Optimization

Professor Marie Roch

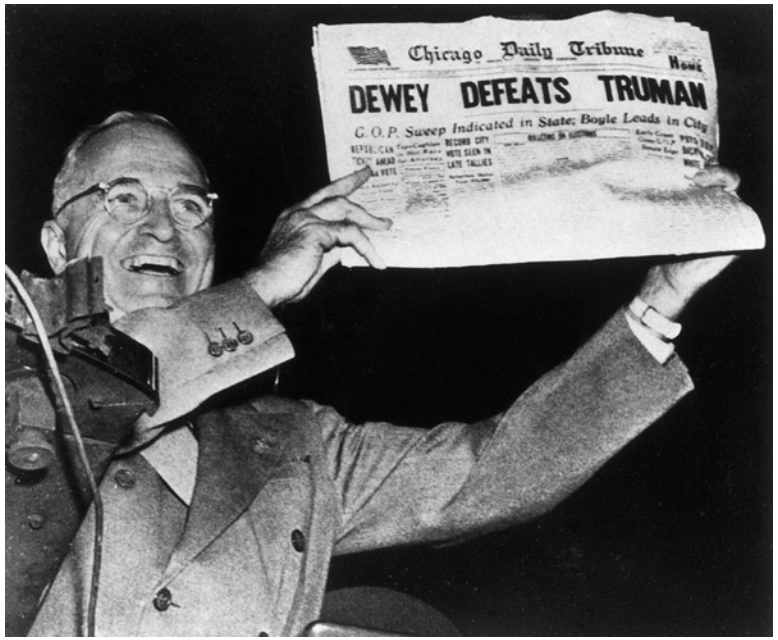
SAN DIEGO STATE UNIVERSITY

Loss functions

Loss function: Penalty for when we get it wrong

examples:

- 0-1 loss $L_{0-1}(\hat{y}) = \begin{cases} 0 & \hat{y} = y \\ 1 & \hat{y} \neq y \end{cases}$
- MSE $L_{MSE}(\hat{y}) = (\hat{y} - y)^2$



1948 US Presidential election won by… Harry Truman

$$L_{0-1}(winner = Dewey) = 1$$

0-1 loss well suited to classification

Risk

- When optimizing, we are not so concerned with the loss of an individual example
- Goal is to minimize the expected loss which is known as the risk

$$J^*(\theta) = E_{(x,y)\sim p_{data}}[L(f(x|\theta), y)]$$

where p_{data} is the actual (and probably unknown) distribution of the data.

Empirical risk

- Since we do not have p_{data} we usually use a training set, $\hat{p}_{data}$, and compute the *empirical risk* using the sample expected value:

$$J(\theta) = E_{(x,y)\sim\hat{p}_{data}}[L(f(x|\theta), y)]$$

Optimizers and classification

- $L_{0\text{-}1}$ makes sense for classification, but what if we want to minimize it?

$$L_{0-1}(\hat{y}) = \begin{cases} 0 & \hat{y} = y \\ 1 & \hat{y} \neq y \end{cases} \qquad \nabla L_{0-1}(\hat{y})?$$

- Difficult to minimize… leads us to *surrogate loss* functions that are easy to optimize

Surrogate loss functions

- We have already seen a couple
  - cross entropy
  - negative log likelihood for binary classifiers

- Advantages
  - easier to optimize
  - can continue to learn *even when empirical loss is 0*
    - might be good: can learn to better distinguish between classes
    - might be bad: can lead to overfitting

Batch learning

1. Compute gradient for each example & target in the *entire training set*
2. Update model in mean gradient direction
3. Go to 1 if not done

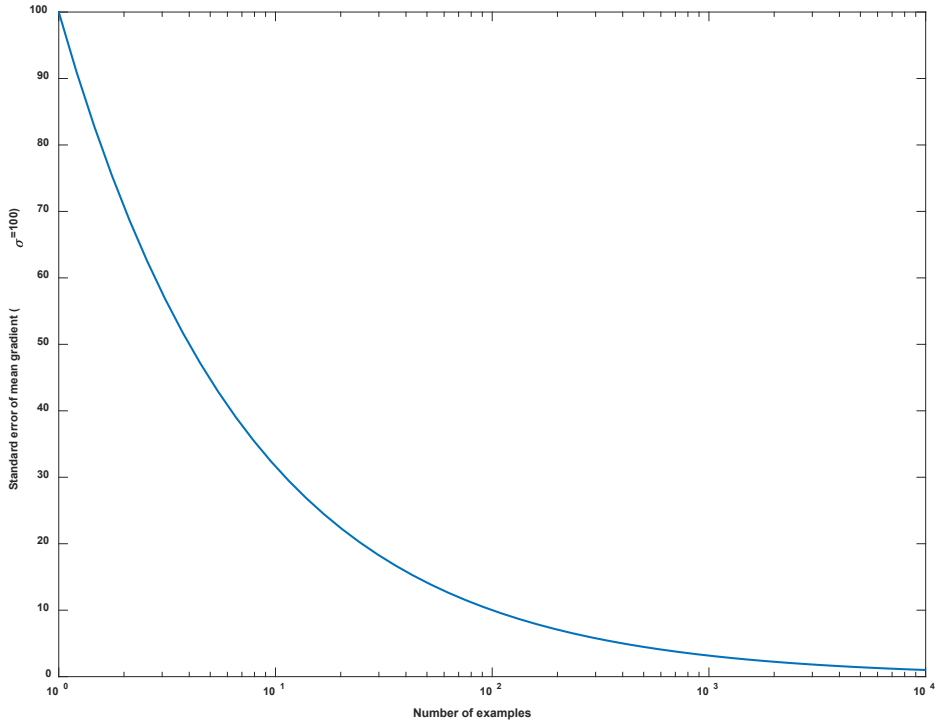
- Tends to have a good estimate of the gradient (standard error of mean estimator is $\sigma/\sqrt{n}$)
- Learns slowly

Stochastic, or minibatch learning

- Standard error: Function of stddev and number of samples

$$\triangleq \sigma/\sqrt{n}$$

- Implies diminishing gains as n grows



Stochastic, or minibatch learning

We can have a small number of examples and achieve decent estimates of the gradient.

Noise in the gradient estimate can serve as a regularizer.

Batch size considerations

- Too small – Underutilizes parallel hardware
- Too large –Excessive memory demands, slow learning

Stochastic batch size

- Gradient only algorithms – small batch sizes okay (e.g. 100)
- Some algorithms rely on the 2nd derivative:
  - 2nd derivative matrix is called a Hessian
  - Requires larger batch sizes to estimate reliably (e.g., 10,000)

Stochastic learning

- Samples are assumed to be independent
- If not, can produce a biased estimator of the loss surrogate and its gradient

- Many data sets have correlated samples; batches from such sets should be sampled randomly

Challenges in optimization

- Ill-conditioned Hessians can wreak havoc

oh, oh…
rabbit hole

British Postal Service, Graham Baker-Smith 2015

Hessian matrix – H: $\left[\frac{\partial^2 y}{\partial x_i \partial x_j}\right]$

$g: \mathbb{R}^m \rightarrow \mathbb{R}^n$

$$\left[\frac{\partial y}{\partial x}\right] = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_M} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & & \frac{\partial y_2}{\partial x_M} \\ \frac{\partial y_3}{\partial x_1} & \frac{\partial y_3}{\partial x_2} & & \frac{\partial y_1}{\partial x_M} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial y_N}{\partial x_1} & \frac{\partial y_N}{\partial x_2} & \cdots & \frac{\partial y_N}{\partial x_M} \end{bmatrix}$$

$$\left[\frac{\partial^2 y}{\partial x_i \partial x_j}\right] = \begin{bmatrix} \frac{\partial^2 y_1}{\partial x_1^2} & \frac{\partial^2 y_1}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 y_1}{\partial x_1 \partial x_M} \\ \frac{\partial^2 y_2}{\partial x_2 \partial x_1} & \frac{\partial^2 y_2}{\partial x_2^2} & & \frac{\partial^2 y_2}{\partial x_2 \partial x_M} \\ \frac{\partial^2 y_3}{\partial x_3 \partial x_1} & \frac{\partial^2 y_3}{\partial x_3 \partial x_2} & & \frac{\partial^2 y_1}{\partial x_3 \partial x_M} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial^2 y_N}{\partial x_M \partial x_1} & \frac{\partial^2 y_N}{\partial x_M \partial x_2} & \cdots & \frac{\partial^2 y_N}{\partial x_M^2} \end{bmatrix}$$

Jacobian (saw earlier)

Hessian is 2nd partial derivative

Goodall et al. 4.3.1

Matrix condition numbers

- Some matrices have eigen decompositions

$$A = Q\Lambda Q^T \text{ where}$$

  - $diag(\Lambda) = \lambda$,
  - Q contains eigen vectors,
  - and $QQ^T = I$
- More generally, every real matrix has a singular value decomposition

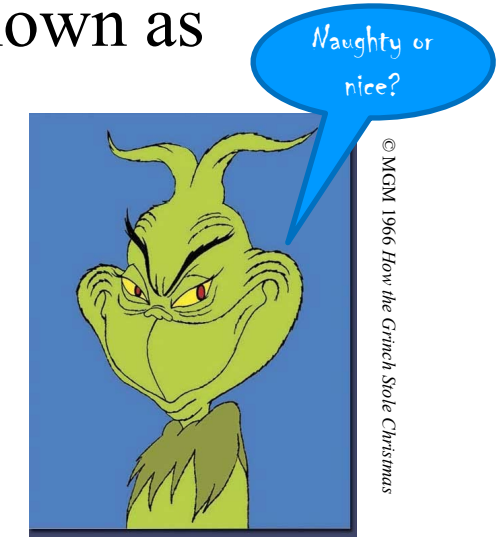
Singular value decomposition (SVD)

$$A = UDV^T$$

- A is $m \times n$
- U is $m \times m, V$ is $n \times n$
- D is diagonal and its elements along the diagonal are known as singular values

SVD is important for

- computing pseudo-inverses
- determining if matrices are well behaved



SVDs and condition numbers

- Condition number $\triangleq \frac{\max_i(D_{i,i})}{\min_i(D_{i,i})}$

- When the condition number is large, small changes in the input can produce large changes in the output

Ill conditioned Hessians can wreak havoc

A 2nd order Taylor-series expansion of the cost function about point moved in a direction opposite the gradient $x^{(0)} - \epsilon g$ shows:

$$f\left(x^{(0)} - \epsilon g\right) \approx f\left(x^{(0)}\right) - \epsilon g^T G + \frac{1}{2}\epsilon^2 g^T H g \quad \text{(Goodfellow et al. 4.9).}$$

When Hessian H is ill conditioned:

- even small ϵ can overshoot and increase the cost.
- Learning rate must be shrunk in this case.

Ill-conditioned Hessians can wreak havoc

To determine if an ill-conditioned Hessian is a problem, monitor:

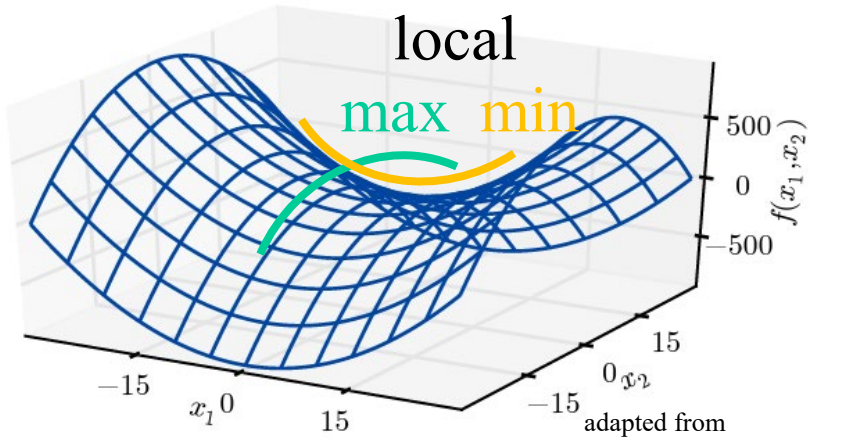
- squared gradient $g^T g$
- and $g^T H g$

Challenges continued

- Local minima
  - Not usually a problem
  - Many local minima have similar valued cost functions
  - However, it is always possible that the global minimum is much lower

Challenges continued – saddle points

- Hessian's eigen values drive loss
- Moving along eigen vectors with
  - + eigen values increase cost
  - - eigen values decrease cost



adapted from Goodfellow et al. Fig 4.5

All + → local min
All - → local max

Saddle points

- In low dimensions, random functions typically have local minima
- In high dimensions, local minima are rare, but saddle points are common

  (As input space $\mathbb{R}^N$ grows, $\frac{\#saddle\ points}{\#\ local\ minima}$ grows exponentially)

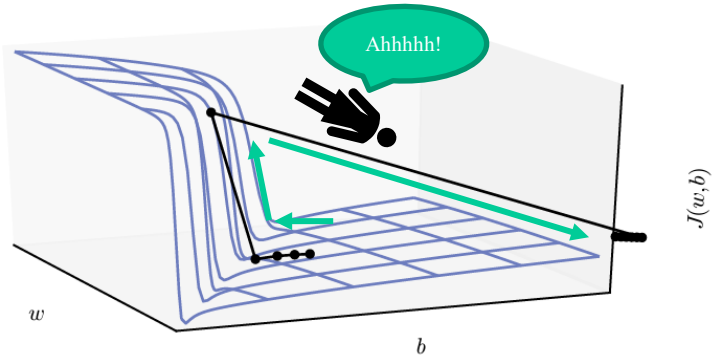
Saddle points

- Theory suggests that saddle points tend to be high cost, so how we handle them is important.
- Gradients at saddle points can be shallow
- First order gradient descent tends to escape many saddle points
- Some techniques try to find points where the gradient is zero (e.g., Newton's method). This can be problematic.

Challenges continued

- Plateaus
  - Wide flat regions. Problematic for all numerical optimization algorithms
- Cliff structures
  - Very steep gradients can result in large jumps
  - Gradient clipping prevents this from occurring (max norm for step size).

Goodfellow et al. Fig 8.3



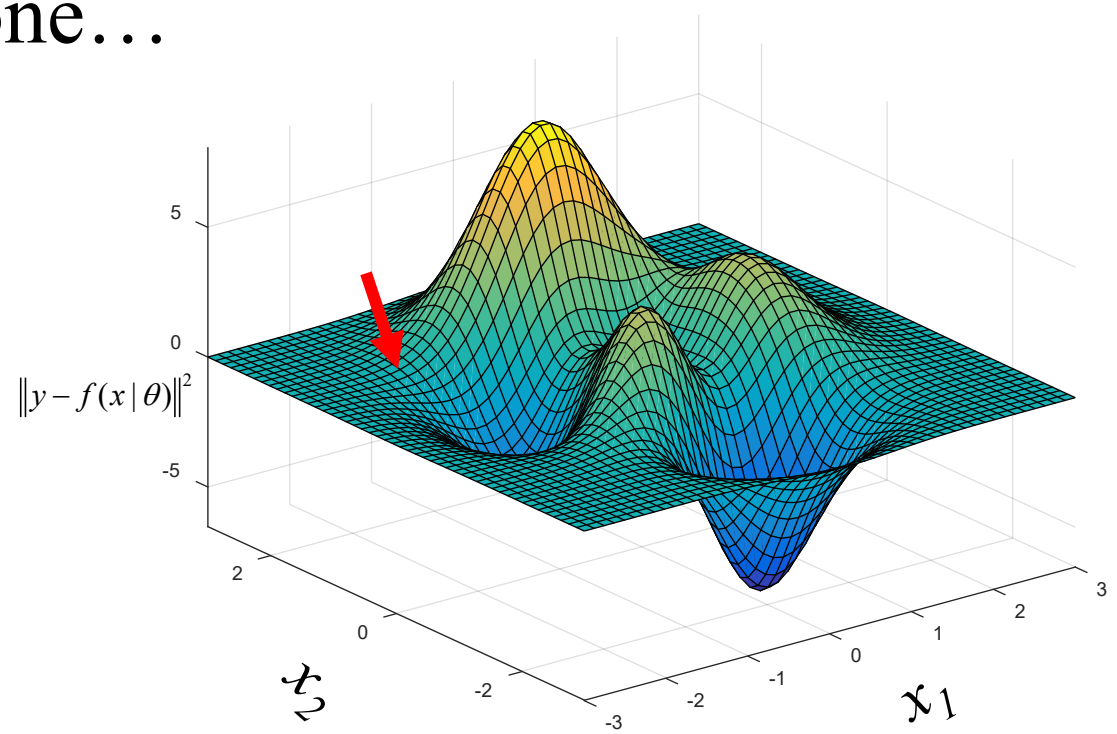
Challenges continued

- Long-term dependencies
  (we will discuss this when we cover recurrent neural nets)

- Inexact gradients
  Just like the distributions we learn, these are only approximations…

Challenges continued

- Our local point in optimization space may just not be a good one…



Ways to cope:

- non-local moves (e.g., simulated annealing)
- find a good starting point (current research direction)

Stochastic gradient descent (SGD)

Given learning rate ϵ

while stop criterion not met

randomly select m examples & labels (x, y)

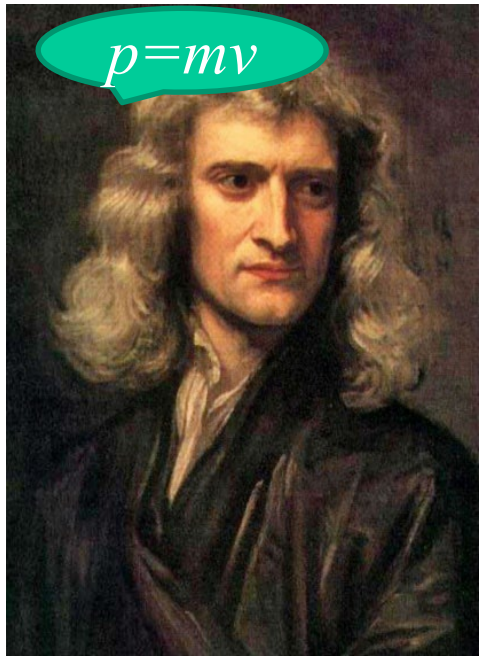
estimate gradient $\hat{g} = \frac{1}{m} \nabla_\theta \sum_i L\left(f(x^{(i)}|\theta), y^{(i)}\right)$

update model $\theta = \theta - \epsilon \hat{g}$

Common to diminish learning rate over time with time specific ϵ_t

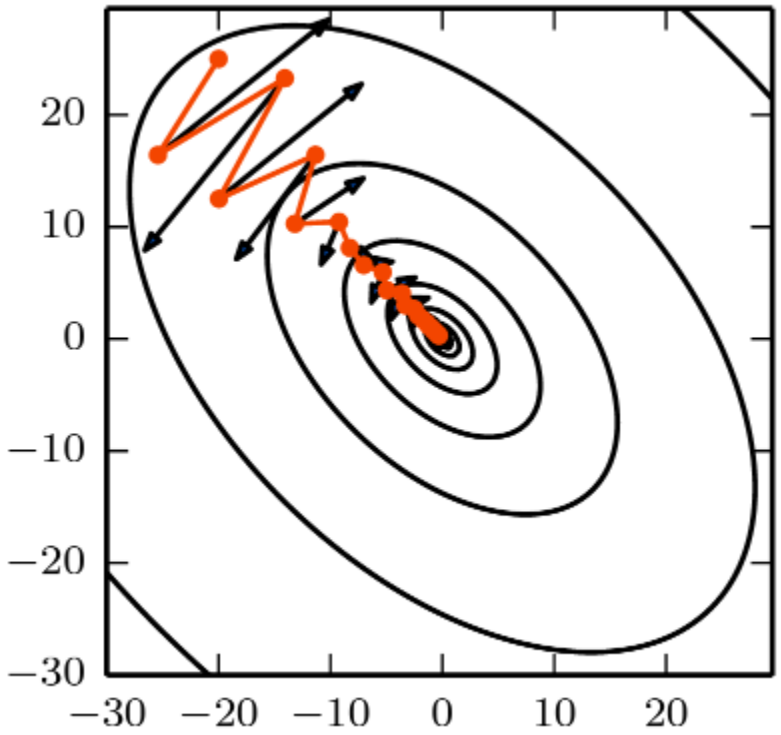
Momentum

Sir Isaac says:



Godfrey Kneller

- Key idea: Use previous gradients to keep us moving in the right direction.



Black gradient vectors grow due to a poorly conditioned Hessian

SGD with momentum

Given learning rate ϵ and initial velocity v

while stop criterion not met

randomly select m examples & labels (x, y)

estimate gradient $\hat{g} = \frac{1}{m}\nabla_\theta \sum_i L\left(f(x^{(i)}|\theta), y^{(i)}\right)$

update velocity $v = \alpha v - \epsilon \hat{g}$

update model $\theta = \theta + v$

Nesterov momentum variant: $\hat{g} = \frac{1}{m}\nabla_\theta \sum_i L\left(f(x^{(i)}|\underbrace{\theta + \alpha v}_{\text{apply momentum to model when estimating gradient}}), y^{(i)}\right)$

(Doesn't help that much with SGD, but does in other cases.)

Parameter initialization

- Key goal: break symmetry between units
- Most initialization based on heuristics
  - biases usually small constants
  - weights from uniform or Gaussian distributions
    - scale seems to be important
    - distribution family does not
  - see Goodfellow et al. for a variety of strategies

Adaptive learning rates

- Learning rate has a large impact on success of neural networks
- Several algorithms have attempted to adapt the learning rates automatically
- RMSProp – Learning rate weighted by a function of moving average of gradients

RMSProp

Given learning rate ϵ, decay rate ρ, $r = 0, \delta = 10^{-6}$

while stop criterion not met

randomly select m examples & labels (x, y)

estimate gradient $\hat{g} = \frac{1}{m} \nabla_\theta \sum_i L\left(f(x^{(i)}|\theta), y^{(i)}\right)$

accumulate gradient^2 $r = \rho r + (1 - \rho)\hat{g} \odot \hat{g}$

update model $\theta = \theta - \frac{\epsilon}{\sqrt{\delta + r}} \odot \hat{g}$

$\odot$ element by element multiplication

$\sqrt{\delta + r}$ element by element root

Adaptive moments (Adam)

- Moments of a random variable are its expected value raised to the n^{th} power:

$$E[X], E[X^2], \ldots, E[X^n]$$

- Adam uses leaky estimates of the first two moments of the gradient, giving it characteristics of both SGD with momentum and RMSProp

Adam

Given step size ϵ, decay $\rho_1, \rho_2 \in [0,1), \delta = 10^{-8}$

s=0, r=0 (moments 1 and 2), t=0

while stop criterion not met

randomly select m examples & labels (x, y)

estimate gradient $\hat{g} = \frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)} | \theta), y^{(i)})$

biased estimators

$s = \rho_1 r + (1 - \rho_1)\hat{g}$ $\quad \hat{E}[g]$

$r = \rho_2 r + (1 - \rho_2)\hat{g} \odot \hat{g}$ $\quad \hat{E}[g^2]$

Adam

(continuation of while loop)

$t = t + 1$

correct for biases

$$\hat{s} = \frac{s}{1 - \rho_1^t} \quad \hat{E}[g]$$

$$\hat{r} = \frac{r}{1 - \rho_2^t} \quad \hat{E}[g^2]$$

update model

similar to SGD w/momentum

$$\theta = \theta - \epsilon \frac{\hat{s}}{\sqrt{\delta + \hat{r}}} \quad \text{element-wise operations}$$

similar to RMSprop

Optimizers

- All the optimizers we have looked at are *first order* optimizers.

- No single algorithm has been shown to be the best

Second order optimizers

- Use the Hessian (or an approximation)
- We will not cover these in detail, but two examples covered in text
  - Newton's method – uses 2nd order Taylor expansion
  - Conjugate gradient descent – when gradient direction changes, pick a direction that does not undo the progress along the gradient