Machine Learning Project 2 Report

Ahmed Tlili, Cyrine Akrout, Ahmed Zguir
EPFL

May 30, 2024

1 Introduction

This report presents our implementation and evaluation of various machine learning methods. The goal is to classify images from the Fashion-MNIST dataset into their respective categories using deep learning techniques, including Multi-Layer Perceptrons (MLPs), Convolutional Neural Networks (CNNs), and Transformer models, all implemented in PyTorch. The class main is a script that utilizes these classes to train a model on a specified set and predicts on given samples. Since the test data is not provided, we compared the performances of the models on the validation set which is 20% of the training set. We also implemented Principal Component Analysis (PCA) for dimensionality reduction to enhance the MLP model's performance.

2 Methods Implemented:

2.1 Multi-Layer Perceptrons (MLPs)

The `MLP` class is a neural network model designed for classification tasks, structured without any convolutional layers. It features a series of fully connected (linear) layers with ReLU activation functions. The model consists of an input layer, three hidden layers (transforming input size to 128, 128 to 32, and 32 to 16 units respectively), and an output layer corresponding to the number of classes.

2.2 Convolutional Neural Networks (CNNs)

The `CNN` class is a neural network model tailored for classification tasks, employing convolutional layers to process input data. The architecture includes three convolutional blocks, each followed by ReLU activations and max-pooling layers, which progressively transform the input into more abstract feature maps. Additionally, dropout layers are introduced after each convolutional block to prevent overfitting. The first convolutional block applies a convolutional layer with 32 filters, followed by a ReLU activation function, a max-pooling layer to reduce spatial dimensions, and a dropout layer with a rate of 0.3. The second convolutional block uses 64 filters, followed by another ReLU activation, max-pooling layer, and dropout with the same rate. The third block increases the number of filters to 128, with subsequent ReLU activation, max-pooling, and dropout. After the convolutional blocks, the output is flattened into a vector and passed through a series of fully connected (linear) layers: The first fully connected layer transforms the flattened vector (with dimensions reduced to $128 \times 3 \times 3$) into 256 units, followed by a ReLU activation and a dropout layer with a rate of 0.5. The second fully connected layer further reduces the dimensionality to 128 units, followed by another ReLU activation. The final layer outputs logits for the predicted classes.

2.3 Vision Transformer(VIT)

The `MyViT` class represents the entire Vision Transformer model, designed for image classification. It includes several key components: the model is initialized with input dimensions (`chw` = (1, 28, 28)), number of patches (`n_patches` = 7), number of transformer blocks (`n_blocks` = 4), hidden dimension size (`hidden_d` = 64), number of attention heads (`n_heads` = 8), and output dimension size (`out_d` = 10). The `patch_size` is calculated based on the input dimensions and the number of patches. The `linear_mapper` maps the input patch vectors to the hidden dimension size. A learnable classification token is added to the sequence of patch tokens, and positional embeddings are generated to provide spatial information. Transformer blocks are created using the `MyViTBlock` class, and an MLP is used for the final classification. During the forward pass, input images are divided into patches using the `patchify` function. Each patch is mapped to the hidden dimension size using the `linear_mapper`. The classification token is prepended to the sequence of patch tokens, and positional embeddings are added. The tokens are passed through the transformer blocks, and the classification token is extracted and passed through the final MLP to obtain the output predictions.

3 Training

3.1 CustomWarmupScheduler Class

The `CustomWarmupScheduler` class is a custom learning rate scheduler. This scheduler gradually increases the learning rate during the warmup phase and then decreases it linearly. `warmup_steps` specifies the number of steps during which the learning rate will increase linearly. `total_steps` specifies the total number of training steps. The `get_lr` method computes the learning rate for the current step. If the current step is less than the warmup steps, the learning rate is increased linearly. After the warmup phase, it decreases linearly.

3.2 Trainer Class

The `Trainer` class is designed to facilitate the training and evaluation of deep neural network models. The `train_all` method orchestrates the full training process by calling the `train_one_epoch` method for each epoch. This method sets the model to training mode and iterates over the batches of data provided by the `dataloader`. For each batch, it performs forward propagation to compute the logits, calculates the loss, performs backpropagation, and updates the model parameters using the optimizer. Loss values are accumulated and printed periodically to monitor training progress. The `predict_torch` method is used for making predictions on validation or test datasets. It sets the model to evaluation mode, turns off gradient computation to save memory and speed up computation, and iterates over the data batches to compute the predictions. The predicted labels are returned after being processed through a softmax function to obtain class probabilities.

4 Performance Comparisons

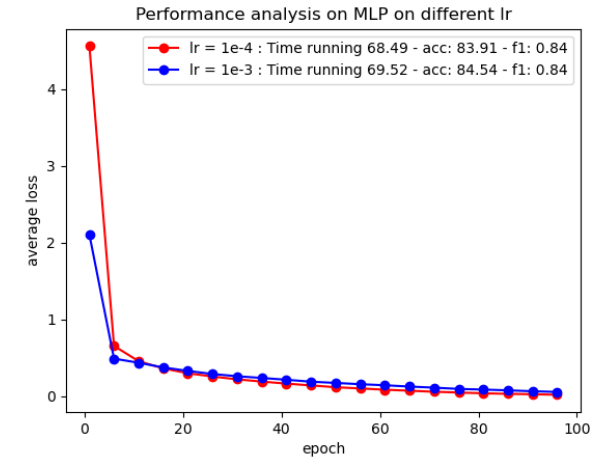

Figure 1: MLP's Accuracy (Validation Set). Learning rate 1e-3 seems to be optimal.

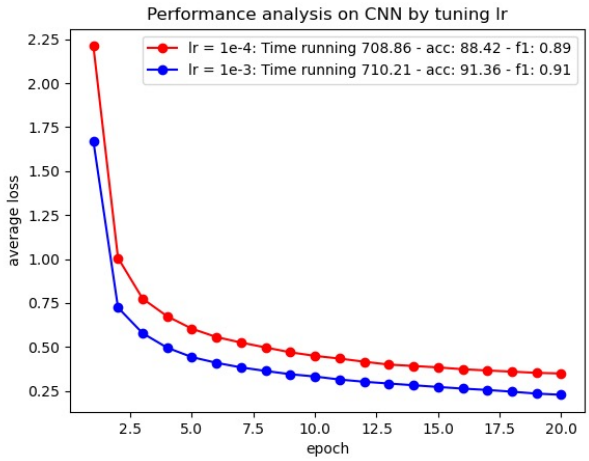

Figure 2: CNN's Accuracy (Validation Set).

Learning rate 1e-3 seems to be optimal.

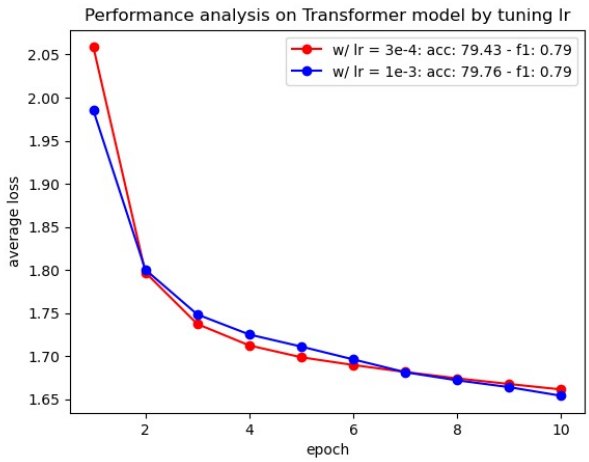

Figure 3: VIT's Accuracy (Validation Set). It takes 15 minutes to run 10 epochs to get an accuracy of almost 80%.

5 Principal Component Analysis (PCA)

The PCA class performs Principal Component Analysis for dimensionality reduction. The `find_principal_components()` method calculates and saves the mean and principal components of the training data, returning the explained variance. It uses the covariance matrix and eigen decomposition to identify the top principal components. The `reduce_dimension()` method applies these components to new data, reducing its dimensionality. The plot provided below shows how PCA affects the performance and speed of the MLP.

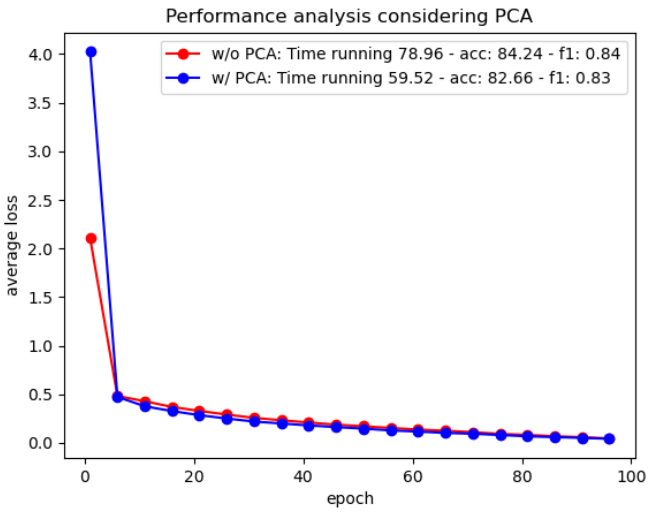

Figure 4: Impact of PCA on MLP's Accuracy (Validation Set).

We clearly notice that the model using PCA (with d = 100) is nearly 30% faster.

6 Conclusion

Finally this is the overall performance on the validation set (since the test data was not provided) after the optimisation of the hyperparameters and running each of the following commands:

```
(1) $ python main.py --data dataset --method nn
--nn_type mlp --lr 1e-3 --max_iters 100
(2) $ python main.py --data dataset --method nn
--nn_type mlp --lr 1e-3 --max_iters 100 --use_pca
--pca_d 100
(3) $ python main.py --data dataset --method nn
--nn_type cnn --lr 1e-3 --max_iters 20
(4) $ python main.py --data dataset --method nn
--nn_type transformer --lr 3e-4 --max_iters 10
```

-	MLP	CNN	VIT
w/o PCA	(1)84.7%	(3)91.12%	(4)80%
w/ PCA	(2)82.66%	-	-

Table 1: Benchmark on Validation Set

The **CNN** model achieved the highest accuracy of 91.12%, making it the best performing model among the three. This is likely due to CNNs being particularly effective at capturing spatial hierarchies in image data through convolutional layers.