Classifying clothes with Fashion-MNIST Dataset

Akari Ishikawa
163282
a163282@dac.unicamp.br

José Carlos Vasques Moreira
176596
j176596@dac.unicamp.br

I. Introduction

In the present project, we explore the classification problem predicting image classes using different classification methods to observe how they behavior and performance.

The idea of classification methods is to, given a training set with features and a class (or label), learn how to label new cases given their features. In this project, we will implement three methods:

- **Logistic Regression:** The model is in the form of $h(x) = g(\theta x)$, where $h(x)$ gives a probability (0 to 1) of x belonging to class 1, g is an activation function (e.g. Sigmoid), and the objective is to find the parameters θ that gives us the best accuracy.
- **Softmax Regression:** The hypothesis function is the same as Logistic Regression but the activation function is now the Softmax.
- **Neural Network:** The model is a stack of layers in which the first one is the input (x) and the last one is the output (e.g. probabilities for each class). In the middle, we can add *hidden layers*, in the form of $h(x) = g(\theta x)$, where x is the activation of the last layer.

We can note that all these methods are quite similar. Even though Logistic and Softmax Regression are not considered neural networks, we can adapt them to be computed as a neural network model. This way, we could build a platform that trains and predicts neural network models that was useful for all experiments. This platform is basically composed by two classes:

- **Layer Class:** Contains as attributes the activation, the weights matrix and the bias.
- **Neural Network Class:** Contains as attributes arrays of Layers, Activation and Deactivation functions for each layer and arrays with training and validation loss, used afterwards for plotting. Holds the functions necessary for training the network (Feed forward and back propagate), and the training and plotting functions.

Using these classes, we built and trained the following models to experiment with:

- **One vs All classifier:** A classifier based on training 10 different logistic regressions, each giving the probability of the example belonging to a particular class, and taking the result to be whichever is the most likely. Simple idea, but very costly to train and doesn't show particularly good results.
- **Softmax regressor:** A single regression model, that uses a softmax function in order to classify between multiple classes. Compared to the One vs All classifier, it is much cheaper, computationally speaking, to train and also puts out a higher accuracy.
- **Neural network with one hidden layer:** A basic neural network, comprised of input, hidden and output layers. While not as cheap to train as the Softamx regressor, it gives out a sizeable difference in accuracy (An increase of roughly 4% for the validation set)
- **Neural network with two hidden layers:** Same principle of a Neural Network with a single hidden layer, with idea that a more complex model would be able to extract more out of the features, to better distinguish between the few classes that still eluded our previous models

This report is organized as follows: In the Data Preprocessing section, we briefly describe the dataset and how we modeled the data before starting work on the problem. In the experiments section, we describe our analyses and the results. In the testing section, we apply the best model obtained in experiments section in the test set and the results.

II. Data preprocessing

The dataset explored is Fashion-MNIST, a collection of 60000 images divided in 50000 training and 10000 testing examples. Each image is a single piece of clothe in 28x28 grey scale, belonging to one of the following 10 classes: T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag or Ankle Boot.

Before starting the analyses, we first opened the training-set csv file, originally with 60000 examples, shuffled it, split it and saved it again in four csv files containing, separately, the training set features (with 50000 examples), the training set labels, the validation set features (with 10000 examples) and the validation set labels.

When working with neural networks, it is important to perform data normalization to avoid the vanishing gradients problem. In this project, we calculated the mean m and standard deviation std of each feature of the training set and then normalized all sets (training, validation and test set) subtracting m from their features and dividing them by std.

All analyses in the Experiments section below were performed with the training and validation sets we uploaded to Google Drive (https://drive.google.com/open?id=1saJ5A0Noc1qKNM-8YOBSc-b9dsAm0SLS) and need to be downloaded to run the experiments.

III. Experiments

A. Grid Search

In our experiments, we added the L2 regularization method to avoid overfitting. In order to find the best learning rate and L2 regularization lambda, we performed a grid search for each analysis.

The grid search split the training set (not including the validation set) in randomly sampled subsets of training and validation sets with a proportion of $7/3$. With these sets, we trained a neural network for 5 epochs with all the combinations of lambdas: $1e-1$, $1e-2$, $1e-3$, $1e-4$ and learning rates: $2e-3$, $2e-4$, $2e-5$, $2e-6$, $2e-7$. We performed a training/prediction twice for each combination and picked the parameters that produced the lowest average loss.

In order to accelerate the training process, we used mini-batch gradient descent with a batch size of 256 for all training sets. In validation, we kept the full-size of the set for forward propagation. In the following training loss decay plots, we showed the training loss vs epochs.

B. Logistic Regression

```
How to execute:
python3 exp1_onevsall.py
grid_search_epochs training_epochs
```

To perform this experiment, we used our platform to build, for each of the 10 classes, a neural network with an input layer with 784 neurons and an output layer with 1 neuron and sigmoid activation.

The sigmoid function (equation 1) transforms a linear value in a probability distribution for a binary class problem. So we applied the sigmoid in our $\theta^T x$ to transform the linear combination in a probability of x of belonging to determined class.

$$h_\theta(x) = \frac{1}{1 + e^-\theta^T x} \quad (1)$$

The class labels were also transformed into binary classes (e.g. in the model that learned how to classify class 5, the labels were 1 for examples from class 5 and 0 otherwise). Unfortunately, due to the way the grid search function works, and the needs of the One vs All classifier to transform the dataset, this model became unviable due to the severe computation cost associated with it.

In this experiment, most of the cost came from the grid search, which made the whole process take about 3 hours, since grid-search had to evaluate each learning rate against each regularization lambda in all 10 models for each iteration. Also, since we were training 10 models together, it generated fluctuations in the training curve, as seen in figure 1.

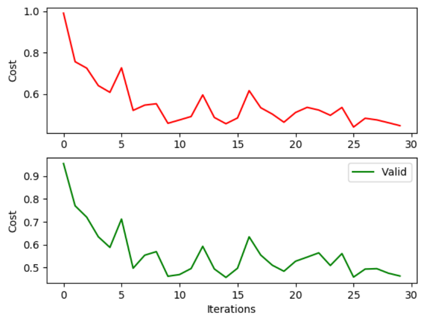

Figure 1. Training Logistic Regression for 30 epochs with batch size = 256, learning rate = 0.002, lambda = 0.1

The grid-search calculated the best learning rate as 0.002 and the best regularization lambda as 0.1, that led to a training accuracy of 82.94% and a validation accuracy of 81%. The final loss obtained was 0.4634.

C. Softmax Regression

```
How to execute:
python3 exp2_softmax_regression.py
grid_search_epochs training_epochs
```

To represent a logistic regression model, we built a neural network using our training platform with two layers: an input with 784 neurons and an output layer with 10 neurons with softmax activation function.

The softmax function (equation 2), as the sigmoid (equation 1), maps a linear function to a probability distribution, however, it is designed for a multi-class problem. So the softmax function transforms our $\theta^T x$ into 10 scores, one for each class i, representing the probability of x of belonging to class i. The label predicted is the class with the highest score.

$$p(C_k|\mathbf{x}) = \frac{e^{\ a_k(\mathbf{x})}}{\sum_{j=1}^{K} e^{\ a_j(\mathbf{x})}} \quad (2)$$

After performing grid search, we found that the best hyper-parameters were learning rate = 0.0002 and lambda = 0.0001. We then trained in the real training/validation set (Fig. 2) and obtained an accuracy of 86.97% in the training set and 85.52% in the validation set. The final loss in the validation set was 0.43.

As we can see, we have good initial results with one of the simpler methods. We also have a reasonable estimation for how many epochs we should use later on to avoid overfitting.

D. Neural Network with one hidden layer

```
How to execute:
python3 exp3_one_hiddenlayer.py
grid_search_epochs training_epochs
```

This neural network architecture is comprised of an input layer with 784 neurons, a hidden layer with 342 neurons and

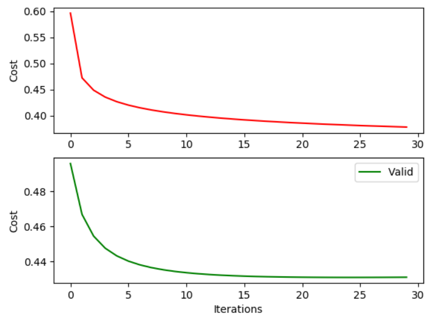

Figure 2. Training Softmax Regression for 30 epochs with batch size = 256, learning rate = 0.0002, lambda = 0.0001

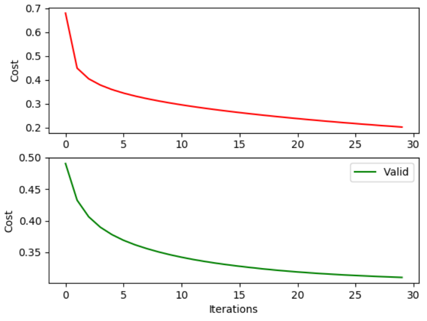

Figure 3. Training Neural Network with one hidden layer for 30 epochs with batch size = 256, learning rate = 0.0002, lambda = 0.0001

342 biases with ReLU activation function and an output layer with 10 neurons with softmax activation (equation 2).

The ReLU function (equation 3) has a low computation cost and eliminates any possible negative activations and does not limit the activation in a range, as sigmoid does, contributing to reduce the problem of vanishing-gradients. Also, ReLU is one of the most popular activation functions in deep learning literature. Following other works, we decided to use ReLU in our project.

$$ReLU = max(0, x) \tag{3}$$

Based on the last experiment, we decided to train this model for 30 epochs. As suggested by grid search we used learning rate = 0.0002 and lambda = 0.0001. With these parameters, we could achieve an accuracy of 93.58% in training and 88.99% in validation and a final loss of 0.31.The training/validation loss decay can be seen in figure 3.

Despite the good accuracies, observing the loss decay, it seemed that the model was still converging, so we tried to run the experiment again but with 50 epochs. However, the accuracy barely changed (96.32% in training and 89.4% in validation) showing that, in fact, the model converged around 30 epochs (Fig. 3). Besides, this small difference in the evaluation metrics are not worth the increased amount of time required to train for 50 epochs.

E. Neural Network with two hidden layers

How to execute:
```
python3 exp4_two_hidden_layer.py
grid_search_epochs training_epochs
```

This architecture is comprised of one input layer with 784 neurons, followed by a hidden layer with 342 neurons with ReLU activation, followed by another hidden layer with 180 neurons with ReLU, and a final output layer with 10 neurons with softmax activation.

Contradicting the good results produced with 30 epochs of training in the previous experiments, the neural network with two hidden layers converged with four epochs and then started

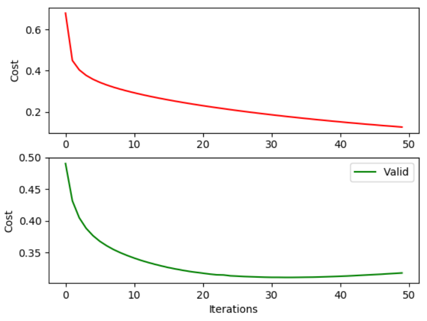

Figure 4. Training Neural Network with one hidden layer for 50 epochs.

to diverge. (Fig. 4). Since we were dealing with a deeper model, we decided to run with a severely reduced epochs: 2 epochs for grid search and 5 for training (Fig. 5).

With 5 epochs, we achieved a validation loss of 0.38 and an accuracy of 86.7% in training and 86.5% and validation, a performance competitive with the 30 epochs of the softmax regression and neural network with one hidden layer, although the neural network with two hidden layers took approximately the same amount of time to train as the one hidden layer net did.

F. Neural Network with two hidden layers with leaky relu

How to execute:
```
python3 exp5_twolayers_activations.py
grid_search_epochs training_epochs
```

Here, the model was exactly the same as the model of experiment 4. One input layer, two hidden layers and a final output layer. However, we changed the ReLU activation functions from both hidden layers to Leaky ReLU.

Leaky ReLU (equation 4) is very similar to ReLU (equation 3), however it does not eliminate the negative values. Instead, it multiplies them by a small α. In our case, we used $\alpha = 0.01$.

$$Leaky\_ReLU = max(0, x) - \alpha * max(0, -x) \tag{4}$$

It showed a validation accuracy of 88.5%, a small improvement compared to the 86.5% of the network with ReLUs. This

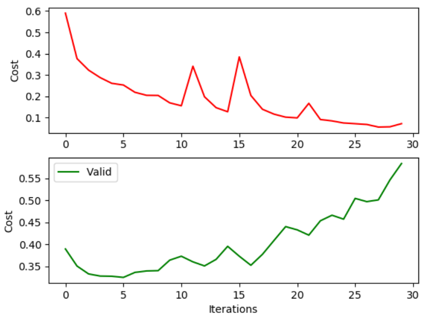

Figure 5. Training Neural Network with two hidden layers for 30 epochs with batch size = 256, learning rate = 0.0002, lambda = 0.001

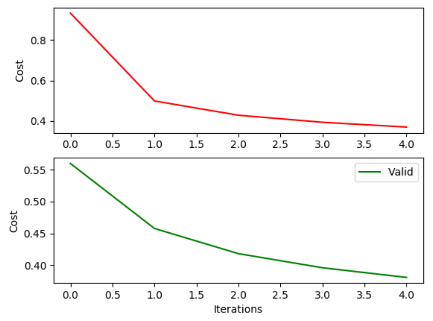

Figure 6. Training Neural Network with two hidden layers for 5 epochs with batch size = 256, learning rate = 0.0002, lambda = 0.001

behavior is already expected. According to [1], Leaky ReLU has a slight improvement with small α and it tends to perform better as α is increased.

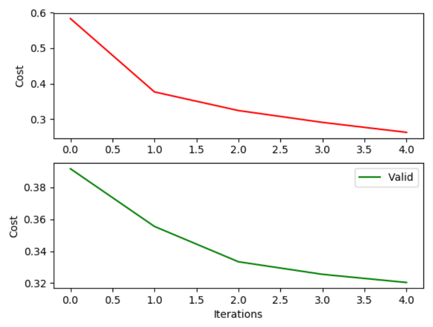

Figure 7. Training Neural Network with two hidden layers with leaky relu activation function for 5 epochs with batch size = 256, learning rate = 0.002, lambda = 0.001

IV. Testing

How to execute:
```
python3 exp6_testing.py
best_model.npy
```
Where best_model.npy is the numpy file with the trained neural network you want to use for testing.

As observed in the last experiments, the best model we obtained was a neural network with one hidden layer, with a ReLU activation in the hidden layer and a softmax in the output layer. In our platform, for each model trained, we saved a *.npy* file with all the architecture and the parameters of the model. This way, we evaluated the performance of the model in the specified test set and obtained a loss of 0.3022, along with an accuracy of 89.17%, a measure higher than some obtained in the experiments.

Observing the confusion matrix in figure 8, we noticed that main source of mistakes were the predictions regarding class 6 (shirt). However, analyzing the errors, the model confused class 6 (shirt), with classes 0 (T-shirt), 2 (Pullover) and 4 (Coat), all very similar clothes.

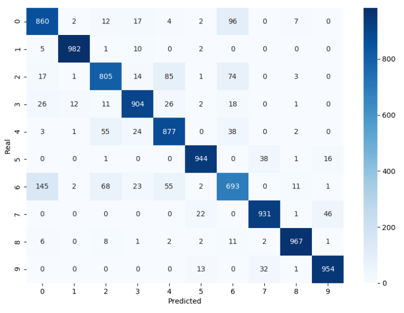

Figure 8. Confusion matrix obtained in the test set using a neural network with one hidden layer.

V. Conclusions

In the end, we achieved good results, achieving close to the same accuracy obtained by others' experiments. Our main source of confusion came from a class (Class 6) that is visually similar enough to other classes that a human being would have trouble telling them apart. This shows our models to be empirically correct, being able to predict the results as accurately as any other for this problem.

References

[1] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.