Fashionable Neural Networks

Matthew Dees, Julius Aguma, and James
University of California, Irvine
{mdees, <Ceasar e-mail>, <James e-mail>}@uci.edu

***Abstract*—We trained a convolutional neural network and a feedforward neural network on the Fashion-MNIST dataset using the Keras API in the Tensorflow library. The goal of this project was to explore the differences in performance between the convolutional and feedforward neural networks on a robust image classification dataset. We found that the convolutional neural network generally performed better than the feedforward neural network under our testing criteria.**

***Index Terms*—fashion-mnist, machine learning, neural networks**

I. Introduction

Due to the growing popularity of computer vision and open-source machine learning tools, we decided to tackle an image classification problem by training two commonly used neural networks, convolutional (CNN) and feedforward (FNN), using the popular Python library Tensorfow. Of the choices provided, the Fashion-MNIST seemed to be the most robust image classification dataset, as it was designed as a drop-in replacement for the popular MNIST dataset. The Fashion-MINST dataset is supposed to be more challenging than the MNIST dataset, as the authors felt the original dataset was two easy to perform well on. Our project consists of a data visualization phase, modeling phase, and evaluation phase. In the data visualization phase we visualize the data in order to gain insights into potential modeling parameters and required normalizations. The modeling phase consists of the training of the CNN and FNN. The evaluation phase compares the performance of the two neural networks on the test data provided by the Fashion-MNIST [1] dataset. Due to the large amount of time required to train these neural networks, we decided to tackle the overfitting problem by adjusting the layers of the networks themselves, rather than by creating an ensemble of different networks. We measured the ratio of correct test image classifications to total number of test images as a way of measuring the strength of the each neural network.

II. Dataset

We decided to use the Fashion-MNIST dataset created by Zalando Research due to the excellent documentation on Github and plethora of utilities available within their Github repository. The dataset consists of a training set and validation set. The training set contains 60,000 images while the validation set contains 10,000 images. Each image is a 28×28 grayscale image (784 total pixels). The training set is of the form $X = Y$, where X is a $60,000 \times 784$ matrix and Y is a $60,000 \times 1$ matrix. Each row in X corresponds to a different image and each column corresponds to a pixel of that image (used as features). The Y matrix is $60,000 \times 1$ matrix where each value is a classification for the corresponding row (image) in the X matrix. Each classification is an integer value which maps to one of 10 different clothing classes.

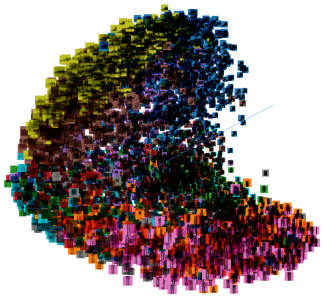

Fig. 1. Tensorboard visualization of the Fashion-MNIST dataset.

To better understand our dataset, we decided to visualize it using Tensorboard [2] and Matplotlib [3]. Our implementation for visualizing the data is located in the **visualization.py** file in the top-level directory. We opted for a Principal Component Analysis (PCA) to get a better idea of what clusters may exist amongst the data. We chose PCA because the data has a relatively high number of features (784), which is not easy to visualize. With PCA, we have access to the features the preserve the most variance in the data, and may offer some insight into how it can be organized. When loaded into the Tensorboard visualization suite we were able to observe images like in Figure 1. Once colored, it was evident that the data actually formed reasonable clusters. This led us to believe that data was reasonably separable, so we *should* see fairly high classification rates with a model using the correct features. Ideally, we would learn which features these are using the neural networks. Using Matplotlib, we were also able to visualize a single image as shown in Figure 2. Upon visualizing this image, it was obvious that we should normalize the data. Since each pixel is a value between 0 and 255, we decided to normalize the data by dividing each pixel by 255. This proved extremely helpful in getting better results from our neural networks.

III. Neural Network Configuration

Our neural networks shared many of the same characteristics, as the only thing that needed to change between the CNN and FNN were the hidden layers between the input and output layers.

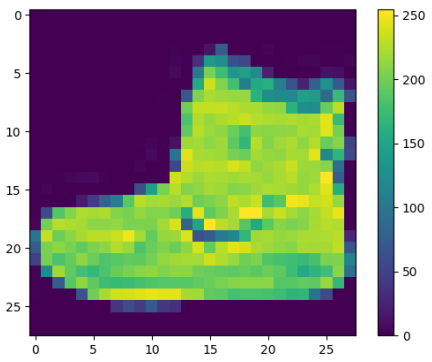

Fig. 2. Image 0 from the Fashion-MNIST dataset visualized using a color map.

A. Activation Function

Within these hidden layers we used the ReLU [4] activation function.

$$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$$

The ReLU function works well because it is not expensive to compute, it doesn't contain any plateaus (so convergence is relatively quick), and it is sparsely activated. Sparse activation is a good thing in this scenario because it will increase the chance of a node not firing, and hence simulating a dropout effect to reduce overfitting.

B. Loss Function

We decided to use the categorical cross-entropy loss function within our neural networks.

$$L(y, \hat{y}) = -\sum_{j=0}^{M}\sum_{i=0}^{N}(y_{ij} \log \hat{y}_{ij})$$

We wanted to use a loss function that returned a low value when the true value matched the value guessed by the model and a high value when the guess was not correct. In the categorical cross-entropy loss function there will only be contribution from a given term y_i when $y_i \approx 1$, indicating a classification chosen by the algorithm. This classification will be correct when $\hat{y_i} \approx 1$, which will cause the overall term to approach zero (since $\log 1 = 0$). This loss function works very well for classification problems with multiple classes in general.

C. Optimization Algorithm

For our optimization algorithm we decided to go with Adam [5] because we wanted to try out one of the newer gradient descent variants produced by modern research. Adam was published in ICLR in 2015, and showed very promising training rates which we hoped to leverage in our project. The Adam algorithm can be used in Tensorflow applications through `tensorflow.keras.optimizers.Adam`.

D. Metric

To evaluate the testing and training performance of our neural networks we decided to use the typical classification accuracy formula:

$$A = \frac{1}{M}\sum_{i=0}^{M}(y_i = \hat{y_i})$$

In the above formula, M is the number of images in the training/test set that is being evaluated. The condition inside of the summation is used as shorthand to indicate the number of correct guesses. Essentially, this formula represents the number of correct guesses over the total number of items. We used this metric because we were familiar with it from class and there was support for it in Tensorflow.

IV. Convolutional Neural Network

V. Feedforward Neural Network

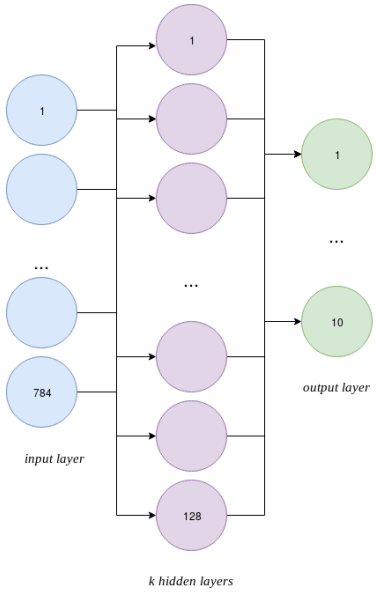

Fig. 3. Diagram of our feedforward neural network design. The singular input layer contains 784 data points. There are k hidden layers of 128 nodes each, and one output layer consisting of 10 data points

Our feedforward neural network (FNN) took a 784×1 vector as input, representing the entire 28×28 image. Each pixel was normalized using the following function as determined in the data visualization stage:

$$norm(x_i) = \frac{x_i}{255}$$

We thought the best way to fairly represent the feedforward neural network was to vary the capacity (variable k in Figure 5), so in our experiments we tested with k values of 1, 6, and 12. In testing these values we held the number of epochs at 20 to keep the training time low. With more time and better hardware we would have liked to increase the number of epochs to around 80.

VI. Evaluation

A. Convolutional Neural Network

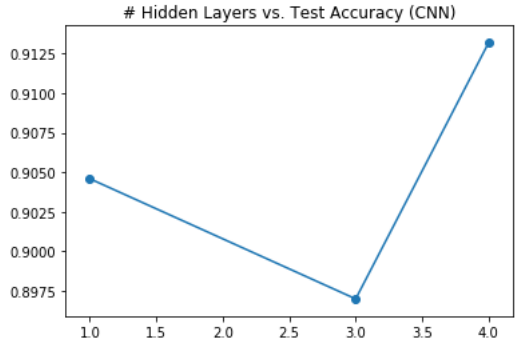

Fig. 4. Evaluation of the convolutional neural network with varying numbers of hidden layers.

B. Feedforward Neural Network

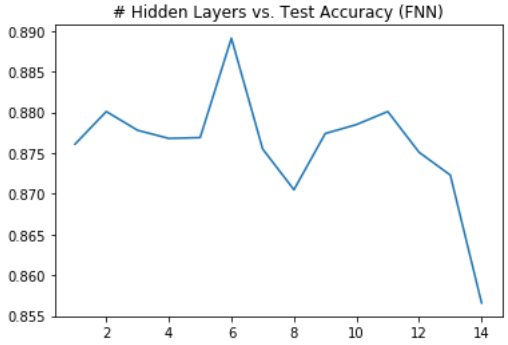

Fig. 5. Evaluation of the feedforward neural network with varying numbers of hidden layers.

VII. Conclusion

VIII. Contributions

A. Matt

Matt was responsible for organizing the Github repository, performing the data visualization phase (visualization.py), re-running experiments for the report, tuning hyperparameters for the FNN and CNN (experiments.ipynb), and for writing the majority of the report.

B. Julius

Julius was responsible for writing the first iteration of the experiment code (cNN.ipynb and fNN.ipynb) and writing preliminary results to the report.

C. James

James was responsible for investigating bootstrapping and cross-validation techniques for use in the neural networks.

References

[1] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017.

[2] Tensorboard. https://github.com/tensorflow/tensorboard.

[3] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[4] Vinod Nair and Geoffrey E. Hinton. *Rectified linear units improve restricted boltzmann machines*. Omnipress, Jun 2010.

[5] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv*, Dec 2014.