INF721 - Deep Learning
L9: Advanced Optimization Algorithms

Prof. Lucas N. Ferreira
Universidade Federal de Viçosa

2024/2

1 Introduction

As discussed last lecture, training deep learning models is an interative process that alternates between training and validation to solve underfitting and overfitting problems. Amongs these two processes, training is by far the slowest one. In deep learning, where we want to train networks with many layers in large datasets, the ability to train models quickly is crucial. In this lecture, we'll discuss advanced optimization algorithms that can help accelerate the training process. These techniques are crucial for improving the efficiency and effectiveness of deep learning models, especially when dealing with large datasets.

2 Mini-batch Gradient Descent

When dealing with large datasets (e.g., 5 million images), it becomes impractical to load the entire dataset into memory for training. Thus, we can't take advange of vectorizaztion to speed up training. Mini-batch gradient descent solves this problem by dividing the training set X into smaller subsets called mini-batches $X^{\{t\}}$. The mini-batch gradient descent algorithm works as follows:

Algorithm 1 Mini-batch Gradient Descent

$n\_batches \leftarrow m//s$
for $e = 1$ to n_epochs **do**
  for $t = 1$ to $n\_batches$ **do**
    $\hat{y}^{\{t\}} \leftarrow \text{forward\_pass}(X^{\{t\}})$
    $l^{\{t\}} \leftarrow \frac{1}{s}\sum L(\hat{y}^{\{t\}}, y^{\{t\}})$
    $dW^{\{t\}}, db^{\{t\}} \leftarrow \text{backward\_pass}(X^{\{t\}}, y^{\{t\}})$
    $W^{[l]} \leftarrow W^{[l]} - \alpha \cdot dW_t$
    $b^{[l]} \leftarrow b^{[l]} - \alpha \cdot db_t$
  end for
end for

Where:

- m is the total number of training examples
- s is the mini-batch size
- α is the learning rate
- $X^{\{t\}}$ and $Y^{\{t\}}$ are the input and target output for mini-batch t

The Gradient Descent algorithm has different names depending on the size of the mini-batch s:

- **Batch Gradient Descent**: $s = m$ (uses the entire dataset)
  - One weight update per epoch for the entire dataset
  - Slow but precise updates due to exact gradient computation
- **Stochastic Gradient Descent**: $s = 1$ (uses one example)
  - m updates per epoch, one for each example
  - Very fast but imprecise weight updates due to noisy gradients
  - Doesn't use vectorization, which can be inefficient
- **Mini-batch Gradient Descent**: $1 < s < m$ (uses a subset of the dataset)
  - Multiple updates per epoch, one for each mini-batch
  - Fast updates with approximate gradients
  - Balances speed and precision

2.1 Choosing Batch Size

- For small datasets: Use Batch Gradient Descent
- For large datasets: Use Mini-batch Gradient Descent
- Mini-batch size (hyperparameter):
  - Typically a power of two (e.g., 32, 64, 128, 256, 512, 1024)
  - Should fit in your CPU/GPU memory

2.2 Learning Curves

Learning curves show the evolution of the loss function over time (iterations or epochs) during training. Here's an example of learning curves for Batch Gradient Descent and Mini-batch Gradient Descent:

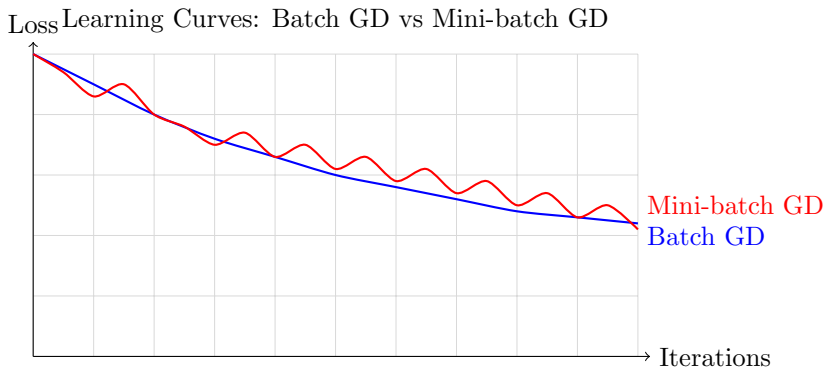

Figure 1: Comparison of learning curves for Batch Gradient Descent (BGD) and Mini-batch Gradient Descent (MBGD)

- Batch GD computes the exact gradient using the entire dataset in each iteration, resulting in a steady, monotonic decrease in loss.
- Mini-batch GD computes approximate gradients using small subsets of the dataset, leading to a noisier, more jagged decrease in loss.

3 Gradient Descent with Momentum

Gradient Descent with Momentum accelerates the optimization process by computing the Exponential Moving Average (see Appendix) of gradients. This helps to smooth out the updates and speed up convergence.

Algorithm 2 Gradient Descent with Momentum

Initialize $V_{dw}, V_{db} = 0$
for each mini-batch **do**
    Compute gradients $dw, db = \text{backward}(X^{\{t\}})$
    $V_{dw} = \beta \cdot V_{dw} + (1 - \beta)dw$
    $V_{db} = \beta \cdot V_{db} + (1 - \beta)db$
    $W^{[l]} = W^{[l]} - \alpha V_{dw}$
    $b^{[l]} = b^{[l]} - \alpha V_{db}$
end for

Where:

- V_{dw}, V_{db} are the averaged gradients

- β is the exponential moving average hyperparameter, also called the momentum hyperparameter (typically 0.9)
- α is the learning rate

3.1 Advantages of Gradient Descent with Momentum

- Accelerates convergence, especially in scenarios with high curvature or small but consistent gradients
- Reduces oscillations in the vertical direction while maintaining fast learning in the horizontal direction
- Helps overcome local minima and saddle points

4 Root Mean Squared Propagation (RMSProp)

RMSProp is another optimization algorithm that adapts the learning rate for each parameter based on the history of squared gradients.

4.1 Algorithm

Algorithm 3 RMSProp

Initialize $S_{dw}, S_{db} = 0$
for each mini-batch **do**
    Compute gradients $dw, db = \text{backward}(X^{\{t\}})$
    $S_{dw} = \beta \cdot S_{dw} + (1 - \beta)dw^2$
    $S_{db} = \beta \cdot S_{db} + (1 - \beta)db^2$
    $W^{[l]} = W^{[l]} - \alpha \frac{dw}{\sqrt{S_{dw}}}$
    $b^{[l]} = b^{[l]} - \alpha \frac{db}{\sqrt{S_{db}}}$
end for

Where:

- S_{dw}, S_{db} are the exponentially weighted average of squared gradients
- β is the decay rate (typically 0.999)
- α is the learning rate

4.2 Advantages of RMSProp

- Adapts the learning rate for each parameter independently
- Handles different scales of gradients effectively
- Helps to mitigate the vanishing gradient problem

5 Adaptive Moment Estimation (Adam)

Adam is an optimization algorithm that combines ideas from both Gradient Descent with Momentum and RMSProp.

Algorithm 4 Adam Optimization

Initialize $V_{dw}, V_{db}, S_{dw}, S_{db} = 0$
for each mini-batch **do**
    Compute gradients $dw, db = \text{backward}(X_t)$
    $V_{dw} = \beta_1 \cdot V_{dw} + (1 - \beta_1)dw$
    $V_{db} = \beta_1 \cdot V_{db} + (1 - \beta_1)db$
    $S_{dw} = \beta_2 \cdot S_{dw} + (1 - \beta_2)dw^2$
    $S_{db} = \beta_2 \cdot S_{db} + (1 - \beta_2)db^2$
    $V_{dw}^{corrected} = \frac{V_{dw}}{1-\beta_1^t}$
    $V_{db}^{corrected} = \frac{V_{db}}{1-\beta_1^t}$
    $S_{dw}^{corrected} = \frac{S_{dw}}{1-\beta_2^t}$
    $S_{db}^{corrected} = \frac{S_{db}}{1-\beta_2^t}$
    $W^{[l]} = W^{[l]} - \alpha \frac{V_{dw}^{corrected}}{\sqrt{S_{dw}^{corrected}}}$
    $b^{[l]} = b^{[l]} - \alpha \frac{V_{db}^{corrected}}{\sqrt{S_{db}^{corrected}}}$
end for

Where:

- V_{dw}, V_{db} are the momentum terms
- S_{dw}, S_{db} are the RMSProp terms
- β_1 is the momentum decay rate (typically 0.9)
- β_2 is the RMSProp decay rate (typically 0.999)
- α is the learning rate
- t is the iteration number

5.1 Advantages of Adam

- Combines the benefits of both Momentum and RMSProp
- Adapts the learning rate for each parameter independently
- Includes bias correction for more accurate initial estimates
- Generally performs well across a wide range of problems

6 Conclusion

This lecture covered several advanced optimization algorithms used in deep learning:

- Mini-batch Gradient Descent
- Gradient Descent with Momentum
- Root Mean Squared Propagation (RMSProp)
- Adaptive Moment Estimation (Adam)

These algorithms aim to improve the training process of neural networks by addressing issues such as slow convergence and noisy gradients. Understanding these optimization techniques is crucial for effectively training deep learning models, especially when dealing with large datasets and complex architectures.

In practice, Adam is often the go-to optimization algorithm for many deep learning practitioners due to its robust performance across various tasks. However, it's important to experiment with different optimizers and hyperparameters to find the best configuration for your specific problem.

Exercises

1. Which of the following statements about mini-batch gradient descent is correct?

    a) It uses the entire dataset for each update

    b) It uses only one example for each update

    c) It balances speed and precision by using subsets of the dataset

    d) It is always slower than batch gradient descent

2. In the context of optimization algorithms, what does the term "momentum" refer to?

    a) The speed at which the algorithm converges

    b) The exponential moving average of gradients

    c) The learning rate of the algorithm

    d) The size of the mini-batch

3. If $\beta = 0.95$ in an exponential moving average, approximately how many recent samples does the average effectively consider?

    a) 10 samples

    b) 20 samples

    c) 30 samples

    d) 40 samples

4. Which optimization algorithm combines ideas from both Gradient Descent with Momentum and RMSProp?

    a) Stochastic Gradient Descent

    b) Batch Gradient Descent

    c) Adam

    d) Mini-batch Gradient Descent

5. In the RMSProp algorithm, what is the purpose of the term S_{dw}?

    a) It stores the exponentially weighted average of gradients

    b) It stores the exponentially weighted average of squared gradients

    c) It represents the learning rate

    d) It represents the mini-batch size

A Moving Averages

In statistics and signal processing, a moving average is a calculation used to analyze data points by creating a series of averages of different subsets of the full dataset. Moving averages are commonly used to smooth out short-term fluctuations and highlight longer-term trends or cycles in time series data. In the context of optimization algorithms, moving averages can help stabilize the learning process and improve convergence.

A.1 Types of Moving Averages

- **Simple Moving Average (SMA)**: The simple moving average is the unweighted mean of the previous T data points.

$$v_t = \frac{1}{T} \cdot \sum_{t=1}^{T} \theta_t$$

- **Weighted Moving Average (WMA)**: The weighted average has multiplying factors to give different weights to data at different positions in the sample window of size T.

$$v_t = \frac{1}{\sum_{t=1}^{T} w_t} \cdot \sum_{t=1}^{T} \theta_t \cdot w_t$$

- **Exponential Moving Average (EMA)**: The exponential moving average applies exponentially decreasing weights to the data points in the sample window. The EMA at time t is calculated as follows:

$$v_t = \beta v_{t-1} + (1 - \beta)\theta_t$$

Where:

- v_t is the moving average at time t
- θ_t is the sample value at time t
- β is the decay rate (typically close to 1, e.g., 0.9)

A.2 Properties of Exponential Moving Average

- v_t is approximately the average of the last $\frac{1}{1-\beta}$ samples
- Higher β values result in slower adaptation to new samples
- Examples:
  - $\beta = 0.9 \approx$ average of last 10 samples
  - $\beta = 0.98 \approx$ average of last 50 samples
  - $\beta = 0.5 \approx$ average of last 2 samples

A.3 Bias Correction

When using EMA, initial average values can be poor estimates. This can be addressed using bias correction:

$$v_t = \frac{\beta v_{t-1} + (1-\beta)\theta_t}{1-\beta^t}$$

This adjustment helps to obtain more accurate estimates, especially in the early stages of averaging.