Objective

To study different Optimizers on model i.e. SGD , Adagrad , RMSprop , Adam.

Optimizers

Stochastic gradient descent (SGD)

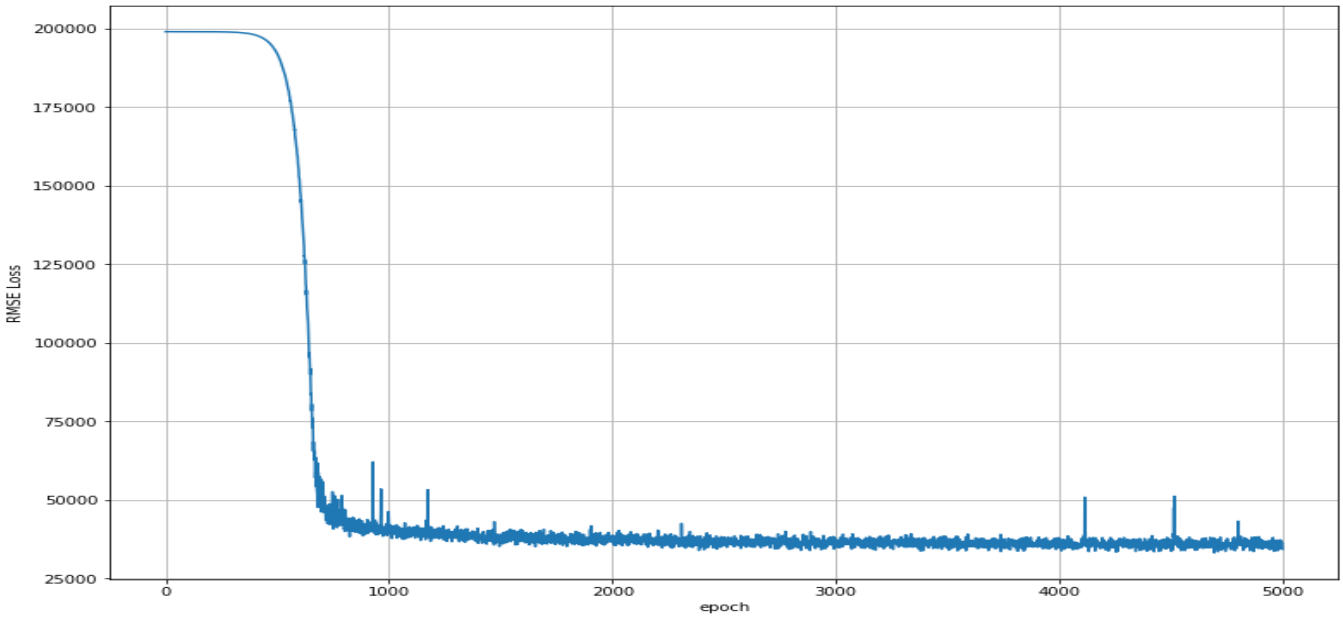
SGD does away with this redundancy by performing one update at a time. It is therefore usually much faster. SGD performs frequent updates with a high variance that cause the **objective function to fluctuate heavily**.

$$W_t = W_{t-1} - \eta \times \frac{dL}{dw_{t-1}}$$

While batch gradient descent converges to the minimum of the basin the parameters are placed in, SGD's fluctuation, on the one hand, enables it to jump to new and potentially better local minima. This ultimately complicates convergence to the exact minimum,

As SGD will keep overshooting. However, it has been shown that when I slowly decrease the learning rate, SGD shows the same convergence behaviour as batch gradient descent, almost certainly converging to a local or the global minimum for non-convex and convex optimization respectively.

Here is Loss Vs epoch plot I got for SGD as an optimizer.



Problem Faced

- A learning rate that is too small leads to 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.

- Learning rate schedules try to adjust the learning rate during training by i.e. annealing, i.e. reducing the learning rate according to a predefined schedule or when the change in objective between epochs falls below a threshold. These schedules and thresholds, however, have to be defined in advance and are thus unable to adapt to a dataset's characteristics.
- The same learning rate applies to all parameter updates. If the data is sparse and the features have very different frequencies, we might not want to update all of them to the same extent, but perform a larger update for rarely occurring features.
- The difficulty arises in fact not from local minima but from saddle points. These saddle points are usually surrounded by a plateau of the same error, which makes it hard for SGD to escape, as the gradient is close to zero in all dimensions.

SGD with Momentum

SGD oscillates across the slopes of the ravine while only making hesitant progress along the bottom towards the local optimum. It does this by adding a fraction γ of the update vector of the past time step to the current update vector.

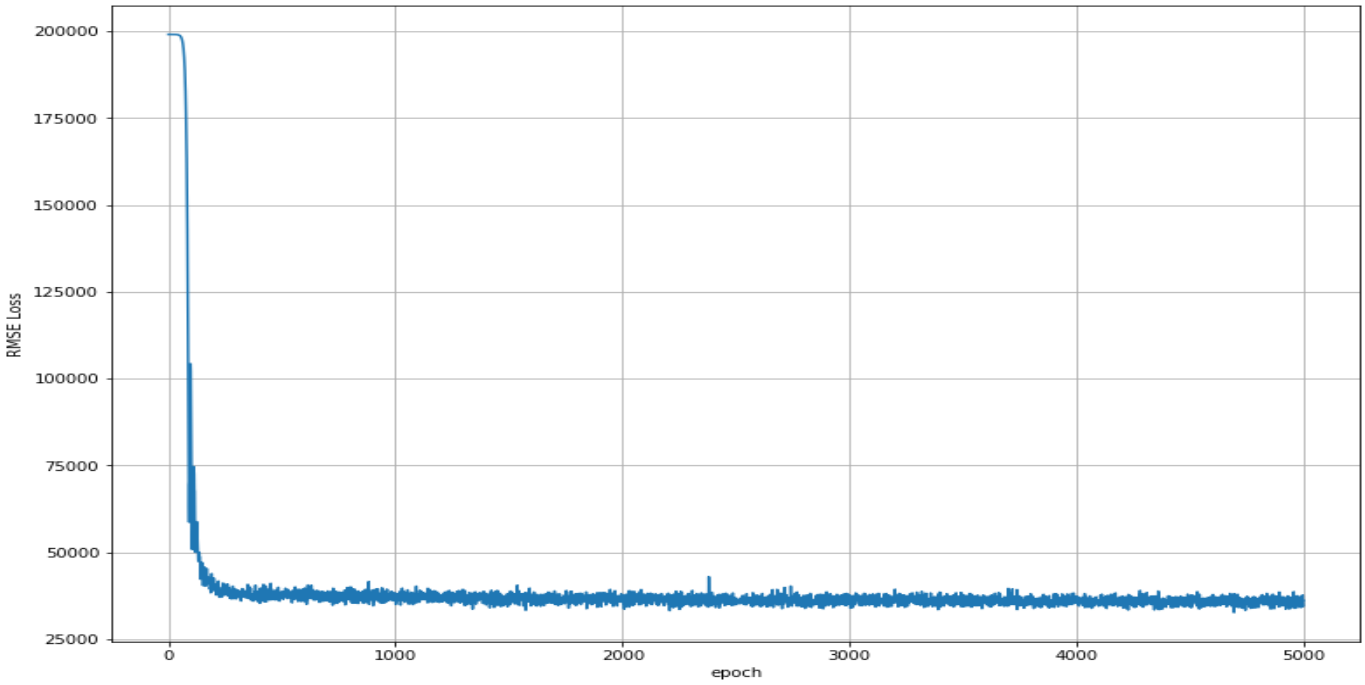
$$W_t = W_{t-1} - \eta \times V_{w_t}$$

$$V_{w_t} = \beta \times V_{w_{t-1}} + (1-\beta) \times \left(\frac{dL}{dw_{t-1}}\right)$$

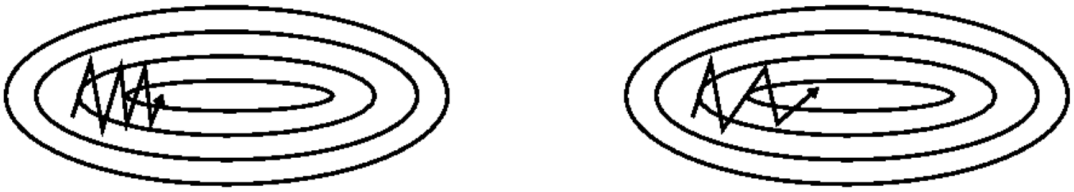
$$V_{w_{t=0}} = 0$$

The momentum term γ is usually set to 0.9 or a similar value. Essentially, when using momentum, we push a ball down a hill. The ball accumulates momentum as it rolls downhill, becoming faster and faster on the way (until it reaches its terminal velocity, if there is air resistance).

Here is a Loss Vs epoch plot I got for SGD with momentum as an optimizer.



The same thing happens to our parameter updates: 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.**



(a) SGD without momentum (b) SGD with momentum

(*Here we can observe the high fluctuation in SGD without momentum make that slow to reach the minimum)

Adagrad

It is an algorithm for gradient-based optimization. It adapts the learning rate to the parameters, performing larger updates for infrequent and smaller updates for frequent parameters. For this reason, it is well-suited for dealing with sparse data.

Adagrad modifies the general learning rate η at each time step t for every parameter W_t based on the past gradients that have been computed for

$$W_t = W_{t-1} - \eta' \times \frac{dL}{dWt-1}$$

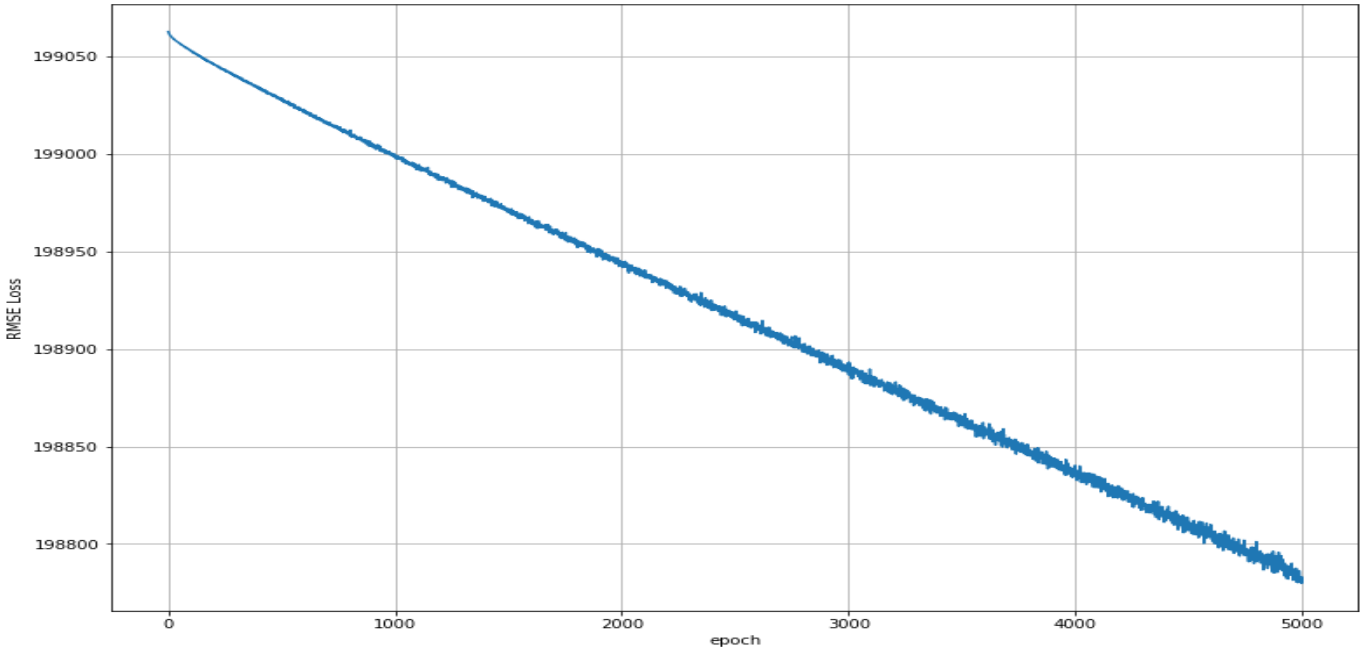
$$\eta' = \frac{\eta}{(\alpha_t+\varepsilon)^{1/2}}$$

$$\alpha_t = \sum_{1}^{t}\left(\frac{dL}{dw_t}\right)^2$$

α_t is the sum of the squares of the gradients w.r.t. W_t up to time step t, while is a smoothing term that avoids division by zero (usually on the order of 1e−8 i.e ε). Interestingly, without the square root operation, the algorithm performs much worse.

As η' contains the sum of the squares of the past gradients w.r.t. to all parameters W.

Here is Loss Vs epoch plot I got for Adagrad as an optimizer.



Benefit

- One of Adagrad's main benefits is that it **eliminates the need to manually tune the learning rate**. Most implementations use a default value of 0.01 and leave it at that.

Drawback

- Adagrad's main weakness is its accumulation of the squared gradients in the denominator: Since every added term is positive, the accumulated sum keeps growing during training. This in turn **causes the learning rate to shrink** and eventually **become infinitesimally small**, at which point the algorithm is no longer able to acquire additional knowledge.

RMSprop

Adaptive learning rate method RMSprop have developed for the need to resolve Adagrad's radically **diminishing learning rates**.

$$W_t = W_{t-1} - \eta' \times \frac{dL}{dwt-1}$$

where,

$$\eta' = \frac{\eta}{(S_{w_t} + \varepsilon)^{1/2}}$$

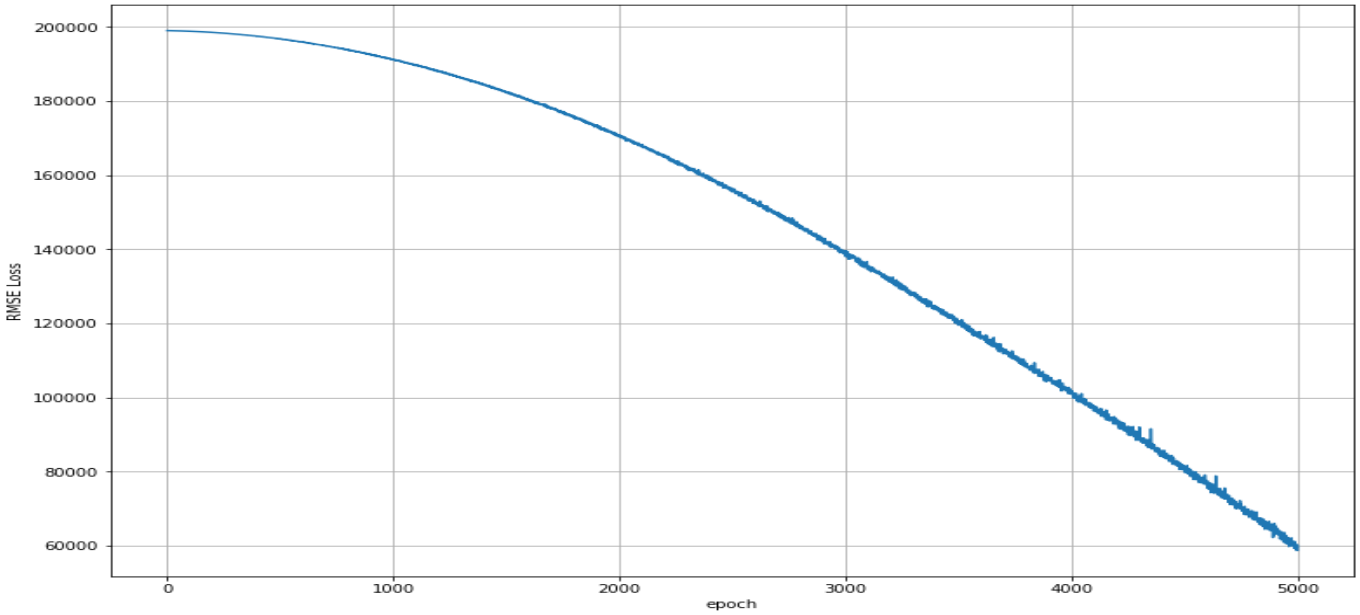
$$S_{w_t} = \beta \times S_{w_{t-1}} + (1-\beta)\left(\frac{dL}{dw_{t-1}}\right)^2$$

$$S_{w_{t=0}} = 0$$

As before in Adagrad the learning gets diminished due to the squared sum of gradients. But here the term S_{w_t} handles the huge increase in the value of alpha.

RMSprop as well divides the learning rate by an exponentially decaying average of squared gradients.
For the general case I prefer β to be set to 0.9, while a good default value for the learning rate η is 0.001.

Here is Loss Vs epoch plot I got for RMSprop as an optimizer.



Adam

It is a method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients S_{w_t} like RMSprop, Adam also keeps an exponentially decaying average of past gradients V_{w_t}, similar to momentum

$$V_{w_t} = \beta_1 \times V_{w_{t-1}} + (1-\beta_1) \times \left(\frac{dL}{dw_{t-1}}\right) \quad , V_{w_{t=0}} = 0$$

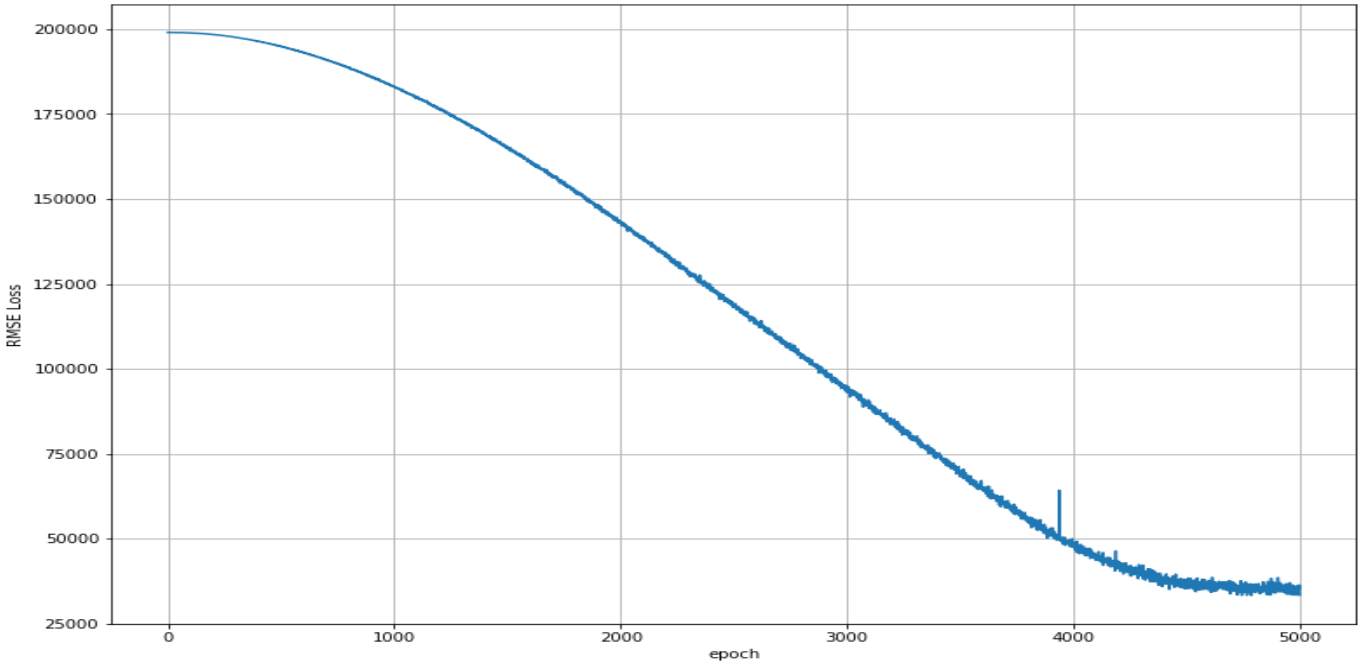
$$S_{w_t} = \beta_2 \times S_{w_{t-1}} + (1-\beta_2)\left(\frac{dL}{dw_{t-1}}\right)^2 \quad , S_{w_{t=0}} = 0$$

S_{w_t} and V_{w_t} are estimates of the first moment (the mean) and the uncentered variance of the gradients respectively. Adam has two hyperparameter i.e β_1, β_2.

They then use these to update the parameters just as in RMSprop, which yields the Adam update rule

$$W_t = W_{t-1} - \frac{\eta}{\sqrt{(S_{w_t} + \varepsilon)}} \times V_{w_t}$$

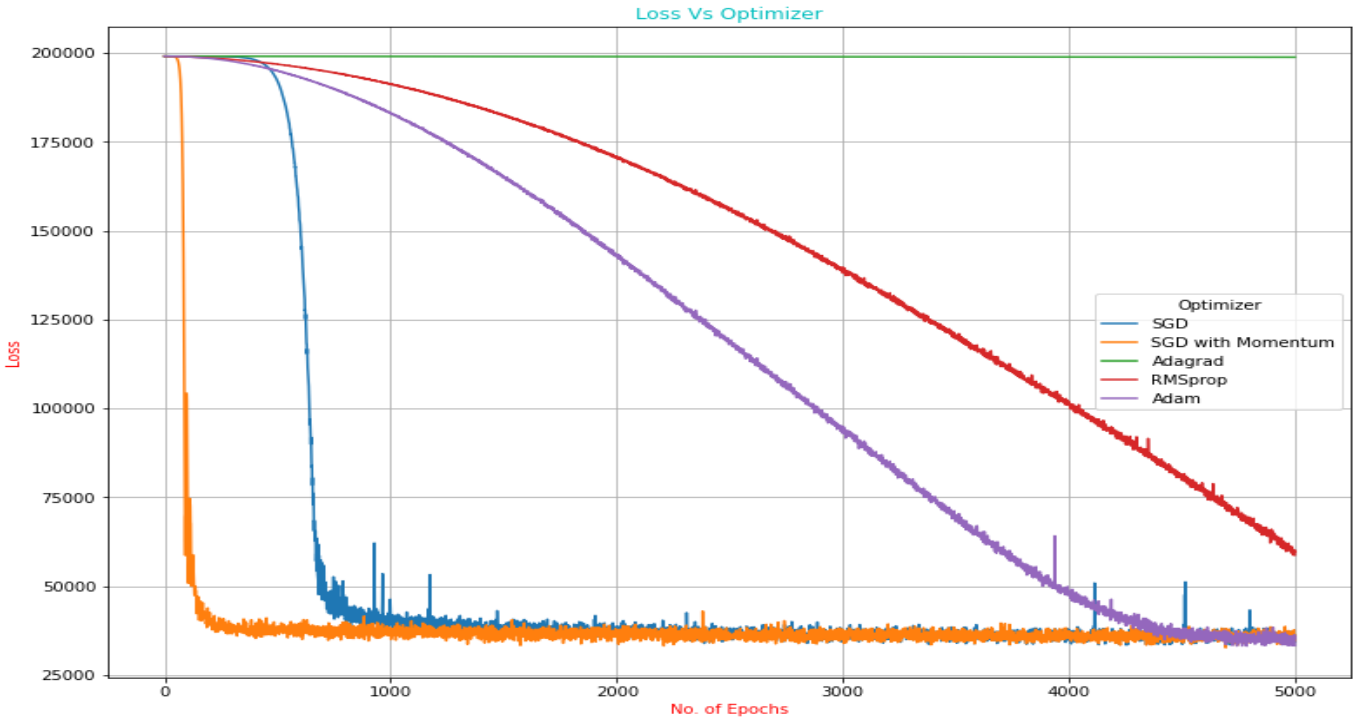
Here is Loss Vs epoch plot I got for Adam as an optimizer.



Adam with less fluctuation and self adjusting learning rate makes it more efficient and effective over other optimisers. From my personal point of view I found this as more effective over others in my algorithm for training.

(*All above optimisers can be used for biases as well)

Conclusion



Here is the plot I observed by comparing different optimisers over my model.

From the above plot one observes all features as I have mentioned above. Adam is found to be the best optimizer as above, causing minimum loss after 5000 epochs.

Future Work

With different dataset & model optimizers behave differently. That makes the study of optimizers on various algorithms more complicated. After this I'll proceed for complex algorithms for study of the same.

You can find the code here

Thank you,
Omm Prakash Sahoo
Ops#0595
EEE