MSDS 458: Artificial Intelligence and Deep Learning

Assignment 1

September 30, 2025

Understanding the building blocks of neural network trained on MNIST dataset

Yingyu Mao

Abstract

This study explored the interpretability of Fully Connected Neural Networks (FCNNs) on the MNIST digit classification task. By systematically varying the number of hidden nodes, I showed that performance improved up to ~32 nodes, after which gains diminished, and many nodes learned noise. Visualization of activations and weights revealed that insufficient nodes cause overlapping latent representations for similar digits, explaining misclassifications. I further demonstrated that FCNNs automatically learn relevant features, rendering preprocessing steps like PCA and feature selection unnecessary or detrimental. These findings illustrate the fundamental learning dynamics of FCNNs, the simplest architecture of neural networks. It provides tools to study the interpretability of more complex architectures.

Keywords: Fully Connected Neural Networks, MNIST, Interpretability, Feature Learning

1.0 Introduction

The ability to interpret handwritten digits by computers opens the door to bridging the human world of analog information with the digital realm of computers. Handwritten digit recognition (HDR) systems are widely deployed in real-world scenarios. For instance, banks use HDR to automatically read the numerical amounts on checks, postal services use it to sort mail by reading ZIP codes, and tax agencies use it to process digitized forms. This automation leads to significant improvements in speed, accuracy, and efficiency, reducing human error and operational costs.

One of the early challenges in handwritten digit recognition is digit classification. It takes an image of a single digit (0-9) and identifies which number it represents. While seemingly simple for humans, this task is complex for machines due to the variations in handwriting styles, sizes, orientations, and noise. While traditional machine learning algorithms like random forest can achieve relatively good accuracy, neural networks provide a new approach for this task as they are able to learn the latent features that define each digit, such as curves, lines, and intersections, without being explicitly programmed with rigid rules as used in traditional machine learning models. They have the potential to push the prediction accuracy beyond traditional machine learning algorithms.

The quintessential example of this application is the MNIST dataset (Modified National Institute of Standards and Technology database). With 70,000 labeled images of handwritten digits, it provides a standard benchmark for training and evaluating classification algorithms. In this paper, I will train several fully connected classifier neural networks for digit classification with the goal of understanding the inner workings of FCNNs.

2.0 Literature Review

Research in understanding the inner works of fully connected neural networks has been focused on their representational power, optimization and training dynamics, and interpretability and representations.

Representational power addresses what kind of function can be approximated by FCNNs. Cybenko & Hornik et al. established the Universal Approximation Theorems. Cybenko's work (Cybenko 1989) demonstrated that a single hidden layer FCNN with sigmoidal activation functions could approximate any continuous function on a compact subset of R^n. Hornik et al. (Hornik, Stinchcombe and White 1989) generalized this result, showing that the universal approximation property is not due to the specific activation function but to the multi-layer structure itself.

The training dynamics of FCNN greatly affects the learning outcome due to the nonconvexity of the loss function for most complex datasets. Choromanska et al. (Choromanska, et al. 2015) modeled the loss landscape of deep non-linear networks (very large number of nodes) using concepts from statistical mechanics: spherical spin glass model, Hamiltonian, phase transition, and mean-field theory. They theorized that while the landscape is filled with local minima, most of the low-lying minima are below a common energy barrier, suggesting that finding any of them is sufficient given large numbers of parameters and layers. Multiple optimizers were developed to navigate the loss landscape efficiently, including momentum (Polyak 1964), Nesterov accelerated gradient (Nesterov 1983), AdaGrad (Duchi, Hazan and Singer 2011), RMSProp, Adam (Kingma and Ba 2014).

The most common approach to understanding feature representation in FCNNs is through visualizing features. Erhan et al. (Erhan, et al. 2009) pioneered techniques for visualizing what neurons in deep networks respond to. By using activation maximization, finding the input that maximally activates a given neuron, they showed that lower layers learn simple features (e.g., edges, colors), while higher layers learn increasingly complex and abstract patterns.

3.0 Methods

In this paper, I focused mostly on understanding the representations of FCNNs to interpret the learning outcome. The main readouts I chose were the activation values of the neurons and their visual representations. The following conditions were used in the experiment design.

<table>
<tr><th>Experiments</th><th>Data preprocessing</th><th>FCNN architecture</th></tr>
<tr><td>1-node hidden layer</td><td rowspan="8">Flattened the 28x28 digit images into 784 vectors
Rescaled the pixel intensities between (0,255) to (0,1)
One-hot encoded the class labels to generate (10,) predicted variable</td><td rowspan="7">Input(784,)
Dense(1-128, ReLU activation, L2 regularization=0.001)
Dense(10, Softmax activation)</td></tr>
<tr><td>2-node hidden layer</td></tr>
<tr><td>4-node hidden layer</td></tr>
<tr><td>8-node hidden layer</td></tr>
<tr><td>16-node hidden layer</td></tr>
<tr><td>32-node hidden layer</td></tr>
<tr><td>128-node hidden layer</td></tr>
<tr><td>32-node hidden layer1
16-node hidden layer2</td><td>Input(784,)
Dense(32, ReLU activation, L2 regularization=0.001)
Dense(16, ReLU activation, L2 regularization=0.001)
Dense(10, Softmax activation)</td></tr>
<tr><td>PCA-preprocessing</td><td>Performed dimension reduction of the input variables with PCA
Used the first 154 principal components to represent the data as input for the FCNN</td><td>Input(154,)
Dense(32, ReLU activation, L2 regularization=0.001)
Dense(10, Softmax activation)</td></tr>
<tr><td>Feature selection with random forest</td><td>Identified the top 70/300 most important features of the dataset
Used these 70/300 features as the input data</td><td>Input(70 or 300,)
Dense(32, ReLU activation, L2 regularization=0.001)
Dense(10, Softmax activation)</td></tr>
</table>

4.0 Model Training Results

All the training performed in this paper used Nadam optimizer, a variant of Adam in combination with the Nesterov trick. The loss function used was categorical cross entropy. Weights of the hidden layers were initiated according to the He normal distribution rules, as it has been demonstrated to be effective to control equal variance of the input and output of a layer in combination with ReLU (He, et al. 2015).

The learning process was controlled by a learning rate scheduler:

```
tf.keras.optimizers.schedules.ExponentialDecay(initial_learning_rate=1e-2, decay_steps=500, decay_rate=0.95)
```

Early stopping was also used to prevent overfitting:

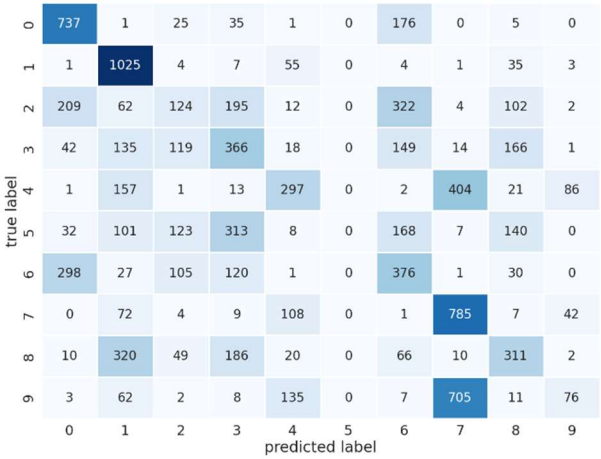
```
tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=5)
```

4.1 Experiment 1: NN with 1-node hidden layer

This architecture collapses all 784 dimensions of the input into a 1-dimension representation. The model is underparameterized. The learning outcome was poor as indicated by the various classification metrics shown below on the left and confusion matrix on the right:

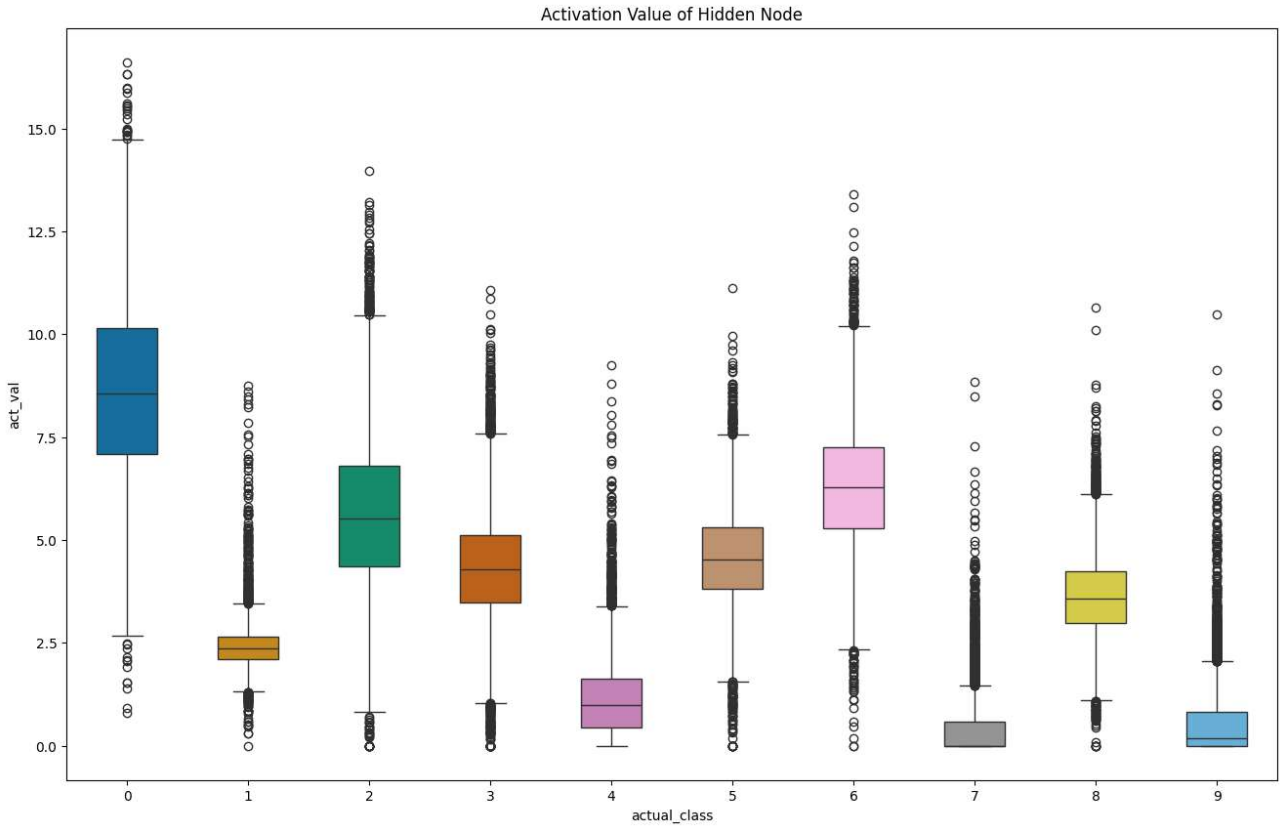
Classification Report

	precision	recall	f1-score	support
0	0.55	0.75	0.64	980
1	0.52	0.90	0.66	1135
2	0.22	0.12	0.16	1032
3	0.29	0.36	0.32	1010
4	0.45	0.30	0.36	982
5	0.00	0.00	0.00	892
6	0.30	0.39	0.34	958
7	0.41	0.76	0.53	1028
8	0.38	0.32	0.35	974
9	0.36	0.08	0.12	1009
accuracy			0.41	10000
macro avg	0.35	0.40	0.35	10000
weighted avg	0.35	0.41	0.36	10000

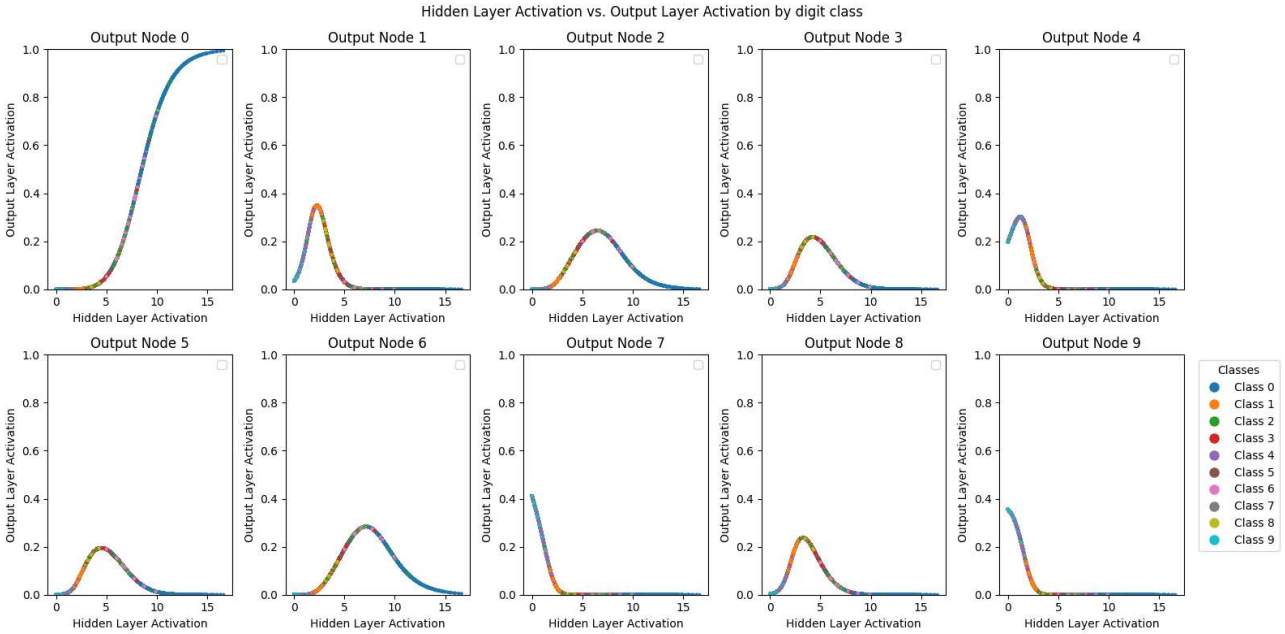


The overall accuracy was only 0.41. None of the digits were predicted to be 5, which were recognized mostly as 3. The model was confident about digit 1 with the highest recall but could not distinguish well between 7 and 9, 0 and 6, 1 and 8, 6 and 2.

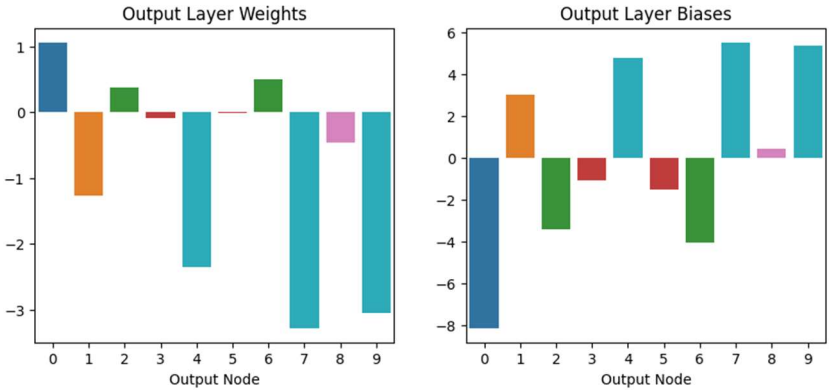
The activation values outputted by the hidden node of each data point determined how the output neurons interpret the signal. The distributions of these activation values are shown below. From this plot we see overlaps of distributions between 4, 7 and 9, 5 and 3, 6 and 2, which explains the prediction outcome described above. The only two distributions that have the least overlap with other classes are 0 and 1, corresponding to the higher recalls for these two classes.



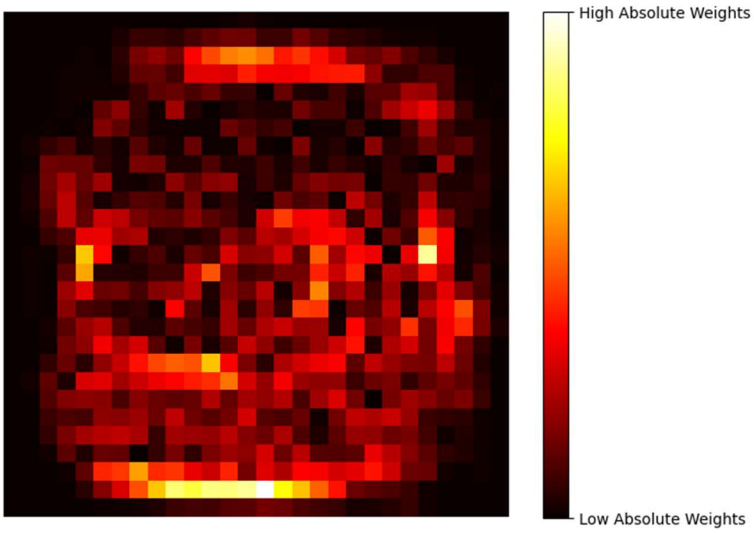
The activation values of the hidden layer are inputs to the output layer, which contains one node per prediction class. Weights and bias between the output nodes and the hidden node further transforms the incoming activation values into logit for each class. The SoftMax activation function converts the logits into probabilities. After visualizing the relationship between the output (activation values) of the hidden node and the output of the 1-node NN model (the predicted class probabilities), it is noticeable that the output nodes 3 and 5 generated similar probability distributions. Nodes 7 and 9 also shared very similar distributions, with node 4 close enough to the two. Nodes 2 and 6 were also indistinguishable. Interestingly, node 1 had relatively distinct output distribution, and the class 1 data points (orange) all generated high probability in this dimension. Although node 0 has wide probability distribution for all data points, class 0 data points (blue) all seemed to have high probability output at this node.



The pattern of the overlapping and distinct distributions replicates that of the activation output of the hidden node between different classes and is a good indicator of the prediction outcome of each class label. This is an interesting behavior in the sense that if lower layers fail to learn the distinct feature between classes, similar activations of the previous layer will prohibit learning by subsequent neurons, which are output nodes in this case dedicated to predicting certain class labels. This can be explained by how gradients are calculated during backpropagation: $\frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}o_i$where o_i is the output of the previous neuron (the hidden node in this case), E is the error signal, o_j is the output of the current neuron (the output neuron in this case), $\frac{\partial E}{\partial o_j}$ is the partial derivative of the loss with respect to the output, $\frac{\partial o_j}{\partial net_j}$ is the partial derivative of the output with respect to the weight transformation of o_i before apply the activation function. Similar outputs from the previous layer (o_i) resulted in similar gradients for the receiving neurons, which converged on similar weights and biases (between class 2 and 6, 3 and 5, 4,7, and 9) as illustrated below. In other words, the error signals of the confused classes were not propagated back to the weights between the input and the hidden node, due to lack of hidden dimensions to distinguish the error signal.



Plotting the absolute weights between the input and the hidden node reveals the pixels that were focused on by the hidden node to engineer the latent feature. The below image shows that this one hidden node captured some curves/lines, such as a group of pixels at the top and another group of pixels at the bottom of the image.



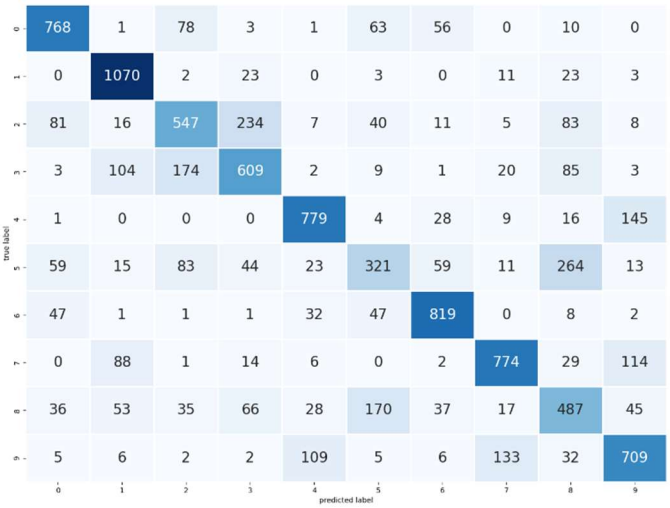
4.2 Experiment 2: NN with 2-node hidden layer

Increasing the hidden layer to 2 nodes provides an additional dimension to represent the original images. As a result, the prediction performance of the NN improved significantly with more stable training outcomes, as indicated by the improved overall accuracy of the prediction and the successfully recalled incidences of each class (located along the diagonal of the confusion matrix).

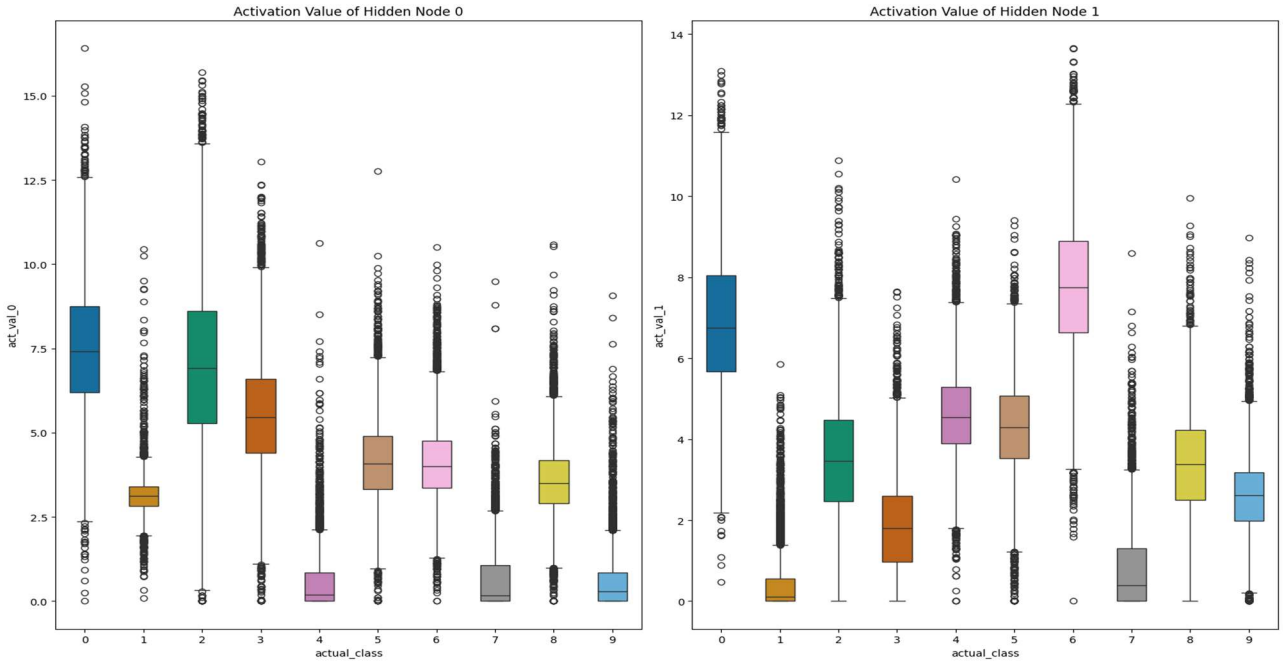
Classification Report

	precision	recall	f1-score	support
0	0.77	0.78	0.78	980
1	0.79	0.94	0.86	1135
2	0.59	0.53	0.56	1032
3	0.61	0.60	0.61	1010
4	0.79	0.79	0.79	982
5	0.48	0.36	0.41	892
6	0.80	0.85	0.83	958
7	0.79	0.75	0.77	1028
8	0.47	0.50	0.48	974
9	0.68	0.70	0.69	1009
accuracy			0.69	10000
macro avg	0.68	0.68	0.68	10000
weighted avg	0.68	0.69	0.68	10000

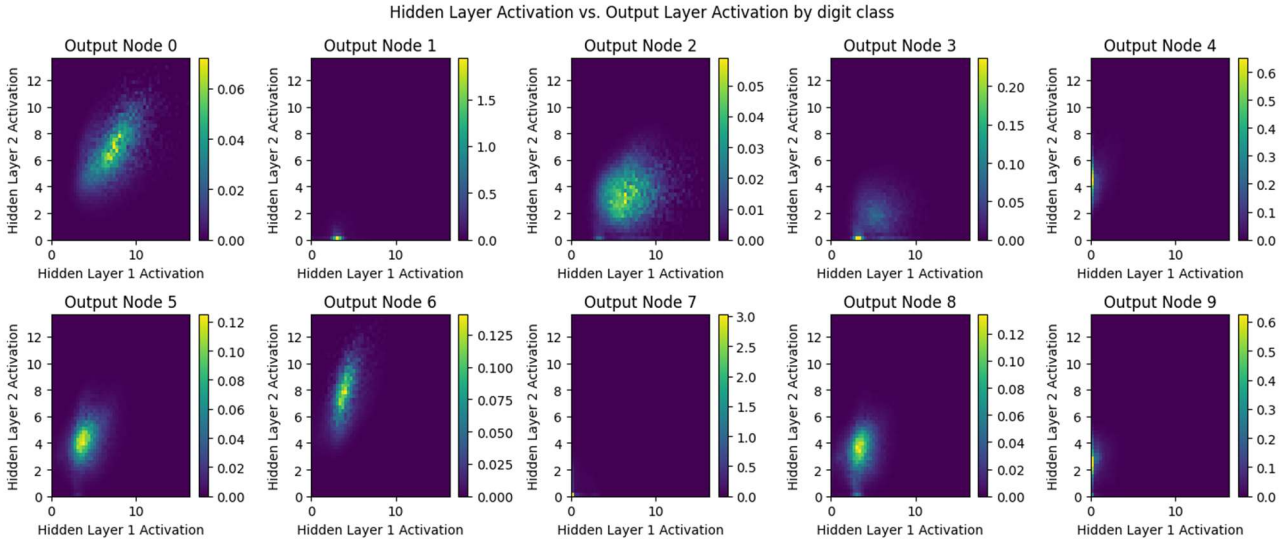
Accuracy Score: 0.6883
Root Mean Square Error: 2.0988568317062506



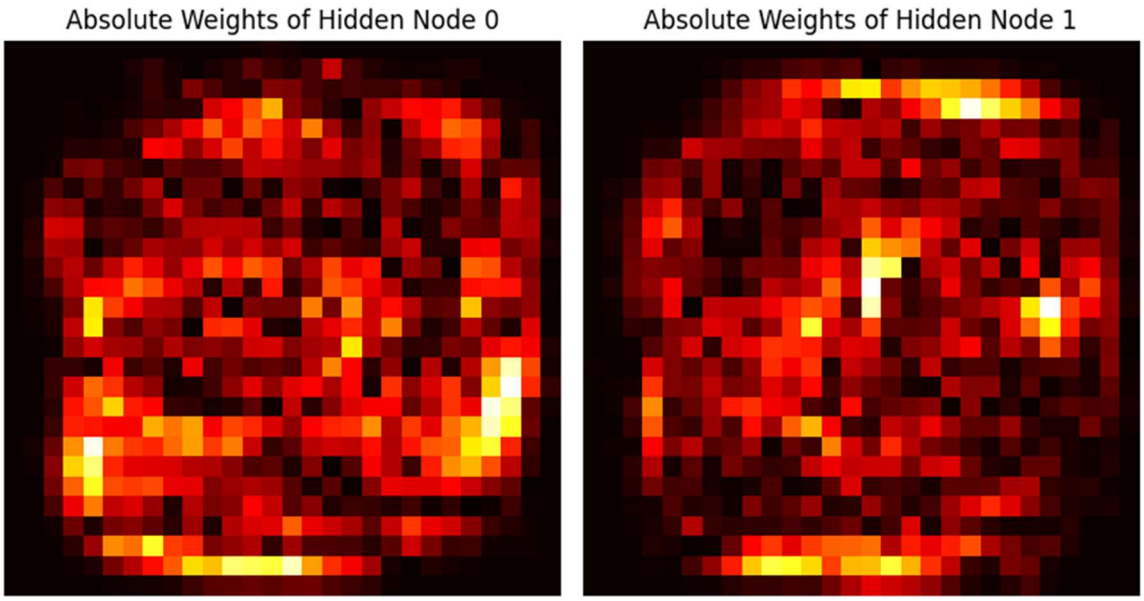
There were still some misclassifications among digits 4, 7 and 9, digits 2 and 3, and digits 5 and 8. The activation value distributions of different digits generated by the two hidden neurons below show less overlap in both latent dimensions. Digits 4, 7, and 9 have strong overlap at hidden node 1. Digits 2-3 pair and 5-8 pair both have some overlap in both hidden nodes. These observations again coincide with the misclassification between these groups.



The relationships between the NN outputs and the activation values of the hidden nodes can be plotted with `hist2d`: the binned activation values of the hidden nodes are displayed along the x and y axis; the sum of the output at each output node are displayed as heatmap. Output nodes 2 and 3, as well as 4-9 and 5-8 pairs, show certain level of overlap, indicating similar responses to the similar activation values from the hidden nodes. For the same reasons described above, there were still insufficient hidden dimensions to distinguish these classes.



Similarly, the pixels that were focused on by the two hidden nodes can be illustrated by plotting the heatmap of the absolute weights connecting the input and each hidden node as below. The two nodes attended to very distinct regions of the original image.



4.3 Experiment 3: NN with 4,8,16,32,128-node hidden layer, and a two-hidden layer model

As the number of hidden nodes was increased from 2 to 4, then 8, 16, and 32 in the NN model, its prediction performance improved steadily as summarized below.

4-node

Classification Report

	precision	recall	f1-score	support
0	0.92	0.96	0.94	980
1	0.91	0.97	0.94	1135
2	0.90	0.87	0.89	1032
3	0.82	0.84	0.83	1010
4	0.84	0.88	0.86	982
5	0.81	0.76	0.79	892
6	0.92	0.92	0.92	958
7	0.91	0.89	0.90	1028
8	0.85	0.78	0.82	974
9	0.82	0.82	0.82	1009
accuracy			0.87	10000
macro avg	0.87	0.87	0.87	10000
weighted avg	0.87	0.87	0.87	10000

Accuracy Score: 0.8723
Root Mean Square Error: 1.4263590010933433

8-node

Classification Report

	precision	recall	f1-score	support
0	0.95	0.98	0.96	980
1	0.96	0.98	0.97	1135
2	0.93	0.93	0.93	1032
3	0.92	0.93	0.93	1010
4	0.93	0.95	0.94	982
5	0.95	0.91	0.93	892
6	0.93	0.95	0.94	958
7	0.94	0.93	0.94	1028
8	0.92	0.90	0.91	974
9	0.93	0.92	0.92	1009
accuracy			0.94	10000
macro avg	0.94	0.94	0.94	10000
weighted avg	0.94	0.94	0.94	10000

Accuracy Score: 0.9368
Root Mean Square Error: 1.0705606008068855

16-node

32-node

Classification Report

	precision	recall	f1-score	support
0	0.96	0.98	0.97	980
1	0.98	0.99	0.98	1135
2	0.94	0.94	0.94	1032
3	0.94	0.94	0.94	1010
4	0.95	0.96	0.96	982
5	0.95	0.90	0.93	892
6	0.95	0.96	0.96	958
7	0.95	0.95	0.95	1028
8	0.93	0.93	0.93	974
9	0.95	0.93	0.94	1009
accuracy			0.95	10000
macro avg	0.95	0.95	0.95	10000
weighted avg	0.95	0.95	0.95	10000

Accuracy Score: 0.9515
Root Mean Square Error: 0.9392550239418472

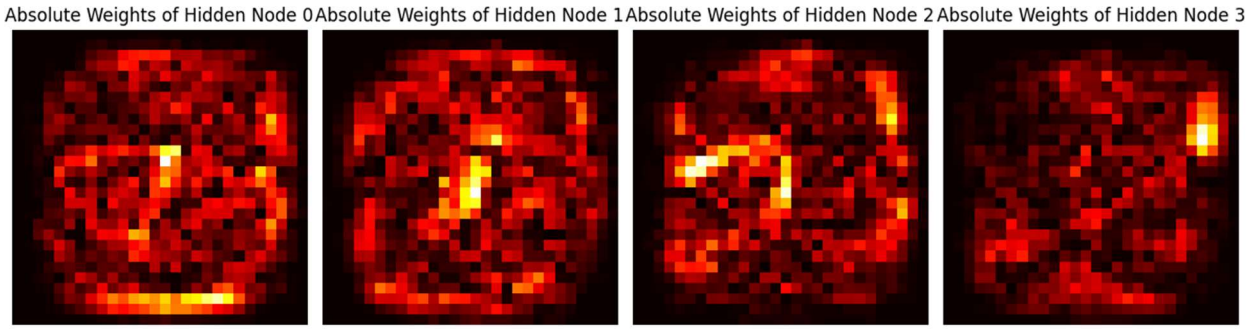
Classification Report

	precision	recall	f1-score	support
0	0.98	0.99	0.98	980
1	0.98	1.00	0.99	1135
2	0.97	0.98	0.97	1032
3	0.97	0.98	0.97	1010
4	0.97	0.98	0.98	982
5	0.99	0.96	0.98	892
6	0.98	0.98	0.98	958
7	0.98	0.96	0.97	1028
8	0.97	0.97	0.97	974
9	0.98	0.96	0.97	1009
accuracy			0.98	10000
macro avg	0.98	0.98	0.98	10000
weighted avg	0.98	0.98	0.98	10000

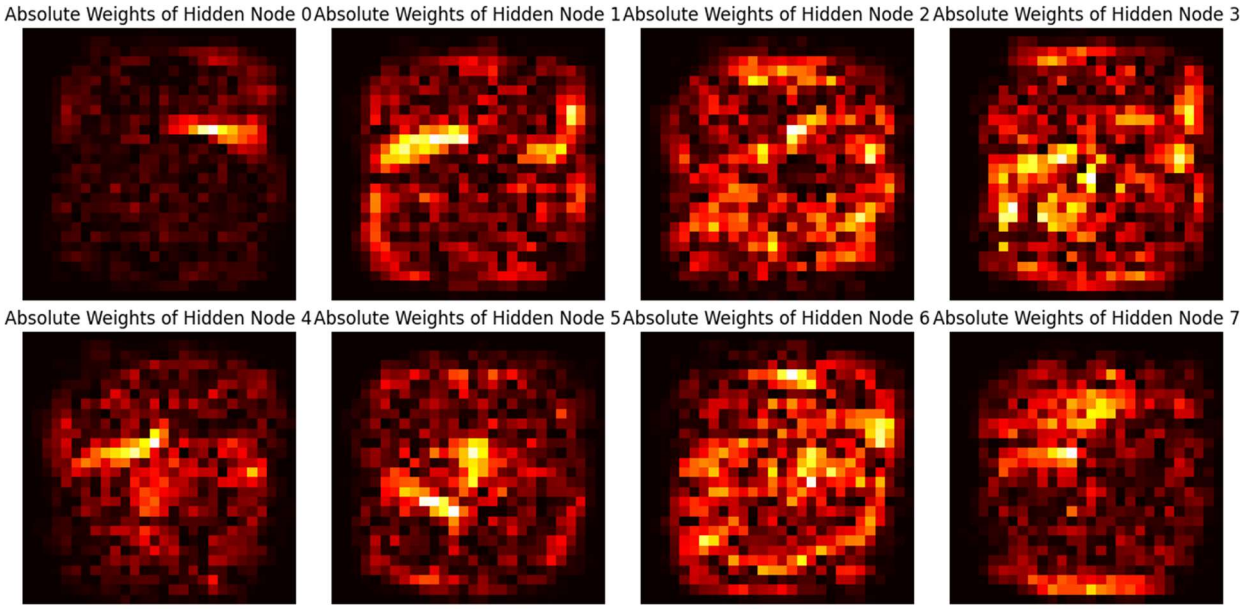
Accuracy Score: 0.9761
Root Mean Square Error: 0.705691150575094

The pixels that were used to construct each latent feature also become more specialized and more focused on smaller regions.

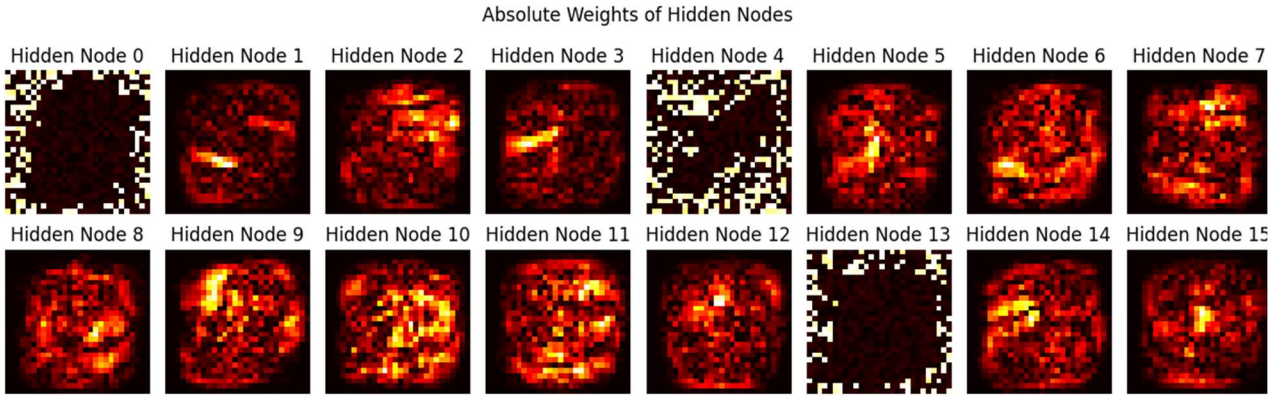
4-node latent feature loadings



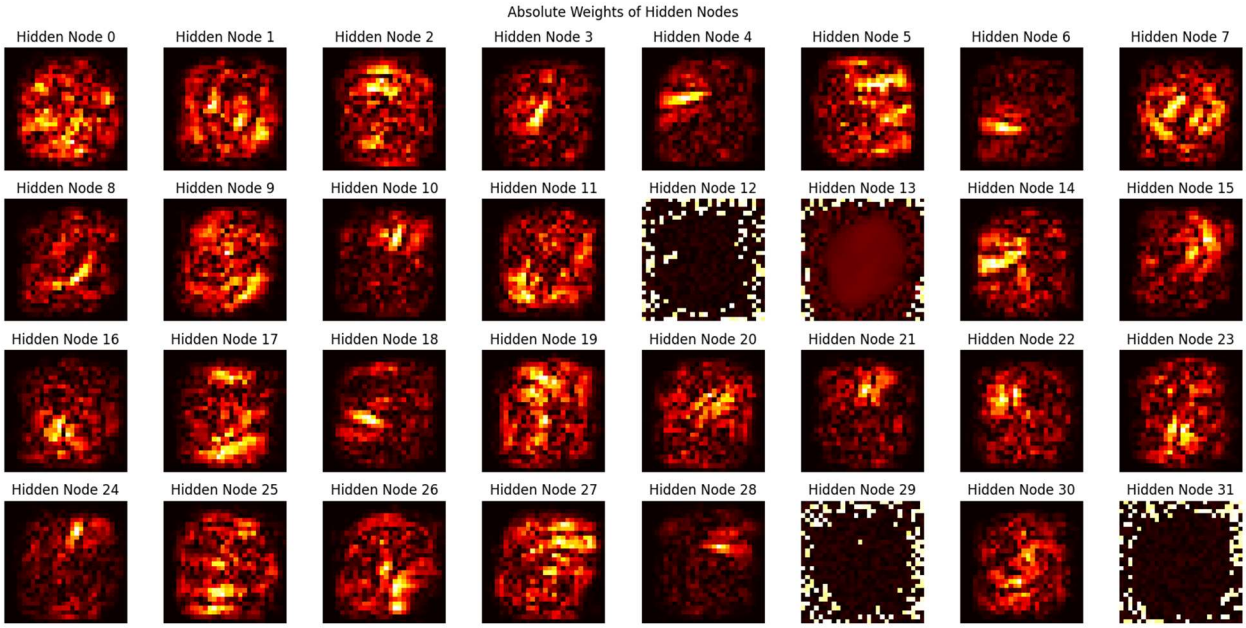
8-node latent feature loadings



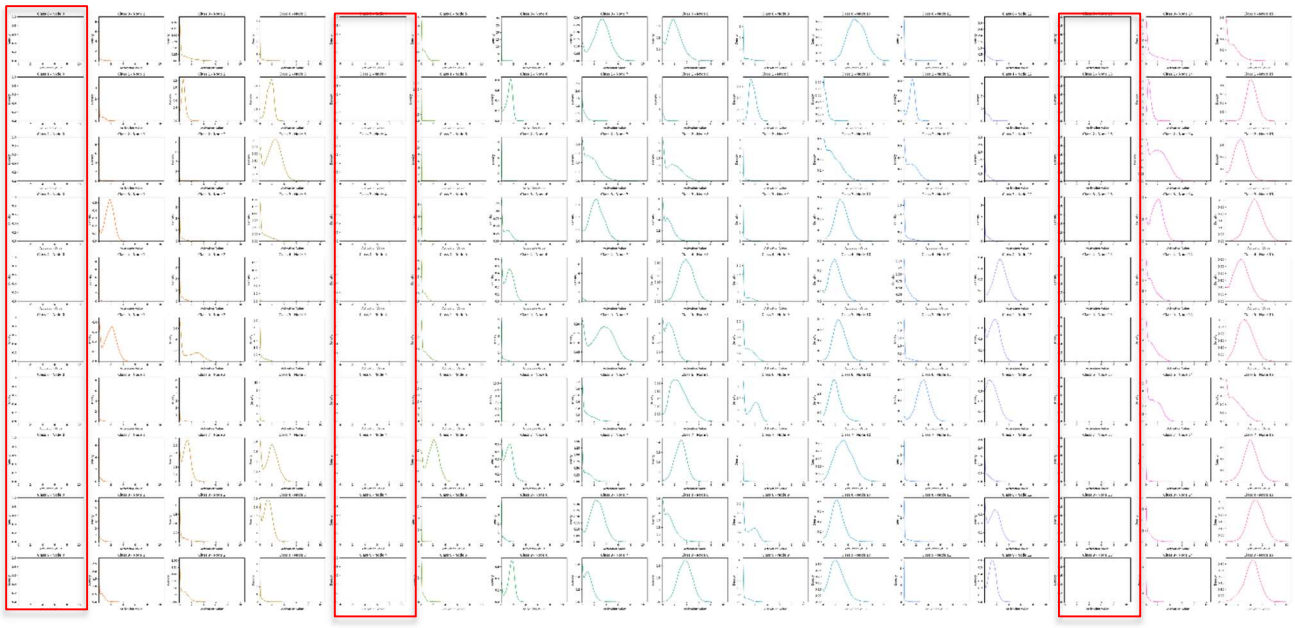
16-node latent feature loadings



32-node latent feature loadings



Interestingly both the 16-node and 32-node NN models resulted in several noise-learning nodes. These nodes (node 0, 4, and 13) didn't output any signal as shown in the KDE plots of the activation values of the hidden nodes of 16-node model (the same type of plot for the 32-node model was too big to display). This is probably due to the operation of ReLU which turned all negative values into 0. Backpropagation could not update weights of neurons with output 0, so in some sense, these neurons died.

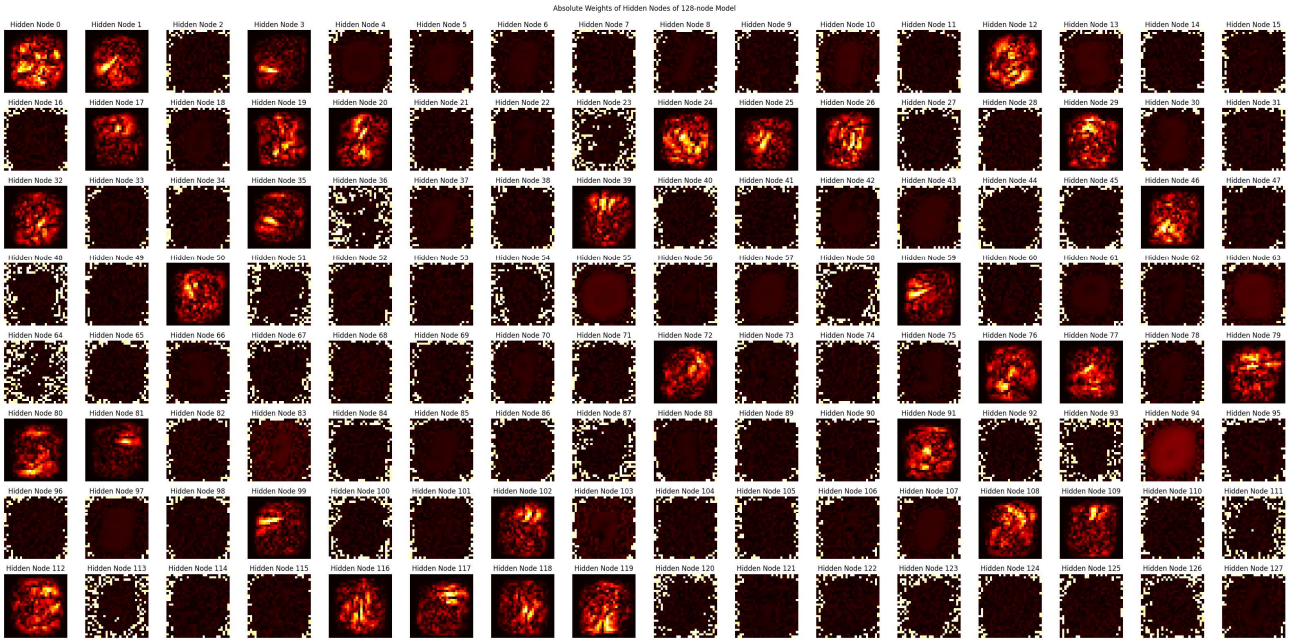


The activation values of the other hidden nodes show distinct signature of distributions across all class labels (the same type of KDE plot for 4-node and 8-node models can be found in the python notebook).

The number of noise-learning nodes did not seem to be proportional to the number of total hidden nodes, rather it is intrinsic to the structure of the data. A 128-node model trained with the settings in this paper only had 33 nodes learning nonzero pixels, while 75% of the hidden nodes attended to

noise. This means only 5 more latent features were learned by the 128-node model compared to the 32-node model. Nevertheless, the 128-node model only slightly improved the prediction performance with the test set (0.9763 overall accuracy), suggesting that a latent space with about 30 dimensions is sufficient to represent most topological information of the MNIST dataset.

Latent feature loadings of the 128-node model



I next trained a 2-hidden-layer model with 32 nodes in the first hidden layer and 16 nodes in the second. This architecture was essentially testing whether combining the latent features of the first hidden layer into 16 nodes will enhance the model's performance. The resulting model had similar performance to the 32-node and 128-node models but no further improvement.

128-node

Classification Report

	precision	recall	f1-score	support
0	0.97	0.98	0.98	980
1	0.99	0.99	0.99	1135
2	0.98	0.98	0.98	1032
3	0.97	0.98	0.97	1010
4	0.98	0.97	0.97	982
5	0.98	0.97	0.97	892
6	0.98	0.98	0.98	958
7	0.98	0.97	0.97	1028
8	0.97	0.98	0.97	974
9	0.97	0.97	0.97	1009
accuracy			0.98	10000
macro avg	0.98	0.98	0.98	10000
weighted avg	0.98	0.98	0.98	10000

Accuracy Score: 0.9763
Root Mean Square Error: 0.7103520254071217

32-node and 16-node

Classification Report

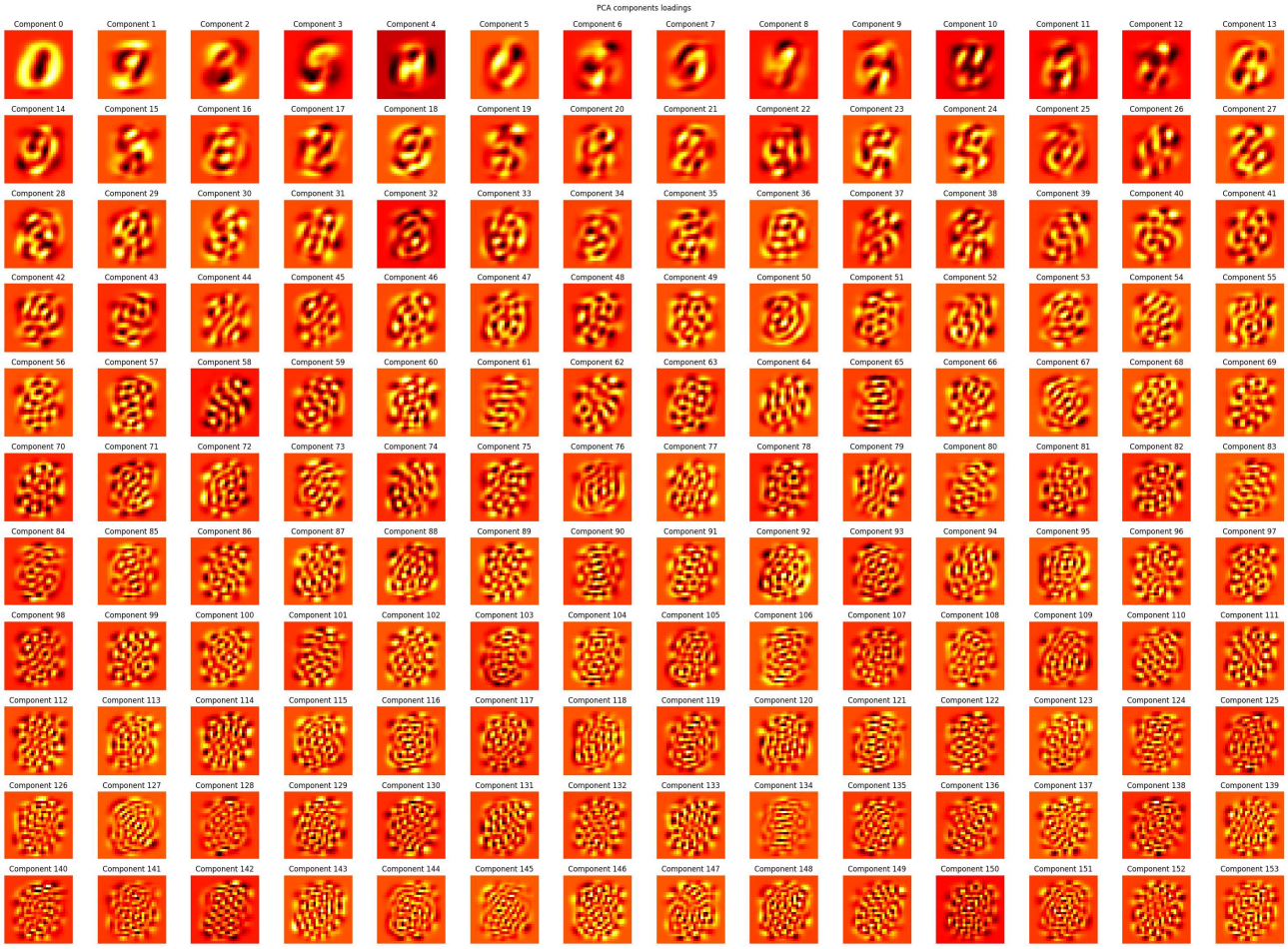
	precision	recall	f1-score	support
0	0.97	0.99	0.98	980
1	0.99	0.99	0.99	1135
2	0.97	0.97	0.97	1032
3	0.97	0.97	0.97	1010
4	0.98	0.98	0.98	982
5	0.98	0.96	0.97	892
6	0.97	0.98	0.97	958
7	0.97	0.96	0.97	1028
8	0.96	0.97	0.97	974
9	0.97	0.96	0.97	1009
accuracy			0.97	10000
macro avg	0.97	0.97	0.97	10000
weighted avg	0.97	0.97	0.97	10000

Accuracy Score: 0.9728
Root Mean Square Error: 0.7237402849088892

The first 32-node hidden layer of this model ended up with 7 noise-attending nodes, while the second 16-node layer had 2 noise-attending nodes (data in python notebook). Again, there did not seem to be any rules for determining how many nodes end up picking up noise (and die). It could be a behavior due to random weight initiation.

4.4 Experiment 4: Train NN with PCA-preprocessed data

FCNN in some sense acts as an encoder that projects the original data into a latent space with reduced dimensions. If the activation function is linear, FCNN is essentially the same as principal component analysis (PCA)-based projections. Next, I switched gears and tested whether linear dimension reduction by PCA affects the learning outcomes. One-hundred-fifty-four principal components from a PCA were used to project the data which preserves 95% of the data variance. Their loadings are plotted as images shown below. The first 10 principal components captured major curves and lines. As the PC ranking goes higher, more intricate feature space was projected.



I then used the compressed data to train the best-performing model architecture from Experiment 4.3, the 32-node model. The performance only reduced slightly.

Classification Report

	precision	recall	f1-score	support
0	0.98	0.99	0.98	980
1	0.98	0.99	0.99	1135
2	0.97	0.98	0.97	1032
3	0.97	0.97	0.97	1010
4	0.98	0.98	0.98	982
5	0.97	0.96	0.97	892
6	0.97	0.98	0.98	958
7	0.98	0.96	0.97	1028
8	0.97	0.97	0.97	974
9	0.97	0.97	0.97	1009
accuracy			0.97	10000
macro avg	0.97	0.97	0.97	10000
weighted avg	0.97	0.97	0.97	10000

Accuracy Score: 0.9744
Root Mean Square Error: 0.7081666470542085

Although linear dimension reduction to 154 dimensions did not lose much information that can be learned by the subsequent 32-hidden-node NN, it is not a necessary step for NN training, unlike in traditional machine learning. NNs can perform dimension reduction directly on the raw data and maybe even do a better job without PCA.

4.5 Experiment 5: Train NN with important features identified by random forest

Another common technique used for data preprocessing is feature selection. Many traditional machine learning algorithms benefit from this procedure. Here, I performed a model-based feature selection using the default settings of `sklearn.ensemble.RandomForestClassifier`. I used either the top 70 or 300 important features to train the downstream 32-hidden-node NN model. The prediction performances for the two are shown below:

Top 70 important features

Classification Report

	precision	recall	f1-score	support
0	0.94	0.96	0.95	980
1	0.96	0.99	0.97	1135
2	0.92	0.87	0.90	1032
3	0.92	0.91	0.91	1010
4	0.90	0.93	0.92	982
5	0.87	0.88	0.88	892
6	0.92	0.91	0.92	958
7	0.91	0.92	0.92	1028
8	0.91	0.91	0.91	974
9	0.91	0.89	0.90	1009
accuracy			0.92	10000
macro avg	0.92	0.92	0.92	10000
weighted avg	0.92	0.92	0.92	10000

Accuracy Score: 0.9185
Root Mean Square Error: 1.147780466814103

Top 300 important features

Classification Report

	precision	recall	f1-score	support
0	0.97	0.98	0.98	980
1	0.99	0.99	0.99	1135
2	0.96	0.96	0.96	1032
3	0.95	0.97	0.96	1010
4	0.96	0.97	0.97	982
5	0.97	0.95	0.96	892
6	0.97	0.98	0.97	958
7	0.97	0.95	0.96	1028
8	0.96	0.97	0.97	974
9	0.96	0.96	0.96	1009
accuracy			0.97	10000
macro avg	0.97	0.97	0.97	10000
weighted avg	0.97	0.97	0.97	10000

Accuracy Score: 0.9688
Root Mean Square Error: 0.7334166619323562

Prior feature selection reduced model performance due to loss of information. This shows that NNs carry out feature selection automatically and can explore different corners of the feature space better than traditional ML algorithms.

5.0 Conclusions

I explored the inner workings of fully connected neural networks (FCNNs) trained on the MNIST dataset, with a focus on how the number of hidden nodes and preprocessing techniques influence model performance and interpretability.

The experiments demonstrated that increasing the number of hidden nodes from 1 to 32 led to significant improvements in classification accuracy, with the 32-node model achieving near-optimal performance. Further increases of the number of hidden nodes (e.g., 128) provided only marginal gains, indicating that a latent space of approximately 30 dimensions is sufficient to capture the topological structure of the MNIST data. Notably, many hidden nodes in larger models attended to noise, suggesting that overparameterization does not necessarily yield better feature learning in FCNNs for this task.

Visualization of hidden layer activations and weight heatmaps revealed that FCNNs automatically learn to focus on discriminative regions of the input images, with more nodes enabling finer and more specialized feature detection. The failure of models with few hidden nodes (e.g., 1 or 2) to distinguish between certain digit classes (e.g., 3/5, 4/7/9) was directly linked to overlapping activation distributions in the latent space, which limited the network's ability to propagate distinct error signals during backpropagation.

Additionally, experiments with PCA-based dimension reduction and random forest-based feature selection showed that preprocessing techniques commonly used in traditional machine learning offered no advantage for FCNNs and could reduce performance. This underscores the ability of neural networks to perform automatic and adaptive feature selection and engineering directly from raw data.

While FCNNs are capable of learning effective representations for digit classification, their performance is constrained by the simplicity of the fully connected architecture. For image data, convolutional neural networks (CNNs), which leverage spatial hierarchies and local receptive fields, are likely better suited to further improve accuracy.

References

Choromanska, Anna, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. 2015. "The loss surfaces of multilayer networks." *arXiv* 1412.0233.

Cybenko, G. 1989. "Approximation by superpositions of a sigmoidal function." *Mathematics of Control, Signals and Systems* 2:303–314.

Duchi, John, Elad Hazan, and Yoram Singer. 2011. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization." *Journal of Machine Learning Research* 12:2121–2159.

Erhan, Dumitru, Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2009. *Visualizing higher-layer features of a deep network.* Technical Report 1341, Montreal, Canada: D´epartement d'Informatique et Recherche Op´erationnelle.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification." *arXiv* 1502.01852.

Hornik, Kurt, Maxwell Stinchcombe, and Halbert White. 1989. "Multilayer feedforward networks are universal approximators." *Neural Networks* Volume 2, Issue 5, Pages 359-366.

Kingma, Diederik P., and Jimmy Ba. 2014. "Adam: A Method for Stochastic Optimization." *arXiv* 1412.6980.

Nesterov, Yurii. 1983. "A Method for Unconstrained Convex Minimization Problem with the Rate of Convergence O(1/k2)." *Doklady AN USSR* 269:543–547.

Polyak, Boris T. 1964. "Some Methods of Speeding Up the Convergence of Iteration Methods." *USSR Computational Mathematics and Mathematical Physics* 5:1-17.