Joshua Whalen
AMATH 482
March 12, 2020

HW5: Neural Networks for Classifying Fashion MNIST

Abstract

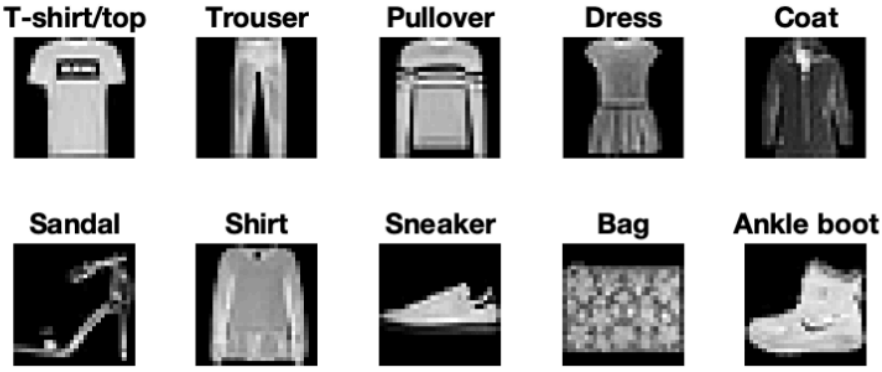
In this assignment, I was tasked with generating multiple neural networks for classifying the fashion MNIST data set, and comparing the efficacy of each network. The fashion MNIST data set includes 60,000 images of clothing items for training and 10,000 images of clothing for testing. The goal of the neural net is to correctly classify the images into one of the ten categories of clothing. I adjusted the depth and width of each network, compared different activation functions, learning rates, filter sizes, etc. Ultimately, the main differences were found between three models which will be discussed in further detail throughout this paper. The first only contained fully connected layers. The second contained fully connected layers with convolution and pooling. The last neural net contained fully connected layers, convolution, pooling, and normalization. Smaller changes in width, depth, and activation functions will also be discussed, but in less detail. The best classification accuracy was achieved by the third neural net with a validation accuracy of 91.3% and a test accuracy of 90.7%. The applications of machine learning are constantly expanding, and it is exciting to see where this technology will go next.

Introduction

In this section, I will give an overview of topics discussed throughout this paper. In previous assignments, I have shown methods of classifying music samples via SVD & LDA. However, those methods required finding a single (sometimes a couple) projection vector that maximized separation between the groups and then project new samples onto that vector to categorize. As we quickly noticed in those examples, finding a perfect projection vector that completely separated the samples is uncommon, and there was often overlap between the classes. This resulted in errors, and I had difficulty reaching a percent accuracy of 70% when separating only 3 groups.

Machine learning takes the LDA concept of weighting groups and sorting based on the weights, and expands it across multiple layers of weighting, looking at different portions of the sample, pooling the sample, normalizing it, etc. This method utilizes a far larger training pool and can categorize samples at a much finer detail. Machine learning has revolutionized the data processing world. Even simple machine learning models with only two layers can achieve classification accuracy far greater than LDA alone.

In this paper, I will be exploring and comparing three different neural nets while trying to classify the Fashion MNIST data set. This data set is composed of 70,000 28-by-28 photos of clothing, which are sorted into 10 groups. Examples of these photos and their corresponding groups are shown in figure 1. The first neural net I will be testing only contains fully connected layers and activation layers, without any convolution or pooling. The second neural net is a copy of the LeNet5 neural net, which is a famous neural net design that is still used to this day. Lastly, the third neural net is one based on the LeNet5, but includes different filter amounts, different activation layers, and normalization. Ultimately, all of these machine learning algorithms achieved a high level of categorization accuracy, but it is interesting to show what changes impact the neural net more or less.



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

Figure 1: This figure shows examples images and their label from the fashion MNIST data set.

Theoretical Background

Machine learning, when broken into its mathematical components, is very complex and difficult to see the bigger picture, which is why neural nets are often depicted in diagrams instead of equations. The general outline of a neural net involves some amount of data, a model, and a loss function. The overarching goal is to compare the data to the model. The model will take the weights generated from the loss function and combines them in the next layer. This process can be repeated multiple times per layer increases the width of the neural net, and multiple layers can stack in-between the input and output layers increasing the depth of the neural net (this is exemplified in figure 2).

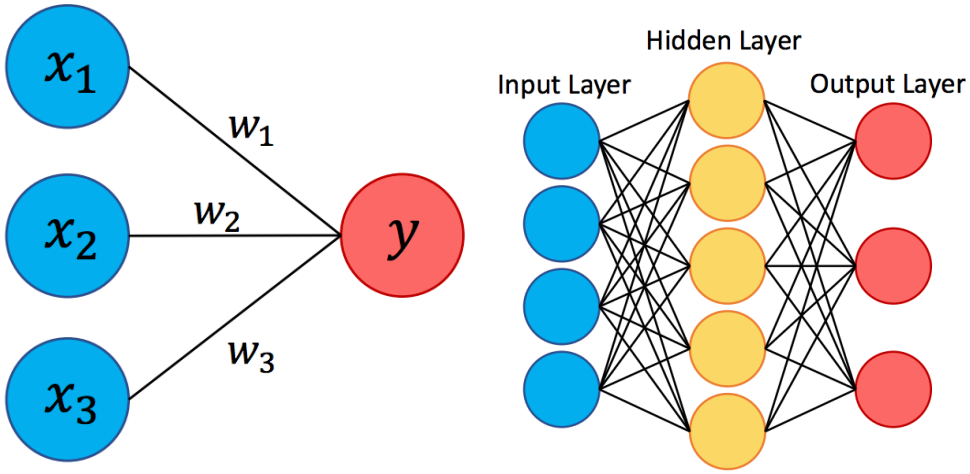
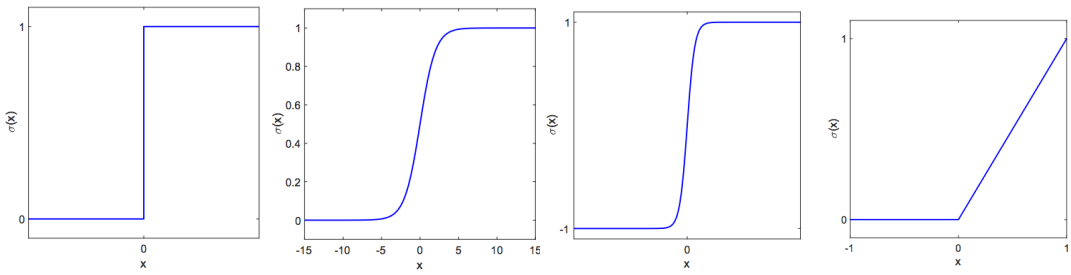


Figure 2: This diagram shows the general outline of a machine learning neural network.

In-between each layer in the neural net there is an activation function, which will generate a weight between neurons in each layer. This weight is based on a relationship between the data, the chosen activation function, and some loss function. The type of activation function will change based on what the machine learning algorithm is trying to achieve. For instance, when categorizing data, you are expecting steep jumps between the data signifying different classes in the data, which means you would want a model that has steep changes in slope like a step function. Or, if you were optimizing some structure to function relationship like thermostability of a protein, you would want a more continuous model without sharp corners. Some examples of activation functions used in this assignment are shown below in figure 3.



(a) Step function (b) Logistic function (a) Hyperbolic tangent (b) ReLU

Figure 3: This figure shows different activation functions used in neural networks.

Algorithm Implementation

Generating a machine learning algorithm in MATLAB is actually very simple. Most of the coding involves reshaping the data into a format that is accepted by the built in MATLAB functions. Once the data has been reshaped, it needs to be split between the training data, validation data, and test data. Generally speaking, the more training data the better as long as there is an adequate amount of validation and test data. The training data is what the model will learn from; the validation data is to confirm the model is not overfitting the training data, and is used to tune hyperparameters; and the test data is kept separate to make sure the hyperparameters are not tuned to only fit the validation data.

Once the data is reshaped and allocated into the three groups, it is time to set up the layers of the machine learning algorithm. MATLAB is a very user friendly application called "Deep Learning Designer." This allows the user to drag and drop layers of the neural network, connect them, and edit parameters, without having to worry about coding syntax. Also there is an additional feature that analyzes the network checking for errors or warnings before exporting it back to the MATLAB editor (the UI is shown in figure 4).

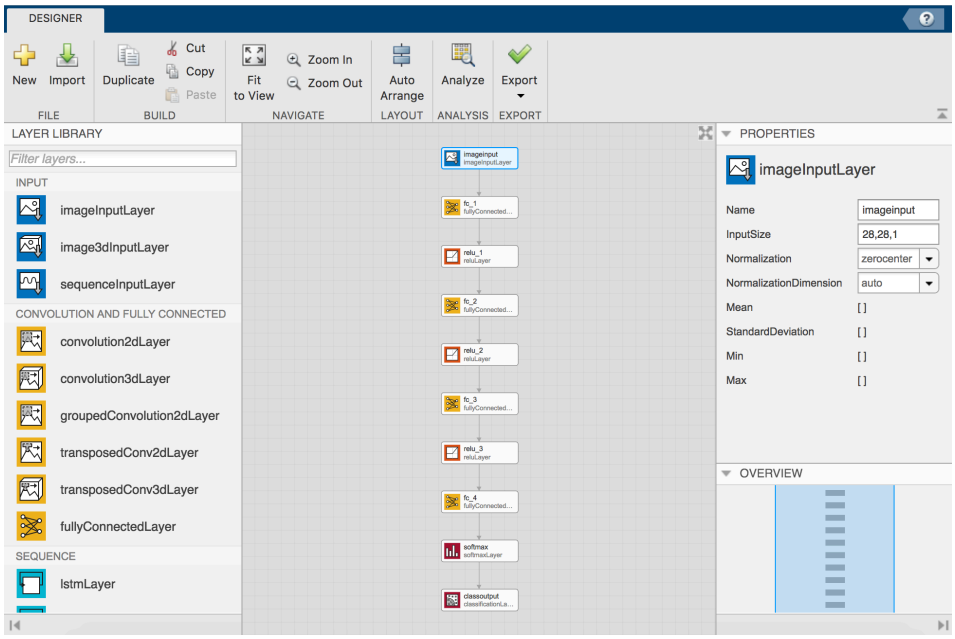


Figure 4: This figure shows the UI of the deep learning designer program built into MATLAB.

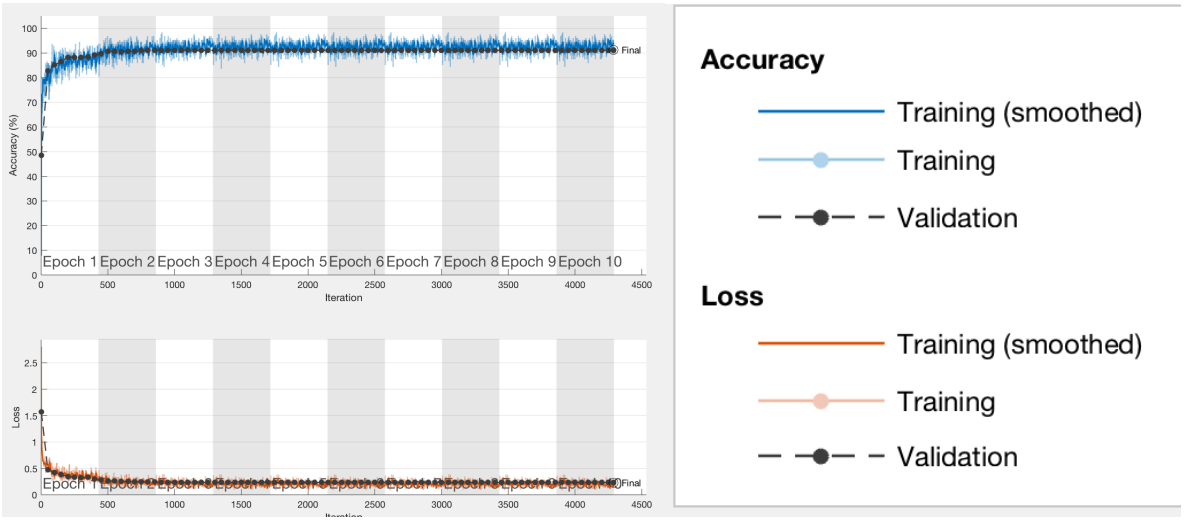


Figure 5: This figure shows an example of the graphs displayed when training a neural network.

After exporting the layers, the model is ready to be trained. There are many additional training options that can be optimized. For this assignment I used the built in 'adam' training design and modified it to only go to 10 epochs (this was the limit for my computer without basically going all night long), and I modified the training rate to starting higher at 0.015 and decrease by a factor of 0.1 every epoch. This helped increase the rate of the training without overfitting later in the training process. Once the training begins a screen will appear denoting the process of the training. The graph will look similar to the one depicted in figure 5. The top graph shows the overall accuracy of the model over the course of the training, and the bottom graph shows the decreasing loss function over the course of the training. It is important that the training and validation lines stay close because if the training line increases away from the validation, it shows the model is becoming over fit to the training data. The results of this training will be shown in the results section of this paper. The layers used in each of the three models is shown in table 1.

Model 1: FC Only	Model 2: LeNet5	Model 3: Custom
Input layer Fully connected(16) Relu layer Fully connected(32) Relu layer Fully connected(64) Relu layer Fully connected(10) Softmax layer Classifier Output	Input layer 5-by-5 Convolution(6) Tanh layer 2-by2 Average pool 5-by-5 Convolution(16) Tanh layer 2-by2 Average pool 5-by-5 Convolution(120) Tanh layer Fully connected(84) Tanh layer Fully connected(10) Softmax layer Classifier Output	Input layer 3-by-3 Convolution(16) Batch normalization layer Elu layer 2-by2 Max pool 3-by-3 Convolution(32) Batch normalization layer Elu layer 2-by2 Max pool 3-by-3 Convolution(64) Batch normalization layer Elu layer Fully connected(64) Batch normalization layer Elu layer Fully connected(10) Softmax layer Classifier Output

Table 1. This table shows the layers implemented in each of the models explored in this paper.

Results

Figure 6: This figure shows the test set confusion matrices for each of the three models. The bottom right is model 1, bottom left is model 2, and top center is model 3.

The test results for each of the three models are shown in the confusion matrices in figure 6. The top matrix is the third neural network model and has the highest test accuracy at 90.7%. The bottom left matrix represents the second model and has a test accuracy of 86.2%. The bottom right matric represents the first model and has a test accuracy of 84.9%.

By adding convolution layers between model to the neural network the accuracy jumped almost 2%. The convolution layers basically summarizes data in a given reading window and consolidating the amount of information in the original input layer. This is similar to how real brain neurons function and have been shown to improve the neural network of machine learning algorithms. Next, in my custom model, I chose to implement slightly smaller window sizes for the convolution and a max pooling instead of average. I did this because I believed there were fine details between the t-shirts and shirts that the algorithm may have been able to detect. I also implemented normalization because it's been shown to speed up computation time and control outliers better. Lastly, I changed the activation function layers from tanh to elu because elu has been shown to be more stable. Ultimately, the activation function did not results in significant change, but both normalizing and changing window size did and the test accuracy increased by 4.5%. I also tried adding deeper and wider layers, but this often resulted in both drastic computation time increase and neuron death.

Although the third model has the highest percent accuracy, it is far from the perfect model. Looking closer at the confusion matrix is seems the model is having the most trouble categorizing between 6 and 0. Looking back at figure 1, these two categories represent t-shirts/tops and shirts. Although in the figure the two categories look fairly distinct, many of the t-shirts/tops have longer sleeve lengths and many of the shirts have shorter sleeve length making it difficult to discern the correct categorization. Furthermore, other more complex machine learning models found on GitHub have claimed to achieved test accuracy ~95%. Therefore, although this model achieve an impresses accuracy above 90%, it is far from perfect.

Conclusion

In conclusion, this machine learning model was able to achieve a classification accuracy of 90.7% on the fashion MNIST data set. The model is implemented multiple techniques commonly used in machine learning including convolution, pooling, normalization, and fully connected layers. There are additional techniques such as random horizontal flips that were not explored in this exampled, but have also been shown to increase accuracy. The fashion MNIST data set is known to be a harder test set compared to the original MNIST data set. So I would be interested to see how well this algorithm functions in the original MNIST data set. This assignment exemplifies the complexity of optimizing a machine learning algorithm while also showing how easy the initial implementation can be.

Appendix A: Functions & Explanations

- Im2double: used to change unit8 format of images to double
- Reshape: changes the shape of a matrix
- Permute: changes the order of the rows/columns in a matrix
- Plotconfusion: plots a confusion matrix

Appendix B: Code

```
clear all; close all; clc;

load('fashion_mnist.mat')
%%
X_train = im2double(X_train);
X_test = im2double(X_test);

X_train = reshape(X_train,[60000 28 28 1]);
X_train = permute(X_train,[2 3 4 1]);

X_test = reshape(X_test,[10000 28 28 1]);
X_test = permute(X_test,[2 3 4 1]);

X_valid = X_train(:,:,:,1:5000);
X_train = X_train(:,:,:,5001:end);

y_valid = categorical(y_train(1:5000))';
y_train = categorical(y_train(5001:end))';
y_test = categorical(y_test)';

% Part 1
layers = [
    imageInputLayer([28 28 1],"Name","imageinput")
    fullyConnectedLayer(16)
    reluLayer
    fullyConnectedLayer(32)
    reluLayer
    fullyConnectedLayer(64)
    reluLayer
    fullyConnectedLayer(10)
    softmaxLayer("Name","softmax")
    classificationLayer("Name","classoutput")];

% Part 2
layers_LeNet5 = [
    imageInputLayer([28 28 1],"Name","imageinput")
    convolution2dLayer([5 5],6,"Name","conv_1","Padding","same")
    tanhLayer("Name","tanh_1")
    averagePooling2dLayer([2
2],"Name","avgpool2d_1","Padding","same","Stride",[2 2])
    convolution2dLayer([5 5],16,"Name","conv_2")
    tanhLayer("Name","tanh_3")
    averagePooling2dLayer([2
2],"Name","avgpool2d_2","Padding","same","Stride",[2 2])
    convolution2dLayer([5 5],120,"Name","conv_3")
    tanhLayer("Name","tanh_2")
    fullyConnectedLayer(84,"Name","fc_1")
    tanhLayer("Name","tanh_4")
```

```
    fullyConnectedLayer(10,"Name","fc_2")
    softmaxLayer("Name","softmax")
    classificationLayer("Name","classoutput")];

layers_LeNet5mut = [
    imageInputLayer([28 28 1],"Name","imageinput")
    convolution2dLayer([3 3],16,"Name","conv_1","Padding","same")
    batchNormalizationLayer("Name","batchnorm_1")
    eluLayer(1,"Name","elu_1")
    maxPooling2dLayer([2 2],"Name","maxpool_2","Padding","same","Stride",[2
2])
    convolution2dLayer([3 3],32,"Name","conv_2","Padding","same")
    batchNormalizationLayer("Name","batchnorm_2")
    eluLayer(1,"Name","elu_2")
    maxPooling2dLayer([2 2],"Name","maxpool_1","Padding","same","Stride",[2
2])
    convolution2dLayer([3 3],64,"Name","conv_3","Padding","same")
    batchNormalizationLayer("Name","batchnorm_4")
    eluLayer(1,"Name","elu_3")
    fullyConnectedLayer(64,"Name","fc_2")
    batchNormalizationLayer("Name","batchnorm_3")
    eluLayer(1,"Name","elu_4")
    fullyConnectedLayer(10,"Name","fc_1")
    softmaxLayer("Name","softmax")
    classificationLayer("Name","classoutput")];
%%
options = trainingOptions('adam', ...
    'MaxEpochs',10,...
    'InitialLearnRate',15e-3, ...
    'LearnRateSchedule','piecewise',...
    'LearnRateDropPeriod',1,...
    'LearnRateDropFactor',0.1,...
    'L2Regularization',1e-4, ...
    'ValidationData',{X_valid,y_valid}, ...
    'Verbose',false, ...
    'Plots','training-progress');

net = trainNetwork(X_train,y_train,layers_LeNet5,options);
%%
figure(1)
y_pred = classify(net,X_valid);
plotconfusion(y_valid,y_pred)

%% Test classifier
figure(2)
y_pred = classify(net,X_test);
plotconfusion(y_test,y_pred)
```