#DLUPC

Day 4 Lecture 1

Optimization for neural network training

Verónica Vilaplana
veronica.vilaplana@upc.edu

Associate Professor
Universitat Politecnica de Catalunya
Technical University of Catalonia

Index

- **Difference between learning and pure optimization**
  - Expected and empirical risk
  - Surrogate loss functions and early stopping
  - Batch and mini-batch algorithms

- **Challenges**
  - Local minima
  - Saddle points and other flat regions
  - Cliffs and exploding gradients

- **Practical algorithms**
  - Stochastic Gradient Descent
  - Momentum
  - Nesterov Momentum
  - Learning rate
  - Adaptive learning rates: adaGrad, RMSProp, Adam

Differences between learning and pure optimization

Optimization for NN training

- **Goal:** Find the parameters that minimize the **expected risk (generalization error)**

$$J(\theta) = \mathrm{E}_{(x,y)\sim p_{data}} L(f_\theta(x), y)$$

  - x input, $f_\theta(x)$ predicted output, y target output, E expectation
  - p_{data} **true** (**unknown**) data distribution, **L loss function (how wrong predictions are)**

- But we only have a training set of samples: we minimize the **empirical risk**, average loss on a finite dataset D

$$J(\theta) = \frac{1}{|D|} \sum_{(x^{(i)},y^{(i)})\in D} L(f(x^{(i)},\theta), y^{(i)})$$

where $\hat{p}_{data}$ is the empirical distribution, $|D|$ is the number of examples in D

Surrogate loss

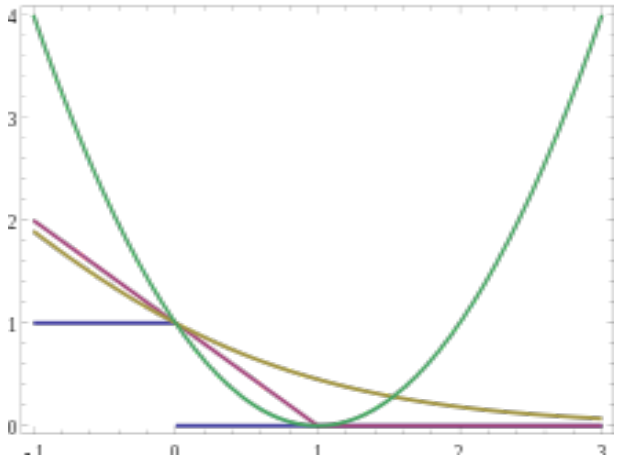
- Often minimizing the real loss is intractable (can't be used with gradient descent)
  - e.g. 0-1 loss $L(f(x),y) = I_{(f(x)\neq y)}$

- We minimize a surrogate loss instead
  - e.g. for the 0-1 loss

hinge $L(f(x),y) = \max(0,1-yf(x))$

square $L(f(x),y) = (1-yf(x))^2$

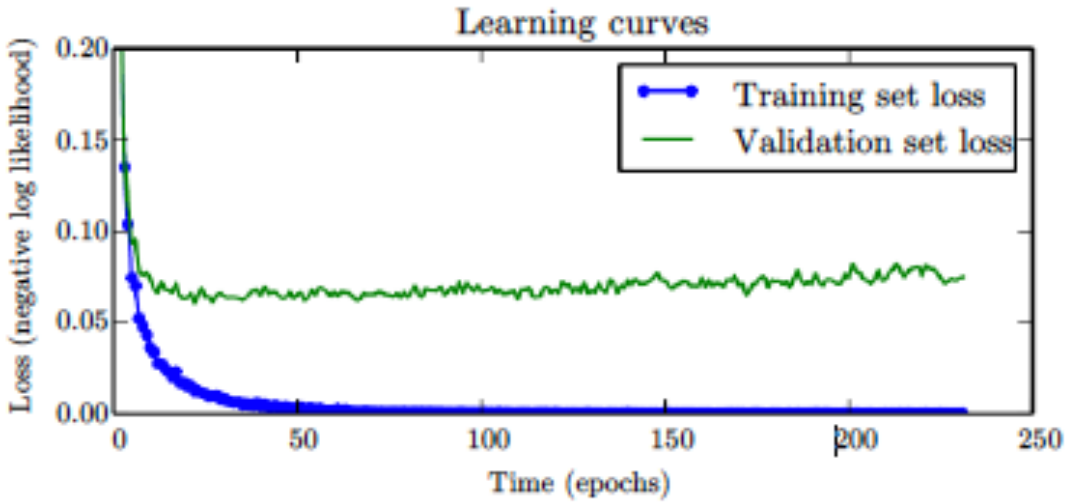
logistic $L(f(x),y) = \log(1+e^{-yf(x)})$



0-1 loss (blue) and surrogate losses (square, hinge, logistic)

Early stopping

- Training algorithms usually **do not halt at a local minimum**
- Convergence criterion based on early stopping:
  - **based on the true underlying loss** (ex 0-1 loss) **measured on a validation set**
  - # training steps = hyperparameter controlling the effective capacity of the model
  - simple, effective, must keep a copy of the best parameters



Training error decreases steadily
Validation error begins to increase

Return parameters at point with lowest validation error

Batch and mini-batch algorithms

- In most optimization methods used in ML the objective function decomposes as a sum over a training set (x^i,y^i)
- Gradient descent: $$\nabla_\theta J(\theta) = \frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)};\theta), y^{(i)})$$

- Using the complete training set can be very expensive, training set may be redundant: **use a subset of the training set**
- How many samples in each update step?
  - **Deterministic or batch** gradient methods: process **all training samples** in a large batch
  - **Stochastic** methods: use **a single example** at a time
    - online methods: samples are drawn from a stream of continually created samples
  - **Mini-batch stochastic** methods: use **several (not all)** samples

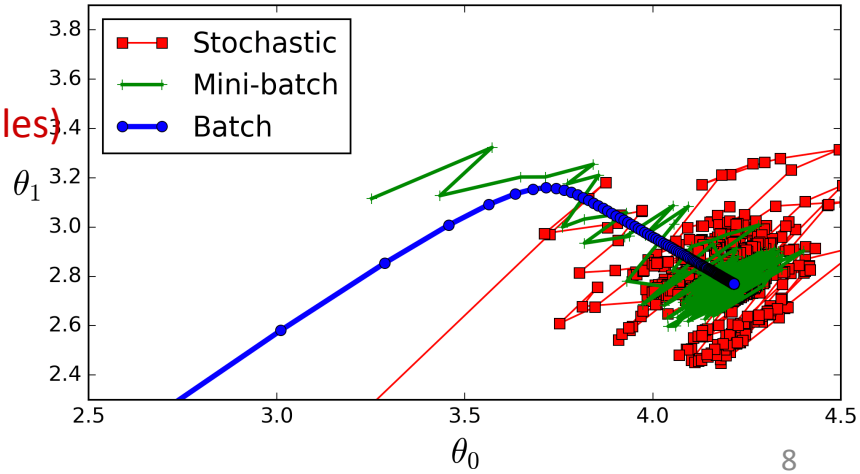
Batch and mini-batch algorithms

Mini-batch size?

- Larger batches: more accurate estimate of the gradient but less than linear return
- Smaller batches provide noisier gradient estimates
- Small batches may offer a regularizing effect (add noise)
  - but may require small learning rate
  - may increase number of steps for convergence

Minbatches should be selected randomly (shuffle samples)

- unbiased estimate of gradients

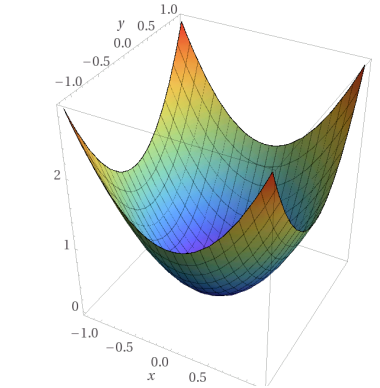


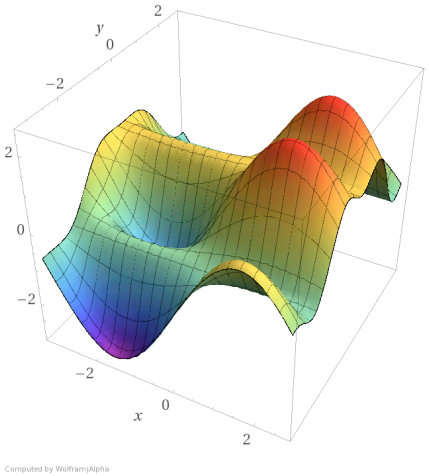
Challenges in NN optimization

Local minima

- Convex optimization
  - any local minimum is a global minimum
  - there are several opt. algorithms (polynomial-time)

- Non-convex optimization
  - **objective function in deep networks is non-convex**
  - deep models may have several local minima
  - but this is not necessarily a major problem!



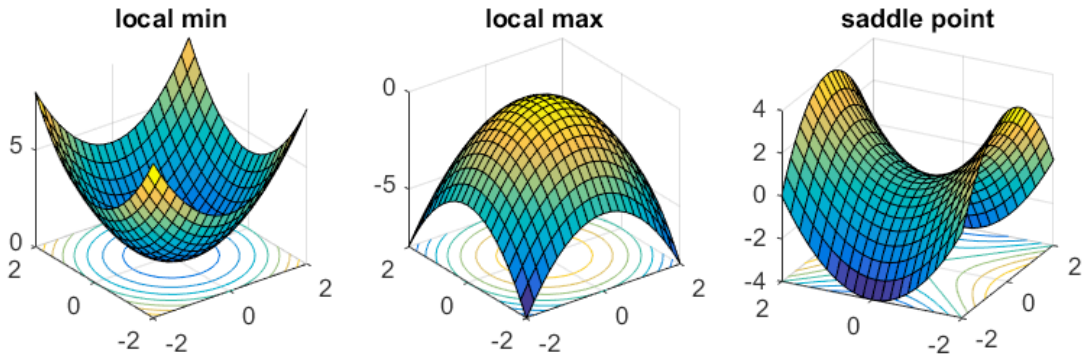


Local minima and saddle points

- Critical points:

$$f : \mathbb{R}^n \to \mathbb{R}$$

$$\nabla_x f(x) = 0$$



- For high dimensional loss functions, local minima are rare compared to saddle points
- Hessian matrix:

  real, symmetric

  eigenvector/eigenvalue decomposition: **curvature**

$$H_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}$$

- Intuition: eigenvalues of the Hessian matrix
  - local minimum/maximum: all positive / all negative eigenvalues: exponentially unlikely as n grows
  - saddle points: both positive and negative eigenvalues

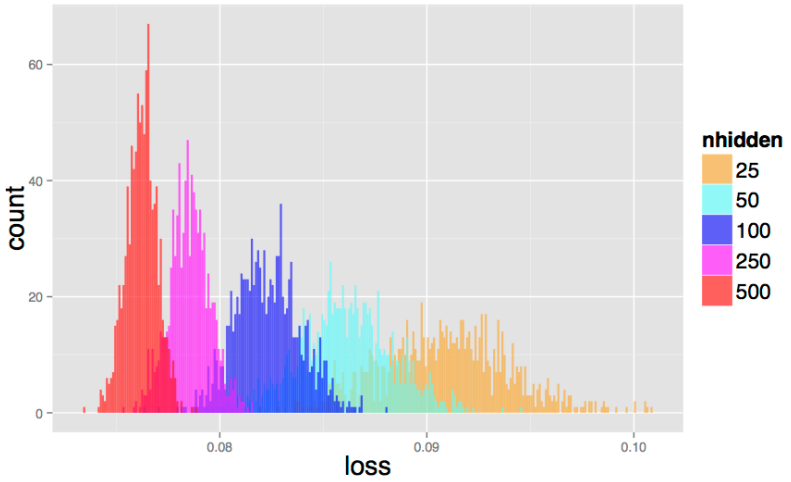
Dauphin et al. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. NIPS 2014

Local minima

- For many random functions local minima are **more likely to have low cost than high cost**.

- It is believed that for many problems including learning deep nets, almost all local minimum have very similar function value to the global optimum

- Finding a local minimum is good enough



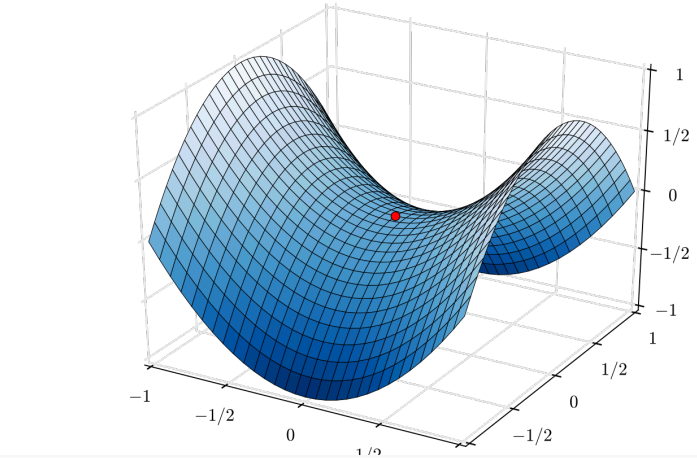
Value of local minima found by running SGD for 200 iterations on a simplified version of MNIST from different initial starting points. As number of parameters increases, local minima tend to cluster more tightly.

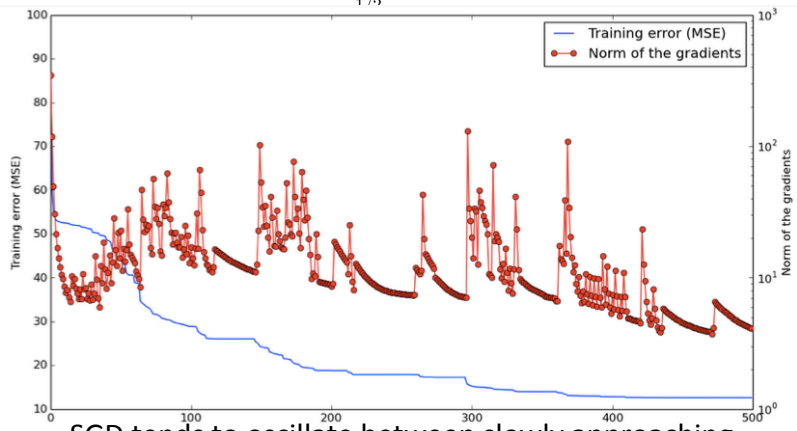
Dauphin et al. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. NIPS 2014

Saddle points

- How to escape from saddle points?

- SGD:
  - initially attracted to saddle points, but unless exact hit, it will be repelled when close
  - hitting critical point exactly is unlikely (estimated gradient is noisy)
  - **saddle points are very *unstable*: noise (stochastic gradient descent) helps convergence, trajectory escapes quickly**





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

S. Credit: K.McGuinness

Algorithms

Stochastic Gradient Descent (SGD)

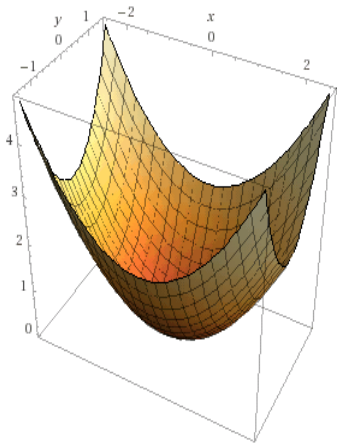
- Most used algorithm for deep learning
- Do not confuse with deterministic gradient descent: **stochastic uses mini-batches**

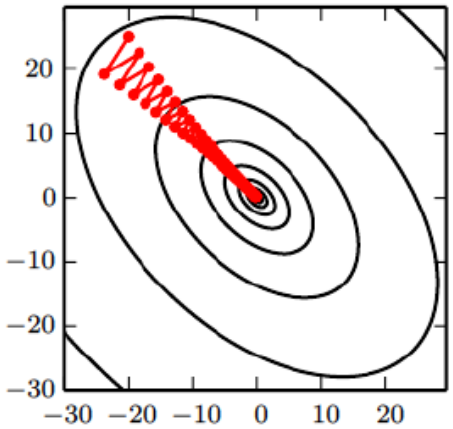
Algorithm

- **Require:** Learning rate α, initial parameter θ
- **while** stopping criterion not met **do**
  - sample a minibatch of m examples from the training set $\{x^{(i)}\}_{i=1\ldots m}$ with $\{y^{(i)}\}_{i=1\ldots m}$ corresponding targets
  - compute gradient estimate $\hat{g} \leftarrow +\frac{1}{m}\nabla_\theta \sum_i L(f(x^{(i)};\theta), y^{(i)})$
  - apply update $\theta \leftarrow \theta - \alpha\hat{g}$
- end while

Problems with SGD

- SGD can be very slow.
- Can get stuck in local minimum or saddle points
- If the loss changes quickly in one direction and slowly in another, SGD slow progress along shallow dimension, jitter along steep direction





Loss function has a high condition number: ratio of largest to smallest singular value of Hessian matrix is large

Momentum

- New variable v (velocity), direction and speed at which parameters move: exponentially decaying average of negative gradient

Algorithm

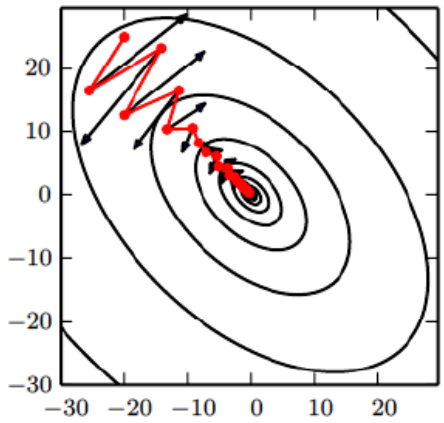
- **Require:** learning rate α, initial parameter θ, **momentum parameter λ , initial velocity v**
- **Update rule:** (g is gradient estimate)

  - compute velocity update $v \leftarrow \lambda v - \alpha g$
  - apply update $\theta \leftarrow \theta + v$

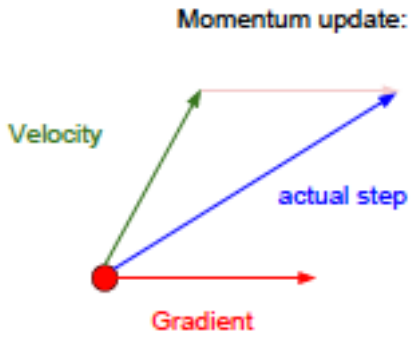
- Typical values λ=.5, .9,.99 (in [0,1})
- Read physical analogy in Deep Learning book (Goodfellow et al)

Momentum

- Momentum is designed to accelerate learning, especially for high curvature, small but consistent gradients or noisy gradients
- New variable v (velocity), direction and speed at which parameters move: exponentially decaying average of negative gradient



local minimum saddle point



Contour lines= a quadratic loss with poor conditioning of Hessian
Path (red) followed by SGD (left) and momentum (right)

Nesterov accelerated gradient (NAG)

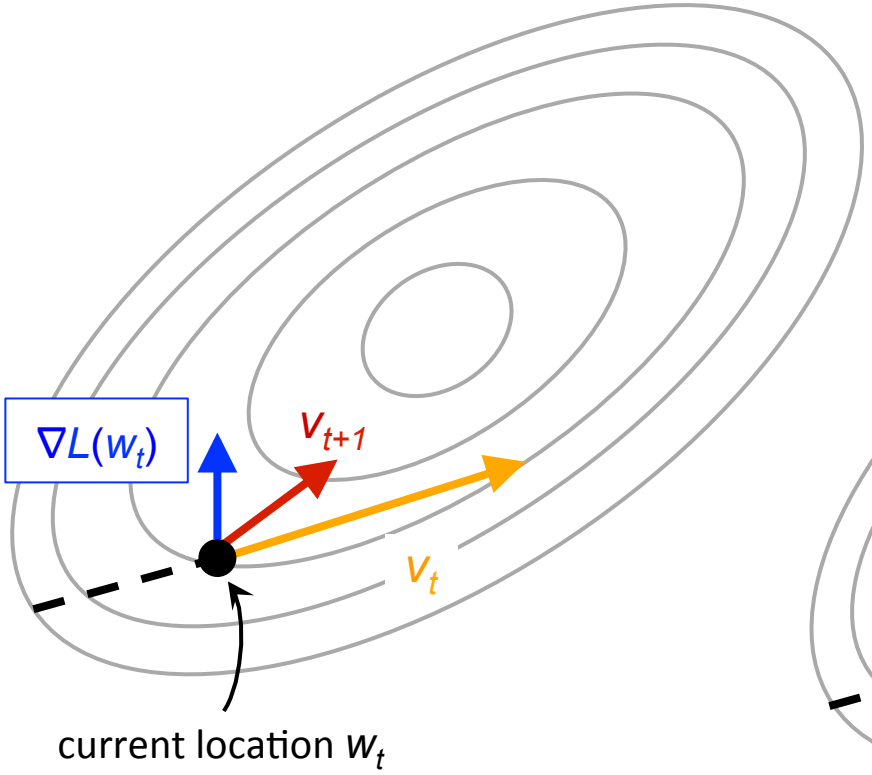
- A variant of momentum, where gradient is evaluated after current velocity is applied:
  - Approximate where the parameters will be on the next time step using current velocity
  - **Update velocity using gradient where we predict parameters will be**

Algorithm

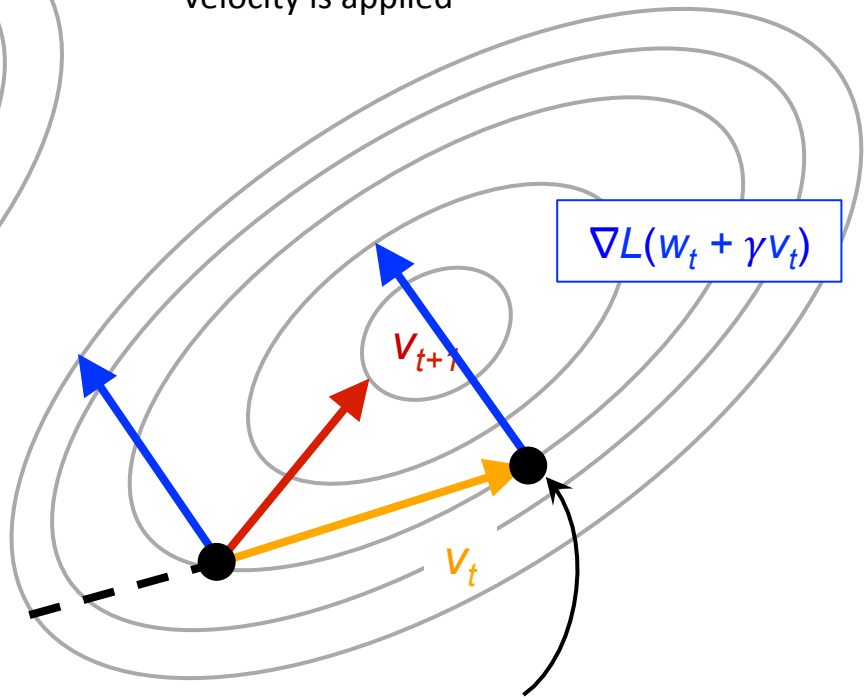
- **Require:** learning rate α, initial parameter θ, momentum parameter λ , initial velocity v
- **Update:**
  - apply interim update $\tilde{\theta} \leftarrow \theta + \lambda v$
  - compute gradient (at interim point) $g \leftarrow +\frac{1}{m}\nabla_{\tilde{\theta}} \sum_i L(f(x^{(i)};\tilde{\theta}), y^{(i)})$
  - compute velocity update $v \leftarrow \lambda v - \alpha g$
  - apply update $\theta \leftarrow \theta + v$

- Interpretation: add a correction factor to momentum

Nesterov accelerated gradient (NAG)



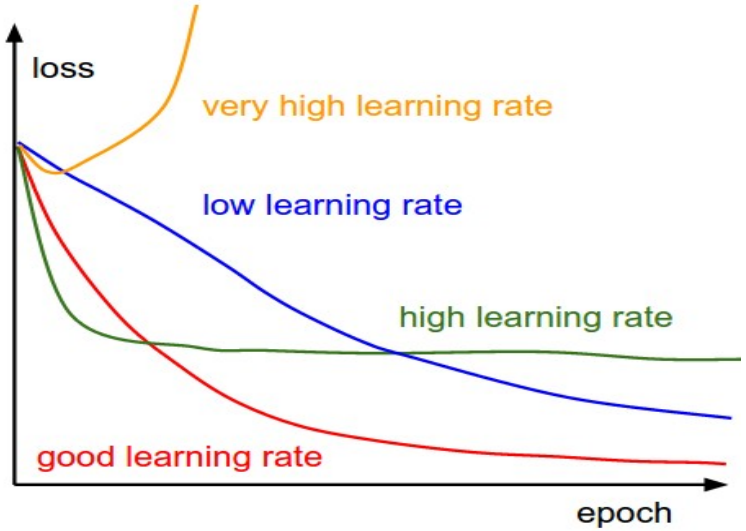
NAG: gradient is evaluated after current velocity is applied



S. Credit: K. McGuinness

SGD: learning rate

- Learning rate is a crucial parameter for SGD
  - To large: overshoots local minimum, loss increases
  - Too small: makes very slow progress, can get stuck
  - Good learning rate: makes steady progress toward local minimum



- Usually: adapt learning rate by monitoring learning curves that plot the objective function as a function of time; In practice it is necessary to gradually decrease learning rate

Adaptive learning rates

- Cost is often sensitive to some directions and insensitive to others
  - Momentum/Nesterov mitigate this issue but introduce another hyperparameter

- **Solution: Use a separate learning rate for each parameter and automatically adapt it through the course of learning**

- **Algorithms (mini-batch based)**
  - AdaGrad
  - RMSProp
  - Adam
  - RMSProp with Nesterov momentum

AdaGrad

- **Adapts the learning rate of each parameter 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
- The net effect is greater progress in the more gently sloped directions of parameter space

- **Require:** learning rate α, initial parameter θ
- **Update:**
  - **accumulate squared gradient** $r \leftarrow r + g \odot g$
  - **compute update** $\Delta\theta \leftarrow -\frac{\alpha}{\sqrt{r}} \odot g$
  - apply update $\theta \leftarrow \theta + \Delta\theta$

sum of all previous squared gradients

updates inversely proportional to the square root of the sum
(elementwise multiplication)

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

RMSProp (Root Mean Square Propagation)

- Modifies AdaGrad to perform better in non-convex surfaces, for aggressively decaying learning rates
- Changes gradient accumulation by an **exponentially decaying average** of sum of squares of gradients

- **Requires:** learning rate α, initial parameter θ, **decay rate ρ**
- **Update:**
  - **accumulate squared gradient** $r \leftarrow \rho r + (1-\rho) g \odot g$
  - compute update $\Delta\theta \leftarrow -\frac{\alpha}{\sqrt{r}} \odot g$
  - apply update $\theta \leftarrow \theta + \Delta\theta$

Geoff Hinton, Unpublished

Adam

- Combination of RMSProp and momentum, but:
  - Keep decaying average of both first-order moment of gradient (momentum) and second-order moment (RMSProp)
  - Includes bias corrections (first and second moments) to account for their initialization at origin

Update:

- **updated biased first moment estimate** $s \leftarrow \rho_1 s + (1-\rho_1) g$
- **update biased second moment** $r \leftarrow \rho_2 r + (1-\rho_2) g \odot g$
- **correct biases** $\hat{s} \leftarrow \frac{s}{1-\rho_1}$ $\hat{r} \leftarrow \frac{r}{1-\rho_2}$
- compute update $\Delta\theta \leftarrow -\alpha \frac{\hat{s}}{\sqrt{\hat{r}}}$ (operations applied elementwise)
- apply update $\theta \leftarrow \theta + \Delta\theta$

ρ1=0.9, ρ2=0.999,α=10^{-3}

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

Example: test function

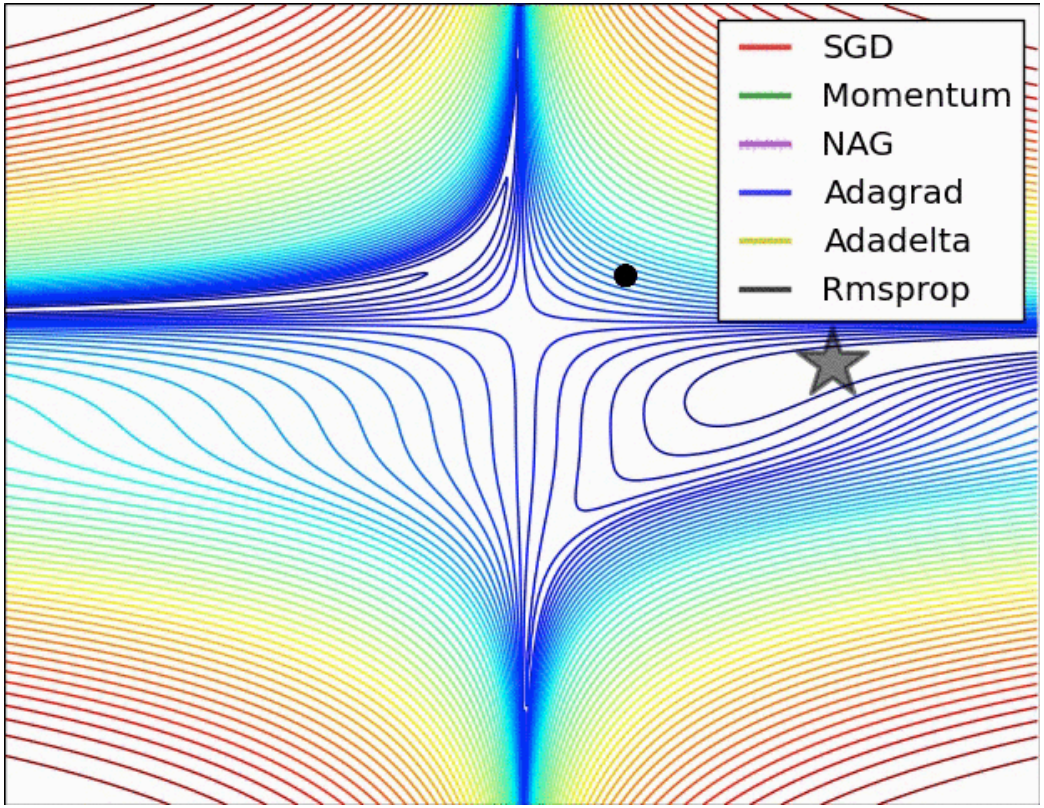
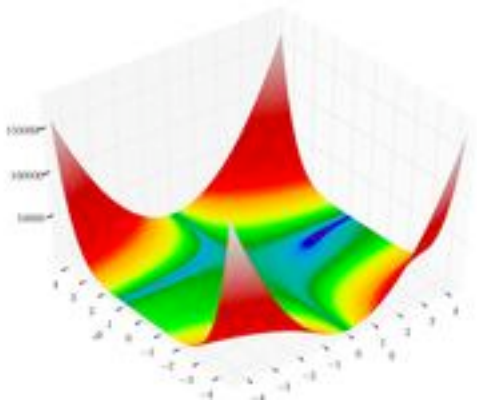


Image credit: Alec Radford.



Beale's function

$$f(\mathbf{x}) = (1.5 - x_1 + x_1 x_2)^2 + (2.25 - x_1 + x_1 x_2^2)^2 + (2.625 - x_1 + x_1 x_2^3)^2$$

Example: saddle point

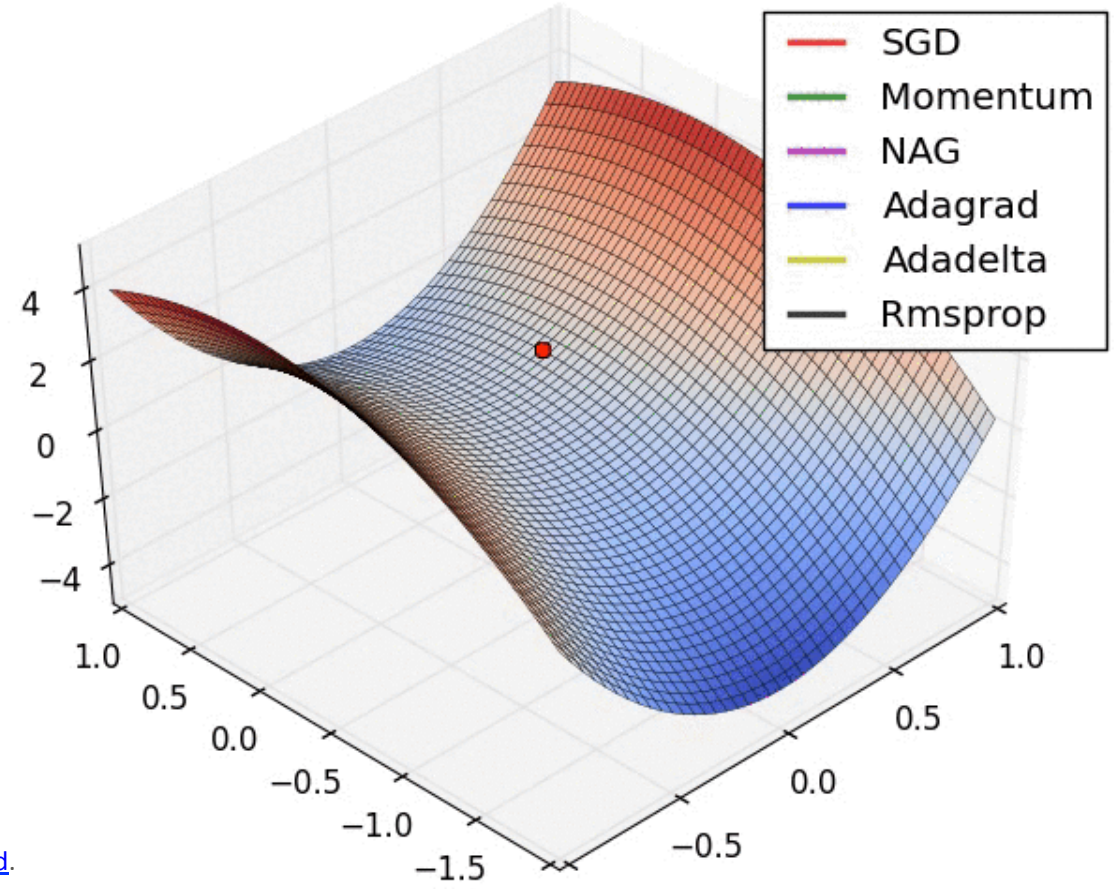


Image credit: Alec Radford.

Parameter initialization

- **Weights**
  - Can't initialize weights to 0 (gradients would be 0)
  - Can't initialize all weights to the same value (all hidden units in a layer will always behave the same; need to break symmetry)
  - **Small random number, e.g., uniform or gaussian distribution** $N(0,10^{-2})$
    - if weights start too small, the signal shrinks as it passes through each layer until it is too tiny to be useful
  - **Xavier Initialization**
    - each neuron: w = randn(n) / sqrt(n) , n inputs
  - **He initialization**
    - each neuron w = randn(n) * sqrt(2.0 /n) , n inputs
- **Biases**
  - initialize all to 0 (except for output unit for skewed distributions, 0.01 to avoid saturating RELU)
- **Alternative:** Initialize using machine learning; parameters learned by unsupervised model trained on the same inputs / trained on unrelated task

Summary

- Optimization for NN is different from pure optimization:
  - GD with mini-batches
  - early stopping
  - non-convex surface, local minima and saddle points
- Learning rate has a significant impact on model performance
- Several extensions to SGD can improve convergence
- Adaptive learning-rate methods are likely to achieve best results
  - RMSProp, Adam

Bibliograpy

- **Goodfellow, I., Bengio, Y., and A., C. (2016), Deep Learning, MIT Press.**
- Choromanska, A., Henaff, M., Mathieu, M., Arous, G. B., and LeCun, Y. (2015), The loss surfaces of multilayer networks. In AISTATS.
- Dauphin, Y. N., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., and Bengio, Y. (2014). Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In Advances in Neural Information Processing. Systems, pages 2933–2941.
- Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 12(Jul):2121–2159.
- Goodfellow, I. J., Vinyals, O., and Saxe, A. M. (2015). Qualitatively characterizing neural network optimization problems. In International Conference on Learning Representations.
- Hinton, G. (2012). Neural networks for machine learning. Coursera, video lectures
- Jacobs, R. A. (1988). Increased rates of convergence through learning rate adaptation. Neural networks, 1(4):295–307.
- Kingma, D. and Ba, J. (2014)- Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.
- Saxe, A. M., McClelland, J. L., and Ganguli, S. (2013). Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In International Conference on Learning Representations