Assignment 3: Neural Networks and Deep Learning

UVA CS 4774
Machine Learning Foundation, Deep Learning and Good Uses

January 20, 2022

a *The assignment should be submitted in the PDF format through Collob. If you prefer hand-writing QA parts of answers, please convert them (e.g., by scanning or using an app like Genuis Scan) into PDF form.*

b *For questions and clarifications, please post on Slack.*

c *Policy on collaboration:*

Homework should be done individually: each student must hand in their own answers. It is acceptable, however, for students to collaborate in figuring out answers and helping each other solve the problems. We will be assuming that, with the honor code, you will be taking the responsibility to make sure you personally understand the solution to any work arising from such collaboration.

d *Policy on late homework:*

Homework is worth full credit at the midnight on the due date. Each student has 10 *extension days to be used at his or her own discretion throughout the entire course. You could use the days in whatever combination you like. For example, all late days on 1 assignment or 1 each day over 10 assignments (for a maximum grade of 90% on each). After you've used all 10 days, you cannot get credit for anything turned in late.*

e *: we will use the EXTRA points in YOUR final grade calculation...*

e *Policy on grading: 100 points in total. 40 points for successful code submission and run by TA. 5 points for each question (in a total of 12 to Submit questions) answered in your report (See "What to Submit"). The overall grade will be divided by 10 and inserted into the grade book.*

1 Image Classification with State-of-the-art (SOTA) Deep Library

In this section you will get familiar with training deep learning models for image classification. Your task is to familiarize yourself with the problem described below and then solve the problem by implementing:

1. A multilayer perceptron (MLP)
2. A convolutional neural network (CNN)
3. A multi-class / softmax logistic classifier + PCA (LG+PCA)
4. 0.5 extra credits will be given to those students who achieve the top 10 in the leader-board. This is a class-wide Kaggle-style competition.

As will be described below, you should experiment with a variety of architectures and hyperparameters to obtain the highest accuracy you can on the test set.

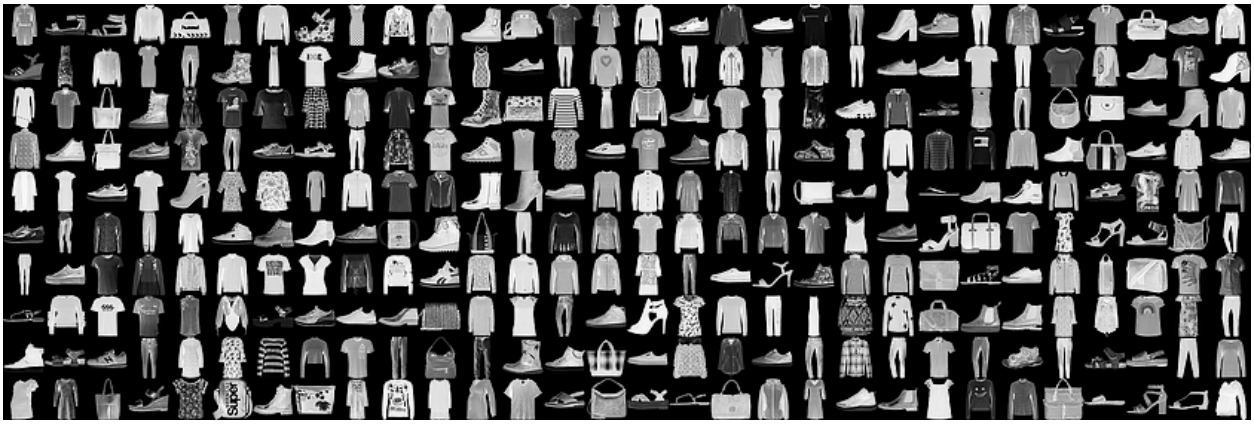

Figure 1: The first 300 images from fashion_train.csv

1.1 About the Data

The problem is to classify images of articles of clothing. The dataset is a modified version of the Fashion-MNIST dataset, which was created by Zalando, a German e-commerce site. There are 10 classes in the dataset:

Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

Each image is 28 by 28 pixels (784 features total), with a single color channel. Each pixel is represented as a value between 0 (black) and 255 (white). The CSV files on Collab contain 1 column of labels followed by 784 columns of pixel values.

The training set is 60,000 labeled examples, while the test set is 10,000 unlabeled examples (dummy labels are included in the first column).

1.2 Setting Up

Please learn to use the following SOTA deep libraries:

- Keras, a user-friendly programming interface to the popular TensorFlow machine learning framework. TensorFlow is a deep learning library that handles tensor (a generalization of a matrix) computations and glues the components of a neural network model together.

Note: It's possible to load Fashion-MNIST using the Keras Datasets API or using the FastAI ImageDataBunch API, but please don't use those preloaded API version. We're using a different version of the data. Please Use the data we provide in your UVA Collab.

1.2.1 Using Google Colab (Recommended for Experimenting)

Google Colab and the provided .ipynb coding template will likely be helpful for completing this assignment (unless you have access to a CUDA-enabled GPU severs). Colab is a Jupyter notebook environment/free cloud service on Google Drive, giving you access to a virtual environment with GPU-support. This option allows you to substantially speed up training time, making experimenting with different models much easier. Instructions:

- BTW: If you do have access to CUDA-enabled GPU servers, you don't need to use Google-Colab. Please skip the rest of items in this list.
- Go to your Google drive > New > More > Colaboratory. Once you have a new Colab open, go to File > upload notebook and select the ipynb template file from your machine.
- Next, go to "Runtime" (top of the screen) > "Change runtime type." Make sure the runtime type option says "Python 3" and hardware accelerator is "GPU" or "TPU" (Tensor Processing Unit). Save.
- To work with our dataset, upload the "fashion_train.csv" file to your Google Drive. In the main method, make sure the "train_file" variable contains the right path. The beginning of the path should remain "/content/drive/My Drive/", but add the correct path after this part. When you run your program, you'll be given a link to an authorization code and be prompted to enter it.
- From here, you should be good to go. A caveat of using Google Colab is that you can't remain connected to the same Colab instance for more than 12 hours. After your 12 hours are up, you'll be reconnected, but if you have a job running, it'll be canceled. Furthermore, this is a free publicly available resource. Performance is usually quite good, but because you'll be sharing resources with other people, it can potentially be slower than expected. Start early to budget enough to experiment thoroughly and avoid Colab slowdowns.

1.2.2 Setting Up Locally (Required for Submission) – DIFFICULT... please start early!!!!

Setting Up Locally (Required for Submission): If you choose Kera and Tensorflow
See the Keras getting started page and follow their installation instructions. Make sure to install TensorFlow first, as they state. Some notes:

- When you're done with experiementing mode from the colab as above, please make sure it's ready to submit by downloading your work as a .py file and then running it with grading_mode = True. This should train your best model and then create the predictions.txt file. To make sure this works, you'll need to install Keras and TensorFlow locally (please don't bother with setting up a GPU or anything of that nature).
- Keras requires TensorFlow. We strongly recommend a virtual environment, or docker, etc.
- On that note, using a virtual environment (such as virtualenv) is always a good idea for projects requiring multiple installations. You can also use a Keras Docker container.
- If you have a CUDA-capable Nvidia GPU, you can make TensorFlow much faster. First, check if your GPU is CUDA-capable. Then see the TensorFlow GPU support instructions. As a warning, going through this process is a bit of a hassle and it can take several hours. There are some pitfalls to avoid, such as making sure you download versions of CUDA and cuDNN that are compatible with TensorFlow.
- Many vision examples in Keras, for example: xray classification with tpus.

Other dependencies: install pandas if you want to use the "get_data" method that comes with the coding template.

1.3 Multilayer Perceptron (MLP)

Architecture: First define a network architecture in the "create_mlp" method by filling out the skeleton code and adding extra layers. For each layer, select the number of units and an activation. Think about the size of the input when selecting the number of units. For the hidden layers, you can experiment with various activation functions, such as tanh, relu, elu, sigmoid, etc. For the output layer, you should select an activation function with a probabilistic interpretation.

Optimizer and Objective Function: Define an optimizer and compile the model. SGD should work fine, but feel free to switch to another optimizer or experiment ("adaptive" optimizers like Adam are very popular and often perform quite well). We recommend starting with a learning rate of around 0.01 and adjusting it as needed. Other options include adding momentum, decay, or nesterov momentum. When you call "compile" you have to specify a loss/objective function. Hint: MSE isn't a good choice in this context.

Training: Call "model.fit" with a validation_split of at least 0.1 (alternatively, 6,000 samples). Select the number of epochs (you shouldn't very many to hit 80% validation accuracy). Optionally, you can select the best model through cross-validation.

What to Submit: Once you're happy with your model (and the validation accuracy is above 80%), retrain on all the data samples, and include the following in your writeup:

- Please provide a summary of your network. For instance, Call print(model.summary()) to view a tabular in Keras. Include this table in your writeup (either by taking a screenshot or by making a new table with the values). Include the total number of params, trainable params, and non-trainable params. Briefly explain your model and why it works.
- Use the object returned by the fit function and matplotlib to create two plots: (1) training loss and validation loss vs epoch number; (2) training accuracy and validation accuracy vs epoch number. Do the plots show evidence of any problems with your model or how many epochs you're using? Please Briefly explain.
- Call the "visualize_weights" function provided in the template. This method creates visualizations of the weights feeding into the units of your first hidden layer by displaying greater weight values as lighter pixels and the lower weights as darker pixels (by default, the .py template will create a folder in your working directory and save the images there; the .ipynb file will just display them). Provide a "num_to_display" argument (such as 10 or 20) and look through them. Please select two visualizations to include in your report. For each one, briefly explain what that unit is attempting to do. Hint: many of the visualizations will appear incomprehensible, but can you find any that seem to resemble articles of clothing?
- Why isn't MSE a good choice for a loss function in for this problem?
- If your best-performing model was an MLP, make sure this model automatically trains, gets predictions, and writes to a "predictions.txt" file before you submit your code.

1.4 Convolutional Neural Network (CNN)

Architecture: Define the network architecture in "create_cnn" by filling out the skeleton code. The minimum requirement is that you use 1 convolutional layer, 1 maxpooling layer, 1 flattening layer, and a dense output layer. This minimum requirement is sufficient to get an accuracy of more than 85% within 10 epochs. Hint: sigmoid and tanh probably won't work well as activation functions for your convolution layers.

Optimizer and Objective Function: Please define the optimizer, loss function, learning rate, and any optimizer parameters you want to use, such as momentum, decay, and nesterov momentum. Select an objective function.

Training: Train the model using a validation_split of at least 0.1 (alternatively, use a validation set of at least 6,000 samples). Tune your model by trying a variety of architectures, hyperparameters, etc. While

cross-validation is not mandatory, you may use scikit-learn's cross-validation and grid search methods, or simply define your own search through the parameter space.

What to Submit: After you've obtained a model you're happy with (and the validation accuracy is greater than 85%), include the following in your report:

- Please provide a summary view of your CNN. For isntance, Call print(model.summary()) in Keras to view a tabular summary of your CNN. Include this table in your writeup (either by taking a screenshot or by making a new table with the values). Briefly explain your model and why it works.
- Please use the history object returned by the fit function and matplotlib to create two plots: (1) training loss and validation loss vs epoch number; (2) training accuracy and validation accuracy vs epoch number. Do the plots show evidence of any problems with your model or how many epochs you're using? Briefly explain.
- How many matrices are outputted by your first convolutional layer when it receives a single testing image?
- What are the dimensions of these matrices?
- What are the dimensions of one of these matrices after it passes through your first maxpooling layer?
- If your best-performing model was a CNN, make sure this model automatically trains, gets predictions, and writes to a "predictions.txt" file before you submit your code.

1.5 Use PCA and Logistic Regression on the same task and compare and discuss

Principal component analysis allows us to reduce the dimensions of our data while retaining as much variance as possible, thus retaining information to separate our data points. Scikit learn provides multiple ways to perform dimension reduction with PCA. For this problem, sklearn.decomposition.PCA is recommended.

We will use the same fashionMNIST dataset for this question.

What to Submit:

- Please select three different values of PCs(the number of principal components) you choose for PCA and present your prediction error results when using the Scikit logistic regression classifier. Please draw a figure showing how the prediction error changes when varying the number of PCs? Please also include the error discussions about the logistic regression without PCA in the discussion and in the figure.
- Please compare results with the MLP and CNN results you have obtained above. A table summarizing all classifiers you have tried is highly recommended.
- The minima requirement for this coding subunit requires your discussion of running the LogRegression(LoG) and PCA+LoG on the datasets.

1.6 Tips

Accuracy: Your MLP validation accuracy should be at least 80% and your CNN validation accuracy should be at least 85%. High 80s or 90s is very possible for both models.

Training time: Training time can vary wildly with model complexity/architecture, number of epochs, batch size, and system specs. You should expect it to take at least 5 minutes to train a single model (MLP or CNN). It wouldn't be unusual for training to take more than 20 minutes. **Start early** to make sure you have enough time to train your models.

Speeding up training: A few suggestions (certainly not exhaustive):

- If you have a (CUDA-enabled) NVIDIA GPU, make sure the TensorFlow backend is using it.
- Increase the batch size, which could increase CPU/GPU utilization.

- Simplify your models by using fewer layers or fewer units per layer (you don't necessarily need an extremely deep model to get high accuracy).
- Check the training loss numbers as your model trains. If they aren't noticeably decreasing (or are even increasing), halt training and make changes.
- If your model (MLP or CNN) is on the right track, it'll probably have an accuracy of more than 50% after one epoch.
- Many techniques that improve accuracy can also speed up training by allowing you to train for fewer epochs.

Improving accuracy: Many, many options, but a few brief pointers:

- Try using Dropout layers, a regularization technique.
- Try an "adaptive" optimizer like Adam, using momentum, changing the learning rate, using Nesterov momentum, etc. Check out the Keras optimizers page. Or checkout the FastAI optimizers page.
- Plot the train and validation loss against epochs to check for overfitting.
- Make sure your architecture makes sense.

1.7 Evaluation

To get full credit:

1. Include each of the items under the "what to submit" sections above.
2. Submit a file called "fashion.py" containing the code for your MLP, CNN, LG+PCA models. We will run this file from the command line using the command:

   python3 fashion.py path/to/fashion_train.csv path/to/fashion_test.csv

   When it runs, it should train your best performing model (just one model–if your best performing model was an MLP have it run that by default. Otherwise, have it run your best performing CNN). It should generate predictions for the fashion_test.csv file and write the predictions to a file called predictions.txt. There should be one predicted label per line. The labels should be numbers in {0, 1, ..., 9}. The predictions.txt file should be created in the same directory as fashion.py.
3. When we run your code, it does not need to generate anything besides the predictions.txt file (don't output any plots or visualizations)
4. **Update: Please Submit your predictions.txt file in addition to the code and writeup. We will grade it in case we have problems running your code or your codes takes too long to generate the output / to save time grading.**

1.8 FAQ:

- Since the labels of the test file are fake, you surely should not use results from zip.test to discuss the performance.
- - Q: can we use predefined functions like logistic regression, or cross validation from scikit-learn?
  - A: Yes.
- - Q: how many classifiers should be made, and how to deal with using the best one but including all of them?
  - A: You need to submit all the codes about the methods and model selection of the methods you tried. And you need to make sure the main function providing the interface for TA to run as the best one that you make!

  - A: please discuss the results of what you have tried in the written part of the submission.

- - Q: Must we implement all the models suggests by the skeleton code?
  - A: No. The template is just a recommendation what you can try.

- - Q: Is testing data being released separately and if so when? - Q: If not, will you simply run the scripts as stated in the homework instructions? (without the parameter for model selection, students just only include the one they want run).
  - A. The feature part of the testing data has been included in the data. We will release its labels when we release the result key of HW3.
  - A: Yes. We will simply run your submitted code as stated in the instruction.

Congratulations ! You have implemented a state-of-the-art machine-learning toolset for an important image labeling task !