Disadvantage in Gradient Descent

- Small learning Rate leads to slow convergence
- Large learning rate leads to hinder convergence
- Reducing learning rate according to pre-defined schedule unable to adapt to a dataset's characteristic
- Same learning rate applies to all parameter updates, even we have different frequencies of features

Variations of GD

1. Momentum
2. Nesterov Accelerated Gradient
3. Adagrad
4. Adadelta/RMSProp
5. Adam

Momentum

- Momentum helps accelerate Gradient Descent(GD) when we have surfaces that curve more steeply in one direction than in another
- For updating the weights it takes the gradient of the current step as well as the gradient of the previous time steps
- This helps us move faster towards convergence
- Convergence happens faster when we apply momentum optimizer to surfaces with curves

$$v_t = \gamma v_{t-1} + \eta \nabla J(\theta; x, y)$$
$$\theta = \theta - v_t$$

Momentum Gradient descent takes gradient of previous time steps into considera

Nesterov Accelerated Gradient(NAG)

- Nesterov acceleration optimization is like a ball rolling down the hill but knows exactly when to slow down before the gradient of the hill increases again.
- We calculate the gradient not with respect to the current step but with respect to the future step
- We evaluate the gradient of the looked ahead and based on the importance then update the weights
- Works slightly better than standard Momentum

$$\theta = \theta - v_t$$

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

$\theta - \gamma v_{t-1}$ is the gradient of looked ahead

ADAGRAD — ADAPTIVE GRADIENT ALGORITHM

- We need to tune the learning rate in Momentum and NAG which is an expensive process
- Adagrad is an adaptive learning rate method
- In Adagrad we adopt the learning rate to the parameters
- We perform larger updates for infrequent parameters and smaller updates for frequent parameters.
- For SGD, Momentum, and NAG we update for all parameters θ at once. We also use the same learning rate η. In Adagrad we use different learning rate for every parameter θ for every time step t
- Adagrad eliminates the need to manually tune the learning rate.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \varepsilon}} \cdot g_t$$

G_t is sum of the squares of the past gradients w.r.t. to all parameters θ

ADADELTA

- Adadelta is an extension of Adagrad and it also tries to reduce Adagrad's aggressive, monotonically reducing the learning rate
- It does this by restricting the window of the past accumulated gradient to some fixed size of w. Running average at time t then depends on the previous average and the current gradient
- In Adadelta we do not need to set the default learning rate as we take the ratio of the running average of the previous time steps to the current gradient

$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

$$\Delta\theta = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g_t]} \cdot g_t$$

RMSPROP

- Root Mean Square Propagation
- RMSProp tries to resolve Adagrad's radically diminishing learning rates by using a moving average of the squared gradient
- It utilizes the magnitude of the recent gradient descents to normalize the gradient
- In RMSProp learning rate gets adjusted automatically and it chooses a different learning rate for each parameter
- It divides the learning rate by the average of the exponential decay of squared gradients

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{(1-\gamma)g^2_{t-1} + \gamma g_t + \varepsilon}} \cdot g_t$$

ADAM — ADAPTIVE MOMENT ESTIMATION

- Another method that calculates the individual adaptive learning rate for each parameter from estimates of first and second moments of the gradients
    - It also reduces the radically diminishing learning rates of Adagrad
- It can be viewed as a combination of Adagrad, which works well on sparse gradients and RMSprop which works well in online and nonstationary settings
- Adam implements the exponential moving average of the gradients to scale the learning rate instead of a simple average as in Adagrad. It keeps an exponentially decaying average of past gradients
    - Adam is computationally efficient and has very little memory requirement

Adam optimizer is one of the most popular gradient descent optimization algori

train.py - E:\DeepLearning\9\9\train.py (3.7.8)

File Edit Format Run Options Window Help

```
from numpy import loadtxt #handle/load dataset

from keras.models import Sequential #Empty working area
from keras.layers import Dense #Dense layer

dataset = loadtxt('pima-indians-diabetes.csv', delimiter=',')
x = dataset[:,0:8]
y = dataset[:,8]
print(x)


model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(x, y, epochs=20, batch_size=10)

_, accuracy = model.evaluate(x, y)
print('Accuracy: %.2f' % (accuracy*100))

model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)
model.save_weights("model.h5")
print("Saved model to disk")
```

Deep Learning Terminology - 1

LSTM

- LSTM stands for Long short-term memory
- LSTM has feedback connections which makes it a "general purpose computer."
- It can process not only a single data point but also entire sequences of data.
- They are a special kind of RNN which are capable of learning long-term dependencies.

- Step 1: The network decides what to forget and what to remember.
- Step 2: It selectively updates cell state values.
- Step 3: The network decides what part of the current sta makes it to the output.

Deep Learning Terminology - 2

Autoencoder

- Autoencoder is an artificial neural network. It can learn representation for a set of data without any supervision.
- The network automatically learns by copying its input to the output
- They can learn efficient ways of representing the data
- It contains three layers:
- **Encoder** - is used to compress the input into a latent space. It encodes the input images as a compressed representation in a reduced dimension. The compressed images are the distorted version of the original image.
- **Decoder** - The decoder layer decodes the encoded image back to its origi dimension. The decoded image is a reduced reconstruction of the original image

Deep Learning Terminology - 3

Bagging and Boosting

- Bagging and Boosting are ensemble techniques to train multiple models using the same learning algorithm and then taking a call.
- With Bagging, we take a dataset and split it into training data and test data. Then we randomly select data to place into the bags and train the model separately.
- With Boosting, the emphasis is on selecting data points which give wrong output to improve the accuracy.