**Application of classification algorithms to MNIST handwritten letters**

**1. Introduction**

Classification algorithms are useful tools that can be used for image processing, email span filters, and tumor detection. To gain a working understanding of classification algorithms and other machine learning concepts, it is necessary to experiment with the material by oneself. To gain a working familiarity with machine learning algorithms, I have coded up and implemented several classification algorithms towards the goal of classifying handwritten numbers.

Generally speaking, algorithm complexity did not translate to improved model complexity. Only modest improvements, if any, were made by increasing the polynomial degree of fit for regression models. Depending on regularization parameters, increasing the polynomial degree of fit could actually result in overfitting. Similarly, neural network accuracy decreased as the number of hidden layers increased due to overfitting.

**2. Datasets and algorithms used**

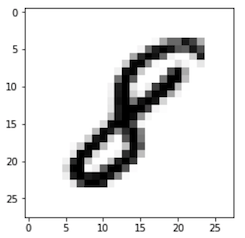
The dataset used in this project is the infamous MNIST dataset[1]. The MNIST dataset consists of 70000 28x28 pixel images of handwritten integer numbers from 0 to 9 (example provided in figure BLAH). Each image has a label that gives the integer number. The goal of working with the MNIST dataset is to create a machine learning algorithm that can accurately classify images into one of the 10 classes of integer numbers from 0 to 9. Each image can be represented as a 28x28 matrix with each element containing a value between 0 and 255 which corresponds to the darkness of the pixel. Alternatively, the 28x28 matrix can be transformed into a 784x1 vector array for easy processing. For n training images, the 784x1 vectors for each image are stacked into a 784xn matrix which serves as my feature matrix. All implemented classification methods essentially consist of an ensemble of binary classifiers for each possible class. An image is classified according to the class of the binary classifier amongst the ensemble that signals the best fit. To facilitate this process, the set of labels is converted to a nx10 binary label matrix rather than a simple nx1 vector of integers. Within the nx10 binary label matrix, an element label of “1” or “-1” is assigned depending on whether image matches the class of the column or not (“1” and “0” for neural networks).

Fig. 1. Example of MNIST image. Image is a 28x28 pixelated image of a handwritten integer.

MNIST designates 60,0000 of the 70,000 images as “training” images and 10,000 images as “testing” images. For this project, the 60,000 “training” images are used as my training dataset and the 10,000 “testing” images are used as my validation dataset. The error rate reported is the ratio of misclassified testing images to total testing images when the trained classifier is used to predict the classes of the validation training set.

Various classification algorithms were tested. The algorithms are split into two broad categories: 1) Regression and support-vector machine algorithms, and 2) Neural network algorithms (summarized in table 2). For the regression algorithms (summarized in table 1), regularization was necessary since number of images, outnumbered the 784 unique features. I used both L1 and L2 regularizations for regressions involving least squares (LS), hinge, and squared hinge loss functions. The 1st- and 2nd- degree polynomial fits were achieved by expanding the feature matrix and weight matrix to include additional intercept and square terms. L1 least squares algorithm was solved via matrix inversion while L2 least squares algorithm implemented stochastic gradient descent. All support vector machine algorithms implemented the LinearSVC function from python sklearn.svm and LinearSVC modules. T

The keras and tensorflow python modules [cite] were used to implement neural networks (network algorithms summarized in table 2). Originally, neural networks were written in python from scratch but the pre-developed code from keras and tensorflow showed an order-of-magnitude improvement in run time and script length. I experimented with different loss functions, activation functions, number of hidden layers, number of epochs, and hidden layer width and observed changes in error rate. The two loss functions I investigated were least squares and categorical cross entropy (CCE), the two activation functions were relu and sigmoid, I investigated 1 and 2 hidden layers for each case.

Table 1. Regression and support-vector machines

|  |  |  |  |
| --- | --- | --- | --- |
| **Loss  Function** | **Regular-**  **ization** | **Loss Eqn. w/**  **Regularization term** | **Fitting**  **Degrees** |
| LS | L1 |  | 0, 1, 2 |
| L2 |  | 0, 1, 2 |
| Hinge | L2 |  | 0, 1, 2 |
| Squared  hinge | L1 |  | 0, 1, 2 |
| L2 |  | 0, 1, 2 |

Table 2. Neural Networks

|  |  |  |
| --- | --- | --- |
| **Loss**  **Function** | **Activation** | **No. Hidden**  **Layers** |
| LS | relu | 1, 2 |
| sigmoid | 1, 2 |
| CCE | relu | 1, 2 |
| sigmoid | 1, 2 |

**3. Results**

**3.1. Regression and support vector machines**

Among the L2-regularized least squares regressions, the 2nd-degree achieved the best accuracy when the entire 60,000-image training set was used. However, when the training set was much smaller (first 1,000 images), lower-order regression was better. The polynomial fit error rates converged near λ=1E7 across all degrees before increasing drastically at higher λ. Beyond this point, the regularization component of the loss function will dominate and the data itself will have less influence on the fit, leading to increasing prediction error.

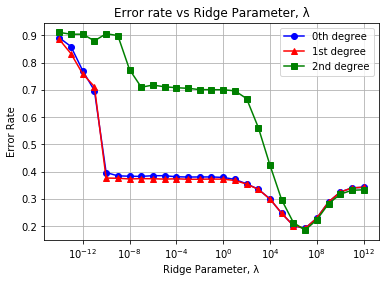
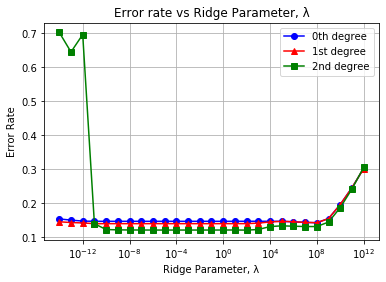


Figure 2. Validation classification error rate for L2-regularized least squares regressions using (a) 60,000 training images and (b) only using the first 1,000 training images

L1-regularized least-squares regression (LASSO regression) trained much faster than L2-regularized least-squares regression due to the absence of matrix inversion in the L1 algorithm. Additionally, L1-regularization was much more effective when working with a limited dataset (fig BLAH b) compared to L2. This makes sense since L1-regularization emphasizes the influence of higher-weight features, captures prominent relationships, and results in a sparse weight matrix. Even for smaller datasets, prominent relationships may be ascertainable. Polynomial degree made little difference in L1-regularized fitting accuracy. This result makes sense since L1 regularization should minimize the influence of extraneous lower-weight features. Adding additional features will not improve performance unless the added features are relatively important.

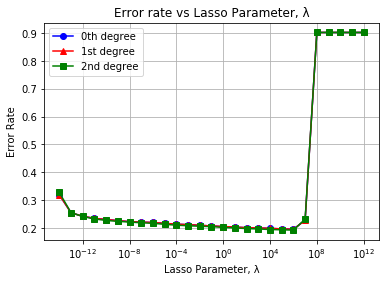
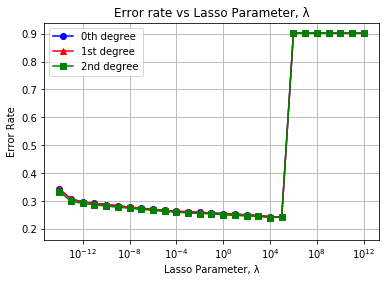
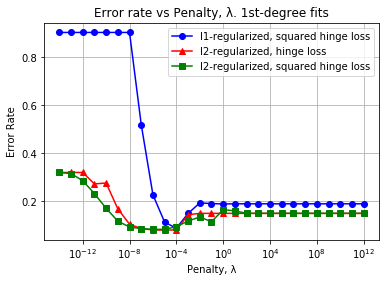
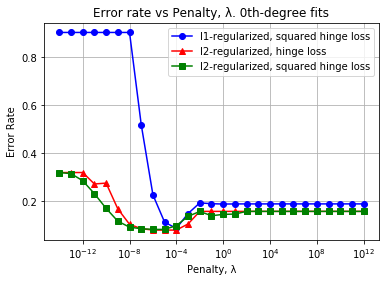
 

Figure 2. Validation classification error rate for L2-regularized least squares regressions using (a) 60,000 training images and (b) only using the first 1,000 training images

The support vector machine algorithms showed notable variability across different regularization schemes (L1 and L2). However, polynomial degree had little impact on fitting performance. Similarly, choosing between hinge loss and squared hinge loss resulted in very similar fits and error rates. Squared hinge loss was chosen in addition to hinge loss since it was compatible with both L1 regularization and L2 regularization in the python sklearn.svm and LinearSVC modules.



(b)

(a)

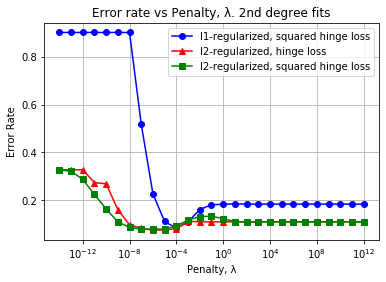


Fig 3. Testing error rate of hinge-loss classifiers for (a) 0th-, (b) 1st-, and (c) 2nd- degree polynomial fitting using various regularization schemes.

(c)

**3.1. Neural Networks**

I experimented with different loss functions, activation functions, number of hidden layers, number of epochs, and hidden layer width and observed changes in error rate. The two loss functions I investigated were least squares and categorical cross entropy (CCE), the two activation functions were relu and sigmoid, I investigated 1 and 2 hidden layers, epoch number ranged from 10 to 60 epochs, and layer width ranged from 10 to 60 nodes. When using the relu activation function, all pixel data was scaled from integer values from 0 to 255 to decimals from 0 to 1 to ensure compatibility. During the sweeps across epoch number, I fixed hidden layer width to 30 nodes. Similarly, during the sweeps across layer width, I fixed the number of epochs to 30 epochs.

epoch number ranged from 10 to 60 epochs, and layer width ranged from 10 to 60 nodes. When using the relu activation function, all pixel data was scaled from integer values from 0 to 255 to decimals from 0 to 1 to ensure compatibility. During the sweeps across epoch number, I fixed hidden layer width to 30 nodes. Similarly, during the sweeps across layer width, I fixed the number of epochs to 30 epochs.

The results of the first loss function, least squares, are shown in fig. 4. All least squares neural net see modest gains in accuracy as epochs or hidden layer width are increased. However, the addition of a second hidden layer does not visibly improve accuracy for the relu models, and in the case of the sigmoid models, accuracy is decreased.

MSE

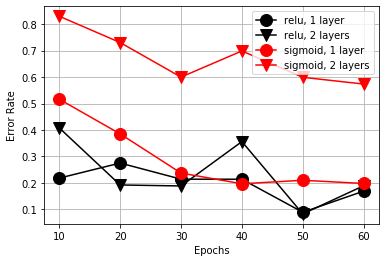
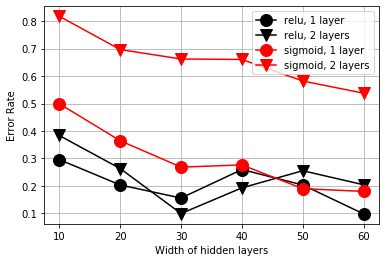
 

Figure 2. Validation classification error rate for L2-regularized least squares regressions using (a) 60,000 training images and (b) only using the first 1,000 training images

Using the categorical cross-entropy loss function in tandem with the sigmoid activation function resulted in significant improvements in accuracy. In terms of accuracy, run time, elegance, ease of use, etc., the model implementing a single sigmoid hidden layer with the categorical cross-entropy loss was the best model among all models evaluated in this project. Of the non-

However, relu was incompatible with CCE even with scaling, resulting in predictions that were generally equal to random guesses.

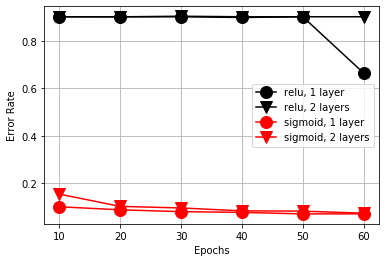
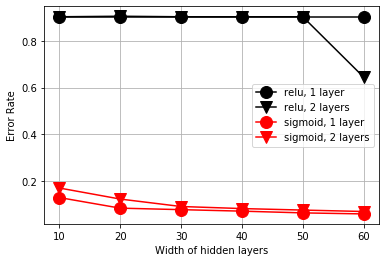
 

Figure 2. Validation classification error rate for L2-regularized least squares regressions using (a) 60,000 training images and (b) only using the first 1,000 training images

**4. Strengths and limitations of methods**

Broadly speaking, Performance could likely be improved by centering and scaling images or by splitting the “7” class into two separate classes, one where class contains 7’s with the central cross (7“ink free” font) and one class is without the central cross (7). Additionally, pixel proximity was not considered in the model. An approach that conserves the relationships between nearby pixels would also likely improve performance.

Preprocessing is an important step to building a classifier. Thankfully, the MNIST dataset is well known for requiring little-to-no preprocessing to test machine learning algorithms https://medium.com/tebs-lab/how-to-classify-mnist-digits-with-different-neural-network-architectures-39c75a0f03e3.

For this reason, preprocessing was avoided in order to focus on algorithm implementation. Similarly, cross-validation can result in a more-robust assessment but is not necessary for achieving tangible results with the MNIST dataset.

Neural network implementation required a bit more finesse and contained more pitfalls to be wary of compared to the regression methods. For instance, the relu activation function and the sigmoid activation function required different data scaling to perform effectively. The second loss function that I used, categorical cross-entropy, could only be used in a “one-hot” encoding, or the binary label matrix that I had settled upon earlier. Additionally, categorical cross entropy is not directly compatible with relu, and would require additional operations, such as performing a softmax operation beforehand.

Furthermore, the addition of more hidden layers actually hindered model accuracy rather than improved it in the case of sigmoid. While this may be suggestive of overfitting, I also found increases in error rate in the training set when moving from 1 hidden layer to 2.

checked the error rate of the training data

Additionally, adding more hidden layers hindered performance

**5. Conclusion**

Surprisingly, the various methods achieved roughly similar error rates in the validation overall.

**6. References**