Homework 2: Report

Anh Nguyen
nguye2aq@mail.uc.edu

Problem 1:

Problem Statement: A simple neuron model was trained on 400 images of 0s and 400 of 1s from the MNIST dataset, then tested on another 200 images of 0s and 1s. Input is a flattened image array of 784 pixels, put into a layer of 784 weights. The output is activated by Hebb's rule learning by reinforcement, with the theta value being iterated over a range from 0 to 40, with steps of 1. The model was trained with the following specifications:

System Specification:

- **num_epochs = 40**
- **η value = 0.01**

I tried smaller learning rate (0.005, 0.0025, …) and more epochs (100,200,400,...) and there was a decrease in the optimal F1 score

Results:

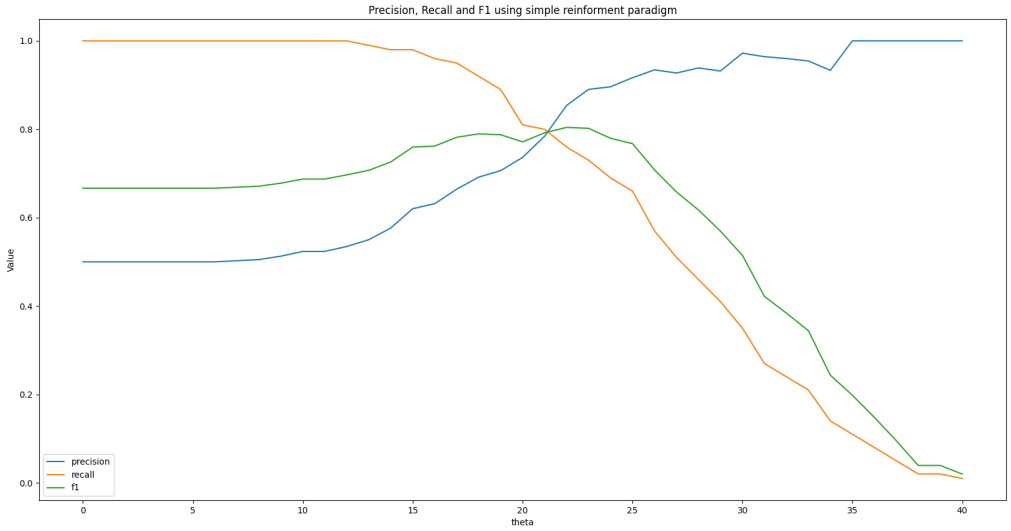

Figure 1a: *Evaluation statistics of model on the test set from iterating over threshold theta from 0 to 40*

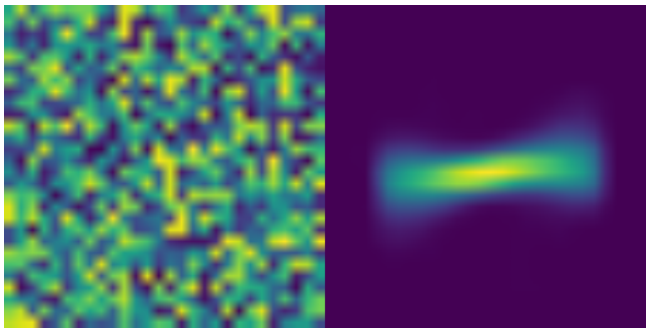

Figure 1b: *Weights of the model before and after training*

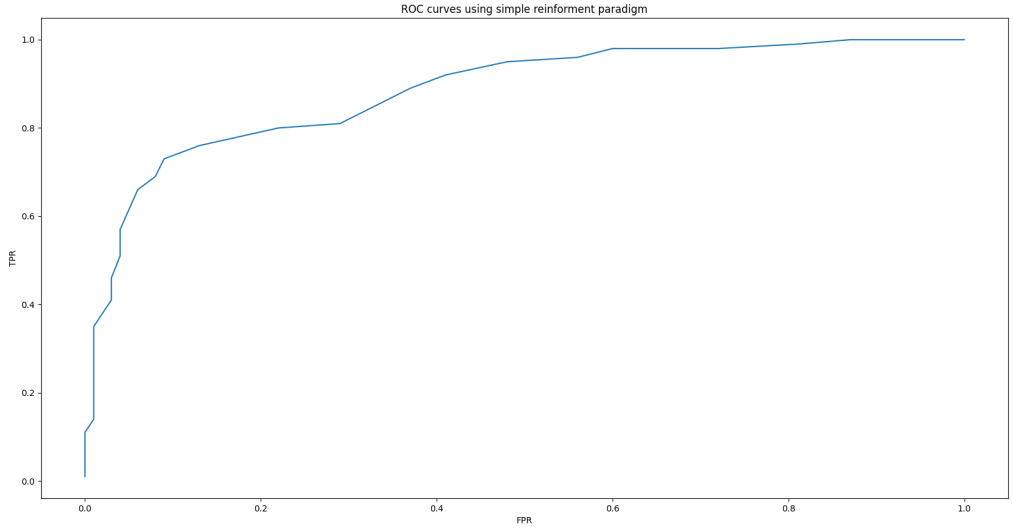

Figure 1c: *ROC curves evaluating the model's performance on the test set*

label	2	3	4	5	6	7	8	9
0	64	49	88	71	73	71	28	71
1	36	51	12	29	27	29	72	29

Figure 1d: *Inference output of model on images of 2 to 9*

Discussion: I found that the optimal threshold value is theta = 22, where TPR = 0.76, FPR = 0.11, and F1 score = 0.81, by finding where the sum of TPR and (1 - FPR) is maximum, and where the precision, recall and f1 graph lines intersect (Fig 1a). We also see the weights of the model were trained (Fig 1b), that the weight heat maps show a 1, that when multiplied with a 1 input (the trained inputs were transposed, so the weights image was transposed) will create an activation spike. For Fig 1d, we found that the model thinks images of 4, 5, 6, 9 are 0, as they have circles and/or arcs in the characters. The model thinks 8s and 3s are 1s, as it could look more slender and longer than others, like a 1. As for 7s being mistaken for 0s, instead of 1s, even though they have more similarities with the later, there are a lot of written 7s in the MNIST that have huge upper convex arcs, making them look more like 0s than 1s.

Problem 2:

Problem Statement: The same set up of 1-layer model and training config were built and run, except it uses perceptrons and updates weights using perceptron learning rules. The model was trained with the following specifications:

System Specification:

- **num_epochs = 400**
- **Learning rate = 0.005**

I started with lr = 0.01 and 40 epochs, and found that by decreasing lr to 0.005 and num_epochs to 400, it boosts f1 and others evaluation stats by 1-2%, very close to 100% on the test set. We do find that the training error fraction plateaued, so we have made the most out of the training data, there's no use for larger 400

Results:

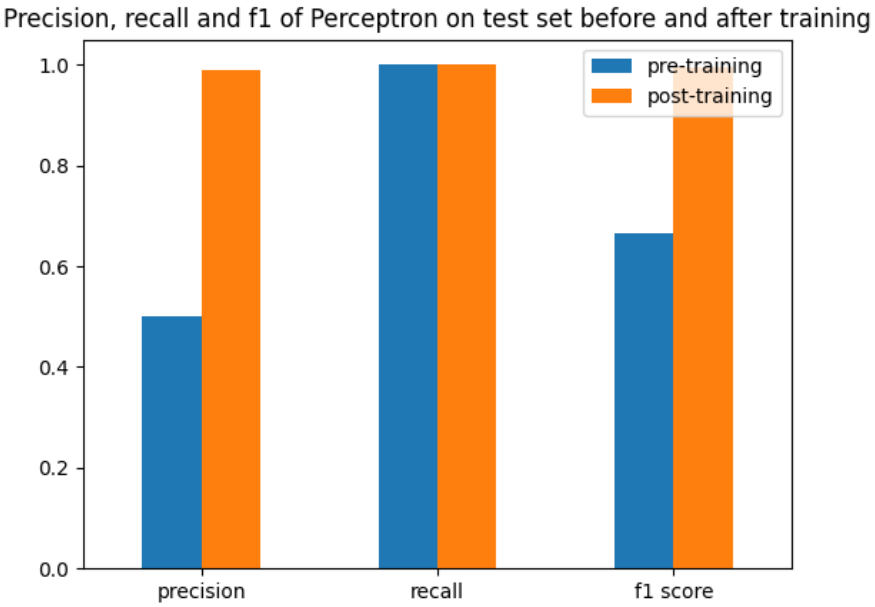

Figure 2a: *Evaluation statistics of model on the test set, compared between before and after training*

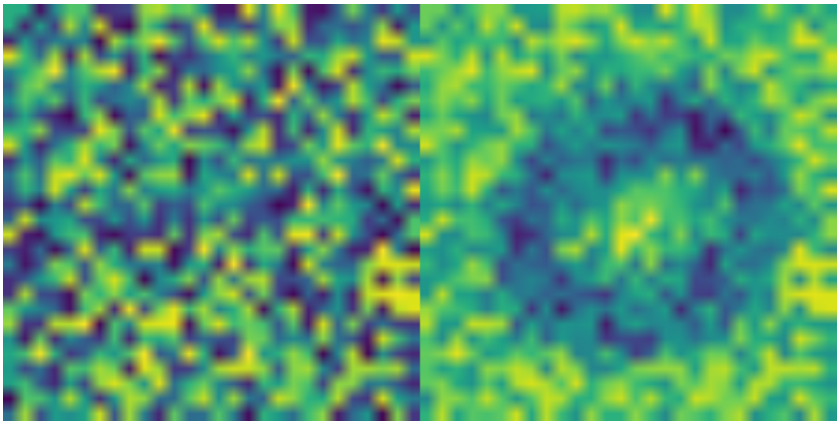

Figure 2b: *Weights of the model before and after training*

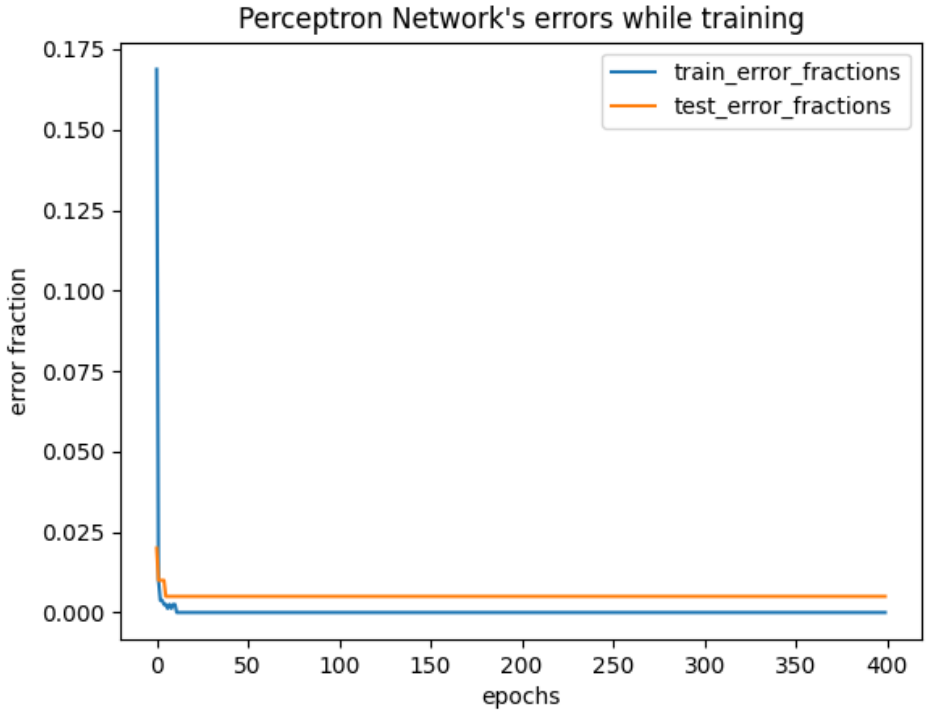

Figure 2c: *Error fractions of the training process by epochs, tested on the train and test set each epoch*

	A	B	C	D	E	F	G	H	I
	label	2	3	4	5	6	7	8	9
	0	20	10	9	39	33	12	11	11
	1	80	90	91	61	67	88	89	89

Figure 2d: *Inference output of model on images of 2 to 9*

Discussion: We found that for the optimal theta, the Hebb model achieved only a f1 score of 0.81 on the test set, but the Perceptron reached almost 100% f1 score (similar to precision and recall). For the Hebb model, the weights make a shape of a 0, while perceptrons' weights make 1. The Perceptron model also has a trainable bias weight, which helped with the accuracy of the model. The weight updating algorithm of the Perceptron looks at the real time output of the model and its loss when considering new weights' values, while the Hebb learning rule does not. This would have helped significantly with the training process as well. The Perceptron learning algorithm is better at achieving better performance metrics.