Linear Least Squares MNIST Handwritten Digits Classifier

ECE 174 Fall 2021 Mini Project 1

Hao Le

NOTE: source code includes main.py script which is meant to be ran. It will demonstrate all the experiments and produced figures outlined in this report. However, the number of testing and training examples, as well as the number of features, has been reduced in the script to allow faster run time for demonstration purposes to the grader. To reproduce results, refer to the comments that specify the original values.

Problem 1

Normal equation proof

a.

The normal equation is as follows:

$$A^H y = A^H A x$$

The range space of A, R(A), is the set of vectors that are the result of all possible linear combinations of its column vectors. Thus, it follows that the $R(A^H)$ is the superset to $R(A^H A)$, because by multiplying by A, the column vectors of A^H are being linearly combined in some manner. However, it can never "break" out of $R(A^H)$ by definition. In other words, a range space can only stay the same, or shrink. We've shown that:

$$A^H A \subseteq A^H$$

Without proof, we know that the rank of $A^H A$ is equal to A^H, or equivalently $\dim(A^H A) = \dim(A^H)$. Thus, it is implied that $R(A^H A) = R(A^H)$.

Going back to the normal equation, since the left-hand side is some vector in $R(A^H)$, the right-hand side must also be the same vector in the same range space – this is always possible. All possible values for x will span the RHS vector across $R(A^H A)$, so there must be at least one x that results in the same LHS vector.

b.

If $A^H A$ has linearly independent columns, then there can only be one unique solution. This is because the any vector in $R(A^H A)$ can be decomposed to a minimum of $\dim(A^H A)$ vectors, and if $A^H A$ has exactly $\dim(A^H A)$ column vectors, then there can only be one unique set of $\dim(A^H A)$ column vectors that will combine to equal the left-hand side vector. In other words, there can only be one unique x to every y.

If $A^H A$ has linearly dependent columns, then for every LHS vector, there are infinite ways to yield the same vector through combinations of $A^H A$'s column vectors. This is because there will be at least one pair of dependent vectors that can have infinite number of linear combinations for any given vector. In other words, for every y, there are infinite x.

c.

If rank(A) = n, then A^HA is a full rank nxn square matrix that is invertible. The normal equation can be uniquely solved:

$$x = (A^HA)^{-1} A^Hy$$

If A is rank deficient i.e., rank(A) < n, then there are infinite solutions. We can perform row reduction to form a parametrized solution by which we can choose an arbitrary parameter to get a specific solution. However, computationally this is complex.

We can use pseudo inverse to find the solution with the smallest norm[1].

To find the pseudo inverse, we start by performing singular value decomposition on A^HA:

$$A^HA = U\sum V^H$$

Both U and V are unitary matrices, hence $UU^H = U^HU = VV^H = V^HV = I$

$\sum$ is a diagonal matrix where the diagonal elements are the singular values of A^HA. If A^HA is full rank, where rank(A^HA) = n, then all the diagonal elements would be non-zero. If A^HA is rank-deficient, there would be zero diagonal elements. $\sum$ is only square if A^HA is square also – in this case we can find a true inverse $\sum^{-1}$ if A^HA is full rank. But if A^HA is rectangular and/or rank-deficient, we have to settle for the pseudo-inverse of $\sum$, $\sum^+$ where reciprocal is performed on the non-zero diagonal elements. In short, pseudo-inverse of a matrix A *tries* its best to generate the matrix's inverse where A^HA's elements are approximately equal that to I.

If we pseudo-invert the SVD:

$$(A^HA)^+ = V^H\sum^+U$$

Now the normal equation with infinite solution can be solved, with x being the minimum norm solution

$$x = (A^HA)^+ A^Hy$$

Using pseudo-inverse is a way to **simply the coding** of solving an underdetermined solution. Picking an arbitrary solution will always work.

[1] https://en.wikipedia.org/wiki/Moore%E2%80%93Penrose_inverse "Minimum norm solution to a linear system"

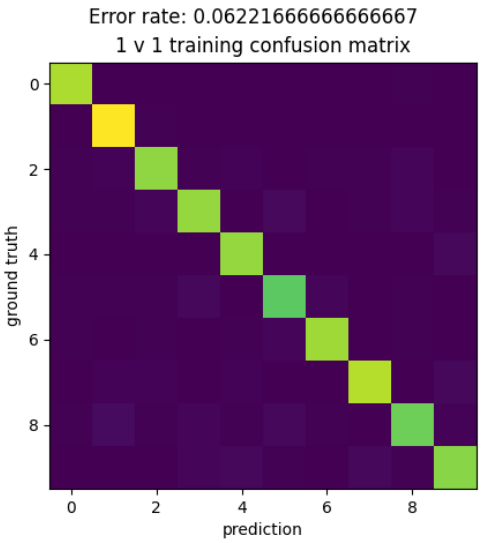
Problem 2

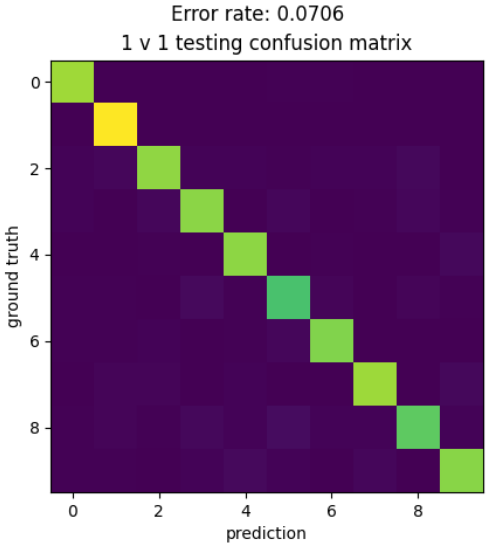
Performance difference between one-versus-one and one-versus-all multiclass classifiers

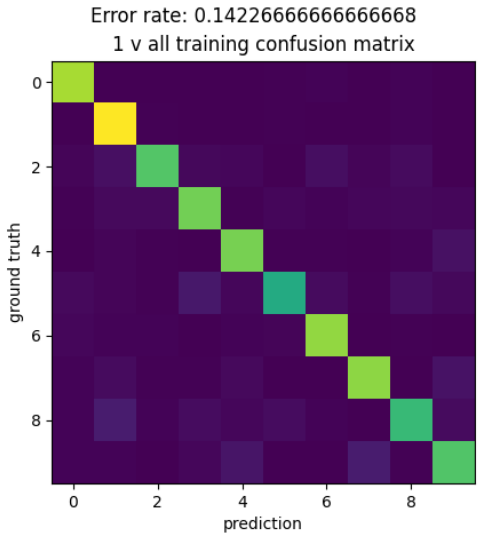
Using the principle of the normal equation, we can now build a binary classifier that maps an input image vector of an MNIST digit and output a prediction from 0-9. Explanation of underlying mathematical principle is commented in source code.

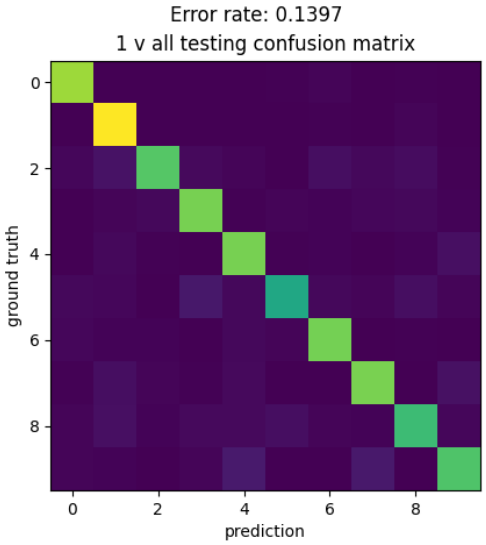
We then train two classifiers: one-versus-one (1V1), and one-versus-all (1VA) on clean training data. For MNIST, a total of 60,000 examples were used.

For this experiment, both training and testing errors (on 10,000 test images) were reported, as well as the confusion matrices, for both classifiers, as shown:









Some observations made:

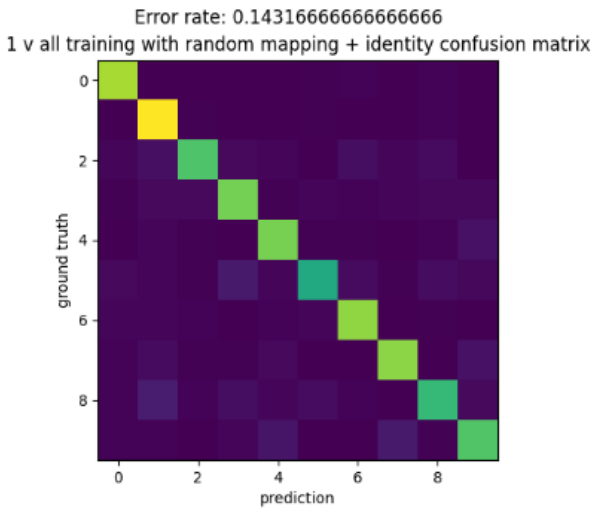
- 1VA classifier performed slightly better on testing than training set, which is not expected but the small discrepancy can be explained by somewhat luck on which data points were in each set.
- 1V1 classifier performed slightly better on training than testing data, which is expected. 1V1 outperformed 1VA classifier by nearly two times. This can be explained by the rigor of 1V1's pairwise selection criteria. It better distinguishes digits that are similar, because the false lookalike eventually loses in votes.
- Both classifiers generalize well to the test set since the difference in performance is not significant
- We see from the confusion matrices that in 1VA, 9 is most predicted on 4 images, as well as 5 on 3, and 9 on 7. This makes sense since handwriting of these digits can be similar. However, there is very little confusion present on one versus one.
- Looking row-wise, we can tell which digits are most confused for others by the presence of predictions outside of the diagonal. In 1VA, 9 is the hardest to recognize, and 0, 1, 3, 6 appears to be the easiest. Perhaps this is due to their unique features.

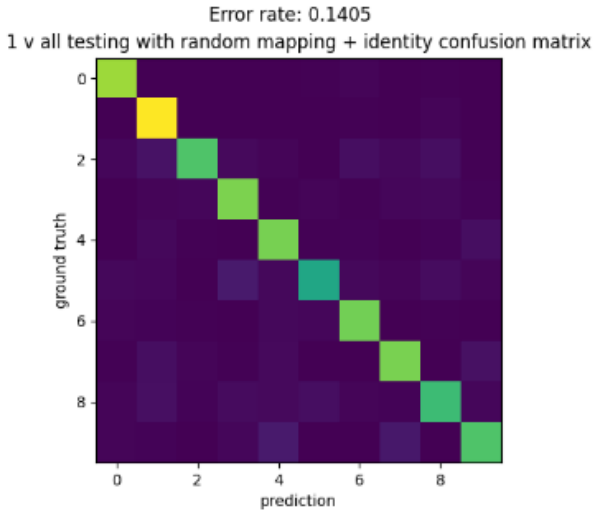
Problem 3

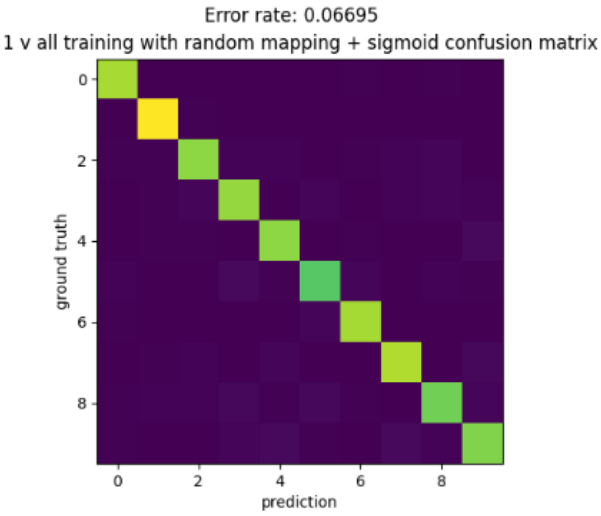
Effect of random mapping and nonlinearity

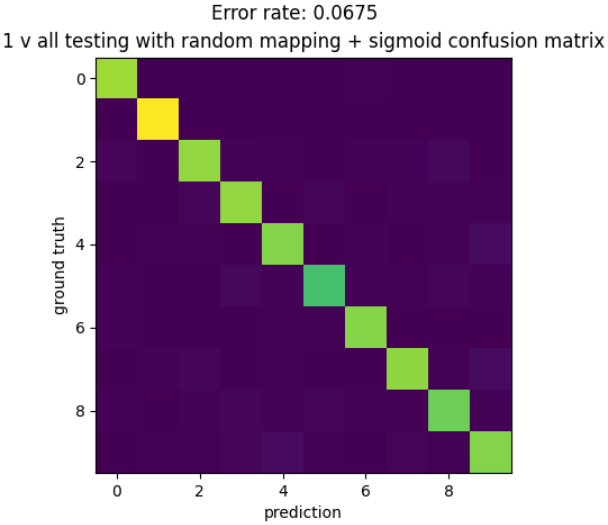
For this experiment, we observe the effect on performance feature mapping and nonlinearity has on classifiers. We will use a Gaussian distribution to randomly generate a feature-mapping matrix and an offset vector. Additionally, the mapped features can be put through a nonlinear function – we experiment with sigmoid, sine, and ReLU. The base trial does not use nonlinearity, and instead passes through the identity function.

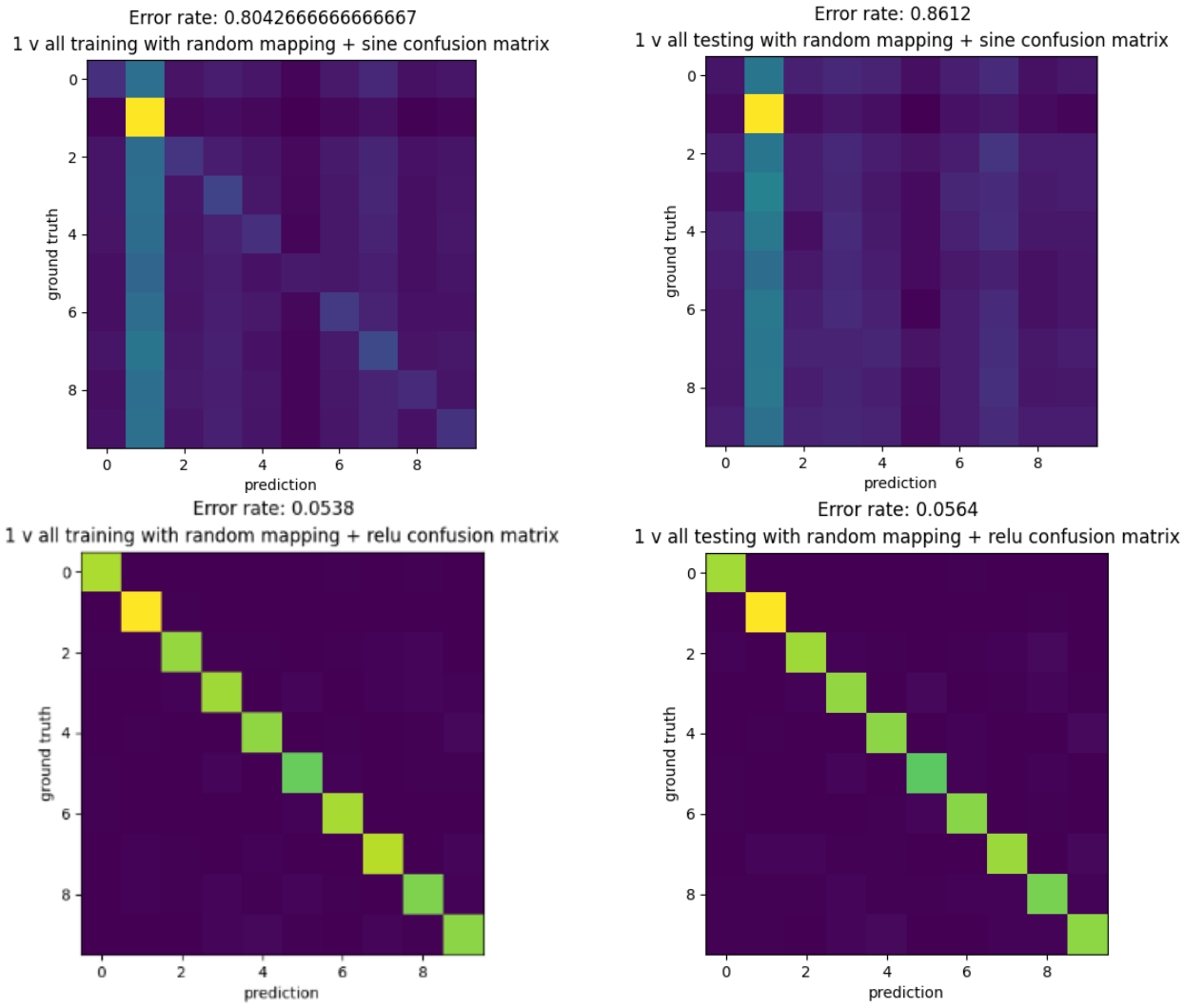
For the number of features, we are keeping a constant 1000. A 1VA classifier is used and trained on 60,000 mapped images. Both training and testing errors (on 10,000 mapped testing images) were reported, as well as the corresponding confusion matrices.









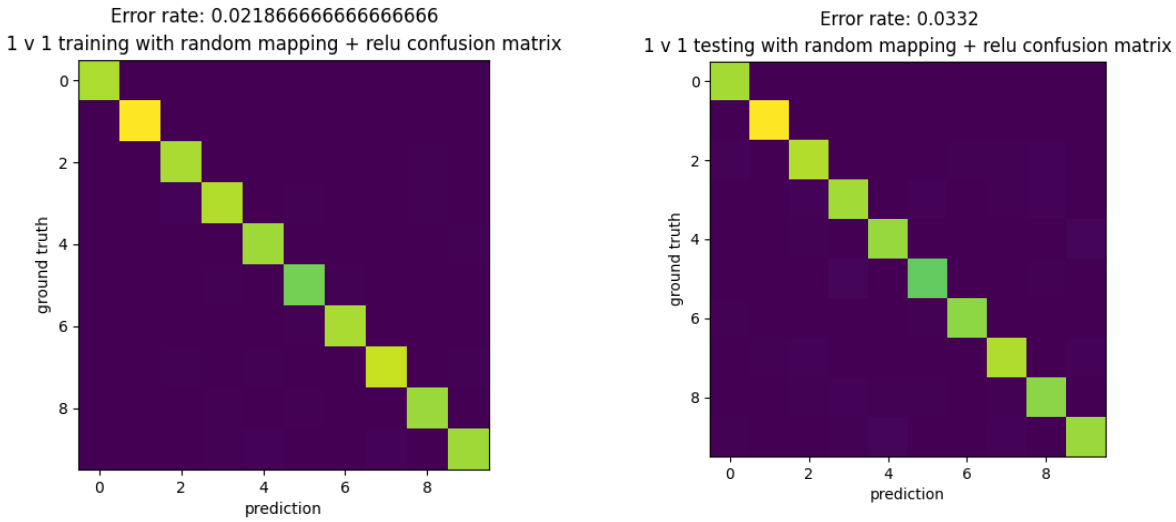


Some observations made:

- All the classifiers, except the one using sine nonlinearity, generalized well to the test set since their performances did not change significantly from training performance.
- We see that random mapping without nonlinearity does not change performance, compared to learning in the pixel domain, significantly in the identity trial. The confusion matrices also exhibit similar behaviors.
- Sine nonlinearity degrades the performance the most on both training and testing. A poor performance on training data set implies that the mapped features of the images follow no clear pattern – in other words the l2-norm of the error of the training data set is very large. It is worth noting that the classifier did manage to correctly predict some training examples, as shown by the diagonals in the confusion matrix.
- Interestingly, the sine classifier's confusion matrices indicate that a prediction of 1 was likely to be made on any image, hence the classifier only does well with images of 1's. This implies least squares weights were biased towards 1 during the training process, likely caused by the boundedness of sine.
- Sigmoid nonlinearity adds about 7% to performance, but ReLU gives the most boost.

We can conclude that feature mapping alone does not boost performance. However, when coupled with unbounded nonlinearity, we can reap the benefits.

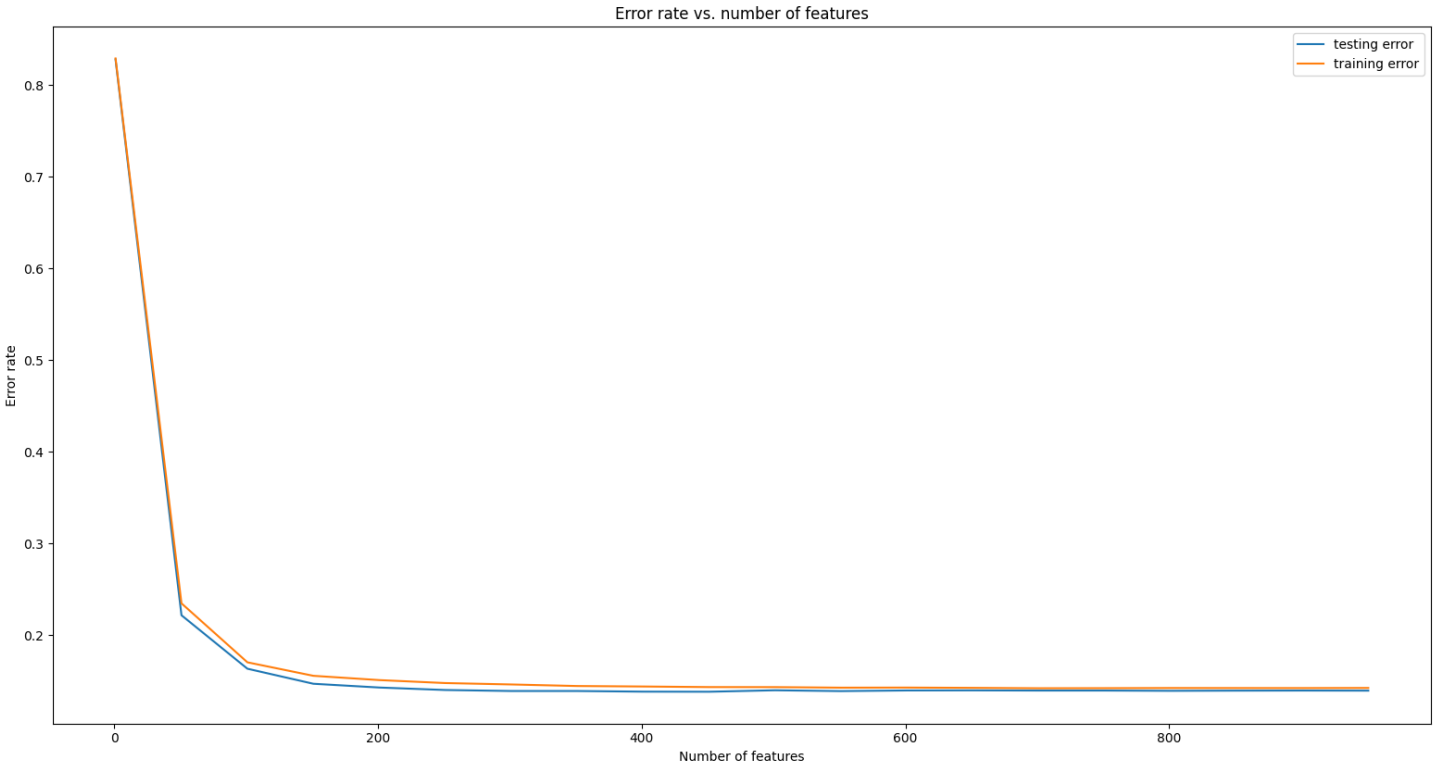
From experiments in problem 2, it was clear that a 1V1 classifier outperforms a 1VA. Thus, in an additional experiment, we evaluate 1VA with ReLU, expecting the best performance yet for any classifier.



When switched to the 1V1 classifier, we get the best test performance (visualized by the clean diagonals of the confusion matrices) of 3% error rate, and our expectations were met.

Effect of varying number of features on performance

For this experiment, we use 1VA classifier with no nonlinearity. We vary the number of features, from 1 to 1000, in increments of 50, and observe the change in training and testing performance.



Some observations made:

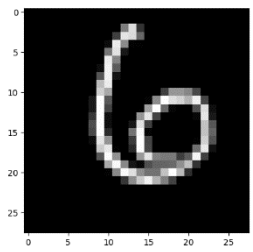
- By adding more features, we reduced both training and testing error (and average error since both errors are very similar).
- The performance gain reached a plateau after around 300 features; this can be considered diminishing returns since training time increases for a mapper with more features.

It can be concluded that the more features, the better performance. This can be explained by the fact that more features mean more degrees of freedom to encode nuances of different digits. Less features can “group” certain digits together in the feature space, leaving the classifier prone to confusion.

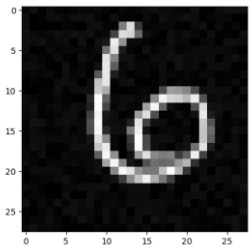
The number of features can be chosen such that the performance is just on the edge of plateauing. Ideally, we want as many features as possible, but this comes at the cost of computational complexity of mapping.

Robustness of classifier to noise

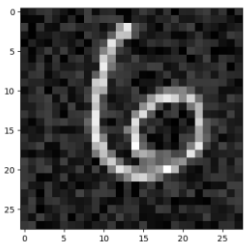
In this experiment, we want to observe how the performance of a 1VA classifier, trained on clean, unmapped data, degrades with noisy test images. For the addition of noise, we add the clean image vector to a noise vector, whose elements are generated via a uniform distribution limited between -1 and 1. Noise amount is a parameter that determines the maximum value the norm of the noise vector can take on. A visualization of increasing noise is shown:



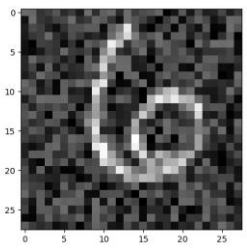
No noise



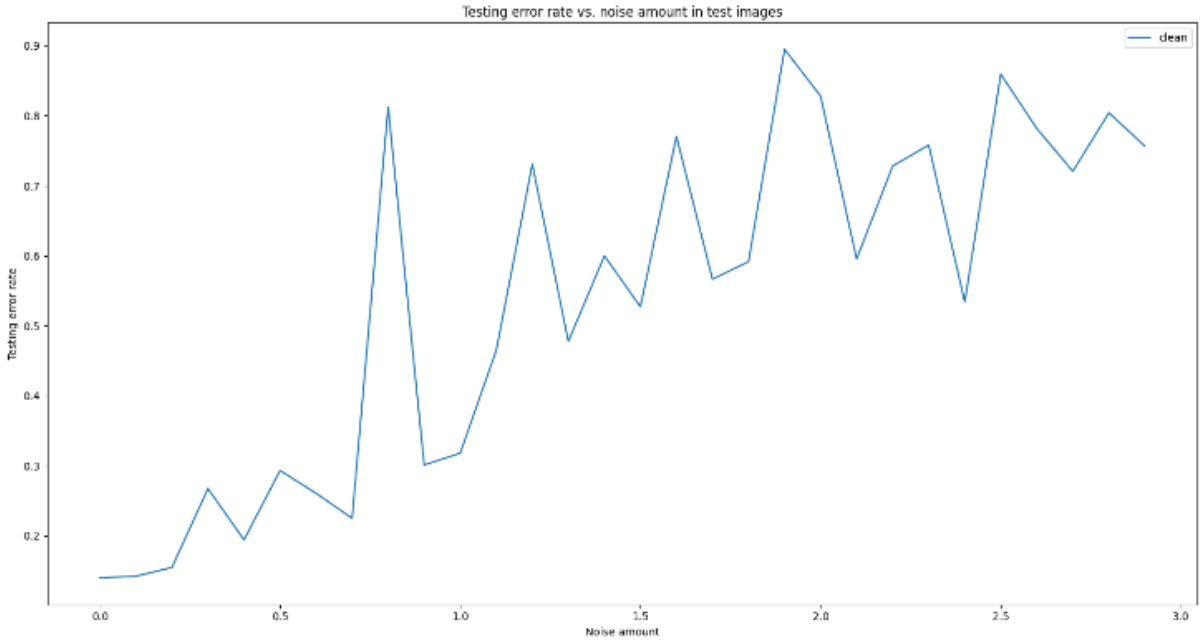
Noise amount of 1



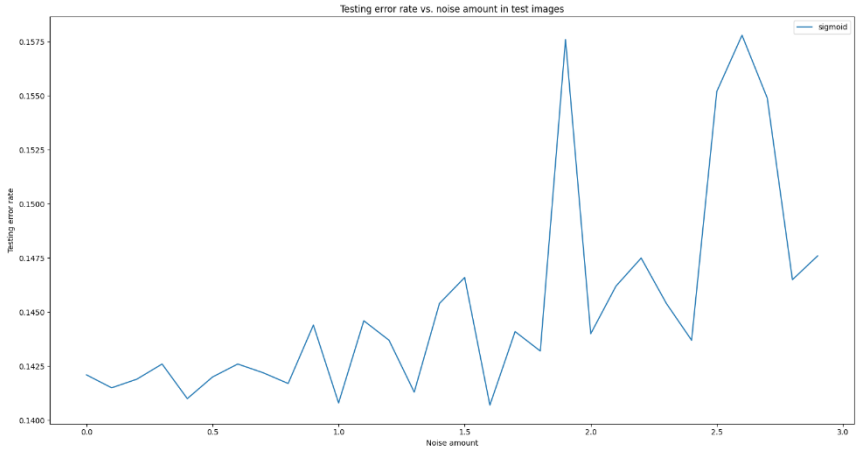
Noise amount of 5

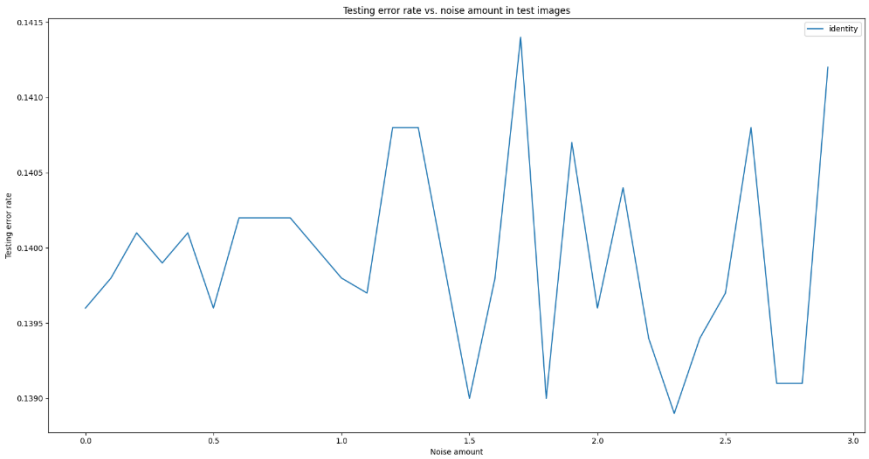


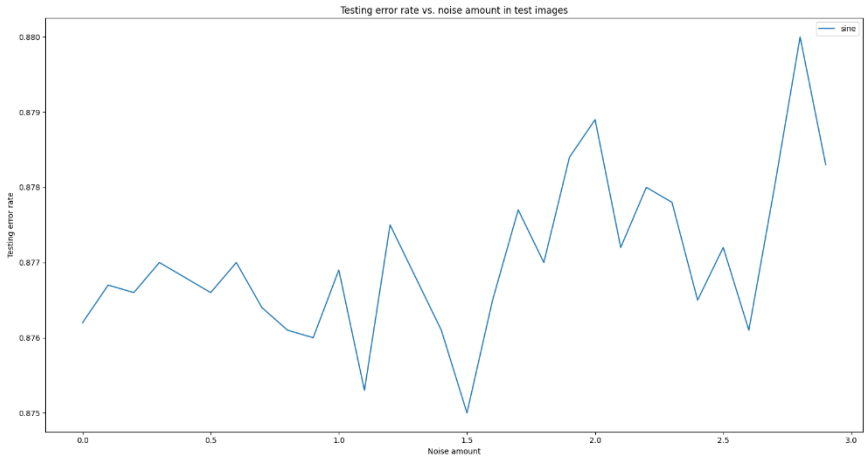
Noise amount of 10

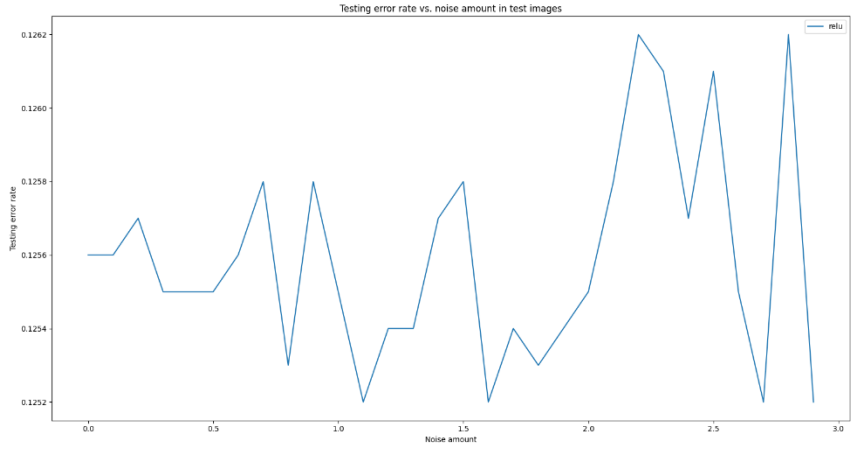


From the graph, we observe a linear increase in the error rate as noise amount increases. In the next experiment, we add random mapping and different nonlinearities to see if this adds resilience to the classifier. In other words, will this prevent the error rate from increasing as noise increases? The results are shown:


Testing error rate vs. noise amount in test images
sigmoid
Testing error rate
Noise amount
0.1400
0.1425
0.1450
0.1475
0.1500
0.1525
0.1550
0.1575
0.0
0.5
1.0
1.5
2.0
2.5
3.0


Testing error rate vs. noise amount in test images
identity
Testing error rate
Noise amount
0.1390
0.1395
0.1400
0.1405
0.1410
0.1415
0.0
0.5
1.0
1.5
2.0
2.5
3.0


Testing error rate vs. noise amount in test images
sine
Testing error rate
Noise amount
0.875
0.876
0.877
0.878
0.879
0.880
0.0
0.5
1.0
1.5
2.0
2.5
3.0


Testing error rate vs. noise amount in test images
relu
Testing error rate
Noise amount
0.1252
0.1254
0.1256
0.1258
0.1260
0.1262
0.0
0.5
1.0
1.5
2.0
2.5
3.0

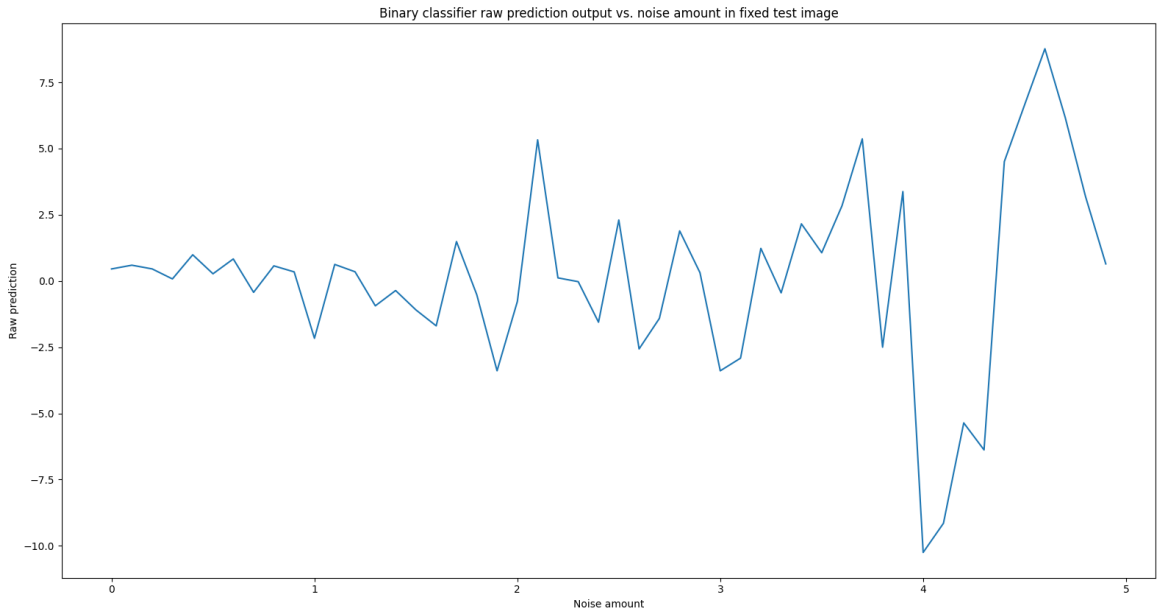
Some observations made:

The addition of feature mapping and nonlinearities made the classifier more robust and resilient to noisy images in terms of the moving average of error rates. The error rates fluctuate randomly as images get noisier, implying that at some point the classifier's predictions are done by chance.

- The previous observation can be confirmed by the fact that both identity and ReLU classifiers appear to be doing better as test images get noisier, and it is the opposite for sine and sigmoid. However, the fluctuations get more extreme. It can be extrapolated that noisier images will get predictions with no consistency.

Effect of noise on binary classifier raw prediction

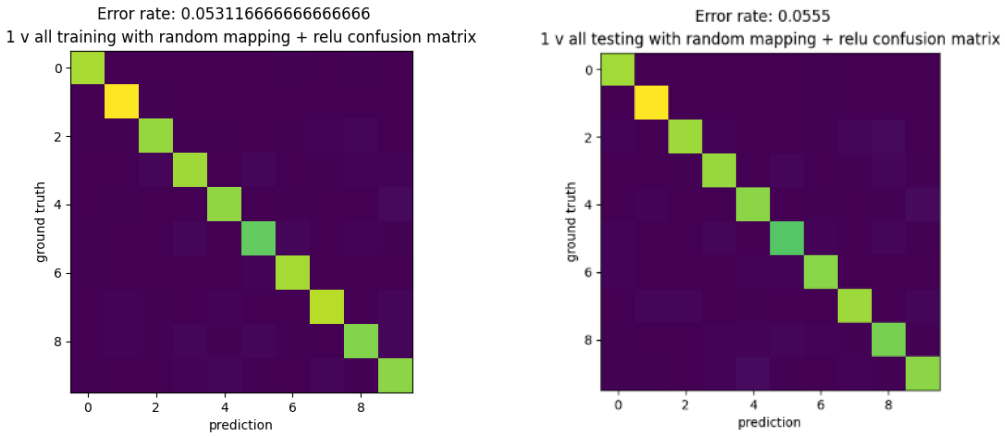
For this experiment, we show a binary classifier an image of increasing noise with the digit class it was trained to target – in other words it is supposed to predict a 1. Since sign(a) nonlinearity is not used, the graph shows that the initial prediction for the clean image (noise amount of 0) of the target digit class is non-zero but not absolutely 1, which is expected.



As noise increases, the prediction stays resilient by being non-zero. Then the classifier breaks down after noise amount of 1, becoming erratic. Predictions take on more extreme values and is random in nature. This explains the error rates for entire test sets seen in previous experiments.

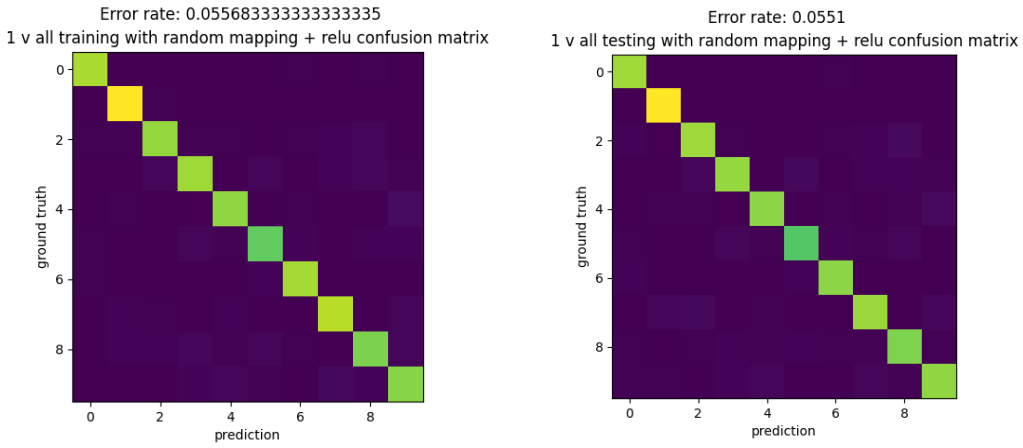
Effect of distribution of random mapping on performance

In this experiment, we vary the parameters of the Gaussian distribution that is used to generate the random mapping matrix and vector and observe the training and testing of the classifier. Additionally, we also changed the distribution to uniform.

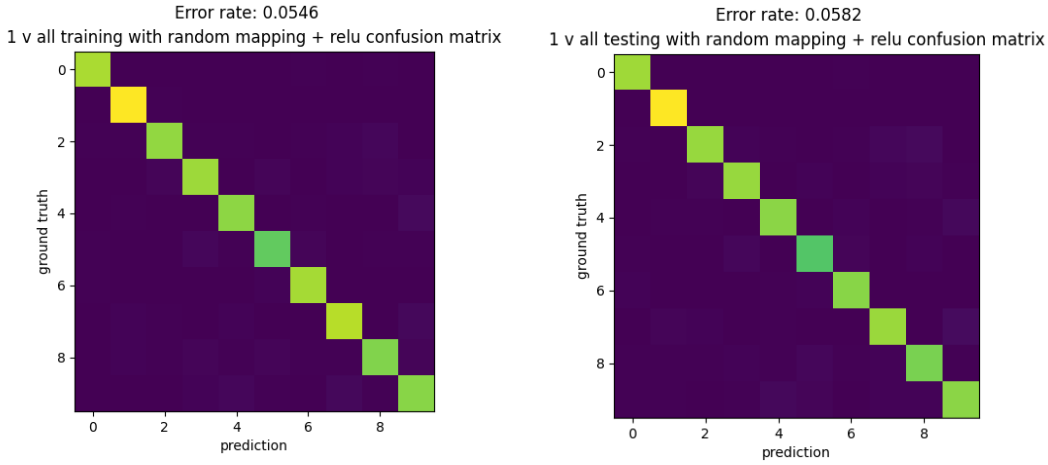


Random mapping of Gaussian distribution - average of 0, and std. of 100

It is observed that varying the Gaussian parameters did not have significant impact on training and testing performance.



Random mapping of uniform distribution – max. of 1, min. of -1



Random mapping of uniform distribution – max. of 100, min. of -100

A similar outcome is observed with uniform distribution – both trials of different limits had no significant impact on performance.

It can be concluded that the distribution used when generating the mapping matrix and vector has no effect on the classifier performance.