On the Optimization of Deep Networks: Implicit Acceleration by Overparameterization

Nadav Cohen

Institute for Advanced Study

Symposium on the Mathematical Theory of Deep Neural Networks

Princeton Neuroscience Institute

20 March 2018

Collaborators

Sanjeev Arora [1 2]

Elad Hazan [1 3]

[1] Princeton University [2] Institute for Advanced Study [3] Google Brain

Outline

Deep Learning

EVERY INDUSTRY WANTS DEEP LEARNING

Cloud Service Provider	Medicine	Media & Entertainment	Security & Defense	Autonomous Machines

➢ Image/Video classification	➢ Cancer cell detection	➢ Video captioning	➢ Face recognition	➢ Pedestrian detection
➢ Speech recognition	➢ Diabetic grading	➢ Content based search	➢ Video surveillance	➢ Lane tracking
➢ Natural language processing	➢ Drug discovery	➢ Real time translation	➢ Cyber security	➢ Recognize traffic sign

NVIDIA

Source

NVIDIA (`www.slideshare.net/openomics/the-revolution-of-deep-learning`)

Why Depth?

Why Depth?

Conventional wisdom:

- Depth boosts expressive power

  The Power of Depth for Feedforward Neural Networks (Eldan & Shamir, 2016)

  On the Expressive Power of Deep Neural Networks (Raghu et al., 2017)

  On the Ability of Neural Nets to Express Distributions (Lee et al., 2017)

  On the Expressive Power of Deep Learning: A Tensor Analysis (C et al., 2016)

Why Depth?

Conventional wisdom:

- Depth boosts expressive power

  The Power of Depth for Feedforward Neural Networks (Eldan & Shamir, 2016)

  On the Expressive Power of Deep Neural Networks (Raghu et al., 2017)

  On the Ability of Neural Nets to Express Distributions (Lee et al., 2017)

  On the Expressive Power of Deep Learning: A Tensor Analysis (C et al., 2016)

- But complicates optimization

  Exploring Strategies for Training Deep Neural Networks (Larochelle et al., 2009)

  Understanding the Difficulty of Training Deep Feedforward Neural Networks (Glorot & Bengio, 2010)

  Training Very Deep Networks (Srivastava et al., 2015)

Why Depth?

Conventional wisdom:

- Depth boosts expressive power

  The Power of Depth for Feedforward Neural Networks (Eldan & Shamir, 2016)

  On the Expressive Power of Deep Neural Networks (Raghu et al., 2017)

  On the Ability of Neural Nets to Express Distributions (Lee et al., 2017)

  On the Expressive Power of Deep Learning: A Tensor Analysis (C et al., 2016)

- But complicates optimization

  Exploring Strategies for Training Deep Neural Networks (Larochelle et al., 2009)

  Understanding the Difficulty of Training Deep Feedforward Neural Networks (Glorot & Bengio, 2010)

  Training Very Deep Networks (Srivastava et al., 2015)

This work:

Depth can accelerate optimization!

Common Approach – Landscape Characterization

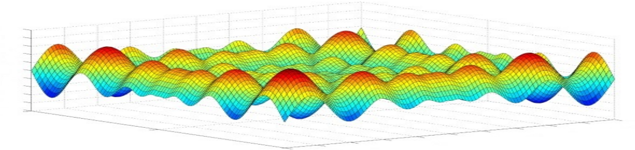
Optimization in deep learning typically studied via characterization of critical points (local min, saddles) in training objective

Deep Learning Without Poor Local Minima (Kawaguchi, 2016)

Identity Matters in Deep Learning (Hardt & Ma, 2016)

No Bad Local Minima: Data Independent Training Error Guarantees... (Soudry & Carmon, 2016)

Spurious Local Minima are Common in Two-Layer ReLU Neural Networks (Safran & Shamir, 2017)



Common Approach – Landscape Characterization

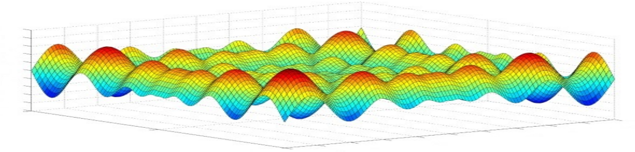
Optimization in deep learning typically studied via characterization of critical points (local min, saddles) in training objective

Deep Learning Without Poor Local Minima (Kawaguchi, 2016)

Identity Matters in Deep Learning (Hardt & Ma, 2016)

No Bad Local Minima: Data Independent Training Error Guarantees... (Soudry & Carmon, 2016)

Spurious Local Minima are Common in Two-Layer ReLU Neural Networks (Safran & Shamir, 2017)



This approach prefers convex objectives – cannot argue in favor of depth

Common Approach – Landscape Characterization

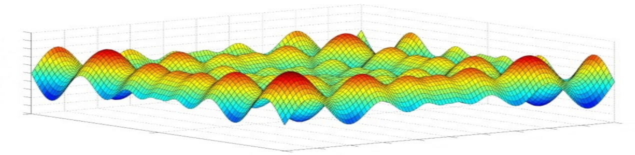
Optimization in deep learning typically studied via characterization of critical points (local min, saddles) in training objective

Deep Learning Without Poor Local Minima (Kawaguchi, 2016)

Identity Matters in Deep Learning (Hardt & Ma, 2016)

No Bad Local Minima: Data Independent Training Error Guarantees... (Soudry & Carmon, 2016)

Spurious Local Minima are Common in Two-Layer ReLU Neural Networks (Safran & Shamir, 2017)



This approach prefers convex objectives – cannot argue in favor of depth

To do so, one must consider dynamics of optimization algorithm

Decoupling Optimization from Expressiveness

Problem:

Expressiveness can interfere with our study – deeper networks may optimize "faster" per being able to reach lower training error

Decoupling Optimization from Expressiveness

Problem:

Expressiveness can interfere with our study – deeper networks may optimize "faster" per being able to reach lower training error

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**

Warm-Up

Training objective for scalar linear regression with ℓ_p loss:

$$L(\mathbf{w}) = \sum_{(\mathbf{x},y)\in S} \tfrac{1}{p}(\mathbf{x}^\top \mathbf{w} - y)^p$$

Warm-Up

Training objective for scalar linear regression with ℓ_p loss:

$$L(\mathbf{w}) = \sum_{(\mathbf{x},y)\in S} \frac{1}{p}(\mathbf{x}^\top \mathbf{w} - y)^p$$

Gradient:

$$\nabla_{\mathbf{w}} = \sum_{(\mathbf{x},y)\in S} (\mathbf{x}^\top \mathbf{w} - y)^{p-1}\mathbf{x}$$

Warm-Up

Training objective for scalar linear regression with ℓ_p loss:

$$L(\mathbf{w}) = \sum_{(\mathbf{x},y)\in S} \tfrac{1}{p}(\mathbf{x}^\top \mathbf{w} - y)^p$$

Gradient:

$$\nabla_{\mathbf{w}} = \sum_{(\mathbf{x},y)\in S} (\mathbf{x}^\top \mathbf{w} - y)^{p-1}\mathbf{x}$$

Gradient descent:

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \eta \nabla_{\mathbf{w}^{(t)}}$$

Warm-Up (cont')

Now overparameterize – replace $\mathbf{w}$ by vector $\mathbf{w}_1$ times scalar ω_2:

$$L(\mathbf{w}_1, \omega_2) = \sum_{(\mathbf{x},y)\in S} \tfrac{1}{p}(\mathbf{x}^\top \mathbf{w}_1 \omega_2 - y)^p$$

Warm-Up (cont')

Now overparameterize – replace $\mathbf{w}$ by vector $\mathbf{w}_1$ times scalar ω_2:

$$L(\mathbf{w}_1, \omega_2) = \sum_{(\mathbf{x},y)\in S} \tfrac{1}{p}(\mathbf{x}^\top \mathbf{w}_1\omega_2 - y)^p$$

Gradients:

$$\begin{aligned}
\nabla_{\mathbf{w}_1} &= \sum_{(\mathbf{x},y)\in S} (\mathbf{x}^\top \mathbf{w}_1\omega_2 - y)^{p-1}\mathbf{x}\omega_2 \\
\nabla_{\omega_2} &= \sum_{(\mathbf{x},y)\in S} (\mathbf{x}^\top \mathbf{w}_1\omega_2 - y)^{p-1}\mathbf{x}^\top \mathbf{w}_1
\end{aligned}$$

Warm-Up (cont')

Now overparameterize – replace $\mathbf{w}$ by vector $\mathbf{w}_1$ times scalar ω_2:

$$L(\mathbf{w}_1, \omega_2) = \sum_{(\mathbf{x},y)\in S} \tfrac{1}{p}(\mathbf{x}^\top \mathbf{w}_1\omega_2 - y)^p$$

Gradients:

$$\nabla_{\mathbf{w}_1} = \sum_{(\mathbf{x},y)\in S} (\mathbf{x}^\top \mathbf{w}_1\omega_2 - y)^{p-1}\mathbf{x}\omega_2$$

$$\nabla_{\omega_2} = \sum_{(\mathbf{x},y)\in S} (\mathbf{x}^\top \mathbf{w}_1\omega_2 - y)^{p-1}\mathbf{x}^\top \mathbf{w}_1$$

Gradient descent:

$$\mathbf{w}_1{}^{(t+1)} \hookleftarrow \mathbf{w}_1{}^{(t)} - \eta\nabla_{\mathbf{w}_1{}^{(t)}}$$

$$\omega_2{}^{(t+1)} \hookleftarrow \omega_2{}^{(t)} - \eta\nabla_{\omega_2{}^{(t)}}$$

Warm-Up (cont')

Question:

How does $\mathbf{w} = \mathbf{w}_1 \omega_2$ behave during gradient descent over $\mathbf{w}_1, \omega_2$?

Warm-Up (cont')

Question:

How does $\mathbf{w} = \mathbf{w}_1 \omega_2$ behave during gradient descent over $\mathbf{w}_1, \omega_2$?

Observation:

Assuming small learning rate ($\eta \ll 1$) and near-zero init ($\mathbf{w}_1{}^{(0)}, \omega_2{}^{(0)} \approx 0$):

$$\begin{aligned} \mathbf{w}^{(t+1)} &= \mathbf{w}_1{}^{(t+1)} \omega_2{}^{(t+1)} \\ &\leftarrow (\mathbf{w}_1{}^{(t)} - \eta \nabla_{\mathbf{w}_1{}^{(t)}})(\omega_2{}^{(t)} - \eta \nabla_{\omega_2{}^{(t)}}) \\ &= \ldots \\ &\approx \mathbf{w}^{(t)} - \rho^{(t)} \nabla_{\mathbf{w}^{(t)}} - \sum\nolimits_{\tau=1}^{t-1} \mu^{(t,\tau)} \nabla_{\mathbf{w}^{(\tau)}} \end{aligned}$$

for suitable $\rho^{(t)}, \mu^{(t,\tau)} \in \mathbb{R}$

Warm-Up (cont')

Question:

How does $\mathbf{w} = \mathbf{w}_1 \omega_2$ behave during gradient descent over $\mathbf{w}_1, \omega_2$?

Observation:

Assuming small learning rate ($\eta \ll 1$) and near-zero init ($\mathbf{w}_1{}^{(0)}, \omega_2{}^{(0)} \approx 0$):

$$\begin{aligned} \mathbf{w}^{(t+1)} &= \mathbf{w}_1{}^{(t+1)} \omega_2{}^{(t+1)} \\ &\leftarrow (\mathbf{w}_1{}^{(t)} - \eta \nabla_{\mathbf{w}_1{}^{(t)}})(\omega_2{}^{(t)} - \eta \nabla_{\omega_2{}^{(t)}}) \\ &= \ldots \\ &\approx \mathbf{w}^{(t)} - \rho^{(t)} \nabla_{\mathbf{w}^{(t)}} - \sum\nolimits_{\tau=1}^{t-1} \mu^{(t,\tau)} \nabla_{\mathbf{w}^{(\tau)}} \end{aligned}$$

for suitable $\rho^{(t)}, \mu^{(t,\tau)} \in \mathbb{R}$

Overparameterization by single scalar gave certain adaptive learning rate and momentum

Outline

Formal Setup

Depth-N linear neural network:

$$\mathbf{x} \mapsto W_N W_{N-1} \cdots W_1 \mathbf{x} \qquad (W_j \text{ -- weight matrices})$$

Formal Setup

Depth-N linear neural network:

$$\mathbf{x} \mapsto W_N W_{N-1} \cdots W_1 \mathbf{x} \qquad (W_j \text{ – weight matrices})$$

Define the **end-to-end weight matrix**:

$$W_e := W_N W_{N-1} \cdots W_1$$

Formal Setup

Depth-N linear neural network:

$$\mathbf{x} \mapsto W_N W_{N-1} \cdots W_1 \mathbf{x} \qquad (W_j \text{ – weight matrices})$$

Define the **end-to-end weight matrix**:

$$W_e := W_N W_{N-1} \cdots W_1$$

Given loss $L(\cdot)$ over linear model, we have **overparameterized loss**:

$$L^N(W_1, \ldots, W_N) := L(W_e)$$

Formal Setup

Depth-N linear neural network:

$$\mathbf{x} \mapsto W_N W_{N-1} \cdots W_1 \mathbf{x} \qquad (W_j \text{ – weight matrices})$$

Define the **end-to-end weight matrix**:

$$W_e := W_N W_{N-1} \cdots W_1$$

Given loss $L(\cdot)$ over linear model, we have **overparameterized loss**:

$$L^N(W_1, \ldots, W_N) := L(W_e)$$

Question:

How does W_e behave during gradient descent over $W_1 \ldots W_N$?

End-to-End Update Rule

Gradient descent over $W_1 \ldots W_N$:

$$W_j^{(t+1)} \leftarrow W_j^{(t)} - \eta \frac{\partial L^N}{\partial W_j}(W_1^{(t)}, \ldots, W_N^{(t)}) \quad \forall j$$

End-to-End Update Rule

Gradient descent over $W_1 \dots W_N$:

$$W_j^{(t+1)} \leftarrow W_j^{(t)} - \eta \frac{\partial L^N}{\partial W_j}(W_1^{(t)}, \dots, W_N^{(t)}) \quad \forall j$$

Theorem

Assuming small learning rate ($\eta \ll 1$) and near-zero init ($W_j^{(0)} \approx 0 \;\; \forall j$), W_e follows the ***end-to-end update rule****:*

$$W_e^{(t+1)} \leftarrow W_e^{(t)} - \eta \sum_{j=1}^{N} \left[W_e^{(t)} (W_e^{(t)})^\top \right]^{\frac{j-1}{N}} \frac{dL}{dW}(W_e^{(t)}) \left[(W_e^{(t)})^\top W_e^{(t)} \right]^{\frac{N-j}{N}}$$

End-to-End Update Rule

Gradient descent over $W_1 \dots W_N$:

$$W_j^{(t+1)} \leftarrow W_j^{(t)} - \eta \frac{\partial L^N}{\partial W_j}(W_1^{(t)}, \dots, W_N^{(t)}) \quad \forall j$$

Theorem

Assuming small learning rate ($\eta \ll 1$) and near-zero init ($W_j^{(0)} \approx 0 \;\; \forall j$), W_e follows the ***end-to-end update rule****:*

$$W_e^{(t+1)} \leftarrow W_e^{(t)} - \eta \sum_{j=1}^{N} \left[W_e^{(t)} (W_e^{(t)})^\top \right]^{\frac{j-1}{N}} \frac{dL}{dW}(W_e^{(t)}) \left[(W_e^{(t)})^\top W_e^{(t)} \right]^{\frac{N-j}{N}}$$

Overparameterization with deep linear net gives closed-form update rule! No dependence on layer widths!

End-to-End Update Rule (cont')

$$W_e^{(t+1)} \hookleftarrow W_e^{(t)} - \eta \sum_{j=1}^{N} \left[W_e^{(t)} (W_e^{(t)})^\top \right]^{\frac{j-1}{N}} \frac{dL}{dW}(W_e^{(t)}) \left[(W_e^{(t)})^\top W_e^{(t)} \right]^{\frac{N-j}{N}}$$

Proof sketch:

End-to-End Update Rule (cont')

$$W_e^{(t+1)} \hookleftarrow W_e^{(t)} - \eta \sum\nolimits_{j=1}^{N} \left[W_e^{(t)} (W_e^{(t)})^\top \right]^{\frac{j-1}{N}} \frac{dL}{dW}(W_e^{(t)}) \left[(W_e^{(t)})^\top W_e^{(t)} \right]^{\frac{N-j}{N}}$$

Proof sketch:

- Assumption $\eta \ll 1$ implies:

  discrete updates $\approx$ continuous differential equations

End-to-End Update Rule (cont')

$$W_e^{(t+1)} \hookleftarrow W_e^{(t)} - \eta \sum_{j=1}^{N} \left[W_e^{(t)} (W_e^{(t)})^\top \right]^{\frac{j-1}{N}} \frac{dL}{dW}(W_e^{(t)}) \left[(W_e^{(t)})^\top W_e^{(t)} \right]^{\frac{N-j}{N}}$$

Proof sketch:

- Assumption $\eta \ll 1$ implies:

  discrete updates $\approx$ continuous differential equations

- Assumption $W_j^{(0)} \approx 0$ implies:

$$(W_{j+1}^{(0)})^\top W_{j+1}^{(0)} \approx W_j^{(0)} (W_j^{(0)})^\top$$

End-to-End Update Rule (cont')

$$W_e^{(t+1)} \leftarrow W_e^{(t)} - \eta \sum_{j=1}^{N} \left[W_e^{(t)} (W_e^{(t)})^\top \right]^{\frac{j-1}{N}} \frac{dL}{dW}(W_e^{(t)}) \left[(W_e^{(t)})^\top W_e^{(t)} \right]^{\frac{N-j}{N}}$$

Proof sketch:

- Assumption $\eta \ll 1$ implies:

  discrete updates $\approx$ continuous differential equations

- Assumption $W_j^{(0)} \approx 0$ implies:

$$(W_{j+1}^{(0)})^\top W_{j+1}^{(0)} \approx W_j^{(0)} (W_j^{(0)})^\top$$

- We show that $(W_{j+1}^{(t)})^\top W_{j+1}^{(t)} = W_j^{(t)} (W_j^{(t)})^\top$ continues $\forall t$

End-to-End Update Rule (cont')

$$W_e^{(t+1)} \leftarrow W_e^{(t)} - \eta \sum_{j=1}^{N} \left[W_e^{(t)} (W_e^{(t)})^\top \right]^{\frac{j-1}{N}} \frac{dL}{dW}(W_e^{(t)}) \left[(W_e^{(t)})^\top W_e^{(t)} \right]^{\frac{N-j}{N}}$$

Proof sketch:

- Assumption $\eta \ll 1$ implies:

  discrete updates $\approx$ continuous differential equations

- Assumption $W_j^{(0)} \approx 0$ implies:

$$(W_{j+1}^{(0)})^\top W_{j+1}^{(0)} \approx W_j^{(0)} (W_j^{(0)})^\top$$

- We show that $(W_{j+1}^{(t)})^\top W_{j+1}^{(t)} = W_j^{(t)} (W_j^{(t)})^\top$ continues $\forall t$

- Left singular spaces of $W_j^{(t)}$ thus coincide with right ones of $W_{j+1}^{(t)}$

End-to-End Update Rule (cont')

$$W_e^{(t+1)} \leftarrow W_e^{(t)} - \eta \sum_{j=1}^{N} \left[W_e^{(t)} (W_e^{(t)})^\top \right]^{\frac{j-1}{N}} \frac{dL}{dW}(W_e^{(t)}) \left[(W_e^{(t)})^\top W_e^{(t)} \right]^{\frac{N-j}{N}}$$

Proof sketch:

- Assumption $\eta \ll 1$ implies:

  discrete updates $\approx$ continuous differential equations

- Assumption $W_j^{(0)} \approx 0$ implies:

$$(W_{j+1}^{(0)})^\top W_{j+1}^{(0)} \approx W_j^{(0)} (W_j^{(0)})^\top$$

- We show that $(W_{j+1}^{(t)})^\top W_{j+1}^{(t)} = W_j^{(t)} (W_j^{(t)})^\top$ continues $\forall t$
- Left singular spaces of $W_j^{(t)}$ thus coincide with right ones of $W_{j+1}^{(t)}$
- Products of the form $W_{j+m} \cdots W_{j+1} W_j$ then simplify

  $\implies$ update for end-to-end matrix W_e can be computed explicitly $\square$

End-to-End Update Rule (cont')

Question:

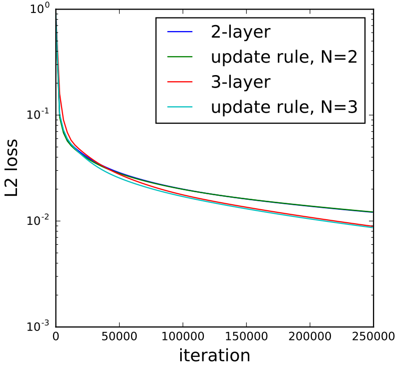
Do our assumptions (small learning rate, near-zero init) hold in practice?

End-to-End Update Rule (cont')

Question:

Do our assumptions (small learning rate, near-zero init) hold in practice?

Empirical validation:



Analytical update rule indeed complies with deep network optimization

End-to-End Update Rule – Interpretation

$$\boxed{W_e^{(t+1)} \hookleftarrow W_e^{(t)} - \eta \sum\nolimits_{j=1}^{N} \left[W_e^{(t)} (W_e^{(t)})^\top \right]^{\frac{j-1}{N}} \frac{dL}{dW}(W_e^{(t)}) \left[(W_e^{(t)})^\top W_e^{(t)} \right]^{\frac{N-j}{N}}}$$

End-to-End Update Rule – Interpretation

$$W_e^{(t+1)} \hookleftarrow W_e^{(t)} - \eta \sum\nolimits_{j=1}^{N} \left[W_e^{(t)} (W_e^{(t)})^\top \right]^{\frac{j-1}{N}} \frac{dL}{dW}(W_e^{(t)}) \left[(W_e^{(t)})^\top W_e^{(t)} \right]^{\frac{N-j}{N}}$$

Claim

End-to-end update rule can be written as:

$$vec\left[W_e^{(t+1)}\right] \hookleftarrow vec\left[W_e^{(t)}\right] - \eta \cdot P_{W_e^{(t)}} vec\left[\tfrac{dL}{dW}(W_e^{(t)})\right]$$

where $P_{W_e^{(t)}}$ is a preconditioning (PSD) matrix with:

- *Eigendirections: formed by sing vectors of $W_e^{(t)}$*
- *Eigenvalues: large (small) if sing vectors have high (low) sing values*

End-to-End Update Rule – Interpretation

$$W_e^{(t+1)} \leftarrow W_e^{(t)} - \eta \sum_{j=1}^{N} \left[W_e^{(t)} (W_e^{(t)})^\top \right]^{\frac{j-1}{N}} \frac{dL}{dW}(W_e^{(t)}) \left[(W_e^{(t)})^\top W_e^{(t)} \right]^{\frac{N-j}{N}}$$

Claim

End-to-end update rule can be written as:

$$vec\left[W_e^{(t+1)}\right] \leftarrow vec\left[W_e^{(t)}\right] - \eta \cdot P_{W_e^{(t)}} vec\left[\tfrac{dL}{dW}(W_e^{(t)})\right]$$

where $P_{W_e^{(t)}}$ is a preconditioning (PSD) matrix with:

- *Eigendirections: formed by sing vectors of $W_e^{(t)}$*
- *Eigenvalues: large (small) if sing vectors have high (low) sing values*

Since we assume near-zero init ($W_e^{(0)} \approx 0$):

Overparameterization induces preconditioning, that promotes movement along directions already taken!

Single Output Case

Single Output Case

Claim

In single (scalar) output case, end-to-end update rule can be written as:

$$W_e^{(t+1)} \leftarrow W_e^{(t)} - \eta \|W_e^{(t)}\|_2^{2-\frac{2}{N}} \left(\frac{dL}{dW}(W_e^{(t)}) + (N-1) Pr_{W_e^{(t)}} \left\{ \frac{dL}{dW}(W_e^{(t)}) \right\} \right)$$

where $Pr_{W_e^{(t)}}\{\cdot\}$ *is the projection onto the direction of* $W_e^{(t)}$

Single Output Case

Claim

In single (scalar) output case, end-to-end update rule can be written as:

$$W_e^{(t+1)} \leftarrow W_e^{(t)} - \eta \| W_e^{(t)} \|_2^{2-\frac{2}{N}} \left(\frac{dL}{dW}(W_e^{(t)}) + (N-1) Pr_{W_e^{(t)}} \left\{ \frac{dL}{dW}(W_e^{(t)}) \right\} \right)$$

where $Pr_{W_e^{(t)}}\{\cdot\}$ is the projection onto the direction of $W_e^{(t)}$

Interpretation:

- $\| W_e^{(t)} \|_2^{2-\frac{2}{N}}$ – adaptive learning rate, increases steps away from init

Single Output Case

Claim

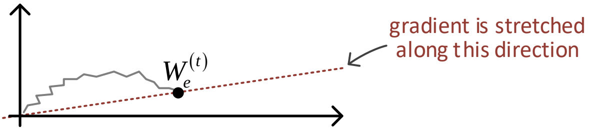
In single (scalar) output case, end-to-end update rule can be written as:

$$W_e^{(t+1)} \hookleftarrow W_e^{(t)} - \eta \|W_e^{(t)}\|_2^{2-\frac{2}{N}} \left(\frac{dL}{dW}(W_e^{(t)}) + (N-1) Pr_{W_e^{(t)}} \left\{ \frac{dL}{dW}(W_e^{(t)}) \right\} \right)$$

where $Pr_{W_e^{(t)}}\{\cdot\}$ *is the projection onto the direction of* $W_e^{(t)}$

Interpretation:

- $\|W_e^{(t)}\|_2^{2-\frac{2}{N}}$ – adaptive learning rate, increases steps away from init
- $(N-1)Pr_{W_e^{(t)}}\{\frac{dL}{dW}(W_e^{(t)})\}$ – "momentum", favors azimuth taken so far



Overparameterization Goes Beyond Regularization

End-to-end update rule (single output case):

$$W_e^{(t+1)} \leftarrow W_e^{(t)} - \eta \|W_e^{(t)}\|_2^{2-\frac{2}{N}} \left(\frac{dL}{dW}(W_e^{(t)}) + (N-1) Pr_{W_e^{(t)}} \left\{ \frac{dL}{dW}(W_e^{(t)}) \right\} \right)$$

Question:

Is this equivalent to gradient descent over some regularized objective?

Overparameterization Goes Beyond Regularization

End-to-end update rule (single output case):

$$W_e^{(t+1)} \hookleftarrow W_e^{(t)} - \eta \|W_e^{(t)}\|_2^{2-\frac{2}{N}} \left(\tfrac{dL}{dW}(W_e^{(t)}) + (N{-}1) Pr_{W_e^{(t)}} \left\{ \tfrac{dL}{dW}(W_e^{(t)}) \right\} \right)$$

Question:

Is this equivalent to gradient descent over some regularized objective?

Theorem

Assuming $\frac{dL}{dW}(0) \neq 0$, *there exists no func (of* W*) whose gradient is:*

$$\|W\|_2^{2-\frac{2}{N}} \left(\tfrac{dL}{dW}(W) + (N{-}1) Pr_W \{ \tfrac{dL}{dW}(W) \} \right)$$

Overparameterization Goes Beyond Regularization

End-to-end update rule (single output case):

$$W_e^{(t+1)} \leftarrow W_e^{(t)} - \eta \|W_e^{(t)}\|_2^{2-\frac{2}{N}} \left(\frac{dL}{dW}(W_e^{(t)}) + (N-1)Pr_{W_e^{(t)}}\left\{ \frac{dL}{dW}(W_e^{(t)}) \right\} \right)$$

Question:

Is this equivalent to gradient descent over some regularized objective?

Theorem

Assuming $\frac{dL}{dW}(0) \neq 0$, there exists no func (of W) whose gradient is:

$$\|W\|_2^{2-\frac{2}{N}} \left(\frac{dL}{dW}(W) + (N-1)Pr_W\{\frac{dL}{dW}(W)\} \right)$$

Effect of overparameterization can't be attained through any modification of the objective!

Overparameterization Goes Beyond Regularization (con't)

Proof sketch:

Overparameterization Goes Beyond Regularization (con't)

Proof sketch:

- Fundamental theorem for line integrals:

  $\oint_{\Gamma} \nabla g = 0$ for any closed curve Γ and differentiable func $g(\cdot)$

Overparameterization Goes Beyond Regularization (con't)

Proof sketch:

- Fundamental theorem for line integrals:

  $\oint_{\Gamma} \nabla g = 0$ for any closed curve Γ and differentiable func $g(\cdot)$

- Define $F(W) := \|W\|_2^{2-\frac{2}{N}} \left(\frac{dL}{dW}(W) + (N-1) Pr_W\{\frac{dL}{dW}(W)\} \right)$

Overparameterization Goes Beyond Regularization (con't)

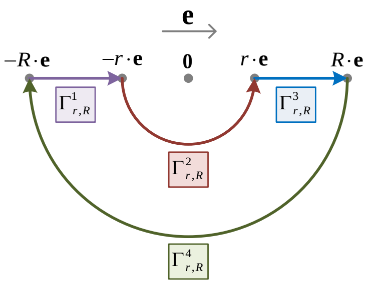
Proof sketch:

- Fundamental theorem for line integrals:

  $\oint_\Gamma \nabla g = 0$ for any closed curve Γ and differentiable func $g(\cdot)$

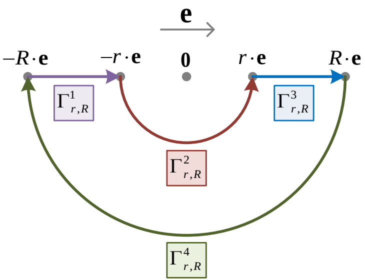
- Define $F(W) := \|W\|_2^{2-\frac{2}{N}} \left(\frac{dL}{dW}(W) + (N-1) Pr_W\{\frac{dL}{dW}(W)\} \right)$

- Let $\mathbf{e} := \frac{dL}{dW}(0) / \left\| \frac{dL}{dW}(0) \right\|$, and $\Gamma_{r,R}$ be a curve as follows:



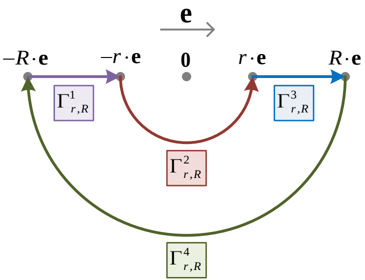
Overparameterization Goes Beyond Regularization (con't)

Proof sketch (cont'):



Overparameterization Goes Beyond Regularization (con't)

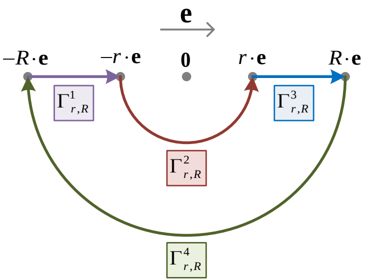
Proof sketch (cont'):



- We compute a lower bound on $\oint_{\Gamma_{r,R}} F = \sum_{i=1}^{4} \int_{\Gamma^i_{r,R}} F$

Overparameterization Goes Beyond Regularization (con't)

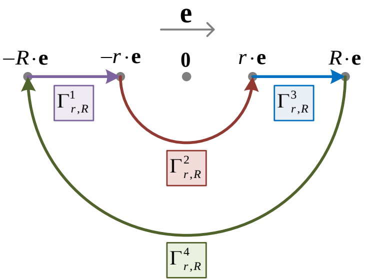
Proof sketch (cont'):



- We compute a lower bound on $\oint_{\Gamma_{r,R}} F = \sum_{i=1}^{4} \int_{\Gamma^i_{r,R}} F$
- Show lower bound is positive for sufficiently small r, R

Overparameterization Goes Beyond Regularization (con't)

Proof sketch (cont'):



- We compute a lower bound on $\oint_{\Gamma_{r,R}} F = \sum_{i=1}^{4} \int_{\Gamma^i_{r,R}} F$
- Show lower bound is positive for sufficiently small r, R
- $F(\cdot)$ thus contradicts fundamental theorem for line integrals

  $\implies$ cannot be a gradient! $\square$

Illustration of Acceleration

Illustration of Acceleration

Simple training objective (ℓ_p regression):

$$L(w_1, w_2) = \frac{1}{p}(w_1 - y_1)^p + \frac{1}{p}(w_2 - y_2)^p$$

Illustration of Acceleration

Simple training objective (ℓ_p regression):

$$L(w_1, w_2) = \frac{1}{p}(w_1 - y_1)^p + \frac{1}{p}(w_2 - y_2)^p$$

Gradient descent:

$$w_i^{(t+1)} \leftarrow w_i^{(t)} - \eta(w_i^{(t)} - y_i)^{p-1} \qquad i = 1, 2$$

Illustration of Acceleration

Simple training objective (ℓ_p regression):

$$L(w_1, w_2) = \tfrac{1}{p}(w_1 - y_1)^p + \tfrac{1}{p}(w_2 - y_2)^p$$

Gradient descent:

$$w_i^{(t+1)} \hookleftarrow w_i^{(t)} - \eta(w_i^{(t)} - y_i)^{p-1} \qquad i = 1, 2$$

Change of variables $\Delta_i = w_i - y_i$:

$$\Delta_i^{(t+1)} \hookleftarrow \Delta_i^{(t)}\left(1 - \eta(\Delta_i^{(t)})^{p-2}\right) \qquad i = 1, 2$$

Illustration of Acceleration

Simple training objective (ℓ_p regression):

$$L(w_1, w_2) = \tfrac{1}{p}(w_1 - y_1)^p + \tfrac{1}{p}(w_2 - y_2)^p$$

Gradient descent:

$$w_i^{(t+1)} \leftarrow w_i^{(t)} - \eta(w_i^{(t)} - y_i)^{p-1} \qquad i = 1, 2$$

Change of variables $\Delta_i = w_i - y_i$:

$$\Delta_i^{(t+1)} \leftarrow \Delta_i^{(t)}\left(1 - \eta(\Delta_i^{(t)})^{p-2}\right) \qquad i = 1, 2$$

To prevent divergence, learning rate must satisfy:

$$\eta < \frac{2}{\max\left\{|\Delta_1^{(0)}|, |\Delta_2^{(0)}|\right\}^{p-2}} \underbrace{\approx}_{w_i^{(0)} \approx 0} \frac{2}{\max\left\{|y_1|, |y_2|\right\}^{p-2}}$$

Illustration of Acceleration (cont')

$$L(w_1, w_2) = \frac{1}{p}(w_1 - y_1)^p + \frac{1}{p}(w_2 - y_2)^p$$

$$\Delta_i = w_i - y_i$$

$$\Delta_i^{(t+1)} \leftarrow \Delta_i^{(t)}\left(1 - \eta(\Delta_i^{(t)})^{p-2}\right)$$

$$\eta < 2/\max\left\{|y_1|, |y_2|\right\}^{p-2}$$

Illustration of Acceleration (cont')

$$L(w_1, w_2) = \frac{1}{p}(w_1 - y_1)^p + \frac{1}{p}(w_2 - y_2)^p$$

$$\Delta_i = w_i - y_i$$

$$\Delta_i^{(t+1)} \leftarrow \Delta_i^{(t)}(1 - \eta(\Delta_i^{(t)})^{p-2})$$

$$\eta < 2/\max\left\{|y_1|, |y_2|\right\}^{p-2}$$

Suppose problem is ill-conditioned: $y_1 \gg y_2$

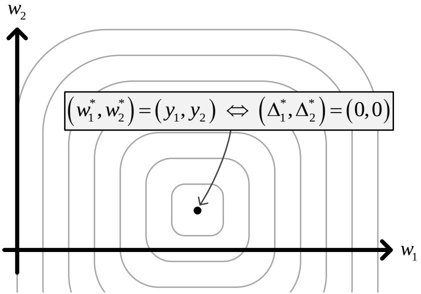


Illustration of Acceleration (cont')

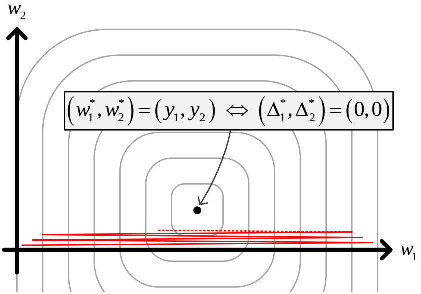
$$L(w_1, w_2) = \frac{1}{p}(w_1 - y_1)^p + \frac{1}{p}(w_2 - y_2)^p$$

$$\Delta_i = w_i - y_i$$

$$\Delta_i^{(t+1)} \hookleftarrow \Delta_i^{(t)}\left(1 - \eta(\Delta_i^{(t)})^{p-2}\right)$$

$$\eta < 2/\max\{|y_1|, |y_2|\}^{p-2}$$

Suppose problem is ill-conditioned: $y_1 \gg y_2$



If $p > 2$: $\eta \ll \frac{2}{y_2^{p-2}} \implies$ coordinate 2 will converge slowly

Illustration of Acceleration (cont')

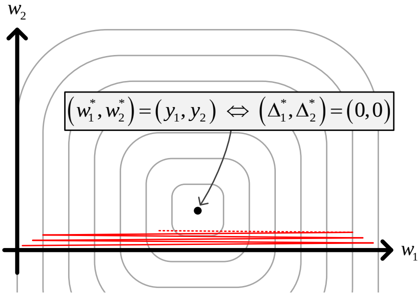
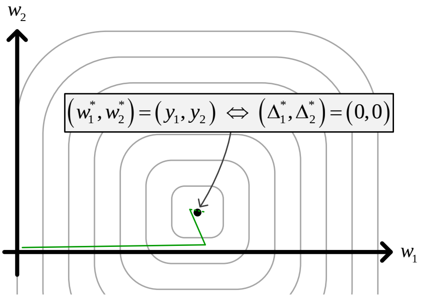


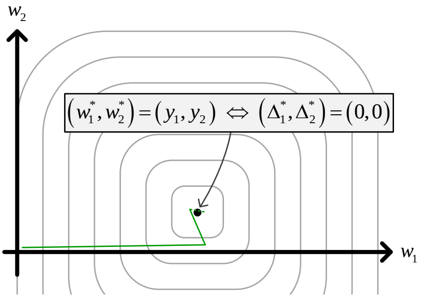
Illustration of Acceleration (cont')



With overparameterization (end-to-end update rule):

- Learning rate effectively multiplied by $(w_1 + w_2)^{2-\frac{2}{N}}$

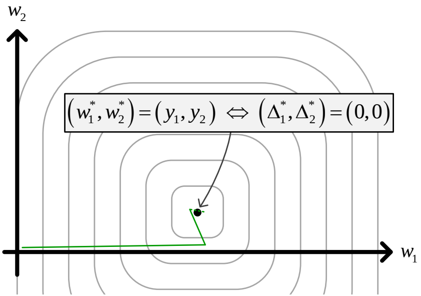
Illustration of Acceleration (cont')



With overparameterization (end-to-end update rule):

- Learning rate effectively multiplied by $(w_1 + w_2)^{2-\frac{2}{N}}$
- As coordinate 1 converges, coordinate 2 accelerates by $\approx y_1^{2-\frac{2}{N}}$

Illustration of Acceleration (cont')



With overparameterization (end-to-end update rule):

- Learning rate effectively multiplied by $(w_1 + w_2)^{2-\frac{2}{N}}$
- As coordinate 1 converges, coordinate 2 accelerates by $\approx y_1^{2-\frac{2}{N}}$
- Optimization may begin with small steps to avoid divergence, increasing step size as safe grounds are reached

Outline

ℓ_p Regression

ℓ_p Regression

Dataset: Scalar regression problem from UCI ML Repository

ℓ_p Regression

Dataset: Scalar regression problem from UCI ML Repository

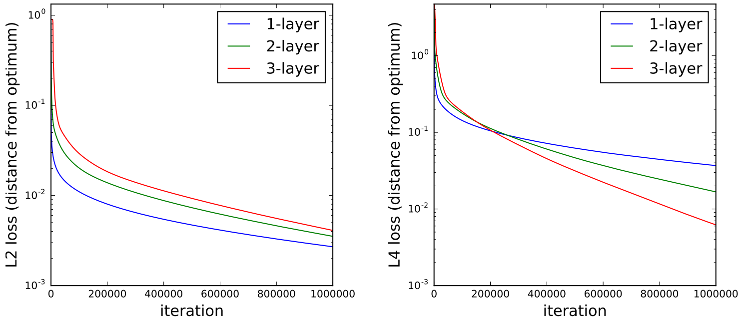
Models: Linear nets with size-1 hidden layers (no computational overhead)

ℓ_p Regression

Dataset: Scalar regression problem from UCI ML Repository

Models: Linear nets with size-1 hidden layers (no computational overhead)

Results:

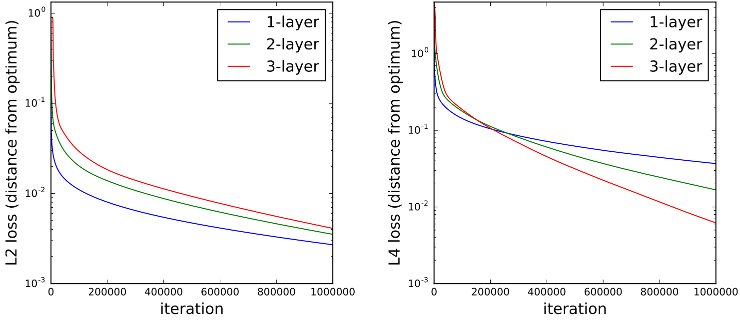


ℓ_p Regression

Dataset: Scalar regression problem from UCI ML Repository

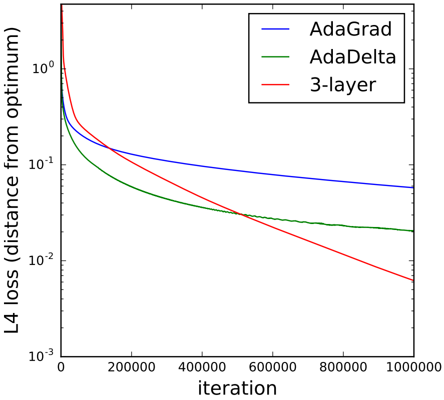
Models: Linear nets with size-1 hidden layers (no computational overhead)

Results:

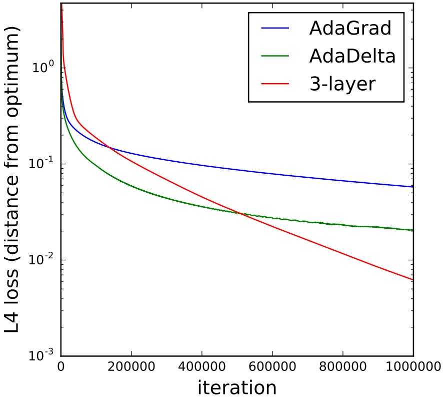


Overparameterization significantly accelerated ℓ_p regression for $p > 2$ (in line with qualitative analysis)!

Overparameterization vs. Explicit Accelerators



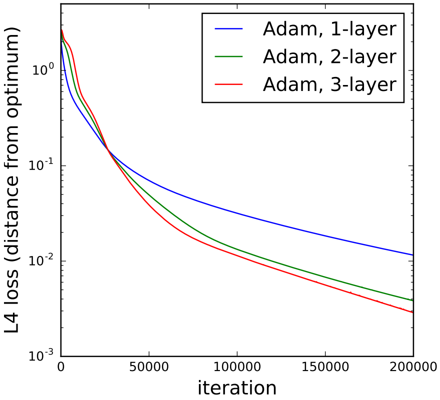
Overparameterization vs. Explicit Accelerators



Implicit acceleration of overparameterization was faster than explicit acceleration of AdaGrad and AdaDelta!

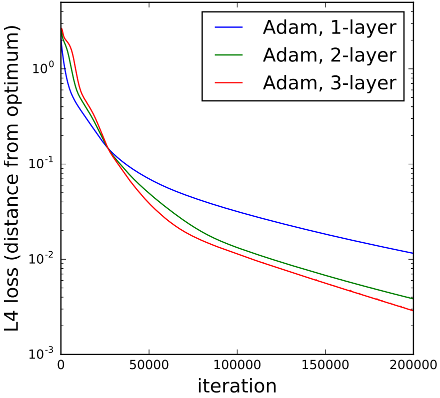
Overparameterization vs. Explicit Accelerators (cont')

Overparameterization was slower than Adam, but when applied on top:



Overparameterization vs. Explicit Accelerators (cont')

Overparameterization was slower than Adam, but when applied on top:



Overparameterization may help not only gradient descent, but also state-of-the-art algorithms

Going Deeper with Residual Networks

Question:

If depth indeed accelerates, why not go deeper and deeper?

Going Deeper with Residual Networks

Question:

If depth indeed accelerates, why not go deeper and deeper?

$$W_e^{(t+1)} \leftarrow W_e^{(t)} - \eta \| W_e^{(t)} \|_2^{2-\frac{2}{N}} \left(\frac{dL}{dW}(W_e^{(t)}) + (N-1) Pr_{W_e^{(t)}} \left\{ \frac{dL}{dW}(W_e^{(t)}) \right\} \right)$$

Vanishing gradient problem:

With large N:

small weight init $\implies$ $W_e := W_N \cdots W_1$ starts extremely close to 0

$\implies$ steps are severely attenuated

Going Deeper with Residual Networks

Question:

If depth indeed accelerates, why not go deeper and deeper?

$$\boxed{W_e^{(t+1)} \leftarrow W_e^{(t)} - \eta \|W_e^{(t)}\|_2^{2-\frac{2}{N}} \left(\frac{dL}{dW}(W_e^{(t)}) + (N-1) Pr_{W_e^{(t)}} \left\{ \frac{dL}{dW}(W_e^{(t)}) \right\} \right)}$$

Vanishing gradient problem:

With large N:

small weight init $\implies$ $W_e := W_N \cdots W_1$ starts extremely close to 0

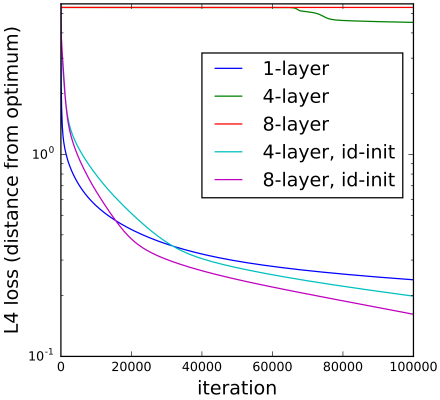
$\implies$ steps are severely attenuated

Resolution:

- Larger weight init (still small enough to prevent W_e "explosion")
- Specifically: identity init $\longleftrightarrow$ **linear residual networks**

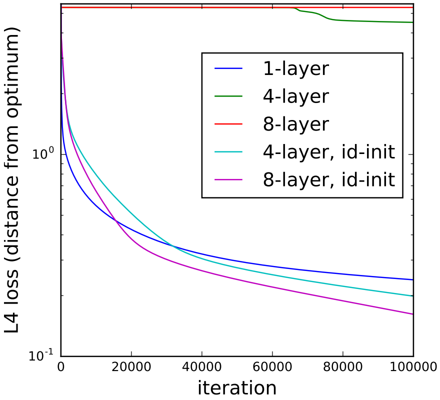
Going Deeper with Residual Networks (cont')

Deep nets with near-zero vs. identity init:



Going Deeper with Residual Networks (cont')

Deep nets with near-zero vs. identity init:



Identity init eliminated vanishing grad problem
$\Longrightarrow$ **linear residual nets can go deeper!**

Non-Linear Convolutional Network

[1] https://github.com/tensorflow/models/tree/master/tutorials/image/mnist

Non-Linear Convolutional Network

TensorFlow ConvNet tutorial for MNIST:[1]

- Architecture:
  - 5×5 conv, 32 channels, ReLU
  - 2×2 max pooling
  - 5×5 conv, 64 channels, ReLU
  - 2×2 max pooling
  - $7 \cdot 7 \cdot 64 \rightarrow 512$ dense, ReLU
  - $512 \rightarrow 10$ dense
- Training:
  - SGD+momentum, batch size 64
  - Momentum coeff 0.9, learning rate 0.01 (gradually decays)
  - Weight init $\sim \mathcal{N}(0, 0.1)$
  - Dropout 0.5 after last ReLU

[1] `https://github.com/tensorflow/models/tree/master/tutorials/image/mnist`

Non-Linear Convolutional Network

TensorFlow ConvNet tutorial for MNIST:[1]

- Architecture:
    - 5×5 conv, 32 channels, ReLU
    - 2×2 max pooling
    - 5×5 conv, 64 channels, ReLU
    - 2×2 max pooling
    - $7 \cdot 7 \cdot 64 \rightarrow 512$ dense, $512 \rightarrow 512$ dense, ReLU
    - $512 \rightarrow 10$ dense, $10 \rightarrow 10$ dense
- Training:
    - SGD+momentum, batch size 64
    - Momentum coeff 0.9, learning rate 0.01 (gradually decays)
    - Weight init $\sim \mathcal{N}(0, 0.1)$
    - Dropout 0.5 after last ReLU

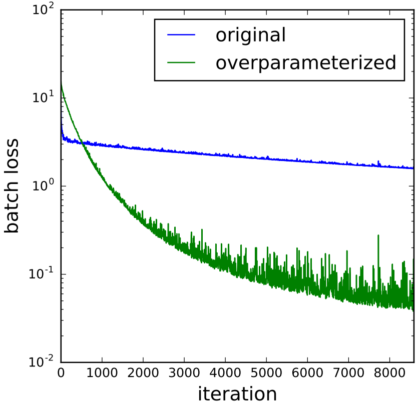
Sanity test:

Overparameterize – add excess matrix to each dense layer

[1] https://github.com/tensorflow/models/tree/master/tutorials/image/mnist

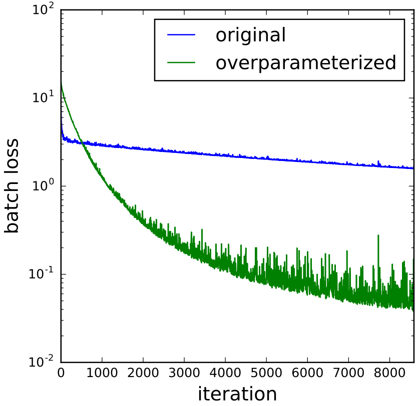
Non-Linear Convolutional Network (cont')

Training convergence:



Non-Linear Convolutional Network (cont')

Training convergence:



With $+15\%$ in params, overparameterization accelerated non-linear net by orders-of-magnitude!

Outline

1. Prelude
2. Theoretical Analysis
3. Experiments
4. Conclusion

Conclusion

Conclusion

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

Conclusion

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

Conclusion

- Depth radically changes obj landscape, makes it highly non-convex
- Conventional wisdom: this complicates optimization
- For linear neural networks (overparameterization), we show:

Conclusion

- Depth radically changes obj landscape, makes it highly non-convex
- Conventional wisdom: this complicates optimization
- For linear neural networks (overparameterization), we show:
  - Depth induces on gradient descent a preconditioning scheme

Conclusion

- Depth radically changes obj landscape, makes it highly non-convex
- Conventional wisdom: this complicates optimization
- For linear neural networks (overparameterization), we show:
  - Depth induces on gradient descent a preconditioning scheme
  - Preconditioner has closed-form description, and can be seen as certain combination of adaptive learning rate and "momentum"

Conclusion

- Depth radically changes obj landscape, makes it highly non-convex
- Conventional wisdom: this complicates optimization
- For linear neural networks (overparameterization), we show:
  - Depth induces on gradient descent a preconditioning scheme
  - Preconditioner has closed-form description, and can be seen as certain combination of adaptive learning rate and "momentum"
  - The effect cannot be attained via any regularizer

Conclusion

- Depth radically changes obj landscape, makes it highly non-convex
- Conventional wisdom: this complicates optimization
- For linear neural networks (overparameterization), we show:
  - Depth induces on gradient descent a preconditioning scheme
  - Preconditioner has closed-form description, and can be seen as certain combination of adaptive learning rate and "momentum"
  - The effect cannot be attained via any regularizer
  - Can lead to significant acceleration, with no change in expressiveness, and negligible computational overhead

Conclusion

- Depth radically changes obj landscape, makes it highly non-convex
- Conventional wisdom: this complicates optimization
- For linear neural networks (overparameterization), we show:
  - Depth induces on gradient descent a preconditioning scheme
  - Preconditioner has closed-form description, and can be seen as certain combination of adaptive learning rate and "momentum"
  - The effect cannot be attained via any regularizer
  - Can lead to significant acceleration, with no change in expressiveness, and negligible computational overhead
- **Perspective:**

  Understanding optimization in deep learning likely requires direct analysis of specific problems, models and algorithms

Outline

Thank You