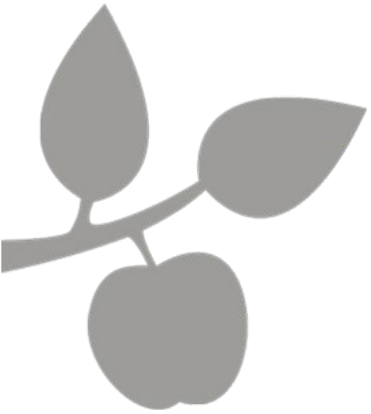
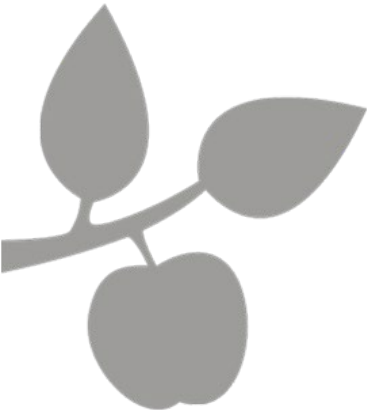


SDU Summer School

Deep Learning

Summer 2022

Welcome to the Summer School



Gradient-Based Learning

- **Recap: Definition**
- **Stochastic Gradient Descent**
- **Problems with Deep Learning**
- **Different Optimizers**

Training Feedforward Networks

- We have seen how to construct a FNN
- We can input a data point into the FNN and receive a prediction
- We now need to define a function which judges the quality of our predictions and allows us to optimize the network, i.e., train the network.

Training Feedforward Networks

- We already two such error functions:

- Mean-Squared-Error (minimize):

$$J(\theta) = \frac{1}{n}\sum_{i}^{n}\left\|y^{(i)} - \hat{y}^{(i)}\right\|^2$$

- For Logistic Regression we have seen the MLE (maximize)

$$L(X, \boldsymbol{\theta}) = \prod_{i=1}^{n} p(y^{(i)}|\boldsymbol{x}^{(i)}; \boldsymbol{\theta})$$

Cross Entropy Loss

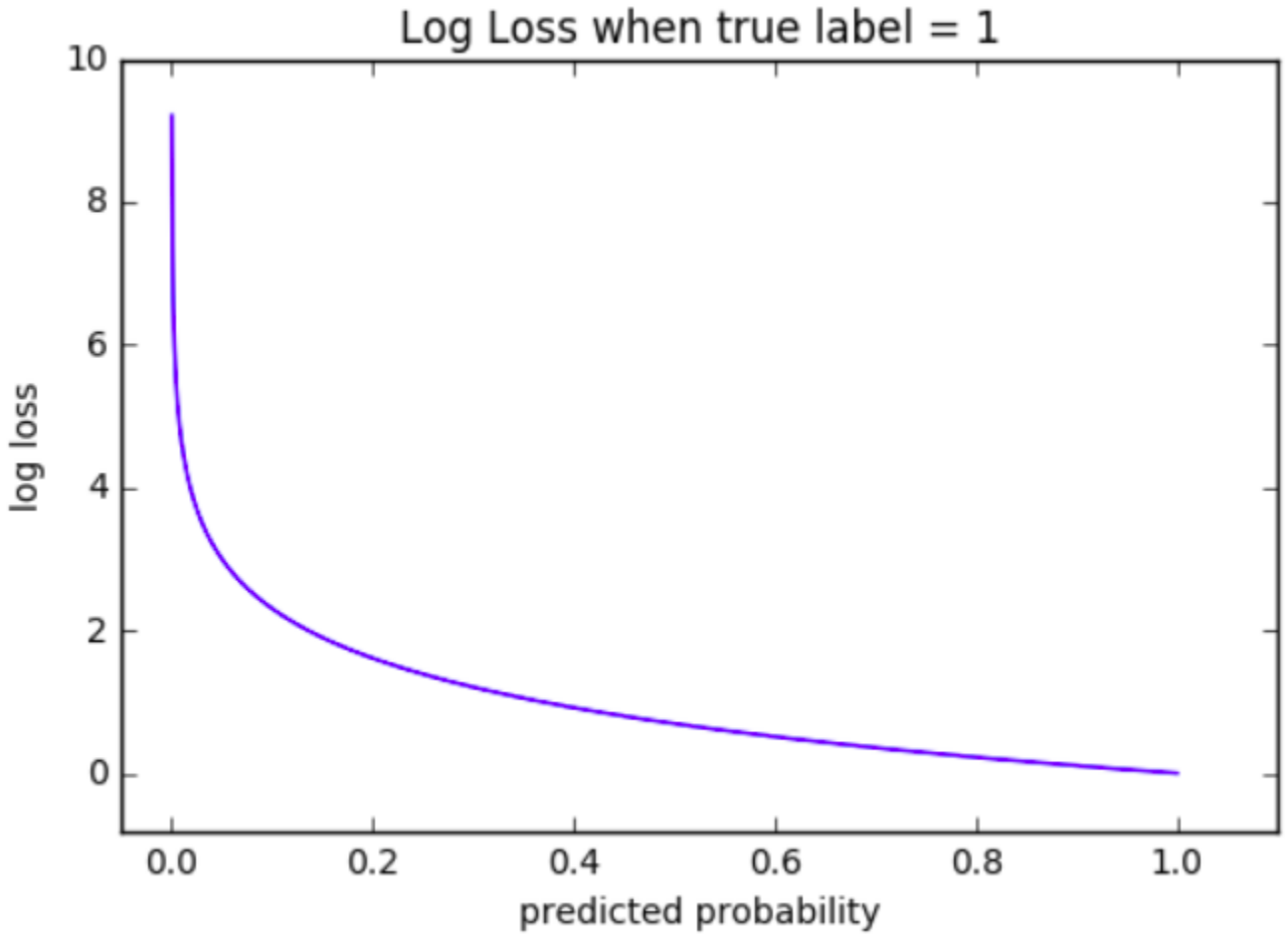
- For Neural Networks, we usually us the cross-entropy loss
- Minimizing the cross-entropy loss corresponds to maximize the log likelihood:

$$J(\boldsymbol{\theta}) = -\mathbb{E}[p(y|\boldsymbol{x};\boldsymbol{\theta})]$$

- In case of our 2-case logistic regression:

$$-\frac{1}{n}\sum_{i=1}^{n}\left[y^{(i)}\log h_{\boldsymbol{\theta}}(\boldsymbol{x}) + \left(1-y^{(i)}\right)\log\left(1-h_{\boldsymbol{\theta}}(\boldsymbol{x})\right)\right]$$

Cross Entropy Loss

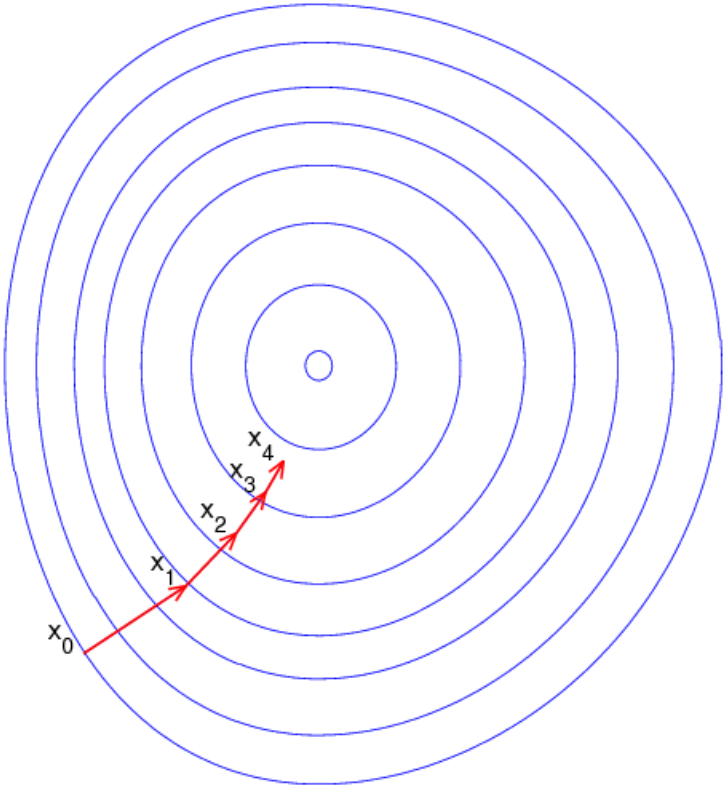


Our Recipe

- We have the model defined
  - By constructing the neural network
- We have a loss function
  - For example the cross-entropy
- We have a goal:
  - Modify the model parameters such that we minimize the loss function
- How to minimize a function?
  - Find the nulls of the derivate?
  - Not possible, function too complicated.

The Central Idea

- Update the model parameters following the steepest slope



Formally

- Need partial derivatives

$$\frac{\partial}{\partial x_i} f(\boldsymbol{x})$$

- Measures how f changes as only variable x_i increases at point x

- Gradient is vector containing all of the partial derivatives denoted with $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$
  - Element i of the gradient is the partial derivative of f wrt x_i
  - Critical points are where every element of the gradient is equal to zero
  - A function can be minimized when moving in the direction opposite to the gradient

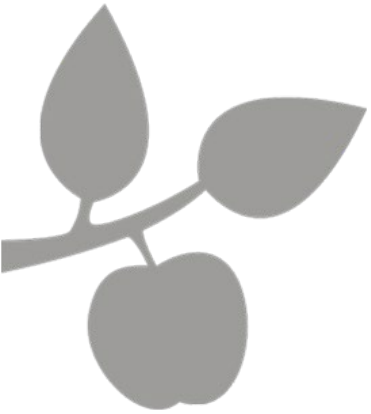
Update the weights

- We can decrease f by moving in the direction of the negative gradient vector

- Steepest descent proposes a new point

$$\boldsymbol{x}' = \boldsymbol{x} - \epsilon \nabla_{\boldsymbol{x}} f(\boldsymbol{x})$$

- With ϵ being the learning rate (there are many methods of defining ϵ)
- Ascending an objective function of discrete parameters is called hill climbing



Gradient-Based Learning

- Recap: Definition
- Stochastic Gradient Descent
- Problems with Deep Learning
- Different Optimizers

Stochastic Gradient Descent (SGD)

- A recurring problem in machine learning:
  - large training sets are necessary for good generalization
  - but large training sets are also computationally expensive

- Nearly all deep learning is powered by one very important algorithm: **Stochastic Gradient Descent**

Insight of SGD

- Insight: Gradient descent based on only a sample (we don't have the universe as data) is an **expectation**
  - Expectation may be approximated using small set of samples

- In each step of SGD we can sample a minibatch of examples

$$B = \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m')}\}$$

  - drawn uniformly from the training set
  - Minibatch size m' is typically chosen to be small: 1 to a hundred
  - Crucially m' is held fixed even if sample set is in billions
  - We may fit a training set with billions of examples using updates computed on only a hundred examples

SGD Estimate on minibatch

- Estimate of gradient is formed as

$$\boldsymbol{g} = \frac{1}{m'} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m'} L\left(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}\right)$$

using only the examples of the minibatch

- SDG then simply follows the estimated gradient downhill

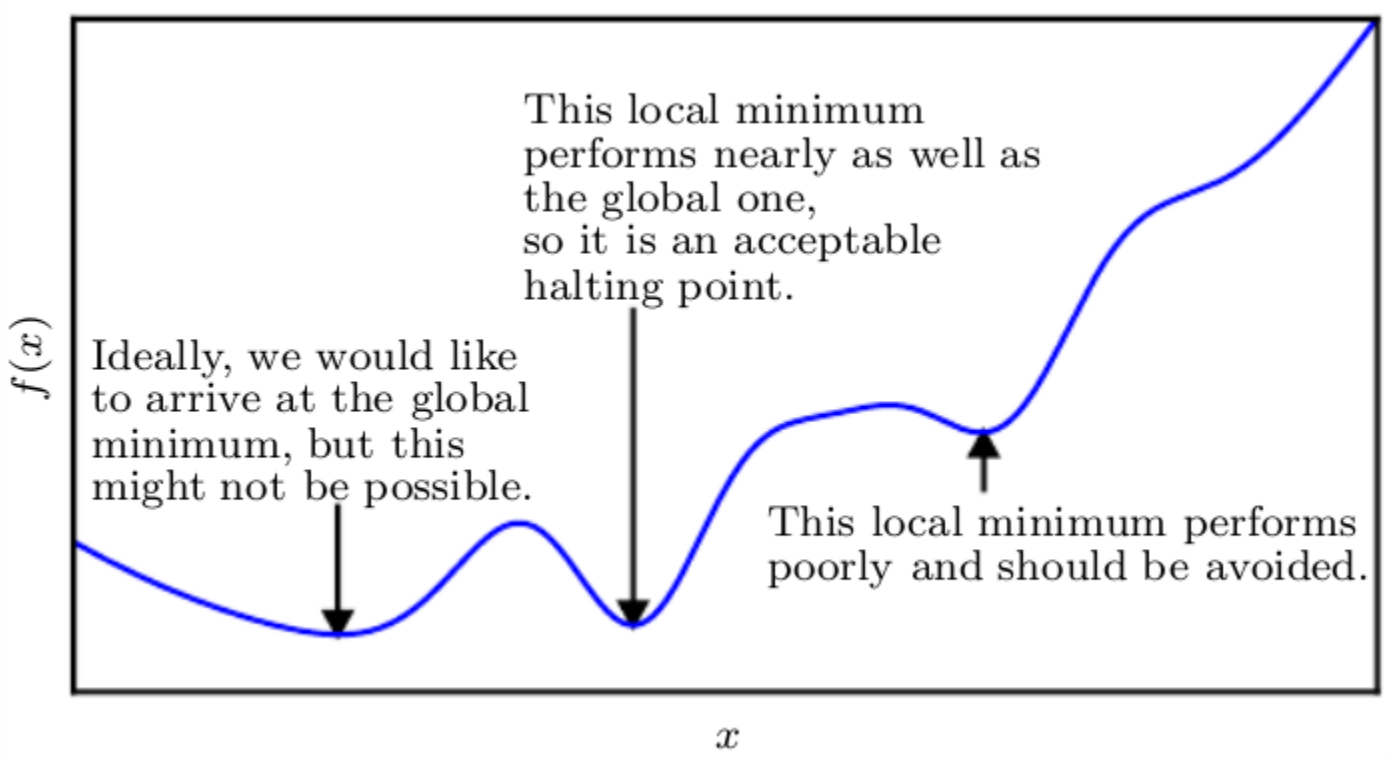
$$\boldsymbol{\theta}' = \boldsymbol{\theta} - \epsilon \boldsymbol{g}$$

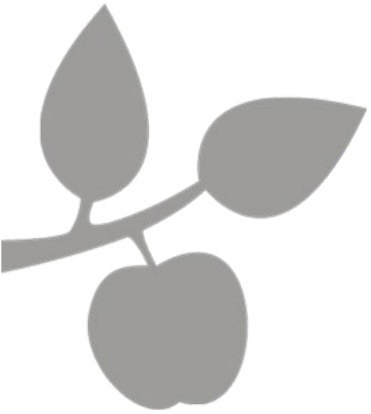
How good is SGD?

- In the past gradient descent was regarded as slow and unreliable
- Application of gradient descent to non-convex optimization problems was regarded as unprincipled
- SGD is not guaranteed to arrive at even a local minimum in reasonable time

- **But it often finds a very low value of the cost function quickly enough**
- As $m \rightarrow \infty$ the model will eventually converge to its best possible test error before SGD has sampled every example

Good Enough in Practice





Gradient-Based Learning

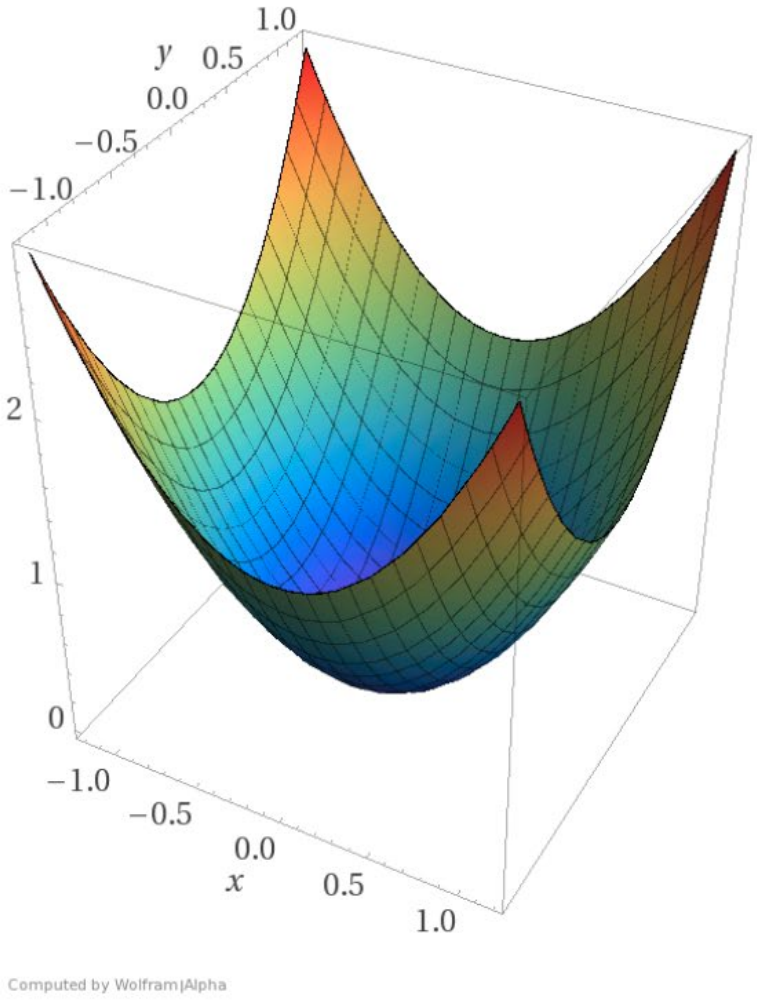
- Recap: Definition
- Stochastic Gradient Descent
- Problems with Deep Learning
- Different Optimizers

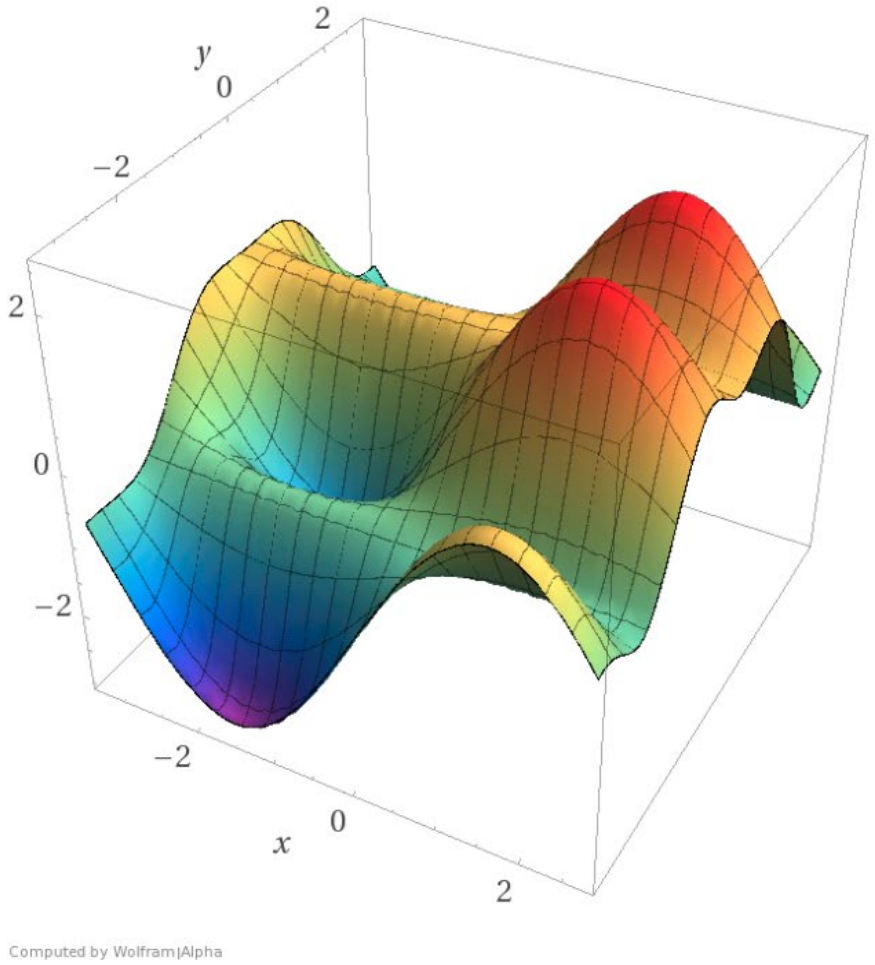
Specialties of Deep Learning

- Neural Network training not different from ML models with gradient descent. The components are needed:
  1. optimization procedure, e.g., gradient descent
  2. cost function, e.g., MLE
  3. model family, e.g., linear with basis functions

- Difference: **nonlinearity causes non-convex loss**
  - Use iterative gradient-based optimizers that merely drives cost to low value
  - No guarantees in comparison to convex optimizations
  - The initialization matters

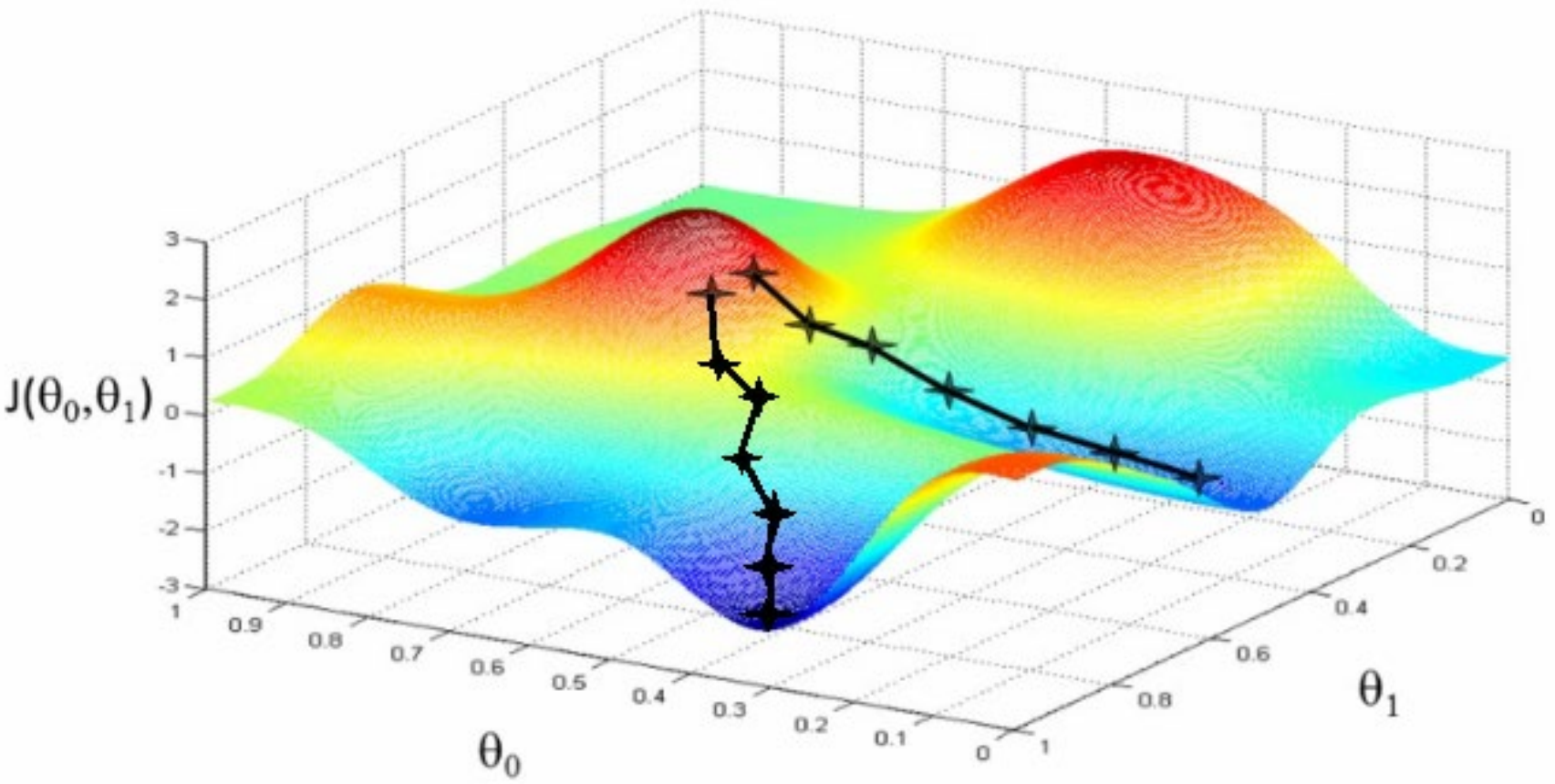
Convex vs. Non-Convex





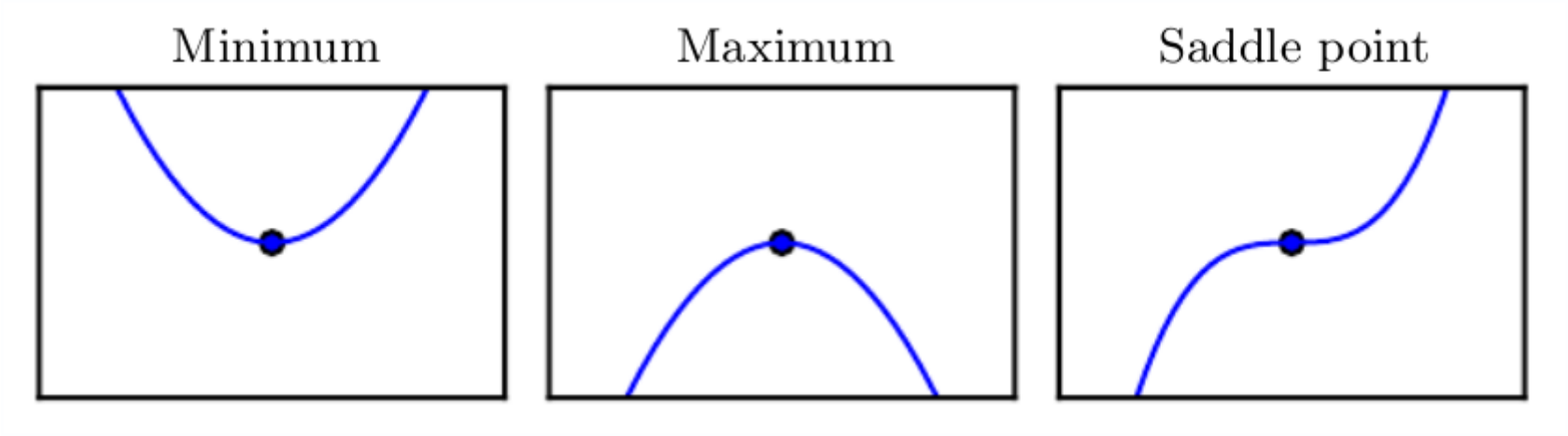
- https://www.matroid.com/blog/post/the-hard-thing-about-deep-learning

Problem: We can end-up in local minima



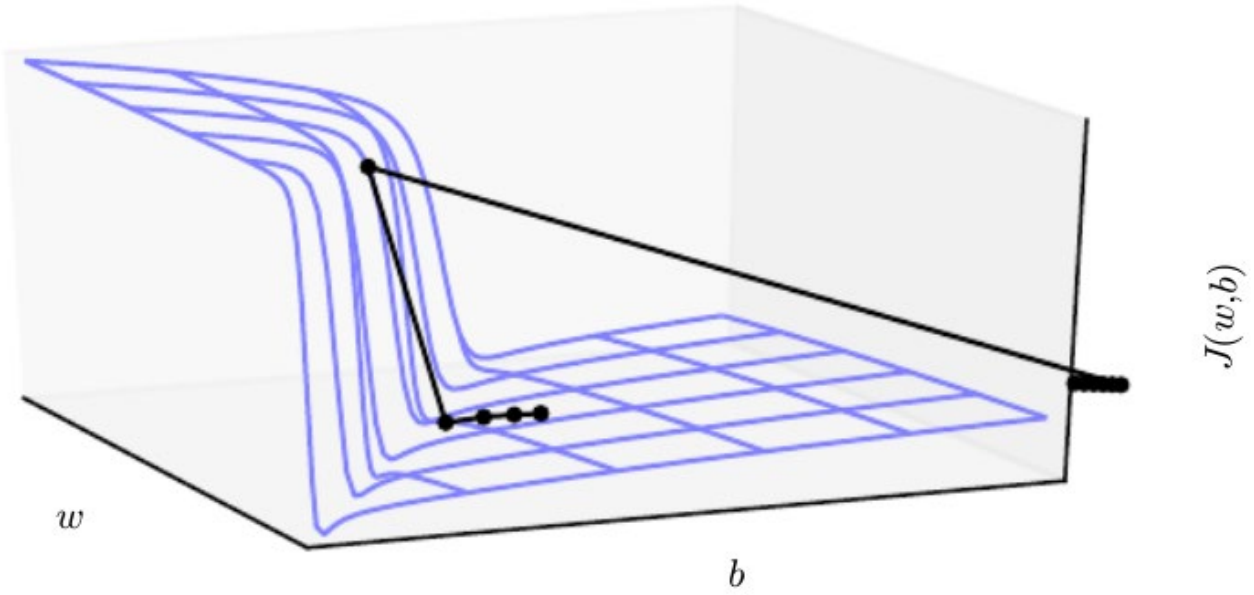
Problem: Stationary points, Local Optima

- When $f'(x) = 0$ derivative provides no information about direction of move
- Points where $f'(x) = 0$ are known as stationary or critical points
  - **Local minimum/maximum**: a point where $f(x)$ lower/ higher than all its neighbors
  - **Saddle Points**: neither maxima nor minima



Cliffs and Exploding Gradients

- Neural networks with many layers have steep regions i.e., cliffs
  - Result from multiplying several large weights
  - E.g., RNNs with many factors at each time step
- Gradient update step can move parameters extremely far, jumping off cliff altogether
- Cliffs dangerous from either direction
- Gradient clipping heuristics can be used



Inexact Gradients

- Optimization algorithms assume we have access to exact gradient or Hessian matrix

- In practice we have a noisy or biased estimate
  - Every deep learning algorithm relies on sampling-based estimates
    - In using minibatch of training examples
  - In other case, objective function is intractable
    - In which case gradient is intractable as well

Poor Correspondence between Local and Global Structure

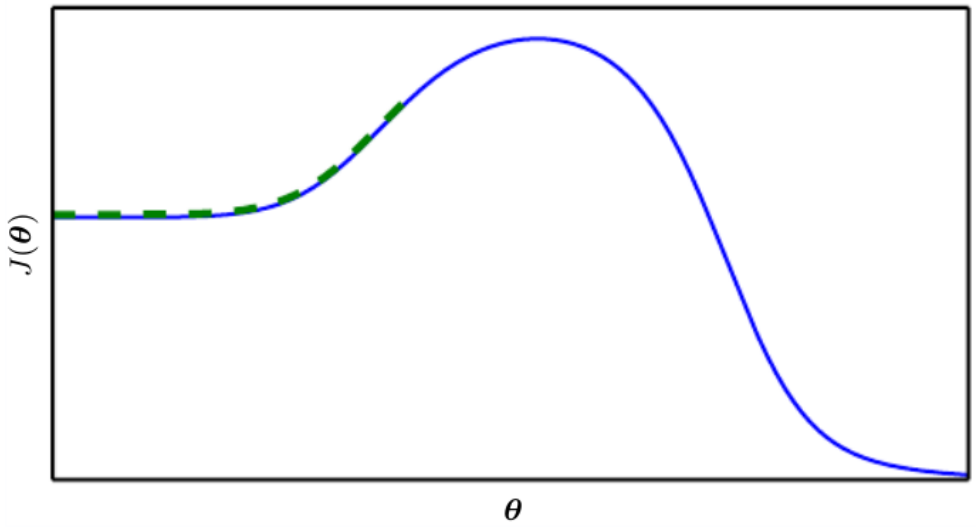
- It can be difficult to make a single step if:
  - $J(\theta)$ is poorly conditioned at the current point θ
  - θ lies on a cliff
  - θ is a saddle point hiding the opportunity to make progress downhill from the gradient

- It is possible to overcome all these problems and still perform poorly:
  - if the direction that makes most improvement locally does not point towards distant regions of much lower cost

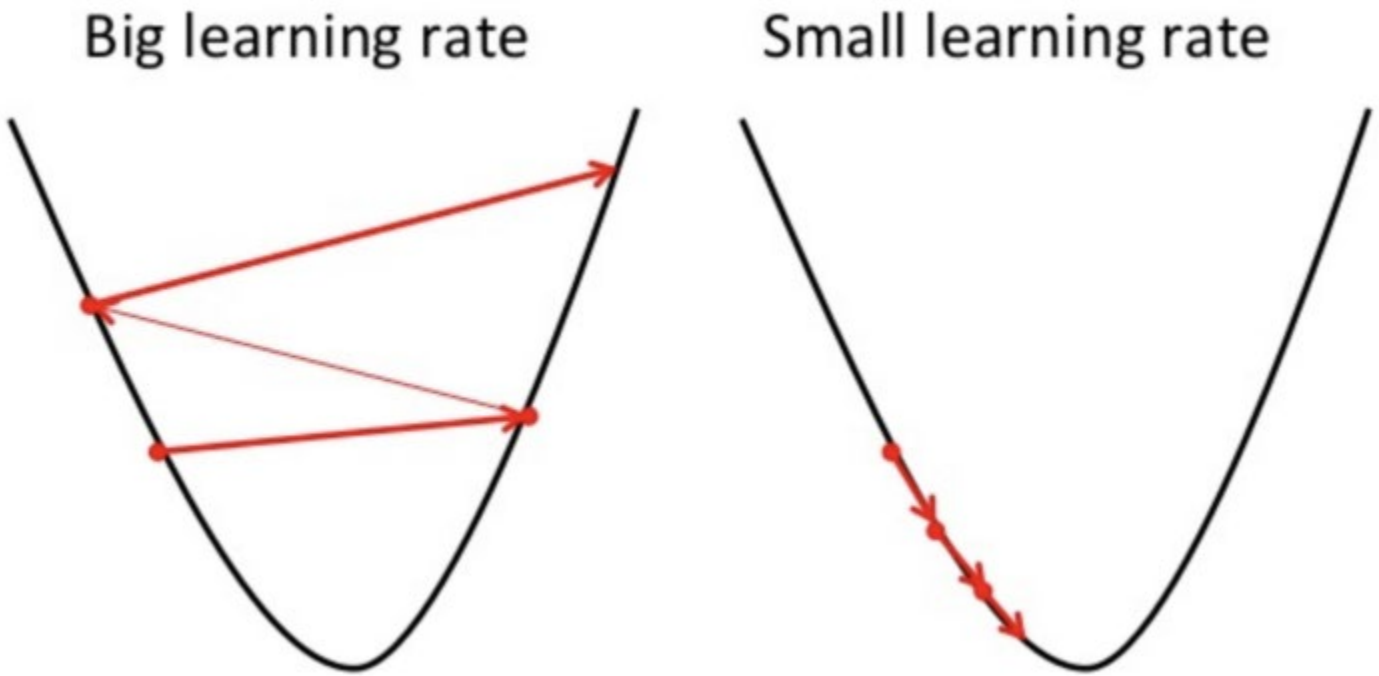
Need for good initial points

- Optimization based on local downhill moves can fail if local surface does not point towards the global solution

- Research directions are aimed at finding good initial points for problems with a difficult global structure
  - For example: Trajectory of circumventing mountains may be long and result in excessive training time



Problem: The Learning Rate



Convergence of Steepest Descent

- Steepest descent converges when every element of the gradient is zero

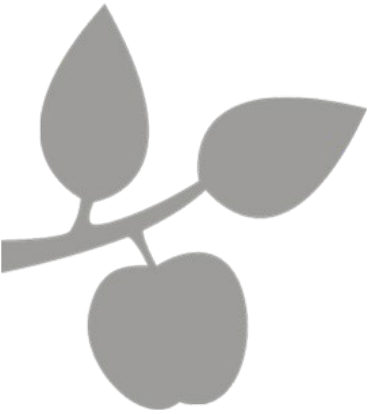
- Pure math way of life:
  - Find literally the smallest value of $f(x)$
  - Or maybe: find some critical point of $f(x)$ where the value is locally smallest

- **Deep learning way of life:**
  - **Decrease the value of $f(x)$ a lot**
  - **But we have a highly non-convex problem (because of the activation functions) => No guarantees!**

About the Gradient

- Gradient must be large and predictable enough to serve as good guide to the learning algorithm
- Functions that **saturate** (become very flat) undermine this
  - Because the gradient becomes very small
  - Happens when activation functions producing output of hidden/output units saturate

- **Negative log-likelihood** helps avoid saturation problem for many models
  - Many output units involve exp functions that saturate when its argument is very negative
  - Log function in Negative log likelihood cost function undoes exp of some units



Gradient-Based Learning

- Recap: Definition
- Stochastic Gradient Descent
- Problems with Deep Learning
- Different Optimizers

Now, why are there different optimizers?

- The actual implementations try to avoid or minder the aforementioned problems
- They differentiate in the way they are doing it, often the differences are rather subtle
- Keras comes with the following optimizers:
  - SGD
  - RMSProp
  - Adagrad
  - Adadelta
  - Adam
  - Adamax
  - Nadam

Tricks they use: Decreasing Learning Rate

- Batch* gradient descent (i.e. using all samples) can use a fixed learning rate
  - Since true gradient becomes small and then 0

- SGD has a source of error
  - Random sampling of m training samples
  - Sufficient condition for SGD convergence

$$\sum \varepsilon = \infty; \sum \varepsilon^2 < \infty$$

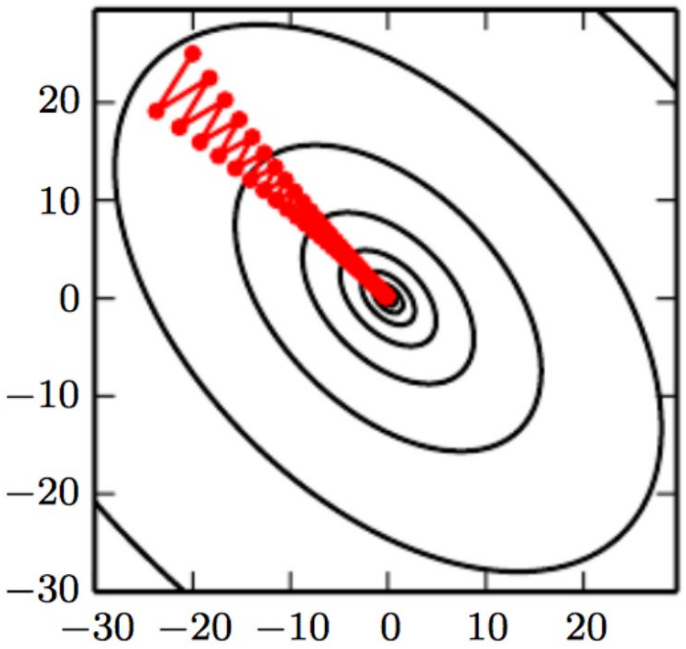
  - Common to decay learning rate linearly until iteration τ: $\varepsilon_k = (1-\alpha)\varepsilon_0 + \alpha\varepsilon_\tau$ with $\alpha = \frac{k}{\tau}$.
  - After iteration τ, it is common to leave ε constant

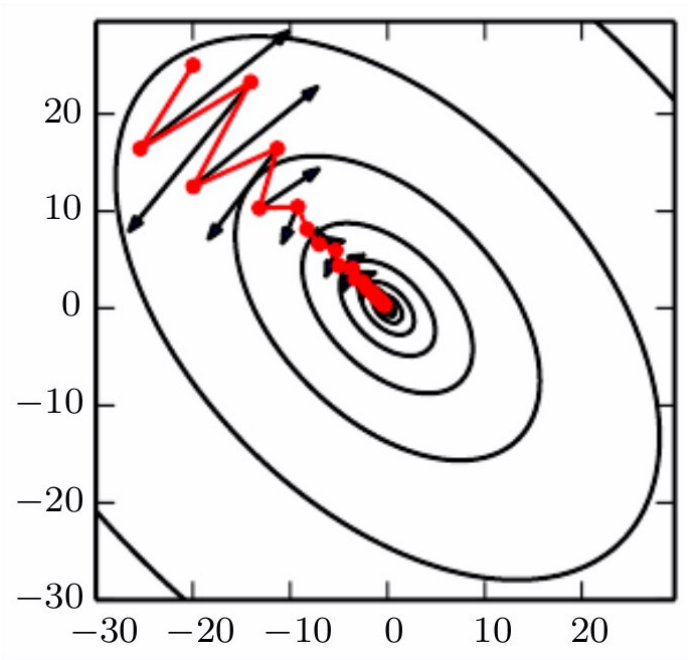
➢ *This is unfortunately a very confusing terminology: Methods which use the entire dataset are called Batch-methods. Do not confuse with the term "minibatch" from SDG.

Tricks they use: Momentum

- Momentum helps accelerate Gradient Descent(GD) when we have
  - Surfaces that curve more steeply in one direction than in another direction
  - Facing high curvature
  - Small but consistent gradients
  - Noisy gradients
- It also dampens the oscillation as shown in the next slide
- Algorithm accumulates moving average of past gradients and move in that direction, while exponentially decaying

Tricks they use: Momentum





Tricks they use: Momentum

- Introduce variable v, or velocity
  - It is the direction and speed at which parameters move through parameter space
  - Momentum is physics is mass times velocity
  - The momentum algorithm assumes unit mass
  - A hyperparameter $\alpha \in [0,1)$ determines exponential decay

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \varepsilon \nabla_{\theta} \left(\frac{1}{m} \sum_{i=1}^{m} L\left(f(x^{(i)}; \theta), y^{(i)} \right) \right)$$

$$\theta \leftarrow \theta + \boldsymbol{v}$$

  - The larger α is relative to ε, the more previous gradients affect the current direction

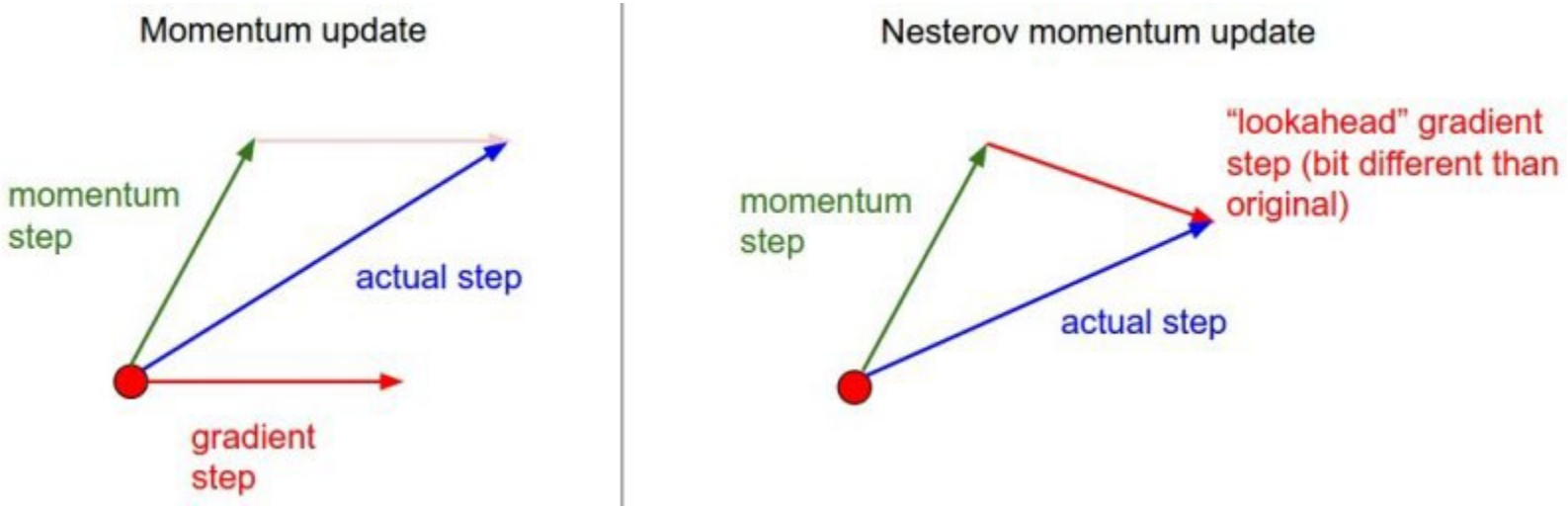
Tricks they use:
Nesterov Accelerated Gradient

- Nesterov acceleration optimization is like a ball rolling down the hill but knows exactly when to slow down before the gradient of the hill increases again.

- We calculate the gradient not with respect to the current step but with respect to the future step.

- We evaluate the gradient of the looked ahead and based on the importance then update the weights.

Tricks they use:
Nesterov Accelerated Gradient



Tricks they use:
Adaptive Learning Rate

- We have discussed how important the learning rate is
  - Set too small, we train very slow
  - Set too large, we might wildly jump above minima
- General Idea of adaptive learning:
  - Modify the learning rate for individual parameters
  - Based on their history
  - The different optimizers mainly distinguish each other how the adapt the learning rate.
- This is were the "ADA" comes from in most optimizers

Back to the Optimizers

- SGD
- RMSProp
- Adagrad
- Adadelta
- Adam
- Adamax
- Nadam

Back to the Optimizers

- **SGD**
  - Standard gradient descent
  - Can use decay, momentum, and Nesterov

- Adagrad
- RMSProp
- Adadelta
- Adam
- Adamax
- Nadam

Back to the Optimizers

- SGD
- **Adagrad**
  - We perform larger updates for infrequent parameters and smaller updates for frequent parameters. (good for word embeddings where infrequent words require larger updates)
  - Often has radically quick diminishing learning rates
- RMSProp
- Adadelta
- Adam
- Adamax
- Nadam

Normal SDG for single parameter θ_i

$$\theta_{t+1,i} = \theta_{t,i} - \eta \cdot g_{t,i}$$

Adagrad Method:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

with G_t as the diagonal matrix with the elements i, i being the sum of squares of the past updated parameters.

Back to the Optimizers

- SGD
- Adagrad
- **RMSProp**
  - "Root Mean Square Propagation"
  - Tries to ease on the diminishing learning rate decay from Adagrad
  - Uses an exponentially decaying running average of the squared gradients $E[g^2]$, similar to momentum.
  - Decay normally set to $\gamma = 0.9$
- Adadelta
- Adam
- Adamax
- Nadam

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$

$$\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

$$\Delta\theta_t = -\frac{\eta}{RMS[g]_t} g_t$$

Back to the Optimizers

- SGD
- Adagrad
- RMSProp
- **Adadelta**
  - Very similar to RMSProp (came up around the same time)
  - They introduce the root squared parameter updates
  - Replaces the need to set a learning rate

$$E[\Delta\theta^2]_t = \gamma E[\Delta\theta^2]_{t-1} + (1-\gamma)\Delta\theta_t^2$$

$$RMS[\Delta\theta]_t = \sqrt{E[\Delta\theta^2]_t + \epsilon}$$

$$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

- Adam
- Adamax
- Nadam

Back to the Optimizers

- SGD
- Adagrad
- RMSProp
- Adadelta
- **Adam**
  - Adaptive Moment Estimation
  - Next to the exp. dec. avg. of past squared gradients v_t Adam also keeps an exp. dec. avg. of past gradients m_t
  - Whereas momentum can be seen as a ball running down a slope, Adam behaves like a heavy ball with friction, which thus prefers flat minima in the error surface.
- Adamax
- Nadam

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

Bias corrected momentums

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \qquad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

Back to the Optimizers

- SGD
- Adagrad
- RMSProp
- Adadelta
- Adam
- **Adamax**
  - Same as Adam, but based on the max-norm L^∞.
- Nadam

$$
\begin{aligned}
u_t &= \beta_2^\infty v_{t-1} + (1-\beta_2^\infty)|g_t|^\infty \\
&= \max(\beta_2 \cdot v_{t-1}, |g_t|)
\end{aligned}
$$

Back to the Optimizers

- SGD
- Adagrad
- RMSProp
- Adadelta
- Adam
- Adamax
- **Nadam**
  - Nesterov-accelerated Adaptive Moment Estimation
  - Adam is basically RMSProp with momentum
  - We have seen that Nesterov is often more efficient
  - Welcome to Nadam: Adam which uses the Nesterov momentum.

Do we have everything?

- We have the model defined
- We have a loss function
- We have the optimization goal
- We the optimization algorithm

But what about the gradient?