Optimization Algorithms, teaching notes

IllusionCraft

Prepared By: Abhijit Singh Jowhari

1. Gradient Descent

Introduction

Gradient Descent is one of the most fundamental optimization algorithms used in machine learning. It iteratively adjusts parameters to minimize a given loss function.

Mathematical Formulation

Given a loss function $L(\theta)$, where θ represents the parameters of the model, the goal is to find the parameters θ that minimize L.

1. **Loss Function:** $L(\theta)$ 2. **Gradient:** The gradient of the loss function $\nabla_\theta L(\theta)$ is a vector of partial derivatives of L with respect to θ. 3. **Update Rule:**

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L(\theta_t)$$

where η is the learning rate.

Detailed Steps

1. **Initialization:** Initialize the parameters θ randomly. 2. **Compute Gradient:** Calculate the gradient of the loss function at the current parameters. 3. **Update Parameters:** Adjust the parameters in the direction of the negative gradient. 4. **Repeat:** Continue steps 2 and 3 until convergence (when the change in the loss function is below a threshold or after a fixed number of iterations).

Convergence and Learning Rate

- **Learning Rate (η):** A small learning rate may lead to slow convergence, while a large learning rate can cause divergence. Often, the learning rate is chosen through experimentation or using techniques like learning rate schedules. - **Convergence:** The algorithm converges when $\|\nabla_\theta L(\theta)\|$ is close to zero.

Advantages and Disadvantages

- **Advantages:** - Simplicity and ease of implementation. - Suitable for convex optimization problems. - **Disadvantages:** - Can be slow for large datasets. - Sensitive to the choice of learning rate. - Can get stuck in local minima for non-convex problems.

2. Stochastic Gradient Descent (SGD)

Introduction

Stochastic Gradient Descent is a variant of Gradient Descent where the gradient is estimated using a single or a small subset of data points, rather than the entire dataset.

Mathematical Formulation

1. **Loss Function:**

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} L_i(\theta)$$

where N is the number of training samples and L_i is the loss for the i-th sample. 2. **Stochastic Gradient:** The gradient is estimated using a single sample or a mini-batch:

$$\nabla_\theta L(\theta) \approx \nabla_\theta L_i(\theta)$$

3. **Update Rule:**

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L_i(\theta_t)$$

Detailed Steps

1. **Initialization:** Initialize the parameters θ randomly. 2. **Shuffling:** Randomly shuffle the training data. 3. **Iterate Over Data:** For each sample i in the training data: - Compute the gradient of the loss function for the sample. - Update the parameters using the computed gradient. 4. **Repeat:** Continue the process for a fixed number of epochs or until convergence.

Advantages and Disadvantages

- **Advantages:** - Faster iterations since gradients are computed on fewer data points. - Can escape local minima due to its stochastic nature. - Often reaches a good enough solution faster than batch gradient descent. - **Disadvantages:** - Higher variance in the parameter updates can lead to a less stable convergence path. - Requires more iterations to converge compared to batch gradient descent. - The noisy updates may cause the algorithm to never converge to the exact minimum.

3. Adam (Adaptive Moment Estimation)

Introduction

Adam is an advanced optimization algorithm that combines the benefits of two other extensions of stochastic gradient descent: Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp).

Mathematical Formulation

Adam maintains two moving averages of the gradient: the first moment (mean) and the second moment (uncentered variance).

1. **Initialize Parameters:** - $m_0 = 0$ (first moment vector) - $v_0 = 0$ (second moment vector) - $t = 0$ (time step) 2. **Hyperparameters:** - α: Learning rate - β_1: Decay rate for the first moment (typically 0.9) - β_2: Decay rate for the second moment (typically 0.999) - ϵ: A small constant to prevent division by zero (typically 10^{-8})

3. **Update Rule:** - Increment time step: $t = t + 1$ - Compute gradient: $g_t = \nabla_\theta L(\theta_t)$ - Update biased first moment estimate:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

- Update biased second moment estimate:

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

- Correct bias in first moment:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

- Correct bias in second moment:

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

- Update parameters:

$$\theta_{t+1} = \theta_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

Detailed Steps

1. **Initialization:** Initialize the parameters θ, m_0, v_0, and set $t = 0$. 2. **Iterate:** For each step: - Increment t. - Compute the gradient g_t. - Update the biased first and second moment estimates. - Compute bias-corrected estimates. - Update the parameters. 3. **Repeat:** Continue until convergence or for a predetermined number of iterations.

Advantages and Disadvantages

- **Advantages:** - **Adaptive Learning Rate:** Adjusts the learning rate for each parameter individually. - **Bias Correction:** Provides bias-corrected first and second moment estimates, leading to more accurate updates. - **Efficiency:** Combines the advantages of AdaGrad (good for sparse gradients) and RMSProp (good for non-stationary objectives). - **Robustness:** Well-suited for large-scale problems and high-dimensional parameter spaces. - **Disadvantages:** - **Complexity:** More complex to implement and understand compared to simpler algorithms like SGD. - **Resource Intensive:** Requires more memory to store moment estimates.

Practical Considerations

Choosing Hyperparameters

- **Gradient Descent:** Learning rate (η). - **SGD:** Learning rate (η), batch size. - **Adam:** Learning rate (α), β_1, β_2, ϵ.

Hyperparameters significantly influence the performance of optimization algorithms. Techniques such as grid search, random search, and Bayesian optimization can help in tuning these parameters.

Implementation in Libraries

Modern machine learning libraries like TensorFlow, PyTorch, and Keras provide built-in implementations of these optimization algorithms. For example:

- In **TensorFlow**:

```
optimizer = tf.optimizers.Adam(learning_rate=0.001)
```

- In **PyTorch**:

```
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

- In **Keras**:

```
optimizer = keras.optimizers.Adam(learning_rate=0.001)
```

Real-life Examples

1. **Gradient Descent in Linear Regression:** - Used to find the best-fit line by minimizing the mean squared error between the predicted and actual values.

2. **SGD in Image Classification:** - Commonly used in training deep neural networks for tasks like image recognition, where the dataset is large and using the entire dataset for each update is computationally expensive.

3. **Adam in Natural Language Processing (NLP):** - Frequently used in training transformers and other deep learning models for tasks such as language translation, where adaptive learning rates and robust convergence properties are crucial.

Monitoring and Debugging

- **Track Loss:** Monitor the loss function value over iterations to ensure the algorithm is converging. - **Learning Rate Adjustments:** If the loss is not decreasing, try adjusting the learning rate. - **Visualization:** Plot the loss function and parameter values to understand the optimization path and detect issues like vanishing gradients or exploding gradients.

Practical Tips

- **Initialization:** Proper initialization of parameters can prevent issues like slow convergence or getting stuck in local minima. Techniques such as Xavier or He initialization are commonly used. - **Batch Size:** In SGD, choosing an appropriate batch size is critical. A small batch size introduces noise but can help escape local minima, while a large batch size provides more stable updates. - **Learning Rate Schedules:** Dynamic learning rate adjustments (e.g., reducing the learning rate when the improvement in loss is minimal) can improve convergence.

End of Document