxations to convexity: Quasiconvexity, Convexity from any point to optimal point. These are not sufficient because they all still have one local/global optimum.

This is not true for many problems because there are many symmetries in the objective function.

- Problem asks for multiple components but the components have no ordering.

  Different permutations are essentially the same solution. However, Gradient descent does not know the objective function has symmetry unless we find some way to give this information to the algorithm.

  A convex combination of 2 equivalent solutions is not optimal. The objective function has several equivalent global optima. There is no way to tell gradient descent to be invariant to the permutation group.

  If we are at a saddle point—ex. take the average of solutions $(x_1, x_2, x_3, \ldots$, $(x_2, x_1, x_3, \ldots)$, $(\frac{x_1+x_2}{2}, \frac{x_1+x_2}{2}, x_3, \ldots)$. The first two coordinates will stay the same.

  (Sometimes people reformulate with one global optima/do a convex relaxation but don't use this in practice because they would just like to use gradient descent.)

Plan:

1. Geometry: identify property of objective function
2. Algorithm: fast algorithm using the geometry
3. Landscape: show specific problems have the geometric structure

3.1 Geometry: Strict saddle functions

Problem: optimize a smooth function $f(x)$ with no constraints.

We are worried about flat regions/saddle points (ex. the origin in $y = x_1^2 - x_2^2$) and local optima. Local search can't get over local optima. So we ask:

For what objective functions are all local optima global optima?

It can be easy to find a local min even with saddle points. [Ge+15] Sometimes all local min are permutations of global min.

A strict saddle function is a function where all points are in one of three cases:

1. near a local minimum (in a strongly-convex ball)
2. near a saddle point (Hessian has negative eigenvalue)
3. has a large gradient

Example: $f(x) = \sum_{i=1}^n x_i^4 - \sum_{i=1}^n x_i^2$. There are 4 equivalent local/global optima, $\pm(1,0),(0,1)$. Saddle points are $(0,0), \pm(1,1), \pm(1,-1)$.

This function can be used to solve PCA.

Other problems that satisfy strict saddle condition:

- eigenvector, generalized eigenvector
- some tensor problems [Ge+15]
- dictionary learning [Sun Qu Wright 15]
- Community/synchronization [Bandeira, Boumal, Voroninski]
- Matrix completion [GLM16] [Bhojanapalli, Neyshabur, Srebro 16]

Open problem: How can we modify objectives to make them satisfy the strict saddle condition? Use re-parametrization, regularization, smoothing/homotopy. In homotopy, smooth by convolving with a Gaussian with large enough variance to make it convex. Then gradually reduce the radius of the Gaussian. Are there functions that have a bad optimization landscape but for which this helps? This works for a particular version of PCA.

What other geometric properties can we use?

1. Degenerate saddle [AG16] "monkey saddle"—3rd order derivative is nonzero (it has 3 descreasing directions so a monkey can put its tail on the saddle),
2. most local minima are good [Cho+15], based off spin-glass models.

For higher-order degeneracies, it can be NP-hard to even decide whether a point is a local min or a saddle.

With probability 1 a random function doesn't have a degenerate saddle point. Perturbing, we get rid of them, but there may be other local min.

What can we do on saddle points?

- Rely on second-order information.
- Find the negative eigenvector, go along that direction.

Problem: requires us to compute the Hessian. d can be large; we have to compute a $d \times d$ matrix. Can we do this without computing the Hessian?

- Frieze, Jerrum, Kannan 94, Learning linear transformations (specific objective function). Considering first and second order conditions you can always find a global optimum
- Nesterov Polyak 03, Cubic regularization (needs 2nd order information)
- [Ge+15] Stochastic gradient converges for strict-saddle function.
- Sun Qu Wright 15. Ridable functions, use trust region algorithm
- Lee Simchowitz Jordan Recht 16. Gradient descent also converges asymptotically. It doesn't bound the number of steps.

All of these assume the Hessian is Lipschitz. Assumptions are comparable.

3.2 Algorithm: Noisy stochastic gradient

Idea: saddle points are not stable. Gradient at X gives no information, but gradients nearby approximates Hessian. Random walk around X should discover the descending direction.

Consider a quadratic function $f(x) = \frac{1}{2}x^T Hx$, where H is the (constant) Hessian. $(0, 0)$ is a saddle point. Assume the axis is aligned, so H is diagonal $\begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \lambda_d \end{pmatrix}$. Assume $\lambda_d \le -\gamma$ and for all i, $|\lambda_i| \le 1$.

Start from $x^0 \sim N(0, \varepsilon I)$. Usually $\eta = 1/\text{smoothness}$; smoothness is a constant here so think of $\eta = \frac{1}{2}$.

$$\begin{aligned} \nabla f(x) &= Hx && (1) \\ x^{(t+1)} &= x^{(t)} - \eta \nabla f(x^{(t)}) && (2) \\ &= x^{(t)} - \eta H x^{(t)} && (3) \\ &= (I - \eta H)x^{(t)} && (4) \\ x^{(t)} &= (I - \eta H)^t x^{(0)}. && (5) \end{aligned}$$

In $\log(d)/(\eta\gamma)$ steps the large direction blows up.

If objective method is quadratic function, this is a power method.

Recent work:

- Agarwal et al 2016, Carmon Duchi Hinder Sidford 2016: find approximate local min in $O(\varepsilon^{-1.75}\ln d)$ iterations. (Cubic regularization is $O(\varepsilon^{-1.5})$, gradient descent (convex) is $O(\varepsilon^{-2})$.) Try to solve the local problem by computing a Hessian-vector product.
- Jin et al 2017, Gradient descent with noise can also achieve $O(\varepsilon^{-2}\operatorname{poly}(\ln d))$ for non-convex functions.

Tight analysis for gradient descent (Jin et al 2017): After adding noise, distance to saddle point is ≈ 1 but correlation with negative direction is $d^{-.5}$. But the Hessian can be heavily distorted. (Ex. If only λ_d is negative, for what points can I escape? As long as there is enough correlation with e_d, i.e. the point is not in a narrow band. For a general function it's much harder to characterize which points can escape; the band of bad points is distorted.)

If a point is stuck at saddle, then moving in a negative-eigenvector directoin, we can escape. A random perturbation whp takes us outside of the band.

We want faster algorithm in stochastic setting, simple accelerated algorithms (preferable stochastic). How to handle constraints?

3.3 Landscape: Matrix completion

Matrix completion: Given an unknown low rank matrix M, observe a random subset of the entries of M. The goal is to recover M. Typical applications include recommender systems and collaborative filtering. Ex. factor a user-movie rating matrix (with missing entries) into user x Genre, Genre x movie matrices.

Previous works:

- Convex relaxation (nuclear norm minimization): optimal sample complexity, but requires solving SDP.
- Non-convex optimization (most analyses is with good initialization)

  Known: initialization is sufficiently good so that the function is convex locally.

We show that even if you don't do initialization, nonconvexity is still not a problem!

Assume M is of the form $M = ZZ^T$. Each entry is observed with probability p independently. Let Ω be the set of observed entries. The objective is

$$\min_X \underbrace{\sum_{(i,j)\in\Omega} (M_{ij} - (XX^T)_{ij})^2}_{f(X)}.$$

This is a degree 4 polynomial.

Theorem 3.1. *Understandard assumptions, with $\gtrsim dr^6$ observed entries, whp all local min are global minima.*

(Optimally, we we can hope for dr.) Note that f is not convex (there are saddle points) and f has no degenerate saddle points.

Warm-up: observe all entries

$$g(X) = \sum_{(i,j)\in[d]\times[d]} (M_{ij} - (XX^T)_{ij})^2.$$

How to go from this to random observation case? The eigenvectors could be very different in the partially observed matrix. Note that linear statements are preserved by subsampling the entries of the matrix

$$p \sum_{(i,j)\in[d]\times[d]} W_{ij} \approx \sum_{(i,j)\in\Omega} W_{ij}$$

and combinations of linear statements are still preserved. ("Linear" means that we have sums $\sum_{(i,j)}$; it does not have to be linear in the W_{ij}.) This allows us to translate the full observation case proof to the partial observation case proof. Use concentration inequalities.

Idea: Analyze the full observation case, redo step 1 by combining linear statements. Then proof generalizes to the partial observation case.

[Park Kyrillidis Caramanis Sanghavi 16] Asymmetrix matrix sensing. [Ge Jin Zheng 17] gives a framework for many matrix problems including asymmetric matrix sensing, completion, robust PCA with much simpler proof.

Open problems:

- How to prove results on optimization landscape?
- Does over-parametrization help? (Hardt Ma Recht 16)
- Is the landscape robust against model misspecification?
- What is the optimization landscape for a random over-complete tensor?

3.3.1 Proof

Rank 1 case for matrix completion: The objective function for the full observation case is $g(x) = \left\|M - xx^T\right\|_F^2$.

Lemma 1: $\nabla g(x) = 0$ implies $\langle x, z\rangle^2 = \|x\|^4$. either x is close to 0 or correlated with z. $\nabla g(x) = 0$ gives the linear property

$$\langle x, \nabla g(x)\rangle = 0 \implies \left\langle x, (zz^T - xx^T)x\right\rangle = 0.$$

Translate the the partial observation case: $\langle x, z\rangle^2 \approx \|x\|^4$. Use concentration to get $\left\langle x, (P_\Omega(zz^T) - P_\Omega(xx^T))x\right\rangle = 0$. Do concentration inequality over all x. (This is why you need at least dr entries.)

Lemma 2: $\nabla^2 g(x) \succeq 0$ implies $\|x\|^2 \geq \frac{1}{3}$.

For the partial observation case, get $\geq \frac{1}{4}$.

Combining these two lemmas, cor. 3: x is local min implies $\langle x, z\rangle^2 \geq \frac{1}{9}$ ($\geq \frac{1}{16}$). More similar steps will show that local min is actually one of $\pm z$.

Matrix sensing:

- Unknown matrix $M^* = (U^*)(U^*)^T$ where M^* is $d \times d$ and U^* is $d \times r$, $r < d$.
- Known: sensing matrix $A_1, A_2, \dots, A_m \in \mathbb{R}^{d\times d}$, $\langle A_i, M^* \rangle = b_i$

  Matrix completion is the special case that the sensing matrices are E_{jk}.
- Objective:

$$\min_U \frac{1}{2m} \sum_{i=1}^m (\langle A_i, UU^T \rangle - b_i)^2. \tag{6}$$

The RIP property:

Definition 3.2: $\{A_1, \dots, A_m\}$ is (r, δ)-RIP if for all M or rank r,

$$\frac{1}{m} \sum_{i=1}^m \langle A_i, M \rangle^2 = (1 \pm \delta) \|M\|_F^2 .$$

Theorem 3.3 (Srebro et al)**.** *If $\{A_i\}$'s are $(4r, \frac{1}{20})$-RIP, then all local optima of* (6) *satisfy $UU^T = M^*$.*

Look at first and second-order optimality coneditions. Instantiate by taking the inner product with a gradient. Instantiate Hessian with some direction.

Intuition: We want to choose $\Delta = U - U^*$.

Problem: U, U^* are only unique up to rotations.

Define $\Delta = U - U^* R$. Let

$$R = \text{argmin}_{R \text{ is rotation}} \|U - U^* R\| .$$

A useful property (which relies on this rotation)

$$\left\|\Delta\Delta^T\right\|_F^2 \le 2 \left\|UU^T - M^*\right\|_F^2 .$$

Rewrite the objective

$$f(M) = \min_{\text{rank}(M)=r} \frac{1}{2m} \sum_{i=1}^m (\langle A_i, M \rangle - b_i)^2.$$

The benefit of this is that this is a quadratic function over M. It has a fixed Hessian. The Hessian is a $d^2 \times d^2$ matrix but we can rewrite it as $f(M) = \frac{1}{2}(M - M^*) : H : (M - M^*)$ where : here means to treat the left and right matrices as vectors and do vector-matrix-vector multiplication like a quadratic form. So

$$M : H : N = \frac{1}{m} \sum_{i=1}^m \langle A_i, M \rangle \langle A_i, N \rangle .$$

Lemma:

$$\langle \nabla f(U), \Delta \rangle = (UU^T - M^*) : H : (U\Delta^T + \Delta U^T) \tag{7}$$

$$\Delta : \nabla^2 f(U) : \Delta = (U\Delta^T + \Delta U^T) : H : (U\Delta^T + \Delta U^T) + 2(UU^T - M^*) : H : \Delta\Delta^T \ge 0. \tag{8}$$

Goal: replace everything here with either $A = UU^T - M^*$ or $B = \Delta\Delta^T$. Replacing $A + B = U\Delta^T + \Delta U^T$,

$$A : H : (A+B) = 0 \tag{9}$$
$$(A+B) : H : (A+B) + 2A : H : B \geq 0 \tag{10}$$
$$B : H : B - 3A : H : A \geq 0 \tag{11}$$
$$\Delta\Delta^T : H : \Delta\Delta^T - 3(UU^T - M^*) : H : (UU^T - M^*) \geq 0. \tag{12}$$

What does this mean? If $H = I$, this is just

$$\left\|\Delta\Delta^T\right\|_F - 3\left\|UU^T - M^*\right\|_F^2$$

If H is RIP, it is close to this.

References

[AG16] Anima Anandkumar and Rong Ge. “Efficient approaches for escaping higher order saddle points in non-convex optimization”. In: *arXiv preprint arXiv:1602.05908* (2016).

[Cho+15] Anna Choromanska et al. “The Loss Surfaces of Multilayer Networks.” In: *AISTATS*. 2015.

[Cyb89] George Cybenko. “Approximation by superpositions of a sigmoidal function”. In: *Mathematics of Control, Signals, and Systems (MCSS)* 2.4 (1989), pp. 303–314.

[ES15] Ronen Eldan and Ohad Shamir. “The Power of Depth for Feedforward Neural Networks”. In: *arXiv preprint arXiv:1512.03965* (2015).

[Fun89] Ken-Ichi Funahashi. “On the approximate realization of continuous mappings by neural networks”. In: *Neural networks* 2.3 (1989), pp. 183–192.

[Ge+15] Rong Ge et al. “Escaping From Saddle Points-Online Stochastic Gradient for Tensor Decomposition.” In: *COLT*. 2015, pp. 797–842.

[GLM16] Rong Ge, Jason D Lee, and Tengyu Ma. “Matrix completion has no spurious local minimum”. In: *Advances in Neural Information Processing Systems*. 2016, pp. 2973–2981.

[HSW89] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators”. In: *Neural networks* 2.5 (1989), pp. 359–366.

[LST15] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. “Human-level concept learning through probabilistic program induction”. In: *Science* 350.6266 (2015), pp. 1332–1338.

[RST15] Benjamin Rossman, Rocco A Servedio, and Li-Yang Tan. “An average-case depth hierarchy theorem for boolean circuits”. In: *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*. IEEE. 2015, pp. 1030–1048.

[Tel16] Matus Telgarsky. "Benefits of depth in neural networks". In: *arXiv preprint arXiv:1602.04485* (2016).

[Zha+16] Chiyuan Zhang et al. "Understanding deep learning requires rethinking generalization". In: *arXiv preprint arXiv:1611.03530* (2016).