Optimization

Professor Marie Roch

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

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

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

Empirical risk

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

\[ J(\theta) = E_{(x,y) \sim \hat{p}_{\text{data}}}[L(f(x \mid \theta), y)] \]
Optimizers and classification

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

\[ L_{0-1}(\hat{y}) = \begin{cases} 0 & \hat{y} = y \\ 1 & \hat{y} \neq y \end{cases} \quad \nabla L_{0-1}(\hat{y})? \]

- 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 driven by $\sigma/\sqrt{n}$
- 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)
• Algorithms that rely on Hessians require more data to estimate (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
Matrix condition numbers

• We have seen that some matrices have eigendecompositions
  \[ A = Q \Lambda Q^T \]  where \( \Lambda \) contains eigenvalues, and \( Q \) contains eigenvectors, 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 $\Delta \geq \frac{\max_{i} (D_{ii})}{\min_{i} (D_{ii})}$

- 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 shows

$$f(x^{(0)} - \epsilon g) \approx f(x^{(0)}) - \epsilon g + \frac{1}{2} \epsilon^2 g^T H g$$  \hspace{1cm} (Goodfellow et al. 4.9)

so when H is ill conditioned, even smaller values of $\epsilon$ can cause us to 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 $g^T g$
• $g^T H g$

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 has eigen values with +/- values indicating.
  • Moving along eigen vectors with
    – + eigen values increase cost
    – - eigen values decrease cost

Saddle points

• In low dimensions, random functions typically have local minima
• In high dimensions, local minima are rare, but saddle points are common
  (saddle points:local minima ratio grows exponentially with dimensionality)
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 prevent this from occurring (max norm for step size)
Challenges continued

• Long-term dependencies
  (discussed in RNN context)

• 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 $\epsilon$
while stop criterion not met
randomly select $m$ examples & labels $(x, y)$
estimate gradient $\hat{g} = \frac{1}{m} \nabla \sum L(f(x^i | \theta), y^i)$
update model $\theta = \theta - \epsilon \hat{g}$

Common to diminish learning rate over time with
time specific $\epsilon_t$

Momentum

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

Sir Isaac says: $p = mv$

Black gradient vectors grow due to a poorly conditioned Hessian
SGD with momentum

Given learning rate $\epsilon$ and initial velocity $v$
while stop criterion not met
randomly select $m$ examples & labels $(x, y)$
estimate gradient $\hat{g} = \frac{1}{m} \nabla \theta \sum L(f(x^{(i)} | \theta), y^{(i)})$
update velocity $v = \alpha v - \epsilon \hat{g}$
update model $\theta = \theta + v$

Nesterov momentum variant: $\hat{g} = \frac{1}{m} \nabla \theta \sum L(f(x^{(i)} | \theta + \alpha v), y^{(i)})$
(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 $\epsilon$, decay rate $\rho$, $r = 0$, $\delta = 10^{-6}$

while stop criterion not met

randomly select $m$ examples & labels $(x, y)$
estimate gradient $\hat{g} = \frac{1}{m} \nabla_x \sum_r L(f(x) | (\theta), y)$$accumulate gradient^2 r = \rho r + (1 - \rho) \hat{g} \odot \hat{g}$$update model \theta = \theta - \frac{\epsilon}{\sqrt{\delta + r}} \odot 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\textsuperscript{th} power:
  \[ E[X], E[X^2], ..., E[X^n] \]
- 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 \( \epsilon \), 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 \( \hat{g} = \frac{1}{m} \sum_{i} L(f(x^{(i)} | \theta), y^{(i)}) \)
- 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)

\[ t = t + 1 \]
\[ \hat{s} = \frac{s}{1 - \rho^t_i} \hat{E}[g] \]
\[ \hat{r} = \frac{r}{1 - \rho^t_i} \hat{E}[g^2] \]

update model

\[ \theta = \theta - \epsilon \frac{\hat{s}}{\sqrt{\hat{\delta} + \hat{r}}} \]

element-wise operations

similar to SGD w/momentum

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 2\textsuperscript{nd} order Taylor expansion
  - Conjugate gradient descent – when gradient direction changes, pick a direction that does not undo the progress along the gradient