MNIST Handwritten Digit Classification Using SVM and Deep Learning

Omar H. Abdelkader[1]

[1]University of Maryland, College Park
Electrical and Computer Engineering – A.V. Williams Building
8223 Paint Branch Drive – College Park, MD – 20740

oabdelka@umd.edu

Abstract. *The MNIST database is a dataset of handwritten digits commonly used for training and testing machine learning models. In this paper, we propose implementing the linear Support Vector Machine (SVM) classifier and a convolutional neural network modeled after LeNet-5 to achieve handwritten digit recognition.*

Keywords*: support vector machine, convolutional neural network, LeNet-5.*

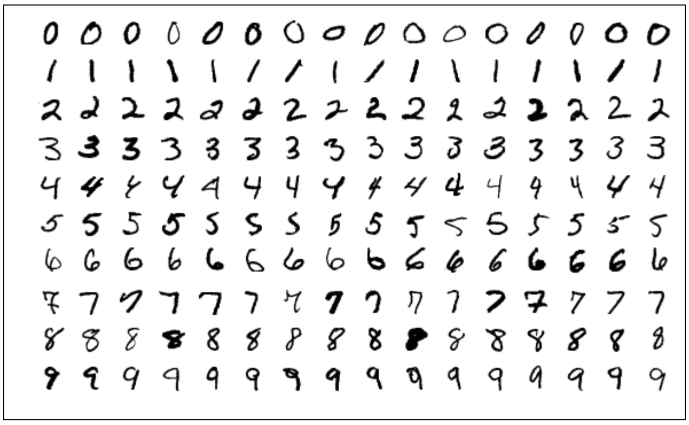

Figure 1. MNIST Handwritten Digits Sample

1. Introduction

Support vector machines are a technique for solving supervised learning problems. In general, for two classes, SVM seeks to define a decision boundary that is maximally far away from any data point. The advent of the kernel trick, which allowed one to solve nonlinear classification problems by mapping the data onto a higher dimension space to increase separability made SVM the de facto classification technique up until very recently.

Currently, advances in hardware and access to more data have caused multilayer neural networks, or deep learning models, to become quite popular. Like SVM with the kernel trick, deep learning models are able learn nonlinear mappings via the connection of multiple layers of perceptron nodes. Through the use of stochastic gradient descent, training multilayer neural networks is also trivial to parallelize. We will construct a model inspired by LeNet-5[1], a convolutional neural network architecture specifically designed for handwritten and machine-printed character recognition.

[1] http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

2. Theoretical Framework

The MNIST dataset is composed of 60000 labeled training images, and 10000 labeled testing images. Each image is grayscale and of size 28×28 pixels.

In this paper, we implement support vector machine classification using scikit-learn[2], an open-source Python library for machine learning, and LeNet-5 using PyTorch[3], an open-source Python library for deep learning classification.

3. Methodology

The MNIST dataset available on Yann LeCun's website[4] is already split into training and testing datasets. As such, no consideration is made to additionally preprocess the data in any form.

3.1. Support Vector Machine Classifier

Traditionally, SVM is used to solve two-class classification problems, however the procedure can be easily generalized for c classes. Under linear conditions, scikit-learn offers two strategies for solving multi-class SVM.

- **SVC** When used with the linear kernel argument, creates a "one-vs-one" model. This model constructs $\binom{c}{2}$ pairwise classifiers. Each classifier trains data from two classes.
- **LinearSVC** The "one-vs-rest" classifier. Trains c models, where each model considers samples of a particular class as positive, and the samples belonging to the remaining classes as negative.

According to scikit-learn's documentation, in the linear case, the algorithm used by the "one-vs-rest" classifier implementation is much faster than the "one-vs-one" classifier. As such, we use the `LinearSVC` classifier in this report.

Linear SVM seeks to define a decision hyperplane of the form $\vec{w}^T\vec{x}+b = 0$. The decision hyperplane normal vector, $\vec{w}$, and intercept term, b, are commonly referred to as the *weight vector* and *bias* in machine learning literature, respectively. A good choice for $\vec{w}$ and b is one that both maximizes the margin, and correctly separates as many training samples as possible. The trade-off between these two criteria is defined by the training accuracy penalty coefficient, C. Smaller values of C prioritize the former criterion, while larger values of C will prioritize the latter.

We select the optimal value of C through K-Fold Cross Validation. This strategy involves partitioning the training set into K equally sized subsets. For every iteration, one subset is withheld as the validation set, and the remaining $K - 1$ subsets are used as training data. This process is repeated K times such that each subset is used as the validation set once. The results for each fold are then averaged to produce a single result. This approach reduces the dependence of the results on the specific training set that is used.

Fortunately, scikit-learn offers `GridSearchCV`, a class for hyperparameter tuning using K-Fold Cross Validation out of the box. In the code snippet below, we select the optimal

[2] `http://scikit-learn.org/`

[3] `https://pytorch.org/`

[4] `http://yann.lecun.com/exdb/mnist/`

value of C from a subset defined in $tuned\_parameters$. Using $K = 5$ folds, we find that $C = 1$ is the best parameter.

5-Fold Cross Validation Hyperparameter Tuning using GridSearchCV

```
# Set the parameters by cross-validation
tuned_parameters = {'C': [0.01, 0.1, 1, 10, 100]}
clf = GridSearchCV(svm.LinearSVC(), tuned_parameters, cv=5)
clf.fit(X_train, y_train.tolist())
print(clf.best_params_)

>> {'C':1}
```

3.2. Deep Learning

Deep learning models have can have a large number of hyperparameters that need to be tuned. These include the number of layers to add, the number of filters at each layer, whether to subsample, kernel size, stride, padding, dropout, learning rate, momentum, batch size and so on. The number of possible combinations that these variables can take is infinite, making it very difficult to use cross-validation to estimate any of these hyperparameters without specialized GPU hardware to accelerate the process.

As such, we present a model very similar to LeNet-5 (shown below). The original LeNet architecture pads the input image with 0-pixels to make it of size 32×32 pixels, followed by two successive layers of convolution and subsampling with tanh activation. The next two layers are fully connected linear layers with tanh activation, and the final layer is a layer of Gaussian connections, which are fully connected nodes that implement mean-squared-error as the loss function.

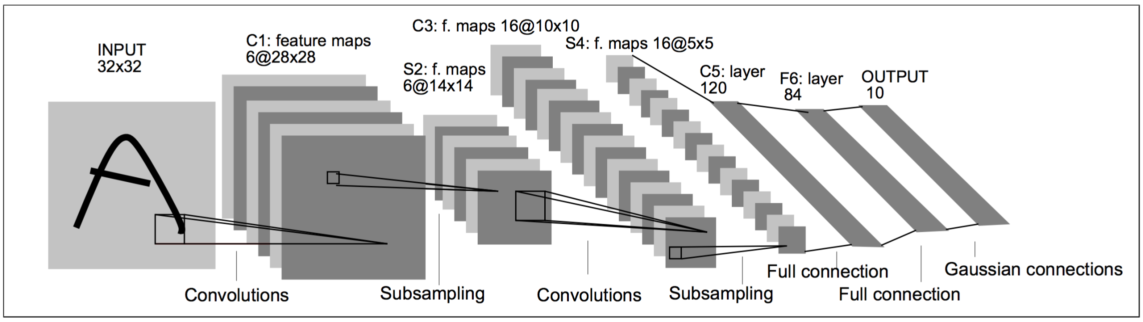

Figure 2. LeNet-5 Architecture

The model we present in this paper largely follows the LeNet-5 architecture, but with a few minor changes. To start, we introduce the softmax function as the activation function for the final fully connected layer. The softmax function normalizes the output vector such that it is a valid probability distribution over the possible 10 classes. When used in conjunction with the negative log likelihood loss, we can heavily penalize cases where we predict the wrong output class. In addition, we choose ReLu over tanh as the activation function for the other layers due to the reduced likelihood of vanishing gradient in deeper layers of the network.

To optimize the network, we use PyTorch's base Stochastic Gradient Descent class with a learning rate of $\eta = 0.01$ and momentum value of $\gamma = 0.5$. Momentum is a method that helps accelerate SGD in the relevant direction. It works by adding a fraction γ of the update vector of the past time step to the current update vector[5]. This kind of behavior is desirable when SGD approaches local minima or saddle points.

In order to select the optimal weights for the test set, we partition the training set using a $(90 : 10)$ split to create a new training set and a validation, or hold-out, set. We train the new training set for a predetermined number of epochs, and after each epoch, we compute the loss on the validation set. After the training phase is complete, we use the weights from the epoch with the lowest average validation loss on the testing set. This is a common practice in machine learning to prevent overfitting.

4. Results

All benchmarks and results presented below were run on a 2015 MacBook Pro with a 2.2 GHz Intel Core i7 and 16 GB 1600 MHz DDR3 memory.

4.1. Support Vector Machine Classifier

For $C = 1$, one run of the linear SVM classifier completed in 2 minutes and 21 seconds, and achieved a training and testing accuracy of 89.41% and 88.43%, respectively. Recall, increasing the penalty parameter C could have potentially increased the training accuracy, however this would have been at the expense of a large margin, which could have caused the model to overfit to the training data.

Another way to analyze and rank the performance of the SVM classifier would be to examine the confusion matrix.

Table 1. Linear SVM Confusion Matrix

	0	1	2	3	4	5	6	7	8	9
0	948	0	1	3	0	3	14	7	4	0
1	0	1106	3	2	0	1	5	1	17	0
2	23	22	852	15	10	3	23	27	52	5
3	16	0	19	873	4	25	9	26	33	5
4	4	2	5	2	899	0	20	16	10	24
5	16	1	3	46	10	711	30	27	45	3
6	12	2	6	0	3	9	919	0	7	0
7	2	9	7	3	9	0	1	970	4	23
8	16	12	8	23	16	38	13	31	806	11
9	13	8	2	14	54	2	0	136	21	759

[5] http://ruder.io/optimizing-gradient-descent

The confusion matrix tells us how likely the model is to mistake a particular class with another. The row labels represent the predicted class labels, and the column labels represent the actual class labels. Correct classifications are along the left diagonal of the confusion matrix. The model most correctly classified the number one, and least correctly classified the number five. The most common mistake the classifier made was mistaking the number seven for the number nine, $(7 \rightarrow 9)$. Other frequent errors included $(8 \rightarrow 2)$, and $(4 \rightarrow 9)$. Visually, these results are rather intuitive. The number 1, for instance, has a very distinct shape with respect to the other digits in the Arabic numeral system.

4.1.1. Dimensionality Reduction

Principal Component Analysis (PCA) is a technique for reducing a given data set to a lower dimension. PCA works by projecting the original data onto a subset of the feature vectors, known as the principal components. In order to compute which directions to project onto, one computes the scatter matrix of the centered data and solves for its eigenvalues and eigenvectors.

A major shortcoming of PCA is that it does not preserve between-class scatter very well. In other words, samples that appear to belong to distinct classes in a higher dimension space appear to belong to the same class when projected onto a lower dimension space. Fisher's Linear Discriminant Analysis projects a 2-class data set of d dimensions onto a line for which the projected samples are well separated.

Generalizing this procedure to c classes is known as Multiple Discriminant Analysis (MDA). Naturally, this involves at most $c - 1$ discriminant functions; thus, the projection is from a d-dimensional space to an m-dimensional space, where $m \leq c - 1 < d$.

Table 2. Post-LDA Linear SVM Confusion Matrix

	0	1	2	3	4	5	6	7	8	9
0	948	0	2	2	0	7	10	2	8	1
1	0	1089	4	6	3	3	5	2	23	0
2	18	19	864	29	14	3	26	16	36	7
3	4	2	30	885	4	25	4	18	26	12
4	1	14	9	0	887	2	11	2	11	45
5	15	6	3	52	18	715	16	16	40	11
6	25	6	15	0	14	18	872	1	7	0
7	4	21	20	9	16	3	0	900	0	55
8	12	23	12	30	23	48	14	7	785	20
9	13	9	5	14	61	6	1	66	10	824

In the last project[6], we saw that MDA preserved between class scatter much more effec-

[6] https://github.com/omikader/facial-recognition

tively in comparison with PCA; as such, we only consider MDA in this project. After reducing the dimension size of the data from 784 to $m = c - 1 = 9$, one run of the linear SVM classifier completed in 16.22 seconds and achieved a training and testing accuracy of 87.68% and 87.69%, respectively.

The performance of the post-MDA linear SVM classifier is very similar to that of the unreduced linear SVM model, but were obtained in a fraction of the computation time. In addition, like the unreduced SVM classifier, the training and testing accuracies for the reduced model are both very close to one another. This indicates that neither model overfit to the training data.

4.2. Deep Learning

We ran with the model defined in section 3.2 for 50 epochs and found that the 24^{th} epoch had lowest average loss on the validation set. Running the weights from the model for this epoch achieved a training accuracy and testing accuracy of 98.7% and 98.9%, respectively. The entire procedure took approximately 15 minutes and 53 seconds to run from start to finish. This is a significant accuracy boost from the linear SVM classifier results in sections 4.1 and $4.1.1$.

5. Conclusion

Once again, we see that Fisher's Multiple Discriminant Analysis is a viable technique for dimensionality reduction in the field of image classification. The accuracy of the linear SVM model was hardly affected by reducing the dimension size from 784 to 9.

Neither the linear SVM classifier, nor the convolutional neural network overfit to the training data. This is expected, as both models implement safeguards to prevent overfitting from occurring. For linear SVM, this is in the form of maintaining maximum margin between classes. For deep learning models, it is done through using convolutional layers or dropout as opposed to strictly using fully connected layers.

6. Considerations for Future Research

Investigating the performance of an SVM classifier that leverages the kernel trick would have been a more legitimate comparison to the deep learning model than the linear SVM classifier. In addition, it would have been beneficial to examine the effect of implementing dropout on the generalization performance of the deep learning model.

No preprocessing or segmentation was done to the data before training either model. Preprocessing the dataset is a common practice in deep learning image classification. This usually takes the form of data augmentation and normalization. Normalization is done to ensure each feature to has a similar range such that the gradients training the model do not go out of control. Fortunately, for common datasets like MNIST, PyTorch can normalize the data during the tensor transformation step.