Illustrations

March 10, 2019

```
In [1]: import matplotlib.pyplot as plt
        import numpy as np
```

The plots for various optimizers for gradient descent will be plotted here to have an apples to apples comparision of which descent algorithm outperforms the Stochastic Gradient Descent

```
In [8]: SGD = np.loadtxt("SGD.txt")
        Momen = np.loadtxt("Momen.txt")
        Nes = np.loadtxt("Nes.txt")
        AdaG = np.loadtxt("AdaG.txt")
        Adam = np.loadtxt("Adam.txt")
        RMS = np.loadtxt("RMS.txt")
```

```
In [9]: import matplotlib.pyplot as plt
```

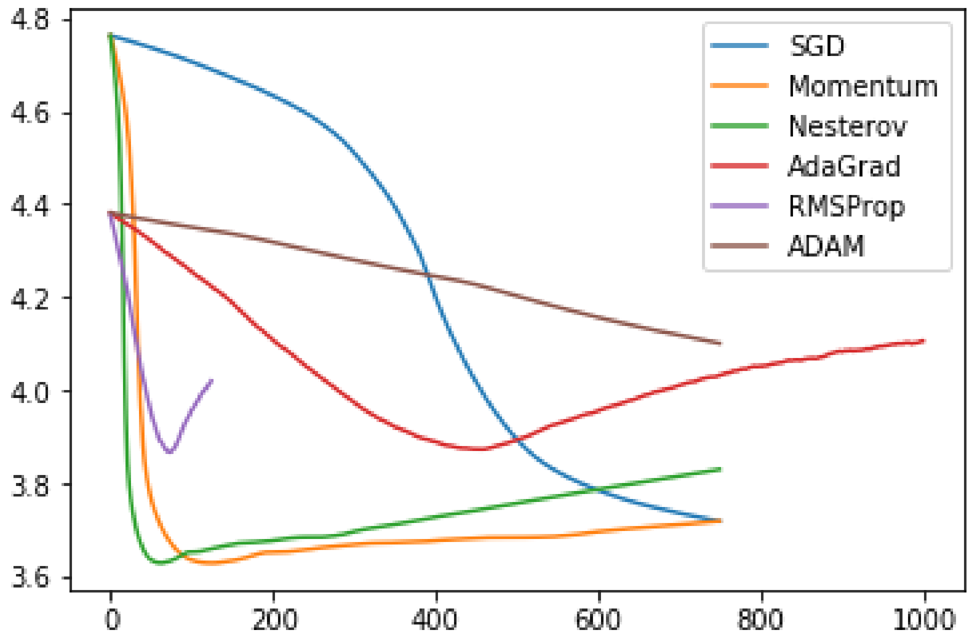
Note: 1. For sake of simplicity and compactness, SGD has been coded for minibatch size 1 and been followed along. A decay of 1e-9 has been implimented for SGD.

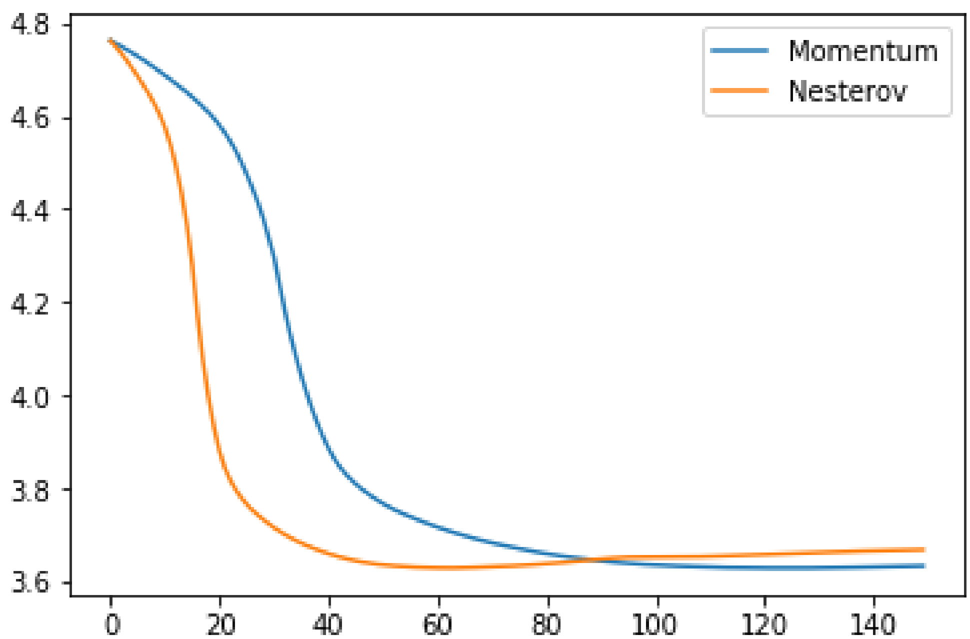
2. One wierd observation is that to reach an accuracy of over 80%, I had to train the network further even when the loss has seemingly reached the minimum most value.

3. Also, the loss being taken here is the average loss over an epoch. That might be the reason, but nevertheless

```
In [13]: labels = ["SGD","Momentum","Nesterov","AdaGrad","RMSProp","ADAM"]
         plt.plot(SGD[:750])
         plt.plot(Momen[:750])
         plt.plot(Nes[:750])
         plt.plot(AdaG)
         plt.plot(Adam[:750])
         plt.plot(RMS[:750])
         plt.legend(labels)
         plt.show()

         labels = ["Momentum","Nesterov"]
         plt.plot(Momen[:150])
         plt.plot(Nes[:150])
         plt.legend(labels)
         plt.show()
```





0.1 Observations

1. Stochastic Gradient Descent took the longest to converge

2. AdaGrad and RMSProp had to be trained further even though the loss had seemingly reached a minimum (loss in this scope is the avg loss over an epoch)
3. Adam was the fastest to converge followed by RMSProp
4. Nesterov is highly sensitive to hyper-parameter tuning compared to Momentum approach