Optimization for Deep Learning

CS 217 Stanford

Boris Ginsburg

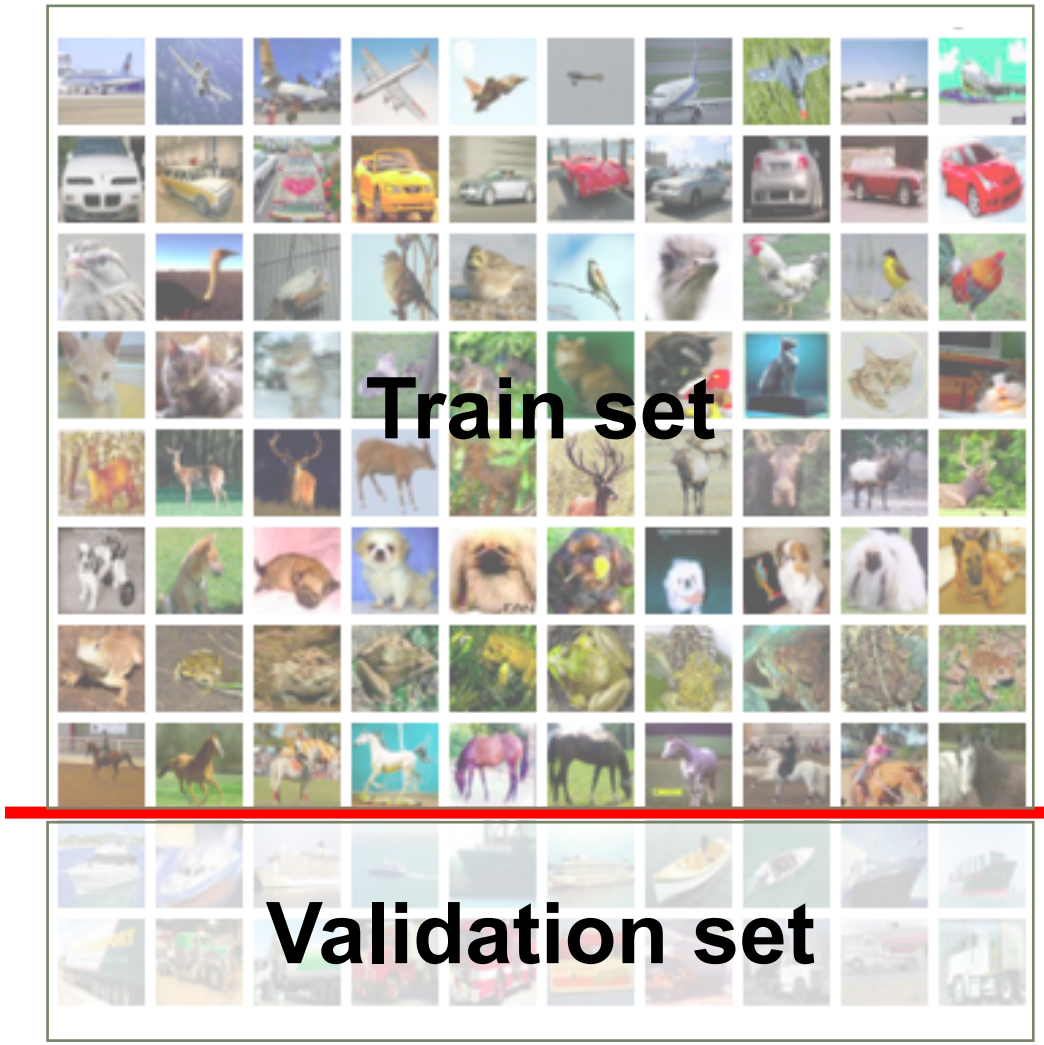
bginsburg@nvidia.com

Agenda

- Introduction to Stochastic Gradient Descent (SGD)
- Optimization algorithms for Deep learning
  - SGD with momentum
  - Adam
- Regularization
  - Deep learning and non-convex optimization
  - Loss surface
  - Good, bad and ugly minima
- Advanced topics
  - Large Batch Training
  - Layer-wise Adaptive Rate Control

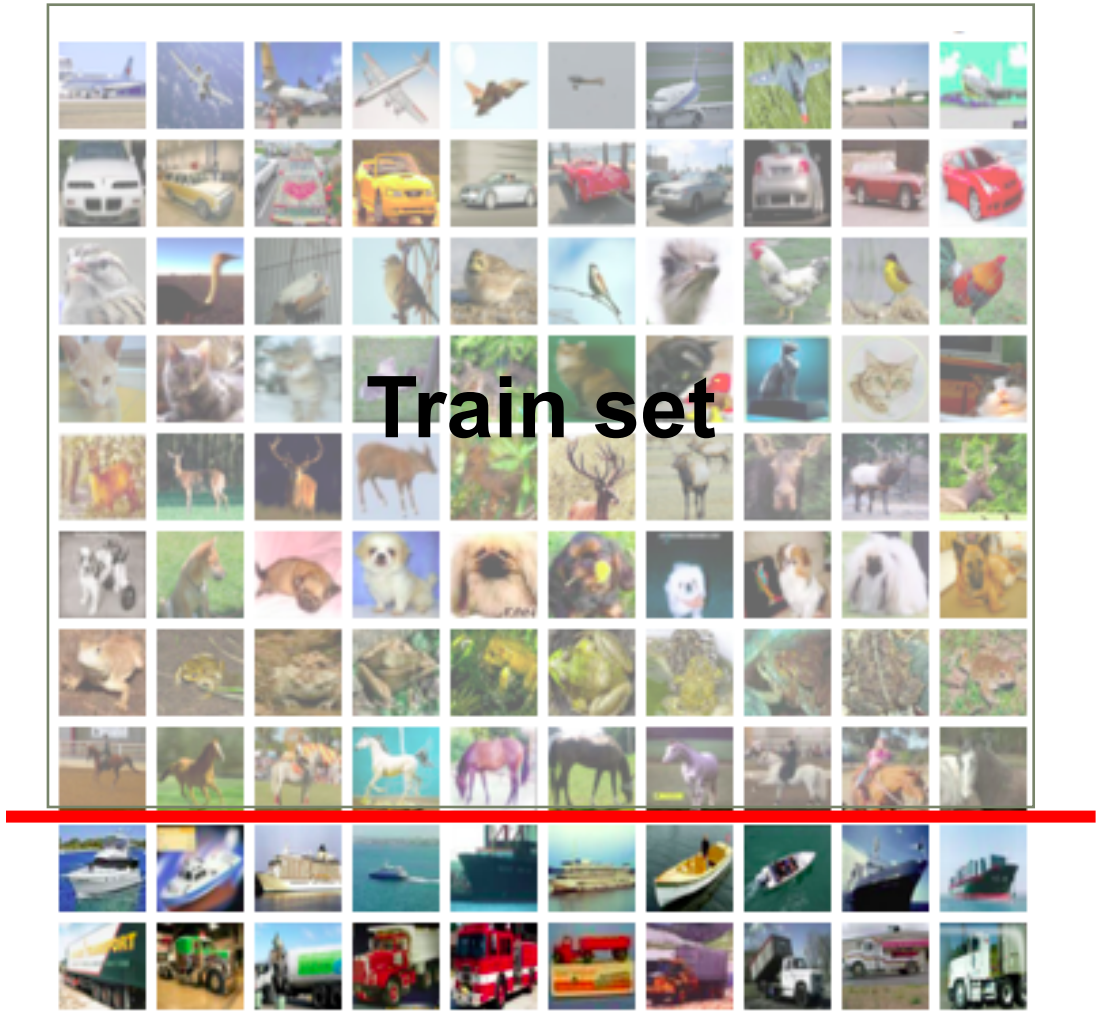
Training procedure

1. Split dataset into training and test
2. Define model:
   - Data preprocessing, network, loss, …
3. Define training parameters:
   - Optimization algorithm, batch size, regularization,…
4. Train for N epoch on training set



Training procedure

1. Split dataset into training and test
2. Define model:
   - Data preprocessing, network, loss, …
3. Define training parameters:
   - Optimization algorithm, batch size, regularization,…
4. Train for N epoch on training set
5. Test on validation set.
6. If not-state of the art goto 2



Part 1: Optimization

Stochastic Gradient Methods: SGD with momentum, Adam, RMSPROP,..

Learning rate

Loss Function Surface

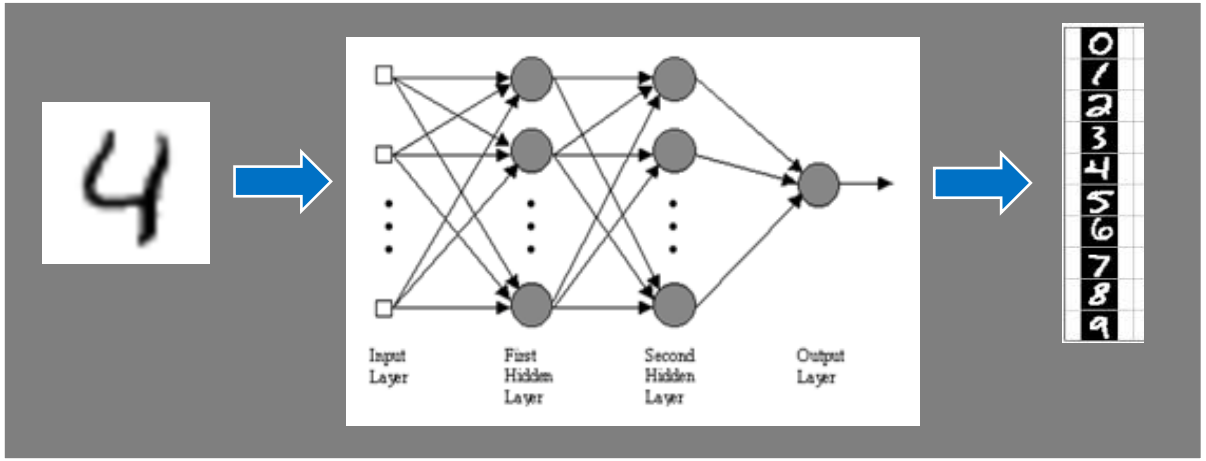
Training as optimization problem

Neural Network is just a complex function

$$y=f(x, w),$$

where

x - input (e.g. image)
w - network parameters (weights)
y - output (e.g. probability that x belongs to each classes)



Batch Gradient Descent

We want to minimize loss over training set with N samples (x_n, y_n):

$$L(w) = \frac{1}{N} \sum_{n=1}^{N} E(f(x_n, w), y_n)$$

Batch GD:

1. accumulate gradients over all samples in training set

$$\frac{\partial E}{\partial w}(t) = \frac{1}{N} \sum_{n=1}^{N} \frac{\partial E}{\partial w}(w, (x_n, y_n))$$

2. update W:

$$w(t+1) = w(t) - \lambda * \frac{\partial E}{\partial w}(t)$$

Issue:
Imagenet has > 10^6 images→ gradient computation for the whole set is expensive

Stochastic Gradient Descent

Stochastic Gradient Descent (on-line learning):

1. Randomly choose sample (x_k, y_k):
2. $W(t+1) = W(t) - \lambda * \frac{\partial E}{\partial w}(w, (x_k, y_k))$

Stochastic Gradient Descent with mini-batches:

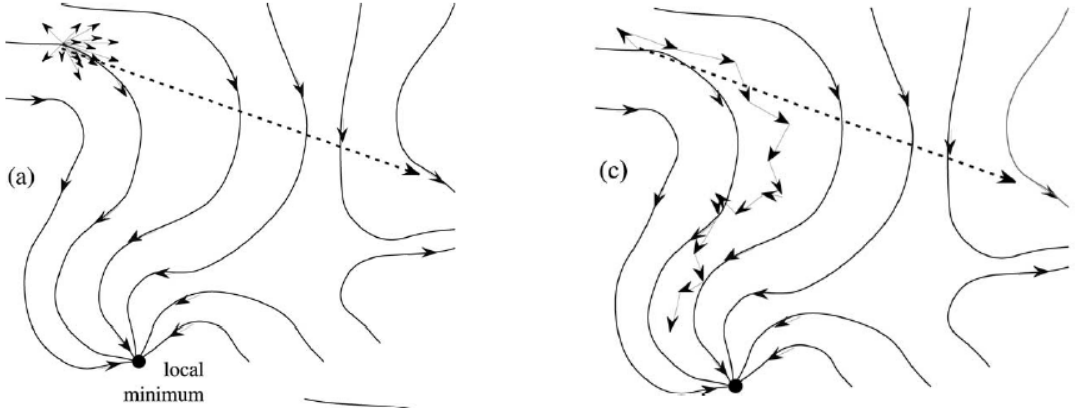
1. divide the dataset into small mini-batches
2. compute the gradient using a single mini-batch
3. make an update
4. move to the next mini-batch …

Don’t forget to shuffle dataset between epochs!

Stochastic Gradient Descent

Batch GD: true gradient computation is heavy, step is small

SGD: gradient is noisy, zig-zags around “true” gradient



Batch GD **SGD**

Wilson, “On the general inefficiency of batch training for gradient descent learning”, 2003

SGD with mini-batch:
- faster than batch GD
- behaves better in local minima and saddle points

“Use stochastic gradient descent when training time is the bottleneck”

Leon Bottou, Stochastic Gradient Descent Tricks

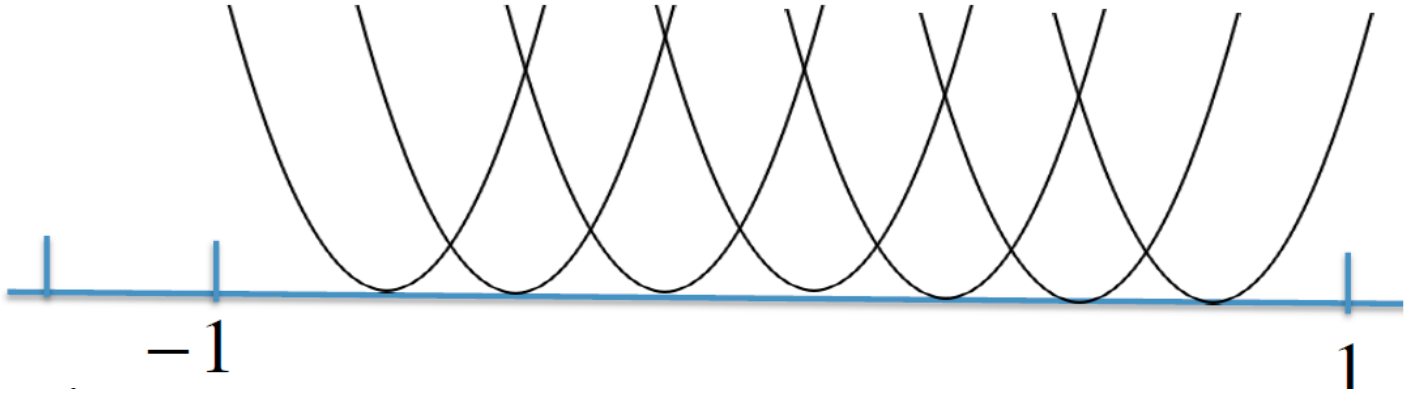
Stochastic Gradients Methods

Machine learning: minimize loss over training set with N samples (x_n, y_n):

$$L(w) = \frac{1}{N} \sum_{n=1}^{N} E(f(w, (x_n, y_n)) = \frac{1}{N} \sum_{n=1}^{N} l_n(w) \rightarrow \min$$

Stochastic optimization: $L(w) = \frac{1}{N} \sum_{n=1}^{N} l_n(w) \rightarrow \min$

EXAMPLE: Lets' take $l_n(w) = (w - w_n)^2$ with w_n evenly distributed in [-1; 1]



Bottou, Curtis, Nocedal "Optimization Methods for Large Scale Machine learning, 2016

Batch Gradient (BG) vs Stochastic Gradient (SG)

BG has linear convergence: after t steps the training error satisfies

$$E(w_t) - E^* < O(\rho^t)$$

where $0 < \rho < 1$. But the iteration cost is proportional to the dataset size!

SG converges with slower sub-linear rate:

$$E(w_t) - E^* < O\left(\frac{1}{t}\right)$$

However iteration cost is cheap, and does not depend on the dataset size.

SG is used with decreasing learning rate, typically: $\lambda(t) \sim \frac{1}{t}$.

Another popular strategy is run with a fixed step size, and, if progress appears to stall, a smaller step size is selected and the process is repeated

Bottou, Curtis, Nocedal "Optimization Methods for Large Scale Machine learning, 2016

In SGD We Trust

Stochastic Gradient Methods play a key role in Machine Learning

Optimization Methods for Large-Scale Machine Learning

Léon Bottou* Frank E. Curtis† Jorge Nocedal‡

June 16, 2016

Minimizing Finite Sums with the Stochastic Average Gradient

Mark Schmidt
schmidtm@cs.ubc.ca

Nicolas Le Roux
nicolas@le-roux.name

Francis Bach
francis.bach@ens.fr

Stochastic Gradients Methods for DL

Two most popular SG methods:

- SGD with momentum (ConvNets)
- Adam (RNNs, Attention)

Other useful SG methods:

- RMSProp
- SAG (Stochastic Average Gradient)
- NAG (“Nesterov’s Accelerated Gradient”)
- Averaged SGD

S. Ruder, “An overview of gradient descent optimization algorithms”
http://ruder.io/optimizing-gradient-descent/

Classical GD with momentum

The momentum method (Polyak, 1964) is a classical convex technique for accelerating GD

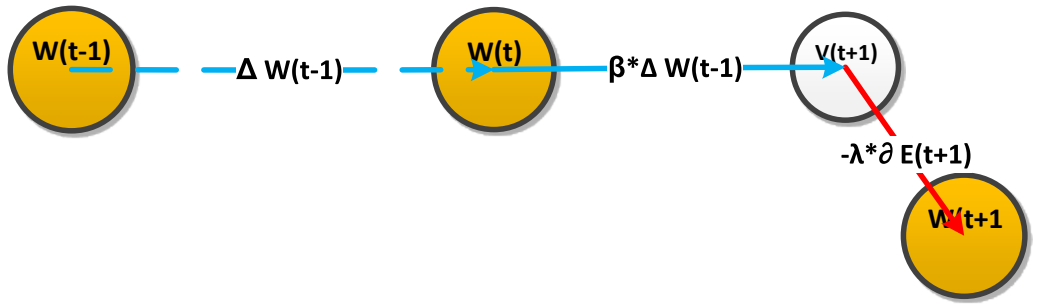
1. Compute momentum:

$$\boldsymbol{M(t+1) = \beta * M(t) - \lambda(t) * \frac{\partial E}{\partial w}(t)}$$

here $\boldsymbol{\beta}$ is momentum coeff $0 < \boldsymbol{\beta} < 1$ and $\boldsymbol{\lambda}$ is learning rate

2. Update weights:

$$W(t+1) = W(t) + M(t+1)$$



G. Goh, "Why Momentum Really Works" , https://distill.pub/2017/momentum/

SGD with momentum

1. Compute SGD over current batch B with samples $(\boldsymbol{x_n, y_n})$:

$$\boldsymbol{G(t) = \frac{1}{B}\sum_{n=1}^{B} \frac{\partial E(w, (x_n, y_n))}{\partial w}}$$

2. Compute momentum:

$$\boldsymbol{M(t+1) = \beta * M(t) - \lambda(t) * G(t)}$$

2. Update weights update:

$$W(t+1) = W(t) + M(t+1)$$

I. Sutskever , "On the importance of initialization and momentum in deep learning"

SGD with momentum as gradients averaging

Let's assume that $\boldsymbol{\lambda}$ is fixed for simplicity.

$$\boldsymbol{M(t+1) = \quad \beta * M(t) - \lambda * G(t) =}$$

$$\boldsymbol{= -\lambda * G(t) + \beta * \big(M(t-1) - \lambda * G(t-1)\big) =}$$

$$\boldsymbol{= -\lambda * G(t) - \lambda * \beta * G(t-1) + \beta * M(t-1) = \cdots}$$

$$\boldsymbol{= -\lambda * \left(\sum_{k=0}^{t} \beta^{t-k} * G(k)\right)}$$

Momentum works as weighted moving average of stochastic gradients:

- First average gradients over batch
- Then weighted "moving" average over previous steps.
- $\boldsymbol{\beta}$ controls how large is averaging window

Stochastic Average Gradient (SAG)

1. Compute SGD over current batch B with samples $(\boldsymbol{x_n, y_n})$:

$$\boldsymbol{G(t) = \frac{1}{B}\sum_{n=1}^{B} \frac{\partial E(w,(x_n,y_n))}{\partial w}}$$

2. Compute **moving average** SG :

$$\boldsymbol{M(t+1) = \beta * M(t) + (1-\beta) * G(t)}$$

2. Update weights update:

$$W(t+1) = W(t) - \boldsymbol{\lambda(t)} * M(t+1)$$

Adagrad and RMSProp

Adagrad (Duchi…, 2010) adapts learning rate for each weight using second momentum:

$$V_i(t) = V_i(t-1) + G_i^2(t)$$

$$\Delta W_i(t+1) = -\frac{\boldsymbol{\lambda}}{\sqrt{V_i(\boldsymbol{t})+\boldsymbol{\epsilon}}} * G_i(t)$$

But Adagrad decreases step sizes too aggressively early in the optimization

RMSProp (Tieleman & Hinton..., 2012) computes running average of 2nd momentum for gradients

$$V_i(t) = \gamma * V_i(t-1) + (1-\gamma) * G_i^2(t)$$

$$\Delta W_i(t+1) = -\frac{\boldsymbol{\lambda(t)}}{\sqrt{V_i(\boldsymbol{t})+\boldsymbol{\epsilon}}} * G_i(t)$$

RMSPROP requires to "manually" decrease global learning rate $\boldsymbol{\lambda(t)}$

Adam

1. Compute stochastic gradient :

$$G(t) = \frac{1}{B}\sum_{n=1}^{B}\frac{\partial E(w,(x_n,y_n))}{\partial w}$$

2. Compute 1st and 2nd momentum:

$$M(t+1) = \beta_1 * M(t) + (1-\beta_1) * G(t)$$

$$V(t+1) = \beta_2 * V(t) + (1-\beta_2) * G^2(t)$$

3. Update weights:

$$W(t+1) = W(t) - \lambda * \frac{M(t+1)}{\sqrt{V(t+1)} + \epsilon}$$

Adam

1. Compute stochastic gradient :

$$G(t) = \frac{1}{B}\sum_{n=1}^{B}\frac{\partial E(w,(x_n,y_n))}{\partial w}$$

2. Compute 1st and 2nd momentum and fix the bias

$$M(t+1) = \beta_1 * M(t) + (1-\beta_1) * G(t) \qquad \widehat{M}(t+1) = \frac{M(t+1)}{1-\beta_1^t}$$

$$V(t+1) = \beta_2 * V(t) + (1-\beta_2) * G^2(t) \qquad \hat{V}(t+1) = \frac{V(t+1)}{1-\beta_2^t}$$

3. Update weights:

$$W(t+1) = W(t) - \lambda * \frac{\widehat{M}(t+1)}{\sqrt{\hat{V}(t+1)} + \epsilon}$$

Exercise: There is a bug in this algorithm. Can you build a counterexample?
Hint: https://arxiv.org/pdf/1705.08292.pdf and https://openreview.net/forum?id=ryQu7f-RZ

Hyperparameters

There are a lot of hyperparameters:

- Model + initialization
- Optimization algorithm
- **Learning Rate**
- Regularization (weight decay, data augmentation, dropout,..)
- Mini-Batch size
- Training duration

Learning Rate

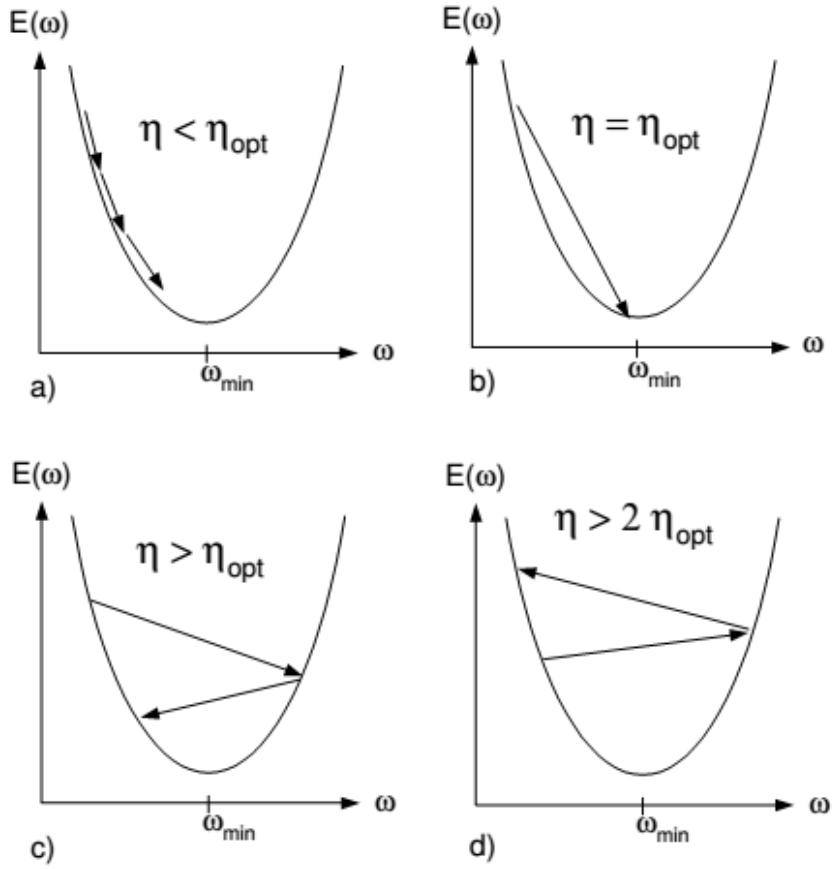
Classical convex optimization:

- GD – fixed small LR
- GD with momentum – decreasing LR

$$\sum_{t=1}^{\infty} \frac{1}{\lambda^2(t)} < \infty \text{ and } \sum_{t=1}^{\infty} \frac{1}{\lambda(t)} = \infty$$

Deep Learning:

- Function non-convex
- SGD – decreasing LR



Why SG-based methods work?

Deep Learning:

- loss function is non-convex:
  - large number of solutions
  - large number of saddles (and local minima?)
- "No bad local minima" hypothesis:
  - large NNs have no local minima, only saddle points and a global minima
- SG algorithms can easily escape from saddle points

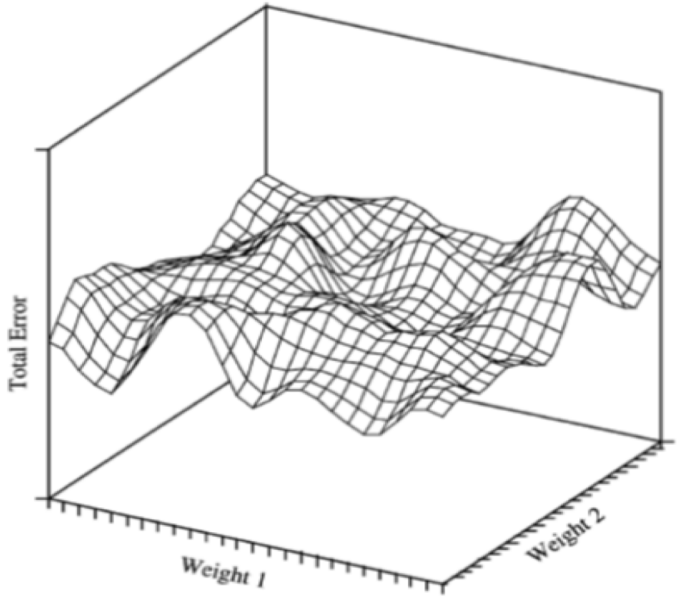


Fig. 1. Illustration of an error surface. Two weights define the x–y plane, and the height indicates the error at each point in weight space.

Wilson, *The general inefficiency of batch training for gradient descent learning*

Baldi, P. and Hornik, K. Neural networks and principal component analysis: Learning from examples without local minima. Neural Networks, 2(1):53–58, 1989.

R. Pascanu, "On the saddle point problem for non-convex optimization", http://arxiv.org/abs/1405.4604

Dauphin, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization", http://arxiv.org/pdf/1406.2572v1.pdf

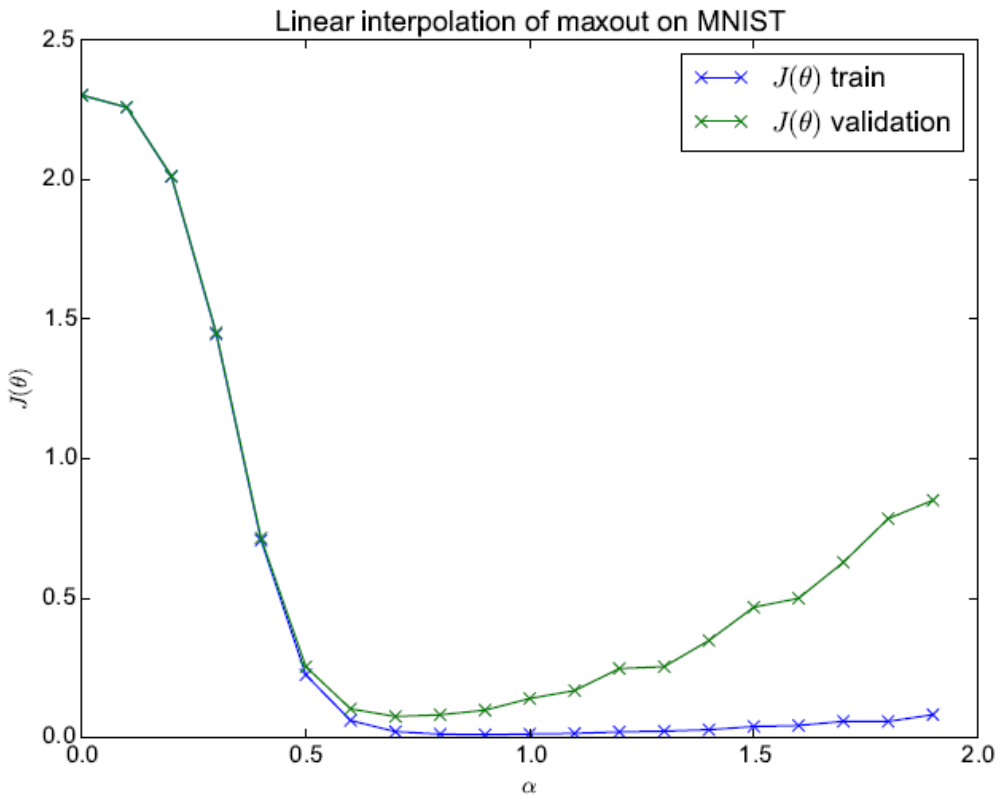
Loss function surface

Simple technique to visualize loss function:

compute loss function along the line which connects initial point X_0 with solution X_1:

$$x(\theta) = x_0 * (1 - \theta) + x_{min} * \theta$$

"These experiments show that SGD does encounter obstacles, such as a ravine that shapes its path, but we never found evidence that local minima or saddle points slowed the SGD trajectory"

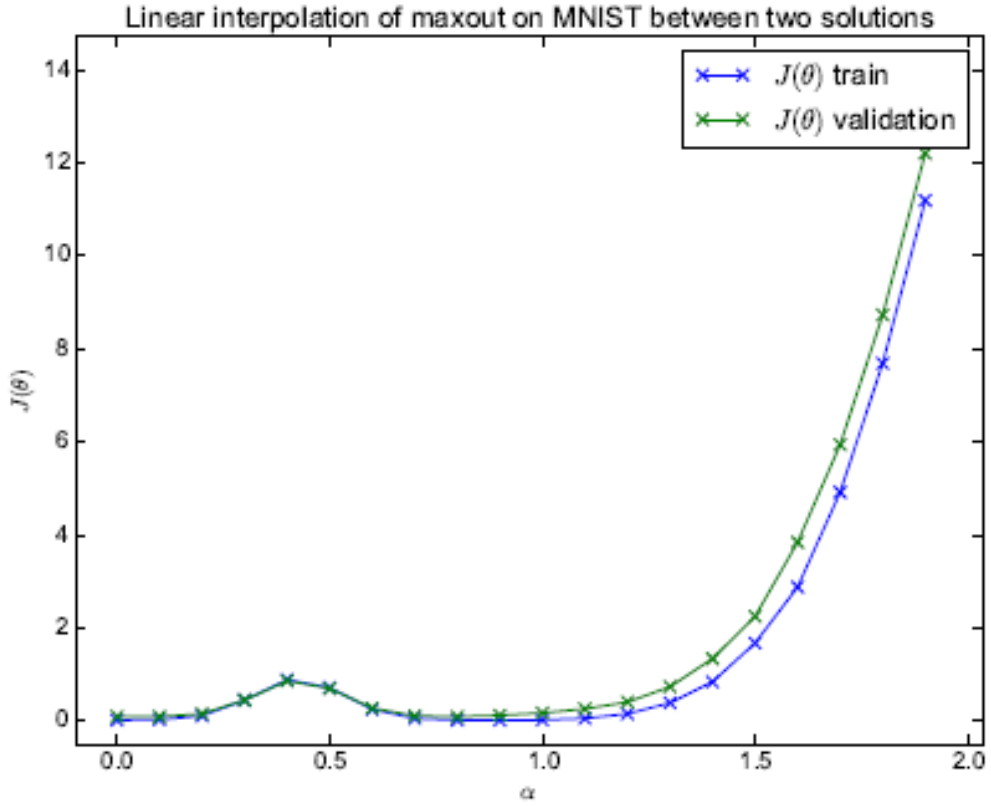


I.Goodfellow, O.Vinyals, A. Saxe, "Qualatively Characterizing NN optimization problems"

Loss function surface

What if we connect two solutions?

There is a small saddle between them…



I.Goodfellow, O.Vinyals, A. Saxe, "Qualitatively characterizing NN optimization problems

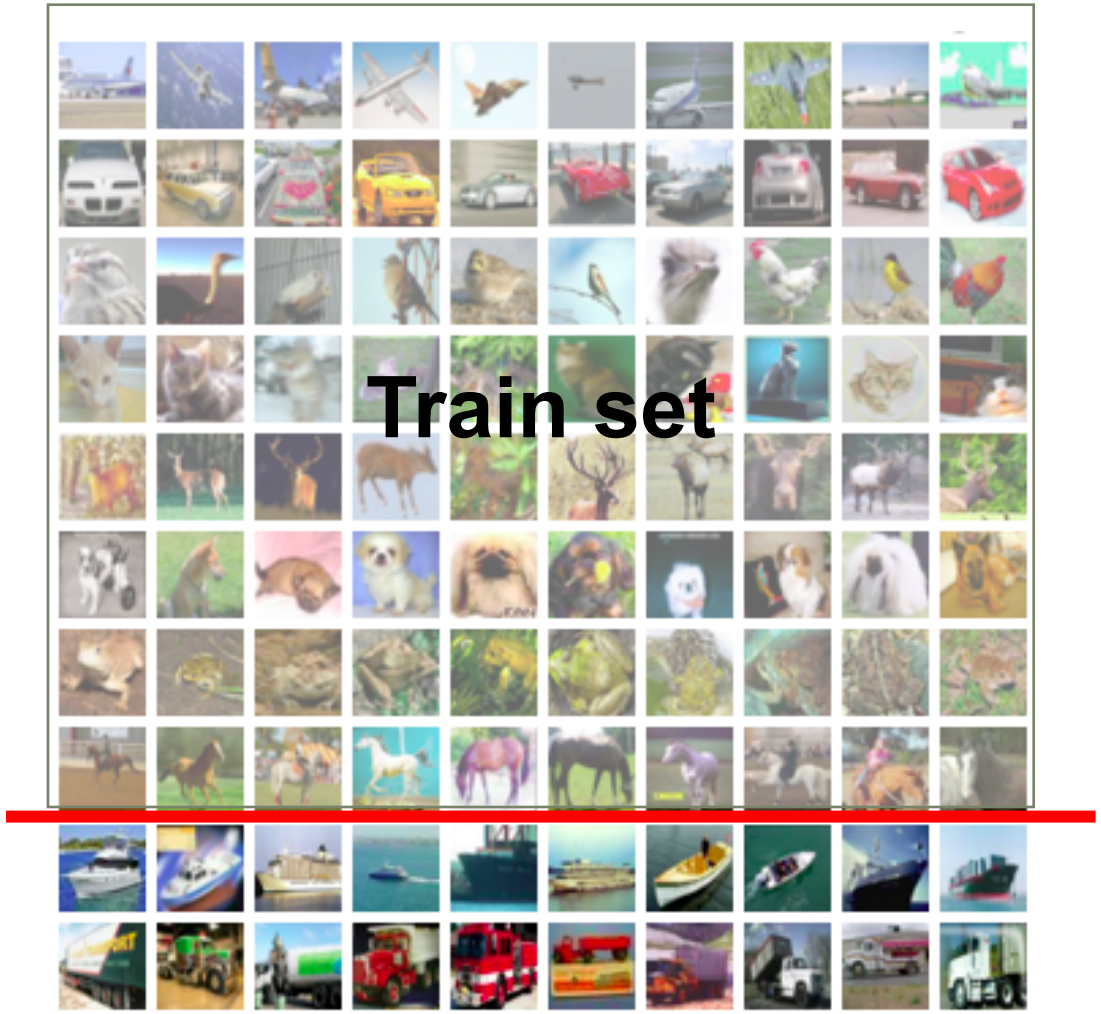
Part 2: Generalization

Training vs Testing

Good, bad and ugly minima

Machine Learning = Optimization + Generalization

1. Split dataset into training and test
2. Define model:
   - Data preprocessing, network, loss, …
3. Define training parameters:
   - Optimization algorithm, batch size, regularization,…
4. Train for N epoch on training set
5. Test on testing set.
6. If not-state of the art goto 2



Machine Learning = Optimization + Generalization

We minimize the loss on training set:

$$L(w) = \frac{1}{N} \sum_{n=1}^{N} E(f(x_n, w), y_n)$$

But we test it on different set of samples.

Generalization gap = testing loss - training loss

How we can reduce it:

- Data augmentation,
- weight decay,
- dropout,
- batch-norm,…

Let's take a look at this problem from *optimization* point of view

Generalization: Convex Case

Let's take some nice machine learning problem with smooth and convex function.

Train loss: $y(w) = (w - x_0)^2$ for x_0=1

Test loss: $y(w) = (w - x_1)^2$ where x_1=0.98

Training:

- Set initial w_0=0
- Optimization algorithm:
  - Gradient Descent
  - learning rate = fixed , $\lambda = 0.1$
  - Num of steps 10

Found solution $w_1 = 0.995$.

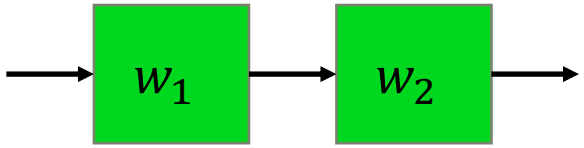
Training loss = 0.000025

Test loss = 0.000225

We found a good minima!

Generalization: Deep Learning

Change our simple quadratic problem to 2-layer MLP (for simplicity w/o non-linear part)



We call this model *deep linear network.*

Now we have complex non-convex optimization problem:

minimize $y(w_1, w_2) = (w_1 * w_2 - 1)^2$

This is non-convex functions:

- many saddle points,
- no local minima
- many global solutions: $w_1 * w_2 = 1$

Minima: good, bad and ugly

The error function

$$y(w_1, w_2) = (w_1 * w_2 - 1)^2$$

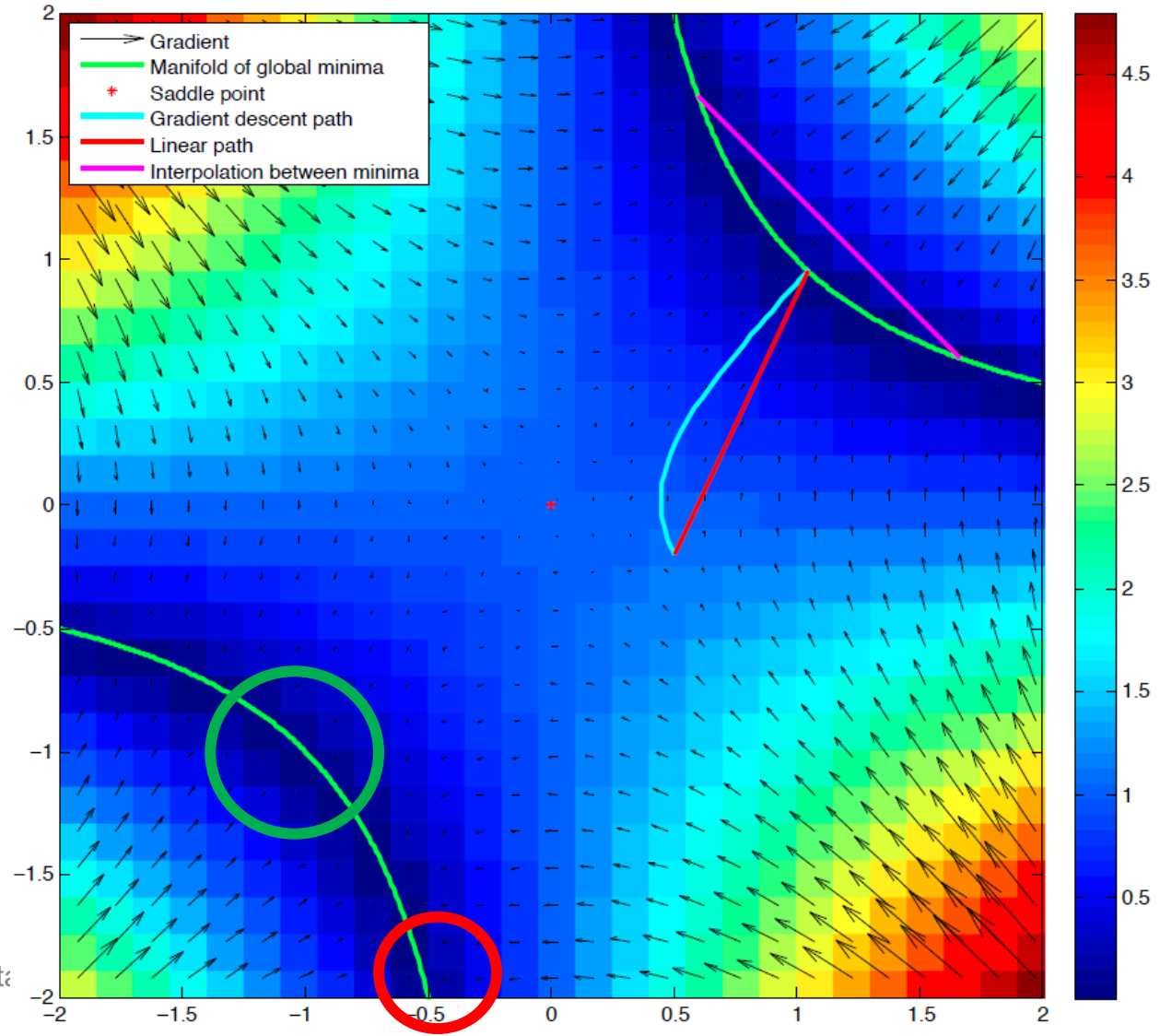
Saddle point at (0,0)

No local (“ugly”) minima

All minima are global

Some of them are better than others:

- sharp minima – bad regularization
- flat minima – good regularization



I.Goodfellow, O.Vinyals, A. Saxe,
“Qualatively Characterizing NN optimization problems

Optimization is hard, Generalization is harder

- Most popular. convolutional networks for image classification have millions of parameters; trained with SG methods they easily fit a **random labeling** of the training data
- Why large neural networks generalize so well in practice?

UNDERSTANDING DEEP LEARNING REQUIRES RE-THINKING GENERALIZATION

Chiyuan Zhang*
Massachusetts Institute of Technology
chiyuan@mit.edu

Samy Bengio
Google Brain
bengio@google.com

Moritz Hardt
Google Brain
mrtz@google.com

Benjamin Recht†
University of California, Berkeley
brecht@berkeley.edu

Oriol Vinyals
Google DeepMind
vinyals@google.com

Part 3: Advanced Topics

Ultra-fast training

Large batch training

Layer-wise Adaptive Rate Control (LARC)

Large Batch: Key to Superfast Training

Accurate, Large Minibatch SGD:
Training ImageNet in 1 Hour

Priya Goyal
Lukasz Wesolowski

Piotr
Aapo

Achieving Deep Learning Training in less than 40 Minutes on ImageNet-1K & Best Accuracy and Training Time on ImageNet-22K & Places-365 with Scale-out Intel® Xeon®/Xeon Phi™ Architectures

SURFsara CPU, HPC, HPC, machine learning, Supercomputing, Supercomputing 04 SEP 2017

Extremely Large Minibatch SGD:
Training ResNet-50 on ImageNet in 15 Minutes

Takuya Akiba
Preferred Networks, Inc.
akiba@preferred.jp

Shuji Suzuki
Preferred Networks, Inc.
ssuzuki@preferred.jp

Keisuke Fukuda
Preferred Networks, Inc.
kfukuda@preferred.jp

Ultra-Fast Training=Large Batch

Ultra-fast training= Large Scale Data-parallel Synchronous SGD

- Global batch size B = num_of_nodes * batch_per_node
- Number of iterations N = dataset_size * num_of_epochs / B

Large Batch cooking book:

1. LR ~ $\mathrm{k} * B$
2. LR ~ $\mathrm{k} * \sqrt{B}$

Why training with large LR is difficult? The weight update

$$|\Delta w(t+1)| \sim |\lambda * G(t)|$$

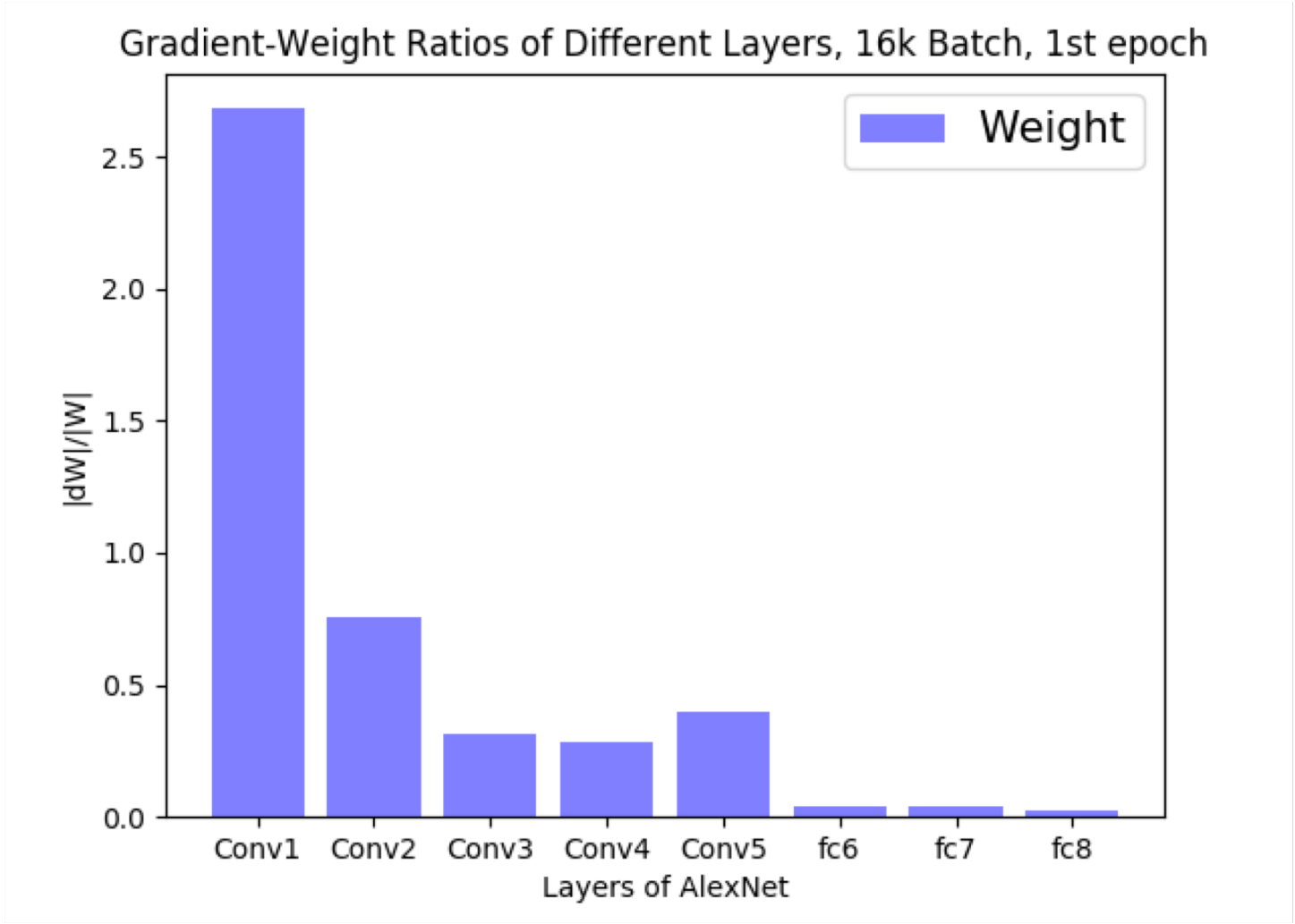
When learning rateλ is large

$$|\Delta w(t+1)| > |w(t)|$$

and training diverges.

Remedy: slow LR warm-up during initial stage to prevent divergence.

Large Batch Training and LR scaling



Alexnet: $\frac{|\nabla L(W)|}{|W|}$ for different layers in the start of training

Layer-wise Adaptive Rate Control (LARC)

Control magnitude of the layer *k* update through local learning rate λ_k:

$$\Delta w_k(t+1) = \lambda_k * G_k(w(t))$$

where:

$G_k(w(t))$– stochastic gradient of L wrt w_k,

λ_k - local LR for layer k, defined as

$$\lambda_k = min(\gamma,\ \eta * \frac{||w_k(t)||_2}{||G_k(w(t))||_2}$$

where

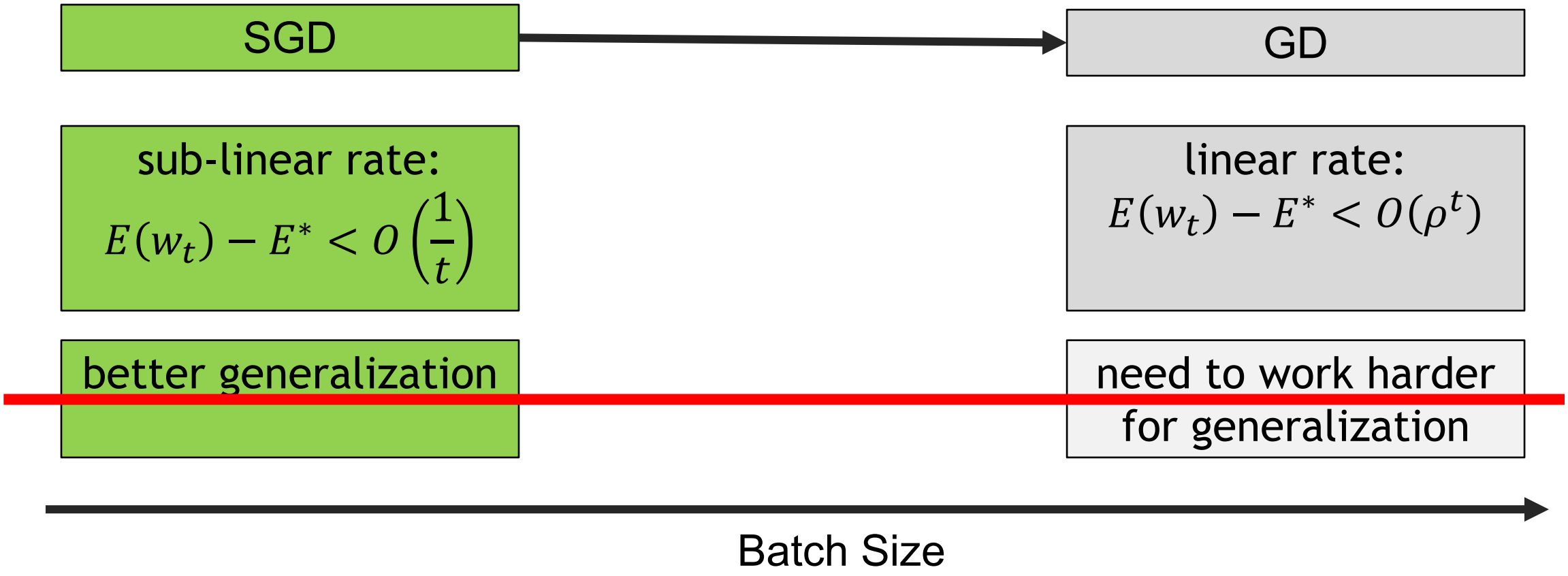
η is trust coefficient (how much we trust stochastic gradient)

γ is global LR policy (steps, exponential decay,…)

Using LARC we trained RN-50 with batch upto 64K

Large Batch: SGD vs GD

Large batch: SG becomes less noisy and close to "true" gradient



Large Batch Training: Analysis

ON LARGE-BATCH TRAINING FOR DEEP LEARNING: GENERALIZATION GAP AND SHARP MINIMA

Nitish Shirish Keskar*
Northwestern University
Evanston, IL 60208
keskar.nitish@u.northwestern.edu

Dheevatsa
Intel Corpor
Bangalore,
dheevats

DON'T DECAY THE LEARNING RATE, INCREASE THE BATCH SIZE

Samuel L. Smith*, Pieter-Jan Kindermans* & Quoc V. Le
Google Brain
{slsmith, pikinder, qvl}@google.com

Train longer, generalize better: clo
generalization gap in large batch traini
networks

Elad Hoffer[1]*, **Itay Hubara***, **Daniel Soudry**[2]
[1]Technion - Israel Institute of Technology, Haifa, Israel
[2]Columbia University, New York, New York, USA
{elad.hoffer, itayhubara, daniel.soudry}@gmail.com

Summary

- Deep Learning = Data + Model + Optimization + Generalization
- Stochastic Optimization is different from classical convex optimization
- Optimization algorithms
  - SGD with momentum
  - Adam
  - Learning rate policy
- Generalization:
  - Training loss vs testing loss
  - There are no local (“ugly”) minima, only good (“flat”) and bad (“sharp”)