COMP-598 Mini-Project #3 Classification of Modified MNIST

Ayush Jain, Paul Pereira, and Suzin You (Kaggle Team: APS)

I. Introduction

The MNIST dataset of handwritten digits is popular in introducing machine learning for a reason. It is difficult enough to classify due to the irregularities in handwriting, but also easy enough since there are no other irregularities: all digits are oriented correctly and there is no noise around the handwriting itself. In many real life situations, however, we are required to identify digits written against dark, textured, or patterned backgrounds viewed from various angles.

In this project we use four different classifiers to categorize 50,000 modified examples from the MNIST dataset that have been re-scaled (from 28px×28px to 48px×48px), rotated at a random angle, layered with background patterns, and embossed. As a baseline approach we perform one-versus-all logistic regression and linear support vector machine algorithms. In both methods we utilize principal component analysis to reduce the dimensions of input space. We then compare the results with the ones obtained from fully connected feed-forward neural network, fully implemented by us, as well as convolutional network based on [1].

II. Methodology

We used NumPy package, scikit-learn library, and theano [2] to perform feature selection and classification. The lasagne module was used to implement the convolutional neural networks. The MNIST tutorial was used as a baseline for the convolutional neural network implementation. Our methodology is presented below.

A. Data preprocessing

To make the neural networks more robust, we rotate the input data at random angles before training. By doing so the neural network learns to make good predictions invariant under rotation.

B. Feature Selection

As neural networks do not generally require feature selection—in fact, feature selection process is considered to be embedded in the network—we present how we reduced feature dimensionality using principal component analysis.

1) Principal Component Analysis: The key idea of principal component analysis (PCA) is to reduce the dimensionality of the dataset that may contain possibly correlated variables, while retaining “as much as possible of the variation present in the data set” [3]. We achieve this through eigen decomposition of the data co-variance matrix, and thereby transforming the data to linearly uncorrelated variables called *principal components*.

We test with 16, 64, 256, and 576 principal components.

C. Classification Algorithms

We used four classification algorithms to tackle our problem. In the following we denote by X the $n \times m$ matrix representing n examples in terms of m features, X_i the i'th example, $y \in \mathbb{R}^n$ the output, and $\mathbf{w} \in \mathbb{R}^m$ the weight vector.

In logistic regression and linear support vector machines, we use only 20,000 examples for training with 5-fold cross validation and 10,000 examples for testing.

In neural networks we use 70% of the 50,000 examples as the training set, 20% as the validation set, and the remaining 10% as the test set.

1) Logistic Regression: Logistic regression assumes that the conditional probability of a class k given the input x has distribution

$$p_k(x;\mathbf{w}) = \frac{\exp(\mathbf{w}_k^T x)}{1+\sum_{i=1}^{K-1}\exp(\mathbf{w}_i^T x)}$$
$$p_K(x;\mathbf{w}) = \frac{1}{1+\sum_{i=1}^{K-1}\exp(\mathbf{w}_i^T x)}$$

where k runs from 1 to $K-1$ and K is the number of classes, so that the conditional probabilities sum up to 1. Hence the model is described by $K-1$ log-odds:

$$\log\frac{p_k(x;\mathbf{w})}{p_K(x;\mathbf{w})} = \mathbf{w}_k^T x, \quad k \in \{1,\dots,K-1\}. \tag{1}$$

The model is fit to the training data via the method of maximum likelihood. Thus the natural error function of choice is

$$\mathrm{Err}_{\mathrm{LR}}(\mathbf{w}) := \frac{\partial \ell(\mathbf{w};X)}{\partial \mathbf{w}} = \frac{\partial}{\partial \mathbf{w}}\sum_{i=1}^{n}\log p_{y_i}(x_i;\mathbf{w}). \tag{2}$$

This set of equations, nonliner in $\mathbf{w}$, has no closed form solution, but it can be approximated using gradient descent. We used the built-in Newton conjugate gradient method in scikit-learn's logistic regression classifier.

2) Linear Support Vector Machines: Support Vector Machine (SVM) is a powerful method of classification that uses as decision boundaries hyperplanes that create the greatest margin M between the data points from one label and the data points from another. To find the boundary $\mathbf{w}^T X = 0$ where $\mathbf{w}$ is the vector of weights and X is the input matrix, the optimization problem of interest is

$$\max_{\mathbf{w}} M$$
$$\text{subject to } \frac{y_i \mathbf{w}^T X_i}{\|\mathbf{w}\|} \geq M \text{ for } i = 1, \ldots, n$$

where $\frac{y_i \mathbf{w}^T X_i}{\|\mathbf{w}\|}$ is the distance from the data point to the boundary. If the datapoints are not separable we allow slacks ξ_i so that the constraint is $\gamma_i \geq M(1 - \xi_i)$. Misclassification occurs when $\xi_i > 1$ so we may impose $\sum_{i=1}^{n} \xi_i \leq K$ to bound the total number of misclassifications. Such a bound can be found through cross-validation.

We girdsearch over error term penalty $C = 0.01, 0.5, 0.001$ with the same set of number of components for PCA.

3) Fully Connected Feed-forward Neural Networks: An artificial neural network (ANN) is a two-stage regression or classification model inspired by the neural networks in the brains. We will be dealing with a $K = 10$-class classification task. The input layer consists of the features as nodes and the output layer consists of K nodes representing each class. The layers in between the input and output layers are called the hidden layers. In a fully-connected neural network, all inputs are fed into each node of the first hidden layer as $\mathbf{w}^T X$ for some weight vector for the features $\mathbf{w}$. Each hidden node "activates" according to the *sigmoid* function

$$\sigma(cz) = \frac{1}{1 + \exp(-cz)}$$

where larger c results in sharper activation rate at $z = 0$. Typically no more than a single hidden layer is needed to obtain a good classifier.

In order to train a neural network, we first initialize the weights to a small random number. We then repeat the following until convergence: First we pick an example from our training set. We calculate the prediction o of the neural network on that example. We then use backpropagation to compute the share of the correction for each unit in the network. For the output unit the correction is $\delta = o * (1 - o) * (y - o)$ where as for any hidden unit h, the correction is $\delta_h = o_h * w_{N+H+1} * \delta$. We then update the weights of the hidden unit h using gradient descent: $w_{h,i} = w_{h,i} + \alpha * \delta_h * x_{h,i}$

We avoid overfitting by L_2 regularization of the MSE.

In order to ensure our predictions are invariant under the rotation of the digits, we post-process the output: the prediction is rotated in 60 increments and the best matching digit is outputted as the final prediction.

Our implementation of Artificial Neural Network is not able to handle the large number of features in the dataset. Near convergence, some times we observed large activation values (of order 2) for datasets with large number of features causing sigmoid values to approach hard boundary values and hence low accuracy. For datasets with smaller feature-sets, like sklearn's iris dataset or auto-regression tasks, our implementation achieves very high accuracy. We expect this implementation to achieve good results with some feature extraction layer appended before it.

4) Convolutional neural networks: Convolutional neural networks (CNNs) are special neural networks that have been shown to produce excellent results on image classification problems. They make use of convolutional and pooling layers in order to make use of the spacial dependencies in images.

A convolutional layer has its units split into multiple feature maps (or filter). Each unit of a feature map takes as input a different local region of the image. The size of the region and the number of feature maps are specified as hyper-parameters. Furthermore, all the units in a feature map have the same set of weights (meaning for a given feature map, the same filter is applied to all the local regions).

A pooling layer takes as input the output of a convolutional layer and performs a downsampling operation in order to reduce the variance of the model.

To train the CNN, we use gradient descent with a categorical cross entropy loss function. The optimization problem of interest is therefore to minimize

$$\mathrm{L} = \sum_{i=1}^{n} \sum_{j=1}^{K} \log(p_{i,j}) * t_{i,j}. \quad (3)$$

where $p_{i,j}$ is the probability that instance i belongs to category j according to the output layer of our neural network. The activation function of the output layer is softmax since it lends well to multi-classification problems.

D. Hyper-parameter selection

1) Convolutional Neural Networks: In order to narrow down designs for our CNN, we looked at previous implementations of convolutional neural networks. The first implementation we looked at was LeNet5 [1], first used to classify the MNIST dataset. Our model uses a similar architecture as LeNet5:

1) 1 input layer
2) 2 convolutional-pooling layers
3) 1 fully connected hidden layer
4) 1 fully connected output layer

presented in order. We reduce the number of hidden layers to one to keep the training time low.

We used three different model configurations: for the first version of our model we keep the number of feature maps at each convolutional layer roughly the same as LeNet5 at 8 and 16(Model 1). In hopes of better results we also tried a model with 16 and 32 feature maps(Model 2). The hidden layer has 128 units in the first model and 256 in the second model to keep the reduction factor consistent. For the third model, we add a third convolutional layer between the second convolutional layer and the second pooling layer, following the notion of stacking convolutional layers before pooling layers, first used in AlexNet [4](Model 3). The third convolutional layer has the same number of feature maps as the second (32).

In order to select the best model, we trained our convolutional neural networks on 70% of the training data, estimated the accuracy of each model on a validation set corresponding

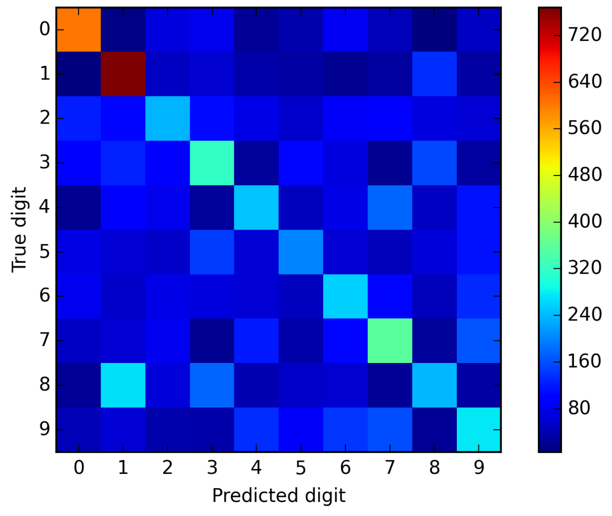

Fig. 1. Confusion matrix of predictions made using multinomial logistic regression with PCA

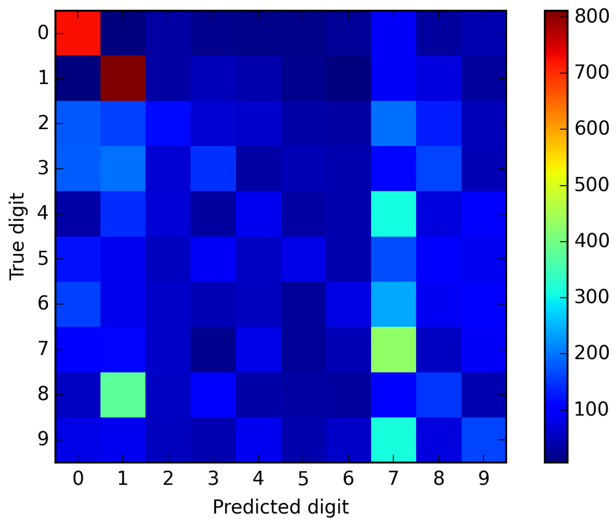

Fig. 2. Confusion matrix of predictions made by linear SVM with PCA

to 20% of the data and tested our best model on the remaining 10%

To lighten the computational load, instead of using a validation set to select the other parameters of our model, we relied on previous work ([5] and [6]) to find a configuration for our parameters that was likely to produce good results. We initialized the learning rate at 0.1 and reduced it by 0.02 every 50 epochs. The network was trained over 200 epochs. We did not use regularization as it was found to have little effect towards getting better results [5]. In order to reduce the risk of overfitting we used a 0.5 dropout on our hidden layer and output layer [6]. This means that whenever we use backpropagationto update the parameters of our convolutional neural network, each unit in our hidden layer has propability 0.5 of not being included in the update process. Finally we set the kernel size (the size of the local region each unit uses as input) of each convolutional layer to be 5×5.

III. Results

Through girdsearch and cross validation, we chose $C_{\text{logreg}} = 0.01$ to be the inverse of L_2 penalty for multinomial logistic regression and $C_{\text{SVM}} = 0.01$ for the error penalty in SVM. In both logistic regression and linear SVM, we used 576 principal components in PCA following the results from gridsearch.

The confusion matrices of logistic regression and linear SVM is shown in figures III and III, respectively.

In both learning curves III and III, the training score is very high in the beginning and then decreases, and the validation score is very low at the beginning and then gradually increases (although in the case of SVM it slightly decreases after about 9000 examples). Such a trend is likely due to the fact that our training examples are not linearly separable. The more examples we take into account of linear separation, the more examples we will misclassify. At any rate since both methods

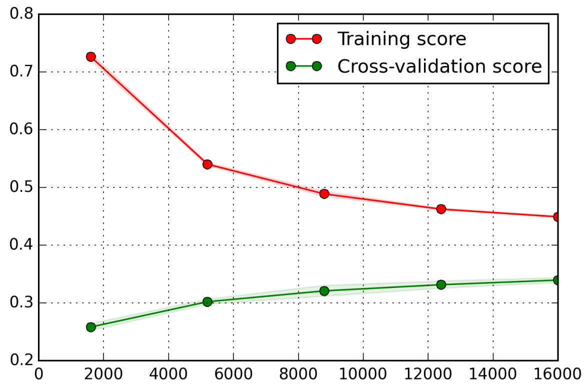

Fig. 3. Learning curve of multinomial logistic regression with 5-fold cross validation. The x-axis is the number of training examples and the y-axis is the score.

falsely assume that the data points are linearly separable, our validation score cannot improve beyond a certain point.

The following table illustrates the performance of the three model configurations of CNN:

TABLE I
Accuracy of proposed CNN models on validation set

Model	Accuracy
Model 1	0.9492
Model 2	0.9725
Model 3	0.9739

Model 3 performed the best on our validation with an accuracy of 97.39%. This is only slighly better than model 2 that implements one less convolutional layer. Model 1 had an accuracy of 94.92%, about 3% less than the other two model. Since model 3 performed best on the validation set, we used this model to make predictions on our test set. In order to

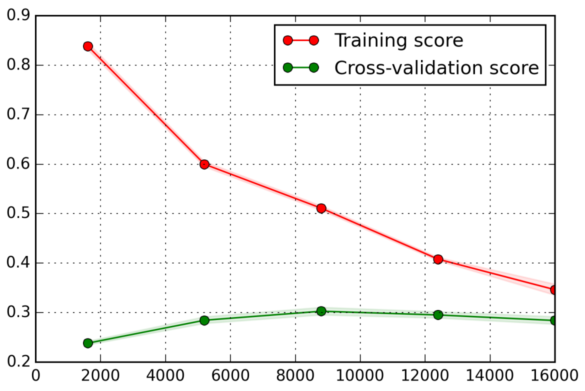

Fig. 4. Learning curve of linear SVM with 5-fold cross validation. The x-axis is the number of training examples and the y-axis is the score.

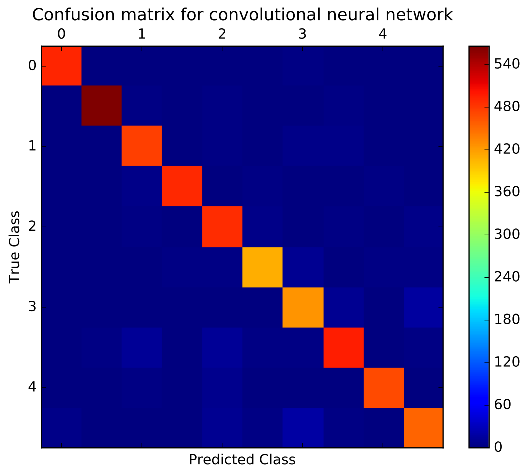

Fig. 5. Confusion matrix of predictions made by CNN

evaluate our model, we look at the accuracy, the precision, the recall and the f1-score on the test. We find that for these four scores, model 3 performed very well (all score above 0.95). The confusion matrix we obtained on the test set is seen in . Overall, the results indicated that Model 3 is a excellent classifier for the Modifier MNIST classification problem.

TABLE II
SCORES OBTAINED USING MODEL 3

Accuracy	Precision	Recall	F1-score
0.9542	0.959637	0.959739	0.95254

IV. Discussion

A. Feature Selection

In both multinomial logistic regression and linear SVM, PCA with the most number of principal components (576) out of all candidates 16, 64, 256, 576 produced the best result. Figure IV-A shows the explained variance of each principal component. We see that the first 200 principal components account for most of the variance in the 20,000 training examples.

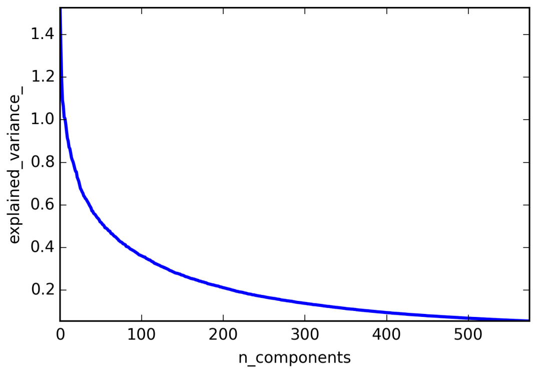

Fig. 6. Explained variance of each principal component. There are 576 principal components and PCA was performed on 20,000 training examples

B. Classifier Performance

As expected from the accuracy scores in III, we see a weak trend of heat (higher number of accurate predictions) along the diagonals of figures III and III, but the trend in logistic regression case is much stronger than linear SVM. In both figures, 0 and 1 which are the simplest, hence more likely to be uniform across different handwriting, are indeed most often correctly classified. A bizarre trend of incorrectly classifying an 8 as a 1 is seen in both.

As demonstrated multiple times, the convolutional neural networks performed excellently on the modified MNIST dataset. The fact that our most complex model performed better compared to the other two suggests that in order to further improve our results we would need to consider even larger models. Increasing the number of feature maps in our convolution layers appears to have been a good intuition since model 2 performs about 3% better than model 1. The number of feature maps in model 1 were chosen for a dataset that was cleaner(MNIST). It seems only logical that in order to perform well on a modified version of MNIST we would need to increase the number of filters that convolutional layers learn. Whereas before a single filter might have been enough to detect for some feature of the image, multiple filters would now be required to extract the same feature. For example, if previously we had a filter detecting long vertical lines in the MNIST, we would need multiple filter to detect that same long line, where each filter would be attuned to one orientation. On the other hand, the fact that model 1 still performed with 0.94 accuracy on the validation set suggests that even a simple convolutional neural network is well equipped to deal with high variability in the input data for image classification problems.

TABLE III
PERFORMANCE OF CHOSEN MODEL ON TEST DATA

Classifier	Parameters	Accuracy Score
Multinomial logistic regression	$C_{\text{logreg}} = 0.01$	0.3508
SVM with Gaussian	$C_{\text{SVM}} = 0.01$	0.2786
Fully connected feed-forward NN	-	-
Convolutional NN	$\alpha = 0.1$, CL= 3, Filters= $[16, 32, 32]$	0.9739

One possible explanation for the 5% error on the test set may be the difficulty that our model 3 had in correctly classifying images with true value 7. A small number of them were classified as 2 or 4. Furthermore, the convolutional neural network classfied some 6's as 9's and some 9's as 6's. This is less surprising since 6's and 9's look very similar, especially considering the original images were rotated.

C. Future Work

Due to limit on computational power, linear SVM method was only tested with three different error penalty constants. A gridsearch through different loss functions, L_1 and L_2 regularization methods, and with possibly more C_{SVM} values may result in better performance.

However, as demonstrated by the learning curves, any method with linear separability assumption on the dataset will not achieve good accuracy rate.

For the convolutional neural network, it would be interesting to perform an extensive grid search in order to find optimal parameters for model 3. Indeed we have shown that by changing the architecture of the convolutional neural network we can improve our predictions. The next stepo would be to know if optimizing the model on a lower scale (for example the learning rate) we can further improve our results. Again, more computational power and time would be needed for this however.

V. Statement of Contributions

Ayush implemented the fully-connected feedforward neural network. Paul implemented the convolutional neural network. Suzin tested for performance by logistic regression and SVM algorithms from scikit-learn. All three contributed to the report.

VI. Integrity of work

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

References

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, November 1998.

[2] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing In Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[3] I. Jolliffe, *Principal Component Analysis*. Springer Verlag, 1986.

[4] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[5] P. Y. Simard, D. Steinkraus, and J. C. Platt, "Best practices for convolutional neural networks applied to visual document analysis." Institute of Electrical and Electronics Engineers, Inc., August 2003. [Online]. Available: http://research.microsoft.com/apps/pubs/default.aspx?id=68920

[6] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html