1. Source Code:

Colab: https://colab.research.google.com/drive/1G0zK954c-RR3sbiyiSPurhLPrdjs9BG5
GitHub: https://github.com/steve3p0/cs510dl

2. Comparison of Two Fully Connected Networks

We were tasked with training and comparing two fully connected NN classifiers for recognizing images of clothing as specified by the Fashion MNIST data. Both networks shared the following hyperparameters and only differed by their number of layers. Model A had only one hidden layer and Model B had 2. The input vector has 784 cells that represent a 28 x 28 pixelated image of clothing.

Shared Hyperparameters	
Batch Size	30
Learning Rate (η)	0.001
Momentum (α)	0
Activation Function (ϕ)	ReLU
Neurons per Hidden Layer	1024
Input Vector	784
Output Vector	10

Table 2.1: Shared Hyperparameters

Metrics		
	Model A	Model B
Hidden Layers	1 x 1024	2 x 1024
Test Accuracy	85.72%	85.48%
Test Loss	0.2699	0.3479
Train Loss	0.3758	0.2790
Run-Time	4 m 27 s	4 m 48 s
Epochs	20	20

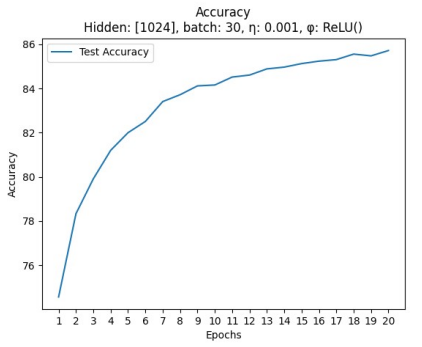
Table 2.2: Shared Hyperparameters

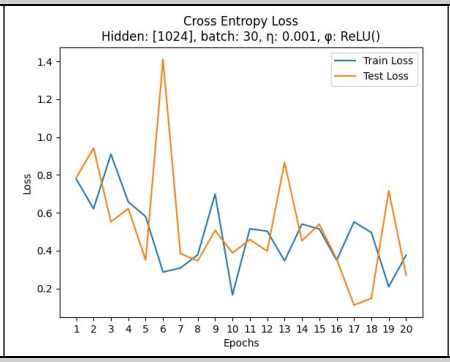
Submit a one paragraph description of the difference in accuracy of the two networks and why.

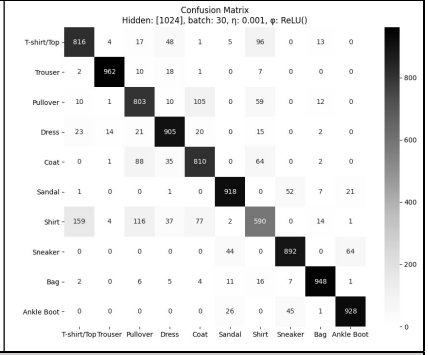
Both networks report an accuracy of roughly 85%. The difference in accuracy is almost non-existent. Additionally, both networks have similar cross-entropy loss values and even run times. And it is obvious from the graphs below that adding a hidden layer to Model B, had negligible impact. It appears based on the cross-entropy loss graphs that both of these networks overfit and may not generalize well. The confusion matrix tells us that both models made the essentially the same errors. I generated heat maps as well to illustrate what the models were learning about the clothing images, but they look just like noise.

Why didn't an extra hidden layer improve accuracy? I am not sure. It would stand to reason that the network could abstract more information about the different clothing images with an additional hidden layer. However, perhaps a hidden space of 1028 neurons was enough?

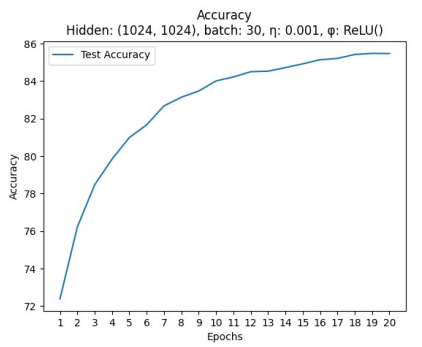
Model A

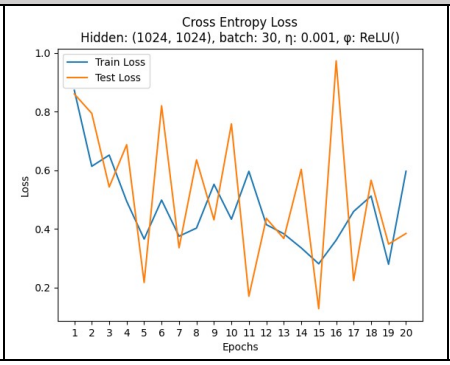


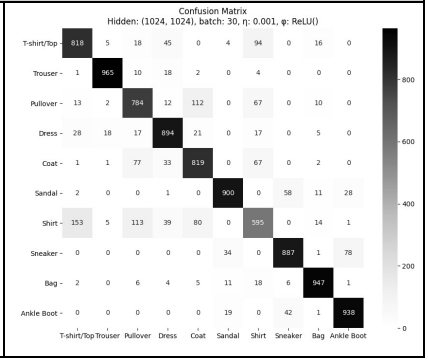




Model B







3. Experimenting on Model B with different Hyperparameters

Tasks:

- Submit a table containing accuracy for the 3*4*2 = 24 cases.
- For the best and worst cases, write a one para description that explains the difference.

Table of all 24 Cases

No.	Model	Train Loss	Test Loss	Test Accuracy	Epoch
1.	Batch=1, φ=ReLU, η=1.0	3.4573	1.8199	37.22%	1/20
2.	Batch=1, φ=ReLU, η=0.1	0.0000	0.0978	90.42%	20/20
3.	Batch=1, φ=ReLU, η=0.01	0.0001	0.0000	90.76%	17/20
4	Batch=1, φ=ReLU, η=0.001	2.7080	0.1101	87.31%	19/20
5	Batch=1, φ=Sigmoid, η=1.0	0.0000	0.5299	81.54%	20/20
6	Batch=1, φ=Sigmoid, η=0.1	4.3306	0.0000	90.29%	13/20
7	Batch=1, φ=Sigmoid, η=0.01	0.0107	0.2466	88.19%	20/20
8	Batch=1, φ=Sigmoid, η=0.001	5.9466	0.0267	81.68%	19/20
9	Batch=10, φ=ReLU, η=1.0	1.2072	24.9780	39.25%	6/20
10	Batch=10, φ=ReLU, η=0.1	0.0075	1.4949	90.62%	20/20
11	Batch=10, φ=ReLU, η=0.01	0.0631	0.4080	90.05%	18/20
12	Batch=10, φ=ReLU, η=0.001	0.4604	0.1496	86.37%	20/20
13	Batch=10, φ=Sigmoid, η=1.0	0.5974	0.1035	88.96%	18/20
14	Batch=10, φ=Sigmoid, η=0.1	0.0683	0.7187	89.92%	20/20
15	Batch=10, φ=Sigmoid, η=0.01	1.4434	1.1528	86.39%	20/20
16	Batch=10, φ=Sigmoid, η=0.001	1.3892	0.4149	75.46%	20/20
17	Batch=100, φ=ReLU, η=1.0	0.1947	0.3007	89.99%	20/20
18	Batch=100, φ=ReLU, η=0.1	0.3017	0.3189	88.14%	20/20
19	Batch=100, φ=ReLU, η=0.01	0.5334	0.5139	82.60%	20/20
20	Batch=100, φ=ReLU, η=0.001	1.1868	1.1994	69.52%	20/20
21	Batch=100, φ=Sigmoid, η=1.0	0.3949	0.3946	86.00%	20/20
22	Batch=100, φ=Sigmoid, η=0.1	0.5750	0.5655	79.45%	20/20
23	Batch=100, φ=Sigmoid, η=0.01	1.5066	1.5241	57.61%	20/20
24	Batch=100, φ=Sigmoid, η=0.001	2.2758	2.2762	56.91%	20/20

* I assume that losses reported as 0.0000 did not include enough floating-point precision.

Difference between Best and Worst Cases:

The only difference in hyperparameters between the best and worst is the learning rate.

Model	Batch Size	Activation Function φ	**Learning Rate η**	Train Loss	Test Loss	Test Accuracy	Epoch
Case #1	1	ReLU	**1.00**	3.4573	1.8199	37.22%	1/20
Case #3	1	ReLU	**0.01**	0.0001	0.0000	90.76%	17/20

With its batch size set to the minimum (1) and its learning rate set to the maximum (1.0), Case #1 was already set up for failure. This combination of hyperparameters will lead to unstable gradients. The model weights are updated after every single sample at the maximum learning rate. This is why Case #1 reached its highest accuracy just after the first epoch. As the gradients got larger and larger, it became difficult to minimize the cost function. Also, depending on your activation function, your gradients might vanish or explode. Case #1 used ReLU as its activation function. ReLU has no upper bound on the input it passes thru when activated. Originally, Cases #1, 2, and 9 reported losses of NaN, because the gradients exploded. To solve this, I used gradient scaling, with a maximum norm of 2, to clip the gradient so that it stays within a reasonable range.

Case #3 reported the highest accuracy. Its learning rate was .01. This meant that the model weights were updated after every single sample at a more reasonable rate.

4. Training on Polluted Data

From 3 above, pick the best set of parameters for the 2 FC layer network. Pick 9 sets of 1% of images from each of the 10 categories and add them to the other 9 categories. This is called pollution. Run the best case on this data (training and then test on the unpolluted test data)

Description	Model	Train Loss	Test Loss	Test Accuracy	Epoch
Best Model: Clean Train Data	Batch=1, φ=ReLU, η=0.01	0.0001	0.0000	90.76%	17/20
Best Model: 1% Polluted Train	Batch=1, ϕ=ReLU, η=0.01	0.0000	0.0002	90.12%	19/20

I do not see any real change in accuracy.
I polluted 1% of the training data per instructions in the assignment handout.
I tested on the original unpolluted test data.

Pick 9 sets of 1% of images from each of the 10 categories and add them to the other 9 categories.
Perhaps I did not pollute the training data enough?

I polluted the data thru bash using a combination of awk and sed. I sorted and split the training file by the first character into 10 files of 10,000 samples each. Then, in each file, incremented the label by 1 for the first 60 lines. For label, 9, I changed it to zero. I then appended all these files into one and shuffled the order around, and opened this new polluted file and trained on it, just like I did the unpolluted training file.

What was the expected outcome?