FIT5201 Assignment 2 Task 3: Unsupervised Learning

Student ID: 31237223

Name: Darren Jer Shien Yee

Question 3 Imports

In [2]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import pandas as pd
import torch
from torch import tensor
from torch.optim import Adam
from torch import nn
from torch.utils.data import TensorDataset, DataLoader
```

Question 3.1 Load provided data

In [3]:
```
labeled_train_data = pd.read_csv('Task2C_labeled.csv')
unlabeled_train_data = pd.read_csv('Task2C_unlabeled.csv')
test_data = pd.read_csv('Task2C_test.csv')
X_train_labeled = np.array(labeled_train_data.iloc[:,1:])
y_train_labeled = np.array(labeled_train_data.iloc[:,0])
X_train_unlabeled = np.array(unlabeled_train_data)
X_test_labeled = np.array(test_data.iloc[:,1:])
y_test_labeled = np.array(test_data.iloc[:,0])
```

Question 3.2 Autoencoder code imported from lecture

In [7]:
```
def moving_average(alist, window_size=3):
    numbers_series = pd.Series(alist)
    windows = numbers_series.rolling(window_size)
    moving_averages = windows.mean()

    moving_averages_list = moving_averages.tolist()
    return(moving_averages_list[window_size - 1:])


def normalize(x, m=None, s=None):
    if m is None or s is None:
        #print('Normalizing data: No mean and/or sd given. Assuming it is traini
        m,s = x.mean(), x.std()

    return (x-m)/s

def get_dataloader(X_train,Y_train=None, autoencoder=False,bs=128, standardize=T
    """
    Retrieves a data loader to use for training. In case autoencoder=True, Y_tra
    The function returns the dataloader only if return_dataset is False otherwis
    where train_dataset is the Dataset object after preprocessing.
    """
    try:
        X_train= np.array(X_train).astype(np.float32)
        if standardize: X_train = normalize(X_train)
        if not autoencoder: Y_train = np.array(Y_train)
    except Exception as e:
```

```
        raise Exception('Make sure your input and labels are array-likes. Your i
    # transform into tensors
    if autoencoder:
        Y_train = X_train

    X_train, Y_train = map(tensor, (X_train, Y_train))
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
    X_train = X_train.to(device)
    Y_train = Y_train.to(device)

    train_ds = TensorDataset(X_train,Y_train)
    train_dl = DataLoader(train_ds, batch_size=16)

    if return_dataset: return train_dl,train_ds

    return train_dl

def train_autoencoder(X_train,hidden,activation='Tanh',epochs=10, trace=True, **
    """
    Trains an Autoencoder and returns the trained model

    Params:
    X_train: Input data to train the autoencoder. Can be a dataframe, numpy, 2-D

    hidden: a list of sizes for the hidden layers ex: ([100,2,100]) will train a

    activation (default='Tanh'): Activation type for hidden layers, output layer

    epochs: Number of epochs to train autoencoder

    trace: if true, will display epoch progress and will plot the loss plot at t

    **kwargs: passed to Adam optimizer, lookup adam optimizer for more details
    """
    train_dl = get_dataloader(X_train,autoencoder=True)

    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

    # Building the autoencoder
    n_inps = [X_train.shape[-1]]
    n_out = n_inps
    layer_dims = n_inps + hidden + n_out
    layers = []
    try:
        non_linearity = getattr(nn,activation)()
    except AttributeError:
        raise Exception('Activation type not found, note that it is case senstiv

    for i in range(len(layer_dims)-1):
        layers.extend([nn.Linear(layer_dims[i], layer_dims[i+1]), non_linearity]

    layers.pop()  # to remove the last non-linearity

    model = nn.Sequential(*layers)
    model = model.to(device)
    print('Training Model on %s'%(device))
    # to capture training loss
    losses = []
    epoch_losses = []
    # define optimizer with learning rate
```

```
    optim = Adam(model.parameters(), **kwargs)
    # we use MSE error for reconstruction loss
    loss_criterion = nn.MSELoss()
    # calculate printing step - optional
    printing_step = int(epochs/10)
    # start training
    for epoch in range(epochs):
        for xb,yb in train_dl:
            preds = model(xb)
            loss = torch.mean(torch.norm(preds - yb, p=2,dim=1)) # changed loss
            losses.append(loss.item())
            loss.backward()
            optim.step()
            model.zero_grad()
        # after epoch
        epoch_loss = np.mean(losses[-len(train_dl):]) # average loss across all
        epoch_losses.append(epoch_loss)
        if trace and not epoch%printing_step:
            print(f'Epoch {epoch} out of {epochs}. Loss:{epoch_loss}')
    return model, epoch_losses

def get_deepfeatures(trained_model, X_input,layer_number):
    '''
    Gets deep features of a given `layer_number` upon passing `X_input` through
    '''
    X_input = get_dataloader(X_input,autoencoder=True,return_dataset=True)[1].te
    result = []
    def save_result(m,i,o):
        result.append(o.data)
    hook = trained_model[layer_number].register_forward_hook(save_result)

    with torch.no_grad():
        trained_model(X_input)

    hook.remove()

    return (result[0].cpu().numpy())
```

Question 3.2 Train Autoencoder

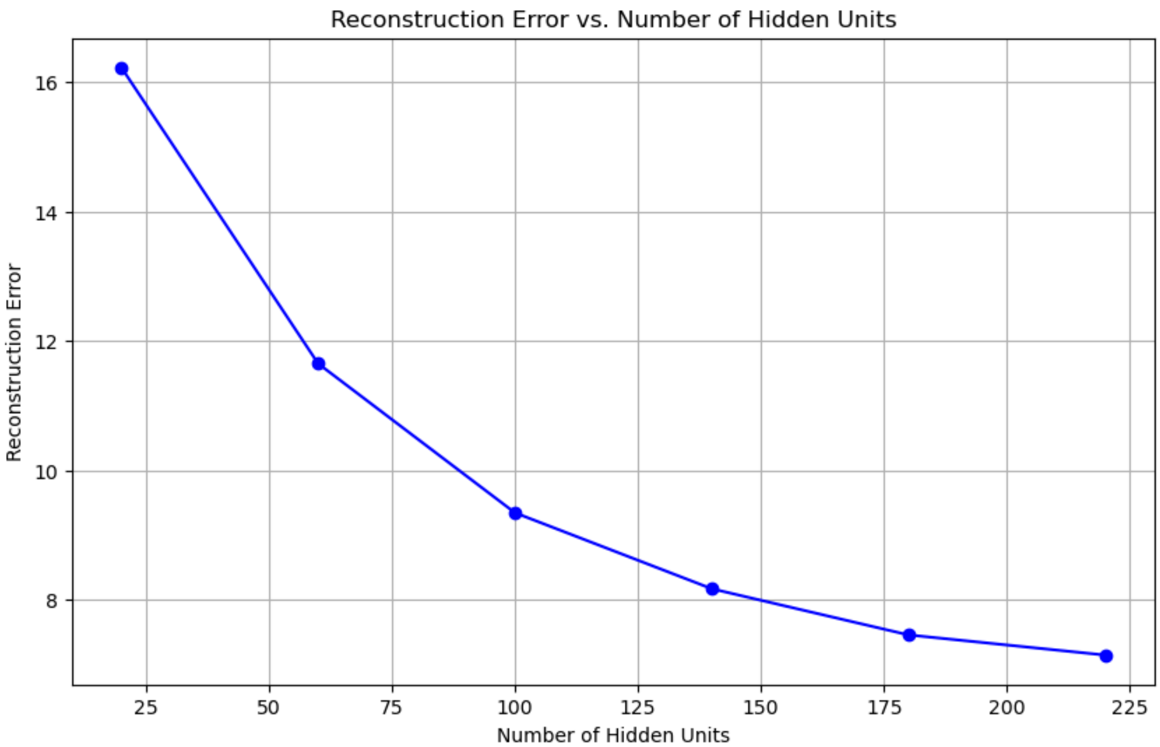
In [8]:

```
epochs = 10
lr = 0.001
activation = 'Tanh'
hidden = [[i] for i in range(20, 221, 40)]
trained_models = []
for h in hidden:
    AE_model, losses = train_autoencoder(
        X_train = np.concatenate((X_train_labeled, X_train_unlabeled),axis=0),
        hidden = h ,
        epochs = epochs , # maximum number of epoches
        activation = activation , # activation function
        lr = lr   #learning rate
    )
    trained_models.append([h,AE_model, losses[-1]])
```

```
Training Model on cuda:0
Epoch 0 out of 10. Loss:24.683003651727105
Epoch 1 out of 10. Loss:21.6570786741591
Epoch 2 out of 10. Loss:20.363965948832405
Epoch 3 out of 10. Loss:19.30880306676491
Epoch 4 out of 10. Loss:18.46648810081875
Epoch 5 out of 10. Loss:17.797488674675066
Epoch 6 out of 10. Loss:17.270203187293614
Epoch 7 out of 10. Loss:16.872740135979406
Epoch 8 out of 10. Loss:16.537404394641364
Epoch 9 out of 10. Loss:16.230592344225066
Training Model on cuda:0
Epoch 0 out of 10. Loss:21.885610285493517
Epoch 1 out of 10. Loss:17.770622548368788
Epoch 2 out of 10. Loss:15.752664300584302
Epoch 3 out of 10. Loss:14.457949933317519
Epoch 4 out of 10. Loss:13.548814390123505
Epoch 5 out of 10. Loss:12.931614158079796
Epoch 6 out of 10. Loss:12.479356057865104
Epoch 7 out of 10. Loss:12.107148032827476
Epoch 8 out of 10. Loss:11.904072250287557
Epoch 9 out of 10. Loss:11.65289321388166
Training Model on cuda:0
Epoch 0 out of 10. Loss:20.341893933483007
Epoch 1 out of 10. Loss:15.43559779334314
Epoch 2 out of 10. Loss:13.342388143244477
Epoch 3 out of 10. Loss:12.13748730335039
Epoch 4 out of 10. Loss:11.358091098746074
Epoch 5 out of 10. Loss:10.761766148596696
Epoch 6 out of 10. Loss:10.25779078178799
Epoch 7 out of 10. Loss:9.864965664971734
Epoch 8 out of 10. Loss:9.56444238879017
Epoch 9 out of 10. Loss:9.346795101755673
Training Model on cuda:0
Epoch 0 out of 10. Loss:19.23789304556306
Epoch 1 out of 10. Loss:14.020434704023538
Epoch 2 out of 10. Loss:11.910471463940807
Epoch 3 out of 10. Loss:10.805243659265262
Epoch 4 out of 10. Loss:10.077409056044116
Epoch 5 out of 10. Loss:9.634704019605499
Epoch 6 out of 10. Loss:9.194839860975128
Epoch 7 out of 10. Loss:8.898982362648875
Epoch 8 out of 10. Loss:8.51799388275933
Epoch 9 out of 10. Loss:8.171858640061211
Training Model on cuda:0
Epoch 0 out of 10. Loss:18.565098310254285
Epoch 1 out of 10. Loss:13.096210066805181
Epoch 2 out of 10. Loss:11.01722405620457
Epoch 3 out of 10. Loss:9.886338774690923
Epoch 4 out of 10. Loss:9.166701936230217
Epoch 5 out of 10. Loss:8.676317731129755
Epoch 6 out of 10. Loss:8.360383800624572
Epoch 7 out of 10. Loss:8.204171362611437
Epoch 8 out of 10. Loss:7.819906947539025
Epoch 9 out of 10. Loss:7.458047222845333
Training Model on cuda:0
Epoch 0 out of 10. Loss:17.897184371948242
Epoch 1 out of 10. Loss:12.315040195111147
Epoch 2 out of 10. Loss:10.31450366973877
Epoch 3 out of 10. Loss:9.27640665191965
```

```
Epoch 4 out of 10. Loss:8.720615858884202
Epoch 5 out of 10. Loss:8.182529916468354
Epoch 6 out of 10. Loss:7.722476831416494
Epoch 7 out of 10. Loss:7.428624330107699
Epoch 8 out of 10. Loss:7.289966642242117
Epoch 9 out of 10. Loss:7.146594352328901
```

In [9]:
```
num_rows, num_columns = X_train_unlabeled.shape
reconstruction_error = []
for h in range(len(hidden)):
    reconstruction_error.append([hidden[h],trained_models[h][2]])
hidden_units = [reconstruction_error[0][0] for reconstruction_error in reconstru
reconstruction_errors = [reconstruction_error[1] for reconstruction_error in rec
# Plotting
plt.figure(figsize=(10, 6))
plt.plot(hidden_units, reconstruction_errors, marker='o', color='b')
plt.title('Reconstruction Error vs. Number of Hidden Units')
plt.xlabel('Number of Hidden Units')
plt.ylabel('Reconstruction Error')
plt.grid(True)
plt.show()
```



Question 3.4 Building 3-Layer NN

In [10]:
```
def train_classifier(X_train,Y_train,hidden,activation='Tanh',epochs=20, trace=F
    train_dl = get_dataloader(X_train,Y_train,autoencoder=False)
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
    # Building the autoencoder
    n_inps = [X_train.shape[-1]]
    n_out = [len(np.unique(Y_train))]   # is not a good idea if you are expectin
    layer_dims = n_inps + hidden + n_out
    layers = []
    try:
        non_linearity = getattr(nn,activation)()
    except AttributeError:
        raise Exception('Activation type not found, note that it is case senstiv
```

```
    for i in range(len(layer_dims)-1):
        layers.extend([nn.Linear(layer_dims[i], layer_dims[i+1]), non_linearity]

    layers.pop()  # to remove the last non-linearity
    model = nn.Sequential(*layers)
    model = model.to(device)
    # to capture training loss
    losses = []
    epoch_losses =[]
    # define optimizer with learning rate
    optim = Adam(model.parameters(),**kwargs)
    # we use MSE error for reconstruction loss
    loss_criterion = nn.CrossEntropyLoss()
    # calculate printing step - optional
    printing_step = int(epochs/10)
    # start training
    for epoch in range(epochs):
        for xb,yb in train_dl:
            preds = model(xb)
            loss = loss_criterion(preds,yb)
            losses.append(loss.item())
            loss.backward()
            optim.step()
            model.zero_grad()
        # after epoch
        epoch_loss = np.mean(losses[-len(train_dl):]) # average loss across all
        epoch_losses.append(epoch_loss)
        if trace and not epoch%printing_step:
            print(f'Epoch {epoch} out of {epochs}. Loss:{epoch_loss}')
    return model, epoch_losses
```

Question 3.4 Fit and test/train error calculation

In [52]:

```
hidden_units = [[i] for i in range(20, 221, 40)]
eta = 0.001
model_nn = []
losses_nn = []
# For each hidden unit number, we train the nn model and calculate the testing e
for h in hidden_units:
    model, epoch_losses = train_classifier(X_train_labeled, y_train_labeled,h)
    model_nn.append(model)
    test_dl = get_dataloader(X_test_labeled,y_test_labeled,autoencoder=False)
    loss_criterion = nn.CrossEntropyLoss()
    for xb,yb in test_dl:
        preds = model(xb)
        loss = loss_criterion(preds,yb)
        losses.append(loss.item())
    total_loss = np.mean(losses[-len(test_dl):])
    losses_nn.append(total_loss)
```

Question 3.5 Augmented self-taught network

In [53]:

```
X_train_learnt = []
X_test_learnt = []
for h in range(len(hidden)):
    # Extract the deep features from layer 1 (middle layer) of the auto encoder,
    X_train_learnt.append([hidden[h],np.concatenate((X_train_labeled,get_deepfea
```

```
    # Pad testing features with 0 to ensure it matches the features required
    zero_features = np.zeros_like(get_deepfeatures(trained_models[h][1],X_test_l
    X_test_learnt.append([hidden[h],np.concatenate((X_test_labeled,zero_features
hidden_units = [i for i in range(20, 221, 40)]
eta = 0.001
model_nn_learnt = []
losses_nn_learnt = []
#Train model with additional features and calculate the cross entropy loss on th
for h in range(len(hidden)):
    model, epoch_losses = train_classifier(X_train_learnt[h][1], y_train_labeled
    model_nn_learnt.append(model)
    test_dl = get_dataloader(X_test_learnt[h][1],y_test_labeled,autoencoder=Fals
    loss_criterion = nn.CrossEntropyLoss()
    for xb,yb in test_dl:
        preds = model(xb)
        loss = loss_criterion(preds,yb)
        losses.append(loss.item())
    total_loss = np.mean(losses[-len(test_dl):])
    losses_nn_learnt.append(total_loss)
```

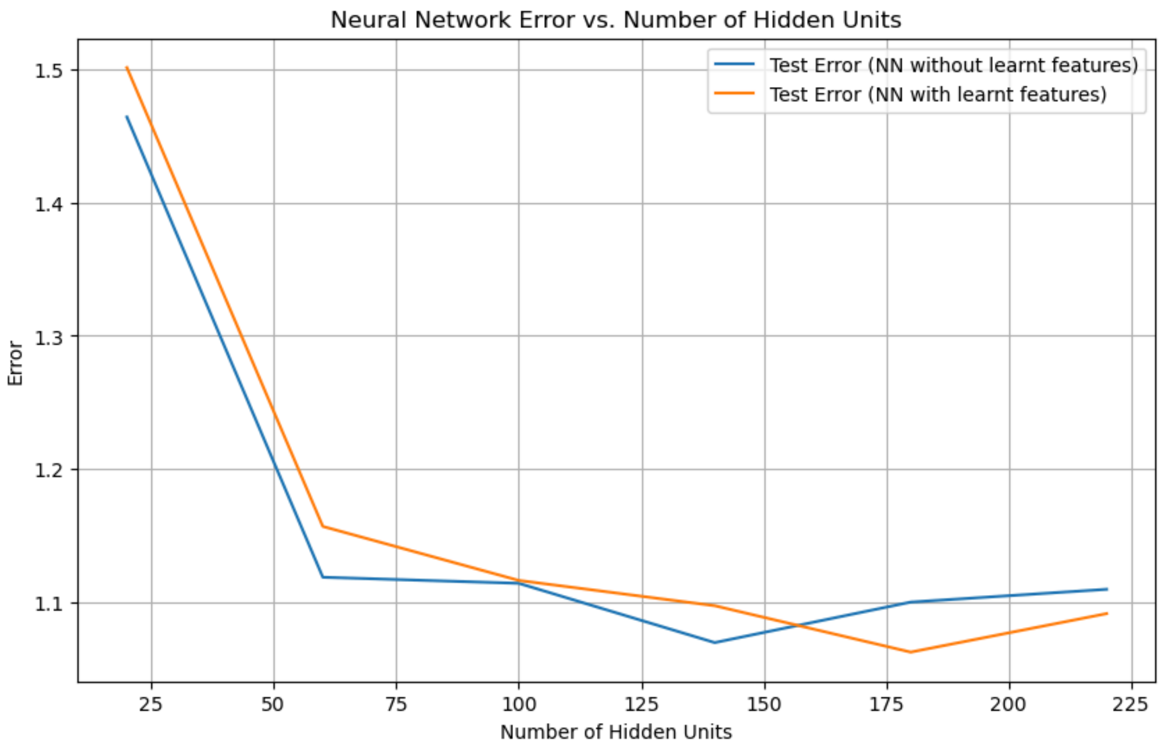
Question 3.6: Plot

In [54]:

```
import matplotlib.pyplot as plt
# Extracting data
plt.figure(figsize=(10, 6))

# Plotting errors for neural networks without learned features
plt.plot(hidden, losses_nn, label='Test Error (NN without learnt features)')
plt.plot(hidden, losses_nn_learnt, label='Test Error (NN with learnt features)')

plt.xlabel('Number of Hidden Units')
plt.ylabel('Error')
plt.title('Neural Network Error vs. Number of Hidden Units')
plt.legend()
plt.grid(True)
plt.show()
```



Question 3.6: Analysis

As we can see, the NN without learnt features starts out with a lower loss throughout the first combinations of hidden units which makes sense since the data is easier to learn which meant that the NN was able to capture important features and generalize well as well. Since lower hidden units represents lower model complexity, the NN can adequately capture the underlying patterns within the features without overfitting or underfitting (which is what we are seeing with the NN with learnt features since the model is too simple to capture the pattern for the additional features).
However, as we increase the number of hidden units, we can see a big improvement in performance for the NN with learnt features as well as a decrease in performence for the one without. An increase in hidden units means the model's complexity increase which benefits the data with more features since it is now able to fully capture the underlying data instead of underfitting like we have seen previously. This increase also caused the NN without learnt feature to overfit and capture noise since the model is too complex for the data that we are using which decreases its generalization performance.