SESSION 10 - Advanced Concepts in Training & Learning Rates

Table of Contents
Table of Figures
Table of Tables
Table of Equations
Table of Code
LEARNING RATE
Weight Update
Constant Learning Rate Algorithms
Adaptive Learning Rate Algorithms
Momentum Gradient Descent
Nesterov Accelerated Momentum
Adam
AdaGrad
RMSProp
AdaDelta
Nadam
Adamax
Adam Variants
Comparing Different Approach
LEARNING RATE
DECAY OPTIMIZER
CLOSE
EXCHANGE
DOCUMENTARY

LEARNING RATE

Recognizing very complicated functions, which contains thousands or millions of parameters, it is very difficult and tedious to find optimal parameters.

Training such complicated functions with millions of parameters, it may take days or even weeks to find optimal parameters. For such cases we need to use some more advanced optimizers with adaptive learning rates.

Take a look at following image, it shows how we reach to global minima for convex function.

Fig 1 - Gradient Descent Algorithm

Now consider a complicated function, which looks like this.

Fig 2 - Local minima in Gradient Descent

Weight Update

$$w' = w - \eta * \left(\frac{\partial L}{\partial w}\right)$$

Where w is a weight, w' is a new weight, η is a learning rate, $\frac{\partial L}{\partial w}$ is a gradient of loss w.r.t. weight parameter. Before it is time to go,

Let's quickly review some commonly used terms in training and learning rate.

Constant Learning Rate Algorithms

Gradient descent is a first-order optimization algorithm. To find a local minimum of a function using gradient descent, we take steps proportional to the negative of the gradient (or approximate gradient) of the function at the current point.

Batch Gradient Descent is a variant of gradient descent which calculates the gradient of the cost function with respect to the parameters for the entire training dataset. Since we need to calculate the gradients for the whole dataset to perform just one update, batch gradient descent can be very slow and hard to control for datasets which don't fit in memory.

Stochastic Gradient Descent (SGD) updates the parameters for each training example $x^{(i)}$ and label $y^{(i)}$.

Mini-batch Gradient Descent finally takes the best of both worlds and performs an update for every mini-batch of n training examples.

Let's quickly review some commonly used terms in training and learning rate.

Adaptive Learning Rate Algorithms

Adaptive gradient descent algorithm is an extension of Stochastic Gradient Descent. Instead of using a constant learning rate, it adapts the learning rate for each parameter at each time step.

Adaptive Gradient (AdaGrad) is a gradient-based optimization algorithm that adapts the learning rate, perform larger updates for infrequent and smaller updates for frequent parameters. It is well-suited for dealing with sparse data. Adagrad uses the same learning rate for all parameters for every time step.

Adaptive Delta (AdaDelta) is an extension of Adagrad that seeks to reduce its aggressive, monotonically decreasing learning rate. Instead of accumulating all past squared gradients, Adadelta restricts the window of accumulated past gradients to some fixed size w.

Adaptive Moment Estimation (Adam) is another method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients like Adadelta or RMSprop, Adam also keeps an exponentially decaying average of past gradients, similar to momentum.

Adam

Momentum Gradient Descent

SGD has a hard time navigating through ravines, a common problem for local optima. In such cases, SGD oscillates across the slopes of the ravine while only making hesitant progress along the bottom towards the local optimum.

With such a ravine, the optimal path to the local optimum is a small step in the direction of the local optimum, then a large step down the slope, then a small step in the direction of the local optimum.

Momentum helps accelerate SGD in the relevant direction and dampens oscillations as shown in the following image.

Fig 3 - Momentum Gradient Descent

Let's quickly review some commonly used terms in training and learning rate.

Nesterov Accelerated Momentum

Nesterov accelerated momentum (NAG) is a way to give our momentum term this kind of prescience. We know that we will use our momentum γv_{t-1} to move the parameters θ. Computing $\theta - \gamma v_{t-1}$ thus gives us an approximation of the next position of the parameters (the gradient is missing for the full update), a rough estimate that our momentum is about to carry us to.

Now we can effectively look ahead by setting the momentum term γ to small value and calculate the gradient not with respect to our current parameters θ, but with respect to the approximate future position of our parameters:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1})$$

$$\theta = \theta - v_t$$

Momentum term γ is commonly set to 0.9.

We now can adapt our update to the slope of our error function and speed up SGD by a factor of $\frac{1}{1-\gamma}$ in the direction of the local optimum.

Fig 4 - Nesterov Accelerated Momentum

Let's quickly review some commonly used terms in training and learning rate.

Adam

Adaptive Moment Estimation (Adam) is another method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients like Adadelta and RMSprop, Adam also keeps an exponentially decaying average of past gradients, similar to momentum.

Adam also keeps an exponentially decaying average of past squared gradients v_t and a decaying average of past gradients m_t:

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

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

m_t and v_t are estimates of the first moment (the mean) and the second moment (the uncertainty) of the gradients respectively, hence the name of the gradient descent variant.

As m_t and v_t are initialized as vectors of zeroes, the authors of Adam observe that they are biased towards zero, especially during the initial time steps, and especially when the decay rates are small (i.e. β_1 and β_2 close to 1).

They counteract these biases by computing bias-corrected first and second moment estimates:

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

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

They then use these to update the parameters just as we have seen in Adadelta and RMSprop, which yields the Adam update rule:

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

Let's quickly review some commonly used terms in training and learning rate.

AdaGrad

Adaptive Gradient (AdaGrad) is a gradient-based optimization algorithm that adapts the learning rate, perform larger updates for infrequent and smaller updates for frequent parameters. It is well-suited for dealing with sparse data. Adagrad uses the same learning rate for all parameters for every time step.

Let's quickly review some commonly used terms in training and learning rate.

RMSProp

Root Mean Square Propagation (RMSProp) is an unpublished, adaptive learning rate method proposed in Geoff Hinton's lecture notes. RMSProp and Adadelta were developed independently around the same time stemming from the need to resolve Adagrad's radically diminishing learning rates.

RMSProp divides the learning rate by an exponentially decaying average of squared gradients. Hinton suggests γ to be set to 0.9, while a good default value for the learning rate η is 0.001.

Let's quickly review some commonly used terms in training and learning rate.

AdaDelta

Adaptive Delta (AdaDelta) is an extension of Adagrad that seeks to reduce its aggressive, monotonically decreasing learning rate. Instead of accumulating all past squared gradients, Adadelta restricts the window of accumulated past gradients to some fixed size w.

Let's quickly review some commonly used terms in training and learning rate.

Nadam

Nesterov-accelerated Adaptive Moment Estimation (Nadam) combines Adam and Nesterov accelerated momentum. Since Adam incorporates momentum, it is trivial to add Nesterov momentum.

Let's quickly review some commonly used terms in training and learning rate.

Adamax

Adamax is a variant of Adam. Adam computes a weight update for each parameter w_t by dividing the decaying average of the past and past squared gradients v_t by $\sqrt{v_t}$. This leads to a weight update of the form $\frac{\eta}{\sqrt{v_t}} m_t$. Adamax generalizes the update to the L^p norm. Instead of using v_t for $p = 2$, we can use v_t for $p = \infty$. With $p \to \infty$, v_t converges to the following more stable value. For ease of writing, we denote this as u_t:

$$u_t = \beta_2^\infty v_{t-1} + (1 - \beta_2^\infty) |g_t|^\infty$$

$$= \max(\beta_2 \cdot v_{t-1}, |g_t|)$$

Let's quickly review some commonly used terms in training and learning rate.

Adam Variants

Adam with Decoupled Weight Decay (AdamW) proposes a simple fix to improve Adam's performance by decoupling the weight decay from the gradient update. Adam with Decoupled Weight Decay and Adam with Warm Restarts (AdamWR) proposes a warm restart variant for Adam. Adam with Warm Restarts and Adam with Decoupled Weight Decay (AdamWW) combines AdamW and AdamWR.

Let's quickly review some commonly used terms in training and learning rate.

Comparing Different Approach

Let's review the ways they are moving towards the global minimum.

Figure 1 - Gradient Descent Algorithms for Optimization Problem

Let's review the ways they are moving towards the saddle point.

Figure 2 - Gradient Descent Algorithms for Saddle Point Problem

LEARNING RATE

Fig 1 - Learning Rate

Let's take some example, we will use Cost Function $J(\theta)$.

Fig 2 - Cost Function

Now it looks like this.

Fig 3 - Cost Function

Now it looks like this.

Fig 4 - Cost Function

Now it looks like this.

Fig 5 - Cost Function

Now it looks like this.

Fig 6 - Cost Function

Now it looks like this.

Fig 7 - Cost Function

Now it looks like this.

Fig 8 - Cost Function

Now it looks like this.

Fig