SGD Optimizations: Adaptive Deep Learning Training

Overview

In this project, I used a combination of advanced techniques for deep learning model training optimization. These include adaptive learning rates, data normalization, advanced training techniques, and optimizer tuning, all to a synthetic noisy sine wave dataset. SGD and Adam optimizers are employed in training the model to maximize convergence and minimize loss in training to achieve efficient learning even in the presence of noisy data.

Key Concepts Implemented

1. Data Normalization

Normalization of the data is an extremely critical preprocessing operation in machine learning models. I normalized the dataset based on the formula given below, ensuring features were scaled proportionally:

$$x_{\text{norm}} = \frac{x - \mu}{\sigma}$$

Where μ is the mean and σ is the standard deviation of the data. This normalization was done to prevent the model from becoming biased towards features with high values or other scales.

2. Activation Function: ReLU

ReLU activation function introduces non-linearity into the model. The ReLU function is defined as:

$$\text{ReLU}(x) = \max(0, x)$$

With this activation function, the model has the ability to learn complex relationships between input and output and thus identify more sophisticated patterns in the data.

3. Weight Initialization: Xavier Initialization

In order to prevent exploding or disappearing gradients, I have used Xavier Initialization to initialize the model weights. This will ensure weight scaling appropriately and enable gradients to flow through backpropagation smoothly. The mathematical formula for Xavier initialization is:

$$W \sim \mathcal{U}\left(-\sqrt{\frac{6}{n_{\text{in}} + n_{\text{out}}}}, \sqrt{\frac{6}{n_{\text{in}} + n_{\text{out}}}}\right)$$

Where n(in) and n(out) are the number of input and output units for a given layer.

4. Loss Function: Mean Squared Error (MSE)

For training the model, I used the **Mean Squared Error (MSE)** loss function. The MSE loss function quantifies the difference between the model's predictions y^ and the true values y The formula for MSE is:

$$\text{MSE}(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

This loss function was chosen because it is effective for regression problems like this one, where the goal is to minimize the error between predicted and true values.

5. Optimizers: SGD and Adam

I implemented two different optimizers: **Stochastic Gradient Descent (SGD)** and **Adam**. Both optimizers are used to update the weights of the model during training.

- **SGD** uses the following update rule:

$$W = W - \eta \cdot \nabla_w J(W)$$

Where η is the learning rate, and ∇wJ(W) is the gradient of the loss function with respect to the weights W.

- **Adam**, which adjusts the learning rate dynamically, combines the benefits of **Momentum** and **RMSprop**. The Adam update rule is based on the following formulas:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla_w J(W)$$

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

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

The Adam update rule is then:

$$W = W - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

Where β1 and β2 are parameters controlling the exponential decay rates of the moving averages of the first and second moments, and ϵ(epsilon) is a small constant to prevent division by zero.

6. Learning Rate Scheduling: Cyclic Learning Rate (CyclicLR)

To further optimize the training process, I implemented a **Cyclic Learning Rate (CyclicLR)** scheduler. This scheduler adjusts the learning rate dynamically throughout the training process. The new learning rate is calculated using the following formula:

$$\text{LR}_{\text{new}} = \text{base\_lr} + (\text{max\_lr} - \text{base\_lr}) \cdot \left(1 - \text{cycle\_progress}\right)$$

This cyclical adjustment of the learning rate helped the model avoid local minima, leading to more efficient training and potentially better generalization.

Training Pipeline

1. Data Generation

The first step in the training pipeline was generating the synthetic noisy sine wave dataset. I created this dataset as a simple example for regression tasks and served as input-output pairs for the training process.

2. Model Training

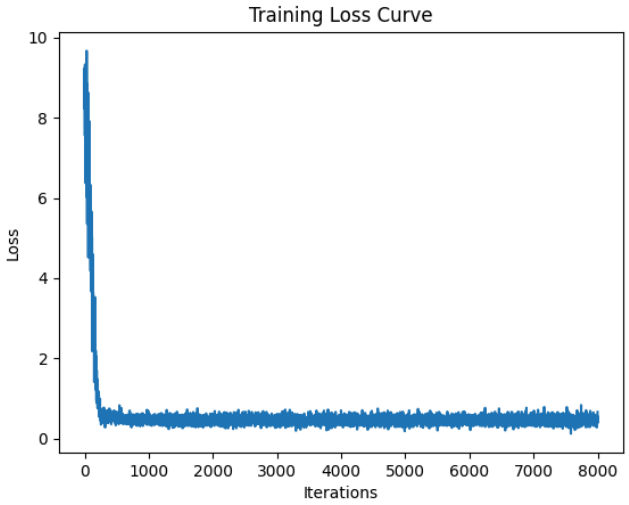
Once the data was generated, I trained the model using the configurations specified in the `config.yaml` file. Key hyperparameters such as learning rate, batch size, optimizer choice, and the number of epochs were defined in the configuration to allow for flexibility during experimentation.

3. Adaptive Learning Rates

I used an adaptive learning rate mechanism through the **Adam optimizer** combined with the **CyclicLR scheduler.** This enabled the model to learn the learning rate adaptively during training, making it converge better and avoid local minima.

4. Evaluation

Loss value and learning rate at each epoch were monitored while training. They were representative of the performance of the model, and I would proceed to modify the training process and outcomes accordingly.



Results

The model's performance after training was good enough to fit noisy sine wave function. Adaptive learning rates and careful initialization helped the model to learn the loss and also converge well.

Conclusion

Through the implementation of this project, I employed a sequence of techniques to optimize the training of deep learning models. Through the use of Xavier initialization, adaptive learning rate, and sophisticated optimizers such as Adam, I was able to enhance the convergence and performance of the model. All these techniques are widely applied in deep learning operations and can be adapted for complex models and big datasets