Neural Networks Classification on Fashion-MNIST

Group 47 - Preetesh Rambarun, Ben Cheung, Andrew Chao

Abstract—The neural network is one of the most broadly used machine learning algorithms for its flexibility and expressiveness. Some of the most common types of neural networks include perceptron, multi-layer perceptron, convolutional, and recurrent. One particular use case in which neural networks find extensive applications is image classification. In this report, we implement a Multi-Layer Perceptron (MLP) model to classify images from the Fashion-MNIST dataset. We investigate the effect of different MLP designs on accuracy and efficiency, including depth, width and choices of activation functions. We also compare our MLP implementation with a Convolutional Neural Network model (CNN).

1 Introduction

Neural networks are collections of machine learning algorithms that in a sense mimic the interactions of biological neurons in the human nervous system. The adaptiveness of a neural network allows the algorithm to make improvements by learning from mistakes, and the complexity permits a neural network to perform generalized inference tasks. In the modern tech-driven world, neural networks contribute to complex applications like facial recognition, text translation, credit card fraud detection, and medical diagnosis. [1]

In this project, we implement one of the fundamental neural network models, Multi-Layer Perceptron (MLP), and test the implementation on the image classification tasks on the Fashion-MNIST dataset [2]. The Fashion-MNIST dataset provides a collection of fashion-based images, and the dataset is developed with the objective of providing a benchmark dataset for neural network models alternative to the well-known MNIST dataset [2].

To evaluate our model performance, we study the effect of various MLP architectures, for instance, the number of units in hidden layers (i.e. width), the number of hidden layers (i.e. depth), and the activation functions between linear layers. To compare the optimal performance of different architectures, we train and test our model based on the optimal hyperparameter combination provided by grid search, and implement the early stopping feature to prevent overfitting. We also compare and contrast our MLP implementation with a Convolutional Neural Network (CNN) using the same processed image data.

By varying the different parameters of our MLP implementation, we achieve an optimal accuracy of over 80% for most of the MLP architectures in trials. We notice that the width is the determining factor of run-time and memory usage: an increase in width contributes to a drastic increase in training time and memory used for training, while an increase in depth or a change of activation functions has little effect on these two aspects. We also remark that the CNN achieves a similar accuracy but a worse efficiency as compared to most of the test runs on our MLPs.

1.1 Related Works

The Fashion-MNIST dataset has been widely used in works related to computer vision. This emerges as an increasingly important application as computers are proven to outperform humans in interactions with large image datasets through machine learning processes. The Fashion-MNIST dataset serves as a benchmark dataset along with the more well-known MNIST dataset. As an example, the CNN model developed in [3] was tested on both the Fashion-MNIST dataset and the MNIST dataset, and they observed a high accuracy on Fashion-MNIST (92%) and an almost perfect accuracy of 99.7% on the original MNIST dataset.

The accuracy results in the above example echo the motivations for creating the Fashion-MNIST dataset from the creators [2]: the MNIST dataset is considered to be simple in present-day's view with the numerous deep learning techniques developed since the dataset's introduction more than two decades ago.

2 Models

This report's primary model in focus is the MLP model. It is based on the linear learning algorithm of perceptron. The combination of non-linear activation functions and a multi-layer structure of MLP resolves the divergence issue for non-linear separable data for the ordinary perceptron algorithm.

2.1 MLP Model

We implement the MLP model based on the object-oriented programming (OOP) principle. In our implementation, with reference to the course's Colab codes [4], an MLP object is defined by a list of alternating linear layers and activation function layers and an optimizer. We highlight some features of our MLP class implementation.

2.1.1 Linear Layer

We implement L2-regularization to the linear layers as a base feature, and initiate $\lambda = 0$ by default for the ordinary unregularized case. In addition, we set a parameter `noise_scale` to control the scales of the weight and bias initialization. This mitigates the vanishing gradient problem, which we elaborate in the next sections.

2.1.2 Activation Function Layer

The activation functions we implement include `ReLU`, `tanh`, `LeakyReLU`, `sigmoid` (σ), `softplus` and Exponential Linear Unit (`ELU`).

Moreover, in order to produce multi-class categorical predictions, we use the `softmax` function as the output layer. It is noticed that `softmax` is prone to vanishing gradient problem: the c-th coordinate of $\texttt{softmax}(a)$ is calculated by

$$\hat{y}_c = \frac{\exp(a_c)}{\sum_c \exp(a_c)},$$

which can incur large numbers in the $\exp$ calculations despite the fact that $\hat{y}_c \in [0, 1]$. This numerical instability issue is resolved by using the equivalent formula

$$\hat{y}_c = \frac{\exp(a_c - a_{\max})}{\sum_c \exp(a_c - a_{\max})},$$

where $a_{\max}$ is the maximum entry of a. This alternative formula guarantees that each $\exp$ takes a non-positive argument and hence bounds each $\exp(\cdot)$ term between 0 and 1.

2.1.3 Optimizer

To address computation cost concerns, we implement mini-batch gradient descent as the default setting. This prevents extensive memory usage with the trade-off of an increase in run-time.

We implement a simple early-stopping criterion for our MLP model. Given a validation set, the early-stopping criterion is provoked when the validation loss increases or the validation accuracy decreases. The training is stopped in k rounds after the criterion is triggered, where k is a parameter of the fitting function commonly known as "patience".

2.2 MLP Gradient Verification

To ensure the correctness of the MLP implementation, we implement a gradient check function. This verifies that the numeric gradient ∇_n calculated by small perturbations on the weights and biases approximates the derived gradient ∇_d closely. We evaluate the closeness of approximation in two senses:

- *Absolute*: we check that $\|\nabla_n - \nabla_d\|$ is small.
- *Relative*: we notice that the measure proposed in Lecture 4.3 behave poorly when the values are close to zero. We adopt the approach in [8] to check that the following quantity is small:

$$\text{Perturbation factor} = \frac{\|\nabla_n - \nabla_d\|}{\|\nabla_n\| - \|\nabla_d\|}.$$

The gradient check results are in Appendix A.

2.3 CNN Model

The `Keras` library, a deep learning API built on top of TensorFlow [5], is used to implement our CNN model, with reference to the course's Colab codes. [6] `Keras` is highly optimized and allows users to configure hyperparameters, activation functions, and operations (e.g., dropout factor and max pooling) easily. For this work, we create a 2-convolutional and 2-fully connected layers with varying operations and parameters (e.g., filer size and stride size) to compare with our MLP model.

2.4 Hyperparameter Tuning

Hyperparameter tuning is a crucial step to optimize the performance of MLPs. We make use of the `GridSearchCV` function in `sk-learn` library [7] for the tuning. The primary hyperparameter for tuning is the learning rate (lr), L2 regularization penalty (λ), and for some parameters in activation functions, such as `LeakyReLU` and `Softplus`.

For run-time concerns, we conduct the tuning with the following validation strategy: for each combination of hyperparameters in test, we run the fitting procedure once on a fixed choice of the training set and validation set. For cases where more than one hyperparameter requires tuning, the number of hyperparameter combinations may become too high for an effective exhaustive search. In such cases, we use `RandomizedSearchCV` [9] from `sk-learn` to check on a random subset of hyperparameter configurations.

We do not tune the number of epochs in the procedure since it requires tracking of the validation loss and accuracy per epoch. The built-in performance evaluation of `GridSearchCV` does not provide such information. We get around the issue by passing a

tuple of lists to the fitting function of our MLP class implementation to record the histories over all epochs for each test. This permits us to directly inspect the loss and accuracy curves for signs of overfitting. In combination with the early stopping criterion implemented, the hyperparameter tuning is performed effectively in choosing the parameters yielding the best performance while avoiding overfitting.

3 Fashion-MNIST Dataset

3.1 Data Inspection

It is inspected that Fashion-MNIST consists of a training set of 60,000 examples and a test set of 10,000 examples; the dataset is a collection of fashion-related images. Each data sample is a 28x28 square pixel grayscale image, associated with a label from 10 different classes. Both the training and testing datasets are equidistributed over the 10 classes.

3.2 Data Processing

We vectorize each image sample to an array of 784 entries. For each of the training and testing datasets, we prepare a normalized version and retain an unnormalized version. For each version of the training set, we keep 20% of the testing data (i.e. 12,000 samples) as the validation set for hyperparameter tuning and validation during fitting.

4 Results

In this section, we present the results based on the investigation of different numbers of hidden layers, hidden units, and different choices of activation functions of our MLP and CNN implementation.

4.1 Basic MLP Models with ReLU

We inspect the performances of MLPs with 0, 1 and 2 hidden layers, respectively. We use 128 units per hidden layer and ReLU activation and 10 epochs for this experiment. Table 1 summarizes the test results.

We observe that the optimal test accuracy for 0-hidden-layer MLP is 83.43%, which is about 3% lower than those for 1-hidden-layer MLP (86.09%) and 2-hidden-layer MLP (85.46%). As anticipated, non-linearity granted by MLPs with hidden layers increases the test accuracy. A 0-hidden-layer MLP with softmax output is simply equivalent to a multiclass classification model. With non-linear activation functions inserted between the linear layers, the model is able to handle more complex relations. On the other hand, the 3% improvement is somewhat lower than expected, but it can be reasoned that the 10 categories are mostly distinguished and the linear separability assumption is not greatly violated.

The more discernible element of the model performance is the training time. As shown in Table 1, the MLP with no hidden layers has a significantly shorter training time. This is expected because the presence of hidden layers introduces substantially more weight and bias parameters to be updated. This is evident from the number of parameters to be checked by gradient check: only 7,850 parameters are checked without hidden layer, while there are 101,770 and 118,282 weight and bias parameters for 1 and 2 hidden layer cases respectively.

TABLE 1: **Training performance with different number of hidden layers (learning rate = lr)**

Hidden layers	Hidden units	lr	Run-time	Accuracy
0	N/A	0.15	25s	83.43%
1	128	0.25	3min 40s	86.09%
2	128	0.05	4min 22s	85.46%

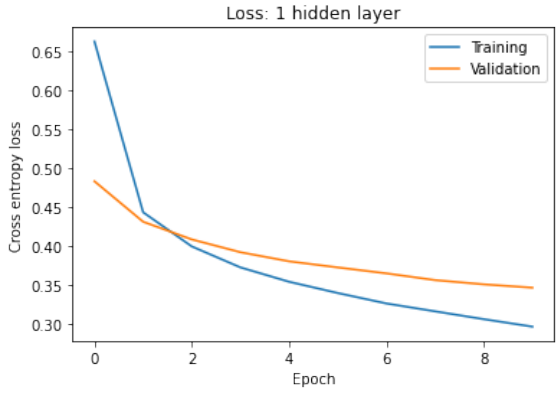


Fig. 1: **One Hidden Layer MLP loss curves**

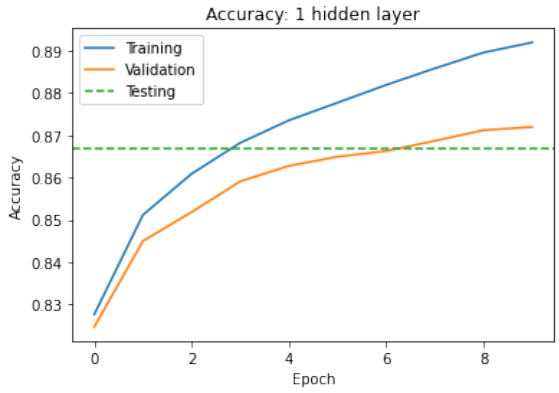


Fig. 2: **One Hidden Layer MLP accuracy curves**

As an example, Figures 1 and 2 show the loss and accuracy curves for 1-hidden-layer MLP. Appendix B shows the complete results for this task.

4.2 Effects of activation functions

We compare the performance of 2 layer MLPs with different activation functions, 128 hidden units each, and 10 epochs. Besides `ReLU` and $\tanh$ activation functions, we implement four additional activation functions, namely `LeakyReLU`, `Sigmoid`, `Softplus` and `ELU`, and compare their performance. The result is summarized in Table 2. For clarity, we fine tune the two activation functions with extra parameters here:

- LeakyReLU: $f_a(x) = \max(x, 0) + a \times \min(x, 0)$.
- ELU: $f_a(x) = \max(x, 0) + \min(a(e^x - 1), 0)$.

The detailed results can be found in Appendix C.

TABLE 2: **Training performance with different activation functions (learning rate = lr; Hyper-parameters = H. parameters)**

Activation	lr	H. parameters	Run-time	Accuracy
ReLU	0.15	N/A	4min 22s	85.46%
tanh	0.10	N/A	4min 21s	85.44%
LeakyReLU	0.20	$a = 0.10$	3min 09s	85.53%
Sigmoid	0.20	N/A	4min 22s	85.97%
Softplus	0.15	N/A	4min 07s	84.97%
ELU	0.15	$a = 0.10$	4min 16s	86.44%

It is known that the activation functions $\tanh$ and `Sigmoid` are prone to vanishing gradient problems. However, based on our results, we see no considerable differences between these two activation functions and the other ones in terms of accuracy and run-time. This observation could be attributed to the relatively shallow depth (2) of the MLPs, so the vanishing gradient problem can be mitigated by simply initializing a suitable `noise_scale` for each linear layer.

4.3 L2 Regularization

To investigate the effect of L2 regularization, we consider a 2-hidden-layer MLP with ReLU activation and 128 hidden units for both layers. For this case, we have two hyperparameters: lr and λ, to be tuned. Based on the optimal hyperparameters $(\text{lr}, \lambda) = (0.02, 0.005)$, the test accuracy with L2 regularization is 85.04%. We check the gradient and ensure the correctness of implementation (Appendix A). Compared with the test accuracy without regularization(85.46%), we can see that the L2 regularization poses little effect on the performance of the Fashion-MNIST dataset.

As an additional experiment, we investigate the effect of different λ across layers. We perform a random search for the optimal combination of (lr, λ) due to computational time concerns. By hyperparameter tuning, the optimal hyperparameters are $(\text{lr}, \lambda_1, \lambda_2, \lambda_o) = (0.02, 0.005, 0.01, 0.005)$. The test accuracy using this hyperparameter configuration is 84.45%, which is comparable to the uniform λ case. The detailed results of regularization-related experiments can be found in Appendix D.

4.4 Unnormalized Images

As another test, we train the 2-hidden-layer MLP with unnormalized images. The optimal learning rate is $\text{lr} = 15 \times 10^{-5}$. As expected, this optimal lr is smaller in magnitude compared with the lr for the normalized case. The test accuracy for the unnormalized case is 81.04%, which is lower than the normalized case (86.40%) by almost 5%. We also remark that more fluctuations in loss and accuracy curves for training, validation and testing are observed. The complete results can be found in Appendix E.

4.5 Convolutional Neural Network

We construct a 2-layer CNN with several CNN-specific features for image processing. We use max-pooling to reduce the dimensions of feature map and dropout to prevent overfitting. We run the model with the same settings: 32 filters, 128 batch size, 3 kernal size, 30 epochs, 2 strides, 2 max-poolings, 128 hidden units and 0.25 dropout rate. We test it with three different commonly used optimizers: `rmsprop`, `SGD`, and `adam`. The detailed structure of the CNN model can be found in Appendix F.

TABLE 3: **CNN performance summary**

Optimizer	Test accuracy	Run-time(s)
SGD	88.33%	83.15
adam	92.12%	82.74
rmsprop	92.26%	96.03

As for the comparison between the MLP and CNN models, we notice that for the CNN models, the test accuracy improves as compared with MLP, and the training time is substantially shorter than our model. However, we point out that the CNN accuracy using SGD (88.33%) is highly comparable to the best MLP accuracy we attained (86.68%), achieved by an MLP with 1 hidden layer of 128 units. We refer readers to Appendix F for the extensive training results of CNN.

4.6 Different MLP Architecture Features

We also investigate the performance of our model with different combinations of parameters: the number of hidden layers, the number of units in hidden layers, and the choice of activation functions. Due to space limitations, the performance summary for each experiment can be found in Appendix G. We highlight several discussion points in the section.

An important observation is that the increased depth does not improve the model performance greatly. The optimal testing accuracies for MLPs with 3 and 4 hidden layers are 86.48% and 85.52% respectively, which are comparable to the 2-hidden-layer case. The training times for deeper neural networks are also similar.

In contrast, a varying width poses a more observable impact on the training efficiency. The training accuracy for narrower (16 or 32 units per hidden layer) or (256 units per hidden layer) is comparable to the base case of 128 units per hidden layer. However, the training time is basically proportional to the width. As an example, for a width of 256, the training time for MLPs with 1 and 2 hidden layers are 7 min 44 s and 8 min 34 s respectively. These are almost double of the respective time marks (3 min 40 s and 4 min 22 s) for the width-128 cases.

As a side experiment, we test with MLPs of non-uniform widths, and no significant improvement or degradation is observed. A more interesting experiment is the choice of activation functions for MLPs with greater depth. We test the 3-hidden-layer MLPs with ReLU activation replaced by `tanh` and `Sigmoid`, two activation functions tending to induce the vanishing gradient problem. Our experiments reflect that `Sigmoid` is more prone to the vanishing gradient problem: the optimal test accuracy attained by `tanh` activation remains high at 85.58%, but the accuracy attained by `Sigmoid` activation deteriorates to 53.80%.

4.7 Different Training Set Sizes

We also investigate the effect of varying training set sizes by training on a 2-hidden-layer width-128 MLP. The test performances are tabulated below. As anticipated, the training time and accuracy increase with the training set size.

TABLE 4: **Training with different training set sizes**

Training set size	Run-time	Accuracy	Epochs
10	1 s	16.84 %	3
100	8 s	64.29 %	10
1,000	8 s	72.09 %	9
10,000	21 s	81.23 %	4

In all cases, the early stopping condition is reached due to worsening validation loss or accuracy.

5 Discussion and Conclusion

5.1 Depth and Width of MLPs

It is a widely-accepted principle that increasing MLP depth is preferable to MLP width. From our experiments, the contrast between the choices is less noticeable than one would expect. We observe a consistent test accuracy of over 80% and not beyond 87% for almost every depth or width structure tested. However, the effect of layer width on training efficiency is more prominent. The increased training time for wider hidden layers is clearly reflected in Table 6.

The deterred training efficiency can be attributed to the increase in the number of parameters; for the same reason, increased widths cause the training procedure to be more memory-demanding.

5.2 Comparison between CNN and MLP Models

As shown in Table 3, by using an `rmsprop` or `adam` optimizer, the CNN model outperforms our MLP model in terms of training accuracy and efficiency. However, the more interesting comparison is the 88.33% CNN test accuracy attained by an `SGD` optimizer, which is very close to 86.68%, the best accuracy achieved by our MLP model. Noting that the optimizer we adopt is mini-batch gradient descent, a close variant to SGD, one may expect that the optimizer plays a significant role in the model accuracy. For future works, the implementation of `rmsprop` or `adam` optimizer for the MLP model serves as a sensible first step for improvement.

On the other hand, the CNN models built from the `Keras` library can be trained efficiently. This is likely due to the highly optimized codes of `Keras` library with GPU usage.

5.3 Simplicity of Fashion-MNIST Dataset

The experiments conducted reflect the relative simplicity of the Fashion-MNIST dataset, a feature of the dataset by design. While the dataset can be used for effective training, the effect of depth, regularization or activation function becomes relatively obscured.

5.4 Future Works

As mentioned, the implementation of more advanced optimizers is one of the probable improvements for our MLP model. Other kinds of regularization like L1-regularization may also be implemented.

For a more comprehensive study on the effects of different MLP structures, one may conduct similar testing over datasets of different sample sizes, feature sizes and numbers of output categories.

6 Statement Of Contributions

Ben implemented the MLP model, hyperparameter tuning, and model analysis. Andrew implemented the CNN model and contributed to L2 regularization. Preetesh performed reviews on related works and literature, and model and results analysis.

REFERENCES

[1] Chandra, R. (2022, September 22). Neural networks: Applications in the real world. upGrad blog. Retrieved November 17, 2022, from https://www.upgrad.com/blog/neural-networks-applications-in-the-real-world/

[2] Xiao, Han, Kashif Rasul, and Roland Vollgraf. "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms." arXiv preprint arXiv:1708.07747 (2017).

[3] Ezeugwa, Gerrard. (2022). DEVELOPING A DEEP LEARNING-BASED IMAGE MULTICLASS CLASSIFIER: CASE STUDY OF FASHION MNIST DATASET. 10.13140/RG.2.2.18165.04326.

[4] Deep Multilayer Perceptron (MLP), Harley Wiltzer and Yue Li. COMP 551 notebook. Available at https://github.com/yueliyl/comp551-notebooks/blob/master/NumpyDeepMLP.ipynb?fbclid=IwAR2m_cj79-ustBQtnIhhmvMlT43mAIQ0BUGGfYCrXrYsUyaKNS_Zrv7szvk

[5] Chollet, F. & others, 2015. Keras. Available at: https://github.com/fchollet/keras.

[6] CNN , Abdelrahman Ayad and Yue Li. COMP 551 notebook. Available at https://github.com/yueliyl/comp551-notebooks/blob/master/CNN.ipynb

[7] GridSearchCV, Scikit-learn. Available at https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

[8] CSC 411 Lecture 8: Linear Classification II, Mengye Ren and Matthew MacKay. University of Toronto. Retrieved December 4, 2022, from https://www.cs.toronto.edu/~mren/teach/csc411_19s/lec/lec08.pdf.

[9] RandomizedSearchCV, Scikit-learn. Available at https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html

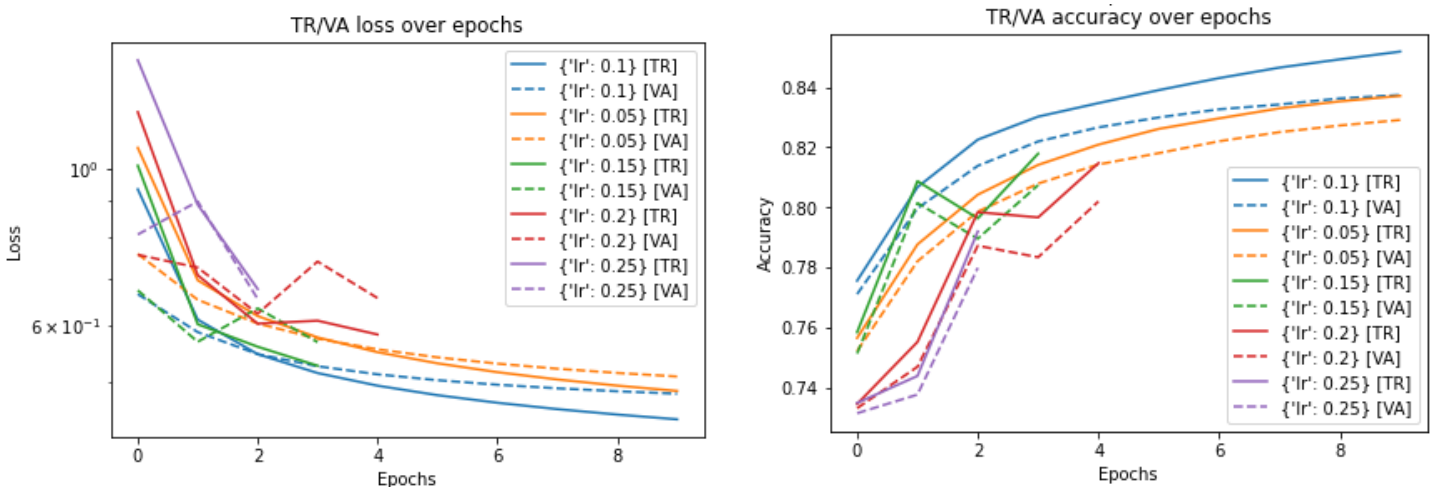
APPENDIX A
GRADIENT COMPUTATION

TABLE 5: **Gradient check results**

Hidden layers	L2-dis. between numeric and derived gradient	Perturbation factor
0	5.3447e-05	3.8649e-06
1	6.6666e-04	5.5015e-05
2	5.3447e-05	3.8649e-06
2 (with L2 penalty)	1.4917e-03	1.5597e-04

APPENDIX B
NORMALIZED IMAGES WITH 0 AND 2 HIDDEN LAYERS

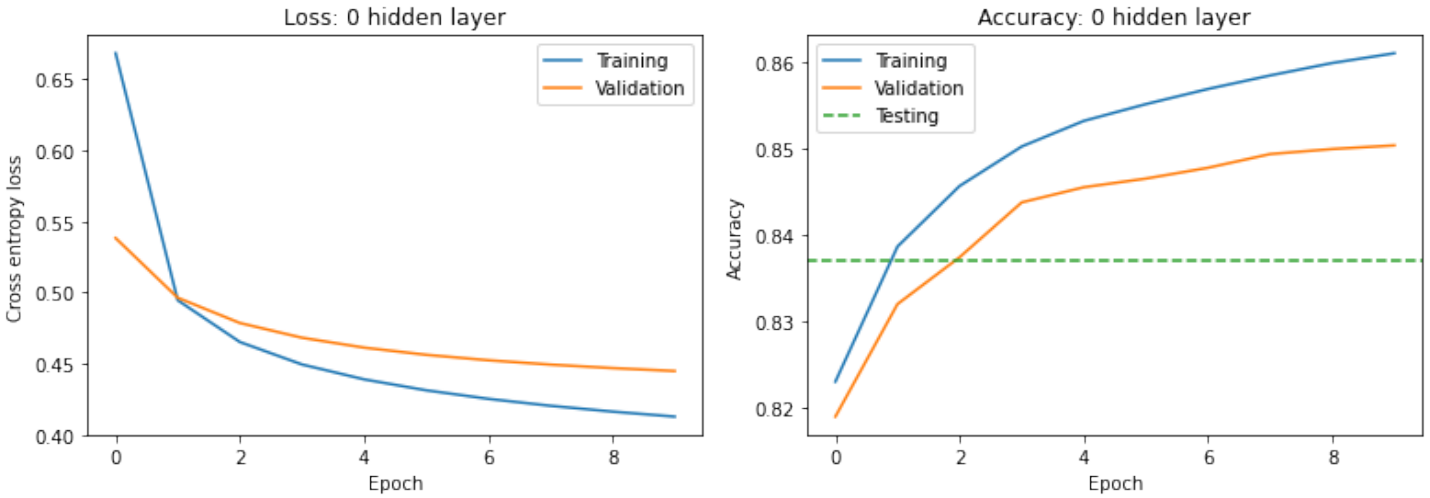
The results for 1 hidden layer have been shown in Section 4.1. Figure 4 shows the 0-hidden-layer MLP loss and accuracy with the optimal learning rate 0.1.



(a) 0 layer MLP loss over epochs

(b) 0-hidden-layer loss over accuracy

Fig. 3: **GridSearchCV results for lr for 0-hidden-layer MLP**



(a) 0 layer MLP convergence curve results

(b) 0 layer MLP accuracy results

Fig. 4: **0 layer experiment results with the optimal lr = 0.1**

Figure 6 shows the 2-hidden-layer MLP loss and accuracy with the optimal learning rate 0.08.

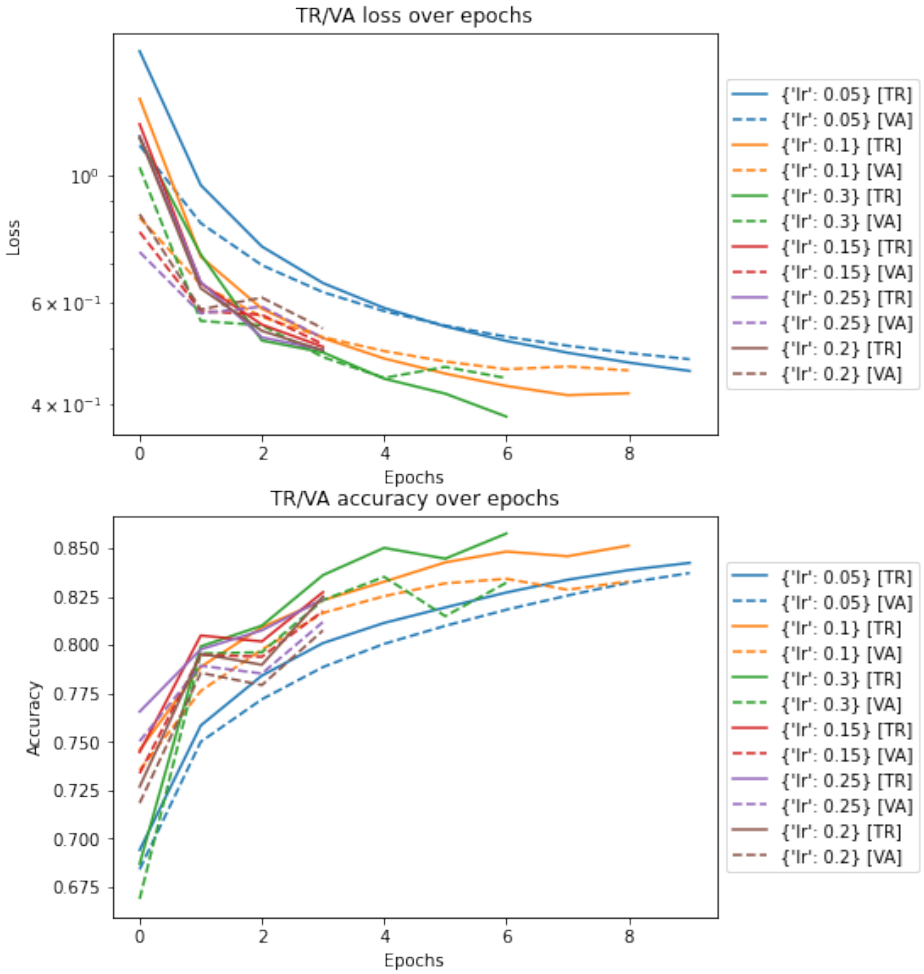
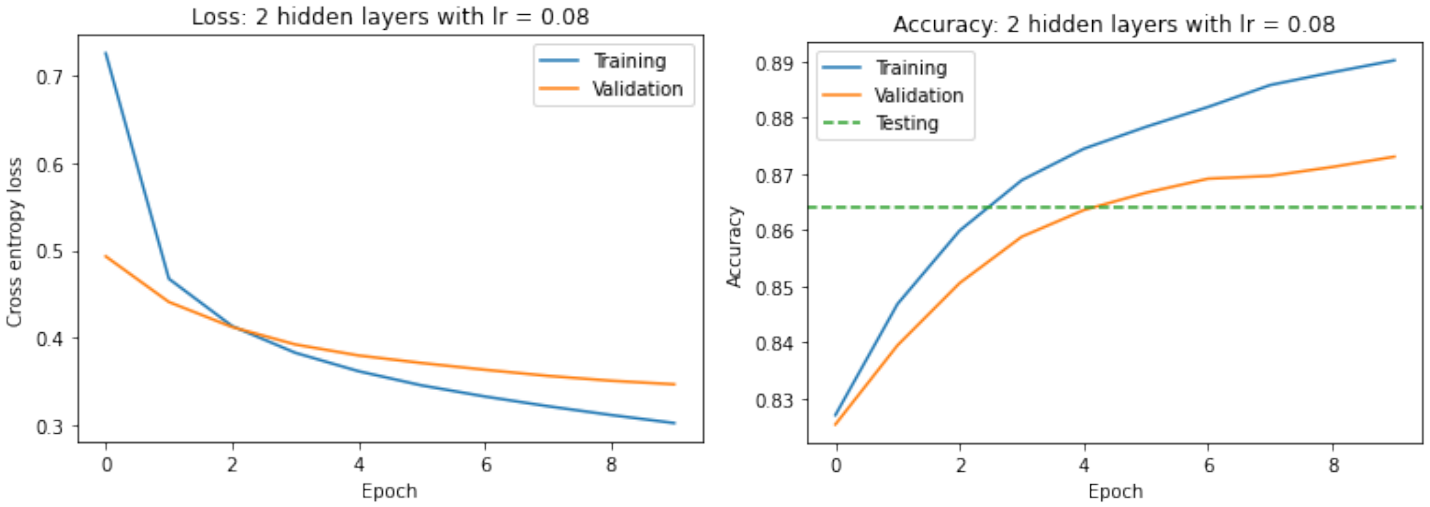


Fig. 5: **GridSearchCV results for lr for 2 layer MLP**



(a) 2 layer MLP convergence curve results

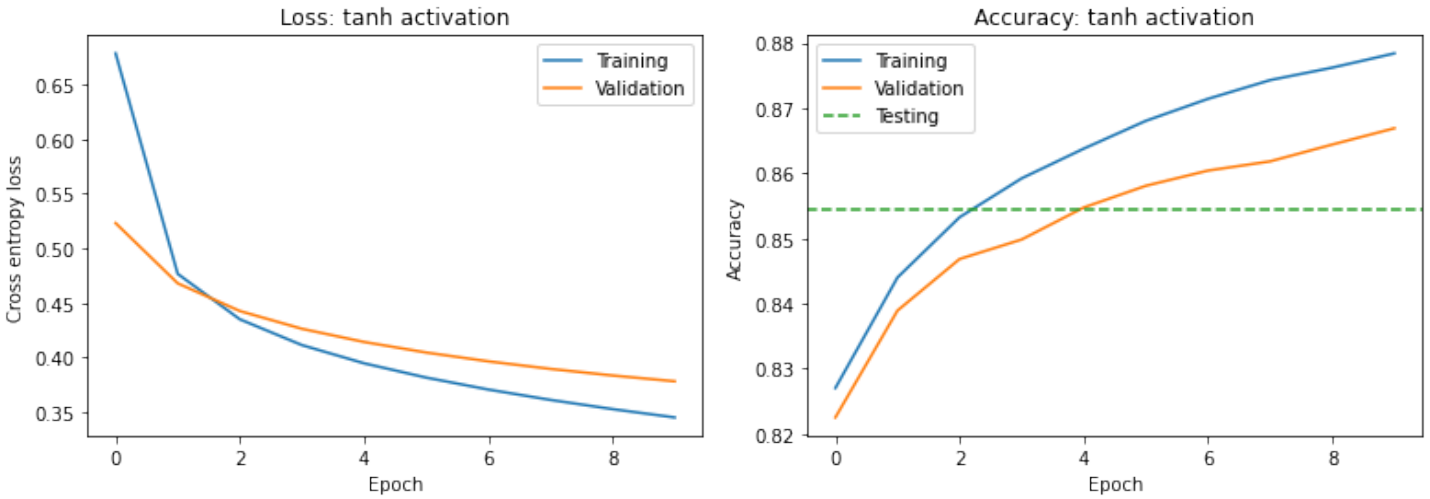
(b) 2 layer MLP accuracy results

Fig. 6: **2 layer experiment results with the optimal lr = 0.08**

APPENDIX C
DIFFERENT ACTIVATION FUNCTIONS ON MLP

C.1 Hyperbolic Tangent

Testing accuracy = 85.44%



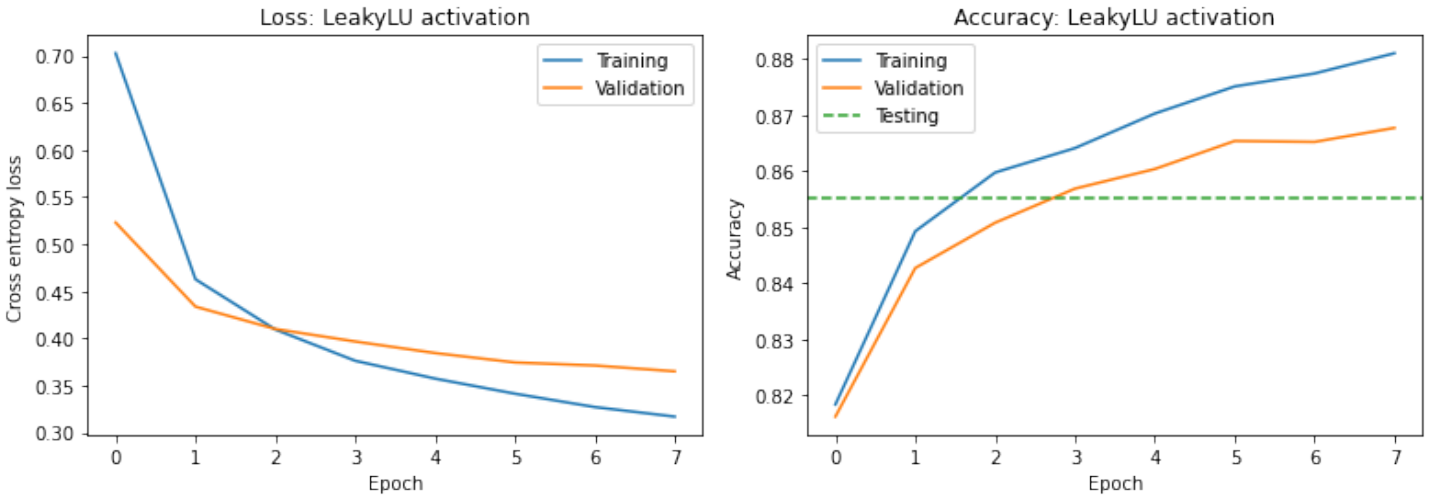
(a) Training and Validation loss curves

(b) Training and Validation accuracy curves

Fig. 7: **Hyperbolic Tangent Activation Results**

C.2 LeakyReLU

Testing accuracy = 85.53%



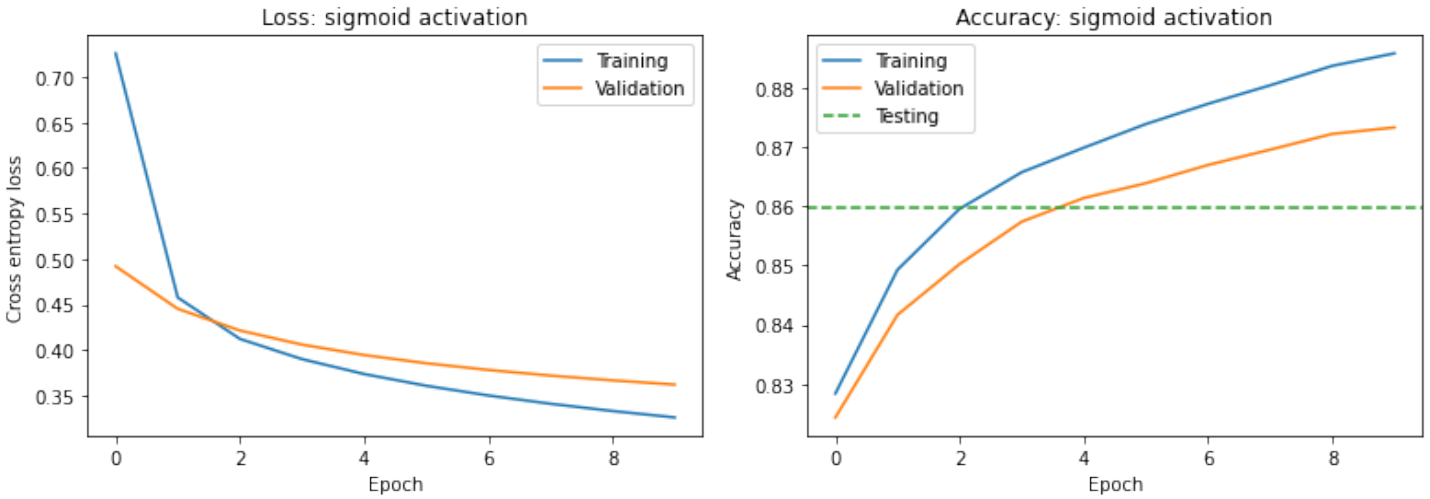
(a) Training and Validation loss curves

(b) Training and Validation accuracy curves

Fig. 8: **LeakyReLU Activation Results**

C.3 Sigmoid

Testing accuracy = 85.97%



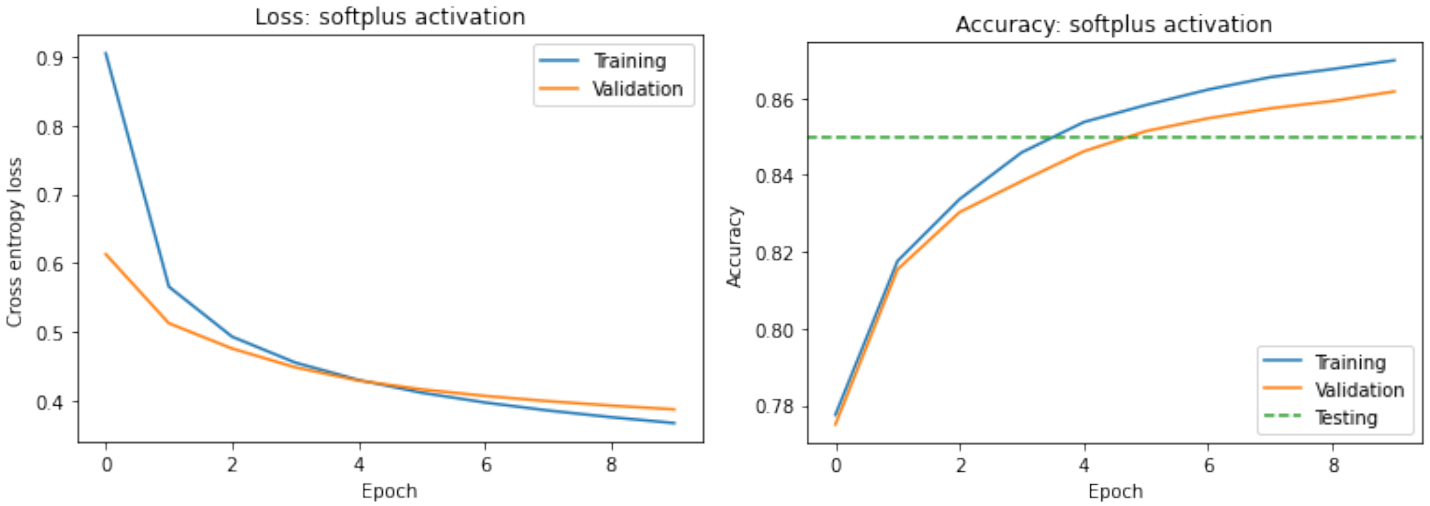
(a) Training and Validation loss curves

(b) Training and Validation accuracy curves

Fig. 9: **Sigmoid Activation Results**

C.4 Softplus

Testing accuracy = 84.97%



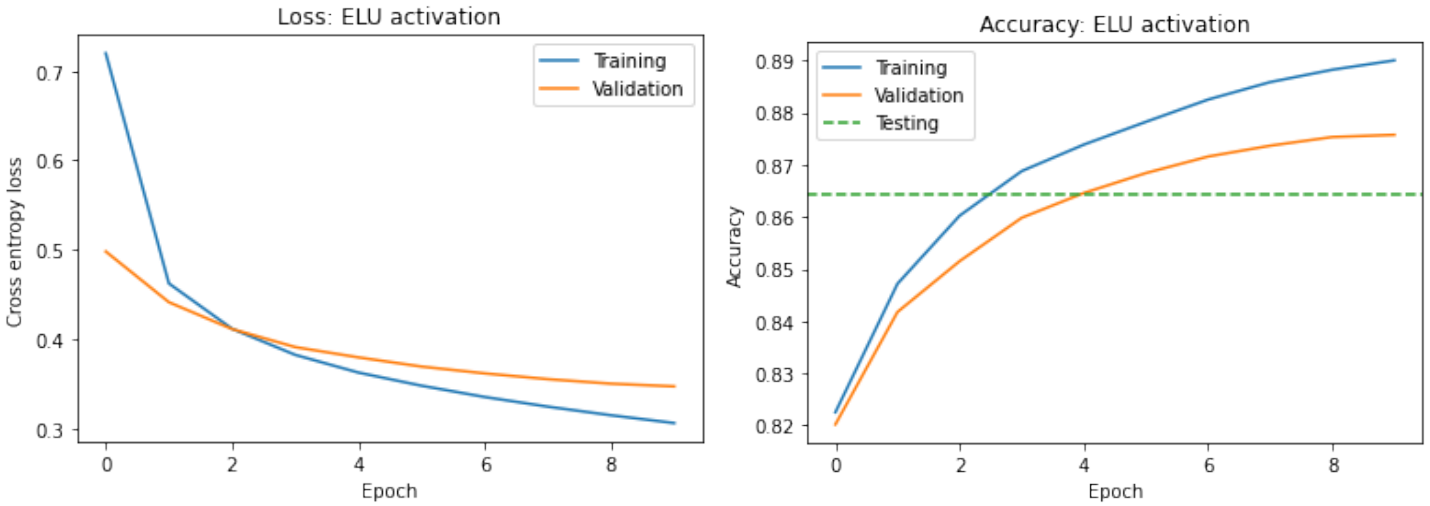
(a) Training and Validation loss curves

(b) Training and Validation accuracy curves

Fig. 10: **Softplus Activation Results**

C.5 ELU

Testing accuracy = 86.44%



(a) Training and Validation loss curves

(b) Training and Validation accuracy curves

Fig. 11: **LeakyReLU Activation Results**

APPENDIX D
L2 REGULARIZATION WITH 2 MLP LAYERS

Figure 12 shows the GridSearchCV results for the lr and λ. We determine the optimal values are lr $= 0.2$ and $\lambda = 0.005$. Figure 13 shows the test accuracy results.

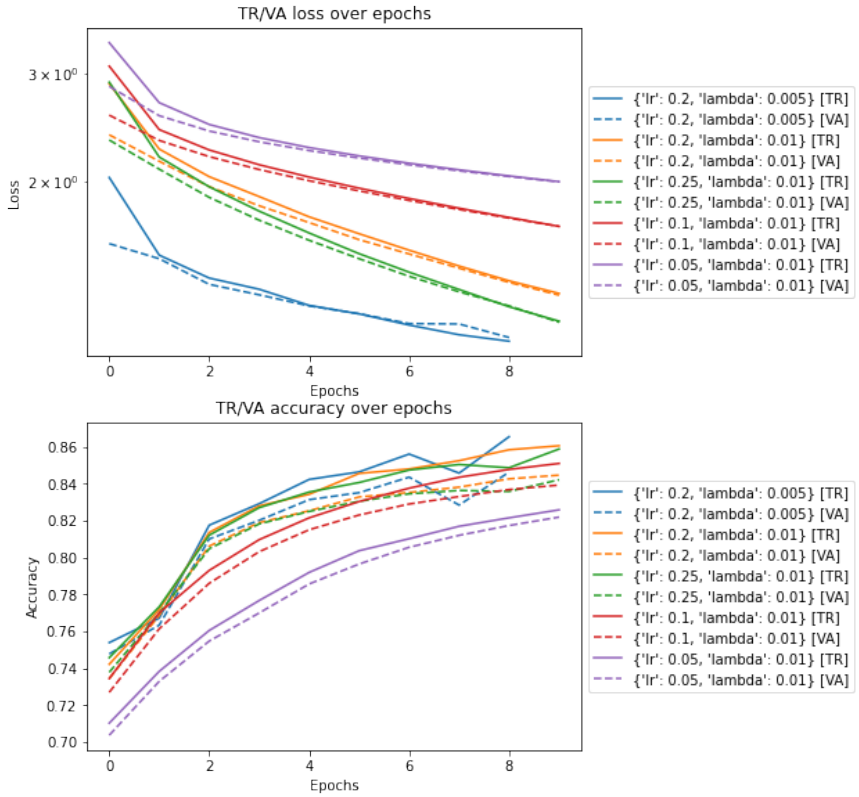
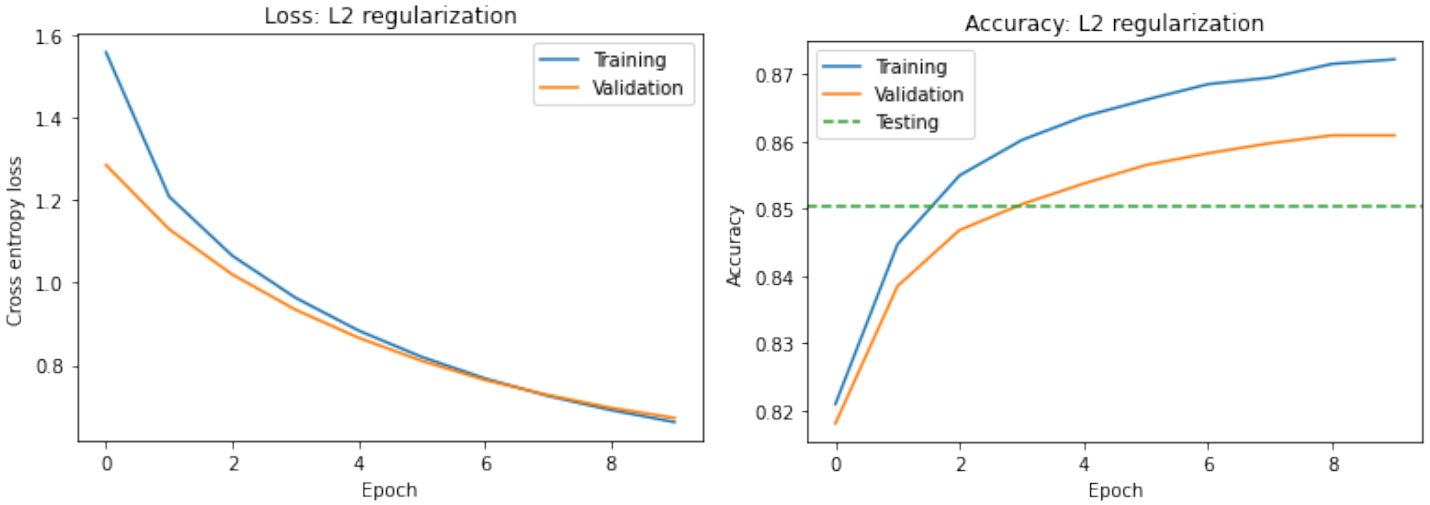


Fig. 12: **GridSearchCV results for lr and λ for L2-regularization 2 layer MLP**



(a) L2 regularization convergence curve

(b) L2 regularization accuracy results

Fig. 13: **L2-regularization experiment results**

Figure 14 shows the GridSearchCV results for the lr and λ with λ varying across layers. We determine the optimal values are lr $= 0.25$ and $(\lambda_1, \lambda_2, \lambda_o) = (0.005, 0.01, 0.005)$. Figure 15 shows the test accuracy results.

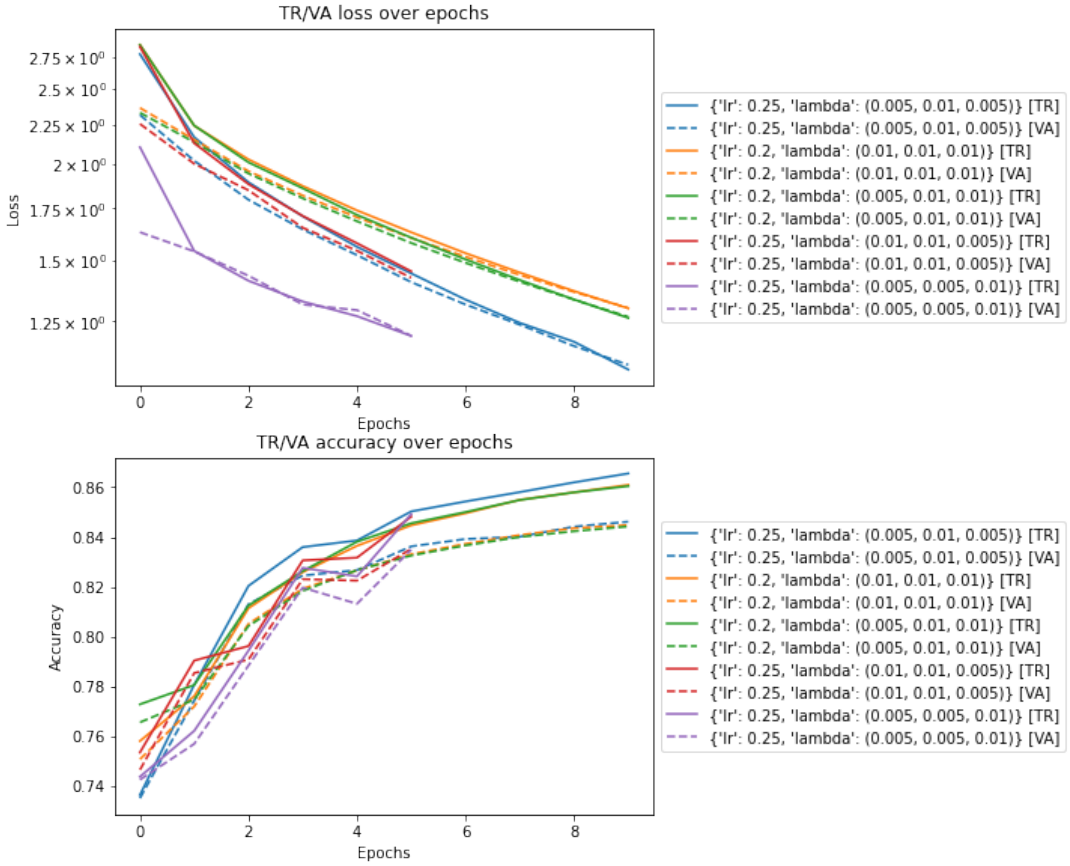
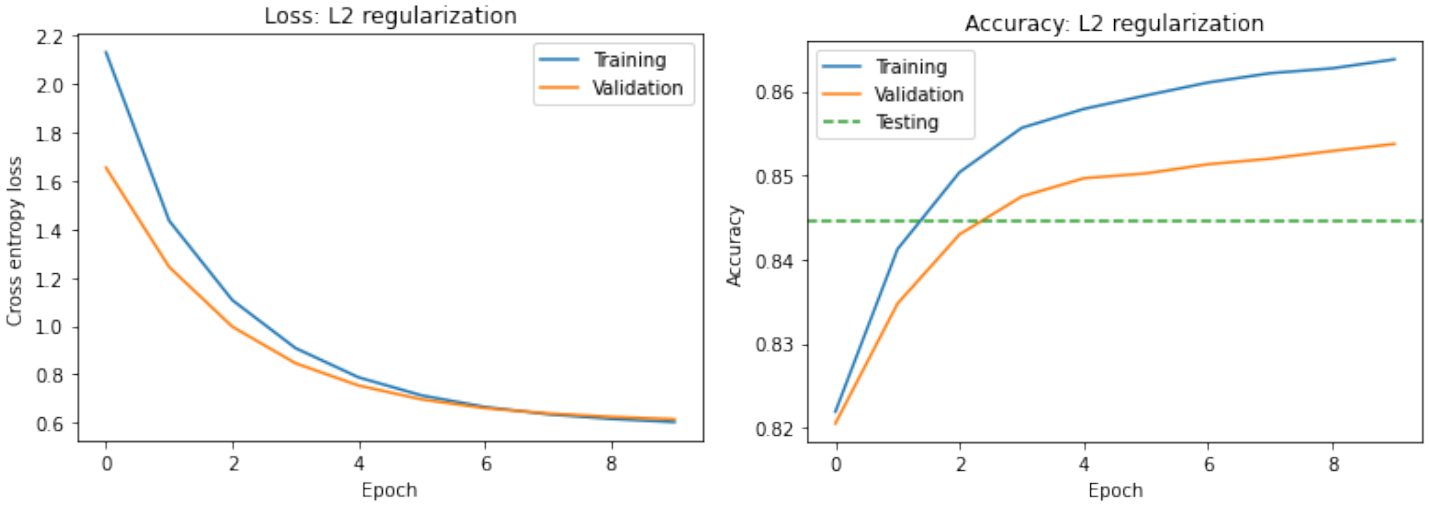


Fig. 14: **GridSearchCV results for lr and λ for L2-regularization 2 layer MLP with varying λ**



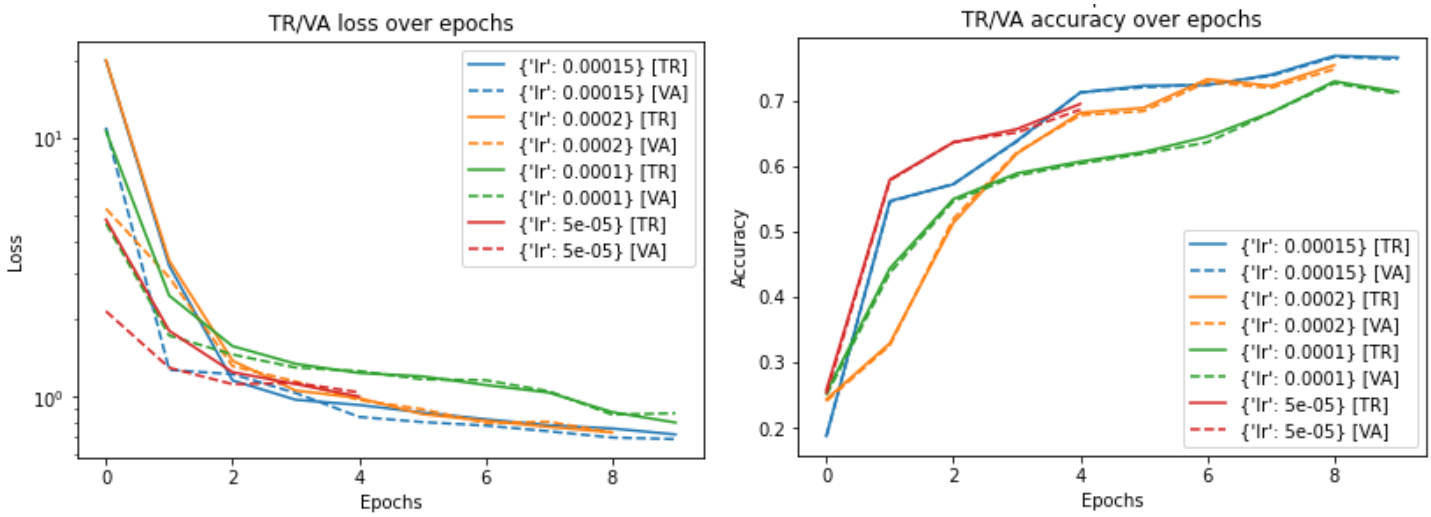
(a) L2 regularization convergence curve

(b) L2 regularization accuracy results

Fig. 15: **L2-regularization experiment results with lr** $= 0.25$**,** $(\lambda_1, \lambda_2, \lambda_o) = (0.005, 0.01, 0.005)$

APPENDIX E
UNNORMALIZED IMAGES WITH 2 MLP LAYERS

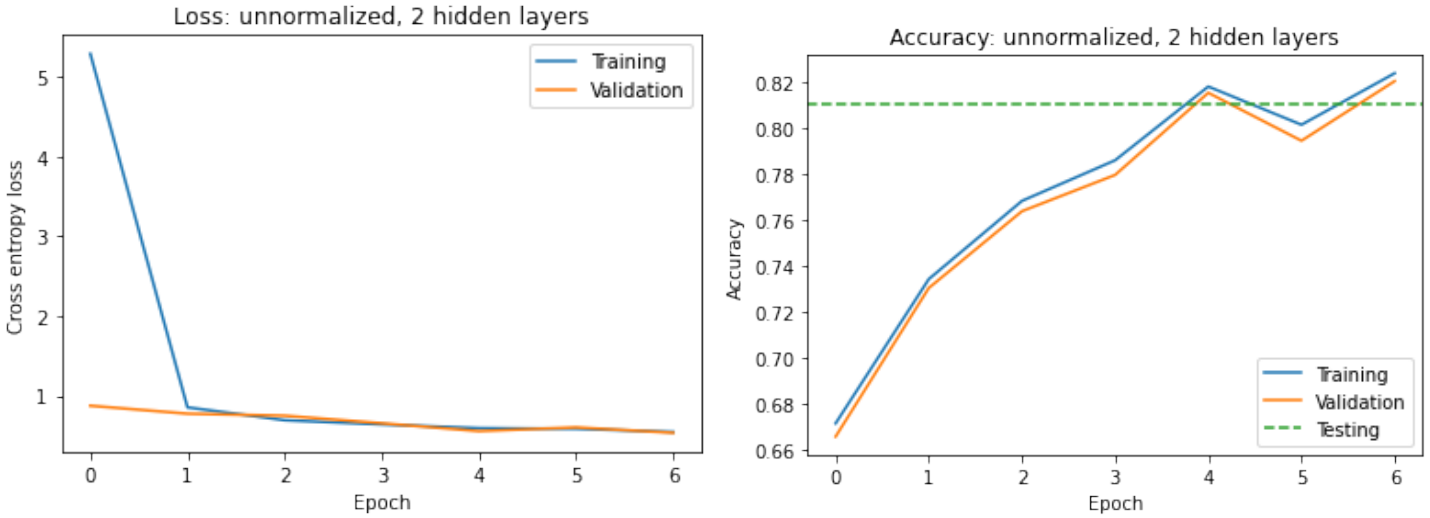
Figure 16 shows the GridSearchCV results for the lr. We determine the optimal values are lr = 0.00015. Figure 17 shows the test accuracy results.



(a) Unnomalized image 2 layer MLP loss over epochs

(b) Unnomalized image 2 layer MLP loss over accuracy

Fig. 16: **GridSearchCV results for lr for unnormalized 2 layer MLP**



(a) Unnomalized images convergence curve

(b) Unnomalized images accuracy curve

Fig. 17: **Unnomalized images with 2 layer MLP experiment results**

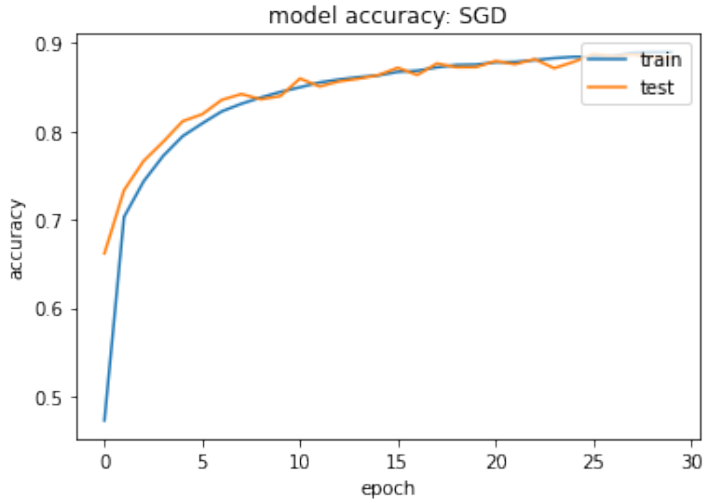
APPENDIX F
CNN MODEL RESULTS

The following table summarizes the CNN model structure.

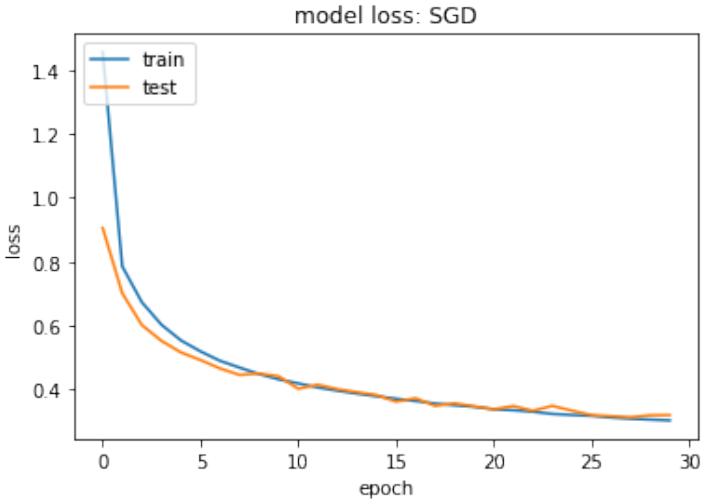
TABLE 6: **CNN model structure**

Layer (type)	Output shape	Number of parameters
Conv_2d	(None,28,28,32)	320
Max_pooling2d_2	(None,14,14,32)	0
Conv_2	(None,12,12,64)	18496
Max_pooling2d_2	(None,6,6,64)	0
Dropout	(None,6,6,64)	0
Flatten	(None,2304)	0
Dense	(None,128)	295040
Dense	(None,128)	16512
Dense	(None,10)	1290
Total parameters	331,658	
Trainable parameters	331,658	
Non-trainable parameters	0	

The following figures show the loss and accuracy curves for the CNN model using the three different optimizers, `SGD`, `adam` and `rmsprop`.

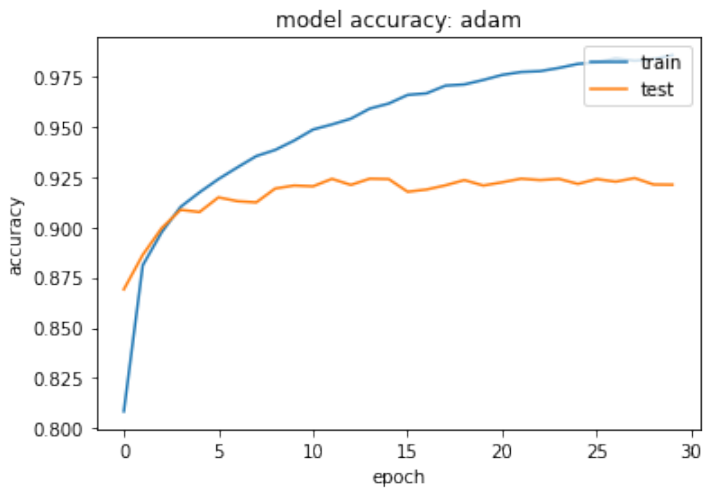


(a) Loss curves for CNN with SGD optimizer

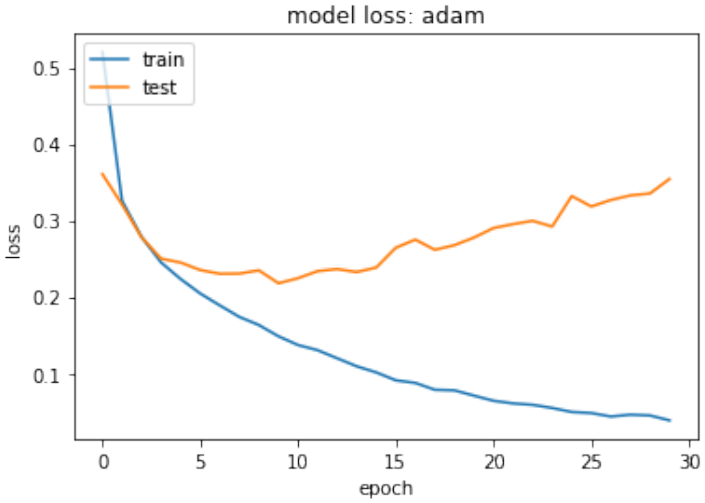


(b) Accuracy curves for CNN with SGD optimizer

Fig. 18: **CNN training results with SGD optimizer**

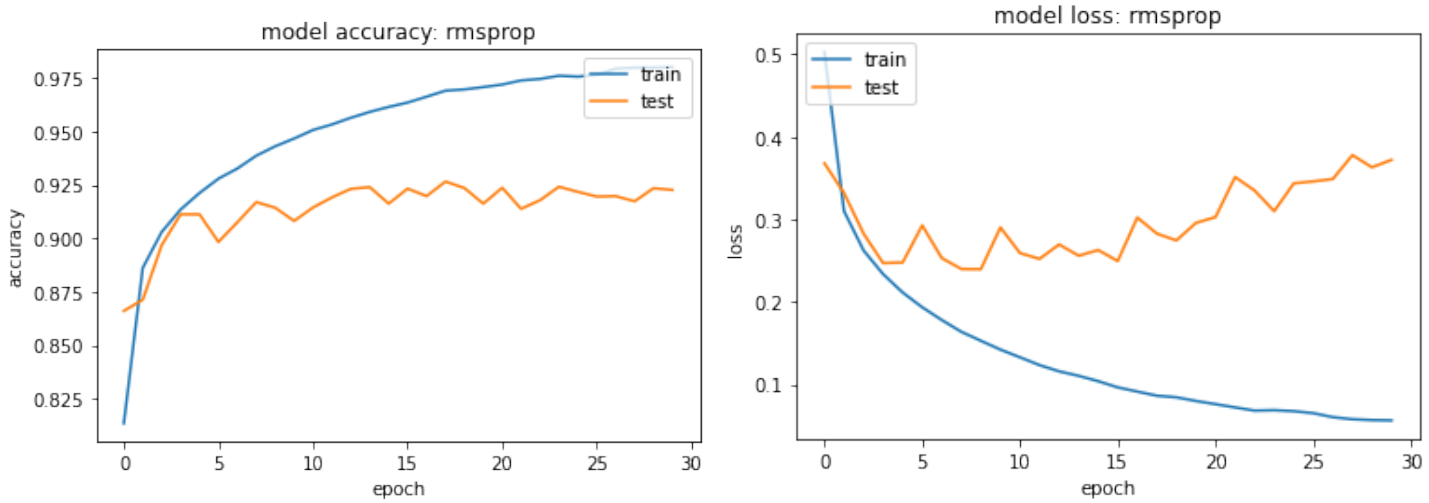


(a) Loss curves for CNN with ADAM optimizer



(b) Accuracy curves for CNN with ADAM optimizer

Fig. 19: **CNN training results with ADAM optimizer**



(a) Loss curves for CNN with rmsprop optimizer (b) Accuracy curves for CNN with rmsprop optimizer

Fig. 20: **CNN training results with rmsprop optimizer**

APPENDIX G
DIFFERENT MLP ARCHITECTURE

TABLE 7: **Different MLP Architecture**

Test	Hidden layers	Hidden units	Activation	Learning rate	Run-time	Test Accuracy (%)
Effects of depth						
1	3	128	All ReLU	0.1	10 min 54s	86.48%
2	4	128	All ReLU	0.15	5 min 03s	85.52%
2	3	128	All tanh	0.1	5 min 59s	85.58%
2	3	128	All Sigmoid	0.05	5 min 59s	53.08%
Effects of width						
3	1	16	ReLU	0.25	34s	85.14%
4	2	16	All ReLU	0.3	40s	84.15%
5	1	32	ReLU	0.2	1 min 05s	85.19%
6	2	32	All ReLU	0.25	4 min	85.87%
5	1	256	ReLU	0.15	7 min 44s	86.37%
6	2	256	All ReLU	0.2	8 min 34s	86.62%
Effects of varying hidden units						
9	2	16, 128	All ReLU	0.15	1 min 11s	84.88%
10	2	128, 16	All ReLU	0.2	1 min	84.91%