Geometry of Optimization and Implicit Regularization in Deep Learning

B. Neyshabur, R. Tomioka, R. Salakhutdinov, N. Srebro. 2017.

Si Yi (Cathy) Meng

Oct 30, 2019

UBC MLRG

Introduction

- Learning relies on **inductive bias**.
    - Capacity control $\rightarrow$ generalization

Introduction

- Learning relies on **inductive bias**.
  - Capacity control $\rightarrow$ generalization
- Consider feedforward neural networks:
  - Hypothesis class: weight vectors
  - Highly expressive
  - In general, empirical risk minimization is an NP-hard problem.

Introduction

- Learning relies on **inductive bias**.
    - Capacity control $\rightarrow$ generalization
- Consider feedforward neural networks:
    - Hypothesis class: weight vectors
    - Highly expressive
    - In general, empirical risk minimization is an NP-hard problem.
- **Why do we succeed in learning such models?**

Outline

- Implicit regularization [3]
  - Is it the network size? Or is it something else?

Outline

- Implicit regularization [3]
  - Is it the network size? Or is it something else?
- Geometry of optimization [4]
  - Magnitude/scale measures for neural networks

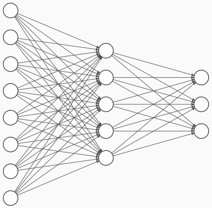
Outline

- Implicit regularization [3]
    - Is it the network size? Or is it something else?
- Geometry of optimization [4]
    - Magnitude/scale measures for neural networks
- Path-SGD [1]

Implicit Regularization

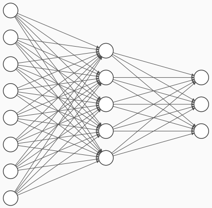
Implicit Regularization - Experiment

- D input features
- C output classes
- H hidden units, $\sigma_{\text{ReLU}}(x) = \max(0, x)$
- n training pairs



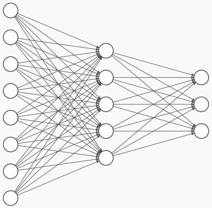
Implicit Regularization - Experiment

- D input features
- C output classes
- H hidden units, $\sigma_{\text{ReLU}}(x) = \max(0, x)$
- n training pairs
- *Truncated* softmax cross entropy loss



Implicit Regularization - Experiment

- D input features
- C output classes
- H hidden units, $\sigma_{\text{ReLU}}(x) = \max(0, x)$
- n training pairs
- *Truncated* softmax cross entropy loss
- SGD + momentum + diminishing step sizes
- No explicit regularization



Implicit Regularization - Experiment

What happens when we increase H?

Implicit Regularization - Experiment

What happens when we increase H?

Expectation:

- Training error $\downarrow$.
- Test error might initially $\downarrow$ and then $\uparrow$.

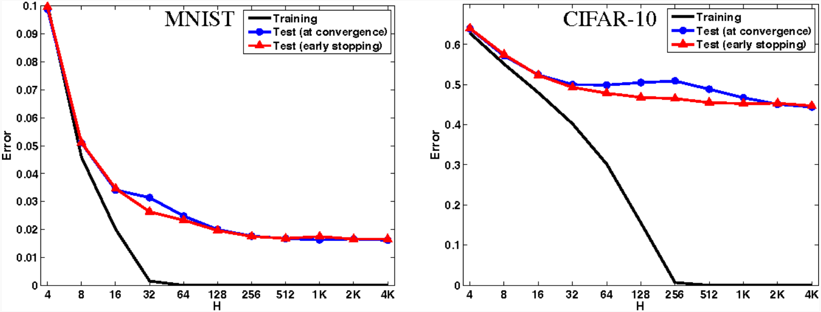
Implicit Regularization - Experiment

What happens when we increase H?

Expectation:

- Training error $\downarrow$.
- Test error might initially $\downarrow$ and then $\uparrow$.

Actual results:



Implicit Regularization - Experiment

- Perhaps it's due to the optimization algorithm:
  - Tries to find a solution with small *complexity*.
  - Increasing the network size might help lower this *complexity*.

Different optimization algorithms
$\implies$ Different implicit regularization
$\implies$ Different generlization

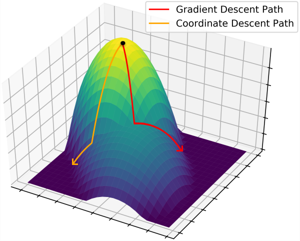


Figure 1: Gunasekar et al. https://bit.ly/32WEbXg

Geometry of Optimization

Geometry of Optimization

- Optimization is tied to a distance metric
  - Steepest descent w.r.t. ℓ_2 norm $\implies$ gradient descent
  - Steepest descent w.r.t. ℓ_1 norm $\implies$ coordinate descent
  - Steepest descent w.r.t. quadratic norm measured by the local (PD) Hessian $\implies$ Newton's method
  - Mirror descent w.r.t. entropic divergence $\implies$ exponentiated gradient descent
  - ...
- **What's the appropriate metric for neural networks?**

Notation

- Feedforward neural network $f_{G,w,\sigma} : \mathbb{R}^D \to \mathbb{R}^C$
- Represented by a directed acyclic graph $G(V, E)$

Notation

- Feedforward neural network $f_{G,w,\sigma} : \mathbb{R}^D \to \mathbb{R}^C$
- Represented by a directed acyclic graph $G(V, E)$
- $w :=$ weights vector
- $d :=$ depth of network

Notation

- Feedforward neural network $f_{G,w,\sigma} : \mathbb{R}^D \to \mathbb{R}^C$
- Represented by a directed acyclic graph $G(V, E)$
- $w :=$ weights vector
- $d :=$ depth of network
- $V_{\text{in}}^i = V_{\text{out}}^{d-i} :=$ set of hidden units in layer i
- $\sigma = \sigma_{\text{ReLU}}$

Notation

- Feedforward neural network $f_{G,w,\sigma} : \mathbb{R}^D \to \mathbb{R}^C$
- Represented by a directed acyclic graph $G(V, E)$
- $w :=$ weights vector
- $d :=$ depth of network
- $V_{\text{in}}^i = V_{\text{out}}^{d-i} :=$ set of hidden units in layer i
- $\sigma = \sigma_{\text{ReLU}}$
- $L(w) = \frac{1}{n} \sum_{i=1}^n \ell(f_{G,w,\sigma}(x_i), y_i)$ where $\{(x_i, y_i)\}$ are training examples

Notation

- Feedforward neural network $f_{G,w,\sigma} : \mathbb{R}^D \to \mathbb{R}^C$
- Represented by a directed acyclic graph $G(V, E)$
- $w :=$ weights vector
- $d :=$ depth of network
- $V_{\text{in}}^{i} = V_{\text{out}}^{d-i} :=$ set of hidden units in layer i
- $\sigma = \sigma_{\text{ReLU}}$
- $L(w) = \frac{1}{n} \sum_{i=1}^{n} \ell(f_{G,w,\sigma}(x_i), y_i)$ where $\{(x_i, y_i)\}$ are training examples
- Update step: $w^{(t+1)} = w^{(t)} + p^{(t)}$
    - For gradient descent, $p^t = -\eta^{(t)} \nabla L(w^{(t)})$

Rescaling and Unbalancedness

- ReLU activation has the non-negative homogeneity property:
  - $\sigma_{\text{ReLU}}(c \cdot x) = c \cdot \sigma_{\text{ReLU}}(x)$ for $c \geq 0$

Rescaling and Unbalancedness

- ReLU activation has the non-negative homogeneity property:
  - $\sigma_{\text{ReLU}}(c \cdot x) = c \cdot \sigma_{\text{ReLU}}(x)$ for $c \geq 0$
- Define the *rescaling function* $\rho_{c,v}(w)$ such that for any $(u_1 \to u_2) \in E$,

$$\rho_{c,v}(w_{u_1 \to u_2}) = \tilde{w}_{u_1 \to u_2} = \begin{cases} c \cdot w_{u_1 \to u_2} & \text{if } u_2 = v, \\ \frac{1}{c} \cdot w_{u_1 \to u_2} & \text{if } u_1 = v, \\ w_{u_1 \to u_2} & \text{otherwise.} \end{cases}$$

Rescaling and Unbalancedness

- ReLU activation has the non-negative homogeneity property:
  - $\sigma_{\text{ReLU}}(c \cdot x) = c \cdot \sigma_{\text{ReLU}}(x)$ for $c \geq 0$
- Define the *rescaling function* $\rho_{c,v}(w)$ such that for any $(u_1 \to u_2) \in E$,

$$\rho_{c,v}(w_{u_1 \to u_2}) = \tilde{w}_{u_1 \to u_2} = \begin{cases} c \cdot w_{u_1 \to u_2} & \text{if } u_2 = v, \\ \frac{1}{c} \cdot w_{u_1 \to u_2} & \text{if } u_1 = v, \\ w_{u_1 \to u_2} & \text{otherwise.} \end{cases}$$

- $f_{G,w,\sigma_{\text{ReLU}}} = f_{G,\rho_{c,v}(w),\sigma_{\text{ReLU}}}$

Rescaling and Unbalancedness

- ReLU activation has the non-negative homogeneity property:
  - $\sigma_{\text{ReLU}}(c \cdot x) = c \cdot \sigma_{\text{ReLU}}(x)$ for $c \geq 0$
- Define the *rescaling function* $\rho_{c,v}(w)$ such that for any $(u_1 \to u_2) \in E$,

$$\rho_{c,v}(w_{u_1 \to u_2}) = \tilde{w}_{u_1 \to u_2} = \begin{cases} c \cdot w_{u_1 \to u_2} & \text{if } u_2 = v, \\ \frac{1}{c} \cdot w_{u_1 \to u_2} & \text{if } u_1 = v, \\ w_{u_1 \to u_2} & \text{otherwise.} \end{cases}$$

- $f_{G,w,\sigma_{\text{ReLU}}} = f_{G,\rho_{c,v}(w),\sigma_{\text{ReLU}}}$
- Two networks are **rescaling equivalent** if one can be transformed to another via a sequence of rescalings, denoted $w \sim \tilde{w}$.

Rescaling and Unbalancedness

- ReLU activation has the non-negative homogeneity property:
  - $\sigma_{\text{ReLU}}(c \cdot x) = c \cdot \sigma_{\text{ReLU}}(x)$ for $c \geq 0$
- Define the *rescaling function* $\rho_{c,v}(w)$ such that for any $(u_1 \to u_2) \in E$,

$$\rho_{c,v}(w_{u_1 \to u_2}) = \tilde{w}_{u_1 \to u_2} = \begin{cases} c \cdot w_{u_1 \to u_2} & \text{if } u_2 = v, \\ \frac{1}{c} \cdot w_{u_1 \to u_2} & \text{if } u_1 = v, \\ w_{u_1 \to u_2} & \text{otherwise.} \end{cases}$$

- $f_{G,w,\sigma_{\text{ReLU}}} = f_{G,\rho_{c,v}(w),\sigma_{\text{ReLU}}}$
- Two networks are **rescaling equivalent** if one can be transformed to another via a sequence of rescalings, denoted $w \sim \tilde{w}$.
- An optimization algorithm is **rescaling invariant** if the updates of rescaling equivalent networks remain rescaling equivalent.

Rescaling and Unbalancedness

- ReLU activation has the non-negative homogeneity property:
  - $\sigma_{\text{ReLU}}(c \cdot x) = c \cdot \sigma_{\text{ReLU}}(x)$ for $c \geq 0$
- Define the *rescaling function* $\rho_{c,v}(w)$ such that for any $(u_1 \to u_2) \in E$,

$$\rho_{c,v}(w_{u_1 \to u_2}) = \tilde{w}_{u_1 \to u_2} = \begin{cases} c \cdot w_{u_1 \to u_2} & \text{if } u_2 = v, \\ \frac{1}{c} \cdot w_{u_1 \to u_2} & \text{if } u_1 = v, \\ w_{u_1 \to u_2} & \text{otherwise.} \end{cases}$$

- $f_{G,w,\sigma_{\text{ReLU}}} = f_{G,\rho_{c,v}(w),\sigma_{\text{ReLU}}}$
- Two networks are **rescaling equivalent** if one can be transformed to another via a sequence of rescalings, denoted $w \sim \tilde{w}$.
- An optimization algorithm is **rescaling invariant** if the updates of rescaling equivalent networks remain rescaling equivalent.
  - i.e. $w^{(0)} \sim \tilde{w}^{(0)} \implies w^{(t)} \sim \tilde{w}^{(t)}$ after t updates
- We say that a network is **balanced** if the norm of the weights are roughly the same.

Rescaling and Unbalancedness

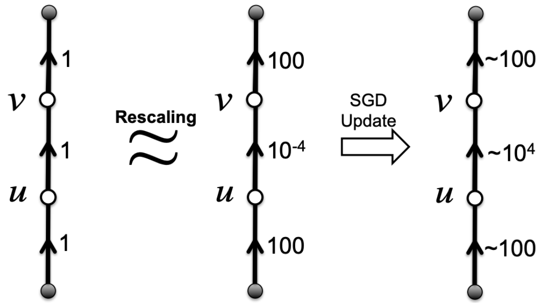
- Gradient descent is not rescaling invariant
  - Scaling down the weights will scale up the gradients.

Rescaling and Unbalancedness

- Gradient descent is not rescaling invariant
    - Scaling down the weights will scale up the gradients.
- It also performs poorly on unbalanced networks.
    - Blow up the smaller weights while keeping the larger weights almost unchanged.

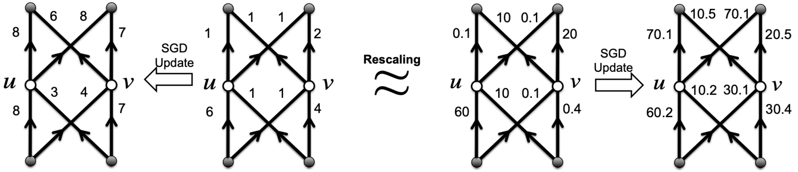
Rescaling and Unbalancedness

- Gradient descent is not rescaling invariant
  - Scaling down the weights will scale up the gradients.
- It also performs poorly on unbalanced networks.
  - Blow up the smaller weights while keeping the larger weights almost unchanged.
- Consider $x = 1$, $\eta = 1$, $\frac{\partial L}{\partial \hat{y}} = -1$,



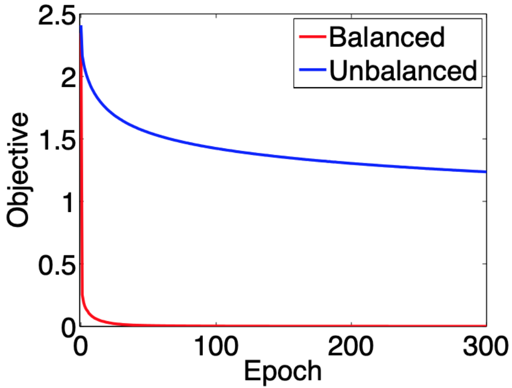
(b) Weight Explosion in an unbalanced network

Rescaling and Unbalancedness



(c) Poor updates in an unbalanced network

Rescaling and Unbalancedness



(a) Training on MNIST

Group-norm

Define the group-norm type regularizer parameterized by $p \geq 1$ and $q \leq \infty$ as,

$$\mu_{p,q}(w) = \left(\sum_{v \in V} \left(\sum_{(u \to v) \in E} \left| w_{(u \to v)} \right|^p \right)^{q/p} \right)^{1/q}$$

Group-norm

Define the group-norm type regularizer parameterized by $p \geq 1$ and $q \leq \infty$ as,

$$\mu_{p,q}(w) = \left(\sum_{v \in V} \left(\sum_{(u \to v) \in E} \left| w_{(u \to v)} \right|^p \right)^{q/p} \right)^{1/q}$$

- $p = q = 2$ gives us ℓ_2 regularization or weight decay.

Group-norm

Define the group-norm type regularizer parameterized by $p \geq 1$ and $q \leq \infty$ as,

$$\mu_{p,q}(w) = \left(\sum_{v \in V} \left(\sum_{(u \to v) \in E} \left| w_{(u \to v)} \right|^p \right)^{q/p} \right)^{1/q}$$

- $p = q = 2$ gives us ℓ_2 regularization or weight decay.
- $p = q = 1$ gives us ℓ_1 regularization.

Group-norm

Define the group-norm type regularizer parameterized by $p \geq 1$ and $q \leq \infty$ as,

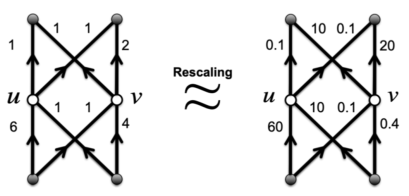
$$\mu_{p,q}(w) = \left(\sum_{v \in V} \left(\sum_{(u \to v) \in E} \left| w_{(u \to v)} \right|^p \right)^{q/p} \right)^{1/q}$$

- $p = q = 2$ gives us ℓ_2 regularization or weight decay.
- $p = q = 1$ gives us ℓ_1 regularization.
- $q = \infty$ gives us the per-unit "max-norm" regularization:
  - $\mu_{p,\infty}(w) = \sup_{v \in V} \left(\sum_{(u \to v) \in E} \left| w_{(u \to v)} \right|^p \right)^{1/p}$
  - Shown to be effective in ReLU networks.

Path-norm

$$\mu_{p,\infty}(w) = \sup_{v \in V} \left(\sum_{(u \to v) \in E} \left| w_{(u \to v)} \right|^p \right)^{1/p}$$

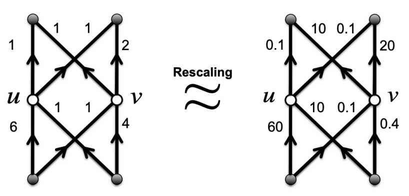
- Problem with $\mu_{p,\infty}$?
  - Not rescaling invariant – value is different for rescaling equivalent networks.



Path-norm

$$\mu_{p,\infty}(w) = \sup_{v \in V} \left(\sum_{(u \to v) \in E} \left| w_{(u \to v)} \right|^p \right)^{1/p}$$

- Problem with $\mu_{p,\infty}$?
  - Not rescaling invariant – value is different for rescaling equivalent networks.



  - Left: $\mu_{1,\infty} = 7$, Right: $\mu_{1,\infty} = 70$.
- To use it as a penalty term, we should seek the minimum $\mu_{p,\infty}$ among all rescaling equivalent networks.

Path-norm

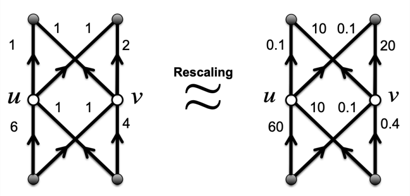
- Consider the **path vector** $\pi(w)$:
  - Number of entries $=$ number of paths from input units to output units.
  - Each entry $=$ product of weights along that path.

Path-norm

- Consider the **path vector** $\pi(w)$:
  - Number of entries = number of paths from input units to output units.
  - Each entry = product of weights along that path.
- Define the ℓ_p-**path regularizer** as

$$\|\pi(w)\|_p = \left(\sum_{v_{\text{in}}[i] \xrightarrow{e_1} v1 \xrightarrow{e_2} \dots \xrightarrow{e_d} v_{\text{out}}[j]} \left| \prod_{k=1}^{d} w_{e_k} \right|^p \right)^{1/p}.$$

Path-norm



$$\pi(w) = [6, 6, 1, 2, 1, 1, 4, 8], \qquad \|\pi(w)\|_1 = 29$$

- $\|\pi(w)\|_p$ is rescaling invariant.
- To compute it efficiently, we can use dynamic programming on the equivalent form written as nested sums.
- **Lemma [4]**: $\|\pi(w)\|_p = \min_{\tilde{w} \sim w} \left(\mu_{p,\infty}(\tilde{w})\right)^d$.

Path-SGD

Path-SGD

Steepest descent direction with respect to the path regularizer $\|\pi(w)\|_p$

$$
\begin{aligned}
w^{(t+1)} &= \arg\min_w \eta \langle \nabla L(w^{(t)}), w \rangle + \frac{1}{2} \|\pi(w) - \pi(w^{(t)})\|_p^2 \\
&= \arg\min_w \eta \langle \nabla L(w^{(t)}), w \rangle + \left(\sum_{v_{\text{in}}[i] \xrightarrow{e_1} \dots \xrightarrow{e_d} v_{\text{out}}[j]} \left(\prod_{k=1}^{d} w_{e_k} - \prod_{k=1}^{d} w_{e_k}^{(t)} \right)^p \right)^{\frac{2}{p}} \\
&= \arg\min_w J^{(t)}(w)
\end{aligned}
$$

Path-SGD

Steepest descent direction with respect to the path regularizer $\|\pi(w)\|_p$

$$
\begin{aligned}
w^{(t+1)} &= \arg\min_w \eta\langle\nabla L(w^{(t)}), w\rangle + \frac{1}{2}\|\pi(w) - \pi(w^{(t)})\|_p^2 \\
&= \arg\min_w \eta\langle\nabla L(w^{(t)}), w\rangle + \left(\sum_{v_{\text{in}}[i] \xrightarrow{e_1} \dots \xrightarrow{e_d} v_{\text{out}}[j]} \left(\prod_{k=1}^{d} w_{e_k} - \prod_{k=1}^{d} w_{e_k}^{(t)}\right)^p\right)^{\frac{2}{p}} \\
&= \arg\min_w J^{(t)}(w)
\end{aligned}
$$

Update each edge weight independently,

$$
w_e^{(t+1)} = \arg\min_{w_e} J^{(t)}(w) \qquad \text{s.t. } \forall_{e' \neq e}\ w_{e'} = w_{e'}^{(t)}
$$

Path-SGD

Take the partial derivative with respect to w_e and set it to zero gives us the update rule

$$w_e^{(t+1)} = w_e^{(t)} - \frac{\eta}{\gamma_p(w^{(t)}, e)} \frac{\partial L}{\partial w}(w^{(t)})$$

where

$$\gamma_p(w, e) = \left(\sum_{v_{\text{in}}[i] \ldots \xrightarrow{e} \ldots v_{\text{out}}[j]} \prod_{e_k \neq e} |w_{e_k}|^p \right)^{2/p} .$$

Path-normalized gradient descent or **Path-SGD** when stochastic.

Path-SGD

Take the partial derivative with respect to w_e and set it to zero gives us the update rule

$$w_e^{(t+1)} = w_e^{(t)} - \frac{\eta}{\gamma_p(w^{(t)}, e)} \frac{\partial L}{\partial w}(w^{(t)})$$

where

$$\gamma_p(w, e) = \left(\sum_{v_{\text{in}}[i] \ldots \xrightarrow{e} \ldots v_{\text{out}}[j]} \prod_{e_k \neq e} |w_{e_k}|^p \right)^{2/p} .$$

Path-normalized gradient descent or **Path-SGD** when stochastic.

- Approximate steepest descent with respect to the path norm.
- Rescaling invariant.

Path-SGD: Efficient Implementation

- The update rule requires going through all paths in the network, which is exponential in the number of layers.

Path-SGD: Efficient Implementation

- The update rule requires going through all paths in the network, which is exponential in the number of layers.

Algorithm 1 Path-SGD update rule

$\forall_{v \in V_{\text{in}}^0} \ \gamma_{\text{in}}(v) = 1$ ▷ Initialization
$\forall_{v \in V_{\text{out}}^0} \ \gamma_{\text{out}}(v) = 1$
for $i = 1$ **to** d **do**
  $\forall_{v \in V_{\text{in}}^i} \gamma_{\text{in}}(v) = \sum_{(u \to v) \in E} \gamma_{\text{in}}(u) \left| w_{(u,v)} \right|^p$
  $\forall_{v \in V_{\text{out}}^i} \gamma_{\text{out}}(v) = \sum_{(v \to u) \in E} \left| w_{(v,u)} \right|^p \gamma_{\text{out}}(u)$
end for
$\forall_{(u \to v) \in E} \ \gamma(w^{(t)}, (u, v)) = \gamma_{\text{in}}(u)^{2/p} \gamma_{\text{out}}(v)^{2/p}$
$\forall_{e \in E} w_e^{(t+1)} = w_e^{(t)} - \frac{\eta}{\gamma(w^{(t)}, e)} \frac{\partial L}{\partial w_e}(w^{(t)})$ ▷ Update Rule

Path-SGD: Efficient Implementation

- The update rule requires going through all paths in the network, which is exponential in the number of layers.

Algorithm 1 Path-SGD update rule

$\forall_{v \in V_{\text{in}}^0}\ \gamma_{\text{in}}(v) = 1$ ▷ Initialization
$\forall_{v \in V_{\text{out}}^0}\ \gamma_{\text{out}}(v) = 1$
for $i = 1$ **to** d **do**
  $\forall_{v \in V_{\text{in}}^i} \gamma_{\text{in}}(v) = \sum_{(u \to v) \in E} \gamma_{\text{in}}(u) \left| w_{(u,v)} \right|^p$
  $\forall_{v \in V_{\text{out}}^i} \gamma_{\text{out}}(v) = \sum_{(v \to u) \in E} \left| w_{(v,u)} \right|^p \gamma_{\text{out}}(u)$
end for
$\forall_{(u \to v) \in E}\ \gamma(w^{(t)}, (u,v)) = \gamma_{\text{in}}(u)^{2/p} \gamma_{\text{out}}(v)^{2/p}$
$\forall_{e \in E} w_e^{(t+1)} = w_e^{(t)} - \frac{\eta}{\gamma(w^{(t)}, e)} \frac{\partial L}{\partial w_e}(w^{(t)})$ ▷ Update Rule

- One update can now be computed in one forward-backward pass on a minibatch.

Path-SGD: Experiments

Setup:

- Compare ℓ_2-Path-SGD against (constant step size) SGD and AdaGrad.

Path-SGD: Experiments

Setup:

- Compare ℓ_2-Path-SGD against (constant step size) SGD and AdaGrad.
- Feedforward networks with 2 hidden layers (4000 hidden units).
    - Both dropout ($p = 0.5$) and no dropout
    - Balanced and unbalanced initializations

Path-SGD: Experiments

Setup:

- Compare ℓ_2-Path-SGD against (constant step size) SGD and AdaGrad.
- Feedforward networks with 2 hidden layers (4000 hidden units).
  - Both dropout ($p = 0.5$) and no dropout
  - Balanced and unbalanced initializations
- Batch size $= 100$

Path-SGD: Experiments

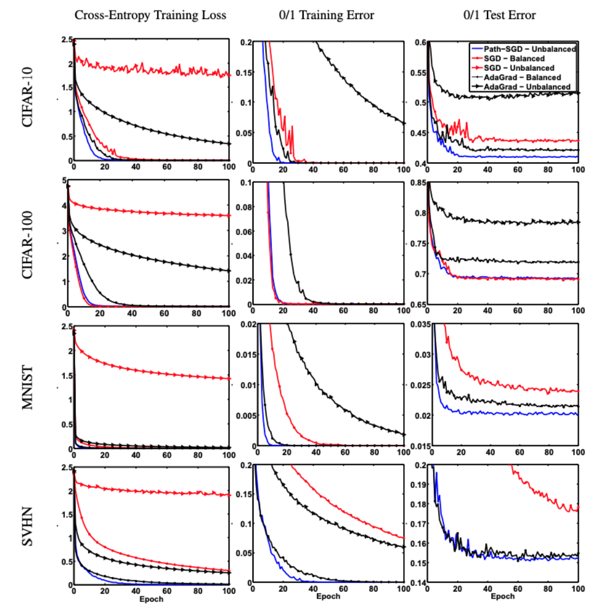
Setup:

- Compare ℓ_2-Path-SGD against (constant step size) SGD and AdaGrad.
- Feedforward networks with 2 hidden layers (4000 hidden units).
  - Both dropout ($p = 0.5$) and no dropout
  - Balanced and unbalanced initializations
- Batch size = 100

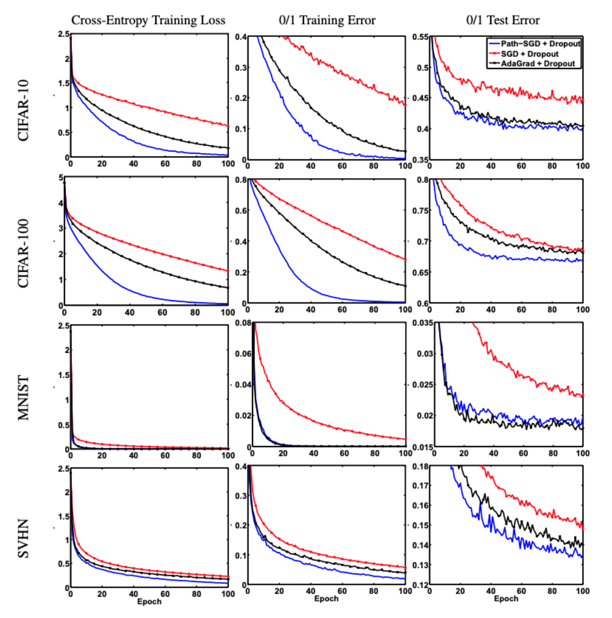
Table 1: General information on datasets used in the experiments on feedforward networks.

Data Set	Dimensionality	Classes	Training Set	Test Set
CIFAR-10	3072 (32×32 color)	10	50000	10000
CIFAR-100	3072 (32×32 color)	100	50000	10000
MNIST	784 (28×28 grayscale)	10	60000	10000
SVHN	3072 (32×32 color)	10	73257	26032

Experiment Results: without dropout



Experiment Results: with dropout



Conclusion

Conclusion

Summary:

- Implicit regularization from optimization plays a role in the generalization of feedforward neural networks.
- Proposed an alternative to SGD that uses a different geomtry (path-norm) that is rescaling invariant.
- Path-SGD seems to work well compared to constant step size SGD and AdaGrad.

Conclusion

Future directions:

- Combine Path-SGD with AdaGrad?
- Other rescaling invariant metric/geometry?
- Considerations for other activation functions that don't necessarily have non-negative homogeneity?

References i

B. Neyshabur, R. R. Salakhutdinov, and N. Srebro.
Path-SGD: Path-normalized optimization in deep neural networks.
In *Advances in Neural Information Processing Systems*, pages 2422–2430, 2015.

B. Neyshabur, R. Tomioka, R. Salakhutdinov, and N. Srebro.
Geometry of Optimization and Implicit Regularization in Deep Learning.
CoRR, 2017.

References ii

B. Neyshabur, R. Tomioka, and N. Srebro.
In Search of the Real Inductive Bias: On the Role of Implicit Regularization in Deep Learning.
In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*, 2015.

B. Neyshabur, R. Tomioka, and N. Srebro.
Norm-based capacity control in neural networks.
In *Conference on Learning Theory*, pages 1376–1401, 2015.