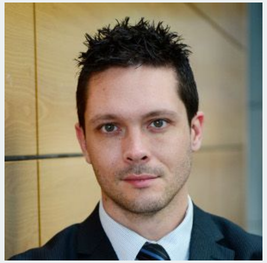
Insight
Centre for Data Analytics

Artificial Intelligence with Deep Learning

UPC School Barcelona
2019

DCU

Neural Network Optimization



Kevin McGuinness
kevin.mcguinness@dcu.ie

Assistant Professor
School of Electronic Engineering
Dublin City University

Convex optimization

A function is convex if for all α ∈ [0,1]:

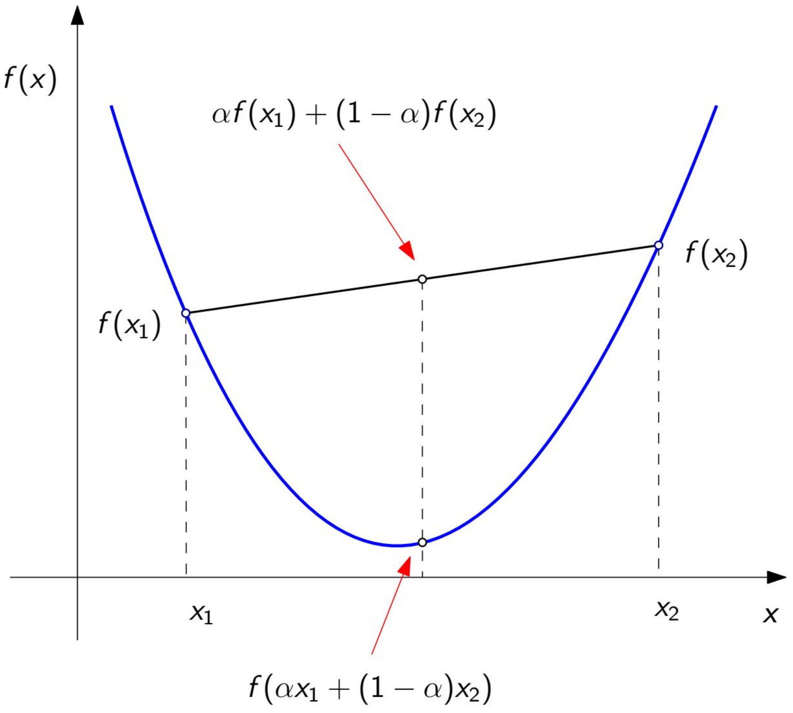
$$\alpha f(x_1) + (1-\alpha) f(x_2) \geq f(\alpha x_1 + (1-\alpha) x_2)$$

Examples

- Quadratics
- 2-norms

Properties

- All local minima have same value as the global minimum



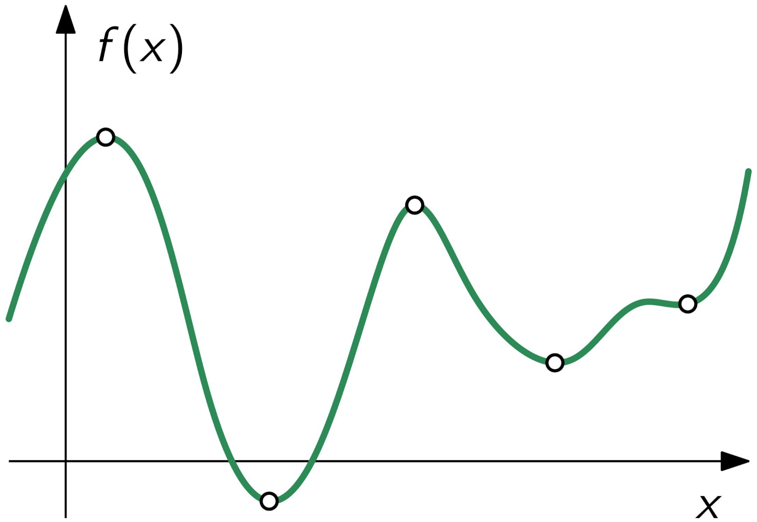
$f(x)$
$\alpha f(x_1) + (1-\alpha) f(x_2)$
$f(x_2)$
$f(x_1)$
x_1
x_2
x
$f(\alpha x_1 + (1-\alpha) x_2)$

Non-convex optimization

Objective function in deep networks is non-convex

- May be many local minima
- Plateaus: flat regions
- Saddle points

Q: Why does SGD seem to work so well for optimizing these complex non-convex functions??

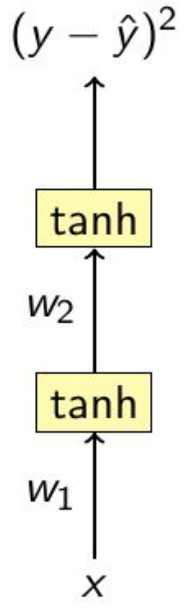


Non-convex loss surfaces

The simplest possible multi-layer perceptron:

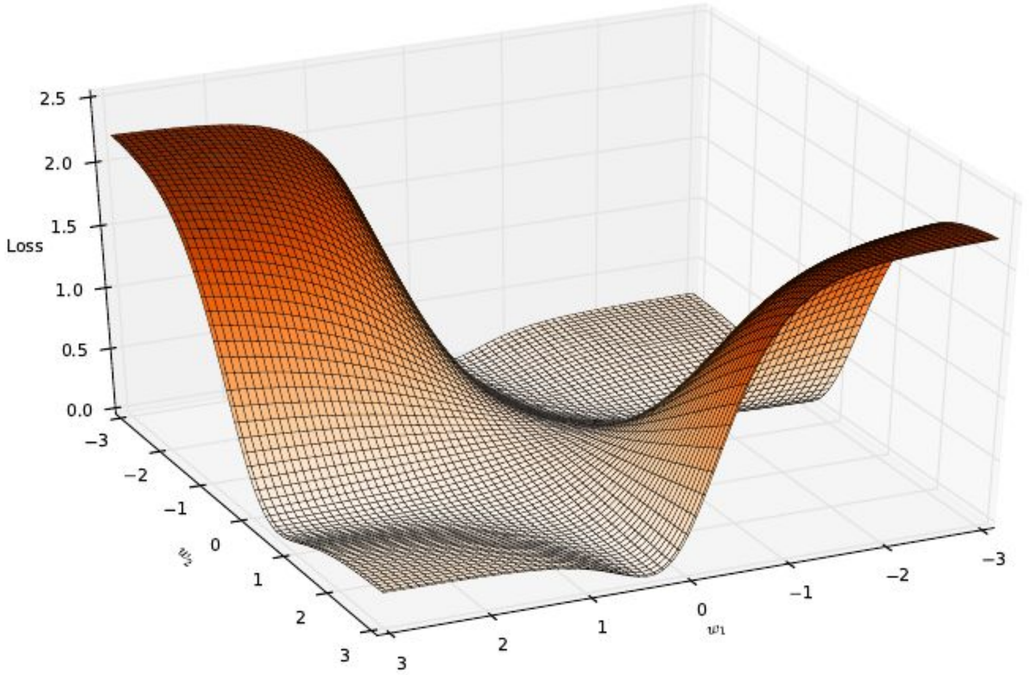
$$\hat{y} = \tanh(w_1 \tanh(w_2 x))$$

- Single scalar input x
- Single scalar output $\hat{y}$
- Only 2 parameters: w_1, w_2
- Square loss
- Single example $T = (0.5, 0.5)$



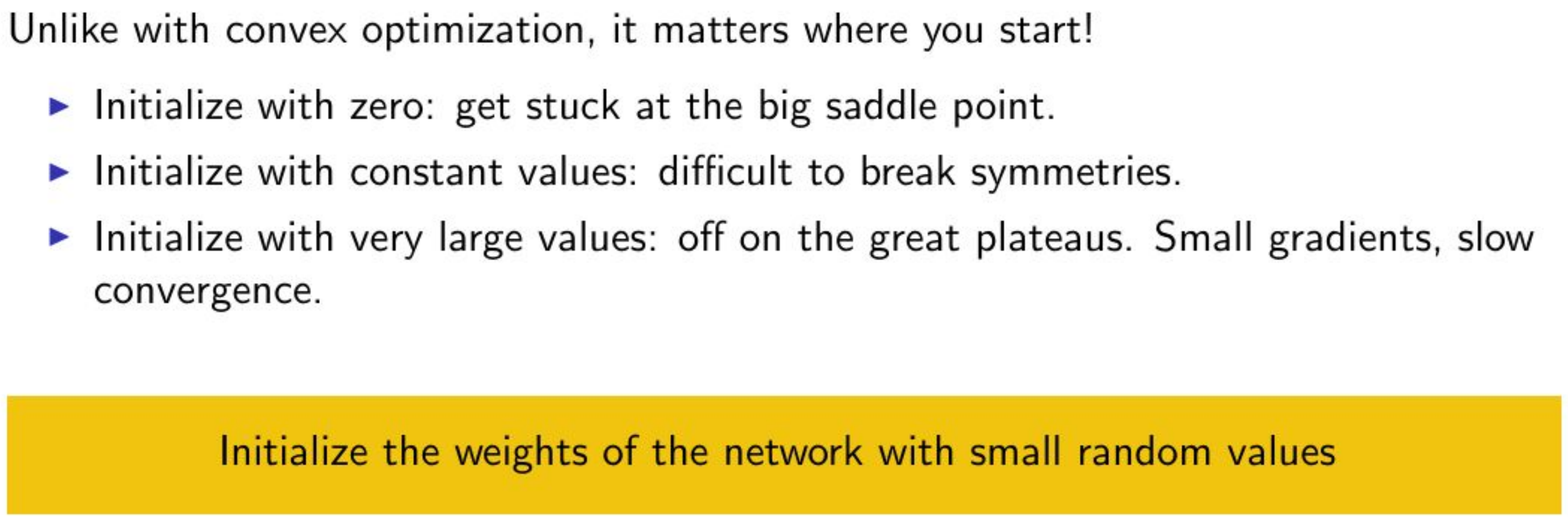
Non-convex loss surfaces

- Big saddle point near zero
- Plateaus in distance
- Symmetry in positive and negative directions
- Two ravines with equal loss



$$\mathcal{L}(w_1, w_2) = (0.5 - \tanh(w_1 \tanh(w_2\, 0.5))^2$$

Weight initialization



Unlike with convex optimization, it matters where you start!

- Initialize with zero: get stuck at the big saddle point.
- Initialize with constant values: difficult to break symmetries.
- Initialize with very large values: off on the great plateaus. Small gradients, slow convergence.

Initialize the weights of the network with small random values

Drawing weights from a normal distribution usually works well

Weight initialization

Need to pick a starting point for gradient descent: an initial set of weights

Zero is a very **bad idea**!

- Zero is a **critical point**
- Error signal will not propagate
- Gradients will be zero: no progress

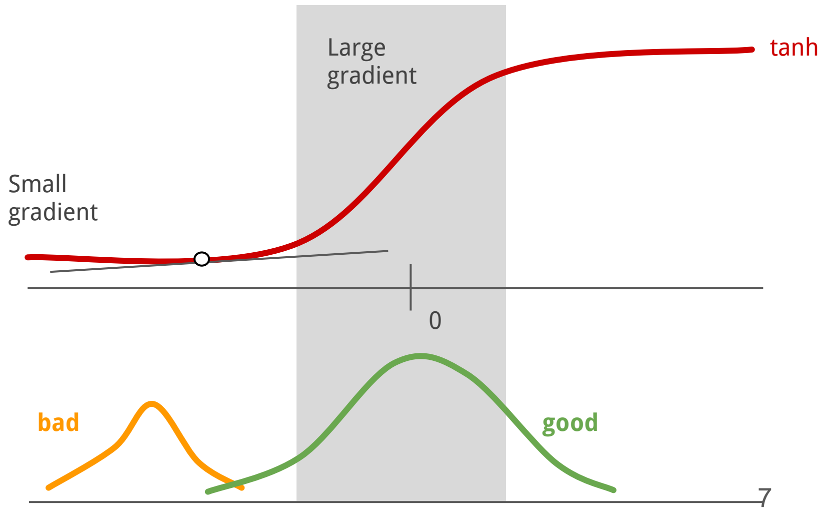
Constant value also bad idea:

- Need to break symmetry

Use **small random values:**

- E.g. zero mean Gaussian noise with constant variance

Ideally we want inputs to activation functions (e.g. sigmoid, tanh, ReLU) to be mostly **in the linear area** to allow larger gradients to propagate and converge faster.

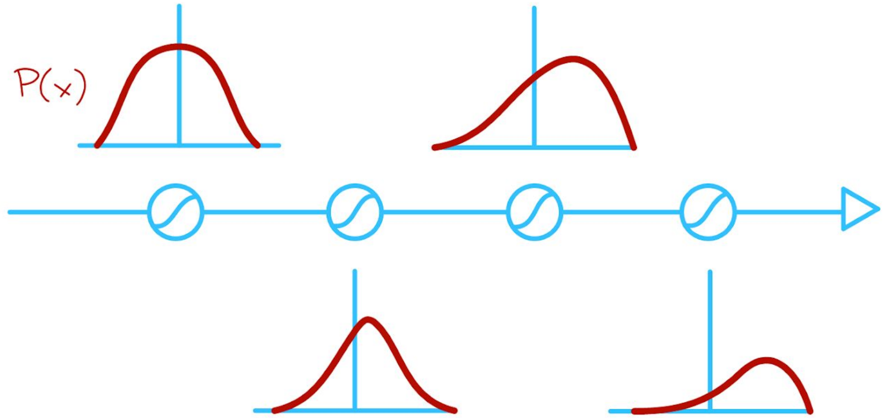


Batch normalization

As learning progresses, the distribution of layer inputs changes due to parameter updates.

This can result in most inputs being in the nonlinear regime of the activation function and slow down learning.

Batch normalization is a technique to reduce this effect.

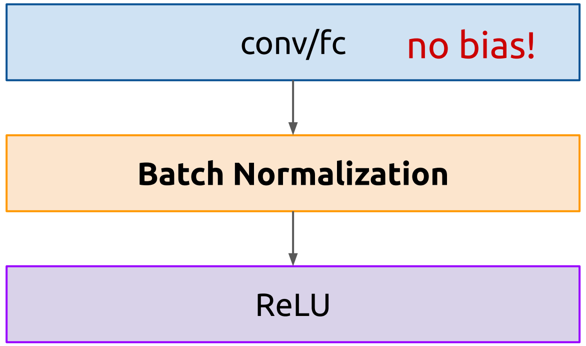


Batch normalization

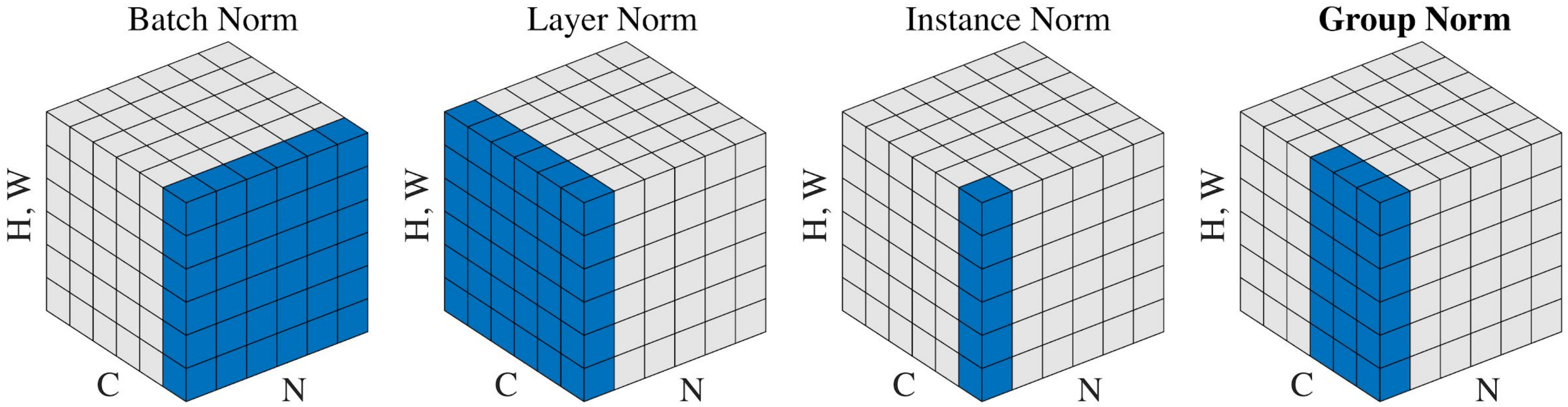
Works by **re-normalizing layer inputs to have zero mean and unit standard deviation** with respect to running batch estimates.

Also adds a **learnable scale and bias** term to allow the network to still use the nonlinearity.

Usually **allows much higher learning rates!**



Beyond batch norm: group normalization



Yuxin Wu, Kaiming He, **Group Normalization**, https://arxiv.org/abs/1803.08494

Local minima

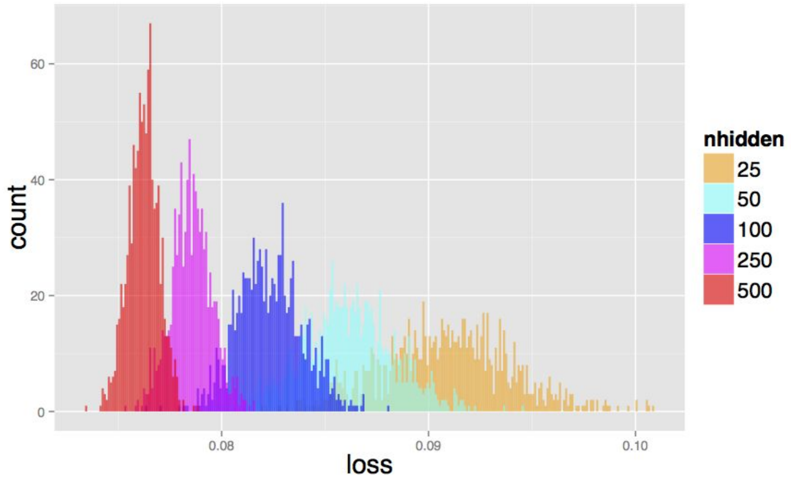
Q: Why doesn't SGD get stuck at local minima?

A: It does.

But:

- Theory and experiments suggest that for high dimensional deep models, value of loss function at most local minima is close to value of loss function at global minimum.

Most local minima are good local minima!



Value of local minima found by running SGD for 200 iterations on a simplified version of MNIST from different initial starting points. As number of parameters increases, local minima tend to cluster more tightly.

Choromanska et al. **The loss surfaces of multilayer networks**, AISTATS 2015 http://arxiv.org/abs/1412.0233

Saddle points

Q: Are there many saddle points in high-dimensional loss functions?

A: Local minima dominate in low dimensions, but **saddle points dominate in high dimensions**.

Why?

Eigenvalues of the Hessian matrix

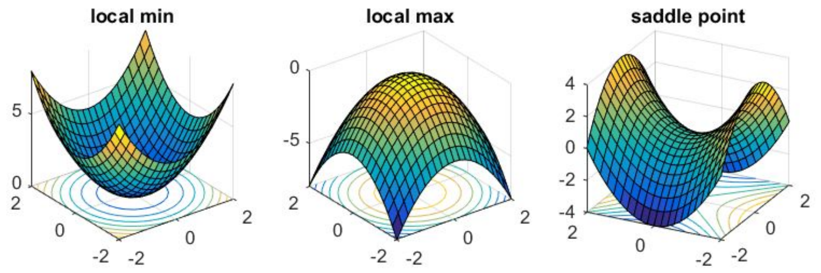
$$\mathbf{H}_{i,j} = \frac{\partial^2 f}{\partial x_i \partial x_j}.$$

Intuition

Random matrix theory: P(eigenvalue > 0) ~ 0.5

At a critical point (zero grad) in *N* dimensions we need *N* positive eigenvalues to be local min.

As *N* grows it becomes exponentially unlikely to randomly pick all eigenvalues to be positive or negative, and therefore most critical points are saddle points.



Dauphin et al. **Identifying and attacking the saddle point problem in high-dimensional non-convex optimization.** NIPS 2014
http://arxiv.org/abs/1406.2572

Saddle points

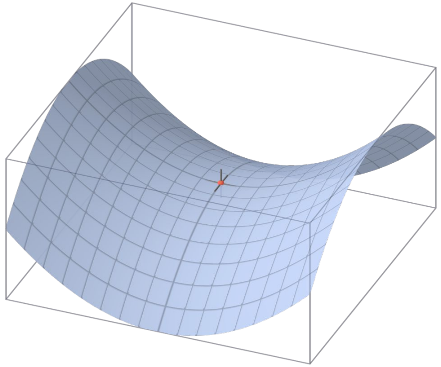
Q: Does SGD get stuck at saddle points?

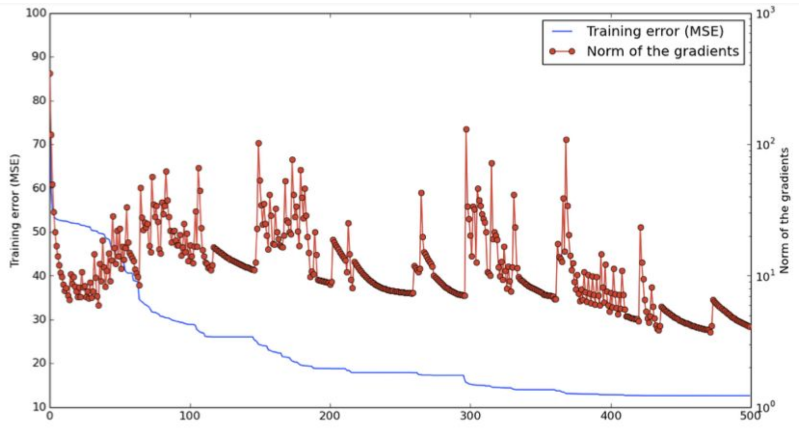
A: No, not really

Gradient descent is initially attracted to saddle points, but unless it hits the critical point exactly, it will be repelled when close.

Hitting critical point exactly is unlikely: estimated gradient of loss is stochastic

Warning: Newton's method works poorly for neural nets as it is attracted to saddle points





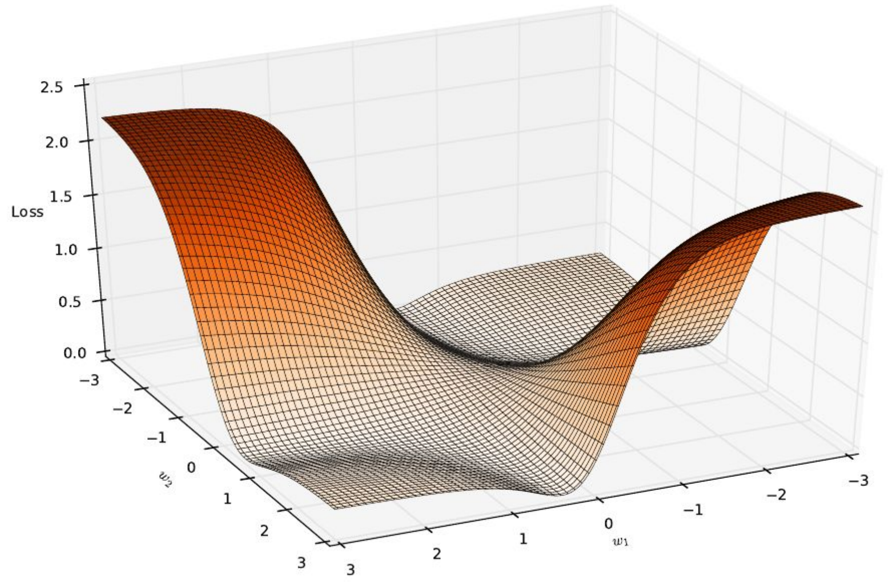
SGD tends to oscillate between slowly approaching a saddle point and quickly escaping from it

Plateaus

Regions of the weight space where loss function is mostly flat (small gradients).

Can sometimes be avoided using:

- Careful initialization
- **Non-saturating transfer functions**
- Dynamic gradient scaling
- Network design
- Loss function design



Activation functions

(AKA. transfer functions, nonlinearities, units)

Question:

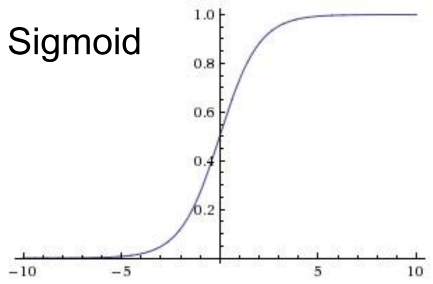
- Why do we need these nonlinearities at all? Why not just make everything linear?

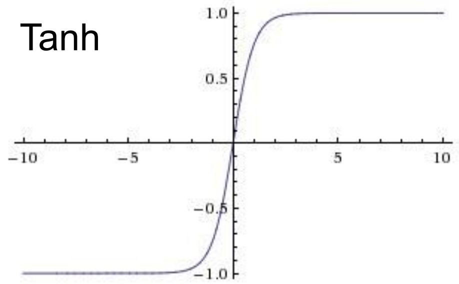
Desirable properties

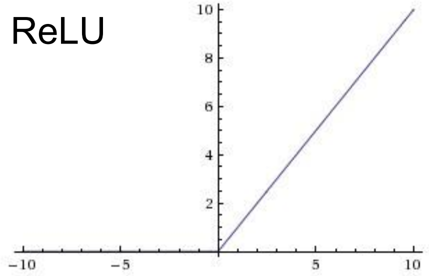
- Mostly smooth, continuous, differentiable
- Fairly linear

Common nonlinearities

- Sigmoid
- Tanh
- ReLU = max{0, x}







Problems with sigmoids

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

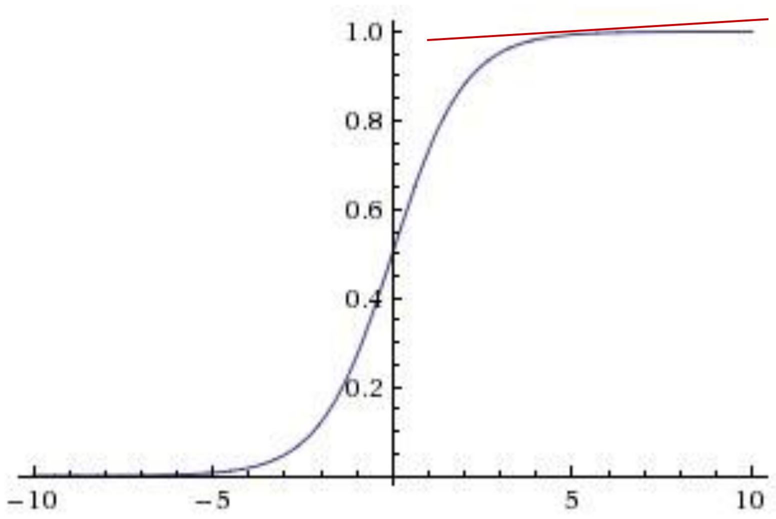
Classic NN literature uses sigmoid activation functions:

- Soft, continuous approximation of a step function
- Nice probabilistic interpretation

Avoid in practice

- Sigmoids saturate and kill gradients
- Sigmoids slow convergence
- Sigmoids are not zero-centered
- OK to use on last layer

Prefer ReLUs!



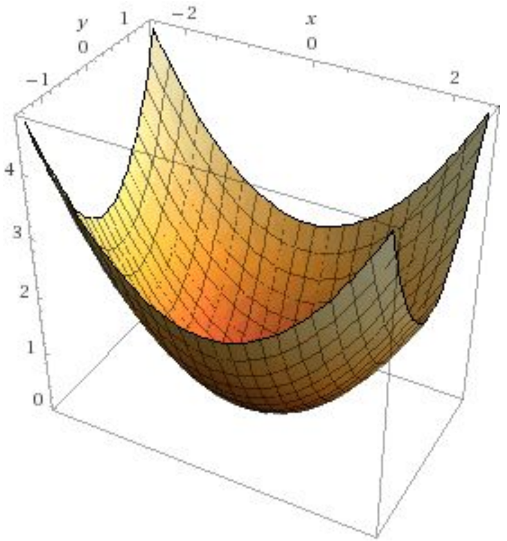
First-order optimization algorithms

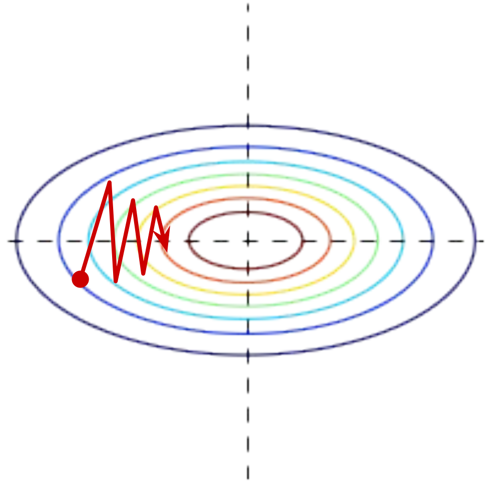
(SGD bells and whistles)

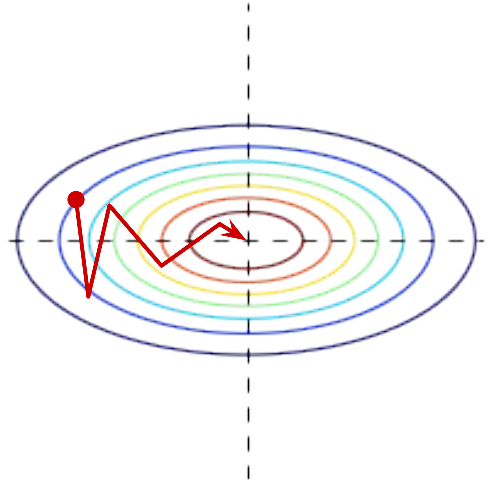
Vanilla mini-batch SGD

$$\theta_t = \theta_{t-1} - \alpha \underbrace{\nabla_\theta \mathcal{L}(\theta_{t-1})}_{\text{Evaluated on a mini-batch}}$$

Momentum







$$v_t = \gamma v_{t-1} + \alpha \nabla_\theta \mathcal{L}(\theta_{t-1})$$

$$\theta_t = \theta_{t-1} - v_t$$

2x memory for parameters!

Illustration of the effect of Momentum

Nesterov accelerated gradient (NAG)

Approximate what the parameters will be on the next time step by using the current velocity.

Update the velocity using gradient **where we predict we will be**, instead of where we are now.

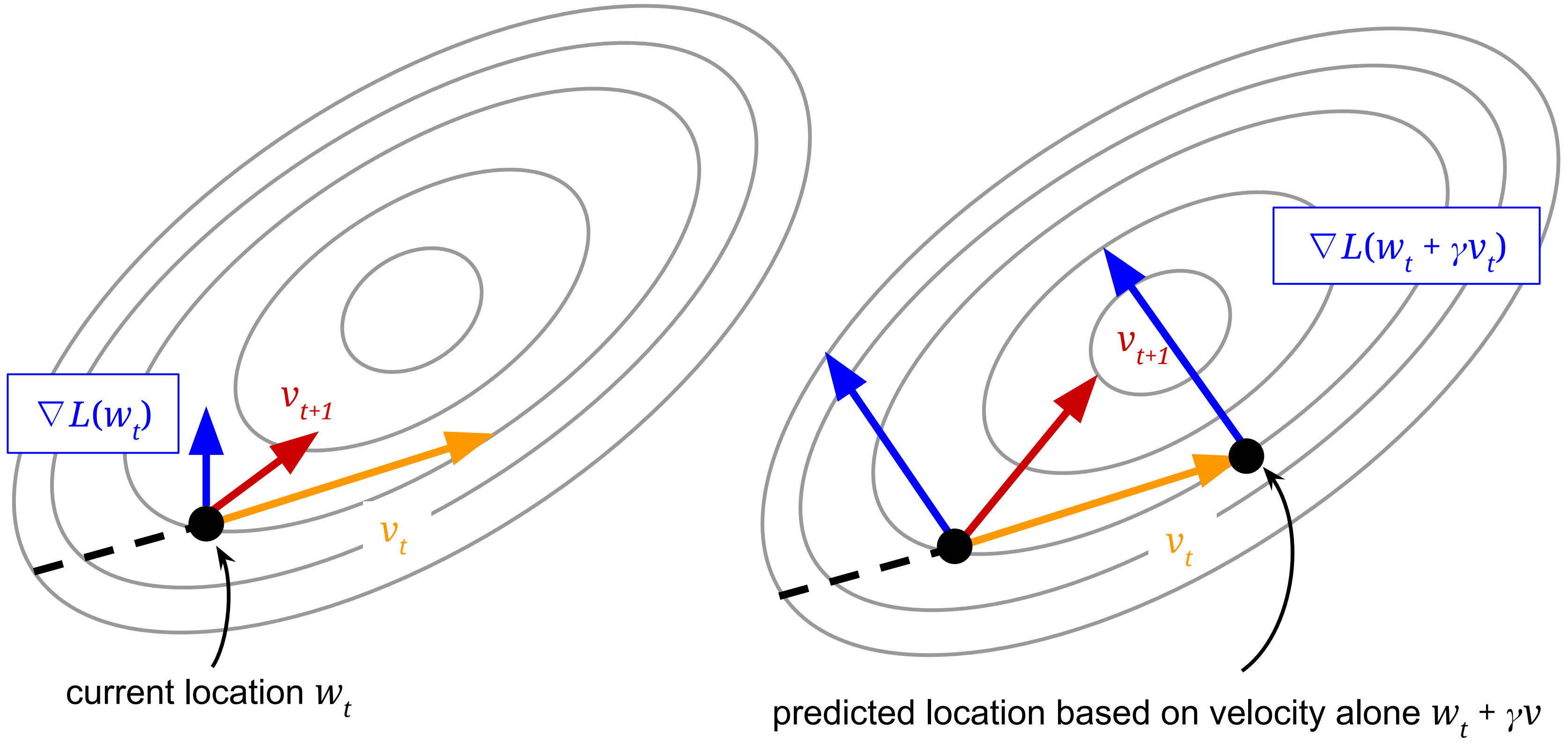
$$v_t = \gamma v_{t-1} + \alpha \nabla_\theta \mathcal{L}(\theta_{t-1} - \gamma v_{t-1})$$

$$\theta_t = \theta_{t-1} - v_t$$

What we expect the parameters to be based on momentum alone

Nesterov, Y. (1983). **A method for unconstrained convex minimization problem with the rate of convergence o(1/k2)**.

NAG illustration


$\nabla L(w_t)$
v_{t+1}
v_t
current location w_t
$\nabla L(w_t + \gamma v_t)$
v_{t+1}
v_t
predicted location based on velocity alone $w_t + \gamma v$

Adagrad

Adapts the learning rate for each of the parameters based on sizes of previous updates.

- Scales updates to be larger for parameters that are updated less
- Scales updates to be smaller for parameters that are updated more

Store **sum of squares** of gradients so far in diagonal of matrix G_t

$$G_t = \sum_{i=0}^{t} \text{diag}(\nabla \mathcal{L}(\theta)_i)^2$$

Gradient of loss at timestep *i*

Update rule: $\theta_t = \theta_{t-1} - \alpha G_t^{-\frac{1}{2}} \nabla \mathcal{L}(\theta_{t-1})$

Duchi et al. **Adaptive Subgradient Methods for Online Learning and Stochastic Optimization**. JMRL 2011

RMSProp

Modification of Adagrad to address aggressively decaying learning rate.

Instead of storing sum of squares of gradient over all time steps so far, use a **decayed moving average** of sum of squares of gradients

$$G_t = \gamma G_{t-1} + (1-\gamma)\operatorname{diag}(\nabla \mathcal{L}(\theta))^2$$

Update rule: $\theta_t = \theta_{t-1} - \alpha G_t^{-\frac{1}{2}} \nabla \mathcal{L}(\theta_{t-1})$

Geoff Hinton, Unpublished

Adam

Combines momentum and RMSProp

Keep decaying average of both first-order moment of gradient (momentum) and second-order moment (like RMSProp)

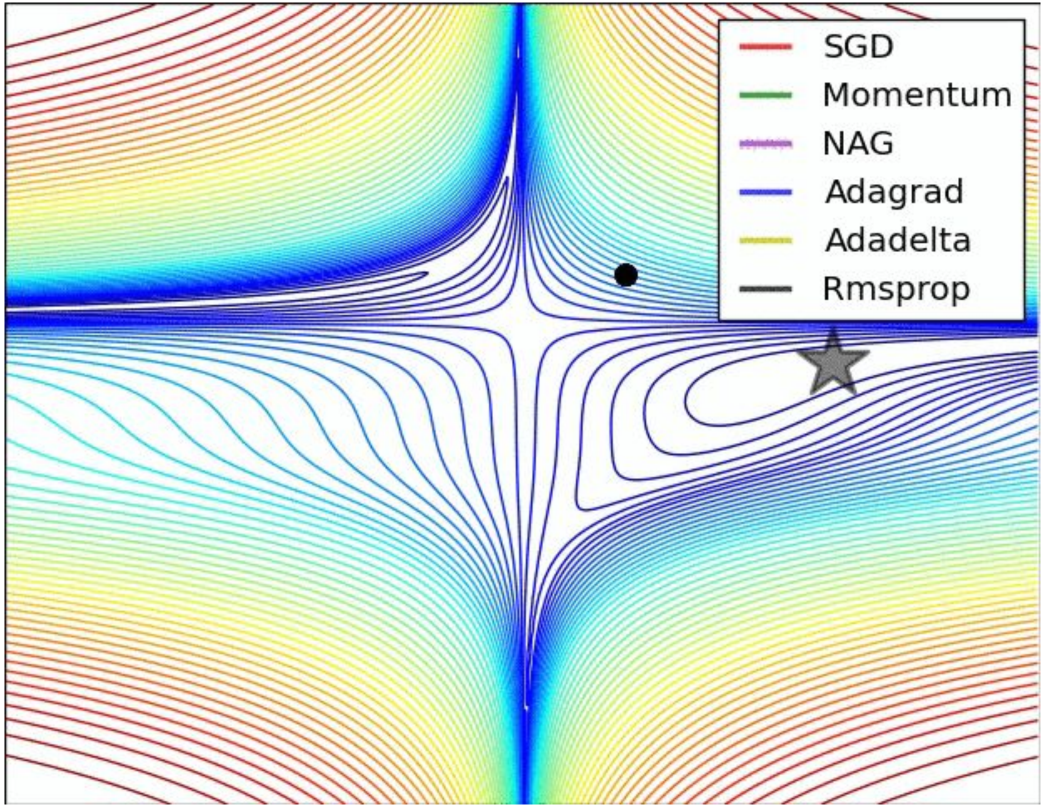
First-order: $v_t = \gamma_1 v_{t-1} + (1 - \gamma_1) \nabla \mathcal{L}(\theta_{t-1})$

Second-order: $G_t = \gamma_2 G_{t-1} + (1 - \gamma_2) \mathrm{diag}(\nabla \mathcal{L}(\theta))^2$

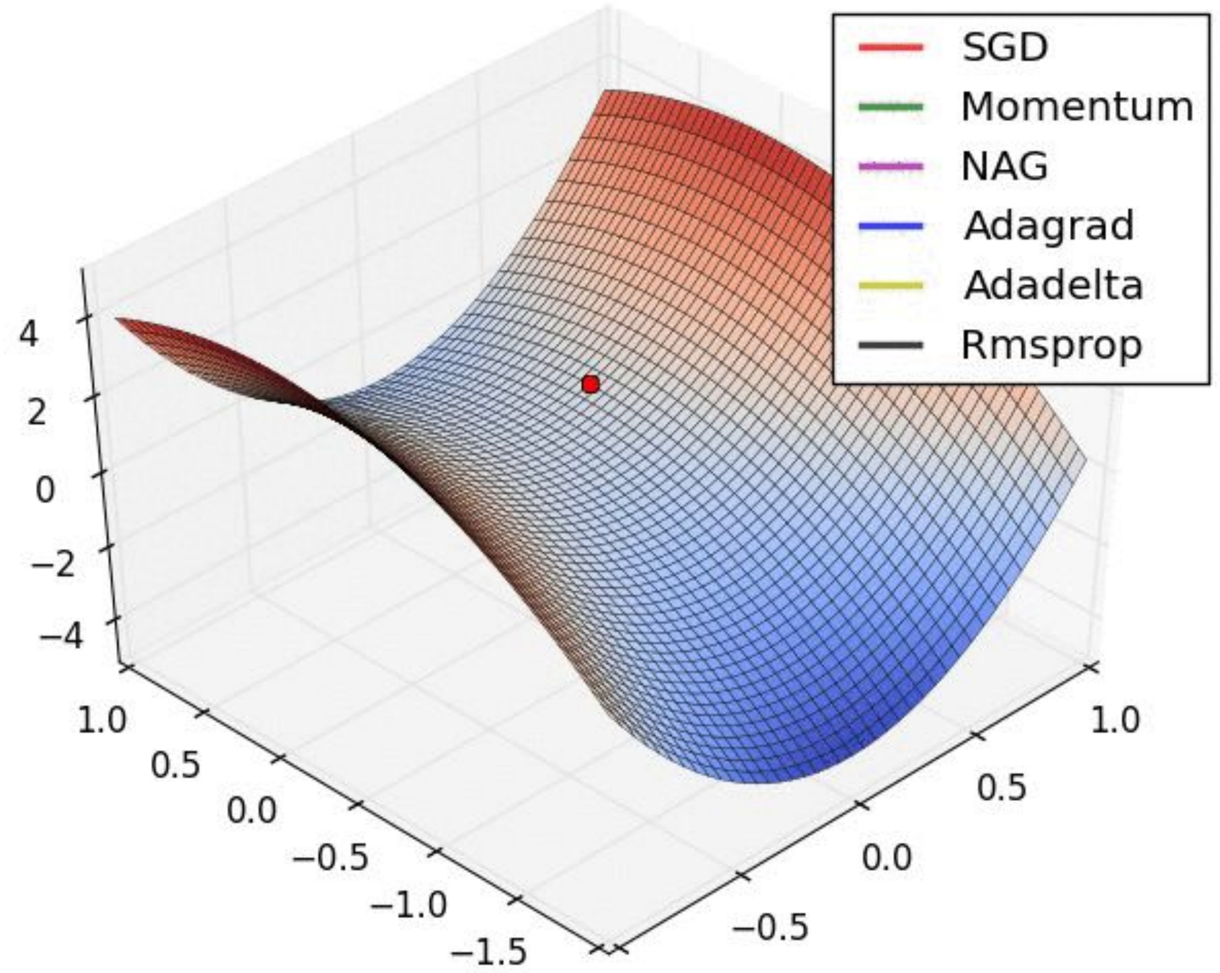
Update rule: $\theta_t = \theta_{t-1} - \alpha G_t^{-\frac{1}{2}} v_t$

3x memory!

Kingma et al. **Adam: a Method for Stochastic Optimization**. ICLR 2015



Images credit: Alec Radford.



Images credit: Alec Radford.

Fancy optimizers and large batch sizes

Fancy optimizers (e.g. Adam) and larger batch sizes often result in faster convergence.

That's always a good thing, right?

Fancy optimizers and large batch sizes

Fancy optimizers (e.g. Adam) and larger batch sizes often result in faster convergence.

That's always a good thing, right?

Not always!

They converge faster, but not necessarily to a better region of the loss surface. Often to a region with far worse generalization performance!

Loss curves for different first-order optimizers on MNIST

Fancy optimizers and large batch sizes

What's going on?

Suspected by many researchers that larger batches and fancy optimizers do not explore as much of the loss surface or converge to **sharper local minima**, which hurts generalization.

Idea is that regions that are sensitive to small changes in the parameters (sharp minima) are not as good as regions that are robust to changes.

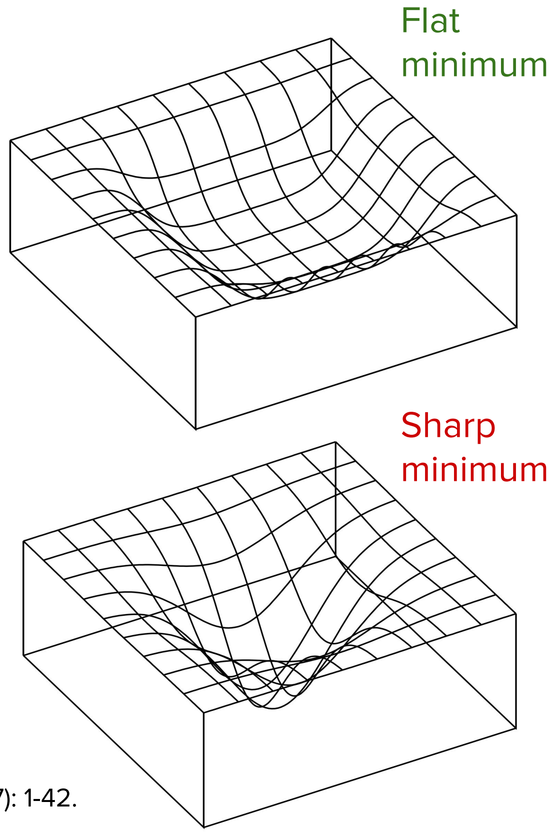
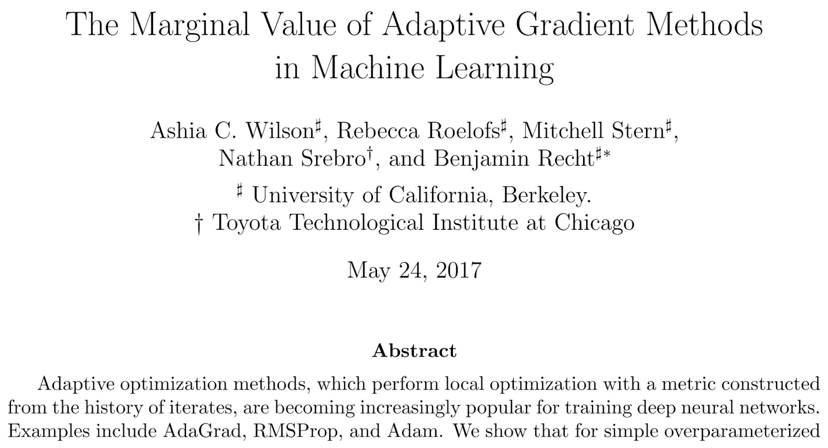


Figure credit: Hochreiter, Sepp, and Jürgen Schmidhuber. Flat minima. Neural Computation 9.1 (1997): 1-42.

Fancy optimizers and large batch sizes



The Marginal Value of Adaptive Gradient Methods in Machine Learning

Ashia C. Wilson[♯], Rebecca Roelofs[♯], Mitchell Stern[♯], Nathan Srebro[†], and Benjamin Recht[♯*]

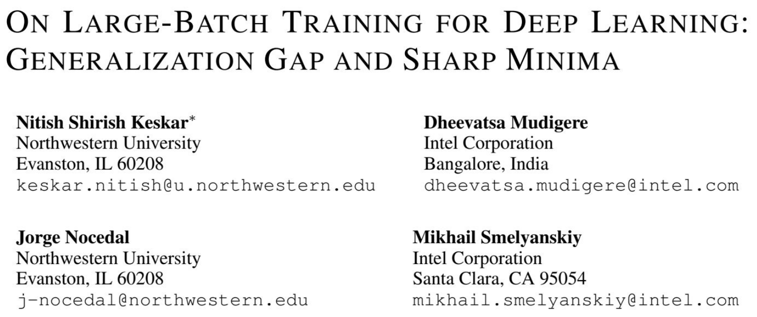
[♯] University of California, Berkeley.
[†] Toyota Technological Institute at Chicago

May 24, 2017

Abstract

Adaptive optimization methods, which perform local optimization with a metric constructed from the history of iterates, are becoming increasingly popular for training deep neural networks. Examples include AdaGrad, RMSProp, and Adam. We show that for simple overparameterized

NIPS 2017



On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima

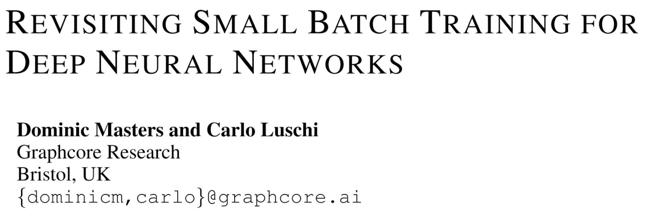
Nitish Shirish Keskar[*]
Northwestern University
Evanston, IL 60208
keskar.nitish@u.northwestern.edu

Dheevatsa Mudigere
Intel Corporation
Bangalore, India
dheevatsa.mudigere@intel.com

Jorge Nocedal
Northwestern University
Evanston, IL 60208
j-nocedal@northwestern.edu

Mikhail Smelyanskiy
Intel Corporation
Santa Clara, CA 95054
mikhail.smelyanskiy@intel.com

ICLR 2017

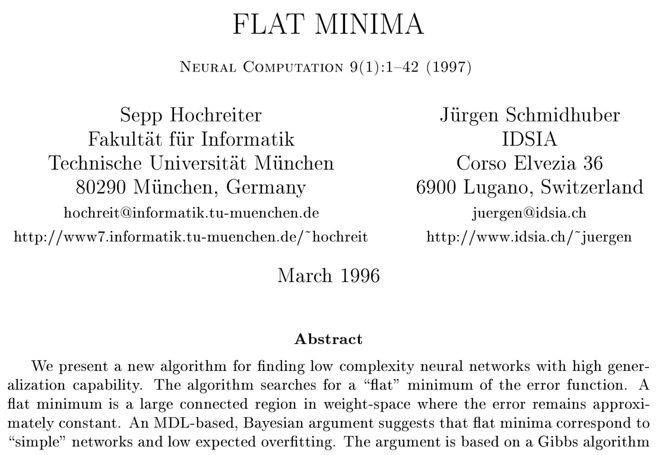


Revisiting Small Batch Training for Deep Neural Networks

Dominic Masters and Carlo Luschi
Graphcore Research
Bristol, UK
{dominicm,carlo}@graphcore.ai

arXiv 2018

Jury is still out on sharp local minima hypothesis



FLAT MINIMA

NEURAL COMPUTATION 9(1):1–42 (1997)

Sepp Hochreiter
Fakultät für Informatik
Technische Universität München
80290 München, Germany
hochreit@informatik.tu-muenchen.de
http://www7.informatik.tu-muenchen.de/~hochreit

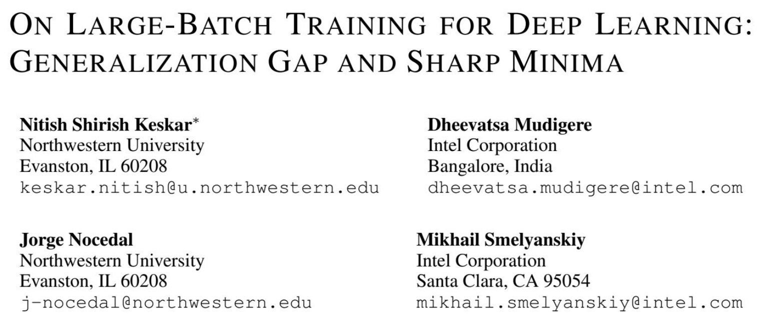
Jürgen Schmidhuber
IDSIA
Corso Elvezia 36
6900 Lugano, Switzerland
juergen@idsia.ch
http://www.idsia.ch/~juergen

March 1996

Abstract

We present a new algorithm for finding low complexity neural networks with high generalization capability. The algorithm searches for a "flat" minimum of the error function. A flat minimum is a large connected region in weight-space where the error remains approximately constant. An MDL-based, Bayesian argument suggests that flat minima correspond to "simple" networks and low expected overfitting. The argument is based on a Gibbs algorithm

Neural Computation 1997



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

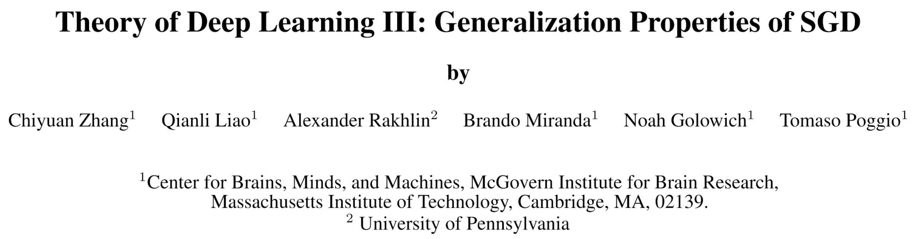
Nitish Shirish Keskar*
Northwestern University
Evanston, IL 60208
keskar.nitish@u.northwestern.edu

Dheevatsa Mudigere
Intel Corporation
Bangalore, India
dheevatsa.mudigere@intel.com

Jorge Nocedal
Northwestern University
Evanston, IL 60208
j-nocedal@northwestern.edu

Mikhail Smelyanskiy
Intel Corporation
Santa Clara, CA 95054
mikhail.smelyanskiy@intel.com

ICLR 2017



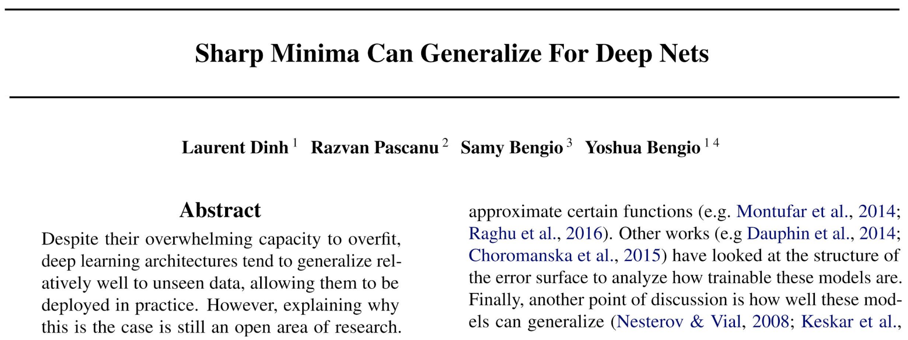
Theory of Deep Learning III: Generalization Properties of SGD

by

Chiyuan Zhang[1] Qianli Liao[1] Alexander Rakhlin[2] Brando Miranda[1] Noah Golowich[1] Tomaso Poggio[1]

[1]Center for Brains, Minds, and Machines, McGovern Institute for Brain Research, Massachusetts Institute of Technology, Cambridge, MA, 02139.
[2] University of Pennsylvania

2017



Sharp Minima Can Generalize For Deep Nets

Laurent Dinh [1] **Razvan Pascanu** [2] **Samy Bengio** [3] **Yoshua Bengio** [1 4]

Abstract

Despite their overwhelming capacity to overfit, deep learning architectures tend to generalize relatively well to unseen data, allowing them to be deployed in practice. However, explaining why this is the case is still an open area of research.

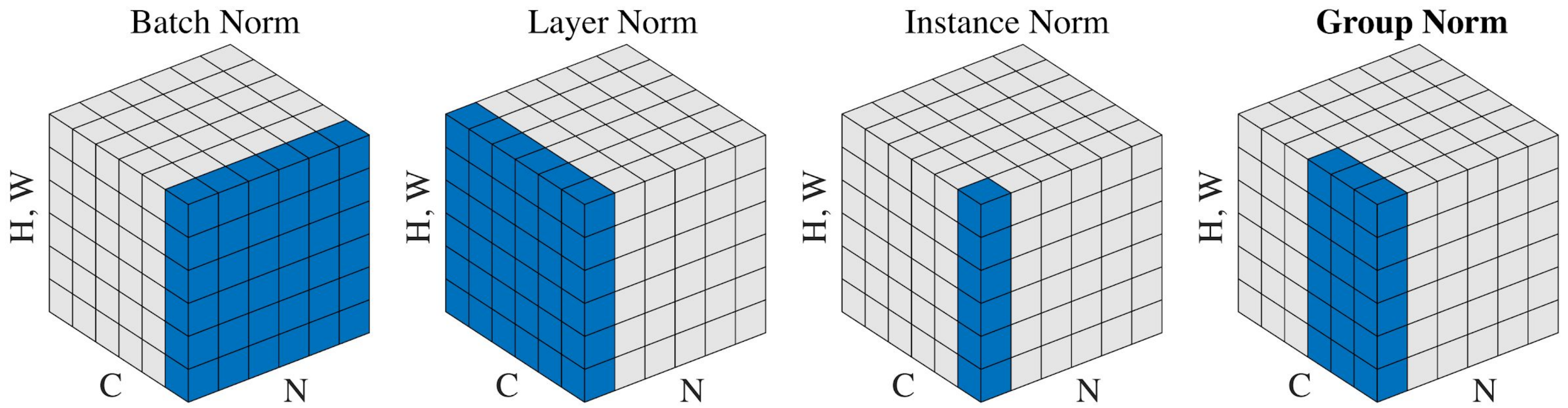
approximate certain functions (e.g. Montufar et al., 2014; Raghu et al., 2016). Other works (e.g Dauphin et al., 2014; Choromanska et al., 2015) have looked at the structure of the error surface to analyze how trainable these models are. Finally, another point of discussion is how well these models can generalize (Nesterov & Vial, 2008; Keskar et al.,

ICML 2017

Small batches and batch norm

Batch norm won't work well unless batch sizes are large enough.

Use group normalization or batch renormalization instead.



Stochastic weight averaging (SWA)

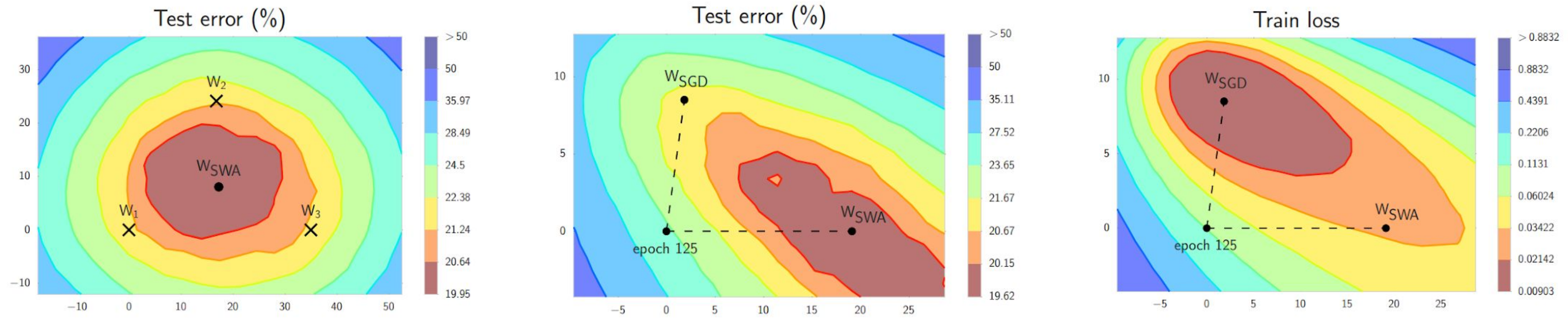


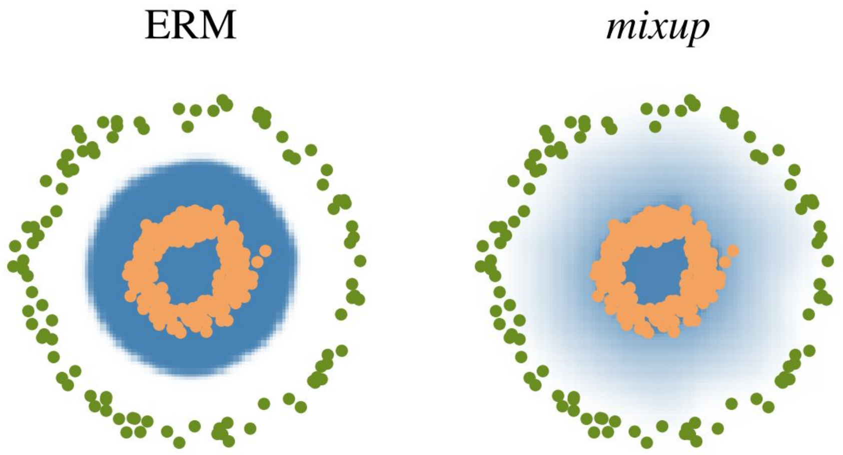
Figure 1: Illustrations of SWA and SGD with a Preactivation ResNet-164 on CIFAR-100[1]. **Left**: test error surface for three FGE samples and the corresponding SWA solution (averaging in weight space). **Middle** and **Right**: test error and train loss surfaces showing the weights proposed by SGD (at convergence) and SWA, starting from the same initialization of SGD after 125 training epochs.

$$w_{\text{SWA}} \leftarrow \frac{w_{\text{SWA}} \cdot n_{\text{models}} + w}{n_{\text{models}} + 1},$$

Izmailov et al. Averaging Weights Leads to Wider Optima and Better Generalization, UAI 2018

Mixup

Can be thought of as an **augmentation** or **regularization** technique



```
# y1, y2 should be one-hot vectors
for (x1, y1), (x2, y2) in zip(loader1, loader2):
    lam = numpy.random.beta(alpha, alpha)
    x = Variable(lam * x1 + (1. - lam) * x2)
    y = Variable(lam * y1 + (1. - lam) * y2)
    optimizer.zero_grad()
    loss(net(x), y).backward()
    optimizer.step()
```

Zhang et al., mixup: Beyond Empirical Risk Minimization, ICLR 2018

Summary

- Non-convex optimization means local minima and saddle points
- In high dimensions, there are many more saddle points than local optima
- Saddle points attract, but usually SGD can escape
- Choosing a good learning rate is critical
- Weight initialization is key to ensuring gradients propagate nicely (also batch normalization)
- Several SGD extensions that can help improve convergence
- Faster convergence is not always a good thing!

Questions?