UNIVERSITÄT DES SAARLANDES

Saarland University, Department of Computer Science
Neural Network Assignment 8

Deborah Dormah Kanubala (7025906) , Irem Begüm Gündüz (7026821), Anh Tuan Tran (7015463)

January 10, 2023

Exercise 8.1 Optimization Algorithms

Exercise 8.1.a

(a) No, it is non-convex in nature [1]. One is not guaranteed to converge to the global minima and there may be multiple local minima. Also, we do make use of non-linear activation functions, and introducing this non-linearity will have a non-convex error surface.

(b) We would need Deep Neural Network to be convex because :

- we can find the best solution with a convex function since the solution of non-convex is sub-optimal.
- it will easily converge and less time used in training.

Exercise 8.1.b

(a) The idea of AdaGrad is to use different learning rates for each iteration, thus having different learning rates for each weight. This is because some features are sparse while others are not. As such, modifying the learning rate for each weight will be great for optimization [2].

Compute gradient estimates:

$$g \longleftarrow \frac{1}{m}\nabla_\theta Loss\ function$$

Accumulate squared gradient

$$r \longleftarrow r + g \odot g \tag{1}$$

compute the update

$$\nabla\theta \longleftarrow -\frac{\epsilon}{\delta + \sqrt{r}} \odot g$$

Apply update:

$$\theta \longleftarrow \theta + \nabla\theta$$

(b) AdaGrad is well suited for sparse data. Since the learning rate of parameters is based on the sum of squares of the gradients of that parameter, parameters that have not been updated frequently have a larger learning rate hence allowing the model to converge faster. Also, the AdaGrad cost function is the same in all directions. This therefore may lead to quick converges in some directions and slow in others.

(c) As the number of iterations increases, the learning rate will decrease for each parameter and may become infinitesimally small. Thereby making the old weights almost the same weight as the new weight leading to slow convergence.

Exercise 8.1.c

(a) The similarity between these two is that they are both optimization techniques introduced to improve the performance of machine learning models by exponentially decaying the gradient. They both also propose different methods to adjust the learning rate of parameters individually.

RMSProp solves the problem (disadvantage of AdaGrad) by allowing the moving average of the sum of squared gradients for each weight [5] rather than the sum. It also includes a decay term to make the running average of the squared gradient exponentially decaying. Its update is given as:

$$g \longleftarrow \frac{1}{m}\nabla_\theta Loss\ function$$

Accumulate squared gradient

$$r \longleftarrow \rho r + (1-\rho) g \odot g \tag{2}$$

compute the update

$$\nabla\theta \longleftarrow -\frac{\epsilon}{\delta + \sqrt{r}} \odot g$$

Apply update:

$$\theta \longleftarrow \theta + \nabla\theta$$

Momentum accelerates learning by exponentially decaying the moving average of past gradients and continues to move in their direction. It introduces the velocity variable v which is the direction and speed at which parameters move through the parameter space. It solves primarily two problems [3]:

- Poor conditioning of the Hessian Matrix
- Variance in the stochastic gradient

The update rule is given by:

$$v \longleftarrow \alpha - \epsilon\nabla_\theta(\frac{1}{m} Loss\ function)$$
$$\theta \longleftarrow \theta + v$$

Yes, it is possible to combine these two and have RMSProp with Momentum otherwise known as ADAM (Adaptive Momentum Estimation) and this leverages the advantages of these two. It uses the moving average gradient from momentum to adjust the learning and uses the running average of the squared gradient from RMSProp to normalize the gradient.

(b) Adam uses the historical gradient information of each parameter to adjust the learning rate individually. [4]

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\Delta_\theta f_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\Delta_\theta f_t)^2$$

$$\hat{m}_t = m_t/(1 - \beta_1^t)$$

$$\hat{v}_t = v_t/(1 - \beta_2^t)$$

$$\theta_t = \theta_{t-1} - \alpha * \hat{m}_t/(\sqrt{(}\hat{v}_t) + \epsilon)$$

Advantages

- It is invariant to diagonal rescaling of the gradients.
- Well suited for problems that are large in terms of data and/or parameters

Exercise 8.1.d

AdemW differs from Adam with its implementation of weight decay. In Adam, the regularization term is added to the cost function. Afterward, the gradients computed while also adding weight decay. However, this leads gradients to be effected not only from gradients of the loss function but also of the regularization term. AdamW is offered as a solution by adding weight decay after the controlling the parameter wise step size. Therefore, the gradients no longer effected from weight decay or regularization term.

References

[1] Si Chen and Yufei Wang. Convolutional neural network and convex optimization. *Dept. of Elect. and Comput. Eng., Univ. of California at San Diego, San Diego, CA, USA, Tech. Rep*, 2014.

[2] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. book in preparation for mit press. *URL¡ http://www. deeplearningbook. org*, 1, 2016.

[4] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[5] Zijun Zhang. Improved adam optimizer for deep neural networks. In *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, pages 1–2. Ieee, 2018.