Lecture 19 Additional Material: Optimization

CS109A Introduction to Data Science
Pavlos Protopapas and Kevin Rader
Outline

Optimization

- Challenges in Optimization
- Momentum
- Adaptive Learning Rate
- Parameter Initialization
- Batch Normalization
Learning vs. Optimization

Goal of learning: **minimize generalization error**

In practice, **empirical risk minimization**:

\[
J(\theta) = \mathbb{E}_{(x,y) \sim p_{data}} \left[ L(f(x; \theta), y) \right]
\]

\[
\hat{J}(\theta) = \frac{1}{m} \sum_{i=1}^{m} L(f(x^{(i)}; \theta), y^{(i)})
\]

Quantity optimized different from the quantity we care about
Batch vs. Stochastic Algorithms

Batch algorithms

• Optimize empirical risk using exact gradients

Stochastic algorithms

• Estimates gradient from a small random sample

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

*Large mini-batch*: gradient computation expensive

*Small mini-batch*: greater variance in estimate, longer steps for convergence
Critical Points

Points with zero gradient

$2^{nd}$-derivate (Hessian) determines curvature

Goodfellow et al. (2016)
Stochastic Gradient Descent

Take small steps in direction of **negative gradient**
Sample $m$ examples from training set and compute:

Update parameters:

$$g = \frac{1}{m} \sum_i \nabla L(f(x^{(i)}; \theta), y^{(i)})$$

$$\theta = \theta - \epsilon_k g$$

In practice: **shuffle** training set once and pass through multiple times
Stochastic Gradient Descent

$$J(\theta)$$

Oscillations because updates do not exploit curvature information

Goodfellow et al. (2016)
Outline

Optimization

• **Challenges in Optimization**
• Momentum
• Adaptive Learning Rate
• Parameter Initialization
• Batch Normalization
Local Minima

Ideally, we would like to arrive at the global minimum, but this might not be possible.

This local minimum performs nearly as well as the global one, so it is an acceptable halting point.

This local minimum performs poorly, and should be avoided.

Goodfellow et al. (2016)
Local Minima

Old view: local minima is major problem in neural network training

Recent view:

- For sufficiently large neural networks, most local minima incur low cost
- Not important to find true global minimum
Saddle Points

Recent studies indicate that in high dim, saddle points are more likely than local min

Gradient can be very small near saddle points

Goodfellow et al. (2016)
Saddle Points

SGD is seen to escape saddle points
- Moves down-hill, uses noisy gradients

Second-order methods get stuck
- Solves for a point with zero gradient

Goodfellow et al. (2016)
Poor Conditioning

Poorly conditioned Hessian matrix

- **High curvature**: small steps leads to huge increase

Learning is slow despite strong gradients

Oscillations slow down progress
No Critical Points

Some cost functions do not have critical points. In particular classification.
No Critical Points

Gradient norm increases, but validation error decreases

Convolution Nets for Object Detection

Goodfellow et al. (2016)
Exploding and Vanishing Gradients

\[ h_1 = Wx \]
\[ h_i = Wh_{i-1}, \quad i = 2 \ldots n \]
\[ y = \sigma(h_1^n + h_2^n), \quad \text{where} \quad \sigma(s) = \frac{1}{1 + e^{-s}} \]
Exploding and Vanishing Gradients

Suppose \( \mathbf{W} = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \):

\[
\begin{bmatrix}
  h_1^1 \\
  h_2^1 \\
\end{bmatrix} = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad ... \quad
\begin{bmatrix}
  h_1^n \\
  h_2^n \\
\end{bmatrix} = \begin{bmatrix} a^n & 0 \\ 0 & b^n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}
\]

\( y = \sigma(a^n x_1 + b^n x_2) \)

\[
\nabla y = \sigma'(a^n x_1 + b^n x_2) \begin{bmatrix} na^{n-1} x_1 \\ nb^{n-1} x_2 \end{bmatrix}
\]
Exploding and Vanishing Gradients

Suppose $x = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

Case 1: $a = 1, b = 2$:

$y \to 1, \nabla y \to \begin{bmatrix} n \\ n2^{n-1} \end{bmatrix}$ Explodes!

Case 2: $a = 0.5, b = 0.9$:

$y \to 0, \nabla y \to \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ Vanishes!
Exploding and Vanishing Gradients

Exploding gradients lead to cliffs
Can be mitigated using gradient clipping

Goodfellow et al. (2016)
Outline

Optimization

- Challenges in Optimization
- **Momentum**
- Adaptive Learning Rate
- Parameter Initialization
- Batch Normalization
Momentum

SGD is slow when there is high curvature

Average gradient presents faster path to opt:
  – vertical components cancel out
Momentum

Uses \textit{past gradients} for update

Maintains a new quantity: ‘velocity’

Exponentially decaying average of gradients:

\[
\mathbf{v} = \alpha \mathbf{v} + (-\varepsilon \mathbf{g})
\]

Current gradient update

\(\alpha\) controls how quickly effect of past gradients decay
Momentum

Compute gradient estimate:

\[ g = \frac{1}{m} \sum_i \nabla_\theta L(f(x^{(i)}; \theta), y^{(i)}) \]

Update velocity:

\[ v = \alpha v - \varepsilon g \]

Update parameters:

\[ \theta = \theta + v \]
Momentum

Damped oscillations: gradients in opposite directions get cancelled out

Goodfellow et al. (2016)
Nesterov Momentum

Apply an *interim* update:

\[ \tilde{\theta} = \theta + \nu \]

Perform a correction based on gradient at the interim point:

\[
g = \frac{1}{m} \sum_i \nabla_{\theta} L(f(x^{(i)}; \tilde{\theta}), y^{(i)})
\]

\[
\nu = \alpha \nu - \epsilon g
\]

\[
\theta = \theta + \nu
\]

Momentum based on look-ahead slope
Outline

Optimization

• Challenges in Optimization
• Momentum
• *Adaptive Learning Rate*
• Parameter Initialization
• Batch Normalization
Adaptive Learning Rates

Oscillations along vertical direction
- Learning must be slower along parameter 2

Use a different learning rate for each parameter?
AdaGrad

- Accumulate squared gradients:
  \[ r_i = r_i + g_i^2 \]
- Update each parameter:
  \[ \theta_i = \theta_i - \frac{\varepsilon}{\delta + \sqrt{r_i}} g_i \]
- Greater progress along gently sloped directions

Inversely proportional to cumulative squared gradient
RMSProp

- For non-convex problems, AdaGrad can prematurely decrease learning rate
- Use **exponentially weighted average** for gradient accumulation

\[
\begin{align*}
  r_i &= \rho r_i + (1 - \rho) g_i^2 \\
  \theta_i &= \theta_i - \frac{\varepsilon}{\delta + \sqrt{r_i}} g_i
\end{align*}
\]
Adam

- **RMSProp + Momentum**
- Estimate first moment:
  \[ v_i = \rho_1 v_i + (1 - \rho_1) g_i \]
- Estimate second moment:
  \[ r_i = \rho_2 r_i + (1 - \rho_2) g_i^2 \]
- Update parameters:
  \[ \theta_i = \theta_i - \frac{\varepsilon}{\delta + \sqrt{r_i}} v_i \]

Also applies bias correction to \( v \) and \( r \)

Works well in practice, is fairly robust to hyper-parameters
Outline

Optimization

• Challenges in Optimization
• Momentum
• Adaptive Learning Rate
• Parameter Initialization
• Batch Normalization
Parameter Initialization

- Goal: **break symmetry** between units
  - so that each unit computes a different function
- Initialize all weights (not biases) **randomly**
  - Gaussian or uniform distribution
- **Scale of initialization?**
  - Large $\rightarrow$ grad explosion, Small $\rightarrow$ grad vanishing
Xavier Initialization

• Heuristic for all outputs to have unit variance

• For a fully-connected layer with $m$ inputs:

\[ W_{ij} \sim N\left(0, \frac{1}{m}\right) \]

• For ReLU units, it is recommended:

\[ W_{ij} \sim N\left(0, \frac{2}{m}\right) \]
Normalized Initialization

- Fully-connected layer with $m$ inputs, $n$ outputs:

$$W_{ij} \sim U\left(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right)$$

- Heuristic trades off between initialize all layers have same activation and gradient variance

- **Sparse** variant when $m$ is large
  - Initialize $k$ nonzero weights in each unit
Bias Initialization

• Output unit bias
  • Marginal statistics of the output in the training set

• Hidden unit bias
  • Avoid saturation at initialization
  • E.g. in ReLU, initialize bias to 0.1 instead of 0

• Units controlling participation of other units
  • Set bias to allow participation at initialization
Outline

Challenges in Optimization
Momentum
Adaptive Learning Rate
Parameter Initialization
Batch Normalization
Feature Normalization

Good practice to normalize features before applying learning algorithm:

Features in **same scale**: mean 0 and variance 1
  - Speeds up learning

\[
\boldsymbol{x}' = \frac{\boldsymbol{x} - \mu}{\sigma}
\]

- Feature vector
- Vector of mean feature values
- Vector of SD of feature values
Feature Normalization

Before normalization

After normalization
Internal Covariance Shift

Each hidden layer changes distribution of inputs to next layer: slows down learning

Normalize inputs to layer 2

Normalize inputs to layer n
Batch Normalization

Training time:
- Mini-batch of activations for layer to normalize

\[
H = \begin{bmatrix}
H_{11} & \cdots & H_{1K} \\
\vdots & \ddots & \vdots \\
H_{N1} & \cdots & H_{NK}
\end{bmatrix}
\]

- \(K\) hidden layer activations
- \(N\) data points in mini-batch
Batch Normalization

Training time:
– Mini-batch of activations for layer to normalize

where

\[ H' = \frac{H - \mu}{\sigma} \]

\[ \mu = \frac{1}{m} \sum_i H_i \]

\[ \sigma = \sqrt{\frac{1}{m} \sum_i (H - \mu)^2 + \delta} \]

Vector of mean activations across mini-batch
Vector of SD of each unit across mini-batch
Batch Normalization

Training time:
- Normalization can reduce expressive power
- Instead use:

$$\gamma H' + \beta$$

Learnable parameters

- Allows network to control range of normalization
Batch Normalization

Batch 1

Add normalization operations for layer 1

Batch $N$

$$\mu^1 = \frac{1}{m} \sum_i H_i,$$

$$\sigma^1 = \sqrt{\frac{1}{m} \sum_i (H - \mu)^2 + \delta}$$
Batch Normalization

\[ \mu^2 = \frac{1}{m} \sum_{i} H_{i,\cdot} \]

\[ \sigma^2 = \sqrt{\frac{1}{m} \sum_{i} (H - \mu)^2_i + \delta} \]

Add normalization operations for layer 2 and so on ...
Batch Normalization

Differentiate the joint loss for $N$ mini-batches

Back-propagate through the norm operations

Test time:
- Model needs to be evaluated on a single example
- Replace $\mu$ and $\sigma$ with running averages collected during training