

Polytechnique Montréal
Department of Computer and Software Engineering



Report Assignment 4
Machine Learning

Missipsa Annane, 2252373

28 november 2025, 10PM

1 Putting It All Together (35 points)

The models (PyTorch and Custom MLP) were trained on Fashion-MNIST for 10 epochs with a batch size of 128. All figures include both curves (Custom and PyTorch) on the same plot.

Validation Accuracy (20 pts)

Learning Rate = 0.0001 (Very Small)

```
python fmnist.py --lr 0.0001 --epochs 10 --batch-size 128 --plot
```

At this learning rate, both models learn very slowly. The validation accuracy remains low (around 66% for the custom model and 39% for PyTorch). Such slow convergence is expected because the parameter updates are extremely small for this number of epoch.

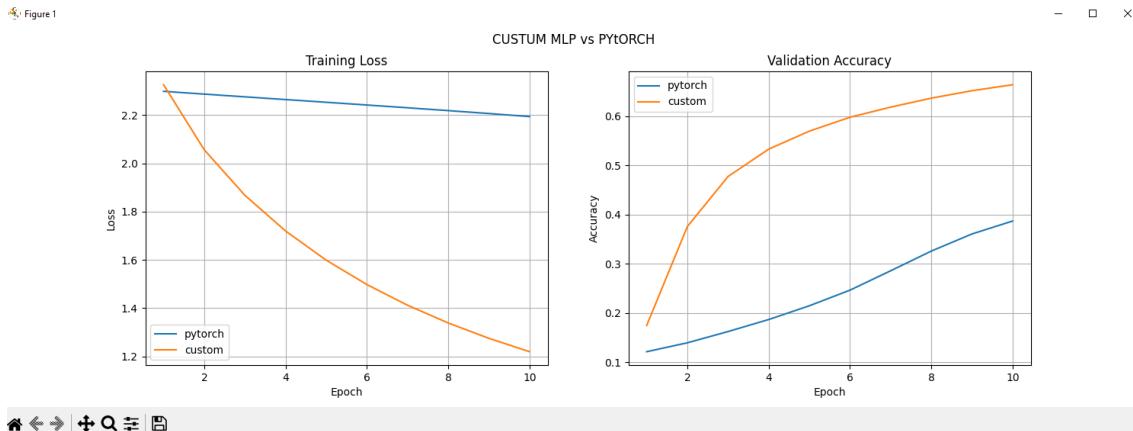
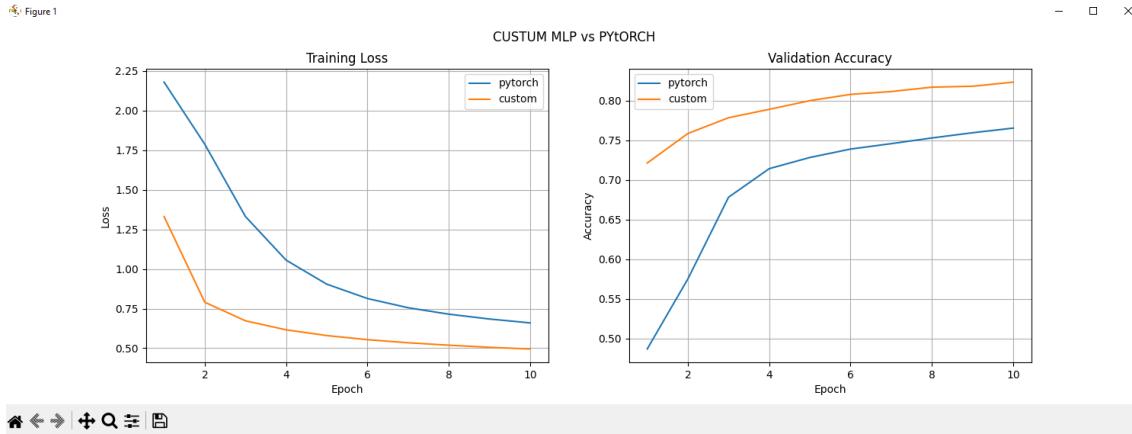


Figure 1: Training loss and validation accuracy for LR = 0.0001

Learning Rate = 0.002 (Medium / Good)

```
python fmnist.py --lr 0.002 --epochs 10 --batch-size 128 --plot
```

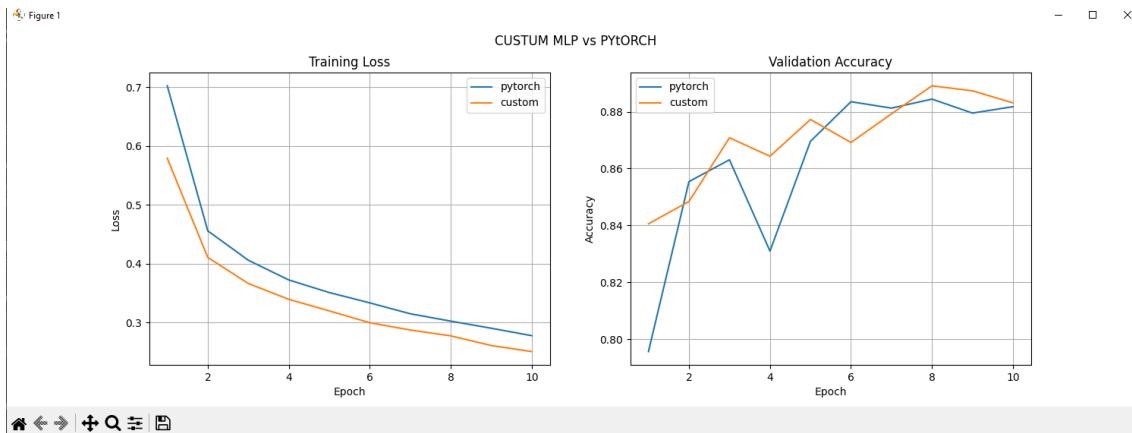
With this learning rate, both models converge in a stable way. The training loss decreases steadily over the epochs for both curves, but the custom MLP always has a lower loss than the PyTorch model. Similarly, the custom model reaches a higher validation accuracy (around 82% at the end) and converges faster, while the PyTorch model also improves, it does so more slowly and its validation accuracy levels off around 76%–77%, noticeably below the custom model.

Figure 2: Training loss and validation accuracy for $LR = 0.002$

Learning Rate = 0.1 (Very Large)

```
python fmnist.py --lr 0.1 --epochs 10 --batch-size 128 --plot
```

With a learning rate of 0.1, training becomes unstable for both models. The PyTorch model shows oscillating validation accuracy between roughly 0.79 and 0.89 across epochs, rather than improving smoothly. The custom model behaves similarly, fluctuating around 0.84–0.89 instead of converging. Although the final accuracies appear high, the curves are irregular and non-monotonic, indicating that the updates are overshooting and preventing stable optimization.

Figure 3: Training loss and validation accuracy for $LR = 0.1$

Test Performance (15 Points)

Batch Size and Learning Rate. Across all experiments, the training process was stable with the default batch size of 128 and the optimal learning rate identified in the previous section. Although the instructions mention that batch size may need to be changed to ensure convergence, this was not necessary here. I initially trained with 10 epochs per configuration, then verified with 15 and later with 7 epochs. All configurations converged properly regardless of the epoch count, and no instability or divergence was observed.

Because my hardware allowed fast training, reducing or increasing the batch size had no practical advantage. The training time per epoch remained short and stable across all experiments, so I kept the default batch size unchanged. Similarly, the

learning rate that performed best in the validation-accuracy experiment ($\eta = 0.002$) remained effective across all hidden-layer variations, and no further tuning was required.

Table 1 reports the effect of modifying the first hidden layer while keeping the second layer fixed at 128 units. Very small layers (8 or 16 units) slightly reduce the test accuracy compared to wider configurations, as expected from their limited capacity. Increasing the first hidden layer to 64 units improves performance for both models, and using 1024 units yields the highest accuracy for the custom MLP, at the cost of a noticeable increase in training time. (Hidden Size 1 = 8 experiment shows a higher training time because it was run earlier on a different hardware setup and with more epochs. Due to time constraints, I did not rerun it under the final standardized settings.)

Hidden Size 1	Model	Test Acc. (%)	Time (s)	Notes
8	PyTorch	82.84	326	Converges steadily; slightly slower.
	Custom	83.29	307	Faster convergence; slightly better accuracy.
16	PyTorch	83.01	146	Slightly slower; similar convergence.
	Custom	83.29	144	Faster and marginally better accuracy.
64	PyTorch	83.09	145	Lower accuracy; slightly faster.
	Custom	84.22	147	Better accuracy; slightly slower.
1024	PyTorch	83.46	145	Lower accuracy; faster than custom.
	Custom	85.79	168	Highest accuracy; slower due to large hidden size.

Table 1: Effect of varying the first hidden layer size while keeping the rest of the network unchanged.

Table 2 summarizes the effect of modifying the second hidden layer while keeping the first layer fixed at 256 units. As expected, very small layers (8 or 16 units) reduce the model’s expressive power and slightly lower the overall accuracy. Increasing the second hidden size to 64 improves stability and performance for both models. The largest configuration (1024 units) yields the best accuracy for the custom MLP, but also increases computation time due to the greater number of parameters. Across all settings, the custom model consistently achieves slightly higher test accuracy than the PyTorch baseline, while the training times remain similar except for the largest hidden size.

Hidden Size 2	Model	Test Accuracy (%)	Train Time (s)	Notes
8	PyTorch	83.14	145	Lower accuracy; similar speed.
	Custom	83.80	145	Better accuracy; same speed.
16	PyTorch	83.41	145	Lower accuracy; slightly faster.
	Custom	84.48	146	Better accuracy; similar speed.
64	PyTorch	83.55	147	Good stability; moderate accuracy.
	Custom	84.60	149	Higher accuracy; slightly slower.
1024	PyTorch	83.90	170	Larger layer increases cost; minor accuracy gain.
	Custom	85.40	176	Strong accuracy; slower due to large hidden size.

Table 2: Effect of varying the second hidden layer size while keeping the first hidden layer unchanged.

Table 3 shows the results of removing the second hidden layer entirely. Even with a simplified architecture, both models reach competitive performance on the test set. The custom MLP achieves a slightly higher accuracy and smoother learning behavior across epochs, while the PyTorch version converges marginally faster but with lower final performance. This indicates that the custom implementation maintains good representational capacity even with a reduced architecture, and continues to generalize slightly better than the PyTorch baseline.

Model	Test Accuracy (%)	Train Time (s)	Notes
PyTorch (1-layer MLP)	84.66	210	Slightly lower accuracy; slower early learning.
Custom (1-layer MLP)	85.49	222	Higher accuracy across epochs; better generalization.

Table 3: Effect of removing the second hidden layer while keeping the original first hidden layer size (256).

Overall, increasing the hidden layer sizes tends to improve test accuracy at the cost of longer training times, with diminishing returns for very large layers (1024 units). Smaller layers are faster but underperform slightly, while the original (256, 128) architecture offers a good compromise between speed and performance. In all cases, the custom MLP remains slightly more accurate than the PyTorch baseline.

2 [BONUS] Theoretical question (extra 50 points)

We consider the deep linear network

$$f(x; W_{0:L}) = W_L W_{L-1} \cdots W_0 x,$$

with training loss

$$\mathcal{L}(W_{0:L}) = \frac{1}{2} \|Y - W_L \cdots W_0 X\|_F^2,$$

where $X \in \mathbb{R}^{n_0 \times m}$, $Y \in \mathbb{R}^{n_{L+1} \times m}$, and we assume:

- $L \geq 1$,
- $YX^\top \neq 0$,
- there are no bottlenecks: $\min_\ell n_\ell = \min\{n_0, n_{L+1}\}$.

A point is a *saddle point* if:

1. it is a **critical point**: $\nabla \mathcal{L} = 0$,
2. but is **not** a local or global minimum.

We prove both properties for $W_{0:L} = 0$.

1. $W_{0:L} = 0$ is a critical point

Define the global matrix

$$A = W_L \cdots W_0.$$

Then the loss is

$$\mathcal{L}(W_{0:L}) = g(A), \quad g(A) = \frac{1}{2} \|Y - AX\|_F^2.$$

The gradient of g with respect to A is

$$\nabla_A g(A) = (AX - Y)X^\top.$$

To compute the gradient with respect to each W_k , we apply the chain rule for matrix products:

$$W_{k+1:+} := W_L W_{L-1} \cdots W_{k+1} \quad (\text{product of all layers after } W_k),$$

$$W_{k-1:-} := W_{k-1} W_{k-2} \cdots W_0 \quad (\text{product of all layers before } W_k).$$

At $W_0 = \cdots = W_L = 0$, every truncated product $W_{k+1:+}$ or $W_{k-1:-}$ contains a zero matrix, so

$$\nabla_{W_k} \mathcal{L}(0, \dots, 0) = 0.$$

Thus,

$$\nabla \mathcal{L}(0, \dots, 0) = 0,$$

so $W_{0:L} = 0$ is a **critical point**.

2. $W_{0:L} = 0$ is not a global minimum

At $W_{0:L} = 0$, we have $A = 0$, so the loss equals

$$\mathcal{L}(0, \dots, 0) = \frac{1}{2} \|Y\|_F^2.$$

Consider instead the optimal linear map

$$A^* = \arg \min_A \frac{1}{2} \|Y - AX\|_F^2.$$

It satisfies

$$\nabla_A g(A^*) = 0.$$

Since

$$\nabla_A g(0) = -YX^\top \neq 0,$$

we have $A = 0$ is *not* optimal, so

$$g(A^*) < g(0).$$

Finally, because there are *no bottlenecks*, any matrix A^* can be expressed as a product

$$A^* = W_L^* \cdots W_0^*.$$

Therefore

$$\mathcal{L}(W_0^*, \dots, W_L^*) = g(A^*) < g(0) = \mathcal{L}(0, \dots, 0),$$

showing that $W_{0:L} = 0$ is **not** a global minimum.

Conclusion

We have shown that:

- $W_{0:L} = 0$ is a critical point,
- but it is not a global minimizer.

Therefore, $W_{0:L} = 0$ is a **saddle point** of the loss \mathcal{L} .

References

- [1] If you use ChatGPT to edit grammar in your report, you have to explicitly state it in the report : I used ChatGPT to help me format mathematical equations as I do them hand-written (see previous assignments). I wanted to learn more LaTex and used the help.
- [2] Illuri Sandeep. *Saddle Points in Deep Learning: The Invisible Roadblocks in Optimization*. Medium blog, 2025. <https://illuri-sandeep5454.medium.com/saddle-points-in-deep-learning-the-invisible-roadblocks-in-optimization-bc70263>
- [3] El Mehdi Achour, François Fleuret, Sébastien Gerchinovitz. *The Loss Landscape of Deep Linear Neural Networks: a Second-order Analysis*. *Journal of Machine Learning Research*, 25:1–76, 2024. <https://jmlr.org/papers/volume25/23-0493/23-0493.pdf>
- [4] Khan Academy. *Saddle points*. YouTube video, 2016. https://youtu.be/8aAU4r_pUUU?si=RDNbqUfBanbTzpL7U