1 Learning to classify the Fashion apparel

Fashion-MNIST is a dataset of Zalando's article images—consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. Fashion-MNIST is intended to serve as a direct drop-in replacement for the original MNIST dataset for benchmarking machine learning algorithms. It shares the same image size and structure of training and testing splits.

The goal of this task to leverage the power of Machine Learning, coupled with the comfort of PyTorch, to implement an end-to-end Multi-Layered Perceptron based system which can achieve a high classification accuracy on the Fashion-MNIST dataset.

1.1 Building a Vanilla Classifier

1.1.1 Dependencies

Recommended Configuration: (i) Python 3.6 (ii) PyTorch 1.3.1 (iii) Tensorboard 1.14.0 (iv) Numpy 1.16.6 (v) termcolor 1.1.0

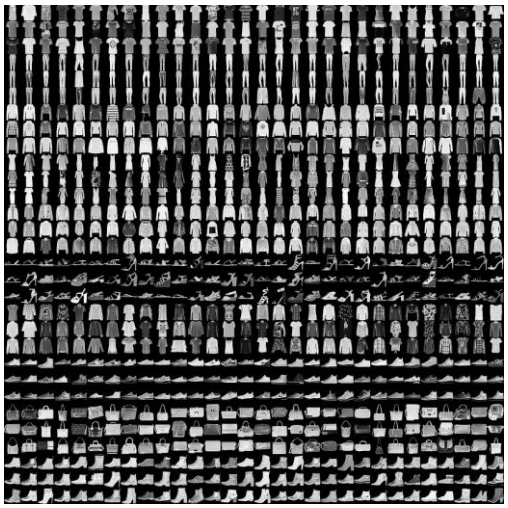

Figure 1: Fashion-MNIST

1.1.2 Starting with the data

PyTorch provides an effective, yet simple to deal with method to handle large datasets in parallel, using the torch.utils.data.dataloader class. The dataloader class, can automatically divide the dataset into batches, while parallelising the tasks amongst multiple processes, To create a dataloader, first you have to create a "Dataset" class (inheriting the torch.utils.data.Dataset class), and overriding the len __ and getitem __ methods. **Please note that we have provided our own specialized dataset , and thus you can't use the Fashion MNIST loader directly from PyTorch.**

(a) In the file data.py complete the methods provided, ensuring that the images are scaled to [0,1]. Please explain the steps undertaken for processing the data in the write-up. Please ensure that you use the hyperparameters from the config file.

1.1.3 Designing the architecture

(a) Define the architecture in the class Network in network.py. Use an architecture containing two hidden layers (784 → 100 → 10) with a ReLU activation after the first layer and a Softmax after the final logits to get probability scores. Note that the input images are sized 28 × 28 and thus the input is obtained by flattening the images, making it sized 784. The architecture has been visually delineated in Figure 2

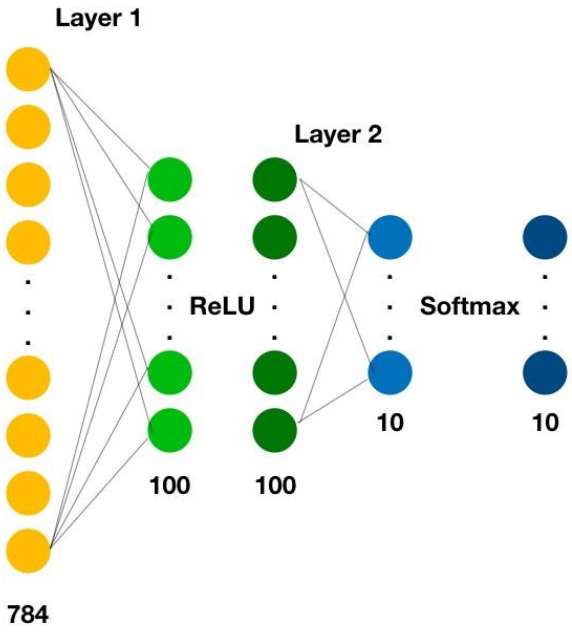

Figure 2: Architecture for Task 1

(b) Write code for saving and loading the model in the load and save methods of Network, which can be used to stop the training in an intermediate epoch and load it later to resume training.

1.1.4 A holistic view of the training

TensorBoard provides the visualization and tooling needed for machine learning experimentation, by exposing an extremely easy to use interface for plotting scalars, images, histograms while training. Plot the following things after every epoch (see below for settings to train the model) using tensorboard:

(a) Training Loss

(b) Test Accuracy

(c) Visualize the 784 weights incoming into any node in the first hidden layer by reshaping in the shape of images, and do this for 2 any hidden nodes.
[Hint: Think about the weights connecting all the input nodes to one node in the first layer]

(d) Current learning rate

1.1.5 Training the model

It's finally time to train your model on the specialized Fashion-MNIST dataset. Using the file train.py, complete the following functionalities in the code:

(a) Create a criterion object, which defines the objective function of our model.

(b) Create an optimizer **(Stochastic Gradient Descent)**, using the hyperparameters provided via the config file.

(c) Complete the backpropogation by resetting the gradients, doing a backward pass, and then updating the parameters of your model.

(d) Learning Rate Scheduling - Use the ReduceLROnPlateau scheduler to anneal the learning rate based on the test set accuracy. Read about the patience and factor parameters of this scheduler, and explain their role. Use the hyper-parameters from the config file

(e) Complete the run method, which takes in the network, and runs it over all the data points.

(f) Finally, train the model using the hyper-parameters in the config file. As a milestone, you should be able to achieve around 75% accuracy on the test set, within the 20 epochs. Include the tensorboard screenshots in the report as well as the final train loss and test accuracy

1.2 Improving our Vanilla Classifier

Complete the following steps **(in order)**. Also, please note that the steps consist of ablation experiments for our training. We will fix the prior choices while performing the next experiment, for our comparisons. Please refer to the TL;DR section for the final set of results to be reported:

(a) Initialize the weights of your Multi-Layered Perceptron in the init weights function using the following strategies:

   (a) All weights and biases zeros

   (b) Using Xaiver Normal initialization for the weights and zeros for the biases

   At the end, report performance on the test set using the two initialization strategies and explain the difference in performance, if any.

(b) Now, think of how we can do data augmentation for our task. Please note that we have modified our dataset, and it is **strongly recommended** to visually inspect both the training and test images. Now train your classifier with data augmentation and report the test accuracy after 20 epochs. Also, explain the augmentation that you have added, and the reason for the change in performance, if any. *[Note: Use the Xavier Normal initialization from the previous experiment]*

(c) Read about Dropout and how it affects training of deep networks. Now, add dropout to your network and include in the write-up the effect of dropout in the performance of your model. Explain the effect of dropout if it causes any, and explain why dropout doesn't help, if it doesn't. *[Note: Use Xavier Normal initialization, and data augmentation from the prior parts]*

TL;DR for reporting results

To make sure you have not missed any part of the question, results for the following experiments have to be reported:

(a) Vanilla Classifier (VC)

(b) VC + Zero weights initialization

(c) VC + Xavier normal weights initialization (XNW)

(d) VC + XNW + Data Augmentation (DA)

(e) VC + XNW + DA + Dropout