Lecture 4: Optimization for Training Deep Models

Ali Harakeh

University of Waterloo

WAVE Lab
ali.harakeh@uwaterloo.ca

June 13, 2017

Overview

Section 1

Optimization: Introduction

Introduction

- Of all the many optimization problems involved in deep learning, the most difficult is neural network training.
- It is quite common to invest days to months of time on hundreds of machines in order 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 reduces a cost function $J(\theta)$.

Section 2

Review Of First and Second Order Methods

First Order Optimization Algorithms

- Optimization algorithms that use only the gradient are termed **first order optimization** algorithms.
- An example would be **gradient decent** where the update rule is:

$$\mathbf{x} \leftarrow \mathbf{x} - \epsilon \nabla_{\mathbf{x}} f(\mathbf{x})$$

- First order algorithms are optimal for neural network training since the target loss functions can be decomposed to a sum over training data.

Second Order Optimization Algorithms

- Optimization algorithms that make use of the Hessian matrix are termed **second order optimization** algorithms.
- The Hessian Matrix is a matrix of second derivatives of a function $\mathbf{f} : \mathbb{R}^n \rightarrow \mathbb{R}$ with each element $H_{i,j} = \frac{\partial^2}{\partial x_i \partial x_j} f(\mathbf{x})$.
- The matrix $H \in \mathbb{S}^n$ anywhere that the second derivative is continuous. This implies that $H_{i,j} = H_{j,i}$, the differential operators are commutative.

Second Order Optimization Algorithms

- The simplest second order optimization algorithm is **Newton's Method**. It has the following update rule:

$$\mathbf{x}^* = \mathbf{x}^{(0)} - H^{-1}\nabla_x f(\mathbf{x}^{(0)})$$

- Newton's method consists of applying the above equation iteratively to jump to the minimum of the function directly.

Non Deep Disadvantages of Second Order Optimization Algorithms

- The above algorithm requires the inversion of the Hessian matrix which in turn confers two disadvantages:
  - Inversion is computationally intensive when the Hessian matrix is large.
  - Inversion is highly unstable when the Hessian's **condition number** is large.
- The **condition number** of the matrix is the ratio of the magnitude of the largest singular value to the smallest.

Lipschitz Continuity

- The optimization algorithms that will be discussed in this lecture can be applied to a wide variety of functions.
- However, these algorithms come with no guarantees when it come to deep learning.
- In the context of deep learning, we sometimes gain some guarantees by restricting ourselves to functions that are either **Lipschitz continuous** or have **Lipschitz continuous derivatives**.

Lipschitz Continuity

- A Lipschitz continuous function is a function f whose rate of change is bounded by a Lipschitz constant $\mathcal{L}$:

$$\forall \mathbf{x} \forall \mathbf{y}, |f(\mathbf{x}) - f(\mathbf{y})| \leq \mathcal{L}||\mathbf{x} - \mathbf{y}||_2.$$

- This property is useful because it allows us to quantify our assumption that a small change in the input made by an algorithm such as gradient descent will have a small change in the output.
- Lipschitz continuity is also a fairly weak constraint,and many optimization problems in deep learning can be made Lipschitz continuous with relatively minor modification.

Section 3

The Difference Between Learning and Pure Optimization

How Learning Differs From Pure Optimization

- Learning differs from pure optimization in many ways, the most prominent one being the observability of the true loss function.
- In most scenarios, we care about a performance measure P that is defined on the test set. This measure is usually intractable.
- We optimize P indirectly by reducing a different cost function $J(\theta)$ in hope that doing so will improve P.
- This is in contrast to pure optimization where minimizing $J(\theta)$ is the goal itself.

The Form Of Loss Functions In Deep Learning

- In context of deep learning, the Loss function can be written as an average over the training set:

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

- Notice that the expectation is over the training data. We would usually prefer to minimize the expectation over *the data generating distribution* p_{data} rather than over the finite training set:

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

Empirical Risk Minimization

- The quantity J^* is referred to as the **risk**.
- If we knew p_{data}, **risk minimization** will be reduced to a standard optimization task.
- Since p_{data} is not known, we minimize the **empirical risk**:

$$\mathbb{E}_{(\mathbf{x},y)\sim\hat{p}_{data}} L(f(\mathbf{x};\theta),y) = \frac{1}{m}\sum_{i=1}^{m} L(f(\mathbf{x}^{(i)};\theta),y^{(i)})$$

- This whole process 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.

Empirical Risk Minimization

- A variety of theoretical results establish conditions under which the true risk can be expected to decrease by various amounts.
- What is the problem with empirical risk minimization ?

Empirical Risk Minimization

- Empirical risk minimization is prone to overfitting. Models with high enough capacity can simply memorize the training set.
- Furthermore, the most effective optimization algorithms rely on gradient decent. However, many useful Loss functions have no useful derivatives. (0-1 Loss ?)
- The above two problems means that in the context of deep learning, we cannot usually use empirical risk minimization.
- We should rely on 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.

Surrogate Loss Functions

- Instead of minimizing empirical risk, we minimize **surrogate loss functions**.
- A **surrogate loss function** acts as a proxy to empirical risk while being "nice" enough to be optimized efficiently.
- Example: SVM loss is a surrogate to the 0-1 loss for classification.

Early Stopping

- In contrast to standard optimization, training algorithms do not halt at local minima, but when **early stopping halt criterion** is satisfied.
- Typically, the early stopping criterion is based on an underlying loss function such as the 0-1 loss measured on the validation set, and is designed to halt the algorithm before overfitting occurs.
- This can be roughly thought of as a way to reincorporate the true loss function in the learning process.

Section 4

Batch and Minibatch Algorithms

Decomposing Machine Learning Loss Functions

- 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:

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

- The gradient in this case is also an expectation over training data:

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

Computing The Expectation

- Computing this expectation exactly is very expensive
- It requires evaluating the model on every example in the entire dataset.
- However, we can compute these expectations by randomly sampling a small number of examples from the dataset at every iteration.
- Why does it work ?

Computing The Expectation

- Standard error of the mean estimated from n samples is $\frac{\sigma}{\sqrt{n}}$, where σ is the standard deviation of the value of the samples.
- The denominator shows that there is a **less than linear** return to using more examples to estimate the gradient.
- Furthermore, optimization algorithms usually converge faster if they are allowed to rapidly compute **approximate** estimates of the gradient rather than slowly computing exact gradients.

Computing The Expectation

- Another consideration that motivates the estimation of the gradient from a small number of samples is the **redundancy** in the training set.
- At worst case where all m training examples are identical, evaluating the gradient at a single training example will return its true value.
- Complexity is $\Theta(m)$ for the naive approach vs $\Theta(1)$ for the approximation!
- In practice, we are unlikely to truly encounter this worst-case situation, but we may find large numbers of examples that all make very similar contributions to the gradient.

Batch, Online, and Minibatch Algorithms

- Optimization algorithms that use the entire training set to compute the gradient are called **batch** or **deterministic** gradient methods. Ones that use a single training example for that task are called **stochastic** or **online** gradient methods.
- Most of the algorithms we use for deep learning fall somewhere in between !
- These are called **minibatch** or **minibatch stochastic** methods.
- We will be calling them **stochastic** methods! Take that optimization experts!

What Minibatch Size Should We Use ?

- Larger batches provide a more accurate estimate of the gradient, but with less than linear returns.
- Multicore architectures are usually underutilized by extremely small batches.This motivates using some absolute minimum batch size, below which there is no reduction in the time to process a minibatch.

What Minibatch Size Should We Use ?

- 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 to offer better runtime. Typical power of 2 batch sizes range from 32 to 256, with 16 sometimes being attempted for large models.
- Small batches can offer a regularizing effect. The best generalization error is often achieved with batch size of 1 (Wilson and Martinez, 2003).

Sampling Minibatches

- It is **extremely crucial** that minibatches are sampled at **random**.
- Why ?

Sampling Minibatches

- Computing an unbiased estimate of the expected gradient from a set of samples requires that those samples be independent.
- Also, two subsequent gradient estimates are required to be independent from each other, so two subsequent minibatches of examples should also be independent from each other.

Sampling Minibatches: Permuting The Dataset

- Be very careful when your dataset is arranged in a correlated manner (most of datasets naturally are).
- Example: Consecutive video frames for object detection for autonomous driving.
- It is absolute necessity to shuffle the examples before selecting every minibatch.

Sampling Minibatches: Permuting The Dataset

- In practice shuffling the dataset before every minibatch selection is not feasible.
- We usually only shuffle the dataset once and store it on the hard drive imposing a fixed set of possible minibatches of consecutive examples that all models trained thereafter will use.
- The above method is not exactly random selection, but does not seem to have a detrimental effect on training.
- Failing to shuffle the dataset at all will seriously reduce the effectiveness of your algorithm.

Section 5

Challenges In Deep Model Training

Optimization Is a Hard Problem

- General optimization by itself is an extremely difficult task.
- Traditionally, machine learning has avoided this difficulty by carefully designing the objective function and constraints to insure the problem is **convex**.
- Convex optimization is not without complications. However, this is not a problem for deep models as problems we usually face in that context are **non-convex**.
- This section summarizes the most prominent complications we face in deep model training.

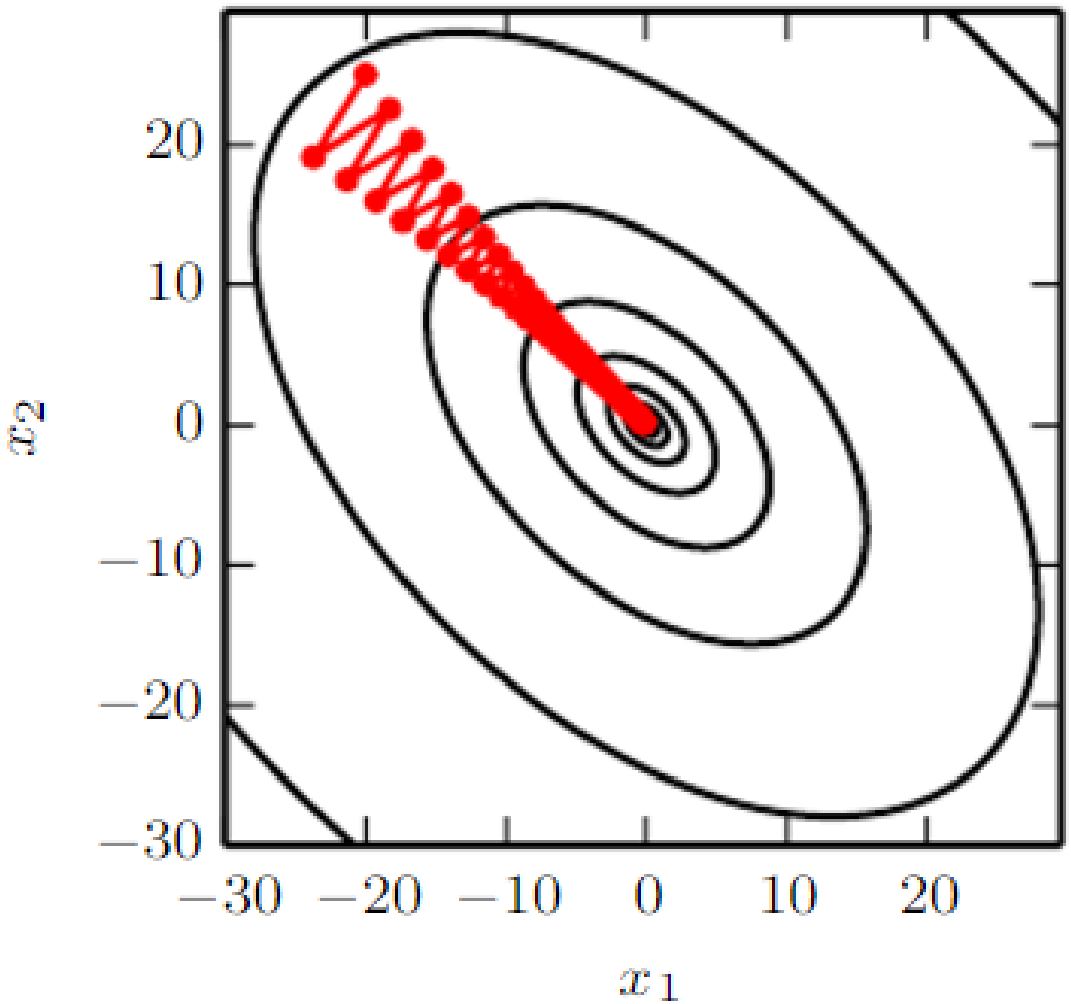
Ill-Conditioning

- Ill-conditioning of the Hessian matrix is a prominent problem in most numerical optimization problems, convex or otherwise.
- Ill-conditioning is manifested in SGD by causing the algorithm to get **stuck**, in a sense that even very small steps increase the cost function.
- Even if the algorithms doesn't get stuck, learning will proceed very slowly when the Hessian matrix has a large condition number.
- Proof or Intuition ?

Ill-Conditioning

- In multiple dimensions, there is a different second derivative for each direction at a single point.
- The condition number of the Hessian at this point measures how much the second derivatives differ from each other.
- When the Hessian has a large condition number, gradient descent performs poorly. This is because in one direction, the derivative increases rapidly, while in another direction, it increases slowly.
- Gradient descent is unaware of this change in the derivative so it does not know that it needs to explore preferentially in the direction where the derivative remains negative for longer.

Ill-Conditioning



Local Minima

- When optimizing a convex function, we know that we have reached a good solution if we find a critical point of any kind.
- With non-convex functions, such as neural nets, that is not the case.
- Functions involved in deep models are guaranteed to have an extremely large number of local minima.
- However, we will see that local minima are not necessarily a major problem.

Local Minima: Model identifiability Problem

- Any model with multiple equivalently parametrized latent variables will have multiple local minima because of the **model identifiability problem**.
- A model is said to be identifiable if a suffeciently large training set can rule out all but one setting of the model's parameters.
- Are deep models identifiable ?

Local Minima: Model identifiability Problem

- The answer is no. In fact, if we have m layers with n hidden units each, there are $n!^m$ ways of arranging the hidden units to obtain equivalent models.
- Proof ?
- This is called **weight space symmetry**.

Local Minima: Model identifiability Problem

- There are many additional causes for non-identifiability in deep models.
- For example, in any ReLU or Max-Out unit, we can scale all incoming weights and biases by $\frac{1}{\alpha}$ and then scale all the outgoing weights by α to achieve the same final value.
- If the cost function does not include regularization terms that depend directly on weights (norm penalty), every local minimum of a ReLU or Max-Out unit lies on an $m \times n$ dimensional hyperbola of equivalent local minima.

Local Minima: Model identifiability Problem

- Model identifiability issues mean that there can be an extremely large or even uncountably infinite amount of local minima in the cost function of deep models.
- However, these local minima are all equivalent in value and are not a problematic form of non-convexity.
- Local minima are only truly problematic if they have a much higher cost than the global minimum.
- 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.

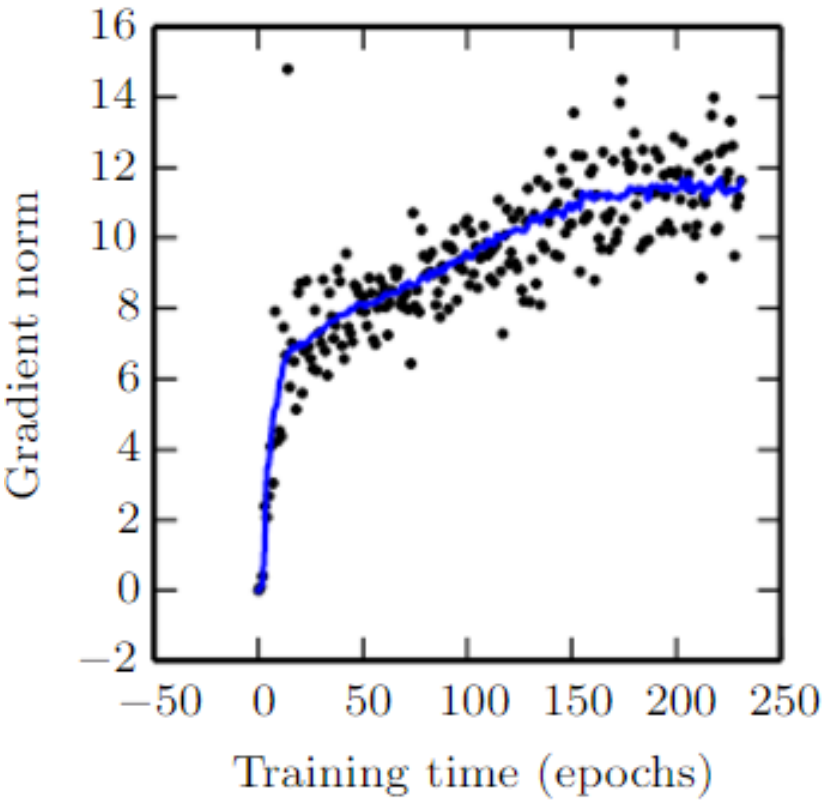
Local Minima: How Problematic Are They ?

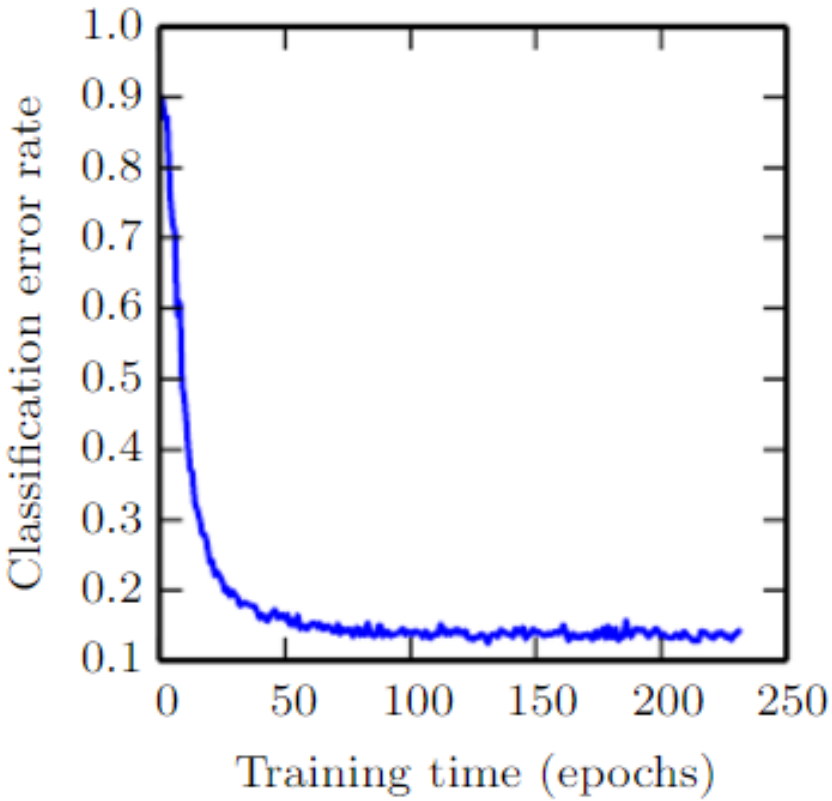
- It remains an open question whether there are many local minima of high cost for networks of practical interest and whether optimization algorithms encounter them.
- Experts now suspect that, for sufficiently large neural networks, most local minima have a low cost function value.
- Furthermore, it is not important to find a true global minimum. We can do with finding a point in parameter space that has low but not minimal cost (Saxe et al., 2013; Dauphin et al., 2014; Goodfellow et al., 2015; Choromanskaet al., 2014).

Local Minima: Conclusion

- Many practitioners attribute nearly all difficulty with neural network optimization to local minima.
- Please test for specific cases before you arrive to a conclusion.
- A simple test to try is the **gradient norm test**. If the gradient norm does not shrink over time, the problem is not a local minima problem.
- If it does shrink, the test is inconclusive. Why?

Local Minima: Conclusion





Saddle Points

- For many high-dimensional non-convex functions, local **minima** and **maxima** are in fact rare compared to **saddle points**.
- A saddle point is a point where the Hessian matrix has both positive and negative eigenvalues.
- 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.

Saddle Points

- Many classes of random functions exhibit the following behaviour:
  - In lower dimensional spaces, local minima and maxima are common.
  - In higher dimensional spaces, local minima and maxima are rare, saddle points are much more common.
- For a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$, the expected ratio of the number of saddle points to local minima grows exponentially with n.
- What is the intuition behind this phenomenon ?

Saddle Points

- Another amazing phenomenon that occurs in many random functions is that the eigenvalues of the Hessian becomes more likely to be positive as we reach regions of low cost.
- This means that local minima are much more likely to have a low cost than a high cost.
- Also, critical points with high cost are much more likely to be saddle points and those with extremely high cost are much more likely yo be local maxima.

Plateaus, Saddle Points, and Other Flat Regions

- Baldik and Hornik (1989) showed theoretically that shallow autoencoders with no non-linearities have global minima and saddle points but no local minima with a cost value greater than the global minimum.
- Saxe et al. (2013) provided exact solutions to the complete learning dynamics in linear networks and showed that learning in these models captures many of the qualitative features observed in the training of deep models with non-linear activation functions.
- Dauphit et al. (2014) showed experimentally that real neural networks also have loss functions that contain very many high-cost saddle points.

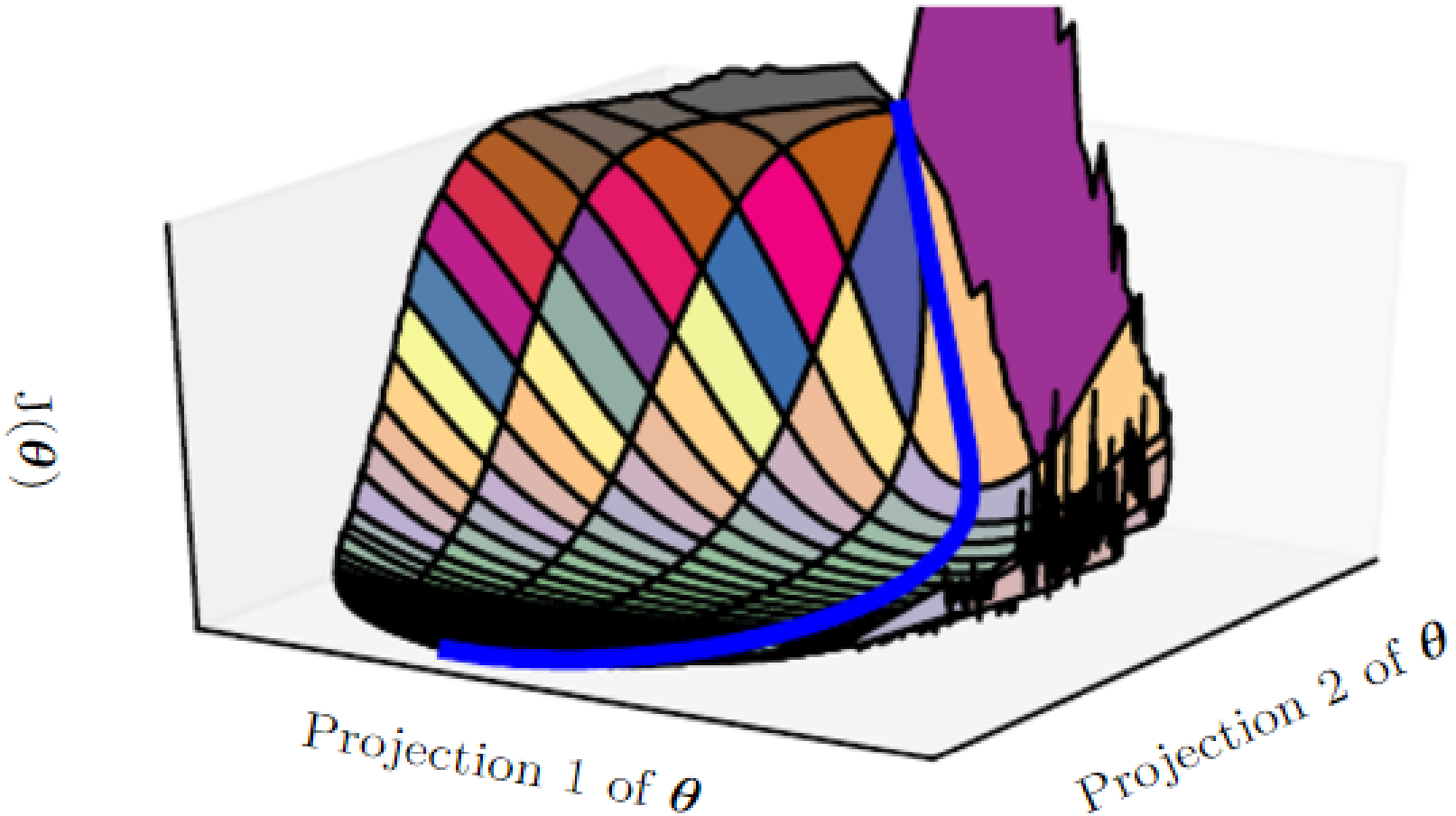
Saddle Points

- What are the implications of the proliferation of saddle points in the cost functions of deep models ?

Saddle Points: The Effect On First Order Optimization 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.
- Goodfellow et al. (2015) empirically showed that the gradient decent trajectory rapidly escapes saddle points.
- They also argued that that continuous-time gradient descent may be shown analytically to be repelled from, rather than attracted to, a nearby saddle point, but the situation may be different for more realistic uses of gradient descent.

Saddle Points: The Effect On First Order Optimization Algorithms

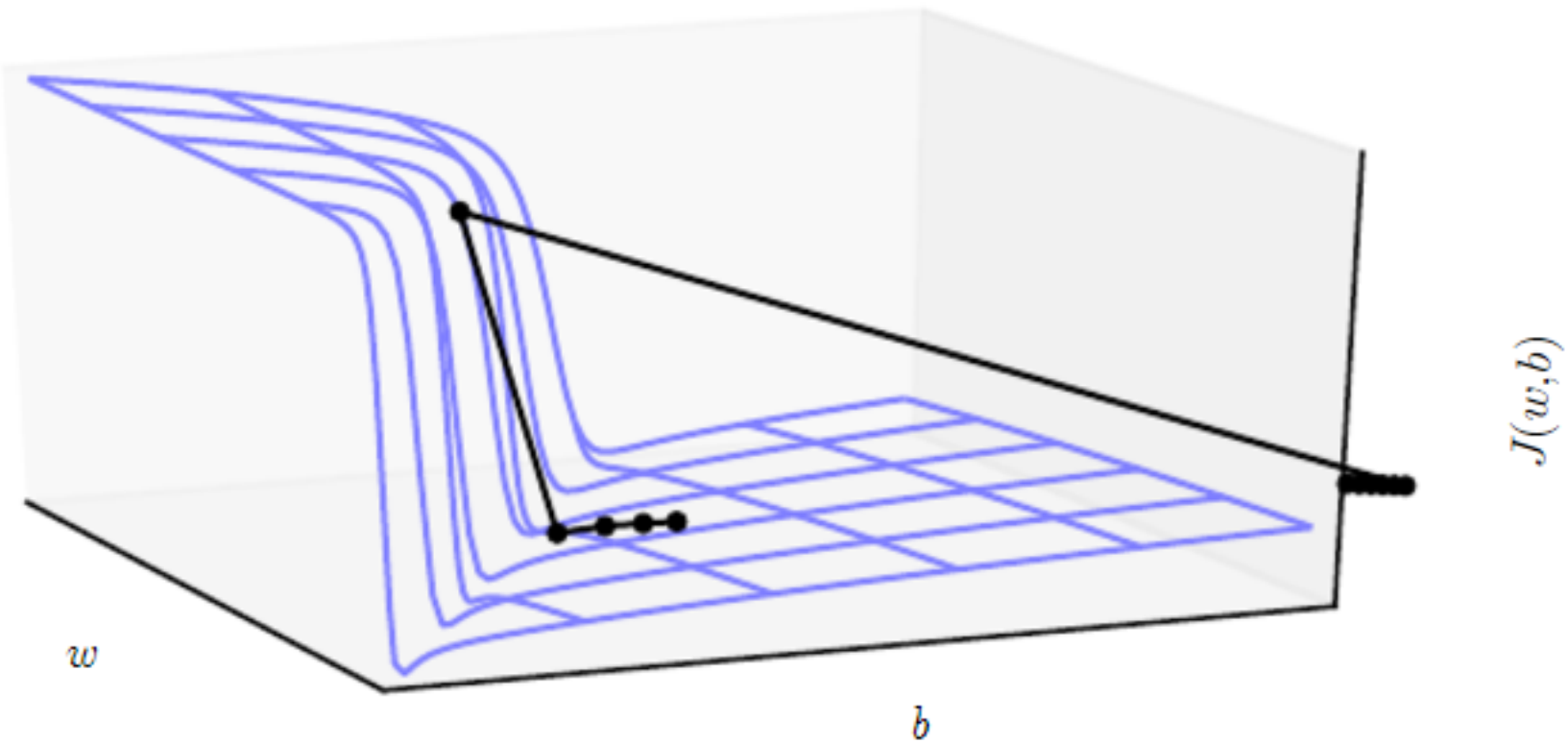


Saddle Points: The Effect On Second Order Optimization Algorithms

- For Newton's Method, saddle points constitute a major problem. This is because unlike gradient decent, which is designed to move downhill, Newton's method actively seeks solutions at critical points where the gradient is zero.
- The proliferation of saddle points in high dimensional spaces explains why second order methods have failed to replace gradient decent for deep learning.
- Dauphin et al. (2014) introduced a **saddle-free Newton Method** for second order optimization.
- Second-order methods remain difficult to scale to large neural networks, but this saddle-free approach holds promise if it could be scaled.

Cliffs And Exploding Gradients

- Neural networks with many layers often have extremely steep regions reassembling cliffs.



Cliffs And Exploding Gradients

- Cliffs result from the multiplication of several large weights together.
- 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.
- Very very dangerous when approached from above or bellow.
- Solutions ?

Cliffs And Exploding Gradients: Gradient Clipping

- Gradients do not specify the optimal step size, but only the optimal direction within and **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.

Additional Problems Faced In Neural Network Optimization

- **Long Term Dependencies**: Arises when the computational graph is very deep. The result of this problem is vanishing and exploding gradients.
- **Inexact Gradients**: In practice, we usually only have a noisy or even biased estimate of the gradient and the Hessian. Sometimes, gradients for our loss functions are even intractable. Not a big issue in neural network training. Surrogate loss functions tend to perform well enough in practice.

Additional Problems Faced In Neural Network Optimization

- **Poor Correspondence between Local and Global Structure:** It is possible to overcome all of the above 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.
- Initialization is really important !

Section 6

Conclusion

Conclusion

- Optimizing cost functions for deep networks is really hard.
- We almost never arrive to a global minimum, our goal is to reduce the generalization error rather than the cost function itself.
- 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.
- Also, initialization is very important for stable performance of our optimization algorithms.

Next Lecture

- Parameter Initialization Strategies.
- SGD Varients: SGD, Momentum, Nestrov Momentum, AdaGrad, RMS-Prop, Adam.
- **Tentative:** Newton's Method, Conjugate Gradients, Broyden-Fletcher-Goldfarb-Shanno Algorithms (BFGS), Limited Memory BFGS (L-BFGS).
- Batch Norm, Coordinate Descent, Polyak Averaging, Greedy Supervised Pretraining.
- Curriculum Learning.