# Final Project of BIS 557: Comparisons between different regularization approaches applied on neural network models

**Name**: Kangxin Wang

## Introduction

Deep learning has evolved hand-in-hand with the digital era, which has brought about an explosion of data in all forms and from every region of the world. In this era of big data, data from social media, internet search engines, e-commerce platforms, among others has become readily accessible and contains tons of hidden informations waiting to be revealed. Deep learning, as a subset of machine learning, utilized a hierarchical level of artificial neural networks to carry out the process of machine learning. But one of the issues that is worrisome is that training a deep neural network could relatively easier overfit the data, compromises the generalizability of the model. Some approaches that we have in mind when dealing with this issue are called regularization, but the effect of them may vary over different forms of data. In this extended abstract, some regularizing techniques will be introduced, tested on the MNIST handwritten digits data and model performance has been compared in terms of mean squared error.

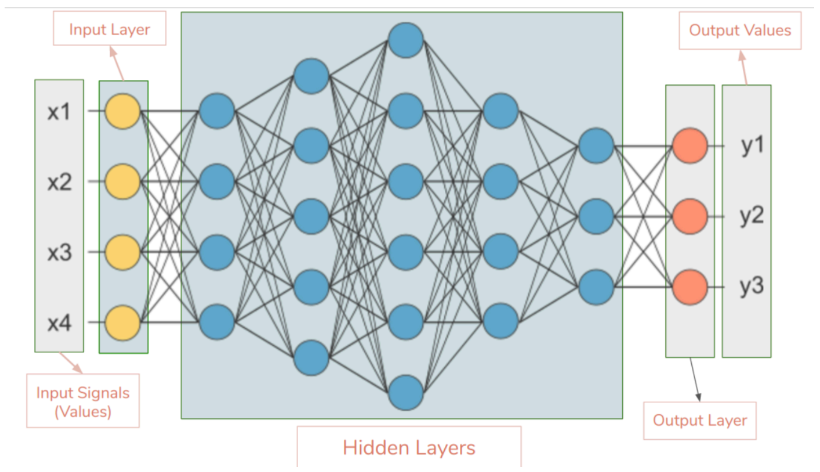
## Data description

In this project, the MNIST handwritten digits dataset has been used, as it is a widely used and deeply understood dataset. The dataset can be downloaded from the website http://yann.lecun.com/exdb/mnist/. The MNIST database contains 60,000 training images and 10,000 testing images. These are images of handwritten digits from 0 to 9, 28 pixels by 28 pixels in size. As this is a well-developed dataset, most standard implementations of neural networks achieve an accuracy of over 98% in correctly classifying the handwritten digits. Each sample image is represented by a numerical vector of length 28*28 = 784, which can be fed into the neural network classification model. A sample of the subset image dataset is shown below.

## Model description

The baseline model utilized the Sequential API from Keras, using 6 hidden Dense layers, each with a ReLU activation function, which is a non-linear function allowing complex relationships providing more sensitivity to the activation sum input and avoid easy saturation. Taking advantage of the Keras API, the model is compiled using Adam optimizer and used Mean squared error loss function. The baseline model is trained with 100 epochs with batch size of 64.

After configuring the baseline model, the training and testing error rate are computed and visualized.



The first regularization method is data augmentation. This is one of the simplest ways to reduce overfitting of neural networks on image data exclusively. By taking advantage of the properties of image data, we can increase the training size by rotating, flipping, scaling, shifting the image, and ect. These transformations of image data will largely increase the size of training data, so as to improve the model accuracy.

The second regularization method used L2 norm via Tensorflow. By adding in weight decay in the dense layers from the baseline model. The basic idea of L2 norm regularization is shown as followed.

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L\left(\hat{y}^{(i)}, y^{(i)}\right) + \frac{\lambda}{2m}||w||^2$$

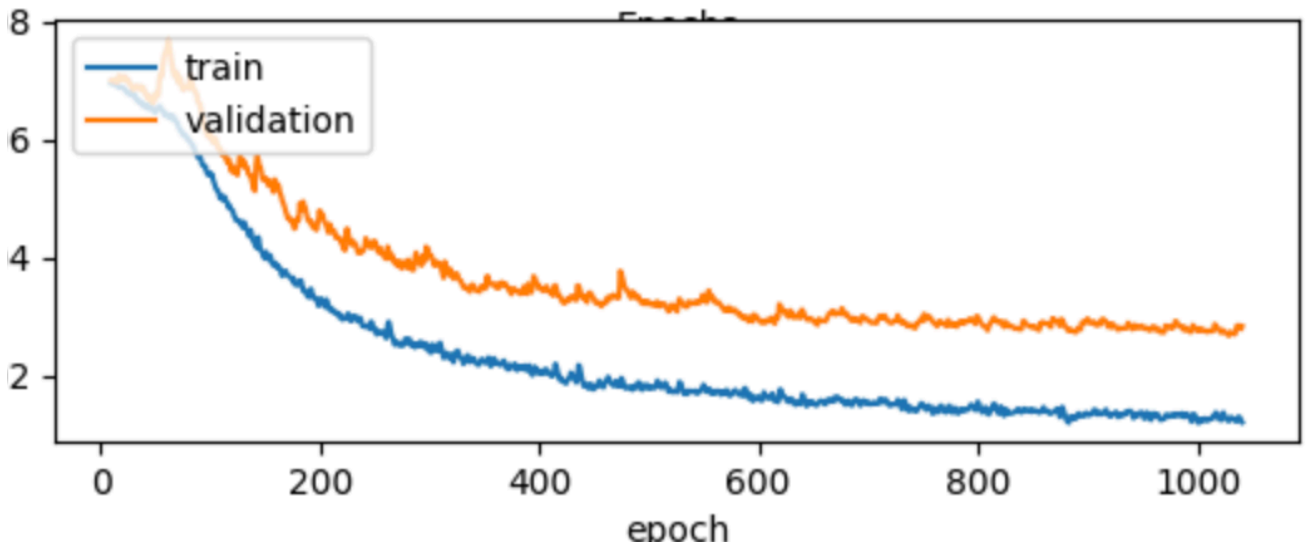
This is the cost function taking weights and biases of the model as arguments. The terms w and b represents the weights and biases that the model has learned correspondingly, and the latter part does the regularization job. The result of this implementation is that the weight matrices of the network are penalized according to the value of the regularization parameter lambda. The lambda, obviously is a hyper-parameter which requires fine-tune. When this value is high, the weight matrices will be heavily penalized, so that the net activations of the neural network are reduced and forward pass effect is diminished. However. If the lambda is well-chosen, the simplified neural network will be more generalizable.

The third regularization that have been used is L1 norm. Similar to the L2 norm, a regularizing term is added to the cost function. However, instead of adding a squared term of w, we only penalize the cost function by the absolute value of the weights.

$$Cost\,function = Loss + \frac{\lambda}{2m} * \sum \|w\|$$

## Methodology

As the aim of this project is to compare the effectiveness of different regularization approaches taking care of the overfitting issue, we compare the validation loss and training loss of the models over epochs. If the validation loss is always greater than training loss, that would suggest the model has overfitted the data, since it couldn't retain the same predictive accuracy on validation dataset. Besides, the overall difference between the validation loss with training loss also suggests the severity of the overfitting issue. The codes visualized the results from the above proposed models and will be discussed in the next session.



## Results and Discussion

```
model.compile(loss='mse', optimizer='adam', metrics=['accuracy'])

trained_model = model.fit(train_data, train_targets, nb_epoch=100, batch_size=128,
                          validation_data=(test_data, test_targets))
```

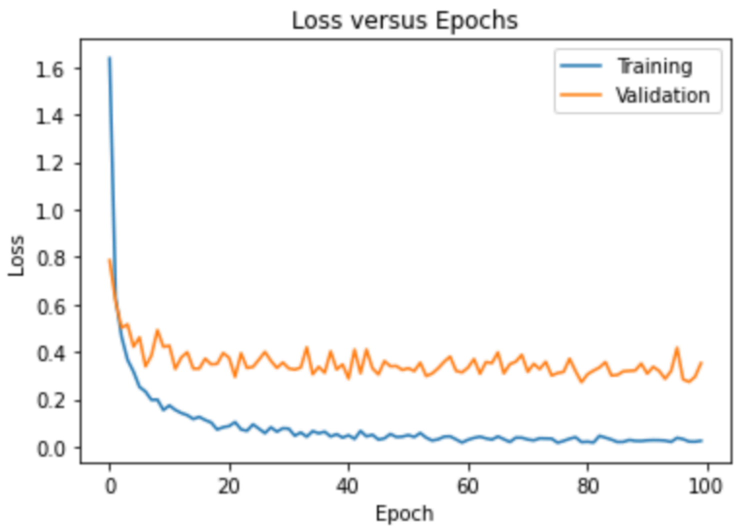
The above is code for building up the baseline model architecture. The model utilized Adam optimizer, chose mean squared error as its loss function, and calculated both mean absolute error and model accuracy. Trained on 53550 samples, validated on

9450 samples, over 100 epochs, the following loss comparison is plotted for the un-regularized model.

```
def get_model():
    model = Sequential([
        Dense(units = 128, activation = 'relu', input_shape = (train_data.shape[1],)),
        Dense(units = 128, activation = 'relu'),
        Dense(units = 128, activation = 'relu'),
        Dense(units = 128, activation = 'relu'),
        Dense(units = 128, activation = 'relu'),
        Dense(units = 128, activation = 'relu'),
        Dense(units = 1)
    ])
    return model
model = get_model()
```

```
WARNING:tensorflow:From /opt/anaconda3/lib/python3.7/site-packages/tensorflow/python/ops/init_ops.py:1251: calling Va
rianceScaling.__init__ (from tensorflow.python.ops.init_ops) with dtype is deprecated and will be removed in a future
version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the constructor
```

```
# Compile the model
model.compile(optimizer = 'adam',
              loss = 'mse',
              metrics = ['mae', 'accuracy'])
```

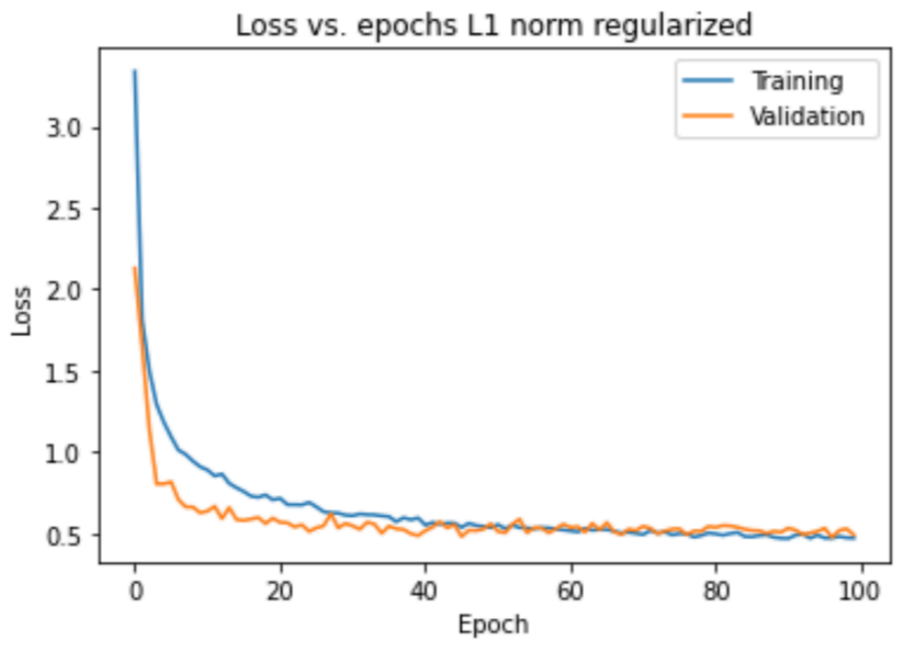


From the above graph, as we discussed earlier, the validation loss is almost always greater than the training loss, and as epoch goes up, the difference between validation loss and training loss remains relatively constant, while the spikes in validation loss is observably more obvious than that in training loss. This shows that the un-regularized model indeed encountered some degree of overfitting issue.

```
model.evaluate(test_data, test_targets)
```

```
7000/7000 [==============================] - 0s 69us/sample - loss: 0.5096 - mean_absolute_error: 0.1900

[0.5096040847131185, 0.19003473]
```



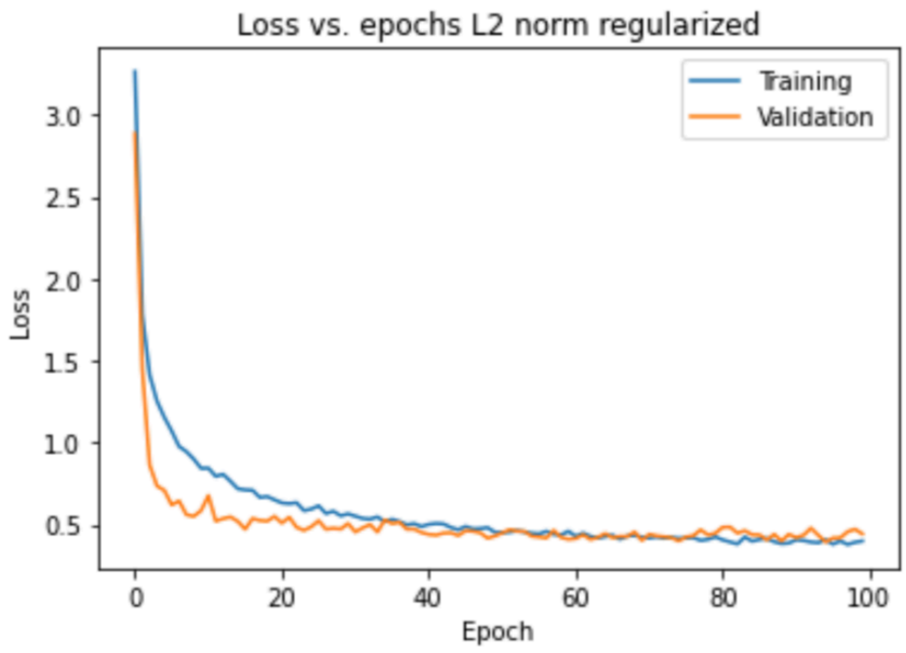
The above graph shows the comparison between training loss and validation loss in the model regularized by L1 norm. An immediate effect can be observed as compared to the un-regularized model, both validation and training loss decayed as epochs went up, and as expected, the validation loss curve is much smoother, most importantly, it suggest that the validation loss is lower than training loss, which means that the overfitting issue has been largely resolved by L1 norm regularization.

```
# Evaluate the model on the test set
model.evaluate(test_data, test_targets)
```

```
7000/7000 [==============================] - 0s 61us/sample - loss: 0.3984 - mean_absolute_error: 0.1739

[0.398417415363448, 0.1739378]
```

The following graph shows the comparison between training and validation loss in the model which is now, regularized by L2 norm. Comparing to the previous L1 norm, we can hardly visually see much difference in terms of the effectiveness of dealing with overfitting, the validation curve seems to be a little less than that in L1 norm. When epoch is 0, the validation loss is clearly much larger than that in the last scenario.



---

Further discussion comparing L1 and L2 norm regularization:

The fundamental difference between L1 and L2 regularizations is that in L2 regularization, the entries of weight matrices decrease, but not necessarily become zero, since it is of the quadratic form. However L1 regularization forces the weights all the way to zero.

## Conclusions

This project compared several regularization techniques that could be applied to deep learning models to help reducing the overfitting issues. As discussed, on MNIST handwritten digits dataset, each of these techniques successfully taken care of the issue. What's worth-noticing is that the data augmentation for image dataset is relatively easily done, as simply rotating, flipping the image would effectively increase the training size, however in practice, not all forms of data can be augmented in such ways. For non-image dataset, one could try to add random (Gaussian noise to a subset of the training data so as increased the training dataset. The exact augmentation process will be largely decide by domain knowledge. L1 and L2 norm regularizations have similar effects on the MNIST data, but fundamentally L1 encourages the weight parameter values to be zero, while L2 encourages the weight parameter values towards zero. In practice, fine-tuning the penalization parameter is rather important as it directly effect on how much simplified the model is after regularization.