# Report for Project: Classification of modified MNIST
# CS420 Machine Learning
## Spring 2018

Zirun Wang
515030910024
Shanghai Jiao Tong Univ.
wzrain@163.com

Minchao Li
515030910361
Shanghai Jiao Tong Univ.
marshallee413@sina.com

## Abstract

*MNIST database of handwritten digits is one of the most well-known datasets for image classification tasks. It has 60000 training samples and 10000 testing samples with 10 categories referring to 10 digits. Each sample is a 28*28 pixel image showing the digit of its category. The digit is centered in the pixel image. Here we need to deal with a modified version of MNIST dataset. Samples in this modified version are expanded to 45*45 pixel images and digits are no longer centered in the images. In each sample, the digit is randomly put somewhere in the image and noise is randomly added. So we need to preprocess the data to make it uniform. Then we use logistic regression as a traditional approach and some deep learning approaches like fully-connected network, convolutional neural network(CNN), Variational Autoencoder(VAE) with Support Vector Machine(SVM) to classify the data.*

## 1. Methods

The samples of the official version MNIST dataset are in a uniform pattern and the features of most handwritten-digit images are extremely clear. So it's actually an entry-level task to classify this famous dataset, no matter by a traditional method, or mighty neural network models. In the given modified MNIST dataset, however, the main feature of a image-a handwritten digit-is not in the middle and may occur anywhere in that image. And there are a lot of noise points scattered in those images. When we need to deal with this kind of images with features in different positions and some noise, convolutional neural network(CNN) is always the model that springs to mind. However, for a classification model, especially models that cannot extract local features but rely on data itself, a cleaner and more inerratic dataset can improve the result way more efficiently than struggling to adjust parameters to build a better model. For this modified version of MNIST, we've already had high standards set by outstanding classification results from various models on the official version MNIST dataset which is cleaner and much more inerratic. So trying to make the modified MNIST cleaner and more learnable should be considered before selecting algorithms.

### 1.1. Data Processing

There are two main tasks to do. The first one is to remove the noise. After observing the image, we find that most of the noise is small isolated noise points scattered over the image. Pixel-wise speaking, these points are nonzero pixels in a small range forming a small pixel "island". Therefore we can search thoughout the image to find out if there are adjacent nonzero pixels forming so small an "island" that it cannot represent a digit. And we can set pixels in these "islands" as 0 to remove the noise point. Note that in some images there are some noise points that actually weakly connected to the digit and form a small "peninsula", so they cannot be removed by searching for "island"s. But we reckon that these are relatively inconsequential in classification, so those "peninsula"s remain in the data.

The second one is to center the digits. After some searching processes above, pixels in a sample are all zero except those which represent a digit. Therefore we can trim samples by deleting some idle rows and columns to center digits while reducing the dimension of data at the mean time. To make the data uniform, we set an identical size for every sample when trimming them.

In addition, to reduce the number of parameters when training a model with huge amounts of parameters, we may use Principal Component Analysis(PCA) to reduce the dimension of the original data.

### 1.2. Logistic Regression

Logistic regression is a supervised learning method. For a binary classification problem, let $X = (x_1, x_2, ..., x_n)$

denote the samples where $x_i$ is one sample, and $Y$ denote the output($Y \in \{0, 1\}$). Use sigmoid function to map the output between 0 and 1, then we have

$$P(Y = 1|x_i) = \frac{1}{1 + e^{-\theta^T x_i}}$$
$$P(Y = 0|x_i) = \frac{e^{-\theta^T x_i}}{1 + e^{-\theta^T x_i}}$$

and the sample is put into the category with the larger probability. And the log odds should be

$$logits(x) = log\frac{P(Y = 1|x)}{1 - P(Y = 1|x)} = \theta^T x$$

Let the hypothesis function $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$, then we use log likelihood function as the loss function

$$\begin{aligned} L(\theta) &= \sum_{i=1}^{n}[y_i log(h_\theta(x_i)) + (1 - y_i)log(1 - h_\theta(x_i))] \\ &= \sum_{i=1}^{n}[y_i log\frac{h_\theta(x_i)}{1 - h_\theta(x_i)} + log(1 - h_\theta(x_i))] \\ &= \sum_{i=1}^{n}[y_i logits(x_i) - log(1 + e^{\theta^T xi})] \end{aligned}$$

then maximize $L(\theta)$ to get the estimate value of $\theta$.[5]

However, we are going to solve a multi-class classification problem, so we extend the binary logistic regression to softmax regression to fit our problem. In softmax regression, there are $K$ categories, where

$$P(y = k|x) = \frac{e^{\theta_k^T x}}{\sum_{i=1}^{K} e^{\theta_i^T x}}$$

denotes the probability of a sample belonging to the $k$th category. Also here each category has a parameter $\theta_i$, which reflects the unique property of each category, so the parameter is actually a matrix

$$\theta = \begin{bmatrix} —\theta_1^T— \\ —\theta_2^T— \\ \vdots \\ —\theta_k^T— \end{bmatrix}$$

So the log likelihood loss function looks like

$$L(\theta) = -\frac{1}{n}\left[\sum_{i=1}^{n}\sum_{j=1}^{k}\chi_j(y_i)\log\frac{e^{\theta_j^T x_i}}{\sum_{l=1}^{k} e^{\theta_l^T x_i}}\right]$$

where $\chi_j(i)$ is characteristic function, if $i = j$ then the value is 1, else the value is 0.

So briefly speaking, in Logistic Regression, every class has a group of parameters. Every one of these parameter corresponds to one dimension of the original data. This means, if samples from the same class have an analogous pattern and samples from different classes are organized in significantly different ways, we can use logistic regression to learn different parameters for each class and classify the data. And for this modified version of MNIST, after preprocessing, if two samples represent the same digit, they tend to have similar values in any row of the pixel images. So we can transform the pixel images into vectors and samples from the same class can still have similar patterns. Then we can try to use Logistic Regression to do classification.

### 1.3. Fully-Connected Network

Although logistic regression uses sigmoid function, it is actually a linear classifier, because the classification is based on the value of $logits(x) = \theta^T x$. Neural Networks, however, use the non-linear values that comes from such non-linear functions like the sigmoid function to create a non-linear classifier to fit any functions.

Fully-connected networks are the most elementary neural network. It is composed of multiple processing layers to learn representations of data with multiple levels of abstraction. The minimum unit of a fully network is a neuron. Primitively, a neuron is a Perceptron, which is basically a linear classifier. If $\theta^T x > threshold$ then the output is 1, else the output is 0, where $\theta$ is the weight of the perceptron and $x$ is the input vector of the perceptron. Based on perceptron, activation functions are used to smoothly map $\theta^T x$ to $(0, 1)$ so that a small change in the weight just cause only a small corresponding change in the output. Activation functions add non-linear factors into the model. Sigmoid function, tanh function and ReLU function are some commonly-used activation functions. If sigmoid function is used as the activation function, a neuron is actually a logistic regression classifier.

Neurons are stratified as layers. The input layer, which is the top layer, is the same size as the dimension of input samples. The output layer, which is the bottom layer, is the same size as the number of classes. Hidden layers are in the middle and ingeniously designed to grab characteristics of the data. The output values of a certain layer will be input values of every neuron in the next layer. The weight of a hidden-layer neuron represents a kind of decision. With the decision enhanced through the layers, the neural network will provide a reliable classification result.

When training a neural network, we also need some methods and tricks as follows.

- **Gradient Descent** Training a model is a process of optimizing a loss function to make the model fit the data well. But some loss functions are hard to optimize directly. So we use the gradient descent method, which

is a iterative algorithm to optimize a function. To find a local minimum of a function using gradient descent, the iteration step is like

$$\theta_{n+1} = \theta_n - \eta \nabla L(\theta_n)$$

one takes steps proportional to the negative of the gradient (or of the approximate gradient) of the function at the current point. Here $\nabla L$ is the gradient operator, which is the fastest-descent direction on the tangent plane of the current point and $\eta$ is the learning rate. Then we have

$$\Delta L = -\eta ||\nabla L||^2$$

when $\eta$ is small enough, so the function becomes smaller and smaller and converges to the local minimum value. If the function is convex, then we can get the global solution.[3]

- **Regularization** In fully-connected networks, the number of parameters are usually enormous, so it's likely to suffer from overfitting. So during the training process, we need to try reducing the number of parameters. One regularization way is to limit the number of nonzero parameters. Primitively we use L0 norm to control the number of nonzero parameters in the model. But L0 norm is hard to calculate, so we use L1 norm

$$||\theta||_1 = \sum_{i=1}^{N} |\theta_i|$$

which is a good approximation of L0. Under the constraint of L1, the parameters will be more sparse, so it's more unlikely to go overfitting and the model complexity is reduced. Besides L2 norm is also used

$$||\theta||_2 = \sqrt{\sum_{i=1}^{N} \theta_i^2}$$

to make the parameters more smooth without discarding some features. We can use norms to constrain the original loss function

$$min\{L(\theta) + \lambda \Omega(\theta)\}$$

where $\Omega$ is the L1 or L2 norm and $\lambda$ reflects the strength of the constrain.
Besides, we can also apply dropout method to prevent overfitting. In dropout method, every neuron has a probability to be temporarily inactive, so every time we are training a simpler and maybe slightly different model.

### 1.4. Convolutional Neural Network

In Logistic Regression and Fully-connected Networks, we need to transform pixel matrices into vectors. This process actually discard some information from pixels that ar adjacent in same columns of the original images. These two models focus more on the data itself, rather than the FEATURES of the data. A convolutional Neural Network(CNN), however, introduce convolution and pooling layers to learn the actual features of given data. In this structure, convolution/pooling layers connect only to local neighborhoods so that CNN is a sparsely connected neural network. The structure of convolutional and pooling layers is shown in Figure 1.

- **Convolution Layers** We use convolution kernels to extract features. A convolution kernel will scan through the values of a layer. At every position it scans, every weight in the convolution kernel multiplies by the value in the corresponding position of the weight. If the feature in the neighborhood is similar to the convolution kernel, the multiplication result will be quite a large number, so a convolutional kernel is also called a filter. After a scanning process, every filter will get a feature map of the same size with the former layer.
Use 1-dimension data as an example for the sake of simplicity. For an input vector with $n$ components $\mathbf{x} = (x_0, x_1, .., x_{n-1})$, a convolution layer outputs a vector of the same size $\mathbf{y} = (y_0, y_1, .., y_{n-1})$ where

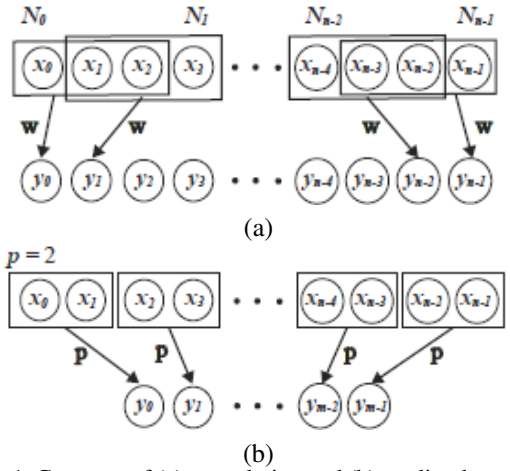
$$y_i = \sum_{j \in N_i, k \in [0,m]} w_k x_j$$

where $N_i$ is a fixed-length set of continuous indices of $\mathbf{x}$ and $w_k$ is the parameter of one filter whose length is $m$. Since the parameters carried by a filter is shared during the convolutional process, the number of parameters is reduced.

- **Pooling Layers**
Pooling layers are basically using one value to represent an area of data. They are usually used right after convolutional layers to downsample the feature map. They actually simplify the data and remove some redundant information. After convolution/pooling layers, the translation invariance is introduced. It assures that certain features can be extracted even if they are located in different positions in images. Here we use max pooling, which use the maximum value of the input area as the output:

$$y_i = \max_{j \in N_i} x_j$$

where $N_i$ is a fixed-length set of continuous indices of $\mathbf{x}$. The length of $N_i$ in convolutional/pooling layers is set by the moving stride of the filter.




(a)

(b)

Figure 1. Concepts of (a) convolution and (b) pooling layers

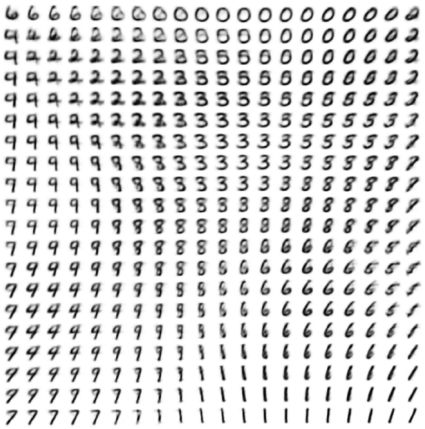


Figure 2. learned MNIST manifold

After features are extracted by convolutional/pooling layers, the fully-connected network is employed to classify these features.

### 1.5. Variational Autoencoder with SVM

Variational Autoencoder[1], intuitively, is model that encodes the original data into latent space and decode the latent vector back to reconstruct the original data. For decoder part, neural network is used to map from latent vector to reconstructed data, while for encoder part, Gaussian distribution $\mathcal{N}(\mu(X),\, \Sigma(X)\dot{I})$ is used to form the latent vector from the original data. The reason why we use Gaussian distribution as the mapping is that $P(z|X)$ is hard to compute, therefore, we use a easier pattern (Gaussian distribution) to approximate to the real one. The assessment of approximation is evaluated by the following formula:

$$\begin{aligned} & logP(X) - KL(Q(z|X)||P(z|x)) \\ &= logP(X) - [E_{z\sim Q}(log(Q(z|X) \\ &\quad - log(P(X|z) - log(P(z)))) + logP(X))] \\ &= E_{z\sim Q}(log(P(X|z))) - E_{z\sim Q}(log(Q(z|X) \\ &\quad - log(P(z|X)))) \\ &= E_{z\sim Q}(log(P(X|z))) - KL(Q(z)||P(z)) \end{aligned} \tag{1}$$

where we assume $P(z) \sim \mathcal{N}(0,\, I)$ to make the computation easier.

Notice that we have $logP(X) > logP(X) - KL[Q(z|X)||P(z|X)]$, our task is to maximize the lower bound as to increase the likelihood of the model. In equation 1, the first term is proportional to similarity between the original data and the reconstructed ones (can be evaluated by cross entrophy) and the second term can be directed computed

$$KL(\mathcal{N}(\mu(X),\, \Sigma(X) \times I)||\mathcal{N}(0,\, I))$$

$$= \frac{1}{2}(tr(\Sigma(X))+(\mu(X))^T(\mu(X))-d-log(det(\Sigma(X)\times I)))$$

which can also be treated as the regularization term of the model.

From above, $\mu$, $\Sigma$ are the outputs of the neural network as the encoder part with the input as MNIST data (treated as vector). Both neural networks in the encoder and decoder part are trained through back propagation.

Inspired by the learned MNIST manifold demonstrated in Auto-Encoding Variational Bayes[1], as is shown in Fig.2, the 2-dimension latent space of MNIST data is well ordered according to the number. The fact that the margin in the 2-dimension latent space is somewhat obvious in some extent leads us the trial of using the latent space as the input to do the classification on MNIST data. We finally choose SVM as the classification model because it is mentioned in class and not used by the previous methods.

## 2. Experiment

Here we explain how the methods we mentioned above are applied. The training data has 60000 samples and 10 classes. Each sample is a 45x45 pixel image.

### 2.1. Data Processing

The noise-removing process is implemented by depth first search(DFS). The traversal process start from the first pixel of a sample. If a non-zero pixel remains unvisited, start a searching process recursively until every adjacent pixel of pixels which are visited during the current searching process is zero or visited. Then we can find an “island”. If the size of this “island” smaller than 40, it will be regarded as a noise point and turned into black.

When trimming the noise-removed images, we find that most of the images have more or equal than 21 idle rows and 25 idle columns, where “idle” means the values of the corresponding row or column are all zero. So the images can

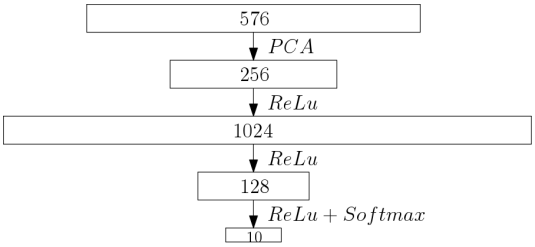



Figure 3. Structure of the fully-connected network model

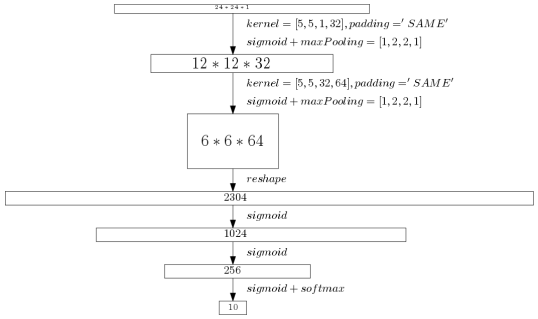



Figure 4. Structure of the convolutional neural network model

be cut into 24x20 to remove most "idle" rows and columns. However VAE models requires the input images to be square, so images are cut into 24x24. When dealing with an image, We first remove all the idle rows and columns of it. If the remaining image has less than 24 rows, calculate the number of rows required to make up for 24, and put half of them to the top of the image and the rest of them to the bottom. If the remaining image has more than 24 rows, randomly choose some rows to delete from the image to make sure the row number is 24. Columns can be dealt with in the same way. Then the digit is in the middle of the image.

### 2.2. Logistic Regression

We use the API `sklearn.linear_model.LogisticRegression()` to build the model and investigate the effect of logistic regression models on the modified version of MNIST images which are reshaped into 2025-dimension vectors. Then we use the data with no noise and centered digits and reshape the data to 480-dimension vectors to build the model again. We also implement the logistic regression with tensorflow. The loss function is the cross entropy loss implemented in the `tf.nn.softmax_cross_entropy_with_logits` where the logits is $\mathbf{\Theta}^T x + b$.

### 2.3. Fully-Connected Network

The structure of the model is shown in Fig.3. First the input images are reshaped into 576-dimension vectors and PCA is applied to reduce the dimension to 256. We use tensorflow to implement a four-layer fully connected neural network. 256 is the size of the input layer.The numbers of neurons in each hidden layer are 1024,128 respectively. And since there are 10 categories, the output layer has 10 neurons. The activation used in hidden layers is ReLU function. Plus, dropout is used to prevent overfitting and the keep probability is set to 0.9. We also employ L2 norm to regularize and smooth the data in the entropy loss function, with the weight decay $\lambda = 0.0001$.

### 2.4. Convolutional Neural Network

Two convolutional/pooling layers are implemented in our model. In the first convolutional layer, there are 32 filters with the size of 5x5x1, and the stride is 1. We use max pooling in the corresponding pooling layer. The pooling kernel is 2x2 and the stride is 2. In the second convolutional layer, there are 64 filters with the size of 5x5x32, and the stride is 1. The second pooling layer has the same structure as the first one. Then we employ three fully connected layers. The number of neurons are 1024, 256, 10(the softmax layer) respectively. The sigmoid function is used as the activation function in the whole network, and dropout with a keep probability of 0.9 is used in fully-connected layers. The structure of the CNN model is shown in Fig.4

### 2.5. Variational Autoencoder with SVM

The data is first normalized by being divided by 255. Both encoder and decoder neural network are implemented with 2 hidden layers. For encoder part, the activation function of the first layer is relu and that of the second layer is tanh. The output layer has the double size of the size of the latent layer, which define the first half as $\mu$ and the second half as $\Sigma$. For decoder part, the activation function of the first layer is tanh and that of the second layer is relu. Sigmoid function is implemented to the output layer to restrict the data within $(0, 1)$. Cross entrophy $E$ is used to measure the different between original data and the reconstructed data and KL divergence $KL$ is calculated by the definition. The overall loss is defined as $-E + KL$ and is optimized by AdamOptimizer with learning rate as $10^{-3}$ and keep probability as $0.9$. The model is trained with batch size as 100 and 100 epochs. The size of the latent space is an optional input, which is a positive interger. The structure of the VAE model is shown in Fig.5

## 3. Result

### 3.1. Data Processing

The search process removed most of the noise points and digits are centered in the image. The result is shown in Figure 6.

### 3.2. Logistic Regression

When the original data is used, the logistic regression model can hardly perform a proper classification, resulting

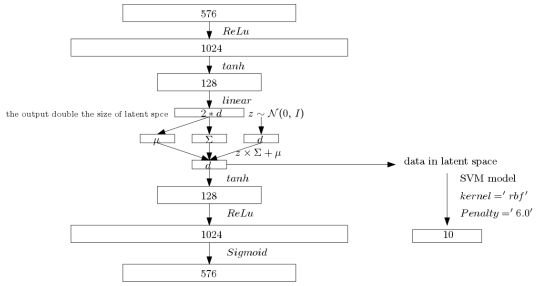



Figure 5. Structure of the variational autoencoder model

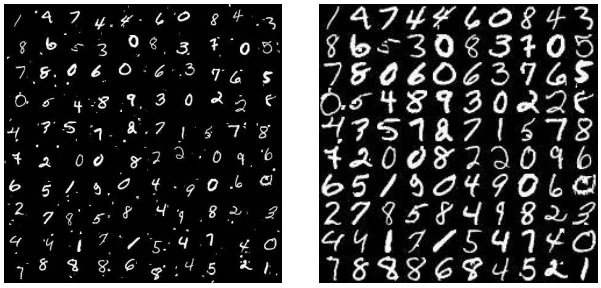


Figure 6. Comparison between original data and processed data

in a 12% accuracy. This means the model basically randomly classfies data. After images are processed, the model built by `sklearn` API can achieve an accuracy of 90%, which indicates that data processing provides a huge improvement for classification. The model built by tensorflow can give us a 87.8% accuracy.

### 3.3. Fully-Connected Network

With the un-processed data, the accuracy is 10% in the first trial. Noticing that the accuracy in training set is relatively higher than that of test set, I set keep probability lower from 1 to 0.9. Still, we modified the structure (size of hidden layer and the number of layers) as well as the learning rate. Finally, accuarcy reached 94.26%. And with the same structure and parameter, the model achieves an accuracy of 98.5% from the processed data.

### 3.4. Convolutional Neural Network

The CNN model performs really well even if the data is not processed, giving us a 95% accuracy, and after employing the processed data, the accuracy improved to 99.43%.

### 3.5. Variational Autoencoder with SVM

The size of the latent space was first set to 2, as is inspired by Fig.2, The result, however, is not ideal. For the un-processed data, the accuracy had only reached 20% and with the processed data, the accuracy only reached 80%.

The dimension of the latent space was then set as 25, and the result was much better. If the original data is used, the accuracy fluctuates around 80%. After the processed data is employed, the accuracy improved to 96.5%. The reconstructed image from the VAE model is shown in Fig.7.

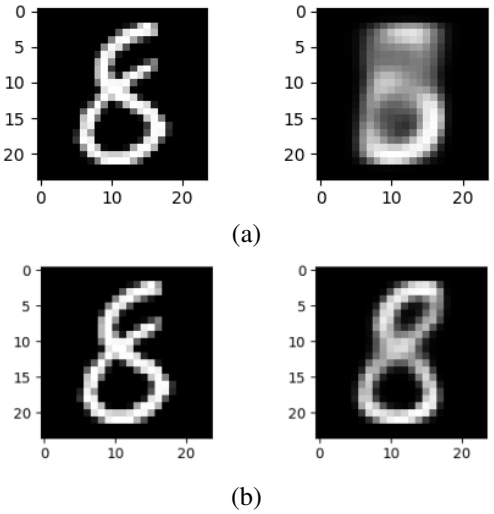



Figure 7. VAE reconstruction result. (a) In the 2-dimension case, the reconstructed image is still blurry. (b) The result becomes much better when the dimension increases to 25. This accounts for the low performance of 2-dimension latent space and the better one of 25-dimension latent space.

## 4. Code Implementation and Appendices

For codes and result screenshots, please refer to Github-MarshalLeeeeee.

## References

[1] M. W. Diederik P. Kingma, *Auto-Encoding Variational Bayes*. 2013.