Learning and generalization in over-parameterized neural networks, going beyond kernels

Yuanzhi Li

Stanford University

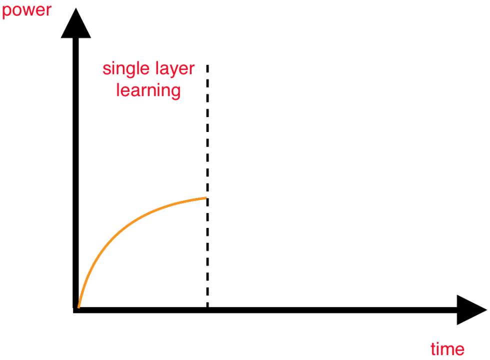
date: Today

A bit of history of machine learning

- Practical machine learning (Phase 1):

A bit of history of machine learning

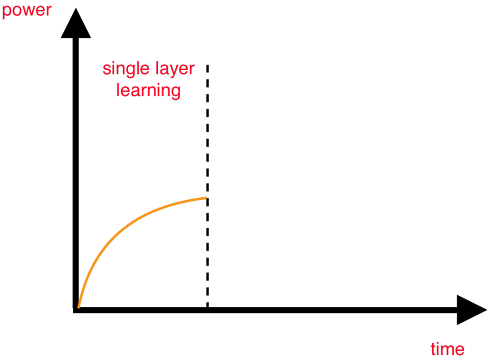
- Practical machine learning (Phase 1):



-

A bit of history of machine learning

- Practical machine learning (Phase 1):



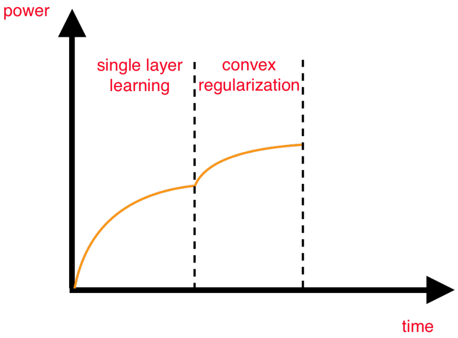
-
- Single layer perceptrons (linear regressions, kernel methods, linear regression over feature mappings).

A bit of history of machine learning

- Practical machine learning (Phase 2):

A bit of history of machine learning

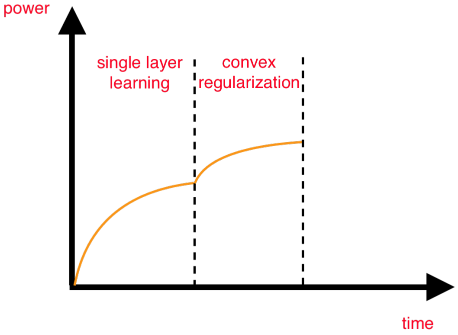
- Practical machine learning (Phase 2):



-

A bit of history of machine learning

- Practical machine learning (Phase 2):



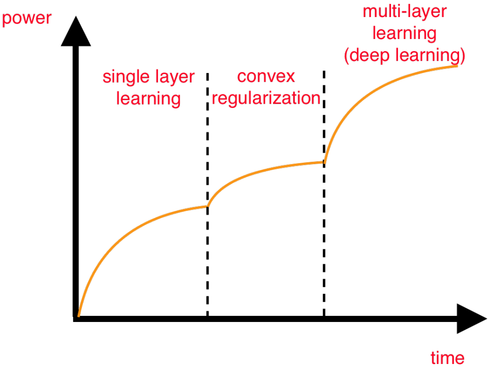
-
- Feature mappings with convex regularizations (e.g. ℓ_1 regularizations for Lasso, nuclear norm regularizations for matrix completion, matrix sensing, PSD regularizations for SOS).

A bit of history of machine learning

- Practical machine learning (Phase 3):

A bit of history of machine learning

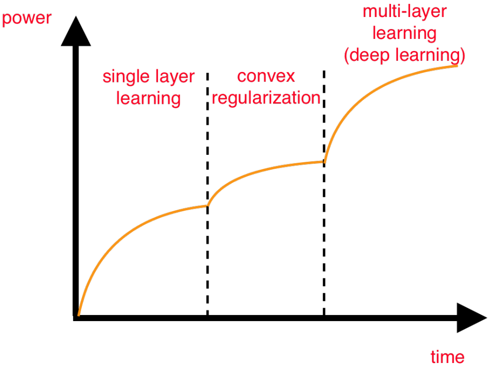
- Practical machine learning (Phase 3):



-

A bit of history of machine learning

- Practical machine learning (Phase 3):



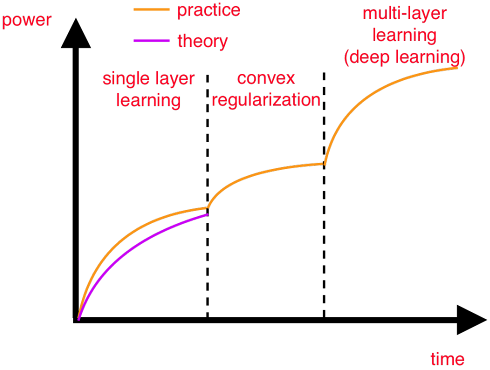
-
- Multi-layer perceptrons (Deep learning) and non-convex algorithms.

A bit of history of machine learning

- Theoretical machine learning (Phase 1):

A bit of history of machine learning

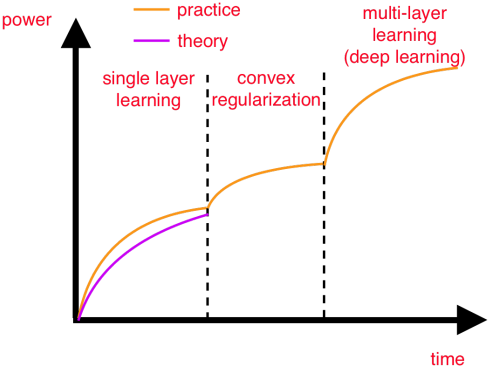
- Theoretical machine learning (Phase 1):



-

A bit of history of machine learning

- Theoretical machine learning (Phase 1):



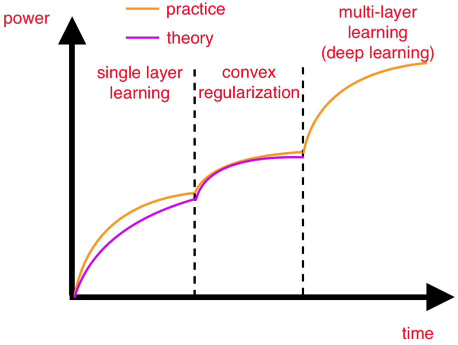
-
- We understand most of the fundamental questions in these methods, both statistically (sample complexity) and computationally.

A bit of history of machine learning

- Theoretical machine learning (Phase 2):

A bit of history of machine learning

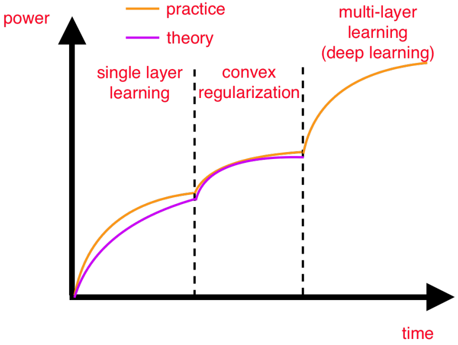
- Theoretical machine learning (Phase 2):



-

A bit of history of machine learning

- Theoretical machine learning (Phase 2):



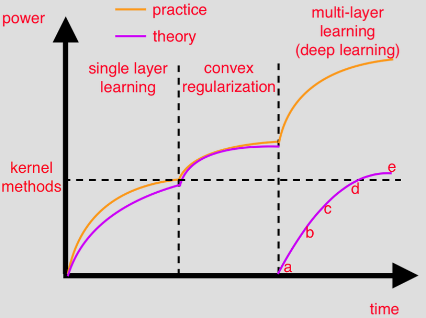
-
- We understand most of the fundamental questions in convex regularizations, both statistically (sample complexity) and computationally.

A bit of history of machine learning

- Theoretical machine learning (Phase 3):

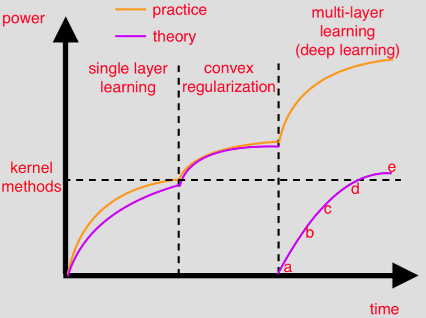
A bit of history of machine learning

- Theoretical machine learning (Phase 3):



-

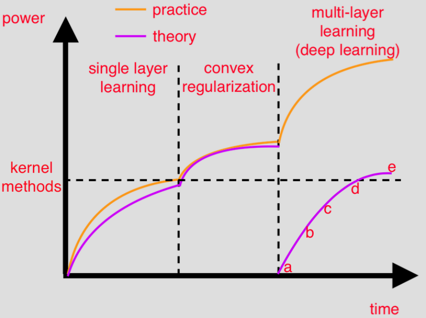
- Theoretical machine learning (Phase 3):



-
- PAC learning setting:

A bit of history of machine learning

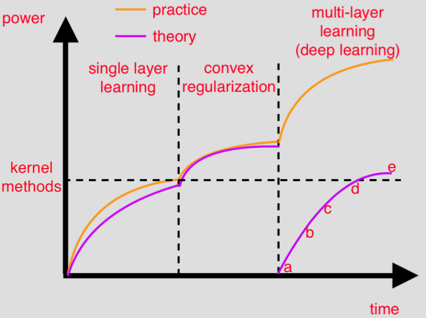
- Theoretical machine learning (Phase 3):



-
- PAC learning setting:
  - (a). Linear networks, neural networks with one neuron (e.g. learning a single ReLU/Sigmoid), neural networks over one dimension inputs.

A bit of history of machine learning

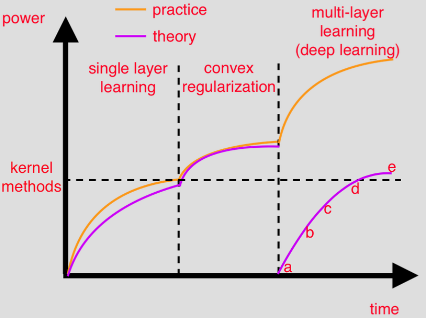
- Theoretical machine learning (Phase 3):



-
- PAC learning setting:
  - (a). Linear networks, neural networks with one neuron (e.g. learning a single ReLU/Sigmoid), neural networks over one dimension inputs.
  - (b). Training last layers of neural networks: Conjugate kernel.

A bit of history of machine learning

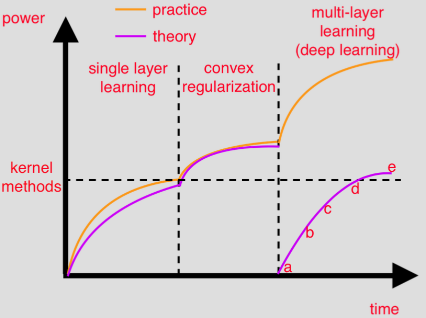
- Theoretical machine learning (Phase 3):



-
- PAC learning setting:
  - (a). Linear networks, neural networks with one neuron (e.g. learning a single ReLU/Sigmoid), neural networks over one dimension inputs.
  - (b). Training last layers of neural networks: Conjugate kernel.
  - (c). Learning neural networks with kernels methods.

A bit of history of machine learning

- Theoretical machine learning (Phase 3):



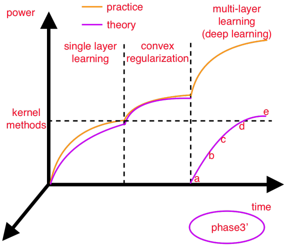
-
- PAC learning setting:
  - (a). Linear networks, neural networks with one neuron (e.g. learning a single ReLU/Sigmoid), neural networks over one dimension inputs.
  - (b). Training last layers of neural networks: Conjugate kernel.
  - (c). Learning neural networks with kernels methods.
  - (d). Neural networks can learn functions that are learnable by kernels: The neural tangent kernel (NTK) approach.

A bit of history of machine learning (side note)

- Theoretical machine learning (Phase 3'):

A bit of history of machine learning (side note)

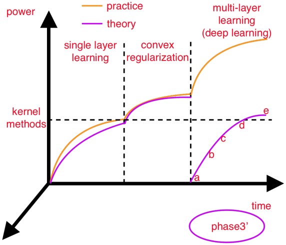
- Theoretical machine learning (Phase 3'):



-

A bit of history of machine learning (side note)

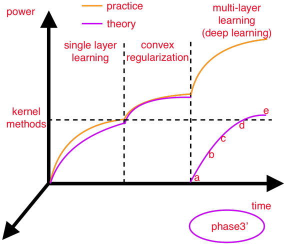
- Theoretical machine learning (Phase 3'):



-
- Learning neural networks with strong distributional assumptions of the inputs(usually spherical Gaussian).

A bit of history of machine learning (side note)

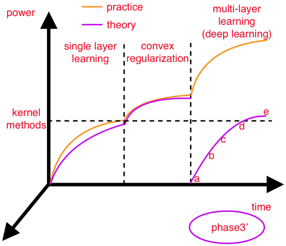
- Theoretical machine learning (Phase 3'):



-
- Learning neural networks with strong distributional assumptions of the inputs(usually spherical Gaussian).
- Unrealistic in practice: In fact, the machine learning community interested in real-world applications finds the this setting questionable:

A bit of history of machine learning (side note)

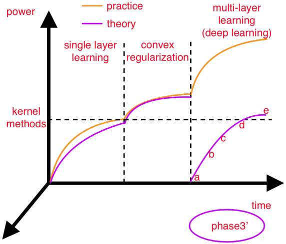
- Theoretical machine learning (Phase 3'):



-
- Learning neural networks with strong distributional assumptions of the inputs(usually spherical Gaussian).
- Unrealistic in practice: In fact, the machine learning community interested in real-world applications finds the this setting questionable:
    - They will punch you in the nose if you try to tell them about algorithms in this framework. – Ryan O'Donnell

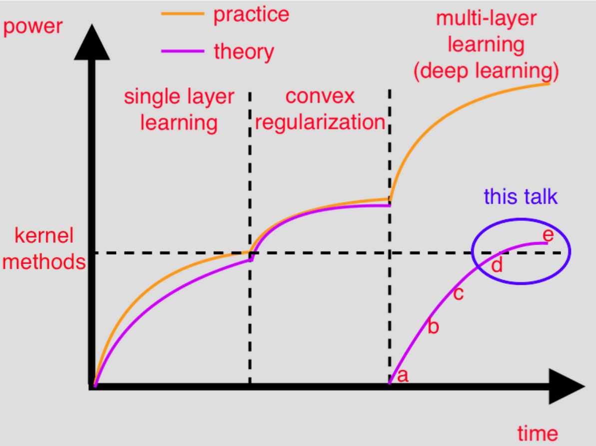
A bit of history of machine learning (side note)

- Theoretical machine learning (Phase 3'):



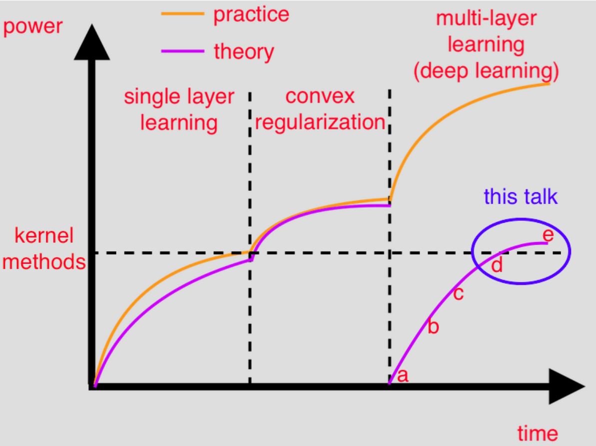
-
- Learning neural networks with strong distributional assumptions of the inputs(usually spherical Gaussian).
- Unrealistic in practice: In fact, the machine learning community interested in real-world applications finds the this setting questionable:
  - They will punch you in the nose if you try to tell them about algorithms in this framework. – Ryan O'Donnell
- This talk focuses on (distribution free) PAC learning.

In this talk

- 

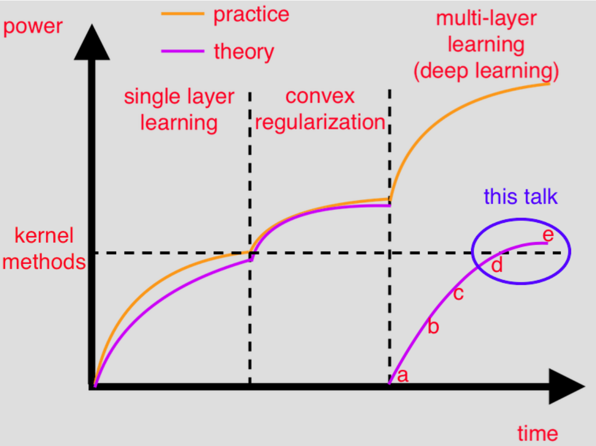
power
practice
theory
multi-layer
learning
(deep learning)
single layer
learning
convex
regularization
this talk
kernel
methods
e
d
c
b
a
time

In this talk

- 

- We will cover (d): neural tangent kernel (NTK) and (e): beyond kernel analysis for neural networks.

In this talk

- 

- We will cover (d): neural tangent kernel (NTK) and (e): beyond kernel analysis for neural networks.
- In (distribution free) PAC learning setting.

Tangent kernel and neural tangent kernel

- Recall that a (mercer) kernel is a function $K(x, y) = \langle \phi(x), \phi(y) \rangle$ over pair of inputs x, y. ϕ is called the feature mapping.

Tangent kernel and neural tangent kernel

- Recall that a (mercer) kernel is a function $K(x,y) = \langle \phi(x), \phi(y) \rangle$ over pair of inputs x, y. ϕ is called the feature mapping.
- Given a (smooth) function $f(W,x)$ with parameters W and input x, we can taylor expand at fixed parameters W_0:

Tangent kernel and neural tangent kernel

- Recall that a (mercer) kernel is a function $K(x, y) = \langle \phi(x), \phi(y) \rangle$ over pair of inputs x, y. ϕ is called the feature mapping.
- Given a (smooth) function $f(W, x)$ with parameters W and input x, we can taylor expand at fixed parameters W_0:
- $f(W, x) = f(W_0, x) + \langle \nabla_W f(W_0, x), W - W_0 \rangle + O(\|W - W_0\|_F^2)$

Tangent kernel and neural tangent kernel

- Recall that a (mercer) kernel is a function $K(x, y) = \langle \phi(x), \phi(y) \rangle$ over pair of inputs x, y. ϕ is called the feature mapping.
- Given a (smooth) function $f(W, x)$ with parameters W and input x, we can taylor expand at fixed parameters W_0:
- $f(W, x) = f(W_0, x) + \langle \nabla_W f(W_0, x), W - W_0 \rangle + O(\|W - W_0\|_F^2)$
- Tangent kernel (given by feature mappings): $\nabla_W f(W_0, x)$.

Tangent kernel and neural tangent kernel

- Recall that a (mercer) kernel is a function $K(x, y) = \langle \phi(x), \phi(y) \rangle$ over pair of inputs x, y. ϕ is called the feature mapping.
- Given a (smooth) function $f(W, x)$ with parameters W and input x, we can taylor expand at fixed parameters W_0:
- $f(W, x) = f(W_0, x) + \langle \nabla_W f(W_0, x), W - W_0 \rangle + O(\|W - W_0\|_F^2)$
- Tangent kernel (given by feature mappings): $\nabla_W f(W_0, x)$.
- Neural tangent kernel (NTK) $K(x, y) = \langle \nabla_W f(W_0, x), \nabla_W f(W_0, y) \rangle$ where f is the neural network, W_0 is usually the (random) initialization.

Neural tangent kernel

Theorem (ALS'18, arXiv:1811.03962)

- *Given a (convolution/fully connected/(BN)) L-layer ReLU neural network $f(W, x)$ with m-neurons per layer, given N distinct training examples, when the following Oops! conditions are satisfied:*

Neural tangent kernel

Theorem (ALS'18, arXiv:1811.03962)

- *Given a (convolution/fully connected/(BN)) L-layer ReLU neural network $f(W, x)$ with m-neurons per layer, given N distinct training examples, when the following Oops! conditions are satisfied:*
    - *Over-parameterization:* $m \geq poly(N, L)$.

Neural tangent kernel

Theorem (ALS'18, arXiv:1811.03962)

- *Given a (convolution/fully connected/(BN)) L-layer ReLU neural network $f(W, x)$ with m-neurons per layer, given N distinct training examples, when the following Oops! conditions are satisfied:*
  - *Over-parameterization: $m \geq poly(N, L)$.*
  - *Proper initialization (Typically $N\left(0, \frac{2}{m}\right)$ for each parameter in each hidden neuron).*

Neural tangent kernel

Theorem (ALS'18, arXiv:1811.03962)

- *Given a (convolution/fully connected/(BN)) L-layer ReLU neural network $f(W, x)$ with m-neurons per layer, given N distinct training examples, when the following Oops! conditions are satisfied:*
  - *Over-parameterization: $m \geq poly(N, L)$.*
  - *Proper initialization (Typically $N\left(0, \frac{2}{m}\right)$ for each parameter in each hidden neuron).*
  - *Small learning rate: $\eta \ll \frac{1}{\sqrt{m}}$.*

Neural tangent kernel

Theorem (ALS'18, arXiv:1811.03962)

- *Given a (convolution/fully connected/(BN)) L-layer ReLU neural network $f(W, x)$ with m-neurons per layer, given N distinct training examples, when the following Oops! conditions are satisfied:*
  - *Over-parameterization: $m \geq poly(N, L)$.*
  - *Proper initialization (Typically $N\left(0, \frac{2}{m}\right)$ for each parameter in each hidden neuron).*
  - *Small learning rate: $\eta \ll \frac{1}{\sqrt{m}}$.*
- *Then SGD can find a point W^* efficiently with small ($o(1)$) training loss (cross entropy, ℓ_2 etc) such that:*
  $f(W^*, x) = f(W_0, x) + \langle \nabla_W f(W_0, x), W^* - W_0 \rangle + o(1)$

Neural tangent kernel

Theorem (ALS'18, arXiv:1811.03962)

- *Given a (convolution/fully connected/(BN)) L-layer ReLU neural network $f(W, x)$ with m-neurons per layer, given N distinct training examples, when the following Oops! conditions are satisfied:*
  - *Over-parameterization: $m \geq poly(N, L)$.*
  - *Proper initialization (Typically $N\left(0, \frac{2}{m}\right)$ for each parameter in each hidden neuron).*
  - *Small learning rate: $\eta \ll \frac{1}{\sqrt{m}}$.*
- *Then SGD can find a point W^* efficiently with small ($o(1)$) training loss (cross entropy, ℓ_2 etc) such that:*
  $f(W^*, x) = f(W_0, x) + \langle \nabla_W f(W_0, x), W^* - W_0 \rangle + o(1)$
- *SGD learns (efficiently) the NTK solution.*

Neural tangent kernel

Theorem (ALS'18, arXiv:1811.03962)

- *Given a (convolution/fully connected/(BN)) L-layer ReLU neural network $f(W, x)$ with m-neurons per layer, given N distinct training examples, when the following Oops! conditions are satisfied:*
  - *Over-parameterization: $m \geq poly(N, L)$.*
  - *Proper initialization (Typically $N\left(0, \frac{2}{m}\right)$ for each parameter in each hidden neuron).*
  - *Small learning rate: $\eta \ll \frac{1}{\sqrt{m}}$.*
- *Then SGD can find a point W^* efficiently with small ($o(1)$) training loss (cross entropy, ℓ_2 etc) such that:*
  $f(W^*, x) = f(W_0, x) + \langle \nabla_W f(W_0, x), W^* - W_0 \rangle + o(1)$
- *SGD learns (efficiently) the NTK solution.*
- *$\nabla_W f(W_0, x)$: the feature mapping, $W^* - W_0$: linear regression.*

Neural tangent kernel

Theorem (ALS'18, arXiv:1811.03962)

- *Given a (convolution/fully connected/(BN)) L-layer ReLU neural network $f(W, x)$ with m-neurons per layer, given N distinct training examples, when the following Oops! conditions are satisfied:*
  - *Over-parameterization: $m \geq poly(N, L)$.*
  - *Proper initialization (Typically $N\left(0, \frac{2}{m}\right)$ for each parameter in each hidden neuron).*
  - *Small learning rate: $\eta \ll \frac{1}{\sqrt{m}}$.*
- *Then SGD can find a point W^* efficiently with small ($o(1)$) training loss (cross entropy, ℓ_2 etc) such that:*
  $f(W^*, x) = f(W_0, x) + \langle \nabla_W f(W_0, x), W^* - W_0 \rangle + o(1)$
- *SGD learns (efficiently) the NTK solution.*
- *$\nabla_W f(W_0, x)$: the feature mapping, $W^* - W_0$: linear regression.*
- *If Oops, then training neural network $f \approx$ linear regression over feature mappings $\phi(x) = \nabla_W f(W_0, x)$.*

Neural tangent kernel

- The theorem is quite simple, but there are two difficulties in the proof:

Neural tangent kernel

- The theorem is quite simple, but there are two difficulties in the proof:
- ReLU activation:
  $f(W, x) = f(W_0, x) + \langle \nabla_W f(W_0, x), W - W_0 \rangle + O(\| W - W_0 \|_F^2)$ for smooth function $f(W, x)$, but ReLU network is not smooth.

Neural tangent kernel

- The theorem is quite simple, but there are two difficulties in the proof:
- ReLU activation: $f(W,x) = f(W_0,x) + \langle \nabla_W f(W_0,x), W - W_0 \rangle + O(\|W - W_0\|_F^2)$ for smooth function $f(W,x)$, but ReLU network is not smooth.
- $m \geq \mathrm{poly}(N, L)$: Even for smooth activation, $O(\|W - W_0\|_F^2) \to \text{smoothness} \times \|W - W_0\|_F^2$.

Neural tangent kernel

- The theorem is quite simple, but there are two difficulties in the proof:
- ReLU activation: $f(W,x) = f(W_0,x) + \langle \nabla_W f(W_0,x), W - W_0 \rangle + O(\|W - W_0\|_F^2)$ for smooth function $f(W,x)$, but ReLU network is not smooth.
- $m \geq \text{poly}(N, L)$: Even for smooth activation, $O(\|W - W_0\|_F^2) \to \text{smoothness} \times \|W - W_0\|_F^2$.
- L-layer network is typically 2^L instead of $\text{poly}(L)$ smooth: Need to use non-worst case bounds.

Neural tangent kernel (proof intuition)

- Two layer network: $\sum_{i=1}^{m} a_i \mathrm{ReLU}(\langle w_i, x \rangle)$.

Neural tangent kernel (proof intuition)

- Two layer network: $\sum_{i=1}^{m} a_i \mathrm{ReLU}(\langle w_i, x \rangle)$.
- Proper initialization: $a_i \sim N(0,1)$, $w_i^{(0)} \sim N\left(0, \frac{2}{m}\right)$. Output at init is $O(1)$ due to cancellation.

Neural tangent kernel (proof intuition)

- Two layer network: $\sum_{i=1}^{m} a_i \mathrm{ReLU}(\langle w_i, x \rangle)$.
- Proper initialization: $a_i \sim N(0,1)$, $w_i^{(0)} \sim N\left(0, \frac{2}{m}\right)$. Output at init is $O(1)$ due to cancellation.
- $\nabla_{w_i} f(W^{(0)}, x) = 1_{\langle w_i^{(0)}, x \rangle \geq 0} x$.

Neural tangent kernel (proof intuition)

- Two layer network: $\sum_{i=1}^{m} a_i \mathrm{ReLU}(\langle w_i, x\rangle)$.
- Proper initialization: $a_i \sim N(0,1)$, $w_i^{(0)} \sim N\left(0, \frac{2}{m}\right)$. Output at init is $O(1)$ due to cancellation.
- $\nabla_{w_i} f(W^{(0)}, x) = 1_{\langle w_i^{(0)}, x\rangle \geq 0} x$.
- Simple observation: There exists $W^* = (w_1^*, \cdots, w_m^*)$ with each $\|w_i^* - w_i^{(0)}\|_2 \leq \frac{1}{m}$ so W^* has small training loss.

Neural tangent kernel (proof intuition)

- Two layer network: $\sum_{i=1}^{m} a_i \mathrm{ReLU}(\langle w_i, x \rangle)$.
- Proper initialization: $a_i \sim N(0,1)$, $w_i^{(0)} \sim N\left(0, \frac{2}{m}\right)$. Output at init is $O(1)$ due to cancellation.
- $\nabla_{w_i} f(W^{(0)}, x) = 1_{\langle w_i^{(0)}, x \rangle \geq 0} x$.
- Simple observation: There exists $W^* = (w_1^*, \cdots, w_m^*)$ with each $\|w_i^* - w_i^{(0)}\|_2 \leq \frac{1}{m}$ so W^* has small training loss.
- $\langle w_i^* - w_i^{(0)}, x \rangle \approx \frac{1}{m}, \langle w_i^{(0)}, x \rangle \approx \frac{1}{\sqrt{m}}$.

Neural tangent kernel (proof intuition)

- Two layer network: $\sum_{i=1}^{m} a_i \mathrm{ReLU}(\langle w_i, x \rangle)$.
- Proper initialization: $a_i \sim N(0,1)$, $w_i^{(0)} \sim N\left(0, \frac{2}{m}\right)$. Output at init is $O(1)$ due to cancellation.
- $\nabla_{w_i} f(W^{(0)}, x) = 1_{\langle w_i^{(0)}, x \rangle \geq 0} x$.
- Simple observation: There exists $W^* = (w_1^*, \cdots, w_m^*)$ with each $\|w_i^* - w_i^{(0)}\|_2 \leq \frac{1}{m}$ so W^* has small training loss.
- $\langle w_i^* - w_i^{(0)}, x \rangle \approx \frac{1}{m}, \langle w_i^{(0)}, x \rangle \approx \frac{1}{\sqrt{m}}$.
- $1_{\langle w_i^*, x \rangle \geq 0} \approx 1_{\langle w_i^{(0)}, x \rangle \geq 0}$: The activation pattern does not change too much, network stays close to its NTK.

Neural tangent kernel (proof intuition)

- Two layer network: $\sum_{i=1}^{m} a_i \mathrm{ReLU}(\langle w_i, x\rangle)$.
- Proper initialization: $a_i \sim N(0,1)$, $w_i^{(0)} \sim N\left(0, \frac{2}{m}\right)$. Output at init is $O(1)$ due to cancellation.
- $\nabla_{w_i} f(W^{(0)}, x) = 1_{\langle w_i^{(0)}, x\rangle \geq 0} x$.
- Simple observation: There exists $W^* = (w_1^*, \cdots, w_m^*)$ with each $\|w_i^* - w_i^{(0)}\|_2 \leq \frac{1}{m}$ so W^* has small training loss.
- $\langle w_i^* - w_i^{(0)}, x\rangle \approx \frac{1}{m}, \langle w_i^{(0)}, x\rangle \approx \frac{1}{\sqrt{m}}$.
- $1_{\langle w_i^*, x\rangle \geq 0} \approx 1_{\langle w_i^{(0)}, x\rangle \geq 0}$: The activation pattern does not change too much, network stays close to its NTK.
- The (explicit) two layer proof is originated in [LL'18], arXiv:1808.01204: Learning Overparameterized Neural Networks via Stochastic Gradient Descent on Structured Data.

- When the following Oops! conditions hold:

Neural tangent kernel (take away message)

- When the following Oops! conditions hold:
- Over-parameterization: $m \geq \text{poly}(N, L)$.

Neural tangent kernel (take away message)

- When the following Oops! conditions hold:
- Over-parameterization: $m \geq \text{poly}(N, L)$.
- Proper initialization: (typically $N\left(0, \frac{2}{m}\right)$ per parameter in each hidden neuron).

Neural tangent kernel (take away message)

- When the following Oops! conditions hold:
- Over-parameterization: $m \geq \text{poly}(N, L)$.
- Proper initialization: (typically $N\left(0, \frac{2}{m}\right)$ per parameter in each hidden neuron).
- Small learning rate: $\eta \ll \frac{1}{\sqrt{m}}$.

Neural tangent kernel (take away message)

- When the following Oops! conditions hold:
- Over-parameterization: $m \geq \text{poly}(N, L)$.
- Proper initialization: (typically $N\left(0, \frac{2}{m}\right)$ per parameter in each hidden neuron).
- Small learning rate: $\eta \ll \frac{1}{\sqrt{m}}$.
- Then a neural network trained by SGD is equivalent to learning via its NTK → Convex optimization.

Neural tangent kernel (take away message)

- When the following Oops! conditions hold:
- Over-parameterization: $m \geq \text{poly}(N, L)$.
- Proper initialization: (typically $N\left(0, \frac{2}{m}\right)$ per parameter in each hidden neuron).
- Small learning rate: $\eta \ll \frac{1}{\sqrt{m}}$.
- Then a neural network trained by SGD is equivalent to learning via its NTK → Convex optimization.
- 

Neural tangent kernel (take away message)

- When the following Oops! conditions hold:
- Over-parameterization: $m \geq \text{poly}(N, L)$.
- Proper initialization: (typically $N\left(0, \frac{2}{m}\right)$ per parameter in each hidden neuron).
- Small learning rate: $\eta \ll \frac{1}{\sqrt{m}}$.
- Then a neural network trained by SGD is equivalent to learning via its NTK → Convex optimization.
- 
- Everyone is happy, except...

Neural tangent kernel (the limitation)

- 

Neural tangent kernel (the limitation)

- 
- If neural network is really NTK, then why do we use neural networks instead of kernel methods?–John N. Tsitsiklis

Neural tangent kernel (the limitation)

- 
- If neural network is really NTK, then why do we use neural networks instead of kernel methods?–John N. Tsitsiklis
- Neural networks do better in practice: On CIFAR-10, best neural network can achieve $\geq 99\%$ accuracy, but kernel methods can only achieve 85% (using random triangle features). Or 77% (using infinite-width NTK), 65% (using 128-channels finite-width NTK).

- 
- If neural network is really NTK, then why do we use neural networks instead of kernel methods?–John N. Tsitsiklis
- Neural networks do better in practice: On CIFAR-10, best neural network can achieve $\geq 99\%$ accuracy, but kernel methods can only achieve 85% (using random triangle features). Or 77% (using infinite-width NTK), 65% (using 128-channels finite-width NTK).
- Can we prove that neural networks do better than NTKs in theory?

This work: Beyond kernel analysis of neural networks

- We prove [AL'19]: Oops!

This work: Beyond kernel analysis of neural networks

- We prove [AL'19]: Oops!
- With proper initialization.

This work: Beyond kernel analysis of neural networks

- We prove [AL'19]: Oops!
- With proper initialization.
- Either change over-parameterization → semi-over-parametrization.

This work: Beyond kernel analysis of neural networks

- We prove [AL'19]: Oops!
- With proper initialization.
- Either change over-parameterization → semi-over-parametrization.
- (Or change small learning rate → large learning rate with noise).

This work: Beyond kernel analysis of neural networks

- We prove [AL'19]: Oops!
- With proper initialization.
- Either change over-parameterization → semi-over-parametrization.
- (Or change small learning rate → large learning rate with noise).
- Then three layer ResNet can (distribution free) PAC learn a concept class more efficient then any kernel methods.

This work: Beyond kernel analysis of neural networks

- We prove [AL'19]: Oops!
- With proper initialization.
- Either change over-parameterization → semi-over-parametrization.
- (Or change small learning rate → large learning rate with noise).
- Then three layer ResNet can (distribution free) PAC learn a concept class more efficient then any kernel methods.
- We prove both upper and lower bounds.

This work: Beyond kernel analysis of neural networks

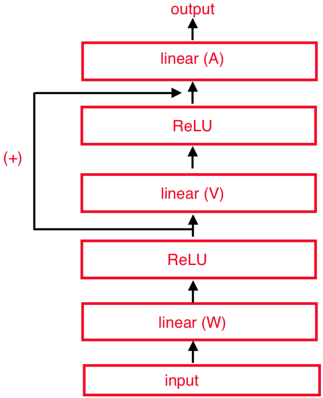
- We prove [AL'19]: Oops!
- With proper initialization.
- Either change over-parameterization → semi-over-parametrization.
- (Or change small learning rate → large learning rate with noise).
- Then three layer ResNet can (distribution free) PAC learn a concept class more efficient then any kernel methods.
- We prove both upper and lower bounds.
- Efficiency: Sample complexity/running time or memory.

The (learner) network

- Three layer ResNet with ReLU activations:

The (learner) network

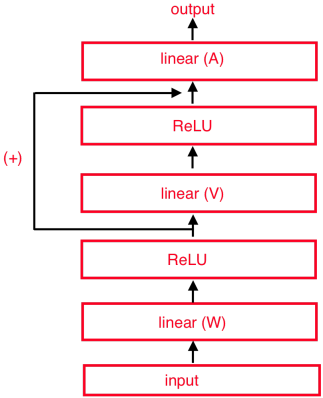
- Three layer ResNet with ReLU activations:



-

The (learner) network

- Three layer ResNet with ReLU activations:



-
- $R(x) = A\mathrm{ReLU}(V\mathrm{ReLU}(Wx)) + A\mathrm{ReLU}(Wx).$

The concept class

- Three layer ResNet with (infinite order) smooth activations:

The concept class

- Three layer ResNet with (infinite order) smooth activations:
- $H(x) = F(x) + \alpha G(F(x))$, $\alpha = o(1)$.

The concept class

- Three layer ResNet with (infinite order) smooth activations:
- $H(x) = F(x) + \alpha G(F(x))$, $\alpha = o(1)$.
- F: base signal (simple and large), $G(F)$: composite signal (small but hard).

The concept class

- Three layer ResNet with (infinite order) smooth activations:
- $H(x) = F(x) + \alpha G(F(x))$, $\alpha = o(1)$.
- F: base signal (simple and large), $G(F)$: composite signal (small but hard).

F

1 + 2 = 3

3 x 4 = 12

16/4 = 3

(ab)c = a(bc)

(a + b) + c
= a + (b + c)

a/(bc) = (a/b)/c

G

((1 + 3 + 2) x 4 - 4 x 2)/4
= 3

-

The concept class

- Three layer ResNet with (infinite order) smooth activations:
- $H(x) = F(x) + \alpha G(F(x))$, $\alpha = o(1)$.
- F: base signal (simple and large), $G(F)$: composite signal (small but hard).

F

1 + 2 = 3

3 x 4 = 12

16/4 = 3

(ab)c = a(bc)

(a + b) + c
= a + (b + c)

a/(bc) = (a/b)/c

G

((1 + 3 + 2) x 4 - 4 x 2)/4
= 3

-
- $F = (f_1, \cdots, f_k), f_j(x) = \sum_i \phi_{f,i,j}(\langle w_{i,j}, x\rangle)$, $\|w_{i,j}\|_2 = 1$,
  $G = (g_1, \cdots, g_k), g_j(x') = \sum_i \phi_{g,i,j}(\langle v_{i,j}, x'\rangle)$, $\|v_{i,j}\|_2 = 1$.

The concept class

- Three layer ResNet with (infinite order) smooth activations:
- $H(x) = F(x) + \alpha G(F(x))$, $\alpha = o(1)$.
- F: base signal (simple and large), $G(F)$: composite signal (small but hard).

F
1 + 2 = 3
3 x 4 = 12
16/4 = 3
(ab)c = a(bc)
(a + b) + c
= a + (b + c)
a/(bc) = (a/b)/c

G
((1 + 3 + 2) x 4 - 4 x 2)/4
= 3

-
- $F = (f_1, \cdots, f_k), f_j(x) = \sum_i \phi_{f,i,j}(\langle w_{i,j}, x\rangle)$, $\|w_{i,j}\|_2 = 1$,
  $G = (g_1, \cdots, g_k), g_j(x') = \sum_i \phi_{g,i,j}(\langle v_{i,j}, x'\rangle)$, $\|v_{i,j}\|_2 = 1$.
- Complexity measure of the functions: For $\phi(\langle w, x\rangle)$ where $\|x\|_2 = \|w\|_2 = 1$, and $\phi(z) = \sum_{i\geq 0} a_i z^i$.

The concept class

- Three layer ResNet with (infinite order) smooth activations:
- $H(x) = F(x) + \alpha G(F(x))$, $\alpha = o(1)$.
- F: base signal (simple and large), $G(F)$: composite signal (small but hard).

F
1 + 2 = 3
3 x 4 = 12
16/4 = 3
(ab)c = a(bc)
(a + b) + c
= a + (b + c)
a/(bc) = (a/b)/c

G
((1 + 3 + 2) x 4 - 4 x 2)/4
= 3

-
- $F = (f_1, \cdots, f_k), f_j(x) = \sum_i \phi_{f,i,j}(\langle w_{i,j}, x\rangle)$, $\|w_{i,j}\|_2 = 1$,
  $G = (g_1, \cdots, g_k), g_j(x') = \sum_i \phi_{g,i,j}(\langle v_{i,j}, x'\rangle)$, $\|v_{i,j}\|_2 = 1$.
- Complexity measure of the functions: For $\phi(\langle w, x\rangle)$ where $\|x\|_2 = \|w\|_2 = 1$, and $\phi(z) = \sum_{i\geq 0} a_i z^i$.
- $\mathcal{C}(\phi) = \sum_{i\geq 0} |a_i| \times i$, $\mathcal{C}(F) = \sum_{i,j} \mathcal{C}(\phi_{f,i,j})$, $\mathcal{C}(G) = \sum_{i,j} \mathcal{C}(\phi_{g,i,j})$.

The concept class

- Three layer ResNet with (infinite order) smooth activations:
- $H(x) = F(x) + \alpha G(F(x))$, $\alpha = o(1)$.
- F: base signal (simple and large), $G(F)$: composite signal (small but hard).

F

1 + 2 = 3

3 x 4 = 12

16/4 = 3

(ab)c = a(bc)

(a + b) + c
= a + (b + c)

a/(bc) = (a/b)/c

G

((1 + 3 + 2) x 4 - 4 x 2)/4
= 3

-
- $F = (f_1, \cdots, f_k), f_j(x) = \sum_i \phi_{f,i,j}(\langle w_{i,j}, x\rangle)$, $\|w_{i,j}\|_2 = 1$,
  $G = (g_1, \cdots, g_k), g_j(x') = \sum_i \phi_{g,i,j}(\langle v_{i,j}, x'\rangle)$, $\|v_{i,j}\|_2 = 1$.
- Complexity measure of the functions: For $\phi(\langle w, x\rangle)$ where $\|x\|_2 = \|w\|_2 = 1$, and $\phi(z) = \sum_{i\geq 0} a_i z^i$.
- $\mathcal{C}(\phi) = \sum_{i\geq 0} |a_i| \times i$, $\mathcal{C}(F) = \sum_{i,j} \mathcal{C}(\phi_{f,i,j})$, $\mathcal{C}(G) = \sum_{i,j} \mathcal{C}(\phi_{g,i,j})$.
- The (sample/time) complexity of learning ϕ using kernel methods.

The Learning algorithm

- $R(x) = A\mathrm{ReLU}(V\mathrm{ReLU}(Wx)) + A\mathrm{ReLU}(Wx)$, loss ℓ_2.

The Learning algorithm

- $R(x) = A\text{ReLU}(V\text{ReLU}(Wx)) + A\text{ReLU}(Wx)$, loss ℓ_2.
- Either change over-parameterization → semi-over-parametrization: $V \to V'A$ (low rank, similar to bottleneck structure in ResNet), then the training algorithm is standard SGD (small learning rate, without any regularization).

The Learning algorithm

- $R(x) = A\mathrm{ReLU}(V\mathrm{ReLU}(Wx)) + A\mathrm{ReLU}(Wx)$, loss ℓ_2.
- Either change over-parameterization → semi-over-parametrization: $V \to V'A$ (low rank, similar to bottleneck structure in ResNet), then the training algorithm is standard SGD (small learning rate, without any regularization).
- Or change small learning rate → large learning rate with noise.

The Learning algorithm

- $R(x) = A\mathrm{ReLU}(V\mathrm{ReLU}(Wx)) + A\mathrm{ReLU}(Wx)$, loss ℓ_2.
- Either change over-parameterization → semi-over-parametrization: $V \to V'A$ (low rank, similar to bottleneck structure in ResNet), then the training algorithm is standard SGD (small learning rate, without any regularization).
- Or change small learning rate → large learning rate with noise.
  - $v_i = (1 - \eta\lambda_v)v_i - \eta(\nabla_{v_i}\mathrm{Loss} + \xi_{v_i})$,
    $w_i = (1 - \eta\lambda_w)w_i - \eta(\nabla_{w_i}\mathrm{Loss} + \xi_{w_i})$.

The Learning algorithm

- $R(x) = A\mathrm{ReLU}(V\mathrm{ReLU}(Wx)) + A\mathrm{ReLU}(Wx)$, loss ℓ_2.
- Either change over-parameterization → semi-over-parametrization: $V \to V'A$ (low rank, similar to bottleneck structure in ResNet), then the training algorithm is standard SGD (small learning rate, without any regularization).
- Or change small learning rate → large learning rate with noise.
  - $v_i = (1 - \eta\lambda_v)v_i - \eta(\nabla_{v_i}\mathrm{Loss} + \xi_{v_i})$,
    $w_i = (1 - \eta\lambda_w)w_i - \eta(\nabla_{w_i}\mathrm{Loss} + \xi_{w_i})$.
  - $\xi_{v_i} \sim N(0, \mathbb{E}_x[\mathrm{ReLU}(Wx)\mathrm{ReLU}(Wx)^\top])$, $\xi_{w_i} \sim N(0, \mathbb{E}_x[xx^\top])$.

The Learning algorithm

- $R(x) = A\mathrm{ReLU}(V\mathrm{ReLU}(Wx)) + A\mathrm{ReLU}(Wx)$, loss ℓ_2.
- Either change over-parameterization → semi-over-parametrization: $V \to V'A$ (low rank, similar to bottleneck structure in ResNet), then the training algorithm is standard SGD (small learning rate, without any regularization).
- Or change small learning rate → large learning rate with noise.
  - $v_i = (1 - \eta\lambda_v)v_i - \eta(\nabla_{v_i}\mathrm{Loss} + \xi_{v_i})$,
    $w_i = (1 - \eta\lambda_w)w_i - \eta(\nabla_{w_i}\mathrm{Loss} + \xi_{w_i})$.
  - $\xi_{v_i} \sim N(0, \mathbb{E}_x[\mathrm{ReLU}(Wx)\mathrm{ReLU}(Wx)^\top])$, $\xi_{w_i} \sim N(0, \mathbb{E}_x[xx^\top])$.
  - Gaussian approximation of the SGD-type of noise.

The Learning algorithm

- $R(x) = A\mathrm{ReLU}(V\mathrm{ReLU}(Wx)) + A\mathrm{ReLU}(Wx)$, loss ℓ_2.
- Either change over-parameterization → semi-over-parametrization: $V \to V'A$ (low rank, similar to bottleneck structure in ResNet), then the training algorithm is standard SGD (small learning rate, without any regularization).
- Or change small learning rate → large learning rate with noise.
  - $v_i = (1 - \eta\lambda_v)v_i - \eta(\nabla_{v_i}\mathrm{Loss} \ + \xi_{v_i})$,
    $w_i = (1 - \eta\lambda_w)w_i - \eta(\nabla_{w_i}\mathrm{Loss} \ + \xi_{w_i})$.
  - $\xi_{v_i} \sim N(0, \mathbb{E}_x[\mathrm{ReLU}(Wx)\mathrm{ReLU}(Wx)^\top])$, $\xi_{w_i} \sim N(0, \mathbb{E}_x[xx^\top])$.
  - Gaussian approximation of the SGD-type of noise.
  - Intuitively, eventually $\mathrm{ReLU}(Wx)$ aligns with A through the signal $A\mathrm{ReLU}(Wx)$, so $\mathbb{E}_x[\mathrm{ReLU}(Wx)\mathrm{ReLU}(Wx)^\top]) \approx AA^\top$.

The Theorem(upper and lower bound)

The Theorem(upper and lower bound)

Theorem (AL'19, arXiv:1905.10337)

Over any distribution of x on the unit ball, three layer ResNet R learns $H(x) = F(x) + \alpha G(F(x))$ up to generalization error $O(\alpha^2)$ (meaning $H(x) - R(x) \approx \alpha^2$) under following conditions:

The Theorem(upper and lower bound)

Theorem (AL'19, arXiv:1905.10337)

Over any distribution of x on the unit ball, three layer ResNet R learns $H(x) = F(x) + \alpha G(F(x))$ up to generalization error $O(\alpha^2)$ (meaning $H(x) - R(x) \approx \alpha^2$) under following conditions:

- *Number of neurons: poly$(\mathcal{C}(F), \mathcal{C}(G), 1/\alpha)$.*

The Theorem(upper and lower bound)

Theorem (AL'19, arXiv:1905.10337)

Over any distribution of x on the unit ball, three layer ResNet R learns $H(x) = F(x) + \alpha G(F(x))$ up to generalization error $O(\alpha^2)$ (meaning $H(x) - R(x) \approx \alpha^2$) under following conditions:

- *Number of neurons: poly$(\mathcal{C}(F), \mathcal{C}(G), 1/\alpha)$.*
- *Sample complexity: $O\left(\mathcal{C}(F)^2 + \mathcal{C}(G)^2\right)/\alpha^4$.*

The Theorem(upper and lower bound)

Theorem (AL'19, arXiv:1905.10337)

Over any distribution of x on the unit ball, three layer ResNet R learns $H(x) = F(x) + \alpha G(F(x))$ up to generalization error $O(\alpha^2)$ (meaning $H(x) - R(x) \approx \alpha^2$) under following conditions:

- *Number of neurons: poly*$(\mathcal{C}(F), \mathcal{C}(G), 1/\alpha)$.
- *Sample complexity:* $O\left(\mathcal{C}(F)^2 + \mathcal{C}(G)^2\right)/\alpha^4$.
- *Running time: poly*$(\mathcal{C}(F), \mathcal{C}(G), 1/\alpha)$.

The Theorem(upper and lower bound)

Theorem (AL'19, arXiv:1905.10337)

Over any distribution of x on the unit ball, three layer ResNet R learns $H(x) = F(x) + \alpha G(F(x))$ up to generalization error $O(\alpha^2)$ (meaning $H(x) - R(x) \approx \alpha^2$) under following conditions:

- *Number of neurons: poly$(\mathcal{C}(F), \mathcal{C}(G), 1/\alpha)$.*
- *Sample complexity: $O\left(\mathcal{C}(F)^2 + \mathcal{C}(G)^2\right)/\alpha^4$.*
- *Running time: poly$(\mathcal{C}(F), \mathcal{C}(G), 1/\alpha)$.*

The Theorem(upper and lower bound)

Theorem (AL'19, arXiv:1905.10337)

Over any distribution of x on the unit ball, three layer ResNet R learns $H(x) = F(x) + \alpha G(F(x))$ up to generalization error $O(\alpha^2)$ (meaning $H(x) - R(x) \approx \alpha^2$) under following conditions:

- *Number of neurons: poly*$(\mathcal{C}(F), \mathcal{C}(G), 1/\alpha)$.
- *Sample complexity:* $O\left(\mathcal{C}(F)^2 + \mathcal{C}(G)^2\right)/\alpha^4$.
- *Running time: poly*$(\mathcal{C}(F), \mathcal{C}(G), 1/\alpha)$.

Theorem (AL'19)

Over some distributions of x on the unit ball and for some class of G, F, any kernel methods that learns the concept function $H(x) = F(x) + \alpha G(F(x))$ up to generalization error $o(\alpha)$ must use

The Theorem(upper and lower bound)

Theorem (AL'19, arXiv:1905.10337)

Over any distribution of x on the unit ball, three layer ResNet R learns $H(x) = F(x) + \alpha G(F(x))$ up to generalization error $O(\alpha^2)$ (meaning $H(x) - R(x) \approx \alpha^2$) under following conditions:

- *Number of neurons: poly$(\mathcal{C}(F), \mathcal{C}(G), 1/\alpha)$.*
- *Sample complexity: $O\left(\mathcal{C}(F)^2 + \mathcal{C}(G)^2\right)/\alpha^4$.*
- *Running time: poly$(\mathcal{C}(F), \mathcal{C}(G), 1/\alpha)$.*

Theorem (AL'19)

Over some distributions of x on the unit ball and for some class of G, F, any kernel methods that learns the concept function $H(x) = F(x) + \alpha G(F(x))$ up to generalization error $o(\alpha)$ must use

- *Sample complexity: $\Omega\left(\mathcal{C}(G(F))^2\right)$.*

The Theorem(upper and lower bound)

Theorem (AL'19, arXiv:1905.10337)

Over any distribution of x on the unit ball, three layer ResNet R learns $H(x) = F(x) + \alpha G(F(x))$ up to generalization error $O(\alpha^2)$ (meaning $H(x) - R(x) \approx \alpha^2$) under following conditions:

- *Number of neurons: poly$(\mathcal{C}(F), \mathcal{C}(G), 1/\alpha)$.*
- *Sample complexity: $O\left(\mathcal{C}(F)^2 + \mathcal{C}(G)^2\right)/\alpha^4$.*
- *Running time: poly$(\mathcal{C}(F), \mathcal{C}(G), 1/\alpha)$.*

Theorem (AL'19)

Over some distributions of x on the unit ball and for some class of G, F, any kernel methods that learns the concept function $H(x) = F(x) + \alpha G(F(x))$ up to generalization error $o(\alpha)$ must use

- *Sample complexity: $\Omega\left(\mathcal{C}(G(F))^2\right)$.*

- Kernel methods: $K(x, y) = \langle \Phi(x), \Phi(y) \rangle$, predict $\langle \Phi(x), \sum_i w_i \Phi(x_i) \rangle$ for some weights $\{w_i\}$ over training examples $\{x_i, y_i\}$.

Neural network vs Kernels

- In some case, the complexity of $\mathcal{C}(G(F))$ can be *much larger* than $\mathcal{C}(G)$,$\mathcal{C}(F)$:

Neural network vs Kernels

- In some case, the complexity of $\mathcal{C}(G(F))$ can be *much larger* than $\mathcal{C}(G)$,$\mathcal{C}(F)$:
- $F(x) = \sqrt{d}(\langle w, x\rangle)$, $\|w\|_2 = 1, x$ is a random unit vector in dimension d: $\mathcal{C}(F) = \sqrt{d}$.

Neural network vs Kernels

- In some case, the complexity of $\mathcal{C}(G(F))$ can be *much larger* than $\mathcal{C}(G)$,$\mathcal{C}(F)$:
- $F(x) = \sqrt{d}(\langle w, x\rangle)$, $\|w\|_2 = 1, x$ is a random unit vector in dimension d: $\mathcal{C}(F) = \sqrt{d}$.
- $G(z) = z^{10}$, $\mathcal{C}(G) = O(1)$.

Neural network vs Kernels

- In some case, the complexity of $\mathcal{C}(G(F))$ can be *much larger* than $\mathcal{C}(G)$,$\mathcal{C}(F)$:
- $F(x) = \sqrt{d}(\langle w, x\rangle)$, $\|w\|_2 = 1, x$ is a random unit vector in dimension d: $\mathcal{C}(F) = \sqrt{d}$.
- $G(z) = z^{10}$, $\mathcal{C}(G) = O(1)$.
- $G(F(x)) = (\sqrt{d})^{10}(\langle w, x\rangle)^{10}$, so $\mathcal{C}(G(F)) = d^5$.

Neural network vs Kernels

- In some case, the complexity of $\mathcal{C}(G(F))$ can be *much larger* than $\mathcal{C}(G)$,$\mathcal{C}(F)$:
- $F(x) = \sqrt{d}(\langle w, x\rangle)$, $\|w\|_2 = 1, x$ is a random unit vector in dimension d: $\mathcal{C}(F) = \sqrt{d}$.
- $G(z) = z^{10}$, $\mathcal{C}(G) = O(1)$.
- $G(F(x)) = (\sqrt{d})^{10}(\langle w, x\rangle)^{10}$, so $\mathcal{C}(G(F)) = d^5$.
- ResNet learns $\sqrt{d}(\langle w, x\rangle) + \alpha\left(\sqrt{d}(\langle w, x\rangle)\right)^{10}$ up to accuracy $O(\alpha^2)$ with $O(d/\text{poly}(\alpha))$ samples, for any $\alpha = o(1)$.

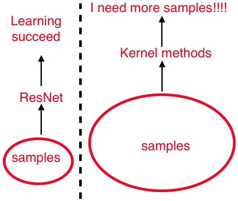
Neural network vs Kernels

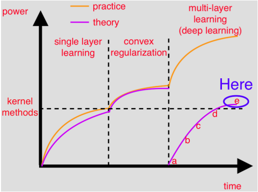
- In some case, the complexity of $\mathcal{C}(G(F))$ can be *much larger* than $\mathcal{C}(G)$,$\mathcal{C}(F)$:
- $F(x) = \sqrt{d}(\langle w, x\rangle)$, $\|w\|_2 = 1, x$ is a random unit vector in dimension d: $\mathcal{C}(F) = \sqrt{d}$.
- $G(z) = z^{10}$, $\mathcal{C}(G) = O(1)$.
- $G(F(x)) = (\sqrt{d})^{10}(\langle w, x\rangle)^{10}$, so $\mathcal{C}(G(F)) = d^5$.
- ResNet learns $\sqrt{d}(\langle w, x\rangle) + \alpha\left(\sqrt{d}(\langle w, x\rangle)\right)^{10}$ up to accuracy $O(\alpha^2)$ with $O(d/\text{poly}(\alpha))$ samples, for any $\alpha = o(1)$.
- Kernel methods must use d^{10} samples to learn up to accuracy $o(\alpha)$

Neural network vs Kernels

- In some case, the complexity of $\mathcal{C}(G(F))$ can be *much larger* than $\mathcal{C}(G)$,$\mathcal{C}(F)$:
- $F(x) = \sqrt{d}(\langle w, x\rangle)$, $\|w\|_2 = 1, x$ is a random unit vector in dimension d: $\mathcal{C}(F) = \sqrt{d}$.
- $G(z) = z^{10}$, $\mathcal{C}(G) = O(1)$.
- $G(F(x)) = (\sqrt{d})^{10}(\langle w, x\rangle)^{10}$, so $\mathcal{C}(G(F)) = d^5$.
- ResNet learns $\sqrt{d}(\langle w, x\rangle) + \alpha\left(\sqrt{d}(\langle w, x\rangle)\right)^{10}$ up to accuracy $O(\alpha^2)$ with $O(d/\text{poly}(\alpha))$ samples, for any $\alpha = o(1)$.
- Kernel methods must use d^{10} samples to learn up to accuracy $o(\alpha)$
- So we provably have the following pictures:

Neural network vs Kernels

- In some case, the complexity of $\mathcal{C}(G(F))$ can be *much larger* than $\mathcal{C}(G), \mathcal{C}(F)$:
- $F(x) = \sqrt{d}(\langle w, x\rangle)$, $\|w\|_2 = 1, x$ is a random unit vector in dimension d: $\mathcal{C}(F) = \sqrt{d}$.
- $G(z) = z^{10}$, $\mathcal{C}(G) = O(1)$.
- $G(F(x)) = (\sqrt{d})^{10}(\langle w, x\rangle)^{10}$, so $\mathcal{C}(G(F)) = d^5$.
- ResNet learns $\sqrt{d}(\langle w, x\rangle) + \alpha\left(\sqrt{d}(\langle w, x\rangle)\right)^{10}$ up to accuracy $O(\alpha^2)$ with $O(d/\mathrm{poly}(\alpha))$ samples, for any $\alpha = o(1)$.
- Kernel methods must use d^{10} samples to learn up to accuracy $o(\alpha)$
- So we provably have the following pictures:
- 



Extension to feature mappings

Extension to feature mappings

Theorem (AL'19)

Over some distributions of x on the unit ball and for some class of G, F, linear regression over any feature mappings (with any regularization) that learns the concept function $H(x) = F(x) + \alpha G(F(x))$ up to generalization error $o(\alpha)$ must use

Extension to feature mappings

Theorem (AL'19)

Over some distributions of x on the unit ball and for some class of G, F, linear regression over any feature mappings (with any regularization) that learns the concept function $H(x) = F(x) + \alpha G(F(x))$ up to generalization error $o(\alpha)$ must use

- *Number of features: $\Omega\left(\mathcal{C}(G(F))^2\right)$.*

Extension to feature mappings

Theorem (AL'19)

Over some distributions of x on the unit ball and for some class of G, F, linear regression over any feature mappings (with any regularization) that learns the concept function $H(x) = F(x) + \alpha G(F(x))$ up to generalization error $o(\alpha)$ must use

- *Number of features: $\Omega\left(\mathcal{C}(G(F))^2\right)$.*

- Running times is slow: $\Omega\left(\mathcal{C}(G(F))^2\right)$ v.s. poly$(\mathcal{C}(F), \mathcal{C}(G), 1/\alpha)$ for Neural networks.

Extension to feature mappings

Theorem (AL'19)

Over some distributions of x on the unit ball and for some class of G, F, linear regression over any feature mappings (with any regularization) that learns the concept function $H(x) = F(x) + \alpha G(F(x))$ up to generalization error $o(\alpha)$ must use

- *Number of features: $\Omega\left(\mathcal{C}(G(F))^2\right)$.*

- Running times is slow: $\Omega\left(\mathcal{C}(G(F))^2\right)$ v.s. poly$(\mathcal{C}(F), \mathcal{C}(G), 1/\alpha)$ for Neural networks.
- Upper bound extension to convolution nets such as $H(x) = \sum_j F(x^{(j)}) + \alpha G(\sum_j v_j F(x^{(j)}))$ is also easy.

Extension to feature mappings

Theorem (AL'19)

Over some distributions of x on the unit ball and for some class of G, F, linear regression over any feature mappings (with any regularization) that learns the concept function $H(x) = F(x) + \alpha G(F(x))$ up to generalization error $o(\alpha)$ must use

- *Number of features:* $\Omega\left(\mathcal{C}(G(F))^2\right)$.

- Running times is slow: $\Omega\left(\mathcal{C}(G(F))^2\right)$ v.s. $\mathrm{poly}(\mathcal{C}(F), \mathcal{C}(G), 1/\alpha)$ for Neural networks.
- Upper bound extension to convolution nets such as $H(x) = \sum_j F(x^{(j)}) + \alpha G(\sum_j v_j F(x^{(j)}))$ is also easy.
- Upper bound can also be extended: $O(\alpha^2) \to \approx 0$ with distributional assumptions.

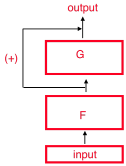
Intuition behind the result

- ResNet is performing implicit hierarchical learning.

Intuition behind the result

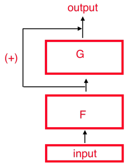
- ResNet is performing implicit hierarchical learning.
- ResNet first learns $F(x)$ using (the first layer) NTK, then it feeds $F(x)$ to the second layer to learn G using (the second layer) NTK. So the complexities are $\mathcal{C}(F), \mathcal{C}(G)$.

- ResNet is performing implicit hierarchical learning.
- ResNet first learns $F(x)$ using (the first layer) NTK, then it feeds $F(x)$ to the second layer to learn G using (the second layer) NTK. So the complexities are $\mathcal{C}(F), \mathcal{C}(G)$.



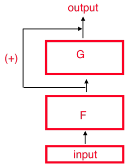
-

Intuition behind the result

- ResNet is performing implicit hierarchical learning.
- ResNet first learns $F(x)$ using (the first layer) NTK, then it feeds $F(x)$ to the second layer to learn G using (the second layer) NTK. So the complexities are $\mathcal{C}(F), \mathcal{C}(G)$.
- 

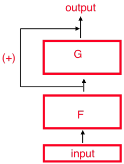
- Kernel methods learn $G(F)$ from scratch, as if there is NO $F(x)$, which is inefficient.

Intuition behind the result

- ResNet is performing implicit hierarchical learning.
- ResNet first learns $F(x)$ using (the first layer) NTK, then it feeds $F(x)$ to the second layer to learn G using (the second layer) NTK. So the complexities are $\mathcal{C}(F), \mathcal{C}(G)$.
- 

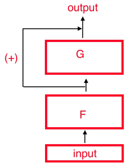
- Kernel methods learn $G(F)$ from scratch, as if there is NO $F(x)$, which is inefficient.
- Note (implicit hierarchical learning):

Intuition behind the result

- ResNet is performing implicit hierarchical learning.
- ResNet first learns $F(x)$ using (the first layer) NTK, then it feeds $F(x)$ to the second layer to learn G using (the second layer) NTK. So the complexities are $\mathcal{C}(F), \mathcal{C}(G)$.
- 

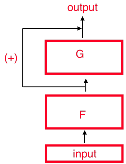
- Kernel methods learn $G(F)$ from scratch, as if there is NO $F(x)$, which is inefficient.
- Note (implicit hierarchical learning):
  - The two layers of ResNet are trained *simultaneously*.

Intuition behind the result

- ResNet is performing implicit hierarchical learning.
- ResNet first learns $F(x)$ using (the first layer) NTK, then it feeds $F(x)$ to the second layer to learn G using (the second layer) NTK. So the complexities are $\mathcal{C}(F), \mathcal{C}(G)$.
- 

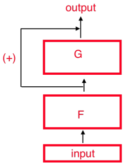
- Kernel methods learn $G(F)$ from scratch, as if there is NO $F(x)$, which is inefficient.
- Note (implicit hierarchical learning):
  - The two layers of ResNet are trained *simultaneously*.
  - There's no change of learning rate during the course of training.

Intuition behind the result

- ResNet is performing implicit hierarchical learning.
- ResNet first learns $F(x)$ using (the first layer) NTK, then it feeds $F(x)$ to the second layer to learn G using (the second layer) NTK. So the complexities are $\mathcal{C}(F), \mathcal{C}(G)$.
- 

- Kernel methods learn $G(F)$ from scratch, as if there is NO $F(x)$, which is inefficient.
- Note (implicit hierarchical learning):
  - The two layers of ResNet are trained *simultaneously*.
  - There's no change of learning rate during the course of training.
  - No regularization on the first hidden layer (F) (first variant).

Intuition behind the result

- ResNet is performing implicit hierarchical learning.
- ResNet first learns $F(x)$ using (the first layer) NTK, then it feeds $F(x)$ to the second layer to learn G using (the second layer) NTK. So the complexities are $\mathcal{C}(F), \mathcal{C}(G)$.
- 

- Kernel methods learn $G(F)$ from scratch, as if there is NO $F(x)$, which is inefficient.
- Note (implicit hierarchical learning):
  - The two layers of ResNet are trained *simultaneously*.
  - There's no change of learning rate during the course of training.
  - No regularization on the first hidden layer (F) (first variant).
  - ResNet needs to distribute the learning of F, G automatically between two layers and avoid *over-fitting*.

Intuition 2

- Implicit hierarchical learning: ResNet first learns $F(x)$ using NTK up to accuracy α from $F(x) + \alpha G(F(x))$

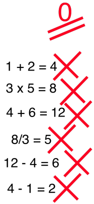
Intuition 2

- Implicit hierarchical learning: ResNet first learns $F(x)$ using NTK up to accuracy α from $F(x) + \alpha G(F(x))$
- Then it feeds $F(x) \pm \alpha$ to the second layer to learn G using NTK up to accuracy α.

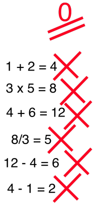
Intuition 2

- Implicit hierarchical learning: ResNet first learns $F(x)$ using NTK up to accuracy α from $F(x) + \alpha G(F(x))$
- Then it feeds $F(x) \pm \alpha$ to the second layer to learn G using NTK up to accuracy α.
- So the total accuracy is α^2.

Intuition 2

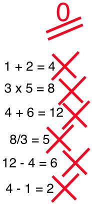
- Implicit hierarchical learning: ResNet first learns $F(x)$ using NTK up to accuracy α from $F(x) + \alpha G(F(x))$
- Then it feeds $F(x) \pm \alpha$ to the second layer to learn G using NTK up to accuracy α.
- So the total accuracy is α^2.
- 

Intuition 2

- Implicit hierarchical learning: ResNet first learns $F(x)$ using NTK up to accuracy α from $F(x) + \alpha G(F(x))$
- Then it feeds $F(x) \pm \alpha$ to the second layer to learn G using NTK up to accuracy α.
- So the total accuracy is α^2.
- 

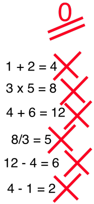
- The total accuracy is α since F is only learnt (in the first layer) up to accuracy α.

Intuition 2

- Implicit hierarchical learning: ResNet first learns $F(x)$ using NTK up to accuracy α from $F(x) + \alpha G(F(x))$
- Then it feeds $F(x) \pm \alpha$ to the second layer to learn G using NTK up to accuracy α.
- So the total accuracy is α^2.
- 

- The total accuracy is α since F is only learnt (in the first layer) up to accuracy α.
- The second layer needs to help the first layer to "correct" the error in F.

Intuition 2

- Implicit hierarchical learning: ResNet first learns $F(x)$ using NTK up to accuracy α from $F(x) + \alpha G(F(x))$
- Then it feeds $F(x) \pm \alpha$ to the second layer to learn G using NTK up to accuracy α.
- So the total accuracy is α^2.
- 

- The total accuracy is α since F is only learnt (in the first layer) up to accuracy α.
- The second layer needs to help the first layer to "correct" the error in F.
- It is important that both layers are trained together.

Key message

- Both layers are still (individual) NTK, but after learning, the first layer feeds better inputs to the second layer NTK.

Key message

- Both layers are still (individual) NTK, but after learning, the first layer feeds better inputs to the second layer NTK.
- Implicit hierarchical learning.

Experiments

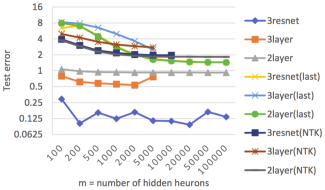
- $x \in \{-1, 1\}^{30}$, $y \in \mathbb{R}^8$, $\alpha = 0.3$. $H(x) = \beta F(x) + \alpha G(F(x))$, $\beta = 1$.

Experiments

- $x \in \{-1,1\}^{30}$, $y \in \mathbb{R}^8$, $\alpha = 0.3$. $H(x) = \beta F(x) + \alpha G(F(x))$, $\beta = 1$.
- $F(x) = (x_1 x_2, \cdots, x_{15} x_{16})$,
  $G(y) = (y_1 y_2 y_3 y_4, \cdots, y_1 y_2 y_3 y_4, y_5 y_6 y_7 y_8, \cdots, y_5 y_6 y_7 y_8)$.

Experiments

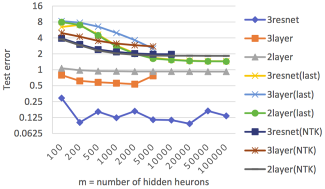
- $x \in \{-1,1\}^{30}$, $y \in \mathbb{R}^8$, $\alpha = 0.3$. $H(x) = \beta F(x) + \alpha G(F(x))$, $\beta = 1$.
- $F(x) = (x_1 x_2, \cdots, x_{15} x_{16})$,
  $G(y) = (y_1 y_2 y_3 y_4, \cdots, y_1 y_2 y_3 y_4, y_5 y_6 y_7 y_8, \cdots, y_5 y_6 y_7 y_8)$.



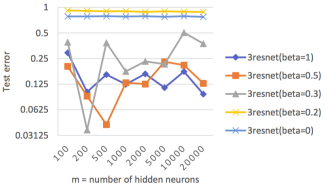
(a) $N = 1000$ and vary m

-

- $x \in \{-1, 1\}^{30}$, $y \in \mathbb{R}^8$, $\alpha = 0.3$. $H(x) = \beta F(x) + \alpha G(F(x))$, $\beta = 1$.
- $F(x) = (x_1 x_2, \cdots, x_{15} x_{16})$,
  $G(y) = (y_1 y_2 y_3 y_4, \cdots, y_1 y_2 y_3 y_4, y_5 y_6 y_7 y_8, \cdots, y_5 y_6 y_7 y_8)$.



(a) $N = 1000$ and vary m

- 

(b) sensitivity test on $\alpha = 0.3$

Experiments for better inputs to NTKs

- $\mathrm{ReLU}(\langle w, x\rangle) = 1_{\langle w,x\rangle \geq 0}\langle w, x\rangle$.

Experiments for better inputs to NTKs

- $\mathrm{ReLU}(\langle w, x\rangle) = 1_{\langle w,x\rangle \geq 0}\langle w, x\rangle$.
- $1_{\langle w,x\rangle \geq 0}x$: feature mapping, w: weights.

Experiments for better inputs to NTKs

- $\mathrm{ReLU}(\langle w, x \rangle) = 1_{\langle w,x \rangle \geq 0} \langle w, x \rangle$.
- $1_{\langle w,x \rangle \geq 0} x$: feature mapping, w: weights.
- We change every ReLU to $\tilde{1}_{\langle w_1,x \rangle \geq 0} \langle w_2, x \rangle$ ($\tilde{1}$ is a smooth approximation of indicator).

Experiments for better inputs to NTKs

- $\mathrm{ReLU}(\langle w, x\rangle) = 1_{\langle w,x\rangle \geq 0}\langle w, x\rangle$.
- $1_{\langle w,x\rangle \geq 0} x$: feature mapping, w: weights.
- We change every ReLU to $\tilde{1}_{\langle w_1,x\rangle \geq 0}\langle w_2, x\rangle$ ($\tilde{1}$ is a smooth approximation of indicator).
- During the training, w_1 stays at random initialization, only w_2 is trained: Every layer is a NTK.

Experiments for better inputs to NTKs

- $\mathrm{ReLU}(\langle w, x\rangle) = 1_{\langle w,x\rangle\geq 0}\langle w, x\rangle$.
- $1_{\langle w,x\rangle\geq 0}x$: feature mapping, w: weights.
- We change every ReLU to $\tilde{1}_{\langle w_1,x\rangle\geq 0}\langle w_2, x\rangle$ ($\tilde{1}$ is a smooth approximation of indicator).
- During the training, w_1 stays at random initialization, only w_2 is trained: Every layer is a NTK.
- It does not learn a better feature mapping, rather just feeding better inputs to the feature mapping.

Experiments for better inputs to NTKs

- $\text{ReLU}(\langle w, x\rangle) = 1_{\langle w,x\rangle \ge 0}\langle w, x\rangle$.
- $1_{\langle w,x\rangle \ge 0}x$: feature mapping, w: weights.
- We change every ReLU to $\tilde{1}_{\langle w_1,x\rangle \ge 0}\langle w_2, x\rangle$ ($\tilde{1}$ is a smooth approximation of indicator).
- During the training, w_1 stays at random initialization, only w_2 is trained: Every layer is a NTK.
- It does not learn a better feature mapping, rather just feeding better inputs to the feature mapping.
- ResNet-32 8x wide: CIFAR-10: 94.55% (original) v.s. 93.85%, CIFAR-100: 77.04% (original) v.s. 75.27%.

Summary

- This talk we show:

Summary

- This talk we show:
- Neural network is NTK under the following Oops! conditions:

Summary

- This talk we show:
- Neural network is NTK under the following Oops! conditions:
  - Over-parameterization.

Summary

- This talk we show:
- Neural network is NTK under the following Oops! conditions:
  - Over-parameterization.
  - Proper initialization.

Summary

- This talk we show:
- Neural network is NTK under the following Oops! conditions:
  - Over-parameterization.
  - Proper initialization.
  - Small learning rate.

Summary

- This talk we show:
- Neural network is NTK under the following Oops! conditions:
    - Over-parameterization.
    - Proper initialization.
    - Small learning rate.
- Neural network goes beyond NTK when some conditions are broke.

Summary

- This talk we show:
- Neural network is NTK under the following Oops! conditions:
  - Over-parameterization.
  - Proper initialization.
  - Small learning rate.
- Neural network goes beyond NTK when some conditions are broke.
- More work: Small learning rate $\rightarrow$ Scheduled learning rate (Large learning rate + small learning rate): [LMW' 19]

Summary

- This talk we show:
- Neural network is NTK under the following Oops! conditions:
    - Over-parameterization.
    - Proper initialization.
    - Small learning rate.
- Neural network goes beyond NTK when some conditions are broke.
- More work: Small learning rate → Scheduled learning rate (Large learning rate + small learning rate): [LMW' 19]
- Towards explaining the regularization effect of initial large learning rate in training neural networks, arXiv 1907.04595

Summary

- This talk we show:
- Neural network is NTK under the following Oops! conditions:
  - Over-parameterization.
  - Proper initialization.
  - Small learning rate.
- Neural network goes beyond NTK when some conditions are broke.
- More work: Small learning rate → Scheduled learning rate (Large learning rate + small learning rate): [LMW' 19]
- Towards explaining the regularization effect of initial large learning rate in training neural networks, arXiv 1907.04595
- Changing the initialization: Learning ReLU network with over-parameterized ReLU network [LMZ' 19] (to appear).

Future direction

- This talk we show:

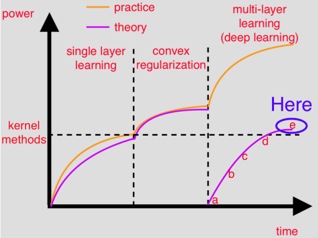
Future direction

- This talk we show:
- Neural network is provably better than any kernel methods or linear regression over feature mappings in some learning tasks.

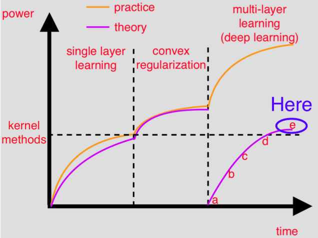
Future direction

- This talk we show:
- Neural network is provably better than any kernel methods or linear regression over feature mappings in some learning tasks.
    - Efficient, (the upper bound) distribution free regime.

Future direction

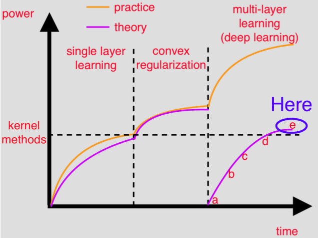
- This talk we show:
- Neural network is provably better than any kernel methods or linear regression over feature mappings in some learning tasks.
  - Efficient, (the upper bound) distribution free regime.
  - 

Future direction

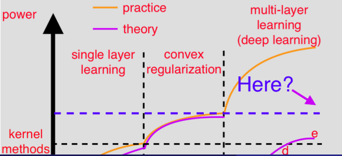
- This talk we show:
- Neural network is provably better than any kernel methods or linear regression over feature mappings in some learning tasks.
  - Efficient, (the upper bound) distribution free regime.
  - 

- Is neural network better than any convex learning algorithm? (even under some distributions?): powerful algorithms such as SOS etc.

Future direction

- This talk we show:
- Neural network is provably better than any kernel methods or linear regression over feature mappings in some learning tasks.
  - Efficient, (the upper bound) distribution free regime.
  - 

- Is neural network better than any convex learning algorithm? (even under some distributions?): powerful algorithms such as SOS etc.



The fundamental question

Question

- Is non-convex optimization (training neural networks) provably better than convex optimization in efficiently computable regime? Or the reason that any non-convex optimization algorithm being efficiently computable is because there is an underlying convex optimization task associated to it?

The fundamental question

Question

- Is non-convex optimization (training neural networks) provably better than convex optimization in efficiently computable regime? Or the reason that any non-convex optimization algorithm being efficiently computable is because there is an underlying convex optimization task associated to it?
- Complexity: P/poly v.s. NP.

The fundamental question

Question

- Is non-convex optimization (training neural networks) provably better than convex optimization in efficiently computable regime? Or the reason that any non-convex optimization algorithm being efficiently computable is because there is an underlying convex optimization task associated to it?
- Complexity: P/poly v.s. NP.
- Theoretical machine learning: Convex-P/poly v.s. Non-convex-P/poly.

Convex circuit v.s. non-convex circuit

Convex circuit v.s. non-convex circuit

Definition (Convex circuit)

Convex circuit v.s. non-convex circuit

Definition (Convex circuit)

- A (k-output) circuit C with two inputs $W \in \mathbb{R}^m, x \in \mathbb{R}^d$ is a convex circuit over loss $L : \mathbb{R}^{2k} \to \mathbb{R}$ and a set $\mathcal{S} \in \mathbb{R}^{d+k}$ if given any $(x, y) \in \mathcal{S}$:

Convex circuit v.s. non-convex circuit

Definition (Convex circuit)

- A (k-output) circuit C with two inputs $W \in \mathbb{R}^m, x \in \mathbb{R}^d$ is a convex circuit over loss $L : \mathbb{R}^{2k} \to \mathbb{R}$ and a set $\mathcal{S} \in \mathbb{R}^{d+k}$ if given any $(x, y) \in \mathcal{S}$:
- $L(C(W, x), y)$ is (strictly) convex/(one point) convex in W.

Convex circuit v.s. non-convex circuit

Definition (Convex circuit)

- A (k-output) circuit C with two inputs $W \in \mathbb{R}^m, x \in \mathbb{R}^d$ is a convex circuit over loss $L : \mathbb{R}^{2k} \to \mathbb{R}$ and a set $\mathcal{S} \in \mathbb{R}^{d+k}$ if given any $(x, y) \in \mathcal{S}$:
- $L(C(W, x), y)$ is (strictly) convex/(one point) convex in W.
- Learning: $W^* = \operatorname{argmin} \sum_i L(C(W, x_i), y_i)$ over samples $\{x_i, y_i\}_{i \in [N]}$.

Convex circuit v.s. non-convex circuit

Definition (Convex circuit)

- A (k-output) circuit C with two inputs $W \in \mathbb{R}^m, x \in \mathbb{R}^d$ is a convex circuit over loss $L : \mathbb{R}^{2k} \to \mathbb{R}$ and a set $\mathcal{S} \in \mathbb{R}^{d+k}$ if given any $(x, y) \in \mathcal{S}$:
  - $L(C(W, x), y)$ is (strictly) convex/(one point) convex in W.
  - Learning: $W^* = \text{argmin} \sum_i L(C(W, x_i), y_i)$ over samples $\{x_i, y_i\}_{i \in [N]}$.
  - Given x, making prediction using $C(W^*, x)$.

Convex circuit v.s. non-convex circuit

Definition (Convex circuit)

- A (k-output) circuit C with two inputs $W \in \mathbb{R}^m, x \in \mathbb{R}^d$ is a convex circuit over loss $L : \mathbb{R}^{2k} \to \mathbb{R}$ and a set $\mathcal{S} \in \mathbb{R}^{d+k}$ if given any $(x, y) \in \mathcal{S}$:
  - $L(C(W, x), y)$ is (strictly) convex/(one point) convex in W.
  - Learning: $W^* = \operatorname{argmin} \sum_i L(C(W, x_i), y_i)$ over samples $\{x_i, y_i\}_{i \in [N]}$.
  - Given x, making prediction using $C(W^*, x)$.

Convex circuit v.s. non-convex circuit

Definition (Convex circuit)

- A (k-output) circuit C with two inputs $W \in \mathbb{R}^m, x \in \mathbb{R}^d$ is a convex circuit over loss $L : \mathbb{R}^{2k} \to \mathbb{R}$ and a set $\mathcal{S} \in \mathbb{R}^{d+k}$ if given any $(x, y) \in \mathcal{S}$:
- $L(C(W, x), y)$ is (strictly) convex/(one point) convex in W.
- Learning: $W^* = \operatorname{argmin} \sum_i L(C(W, x_i), y_i)$ over samples $\{x_i, y_i\}_{i \in [N]}$.
- Given x, making prediction using $C(W^*, x)$.

Definition (Poly size non-convex circuit)

Convex circuit v.s. non-convex circuit

Definition (Convex circuit)

- A (k-output) circuit C with two inputs $W \in \mathbb{R}^m, x \in \mathbb{R}^d$ is a convex circuit over loss $L : \mathbb{R}^{2k} \to \mathbb{R}$ and a set $\mathcal{S} \in \mathbb{R}^{d+k}$ if given any $(x, y) \in \mathcal{S}$:
- $L(C(W, x), y)$ is (strictly) convex/(one point) convex in W.
- Learning: $W^* = \text{argmin} \sum_i L(C(W, x_i), y_i)$ over samples $\{x_i, y_i\}_{i \in [N]}$.
- Given x, making prediction using $C(W^*, x)$.

Definition (Poly size non-convex circuit)

- C is a poly-size (arithmetic) circuit.

Convex circuit v.s. non-convex circuit

Definition (Convex circuit)

- A (k-output) circuit C with two inputs $W \in \mathbb{R}^m, x \in \mathbb{R}^d$ is a convex circuit over loss $L : \mathbb{R}^{2k} \to \mathbb{R}$ and a set $\mathcal{S} \in \mathbb{R}^{d+k}$ if given any $(x, y) \in \mathcal{S}$:
- $L(C(W, x), y)$ is (strictly) convex/(one point) convex in W.
- Learning: $W^* = \operatorname{argmin} \sum_i L(C(W, x_i), y_i)$ over samples $\{x_i, y_i\}_{i \in [N]}$.
- Given x, making prediction using $C(W^*, x)$.

Definition (Poly size non-convex circuit)

- C is a poly-size (arithmetic) circuit.
- $L(C(W, x), y)$ doesn't have to be convex.

Convex circuit v.s. non-convex circuit

Definition (Convex circuit)

- A (k-output) circuit C with two inputs $W \in \mathbb{R}^m, x \in \mathbb{R}^d$ is a convex circuit over loss $L : \mathbb{R}^{2k} \to \mathbb{R}$ and a set $\mathcal{S} \in \mathbb{R}^{d+k}$ if given any $(x, y) \in \mathcal{S}$:
- $L(C(W, x), y)$ is (strictly) convex/(one point) convex in W.
- Learning: $W^* = \operatorname{argmin} \sum_i L(C(W, x_i), y_i)$ over samples $\{x_i, y_i\}_{i \in [N]}$.
- Given x, making prediction using $C(W^*, x)$.

Definition (Poly size non-convex circuit)

- C is a poly-size (arithmetic) circuit.
- $L(C(W, x), y)$ doesn't have to be convex.
- Learning: $W = W - \eta \sum_i \nabla_W L(C(W, x_i), y_i)$, in polynomially many iterations.

Convex circuit v.s. non-convex circuit

Definition (Convex circuit)

- A (k-output) circuit C with two inputs $W \in \mathbb{R}^m, x \in \mathbb{R}^d$ is a convex circuit over loss $L : \mathbb{R}^{2k} \to \mathbb{R}$ and a set $\mathcal{S} \in \mathbb{R}^{d+k}$ if given any $(x, y) \in \mathcal{S}$:
  - $L(C(W, x), y)$ is (strictly) convex/(one point) convex in W.
  - Learning: $W^* = \operatorname{argmin} \sum_i L(C(W, x_i), y_i)$ over samples $\{x_i, y_i\}_{i \in [N]}$.
  - Given x, making prediction using $C(W^*, x)$.

Definition (Poly size non-convex circuit)

-
  - C is a poly-size (arithmetic) circuit.
  - $L(C(W, x), y)$ doesn't have to be convex.
  - Learning: $W = W - \eta \sum_i \nabla_W L(C(W, x_i), y_i)$, in polynomially many iterations.

- Question: is non-convex-P/poly better than convex-P/poly?

The fundamental question in theoretical machine learning

- When L is ℓ_2 and the support of y is the full space: easy to prove that $C(W,x)$ must be linear in W if $L(C(W,x),y)$ is convex.

The fundamental question in theoretical machine learning

- When L is ℓ_2 and the support of y is the full space: easy to prove that $C(W, x)$ must be linear in W if $L(C(W, x), y)$ is convex.
- Linear regression over feature mappings $\implies$ we get a provable separation in this work.

The fundamental question in theoretical machine learning

- When L is ℓ_2 and the support of y is the full space: easy to prove that $C(W, x)$ must be linear in W if $L(C(W, x), y)$ is convex.
- Linear regression over feature mappings $\implies$ we get a provable separation in this work.
- On the other hand, we also see a lot of cases where the opposite is true, especially for one-point convexity:

The fundamental question in theoretical machine learning

- When L is ℓ_2 and the support of y is the full space: easy to prove that $C(W, x)$ must be linear in W if $L(C(W, x), y)$ is convex.
- Linear regression over feature mappings $\implies$ we get a provable separation in this work.
- On the other hand, we also see a lot of cases where the opposite is true, especially for one-point convexity:
- Matrix sensing/Matrix completion, can be solved by $UV^\top$ (non-convex) or nuclear norm regularization on M (convex). Actually $\|U\|_F^2 + \|V\|_F^2$ is very similar to $\|UV^\top\|_*$ as well.

The fundamental question in theoretical machine learning

- When L is ℓ_2 and the support of y is the full space: easy to prove that $C(W, x)$ must be linear in W if $L(C(W, x), y)$ is convex.
- Linear regression over feature mappings $\implies$ we get a provable separation in this work.
- On the other hand, we also see a lot of cases where the opposite is true, especially for one-point convexity:
- Matrix sensing/Matrix completion, can be solved by $UV^\top$ (non-convex) or nuclear norm regularization on M (convex). Actually $\|U\|_F^2 + \|V\|_F^2$ is very similar to $\|UV^\top\|_*$ as well.
- One point convexity (convex w.r.t. W^*) for many non-convex learning algorithms: Learning one hidden layer network, Dictionary learning, Quadratic programming, Blind deconvolution, this work.

The fundamental question in theoretical machine learning

- When L is ℓ_2 and the support of y is the full space: easy to prove that $C(W, x)$ must be linear in W if $L(C(W, x), y)$ is convex.
- Linear regression over feature mappings $\Longrightarrow$ we get a provable separation in this work.
- On the other hand, we also see a lot of cases where the opposite is true, especially for one-point convexity:
- Matrix sensing/Matrix completion, can be solved by $UV^\top$ (non-convex) or nuclear norm regularization on M (convex). Actually $\|U\|_F^2 + \|V\|_F^2$ is very similar to $\|UV^\top\|_*$ as well.
- One point convexity (convex w.r.t. W^*) for many non-convex learning algorithms: Learning one hidden layer network, Dictionary learning, Quadratic programming, Blind deconvolution, this work.

The fundamental question in theoretical machine learning

- When L is ℓ_2 and the support of y is the full space: easy to prove that $C(W, x)$ must be linear in W if $L(C(W, x), y)$ is convex.
- Linear regression over feature mappings $\Longrightarrow$ we get a provable separation in this work.
- On the other hand, we also see a lot of cases where the opposite is true, especially for one-point convexity:
- Matrix sensing/Matrix completion, can be solved by $UV^\top$ (non-convex) or nuclear norm regularization on M (convex). Actually $\|U\|_F^2 + \|V\|_F^2$ is very similar to $\|UV^\top\|_*$ as well.
- One point convexity (convex w.r.t. W^*) for many non-convex learning algorithms: Learning one hidden layer network, Dictionary learning, Quadratic programming, Blind deconvolution, this work.

Question (Fundamental question)

Is non-convex optimization (training neural networks) provably better than convex/one point convex optimization in efficiently computable regime?