CS698 Final Project Report
Adaptive Bidirectional Backpropagation

Si Ao Chen
University of Waterloo
sachen@uwaterloo.ca

Xinyu Liu
University of Waterloo
x389liu@edu.uwaterloo.ca

***Abstract*—The backpropagation algorithm is currently the most popular algorithm used in developing deep neural networks. However, the backpropagation phase of this method is not possible to operate in an actual neural system. Hence, this report presents two new implementations of error backpropagation that is biologically plausible. In particular, the new implementations consist of bidirectional training with two weight matrices, one for each direction. Several types of datasets will be used to to conduct a thorough performance comparison between the bidirectional methods and the traditional methods.**

I. Introduction

In deep neural networks (DNN), backpropagation (BP) method is considered to be one of the most important algorithms to build a successful model. However, it has been suggested that in a biological neural system, it is not possible to build a backward channel that is connected symmetrically to the corresponding feedforward weight. Furthermore, there is no solid evidence indicating that errors can be backpropagated in the brain. As a result, it will prevent the backward algorithm to obtain the value of the feedforward weight, which is crucial in calculating the weight gradients.

To adjust the connection weight without any prior information of the feedforward phase, two models are introduced: the feedback-alignment (FA) model and the direct feedback-alignment (DFA) model. The basic idea about the FA model is that it removes the assumption of knowing the connection weight and propagate some random feedback from the output layer to the hidden layers in order to update the weight gradients. The difference between DFA and FA is that in DFA, the feedback error signal can transmit directly from the output layer to any of the hidden layers. Figure 1 shows the main idea of BP, FA and DFA, where the black arrows indicate the feed-forward phase and the red arrows indicate the feedback phase.

However, since FA and DFA models send random and fixed feedback, their performance is not optimized. Hence, the authors of [1] proposed two high performance bidirectional

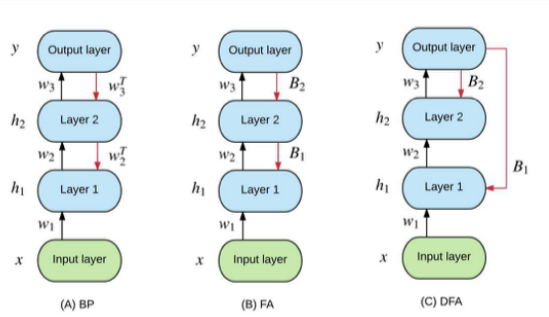

Fig. 1. General workflows of backpropagation(BP), feedback-alignment(FA) and direct feedback-alignment(DFA) methods, obtained from [1]

neural network models that discard the assumption that the feedback weights are random and fixed. In particular, each of these two models consists of two weight matrices: the feed-forward and the feedback weight matrices, where the former weight matrix is tuned during the forward learning phase, and the latter weight matrix is adjusted in the backward learning phase. The ideas behind these two models are derived from the concept of a biologically plausible perceptron (BioPP) model, which is a more logical NN model in the neuro-science perspective than the BP method.

II. Background

The BioPP model is built based on the concept of long-term potentiation (LTP), which essentially describes the phenomenon of the formation of strong connection between two neurons due to increasing number of receptors at the dendrites in order to encourage more ions capture. Therefore, LTP is an important factor to learning and memory formation.

The BioPP model consists of three parts:

- Signals: BioPP consists of two groups of signals, which are feed-forward signals used in the forward phase for inference purposes, and feedback error signals used in the backward phase for synaptic weights adjustments.

- Weights: quantity of the signal that a neuron desires to receive based on the incoming error signals, after which it will send error signals to other neurons.
- Activations and Biases: same definition as in the conventional BP algorithm.

The BioPP model is shown in Figure 2 below, where the curve and the square indicate respectively the synaptic weights and the receptors. The blue and the red colours indicate the forward phase and the backward phase, respectively.

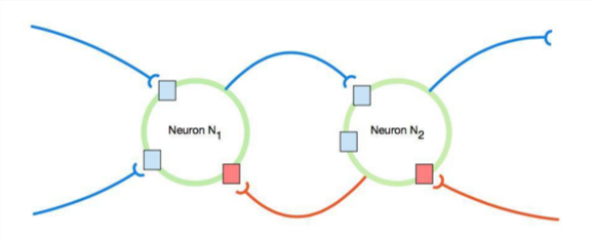

Fig. 2. An example of a simple BioPP model, obtained from [1]

However, there are a few restrictions to the BioPP model:

- Due to the fact that axons transmit neural signals in one direction, it is not possible to obtain error signals using incoming weights.
- Weights adjustment in a neuron is achieved locally based on various error signals.
- It should be expected that any incoming weight is adaptive.

In FA and in DFA, since some of their weight matrices are fixed, these two models do not satisfy some of the restrictions of the BioPP model. Also, because of the nature of their weight matrices, FA and DFA may not be able to produce a desired output. Hence, the author of [1] introduces Bidirectional-FA (BFA) and Bidirectional-DFA (BDFA) algorithms that are bidirectional learning models derived based on the BioPP model.

BFA and BDFA are similar to FA and DFA, but their forward and feedback weights are trainable through a specific pipeline. In particular, during the training process, the error signals in BFA are received layer by layer, while signals in BDFA are transmitted directly from the output layer. In theory, adaptive weights generated in bidirectional models are expected to outperform random fixed weights used in FA and DFA. In this project, the two bidirectional models (i.e. BFA and BDFA) will be implemented and trained using datasets of various types such as image classification, regression and adversarial inputs to evaluate their performance. Figure 3 and Figure 4 visualize the whole pipeline of the two new algorithms, where the black arrows indicate the feed-forward phase and the red arrows indicate the feedback phase.

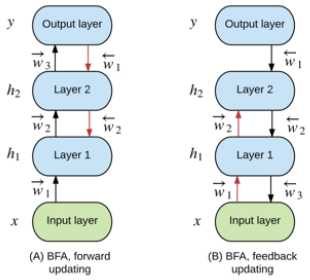

Fig. 3. Pipeline for BFA algorithm, obtained from [1]

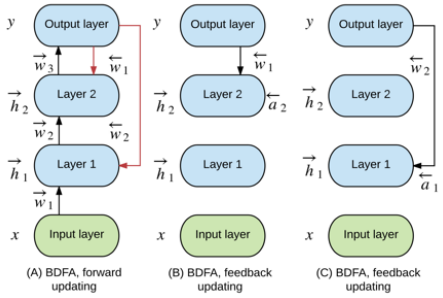

Fig. 4. Pipeline for BDFA algorithm, obtained from [1]

III. Implementations

In this paper, the DNN we are constructing contains up to three hidden layers. In order to conduct performance comparison, four algorithms will be built: FA, DFA, BFA and BDFA. The implementation is built on the backpropagation algorithm that was learned from [2]. More specifically, the code that is used to extend is from assignment 3 and assignment 5, where we update the Network class and the BNetwork subclass. Thus, we have two files in total:

- **Network.py**: store the Network class and all the helper functions such as activation functions and cost functions.
- **BNetwork.ipynb**: store the new BNetwork subclass and functions generating datasets to perform various testing on the new models.

Also, it is worth mentioning that the bias will set to 0 in the new architectures as it is not a crucial factor in the performance comparison between the two versions of FA and DFA. Also, the explanation in this section follows closely the notations learned and practiced in [2], and all the models discussed in this section have two hidden layers.

A. Feedback Alignment and Direct Feedback Alignment

In a typical FA model, its feedforward phase is identical to the BP method. However, its hidden layers are updated in the following way:

$$\frac{\delta E}{\delta z_y} = \hat{y} - y = e \tag{1}$$

$$\frac{\delta E}{\delta z_2} = (B_2 e) \odot \sigma'(z_2) \tag{2}$$

$$\frac{\delta E}{\delta z_1} = (B_1 \frac{\delta E}{\delta z_2}) \odot \sigma'(z_1) \tag{3}$$

where B is the weight matrix of FA. The only difference between FA and DFA is the calculation of $\frac{\delta E}{\delta z_1}$, because in DFA it is derived as follows:

$$\frac{\delta E}{\delta z_1} = (B_1 e) \odot \sigma'(z_1) \tag{4}$$

The weight gradient update for BP, FA and DFA are identical and follows what we learned in [2].

In BNetwork.ipynb, a new argument $mode$ is added in the $BackProp$ function to act as a switch between different calculations of $\frac{\delta E}{\delta z}$'s using different networks, and its default value is set to BP, i.e. the traditional backpropagation method. Therefore, when users do not specify $mode$, the traditional backpropagation method is implemented. When $mode$ is FA, a new weight matrix is generated to replace the weight matrix in the formula of the backpropagation method. If $mode$ is DFA, then the same code in FA is used, except that the gradients used in the formulas are fixed to the top gradient $\frac{\delta l}{\delta z_y}$, where l is the cost function, and z_y is the output layer.

B. Bidirectional Feedback Alignment and Bidirectional Direct Feedback Alignment

For BFA, cross-entropy loss function is used in the feedforward direction, and MSE is generally used as the loss function in the feedback phase. Note that variables with $\overrightarrow{\cdot}$ are components in the feedforward phase, whereas variables with $\overleftarrow{\cdot}$ are components in the feedback phase. Also, the learning rate is ignore in the mathematical derivations for simplifying the formulas. Figure 3 is very useful to visualize this procedure. The hidden layers' input gradients are calculated as follows:

$$\frac{\delta E}{\delta z_y} = \frac{\delta \overrightarrow{l}}{\delta z_y} = \hat{y} - y = \overrightarrow{e} \tag{5}$$

$$\frac{\delta E}{\delta \overrightarrow{z_2}} = \frac{\delta \overrightarrow{l}}{\delta \overrightarrow{z_2}} = (\overleftarrow{w_1}\,\overrightarrow{e}) \odot \sigma'(\overrightarrow{z_2}) \tag{6}$$

$$\frac{\delta E}{\delta \overrightarrow{z_1}} = \frac{\delta \overrightarrow{l}}{\delta \overrightarrow{z_1}} = (\overleftarrow{w_2} \frac{\delta E}{\delta \overrightarrow{z_2}}) \odot \sigma'(\overrightarrow{z_1}) \tag{7}$$

It is worth mentioning that in the equations above, the weights used to update the feedforward z's are feedback weights. The weight gradients are calculated in the same manner as in the BP model we learned in [2]. In the feedback phase, the layers' input and gradients are calculated as follows:

$$\overleftarrow{z_1} = \overleftarrow{w_1} y, \quad \overleftarrow{h_1} = \sigma(\overleftarrow{z_1}) \tag{8}$$

$$\overleftarrow{z_2} = \overleftarrow{w_2}\overleftarrow{h_1}, \quad \overleftarrow{h_2} = \sigma(\overleftarrow{z_2}) \tag{9}$$

$$\overleftarrow{z_x} = \overleftarrow{w_3}\overleftarrow{h_2}, \quad \hat{x} = \sigma_x(z_x) \tag{10}$$

$$\frac{\delta E}{\delta z_x} = \hat{x} - x = \overleftarrow{e} \tag{11}$$

$$\frac{\delta E}{\delta \overleftarrow{z_2}} = \frac{\delta \overleftarrow{l}}{\delta \overleftarrow{z_2}} = (\overrightarrow{w_1}\overleftarrow{e}) \odot \sigma'(\overleftarrow{z_2}) \tag{12}$$

$$\frac{\delta E}{\delta \overleftarrow{z_1}} = \frac{\delta \overleftarrow{l}}{\delta \overleftarrow{z_1}} = (\overrightarrow{w_2} \frac{\delta E}{\delta \overleftarrow{z_2}}) \odot \sigma'(\overleftarrow{z_1}) \tag{13}$$

$$\frac{\delta E}{\delta \overleftarrow{w_1}} = -\frac{\delta E}{\delta \overleftarrow{z_1}} y^T, \quad \frac{\delta E}{\delta \overleftarrow{w_2}} = -\frac{\delta E}{\delta \overleftarrow{z_2}} \overleftarrow{h_1}^T, \quad \frac{\delta E}{\delta \overleftarrow{w_3}} = -\overleftarrow{e}\,\overleftarrow{h_2}^T \tag{14}$$

BDFA model has the same feedforward path, but it has a different gradients update formula compared to the other models discussed in this report. Figure 4 is very useful to visualize this procedure. The BDFA model has the following gradients update and feedback rules.

$$\frac{\delta E}{\delta z_y} = \frac{\delta \overrightarrow{l}}{\delta z_y} = \hat{y} - y = \overrightarrow{e} \tag{15}$$

$$\frac{\delta E}{\delta \overrightarrow{z_2}} = \frac{\delta \overrightarrow{l}}{\delta \overrightarrow{z_2}} = (\overleftarrow{w_1}\,\overrightarrow{e}) \odot \sigma'(\overrightarrow{z_2}) \tag{16}$$

$$\frac{\delta E}{\delta \overrightarrow{z_1}} = \frac{\delta \overrightarrow{l}}{\delta \overrightarrow{z_1}} = (\overleftarrow{w_2}\,\overrightarrow{e}) \odot \sigma'(\overrightarrow{z_1}) \tag{17}$$

$$\frac{\delta E}{\delta \overrightarrow{w_1}} = -\frac{\delta E}{\delta \overrightarrow{z_1}} x^T, \quad \frac{\delta E}{\delta \overrightarrow{w_2}} = -\frac{\delta E}{\delta \overrightarrow{z_2}} \overrightarrow{h_1}^T, \quad \frac{\delta E}{\delta \overrightarrow{w_3}} = -\overrightarrow{e}\,\overrightarrow{h_2}^T \tag{18}$$

The calculation of the input to the hidden layers in the feedback phase is as follows:

$$\overleftarrow{a_1} = \overleftarrow{w_2} y, \quad \overleftarrow{a_2} = \overleftarrow{w_1} y \tag{19}$$

The feedback weights gradients is:

$$\frac{\delta E}{\delta \overleftarrow{w_1}} = -\overleftarrow{l_1}\, y^T, \quad \frac{\delta E}{\delta \overrightarrow{w_2}} = -\overleftarrow{l_2}\, y^T \tag{20}$$

where

$$\overleftarrow{l_1} = 1 - \sigma(\overleftarrow{z_1} \cdot \overrightarrow{h_1}), \quad \overleftarrow{l_2} = 1 - \sigma(\overleftarrow{z_2} \cdot \overrightarrow{h_2}) \tag{21}$$

In order to build BFA and BDFA, we have to add $W_{feedback}$ and W_{BDFA} weight matrices in the Network class. During the network construction, every time a new layer is added, the traditional W and $W_{feedback}$ will both be updated in the network in the same way, i.e. assign a weight matrix with random value based on the Normal distribution, except that the dimensions of the added weight matrices in W and in $W_{feedback}$ are reversed because $W_{feedback}$ is used in the opposite direction as W. On the other hand, the weight matrix

W_{BDFA} stores the weights of the feedback phase for BDFA. When a layer is added to the network, W_{BDFA} is wiped out and the weight matrix between each layer is recalculated since the output layer is changed when adding a new layer, and each component in W_{BDFA} is constructed corresponding to the output layer.

Similarly, two new important fields for feedback phase are added to Layer class, namely $z_{feedback}$ and $h_{feedback}$. These two fields are used during the feedback phase to store the layer's input values and output values, respectively.

In the function *TopGradient*, the argument *mode* (default is *None*) is added for calculating the cost function for BFA in the feedback phase, where it uses MSE cost function and the input layer to calculate the top gradient.

In addition to the changes in BNetwork.ipynb for implementation of FA and DFA, several new functions and updates to the old functions are introduced to incorporate the feedback phase smoothly with the traditional methods. For example, in the *BackProp* function, the argument *mode* has two more possible values: *BFA* and *BDFA*. The sole difference between modes *BP* and *BFA* is that in the calculation of the new $\frac{\delta E}{\delta z}$, $W_{feedback}$ is used in *BFA* instead of W, whereas in *BDFA*, top gradient and W_{BDFA} are used instead of previous $\frac{\delta E}{\delta z}$ and W in *BP*.

A new function *FeedBack* is added to conduct the forward process in the feedback phase. It has inputs *y* and *mode*, where *y* is treated as the input layer. If *mode* is *BFA*, then similar idea as the traditional feedforward function is applied using $z_{feedback}$, $h_{feedback}$ and $W_{feedback}$ of each layer, however the order of update is from the output layer to the input layer. When the mode is *BDFA*, $z_{feedback}$ is updated using y and W_{BDFA}.

Another new function called *BackPropFeedBack* is created to perform the error propagation process of the feedback phase. It has inputs *x*, *y*, *lrate* and *mode*. The goal of this function is to update the feedback weight matrix, and the top gradient is calculated using MSE cost function and the input layer. When *mode* is *BFA*, each $\frac{\delta E}{\delta w_{feedback}}$ is the product of $h_{feedback}$ and $\frac{\delta E}{\delta z_{feedback}}$. Then, W is used to update $\frac{\delta E}{\delta z_{feedback}}$ using the same logic as in the traditional BackProp function. If the mode is *BDFA*, the changes we need to made are in the $\frac{\delta E}{\delta z}$ update formula of the Backprop function, where the top gradient is used to update each $\frac{\delta E}{\delta z_{feedback}}$ multiplied with the appropriate W_{BDFA}.

Finally, during each epoch in SGD, after FeedForward and BackProp functions are called, FeedBack and BackPropFeedBack are called in this order to complete the bidirectional error propagation.

IV. Experiments

In this section, we compare the performance of BFA and BDFA to that of FA and DFA in various ways.

A. MNIST dataset

All networks discussed above are trained using 50,000 training samples from MINIST dataset, and there are 10,000 testing samples to evaluate their classification accuracy. Each network is trained 100 epochs with learning rate 0.01 and batch size 128. The activation functions for the output layer and hidden layers are Softmax and Tanh, respectively. Furthermore, we use Categorical Cross-Entropy as our loss function.

Model	BP	FA	DFA	BFA	BDFA
1x400	3.40	7.45	7.55	3.94	2.56
2x400	3.47	8.17	8.03	4.34	2.72
3x400	5.05	11.26	11.4	4.87	2.61

TABLE I
TEST ERROR RATE (%) FOR BP, FA, DFA, BFA AND BDFA ON MNIST. 1 X 400 MEANS THERE IS 1 HIDDEN LAYER CONTAINING 400 NEURONS.

Model	BP	FA	DFA	BFA	BDFA
1x400	284.49	291.93	324.15	530.62	330.34
2x400	463.07	661.62	450.57	833.42	489.48
3x400	620.03	1065.35	589.24	1134.25	481.31

TABLE II
TRAINING TIME (S) FOR BP, FA, DFA, BFA AND BDFA ON MNIST.

The classification performance on MINIST dataset are shown in TABLE I. The BDFA model gives us the most accurate results on MNIST. The error rates of FA and DFA are very close and are significantly higher than the error rates of BFA and BDFA. This is an evidence that trainable feedback weights would be preferred to random feedback weights during the training process.

However, there is a time trade off for high accuracy. As shown in TABLE II, BFA and BDFA are generally more time consuming than FA and DFA, which is because training weights typically takes more time than generating random weight matrices.

B. Flipped BFA and BDFA

In traditional BFA and BDFA, FeedForward and BackProp functions are called before FeedBack and BackPropFeedBack. We would like to determine if the classification accuracy of bidirectional models would be affected if functions are called in different orders. Two new models called "Flipped BFA" and "Flipped BDFA" are created, which always feed the data in before backward propagation. In other words, the order of functions calls in our new models becomes FeedForward, FeedBack, BackProp and BackPropFeedBack.

We run the experiments using MNIST dataset with the exactly same settings as in the last section, and the results are shown in TABLE III and TABLE IV. We can observe that "Flipped" models have slightly lower accuracy but longer running time compared to traditional BFA and BDFA models. Hence, finding an appropriate order of function calls during the training phase is important to achieve a good classification performance.

Model	BFA	Flipped BFA	BDFA	Flipped BDFA
1x400	3.94	4.2	2.56	2.56
2x400	4.34	4.43	2.72	2.97
3x400	4.87	4.92	2.61	3.05

TABLE III
TEST ERROR RATE (%) FOR BFA, FLIPPED BFA, BDFA AND FLIPPED BDFA ON MNIST.

Model	BFA	Flipped BFA	BDFA	Flipped BDFA
1x400	530.62	545.83	330.34	293.32
2x400	833.42	1023.66	489.48	556.95
3x400	1134.25	1425.34	481.31	757.89

TABLE IV
TRAINING TIME (S) FOR BFA, FLIPPED BFA, BDFA AND FLIPPED BDFA ON MNIST.

C. Regression

Besides classification, we would like to apply our BFA and BDFA models on other types of problems. For instance, we would like to train them on regression-type dataset on both 2 dimensions and 3 dimensions.

1) 2D Regression: Using the code from Assignment 3 of [2], 5 training samples and 300 test samples are generated randomly from

$$y = ax^2 + mx + b \tag{22}$$

where a, m and b are real number constants. Our models are aimed to find the best fitting curve for the given dataset. Each network has 1 hidden layer with 10 neurons, and it is trained 5000 epochs with learning rate 0.8 and batch size 10. Arctan and Identity activation functions are used at hidden layer and output layer, respectively, and MSE is used as loss function.

Regression models learned by different types of algorithms are displayed in Figure 5, 6 and 7. All networks except DFA are able to fit data points with reasonable curves. In fact, the DFA algorithm transmits loss information directly from output layer to each hidden layer instead of learning the mapping layer by layer, which might result in a weird fitted regression model as shown above. Although the BDFA network transmits loss information to hidden layers in the same way as DFA,

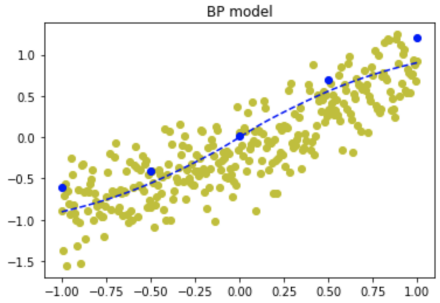

Fig. 5. 2D Regression model learned by BP algorithm

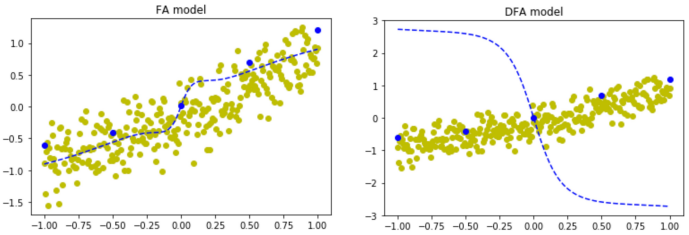

Fig. 6. 2D Regression model learned by FA and DFA algorithm

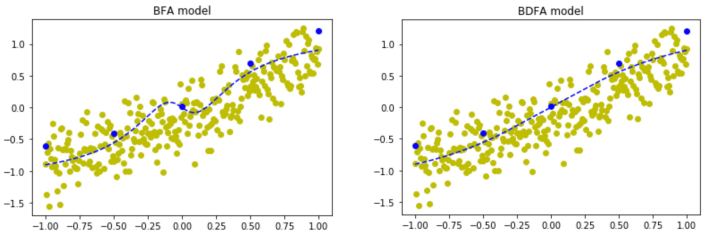

Fig. 7. 2D Regression model learned by BFA and BDFA algorithm

its learned regression model fits training points very well because the BDFA algorithm uses trainable feedback weights to deliver error signals instead of fixed weight matrices. Therefore, bidirectional learning models could be used to solve regression problems. In particular, for the DFA model, it is recommended to use the bidirectional version to improve the fitting performance.

2) 3D Regression: Since most of our algorithms work very well in fitting 2 dimensional data, we are interested in evaluating their performance in a higher dimensional world. Therefore, 20 x 20 training samples and 300 x 300 test samples are generated from the 3 dimensional expression below

$$z = ax^2 + my + b \tag{23}$$

where a, m and b are real number constants. Similar to the previous section, the goal of this experiment is to find the best fit surface for a given set of data points. Each network has

2 hidden layers with 20 neurons, and it is trained for 5000 epochs with learning rate 1 and batch size 20. Arctan and Identity activation functions are used at hidden layers and the output layer, respectively, and MSE is used as loss function.

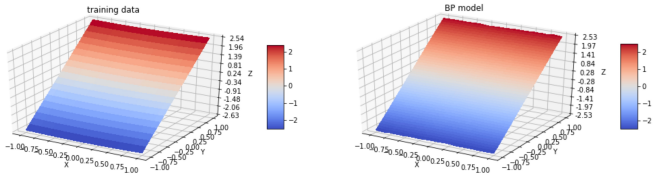

Fig. 8. 3D Regression training data and model learned by BP algorithm

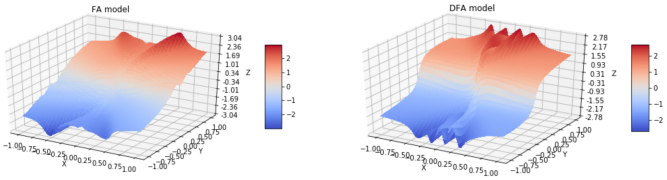

Fig. 9. 3D Regression model learned by FA and DFA algorithm

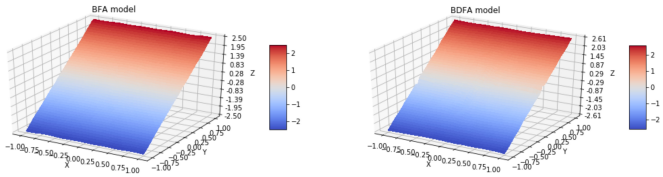

Fig. 10. 3D Regression model learned by BFA and BDFA algorithm

Three-dimensional surfaces created by different networks are displayed in Figure 8, 9 and 10. From Figure 8 and Figure 10, we can see that the surface generated by BP, BFA and BDFA are very flat and their shapes are similar to the plane generated using the training data. However, the surfaces created with FA and DFA algorithms have much higher curvature than others, which is very different from the plane generated by the training data. Hence, bidirectional learning algorithms outperform the FA and DFA on solving high dimensional regression problems.

D. Adversarial input

Based on the experiments conducted above, one could think that the bidirectional models are supposed to have better weight matrices. Therefore, it would be interesting to test its robustness against targeted and untargeted adversarial inputs. In particular, since the adversarial inputs are tailored specifically to each NN model, it would be impossible to compare the results by feeding one adversarial input to all the models. Hence, the main criteria to test the robustness is to compare the computation time of each model to successfully generate their corresponding adversarial input. Intuitively, a more robust model against adversarial attack would consume more time to generate an adversarial input.

Moreover, one might think that the time consumed to generate an adversarial input for BDFA and BFA can depend on the number of hidden layers in each model and the number of epochs used to train the models. Using the code from Assignment 5 of [2], we can successfully create adversarial inputs for different models. The table below showcases the average computation time over 100 trials of generating an adversarial input using either FA, DFA, BFA or BDFA, where the default number of hidden layer is one:

Model description	Untargeted time (sec)	Targeted time (sec)
BP, pretrained	0.0246	0.0131
FA, epoch = 50	0.0674	0.0183
FA, epoch = 50 hidden = 2	0.0548	0.0403
FA, epoch = 100	0.1186	0.0710
DFA, epoch = 50	0.0801	0.0455
DFA, epoch = 50 hidden = 2	0.0336	0.0172
DFA, epoch = 100	0.0566	0.0223
BFA, epoch = 50	0.0798	0.1323
BFA, epoch = 50 hidden = 2	0.0438	0.0511
BFA, epoch = 100	0.0674	0.0691
BDFA, epoch = 50	0.1559	0.0835
BDFA, epoch = 50 hidden = 2	0.1008	0.0848
BDFA, epoch = 100	0.0779	0.0694

TABLE V
COMPUTATION TIME OF ADVERSARIAL INPUTS USING FA, DFA, BFA AND BDFA

As we can see in Table 5, there is no clear pattern to the differences in computation time between the bidirectional models and unidirectional models. Moreover, the number of epochs and the number of hidden layers does not play an important role in reducing the computation time. One possible explanation is that the structure of the NN models is still too simple to make a distinguishable difference between bidirectional models and unidirectional models.

V. FURTHER DISCUSSIONS

In this project, many experiments have been completed with interesting results. However, there are a few inexplicable observations that require more time and energy to understand it. First, the training and test accuracies obtained using the bidirectional models are both above 95%, but it is very difficult to perform a good feature mapping to the original digit images. In fact, the resulting image after completing the feedback

phase does not show any digit. One possible explanation is that it requires more iterations to the feedback phase until the resulting image reaches a stable state.

Futhermore, the experiments show that the order of conducting different stages has a direct influence to the test accuracy. In fact, when we change the order of backprogation of the feedforward phase and the feedback phase, the test error rates shown in Table 3 are all increased. However, the none of the steps in the feedback phase depends on the results from the execution of the backpropagation of the feedforward phase. Therefore, more studies are required to understand this phenomenon.

Also, more research is required to ameliorate the results from the adversarial input experiment. More specifically, the computation time of applying both untargeted and targeted adversarial input to the bidirectional and unidircetional models do not have a clear distinction. In addition, Adding more epochs and number of hidden layers does not help the robustness of the model against adversarial attack. One possible explanation is that the structure of the bidirectional models must be more complex in order to observe a consistent increase in the computation time of adversarial inputs.

VI. Conclusion

In this project, new bidirectional algorithms have been thouroughly discussed, namely the BFA and the BDFA models. Through various experiments, it shows clearly that bidirectional models outperform unidirectional models. To the best of the knowledge of the authors in [1], this research paper is the first study that demonstrates an increase in effectiveness of adaptive asymmetric backpropagation channels in BFA and BDFA compared to the conventional FA and DFA that have random and fixed backpropagation channels. Even though these two bidirectional models are educational guesses of how an actual neural system would work in human brains, but it shows that researchers are closer to understand the true logic of our brains.

References

[1] L.Hongyin, F.Jie and G.James, “Adaptive Bidirectional Backpropagation: Towards Biologically Plausible Error Signal Transmission in Neural Networks,” arXiv:1702.07097, April 2018.

[2] O.Jeff, ”CS698: Neural Network Assignments”, University of Waterloo, April 2019.