CSC3022F ANN Assignment

Jing Yeh
YHXJIN001

Contents

Experiment Procedure

Each model is evaluated 5 times. We then take the mean accuracy on the test set, and the mean number of epochs taken for training. Sample size of 5 is chosen to balance between statistical robustness and time constraint.

Training

After reading in the training set data from FashionMNIST, we further split the set into the training set we will use and a validation set, using a 5:1 split.

The neural network is trained using a forward pass and a backward pass. For each epoch, the forward pass feeds the input features through the model. We store the predictions of the model and then pass through our choice of loss function, which is Cross Entropy in this case as we are performing a multiclass classification task. The gradients and updates to the model parameters are done by the selected optimiser during the backwardpass.

Afterward, we evaluate the performance of the model on the validation set by comparing the label of data in the validation set, and the model's prediction when being fed the same data. We calculate the percentage that the model correctly predicts the image class as the validation accuracy.

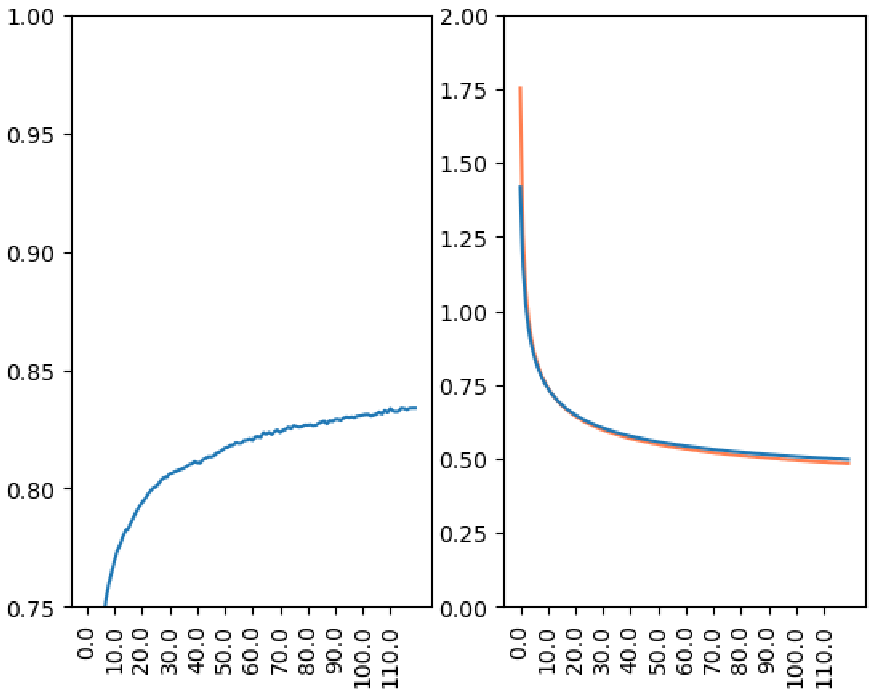
NoConv1Layer

We begin with a simple 1 layer, linear model.

```
class ModelNoConv1Layer(nn.Module):
    def __init__(self, input_size, num_classes):
        super(ModelNoConv1Layer, self).__init__()
        self.network = nn.Sequential(
            nn.Flatten(),
            nn.Linear(784, num_classes)
            # softmax built into cross entropy loss already
        )
    def forward(self, x):
        return self.network(x)
```

Initial Hyperparameters:

- optimizer: SGD and Adam
- batch size = 64
- loss CE
- learn rate 1e-3



With SGD, we have not achieved convergence even after 120 epochs. We achieved a 82.43% accuracy on this. We can speed up training by using a smaller learning rate. However, a smarter solution will be to simply use an optimiser that automatically adjusts the learning rate for us. One such optimiser is Adam

Early stopping

Furthermore, to facilitate the training process, I have implemented an early mechanism that compare the current validation epoch loss to the previous 3 validation losses, with a patience parameter set to 3 epochs. This will break the training loop if we don't see improvements in validation loss after 3 epochs, comparing to the previous three epoch's validation losses

Based on our early stopping mechanism, the model stops improving after epoch 19. Our test accuracy has also improved to 84%, after switching to using Adam. We will experiment with adding more layers to improve the performance of the model

ModelNoConv

```
class ModelNoConv(nn.Module):
    def __init__(self, input_size, num_classes):
        super(ModelNoConv, self).__init__()
        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(input_size, 512),
            nn.ReLU(),
            nn.Linear(256, 128),
            nn.ReLU(),
            nn.Linear(128,64),
            nn.ReLU(),
            nn.Linear(64, num_classes)
            # softmax built into cross entropy loss already
        )
```

```
    def forward(self, x):
        return self.classifier(x)
```

We have added a few more layers to the model:

- With each layer halving the dimensionality of features to facilitate classification.
- Added Relu activation functions to introduce non linearity

The training process terminated after 8 epochs, with no significant improvement in test accuracy. It might be the window size of our early stopping mechanism is too small, while the loss value being minimised by Adam tend to fluctuates [2]. We now compare the current epoch validation loss to the previous 5 losses instead, as opposed to 3.

This has significantly improved our model. The training loop terminates at epoch 15. The mean test accuracy has now been improved to 87.94%.

We will investigate if adding dropout layers to our model could improve its performance.

ModelNoConvDropout

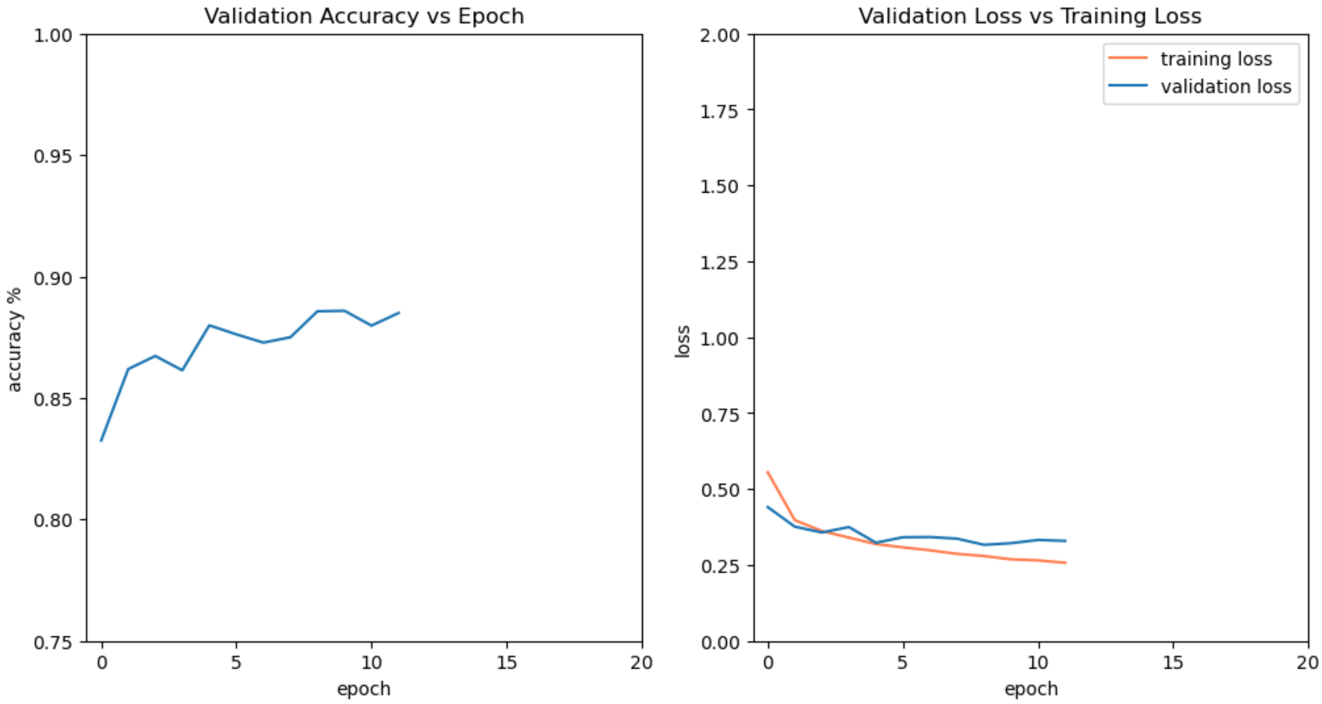
```
class ModelNoConvDropout(nn.Module):
    def __init__(self, input_size, num_classes, dropout):
        super(ModelNoConvDropout, self).__init__()
        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Dropout(dropout),
            nn.Linear(input_size, 512),
            nn.Dropout(dropout),
            nn.ReLU(),
            nn.Dropout(dropout),
            nn.Linear(512, 256),
            nn.Dropout(dropout),
            nn.ReLU(),
            nn.Linear(256,128),
            nn.Dropout(dropout),
            nn.Linear(128, 64),
            nn.Dropout(dropout),
            nn.ReLU(),
            nn.Linear(num_classes)
            # softmax built into cross entropy loss already
        )
    def forward(self, x):
        return self.classifier(x)
```

We will start with a dropout rate of 50%. This does not improve our model's performance (Mean test set accuracy = 84.24%). It might be because our dropout rate is too large, so we try dropout rates of 10%, 20%, 30%, and 40% as well.

Dropout %	Mean Epochs	Mean Accuracy
5%	13	88.26%
10%	19	88.63%
20%	15	87.51%
30%	14	86.49%
40%	17	85.47%
50%	19	84.24%

It seems like adding dropout layers can improve the performance of the model. However, this is achieved by using a lower than usual dropout rate of 10 percent. Typically dropout rate ranges from 20% to 50% [2], but because our model is relatively small, 10% seems to suffice. Anything below that will essentially have the same effect as having no dropout layers, as the probability of deactivating neurons is too small to be effective.

The below graph shows the training\validation losses and validation accuracies after each epoch



Learning Rate

To see the effects of learning rate on our current best performing model, 5 different learning rates are used

Learn Rate	Mean Epochs	Mean Accuracy
1e-5	100+	87.54%
1e-4	38	89.29%
1e-3	15	88.63%
1e-2	9	71.49% (with high s.d.)
1e-1	5	11%

L2 Regularisation (Weight Decay)

We now experiment with adding weight decays, with our new best model (learning rate = 1e-4):

Decay	Mean Epochs	Mean Accuracy
1e-5	100+	89.25%
1e-4	32	88.96%
1e-3	26	88.02%
1e-2	38	85.08%
1e-1	18	47.34%

We don't see much improvements after applying weight decay to our model. This might be because the input size is relatively small so the need for this technique is not as big. We can thus observe that the lower the weight decay, the better and closer to our original model is, while the higher the weight decay (the fewer features we allow), the lower the mean accuracy, which is a sign of underfitting. This shows that our model seems to be using an optimal number of features.

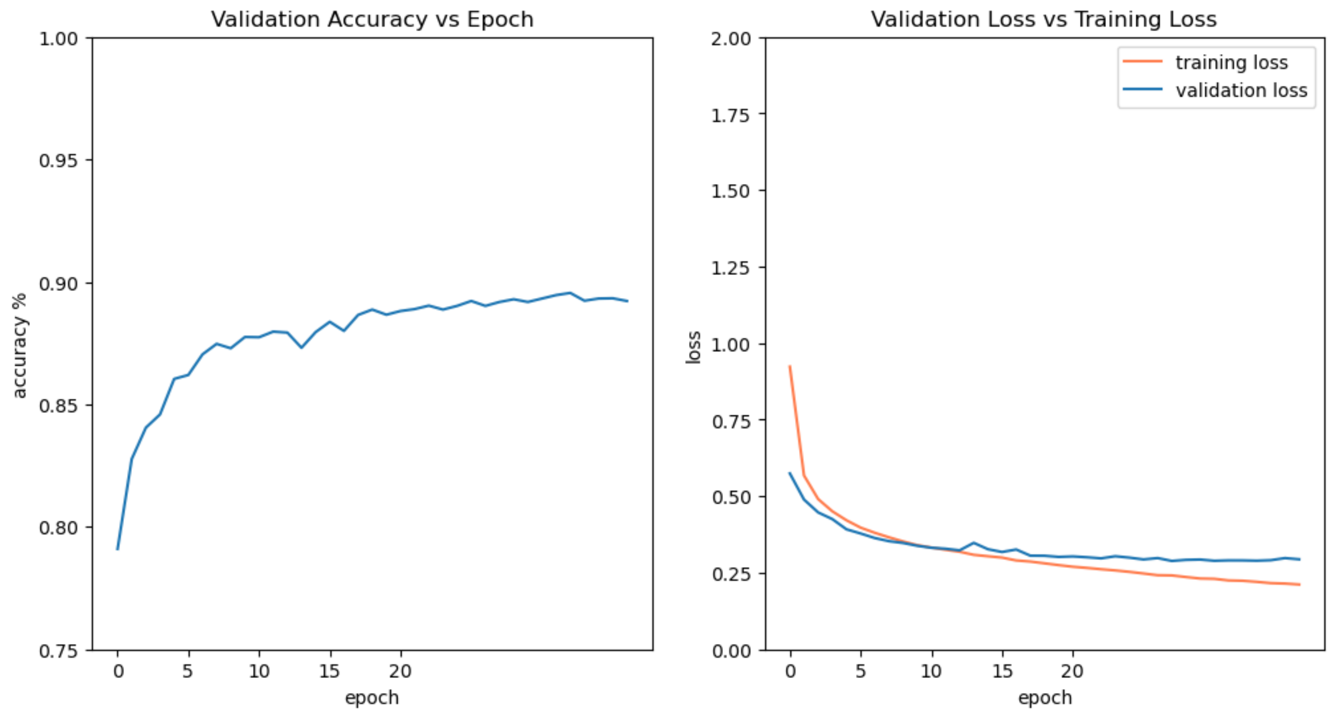
Final Model

After the finetuning processes, we have found a optimal model for the classification of FashionMNIST dataset. The best performing architecture is the ModelNoConvDropout model with 4 hidden layers. The input layer takes in 784 features (flattened 28*28 images). The hidden layers have 512, 256, 128, and 64 units respectively, each followed by a ReLU activation function and a Dropout layer with dropout rate set to 0.1. The final output layer maps to 10 logits. We do not apply softmax here, as it is built into CrossEntropyLoss already.

We use Adam optimiser with no weight decay and 1e-4 learning rate during training. Cross Entropy Loss is the loss function chosen for this task.

We achieved a mean accuracy of 89.29% on our test dataset.

This is the final training graph



Bibliography

[1]G. E. Hinton and R. R. Salakhutdinov, “Reducing the Dimensionality of Data with Neural Networks,” *Science*, vol. 313, no. 5786, pp. 504–507, Jul. 2006, doi: https://doi.org/10.1126/science.1127647.

[2]Aurélien Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. “O’Reilly Media, Inc.,” 2022.

[3]G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning : with Applications in R*. Springer, 2013.

[4]P. Kashyap, “Understanding Dropout in Deep Learning: A Guide to Reducing Overfitting,” *Medium*, Oct. 30, 2024. https://medium.com/@piyushkashyap045/understanding-dropout-in-deep-learning-a-guide-to-reducing-overfitting-26cbb68d5575

[5]“A Guide to Making Deep Learning Models Generalize Better,” *Turing.com*, Aug. 18, 2022. https://www.turing.com/kb/making-deep-learning-models-generalize-better#variance-bias-trade-off