Gradient descent

Fuhao Zou

Wuhan National Laboratory for Optoelectronics
Huazhong University of Science & Technology

2019.4

Table of contents

Table of Contents

Why Gradient Descent?

Basic idea:

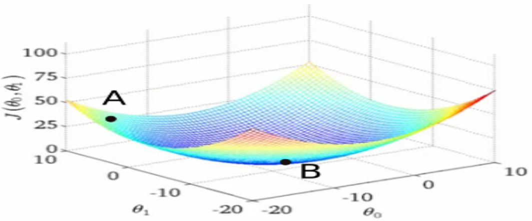
The process of traning a net is ,as a matter of fact,optimizing billions of parameters of the net.And we need to find a proper way to minimize a convex, continuous and differentiable loss function $\mathbf{l}(\mathbf{w})$

- The value of this loss function gives us a measure how far from perfect is the performance of our network on a given dataset.
- When we randomly initialyze weights in the begining, the value of this loss function is uaually huge.
- We need to optimize the loss function to get a better the performance of our network.

Gradient Descent

In Machine Learning,Gradient Descent is an optimization algorithm to minimize the loss function, and it can be described as:

```
while True:
weights_grad = evaluate_gradient(loss_fun,data,weigths)
weights+=-stepsize*weights_grad
```



Gradient Descent

Gradient Descent

Assume $\mathbf{f}(\mathbf{x})$ is a continuously differentiable function.Given ϵ with a small enough absolute value,according to the Taylor's expansion formula,we get the following approximation:

$$\mathbf{f}(\mathbf{x}+\epsilon) \approx \mathbf{f}(\mathbf{x}) + \epsilon\mathbf{f}'(\mathbf{x})$$

The gradient of a one-dimensional function is a scalar, also known as a derivative.Next, find a constant $\eta > 0$,to make $|\eta\mathbf{f}'(\mathbf{x})|$ sufficiently small so that we can replace ϵ with $-\eta\mathbf{f}(\mathbf{x})$ and get:

$$\mathbf{f}\Big(\mathbf{x} - \eta\mathbf{f}'(\mathbf{x})\Big) \approx \mathbf{f}(\mathbf{x}) \ - \eta\mathbf{f}'(\mathbf{x})^2$$

If the derivative $\mathbf{f}'(\mathbf{x}) \neq 0$,then $\eta\mathbf{f}'(\mathbf{x})^2 > 0$,so

$$f\Big(\mathbf{x} - \eta\mathbf{f}'(\mathbf{x})\Big) \leq \mathbf{f}(\mathbf{x})$$

This means,if we use

$$\mathbf{x} \leftarrow \mathbf{x} - \eta\mathbf{f}'(\mathbf{x})$$

to iterate $\mathbf{x}$,the value of function $\mathbf{f}(\mathbf{x})$ might decline.

Gradient Descent

The positive η in the the above gradient descent algorithm is usually called the learning rate. This is a hyper-parameter and needs to be set manually. If we use a learning rate that is too small it will cause $\mathbf{x}$ to update at a very slow speed, requiring more iterations to get a better solution. Now assume $f(x) = x^2, \eta = 0.05, iterations = 10$

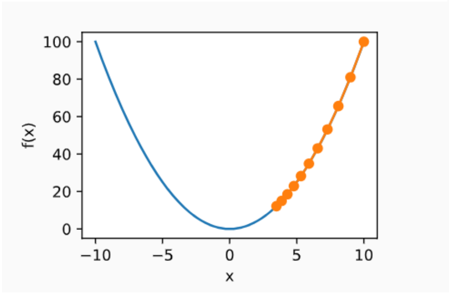


Fig: η=0.05

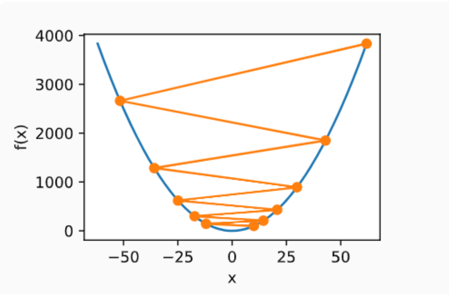


Fig: η=1.1

Table of Contents

Batch Gradient Descent

Basic idea:

Parameters are updated after computing the gradient of error with respect to the entire training set.

Example

test-BGD.py,try to set the different learning rate to see the output:

`https://github.com/IEC-lab/MachineLearning2019/blob/master/GD/test-BGD.py`

Summary for BGD

Upsides:

- Fewer updates to the model means this variant of gradient descent is more computationally efficient than stochastic gradient descent.
- The decreased update frequency results in a more stable error gradient and may result in a more stable convergence on some problems.
- The separation of the calculation of prediction errors and the model update lends the algorithm to parallel processing based implementations.

Summary for BGD

Downsides:

- The more stable error gradient may result in premature convergence of the model to a less optimal set of parameters.
- The updates at the end of the training epoch require the additional complexity of accumulating prediction errors across all training examples.
- Commonly, batch gradient descent is implemented in such a way that it requires the entire training dataset in memory and available to the algorithm.
- Model updates, and in turn training speed, may become very slow for large datasets.

Table of Contents

Stochastic Gradient Descent

Quiz

In Machine Learning,the training set is enormous and no simple formulas exist, evaluating the sums of gradients becomes very expensive, because evaluating the gradient requires evaluating all the summand functions' gradients. How to minimize the computational cost at every iteration?

Stochastic Gradient Descent

Solution

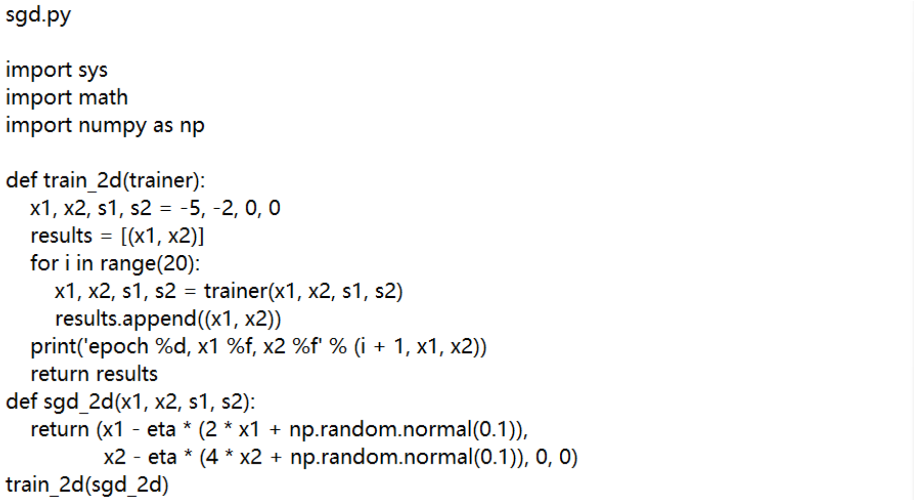
Samples a subset of the training set at every step instead of the whole set.Parameters are updated after computing the gradient of error with respect to a single training example.

In pseudocode, stochastic gradient descent can be presented as follows:

```
• Choose an initial vector of parameters $x$ and learning rate $\eta$
• Repeat until an approximate minimum is obtained:
    • Randomly shuffle examples in the training set.
    • For $i$=1,2,...,n,do:
        • $x = x - \eta f_i'(x)$
```

Example

We can add random noise with a mean of 0 to the gradient to simulate a SGD:



```
sgd.py

import sys
import math
import numpy as np

def train_2d(trainer):
    x1, x2, s1, s2 = -5, -2, 0, 0
    results = [(x1, x2)]
    for i in range(20):
        x1, x2, s1, s2 = trainer(x1, x2, s1, s2)
        results.append((x1, x2))
    print('epoch %d, x1 %f, x2 %f' % (i + 1, x1, x2))
    return results
def sgd_2d(x1, x2, s1, s2):
    return (x1 - eta * (2 * x1 + np.random.normal(0.1)),
              x2 - eta * (4 * x2 + np.random.normal(0.1)), 0, 0)
train_2d(sgd_2d)
```

Example

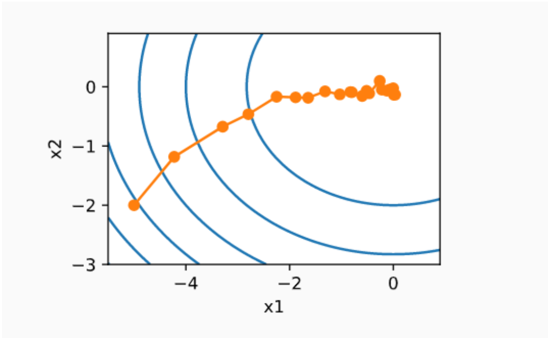


Fig: epoch=20

As we can see,the iterative trajectory of the independent variable in the SGD is more tortuous than in the gradient descent.This is due to the noise added in the experiment, which reduced the accuracy of the simulated stochastic gradient. In practice, such noise usually comes from individual examples in the training data set.

Example

test-SGD.py:

`https://github.com/IEC-lab/MachineLearning2019/blob/master/GD/test-SGD.py`

Summary for SGD

Upsides:

- The frequent updates immediately give an insight into the performance of the model and the rate of improvement.
- This variant of gradient descent may be the simplest to understand and implement, especially for beginners.
- The increased model update frequency can result in faster learning on some problems.
- The noisy update process can allow the model to avoid local minima (e.g. premature convergence).

Summary for SGD

Downsides:

- Updating the model so frequently is more computationally expensive than other configurations of gradient descent, taking significantly longer to train models on large datasets.
- The frequent updates can result in a noisy gradient signal, which may cause the model parameters and in turn the model error to jump around (have a higher variance over training epochs).
- The noisy learning process down the error gradient can also make it hard for the algorithm to settle on an error minimum for the model.

Table of Contents

Mini-batch Stochastic Gradient Descent

Basic idea:

Mini-batch gradient descent seeks to find a balance between the robustness of stochastic gradient descent and the efficiency of batch gradient descent. Parameters are updated after computing the gradient of error with respect to a subset of the training set

Mini-batch Stochastic Gradient Descent

In pseudocode, Mini stochastic gradient descent can be presented as follows:

```
Let theta = model parameters and max_iters = number of epochs.
for itr = 1, 2, 3,..., max_iters:
    for mini_batch(X_mini, y_mini):
      • Forward Pass on the batch X_mini:
          • Make predictions on the mini-batch
          • Compute error in predictions ($J(theta)$) with the current values of the
            parameters
      • Backward Pass:
          • Compute gradient($theta$) = partial derivative of $J(theta)$ w.r.t. $theta$
      • Update parameters:
          • $theta = theta \quad learning\_rate * gradient(theta)$
```

Comparison with BGD&SGD

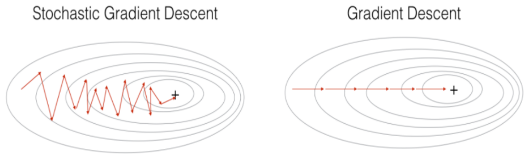


Fig: SGD vs BGD

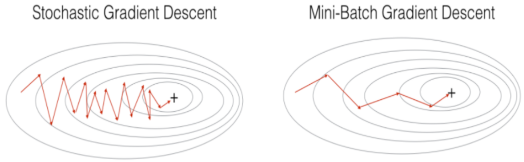


Fig: SGD vs MSGD

Comparison with BGD&SGD

In summary, the difference between gradient descent, mini-batch gradient descent, and stochastic gradient descent is the number of examples you use to perform one update step. With a well tuned mini-batch size, it outperforms gradient descent or stochastic gradient descent.

Batch size

Mini-batch sizes, commonly called "batch sizes" for brevity, are often tuned to an aspect of the computational architecture on which the implementation is being executed. Such as a power of two that fits the memory requirements of the GPU or CPU hardware like 32, 64, 128, 256, and so on.

Batch size is a slider on the learning process.

- Small values give a learning process that converges quickly at the cost of noise in the training process.
- Large values give a learning process that converges slowly with accurate estimates of the error gradient.

Configure batch size

Tip1:

- A good default for batch size might be 32. -Revisiting Small Batch Training for Deep Neural Networks, 2018.

Tip2:

- It is a good idea to review learning curves of model validation error against training time with different batch sizes when tuning the batch size.

Tip3:

- Tune batch size and learning rate after tuning all other hyperparameters.

Example

test-MSGD.py:

`https://github.com/IEC-lab/MachineLearning2019/blob/master/GD/test-MSGD.py`

Summary for MSGD

Upsides:

- The model update frequency is higher than batch gradient descent which allows for a more robust convergence, avoiding local minima.
- The batched updates provide a computationally more efficient process than stochastic gradient descent.
- The batching allows both the efficiency of not having all training data in memory and algorithm implementations.

Summary for MSGD

Downsides:

- Mini-batch requires the configuration of an additional mini-batch size hyperparameter for the learning algorithm.
- Error information must be accumulated across mini-batches of training examples like batch gradient descent.

The end!