CS599 (Deep Learning)

Homework – 09

1. **Python Code:**

```
import torch
import pandas as pd
import matplotlib
import numpy as np
import math
matplotlib.use("agg")

from sklearn.model_selection import KFold, GridSearchCV, ParameterGrid
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import LassoCV
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from collections import Counter


data_set_dict = {"forest_fires": ("forestfires.csv", ",", True),
          "air_foil": ("airfoil_self_noise.tsv", "\t", False)}
data_dict = {}
hist_df_list = []


for data_name, (file_name, sep, log_trans) in data_set_dict.items():
  data_df = pd.read_csv(file_name, sep = sep, header = 0)
  data_nrow, data_ncol = data_df.shape
  label_col_num = data_ncol - 1
  data_label_vec = data_df.iloc[:, label_col_num]
  if log_trans:
    data_label_vec = np.log(data_label_vec + 1)
  label_sd = math.sqrt(data_label_vec.var())
  standard_label_vec = (
    data_label_vec - data_label_vec.mean()
  )/ label_sd
  is_feature_col = (
          np.arange(data_ncol) != label_col_num
          ) & (
            data_df.dtypes != "object"
          )
  data_features = data_df.loc[:, is_feature_col]
  feature_nrow, feature_ncol = data_features.shape
```

```
    feature_mean = data_features.mean().to_numpy().reshape(1, feature_ncol)
    feature_std = data_features.std().to_numpy().reshape(1, feature_ncol)
    feature_scaled = (data_features - feature_mean)/feature_std
    print("%s %s" %(data_name, data_features.shape))
    input_tensor = torch.from_numpy(
        feature_scaled.to_numpy()
    ).float()
    output_tensor = torch.from_numpy(
        standard_label_vec.to_numpy()
    ).float().reshape(data_nrow, 1)
    data_dict[data_name] = (feature_scaled, standard_label_vec)
    hist_df_list.append(pd.DataFrame({
                "data_name": data_name,
                "label": standard_label_vec})
                )
hist_df = pd.concat(hist_df_list)

class TorchModel(torch.nn.Module):
    def __init__(self, units_per_layer):
        super(TorchModel, self).__init__()
        seq_args = []
        second_to_last = len(units_per_layer)-1
        for layer_i in range(second_to_last):
            next_i = layer_i+1
            layer_units = units_per_layer[layer_i]
            next_units = units_per_layer[next_i]
            seq_args.append(torch.nn.Linear(layer_units, next_units))
            if layer_i < second_to_last-1:
                seq_args.append(torch.nn.ReLU())
        self.stack = torch.nn.Sequential(*seq_args)
    def forward(self, features):
        return self.stack(features)

class CSV(torch.utils.data.Dataset):
    def __init__(self, features, labels):
        self.features = features
        self.labels = labels
    def __getitem__(self, item):
        return self.features[item,:], self.labels[item]
    def __len__(self):
        return len(self.labels)

class TorchLearner:
    def __init__(
            self, units_per_layer, step_size=0.1,
            batch_size=20, max_epochs=50):
        self.max_epochs = max_epochs
        self.batch_size=batch_size
        self.model = TorchModel(units_per_layer)
        self.loss_fun = torch.nn.MSELoss()
        self.optimizer = torch.optim.SGD(
            self.model.parameters(), lr=step_size)
    def fit(self, split_data_dict):
        ds = CSV(
            split_data_dict["subtrain"]["X"],
            split_data_dict["subtrain"]["y"])
        dl = torch.utils.data.DataLoader(
            ds, batch_size=self.batch_size, shuffle=True)
        train_df_list = []
        for epoch_number in range(self.max_epochs):
            #print(epoch_number)
            for batch_features, batch_labels in dl:
                self.optimizer.zero_grad()
                loss_value = self.loss_fun(
                    self.model(batch_features), batch_labels)
                loss_value.backward()
                self.optimizer.step()
            for set_name, set_data in split_data_dict.items():
                pred_vec = self.model(set_data["X"])
                set_loss_value = self.loss_fun(pred_vec, set_data["y"])
                train_df_list.append(pd.DataFrame({
                    "set_name":[set_name],
                    "loss":float(set_loss_value),
                    "epoch":[epoch_number]
                }))
        self.train_df = pd.concat(train_df_list)
    def decision_function(self, test_features):
        with torch.no_grad():
            pred_vec = self.model(test_features)
        return pred_vec.numpy()

    def predict(self, test_features):
        pred_scores = self.decision_function(test_features)
        return pred_scores


class TorchLearnerCV:
    def __init__(self, n_folds, units_per_layer=[data_ncol,1]):
        self.units_per_layer = units_per_layer
        self.n_folds = n_folds
    def fit(self, train_features, train_labels):
        train_nrow, train_ncol = train_features.shape
        times_to_repeat=int(math.ceil(train_nrow/self.n_folds))
        fold_id_vec = np.tile(torch.arange(self.n_folds), times_to_repeat)[:train_nrow]
        np.random.shuffle(fold_id_vec)
        cv_data_list = []
        for validation_fold in range(self.n_folds):
            is_split = {
                "subtrain":fold_id_vec != validation_fold,
                "validation":fold_id_vec == validation_fold
                }
            split_data_dict = {}
            for set_name, is_set in is_split.items():
                set_y = train_labels[is_set]
                split_data_dict[set_name] = {
                    "X":train_features[is_set,:],
                    "y":set_y}
            learner = TorchLearner(self.units_per_layer)
            learner.fit(split_data_dict)
            cv_data_list.append(learner.train_df)
        self.cv_data = pd.concat(cv_data_list)
        self.train_df = self.cv_data.groupby(["set_name","epoch"]).mean().reset_index()
        #print(self.train_df)
        valid_df = self.train_df.query("set_name=='validation'")
        #print(valid_df)
        best_epochs = valid_df["loss"].argmin()
        self.min_df = valid_df.query("epoch==%s"%(best_epochs))
        print("Best Epoch: ", best_epochs)
        self.final_learner = TorchLearner(self.units_per_layer, max_epochs=(best_epochs + 1))
        self.final_learner.fit({"subtrain":{"X":train_features,"y":train_labels}})
        return self.cv_data
    def predict(self, test_features):
        return self.final_learner.predict(test_features)



test_loss_data_frames = []
loss_data_dict = {}
min_df_dict = {}
for data_name, (data_features, data_labels) in data_dict.items():
    kf = KFold(n_splits=3, shuffle=True, random_state=3)
    enum_obj = enumerate(kf.split(data_features))
    for fold_num, index_tup in enum_obj:
        zip_obj = zip(["train", "test"], index_tup)
        split_data = {}
        for set_name, set_indices in zip_obj:
            split_data[set_name] = (torch.from_numpy(data_features.iloc[set_indices,
:].to_numpy()).float(),
                                    torch.from_numpy(np.ravel(data_labels.iloc[set_indices])).float())
        #x = {data_name:X.shape for data_name, (X,y) in split_data.items()}
        #print(f"{data_name}: ", x)
        train_features, train_labels = split_data["train"]
        nrow, ncol = train_features.shape
        #print(f"{data_name}: ", nrow, ncol)
```

```
test_features, test_labels = split_data["test"]

#kneighbors
knn = KNeighborsRegressor()
hp_parameters = {"n_neighbors": list(range(1, 21))}
grid = GridSearchCV(knn, hp_parameters, cv=3)
grid.fit(train_features, train_labels)
best_n_neighbors = grid.best_params_['n_neighbors']
print("Best N-Neighbors = ", best_n_neighbors)
knn = KNeighborsRegressor(n_neighbors=best_n_neighbors)
knn.fit(train_features, train_labels)
knn_pred = knn.predict(test_features)
#print(test_labels)
#print(knn_pred)
#loss = mean_squared_error(test_labels, knn_pred)
#print(f"Knn Loss {data_name} : ", loss)

#linear model
pipe = make_pipeline(StandardScaler(), LassoCV(cv=3, max_iter=2000))
pipe.fit(train_features, train_labels)
lr_pred = pipe.predict(test_features)

#loss_linear = mean_squared_error(test_labels, lr_pred)
#print(f"Linear_loss {data_name} : ", loss_linear)

#Featureless
y_train_series = pd.Series(train_labels)
mean_train_label = y_train_series.mean()
print("Mean Train Label = ", mean_train_label)

# create a featureless baseline
featureless_pred = np.repeat(mean_train_label, len(test_features))
#featureless_loss = mean_squared_error(test_labels, featureless_pred)
#print(f"Featureless Loss {data_name} : ", featureless_loss)

#TorchLearnerCV
linear_learner = TorchLearnerCV(3, [ncol, 1])
#print("ncol:", ncol)
linear_loss = linear_learner.fit(train_features, train_labels.reshape(nrow,1))
ll_pred = linear_learner.predict(test_features)
#print(ll_pred)
#loss_torchlinear = mean_squared_error(test_labels, ll_pred)
#print(f"Torch Linear_loss {data_name} : ", loss_torchlinear)

#TorchLearnerCV + Deep
deep_learner = TorchLearnerCV(3, [ncol, 100, 10, 1])
deep_loss = deep_learner.fit(train_features, train_labels.reshape(nrow, 1))
dl_pred = deep_learner.predict(test_features)
```

```
    #loss_deeplearner = mean_squared_error(test_labels, dl_pred)
    #print(f"Torch Deep_loss {data_name} : ", loss_deeplearner)

    linear_loss = linear_loss.groupby(['set_name', 'epoch']).mean().reset_index()
    deep_loss = deep_loss.groupby(['set_name', 'epoch']).mean().reset_index()

    valid_df = linear_loss.query("set_name=='validation'")
    index_min = valid_df["loss"].argmin()
    min_df = valid_df.query("epoch==%s" % index_min)

    valid_df_deep = deep_loss.query("set_name=='validation'")
    index_min_deep = valid_df_deep["loss"].argmin()
    min_df_deep = valid_df_deep.query("epoch==%s" % index_min_deep)

    min_df_dict[data_name] = {'min_df linear': min_df,
                 'min_df deep': min_df_deep}

    loss_data_dict[data_name] = {'TorchLearnerCV Linear': linear_loss,
          'TorchLearnerCV Deep': deep_loss}

    # store predict data in dict
    pred_dict = {'KNeighborsRegressor + GridSearchCV': knn_pred,
          'LassoCV': lr_pred,
          'TorchLearnerCV Linear': ll_pred,
          'TorchLearnerCV Deep': dl_pred,
          'featureless': featureless_pred}
    test_square_loss = {}
    for algorithm, predictions in pred_dict.items():
      #print(f"{algorithm}:", predictions.shape)
      test_loss = mean_squared_error(test_labels, predictions)
      #accuracy = np.mean(test_labels == predictions)
      test_square_loss[algorithm] = test_loss

    for algorithm, test_loss in test_square_loss.items():
      print(f"{algorithm} Test Square Loss: {test_loss}")
      test_loss_df = pd.DataFrame({
        "data_set": [data_name],
        "fold_id": [fold_num],
        "algorithm": [algorithm],
        "test square loss": [test_square_loss[algorithm]]})
      test_loss_data_frames.append(test_loss_df)
    print(f"****************************End of
{data_name}({fold_num})*****************************")



total_test_loss_df = pd.concat(test_loss_data_frames, ignore_index = True)
print(total_test_loss_df)
```

```
import plotnine as p9
gg = p9.ggplot(total_test_loss_df, p9.aes(x ='test square loss', y = 'algorithm'))+\
       p9.facet_grid('.~data_set') + p9.geom_point()

gg.save("Test_square_loss.png", height = 8, width = 12)


forest_fires_loss = loss_data_dict["forest_fires"]
air_foil_loss = loss_data_dict["air_foil"]
forest_fires_min = min_df_dict["forest_fires"]
air_foil_min = min_df_dict["air_foil"]

gg1 = p9.ggplot() + p9.geom_line(p9.aes(x ='epoch', y = 'loss', color = 'set_name'), data =
forest_fires_loss["TorchLearnerCV Linear"])\
  + p9.geom_point(p9.aes(x ='epoch', y = 'loss', color = 'set_name'), data =
forest_fires_min["min_df linear"])  #p9.geom_line(p9.aes(fill = 'setname'))

gg1.save("Torch_validation_graph1.png", height = 8, width = 12)

gg2 = p9.ggplot() + p9.geom_line(p9.aes(x ='epoch', y = 'loss', color = 'set_name'), data =
forest_fires_loss["TorchLearnerCV Deep"])\
  + p9.geom_point(p9.aes(x ='epoch', y = 'loss', color = 'set_name'), data =
forest_fires_min["min_df deep"])  #p9.geom_line(p9.aes(fill = 'setname'))

gg2.save("Torch_validation_graph2.png", height = 8, width = 12)

gg3 = p9.ggplot() + p9.geom_line(p9.aes(x ='epoch', y = 'loss', color = 'set_name'), data =
air_foil_loss["TorchLearnerCV Linear"])\
  + p9.geom_point(p9.aes(x ='epoch', y = 'loss', color = 'set_name'), data =
air_foil_min["min_df linear"])  #p9.geom_line(p9.aes(fill = 'setname'))

gg3.save("Torch_validation_graph3.png", height = 8, width = 12)

gg4 = p9.ggplot() + p9.geom_line(p9.aes(x ='epoch', y = 'loss', color = 'set_name'), data =
air_foil_loss["TorchLearnerCV Deep"])\
  + p9.geom_point(p9.aes(x ='epoch', y = 'loss', color = 'set_name'), data =
air_foil_min["min_df deep"])  #p9.geom_line(p9.aes(fill = 'setname'))

gg4.save("Torch_validation_graph4.png", height = 8, width = 12)
```

2. **Output:**

```
>>> for data_name, (data_features, data_labels) in data_dict.items():
...     kf = KFold(n_splits=3, shuffle=True, random_state=3)
...     enum_obj = enumerate(kf.split(data_features))
...     for fold_num, index_tup in enum_obj:
```

```
...       zip_obj = zip(["train", "test"], index_tup)
...       split_data = {}
...       for set_name, set_indices in zip_obj:
...          split_data[set_name] = (torch.from_numpy(data_features.iloc[set_indices,
:].to_numpy()).float(),
...                                torch.from_numpy(np.ravel(data_labels.iloc[set_indices])).float())
...       #x = {data_name:X.shape for data_name, (X,y) in split_data.items()}
... ...
...
...       for algorithm, test_loss in test_square_loss.items():
...          print(f"{algorithm} Test Square Loss: {test_loss}")
...          test_loss_df = pd.DataFrame({
...             "data_set": [data_name],
...             "fold_id": [fold_num],
...             "algorithm": [algorithm],
...             "test square loss": [test_square_loss[algorithm]]})
...          test_loss_data_frames.append(test_loss_df)
...       print(f"***************************End of
{data_name}({fold_num})******************************")
```

```
Best N-Neighbors =  19
Mean Train Label =  0.0049645514
Best Epoch:  0
Best Epoch:  2
KNeighborsRegressor + GridSearchCV Test Square Loss: 1.0609387159347534
LassoCV Test Square Loss: 1.0739297224352429
TorchLearnerCV Linear Test Square Loss: 1.2798919677734375
TorchLearnerCV Deep Test Square Loss: 1.073811411857605
featureless Test Square Loss: 1.0739296674728394
***************************End of forest_fires(0)******************************
Best N-Neighbors =  20
Mean Train Label =  -0.019172536
Best Epoch:  0
Best Epoch:  3
KNeighborsRegressor + GridSearchCV Test Square Loss: 0.988243818283081
LassoCV Test Square Loss: 0.9922808202492903
TorchLearnerCV Linear Test Square Loss: 1.0052711963653564
TorchLearnerCV Deep Test Square Loss: 0.989805281162262
featureless Test Square Loss: 0.9922808408737183
***************************End of forest_fires(1)******************************
Best N-Neighbors =  17
Mean Train Label =  0.014222353
Best Epoch:  10
Best Epoch:  0
KNeighborsRegressor + GridSearchCV Test Square Loss: 1.0026485919952393
LassoCV Test Square Loss: 0.9305249905775753
TorchLearnerCV Linear Test Square Loss: 1.1973176002502441
TorchLearnerCV Deep Test Square Loss: 0.9264524579048157
```

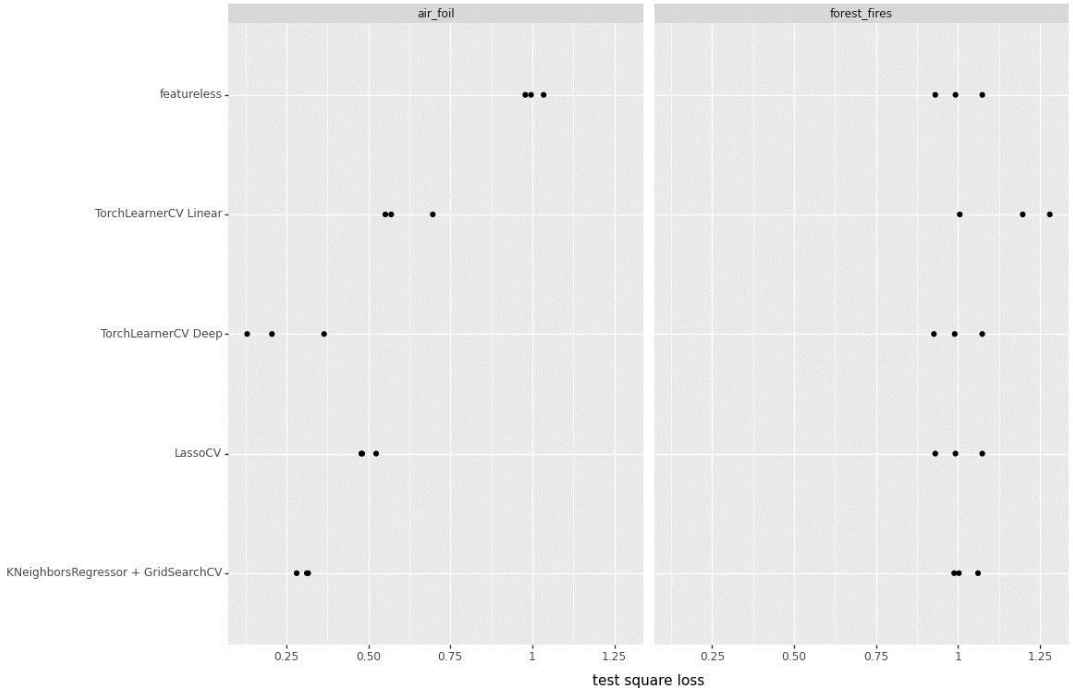
```
featureless Test Square Loss: 0.9305248856544495
***************************End of forest_fires(2)****************************
Best N-Neighbors =  13
Mean Train Label =  -0.034039237
Best Epoch:  42
Best Epoch:  44
KNeighborsRegressor + GridSearchCV Test Square Loss: 0.31644493341445923
LassoCV Test Square Loss: 0.47843713543411764
TorchLearnerCV Linear Test Square Loss: 0.5696210265159607
TorchLearnerCV Deep Test Square Loss: 0.1302846521139145
featureless Test Square Loss: 0.9787271022796631
***************************End of air_foil(0)****************************
Best N-Neighbors =  9
Mean Train Label =  0.00056366913
Best Epoch:  44
Best Epoch:  49
KNeighborsRegressor + GridSearchCV Test Square Loss: 0.2813766300678253
LassoCV Test Square Loss: 0.48135759164756325
TorchLearnerCV Linear Test Square Loss: 0.6965237259864807
TorchLearnerCV Deep Test Square Loss: 0.2059951275587082
featureless Test Square Loss: 0.9960943460464478
***************************End of air_foil(1)****************************
Best N-Neighbors =  11
Mean Train Label =  0.033475567
Best Epoch:  42
Best Epoch:  45
KNeighborsRegressor + GridSearchCV Test Square Loss: 0.3124180734157562
LassoCV Test Square Loss: 0.5237005567324898
TorchLearnerCV Linear Test Square Loss: 0.5518736839294434
TorchLearnerCV Deep Test Square Loss: 0.3650369644165039
featureless Test Square Loss: 1.0345804691314697
***************************End of air_foil(2)****************************
```

```
>>> total_test_loss_df = pd.concat(test_loss_data_frames, ignore_index = True)

>>> print(total_test_loss_df)

     data_set    fold_id                 algorithm                   test square loss
0  forest_fires     0      KNeighborsRegressor + GridSearchCV        1.060939
1  forest_fires     0                LassoCV                          1.073930
2  forest_fires     0      TorchLearnerCV Linear                     1.279892
3  forest_fires     0      TorchLearnerCV Deep                       1.073811
4  forest_fires     0                featureless                      1.073930
5  forest_fires     1      KNeighborsRegressor + GridSearchCV        0.988244
6  forest_fires     1                LassoCV                          0.992281
7  forest_fires     1      TorchLearnerCV Linear                     1.005271
8  forest_fires     1      TorchLearnerCV Deep                       0.989805
9  forest_fires     1                featureless                      0.992281
```

```
10  forest_fires     2     KNeighborsRegressor + GridSearchCV     1.002649
11  forest_fires     2                LassoCV                      0.930525
12  forest_fires     2     TorchLearnerCV Linear                  1.197318
13  forest_fires     2     TorchLearnerCV Deep                    0.926452
14  forest_fires     2                featureless                  0.930525
15     air_foil      0     KNeighborsRegressor + GridSearchCV     0.316445
16     air_foil      0                LassoCV                      0.478437
17     air_foil      0     TorchLearnerCV Linear                  0.569621
18     air_foil      0     TorchLearnerCV Deep                    0.130285
19     air_foil      0                featureless                  0.978727
20     air_foil      1     KNeighborsRegressor + GridSearchCV     0.281377
21     air_foil      1                LassoCV                      0.481358
22     air_foil      1     TorchLearnerCV Linear                  0.696524
23     air_foil      1     TorchLearnerCV Deep                    0.205995
24     air_foil      1                featureless                  0.996094
25     air_foil      2     KNeighborsRegressor + GridSearchCV      0.312418
26     air_foil      2                LassoCV                      0.523701
27     air_foil      2     TorchLearnerCV Linear                 0.551874
28     air_foil      2     TorchLearnerCV Deep                   0.365037
29     air_foil      2                featureless                 1.034580
```

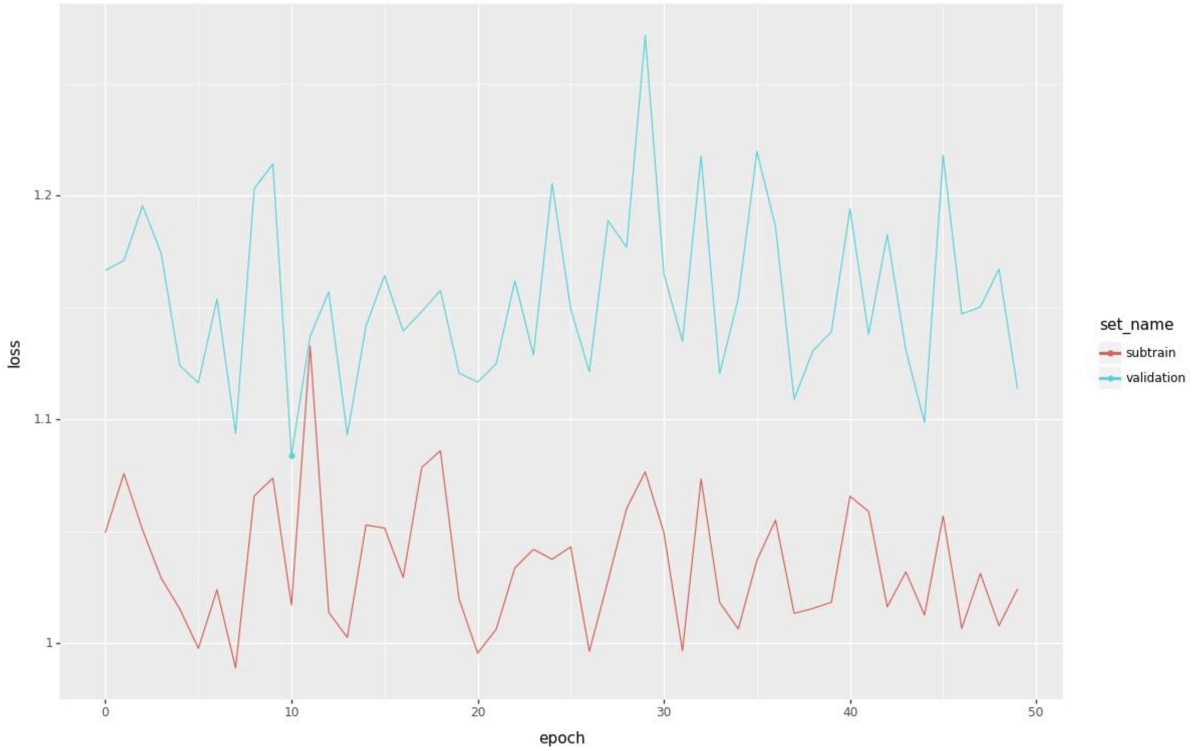
Test Loss Square Graph:

```
>>> gg = p9.ggplot(total_test_loss_df, p9.aes(x ='test square loss', y = 'algorithm'))+\
...        p9.facet_grid('.~data_set') + p9.geom_point()

>>> gg.save("Test_square_loss.png", height = 8, width = 12)
```



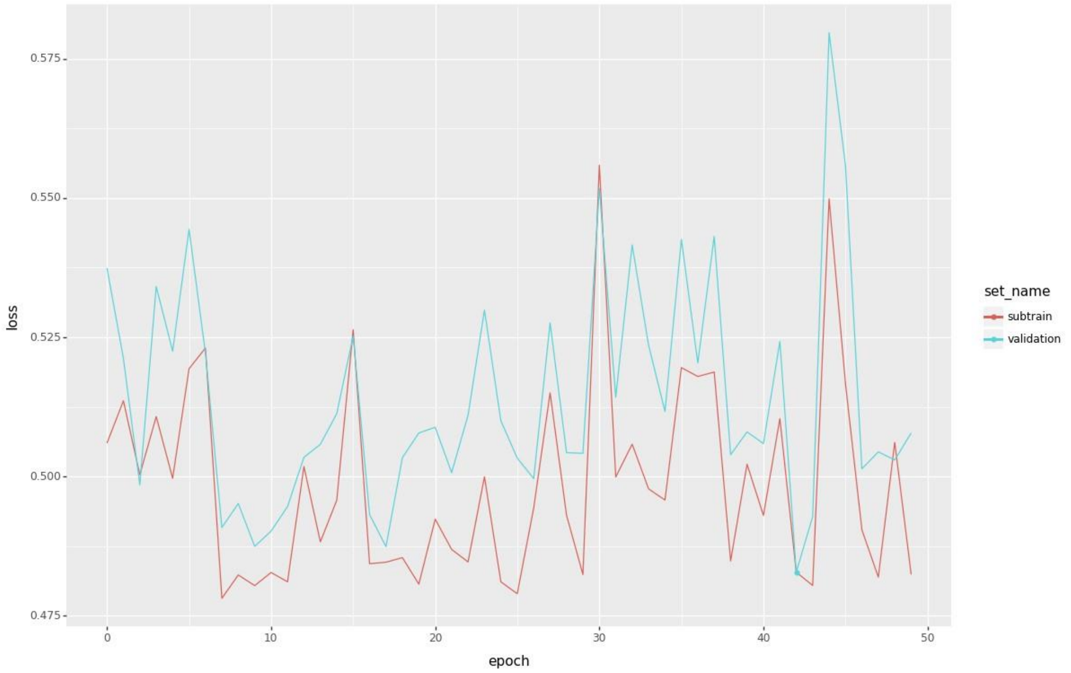
Linear subtrain/validation loss graph (Forest Fires):

```
>>> gg1 = p9.ggplot() + p9.geom_line(p9.aes(x ='epoch', y = 'loss', color = 'set_name'), data
= forest_fires_loss["TorchLearnerCV Linear"])\
...   + p9.geom_point(p9.aes(x ='epoch', y = 'loss', color = 'set_name'), data =
forest_fires_min["min_df linear"])  #p9.geom_line(p9.aes(fill = 'setname'))

>>> gg1.save("Torch_validation_graph1.png", height = 8, width = 12)
```



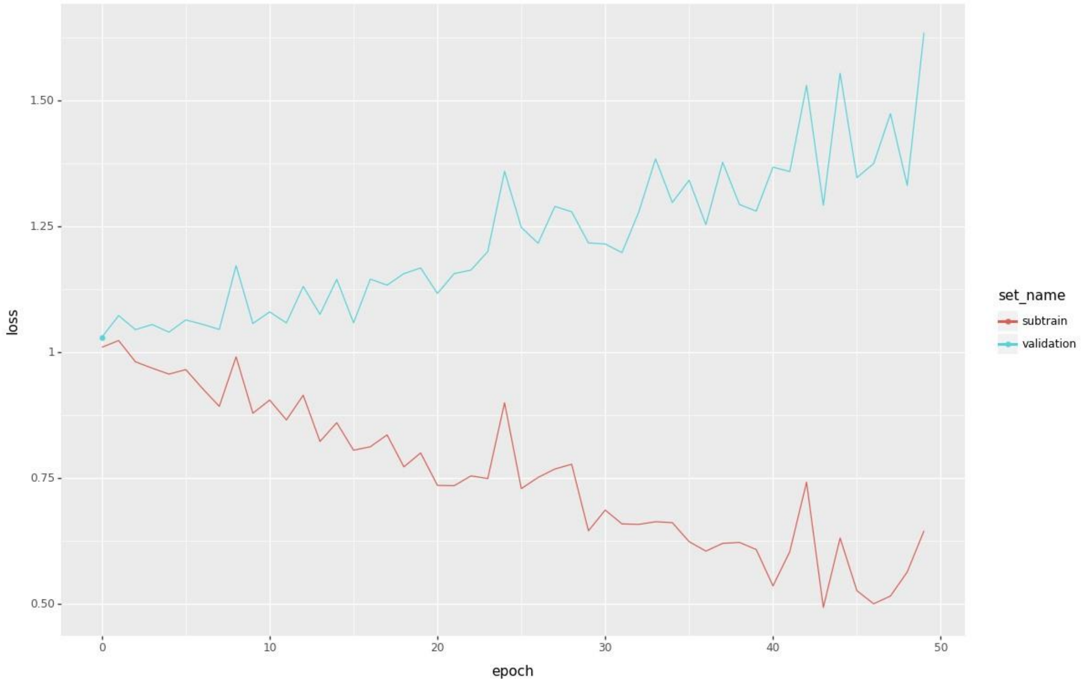
Linear subtrain/validation loss graph (Air Foil):

```
>>> gg3 = p9.ggplot() + p9.geom_line(p9.aes(x ='epoch', y = 'loss', color = 'set_name'), data
= air_foil_loss["TorchLearnerCV Linear"])\
...   + p9.geom_point(p9.aes(x ='epoch', y = 'loss', color = 'set_name'), data =
air_foil_min["min_df linear"])  #p9.geom_line(p9.aes(fill = 'setname'))

>>> gg3.save("Torch_validation_graph3.png", height = 8, width = 12)
```



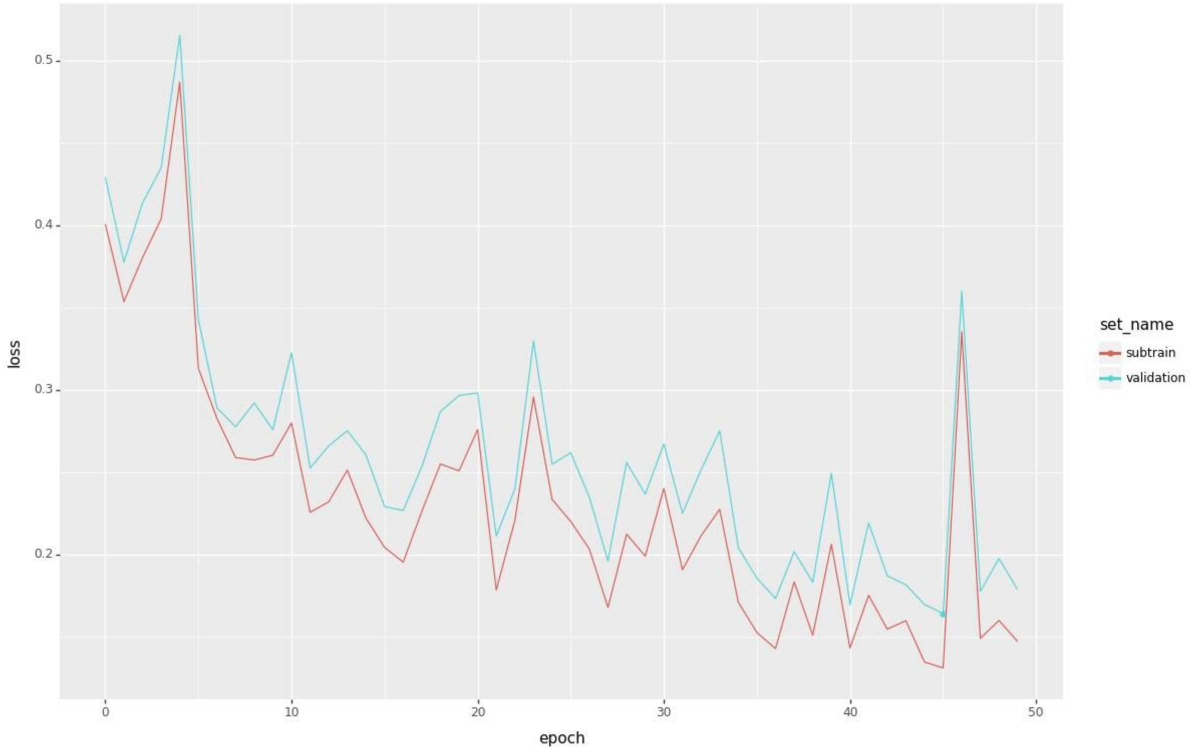
Deep subtrain/validation loss graph (Forest Fires):

```
>>> gg2 = p9.ggplot() + p9.geom_line(p9.aes(x ='epoch', y = 'loss', color = 'set_name'), data
= forest_fires_loss["TorchLearnerCV Deep"])\
...   + p9.geom_point(p9.aes(x ='epoch', y = 'loss', color = 'set_name'), data =
forest_fires_min["min_df deep"])  #p9.geom_line(p9.aes(fill = 'setname'))

>>> gg2.save("Torch_validation_graph2.png", height = 8, width = 12)
```



Deep subtrain/validation loss graph (Air Foil):

```
>>> gg4 = p9.ggplot() + p9.geom_line(p9.aes(x ='epoch', y = 'loss', color = 'set_name'), data
= air_foil_loss["TorchLearnerCV Deep"])\
...   + p9.geom_point(p9.aes(x ='epoch', y = 'loss', color = 'set_name'), data =
air_foil_min["min_df deep"])  #p9.geom_line(p9.aes(fill = 'setname'))

>>> gg4.save("Torch_validation_graph4.png", height = 8, width = 12)
```


loss
epoch
set_name
subtrain
validation