Steps to follow in ANN:

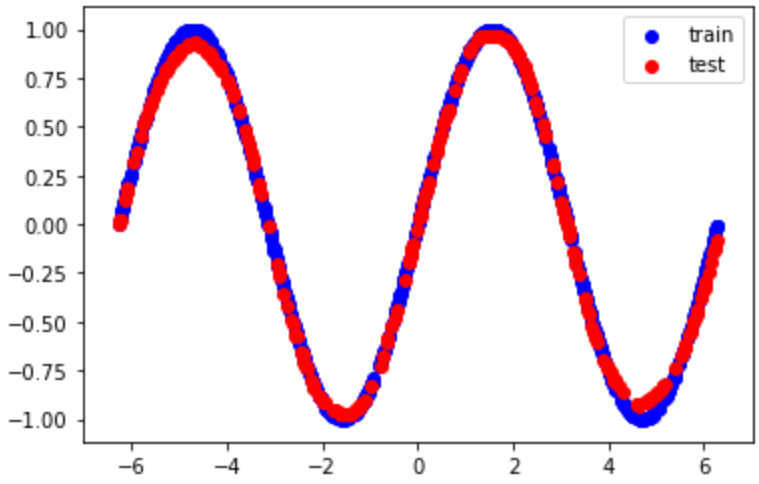
L_layer_model_minib() -- Main function

- initialize_parameters_deep_he(layers_dims) *-- initialise weights*
- initialize_adam(parameters)*-- initialise the optimiser*
- random_mini_batches(X, Y, mini_batch_size, seed)*--defines randomised batches.*
- *For i in mini batches*
    - L_model_forward(X,parameters,activation*-- computes the forward pass*
        - *Loop over the layers and do forward prop on each*
            - linear_activation_forward(A_prev, W, b, activation)*--compute the linear forward and then apply activation*
                - linear_forward(A, W, b):
                - Z = W.dot(A) + b
                - A, activation_cache = activation(Z)
    - compute_cost(AL, Y,parameters,lambd,regularisation,cost_func='mse') *-- computes the cost*
        - Mse : cost=np.mean(np.square(AL-Y))*0.5
        - Log :cost = (1./m) * np.sum(-np.dot(Y,np.log(AL+epsilon).T) - np.dot(1-Y, np.log(1-AL+epsilon).T))
    - L_model_backward(AL, Y, caches,activation,regularisation,lambd,cost_func) *--perform backprop*
        - Mse : dAL=(AL-Y)
        - Log : dAL = - (np.divide(Y, AL) - np.divide(1 - Y, 1 - AL))
        - *Loop over the layers and do back prop on each*
            - linear_activation_backward(dA, cache,regularisation,lambd,activation)*--compute back prop from activation to the weights*
                - dZ = activation_backward(dA, activation_cache)
                - linear_backward(dZ, cache,regularisation,lambd)
                - db = 1./m * np.sum(dZ, axis = 1, keepdims = True)
                - dA_prev = np.dot(W.T,dZ)
    - update_parameters(parameters, grads, learning_rate*)--finally update all the weights in the network*
- cost_avg = cost_total / batches *--compute the avg costs over the batch*
- predicterr(valid_x,valid_y,parameters,lambd,activation=activation,regularisation='none',cost_func=cost_func) *--predict the validation cost*
- plt.plot(costs) *--plot the costs*

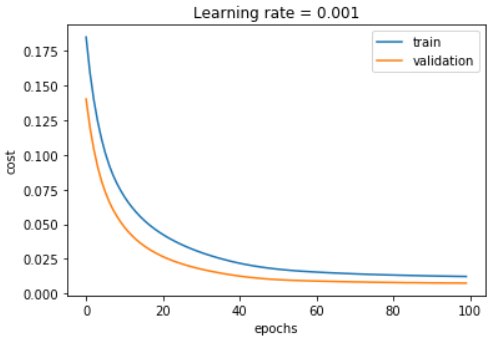
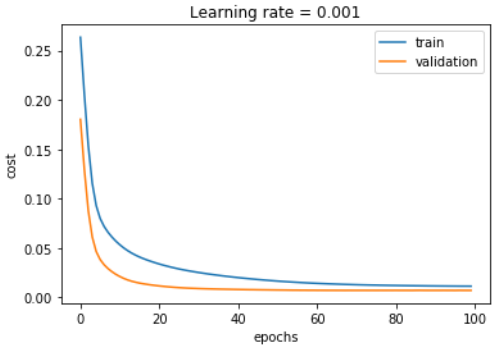
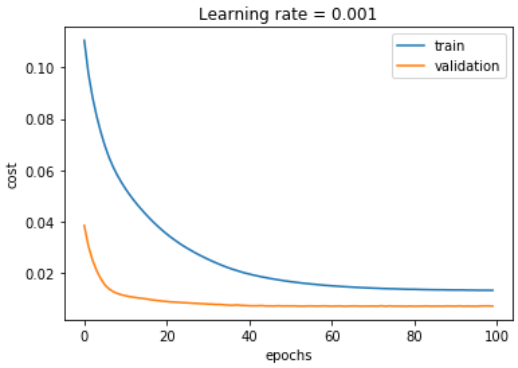
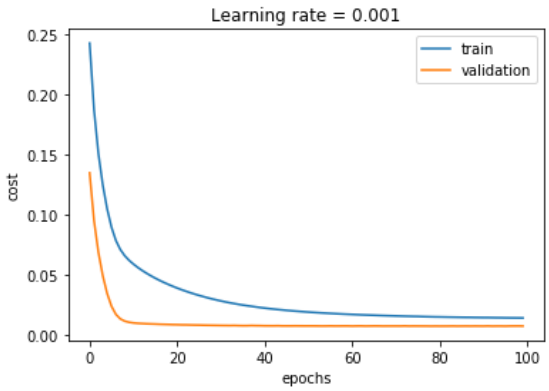
Best Model for Sin Curve

layers_dims = [1,20,20,1]

parameters = L_layer_model_minib(
train_x, train_y, layers_dims, valid=False,
 num_iterations = 600,
learning_rate = 0.001,
print_cost = True,
lambd=0.1,
optimizer="adam",
beta = 0.9,beta1 = 0.9, beta2 = 0.999,epsilon = 1e-8,
activation='tanh',
regularisation='none',
mini_batch_size=64,
cost_func="mse",
he_init=True)

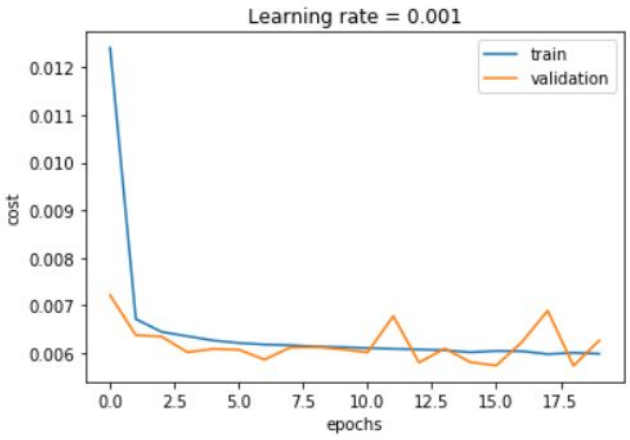
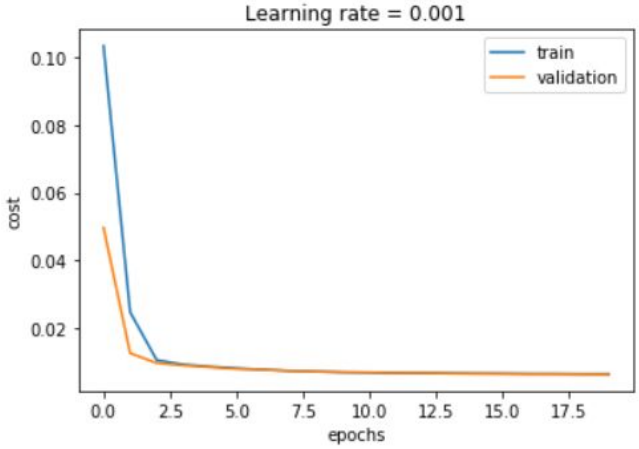


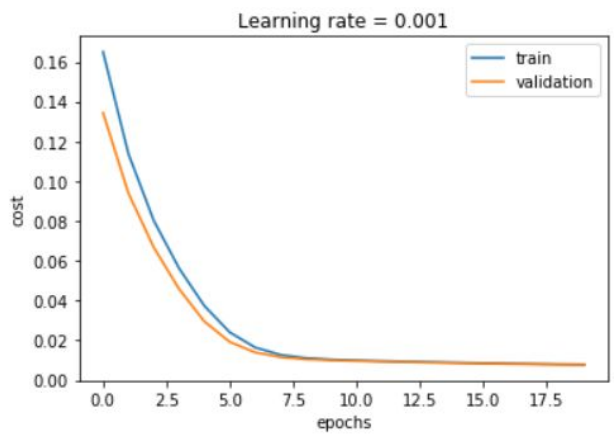
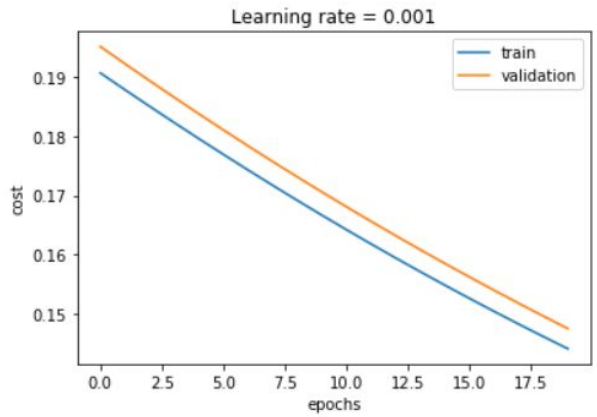
Architecture Comparisons

layers_dims = [4,5,1] Training mape : 0.7941098689193549 Valid mape : 0.8063094442632108 Testing mape : 0.7990991185251256	
layers_dims = [4,10,1] Training mape : 0.7740370114496019 Valid mape : 0.779183617465357 Testing mape : 0.7870296006952487	
layers_dims = [4,10,5,1] Training mape : 0.7692920370372286 Valid mape : 0.7763830442763537 Testing mape : 0.7821588242726298	
layers_dims = [4,10,10,1] Training mape : 0.7716003838887064 Valid mape : 0.779187773992918 Testing mape : 0.7839908249893479	

Batch Size Comparisons

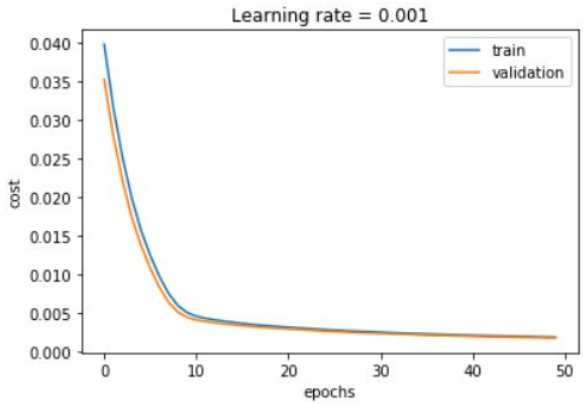
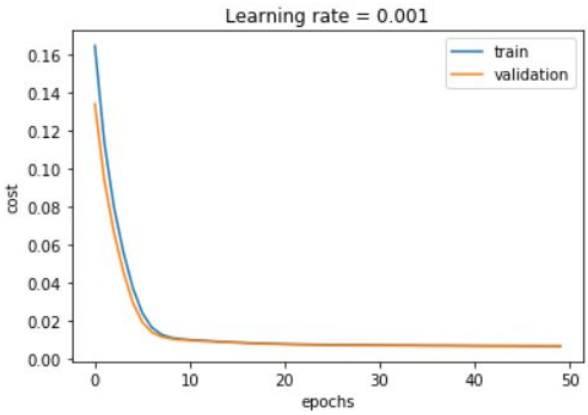
Freezing other parameters and changin batch sizes

layers_dims = [4,10,10,1] num_iterations = 20, learning_rate = 0.001 **mini_batch_size =1(SGD)** optimizer="adam" activation=Tanh he_init=True cost_func='mse' regularisation =’none’ Training mape : 0.6939322934163785 Valid mape : 0.7115727145429904 Testing mape : 0.7354052660571078	layers_dims = [4,10,10,1] num_iterations = 20, learning_rate = 0.001 **mini_batch_size =64** optimizer="adam" activation=Tanh he_init=True cost_func='mse' regularisation =’none’ Training mape : 0.7226041626498847 Valid mape : 0.7243184792599733 Testing mape : 0.7423342115034387
	

layers_dims = [4,10,10,1] num_iterations = 20, learning_rate = 0.001 **mini_batch_size =256** optimizer="adam" activation=Tanh he_init=True cost_func='mse' regularisation =’none’ Training mape : 0.787330891219502 Valid mape : 0.8000444029923501 Testing mape : 0.8021629604210103	layers_dims = [4,10,10,1] num_iterations = 20, learning_rate = 0.001 **mini_batch_size =6888(batch)** optimizer="adam" activation=Tanh he_init=True cost_func='mse' regularisation =’none’ Training mape : 3.600072792312424 Valid mape : 3.702112026264359 Testing mape : 3.6967978640657115
	

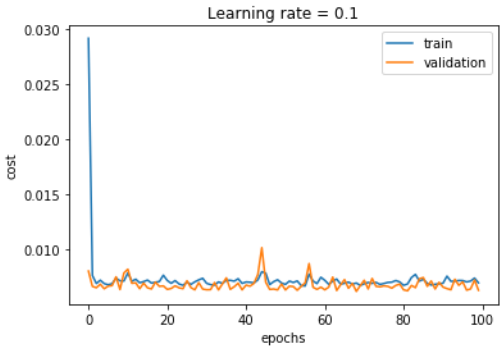
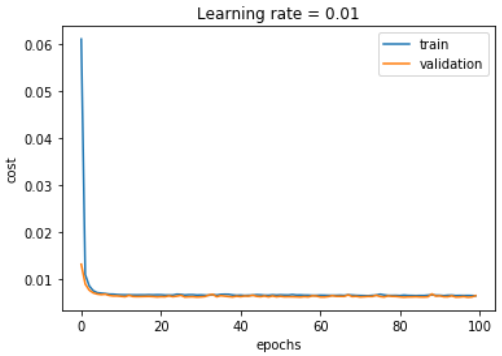
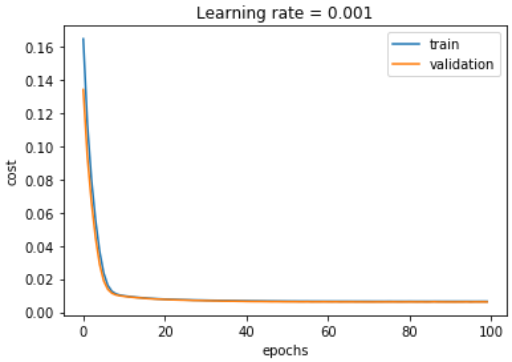
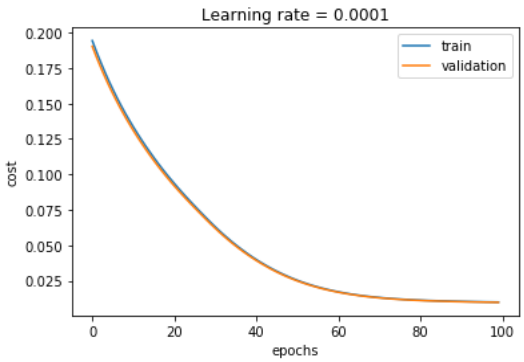
- As batch size increases learning becomes slow-*more iterations needed*
- As batch size increases learning per epoch becomes faster - *vectorisation comes into play*
- Even with sufficient epoch SGD tends to give slightly better results
- SGD graph for validation is usually a bit zig zag and not smooth(optimisers will help)

Output Activation Comparisons

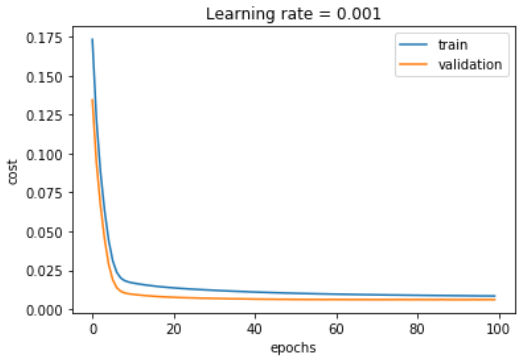
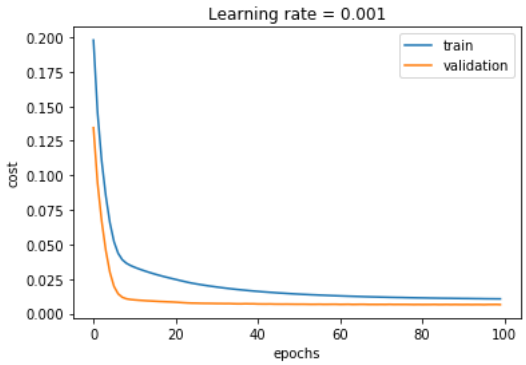
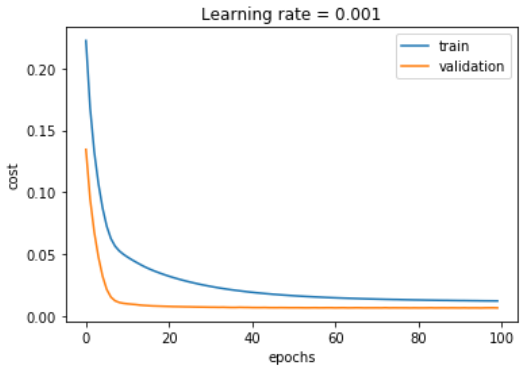
layers_dims = [4,10,10,1] num_iterations = 50, learning_rate = 0.001 mini_batch_size =256 optimizer="adam" **activation="sigmoid"** he_init=True cost_func='mse' regularisation =’none’ Training mape : 0.7726484973627554 Valid mape : 0.7767228155762528 Testing mape : 0.7946247880970604	layers_dims = [4,10,10,1] num_iterations = 50, learning_rate = 0.001 mini_batch_size =256 optimizer="adam" **activation="tanh"** he_init=True cost_func='mse' regularisation =’none’ Training mape : 0.7264987907448855 Valid mape : 0.732542065744155 Testing mape : 0.7536587432898809
	

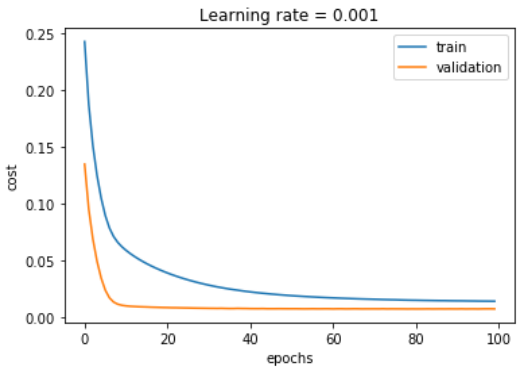
- Tanh converges faster

Learning Rate:

lr=0.1 Training mape : 0.7189152217350819 Valid mape : 0.7298132876871712 Testing mape : 0.7388300192620225	
lr=0.01 Training mape : 0.7169999218682349 Valid mape : 0.7318793588526947 Testing mape : 0.7451245775566842	
lr=0.001 Training mape : 0.7208036807521352 Valid mape : 0.7226891777193623 Testing mape : 0.7271498827770438	
lr=0.0001 Training mape : 0.9026285294068407 Valid mape : 0.920032185594116 Testing mape : 0.9188982131125064	

Regularisation

Lambd = 0.1 Training mape : 0.7208036807521352 Valid mape : 0.7226891777193623 Testing mape : 0.7271498827770438	
Lambd = 0.4 Training mape : 0.7367135963465501 Valid mape : 0.744504210476673 Testing mape : 0.7495242041036693	
Lambd = 0.7 Training mape : 0.7581294783825394 Valid mape : 0.766588173560549 Testing mape : 0.7707128884610976	

Lambd = 0.95 Training mape : 0.7716003838887064 Valid mape : 0.779187773992918 Testing mape : 0.7839908249893479	

Learning Rate

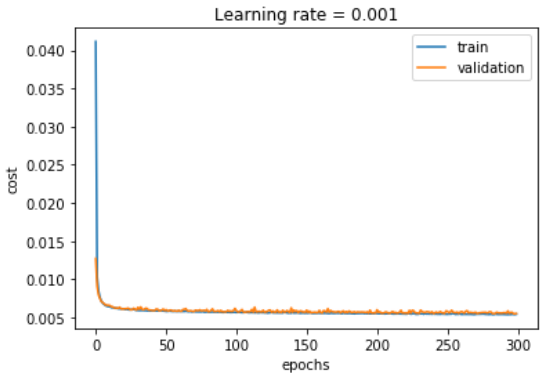
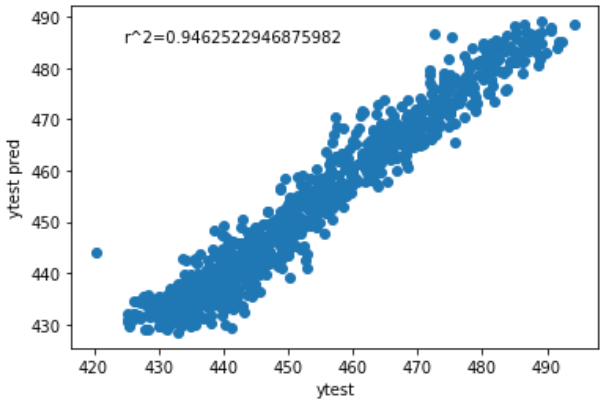
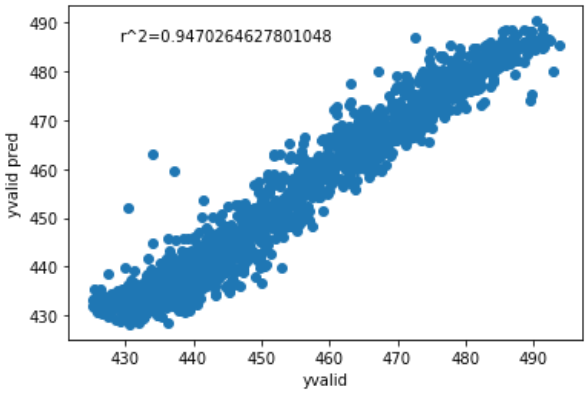
- As Learning rate decreases we can observer the model learns slower
- More iterations will be required
- Higher the Learning rate higher the chances to encounter and fall into local minimas as we can see in the first graph. The graph is more noisy when dropping.
- Lower the learning rate smoother the graph

Regularisation

- As we increase lambd we decrease the rate of learning during training
- Validation loss is maintained constantly
- Convergence for higher lambd requires more iterations as we can see in maps vals.

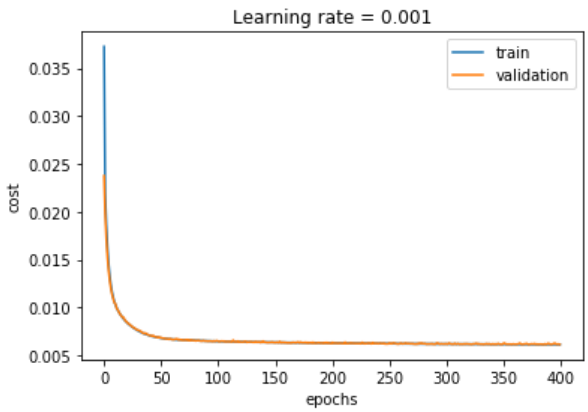
Best Model for CCPP Dataset:

layers_dims = [4,20,20,1]
parameters = L_layer_model_minib(
X_train, y_train,layers_dims,valid=True,valid_x=X_valid,valid_y=y_valid,
num_iterations = 300,
he_init=True,
mini_batch_size =64,
learning_rate = 0.001,
print_cost = True,
regularisation='none', lambd=0.1,
optimizer="adam",
beta = 0.9, beta1 = 0.9, beta2 = 0.999, epsilon = 1e-8,
activation='tanh',
cost_func='mse')

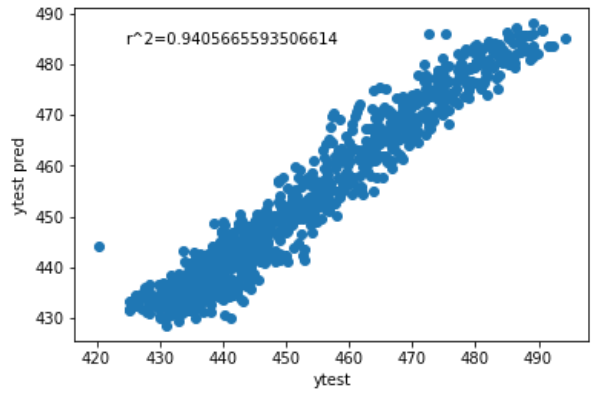
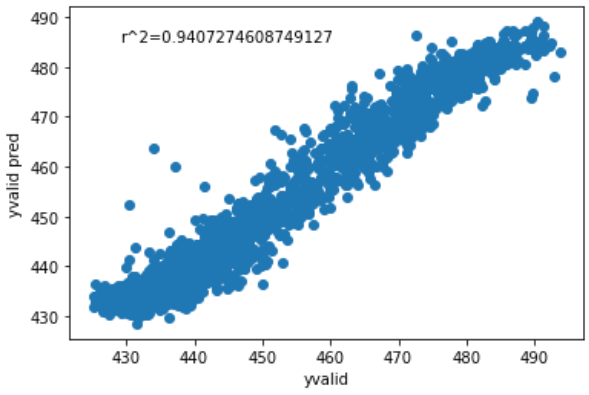
<table>
<tr>
<td>Training mape : 0.6552409400797429
Valid mape : 0.6688540464907443
Testing mape : 0.6783104901023306</td>
<td> 
</td>
</tr>
<tr>
<td> 
</td>
<td> 
</td>
</tr>
</table>

Bonus :

SGD with Momentum -

layers_dims = [4,10,10,1] num_iterations = 400, learning_rate = 0.001 mini_batch_size =1 optimizer="momentum" activation="tanh" he_init=True cost_func='mse' regularisation =’none’ Training mape : 0.7107204433769398 Valid mape : 0.7157541114288213 Testing mape : 0.7271898861662556	

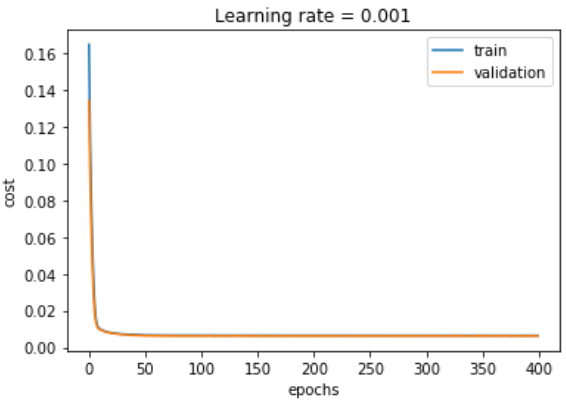
R2-

Adam with Batches-

layers_dims = [4,10,10,1]
num_iterations = 400,
learning_rate = 0.001
mini_batch_size =256
optimizer="adam"
activation="tanh"
he_init=True
cost_func='mse'
regularisation =’L2’

Training mape : 0.7144425261228652
Valid mape : 0.7197032291132449
Testing mape : 0.7249401718028659



R2-