Path-SGD: Path-Normalized Optimization in Deep Neural Networks

Ramchandran Muthukumar

April 9, 2019

Quiz

- Which of these algorithms perform better for balanced network as opposed to unbalanced network
  1. Gradient descent
  2. Stochastic gradient descent
  3. AdaGrad
  4. Path-SGD
- Modifying weights of RELU Feedforward Neural Network _____ changes the predictions.
  1. always
  2. sometimes
  3. never

Outline

Geometry

Path-SGD: Approximate Path-Regularized Steepest Descent

Experiments

Conclusion

Motivation - Link Between Geometry and Optimization Algorithms

- Geometry $=$ measure of distance, norm or divergence.
- Some hoices are ℓ_1 and ℓ_2 norms.
- Optimization algorithms are tied to geometry inherently.
- Gradient descent $=$ steepest descent w.r.t. ℓ_2 norm.
- Coordinate descent $=$ steepest descent w.r.t. ℓ_1 norm.

Impact of geometry on learning

- The choice of geometry $\Rightarrow$ a choice of regularization on weights.
- Ideal properties of a geometry -
  1. faster optimization
  2. better implicit regularization
- Is ℓ_2 geometry the best choice for learning deep neural networks?

Notations

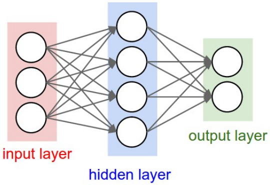
Definition (RELU Feed-forward Neural Network)

A feed-forward neural network computes a function $f : \mathbb{R}^D \to \mathbb{R}^C$.
It is defined by a directed acyclic graph $G(V, E)$ with

1. D input nodes $v_{in}[1], \ldots, v_{in}[D] \in V$
2. C output nodes $v_{out}[1], \ldots, v_{out}[C] \in V$.

For two vertices $u_1, u_2 \in V$, the weight on the corresponding edge $u_1 \to u_2$ is denoted as $w_{(u_1 \to u_2)} \in \mathbb{R}$.
The network is parameterized by the collection of edge weights $\mathbf{w}$.
The activation function $\sigma_{\text{RELU}}(x) = \max\{0, x\}$ acts on the internal nodes (hidden units).
We denote the function computed by this network as $f_{G,\mathbf{w},\sigma_{\text{RELU}}}$.

RELU Networks are Rescaling Invariant



- Consider any hidden unit v.
- Scale down the incoming edges to v by a factor c
- Scale up the outgoing edges from v by the same factor c.
- The resulting network still computes the same function.
- Therefore RELU Networks are invariant to such a rescaling.

What kind of activation functions would be similarly rescaling invariant?

Mathematical description of rescaling

Lemma

RELU activation is non-negative homogeneous. For any scalar $c \geq 0$ and any $x \in \mathbb{R}$, $\sigma_{\text{RELU}}(cx) = c\sigma_{\text{RELU}}(x)$.

Definition (Rescaling Function)

For any node $v \in V$, a scaling factor $c > 0$ and the weights of the network $\mathbf{w}$, We define the *rescaling function* $\rho_{c,v}(\cdot) : w \mapsto \tilde{w}$ such that for all edges $(u_1 \to u_2) \in E$,

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

A network rescaled with $\rho_{c,v}(\cdot)$ computes the same function, i.e.

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

.

Rescaling Invariance

Definition (Rescaling Equivalent Networks)

Two RELU feedforward neural networks defined on the graph $G(V, E)$ with weights w and $\tilde{w}$ are *rescaling equivalent* if and only if one of them can be transformed to another by applying a sequence of rescaling functions $\rho_{c,v}$.
We denote this property by $w \sim \tilde{w}$

Rescaling Invariance

Definition (Rescaling Invariant Optimization Method)

An optimization method is *rescaling invariant* if its updates on rescaling equivalent networks are rescaling equivalent.
Let the initial weight matrices from rescaling equivalent networks are $\tilde{w}^{(0)} \sim w^{(0)}$.
For all iterations t under the optimization method, the weight matrices remain rescaling equivalent and we have $\tilde{w}^{(t)} \sim w^{(t)}$.

Rescaling and Unbalanced Networks

Definition (Balanced Networks)

We say that a network is *balanced* if the norm of incoming weights to different units are roughly the same or within a small range.

Balanced networks can be rescaled to equivalent unbalanced networks.

Gradient descent and SGD perform very poorly on "unbalanced" networks.

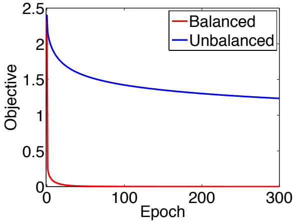
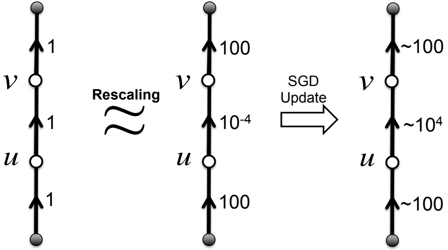


Figure: Comparison of SGD on equivalent balanced and unbalanced networks for training on MNIST

Issues with Unbalanced Networks

- Gradient descent is **not** rescaling invariant.
- Scaling down the weights of an edge will scale up the gradient.
- Larger weights remain almost unchanged
- Smaller weights blow up



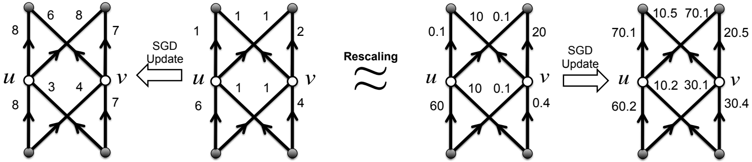
Can you think of a way to fix this?

Possible Fix?

- We can rescale the weights after each update to a more balanced network.

Additional Issues with Unbalanced Network

- Very different relative changes in weights compared to a balanced network.



- After just a single update two rescaling equivalent networks could end up computing very different functions

Rescaling Invariant

- Predictions (or function computed) by RELU Feed-forward neural networks are rescaling invariant.
- We want a geometry and a corresponding optimization method that is also rescaling invariant
- This paper considers a rescaling invariant geometry inspired by max-norm regularization.

Scale measures for deep networks

Definition (Group-Norm Regularizer)

The generic group-norm regularizer parametrized by $1 \leq p, q \leq \infty$ is defined as

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

- When $p = q = 1$ this is ℓ_1 regularization.
- When $p = q = 2$ this is weight decay (most commonly used).

Definition (Max-norm regularizer)

The max-norm regularizer is defined as

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

This is equivalent to setting $q = \infty$ in the per-unit regularizer.

Issues with Max-norm regularizer

- Max-norm regularization is extreme because the value of regularizer corresponds to the highest value among all nodes.
- But it has empirically been shown to be effective for RELU networks.
- This could be because of RELU networks can be rebalanced such that all hidden units have the similar norm.

Issues with Max-norm regularizer

- The max-norm regularizer $\mu_{p,\infty}$ is also **not** rescaling invariant.
- We want a rescaling-invariant regularizer.
- We can instead look for the minimum value of the max-norm regularizer among all rescaling equivalent networks.

Paths in Feed-forward neural network

Definition (Path vector)

Consider a RELU feed-forward network parameterized by the graph $G(V, E)$ and edge weights $\mathbf{w}$.
Let $P_{i,j} = \{e_1, e_2, \ldots, e_d\}$ be a path from the i-th input unit $v_{in}[i]$ to the j-th output unit $v_{out}[j]$. Let $|P|$ be the number of such unique paths.
The *path vector* $\pi(\mathbf{w}) \in \mathbb{R}^{|P|}$ is defined as the vector where each co-ordinate is the product of all weights along a path.

$$\pi_{P_{i,j}}(\mathbf{w}) = \prod_{k=1}^{d} \mathbf{w}_{e_k}$$

ℓ_p-path regularizer

Definition (ℓ_p-path regularizer)

The ℓ_p-path regularizer $\phi_p(\mathbf{w})$ is defined as the ℓ_p norm of $\pi(\mathbf{w})$:

$$\phi_p(\mathbf{w}) = \|\pi(\mathbf{w})\|_p = \left(\sum_{v_{in}[i] \xrightarrow{e_1} v_1 \xrightarrow{e_2} v_2 \ldots \xrightarrow{e_d} v_{out}[j]} \left| \prod_{k=1}^{d} \mathbf{w}_{e_k} \right|^p \right)^{1/p} \tag{2}$$

What would be the complexity of computing $\phi_p(\mathbf{w})$?

Computation of ℓ_p-path regularizer

- The ℓ_p-path regularizer can be computed efficiently in a **single forward step** by the following recursive definition

$$\phi_{p,v}(\mathbf{w}) = \sum_{(u \to v) \in E} \phi_{p,u}(\mathbf{w}) \cdot (\mathbf{w}_{u \to v})^p$$

$$\phi_p(\mathbf{w}) = \sum_{u \in V_{out}} \phi_{p,u}(\mathbf{w})$$

Link between ℓ_p-path regularizer and max-norm regularizer

Lemma

The ℓ_p-path regularizer and per-unit regularizer satisfy the following relation :

$$\phi_p(\mathbf{w}) = \min_{\tilde{\mathbf{w}} \sim \mathbf{w}} \left(\mu_{p,\infty}(\tilde{\mathbf{w}}) \right)^d$$

where d is the depth of the neural network.

- Hence the ℓ_p path-regularizer ϕ_p is invariant to rescaling. For any $\tilde{\mathbf{w}} \sim \mathbf{w}$, $\phi_p(\tilde{\mathbf{w}}) = \phi_p(\mathbf{w})$

Outline

Geometry

Path-SGD: Approximate Path-Regularized Steepest Descent

Experiments

Conclusion

Steepest Descent w.r.t. ℓ_p-path regularizer

- The steepest descent update is given by the solution to the optimization problem

$$\begin{aligned}\mathbf{w}^{(t+1)} &:= \arg\min_{\mathbf{w}} \quad \eta \nabla L(\mathbf{w}^{(t)})(\mathbf{w} - \mathbf{w}^{(t)}) + \frac{1}{2}\left\|\pi(\mathbf{w}) - \pi(\mathbf{w}^{(t)})\right\|_p^2 \\ &= \arg\min_{\mathbf{w}} \quad J^{(t)}(\mathbf{w})\end{aligned}$$

- Here the path regularizer term is

$$\left\|\pi(\mathbf{w}) - \pi(\mathbf{w}^{(t)})\right\|_p^2 = \left(\sum_{v_{in}[i] \xrightarrow{e_1} v_1 \xrightarrow{e_2} v_2 \dots \xrightarrow{e_d} v_{out}[j]} \left(\prod_{k=1}^{d} \mathbf{w}_{e_k} - \prod_{k=1}^{d} \mathbf{w}_{e_k}^{(t)}\right)^p\right)^{2/p}$$

Steepest descent w.r.t. ℓ_p-path regularizer

- The steepest descent step is hard to calculate exactly.
- Instead, we will update each coordinate $\mathbf{w}_e$ independently (and synchronously)

$$\mathbf{w}_e^{(t+1)} = \arg\min_{\mathbf{w}_e} \; J^{(t)}(\mathbf{w}) \qquad \text{s.t. } \forall_{e' \neq e} \;\; \mathbf{w}_{e'} = \mathbf{w}_{e'}^{(t)} \tag{3}$$

- Taking the partial derivative with respect to $\mathbf{w}_e$ and setting it to zero we obtain:

$$0 = \eta \frac{\partial L}{\partial \mathbf{w}_e}(\mathbf{w}^{(t)}) - \left(\mathbf{w}_e - \mathbf{w}_e^{(t)}\right) \left(\sum_{v_{\text{in}}[i] \cdots \xrightarrow{e} \ldots v_{\text{out}}[j]} \; \prod_{e_k \neq e} \left| {\mathbf{w}_e^{(t)}}^p \right| \right)^{2/p}$$

- $v_{\text{in}}[i] \cdots \xrightarrow{e} \ldots v_{\text{out}}[j]$ denotes the paths from the i-th input unit to the j-th output unit that includes edge e.

Path-SGD

- Solving for $\mathbf{w}_e$ gives the update rule:

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

- Here $\gamma_p(\mathbf{w}, e)$ is defined as

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

- The optimization algorithms with the above update rule is called **path-normalized gradient descent**.
- In stochastic settings, we refer to it as **Path-SGD** .

Path-SGD

Theorem

Path-SGD is rescaling invariant, i.e. for any $c > 0$ and $v \in E$,

$$\tilde{\mathbf{w}}^{(t)} = \rho_{c,v}(\mathbf{w}^{(t)}) \quad \Rightarrow \quad \tilde{\mathbf{w}}^{(t+1)} = \rho_{c,v}(\mathbf{w}^{(t+1)})$$

- If edge e is neither incoming nor outgoing edge of the node v, then $\tilde{\mathbf{w}}(e) = \mathbf{w}(e)$ and $\tilde{\mathbf{w}}_e^{(t+1)} = \mathbf{w}_e^{(t+1)}$
- If edge e is an incoming edge to v, then $\tilde{\mathbf{w}}^{(t)}(e) = c\mathbf{w}^{(t)}(e)$.
- In this case, outgoing edges of v are divided by c. Therefore

$$\gamma_p(\tilde{\mathbf{w}}^{(t)}, e) = \frac{\gamma_p(\mathbf{w}^{(t)}, e)}{c^2}, \quad \frac{\partial L}{\partial \mathbf{w}_e}(\tilde{\mathbf{w}}^{(t)}) = \frac{\partial L}{c \partial \mathbf{w}_e}(\mathbf{w}^{(t)})$$

Path-SGD

- The Path-SGD update on $\tilde{\mathbf{w}}$ is

$$\begin{aligned}\tilde{\mathbf{w}}_e^{(t+1)} &= c\mathbf{w}_e^{(t)} - \frac{c^2\eta}{\gamma_p(\mathbf{w}^{(t)}, e)} \cdot \frac{\partial L}{c\partial \mathbf{w}_e}(\mathbf{w}^{(t)}) \\ &= c\left(\mathbf{w}^{(t)} - \frac{\eta}{\gamma_p(\mathbf{w}^{(t)}, e)} \cdot \frac{\partial L}{\partial \mathbf{w}_e}(\mathbf{w}^{(t)})\right) = c\mathbf{w}_e^{(t+1)}.\end{aligned}$$

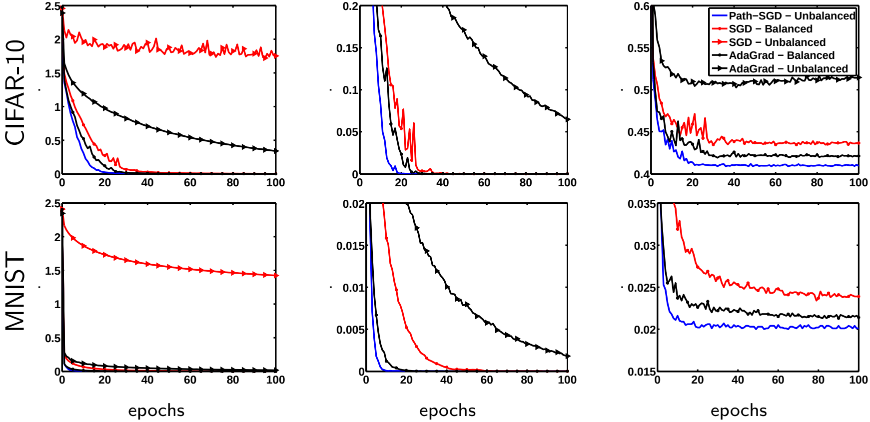
- A similar argument follows when e is an outgoing edge of node v.
- Therefore, Path-SGD is rescaling invariant.

Outline

Computational Setup

- Feed-forward networks with 2 layers of 4000 hidden units each.
- Minibatches of size 100, step size $10^{-\alpha}$ for integer $\alpha < 10$.
- Initialization for edge weights
  1. Balanced : weights drawn i.i.d from Gaussain Distribution.
  2. Unbalanced : Choose balanced weights and rescale 2000 hidden units by scale factor c.
- Dropout with 0.5 probability of retaining each unit.
- Dropout experiments only on balanced networks.

Experiments without Dropout : CIFAR-10 and MNIST Dataset

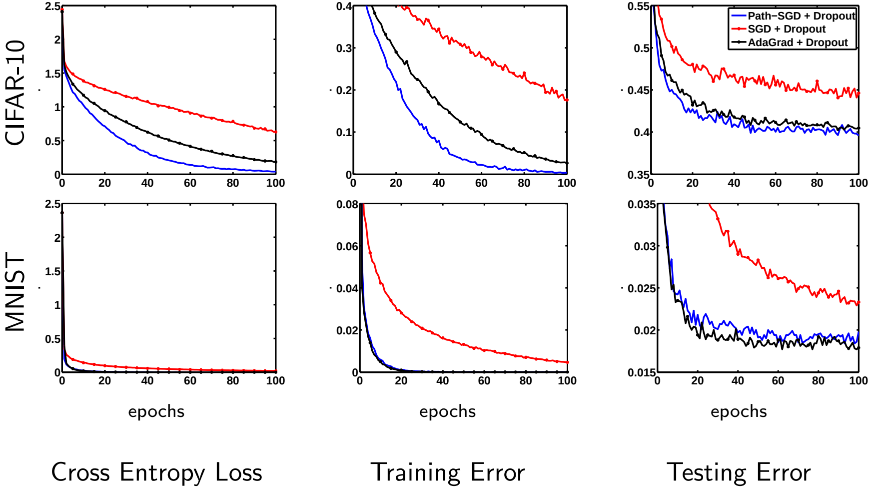


Cross Entropy Loss   Training Error   Testing Error

Discussion for no-dropout case

- Learning curves of Path-SGD on balanced and unbalanced initializations were nearly identical.
- Unbalanced initialization drastically affects SGD and AdaGrad.
- Path-SGD converges faster than SGD and AdaGrad.
- Path-SGD also has lesser testing error (better implicit regularization?).

Experiments with Dropout: CIFAR-10 and MNIST Dataset



Discussion for dropout case

- Path-SGD is atleast as fast as SGD and AdaGrad. (Much faster for CIFAR-10).
- Path-SGD again has lesser testing error.
- Thus empirically, Path-SGD achieves the same accuracy faster and generalizes better.

Outline

Conclusion : Effects of Geometry

- Choosing alternative geometry can result in faster optimization and better generalization.
- Rescaling Invariance is an appropriate property for RELU networks.
- Path-SGD empirically performs better even for unbalanced networks.
- Combining Path-SGD with momentum or other heuresitcs might improve results.