IET Image Processing

Research Article

IET Journals
The Institution of Engineering and Technology

Accelerating convolutional neural network training using ProMoD backpropagation algorithm

ISSN 1751-9659
Received on 19th July 2019
Revised 6th January 2020
Accepted on 26th February 2020
E-First on 5th October 2020
doi: 10.1049/iet-ipr.2019.0761
www.ietdl.org

Ahmet Gürhanlı[1] ✉

[1]*Department of Computer Engineering, Istanbul Aydin University, Beşyol Mah. İnönü Cad. No: 38 Sefaköy–Küçükçekmece, Istanbul, Turkey*
✉ *E-mail: ahmetgurhanli@aydin.edu.tr*

Abstract: Convolutional neural networks (CNNs) play an important role in image recognition applications. Fast training of image recognition systems is a crucial point, because the system should be trained for each new image class. These networks are trained using lengthy calculations. Focus of engineering is on obtaining a fast, but stable optimisation method. Momentum technique which is used in backpropagation algorithms is like a proportional–integral (PI) controller that is widely employed in automatic control systems. It takes the integral of past errors and helps reaching the training targets. Proportional + momentum + derivative (ProMoD) method adds gradient of update matrices to the training process and builds an optimiser such as the widely used PI–derivative controller. The method accelerates the movement toward the target accuracy levels. This is achieved by doing bigger corrections in the beginning using the differences in the calculated update matrices. In this research, ProMoD method is tested on image recognition applications and CNNs. Modified national institute of standards and technology database (MNIST) and Fashion-MNIST datasets are used for evaluating the performance. Experimental results showed that ProMoD might perform much faster in training of CNNs and consume proportionally less power with respect to the momentum and stochastic gradient descent (SGD) techniques.

1 Introduction

Machine-learning applications require fast and stable training strategies. Plenty of optimisation approaches have been developed and evaluated for artificial neural networks. Stochastic gradient descent is at the heart of most of the optimisation methods. It aims to find the global minimum of a cost function that measures the error level of the neural network. Optimisation is done on the weight values of the neural network. It is not that straightforward, because the number of weight matrices and size of these matrices may be high in a deep network. So, an optimal result is searched using a plenty of variables.

There are numerous alternatives that can be chosen for the gradient descent technique. Batch gradient descent evaluates the cost function for the whole dataset and makes one update to the weight values. On the contrary, stochastic gradient descent changes the weights for each member of the training set. Minibatch gradient descent is a trade-off between these two methods. It calculates the average error of the network for a subgroup of the training set. It aims both a stable convergence and a fast learning. In image recognition applications, minibatch is the preferred choice.

Momentum technique uses two terms for optimisation. First term is obtained directly from backpropagated error values. Second term is a scaled version of the update term in the previous iteration. Second term keeps the history of updates and ensures moving toward a global minimum of the cost function.

Several methods have been proposed for adding a third term to the optimisation rules. However, a term that would use the difference between current update matrix and the one in the previous iteration has not been evaluated. Proportional + momentum + derivative (ProMoD) takes that difference as the third term and builds an optimisation method truly like the proportional–integral–derivative (PID) controller.

Using the derivative action, we have a third control term to affect the speed of learning, besides PI actions. The difference between consecutive update matrices is expected to be larger in the beginning of the training and that difference approaches zero toward the end of the process. So, a scaled version of this difference can be used together with PI actions to accelerate the training. This difference is used to make bigger updates in the initial iterations. That is how ProMoD contributes to the speed of training. ProMoD is presented in our previous papers using simple logic functions. This paper explains how to use ProMoD algorithm in image recognition, and evaluates the results using widely adopted datasets.

2 Literature review

Ruder [1] provides a good summary of gradient descent optimisation techniques in the literature. Several parameters should be calibrated very carefully when using a minibatch trainer. Learning rate is an important feature that needs good adjustment. Speed and stability concerns should be handled together. Local minimums should not be considered as the final global minimum. In the literature, it is possible to find plenty of solutions targeting a fast and stable learning. Momentum method [2] adds the errors cumulatively, and the error summation plays a bigger role when determining the weight updates when compared with immediate output error. Nesterov accelerated gradient [3] focuses on making a prediction for the future behaviour. This way an accelerated movement toward the target can be achieved. Adagard [4] aims the problem arising from infrequent parameters and makes bigger updates for them. Adadelta [5] is related to Adagard and works on a certain number of past gradients. Adaptive moment estimation (Adam) [6] employs adaptive learning rates by recording a decaying average of past gradients. Nesterov [7] proposed a technique that solves a convex problem. Nadam [8] is an improvement over Adam and Nesterov. Prior to gradient calculation, it changes the parameters using the momentum step. Sutskever [9] focused on initialisation methods for improving deep learning algorithms. Adaptive methods have been evaluated by several researchers [10–12]. Martens [13] suggested an approach that employs Hessian-free optimisation. Dauphin [14] dealt with the saddle point problem. Pascanu performed a research on natural gradients [15]. Krylov subspace descent has been evaluated by Vinyals [16]. A method called RMSProp [17] works well in online and non-stationary situations. Several other approaches consider finding the global minimum of a cost function for a neural network [18–21]. Various techniques resulting from control theory have been proposed in several research [22–24]. Zeraatkar used a quasi-

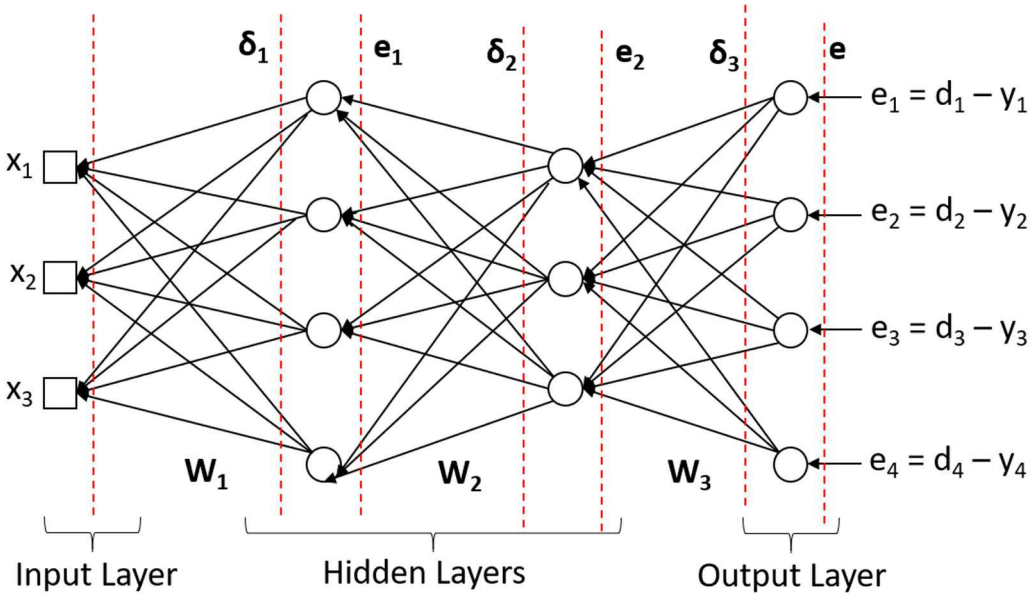

Fig. 1 *Backpropagation on a neural network*

PID approach, in that work, the performance is evaluated using only exclusive OR and 3 bit parity applications [25]. Derivation and integration is done using the output error of a neuron. PID-based methods are expected to possess a substantial position in deep learning in the future.

ProMoD [26] method is also based on PID control, which is a very important technique in industrial applications. ProMoD method adds derivative action to PI actions that are present in the famous momentum technique. Its aim to accelerate the learning rate by applying bigger updates in the beginning of the training process.

ProMoD is very similar to the quasi-PID algorithm developed by Zenaatkar, but not the same [25]. Zenaatkar uses the error values at the output of a neuron to obtain the PID terms. However, ProMoD uses the update matrices instead of output errors to calculate the three terms. Three-term backpropagation algorithm developed by Zweiri takes the difference between the output and target at each iteration and adds this as a third term to the classical momentum method [22]. So, to our knowledge, ProMoD is the first algorithm that takes the integral and derivative of proportional weight update matrices obtained from backpropagated error values [u_t in (16)] and uses them in calculation of final parameter update matrix. The integral parameter β in this study has different strengths compared with the momentum parameter γ in momentum technique [1]. Since β in ProMoD is multiplied with the integral of proportional update matrices. However, γ in momentum is multiplied by the update matrix of the previous iteration. So, different tuning values are needed for the mentioned parameters.

This paper, whose initial abstract was included in the proceedings of ARICMTC Istanbul 2019 [27], reports the results of employing ProMoD method in image recognition applications. ProMoD was first introduced in the conference IBICA 2018, without using the name 'ProMoD'. This first introduction is based on learning a simple logic function and did not include any research on image recognition and convolutional neural networks (CNNs). An improved version of this conference paper about learning logic functions is under review in a journal. This paper reports the usage of ProMoD in image recognition and CNNs. A CNN initially using the momentum method has been served as the basis. Initial code is taken from Kim's book [28], and the optimisation algorithm is changed as proposed by the ProMoD method. The results are compared with the famous SGD and momentum techniques.

3 Background: ProMoD backpropagation algorithm

A neural network is illustrated in Fig. 1. Input layer is only for taking the input data and no operation is performed here. Fig. 1 shows the backpropagation phase, but when doing the classification task, data flows from left to right [28]. Input vector is multiplied by $\boldsymbol{W}_1$ matrix. This produces the input vector for the activation function. Activation function is a non-linear operator. Its non-linear feature makes adding more layers to the network meaningful. The output of the activation function produces the vector for the next layer. This operation continues until reaching the output layer.

When training the network using a version of the backpropagation algorithm, first, the differences between correct results and obtained ones are calculated. These differences are called as errors and backpropagated into the network. The weight matrices lie between the layer output vectors and delta vectors, which reflect some information about the errors at the next layer. The error at the output of a node is obtained by multiplying the transpose of the weight matrix and backpropagated delta of the next node. Next task is passing the error vector back through the activation function. This is done by taking the derivative of the activation function with respect to the input vector at that node and then taking Hadamard product of the result vector with the error vector. The result of this Hadamard product is called as the delta vector.

So, the update matrix of each layer is calculated by multiplying the delta vector, which represents the errors at the layer output with the transpose of the layer input matrix.

Minibatch method does not update the weight matrices for each of the training set members. A batch size is determined, and the average weight update is calculated for that preferred batch size. This way the training process becomes more stable. Minibatch ProMoD algorithm, which is a modification of the backpropagation algorithm provided in Kim's book [28], can be explained as below [29]:

1. Set initial values of weight matrices. Set the integral matrices $\boldsymbol{I}_k$ and previous proportional update matrices $\boldsymbol{P}_k^-$ as zero for each layer.
2. Set weight update totals (**WUTs**) as zero

$$\mathbf{WUT}_k \leftarrow \mathbf{0} \tag{1}$$

3. Get an input from the dataset.
4. Obtain the output $\boldsymbol{y}_o$.
5. Produce the error $\boldsymbol{e}_o$ by subtracting $\boldsymbol{y}_o$ from the correct output $\boldsymbol{d}_o$ (subscript 'o' stands for output layer)

$$\boldsymbol{e}_o \leftarrow \boldsymbol{d}_o - \boldsymbol{y}_o \tag{2}$$

6. Obtain the delta ($\boldsymbol{\delta}_o$) for the output layer. This is done by taking the derivative of the activation function with respect to the input vector at that node and then taking Hadamard product of the result vector with the error vector

$$\boldsymbol{\delta}_o \leftarrow \varphi'(\boldsymbol{v}_o) \odot \boldsymbol{e}_o \tag{3}$$

7. Using the output node delta, calculate the deltas for other nodes. First, obtain error vector $\boldsymbol{e}_k$ by multiplying the transpose of weight matrix of next layer with the delta of next layer. Calculate the delta the same way used for the output node. Produce the weight update matrices ($\mathbf{WU}_k$) for all layers by multiplying delta and transpose of input vector of that layer $\boldsymbol{x}_k$. Add the calculated weight updates to the total weight updates ($\mathbf{WUT}_k$) for each layer k

$$\boldsymbol{e}_k \leftarrow \boldsymbol{W}_{k+1}^{\mathrm{T}} \boldsymbol{\delta}_{k+1} \tag{4}$$

$$\boldsymbol{\delta}_k \leftarrow \varphi'(\boldsymbol{v}_k) \odot \boldsymbol{e}_k \tag{5}$$

$$\mathbf{WU}_k \leftarrow \boldsymbol{\delta}_k \boldsymbol{x}_k^{\mathrm{T}} \tag{6}$$

$$\mathbf{WUT}_k \leftarrow \mathbf{WUT}_k + \mathbf{WU}_k \tag{7}$$

8. Repeat 7 for all hidden layers.
9. Repeat 3–8 for number of items determined by the batch size.
10. Obtain the average weight update matrix

$$\mathbf{WU}_k \leftarrow \mathbf{WUT}_k/\mathrm{Batch\_Size} \tag{8}$$

11. Update the weight matrices using the ProMoD method.
    a. Obtain the proportional update

$$\boldsymbol{P}_k \leftarrow \mathbf{WU}_k \tag{9}$$

    b. Obtain the integral update

$$\boldsymbol{I}_k \leftarrow \boldsymbol{I}_k + \boldsymbol{P}_k \tag{10}$$

    c. Obtain the derivative update

$$\boldsymbol{D}_k \leftarrow \boldsymbol{P}_k - \boldsymbol{P}_{\bar{k}} \tag{11}$$

$$\boldsymbol{P}_{\bar{k}} \leftarrow \boldsymbol{P}_k \tag{12}$$

    d. Multiply PID terms with their impact coefficients (α, β and γ, respectively) and get the final update matrix for each layer. Add this update to the current weight matrix

$$\Delta \boldsymbol{W}_k \leftarrow \alpha \boldsymbol{P}_k + \beta \boldsymbol{I}_k + \gamma \boldsymbol{D}_k \tag{13}$$

$$\boldsymbol{W}_k \leftarrow \boldsymbol{W}_k + \Delta \boldsymbol{W}_k \tag{14}$$

12. Repeat 2–11 for all batches; terminate the algorithm if desired error level is reached.
13. Repeat 2–12 until desired error level is reached.

As seen in the algorithm, the weight updates for a batch is calculated for each layer in the artificial neural network. Then, using the weight update of each layer, PID terms are obtained and used to change the weights in the system. Mathematical update rule of the ProMoD is given below:

$$\boldsymbol{u}_t = \boldsymbol{B}\left(\boldsymbol{J}\left(\boldsymbol{\theta}; \boldsymbol{x}^{(i:i+n)}; \boldsymbol{y}^{(i:i+n)}\right)\right) \tag{15}$$

$$\boldsymbol{v}_t = \alpha \boldsymbol{u}_t + \beta \sum_0^t \boldsymbol{u}_t + \gamma(\boldsymbol{u}_t - \boldsymbol{u}_{t-1}) \tag{16}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \boldsymbol{v}_t \tag{17}$$

In (15), θ represents the parameters to be optimised. J is the cost function whose parameters are weight matrix entries θ, the input vector $\boldsymbol{x}$ in a batch from i to $i+n$ and correct output vector $\boldsymbol{y}$ for that batch. B is the backpropagation function, which takes the cost function result of the output node and reflects it as a proportional update matrix $\boldsymbol{u}_t$ to each layer in the network using backpropagation equations described in the ProMoD algorithm, where t represents the time step. Equation (16) shows how PID terms are obtained from matrix $\boldsymbol{u}_t$. α, β and γ are coefficients used to adjust the effect of each term. Calculated final update matrix $\boldsymbol{v}_t$ is added to the parameter matrix θ in order to calculate the updated parameters as shown in (17).

4 Image recognition using the ProMoD algorithm

CNNs are widely used in image recognition applications. Fig. 2 illustrates how a CNN works. Classifier network is an artificial neural network as in Fig. 1. The image to be classified is passed through a feature extraction network before being fed to the classifier. Deep learning techniques use automated feature extractors. Mathematical models that aim to extract special features are not applied in convolution filters. The filter values are trained together with the weight matrices of the classifier network using backpropagation.

Convolution and pooling operations form a layer in the feature extractor and this pair can be chained one after another. Number of pairs that will give accurate and fast results may change from application to application.

First, a convolution filter scans the input image. At each stop, a portion of the input image matrix is multiplied with a convolution filter. This operation is repeated for all convolution filters and several features are extracted from the image.

The pooling layer aims to overcome the overfitting problem by reducing the matrix size of convolution outputs. This can be done using several methods. Taking the average of neighbour elements is widely utilised. Another method is taking the maximum value in a region. Pooling layer does not contain a matrix or filter to be trained. The feature extractor that is used in this research has only one layer. It contains 20 convolution filters. These 20 filters are combined into a three-dimensional (3D) array when being trained using backpropagation. The ProMoD algorithm is applied to this

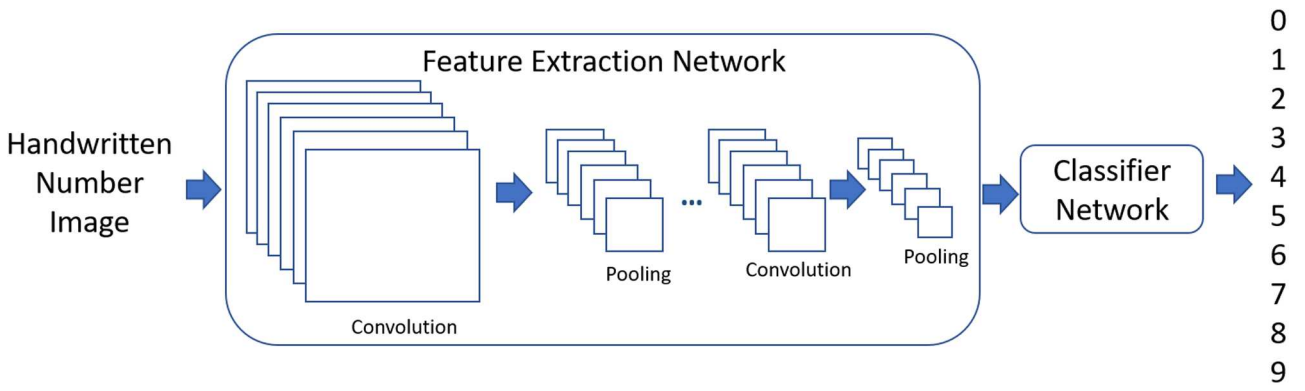

Fig. 2 *Image recognition using CNNs*

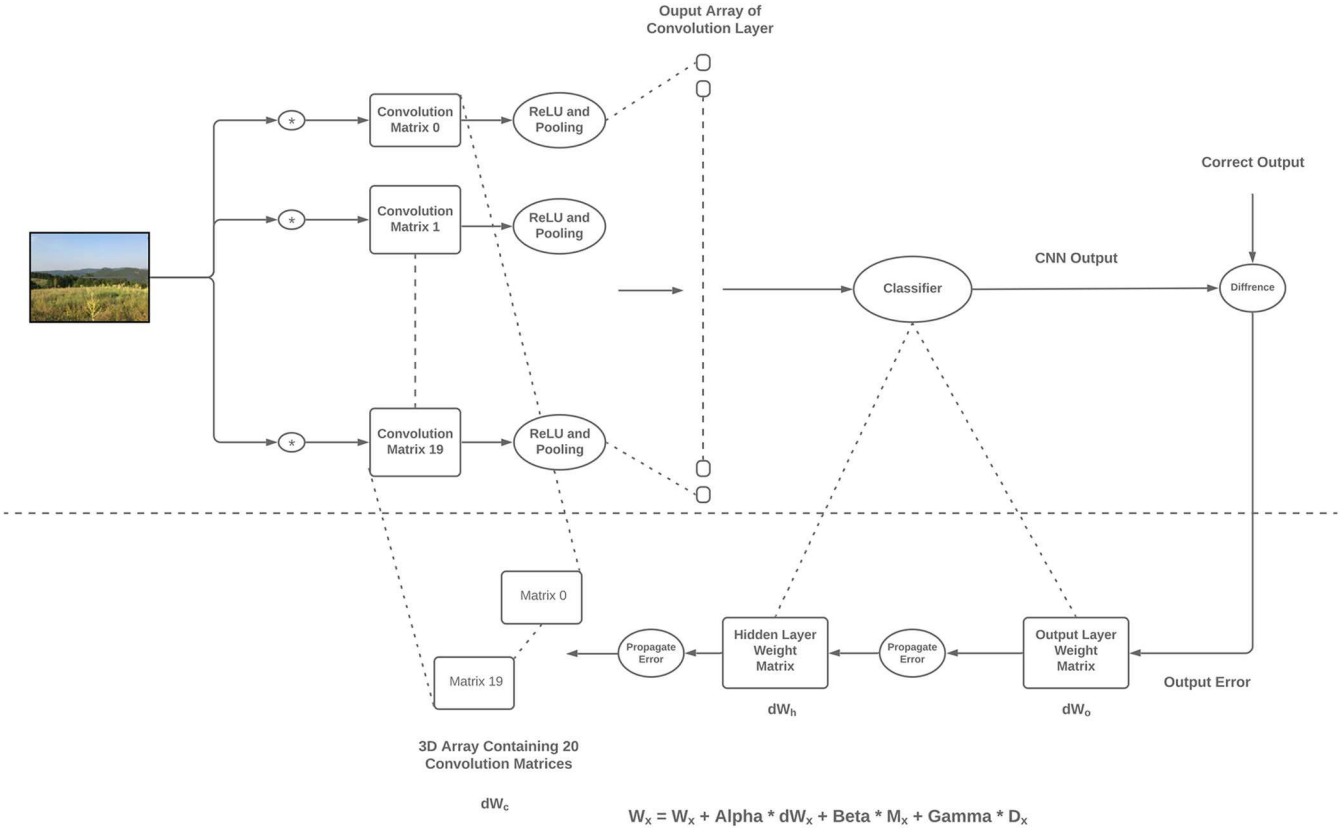

Fig. 3 *Forward pass and backpropagation in a CNN*

Table 1 Calibration values used for PID actions in three methods

	SGD	Momentum	ProMoD
proportional coefficient (α)	0.12	0.05	0.2
integral coefficient (β)	NA	0.75	0.015
derivative coefficient (γ)	NA	NA	0.04

Table 2 Training error values for first 138 batches and final accuracies after 300 batches of SGD, momentum and ProMoD algorithms for MNIST dataset

Batch number	SGD	Momentum	ProMoD
1	0.5711	0.5711	0.5711
2	0.5692	0.5699	0.5684
3	0.5680	0.5692	0.5650
4	0.5661	0.5680	0.5600
5	0.5657	0.5684	0.5510
6	0.5628	0.5661	0.5397
7	0.5563	0.5614	0.5192
8	0.5497	0.5562	0.5433
9	0.5358	0.5477	0.5420
10	0.5216	0.5349	0.4706
11	0.4991	0.5139	0.3717
12	0.5000	0.4918	0.3417
13	0.4665	0.4617	0.4275
14	0.4380	0.4192	0.4487
15	0.4109	0.3631	0.4469
16	0.4064	0.3660	0.4765
17	0.4653	0.3652	0.3438
18	0.3915	0.3519	0.3429
19	0.3751	0.2599	0.2514
20	0.3294	0.3308	0.2475
21	0.3062	0.2243	0.2063
22	0.3020	0.2582	0.2401

3D array, which contains all the convolution filters as illustrated in Fig. 3.

The classifier used in this work contains one hidden layer and one output layer. The weight matrices of these two layers are also trained by the ProMoD algorithm.

5 Experimental setup

MNIST dataset which contains 70,000 handwritten numbers and Fashion-MNIST dataset which contains 70,000 images of wearable items such as trousers, sandals etc. are used to evaluate the appropriateness of ProMoD algorithm for image recognition applications. About 60,000 images are used for training and 10,000 separate images are used for testing. Test images are not used when training the network.

Initial code is taken from Kim's book [28]. The code is written in Python language and it is using the library NUMPY for mathematical calculations. Datasets are taken from KERAS. SPYDER is used as Python IDE.

Originally, momentum approach is used for training the convolution matrices and classifier matrices. The trainer has been modified to implement the SGD and ProMoD algorithms. The training speeds of three approaches have been compared. The calibration values used for PID parameters used in three methods are given in Table 1. It should be noted that the update rule of momentum method is different for integral term and so it uses a different coefficient for that term.

6 Experimental results

Table 2 and Fig. 4 show the result of training using MNIST dataset. Table 2 shows the results until batch number 138 because of space

Batch number	SGD	Momentum	ProMoD
23	0.2971	0.1610	0.1806
24	0.3197	0.2003	0.1907
25	0.3195	0.2453	0.1768
26	0.3137	0.2153	0.1943
27	0.2724	0.1797	0.1794
28	0.2467	0.1600	0.1406
29	0.2628	0.1816	0.1371
30	0.2475	0.1731	0.1142
31	0.2310	0.1248	0.0916
32	0.3232	0.1413	0.1276
33	0.2559	0.1329	0.1352
34	0.2569	0.1499	0.1048
35	0.2553	0.1809	0.1489
36	0.2379	0.1500	0.1246
37	0.2649	0.2115	0.1574
38	0.2170	0.1398	0.0943
39	0.2700	0.1924	0.1541
40	0.2387	0.1636	0.1579
41	0.2618	0.1610	0.0911
42	0.2258	0.1718	0.0919
43	0.1777	0.1408	0.0867
44	0.2311	0.1938	0.1374
45	0.2300	0.1516	0.1144
46	0.2054	0.1417	0.0877
47	0.1884	0.1549	0.0908
48	0.1645	0.1439	0.0857
49	0.1491	0.1373	0.0825
50	0.1298	0.1065	0.0596
51	0.1569	0.1377	0.0846
52	0.1319	0.1523	0.1219
53	0.1420	0.1214	0.0790
54	0.1872	0.1658	0.0824
55	0.1343	0.1150	0.0549
56	0.1470	0.1202	0.0576
57	0.1568	0.1287	0.0768
58	0.1701	0.1373	0.0795
59	0.1783	0.1513	0.1005
60	0.1593	0.1275	0.0801
61	0.1596	0.1193	0.0762
62	0.1821	0.1496	0.0889
63	0.1809	0.1573	0.0945
64	0.1804	0.1609	0.0988
65	0.1759	0.1577	0.0750
66	0.1819	0.1714	0.1057
67	0.1524	0.1235	0.0595
68	0.1148	0.1006	0.0456
69	0.1690	0.1607	0.0840
70	0.1604	0.1541	0.0792
71	0.1544	0.1445	0.0764
72	0.1735	0.1537	0.0935
73	0.1598	0.1350	0.0619
74	0.2150	0.2122	0.1438
75	0.1612	0.1432	0.0793
76	0.1229	0.1337	0.0651
77	0.1255	0.1152	0.0577
78	0.1281	0.1020	0.0559
79	0.1608	0.1380	0.0733
80	0.1675	0.1275	0.0750
81	0.1622	0.1221	0.0602
82	0.1498	0.0953	0.0387
83	0.1213	0.0882	0.0426
84	0.1370	0.1243	0.0528
85	0.1577	0.1298	0.0661
86	0.1327	0.1114	0.0558
87	0.1137	0.0960	0.0411
88	0.1519	0.1325	0.0589
89	0.1616	0.1187	0.0582
90	0.1446	0.1156	0.0713
91	0.0980	0.0705	0.0426
92	0.1034	0.0807	0.0326
93	0.1299	0.0934	0.0484
94	0.1028	0.0797	0.0363
95	0.0978	0.0751	0.0281
96	0.1421	0.1172	0.0492
97	0.1348	0.0876	0.0484
98	0.1365	0.0953	0.0375
99	0.0795	0.0556	0.0257
100	0.1091	0.0979	0.0483
101	0.1386	0.1230	0.0626
102	0.1130	0.0866	0.0505
103	0.0869	0.0756	0.0337
104	0.1208	0.1024	0.0614
105	0.1543	0.1092	0.0485
106	0.1035	0.0930	0.0513
107	0.1182	0.1229	0.0434
108	0.1240	0.0972	0.0397
109	0.0998	0.0847	0.0333
110	0.0726	0.0801	0.0407
111	0.1245	0.1113	0.0645
112	0.0991	0.0803	0.0391
113	0.1701	0.1514	0.0609
114	0.1247	0.1012	0.0635
115	0.1060	0.0633	0.0284
116	0.1056	0.1064	0.0399
117	0.0998	0.0670	0.0257
118	0.1070	0.0845	0.0353
119	0.1205	0.1023	0.0464
120	0.1198	0.0947	0.0268
121	0.1128	0.0876	0.0345
122	0.1178	0.0997	0.0350
123	0.1217	0.1031	0.0559
124	0.1635	0.1393	0.0648
125	0.1882	0.1088	0.0578
126	0.1474	0.0772	0.0329
127	0.1160	0.0761	0.0378
128	0.0933	0.0698	0.0272
129	0.0881	0.0705	0.0434
130	0.1174	0.0944	0.0544
131	0.0788	0.0588	0.0269
132	0.1026	0.0798	0.0382
133	0.1192	0.0965	0.0493
134	0.1422	0.1223	0.0686
135	0.1244	0.0770	0.0396
136	0.1175	0.0890	0.0517
137	0.1021	0.0876	0.0323
138	0.0951	0.0819	0.0322
accuracy after 300 batches, %	94.87	95.93	96.36

considerations, the test is run until 300 batches and the batch size is taken as 200 images. Absolute values of output nodes' errors are accumulated for a batch, and after the batch is processed the norm

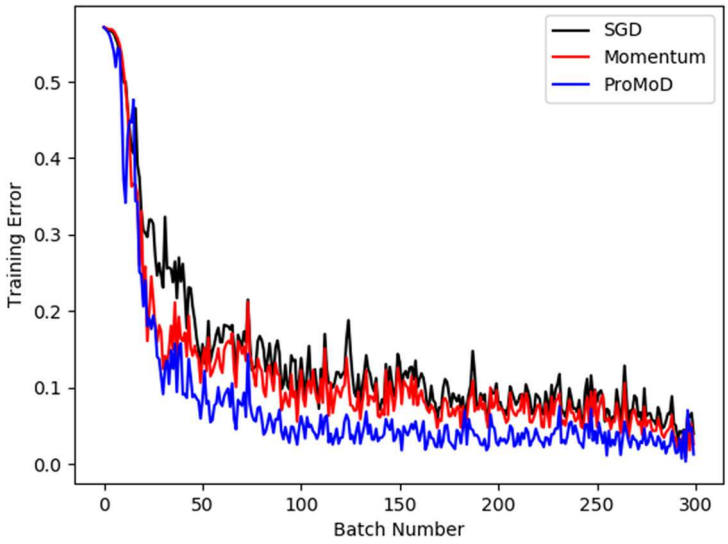

Fig. 4 *Progress of training for SGD, momentum and ProMoD methods for MNIST dataset*

of the accumulated error vector is taken and divided by batch size and the result is used as the cost function. Table 2 and Fig. 4 depict the progress of that cost function. Final accuracies that are obtained using test dataset are 94.87, 95.93 and 96.36% for SGD, momentum and ProMoD methods, respectively. Since the worst accuracy is 94.59%, error level of 0.0541 (1–0.9459) is taken as threshold for evaluating the training speed of the algorithms. The batch number that first gives that error level or below is taken as the required number of batches to reach the desired accuracy and marked with yellow in Table 2. Required batches to be processed are 267, 170 and 69 for SGD, momentum and ProMoD methods, respectively. So, ProMoD reached that level 3.87 times faster than SGD and 2.46 times faster than momentum methods.

Table 3 and Fig. 5 show the result of training using Fashion-MNIST dataset. Accuracies after processing 300 batches are 76.9, 79.62 and 82.58% for SGD, momentum and ProMoD methods, respectively. Owing to the worst accuracy, the error threshold is taken as 0.231. Required batches to be processed to reach that error level are 232, 104 and 70 for SGD, momentum and ProMoD methods, respectively. According to Fashion-MNIST, ProMoD reached the desired level 3.31 times faster than SGD and 1.49 times faster than momentum.

Table 4 depicts a comparison between ProMoD and momentum using several important aspects. We have seen that ProMoD is faster in training and final accuracies settle at very close levels. Stability of ProMoD is more sensitive on batch size; this issue is subject of another research. Stability of momentum depends on PI actions, besides these two terms the derivative action affects stability of the ProMoD method, as well.

7 Conclusion

ProMoD backpropagation algorithm, which adds derivative action to the well known momentum method, has been tested in an image recognition application. MNIST and Fashion-MNIST datasets are used for testing the efficiency of the algorithm. It has been observed that for MNIST dataset, ProMoD achieved reaching the desired accuracy levels much faster than the momentum and SGD method. ProMoD reached the desired error level 3.87 times faster than SGD and 2.46 times faster than momentum method when tested using MNIST dataset. Owing to Fashion-MNIST test ProMoD performed 3.31 times faster than SGD and 1.49 times faster than momentum. Several aspects of the ProMoD algorithm, such as the batch size, calibration of PID parameters are subjects of future research. Comparison with adaptive algorithms will also be investigated in future works.

Table 3 Training error values for first 138 batches and final accuracies after 300 batches of SGD, momentum and ProMoD algorithms for Fashion-MNIST dataset

Batch number	SGD	Momentum	ProMoD
1	0.5698	0.5698	0.5698
2	0.5690	0.5697	0.5680
3	0.5651	0.5670	0.5622
4	0.5606	0.5652	0.5432
5	0.5540	0.5596	0.5473
6	0.5496	0.5539	0.5658
7	0.5369	0.5437	0.5510
8	0.5161	0.5135	0.4798
9	0.4821	0.4629	0.5296
10	0.5322	0.4321	0.5467
11	0.5334	0.4345	0.5059
12	0.5183	0.6001	0.4737
13	0.5007	0.4931	0.5276
14	0.4704	0.4848	0.5660
15	0.4998	0.5606	0.5584
16	0.4970	0.5612	0.5238
17	0.4508	0.5613	0.5025
18	0.3991	0.5569	0.4566
19	0.4171	0.5465	0.4463
20	0.4836	0.5236	0.4747
21	0.4981	0.4881	0.5044
22	0.4525	0.4742	0.5301
23	0.4599	0.4567	0.4824
24	0.3497	0.3971	0.4377
25	0.3380	0.3674	0.4119
26	0.4111	0.3698	0.4249
27	0.4178	0.3646	0.4046
28	0.4249	0.3659	0.3964
29	0.3648	0.3208	0.3530
30	0.3702	0.3425	0.3524
31	0.3574	0.3005	0.3793
32	0.3840	0.3135	0.4224
33	0.3737	0.3304	0.4389
34	0.3593	0.3156	0.3928
35	0.3335	0.3002	0.3609
36	0.3282	0.3042	0.3330
37	0.3495	0.3256	0.3451
38	0.3743	0.3214	0.3441

Batch number	SGD	Momentum	ProMoD
39	0.3785	0.3583	0.3412
40	0.3583	0.3114	0.3461
41	0.3853	0.3277	0.3479
42	0.3424	0.3705	0.3470
43	0.3327	0.3301	0.3346
44	0.3254	0.3374	0.3479
45	0.3485	0.3401	0.3196
46	0.3723	0.3118	0.2995
47	0.3238	0.2810	0.2811
48	0.3416	0.3125	0.3068
49	0.3530	0.3220	0.3535
50	0.3700	0.3190	0.3429
51	0.3171	0.2629	0.2721
52	0.3366	0.3076	0.2878
53	0.2936	0.2508	0.2629
54	0.3066	0.2879	0.2777
55	0.3371	0.2708	0.2699
56	0.3083	0.2758	0.2476
57	0.3466	0.2754	0.2596
58	0.3600	0.2996	0.2693
59	0.3404	0.2903	0.2634
60	0.3628	0.3284	0.3176
61	0.3823	0.3217	0.3130
62	0.3107	0.3020	0.2893
63	0.2812	0.2800	0.2431
64	0.3053	0.2824	0.2448
65	0.2985	0.2557	0.2346
66	0.2979	0.2488	0.2427
67	0.3376	0.2371	0.2383
68	0.3465	0.2908	0.2793
69	0.3148	0.2782	0.2593
70	0.2932	0.2540	0.2310
71	0.3149	0.2950	0.2505
72	0.2812	0.2486	0.2398
73	0.2961	0.2770	0.2488
74	0.3085	0.2646	0.2709
75	0.2779	0.2646	0.2291
76	0.2911	0.2468	0.2177
77	0.2848	0.2618	0.2293
78	0.3008	0.2581	0.2465
79	0.2816	0.2799	0.2287
80	0.2724	0.2471	0.2172
81	0.2798	0.2576	0.2318
82	0.2756	0.2617	0.2076
83	0.2706	0.2523	0.2209
84	0.3133	0.2711	0.2729
85	0.3255	0.2788	0.2587
86	0.2965	0.2586	0.2194
87	0.3092	0.2786	0.2444
88	0.2988	0.2673	0.2501
89	0.2881	0.2593	0.2371
90	0.2877	0.2565	0.2370
91	0.3080	0.2875	0.2505
92	0.3242	0.2714	0.2253
93	0.3286	0.2692	0.2283
94	0.3022	0.2509	0.2292
95	0.3062	0.2746	0.2456
96	0.3094	0.2819	0.2609
97	0.2911	0.2673	0.2381
98	0.2949	0.2721	0.2407
99	0.2569	0.2351	0.1982
100	0.3082	0.2931	0.2542
101	0.2992	0.2620	0.2502
102	0.2776	0.2627	0.2155
103	0.3154	0.2657	0.2340
104	0.2858	0.2313	0.2342
105	0.2882	0.2431	0.1917
106	0.2733	0.2452	0.2321
107	0.2807	0.2624	0.2194
108	0.2975	0.2395	0.2230
109	0.2835	0.2486	0.2184
110	0.3171	0.2556	0.2309
111	0.2766	0.2408	0.1946
112	0.2866	0.2519	0.2068
113	0.2856	0.2593	0.2118
114	0.2769	0.2483	0.2088
115	0.2877	0.2677	0.2312
116	0.2782	0.2530	0.2111
117	0.2647	0.2585	0.2235
118	0.2752	0.2718	0.2113
119	0.2662	0.2497	0.2058
120	0.2463	0.2176	0.1726
121	0.3098	0.2861	0.2137
122	0.2834	0.2444	0.1969
123	0.2958	0.2638	0.2128
124	0.2721	0.2515	0.1871
125	0.2373	0.2360	0.1912
126	0.2367	0.2058	0.1584
127	0.2259	0.1938	0.1480
128	0.2834	0.2346	0.1992
129	0.2550	0.2232	0.2143
130	0.2707	0.2416	0.1820
131	0.2692	0.2511	0.1902
132	0.3014	0.2702	0.2182
133	0.2319	0.2134	0.1650
134	0.2457	0.2175	0.1677
135	0.2344	0.2172	0.1721
136	0.2564	0.2307	0.1849
137	0.2837	0.2508	0.2045
138	0.2791	0.2445	0.1988
accuracy after 300 batches, %	76.9	79.62	82.58

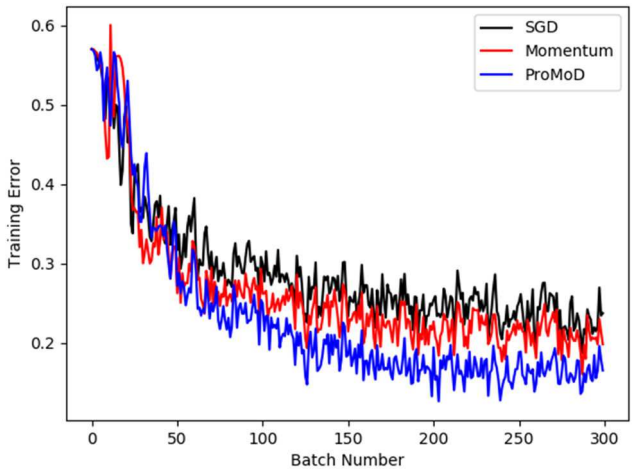

Fig. 5 *Progress of training for SGD, momentum and ProMoD methods for Fashion-MNIST dataset*

Table 4 Comparison of ProMoD with traditional momentum method

	ProMoD	Momentum
terms	PID	PI
learning speed	faster	slower
accuracy	similar	similar
batch size	more sensitive	less sensitive
stability	depends on α, β and γ terms	depends on α and β terms

8 References

[1] Ruder, S.: 'An overview of gradient descent optimization algorithms'. arXiv preprint arXiv:1609.04747, 2017

[2] Qian, N.: 'On the momentum term in gradient descent learning algorithms', *Neural Netw.*, 1999, **12**, (1), pp. 145–151. doi:10.1016/S0893-6080(98)00116-6

[3] Nesterov, Y.: 'A method for unconstrained convex minimization problem with the rate of convergence $o(1/k^2)$', *Doklady ANSSSR (Translated as Soviet. Math. Docl.)*, 1983, **269**, pp. 543–547

[4] Duchi, J., Hazan, E., Singer, Y.: 'Adaptive subgradient methods for online learning and stochastic optimization', *J. Mach. Learn. Res.*, 2011, **12**, pp. 2121–2159

[5] Zeiler, M.D.: 'ADADELTA: an adaptive learning rate method'. arXiv preprint arXiv:1212.5701, 2012

[6] Kingma, D.P., Ba, J.L.: 'Adam: a method for stochastic optimization'. Int. Conf. on Learning Representations, San Diego, CA, USA., 2015, pp. 1–13

[7] Nesterov, Y.: 'A method of solving a convex programming problem with convergence rate O(1/k2)', *In Soviet Mathematics Doklady*, 1983, **27**, pp. 372–376

[8] Dozat, T.: 'Incorporating Nesterov momentum into Adam'. ICLR Workshop, San Juan, Puerto Rico, 2016, (1), pp. 2013–2016

[9] Sutskever, I., Martens, J., Dahl, G.*, et al.*: 'On the importance of initialization and momentum in deep learning'. Proc. of the 30th Int. Conf. on Machine Learning (ICML-13), Atlanta, GA, USA., 2013, pp. 1139–1147

[10] Dauphin, Y., de Vries, H., Bengio, Y.: 'Equilibrated adaptive learning rates for non-convex optimization'. Advances in Neural Information Processing Systems, Montreal, Canada, 2015, pp. 1504–1512

[11] Andayani, U., Nababan, U.E.B., Siregar, B.*, et al.*: 'Optimization backpropagation algorithm based on Nguyen–Widrom adaptive weight and adaptive learning rate'. 2017 Fourth Int. Conf. on Industrial Engineering and Applications (ICIEA), Nagoya, 2017, pp. 363–367

[12] Duchi, J., Hazan, E., Singer, Y.: 'Active subgradient methods for online learning and stochastic optimization', *J. Mach. Learn. Res.*, 2011, **12**, pp. 2121–2159

[13] Martens, J.: 'Deep learning via H non-stationary Hessian-free optimization'. Proc. of the 27th Int. Conf. on Machine Learning (ICML-10), Haifa, Israel, 2010, pp. 735–742

[14] Dauphin, Y., Pascanu, R., Gulcehre, C.*, et al.*: 'Identifying and attacking the saddle point problem in high-dimensional non-convex optimization'. NIPS'2014, Montreal, Canada, 2014

[15] Pascanu, R., Bengio, Y.: 'Revisiting natural gradient for deep networks'. Int. Conf. on Learning Representations, Banff, Canada, 2014

[16] Vinyals, O., Povey, D.: 'Krylov subspace descent for deep learning'. arXiv preprint arXiv:1111.4259, 2011

[17] Tieleman, T., Hinton, G.: 'Lecture 6.5 – RMSProp, COURSERA: neural networks for machine learning'. Technical report, 2012

[18] Rubio, J.J., Cruz, D.R., Elias, I.*, et al.*: 'ANFIS system for classification of brain signals', *J. Intell. Fuzzy Syst.*, 2019, **37**, (3), pp. 4033–4041

[19] Rubio, J.J.: 'SOFMLS: online self-organizing fuzzy modified least-squares network', *IEEE Trans. Fuzzy Syst.*, 2009, **17**, (6), pp. 1296–1309

[20] Giap, C.N., Son, L.H., Chiclana, F.: 'Dynamic structural neural network', *J. Intell. Fuzzy Syst.*, 2018, **34**, (4), pp. 2479–2490

[21] Rubio, J.J., Garcia, E., Ochoa, G.: 'Unscented Kalman filter for learning of a solar dryer and a greenhouse', *J. Intell. Fuzzy Syst.*, 2019, **37**, (5), pp. 6731–6741, doi: 10.3233/IFS-190216

[22] Zweiri, Y.H., Whidborn, J.F., Seneviratne, L.D.: 'A three-term backpropagation algorithm', *Neurocomputing*, 2003, **50**, pp. 305–318

[23] Bhaya, A., Kaszkurewicz, E.: 'A control-theoretic approach to the design of zero finding numerical methods', *IEEE Trans. Autom. Control*, 2007, **52**, (6), pp. 1014–1026

[24] Zweiri, Y.H., Seneviratne, L.D., Althoefer, K.: 'Stability analysis of a three-term backpropagation algorithm', *Neural Netw.*, 2005, **18**, pp. 1341–1347

[25] Zeraatkar, E., Karimaghaee, P., Noroozi, N.: 'A quasi-PID backpropagation algorithm based on Lyapunov stability theory for neural network'. 19th Iranian Conf. on Electrical Engineering, Tehran, 2011, pp. 1–6

[26] Gürhanlı, A., Çevik, T., Çevik, N.: 'Effect of derivative action on back-propagation algorithms'. Innovations in Bio-Inspired Computing and Applications. IBICA 2018. Advances in Intelligent Systems and Computing, Kochi, India, 2019, vol. 939

[27] Gürhanlı, A.: 'Image recognition using ProMoD backpropagation algorithm'. Applied Research Int. Conf. on Millennial Technology Challenges (ARICMTC), Istanbul, 2019

[28] Kim, P.: '*MATLAB deep learning with machine learning, neural networks and artificial intelligence*' (Apress, Soul, 2017, 1st edn.), doi: 10.1007/978-1-4842-2845-6

[29] Gürhanlı, A.: 'Effect of batch size on accuracy of ProMod backpropagation algorithm used in image recognition'. Proc. ISENSA 2019, Int. Symp. Engineering Natural Sciences and Architecture, Kocaeli, 2019