International Conference on Machine Learning (ICML) 2018

On the Optimization of Deep Networks: Implicit Acceleration by Overparameterization

Sanjeev Arora[*†] **Nadav Cohen**[†] Elad Hazan[*‡]

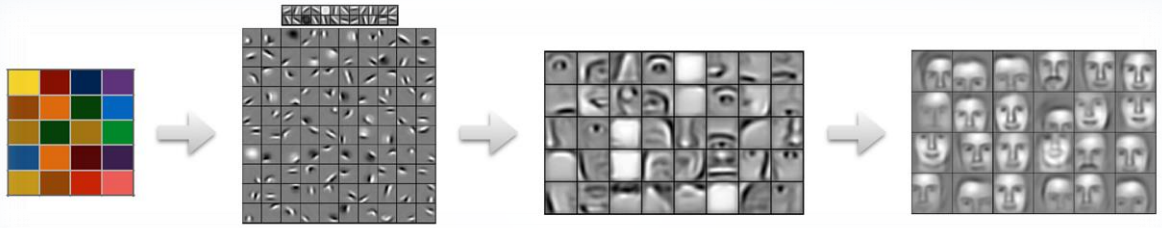
[*] *Princeton University* [†] *Institute for Advanced Study* [‡] *Google Brain*

How Does Depth Help?

How Does Depth Help?

Conventional wisdom:

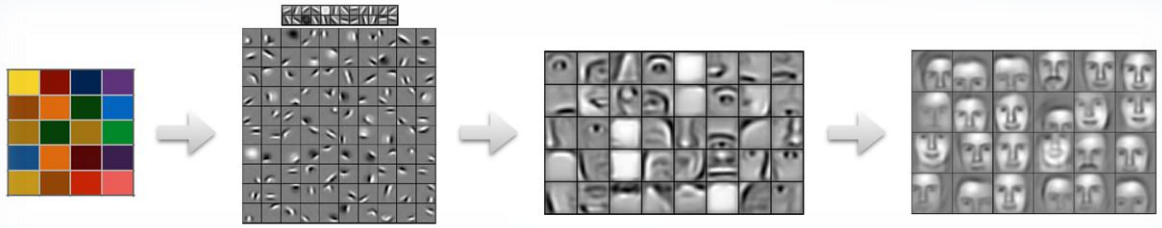
- Depth boosts expressive power



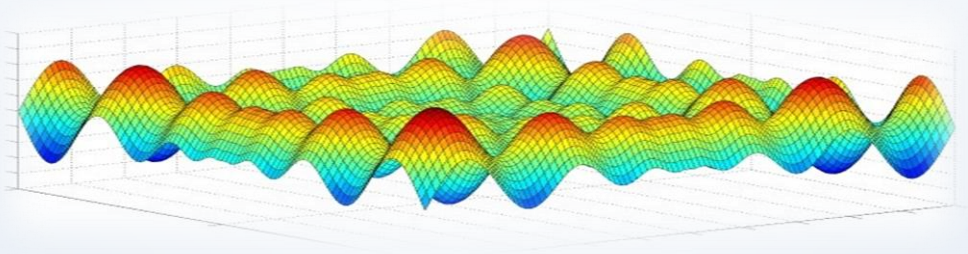
How Does Depth Help?

Conventional wisdom:

- Depth boosts expressive power



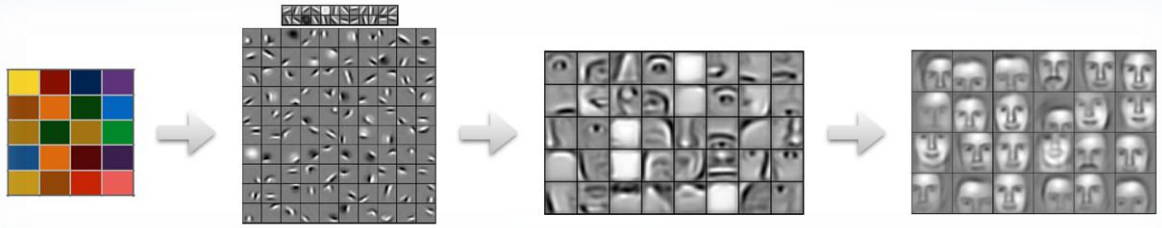
- But complicates optimization



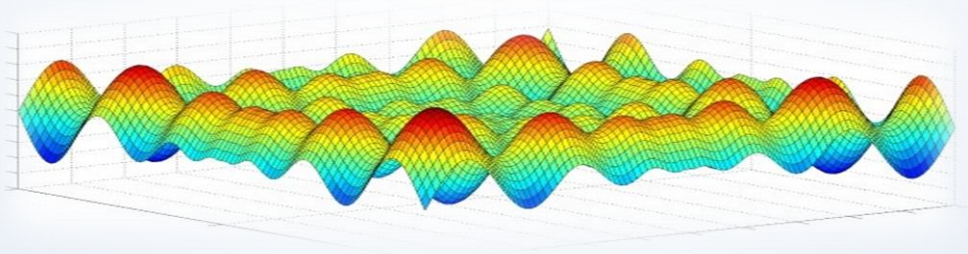
How Does Depth Help?

Conventional wisdom:

- Depth boosts expressive power



- But complicates optimization



This work:

Depth can accelerate optimization!

Decoupling Optimization from Expressiveness

Problem:

Expressiveness can interfere with our study – deeper nets may seem to optimize faster per being able to reach lower training err

Decoupling Optimization from Expressiveness

Problem:

Expressiveness can interfere with our study – deeper nets may seem to optimize faster per being able to reach lower training err

Resolution:

- We focus on models whose expressiveness is oblivious to depth – **linear neural networks**

Decoupling Optimization from Expressiveness

Problem:

Expressiveness can interfere with our study – deeper nets may seem to optimize faster per being able to reach lower training err

Resolution:

- We focus on models whose expressiveness is oblivious to depth – **linear neural networks**
- Adding layers amounts to replacing matrix param by product of matrices – **overparameterization**

Decoupling Optimization from Expressiveness

Problem:

Expressiveness can interfere with our study – deeper nets may seem to optimize faster per being able to reach lower training err

Resolution:

- We focus on models whose expressiveness is oblivious to depth – **linear neural networks**
- Adding layers amounts to replacing matrix param by product of matrices – **overparameterization**

Linear neural networks were studied extensively, cf. [Saxe et al. 2013; Kawaguchi 2016; Hardt & Ma 2016]

Warm-Up

Training objective for linear regression with ℓ_p loss:

$$L(\boldsymbol{w}) = \sum_{(\boldsymbol{x},y)\in S} \frac{1}{p}\left(\boldsymbol{x}^T\boldsymbol{w} - y\right)^{\boldsymbol{p}}$$

Warm-Up

Training objective for linear regression with ℓ_p loss:

$$L(\boldsymbol{w}) = \sum_{(\boldsymbol{x},y)\in S} \frac{1}{p}\left(\boldsymbol{x}^T\boldsymbol{w} - y\right)^{\boldsymbol{p}}$$

convex

Warm-Up

Training objective for linear regression with ℓ_p loss:

$$L(\boldsymbol{w}) = \sum_{(\boldsymbol{x},y)\in S} \frac{1}{p}\left(\boldsymbol{x}^T\boldsymbol{w} - y\right)^{\boldsymbol{p}}$$

convex

Overparameterize – replace vector $\boldsymbol{w}$ by vector $\boldsymbol{w_1}$ times scalar w_2:

$$\tilde{L}(\boldsymbol{w_1}, w_2) = \sum_{(\boldsymbol{x},y)\in S} \frac{1}{p}\left(\boldsymbol{x}^T\boldsymbol{w_1}w_2 - y\right)^{\boldsymbol{p}}$$

Warm-Up

Training objective for linear regression with ℓ_p loss:

$$L(\boldsymbol{w}) = \sum_{(\boldsymbol{x},y)\in S} \frac{1}{p}\left(\boldsymbol{x}^T\boldsymbol{w} - y\right)^{\boldsymbol{p}}$$

convex

Overparameterize – replace vector $\boldsymbol{w}$ by vector $\boldsymbol{w_1}$ times scalar w_2:

$$\tilde{L}(\boldsymbol{w_1}, w_2) = \sum_{(\boldsymbol{x},y)\in S} \frac{1}{p}\left(\boldsymbol{x}^T\boldsymbol{w_1}w_2 - y\right)^{\boldsymbol{p}}$$

non-convex

Warm-Up

Training objective for linear regression with ℓ_p loss:

$$L(\boldsymbol{w}) = \sum_{(x,y)\in S} \frac{1}{p}\left(\boldsymbol{x}^T\boldsymbol{w} - y\right)^{\boldsymbol{p}}$$

convex

Overparameterize – replace vector $\boldsymbol{w}$ by vector $\boldsymbol{w_1}$ times scalar w_2:

$$\tilde{L}(\boldsymbol{w_1}, w_2) = \sum_{(x,y)\in S} \frac{1}{p}\left(\boldsymbol{x}^T\boldsymbol{w_1}w_2 - y\right)^{\boldsymbol{p}}$$

non-convex

Claim:

- Gradient descent (GD) over $\tilde{L}(\boldsymbol{w_1}, w_2)$ induces on $\boldsymbol{w} = \boldsymbol{w_1}w_2$:

$$\boldsymbol{w}^{(t+1)} \hookleftarrow \boldsymbol{w}^{(t)} - \rho^{(t)} \cdot \nabla L\left(\boldsymbol{w}^{(t)}\right) - \sum_{\tau=1}^{t-1} \mu^{(t,\tau)} \cdot \nabla L\left(\boldsymbol{w}^{(\tau)}\right)$$

Warm-Up

Training objective for linear regression with ℓ_p loss:

$$L(\boldsymbol{w}) = \sum_{(x,y)\in S} \frac{1}{p}\left(\boldsymbol{x}^T\boldsymbol{w} - y\right)^{\boldsymbol{p}}$$

convex

Overparameterize – replace vector $\boldsymbol{w}$ by vector $\boldsymbol{w_1}$ times scalar w_2:

$$\tilde{L}(\boldsymbol{w_1}, w_2) = \sum_{(x,y)\in S} \frac{1}{p}\left(\boldsymbol{x}^T\boldsymbol{w_1}w_2 - y\right)^{\boldsymbol{p}}$$

non-convex

Claim:

- Gradient descent (GD) over $\tilde{L}(\boldsymbol{w_1}, w_2)$ induces on $\boldsymbol{w} = \boldsymbol{w_1}w_2$:

$$\boldsymbol{w}^{(t+1)} \leftarrow\!\shortmid\ \boldsymbol{w}^{(t)} - \rho^{(t)} \cdot \nabla L\left(\boldsymbol{w}^{(t)}\right) - \sum_{\tau=1}^{t-1} \mu^{(t,\tau)} \cdot \nabla L\left(\boldsymbol{w}^{(\tau)}\right)$$

adaptive learning rate

Warm-Up

Training objective for linear regression with ℓ_p loss:

$$L(\boldsymbol{w}) = \sum_{(x,y)\in S} \frac{1}{p}\left(\boldsymbol{x}^T\boldsymbol{w} - y\right)^{\boldsymbol{p}}$$

convex

Overparameterize – replace vector $\boldsymbol{w}$ by vector $\boldsymbol{w_1}$ times scalar w_2:

$$\tilde{L}(\boldsymbol{w_1}, w_2) = \sum_{(x,y)\in S} \frac{1}{p}\left(\boldsymbol{x}^T\boldsymbol{w_1}w_2 - y\right)^{\boldsymbol{p}}$$

non-convex

Claim:

- Gradient descent (GD) over $\tilde{L}(\boldsymbol{w_1}, w_2)$ induces on $\boldsymbol{w} = \boldsymbol{w_1}w_2$:

$$\boldsymbol{w}^{(t+1)} \leftarrow\!\shortmid\ \boldsymbol{w}^{(t)} - \rho^{(t)} \cdot \nabla L\left(\boldsymbol{w}^{(t)}\right) - \sum_{\tau=1}^{t-1} \mu^{(t,\tau)} \cdot \nabla L\left(\boldsymbol{w}^{(\tau)}\right)$$

adaptive learning rate

"momentum"

Warm-Up

Training objective for linear regression with ℓ_p loss:

$$L(\boldsymbol{w}) = \sum_{(x,y)\in S} \frac{1}{p}\left(\boldsymbol{x}^T\boldsymbol{w} - y\right)^{\boldsymbol{p}}$$

convex

Overparameterize – replace vector $\boldsymbol{w}$ by vector $\boldsymbol{w_1}$ times scalar w_2:

$$\tilde{L}(\boldsymbol{w_1}, w_2) = \sum_{(x,y)\in S} \frac{1}{p}\left(\boldsymbol{x}^T\boldsymbol{w_1}w_2 - y\right)^{\boldsymbol{p}}$$

non-convex

Claim:

- Gradient descent (GD) over $\tilde{L}(\boldsymbol{w_1}, w_2)$ induces on $\boldsymbol{w} = \boldsymbol{w_1}w_2$:

$$\boldsymbol{w}^{(t+1)} \leftarrow\!\shortmid\ \boldsymbol{w}^{(t)} - \rho^{(t)} \cdot \nabla L\left(\boldsymbol{w}^{(t)}\right) - \sum_{\tau=1}^{t-1} \mu^{(t,\tau)} \cdot \nabla L\left(\boldsymbol{w}^{(\tau)}\right)$$

adaptive learning rate

"momentum"

- For $p > 2$, this can speed up optimization

Warm-Up

Training objective for linear regression with ℓ_p loss:

$$L(\boldsymbol{w}) = \sum_{(x,y)\in S} \frac{1}{p}\left(\boldsymbol{x}^T\boldsymbol{w} - y\right)^{\boldsymbol{p}}$$

convex

Overparameterize – replace vector $\boldsymbol{w}$ by vector $\boldsymbol{w_1}$ times scalar w_2:

$$\tilde{L}(\boldsymbol{w_1}, w_2) = \sum_{(x,y)\in S} \frac{1}{p}\left(\boldsymbol{x}^T\boldsymbol{w_1}w_2 - y\right)^{\boldsymbol{p}}$$

non-convex

Claim:

- Gradient descent (GD) over $\tilde{L}(\boldsymbol{w_1}, w_2)$ induces on $\boldsymbol{w} = \boldsymbol{w_1}w_2$:

$$\boldsymbol{w}^{(t+1)} \leftarrow\!\shortmid\ \boldsymbol{w}^{(t)} - \rho^{(t)} \cdot \nabla L\left(\boldsymbol{w}^{(t)}\right) - \sum_{\tau=1}^{t-1} \mu^{(t,\tau)} \cdot \nabla L\left(\boldsymbol{w}^{(\tau)}\right)$$

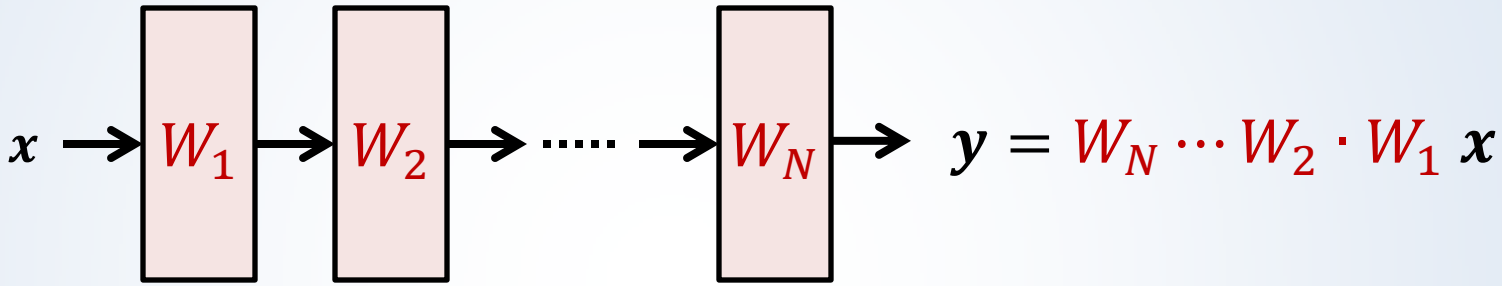
adaptive learning rate

"momentum"

- For $p > 2$, this can speed up optimization (cf. [Saxe et al. 2013])

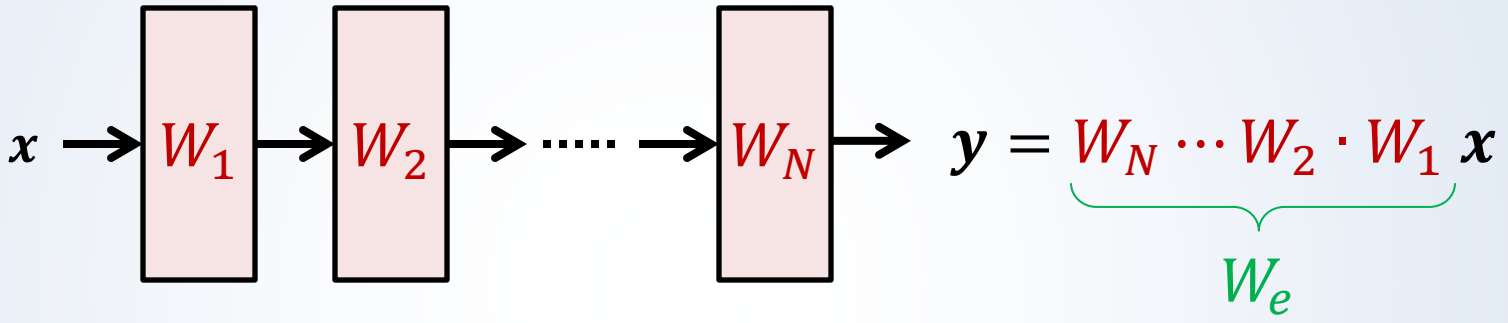
Formal Setup

Linear neural network:



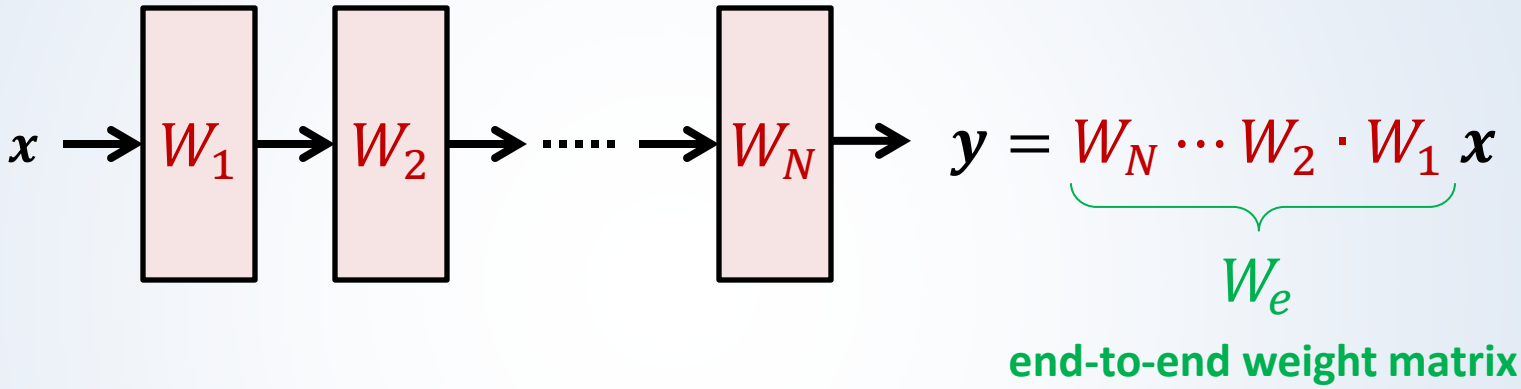
Formal Setup

Linear neural network:



Formal Setup

Linear neural network:

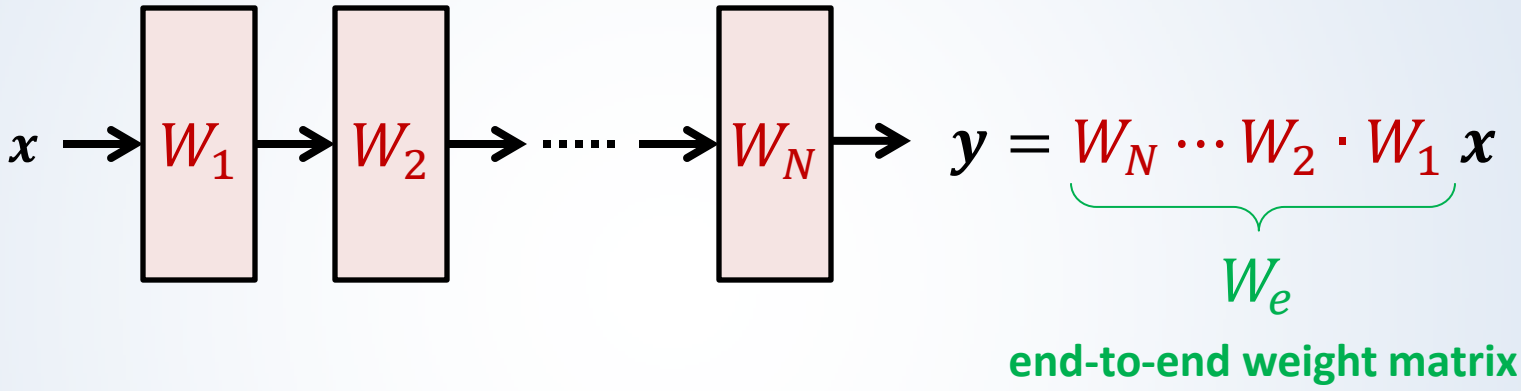


Given loss L over linear model, we have the **overparameterized loss**:

$$\tilde{L}(W_1, \cdots, W_N) = L(W_e)$$

Formal Setup

Linear neural network:



Given loss L over linear model, we have the **overparameterized loss**:

$$\tilde{L}(W_1, \cdots, W_N) = L(W_e)$$

Question:

How does W_e behave during GD over $W_1, \cdots, W_N$?

Implicit Dynamics of Gradient Descent

Theorem:

GD over $W_1, \cdots, W_N$ with small learning rate and near-zero init, induces on W_e the **end-to-end update rule**:

$$W_e^{(t+1)} \longleftarrow W_e^{(t)} - \eta \sum_{j=1}^{N} \left[W_e^{(t)} \left(W_e^{(t)} \right)^T \right]^{\frac{j-1}{N}} \nabla L\left(W_e^{(t)} \right) \left[\left(W_e^{(t)} \right)^T W_e^{(t)} \right]^{\frac{N-j}{N}}$$

Implicit Dynamics of Gradient Descent

Theorem:

GD over $W_1, \cdots, W_N$ with small learning rate and near-zero init, induces on W_e the **end-to-end update rule**:

$$W_e^{(t+1)} \leftarrow\!\shortmid\ W_e^{(t)} - \eta \sum_{j=1}^{N} \left[W_e^{(t)} \left(W_e^{(t)} \right)^T \right]^{\frac{j-1}{N}} \nabla L\left(W_e^{(t)} \right) \left[\left(W_e^{(t)} \right)^T W_e^{(t)} \right]^{\frac{N-j}{N}}$$

which is a:

- preconditioner promoting movement in directions already taken

Implicit Dynamics of Gradient Descent

Theorem:

GD over $W_1, \cdots, W_N$ with small learning rate and near-zero init, induces on W_e the **end-to-end update rule**:

$$W_e^{(t+1)} \longleftarrow W_e^{(t)} - \eta \sum_{j=1}^{N} \left[W_e^{(t)} \left(W_e^{(t)} \right)^T \right]^{\frac{j-1}{N}} \nabla L\left(W_e^{(t)} \right) \left[\left(W_e^{(t)} \right)^T W_e^{(t)} \right]^{\frac{N-j}{N}}$$

which is a:

- preconditioner promoting movement in directions already taken
- certain combination of adaptive learning rate and "momentum"

Implicit Dynamics of Gradient Descent

Theorem:

GD over $W_1, \cdots, W_N$ with small learning rate and near-zero init, induces on W_e the **end-to-end update rule**:

$$W_e^{(t+1)} \longleftarrow W_e^{(t)} - \eta \sum_{j=1}^{N} \left[W_e^{(t)} \left(W_e^{(t)} \right)^T \right]^{\frac{j-1}{N}} \nabla L\left(W_e^{(t)} \right) \left[\left(W_e^{(t)} \right)^T W_e^{(t)} \right]^{\frac{N-j}{N}}$$

which is a:

- preconditioner promoting movement in directions already taken
- certain combination of adaptive learning rate and "momentum"

Overparameterization with deep linear net induces on GD a certain acceleration scheme! No dependence on layer widths!

Overparameterization via Objective Modification?

Overparameterization via Objective Modification?

Theorem:

There exists no objective func (of W_e) over which GD gives the end-to-end update rule

Overparameterization via Objective Modification?

Theorem:

There exists no objective func (of W_e) over which GD gives the end-to-end update rule

Proof sketch:

Fundamental theorem for line integrals:

$\oint_\Gamma \nabla g = 0 \quad \forall$ func g , closed curve Γ

Construct curve on which line integral of end-to-end updates $\neq 0$

Overparameterization via Objective Modification?

Theorem:

There exists no objective func (of W_e) over which GD gives the end-to-end update rule

Proof sketch:

Fundamental theorem for line integrals:

$$\oint_\Gamma \nabla g = 0 \quad \forall \text{ func } g \text{ , closed curve } \Gamma$$

Construct curve on which line integral of end-to-end updates $\neq 0$

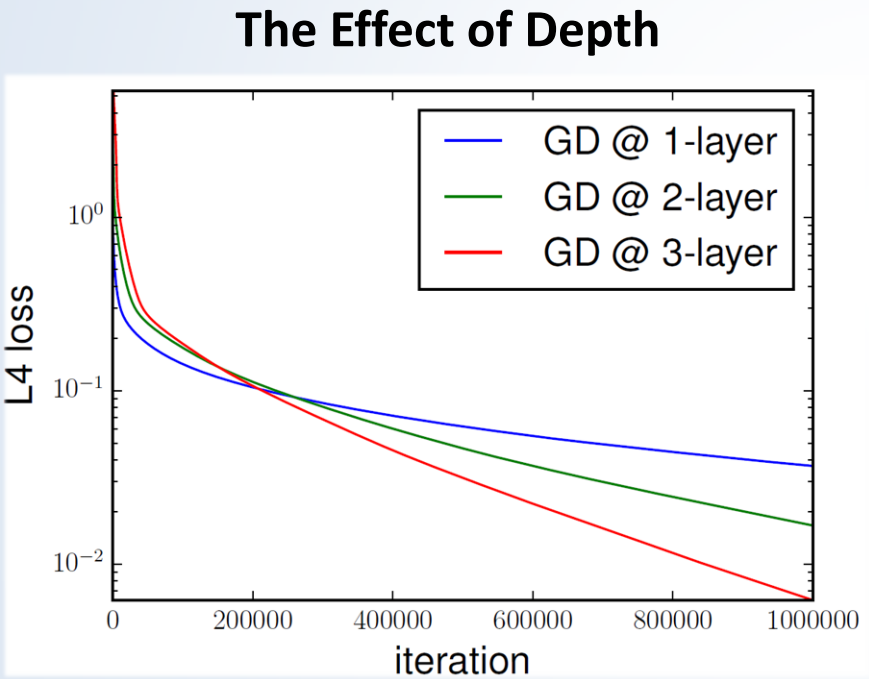
Effect of overparameterization by depth cannot be attained via any modification of the objective!

Experiments – Linear Networks

Regression problem from UCI ML Repository; ℓ_4 loss

Experiments – Linear Networks

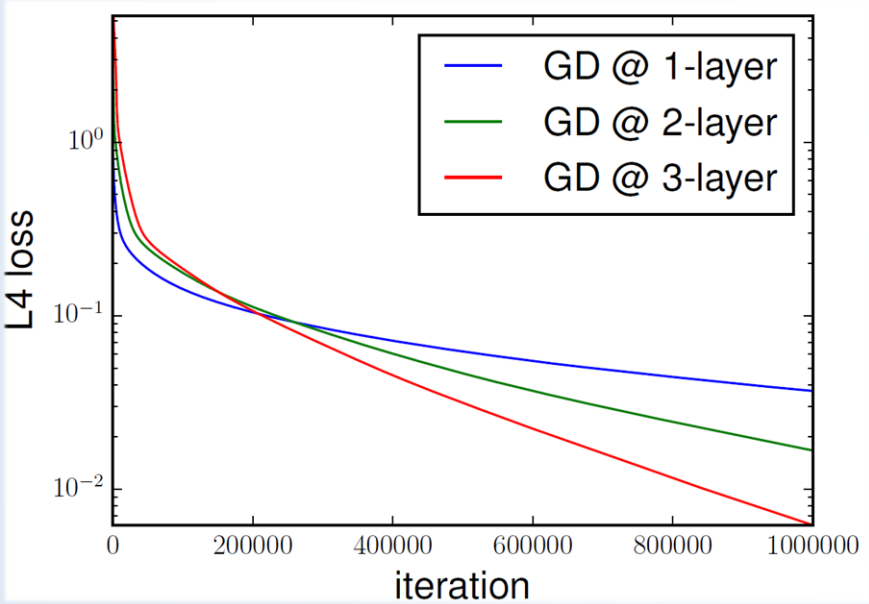
Regression problem from UCI ML Repository; ℓ_4 loss



Experiments – Linear Networks

Regression problem from UCI ML Repository; ℓ_4 loss

The Effect of Depth

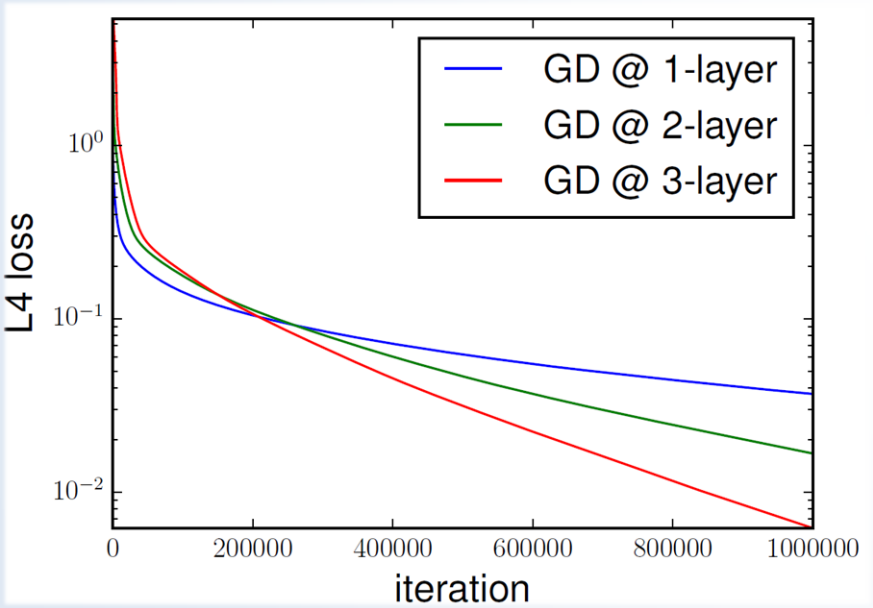


Depth can speed up GD, even w/o any change in expressiveness, and despite introducing non-convexity!

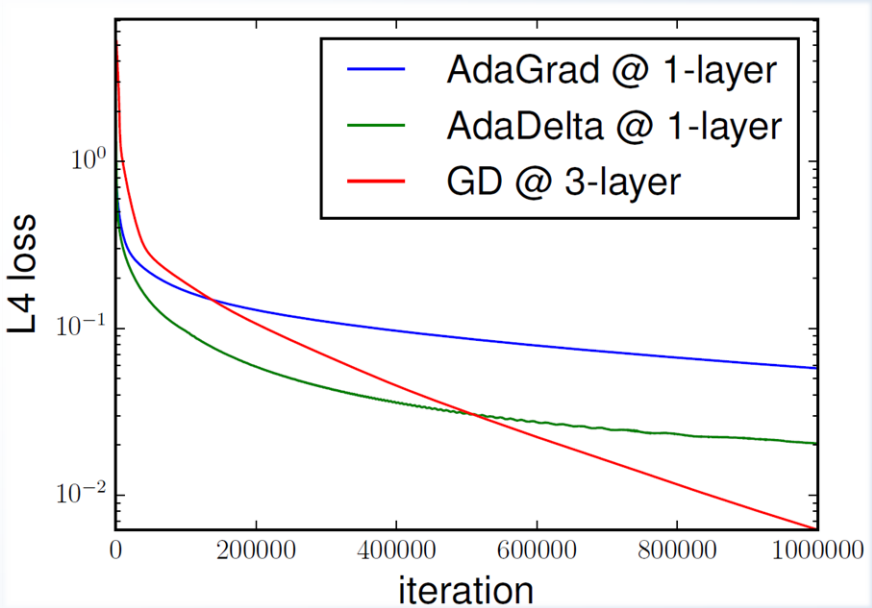
Experiments – Linear Networks

Regression problem from UCI ML Repository; ℓ_4 loss

The Effect of Depth



Depth vs. Explicit Accelerators

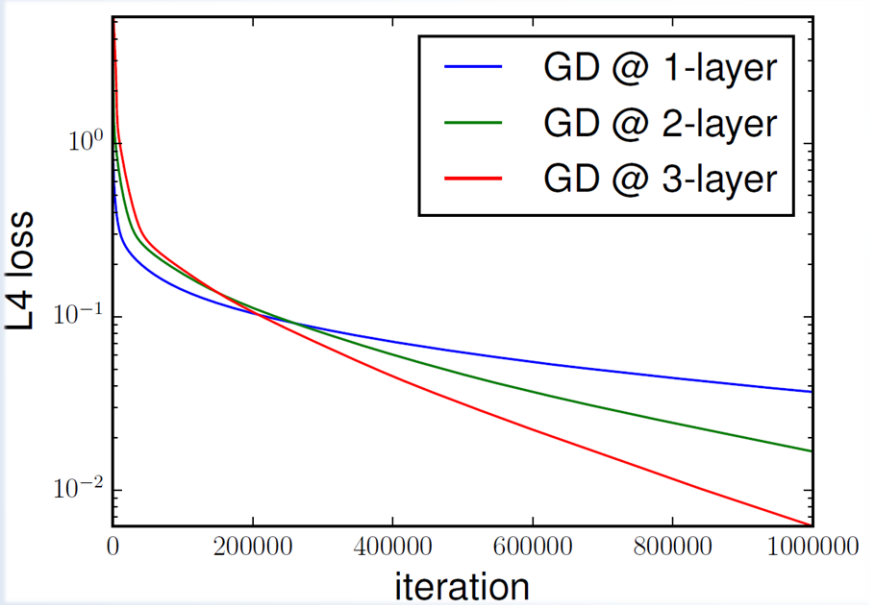


Depth can speed up GD, even w/o any change in expressiveness, and despite introducing non-convexity!

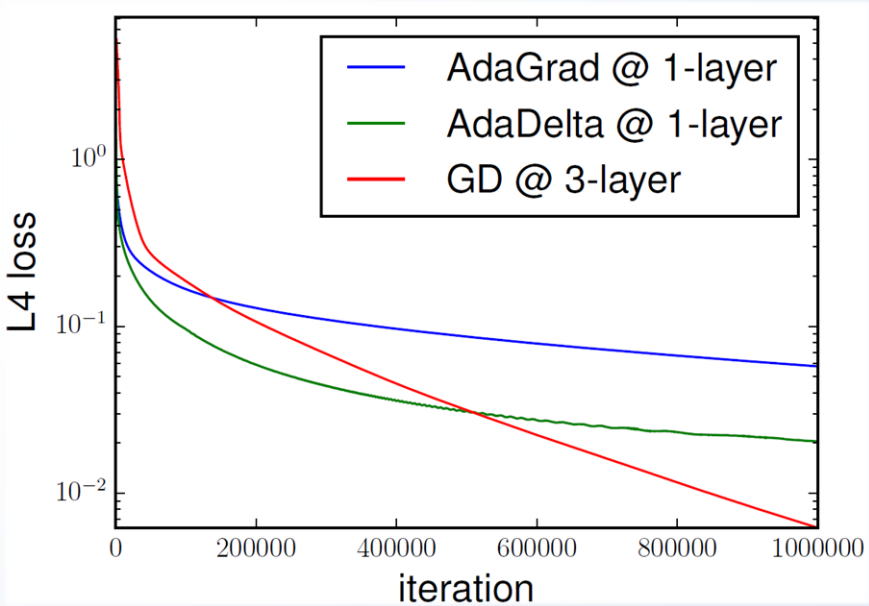
Experiments – Linear Networks

Regression problem from UCI ML Repository; ℓ_4 loss

The Effect of Depth



Depth vs. Explicit Accelerators



Depth can speed up GD, even w/o any change in expressiveness, and despite introducing non-convexity!

This speed up can outperform explicit acceleration methods designed for convex problems!

Experiments – Non-Linear Network

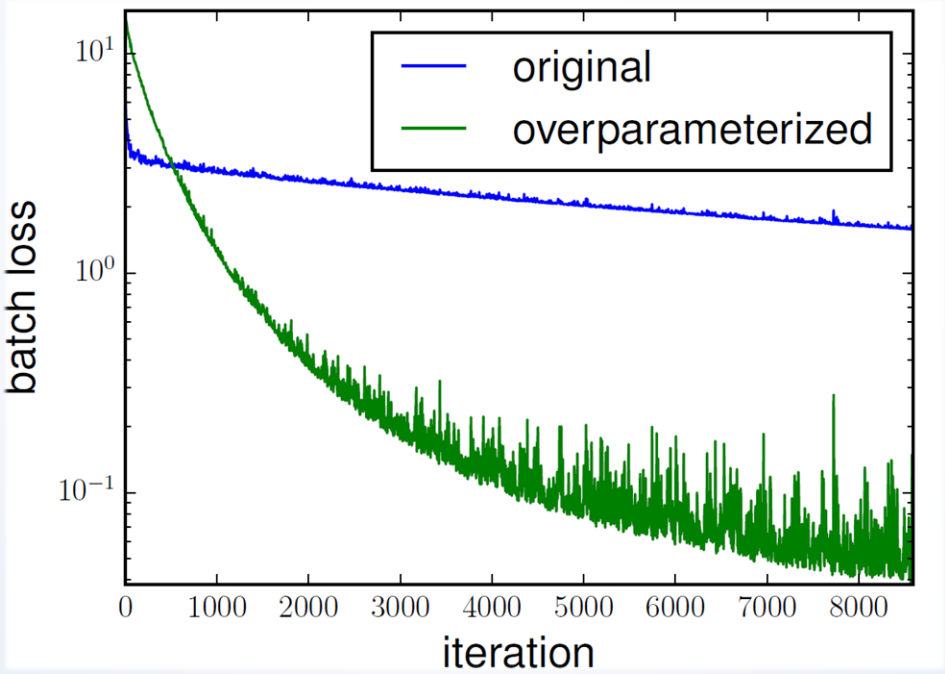
TensorFlow CNN tutorial for MNIST

Overparameterization: fully-connected layers → depth-2 linear nets

Experiments – Non-Linear Network

TensorFlow CNN tutorial for MNIST

Overparameterization: fully-connected layers → depth-2 linear nets



With +15% in params, and no change in expressiveness, overparameterization accelerated by orders of magnitude!

Conclusion

Conclusion

- Depth radically changes obj landscape, turns it highly non-convex

Conclusion

- Depth radically changes obj landscape, turns it highly non-convex
- Conventional wisdom: this complicates optimization

Conclusion

- Depth radically changes obj landscape, turns it highly non-convex
- Conventional wisdom: this complicates optimization
- We show:
  - For linear nets, depth induces on GD a preconditioning scheme

Conclusion

- Depth radically changes obj landscape, turns it highly non-convex
- Conventional wisdom: this complicates optimization
- We show:
  - For linear nets, depth induces on GD a preconditioning scheme
  - Preconditioning combines adaptive learning rate and “momentum”

Conclusion

- Depth radically changes obj landscape, turns it highly non-convex
- Conventional wisdom: this complicates optimization
- We show:
  - For linear nets, depth induces on GD a preconditioning scheme
  - Preconditioning combines adaptive learning rate and “momentum”
  - Effect cannot be attained via any modification of original objective

Conclusion

- Depth radically changes obj landscape, turns it highly non-convex
- Conventional wisdom: this complicates optimization
- We show:
  - For linear nets, depth induces on GD a preconditioning scheme
  - Preconditioning combines adaptive learning rate and "momentum"
  - Effect cannot be attained via any modification of original objective
  - Can lead to significant speed ups, despite no change in expressiveness

Thank You