DS 211: Numerical Optimization Course Project

Study of Adam optimizer and its implementation to train a deep neural network for handwritten digit recognition.

Names of students:

1. Pawar Rushikesh Gajanansa (SR no. 23-1-22581)
2. Dhamale Vivek Shrikrishna (SR no. 23-1-22864)

Introduction:

Many problems in Engineering and sciences can be formulated as optimization of some parameterized objective function with respect to parameters. If the function is differentiable with respect to the parameters, then gradient descent is a relatively efficient optimization technique, since computation of first order partial derivatives with respect to all the parameters has same computational complexity as just evaluating the function. Often, objective function is stochastic and hence stochastic gradient descent is used in such cases.

Adam is a efficient stochastic gradient descent algorithm that requires first order gradients with little memory requirements. The name Adam is derived from adaptive moment estimation. In this project we will investigate Adam algorithm, maths that goes behind this algorithm, along with convergence and advantages. In the end, we have implemented Adam to train a deep neural network for handwritten digit recognition. [1]

Algorithm: [1]

Required inputs: α (step size hyperparameter); $\beta_1, \beta_2 \in [0,1)$ (exponential decay rate hyperparameters for the moment estimates); $f(\boldsymbol{\theta})$ (stochastic objective function with parameter $\boldsymbol{\theta}$); $\boldsymbol{\theta_0}$ (initial parameter vector).

Initialize:

$\boldsymbol{m_0} \leftarrow \boldsymbol{0}$ //Initialize 1st moment vector

$\boldsymbol{v_0} \leftarrow \boldsymbol{0}$ //Initialize 2nd moment vector

$k \leftarrow 0$ //Initialize Iteration count/timestep

While $\boldsymbol{\theta_k}$ is not converged, **do**:

$k \leftarrow k + 1$

$\boldsymbol{g_k} \leftarrow \boldsymbol{\nabla_\theta} f_k(\boldsymbol{\theta}_{k-1})$ //calculate gradient of objective function at iteration k w.r.t. parameter // vector and evaluate it using parameters iterate at k-1 iteration

$\boldsymbol{m}_k \leftarrow \beta_1 \boldsymbol{m}_{k-1} + (1-\beta_1)\boldsymbol{g}_k$ // update biased first moment estimate

$\boldsymbol{v}_k \leftarrow \beta_2 \boldsymbol{v}_{k-1} + (1-\beta_2)\boldsymbol{g}_k^2$ // update biased second moment estimate

$\widehat{\boldsymbol{m}_k} \leftarrow \boldsymbol{m}_k / \left(1-\beta_1^k\right)$ // calculate bias corrected first moment estimate

$\widehat{\boldsymbol{v}_k} \leftarrow \boldsymbol{v}_k / \left(1-\beta_2^k\right)$ // calculate bias corrected second moment estimate

$\boldsymbol{\theta_k} \leftarrow \boldsymbol{\theta}_{k-1} - \alpha\ \widehat{\boldsymbol{m}_k} / \left(\sqrt{\widehat{\boldsymbol{v}_k}} + \epsilon\right)$ // update parameter vector

End while

Return $\boldsymbol{\theta}_k$ //saddle point or optimum point

Notations: bold face letters (e.g. $\boldsymbol{\theta}_k$) indicate vectors, normal face letters are used for scalars.

Operations: $\boldsymbol{g}_k^2$ and $\sqrt{\widehat{\boldsymbol{v}_k}}$ indicate elementwise square and elementwise square root respectively. Vector-Vector division is also elementwise. (ϵ is added to avoid division by zero)

Optimum values of hyperparameters:

Suggested in Adam Paper [1]

$$\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$$

Implemented in top deep learning libraries. [2] (learning rate = step size parameter)

TensorFlow: learning_rate=0.001, beta1=0.9, beta2=0.999, epsilon=1e-08.

Keras: lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0.

Blocks: learning_rate=0.002, beta1=0.9, beta2=0.999, epsilon=1e-08, decay_factor=1.

Lasagne: learning_rate=0.001, beta1=0.9, beta2=0.999, epsilon=1e-08

Caffe: learning_rate=0.001, beta1=0.9, beta2=0.999, epsilon=1e-08

MxNet: learning_rate=0.001, beta1=0.9, beta2=0.999, epsilon=1e-8

Torch: learning_rate=0.001, beta1=0.9, beta2=0.999, epsilon=1e-8

Mathematics behind Adam [1]:

Objective function $f(\boldsymbol{\theta})$:

a Noisy stochastic scalar valued function, parameterized by parameter vector $f(\boldsymbol{\theta}|\mathbf{x})$. Stochasticity might come from evaluation of function at random minibatches ($\mathbf{x_i}$). i.e., different random minibatches will have different datapoints and hence function value will be different. Or stochasticity might be from inherent noise. Hence, we can say f is a random variable. Let $f_1(\boldsymbol{\theta})$, $f_2(\boldsymbol{\theta})$,.., $f_k(\boldsymbol{\theta})$ be the realizations of f at different instances 1, 2...k. In case of deterministic function, we want to minimize the function value and find parameters, that do this for a particular data. But here, as objective function is stochastic, minimizing the expected value of objective function ($E[f(\boldsymbol{\theta})]$) has the same notion. We assume this objective function to be differentiable w.r.t. parameter vector.

Moment estimates:

Gradient of a scalar function is a vector, and it is given by $\boldsymbol{g}_k = \boldsymbol{\nabla_\theta f_k}(\boldsymbol{\theta}_{k-1})$**,** this is derivative of objective functions realization at kth step w.r.t. parameter vector, evaluated at parameter values at k-1th step. As our objective function is scalar, so is the gradient. Gradients follow some underlying distribution ($p(\boldsymbol{g_k})$). We need to estimate mean of this distribution. For this Adam uses exponential moving average estimate of gradient, given by.

$\boldsymbol{m}_k \leftarrow \beta_1 \boldsymbol{m}_{k-1} + (1-\beta_1)\boldsymbol{g}_k$ here, β_1 controls the exponential decay. But this estimate is biased towards zero because of zero initialization as shown below.

$\boldsymbol{m}_1 = \beta_1 \boldsymbol{m}_0 + (1-\beta_1)\boldsymbol{g}_1 = (1-\beta_1)\boldsymbol{g}_1$ Because $\boldsymbol{m}_0 = \mathbf{0}$

$\boldsymbol{m}_2 = \beta_1 \boldsymbol{m}_1 + (1-\beta_1)\boldsymbol{g}_2 = \beta_1(1-\beta_1)\boldsymbol{g}_1 + (1-\beta_1)\boldsymbol{g}_2$

$\boldsymbol{m}_3 = \beta_1 \boldsymbol{m}_2 + (1-\beta_1)\boldsymbol{g}_3 = {\beta_1}^2(1-\beta_1)\boldsymbol{g}_1 + \beta_1(1-\beta_1)\boldsymbol{g}_2 + (1-\beta_1)\boldsymbol{g}_3$

$$\boldsymbol{m}_k = (1-\beta_1)\sum_{i=1}^{k} {\beta_1}^{k-i}\boldsymbol{g}_i$$

If we take expectation on both sides and use linearity property of expectation with assumption that $\boldsymbol{g}_i$ are independent and identically distributed , we get

$$\boldsymbol{E}[\boldsymbol{m}_k] = (1-\beta_1)\sum_{i=1}^{k} {\beta_1}^{k-i}E[\boldsymbol{g}_i] = (1-\beta_1)(1+\beta_1+\beta_1^2+..+\beta_1^{k-1})E[\boldsymbol{g}] = \left(1-{\beta_1}^k\right)E[\boldsymbol{g}]$$

We can see that estimate is biased, especially at start, estimate tends to be zero. To correct this,

$\widehat{\boldsymbol{m}_k} = \boldsymbol{m}_k/\left(1-\beta_1^k\right)$ is done, so that, $\widehat{\boldsymbol{m}_k}$ is unbiased. As shown below.

$E[\widehat{\boldsymbol{m}_k}] = \boldsymbol{E}[\boldsymbol{m}_k]/\left(1-\beta_1^k\right) = \left(1-{\beta_1}^k\right)E[\boldsymbol{g}]/\left(1-\beta_1^k\right) = E[\boldsymbol{g}]$.

Similarly, estimate of second moment of gradient ($\boldsymbol{g^2}$)(elementwise square) is calculated as

$\boldsymbol{v}_k = \beta_2\boldsymbol{v}_{k-1} + (1-\beta_2)\boldsymbol{g}_k^2$, This can also be termed as uncentered variation. And this is also biased and has similar bias correction as of first moment estimate. Derivation for this is exactly same as of derivation given above for first moment estimate.

Parameter update ($\Delta_{\boldsymbol{k}} \stackrel{\text{def}}{=} \alpha\ \widehat{\boldsymbol{m}_k}/\left(\sqrt{\widehat{\boldsymbol{v}_k}}+\epsilon\right)$):

Efficiency of algorithm can be improved by replacing last 3 lines in while loop by following,

$$\widehat{\boldsymbol{m}_k} \leftarrow \boldsymbol{m}_k/\left(1-\beta_1^k\right)$$
$$\widehat{\boldsymbol{v}_k} \leftarrow \boldsymbol{v}_k/\left(1-\beta_2^k\right)$$
$$\boldsymbol{\theta_k} \leftarrow \boldsymbol{\theta}_{k-1} - \alpha\ \widehat{\boldsymbol{m}_k}/\left(\sqrt{\widehat{\boldsymbol{v}_k}}+\epsilon\right)$$

$\Longrightarrow$

$$\alpha_k \leftarrow \alpha\sqrt{\left(1-\beta_2^k\right)}\,/\,\left(1-\beta_1^k\right)$$
$$\boldsymbol{\theta_k} \leftarrow \boldsymbol{\theta}_{k-1} - \alpha_k\ \boldsymbol{m}_k/\left(\sqrt{\boldsymbol{v}_k}+\epsilon\right)$$

We can see that both are equivalent as shown below,

$$\frac{\alpha_k\ \boldsymbol{m}_k}{\left(\sqrt{\boldsymbol{v}_k}+\epsilon\right)} = \frac{\alpha\ \boldsymbol{m}_k\sqrt{\left(1-\beta_2^k\right)}}{\left(1-\beta_1^k\right)\left(\sqrt{\boldsymbol{v}_k}+\epsilon\right)} = \frac{\alpha\ \widehat{\boldsymbol{m}_k}}{\left(\sqrt{\widehat{\boldsymbol{v}_k}}+\epsilon\right)}$$

Important property of Adam's update rule is its careful choice of parameter update size ($\Delta_{\boldsymbol{k}}$). It can be shown that parameter update sizes at each timestep are approximately bounded by step size hyperparameter (α), i.e., $|\Delta_k| \leq \alpha$

This can be understood as establishing a trust region around the current parameter value, beyond which the current gradient estimate does not provide sufficient information. Many times, we have a prior knowledge of parameter distribution, since α sets upper bound on magnitude of parameter update, right order of magnitude of α can be deduced such that optima can be reached from initial guess in some number of iterations.

Suppose $\sqrt{\widehat{\boldsymbol{v}_k}}$ is larger then $\Delta_{\boldsymbol{k}}$ will be smaller, which is desirable because second moment is also uncentered variance which signifies uncertainty in the direction to move and hence smaller update size is desired. And it also provides natural annealing because as $\widehat{\boldsymbol{m}_k}/\left(\sqrt{\widehat{\boldsymbol{v}_k}}\right)$ tends to zero as we reach towards optimum.

Parameter update ($\Delta_{\boldsymbol{k}}$) is invariant to scale of gradients, rescaling the gradients ($\boldsymbol{g}_k$) with factor c will scale $\widehat{\boldsymbol{m}_k}$ with a factor c and $\widehat{\boldsymbol{v}_k}$ with a factor c^2, which cancels out and update remains unchanged.

Convergence in online convex regime: [1]

Here we start by defining regret at iteration T:

$$R(T) = \sum_{t=1}^{T} [L_t(W_t) - L_t(W^*)]$$

Where: $W^* = \arg_w min \sum_{t=1}^{T} L_t(W)$

In convex regime, Adam gives regret bound comparable to best known:

If all batch objectives $L_t(W)$ are convex and have bounded gradients, and all points W_t generated by Adam are within bounded distance from each other, then for every $T \in \mathbb{N}$: Adam achieves the following guarantee, for all $T \geq 1$.

$$\frac{R(T)}{T} = O(\frac{1}{\sqrt{T}})$$

$$\lim_{T \to \infty} \frac{R(T)}{T} = 0$$

Thus, above equation states that, Adam reaches close to minima of loss function (in relative sense with respect to iterations) at large number of iterations.

Advantages of Adam [1] [3]**:**

- Adam is computationally efficient.
- Requires less memory.
- Does not require a stationary objective function. (function might change with time for a non-stationary objective function)
- Works with sparse gradients.
- Naturally performs step annealing, i.e., step size decreases after few epochs automatically.
- Magnitude of parameter update is bounded by step size hyperparameter.

Conclusion: [1]

- In case of logistic regression, Adam converges as fast as Adagrad. Like Adagrad, Adam can take advantage of sparse features and obtain faster convergence rate that normal SGD with momentum.
- Multi-layer neural network are powerful models with non-convex objective functions. In such cases, Adam shows better convergence than other methods.
- In case of CNNs, in experiment of MNIST dataset, Adagrad converges much slower than others in this experiment. Though Adam shows marginal improvement over SGD with momentum, it adapts learning rate scale for different layers instead of hand picking manually as in SGD.

Implementation of Adam to train a deep neural network for handwritten digit recognition.

A convolution neural network is trained on MNIST dataset by using Adam as optimizer. All the parameters of Adam are set to default, and K fold cross validation is used to check overfitting and then the trained model of last fold is saved and used for prediction.

Link of the files : Handwritten Image recognition

References

[1] J. B. Diederik P. Kingma, "Adam: A Method for Stochastic Optimization," in *3rd International Conference for Learning Representations,,* San Diego, 2015. https://doi.org/10.48550/arXiv.1412.6980

[2] J. Brownlee, "Gentle Introduction to the Adam Optimization Algorithm for Deep Learning," [Online]. Available: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/.

[3] N. Nikhil, "Everything you need to know about Adam Optimizer," [Online]. Available: https://medium.com/@nishantnikhil/adam-optimizer-notes-ddac4fd7218#:~:text=Parameters%20update%20are%20invariant%20to,step%2Dsize%20hyper%2Dparameter..