SiTE
AAiT
AAU

Course Title: Deep Learning
Credit Hour: 3
Instructor: Fantahun B. (PhD) ✉ meetfantaai@gmail.com
Office: NB #

Ch-2A Optimization for Deep Learning

[Based on the text book]

Jul-2023, AA

Optimization for Deep Learning (DL)

Agenda:

- How Learning Differs from Pure Optimization
- Challenges in Neural Network Optimization
- Basic Algorithms
- Parameter Initialization Strategies
- Algorithms with Adaptive Learning Rates
- Approximate Second-Order Methods

Optimization for Deep Learning (DL)

Objectives

After completing this chapter students will be able to:

- Compare and contrast pure optimization and optimization in ML
- Identify and discuss some challenges in neural network optimization
- Review some basic optimization algorithms
- Try to conceptualize some heuristics for parameter initialization
- Appreciate the role of algorithms with adaptive learning rates
- Discuss some approximate second order methods as a means of optimization
- Setup a learning algorithm with appropriate optimization technique and initial parameters.

Optimization for DL

- Terminology
  - *Epoch* – The number of times the algorithm runs on the whole training dataset.
  - *Sample/Example* – A single row of a dataset.
  - *Batch* – It denotes the number of samples to be taken to for updating the model parameters.
  - *Learning rate* – It is a parameter that provides the model a scale of how much model weights should be updated.
  - *Cost Function/Loss Function* – A cost function is used to calculate the cost, which is the difference between the predicted value and the actual value.
  - *Weights/ Bias* – The learnable parameters in a model that controls the signal between two neurons.
  - *Parameter/Hyperparameter* –

Optimization for DL

- Of all of the many optimization problems involved in deep learning, the most difficult is neural network training.
  - It is customary to invest days to months on hundreds of machines to solve even a single instance of the neural network training problem.
- Because this problem is so important and so expensive, a specialized set of optimization techniques have been developed for solving it.
- We will focus on one particular case of optimization: finding the parameters θ of a neural network that significantly reduce a cost function $J(\theta)$,
  - which typically includes a performance measure evaluated on the entire training set as well as additional regularization terms.

Optimization for DL: How Learning Differs from Pure Optimization

- Optimization algorithms used for training of deep models differ from traditional optimization algorithms in several ways.
  - In most ML scenarios, we care about some performance measure P, that is defined with respect to the test set and may also be intractable. We therefore optimize P only indirectly. We reduce a different cost function $J(\theta)$ in the hope that doing so will improve P.
  - This is in contrast to pure optimization, where minimizing J is a goal in and of itself.

Optimization for DL: How Learning Differs from Pure Optimization

- Typically, the cost function can be written as an average over the training set, such as

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

where

- L is the per-example loss function,
- f(x; θ) is the predicted output when the input is x,
- p̂data is the empirical distribution.
- y is the target output (in the supervised case).

Optimization for DL: How Learning Differs from Pure Optimization

- Eq. 8.1 defines an objective function with respect to the training set.
- We would usually prefer to minimize the corresponding objective function where the expectation is taken across the data generating distribution *pdata* rather than just over the finite training set:

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

Optimization for DL: How Learning Differs from Pure Optimization Empirical Risk Minimization

- The goal of a machine learning algorithm is to reduce the expected generalization error given by Eq. 8.2.
  - This quantity is known as the **risk**.
- We emphasize here that the expectation is taken over the true underlying distribution pdata.
  - If we knew the true distribution pdata (x, y), risk minimization would be an optimization task solvable by an optimization algorithm.
  - However, when we do not know pdata(x, y) but only have a training set of samples, we have a machine learning problem.

Optimization for DL: How Learning Differs from Pure Optimization

Empirical Risk Minimization

- The simplest way to *convert a ML problem to an optimization problem* is to minimize the expected loss on the training set.
- This means replacing the true distribution p(x, y) with the empirical distribution p̂(x, y) defined by the training set.
- We now minimize the empirical risk

$$\mathbb{E}_{\boldsymbol{x},\mathrm{y}\sim\hat{p}_{\mathrm{data}}(\boldsymbol{x},y)}[L(f(\boldsymbol{x};\boldsymbol{\theta}),y)] = \frac{1}{m}\sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}),y^{(i)}) \qquad (8.3)$$

where m is the number of training examples.

Optimization for DL: How Learning Differs from Pure Optimization
Empirical Risk Minimization

- The training process based on minimizing this average training error is known as empirical risk minimization.
- *Rather than optimizing the risk directly, we optimize the empirical risk, and hope that the risk decreases significantly as well.*
- A variety of theoretical results establish conditions under which the true risk can be expected to decrease by various amounts.
- However, empirical risk minimization is prone to overfitting.

Optimization for DL: How Learning Differs from Pure Optimization
Empirical Risk Minimization

- The most effective modern optimization algorithms are based on ***gradient descent***, but many useful loss functions, such as 0-1 loss, have no useful derivatives (the derivative is either zero or undefined everywhere).
  - These two problems mean that, in the context of deep learning, we rarely use empirical risk minimization.
  - Instead, we must use a slightly different approach, in which the quantity that we actually optimize is even more different from the quantity that we truly want to optimize.

Optimization for DL: How Learning Differs from Pure Optimization Surrogate Loss Functions and Early Stopping

- Sometimes, the loss function we actually care about (say classification error) is not one that can be optimized efficiently.
  - For example, exactly minimizing expected 0-1 loss is typically intractable (exponential in the input dimension), even for a linear classifier (Marcotte and Savard, 1992).
  - What is 0-1 loss?
- In such situations, one typically optimizes a surrogate loss function instead, which acts as a proxy but has advantages.
  - For example, the negative log-likelihood of the correct class is typically used as a surrogate for the 0-1 loss.
  - The negative log-likelihood allows the model to estimate the conditional probability of the classes, given the input, and if the model can do that well, then it can pick the classes that yield the least classification error in expectation.

Optimization for DL: How Learning Differs from Pure Optimization

Surrogate Loss Functions and Early Stopping

- In some cases, a surrogate loss function is able to learn more.
  - For example, the test set 0-1 loss often continues to decrease for a long time after the training set 0-1 loss has reached zero, when training using the log-likelihood surrogate.
  - This is because even when the expected 0-1 loss is zero, one can improve the robustness of the classifier by further pushing the classes apart from each other, obtaining a more confident and reliable classifier, thus extracting more information from the training data than would have been possible by simply minimizing the average 0-1 loss on the training set.

Optimization for DL: How Learning Differs from Pure Optimization

Surrogate Loss Functions and Early Stopping

- A very important difference between optimization in general and optimization as we use it for training algorithms is that training algorithms do not usually halt at a local minimum.
  - Instead, a machine learning algorithm usually minimizes a surrogate loss function but halts when a convergence criterion based on early stopping (Sec. 7.8) is satisfied.
  - Typically the early stopping criterion is based on the true underlying loss function, such as 0-1 loss measured on a validation set, and is designed to cause the algorithm to halt whenever overfitting begins.
  - Training often halts while the surrogate loss function still has large derivatives, which is very different from the pure optimization setting, where an optimization algorithm is considered to have converged when the gradient becomes very small.

Optimization for DL: How Learning Differs from Pure Optimization
Batch and Minibatch Algorithms

- One aspect of machine learning algorithms that separates them from general optimization algorithms is that the objective function usually decomposes as a sum over the training examples.
  - Optimization algorithms for machine learning typically compute each update to the parameters based on an expected value of the cost function estimated using only a subset of the terms of the full cost function.
  - For example, maximum likelihood estimation problems, when viewed in log space, decompose into a sum over each example:

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p_{\mathrm{model}}(\boldsymbol{x}^{(i)}, y^{(i)}; \boldsymbol{\theta}). \tag{8.4}$$

Optimization for DL: How Learning Differs from Pure Optimization
Batch and Minibatch Algorithms

- Maximizing this sum is equivalent to maximizing the expectation over the empirical distribution defined by the training set:

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x},\mathrm{y}\sim\hat{p}_{\mathrm{data}}} \log p_{\mathrm{model}}(\boldsymbol{x}, y; \boldsymbol{\theta}). \tag{8.5}$$

- Most of the properties of the objective function J used by most of our optimization algorithms are also expectations over the training set. For example, the most commonly used property is the gradient:

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

Optimization for DL: How Learning Differs from Pure Optimization Batch and Minibatch Algorithms

- Computing this expectation exactly is very expensive as it requires evaluating the model on every example in the entire dataset.
- In practice, we can compute these expectations by randomly sampling a small number of examples from the dataset, then taking the average over only those examples.
  - Batch/deterministic gradient methods: optimization algorithms that use the entire training set.
  - Stochastic/Online methods: optimization algorithms that use only a single example at a time.
  - Minibatch/Minibach stochastic/Stochastic methods: use more than one but less than all examples. (Most algorithms used for DL fall here).

Optimization for DL: How Learning Differs from Pure Optimization
Batch and Minibatch Algorithms

- Minibatch sizes are generally driven by the following factors:
  - Larger batches provide a more accurate estimate of the gradient, but with less than linear returns.
  - Extremely small batches, usually, underutilize multicore architectures. This motivates using some absolute minimum batch size, below which there is no reduction in the time to process a minibatch.
  - If all examples in the batch are to be processed in parallel (as is typically the case), then the amount of memory scales with the batch size. For many hardware setups this is the limiting factor in batch size.
  - Some kinds of hardware achieve better runtime with specific sizes of arrays. Especially when using GPUs, it is common for power of 2 batch sizes (ranging from 32 to 256, 16 sometimes tried for large models).
  - Small batches can offer a regularizing effect (Wilson and Martinez, 2003), perhaps due to the noise they add to the learning process. Generalization error is often best for a batch size of 1.

Optimization for DL: How Learning Differs from Pure Optimization
Batch and Minibatch Algorithms

- Different kinds of algorithms use different kinds of information from the minibatch in different ways.
  - Some algorithms are more sensitive to sampling error than others, either because they use information that is difficult to estimate accurately with few samples, or because they use information in ways that amplify sampling errors more.
    - Methods that compute updates based only on the gradient g are usually relatively robust and can handle smaller batch sizes like 100.
    - Second-order methods, which use also the Hessian matrix H and compute updates such as $H^{-1}g$, typically require much larger batch sizes like 10,000. These large batch sizes are required to minimize fluctuations in the estimates of $H^{-1}g$.

Optimization for DL: How Learning Differs from Pure Optimization Batch and Minibatch Algorithms

- It is also crucial that the minibatches be selected randomly.
  - Computing an unbiased estimate of the expected gradient from a set of samples requires that those samples be independent.
  - We also wish for two subsequent gradient estimates to be independent from each other, so two subsequent minibatches of examples should also be independent from each other.
- Many datasets are most naturally arranged in a way where successive examples are highly correlated.
  - Hence, shuffling is required.

Optimization for DL: Challenges in Neural Network Optimization

- Optimization in general is an extremely difficult task.
  - Traditionally, machine learning has avoided the difficulty of general optimization by carefully designing the objective function and constraints to ensure that the optimization problem is convex.
  - When training neural networks, we must confront the general non-convex case.
  - Even convex optimization is not without its complications.
- In this section, we summarize several of the most prominent challenges involved in optimization for training deep models.

Optimization for DL: Challenges in Neural Network Optimization

1. Ill-Conditioning

- Some challenges arise even when optimizing convex functions. Of these, the most prominent is ill-conditioning of the Hessian matrix H.
- This is a very general problem in most numerical optimization, convex or otherwise (see section 4.3.1).
- The ill-conditioning problem is generally believed to be present in neural network training problems.
- Ill-conditioning can manifest by causing SGD to get "stuck" in the sense that even very small steps increase the cost function.

Optimization for DL: Challenges in Neural Network Optimization

1. Ill-Conditioning

- Recall from Eq. 4.9 that a second-order Taylor series expansion of the cost function predicts that a gradient descent step of $-\epsilon g$ will add 8.10 to the cost.

$$\frac{1}{2}\epsilon^2 \boldsymbol{g}^\top \boldsymbol{H}\boldsymbol{g} - \epsilon \boldsymbol{g}^\top \boldsymbol{g} \tag{8.10}$$

- Ill-conditioning of the gradient becomes a problem when $1/2\epsilon^2 g^T H g$ exceeds $\epsilon g^T g$.
- To determine whether ill-conditioning is detrimental to a neural network training task, one can monitor the squared gradient norm $g^T g$ and the $g^T H g$ term.

Optimization for DL: Challenges in Neural Network Optimization

1. Ill-Conditioning

- In many cases, the gradient norm does not shrink significantly throughout learning, but the $g^T Hg$ term grows by more than order of magnitude.
- The result is that learning becomes very slow despite the presence of a strong gradient because the learning rate must be shrunk to compensate for even stronger curvature.
- Fig. 8.1 shows an example of the gradient increasing significantly during the successful training of a neural network.

Optimization for DL: Challenges in Neural Network Optimization

1. Ill-Conditioning

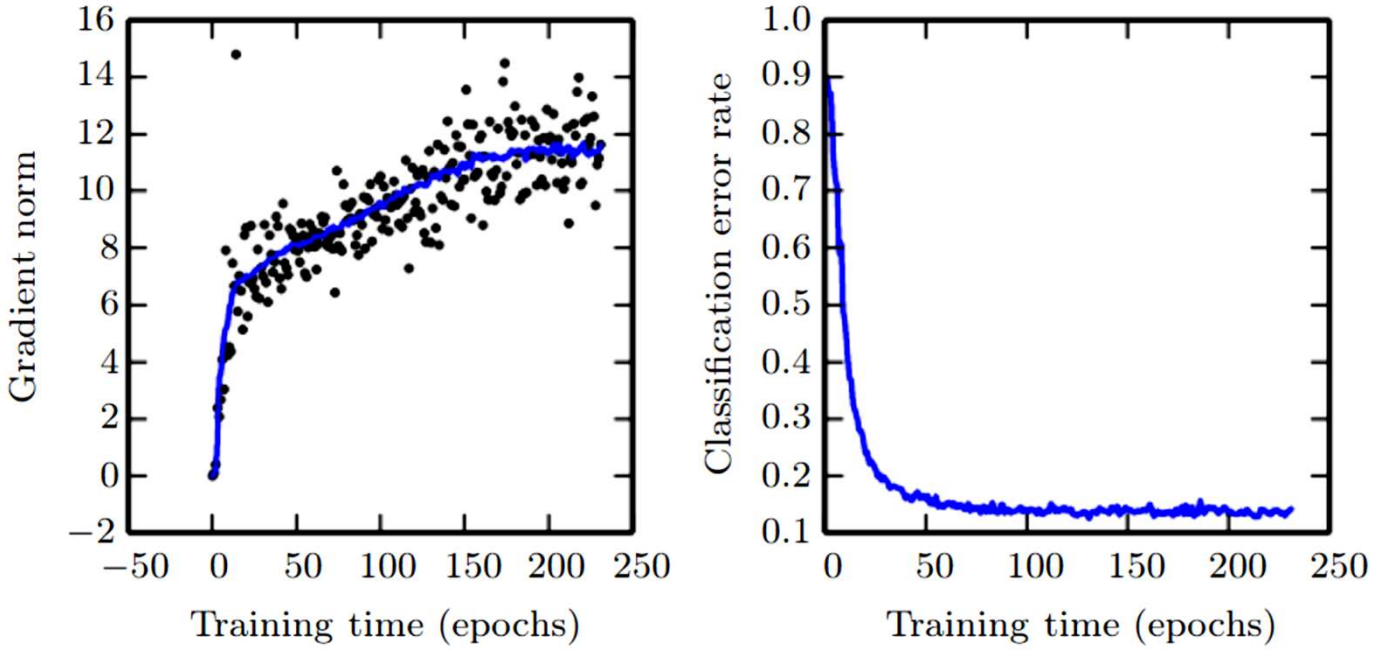


Fig. 8.1 shows an example of the gradient increasing significantly during the successful training of a neural network.

Optimization for DL: Challenges in Neural Network Optimization

2. Local Minima

- One of the most prominent features of a convex optimization problem is that it can be reduced to the problem of finding a local minimum.
  - Any local minimum is guaranteed to be a global minimum.
  - Some convex functions have a flat region at the bottom rather than a single global minimum point, but any point within such a flat region is an acceptable solution.
  - When optimizing a convex function, we know that we have reached a good solution if we find a critical point of any kind.

Optimization for DL: Challenges in Neural Network Optimization
2. Local Minima

- With *non-convex functions*, such as neural nets, it is possible to have *many local minima*.
- Indeed, nearly any deep model is essentially guaranteed to have an extremely large number of local minima. However, this is not necessarily a major problem.
- Neural networks and any models with multiple equivalently parametrized latent variables all have multiple local minima because of the *model identifiability* problem.
  - A model is said to be identifiable if a sufficiently large training set can rule out all but one setting of the model's parameters.

Optimization for DL: Challenges in Neural Network Optimization

2. Local Minima

- Weight space symmetry: Models with latent variables are often not identifiable because we can obtain equivalent models by exchanging latent variables with each other.
  - For example, we could take a neural network and modify layer 1 by swapping the incoming weight vector for unit i with the incoming weight vector for unit j, then doing the same for the outgoing weight vectors.
  - If we have m layers with n units each, then there are $n!^m$ ways of arranging the hidden units. This kind of non-identifiability is known as *weight space symmetry*.

Optimization for DL: Challenges in Neural Network Optimization

2. Local Minima

- In addition to weight space symmetry, many kinds of neural networks have additional causes of non-identifiability.
  - For example, in any rectified linear or maxout network, we can scale all of the incoming weights and biases of a unit by α if we also scale all of its outgoing weights by $1/\alpha$. This means that—if the cost function does not include terms such as weight decay that depend directly on the weights rather than the models' outputs—every local minimum of a rectified linear or maxout network lies on an (m× n)-dimensional hyperbola of equivalent local minima.
- These model identifiability issues mean that there can be an extremely large or even uncountably infinite amount of local minima in a neural network cost function. However, all of these local minima arising from non-identifiability are equivalent to each other in cost function value. As a result, these local minima are not a problematic form of non-convexity.

Optimization for DL: Challenges in Neural Network Optimization

2. Local Minima

- Local minima can be problematic if they have high cost than the global minimum.
- One can construct small neural networks, even without hidden units, that have local minima with higher cost than the global minimum (Sontag and Sussman, 1989; Brady et al., 1989; Gori and Tesi, 1992).
- If local minima with high cost are common, this could pose a serious problem for gradient-based optimization algorithms.
- It remains an open question whether there are many local minima of high cost for networks of practical interest and whether optimization algorithms encounter them.

Optimization for DL: Challenges in Neural Network Optimization

2. Local Minima

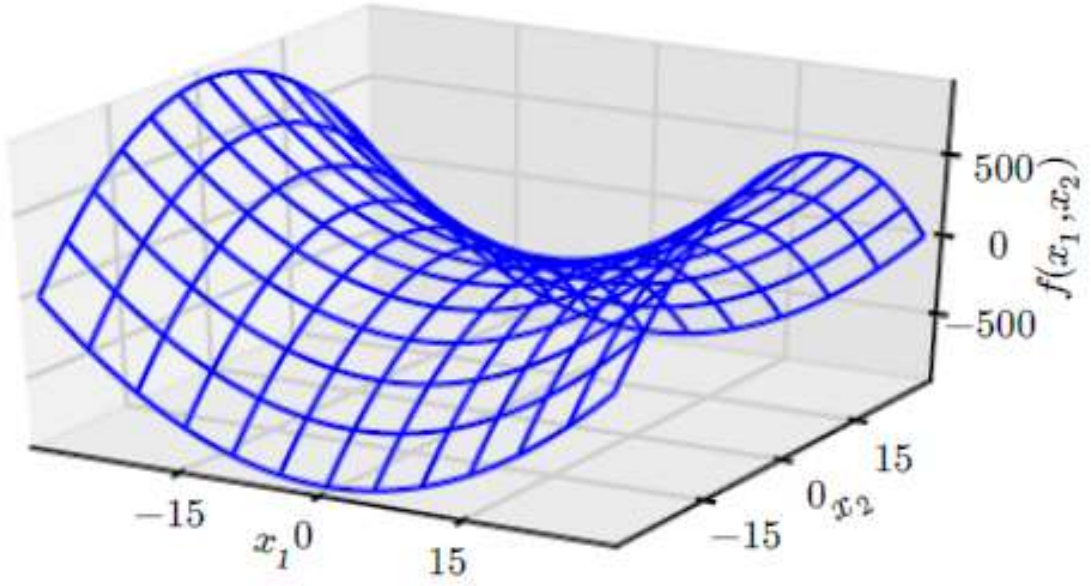
- Experts now suspect that, for sufficiently large neural networks, most local minima have a low cost function value, and that it is not important to find a true global minimum rather than to find a point in parameter space that has low but not minimal cost (Saxe et al., 2013; Dauphin et al., 2014; Goodfellow et al., 2015; Choromanska et al., 2014).
- A test that can rule out local minima as the problem is to plot the norm of the gradient over time.
  - If the norm of the gradient does not shrink to insignificant size, the problem is neither local minima nor any other kind of critical point.

Optimization for DL: Challenges in Neural Network Optimization

3. Plateaus, Saddle Points and Other Flat Regions

- For many high-dimensional non-convex functions, local minima (and maxima) are in fact rare compared to another kind of *point with zero gradient: a saddle point*.
  - Some points around a saddle point have greater cost than the saddle point, while others have a lower cost.
  - At a saddle point, the Hessian matrix has both positive and negative eigenvalues.
  - Points lying along eigenvectors associated with positive eigenvalues have greater cost than the saddle point, while points lying along negative eigenvalues have lower value.
- We can think of a saddle point as being a local minimum along one cross-section of the cost function and a local maximum along another cross-section. (See Fig. 4.5 for an illustration).

Optimization for DL: Challenges in Neural Network Optimization
3. Plateaus, Saddle Points and Other Flat Regions



From (p-90) Fig. 4.5: A saddle point containing both positive and negative curvature. The function in this example is $f(x) = x_1^2 - x_2^2$.

- Along the axis corresponding to x_1, the function curves upward. This axis is an eigenvector of the Hessian and has a positive eigenvalue.
- Along the axis corresponding to x_2, the function curves downward. This direction is an eigenvector of the Hessian with negative eigenvalue.

Optimization for DL: Challenges in Neural Network Optimization

3. Plateaus, Saddle Points and Other Flat Regions

- Many classes of random functions exhibit the following behavior:
  - In low dimensional spaces, local minima are common.
  - In higher dimensional spaces, local minima are rare and saddle points are more common.
  - For a function $f : R^n \rightarrow R$ of this type, the expected ratio of the number of saddle points to local minima grows exponentially with n.
- The Hessian matrix at a local minimum has only positive eigenvalues.
- The Hessian matrix at a saddle point has a mixture of positive and negative eigenvalues.

Optimization for DL: Challenges in Neural Network Optimization

3. Plateaus, Saddle Points and Other Flat Regions

- This happens for many classes of random functions. Does it happen for neural networks?
  - Baldi and Hornik (1989) showed theoretically that shallow autoencoders (feedforward networks trained to copy their input to their output)with no nonlinearities have global minima and saddle points but no local minima with higher cost than the global minimum.
  - They observed without proof that these results extend to deeper networks without nonlinearities.
  - Dauphin et al. (2014) showed experimentally that real neural networks also have loss functions that contain very many high-cost saddle points.
  - Choromanska et al. (2014) provided additional theoretical arguments, showing that another class of high-dimensional random functions related to neural networks does so as well.

Optimization for DL: Challenges in Neural Network Optimization

3. Plateaus, Saddle Points and Other Flat Regions

- What are the implications of the proliferation of saddle points for training algorithms?
  - For first-order optimization algorithms that use only gradient information, the situation is unclear: The gradient can often become very small near a saddle point. On the other hand, gradient descent empirically seems to be able to escape saddle points in many cases.
  - For second-order methods, the proliferation of saddle points in high dimensional spaces make them unsuccessful – that seems why they didn't replace gradient descent for neural network training.

Optimization for DL: Challenges in Neural Network Optimization

3. Plateaus, Saddle Points and Other Flat Regions

- What are the implications of the proliferation of saddle points for training algorithms? …
  - For Newton's method, it is clear that saddle points constitute a problem. Gradient descent is designed to move "downhill" and is not explicitly designed to seek a critical point. Newton's method, however, is designed to solve for a point where the gradient is zero. Without appropriate modification, it can jump to a saddle point.
    - Dauphin et al. (2014) introduced a saddle-free Newton method for second-order optimization and showed that it improves significantly over the traditional version. Second-order methods remain difficult to scale to large neural networks, but this saddle-free approach holds promise if it could be scaled.

Optimization for DL: Challenges in Neural Network Optimization

3. Plateaus, Saddle Points and Other Flat Regions

- There are other kinds of points with zero gradient besides minima and saddle points.
- There are also maxima, which are much like saddle points from the perspective of optimization—many algorithms are not attracted to them, but unmodified Newton's method is.
- Maxima become exponentially rare in high dimensional space, just like minima do.

Optimization for DL: Challenges in Neural Network Optimization

3. Plateaus, Saddle Points and Other Flat Regions

- There may also be *wide, flat regions of constant value*.
- In these locations, the gradient and also the Hessian are all zero.
- Such degenerate locations pose major problems for all numerical optimization algorithms.
- In a convex problem, a wide, flat region must consist entirely of global minima, but in a general optimization problem, such a region could correspond to a high value of the objective function.

Optimization for DL: Challenges in Neural Network Optimization

4. Cliffs and Exploding Gradients

- Neural networks with many layers often have *extremely steep regions* resembling *cliffs*, as illustrated in Fig. 8.3.
- These result from the multiplication of several large weights together.
  - Cliff structures are most common in the cost functions for RNNs, because such models involve a multiplication of many factors, with one factor for each time step.
- On the face of an extremely steep cliff structure, the gradient update step can move the parameters extremely far, usually jumping off of the cliff structure altogether.
- The cliff can be dangerous whether we approach it from above or from below.

Optimization for DL: Challenges in Neural Network Optimization
4. Cliffs and Exploding Gradients

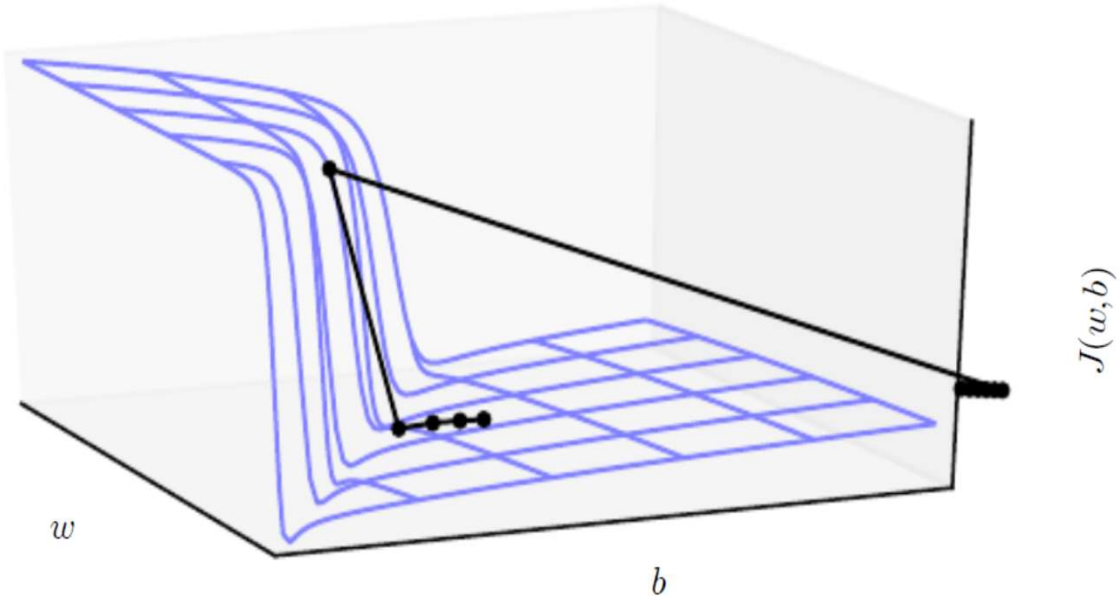


Figure 8.3: The objective function for highly nonlinear deep neural networks or for RNNs often contains sharp nonlinearities in parameter space resulting from the multiplication of several parameters. These nonlinearities give rise to very high derivatives in some places. When the parameters get close to such a cliff region, a gradient descent update can catapult the parameters very far, possibly losing most of the optimization work that had been done.

Optimization for DL: Challenges in Neural Network Optimization

4. Cliffs and Exploding Gradients

- Fortunately its most serious consequences can be avoided using the *gradient clipping heuristic*.
- The basic idea is to recall that the gradient does not specify the optimal step size, but *only the optimal direction* within an infinitesimal region.
- When the traditional gradient descent algorithm proposes to make a very large step, the gradient clipping heuristic intervenes to reduce the step size to be small enough that it is less likely to go outside the region where the gradient indicates the direction of approximately steepest descent.

Optimization for DL: Challenges in Neural Network Optimization

4. Cliffs and Exploding Gradients

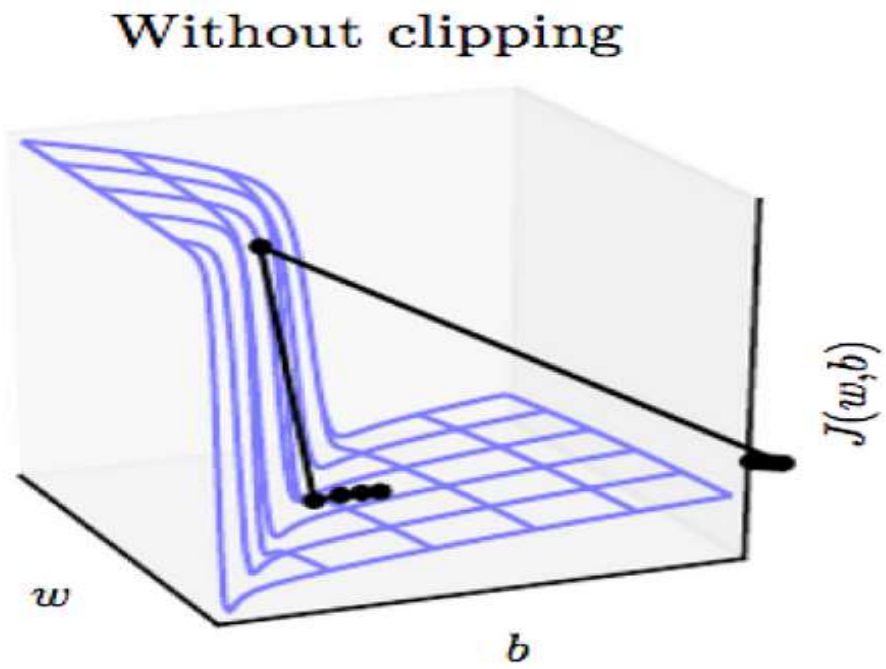
- Example:
- If the gradient gets too large, we rescale it to keep it small.
- More precisely, if $\|g\| \geq c$, then

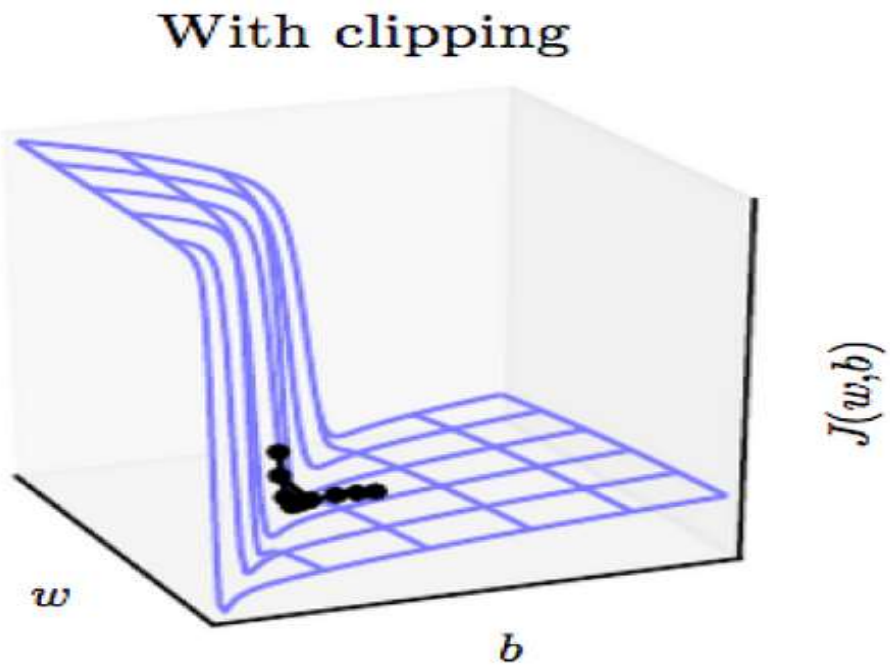
$$\mathbf{g} \leftarrow c \cdot \mathbf{g}/\|\mathbf{g}\|$$

where c is a hyperparameter, g is the gradient, and $\|g\|$ is the norm of g.

- Since $g/\|g\|$ is a unit vector, after rescaling the new g will have norm c.
- Note that if $\|g\| < c$, then we don't need to do anything.

Optimization for DL: Challenges in Neural Network Optimization
4. Cliffs and Exploding Gradients





Optimization for DL: Challenges in Neural Network Optimization

5. Long-Term Dependencies

- Another difficulty that neural network optimization algorithms must overcome arises when the *computational graph* becomes *extremely deep*.
- Feedforward networks with many layers have such deep computational graphs.
  - RNNs construct very deep computational graphs by repeatedly applying the same operation at each time step of a long temporal sequence.
  - Repeated application of the same parameters gives rise to especially pronounced difficulties.

Optimization for DL: Challenges in Neural Network Optimization

5. Long-Term Dependencies

- For example, suppose that a computational graph contains a path that consists of repeatedly multiplying by a matrix W. After t steps, this is equivalent to multiplying by W^t.
- Suppose that W has an eigen decomposition $W = V \text{ diag}(\lambda) V^{-1}$.
- In this simple case, it is straightforward to see that

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

- Any eigenvalues λ_i that are not near an absolute value of 1 will either explode if they are greater than 1 in magnitude or vanish if they are less than 1 in magnitude.

Optimization for DL: Challenges in Neural Network Optimization

5. Long-Term Dependencies

- The *vanishing* and *exploding gradient problem* refers to the fact that gradients through such a graph are also scaled according to $\text{diag}(\lambda)^t$.
- Vanishing gradients:
  - make it *difficult to know which direction the parameters should move* to improve the cost function,
  - exploding gradients can make *learning unstable*.
- The cliff structures described earlier that motivate gradient clipping are an example of the exploding gradient phenomenon.

Optimization for DL: Challenges in Neural Network Optimization

6. Inexact Gradients

- Most optimization algorithms are primarily motivated by the case where we have exact knowledge of the gradient or Hessian matrix.
- In practice, we usually only have a noisy or even biased estimate of these quantities.
  - Nearly every deep learning algorithm relies on sampling-based estimates at least insofar as using a minibatch of training examples to compute the gradient.
  - In other cases, the objective function we want to minimize is actually intractable. When the objective function is intractable, typically its gradient is intractable as well.
  - In such cases we can only approximate the gradient.
- These issues mostly arise with the more advanced models.

Optimization for DL: Challenges in Neural Network Optimization

6. Inexact Gradients

- For example, contrastive divergence gives a technique for approximating the gradient of the intractable log-likelihood of a Boltzmann machine.
- Various neural network optimization algorithms are designed to account for imperfections in the gradient estimate.
- One can also avoid the problem by choosing a surrogate loss function that is easier to approximate than the true loss.

Optimization for DL: Challenges in Neural Network Optimization

7. Poor Correspondence between Local and Global Structure

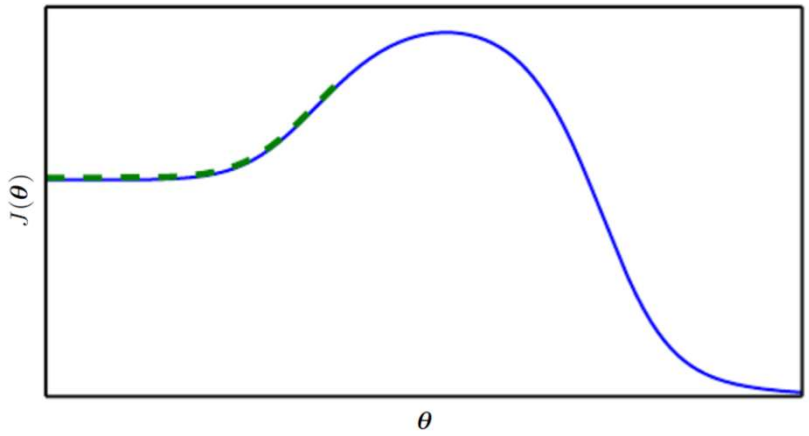
- Many of the problems we have discussed so far correspond to properties of the loss function at a single point
  - it can be difficult to make a single step if $J(\theta)$ is poorly conditioned at the current point θ, or if θ lies on a cliff, or if θ is a saddle point hiding the opportunity to make progress downhill from the gradient.
- It is possible to overcome all of these problems at a single point and still perform poorly if the direction that results in the most improvement locally does not point toward distant regions of much lower cost.
- Goodfellow et al. (2015) argue that much of the runtime of training is due to the length of the trajectory needed to arrive at the solution.

Optimization for DL: Challenges in Neural Network Optimization

7. Poor Correspondence between Local and Global Structure

- Much of research into the difficulties of optimization has focused on whether training arrives at a global minimum, a local minimum, or a saddle point, but in practice neural networks do not arrive at a critical point of any kind.
- Fig. 8.4 shows an example of a failure of local optimization to find a good cost function value even in the absence of any local minima or saddle points.

Figure 8.4: Optimization based on local downhill moves can fail if the local surface does not point toward the global solution. Here we provide an example of how this can occur, even if there are no saddle points and no local minima. This example cost function contains only asymptotes toward low values, not minima. The main cause of difficulty in this case is being initialized on the wrong side of the "mountain" and not being able to traverse it. In higher dimensional space, learning algorithms can often circumnavigate such mountains but the trajectory associated with doing so may be long and result in excessive training time.



Optimization for DL: Challenges in Neural Network Optimization

7. Poor Correspondence between Local and Global Structure

- Indeed, such critical points do not even necessarily exist.
  - For example, the loss function $-\log p(y \mid x; \theta)$ can lack a global minimum point and instead asymptotically approach some value as the model becomes more confident.
- Future research will need to develop further understanding of the factors that influence the length of the learning trajectory and better characterize the outcome of the process.
- Many existing research directions are aimed at finding good initial points for problems that have difficult global structure, rather than developing algorithms that use non-local moves.

Optimization for DL: Challenges in Neural Network Optimization

7. Poor Correspondence between Local and Global Structure

- Sometimes local information provides us no guide, when the function has a wide flat region, or if we manage to land exactly on a critical point (eg. Newton's method).
  - In these cases, local descent does not define a path to a solution at all.
  - In other cases, local moves can be too greedy and lead us along a path that moves downhill but away from any solution.
- Currently, we do not understand which of these problems are most relevant to making neural network optimization difficult, and this is an active area of research.

Optimization for DL: Challenges in Neural Network Optimization

7. Poor Correspondence between Local and Global Structure

- Regardless of which of these problems are most significant, all of them might be avoided
  - if there exists a region of space connected reasonably directly to a solution by a path that local descent can follow, and
  - if we are able to initialize learning within that well-behaved region.
- This last view suggests research into choosing good initial points for traditional optimization algorithms to use.

Optimization for DL: Challenges in Neural Network Optimization

8. Theoretical Limits of Optimization

- Several theoretical results show that there are limits on the performance of any optimization algorithm we might design for neural networks (Blum and Rivest, 1992; Judd, 1989; Wolpert and MacReady, 1997).
- Typically these results have little bearing on the use of neural networks in practice.
- Some theoretical results apply only to the case where the units of a neural network output discrete values. However, most neural network units output smoothly increasing values that make optimization via local search feasible.
- Some theoretical results show that there exist problem classes that are intractable, but it can be difficult to tell whether a particular problem falls into that class.

Optimization for DL: Challenges in Neural Network Optimization

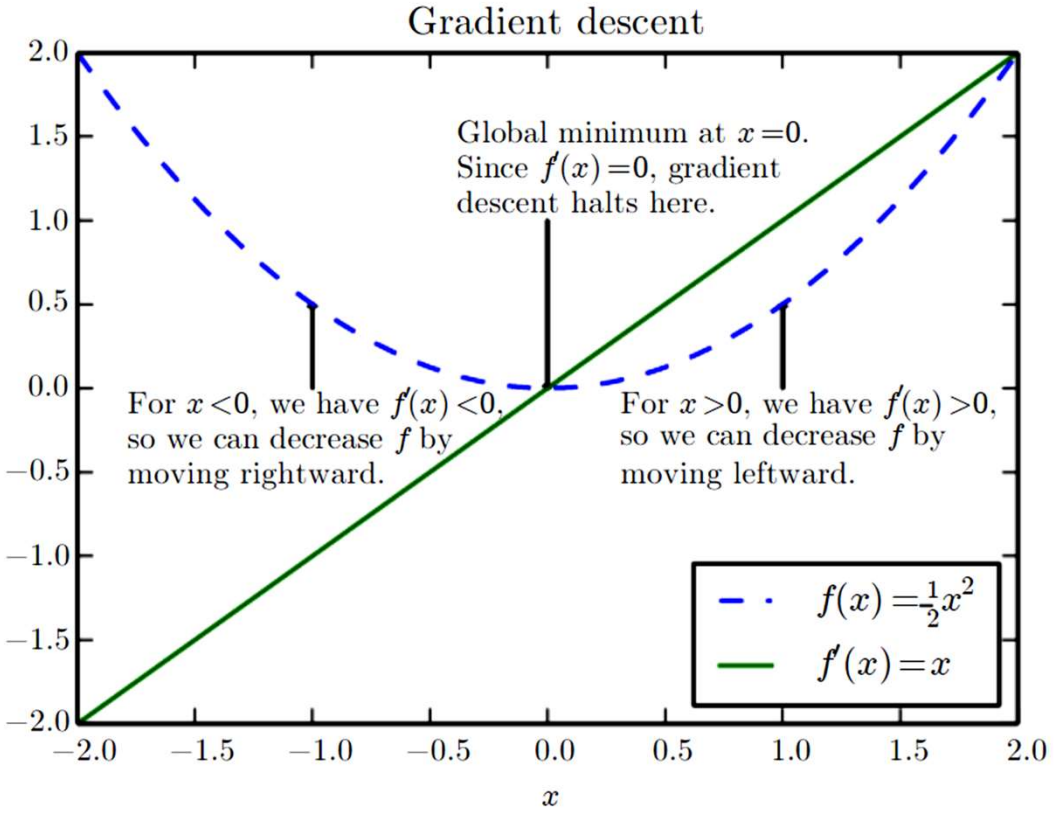
8. Theoretical Limits of Optimization

- Other results show that finding a solution for a network of a given size is intractable, but in practice we can find a solution easily by using a larger network for which many more parameter settings correspond to an acceptable solution.
- Moreover, in the context of neural network training, we usually do not care about finding the exact minimum of a function, but only in reducing its value sufficiently to obtain good generalization error.
- Theoretical analysis of whether an optimization algorithm can accomplish this goal is extremely difficult.
- Developing more realistic bounds on the performance of optimization algorithms therefore remains an important goal for machine learning research.

Optimization for DL: Basic Algorithms

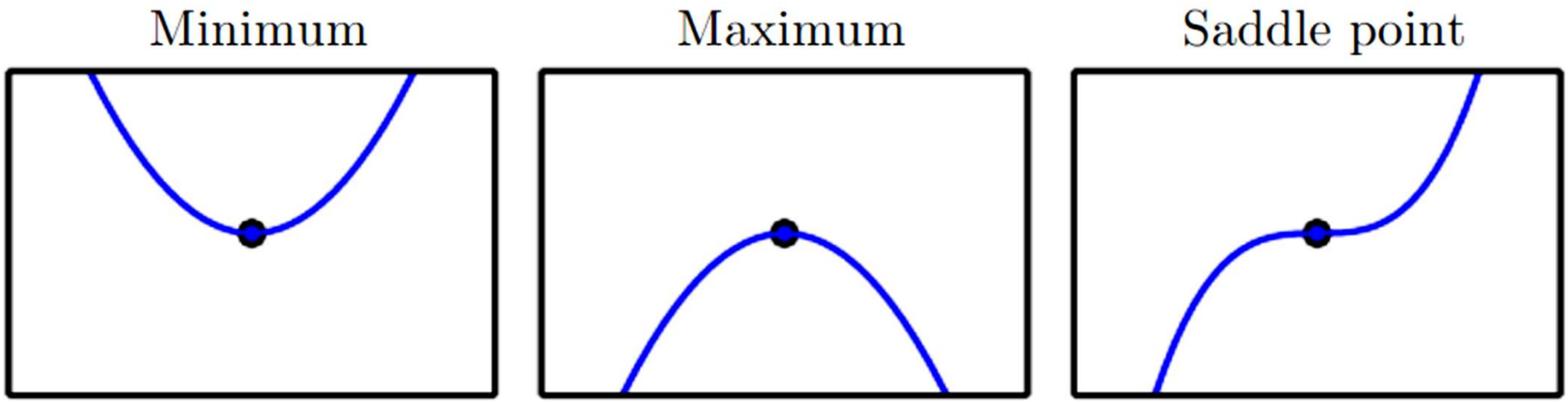
1. Gradient Descent - GD

- $f(x + \epsilon) \approx f(x) + \epsilon f'(x)$.
- We can reduce f(x) by moving x in small steps with opposite sign of the derivative.
- When f'(x) = 0, the derivative provides no information about which direction to move.
- Points where f'(x) = 0 are known as critical points or stationary points.



[P-83] Figure 4.1: An illustration of how the derivatives of a function can be used to follow the function downhill to a minimum. This technique is called ***gradient descent***.

Optimization for DL: Basic Algorithms
1. Gradient Descent



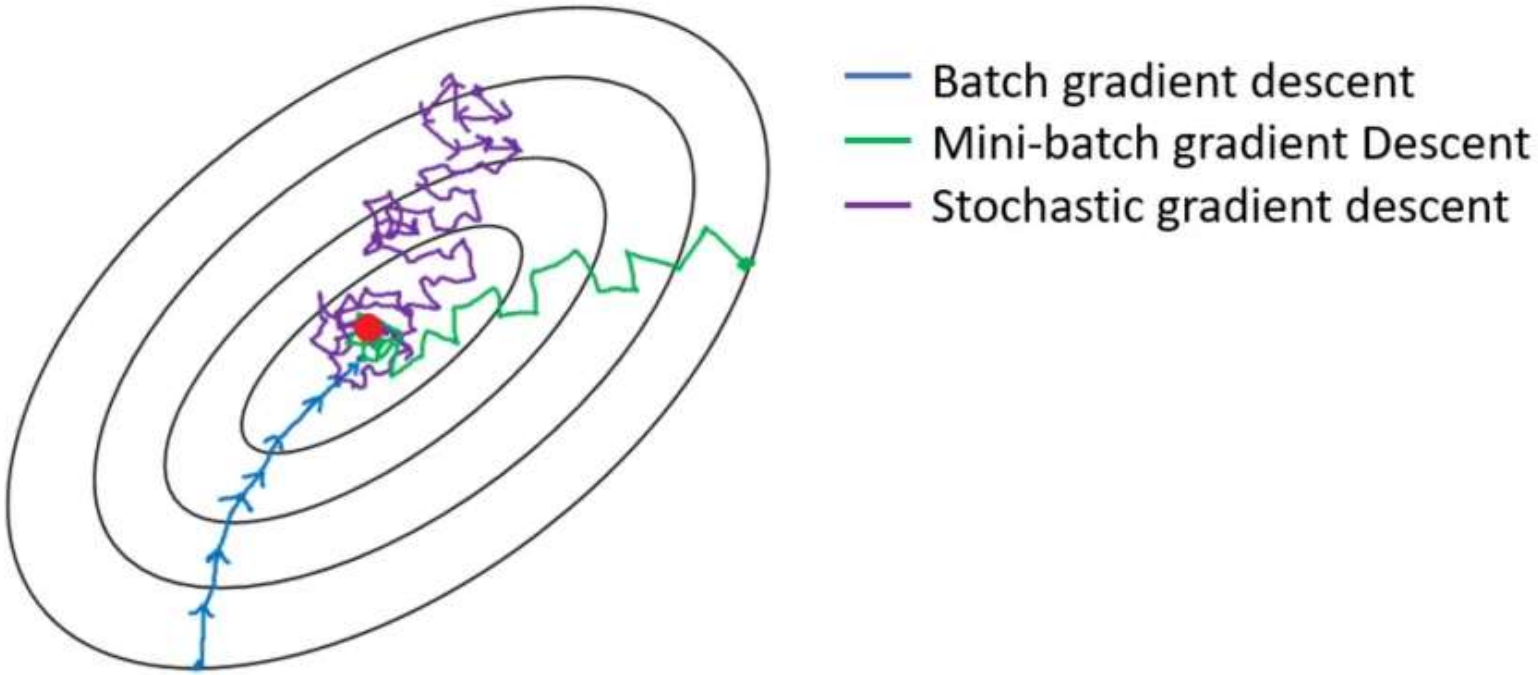
- a local minimum, a point lower than the neighboring points
- a local maximum, a point higher than the neighboring points
- a saddle point, which has neighbors that are both higher and lower than the point itself

Figure 4.2: Examples of each of the three types of critical points in 1-D. A critical point is a point with zero slope. Such a point can either be a local minimum, a local maximum, or a saddle point.

Optimization for DL: Basic Algorithms

1. Gradient Descent - GD

- In Gradient Descent (GD), we perform the forward pass using all the train data before starting the backpropagation pass to adjust the weights.



Optimization for DL: Basic Algorithms

2. Stochastic Gradient Descent -SGD

- ***Stochastic Gradient Descent (SGD):*** is a variation of Gradient Descent that randomly samples **one** training data point from the dataset to be used to compute the gradient per iteration.
- ***Minibach SGD***: It uses a minibach of greater than one and less than all examples.
  - It is possible to obtain an unbiased estimate of the gradient by taking the average gradient on a minibatch of m examples drawn i.i.d from the data generating distribution.
  - A crucial parameter for the SGD algorithm is the learning rate. We describe SGD as using a fixed learning rate ϵ.
  - In practice, it is necessary to gradually decrease the learning rate over time, so we now denote the learning rate at iteration k as ϵ_k.

Optimization for DL: Basic Algorithms

2. Stochastic Gradient Descent -SGD

- This is because the SGD gradient estimator introduces a source of ***noise*** (the random sampling of m training examples) that does not vanish even when we arrive at a minimum.
- By comparison, the true gradient of the total cost function becomes small and then 0 when we approach and reach a minimum using ***batch gradient descent***, so batch gradient descent can use a fixed learning rate.
- A sufficient condition to guarantee convergence of SGD is that

$$\sum_{k=1}^{\infty} \epsilon_k = \infty, \quad \text{and} \tag{8.12}$$

and

$$\sum_{k=1}^{\infty} \epsilon_k^2 < \infty. \tag{8.13}$$

Optimization for DL: Basic Algorithms
2. Stochastic Gradient Descent -SGD

Algorithm 8.1 Stochastic gradient descent (SGD) update at training iteration k

Require: Learning rate ϵ_k.
Require: Initial parameter $\boldsymbol{\theta}$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\boldsymbol{g}}$
end while

Optimization for DL: Basic Algorithms

2. Stochastic Gradient Descent -SGD

- In practice it is common to decay the learning rate linearly until iteration T:

$$\epsilon_k = (1 - \alpha)\epsilon_0 + \alpha\epsilon_\tau \qquad (8.14)$$

- With $\alpha = k/T$. After iteration T, it is common to leave ϵ constant.

- Choosing the learning rate: it may be chosen by trial and error, but it is usually best to choose it by monitoring learning curves that plot the objective function as a function of time.

Optimization for DL: Basic Algorithms

2. Stochastic Gradient Descent -SGD

- ***Convergence rate of an optimization algorithm***: to study the convergence rate of an optimization algorithm it is common to measure the excess error $J(\theta) - \min_{\theta} J(\theta)$, which is the amount that the current cost function exceeds the minimum possible cost.
  - When SGD is applied to a convex problem, the excess error is $O(1/\sqrt{k})$ after k iterations,
  - In the strongly convex case it is $O(1/k)$.
  - These bounds cannot be improved unless extra conditions are assumed!

Optimization for DL: Basic Algorithms

2. Stochastic Gradient Descent -SGD

- Batch gradient descent enjoys better convergence rates than stochastic gradient descent in theory.
- However, the Cramér-Rao bound (Cramér, 1946; Rao,1945) states that generalization error cannot decrease faster than $O(1/k)$.
- Bottouand Bousquet (2008) argue that it therefore *may not be worthwhile to pursue an optimization algorithm that converges faster than $O(1/k)$* for machine learning tasks—faster convergence presumably corresponds to overfitting.

Optimization for DL: Basic Algorithms

3. Momentum

- While SGD remains a very popular optimization strategy, learning with it can sometimes be slow.
- The method of momentum (Polyak, 1964) is designed to accelerate learning, especially in the face of high curvature, small but consistent gradients, or noisy gradients.
- The momentum algorithm *accumulates an exponentially decaying moving average of past gradients and continues to move in their direction.* The effect of momentum is illustrated in Fig. 8.5.
- Formally, the momentum algorithm introduces a variable v that plays the role of velocity—it is the direction and speed at which the parameters move through parameter space.

Optimization for DL: Basic Algorithms

3. Momentum

- The velocity is set to an exponentially decaying average of the negative gradient.
- The name momentum derives from a physical analogy, in which the negative gradient is a force moving a particle through parameter space, according to Newton's laws of motion. *Momentum (in physics) = mass x velocity*. In the momentum learning algorithm, we assume *unit mass*, so the velocity vector v may also be regarded as the momentum of the particle.
- A hyperparameter $\alpha \in [0, 1)$ determines how quickly the contributions of previous gradients exponentially decay.
- The update rule is given by the following formula:

Optimization for DL: Basic Algorithms

3. Momentum

- The update rule is given by:

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \left(\frac{1}{m} \sum_{i=1}^{m} L(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}) \right), \tag{8.15}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}. \tag{8.16}$$

- The velocity v accumulates the gradient elements

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

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

Optimization for DL: Basic Algorithms

3. Momentum

- The SGD algorithm with momentum is given in Algorithm 8.2.

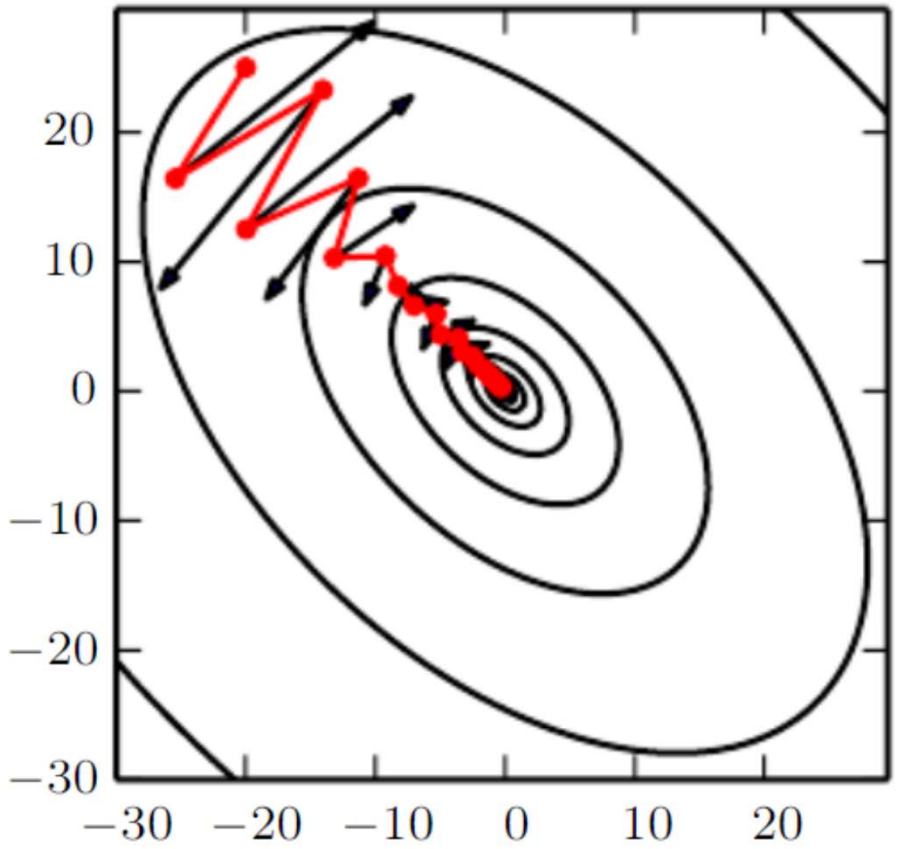
Algorithm 8.2 Stochastic gradient descent (SGD) with momentum

Require: Learning rate ϵ, momentum parameter α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
end while

Optimization for DL: Basic Algorithms

3. Momentum

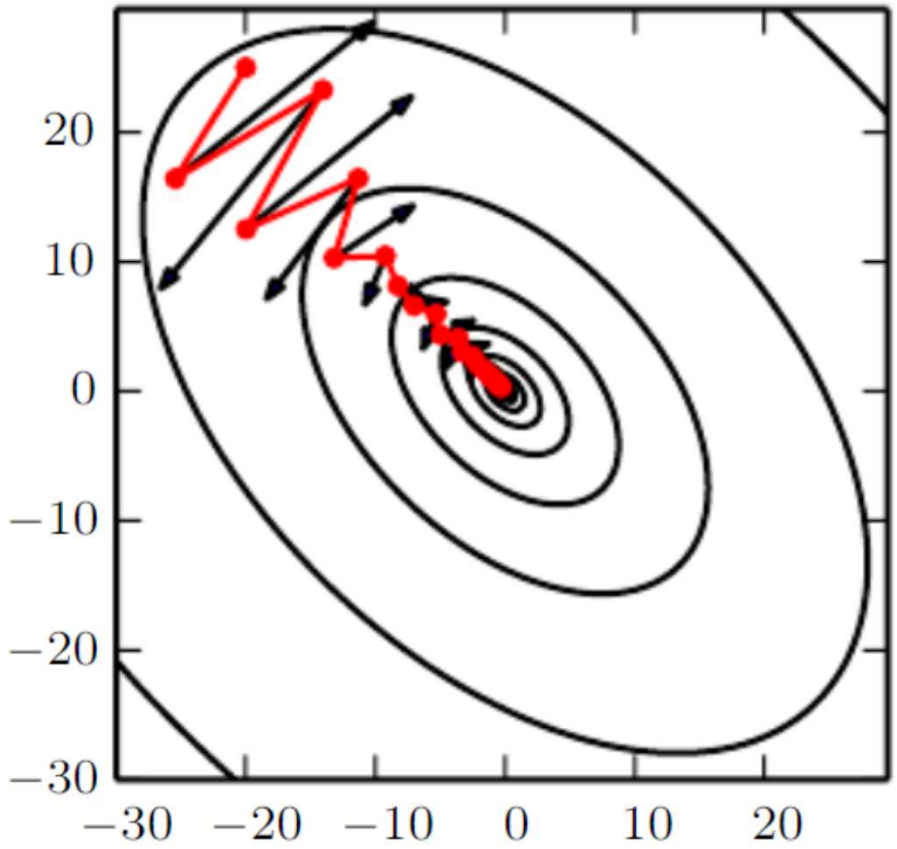
- **Figure 8.5**: Momentum aims primarily to solve two problems: poor conditioning of the Hessian matrix and variance in the stochastic gradient.
- The contour lines depict a quadratic loss function with a poorly conditioned Hessian matrix.
- The red path cutting across the contours indicates the path followed by the momentum learning rule as it minimizes this function.



Optimization for DL: Basic Algorithms

3. Momentum

- **Figure 8.5**: Momentum aims primarily to solve two problems: poor conditioning of the Hessian matrix and variance in the stochastic gradient.
- At each step along the way, we draw an arrow indicating the step that gradient descent would take at that point.
- We can see that a poorly conditioned quadratic objective looks like a long, narrow valley or canyon with steep sides.
- Momentum correctly traverses the canyon lengthwise, while gradient steps waste time moving back and forth across the narrow axis of the canyon.



Optimization for DL: Basic Algorithms

3. Momentum

- Previously, the size of the step was simply $\epsilon||\boldsymbol{g}||$
- Now, the size of the step depends on how large and how aligned a sequence of gradients are.
  - The step size is largest when many successive gradients point in exactly the same direction.
  - If the momentum algorithm always observes gradient g, then it will accelerate in the direction of −g, until reaching a terminal velocity where the size of each step is

$$\frac{\epsilon||\boldsymbol{g}||}{1-\alpha}. \tag{8.17}$$

- It is thus helpful to think of the momentum hyperparameter in terms of $1/(1-\alpha)$. For example, a = 0.9 corresponds to multiplying the maximum speed by 10 relative to the gradient descent algorithm.
- Common values of α used in practice include 0.5, 0.9, and 0.99.

Optimization for DL: Basic Algorithms

4. Nesterov Momentum

- Sutskever et al. (2013) introduced a variant of the momentum algorithm that was inspired by Nesterov's accelerated gradient method (Nesterov, 1983, 2004).
- The update rules in this case are given by:

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \left[\frac{1}{m} \sum_{i=1}^{m} L\left(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta} + \alpha \boldsymbol{v}), \boldsymbol{y}^{(i)} \right) \right], \tag{8.21}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}, \tag{8.22}$$

where the parameters α and ϵ play a similar role as in the standard momentum method.

Optimization for DL: Basic Algorithms

4. Nesterov Momentum

- The *difference between Nesterov momentum and standard momentum is where the gradient is evaluated*.
  - With Nesterov momentum the gradient is evaluated after the current velocity is applied.
- Thus one can interpret Nesterov momentum as attempting to add a correction factor to the standard method of momentum.
- In the convex batch gradient case, Nesterov momentum brings the rate of convergence of the excess error from $O(1/k)$ (after k steps) to $O(1/k^2)$ as shown by Nesterov (1983).
- Unfortunately, in the stochastic gradient case, Nesterov momentum does not improve the rate of convergence.

Optimization for DL: Basic Algorithms

4. Nesterov Momentum

Algorithm 8.3 Stochastic gradient descent (SGD) with Nesterov momentum

Require: Learning rate ϵ, momentum parameter α.

Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding labels $\boldsymbol{y}^{(i)}$.

Apply interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$

Compute gradient (at interim point): $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$

Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$

end while

Continue with Ch_2B Optimization for DL