COMP 551 Project 3 Report: Modified Digits

Team Name on Kaggle: lmy
Liqiang Ding 260457392 Yue Xia 260556275 Marc-Antoine Dumais 260633946
{liqiang.ding, yue.xia, marc-antoine.dumais}@mail.mcgill.ca

I. Introduction

The purpose of this project is to build a classifier which can take an image of 60x60 pixels which contains two handwritten digits on a patterned background and determine the sum of those two digits. For training and cross-validation purposes we were given a set of 100,000 images and the sum of their two digits. We built three classifiers to solve this problem: one using Logistic Regression and Multi-Layer Perceptron (MLP), one using a Feedforward Neural Network, and one using a Convolutional Neural Network.

II. Related work

The automatic recognition of handwritten digits has been a problem of importance for a long time now, but the techniques used have not always been so successful. Feeding raw image data to linear classifiers works up to a certain point, but its accuracy is far from that of a human.

Various methods were invented to extract important features from images, such as directly calculating keypoints descriptors (eg: ORB, SURF, SIFT, etc.), using principal component analysis, or using an autoencoder. These extracted features can then be fed to any classifier, and often leads to an increase in accuracy, as the dimensionality of the input is greatly reduced from that of a raw pixel matrix. Convolutional neural networks represent the next big step in improving the accuracy of the predictions. By learning filters to detect patterns in images, they automatically learn what each digit looks like, similarly to a human brain, and achieve comparable accuracy.

III. Problem representation

There are some pre-processing steps that can be done on the raw dataset, in order to remove the noise and improve the computational efficiency during training and testing.

Initially, we chose to only keep the white letters and eliminate everything else by assigning any pixel values to 0 if the particular pixel value is not equal to 255. Through this step, a great amount of unnecessary computation can be successfully saved. (feature #1,size = 3600)

Next, we took a further step to simplify the computation by converting each image to complete binary images. Instead of keeping 255 for any white pixels, we assigned them with the value of 1. Without performing this step, the output of the sigmoid function would be equal to 0 if the input is a very large number. This will further stop the training process and therefore negatively affect the testing results. By reducing the pixel value to 1 and 0, the chance of getting this kind of errors can be brought to the lowest. (feature #2, size = 3600)

Besides, the computation and training time for neural network is notoriously large. Therefore, we also decided to shrink the dimensions of the training and testing images by half in order to improve the speed. The images are still recognizable after this dimension reduction step, but the computation speed is increased fourfold. (feature #3, size = 900)

Since the image is already in binary, we can further crop the image by removing all empty rows and columns. After that, the resulting images are resized to the same size (20x20 pixels). The cropping is successful for most of the cases (around 80 percent). However some pictures have backgrounds lighter than the digits and some background patterns have the same color as the digits. Thus a new method was used to make the pictures binary. We repetitively applied GaussianBlur [1] and adaptiveThreshold and used a heuristic to check if the output is as desired. Finally the cropping successful rate was increased to around 0.998. After the cropping, the number of features is reduced to 400.(feature #4,size= 400).

Other than using raw pixels data, different features like histogram of oriented gradient (HOG) [2] were tried. An autoencoder implementation was also found and tested [5] with various methods, but was eventually discarded as it didn't result in any increase in accuracy.

IV. Algorithm Selection and Implementation

4.1 Logistic Regression and Multi-Layer Perceptron

We used the Logistic Regression and MLP Classifier implementations from Scikit-learn [2]. All the results are tested by cross validation, 10 percent of the dataset is used as a testing set.

First, we worked on logistic regression, (feature #2, #3, #4) and HOG features of (feature #4) were tested. But it turned out that the accuracy using all of these features was very low, we moved on to use Multi-Layer Perceptron.

To simplify the problem, we assume that each hidden layer has the same number of neurons with constant learning rate of 0.0001 and a batch size of 200 examples. Now we only need to find the number of hidden layer (l) and the size of each hidden layer (w). (feature #4) is employed as it gave best result in logistic regression and a greedy approach is used to find the best combination of l and w. We started at (l=3,w=100), kept increasing w and l until the validation error began to increase. Finally the greedy approach terminates at (l=8,w=500). We also tried other features like (feature #2) and (feature #3), but we did not focus too much on them as the results were not good.

4.2 Feedforward Neural Network

Several hyper-parameters need to be carefully configured for this neural network to work correctly and efficiently. The first one to tweak is the learning rate. If this value is set too high, the neural network is too aggressive to learn, which results in updating weights unstably and oscillating around the optimal value. In some extreme cases, the trained output constantly fluctuates during the entire epochs, and weights are never trained correctly.

Another hyper-parameter to consider is the number of iterations. This value determines how many times to train or update each weights based on the errors between predicted outputs and true outputs. A large number of iterations is necessary if the amount of weight updates towards the correct outputs is small. This makes sure the network has sufficient steps to train each weight without worrying about each single training step finishing too early. On the other hand, if there are no errors between predicted output and the true category for a training example after a small number of iterations, the neural network can safely assume all of the weights received sufficient amount of updating, and therefore jump to the next example. A small number of iterations can also be useful in order to speed up the training time, since the number of iterations directly increases the total training time.

Next, the type of output space needs to be correctly determined. Either classification or regression approach can be used in this case. The advantage of the classification is that it clearly indicates the specific category that an example belongs to. However, there are multiple predicted outputs possibly returned in some situations, especially when the training is not sufficient. Without appropriately incorporating the correct learning rate and number of iterations, this classification approach potentially introduces more complexity and instability to the prediction model. For the purpose of simplicity, the regression approach is applied instead. The positive side of regression is that it is able to return only one single value. By comparing it with the true value, it knows immediately if the prediction is correct or not, unlike the multiple possible answers from classification. However, its drawback is that it is hard to correctly round or truncate the output. In some situations, the prediction is very close to the actual value, whereas it gets incorrectly rounded up or down due to this issue (please read Test and Validation section in detail).

The most important hyper-parameter is the number of hidden layers and their associated number of neurons. In theory, if the number of hidden layers is two or above, any nonlinear functions can be modeled. However, the complexity of our manual implementation increases exponentially, mainly due to the increasing number of updating functions.

Furthermore, unlike the highly optimized implementation by the Theano packages utilizing the GPU to speed up the computation, our manual designed computation speed is very slow. If there are multiple layers and a high number of epochs, we estimate it would take a few weeks to fully train the full size of the dataset. This gets even worse when there are no powerful machines available. Overall, considering these facts, the number of hidden layer in our manual implementation is 1. Once the number of hidden layers is decided, we can start to determine the number of hidden neurons involved. This number simply indicates how general the model can represent each category. In our case, there are 900 input neurons (30*30 pixel for each image), 300 hidden neurons, and one output neuron. Adding some extra neurons could bring up the capability of generalization, whereas having insufficient neurons would make the model not cover all cases. There are no theories explicitly clarifying the relationship between the generalization capability and number of hidden neurons, but a larger number would improve the chances of accurate prediction.

4.3 Convolutional Neural Network

It is well known in the literature that convolutional neural networks (CNNs) can reach very high accuracy when working with image data, as they take into account the proximity of different pixels instead of looking at each pixel independently. For this reason, we decided to find an existing CNN implementation and modify it to work with our specific problem [4]. The implementation we found used the nolearn, Lasagne and Theano libraries to run highly optimized computations in parallel on the GPU.

One of the problems with CNNs is that the number of hyperparameters to choose is very high. Asides from the obvious learning rate and other parameters associated with the underlying gradient descent algorithm, the whole architecture of the network has to be decided. This includes the number of each type of layers (convolutional, pooling, dense, dropout), and the parameters of these various layers (filter numbers, convolution size, pooling size, dense number of neurons, dropout percentage, etc.).

Considering that the implementation found was adapted to the well-known MNIST digit classification dataset, it was deemed preferable to have the same feature dimensions as in the MNIST problem. For this reason, (feature #3) was used, but the two outmost columns and rows were cropped to end up with 28x28 pixels.

V. Testing and Validation

5.1 Logistic Regression and Multi-Layer Perceptron

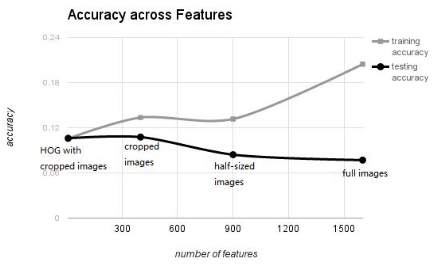

Figure 1: Accuracy of Logistic Regression vs Number of Features

The highest accuracy of Logistic Regression is 0.1.

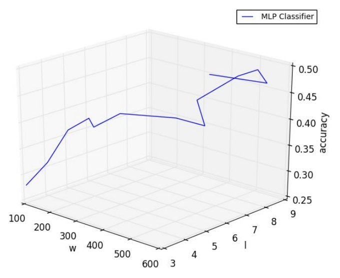

Figure 2: MLP Classifier Accuracy Across *l* And *w*, the Path of the Greedy Approach

For the MLP Classifier using (feature 2) and (feature 3), they gave high training accuracy of over 0.95 but low testing accuracy of below 0.2.

5.2 Feedforward Neural Network

The testing takes a few steps. The first step is to validate the forwarding summation of every single neuron, after feeding inputs to the system. In detail, we manually calculated the expected sum of outputs from each neuron and compared it with the sum from the manual implementation [6]. Even though the efficiency of such a testing method is low, it guarantees the implementation is correct for each single unit which is important for the further prediction. After this step, we tested the backward propagation structure. Instead of using a complicated network, we simply created a network with two input neuron, two hidden neurons and two output neurons. As previously, each output from every single step is manually calculated and compared to the actual results generated by the network. Since this simple example covers all built-in functions in each neuron and each layer, the correct results and weight updates prove that the functionalities of the entire structure will work and is able to be further extended to larger dimensions. The above two tests prove that the system is functional and is able to deal with simple examples, so that the full testing dataset can reliably be fed to the network to validate and test.

The data in the training set was first processed as described in section 3 (problem representation) in order to improve the efficiency. The hyper-parameters are configured as below. The accuracy and predicted output are printed out in each training step. As predicted, the weights are updated accordingly based on the error, and the predicted result moves closer to the true output for each example. However, after an arbitrary number of examples, all of the weights suddenly converge to a fixed set of numbers, resulting in a convergent output of 9, no matter what the actual output is. This issue is very hard to duplicate and could happen while training with any example. However, the chance of such issue occurring is higher if the numerical value of the previous output is larger than the current value. The exact reasons for this is still unknown, however it negatively impacted the results. One of the possible explanations for this issue is that the number of hidden layers is not sufficient enough to handle such a complicated task. In order to simulate any nonlinear function, the hidden layer number has to be at least two, whereas there is only one layer in our case. Another plausible reason could be the number of hidden neurons involved. Due to the limitation of the machine, the number of neurons may not have been high enough to generalize all kinds of images, resulting in the convergence issue.

In short, the implementation is proved to work with simple examples. However, due to the limitation of the computational power, some tradeoffs and compromises are made, which negatively affect the prediction results when dealing with much larger number of examples.

Hidden Layer	Hiddne neurons	Learning rate	Iterations	Neuro bias	Output
1	200	0.001	50	1	regression

Figure 3: Feedforward neural network architecture

5.3 Convolutional Neural Network

As far as testing goes, the convolutional neural network is pretty simple. Since we used a pre-existing implementation using Theano, nolearn and Lasagne, there was no question of the network being correct or not. The biggest challenge we faced was setting the hyperparameters, and choosing the general architecture of the network. Because the training phase of these types of network can take multiple hours, if not days, very few variations could be tested before the day of the deadline.

At first, only 5000 images from the training set were used to train the network, in hope of reducing the training time to a more manageable level. Unfortunately, this was not enough images, as the resulting network predicted nearly only 9's, with an accuracy of 10%. The number of images was then increased to 30000, and after a day of computations, the network slowly worked its way up to 64% accuracy. Then, a better computer was used to handle the whole 100000 images at the same time, and this allowed the network to reach our final accuracy of 86%.

Our final CNN architecture:

- A 28x28 input as described above
- A 5x5 convolutional layer with 32 filters
- A 2x2 pooling layer
- A second 5x5 convolutional layer, but with 144 filters
- A layer to dropout 25% of the inputs
- A dense layer of 256 neurons
- The final layer with 19 output neurons

Throughout, the learning rate was 0.01.

VI. Discussion

6.1 Logistic Regression and Multi-Layer Perceptron Classifier

As is shown in figure 1, the accuracy of logistic regression using different features is very low. Logistic regression has a hard limit in doing this task. As it is indeed very hard to use a single linear function of pixel values to represent the content of an image. Maybe a better selection of features can improve the result. One possible way of feature extraction is to use an autoencoder. But it is worth noting that a reduction of feature size generally improves the validation accuracy, although a large number of features gives lower training error.

The MLP Classifier performs much better. Through experiments, the number of neurons needed is much larger than the size of input and multiple layers works better than a single layer. Also the training time is not very long, within 3 hours. Though this raised the accuracy to 0.50, this method is still not very accurate compared to the 0.99 accuracy of other teams. Maybe some useful features were lost when cropping the images, or the MLP Classifier is just not good enough. But increasing the number of features did not improve the result. We expect other methods, like Convolutional Neural Network, to work better.

6.2 Feedforward Neural Network

There are a few advantages and disadvantages to this manually implemented feedforward neural network. One of the disadvantages is the number of hidden layers. Under the constraint of code complexity, there is only one layer of hidden neurons that got implemented in this manual approach. The code complexity would increase exponentially as the number of hidden layers increases. In order to keep a balance between the computation time and testing accuracy, we chose to use only one layer in this approach, which is the main factor limiting the prediction accuracy. The type of the output space also negatively affects the prediction results. Regression is chosen to deliver the final results. Thus, the rounding and truncation error gets introduced to the framework. For example, given a prediction result of 9.45, the rounded result will display 9, even though the true value is 10. For this reason, even though some results are predicated very close to the correct categories, they still get incorrectly rounded down and up, which lowers the prediction accuracy.

In terms of the advantages, implementation simplicity is worth discussing. The neuron, layer and entire network are implemented separately as three different classes, so that they can be reused as much as possible without duplicating implementations. In each class, a minimal number of functions are implemented, in order to remove any duplicated functions or variables. This design is extremely useful and efficient when the number of neurons increases to a few hundreds, making the future optimization much easier and straightforward. Besides, the learning rate and number of iterations for training each example are relatively balanced. The model is able to update weights stably and have sufficient steps to finish the weight update. This ensures the maximal efficiency and stability of the model.

6.3 Convolutional Neural Network

A big advantage of CNNs is that they somewhat simulate the workings of the human visual cortex. Their convolutional layers each look at a specific area around each pixel and try to match this area with various filters the network learns. This makes a very robust network for analyzing visual images.

The downside of such a network is the staggering amount of computations it takes to train it. A lot of those computations can be done in parallel on a GPU using some of the deep learning python libraries, but that’s assuming we have an Nvidia GPU available. It also takes quite a bit of RAM to train a network with the whole 100 000 images we had available. The main downside however is the length of the training phase, as it can easily take days to reach a good optimized solution. At least, once this is done, images can be fed through the neural network really fast to classify new images.

VII. Statement of Contributions

Yue worked on the part 1 of this project,. He also explored the way to crop the images to reduce the number of features. Liqiang worked on the part 2 of this project. He was responsible for a few approaches to clean up the raw dataset to speed up the computational time. Marc-Antoine worked on the part 3 of the project. He also used an autoencoder to experiment with a different way to represent the image data in fewer dimensions.

We hereby state that all the work presented in this report is that of the authors.

References

[1] Shapiro, L. G. & Stockman, G. C: "Computer Vision", page 137, 150. Prentice Hall, 2001

[2] Grace Tsai ,Histogram of Oriented Gradients,September 28, 2010.

[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg et al., “Scikit-learn: Machine learning in python,” Journal of Machine Learning Research, vol. 12, no. Oct, pp. 2825–2830, 2011.

[4] S. Perone, Christian. "Deep Learning – Convolutional Neural Networks And Feature Extraction With Python". *Terra Incognita*, 2015. Web. 2 Nov. 2016.

[5] Theano, Autoencoders And MNIST. *Triangleinequality For Matters Mathematical*, 2014. Web. 2 Nov. 2016.

[6] Matt Mazur. (2016). *A Step by Step Backpropagation Example*. [online] Available at: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/ [Accessed 8 Nov. 2016].