Survey of Overparametrization and Optimization

Jason D. Lee

University of Southern California

September 25, 2019

1. Overparametrization and Architecture Design

2. Geometric Results on Overparametrization
   - Review Non-convex Optimization
   - Non-Algorithmic Results

3. Algorithmic Results
   - Gradient Dynamics: NTK

4. Limitations

Today's Tutorial

Survey of Optimization and Overparametrization in Deep Learning.

- Can think of this tutorial more as a survey of the literature with my own perspectives and opinions.

1. Overparametrization and Architecture Design

2. Geometric Results on Overparametrization
   - Review Non-convex Optimization
   - Non-Algorithmic Results

3. Algorithmic Results
   - Gradient Dynamics: NTK

4. Limitations

Theoretical Challenges: Two Major Hurdles

1. Optimization
   - Non-convex and non-smooth with exponentially many critical points.
2. Statistical
   - Successful Deep Networks are huge with more parameters than samples (overparametrization).

Theoretical Challenges: Two Major Hurdles

1. Optimization
   - Non-convex and non-smooth with exponentially many critical points.
2. Statistical
   - Successful Deep Networks are huge with more parameters than samples (overparametrization).

Two Challenges are Intertwined

Learning = Optimization Error + Statistical Error.

But Optimization and Statistics Cannot Be Decoupled.

- The choice of optimization algorithm affects the statistical performance (generalization error).
- Improving statistical performance (e.g. using regularizers, dropout . . .) changes the algorithm dynamics and landscape.

Non-convexity

- Practical observation: Gradient methods find high quality solutions.

Non-convexity

- Practical observation: Gradient methods find high quality solutions.
- Theoretical Side: Exponentially many local minima in square loss in simple architecture (one neuron with sigmoid activation) [Auer-Herbster-Warmuth].
- Many other hardness based on intersection of halfspaces for realizable models with positive margin [Klivans-Sherstov, Livni-Shalev-Shwartz-Shamir, Neyshabur-Tomioka-Srebro]

Non-convexity

- Practical observation: Gradient methods find high quality solutions.
- Theoretical Side: Exponentially many local minima in square loss in simple architecture (one neuron with sigmoid activation) [Auer-Herbster-Warmuth].
- Many other hardness based on intersection of halfspaces for realizable models with positive margin [Klivans-Sherstov, Livni-Shalev-Shwartz-Shamir, Neyshabur-Tomioka-Srebro]

Question

Why is (stochastic) gradient descent (GD) successful? Or is it just "alchemy"?

Setting

Loss

$$L_n(\theta) = \sum_i \ell(f_\theta(x_i), y_i) + R(\theta),$$

1. $f_\theta(x)$ is the prediction function (neural network)
2. $\ell(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$ or $\ell(\hat{y}, y) = \log(1 + \exp(-\hat{y}y))$.

Algorithm

Gradient Descent algorithm:

$$\theta_{k+1} = \theta_k - \eta_k \nabla L_n(\theta).$$

Stochastic Gradient:

$$\theta_{k+1} = \theta_k - \eta_k \nabla_\theta \ell(f_\theta(x_i), y).$$

Notation

- All parameters is θ. Weights of individual layers are W_l and output layer weights is a.
- Input dimension $x \in \mathbf{R}^d$, width of network is denoted m, and depth L.
- Total number of parameters $\theta \in \mathbf{R}^p$ and sample size n.

1. Overparametrization and Architecture Design

2. Geometric Results on Overparametrization
   - Review Non-convex Optimization
   - Non-Algorithmic Results

3. Algorithmic Results
   - Gradient Dynamics: NTK

4. Limitations

Architecture Design

Designing the Architecture

Goal: Design the architecture so that gradient decent finds good solutions (e.g. no spurious local minimizers)[a].

[a] Livni et al.

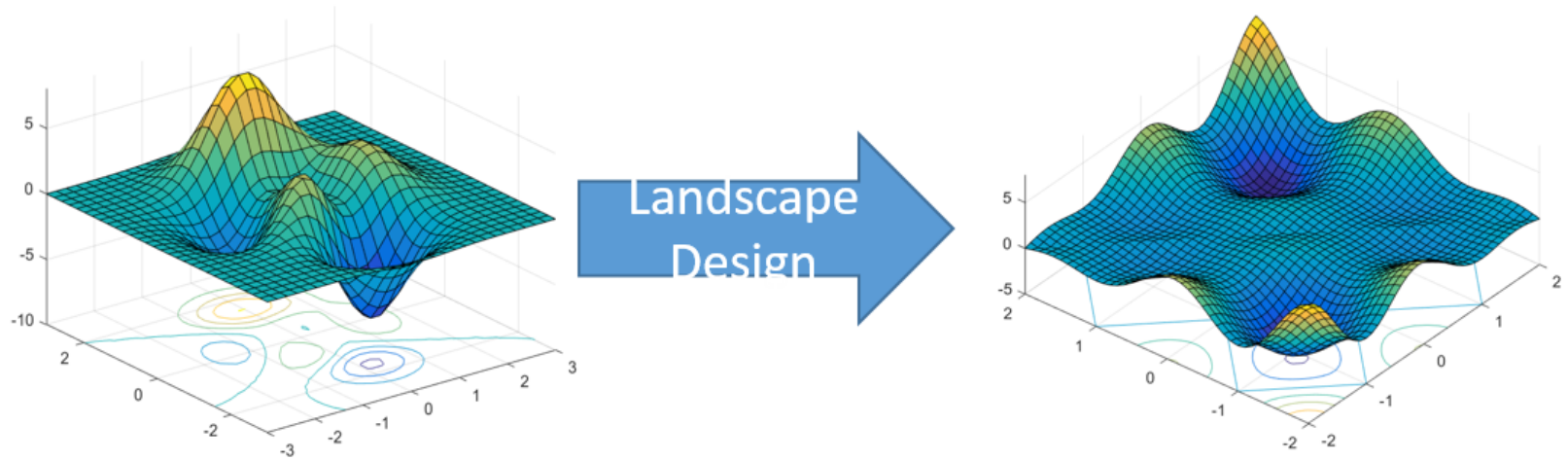


Figure: SGD succeeds on the right loss function, but fails on the left in finding global minima.

Architecture Design: Overparametrization

Empirical Observation: Easier for SGD to optimize larger architectures

Practical Landscape Design - Overparametrization

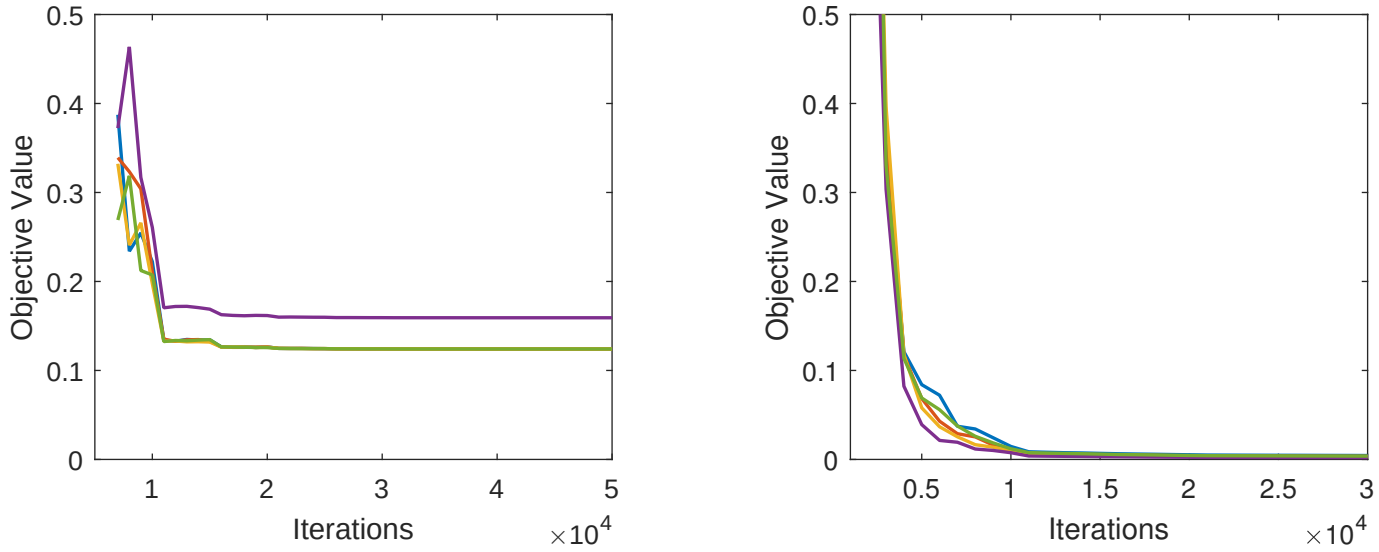


Figure: Experiment first done by Livni-Shalev-Shwartz-Shamir 2014

Overparametrization

Conventional Wisdom on Overparametrization

If SGD is not finding a low training error solution, then fit a more expressive model until the training error is near zero.

Problem

How much over-parametrization do we need to efficiently optimize + generalize?

- Adding parameters increases computational and memory cost.
- Too many parameters may lead to overfitting (???).

How much Overparametrization to Optimize?

Motivating Question

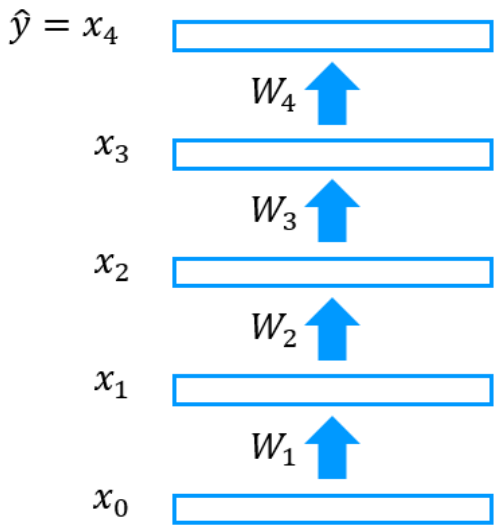
How much overparametrization ensures success of SGD?

- For arbitrary labels, $p \gg n$ is necessary, where p is the number of parameters.
- Can the amount of overparametrization adapt to latent structure in the labels?

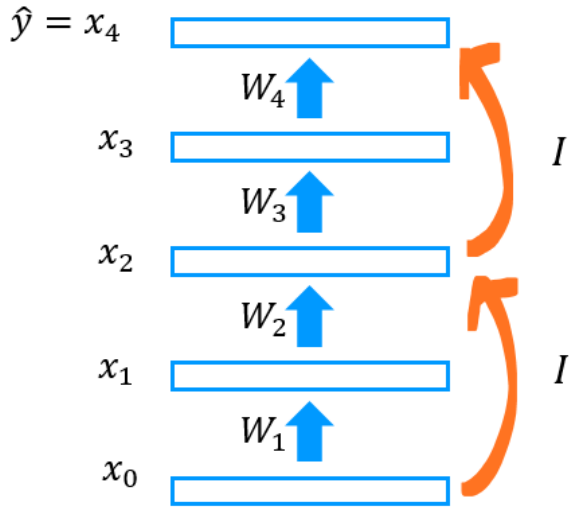
Taxonomy of Overparametrization

- Geometry-based: All local minima are global, All strict local minima are global.
- Local Dynamics (Lazy/ Kernel Regime): Utilize local linear behavior of the network predictions.
- Global Dynamics (Active training, Mean Field): characterized by large changes in the parameter.

Architecture Design: Skip Connections



$$x_i = \sigma(W_i x_{i-1})$$



He et al. 16

$$x_i = \sigma(W_i \sigma(W_{i-1} x_{i-2})) + I \cdot x_{i-2}$$

Skip connections/Resnet avoids gradient vanishing due to depth.

1. Overparametrization and Architecture Design

2. Geometric Results on Overparametrization
   - Review Non-convex Optimization
   - Non-Algorithmic Results

3. Algorithmic Results
   - Gradient Dynamics: NTK

4. Limitations

Overview of Geometric Results

Decompose into two steps:

- Gradient-based algorithms find first-order stationary points or second-order stationary points (Pemantle 92, Ge et al. 15, Lee et al. 16, Jin et al. 17)
- Establish that all first-order/ second-order stationary points (or local minima) are global minimizers.

One-point convexity

Gradient methods can converge even when the function is non-convex.

Quasi-convex

$$\nabla L(\theta)^\top (\theta - \theta^\star) \geq d(\theta, \theta^\star),$$

where d is some distance measure to optimality.

- $d(\theta, \theta^\star) = L(\theta) - L(\theta^\star)$, Quasi convex.
- $d(\theta, \theta^\star) = \eta \|\theta - \theta^\star\|^2 + \beta \|\nabla L(\theta)\|^2$, regularity condition or correlation condition.

Single Neuron/Filter and Local Results

One-point convex

Single neuron/filter models have an even stronger property (with distribution assumptions on x):

$$\nabla_\theta L(\theta)^\top (\theta - \theta^\star) > 0.$$

- $y = \sigma(w^\top x)$ (Candes et al., Kalai et al., Kakade et al., Mei et al., Soltanolkotabi, Goel et al.)
- Single filter: $y = \sum_j \sigma((S_j w)^\top x)$ (Brutzkus and Globerson, Du et al., Goel et al.)
- If $W^\star \approx I$, then two-layer ReLU network is one-point convex in a small region around $W^\star$ (Li and Yuan, Zhong et al.). Resnets have a large basin of attraction around identity.
- Linear Dynamical System (Hardt et al.)

Polyak Condition

In over-parametrized models, we frequently do not know what $\theta^\star$ is because it is non-identifiable.

Polyak Gradient Domination

$$\|\nabla L(\theta)\| \geq L(\theta) - L(\theta^\star).$$

- Local convergence for over-parametrized models (SJL18)
- Global convergence for over-parametrized models (DZPS18, DLLWZ18)
- Linear residual networks (Hardt and Ma) satisfy Polyak condition in a large region around initialization.

Gradient Vanishing

The key is to avoid spurious gradient vanishing.

Gradient Vanishing

The key is to avoid spurious gradient vanishing.

What to do if the gradient is zero?

$$L(\theta) = L(\theta_0) + \underbrace{\nabla L(\theta_0)^\top (\theta - \theta_0)}_{=0} + \frac{1}{2}(\theta - \theta_0)\nabla^2 L(\theta_0)(\theta - \theta_0)$$

Try to find a direction $\theta - \theta_0$ so that $(\theta - \theta_0)\nabla^2 L(\theta_0)(\theta - \theta_0) < 0$.

Algorithms that Avoid Strict Saddle

Using second-order information, it is easy to find SOSP:

Algorithm 1 Second-Order Method (Royer and Wright)

for $k = 0, 1, 2, \ldots$ **do**
  Step 1. (First-Order)
  if $\|\nabla L(\theta_k\| \leq \epsilon_g$ **then**
    Go to Step 2;
  else
    Set $d_k = -\nabla L(\theta_k)$. $\theta_{k+1} = \theta_k + \eta d_k$
  end if
  Step 2. (Second-Order) Compute eigenpair (v_k, λ_k) where $\lambda_k = \lambda_{\min}(\nabla^2 L(\theta_k))$ and $v_k^\top \nabla L(\theta_k) \leq 0$.
  if $\|\nabla L(\theta_k)\| \leq \epsilon_g$ and $\lambda_k \geq -\epsilon_H$ **then**
    Terminate;
  else if $\lambda_k < -\epsilon_H$ **then**
    (Negative Curvature) Set $d_k = v_k$; Set $\theta_{k+1} = \theta_k + \eta d_k$.
  end if
end for

Where will the second-order algorithm terminate?

Second-order algorithm makes progress until both of the following hold:

1. $\nabla L(\theta) = 0$
2. $\nabla^2 L(\theta) \succeq 0$.

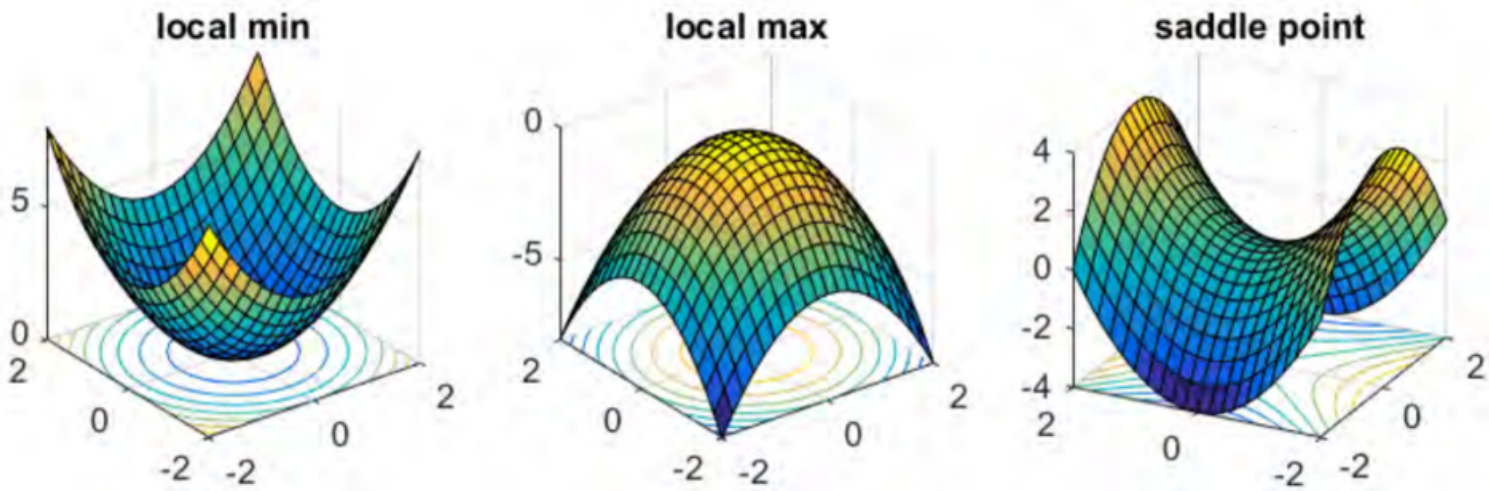
If any of these two conditions are violated, then the algorithm can still make progress. Thus if θ satisfies above two conditions, then we call it *second-order stationary*.

Strict Saddle aka Second-order Stationary Point

A critical point θ^* is second order stationary point (SOSP) if

1. $\nabla L(\theta^*) = 0$,
2. $\nabla^2 L(\theta^*) \succeq 0$.

SOSP $\approx$ local minimum



Detour: Higher-order saddles

There is an obvious generalization to escaping higher-order saddles that requires computing negative eigenvalues of higher-order tensors.

- Third-order saddles can be escaped (Anandkumar and Ge 2016)
- NP-hard to escape 4th order saddles.
- Neural nets of depth L will generally have saddles of order L.
- Escaping second-order stationary points in manifold constrained optimization is the same difficulty as unconstrained. Escaping second-order stationary points in constrained optimization is NP-hard (copositivity testing).

How about Gradient Methods?

Can gradient methods with no access to Hessian avoid saddle-points?

- Typically, algorithms only use gradient access.
- Naively, you may think if gradient vanishes then the algorithm cannot escape since it cannot "access" second-order information.

How about Gradient Methods?

Can gradient methods with no access to Hessian avoid saddle-points?

- Typically, algorithms only use gradient access.
- Naively, you may think if gradient vanishes then the algorithm cannot escape since it cannot "access" second-order information.

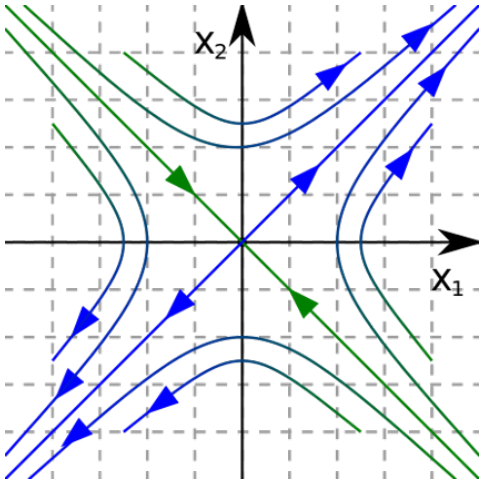
Randomness

The above intuition may hold without randomness, but imagine that $\theta_0 = 0$ and $\nabla L(\theta) = 0$. We run GD from a small perturbation of 0:

$$\theta_{t+1} = (I - \eta H)^t Z.$$

- GD can see second-order information when near saddle-points.

How about Gradient Methods?



Gradient flow diverges from $(0, 0)$ unless initialized on $y = -x$.

This picture completely generalizes to general non-convex functions.

More Intuition

Gradient Descent near a saddle-point is power iteration:

$$f(x) = \frac{1}{2} x^T H x$$

$$x_k = (I - \eta H)^k x_0$$

More Intuition

Gradient Descent near a saddle-point is power iteration:

$$f(x) = \frac{1}{2} x^T H x$$

$$x_k = (I - \eta H)^k x_0$$

- Converges to the saddle point 0 iff x_0 is in the span of the positive eigenvectors.
- As long as there is one negative eigenvector, this set is measure 0.
- Thus for indefinite quadratics, the set of initial conditions that converge to a saddle is measure 0.

Avoiding Saddle-points

Theorem (Pemantle 92, Ge et al. 2015, Lee et al. 2016)

Assume the function f is smooth and coercive ($\lim_{\|x\|\to\infty} \|\nabla f(x)\| = \infty$) , then Gradient Descent with noise finds a point with

$$\|\nabla f(x)\| < \epsilon_g$$
$$\lambda_{\min}(\nabla^2 f(x)) \succeq -\epsilon_H I,$$

in poly$(1/\epsilon_g, 1/\epsilon_H, d)$ steps.
Gradient descent with random initialization asymptotically finds a SOSP.

Gradient-based algorithms find SOSP.

We only need a) gradient non-vanishing or b) Hessian non-negative, so strictly larger set of problems than before.

Why are SOSP interesting?

All SOSP are global minimizers and SGD/GD find the global min:

1. Matrix Completion (GLM16, GJZ17,...)
2. Rank k Approximation (classical)
3. Matrix Sensing (BNS16)
4. Phase Retrieval (SQW16)
5. Orthogonal Tensor Decomposition (AGHKT12,GHJY15)
6. Dictionary Learning (SQW15)
7. Max-cut via Burer Monteiro (BBV16, Montanari 16)
8. Overparametrized Networks with Quadratic Activation (DL18)
9. ReLU network with two neurons (LWL17)
10. ReLU networks via landscape design (GLM18)

What neural net are strict saddle?

Quadratic Activation (Du-Lee, Journee et al., Soltanolkotabi et al.)

$$f(W; x) = \sum_{j=1}^{m} a_j (w_j^\top x)^2$$

with over-parametrization ($m \gtrsim \min(\sqrt{n}, d)$) and any standard loss.

What neural net are strict saddle?

ReLU activation

$$f^*(x) = \sum_{j=1}^{k} \sigma(w_j^{*\top} x)$$

Tons of assumptions:

1. Gaussian x
2. no negative output weights
3. $k \leq d$

Loss function with strict saddle is *complicated*. Essentially the loss encodes tensor decomposition.

More strict saddle

- Two-neuron with orthogonal weights (Luo et al.) proved using extraordinarily painful trigonometry.
- One convolutional filter with non-overlapping patches (Brutzkus and Globerson).

1. Overparametrization and Architecture Design

2. Geometric Results on Overparametrization
   - Review Non-convex Optimization
   - Non-Algorithmic Results

3. Algorithmic Results
   - Gradient Dynamics: NTK

4. Limitations

Non-linear Least Squares (NNLS) Perspective

Folklore

Optimization is "easy" when parameters $>$ sample size.

View the loss as a NNLS:

$$\sum_{i=1}^{n}(f_i(\theta) - y_i)^2 \text{ and } f_i(\theta) = f_\theta(x_i) = \text{prediction with param } \theta$$

Stationary Points of NNLS

- Jacobian $J \in \mathbf{R}^{p \times n}$ has columns $\nabla_\theta f_i(\theta)$.
- Let the error $r_i = f_i(\theta) - y_i$.

The stationarity condition is

$$J(\theta) r(\theta) = 0.$$

J is a tall matrix when over-parametrized, so at "most" points $\sigma_{\min}(J) > 0$.

NNLS continued

Imagine that magically you found a critical point with $\sigma_{\min}(J) > 0$.
Then

$$\|J(\theta)r(\theta)\| \leq \epsilon \Longrightarrow \|r(\theta)\| \leq \frac{\epsilon}{\sigma_{\min}(J)},$$

and thus globally optimal!

Takeaway: If you can find a critical point (which GD/SGD do) and ensure J is full rank, then it is a global optimum.

Other losses

Consider

$$\sum_i \ell(f_i(x), y_i).$$

Critcial points have the form

$$J(\theta)r(\theta) = 0 \text{ and } r_i = \ell'(f_i(\theta), y_i).$$

and so

$$\|r(\theta)\| \leq \frac{\epsilon}{\sigma_{\min}(J)}.$$

For almost all commonly used losses, $\ell(z) \lesssim \ell'(z)$ including cross-entropy.

NNLS (continued)

Question

How to find non-degenerate critical points????

NNLS (continued)

Question

How to find non-degenerate critical points????

Short answer*: No one knows.

NNLS (continued)

Question

How to find non-degenerate critical points????

Short answer*: No one knows.

Nuanced answer: For almost all θ, $J(\theta)$ is full rank when over-parametrized. Thus "almost all" critical points are global minima.

Several Attempts

- Strategy 1: Auxiliary randomness ω , so that $J(\theta, \omega)$ is full rank even when θ depends on the data (Soudry-Carmon). The guarantees **suggest** that SGD with auxilary randomness can find a global minimum.
- Strategy 2: Pretend it is independent (Kawaguchi)
- Strategy 3: Punt on the dependence. Theorems say "Almost all critical points are global" (Nguyen and Hein, Nouiehed and Razaviyayn)

Geometric Viewpoint

Question

What do these results have in common?

Our goal is to minimize $L(f) = \|f - y\|^2$.

- Imagine that you are at f_0 which is non optimal. Due to convexity, $-(f - y)$ is a first-order descent direction.
- Parameter space is $f_\theta(x)$, so let's say $f_{\theta_0} = f_0$. For θ to "mimick" the descent direction, we need

$$J_f(\theta_0)(\theta - \theta_0) = y - f.$$

Inverse Function Theorem (Informal)

- What if J_f is zero? Then we can try to solve $\nabla^2 f(\theta_0)[(\theta - \theta_0)^{\otimes 2}] = -(f - y)$. This will give a second-order descent direction, and allow us to escape all SOSP.
- And so forth: If we can solve $\nabla^k f(\theta_0)[(\theta - \theta_0)^{\otimes k}] = y - f$, this will allow us to escape a k^{th} order saddle.
- Since we do not know $y - f$, we just compute the minimal eigenvector to find such a direction.

Non-Algorithmic No Spurious Local Minima

No Spurious Local Minima (Nouihed and Razaviyayn)

Fundamentally, if the map $f(B_{\theta_0})$ is **locally onto**, then there exists an escape direction.

This does not mean you can efficiently *find* the direction (e.g. 4th order and above). Contrast this to the strict saddle definition.

Relation to overparametrization (Informal):

- $y \in \mathbb{R}^n$, so we need at least $\dim(\theta) = p \geq n$.
- Imagine if you had a two-layer net $f(x) = a^\top \sigma(Wx)$, and the hidden layer is super wide $m \geq n$. Then as long as W is full rank, can only treat a as the variable and solve $\nabla_a f(a_0, W_0)[a - a_0, 0] = y - f$.
  Thus if W is fixed, all critical points in a are global minima.
- Now imagine that W is also a variable. The only potential issue is if $\sigma(WX)$ is a rank degenerate matrix.
- Thus imagine that if (a, W) is a local minimum, where the error is not zero. We can make an infinitesmal perturbation to W to make it full rank. Then a perturbation to a to move in the direction of $y - f$ to escape. Thus there are no spurious local minima.

Papers that are of this "flavor": Poston et al. , Yu, Nguyen and Hein, Nouiehed and Razaviyayn, Haeffele and Vidal, Venturi et al..

Theorem (First form)

Assume that $f(x) = W_L \sigma(W_{L-1} \dots W_1 x)$. There is a layer with width $m_l > n$ and $m_l \geq m_{l+1} \geq m_{l+2} \geq \dots \geq m_L$.
Then almost all local minimizers are global.

Theorem (Second form)

Similar assumptions as above.
There exists a path with non-increasing loss value from every parameter θ to a global min $\theta^\star$. This implies that every strict local minimizer is a global min.

- Generally require you to have $m \geq n$ (at least one layer that is very wide).
- Non-algorithmic and do not have any implications for SGD finding a global minimum (higher-order saddles etc.)

Connection to Frank-Wolfe

In Frank-Wolfe or Gradient Boosting, the goal is to find a search direction that is correlated with the residual.
The direction we want to go in is $f_i(\theta) - y_i$. If the weak classifier is a single neuron, then a two-layer classifier is the boosted version (same as Barron's greedy algorithm):

$$f(x) = \sum_{j=1}^{m} a_j \underbrace{\sigma(w_j^\top x)}_{\text{weak classifier}} .$$

At every step try to find:

$$\sigma(w^\top x_i) = f_i(\theta) - y_i.$$

Similar to GD

Frank-Wolfe basically introduces a neuron at zero and does a local search step on the parameter.
Has the same issues:

- If $\sigma(w^\top x)$ can make first-order progress (meaning strictly positively correlated with $f - y$), then GD will find this.
- Otherwise need to find a direction of higher-order correlation with $f - y$, and this is likely hard.
- Notable exceptions: quadratic activation requires eigenvector (Livni et al.) and monomial activation requires tensor eigenvalue.

1. Overparametrization and Architecture Design

2. Geometric Results on Overparametrization
   - Review Non-convex Optimization
   - Non-Algorithmic Results

3. Algorithmic Results
   - Gradient Dynamics: NTK

4. Limitations

How to get Algorithmic result?

Most NN are not strict saddle, and the "all local are global" style results have no algorithmic implications.
What are the few cases we do have algorithmic results?

- Optimizing a single layer (Random Features).
- Local results (Polyak condition).

How to get Algorithmic result?

Most NN are not strict saddle, and the "all local are global" style results have no algorithmic implications.
What are the few cases we do have algorithmic results?

- Optimizing a single layer (Random Features).
- Local results (Polyak condition).

Let's try to use these two building blocks to get algorithmic results.

Random Features Review

Consider functions of the form

$$f(x) = \int \phi(x;\theta)c(\theta)d\omega(\theta), \quad \sup_{\theta} c(\theta) < \infty$$

Rahimi and Recht showed that this induces an RKHS with kernel

$$K_\phi(x, x') = \mathbb{E}_\omega[\phi(x;\theta)^\top \phi(x';\theta)].$$

Relation to Neural Nets (Warm-up)

Two-layer Net

$f_\theta(x) = \sum_{j=1}^m a_j \sigma(w_j^\top x)$, and imagine if $m \to \infty$.

Define the measure $c\left(\frac{w_j}{\|w_j\|}\right) \propto |a_j| \|w_j\|_2$, then

$$f_c(x) = \int \phi(x;\theta) c(\theta) d\omega(\theta).$$

If m is large enough, any function of the form $f(x) = \int \phi(x;\theta) c(\theta) d\omega(\theta)$ can be approximated by a two-layer network.

Theorem

Let $f = \int \phi(x;\theta)c(\theta)d\omega(\theta)$, *then there is a function of the form* $\hat{f}(x) = \sum_{j=1}^{m} a_j \phi(x;\theta_j)$,

$$\|\hat{f} - f\| \lesssim \frac{\|c\|_\infty}{\sqrt{m}}.$$

- $\mathrm{span}(\{\phi(\cdot,\theta_j)\})$ *is dense in* $H(K_\phi)$

Function classes Learnable via SGD

Proof Strategy (Andoni et al., Daniely):

Function classes Learnable via SGD

Proof Strategy (Andoni et al., Daniely):

1. Assume the target $f^\star \in H(K_\phi)$ or approximable by $H(K_\phi)$ up to tolerance.

Function classes Learnable via SGD

Proof Strategy (Andoni et al., Daniely):

1. Assume the target $f^\star \in H(K_\phi)$ or approximable by $H(K_\phi)$ up to tolerance.
2. Show that SGD learns something as competitive as the best in $H(K_\phi)$.

Step 1

1. Write $K(x, y) = g(\rho) = \sum c_i \rho^i$.

Step 1

1. Write $K(x, y) = g(\rho) = \sum c_i \rho^i$.
2. Thus $\phi(x)_i = \sqrt{c_i} x^i$ is a feature map.

Step 1

1. Write $K(x, y) = g(\rho) = \sum c_i \rho^i$.
2. Thus $\phi(x)_i = \sqrt{c_i} x^i$ is a feature map.
3. Using this, we can write $p(x) = \sum p_j x^j = \langle w, \phi(x) \rangle$ for $w_j = p_j / \sqrt{c_j}$.

Step 1

1. Write $K(x, y) = g(\rho) = \sum c_i \rho^i$.
2. Thus $\phi(x)_i = \sqrt{c_i} x^i$ is a feature map.
3. Using this, we can write $p(x) = \sum p_j x^j = \langle w, \phi(x) \rangle$ for $w_j = p_j / \sqrt{c_j}$.
4. Thus if c_j decay quickly, then $\|w\|_2$ won't be too huge. RKHS norm and sample complexity is governed by $\|w\|_2$.

Step 1

1. Write $K(x, y) = g(\rho) = \sum c_i \rho^i$.
2. Thus $\phi(x)_i = \sqrt{c_i} x^i$ is a feature map.
3. Using this, we can write $p(x) = \sum p_j x^j = \langle w, \phi(x) \rangle$ for $w_j = p_j / \sqrt{c_j}$.
4. Thus if c_j decay quickly, then $\|w\|_2$ won't be too huge. RKHS norm and sample complexity is governed by $\|w\|_2$.

Conclusion: Polynomials and some other simple functions are in the RKHS.

Step 2

Restrict to two-layer.

Optimizing only output layer

Consider $f_\theta(x) = a^\top \sigma(Wx)$, and we only optimize over a. This is a **convex problem**.

Step 2

Restrict to two-layer.

Optimizing only output layer

Consider $f_\theta(x) = a^\top \sigma(Wx)$, and we only optimize over a. This is a **convex problem**.

Algorithm: Initialize w_j uniform over the sphere, then compute

$$\hat{f}(x) = \arg\min_a \sum_i L(f_{a,w}(x_i), y_i).$$

Restrict to two-layer.

Optimizing only output layer

Consider $f_\theta(x) = a^\top \sigma(Wx)$, and we only optimize over a. This is a **convex problem**.

Algorithm: Initialize w_j uniform over the sphere, then compute

$$\hat{f}(x) = \arg\min_a \sum_i L(f_{a,w}(x_i), y_i).$$

Guarantee (via Rahimi-Recht):

$$\|\hat{f} - f\| \lesssim \frac{\|f\|}{\sqrt{m}}.$$

Both layers

If we optimize both layers, the optimization is non-convex.

Morally, this non-convexity is harmless. We only need to show that optimizing w_j does not hurt!

Both layers

If we optimize both layers, the optimization is non-convex.

Morally, this non-convexity is harmless. We only need to show that optimizing w_j does not hurt!

Strategy:

- Initialize $a_j \approx 0$ and $\|w_j\| = O(1)$,

$$\nabla_{a_j} L(\theta) = \sigma(w_j x) \text{ and } \nabla_{w_j} L(\theta) = a_j \sigma'(w_j x) x$$

Both layers

If we optimize both layers, the optimization is non-convex.

Morally, this non-convexity is harmless. We only need to show that optimizing w_j does not hurt!

Strategy:

- Initialize $a_j \approx 0$ and $\|w_j\| = O(1)$,

$$\nabla_{a_j} L(\theta) = \sigma(w_j x) \text{ and } \nabla_{w_j} L(\theta) = a_j \sigma'(w_j x) x$$

- $\nabla_{w_j} L(\theta) \approx 0$, so the w_j do not move under SGD.

Both layers

If we optimize both layers, the optimization is non-convex.

Morally, this non-convexity is harmless. We only need to show that optimizing w_j does not hurt!

Strategy:

- Initialize $a_j \approx 0$ and $\|w_j\| = O(1)$,

$$\nabla_{a_j} L(\theta) = \sigma(w_j x) \text{ and } \nabla_{w_j} L(\theta) = a_j \sigma'(w_j x) x$$

- $\nabla_{w_j} L(\theta) \approx 0$, so the w_j do not move under SGD.
- The a_j converge quickly to their global optimum w.r.t. $w_j = w_j^0$, since $w_j \approx w_j^0$ for all time.

Theorem

Fix a target function $f^\star$ and let $m \gtrsim \|f^\star\|_{\mathcal{H}}^2$. Initialize the network so that $|a_j| \ll \|w_j\|_2$. Then the learned network

$$\|\hat{f} - f^\star\| \lesssim \frac{\|f\|_H}{\sqrt{m}}.$$

- Roughly is what Daniely and Andoni et al. are doing.

Deeper Networks

The idea is similar:

$$f_\theta(x) = \sum a_j \sigma(w_j^{L\top} x^{L-1})$$

- Define $\phi(x; \theta_j) = \sigma(w(0)_j^{L\top} x^{L-1})$, which induces some K_ϕ.
- SGD on just a is simply training random feature scheme for this deep kernel K_ϕ.
- Initialization is special in that the a moves much more than w during training, so kernel is almost stationary.

1. Overparametrization and Architecture Design

2. Geometric Results on Overparametrization
   - Review Non-convex Optimization
   - Non-Algorithmic Results

3. Algorithmic Results
   - Gradient Dynamics: NTK

4. Limitations

Other Induced Kernels

Recap

$$f_\theta(x) = \sum_j a_j \sigma(w_j^\top x)$$

If only a_j changes, then get the kernel

$$K(x, x') = \mathbb{E}[\sigma(w_j^\top x)\sigma(w_j^\top x')].$$

- Somewhat unsatisfying. The non-convexity is all in w_j and it is being fixed throughout the dynamics.

Other Induced Kernels

Recap

$$f_\theta(x) = \sum_j a_j \sigma(w_j^\top x)$$

If only a_j changes, then get the kernel

$$K(x, x') = \mathbb{E}[\sigma(w_j^\top x)\sigma(w_j^\top x')].$$

- Somewhat unsatisfying. The non-convexity is all in w_j and it is being fixed throughout the dynamics.

All weights moving

More general viewpoint. Consider if both a and w move:

$$f_\theta(x) \approx f_0(x) + \nabla_\theta f_\theta(x)^\top (\theta - \theta_0) + O(\|\theta - \theta_0\|^2).$$

Neural Tangent Kernel

Backup and consider $f_\theta(\cdot)$ is any nonlinear function.

$$f_\theta(x) \approx \underbrace{f_0(x)}_{\approx 0} + \nabla_\theta f_\theta(x)^\top (\theta - \theta_0) + O(\|\theta - \theta_0\|^2),$$

Neural Tangent Kernel

Backup and consider $f_\theta(\cdot)$ is any nonlinear function.

$$f_\theta(x) \approx \underbrace{f_0(x)}_{\approx 0} + \nabla_\theta f_\theta(x)^\top (\theta - \theta_0) + O(\|\theta - \theta_0\|^2),$$

Assumptions:

- Second order term is "negligible".
- f_0 is negligible, which can be argued using initialization+overparametrization.

References:

- Kernel Viewpoint: Jacot et al., (Du et al.)[2], (Arora et al.) [2], Chizat and Bach, Lee et al., E et al.
- Pseudo-network: Li and Liang, (Allen-Zhu et al.)$^{\gtrsim 5}$, Zou et al.

Tangent Kernel

Under these assumptions,

$$f_\theta(x) \approx \hat{f}_\theta(x) = (\theta - \theta_0)^\top \nabla_\theta f(\theta_0)$$

- This is a **linear** classifier in θ.
- Feature representation is $\phi(x; \theta_0) = \nabla_\theta f(\theta_0)$.

Tangent Kernel

Under these assumptions,

$$f_\theta(x) \approx \hat{f}_\theta(x) = (\theta - \theta_0)^\top \nabla_\theta f(\theta_0)$$

- This is a **linear** classifier in θ.
- Feature representation is $\phi(x; \theta_0) = \nabla_\theta f(\theta_0)$.

Corresponds to using the kernel

$$K = \nabla f(\theta_0)^\top \nabla f(\theta_0).$$

What is this kernel?

Neural Tangent Kernel (NTK)

$$K = \sum_{l=1}^{L+1} \alpha_l K_l \text{ and } K_l = \nabla_{W_l} f(\theta_0)^\top \nabla_{W_l} f(\theta_0)$$

Two-layer

$$K_1 = \sum_j a_j^2 \sigma'(w_j^\top x)\sigma'(w_j^\top x')x^\top x' \text{ and } K_2 = \sum_j \sigma(w_j^\top x)\sigma(w_j^\top x')$$

Kernel is initialization dependent

$$K_1 = \sum_j a_j^2 \sigma'(w_j^\top x)\sigma'(w_j^\top x')x^\top x' \text{ and } K_2 = \sum_j \sigma(w_j^\top x)\sigma(w_j^\top x')$$

so how a, w is initialized matters a lot.

- Imagine $\|w_j\|^2 = 1/d$ and $|a_j|^2 = 1/m$, then only $K = K_2$ matters (Daniely, Rahimi-Recht).
- "NTK parametrization": $f_\theta(x) = \frac{1}{\sqrt{m}} \sum_j a_j \sigma(w_j x)$, and $|a_j| = O(1)$, $\|w\| = O(1)$, then

$$K = K_1 + K_2.$$

  This is what is done in Jacot et al., Du et al, Chizat & Bach
- Li and Liang consider when $|a_j| = O(1)$ is fixed, and only train w,

$$K = K_1.$$

Initialization and LR

Through different initialization/ parametrization/layerwise learning rate, you can get

$$K = \sum_{l=1}^{L+1} \alpha_l K_l \text{ and } K_l = \nabla_{W_l} f(\theta_0)^\top \nabla_{W_l} f(\theta_0)$$

- NTK should be thought of as this family of kernels.
- Rahimi-Recht, Daniely studied the special case where only K_2 matters and the other terms disappear.

Infinite-width

For theoretical analysis, it is convenient to look at infinite width to remove the randomness from initialization.

Infinite-width

Initialize $a_j \sim N(0, s_a^2/m)$ and $w_j \sim N(0, s_w^2 I/m)$.
Then

$$K_1 = s_a^2 E_w[\sigma'(w_j^\top x)\sigma'(w_j^\top x')x^\top x']$$
$$K_2 = s_w^2 E_w[\sigma(w_j^\top x)\sigma(w_j^\top x')].$$

These have ugly closed forms in terms of $x^\top x', \|x\|, \|x'\|$.

Deep net Infinite-Width

Let $a^{(l)} = W_l \sigma(a^{(l-1)})$ be the pre-activations with $\sigma(a^{(0)}) := x$. When the widths $m_l \to \infty$, the pre-activations follow a Gaussian process. These have covariance function given by:

$$\Sigma^{(0)} = x^\top x'$$

$$A^{(l)} = \begin{bmatrix} \Sigma^{(l-1)}(x,x) & \Sigma^{(l-1)}(x,x') \\ \Sigma^{(l-1)}(x',x) & \Sigma^{(l-1)}(x',x') \end{bmatrix}$$

$$\Sigma^{(l)}(x,x') = \mathbb{E}_{(u,v)\sim A^{(l)}}[\sigma(u)\sigma(v)].$$

$\lim_{m_l \to \infty} K_{L+1} = \Sigma^{(L)}$ gives us the kernel of the last layer (Lee et al., Matthews et al.).
Define the gradient kernels as $\dot{\Sigma}^{(l)}(x,x') = \mathbb{E}_{(u,v)\sim A^{(l)}}[\sigma'(u)\sigma'(v)]$.
Using backprop equations and Gaussian Process arguments (Jacot et al. , Lee et al., Du et al., Yang, Arora et al.) can get

$$K_l(x,x') = \Sigma^{(l-1)}(x,x') \cdot \Pi_{l'=l}^{L} \dot{\Sigma}^{(l')}(x,x')$$

NTK Overview

Recall

$$f_\theta(x) = f_0(x) + \nabla f_\theta(x)(\theta - \theta_0) + O(\|\theta - \theta_0\|^2).$$

Linearized network (Li and Liang, Du et al., Chizat and Bach):

$$\hat{f}_\theta(x) = f_0(x) + \nabla f_\theta(x)^\top (\theta - \theta_0)$$

- The network and linearized network are close if GD ensures $\|\theta - \theta_0\|^2$ is small.
- If $f_0 \gg 1$, then GD will not stay close to the initialization[1]. Thus need to initialize so f_0 doesn't blow up.

[1] Probably need $f_0 = o(\sqrt{m})$, and is the only place neural net structure is used.

Initialization size

Common initialization schemes ensure that norms are roughly preserved at each layer. Initialization ensures $x_j^{(L)} = O(1)$.

$$x^{(l)} = \sigma(W x^{(l-1)})$$

$$f_0(x) = \sum_{j=1}^{m} a_j x_j^{(L)}$$

Initialization size

Common initialization schemes ensure that norms are roughly preserved at each layer. Initialization ensures $x_j^{(L)} = O(1)$.

$$x^{(l)} = \sigma(W x^{(l-1)})$$

$$f_0(x) = \sum_{j=1}^{m} a_j x_j^{(L)}$$

Important Observation

If $a_j^2 \sim \frac{1}{n_{\text{in}}} = \frac{1}{m}$, then $f_0(x) = O(1)$.

- For two-layer case, first noticed by Li and Liang. For deep case, used by Jacot et al., Du et al., Allen-Zhu et al., Zou et al.
- Initialization is a $\sqrt{m}$ factor smaller than the worst-case.

Loss with unique root (Square loss , hinge loss)

Heuristic reasoning:

- Define $J = p \times n$ Jacobian matrix of f. Need to solve $J(\theta - \theta_0) = y - f_0$, which has a solution if $p \gg n$ (and some non-degeneracy).
- $\|\hat{\theta} - \theta_0\|^2 = (y - f_0)^\top (J^\top J)^{-1} (y - f_0)$ and does not depend on m (assuming $J^\top J$ concentrates).

Curvature

As $m \to \infty$ and $f_0 = O(1)$, thus the amount we need to move is constant

$$\|\hat{\theta} - \theta_0\|^2 = (y - f_0)^\top (J^\top J)^{-1} (y - f_0).$$

Curvature

As $m \to \infty$ and $f_0 = O(1)$, thus the amount we need to move is constant

$$\|\hat{\theta} - \theta_0\|^2 = (y - f_0)^\top (J^\top J)^{-1} (y - f_0).$$

Let's look at how "fast" the prediction function deviates from linear, which is given by the Hessian of f_θ.

Curvature

As $m \to \infty$ and $f_0 = O(1)$, thus the amount we need to move is constant

$$\|\hat{\theta} - \theta_0\|^2 = (y - f_0)^\top (J^\top J)^{-1} (y - f_0).$$

Let's look at how "fast" the prediction function deviates from linear, which is given by the Hessian of f_θ. Roughly,

$$\|\nabla^2 f_\theta(x)\| = o_m(1) \approx \frac{1}{\sqrt{m}}.$$

Curvature

As $m \to \infty$ and $f_0 = O(1)$, thus the amount we need to move is constant

$$\|\hat{\theta} - \theta_0\|^2 = (y - f_0)^\top (J^\top J)^{-1} (y - f_0).$$

Let's look at how "fast" the prediction function deviates from linear, which is given by the Hessian of f_θ. Roughly,

$$\|\nabla^2 f_\theta(x)\| = o_m(1) \approx \frac{1}{\sqrt{m}}.$$

Two-layer net (NTK parametrization)

$\nabla^2_{w_j} f(x) = \frac{1}{\sqrt{m}} a_j \sigma''(w_j^\top x) x x^\top$ and $\nabla^2_{a_j, w_j} f(x) = \frac{1}{\sqrt{m}} \sigma'(w_j^\top x) x$

The curvature vanishes as the width increases (due to how we parametrize/initialize).

Curvature

As $m \to \infty$ and $f_0 = O(1)$, thus the amount we need to move is constant

$$\|\hat{\theta} - \theta_0\|^2 = (y - f_0)^\top (J^\top J)^{-1} (y - f_0).$$

Let's look at how "fast" the prediction function deviates from linear, which is given by the Hessian of f_θ. Roughly,

$$\|\nabla^2 f_\theta(x)\| = o_m(1) \approx \frac{1}{\sqrt{m}}.$$

Two-layer net (NTK parametrization)

$\nabla^2_{w_j} f(x) = \frac{1}{\sqrt{m}} a_j \sigma''(w_j^\top x) x x^\top$ and $\nabla^2_{a_j, w_j} f(x) = \frac{1}{\sqrt{m}} \sigma'(w_j^\top x) x$

The curvature vanishes as the width increases (due to how we parametrize/initialize).

Implication on Training Dynamics

Since the curvature can be made small by overparametrization, the gradient flow dynamics of $\hat{f}_{\theta_t}$ and f_{θ_t} can be bounded:

$$\|\hat{f}_{\theta_t} - f_{\theta_t|}\|_\infty \leq O(\|\nabla^2 f_\theta(x)\|) = \frac{1}{\sqrt{m}}.$$

In some of the papers, the linearized function $\hat{f}$ is referred to a pseudo-network (Li and Liang, (Allen-Zhu et al.) [5], Zou et al.)

Training Error

What has been proved:

- Two-layer convergence to global minimizer (Li and Liang, Du et al., Oymak and Soltanolkotabi)
- Deep nets, Convolutional Deep nets, Resnets (Du et al., Allen-Zhu et al., Zou et al.)

The requirements on width are n^2 or worse[2]. However with ResNet the width is not depth dependent (Zhang et al.).

[2] It can be significantly improved with data assumptions to $m \gtrsim \|f\|_K^2$.

What functions can be efficiently learned?

Let K be an induced kernel. The sample complexity of learning a kernel class is

$$n \gtrsim \|f\|_K^2/\epsilon^2.$$

What functions can be efficiently learned?

Let K be an induced kernel. The sample complexity of learning a kernel class is

$$n \gtrsim \|f\|_K^2/\epsilon^2.$$

Write our target function as a linear function in the RKHS (Sridharan et al., Zhang et al.):

$$K(x,x') = g(\rho) = \sum c_i \rho^i = \phi(x)^\top \phi(x') \text{ and } \phi(x)_i = \sqrt{c_i} x^i.$$

$f(x) = x^k$ and σ is a monomial of degree k. Then

$$f(x) = \langle \frac{1}{\sqrt{c_k}} e_k, \phi(x) \rangle \text{ and } \|f\|_K^2 = \frac{1}{c_k}.$$

Polynomials

$$f(x) = \sum a_j x^j = \left\langle \sum_j \frac{a_j}{\sqrt{c_j}} e_j, \phi(x) \right\rangle, \quad \|f\|_K^2 = \sum_j a_j^2 / c_j.$$

Multivariate case: Let $f(x) = \sum_j a_j (w_j^\top x)^j$ with $\|w_j\|_2 = 1$, then $\|f\|_K^2 = \sum_j |a_j|^2 / c_j$ [3].
What is learnable?

- Constant degree polynomials with sample complexity $1/c_k$.
- Functions whose coefficients a_j decay quickly. For two-layer NTK, $c_j \asymp 1/j^2$, so need $\sum_j |a_j|^2 j^2 < \infty$.

[3] If Kernel has nullspace, then should be $\|f\|_K^2 \leq \sum_j |a_j|^2 / c_j$.

Teacher network:

$$f(x) = \sum a_j \sigma(w_j^\top x) = \sum_d \sum_j \alpha_{j,d} (w_j^\top x)^d$$

All such networks are learnable as long as the σ has coefficients decaying fast enough. Deep networks and smooth activations can "recurse" argument (similar to Zhang et al.)

- Arora et al. used this to show when the teacher network has smooth activation that NTK can learn.
- Allen-Zhu et al. used a direct construction to find the RKHS function (pseudo-network) instead of the series expansion of the kernel/target.
- Cao and Gu, Daniely showed that functions in the RKHS are learnable via deep networks.
- Previously known that such teacher networks are learnable with kernel $K(x, y) = 1/(2 - x^\top y)$ and its recursive variant (Sridharan et al., Zhang et al.)

When does the kernel regime hold?

- Square loss: For $m > m_0$, $\|\hat{f}_t - f_t\| = \frac{1}{\sqrt{m}}$ for all time t.
- Logistic loss: For all time t until $L(f_t) \approx \frac{1}{m}$.
- Logistic Loss: For very large times t, not a kernel predictor (Gunasekar et al., Nacson et al.).
- SGD with fresh data and either loss: For small time t, SGD on kernel and SGD of network are same. They will start differing at some point (Mei et al.)

Learning Rate

Learning rate schedule is an important (probably the main reason) that networks trained in practice are not in the Kernel regime.

NTK Parametrization vs Standard Parametrization

Let's consider

$$f_A(x) = \sum_j a_j \sigma(w_j^\top x) \text{ and } f_B(x) = \frac{1}{\sqrt{m}} \sum_j a_j \sigma(w_j^\top x)$$

Assume that $\|x\|^2 = d$.

- A: standard initialiation is $w_j \sim N(0, I/d)$ and $a_j \sim N(0, 1/m)$
- B: NTK initialization is $w_j \sim N(0, I/d)$ and $a_j \sim N(0, 1)$.
- Both initializations ensure that $f_0(x) = O(1)$ and parametrize the same functions.

However to get the same dynamics on f_t, we need to scale learning rate (Lee et al.).

$$\theta_{t+1}^{(A)} = \theta_t^{(A)} - \eta \nabla L_1(\theta^{(A)}) \quad \leftrightarrow \quad \theta_{t+1}^{(B)} = \theta_t^{(B)} - m\eta \nabla L_2(\theta^{(B)})$$

- Thus for NTK to learn the same function as SGD on standard parametrization with constant learning rate, we need an infinite learning rate on NTK parametrization.
- Infinite learning rate means we would leave the kernel regime.

So in practice, if you keep the same learning rate when using wider and wider networks, NTK won't be a good approximation.

1. If $f^\star(x) = \sigma(w^\top x)$ (single ReLU), then need exponential in d many random features to approximate this model (for $K = E[\sigma(wx)\sigma(wy)])$) , or need predictors with exponential in d norm (Yehudai and Shamir). If we choose the model to be $f_\theta(x) = \sum_j \sigma(w_j^\top x)$, then it is learnable with $O(d)$ samples (Soltanolkotabi).
2. With $m = d^k$ can only learn as well as fitting a degree k polynomial (Ghorbani et al.).
3. For a simple distribution realizable by four ReLU with $n \lesssim d^2$ samples, no better than random guessing. The idea is the same as the first bullet. The RKHS inductive bias is very poorly aligned with targets that are "sparse" in neuron space. Intuition: $f^\star$ is 1-sparse in the space of neurons meaning $f^\star(x) = \int \rho(w)\sigma(w^\top x)dw$ and ρ is a dirac delta. The RKHS inductive bias is $\|\rho\|_2$ which is really terrible when ρ is a dirac delta. If you could enforce a $\|\rho\|_1$ inductive bias, then the sample complexity is $O(d)$
   If you could learn with respect to $\|\rho\|_1$, the sample complexity is $n \gtrsim d/\epsilon^2$.

Going beyond NTK

WARNING: What follows are opinions that are only lightly grounded in mathematics.

Going beyond NTK

WARNING: What follows are opinions that are only lightly grounded in mathematics.

Question

Is the kernel regime reflecting the success of deep learning?

Going beyond NTK

WARNING: What follows are opinions that are only lightly grounded in mathematics.

Question

Is the kernel regime reflecting the success of deep learning?

- NTK accuracy and SGD accuracy on the same architecture have a gap (Lee et al., Arora et al.)
- Can we close this gap and how?

Random thoughts:

- Learning rate is key. If you use a small LR, then NTK and SGD find similar predictors (but the test accuracy is not super high). With the same architecture and the LR is tuned, then the test accuracy is higher. NTK implicitly enforces the learning rate to be infinitesmally small, which may hurt learning.
- Logistic Loss: If you try to solve matrix completion with $f_\Theta((i,j)) = (UV^\top)_{ij}$, then NTK simply imputes the observed entries. However if you run GD for a long time, then you get minimum nuclear norm (Srebro, Gunasekar et al.)

Logistic Loss (and maybe one-pass SGD): We know that asymptotically (with numerous assumptions) converges to a stationary point of $\arg\min_{y_i f_\theta(x_i) \geq 1} \|\theta\|_2$ (Nacson et al., Li and Lv)[4] For even simple models, ℓ_2 regularization on parameters leads to interesting inductive bias.

- For deep linear nets, schatten $2/L$ norm, so promotes low rank.
- For linear model $\beta = \theta_1 \odot \theta_2$, gets $\|\beta\|_1$.
- For two-layer ReLU net, $f(x) = \int \rho(w)\sigma(wx)dw$, gets $\|\rho\|_1$ (Neyshabur et al. , Bengio et al., Wei et al.).
- Deep ReLU net, size-free complexity bound (Golowich et al.)

However we should NOT expect to get global max-margin except in special example such as matrix sensing.
Question: If I initialize at the NTK solution, which stationary point of $\arg\min_{y_i f_\theta(x_i) \geq 1} \|\theta\|_2$ do you converge to? This is what happens in super-wide networks with infinitely small LR.

[4] NTK is not stationary point of this.

Thank You.

Questions?