An overview of gradient descent optimization algorithms

**Abstract**

In this paper, we look at different variants of gradient descent, summarize challenges, introduce the most common optimization algorithms, review architectures in a parallel and distributed setting, and investigate additional strategies for optimizing gradient descent.

1 Introduction

Gradient descent is a way to minimize an objective function J(θ) parameterized by a model’s parameters θ ∈ Rd by updating the parameters in the opposite direction of the gradient of the objective function ∇θJ(θ) w.r.t. to the parameters. The learning rate η determines the size of the steps we take to reach a (local) minimum.

2 Gradient descent variants

2.1 Batch gradient descent

Vanilla gradient descent, aka batch gradient descent, computes the gradient of the cost function w.r.t. to the parameters θ for the entire training dataset:

θ = θ − η · ∇θJ(θ) (1)

for i in range(nb\_epochs):

params\_grad = evaluate\_gradient(loss\_function, data, params)

params = params - learning\_rate \* params\_grad

2.2 Stochastic gradient descent

Stochastic gradient descent (SGD) in contrast performs a parameter update for *each* training example x(i) and label y(i):

θ = θ − η · ∇θJ(θ; x(i); y(i)) (2)

for i in range(nb\_epochs):

np.random.shuffle(data)

for example in data:

params\_grad = evaluate\_gradient(loss\_function , example , params)

params = params - learning\_rate \* params\_grad

2.3 Mini-batch gradient descent

Mini-batch gradient descent finally takes the best of both worlds and performs an update for every mini-batch of n training examples:

θ = θ − η · ∇θJ(θ; x(i:i+n); y(i:i+n)) (3)

for i in range(nb\_epochs):

np.random.shuffle(data)

for batch in get\_batches(data, batch\_size=50):

params\_grad = evaluate\_gradient(loss\_function, batch, params)

params = params - learning\_rate \* params\_grad

3 Challenges

* Choosing a proper learning rate can be difficult. A learning rate that is too small leads to painfully Another key challenge of minimizing highly non-convex error functions common for neural networks is avoiding getting trapped in their numerous suboptimal local minima.
* slow convergence, while a learning rate that is too large can hinder convergence and cause the loss function to fluctuate around the minimum or even to diverge.

4 Gradient descent optimization algorithms

4.1 Momentum

Momentum is a method that helps accelerate SGD in the relevant direction. It does this by adding a fraction γ of the update vector of the past time step to the current update vector.

vt = γvt−1 + η∇θJ(θ)

θ = θ − vt

The momentum term γ is usually set to 0.9 or a similar value.

The momentum term increases for dimensions whose gradients point in the same directions and reduces updates for dimensions whose gradients change directions. As a result, we gain faster convergence and reduced oscillation.

4.2 Nesterov accelerated gradient

vt = γ vt−1 + η∇θJ(θ − γvt−1)

θ = θ − vt

4.3 Adagrad

Adagrad is an algorithm for gradient-based optimization that does just this: It adapts the learning rate to the parameters, performing larger updates for infrequent and smaller updates for frequent parameters.

4.4 Adadelta

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.

4.5 RMSprop

RMSprop as well 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.

4.6 Adam

Adaptive Moment Estimation (Adam) [10] is another method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients vt like Adadelta and RMSprop, Adam also keeps an exponentially decaying average of past gradients mt, similar to momentum:

mt = β1mt−1 + (1 − β1)gt

vt = β2vt−1 + (1 − β2)gt2

mt and vt are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients respectively, hence the name of the method.

4.7 AdaMax

The vt factor in the Adam update rule scales the gradient inversely proportionally to the l2 norm of the past gradients (via the vt−1 term) and current gradient |gt|2:

vt = β2vt−1 + (1 − β2)|gt|2 (22)

We can generalize this update to the lp norm. Note that Kingma and Ba also parameterize β2 as β2p:

vt = β2pvt−1 + (1 − β2p)|gt|p (23)

4.8 Nadam

Nadam (Nesterov-accelerated Adaptive Moment Estimation) thus combines Adam and NAG. In order to incorporate NAG into Adam, we need to modify its momentum term mt.

4.9 Visualization of algorithms

4.10 Which optimizer to use?

5 Parallelizing and distributing SGD

5.1 Hogwild!

Processors are allowed to access shared memory without locking the parameters. This only works if the input data is sparse, as each update will only modify a fraction of all parameters.

5.2 Downpour SGD

5.3 Delay-tolerant Algorithms for SGD

5.4 TensorFlow

5.5 Elastic Averaging SGD

6 Additional strategies for optimizing SGD

6.1 Shuffling and Curriculum Learning

Generally, we want to avoid providing the training examples in a meaningful order to our model as this may bias the optimization algorithm. Consequently, it is often a good idea to shuffle the training data after every epoch.

On the other hand, for some cases where we aim to solve progressively harder problems, supplying the training examples in a meaningful order may actually lead to improved performance and better convergence. The method for establishing this meaningful order is called Curriculum Learning.

6.2 Batch normalization

Batch normalization reestablishes these normalizations for every mini-batch and changes are back- propagated through the operation as well. By making normalization part of the model architecture, we are able to use higher learning rates and pay less attention to the initialization parameters. Batch normalization additionally acts as a regularizer, reducing (and sometimes even eliminating) the need for Dropout.

6.3 Early stopping

You should thus always monitor error on a validation set during training and stop (with some patience) if your validation error does not improve enough.

6.4 Gradient noise

7 Conclusion

In this article, we have initially looked at the three variants of gradient descent, among which mini- batch gradient descent is the most popular. We have then investigated algorithms that are most commonly used for optimizing SGD: Momentum, Nesterov accelerated gradient, Adagrad, Adadelta, RMSprop, Adam, AdaMax, Nadam, as well as different algorithms to optimize asynchronous SGD. Finally, we’ve considered other strategies to improve SGD such as shuffling and curriculum learning, batch normalization, and early stopping.