OPTIMISERS - DEEP LEARNING

09 July 2023 11:30

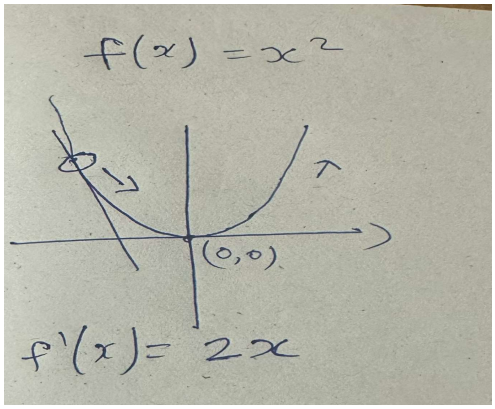
- Deep learning is a subfield of ml, used to perform complex tasks like speech recognition, text classification, etc.
- The deep learning model consists of- activation function, input, output, loss function, etc.
- An **optimizer** is a function that adjusts the attributes of the neural network, which are weights, etc.
- An **optimization algorithm** is the process of improving a model's accuracy and speed and reducing the overall loss - finds the value of the weights to minimize the error while mapping from input to output.
- We can use different optimizers to change the weights in a model.
- **Types of optimizers -**
  - Gradient Descent
  - Stochastic Gradient Descent
  - Stochastic Gradient descent with momentum
  - Mini-Batch Gradient Descent
  - Adagrad
  - RMSProp
  - AdaDelta

Important Deep Learning Terms

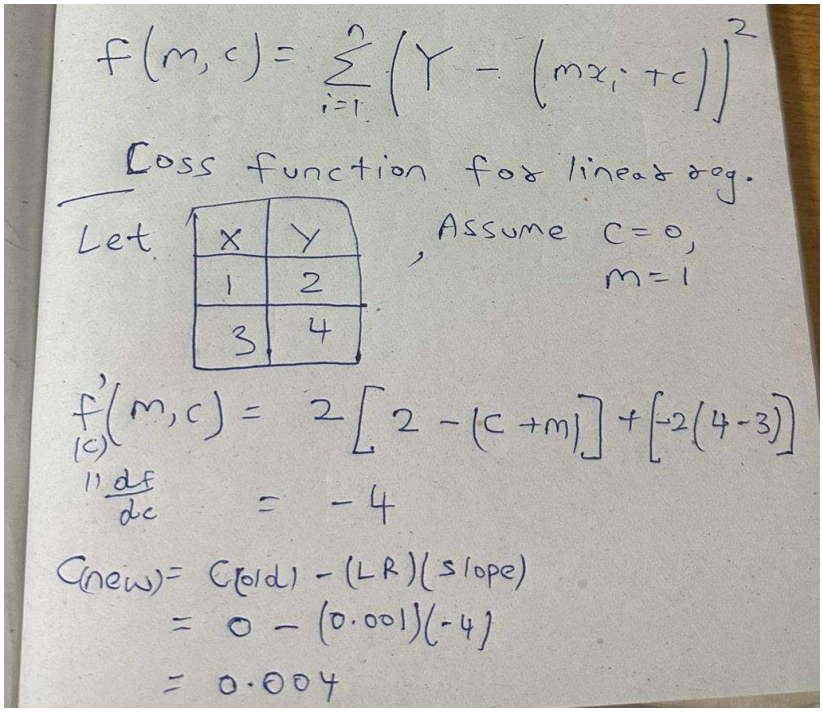
- Epoch – The number of times the algorithm runs on the whole training dataset.
- Sample – A single row of a dataset.
- Batch denotes the number of samples to be taken to for updating the model parameters.
- The learning rate is a parameter that provides the model with a scale of how much model weights should be updated.
- Cost Function/Loss Function – A cost function is used to calculate the cost, which is the difference between the predicted value and the actual value.
- Weights/ Bias – The learnable parameters in a model that controls the signal between two neurons.

Gradient Descent Deep Learning Optimizer

- A gradient is something that is upward/ downward slope
- A gradient descent is an algorithm used to minimize the function by optimizing its parameters.
- For example - after writing a math exam, my friend asks me, how much did I get, I'll tell him to guess the my marks…..he tells 40, then I'll tell no man, 40 is too more, then he says 30, so I'll tell 30 is very less, so he says 35, now coming to the point
- Gradient descent, it starts with a random guess, and then slowly move to the correct path.
- This algorithm uses calculus to modify the values to get local minimum
- New-value = (old-value) - step-size
- Step size = learning rate *slope
- In this example, we need to minimize the function, the min value of function is (0,0)



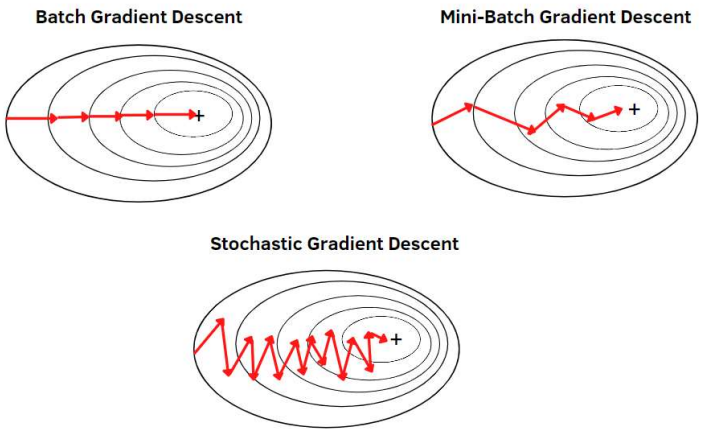
- Gradient descent's job is to minimize the loss function, so the model has more accuracy
- Now, let's see another example.



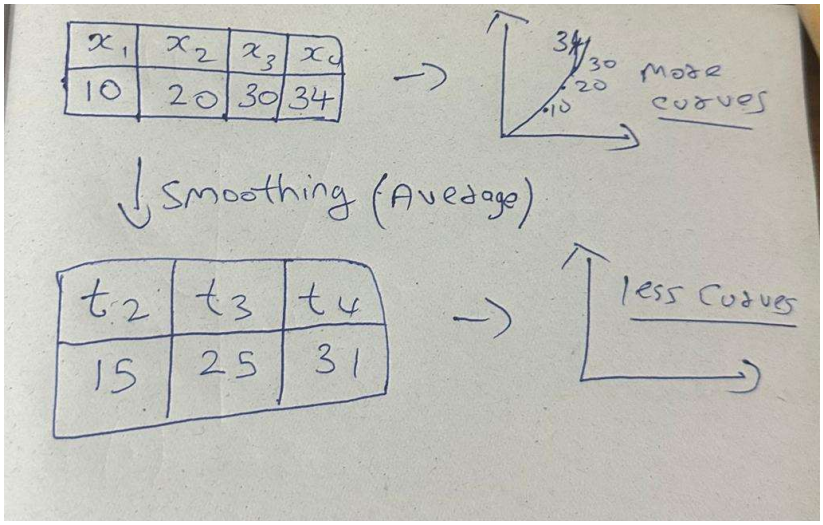
- In the example, first, we should differentiate with respect to 'c', find the new value
- Then differentiate with respect to m, find the new value,,
- We should be doing this, until there is no change in the old and new values
- One drawback of this method is - It is expensive to calculate the gradients if the size of the data is huge.

Stochastic Gradient Descent Deep Learning Optimizer

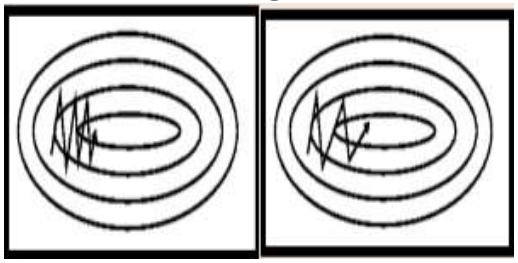
- This concept is used when there are huge datasets
- When there are many rows, The normal gradient descent does the optimization for all values, while this stochastic gradient takes one random row and optimizes . So optimization here becomes more simpler.
- Drawback - when we take one record/row from huge dataset, the results may vary/ biased
- So, here comes the mini batch gradient descent - it takes the batch of records/rows
- SGD uses a higher number of iterations to reach the local minima, More time is taken to optimize the dataset



Stochastic Gradient Descent With Momentum Deep Learning Optimizer



- stochastic gradient descent takes a much more noisy path than the gradient descent algorithm. $\eta'_t = \frac{\eta}{\mathrm{sqrt}(\alpha_t + \epsilon)}$
- Due to this reason, it requires a more significant number of iterations to reach the optimal minimum, and hence computation time is very slow.
- To overcome the problem, we use stochastic gradient descent with a momentum algorithm.



- In the above image, the left part shows the convergence graph of the

stochastic gradient descent algorithm. At the same time, the right side shows SGD with momentum.

- From the image, you can compare the path chosen by both algorithms and realize that using momentum helps reach convergence in less time.
- Note: large momentum and learning rate to make the process even faster. But remember that while increasing the momentum, the possibility of passing the optimal minimum also increases. This might result in poor accuracy and even more oscillations.

Adaptive Gradient Descent Deep Learning Optimizer

- For all the previous Gradient Descent, the learning rates were the same,,,,, but here there are different learning rates for different weights
- Far from the minima, the step size will be more, and closer to the minima, the step size will be less, the learning rate will change every time
- The change in learning rate depends upon the difference in the parameters during training. The more the parameters get changed, the more the learning rate changes
- The Adagrad algorithm uses the below formula to update the weights. Here the alpha(t) denotes the different learning rates at each iteration, n is a constant, and E is a small positive to avoid division by 0.

$$\mathrm{w_t} = \mathrm{w_{t-1}} - \eta'_{\mathrm{t}} \frac{\partial \mathrm{L}}{\partial \mathrm{w(t-1)}}$$

- It is more reliable than gradient descent algorithms and their variants, and it reaches convergence at a higher speed.
- One downside of the AdaGrad optimizer is that it decreases the learning rate aggressively
- There might be a point when the learning rate becomes extremely small.
- and thus the denominator part keeps on increasing
- Due to small learning rates, the model eventually becomes unable to acquire more knowledge, and hence the accuracy of the model is compromised

Root Mean Square Deep Learning Optimizer

- popular optimizers
- The algorithm mainly focuses on accelerating the optimization process by decreasing the number of function evaluations to reach the local minimum.
- The algorithm keeps the moving average of squared gradients for every weight and divides the gradient by the square root of the mean square.

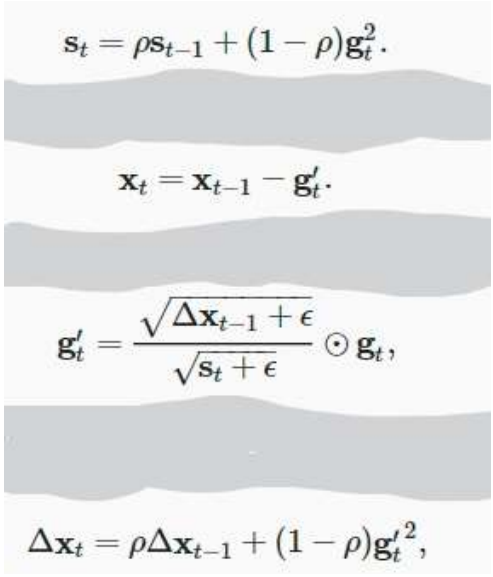
$$v(w,t) := \gamma v(w,t-1) + (1-\gamma)(\nabla Q_i(w))^2$$

$$w := w - \frac{\eta}{\sqrt{v(w,t)}} \nabla Q_i(w)$$

- In simpler terms, if there exists a parameter due to which the cost function oscillates a lot, we want to penalize the update of this parameter
- Suppose you built a model to classify a variety of fishes. The model relies on the factor 'color' mainly to differentiate between the fishes. Due to this, it makes a lot of errors. What RMS Prop does is, penalize the parameter 'color' so that it can rely on other features too. This prevents the algorithm from adapting too quickly to changes in the parameter 'color' compared to other parameters.

AdaDelta Deep Learning Optimizer

- The main problem with the above two optimizers is that the initial learning rate must be defined manually.
- One other problem is the decaying learning rate which becomes infinitesimally small at some point.
- Due to this, a certain number of iterations later, the model can no longer learn new knowledge.
- To deal with these problems, AdaDelta uses two state variables to store the leaky average of the second moment gradient and a leaky average of the second moment of change of parameters in the model.

- 

- Here St and delta Xt denote the state variables, g't denotes rescaled gradient, delta Xt-1 denotes squares rescaled gradients, and epsilon represents a small positive integer to handle division by 0.

1. Import Necessary Libraries

```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K
(x_train, y_train), (x_test, y_test) = mnist.load_data()
print(x_train.shape, y_train.shape)
```

2. Load the Dataset

```
x_train= x_train.reshape(x_train.shape[0],28,28,1)
x_test=  x_test.reshape(x_test.shape[0],28,28,1)
input_shape=(28,28,1)
y_train=keras.utils.to_categorical(y_train)#,num_classes=)
y_test=keras.utils.to_categorical(y_test)#, num_classes)
x_train= x_train.astype('float32')
x_test= x_test.astype('float32')
x_train /= 255
x_test /=255
```

3. Build the Model

```
batch_size=64

num_classes=10

epochs=10

def build_model(optimizer):

model=Sequential()

model.add(Conv2D(32,kernel_size=(3,3),activation='relu',input_shape=input_shape))

model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(256, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy, optimizer= optimizer, metrics=['accur

return model
```

4. Train the Model

```
optimizers = ['Adadelta', 'Adagrad']
for i in optimizers:
model = build_model(i)
hist=model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs,
verbose=1, validation_data=(x_test,y_test))
```

Optimizer	Epoch 1 Val accuracy | Val loss	Epoch 5 Val accuracy | Val loss	Epoch 10 Val accuracy | Val loss	Total Time
Adadelta	.4612 | 2.2474	.7776 | 1.6943	.8375 | 0.9026	8:02 min
Adagrad	.8411 | .7804	.9133 | .3194	.9286 | 0.2519	7:33 min

- The above table shows the validation accuracy and loss at different epochs. It also contains the total time that the model took to run on 10 epochs for each optimizer.

Conclusion

- Each optimizer has its own strengths and weaknesses, and the choice of optimizer will depend on the specific deep-learning task and the characteristics of the data being used.
- The choice of optimizer can significantly impact the speed and quality of convergence during training, as well as the final performance of the deep

learning model.

- The optimizer’s function is to minimize the loss, which measures the discrepancy between the predicted values and actual values, by continuously adjusting the model’s parameters