Implementation

SGD. It can be regarded as stochastic approximation of gradient descent optimization, since it replaces the actual gradient which calculated for the entire dataset by an estimate thereof (calculated from a randomly selected subset of data – in this algorithm we use 1 randomly chosen data from the dataset in each iteration). One advantage of using this method is that it could reduce computational burden especially in high-dimensional optimization and achieves faster iterations in trade for a lower convergence rate.

$$\theta_j = \theta_j - \alpha \nabla_{\theta_j} \mathcal{L}(\theta)$$

As it can be seen, the update rule for the parameters is the same as gradient descent method and the only difference is that we use only a subset of data (1 data point) in each iteration.

- Advantages: faster algorithm compare to simple gradient descent, can be used to learn online.
- Disadvantages: this algorithm preforms frequent updates with a high variance that cause the objective function fluctuate heavily, because of its' fluctuations, SGD will keep overshooting and this ultimately complicates convergence to the exact minimum.

The best result I achieved with this algorithm is provided as follows.

Iterations	Alpha	Cost	Tolerance (Stop Criterion)
29	0.3	933.0125	0.1

SGD with Momentum. As it discussed in PART I. momentum can help the SGD to reach the global minimum and get out form local minimums. Equation for this method is available in previous section.

- Advantages: faster convergence, reduced oscillations.
- Disadvantages: is not adaptive, does not pay attention to each parameters' importance, does not have notion of where it is going so does not slow down before the hill, slopes up again.

The best result I achieved with this algorithm is provided as follows.

Iterations	Alpha	Momentum	Cost	Tolerance (Stop Criterion)
48	0.01	0.9	933.4152	0.1

AdaGrad. In this method, learning rate is adapted component-wise to the parameters by incorporating knowledge of past observations. It preforms larger updates for those parameters that are related to infrequent features and smaller updates for frequent one. It preforms smaller updates as a result, it is well-suited when dealing with sparse data, each parameter has its own learning rate that improves performance on problems with sparse gradients.

SGD with Momentum

$$v_{t+1} \leftarrow \rho v_t + \nabla_\theta \mathcal{L}(\theta)$$
$$\theta_j \leftarrow \theta_j - \epsilon v_{t+1}$$

AdaGrad

$$g_0 = 0$$
$$g_{t+1} \leftarrow g_t + \nabla_\theta \mathcal{L}(\theta)^2$$
$$\theta_j \leftarrow \theta_j - \epsilon \frac{\nabla_\theta \mathcal{L}}{\sqrt{g_{t+1}} + 1e^{-5}}$$

As it can be seen form the equation, the idea in this method is that it keeps the running sum of squared gradients during optimization (g). With using this, we can accelerate the update process along the axis with small gradients by increasing the gradient along that axis. On the other hand, the updates along the axis with the large gradient slow down a bit.

- Advantages: eliminates the need to manually tune the learning rate, convergence is faster and more reliable compare to SGD, update process is adaptive, is incredibly slow.
- Disadvantages: since every added term to the accumulation of the squared gradients, is positive, the total sum keeps growing, causing the learning rate to shrink and eventually become very small.

The best result I achieved with this algorithm is provided as follows.

Iterations	Alpha	Batch size	Cost	Tolerance (Stop Criterion)
10436	0.99	64	933.4152	0.1

If we set alpha to larger values (greater than 1), number of iterations will decrease, but with doing that we are giving more weight to the gradient of the loss function than to the current value of the parameters. Therefore; I prefer to set alpha to the value less than one (0.99). Also, because the animation of fitting a line to the dataset with 10,000 iterations is very large, the gif provided in the output only consists of first 150 iterations of the process. (It is too slow)

RMSProp. This method is AdaGrad version of gradient descent that uses a decaying average of partial gradients in the adaptation of the step size for each parameter. The use of a decaying moving average allows the algorithm to forget early gradients and focus on the most recently observed partial gradients seen during the progress of the search, overcoming the limitation of AdaGrad.

AdaGrad

$$g_0 = 0$$
$$g_{t+1} \leftarrow g_t + \nabla_\theta \mathcal{L}(\theta)^2$$
$$\theta_j \leftarrow \theta_j - \epsilon \frac{\nabla_\theta \mathcal{L}}{\sqrt{g_{t+1}} + 1e^{-5}}$$

RMS Prop

$$g_0 = 0, \alpha \simeq 0.9$$
$$g_{t+1} \leftarrow \alpha \cdot g_t + (1-\alpha)\nabla_\theta \mathcal{L}(\theta)^2$$
$$\theta_j \leftarrow \theta_j - \epsilon \frac{\nabla_\theta \mathcal{L}}{\sqrt{g_{t+1}} + 1e^{-5}}$$

The update step in the case of RMSProp looks exactly the same as in AdaGrad but in RMSProp we multiply the sum of squared gradients by a decay rate and add the current gradient weighted by (1- α).

- Advantages: keeping learning rate optimally high, is adaptive, faster than GD or SGD, overcome getting stuck in saddle points.
- Disadvantages: manual learning rate.

The best result I achieved with this algorithm is provided as follows.

Iterations	Alpha	Momentum	Batch size	Cost	Tolerance (Stop Criterion)
126	0.9	0.9	64	933.0784	0.1

Adam. This method combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems. Instead of adapting the parameter learning rates based on the average first moment (the mean) as in RMSProp, Adam also makes use of the average of the second moments of the gradients (the uncentered variance). Specifically, the algorithm calculates an exponential moving average of the gradient and the squared gradient, and the parameters beta1 and beta2 control the decay rates of these moving averages.

$$m_0 = 0, v_0 = 0$$

$$m_{t+1} \leftarrow \beta_1 m_t + (1 - \beta_1)\nabla_\theta \mathcal{L}(\theta)$$ Momentum

$$v_{t+1} \leftarrow \beta_2 v_t + (1 - \beta_2)\nabla_\theta \mathcal{L}(\theta)^2$$ RMS Prop

$$\theta_j \leftarrow \theta_j - \frac{\epsilon}{\sqrt{v_{t+1}} + 1e^{-5}} m_{t+1}$$ RMS Prop + Momentum

- Advantages: slow down when converging to local minimum whereas momentum, is adaptive, more reliable results, computationally efficient, works well on problems with noisy or sparse gradients and large datasets.
- Disadvantages: does not converge to an optimal minimum in some areas (motivation for AMSGrad), weight decay problem, .

The best result I achieved with this algorithm is provided as follows.

Iterations	Alpha	Beta1	Beta2	Batch size	Cost	Tolerance (Stop Criterion)
40	0.5	0.9	0.999	64	933.093	0.1