Optimizing deep learning model – using learning rate

Bhupen Sinha

25-07-2024

TABLE OF CONTENTS

LEARNING RATE IN DEEP LEARNING

Objective

The learning rate is a hyperparameter that determines the step size at each iteration while moving toward a minimum of a loss function.

It balances between making updates to the model weights too large or too small.

A well-chosen learning rate ensures that the model converges efficiently and effectively.

How It Is Used

In training neural networks, the learning rate is used during the optimization process. When using gradient descent or its variants to minimize the loss function, the learning rate determines the step size at each iteration.

The update rule can be expressed as:

new_weight = old_weight - learning_rate * gradient_of_loss_function

where:

- new_weight is the updated weight.
- old_weight is the current weight.
- learning_rate is the learning rate.
- gradient_of_loss_function is the gradient of the loss function with respect to the weight.

Examples in Keras

In Keras, you can set the learning rate when defining the optimizer. Here are some examples:

```
from keras.optimizers import SGD

optimizer = SGD(learning_rate=0.01)
model.compile(optimizer= optimizer,
              loss     = 'categorical_crossentropy',
              metrics  = ['accuracy'])
```

```
from keras.optimizers import Adam

optimizer = Adam(learning_rate=0.001)
model.compile(optimizer= optimizer,
              loss     = 'categorical_crossentropy',
              metrics  = ['accuracy'])
```

Typical Values for Learning Rates

The appropriate learning rate value can vary depending on the model architecture and the specific problem.

Here are some typical values:

- For **SGD**: 0.01 to 0.1
- For **Adam**: 0.001 to 0.0001
- For **RMSprop**: 0.001 to 0.0001

Key Points on Learning Rates

1. **Too High Learning Rate**: If the learning rate is too high, the model may fail to converge or even diverge, bouncing around without settling to a minimum.
2. **Too Low Learning Rate**: If the learning rate is too low, the training process can be very slow, and it might get stuck in a local minimum.
3. **Learning Rate Scheduling**: Adjusting the learning rate during training can be beneficial. Learning rate schedules or adaptive learning rate methods can help in achieving better performance.
4. **Warm-Up**: Starting with a lower learning rate and gradually increasing it can help stabilize training in the initial phases.
5. **Learning Rate Decay**: Reducing the learning rate as training progresses can help fine-tune the model towards the minimum of the loss function.

TECHNIQUES TO OPTIMIZE LEARNING RATES

1. LEARNING RATE SCHEDULING

Description: Learning rate scheduling involves adjusting the learning rate according to a predefined schedule. Common strategies include step decay and exponential decay.

How to Implement:

- **Step Decay:** Reduce the learning rate by a fixed factor every few epochs.
  - **Illustration:** If you start with a learning rate of 0.1 and reduce it by half every 10 epochs, then:
    - Initial learning rate: 0.1
    - At epoch 10: learning rate = 0.1 * 0.5 = 0.05
    - At epoch 20: learning rate = 0.05 * 0.5 = 0.025

- **Exponential Decay:** Reduce the learning rate exponentially over epochs.
    - **Illustration:** If the initial learning rate is 0.1 and the decay rate is 0.96, then:
        - Learning rate at epoch n = 0.1 * (0.96^n)
        - At epoch 1: learning rate = 0.1 * 0.96 = 0.096
        - At epoch 2: learning rate = 0.1 * 0.96^2 = 0.09216

Pros:

- Effective for many types of problems.
- Simple to implement.

Cons:

- Requires tuning the schedule parameters.
- Not adaptive to the training process.

2. ADAPTIVE LEARNING RATE METHODS

Description: These methods adjust the learning rate based on the performance of the model. Examples include AdaGrad, RMSprop, and Adam.

How to Implement:

- **AdaGrad:** Adapts the learning rate based on the sum of the squares of the past gradients.
    - **Illustration:** For each parameter, the learning rate is scaled inversely proportional to the square root of the sum of past squared gradients.
    - **How It Works:**
        - **Initial Learning Rate:** Start with a global learning rate.
        - **Parameter-Specific Adjustments:** Adjust the learning rate for each parameter based on the sum of the squares of past gradients.
        - **Update Rule:** The learning rate for a parameter is divided by the square root of the sum of the squares of past gradients plus a small constant to prevent division by zero.
    - **Pros:**
        - Effective for sparse data.
        - Automatically adapts learning rates for each parameter.
    - **Cons:**
        - Accumulation of squared gradients can result in excessively small learning rates over time.

- **RMSprop:** Similar to AdaGrad but uses a moving average of the squared gradients to scale the learning rate.
    - **Illustration:** The learning rate is adjusted using the formula: learning rate = initial learning rate / (square root of moving average of past squared gradients).
    - **How It Works:**
        - **Initial Learning Rate:** Start with a global learning rate.
        - **Exponential Decay:** Use an exponentially decaying average of past squared gradients to scale the learning rate.
        - **Update Rule:** The learning rate for each parameter is divided by the root of this moving average.
    - **Pros:**

        - Prevents learning rates from decaying too quickly.
        - Works well in non-stationary settings.
    - **Cons:**
        - Requires tuning of the decay parameter.

- **Adam:** Combines AdaGrad and RMSprop by maintaining estimates of both the mean and variance of the past gradients.
    - **Illustration:** The learning rate is adjusted using the formula: learning rate = initial learning rate * (sqrt(1 - beta2^t) / (1 - beta1^t)), where beta1 and beta2 are decay rates for the moment estimates, and t is the current time step.

How It Works:

- **First Moment (Mean):** Calculate the exponentially decaying average of past gradients.
- **Second Moment (Variance):** Calculate the exponentially decaying average of past squared gradients.
- **Bias Correction:** Apply bias correction to both moments to counteract their initial bias towards zero.
- **Update Rule:** Use these moments to adjust the learning rate for each parameter.

Pros:

- Combines the advantages of both AdaGrad and RMSProp.
- Generally performs well in practice.

Cons:

- Requires tuning of additional hyperparameters (e.g., decay rates for the first and second moments).

3. LEARNING RATE WARM-UP

Description: Learning rate warm-up involves starting with a small learning rate and gradually increasing it to the desired value over a few epochs.

How to Implement:

- **Warm-up:** Start with a small learning rate, such as 0.0001, and increase it linearly over the first few epochs to the target learning rate.
  - **Illustration:** If you want to reach a learning rate of 0.01 over 5 epochs:
    - Learning rate at epoch n = (target learning rate / number of warm-up epochs) * epoch number
    - At epoch 1: learning rate = (0.01 / 5) * 1 = 0.002
    - At epoch 5: learning rate = (0.01 / 5) * 5 = 0.01

Pros:

- Stabilizes training initially.
- Prevents large updates early on.

Cons:

- Adds an additional hyperparameter (warm-up period).
- Slightly longer initial training phase.

4. REDUCELRONPLATEAU

Description: This technique reduces the learning rate when a performance metric has stopped improving.

The ReduceLROnPlateau callback reduces the learning rate when a specified metric (e.g., validation loss) has not improved for a certain number of epochs.

The learning rate is reduced by a factor specified by the user, which can help the model converge more effectively by making smaller updates when training stagnates.

How to Implement:

- **Plateau Monitoring:** Monitor validation loss, and if it doesn't improve for a specified number of epochs (patience), reduce the learning rate by a factor.
  - **Illustration:** If the validation loss doesn't improve for 5 epochs, reduce the learning rate by 20% (factor of 0.8). If starting at 0.01:
    - After no improvement: learning rate = 0.01 * 0.8 = 0.008

Pros:

- Adaptive to the training process.
- Helps avoid overshooting the minimum of the loss function.

Cons:

- May lead to a very small learning rate if the metric plateaus frequently.
- Requires careful monitoring of the metric.

HOW REDUCELRONPLATEAU AND EARLYSTOPPING COMPLEMENT EACH OTHER:

1. **EarlyStopping:**
   - Monitors a specified metric (e.g., validation loss) to stop training when no improvement is seen for a certain number of epochs (patience).
   - Helps to prevent overfitting by halting training when the model performance stops improving.
2. **ReduceLROnPlateau:**
   - Reduces the learning rate when a specified metric has not improved for a certain number of epochs.
   - Helps to refine training by allowing the model to make smaller updates when progress stalls.

Combined Use Case:

- **EarlyStopping** will terminate training when it detects that further training will not improve the model, which can be due to overfitting or reaching the optimal performance.
- **ReduceLROnPlateau** will adjust the learning rate if it detects that the monitored metric has stopped improving, potentially allowing the model to continue training more effectively at a smaller learning rate.

Potential Issue

1. **Learning Rate Adjustment:**
    - **ReduceLROnPlateau** can continuously reduce the learning rate, which might lead to very slow convergence. This could cause the model to keep training for longer periods without significant improvement, potentially interfering with EarlyStopping.
2. **Early Stopping Not Triggering:**
    - If the learning rate is reduced too aggressively, the model might still be able to make small improvements in the loss, but not significant enough to trigger early stopping. This could delay the stopping of training, leading to unnecessarily long training times.

MITIGATION STRATEGIES

1. **Tune Patience Parameters:**
    - Set appropriate values for patience in both ReduceLROnPlateau and EarlyStopping. For example, ensure that the patience of EarlyStopping is large enough to account for the learning rate reduction period. Conversely, set the patience of ReduceLROnPlateau to ensure that it doesn't keep reducing the learning rate indefinitely.
2. **Monitor Different Metrics:**
    - Sometimes monitoring a different metric for EarlyStopping than for ReduceLROnPlateau can help. For example, monitor val_loss for ReduceLROnPlateau but use a different metric like val_accuracy for EarlyStopping.

5. LEARNING RATE FINDER

Description: This method experimentally determines the optimal learning rate by gradually increasing it and observing the loss.

How to Implement:

- **Rate Finding:** Start with a very small learning rate (e.g., 0.0000001) and exponentially increase it every batch until it reaches a large value (e.g., 1). Plot the learning rate vs. loss.
    - **Illustration:** The learning rate starts at 0.0000001 and doubles every 100 batches:
        - Plot the loss as a function of the learning rate. The optimal learning rate is found where the loss decreases the fastest and then starts to increase.

Pros:

- Helps to quickly find an appropriate learning rate.
- Reduces guesswork in selecting the learning rate.

Cons:

- Requires additional initial training runs.
- May not always find the best learning rate if the increase is too rapid.

Each of these techniques can help optimize the learning rate for training a neural network, potentially leading to better performance and faster convergence. The choice of technique may depend on the specific problem, model architecture, and training data. Experimentation and validation are key to finding the most effective approach.

BEST PRACTICES FOR LEARNING RATE IN DEEP LEARNING

1. **Start with Standard Learning Rates:**
   - Use common initial values for different optimizers (e.g., 0.001 for Adam, 0.01 for SGD).
   - These values are often good starting points based on empirical evidence.
2. **Learning Rate Scheduling:**
   - **Step Decay:** Reduce the learning rate by a factor every few epochs (e.g., divide by 10 every 30 epochs).
   - **Exponential Decay:** Gradually decrease the learning rate using an exponential function.
   - **Performance-Based Reduction:** Use callbacks like ReduceLROnPlateau to reduce the learning rate when performance plateaus.
3. **Warm-Up Strategies:**
   - **Learning Rate Warm-Up:** Start with a small learning rate and gradually increase it over a few epochs. This helps in stabilizing training in the initial phase.
   - **Illustration:** Linearly increase the learning rate from a small value to the initial learning rate over the first few epochs.
4. **Adaptive Learning Rate Methods:**
   - **Adam, RMSProp, AdaGrad:** Use optimizers that automatically adjust the learning rate for each parameter based on past gradient information.
   - **Illustration:** Adam uses exponentially decaying averages of past gradients and squared gradients to compute adaptive learning rates.
5. **Monitor and Adjust:**
   - **Validation Performance:** Regularly monitor validation loss/accuracy and adjust learning rates accordingly.
   - **Early Stopping:** Use early stopping to halt training if validation performance stops improving, preventing overfitting.
6. **Learning Rate Range Test:**
   - **Learning Rate Finder:** Perform a range test to identify the optimal learning rate. Start with a very small learning rate and exponentially increase it while monitoring loss.
   - **Illustration:** Plot loss versus learning rate to find the learning rate range where the model improves the most.

7. **Batch Size Considerations:**
   - **Learning Rate and Batch Size:** Larger batch sizes can typically support higher learning rates, whereas smaller batch sizes may require lower learning rates.
   - **Illustration:** If doubling the batch size, consider increasing the learning rate proportionally.
8. **Regularization:**
   - **Weight Decay:** Combine learning rate schedules with weight decay to regularize model parameters and prevent overfitting.
   - **Illustration:** Use weight decay with optimizers like AdamW that decouple weight decay from the learning rate.
9. **Experimentation and Tuning:**
   - **Hyperparameter Tuning:** Systematically experiment with different learning rates and schedules to find the best combination for your specific problem.
   - **Illustration:** Use grid search or random search over a range of learning rates and decay schedules.

KEY POINTS TO REMEMBER (DESIGNING LEARNING RATES)

- **Experimentation:** There is no one-size-fits-all learning rate. Experiment with different values and schedules.
- **Balance:** Finding the right balance between learning rate, batch size, and regularization is crucial.
- **Monitoring:** Continuously monitor the model's performance on validation data and adjust learning rates as necessary.
- **Automation:** Utilize callbacks in Keras to automate the adjustment of learning rates during training.

DOCUMENTED LEARNING RATE SETTINGS IN KNOWN DEEP LEARNING ARCHITECTURES

Different deep learning architectures often have specific learning rate settings that have been found to work well based on empirical evidence. Here are the documented learning rate settings for some well-known architectures:

1. AlexNet

- **Initial Learning Rate:** 0.01
- **Learning Rate Schedule:**
    - Reduce by a factor of 10 manually when the validation accuracy plateaus.
    - In some implementations, the learning rate is reduced every 30 epochs.
- **Optimizer:** Stochastic Gradient Descent (SGD) with momentum.
- **Momentum:** 0.9

2. VGGNet (VGG16 and VGG19)

- **Initial Learning Rate:** 0.01
- **Learning Rate Schedule:**
    - Reduce by a factor of 10 manually when the validation accuracy plateaus.
    - In some implementations, the learning rate is reduced every 30 epochs.
- **Optimizer:** Stochastic Gradient Descent (SGD) with momentum.
- **Momentum:** 0.9

3. ResNet (ResNet50, ResNet101, ResNet152)

- **Initial Learning Rate:** 0.1
- **Learning Rate Schedule:**
    - Reduce by a factor of 10 at specific epochs (e.g., 30, 60, 90 epochs in a 100-epoch training).
- **Optimizer:** Stochastic Gradient Descent (SGD) with momentum.
- **Momentum:** 0.9

4. Inception (Inception-v3)

- **Initial Learning Rate:** 0.045
- **Learning Rate Schedule:**
    - Exponential decay, decaying every 2 epochs with a decay rate of 0.94.

- **Optimizer:** RMSProp.
- **Decay:** 0.9
- **Momentum:** 0.9

5. DenseNet (DenseNet121, DenseNet169, DenseNet201)

- **Initial Learning Rate:** 0.1
- **Learning Rate Schedule:**
    - Reduce by a factor of 10 at specific epochs (e.g., 30, 60, 90 epochs in a 100-epoch training).
- **Optimizer:** Stochastic Gradient Descent (SGD) with momentum.
- **Momentum:** 0.9

6. MobileNet

- **Initial Learning Rate:** 0.045
- **Learning Rate Schedule:**
    - Exponential decay with a decay factor of 0.98 every epoch.
- **Optimizer:** RMSProp.
- **Decay:** 0.9
- **Momentum:** 0.9

7. Transformer Models

- **Initial Learning Rate:** 0.001 (often referred to as the base learning rate).
- **Learning Rate Schedule:**
    - Warm-up for a number of steps (e.g., 4000 steps), followed by an inverse square root decay.
    - After warm-up, the learning rate decreases proportionally to the inverse square root of the step number.
- **Optimizer:** Adam.
- **Adam Parameters:**
    - Beta1: 0.9
    - Beta2: 0.98
    - Epsilon: 1e-9

8. BERT (Bidirectional Encoder Representations from Transformers)

Optimizing deep learning model – using learning rate

- **Initial Learning Rate:** 5e-5
- **Learning Rate Schedule:**
  - Warm-up for a number of steps (e.g., 10,000 steps), followed by linear decay.
  - After warm-up, the learning rate decays linearly to zero.
- **Optimizer:** Adam.
- **Adam Parameters:**
  - Beta1: 0.9
  - Beta2: 0.999
  - Epsilon: 1e-6

USEFUL LINKS AND REFERENCES

-
-

INDEX

No index entries found.