run_framework.py

```
import data_loader_two_by_two as dat
import data_loader_IQN as dat_IQN
import framework.framework as framework
#import the directory_framework.filename_framework (becauase the directory is viewed as a package because it has __init__.py)
import framework.layer as layer
import framework.activation as activation
import framework.loss_funcs as loss_funcs


train_generator, eval_generator = dat.get_data_set()

# train_generator, eval_generator = dat_IQN.get_data_set()

sample=next(train_generator())#this is just to get the dimenstions of one batch
print('sample', sample)

#Find the number of input nodes. Each sample (training example) has shape sample.shape, which in this case is (2,2). The number
#of pixels (if this were an image) is shape[0] * shape[1]

#This is the total #Pixels, because its 2D

#next calculate the number of nodes of a simple NN with one input layer and one output layer with nothing in between
#we're gonna be building an autoencoder, which tries to replicate the input,therefore we have the same number of nodes as pixels and the sample for output and input layers

#DEFINE THE # NODES IN EACH HIDDEN LAYER
N_HIDDEN_NODES = [5, 3, 2]
#the number of layers of the model is just len(N_HIDDEN_NODES)
#define # input and output nodes for each LAYER
N_INPUT_NODES =  sample.shape[0] * sample.shape[1]
N_OUTPUT_NODES = sample.shape[0] * sample.shape[1]
#I think this is genera

N_NODES = [N_INPUT_NODES] +N_HIDDEN_NODES + [N_OUTPUT_NODES]

print("N_NODES = [N_INPUT_NODES] +[N_HIDDEN_NODES] + [N_OUTPUT_NODES]  = ", N_NODES)
#N_NODES=[N_IN_NODES, N_HIDDEN_NODES, N_OUT_NODES]
```

```
EXPECTED_VALUES_RANGE = (-3,4)#the min and max of the data itself
DESIRED_VALUES_RANGE = (-0.5,0.5)#what we want the range to be after scaline


#To do a 1-layer model, do MODEL=[layer.Dense( N_inputs=N_NODES[0],N_outputs
=N_NODES[1],activation=activation.tanh)]
MODEL=[]
#iterate through the list of nodes in each layer
for i_layer in range(len(N_NODES)-1):
    #e.g. 2 layers w one hidden in between has [N_IN_NODES, N_HIDDEN_NODES, N_OUT_NODES]
    MODEL.append(layer.Dense(
        N_inputs=N_NODES[i_layer],
        N_outputs=N_NODES[i_layer+1],
        #the N_output is either the hidden node number or the output node number
        activation=activation.tanh
    ))

print('MODEL=',MODEL)

#each of the arguments to ANN is a class!
autoencoder = framework.ANN(
    model=MODEL,
    expected_input_range=EXPECTED_VALUES_RANGE,
    loss_func=loss_funcs.quadratic_loss
    )#have to call model since its in __init__
    #here its training on the whole set then evaluating on the whole set
autoencoder.train(train_generator)
autoencoder.evaluate(eval_generator)
```