How Learning Differs from Optimization

Learning vs Pure Optimization

- Optimization algorithms for deep learning differ from traditional optimization in several ways:
  - Machine learning acts indirectly
    - We care about some performance measure P defined wrt the training set which may be intractable
    - We reduce a different cost function $J(\boldsymbol{\theta})$ in the hope that doing so will reduce P
- Pure optimization: minimizing J is a goal in itself
- Optimizing algorithms for training Deep models:
  - Includes specialization on specific structure of ML objective function

Typical Cost Function

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

$$J(\theta) = E_{(\boldsymbol{x},y)\sim\hat{p}_{\text{data}}}\left(L(f(\boldsymbol{x};\theta),y)\right)$$

  - Where
    - L is the per-example loss function
    - $f(\boldsymbol{x};\boldsymbol{\theta})$ is the predicted output when the input is $\boldsymbol{x}$
    - $\hat{p}_{data}$ is the empirical distribution
  - In supervised learning y is target output

Typical Cost Function

- We consider the unregularized supervised case
  - where arguments of L are $f(\boldsymbol{x}\,;\boldsymbol{\theta}\,)$ and y
- Trivial to extend to cases:
  - Where parameters $\boldsymbol{\theta}$ and input $\boldsymbol{x}$ are arguments or
  - Exclude output y as argument
  - For regularization or unsupervised learning

Objective wrt data generation is risk

- Objective function wrt training set is

$$J(\theta) = E_{(\boldsymbol{x},y)\sim\hat{p}_{\text{data}}}\Big(L(f(\boldsymbol{x};\theta),y)\Big)$$

L is the per-example loss function

- We would prefer to minimize the corresponding objective function where expectation is across the data generating distribution p_{data} rather than over finite training set

$$J^*(\theta) = E_{(\boldsymbol{x},y)\sim p_{\text{data}}}\Big(L(f(\boldsymbol{x};\theta),y)\Big)$$

  – The goal of a machine learning algorithm is to reduce this expected generalization error

- This quantity is known as *risk*

Empirical Risk

- True risk is $J^*(\boldsymbol{\theta}) = E_{(\boldsymbol{x},y)\sim p_{\text{data}}}\left(L(f(\boldsymbol{x};\boldsymbol{\theta}),y)\right)$
  - If we knew $p_{\text{data}}(\boldsymbol{x},y)$ it would be optimization solved by an optimization algorithm
  - When we do not know $p_{\text{data}}(\boldsymbol{x},y)$ but only have a training set of samples, we have a machine learning problem
- Empirical risk, with m training examples, is

$$J(\boldsymbol{\theta}) = E_{(\boldsymbol{x},y)\sim \hat{p}_{\text{data}}}\left(L(f(\boldsymbol{x};\boldsymbol{\theta}),y)\right) = \frac{1}{m}\sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}),y^{(i)})$$

Empirical Risk Minimization

- Empirical risk, with m training examples, is

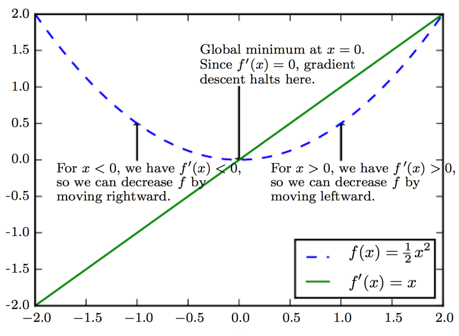
$$J(\boldsymbol{\theta}) = E_{(\boldsymbol{x},y)\sim\hat{p}_{\text{data}}}\Big(L\big(f(\boldsymbol{x};\boldsymbol{\theta}),y\big)\Big) = \frac{1}{m}\sum_{i=1}^{m} L\big(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}),y^{(i)}\big)$$

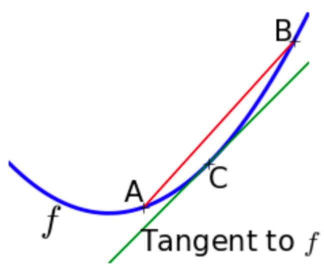
  - Which is the average training error
  - Still similar to straightforward optimization
- But *empirical risk minimization* is not very useful:
  1. Prone to overfitting: can simply memorize training set
  2. SGD is commonly used, but many useful loss functions have 0-1 loss, with no useful derivatives (derivative is either 0 or undefined everywhere)
- We must use a slightly different approach
  - Quantity we must optimize is even more different from what we truly want to optimize

Challenges in Neural Network Optimization

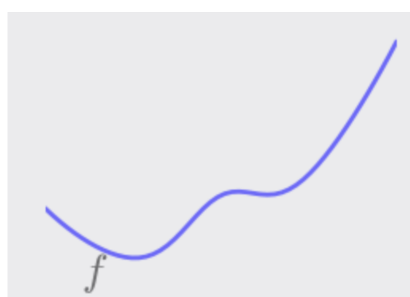
Optimization is a difficult task

- Optimization is an extremely difficult task
  - Traditional ML: careful design of objective function and constraints to ensure convex optimization





  - When training neural networks, we must confront the nonconvex case



Challenges in Optimization

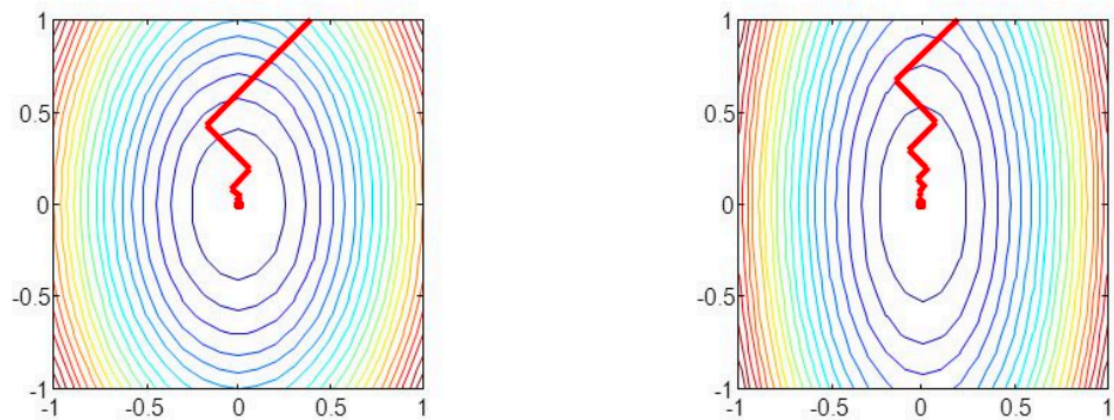
- Summary of challenges in optimization
  1. Ill-conditioning
  2. Local minima
  3. Plateaus, saddle points and other flat regions
  4. Cliffs and exploding gradients
  5. Long-term dependencies
  6. Inexact gradients
  7. Poor correspondence between local & global structure
  8. Theoretical limits of optimization

1. Ill-conditioning of the Hessian

- Even when optimizing convex functions one problem is an ill conditioned Hessian matrix, H
  - Very general problem in optimization, convex or not

Condition number is the ratio of maximal and minimal eigenvalues of the Hessian $\nabla^2 f(x)$,

$$\kappa = \frac{\lambda_{\max}}{\lambda_{\min}}$$



Problem with large condition number is called **ill-conditioned**

Steepest descent **convergence rate is slow** for ill-conditioned problems

Source: https://slideplayer.com/slide/4916524/

Result of Ill-conditioning

- Causes SGD to be stuck: even very small steps cause increase in cost function

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}^{(0)}) + (\boldsymbol{x} - \boldsymbol{x}^{(0)})^T \boldsymbol{g} + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^{(0)})^T H (\boldsymbol{x} - \boldsymbol{x}^{(0)})$$

  - Gradient descent step of $-\varepsilon \boldsymbol{g}$ will add to the cost $-\varepsilon \boldsymbol{g}^T \boldsymbol{g} + \frac{1}{2}\varepsilon^2 g^T H \boldsymbol{g}$

Substituting $\boldsymbol{x}=\boldsymbol{x}^{(0)}-\varepsilon \boldsymbol{g}$

$$f(\boldsymbol{x}^{(0)} - \varepsilon \boldsymbol{g}) \approx f(\boldsymbol{x}^{(0)}) - \varepsilon \boldsymbol{g}^T \boldsymbol{g} + \frac{1}{2}\varepsilon^2 g^T H \boldsymbol{g}$$

  - Ill conditioning becomes a problem when $\frac{1}{2}\varepsilon^2 g^T H \boldsymbol{g} > \varepsilon \boldsymbol{g}^T \boldsymbol{g}$
  - To determine whether ill-conditioning is detrimental monitor $g^T g$ and $g^T H g$ terms
    - Gradient norm doesn't shrink but $g^T H g$ grows order of magnitude
  - Learning becomes very slow despite a strong gradient

2. Local Minima

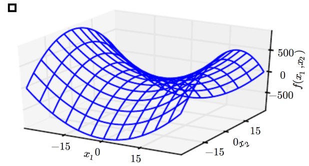
- In convex optimization, problem is one of finding a local minimum
- Some convex functions have a flat region rather than a global minimum point
- Any point within the flat region is acceptable
- With non-convexity of neural nets many local minima are possible
- Many deep models are guaranteed to have an extremely large no. of local minima
- This is not necessarily a major problem

Model Identifiability

- Model is identifiable if large training sample set can rule out all but one setting of parameters
  - Models with latent variables are not identifiable
    - Because we can exchange latent variables
      - If we have m layers with n units each there are $n!^m$ ways of arranging the hidden units
    - This non-identifiability is *weight space symmetry*
  - Another is scaling incoming weights and biases
    - By a factor α and scale outgoing weights by 1/α
- Even if a neural net has uncountable no. of minima, they are equivalent in cost
  - So not a problematic form of non-convexity

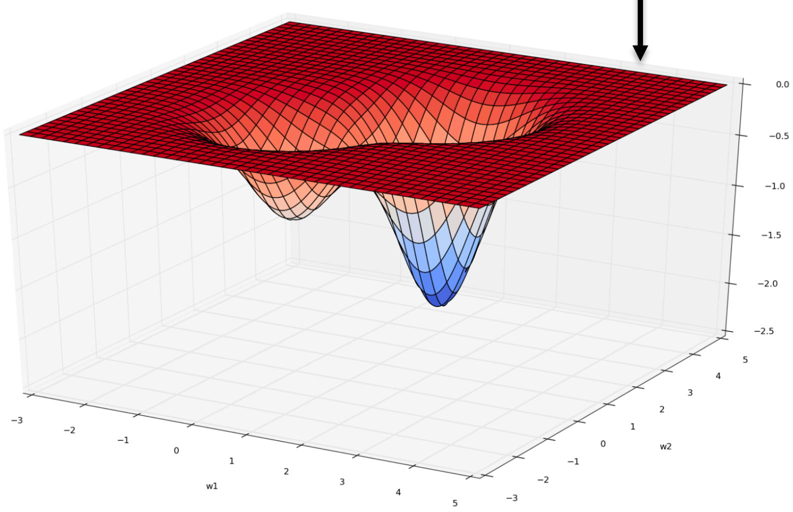
3. Plateaus, Saddle Points etc

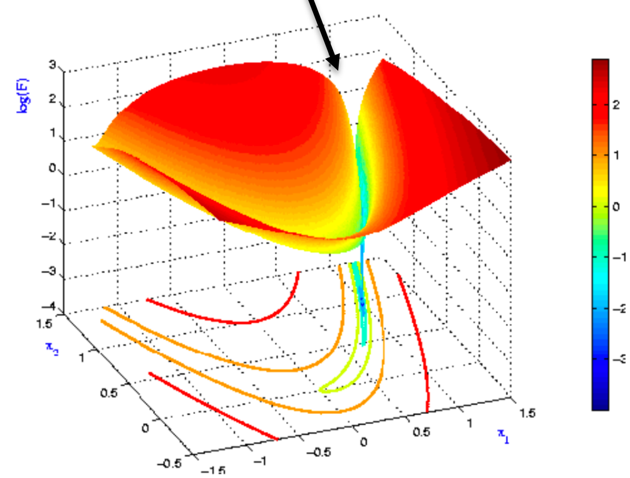
- More common than local minima/maxima are:
  - Another kind of zero gradient points: saddle points
    - At saddle, Hessian has both positive and negative values
      - Positive: cost greater than saddle point
      - Negative values have lower value
    - In low dimensions:
      - Local minima are more common
    - In high dimensions:
      - Local minima are rare, saddle points more common
- For Newton's saddle points pose a problem
  - Explains why second-order methods have not replaced gradient descent



Contains both positive and negative curvature
Function is $f(\boldsymbol{x})=x_1^2-x_2^2$

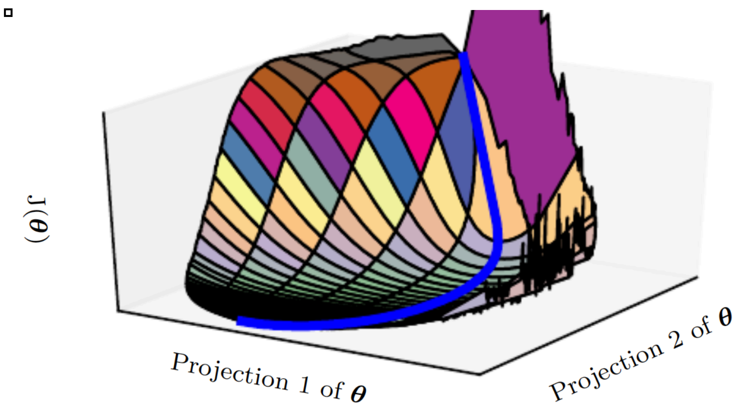
Plateau and Ravine





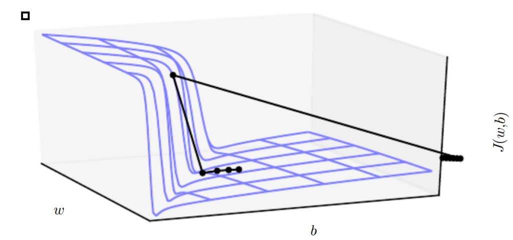
Cost Function of Neural Network

- Visualizations are similar for
  - Feedforward networks
  - Convolutional networks
  - Recurrent networks



- Applied to object recognition and NLP tasks
- Primary obstacle is not multiple minima but saddle points
- Most of training time spent on traversing flat valley of the Hessian matrix or circumnavigating tall “mountain” via an indirect arcing path

4.Cliffs and Exploding Gradients



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

5. Long-Term Dependencies

- When computational graphs become extremely deep, as with
  - feed-forward networks with many layers
  - RNNs which construct deep computational graphs by repeatedly applying the same operation at each time step
- Repeated application of same parameters gives rise to difficulties

6. Inexact Gradients

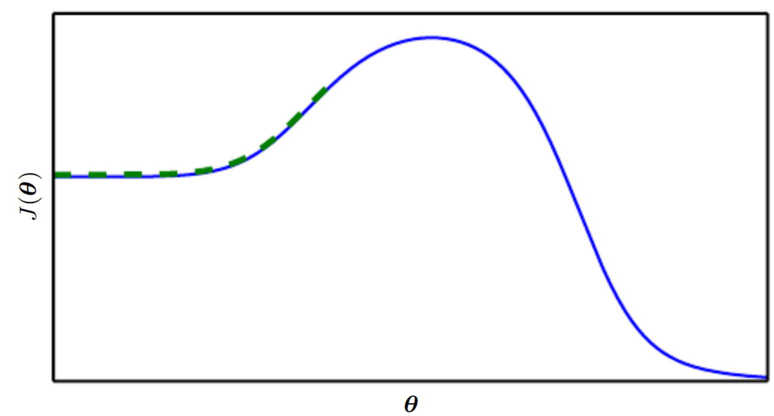
- Optimization algorithms assume we have access to exact gradient or Hessian matrix
- In practice we have a noisy or biased estimate
  - Every deep learning algorithm relies on sampling-based estimates
    - In using minibatch of training examples
  - In other case, objective function is intractable
    - In which case gradient is intractable as well
    - Contrastive Divergence gives a technique for approximating the gradient of the intractable log-likelihood of a Boltzmann machine

7. Poor Correspondence between Local and Global Structure

- It can be difficult to make a single step if:
  - $J(\boldsymbol{\theta})$ is poorly conditioned at the current point $\boldsymbol{\theta}$
  - $\boldsymbol{\theta}$ lies on a cliff
  - $\boldsymbol{\theta}$ is a saddle point hiding the opportunity to make progress downhill from the gradient
- It is possible to overcome all these problems and still perform poorly
  - if the direction that makes most improvement locally does not point towards distant regions of much lower cost

Need for good initial points

- Optimization based on local downhill moves can fail if local surface does not point towards the global solution
- Research directions are aimed at finding good initial points for problems with a difficult global structure
  - Ex: no saddle points or local minima
    - Trajectory of circumventing such mountains may be long and result in excessive training time



8. Theoretical Limits of Optimization

- There are limits on the performance of any optimization algorithm we might design for neural networks
- These results have little bearing on the use of neural networks in practice
  - Some apply only to networks that output discrete values
    - Most neural networks output smoothly increasing values
  - Some show that there exist problem classes that are intractable
    - But difficult to tell whether problem falls in thet class