ISSN 1060-992X, Optical Memory and Neural Networks (Information Optics), 2014, Vol. 23, No. 2, pp. 96–103. © Allerton Press, Inc., 2014.

Training Neural Networks for Classification Using the Extended Kalman Filter: A Comparative Study

A. N. Chernodub

The Institute for Mathematical Machines and Systems of the National Academy of Sciences of Ukraine, Kiev, Ukraine
e-mail: a.chernodub@gmail.com

Received February 09, 2014; in final form, April 04, 2014

Abstract—Feedforward Neural Networks training for classification problem is considered. The Extended Kalman Filter, which has been earlier used mostly for training Recurrent Neural Networks for prediction and control, is suggested as a learning algorithm. Implementation of the cross-entropy error function for mini-batch training is proposed. Popular benchmarks are used to compare the method with the gradient-descent, conjugate-gradients and the BFGS (Broyden–Fletcher–Goldfarb–Shanno) algorithm. The influence of mini-batch size on time and quality of training is investigated. The algorithms under consideration implemented as MATLAB scripts are available for free download.

Keywords: Neural Networks, Extended Kalman Filter, cross-entropy error function

DOI: 10.3103/S1060992X14020088

INTRODUCTION

Neural networks with a super large number of configurable parameters or deep neural networks lead in the state-of-the-art recognition systems [1, 2]. Because of a very large number of free parameters (there may be millions of them) the sequential (sample-by-sample) training algorithms can be used for this kind of models. Usually, a simple gradient descent method with unsupervised pre-training of neural network's layers is used.

However, one of the most effective sequential training algorithms is a Kalman Filter based family of algorithms. These algorithms are largely used to train neural networks to tackle with identification, prediction and control problems. In our research we would like to investigate the possible use of this kind of algorithms for classification problems, including neural networks with a large number of weights.

The Extended Kalman filter (EKF) was offered as far back as early 1980s [3] as an effective tool for training neural networks in the supervised learning paradigm. The EKF method provides better quality and a higher rate of convergence (by an order of magnitude) than the backpropagation method based on gradient descend. The reason is that the EKF algorithm takes into consideration the second-order information about the error surface, which is accumulated in the training process' covariance matrix—it is not used in the backpropagation method. In terms of the training quality this allows EKF-based training algorithms to be ranked together with such popular methods as the conjugate-gradients method, BFGS method, Levenburg–Marquardt method, etc. However, unlike of these approaches, which in fact are batch methods, EKF methods are of sample-by-sample nature. This provides additional advantages. First, the range of applications includes real-time tasks. Second, the chances that the training process comes to a full stop at a local minimum due to the stochastic component become smaller [6, p. 24]. Besides, in this case it is unnecessary to start the regularization procedure to minimize the risk of overfitting because as is shown in [5], regularization implicitly is included into the Kalman recursion.

The approach considered in the above-mentioned pioneer paper of S. Singhal and L. Wu [3] has been developed and given rise to a large class of methods which can be regarded as a Bayesian filtering for parameter estimation [5; 6, p. 759]. Decoupled EKF training methods [4, p. 33; 6, p. 855; 7] have been developed to decrease the amount of computations. Multi-stream training [4, p. 35; 8] and mini-batch training [9; 10] have been introduced to decrease the chances of getting into a local minimum. Researchers have developed new models of Kalman filters: Ensemble Kalman Filter (EnKF) [11], Unscented Kalman Filter (UKF) [4, p. 221; 12], Cubature Kalman Filter (CKF) [6, p. 787; 13] and their more stable variants based on the Cholesky decomposition [4, p. 273; 6, p. 773], etc.

Despite a notable advance in development of Kalman training algorithms, they are frequently used to tackle problems involving dynamic processes. These are the problems where first-order sample-by-sample methods are not effective and second-order batch training algorithms are inapplicable: training of recurrent neural networks [6, p. 850; 11; 14; 15] and real-time training of neural controllers [8; 14]. However, Kalman Filter based training can be used for different models of neural networks such as multilayer perceptrons, radial basis function networks, probabilistic neural networks and similar models, including networks for classification [4, p. 54]. Underlying the whole class of filters under consideration, the Kalman Filter interacts with the dynamics of a training process directly. The Kalman Filter training does not require dynamics in the input data at all.

Our research considers the use of the Extended Kalman Filter for training neural networks to solve classification problems. We give an algorithm for mini-batch training, and discuss the choice of the error function for classification. For experimental purposes we used some datasets from the UCI Machine Learning repository [16] and AT&T Laboratory (Cambridge) [17] and New York University [18] repositories. The sets include both relatively small samples and nearly as large and complicated as those used in training deep neural networks.

1. TRAINING FOR CLASSIFICATION USING THE EXTENDED KALMAN FILTER

Consider a training dataset containing input vectors $X = \{\mathbf{x}(n)\}_{n=1}^{N}$ and output target vectors $T = \{\mathbf{t}(n)\}_{n=1}^{N}$. The dimensionality of vectors T is equal to the number of classes: all components of the vectors are zero except one component which is value "1". The number of the unit component determines the number of the target class for classification.

Also, suppose we have a multilayer perceptron with one hidden layer. The network receives the input vector $\mathbf{x}(n) = \begin{bmatrix} x_1 & \dots & x_L \end{bmatrix}^T$ and generates the output $\mathbf{y}(n) = \begin{bmatrix} y_1 & \dots & y_K \end{bmatrix}^T$ using the following Eq: ($n = 1,\dots, N$):

$$z_j(n) = f\left(\sum_i w_{ji}^{(1)} x_i(n)\right), \tag{1}$$

$$y_o(n) = g\left(\sum_j w_{oj}^{(2)} z_j(n)\right), \tag{2}$$

where z_j is the values of post-synaptic potentials of the hidden layer, $\mathbf{w}^{(1)}$ are the weights of the hidden layer neurons, $f(\cdot)$ are the activation functions of the hidden layer, $\mathbf{w}^{(2)}$ are the weights of the output layer neurons, $g(\cdot)$ are the activation functions of the output layer.

During the training process, the Kalman Filter interacts with it's dynamics. Finding the best weights of the neural networks are regarded as an attempt to evaluate using the Kalman Filter the real state of an unknown "ideal" neural network that provides a zero residual. The state in this case is considered as weights $\mathbf{w}(n)$. The residual is the deviation of the mean-squared error of training $\mathbf{e}(n)$,

$$e_o(n) = t_o(n) - y_o(n), \quad o = 1,\dots, K. \tag{3}$$

This kind of dynamic training process can be described by a couple of equations in the state space (4) and (5). State equation (4) describes an evolution of the network's weights under the influence of random process $\boldsymbol{\xi}(n)$ which is regarded as white noise with a zero mean and known diagonal matrix $\mathbf{Q}$:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \boldsymbol{\xi}(n). \tag{4}$$

Equation for the output (4) is a linearized model of the net at cycle n affected by random process $\boldsymbol{\zeta}(n)$ which is also considered to be white noise with a zero mean and known diagonal covariance matrix $\mathbf{R}$:

$$\mathbf{h}(n) = \frac{\partial \mathbf{y}(\mathbf{w}(n), \mathbf{z}(n), \mathbf{x}(n))}{\partial \mathbf{w}} + \boldsymbol{\xi}(n), \tag{5}$$

where $\mathbf{w}(n)$ are the network's weights, $\mathbf{z}(n)$ are post-synaptic potentials of neurons, $\mathbf{x}(n)$ are input values of the network. The instantaneous values of derivatives $\frac{\partial \mathbf{y}(n)}{\partial \mathbf{w}}$ are computed by the back-propagation method by propagating value "1" at each backward run.

An initialization procedure is done before the training process starts. The covariance matrices of measurement noise $\mathbf{R} = \eta \mathbf{I}$ and dynamic training noise $\mathbf{Q} = \mu \mathbf{I}$ are set. The dimension of matrix $\mathbf{R}$ is $K \times K$,

matrix $\mathbf{Q}$ is $N_w \times N_w$ where K is the number of input neurons, N_w is the number of the network's weights. Coefficient η is inversely proportional to the training speed, coefficient μ defines the noise of measurement. In addition, covariance matrix $\mathbf{P} = c\mathbf{I}$ (in our paper $c = 100$), and zero measurement matrix $\mathbf{H}$ of size $K \times N_w$ are defined.

For each n-th element from the training dataset, $n = 1,\ldots,N$:

1. Forward pass: output vector $\mathbf{y}(n)$ is calculated.

2. Backward pass using the backpropagation method, and Jacobian matrix of observations $\mathbf{H}(n)$ is computed.

$$\mathbf{H}(n) = \begin{bmatrix} \frac{\partial y_1(n)}{\partial w_1} & \frac{\partial y_1(n)}{\partial w_2} & \cdots & \frac{\partial y_1(n)}{\partial w_{N_w}} \\ \frac{\partial y_2(n)}{\partial w_1} & \frac{\partial y_2(n)}{\partial w_2} & \cdots & \frac{\partial y_2(n)}{\partial w_{N_w}} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial y_K(n)}{\partial w_1} & \frac{\partial y_K(n)}{\partial w_2} & \cdots & \frac{\partial y_K(n)}{\partial w_{N_w}} \end{bmatrix}. \tag{6}$$

3. Current residual (3) is calculated and residual matrix $\mathbf{E}(n)$ is filled:

$$\mathbf{E}(n) = \left[e_1(n)\ e_2(n)\ \ldots\ e_K(n)\right]^T. \tag{7}$$

4. New network's weights $\mathbf{w}(n+1)$ and covariance matrices $\mathbf{P}(n+1)$ are calculated:

$$\mathbf{K}(n) = \mathbf{P}(n)\mathbf{H}(n)^T[\mathbf{H}(n)\mathbf{P}(n)\mathbf{H}(n)^T + \mathbf{P}(n)]^{-1}, \tag{8}$$

$$\mathbf{P}(n+1) = \mathbf{P}(n) - \mathbf{K}(n)\mathbf{H}(n)\mathbf{P}(k) + \mathbf{Q}, \tag{9}$$

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mathbf{K}(n)\mathbf{E}(n). \tag{10}$$

2. USING CROSS-ENTROPY AS AN ERROR FUNCTION

Training a neural network for classification aims at getting a robust estimation of a posteriori probabilities $P(\mathbf{y}(n)|\,\mathbf{x}(n))$, $n = 1,\ldots,N$ for classes presented in the training dataset (N is the total number of training samples).

Let there a logistic sigmoid $g(x) = \dfrac{1}{1+e^{-x}}$ was set as the activation function for K output neurons. Then we can introduce a mean-squared error (MSE):

$$E_{\mathrm{MSE}}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}\sum_{o=1}^{K}[t_o(n) - y_o(\mathbf{w},n)]^2. \tag{11}$$

If the neural network solves the classification problem, then it is also possible to determine a cross-entropy error (CE):

$$E_{\mathrm{CE}}(\mathbf{w}) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{o=1}^{K}[t_o(n)\ln y_o(\mathbf{w},n) + (1-t_o(n))\ln(1-y_o(\mathbf{w},n))]. \tag{12}$$

It can be shown [19, p. 100] that the actual distribution of a-posteriori probability is a global minimum for both MSE and CE. So, theoretically the neural network can be trained equivalently with both error functions: they both can approximate the actual a-posteriori distribution within any small error [20]. In practice the use of cross-entropy error often speeds up the convergence, which makes CE more popular in training classifiers [20].

In fact the Kalman filter method solves the problem of square error minimization: according to Eqs. (5)–(9) it is the mean-square error that is minimized when EKF is used. The following changes are needed to make EKF minimize the cross-entropy error.

Let us define a deviation of current cross-entropy error $\mathbf{c}(n)$ for the n-th element of the training sample and the o-th output neuron:

$$c_o(n) = \sqrt{-[t_o(n)\ln y_o(\mathbf{w},n) + (1-t_o(n))\ln(1-y_o(\mathbf{w},n))]}, \quad o = 1,\ldots,K. \tag{13}$$

Its derivative is:

$$\frac{\partial c_o(n)}{\partial w_i} = \frac{1}{c_o(n)}\frac{t_o(n)-y_o(\mathbf{w},n)}{y_o(\mathbf{w},n)(1-y_o(\mathbf{w},n))}\frac{\partial y_o(\mathbf{w},n)}{\partial w_i}, \quad i = 1,\ldots,N_w, \quad o = 1,\ldots,K. \tag{14}$$

Now we can fill the CE-adapted measurement matrix $\mathbf{H}_{\mathrm{CE}}(n)$ and residual matrix $\mathbf{E}_{\mathrm{CE}}(n)$:

$$\mathbf{H}_{\mathrm{CE}}(n) = \begin{bmatrix} \frac{\partial c_1(n)}{\partial w_1} & \frac{\partial c_1(n)}{\partial w_2} & \cdots & \frac{\partial c_1(n)}{\partial w_{N_w}} \\ \frac{\partial c_2(n)}{\partial w_1} & \frac{\partial c_2(n)}{\partial w_2} & \cdots & \frac{\partial c_2(n)}{\partial w_{N_w}} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial c_K(n)}{\partial w_1} & \frac{\partial c_K(n)}{\partial w_2} & \cdots & \frac{\partial c_K(n)}{\partial w_{N_w}} \end{bmatrix}. \tag{15}$$

$$\mathbf{E}_{\mathrm{CE}}(n) = \left[c_1(n)\ c_2(n)\ \ldots\ c_K(n)\right]^T. \tag{16}$$

3. IMPLEMENTATION OF MINI-BATCH TRAINING IN THE EKF METHOD

The simplest and most natural implementation of EKF-based training is an on-line sample-by-sample procedure where network weights are corrected for each new sample from the training dataset. However, with the EKF algorithms it is also possible to use mini-batch training when a few samples are used for single update of network weights [9, 10]. In this case it can be regarded as correction of single neural network's weights with $K \times B$ common outputs (where B is the number of training examples in a batch and K is the number of outputs of the actual network) or as correction of B "virtual" neural networks with shared weights. This kind of weights update is suboptimal for all samples in the mini-batch and is not equivalent to the averaging of unit updates in the sample-by-sample case. When a neural network is trained to tackle regression problems or neural control and streams are taken from different places of the training dataset, this trick becomes a popular Multistream EKF method [8], which is often effectively used for training recurrent networks to decrease chances of stopping in a local minimum.

In the case of mini-batch training, observation matrix $\mathbf{H}_{\mathrm{BATCH}}(n)$ takes the form:

$$\mathbf{H}_{\mathrm{BATCH}}(n) = \begin{bmatrix} \frac{\partial y_1(n)}{\partial w_1} & \frac{\partial y_1(n)}{\partial w_2} & \cdots & \frac{\partial y_1(n)}{\partial w_{N_w}} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial y_K(n)}{\partial w_1} & \frac{\partial y_K(n)}{\partial w_2} & \cdots & \frac{\partial y_K(n)}{\partial w_{N_w}} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial y_1(n+B-1)}{\partial w_1} & \frac{\partial y_1(n+B-1)}{\partial w_2} & \cdots & \frac{\partial y_1(n+B-1)}{\partial w_{N_w}} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial y_K(n+B-1)}{\partial w_1} & \frac{\partial y_K(n+B-1)}{\partial w_2} & \cdots & \frac{\partial y_K(n+B-1)}{\partial w_{N_w}} \end{bmatrix}. \tag{17}$$

Residual matrix $\mathbf{E}_{\mathrm{BATCH}}(n)$ for mini-batch training is

$$\mathbf{E}_{\mathrm{BATCH}}(n) = \left[e_1(n)\ \ldots\ e_K(n)\ \ldots\ e_1(n+B-1)\ \ldots\ e_K(n+B-1)\right]^T. \tag{18}$$

The size of matrix $\mathbf{H}_{\mathrm{BATCH}}(n)$ is $(K^*B) \times N_w$ the size of $\mathbf{E}_{\mathrm{BATCH}}(n)$ is $(K^*B) \times 1$, matrix $\mathbf{R}$ has the size $(K^*B) \times (K^*B)$ in this way. Other computations are similar to (8)–(10). When $B = 1$, the mini-batch training become sample-by-sample training.

Table 1. Datasets for experiments on classification

Name	Classes	Input vector dimensionality	Train samples count	Test samples count	Validation samples count	Total samples count
Cancer	2	9	456	139	104	699
Thyroid	3	21	4680	1440	1080	7200
Faces	40	50	200	120	80	400
Instrument	10	120	50000	16247	14904	81151

4. EXPERIMENTAL RESULTS

We carried out a few experiments to evaluate (1) training quality and speed for sample-by-sample and mini-batch variants of the EKF method; (2) the training quality and speed of EKF-based methods compared to the other popular training algorithms; (3) the effect of using mean-squared and cross-entropy errors in the EKF method on the quality of training. In the experiments we used the free NetLab software [21–23] containing popular Machine Learning algorithms for MATLAB. In order to make the experiments more illustrative, we have developed an additional software add-on "NetLab Kalman Filter Training" which allowed us to implement the Kalman optimization algorithms under consideration. The add-on can be seamlessly integrated to the NetLab to train the same models by changing the training algorithms only. Our add-on "NetLab Kalman Filter training" together with experimental data for this paper can be freely downloaded from Internet [24, 25].

Public datasets from UCI Machine Learning Repository and Cambridge AT&T Laboratories Repository (New York University) were used in the experiments.

Sets "Cancer" and "Thyroid" [16] are breast cancer and thyroid disease diagnostics data obtained by biopsy and other methods. Two target classes ("well" and "ill") are defined for "Cancer" set and three ("normal", sub-normal" and "hyperfunction") for "Thyroid" set. Taken from "ORL Faces Data base" [17], dataset "Faces" is for face recognition. This set has grayscale front face images of 40 people. Here the dimension of input vectors was reduced to 50 using the Principal Component Analysis [6, p.395; 23, p. 561]. Dataset "Instrument" [18] contains audio data recorded at a 20-Hz sample rate. Used for classifying 10 wind instruments, its input data is spectral data for ten 12-sample octaves. Set "Instrument" is the biggest dataset, number of samples is close to number of samples in MNIST-like deep datasets [1].

All the datasets were divided into the three groups: "Train", "Test" and "Validation" in proportion 65 : 20 : 15, except dataset "Faces" which has 5 examples for each person in the "Train" group, 3 and 2 images in "Test" and "Validation" groups. Table 1 gives the summary information about all datasets used in the experiments.

The Methodology of the Experiments

We used multilayer perceptrons that have one hidden layer and uses hyperbolic-tangent functions for hidden neurons and logistic sigmoid functions for output neurons. The number of neurons in the hidden layer varied from 5 to 40, the upper limit of this range depends on the complexity of a particular classification task. The initial weights of generated neural networks were initialized with small random values. To keep the integrity of the experiment, these initial values were the same as at the start for all training algorithms.

In training neural networks we used a "Save best" approach: regular intervals after a few epochs of training a neural network were tested using the "Test" sample; the network which provided the best result for the test dataset then was run on the "Validation" dataset. These results were assumed as final results and put to the tables below. Though the EKF method has more rapid convergence, each training stage takes a longer time. That is why for the EKF method we used 10 training epochs and tested a network after each epoch. For other methods we used 1000 training epochs and ran the network on the "Test" sample each 10 epochs. The number of epochs was chosen so big that the chances that the model picked by the "Save best" belongs to the last epoch are very small.

Starting Conditions

Default NetLab 3.3 parameters were used for all algorithms being tested. In experiments with EKF training algorithms we used the following parameters: learning rate $\eta = 10$, measurement noise $\mu = 10^{-6}$, covariance matrix initialization constant $c = 100$. The PC used in the experiment has a 3.5-GHz Intel Core i7 4770K processor and 16 GB of RAM, GPU was not used.

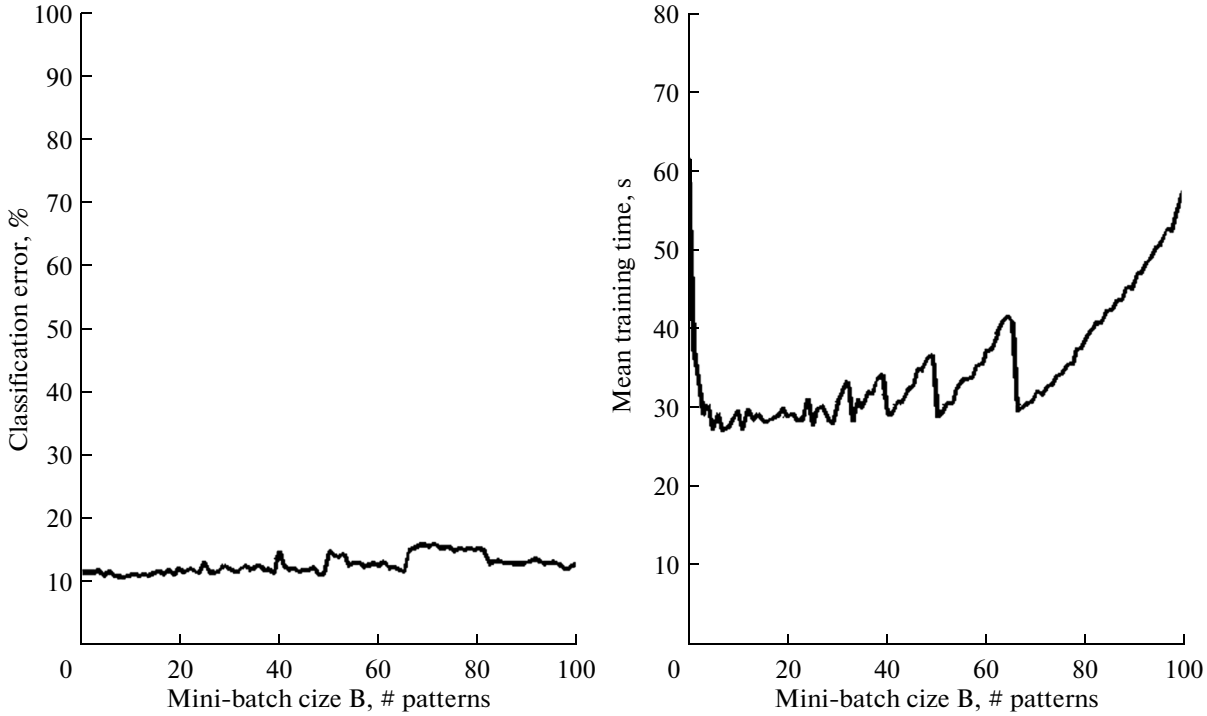

Fig. 1. The accuracy (left) and average time (right) of training versus the number of samples in the min-batch. The "Faces" dataset.

Mini-Batch Size Changing

The "Faces" dataset was used for this kind of experiments. For each experiment we trained 100 neural networks. The size of mini-batch (B) varied from 1 to 100. Figure 1 shows the quality and average time of training depending on the number of samples in the mini-batch.

It is seen from the left plot that the changing of mini-batch size does not produce a significant effect on the training quality, though the multistream technique (an analogue of mini-batch training for prediction problems) in which sampling is a continuous process is considered useful and applied widely. This can be explained by the fact that in the multistream technique we take data for a single batch from different parts of the dataset, thus preventing the final stop at the false minimum of the error surface. However, if we deal with the classification problem, the dataset is a collection of independent measurements which was already tossed on the data preprocessing step, so this doesn't produce an improvement of quality. Theoretically, EKF method's training time grows as $O(N_w^2)$ (where N_w^2 is the number of weights of the network) and is independent of the mini-batch size B because the most expensive operation is considered to be the inversion of $N_w \times N_w$ covariance matrix $\mathbf{P}$ in (8). In practice the size of the batch influences the training time (Fig. 1, right plot) in a complex way. It makes sense to experimentally determine the size of the mini-batch that provides the smallest training time, in the best case the time gain can be doubled. In our experiments the size of the mini-batch was $B = 10$.

Table 2. Classification accuracy of different training methods. Dataset "Cancer"

Training method	Best value, %	Mean value, %	Standard deviation, σ	Training time, s
Gradient Descent	0.96	7.24	13.77	1.5
Scaled Gradients	0.96	1.29	0.55	1.8
BFGS	0.96	1.51	0.75	3.1
EKF MSE	0.96	0.96	0.0	0.4
EKF CE	0.96	1.68	0.93	0.4

Table 3. Classification accuracy of different training methods. Dataset "Thyroid"

Training method	Best value, %	Mean value, %	Standard deviation, σ	Training time, s
Gradient Descent	6.76	6.76	0.00	9.2
Scaled Gradients	3.89	4.53	0.33	10.5
BFGS	2.04	3.69	1.32	32.1
EKF MSE	1.85	2.40	0.33	6.4
EKF CE	2.41	2.74	0.23	6.8

Table 4. Classification accuracy of different training methods. Dataset "Faces"

Training method	Best value, %	Mean value, %	Standard deviation, σ	Training time, s
Gradient Descent	8.75	24.54	11.62	1.8
Scaled Gradients	6.25	20.24	9.77	2.3
BFGS	7.50	55.27	28.67	56.3
EKF MSE	3.75	10.48	3.46	27.6
EKF CE	3.75	10.69	4.43	30.5

Table 5. Classification accuracy of different training methods. Dataset "Instruments"

Training method	Best value, %	Mean value, %	Standard deviation, σ	Training time, s
Gradient Descent	–	–	–	–
Scaled Gradients	7.26	8.74	1.39	246.3
BFGS	7.90	16.05	7.04	1594.9
EKF MSE	5.48	6.41	0.76	916.8
EKF CE	5.17	6.25	1.09	970.2

Testing Other Training Methods and Choosing the Error Function

We carried out a series of experiments in which we took mean-squared and cross-entropy errors to see how the choice of error function influences on the work of the EKF methods. We also tried other popular methods: gradient descent [21, p. 50], scaled conjugate gradients [21, p. 56], BFGS [21, p. 61] to compare them with the EKF approach. The experimental results for different datasets are presented in Tables 2–5. The datasets "Cancer", Thyroid" and "Faces" were used to train 100 neural networks.

CONCLUSIONS

We can draw the following conclusions.

(1) The Extended Kalman Filter method shows excellent results in training neural networks for classification. It is well comparable with popular second-order methods in training quality: the improvement can be as much as 1.5–2 times. When compared with the gradient descent, the EKF does still better; we reached a fourfold improvement.

(2) The EKF method outperforms other methods in training speed by several orders of magnitude, yet requires much more computations. In practice this quality can be very important, e.g. in real-time classification, including small-size samples.

(3) Mini-batch training does not provide a significant effect on training quality. However, it can reduce the training time by half if the mini-batch size is chosen properly.

(4) Popular in classification and tried on the datasets used for EKF, the cross-entropy error did not show neither noticeable gain, no loss in quality and time of training as compared with mean-square error.

(5) As a smart on-line implementation of the secondorder optimization method, the Kalman Filter training methods looks promising for deep learning problems. Yet the current implementation of the EKF

method does not fit for this kind of problems because they require the inversion of matrices holding thousands elements, which is not effective from the viewpoint of computation resources. Deep learning problems require a proper adaptation of Kalman Filter training methods, which can possibly involve significant approximations.

REFERENCES

1. Hinton, G., Deng, L., Yu, D., Dahl, G., Mohamed, A., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T., and Kingsbury, B., Deep neural networks for acoustic modeling in speech recognition, *IEEE Signal Processing Magazine*, vol. 29, no. 6, 2012, pp. 82–97.
2. Ciresan, D., Meier, U., Masci, J., and Schmidhuber, J., Multi-column deep neural network for traffic sign classification. *Neural Networks*, 2012, vol. 34, pp. 333–338.
3. Singhal, S. and Wu, L., Training multilayer perceptrons with the extended Kalman algorithm, *Advances in Neural Information Processing Systems 1*, Touretzky, D.S., Eds., San Mateo, CA: Morgan Kaufmann, 1989, pp. 133–140.
4. Haykin, S., *Kalman Filtering and Neural Networks,* John Wiley & Sons, 2001.
5. Arasaratnam, I. and Haykin, S., Nonlinear Bayesian filters for training recurrent neural networks, *Proc. 7th Mexican International Conference on Artificial Intelligence*, Atizapán de Zaragoza, Mexico, October 27–31, 2008; *Lecture Notes in Computer Science*, 2008, vol. 5317, pp. 12–33.
6. Haykin, S., *Neural Networks and Learning Machines, Third Edition*, New York: Prentice Hall, 2009.
7. Puskorius, G.V. and Feldkamp, L.A., Decoupled extended Kalman filter training of feedforward layered networks, *International Joint Conference on Neural Networks*, 1991, 8-14 Jul 1991, Seattle, vol. 1, pp. 771–777.
8. Puskorius, G.V. and Feldkamp, L.A., Training controllers for robustness: multi-stream DEKF, *Neural Networks*, 1994; *IEEE World Congress on Computational Intelligence*, 27 June–2 July 1996, Orlando, USA, vol. 6, pp. 2377–2382.
9. Li, S., Comparative analysis of backpropagation and extended Kalman filter in pattern and batch forms for training neural networks, *Proc. on International Joint Conference on Neural Networks (IJCNN '01)*, Washington, DC, July 15–19, 2001, vol. 1, pp. 144–149.
10. Chernodub, A.N., Direct method for training feed-forward neural networks using batch extended Kalman filter for multi-step-ahead predictions artificial neural networks and machine learning, *23rd International Conference on Artificial Neural Networks*, 10–13 September 2014, Sofia, Bulgaria (ICANN-2013), *Lecture Notes in Computer Science,* Berlin Heidelberg: Springer-Verlag, 2013, vol. 8131, pp. 138–145.
11. Mirikitani, D.T. and Nikolaev, N., Dynamic modeling with ensemble Kalman filter trained recurrent neural networks, *Seventh International Conference on Machine Learning and Applications (ICMLA'08)*, 11–13 Dec 2008, San-Diego, USA.
12. Wan, E.A. and van der Merwe, R., The unscented Kalman filter for nonlinear estimation, *Proc. of IEEE Symposium*, October 2000 (AS-SPCC), Lake Louise, Alberta, Canada, pp. 153–158.
13. Arasaratnam, I. and Haykin, S., Cubature Kalman filters, *IEEE Transactions on Automatic Control*, vol. 56, no. 6, pp. 1254–1269.
14. Prokhorov, D.V. Toyota Prius HEV neurocontrol and diagnostics, *Neural Networks*, 2008, no. 21, pp. 458–465.
15. Cernansky, M. and Benuskova, L., Simple recurrent network trained by RTRL and extended Kalman filter algorithms, *Neural Network World*, 2003, vol. 3, no. 13, pp. 223–234.
16. http://mlearn.ics.uci.edu/MLRepository.html
17. http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html
18. http://marl.smusic.nyu.edu/wordpress/projects/feature-learning-deep-architectures/deep-learning-python-tutorial/#data
19. Bourlard, H.A. and Morgan, N., *Connectionist Speech Recognition: a Hybrid Approach*. Norwell, MA, USA: Kluwer Academic Publishers, 1993.
20. Golik, P., Doetsch, P., and Ney, H., Cross-entropy vs. squared error training: a theoretical and experimental comparison, *14th Annual Conference of the International Speech Communication Association "Interspeech-2013"*, 25–29 August 2013, Lyon, France, pp. 1756–1760.
21. Nabney, I.T., *Netlab: Algorithms for Pattern Recognition* London: Springer, 2004.
22. http://www1.aston.ac.uk/eas/research/groups/ncrg/resources/netlab/downloads/
23. Bishop, C.M., *Pattern Recognition and Machine Learning,* Springer, 2006.
24. http://chernodub.com/kalman-netlab.zip
25. http://code.google.com/p/kalman-filter-training-for-netlab/