Neural Tangent Kernels (+ etc)

CPSC 532S: Modern Statistical Learning Theory

23 March 2022

cs.ubc.ca/~dsuth/532S/22/

Admin

- A3 Q1 is broken, to be replaced (with something as similar as possible) tonight
  - Whole assignment now due **Monday**
- A4 will be posted soon, due Friday the 8th (last day of term)
  - Yes, the same day as the project report; usual late policy

- Project scope:
  - I’m just looking for signs that you’ve read and understood the papers
  - If you’re doing a lit review: ~3-4 papers is the expected amount
    - Talk about how the settings / conclusions relate to each other, what they leave open, etc
  - An “extension” can absolutely just be poking at the assumptions and talking about when they hold / don’t

What's left in deep learning?

- Talked about **universal approximation** results
  - Shallow, wide networks: proved in 1d, sketched proof in general
  - Mentioned results for deep, narrow networks
  - Relationship to circuit complexity

What's left in deep learning?

- Talked about **universal approximation** results
  - Shallow, wide networks: proved in 1d, sketched proof in general
  - Mentioned results for deep, narrow networks
  - Relationship to circuit complexity

- Some **generalization** bounds
  - VC dimension is known relatively sharply
  - AlexNet can shatter CIFAR-10, but it still generalizes well
  - Norm-based Rademacher bounds possible, but don't seem very tight

What's left in deep learning?

- Talked about **universal approximation** results
  - Shallow, wide networks: proved in 1d, sketched proof in general
  - Mentioned results for deep, narrow networks
  - Relationship to circuit complexity

- Some **generalization** bounds
  - VC dimension is known relatively sharply
  - AlexNet can shatter CIFAR-10, but it still generalizes well
  - Norm-based Rademacher bounds possible, but don't seem very tight

- Ignoring the many issues, approximation + generalization means ERM works
  - …but ERM is NP-hard, even for square loss, even with *one* ReLU

What's left in deep learning?

- Talked about **universal approximation** results
  - Shallow, wide networks: proved in 1d, sketched proof in general
  - Mentioned results for deep, narrow networks
  - Relationship to circuit complexity

- Some **generalization** bounds
  - VC dimension is known relatively sharply
  - AlexNet can shatter CIFAR-10, but it still generalizes well
  - Norm-based Rademacher bounds possible, but don't seem very tight

- Ignoring the many issues, approximation + generalization means ERM works
  - …but ERM is NP-hard, even for square loss, even with *one* ReLU

- What's the **optimization** error for SGD/similar?

Nonconvex optimization

- Neural nets are not convex

Nonconvex optimization

- Neural nets are not convex
- Even **deep linear networks** are not convex

Nonconvex optimization

- Neural nets are not convex
- Even **deep linear networks** are not convex

- But we do know that SGD converges to a *critical point* under fairly mild conditions

Nonconvex optimization

- Neural nets are not convex
- Even **deep linear networks** are not convex

- But we do know that SGD converges to a *critical point* under fairly mild conditions
  - e.g.: if $f \geq f^{\inf}$ is differentiable and β-smooth, and there are A, B, C s.t. for all x, $\mathbb{E}\left[\|\hat{g}(x)\|^2\right] \leq 2A(f(x) - f^{\inf}) + B\|\nabla f(X)\|^2 + C$, then the *best* iterate from $\mathcal{O}(\varepsilon^{-4})$ steps has $\mathbb{E}\left[\|\nabla f(x)\|^2\right] \leq \varepsilon^2$ (Khaled/Richtárik 2020)

Nonconvex optimization

- Neural nets are not convex
- Even **deep linear networks** are not convex

- But we do know that SGD converges to a *critical point* under fairly mild conditions
  - e.g.: if $f \geq f^{\inf}$ is differentiable and β-smooth, and there are A, B, C s.t. for all x, $\mathbb{E}\left[\|\hat{g}(x)\|^2\right] \leq 2A(f(x) - f^{\inf}) + B\|\nabla f(X)\|^2 + C$, then the *best* iterate from $\mathcal{O}(\varepsilon^{-4})$ steps has $\mathbb{E}\left[\|\nabla f(x)\|^2\right] \leq \varepsilon^2$ (Khaled/Richtárik 2020)

- In deep linear nets, local minima are global minima (Kawaguchi 2016, Laurent/von Brecht 2019)

Nonconvex optimization

- Neural nets are not convex
- Even **deep linear networks** are not convex

- But we do know that SGD converges to a *critical point* under fairly mild conditions
  - e.g.: if $f \geq f^{\inf}$ is differentiable and β-smooth, and there are A, B, C s.t. for all x, $\mathbb{E}\left[\|\hat{g}(x)\|^2\right] \leq 2A(f(x) - f^{\inf}) + B\|\nabla f(X)\|^2 + C$, then the *best* iterate from $\mathcal{O}(\varepsilon^{-4})$ steps has $\mathbb{E}\left[\|\nabla f(x)\|^2\right] \leq \varepsilon^2$ (Khaled/Richtárik 2020)

- In deep linear nets, local minima are global minima (Kawaguchi 2016, Laurent/von Brecht 2019)
  - …but there are saddle points, including "bad" ones where $\lambda_{\min}(\nabla^2 f) = 0$

Nonconvex optimization

- Neural nets are not convex
- Even **deep linear networks** are not convex

- But we do know that SGD converges to a *critical point* under fairly mild conditions
  - e.g.: if $f \geq f^{\text{inf}}$ is differentiable and β-smooth, and there are A, B, C s.t. for all x, $\mathbb{E}\left[\|\hat{g}(x)\|^2\right] \leq 2A(f(x) - f^{\text{inf}}) + B\|\nabla f(X)\|^2 + C$, then the *best* iterate from $\mathcal{O}(\varepsilon^{-4})$ steps has $\mathbb{E}\left[\|\nabla f(x)\|^2\right] \leq \varepsilon^2$ (Khaled/Richtárik 2020)

- In deep linear nets, local minima are global minima (Kawaguchi 2016, Laurent/von Brecht 2019)
  - …but there are saddle points, including "bad" ones where $\lambda_{\min}(\nabla^2 f) = 0$
  - …but gradient descent almost surely escapes saddles, reaches a local min (Lee et al. 2016)

Nonconvex optimization

- Neural nets are not convex
- Even **deep linear networks** are not convex

- But we do know that SGD converges to a *critical point* under fairly mild conditions
  - e.g.: if $f \geq f^{\inf}$ is differentiable and β-smooth, and there are A, B, C s.t. for all x, $\mathbb{E}\left[\|\hat{g}(x)\|^2\right] \leq 2A(f(x) - f^{\inf}) + B\|\nabla f(X)\|^2 + C$, then the *best* iterate from $\mathcal{O}(\varepsilon^{-4})$ steps has $\mathbb{E}\left[\|\nabla f(x)\|^2\right] \leq \varepsilon^2$ (Khaled/Richtárik 2020)

- In deep linear nets, local minima are global minima (Kawaguchi 2016, Laurent/von Brecht 2019)
  - …but there are saddle points, including "bad" ones where $\lambda_{\min}(\nabla^2 f) = 0$
  - …but gradient descent almost surely escapes saddles, reaches a local min (Lee et al. 2016)
  - …but it can take exponential time to escape (Du et al. 2017)

Nonconvex optimization

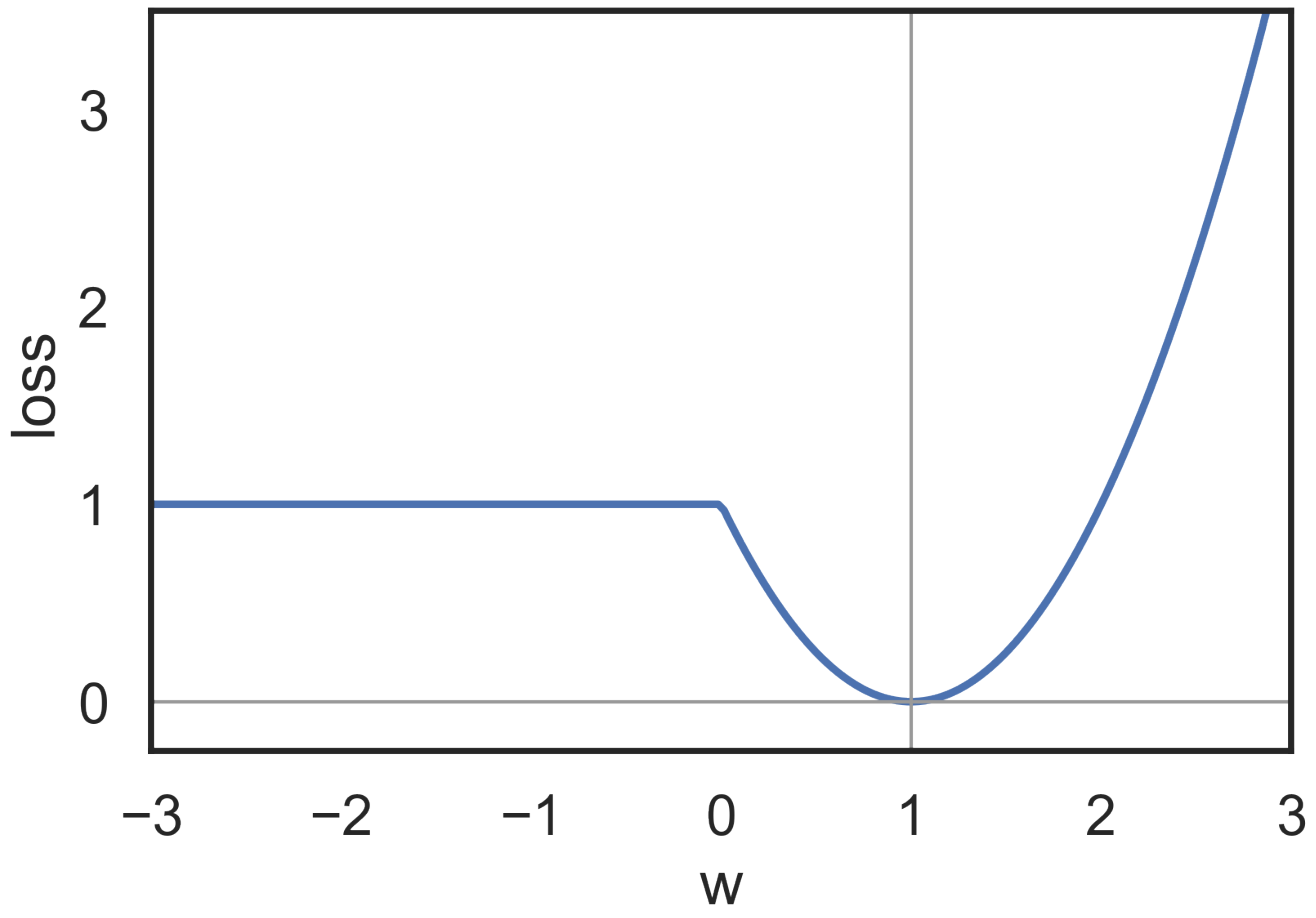
- Neural nets are not convex
- Even **deep linear networks** are not convex

- But we do know that SGD converges to a *critical point* under fairly mild conditions
  - e.g.: if $f \geq f^{\text{inf}}$ is differentiable and β-smooth, and there are A, B, C s.t. for all x, $\mathbb{E}\left[\|\hat{g}(x)\|^2\right] \leq 2A(f(x) - f^{\text{inf}}) + B\|\nabla f(X)\|^2 + C$, then the *best* iterate from $\mathcal{O}(\varepsilon^{-4})$ steps has $\mathbb{E}\left[\|\nabla f(x)\|^2\right] \leq \varepsilon^2$ (Khaled/Richtárik 2020)

- In deep linear nets, local minima are global minima (Kawaguchi 2016, Laurent/von Brecht 2019)
  - …but there are saddle points, including "bad" ones where $\lambda_{\min}(\nabla^2 f) = 0$
  - …but gradient descent almost surely escapes saddles, reaches a local min (Lee et al. 2016)
  - …but it can take exponential time to escape (Du et al. 2017)
  - …but that doesn't happen on deep linear nets [under conditions] (Arora et al. 2019)

Bad local minima in ReLU nets

$h(x) = \text{ReLU}(wx)$ (reals to reals), square loss, $S = \big((1,1)\big)$:



Sub-Optimal Local Minima Exist for Neural Networks with Almost All Non-Linear Activations

Tian Ding*   Dawei Li †   Ruoyu Sun ‡

Nov 4, 2019

(S)GD works on over-parameterized nets

- Okay, so there are bad local minima

(S)GD works on over-parameterized nets

- Okay, so there are bad local minima
- But…does (S)GD actually find them?

(S)GD works on over-parameterized nets

- Okay, so there are bad local minima
- But…does (S)GD actually find them?
- Several papers around 2018-19 showed that:

(S)GD works on over-parameterized nets

- Okay, so there are bad local minima
- But…does (S)GD actually find them?
- Several papers around 2018-19 showed that:
  - If the network is *very overparameterized* (width $\gg n$, possibly $\to \infty$)

(S)GD works on over-parameterized nets

- Okay, so there are bad local minima
- But…does (S)GD actually find them?
- Several papers around 2018-19 showed that:
    - If the network is *very overparameterized* (width $\gg n$, possibly $\to \infty$)
    - and we use an appropriate random initialization

(S)GD works on over-parameterized nets

- Okay, so there are bad local minima
- But…does (S)GD actually find them?
- Several papers around 2018-19 showed that:
  - If the network is *very overparameterized* (width $\gg n$, possibly $\to \infty$)
  - and we use an appropriate random initialization
  - with square loss

(S)GD works on over-parameterized nets

- Okay, so there are bad local minima
- But…does (S)GD actually find them?
- Several papers around 2018-19 showed that:
  - If the network is *very overparameterized* (width $\gg n$, possibly $\to \infty$)
  - and we use an appropriate random initialization
  - with square loss
  - then (S)GD finds a global minimum

(S)GD works on over-parameterized nets

- Okay, so there are bad local minima
- But…does (S)GD actually find them?
- Several papers around 2018-19 showed that:
  - If the network is *very overparameterized* (width $\gg n$, possibly $\to \infty$)
  - and we use an appropriate random initialization
  - with square loss
  - then (S)GD finds a global minimum
- Implicit in these papers:

(S)GD works on over-parameterized nets

- Okay, so there are bad local minima
- But…does (S)GD actually find them?
- Several papers around 2018-19 showed that:
  - If the network is *very overparameterized* (width $\gg n$, possibly $\to \infty$)
  - and we use an appropriate random initialization
  - with square loss
  - then (S)GD finds a global minimum
- Implicit in these papers:
  - Behaviour of deep nets converges to kernel ridge regression with the **neural tangent kernel**

Shallow case

- Let’s start with a depth 2 case (Telgarsky notes section 4)

Shallow case

- Let's start with a depth 2 case (Telgarsky notes section 4)
- $$f(x; W) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \, \sigma(w_j^\top x)$$

Shallow case

- Let's start with a depth 2 case (Telgarsky notes section 4)
- $f(x; W) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \sigma(w_j^\top x)$
  - w_i is the ith row of $W \in \mathbb{R}^{m \times d}$ (as a column vector)

Shallow case

- Let's start with a depth 2 case (Telgarsky notes section 4)
- $$f(x; W) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \sigma(w_j^\top x)$$
  - w_i is the ith row of $W \in \mathbb{R}^{m \times d}$ (as a column vector)
  - Going to treat the a_j as *fixed* for simplicity

Shallow case

- Let's start with a depth 2 case (Telgarsky notes section 4)
- $f(x; W) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \sigma(w_j^\top x)$
  - w_i is the ith row of $W \in \mathbb{R}^{m \times d}$ (as a column vector)
  - Going to treat the a_j as *fixed* for simplicity
- The core idea: think about a **linearization** of f **in** W

Shallow case

- Let's start with a depth 2 case (Telgarsky notes section 4)
- $f(x; W) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \sigma(w_j^\top x)$
  - w_i is the ith row of $W \in \mathbb{R}^{m \times d}$ (as a column vector)
  - Going to treat the a_j as *fixed* for simplicity
- The core idea: think about a **linearization** of f **in** W
  - $f_{W_0}(x; W) = f(x; W_0) + \langle \nabla f(x; W_0), W - W_0 \rangle$

Shallow case

- Let's start with a depth 2 case (Telgarsky notes section 4)
- $f(x; W) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \sigma(w_j^\top x)$
  - w_i is the ith row of $W \in \mathbb{R}^{m \times d}$ (as a column vector)
  - Going to treat the a_j as *fixed* for simplicity
- The core idea: think about a **linearization** of f **in** W
  - $f_{W_0}(x; W) = f(x; W_0) + \langle \nabla f(x; W_0), W - W_0 \rangle$
  - Approximates behaviour of f as we change W; nonlinear in x

Shallow case

- Let's start with a depth 2 case (Telgarsky notes section 4)
- $f(x; W) = \dfrac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \, \sigma(w_j^\top x)$
  - w_i is the ith row of $W \in \mathbb{R}^{m \times d}$ (as a column vector)
  - Going to treat the a_j as *fixed* for simplicity
- The core idea: think about a **linearization** of f **in** W
  - $f_{W_0}(x; W) = f(x; W_0) + \langle \nabla f(x; W_0), W - W_0 \rangle$
  - Approximates behaviour of f as we change W; nonlinear in x
  - We'll see that, for large m and random W_0, $f \approx f_{W_0}$ through training

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

$$f_{W_0}(x; W) = f(x; W_0) + \langle \nabla f(x; W_0), W - W_0 \rangle$$

$$f(x; W) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \, \sigma(w_j^\top x)$$

$$\begin{aligned} f_{W_0}(x; W) &= f(x; W_0) + \langle \nabla f(x; W_0), W - W_0 \rangle \\ &= \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \left[\sigma(w_{0,j}^\top x) + \sigma'(w_{0,j}) x^\top (w_j - w_{0,j}) \right] \end{aligned}$$

$$f(x;W) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \, \sigma(w_j^\top x)$$

$$\begin{aligned} f_{W_0}(x;W) &= f(x;W_0) + \langle \nabla f(x;W_0), W - W_0 \rangle \\ &= \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \left[\sigma(w_{0,j}^\top x) + \sigma'(w_{0,j}) x^\top (w_j - w_{0,j}) \right] \\ &= \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \left(\left[\sigma(w_{0,j}^\top x) - \sigma'(w_{0,j}) w_{0,j}^\top x \right] + \sigma'(w_{0,j}) w_j^\top x \right) \end{aligned}$$

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

$$\begin{aligned} f_{W_0}(x; W) &= f(x; W_0) + \langle \nabla f(x; W_0), W - W_0 \rangle \\ &= \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \left[\sigma(w_{0,j}^\top x) + \sigma'(w_{0,j}) x^\top (w_j - w_{0,j}) \right] \\ &= \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \left(\underbrace{\left[\sigma(w_{0,j}^\top x) - \sigma'(w_{0,j}) w_{0,j}^\top x \right]}_{= 0 \text{ for ReLU}} + \sigma'(w_{0,j}) w_j^\top x \right) \end{aligned}$$

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

$$\begin{aligned} f_{W_0}(x; W) &= f(x; W_0) + \langle \nabla f(x; W_0), W - W_0 \rangle \\ &= \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \left[\sigma(w_{0,j}^\top x) + \sigma'(w_{0,j}) x^\top (w_j - w_{0,j}) \right] \\ &= \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \left(\underbrace{\left[\sigma(w_{0,j}^\top x) - \sigma'(w_{0,j}) w_{0,j}^\top x \right]}_{= 0 \text{ for ReLU}} + \sigma'(w_{0,j}) w_j^\top x \right) \end{aligned}$$

$$f_{W_0}(x; W) = \langle \nabla f(x; W_0), W \rangle$$

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

$$\begin{aligned} f_{W_0}(x; W) &= f(x; W_0) + \langle \nabla f(x; W_0), W - W_0 \rangle \\ &= \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \left[\sigma(w_{0,j}^\top x) + \sigma'(w_{0,j}) x^\top (w_j - w_{0,j}) \right] \\ &= \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \left(\underbrace{\left[\sigma(w_{0,j}^\top x) - \sigma'(w_{0,j}) w_{0,j}^\top x \right]}_{= 0 \text{ for ReLU}} + \sigma'(w_{0,j}) w_j^\top x \right) \end{aligned}$$

$$f_{W_0}(x; W) = \langle \nabla f(x; W_0), W \rangle$$

We'll see shortly that $f - f_0$ *shrinks* as m grows

$$f(x; W) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \, \sigma(w_j^\top x)$$

$$f_{W_0}(x; W) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \left(\sigma(w_{0,j}^\top x) - \sigma'(w_{0,j}) w_{0,j}^\top x + \sigma'(w_{0,j}) w_j^\top x \right)$$

$$f(x;W) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \, \sigma(w_j^\top x)$$

$$f_{W_0}(x;W) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \left(\sigma(w_{0,j}^\top x) - \sigma'(w_{0,j}) w_{0,j}^\top x + \sigma'(w_{0,j}) w_j^\top x \right)$$

If σ is β-smooth, $|a_j| \leq 1$, $\|x\| \leq 1$:

$$f(x; W) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \, \sigma(w_j^\top x)$$

$$f_{W_0}(x; W) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \left(\sigma(w_{0,j}^\top x) - \sigma'(w_{0,j}) w_{0,j}^\top x + \sigma'(w_{0,j}) w_j^\top x \right)$$

If σ is β-smooth, $|a_j| \leq 1$, $\|x\| \leq 1$:

$$\left| f(x; W) - f_{W_0}(x; W) \right| \leq \frac{1}{\sqrt{m}} \sum_{j=1}^{m} |a_j| \, \left| \sigma(w_j^\top x) - \sigma(w_{0,j}^\top x) - \sigma'(w_{0,j}^\top x) x^\top (w_j - w_{0,j}) \right|$$

$$f(x; W) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \, \sigma(w_j^\top x)$$

$$f_{W_0}(x; W) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \left(\sigma(w_{0,j}^\top x) - \sigma'(w_{0,j}) w_{0,j}^\top x + \sigma'(w_{0,j}) w_j^\top x \right)$$

If σ is β-smooth, $|a_j| \leq 1$, $\|x\| \leq 1$:

$$|\sigma(r) - \sigma(s) - \sigma'(s)(r - s)| = \left| \int_r^s \sigma''(z)(s - z) \mathrm{d}z \right|$$

$$\left| f(x; W) - f_{W_0}(x; W) \right| \leq \frac{1}{\sqrt{m}} \sum_{j=1}^{m} |a_j| \, \left| \sigma(w_j^\top x) - \sigma(w_{0,j}^\top x) - \sigma'(w_{0,j}^\top x) x^\top (w_j - w_{0,j}) \right|$$

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

$$f_{W_0}(x; W) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \left(\sigma(w_{0,j}^\top x) - \sigma'(w_{0,j}) w_{0,j}^\top x + \sigma'(w_{0,j}) w_j^\top x \right)$$

If σ is β-smooth, $|a_j| \leq 1$, $\|x\| \leq 1$:

$$|\sigma(r) - \sigma(s) - \sigma'(s)(r - s)| = \left| \int_r^s \sigma''(z)(s - z) \mathrm{d}z \right| \leq \frac{\beta}{2}(r - s)^2$$

$$\left| f(x; W) - f_{W_0}(x; W) \right| \leq \frac{1}{\sqrt{m}} \sum_{j=1}^{m} |a_j| \ \left| \sigma(w_j^\top x) - \sigma(w_{0,j}^\top x) - \sigma'(w_{0,j}^\top x) x^\top (w_j - w_{0,j}) \right|$$

$$f(x; W) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \, \sigma(w_j^\top x)$$

$$f_{W_0}(x; W) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \left(\sigma(w_{0,j}^\top x) - \sigma'(w_{0,j}) w_{0,j}^\top x + \sigma'(w_{0,j}) w_j^\top x \right)$$

If σ is β-smooth, $|a_j| \leq 1$, $\|x\| \leq 1$:

$$|\sigma(r) - \sigma(s) - \sigma'(s)(r-s)| = \left| \int_r^s \sigma''(z)(s-z) \mathrm{d}z \right| \leq \frac{\beta}{2}(r-s)^2$$

$$\left| f(x; W) - f_{W_0}(x; W) \right| \leq \frac{1}{\sqrt{m}} \sum_{j=1}^{m} |a_j| \, \left| \sigma(w_j^\top x) - \sigma(w_{0,j}^\top x) - \sigma'(w_{0,j}^\top x) x^\top (w_j - w_{0,j}) \right|$$

$$\leq \frac{1}{\sqrt{m}} \sum_{j=1}^{m} \frac{1}{2} \beta (w_j^\top x - w_{0,j}^\top x)^2$$

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

$$f_{W_0}(x; W) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \left(\sigma(w_{0,j}^\top x) - \sigma'(w_{0,j}) w_{0,j}^\top x + \sigma'(w_{0,j}) w_j^\top x \right)$$

If σ is β-smooth, $|a_j| \le 1$, $\|x\| \le 1$:

$$|\sigma(r) - \sigma(s) - \sigma'(s)(r - s)| = \left| \int_r^s \sigma''(z)(s - z) \mathrm{d}z \right| \le \frac{\beta}{2}(r - s)^2$$

$$\left| f(x; W) - f_{W_0}(x; W) \right| \le \frac{1}{\sqrt{m}} \sum_{j=1}^{m} |a_j| \left| \sigma(w_j^\top x) - \sigma(w_{0,j}^\top x) - \sigma'(w_{0,j}^\top x) x^\top (w_j - w_{0,j}) \right|$$

$$\le \frac{1}{\sqrt{m}} \sum_{j=1}^{m} \frac{1}{2} \beta (w_j^\top x - w_{0,j}^\top x)^2 \le \frac{\beta}{2\sqrt{m}} \sum_{j=1}^{m} \|w_j - w_{0,j}\|^2 \|x\|^2$$

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

$$f_{W_0}(x; W) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \left(\sigma(w_{0,j}^\top x) - \sigma'(w_{0,j}) w_{0,j}^\top x + \sigma'(w_{0,j}) w_j^\top x \right)$$

If σ is β-smooth, $|a_j| \leq 1$, $\|x\| \leq 1$:

$$|\sigma(r) - \sigma(s) - \sigma'(s)(r - s)| = \left| \int_r^s \sigma''(z)(s - z) \mathrm{d}z \right| \leq \frac{\beta}{2}(r - s)^2$$

$$\left| f(x; W) - f_{W_0}(x; W) \right| \leq \frac{1}{\sqrt{m}} \sum_{j=1}^{m} |a_j| \left| \sigma(w_j^\top x) - \sigma(w_{0,j}^\top x) - \sigma'(w_{0,j}^\top x) x^\top (w_j - w_{0,j}) \right|$$

$$\leq \frac{1}{\sqrt{m}} \sum_{j=1}^{m} \frac{1}{2} \beta (w_j^\top x - w_{0,j}^\top x)^2 \leq \frac{\beta}{2\sqrt{m}} \sum_{j=1}^{m} \|w_j - w_{0,j}\|^2 \|x\|^2 \leq \frac{\beta}{2\sqrt{m}} \|W - W_0\|_F^2$$

Linearization quality

- For a two-layer net with β-smooth hidden activations, second-layer weights $\leq 1/\sqrt{m}$ with linear activation, then for any $\|x\| \leq 1,$ $\quad |f(x; W) - f_0(x; W)| \leq \frac{\beta}{2\sqrt{m}} \|W - W_0\|_F^2$

Linearization quality

- For a two-layer net with β-smooth hidden activations, second-layer weights $\leq 1/\sqrt{m}$ with linear activation, then for any $||x|| \leq 1$, $\quad |f(x;W) - f_0(x;W)| \leq \dfrac{\beta}{2\sqrt{m}} ||W - W_0||_F^2$
  - This holds for *any* W and W_0, but only for this shallow case

Linearization quality

- For a two-layer net with β-smooth hidden activations, second-layer weights $\leq 1/\sqrt{m}$ with linear activation, then for any $\|x\| \leq 1$, $\quad |f(x; W) - f_0(x; W)| \leq \frac{\beta}{2\sqrt{m}} \|W - W_0\|_F^2$
  - This holds for *any* W and W_0, but only for this shallow case
- For two-layer ReLU nets as above, with entries of W_0 iid standard normal: for any $B \geq 0$ and any fixed $x \in \mathbb{R}^d$ with $\|x\| \leq 1$, with probability at least $1 - \delta$ over the draw of W_0,

$$\sup_{W:\, \|W - W_0\|_F \leq B} |f(x; W) - f_{W_0}(x; W)| \leq \frac{2B^{4/3} + B\log(1/\delta)^{1/4}}{m^{1/6}}$$

Linearization quality

- For a two-layer net with β-smooth hidden activations, second-layer weights $\leq 1/\sqrt{m}$ with linear activation, then for any $\|x\| \leq 1, \quad |f(x;W) - f_0(x;W)| \leq \frac{\beta}{2\sqrt{m}}\|W - W_0\|_F^2$
  - This holds for *any* W and W_0, but only for this shallow case
- For two-layer ReLU nets as above, with entries of W_0 iid standard normal: for any $B \geq 0$ and any fixed $x \in \mathbb{R}^d$ with $\|x\| \leq 1$, with probability at least $1 - \delta$ over the draw of W_0,

$$\sup_{W:\, \|W-W_0\|_F \leq B} |f(x;W) - f_{W_0}(x;W)| \leq \frac{2B^{4/3} + B\log(1/\delta)^{1/4}}{m^{1/6}}$$

  - Proof is more annoying: Telgarsky's Lemma 4.1

Linearization quality

- For a two-layer net with β-smooth hidden activations, second-layer weights $\leq 1/\sqrt{m}$ with linear activation, then for any $\|x\| \leq 1$, $\quad |f(x; W) - f_0(x; W)| \leq \frac{\beta}{2\sqrt{m}} \|W - W_0\|_F^2$
  - This holds for *any* W and W_0, but only for this shallow case
- For two-layer ReLU nets as above, with entries of W_0 iid standard normal: for any $B \geq 0$ and any fixed $x \in \mathbb{R}^d$ with $\|x\| \leq 1$, with probability at least $1 - \delta$ over the draw of W_0,
$$\sup_{W:\, \|W - W_0\|_F \leq B} |f(x; W) - f_{W_0}(x; W)| \leq \frac{2B^{4/3} + B \log(1/\delta)^{1/4}}{m^{1/6}}$$
  - Proof is more annoying: Telgarsky's Lemma 4.1
  - Can do multi-layer versions, but approximation degrades with depth

What happens in the linearized model?

- For the ReLU, $f_{W_0}(x; W) = \langle \nabla f(x; W_0), W \rangle$

What happens in the linearized model?

- For the ReLU, $f_{W_0}(x; W) = \langle \nabla f(x; W_0), W \rangle$
- This is a kernel model!

What happens in the linearized model?

- For the ReLU, $f_{W_0}(x; W) = \langle \nabla f(x; W_0), W \rangle$
- This is a kernel model!
  - $k(x, x') = \langle \nabla f(x; W_0), \nabla f(x'; W_0) \rangle$

What happens in the linearized model?

- For the ReLU, $f_{W_0}(x; W) = \langle \nabla f(x; W_0), W \rangle$
- This is a kernel model!
  - $k(x, x') = \langle \nabla f(x; W_0), \nabla f(x'; W_0) \rangle$

$$= \left\langle \begin{bmatrix} a_1 x^\top \sigma'(w_{0,1}^\top x)/\sqrt{m} \\ \vdots \\ a_m x^\top \sigma'(w_{0,m}^\top x)/\sqrt{m} \end{bmatrix}, \begin{bmatrix} a_1 (x')^\top \sigma'(w_{0,1}^\top x')/\sqrt{m} \\ \vdots \\ a_m (x')^\top \sigma'(w_{0,m}^\top x')/\sqrt{m} \end{bmatrix} \right\rangle$$

What happens in the linearized model?

- For the ReLU, $f_{W_0}(x; W) = \langle \nabla f(x; W_0), W \rangle$
- This is a kernel model!
  - $k(x, x') = \langle \nabla f(x; W_0), \nabla f(x'; W_0) \rangle$

$$= \left\langle \begin{bmatrix} a_1 x^\top \sigma'(w_{0,1}^\top x)/\sqrt{m} \\ \vdots \\ a_m x^\top \sigma'(w_{0,m}^\top x)/\sqrt{m} \end{bmatrix}, \begin{bmatrix} a_1 (x')^\top \sigma'(w_{0,1}^\top x')/\sqrt{m} \\ \vdots \\ a_m (x')^\top \sigma'(w_{0,m}^\top x')/\sqrt{m} \end{bmatrix} \right\rangle$$

$$= x^\top x' \left[\frac{1}{m} \sum_{j=1}^{m} \sigma'(w_{0,j}^\top x) \sigma'(w_{0,j}^\top x') \right]$$

What happens in the linearized model?

- For the ReLU, $f_{W_0}(x; W) = \langle \nabla f(x; W_0), W \rangle$
- This is a kernel model!
  - $k(x, x') = \langle \nabla f(x; W_0), \nabla f(x'; W_0) \rangle$

$$= \left\langle \begin{bmatrix} a_1 x^\top \sigma'(w_{0,1}^\top x)/\sqrt{m} \\ \vdots \\ a_m x^\top \sigma'(w_{0,m}^\top x)/\sqrt{m} \end{bmatrix}, \begin{bmatrix} a_1 (x')^\top \sigma'(w_{0,1}^\top x')/\sqrt{m} \\ \vdots \\ a_m (x')^\top \sigma'(w_{0,m}^\top x')/\sqrt{m} \end{bmatrix} \right\rangle$$

$$= x^\top x' \left[\frac{1}{m} \sum_{j=1}^{m} \sigma'(w_{0,j}^\top x) \sigma'(w_{0,j}^\top x') \right] \xrightarrow{m \to \infty} x^\top x' \, \mathbb{E}_w \left[\sigma'(w^\top x) \, \sigma'(w^\top x') \right]$$

arccos kernel

For $\|x\| = 1 = \|x'\|$, $\mathbb{E}_w[\sigma'(w^\top x)\sigma'(w^\top x')] = \frac{1}{2} - \frac{1}{2\pi}\arccos(x^\top x')$:

arccos kernel

For $\|x\| = 1 = \|x'\|$, $\mathbb{E}_w[\sigma'(w^\top x)\sigma'(w^\top x')] = \dfrac{1}{2} - \dfrac{1}{2\pi}\arccos(x^\top x')$:

This kernel is universal on $\{x \in \mathbb{R}^{d+1} : \|x\| = 1, x_{d+1} = 1/\sqrt{2}\}$

Non-ReLU, multi-layer version

- General σ: $f_{W_0}(x; W) = f(x; W_0) - \langle \nabla f(x; W_0), W_0 \rangle + \langle \nabla f(x; W_0), W \rangle$

Non-ReLU, multi-layer version

- General σ: $f_{W_0}(x; W) = f(x; W_0) - \langle \nabla f(x; W_0), W_0 \rangle + \langle \nabla f(x; W_0), W \rangle$
- Fitting f_{W_0} to labels y_i is fitting a function in $\mathscr{H}_k$ to the **residual** $y_i - f(x_i; W_0)$

Non-ReLU, multi-layer version

- General σ: $f_{W_0}(x; W) = f(x; W_0) - \langle \nabla f(x; W_0), W_0 \rangle + \langle \nabla f(x; W_0), W \rangle$
- Fitting f_{W_0} to labels y_i is fitting a function in $\mathcal{H}_k$ to the **residual** $y_i - f(x_i; W_0)$

- For multiple layers, idea is the same: kernel is still
$$k(x, x') = \langle \nabla f(x; W_0), \nabla f(x'; W_0) \rangle$$
but now ∇ uses **all** of the parameters in the network

Non-ReLU, multi-layer version

- General σ: $f_{W_0}(x; W) = f(x; W_0) - \langle \nabla f(x; W_0), W_0 \rangle + \langle \nabla f(x; W_0), W \rangle$
- Fitting f_{W_0} to labels y_i is fitting a function in $\mathscr{H}_k$ to the **residual** $y_i - f(x_i; W_0)$

- For multiple layers, idea is the same: kernel is still
$$k(x, x') = \langle \nabla f(x; W_0), \nabla f(x'; W_0) \rangle$$
but now ∇ uses **all** of the parameters in the network
    - but the linearization results are worse

Non-ReLU, multi-layer version

- General σ: $f_{W_0}(x; W) = f(x; W_0) - \langle \nabla f(x; W_0), W_0 \rangle + \langle \nabla f(x; W_0), W \rangle$
- Fitting f_{W_0} to labels y_i is fitting a function in $\mathscr{H}_k$ to the **residual** $y_i - f(x_i; W_0)$

- For multiple layers, idea is the same: kernel is still
$$k(x, x') = \langle \nabla f(x; W_0), \nabla f(x'; W_0) \rangle$$
but now ∇ uses **all** of the parameters in the network
  - but the linearization results are worse

- Can compute expectation version, even for convolutional nets with pooling

Non-ReLU, multi-layer version

- General σ: $f_{W_0}(x; W) = f(x; W_0) - \langle \nabla f(x; W_0), W_0 \rangle + \langle \nabla f(x; W_0), W \rangle$
- Fitting f_{W_0} to labels y_i is fitting a function in $\mathscr{H}_k$ to the **residual** $y_i - f(x_i; W_0)$

- For multiple layers, idea is the same: kernel is still
$$k(x, x') = \langle \nabla f(x; W_0), \nabla f(x'; W_0) \rangle$$
but now ∇ uses **all** of the parameters in the network
  - but the linearization results are worse

- Can compute expectation version, even for convolutional nets with pooling
  - github.com/google/neural-tangents

NTK correspondence

- So far we know that:
  - $f(\cdot\,; W) \approx f_{W_0}(\cdot\,; W)$ for wide nets with $W \approx W_0$

NTK correspondence

- So far we know that:
  - $f(\cdot\,; W) \approx f_{W_0}(\cdot\,; W)$ for wide nets with $W \approx W_0$
  - $\{f_{W_0}(\cdot\,; W) - f(\cdot\,; W_0) : W \in \mathbb{R}^p\}$ is an RKHS

NTK correspondence

- So far we know that:
  - $f(\cdot\,; W) \approx f_{W_0}(\cdot\,; W)$ for wide nets with $W \approx W_0$
  - $\{f_{W_0}(\cdot\,; W) - f(\cdot\,; W_0) : W \in \mathbb{R}^p\}$ is an RKHS
    - kernel $k(x, x') = \langle \nabla f(x; W_0), \nabla f(x'; W_0) \rangle$

NTK correspondence

- So far we know that:
  - $f(\cdot\,; W) \approx f_{W_0}(\cdot\,; W)$ for wide nets with $W \approx W_0$
  - $\{f_{W_0}(\cdot\,; W) - f(\cdot\,; W_0) : W \in \mathbb{R}^p\}$ is an RKHS
    - kernel $k(x, x') = \langle \nabla f(x; W_0), \nabla f(x'; W_0) \rangle$
    - Infinite-width limit is universal even for shallow, wide nets

NTK correspondence

- So far we know that:
  - $f(\cdot\,; W) \approx f_{W_0}(\cdot\,; W)$ for wide nets with $W \approx W_0$
  - $\{f_{W_0}(\cdot\,; W) - f(\cdot\,; W_0) : W \in \mathbb{R}^p\}$ is an RKHS
    - kernel $k(x, x') = \langle \nabla f(x; W_0), \nabla f(x'; W_0) \rangle$
    - Infinite-width limit is universal even for shallow, wide nets

- The big remaining result:

NTK correspondence

- So far we know that:
  - $f(\cdot\,; W) \approx f_{W_0}(\cdot\,; W)$ for wide nets with $W \approx W_0$
  - $\{f_{W_0}(\cdot\,; W) - f(\cdot\,; W_0) : W \in \mathbb{R}^p\}$ is an RKHS
    - kernel $k(x, x') = \langle \nabla f(x; W_0), \nabla f(x'; W_0) \rangle$
    - Infinite-width limit is universal even for shallow, wide nets

- The big remaining result:
  - Training f for square loss $\approx$ kernel ridge regression with k

NTK correspondence

We'll optimize $L_S(w)$ based on squared loss $\ell(w, (x, y)) = \frac{1}{2}(f(x; w) - y)^2$

NTK correspondence

We'll optimize $L_S(w)$ based on squared loss $\ell(w,(x,y)) = \frac{1}{2}(f(x;w) - y)^2$

Take **gradient flow** on $L_S(w)$: $\frac{\mathrm{d}w_t}{\mathrm{d}t} = -\nabla L_S(w_t)$

NTK correspondence

We'll optimize $L_S(w)$ based on squared loss $\ell(w, (x, y)) = \frac{1}{2}(f(x; w) - y)^2$

Take **gradient flow** on $L_S(w)$: $\frac{\mathrm{d}w_t}{\mathrm{d}t} = -\nabla L_S(w_t)$

$$= -\frac{1}{n}\sum_{i=1}^{n}(f(x_i; w_t) - y_i)\frac{\partial f(x_i; w_t)}{\partial w}$$

NTK correspondence

We'll optimize $L_S(w)$ based on squared loss $\ell(w, (x, y)) = \frac{1}{2}(f(x; w) - y)^2$

Take **gradient flow** on $L_S(w)$: $\frac{\mathrm{d}w_t}{\mathrm{d}t} = -\nabla L_S(w_t)$

$$= -\frac{1}{n}\sum_{i=1}^{n}(f(x_i; w_t) - y_i)\frac{\partial f(x_i; w_t)}{\partial w}$$

This means training set predictions update as:

NTK correspondence

We'll optimize $L_S(w)$ based on squared loss $\ell(w,(x,y)) = \frac{1}{2}(f(x;w)-y)^2$

Take **gradient flow** on $L_S(w)$: $\frac{\mathrm{d}w_t}{\mathrm{d}t} = -\nabla L_S(w_t)$

$$= -\frac{1}{n}\sum_{i=1}^{n}(f(x_i;w_t)-y_i)\frac{\partial f(x_i;w_t)}{\partial w}$$

This means training set predictions update as:

$$\frac{\mathrm{d}f(x_i;w_t)}{\mathrm{d}t} = -\frac{1}{n}\sum_{j=1}^{n}(f(x_j;w_t)-y_j)\left\langle \frac{\partial f(x_i;w_t)}{\partial w}, \frac{\partial f(x_j;w_t)}{\partial w}\right\rangle$$

NTK correspondence

We'll optimize $L_S(w)$ based on squared loss $\ell(w,(x,y)) = \frac{1}{2}(f(x;w)-y)^2$

Take **gradient flow** on $L_S(w)$: $\frac{\mathrm{d}w_t}{\mathrm{d}t} = -\nabla L_S(w_t)$

$$= -\frac{1}{n}\sum_{i=1}^{n}(f(x_i;w_t)-y_i)\frac{\partial f(x_i;w_t)}{\partial w}$$

This means training set predictions update as:

$$\frac{\mathrm{d}f(x_i;w_t)}{\mathrm{d}t} = -\frac{1}{n}\sum_{j=1}^{n}(f(x_j;w_t)-y_j)\left\langle \frac{\partial f(x_i;w_t)}{\partial w}, \frac{\partial f(x_j;w_t)}{\partial w}\right\rangle$$

So the vector $f_S(t) = \big(f(x_1;w_t),\ldots,f(x_n;w_t)\big)$ evolves as $\frac{\mathrm{d}f_S(t)}{\mathrm{d}t} = -\frac{1}{n}k_{SS}^{(w_t)}(f_S(t)-y)$

NTK correspondence

We'll optimize $L_S(w)$ based on squared loss $\ell(w, (x, y)) = \frac{1}{2}(f(x; w) - y)^2$

Take **gradient flow** on $L_S(w)$: $\frac{\mathrm{d}w_t}{\mathrm{d}t} = -\nabla L_S(w_t)$

$$= -\frac{1}{n}\sum_{i=1}^{n}(f(x_i; w_t) - y_i)\frac{\partial f(x_i; w_t)}{\partial w}$$

This means training set predictions update as:

$$\frac{\mathrm{d}f(x_i; w_t)}{\mathrm{d}t} = -\frac{1}{n}\sum_{j=1}^{n}(f(x_j; w_t) - y_j)\left\langle \frac{\partial f(x_i; w_t)}{\partial w}, \frac{\partial f(x_j; w_t)}{\partial w} \right\rangle$$

So the vector $f_S(t) = \big(f(x_1; w_t), \ldots, f(x_n; w_t)\big)$ evolves as $\frac{\mathrm{d}f_S(t)}{\mathrm{d}t} = -\frac{1}{n}k_{SS}^{(w_t)}(f_S(t) - y)$

If $k_{SS}^{(w_t)} = k_{SS}$ is constant over time, exact same dynamics as kernel (ridgeless) regression

NTK correspondence

- As width $\to \infty$, Arora et al. (2019) show $k_{SS}^{(w_t)}$ is roughly constant over training, and converges to its expectation

NTK correspondence

- As width $\to\infty$, Arora et al. (2019) show $k_{SS}^{(w_t)}$ is roughly constant over training, and converges to its expectation

Theorem 3.2 (Equivalence between trained net and kernel regression). *Suppose* $\sigma(z) = \max(0, z)$, $1/\kappa = \mathrm{poly}(1/\epsilon, \log(n/\delta))$ *and* $d_1 = d_2 = \cdots = d_L = m$ *with* $m \geq \mathrm{poly}(1/\kappa, L, 1/\lambda_0, n, \log(1/\delta))$. *Then for any* $\boldsymbol{x}_{te} \in \mathbb{R}^d$ *with* $\|\boldsymbol{x}_{te}\| = 1$, *with probability at least* $1 - \delta$ *over the random initialization, we have*

$$|f_{nn}(\boldsymbol{x}_{te}) - f_{ntk}(\boldsymbol{x}_{te})| \leq \epsilon.$$

NTK correspondence

- As width $\to \infty$, Arora et al. (2019) show $k_{SS}^{(w_t)}$ is roughly constant over training, and converges to its expectation

Theorem 3.2 (Equivalence between trained net and kernel regression). *Suppose* $\sigma(z) = \max(0, z)$, $1/\kappa = \text{poly}(1/\epsilon, \log(n/\delta))$ *and* $d_1 = d_2 = \cdots = d_L = m$ *with* $m \geq \text{poly}(1/\kappa, L, 1/\lambda_0, n, \log(1/\delta))$. *Then for any* $\boldsymbol{x}_{te} \in \mathbb{R}^d$ *with* $\|\boldsymbol{x}_{te}\| = 1$, *with probability at least* $1 - \delta$ *over the random initialization, we have*

$$|f_{nn}(\boldsymbol{x}_{te}) - f_{ntk}(\boldsymbol{x}_{te})| \leq \epsilon.$$

- Proof is kind of gnarly, but basically amounts to showing kernel being close => gradients are close throughout training => final result is close

NTK correspondence

- As width $\to \infty$, Arora et al. (2019) show $k_{SS}^{(w_t)}$ is roughly constant over training, and converges to its expectation

Theorem 3.2 (Equivalence between trained net and kernel regression). *Suppose* $\sigma(z) = \max(0, z)$, $1/\kappa = \text{poly}(1/\epsilon, \log(n/\delta))$ *and* $d_1 = d_2 = \cdots = d_L = m$ *with* $m \geq \text{poly}(1/\kappa, L, 1/\lambda_0, n, \log(1/\delta))$. *Then for any* $\boldsymbol{x}_{te} \in \mathbb{R}^d$ *with* $\|\boldsymbol{x}_{te}\| = 1$, *with probability at least* $1 - \delta$ *over the random initialization, we have*

$$|f_{nn}(\boldsymbol{x}_{te}) - f_{ntk}(\boldsymbol{x}_{te})| \leq \epsilon.$$

- Proof is kind of gnarly, but basically amounts to showing kernel being close => gradients are close throughout training => final result is close
- Scales network so that initialization has $f_S \approx 0$

Showing the NTK correspondence

- Jacot, Gabriel, and Hongler (2018) (earlier) introduced the term NTK

Showing the NTK correspondence

- Jacot, Gabriel, and Hongler (2018) (earlier) introduced the term NTK
  - Pretty abstract approach: argued that gradient descent on NN parameters corresponds to **kernel gradient descent** in function space

Showing the NTK correspondence

- Jacot, Gabriel, and Hongler (2018) (earlier) introduced the term NTK
  - Pretty abstract approach: argued that gradient descent on NN parameters corresponds to **kernel gradient descent** in function space
  - Doing **gradient flow** gives us an explicit formula for prediction function:

$$f_t(x) = f_0(x) + k_S(x) K_{SS}^{-1} (I - e^{-tK_{SS}})(f_S^* - (f_0)_S)$$

Showing the NTK correspondence

- Jacot, Gabriel, and Hongler (2018) (earlier) introduced the term NTK
  - Pretty abstract approach: argued that gradient descent on NN parameters corresponds to **kernel gradient descent** in function space
  - Doing **gradient flow** gives us an explicit formula for prediction function:

$$f_t(x) = f_0(x) + k_S(x) K_{SS}^{-1} (I - e^{-tK_{SS}})(f_S^* - (f_0)_S)$$

  - and so $f_\infty(x) = f_0(x) + k_S(x)\, k_{SS}^{-1}\, (f_S^* - (f_0)_S)$

Showing the NTK correspondence

- Jacot, Gabriel, and Hongler (2018) (earlier) introduced the term NTK
  - Pretty abstract approach: argued that gradient descent on NN parameters corresponds to **kernel gradient descent** in function space
  - Doing **gradient flow** gives us an explicit formula for prediction function:
$$f_t(x) = f_0(x) + k_S(x) K_{SS}^{-1} (I - e^{-tK_{SS}})(f_S^* - (f_0)_S)$$
  - and so $f_\infty(x) = f_0(x) + k_S(x)\, k_{SS}^{-1}\, (f_S^* - (f_0)_S)$
    - If $f_0(x) = 0$, this is just kernel ridge regression

Showing the NTK correspondence

- Jacot, Gabriel, and Hongler (2018) (earlier) introduced the term NTK
  - Pretty abstract approach: argued that gradient descent on NN parameters corresponds to **kernel gradient descent** in function space
  - Doing **gradient flow** gives us an explicit formula for prediction function:
$$f_t(x) = f_0(x) + k_S(x) K_{SS}^{-1} (I - e^{-tK_{SS}})(f_S^* - (f_0)_S)$$
  - and so $f_\infty(x) = f_0(x) + k_S(x)\, k_{SS}^{-1}\, (f_S^* - (f_0)_S)$
    - If $f_0(x) = 0$, this is just kernel ridge regression
    - In general, it's GP regression with prior mean f_0

Showing the NTK correspondence

- Jacot, Gabriel, and Hongler (2018) (earlier) introduced the term NTK
  - Pretty abstract approach: argued that gradient descent on NN parameters corresponds to **kernel gradient descent** in function space
  - Doing **gradient flow** gives us an explicit formula for prediction function:

$$f_t(x) = f_0(x) + k_S(x) K_{SS}^{-1} (I - e^{-tK_{SS}})(f_S^* - (f_0)_S)$$

  - and so $f_\infty(x) = f_0(x) + k_S(x)\, k_{SS}^{-1}\, (f_S^* - (f_0)_S)$
    - If $f_0(x) = 0$, this is just kernel ridge regression
    - In general, it's GP regression with prior mean f_0
  - Proof actually needs infinite width but only really shows for finite time t

NTK regime and scaling

- Chizat and Bach (2019) argue that scaling is the key thing:

NTK regime and scaling

- Chizat and Bach (2019) argue that scaling is the key thing:
  - Using a small scale “zooms in” on the Taylor expansion, and makes the behaviour more linear

NTK regime and scaling

- Chizat and Bach (2019) argue that scaling is the key thing:
  - Using a small scale “zooms in” on the Taylor expansion, and makes the behaviour more linear
- Can give a short-ish proof of NTK behaviour based on this scaling

NTK regime and scaling

- Chizat and Bach (2019) argue that scaling is the key thing:
  - Using a small scale “zooms in” on the Taylor expansion,
    and makes the behaviour more linear
- Can give a short-ish proof of NTK behaviour based on this scaling
  - Basically, things “look strongly convex”

NTK regime and scaling

- Chizat and Bach (2019) argue that scaling is the key thing:
    - Using a small scale “zooms in” on the Taylor expansion, and makes the behaviour more linear
- Can give a short-ish proof of NTK behaviour based on this scaling
    - Basically, things “look strongly convex”

- But…it’s pretty abstract, and takes a bunch of work to connect back to actual network architectures

NTK regime and scaling

- Chizat and Bach (2019) argue that scaling is the key thing:
  - Using a small scale “zooms in” on the Taylor expansion, and makes the behaviour more linear
- Can give a short-ish proof of NTK behaviour based on this scaling
  - Basically, things “look strongly convex”

- But…it’s pretty abstract, and takes a bunch of work to connect back to actual network architectures
- Telgarsky section 8 gives a simplified proof, but it’s a little bit WIP

So, is deep learning just kernels?

- **No.**

So, is deep learning just kernels?

- **No.**
  - Real neural net optimization isn't in "the NTK regime"

So, is deep learning just kernels?

- **No.**
  - Real neural net optimization isn’t in “the NTK regime”
  - NTK regime *doesn’t allow for feature learning* – the kernel doesn’t change…

So, is deep learning just kernels?

- **No.**
  - Real neural net optimization isn't in "the NTK regime"
  - NTK regime *doesn't allow for feature learning* – the kernel doesn't change…
  - There are problems where NNs provably do better than *any* kernel method possibly could (next time!)

So, is deep learning just kernels?

- **No.**
  - Real neural net optimization isn't in "the NTK regime"
  - NTK regime *doesn't allow for feature learning* – the kernel doesn't change…
  - There are problems where NNs provably do better than *any* kernel method possibly could (next time!)

- But NTK is still useful:

So, is deep learning just kernels?

- **No.**
  - Real neural net optimization isn't in "the NTK regime"
  - NTK regime *doesn't allow for feature learning* – the kernel doesn't change…
  - There are problems where NNs provably do better than *any* kernel method possibly could (next time!)

- But NTK is still useful:
  - AFAIK, the main (only?) proofs that GD optimizes deep networks reasonably

So, is deep learning just kernels?

- **No.**
  - Real neural net optimization isn't in "the NTK regime"
  - NTK regime *doesn't allow for feature learning* – the kernel doesn't change…
  - There are problems where NNs provably do better than *any* kernel method possibly could (next time!)

- But NTK is still useful:
  - AFAIK, the main (only?) proofs that GD optimizes deep networks reasonably
  - Can be practically useful in some settings

So, is deep learning just kernels?

- **No.**
  - Real neural net optimization isn't in "the NTK regime"
  - NTK regime *doesn't allow for feature learning* – the kernel doesn't change…
  - There are problems where NNs provably do better than *any* kernel method possibly could (next time!)

- But NTK is still useful:
  - AFAIK, the main (only?) proofs that GD optimizes deep networks reasonably
  - Can be practically useful in some settings
  - Probably a building block for whatever comes next