Laplacian Smoothing Gradient Descent

Stan Osher's Group [1]
Dept of Math, UCLA

[1] Thanks to professor Andrea Bertozzi's group for helping us.

Modern Machine Learning - Nonconvex Optimization

Many machine learning models are nonconvex:

- K-means clustering
- Gaussian Mixture Model
- Deep Learning
- ...

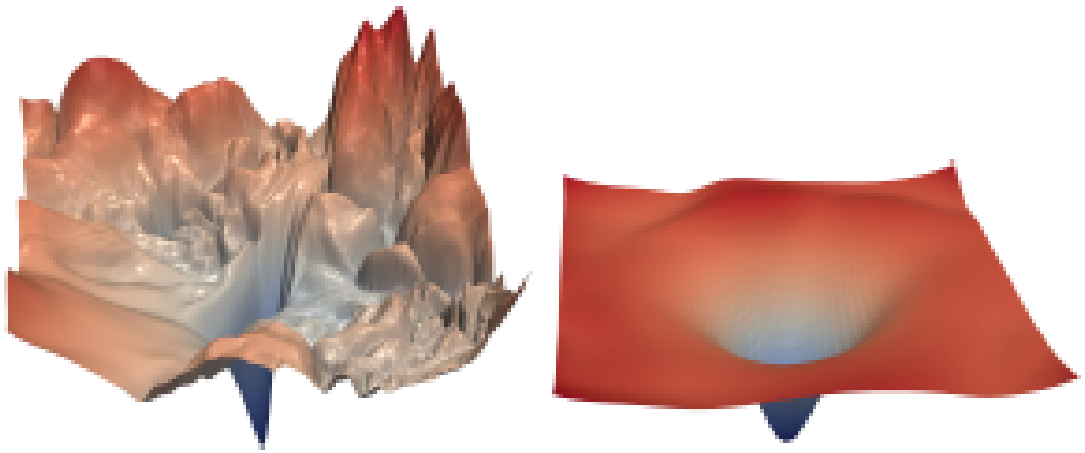


Figure: Landscape of the loss functions for ResNet-56. Left: without skip connection; right: with skip connection.

Li et al, Arxiv1712.09913, 2017.

Modern Machine Learning - Nonconvex Optimization

Given a model

$$\mathbf{y} = f(\mathbf{x}, \mathbf{w}),$$

and training data $\{X, Y\}$. We want to train the model by minimizing the empirical risk function

$$\mathcal{L}(X, Y, \mathbf{w}) \doteq \mathcal{L}(\mathbf{w}).$$

$\mathbf{w}$ is usually trained by stochastic gradient descent

$$\mathbf{w}^{k+1} = \mathbf{w}^k - \gamma \frac{\partial \mathcal{L}(\tilde{X}, \tilde{Y}, \mathbf{w}^k)}{\partial \mathbf{w}},$$

where γ is learning rate, $\{\tilde{X}, \tilde{Y}\}$ is a random mini-batch of training data $\{X, Y\}$

Gradient Descent - Good Landscape

Gradient Descent - Bad Landscape

SGD's Magic

Flat minima generalize better than sharp minima.

Hochreiter, et al, Neural Comput, 1997
Dinh, et al, arXiv:1703.04933, 2017
Keskar, et al, arXiv:1609.04836, 2017

Theory of uniform stability.

Bousquet, et al, JMLR, 2002
Hardt, et al, ICML, 2016

How to Circumvent Sharp Minima?

Problem: Find the flat minima of the empirical loss function $\mathcal{L}(\mathbf{w})$, which is highly nonconvex.

Consider the HJ-PDE with $\mathcal{L}(\mathbf{w})$ as its initial condition

$$\begin{cases} u_t + \frac{1}{2}\left\langle \nabla_{\mathbf{w}} u, (\mathcal{I} - \sigma\Delta)^{-1} \nabla_{\mathbf{w}} u \right\rangle = 0, & (\mathbf{w}, t) \in \Omega \times [0, \infty) \\ u(\mathbf{w}, 0) = \mathcal{L}(\mathbf{w}), & \mathbf{w} \in \Omega \end{cases} \tag{1}$$

where $\mathcal{I}$ and Δ are the identity and Laplacian operators, respectively. σ is a nonnegative parameter.

By Hopf-Lax formula, the unique viscosity solution is

$$u(\mathbf{w}, t) = \inf_{\mathbf{v}} \left\{ \mathcal{L}(\mathbf{v}) + \frac{1}{2t} \left\langle \mathbf{v} - \mathbf{w}, (\mathcal{I} - \sigma\Delta)(\mathbf{v} - \mathbf{w}) \right\rangle \right\}.$$

Convexification

The viscosity solution $u_t(\mathbf{w}) := u(\mathbf{w}, t)$ convexifies $\mathbf{w}$ by bringing down the local maxima while retaining the wide minima. As illustrated in the following figure.

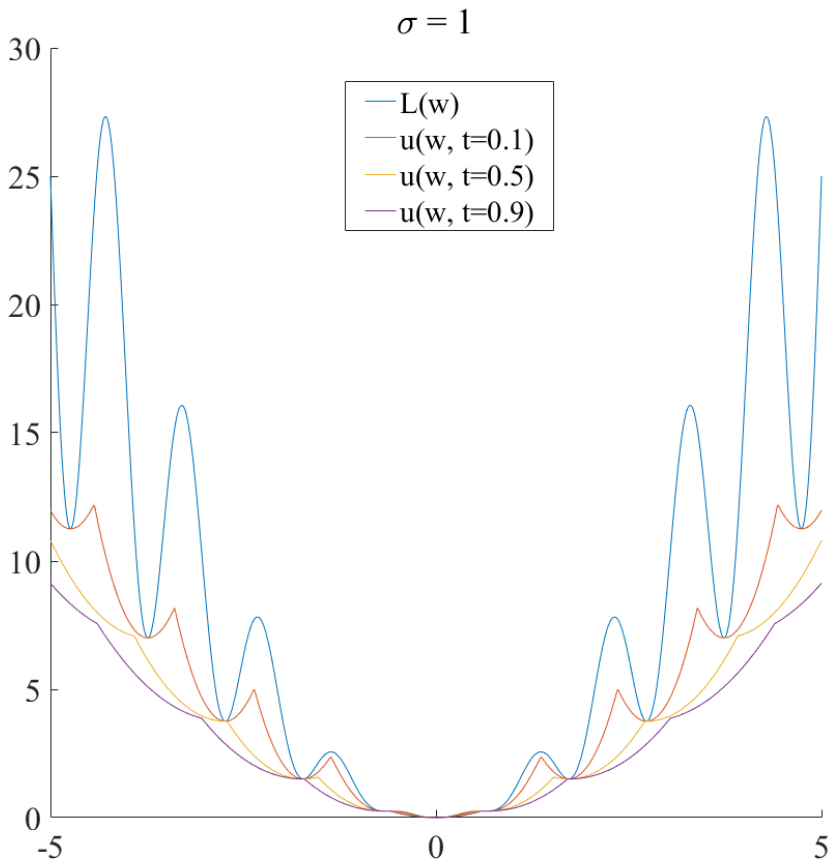


Figure: Convexification of $\mathcal{L}(\mathbf{w}) = \|\mathbf{w}\|^2 (1 + 0.5 \sin(2\pi \|\mathbf{w}\|))$ by Laplacian smoothing (viscosity solutions of HJ-PDE Eq.(1)). The plot shows the cross section of a 5-D problem with $\sigma = 1$ and different t values.

Laplacian Smoothing Gradient Descent

For a convex loss function $\mathcal{L}(\mathbf{w})$, it can be shown that standard GD on $u(\mathbf{w}, t)$ is equivalent to the following Laplacian smoothing implicit GD on $\mathcal{L}(\mathbf{w})$, i.e.,

$$\mathbf{w}^{k+1} = \mathbf{w}^k - \gamma(\mathcal{I} - \sigma\Delta)^{-1}\nabla u_t(\mathbf{w}^k)$$

is equivalent to

$$\mathbf{w}^{k+1} = \mathbf{w}^k - \gamma(\mathcal{I} - \sigma\Delta)^{-1}\nabla\mathcal{L}(\mathbf{w}^{k+1}),$$

where γ is learning rate.

Laplacian Smoothing Gradient Descent

How to implement the implicit update?

Chaudhari et al, arXiv:1704.04932, 2017

We relax implicit scheme to explicit and apply to nonconvex optimization.

$$\mathbf{w}^{k+1} = \mathbf{w}^k - \gamma(\mathcal{I} - \sigma\Delta)^{-1}\nabla\mathcal{L}(\mathbf{w}^k) \doteq \mathbf{w}^k - \gamma\nabla^\sigma\mathcal{L}(\mathbf{w}^k).$$

Laplacian Smoothing Gradient Descent (LS-GD)

An Intuitive Explanation - Not the Whole Story

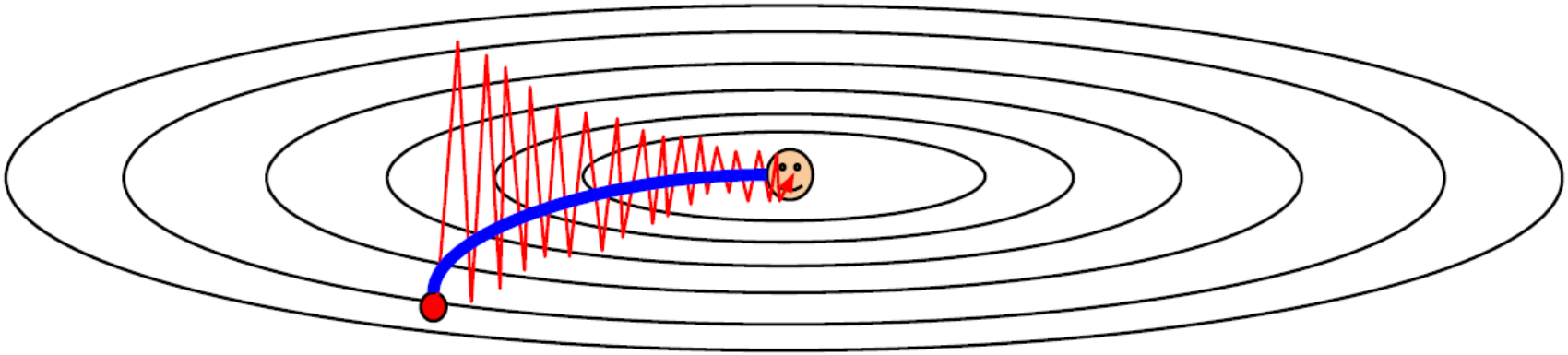


Figure: Preconditioning the gradient to avoid slow progress in shallow directions.

PyTorch and Tensorflow Implementation

Discrete form of ∇^σ with periodic boundary condition:

$$\textbf{inv}\left(\begin{bmatrix} 1+2\sigma & -\sigma & 0 & \dots & 0 & -\sigma \\ -\sigma & 1+2\sigma & -\sigma & \dots & 0 & 0 \\ 0 & -\sigma & 1+2\sigma & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ -\sigma & 0 & 0 & \dots & -\sigma & 1+2\sigma \end{bmatrix}\right)$$

- Thomas algorithm + Sherman Morrison formula.
- **FFT:** Given a vector $\mathbf{a}$, a smoothed vector $\mathbf{b}$ can be obtained by solving $(\mathcal{I} - \sigma\Delta)^{-1}\mathbf{a} = \mathbf{b}$. This is equivalent to $\mathbf{a} = \mathbf{b} - \sigma\Delta \cdot \mathbf{b}$, or $\mathbf{a} = \mathbf{b} - \sigma\mathbf{v} * \mathbf{b}$, where $\mathbf{v} = (-2, 1, 0, \cdots, 0, 1)$ and $*$ is the convolutional operator. Hence, we have

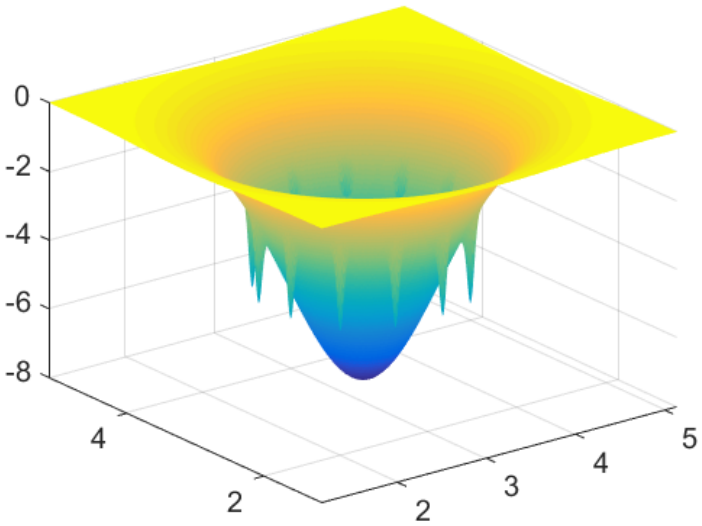
$$\mathbf{b} = \text{ifft}\left(\frac{\text{fft}(\mathbf{a})}{1 - \sigma \cdot \text{fft}(\mathbf{v})}\right),$$

Circumvent Sharp Minima - An Illustration

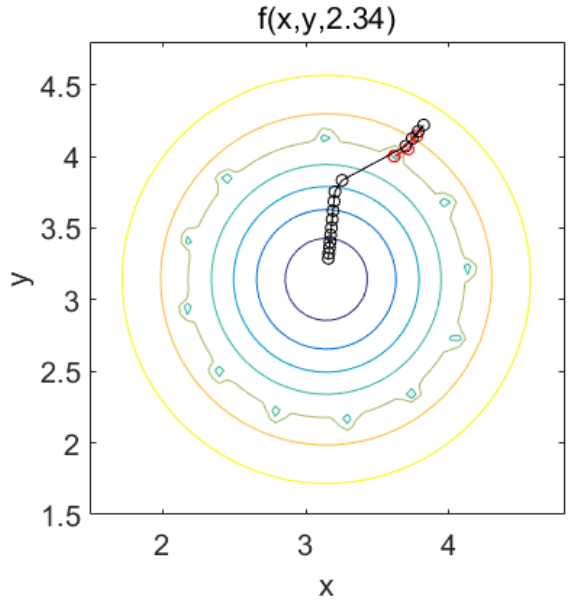
Consider

$$f(x,y,z) = -4e^{-\left((x-\pi)^2+(y-\pi)^2+(z-\pi)^2\right)} - 4\sum_{i/2}\cos(x)\cos(y)e^{-\beta\left((x-r\sin(i/2)-\pi)^2+(y-r\cos(i/2)-\pi)^2\right)} \quad (2)$$

summation over $\{i \in \mathbb{N} | 0 \leq i < 4\pi\}$, $r = 1$, $\beta = \frac{1}{\sqrt{500}}$.



(a)



(b)

Figure: Demo of gradient descent with raw and Laplacian smoothed gradients. Panel (a) depicts a slice of the function given by Eq.(2); panel (b) shows the paths by using two different gradients, where red and black dots are the points on the paths with raw and smoothed gradients, respectively. The learn rate in the gradient descent is set to be 0.02 and the smooth parameter $\sigma = 1.0$.

Comparison with Other Optimization Algorithms

Saddle point problem

$$f(x, y) = 3(x^2 - y^2).$$

Rosenbrock function

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2.$$

Comparison with Other Optimization Algorithms - Escape Saddle Point

(LS-SGD escape saddle point)

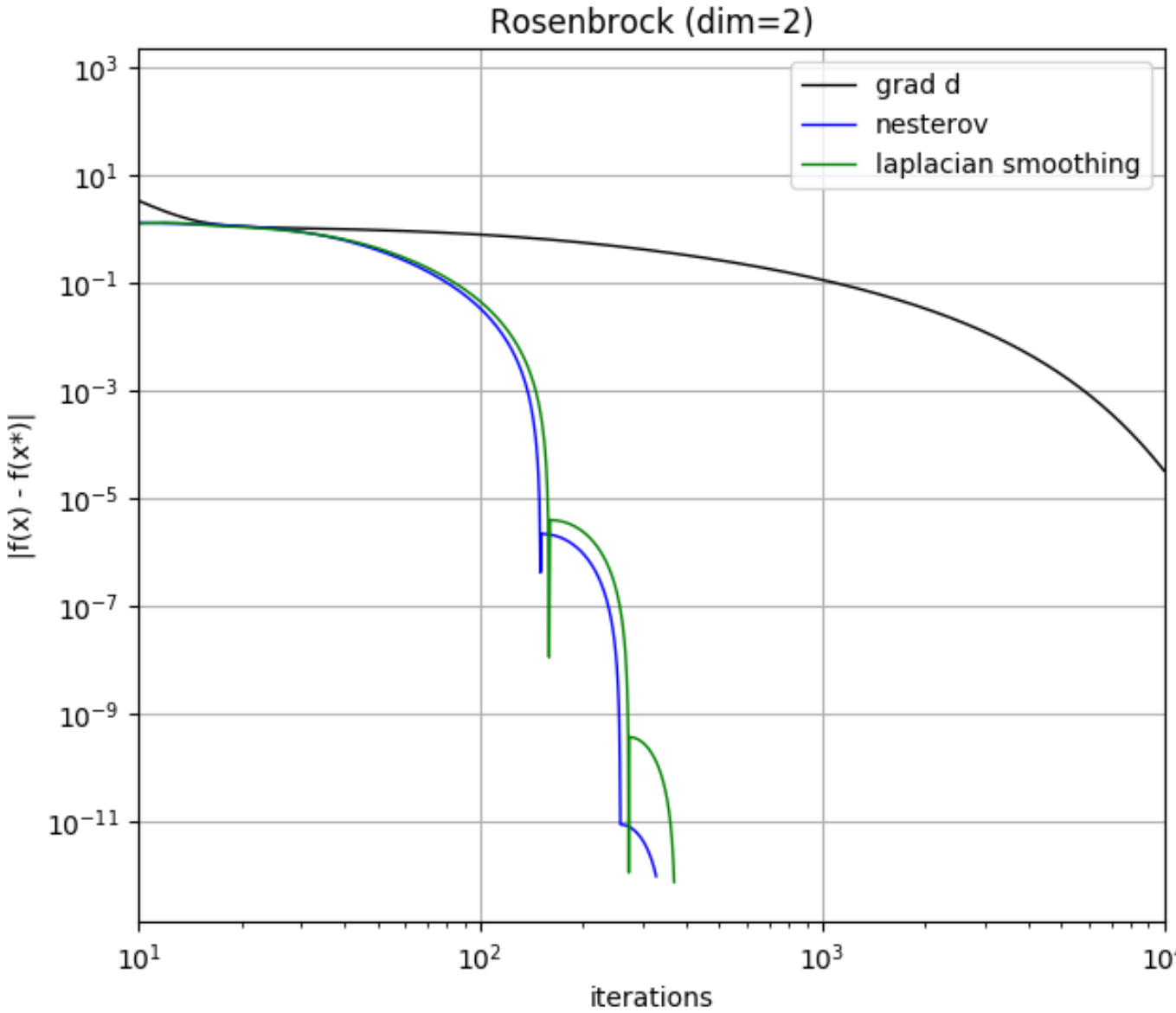
Comparison with Other Optimization Algorithms - Faster Convergence

(LS-SGD converge faster)

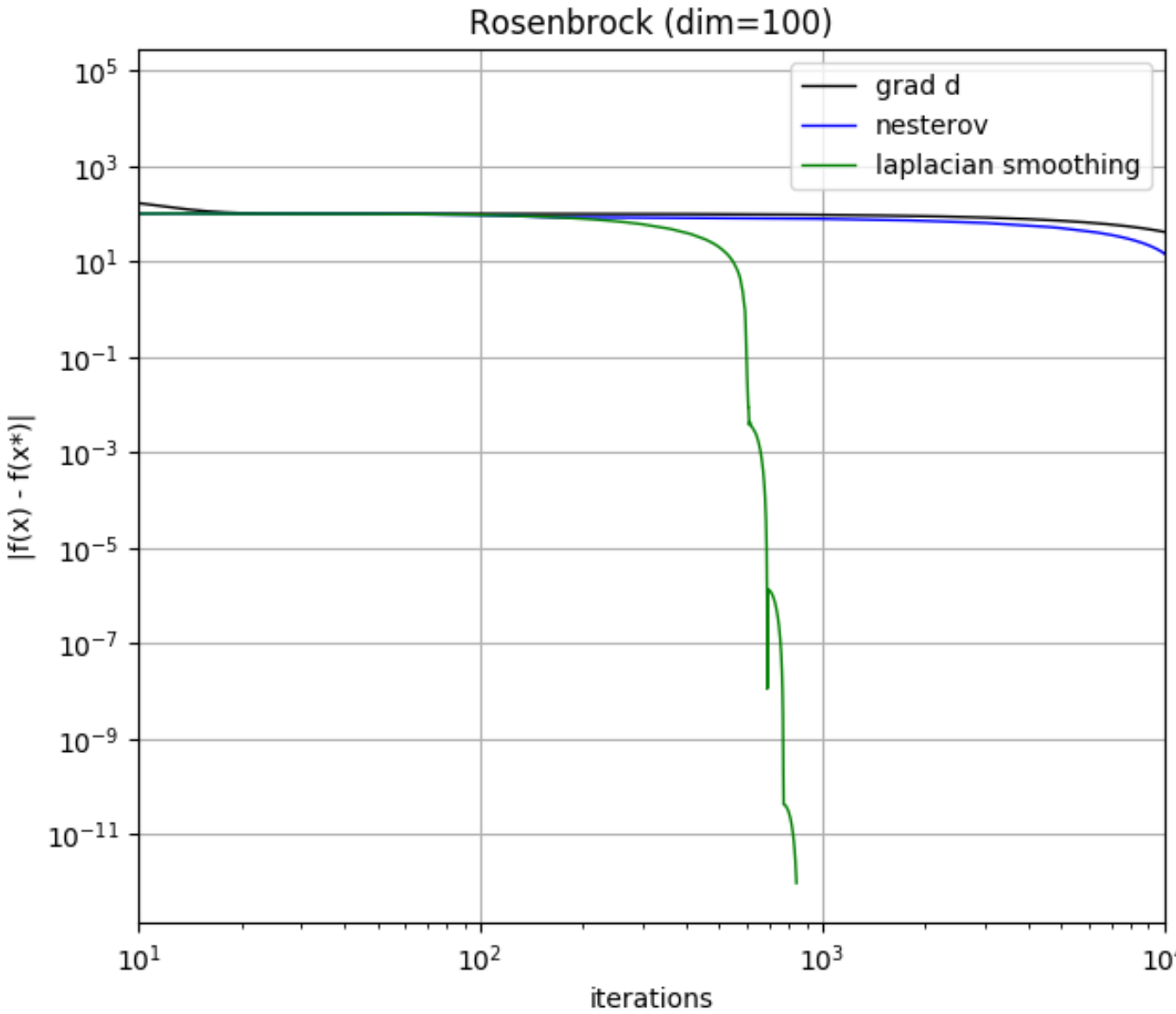
Comparison with Other Optimization Algorithms - Larger Learning Rate

(LS-SGD converges with large learning rate)

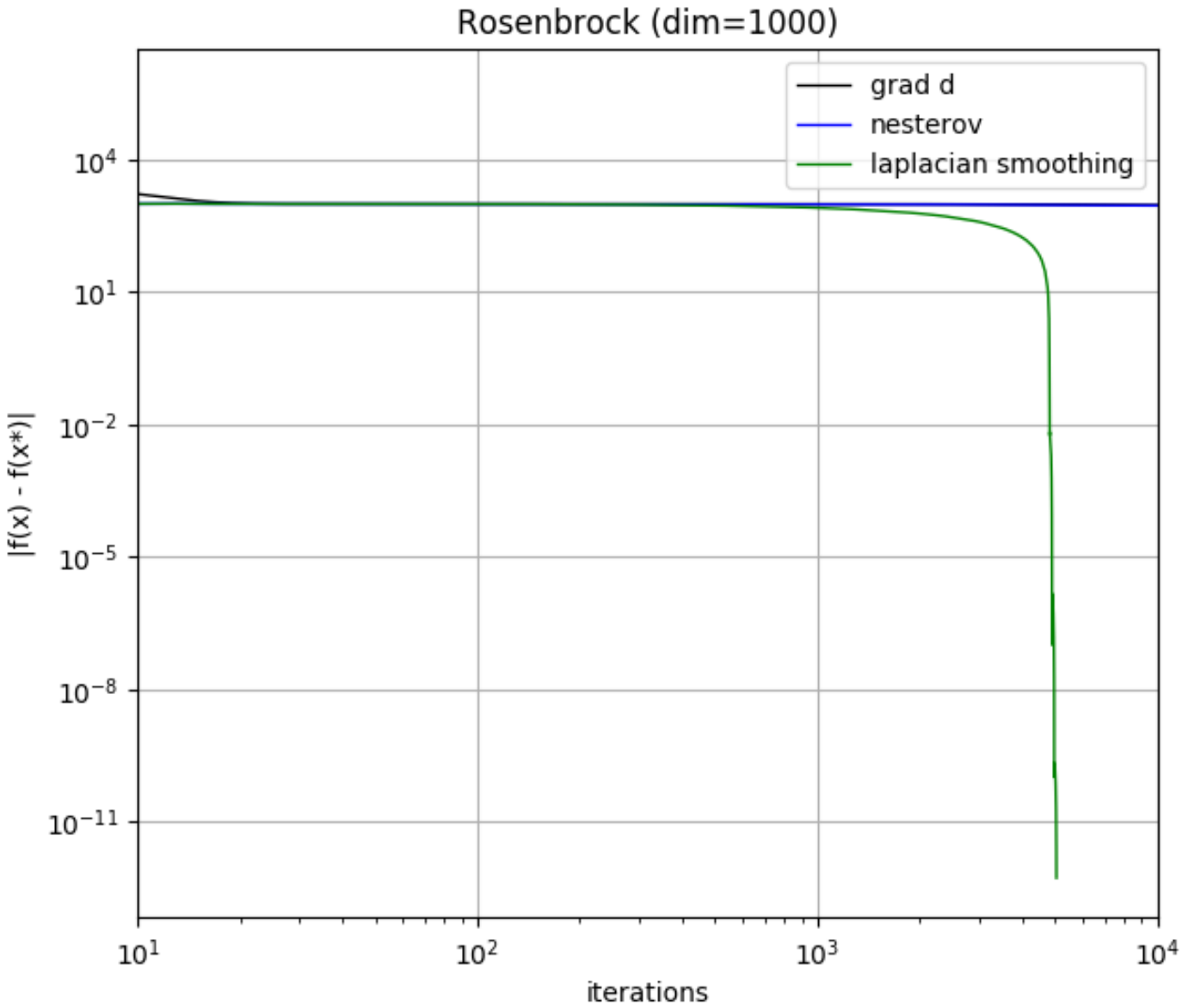
Faster Convergence for High Dimensional Rosenbrock Function -2D



Faster Convergence for High Dimensional Rosenbrock Function -100D



Faster Convergence for High Dimensional Rosenbrock Function -1000D



Softmax Regression

For a given instance $\mathbf{x}$, the probability belonging to k-th class is modeled by

$$P(y = k|\mathbf{x}, \mathbf{w}) = \frac{\exp\left(\mathbf{w}_k^T \cdot \mathbf{x}\right)}{\sum_{j=1}^{K} \exp\left(\mathbf{w}_j^T \cdot \mathbf{x}\right)},$$

where $\mathbf{w} = (\mathbf{w}_1, \mathbf{w}_2, \cdots, \mathbf{w}_K)$.

To learn the weights $\mathbf{w}$, we consider the cross-entropy loss function (maybe with regularization terms)

$$\mathcal{L}(\mathbf{w}) = \sum_i -\log(P(y = y_i|\mathbf{x}_i, \mathbf{w})),$$

where the sum is taken over the training data $\{(\mathbf{x}_i, y_i)\}$.

SGD v.s. LS-SGD - Softmax Regression

Consider the MNIST hand written digits recognition by using the Softmax regression. The models are trained by running 100 epochs of SGD and LS-SGD respectively on the 60000 training instances with batch size 100 and learning rate 0.05.

$\sigma = 0$ $\sigma = 0.2$ $\sigma = 0.5$ $\sigma = 0.8$

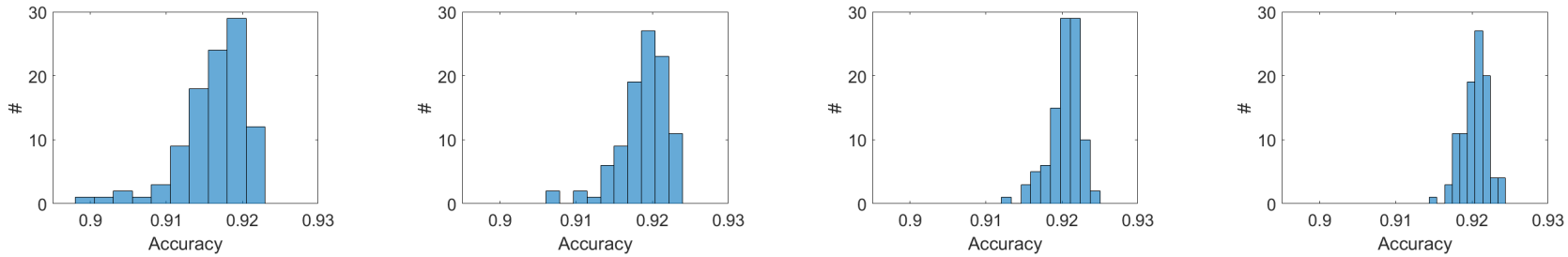


Figure: The histogram of generalization accuracies of the softmax regression model trained with LS-SGD over 100 independent experiments by using different σ.

SGD v.s. LS-SGD - Softmax Regression

Table: Accuracies of the softmax regression model trained with LS-SGD. Statistics over 100 independent experiments for each σ. Note, $\sigma = 0$ represents SGD.

σ	Min	Max	Average	Variance	σ	Min	Max	Average	Variance
0.0	0.899	0.923	0.916	1.81e-5					
0.1	0.906	0.924	0.918	1.10e-5	0.6	0.917	0.925	0.921	3.17e-6
0.2	0.907	0.924	0.919	1.06e-5	0.7	0.915	0.924	0.921	3.30e-6
0.3	0.908	0.924	0.920	8.16e-6	0.8	0.915	0.924	0.921	2.73e-6
0.4	0.910	0.925	0.920	7.48e-6	0.9	0.917	0.924	0.920	2.80e-6
0.5	0.912	0.924	0.920	4.21e-6	1.0	0.914	0.924	0.920	3.86e-6

LeNet5

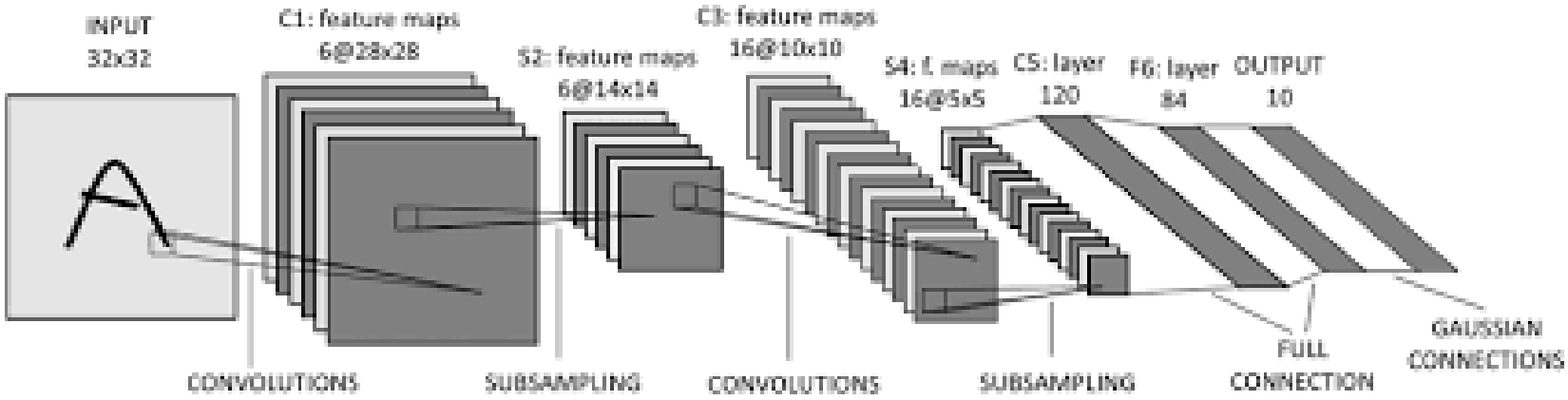


Figure: Architecture of LeNet5.

LeCun et al, Proc. IEEE, 1998

SGD v.s. LS-SGD - LeNet5

Consider MNIST image recognition by LeNet5, where

$$\text{LeNet} : \text{input}_{28\times 28} \to \text{conv}_{20,5,2} \to \text{conv}_{50,5,2} \to \text{fc}_{512} \to \text{softmax}.$$

We train the model by running 100 epochs LS-SGD with learning rate 0.05 and batch size 100.

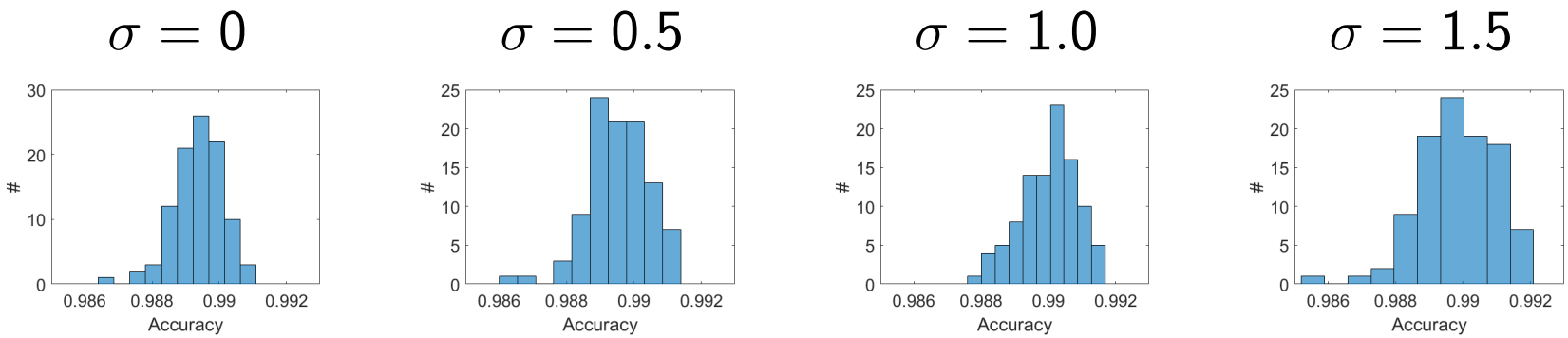


Figure: The histogram of generalization accuracies of the LeNet5 on MNIST trained with LS-SGD over 100 independent experiments by using different σ. No momentum, weight decay, or any other techniques is used to improve the generalization.

SGD v.s. LS-SGD - LeNet5

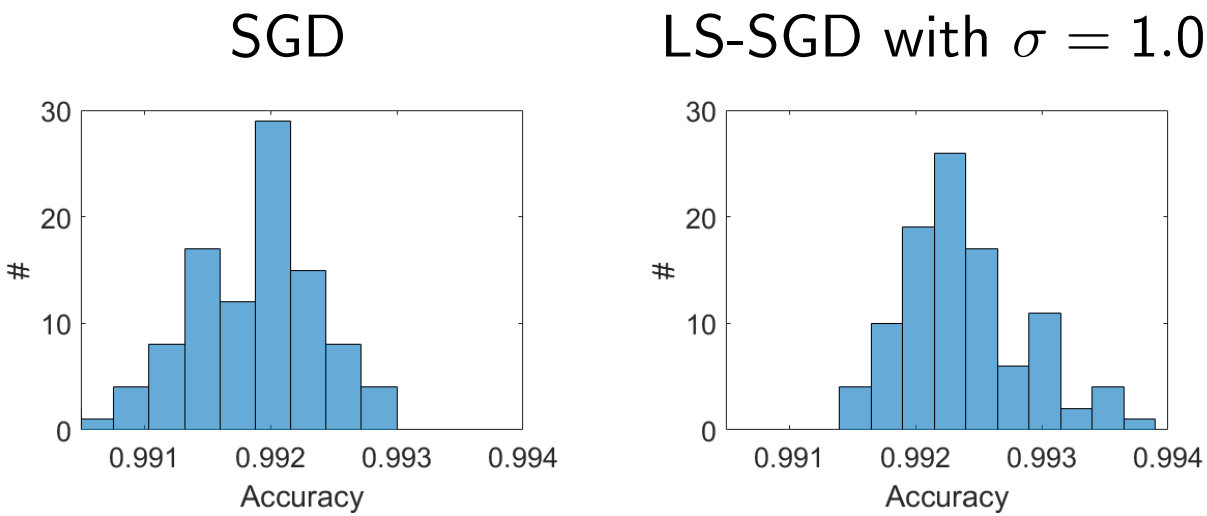


Figure: The histogram of generalization accuracy of the LeNet5 on MNIST trained with LS-SGD over 100 independent experiments by using different σ. The Nesterov momentum and weight decay are used to boost the performance.

Nesterov, 1983

LS-SGD is Suitable for Small Batch Size

The importance of using small batch size.

Wu and He, Arxiv1803.08494, 2018.

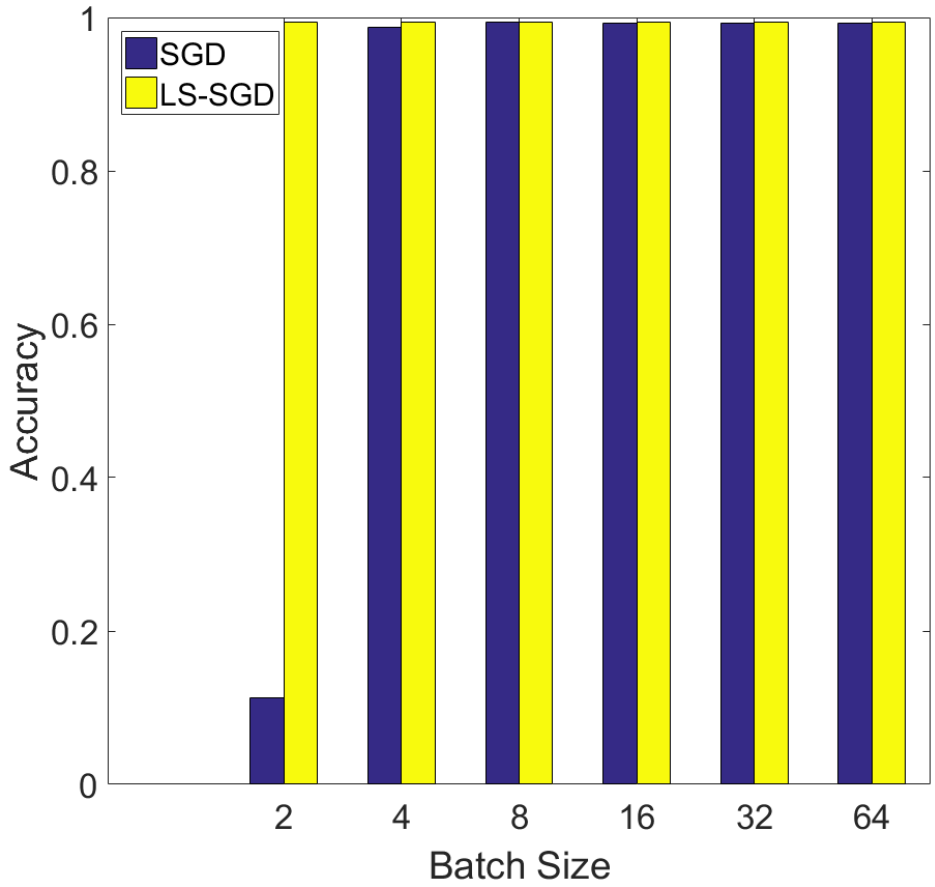


Figure: Generalization accuracy of LeNet5 trained with different batch sizes by SGD and LS-SGD.

ResNet

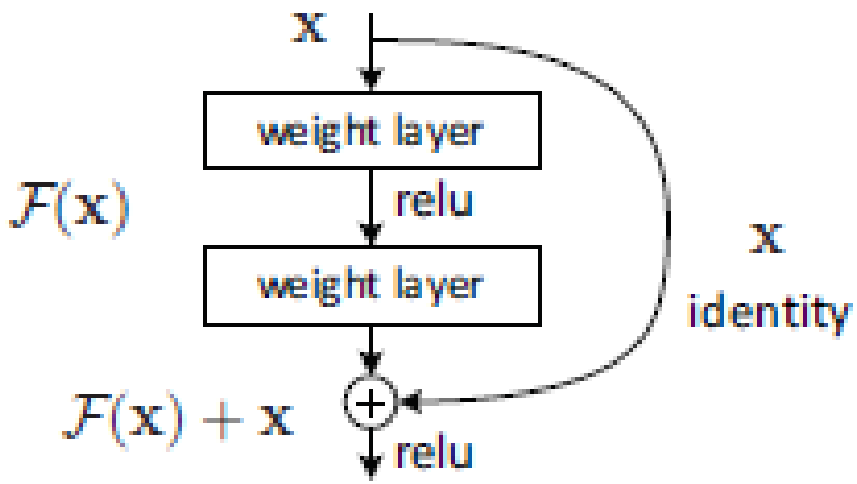


Figure: ResNet build block.

He et al, CVPR, 2016

SGD v.s. LS-SGD - ResNet

On Cifar10, we compare the performance of LS-SGD and SGD on ResNet with the pre-activated ResNet56. We take the same training strategy as reported in the original paper, except that we run 200 epochs with the learning rate decayed by a factor of 5 after every 40 epochs. For ResNet, instead of applying LS-SGD for all epochs. We only use LS-SGD in the first 40 epochs, and the remaining training will be carried out by SGD. The parameter σ is set to 1.0.

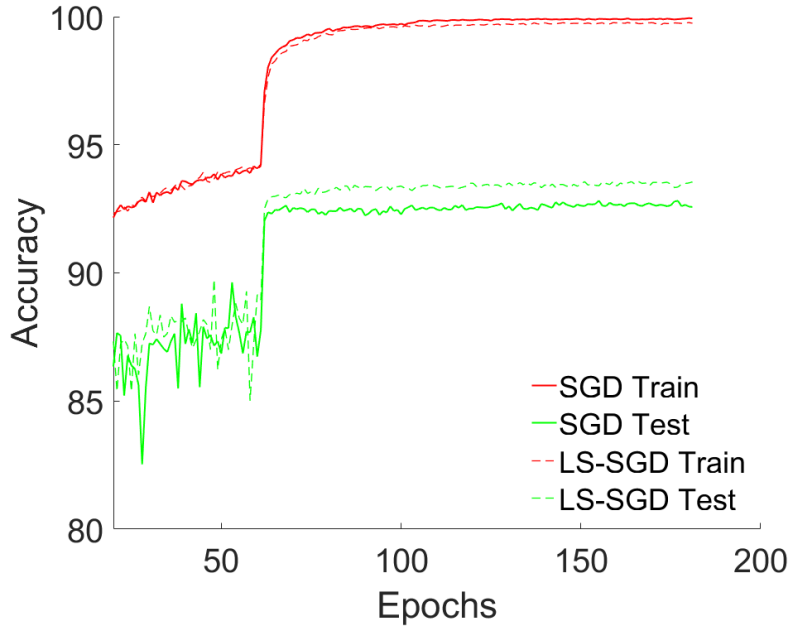


Figure: The evolution of the pre-activated ResNet56's training and generalization accuracies by SGD and LS-SGD.

SGD v.s. LS-SGD - ResNet

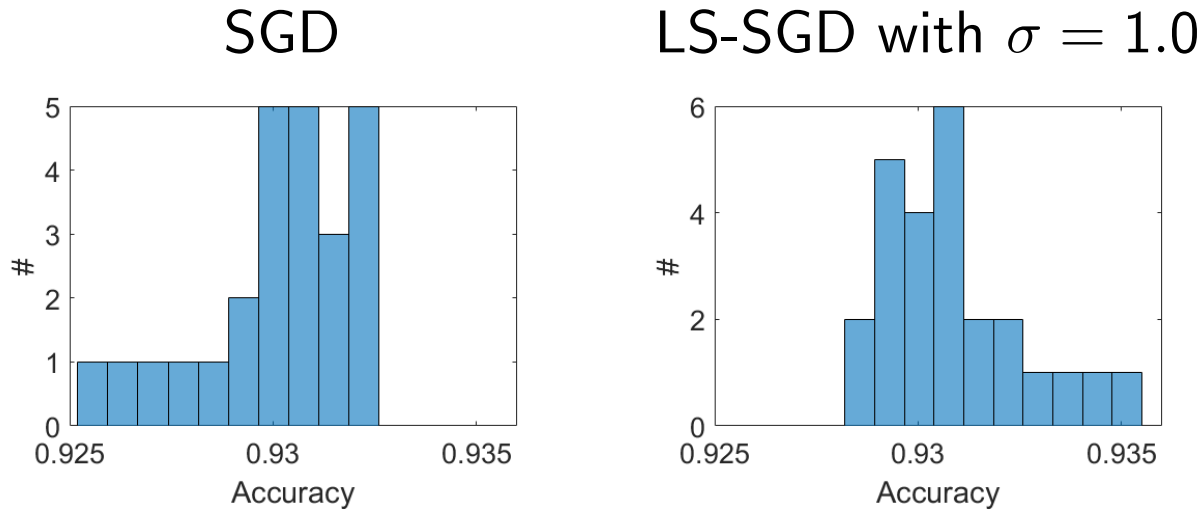


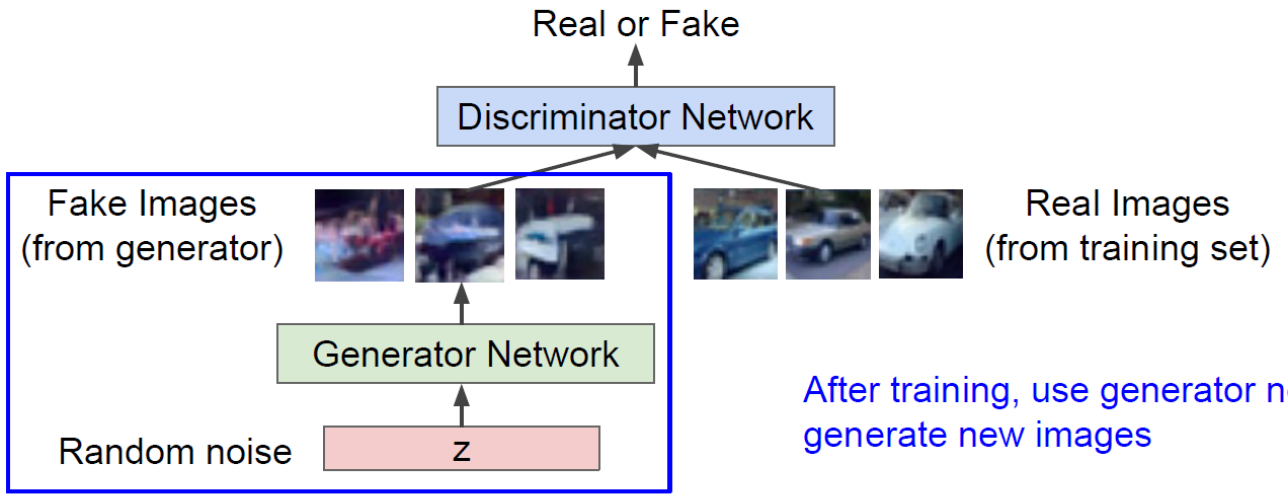
Figure: The histogram of the generalization accuracies of the pre-activated ResNet56 on Cifar10 trained with LS-SGD over 25 independent experiments.

GAN and WGAN

Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Generator network: try to fool the discriminator by generating real-looking images
Discriminator network: try to distinguish between real and fake images



After training, use generator network to generate new images

Fake and real images copyright Emily Denton et al. 2015. Reproduced with permission.

Standard GAN: KL divergence for discriminator loss.
WGAN: Wasserstein distance for discriminator loss.

WGAN

By Kantorovich-Rubinstein duality, Wasserstein distance between two probability distributions can be written as

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{||f||_L \leq 1} \mathbb{E}_{x \sim \mathbb{P}_r}[f(x)] - \mathbb{E}_{x \sim \mathbb{P}_\theta}[f(x)],$$

where the discriminator network is used to learn the Lipschitz function f. RMSProp is used typically to train the discriminator.

Arjovsky et al, ICML, 2017

RMSProp v.s. LS-RMSProp - WGAN

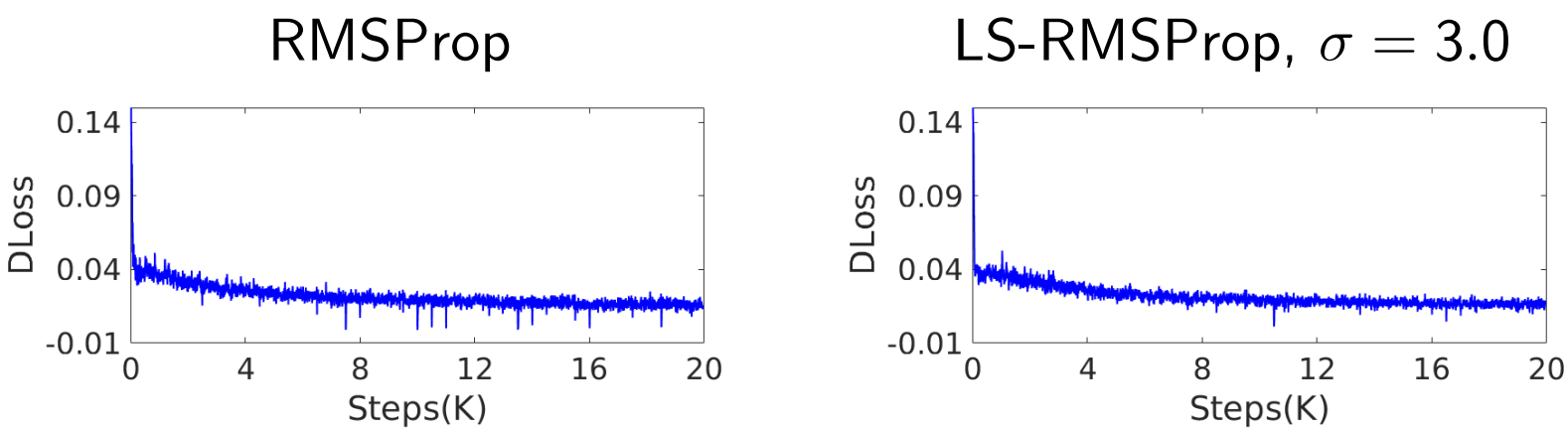
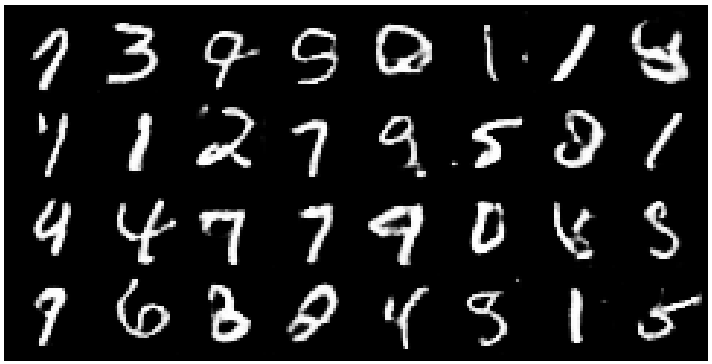


Figure: Critic Loss with $lrD = 0.0001$, $lrG = 0.005$ for RMSProp (Left) and LS-RMSProp (Right), trained for 20K iterations. We apply a mean filter of window size 13 for better visualization. The loss from LS-RMSProp is visibly less noisy.

RMSProp v.s. LS-RMSProp - WGAN

RMSProp



LS-RMSProp, $\sigma = 3.0$

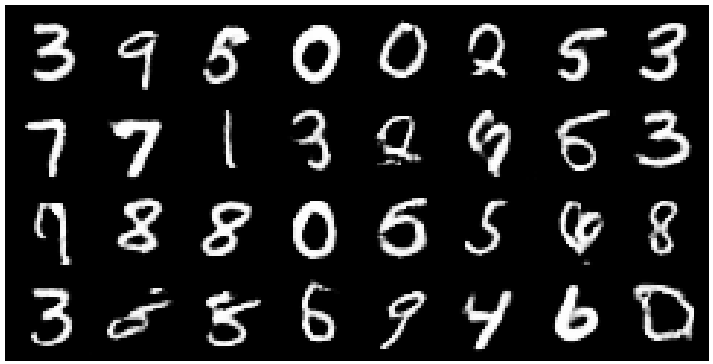


Figure: Samples from WGAN trained with RMSProp (left) and LS-RMSProp (right). The learning rate is set to $lrD = 0.0001$, $lrG = 0.005$ for both RMSProp and LS-RMSProp. The critic is trained for 5 iterations per every step of the generator, and 200 iterations per every 500 steps of the generator.

Reference and Future Work

Ref: Stanley Osher, Bao Wang, Penghang Yin, Xiyang Luo, Minh Pham and Alex Lin, Laplacian Smoothing Gradient Descent, Arxiv 1806.06317

- Choose σ adaptively on-the-fly.
- Solve tri-diagonal matrix by linear scaling algorithm. And GPU implementation.

Deep Learning with Data Dependent Implicit Activation Functions

Stan Osher's Group [2]
Dept of Math, UCLA

[2] Thanks to professor Andrea Bertozzi's group for helping us.

Deep Neural Network: Coarse Grained Representation

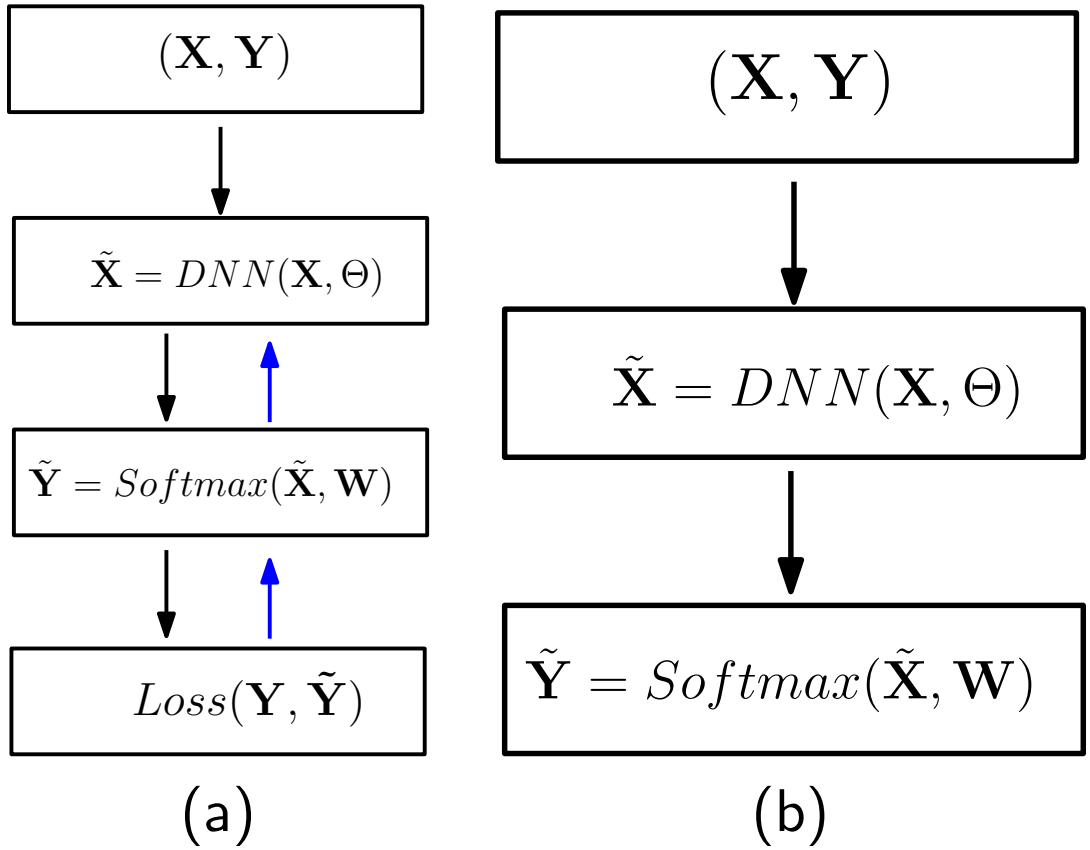


Figure: Training (a) and testing (b) procedures of DNNs with softmax as output activation layer.

Softmax Activation

$$y_i = \frac{e^{\mathbf{w}_i \cdot \mathbf{x}}}{\sum_j e^{\mathbf{w}_j \cdot \mathbf{x}}},$$

which is essentially a linear model on the feature space.

- Does not sufficiently utilize the manifold structure of feature space.
- May not be stable to small perturbation.

We use interpolating function as output activation!

Manifold Interpolation-Implicit Activation

Let $P = \{\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_n\}$ be a set of points on a manifold $\mathcal{M} \subset \mathbf{R}^d$ with the labeled subset $S = \{\mathbf{s}_1, \mathbf{s}_2, \cdots, \mathbf{s}_m\}$.

How to extend the labels of S to P?

Harmonic extension by minimizing the Dirichlet energy:

$$\mathcal{E}(u) = \frac{1}{2} \sum_{\mathbf{x},\mathbf{y} \in P} w(\mathbf{x}, \mathbf{y}) \left(u(\mathbf{x}) - u(\mathbf{y})\right)^2,$$

with the boundary condition:

$$u(\mathbf{x}) = g(\mathbf{x}), \ \mathbf{x} \in S,$$

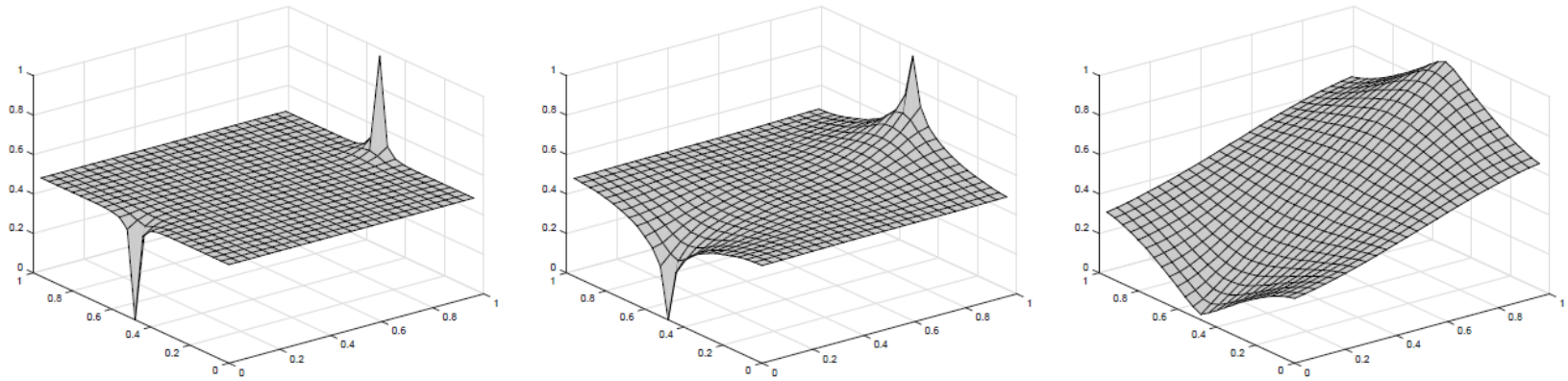
The Euler-Lagrange equation for the above energy minimization problem is:

$$\begin{cases} \sum_{\mathbf{y} \in P} \left(w(\mathbf{x}, \mathbf{y}) + w(\mathbf{y}, \mathbf{x})\right) \left(u(\mathbf{x}) - u(\mathbf{y})\right) = 0 & \mathbf{x} \in P/S \\ u(\mathbf{x}) = g(\mathbf{x}) & \mathbf{x} \in S, \end{cases}$$

We infer the label implicitly!

Manifold Interpolation-Implicit Activation

How about only tiny amount of data is labeled?



$$\begin{cases} \sum_{\mathbf{y}\in P} \left(w(\mathbf{x},\mathbf{y}) + w(\mathbf{y},\mathbf{x})\right)\left(u(\mathbf{x}) - u(\mathbf{y})\right) + \\ \left(\frac{|P|}{|S|} - 1\right) \sum_{\mathbf{y}\in S} w(\mathbf{y},\mathbf{x})\left(u(\mathbf{x}) - u(\mathbf{y})\right) = 0 & \mathbf{x} \in P/S \\ u(\mathbf{x}) = g(\mathbf{x}) & \mathbf{x} \in S, \end{cases}$$

we use the **weighted nonlocal Laplacian (WNLL)** instead of the graph Laplacian (GL)!

Shi et al, JSC, 2017

Manifold Interpolation-Implicit Activation

How many instances should be labeled at least?

$$N\left(1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{N}\right)\approx N\ln N,$$

where N is the number of classes in the dataset.

How to find the weight function w?

Approximate nearest neighbor (ANN) searching!

Muja et al, PAMI, 2014.

Network Structure Design

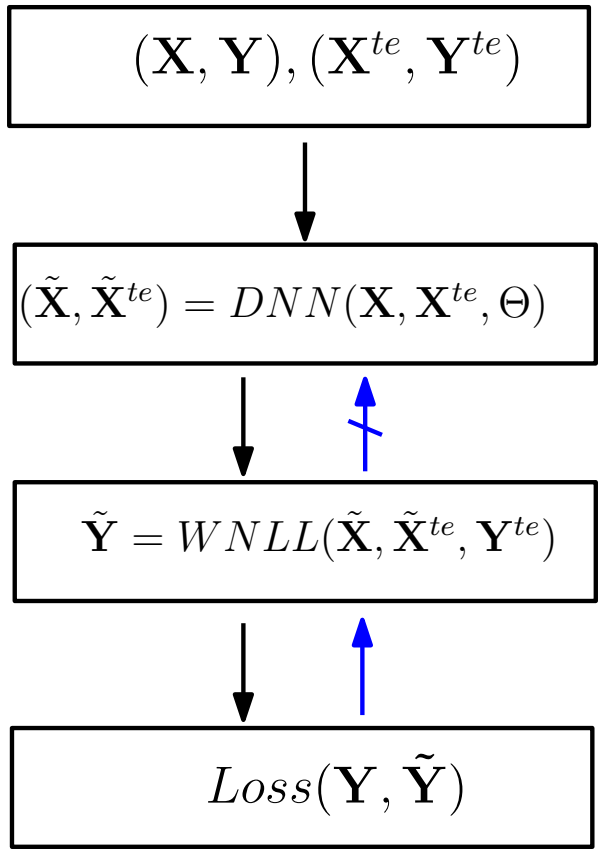


Figure: WNLL activation - first design.

Error cannot be back propagated, since the WNLL is an implicit function whose gradient is not explicitly available!

Network Structure Design

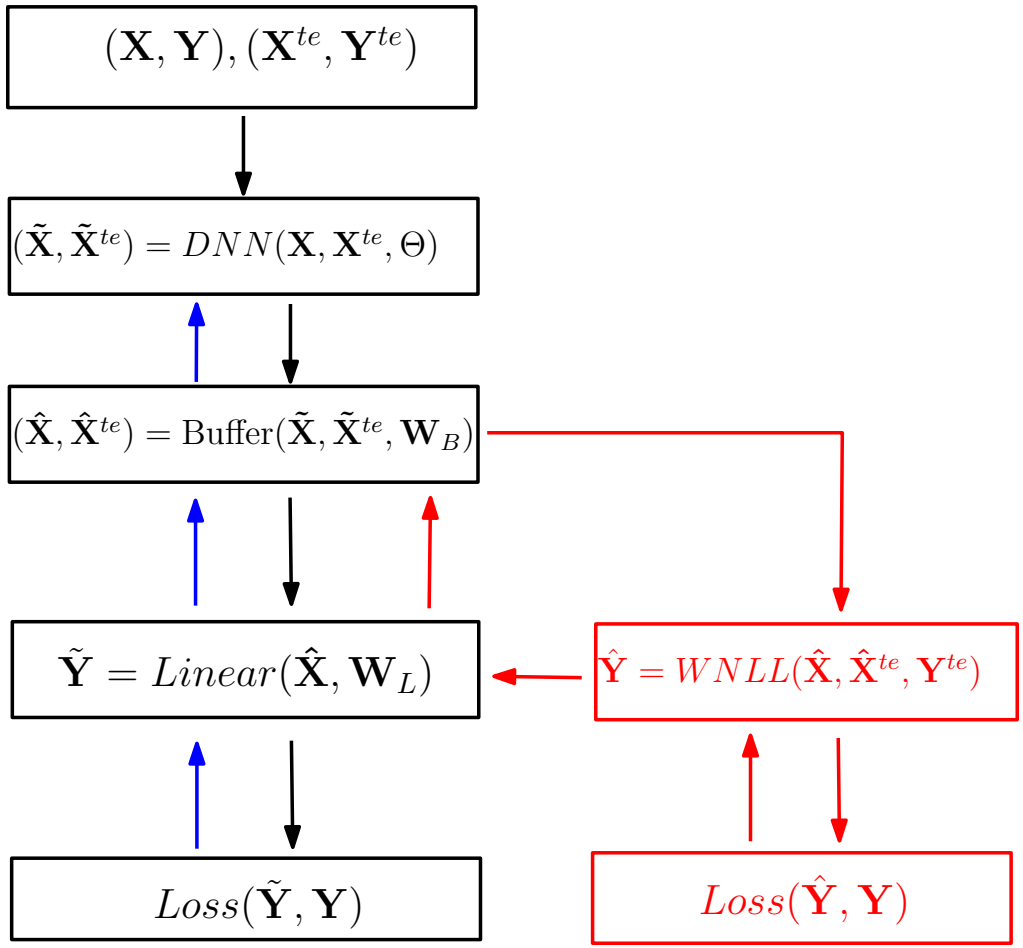


Figure: WNLL activation - second design.

Network Structure Design

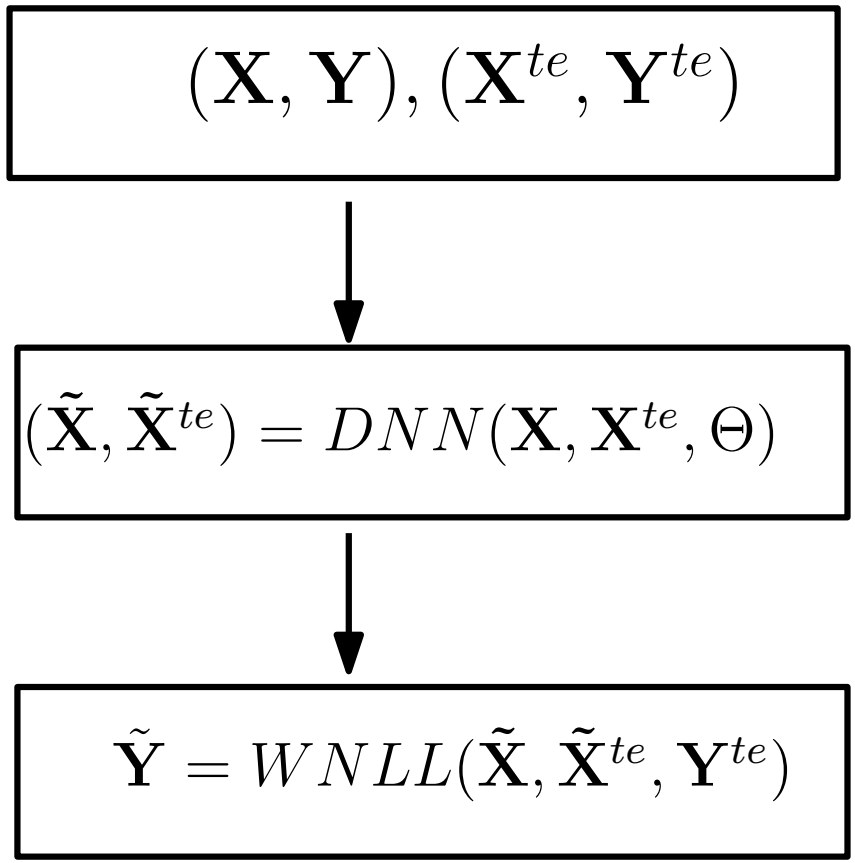


Figure: Deep Neural Network with WNLL Activation - testing.

Training Algorithm

Alternating between training linear and WNLL activated DNNs:

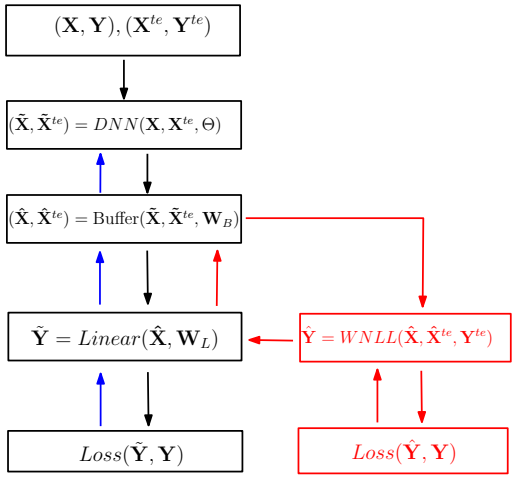


Figure: Deep Neural Network with WNLL Activation.

Train DNNs with linear activation: Run N_1 steps of forward and back propagation, where in kth iteration:

Forward propagation: The training data $\mathbf{X}$ is transformed, respectively, by DNN, Buffer and Linear blocks to the predicted labels $\tilde{\mathbf{Y}}$:

$$\tilde{\mathbf{Y}} = \text{Linear}(\text{Buffer}(\text{DNN}(\mathbf{X}, \Theta^{k-1}), \mathbf{W}_B^{k-1}), \mathbf{W}_L^{k-1}).$$

Then compute loss between the ground truth labels $\mathbf{Y}$ and predicted ones $\tilde{\mathbf{Y}}$, denoted as $\mathcal{L}^{\text{Linear}}$.

Backpropagation: Update weights (Θ^{k-1}, $\mathbf{W}_B^{k-1}$, $\mathbf{W}_L^{k-1}$) by gradient descent:

$$\mathbf{W}_L^k = \mathbf{W}_L^{k-1} - \gamma \frac{\partial \mathcal{L}^{\text{Linear}}}{\partial \tilde{\mathbf{Y}}} \cdot \frac{\partial \tilde{\mathbf{Y}}}{\partial \mathbf{W}_L},$$

$$\mathbf{W}_B^k = \mathbf{W}_B^{k-1} - \gamma \frac{\partial \mathcal{L}^{\text{Linear}}}{\partial \tilde{\mathbf{Y}}} \cdot \frac{\partial \tilde{\mathbf{Y}}}{\partial \hat{\mathbf{X}}} \cdot \frac{\partial \hat{\mathbf{X}}}{\partial \mathbf{W}_B},$$

$$\Theta^k = \Theta^{k-1} - \gamma \frac{\partial \mathcal{L}^{\text{Linear}}}{\partial \tilde{\mathbf{Y}}} \cdot \frac{\partial \tilde{\mathbf{Y}}}{\partial \hat{\mathbf{X}}} \cdot \frac{\partial \hat{\mathbf{X}}}{\partial \tilde{\mathbf{X}}} \cdot \frac{\partial \tilde{\mathbf{X}}}{\partial \Theta}.$$

Training Algorithm

Alternating between training linear and WNLL activated DNNs:

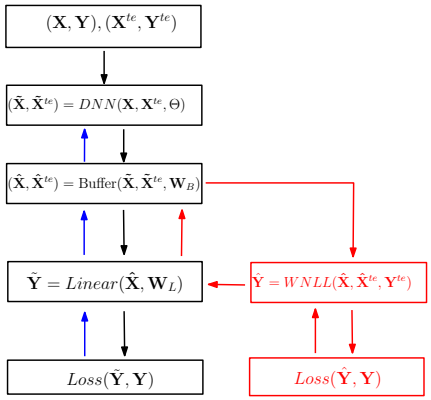


Figure: Deep Neural Network with WNLL Activation.

Train DNNs with WNLL activation: Run N_2 steps of the following forward and back propagation, where in kth iteration, we have:

Forward propagation: The training data $\mathbf{X}$, template $\mathbf{X}^{\text{te}}$ and $\mathbf{Y}^{\text{te}}$ are transformed, respectively, by DNN, Buffer, and WNLL blocks to get predicted labels $\hat{\mathbf{Y}}$:

$$\hat{\mathbf{Y}} = \text{WNLL}(\text{Buffer}(\text{DNN}(\mathbf{X}, \Theta^{k-1}), \mathbf{W}_B^{k-1}), \hat{\mathbf{X}}^{\text{te}}, \mathbf{Y}^{\text{te}}).$$

Then compute loss, $\mathcal{L}^{\text{WNLL}}$, between the ground truth labels $\mathbf{Y}$ and predicted ones $\hat{\mathbf{Y}}$.

Backpropagation: Update weights $\mathbf{W}_B^{k-1}$ only, $\mathbf{W}_L^{k-1}$ and Θ^{k-1} will be tuned in the next iteration in training DNNs with linear activation, by gradient descent.

$$\begin{aligned}\mathbf{W}_B^k &= \mathbf{W}_B^{k-1} - \gamma \frac{\partial \mathcal{L}^{\text{WNLL}}}{\partial \hat{\mathbf{Y}}} \cdot \frac{\partial \hat{\mathbf{Y}}}{\partial \hat{\mathbf{X}}} \cdot \frac{\partial \hat{\mathbf{X}}}{\partial \mathbf{W}_B} \\ &\approx \mathbf{W}_B^{k-1} - \gamma \frac{\partial \mathcal{L}^{\text{Linear}}}{\partial \tilde{\mathbf{Y}}} \cdot \frac{\partial \tilde{\mathbf{Y}}}{\partial \hat{\mathbf{X}}} \cdot \frac{\partial \hat{\mathbf{X}}}{\partial \mathbf{W}_B}.\end{aligned}$$

Theoretical Explanation - I

In training WNLL activated DNNs, the two output activation functions in the auxiliary networks are, in a sense, each competing to minimize its own objective where, in equilibrium, the neural nets can learn better features for both linear and interpolation-based activations. This in flavor is similar to generative adversarial nets (GAN).

Goodfellow, et al, NIPS, 2014.

Theoretical Explanation - II

In the continuum limit, ResNet can be modeled as the following control problem for a transport equation:

$$\begin{cases} \frac{\partial u(\mathbf{x},t)}{\partial t} + \mathbf{v}(\mathbf{x},t) \cdot \nabla u(\mathbf{x},t) = 0 & \mathbf{x} \in \mathbf{X}, t \geq 0 \\ u(\mathbf{x},1) = f(\mathbf{x}) & \mathbf{x} \in \mathbf{X}. \end{cases} \tag{3}$$

Here $u(\cdot, 0)$ is the input of the continuum version of ResNet, which maps the training data to the corresponding label. $f(\cdot)$ is the terminal value which analogous to the output activation function in ResNet which maps deep features to the predicted label. Training ResNet is equivalent to tuning $\mathbf{v}(\cdot, t)$, i.e., continuous version of the weights, s.t. the predicted label $f(\cdot)$ matches that of the training data. If $f(\cdot)$ is a harmonic extension of $u(\cdot, 0)$, the corresponding weights $v(\mathbf{x}, t)$ would be close to zero. This results in a simpler model and may generalize better from a model selection point of view.

Numerical Results

(a) (b)

Figure: CIFAR image recognition tasks.

Accuracy of Some Simple Classifiers

Table: Accuracy of some simple classifiers over different datasets

Dataset	**KNN**	**SVM (RBF Kernel)**	**Softmax**	**WNLL**
Cifar10	32.77% (k=5)	**57.14%**	39.91%	**40.73%**
MNIST	96.40% (k=1)	**97.79%**	92.65%	**97.74%**
SVHN	41.47% (k=1)	**70.45%**	24.66%	**56.17%**

Accuracy Evolution

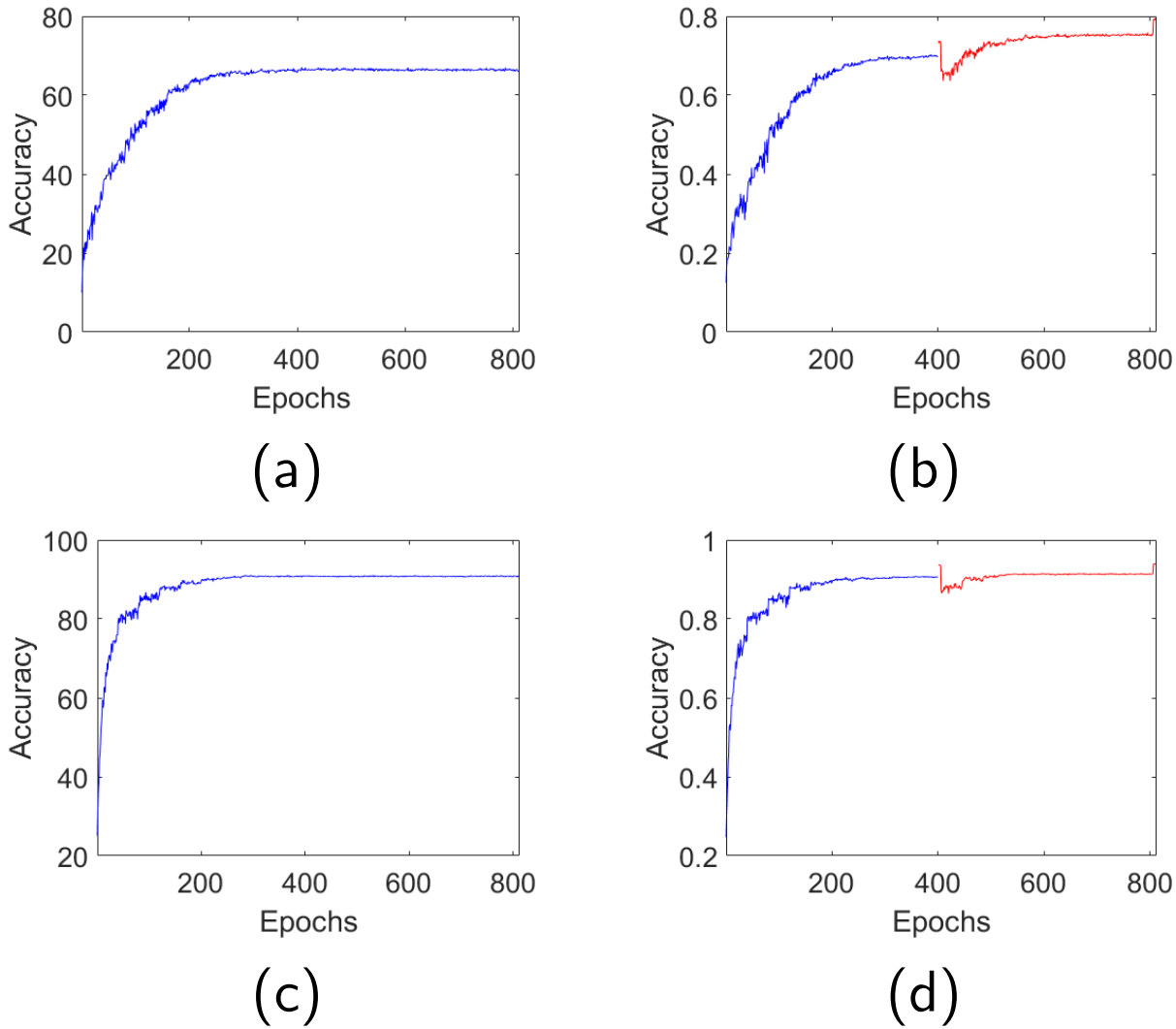


Figure: The evolution of the generation accuracy over the training procedure. Charts (a) and (b) are the accuracy plots for ResNet50 with 1000 number of data for training, where (a) and (b) are plots for the epoch v.s. accuracy of vanilla and WNLL activated DNN. Panels (c) and (d) corresponding to the case of 10000 training data for PreActResNet50. All test are done on Cifar10 dataset.

Degradation of DNN when Lack of Training Data

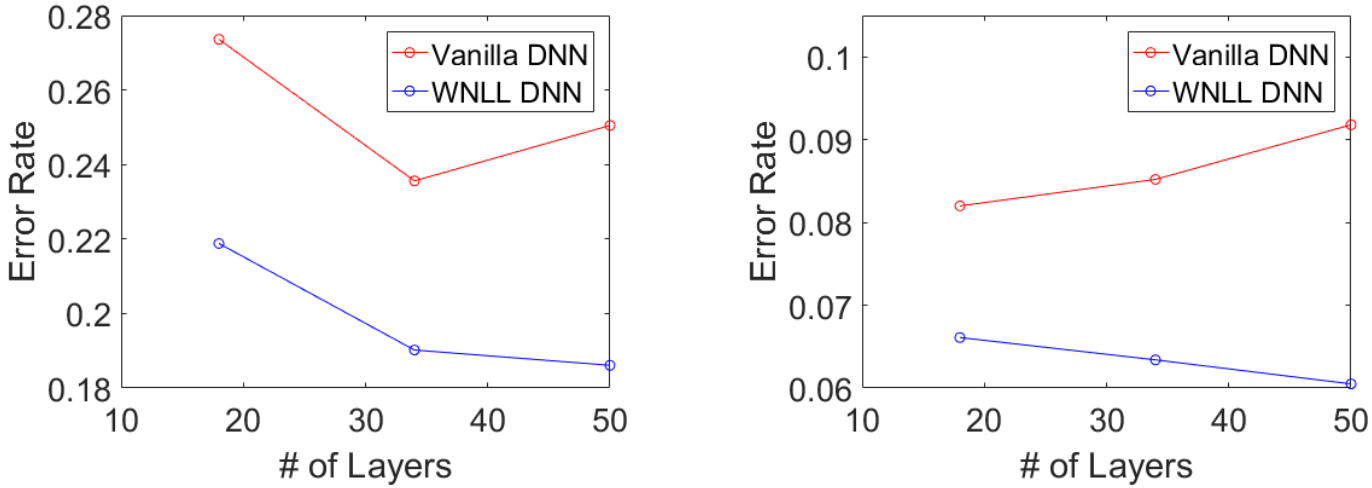


Figure: Taming of the degradation problem of vanilla DNN by WNLL activated DNN. Panels (a) and (b) plot the generation error for cases when 1000 and 10000 training data is used to train the vanilla and WNLL activated DNN, respectively. In each plot, we test three different networks: PreActResNet18, PreActResNet34, and PreActResNet50. It is easy to see that when the vanilla network becomes deeper, the generation error does not decayed, while WNLL activation resolves this degeneracy. All tests are done on Cifar10 dataset.

Performance on CIFAR10

Table: Generalization error rates over the test set of vanilla DNNs, SVM and WNLL activated ones trained over the entire, the first 10000, and the first 1000 instances of training set of CIFAR10. (Median of 5 independent trials)

Network	Whole			10000		1000	
	Vanilla	**WNLL**	**SVM**	**Vanilla**	**WNLL**	**Vanilla**	**WNLL**
VGG13	6.66%	**5.58%**	7.47%	9.12%	**7.64%**	24.85%	**22.56%**
VGG16	6.72%	**5.69%**	7.29%	9.01%	**7.54%**	25.41%	**22.23%**
VGG19	6.95%	**5.92%**	7.99%	9.62%	**8.09%**	25.70%	**22.87%**
ResNet32	7.99%	**5.95%**	8.73%	11.18%	**8.15%**	33.41%	**28.78%**
ResNet44	7.31%	**5.70%**	8.67%	10.66%	**7.96%**	34.58%	**27.94%**
ResNet56	7.24%	**5.61%**	8.58%	9.83%	**7.61%**	37.83%	**28.18%**
ResNet110	6.41%	**4.98%**	8.06%	8.91%	**7.13%**	42.94%	**28.29%**
ResNet18	6.16%	**4.65%**	6.00%	8.26%	**6.29%**	27.02%	**22.48%**
ResNet34	5.93%	**4.26%**	6.32%	8.31%	**6.11%**	26.47%	**20.27%**
ResNet50	6.24%	**4.17%**	6.63%	9.64%	**6.49%**	29.69%	**20.19%**
PreActResNet18	6.21%	**4.74%**	6.38%	8.20%	**6.61%**	27.36%	**21.88%**
PreActResNet34	6.08%	**4.40%**	5.88%	8.52%	**6.34%**	23.56%	**19.02%**
PreActResNet50	6.05%	**4.27%**	5.91%	9.18%	**6.05%**	25.05%	**18.61%**

Performance on CIFAR100

Table: Error rate of vanilla DNN v.s. WNLL activated DNN over the whole Cifar100 dataset. (Median of 5 independent trials)

Network	Vanilla DNN	WNLL DNN
ResNet20	35.79%	**31.53%**
ResNet32	32.01%	**28.04%**
ResNet44	31.07%	**26.32%**
ResNet56	30.03%	**25.36%**
ResNet110	28.86%	**23.74%**
ResNet18	27.57%	**22.89%**
ResNet34	25.55%	**20.78%**
ResNet50	25.09%	**20.45%**
PreActResNet18	28.62%	**23.45%**
PreActResNet34	26.84%	**21.97%**
PreActResNet50	25.95%	**21.51%**

Summary

- DNN with data dependent implicit activation.
- Back propagate the gradient of harmonic function by linear function.
- Resolve the degradation problem.
- Relatively 20%-30% accuracy improvement on both CIFAR10 and CIFAR100.
- Reduce the model's size.
- **On going: imageNet challenge: random interpolation.**

Ref: B. Wang, X. Luo, Z. Li, W. Zhu, Z. Shi, and S. Osher, Deep Neural Nets with Interpolating Function as Output Activation, Arxiv 1802.00168

Thank you!