Foundations of Deep Learning, Winter Term 2021/22

Week 4: Optimization of Neural Networks

Optimization of Neural Networks

Frank Hutter Abhinav Valada

University of Freiburg

Overview of Week 4

Lecture Overview

Foundations of Deep Learning, Winter Term 2021/22

Week 4: Optimization of Neural Networks

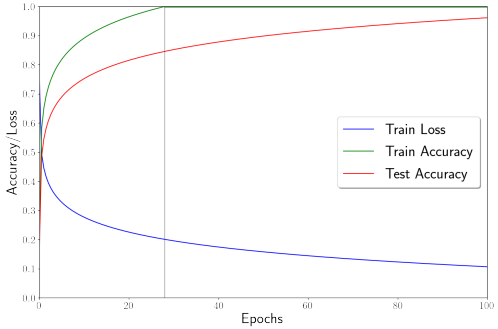
Learning vs. Pure Optimization

Frank Hutter Abhinav Valada

University of Freiburg

Learning vs. Optimization: Surrogate Loss

- Goal: optimize performance measure P
  - P might be intractable (e.g., 0/1 accuracy is not differentiable)
- To exploit gradient-based optimization
  - For a non-differentiable P, we have to introduce a surrogate loss function L
  - E.g., cross entropy loss instead of 0/1 accuracy
- The surrogate loss function may also enable better generalization



Learning vs. Optimization: Empirical Risk Minimization

- Goal: optimize performance on the test set
  - Risk is defined over the true data-generating distribution p_{data}

$$J^*(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},y)\sim p_{data}} L(f(\boldsymbol{x};\boldsymbol{\theta}), y) \tag{8.2}$$

  - But we cannot compute this, since we don't have p_{data}

- Doable in practice: optimize performance on the training data
  - Empirical risk is defined over the empirical data distribution $\hat{p}_{data}$

$$J(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},y)\sim \hat{p}_{data}} L(f(\boldsymbol{x};\boldsymbol{\theta}), y) \tag{8.1}$$

$$= \frac{1}{m}\sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), y^{(i)}) \tag{8.3}$$

Learning vs. Optimization: Which Function to Optimize

- Goal: optimize performance on the test set
  - Risk is defined over the true data-generating distribution p_{data}

$$J^*(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},y)\sim p_{data}} L(f(\boldsymbol{x};\boldsymbol{\theta}), y) \tag{8.2}$$

  - But we cannot compute this, since we don't have p_{data}
- Doable in practice: optimize performance on the training data
  - Empirical risk is defined over the empirical data distribution $\hat{p}_{data}$

$$J(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},y)\sim \hat{p}_{data}} L(f(\boldsymbol{x};\boldsymbol{\theta}), y) \tag{8.1}$$

$$= \frac{1}{m}\sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), y^{(i)}) \tag{8.3}$$

- Optimizing L directly may not generalize
  - We can include regularization terms in the function we optimize to limit overfitting
  - We can even vary these terms over time

Learning vs. Optimization: Decomposition of Objective Function

- Our surrogate loss function decomposes into a sum over data points
- E.g., maximum log likelihood estimation:

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x},\mathrm{y}\sim\hat{p}_{data}} \log\, p_{model}(\boldsymbol{x}, y; \boldsymbol{\theta}) = \sum_{i=1}^{m} \log\, p_{model}(\boldsymbol{x}^{(i)}, y^{(i)}; \boldsymbol{\theta})$$

- The gradient of J is also an expectation over the training set:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x},\mathrm{y}\sim\hat{p}_{data}} \nabla_{\boldsymbol{\theta}} \log\, p_{model}(\boldsymbol{x}, y; \boldsymbol{\theta}) \tag{8.6}$$

  $\Rightarrow$ Computationally expensive for large training sets

- We can do cheaper computations based on subsets of the data
  - We can choose in which order to consider the data: curriculum learning
  - Stochastic gradient descent (SGD) based on mini batches

Questions to Answer for Yourself / Discuss with Friends

- Repetition:
  Name four ways in which learning differs from pure optimization.

- Repetition:
  Why do we *have to* use a surrogate loss if we care about 0/1 accuracy and want to use gradient information? What can be a useful side-effect of doing so?

Lecture Overview

Foundations of Deep Learning, Winter Term 2021/22

Week 4: Optimization of Neural Networks

Basics of Gradient-Based Optimization

Frank Hutter Abhinav Valada

University of Freiburg

The Goal of Our Optimization Problem

- We're interested in problems of the form

$$\underset{\boldsymbol{x}}{minimize}\ f(\boldsymbol{x}),$$

where $\boldsymbol{x}$ is a vector of suitable size.

- A global minimum $\boldsymbol{x}^*$ is a point such that:

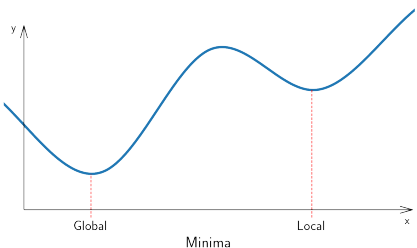
$$f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$$

for all $\boldsymbol{x}$.

- A local minimum $\boldsymbol{x}^+$ is a point such that there exists $r > 0$ with

$$f(\boldsymbol{x}^+) \leq f(\boldsymbol{x})$$

for all points $\boldsymbol{x}$ with $||\boldsymbol{x} - \boldsymbol{x}^+|| < r$



Gradient-Based Optimization: Need For Iterative Solvers

- Analytical way to find a minimum:
  For a local minimum $\boldsymbol{x}^+$, the gradient of f becomes zero:

$$\frac{\partial f}{\partial x_i}(\boldsymbol{x}^+) = 0 \quad \text{for all } i$$

  Hence, calculating all partial derivatives and looking for zeros is a good idea

- But: for neural networks, we can't write down a solution for the minimization problem in closed form
  - even though $\frac{\partial f}{\partial x_i} = 0$ holds at (local) solution points

  $\rightarrow$ need to resort to iterative methods

Gradient Descent: Intuition for the Update Equation

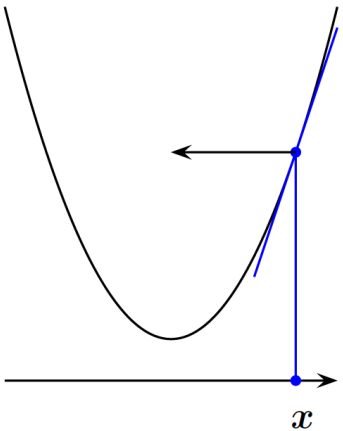
- Numerical way to find a minimum, searching:
  assume we start at point $\boldsymbol{x}$.

  Which is the best direction to search for a point $\boldsymbol{x}'$ with $f(\boldsymbol{x}') < f(\boldsymbol{x})$?

  Which is the best stepwidth?
- general principle:

$$x_i' \leftarrow x_i - \alpha \frac{\partial f}{\partial x_i}$$

$\alpha > 0$ is called learning rate



Gradient Descent: The Full Algorithm

- Gradient descent approach:

  Require: mathematical function f, learning rate $\alpha > 0$
  Ensure: returned vector is close to a local minimum of f

  1: choose an initial point $\boldsymbol{x}$
  2: **while** $||\nabla f(\boldsymbol{x})||$ not close to 0 **do**
  3: $\quad \boldsymbol{x} \leftarrow \boldsymbol{x} - \alpha \nabla f(\boldsymbol{x})$
  4: **end while**
  5: **return** $\boldsymbol{x}$

- Note: $\nabla f := [\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_K}]$ for K dimensions

Questions to Answer for Yourself / Discuss with Friends

- Application of definitions we saw:
  How does the number of local optima relate to the number of global optima?

- Repetition:
  Why does the update equation of gradient descent have the form $\boldsymbol{x} \leftarrow \boldsymbol{x} - \alpha \nabla f(\boldsymbol{x})$?

Lecture Overview

Foundations of Deep Learning, Winter Term 2021/22

Week 4: Optimization of Neural Networks

Ill-Conditioning and Possible Solutions

Frank Hutter Abhinav Valada

University of Freiburg

Recall: Positive Definite Matrices and the Condition Number

Positive Definite Matrix

An $n \times n$ matrix $\boldsymbol{M}$ is called positive definite if the scalar $\boldsymbol{z}^\top \boldsymbol{M}\boldsymbol{z}$ is greater than zero for all non-zero $n \times 1$ vectors $\boldsymbol{z}$.

We write $\boldsymbol{M} \succ 0$ to denote that $\boldsymbol{M}$ is positive definite.

Condition number

The condition number $\kappa(\boldsymbol{Q})$ of a matrix $\boldsymbol{Q}$ with largest eigenvalue λ_{max} and smallest eigenvalue λ_{min} is defined as $\kappa(\boldsymbol{Q}) = \frac{\lambda_{max}}{\lambda_{min}}$

Matrices with large condition number are called ill-conditioned.

The Importance of the Condition Number

Example: Quadratic Function

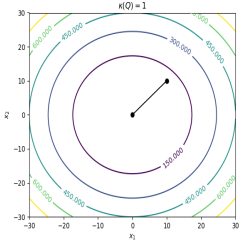
$$f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{Q}\boldsymbol{x} \qquad \text{with } \boldsymbol{Q} \succ 0, \text{ symmetric}$$

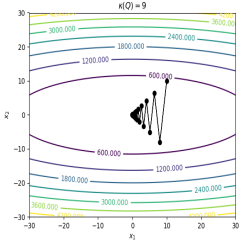
- $\nabla f(\boldsymbol{x}) = \boldsymbol{Q}\boldsymbol{x}$
- $\nabla^2 f(\boldsymbol{x}) = \boldsymbol{Q}$

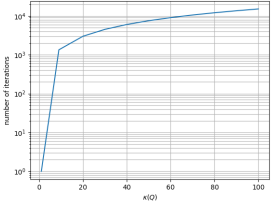
For such a quadratic function, the contraction rate of gradient descent can be shown to be:

$$||\boldsymbol{x}_{k+1} - \boldsymbol{x}^*|| \leq \left(\frac{\kappa(\boldsymbol{Q}) - 1}{\kappa(\boldsymbol{Q}) + 1}\right) ||\boldsymbol{x}_k - \boldsymbol{x}^*||$$

Ill-Conditioning Hurts Performance







Ill-Conditioning: Problems and Solutions

The class of ill-conditioned problems is very broad

- ill-conditioning causes the typical but undesired **zig-zag** behavior.
- condition number issues are the biggest problem for gradient methods in practice

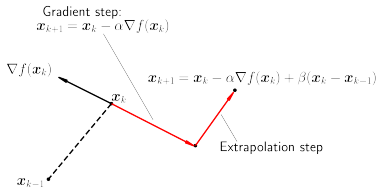
How to alleviate the zig-zag behavior?

- **momentum**
- **preconditioning**

Momentum

- Add an extrapolation step to the gradient step:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha \nabla f(\boldsymbol{x}_k) + \beta(\boldsymbol{x}_k - \boldsymbol{x}_{k-1}) = \boldsymbol{x}_k - \alpha \sum_{i=0}^{k} \beta^{k-i} \boldsymbol{g}_i$$

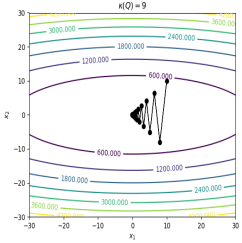


For a quadratic function, the contraction rate of gradient descent with momentum can be shown to be:

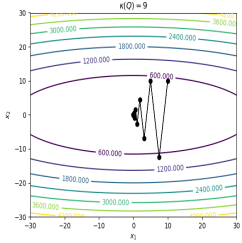
$$||\boldsymbol{x}_{k+1} - \boldsymbol{x}^*|| \leq \left(\frac{\sqrt{\kappa(\boldsymbol{Q})} - 1}{\sqrt{\kappa(\boldsymbol{Q})} + 1} \right) ||\boldsymbol{x}_k - \boldsymbol{x}^*||$$

[Image Credit: Based on Figure 2.1.2 of Bertsekas: "Convex Optimization Algorithms"]

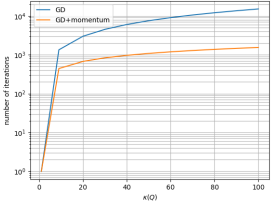
Momentum



- gradient descent with zig-zag



- momentum alleviates zig-zag

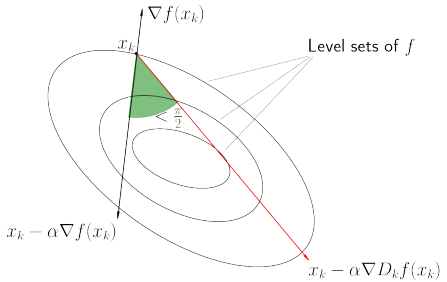


Pre-Conditioning (or Scaling)

- We will use a preconditioning matrix $\boldsymbol{D}_k$, with $\boldsymbol{D}_k \succ 0$ and symmetric
- We pre-multiply the gradient by $\boldsymbol{D}_k$:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha \boldsymbol{D}_k \nabla f(\boldsymbol{x}_k)$$

- This still yields a cost descent method, as long as α is small enough:



[Image Credit: Based on Figure 2.1.1 of Bertsekas: "Convex Optimization Algorithms"]

Pre-Conditioning: Analysis for Quadratic Function

Main idea: modify the "effective condition number".

- Recall the quadratic function example: $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{Q}\boldsymbol{x}$
- Pre-conditioning: $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha \boldsymbol{D}_k \nabla f(\boldsymbol{x}_k)$

- Define a new function: $h_k(\boldsymbol{y}) = f(\boldsymbol{D}_k^{\frac{1}{2}}\boldsymbol{y}) = \frac{1}{2}\boldsymbol{y}^\top \boldsymbol{D}_k^{\frac{1}{2}}\boldsymbol{Q}\boldsymbol{D}_k^{\frac{1}{2}}\boldsymbol{y}$
- Interpret preconditioned gradient descent in $\boldsymbol{x}$ as gradient descent in $\boldsymbol{y}$:

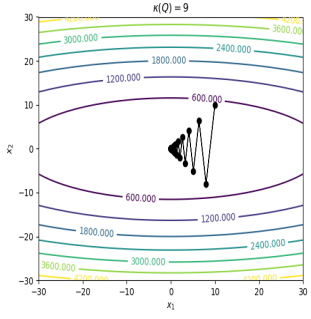
$$\boldsymbol{y}_{k+1} = \boldsymbol{y}_k - \alpha \nabla h_k(\boldsymbol{y}_k)$$

The convergence rate is then determined by $\kappa(\boldsymbol{D}_k^{\frac{1}{2}}\boldsymbol{Q}\boldsymbol{D}_k^{\frac{1}{2}})$

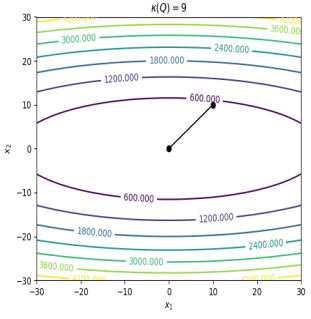
- In this sense, the "best" scaling is the inverse of the Hessian matrix

$$\boldsymbol{D}_k = \nabla^2 f(\boldsymbol{x}_k)^{-1} = \boldsymbol{Q}^{-1}$$

Pre-Conditioning: Performance for Quadratic Function



- gradient descent with zig-zag



- scaling with $\boldsymbol{D}_k = \nabla^2 f(\boldsymbol{x}_k)^{-1}$ (Newton's method)

Questions to Answer for Yourself / Discuss with Friends

- Explanation:
  Explain the main idea behind gradient descent with momentum
- Explanation:
  Explain the main idea behind preconditioning
- Repetition:
  Give the update equation of gradient descent with momentum at step k
- Repetition:
  Give the update equation at step k of preconditioned gradient descent with preconditioning matrix $\boldsymbol{D}_k$
- Repetition of a derivation:
  Why is the inverse Hessian $\boldsymbol{D}_k = \nabla^2 f(\boldsymbol{x}_k)^{-1}$ the optimal preconditioner for gradient descent on a quadratic function $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{Q}\boldsymbol{x}$?

Lecture Overview

Foundations of Deep Learning, Winter Term 2021/22

Week 4: Optimization of Neural Networks

Challenges in the Optimization of Neural Networks

Frank Hutter Abhinav Valada

University of Freiburg

Problem Framework

We would like to find a minimizer of:

$$f(\boldsymbol{\theta}) := \frac{1}{n} \sum_{i=1}^{n} L(m(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), y^{(i)}) \qquad \text{, where } \boldsymbol{\theta} \in \mathbb{R}^d$$

Please note the change of notation from before:

- The pure optimization literature aims to minimize $f(\boldsymbol{x})$
- In the deep learning literature, $\boldsymbol{x}^{(i)}$ is the i-th data point, and we're optimizing with respect to the weights $\boldsymbol{\theta}$

The main challenges in this optimization problem are:

- Non-convexity: The model $m : \mathbb{R}^d \to \mathbb{R}^o$ is a highly non-convex function
- Large datasets: in modern datasets, n is very large
- High dimensionality: In state-of-the-art neural architectures, d is very large
- Structure of the network: can yield hard-to-optimize response surfaces

Non-Convexity: Hessian Matrix

Consider a non-convex function:

$$f(\boldsymbol{\theta}) := \frac{1}{n}\sum_{i=1}^{n} L(m(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), y^{(i)})$$

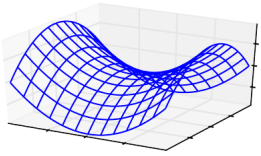
In general, then $H(\boldsymbol{\theta}) := \nabla^2_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) \not\succ 0$. Consequently,

- the inverse $H(\boldsymbol{\theta})^{-1}$ might not exist
- even if $H(\boldsymbol{\theta})^{-1}$ is defined, $-H(\boldsymbol{\theta})^{-1}\nabla f(\boldsymbol{\theta})$ might not be a descent direction!

Non-Convexity: Local Minima

- Weight space symmetry: swapping input and output weights of any two units in a layer preserves the same predictions
  - $\Rightarrow$ The model is not identifiable
    - But all of the symmetric models yield the same performance
- Existence of many poor local minima would be a problem
  - But these are fortunately not common in practice for deep models

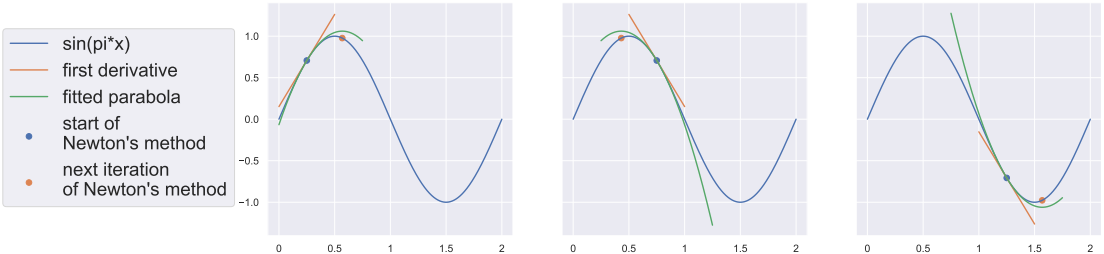
Non-Convexity: Saddle Points



- Local minimum: all eigenvalues of H are positive
- Saddle point: some eigenvalues of H are positive, some are negative
- In high dimensions, saddle points are exponentially more likely than local minima
- At low cost, local minima are more likely than at high cost
- Near saddle points, gradients are small
  - But empirically not a big problem for first order optimizers
- Saddle points are attractors for 2nd order methods $\Rightarrow$ substantial problem

[Image Source: Goodfellow et al., 2016, p. 88, fig 4.5]

Newton's Method is Attracted to Both Minimizers and Maximizers



- Gradient descent: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \nabla f_t(\boldsymbol{\theta}_t)$
- Newton's method: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \boldsymbol{H}^{-1} \nabla f_t(\boldsymbol{\theta}_t)$
- E.g., middle plot: near a maximizer, with negative curvature to the right of the maximizer:
  - Gradient descent moves away from the maximizer
  - Newton's method moves towards the maximizer

Large Training Datasets: Gradient Computation

The exact gradient sums over all n data points:

$$\boldsymbol{g}_k := \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} L(m(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), y^{(i)})$$

- Since n is very large, computing the exact gradient is very expensive

High Dimensionality & Large Training Datasets: Hessian Matrix

The exact Hessian sums over all n data points:

$$H(\boldsymbol{\theta}) := \nabla^2_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \nabla^2_{\boldsymbol{\theta}} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), y^{(i)}) \in \mathbb{R}^{d \times d}$$

- Since both n and d are very large: too expensive to compute
- Since d is very large: too big to store in memory

Structure of the Network: Vanishing and Exploding Gradients

- Illustrative example: consider a recurrent network that multiplies its input by a weight matrix $\boldsymbol{W}$ in each step
- Let $\boldsymbol{W}$ have the following singular value decomposition:

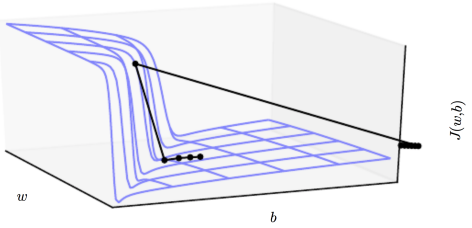
$$\boldsymbol{W} = \left(\boldsymbol{V}\mathrm{diag}(\boldsymbol{\lambda})\boldsymbol{V}^{-1}\right)$$

- Then, repeated multiplication by $\boldsymbol{W}$ yields:

$$\boldsymbol{W}^t = \left(\boldsymbol{V}\mathrm{diag}(\boldsymbol{\lambda})\boldsymbol{V}^{-1}\right)^t = \boldsymbol{V}\mathrm{diag}(\boldsymbol{\lambda})^t\boldsymbol{V}^{-1} \tag{8.11}$$

  - Gradients of this network will also be scaled by $\boldsymbol{\lambda}^t$
  - $\boldsymbol{\lambda} > 1$: exploding gradients
  - $\boldsymbol{\lambda} < 1$: vanishing gradients

Structure of the Network: Cliffs



- Cliffs result from multiplying several large weights together
- Most common in RNNs, because these involve multiplying many factors (see exploding gradients)
- Most serious consequences can be tackled with gradient clipping
  - Gradient only gives the right direction for an infinitesimal step
  - Too large steps likely lead to worsening

[Image Source: Goodfellow et al., 2016, p. 285, fig 8.3]

Questions to Answer for Yourself / Discuss with Friends

- Repetition: List the problems that make DL optimization hard
- Repetition: What are some reasons not to use 2nd order methods/the Hessian in practical deep learning?

Lecture Overview

Foundations of Deep Learning, Winter Term 2021/22

Week 4: Optimization of Neural Networks

Stochastic Gradient Descent (SGD) & Minibatch Sizes

Frank Hutter Abhinav Valada

University of Freiburg

Batch Gradient Descent vs Stochastic Gradient Descent

Batch Gradient Descent

- Initialize $\boldsymbol{\theta}$
- Update Loop (repeated as long as time allows):
  1. $\boldsymbol{g} \leftarrow \frac{1}{n} \nabla_\theta \sum_i^n L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}), \mathbf{y}^{(i)})$
  2. $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \boldsymbol{g}$
- Methods that use the entire dataset to compute gradients are called batch or deterministic gradient methods

Stochastic Gradient "Descent" (SGD)

- Initialize $\boldsymbol{\theta}$
- Update Loop:
  1. sample $\{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), \ldots, (\mathbf{x}^{(m)}, \mathbf{y}^{(m)})\}$
  2. $\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \nabla_\theta \sum_i^m L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}), \mathbf{y}^{(i)})$
  3. $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \hat{\boldsymbol{g}}$
- Methods that use mini-batches of datapoints to compute the gradients are called mini-batch or stochastic gradient methods
  - By now, a batch of data is often imprecisely used to mean a mini-batch (similarly batch size)

Reasons for Stochastic vs. Batch Gradient Descent

1. Diminishing returns in using many data points to estimate gradients
   - Error in estimating the expectation of a random variable Z with standard deviation σ, given m samples $z^{(i)}$:

$$SE(\mu_m) = \sqrt{Var[\frac{1}{m}\sum_{i=1}^{m} z^{(i)}]} = \frac{\sigma}{\sqrt{m}}$$

     $\Rightarrow$ Less than linear return for number of samples
     - E.g., 100$\times$ more samples: only 10$\times$ less noise

2. There is redundancy in the training set
   - Extreme example: all data points are copies of each other
     $\Rightarrow$ the gradient based on a single one would be exact
   - In practice: especially early on in the optimization, small mini batches already tend to point in the right direction

- Taken together: SGD usually makes faster progress by updating more often than batch gradient descent

Minibatch Sizes

- Multi-core architectures are usually underutilized with very small batchsize
  - $\Rightarrow$ Typically set batch size as large as possible given memory constraints
- Small batch sizes have a regularizing effect
- Batch size interacts with learning rate:
  small batch sizes require lower learning rates

Subtle Issue: Randomizing Mini-Batches

- Data collection process can introduce strong bias in successive data samples
  - consider data of blood samples, arranged successively by patient
- Updates for mini-batches of successive data points would be biased accordingly
  $\rightarrow$ poor convergence due to unnecessarily big oscillations in weight updates
- Solution in practice: balance mini-batches approximately by random shuffling of the training data
- Every pass over the entire training data is called an *epoch*, after which the data can be shuffled again.

Questions to Answer for Yourself / Discuss with Friends

- Repetition:
  Name 2 reasons that stochastic gradient descent is faster than batch gradient descent.
- Activation of what you just learned:
  Why does a small batch size work as a regularizer?
- Repetition:
  If you reduce your batch size, how should you change the learning rate?

Lecture Overview

Foundations of Deep Learning, Winter Term 2021/22

Week 4: Optimization of Neural Networks

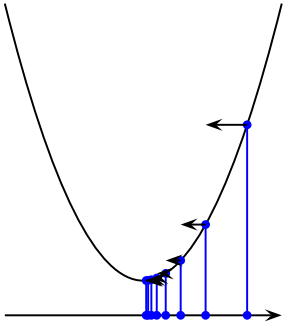
Choosing the Learning Rate

Frank Hutter Abhinav Valada

University of Freiburg

Problems With Suboptimal Choices for Learning Rate

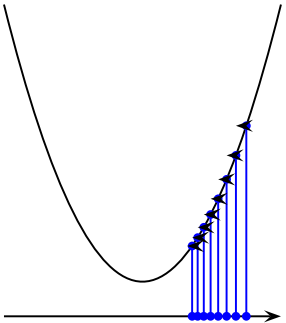
- choice of α
  1. case small α: convergence



Problems With Suboptimal Choices for Learning Rate

- choice of α

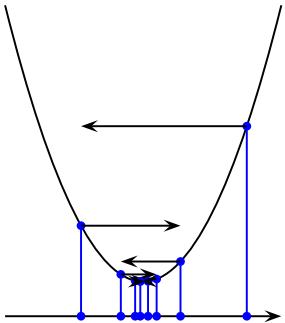
  2. case very small α: convergence, but it may take very long



Problems With Suboptimal Choices for Learning Rate

- choice of α

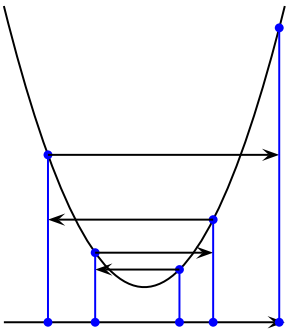
  3. case medium size α: convergence



Problems With Suboptimal Choices for Learning Rate

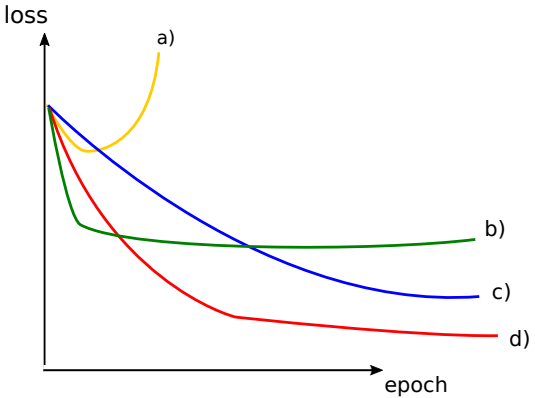
- choice of α

  4. case large α: divergence



Learning Rate Quiz

Which curve denotes low, high, very high, and good learning rate?



Using a Fixed Learning Rate is Usually not a Great Idea

- With a fixed learning rate α SGD will never settle (= converge)
- Sufficient condition for SGD to converge (in the limit of infinitely many updates):

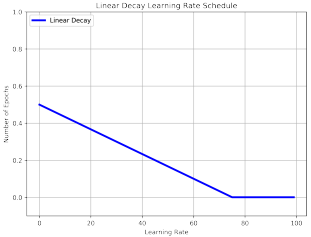
$$\begin{aligned} \sum_{k=1}^{\infty} \alpha_k &= \infty, \text{and} \\ \sum_{k=1}^{\infty} \alpha_k^2 &< \infty \end{aligned}$$

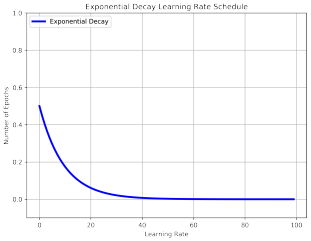
- For example, this can be achieved by setting $\alpha_k := \frac{1}{k}$

Some Standard Learning Rate Schedules in Practice

- Linear decay until iteration τ:
  $\alpha_k = (1-b)\alpha_0 + b\alpha_\tau$,
  with $b = k/\tau$. Then constant.

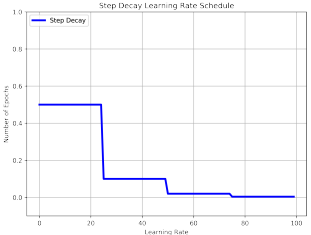
- Exponential decay: $\alpha_k = b\alpha_{k-1} = b^k\alpha_0$



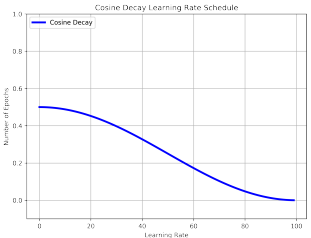


Some Standard Learning Rate Schedules in Practice

- Step decay: decay by a multiplicative factor (e.g., 10) every n epochs (or when no further progress is measured)

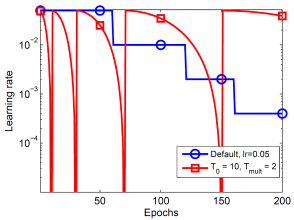


- Cosine decay: $\alpha_k = \frac{1}{2}(1 + cos(\frac{k}{n}\pi)) \times \alpha_0$, where n is the total number of epochs for which we will run SGD



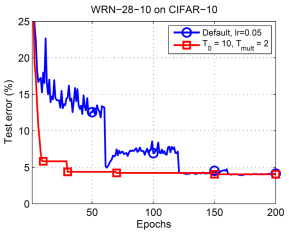
SGD with Warm Restarts (SGDR) [Loshchilov and Hutter, 2016]

- Iterative stages of convergence by aggressive learning rate schedules
  - Quickly cool α down to zero (converge to an OK solution)
  - Heat up α again
  - Cool down α more slowly (converge to a better solution)
- When restarting the learning rate: keep the weights



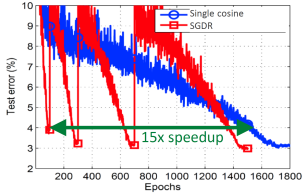
Learning rate schedules

red: SGDR
blue: step decay



Results (for points to be returned)

red: SGDR
blue: step decay



Results (for all intermediate points, on a larger network, given more time)

red: SGDR
blue: single cosine

Questions to Answer for Yourself / Discuss with Friends

- Repetition: What is a sufficient condition for SGD to converge?
- Repetition: List 4 popular learning rate schedules
- Repetition: How can you diagnose that your learning rate is too small / too large?

Lecture Overview

Foundations of Deep Learning, Winter Term 2021/22

Week 4: Optimization of Neural Networks

Stochastic Gradient Descent With Momentum

Frank Hutter Abhinav Valada

University of Freiburg

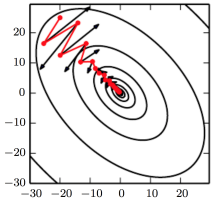
Momentum

Update step:

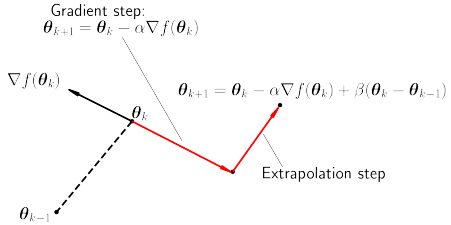
1. $\mathbf{v} \leftarrow \beta \mathbf{v} - \alpha \hat{\mathbf{g}}$
2. $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \mathbf{v}$

- Idea: make updates smoother by keeping a history
- Velocity $\mathbf{v}$ is an exponentially decaying moving average of past gradients
- The above is equivalent to

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \sum_{i=0}^{k} \beta^{k-i} \hat{\boldsymbol{g}}_i$$



Red: SGD with momentum;
Black: directions of SGD without momentum



[Image Sources: Goodfellow et al., 2016, p. 293, fig 8.5 & based on Figure 2.1.2 of Bertsekas: "Convex Optimization Algorithms"]

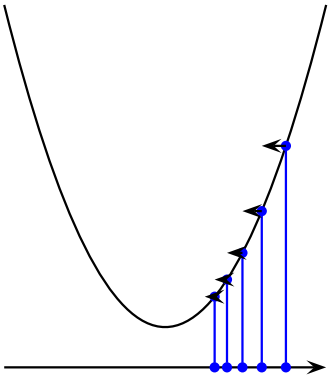
Pros and Cons of Momentum

Advantages of momentum:

- smoothes zig-zagging
- accelerates learning at flat spots
- slows down when signs of partial derivatives change

Disadvantages of momentum:

- additional parameter β
- may cause additional zig-zagging (with strong momentum)



vanilla gradient descent

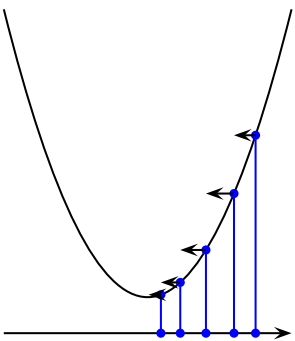
Pros and Cons of Momentum

Advantages of momentum:

- smoothes zig-zagging
- accelerates learning at flat spots
- slows down when signs of partial derivatives change

Disadvantages of momentum:

- additional parameter β
- may cause additional zig-zagging (with strong momentum)



gradient descent with momentum

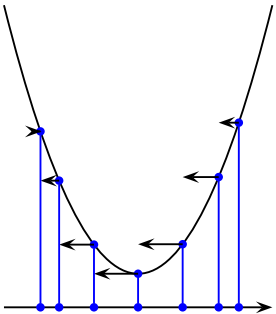
Pros and Cons of Momentum

Advantages of momentum:

- smoothes zig-zagging
- accelerates learning at flat spots
- slows down when signs of partial derivatives change

Disadvantages of momentum:

- additional parameter β
- may cause additional zig-zagging (with strong momentum)



gradient descent with strong momentum

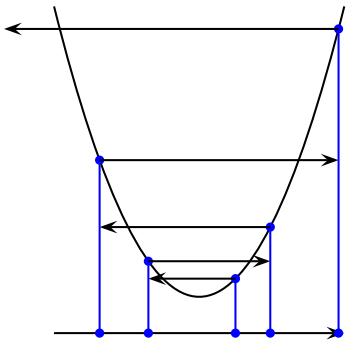
Pros and Cons of Momentum

Advantages of momentum:

- smoothes zig-zagging
- accelerates learning at flat spots
- slows down when signs of partial derivatives change

Disadvantages of momentum:

- additional parameter β
- may cause additional zig-zagging (with strong momentum)



vanilla gradient descent

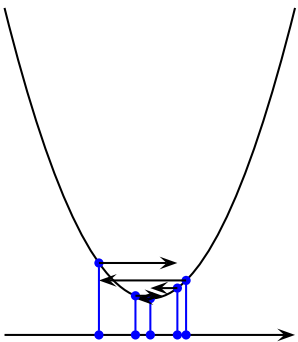
Pros and Cons of Momentum

Advantages of momentum:

- smoothes zig-zagging
- accelerates learning at flat spots
- slows down when signs of partial derivatives change

Disadvantages of momentum:

- additional parameter β
- may cause additional zig-zagging (with strong momentum)



gradient descent with momentum

Questions to Answer for Yourself / Discuss with Friends

- Repetition: Write down the update step for SGD with momentum.
- Repetition: How does the update step in SGD with momentum depend on the gradient we observed t steps ago?

Lecture Overview

Foundations of Deep Learning, Winter Term 2021/22

Week 4: Optimization of Neural Networks

Adaptive Gradient Algorithms

Frank Hutter Abhinav Valada

University of Freiburg

Motivation for Adaptive Gradient Algorithms

- Learning rate has to be adapted for convergence
- Different dimensions may require different learning rates
- We can't introduce a hyperparameter for each dimension
  $\rightarrow$ need to adapt learning rate based on the history
- Basically, we will use a diagonal, adaptive preconditioner matrix that can be seen as very roughly approximating the inverse Hessian

AdaGrad [Duchi et al., 2011]

- Update step:
  1. $\mathbf{r} \leftarrow \mathbf{r} + \hat{\mathbf{g}} \odot \hat{\mathbf{g}}$ // accumulate squared gradient
  2. $\Delta\boldsymbol{\theta} \leftarrow -\frac{\alpha}{\delta + \sqrt{\mathbf{r}}} \odot \hat{\mathbf{g}}$ // scale α by root of cumulative squared gradient
  3. $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

- δ is a small constant to avoid division by zero
- Here, and on the following slides, all operations are element-wise

- Initial gradient has longterm influences, reduces learning rate for the dimension
  - More progress in the dimensions with small slopes
  - Desirable theoretical properties for convex optimization
  - But empirically: sometimes premature and excessive decrease in the effective learning rate

RMSProp

- Improvement on AdaGrad, allowing the algorithm to forget its history
- Exponential moving average of squared gradient (instead of sum)

- Update step:
  1. $\mathbf{r} \leftarrow \rho\mathbf{r} + (1-\rho)\hat{\mathbf{g}} \odot \hat{\mathbf{g}}$ // exponential moving average of sq. gradient
  2. $\Delta\boldsymbol{\theta} \leftarrow -\frac{\alpha}{\delta+\sqrt{\mathbf{r}}} \odot \hat{\mathbf{g}}$
  3. $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

- δ is a small constant to avoid division by zero
- Decay rate ρ is a new hyperparameter controlling how quickly we forget

- Effective and commonly-used optimizer in deep learning
- Can be combined with momentum

Adam [Kingma and Ba, 2014]

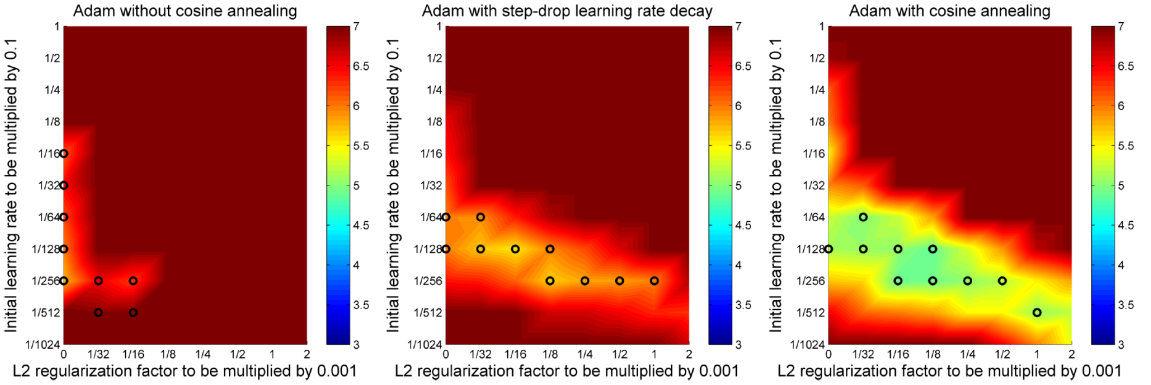
- Update step:
  1. $t \leftarrow t + 1$
  2. $\mathbf{s} \leftarrow \rho_1 \mathbf{s} + (1 - \rho_1)\hat{\mathbf{g}}$ // exponential moving average of gradient
  3. $\mathbf{r} \leftarrow \rho_2 \mathbf{r} + (1 - \rho_2)\hat{\mathbf{g}} \odot \hat{\mathbf{g}}$ // exponential moving average of sq. gradient
  4. $\hat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1-\rho_1^t}$ // correct bias in moving gradient estimate
  5. $\hat{\mathbf{r}} \leftarrow \frac{\mathbf{r}}{1-\rho_2^t}$ // correct bias in moving sq. gradient estimate
  6. $\Delta\boldsymbol{\theta} \leftarrow -\frac{\alpha}{\delta + \sqrt{\hat{\mathbf{r}}}} \odot \hat{\mathbf{s}}$
  7. $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

- δ is a small constant to avoid division by zero
- Decay rates ρ_1 and ρ_2 for first and second moment (gradient and squared gradient)

- Adam: short for Adaptive estimates of low-order moments
- Extension of RMSprop with momentum and bias correction.
- The default optimizer in large parts of deep learning

Learning Rate Schedules for Adaptive Gradient Algorithms

- Even for adaptive gradient algorithms like Adam, you definitely want to explore scheduling the base learning rate!



For more details, see Loshchilov & Hutter, 2017 (`https://arxiv.org/abs/1711.05101`)

[Image source: Loshchilov and Hutter, 2017]

Questions to Answer for Yourself / Discuss with Friends

- Repetition: How does AdaGrad use the historical squared gradient for a particular weight?

- Activation of what you just learned / trick question:
  So, Adam already adaptively sets its learning rate. Why would you need a learning rate schedule for it?

Lecture Overview

Foundations of Deep Learning, Winter Term 2021/22

Week 4: Optimization of Neural Networks

Further Reading, Summary of the Week, References

Frank Hutter Abhinav Valada

University of Freiburg

Summary by Learning Goals

Having heard this lecture, you can now ...

- describe in which ways learning differs from pure optimization
- describe ill-conditioning and possible solutions for it
- give reasons for using stochastic (rather than batch) gradient descent
- describe challenges in optimizing neural networks
- diagnose poor settings of the learning rate
- explain momentum and learning rate schedules
- motivate and contrast various adaptive gradient algorithms

- Further methods not covered here
  - Approximate second-order methods
  - Gradient-free optimization algorithms, e.g., evolution strategies

Further Reading

Read chapter 8 of the Deep Learning Book, which is the main source for this week's material.

References

Goodfellow, I., Bengio, Y., Courville, A. (2016)
Deep Learning
MIT Press.
https://www.deeplearningbook.org/

Loshchilov, I. and Hutter, F. (2017)
Decoupled weight decay regularization
arXiv preprint arXiv:1711.05101
https://arxiv.org/abs/1711.05101

Duchi, J., Hazan, E., Singer, Y. (2011)
Adaptive subgradient methods for online learning and stochastic optimization.
Journal of machine learning research 12
https://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf

Kingma, D. and Ba, J. (2014)
Adam: A method for stochastic optimization
arXiv preprint arXiv:1412.6980
https://arxiv.org/pdf/1412.6980.pdf

Loshchilov, I. and Hutter, F. (2016)
SGDR: Stochastic gradient descent with warm restarts
arXiv preprint arXiv:1608.03983
https://arxiv.org/pdf/1608.03983.pdf

Torrey, L. and Shavlik, J. (2010)
Transfer learning
Handbook of research on machine learning applications and trends: algorithms, methods, and techniques , 242–264
https://ftp.cs.wisc.edu/machine-learning/shavlik-group/torrey.handbook09.pdf

Finn, C., Abbeel, P., Levine, S. (2017)
Model-agnostic meta-learning for fast adaptation of deep networks
arXiv preprint arXiv:1703.03400
https://arxiv.org/pdf/1703.03400.pdf

Noroozi, M., Vinjimoor, A., Favaro, P., Pirsiavash, H. (2018)
Boosting self-supervised learning via knowledge transfer
Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition , 9359–9367
https://openaccess.thecvf.com/content_cvpr_2018/papers/Noroozi_Boosting_Self-Supervised_Learning_CVPR_2018_paper.pdf