Long Term Temporal Averaging for Stochastic Optimization of Deep Neural Networks

Nikolaos Passalis*
Department of Informatics
Aristotle University of Thessaloniki, Thessaloniki 54124 , Greece
passalis@csd.auth.gr

Anastasios Tefas
Department of Informatics
Aristotle University of Thessaloniki, Thessaloniki 54124 , Greece
tefas@aiia.csd.auth.gr

*Corresponding Author: passalis@csd.auth.gr, Tel:+30-231-099-6361, Fax:+30-231-099-8453

Long Term Temporal Averaging for Stochastic Optimization of Deep Neural Networks

Nikolaos Passalis*
Department of Informatics
Aristotle University of Thessaloniki, Thessaloniki 54124 , Greece
passalis@csd.auth.gr

Anastasios Tefas
Department of Informatics
Aristotle University of Thessaloniki, Thessaloniki 54124 , Greece
tefas@aiia.csd.auth.gr

*Corresponding Author: passalis@csd.auth.gr, Tel:+30-231-099-6361, Fax:+30-231-099-8453

Date: Received: date / Accepted: date

Key words: Stochastic Optimization - Temporal Averaging - Deep Convolutional Neural Networks

Abstract:

Deep Learning (DL) models are capable of successfully tackling several difficult tasks. However, training deep neural models is not always a straightforward task due to several well known issues, such as the problems of vanishing and exploding gradients. Furthermore, the stochastic nature of most of the used optimization techniques inevitably leads to instabilities during the training process, even when state-of-the-art stochastic optimization techniques are used. In this work we propose an advanced temporal averaging technique that is capable of stabilizing the convergence of stochastic optimization for neural network training. Six different datasets and evaluation setups are used to extensively evaluate the proposed method and demonstrate the performance benefits. The more stable convergence of the algorithm also reduces the risk of stopping the training process when a bad descent step was taken and the learning rate was not appropriately set.

1. Introduction

Deep Learning (DL) models are capable of successfully tackling several difficult tasks, such as large scale visual recognition [28], object detection [19, 26], realistic voice synthesis [22], and others [16]. This has led to a number of spectacular applications ranging from drones that autonomously perform various tasks [23, 29], to systems that outperform doctors on diagnosing diseases [1, 12, 20, 34], and malware detection systems [36].

However, training deep neural models is not always a straightforward task due to several well known issues, such as the problems of vanishing and exploding gradients [2, 32]. Various methods have been proposed to stabilize and smoothen the convergence of the training procedure [5, 9, 10, 11, 13, 31, 32]. The stochastic nature of most of the used optimization techniques inevitably leads to instabilities during the training process, even when state-of-the-art stochastic optimization methods are used [13], requiring careful fine-tuning of their hyper-parameters. If a slightly larger learning rate than the optimal is selected, then the training process will be unstable (and might not even converge). On the other hand, if the learning rate is too small, then the optimization process will slow down significantly. Recognizing these difficulties, parameter averaging has been proposed to reduce the effect of noise on the stochastic updates and achieve better generalization [13, 21, 25, 27].

The aforementioned problems are often more evident in specific applications. For example, among the most crucial components of an intelligent system capable of performing automated drone-based shooting is estimating the pose of the main actors [23]. To this end, a Convolutional Neural Network (CNN) can be trained to perform the task of facial pose estimation. However, such CNNs exhibit especially unstable behavior that can be also partially attributed to the noisy nature of the data. To understand this, consider the process of facial pose estimation. First, a face is detected using an object detector, such as the YOLO detector [26], or the SSD detector [19]. Then, the bounding box of the face is cropped, resized and fed to the pose estimation CNN. However, the object detector is usually incapable of perfectly centering and determining the bounds of the face introducing a significant amount of noise into the aforementioned process.

In this paper, an advanced temporal averaging technique that is capable of stabilizing the convergence of stochastic optimization for deep neural network training is proposed. The proposed method employs an exponential averaging technique to bias the parameters of the neural network towards stabler states. As it is demonstrated in Section 3, this is equivalent to first taking big descent steps to explore the solution space and then annealing towards stabler states. Six different datasets and evaluation setups are used to extensively evaluate the proposed method and demonstrate the performance benefits. The more stable convergence of the algorithm also reduces the risk of stopping the training process when a bad descent step was taken and the learning rate was not appropriately set, ensuring that the network will perform well at any point of the training process (after a certain number of iterations have been performed).

This paper is an extended version of our previous paper [24]. The contributions of this paper are the following. First, after careful analysis of the proposed algorithm, a more robust version is derived, in which the stable states are selected according to the loss observed during the training and update rate annealing is not required. This also makes the proposed method easier to use, since only two hyper-parameters are to be selected and it is demonstrated that the default values for these hyper-parameters work well for a wide range of problems. Also, the proposed method is more extensively evaluated using additional datasets and evaluation setups. The effect of the hyper-parameters on the performance of the method is also thoroughly evaluated. Furthermore, recognizing the difficulties in the evaluation of stochastic methods, a more careful experimental protocol is used to ensure a fair comparison between the evaluated methods (the experiments were repeated multiple times and the same initialization was used for the corresponding runs between different methods).

The rest of the paper is structured as follows. First, the related work is briefly introduced and discussed in Section 2. Then, the proposed method is presented in detail in Section 3 and the experimental evaluation is provided in Section 4. Finally, conclusions are drawn in Section 5.

2. Related Work

Several methods have been proposed for training deep neural networks as well as for improving the convergence of Stochastic Gradient Descent (SGD). For example, using rectifier activation units [9], batch normalization [11], and residual connections [10, 31], allows for effectively dealing with the problem of vanishing gradients. Furthermore, advanced optimization techniques, such as the Adagrad [4], Adadelta [37], and Adam [13] algorithms are capable of effectively dealing with gradients of different magnitude, improving the convergence speed. Each of these techniques deal with a specific problem that arises during the training of deep neural networks. The method proposed in this paper is complementary to these methods since it addresses a different problem, i.e., improves the stability of the training process. This is also demonstrated in Section 4 where the proposed method is combined with some of the aforementioned methods to improve the stability of the training process.

Parameter averaging techniques were also proposed in some works to deal with the noisy updates of stochastic gradient descent [21, 25, 27], while also studying the convergence properties after averaging

the stochastic updates. In these works, after completing the optimization process, the parameters were replaced with the averaged parameters, as calculated during the training process. A more deliberate technique was proposed in [13], where an exponential moving average over the parameters of the network was used to ensure that higher weight is given to the recent states of the network. However, in these approaches the averaged parameters are not used during the optimization. In [18, 42], a similar approach is used to stabilize the convergence of Q-learning for deep neural networks by keeping a separate slowly changing neural network used for providing the target values. The method proposed in this paper is different from the aforementioned methods, since the weights of the network are not averaged after each iteration. Instead, a number of descent steps are taken, e.g., 10 optimization steps, and after them the parameters of the networks are updated according to the observed loss. This allows for better exploring the solution space, while maintaining the stability that the averaging process offers, as demonstrated in Section 4. Note that global parameter averaging, i.e., averaging the parameters after the training process as in [13, 21, 25, 27], can be also used on top of the proposed method, further improving the stability of the proposed method.

3. Proposed Method

In this Section the used notation is briefly introduced and the proposed Long-Term Temporal Averaging (LT-TA) algorithm is presented in detail. Then, we examine the behavior of the proposed LT-TA algorithm by analyzing the employed parameter update technique.

Let θ denote the parameters of the neural network that is to be optimized towards minimizing a loss function $L(\theta, x)$, where x denotes a batch of the training data. The notation θ_t is used to denote the parameters after t optimization iterations. Also, let $f(\theta, x, \eta)$ be an optimization method that provides the updates for the parameters of the neural network, where η denotes the hyper-parameters of the optimization method, e.g., the learning rate. Any optimization technique can be used ranging from the simple Stochastic Gradient Descent method [7], to more advanced techniques, such as the Adagrad [4], Adadelta [37], or, the more recently proposed, Adam method [13].

The proposed method is shown in Algorithm 1. The LT-TA algorithm keeps track of a stable version of the parameters of the network, as determined by observing the mean loss during the optimization process. If the optimization process proceeds smoothly, then the stable state is not used. On the other hand, if multiple bad descent directions were taken during the last optimization steps, then the network is biased towards a previously stable state, by performing exponentially averaging. Of course, any other averaging or update method can be used to this end. Furthermore, exponential averaging can also be used to update the stable state of the network. However, this approach slightly slowed down the convergence in the conducted experiments, without providing any significant stability improvement, since the stable states are already selected according to a reliable criterion, i.e., the loss observed during the optimization.

The proposed method works as follows. First, the initial version θ_{stable} is set to the initial state of the network (line 2) and the loss inducted by the stable state is initialized to an arbitrary large number (line 3). Also, the variable used to measure the mean current loss is initialized to 0 (line 4). During the optimization (lines 5-16) the proposed algorithm performs regular optimization updates (lines 6-7) and keeps track of the mean loss during the optimization (line 8). However, every N_S iterations the mean loss observed during the last iterations (line 10) is compared to the loss of the latest stable state (line 11). If the current loss is lower, then the stable stable is updated using the current weights (lines 12-13), since the current state can be considered stable. Otherwise, the weights of the network are biased towards the previous stable state using exponential averaging (line 15). Finally, the current loss variable is reset after N_S iterations (line 16).

Input: A training set of data X, the initial parameters of the network θ_0, a loss function $L(\bullet)$, and an optimization method $f(\bullet)$ along with its hyper-parameters η
Parameters: The update rate α, the exploration window N_S, and the number of iterations N
Output: The optimized parameters θ_N

procedure LT-TA Algorithm
  $\theta_{stable} \leftarrow \theta_0$
  $L_{stable} \leftarrow \infty$
  $L_{current} \leftarrow 0$
  for $t \leftarrow 1; t \leq N; t++$ **do**
    Sample a batch x from X
    $\theta_t \leftarrow f(\theta_{t-1}, x, \eta)$
    $L_{current} \leftarrow L_{current} + L(\theta_t, x)$
    if $mod(t, N_S) = 0$ **then**
      $L_{current} \leftarrow L_{current} / N_S$
      if $L_{stable} > L_{current}$ **then**
        $\theta_{stable} \leftarrow \theta_t$
        $L_{stable} \leftarrow L_{current}$
      else
        $\theta_t \leftarrow \alpha\, \theta_t + (1-\alpha)\, \theta_{stable}$
      $L_{current} \leftarrow 0$
  return θ_N

Algorithm 1: Long-Term Temporal Averaging Algorithm

This is equivalent to performing large exploration descent steps during the N_S iterations and then slowing down the learning in order to update the network when bad descent directions were taken. Thus, the stable states work as attractors that bias the network towards them. The parameter α (also called update rate) controls the influence of the stable states on the optimization procedure. The update rate is a positive number that ranges from 0 to 1. For $\alpha = 1$ no temporal averaging is used, while for $\alpha = 0$ the network remains at the stable state. The effect of these hyper-parameters on the performance of the proposed method is thoroughly examined in Section 4. After performing N iterations the algorithm returns the parameters of the network.

In the preliminary version of our technique presented in [24], we used an exponential decay strategy to reduce the update rate during the optimization. However, after careful analysis of the proposed technique we concluded that this process was not critical. Decaying the update rate allows for faster convergence for the first few iterations. Omitting this strategy can initially slightly slow down the convergence speed, but the overall effect after the first few iterations is usually minimal. Therefore, this strategy was removed from the proposed algorithm, effectively making the method easier to use (one less hyper-parameter has to be selected). Furthermore, the stable states are now selected according to the loss observed during the optimization, instead of simply running an exponential averaging on the weights, further increasing the stability and the convergence speed of the proposed method.

To better understand how the proposed algorithm works consider the first N_S iterations when the simple stochastic gradient descent algorithm with learning rate η is used to provide the optimization updates and the method starts from a initially stable state ($\theta_{stable}=\theta_0$.):

$$\begin{aligned}
\theta_1 &= \theta_0 - \eta \frac{\partial \mathscr{L}}{\partial \theta_0}, \\
\theta_2 &= \theta_1 - \eta \frac{\partial \mathscr{L}}{\partial \theta_1}, \\
&\vdots \\
\theta_{N_S-1} &= \theta_{N_S-2} - \eta \frac{\partial \mathscr{L}}{\partial \theta_{N_S-2}}, \\
\theta_{N_S} &= \theta_{N_S-1} - \eta \frac{\partial \mathscr{L}}{\partial \theta_{N_S-1}}.
\end{aligned} \tag{1}$$

It is easy to see that after N_S optimization steps the weights of the network can be expressed as a weighted sum over the descent steps:

$$\theta_{N_S} = \theta_{stable} - \eta \sum_{i=0}^{N_S-1} \frac{\partial \mathscr{L}}{\partial \theta_i}, \tag{2}$$

since $\theta_{stable}=\theta_0$.

If we arrive at a new stable state (the loss is reduced during the N_S optimization steps), the we simply update the stable state with the new weights. On the other hand, if the new state increases the mean loss, then we bias the current state towards the last current stable state:

$$\begin{aligned} \theta_{N_S} &= (1-\alpha)\theta_{stable} + \alpha\theta_{N_S} \\ &= \theta_{stable} - \alpha\eta \sum_{i=0}^{N_S-1} \frac{\partial \mathcal{L}}{\partial \theta_i}. \end{aligned} \tag{3}$$

Therefore, updating the weights by employing the stable state θ_{stable} is equivalent to lowering the learning rate of the previous updates to $\alpha\eta$ while updating the parameters θ_{stable}. However this is not equivalent to performing optimization with the lowered learning rate. To understand this note that the intermediate states $\theta_1, \theta_2, \theta_3, \ldots, \theta_{NS}$ are calculated using the original learning rate η instead of the lowered rate $(1-\alpha)\eta$:

$$\theta_i = \theta_{i-1} - \eta \frac{\partial \mathscr{L}}{\partial \theta_{i-1}}, i < N_S - 1. \tag{4}$$

That is, during the N_S steps the proposed algorithm explores the solution space by taking large steps towards the descent direction, while the stable state θ_{stable} is employed to ensure that the network remains stable, even when the optimization process becomes unstable for a few training steps. This also ensures that relatively large descent steps that overshoot the local minima, which also possibly allow for discovering better local minima, will not affect the stability of the training procedure. To better understand this consider that the stable state will be used to bias the network parameters only if we end up into a consistently worse optimization point after N_S optimization steps.

4. Experiments

In this Section the proposed method is extensively evaluated and compared to other techniques. First, the effect of the two hyper-parameters (update rate α and exploration window N_S) on the stability of the proposed method is examined. Then, the proposed method is evaluated using several different deep neural network architectures, learning setups, and six datasets. Note that the evaluation focuses on lightweight network architectures that process small images, e.g., 32 x 32 pixels. This ensures that the developed networks can be deployed on embedded and mobile systems with limited processing power, such as drones that will assist several cinematography-oriented tasks [23, 43, 44, 45, 46], meeting the real-time requirements of these applications.

The Adam optimizer, with the default hyper-parameters, i.e., learning rate $\eta=0.001$, first moment decay rate $\beta_1=0.9$, and second moment decay rate $\beta_1=0.999$, was used for all the experiments conducted in this paper [13]. It is well known that fine-tuning these parameters (especially the learning rate) for each dataset can improve the obtained results. However, this kind of fine-tuning is a manual and time-consuming process. The proposed LT-TA technique was combined with the Adam algorithm, which was used to provide the parameter updates. A plain temporal averaging (TA) method was also derived by

setting the exploration window to 1, i.e., N_S=1, providing a second baseline for comparing the proposed technique. Note that plain parameter averaging techniques, such as [13, 21, 25, 27], can be also implemented on top of all the evaluated methods, possibly further increasing the performance of the methods. The keras library [40] was used for implementing the proposed method and conducting all the experiments.

For all the conducted experiments the mean value of each evaluated metric (loss or classification error) and the standard deviation of the calculated sample mean are reported. The experiments were repeated 30 times and the corresponding metrics are monitored over the course of the optimization process (the test metrics are monitored only during the last 5 epochs), unless otherwise stated. The bars in the figures are used to mark the 95% confidence intervals for each value. Furthermore, the setup/method with the best performance has been underlined in the following tables. Note that this does not necessary imply that the underlined metric is statistically significantly better than the others.

Table 1: Network architecture used for the MNIST dataset (conv. refers to convolutional layers)

Layer Type	Output Shape
Input	28 x 28 x 1
Conv. (3 x 3, 32 filters)	26 x 26 x 3
Max Pooling (2 x 2)	13 x 13 x 32
Conv. (3 x 3, 64 filters)	11 x 11 x 64
Max Pooling (2 x 2)	5 x 5 x64
Dense (512 neurons)	512
Dropout (p=0.5)	512
Dense (10 neurons)	10

4.1 Parameter Selection

The effect of the update rate α and exploration window N_S on the convergence of the optimization process is examined using the MNIST database of handwritten digits (abbreviated as MNIST) [17]. The MNIST database is a well-known dataset that contains 60,000 train and 10,000 test images of handwritten digits. There are 10 different classes, one for each digit (0 to 9), and the size of each image is 28 x 28. Some sample images are shown in Figure 1. The network architecture used for the conducted experiments is summarized in Table 1. The rectifier activation function was used for all the layers except of the last one, where the softmax activation function was combined with the cross-entropy loss for training the network [8]. The network was regularized using the dropout technique [30]. The optimization ran for 10 epochs with batch size 32 and and the average of the evaluated metrics during the 80% of the optimization (last 8 epochs) are reported. The experiments were repeated 5 times and the mean and the standard deviation are reported.

Figure 1: Samples images from the MNIST dataset

The experimental results for different update rates α are shown in Table 2. The exploration window was set to N_S=100, the optimization ran for 10 epochs with batch size 32, while the experiments were repeated 5 times and the mean and the standard deviation are shown. The mean loss and the mean classification error during the last epochs are reported. The best results were obtained for α=0.995. Note that smaller values tend to reduce the optimization speed by strongly biasing the network towards the older stable states. The obtained results highlight the ability of temporal averaging to improve the convergence of the optimization process over the plain SGD/Adam, since the error and the loss decreases (lower values demonstrate improved performance) as the value of α decreases (up to a certain point).

After selecting the value of α=0. 995 for the update rate, the effect of the exploration windows N_S is evaluated in Table 3. Large exploration windows tend to provide more reliable estimation for the quality of the stable states by calculating the loss over larger temporal windows. The best results were obtained for N_S=50. The selected values (α=0.995 and N_S=50) were used for all the conducted experiments in this Section, except otherwise stated, and proved to provide good performance for a wide range of different datasets, evaluation setup and network architectures.

Table 2: Evaluating the effect of the update rate α on the behavior of the proposed LT-TA method.

α	**Train loss (x 10^{-2})**	**Train error (%)**
0.999	0.878 ± 0.097	0.276 ± 0.032
0.995	0.832 ± 0.050	0.258 ± 0.013
0.99	0.832 ± 0.094	0.263 ± 0.031
0.95	0.939 ± 0.077	0.297 ± 0.025
0.9	1.140 ± 0.112	0.360 ± 0.038

Table 3: Evaluating the effect of the update rate N_S on the behavior of the proposed LT-TA method.

N_S	**Train loss (x 10^{-2})**	**Train error (%)**
1	0.844 ± 0.065	0.267 ± 0.023
20	0.931 ± 0.130	0.293 ± 0.042
50	0.826 ± 0.061	0.256 ± 0.026
100	0.832 ± 0.050	0.258 ± 0.013
200	0.857 ± 0.042	0.272 ± 0.019

4.2 MNIST Evaluation

Apart from the parameter selection experiments, the MNIST dataset was also used to compare the proposed method both to the plain Adam algorithm, as well as to a plain Temporal Averaging (TA) baseline. Again, note that for all the conducted experiments (TA and LT-TA methods) the value of α was set 0.995. The evaluation results are shown in Table 4. The optimization ran for 10 epochs with batch size 32.

The proposed LT-TA method performs slightly better than both the plain Adam (reported as “Baseline'” method in Table 4) and TA methods. The same behavior is also observed for the test set (Table 5), where the proposed method achieves the lowest classification error (0.726% vs. 0.740% for the next best performing method). Note that actually the TA method fails to improve the convergence, since it cannot reliably select the stable states (the loss is calculated over a window with size 1). For the test set, instead of reporting the accuracy at the last epoch, the average over the last 5 epochs is reported to evaluate the stability of the obtained solutions. A low average test error over the last 5 epochs demonstrates that the optimization could be stopped at any of these epochs, i.e., the risk of stopping when a bad descent step was taken is smaller.

Table 4: MNIST Train Evaluation: Comparing the proposed LT-TA method to both the plain Adam (Baseline) and Temporal Averaging during the optimization.

Method	**Train loss (x 10^{-2})**	**Train error (%)**
Baseline	1.241 ± 0.071	0.386 ± 0.023
TA	3.243 ± 0.392	1.004 ± 0.122
LT-TA	1.214 ± 0.078	0.374 ± 0.026

Table 5: MNIST Test Evaluation: Comparing the proposed LT-TA method to both the plain Adam (Baseline) and Temporal Averaging using the supplied test set.

Method	Test loss (x 10^{-2})	Test error (%)
Baseline	2.654 ± 0.166	0.740 ± 0.055
TA	3.395 ± 0.357	1.096 ± 0.118
LT-TA	2.552 ± 0.170	0.726 ± 0.049

To demonstrate the improved stability of the proposed method, the loss during the optimization process (learning curve) is also provided in Figure 2. The proposed LT-TA method leads to slightly reduced loss at almost all of the optimization epochs, while it is also capable of slightly reducing the fluctuations caused by the stochastic nature of the updates.

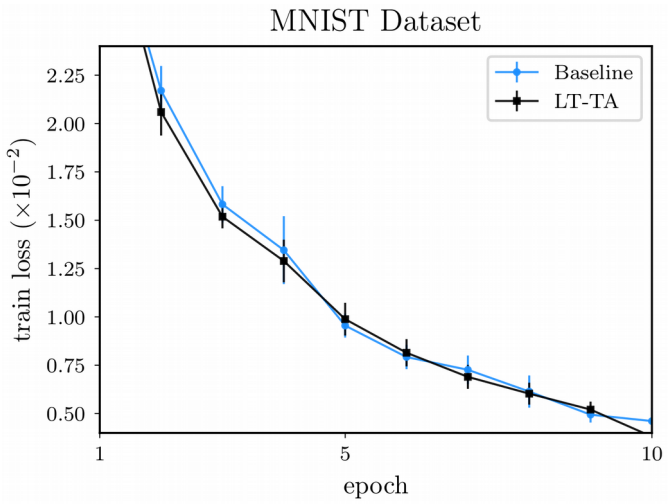

Figure 2: MNIST Evaluation: Loss during the optimization process. Figure best viewed in color.

The proposed method was also evaluated using a different setup for the employed optimizer by increasing the learning rate to η=0.005. The train evaluation is provided in Table 6, while the test evaluation in Table 7. Since the convergence was significantly more noisy, the update rate was reduced to 0.99 in order to more strongly bias the network towards the stable states. Again, the same behavior as before is observed. The proposed LT-TA method improves slightly the convergence, reducing the train loss from 4.397 (Baseline) to 4.147 and the test loss from 6.319 (Baseline) to 6.081. The same behavior is also observed in the learning curves plotted in Figure 3. Note that again the plain TA method was not capable of reliably selecting the stable states and, thus, failed to improve the convergence.

Table 6: MNIST Train Evaluation (higher learning rate): Comparing the proposed LT-TA method to both the plain Adam (Baseline) and Temporal Averaging during the optimization.

Method	Train loss (x 10^{-2})	Train error (%)
Baseline	4.397 ± 0.875	1.328 ± 0.292
TA	51.840 ± 89.162	19.557 ± 34.605
LT-TA	4.147 ± 0.604	1.270 ± 0.212

Table 7: MNIST Test Evaluation (higher learning rate): Comparing the proposed LT-TA method to both the plain Adam (Baseline) and Temporal Averaging using the supplied test set.

Method	Test loss (x 10^{-2})	Test error (%)
Baseline	6.319 ± 0.370	1.666 ± 0.195
TA	51.749 ± 89.194	19.523 ± 34.565
LT-TA	6.081 ± 0.552	1.629 ± 0.235

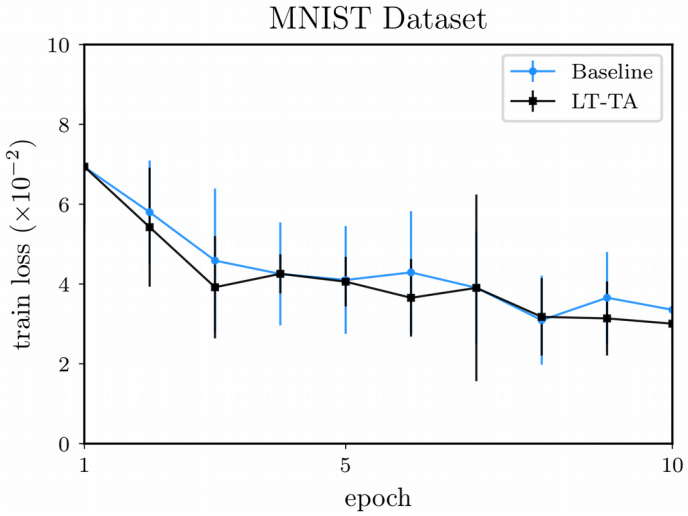

Figure 3: MNIST Evaluation (higher learning rate): Loss during the optimization process.

4.3 CIFAR10 Evaluation

The CIFAR10 dataset [15], contains 60,000 32x32 color images that belongs to 10 different categories: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. Some sample images are shown in Figure 4. The dataset is already split into 50,000 train and 10,000 test images. The CIFAR10 dataset is a labeled subset of the 80 million tiny images dataset [33]. A deeper and more complex network was used for the CIFAR10 dataset, since it is a significantly more complex dataset than the MNIST dataset. The used network architecture is shown in Table 8. Again, rectifier activation functions were used for all the layers except of the last one, where the softmax activation function was combined with the cross-entropy loss for training the network. Padded convolutions were used for some layers to avoid reducing too much the size of the output feature maps.

Figure 4: Samples images from the CIFAR10 dataset

Table 8: Network architecture used for the CIFAR10 dataset (conv. refers to convolutional layers)

Layer Type	Output Shape
Input	32 x 32 x 3
Padded Conv. (3 x 3, 32 filters)	32 x 32 x 3
Conv. (3 x 3, 64 filters)	30 x 30 x 64
Max Pooling (2 x 2)	15 x 15 x 64
Dropout (p = 0.25)	15 x 15 x 64
Padded Conv. (3 x 3, 64 filters)	15 x 15 x 64
Conv. (3 x 3, 128 filters)	13 x 13 x 128
Max Pooling (2 x 2)	6 x 6 x 128
Dropout (p = 0.25)	6 x 6 x 128
Dense (512 neurons)	512
Dropout (p=0.5)	512
Dense (10 neurons)	10

Table 9: CIFAR10 Train Evaluation: Comparing the proposed LT-TA method to both the plain Adam (Baseline) and Temporal Averaging during the optimization.

Method	Train loss (x 10^{-2})	Train error (%)
Baseline	54.550 ±1.792	19.123 ± 0.649
TA	75.302 ± 7.327	26.608 ± 2.661
LT-TA	52.921 ± 2.360	18.564 ± 0.846

Table 10: CIFAR10 Test Evaluation: Comparing the proposed LT-TA method to both the plain Adam (Baseline) and Temporal Averaging using the supplied test set.

Method	Test loss (x 10^{-2})	Test error (%)
Baseline	53.973 ± 1.263	17.740 ± 0.358
TA	73.756 ± 7.542	25.478 ± 2.796
LT-TA	53.400 ± 1.348	17.615 ± 0.473

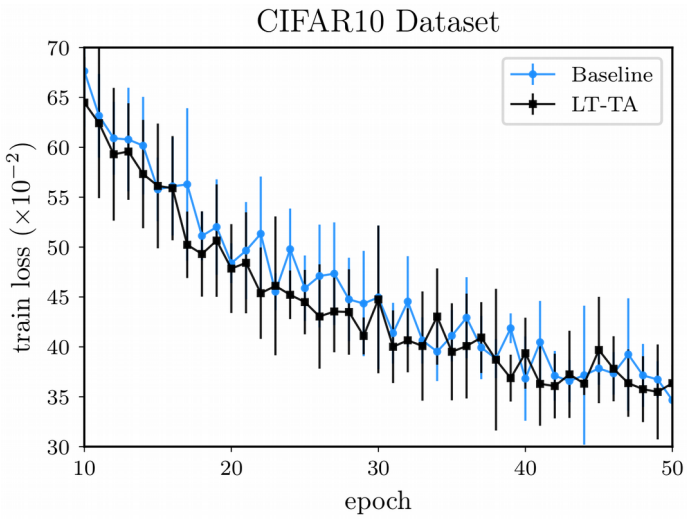

Figure 5: Loss during the optimization process. Figure best viewed in color.

The evaluation results for the train set are shown in Table 9, while the test evaluation results are provided in Table 10. The optimization ran for 50 epochs with batch size 32, while all the epochs were used to evaluate the quality and stability of the optimization process using the train data. The proposed LT-TA method leads to slightly improved results than the other techniques for both the training and testing evaluations. The learning curves for the evaluated methods are provided in Figure 5.

4.4 Fashion MNIST Evaluation

The Fashion MNIST [35] is a dataset that contains 60,000 train samples and 10,000 test samples that belong to one of the following 10 classes: t-shirt, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, and ankle boot. Some sample images are shown in Figure 6. Fashion MNIST was designed as a drop-in replacement for the MNIST dataset. The same evaluation setup as with the MNIST dataset (Section 4.2) is used, apart from using a slightly more complex network architecture (Table 11) and running the optimization for 20 epochs.

Figure 6: Samples images from the Fashion MNIST dataset

Table 11: Network architecture used for the Fashion MNIST dataset (conv. refers to convolutional layers)

Layer Type	Output Shape
Input	28 x 28 x 1
Conv. (3 x 3, 32 filters)	26 x 26 x 32
Conv. (3 x 3, 64 filters)	24 x 24 x 64
Max Pooling (2 x 2)	12 x 12 x 64
Conv. (3 x 3, 64 filters)	10 x 10 x 64
Max Pooling (2 x 2)	5 x 5 x 64
Dense (512 neurons)	512
Dropout (p=0.5)	512
Dense (10 neurons)	10

Table 12: Fashion MNIST Train Evaluation: Comparing the proposed LT-TA method to both the plain Adam (Baseline) and Temporal Averaging during the optimization.

Method	Train loss (x 10^{-2})	Train error (%)
Baseline	10.372 ± 0.673	3.804 ± 0.267
TA	27.328 ± 2.543	10.132 ± 0.974
LT-TA	10.214 ± 0.370	3.744 ± 0.141

Table 13: Fashion MNIST Test Evaluation: Comparing the proposed LT-TA method to both the plain Adam (Baseline) and Temporal Averaging using the supplied test set.

Method	Test loss (x 10^{-2})	Test error (%)
Baseline	32.284 ± 1.330	7.691 ± 0.164
TA	29.579 ± 2.544	10.821 ± 0.935
LT-TA	32.312 ± 0.854	7.725 ± 0.126

For the training evaluation the results are similar to the regular MNIST data (Table 12). That is, the proposed LT-TA method performs slightly better than both the baseline and plain TA methods. However, for the test evaluation shown in Table 13, the Baseline method performs slightly better than the LT-TA method. This behavior can attributed to overfitting phenomena that might occur when the LT-TA method manages to find a better local minimum of the loss function than the simple TA method. Therefore, even though the LT-TA actually performs better on the training set, it can sometimes lead to overfitting lowering the accuracy on the test set. This behavior is well known and can be prevented using more aggressive regularization, e.g., dropout with higher rate [30].

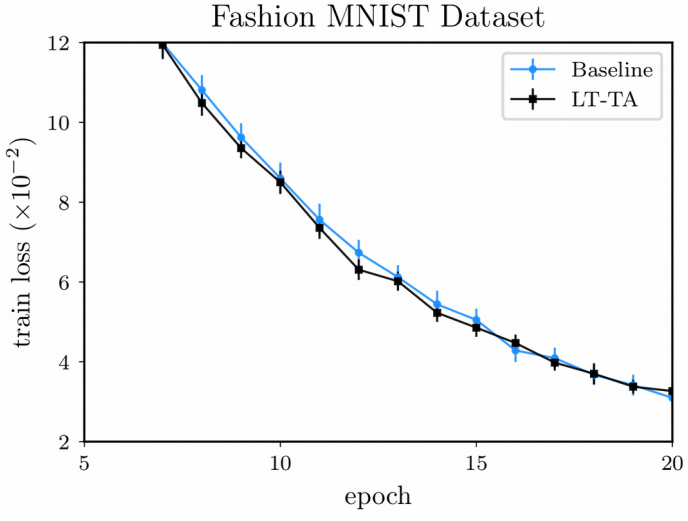

Figure 7: Fashion MNIST Evaluation: Loss during the optimization process. Figure best viewed in color.

4.5 HPID Evaluation

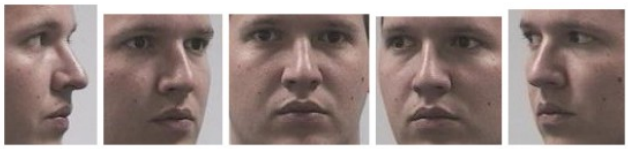

Figure 8: Cropped face images from the HPID dataset

The Head Pose Image Dataset (HPID) [6] is a dataset that contains 2,790 face images of 15 subjects in various poses taken in a constrained environment. All images were resized to 32x32 pixels before feeding them to the used CNN. Some sample images are shown in Figure 8. The horizontal pose of the face images (pan) is to be predicted. The HPID dataset provides discrete targets (13 steps) that were converted into a continuous value used for training/testing the model. The predefined train/test splits were used. The network architecture used for predicting the pose is shown in Table 14. The rectifier activation function were used for the hidden layers (no activation was used for the output, since the yaw is directly predicted), while the mean squared loss was used for training.

During the training the following data augmentation techniques were also used to prevent overfitting the network and ensure good generalization performance:

1. random vertical flip with probability 0.5
2. random horizontal shift up to 5%
3. random vertical shift up to 5%
4. random zoom up to 5%
5. random rotation up to 10 degrees

The vertical and horizontal shifts simulate the behavior of face detectors that are usually unable to perfectly align the face in the images, while the zoom and rotation transformations further increase the scale/rotation invariance of the network.

Table 14: Network architecture used for the HPID dataset (conv. refers to convolutional layers)

Layer Type	Output Shape
Input	32 x 32 x 3
Padded Conv. (3 x 3, 32 filters)	32 x 32 x 32
Max Pooling (2 x 2)	16 x 16 x 32
Dropout (p=0.25)	16 x 16 x 32
Padded Conv. (3 x 3, 64 filters)	16 x 16 x 64
Max Pooling (2 x 2)	8 x 8 x 64
Dropout (p=0.5)	8 x 8 x 64
Dense (256 neurons)	256
Dropout (p=0.5)	256
Dense (10 neurons)	1

The evaluation results for the optimization process are reported in Table 15. The optimization ran for 50 epochs with batch size 32. The LT-TA perform slightly better than the plain Adam (Baseline) method and the TA method achieving the lowest overall loss/error. This behavior is also reflected in the test evaluation (Table 16), where the evaluated method improves slightly the test error, as well as in the learning curves shown in Figure 9. Finally, note that due to differences in the used network architecture and the removal of the Local Constrant Normalization (LCN) layers (LCN is not available in the keras framework used for implementing the proposed method [3]) the evaluation results are different from those reported in [24].

Table 15: HPID Train Evaluation: Comparing the proposed LT-TA method to both the plain Adam (Baseline) and Temporal Averaging during the optimization.

Method	Train loss (x 10^{-2})	Train error (%)
Baseline	0.168 ± 0.064	10.102 ± 1.730
TA	0.185 ± 0.062	10.808 ± 1.644
LT-TA	0.167 ± 0.056	10.030 ± 1.465

Table 16: HPID Test Evaluation: Comparing the proposed LT-TA method to both the plain Adam (Baseline) and Temporal Averaging using the supplied test set.

Method	Test loss (x 10^{-2})	Test error (%)
Baseline	0.077 ± 0.008	7.753 ± 0.383
TA	0.081 ± 0.011	7.949 ± 0.551
LT-TA	0.077 ± 0.008	7.740 ± 0.419

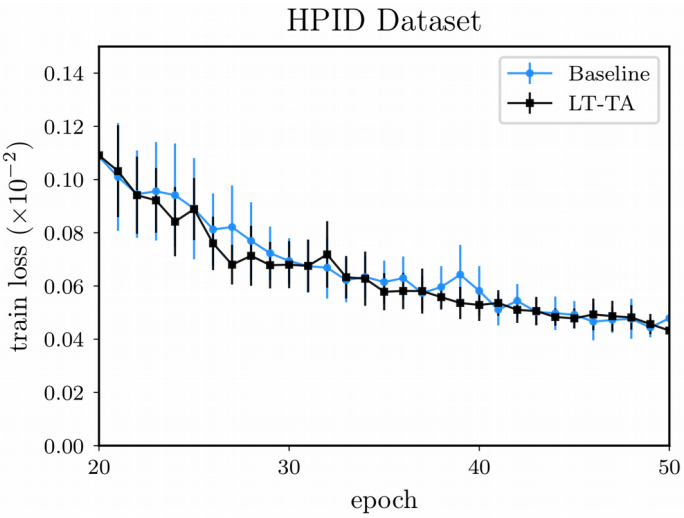

Figure 9: HPID Evaluation: Loss during the optimization process. Figure best viewed in color.

4.6 AFLW Evaluation

The Annotated Facial Landmarks in the Wild (AFLW) dataset [14], is a large-scale dataset for facial landmark localization. The 75% of the images were used to train the models, while the rest 25% for evaluating the accuracy of the models. The face images were cropped according to the supplied annotations and then resized to 32x32 pixels. Face images smaller than 16x16 pixels were not used for training or evaluating the model. Some sample images are shown in Figure 10. The AFLW dataset provides continuous pose targets that were directly used for training the network. The same data augmentation technique as for the HPID dataset was used (Section 4.5). The network architecture used for predicting the pose is shown in Table 17. The rectifier activation function was used for the hidden layers (no activation was used for the output, since the yaw is directly predicted), while the mean squared loss was used for training. As before, the optimization ran for 30 epochs with batch size 32.

Figure 10: Cropped face images from the ALFW dataset

Table 17: Network architecture used for the AFLW dataset (conv. refers to convolutional layers)

Layer Type	Output Shape
Input	32 x 32 x 3
Padded Conv. (3 x 3, 16 filters)	32 x 32 x 16
Padded Conv. (3 x 3, 16 filters)	32 x 32 x 16
Max Pooling (2 x 2)	16 x 16 x 16
Dropout (p = 0.25)	16 x 16 x 16
Padded Conv. (3 x 3, 64 filters)	16 x 16 x 64
Padded Conv. (3 x 3, 64 filters)	16 x 16 x 64
Max Pooling (2 x 2)	8 x 8 x 64
Dropout (p = 0.25)	8 x 8 x 64
Dense (256 neurons)	256
Dropout (p=0.5)	256
Dense (10 neurons)	10

Table 18: AFLW Train Evaluation: Comparing the proposed LT-TA method to both the plain Adam (Baseline) and Temporal Averaging during the optimization.

Method	Train loss (x 10^{-2})	Train error (%)
Baseline	0.257 ± 0.039	12.838 ± 0.907
TA	0.286 ± 0.041	13.700 ± 0.942
LT-TA	0.257 ± 0.039	12.835 ± 0.933

Table 19: AFLW Test Evaluation: Comparing the proposed LT-TA method to both the plain Adam (Baseline) and Temporal Averaging using the supplied test set.

Method	Test loss (x 10^{-2})	Test error (%)
Baseline	0.160 ± 0.010	10.013 ± 0.343
TA	0.195 ± 0.020	11.192 ± 0.599
LT-TA	0.160 ± 0.011	10.017 ± 0.384

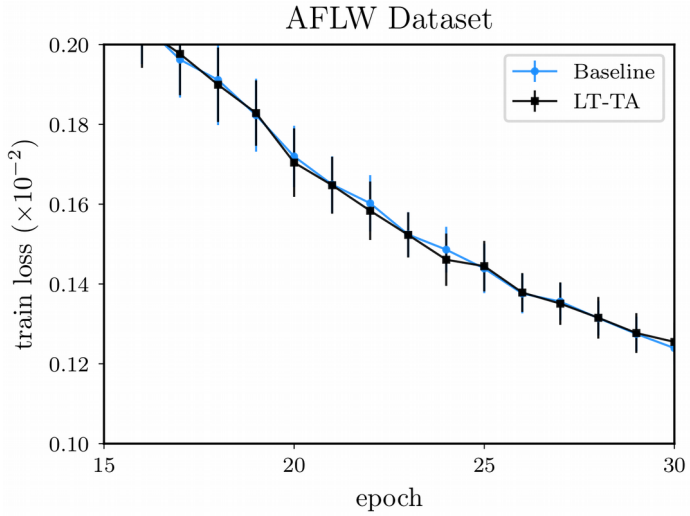

Figure 11: AFLW Evaluation: Loss during the optimization process. Figure best viewed in color.

The evaluation results are reported in Tables 18 and 19, while the corresponding learning curves are shown in Figure 11. The proposed method slightly improves the results all the other evaluated methods both for the metrics evaluated during the optimization and for the testing evaluation.

4.7 IMDB Evaluation

Finally, the proposed method was evaluated using the IMDB Movie review sentiment classification dataset [38]. The preprocessed dataset that contains 25,000 training reviews and 25,000 testing reviews, as supplied by the keras library [40], was used for the conducted experiments. A Long Short Term Memory (LSTM) network was used for the classification into the two possible sentiments [39]: positive and negative. The maximum length of a document was restricted to 500 words, while only the 10,000 most frequent words were used for constructing the word embedding model used for the classification. The network architecture used for the conducted experiments is shown in Table 20. Rectifier activation units [9] were used for the first fully connected layer, while the sigmoid activation function was used for the output layer. The network was trained using the binary cross-entropy loss.

Table 20: Network architecture used for the AFLW dataset (conv. refers to convolutional layers)

Layer Type	Output Shape
Input	500
Embedding Layer (10000 words, 32-dimensional vectors)	500 x 32
LSTM Layer (32 units)	32
Dense Layer (500 neurons)	500
Dense Layer (1 neuron)	1

The evaluation results for the train set are reported in Table 21, while the test evaluation results are reported in Table 22. The proposed method improves the train loss and error (the loss is reduced from 22.133 to 21.273, while the train error from 10.625% to 10.039%). The same behavior is also observed for the test loss and error (Table 22), as well as in the learning curves shown in Figure 12. Note that in this experiment a recurrent network architecture was used (LSTM) to evaluate the proposed method under a more challenging setup. Recall that recurrent networks are more prone to undesired phenomena that can cause instabilities during the training process, such as vanishing or exploding gradients. Even though the proposed method cannot directly mitigate such phenomena, biasing the network towards stabler states can improve both the stability of the training process as well as the training/testing accuracy, when such phenomena occur.

Table 21: IMDB Train Evaluation: Comparing the proposed LT-TA method to both the plain Adam (Baseline) and Temporal Averaging during the optimization.

Method	Train loss (x 10^{-2})	Train error (%)
Baseline	22.133 ± 5.927	10.625 ± 3.864
TA	31.138 ± 6.927	14.971 ± 4.862
LT-TA	21.273 ± 5.111	10.039 ± 3.245

Table 22: IMDB Test Evaluation: Comparing the proposed LT-TA method to both the plain Adam (Baseline) and Temporal Averaging using the supplied test set.

Method	Test loss (x 10^{-2})	Test error (%)
Baseline	48.457 ± 5.032	14.518 ± 0.941
TA	39.245 ± 3.075	15.736 ± 2.353
LT-TA	48.357 ± 5.739	14.318 ± 0.695

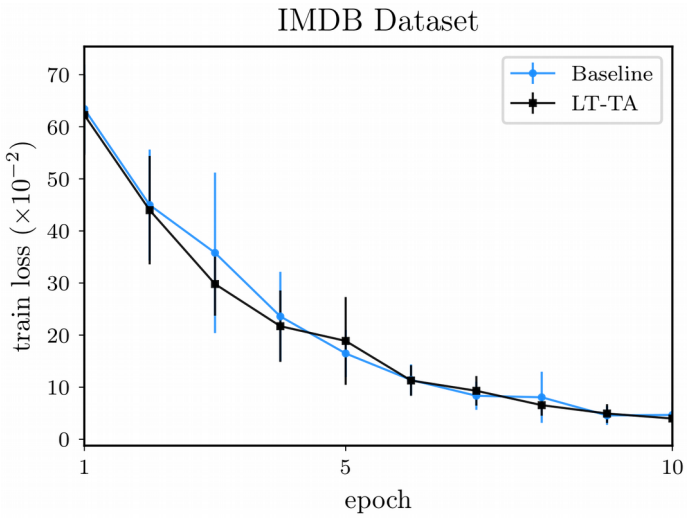

Figure 12: IMDB Evaluation: Loss during the optimization process. Figure best viewed in color.

4.8 Statistical Significance Analysis

The statistical significance of the obtained results was also evaluated using the Wilcoxon signed rank test *considering all the datasets* [41]. For conducting the statistical test we *directly* used the average loss values observed for the train and test sets using all the available datasets, as they are reported in Tables 4, 5, 6, 7, 9, 10, 12, 13, 15, 16, 18, 19, 21 and 22. By performing hypothesis testing on the data

collected from all the datasets we can conclude that the average loss of the proposed method is lower than the baseline approach (p-value = 0.008). The statistical test was conducted using the SciPy library [47].

5. Conclusions

In this work we proposed a Long-Term Temporal Averaging technique that first takes big descent steps to explore the solution space and then employs an exponential running averaging technique to bias the current parameters towards stabler states, if the optimization process becomes unstable. The more stable convergence of the algorithm also reduces the risk of stopping the training process when a bad descent step was taken and the learning rate was not appropriately set. It was demonstrated, using extensive experiments on six datasets and different evaluation setups, that the proposed technique can improve the behavior of stochastic optimization techniques for training deep neural networks.

Acknowledgements

The research leading to these results has been partially funded from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE). This publication reflects the authors' views only. The European Commission is not responsible for any use that may be made of the information it contains. The authors would like to thank the anonymous reviewers for their helpful and constructive comments that greatly contributed to improving the final version of this manuscript.

Compliance with ethical standards

Conflict of interest

The authors declare that they have no conflicts of interest.

Ethical approval

This article does not contain any studies with human participants or animals performed by any of the authors.

Bibliography

[1] Arunkumar, R., Karthigaikumar, P.: Multi-retinal disease classification by reduced deep learning features. Neural Computing and Applications 28(2), 329-334 (2017)
[2] Bengio, Y., Simard, P., Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks 5(2), 157-166 (1994)
[3] Chollet, F., et al.: Keras. https://github.com/fchollet/keras (2015)
[4] Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research 12(Jul), 2121-2159 (2011)
[5] Farzad, A., Mashayekhi, H., Hassanpour, H.: A comparative performance analysis of different activation functions in lstm networks for classification. Neural Computing and Applications pp. 1-15 (2017)
[6] Gourier, N., Hall, D., Crowley, J.L.: Estimating face orientation from robust detection of salient facial structures. In: FG NET Workshop on Visual Observation of Deictic Gestures (2004)
[7] Haykin, S., Network, N.: A comprehensive foundation. Neural Networks 2(2004), 41 (2004)
[8] Haykin, S.S., Haykin, S.S., Haykin, S.S., Haykin, S.S.: Neural networks and learning machines, vol. 3. Pearson Upper Saddle River, NJ, USA: (2009)
[9] He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 1026-1034 (2015)
[10] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of

the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770-778 (2016)
[11] Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: Proceedings of the International Conference on Machine Learning, pp. 448-456 (2015)
[12] Jiang, F., Grigorev, A., Rho, S., Tian, Z., Fu, Y., Jifara, W., Adil, K., Liu, S.: Medical image semantic segmentation based on deep learning. Neural Computing and Applications pp. 1-9 (2018)
[13] Kingma, D., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
[14] Koestinger, M., Wohlhart, P., Roth, P.M., Bischof, H.: Annotated facial landmarks in the wild: A large-scale, real-world database for facial landmark localization.
In: First IEEE International Workshop on Benchmarking Facial Image Analysis Technologies (2011)
[15] Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images (2009)
[16] LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature 521(7553), pp. 436-444 (2015)
[17] LeCun, Y., Cortes, C., Burges, C.J.: The mnist database of handwritten digits (1998)
[18] Van Hasselt, Hado, Arthur Guez, and David Silver: Deep Reinforcement Learning with Double Q-Learning." In: Proceedings of the AAAI Conference on Artificial Intelligence (2016)
[19] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y., Berg, A.C.: SSD: Single shot multibox detector. In: Proceedings of the European Conference on Computer Vision, pp. 21-37 (2016)
[20] Liu, Y., Gadepalli, K., Norouzi, M., Dahl, G.E., Kohlberger, T., Boyko, A., Venugopalan, S., Timofeev, A., Nelson, P.Q., Corrado, G.S., et al.: Detecting cancer metastases on gigapixel pathology images. arXiv preprint arXiv:1703.02442 (2017)
[21] Moulines, E., Bach, F.R.: Non-asymptotic analysis of stochastic approximation algorithms for machine learning. In: Proceedings of the Advances in Neural Information Processing Systems, pp. 451-459 (2011)
[22] Van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., Kavukcuoglu, K.: Wavenet: A generative model for raw audio. In: 9th ISCA Speech Synthesis Workshop, pp. 125-125
[23] Passalis, N., Tefas, A.: Concept detection and face pose estimation using lightweight convolutional neural networks for steering drone video shooting. In: Proceedings of the European Signal Processing Conference, pp. 71-75 (2017)
[24] Passalis, N., Tefas, A.: Improving face pose estimation using long-term temporal averaging for stochastic optimization. In: Proceedings of the International Conference on Engineering Applications of Neural Networks, pp. 194-204 (2017)
[25] Polyak, B.T., Juditsky, A.B.: Acceleration of stochastic approximation by averaging. SIAM Journal on Control and Optimization 30(4), pp. 838-855 (1992)
[26] Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 779-788 (2016)
[27] Ruppert, D.: Efficient estimations from a slowly convergent robbins-monro process. Tech. rep., Cornell University Operations Research and Industrial Engineering (1988)
[28] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision 115(3), pp. 211-252 (2015). DOI rm10.1007/s11263-015-0816-y
[29] Smolyanskiy, N., Kamenev, A., Smith, J., Birchfield, S.: Toward low-flying autonomous mav trail navigation using deep neural networks for environmental awareness. arXiv preprint arXiv:1705.02550 (2017)
[30] Srivastava, N., Hinton, G.E., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research 15(1), pp. 1929-1958 (2014)
[31] Srivastava, R.K., Greff, K., Schmidhuber, J.: Highway networks. arXiv preprint arXiv:1505.00387 (2015)
[32] Sutskever, I., Martens, J., Dahl, G., Hinton, G.: On the importance of initialization and momentum in

deep learning. In: Proceedings of the International Conference on Machine Learnin, pp. 1139-1147 (2013)
[33] Torralba, A., Fergus, R., Freeman, W.T.: 80 million tiny images: A large data set for nonparametric object and scene recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence 30(11), pp. 1958-1970 (2008)
[34 Wang, X., Guo, Y., Wang, Y., Yu, J.: Automatic breast tumor detection in abvs images based on convolutional neural network and superpixel patterns. Neural Computing and Applications pp. 1-13 (2017)
[35] Xiao, H., Rasul, K., Vollgraf, R.: Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms (2017)
[36] Yuxin, D., Siyi, Z.: Malware detection based on deep learning algorithm. Neural Computing and Applications pp. 1-12
[37] Zeiler, M.D.: Adadelta: an adaptive learning rate method. arXiv preprint arXiv:1212.5701 (2012)
[38] Maas, Andrew L., Daly, Raymond E.,, Pham, Peter T., Huang, Dan, Ng, Andrew Y. Potts, Christopher: Learning word vectors for sentiment analysis. Proceedings of the Annual Meeting of the Association for Computational Linguistics: Human Language Technologies pp. 142-150 (2011)
[39] Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation 9(8), pp. 1735-1780 (1997)
[40] Chollet, Francois, et al.: Keras. https://keras.io (2015)
[41] J. D. Gibbons, S. Chakraborti, Nonparametric statistical inference, Springer (2011)
[42] Anschel, Oron, Nir Baram, and Nahum Shimkin: Averaged-DQN: Variance Reduction and Stabilization for Deep Reinforcement Learning. In: Proceedings of the International Conference on Machine Learning (2017)
[43] Passalis, N., Tefas, A., I. Pitas: Efficient Camera Control using 2D Visual Information for Unmanned Aerial Vehicle-based Cinematography. In: Proceedings of the IEEE International Symposium on Circuits and Systems, pp. 1-5 (2018)
[44] Nousi, Paraskevi, and Anastasios Tefas. "Discriminatively trained autoencoders for fast and accurate face recognition." In: Proceedings of the International Conference on Engineering Applications of Neural Networks, pp. 205-2015 (2017)
[45] Nousi, Paraskevi, and Anastasios Tefas. "Self-supervised autoencoders for clustering and classification." Evolving Systems, pp. 1-14 (2018)
[46] Mademlis, Ioannis, et al. "Challenges in Autonomous UAV Cinematography: An Overview." In: Proceedings of the IEEE International Conference on Multimedia and Expo, (2018)
[47] Jones E, Oliphant E, Peterson P, et al. SciPy: Open Source Scientific Tools for Python, 2001, http://www.scipy.org/ (Online; accessed 2018-07-31)