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.

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.

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

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

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.

- 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]

- 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.

[3] We use "Avi" notation:
10 is a small constant.
100 is big-Oh.
1000 is big-Oh with scary constant.

- 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.

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.

[4] Keep in mind:

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

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*

$$\sup_{f:[0,1]^d \to \mathbb{R}, f \text{ continuous}} \inf_{g \in \operatorname{span}(RECT)} \|f - g\|_1 = 0.$$

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 $g \in \operatorname{span}(RECT)$,

- 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}.

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

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.

- 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]

[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.

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 $\underbrace{-1,+1,\ldots}_{2^{10k^2-1}}$.
- 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.

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

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 $\sqrt{}$ 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

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

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 $\lambda_d \leq -\gamma$ and for all i, $|\lambda_i| \leq 1$.

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

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

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 $O(\varepsilon^{-2}\,\text{poly}(\ln d))$ 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 $A_1, A_2, \ldots, A_m \in \mathbb{R}^{d\times d}$, $\langle A_i, M^*\rangle = b_i$

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

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

The RIP property:

Definition 3.2: $\{A_1, \ldots, A_m\}$ 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)

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

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:

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

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

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.

4 Topic models: Proof to Practice (Ravi Kannan)

Check out the book "Foundations of Data Science."

Topic Models posit a stochastic generation process for document corpora and devise algorithms to learn the model from real data. Currently, there are two methods of validation:

improved efficiency on benchmark corpora up to billions of words and mathematically proven error and time bounds tested on smaller cases. I will present our recent effort where the two meet. The main new algorithm ingredient is an importance sampling procedure inspired by Randomized Linear Algebra. Whereas known topic models posit a near low-rank data matrix, we start with a new high-rank model which allows for realistic noise. The algorithm empirically performs better to scale than the state of the art.

4.1 Problem

- The input is a corpus of s documents ($> 10^6$). Each is a d-vector of word frequencies ($d > 5000$).
- Assume there are k unknown topics (> 500) unknown topics, and each document is approximately a convex combination of topics.
- Find topics. This is NP-hard in general. Assume there is a generative model ("hidden hand").
- LDA model: For each document, use a Dirichlet distrution to generate a k-vector of topic weights and take that weighted combination of topics to get a probability distribution over words. Now draw words from that distribution to create the document. You only see the frequency of the words generated.
- You can put any prior on the weights; or you may not put a prior at all.

For this talk k is given.

Think of the input as a matrix of words×documents. The output is a words×topics number. Ex post facto, a human can sit down and assign names to the topcs, like "politics". The topics are not named beforehand.

This is unsupervised learning.

Think of the word-topic vectors μ_i as corners of a simplex. The "hidden hand" picks probability vectors for each document, in the simplex. Now generate a document according to the multinomial distribution of those probability vectors. The word-document vectors could be way outside the simplex—there is enormous variance: the number of words in the dictionary could be much larger than the document size.

In a different model, you can get by with just 3 words per document (Anandkumar).

The learning problem:

- Find maximum likelihood estimator of topic matrix.
- Find hidden variables: topic weights vector for each doc, or even the topic assignment of each word in each document.
-

A standard algorithms is to run a Markov chain on k^{sm} states.

There are many applications, and empirical successes in speeding this up, but no proof of fast convergence.

4.2 Provable algorithms

What have people focused on?

- ML, statistics:
  - Model. There are often nice statistical/mathematical underpinnings.
    Ex. The Dirichlet distribution is the conjugate prior to the multinomial.
  - Simple, efficient algorithms: Lloyd's k-means, Expectation-minimization, Gibbs sampling, alternating minimization, Metropolis-Hastings
  - Simple proofs of convergence to local minima. But often there are no proofs of finite sample error, time bounds, or closeness to global optima.
- Theory
  - Prove that if the data was generated by a hidden model, then in poly time, approximate the generating model from data, ideally with widely used or usable algorithms. (Here we abbreviate "with proven poly time and error bounds" by "provable".)
  - Gaussian Mixture Modeling: Dasgupta Schulman; Arora Kannan, Vempala, Wang; Kalai, Moitra, G Valiant
  - Kumar, Kannan; Awashti, Or. Alternating minimzation, Jain, Netrapalli; Ge,...

The agnostic case is harder.

Topic modeling is a case of non-negative matrix factorization. Exact NMF is that we are given $A = BC$ with all nonnegative entries and asked to find B, C.

- Arora, Ge, Kannan, Moitra: Provable algorithms for NMP (under assumptions)
- Arora, Ge, Moitra,...: Provable algorithm for topic modeling (under assumptions)
- Improvements: Recht, Re, Tropp, Bittoff; Gilis. Improvments on NMF algorithms
- Bansal, Bhattacharyya, Kannan: under realistic, empirically verified assumptions
- Bh, Goyal, Kannan, Pani: Provable algorithm for NMF under a realistic noise model (noise swamps data), empirical performance (ICML2016)

  Previous algorithms assume noise for a data point is ε times the data point.
- Bh, Kannan, Simhadri, Dave, Horsala: Provably learning deep topics from a billion tokens.

Provable algorithms for exact NMF: Intuition: if there was one x at each corner, we can find them by one linear program per data point. You can improve this to solving just one convex problem. This is not scalable.

This is not robust to noise—if you only rely on one point, and the point is wrong...

4.3 TSVD

Our model makes the assumptions:

1. Each topic has a set of catchwords.
   - Each catchword has a higher frequency in the topic than in other topics.
   - All catchwords for a topic together have frequency at least a constant, say 0.1.
   - Frequency of each catchwords in a topic (ex. batter, bases, homerun in baseball) is ≥ 1.1 times frequency in any other topic.
   - This replaces the anchor words assumptions of earlier papers.
     If "homerun" is an anchor word for baseball, it only occurs in the baseball topic and it occupies a constant frequency (ex. 0.1 of the words).
2. Each document has a dominant topic whose weight is say ≥ 0.2 (when $k = 100$) and the weight of each other topic in document is ≤ 0.15.
3. Nearly pure documents: for each topic, there is a fraction, say $\frac{1}{10k}$ whose weight on the topic is at least 0.9.

All three assumptions are empirically verified on real data. Provably, under the traditional LDA model with reasonable hyperparameters, a majority of docs have dominant topics and a fraction are nearly pure.

SVD gets a bad rep:

- Latent semantic indexing: Dumais... 1990 propose SVD as denoiser.
- Papadimitriou, Raghavan, Tomaki, Vempala: LSI provably does Topic Modeling when each doc is purely on one topic.
- Unintended consequence: SVD does not help when there are multiple topics per document (folklore)
- Arora, Ge, Moitra, Beyond SVD: LP based algorithm for topic modeling.

SVD and spectral decomposition do scale up; there are good algorithms from numerical analysis. But truncated SVD to rank 10000+ with 10^9+ nonzeros still takes significant computing power.

SVD mixes with negative weights, so solely relying on it is a mismatch to the model.

Here is our unsampled threshold SVD (TSVD) algorithm.

1. Threshold data. (For each word find a threshold and set the frequencies of the word to 0 if below the threshold.)
2. SVD and cluster by dominant topic. (Use k-means clustering on the thresholded topics. There is no convergence in general. We leverage earlier theory: we prove that clustering in SVD projection gives a good start.) (We are not done: we cannot just take the average in each cluster because we need the corners.)

3. Find catchwords (most frequent words in each cluster). (We prove that words with $1-\frac{1}{k}$-fracile frequency in this cluster that is > 1.1 times other clusters are approximately the set of catchwords for the topic.)

4. Find pure documents (docs with highest total frequency of catchwords for each topic among all docs).

Why TSVD?

1. Provable bounds on reconstruction error and time

2. Performs better on many quality metrics

3. Fast in practice and parallelizable.

4.4 Deep topic models

Now I want to scale up to larger k.

The limitation of known models: Suppose we are given $d \times s$ data matrix A, find $d \times k$ topic matrix M for which there is high (nearly max) likelihood. Assume no prior on the topic weight matrix W. We have the problems

1. If $k = d$, $M = I$, $W = A$ (each topic is a single word) is the MLE.

   Zipf's law says that there exists $d_0 = o(d)$, d_0 high frequency words form $1 - \varepsilon$ of data.

   MLE is still trivial if $k \geq d_0$.

2. With large k number of free parameters grows, overfitting

3. Sample complexity grows as high power of k.

We replace the model.

- Assume there are $k_0 \ll k$ basic topics and each deep topic (ex. primary and secondary classification) consists of one or two (or at most q) basic topics.

  This is like a hierarchical model.

  The number k of deep topics satisfies $k_0 \leq k = O(k_0^q)$.

- Set of catchwords for basic topics is defined as before. There is a set of keywords for a deep topic: weighted combination of sets of catchwords comprising its basic topics.

- This solves the problem of k being large: We do topic modeling on k_0, not k topics.

- Matrix formulation: $d \times k$ topic matrix M, $d \times k_0$ $M^{(1)}$ of basic topics,

  $$M = M^{(1)} M^{(2)}$$

  where $M^{(2)}$ has at most q nonzeros per column.

  Take weighted combination of deep topics which are themselves combinations of basic topics.

- The number of free parameters is $dk_0 + k_0 q \ll dk$.

In this model the names on the second level need to be the same.

Note the framework of topic modeling requires a ground truth.

We test on PubMed, Wikipedia, ProductAds, with $k_0 = 2000$, $k = 5K, 10K, 20K, 50K, 100K$.

We do well on coherence (do words found cooccur? for each document take top 5 words, for each pair within here, see how many times they occur together), not so well on complexity.

5 Generalization: FTW or WTF? (Ben Recht)

Despite their massive size, successful deep artificial neural networks can exhibit a remarkably small difference between training and test performance. Conventional wisdom attributes small generalization error either to properties of the model family, or to the regularization techniques used during training. Through extensive systematic experiments, we show how these traditional approaches fail to explain why large neural networks generalize well in practice. Specifically, our experiments establish that state-of-the-art convolutional networks for image classification trained with stochastic gradient methods easily fit a random labeling of the training data. This phenomenon is qualitatively unaffected by explicit regularization, and occurs even if we replace the true images by completely unstructured random noise. We interpret our experimental findings by comparison with traditional linear models, and suggest some possible paths towards understanding how contemporary deep networks generalize.

Joint work with Chiyuan Zhang, Yoshua Bengio, Moritz Hardt, Oriol Vinyals.

Given iid samples $S = \{z_1, \ldots, z_n\}$ from distribution D, find a good predictor function f, $R[f] = \mathbb{E}_z \operatorname{loss}(f; z)$. We minimize empirical risk using SGD, $R_S[f] = \frac{1}{n} \sum_{i=1}^{n} \operatorname{loss}(f; z_i)$.

Generalization error is

$$R[f] - R_S[f],$$

the difference between something I can compute $R_S[f]$ and something I can't.

Fundamental theorem of machine learning:

$$R[f] = (R[f] - R_S[f]) + R_S[f] \tag{13}$$

$$\text{(population risk)} = \text{(generalizatoin error)} + \text{(training error)} \tag{14}$$

Note: Small training error implies risk is close to generalization error. Zero training error does not imply overfitting.

Another formulation (in terms of things I can't compute...)

$$R[f] = (R[f] - R[f_H]) + (R[f_H] - R[f_*]) + R[f_*]$$

The purpose is usually for analysis. But for the first equation I can estimate with another sample.

"If you have ultra-intelligent AI, we would be so far below them in intelligence that we would be like a pet." –Elon Musk.

Right now, we are pets of dumb AI, feeding us confirmation bias. References: The selling of the president (1968, Voting is not intellectual), Amusing ourselves to death (1985), mechanics of doubt (lack of truck in experts and science, 2010), Trump (2016).

Supervised learning means that our models offer no insight. Maxwell's equation predicts crazy things like electricity travels in waves, which is true. Convolution filter banks give weird shit.

(Mark Zuckerberg: facebook only a platform, except it's an instrument for social good.)

Why are we building new "tools for fact checking," before even knowing how we f****d up?

Fix by adding more AI?

If we're just training a cat, we don't care. If we're building. end-to-end trained deep net, we won't know if it mistakes a tractor trailer for a cloud. NN is good for Parlor tricks for NIPS paper, but for more serious things...

Humans following rules: replace with robot. Mathematical modeling: do not replace with robot.

"The technology either exists or can be developed. But then the question is how does it amke sense to deploy it? And this isn't my department." Yann Lecun

"Once the rockets are up, who cares where they come down? That's not my department." cf. Tom Lehrer, 1965

We want trustable, scalable, predictable ML.

Generalization: think about the FDA, pharmaceutical company. They don't believe in generalization. Do from scratch, take another sample. Even with constraints on randomization, people screw it up.

Medicine (biology) vs. cars (mechanical engineering) are different. Biology is much worse understood vs. cars. [8]

Picture from Elements of Statistical Learning: As model complexity goes up,

1. bias goes down,
2. variance goes up,
3. total error goes down and then up; there is an optimal model complexity.

But deep models have huge model complexity! How do you explain that? Models where parameters $p > 20n$ are common.

We don't have right notion of model complexity?

Graph ignores algorithms

How to reduce generalization error?

- model capacity
- regularization (norms, dropout, etc.)
- implicit regularization (early stopping)
- data augmentation (fake data, crops, shifts, etc.). This is important. Include symmetries that don't change the label class. This forces functions to be invariant—You're restricing the model class?

[8] Consider 777 planes, which have been around for 20 years. 2 disappeared, one crashed in SFO seawall, one person died from being hit by the fire truck.

Drugs kill a lot of people and planes kill no one.

All of these are sufficient but by no means necessary!

What happens when I turn off the regularizers? $n = 50000$, $d = 3072$, $k = 10$. Turning regularization off, you get training error to 0. This is effectively early stopping.

- CudaConvNet: 145578 parameters. $p/n = 2.9$ (ratio of parameters to training examples), ≈ 5 layers. Note VC dimension is about parameters times layers. Train loss is $10^{-4} \approx 0$ (logloss), test error is 23%.

  Note we are using float32 on GPUs. Relative error from adding up columns of matrix can be 10^{-4}. For doubles, error can be $\approx 1e-16 \cdot 10^8 = 10^{-8}$. For floats, error can be $\approx 1e-8 \cdot 10^8 = 1$.

  GPU's does dot products nondeterministically; for all purposes it's random. Doing the dot product twice you can get different answers.

  This is a weird dataset. Get something to work on CIFAR-10 then on ImageNet. I have no idea what these pictures have to do with my perceptions...

- CudaConvNet with regularization: train loss 0.34, 10% test error.

  Using regularization prevent going to 0, bump up 5%. Random filters and SVM gets 22%.

  I can tune more or go deeper.

- MicroInception: 1649402, $p/n = 33$, train loss 0, 14% test error. [9]

- ResNet. 2401440. $p/n = 48$, train loss 0, test error.

A community obessed with overfitting sure uses a lot of parameters! –Michael Mahoney.

Are these nets tuned to the data, or work with everything, learning some nearest neighbor hash? Do CIFAR 10 with random labels.

They can memorize any sign pattern, which makes sense because the VC dimension is absurd. (If $p/n > 1$ and I can't fit any sign pattern, something is more wrong.)

Apply the same random permutation to every image: pixels and color channels. The net is "fully convolutional" but loss still goes to 0 quickly. Applying a different random permutation to every image: still goes to 0. In order of how quickly error goes to 0:

- true labels
- shuffled pixels
- gaussian
- random pixels
- random labels

[9] Implementation is just torture. Not everything works. How you do tuning, add things, subtract things is tricky. Much better to have other people do work for you. I spent a lot of time implementing CudaConvNet in tensorflow.

I didn't subtract global mean before applying the convnet. If you don't, it doesn't work, 30% error!

It's basically fitting delta functions. Only L^2 error is unchanged by the shuffling.

Maybe this is just margins?

For Inception, 27 million parameters, $p > 20n$. The dog breeds are accurate but the other pictures are weird... Each row is CPU year!

- Fake data, l2reg/dropout, Train top-1 13.7%, Top-1 23.4%, Train Top-5 2.5%, Test Top-5 6.5%.
- Fake data, no reg, 1.0% train top-5, 9.0% test
- No fake data, l2 reg/dropoout, 0% train, 11.2% test.
- Random labels. Train top-5 0.9% error, test top-5 99.5% error.

80 bytes per image.

Nearest neighbor: compute closest distance. For many distributions, that works. There are counterexamples where it fails. Maybe the marginals fall into that category? Pixel distance doesn't work.

Two main successes of deep learning:

- Taking advantage of where we are in Moore's law (Moore's law is over). Google is obsessed with scale.
- Everyone posts code. (Except: Google writes a lot of papers without publishing code :(.)

Deep learning conflates too many issues. It's easy to write a paper: "this is what's happening in deep nets" It's harder to port a deepnet model from CudaConv to tensorflow.

Linearization principle is useful for everything in deep learning.

Take any crazy thing people are doing in deep learning, does it work with linear models? For GAN's, no.

Avoiding overfitting is hard: this is true for convex case too

$$\min \|y - \Phi x\|^2,$$

here, Φ is $n \times p$, $n < p$. (Deep learning paper: "local min of energy landscape in wide valleys") If $p = 10n$, most eigenvalues are 0. There is a subspace (infinitely wide valley) of minima.

Regularize to leverage structure: sparsity, rank, regularization... do these apply to deep nets?

Why do we generalize when fitting labels exactly? This happens for linear models. If you run SGD to find min norm solution $\min \sum_{i=1}^n (w^T x_i - y_i)^2$, you end up finding $\min_{Xw=y} \|w\|$.

Let $p = \infty$. We're going back to 2003, kernels! This does not necessarily mean overfitting.

Even if you put a margin constraint, I don't know that $\min \|w\|$ generalizes (?).

Minimum norm solution solves $Kc = y$. Fit $Kc = y$ where K is the Gaussian kernel. $60k \times 60k$ solve takes under 3 minutes with 24 cores. For MNIST, no preprocessing, test error is 1.2%! Doing gabor filters first, this because 0.6%. For CIFAR10, get 46% error (about the same as if I shuffle for convnet). CIFAR10 with 2-layer conv-net, 32K random

filters (random wavelet transform), get 16%. Adding L2 regularization gets this down to 14%. This is what MicroInception got!

What's the difference between deep and shallow? Linear system solves as n^3, storage goes as n^2, hard to do with ImageNet. I use scipy.linalg.solve; it's probably doing QR. You don't actually want to run this on your phone. I don't want to have to store the dataset on my phone.

The year when AlexNet won: 2nd place was 74% top-5 error with Fisher kernels.

Overfitting with kernels. In statistically learning, when all points are classified correctly,

$$\mathbb{E}\,(\text{test error}) \leq C\frac{\|f\|_k^2}{n},$$

C suppresses constants and log terms. Key parameter is $M_n = \frac{\|f\|_k}{\sqrt{n}}$. For MNIST, $M_n \geq 0.9$ for pixels, $M_n \geq 1.6$ for Gabor filtered data.

FOr f_1, f_2, f_3 min norm function achieving 0 training loss, achieving 0 and fitting random labels on test set, achieving 0 on random training labels, $\|f_1\| = 3.8e2$, $\frac{\|f_1\|}{n^{\frac{1}{2}}} = 1.6$, $\|f_2\| = 4.9e3 = 13\,\|f_1\|$, $\|f_3\| = 7.8e3 = 21\,\|f_1\|$. Note nearest neighbors gives 3% nearest neigbors on MNIST.

Is statistical learning theory useless?

1. $\sqrt{\frac{d}{n}}$ is useless.
2. Approximation theory doesn't tell us about finite samples.

What can deep learning learn from linear regression?

- min norm is good: is canonical form
- interpolation need not mean overfitting
- stable algorithms lead to stable models.

Stability and robustness are critical for guaranteeing safe, reliable performance of machine learning.

Are neural nets stable? No, there are adversarial examples. What does it mean to perturb an image? What distance?

Opportunities: in 90's, wavelets and nonlinearity worked. Convnets have been sufficient in every task in CV, but are not necessary. If you get rid of them, I feel safer.

1. Nearest neighbor: Do neural nets fall back on neasest neighbors?
2. Explain the effect of class on training on the norm of the solution. (Explain the difference in times of training error going to 0 for true labels, shuffled, random, etc.)
3. Margin bound? Min norm is $(y^T K^* y)^{\frac{1}{2}}$. (Can you actually compute this? K is not well conditioned.)

With the same random seeds, in 10 sgd steps on GPU, get orthogonal models. (On CPU they stay the same.) Generalization error changes with different initialization. Numerical instability is more of an issue than local min.

Where do you chase your error?

6 Secure deep learning (Dawn Song, UC Berkeley)

In this talk, I will discuss recent advances and key questions and challenges at the intersection of Security and Deep Learning, including adversarial examples fooling deep learning systems, and how to achieve better generalization and provide provable guarantee of perfect generalization in certain domains such as neural programing architectures. I will also discuss a new project on AI for data science.

Deep learning has great successes: image recognition, AlphaGo, etc. DeepMind reduces Google cooling cost by 40%.

Deep learning systems are easily fooled. Adding small adversarial perturbation, a image looks identical to the human eye but are misclassified.

Researchers have shown that misclassified examples happen in the real world. Print out adversarial examples, take a photo of the printouts and feeding to the classifier again. The examples remain adversarial.

Compute adversarial perturbation, print out on glasses and give to person to wear. The person is misrecognized.

Adversarial examples can have real-world consequences. Ex. take a STOP sign and perturb adversarially so that it is recognized as a YIELD sign.

Adversarial examples are a prevalent phenomenon in deep learning systems. We consider a weaker threat model when the taret model is unknown (black-box) and other models (generative, etc.).

Ex. completely wrong image captioning.

Adversarial examples can work in a black-box system. clafifai is a startup with an API providing image recognition. A few years ago they won the ImageNet classification. The model, training data, and label set are unknown. Given the completely black-box model, can we generate attacks on this system?

We do black-box attacks based on transferability. Take a white-box model, generate adversarial examples. They transfer to the black-box system even though it's a completely different model.

Are adversarial examples unique to deep learning? I focus on deep nets because they're the highest-performing model, but adversarial examples are no means unique to deep nets. Linear classifiers have adversarial examples.

Theory is not so clear-cut; it's not just an overfit issue.

Another interesting phenomenon: you can compare with adding noise. Nets are relatively robust against noise. The amount of adversarial perturbation needed for misclassification is much larger than amount of random noise needed.

There are 2 types of attacks: non-targeted and targeted attack.

1. Non-targeted: make network misclassify.

2. Targeted: Have a target in mind, ex. misclassify a cat into an elephant.

Non-targeted attack is easy to do. Previous work has done this; we show it can be done also in the black-box model.

Ex. try to misclassify a "rosehip" as a "stufa". The perturbation is larger in the black-box setting. Clarifai misclassifies this as "temple", etc. From water buffalo to rugby ball gives "game", "sport", etc. From broom to jacamar (bird) gives "bird", etc.

Generative models: VAE-GAN has encoder and decoder. Generative model can be used as compression scheme. This can do better than traditional image compression. Attacker's goal is for decompressor to reconstruct a different image from the one the compressor sees. This is a white-box model. Ex. a security camera throws away the original recording and only saves the compressed recording. An attack means the decompressed version is very different.

Adversarial examples for generative models (VAE-GAN), Jernej Kos, Ian Fischer, Dawn Song: Can change all pictures to 0! Here the compressed version is 250-dimensional. Face works as well.

Comment: Nearest neighbor is not resistant.

Adversarial examples on A3C agent on Pong. Jernej Kos and Dawn Song, Delving into adversarial attacks on deep policies.

Input is raw image every frame. Input is current image with 3 frames before it, goal is to predict the best action. We compare adding random noise to each frame and adding adversarial perturbation at pixel level. In both cases learner does poorly, but adversarial perturbation is more effected; the amount needed is tiny.

Can you perturb the opponent's strategy adversarially? The way RL deep nets work is that they don't correctly do alpha-beta or minimax search.

Attacks guided by value function work better. The attacker can just select a subset of frames (10%) to inject adversarial perturbation.

Adversarial examples are prevalent! Questions:

1. How to generate?
2. Why do they exist?
3. How to defend against adversarial examples?

We can formulate as optimization problem. Assume $y = f(x; \theta)$. We want

$$\text{argmax}_{x^*, d(x,x^*) \leq B}\, l(f(x^*; \theta), y).$$

Optimization problem is

$$\text{argmin}_{x^*}\, -l(f(x^*; \theta), y) + \lambda d(x, x^*).$$

Optimization-based methods are expensive, requiring computing gradient many times. Fast gradient sign method,

$$x^* \leftarrow\!\shortmid\, x + B \operatorname{sign} \frac{\partial l}{\partial x}.$$

Many times this is good enough. In general it doesn't give you the best example—it finds examples with larger perturbation. Ex. in deep RL, you have generate examples for every

frame. There you want to do it fast. Parameter space is frozen; compute gradient in input space. Think of this as the dual setting.

When you use B, you get a solution for certain norms.

Why do adversarial examples exist?

- overfitting?
- underfitting?
- excessive linearity?
- difference in training data vs. testing data distribution?

We don't have much theoretical understanding!

Lipschitz constant high? What causes this issue? Nothing in the optimization problem prevents Lipschitz constant? People have not figured out how to make the optimization problem work. Also there are examples where adding a small perturbation, ex. a mustache, should change the classification. Is there some weaker notion of Lipschitz? We want to find adversarial perturbations that are small in certain distance measures. Another way to look at this: for images that are similar, results should be similar as well. Issue: how to capture this similarity? If we have a way to capture this, the problem is halfway solved.

[10]

Picture: decision regions of different models. Show plot in 2 dimensions: adversarial perturbation direction, and randomly picked orthogonal direction. Color based on class label given by network. Along adversarial direction, you need to move much less to be misclassified.

Targeted attacks are generated using ensemble methods. Picture: region where targeted attack succeeds.

RL: Action-decision space is extremely fragmented. Decision boundary is very rugged. 3 actions: move up, down, or don't move.

With random noise retraining and FGSM retraining, decision boundary got more fragmented! Retraining helped to be more robust (reward improved), but decision space is more fragmented. It's not clear it learned the right thing. Maybe this just means it's harder for the attack to do well.

Do you evaluate against the same attack? We tried with different perturbations. [11]

What kind of defense can we develop?

1. Generate adversarial examples to add to training set. Retrain the model and repeat.

   For the most part it is very easy to find new adversarial examples.

[10] All noise you've added is orthogonal to natural manifold of image. Have people studied noise in the manifolds; can we measure Lipschitz relative to the manifold? But the definition of manifold is this encoder-decoder thing?

They are not in the right manifold.

[11] Human decision making probably is also attackable. Ex. priming! In an ad, have a frame of something that primes you; this was made illegal. Magicians do this. Human brains are not resilient.

Human vision system is more robust against these types of attacks.

Modern camouflage have weird pixel patterns.

Sometimes people do see that the learned model, once retrained, is a little more resilient. In general it doesn't work well vs. optimization-based attacks.

2. Input perturbation: Add noise, Gaussian blur. Denoising autoencoder.

   Input perturbation doesn't help. It can make the attack less effective, but the classifier is also less effective.

   If your attacker knows the whole system, it can generate attacks against the whole auto-encoder with classifier network.

3. Regularization: dropout, weight decay

4. architecture changes: other non-linear units

5. ensembles

6. stochastic networks (at inference time): swapout network

7. detection

DataGrad: regularization terms that try to drive gradient around data to 0.
Consensus: there is no good defense.
Two attack methods: optimization based method is stronger attack. People observed: retraining using fast gradient. Set ε to be different values. For FG, when you train with one ε, it's more resilient against that ε, but for other ε's it's not resilient for other ε's. Attack can be more resilient against this but not against optimization-based methods. It does not get better with optimization-based methods. In a relative sense the network gets better (now need to generate with bigger ε, but not that much larger). If you pick ε, when you use small ε it sometimes doesn't get better.
Neural networks are not learning robust/resilient representations.
"We don't know why they work, or why they don't work."

6.1 Adversarial machine learning

Bigger picture: Adversarial machine learning.

1. Learning in the presence of adversaries.

2. Inference time: adversarial example fools learning system

3. Training time: poison training dataset.

   - Ex. Tay Twitter chatbot: People fed it politically incorrect things.
   - Attacker can selectively show learning training data points (even with correct labels) to fool learning system to learn wrong model.

4. Adversarial machine learning particularly important for security critical systems.

Goal: create input-based filtering for vulnerable program that finds exploits.

In security world, this has been widely deployed. How to generate signatures quickly? A signature f, given input x, decides whether x is an exploit, or benign.

Question: how to generate signatures for new attacks?

We have benign inputs (from normal traffic) and labeled attacks and want to learn a classifier. How well can this approach work?

A zero-day attack: you use a web browser with vulnerability. Before this day, no one knew about vulnerability. Attacker was first one to find vulnerability and first one able to create attack. Attacker has full control over what examples the learner sees. Attacker wants to compromise the machines by sending attack inputs. Learner can see all traffic. At the beginning the classifier is not effective. Once machines are compromised, it learns this and gets training inputs. The attacker has control over the examples the learner sees. The attacker can change the inputs. The bitstrings won't look the same. Attacker's goal is to evade the classifier. There is a build-and-break cycle.

We want the learner to learn fast, without too many iterations.

Limits of learning-based signature generation with adversaries (Venkataraman-Blum-Song). There are fundamental lower bounds on how quickly any algorithm can converge to best signature. This is equivalent to the mistake0bound model of online learning.

There exists a sequence of samples such that any deterministic algorithm makes $n \ln r$ mistakes and any randomized algorithm makes $\frac{1}{2} n \ln r$ mistakes. Bounds are almost tight, achieved by Winnow.

Security will be one of the biggest challenges in deploying AI. We can look at security at different levels.

1. Software level: no software vulnerabilities (buffer overgflows). We've studied this in security community: formal verification

2. Learning level: Evaluate systems under adversarial events, not normal evens.

   Compare vs. how we evaluate traditional software.

   In regression testing, run program on normal inputs, prevent normal users from encountering errors. In security testing, run abnormal/adversarial inputs, prevent attackers from finding exploitable errors.

   In software engineering people tend to just do regression testing, and don't think about security testing, and are surprised when their system is attacked.

   In learning system: in regression testing, train on noisy training data, estimate resiliency against noisy training inputs, and test on normal inputs to estimate generalization error. In security testing, train on poisoned training data: estimate resiliency against poisoned training inputs, and test on abnormal/inputs to estimate resiliency against adversarial inputs.

   - We need to reason about complex and symbolic programs. There is decades of work. We have entered the era of formally verified systems: seL4, IronClad, CertiKOS, etc., 100 person-years... This is due to powerful formal verification tools and dedifcated teams.

There are not sufficient tools to reason about non-symbolic programs.

For non-symbolic programs there are no precisely specified properties and goals (ex. image classification, self-driving cars). There is no good understanding of how learning system works. Traditional symbolic reasoning techniques do not apply.

Design new architectures and approaches with stronger generalization and security guarantees!

- Limitation of existing neural architectures. Neural programming architectures. Train neural networks to learn programming tasks (addition, etc.). Neural programming interpreter (Reed-Freitas, ICLR-2016), neural Turing machines, neural GPU, neural RAM, differentiable neural computer.

  The problem with all these approaches: they learn programs that do not generalize well. They learn addition for 10-digit numbers but not 13-digit numbers.

  Besides perception, we need to do symbolic reasoning; we want neural network to have this capability.

  There is no provable guarantees about generalization of learned programs.

  Our approach: making neural programming architectures generalize via recursion. (Jonathon Cai, Richard Shin, Dawn Song.)

  Using recursion, a problem is reduced to sub-problems, with base cases and reduction rules. Learn recursive neural programs!

  This is Turing-complete (not the point).

  Learn faster and generalize better. It can give perfect generalization and we can give a proof of perfect generalization. It is a proof in structured induction setting.

  The neural network has I/O.

  Lessons:

  - program architecture impacts provability
  - similar in program verification for symbolic programs
  - well-designed programs with good architectures are easier to prove properties of
  - arbitrary programs (bad code) are difficult to prove properties of
  - Caution for end-to-end monolithic neural networks, harder to train, generalize, interpret. I predict less of this in the future.
  - Recursive modular neural architectures are easier to reason, prove, generalize.
  - Explore new architectures and approaches enabling strong generalizations and security properties for broader tasks.

  We want to reason about how to combine components. Building large, complex systems require compositional reasoning. Each component provides abstraction.

3. Distributed level.

A new project: AI for data science. We don't have enough human analysts to write scripts to analyze them. The pipeline involves cleansing, feature engineering, model selection and architecture search... Data is growing faster than capacity of data scientists.

Automate data science pipeline while leveraging human feedback.

7 On the Inductive Bias of Dropout (Phil Long, Google)

Abstract: Dropout is a technique for training neural networks in which, before each update of the weights, a random half of the nodes are temporarily removed. Surprisingly, this significantly improves the accuracy, so that dropout is now widely used.

This talk is about the inductive bias of dropout: how does dropout affect what kinds of networks tend to be produced? We build on the work of Wager et al., who showed that in some cases of linear classifiers, using dropout is akin to adding a penalty term to the training loss, somewhat like the Tikhonov regularization used in traditional weight decay.

We begin by focusing on logistic regression without any hidden nodes. We characterize when the dropout-regularized criterion has a unique minimizer, and when the dropout-regularization penalty goes to infinity with the weights. We show that the dropout regularization penalty can be non-monotonic as individual weights increase from 0, and that the dropout regularization penalty cannot be approximated to within any factor by a convex function of the weights.

Next, we consider the case of deep networks with Relu units and quadratic loss. We show that dropout training is insensitive to the scale of the inputs in ways that traditional weight decay is not. Also, in a variety of cases, dropout leads to the use of negative weights, even when trained on data where the output is a simple monotone function of the input. Some experiments with synthetic data support this theoretical analysis.

This is joint work with Dave Helmbold.

Dropout is a method for training neural networks. Drop out half of nodes before each update. It is a simple technique that significantly improves accuracy. It's commonly used but mysterious.

How does dropout affect the preference for breaking ties (inductive bias)?

For example, an algorithm could prefer functions that concentrate in low frequency, functions that change slowly, etc.

7.1 Logistic regression

We studied dropout for linear classification via convex optimization, and deep learning with ReLUs and quadratic loss.

Classify x into $\{-1, 1\}$ using $\text{sign}(w \cdot x)$. Loss is $\ell(w, x, y) = \psi(y(w \cdot x))$ where $\psi(z) = \ln(1 + \exp(-z))$.

We abstract away sampling issues; consider minimizer of loss with respect to the underlying distribution.

What kind of training sets is dropout willing to fit?

Details:

- each feature x_i is replaced with 0 with probability q, and $\frac{x_i}{1-q}$ with probability $1-q$.
- I.e., if components of $r \in \{0, \frac{1}{1-q}\}^n$ are independently 0 with probability q, then x is replaced with $x \odot r$.
- Let $w^*(P,q) = \text{argmin}_w \mathbb{E}_{(x,y)\sim P,r}(\ell(w, x \odot r, y))$ where P is joint probability distribution.

The starting point for this research is Wager, Wang, Liang 2013.

$$\mathop{\mathbb{E}}_{(x,y)\sim P,r}[\ell(y(w \cdot (x \odot r)))] = \mathop{\mathbb{E}}_{(x,y)\sim P}(\ell(y(w \cdot x))) + reg_{D,q}(w)$$

where the 2nd term doesn't involve the class labels. Dropout regularizes somewhat like $\frac{\lambda}{2}\|w\|^2$. It acts like a regularizer. Properties:

- $reg_{D,q}(w) \geq 0$
- $reg_{D,q}(0) = 0$
- if $reg_{D,q}(w) > \ln 2$ then $w \neq w^*$, because 0 would be better.
- $reg_{D,q}(w_1, 0, \ldots, 0) < \ln 2$ for all $D, q \in (0,1)$ and w_1.
- Except for degenerate cases, $\lim_{c\to\infty} reg_{D,q}(cw_1, cw_2, 0, \ldots) = \infty$.
- $reg_{D,q}$ cannot be approximated to within any factor by a convex function.
- But dropout criterion is convex function of w.
- If D is concentrated on $(1,1)$, then $reg_{D,q}$ is not monotone.

Another question: For which data distributions P is w^* close to Bayes optimal?

Proposition 7.1: For any finite domain $X \subseteq \mathbb{R}^n$ and any distribution P with support in X, for all $q \in (0,1)$ the dropout criterion has a unique minimum iff there is not a feature i such that $\mathbb{P}_{(x,y)}(yx_i \geq 0) = 1$ or $\mathbb{P}_{(x,y)}(yx_i \leq 0) = 1$.

When no unique minimum exists, we can do better with larger weights.

As q gets small, effect of regularization fades away. Compare inductive bias of dropout by comparing with other regularizers, $\frac{\lambda}{2}\|w\|^2$. Let

$$err_P(w) = \mathbb{P}_{(x,y)\sim P}[\text{sign}(w \cdot x) \neq y].$$

Characterize P, Q such that $err_P(w^*(P,q))$ is close to optimal, $err_P(v(P,\lambda))$ is far from optimal, and vice versa.

Dropout prefers to focus on a single feature.

7.2 Dropout in deep networks

We consider deep nets that use ReLUs, $\max\{w \cdot x + b, 0\}$. Number of inputs is K, width is n, depth is d, parameters is W, function is f_W, dropout pattern in R, dropout-corrupted function is $f_{W,R}$.

Dropout criterion is $J_P(W) = \mathbb{E}_{(x,y)\sim P}((f_{W,R}(x) - y)^2)$.

Dropout is scale-free in a strong sense. Let Q be obtained from P by rescaling each of $(x_1, \ldots, x_K)$. Then $err_Q(W_Q^*) = err_P(W_P^*)$.

Proof. If we scale down the features, we scale up the weights.

Weight decay is not scale-free.

Say f is non-decreasing if it is non-decreasing in any input. A network of non-decreasing units is non-decreasing. Dropout uses negative weights for simple monotone functions. ($\mathbb{P}((0, \ldots, 0), 0) = \frac{1}{2}$, $\mathbb{P}((1, \ldots, 1), 1) = \frac{1}{2}$, $n \gg K(d + K)$.) With non-negative weights, fitting data leads to $\frac{1}{K}$ variance at output. Using negative weights reduces variance.

7.3 Experiments

Generate synthetic data S. Modify S by rescaling inputs. Train with weight decay and dropout. Dropout produced more negative weights than Tikhonov regularization ($\frac{\lambda}{2} \|w\|^2$).

Q: My intuition is that dropout encourages spreading out weights (different from here). L^2 also encourages spreading. Is dropout somehow between L^2 and L^1? If you have 2 features that do the same thing, dropout prefers you include them both. Dropout prefers features that work together. It doesn't like features that compete. (This arises in practice.) If dropping out can flip... Dropout prefers stability.

Q: Why not used so much? New techniques inspired by dropout replace: batch normalization, residual nets, $\cos(w \cdot x)$.

Does dropout help in generalization? Being reluctant to fit some kind of data helps in generalization.

Adversarial examples: Does dropout make it better/worse?

8 Large-scale Machine Learning: Theory and Practice (Anima Anandkumar)

Large-scale machine learning requires blending computational thinking with statistical frameworks. Designing fast, efficient and distributed learning algorithms with statistical guarantees is an outstanding grand challenge. I will present perspectives from theory and practice. I will demonstrate how spectral optimization can reach the globally optimal solution for many learning problems despite being non-convex. This includes unsupervised learning of latent variable models, training neural networks and reinforcement learning of partially observable Markov decision processes. In practice, tensor methods yield enormous gains both in running times and learning accuracy over traditional methods such as variational inference. I will also discuss the broad challenges in non-convex optimization and the recent progress in this area.

Currently I see a huge gap; my goal is to bridge this. I'll discuss the challenges and some of the solutions. In the second part you'll see recent things Amazon is working on, infrastructure and software support, that will make it easy for researchers to deploy algorithms on the cloud.

Recent successes include image classification, speech recognition, and text processing.

Two important challenges:

1. Guaranteed methods: if you don't understand when algorithms succeed, when to declare success, you require a lot of trial and error.

2. Unsupervised learning: most recent successes are attributable to lots of labeled data.

   Humans do a lot of unsupervised learning.

On the computational side, you can cast most ML problems as an optimization problem. In most cases this is nonconvex. As soon as you have hidden neurons, there are symmetries in the objective function.

In nonconvex optimization we have multiple local optima.

Not all nonconvex problems are created equal. I'll focus on matrix and landscape problems which we understand a lot better. Don't limit to the given objective function; we can change the objective function and still solve the underlying ML problem.

The push in practice is towards local search heuristics but in theory we know it doesn't always succeed. When does it succeed; for what problems?

There are also saddle points, where the gradient vanishes in some directions. How quickly can you find the direction of escape? In practice you do see plateaus (?). This makes it hard to know when to stop.

The number of saddle points explodes as you increase the number of dimensions.

Do guaranteed learning through tensor methods.

- Replace objective function: max likelihood vs. best tensor decomposition

- Preserve global optimum (infinite samples): the method of moments provides a consistent estimator under simple conditions. The set of parameters from maximizing likelihood or fitting moments are identical.

$$\text{argmax}_\theta \, p(x;\theta) = \text{argmin}_\theta \left\| \widehat{T}(x) - T(\theta) \right\|_F^2$$

  where $\widehat{T}(x)$ is the empirical tensor and $T(\theta)$ is low rank tensor based on θ.

- Finding globally optimal tensor decomposition: simple algorithms succeed under mild and natural conditions for many learning problems.

Classical statistics had method of moments, but didn't use it in practice because not robust. They require lots of samples. But now we have lots of samples. Adding noise helps smooth/condition the problem better. There are techniques to overcome robustness.

It's the same reasons that complexity theorists were pessimistic. Min-max bounds are worst case. Under adversarial noise these problems are hard.

8.1 Why tensors?

First and second moments aren't enough to learn everything about the model. First and second moments just give a Gaussian approximation.

Tensors can encode additional information and constraints.

Matrix decomposition helps discover latent factors. But decomposition is not guaranteed to be unique. If you do SVD you only get orthogonal factors. But these factors may not be orthogonal to one another. This is a shortcoming of matrix decomposition, you only have uniqueness under orthogonality.

Ex. decompose students' exam scores into verbal and math component.

If I have 2 sets of matrices (ex. oral and written tests), I have more information. I can hypothesize that the same factors occur between these sets.

Can we then have uniqueness?

We can't solve it all the time. When can we solve it?

Combine matrix slices as a tensor.

$$T = u \otimes v \otimes w + \widetilde{u} \otimes \widetilde{v} \otimes \widetilde{w} \tag{15}$$

$$T_{i_1,i_2,i_3} = u_{i_1} v_{i_2} w_{i_3} + \widetilde{u}_{i_1} \widetilde{v}_{i_2} \widetilde{w}_{i_3}. \tag{16}$$

The problem of shared matrix decomposition is a tensor problem. This gives notion of CP rank. There are also other notions, ex. Tucker decomposition.

Instead of asking for uniqueness of matrix decomposition, I ask for uniqueness of tensor decompositions. We expect conditions to be weaker.

The goal here is to learn the latent factors.

Kruskal 1977: Tensor decomposition is unique when rank 1 pairs are linearly independent. In the matrix case, tensor decomposition is unique when rank one pairs are orthogonal.

Tensor contraction extends the notion of matrix product, $T(u, v, \cdot) = \sum_{i,j} u_i v_j T_{i,j}$.

Simplest method to compute top eigenvector is the power method. Whether you can extend to tensors, we'll see.

Extend matrix products to tensor products, there is a lot of room for improvement in hardware: StridedBatchedGEMM, CuBLAS 8.0. Vector-vector is level 1, matrix-vector is 2, matrix-matrix is 3. "Tensor contractions with extended BLAS kernels on CPU."

We can also extend to asymmetric tensors: Alternate between 2 directions.

Extend matrix power to tensor power method

$$v \mapsto \frac{T(v, v, \cdot)}{\|T(v, v, \cdot)\|}.$$

If tensors are orthogonal the only stable points are the components. If you start from a random points you will converge.

But my goal was to move away from orthogonality. We'll use this as a subroutine for the more general decomposition.

Local search, etc. has no guarantees.

For the general tensor we can't directly run the power method because a component may not even be stationary.

First convert convert tensor into orthogonal one. Find whitening matrix. Then use power method. Recover original components when whitening matrix is invertible. This is exactly when components are linearly independent.

Find W using SVD of matrix slice.

Greedy may not work: best rank 2 and rank 3 may not share components.

(Pseudo-whitening: rescale the components. Condition number and robustness are much worse.)

How robust are these? They are much less robust than matrix decompositions. When $\|E\| < \frac{\lambda_{\min}}{\sqrt{d}}$ power method recovers $\{v_i\}$ up to error $\|E\|$ with linear number of restarts. (Matrix methods don't have $\sqrt{d}$ factor.)

$$\widehat{T} = T + E, \quad T = \sum_i \lambda_i v_i^{\otimes 3}, \quad \|E\| = \max_{x:\|x\|=1} |E(x,x,x)|.$$

Homotopy analysis: we can match SoS guarantees with power method when noise is Gaussian.

How to apply to different problems and do in a computationally efficient manner? We want number of latent factors to be much smaller than the dimension (ex. the vocabulary could be in the hundreds of thousands). We can further reduce complexity by sketching on tensors. We know a lot about matrix sketching, but not much about tensor sketching. We know how to conv-sketch vectors. Sketch using count-sketch. A set of simple operations helps maintain a sketch; do power method on sketch instead. You can keep increasing the order of the tensor! Determine the dimension of the sketch you would like and do operations in this space.

"Guaranteed tensor decomposition via sketching."

One practical application: visual question answering. We use multimodal processing. This is bilinear. We can do count sketch and FFT operations.

Conv-sketch is like random neural net layer?

Individually sketch vectors and take convolution. In frequency domain this is multiplication and addition (take FFT and then componentwise multiplication). Given a bunch of rank 1 tensors we can do this in streaming fashion.

Use as intermediate layer in neural network.

Sketch while preserving higher-order moments.

Current sketching techniques only gives elementwise accuracy. Can you sketch for different norms? What assumptions? Tradeoff between accuracy and length?

8.2 Learning using tensor methods

Given hidden variable and observed variables that are conditionally independent, any order moments have a decomposition,

$$M_r = \mathbb{E}[x_1 \otimes x_2 \otimes x_3] = \sum \lambda_i a_i^{\otimes 3}.$$

We can use this to extract topics from documents. "Two SVDs suffice: spectral decompositions for probabilistic topic modeling and latent Dirichlet allocation."

Looking at cooccurrences of word triplets, get 3rd order moments. Do small adjustments. If you find the decomposition, each will be one of columns of topic-word matrix.

You can also do this for learning communities in social networks, if you hypothesis how people connect is a function of their underlying communities. Mixed membership model: people belong in different proportions to different communities. You can look at common friendships among triples. "A tensor spectral approach to learning mixed membership community models."

Experiments: learn topics from PubMed on Spark, network communities from social network data (we have ground truth, ex. schools people went to—how many of these did we recover?).

"Online tensor methods for training latent variable models." (Local optimum is a problem for variational inference.)

Overcomplete dictionary learning (power method). See Learning sparsely used overcomplete dictionaries, Overcomplete tensor decomposition, Convolutional dictionary learning through tensor factorization.

We can extend to other frameworks. Learn convolutional model. Impose shift-invariant constraints. No guarantee for global optimization: we have problem of symmetry even in rank 1 case. For higher rank it's more challenging.

Use to learn embeddings of sentences.

Pretrained word2vec didn't seem to help because there were too many out-of-domain words. If we had a bigger corpus with in-domain words, it may possibly help.

Local optimal in backpropagation: "few researchers dare to train their models from scratch... small miscalibration of initial weights leads to vanishing or exploding gradients... poor convergence." "Data-dependent initializations of convolutional neural networks," ICLR 2016. There are exponential number (in dimension) of local optimal in backprop.

Moments of neural network,

$$\mathbb{E}[y|x] = f(x) = \left\langle a_2, \sigma(A_1^T x) \right\rangle .$$

"Score function features for discriminative learning: matrix and tensor framework." Hardness of supervised learning comes from lack of knowledge of distribution of input. Score functions

$$S_m(x) := (-1)^m \frac{\nabla^{(m)} p(x)}{p(x)}.$$

For Gaussian x, these are Hermite polynomials.

Reinforcement learning of POMDP's. RL is harder because actions change the state; you want to learn aspects of the environments. Turn into tensor decomposition problem and do explore/exploit using upper confidence bounds to get strong regret guarantees.

Deep reinforcement learning is providing good representation of screen. When you have only partial info, DQN isn't the right framework. Get representations from deep learning, and then use POMDP methods.

8.3 Machine learning at scale

Practical considerations for machine learning:

1. software packages: don't require you to start from scratch, provide primitives.
2. diverse and large datasets: we need good quality and amount.
3. utility computing: gap between academia and industry. AWS credits for research, courses.

Desirable attributes in ML software package:

1. programmability: higher-level functions
2. portability: GPU, CPU, smartphone...
3. efficiency

Amazon's MXNET is completely open-source. This is important. With the other companies, the open-source version does not scale well.

Performance guarantee regardless of which front-end language is used.

Mix of imperative and declarative programming. Imperative programming gives no room to optimize for parallel computing. In declarative programming, build computation graph, so compiler can do optimization. It's essential for performance to know sequence of operations.

Mixed programming paradigm: imperative API and declarative symbolic executor. You can mix paradigms. Embed symbolic expressions into imperative programming. Only mxnet does this. When performance is not the bottleneck, you can do imperative.

If you have memory limitations (don't have room to store all activations) you can trade off memory for computation. This is good for small devices. In forward pass, instead of storing all computations, only select subset to store. In the backward pass, recompute. Determine what you want to store or recompute for complex network architectures. You can save a lot of memory from doing this. Compression: quantization, etc. It can be done in a more principled manner; deep learning has to catch up with compression schemes. Mxnet has hierarchical parameter server.

AWS grand program: `http://aws.amazon.com/grants`

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).