Training a digital model of a deep spiking neural network using backpropagation

V Bondarev[1,*]

[1]Sevastopol State University, Sevastopol, 299053, Russia

Abstract. Deep spiking neural networks are one of the promising event-based sensor signal processing concepts. However, the practical application of such networks is difficult with standard deep neural network training packages. In this paper, we propose a vector-matrix description of a spike neural network that allows us to adapt the traditional backpropagation algorithm for signals represented as spike time sequences. We represent spike sequences as binary vectors. This enables us to derive expressions for the forward propagation of spikes and the corresponding spike training algorithm based on the back propagation of the loss function sensitivities. The capabilities of the proposed vector-matrix model are demonstrated on the problem of handwritten digit recognition on the MNIST data set. The classification accuracy on test data for spiking neural network with 3 hidden layers is equal to 98.14%.

1 Introduction

In recent years, conventional deep neural networks have been widely used in solving various artificial intelligence problems: image classification, object detection, speech recognition, natural language processing and much more [1]. Training such deep neural networks typically requires powerful GPUs and computing clusters. Therefore, the application of conventional deep neural networks in areas with critical power consumption is problematic. An alternative are biologically inspired spiking neural networks (SNNs). SNNs had long shown great theoretical potential as efficient computing units [2], and recent advances in SNN hardware have renewed research interest in this area [3].

Deep SNNs are similar to conventional deep neural networks in terms of topology, but differ in neuron models. Spiking neurons possess memory and are characterized by a non-differentiable activation function [4]. Therefore, training algorithms based on backpropagation used in regular deep neural networks cannot be directly applied to SNNs.

There are three main approaches to train deep SNNs: conversion of a trained conventional deep neural network to SNN [5]; unsupervised learning based on local learning rules such as STDP [6]; direct SNN training based on the spike version of the backpropagation algorithm [7, 8]. The paper considers a digital model of a multilayer SNN and the corresponding spike training algorithm based on the standard backpropagation algorithm.

* Corresponding author: bondarev@sevsu.ru

To derive such a digital model and training algorithm, we use the vector-matrix description of a single-layer SNN, considered in [9-11]. The purpose of this paper is to expand the scope of the vector-matrix notation [9] to a multilayer SNN. This will allow us to make use of standard deep neural network training packages.

2 Analog model of spiking neuron

Several models of spiking neurons are known [12]. Here we use the simplified LIF (Leaky Integrate and Fire) neuron model. In accordance with this model, the state of the *i*-th neuron is determined by the value of its membrane potential $v_i(t)$. The dynamics of $v_i(t)$ at the time interval $t \in \left[t_i^{l-1}, t_i^l\right)$ between the output spikes of the neuron can be described as a convolution integral

$$v_i(t) = \int_{t_i^{l-1}}^{t} h_i\,(t-\tau) net_i(\tau) d\tau, \tag{1}$$

where $h_i(t)$ is the impulse response of the leaky integrator, t_i^{l-1} is the time of the output spike appearance at time moment *l*-1. If the value of the membrane potential $v_i(t)$ is greater or equal to the threshold *s*, then the neuron emits an output spike at time t_i^l and the integrator is reset to zero. The value of the network function $net_i(t)$ of the *i*-th neuron is represented as a sum of weighted input spikes arriving from other neurons in the network

$$net_i(t) = \sum_{j=1}^{J} w_{ij} \sum_l \delta(t - t_j^l), \tag{2}$$

where w_{ij} is the weight of the connection between neuron *i* and neuron *j*, $\delta(t - t_j^l)$ is a delta function representing the output spike of the *j*-th neuron at time t_j^l.

3 Feedforward model of SNN

To construct a digital feedforward SNN model, we perform discretization of the above expressions. Following the approach described in [9], we represent the spike sequences at time step *k* for all SNN inputs in the form of binary vectors $\mathbf{b}[k]$. The element $b_j[k]$ of the vector $\mathbf{b}[k]$ is equal to 1, if at step *k* there is a spike at the input *j*, otherwise it is equal to 0. Then from (1) and (2) we can derive the basic expressions of the digital SNN model for the forward propagation of spikes:

$$\mathbf{net}^m[k] = \mathbf{W}^m \mathbf{b}^{m-1}[k], \tag{3}$$

$$\mathbf{v}^m[k] = \beta \mathbf{v}^m[k-1] + \mathbf{net}^m[k], \tag{4}$$

$$b_i{}^m[k] = 1, \text{ if } v_i{}^m[k] \geq s; \text{ otherwise } b_i{}^m[k] = 0, \tag{5}$$

$$v_i{}^m[k^+] = 0, \text{ if } v_i{}^m[k] \geq s, \tag{6}$$

where *m* is the index of neural network layer, $\mathbf{W}^m$ is the weight matrix for the layer *m*, $\mathbf{v}^m[k]$ is the vector of neuron membrane potentials of the layer *m*, $\mathbf{net}^m[k]$ is the vector of network functions for the layer *m*, $\beta = e^{-T/\tau}$ is the leakage coefficient of the integrator (T is the time sampling step, τ is the time constant of the leaky integrator). The matrix $\mathbf{W}^m$ has size *I* x *J*, where *I* is the number of neurons in layer *m*, *J* is the number of neurons in layer *m*-1. The recursive formula (4) describes the dynamics of a leaky integrator. If membrane potential $v_i{}^m[k]$ is greater or equal to *s* then at the end of time step k^+ the integrator is reset to zero

(6), except the last hidden layer *M*. In the last hidden layer, weighted input spikes are simply accumulated to form the score values used by the classifier to make decisions.

4 Backpropagation model of SNN

Using the mean square error as the loss function we can derive an appropriate SNN training algorithm by analogy with the standard backpropagation approach, which is widely used for training multilayer neural networks.

The main problem which arises here is that the spike generation function (5) represented by the Heaviside step function $H(\mathbf{v}^m - s)$ is not differentiable, since at the moment of spike formation it has a unit jump. Therefore, for neurons of hidden layers (during backpropagation) it is necessary to approximate the spike generation function in the form of a function $f(\mathbf{v}^m - s)$, which has a derivative. For this, we can use various local functions described in [13].

It can be shown that in the case of applying the specified approximation for the SNN model (3-6) and minimizing the loss function based on stochastic gradient descent, the SNN weights will be corrected as follows:

$$\mathbf{W}^m[k] = \mathbf{W}^m[\mathrm{n}-1] - \alpha\Delta^{\boldsymbol{m}}[k](\mathbf{b}^{m-1}[k])^{\mathrm{T}}, \tag{7}$$

$$\Delta^{\boldsymbol{m}}[k] \approx f'(\mathbf{v}^m[k]-\mathrm{s})\,(\mathbf{W}^{m+1}[k])^{\mathbf{T}}\Delta^{\boldsymbol{m+1}}[k], \tag{8}$$

$$\Delta^{\boldsymbol{M}}[k] = -\mathbf{e}[k]/k, \tag{9}$$

where $\Delta^{\boldsymbol{m}}$ is the sensitivity vector of the loss function with respect to the vector of network functions $\mathbf{net}^m$ of the layer *m*, $f'(\mathbf{v}^m - s)$ is an approximation of the gradient of the spike generation function, *M* is the index of the last layer, **e** is the network error vector , α is the learning rate.

Expression (8) determines the sensitivity of layer *m* through the sensitivity of the next layer *m*+1, providing the backward propagation of the sensitivities, starting from layer *M*. In this case, the sensitivity vector $\Delta^{\boldsymbol{M}}$ of the last layer is determined by direct differentiation of the used loss function with respect to $\mathbf{net}^M$. For the loss function in the form of the mean square error, the sensitivity vector of the final layer is calculated in accordance with (9).

5 Results

To study the possibilities of the proposed model, we modeled a multilayer SNN with three fully connected hidden layers. The network has been trained to recognize images of handwritten digits in the MNIST training data set, which contains 60,000 training images and 10,000 test images. The network architecture is shown in figure 1. The number of hidden LIF neurons in each layer is depicted in figure 1. In total, the network contained 328,200 parameters

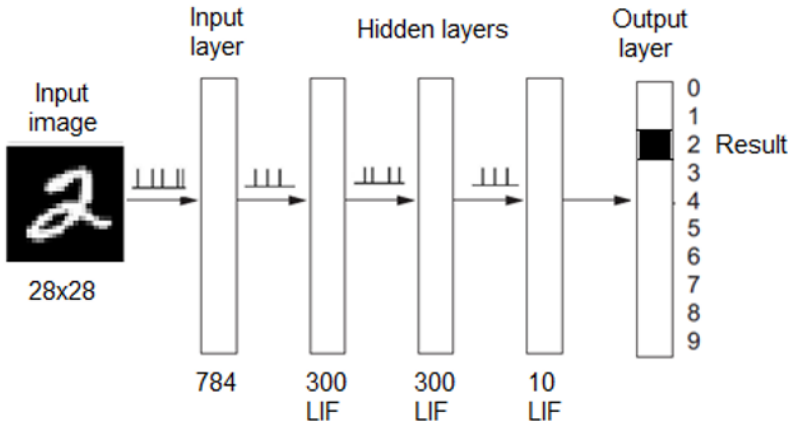

Fig. 1. Three-layer SNN architecture.

A spike sequences are generated from a 28 x 28 pixel image of a handwritten digit. The normalized intensity (in the range [0, 1]) of each pixel of the original image was converted into a spike stream with a Poisson distribution. For this, at each time step, the pixel intensity was compared with a random number with a uniform distribution. If the pixel intensity exceeded the value of a random number, then a spike was generated at the corresponding time step. As a result, spike sequences of a given length were formed with an average spike frequencies equal to the intensities of the input image pixels. The spike sequence length is up to 255 time steps. With a normalized pixel intensity of 0.5, the average number of spikes in the sequence is 127.

Figure 2 shows examples of recovering the original image using spike sequences of different lengths. Recovery was performed by counting the total number of spikes at each SNN input. Figure 2 demonstrates that an image reconstructed from a spike sequence of 20 steps can be successfully recognized. This can be used to reduce training time by limiting the length of the spike sequence.

Fig. 2. Image recovering from spike sequences of different lengths.

Forward propagation of input spikes was performed in accordance with the model (3-6). The output layer of the neural network uses the softmax classifier. The softmax output layer ensures that the values of each output neuron fall within the range [0, 1] and have a sum equal to 1. The loss function in this case is specified as the average cross-entropy

$$L = 1/\mathrm{N} \sum_{i=1}^{N} (-\ln(P(i,d))), \tag{10}$$

where $P(i,d)$ is the probability of belonging of the i-th image to the correct class d, N is the number of processed images. Calculation of $P(i,d)$ is carried out using the softmax function [1]

$$P(i,d) = \frac{\exp(v_d^M)}{\sum_{j=0}^{9} v_j^M}, \quad (11)$$

In this case, the elements of the vector Δ^M in (6) are equal to

$$\Delta_d{}^M = P(i,d)\text{-}1, \ \Delta_j{}^M = P(i,j), \quad (12)$$

where $P(i,j)$ is the probability of the i-th image belonging to the class j, $j \neq d$, j, $d \in 0..9$.

The derivative of the spike generation function was approximated using an exponentially decaying function

$$f'(\mathbf{v}^m - s) = \exp(-|\mathbf{v}^m - s|). \quad (13)$$

To optimize the neural network parameters, we used the adaptive moment estimation algorithm (Adam) with the parameters β1 = 0.9, β2 = 0.999 [14]. The training was carried out using mini-batches that were selected from the training set. The mini-batch size was equal to 64. The main SNN parameters were as follows: the initial learning rate α = 0.0001, the leakage coefficient of the integrator β = 0.8187, the number of neurons in the input layer 784, the number of neurons in hidden layers, respectively, 300, 300, and 10.

The length of the spike sequence was chosen based on the dynamics of the membrane potentials of neurons in the output layer, which is shown in figure 3. As follows from figure 3, after about 15-20 time steps the transition processes in the spiking neurons are finished. Therefore, during training the length of the spike sequences were equal to 20 time steps. During testing, the relative position of the trajectories of membrane potentials ordinary is not changed from the first step, i.e. the neuron of the output layer, which has the highest value of the membrane potential and fixes the class of belonging of the input image, has the same index throughout the entire length of the spike sequence. Therefore, during testing, we used reduced spike sequences of the 5 time steps length.

The initialization of the weights and thresholds of the neural network was carried out similarly to [7]. The neural network weight matrix was initialized with uniformly distributed random numbers from the range $[-\sqrt{3/h}, \sqrt{3/h}]$, where h is the number of neurons in the hidden layer. The initialization of the neuron thresholds was set with positive numbers: the threshold of all neurons in the first hidden layer was equal to $5\sqrt{3/h_1}$, and the threshold of neurons in the second layer was equal to $3\sqrt{3/h_2}$, where h_1 is the number of neurons in the first hidden layer, and h_2 is the number of neurons the second hidden layer.

Figure 4 shows SNN accuracy curves obtained on the training and test sets. The training was carried out during 20 epochs. It can be seen that the training accuracy reaches the value of 99.73%. The classification accuracy on test data reaches the value of 98.14%. This is comparable to conventional neural networks of similar architecture. A further progress in accuracy is possible by increasing the number of training epochs and applying regularization.

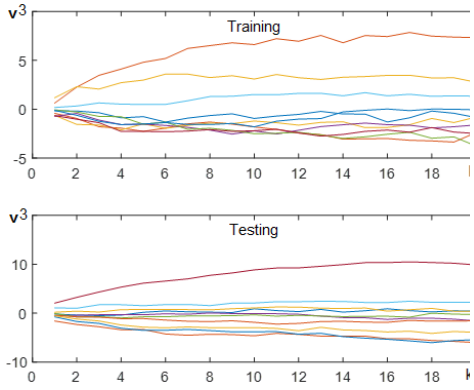

Fig. 3. Trajectories of membrane potentials of last layer neurons during training and testing.

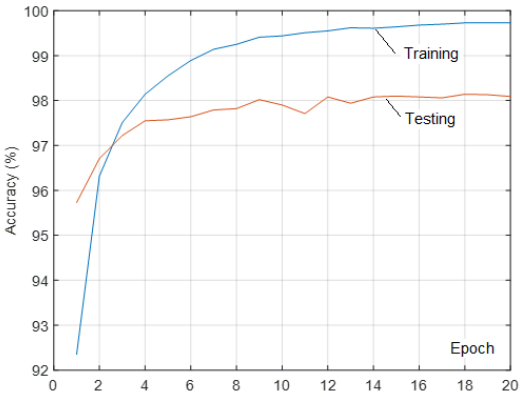

Fig. 4. Dependence of training and testing accuracy on the number of epochs.

6 Conclusion

A digital model of a spike neural network and a corresponding spike training algorithm based on error back propagation are proposed. The description of the digital SNN model and the algorithm for its training is performed in the generally accepted vector-matrix form, which allows further transfer of the known deep architectures of conventional neural networks to the SNN area. In particular, the given approach makes it possible to implement convolutional neural networks.

An example of the application of the proposed vector-matrix model for handwritten digit recognition on the MNIST training set is given. When trained during 20 epochs, the accuracy of the classification of handwritten numbers is 98.14%. This is comparable to the accuracy of the classification of conventional neural networks of similar architecture.

The general advantages of SNN over conventional neural network models are associated with lower power consumption, fast direct computation, and the possibility of online learning.

Deep SNNs are one of the promising concepts for signal processing of modern neuromorphic visual and audio sensors based on events. In combination with such sensors, SNN provides fast preliminary information processing. This means that the results of the network operation in the first approximation are available immediately after receiving several input spikes. It is shown, using the example of the MNIST set, that during testing, about 5 input spikes are enough to form conclusions. At the same time, deep SNN improves the classification accuracy as the number of processed input spikes increases. In this way, SNNs are able to get preliminary inference by several spikes in short time. The quality of such inference can be improved by later processed spikes.

References

1. Goodfellow I, Bengio Y and Courville A 2016 *Deep learning* (Cambridge, London: MIT Press)
2. Maass W 2002 *Models of Neural Networks. Early Vision and Attention* (New York: Springer) p 373
3. Davies M et al 2018 *IEEE Micro* **1(38)** 82–99

4. Shrestha S B and Orchard G 2018 *Advances in Neural Information Processing Systems* (Montréal, QC) p 1412
5. Hu Y, Tang H, Wang Y and Pan G 2018 *Spiking deep residual network* arXiv:1805.01352
6. Markram H, Lübke J, Frotscher M and Sakmann B 1997 *Science* **275** 213–15
7. Lee J H, Delbruck T and Pfeiffer M 2016 *Front. Neurosci* **10** 508
8. Lee C, Sarwar S S, Panda P, Srinivasan G and Roy K 2020 *Front. Neurosci.* **14** 119
9. Bondarev V 2016 *Lecture Notes in Computer Science* (Springer, Cham) p 647
10. Bondarev V 2018 *Studies in Computational Intelligence* (Springer, Cham) p 53
11. Bondarev V 2018 *Lecture Notes in Computer Science* (Springer, Cham)
12. Gerstner W and Kistler W M 2002 *Spiking neuron models: Single neurons, populations, plasticity* (Cambridge: Cambridge university press)
13. Neftci E O, Mostafa H and Zenke F 2019 *IEEE Signal Processing Magazine* **6(36)** 51–63
14. Kingma D P and Ba J 2014 *Adam: A method for stochastic optimization* arXiv:1412.6980