# University of Tehran
## Electrical and Computer Engineering Department
### Neural Networks and Deep Learning
### Homework 2

| | |
|---|---|
| Name | Danial Saeedi |
| Student No | 810198571 |
| Date | Monday, November 22, 2021 |

# Table of contents

# 1. Ionosphere Classification using Multilayer Perceptron

For these exercises we'll perform a binary classification on the Ionosphere Data Set available from the UC Irvine Machine Learning Repository The goal is to classify radar returns from the ionosphere.

## Data Set Description

This radar data was collected by a system in Goose Bay, Labrador. This system consists of a phased array of 16 high-frequency antennas with a total transmitted power on the order of 6.4 kilowatts. See the paper for more details. The targets were free electrons in the ionosphere. "Good" radar returns are those showing evidence of some type of structure in the ionosphere. "Bad" returns are those that do not; their signals pass through the ionosphere.

Received signals were processed using an autocorrelation function whose arguments are the time of a pulse and the pulse number. There were 17 pulse numbers for the Goose Bay system. Instances in this database are described by 2 attributes per pulse number, corresponding to the complex values returned by the function resulting from the complex electromagnetic signal.

1. **Number of Instances:** 351
2. **Number of Attributes:** 34 plus the class attribute (Note that all 34 predictor attributes are continuous.)
3. **Attribute Information:** All 34 are continuous, as described above the 35th attribute is either "good" or "bad" according to the definition. This is a binary classification task.

## Loading the data

*Figure 1 The Ionosphere DataFrame*

| 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| -0.05889 | 0.85243 | 0.02306 | 0.83398 | -0.37708 | 1.00000 | 0.03760 | ... | -0.51171 | 0.41078 | -0.46168 | 0.21266 | -0.34090 | 0.42267 | -0.54487 | 0.18641 | -0.45300 | g |
| -0.18829 | 0.93035 | -0.36156 | -0.10868 | -0.93597 | 1.00000 | -0.04549 | ... | -0.26569 | -0.20468 | -0.18401 | -0.19040 | -0.11593 | -0.16626 | -0.06288 | -0.13738 | -0.02447 | b |
| -0.03365 | 1.00000 | 0.00485 | 1.00000 | -0.12062 | 0.88965 | 0.01198 | ... | -0.40220 | 0.58984 | -0.22145 | 0.43100 | -0.17365 | 0.60436 | -0.24180 | 0.56045 | -0.38238 | g |
| -0.45161 | 1.00000 | 1.00000 | 0.71216 | -1.00000 | 0.00000 | 0.00000 | ... | 0.90695 | 0.51613 | 1.00000 | 1.00000 | -0.20099 | 0.25682 | 1.00000 | -0.32382 | 1.00000 | b |
| -0.02401 | 0.94140 | 0.06531 | 0.92106 | -0.23255 | 0.77152 | -0.16399 | ... | -0.65158 | 0.13290 | -0.53206 | 0.02431 | -0.62197 | -0.05707 | -0.59573 | -0.04608 | -0.65697 | g |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 0.08298 | 0.73739 | -0.14706 | 0.84349 | -0.05567 | 0.90441 | -0.04622 | ... | -0.04202 | 0.83479 | 0.00123 | 1.00000 | 0.12815 | 0.86660 | -0.10714 | 0.90546 | -0.04307 | g |
| 0.00419 | 0.95183 | -0.02723 | 0.93438 | -0.01920 | 0.94590 | 0.01606 | ... | 0.01361 | 0.93522 | 0.04925 | 0.93159 | 0.08168 | 0.94066 | -0.00035 | 0.91483 | 0.04712 | g |
| -0.00034 | 0.93207 | -0.03227 | 0.95177 | -0.03431 | 0.95584 | 0.02446 | ... | 0.03193 | 0.92489 | 0.02542 | 0.92120 | 0.02242 | 0.92459 | 0.00442 | 0.92697 | -0.00577 | g |
| -0.01657 | 0.98122 | -0.01989 | 0.95691 | -0.03646 | 0.85746 | 0.00110 | ... | -0.02099 | 0.89147 | -0.07760 | 0.82983 | -0.17238 | 0.96022 | -0.03757 | 0.87403 | -0.16243 | g |
| 0.13533 | 0.73638 | -0.06151 | 0.87873 | 0.08260 | 0.88928 | -0.09139 | ... | -0.15114 | 0.81147 | -0.04822 | 0.78207 | -0.00703 | 0.75747 | -0.06678 | 0.85764 | -0.06151 | g |

# Data pre-processing

## Dealing with categorical columns

Categorical data are variables that contain label values rather than numeric values. Many machine learning algorithms cannot operate on label data directly. They require all input variables and output variables to be numeric.

Let's assign good and bad category with 1 and 0 respectively. Note that the 0 column acts like categorical feature.

```
df['target'] = df['target'].replace({'g':1, 'b' : 0})
```

As you can see, the dataset is not balanced:

*Figure 2 Value Counts on Target*

```
g    225
b    126
Name: target, dtype: int64
```

## Removing Redundant Column

As you can see, the 1 column is 100% filled with 0.  So this column doesn't give us any specific information.  We are going to drop this column.

```
In [111]: df[1].value_counts()/ len(df) * 100

Out[111]: 0    100.0
          Name: 1, dtype: float64

In [112]: df = df.drop(1,axis = 1)
```

## Train/Test Split

There are a number of ways to split the data into training and testing sets. The most common approach is to use some version of random sampling.

1. **Random sample:** Completely random sampling is a straightforward strategy to implement and usually protects the process from being biased towards any characteristic of the data.

2. **Stratified random sample:** However random sampling approach can be problematic when the response is not evenly distributed across the outcome. A less risky splitting strategy would be to use a stratified random sample based on the outcome. For classification models, this is accomplished by selecting samples at random within each class. This approach ensures that the frequency distribution of the outcome is approximately equal within the training and test sets.

3. **Splitting using the temporal component**: Non-random sampling can also be used when there is a good reason. One such case would be when there is an important temporal aspect to the data. Here it may be prudent to use the most recent data as the test set.

Since the categories are not **evenly distributed**, we're going to use **Stratified Random Sampling** for this problem.

In [113]:
```
X = df.drop('target',axis = 1).to_numpy()
y = df['target'].to_numpy()
```

In [114]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

## Scaling the data

The data is already scaled.

## Convert numpy arrays to tensors

We have to convert the numpy arrays to tensors to work with PyTorch.

In [116]:
```
y_col = ['target']
# Categorical Columns
cat_cols = [0]
# Continuous Columns
cont_cols = [x for x in range(2,34)]
```

In [117]:
```
test_cats = X_test[:, 0:1]
test_conts = X_test[:, 1:]
test_cats = torch.tensor(test_cats, dtype = torch.int64)
test_conts = torch.tensor(test_conts, dtype=torch.float)

test_conts
```

Out[117]:
```
tensor([[ 0.6068, -0.0271,  0.6712,  ..., -0.0271,  0.5017,  0.0678],
        [ 0.6234, -0.0349,  0.5908,  ..., -0.0096,  0.6137, -0.0915],
        [ 1.0000,  0.0980,  1.0000,  ...,  0.5452, -0.4281,  0.4453],
        ...,
        [ 0.9954, -0.0589,  0.8524,  ..., -0.5449,  0.1864, -0.4530],
        [-1.0000,  1.0000, -1.0000,  ..., -0.3232,  0.1494,  1.0000],
        [ 0.6585,  0.4362,  0.4468,  ...,  0.7676, -0.1853,  0.7436]])
```

In [118]:
```
# Convert labels to a tensor
y_test_tensor = torch.tensor(y_test).flatten()

y_test_tensor[:10]
```

Out[118]:
```
tensor([1, 1, 1, 1, 1, 1, 1, 0, 1, 1])
```

## Set an embedding size

Since the data set contains categorical feature, we can use embdeddings to deal with categorical features.

In [119]:
```
cat_szs = [len(df[col].cat.categories) for col in cat_cols]
emb_szs = [(size, min(50, (size+1)//2)) for size in cat_szs]
emb_szs
```

Out[119]:
```
[(2, 1)]
```

## Define the model

Paramaters of our model :

- **emb_szs:** each categorical variable size is paired with an embedding size
- **n_cont:** number of continuous variables
- **out_sz:** output size
- **layers:** layer sizes
- **p:** dropout probability for each layer

In [120]:
```
import torch.nn as nn
seed = 42
torch.manual_seed(seed)

if torch.cuda.is_available():
    torch.cuda.manual_seed_all(seed)

class TabularModel(nn.Module):

    def __init__(self, emb_szs, n_cont, out_sz, layers, p=0.5, activation_function = nn.ReLU(inplace=True)):
        super().__init__()
        self.embeds = nn.ModuleList([nn.Embedding(ni, nf) for ni,nf in emb_szs])
        self.emb_drop = nn.Dropout(p)
        self.bn_cont = nn.BatchNorm1d(n_cont)

        layerlist = []
        n_emb = sum((nf for ni,nf in emb_szs))
        n_in = n_emb + n_cont

        for i in layers:
            layerlist.append(nn.Linear(n_in,i))
            layerlist.append(activation_function)
            layerlist.append(nn.BatchNorm1d(i))
            layerlist.append(nn.Dropout(p))
            n_in = i
        layerlist.append(nn.Linear(layers[-1],out_sz))

        self.layers = nn.Sequential(*layerlist)

    def forward(self, x_cat, x_cont):
        embeddings = []
        for i,e in enumerate(self.embeds):
            embeddings.append(e(x_cat[:,i]))
        x = torch.cat(embeddings, 1)
        x = self.emb_drop(x)

        x_cont = self.bn_cont(x_cont)
        x = torch.cat([x, x_cont], 1)
        x = self.layers(x)
        return x
```

## Part A : Model Architecture

The input layer has 33 neurons (1 categorical and 32 continuous features). And we also need 2 neurons for output layer since the problem is a binary classification. There are two hidden layers with 200 and 100 neurons and the activation function is ReLU for each neuron in hidden layer. To avoid overfitting, we use Dropout.

## Train the model

Stochastic mini batch based is used to train the model

Stochastic mini batch based: training data set are randomly divided to a certain number of subsets (mini-batches) then at each time a min-batch is chosen and applied.

## Part B: Trying different neurons in hidden layers

### 1) 200 and 100 neurons in hidden layers (Best Model)

In [123]:
```
#Hidden Layer
hidden_layer = [200,100]
dropout_prob = 0.4
model = TabularModel(emb_szs = emb_szs, n_cont = len(cont_cols), out_sz = 2,
                     layers = hidden_layer , p = 0.4)
model
```

Out[123]:
```
TabularModel(
  (embeds): ModuleList(
    (0): Embedding(2, 1)
  )
  (emb_drop): Dropout(p=0.4, inplace=False)
  (bn_cont): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (layers): Sequential(
    (0): Linear(in_features=33, out_features=200, bias=True)
    (1): ReLU(inplace=True)
    (2): BatchNorm1d(200, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): Dropout(p=0.4, inplace=False)
    (4): Linear(in_features=200, out_features=100, bias=True)
    (5): ReLU(inplace=True)
    (6): BatchNorm1d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): Dropout(p=0.4, inplace=False)
    (8): Linear(in_features=100, out_features=2, bias=True)
  )
)
```

### Define loss function and optimizer

In [124]:
```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

In [125]:
```
%%time
epochs = 20
batch_size = 8
training_losses,valid_losses = train(epochs, batch_size,model,criterion,optimizer)
```

```
Epoch:   1  Training Loss: 0.62190312  Test Loss: 0.34160656
Epoch:  11  Training Loss: 0.39467219  Test Loss: 0.16658089
Epoch: 20  Training Loss: 0.02823020  Test Loss: 0.22117382
CPU times: user 1.43 s, sys: 793 ms, total: 2.23 s
Wall time: 1.55 s
```

*Figure 3 Loss per epoch*

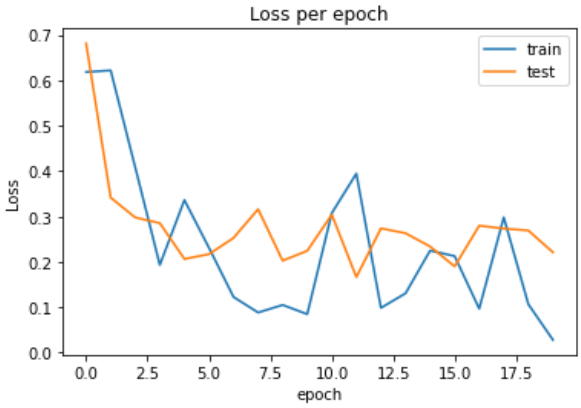
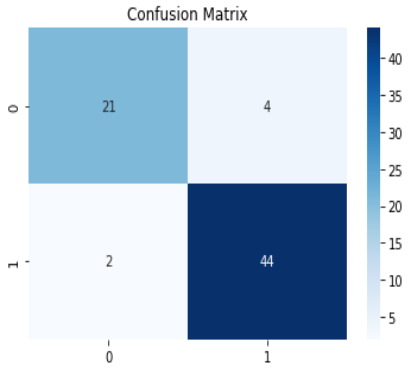



*Figure 4 Classification Report*

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| bad | 0.91 | 0.84 | 0.87 | 25 |
| good | 0.92 | 0.96 | 0.94 | 46 |
| accuracy | | | 0.92 | 71 |
| macro avg | 0.91 | 0.90 | 0.91 | 71 |
| weighted avg | 0.92 | 0.92 | 0.91 | 71 |

*Figure 5 Confusion Matrix*



## 2) 50 and 25 neurons in hidden layer

In [94]:
```
#Hidden Layer
hidden_layer = [50,25]
dropout_prob = 0.4
model = TabularModel(emb_szs = emb_szs, n_cont = len(cont_cols), out_sz = 2,
                     layers = hidden_layer , p = 0.4)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

epochs = 100
batch_size = 8
training_losses,valid_losses = train(epochs, batch_size,model,criterion,optimizer)
```

```
Epoch:   1  Training Loss: 0.71714860  Test Loss: 0.51422668
Epoch:  11  Training Loss: 0.26386631  Test Loss: 0.34749949
Epoch:  21  Training Loss: 0.08324081  Test Loss: 0.39248392
Epoch:  31  Training Loss: 0.14792909  Test Loss: 0.23937041
Epoch:  41  Training Loss: 0.08432895  Test Loss: 0.25949612
Epoch:  51  Training Loss: 0.06243191  Test Loss: 0.22705834
Epoch:  61  Training Loss: 0.04060852  Test Loss: 0.26546901
Epoch:  71  Training Loss: 0.11490460  Test Loss: 0.25009045
Epoch:  81  Training Loss: 0.14397264  Test Loss: 0.31953943
Epoch:  91  Training Loss: 0.10785655  Test Loss: 0.33026770
Epoch: 100  Training Loss: 0.22531484  Test Loss: 0.14982951
```

*Figure 6 Loss per epoch*

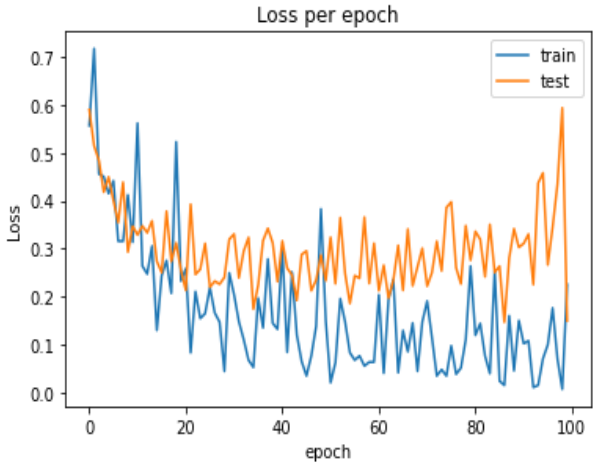
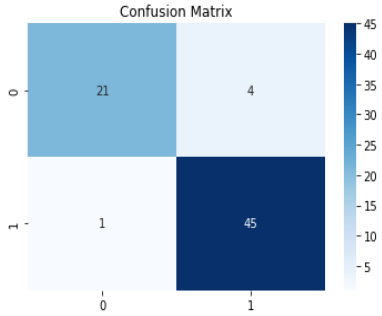



*Figure 7 Classification Report*

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| bad | 0.95 | 0.84 | 0.89 | 25 |
| good | 0.92 | 0.98 | 0.95 | 46 |
| accuracy | | | 0.93 | 71 |
| macro avg | 0.94 | 0.91 | 0.92 | 71 |
| weighted avg | 0.93 | 0.93 | 0.93 | 71 |

*Figure 8 Confusion Matrix*



## 3) 10 and 5 neurons in hidden layers

```
In [96]: #Hidden Layer
hidden_layer = [10,5]
dropout_prob = 0.4
model = TabularModel(emb_szs = emb_szs, n_cont = len(cont_cols), out_sz = 2,
                     layers = hidden_layer , p = 0.4)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

epochs = 100
batch_size = 8
training_losses,valid_losses = train(epochs, batch_size,model,criterion,optimizer)
```

```
Epoch:   1  Training Loss: 0.91334093  Test Loss: 0.63836235
Epoch:  11  Training Loss: 0.42331275  Test Loss: 0.46629602
Epoch:  21  Training Loss: 0.38164279  Test Loss: 0.39110014
Epoch:  31  Training Loss: 0.25546923  Test Loss: 0.42145112
Epoch:  41  Training Loss: 0.40848580  Test Loss: 0.42181981
Epoch:  51  Training Loss: 0.13651922  Test Loss: 0.31835487
Epoch:  61  Training Loss: 0.36991757  Test Loss: 0.26481104
Epoch:  71  Training Loss: 0.26827812  Test Loss: 0.32917529
Epoch:  81  Training Loss: 0.52536446  Test Loss: 0.30419219
Epoch:  91  Training Loss: 0.15110216  Test Loss: 0.26998088
Epoch: 100  Training Loss: 0.24640682  Test Loss: 0.30450398
```

*Figure 9 Loss per epoch*

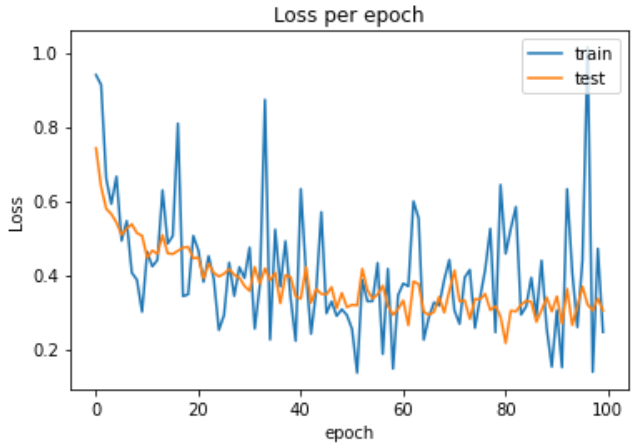
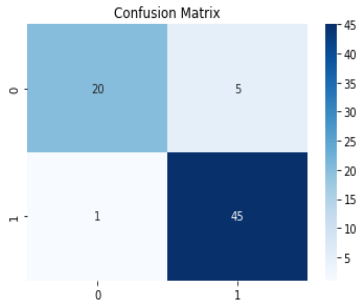



*Figure 10 Classification Report*

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| bad | 0.95 | 0.80 | 0.87 | 25 |
| good | 0.90 | 0.98 | 0.94 | 46 |
| accuracy | | | 0.92 | 71 |
| macro avg | 0.93 | 0.89 | 0.90 | 71 |
| weighted avg | 0.92 | 0.92 | 0.91 | 71 |

*Figure 11 Confusion Matrix*



## Part C: Trying different batch sizes

I got better performance with batch_size = 256.

In [98]:
```
#Hidden Layer
hidden_layer = [200,100]
dropout_prob = 0.4
model = TabularModel(emb_szs = emb_szs, n_cont = len(cont_cols), out_sz = 2,
                     layers = hidden_layer , p = 0.4)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
model
```

Out[98]:
```
TabularModel(
  (embeds): ModuleList(
    (0): Embedding(2, 1)
  )
  (emb_drop): Dropout(p=0.4, inplace=False)
  (bn_cont): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (layers): Sequential(
    (0): Linear(in_features=33, out_features=200, bias=True)
    (1): ReLU(inplace=True)
    (2): BatchNorm1d(200, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): Dropout(p=0.4, inplace=False)
    (4): Linear(in_features=200, out_features=100, bias=True)
    (5): ReLU(inplace=True)
    (6): BatchNorm1d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): Dropout(p=0.4, inplace=False)
    (8): Linear(in_features=100, out_features=2, bias=True)
  )
)
```

## 1) Batch size = 32

```
In [99]: %%time
epochs = 100
batch_size = 32
training_losses,valid_losses = train(epochs, batch_size,model,criterion,optimizer)
```

```
Epoch:   1  Training Loss: 0.32384023  Test Loss: 0.30881387
Epoch:  11  Training Loss: 0.04905418  Test Loss: 0.10539200
Epoch:  21  Training Loss: 0.06426143  Test Loss: 0.17545575
Epoch:  31  Training Loss: 0.01057324  Test Loss: 0.18864432
Epoch:  41  Training Loss: 0.00639555  Test Loss: 0.10483795
Epoch:  51  Training Loss: 0.01206115  Test Loss: 0.23777319
Epoch:  61  Training Loss: 0.00609993  Test Loss: 0.15787815
Epoch:  71  Training Loss: 0.00419355  Test Loss: 0.20059787
Epoch:  81  Training Loss: 0.00113583  Test Loss: 0.20238093
Epoch:  91  Training Loss: 0.00329989  Test Loss: 0.15064089
Epoch: 100  Training Loss: 0.00137709  Test Loss: 0.18729563
CPU times: user 2.56 s, sys: 1.65 s, total: 4.21 s
Wall time: 2.79 s
```

*Figure 12 Loss per epoch*

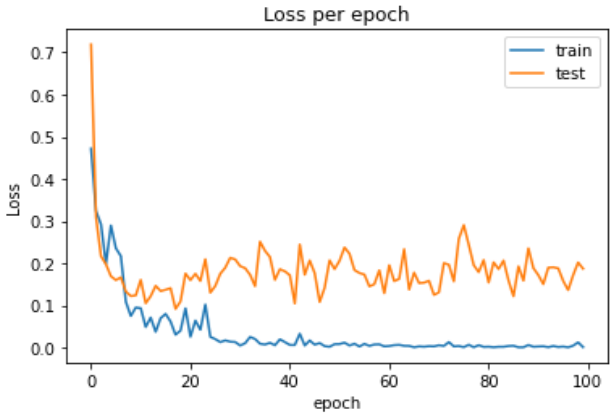
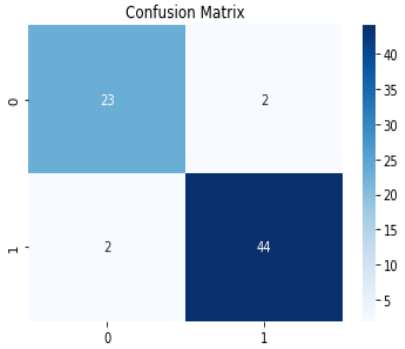



*Figure 13 Classification Report*

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| bad | 0.92 | 0.92 | 0.92 | 25 |
| good | 0.96 | 0.96 | 0.96 | 46 |
| accuracy | | | 0.94 | 71 |
| macro avg | 0.94 | 0.94 | 0.94 | 71 |
| weighted avg | 0.94 | 0.94 | 0.94 | 71 |

*Figure 14 Confusion Matrix*



## 2) Batch size = 64

*Figure 15 Loss per epoch*

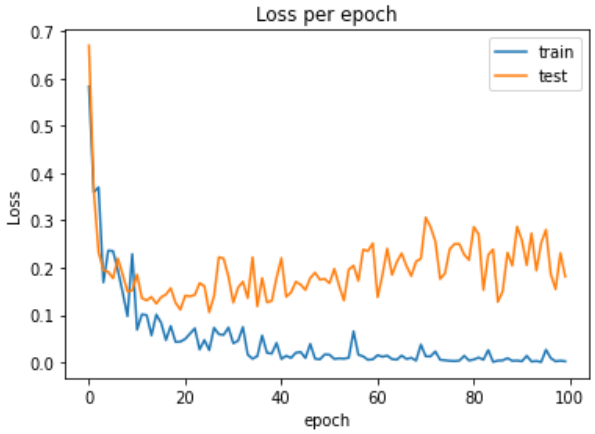
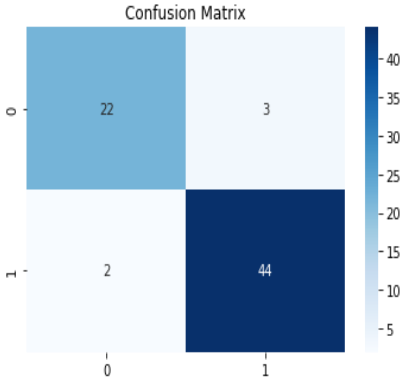



*Figure 16 Classification Report*

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| bad | 0.92 | 0.88 | 0.90 | 25 |
| good | 0.94 | 0.96 | 0.95 | 46 |
| accuracy | | | 0.93 | 71 |
| macro avg | 0.93 | 0.92 | 0.92 | 71 |
| weighted avg | 0.93 | 0.93 | 0.93 | 71 |

*Figure 17 Confusion Matrix*



## 3) Batch size = 256 (Best batch size among these 3)

*Figure 18 Loss per epoch*

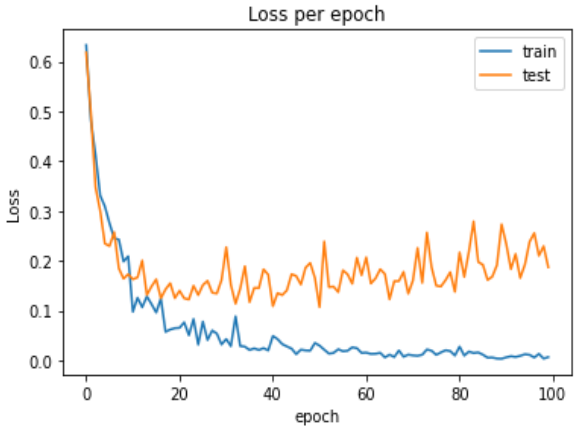
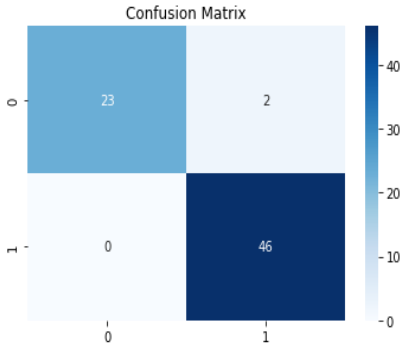



*Figure 19 Classification Report*

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| bad | 1.00 | 0.92 | 0.96 | 25 |
| good | 0.96 | 1.00 | 0.98 | 46 |
| accuracy | | | 0.97 | 71 |
| macro avg | 0.98 | 0.96 | 0.97 | 71 |
| weighted avg | 0.97 | 0.97 | 0.97 | 71 |

*Figure 20 Confusion Matrix*



## Part D: Trying different activation functions

**ReLu is less computationally expensive than tanh and sigmoid** because it involves simpler mathematical operations. That is a good point to consider when we are designing deep neural nets.

### 1) ReLU

In [146]:
```
%%time
#Hidden Layer
hidden_layer = [200,100]
model = TabularModel(emb_szs = emb_szs, n_cont = len(cont_cols), out_sz = 2,
                     layers = hidden_layer , p = 0.4,activation_function = nn.ReLU(inplace=True))
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

epochs = 100
batch_size = 256
training_losses,valid_losses = train(epochs, batch_size,model,criterion,optimizer)
```

```
Epoch:   1  Training Loss: 0.60060865  Test Loss: 0.49685556
Epoch:  11  Training Loss: 0.19695938  Test Loss: 0.20009656
Epoch:  21  Training Loss: 0.06015388  Test Loss: 0.19020692
Epoch:  31  Training Loss: 0.03675300  Test Loss: 0.14230473
Epoch:  41  Training Loss: 0.01881593  Test Loss: 0.16851123
Epoch:  51  Training Loss: 0.02255606  Test Loss: 0.16645643
Epoch:  61  Training Loss: 0.01423552  Test Loss: 0.10462214
Epoch:  71  Training Loss: 0.01087691  Test Loss: 0.10617708
Epoch:  81  Training Loss: 0.01476396  Test Loss: 0.20701312
Epoch:  91  Training Loss: 0.00730449  Test Loss: 0.12291852
Epoch: 100  Training Loss: 0.01103293  Test Loss: 0.24056059
CPU times: user 835 ms, sys: 479 ms, total: 1.31 s
Wall time: 876 ms
```

*Figure 21 Loss per epoch*

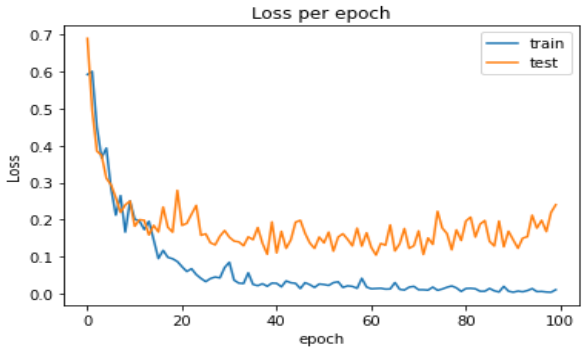
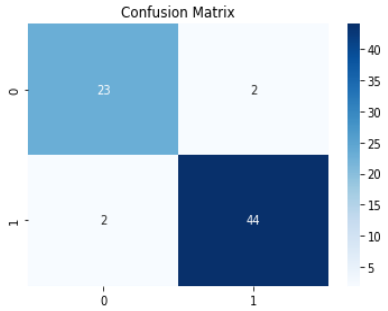



*Figure 22 Classification Report*

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| bad | 0.92 | 0.92 | 0.92 | 25 |
| good | 0.96 | 0.96 | 0.96 | 46 |
| accuracy | | | 0.94 | 71 |
| macro avg | 0.94 | 0.94 | 0.94 | 71 |
| weighted avg | 0.94 | 0.94 | 0.94 | 71 |

*Figure 23 Confusion Matrix*




## 2) Tanh

In [158]:
```
%%time
#Hidden Layer
hidden_layer = [200,100]
model = TabularModel(emb_szs = emb_szs, n_cont = len(cont_cols), out_sz = 2,
                     layers = hidden_layer , p = 0.4,activation_function = nn.Tanh())
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

epochs = 100
batch_size = 256
training_losses,valid_losses = train(epochs, batch_size,model,criterion,optimizer)
```

```
Epoch:   1  Training Loss: 0.55543119  Test Loss: 0.54752296
Epoch:  11  Training Loss: 0.31336850  Test Loss: 0.33012828
Epoch:  21  Training Loss: 0.18855424  Test Loss: 0.34263989
Epoch:  31  Training Loss: 0.18726522  Test Loss: 0.37712812
Epoch:  41  Training Loss: 0.07815560  Test Loss: 0.43446690
Epoch:  51  Training Loss: 0.06874162  Test Loss: 0.36460590
Epoch:  61  Training Loss: 0.08651375  Test Loss: 0.35500273
Epoch:  71  Training Loss: 0.04175164  Test Loss: 0.41723594
Epoch:  81  Training Loss: 0.02692732  Test Loss: 0.33744493
Epoch:  91  Training Loss: 0.04059235  Test Loss: 0.40713224
Epoch: 100  Training Loss: 0.08066297  Test Loss: 0.31222811
CPU times: user 882 ms, sys: 496 ms, total: 1.38 s
Wall time: 896 ms
```

*Figure 24 Loss per epoch*

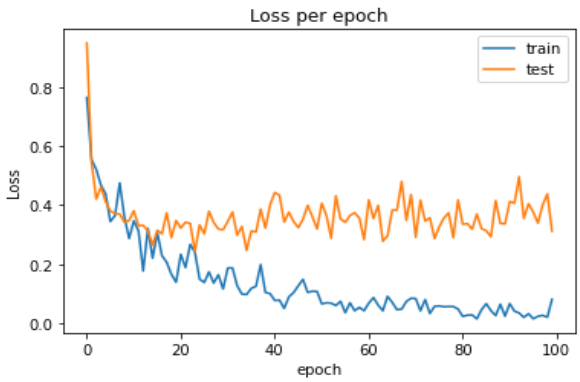
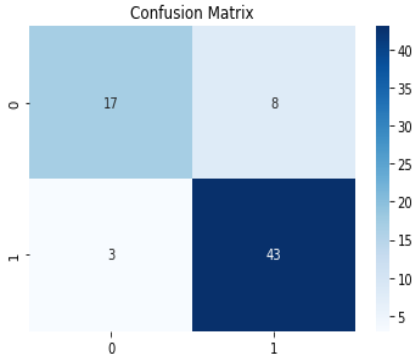



*Figure 25 Classification Report*

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| bad | 0.85 | 0.68 | 0.76 | 25 |
| good | 0.84 | 0.93 | 0.89 | 46 |
| accuracy | | | 0.85 | 71 |
| macro avg | 0.85 | 0.81 | 0.82 | 71 |
| weighted avg | 0.85 | 0.85 | 0.84 | 71 |

*Figure 26 Confusion Matrix*




## 3) Sigmoid

In [158]:
```
%%time
#Hidden Layer
hidden_layer = [200,100]
model = TabularModel(emb_szs = emb_szs, n_cont = len(cont_cols), out_sz = 2,
                     layers = hidden_layer , p = 0.4,activation_function = nn.Tanh())
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

epochs = 100
batch_size = 256
training_losses,valid_losses = train(epochs, batch_size,model,criterion,optimizer)
```

```
Epoch:   1  Training Loss: 0.55543119  Test Loss: 0.54752296
Epoch:  11  Training Loss: 0.31336850  Test Loss: 0.33012828
Epoch:  21  Training Loss: 0.18855424  Test Loss: 0.34263989
Epoch:  31  Training Loss: 0.18726522  Test Loss: 0.37712812
Epoch:  41  Training Loss: 0.07815560  Test Loss: 0.43446690
Epoch:  51  Training Loss: 0.06874162  Test Loss: 0.36460590
Epoch:  61  Training Loss: 0.08651375  Test Loss: 0.35500273
Epoch:  71  Training Loss: 0.04175164  Test Loss: 0.41723594
Epoch:  81  Training Loss: 0.02692732  Test Loss: 0.33744493
Epoch:  91  Training Loss: 0.04059235  Test Loss: 0.40713224
Epoch: 100  Training Loss: 0.08066297  Test Loss: 0.31222811
CPU times: user 882 ms, sys: 496 ms, total: 1.38 s
Wall time: 896 ms
```

*Figure 27 Loss per epochs*

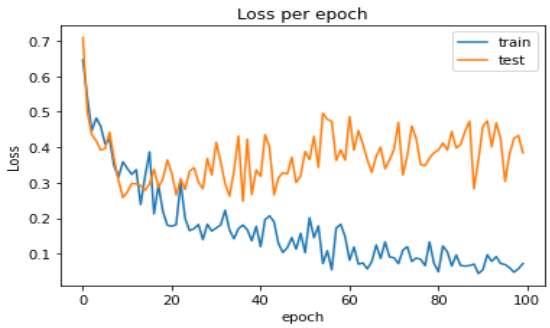
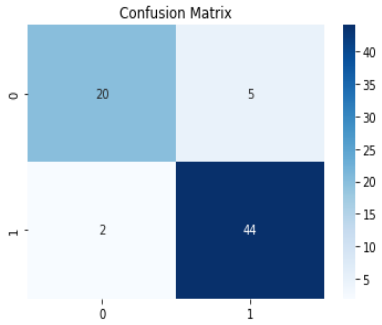



*Figure 28 Classification Report*

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| bad | 0.91 | 0.80 | 0.85 | 25 |
| good | 0.90 | 0.96 | 0.93 | 46 |
| accuracy | | | 0.90 | 71 |
| macro avg | 0.90 | 0.88 | 0.89 | 71 |
| weighted avg | 0.90 | 0.90 | 0.90 | 71 |

*Figure 29 Confusion Matrix*



## Part E: Trying different loss functions

### 1) Cross Entropy Loss (Better Loss Function)

```
In [103]: %%time
          #Hidden Layer
          hidden_layer = [200,100]
          model = TabularModel(emb_szs = emb_szs, n_cont = len(cont_cols), out_sz = 2,
                               layers = hidden_layer , p = 0.4)
          criterion = nn.CrossEntropyLoss()
          optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

          epochs = 100
          batch_size = 256
          training_losses,valid_losses = train(epochs, batch_size,model,criterion,optimizer)
```

```
Epoch:   1  Training Loss: 0.48220897  Test Loss: 0.49733526
Epoch:  11  Training Loss: 0.12577476  Test Loss: 0.16659279
Epoch:  21  Training Loss: 0.07638016  Test Loss: 0.12460247
Epoch:  31  Training Loss: 0.02857061  Test Loss: 0.15179414
Epoch:  41  Training Loss: 0.04276483  Test Loss: 0.13513318
Epoch:  51  Training Loss: 0.02148081  Test Loss: 0.23879696
Epoch:  61  Training Loss: 0.01300868  Test Loss: 0.15446523
Epoch:  71  Training Loss: 0.00925980  Test Loss: 0.22571531
Epoch:  81  Training Loss: 0.00959276  Test Loss: 0.16765778
Epoch:  91  Training Loss: 0.00835790  Test Loss: 0.18334152
Epoch: 100  Training Loss: 0.00654143  Test Loss: 0.18724841
CPU times: user 749 ms, sys: 448 ms, total: 1.2 s
Wall time: 784 ms
```

*Figure 30 Loss per epoch*

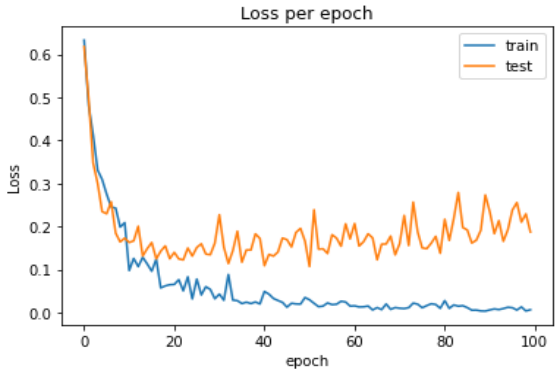
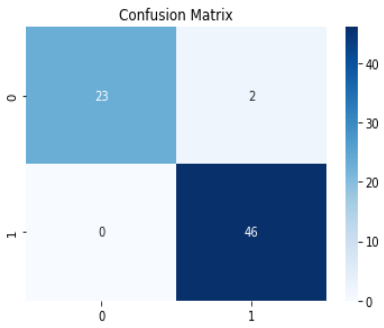



*Figure 31 Classification Report*

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| bad | 1.00 | 0.92 | 0.96 | 25 |
| good | 0.96 | 1.00 | 0.98 | 46 |
| accuracy | | | 0.97 | 71 |
| macro avg | 0.98 | 0.96 | 0.97 | 71 |
| weighted avg | 0.97 | 0.97 | 0.97 | 71 |

*Figure 32 Confusion Matrix*



## 2) Log Loss

```
In [30]: %%time
         #Hidden Layer
         hidden_layer = [200,100]
         model = TabularModel(emb_szs = emb_szs, n_cont = len(cont_cols), out_sz = 2,
                              layers = hidden_layer , p = 0.4)
         criterion = nn.NLLLoss()
         optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

         epochs = 100
         batch_size = 256
         training_losses,valid_losses = train(epochs, batch_size,model,criterion,optimizer)
```

```
Epoch:   1  Training Loss: -0.20925575  Test Loss: -0.13883080
Epoch:  11  Training Loss: -1.91808307  Test Loss: -1.84153664
Epoch:  21  Training Loss: -3.52905369  Test Loss: -3.97414684
Epoch:  31  Training Loss: -5.58826303  Test Loss: -5.91130257
Epoch:  41  Training Loss: -8.92961025  Test Loss: -8.63926506
Epoch:  51  Training Loss: -12.42132950  Test Loss: -11.70402718
Epoch:  61  Training Loss: -15.58216858  Test Loss: -14.69748211
Epoch:  71  Training Loss: -19.88549995  Test Loss: -17.27301598
Epoch:  81  Training Loss: -23.07675362  Test Loss: -21.15622330
Epoch:  91  Training Loss: -26.66721153  Test Loss: -25.09278107
Epoch: 100  Training Loss: -29.12736320  Test Loss: -27.17329597
CPU times: user 778 ms, sys: 460 ms, total: 1.24 s
Wall time: 819 ms
```

*Figure 33 Loss per epoch*

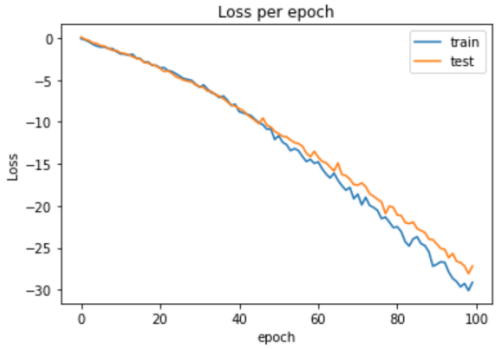
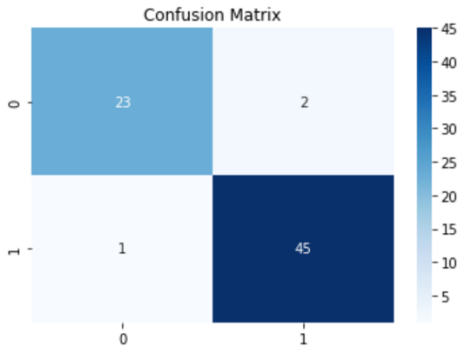



*Figure 34 Classification Report*

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| bad | 0.96 | 0.92 | 0.94 | 25 |
| good | 0.96 | 0.98 | 0.97 | 46 |
| accuracy | | | 0.96 | 71 |
| macro avg | 0.96 | 0.95 | 0.95 | 71 |
| weighted avg | 0.96 | 0.96 | 0.96 | 71 |

*Figure 35 Confusion Matrix*



## Difference between Cross-entropy and Log Loss

*Figure 36 Cross Entropy*

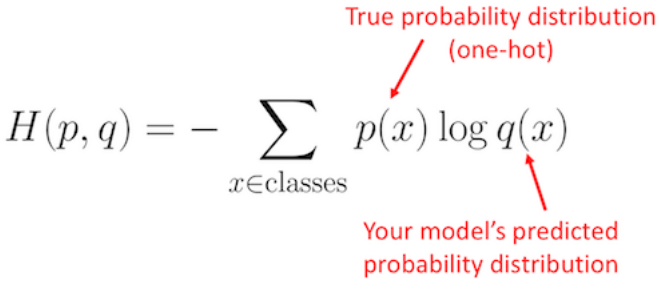



*Figure 37 Log Loss*

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Binary Cross-Entropy / Log Loss

## Part F: Trying different optimizer

### 1) Adam

```
In [148]: %%time
          #Hidden Layer
          hidden_layer = [200,100]
          model = TabularModel(emb_szs = emb_szs, n_cont = len(cont_cols), out_sz = 2,
                               layers = hidden_layer , p = 0.4,activation_function = nn.ReLU(inplace=True))
          criterion = nn.CrossEntropyLoss()
          optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

          epochs = 100
          batch_size = 256
          training_losses,valid_losses = train(epochs, batch_size,model,criterion,optimizer)
```

```
Epoch:   1  Training Loss: 0.44565174  Test Loss: 0.51320082
Epoch:  11  Training Loss: 0.14198481  Test Loss: 0.27689543
Epoch:  21  Training Loss: 0.04861319  Test Loss: 0.16317168
Epoch:  31  Training Loss: 0.03530030  Test Loss: 0.11977072
Epoch:  41  Training Loss: 0.01898927  Test Loss: 0.18118089
Epoch:  51  Training Loss: 0.01081735  Test Loss: 0.13353635
Epoch:  61  Training Loss: 0.01476086  Test Loss: 0.13677691
Epoch:  71  Training Loss: 0.00404578  Test Loss: 0.14645304
Epoch:  81  Training Loss: 0.00571570  Test Loss: 0.14776009
Epoch:  91  Training Loss: 0.00554995  Test Loss: 0.22054343
Epoch: 100  Training Loss: 0.00683897  Test Loss: 0.21742626
CPU times: user 801 ms, sys: 470 ms, total: 1.27 s
Wall time: 843 ms
```

*Figure 38 Loss per epochs*

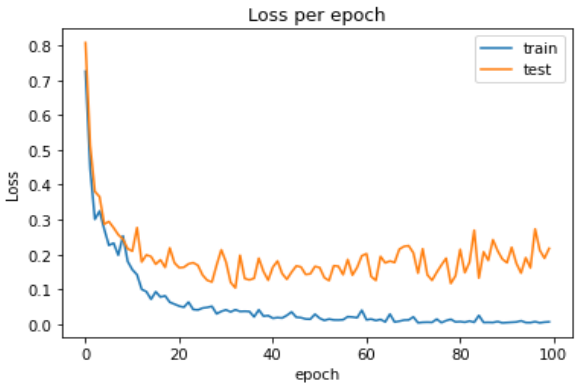
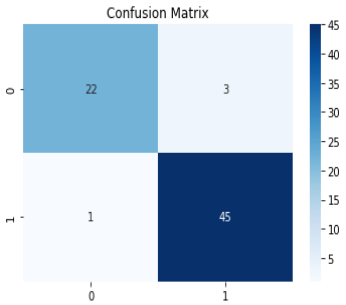



*Figure 39 Classification Report*

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| bad | 0.96 | 0.88 | 0.92 | 25 |
| good | 0.94 | 0.98 | 0.96 | 46 |
| accuracy | | | 0.94 | 71 |
| macro avg | 0.95 | 0.93 | 0.94 | 71 |
| weighted avg | 0.94 | 0.94 | 0.94 | 71 |

*Figure 40 Confusion Matrix*



## 2) SGD

```
In [150]: %%time
          #Hidden Layer
          hidden_layer = [200,100]
          model = TabularModel(emb_szs = emb_szs, n_cont = len(cont_cols), out_sz = 2,
                               layers = hidden_layer , p = 0.4,activation_function = nn.ReLU(inplace=True))
          criterion = nn.CrossEntropyLoss()
          optimizer = torch.optim.SGD(model.parameters(), lr=0.001, momentum=0.9)

          epochs = 100
          batch_size = 256
          training_losses,valid_losses = train(epochs, batch_size,model,criterion,optimizer)
```

```
Epoch:   1  Training Loss: 0.82231981  Test Loss: 0.95999587
Epoch:  11  Training Loss: 0.46583188  Test Loss: 0.52111655
Epoch:  21  Training Loss: 0.36435744  Test Loss: 0.34419987
Epoch:  31  Training Loss: 0.27025244  Test Loss: 0.27912906
Epoch:  41  Training Loss: 0.18643481  Test Loss: 0.27150407
Epoch:  51  Training Loss: 0.24527447  Test Loss: 0.23730910
Epoch:  61  Training Loss: 0.13481633  Test Loss: 0.21255817
Epoch:  71  Training Loss: 0.17034173  Test Loss: 0.22187352
Epoch:  81  Training Loss: 0.08256936  Test Loss: 0.21713212
Epoch:  91  Training Loss: 0.06940372  Test Loss: 0.16129522
Epoch: 100  Training Loss: 0.10985083  Test Loss: 0.24950041
CPU times: user 710 ms, sys: 445 ms, total: 1.15 s
Wall time: 745 ms
```

*Figure 41 Loss per epochs*

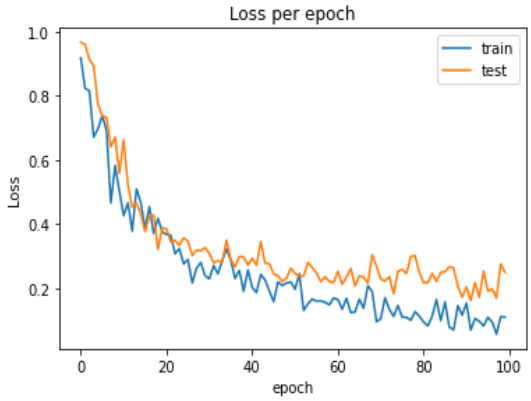
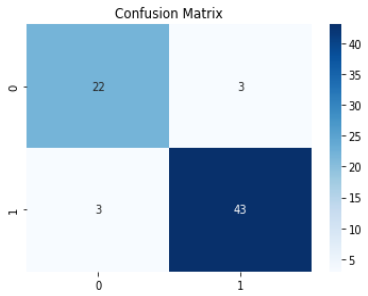



*Figure 42 Classification Report*

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| bad | 0.88 | 0.88 | 0.88 | 25 |
| good | 0.93 | 0.93 | 0.93 | 46 |
| accuracy | | | 0.92 | 71 |
| macro avg | 0.91 | 0.91 | 0.91 | 71 |
| weighted avg | 0.92 | 0.92 | 0.92 | 71 |

*Figure 43 Confusion Matrix*



## Part G: Trying different hidden layers

### Experiment 1) Two Hidden Layers

The first and second hidden layer has 200 and 100 neurons respectively.

```
In [152]: %%time
          #Hidden Layer
          hidden_layer = [200,100]
          model = TabularModel(emb_szs = emb_szs, n_cont = len(cont_cols), out_sz = 2,
                               layers = hidden_layer , p = 0.4,activation_function = nn.ReLU(inplace=True))
          criterion = nn.CrossEntropyLoss()
          optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

          epochs = 100
          batch_size = 256
          training_losses,valid_losses = train(epochs, batch_size,model,criterion,optimizer)
```

```
Epoch:   1  Training Loss: 0.53356183  Test Loss: 0.52898955
Epoch:  11  Training Loss: 0.10218924  Test Loss: 0.17803220
Epoch:  21  Training Loss: 0.04265514  Test Loss: 0.15224102
Epoch:  31  Training Loss: 0.01977436  Test Loss: 0.14956880
Epoch:  41  Training Loss: 0.03608501  Test Loss: 0.18900728
Epoch:  51  Training Loss: 0.01910626  Test Loss: 0.17464301
Epoch:  61  Training Loss: 0.01985645  Test Loss: 0.17634441
Epoch:  71  Training Loss: 0.01201971  Test Loss: 0.16345198
Epoch:  81  Training Loss: 0.01338158  Test Loss: 0.19146638
Epoch:  91  Training Loss: 0.00607552  Test Loss: 0.17287949
Epoch: 100  Training Loss: 0.00726395  Test Loss: 0.15588455
CPU times: user 902 ms, sys: 507 ms, total: 1.41 s
Wall time: 948 ms
```

*Figure 44 Loss per epoch*

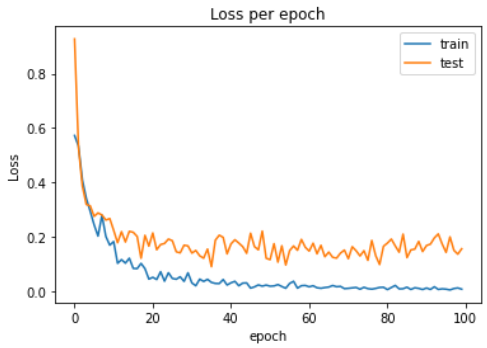
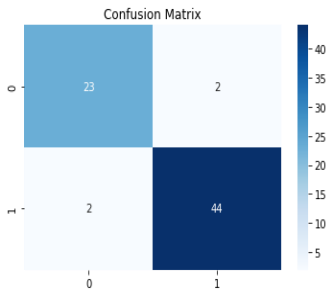



*Figure 45 Classification Report*

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| bad | 0.92 | 0.92 | 0.92 | 25 |
| good | 0.96 | 0.96 | 0.96 | 46 |
| accuracy | | | 0.94 | 71 |
| macro avg | 0.94 | 0.94 | 0.94 | 71 |
| weighted avg | 0.94 | 0.94 | 0.94 | 71 |

*Figure 46 Confusion Matrix*



## Experiment 2) Three Hidden Layers

The first, second and third hidden layer has 200, 100 and 50 neurons respectively.

```
In [154]: %%time
          #Hidden Layer
          hidden_layer = [200,100,50]
          model = TabularModel(emb_szs = emb_szs, n_cont = len(cont_cols), out_sz = 2,
                               layers = hidden_layer , p = 0.4,activation_function = nn.ReLU(inplace=True))
          criterion = nn.CrossEntropyLoss()
          optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

          epochs = 100
          batch_size = 256
          training_losses,valid_losses = train(epochs, batch_size,model,criterion,optimizer)
```

```
Epoch:   1  Training Loss: 0.55154443  Test Loss: 0.73761421
Epoch:  11  Training Loss: 0.32727692  Test Loss: 0.33595634
Epoch:  21  Training Loss: 0.11399386  Test Loss: 0.26399460
Epoch:  31  Training Loss: 0.09092387  Test Loss: 0.25530928
Epoch:  41  Training Loss: 0.08642671  Test Loss: 0.18229865
Epoch:  51  Training Loss: 0.04091557  Test Loss: 0.16349091
Epoch:  61  Training Loss: 0.03778292  Test Loss: 0.22665599
Epoch:  71  Training Loss: 0.02927524  Test Loss: 0.14000237
Epoch:  81  Training Loss: 0.03671245  Test Loss: 0.15996775
Epoch:  91  Training Loss: 0.01751313  Test Loss: 0.20565061
Epoch: 100  Training Loss: 0.02489981  Test Loss: 0.19093679
CPU times: user 1.05 s, sys: 626 ms, total: 1.67 s
Wall time: 1.1 s
```

*Figure 47 Loss per epoch*

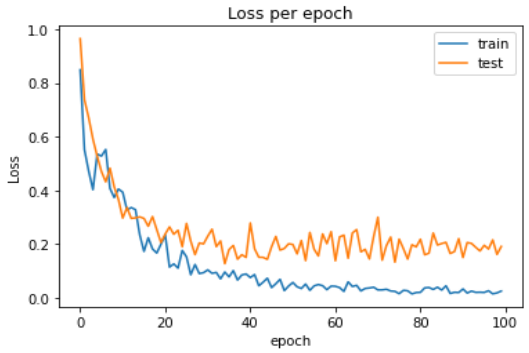
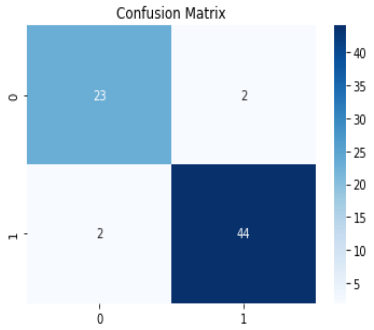



*Figure 48 Classification Report*

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| bad | 0.92 | 0.92 | 0.92 | 25 |
| good | 0.96 | 0.96 | 0.96 | 46 |
| accuracy | | | 0.94 | 71 |
| macro avg | 0.94 | 0.94 | 0.94 | 71 |
| weighted avg | 0.94 | 0.94 | 0.94 | 71 |

*Figure 49 Confusion Matrix*



## Experiment 3: Four Hidden Layers

The first, second, third and fourth hidden layer has 1000,500, 250 and 125 neurons respectively.

```
In [156]: %%time
          #Hidden Layer
          hidden_layer = [1000,500,250,125]
          model = TabularModel(emb_szs = emb_szs, n_cont = len(cont_cols), out_sz = 2,
                               layers = hidden_layer , p = 0.4, activation_function = nn.ReLU(inplace=True))
          criterion = nn.CrossEntropyLoss()
          optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

          epochs = 100
          batch_size = 256
          training_losses,valid_losses = train(epochs, batch_size,model,criterion,optimizer)
```

```
Epoch:   1  Training Loss: 0.36121634  Test Loss: 0.39482105
Epoch:  11  Training Loss: 0.04201045  Test Loss: 0.15041414
Epoch:  21  Training Loss: 0.02160096  Test Loss: 0.18810269
Epoch:  31  Training Loss: 0.01818898  Test Loss: 0.13590696
Epoch:  41  Training Loss: 0.00961359  Test Loss: 0.20043676
Epoch:  51  Training Loss: 0.00778492  Test Loss: 0.17540097
Epoch:  61  Training Loss: 0.00382747  Test Loss: 0.14473963
Epoch:  71  Training Loss: 0.00249039  Test Loss: 0.31123099
Epoch:  81  Training Loss: 0.00702321  Test Loss: 0.28855249
Epoch:  91  Training Loss: 0.00185994  Test Loss: 0.31827003
Epoch: 100  Training Loss: 0.00171150  Test Loss: 0.28393278
CPU times: user 4.98 s, sys: 1.68 s, total: 6.67 s
Wall time: 3.87 s
```

*Figure 50 Loss per epoch*

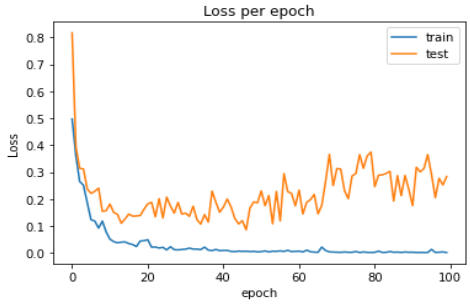
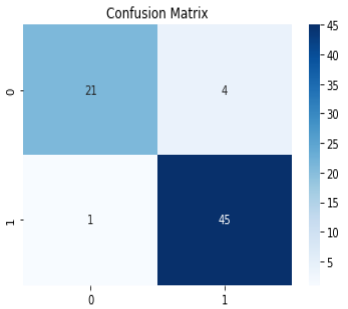



*Figure 51 Classification Report*

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| bad | 1.00 | 0.88 | 0.94 | 25 |
| good | 0.94 | 1.00 | 0.97 | 46 |
| accuracy | | | 0.96 | 71 |
| macro avg | 0.97 | 0.94 | 0.95 | 71 |
| weighted avg | 0.96 | 0.96 | 0.96 | 71 |

*Figure 52 Confusion Matrix*



## Part H: Conclusion

According to previous parts, these are the parameters:

- Batch size: 256
- Loss Function: CrossEntropyLoss
- Activation Function: ReLU
- Learning Rate: 0.001
- Optimizer: Adam
- Four hidden layers with 1000, 500, 250 and 125 neurons.

We reached 96% accuracy with these parameters. Note that these parameters were found through experiments.

## Part I: What happens when labels are non-uniformly distributed? How do you deal with it?

Suppose there are 50 data labeled A and 200 data labeled B. The model is more biased towards class B after training because the model has seen more class B than A. So if our dataset is not uniformly distributed, we should try these methods:

### 1) Stratified sampling

This method is used to deal with imbalanced labels in this exercise.  It is done by dividing the population into subgroups or into strata, and the right number of instances are sampled from each stratum to guarantee that the test set is representative of the entire population.

### 2) Oversampling

Randomly duplicate examples in the minority class.

### 3) Undersampling

Randomly delete examples in the majority class.

### 4) Data augmentation

Data augmentation in data analysis are techniques used to increase the amount of data by adding slightly modified copies of already existing data or newly created synthetic data from existing data. It acts as a regularizer and helps reduce overfitting when training a machine learning model.

## 2. Multi-layer Perceptron Regression Problem

The dataset can be found in data/Reg-Data.txt This dataset has 68 continuous features, longitude and latitude in the end of each line. The goal is to predict the longitude and latitude based on the 68 continuous features.

### Loading the data

In [2]:
```
df = pd.read_csv('data/Reg-Data.txt',header=None)
df['longitude'] = df[68]
df['latitude'] = df[69]
df = df.drop([68,69],axis = 1)
X = df.drop(['longitude','latitude'],axis = 1).values
y = df[['longitude','latitude']].values
df
```

Out[2]:

| 5 | 6 | 7 | 8 | 9 | ... | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | longitude | latitude |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2.094097 | 0.576000 | -1.205671 | 1.849122 | -0.425598 | ... | -1.504263 | 0.351267 | -1.018726 | -0.174878 | -1.089543 | -0.668840 | -0.914772 | -0.836250 | -15.75 | -47.95 |
| 0.290280 | -0.077659 | -0.887385 | 0.432062 | -0.093963 | ... | -0.495712 | -0.465077 | -0.157861 | -0.157189 | 0.380951 | 1.088478 | -0.123595 | 1.391141 | 14.91 | -23.51 |
| 0.880213 | 0.406899 | -0.694895 | -0.901869 | -1.701574 | ... | -0.637167 | 0.147260 | 0.217914 | 2.718442 | 0.972919 | 2.081069 | 1.375763 | 1.063847 | 12.65 | -8.00 |
| 0.805396 | 1.497982 | 0.114752 | 0.692847 | 0.052377 | ... | -0.178325 | -0.065059 | -0.724247 | -1.020687 | -0.751380 | -0.385005 | -0.012326 | -0.392197 | 9.03 | 38.74 |
| 0.869295 | -0.265858 | -0.401676 | -0.872639 | 1.147483 | ... | -0.919463 | -0.667912 | -0.820172 | -0.190488 | 0.306974 | 0.119658 | 0.271838 | 1.289783 | 34.03 | -6.85 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 0.943453 | 0.114960 | -0.335898 | 0.826753 | -0.393786 | ... | -0.558717 | 0.998897 | -0.106835 | 1.526307 | 0.646088 | 2.467278 | 1.867699 | 1.719302 | -6.17 | 35.74 |
| 2.128963 | -1.875967 | 0.094232 | -1.429742 | 0.873777 | ... | 0.223143 | -0.032425 | 0.226782 | 0.182107 | 0.517466 | 1.126762 | 2.220671 | 4.422651 | 11.55 | 104.91 |
| 1.152162 | 0.241470 | 0.229092 | 0.019036 | -0.068804 | ... | 0.449239 | -0.965270 | -0.590039 | -0.804297 | 0.044170 | -0.718175 | -0.983640 | -0.573822 | 41.33 | 19.80 |
| 0.192081 | 0.069821 | 0.264674 | -0.411533 | 0.501164 | ... | 1.941398 | 1.769292 | 0.738616 | 1.240377 | -0.546002 | -0.137473 | -0.781036 | -0.832167 | 54.68 | 25.31 |
| 0.954948 | -1.527722 | -1.591471 | -3.678713 | -5.930209 | ... | 5.121875 | 4.103031 | 3.673086 | 0.960420 | 1.067164 | 5.244305 | 2.506568 | 1.462580 | 54.68 | 25.31 |

### Train/Test Split

15% of the dataset is assigned to test set. 10% of the remaining data is assigned to validation set. And the rest is assigned to training set.

In [3]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.15, random_state=42
)

X_train, X_val, y_train, y_val = train_test_split(
    X_train, y_train, test_size=0.1, random_state=42)
```

## Data Preprocessing

### Dealing with null values

There are no null values.

```
df.isnull().sum()
```

```
0            0
1            0
2            0
3            0
4            0
            ..
65           0
66           0
67           0
longitude    0
latitude     0
Length: 70, dtype: int64
```

### Scaling features to a range

68 continuous features are not between 0 and 1. So let's use MinMaxScaler to scale this dataset.

In [5]:
```
from sklearn import preprocessing
scaler = preprocessing.MinMaxScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)
```

### Converting the numpy arrays to tensors

In [6]:
```
X_train_tensor = torch.tensor(X_train, dtype=torch.float)
X_val_tensor = torch.tensor(X_val, dtype=torch.float)
X_test_tensor = torch.tensor(X_test, dtype=torch.float)

y_train_tensor = torch.tensor(y_train, dtype=torch.float).reshape(-1,2)
y_val_tensor = torch.tensor(y_val, dtype=torch.float).reshape(-1,2)
y_test_tensor = torch.tensor(y_test, dtype=torch.float).reshape(-1,2)
```

## Part 1: Linear Regression using Sklearn

In [7]:
```
%%time
from sklearn.linear_model import LinearRegression
reg = LinearRegression().fit(X_train, y_train)

print(f'Loss : {reg.score(X_test, y_test)}')
```

```
Loss : 0.14647679073528275
CPU times: user 28.7 ms, sys: 16.6 ms, total: 45.3 ms
Wall time: 57.8 ms
```

In [8]:
```
from sklearn.metrics import mean_squared_error, mean_absolute_error
y_pred = reg.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

print(f'MSE : {mse}  MAE : {mae}')
```

```
MSE : 1170.5617551861633  MAE : 24.352326787139795
```

**Bonus part :**

Linear regression is one of the basic and well used type of predictive analysis in machine learning. It is used to predict the value of a variable based on the value of another variable. The variable you want to predict is called the dependent variable. The variable you are using to predict the other variable's value is called the independent variable.

This form of analysis estimates the coefficients of the linear equation, involving one or more independent variables that best predict the value of the dependent variable. Linear regression fits a straight line or surface that minimizes the discrepancies between predicted and actual output values. There are simple linear regression calculators that use a “least squares errors” method to discover the best-fit line for a set of paired data.

**Ridge** is part of the Linear Regression model where the input and output are assumed to have a linear relationship. In python programming, there is scikit-learn library for predictive data analysis. In scikit-learn, linear regression refers to  least square linear regression method without regularization (regularization means penalty on weights).

However, ridge has different method of regularization. It penalizes the model for the sum of squared value of the weights. So the weights tend to have smaller values and tend to penalize the extremes of the weights in a group of weights that are more evenly distributed.

Here is the objective function in ridge:

$$Cost\ (W) = RSS(W) + \ \lambda * (sum\ of\ squares\ of\ weights)$$

$$= \sum_{i=1}^{N} \left\{ y_i \ - \sum_{j=0}^{M} w_j \, x_{ij} \right\}^2 + \lambda \sum_{j=0}^{M} w_j^2$$

One of the advantages of using Ridge is that it tends to give small but well distributed weights, because the L2 regularization cares more about driving big weight to small weights, instead of driving small weights to zeros.

In some problems with a few predictors which are really relevant in predictions, Ridge can be a good regularized linear model for predictive analytics.

## Defining the model

In [292]:
```
import torch.nn as nn

class Model(nn.Module):
    def __init__(self, n_cont, out_sz, layers, p=0.5, activation_function = nn.ReLU(inplace=True),
                 enable_dropout = False, enable_batch_normalization = False) :
        super().__init__()
        self.enable_batch_normalization = enable_batch_normalization
        if enable_batch_normalization :
            self.bn_cont = nn.BatchNorm1d(n_cont)

        layerlist = []
        n_in = n_cont
        if len(layers) == 0 :
            layerlist.append(nn.Linear(n_in,out_sz))
            if activation_function != None :
                layerlist.append(activation_function)
            if enable_dropout == True :
                layerlist.append(nn.Dropout(p))
        else :
            for i in layers:
                layerlist.append(nn.Linear(n_in,i))
                if activation_function != None :
                    layerlist.append(activation_function)
                if enable_batch_normalization :
                    layerlist.append(nn.BatchNorm1d(i))
                if enable_dropout == True :
                    layerlist.append(nn.Dropout(p))
                n_in = i
            layerlist.append(nn.Linear(layers[-1],out_sz))

        self.layers = nn.Sequential(*layerlist)

    def forward(self, x_cont):
        if self.enable_batch_normalization :
            x_cont = self.bn_cont(x_cont)
        x = self.layers(x_cont)
        return x
```

# Part 2: Neural network without any hidden layer

## Loss function: MSE & optimizer: SGD

I got the best result with MSE loss function and SGD optimizer.

```
In [12]: torch.manual_seed(42)
         in_features = 68
         out_features = 2
         hidden_layers = []
         model = Model(in_features, out_features, hidden_layers, p=0.5)
         model
```

```
Out[12]: Model(
           (layers): Sequential(
             (0): Linear(in_features=68, out_features=2, bias=True)
             (1): ReLU(inplace=True)
           )
         )
```

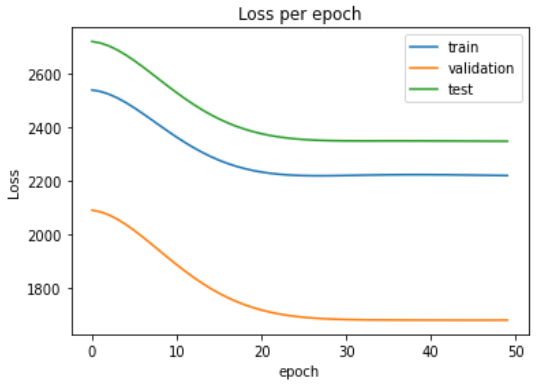
```
In [13]: criterion = nn.MSELoss()
         optimizer = torch.optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
```

### *Training for 50 epochs*

The performance from Linear Regression is better than this model.

```
In [14]: %%time
         epochs = 50
         training_losses,valid_losses,test_losses = train(model,criterion,optimizer,epochs)
```

```
Epoch:   1  Training Loss: 2539.95239258 Validation Loss: 2090.665771484375 Test Loss: 2721.414794921875
Epoch:   6  Training Loss: 2472.99877930 Validation Loss: 2014.5233154296875 Test Loss: 2649.400634765625
Epoch:  11  Training Loss: 2363.45410156 Validation Loss: 1886.6807861328125 Test Loss: 2529.57275390625
Epoch:  16  Training Loss: 2277.40405273 Validation Loss: 1780.03662109375 Test Loss: 2431.60888671875
Epoch:  21  Training Loss: 2233.26196289 Validation Loss: 1717.48193359375 Test Loss: 2376.513671875
Epoch:  26  Training Loss: 2220.07812500 Validation Loss: 1690.3802490234375 Test Loss: 2354.86279296875
Epoch:  31  Training Loss: 2220.53491211 Validation Loss: 1682.01318359375 Test Loss: 2349.869140625
Epoch:  36  Training Loss: 2223.04565430 Validation Loss: 1680.255859375 Test Loss: 2349.72412109375
Epoch:  41  Training Loss: 2223.42895508 Validation Loss: 1679.8043212890625 Test Loss: 2349.59521484375
Epoch:  46  Training Loss: 2221.88696289 Validation Loss: 1679.59326171875 Test Loss: 2348.9033203125
Epoch:  50  Training Loss: 2220.21948242 Validation Loss: 1679.782958984375 Test Loss: 2348.414306640625
CPU times: user 30.5 ms, sys: 4.26 ms, total: 34.7 ms
Wall time: 31.9 ms
```

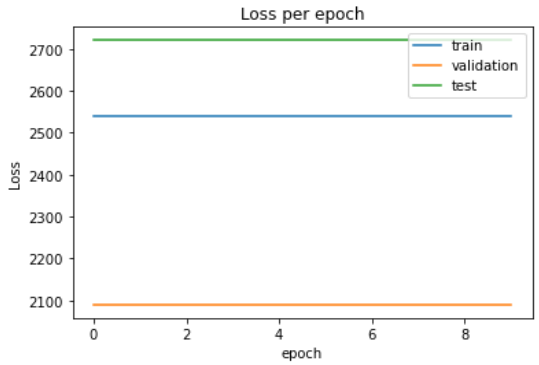


### *Training for 10 epochs*

The performance from Linear Regression is better than this model.

```
In [16]: epochs = 10
         model = Model(in_features, out_features, hidden_layers, p=0.5)
         training_losses,valid_losses,test_losses = train(model,criterion,optimizer,epochs)
```

```
Epoch:   1  Training Loss: 2539.49804688 Validation Loss: 2089.98681640625 Test Loss: 2720.573974609375
Epoch:   6  Training Loss: 2539.49804688 Validation Loss: 2089.98681640625 Test Loss: 2720.573974609375
Epoch:  10  Training Loss: 2539.49804688 Validation Loss: 2089.98681640625 Test Loss: 2720.573974609375
```



## Loss function: MSE & optimizer: Adam

In [18]:
```
torch.manual_seed(42)
in_features = 68
out_features = 2
hidden_layers = []
model = Model(in_features, out_features, hidden_layers, p=0.5)
model
```

Out[18]:
```
Model(
  (layers): Sequential(
    (0): Linear(in_features=68, out_features=2, bias=True)
    (1): ReLU(inplace=True)
  )
)
```

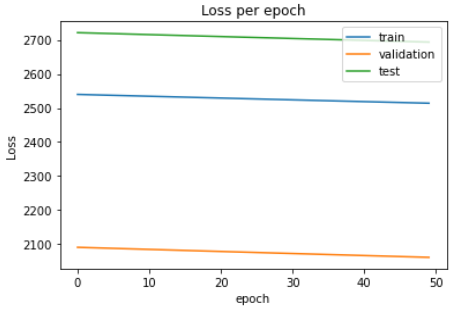
In [19]:
```
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

### *Training for 50 epochs*

The performance from Linear Regression is better than this model.

In [20]:
```
%%time
epochs = 50
training_losses,valid_losses,test_losses = train(model,criterion,optimizer,epochs)
```

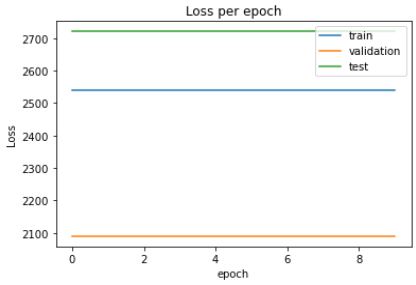
```
Epoch:   1  Training Loss: 2539.95239258 Validation Loss: 2090.665771484375 Test Loss: 2721.414794921875
Epoch:   6  Training Loss: 2537.25439453 Validation Loss: 2087.596435546875 Test Loss: 2718.52197265625
Epoch:  11  Training Loss: 2534.56469727 Validation Loss: 2084.53515625 Test Loss: 2715.641845703125
Epoch:  16  Training Loss: 2531.88916016 Validation Loss: 2081.48779296875 Test Loss: 2712.775634765625
Epoch:  21  Training Loss: 2529.22778320 Validation Loss: 2078.455322265625 Test Loss: 2709.923828125
Epoch:  26  Training Loss: 2526.58251953 Validation Loss: 2075.438232421875 Test Loss: 2707.087646484375
Epoch:  31  Training Loss: 2523.95239258 Validation Loss: 2072.437744140625 Test Loss: 2704.267333984375
Epoch:  36  Training Loss: 2521.33886719 Validation Loss: 2069.4541015625 Test Loss: 2701.46337890625
Epoch:  41  Training Loss: 2518.74194336 Validation Loss: 2066.487548828125 Test Loss: 2698.676513671875
Epoch:  46  Training Loss: 2516.16162109 Validation Loss: 2063.5380859375 Test Loss: 2695.90673828125
Epoch:  50  Training Loss: 2514.10986328 Validation Loss: 2061.191650390625 Test Loss: 2693.703125
CPU times: user 35.6 ms, sys: 3.3 ms, total: 38.9 ms
Wall time: 37.2 ms
```




### *Training for 10 epochs*

In [22]:
```
epochs = 10
model = Model(in_features, out_features, hidden_layers, p=0.5)
training_losses,valid_losses,test_losses = train(model,criterion,optimizer,epochs)
```

```
Epoch:   1  Training Loss: 2539.49804688 Validation Loss: 2089.98681640625 Test Loss: 2720.573974609375
Epoch:   6  Training Loss: 2539.49804688 Validation Loss: 2089.98681640625 Test Loss: 2720.573974609375
Epoch:  10  Training Loss: 2539.49804688 Validation Loss: 2089.98681640625 Test Loss: 2720.573974609375
```



## Loss function: MAE & optimizer: SGD

In [24]:
```
torch.manual_seed(42)
in_features = 68
out_features = 2
hidden_layers = []
model = Model(in_features, out_features, hidden_layers, p=0.5)
criterion = nn.L1Loss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
model
```

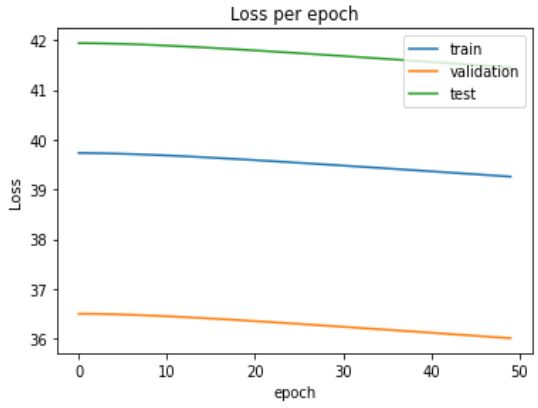
Out[24]:
```
Model(
  (layers): Sequential(
    (0): Linear(in_features=68, out_features=2, bias=True)
    (1): ReLU(inplace=True)
  )
)
```

### *Training for 50 epochs*

The performance from Linear Regression is better than this model.

In [25]:
```
%%time
epochs = 50
training_losses,valid_losses,test_losses = train(model,criterion,optimizer,epochs)
```

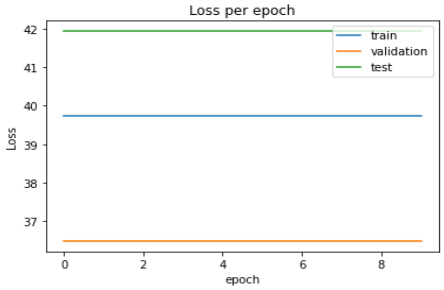
```
Epoch:   1  Training Loss: 39.73236465 Validation Loss: 36.50029754638672 Test Loss: 41.94256591796875
Epoch:   6  Training Loss: 39.71693039 Validation Loss: 36.484397888183594 Test Loss: 41.92665100097656
Epoch:  11  Training Loss: 39.68357849 Validation Loss: 36.44974899291992 Test Loss: 41.8923454284668
Epoch:  16  Training Loss: 39.63965225 Validation Loss: 36.40412902832031 Test Loss: 41.847164154052734
Epoch:  21  Training Loss: 39.58948517 Validation Loss: 36.35200881958008 Test Loss: 41.795562744140625
Epoch:  26  Training Loss: 39.53562927 Validation Loss: 36.29606246948242 Test Loss: 41.74016189575195
Epoch:  31  Training Loss: 39.47959900 Validation Loss: 36.23785400390625 Test Loss: 41.682533264160156
Epoch:  36  Training Loss: 39.42227936 Validation Loss: 36.178314208984375 Test Loss: 41.62357711791992
Epoch:  41  Training Loss: 39.36420059 Validation Loss: 36.11798095703125 Test Loss: 41.5638427734375
Epoch:  46  Training Loss: 39.30567551 Validation Loss: 36.05718231201172 Test Loss: 41.503639221191406
Epoch:  50  Training Loss: 39.25865555 Validation Loss: 36.00834274291992 Test Loss: 41.45527648925781
CPU times: user 31.2 ms, sys: 2.98 ms, total: 34.1 ms
Wall time: 32.5 ms
```



### *Training for 10 epochs*

In [27]:
```
epochs = 10
model = Model(in_features, out_features, hidden_layers, p=0.5)
training_losses,valid_losses,test_losses = train(model,criterion,optimizer,epochs)
```

```
Epoch:   1  Training Loss: 39.72853088 Validation Loss: 36.49102020263672 Test Loss: 41.93229293823242
Epoch:   6  Training Loss: 39.72853088 Validation Loss: 36.49102020263672 Test Loss: 41.93229293823242
Epoch:  10  Training Loss: 39.72853088 Validation Loss: 36.49102020263672 Test Loss: 41.93229293823242
```




## Loss function: MAE & optimizer: Adam

In [29]:
```
torch.manual_seed(42)
in_features = 68
out_features = 2
hidden_layers = []
model = Model(in_features, out_features, hidden_layers, p=0.5)
criterion = nn.L1Loss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
model
```

Out[29]:
```
Model(
  (layers): Sequential(
    (0): Linear(in_features=68, out_features=2, bias=True)
    (1): ReLU(inplace=True)
  )
)
```

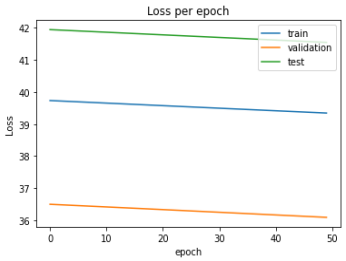
### *Training for 50 epochs*

The performance from Linear Regression is better than this model.

In [30]:
```
%%time
epochs = 50
training_losses,valid_losses,test_losses = train(model,criterion,optimizer,epochs)

show_diagrams(training_losses,valid_losses,test_losses)
```

```
Epoch:   1  Training Loss: 39.73236465 Validation Loss: 36.50029754638672 Test Loss: 41.94256591796875
Epoch:   6  Training Loss: 39.69264221 Validation Loss: 36.45890808105469 Test Loss: 41.90190887451172
Epoch:  11  Training Loss: 39.65286636 Validation Loss: 36.41733932495117 Test Loss: 41.86124038696289
Epoch:  16  Training Loss: 39.61309814 Validation Loss: 36.375770568847656 Test Loss: 41.82057189941406
Epoch:  21  Training Loss: 39.57332611 Validation Loss: 36.33420181274414 Test Loss: 41.7799072265625
Epoch:  26  Training Loss: 39.53355789 Validation Loss: 36.292625427246094 Test Loss: 41.7392463684082
Epoch:  31  Training Loss: 39.49378967 Validation Loss: 36.25106430053711 Test Loss: 41.698577880859375
Epoch:  36  Training Loss: 39.45402145 Validation Loss: 36.209495544433594 Test Loss: 41.65791320800781
Epoch:  41  Training Loss: 39.41425705 Validation Loss: 36.167930603027344 Test Loss: 41.617252349853516
Epoch:  46  Training Loss: 39.37449265 Validation Loss: 36.126365661621094 Test Loss: 41.57659149169922
Epoch:  50  Training Loss: 39.34267807 Validation Loss: 36.093116760253906 Test Loss: 41.544063568115234
```



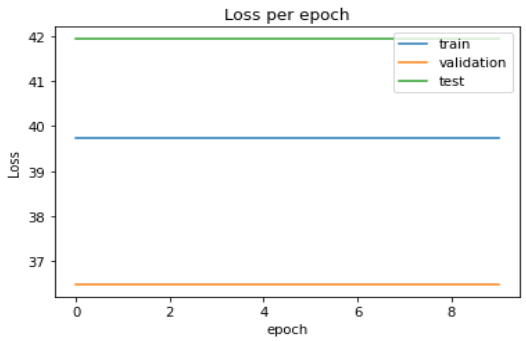
### *Training for 10 epochs*

The performance from Linear Regression is better than this model.

```
In [31]: epochs = 10
         model = Model(in_features, out_features, hidden_layers, p=0.5)
         training_losses,valid_losses,test_losses = train(model,criterion,optimizer,epochs)

         show_diagrams(training_losses,valid_losses,test_losses)
```

```
Epoch:   1  Training Loss: 39.72853088 Validation Loss: 36.49102020263672 Test Loss: 41.93229293823242
Epoch:   6  Training Loss: 39.72853088 Validation Loss: 36.49102020263672 Test Loss: 41.93229293823242
Epoch:  10  Training Loss: 39.72853088 Validation Loss: 36.49102020263672 Test Loss: 41.93229293823242
```



## Part 3

ReLU was chosen as the activation function because the range of other activation functions is not between (-180,180). As you can see, the results almost stayed the same with or without the activation function.

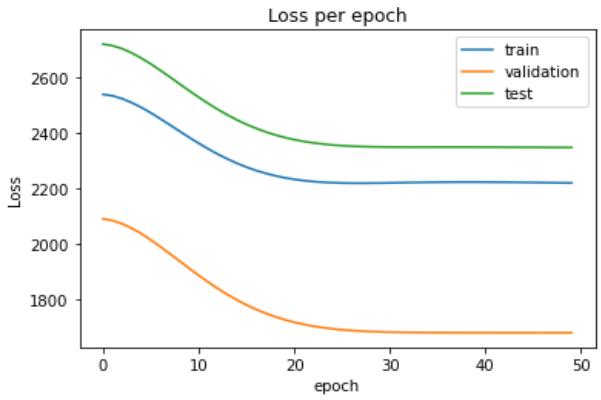
In [38]:
```
in_features = 68
out_features = 2
hidden_layers = []
model = Model(in_features, out_features, hidden_layers, p=0.5, activation_function = nn.ReLU(inplace = True))
criterion = nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
model
```

Out[38]:
```
Model(
  (layers): Sequential(
    (0): Linear(in_features=68, out_features=2, bias=True)
    (1): ReLU(inplace=True)
  )
)
```

### 50 Epochs

In [39]:
```
%%time
epochs = 50
training_losses,valid_losses,test_losses = train(model,criterion,optimizer,epochs)
```

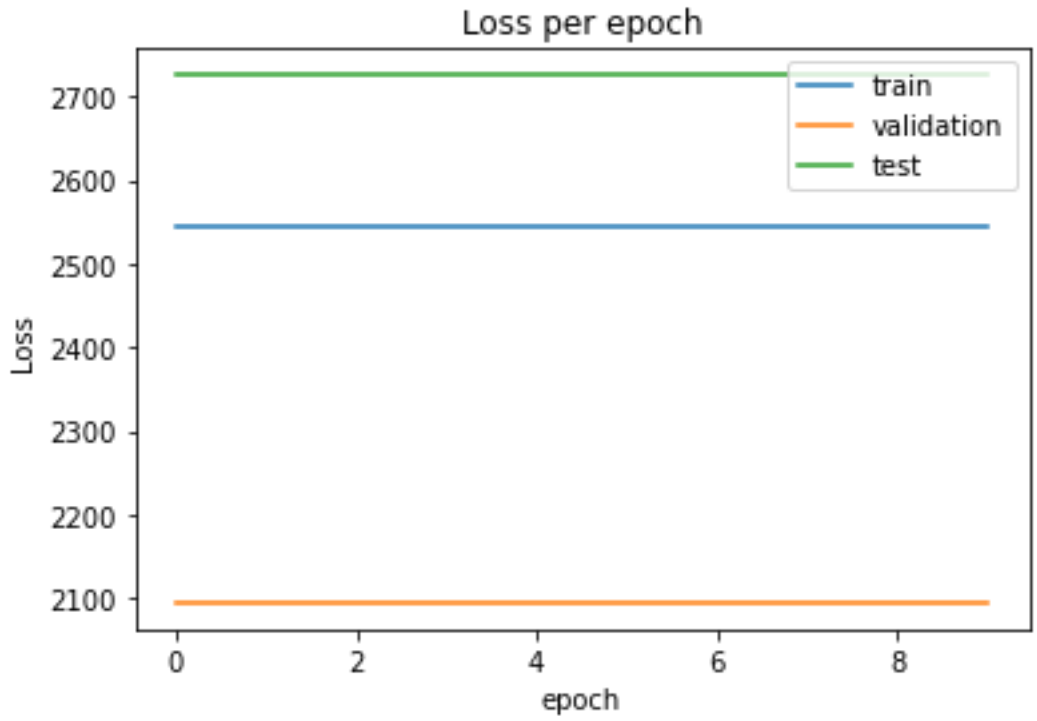
```
Epoch:   1  Training Loss: 2539.95239258 Validation Loss: 2090.665771484375 Test Loss: 2721.414794921875
Epoch:   6  Training Loss: 2472.99877930 Validation Loss: 2014.5233154296875 Test Loss: 2649.400634765625
Epoch:  11  Training Loss: 2363.45410156 Validation Loss: 1886.6807861328125 Test Loss: 2529.57275390625
Epoch:  16  Training Loss: 2277.40405273 Validation Loss: 1780.03662109375 Test Loss: 2431.60888671875
Epoch:  21  Training Loss: 2233.26196289 Validation Loss: 1717.48193359375 Test Loss: 2376.513671875
Epoch:  26  Training Loss: 2220.07812500 Validation Loss: 1690.3802490234375 Test Loss: 2354.86279296875
Epoch:  31  Training Loss: 2220.53491211 Validation Loss: 1682.01318359375 Test Loss: 2349.869140625
Epoch:  36  Training Loss: 2223.04565430 Validation Loss: 1680.255859375 Test Loss: 2349.72412109375
Epoch:  41  Training Loss: 2223.42895508 Validation Loss: 1679.8043212890625 Test Loss: 2349.59521484375
Epoch:  46  Training Loss: 2221.88696289 Validation Loss: 1679.59326171875 Test Loss: 2348.9033203125
Epoch:  50  Training Loss: 2220.21948242 Validation Loss: 1679.782958984375 Test Loss: 2348.414306640625
CPU times: user 29 ms, sys: 2.85 ms, total: 31.8 ms
Wall time: 30.3 ms
```



## 10 Epochs

In [45]:
```
%%time
model = Model(in_features, out_features, hidden_layers, p=0.5, activation_function = nn.ReLU(inplace = True))
epochs = 10
training_losses,valid_losses,test_losses = train(model,criterion,optimizer,epochs)
```

```
Epoch:   1  Training Loss: 2543.59814453 Validation Loss: 2094.75 Test Loss: 2725.2822265625
Epoch:   6  Training Loss: 2543.59814453 Validation Loss: 2094.75 Test Loss: 2725.2822265625
Epoch:  10  Training Loss: 2543.59814453 Validation Loss: 2094.75 Test Loss: 2725.2822265625
CPU times: user 8.09 ms, sys: 2.13 ms, total: 10.2 ms
Wall time: 8.5 ms
```



## Part 4: Trying different batch sizes

Different batch sizes did make the performance of the model much better. I did 3 experiments to prove this point. For example:

Batch Size = 8 and with batch normalization:

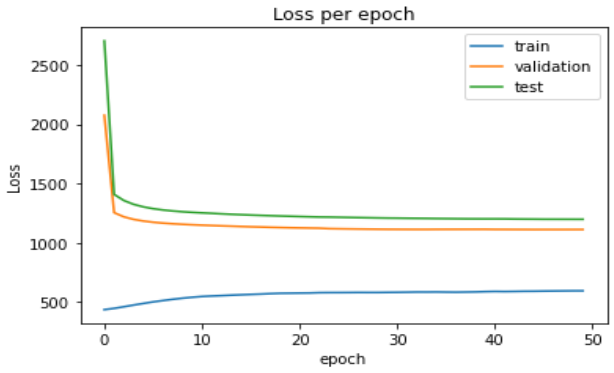
Epoch: 100 Training Loss: 14.30019569 Validation Loss: 24.45240592956543 Test Loss: 25.2082080841

### Experiment 1: batch_size = 4

In [150]:
```
model = Model(in_features, out_features, hidden_layers, p=0.5, activation_function = nn.ReLU(inplace = True))
criterion = nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
```

In [151]:
```
batch_size = 4
epochs = 50
training_losses,valid_losses,test_losses = mini_batch_train(epochs, batch_size,model,criterion,optimizer)
```

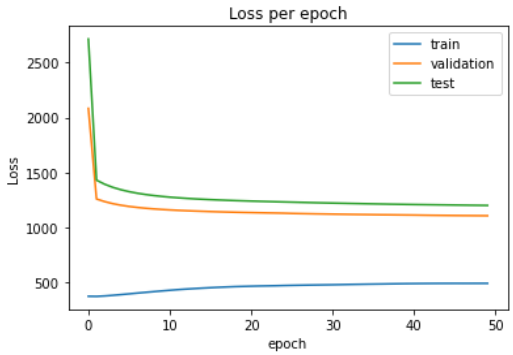
```
Epoch:   1  Training Loss: 427.58666992 Validation Loss: 2089.55908203125 Test Loss: 2718.659423828125
Epoch:   6  Training Loss: 505.98336792 Validation Loss: 1176.8876953125 Test Loss: 1290.311767578125
Epoch:  11  Training Loss: 548.95605469 Validation Loss: 1149.6524658203125 Test Loss: 1253.13916015625
Epoch:  16  Training Loss: 565.22882080 Validation Loss: 1135.522705078125 Test Loss: 1234.542724609375
Epoch:  21  Training Loss: 575.98181152 Validation Loss: 1125.8265380859375 Test Loss: 1221.8487548828125
Epoch:  26  Training Loss: 580.68676758 Validation Loss: 1117.365966796875 Test Loss: 1214.1043701171875
Epoch:  31  Training Loss: 582.00805664 Validation Loss: 1113.532958984375 Test Loss: 1208.17333984375
Epoch:  36  Training Loss: 583.61663818 Validation Loss: 1113.613037109375 Test Loss: 1204.030029296875
Epoch:  41  Training Loss: 589.13903809 Validation Loss: 1112.99462890625 Test Loss: 1202.0809326171875
Epoch:  46  Training Loss: 591.75384521 Validation Loss: 1112.2025146484375 Test Loss: 1199.3067626953125
Epoch: 50  Training Loss: 594.02160645 Validation Loss: 1112.171630859375 Test Loss: 1198.392333984375
```




### Experiment 2: batch_size = 8

In [142]:
```
model = Model(in_features, out_features, hidden_layers, p=0.5, activation_function = nn.ReLU(inplace = True))
criterion = nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
batch_size = 8
epochs = 50
training_losses,valid_losses,test_losses = mini_batch_train(epochs, batch_size,model,criterion,optimizer)
```

```
Epoch:   1  Training Loss: 372.92962646 Validation Loss: 2083.39453125 Test Loss: 2714.92822265625
Epoch:   6  Training Loss: 394.96588135 Validation Loss: 1190.5777587890625 Test Loss: 1325.141845703125
Epoch:  11  Training Loss: 427.66928101 Validation Loss: 1158.982666015625 Test Loss: 1275.1922607421875
Epoch:  16  Training Loss: 451.09417725 Validation Loss: 1143.9720458984375 Test Loss: 1252.5242919921875
Epoch:  21  Training Loss: 464.68304443 Validation Loss: 1134.73779296875 Test Loss: 1239.4515380859375
Epoch:  26  Training Loss: 472.27993774 Validation Loss: 1126.9888916015625 Test Loss: 1229.7587890625
Epoch:  31  Training Loss: 477.84628296 Validation Loss: 1120.70751953125 Test Loss: 1221.22509765625
Epoch:  36  Training Loss: 483.86517334 Validation Loss: 1115.75390625 Test Loss: 1213.975341796875
Epoch:  41  Training Loss: 488.56750488 Validation Loss: 1112.1705322265625 Test Loss: 1208.0264892578125
Epoch:  46  Training Loss: 490.40365601 Validation Loss: 1108.03125 Test Loss: 1203.39697265625
Epoch: 50  Training Loss: 490.58447266 Validation Loss: 1106.0943603515625 Test Loss: 1199.94189453125
```

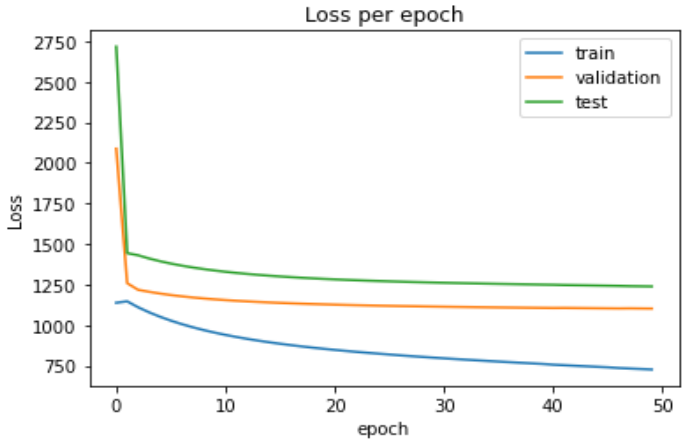



## Experiment 3: batch_size = 16

In [155]:
```
model = Model(in_features, out_features, hidden_layers, p=0.5, activation_function = nn.ReLU(inplace = True))
criterion = nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
batch_size = 16
epochs = 50
```

In [156]:
```
training_losses,valid_losses,test_losses = mini_batch_train(epochs, batch_size,model,criterion,optimizer)
```

```
Epoch:   1  Training Loss: 1137.62475586 Validation Loss: 2087.594482421875 Test Loss: 2716.100830078125
Epoch:   6  Training Loss: 1027.18579102 Validation Loss: 1185.0115966796875 Test Loss: 1378.986572265625
Epoch:  11  Training Loss: 939.36700439 Validation Loss: 1153.867431640625 Test Loss: 1328.41845703125
Epoch:  16  Training Loss: 884.70690918 Validation Loss: 1136.6549072265625 Test Loss: 1299.613525390625
Epoch:  21  Training Loss: 846.92059326 Validation Loss: 1125.843017578125 Test Loss: 1281.531494140625
Epoch:  26  Training Loss: 818.51379395 Validation Loss: 1118.2628173828125 Test Loss: 1269.41455078125
Epoch:  31  Training Loss: 794.95281982 Validation Loss: 1112.9783935546875 Test Loss: 1260.6048583984375
Epoch:  36  Training Loss: 774.90289307 Validation Loss: 1108.6722412109375 Test Loss: 1253.927978515625
Epoch:  41  Training Loss: 755.68096924 Validation Loss: 1105.123046875 Test Loss: 1248.3525390625
Epoch:  46  Training Loss: 739.53369141 Validation Loss: 1103.008056640625 Test Loss: 1242.53662109375
Epoch: 50  Training Loss: 726.27789307 Validation Loss: 1101.7298583984375 Test Loss: 1237.94091796875
```



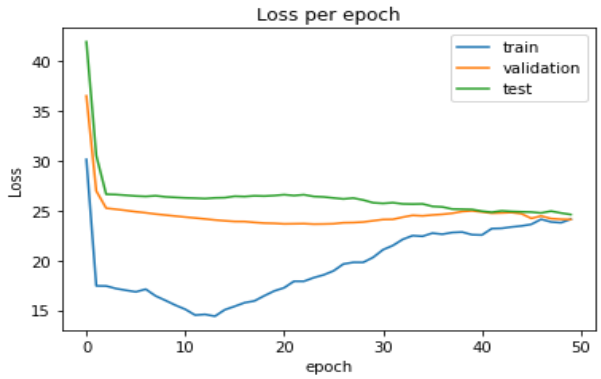
# Part 5: Neural Network with 1 or 2 hidden layers

## One hidden layer

In [263]:
```
hidden_layers = [128]
model = Model(in_features, out_features, hidden_layers, p=0.4, activation_function = nn.ReLU(inplace=True))
criterion = nn.L1Loss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
batch_size = 4
epochs = 50
```

In [264]:
```
training_losses,valid_losses,test_losses = mini_batch_train(epochs, batch_size,model,criterion,optimizer)
```

```
Epoch:   1  Training Loss: 30.14392090 Validation Loss: 36.52543258666992 Test Loss: 41.9664421081543
Epoch:   6  Training Loss: 16.83256531 Validation Loss: 24.884592056274414 Test Loss: 26.46668243408203
Epoch:  11  Training Loss: 15.08112907 Validation Loss: 24.34933853149414 Test Loss: 26.275875091552734
Epoch:  16  Training Loss: 15.37900925 Validation Loss: 23.90640640258789 Test Loss: 26.44235610961914
Epoch:  21  Training Loss: 17.24248505 Validation Loss: 23.663835525512695 Test Loss: 26.598873138427734
Epoch:  26  Training Loss: 18.94375420 Validation Loss: 23.677976608276367 Test Loss: 26.268030166625977
Epoch:  31  Training Loss: 21.06715775 Validation Loss: 24.10591697692871 Test Loss: 25.723140716552734
Epoch:  36  Training Loss: 22.73626328 Validation Loss: 24.559385299682617 Test Loss: 25.42424964904785
Epoch:  41  Training Loss: 22.54436302 Validation Loss: 24.864803314208984 Test Loss: 24.944486618041992
Epoch:  46  Training Loss: 23.61811447 Validation Loss: 24.21803855895996 Test Loss: 24.85001564025879
Epoch: 50  Training Loss: 24.12952042 Validation Loss: 24.108455657958984 Test Loss: 24.60411834716797
```

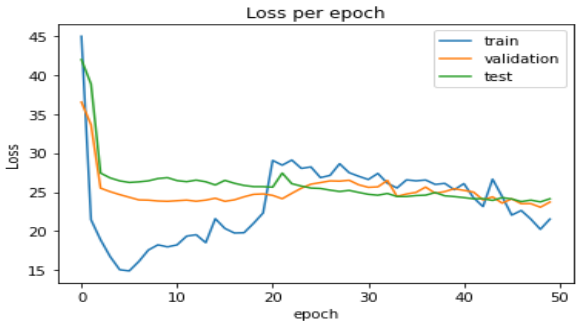


## Two hidden layers

```
In [240]: hidden_layers = [128,64]
model = Model(in_features, out_features, hidden_layers, p=0.4, activation_function = nn.ReLU(inplace=True))
criterion = nn.L1Loss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
batch_size = 4
epochs = 50
```

```
In [241]: training_losses,valid_losses,test_losses = mini_batch_train(epochs, batch_size,model,criterion,optimizer)
show_diagrams(training_losses,valid_losses,test_losses)
```

```
Epoch:   1  Training Loss: 44.93004608 Validation Loss: 36.51309585571289 Test Loss: 41.948997497558594
Epoch:   6  Training Loss: 14.92746639 Validation Loss: 24.345020294189453 Test Loss: 26.23956871032715
Epoch:  11  Training Loss: 18.25562668 Validation Loss: 23.905046463012695 Test Loss: 26.474849700927734
Epoch:  16  Training Loss: 20.36013603 Validation Loss: 23.824430465698242 Test Loss: 26.506689071655273
Epoch:  21  Training Loss: 29.05882263 Validation Loss: 24.586156845092773 Test Loss: 25.635631561279297
Epoch:  26  Training Loss: 26.85707474 Validation Loss: 26.22180938720703 Test Loss: 25.470003128051758
Epoch:  31  Training Loss: 26.61475754 Validation Loss: 25.614816665649414 Test Loss: 24.72467613220215
Epoch:  36  Training Loss: 26.44740677 Validation Loss: 24.970491409301758 Test Loss: 24.565185546875
Epoch:  41  Training Loss: 26.09021187 Validation Loss: 25.226253509521484 Test Loss: 24.300188064575195
Epoch:  46  Training Loss: 22.06313324 Validation Loss: 24.140180587768555 Test Loss: 24.13030433654785
Epoch: 50  Training Loss: 21.56114197 Validation Loss: 23.735363006591797 Test Loss: 24.15113067626953
```




## The winner model

As you can see, this model with 128 and 64 neurons in hidden layers works better because the MAE and MSE is better. And also the MAE of this part is better than previous parts.

This Model:

```
Epoch: 50  Training Loss: 19.60 Validation Loss: 24.79 Test Loss: 24.29659843
```

Previous Models :

```
Epoch:  50  Training Loss: 39.25 Validation Loss: 36.0083 Test Loss: 41.45527
```

The winner model parameters:

- 2 hidden layers with 128 and 64 neurons respectively.
- Activation Function : ReLU
- Optimizer : SGD
- Batch Size : 4
- Learning Rate : 0.001

## How to choose hidden layer neurons?

Deciding the number of neurons in the hidden layers is a very important part of deciding your overall neural network architecture.

- The number of hidden neurons should be between the size of the input layer and the size of the output layer.
- The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer.
- The number of hidden neurons should be less than twice the size of the input layer.

# Part 6: Add Dropout Layer

## Adding dropout layer to the model without any hidden layer

The training/validation/test loss got better with Dropout layer.

**Neural Network without hidden layer (With Dropout) :**

```
Epoch:  50  Training Loss: 24.85396385 Validation Loss: 24.982773 Test Loss: 27.90833282470703
```

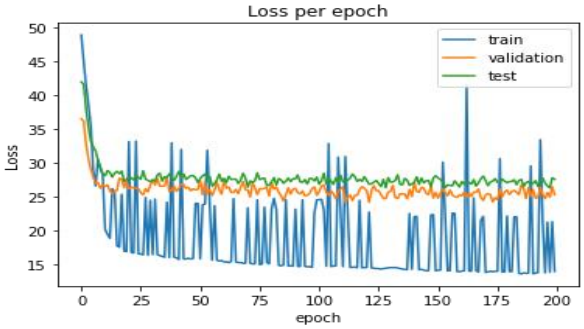
**Neural Network without hidden layer (Without Dropout) :**

```
Epoch: 50 Training Loss: 39.25 Validation Loss: 36.0082 Test Loss: 41.455276489257
```

In [280]:
```
hidden_layers = []
model = Model(in_features, out_features, hidden_layers, p=0.1, activation_function = nn.ReLU(inplace=True),
              enable_dropout = True)
criterion = nn.L1Loss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
```

In [281]:
```
batch_size = 4
epochs = 200
training_losses,valid_losses,test_losses = mini_batch_train(epochs, batch_size,model,criterion,optimizer)
```

```
Epoch:   1  Training Loss: 48.91851044 Validation Loss: 36.55374526977539 Test Loss: 41.99649429321289
Epoch:   6  Training Loss: 29.00000954 Validation Loss: 27.95040512084961 Test Loss: 32.39714431762695
Epoch:  11  Training Loss: 20.17964172 Validation Loss: 26.535470962524414 Test Loss: 28.122438430786133
Epoch:  16  Training Loss: 17.78153610 Validation Loss: 25.981098175048828 Test Loss: 28.77781105041504
Epoch:  21  Training Loss: 33.17590332 Validation Loss: 26.1761531829834 Test Loss: 27.93320083618164
Epoch:  26  Training Loss: 16.58538437 Validation Loss: 25.626785278320312 Test Loss: 27.61595344543457
Epoch:  31  Training Loss: 16.50260735 Validation Loss: 27.602161407470703 Test Loss: 27.487037658691406
Epoch:  36  Training Loss: 16.08199692 Validation Loss: 26.613645553588867 Test Loss: 28.00016212463379
Epoch:  41  Training Loss: 16.03327942 Validation Loss: 25.81369972229004 Test Loss: 28.015548706054688
Epoch:  46  Training Loss: 15.94164371 Validation Loss: 26.429584503173828 Test Loss: 27.85466957092285
Epoch:  51  Training Loss: 15.84828758 Validation Loss: 27.44176483154297 Test Loss: 27.720304489135742
Epoch:  56  Training Loss: 15.65758514 Validation Loss: 25.20218849182129 Test Loss: 26.5815486907959
Epoch:  61  Training Loss: 15.40086746 Validation Loss: 27.587690353393555 Test Loss: 27.758319854736328
Epoch:  66  Training Loss: 15.40049934 Validation Loss: 26.116243362426758 Test Loss: 27.626998901367188
Epoch:  71  Training Loss: 23.38786316 Validation Loss: 26.015369415283203 Test Loss: 26.451082229614258
Epoch:  76  Training Loss: 14.99820232 Validation Loss: 26.236469268798828 Test Loss: 27.564205169677734
Epoch:  81  Training Loss: 23.23807335 Validation Loss: 27.148481369018555 Test Loss: 27.22913360595703
Epoch:  86  Training Loss: 14.94363499 Validation Loss: 24.415102005004883 Test Loss: 27.009679794311523
Epoch:  91  Training Loss: 23.18753052 Validation Loss: 26.27229881286621 Test Loss: 27.65817642211914
Epoch:  96  Training Loss: 14.69270039 Validation Loss: 25.230039596557617 Test Loss: 26.912858963012695
Epoch: 101  Training Loss: 24.58831024 Validation Loss: 25.75014305114746 Test Loss: 26.883270263671875
Epoch: 106  Training Loss: 14.72532082 Validation Loss: 25.908414840698242 Test Loss: 27.29370880126953
Epoch: 111  Training Loss: 14.71242523 Validation Loss: 26.5212459564209 Test Loss: 27.193021774291992
Epoch: 116  Training Loss: 14.65519810 Validation Loss: 25.2363224029541 Test Loss: 28.01066780090332
Epoch: 121  Training Loss: 14.57679462 Validation Loss: 25.95857810974121 Test Loss: 27.436643600463867
Epoch: 126  Training Loss: 14.41927433 Validation Loss: 25.9408016204834 Test Loss: 26.98366928100586
Epoch: 131  Training Loss: 14.56833076 Validation Loss: 25.818777084350586 Test Loss: 27.193580627441406
Epoch: 136  Training Loss: 14.34829903 Validation Loss: 25.430875778198242 Test Loss: 26.788175582885742
Epoch: 141  Training Loss: 22.06928062 Validation Loss: 27.168508529663086 Test Loss: 27.150175094604492
Epoch: 146  Training Loss: 14.07982731 Validation Loss: 25.523914337158203 Test Loss: 27.37993812561035
Epoch: 151  Training Loss: 14.18644714 Validation Loss: 24.820507049560547 Test Loss: 27.806318283081055
Epoch: 156  Training Loss: 14.11717510 Validation Loss: 26.010047912597656 Test Loss: 26.716808319091797
Epoch: 161  Training Loss: 14.06408119 Validation Loss: 25.16026496887207 Test Loss: 27.2659912109375
Epoch: 166  Training Loss: 24.84433174 Validation Loss: 25.7161865234375 Test Loss: 27.564834594726562
Epoch: 171  Training Loss: 13.84403324 Validation Loss: 25.27330780029297 Test Loss: 27.407472610473633
Epoch: 176  Training Loss: 14.01500893 Validation Loss: 24.578033447265625 Test Loss: 27.146751403808594
Epoch: 181  Training Loss: 13.98437023 Validation Loss: 25.360231399536133 Test Loss: 27.28648567199707
Epoch: 186  Training Loss: 13.63954067 Validation Loss: 26.7778263092041 Test Loss: 27.594186782836914
Epoch: 191  Training Loss: 13.64735317 Validation Loss: 26.058271408081055 Test Loss: 27.022550582885742
Epoch: 196  Training Loss: 13.77493668 Validation Loss: 24.7509708404541 Test Loss: 27.028215408325195
Epoch: 200  Training Loss: 14.00575352 Validation Loss: 25.327442169189453 Test Loss: 27.601734161376953
```


Loss per epoch
train
validation
test
Loss
epoch
50
45
40
35
30
25
20
15
0
25
50
75
100
125
150
175
200

## Adding dropout layer to the model with one hidden layer

The training/validation/test loss got better with Dropout layer.

Neural Network with one hidden layer (With Dropout) :

**Neural Network with one hidden layer (With Dropout) :**

```
Epoch: 200  Training Loss: 20.76536369 Validation Loss: 23.60884284973
1445 Test Loss: 23.40304946899414
```

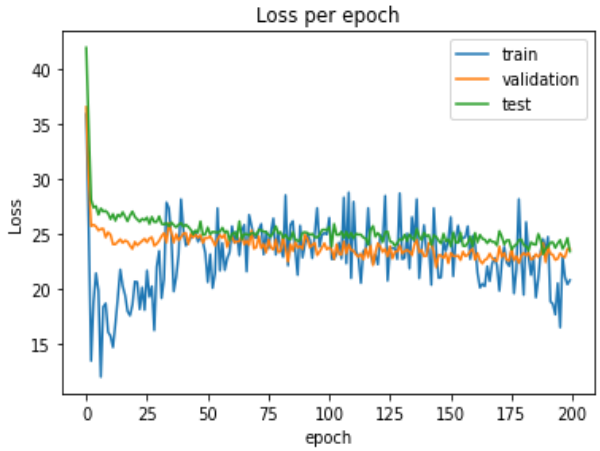
**Neural Network with one hidden layer (Without Dropout) :**

```
Epoch: 50  Training Loss: 24.12952042 Validation Loss: 24.108455657958
984 Test Loss: 24.60411834716797
```

In [277]:
```
hidden_layers = [128]
model = Model(in_features, out_features, hidden_layers, p=0.4,
              activation_function = nn.ReLU(inplace=True),enable_dropout= True)
```

In [278]:
```
%%time
criterion = nn.L1Loss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
batch_size = 4
epochs = 200
training_losses,valid_losses,test_losses = mini_batch_train(epochs, batch_size,model,criterion,optimizer)
```

```
Epoch:   1  Training Loss: 35.93862915 Validation Loss: 36.58230209350586 Test Loss: 42.014163970947266
Epoch:   6  Training Loss: 19.84797096 Validation Loss: 25.3197078704834 Test Loss: 26.75871467590332
Epoch:  11  Training Loss: 15.65837860 Validation Loss: 24.922916412353516 Test Loss: 26.353174209594727
Epoch:  16  Training Loss: 20.23684311 Validation Loss: 24.187549591064453 Test Loss: 26.995010375976562
Epoch:  21  Training Loss: 20.63908768 Validation Loss: 24.271818161010742 Test Loss: 26.56427574157715
Epoch:  26  Training Loss: 21.65841293 Validation Loss: 24.238801956176758 Test Loss: 26.508209228515625
Epoch:  31  Training Loss: 23.44415092 Validation Loss: 24.38168716430664 Test Loss: 26.60601234436035
Epoch:  36  Training Loss: 24.78095818 Validation Loss: 25.14406394958496 Test Loss: 26.06496238708496
Epoch:  41  Training Loss: 25.27487564 Validation Loss: 25.044992446899414 Test Loss: 25.645162582397461
Epoch:  46  Training Loss: 25.14847946 Validation Loss: 24.915300369262695 Test Loss: 25.322507858276367
Epoch:  51  Training Loss: 20.59662437 Validation Loss: 24.61802101135254 Test Loss: 25.588533401489258
Epoch:  56  Training Loss: 21.65007782 Validation Loss: 24.890674591064453 Test Loss: 25.189725875854492
Epoch:  61  Training Loss: 25.69818306 Validation Loss: 24.71715545654297 Test Loss: 24.993642807006836
Epoch:  66  Training Loss: 25.82472610 Validation Loss: 24.035869598388672 Test Loss: 24.97555923461914
Epoch:  71  Training Loss: 24.62645340 Validation Loss: 23.784961700439453 Test Loss: 24.863924026489258
Epoch:  76  Training Loss: 23.65763474 Validation Loss: 24.469436645507812 Test Loss: 24.904653549194336
Epoch:  81  Training Loss: 23.65078735 Validation Loss: 23.45296859741211 Test Loss: 25.153284072875977
Epoch:  86  Training Loss: 26.17167664 Validation Loss: 23.453081130981445 Test Loss: 25.171171188354492
Epoch:  91  Training Loss: 24.08959198 Validation Loss: 24.90431022644043 Test Loss: 25.15203285217285
Epoch:  96  Training Loss: 27.36040115 Validation Loss: 23.06416893005371 Test Loss: 24.67131805419922
Epoch: 101  Training Loss: 26.47813416 Validation Loss: 23.771381378173828 Test Loss: 25.629823684692383
Epoch: 106  Training Loss: 22.73594856 Validation Loss: 23.090112686157227 Test Loss: 25.20063591003418
Epoch: 111  Training Loss: 27.93334579 Validation Loss: 23.4656925201416 Test Loss: 24.599332809448242
Epoch: 116  Training Loss: 23.53633118 Validation Loss: 23.65341567993164 Test Loss: 24.579978942871094
Epoch: 121  Training Loss: 22.22357178 Validation Loss: 24.14432716369629 Test Loss: 25.084861755371094
Epoch: 126  Training Loss: 23.74415970 Validation Loss: 23.87592315673828 Test Loss: 24.30201530456543
Epoch: 131  Training Loss: 22.70380402 Validation Loss: 23.203033447265625 Test Loss: 24.44444465637207
Epoch: 136  Training Loss: 24.29734421 Validation Loss: 23.994916915893555 Test Loss: 24.478443145751953
Epoch: 141  Training Loss: 23.29875565 Validation Loss: 23.00520896911621 Test Loss: 24.779600143432617
Epoch: 146  Training Loss: 20.97754669 Validation Loss: 23.055997848510742 Test Loss: 24.392404556274414
Epoch: 151  Training Loss: 26.52276230 Validation Loss: 23.254549026489258 Test Loss: 23.84941291809082
Epoch: 156  Training Loss: 24.65549850 Validation Loss: 23.368194580078125 Test Loss: 23.919918060302734
Epoch: 161  Training Loss: 24.00524712 Validation Loss: 22.633182525634766 Test Loss: 25.09300994873047
Epoch: 166  Training Loss: 22.03892326 Validation Loss: 22.74378204345703 Test Loss: 24.145103454589844
Epoch: 171  Training Loss: 19.80419540 Validation Loss: 23.8389892578125 Test Loss: 25.266706466674805
Epoch: 176  Training Loss: 23.60344315 Validation Loss: 23.638216018676758 Test Loss: 23.584327697753906
Epoch: 181  Training Loss: 19.43172264 Validation Loss: 23.256284713745117 Test Loss: 23.993833541870117
Epoch: 186  Training Loss: 19.19564247 Validation Loss: 23.14932632446289 Test Loss: 24.008541107177734
Epoch: 191  Training Loss: 24.72771835 Validation Loss: 23.82563018798828 Test Loss: 23.814414978027344
```


Loss per epoch
train
validation
test
Loss
epoch
40
35
30
25
20
15
0
25
50
75
100
125
150
175
200

## Adding dropout layer to the model with two hidden layer

The training/validation/test loss didn't change much with Dropout layer.

**Neural Network with two hidden layer (With Dropout) :**

```
Epoch: 200  Training Loss: 8.80532551 Validation Loss: 21.468709945678
71 Test Loss: 24.55977439880371
```

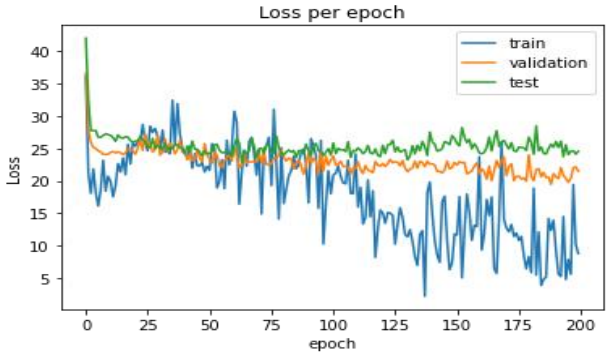
**Neural Network with two hidden layer (Without Dropout) :**

```
Epoch: 50  Training Loss: 21.56114197 Validation Loss: 23.735363006591
797 Test Loss: 24.15113067626953
```

In [289]:
```
hidden_layers = [128,64]
model = Model(in_features, out_features, hidden_layers, p=0.1,
            activation_function = nn.ReLU(inplace=True),enable_dropout= True)
```

In [290]:
```
%%time
criterion = nn.L1Loss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
batch_size = 4
epochs = 200
training_losses,valid_losses,test_losses = mini_batch_train(epochs, batch_size,model,criterion,optimizer)
```

```
Epoch:   1  Training Loss: 36.05513763 Validation Loss: 36.544559478759766 Test Loss: 41.977840423583984
Epoch:   6  Training Loss: 16.13249588 Validation Loss: 24.628419876098633 Test Loss: 26.716459274291992
Epoch:  11  Training Loss: 19.86447334 Validation Loss: 24.418012619018555 Test Loss: 27.00939178466797
Epoch:  16  Training Loss: 23.51426697 Validation Loss: 24.027175903320312 Test Loss: 26.552614212036133
Epoch:  21  Training Loss: 25.33156776 Validation Loss: 24.0824031829834 Test Loss: 26.102645874023438
Epoch:  26  Training Loss: 24.60921097 Validation Loss: 25.469295501708984 Test Loss: 25.13011932373047
Epoch:  31  Training Loss: 24.69830513 Validation Loss: 23.946611404418945 Test Loss: 25.06675910949707
Epoch:  36  Training Loss: 32.42048264 Validation Loss: 25.299732208251953 Test Loss: 24.9273681640625
Epoch:  41  Training Loss: 24.16457176 Validation Loss: 24.6064510345459 Test Loss: 24.201364517211914
Epoch:  46  Training Loss: 22.35148621 Validation Loss: 24.778512954711914 Test Loss: 24.8524112701416
Epoch:  51  Training Loss: 21.62723923 Validation Loss: 23.363582611083984 Test Loss: 24.196895599365234
Epoch:  56  Training Loss: 23.22577667 Validation Loss: 22.14393424987793 Test Loss: 24.323637008666992
Epoch:  61  Training Loss: 30.71269226 Validation Loss: 25.2020320892334 Test Loss: 25.532175064086914
Epoch:  66  Training Loss: 22.25423241 Validation Loss: 22.77566146850586 Test Loss: 24.20600700378418
Epoch:  71  Training Loss: 24.77413177 Validation Loss: 24.281808853149414 Test Loss: 24.94186782836914
Epoch:  76  Training Loss: 19.27120972 Validation Loss: 22.558137893676758 Test Loss: 24.37480354309082
Epoch:  81  Training Loss: 16.51311874 Validation Loss: 23.67848014831543 Test Loss: 23.983938217163086
Epoch:  86  Training Loss: 21.85071945 Validation Loss: 22.7353572845459 Test Loss: 23.585485458374023
Epoch:  91  Training Loss: 16.61043930 Validation Loss: 24.279996871948242 Test Loss: 24.62078094482422
Epoch:  96  Training Loss: 26.20967102 Validation Loss: 24.538633346557617 Test Loss: 24.51776885986328
Epoch: 101  Training Loss: 20.88981247 Validation Loss: 23.811182022094727 Test Loss: 25.839509963989258
Epoch: 106  Training Loss: 19.53818512 Validation Loss: 22.240697860717773 Test Loss: 24.494585037231445
Epoch: 111  Training Loss: 15.94932652 Validation Loss: 22.92808723449707 Test Loss: 25.077898025512695
Epoch: 116  Training Loss: 14.44609928 Validation Loss: 21.5162296295166 Test Loss: 24.282947540283203
Epoch: 121  Training Loss: 15.02576828 Validation Loss: 22.66530990600586 Test Loss: 25.454570770263672
Epoch: 126  Training Loss: 10.25166321 Validation Loss: 22.972911834716797 Test Loss: 24.470129013061523
Epoch: 131  Training Loss: 11.38549709 Validation Loss: 21.407623291015625 Test Loss: 23.221961975097656
Epoch: 136  Training Loss: 11.71772194 Validation Loss: 23.0086727142334 Test Loss: 24.634403228759766
Epoch: 141  Training Loss: 14.94503498 Validation Loss: 22.651138305664062 Test Loss: 25.974308013916016
Epoch: 146  Training Loss: 17.64225769 Validation Loss: 21.68389129638672 Test Loss: 26.021772384643555
Epoch: 151  Training Loss: 11.65345955 Validation Loss: 22.43124008178711 Test Loss: 25.762910842895508
Epoch: 156  Training Loss: 15.23133850 Validation Loss: 21.300098419189453 Test Loss: 26.38555145263672
Epoch: 161  Training Loss: 9.33214951 Validation Loss: 21.84768295288086 Test Loss: 24.145343780517578
Epoch: 166  Training Loss: 6.51946354 Validation Loss: 21.576528549194336 Test Loss: 26.33431053161621
Epoch: 171  Training Loss: 12.72805786 Validation Loss: 20.804481506347656 Test Loss: 25.484773635864258
Epoch: 176  Training Loss: 10.81139183 Validation Loss: 20.7154541015625 Test Loss: 25.19581413269043
Epoch: 181  Training Loss: 5.85311890 Validation Loss: 20.688663482666016 Test Loss: 25.925907135009766
Epoch: 186  Training Loss: 4.82625294 Validation Loss: 21.302797317504883 Test Loss: 24.234458923339844
Epoch: 191  Training Loss: 9.20905590 Validation Loss: 20.603166580200195 Test Loss: 26.03782844543457
Epoch: 196  Training Loss: 7.88236904 Validation Loss: 19.759687423706055 Test Loss: 24.048927307128906
Epoch: 200  Training Loss: 8.80532551 Validation Loss: 21.46870994567871 Test Loss: 24.55977439880371
CPU times: user 19.5 s, sys: 149 ms, total: 19.6 s
Wall time: 19.6 s
```




## Part 7: Batch Normalization

The performance of the model with batch normalization is slightly better. And I also tried different batch sizes for each model. The results with batch_size = 8 was better. So batch size does make a difference.

For example, in the model without any hidden layer the performance with **different batch sizes** were **different**.

**Batch Size = 4 :**

```
Epoch: 100  Training Loss: 16.72462845 Validation Loss: 24.996891021728516 Test Loss: 26.932558059692383
```

**Batch Size = 8 :**

```
Epoch: 100  Training Loss: 14.30019569 Validation Loss: 24.45240592956543 Test Loss: 25.208208084106445
```

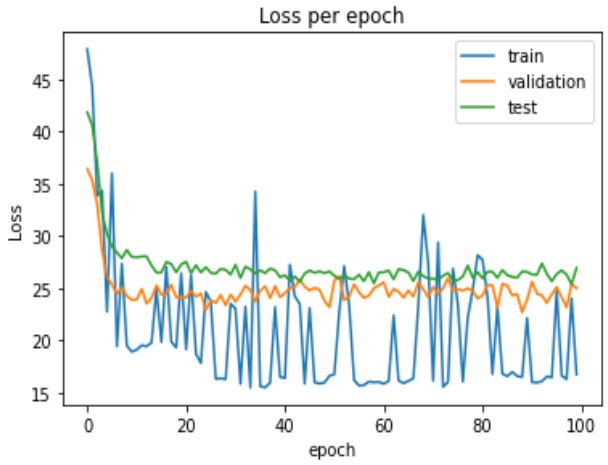
## 1) Zero hidden layer

In [303]:
```
hidden_layers = []
model = Model(in_features, out_features, hidden_layers, p=0.1, activation_function = nn.ReLU(inplace=True),
              enable_dropout = True, enable_batch_normalization = True)
criterion = nn.L1Loss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
```

### *Batch Size = 4*

In [304]:
```
%%time
batch_size = 4
epochs = 100
training_losses,valid_losses,test_losses = mini_batch_train(epochs, batch_size,model,criterion,optimizer)
```

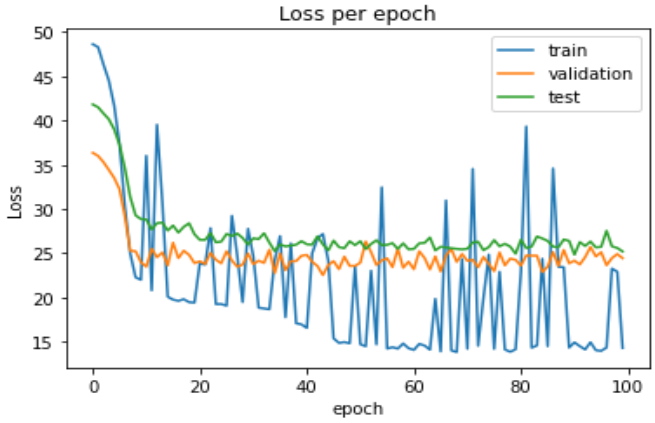
```
Epoch:   1  Training Loss: 47.89925385 Validation Loss: 36.41914367675781 Test Loss: 41.84770965576172
Epoch:   6  Training Loss: 35.99623871 Validation Loss: 25.330913543701172 Test Loss: 28.9970703125
Epoch:  11  Training Loss: 19.09636879 Validation Loss: 23.897125244140625 Test Loss: 27.95857810974121
Epoch:  16  Training Loss: 19.83423996 Validation Loss: 24.370895385742188 Test Loss: 26.506258010864258
Epoch:  21  Training Loss: 19.12012100 Validation Loss: 24.071006774902344 Test Loss: 27.53255844116211
Epoch:  26  Training Loss: 23.70106888 Validation Loss: 23.84262466430664 Test Loss: 26.473669052124023
Epoch:  31  Training Loss: 23.03184128 Validation Loss: 23.698627471923828 Test Loss: 27.24036979675293
Epoch:  36  Training Loss: 15.59183121 Validation Loss: 24.758501052856445 Test Loss: 26.712799072265625
Epoch:  41  Training Loss: 16.38268280 Validation Loss: 24.520681381225586 Test Loss: 26.229124069213867
Epoch:  46  Training Loss: 23.09301376 Validation Loss: 24.77639389038086 Test Loss: 26.68712615966797
Epoch:  51  Training Loss: 16.73019791 Validation Loss: 25.727230072021484 Test Loss: 26.18926239013672
Epoch:  56  Training Loss: 15.65897942 Validation Loss: 24.607789993286133 Test Loss: 26.283353805541992
Epoch:  61  Training Loss: 15.83553028 Validation Loss: 25.557931900024414 Test Loss: 26.523502349853516
Epoch:  66  Training Loss: 16.10542488 Validation Loss: 24.783859252929688 Test Loss: 26.49845314025879
Epoch:  71  Training Loss: 16.12506294 Validation Loss: 25.119937896728516 Test Loss: 25.904773712158203
Epoch:  76  Training Loss: 23.43534088 Validation Loss: 24.900409698486328 Test Loss: 25.774229049682617
Epoch:  81  Training Loss: 27.71670532 Validation Loss: 24.300111770629883 Test Loss: 25.933366775512695
Epoch:  86  Training Loss: 16.55937576 Validation Loss: 25.249061584472656 Test Loss: 26.246530532836914
Epoch:  91  Training Loss: 15.98195076 Validation Loss: 25.618528366088867 Test Loss: 26.312997817993164
Epoch:  96  Training Loss: 24.74935913 Validation Loss: 25.098472595214844 Test Loss: 26.34810447692871
Epoch: 100  Training Loss: 16.72462845 Validation Loss: 24.996891021728516 Test Loss: 26.932558059692383
```



## *Batch Size = 8*

In [307]:
```
%%time
batch_size = 8
epochs = 100
model = Model(in_features, out_features, hidden_layers, p=0.1, activation_function = nn.ReLU(inplace=True),
            enable_dropout = True, enable_batch_normalization = True)
criterion = nn.L1Loss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
training_losses,valid_losses,test_losses = mini_batch_train(epochs, batch_size,model,criterion,optimizer)
show_diagrams(training_losses,valid_losses,test_losses)
```

```
Epoch:   1  Training Loss: 48.63601685 Validation Loss: 36.36231994628906 Test Loss: 41.83192825317383
Epoch:   6  Training Loss: 37.35523224 Validation Loss: 32.26008605957031 Test Loss: 37.29832458496094
Epoch:  11  Training Loss: 36.02733994 Validation Loss: 23.488845825195312 Test Loss: 28.839853286743164
Epoch:  16  Training Loss: 19.76366425 Validation Loss: 26.195140838623047 Test Loss: 28.139598846435547
Epoch:  21  Training Loss: 23.87018013 Validation Loss: 24.1007137298584 Test Loss: 26.526342391967773
Epoch:  26  Training Loss: 19.08089066 Validation Loss: 25.232446670532227 Test Loss: 27.193174362182617
Epoch:  31  Training Loss: 24.64947510 Validation Loss: 23.769466400146484 Test Loss: 26.69806671142578
Epoch:  36  Training Loss: 26.95101547 Validation Loss: 24.962451934814453 Test Loss: 26.008960723876953
Epoch:  41  Training Loss: 16.56210709 Validation Loss: 24.823184967041016 Test Loss: 26.02180290222168
Epoch:  46  Training Loss: 15.37811852 Validation Loss: 24.121461868286133 Test Loss: 26.442373275756836
Epoch:  51  Training Loss: 14.72021866 Validation Loss: 23.943065643310547 Test Loss: 26.369050979614258
Epoch:  56  Training Loss: 14.23328686 Validation Loss: 24.432334899902344 Test Loss: 25.963966369628906
Epoch:  61  Training Loss: 14.07444668 Validation Loss: 23.199207305908203 Test Loss: 25.501476287841797
Epoch:  66  Training Loss: 13.91252899 Validation Loss: 22.950605392456055 Test Loss: 25.74700927734375
Epoch:  71  Training Loss: 14.20800018 Validation Loss: 24.14919662475586 Test Loss: 25.51437759399414
Epoch:  76  Training Loss: 14.19433022 Validation Loss: 22.977994918823242 Test Loss: 26.49715232849121
Epoch:  81  Training Loss: 22.89653015 Validation Loss: 23.608970642089844 Test Loss: 26.540042877197266
Epoch:  86  Training Loss: 14.49623108 Validation Loss: 23.483076095581055 Test Loss: 26.404197692871094
Epoch:  91  Training Loss: 14.93025589 Validation Loss: 24.176942825317383 Test Loss: 24.851224899291992
Epoch:  96  Training Loss: 13.97304344 Validation Loss: 25.126245498657227 Test Loss: 25.73317527770996
Epoch: 100  Training Loss: 14.30019569 Validation Loss: 24.45240592956543 Test Loss: 25.208208084106445
```



## 2) One hidden layer

In [313]:
```
hidden_layers = [128]
model = Model(in_features, out_features, hidden_layers, p=0.1, activation_function = nn.ReLU(inplace=True),
              enable_dropout = True, enable_batch_normalization = True)
criterion = nn.L1Loss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
```

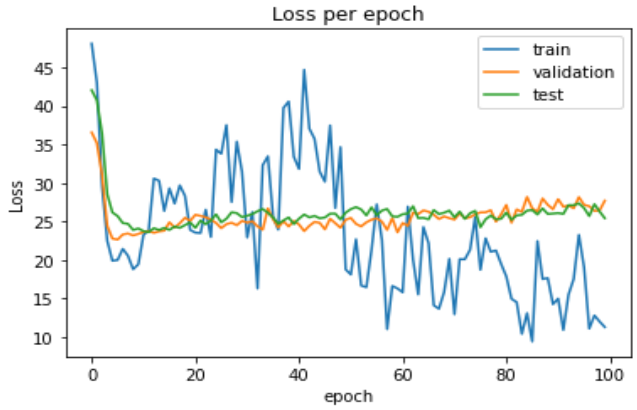
### *Batch Size = 4*

**Batch Size = 4**

In [314]:
```
%%time
batch_size = 4
epochs = 100
training_losses,valid_losses,test_losses = mini_batch_train(epochs, batch_size,model,criterion,optimizer)
show_diagrams(training_losses,valid_losses,test_losses)
```

```
Epoch:   1  Training Loss: 48.06127167 Validation Loss: 36.54461669921875 Test Loss: 42.019866943359375
Epoch:   6  Training Loss: 20.00753403 Validation Loss: 22.651248931884766 Test Loss: 25.71036148071289
Epoch:  11  Training Loss: 23.27529526 Validation Loss: 23.633024215698242 Test Loss: 23.758832931518555
Epoch:  16  Training Loss: 29.30154419 Validation Loss: 24.899208068847656 Test Loss: 23.916160583496094
Epoch:  21  Training Loss: 23.55874825 Validation Loss: 25.87247085571289 Test Loss: 24.181652069091797
Epoch:  26  Training Loss: 33.77556992 Validation Loss: 24.133867263793945 Test Loss: 24.89191436767578
Epoch:  31  Training Loss: 22.93031883 Validation Loss: 24.85866355895996 Test Loss: 25.745868682861328
Epoch:  36  Training Loss: 25.61676025 Validation Loss: 25.30338478088379 Test Loss: 25.51007652282715
Epoch:  41  Training Loss: 31.80470467 Validation Loss: 24.620912551879883 Test Loss: 25.36992645263672
Epoch:  46  Training Loss: 30.15955925 Validation Loss: 23.98025894165039 Test Loss: 25.523984909057617
Epoch:  51  Training Loss: 18.12349129 Validation Loss: 25.464466094970703 Test Loss: 26.60573959350586
Epoch:  56  Training Loss: 27.22848511 Validation Loss: 25.41373634338379 Test Loss: 25.804195404052734
Epoch:  61  Training Loss: 15.81865120 Validation Loss: 24.806629180908203 Test Loss: 25.950115203857422
Epoch:  66  Training Loss: 22.18802643 Validation Loss: 26.267011642456055 Test Loss: 25.285261154174805
Epoch:  71  Training Loss: 12.97052193 Validation Loss: 25.250246047973633 Test Loss: 25.54552459716797
Epoch:  76  Training Loss: 18.73601151 Validation Loss: 26.17971420288086 Test Loss: 24.26059341430664
Epoch:  81  Training Loss: 17.87577057 Validation Loss: 27.15157699584961 Test Loss: 26.155323028564453
Epoch:  86  Training Loss: 9.44549370 Validation Loss: 26.70009422302246 Test Loss: 26.491859436035156
Epoch:  91  Training Loss: 15.01011753 Validation Loss: 27.900081634521484 Test Loss: 26.09693145751953
Epoch:  96  Training Loss: 19.07786942 Validation Loss: 27.16149139404297 Test Loss: 26.74620819091797
Epoch: 100  Training Loss: 11.31524658 Validation Loss: 27.698793411254883 Test Loss: 25.42986488342285
```

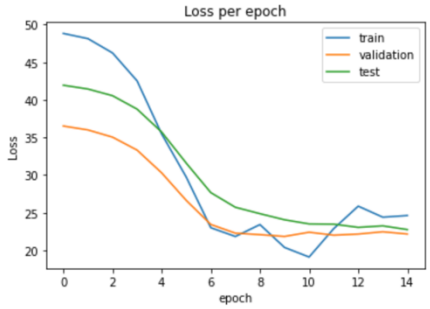
```
CPU times: user 28.2 s, sys: 16.2 s, total: 44.4 s
Wall time: 31.4 s
```



## *Batch Size = 8*

In [315]:
```
%%time
batch_size = 8
epochs = 15
model = Model(in_features, out_features, hidden_layers, p=0.1, activation_function = nn.ReLU(inplace=True),
            enable_dropout = True, enable_batch_normalization = True)
criterion = nn.L1Loss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
training_losses,valid_losses,test_losses = mini_batch_train(epochs, batch_size,model,criterion,optimizer)
show_diagrams(training_losses,valid_losses,test_losses)
```

```
Epoch:   1  Training Loss: 48.82366562 Validation Loss: 36.51918029785156 Test Loss: 41.9394645690918
Epoch:   6  Training Loss: 29.73490524 Validation Loss: 26.638010025024414 Test Loss: 31.59139060974121
Epoch:  11  Training Loss: 19.11078644 Validation Loss: 22.40689468383789 Test Loss: 23.510875701904297
Epoch: 15  Training Loss: 24.62631226 Validation Loss: 22.167144775390625 Test Loss: 22.75712013244629
```




```
CPU times: user 2.17 s, sys: 1.17 s, total: 3.34 s
Wall time: 2.35 s
```

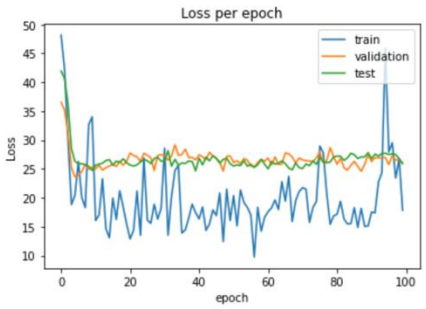
## 3) Two hidden layers

In [316]:
```
hidden_layers = [128,64]
model = Model(in_features, out_features, hidden_layers, p=0.1, activation_function = nn.ReLU(inplace=True),
              enable_dropout = True, enable_batch_normalization = True)
criterion = nn.L1Loss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
```

### *Batch Size = 4*

In [317]:
```
%%time
batch_size = 4
epochs = 100
training_losses,valid_losses,test_losses = mini_batch_train(epochs, batch_size,model,criterion,optimizer)
show_diagrams(training_losses,valid_losses,test_losses)
```

```
Epoch:   1  Training Loss: 48.14212418 Validation Loss: 36.50466537475586 Test Loss: 41.930198669433594
Epoch:   6  Training Loss: 26.26562500 Validation Loss: 23.91388702392578 Test Loss: 25.908340454101562
Epoch:  11  Training Loss: 16.04480362 Validation Loss: 24.997468948364258 Test Loss: 25.63582420349121
Epoch:  16  Training Loss: 19.93900681 Validation Loss: 25.746519088745117 Test Loss: 25.635622024536133
Epoch:  21  Training Loss: 12.85325432 Validation Loss: 27.74827003479004 Test Loss: 25.672225952148438
Epoch:  26  Training Loss: 16.12883759 Validation Loss: 27.333850860595703 Test Loss: 26.40426254272461
Epoch:  31  Training Loss: 28.56284332 Validation Loss: 27.293407440185547 Test Loss: 26.3643798828125
Epoch:  36  Training Loss: 13.88274956 Validation Loss: 27.461122512817383 Test Loss: 26.006126403808594
Epoch:  41  Training Loss: 16.32897758 Validation Loss: 27.43752098083496 Test Loss: 26.84707260131836
Epoch:  46  Training Loss: 16.89708519 Validation Loss: 26.75055694580078 Test Loss: 26.855558395385742
Epoch:  51  Training Loss: 20.35611725 Validation Loss: 26.08938217163086 Test Loss: 25.47571563720703
Epoch:  56  Training Loss: 17.01973724 Validation Loss: 25.47609519958496 Test Loss: 25.791196823120117
Epoch:  61  Training Loss: 17.58226013 Validation Loss: 26.878965377807617 Test Loss: 24.984392166137695
Epoch:  66  Training Loss: 19.39099503 Validation Loss: 27.758543014526367 Test Loss: 25.909059524536133
Epoch:  71  Training Loss: 21.73857117 Validation Loss: 26.542055130004883 Test Loss: 25.056201934814453
Epoch:  76  Training Loss: 28.94837570 Validation Loss: 27.87520408630371 Test Loss: 26.980623245239258
Epoch:  81  Training Loss: 17.14282036 Validation Loss: 25.788959503173828 Test Loss: 27.184818267822266
Epoch:  86  Training Loss: 18.27406311 Validation Loss: 26.293060302734375 Test Loss: 27.43996238708496
Epoch:  91  Training Loss: 17.51123810 Validation Loss: 26.274627685546875 Test Loss: 26.73206329345703
Epoch:  96  Training Loss: 27.93403435 Validation Loss: 25.737852096557617 Test Loss: 27.47972297668457
Epoch: 100  Training Loss: 17.81360435 Validation Loss: 25.942100524902344 Test Loss: 25.92894172668457
```

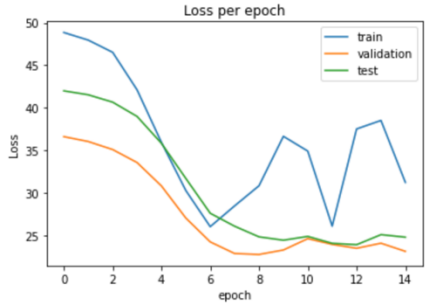



```
CPU times: user 34.6 s, sys: 21.3 s, total: 55.9 s
Wall time: 38 s
```

## *Batch Size = 8*

In [318]:
```
%%time
batch_size = 8
epochs = 15
model = Model(in_features, out_features, hidden_layers, p=0.1, activation_function = nn.ReLU(inplace=True),
              enable_dropout = True, enable_batch_normalization = True)
criterion = nn.L1Loss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
training_losses,valid_losses,test_losses = mini_batch_train(epochs, batch_size,model,criterion,optimizer)
show_diagrams(training_losses,valid_losses,test_losses)
```

```
Epoch:   1  Training Loss: 48.86345673 Validation Loss: 36.5994873046875 Test Loss: 41.991634368896484
Epoch:   6  Training Loss: 30.25541306 Validation Loss: 27.01405143737793 Test Loss: 31.67629051208496
Epoch:  11  Training Loss: 34.89363480 Validation Loss: 24.5869140625 Test Loss: 24.848676681518555
Epoch: 15  Training Loss: 31.21176147 Validation Loss: 23.08993911743164 Test Loss: 24.759292602539062
```



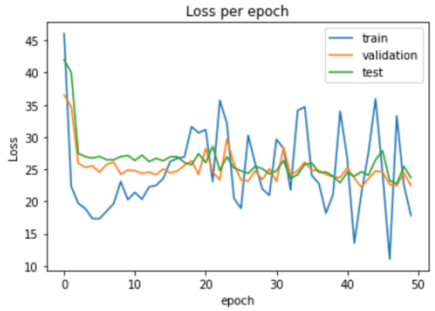

```
CPU times: user 2.93 s, sys: 1.7 s, total: 4.63 s
Wall time: 3.19 s
```

## Part 8 : More hidden layers

In [345]:
```
hidden_layers = [128,90,70]
model = Model(in_features, out_features, hidden_layers, p=0.1, activation_function = nn.ReLU(inplace= True),
              enable_dropout = True)
criterion = nn.L1Loss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
```

In [346]:
```
%%time
epochs = 50
batch_size = 4
training_losses,valid_losses,test_losses = mini_batch_train(epochs, batch_size,model,criterion,optimizer)
show_diagrams(training_losses,valid_losses,test_losses)
```

```
Epoch:   1  Training Loss: 46.06908417 Validation Loss: 36.54243469238281 Test Loss: 41.98584747314453
Epoch:   6  Training Loss: 17.31212234 Validation Loss: 24.53360939025879 Test Loss: 26.974668502807617
Epoch:  11  Training Loss: 21.39644623 Validation Loss: 24.79041290283203 Test Loss: 26.36368751525879
Epoch:  16  Training Loss: 26.22759819 Validation Loss: 24.42110252380371 Test Loss: 26.93545150756836
Epoch:  21  Training Loss: 31.13757896 Validation Loss: 28.21663475036621 Test Loss: 26.022268295288086
Epoch:  26  Training Loss: 18.93076324 Validation Loss: 23.324853897094727 Test Loss: 24.75325584411621
Epoch:  31  Training Loss: 29.64721680 Validation Loss: 23.094045639038086 Test Loss: 24.759601593017578
Epoch:  36  Training Loss: 24.01453400 Validation Loss: 24.86227798461914 Test Loss: 25.942150115966797
Epoch:  41  Training Loss: 26.77181244 Validation Loss: 25.231740951538086 Test Loss: 24.518226623535156
Epoch:  46  Training Loss: 24.27566719 Validation Loss: 24.550844192504883 Test Loss: 27.87493324279785
Epoch: 50  Training Loss: 17.82063675 Validation Loss: 22.454164505004883 Test Loss: 23.68338394165039
```




```
CPU times: user 8.49 s, sys: 89.8 ms, total: 8.58 s
Wall time: 8.6 s
```

## Conclusion

According to previous parts, these are the parameters:

- Batch size: 2564
- Loss Function: MAE
- Activation Function: ReLU
- Learning Rate: 0.001
- Momentum: 0.9
- Optimizer: SGD
- Four hidden layers with 128, 90 and 70 neurons.
- Batch Normalization is used.

## 3. Dimensionality Reduction

**Note:** In both AutoEncoder and PCA, we continue reducing dimension until the performance of the model becomes worse and then we stop at that point.

### Principal Component Analysis

#### Step 1) Standardization

The aim of this step is to standardize the range of the continuous initial variables so that each one of them contributes equally to the analysis.

$$z = \frac{value - mean}{standard\ deviation}$$

#### Step 2) Calculate Covariance Matrix

The covariance matrix is a $p \times p$ symmetric matrix (where $p$ is the number of dimensions) that has as entries the covariances associated with all possible pairs of the initial variables. For example, for a 3-dimensional data set with 3 variables $x$, $y$, and $z$, the covariance matrix is a 3×3 matrix of this from:

$$\begin{bmatrix} Cov(x,x) & Cov(x,y) & Cov(x,z) \\ Cov(y,x) & Cov(y,y) & Cov(y,z) \\ Cov(z,x) & Cov(z,y) & Cov(z,z) \end{bmatrix}$$

#### Step 3) Calculate eigenvalues and eigenvectors

Eigenvectors and eigenvalues are the linear algebra concepts that we need to compute from the covariance matrix in order to determine the ***principal components*** of the data. Before getting to the explanation of these concepts, let's first understand what do we mean by principal components.

#### Step 4) Feature Vector

In this step, what we do is, to choose whether to keep all these components or discard those of lesser significance (of low eigenvalues), and form with the remaining ones a matrix of vectors that we call *Feature vector*.

#### Step 5) Recast the data along the principal components axes

$$\boldsymbol{a}^T\boldsymbol{x} = 0, \quad \boldsymbol{x} \in \{\boldsymbol{x}^i\}_{i=1}^m$$

$$R = \sum_{i=1}^{m} \boldsymbol{x}^i \boldsymbol{x}^{iT} = \mathrm{V}\Lambda\mathrm{V}^T$$

$$\Lambda = \begin{bmatrix} \Lambda^1 & 0 \\ 0 & \Lambda^2 \end{bmatrix} \quad \Lambda^1 = diag([\lambda_1, .., \lambda_r]) = \mathbf{0} \quad \Lambda^2 = diag([\lambda_{r+1} \ldots., \lambda_n])$$

$$\mathrm{V} = \left[\underbrace{v_1 .. v_r}_{\mathrm{V}_1} \underbrace{v_{r+1} \ldots v_n}_{\mathrm{V}_2}\right] = [\mathrm{V}_1\ \mathrm{V}_2]$$

r independent Correlations: $v_j^T x = 0 (j \epsilon \{1, \ldots, r\})$

# 1) Dimension Reduction on Question 1

## Defining the AutoEncoder

In [109]:
```
encoding_dim = 17
ncol = 34
input_dim = Input(shape = (ncol, ))

# Encoder Layers
encoded1 = Dense(28, activation = 'relu')(input_dim)
encoded2 = Dense(22, activation = 'relu')(encoded1)
encoded3 = Dense(encoding_dim, activation = 'relu')(encoded2)

# Decoder Layers
decoded1 = Dense(22, activation = 'relu')(encoded3)
decoded2 = Dense(28, activation = 'relu')(decoded1)
decoded13 = Dense(ncol, activation = 'sigmoid')(decoded2)

# Combine Encoder and Deocder layers
autoencoder = Model(inputs = input_dim, outputs = decoded13)

# Compile the Model
autoencoder.compile(optimizer = 'adam', loss = 'binary_crossentropy')
```

In [110]:
```
autoencoder.summary()
```

```
Model: "model_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_7 (InputLayer)        [(None, 34)]              0

 dense_20 (Dense)            (None, 28)                980

 dense_21 (Dense)            (None, 22)                638

 dense_22 (Dense)            (None, 17)                391

 dense_23 (Dense)            (None, 22)                396

 dense_24 (Dense)            (None, 28)                644

 dense_25 (Dense)            (None, 34)                986

=================================================================
Total params: 4,035
Trainable params: 4,035
Non-trainable params: 0
```

## Training the AutoEncoder

In [111]:
```
autoencoder.fit(train_scaled, train_scaled, epochs = 200, batch_size = 32,
                shuffle = False, validation_data = (test_scaled, test_scaled))
```

```
Epoch 192/200
9/9 [==============================] - 0s 6ms/step - loss: 0.4784 - val_loss: 0.5148
Epoch 193/200
9/9 [==============================] - 0s 6ms/step - loss: 0.4783 - val_loss: 0.5149
Epoch 194/200
9/9 [==============================] - 0s 5ms/step - loss: 0.4782 - val_loss: 0.5146
Epoch 195/200
9/9 [==============================] - 0s 7ms/step - loss: 0.4782 - val_loss: 0.5149
Epoch 196/200
9/9 [==============================] - 0s 8ms/step - loss: 0.4782 - val_loss: 0.5147
Epoch 197/200
9/9 [==============================] - 0s 6ms/step - loss: 0.4780 - val_loss: 0.5146
Epoch 198/200
9/9 [==============================] - 0s 8ms/step - loss: 0.4780 - val_loss: 0.5146
Epoch 199/200
9/9 [==============================] - 0s 6ms/step - loss: 0.4780 - val_loss: 0.5145
Epoch 200/200
9/9 [==============================] - 0s 6ms/step - loss: 0.4780 - val_loss: 0.5144
```

Out[111]:
```
<keras.callbacks.History at 0x7fbc1a230fd0>
```

## Use AutoEncoder to reduce dimension of train and test data

In [112]:
```
encoder = Model(inputs = input_dim, outputs = encoded3)
encoded_input = Input(shape = (encoding_dim, ))
```

In [113]:
```
encoded_train = pd.DataFrame(encoder.predict(train_scaled))
encoded_train = encoded_train.add_prefix('feature_')
encoded_train = encoded_train.values

encoded_test = pd.DataFrame(encoder.predict(test_scaled))
encoded_test = encoded_test.add_prefix('feature_')
encoded_test = encoded_test.values

encoded_train = encoded_train.astype(np.float)
encoded_test = encoded_test.astype(np.float)
```

In [114]:
```
encoded_train
```

Out[114]:
```
array([[12.44891834,  3.57403398,  3.40646195, ...,  0.        ,
        14.05890369,  4.28539228],
       [10.51262093,  3.3840394 ,  2.11868024, ...,  0.        ,
        11.56116104,  3.17635918],
       [10.3824501 ,  3.22484565,  1.71931291, ...,  0.        ,
        11.5196476 ,  2.9691503 ],
       ...,
       [ 7.04390669,  0.93778944,  3.55841637, ...,  0.        ,
         5.81766605,  0.        ],
       [ 5.88840103,  2.27328229,  0.9406842 , ...,  0.        ,
         6.52591705,  2.90134454],
       [ 7.61486101,  3.39422536,  2.08512163, ...,  0.        ,
         8.39262009,  2.82381725]])
```

## Trying the encoded training set on the model

In [130]:
```
in_features = encoding_dim
out_features = 2
hidden_layers = [1000, 500, 250, 125]
batch_size = 8
epochs = 30

model2 = Model(in_features, out_features, hidden_layers, p=0.4, activation_function = nn.ReLU(inplace = True),
               enable_dropout = True)

criterion = nn.CrossEntropyLoss()

optimizer = torch.optim.Adam(model2.parameters(), lr=0.001)
```

In [131]:
```
%%time
training_losses,valid_losses = mini_batch_train(encoded_train,encoded_test,
                                                y_train,y_test,epochs, batch_size,model2,criterion,optimizer)
```

```
Epoch:   1  Training Loss: 0.45541608 Test Loss: 0.7531951665878296
Epoch:   6  Training Loss: 0.07905960 Test Loss: 0.4581572413444519
Epoch:  11  Training Loss: 0.27242512 Test Loss: 0.30584418773651123
Epoch:  16  Training Loss: 0.09454083 Test Loss: 0.24069122970104218
Epoch:  21  Training Loss: 0.27789414 Test Loss: 0.2812822163105011
Epoch:  26  Training Loss: 0.49703926 Test Loss: 0.20862886309623718
Epoch: 30  Training Loss: 0.18928412 Test Loss: 0.24080748856067657
CPU times: user 14.4 s, sys: 160 ms, total: 14.6 s
Wall time: 14.5 s
```

*Figure 53 Loss per epoch*

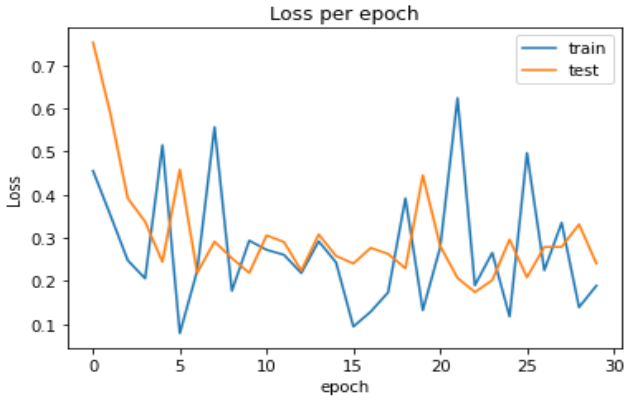
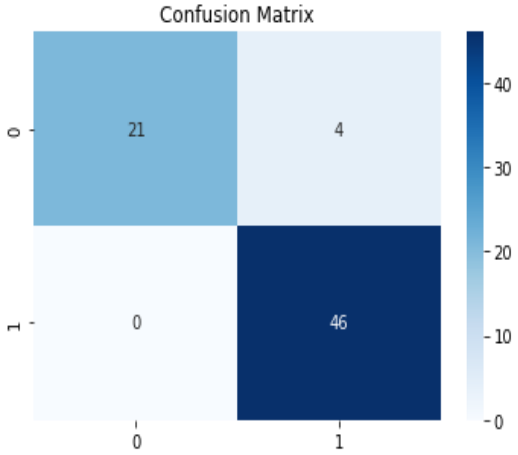



*Figure 54 Classification Report*

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| bad | 1.00 | 0.84 | 0.91 | 25 |
| good | 0.92 | 1.00 | 0.96 | 46 |
| accuracy | | | 0.94 | 71 |
| macro avg | 0.96 | 0.92 | 0.94 | 71 |
| weighted avg | 0.95 | 0.94 | 0.94 | 71 |

*Figure 55 Confusion Matrix*




## The effect of dimension reduction on the model's result

In question 1, we achieved 96% precision but with dimension reduction the precision is 88%. The performance of the model became worse due to possible data loss from dimension reduction.

# Dimensionality Reduction using PCA

## Performing standard Scaling

In [18]:
```
from sklearn.preprocessing import StandardScaler

X_scaled = StandardScaler().fit_transform(X)
X_scaled[1]
```

Out[18]:
```
array([ 0.34843328,  0.        ,  0.72164805, -0.52781074,  0.63430773,
       -1.03758697, -1.33910577, -2.02945199,  0.96407416, -0.46948184,
        0.05785987, -1.68478062, -0.09090821, -1.599628  , -1.32077439,
       -2.28587042, -0.52979769, -1.24735087, -0.04525228, -1.88290035,
       -0.76886206, -0.8915184 , -0.90069558, -0.56945458, -1.03778979,
       -0.38305389, -1.44784862, -0.20841918, -0.9891848 , -0.17352964,
       -0.90906286, -0.11521328, -0.93260505, -0.08328554])
```

## Covariance matrix

In [19]:
```
features = X_scaled.T
cov_matrix = np.cov(features)
cov_matrix[3]
```

Out[19]:
```
array([-6.54750578e-03,  0.00000000e+00,  1.43774417e-01,  1.00285714e+00,
        1.15514103e-03, -1.90851343e-01, -5.41838976e-02,  2.55731541e-01,
       -3.03180852e-01,  2.08290745e-01, -1.90632900e-01,  3.16779785e-01,
       -1.49642618e-01,  2.37280420e-01, -2.53872855e-01,  1.86403368e-01,
       -2.51860511e-01, -1.47872635e-01, -3.33162250e-01,  1.67721934e-01,
       -2.81886795e-01, -3.55018389e-02, -1.44129607e-01,  1.64665149e-01,
       -1.04930880e-01, -2.37664211e-01, -4.70435699e-02,  7.45165168e-04,
       -4.12078088e-02,  3.43278732e-01, -1.72767939e-01, -1.23138974e-01,
       -1.54404292e-01,  3.47073068e-02])
```

## Eigendecomposition

Here we will calculate eigen values and eigen vectors :

In [20]:
```
values, vectors = np.linalg.eig(cov_matrix)
values[:5]
```

Out[20]:
```
array([8.83731976, 4.25075499, 2.72400955, 2.39537094, 1.95572286])
```

In [21]:
```
explained_variances = []
for i in range(len(values)):
    explained_variances.append(values[i] / np.sum(values))

print(np.sum(explained_variances), '\n', explained_variances)
```

```
1.0000000000000004
 [0.26703461231476505, 0.1284437749715908, 0.0823105708404918, 0.07238019752536178, 0.059095484710595086, 0.035437057
64437251, 0.03383567924112817, 0.030600314982258257, 0.02801452920722827, 0.02523130475100366, 0.023396779243677227,
0.02153966622577981 4, 0.01846089891074043, 0.01706554924704 4426, 0.016056337886557113, 0.014579057076879247, 0.013454
670453810593, 0.002106282995601896, 0.012429550944717805, 0.0029267372726344445, 0.011428814796377265, 0.011286355041
708948, 0.01022673480854499, 0.009256607731627528, 0.008435629494280455, 0.003897518646238352, 0.004144880919084293,
0.007408922743682907, 0.005107368467253884, 0.005266100873078514, 0.005882071869741417, 0.006767689272421887, 0.00649
22488897214605, 0.0]
```

## Projection

In [22]:
```
projected_1 = X_scaled.dot(vectors.T[0])
projected_2 = X_scaled.dot(vectors.T[1])
projected_3 = X_scaled.dot(vectors.T[2])
projected_4 = X_scaled.dot(vectors.T[3])
projected_5 = X_scaled.dot(vectors.T[4])
projected_6 = X_scaled.dot(vectors.T[5])
projected_7 = X_scaled.dot(vectors.T[6])
projected_8 = X_scaled.dot(vectors.T[7])
projected_9 = X_scaled.dot(vectors.T[8])
projected_10 = X_scaled.dot(vectors.T[9])
```

In [23]:
```
res = pd.DataFrame(projected_1, columns=['PC1'])
res['PC2'] = projected_2
res['PC3'] = projected_3
res['PC4'] = projected_4
res['PC5'] = projected_5
res['PC6'] = projected_6
res['PC7'] = projected_7
res['PC8'] = projected_8
res['PC9'] = projected_9
res['PC10'] = projected_10

res['target'] = y
res
```

*Figure 56 Dimension Reduced Dataset*

| | PC1 | PC2 | PC3 | PC4 | PC5 | PC6 | PC7 | PC8 | PC9 | PC10 | target |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1.668242 | 2.054597 | 0.064908 | -1.712482 | 0.009364 | 0.479606 | -0.730332 | 0.110340 | 0.718496 | -0.874031 | 1 |
| **1** | -0.823940 | 2.583636 | -1.975118 | -1.392990 | -0.383751 | 0.763313 | -1.535862 | 0.306477 | 0.489140 | -3.214970 | 0 |
| **2** | 2.034095 | 0.748245 | 0.837354 | -1.108202 | 0.120072 | 0.062519 | -0.220461 | 0.100407 | 0.354253 | -0.322654 | 1 |
| **3** | -1.252374 | -1.363326 | -0.740357 | 2.670223 | -1.934133 | 1.241942 | 0.164094 | -0.526752 | -0.775621 | -3.025268 | 0 |
| **4** | -0.017756 | 1.845357 | 0.343250 | -2.716045 | -0.298941 | 0.297013 | -0.590563 | -0.370771 | 1.042176 | -1.197267 | 1 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **346** | 3.156856 | 0.000433 | 0.410117 | 0.097405 | 0.503017 | -0.297292 | -0.409806 | -0.126291 | -0.486821 | 0.406002 | 1 |
| **347** | 3.562330 | -0.324750 | 0.639838 | 0.265659 | 0.266194 | -0.140936 | -0.076933 | -0.037992 | -0.282564 | 0.232936 | 1 |
| **348** | 3.512043 | -0.204754 | 0.639182 | 0.180498 | 0.164201 | -0.092654 | -0.060033 | -0.033479 | -0.316029 | 0.255157 | 1 |
| **349** | 3.308935 | 0.343454 | 0.681005 | -0.058668 | 0.051518 | 0.013579 | 0.141086 | 0.077646 | -0.212691 | 0.132033 | 1 |
| **350** | 2.705653 | 0.230427 | 0.529344 | 0.036980 | 0.372647 | -0.308558 | -0.325846 | 0.041552 | -0.106898 | 0.329442 | 1 |

351 rows × 11 columns

## Trying the new dataset

In [25]:
```
in_features = len(res.columns) - 1
out_features = 2
hidden_layers = [1000, 500, 250, 125]
batch_size = 8
epochs = 10

model2 = Model(in_features, out_features, hidden_layers, p=0.4, activation_function = nn.ReLU(inplace = True),
               enable_dropout = True)

criterion = nn.CrossEntropyLoss()

optimizer = torch.optim.Adam(model2.parameters(), lr=0.001)
```

In [26]:
```
%%time
training_losses,valid_losses = mini_batch_train(X_train.values,X_test.values,
                                                y_train.values,y_test.values,epochs,
                                                batch_size,model2,criterion,optimizer)
```

```
Epoch:   1  Training Loss: 0.31482306 Test Loss: 0.6939573884010315
Epoch:   6  Training Loss: 0.32771161 Test Loss: 0.1007436066865921
Epoch: 10  Training Loss: 0.07762404 Test Loss: 0.1085464209318161
CPU times: user 2.92 s, sys: 46.8 ms, total: 2.97 s
Wall time: 2.98 s
```

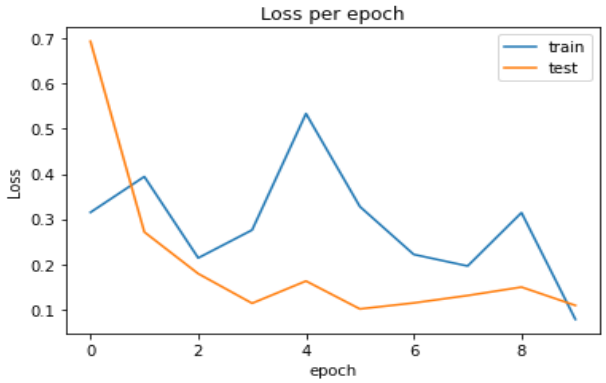
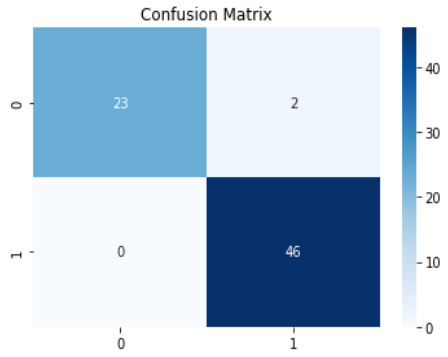



*Figure 57 Classification Report*

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| bad | 1.00 | 0.92 | 0.96 | 25 |
| good | 0.96 | 1.00 | 0.98 | 46 |
| accuracy | | | 0.97 | 71 |
| macro avg | 0.98 | 0.96 | 0.97 | 71 |
| weighted avg | 0.97 | 0.97 | 0.97 | 71 |



The performance of our model with PCA method increased by 1% because by removing correlations among input dimensions, nullity spaces of the input data will be removed and the variance of the feature space will be maximized.

But the performance of the model didn’t change much with AutoEncoder.

# 2) Dimension Reduction on Question 2

## Dimensionality Reduction using AutoEncoder

### Defining the AutoEncoder

In [161]:
```
import keras
encoding_dim = 30
ncol = 68
input_dim = Input(shape = (ncol, ))

# Encoder Layers
encoded1 = Dense(60, activation = 'relu')(input_dim)
encoded2 = Dense(50, activation = 'relu')(encoded1)
encoded3 = Dense(40, activation = 'relu')(encoded2)
encoded4 = Dense(encoding_dim, activation = 'relu')(encoded3)

# Decoder Layers
decoded1 = Dense(40, activation = 'relu')(encoded4)
decoded2 = Dense(50, activation = 'relu')(decoded1)
decoded3 = Dense(60, activation = 'relu')(decoded2)
decoded4 = Dense(ncol, activation = 'sigmoid')(decoded3)

# Combine Encoder and Deocder layers
autoencoder2 = keras.models.Model(inputs = input_dim, outputs = decoded4)

# Compile the Model
autoencoder2.compile(optimizer = 'adam', loss = 'mse')
```

In [162]:
```
autoencoder2.summary()
```

```
Model: "model_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_11 (InputLayer)       [(None, 68)]              0

 dense_34 (Dense)            (None, 60)                4140

 dense_35 (Dense)            (None, 50)                3050

 dense_36 (Dense)            (None, 40)                2040

 dense_37 (Dense)            (None, 30)                1230

 dense_38 (Dense)            (None, 40)                1240

 dense_39 (Dense)            (None, 50)                2050

 dense_40 (Dense)            (None, 60)                3060

 dense_41 (Dense)            (None, 68)                4148

=================================================================
Total params: 20,958
Trainable params: 20,958
Non-trainable params: 0
```

### Training the AutoEncoder

In [163]:
```
autoencoder2.fit(train_scaled, train_scaled, epochs = 9,
                 batch_size = 32, shuffle = False, validation_data = (test_scaled, test_scaled))
```

```
Epoch 1/9
29/29 [==============================] - 1s 9ms/step - loss: 0.0619 - val_loss: 0.0470
Epoch 2/9
29/29 [==============================] - 0s 4ms/step - loss: 0.0235 - val_loss: 0.0332
Epoch 3/9
29/29 [==============================] - 0s 3ms/step - loss: 0.0158 - val_loss: 0.0303
Epoch 4/9
29/29 [==============================] - 0s 3ms/step - loss: 0.0150 - val_loss: 0.0304
Epoch 5/9
29/29 [==============================] - 0s 4ms/step - loss: 0.0146 - val_loss: 0.0291
Epoch 6/9
29/29 [==============================] - 0s 4ms/step - loss: 0.0136 - val_loss: 0.0272
Epoch 7/9
29/29 [==============================] - 0s 5ms/step - loss: 0.0128 - val_loss: 0.0258
Epoch 8/9
29/29 [==============================] - 0s 4ms/step - loss: 0.0123 - val_loss: 0.0251
Epoch 9/9
29/29 [==============================] - 0s 3ms/step - loss: 0.0120 - val_loss: 0.0250
```

Out[163]:
```
<keras.callbacks.History at 0x7fbb0b694810>
```

## Use AutoEncoder to reduce dimension of train and test data

In [164]:
```
encoder = keras.models.Model(inputs = input_dim, outputs = encoded4)
encoded_input = Input(shape = (encoding_dim, ))
```

In [165]:
```
encoded_train = pd.DataFrame(encoder.predict(train_scaled))
encoded_train = encoded_train.add_prefix('feature_')
encoded_train = encoded_train.values

encoded_test = pd.DataFrame(encoder.predict(test_scaled))
encoded_test = encoded_test.add_prefix('feature_')
encoded_test = encoded_test.values
```

In [166]:
```
encoded_train
```

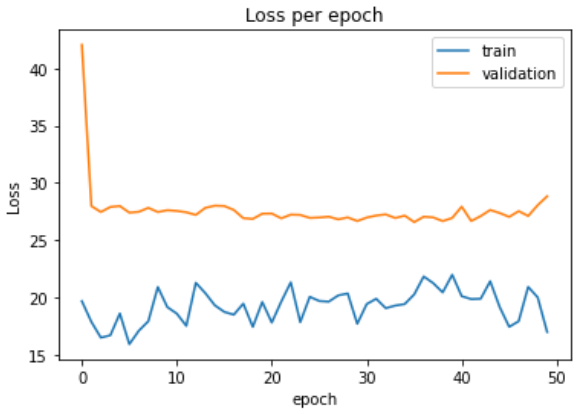
Out[166]:
```
array([[0.        , 0.5062831 , 0.        , ..., 0.79190516, 1.1676931 ,
        1.2954531 ],
       [0.        , 0.5547495 , 0.        , ..., 1.0386316 , 1.014297  ,
        1.1831064 ],
       [0.        , 0.2798609 , 0.        , ..., 1.68701   , 1.0890676 ,
        0.96427184],
       ...,
       [0.        , 0.57057875, 0.        , ..., 1.833194  , 1.0275724 ,
        0.7917586 ],
       [0.        , 0.5927711 , 0.        , ..., 0.8613547 , 1.1037238 ,
        1.1621649 ],
       [0.        , 0.5497524 , 0.        , ..., 0.28247762, 0.9013232 ,
        1.1970714 ]], dtype=float32)
```

## Trying the encoded training set on the model

In [169]:
```
hidden_layers = [128,90,70]
in_features = encoding_dim
out_features = 2
model2 = Model(in_features, out_features, hidden_layers, p=0.1, activation_function = nn.ReLU(inplace= True),
               enable_dropout = True)
criterion = nn.L1Loss()
optimizer = torch.optim.SGD(model2.parameters(), lr=0.001, momentum=0.9)
```

In [170]:
```
%%time
epochs = 50
batch_size = 4
training_losses,valid_losses = mini_batch_train2(encoded_train,encoded_test,
                                                 y_train,y_test,epochs, batch_size,model2,criterion,optimizer)
```

```
Epoch:   1  Training Loss: 19.69710732 Test Loss: 42.0395393371582
Epoch:   6  Training Loss: 15.94266891 Test Loss: 27.40904998779297
Epoch:  11  Training Loss: 18.60629654 Test Loss: 27.558895111083984
Epoch:  16  Training Loss: 18.75740814 Test Loss: 27.98607063293457
Epoch:  21  Training Loss: 17.82673264 Test Loss: 27.332050323486328
Epoch:  26  Training Loss: 19.70602417 Test Loss: 26.985719680786133
Epoch:  31  Training Loss: 19.44206619 Test Loss: 26.982397079467773
Epoch:  36  Training Loss: 20.28242493 Test Loss: 26.584331512451172
Epoch:  41  Training Loss: 20.12581635 Test Loss: 27.93431854248047
Epoch:  46  Training Loss: 17.45397758 Test Loss: 27.03972625732422
Epoch: 50  Training Loss: 17.00311279 Test Loss: 28.837860107421875
CPU times: user 11.2 s, sys: 21.2 ms, total: 11.2 s
Wall time: 11.2 s
```



# Dimensionality Reduction using PCA

## Performing Standard Scaling

In [172]:
```
from sklearn.preprocessing import StandardScaler

X_scaled = StandardScaler().fit_transform(X)
X_scaled[1]
```

Out[172]:
```
array([ 0.24819123, -0.07413895, -0.59013731,  0.54780853,  0.84961427,
        0.28680671, -0.05712363, -0.89095684,  0.45982203, -0.07521558,
        0.0380314 ,  0.41976217, -0.02095287,  0.00224764, -1.57297582,
       -0.1222196 ,  0.67467924,  0.93747796,  0.60932402, -0.32857775,
       -0.57011705,  0.34725769, -0.33793639, -0.33057652, -0.07452978,
       -0.68002137, -0.41588148,  0.26229386,  0.07187932,  0.28514   ,
        0.45820445,  0.97864501,  0.57756592,  0.75298774,  1.61279895,
        1.05038552, -0.38265972,  0.07540993, -0.2138075 , -0.29413032,
       -0.53670959,  0.16069277, -0.64768538, -0.27983517, -0.45270878,
       -0.36890013,  0.59450908,  0.18685785, -0.06677047, -0.4663851 ,
       -0.17954285,  1.78871821,  2.14482286, -0.34374603, -0.2030083 ,
        0.39149326, -0.61411527, -0.65849174, -0.10025101, -0.95588575,
       -0.51552421, -0.48051398, -0.18197539, -0.17826406,  0.35354694,
        1.06590697, -0.14545659,  1.35971354])
```

## Covariance matrix

In [173]:
```
features = X_scaled.T
cov_matrix = np.cov(features)
cov_matrix[3]
```

Out[173]:
```
array([-0.22443836, -0.25844304,  0.17742925,  1.00094518, -0.13305504,
        0.26527902,  0.29336275,  0.18259027,  0.23734682,  0.08216195,
        0.14319146,  0.08529367,  0.16993029,  0.05622366,  0.04105032,
        0.08972513, -0.04116812,  0.15848194,  0.19076913,  0.4032434 ,
        0.84206079,  0.44480593,  0.58395767,  0.42263976,  0.30162173,
        0.29285195,  0.29315428,  0.2781375 ,  0.21431387,  0.21547585,
        0.13545668,  0.15424026,  0.09948574,  0.11527738, -0.01882112,
       -0.02385422,  0.03186047,  0.16487642, -0.10124786,  0.00632797,
       -0.12168826, -0.14523265, -0.08378495, -0.13058931, -0.1143681 ,
       -0.15750107, -0.18638738, -0.19916291, -0.1698706 , -0.22857782,
       -0.22309799,  0.08391188,  0.15681993,  0.15707183, -0.1848656 ,
       -0.12712475,  0.06814443, -0.04443274, -0.08804227, -0.09538353,
       -0.05932736, -0.06267707, -0.12273963, -0.1287769 , -0.16338027,
       -0.12223127, -0.15265879, -0.15143429])
```

## Eigendecomposition

Here we will calculate eigen values and eigen vectors :

In [174]:
```
values, vectors = np.linalg.eig(cov_matrix)
values[:5]
```

Out[174]:
```
array([13.60778002,  8.06754586,  5.24429095,  4.76379249,  3.80692885])
```

In [175]:
```
explained_variances = []
for i in range(len(values)):
    explained_variances.append(values[i] / np.sum(values))

print(np.sum(explained_variances), '\n', explained_variances)
```

```
1.0
 [0.1999254466503587, 0.11852834974323313, 0.07704910047831849, 0.06998961913963109, 0.05593138262970693, 0.033164151
01203388, 0.030668654901843367, 0.027492278543740273, 0.025202993888888293, 0.02174396379890392, 0.02053118906873866,
0.01915712035941034, 0.018458931361637437, 0.016369555732943075, 0.014785691602245158, 0.014317792881302038, 0.013866
474679956263, 0.013586508496227659, 0.012412975257254688, 0.011915336093598383, 0.011690322373967915, 0.0102442061741
1587, 0.009332319707971094, 0.009074855853224666, 0.008415330963318602, 0.007896110394006969, 0.0072795733330148615,
0.007097427760417232, 0.0068779191450260195, 0.006632993106598004, 0.0064815650143798, 0.006159730443738818, 0.005927
08841996051, 0.005801338159355524, 0.0052701492962598075, 0.004933508670377504, 0.004562956928912433, 8.7368560365716
54e-05, 0.00419440372605572, 0.004141036189758193, 0.0039757671932681305, 0.00027173204702136955, 0.00354140241439308
66, 0.0006427288668363894, 0.0032224002578077108, 0.0007489095012935197, 0.000837216236984873, 0.0009326728415210443,
0.0009535125718978692, 0.0010221672137442587, 0.001096506552504941, 0.0011512200755572576, 0.0012184250193473958, 0.0
01292998415696224, 0.0014900630946978498, 0.0015588181935276833, 0.0016564294701470596, 0.003039708211417397, 0.00295
674563426288, 0.002823541655146448, 0.0027121989685182516, 0.0025938795659822033, 0.0025469670453787184, 0.0023783553
431561623, 0.0018298739935740695, 0.0021966971892318447, 0.0020412203339321405, 0.002070121552361585]
```

## Projection

In [178]:
```
projected_1 = X_scaled.dot(vectors.T[0])
projected_2 = X_scaled.dot(vectors.T[1])
projected_3 = X_scaled.dot(vectors.T[2])
projected_4 = X_scaled.dot(vectors.T[3])
projected_5 = X_scaled.dot(vectors.T[4])
projected_6 = X_scaled.dot(vectors.T[5])
projected_7 = X_scaled.dot(vectors.T[6])
projected_8 = X_scaled.dot(vectors.T[7])
projected_9 = X_scaled.dot(vectors.T[8])
projected_10 = X_scaled.dot(vectors.T[9])
```

In [179]:
```
res = pd.DataFrame(projected_1, columns=['PC1'])
res['PC2'] = projected_2
res['PC3'] = projected_3
res['PC4'] = projected_4
res['PC5'] = projected_5
res['PC6'] = projected_6
res['PC7'] = projected_7
res['PC8'] = projected_8
res['PC9'] = projected_9
res['PC10'] = projected_10

res['longitude'] = y[:,0]
res['latitude'] = y[:,1]
res
```

*Figure 58 Dimension Reduced Dataset*

| | PC1 | PC2 | PC3 | PC4 | PC5 | PC6 | PC7 | PC8 | PC9 | PC10 | longitude | latitude |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | -1.068725 | 8.437277 | -5.197232 | 8.031617 | 5.961671 | 1.251664 | 3.256382 | -1.772884 | 0.748368 | -0.797154 | -15.75 | -47.95 |
| **1** | 0.190528 | 0.862663 | -0.072849 | 2.825874 | -0.648760 | -0.943278 | -0.801290 | -2.236496 | 0.254405 | -0.098439 | 14.91 | -23.51 |
| **2** | 4.149918 | -1.397989 | 2.099784 | 0.745889 | -2.962653 | 0.924669 | -0.798065 | -1.333758 | 0.223061 | 1.625642 | 12.65 | -8.00 |
| **3** | -0.947960 | 4.093904 | -2.281610 | -0.324632 | -1.668090 | 1.755334 | -0.779106 | -1.675249 | -0.454894 | 1.931967 | 9.03 | 38.74 |
| **4** | 0.366876 | -1.564429 | -0.080291 | 2.907154 | 0.253194 | -0.759745 | 0.033464 | -1.173413 | -1.139334 | 0.068729 | 34.03 | -6.85 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **1054** | 6.036341 | 1.331828 | 1.964494 | 3.493875 | -1.804094 | 0.370083 | -1.313056 | 0.240942 | -2.254625 | 2.249588 | -6.17 | 35.74 |
| **1055** | 5.145328 | -3.306242 | -2.368449 | 2.543880 | -1.145478 | -2.740121 | 0.461468 | 1.368067 | -2.773428 | 0.671453 | 11.55 | 104.91 |
| **1056** | -3.000066 | -1.801716 | 0.673602 | -0.141599 | -0.413566 | -0.880672 | -1.623246 | -0.581166 | 0.354623 | -1.247729 | 41.33 | 19.80 |
| **1057** | 1.149372 | 0.436830 | -1.987901 | -4.579032 | 1.235907 | 2.588423 | 1.105896 | 3.567083 | -0.738530 | 0.784035 | 54.68 | 25.31 |
| **1058** | 17.074460 | -3.121862 | 3.309241 | -1.052874 | 0.992973 | 3.223204 | -0.174479 | 0.475751 | 3.641558 | -1.277468 | 54.68 | 25.31 |

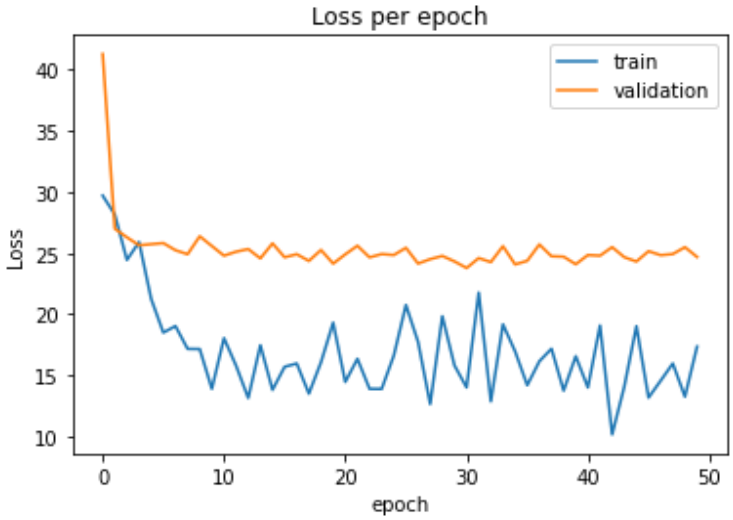
1059 rows × 12 columns

## Trying the new dataset

In [183]:
```
hidden_layers = [128,90,70]
in_features = 10
out_features = 2
model2 = Model(in_features, out_features, hidden_layers, p=0.1, activation_function = nn.ReLU(inplace= True),
            enable_dropout = True)
criterion = nn.L1Loss()
optimizer = torch.optim.SGD(model2.parameters(), lr=0.001, momentum=0.9)
```

In [184]:
```
%%time
epochs = 50
batch_size = 4
training_losses,valid_losses = mini_batch_train2(X_train.values,
                                                 X_test.values,y_train.values,y_test.values,
                                                 epochs, batch_size,model2,criterion,optimizer)
```

```
Epoch:   1  Training Loss: 29.69846535 Test Loss: 41.27515411376953
Epoch:   6  Training Loss: 18.49985313 Test Loss: 25.81812858581543
Epoch:  11  Training Loss: 18.04673576 Test Loss: 24.789751052856445
Epoch:  16  Training Loss: 15.70655823 Test Loss: 24.658926010131836
Epoch:  21  Training Loss: 14.46919918 Test Loss: 24.899093627929688
Epoch:  26  Training Loss: 20.75527191 Test Loss: 25.43099021911621
Epoch:  31  Training Loss: 14.02038288 Test Loss: 23.78188705444336
Epoch:  36  Training Loss: 14.19924927 Test Loss: 24.3695068359375
Epoch:  41  Training Loss: 14.01482391 Test Loss: 24.841533660888672
Epoch:  46  Training Loss: 13.18558884 Test Loss: 25.156126022338867
Epoch: 50  Training Loss: 17.35544014 Test Loss: 24.68306541442871
CPU times: user 9.61 s, sys: 25.9 ms, total: 9.64 s
Wall time: 9.67 s
```




The Mean Absolute Error didn’t change much with this dataset in PCA and AutoEncoder method.