Optimizing Performance of Feedforward and Convolutional Neural Networks through Dynamic Activation Functions

Anonymous authors
Paper under double-blind review

Abstract

Deep learning training algorithms have been an enormous success in recent years in many fields, including speech, text, image, and video. Deeper and deeper layers are proposed with huge success, with ResNet structures having around 152 layers. Shallow convolution neural networks(CNNs) are still active research, where some phenomena are still unexplained. Activation functions used in the network are of utmost importance, as they provide non-linearity to the networks. ReLU's are the most commonly used activation function. We show the hidden layer's complex piece-wise linear(PWL) activation. We show that these PWL activations work much better than ReLU activations in our networks for convolution neural networks and multilayer perceptrons. Result comparisons in PyTorch for shallow and deep CNNs are given to strengthen our case further.

1 Introduction

Deep learning architectures have found tremendous success in speech recognition, image recognition, language recognition, and translation in recent years. For speech, language recognition, and translation, deep recurrent neural networks(RNNs)Rumelhart et al. (1986a)Jordan (1997)Hopfield (1982) have shown improvements over older technologies. These networks can process images and sequences of images such as videos, text, and speech. The RNN structure consists of cells and gates. These cells store important information over time, and the gates decide the passage of information in and out of the cells. RNNs face vanishing gradient problems and cannot process words over a long period. To address this situation, networks such as long shot term memory(LSTM)Hochreiter & Schmidhuber (1997), gated recurrent unit(GRU)Chung et al. (2014), and transformersVaswani et al. (2017) are shown. These networks are designed to handle long sequences over time and are also used for image processing. Convolutional Neural Networks(CNNs)(Lecun et al. (1998), Zhang (1988), Zhang et al. (1990)) are more widely used in image based applications. Combination networks such as CNN-LSTM, CNN-transformer are also used for visual recognitionDonahue et al. (2014)Liu et al. (2021) and time series analysisShi et al. (2015)

CNNs are used for image-based applications as a feature extractor, where one does not need to explicitly extract features for classifying images. Applications for CNNs include in diabetic retinopathy screeningGulshan et al. (2016), lesion detectionLakhani & Sundaram (2017)Kijowski et al. (1987), skin lesion classificationNahata & Singh (2020), human action recognitionIjjina & Chalavadi (2016) Parisi (2020), face recognition Kamencay et al. (2017) Wechsler et al. (1998), document analysisSimard et al. (2003) Marinai et al. (2005) and in many other applications. CNNs can be trained using gradient approaches such as backpropagation (Rumelhart et al. (1986b), Kaminski & Strumillo (1997), Lippmann (1987), Lakshmi Narasimha et al. (2008)), and conjugate gradient (Hestenes & Stiefel (1952),Shewchuk (1994),Charalambous (1992), Fletcher (1987)).

Despite their popularity, CNNs still have some limitations, such as their poorly understood shift-invariance, overfitting of the data, and the use of oversimplified nonlinear activation functions such as ReLU Nair & Hinton (2010), and leaky ReLU Maas (2013b)Nwankpa et al. (2018).

Nonlinear activation functions such as ReLUNair & Hinton (2010) and leaky ReLU Maas (2013b) have been widely used in several computer visionGlorot et al. (2010) and deep neural networkGoodfellow et al. (2016) applications. These activation functions are not as complex as sigmoids Bishop (2006b) or hyperbolic

tangent functions(Tanh)Abdelouahab et al. (2017) but are favored because they partially solve the vanishing gradient problem Hochreiter (1998). These ReLU activations do not guarantee optimal results, as different sets of activations can lead to optimal results for each filter. For example, a CNN for an image classification application with 20 filters might need 20 different activations. The number of filters required for a particular application has yet to be discovered.

Although these activations lead to universal approximationCybenko (1989) in multilayer perceptrons, many attempts have been made to create adaptive or fixed piecewise linear activation function [Nicolae (2018), Guarnieri et al. (1999), Barry & Goldman (1988),Campolucci et al. (1996),Jagtap et al. (2020)]. Adaptive activation functions for deep CNNs are introduced inAgostinelli et al. (2015), where the author trains the slope and hinges on the curve using gradient descent techniques. The author has shown promising results in terms of testing accuracies in CIFAR-10 and CIFAR-100 image recognition datasetKrizhevsky (2009) and high-energy physics involving Higgs boson decay modesBaldi et al. (2015).

This paper briefly reviews the multilayer perceptron's (MLP's) architecture, notation, training, and properties. Section 2 reviews the CNN architecture, notation, and its back propagation algorithm. In Section 3, we investigate a trainable piecewise linear (PWL) activation since they can approximate the optimal mix of activations through universal approximation. In section 4, we compare our results with ReLU activations. Finally, Section 7 discusses additional work and concludes this paper.

2 Prior Work

In this section, we review our notation for a single hidden layer cascade connected MLP, briefly summarize several commonly used feedforward classifier training methods, and describe some of the MLP's properties.

2.1 MLP Structure and notation

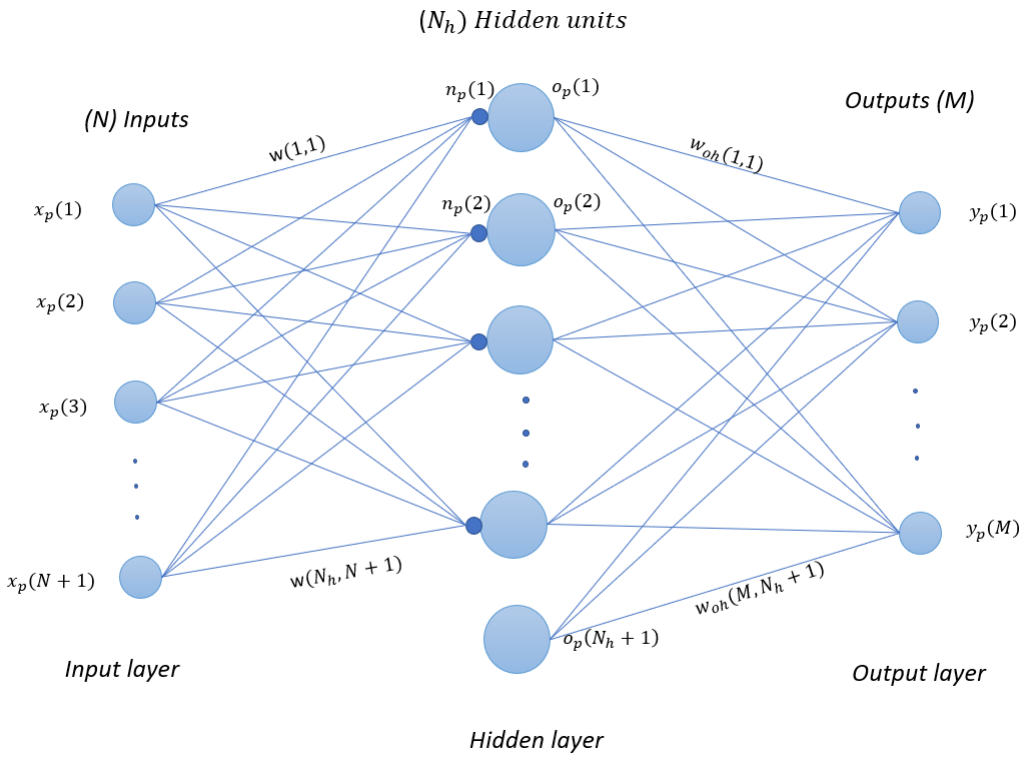


Figure 1: Single Hidden Layer MLP

A cascade-connected MLP with one hidden layer is shown in figure 1. Input weight $w(k,n)$ connects the n^{th} input to the k^{th} hidden unit. Output weight $w_{oh}(i,k)$ connects the k^{th} hidden unit's activation $o_p(k)$ to the i^{th} output. $y_p(i)$ is the p^{th} pattern, i^{th} output activation which is linear activation in figure1. In the training pattern $\{\mathbf{x}_p, \mathbf{t}_p\}$ for a MLP, the p^{th} input vector $\mathbf{x}_p$ is initially of dimension N and the p^{th} desired output (target) vector $\mathbf{t}_p$ has dimension M. The pattern number p varies from 1 to N_v. The threshold is handled by augmenting $\mathbf{x}_p$ with an extra element $X_p(n+1)$ which is equal to one where, $\mathbf{x}_p = [x_p(1), x_p(2), ..., x_p(N+1)]^T$

For the p^{th} pattern, the k^{th} hidden unit's net function $n_p(k)$ is then

$$n_p(k) = \sum_{n=1}^{N+1} w(k,n) \cdot x_p(n) \tag{1}$$

which can be summarized as

$$\mathbf{n}_p = \mathbf{W} \cdot \mathbf{x}_p \tag{2}$$

where $\mathbf{n}_p$ denotes the N_h dimensional column vector of net function values and the input weight matrix $\mathbf{W}$ is N_h by (N+1). For the p^{th} pattern, the k^{th} hidden unit's output, $o_p(k)$, is given as

$$o_p(k) = f(n_p(k)) \tag{3}$$

where $f(.)$ denotes a nonlinear hidden layer activation function, such as ReLUNair & Hinton (2010) which is represented as

$$f(n_p(k)) = \begin{cases} n_p(k), & if \quad n_p(k) \geq 0 \\ 0, & if \quad n_p(k) < 0 \end{cases} \tag{4}$$

The threshold in the hidden layer is handled by augmenting $\mathbf{o}_p$ with an extra element $o_p(N_h+1)$ which is equal to one where $\mathbf{o}_p = [o_p(1), o_p(2), ..., o_p(N_h+1)]^T$. The network's output vector for the p^{th} pattern is $\mathbf{n}_{po}$. The i^{th} element $n_{po}(i)$ of the M-dimensional output vector $\mathbf{n}_{po}$ is

$$n_{po}(i) = \sum_{k=1}^{N_h+1} w_o(i,k) \cdot o_p(k) \tag{5}$$

which can be summarized as

$$\mathbf{N}_{po} = \mathbf{W}_o \cdot \mathbf{o}_p \tag{6}$$

$\mathbf{W_o}$ is the output weight matrix with dimensions M by (N_h+1).

The output layer net vector $\mathbf{n}_{po}$ is passed through an activation function for i^{th} output which is given in terms of p^{th} pattern and i^{th} output, $y_p(i)$ as,

$$y_p(i) = f_o(\mathbf{n}_{po}(i)) \tag{7}$$

where $f_o(.)$ denotes an output hidden layer activation function. The most commonly used activation function for approximation data is the linear activation; sigmoid activation is mainly used in logistic regression, and softmax activation Zhang & Sabuncu (2018) is used for classification models, which is defined in equation (8).

The softmax output activation function is

$$y_p(i) = \frac{exp(n_{po}(i))}{\sum_{k=1}^{M} exp(n_{po}(k))} \tag{8}$$

The most commonly used objective function for a classification task is cross entropy loss functionZhang & Sabuncu (2018)

$$E_{ce} = \frac{1}{N_v} \sum_{p=1}^{N_v} [- \sum_{i=1}^{M} t_p(i) \cdot log(y_p(i))] \tag{9}$$

where $t_p(i)$ is the p^{th} pattern and i^{th} class one hot encoded output. $t_p(i)$ is found from $ic_p(i)$, where $ic_p(i)$ is the class number. For approximation or regression, mean square error(MSE)Sammut & Webb (2010) defined in equation (10) is the most widely used objective function. The MSE can also be used in the classification task with output resetGORE et al. (2005)Li et al. (2004).

$$E = \frac{1}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{M} [t_p(i) - y_p(i)]^2 \tag{10}$$

2.2 CNN structure and notation

In this section, we first review notation and training of a convolutional neural network with a single convolution layer. Then, we extend the notation to cover CNNs with multiple hidden layers.

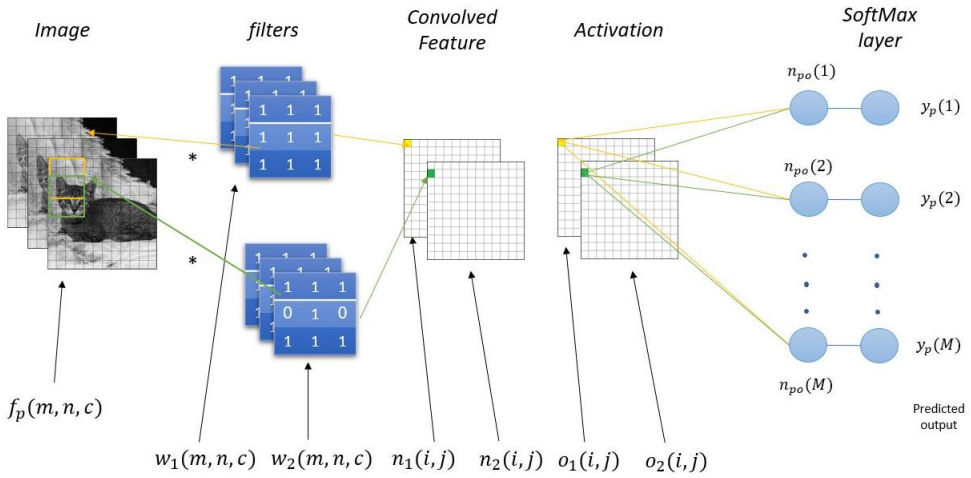


Figure 2: Shallow CNN with Linear softmax cross-entropy classifier

The CNN network structure is shown in figure 2. Let $\mathbf{f}_p$ denote the p^{th} input image and let $i_c(p)$ denote the correct class number of the p^{th} pattern, where p varies from 1 to N_v, and N_v is the total number of training images or patterns.

During forward propagation, a filter of size N_f x N_f is convolved over the image f_1 with N_r rows N_c columns.The number of channels is denoted by C, where color input images have C equal to 3 and grayscale images have C equal to 1.

For the k^{th} filter, the net function output for the i^{th} row and j^{th} column is

$$n_p(k,i,j) = t_r(k) + \sum_{m=1}^{N_f} \sum_{n=1}^{N_f} \sum_{c=1}^{C} w_f(k,m,n,c) \cdot f_p(m + (i-1)s, n + (j-1)s, c) \tag{11}$$

where, $\mathbf{n}_p$ is of size (K by M_o by N_o), where K is the number of filters, M_o is the height of the convolved image output and N_o is the width of the convolved image output. $w_f(k,m,n,c)$ is the filter of size (K by N_f by N_f by C). The threshold vector $\mathbf{t_r}$ is added to the net function output as shown in equation (11). The stride s is the number of filter shifts over input images. Note that the output $n_p(k,i,j)$ in 11 us a threshold plus a sum of C separate 2-D convolution, rather than a 3-D convolution.

To achieve non-linearity, the convolved image with element $n_p(k,i,j)$ is passed through a ReLU activation Nair & Hinton (2010) as

$$o_p(k,i,j) = f'(n_p(k,i,j)) \tag{12}$$

where, $o_p(k,i,j)$ is the k^{th} filter's hidden unit activation output for the i^{th} row, j^{th} column for the p^{th} pattern of size (K by N_{rb} by N_{cb}), where N_{rb} by N_{cb} is the row and column size of the output of the convolved image respectively.

The net function $\mathbf{n}_{po}$ for i^{th} element of the CNN's output layer for the p^{th} pattern is

$$n_{po}(i) = t_o(i) + \sum_{m=1}^{M_o} \sum_{n=1}^{N_o} \sum_{k=1}^{K} w_o(i,m,n,k) \cdot o_p(k,m,n) \tag{13}$$

where $\mathbf{W_o}$ is the 4-dimensional matrix of size (M by M_o by N_o by K), which connects hidden unit activation outputs or features to the output layer net vector $\mathbf{n}_{po}$, $\mathbf{o}_p$ is a 3-dimensional hidden unit activation output matrix of size ($M_o \cdot N_o \cdot K$) and $\mathbf{t_o}$ is the vector of biases added to net output function as in equation (13).

Before calculating the final error, the vector $\mathbf{n}_{po}$ is passed through an activation function such as softmax in 8. Finally, the cross entropy loss function is calculated using equation 9. The objective function is reduced with respect to the unknown weights. In this chapter, we discuss CNN training of classification models. We minimise the loss function E_{ce} using steepest descent Lemaréchal (2012)Lemaréchal (1944)Barton (1991), Tyagi et al. (2021)Tyagi (2018). The most common optimizer used for CNN weight training is the Adams optimizer Ada (2014). It is computationally efficient and easier to implement than optimal learning factor LeCun et al. (1998b). It uses momentum and adaptive learning rates to converge faster, which is said to be inherited from RMSPropTieleman & Hinton (2012) and AdaGradDuchi et al. (2011). The default parameters are given in Ada (2014).

3 Training algorithm

3.1 Scaled conjugate gradient algorithm

Conjugate gradient (CG) Tyagi et al. (2022) line searches in successive conjugate directions and converges faster than the steepest descent. To train an MLP using the CG algorithm (CG-MLP), we update all the network weights $\mathbf{w}$ simultaneously as follows:

$$\mathbf{w} \leftarrow \mathbf{w} + z \cdot \mathbf{p} \tag{14}$$

where z is the learning rate that can be derived as LeCun et al. (1998a),Tyagi et al. (2022).

$$z = -\frac{\frac{\partial E(\mathbf{w}+z\cdot\mathbf{p})}{\partial z}}{\frac{\partial^2 E(\mathbf{w}+z\cdot\mathbf{p})}{\partial z^2}}\Big|_{z=0} \tag{15}$$

The direction vector $\mathbf{p}$ is obtained from the gradient $\mathbf{g}$ as

$$\mathbf{p} \leftarrow -\mathbf{g} + B_1 \cdot \mathbf{p} \tag{16}$$

where $\mathbf{p} = vec\,(\mathbf{P}, \mathbf{P}_{\text{oh}}, \mathbf{P}_{\text{oi}})$ and $\mathbf{P}$, $\mathbf{P}_{\text{oh}}$ and $\mathbf{P}_{\text{oi}}$ are the direction vectors corresponding to weight arrays $(\mathbf{W}, \mathbf{W}_{\text{oh}}, \mathbf{W}_{\text{oi}})$. CG uses backpropagation to calculate $\mathbf{g}$. B_1 is the ratio of the gradient energy from two consecutive iterations. If the error function were quadratic, CG would converge in N_w iterations Boyd & Vandenberghe (2004), where the number of network weights is $N_w = dim(\mathbf{w})$. CG is scalable and widely used in training large datasets, as the network Hessian is not calculated Le et al. (2011). Therefore, in a CG, the step size is determined using a line search along the direction of the conjugate gradient.

SCG Møller (1993) scales the conjugate gradient direction by a scaling factor determined using a quasi-Newton approximation of the Hessian matrix. This scaling factor helps to accelerate the algorithm's convergence, especially for problems where the condition number of the Hessian matrix is large. SCG requires the computation of the Hessian matrix (or an approximation) and its inverse. A critical difference between CG and SCG is how the step size is determined during each iteration, with SCG using a scaling factor that helps to accelerate convergence. Other variations of CG exist Tyagi et al. (2014). However, in this study, we choose to use SCG.

3.2 Levenberg-Marquardt algorithm

The Levenberg-Marquardt (LM) algorithm Tyagi et al. (2022) is a hybrid first- and second-order training method that combines the fast convergence of the steepest descent method with the precise optimization of the Newton method Levenberg (1944). However, inverting the Hessian matrix $\mathbf{H}$ can be challenging due to its potential singularity or ill-conditioning Bishop (2006a). To address this issue, the LM algorithm introduces a damping parameter λ to the diagonal of the Hessian matrix as

$$\mathbf{H}_{LM} = \mathbf{H} + \lambda \cdot \mathbf{I} \tag{17}$$

where $\mathbf{I}$ is an identity matrix with dimensions equal to those of $\mathbf{H}$. The resulting matrix $\mathbf{H}_{LM}$ is then nonsingular, and the direction vector $\mathbf{d}_{LM}$ can be calculated by solving:

$$\mathbf{H}_{LM}\mathbf{d}_{LM} = \mathbf{g} \tag{18}$$

The constant λ represents the LM algorithm's trade-off value between first and second order. When λ is close to zero, LM approximates Newton's method and has minimal impact on the Hessian matrix. When λ is large, LM approaches the steepest descent, and the Hessian matrix approximates an identity matrix. However, the disadvantage of the LM algorithm is that it scales poorly and is only suitable for small data sets Tyagi et al. (2022).

3.3 Basic MOLF-Adapt

In basic MOLF-Adapt MLP training Tyagi et al. (2020), the input weight matrix $\mathbf{W}$ is initialized randomly using zero-mean Gaussian random numbers. To initialize the output weight matrix $\mathbf{W}_{\mathrm{o}}$, we use output weight optimization (OWO) Tyagi et al. (2022). OWO minimizes the error function from equation (10) with respect to $\mathbf{W}_{\mathrm{o}}$ by solving the M sets of N_u equations in N_u unknowns given by

$$\mathbf{C} = \mathbf{R} \cdot \mathbf{W}_{\mathbf{o}}^{\mathbf{T}} \tag{19}$$

where the cross-correlation matrix $\mathbf{C}$ and the auto-correlation matrix $\mathbf{R}$ are $\mathbf{C} = \frac{1}{N_v}\sum_{p=1}^{N_v} \mathbf{X}_{\mathrm{ap}} \cdot \mathbf{t}_{\mathrm{p}}^T$ and $\mathbf{R} = \frac{1}{N_v}\sum_{p=1}^{N_v} \mathbf{X}_{\mathrm{ap}} \cdot \mathbf{X}_{\mathrm{ap}}^T$ respectively. In terms of optimization theory, solving equation (19) is merely Newton's algorithm for the output weights Tyagi et al. (2022). After initialization of $\mathbf{W}$, $\mathbf{W}_{\mathrm{oi}}$, $\mathbf{W}_{\mathrm{oh}}$, we begin a two step procedure in which we modify $\mathbf{W}$ and perform OWO to modify $\mathbf{W}_{\mathrm{o}}$. In the $\mathbf{W}$ modification step, we first find the input weight negative gradient matrix $\mathbf{G}$ and solve

$$\mathbf{D} \cdot \mathbf{R}_{\mathrm{i}} = \mathbf{G} \tag{20}$$

for $\mathbf{D}$, where $\mathbf{R}_{\mathrm{i}} = \frac{1}{N_v}\sum_{p=1}^{N_v} \mathbf{x}_{\mathrm{ap}} \cdot \mathbf{x}_{\mathrm{ap}}^T$. It has been shown Tyagi et al. (2020) that the improved input weight change matrix $\mathbf{D}$ is the negative gradient matrix that results when the inputs are whitened. In MOLF, the idea is to use an N_h dimensional learning factor vector $\mathbf{z}$ and to write the updated outputs as

$$\begin{aligned} y_p(i) &= \sum_{n=1}^{N+1} w_{oi}(i,n)x_p(n) + \sum_{k=1}^{N_h} w_{oh}(i,k)f(n_p(k)) \\ n_p(k) &= \sum_{n=1}^{N+1} (w(k,n) + z_k d(k,n)x_p(n)) \end{aligned} \tag{21}$$

where, $d(k,n)$ is an element of the matrix $\mathbf{D}$. We use Newton's method to obtain $\mathbf{z}$ by solving

$$\mathbf{H}_{\mathrm{molf}} \cdot \mathbf{z} = \mathbf{g}_{\mathrm{molf}} \tag{22}$$

where $\mathbf{H}_{\mathrm{molf}}$ and $\mathbf{g}_{\mathrm{molf}}$ are the Hessian and negative gradient, respectively, of the error with respect to $\mathbf{z}$. Our implementation of Newton's method solves equation (22) for $\mathbf{z}$ using orthogonal least squares (OLS) Tyagi

et al. (2020). After finding $\mathbf{z}$, we update the input weight matrix $\mathbf{W}$ as

$$\mathbf{W} \leftarrow \mathbf{W} + diag(\mathbf{z}) \cdot \mathbf{D} \tag{23}$$

OWO solves Eq. 19 for the output weights in the second half of a training epoch. However, in every third epoch, the basis functions are pruned. In an MLP, pruning removes less useful weights or hidden units to promote sparsity, prevent overtraining, and reduce testing errors. The previous work from the authors Tyagi et al. (2020) contains the algorithm details.

3.4 Softmax classifier

The softmax classifier Tyagi et al. (2022) is a generalized logistic regression classifier that outputs approximate class probabilities. Structurally, it is a linear model with softmax functions Duda et al. (2012) at the output units. For the p^{th} pattern, it maps the input vector $\mathbf{x}_\mathrm{p}$ to the output class labels as

$$\mathbf{y}_\mathrm{p} = \mathbf{W}_\mathrm{s} \cdot \mathbf{x}_\mathrm{p} \tag{24}$$

where $\mathbf{W}_\mathrm{s}$ is a weight matrix. The performance measure is a cross-entropy loss function Tyagi et al. (2022) defined as

$$E_{softmax} = -\frac{1}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{M} log(\frac{e^{y_p(i)}}{\sum_{j=1}^{M} e^{y_p(j)}}) \tag{25}$$

The softmax classifier is often trained using the L-BFGS training algorithmTyagi et al. (2022).

3.5 Piecewise Linear Unit Activation

Piecewise linear (PWL) functions are composed of ReLU activations Goodfellow et al. (2016). Activations such as sigmoidBishop (2006b) and Abdelouahab et al. (2017) can be approximated using ReLU units. Several investigators have tried adaptive PWL activation functions in MLPsGuarnieri et al. (1999) and deep learningAgostinelli et al. (2015) and have published promising results. One includes hybrid piecewise linear units(PLU) with an activation function that is a combination of Tanh and ReLU activationsNicolae (2018).

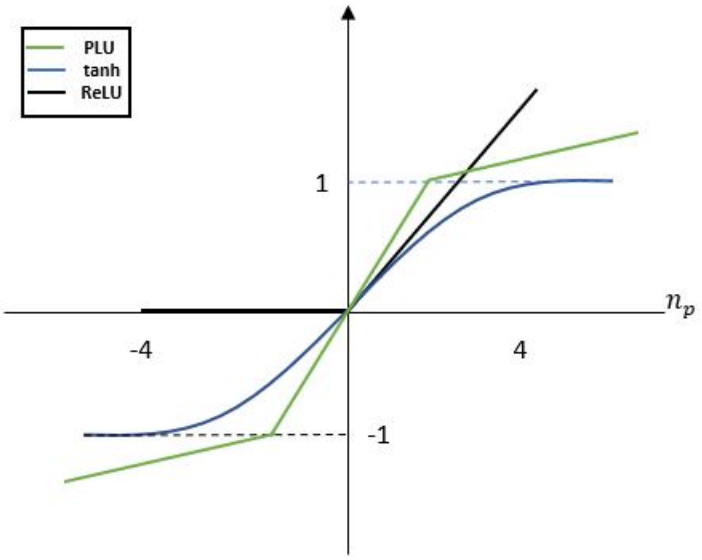


Figure 3: Fixed PWL activations

From figure 3, we see that PLU combines ReLU and Tanh activations. The equation for calculating fixed PWL activations is given as

$$o_p(k) = max\left[\alpha(n_p(k) + c) - c, min(\alpha(n_p(k) + c) - c, n_p(k))\right] \tag{26}$$

where, α and c are user chosen parameters. It is proposed that the α can be a trainable parameter. The paper Nicolae (2018) demonstrates the performance of fixed PWL in an MLP for parametric functions, 3D surface approximation, and invertible network datasets. The author shows that fixed PLUs work better than ReLU functions as PLUs are represented using more hinges than ReLU functions. The author also shows promising results using CNN for the CIFAR-10 Krizhevsky (2009) dataset. The fixed PWL activation function has only three linear segments, hinges $H = 3$, and will not be adaptive until the α parameter is trained in every iteration. Since the H is fixed and there is minimal or no training, there is no universal approximation.

3.6 Piecewise Linear Activation

An alternate piecewise linear activation has been demonstratedAgostinelli et al. (2015), designed explicitly for deep networks with trainable PWL activations. This method, therefore, outperforms the fixed PWL activation in section 3.5. The adaptive PWL activations here can equal those of section 3.5 and generate more complicated curves. The author has implemented an adaptive piecewise linear activation unit where the number of hinges H is a user-chosen hyperparameter. The author shows the best results for CIFAR-10 data using $H = 5$ and $H = 2$ and for CIFAR-100 data using $H = 2$ and $H = 1$(no activation hinge training). The initialization for their adaptive activations is not correctly specified.

The equation for calculating the adaptive activation is given as

$$o_p(k) = max(0, n_p(k)) + \sum_{s=1}^{H} a_k^s \cdot max(0, -n_p(k) + b_k^s) \tag{27}$$

where, a_i^s and b_i^s for $i = 1..H$ are learned using gradient descent. a_i variables control the slopes of the linear segments and b_i^s determine the locations of sample points.

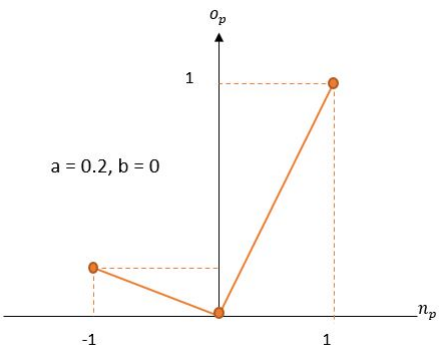


Figure 4: 2 ReLU curves

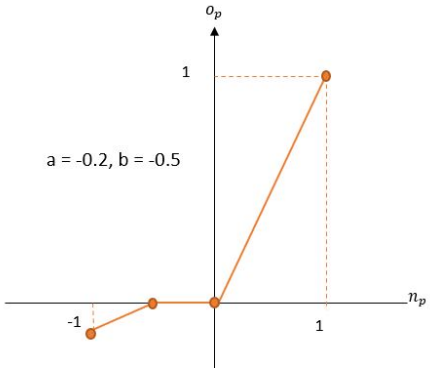


Figure 5: 4 ReLU curves

figure 4 shows adaptive PWL with slope a as 0.2 and b as 0. Similarly, figure 5 shows adaptive PWL with slope a as -0.2 and b as -0.5.

4 Proposed work

4.1 Mathematical Background

Section 3.6 describes an adaptive PWL activation that trains the locations and slopes of the hinges. The author claims that a small number of hinges achieved better results. In section 3.5, we see that a network

with two hinges outperforms ReLU for a particular application. Only one hinge is needed on the PWL curve to approximate a linear output. Similarly, to approximate a quadratic output, the number of hinge sets on the PWL curve should be more significant than three. Therefore, the number of hinges should not be less for more complicated datasets. This can also result in fewer hidden layers and filters as the network does not need to train longer.

We further investigate the use of PWL activations in CNNsRane (2016), and we first determine that PWL activations can approximate any other existing activation functions. Consider a CNN filter's net function n_1 defined as

$$n_1 = t + \sum_{m=1}^{N} w_i(m) \cdot x(m) \tag{28}$$

where t is the threshold, $w_i(m)$ is the m^{th} filter weight, and $x(m)$ is the m^{th} input to the net function. The filter can be represented as $\{\mathbf{w_i}, t\}$. The continuous PWL activation $f(n_1)$ is

$$f(n_1) = \sum_{k=1}^{N_s} a_k \cdot r(n_1 - ns_k) \tag{29}$$

where N_s denotes the number of segments in the PWL curve, $r()$ denotes a ramp (ReLU) activation, and ns_k is the net function value at which the k^{th} ramp switches on.

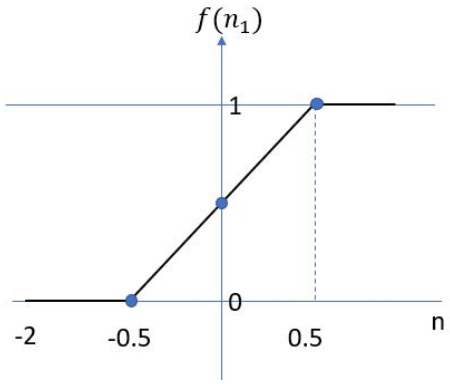


Figure 6: Approximate sigmoid using 2 ReLU curves

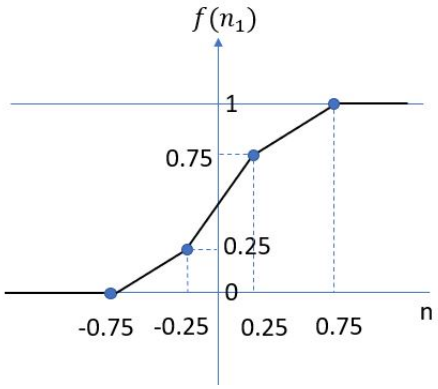


Figure 7: Approximate sigmoid using 4 ReLU curves

Figures 6 and 7 show approximate sigmoid curves generated using ReLU activations where figure 6 has $N_s = 2$ ReLU curves and figure 7 has $N_s = 4$ ReLU curves. Comparing the two figures, we see that larger values of N_s lead to better approximation. The contribution of $f(n_1)$ to the j^{th} net function $n_2(j)$ in the following layer is

$$+n_2(j) = f(n_1) \cdot w_o(j) \tag{30}$$

Decomposing the PWL activation into its N_s components, we can write

$$\begin{aligned} +n_2(j) &= \sum_{k=1}^{N_s} a_k \cdot r(n_1 - ns_k) \cdot w_o(j) \\ &= \sum_{k=1}^{N_s} w'_o(j,k) \cdot r(n_1(k)) \end{aligned} \tag{31}$$

where $w'_o(j,k)$ is $a_k \cdot w_o(j)$ and $n_1(k)$ is $n_1 - ns_k$. A single PWL activation for filter $\{\mathbf{w_i}, t\}$ has now become N_s ReLU activations $f(n_1(k))$ for N_s filters, where each ramp $r(n_1 - d_k)$, is the activation output of a filter. These N_s filters are identical except for their thresholds. Although ReLU activations are efficiently computed, they have the disadvantage that back-propagating through the network activates a ReLU unit only when the

net values are positive and zero; this leads to problems such as dead neuronsMaas (2013a), which means if a neuron is not activated initially or during training, it is deactivated. This means it will never turn on, causing gradients to be zero, leading to no training of weights. Such ReLU units are called dying ReLULu (2020). Using section **??**, In the next section, we will define a much more robust PWL calculation which can be initialized using any pre-defined activation such ReLUNair & Hinton (2010), leaky ReLUMaas (2013b) and also which can be differentiable.

4.2 Piecewise Linear Activations(PLA) and initialization

PWL activations in subsection **??** have some limitations. The calculation fails when the distance between the heights of two hinges is very close, which can happen as we train the hinges. In this section, we derive new notation and calculate PWL activation activation using linear interpolation.

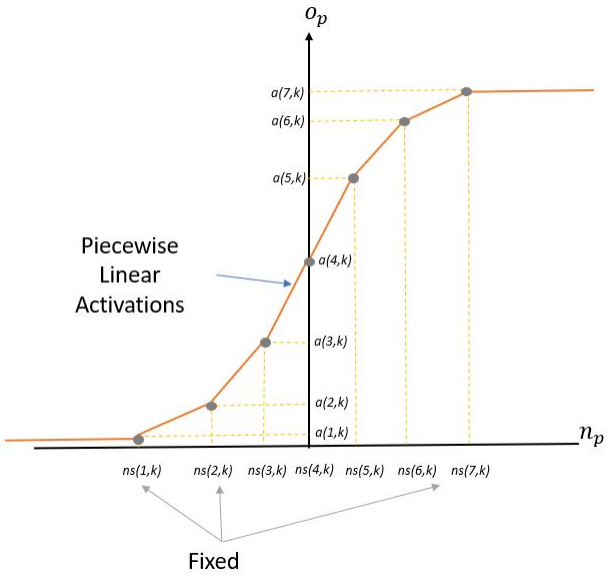


Figure 8: Piecewise Linear Curve

Figure 8 show a PWL activation for K hidden units, which consists of multiple ramps where $ns(1,k)$ is the first hinge of the k^{th} hidden unit and $a(1,k)$ is its activation value. These hinge values ns are constant throughout training. From the figure, we can observe that the PWL curve which passes through the activation of each of the 7 hinges. We define total number of hinges as H. The equation for the above figure is given in equation **??**. Let s denote the maximum value of a net function, and r denote the minimum value of the net function. The activations are calculated between two points, where the net values between the first two hinges are calculated with $ns(1,k)$ denoted as m_1 and $ns(2,k)$ denoted as m_2. Similarly, activations output between the next two hinges are calculated by denoting $ns(2,k)$ denoted as m_1 and $ns(3,k)$ denoted as m_2. We do this for H hinges. m_1 and m_2 for each hinge is calculated as $m_1 = \lceil \frac{n_p}{\delta ns} \rceil$ and $m_2 = m_1 + 1$. Given the net function $n_p(k)$, $o_p(k)$ is calculated as,

$$w_{1p}(k) = \frac{ns(m_2,k) - n_p(k)}{ns(m_2,k) - ns(m_1,k)} \tag{32}$$

$$w_{2p}(k) = \frac{n_p(k) - ns(m_1,k)}{ns(m_2,k) - ns(m_1,k)} \tag{33}$$

$$o_p(k) = \begin{Bmatrix} a(H,k) & for & n_p(k) > s \\ w_{1p}(k) \cdot a(m_1,k) + w_{2p}(k) \cdot a(m_2,k) & for & s > n_p(k) > r \\ a(1,k) & for & n_p(k) < r \end{Bmatrix} \tag{34}$$

where, $w_{1p}(k)$ and $w_{2p}(k)$ are the slope equation for each of the two hinges for k^{th} hidden unit. $n_p(k)$ is the p^{th} pattern and k^{th} hidden unit net function and $o_p(k)$ is its activation function. For all the activation values less $ns(1,k)$ has zero slope hence $n_p(k) = a(1,k)$. Similarly, for all the values greater than $ns(H,k)$ has zero slope; therefore, $n_p(k) = a(H,k)$.

4.3 Example of PWL activation calculation

For initialization of PWL activation, we first need to initialize the PWL using the most widely used activations such as sigmoidBishop (2006b), ReLUNair & Hinton (2010) and leaky ReLUMaas (2013b), etc. We then decide the total number H on the net function. This method can be done individually for each of the hidden units. This paper uses the same number of ns hinges for each hidden unit. We find the minimum and maximum hinge values from the net function output. This is achieved by randomly selecting data from each of the classes and performing convolution, then selecting the minimum and maximum value from the convolution output.

Below, we show the calculation for PWL activations for $k = 1$ hidden unit as follows. As discussed above, we first need to find the initial activation. For this example, we use sigmoid activation as shown in figure 9

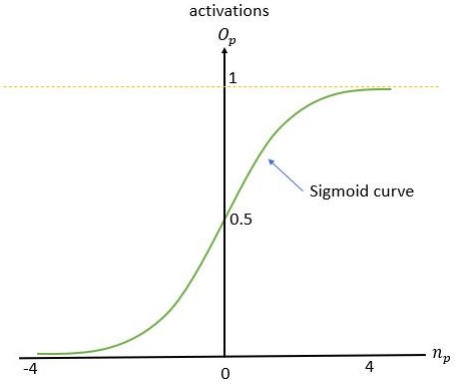


Figure 9: Sigmoid Curve

The figure shows that x axis is the net function n_p and the y axis is its corresponding sigmoid activation. The range of sigmoid is from 0 to 1. Step 2 is finding the minimum and maximum value from convolution output. In this case, we select the values as -4 and 4. Now, we need to decide ns samples on the sigmoid curve. This step is user chosen. Here, for this example we choose $H = 7$. We show our selection of hinges and activations in table format as shown in Table 1

H	1	2	3	4	5	6	7
Fixed hinges (ns_1)	-4	-2.67	-1.33	0	1.33	2.67	4
Activations for hinges(a_1)	0.02	0.07	0.21	0.5	0.79	0.94	0.98

Table 1: PWL samples and activations for one hidden unit

From table 1, we can see that there are total $H = 7$ hinges ranging from -4 to 4, which are our minimum and maximum values from the net function and its sigmoid activations as activation samples a. Finally, we plot

these points on the sigmoid curve from figure 9. The curve after plotting these points should look like figure 10

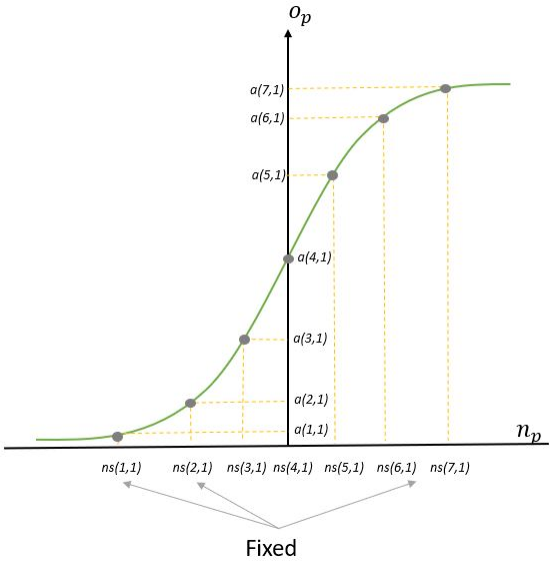


Figure 10: Sigmoid with Fixed Samples

Figure 10 is the plot for a fixed piecewise sigmoid activation for net versus activations values where 7 hinges are plotted onto the sigmoid curve. For the final piecewise linear curve, we remove the sigmoid curve and linearly join 2 points using the linear interpolation technique.

Linear interpolation involves estimating a new value of a function between two known fixed points Hazewinkel (2001)Davis (1963).

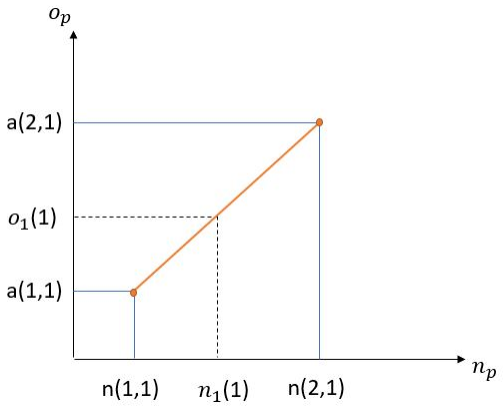


Figure 11: Linear interpolation between 2 points

figure 11 relates the use of linear interpolation between 2 fixed ns points. If we have a new sample net value $n_1(1)$, its corresponding activation value is as shown in the figure. To find $o_1(1)$ between $a(1,1)$ and $a(2,1)$, we use the following equation.

$$o_1(1) = \frac{ns(2,1) - n_1(1,1)}{ns(2,1) - ns(1,1)} \cdot a(1,1) + \frac{n_1(1) - ns(1,1)}{ns(2,1) - ns(1,1)} \cdot a(2,1) \tag{35}$$

Finally, we use the equation 34 to find all the activation output. The plot of net versus activation is shown in figure 8.

4.4 PLA gradients

The above-discussed PWL activations $\mathbf{A}$ are trained via the steepest descent. The negative gradient matrix $\mathbf{G_a}$ with respect to E_{ce} is calculated as,

$$g_a(k,m) = -\frac{\partial E_{ce}}{\partial a(k,m)} \tag{36}$$

where k is the hidden unit number and m is the H^{th} hinge.

$$g_a(k,m) = \frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}(t_p(i) - y_p(i)) \cdot \frac{\partial y_p(i)}{\partial a(u,m)} \tag{37}$$

$$\frac{\partial y_p(i)}{\partial a(u,m)} = w_{oh}(i,u) \cdot \frac{\partial o_p(i)}{\partial a(u,m)} \tag{38}$$

$$\frac{\partial o_p(i)}{\partial a(u,m)} = w_{oh}(i,u) \cdot ((\delta(m-m_1) \cdot w_1(p,u)) + (\delta(m-m_2) \cdot w_2(p,u))) \tag{39}$$

where, for the p^{th} pattern and k^{th} hidden unit of the net value we find m_1 and m_2, where the p^{th} pattern of k^{th} hidden unit of the net value lies between the two fixed piecewise linear sample values m_1 and m_2 of the u^{th} hidden unit as described in the search algorithm. Also we need to find $w_1(p,u)$ and $w_2(p,u)$ from equations 32 and 33. A search algorithm is used to find the correct m sample for a particular pattern's hidden unit is foundRane (2016). The equation 39 solves for the p^{th} patterns u^{th} hidden unit of the piecewise linear activations and accumulates the gradient for all the p^{th} patterns of their respective u^{th} hidden units.

Adams optimizerAda (2014) is used to find the learning factor and update the activation weights. which are updated as follows

$$\mathbf{A} = \mathbf{A} + z \cdot \mathbf{G_a} \tag{40}$$

4.5 PLA OLF

Using the gradient $\mathbf{G_a}$, the optimal learning factor for activations training is calculated as, The activation function vector $\mathbf{o}_p$ can be related to its gradient as,

$$\mathbf{o}_p(k) = w_1(p,k) \cdot [a(k,m_1) + z \cdot g_o(k,m_1)] + w_2(p,k) \cdot [a(k,m_2) + z \cdot g_o(k,m_2)] \tag{41}$$

The first partial derivative of E with respect to z is

$$\frac{\partial E}{\partial z} = \frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}(t_p(i) - y_p(i)) \cdot \frac{\partial y_p(i)}{\partial z} \tag{42}$$

where

$$\frac{\partial y_p(i)}{\partial z} = \sum_{k=1}^{N_h} w_{oh}(i,k) \cdot ((w_1(p,k) \cdot g_o(k,m_1)) + (w_2(p,k) \cdot g_o(k,m_2))) \tag{43}$$

where, m_1 and m_2 for the p^{th} pattern and k^{th} hidden unit of the net vector $n_p(k)$ is again found, and find $g_o(k,m_1)$ and $g_o(k,m_2)$ from the gradient calculated from equations (37, 38, 39)

Also, the Gauss-NewtonShepherd (1997) approximation of the second partial is

$$\frac{\partial^2 E(z)}{\partial z^2} = \frac{2}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{M} \left[\frac{\partial y_p(i)}{\partial z} \right]^2 \tag{44}$$

Thus, the learning factor is calculated as

$$z = \frac{\frac{-\partial^2 E(z)}{\partial z^2}}{\frac{\partial E}{\partial z}} \tag{45}$$

After finding the optimal learning factor, the piecewise linear activations, $\mathbf{A}$, are updated in a given iteration as

$$\mathbf{A} = \mathbf{A} + z \cdot \mathbf{G_a} \tag{46}$$

where $\mathbf{z}$ is a scalar optimal learning factor and $\mathbf{G_a}$ is the gradient matrix calculated in equation 45 and 36.

The Adapt-ACT-OLF algorithm can be summarized as follows:

Algorithm 1 Adapt-ACT-OLF algorithm

1: Initialize $\mathbf{W}, \mathbf{W_{oi}}, \mathbf{W_{oh}}, N_{it}$
2: Initialize Fixed hinges **ns** and hinge activation **a** as described in subsection 4.3 , it$\leftarrow$ 0
3: **while** it $< N_{it}$ **do**
4: Find gradient G and G_{hwo} from equations 36 and 20.
5: Find learning factor z
6: Calculate gradient and learning factor for activation from equation 36 and equation 45 respectively and update the activations as in equation 46.
7: **OWO step** : Solve equation (19) to obtain $\mathbf{W_o}$
8: it $\leftarrow$ it + 1
9: **end while**

4.6 PLA adavantages

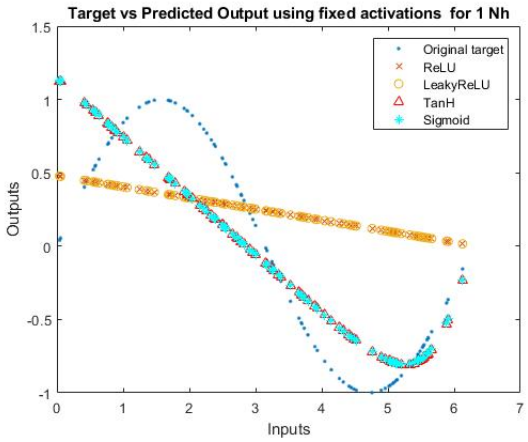
In this subsection, we demonstrate the advantage of adaptive activations using a simple sine data function and a more complicated Rosenbrock functionRosenbrock (1960). For each of the experiments, we will compare the approximation results for Basic MOLF-Adapt algorithm explained in section 3.3 with constant activation functions such as sigmoid, Tanh,ReLU and leaky ReLU and with Adapt-ACT-OLF algorithm described in 1 with initial activations as sigmoid, Tanh, ReLU and leaky ReLU respectively.

4.6.1 Sinusoidal Approximation

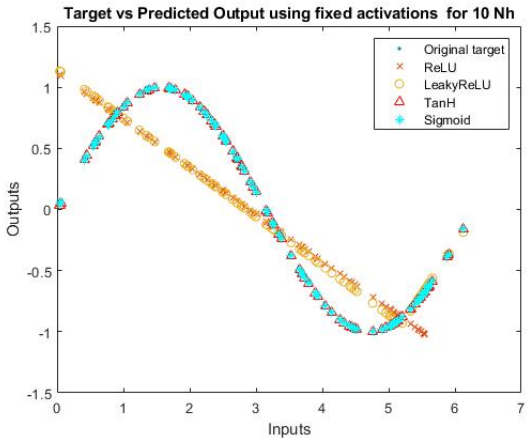
The sine data used for training is generated using one feature and 5000 uniformly distributed random samples in the range of 0 to 4π. The output generated is the sine function of the uniformly distributed random samples. Similarly, we randomly choose 100 uniformly distributed random samples for testing data in the range of 0 to 2π. The training for each of the basic molf-adapt and Adapt-ACT-OLF algorithm performed using 100 iterations. We use 1 hidden unit for each training algorithm with 20 samples for the Adapt-ACT-OLF algorithm. We also show results for basic molf-adapt with 10 hidden units. Here, we used 20 samples as piecewise linear activations would have 10 smaller ReLu like functions.

Figure 12 shows Basic MOLF-Adapt model prediction with one hidden unit after 100 iterations using the above-mentioned fixed activations. We can observe From the figure that none of the fixed activations approximate the sine function labeled as original target in figure12a. Next, we increase the hidden unit to

10. Figure 12b shows Basic MOLF-Adapt model prediction with ten hidden units after 100 iterations using the above-mentioned fixed activations. From the figure, we can observe that Tanh and sigmoid functions, which are curve-based approximates, but activation functions such as relu and leaky relu, which are a form of piecewise linear, do not approximate even if there are more hidden units. Next, we will demonstrate adaptive activations trained with initial activations as mentioned above.



(a) Basic MOLF-Adapt with fixed activations with one hidden unit



(b) Basic MOLF-Adapt with fixed activations with ten hidden units

Figure 12: Basic MOLF-Adapt with fixed activations

Figure 13 shows Adapt-ACT-OLF model prediction with ten hidden units after 100 iterations with initial activations as each of the mentioned fixed activations. From the figure, we can observe that the model trained using the adaptive activations approximates the function with similar output no matter what initial activations are used.

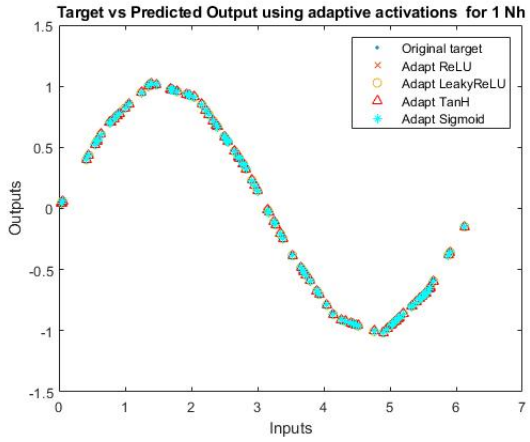
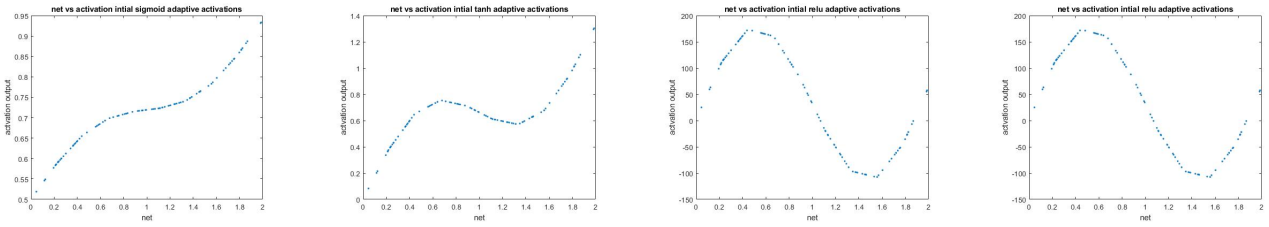


Figure 13: Adaptive activations algorithm with 20 samples

Figure 14 shows the hidden units after the model is trained for each fixed initial hidden unit. We can observe that figure 14c and figure 14d are trained with initial activation as ReLU and Leaky ReLU mimics the input versus output, which is sinusoidal which very high activation output but the activation outputs for sigmoid and Tanh are not as high as shown in figure 14a and figure 14b. We also observed that as the activation output is so high, the trained output weights were equally small, with the input weight range being close to similar to the model trained with Tanh's activation.

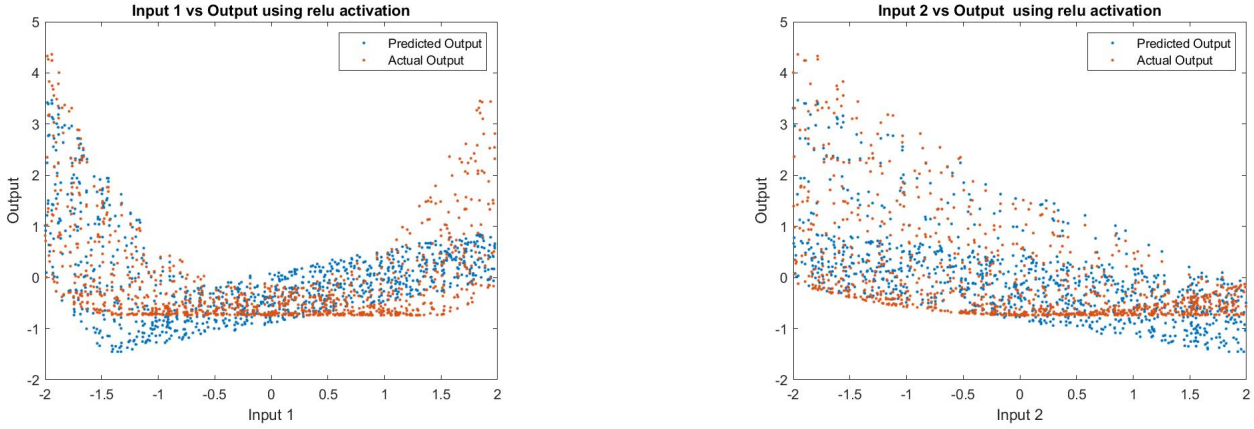


(a) Sigmoid (b) Tanh (c) ReLU (d) Leaky ReLU

Figure 14: Adaptive Activation hidden units

4.6.2 Rosenbrock Approximation

In this subsection, we demonstrate an approximation of the rosenbrockRosenbrock (1960). The input is generated using constant values a= 1 and b = 100, and 1000 uniformly distributed random samples are generated for each of the two inputs. The Output generated is the rosenbrockRosenbrock (1960) function. Here, we normalize the inputs and outputs with zero mean and standard deviation on 1 for ease of training. We saw no difference in results.

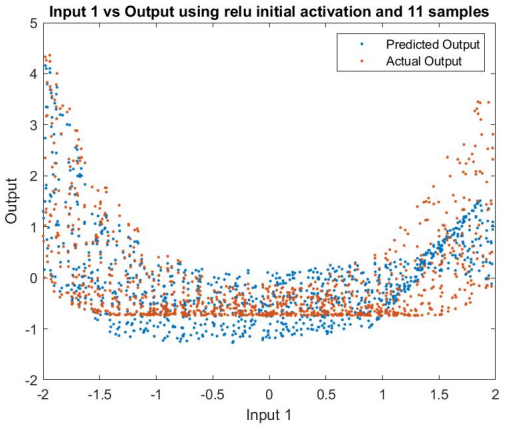


(a) ReLU Activations Input 1 vs Output (b) ReLU Activations Input 2 vs Output

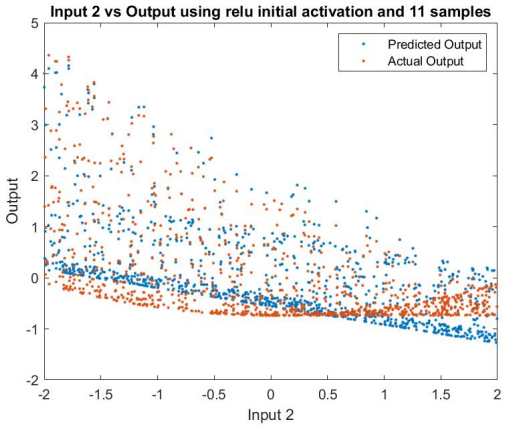
Figure 15: Rosenbrock Approximation with Fixed ReLU Activations

Relu Activations Figure 15 shows the Rosenbrock approximation using ReLU activations, and Figure **??** shows the Rosenbrock approximation using adaptive ReLU activations. Here, Subfigure 15a shows a scatter plot of predicted versus the actual Output for the trained model using fixed ReLU activations. Subfigure **??** shows results for the adaptive model trained using ReLU as initial activations using 11 samples. Comparing both the scatter plots, we can observe that the approximation results of the adaptive ReLU model are better than the model trained using fixed ReLU activation. Similarly, Subfigure 15b shows a scatter plot of predicted versus the actual Output for the trained model using fixed ReLU activations for the second input. Subfigure **??** shows results for the adaptive model trained using ReLU as initial activations using 11 samples. Again we can observe by comparing both scatter plots that the approximation results of the adaptive ReLU model is better than the model trained using fixed ReLU activation.

Leaky Relu Activations Figure 17 shows the Rosenbrock approximation using Leaky ReLU activations, and Figure 18 shows the Rosenbrock approximation using adaptive Leaky ReLU activations. Here, Subfigure 17a, shows a scatter plot of predicted versus the actual Output for the trained model using fixed Leaky ReLU activations. Subfigure 18a shows results for the adaptive model trained using Leaky ReLU as initial activations using 11 samples. Comparing both the scatter plots, we can observe that the approximation

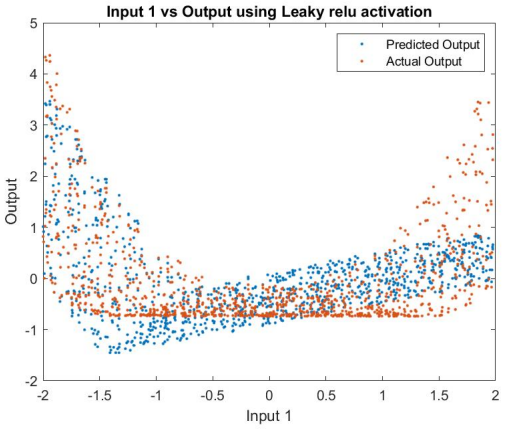


(a) Adaptive ReLU Activations Input 1 vs Output

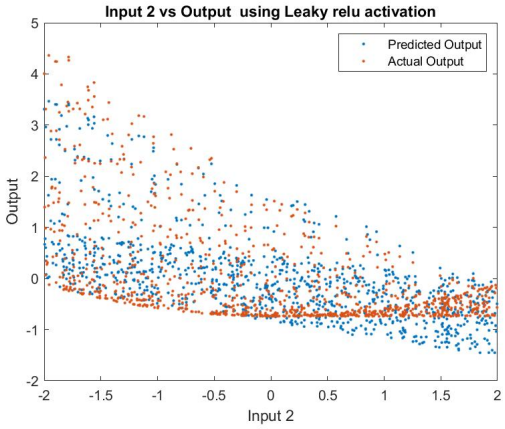


(b) Adaptive ReLU Activations Input 2 vs Output

Figure 16: Rosenbrock Approximation with Adaptive ReLU Activations with 11 samples



(a) Leaky ReLU Activations Input 1 vs Output

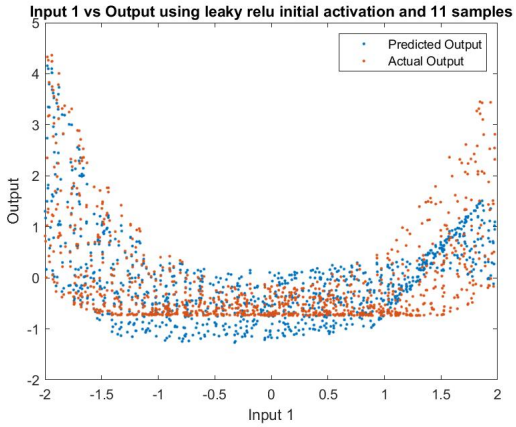


(b) Leaky ReLU Activations Input 2 vs Output

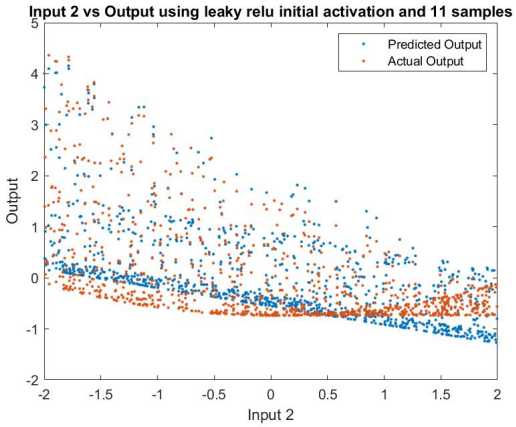
Figure 17: Rosenbrock Approximation with Fixed Leaky ReLU Activations

results of the adaptive Leaky ReLU model is better than the model trained using fixed Leaky ReLU activation. Similarly, Subfigure 17b , shows a scatter plot of predicted versus the actual Output for the trained model using fixed Leaky ReLU activations for the second input. Subfigure 18b shows results for the adaptive model trained using Leaky ReLU as initial activations using 11 samples. Again, we can observe by comparing both scatter plots that the approximation results of the adaptive Leaky ReLU model are better than the model trained using fixed Leaky ReLU activation.

Sigmoid Activations Figure 19 shows the Rosenbrock approximation using Sigmoid activations, and Figure 20 shows the Rosenbrock approximation using adaptive Sigmoid activations. Here, Subfigure 19a, shows a scatter plot of predicted versus the actual Output for trained model using fixed Sigmoid activations. Subfigure 20a shows results for the adaptive model trained using Sigmoid as initial activations using 11 samples. Comparing both the scatter plots, we can observe that the approximation results of the adaptive Sigmoid model is better than the model trained using fixed Sigmoid activation. Similarly, Subfigure 19b , shows a scatter plot of predicted versus the actual Output for the trained model using fixed Sigmoid activations for the second input. Subfigure 20b shows results for the adaptive model trained using Sigmoid as initial activations using 11 samples. Again, we can observe by comparing both scatter plots that the

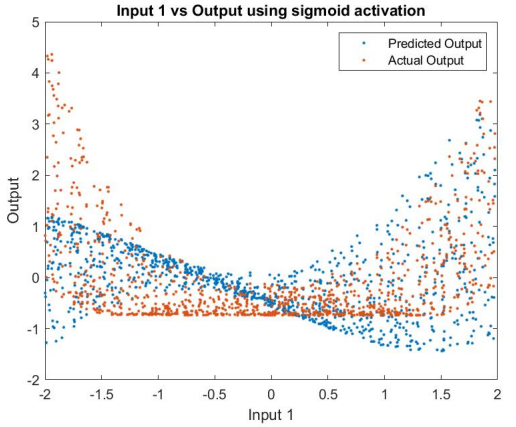


(a) Adaptive Leaky ReLU Activations Input 1 vs Output

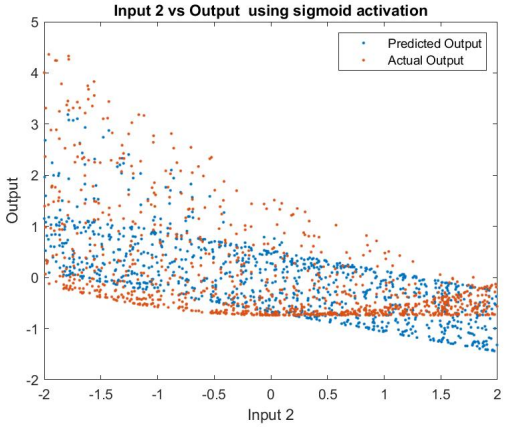


(b) Adaptive Leaky ReLU Activations Input 2 vs Output

Figure 18: Rosenbrock Approximation with Adaptive Leaky ReLU Activations with 11 samples



(a) Sigmoid Activations Input 1 vs Output

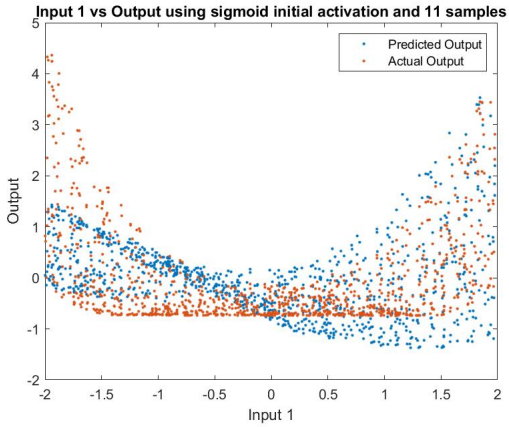


(b) Sigmoid Activations Input 2 vs Output

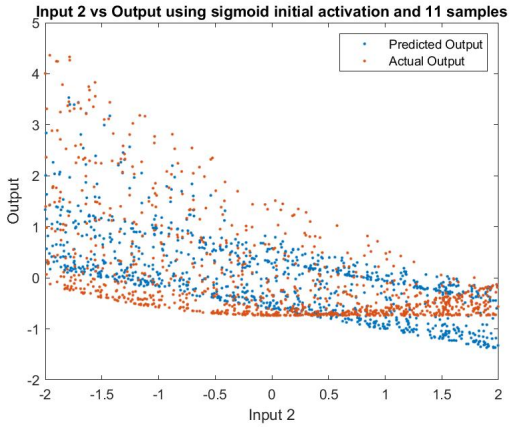
Figure 19: Rosenbrock Approximation with Fixed Sigmoid Activations

approximation results of the adaptive Sigmoid model is better than the model trained using fixed Sigmoid activation.

Tanh Activations Figure 21 shows the Rosenbrock approximation using Tanh activations and Figure 22 shows the Rosenbrock approximation using adaptive Tanh activations. Here, Subfigure 21a, shows a scatter plot of predicted versus the actual Output for the trained model using fixed Tanh activations. Subfigure 22a shows results for the adaptive model trained using Tanh as initial activations using 11 samples. Comparing both the scatter plots, we can observe that the approximation results of the adaptive Tanh model is better than the model trained using fixed Tanh activation. Similarly, Subfigure 21b , shows a scatter plot of predicted versus the actual Output for the trained model using fixed Tanh activations for the second input. Subfigure 22b shows results for the adaptive model trained using Tanh as initial activations using 11 samples. Again, we can observe by comparing both scatter plots that the approximation results of the adaptive Tanh model are better than the model trained using fixed Tanh activation.

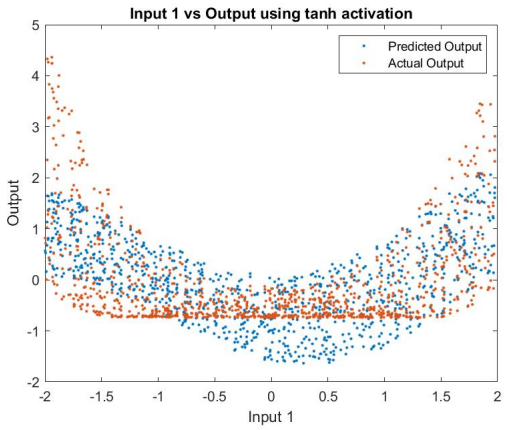


(a) Adaptive Sigmoid Activations Input 1 vs Output

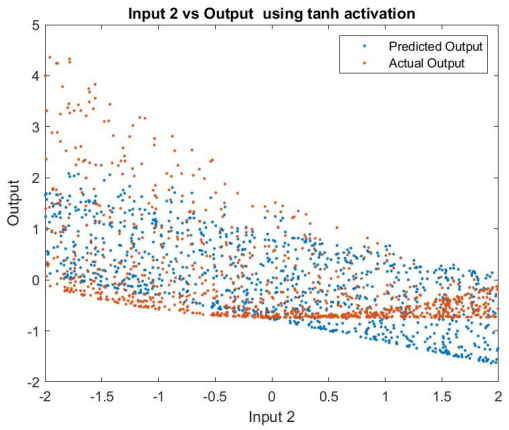


(b) Adaptive Sigmoid Activations Input 2 vs Output

Figure 20: Rosenbrock Approximation with Adaptive Sigmoid Activations with 11 samples



(a) Tanh Activations Input 1 vs Output



(b) Tanh Activations Input 2 vs Output

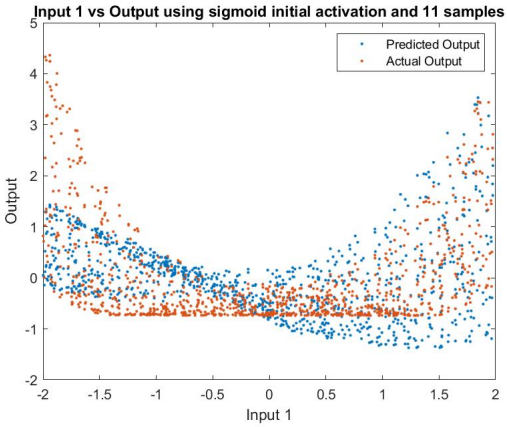
Figure 21: Rosenbrock Approximation with Fixed Tanh Activations

5 Experimental Methods and Results

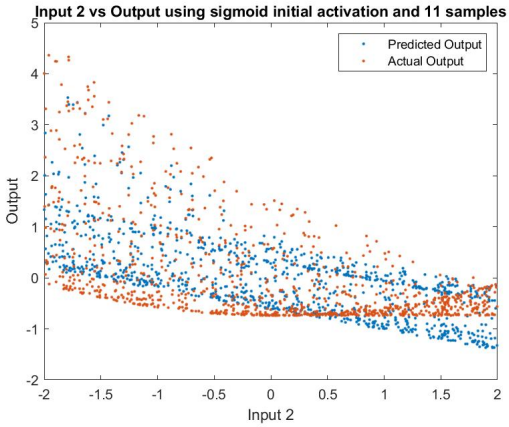
In this section, we discuss experiment results for the proposed algorithm in which we demonstrate relative testing results for widely available approximation and classification data. Finally, we show results for a shallow convolutional neural network architecture. The computational cost is measured on a Windows 10, Intel i-7, 3 Mhz CPU platform with 32 GB RAM. In this section, we demonstrate network performance comparison between Adapt-ACT-OLF, Adapt-ACT-OLF,CG-MLPRane et al. (2023) Tyagi et al. (2014)Tyagi et al. (2022) and LM Battiti (1992)Hagan & Menhaj (1994) for approximation and classification data. We also show results for shallow convolution neural networks with custom networks and deep CNNs using transfer learning.

5.1 Approximation Datasets Results

From Table 2, we can observe that Adapt-OLF is the top performer in 5 out of the 7 data sets in terms of testing MSE. The following best-performing algorithm is MOLF-Adapt for weather data with a smaller margin and is also tied in comparison to testing MSE for twod datasets. Adapt-OLF has slightly more parameters depending on the number of samples, but the testing MSE is substantially reduced. The next



(a) Adaptive Tanh Activations Input 1 vs Output



(b) Adaptive Tanh Activations Input 2 vs Output

Figure 22: Rosenbrock Approximation with Adaptive Tanh Activations with 11 samples

Dataset	SCG/Nh	CG-MLP/Nh	LM/Nh	MOLF-Adapt L_ReLU/Nh	Adapt-OLF 3samples/Nh	Adapt-OLF 5 samples/Nh	Adapt-OLF 9samples/Nh
Oh7	1.971/30	1.52/100	**1.41**/30	1.51/15	1.49/20	1.46/20	1.44/15
White Wine	0.6/20	0.56/100	0.57/30	0.55/30	**0.54/100**	0.56/20	**0.54/100**
twod	0.5/30	0.23/100	0.17/15	**0.149/15**	**0.149/15**	0.18/15	0.15/15
Superconductor	230.91/15	180.21/100	170.2/100	144.46/100	142.53/100	**139.62/100**	142.53/100
F24	1.14/20	0.31/100	0.30/30	0.281/100	0.282/100	0.283/100	**0.279/100**
Concrete	61.11/5	34.64/30	32.12/20	32.29/100	**30.70/100**	34.34/10	35.56/100
Weather	316.68/15	283.23/30	286.27/30	**283.20/10**	284.39/15	284.34/15	284.73/15

Table 2: 10-fold cross validation mean square error (MSE) testing results for approximation dataset, (best testing MSE is in bold)

best performer is LM for the oh7 dataset. However, LM being a second-order method, its performance comes at a significant cost of computation – almost two orders of magnitude greater than the rest of the models proposed

5.2 Classifier Datasets Results

From Table 3, we can observe that Adapt-OLF is the top performer in 5 out of the 6 data sets in terms of testing MSE. The following best-performing algorithm is MOLF-Adapt for weather data, which has a smaller margin. As LM requires a significant cost of computation, it cannot be used for pixel-based inputs hence lose out on a majority of the image classification-based use cases.

5.3 CNN Results

5.3.1 Shallow CNN results

This section demonstrates shallow CNN results with ReLU, leaky ReLU, and adaptive activations. We will be using one, two, and three VGG blocks Simonyan & Zisserman (2014). We will also be using the CIFAR-10 dataset for benchmarking our results. Figure 23 is the One VGG block.

Dataset	SCG/Nh	CG-MLP/Nh	LM/Nh	MOLF-Adapt L_ReLU/Nh	Adapt-OLF 3samples/Nh	Adapt-OLF 5 samples/Nh	Adapt-OLF 9samples/Nh
GongTrn	10.28/100	10.46/100	8.94/30	**8.62**/30	8.65/30	8.64/30	8.72/30
Comf18	15.69/100	14.50/100	12.63/5	11.83/20	11.93/30	11.87/30	**11.79/30**
f17c	3.22/100	3.69/100	3.96/100	2.45/100	2.38/100	2.41/100	**2.34/100**
Speechless	44.26/100	43.07/100	39.72/100	36.65/100	**35.96/100**	38.94/100	37.6/100
Cover	27.39/100	29.87/100	NA	20.1/30	19.43/30	19.47/30	**19.42/30**
Scrap	25.58/100	20.77/100	NA	19.9/100	19.57/100	**18.8/100**	19.2/100

Table 3: 10-fold cross validation Percentage of Error (PE) testing results for classification dataset, (best testing MSE is in bold)

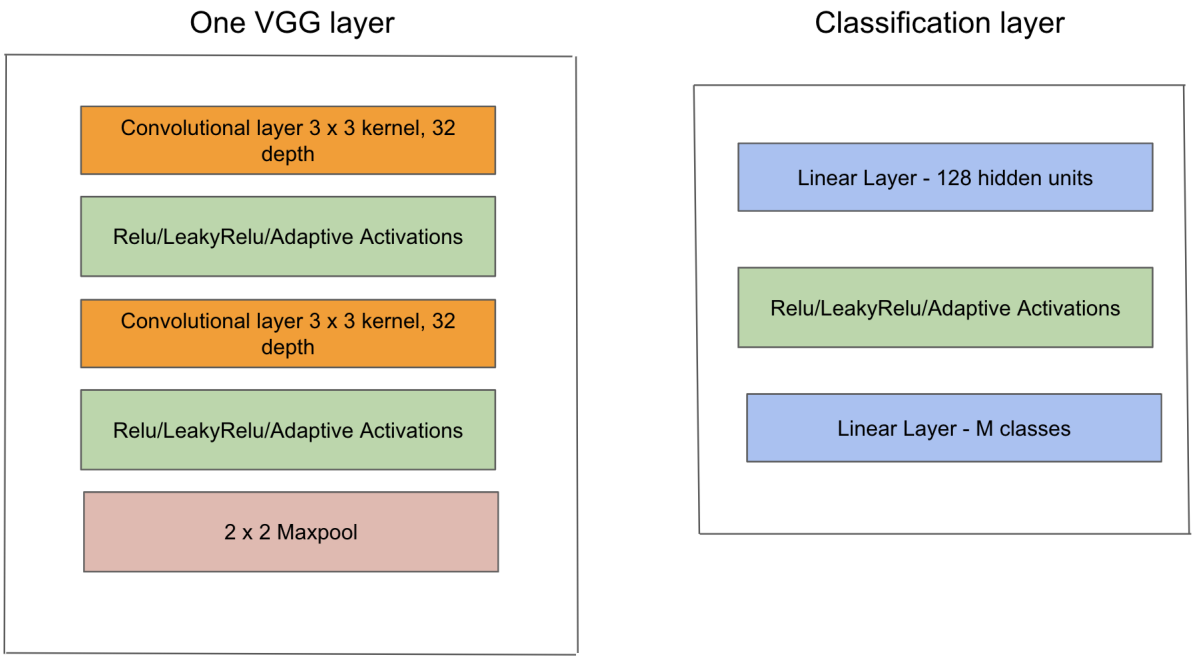


Figure 23: 1 VGG layer with classifier

From Figure 23 we can see that there is one VGG layer and a classification layer. The one VGG layer consists of 2 convolution layers with 3x3 kernel and 32 filters in the first and and 3rd layers, and activations are in the 2nd and 4th layers. The final layer is a 2 x 2 maxpool layer. Similarly, 2 VGG layer network has 2 VGG layers and one classification layer where the first VGG layer has the same configuration as in the figure with 32 filters, and the 2nd VGG layer consists of convolution layers with 3 x 3 kernels and a depth as 64 for each convolution layer as shown in Figure 24

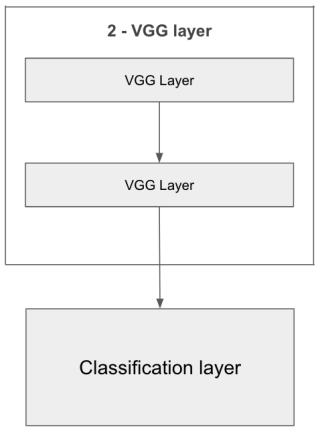


Figure 24: Two VGG layer with classifier

Similarly, in the 3 VGG layer, we have 2 VGG layers as mentioned above, and a 3rd VGG layer has the same configuration as the 2nd VGG layer with 3 x 3 kernels, but the depth is 64 filters for each convolution layer, as shown in Figure 25

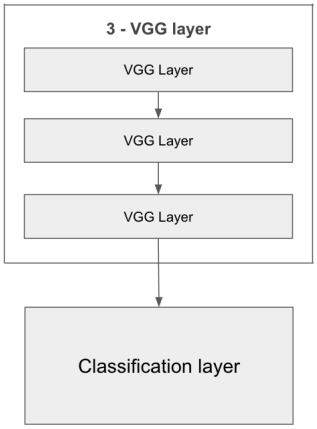


Figure 25: Three VGG layer with classifier

One thing to note here is that when the model is used with ReLU activations, all the activations used are ReLU, similarly with leaky ReLU, but in adaptive activations, the last VGG layer activations are trained and the remaining are not trained and are either kept as ReLU or leaky ReLU. For example, In 2 - VGG layer, the 2nd VGG layer's adaptive activation functions are trained, and the first VGG layer's adaptive activations are not used as trainable parameters. The number of samples(n) used in the activations for the below experiment is kept as 3 as follows [minimum value, 0, maximum value], where the first is the minimum value of the output of the convolution layer before the adaptive activation, the second value is *0*. The final value is the maximum value of the output of the convolution layer before the adaptive activation layer. For example, if we train a 2 VGG layer model, as described above, activations in the VGG1 layer are not trained. The output of the VGG1 layer, the 2 x 2 maxpool output, is used as input to the 2nd VGG layer, where th output of the first convolution layer's maximum and minimum value in the 2nd VGG layer is used. The activation function used in the adaptive activations where the activations are not trainable is leaky ReLU, and also, the initialization of the adaptive activaions is done using leaky ReLU. As suggested earlier, any

activations can be used in the initialization. The decision here was taken based on the results seen in the Table 4 for the model with leaky ReLU activations.

Table 4 shows results for 1-vgg layers, 2-vgg layers, 3-vgg layers model on CIFAR-10Krizhevsky (2009) dataset with Glorot normal initialization From the table we can observe that as the number of vgg layer increase, adaptive activation gives better accuracy. One thing to note is that in 3-VGG layer model, only the 3rd vgg layer activations are trained, and we can observe a significant difference in the accuracy.

Dataset	Models	Weight Initialization	Adaptive Activations	ReLU Activations	LeakyReLU Activations
CIFAR-10	1 - VGG layers	Glorot Normal	**67.8**	66.56	66.45
CIFAR-10	2 - VGG layers	Glorot Normal	**74.2**	71.82	73.09
CIFAR-10	3 - VGG layers	Glorot Normal	**75.53**	72.58	73.3

Table 4: 10-fold cross validation mean square error (MSE) testing results for classification dataset, (best testing MSE is in bold)

5.3.2 Transfer Learning using Deep CNN results

In this section, we use 2 of the widely use Pretrained deep learning models, VGG11 and ResNet18. These models are pretrained using Image net data. We use a transfer learning approach where we modify the final classification layer with a new linear layer with a number of classes equal to 10, which is the total number of classes in CIFAR-10 and train the model for at least 100 iterations. The importance of transfer learning is that the model already has learnt the important features, which helps generate good results with fewer data and iterations. Similarly, in the model with adaptive activations, we change the final linear layer and also change some of the final layers, such as in ResNet18 architecture, we replace the layer 4 basic blocks ReLU activation function with adaptive activations and in VGG11, the ReLU activation after the 7th and 8th convolution layer, which also are the last 2 activations in the feature layer with adaptive activations. Two main reasons for using adaptive activations in the final feature layer. 1st - less number of parameters and 2nd - more complex and abstract are in the deeper layerZeiler & Fergus (2013) . From the results shown in 5, we can observe that adaptive activations give better results but with a slight increase in parameters and hence training time.

Models	Adaptive Activations	ReLU Activations
VGG11	**91.78**	91.44
ResNet18	**95.30**	95.1

Table 5: 10-fold cross validation mean square error (MSE) testing results for classification dataset (best testing MSE is in bold)

6 Conclusion and Future Work

The proliferation of progressively deeper architectures, exemplified by structures like ResNet boasting up to 152 layers, attests to the remarkable success of neural network models. While deep architectures garner considerable attention, exploring shallow CNN remains an open research topic to understand how neural network learning plays out. Amidst these developments, the role of activation functions within neural networks emerges as a paramount consideration, as they imbue networks with essential nonlinearity. Although Rectified Linear Units (ReLUs) have surfaced as a ubiquitous choice, this study introduces a novel and intricate

approach to complex piece-wise linear (PWL) activations within hidden layers. Through experimentation, we have successfully demonstrated the superior efficacy of PWL activations over ReLUs in CNN and dense networks like multilayer perceptrons. A new adaptive piecewise linear activation has been introduced. We demonstrate our results on simple and complex functions and show that fixed activation functions cannot approximate any function. We also show results for various MLP structures compared to PLA activations and different sets of samples with traditional fixed activations for approximation and classification data. Finally, we show shallow CNN results with fixed and PLA activations. From all the experiments, we observed that adaptive activations give better results. The drawback of PLA activations is that they require more computations, but the achieved results are comparatively higher than those with fixed activations. We also observed that applications with curved outputs involve using more PWL samples to approximate. The current study indicates the potency of PWL activations. It contributes a valuable perspective to the ongoing discourse surrounding activation functions, encouraging researchers and practitioners alike to consider the nuanced intricacies of neural network design for optimal performance across many applications.

References

Adam: A method for stochastic optimization, 2014. URL http://arxiv.org/abs/1412.6980. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

Kamel Abdelouahab, Maxime Pelcat, and Francois Berry. Why tanh is a hardware friendly activation function for cnns. In *Proceedings of the 11th International Conference on Distributed Smart Cameras*, ICDSC 2017, pp. 199–201, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450354875. doi: 10.1145/3131885.3131937. URL https://doi.org/10.1145/3131885.3131937.

Forest Agostinelli, M. Hoffman, Peter Sadowski, and P. Baldi. Learning activation functions to improve deep neural networks. *CoRR*, abs/1412.6830, 2015.

P. Baldi, P. Sadowski, and D. Whiteson. Enhanced higgs boson particle search with deep learning. *Physical Review Letters*, 114(11), Mar 2015. ISSN 1079-7114. doi: 10.1103/physrevlett.114.111801. URL http://dx.doi.org/10.1103/PhysRevLett.114.111801.

Phillip J. Barry and Ronald N. Goldman. A recursive evaluation algorithm for a class of catmull-rom splines. In *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '88, pp. 199–204, New York, NY, USA, 1988. Association for Computing Machinery. ISBN 0897912756. doi: 10.1145/54852.378511. URL https://doi.org/10.1145/54852.378511.

Simon A Barton. A matrix method for optimizing a neural network. *Neural Computation*, 3(3):450–459, 1991.

Roberto Battiti. First-and second-order methods for learning: between steepest descent and newton's method. *Neural computation*, 4(2):141–166, 1992.

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006a. ISBN 0387310738.

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006b.

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, USA, 2004.

P. Campolucci, F. Capperelli, S. Guarnieri, F. Piazza, and A. Uncini. Neural networks with adaptive spline activation function. In *Proceedings of 8th Mediterranean Electrotechnical Conference on Industrial Applications in Power Systems, Computer Science and Telecommunications (MELECON 96)*, volume 3, pp. 1442–1445 vol.3, 1996.

C. Charalambous. Conjugate gradient algorithm for efficient training of artificial neural networks. *IEE Proceedings G (Circuits, Devices and Systems)*, 139:301–310(9), June 1992. ISSN 0956-3768. URL https://digital-library.theiet.org/content/journals/10.1049/ip-g-2.1992.0050.

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014. URL http://arxiv.org/abs/1412.3555.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

P. J. Davis. Interpolation and approximationr. *Blaisdell Pub*, 1963.

Jeff Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. *CoRR*, abs/1411.4389, 2014. URL http://arxiv.org/abs/1411.4389.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification*. John Wiley & Sons, 2012.

Roger Fletcher. *Practical Methods of Optimization*. John Wiley & Sons, New York, NY, USA, second edition, 1987.

Xavier Glorot, Antoine Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 15, 01 2010.

Gene H Golub and Charles F Van Loan. *Matrix computations*, volume 3. JHU Press, 2012.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

R. G. GORE, J. LI, M. T. MANRY, L. M. LIU, C. YU, and J. WEI. Iterative design of neural network classifiers through regression. *International Journal on Artificial Intelligence Tools*, 14(01n02):281–301, 2005. doi: 10.1142/S0218213005002107. URL https://doi.org/10.1142/S0218213005002107.

Samantha Guarnieri, Francesco Piazza, and Aurelio Uncini. Multilayer feedforward networks with adaptive spline activation function. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 10:672–83, 02 1999. doi: 10.1109/72.761726.

Varun Gulshan, Lily Peng, Marc Coram, Martin C. Stumpe, Derek Wu, Arunachalam Narayanaswamy, Subhashini Venugopalan, Kasumi Widner, Tom Madams, Jorge Cuadros, Ramasamy Kim, Rajiv Raman, Philip C. Nelson, Jessica L. Mega, and Dale R. Webster. Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs. *JAMA*, 316(22):2402–2410, 12 2016. ISSN 0098-7484. doi: 10.1001/jama.2016.17216. URL https://doi.org/10.1001/jama.2016.17216.

Martin T Hagan and Mohammad B Menhaj. Training feedforward networks with the marquardt algorithm. *IEEE transactions on Neural Networks*, 5(6):989–993, 1994.

Michiel Hazewinkel. "linear interpolation". In *Encyclopedia of Mathematics*, 2001.

Magnus R. Hestenes and Eduard Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards*, 49:409–435, 1952.

Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 6(2):107–116, April 1998. ISSN 0218-4885. doi: 10.1142/S0218488598000094. URL https://doi.org/10.1142/S0218488598000094.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

J J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982. ISSN 0027-8424. doi: 10.1073/pnas.79.8.2554. URL https://www.pnas.org/content/79/8/2554.

Earnest Paul Ijjina and Krishna Mohan Chalavadi. Human action recognition using genetic algorithms and convolutional neural networks. *Pattern Recognition*, 59:199 – 212, 2016. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2016.01.012. URL http://www.sciencedirect.com/science/article/pii/S0031320316000169. Compositional Models and Structured Learning for Visual Recognition.

Ameya Dilip Jagtap, K. Kawaguchi, and G. Karniadakis. Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks. *Proceedings of the Royal Society A*, 476, 2020.

Michael I. Jordan. Serial order: A parallel distributed processing approach. *Advances in psychology*, 121: 471–495, 1997.

Patrik Kamencay, Miroslav Benco, Tomas Mizdos, and Roman Radil. A new method for face recognition using convolutional neural network. *Advances in Electrical and Electronic Engineering*, 15, 11 2017. doi: 10.15598/aeee.v15i4.2389.

W. Kaminski and P. Strumillo. Kernel orthonormalization in radial basis function neural networks. *IEEE Transactions on Neural Networks*, 8(5):1177–1183, Sep. 1997. ISSN 1941-0093. doi: 10.1109/72.623218.

Richard Kijowski, Fang Liu, Francesco Caliva, and Valentina Pedoia. Deep learning for lesion detection, progression, and prediction of musculoskeletal disease. *Journal of Magnetic Resonance Imaging*, n/a (n/a), 1987. doi: 10.1002/jmri.27001. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/jmri.27001.

Alex Krizhevsky. Learning multiple layers of features from tiny images. In *Semantic Scholar*, 2009. URL https://api.semanticscholar.org/CorpusID:18268744.

Paras Lakhani and Baskaran Sundaram. Deep learning at chest radiography: Automated classification of pulmonary tuberculosis by using convolutional neural networks. *Radiology*, 284(2):574–582, 2017. doi: 10.1148/radiol.2017162326. URL https://doi.org/10.1148/radiol.2017162326. PMID: 28436741.

Pramod Lakshmi Narasimha, Walter Delashmit, Michael Manry, Jiang Li, Francisco Maldonado, and Dr Zhang. An integrated growing-pruning method for feedforward network training. *Neurocomputing*, 71:2831–2847, 08 2008. doi: 10.1016/j.neucom.2007.08.026.

Quoc V Le, Jiquan Ngiam, Adam Coates, Abhik Lahiri, Bobby Prochnow, and Andrew Y Ng. On optimization methods for deep learning. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, pp. 265–272, 2011.

Y. LeCun, L. Bottou, G. Orr, and K. Muller. Efficient backprop. In G. Orr and Muller K. (eds.), *Neural Networks: Tricks of the trade*, pp. 9–50. Springer, 1998a.

Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, pp. 9–50, Berlin, Heidelberg, 1998b. Springer-Verlag. ISBN 3540653112.

Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pp. 2278–2324, 1998.

C Lemaréchal. The method of steepest descent for non-linear minimization problems. *Quart. Appl. Math. 2*, pp. 258–261, 1944. doi: https://doi.org/10.1090/qam/10667.

C Lemaréchal. Cauchy and the gradient method. *doc Math extra*, pp. 251–254, 2012.

Kenneth Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly of applied mathematics*, 2(2):164–168, 1944.

Jiang Li, Michael Manry, Li-min Liu, Changhua Yu, and John Wei. Iterative improvement of neural classifiers. In *Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2004*, volume 2, 01 2004.

R P Morgan Lippmann. An introduction to computing with neural nets. *IEEE ASSP Magazine*, 4:4–22, 1987.

Yun Liu, Guolei Sun, Yu Qiu, Le Zhang, Ajad Chhatkuli, and Luc Van Gool. Transformer in convolutional neural networks. *CoRR*, abs/2106.03180, 2021. URL https://arxiv.org/abs/2106.03180.

Lu Lu. Dying ReLU and initialization: Theory and numerical examples. *Communications in Computational Physics*, 28(5):1671–1706, jun 2020. doi: 10.4208/cicp.oa-2020-0165. URL https://doi.org/10.4208%2Fcicp.oa-2020-0165.

Andrew L. Maas. Rectifier nonlinearities improve neural network acoustic models. In *Semantic Scholar*, 2013a. URL https://api.semanticscholar.org/CorpusID:16489696.

Andrew L. Maas. Rectifier nonlinearities improve neural network acoustic models. In *Semantic Scholar*, 2013b. URL https://api.semanticscholar.org/CorpusID:16489696.

S. Marinai, M. Gori, and G. Soda. Artificial neural networks for document analysis and recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27:23–35, 2005.

Martin Fodslette Møller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural networks*, 6(4):525–533, 1993.

Hardik Nahata and Satya P. Singh. *Deep Learning Solutions for Skin Cancer Detection and Diagnosis*, pp. 159–182. Springer International Publishing, Cham, 2020. ISBN 978-3-030-40850-3. doi: 10.1007/978-3-030-40850-3_8. URL https://doi.org/10.1007/978-3-030-40850-3_8.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

Andrei Nicolae. PLU: the piecewise linear unit activation function. *CoRR*, abs/1809.09534, 2018. URL http://arxiv.org/abs/1809.09534.

Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning, 2018.

German I. Parisi. Human action recognition and assessment via deep neural network self-organization, 2020.

Chinmay Rane, Kanishka Tyagi, Sanjeev Malalur, Yash Shinge, and Michael Manry. Optimal input gain: All you need to supercharge a feed-forward neural network, 2023.

Chinmay Appa Rane. Multilayer perceptron with adaptive activation function. *Masters Thesis*, 2016. URL https://rc.library.uta.edu/uta-ir/bitstream/handle/10106/25934/RANE-THESIS-2016.pdf.

H.H Rosenbrock. An automatic method for finding the greatest or least value of a function. *The Computer Journal*, pp. 175–184, 1960.

D. Rumelhart, Geoffrey E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Learning Internal Representations by Error Propagation*, 1986a.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986b.

Claude Sammut and Geoffrey I. Webb (eds.). *Mean Squared Error*, pp. 653–653. Springer US, Boston, MA, 2010. ISBN 978-0-387-30164-8. doi: 10.1007/978-0-387-30164-8_528. URL https://doi.org/10.1007/978-0-387-30164-8_528.

Adrian J. Shepherd. Second-order methods for neural networks - fast and reliable training methods for multi-layer perceptrons. In *Perspectives in neural computing*, 1997.

Jonathan R Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, Carnegie Mellon University, USA, 1994.

Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. *CoRR*, abs/1506.04214, 2015. URL http://arxiv.org/abs/1506.04214.

P.Y. Simard, D. Steinkraus, and J.C. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.*, pp. 958–963, 2003. doi: 10.1109/ICDAR.2003.1227801.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. URL http://arxiv.org/abs/1409.1556.

T Tieleman and G Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning, 4(2)*, pp. 26–31, 2012.

Kanishka Tyagi. *Automated multistep classifier sizing and training for deep learners*. PhD thesis, Department of Electrical Engineering, The University of Texas at Arlington, Arlington, TX, 2018.

Kanishka Tyagi and Michael Manry. Multi-step training of a generalized linear classifier. *Neural Processing Letters*, 50:1341–1360, 2019.

Kanishka Tyagi, Nojun Kwak, and Michael T Manry. Optimal conjugate gradient algorithm for generalization of linear discriminant analysis based on l1 norm. In *ICPRAM*, pp. 207–212, 2014.

Kanishka Tyagi, Son Nguyen, Rohit Rawat, and Michael Manry. Second order training and sizing for the multilayer perceptron. *Neural Processing Letters*, 51(1):963–991, 2020.

Kanishka Tyagi, Chinmay Rane, Bito Irie, and Michael Manry. Multistage newton's approach for training radial basis function neural networks. *SN Computer Science*, 2(5):1–22, 2021.

Kanishka Tyagi, Chinmay Rane, and Michael Manry. Supervised learning. In *Artificial Intelligence and Machine Learning for EDGE Computing*, pp. 3–22. Elsevier, 2022.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL http://arxiv.org/abs/1706.03762.

Harry Wechsler, P. Jonathon Phillips, Vicki Bruce, Françoise Fogelman Soulié, and Thomas S. Huang. *Face Detection and Recognition*, pp. 174–185. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998. ISBN 978-3-642-72201-1. doi: 10.1007/978-3-642-72201-1_9. URL https://doi.org/10.1007/978-3-642-72201-1_9.

Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013. URL http://arxiv.org/abs/1311.2901.

Wei Zhang. Shift-invariant pattern recognition neural network and its optical architecture. *Proceedings of Annual Conference of the Japan Society of Applied Physics*, 1988. URL https://drive.google.com/file/d/1nN_5odSG_QVae54EsQN_qSz-0ZsX6wA0/view.

Wei Zhang, Kouichi Itoh, Jun Tanida, and Yoshiki Ichioka. Parallel distributed processing model with local space-invariant interconnections and its optical architecture. *Applied optics*, 29 32:4790–7, 1990.

Zhilu Zhang and Mert R. Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, pp. 8792–8802, Red Hook, NY, USA, 2018. Curran Associates Inc.

A Appendix: Training weights by orthogonal least squares

OLS is used to solve for the output weights, pruning of hidden units Tyagi et al. (2020), input units Tyagi & Manry (2019) and deciding on the number of hidden units in a deep learner Tyagi (2018). OLS is a transformation of the set of basis vectors into a set of orthogonal basis vectors thereby measuring the individual contribution to the desired output energy from each basis vector.

In an autoencoder, we are mapping from an (N+1) dimensional augmented input vector to it's reconstruction in the output layer. The output weight matrix $\mathbf{W_{oh}} \in \Re^{N \times N_h}$ and y_p in elements wise will be given as

$$y_p(i) = \sum_{n=1}^{N+1} w_{oh}(i,n) \cdot x_p(n) \tag{47}$$

To solve for the output weights by regression , we minimize the MSE as in equation 10. In order to achieve a superior numerical computation, we define the elements of auto correlation $\mathbf{R} \in \Re^{N_h \times N_h}$ and cross correlation matrix $\mathbf{C} \in \Re^{N_h \times M}$ as follows :

$$r(n,l) = \frac{1}{N_v} \sum_{p=1}^{N_v} O_p(n) \cdot O_p(l) \qquad c(n,i) = \frac{1}{N_v} \sum_{p=1}^{N_v} O_p(n) \cdot t_p(i) \tag{48}$$

Substituting the value of $y_p(i)$ in equation 10 we get,

$$E = \frac{1}{N_v} \sum_{p=1}^{N_v} \sum_{m=1}^{M} [t_p(m) - \sum_{k=1}^{N_h} w_o h(i,k) \cdot O_p(k)]^2 \tag{49}$$

Differentiating with respect to $\mathbf{W_{oh}}$ and using equation 48 we get

$$\frac{\partial E}{w_{oh}(m,l)} = -2[c(l,m) - \sum_{k=1}^{N_h+1} w_{oh}(m,k) r(k,l)] \tag{50}$$

Equating equation 50 to zero we obtain a M set of $N_h + 1$ linear equations in $N_h + 1$ variables. In a compact form it can be written as

$$\mathbf{R} \cdot \mathbf{W}^T = \mathbf{C} \tag{51}$$

By using orthogonal least square, the solution for computation of weights in equation 51 will speed up. For convineance, let $N_u = N_h + 1$ and the basis functions be the hidden units output $\mathbf{O} \in \Re^{(N_h+1)\times 1}$ augmented with a bias of $\mathbf{1}$. For an unordered basis function $\mathbf{O}$ of dimension N_u , the m^{th} orthonormal basis function $\mathbf{O}^{'}$ is defines as « add reference »

$$O_m^{'} = \sum_{k=1}^{m} a_{mk} \cdot O_k \tag{52}$$

Here a_{mk} are the elements of triangular matrix $\mathbf{A} \in \Re^{N_u \times N_u}$

For $m = 1$

$$O_1^{'} = a_{11} \cdot O_1 \quad a_{11} = \frac{1}{\|O\|} = \frac{1}{r(1,1)} \tag{53}$$

for $2 \leq m \leq N_u$, we first obtain

$$c_i = \sum_{q=1}^{i} a_{iq} \cdot r(q, m) \tag{54}$$

for $1 \leq i \leq m-1$. Second, we set $b_m = 1$ and get

$$b_{jk} = -\sum_{i=k}^{m=1} c_i \cdot a_{ik} \tag{55}$$

for $1 \leq k \leq m-1$. Lastly we get the coeffeicent A_{mk} for the triangular matrix $\mathbf{A}$ as

$$a_{mk} = \frac{b_k}{[r(m,m) - \sum_{i=1}^{m-1} c_i^2]^2} \tag{56}$$

Once we have the orthonormal basis functions, the linear mapping weights in the orthonormal system can be found as

$$w^{'}(i, m) = \sum_{k=1}^{m} a_{mk} c(i, k) \tag{57}$$

The orthonormal system's weights $\mathbf{W}^{'}$ can be mapped back to the original system's weights $\mathbf{W}$ as

$$w(i, k) = \sum_{m=k}^{N_u} a_{mk} \cdot w_o^{'}(i, m) \tag{58}$$

In an orthonormal system, the total training error can be written from equation 10 as

$$E = \sum_{i=1}^{M} \sum_{p=1}^{N_v} [\langle t_p(i), t_p(i) \rangle - \sum_{k=1}^{N_u} (w^{'}(i, k))^2] \tag{59}$$

Orthogonal least square is equivalent of using the **QR** decomposition Golub & Van Loan (2012) and is useful when equation equation 51 is ill-conditioned meaning that the determinant of $\mathbf{R}$ is $\mathbf{0}$.