Assignment-3 Neural Networks ELL 409

Jaskeerat Singh Saluja (2019MT60752)

November 11, 2021

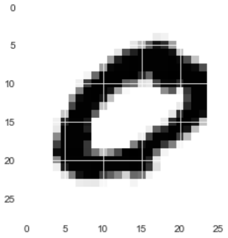
Table of contents

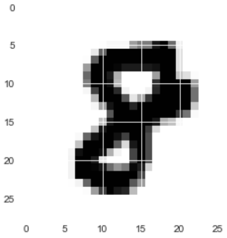
Theory and python implemetation

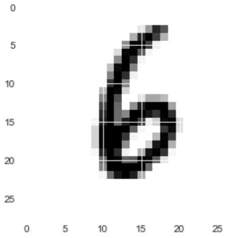
- The neural nets class have been implemented in neuralNet.py .
- The class has flexibility for neuron type , cost-function , eta ,regularization ,batch-size.
- Several activation function are also available and are imolemented in neuron.py
- To perform cross validation GridSearchCv class has also been implemeted in utils/search.py
- For activation function : 'sigmoid' and 'tanh' are used .
- To specify the structure of neural net , list represtation is used to initilize the network.
- For example [n1,n2,n3..nk] implies k layers with n1 as input and nk as output layer , hence total of k-2 hidden layers.

Visualizing the given data

On transforming the feature vector from $(784, 1)$ to $(28, 28)$ size and plotting we have the following images provided.







Using Sigmoid activation function

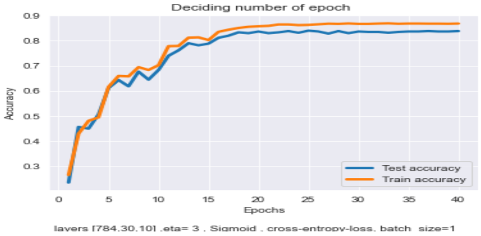
Finding best number of epochs

1. Neural nets gets overfit on high epoch and are underfit on low number of epochs
2. Epoch =25 is good value to start the tuninng the neural net.

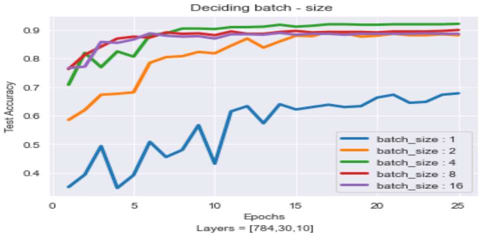
Tuning the mini-batch -size

1. On right , the plot shows variation in test accuracy vs batch-size
2. $batch\_size = 4$ and $epoch = 20$ works well

Batch size =4 , epochs =20



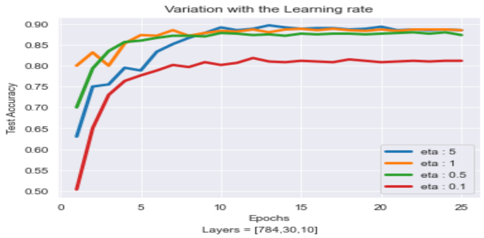
layers [784,30,10] ,eta= 3 , Sigmoid , cross-entropy-loss, batch_size=1



Sigmoid Activation continued ....

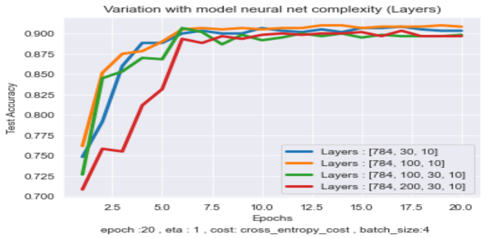
Finding best learning rate η

1. SGD is sesitive to learning rate.
2. The SGD implementation uses **decaying learning rate strategy**



Tuning number and size of layers

1. More number of hidden layers represents higher abstraction of the neural net.
2. Choosing least complex neural net with good test accuracy is the goal.

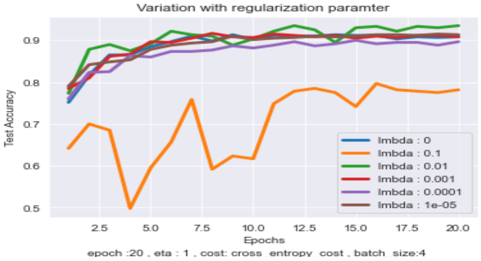


Layers $[784, 30, 10]$ is least complex and performs as good as $[784, 100, 10]$ and learning rate $\eta = 1$

Sigmoid Activation continued ....

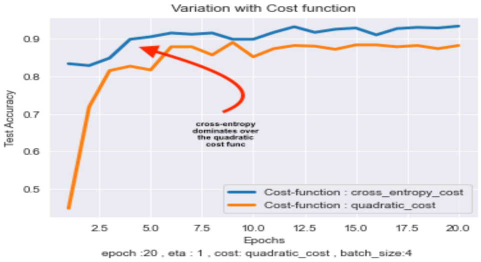
L-2 regularization λ

1. Regularization (L-2) forces w to take lower values of lmbda .Thus prevents model from over-fitting
2. λ in range $[1e-2, 1e-3]$ is better hyperparameter space to work with.



Choosing the cost function

1. Cross entropy learn much faster than the quadratic loss function , since qudratic cost function have a phenomenon called **slowing down**
2. Figure on right display the **slowing down** nicely.



Sigmoid Activation continued ....

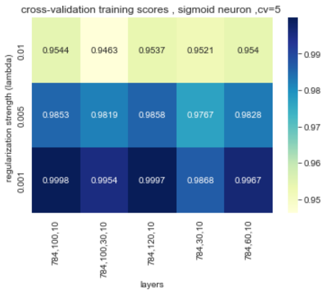
So far now we have narrowed down our hyperparameter space to small subspace where we can perform k-fold cross validation and further tune the best parameters.

epoch	15
mini batch size	4
learning rate	1
layers	$[784, 30, 10]$, $[784, 100, 30]$
lambda	$[1e-2, 5e-3, 1e-3]$
Cost function	cross entropy loss

We have 6 possible good configuration for our sigmoid activation function . We now perform k-fold cross validation to get the best hyperparameter . (CV =5)

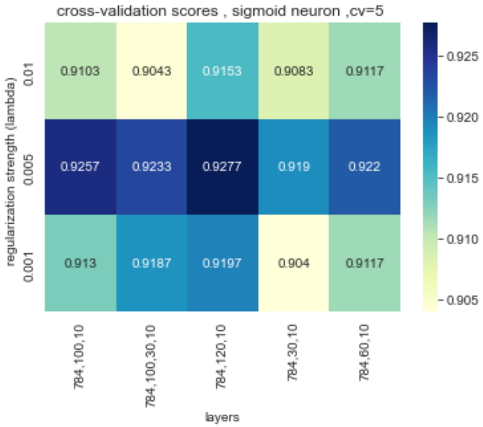
Cross-Validation (cv=5)

1. On rights we have variation of lambda vs neural net layers
2. Best Test accuracy 92.77 percent and corresponsing Train accuracy 98.58



CV training scores

Best hyperparameters : Layer sizes: [784,120,10] and $\lambda = 5e-3$



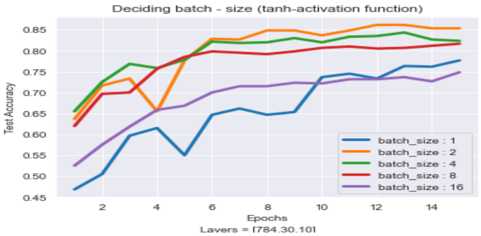
CV Test scores

Using tanh activation function

- **epoch=15** ,Cost function : **cross entropy cost**

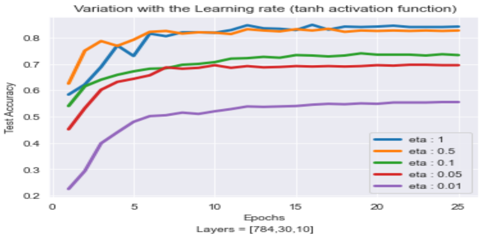
Tuning the mini-batch -size

1. On right , the plot shows variation in test accuracy vs batch-size
2. *batch_size* = 4 works well



Learning rate

1. On right , the plot shows variation in test accuracy vs learning rate
2. $\eta = 0.5$



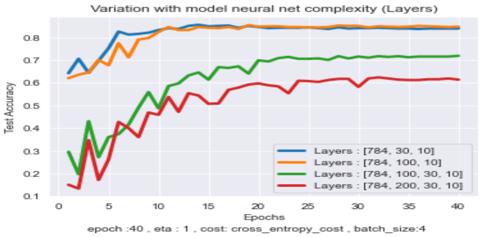
tanh function continued ....

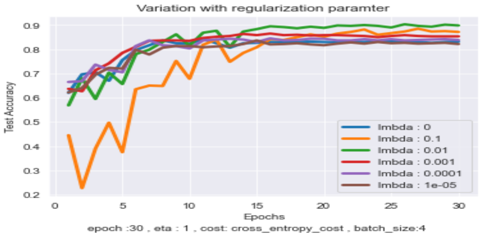
Tuning the layers

1. On right , the plot shows variation in test accuracy vs different layers in neural nets
2. layer =[784,30,10] works quite well , even with low complexity.
3. **Observation :** 2 hidden layer networks took more epoch before settling

Tuning the regularization strength

1. On right , the plot shows variation in test accuracy vs different λ
2. $\lambda = [1e-1, 1e-2, 1e-3]$ is good range where model performs good





tanh Activation continued ....

So far now we have narrowed down our hyperparameter space to small subspace where we can perform k-fold cross validation and further tune the best parameters.

epoch	20
mini batch size	4
learning rate	0.5
layers	$[784, 30, 10]$, $[784, 100, 30]$
lambda	$[1e - 1, 5e - 2, 1e - 2, 5e - 3]$
Cost function	cross entropy loss

We have 8 possible good configuration for our sigmoid activation function . We now perform k-fold cross validation to get the best hyperparameter . (CV =5) . Will take 12-15 minutes to train

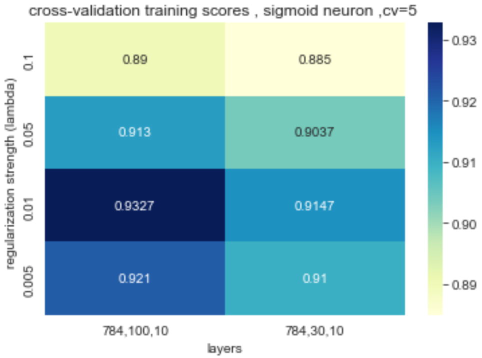
tanh Activation continued ....

Cross-Validation (cv=5)

1. On rights we have variation of lambda vs neural net layers
2. Best Test accuracy **93.27** percent and corresponsing Train accuracy **99.95**

Observations

- **tanh activation(93.27) performs better** than sigmoid neuron (92.77) by a significant 0.5 improvement.
- $\lambda = 0.01$ and layer network =[784,100,10] works best



CV Test scores

Comparison with Tenserflow library

1. Using 90:10 split of data , then the best validation score using the **tanh** activation function achieved is 92.89 which is quite close to our 93.27 cross validation scores.
2. Using 90:10 split of data , then the best validation score using the **sigmoid** activation function achieved is 92.09 which is quite close to our 92.77 cross validation scores.

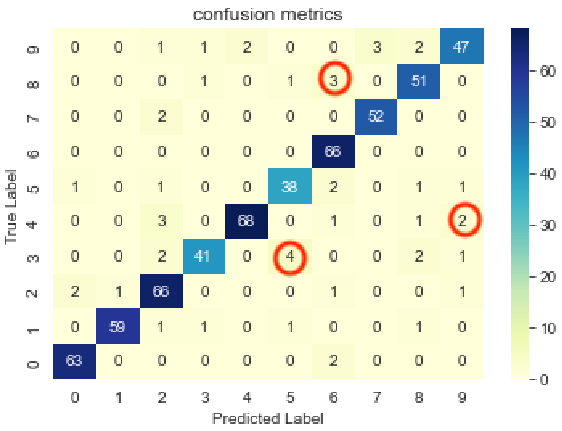
Analysing errors made by neural nets

Observations :

- On right we have confusion metric ,on **test data** which is never seen by the neural net.
- 4 is confused with 9
- 3 is predicted wrongly as 5
- 8 with 6

Reasoning

- Digits with similar geometry are the ones wrongly predicted by the neural net
- Digits which donot share geometry like (1,6) are never wrongly predicted.



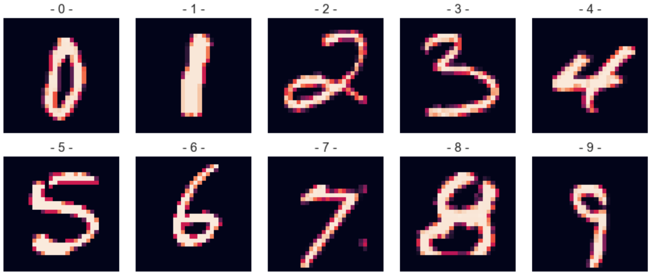
True Label \ Predicted Label	0	1	2	3	4	5	6	7	8	9
9	0	0	1	1	2	0	0	3	2	47
8	0	0	0	1	0	1	3	0	51	0
7	0	0	2	0	0	0	0	52	0	0
6	0	0	0	0	0	0	66	0	0	0
5	1	0	1	0	0	38	2	0	1	1
4	0	0	3	0	68	0	1	0	1	2
3	0	0	2	41	0	4	0	0	2	1
2	2	1	66	0	0	0	1	0	0	1
1	0	59	1	1	0	1	0	0	1	0
0	63	0	0	0	0	0	2	0	0	0

epoch =10 , tanh activation neuron

Visualizing the hidden layers

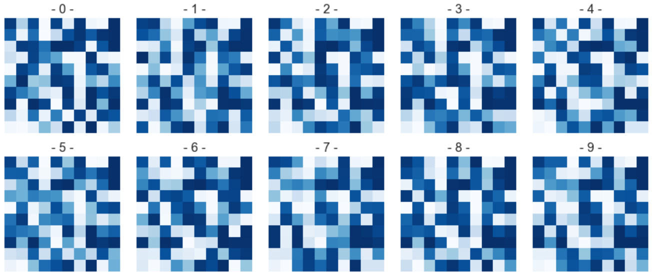
1. Neural net model with layers [784,100,10] trained .
2. Below we visualize the ouput by each layer of the neural net.

Outputs First Layer (Input Layer)



Visualizing the hidden layers

Outputs Second Layer (Hidden Layer)



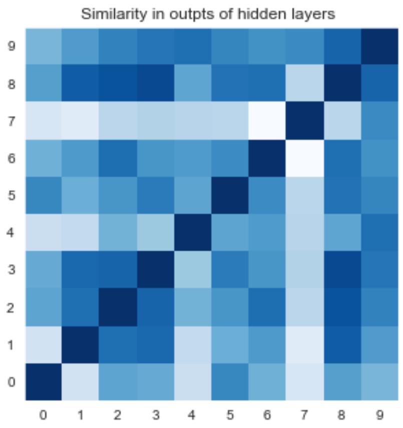
- These patterns look sufficiently random - except for the digits (4 , 9) , (3 , 5) , which look remarkably similar
- Above **each pixel is mean activation output** from a neuron in hidden layer.
- **Dark blue** mean that specific neuron is active while **white means** that neuron is inactive.
- **NOTE:** The neuron in the bottom left corner is only **active** when input is 1 , while it is inactive for inputs other than 1. **Possible reason:** 1 is only geometric shape which is just a straight line i.e no curve /bends . All other digits have curved parts . **This is a possible reason**.

Visualizing hidden layers

- Since the outputs are (100,1) in the hidden layer, representing the activations of the neurons.
- Thus to check for similarity is outputs of digit i with digit j . One solution is to take dot product of the mean squared outputs vectors for ith and jth digit.

Conclusions

- Despite variations in the shapes of hand-written digits, the same groups of neurons is involved in the identification of the same digits.
- Similarities in the shapes of digits translate into similarities in the groups of neurons that are involved in their identification in the first hidden layer, but not so much in the second hidden layer



PCA and Neural networks

Dimensionality reduction in neural networks

- When number of hidden units <number of input units . The net performs dimensionality reduction of input image to low dimension representation.
- Each synthesized dimension (each hidden unit) is logistic function of inputs $h_k(x) = \frac{1}{1+\exp(w_0+\sum_{i=1}^{N} w_i x_i)}$.
- Hence an N-input input layer when input to M unit hidden input layer , we get non-linear transformation of the N-dimesnion to M-dimension data.

PCA

- Given data points in d-dimensional space, project into lower dimensional space while preserving as much information as possible (linear transformation)

Training on PCA data

No hidden layer (Logistic regression)

- Neural net 25 – 10
- The PCA data contained 25 feature representation of the image data given to us in previous assignment .
- Training neural net with no hidden layer directly on this gives us model with best testing accuracy of 85 percentage.

1 hidden layer (17 hidden units)

- Neural net 25 – 17 – 10
- Best accuracy (testing)achieved is 89.05 percent , with 91 percent training accuracy.
- Adding one hidden layer help improve the accuracy of the model significantly , because neural net performs non-linear tranformation of this PCA data and learns best out of it.

2 hidden layer (20 and 15 hidden units)

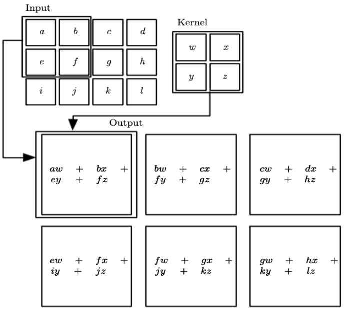
- Neural net 25 – 20 – 15 – 10
- Best accuracy (testing)achieved is 90.1 percent , with 92 percent training accuracy.
- Adding another hidden layer help improve the accuracy of the model slightly.

Comparison with raw pixel data

- Neural net trained on 784 pixel data (94 percent accuracy) performed way better than the neural net trained on PCA features (90 percent accuracy).
- Though PCA does not throw away every other pixel and it only transforms the data to have important features. Keeping maximum variance in the dataset.
- Reducing Dimensions in an image where pixels are the features, would mean downsampling the image

Using Convolutional networks

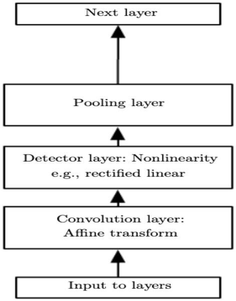
- Convolution solves three important ideas that improves a machinelearning model:
  - sparse interactions : In ANN every output unit interacts with every input unit . Convolution network uses sparse connections by taking kernel smaller than the input.
  - parameter sharing
  - equivariant representations



Stages in Convolutional network

A typical layer of a convolutional network consists of three stages:

- In the first stage, the layer performs several convolutions in parallel to produce aset of linear activations.
- In the second stage, each linear activation is run through a nonlinear activation function, such as the reLU.
- In the third stage, we use apooling function to modify the output of the layer further
- A pooling function replaces the output of the net at a certain location with asummary statistic of the nearby outputs
- For example : MAX Pooling ,operation reports the maximum output within a rectangularneighborhood.
- Now this 2D data is Flattened and fed into our old ANN networks.
- The first 3 steps performs the feature extraction , thus we then train the neural net on this.



Regularization

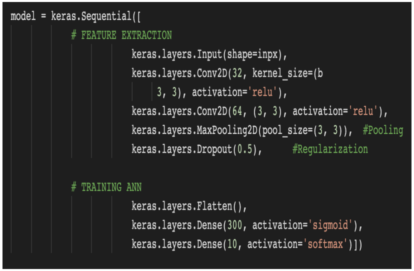
For regularization we can use Dropout after the pooling layer.

CNN on MNIST DATA

- We used 1 layer of convolution followed by pooling . Then we use Dropout for regularization purposes .
- After 3 of above layers , we then perform our old ANN on the activation function.
- The code below nicely explains our neural net structure.

Results

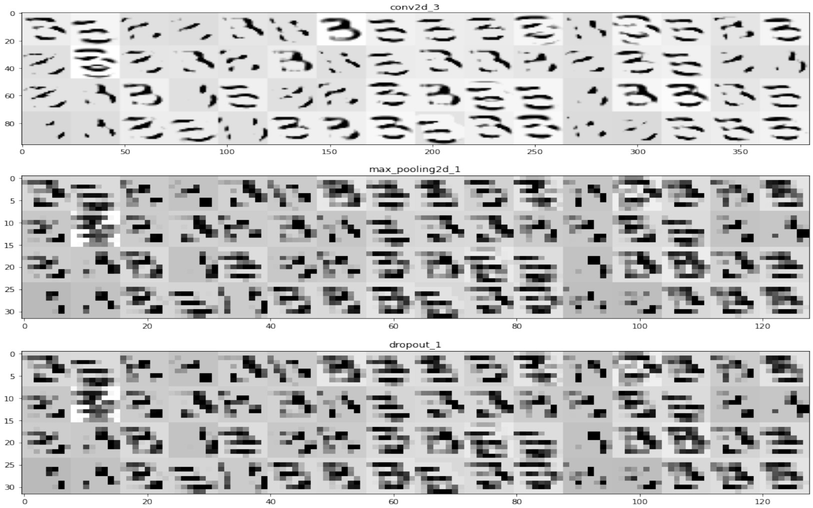
- Training acccuracy =99.36 percent
- Testing accuracy = 99.22 percent



```
model = keras.Sequential([
            # FEATURE EXTRACTION
                        keras.layers.Input(shape=inpx),
                        keras.layers.Conv2D(32, kernel_size=(b
                             3, 3), activation='relu'),
                        keras.layers.Conv2D(64, (3, 3), activation='relu'),
                        keras.layers.MaxPooling2D(pool_size=(3, 3)),  #Pooling
                        keras.layers.Dropout(0.5),      #Regularization

            # TRAINING ANN
                        keras.layers.Flatten(),
                        keras.layers.Dense(300, activation='sigmoid'),
                        keras.layers.Dense(10, activation='softmax')])
```

Visualization of hidden layer neurons

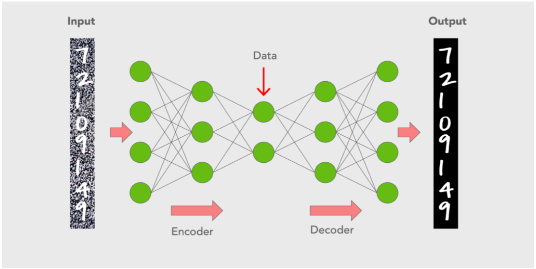


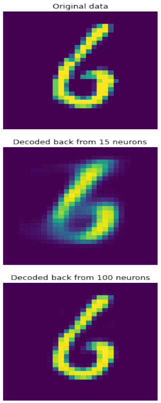
Observations

- This first layer retains almost the full shape of the image, and also most of the information present in the image
- As we go deeper into the network we can see that the activations become more complex and abstract. It starts encoding high-level features such as edges, curves and angles.

Using AutoEncoders

- Auto-encoders are neural nets used for feature extraction . It is unsupervised algorithms , i.e does not used labels.
- The Key idea is to transform image into low dimesnion called encoder network ,the reconstructing the image back from the decoder network . The model is trained so that it does this task with least loss.
- Hence we get non-linear trandformation of image to lower dimensions.





Final Conclusions

Observation

I have a very simple autoencoder neural net i.e just 3 layers with 784,15,784 . Then performed Training on it.

- Still with so simple autoencoder network we are able to reconstruct the digit images .
- Now after training we encode the training images and get the encoded outputs
- Train another ANN for the encoded-outputs vs labels (Supervised)
- Got 96 percent validation accuracy with so simple network.

Further Imrovements

- We can use CNN to train the auto-encoder network. This would improve the model drastically.
- There have been reports with deep CNN auto-encoder network with 15 encoding dimesion , giving more than 98 percent accuracy.

Thanking you