Beyond Gradient Descent

Dr. Sanjay Chatterji

CS 837

The Challenges with Gradient Descent

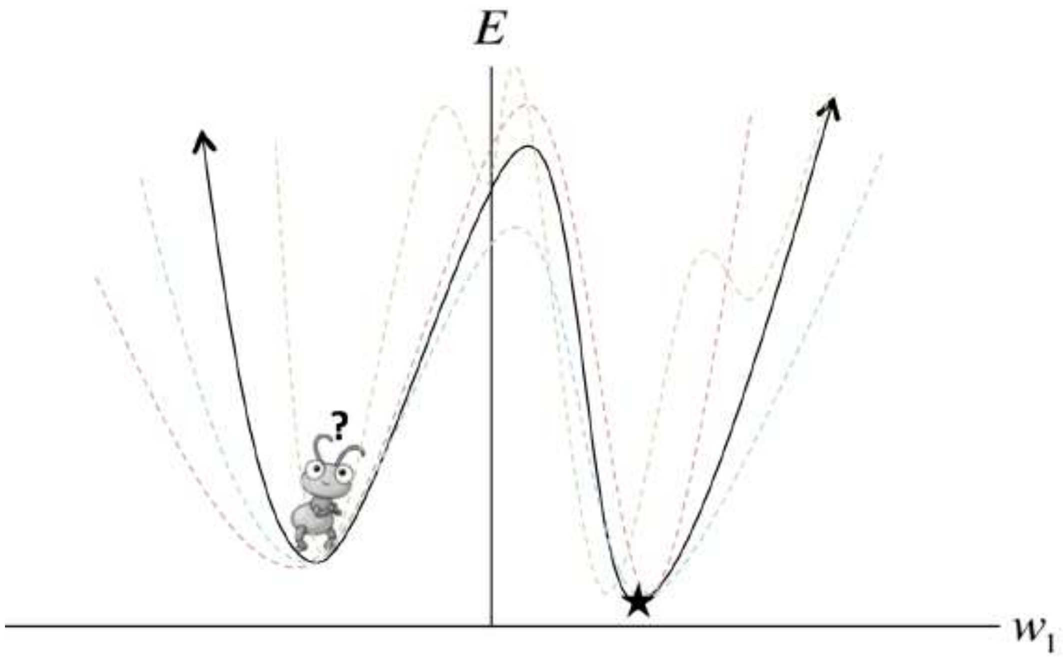
- Deep neural networks are able to crack problems that were previously deemed intractable.
- Training deep neural networks end to end, however, is fraught with difficult challenges
  - massive labelled datasets (ImageNet, CIFAR, etc.)
  - better hardware in the form of GPU acceleration
  - several algorithmic discoveries
- More recently, breakthroughs in optimization methods have enabled us to directly train models in an end-to-end fashion.
- The primary challenge in optimizing deep learning models is that we are forced to use minimal local information to infer the global structure of the error surface.
- This is a hard problem because there is usually very little correspondence between local and global structure.

Organization of the chapter

- The next couple of sections will focus primarily on local minima and whether they pose hurdles for successfully training deep models.
- In subsequent sections, we will further explore the nonconvex error surfaces induced by deep models, why vanilla mini-batch gradient descent falls short, and how modern nonconvex optimizers overcome these pitfalls.

Local Minima in the Error Surfaces

- Assume an ant on the continental United States map, and goal is to find the lowest point on the surface.
- The surface of the US is extremely complex (a non convex surface).
- Even a mini-batch version won't save us from a deep local minimum.



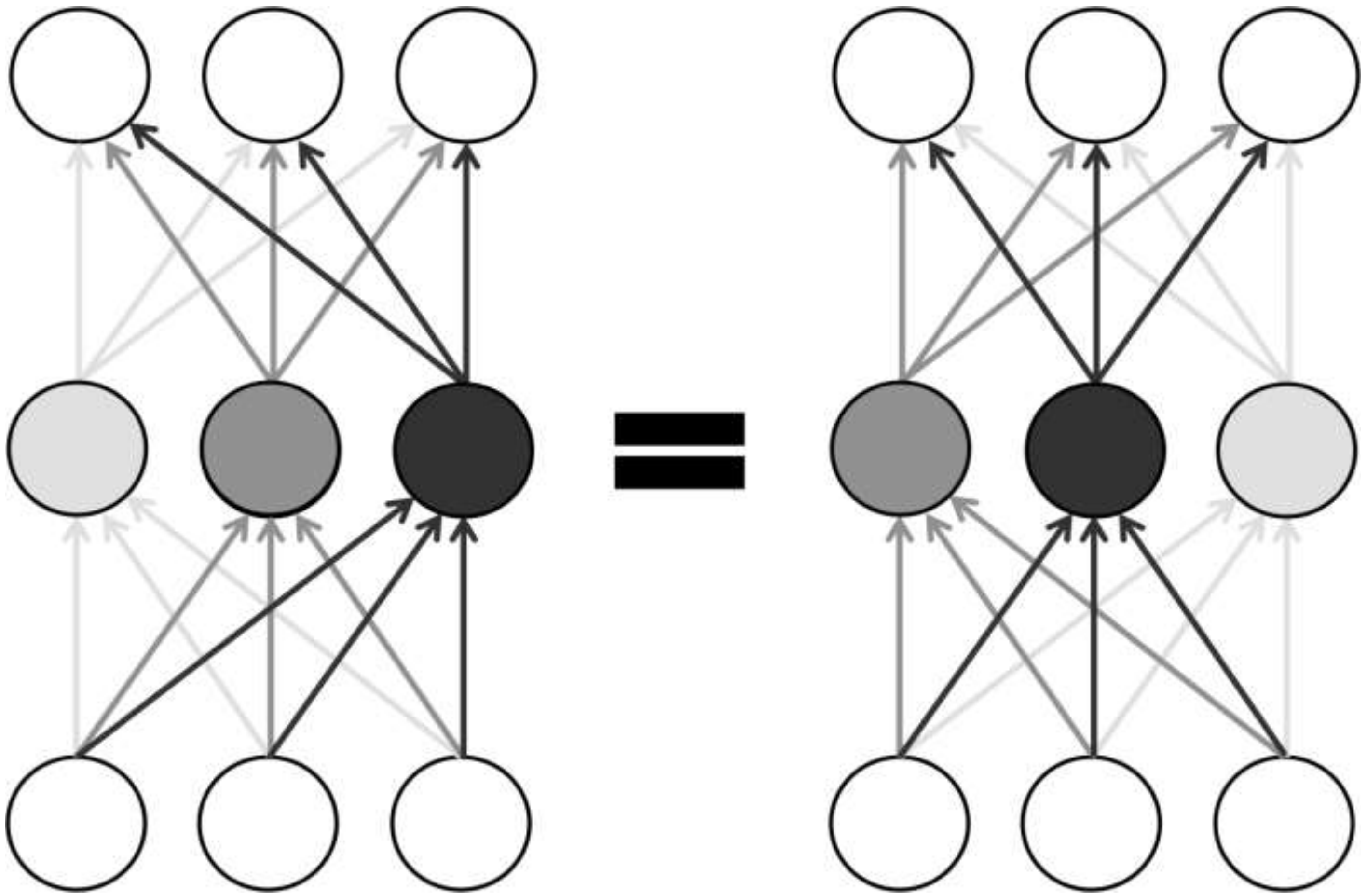
Couple of critical questions

- Theoretically, local minima pose a significant issue.
- But in practice, how common are local minima in the error surfaces of deep networks?
- In which scenarios are they actually problematic for training?
- In the following two sections, we’ll pick apart common misconceptions about local minima.

Model Identifiability

- The first source of local minima is tied to a concept commonly referred to as model identifiability.
- Deep neural networks has an infinite number of local minima. There are two major reasons.
  - within a layer any rearrangement (symmetric) of neurons will give same final output. Total $n!^l$ equivalent configurations.
  - As ReLU uses a piecewise linear function, we can multiply incoming weights by k while scaling outgoing weights by 1/k without changing the behavior of the network.

Continued…



Local minima arise for non-identifiability are not inherently problematic

- All non-identifiable configurations behave in an indistinguishable fashion
- Local minima are only problematic when they are spurious: configuration of weights at local minima in a neural network has higher error than the configuration at the global minimum.

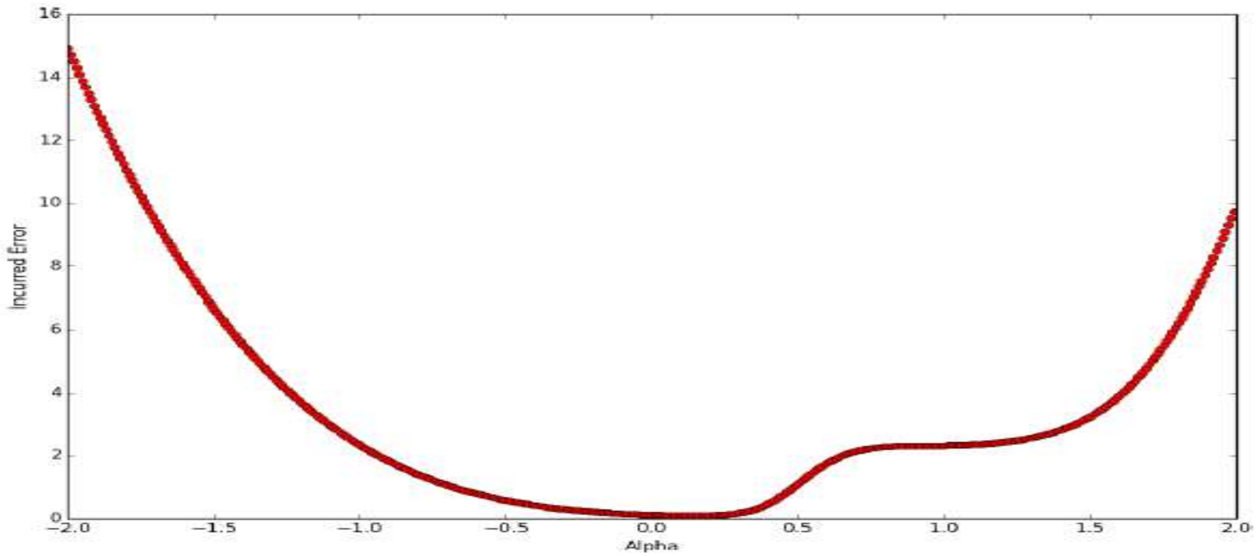
How Pesky Are Spurious Local Minima in Deep Networks?

- For many years, deep learning practitioners blamed all of their troubles in training deep networks on spurious local minima.
- Recent studies indicate that most local minima have error rates and generalization characteristics similar to global minima.
- We might try to tackle this problem by plotting the value of the error function over time as we train a deep neural network.
- This strategy, however, doesn't give us enough information about the error surface.
- Instead of analyzing the error function over time, Goodfellow et al. (2014) investigated error surface between an initial parameter vector and a successful final solution using linear interpolation.

$$\theta_\alpha = \alpha \cdot \theta_f + (1 - \alpha) \cdot \theta_i.$$

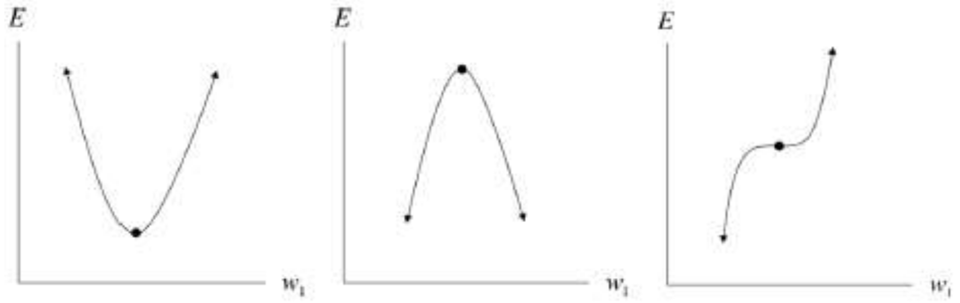
Continued..

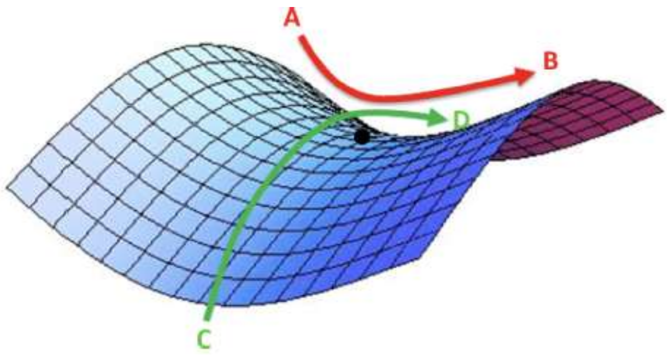
- If we run this experiment over and over again, we find that there are no truly troublesome local minima that would get us stuck.
- Vary the value of alpha to see how the error surface changes as we traverse the line between the randomly initialized point and the final SGD solution.
- The true struggle of gradient descent isn't the existence of troublesome local minima, but to find the appropriate direction to move.



Flat Regions in the Error Surface

- The gradient approaches zero in a peculiar flat region (alpha = 1). Not local minima.
- zero gradient might slow down learning.
- More generally, given an arbitrary function, a point at which the gradient is the zero vector is called a critical point.
- These "flat" regions that are potentially pesky but not necessarily deadly are called *saddle points*.
- As function has more and more dimensions, saddle points are exponentially more likely than local minima.





Continued..

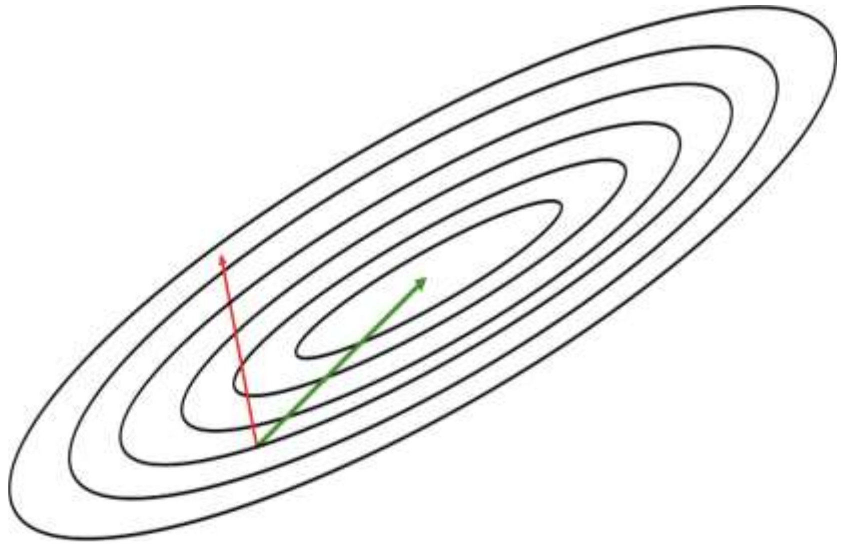
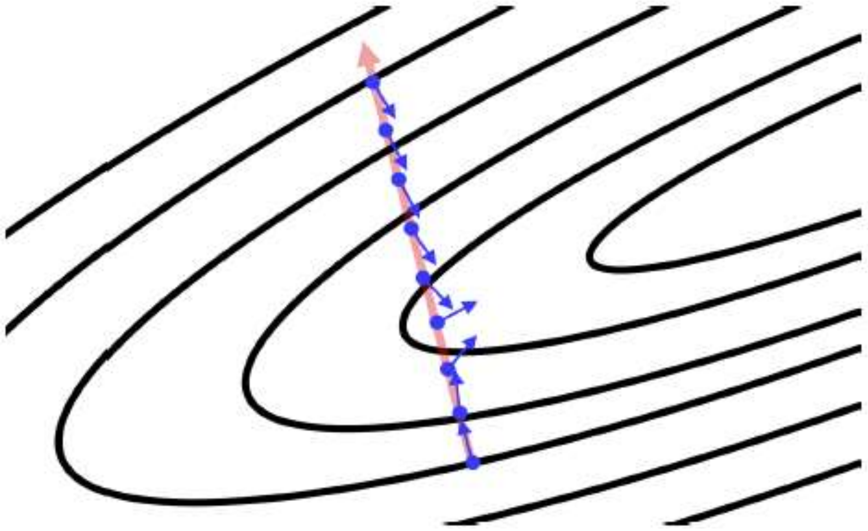
- In *d*-dimensional parameter space, we can slice through a critical point on *d* different axes.
- A critical point can only be a local minimum if it appears as a local minimum in every single one of the *d* one-dimensional subspaces
- critical point can come in one of three different flavors in a one dimensional subspace
  - probability that a random critical point is in a random function is $1/3^d$
  - a random function with *k* critical points has an expected number of $k/3^d$ local minima.
  - as the dimensionality of our parameter space increases, local minima become exponentially more rare.

When the Gradient Points in the Wrong Direction

- Upon analyzing the error surfaces of deep networks, it seems like the most critical challenge to optimizing deep networks is finding the correct trajectory to move in.
- Gradient isn’t usually a very good indicator of the good trajectory.
- When the contours are perfectly circular does the gradient always point in the direction of the local minimum?
- If the contours are extremely elliptical, the gradient can be as inaccurate as 90 degrees away from the correct direction!
- For every weight *wi* in the parameter space, the gradient computes “how the value of the error changes as we change the value of *wi”:* $\partial E/\partial wi$
- It gives us the direction of steepest descent.
- If contours are perfectly circular and we take big step, the gradient doesn’t change direction as we move.

Gradient using second derivatives

- The gradient changes under our feet as we move in a certain direction. Compute second derivatives.
- We can compile this information into a special matrix known as the *Hessian matrix* (***H***).
- Computing the Hessian matrix exactly is a difficult task.

Momentum-Based Optimization

- The problem of an ill-conditioned Hessian matrix manifests itself in the form of gradients that fluctuate wildly.
- One popular mechanism for dealing with ill-conditioning bypasses the computation of the Hessian, and focuses on how to cancel out these fluctuations over the duration of training.
- There are two major components that determine how a ball rolls down an error surface.
  - Acceleration
  - Motion

Momentum-Based Optimization

- Our goal, then, is to somehow generate an analog for velocity in our optimization algorithm.
- We can do this by keeping track of an *exponentially weighted decay* of past gradients.
- We use the momentum hyperparameter *m* to determine what fraction of the previous velocity to retain in the new update.
- Momentum significantly reduces the volatility of updates. The larger the momentum, the less responsive we are to new updates.

$$\mathbf{v}_i = m\mathbf{v}_{i-1} - \epsilon\mathbf{g}_i$$

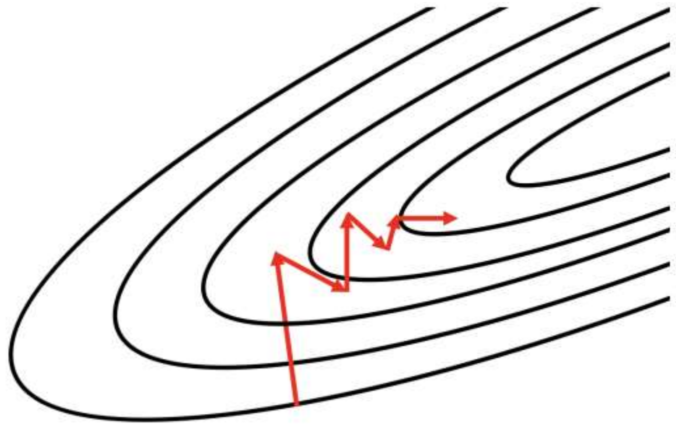
$$\theta_i = \theta_{i-1} + \mathbf{v}_i$$

A Brief View of Second-Order Methods

- Computing the Hessian is a computationally difficult task.
- Momentum afforded us significant speedup without having to worry about it altogether.
- Several second-order methods, have been researched over the past several years that attempt to approximate the Hessian directly.
  - Conjugate gradient descent (*conjugate direction* relative to the previous choice)
  - *Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm (*inverse of the Hessian matrix)

Second-Order Methods

- The conjugate direction is chosen by using an indirect approximation of the Hessian to linearly combine the gradient and our previous direction.
- With a slight modification, this method generalizes to the nonconvex error surfaces we find in deep networks.
- BFGS has a significant memory footprint, but recent work has produced a more memory-efficient version known as *L-BFGS*.



Learning Rate Adaptation

- One of the major breakthroughs in modern deep network optimization was the advent of learning rate adaption.
- The basic concept behind learning rate adaptation is that the optimal learning rate is appropriately modified over the span of learning to achieve good convergence properties.
- Some popular adaptive learning rate algorithms
  - AdaGrad
  - RMSProp
  - Adam

AdaGrad—Accumulating Historical Gradients

- It attempts to adapt the global learning rate over time using an accumulation of the historical gradients.
- Specifically, we keep track of a learning rate for each parameter.
- This learning rate is inversely scaled with respect to the root mean square of all the parameter's historical gradients (gradient accumulation vector r).
- Note that we add a tiny number δ(~10^{-7}) to the denominator in order to prevent division by zero.

$$\mathbf{r}_i = \mathbf{r}_{i-1} + \mathbf{g}_i \odot \mathbf{g}_i$$

$$\theta_i = \theta_{i-1} - \frac{\epsilon}{\delta \oplus \sqrt{\mathbf{r}_i}} \odot \mathbf{g}$$

Simply using a naive accumulation of gradients isn’t sufficient

- This update mechanism means that the parameters with the largest gradients experience a rapid decrease in their learning rates, while parameters with smaller gradients only observe a small decrease in their learning rates.
- AdaGrad also has a tendency to cause a premature drop in learning rate, and as a result doesn’t work particularly well for some deep models.
- While AdaGrad works well for simple convex functions, it isn’t designed to navigate the complex error surfaces of deep networks.
- Flat regions may force AdaGrad to decrease the learning rate before it reaches a minimum.

RMSProp—Exponentially Weighted Moving Average of Gradients

- Lets bring back a concept we introduced earlier while discussing momentum to dampen fluctuations in the gradient.
- Compared to naive accumulation, exponentially weighted moving averages also enable us to “toss out” measurements that we made a long time ago.
- The decay factor *ρ* determines how long we keep old gradients.
- The smaller the decay factor, the shorter the effective window. Plugging this modification into AdaGrad gives rise to the RMSProp learning algorithm.

$$\mathbf{r}_i = \rho \mathbf{r}_{i-1} + (1-\rho)\mathbf{g}_i \odot \mathbf{g}_i$$

Adam—Combining Momentum and RMSProp

- Spiritually, we can think about Adam as a variant combination of RMSProp and momentum.
- We want to keep track of an exponentially weighted moving average of the gradient (*first moment* of the gradient).
- Similarly to RMSProp, we can maintain an exponentially weighted moving average of the historical gradients (*second moment* of the gradient).

$\mathbf{m}_i = \beta_1 \mathbf{m}_{i-1} + (1-\beta_1)\mathbf{g}_i$

$\mathbf{v}_i = \beta_2 \mathbf{v}_{i-1} + (1-\beta_2)\mathbf{g}_i \odot \mathbf{g}_i$

Bias in Adam

- However, it turns out these estimations are biased relative to the real moments because we start off by initializing both vectors to the zero vector.
- In order to remedy this bias, we derive a correction factor for both estimations.
- Recently, Adam has gained popularity because of its corrective measures against the zero initialization bias (a weakness of RMSProp) and its ability to combine the core concepts behind RMSProp with momentum more effectively.
- The default hyperparameter settings for Adam for TensorFlow generally perform quite well, but Adam is also generally robust to choices in hyperparameters.
- The only exception is that the learning rate may need to be modified in certain cases from the default value of 0.001.

The Philosophy Behind Optimizer Selection

- We’ve discussed several strategies that are used to make navigating the complex error surfaces of deep networks more tractable.
- These strategies have culminated in several optimization algorithms.
- While it would be awfully nice to know when to use which algorithm, there is very little consensus among expert practitioners.
- Currently, the most popular algorithms are mini-batch gradient descent, mini-batch gradient with momentum, RMSProp, RMSProp with momentum, Adam, and AdaDelta.

Thank You