Task 2

1. Train a multilayer perceptron on the MNIST dataset using the traditional train/test split as given by mnist.load\_data in keras.

2. Use a separate 10000 samples (from the training set) for model selection and to compute learning curves (accuracy vs epochs, not vs n\_samples)

3. Compare a “vanilla” model with a model using drop-out. Visualize learning curves for all models.

Solution

Part 1

Train a multilayer perceptron on the MNIST dataset using the traditional train/test split as given by mnist.load\_data in keras.

We have used “adam” optimizer with hidden nueron size of 32 and hidden layer with “tanh” activation, in last layer we have used softmax for mutiple category of target variable

*def make\_model(optimizer='adam', hidden\_size=32):*

*model = Sequential([*

*Dense(hidden\_size, input\_shape=[784,]),*

*Activation('relu'),*

*Dense(hidden\_size),*

*Activation('tanh'),*

*Dense(10),*

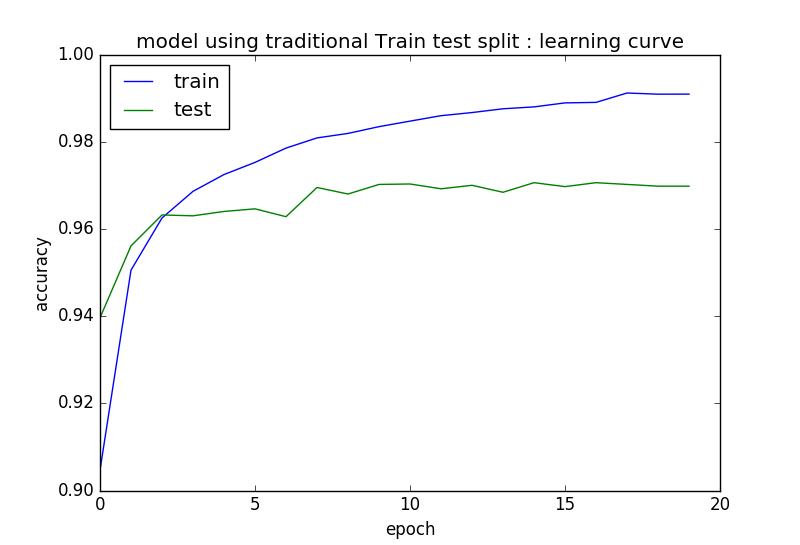
*Activation('softmax')])*

*# fitting Multi-layer perceptron to the data using keras*

*model1=clf.fit(X\_train, y\_train, validation\_data=(X\_test, y\_test),epochs=20, verbose=1)*

We have considered 20 epochs ( This is the number pass through the full traing data ) and 20% of the data as validation dataset . Verbose = 1 ( shows the progress bar )

we can see post 5 epoch there is no significant gain in accuracy for test datasets.



Accuracy on test data with epoch 5 is 0.966

Here also we have saved the model and its weight to save the training time.

Task2\_model1.json

Task2\_model1.h5

have also written the code to load and test the results.

Part 2

Use a separate 10000 samples (from the training set) for model selection and to compute learning curves (accuracy vs epochs, not vs n\_samples)

Here we have taken 10000 sample from the train data itself as suggested above.

*X1=X\_train[:50000]*

*Y1=y\_train[:50000]*

*X2=X\_train[50000:]*

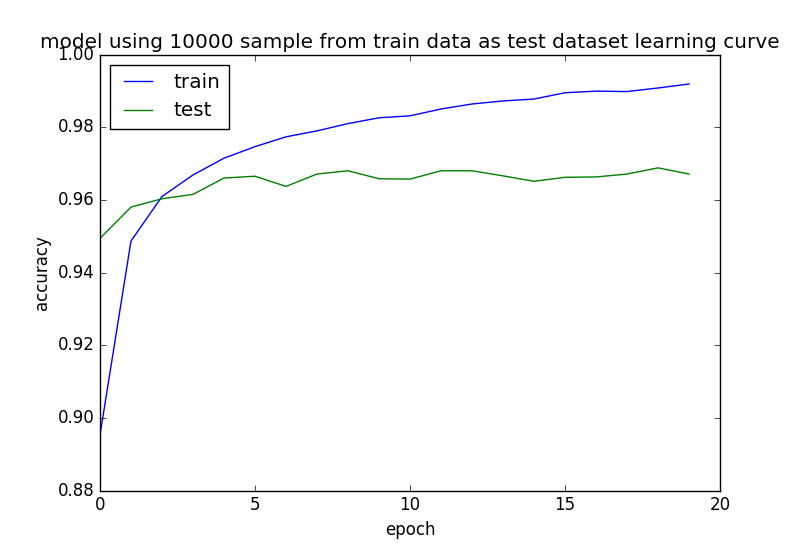
*Y2=y\_train[50000:]*

*num\_classes=10*

*Y2 = keras.utils.to\_categorical(Y2, num\_classes)*

*model2 =clf.fit(X1, Y1, validation\_data=(X2, Y2),epochs=20, verbose=1)*

results are as below , we dont see much of change from the above model1 results. The accuracy on test data is 0.962



here as well i have saved the trained model as below and loading script is also provided in the same.

Task2\_model2.json

Task2\_model2.h5

Part 3

Compare a “vanilla” model with a model using drop-out and here we have used traditionaltrain and test datasets

Vanila model is nothing but default model.

Here we only have used “relu” activation function in all layers

*# vanila model*

*def make\_model(optimizer='adam', hidden\_size=32):*

*model = Sequential([*

*Dense(hidden\_size, input\_shape=[784,]),*

*Activation('relu'),*

*Dense(hidden\_size),*

*Activation('relu'),*

*Dense(10),*

*Activation('softmax')])*

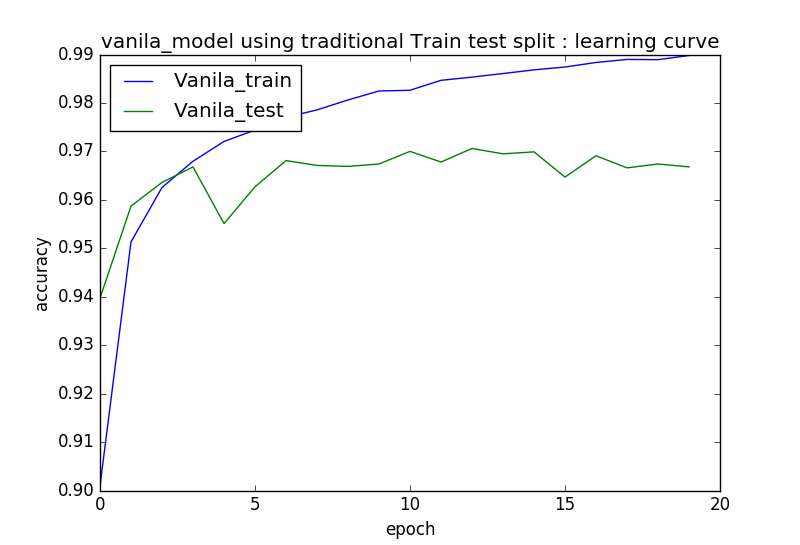
*model.compile(optimizer=optimizer, loss = 'categorical\_crossentropy', metrics=['accuracy'])*

*return model*

*clf = KerasClassifier(make\_model)*

*vanila\_model=clf.fit(X\_train, y\_train, validation\_data=(X\_test, y\_test),epochs=20, verbose=1)*

Results for vanila model as below.



Here also we dont see significant improvement in accuracy for test data set post 5 epoch.

Accuracy on test data at epoch 5 is 0.964

model saved for future porpose

Task2\_model31.json

Task2\_model31.h5

loading script is also provided.

# drop out model

Drop out is method regularization in neural networks, which randomly drops nurons from in respective hidden layers with proportion as specified.

We have applied droput layer at hidden layer and output layer with proportion equl to 0.2

*def make\_model(optimizer='adam', hidden\_size=32, dropout=0.2):*

*model = Sequential([*

*Dense(hidden\_size, input\_shape=[784,]),*

*Activation('relu'),*

*Dropout(rate = dropout),*

*Dense(hidden\_size),*

*Activation('relu'),*

*Dropout(rate = dropout),*

*Dense(10),*

*Activation('softmax')*

*])*

*model.compile(optimizer=optimizer, loss = 'categorical\_crossentropy', metrics=['accuracy'])*

*return model*

*clf = KerasClassifier(make\_model)*

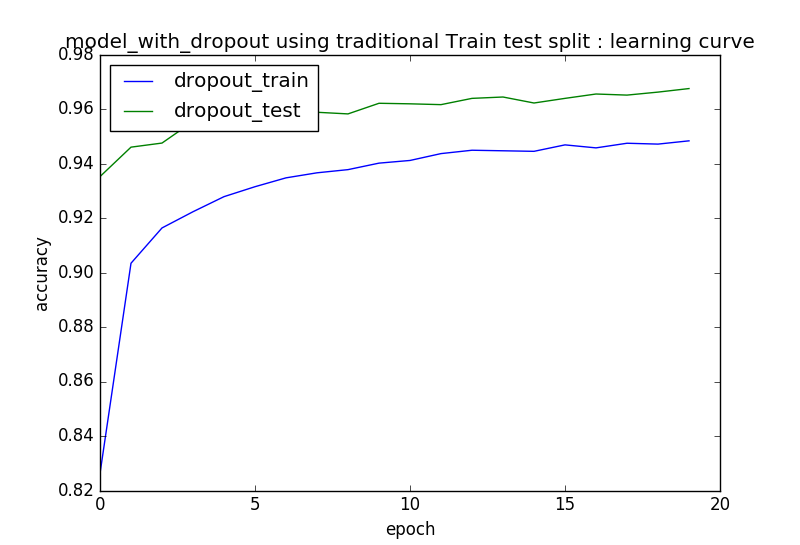
*model\_with\_dropout=clf.fit(X\_train, y\_train, validation\_data=(X\_test, y\_test),epochs=20, verbose=1)*

accuracy on test data is 0.953 which is bit lower than vanila model, this can be improved further after we twek the drop-out param using grid search.

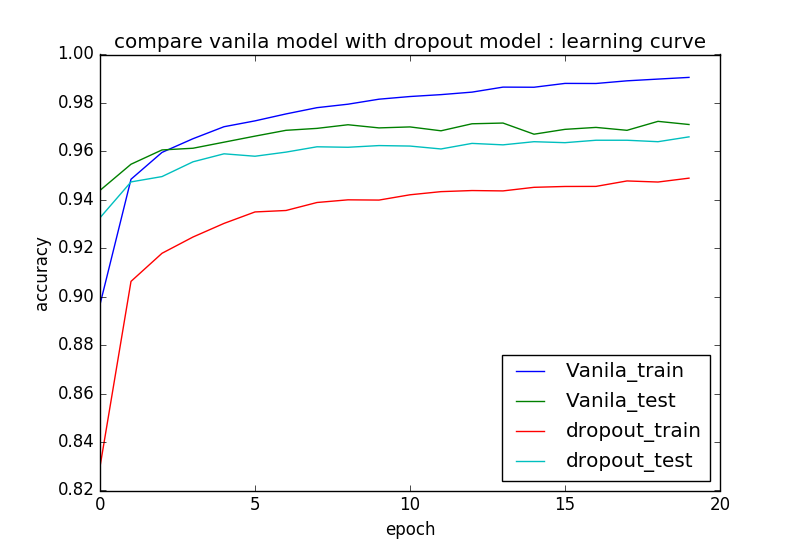
Model are saved for future porpose

Task2\_model32.json

Task2\_model32.h5



Comparing vanila model and drop-out model graphically

as we can see below the drop-out mode has perfomed quite simmilar with vanila model, tweking drop-out param may increase the accuracy further.