Deep neural network based on F-neurons and its learning

Yevgeniy Bodyanskiy
Xarkivs'kyj nacional'nyj universytet radioelektroniki

Serhii Kostiuk (serhii.kostiuk@nure.ua)
Kharkiv National University of Radio Electronics  https://orcid.org/0000-0003-4196-2524

Research Article

Keywords: Adaptive activation function, Deep neural network, Fuzzy activation function, F-neuron, Neofuzzy neuron

Posted Date: November 3rd, 2022

DOI: https://doi.org/10.21203/rs.3.rs-2032768/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License. Read Full License
Deep neural network based on F-neurons and its learning

Yevgeniy Bodyanskiy\textsuperscript{1†} and Serhii Kostiuk\textsuperscript{2*†}

\textsuperscript{1}Control Systems Research Laboratory, Kharkiv National University of Radio Electronics, Nauky Ave. 14, Kharkiv, 61166, Ukraine, ORCID ID: 0000-0001-5418-2143.

\textsuperscript{2*}Department of Artificial Intelligence, Kharkiv National University of Radio Electronics, Nauky Ave. 14, Kharkiv, 61166, Ukraine, ORCID ID: 0000-0003-4196-2524.

*Corresponding author(s). E-mail(s): serhii.kostiuk@nure.ua;
Contributing authors: yevgeniy.bodyanskiy@nure.ua;†These authors contributed equally to this work.

Abstract

Artificial neural networks are widely used in data processing and Data Mining. In contrast to traditional artificial neural networks, deep neural networks employ many artificial neuron blocks and more than three layers. An increase in the number of neuron blocks and layers improves the approximation capabilities but leads to the effects of exploding and vanishing gradients. Deep neural networks often employ piece-wise activation functions like ReLU to overcome the effects of exploding and vanishing gradients. We propose F-neuron as an adaptive alternative to piece-wise activation functions. Like ReLU, F-neuron does not suffer from the effects of exploding and vanishing gradient. F-neuron changes its form during the training, can approximate any currently used function, and synthesize new task-specific functions. We show that F-neuron can synthesize new task-specific functions and achieve higher approximation quality in existing neural network architectures. We evaluate the performance of the F-neuron on two image classification datasets (Fashion-MNIST and CIFAR-10) in two different architectures (LeNet-5 and KerasNet).

Keywords: Adaptive activation function, Deep neural network, Fuzzy activation function, F-neuron, Neo-fuzzy neuron

1 Introduction

Universal approximation capabilities and the ability to learn from the observed data allowed Artificial Neural Networks (ANNs) to become widely used in data processing in general and Data Mining in particular. The traditional ANN implementations are based on the elementary perceptron of F. Rosenblatt, employ sigmoidal activation functions (Cybenko, 1989) and generally have three or fewer layers of artificial neuron blocks (Cichocki and Unbehauen, 1993; Hornik, 1991). Deep neural networks (DNNs) were developed (LeCun et al, 2015; Schmidhuber, 2015; Goodfellow et al, 2016; Graupe, 2016) on the ideas of traditional ANNs to provide a higher approximation quality by increasing the number of layers and, hence, the number of trainable parameters in the network. This increase in the number of layers and parameters has led to significant computational difficulties in DNNs, namely the effects of vanishing and exploding gradients. The effects of vanishing and exploding gradients slow down the training process and, in extreme cases, cause the process to stop without
reaching an optimal set of values for the trainable parameters.

Sigmoidal functions are often replaced with piece-wise activation functions to solve computational difficulties in DNNs. Piece-wise activation functions, such as the rectified linear unit (ReLU) function (Goodfellow et al., 2016) and its modifications, do not cause the effects of vanishing and exploding gradients due to the properties of their derivative functions. At the same time, such piece-wise activation functions do not satisfy conditions of the G. Cybenko theorem (Cybenko, 1989), so the total number of layers and the number of neuron blocks in each layer shall be increased to provide the required approximation quality. As an alternative, it is possible to improve the approximation quality by replacing existing activation functions with functions that can adapt and change their form based on the current task, similarly to synaptic weights.

We propose a fuzzy neuron (F-neuron) as an adaptive alternative for the existing piece-wise activation functions. Section 2 defines the structure of F-neuron and its learning algorithm, Section 3 describes a multilayer network architecture based on F-neurons, Section 4 evaluates the performance of F-neuron comparing to ReLU in the established ANN implementations.

## 2 F-neuron and its learning algorithm

Fig. 1 shows the structure of the neural network node we name F-neuron. F-neuron is a hybrid of the traditional Adaline-type neuron (elementary perceptron) and Neo-Fuzzy Neuron (NFN) that is characterized by its high approximation capabilities as an independent online data processing system (Kolodyazhniy et al., 2005; Bodyanskiy et al., 2016). In addition, F-neuron is a simplified version of Double Neo-Fuzzy Neuron (DNFN) (Bodyanskiy et al., 2007). DNFN is based on NFN and has good properties, but DNFN is relatively complex from a computational standpoint.

F-neuron has two separate layers in its structure. The first layer contains a set of trainable synaptic weights that transforms the \((n + 1)\)-dimensional input vector \(x(k) = (1, x_1(k), ..., x_i(k), ..., x_n(k))^T\) (where \(k = 1, 2, ...,\)

![Fig. 1 F-neuron](image-url)

is a discrete-time index) into the internal activation signal:

\[
u_j(k) = \sum_{i=0}^{n} w_{ji}x_i(k) = w_j^T x(k).
\]

The second layer is a non-linear synapse \(NS_j\) of Neo-Fuzzy Neuron, which performs a non-linear transformation on the internal activation signal \(u_j(k)\). Essentially, the second layer implements fuzzy reasoning of zero order per Takagi–Sugeno. The second layer contains \(h\) non-linear membership functions \(\mu_j(k) = (\mu_{j1}(u_j(k)), ..., \mu_{jk}(u_j(k)), ..., \mu_{jh}(u_j(k)))^T\) and \(h\) trainable weights \(\bar{w}_j = (\bar{w}_{j1}, ..., \bar{w}_{ji}, ..., \bar{w}_{jk})^T\) so that the following signal appears on the F-neuron output:

\[
\hat{y}_j = \sum_{l=1}^{h} \bar{w}_{jl} \mu_j(k) = \bar{w}_j^T \mu_j(k). \tag{1}
\]

\(NS_j\) implements a discrete F-transform of I.Perfilieva (Perfilieva, 2004, 2006, 2015), so it provides an arbitrarily accurate approximation of the original activation function \(\psi_{\text{orig},j}(u_j)\). \(NS_j\) can approximate any generic activation function and synthesize new task-specific activation functions by tuning the weights vector \(\bar{w}_j\) according to the learning criterion \(E_j(k)\). The weights vector \(\bar{w}_j\) can be tuned in a sequential online mode (Bodyanskiy and Teslenko, 2007).

Thus F-neuron implements a non-linear transformation:

\[
\hat{y}_j(k) = \psi_j(u_j(k)) = \sum_{l=1}^{h} \bar{w}_{jl} \mu_j \left( \sum_{i=0}^{n} w_{ji} x_i(k) \right).
\]

Triangular functions of the following form are usually used (Yamakawa et al., 1992; Uchino...
and Yamakawa, 1994; Miki and Yamakawa, 1999) as the membership functions in the non-linear synapse $NS_j$:

$$
\mu_{jl}(u_j) = \begin{cases} 
\frac{u_j - c_{j,l-1}}{c_{j,l-1} - c_{j,l-1}} & \text{if } u_j \in [c_{j,l-1}, c_{j,l}], \\
\frac{c_{j,l+1} - u_j}{c_{j,l+1} - c_{j,l}} & \text{if } u_j \in [c_{j,l}, c_{j,l+1}], \\
0 & \text{otherwise}
\end{cases}
$$

or

$$
\mu_{jl}(u_j) = \begin{cases} 
\Delta_c^{-1}(u_j - c_{j,l-1}) & \text{if } u_j \in [c_{j,l-1}, c_{j,l}], \\
\Delta_c^{-1}(c_{j,l+1} - u_j) & \text{if } u_j \in [c_{j,l}, c_{j,l+1}], \\
0 & \text{otherwise}
\end{cases}
$$

If the internal activation signal belongs to the $u_{j,\min} \leq u_j(k) \leq u_{j,\max}$ interval, then $c_{ji} = u_{j,\min}$, $c_{jh} = u_{j,\max}$ and $\Delta_c = (h-1)^{-1}(c_j - c_j)$, where $c_j$ is the center coordinate of the membership function $\mu_{jl}(u_j)$.

Membership functions (2) implement a unity (Ruspini) partitioning of the input space:

$$
\mu_{j, l-1}(u_j) + \mu_{jl}(u_j) + \mu_{jl+1}(u_j) = 1,
$$

so non-linear transformation (1) can be expressed in the following form for $u_j(k) \in [c_{jl}, c_{jl+1}]$:

$$
\hat{y}_j(k) = \sum_{l=1}^{h} \tilde{w}_{jl} \mu_{jl}(u_j(k))
= \tilde{w}_{jl} \mu_{jl}(u_j(k)) + \tilde{w}_{j,l+1} \mu_{jl+1}(u_j(k))
= \Delta_c^{-1}(c_{j,l+1} - c_{j,l}) \tilde{w}_{jl} + \Delta_c^{-1}(\tilde{w}_{j,l+1} - \tilde{w}_{jl}) u_j(k)
= v_{j0,l,l+1} + v_{j1,l,l+1} u_j(k)
= v_{j0,l,l+1} + v_{j1,l,l+1} \sum_{i=0}^{n} w_{ji} x_i(k).
$$

The training process complexity does not depend on the number of membership functions $(h)$ in the non-linear synapse. It can be seen from the expression that only the two adjacent membership functions $(\mu_{jl}(u_j(k)))$ and $(\mu_{jl+1}(u_j(k)))$ fire simultaneously, so only two synaptic weights $(\tilde{w}_{jl}$ and $\tilde{w}_{j,l+1})$ shall be tuned at any moment.

While the total number of synaptic weights is $n + h + h$, only $n + 3$ synaptic weights shall be tuned for an F-neuron at any moment. By comparison, $n + 1$ synaptic weights shall be tuned for an elementary perceptron of F. Rosenblatt.

Thus the relationship between weights $\tilde{w}_{jl}$, $\tilde{w}_{j,l+1}$ and $v_{j0,l,l+1}$, $v_{j1,l,l+1}$ is defined by simple expressions:

$$
\begin{align*}
\tilde{w}_{jl} &= v_{j0,l,l+1} + c_j v_{j1,l,l+1}, \\
\tilde{w}_{j,l+1} &= v_{j0,l,l+1} + c_j v_{j1,l,l+1}.
\end{align*}
$$

F-neuron’s training process is based on the minimization of the traditional local quadratic criterion

$$
E_j(k) = \frac{1}{2} e_j^2(k) = \frac{1}{2} \left( y_j(k) - v_{j0} - v_{j1}^T x(k) \right)^2
= \frac{1}{2} \left( y_j(k) - v_{j0} - v_{j1} u_j(k) \right)^2
= \frac{1}{2} \left( y_j(k) - \tilde{w}_j^T \mu_j(k) \right)^2,
$$

where $y_j(k)$ is a reference signal and indexes $l$, $l_1$ of $v_{j0,l,l+1}$, $v_{j1,l,l+1}$ are omitted for brevity.

Weights $\tilde{w}_j$ of the non-linear synapse $NS_j$ can be tuned according to the gradient procedure:

$$
\tilde{w}_{jl}(k) = \tilde{w}_{jl}(k-1) + \tilde{\eta}(k)
\times \left( y_j(k) - \sum_{l=1}^{h} \tilde{w}_{jl}(k-1) \mu_{jl}(u_j(k)) \right)
\times (\mu_{jl}(u_j(k)),
$$

or in a vector form:

$$
\tilde{w}_j(k) = \tilde{w}_j(k-1) + \tilde{\eta}(k)
\times (y_j(k) - \tilde{w}_j^T (k-1) \mu_j(k)) \mu_j(k).
$$

Here the learning rate parameter $\tilde{\eta}(k)$ is selected as (Bodyanskiy et al, 2001; Otto et al, 2003)

$$
\tilde{\eta}^{-1}(k) = \tilde{\tau}(k) = \alpha \tilde{\tau}(k-1) + ||\mu_j(k)||^2,
$$
where $0 \leq \alpha \leq 1$ is a forgetting factor that defines a balance between the training process’s filtering and tracking properties.

The computations can be simplified by only tuning two synaptic weights ($v_{j0}$ and $v_{j1}$) on each discrete time step $k$:

\[ v_{j0}(k) = v_{j0}(k-1) + \tilde{\eta}(k) \times (y_j(k) - v_{j0}(k-1) - v_{j1}(k-1)u_j(k)) \]
\[ = v_{j0}(k-1) + \tilde{\eta}(k)e_j(k), \]
\[ v_{j1}(k) = v_{j1}(k-1) + \tilde{\eta}(k) \]
\[ \times (y_j(k) - v_{j0}(k-1) - v_{j1}(k-1)u_j(k)) \times u_j(k) \]
\[ = v_{j1}(k-1) + \tilde{\eta}(k)e_j(k)u_j(k), \]

or in a vector form:

\[ v_j(k) = v_j(k-1) + \tilde{\eta}(k)e_j(k)\tilde{u}_j(k), \]

where $v_j(k) = (v_{j0}(k), v_{j1}(k))^T$, $\tilde{u}_j(k) = (1, u_j(k))^T$.

\section{3 Multilayer perceptron based on F-neurons}

Without losing generality, we demonstrate the learning process on a 3-layer architecture, illustrated in Fig. 2. The architecture contains $n_0 = n$ inputs, $n$ F-neurons in the first hidden layer, $n_2$ in the second, and $n_3 = m$ in the output layer. Each input element is represented by an $(n+1) \times 1$-vector $x(k) = (1, x_1(k), x_2(k), ..., x_n(k))^T$, the output signal – by an $(m \times 1)$-vector $\hat{y}(k) = (\hat{y}_1(k), \hat{y}_2(k), ..., \hat{y}_m(k))^T$, and the reference signal – by an $(m \times 1)$-vector $y(k) = (y_1(k), y_2(k), ..., y_m(k))^T$.

We use the following function as the learning criterion:

\[ E(k) = \sum_{j=1}^{m} E_j(k) = \frac{1}{2} \sum_{j=1}^{m} c_j^2(k) \]
\[ = \frac{1}{2} \sum_{j=1}^{m} (y_j(k) - \hat{y}_j(k))^2. \]

The learning criterion is minimized by tuning synaptic weights $w_{ji}^{[s]}$ and $v_{ji}^{[s]}$, where $i = 0, 1, ..., n_s$, $q = 0, 1, ..., n_s$ is the number of neurons in the $s$-th layer, and $s = 1, 2, 3$ is the layer index.

The training process is based on the error backpropagation procedure. In contrast to the standard procedure, the training process for an F-neuron includes tuning the activation function parameters (weights) in the non-linear synapses $N_j^{[s]}$.

In the general case, the training procedure can be defined as

\[ \begin{cases} v_{jq}^{[s]}(k) - v_{jq}^{[s]}(k-1) = \Delta v_{jq}^{[s]}(k) \\ w_{ji}^{[s]}(k) - w_{ji}^{[s]}(k-1) = \Delta w_{ji}^{[s]}(k) \\ \end{cases} \]

where the base step is the computation of partial derivatives $\frac{\partial E(k)}{\partial v_{jq}^{[s]}}$, $\frac{\partial E(k)}{\partial w_{ji}^{[s]}}$.

The training procedure for the output layer is the simplest from the computational perspective and is defined by the following relationship:
The base step here is the computation of the partial derivative \( \frac{-\delta E(k)}{\delta o_j^2} \) used in equation (3), which defines the \( \delta_j^2 \)-error value (4).

In can be shown that

\[
\frac{-\partial E(k)}{\partial o_j^2} = -\sum_{i=1}^{n_3} \frac{\partial E(k)}{\partial u_i^3} \cdot \frac{\partial u_i^3}{\partial o_j^2}
\]

\[
= \sum_{i=1}^{n_3} \left( -\frac{\partial E(k)}{\partial u_i^3} \right) \cdot \frac{\partial}{\partial o_j^2} \left( \sum_{i=1}^{n_2} w_i^{[1]}(k) o_i^2(k) \right)
\]

\[
= \sum_{i=1}^{n_3} \delta_i^{[3]}(k) \cdot \frac{\partial}{\partial o_j^2} \left( \sum_{i=1}^{n_2} w_i^{[1]}(k) o_i^2(k) \right)
\]

\[
= \sum_{i=1}^{n_3} \delta_i^{[3]}(k) w_i^{[3]}(k),
\]

whence

\[
\delta_j^{[2]}(k) = \frac{\partial o_j^2}{\partial u_j^2} \sum_{i=1}^{n_3} \delta_i^{[3]}(k) \cdot w_i^{[3]}(k)
\]

\[
= o_j^2(k) \sum_{i=1}^{n_3} \delta_i^{[3]}(k) w_i^{[3]}(k),
\]

where

\[
\delta_j^{[2]} = -\frac{\partial E(k)}{\partial u_j^2} = -\frac{\partial E(k)}{\partial o_j^2} \cdot \frac{\partial o_j^2}{\partial u_j^2}.
\]
\[ v^{[2]}_{jq}(k) = v^{[2]}_{jq}(k - 1) + \eta^{[2]}(k) \left( \sum_{i=1}^{n_3} \delta^{[3]}(k) w^{[3]}_{ij}(k) \right) \]

\[ \times \begin{cases} 1 & \text{if } q = 0, \\ u^{[2]}_{ij}(k) & \text{if } q = 1. \end{cases} \]

Training of the first hidden layer is performed according to the relation:

\[
\begin{align*}
\Delta w^{[1]}_{ij}(k) &= \eta^{[1]}(k) \left( \sum_{l=1}^{n_2} \delta^{[2]}(k) w^{[2]}_{kl}(k) \right) \\
&\times \begin{cases} 1 & \text{if } q = 0, \\ u^{[1]}_{jk}(k) & \text{if } q = 1. \end{cases}
\end{align*}
\]

\[
\begin{align*}
\Delta w^{[1]}_{ji}(k) &= \eta^{[1]}(k) \delta^{[1]}(k)x_i(k), \\
\delta^{[1]}(k) &= v^{[1]}_{ji}(k) \sum_{i=1}^{n_2} \delta^{[2]}(k) w^{[2]}_{ij}(k). 
\end{align*}
\]

In the case of a DNN, when the neural network contains many hidden layers, the training process for the \( s \)-th layer is implemented according to the general relation:

\[
\begin{align*}
\Delta v^{[s]}_{jq}(k) &= \eta^{[s]}(k) \left( \sum_{i=1}^{n_{s+1}} \delta^{[s+1]}(k) w^{[s+1]}_{ij}(k) \right) \\
&\times \begin{cases} 1 & \text{if } q = 0, \\ u^{[s]}_{ij}(k) & \text{if } q = 1. \end{cases} \\
\Delta w^{[s]}_{ij}(k) &= \eta^{[s]}(k) \delta^{[s]}(k) \alpha_{j}^{[s+1]}(k), \\
\delta^{[s]}(k) &= v^{[s]}_{ji}(k) \sum_{i=1}^{n_{s+1}} \delta^{[s+1]}(k) w^{[s+1]}_{ij}(k). 
\end{align*}
\]

The main difference between the proposed neural network and the traditional multilayer perceptron with an arbitrary number of hidden layers is that the proposed network tunes its activation function during training by adapting its form to the defined task.

4 Experiments

We compare performance of the F-neuron to performance of the regular ReLU function in two convolutional neural network (CNN) implementations: LeNet-5 (Lecun et al., 1998) and KerasNet (Chollet et al., 2015). The baseline implementations use the ReLU non-linearity for all convolutional and fully-connected layers except the last fully-connected layer. Derived network variants use F-neuron as the activation function in the second-to-last fully-connected layer. All other properties of the network and the training procedure remain the same between the baseline and derived implementations.

The experiment is implemented in Python 3.8 using PyTorch 1.10.2 (Paszke et al., 2019). The implementation is publicly available on GitHub: github.com/s-kostyuk/f-neuron/tree/v1.0.0.

4.1 Dataset

The performance of the CNN implementations was evaluated on the Fashion-MNIST (Xiao et al., 2017) and CIFAR-10 (Krizhevsky, 2009) datasets.

The Fashion-MNIST dataset contains 60000 data items, and we use the 5:1 split between the training and evaluation sub-sets. Each data item carries a 1x28x28 vector of floating-point values (image pixel values) in the \([0.0,1.0]\) range and a class label corresponding to the vector in the \([0,9]\) range. Each class label is transformed into a vector with one-hot encoding to be compatible with the evaluated neural network implementation. We augment the dataset by randomly shifting the input vectors and applying a random horizontal flip to every second vector in the dataset.

The CIFAR-10 dataset contains 60000 data items with a 5:1 split between the training and evaluation sub-sets. Each data item carries a 3x32x32 vector of floating-point values, normalized to the \([0.0,1.0]\) range, and a corresponding class label in the \([0,9]\) range. The class labels are transformed into one-hot vectors for training and evaluation. We use the same data augmentation methods described earlier for the Fashion-MNIST dataset.

4.2 Network architecture

LeNet-5 and KerasNet are instances of a convolutional neural network with different layer arrangements and a different number of trainable parameters. Due to a simpler architecture, the LeNet-5 implementation is easier to train but generally provides worse approximation capabilities than a more complex KerasNet.

The baseline implementation of LeNet-5 consists of 4 layers:
1. 5x5 convolution with 20 output channels, 2x2 max-pooling, ReLU activation;
2. 5x5 convolution with 50 output channels, 2x2 max-pooling, ReLU activation;
3. fully-connected layer with 500 output features, ReLU activation;
4. fully-connected layer with 10 output features, SoftMax activation.

The combined count of trainable parameters across all layers in the baseline implementation depends on the input dataset: 431 000 for Fashion-MNIST and 657 000 for CIFAR-10.

The ReLU activation in the third layer is replaced by an F-neuron in the derivative implementation of LeNet-5. The F-neuron has 14 trainable weight parameters corresponding to 12 triangular membership functions on the [-1.0, +1.0] range, an inverse ramp function on the (-∞, -1.0) range, and a positive ramp function on the (+1.0, +∞) range. We use a separate F-neuron instance per each output feature in the layer, so the total number of F-neuron parameters in the LeNet-5 implementation is 7000.

The total count of trainable parameters in the derived LeNet-5 implementation with F-neurons is 438 000 for Fashion-MNIST and 664 000 for CIFAR-10.

The baseline implementation of KerasNet consists of 6 layers:
1. 3x3 convolution with 32 output channels and 1x1 padding, ReLU activation;
2. 3x3 convolution with 32 output channels and no padding, ReLU activation, 2x2 max pooling, and dropout (p = 0.25);
3. 3x3 convolution with 64 output channels and 1x1 padding, ReLU activation;
4. 3x3 convolution with 64 output channels and no padding, ReLU activation, 2x2 max pooling, and dropout (p = 0.25);
5. fully-connected layer with 512 output features, ReLU activation, and dropout (p = 0.5);
6. fully-connected layer with 10 output features, SoftMax activation.

The combined count of trainable parameters in the baseline implementation is 897 002 for Fashion-MNIST and 1 258 026 for CIFAR-10, correspondingly.

### 4.3 Training procedures and initialization

We train the LeNet-5 and KerasNet implementations separately on the Fashion-MNIST and CIFAR-10 datasets for 100 epochs with a batch size of 100. We use the RMSprop optimizer with the initial learning rate of 1e-4 and decrease the learning rate by 1e-6 on each minibatch.

We evaluate three different methods of the F-neuron parameter initialization:
- ramp function – member function weights uniformly increasing from -1.0 to +1.0;
- constant initialization – all weights set to 1.0;
- random initialization – uniformly selected random numbers between -1.0 and +1.0.

Initialization of all other parameters (weights) in all network variants is uniformly random.

### 4.4 Analysis of results

We record each network variant’s training set loss and the test set accuracy. The network variants with F-neurons outperform the baseline implementations with ReLU on both the CIFAR-10 and Fashion-MNIST datasets. The random F-neuron parameter initialization shows the best results for LeNet-5 on Fashion-MNIST, KerasNet on Fashion-MNIST, and KerasNet on CIFAR-10. Table 1 summarizes the best-achieved training results for each of the evaluated network variants, datasets, and parameter initialization options.

We note that the KerasNet implementation with random initial F-neuron parameter values trains faster and shows less tendency to overfit on both Fashion-MNIST and CIFAR-10 datasets compared to other implementations. Fig. 3 illustrates the dependency between the training loss, the test set error, and the training epoch for the KerasNet network trained on the CIFAR-10 dataset.
Table 1 Best test set accuracy, up to 100 epochs

<table>
<thead>
<tr>
<th>Network</th>
<th>Activ.</th>
<th>Init.</th>
<th>Fashion-MNIST</th>
<th>CIFAR-10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Acc.</td>
<td>Epoch</td>
<td>Acc.</td>
<td>Epoch</td>
</tr>
<tr>
<td>LeNet-5</td>
<td>ReLU</td>
<td>N/A</td>
<td>91.36%</td>
<td>76.39%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>100</td>
<td>95</td>
</tr>
<tr>
<td>LeNet-5</td>
<td>Fuzzy</td>
<td>Ramp</td>
<td>91.31%</td>
<td>77.42%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>92</td>
<td>88</td>
</tr>
<tr>
<td>LeNet-5</td>
<td>Fuzzy</td>
<td>Random</td>
<td>92.39%</td>
<td>76.99%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>91</td>
<td>78</td>
</tr>
<tr>
<td>KerasNet</td>
<td>ReLU</td>
<td>N/A</td>
<td>91.48%</td>
<td>79.67%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>85</td>
<td>95</td>
</tr>
<tr>
<td>KerasNet</td>
<td>Fuzzy</td>
<td>Ramp</td>
<td>92.13%</td>
<td>79.96%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>100</td>
<td>95</td>
</tr>
<tr>
<td>KerasNet</td>
<td>Fuzzy</td>
<td>Random</td>
<td>93.32%</td>
<td>82.68%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>91</td>
<td>95</td>
</tr>
</tbody>
</table>

Fig. 3 Dependency between the loss, accuracy and the training epoch for the KerasNet network on CIFAR-10

F-neuron changes its activation function form during the training process. F-neurons initialized as ramp functions keep their general form after training on the Fashion-MNIST dataset but are strongly deformed after training on the CIFAR-10 dataset. Fig. 4 illustrates the form of randomly selected F-neurons after training on different datasets.

F-neurons initialized with random weights and initialized with constant weights show a highly non-linear form after training on any dataset. Fig. 5 illustrates the form of randomly selected F-neurons after training on different datasets.

5 Conclusion

In this paper, we presented F-neuron as an alternative for the standard neurons with piece-wise activation functions, such as ReLU. The main difference between the network with F-neurons and the elementary perceptron of F. Rosenblatt is the usage of non-linear synapses of the neo-fuzzy neuron instead of the fixed non-linear activation functions. Synapses of the neo-fuzzy neuron implement a discrete F-transform in its adaptive form and allow approximation of the existing activation functions with an arbitrary level of accuracy. The activation function form is tuned to the current task for each of the F-neurons in the network, allowing the network to optimize the learning criterion and improve the quality of the resulting solution.

References

Fig. 4 The activation function form of F-neurons initialized as ramp functions

Fig. 5 The activation function form of F-neurons in LeNet-5 initialized with random or constant weights


**Statements and Declaration**

**Funding** The authors declare that no funds, grants, or other support were received during the preparation of this manuscript.

**Competing Interests** The authors have no relevant financial or non-financial interests to disclose.

**Author Contributions** All authors contributed to the study conception and design. Initial draft and formal analysis by Yevgeniy Bodyankiy. Editing, software implementation and experimental evaluation by Serhii Kostiuk. Illustrations and final manuscript by Yevgeniy Bodyankiy and Serhii Kostiuk. All authors read and approved the final manuscript.

**Data Availability** The datasets analysed during the current study are available in the corresponding repositories: https://github.com/zalandoresearch/fashion-mnist for Fashion-MNIST and https://www.cs.toronto.edu/~kriz/cifar.html for CIFAR-10. Raw training results are available from the corresponding author on reasonable request.

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