**CS 599 (Deep Learning)**

**Homework – 06**

1. **Python Code:**

import torch

import pandas as pd

import matplotlib

import numpy as np

matplotlib.use("agg")

from sklearn.model\_selection import KFold, GridSearchCV, ParameterGrid

from sklearn.neighbors import KNeighborsClassifier

from sklearn.linear\_model import LogisticRegressionCV

from sklearn.pipeline import make\_pipeline

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score

from collections import Counter

data\_set\_dict = {"zip": ("zip.test.gz", 0),

"spam": ("spam.data", 57)}

data\_dict = {}

for data\_name, (file\_name, label\_col\_num) in data\_set\_dict.items():

data\_df = pd.read\_csv(file\_name, sep=" ", header=None)

data\_label\_vec = data\_df.iloc[:, label\_col\_num]

is\_01 = data\_label\_vec.isin([0, 1])

data\_01\_df = data\_df.loc[is\_01, :]

is\_label\_col = data\_df.columns == label\_col\_num

data\_features = data\_01\_df.iloc[:, ~is\_label\_col]

data\_labels = data\_01\_df.iloc[:, is\_label\_col]

data\_dict[data\_name] = (data\_features, data\_labels)

spam\_features, spam\_labels = data\_dict.pop("spam")

spam\_nrow, spam\_ncol = spam\_features.shape

spam\_mean = spam\_features.mean().to\_numpy().reshape(1, spam\_ncol)

spam\_std = spam\_features.std().to\_numpy().reshape(1, spam\_ncol)

spam\_scaled = (spam\_features - spam\_mean)/spam\_std

data\_dict["spam\_scaled"] = (spam\_scaled, spam\_labels)

{data\_name:X.shape for data\_name, (X,y) in data\_dict.items()}

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:

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, max\_epochs, batch\_size, step\_size):

self.max\_epochs = max\_epochs

self.batch\_size = batch\_size

self.step\_size = step\_size

self.model = TorchModel(units\_per\_layer)

self.optimizer = torch.optim.SGD(self.model.parameters(), lr = self.step\_size)

self.loss\_fun = torch.nn.BCEWithLogitsLoss()

def take\_step(self, X, y):

self.optimizer.zero\_grad()

pred\_tensor = self.model(X)

loss\_value = self.loss\_fun(pred\_tensor, y)

loss\_value.backward()

self.optimizer.step()

return loss\_value.item()

def fit(self, X, y):

ds = CSV(X, y)

dl = torch.utils.data.DataLoader(

ds, batch\_size = self.batch\_size, shuffle = True)

for epoch in range(self.max\_epochs):

for batch\_features, batch\_labels in dl:

loss = self.take\_step(batch\_features, batch\_labels)

def decision\_function(self, X):

with torch.no\_grad():

pred\_vec = self.model(X)

return pred\_vec.numpy()

def predict(self, X):

pred\_scores = self.decision\_function(X)

return np.where(pred\_scores > 0, 1, 0)

class TorchLearnerCV:

def \_\_init\_\_(self, units\_per\_layer, max\_epochs, batch\_size, step\_size, n\_splits):

self.units\_per\_layer = units\_per\_layer

self.max\_epochs = max\_epochs

self.batch\_size = batch\_size

self.step\_size = step\_size

self.n\_splits = n\_splits

def fit(self, X, y):

kf = KFold(n\_splits = self.n\_splits, shuffle = True, random\_state = 5)

best\_loss = float('inf')

best\_model = None

best\_epochs = 0

valid\_loss = []

sub\_train\_loss = []

loss\_val = []

scores = pd.DataFrame(columns = ["validation\_fold", "setname", "loss\_value", "epoch"])

for max\_epoch in range(1, self.max\_epochs + 1):

loss\_values = []

for fold\_num, (train\_index, val\_index) in enumerate(kf.split(X)):

subtrain\_data = {"X": X[train\_index], "y": y[train\_index]}

val\_data = {"X": X[val\_index], "y": y[val\_index]}

learner = TorchLearner(self.units\_per\_layer, max\_epoch, self.batch\_size, self.step\_size)

learner.fit(subtrain\_data["X"], subtrain\_data["y"])

val\_loss = learner.take\_step(val\_data["X"], val\_data["y"])

subtrain\_loss = learner.take\_step(subtrain\_data["X"], subtrain\_data["y"])

loss\_values.append(val\_loss)

loss\_dict = {"validation": val\_loss, "subtrain": subtrain\_loss}

for setname, loss\_value in loss\_dict.items():

loss\_val\_row = pd.DataFrame({

"validation\_fold": [fold\_num],

"setname": [setname],

"loss\_value": [loss\_value],

"epoch" : [max\_epoch]})

loss\_val.append(loss\_val\_row)

loss\_df = pd.concat(loss\_val)

loss\_values\_mean = np.mean(loss\_values)

if loss\_values\_mean < best\_loss:

best\_loss = loss\_values\_mean

best\_model = learner

best\_epochs = max\_epoch

print("best\_epoch: ", best\_epochs)

self.best\_model = best\_model

self.best\_model.fit(X,y)

return loss\_df

def predict(self, X):

return self.best\_model.predict(X)

zip\_features, zip\_labels = data\_dict["zip"]

input\_tensor = torch.from\_numpy(zip\_features.to\_numpy()).float()

output\_tensor = torch.from\_numpy(zip\_labels.to\_numpy()).float()

accuracy\_data\_frames = []

loss\_data\_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))

data\_nrow, data\_ncol = data\_features.shape

for fold\_num, (train\_index, test\_index) in enum\_obj:

X\_train, X\_test = torch.from\_numpy(data\_features.iloc[train\_index].to\_numpy()).float(), torch.from\_numpy(data\_features.iloc[test\_index].to\_numpy()).float()

y\_train, y\_test = torch.from\_numpy(data\_labels.iloc[train\_index].to\_numpy()).float(), torch.from\_numpy(data\_labels.iloc[test\_index].to\_numpy()).float()

input\_tensor = torch.from\_numpy(data\_features.to\_numpy()).float()

output\_tensor = torch.from\_numpy(data\_labels.to\_numpy()).float()

# K-nearest neighbors

knn = KNeighborsClassifier()

hp\_parameters = {"n\_neighbors": list(range(1, 21))}

grid = GridSearchCV(knn, hp\_parameters, cv=3)

grid.fit(X\_train, y\_train.ravel())

best\_n\_neighbors = grid.best\_params\_['n\_neighbors']

print("Best N-Neighbors = ", best\_n\_neighbors)

knn = KNeighborsClassifier(n\_neighbors=best\_n\_neighbors)

knn.fit(X\_train, y\_train.ravel())

knn\_pred = knn.predict(X\_test)

# Logistic Regression

pipe = make\_pipeline(StandardScaler(), LogisticRegressionCV(cv=3, max\_iter=2000))

pipe.fit(X\_train, y\_train.ravel())

lr\_pred = pipe.predict(X\_test)

y\_train\_series = pd.Series(y\_train.ravel())

#TorchLearnerCV

torch\_learner = TorchLearnerCV([data\_ncol, 100, 1], 15, 100, 0.15, 3)

loss = torch\_learner.fit(X\_train, y\_train)

tl\_pred = torch\_learner.predict(X\_test)

#TorchLearnerCV + deep

torch\_learner\_deep = TorchLearnerCV([data\_ncol, 100, 10, 5, 1], 10, 100, 0.5, 3)

deep\_loss = torch\_learner\_deep.fit(X\_train, y\_train)

tl\_deep\_pred = torch\_learner\_deep.predict(X\_test)

most\_frequent\_class = y\_train\_series.value\_counts().idxmax()

print("Most Frequent Class = ", most\_frequent\_class)

# create a featureless baseline

featureless\_pred = np.full\_like(y\_test.ravel(), most\_frequent\_class)

# store predict data in dict

pred\_dict = {'gridSearch + nearest neighbors': knn\_pred,

'linear\_model': lr\_pred,

'TorchLearnerCV': tl\_pred,

'TorchLearnerCV + Deep': tl\_deep\_pred,

'featureless': featureless\_pred}

test\_accuracy = {}

loss\_data\_dict[data\_name] = {'TorchLearnerCV': loss,

'TorchLearnerCV + Deep': deep\_loss}

for algorithm, predictions in pred\_dict.items():

accuracy = accuracy\_score(y\_test, predictions)

test\_accuracy[algorithm] = accuracy

for algorithm, accuracy in test\_accuracy.items():

print(f"{algorithm} Test Accuracy: {accuracy \* 100}")

accuracy\_df = pd.DataFrame({

"data\_set": [data\_name],

"fold\_id": [fold\_num],

"algorithm": [algorithm],

"accuracy": [test\_accuracy[algorithm]]})

accuracy\_data\_frames.append(accuracy\_df)

print(f"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of {data\_name}({fold\_num})\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

total\_accuracy\_df = pd.concat(accuracy\_data\_frames, ignore\_index = True)

print(total\_accuracy\_df)

import plotnine as p9

gg = p9.ggplot(total\_accuracy\_df, p9.aes(x ='accuracy', y = 'algorithm'))+\

p9.facet\_grid('.~data\_set') + p9.geom\_point()

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

zip\_loss = loss\_data\_dict["zip"]

spam\_loss = loss\_data\_dict["spam\_scaled"]

gg1 = p9.ggplot(zip\_loss["TorchLearnerCV"], p9.aes(x ='epoch', y = 'loss\_value', fill = 'setname')) + p9.facet\_grid('.~validation\_fold') + p9.geom\_point() + p9.ggtitle("Zip Data(TorchLearner): Subtrain/Validation vs Epoch")

gg1.save("valid\_graph1.png", height = 8, width = 12)

gg2 = p9.ggplot(zip\_loss["TorchLearnerCV + Deep"], p9.aes(x ='epoch', y = 'loss\_value', fill = 'setname')) + p9.facet\_grid('.~validation\_fold') + p9.geom\_point() + p9.ggtitle("Zip Data(TorchLearner + Deep): Subtrain/Validation vs Epoch")

gg2.save("valid\_graph2.png", height = 8, width = 12)

gg3 = p9.ggplot(spam\_loss["TorchLearnerCV"], p9.aes(x ='epoch', y = 'loss\_value', fill = 'setname')) + p9.facet\_grid('.~validation\_fold') + p9.geom\_point() + p9.ggtitle("Spam\_scaled Data(TorchLearner): Subtrain/Validation vs Epoch")

gg3.save("valid\_graph3.png", height = 8, width = 12)

gg4 = p9.ggplot(spam\_loss["TorchLearnerCV + Deep"], p9.aes(x ='epoch', y = 'loss\_value', fill = 'setname')) \

+ p9.facet\_grid('.~validation\_fold') + p9.geom\_point() + p9.ggtitle("spam\_scaled Data(TorchLearner + Deep): Subtrain/Validation vs Epoch")

gg4.save("valid\_graph4.png", height = 8, width = 12)

1. **Outputs:**

**>>> 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))**

**... data\_nrow, data\_ncol = data\_features.shape**

**...**

**... for fold\_num, (train\_index, test\_index) in enum\_obj:**

**... X\_train, X\_test = torch.from\_numpy(data\_features.iloc[train\_index].to\_numpy()).float(), torch.from\_numpy(data\_features.iloc[test\_index].to\_numpy()).float()**

**... y\_train, y\_test = torch.from\_numpy(data\_labels.iloc[train\_index].to\_numpy()).float(), torch.from\_numpy(data\_labels.iloc[test\_index].to\_numpy()).float()**

**...**

**... input\_tensor = torch.from\_numpy(data\_features.to\_numpy()).float()**

**... ...**

**...**

**... for algorithm, accuracy in test\_accuracy.items():**

**... print(f"{algorithm} Test Accuracy: {accuracy \* 100}")**

**... accuracy\_df = pd.DataFrame({**

**... "data\_set": [data\_name],**

**... "fold\_id": [fold\_num],**

**... "algorithm": [algorithm],**

**... "accuracy": [test\_accuracy[algorithm]]})**

**... accuracy\_data\_frames.append(accuracy\_df)**

**... print(f"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of {data\_name}({fold\_num})\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")**

Best N-Neighbors = 1

best\_epoch: 11

best\_epoch: 9

Most Frequent Class = 0.0

gridSearch + nearest neighbors Test Accuracy: 100.0

linear\_model Test Accuracy: 99.51923076923077

TorchLearnerCV Test Accuracy: 98.5576923076923

TorchLearnerCV + Deep Test Accuracy: 99.03846153846155

featureless Test Accuracy: 58.65384615384615

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of zip(0)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Best N-Neighbors = 1

best\_epoch: 14

best\_epoch: 5

Most Frequent Class = 0.0

gridSearch + nearest neighbors Test Accuracy: 99.51923076923077

linear\_model Test Accuracy: 99.03846153846155

TorchLearnerCV Test Accuracy: 96.63461538461539

TorchLearnerCV + Deep Test Accuracy: 96.63461538461539

featureless Test Accuracy: 57.21153846153846

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of zip(1)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Best N-Neighbors = 4

best\_epoch: 14

best\_epoch: 7

Most Frequent Class = 0.0

gridSearch + nearest neighbors Test Accuracy: 99.03381642512076

linear\_model Test Accuracy: 99.03381642512076

TorchLearnerCV Test Accuracy: 97.58454106280193

TorchLearnerCV + Deep Test Accuracy: 98.55072463768117

featureless Test Accuracy: 57.00483091787439

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of zip(2)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Best N-Neighbors = 4

best\_epoch: 14

best\_epoch: 10

Most Frequent Class = 0.0

gridSearch + nearest neighbors Test Accuracy: 88.72229465449804

linear\_model Test Accuracy: 91.52542372881356

TorchLearnerCV Test Accuracy: 91.98174706649283

TorchLearnerCV + Deep Test Accuracy: 93.08996088657105

featureless Test Accuracy: 60.88657105606258

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of spam\_scaled(0)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Best N-Neighbors = 5

best\_epoch: 15

best\_epoch: 7

Most Frequent Class = 0.0

gridSearch + nearest neighbors Test Accuracy: 90.80834419817471

linear\_model Test Accuracy: 91.78617992177314

TorchLearnerCV Test Accuracy: 92.69882659713168

TorchLearnerCV + Deep Test Accuracy: 91.85136897001304

featureless Test Accuracy: 60.104302477183836

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of spam\_scaled(1)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Best N-Neighbors = 9

best\_epoch: 15

best\_epoch: 10

Most Frequent Class = 0.0

gridSearch + nearest neighbors Test Accuracy: 90.54142204827136

linear\_model Test Accuracy: 92.49836921069797

TorchLearnerCV Test Accuracy: 91.71559034572732

TorchLearnerCV + Deep Test Accuracy: 92.75929549902152

featureless Test Accuracy: 60.79582517938682

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of spam\_scaled(2)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**>>> total\_accuracy\_df = pd.concat(accuracy\_data\_frames, ignore\_index = True)**

**>>> print(total\_accuracy\_df)**

data\_set fold\_id algorithm accuracy

0 zip 0 gridSearch + nearest neighbors 1.000000

1 zip 0 linear\_model 0.995192

2 zip 0 TorchLearnerCV 0.985577

3 zip 0 TorchLearnerCV + Deep 0.990385

4 zip 0 featureless 0.586538

5 zip 1 gridSearch + nearest neighbors 0.995192

6 zip 1 linear\_model 0.990385

7 zip 1 TorchLearnerCV 0.966346

8 zip 1 TorchLearnerCV + Deep 0.966346

9 zip 1 featureless 0.572115

10 zip 2 gridSearch + nearest neighbors 0.990338

11 zip 2 linear\_model 0.990338

12 zip 2 TorchLearnerCV 0.975845

13 zip 2 TorchLearnerCV + Deep 0.985507

14 zip 2 featureless 0.570048

15 spam\_scaled 0 gridSearch + nearest neighbors 0.887223

16 spam\_scaled 0 linear\_model 0.915254

17 spam\_scaled 0 TorchLearnerCV 0.919817

18 spam\_scaled 0 TorchLearnerCV + Deep 0.930900

19 spam\_scaled 0 featureless 0.608866

20 spam\_scaled 1 gridSearch + nearest neighbors 0.908083

21 spam\_scaled 1 linear\_model 0.917862

22 spam\_scaled 1 TorchLearnerCV 0.926988

23 spam\_scaled 1 TorchLearnerCV + Deep 0.918514

24 spam\_scaled 1 featureless 0.601043

25 spam\_scaled 2 gridSearch + nearest neighbors 0.905414

26 spam\_scaled 2 linear\_model 0.924984

27 spam\_scaled 2 TorchLearnerCV 0.917156

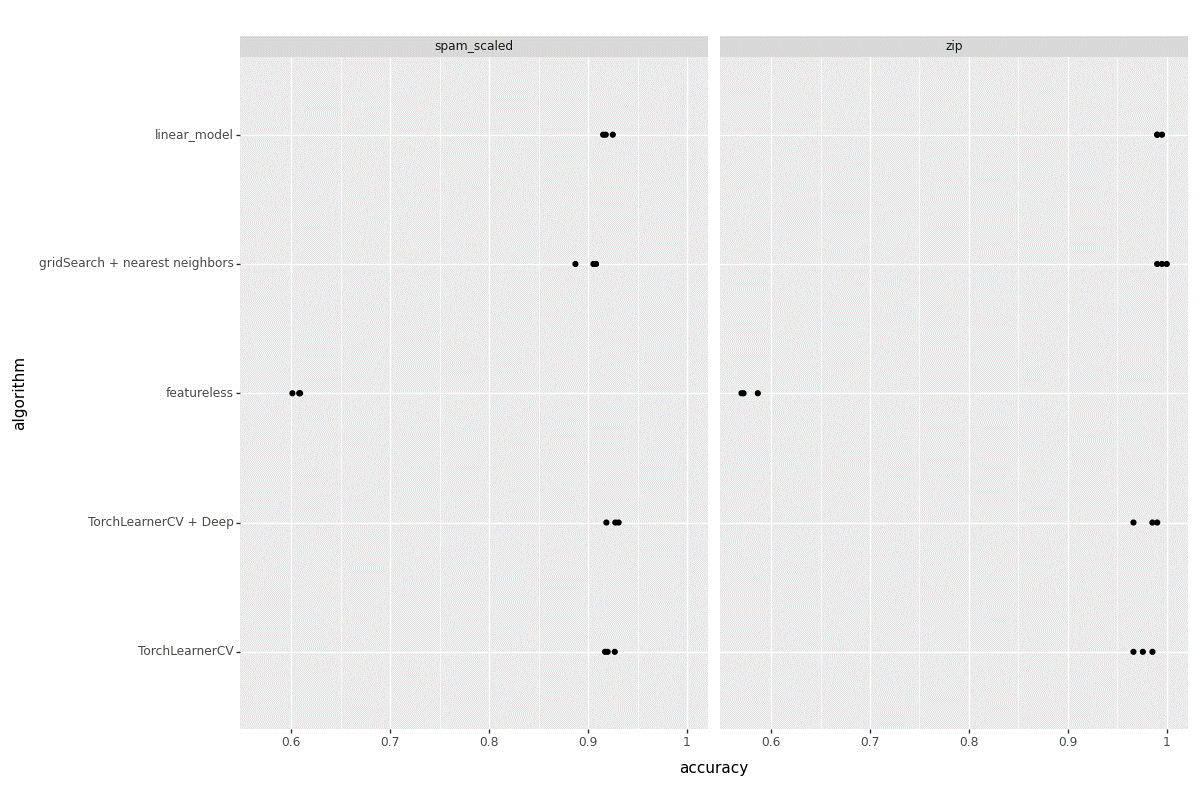
28 spam\_scaled 2 TorchLearnerCV + Deep 0.927593

29 spam\_scaled 2 featureless 0.607958

**>>> gg = p9.ggplot(total\_accuracy\_df, p9.aes(x ='accuracy', y = 'algorithm'))+\**

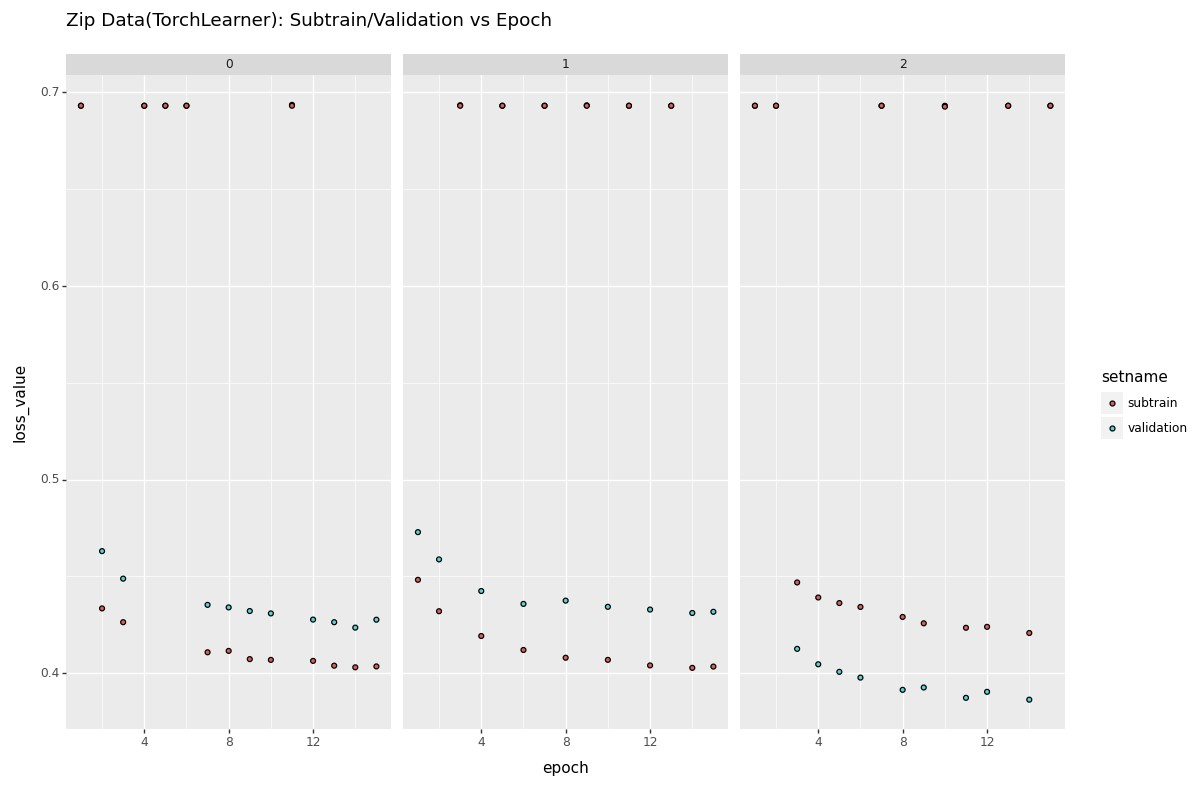
**... p9.facet\_grid('.~data\_set') + p9.geom\_point()**

**>>> gg.save("output.png", height = 8, width = 12)**



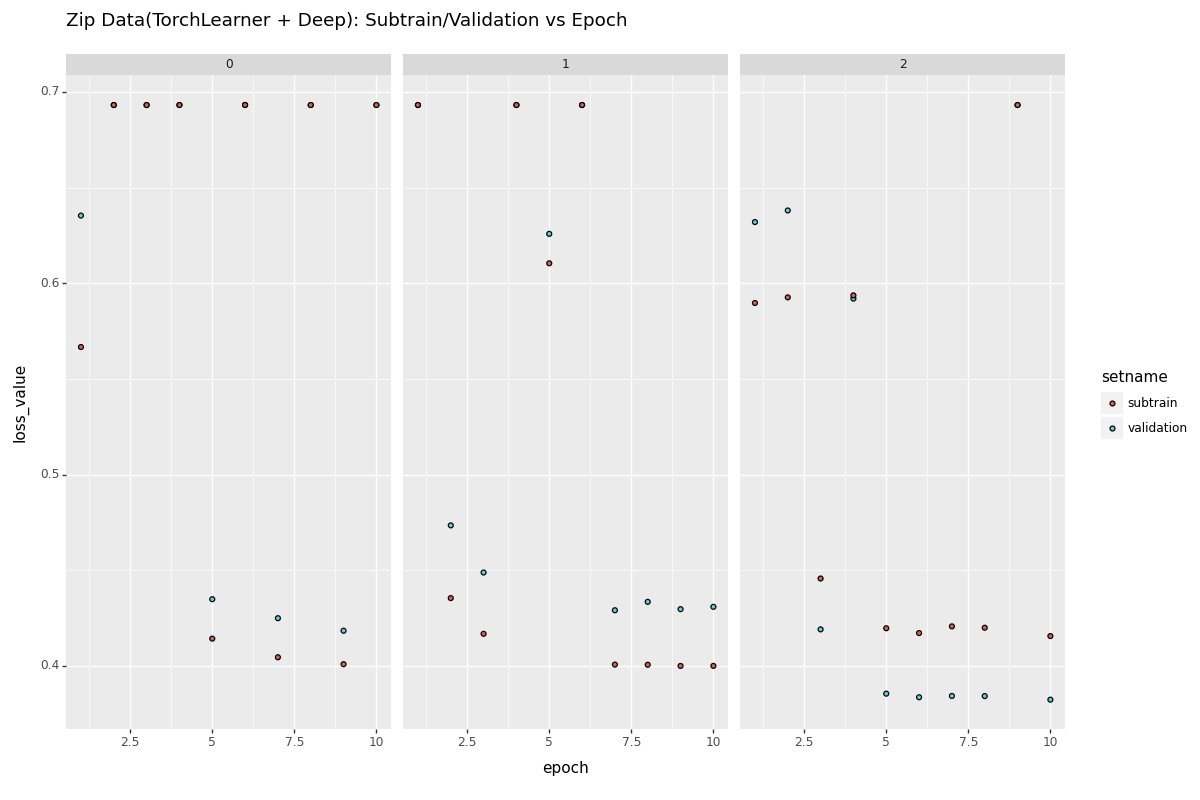
**>>> gg1 = p9.ggplot(zip\_loss["TorchLearnerCV"], p9.aes(x ='epoch', y = 'loss\_value', fill = 'setname')) + p9.facet\_grid('.~validation\_fold') + p9.geom\_point() + p9.ggtitle("Zip Data(TorchLearner): Subtrain/Validation vs Epoch") #p9.geom\_line(p9.aes(fill = 'setname'))**

**>>> gg1.save("valid\_graph1.png", height = 8, width = 12)**



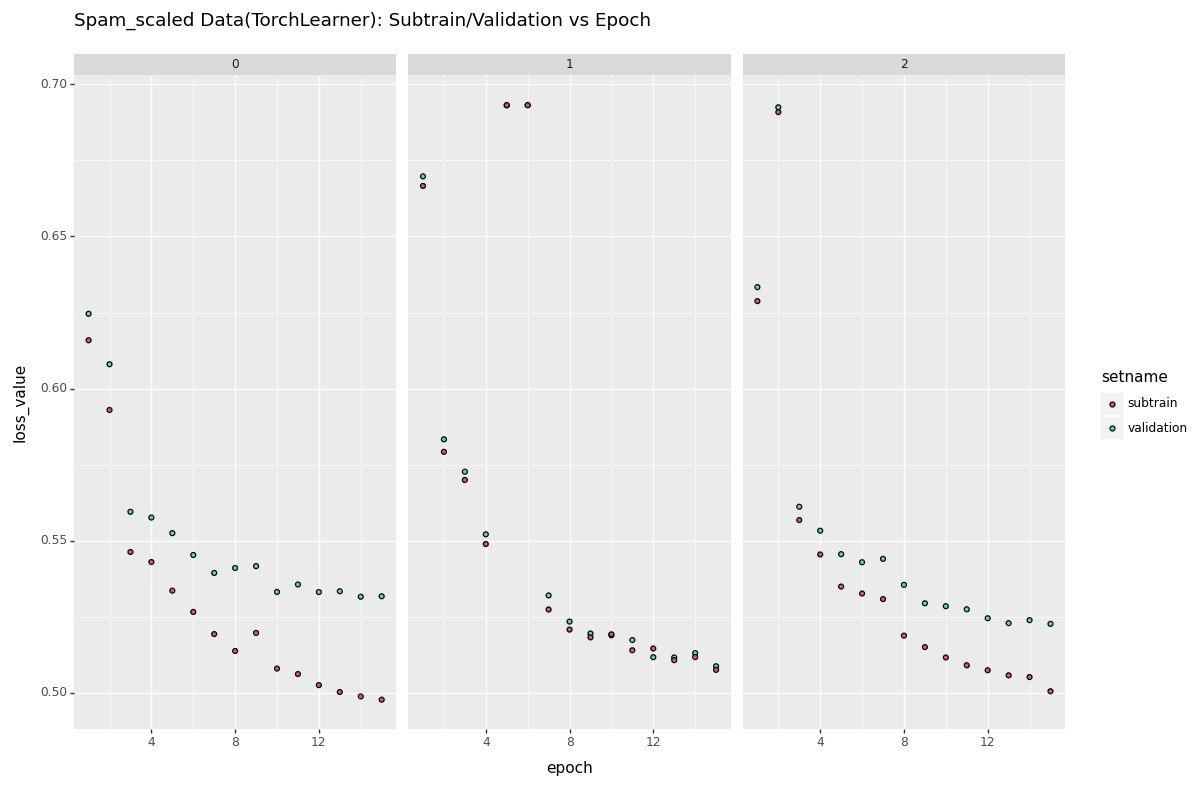
**>>> gg2 = p9.ggplot(zip\_loss["TorchLearnerCV + Deep"], p9.aes(x ='epoch', y = 'loss\_value', fill = 'setname')) + p9.facet\_grid('.~validation\_fold') + p9.geom\_point() + p9.ggtitle("Zip Data(TorchLearner + Deep): Subtrain/Validation vs Epoch") #p9.geom\_line(p9.aes(fill = 'setname'))**

**>>> gg2.save("c:/Users/Anudeep Kumar/OneDrive/Desktop/Fall 2023/CS599-Deep Learning/Homework/HW06/valid\_graph2.png", height = 8, width = 12)**



**>>> gg3 = p9.ggplot(spam\_loss["TorchLearnerCV"], p9.aes(x ='epoch', y = 'loss\_value', fill = 'setname')) + p9.facet\_grid('.~validation\_fold') + p9.geom\_point() + p9.ggtitle("Spam\_scaled Data(TorchLearner): Subtrain/Validation vs Epoch") #p9.geom\_line(p9.aes(fill = 'setname'))**

**>>> gg3.save("valid\_graph3.png", height = 8, width = 12)**



**>>> gg4 = p9.ggplot(spam\_loss["TorchLearnerCV + Deep"], p9.aes(x ='epoch', y = 'loss\_value', fill = 'setname')) \**

**... + p9.facet\_grid('.~validation\_fold') + p9.geom\_point() + p9.ggtitle("spam\_scaled Data(TorchLearner + Deep): Subtrain/Validation vs Epoch") #p9.geom\_line(p9.aes(fill = 'setname'))**

**>>> gg4.save("valid\_graph4.png", height = 8, width = 12)**

