Ex 8.1

a) Adagrad optimizer:

i) Adagrad optimizes the network by separately tuning the hyperparam- -eters giving priority to prominent features. This allows us to efficiently deal with sparse data where some features hold lesser significance. The hyperparameters such as learning rate is adjusted in every layer according to the sparsity of data. For a hyperparameter θ_i at time step t, the update rule is modified as below:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g(t)$$

where G_t contains the sum of the squares of the past gradients with respect to all hyperparameters (diagonally). In each time step the hyperparameters are adjusted along with the initial rate η. Whereas in Stochastic gradient descent algorithm we use a fixed learning rate. This can affect the performance of the network if the learning rate is too big or small. Adagrad adaptively solves this problem and leads the network to converge faster.

ii) Adagrad performs its best when presented with sparse data. However, in practice, it can lead to G_t

have very large values as it accumulates gradient information at each time step. As this values are in the denominator, it can lead the learning rate to 0. At this point, the model won't be able to learn any new information.

iii) The key property of the algorithm is the matrix G_t which is described as below:

$$G_t = diag\left(\sum_{\tau=1}^{t} g_\tau g_\tau^T\right)$$

where $g_\tau = \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta \mathcal{L}(x^{(i)}, y^{(i)}, \theta_t)$

G_t as described in (i) accumulates the outer products of the gradients. This directly controls the learning rate with an added ϵ to avoid division by zero. However, it can be impractical to compute the square root of the whole matrix.

b) Adam optimizer:

i) Both RMSprop and Momentum work towards preventing the oscillating motion of simple SGD by introducing running averages to the weight update rule. However, RMSprop considers the mean squared average of the gradients of previous step whereas momentum considers only the weighted average of gradients. RMSprop also takes accounts into the horizontal and vertical oscillations by setting different update rule for weights and biases. It is possible to combine both of these algorithms to reap benefit of both in the following way:

$$w_{t+1} = w_t - \eta \frac{v_t}{\sqrt{s_t + \epsilon}} * g_t$$

where,

$$v_t = \beta_1 * v_{t-1} - (1 - \beta_1) * g_t$$

$$s_t = \beta_2 * s_{t-1} - (1 - \beta_2) * g_t^2$$

v_t comes from Momentum and s_t comes from RMSprop algorithm. This allows us to use both the exponential weighted average of gradients and root squared average of gradients to compute our hyperparameters. This leads to a faster convergence and lesser iterations.

ii) Adam is similar to RMSprop with momentum with two added corrections of adjusting time step size and bias correction before the update. Given the definition of v_t and s_t in (i), bias corrected moments are calculated in Adam algorithm as below:

$$\hat{v}_t = \frac{v_t}{1 - \beta_1^t}$$

$$\hat{s}_t = \frac{s_t}{1 - \beta_2^t}$$

The time-step size is also controlled by these two moments:

$$\Delta t = \eta \cdot \frac{\hat{v}_t}{\sqrt{\hat{s}_t}}$$

This time step size is invariant to the scaling of gradients and adjusts itself as per the sparsity of the data. Our final step of the parometer update is then:

$$w_{t+1} = w_t - \eta \cdot \frac{\hat{v}_t}{\sqrt{\hat{s}_t + \epsilon}} * g_t$$

The main difference between Adam and RMSprop with momentum is therefore the moments considered in each time step. Adam considers running average of first and second moments whereas RMSprop considers momentum of the scaled gradients. RMSprop also lacks the bias-correction term which can lead to slower convergence or no convergence at all given sparse data. Hence, using Adam is preferable to train the neural network.