Connected Hidden Neurons (CHNNet): An Artificial Neural Network for Rapid Convergence

by

Rafiad Sadat Shahir
20101580
Zayed Humayun
20141030
Mashrufa Akter Tamim
20101586
Shouri Saha
20101349

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science

Department of Computer Science and Engineering
School of Data and Sciences
Brac University
September 2023

Declaration

It is hereby declared that

1. The thesis submitted is our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

Rafiad Sadat Shair	Zayed Humayun
20101580	20141030
Mashrufa Akter Tamim	Shouri Saha
20101586	20101349

Approval

The thesis titled "Connected Hidden Neurons (CHNNet): An Artificial Neural Network for Rapid Convergence" submitted by

1. Rafiad Sadat Shahir (20101580)
2. Zayed Humayun (20141030)
3. Mashrufa Akter Tamim (20101586)
4. Shouri Saha (20101349)

Of Summer, 2023 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on September 21, 2023.

Examining Committee:

Supervisor:
(Member)

Md. Golam Rabiul Alam, PhD

Professor
Department of Computer Science and Engineering
BRAC University

Program Coordinator:
(Member)

Md. Golam Rabiul Alam, PhD

Professor
Department of Computer Science and Engineering
BRAC University

Head of Department:
(Chair)

Sadia Hamid Kazi

Associate Professor
Department of Computer Science and Engineering
Brac University

Abstract

Despite artificial neural networks being inspired by the functionalities of biological neural networks, unlike biological neural networks, conventional artificial neural networks are often structured hierarchically, which can impede the flow of information between neurons as the neurons in the same layer have no connections between them. Hence, we propose a more robust model of artificial neural networks where the hidden neurons, residing in the same hidden layer, are interconnected that leads to rapid convergence. With the experimental study of our proposed model as fully connected layers in deep networks, we demonstrate that the model results in a noticeable increase in convergence rate compared to the conventional feed-forward neural network.

Keywords: Artificial Neural Network, Connected Hidden Neurons, Rapid Convergence.

Table of Contents

List of Figures

List of Tables

Nomenclature

List of several symbols & abbreviation used within the document:

ANN	Artificial Neural Network
CHN	Connected Hidden Neurons
CNN	Convolutional Neural Network
ESN	Echo State Network
FNN	Feed-forward Neural Network
LSM	Liquid State Machine
MLP	Multilayer Perceptron
MNIST	Modified National Institute of Standards and Technology
RNN	Recurrent Neural Network
SNN	Spiking Neural Network
SOTA	State of the Art

Chapter 1

Introduction

1.1 Neural Networks

The biological neural networks process large amounts of data passed by senses from different parts of the body [5]. A brain can have approximately 100 billion neurons and 100 trillion neural connections, which implies that each neuron can have connections with 1000 other neurons [17]. Moreover, the neurons in the brain form complex and dense connections among themselves, which is important for efficient and flexible information processing [14]. Although the operation of biological neurons served as inspiration for neural networks as they are used in computers, many of the designs have since gotten very disconnected from biological reality [13]. Artificial neural networks (ANNs) often follow hierarchical structures with simple neural connections that can impede the flow of information between neurons, as the neurons in the same layer have no connections between them. In some scenarios, to improve the generalization power of new and unseen data, it is important to have more connections among the neurons, as a network with more connections can learn more robust and meaningful features [20]. Moreover, having more connections among the neurons can potentially speed up the convergence rate, as it helps to learn complex patterns and relations in the data [18].

1.2 Problem Statement

We hypothesize that designing a neural network model with an increased number of neural connections will result in a performance gain in terms of learning. In conventional ANNs, specifically in feed-forward neural networks (FNNs), to increase the number of connections while keeping the number of layers fixed, the number of neurons per hidden layer has to be increased [18]. However, increasing the number of neurons can lead to a slow convergence problem in the model [22]. To achieve rapid learning, extensive research has been conducted on various aspects of neural network design, e.g. adaptive gradient methods such as the Adam optimizer [16], and activation functions such as the rectified linear unit (ReLU) [11]. With a particular focus on the architectural elements that can be adjusted to achieve rapid learning, we propose to connect the hidden neurons of the networks in order to increase the number of neural connections in the network. We propose that the model has the potential to achieve rapid convergence compared to the conventional FNNs while applying the same training strategies. However, connecting all the hidden neurons in a network is compute-intensive, and thus we design an ANN model where the hidden neurons, residing in the same hidden layer, are interconnected, which preserves the parallel computability property of the model as well.

1.3 Research Contribution

The primary contributions of the paper are summarized as follows:

- We introduced a neural network model, namely CHNNet (Connected Hidden Neurons), in which we created connections among the hidden neurons residing in the same hidden layer, enabling robust information sharing among the neurons.
- We formulated mathematical equations to calculate the activations of the hidden layers in forward propagation and revised the backpropagation algorithm to calculate the gradients based on the formulated forward propagation equations. Moreover, We provided proof of our claim of rapid convergence.
- The proposed model is different from conventional RNNs in calculating the input from the hidden neurons and is not architecturally equivalent to two conventional FNN layers connected to the previous layer through skip connections.
- We tested the proposed model on benchmark datasets and demonstrated that the model depicted a noticeable increase in convergence rate compared to the conventional FNN model.
- As our model generates a larger number of parameters compared to the conventional FNN model, we tested the proposed model against the FNN model with an increased number of parameters and showed that the model outperformed the FNN model in the mentioned configuration as well.

The paper is organized as follows: In Chapter 2, we discussed the state of knowledge in our subject area as of yet. Subsequently, we described the forward propagation, backpropagation mechanisms of the proposed model, and proof of rapid convergence in Chapter 3 and evaluated the performance of the model against the conventional FNN model in Chapter 4. Finally, we stated our concluding remarks in Chapter 5, along with a brief discussion and our future research directions.

Chapter 2

Literature Review

In the infancy of neural networks, [1] specified significant drawbacks of perceptrons and suggested the raw idea of Multilayer Perceptron (MLP). The architecture they proposed is hierarchical in structure and has no mention of connections among hidden neurons, residing in the same layer. Further, a few FNN architectures were analyzed in the literature by [6], none of which featured connections among the hidden neurons of the same layer.

Thus far, a number of ANNs have been introduced using different approaches to establish connections among neurons. A Recurrent Neural Network (RNN) has self-connections among hidden neurons through time; that is, the self-connections work as information carriers from one time step to another [6]. The Hopfield Neural Network, a single-layered neural network introduced by [3], has neurons symmetrically connected to all other neurons through bidirectional connections. While Hopfield use positive feedback to stabilize the network output, [15] proposed using negative feedback which regulate the inputs during recognition phase. Similar to the Hopfield Network, the Boltzmann Machine has its neurons connected symmetrically with all other neurons, with the exception that the neurons are divided into visible units and hidden units [4]. Neural networks like the Echo State Network (ESN) [7] and Liquid State Machine (LSM) [8] have featured a pool of neurons, namely a reservoir, which consists of numerous randomly connected neurons, providing them with non-linear modeling ability. However, as the reservoir is randomized, it requires numerous trials and sometimes even luck [9]. Additionally, in Spiking Neural Networks (SNNs), recurrent connections [26] and self-connections [28] in the hidden layer have been proposed, which require a spiking version of the actual input data to be implemented. The referred ANNs have recurrent connections among the neurons that are different from the proposed connections among the hidden neurons of our model.

In the contemporary period, designing new paths for information flow in neural networks has attained noticeable success. Convolutional Neural Network (CNN) architectures like DenseNet [23], ResNet [19], and UNet++[25], which use skip connections to directly pass information from a layer to a deeper layer, have reached state-of-the-art (SOTA) performance. Moreover, [27] have introduced the Group Neural Network, which, to overcome the blockade at information passing, features a group of neurons that can connect freely with each other. However, due to its irregular architecture, the training of the network cannot be accelerated through parallel computing. The mentioned ANNs use different approaches to enable information flow among the hidden neurons than ours.

Chapter 3

Methodology

The proposed architecture features additional self-connections and interconnections among the hidden neurons, as shown in figure 3.1.

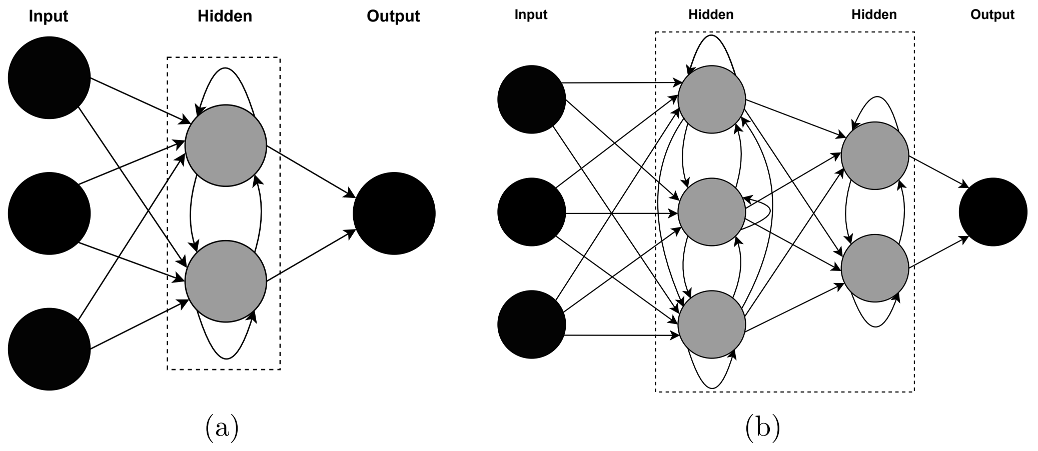


Figure 3.1: Proposed architecture of CHNNet with (a) one hidden layer and (b) two hidden layers.

We have formulated mathematical equations for forward propagation and revised the backpropagation algorithm for hidden layers only, as no new connections have been introduced in the input and output layers.

3.1 Forward propagation

Rumelhart, Hinton, and Williams [6] note that the process of calculating the activations of each layer in the forward direction is straightforward and can be done quickly using matrix operations. Mathematically, forward propagation can be expressed as follows:
Let f be the activation function. Then, for the l^{th} hidden layer, the input is $\boldsymbol{A}^{[l-1]}$ and the output $\boldsymbol{A}^{[l]}$ is computed as:

$$\boldsymbol{Z}^{[l]} = \boldsymbol{W}^{[l]}\boldsymbol{A}^{[l-1]} + \boldsymbol{B}^{[l]}$$

$$\boldsymbol{A}^{[l]} = f(\boldsymbol{Z}^{[l]})$$

where $\boldsymbol{W}^{[l]}$ is the weight matrix connecting $(l-1)^{th}$ layer to l^{th} layer, $\boldsymbol{B}^{[l]}$ is the bias matrix of l^{th} layer, and $\boldsymbol{Z}^{[l]}$ and $\boldsymbol{A}^{[l]}$ are the pre-activation and post-activation of l^{th} layer respectively.
Unlike the conventional FNN architecture, in CHNNet, information from one hidden neuron is consolidated into other hidden neurons residing in the same hidden layer. Therefore, for the forward propagation, we have two sets of weight matrices, one connecting the $(l-1)^{th}$ layer to l^{th} layer and the other connecting hidden neurons of the l^{th} layer to other hidden neurons of the layer. Then for layer l, the input is $\boldsymbol{A}^{[l-1]}$ and the pre-activation $\boldsymbol{Z}^{[l]}$ is proposed to be computed as:

$$\boldsymbol{Z}^{[l]} = \boldsymbol{W}_1^{[l]}\boldsymbol{A}^{[l-1]} + \boldsymbol{W}_2^{[l]}\boldsymbol{H}^{[l]} + \boldsymbol{B}^{[l]} \tag{3.1}$$

where $\boldsymbol{W}_1^{[l]}$ is the weight matrix connecting $(l-1)^{th}$ layer to l^{th} layer, $\boldsymbol{W}_2^{[l]}$ is the weight matrix connecting hidden neurons of l^{th} layer to other hidden neurons of the layer, $\boldsymbol{B}^{[l]}$ is the bias matrix l^{th} layer, $\boldsymbol{H}^{[l]}$ is the input from the hidden neurons of l^{th} layer, and $\boldsymbol{Z}^{[l]}$ and $\boldsymbol{A}^{[l]}$ are the pre-activation and post-activation of l^{th} layer respectively.
The input from the hidden neurons, $\boldsymbol{H}^{[l]}$ in equation 3.1, is the new term introduced in the conventional forward propagation equation. As yet, there are not many mechanisms available to calculate the output of the hidden neurons given an input. In the proposed model, the pre-activation of the l^{th} hidden layer is used to calculate $\boldsymbol{H}^{[l]}$. Thereby, for l^{th} layer, the input is $\boldsymbol{A}^{[l-1]}$ and the input from the hidden neurons $\boldsymbol{H}^{[l]}$ is computed as:

$$\boldsymbol{H}^{[l]} = \boldsymbol{W}_1^{[l]}\boldsymbol{A}^{[l-1]} + \boldsymbol{B}^{[l]} \tag{3.2}$$

Finally, the post-activation $\boldsymbol{A}^{[l]}$ of l^{th} layer is computed as:

$$\boldsymbol{A}^{[l]} = f(\boldsymbol{Z}^{[l]})$$

Though the forward propagation mechanism of the proposed model echoes the forward propagation mechanism of conventional RNNs, in conventional RNNs, the activations of the hidden neurons, obtained from the prior time step, are used to calculate the output of the hidden layer, whereas in CHNNet, the current pre-activations of the hidden neurons are used to calculate the output of the hidden layer. Moreover, there is an argument to be made that the choice of mechanism to calculate the input from the hidden neurons, $\boldsymbol{H}^{[l]}$, has conceptually led to generating two conventional FNN layers connected to the previous layer through skip connections. However, no non-linear activation function has been used to calculate the value of $\boldsymbol{H}^{[l]}$. Thereby, as artificial neurons need to be associated with a non-linear activation function [1], $\boldsymbol{H}^{[l]}$ cannot be considered as the input from an individual layer of neurons.

3.2 Backpropagation

As described by Rumelhart, Hinton, and Williams [6], the backpropagation can be mathematically expressed as follows:
Let $\boldsymbol{Y}$ be the output, f be the activation function, and $\boldsymbol{E}$ be the cost. The upstream gradient for the output layer $\boldsymbol{D}_u^{(L)}$, where L is the number of layers in the network, is computed as:

$$\boldsymbol{D}_u^{[L]} = \nabla_{\boldsymbol{Y}}\boldsymbol{E}$$

Let $\nabla_{\boldsymbol{Z}^{[l]}}\boldsymbol{E} = \boldsymbol{D}^{[l]}$, where, $\boldsymbol{Z}^{[l]}$ be the pre-activation of l^{th} layer. For l^{th} layer, $\boldsymbol{D}^{[l]}$ is computed as:

$$\boldsymbol{D}^{[l]} = \boldsymbol{D}_u^{[l]} * f'(\boldsymbol{Z}^{[l]})$$

Then, the partial derivatives of the cost with respect to weight matrix $\boldsymbol{W}^{[l]}$ and bias matrix $\boldsymbol{B}^{[l]}$ are computed as:

$$\nabla_{\boldsymbol{W}^{[l]}}\boldsymbol{E} = \boldsymbol{D}^{[l]}\boldsymbol{A}^{[l-1]^{\mathsf{T}}}$$

$$\nabla_{\boldsymbol{B}^{[l]}}\boldsymbol{E} = \boldsymbol{D}^{[l]}$$

where $\boldsymbol{A}^{[l-1]}$ is the input for l^{th} layer. Finally, the weight matrix $\boldsymbol{W}^{[l]}$ and bias matrix $\boldsymbol{B}^{[l]}$ are updated using the following equations:

$$\boldsymbol{W}^{[l]} \rightarrow \boldsymbol{W}^{[l]} - \eta\nabla_{\boldsymbol{W}^{[l]}}\boldsymbol{E} \tag{3.3}$$

$$\boldsymbol{B}^{[l]} \rightarrow \boldsymbol{B}^{[l]} - \eta\nabla_{\boldsymbol{B}^{[l]}}\boldsymbol{E} \tag{3.4}$$

where η is the learning rate of the network.
In contrast to conventional FNN architectures, CHNNet has two sets of weight matrices. The weight matrix $\boldsymbol{W}_1^{[l]}$ and bias matrix $\boldsymbol{B}^{[l]}$ are updated using equation 3.3 and equation 3.4 respectively. The partial derivative of the cost with respect to weight matrix $\boldsymbol{W}_2^{[l]}$ is computed as:

$$\nabla_{\boldsymbol{W}_2^{[l]}}\boldsymbol{E} = \boldsymbol{D}^{[l]}\boldsymbol{H}^{[l]^{\mathsf{T}}} \tag{3.5}$$

Then, the weight matrix $\boldsymbol{W}_2^{[l]}$ is updated using the following equation:

$$\boldsymbol{W}_2^{[l]} \rightarrow \boldsymbol{W}_2^{[l]} - \eta\nabla_{\boldsymbol{W}_2^{[l]}}\boldsymbol{E} \tag{3.6}$$

In the end, for $(l-1)^{th}$ layer, the upstream gradient $\boldsymbol{D}_u^{[l-1]}$ is computed as:

$$\boldsymbol{D}_u^{[l-1]} = \boldsymbol{D}^{[l]}\boldsymbol{W}_1^{[l]^{\mathsf{T}}}$$

The process of backpropagation for a hidden layer is summarized in Algorithm 1.

3.3 Proof of rapid convergence

For the purpose of proving the claim of rapid convergence, let, at time step t, $C_C(w_1^t, w_2^t)$ be the cost of a hidden layer of CHNNet with weights w_1^t and w_2^t and $F_C(w_1^t)$ be the cost of a hidden layer of the conventional FNN with weight w_1^t. Mathematically, the cost function $C_C(w_1^t, w_2^t)$ and $F_C(w_1^t)$ can be written as:

$$C_C(w_1^t, w_2^t) = ||O^* - f(C_F(w_1^t, w_2^t))||$$

$$F_C(w_1^t) = ||O^* - f(F_F(w_1^t))||$$

where, $C_F(w_1^t, w_2^t)$ is the pre-activation of the hidden layer of CHNNet with weights w_1^t and w_2^t at time step t, $F_F(w_1^t)$ is the pre-activation of the hidden layer of the conventional FNN with weight w_1^t at time step t, f is the activation function and O^* is the optimal output of the hidden layer.

```
Algorithm 1 Backpropagation algorithm for a single hidden layer in CHNNet
  Computing Delta:
      $\boldsymbol{D}^{[l]} = \boldsymbol{D}_u^{[l]} * f'(\boldsymbol{Z}^{[l]})$
  Computing Gradients:
      $\nabla_{\boldsymbol{W}_1^{[l]}} \boldsymbol{E} = \boldsymbol{D}^{[l]} {\boldsymbol{A}^{[l-1]}}^{\mathsf{T}}$
      $\nabla_{\boldsymbol{W}_2^{[l]}} \boldsymbol{E} = \boldsymbol{D}^{[l]} {\boldsymbol{H}^{[l]}}^{\mathsf{T}}$
      $\nabla_{\boldsymbol{B}^{[l]}} \boldsymbol{E} = \boldsymbol{D}^{[l]}$
  Updating Weights:
      $\boldsymbol{W}_1^{[l]} \to \boldsymbol{W}_1^{[l]} - \eta \nabla_{\boldsymbol{W}_1^{[l]}} \boldsymbol{E}$
      $\boldsymbol{W}_2^{[l]} \to \boldsymbol{W}_2^{[l]} - \eta \nabla_{\boldsymbol{W}_2^{[l]}} \boldsymbol{E}$
      $\boldsymbol{B}^{[l]} \to \boldsymbol{B}^{[l]} - \eta \nabla_{\boldsymbol{B}^{[l]}} \boldsymbol{E}$
  Computing Downstream Gradient:
      $\boldsymbol{D}_u^{[l-1]} = \boldsymbol{D}^{[l]} {\boldsymbol{W}_1^{[l]}}^{\mathsf{T}}$
```

Let at time step t, $m^t = F_F(w_1^t) = w_1^t a_{l-1}^t + b^t$ and $c^t = w_2^t h_l^t$ where, a_{l-1}^t is the activation of the $(l-1)^{th}$ layer and h_l^t is the input from the hidden neurons of the l^{th} hidden layer. Thus, $C_F(w_1^t, w_2^t)$ can be written as:

$$C_F(w_1^t, w_2^t) = m^t + c^t \tag{3.7}$$

Here, $h_l^t = m^t$, and thus we get from equation 3.7,

$$C_F(w_1^t, w_2^t) = m^t + w_2^t m^t$$
$$\Rightarrow C_F(w_1^t, w_2^t) = (1 + w_2^t) F_F(w_1^t) \tag{3.8}$$

When $w_2^{t-1} = 0$ at time step $t-1$, we get using equation 3.8,

$$||O^* - f(C_F(w_1^{t-1}, w_2^{t-1}))|| = ||O^* - f(F_F(w_1^{t-1}))|| \Rightarrow C_C(w_1^{t-1}, w_2^{t-1}) = F_C(w_1^{t-1}) \tag{3.9}$$

Then, as gradient descend is guaranteed to converge according to the convergence theorem of gradient descend, w_2 is updated such that $f(C_F) \to O^*$. Therefore, at time step t we get,

$$||O^* - f(C_F(w_1^t, w_2^t))|| \leq ||O^* - f(F_F(w_1^t))|| \Rightarrow C_C(w_1^t, w_2^t) \leq F_C(w_1^t) \tag{3.10}$$

Using equation 3.9 and inequality 3.10, we get,

$$C_C(w_1^{t-1}, w_2^{t-1}) - C_C(w_1^t, w_2^t) \geq F_C(w_1^{t-1}) - F_C(w_1^t) \tag{3.11}$$

Equation 3.11 implies that the difference between the cost of CHNNet, generated at two sequential time steps, is greater than that of the conventional FNN; that is, CHNNet converges faster than the conventional FNN.

Chapter 4

Performance Evaluation

To evaluate the performance of the proposed model, we used the software library TensorFlow. Using the library, we constructed a layer, namely the CHN Layer, implementing the forward propagation and backpropagation mechanisms described in Chapter 3. Using the layer, along with other layers provided by the TensorFlow library, we performed all our experiments. Our goal was not to get SOTA performance on the benchmark datasets. Rather, it was to achieve a better convergence rate than the conventional FNN. We compared the performance of the CHN layers with that of the Dense layers provided by the TensorFlow library, which implement conventional forward propagation and backpropagation mechanisms of FNN. In our initial experiments, CHNNet generated a larger number of trainable parameters compared to the conventional FNN, and thus we conducted some additional experiments with CHNNet and FNN having nearly equal numbers of trainable parameters.

4.1 Datasets

We evaluated the performance of the CHNNet on four benchmark datasets of different sizes and diverse features, namely the Boston Housing [2], MNIST [12], Fashion MNIST [24], and Extended MNIST [21] datasets. The Boston Housing dataset, used for linear regression experiments, contains information concerning housing in the Boston, MA area, with 506 instances and 14 attributes in total. The MNIST dataset, consisting of 60,000 training samples and 10,000 testing samples, holds 28x28 images of handwritten digits divided into 10 classes. The Fashion MNIST (FMNIST) dataset has the same features as the MNIST dataset, with the exception that the dataset contains 10 classes of fashion accessories instead of handwritten digits. The FMNIST dataset is more complex compared to the MNIST dataset. The Extended MNIST (EMNIST) dataset, consisting of 697,932 training samples and 116,323 testing samples, contains 28x28 images of handwritten digits and letters divided into 62 classes. The EMNIST dataset is profoundly more complex than both the MNIST and FMNIST datasets.

4.2 Hyperparameters

We chose three different architectures that vary in terms of the number of hidden neurons each and the total number of layers for each of the datasets, and conducted each experiment with three seeds. The loss functions for the experiments were chosen depending on the type of dataset and the desired output format. We used mean square error in linear regression problems [10] and categorical cross entropy in multi-class classification problems [18]. The optimizers and learning rates are selected based on the improvement they could bring to the performance of the models.

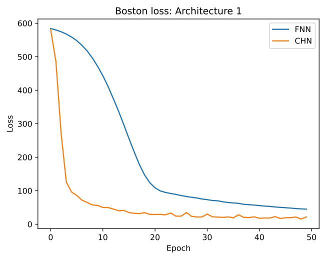
4.3 Experiments on Deep Networks

4.3.1 Training Parameters

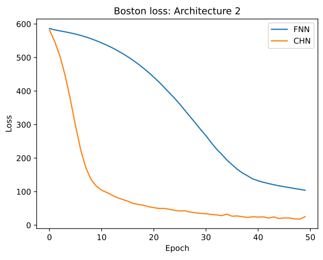
For the Boston Housing dataset, we used networks consisting of 6 hidden layers with 64 neurons each, 5 hidden layers with 128 neurons each, and a network as ”160-128-96-64-1”. We used the RMSprop optimizer with a learning rate of 0.0003, 0.0001, 0.0002 respectively, mean square error as the loss function, and batches of size 128 for training the models. While training on the MNIST dataset, we used networks consisting of 4 hidden layers with 96 neurons each, 6 hidden layers with 256 neurons each, and a network as ”288-256-224-192-160-128-96-64-10”. Additionally, We used the RMSprop optimizer with a learning rate of 0.0001, sparse categorical cross entropy as the loss function and batches of size 512. We used networks consisting of 3 hidden layers with 512 neurons each, 6 hidden layers with 256 neurons each, and a network as ”928-800-672-544-416-288-160-32-10” for training on the FMNIST dataset. Moreover, we used the SGD optimizer with a learning rate of 0.001, sparse categorical cross entropy as the loss function and batches of size 32 for the training. For the EMNIST dataset, we used networks consisting of 3 hidden layers with 768 neurons each, 6 hidden layers with 320 neurons each, and a network as ”1024-896-768-640-512-348-256-128-62”. We used the SGD optimizer with a learning rate of 0.001, sparse categorical cross entropy as the loss function and batches of size 32 for training the models. The networks had ReLU activation in the hidden layers, linear activation in the output layer of the Boston dataset, and softmax activation in the output layer of the MNIST, FMNIST, and EMNIST datasets. Further, we performed t-tests on the sets of accuracies achieved by the conventional FNN and CHNNet through the networks and obtained the p-values and t-statistics. A small p-value indicates that the mean accuracies of FNN and CHNNet are not identical. Furthermore, smaller p-values are associated with larger t-statistics.

4.3.2 Test Results

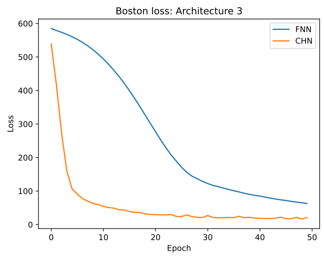
The CHNNet showed a considerable performance gain in terms of convergence compared to the conventional FNN with all the architectures, as portrayed in figure 4.1. In addition, CHNNet showed a better performance, on average, than the conventional FNN in terms of mean loss and mean accuracy, as shown in table 4.1 and table 4.2. Especially with all the architectures on the Boston dataset, CHNNet depicted a noteworthy decrease in mean loss. Moreover, in terms of accuracy, the experiments depicted negative t-statistics with all the architectures, which suggests that CHNNet had a higher mean accuracy than the conventional FNN in the experiments.



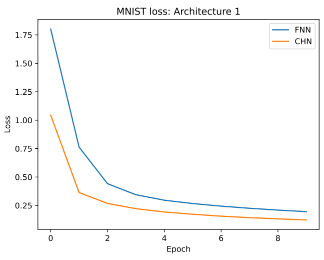
(a)



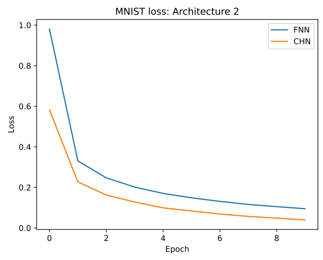
(b)



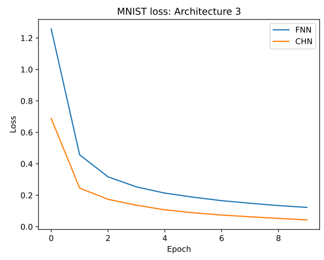
(c)



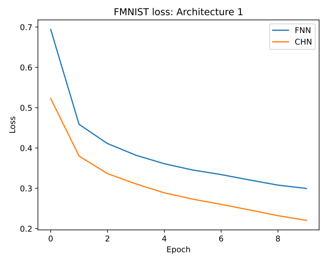
(d)



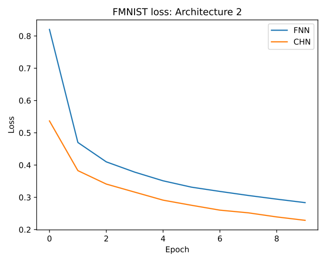
(e)



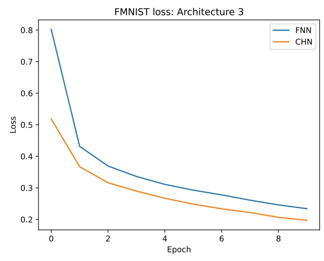
(f)



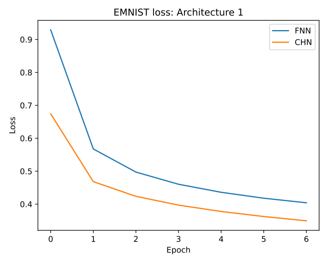
(g)



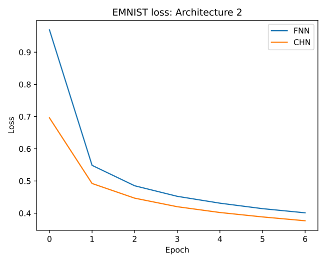
(h)



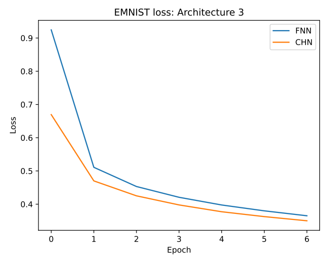
(i)



(j)



(k)



(l)

Figure 4.1: Loss curves of three different architectures of CHNNet and FNN on the (a)-(c) Boston Housing, (d)-(f) MNIST, (g)-(i) FMNIST, and (j)-(l) EMNIST datasets.

4.4 Experiments with Equal Parameters

In the experiments conducted previously, the CHNNet generated more trainable parameters compared to the conventional FNN. Hence, we conducted additional experiments with an increased number of hidden neurons in the Dense layers to

Table 4.1: Loss measurement of CHNNet and FNN in deep networks.

		FNN		CHNNet	
Datasets	Model	Trainable Params	Mean Loss (±std)	Trainable Params	Mean Loss (±std)
Boston	Arch-1	21,761	47.65(±3.49)	46,337	27.13(±1.72)
	Arch-2	67,969	102.57(±5.25)	149,889	26.71(±0.12)
	Arch-3	41,505	69.10(±2.19)	96,801	24.97(±1.54)
MNIST	Arch-1	104,266	0.187(±0.004)	141,130	0.142 (±0.003)
	Arch-2	532,490	0.125(±0.013)	925,706	0.110(±0.010)
	Arch-3	471,562	0.153(±0.008)	762,378	0.138(±0.027)
FMNIST	Arch-1	932,362	0.351(±0.004)	1,718,794	0.319(±0.003)
	Arch-2	532,490	0.339(±0.004)	925,706	0.336(±0.014)
	Arch-3	2,774,602	0.334(±0.004)	5,305,930	0.342(±0.013)
EMNIST	Arch-1	1,831,742	0.429(±0.001)	3,601,214	0.411(±0.004)
	Arch-2	784,702	0.429(±0.001)	1,193,982	0.426(±0.001)
	Arch-3	3,567,934	0.422(±0.005)	6,910,270	0.419(±0.004)

Table 4.2: Accuracy measurement of CHNNet and FNN in deep networks.

Datasets	Model	FNN Mean Accuracy (±std)	CHNNet Mean Accuracy (±std)	p-value	t-statistics
MNIST	Arch-1	94.47(±0.17)	95.62(±0.13)	0.03	-6.20
	Arch-2	96.13(±0.46)	96.71 (±0.30)	0.13	-2.47
	Arch-3	95.41(±0.28)	96.16(±0.75)	0.41	-1.02
FMNIST	Arch-1	87.35(±0.24)	88.68 (±0.11)	0.03	-5.51
	Arch-2	87.71(±0.10)	88.11(±0.20)	0.10	-2.93
	Arch-3	88.13(±0.32)	88.56(±0.30)	0.29	-1.42
EMNIST	Arch-1	84.76(±0.02)	85.25(±0.05)	0.004	-14.97
	Arch-2	84.67(±0.07)	84.73(±0.02)	0.46	-0.91
	Arch-3	84.89 (±0.21)	85.16(±0.09)	0.15	-2.29

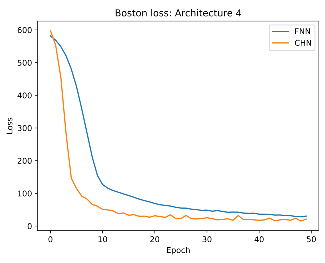
evaluate the performance of CHNNet compared to FNN with a nearly equal number of trainable parameters.

4.4.1 Training Parameters

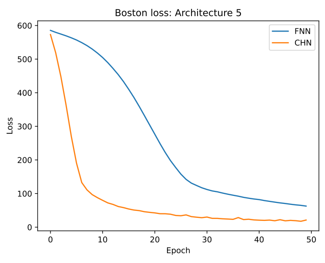
The training parameters were the same as in previous experiments, except that we increased the number of hidden neurons in the Dense layers. Hence, for the conventional FNN, we used an architecture featuring 6 hidden layers, each containing 95 neurons, 5 hidden layers with 190 neurons each, and 4 hidden layers with "220-198-198-124" neurons in the respective hidden layers for the Boston Housing Dataset. For the MNIST dataset, architectures featured 4 hidden layers, each containing 126 neurons, 6 hidden layers with 360 neurons each, and 8 hidden layers with "360-334-304-268-238-208-176-142" neurons in the respective hidden layers. For the FMNIST dataset, we used architectures with 3 hidden layers with 749 neurons each, 6 hidden layers with 358 neurons each, 8 hidden layers with "1184-1056-928-800-704-604-448-352" neurons in the respective hidden layers. For the EMNIST dataset, architectures featured 3 hidden layers with 1152 neurons each, 6 hidden layers with 412 neurons each, and 8 hidden layers with "1272-1144-1016-978-760-632-504-376" neurons in the respective hidden layers.

4.4.2 Test Results

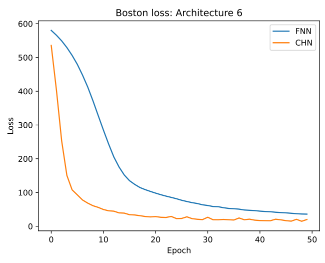
Despite increasing the number of neurons in the hidden layers of the conventional FNN, CHNNet showed a considerably faster convergence rate with all the architecture, as depicted in figure 4.2. Moreover, the CHNNet commonly performed better in terms of mean loss and mean accuracy compared to the conventional FNN, as illustrated in tables 4.3 and 4.4. The experiments depicted negative t-statistics with the architectures except for one architecture on both the MNIST and FMNIST datasets and two architectures on the EMNIST dataset, which is interpreted as CHNNet commonly outperforming FNN in terms of mean loss and mean accuracy. Further, it can be concluded that the larger number of parameters generated by CHNNet is not a matter of concern, as even with a nearly equal number of parameters in both CHNNet and the conventional FNN model, CHNNet outperformed the FNN model in terms of convergence.



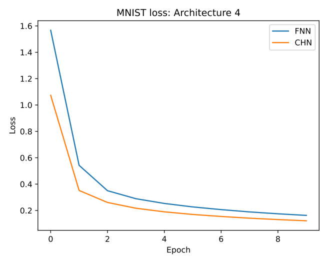
(a)



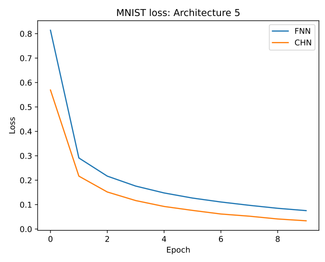
(b)



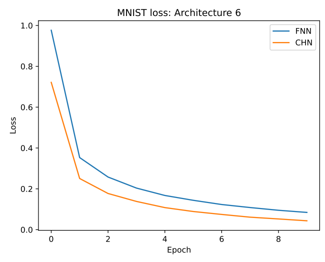
(c)



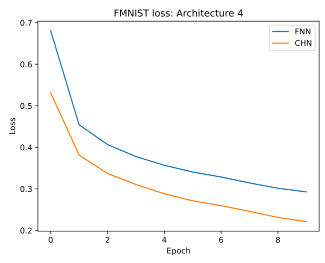
(d)



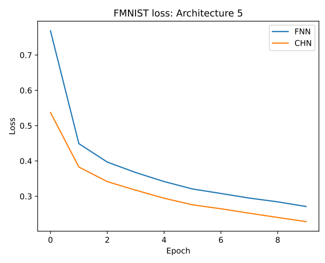
(e)



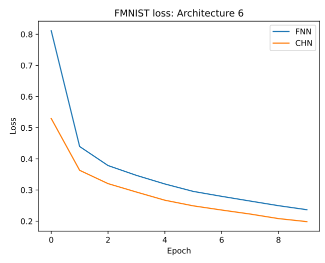
(f)



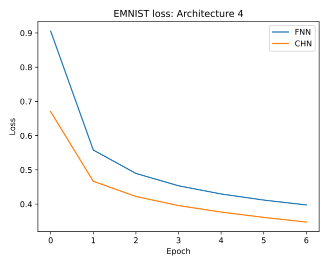
(g)



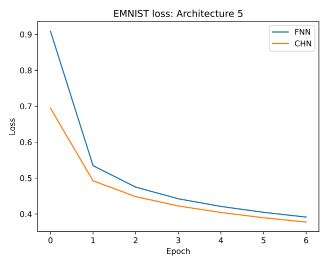
(h)



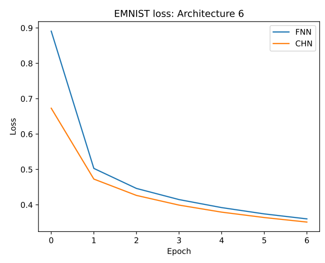
(i)



(j)



(k)



(l)

Figure 4.2: Loss curves of three different architectures of CHNNet and FNN on the (a)-(c) Boston Housing, (d)-(f) MNIST, (g)-(i) FMNIST, and (j)-(l) EMNIST datasets with nearly equal numbers of parameters.

Table 4.3: Loss measurement of CHNNet and FNN with nearly equal number of parameters.

		FNN		CHNNet	
Datasets	Model	Trainable Params	Mean Loss (±std)	Trainable Params	Mean Loss (±std)
Boston	Arch-4	46,337	30.13(±1.01)	47,026	28.02(±1.21)
	Arch-5	148,001	60.63(±5.80)	149,889	24.23(±0.23)
	Arch-6	97,475	39.54(±3.57)	96,801	23.73(±1.19)
MNIST	Arch-4	148,186	0.164(±0.002)	141,130	0.151(±0.004)
	Arch-5	936,010	0.115(±0.015)	925,706	0.111(±0.006)
	Arch-6	763,836	0.146(±0.035)	762,378	0.228(±0.085)
FMNIST	Arch-4	1,718,965	0.345(±0.004)	1,718,794	0.320(±0.003)
	Arch-5	927,230	0.339(±0.006)	925,706	0.336(±0.008)
	Arch-6	5,327,238	0.327(±0.006)	5,305,930	0.341(±0.015)
EMNIST	Arch-4	3,632,318	0.426(±0.004)	3,601,214	0.411(±0.001)
	Arch-5	1,199,806	0.425(±0.004)	1,193,982	0.430(±0.002)
	Arch-6	6,370,056	0.411(±0.003)	6,369,086	0.418(±0.001)

Table 4.4: Accuracy measurement of CHNNet and FNN with nearly equal numbers of parameters.

Datasets	Model	FNN Mean Accuracy (±std)	CHNNet Mean Accuracy (±std)	p-value	t-statistics
MNIST	Arch-4	95.12(±0.08)	95.39(±0.18)	0.10	-2.92
	Arch-5	96.59(±0.35)	96.80(±0.13)	0.46	–0.92
	Arch-6	95.57(±1.04)	94.02(±2.16)	0.21	1.83
FMNIST	Arch-4	87.52(±0.19)	88.64(±0.23)	0.004	-15.83
	Arch-5	87.78(±0.02)	88.37(±0.10)	0.02	-7.37
	Arch-6	88.45(±0.27)	88.43(±0.31)	0.96	0.05
EMNIST	Arch-4	84.84(±0.14)	85.20(±0.06)	0.03	-5.83
	Arch-5	84.71(±0.11)	84.66(±0.09)	0.10	2.90
	Arch-6	85.18(±0.10)	85.07(±0.07)	0.15	2.33

Chapter 5

Conclusion

5.1 Conclusion

We designed an ANN, namely CHNNet, that is different from the existing feed-forward networks in connecting the hidden neurons of the same layer. In addition, we described the forward propagation and backpropagation mechanisms of the proposed model and provided proof of the claim of rapid convergence. In the experiments we conducted, CHNNet showed a noticeable increase in convergence rate compared to the conventional FNN model without compromising loss or accuracy. However, the proposed model generated a larger number of parameters compared to the conventional FNN model. Thus, we conducted additional experiments and concluded that the larger number of trainable parameters generated by CHNNet is not a matter of concern, as even with a nearly equal number of trainable parameters in both CHNNet and the conventional FNN model, CHNNet outperformed the FNN model in terms of convergence with no compromises with loss and accuracy.

5.2 Future Work

As CHNNet generates a large number of parameters, some algorithms can be proposed to reduce the number of connections among the neuron systematically. Moreover, the model has not been tested as a fully connected layer in CNN architectures, which also has the potential to generate compelling outcomes. It would be interesting to see if more efficient CNN architectures can be developed utilizing the features of CHNNet. Further, there is the opportunity for research on implementing RNN models based on the architecture of CHNNet.

Bibliography

[1] M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry.* Cambridge, MA, USA: MIT Press, 1969.

[2] D. Harrison and D. L. Rubinfeld, “Hedonic housing prices and the demand for clean air,” *Journal of Environmental Economics and Management*, vol. 5, no. 1, pp. 81–102, 1978, ISSN: 0095-0696. DOI: https://doi.org/10.1016/0095-0696(78)90006-2. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0095069678900062.

[3] J. Hopfield, “Neural networks and physical systems with emergent collective computational abilities,” *Proceedings of the National Academy of Sciences of the United States of America*, vol. 79, pp. 2554–2558, May 1982. DOI: 10.1073/pnas.79.8.2554.

[4] G. Hinton and T. Sejnowski, “Learning and relearning in boltzmann machines,” *Parallel Distributed Processing*, vol. 1, Jan. 1986.

[5] G. Palm, “Warren mcculloch and walter pitts: A logical calculus of the ideas immanent in nervous activity,” pp. 229–230, Jan. 1986. DOI: 10.1007/978-3-642-70911-1_14.

[6] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning internal representations by error propagation,” in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations.* Cambridge, MA, USA: MIT Press, 1986, pp. 318–362, ISBN: 026268053X.

[7] H. Jaeger, “The” echo state” approach to analysing and training recurrent neural networks-with an erratum note’,” *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, vol. 148, Jan. 2001.

[8] W. Maass, T. Natschläger, and H. Markram, “Real-time computing without stable states: A new framework for neural computation based on perturbations,” *Neural computation*, vol. 14, pp. 2531–2560, Dec. 2002. DOI: 10.1162/089976602760407955.

[9] M. Ozturk, D. Xu, and J. Principe, “Analysis and design of echo state networks,” *Neural computation*, vol. 19, pp. 111–138, Feb. 2007. DOI: 10.1162/neco.2007.19.1.111.

[10] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference and prediction*, 2nd ed. Springer, 2009. [Online]. Available: http://www-stat.stanford.edu/~tibs/ElemStatLearn/.

[11] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” vol. 15, Jan. 2010.

[12] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, vol. 2, 2010.

[13] D. Akomolafe, "Scholars research library comparative study of biological and artificial neural networks," *European Journal of Applied Engineering and Scientific Research*, vol. 2, pp. 36–46, Jan. 2013.

[14] O. Sporns, "Structure and function of complex brain networks," *Dialogues in clinical neuroscience*, vol. 15, pp. 247–62, Sep. 2013. DOI: 10.31887/DCNS.2013.15.3/osporns.

[15] T. Achler, "Symbolic neural networks for cognitive capacities," *Biologically Inspired Cognitive Architectures*, vol. 9, pp. 71–81, 2014.

[16] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, Dec. 2014.

[17] M. Glasser, T. Coalson, E. Robinson, *et al.*, "A multi-modal parcellation of human cerebral cortex," *Nature*, vol. 536, Jul. 2016. DOI: 10.1038/nature18933.

[18] I. J. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016, http://www.deeplearningbook.org.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," Jun. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[20] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires rethinking generalization," *Communications of the ACM*, vol. 64, Nov. 2016. DOI: 10.1145/3446776.

[21] G. Cohen, S. Afshar, J. Tapson, and A. V. Schaik, "Emnist: Extending mnist to handwritten letters," *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017. DOI: 10.1109/ijcnn.2017.7966217.

[22] A. Gron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 1st. O'Reilly Media, Inc., 2017, ISBN: 1491962291.

[23] G. Huang, Z. Liu, L. van der Maaten, and K. Weinberger, "Densely connected convolutional networks," Jul. 2017. DOI: 10.1109/CVPR.2017.243.

[24] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms," Aug. 2017.

[25] Z. Zhou, M. M. Rahman Siddiquee, N. Tajbakhsh, and J. Liang, "Unet++: A nested u-net architecture for medical image segmentation," in Sep. 2018, vol. 11045, pp. 3–11, ISBN: 978-3-030-00888-8. DOI: 10.1007/978-3-030-00889-5_1.

[26] S.-Q. Zhang, Z.-Y. Zhang, and Z.-H. Zhou, "Bifurcation spiking neural network," *Journal of Machine Learning Research*, vol. 22, no. 253, pp. 1–21, 2021.

[27] J. Liu, M. Gong, L. Xiao, W. Zhang, and F. Liu, "Evolving connections in group of neurons for robust learning," *IEEE Transactions on Cybernetics*, vol. 52, no. 5, pp. 3069–3082, 2022. DOI: 10.1109/TCYB.2020.3022673.

[28] S.-Q. Zhang and Z.-H. Zhou, "Theoretically provable spiking neural networks," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35, Curran Associates, Inc., 2022, pp. 19 345–19 356.

Appendix A

CHNNet Supplements

Dataset

All the datasets is automatically imported from TensorFlow library while executing the code.

Environment Setup

All the experiments accompanying the paper have been conducted using TensorFlow 2.10. A detailed guideline for installing TensorFlow with pip can be found at tensorflow.org.

Requirements

The experiments are carried out in a Python 3.8 environment. The following additional packages are required to run the tests:

- tensorflow-datasets (version 1.2.0)
- matplotlib (version 3.7.1)
- pandas (version 1.5.3)
- scikit-learn (version 1.2.1)
- scipy (version 1.6.2)

The dependencies can be installed manually or by using the following command:

```
pip install -r ./requirements.txt
```

It is recommended to use a virtual environment for installing all the modules.

Potential Errors

While using tensorflow-dataset, you can encounter the following error:

```
importError: cannot import 'builder' from google.protobuf.internal
```

To fix this error, you can install protobuf version 3.20 using the following command:

```
pip install protobuf==3.20
```

Code Explanation

All the required files for executing the tests can be found at
github.com/ThesisG/CHNNet

CHN Layer

The CHN layer is coded in the CHNLayer.py file using Layer superclass of Keras. The variables named kernel_Input_Units and kernel_Hidden_Units inside the build function represent the two sets of weights mentioned in the paper.

The call method defines the forward pass of the layer and can handle different types of inputs, while the backpropagation of the layer is handled by tensorflow itself. TensorFlow's automatic differentiation mechanism has been used to calculate the gradients during backpropagation.

Test Files

The py files named {dataset_name}Test.py holds the codes for the tests on that respective dataset. By executing the codes in these files, the test results for the respective dataset can be generated.

Parameters

The following adjustable parameters allow for customization of the model's architecture, training duration, and optimization strategy based on the specific task and dataset:

- epochs: represents the number of epochs for training.
- batchSize: represents the size of each batch for training.
- CHN_hn: represents the number of hidden neurons in the n^{th} hidden layer of the CHNLayer.
- MLP_hn: represents the number of hidden neurons in the n^{th} hidden layer of the Dense layer.
- loss: determines the objective function used to measure the model's performance and guide its learning during training.
- optimizer: determines the algorithm used to optimize the neural network model during training.

Results

- When the training is complete, the model summary and statistical test results are displayed on the terminal.
- The loss graphs for each seed are displayed in separate windows afterward.