Exploration of Forward-Forward Network

Yanlin Li
University of Toronto
tristal.li@mail.utoronto.ca

Yuanhan Chen
University of Toronto
yuanhan.chen@mail.utoronto.ca

Abstract

This paper aims to investigate and demystify the Forward-Forward Algorithm in Hinton's (2022) recent paper. A novel procedure of initializing negative data is introduced, which includes random initialization within each epoch. This enables the algorithm to cover more negative cases within limited speed decrease. Logistic loss function with a new parameter called margin is proposed in this project, which improves the training result under same model configurations. Moreover, we clarified the method used in the algorithm and tested it under different model sizes, activation functions, and other related hyperparameters. [1]

1 Introduction

The idea of backpropagation (Rumelhart et al., 1986), a method of computing gradients based on chain rule, has fostered the success in deep learning during the last decade. Despite its popularity, the relationship between the working mechanism of human neurons and the learning process of neural network has attracted some recent attention.

Although Hinton (2022) has proposed a new idea of Forward-Forward Algorithm (FFA), the paper did not include a comprehensive evaluation of the algorithm under various model configurations. No detailed implementations or the hyperparameters used are provided, and some details of the training process are vague to the readers. This project dived deeper into Hinton's work and evaluated the performance of FFA with various configurations on Multilayer Perceptrons (MLP), including the loss function, the creation of negative data, unsupervised cases, model sizes and different architectures. A comprehensive explanation is provided with some modification in data initialization and loss function which improves the performance.

2 Related Works

Geoffery Hinton (2022) proposed a new training algorithm called Forward-Forward Algorithm (FFA) in his recent paper. Instead of back-propagating the loss function throughout the network, FFA uses two forward steps to learn each layer greedily. Although comparatively more computationally expansive than normal backpropagation, this algorithm mimics how human cortex process the signal. It does not require the program to store full knowledge of the forward pass, so that a black box of computation in forward pass can still work with FFA. (Hinton, 2022)

Our project extends the scope of Hinton's work in exploring different model configurations (model size, skip connection, activation functions, and threshold). We also proposed a modified data initialization step and a new loss function.

[1] Github:https://github.com/Tristal25/Forward-Forward-Experiments.git

Preprint. Under review.

3 Method

3.1 Model Architecture

In Hinton's paper (Hinton, 2022), two main architectures are used: Multilayer Perceptions (MLP), and local receptive fields. For simplicity, we only used MLP for experiments. Figure 1 represents the architecture we used. Each hidden layer contains a layer of MLP with selected activation function. Skip connections are optional. FFA was conducted inside each hidden layer, before the layer norm, so that backpropagation is avoided.

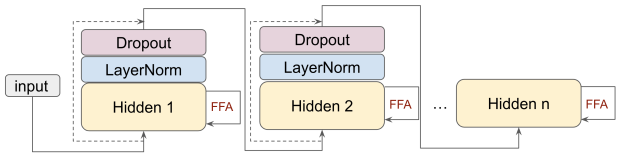


Figure 1: MLP architecture

3.2 Positive and Negative Data, Training Steps

FFA learns two forward steps in each layer, one on real data (positive data), and another on fake data (negative data). Goodness is evaluated in each layer, and the training goal is to increase the goodness for positive data and decrease the goodness for negative data. (Hinton, 2022) In practice, the initialization of negative data is of huge importance to the final model performance. We investigated three possible methods in negative data initialization: initiation before start, random initialization, and full coverage of negative cases. (See details of these methods and training steps in A.1)

3.2.1 Supervised Case

During supervised learning, labels are part of the training process in each layer, so we need to pass them as inputs. The solution proposed in Hinton's paper (2022) is to add the labels into the training data. In image classification, the first several pixels of the images are replaced by the one-hot encoder of the labels (correct for positive data and fake for negative data). A couple of examples can be found in Appendix A.1.

3.2.2 Unsupervised Case

In case of unsupervised learning, positive data is just the training data. Negative data is initialized using a mask to concatenate the selected sample with another random sample. A mask is a random bit image consisting of large regions of zeros and ones. We initialize a new random mask when we create each negative image in order to maximize the randomization. See A.2 for details of initialization steps of masks and the ways to generalize negative images.

The training method is also different in unsupervised case. Contrastive learning is used perform the training (Details about contrastive learning is in A.3). This trains an encoder for the input image.

3.3 Loss and Threshold

When training each layer, a loss function is used to evaluate the performance of classification. (See more about the loss function used in paper in A.4.1) We improved the loss by changing it into a logistic loss function, which in practice yields a better result. (See more details in A.4.2) An additional parameter, margin, is introduced to push the goodness (mean/sum value of activations) further to one side of threshold. This method is inspired by Support Vector Machine (Cortes et al., 1995) The loss function we are using sums up the positive and negative part, which enables the forward steps to be done in parallel and take advantage of the GPUs.

$$loss = \frac{1}{n}\sum_{i}\left(\log\left(1+\exp(-\frac{1}{m}\sum_{j}y_{ij}^{pos}-\theta+\epsilon)\right)+\log\left(1+\exp(\frac{1}{m}\sum_{j}y_{ij}^{neg}-\theta-\epsilon)\right)\right)$$

where m denotes the hidden size, n is the batch size, y_{ij}^{pos} is the output of j^{th} hidden unit and i^{th} positive sample before layer normalization, and y_{ij}^{neg} is the one for negative sample. θ is the threshold and ϵ is the margin. We choose to use mean to represent the goodness, because with lower range of values, it can be easier for us to find an optimal threshold θ.

3.4 Prediction and Evaluation

3.4.1 Supervised Case

In this step, all possible labels are overlaid on the test image. The goodness of the last layer are collected. The label with highest goodness is the predicted value. See full steps in A.5.

3.4.2 Unsupervised Case

After the training is done, a linear classifier is trained on the output encoded vectors and their targets (this step is supervised). Then we can pass training data or test data through the encoder learned by the unsupervised training and pass to linear classifier to predict the class of input image.

4 Experiments

Training Data and Batching: MNIST (Deng, 2012) is used for major experiments. MNIST is a large database of handwritten digits consisting of 60000 training images and 10000 testing images. The pictures are of black and white and of size 28×28. There are 10 labels in total. Our training and testing batch sizes are both 10000.

Experiment Design: We tested two three ways of initializing negative data (not including the one for all labels because of limitations of hardware capability), and evaluated the performance of out new loss function. The configurations of MLP below are then evaluated with the chosen data initialization methods and loss function. All the experiments are run in supervised case for consistency. Unsupervised learning is then run under the best configuration.

1. Model size: Number of layers, number of hidden units each layer
2. Activation functions: ReLU, Leaky ReLU, Tanh, Logistic, ELU, GELU
3. Skip connection
4. Threshold and margin

Hardware and Software Tools: We used PyTorch framework to implement the model. One Nvidia GeForce RTX 3080 graphic card was used to do the calculations.

Optimizer: We used Adam optimizer (Kingma et al., 2014) with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 1e-08$. The learning rate is $lr = 0.03$.

5 Discussions

For convenience, the data showing in the following sections uses the default configuration unless mentioned which hyper-parameter is changed. The default configuration: 4 layer MLP, hidden size 500, learning rate 0.03, 10000 batch size, 2000 epoch, ELU activation function, use layer norm and dropout=0.2 every layer, no skip-connection.

5.1 Analyzing ways of Initializing Negative Data and Different Loss Functions

Table 1: Ways of Initializing Negative Data and Different Loss Functions

	Random Neg, New Loss	Non-random Neg, New Loss	Random Neg, Original Loss
Test Accuracy	97.35%	95.76%	95.28%
Time Spent	320s	234s	249s

As is shown in Table 1, random negative data initialization can improve the accuracy under fixed model configuration. We changed the threshold and hidden size for original loss function in order to reach a good performance level. The test accuracy is also lower than our new method.

5.2 Model Configurations

5.2.1 Model Size

For more stable result, we chose to train 1000 epochs with threshold 10 in this part. (Chosen among 60-100 epochs, threshold 0.5 - 10.5, see more in A.6.1)

Table 2: Test Accuracy vs Model Size

Hidden sizes	100	200	300	400	500
Test Accuracy	95.35%	96.11%	96.50%	96.59%	96.90%

Although it looks like larger model has better accuracy on MNIST, after we run tests on hidden size 510, 525, 550, 600, 700, it's interesting to find that their accuracy are all <10%, 500 should be the best hidden size for our architecture on MNIST.

5.2.2 Activation Functions

The results below (Table 3, Default configuration) shows that ELU and ReLU are suitable in this case.

Table 3: Test Accuracy vs Activations

Activation	ELU	ReLU	Logistic	Tanh	Leaky ReLU	GELU
Test Accuracy	97.35%	97.08%	<10%	<10%	<10%	<10%

5.2.3 Skip Connection

Adding skip connection to the default configuration lowers the accuracy from 97.35% to 97.2%. And sometime causes the network to fail (acc <10%). So we will not include skip connection.

5.2.4 Threshold and Margin

We looped through thresholds from 0.5 to 10.5 with 0.5 increment and found that higher threshold needs more epoches to train otherwise the accuracy will be too low. However, once we have enough epoches (e.g. $\geq$1000) the performance will be much better than very small threshold like 0.5 whose accuracy is only about 90%. A margin of 1 can be useful in classifying some edge cases, which improves the accuracy from 97.35% to 97.38%.

6 Conclusion

We hypothesized that the changed method of negative data generalization and new loss function with margin can improve the performance of FFA. The best model (Full configuration & comparison: A.6.2) we trained using our default configuration with margin=1 achieved test accuracy of 97.38%, which out-performed the model without randomized negative data (95.76%), and the one with original loss function (95.28%). With all other configurations kept the same, the result we got can support our hypothesis. Our experiment also selected an optimal model size (4 layer, 500 hidden), and two suitable activation functions (ELU and ReLU).

The major limitation of this project is the poor performance of unsupervised case (78.58%) A.6.2. This may be the result of the current method of generating negative data. (See A.2.1) Also, we only included MLP in our analysis. Future tests using local receptive fields for image classification, data augmentation, and Recurrent Networks for top-down effects in MLP may improve the result. Moreover, it is computationally expansive to tune thresholds, so the idea of learnable thresholds worth exploration. Together with a better goodness function, the model may achieve a higher performance.

7 Reference

[1] Hinton, G. E. (2022). The Forward-Forward Algorithm: Some Preliminary Investigations. *arXiv preprint arXiv:2212.13345*

[2] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. Nature, 323:533–536.

[3] Hinton, G. and Sejnowski, T. (1986). Learning and relearning in Boltzmann machines. In Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations, pages 282–317*. MIT Press, Cambridge, MA.

[4] Cortes, C. and Vapnik, V. (1995). Support-vector networks. Machine Learning, 20(3), 273–297.

[5] Gutmann, M. and Hyvärinen, A. (2010). Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 297–304.

[6] Deng, L. (2012). The mnist database of handwritten digit images for machine learning research. IEEE Signal Processing Magazine, 29(6), 141–142.

[7] Kingma, D. P. and Ba, J. (2014) Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*

[8] Pezeshki, M., Rahman, H. and Yun, J. (2023), Implementation of forward-forward (FF) training algorithm - an alternative to back-propagation, pytorch_forward_forward, https://github.com/mohammadpz/pytorch_forward_forward.git

A Appendix

A.1 Initialize Negative data and Training Steps

The three possible methods of initializing negative data we could think of are:

1. Initialize before training. Fix a certain negative data for each training sample. This is the quickest possible solution. However, there are many possible false label or forms of false data for each training example. Without providing enough false information, the performance is limited.

2. (Only available for supervised learning) Initialize before training. Generate all possible false label for each training example. Each training step will consist of $len(unique(label))$ forward passes, instead of two. This method maximized the information passed in each training step, so the model can learn more from each training sample. However, this method is too slow at our first try and not applicable for unsupervised case. Limited by our hardware capacity, we choose not to include it in our experiment.

3. Initialize during training. Randomly initialize negative data in each epoch. This method balanced the training speed and the exploitation of training data.

In our experiment, we compared the first and the third methods.

The psudocode for the training step in our final model is:

```
for epoch in epochs:
    pos_data, neg_data = generate_data(x, y)
    for layer in layers:
        train_layer(layer, pos_data, neg_data)
        pos_data, neg_data = layer(pos_data, neg_data)
```

Here is an example in supervised cases (Figure 2):

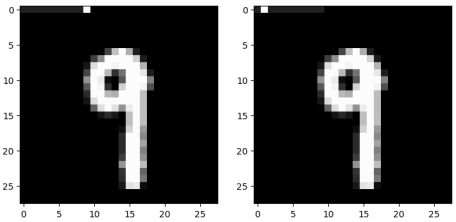


Figure 2: Image on the left is a positive data, the other is a corresponding negative data

A.2 Negative Data in Unsupervised Cases

The steps of initializing masks are:

1. Initialize same size of random data as each training example. Each data point is a random number between 0 and 1.
2. Use filter [1/4, 1/2, 1/4] to blur the image twice: once horizontally (using horizontal vector as filter), and once vertically (using vertical vector as filter). This can be implemented using convolution layer with padding 1 on the filter direction (padding = (0,1) for horizontal blur and padding = (1,0) for vertical blur).
3. Using a threshold of 0.5 to convert the mask into binary.

Then, two images are combined according to this figure (Figure 3) in Hinton's paper (2022).

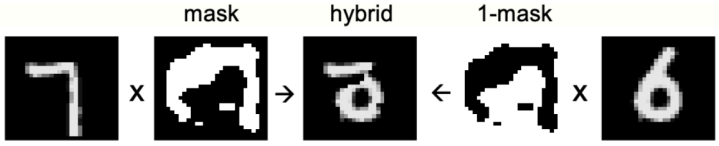


Figure 3: A hybrid image used as negative data in unsupervised case

A.2.1 Possible Edge Cases

This figure (Figure 4) shows a possible case, in which the negative data is approximately the same as the positive one.

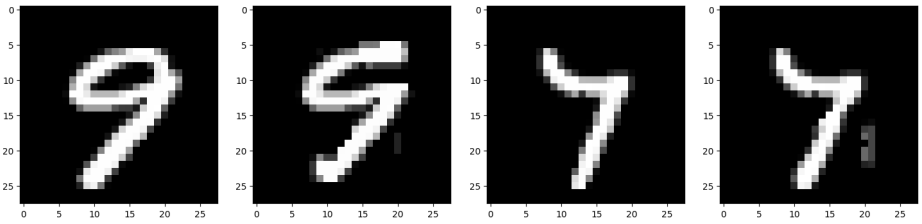


Figure 4: Mask produced negative data is approximately the same as the positive one

A.3 Contrastive Learning Technical Details

Contrastive learning is a type of unsupervised or self-supervised learning technique.The main idea is to encourage the model to recognize similarities and differences between instances, thus helping it to build a better understanding of the underlying structure in the data. In contrastive learning, a model is trained to identify similar and dissimilar pairs of data points. For each data point, a positive pair is created by generating an augmented version of the same data point, while negative pairs are formed by selecting other unrelated data points. The model learns by minimizing the distance between the positive pairs and maximizing the distance between the negative pairs in the feature space.

A.4 Goodness functions

A.4.1 Goodness in Original Paper

By the original definition of the goodness function, we expect high goodness for positive data and low goodness for negative data. In Hinton's paper (2022), the probability of a training data being positive data is used in training:

$$p(positive) = \sigma\left(\sum_j y_j^2 - \theta\right)$$

where y_j is the output of j^{th} hidden unit in the layer before layer normalization, and θ is a fixed threshold.

A.4.2 Logistic Loss Function

The logistic loss function is:

$$L(z_i) = \log(1 + \exp(-z_i))$$

This function grows linearly with negative values, so if the activation of positive data is far smaller than the threshold, the resulting loss will be high. This acts like a layer-wise loss function for classification, and enables our model to train greedily layer by layer.

A.5 Prediction Steps for Supervised Learning

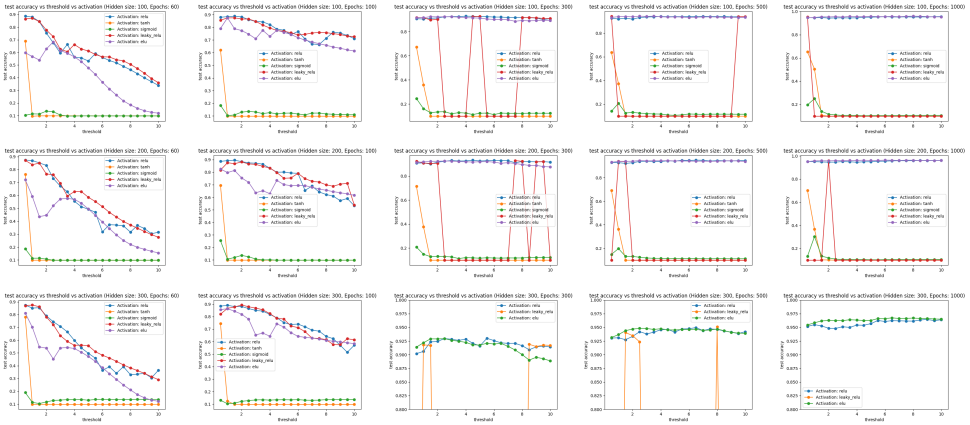
Here is the detailed steps of prediction:

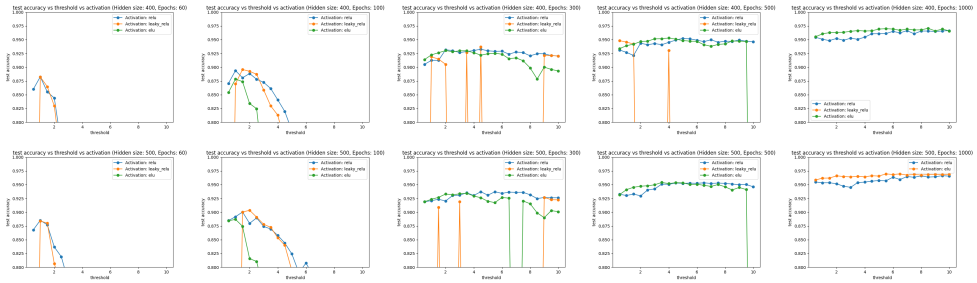
1. Add all possible labels to the test example (same procedure as the training steps), so there are $len(unique(label))$ test data in all for each test case.
2. Pass each test data (all labels) into the model and collect the last layer activations.
3. Compute $goodness = \frac{1}{m}\sum_j y_j^{pred}$ (y_j^{pred}'s are the last layer activations) for all labels and compute the maximum of all. The label with maximum goodness is the predicted label.

A.6 More Graphical Results

A.6.1 Model Tuning

Here are the graphs we used in choosing activation functions, thresholds, model sizes, and number of epochs to train.





A.6.2 Full Model Configurations and Final Comparison

Table 4: Performances and Architecture of Best Models

Source	Supervised	Accuracy	epochs	Layers	Hidden
Backpropagation	True	98.6%	20	2	784-800-10
Hinton's Paper	True	98.64%	60	4	784-2000*3
This project	True	97.36%	2000	4	784-500*3
Hinton's Paper	False	98.63%	100	4	784-2000*3
This project	False	78.58%	100	4	784-2000*3

Table 5: Full Reproducible Configurations of Best Models

Source	Supervised	epochs	Activation	Dropout	Threshold	Margin
Backpropagation	True	20	ReLU	False	N/A	N/A
Hinton's Paper	True	60	ReLU	False	unknown	N/A
This project	True	2000	ELU	0.2	10	1
Hinton's Paper	False	100	ReLU	False	unknown	N/A
This project	False	100	ReLU	False	0.5	0