DATA_598_HW_4

January 30, 2022

1

Homework 4: AutoEncoders

Apoorv Sharma

`<center> DATA 598 (Winter 2022), University of Washington </center>`

2 1. Denoising AutoEncoders and Step Decay Learning Rates

In this exercise, we will use autoencoders to denoise (= de-noise, or remove the noise from) an image. We will also implement a step decay learning rate, a commonly used trick (for all deep kinds of deep nets, not just autoencoders).

Suppose we have and image x and *corrupt* it by some means to get $x' = C(x)$. Example corruptions including adding Gaussian noise or deleting random pataches in the image. A denoising autoencoder with encoder h_w and decoder g_v (with respective parameters w and v) takes in the corrupted input x' and returns $\hat{x} = g(h(x'))$ that approximates the noise-less image x.

We will train a denoising autoencoder to reconstruct the noiseless images from the noisy ones, by minimizing the corresponding reconstruction error:

$$\min_{w,v} \mathbb{E} \left|\left|x - g_v \circ h_w(C(x))\right|\right|^2$$

We will use a step decay learning rate schedule

$$\gamma_t = \frac{\gamma_0}{2^{\lfloor t/t_o \rfloor}}$$

in epoch t, where γ_0 is a given initial learning rate, and t_0 threshold. The learning is cut by a factor of 2 every t_0 epochs:

$$\gamma_0 \cdots \gamma_0, \frac{\gamma_0}{2} \cdots \frac{\gamma_0}{2}, \frac{\gamma_0}{4} \cdots \frac{\gamma_0}{4}$$

A larger learning rate makes faster progress initially whereas a smaller learning rate is more helpful closer to convergence. The step-decay schedule aims to get the best of both worlds.

[1]:
```
import torch
from torch.nn.functional import relu
```

```
from torchvision.datasets import MNIST

import numpy as np
import matplotlib.pyplot as plt

import math
import pickle
```

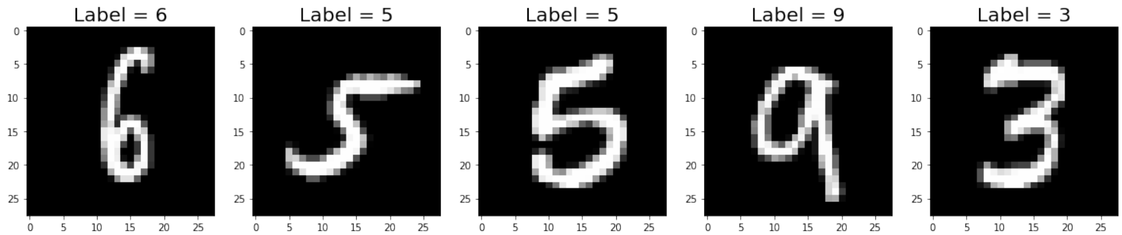
2.0.1 Download and Process MINST dataset

Perform the same preprocessing as in this week's lab.

[2]:
```
# download dataset (~117M in size)
train_dataset = MNIST('./data', train=True, download=True)
X_train = train_dataset.data # torch tensor of type uint8
y_train = train_dataset.targets # torch tensor of type Long
test_dataset = MNIST('./data', train=False, download=True)
X_test = test_dataset.data
y_test = test_dataset.targets

# choose a subsample of 10% of the data:
idxs_train = torch.from_numpy(
    np.random.choice(X_train.shape[0], replace=False, size=X_train.shape[0]//
 ↪10)).long()
X_train, y_train = X_train[idxs_train], y_train[idxs_train]
# idxs_test = torch.from_numpy(
#     np.random.choice(X_test.shape[0], replace=False, size=X_test.shape[0]//
 ↪10))
# X_test, y_test = X_test[idxs_test], y_test[idxs_test]

print(f'X_train.shape = {X_train.shape}')
print(f'n_train: {X_train.shape[0]}, n_test: {X_test.shape[0]}')
print(f'Image size: {X_train.shape[1:]}')
```

```
X_train.shape = torch.Size([6000, 28, 28])
n_train: 6000, n_test: 10000
Image size: torch.Size([28, 28])
```

[3]:
```
f, ax = plt.subplots(1, 5, figsize=(20, 4))
for i, idx in enumerate(np.random.choice(X_train.shape[0], 5)):
    ax[i].imshow(X_train[idx], cmap='gray', vmin=0, vmax=255)
    ax[i].set_title(f'Label = {y_train[idx]}', fontsize=20)
```



[4]:
```python
# NOTE: Run this cell only once. If you have to rerun it multiple times,
# make sure that you run it after rerunning the previous 2 cells.

# Normalize dataset: pixel values lie between 0 and 255
# Normalize them so the pixelwise mean is zero and standard deviation is 1

X_train = X_train.float()  # convert to float32
# NOTE: we are returning a single mean/std over all the pixels, rather than a pixel-wise one
mean, std = X_train.mean(), X_train.std()
X_train = (X_train - mean) / (std + 1e-6)  # avoid divide by zero
# X_train /= torch.norm(X_train, dim=(1, 2)).max()

X_test = X_test.float()
X_test = (X_test - mean) / (std + 1e-6)
# X_test /= torch.norm(X_test, dim=(1, 2)).max()

n_class = np.unique(y_train).shape[0]
```

2.0.2 Create convolutional autoencoder

Use the same convolutional autoencoder as in this week's lab, with a lower latent dimension of 40.

[5]:
```python
class EncoderModule(torch.nn.Module):
    def __init__(self, lower_dimension):
        super().__init__()
        # (B, 1, 28, 28) -> (B, 4, 12, 12)
        self.conv1 = torch.nn.Conv2d(1, 4, kernel_size=5, stride=2, padding=0)
        self.conv2 = torch.nn.Conv2d(4, 8, kernel_size=3, stride=2, padding=0)
        # Flatten (B, 8, 5, 5) -> (B, 8*5*5): do this in `forward()`
        # (B, 8*5*5) -> (B, lower_dimension); 8*5*5 = 200
        self.linear = torch.nn.Linear(200, lower_dimension)

    def forward(self, images):
        out = relu(self.conv1(images))  # conv1 + relu
        out = relu(self.conv2(out))  # conv2 + relu
        out = out.view(out.shape[0], -1)  # flatten
```

```
        out = self.linear(out)  # Linear
        return out
```

[6]:
```
class DecoderModule(torch.nn.Module):
    def __init__(self, lower_dimension):
        super().__init__()
        # (B, lower_dimension) -> (B, linear)
        self.linear_t = torch.nn.Linear(lower_dimension, 200)
        # Unflatten (B, 8*5*5) -> (B, 8, 5, 5); do this in `forward()`
        # (B, 8, 5, 5) -> (B, 4, 12, 12)
        self.conv2_t = torch.nn.ConvTranspose2d(8, 4, kernel_size=3, stride=2,␣
↪padding=0, output_padding=1)
        # (B, 4, 12, 12) -> (B, 1, 28, 28)
        self.conv1_t = torch.nn.ConvTranspose2d(4, 1, kernel_size=5, stride=2,␣
↪padding=0, output_padding=1)

    def forward(self, x):
        # Apply in reverse order
        out = relu(self.linear_t(x))  # linear_t + relu
        out = out.view(out.shape[0], 8, 5, 5)  # Unflatten
        out = relu(self.conv2_t(out))  # conv2_t + relu
        out = self.conv1_t(out)  # conv1_t (note: no relu at the end)
        return out
```

[7]:
```
class AutoEncoder(torch.nn.Module):
    def __init__(self, lower_dimension):
        super().__init__()
        self.encoder = EncoderModule(lower_dimension)
        self.decoder = DecoderModule(lower_dimension)

    def forward(self, images):
        # Pass the images through the encoder to get the representations.
        # Then, pass the representations through the decoder to get the␣
↪reconstructed images
        # images -> encoder(.) -> decoder(.)
        out = self.encoder(images)
        out = self.decoder(out)

        return out

    def encode_images(self, images):
        return self.encoder(images)

    def decode_representations(self, representations):
        return self.decoder(representations)
```

2.0.3 Corruption function

As the corruption function $C(\cdot)$, we zero out a randomly chosen 14×14 patch in the original image

[8]:
```
def corrupt_image_batch(images):
    # Add a 14x14 square of zeros in a 28x28 image
    # images: (B, 1, 28, 28)

    patch_size = 14  # zero out a 14x14 patch
    batch_size = images.shape[0]
    height, width = images.shape[-2:]  # height and width of each image

    starting_h = np.random.choice(height - patch_size, size=batch_size,␣
↪replace=True)
    starting_w = np.random.choice(width - patch_size, size=batch_size,␣
↪replace=True)

    images_corrupted = images.clone()  # corrupt a copy so we do not lose the␣
↪originals
    for b in range(batch_size):
        h = starting_h[b]
        w = starting_w[b]
        images_corrupted[b, 0, h:h+patch_size, b:b+patch_size] = 0  # set to 0
    return images_corrupted
```

2.0.4 Training Functions

[9]:
```
def loss_function(true_images, reconstructed_images):  # square loss
    residual = (true_images - reconstructed_images).view(-1)  # flatten into a␣
↪vector
    # return the average over examples
    return 0.5 * torch.norm(residual) ** 2 / (true_images.shape[0])

def compute_objective(model, original_images, corrupted_images):
    # reshape images from (B, 28, 28) -> (B, 1, 28, 28) as required by the model
    reconstructed_images = model(corrupted_images)
    return loss_function(original_images.unsqueeze(1), reconstructed_images)
```

[10]:
```
@torch.no_grad()
def compute_logs(model, verbose=False): # Only report loss
    train_loss = compute_objective(model, X_train, corrupt_image_batch(X_train.
↪unsqueeze(1)))
    test_loss = compute_objective(model, X_test, corrupt_image_batch(X_test.
↪unsqueeze(1)))
    if verbose:
        print('Train Loss = {:.3f}, Test Loss = {:.3f}'.format(
                train_loss.item(), test_loss.item(),
```

```
    ))
    return (train_loss, test_loss)

def minibatch_sgd_one_pass(model, X, learning_rate, batch_size, verbose=False):
    num_examples = X.shape[0]
    average_loss = 0.0
    num_updates = int(round(num_examples / batch_size))
    for i in range(num_updates):
        idxs = np.random.choice(num_examples, size=(batch_size,))

        corrupted_images = corrupt_image_batch(X[idxs].unsqueeze(1))
        # compute the objective.
        objective = compute_objective(model, X[idxs], corrupted_images)
        average_loss = 0.99 * average_loss + 0.01 * objective.item()
        if verbose and (i+1) % 100 == 0:
            print("{:.3f}".format(average_loss))

        # Exercise:
        # compute the gradient using automatic differentiation
        gradients = torch.autograd.grad(outputs=objective, inputs=model.
↪parameters())

        # Perform the SGD update
        with torch.no_grad():
            for (w, g) in zip(model.parameters(), gradients):
                w -= learning_rate * g

    return model
```

2.0.5 Train Model

Train the model for 40 epochs starting with $\gamma_0 = 2.5 \cdot 10^{-4}$ and $t_0 = 10$ (i.e., halve the learning rate every 10 epochs).

[11]:
```
initial_learning_rate = 2.5e-4
learning_rate_threshold = 10
batch_size = 1
lower_dimension = 40 # use a lower dimensionality of 40
num_epochs = 40

logs = []

model = AutoEncoder(lower_dimension)
print(f'Iteration 0, LR: {initial_learning_rate}', end=', ')
logs.append(compute_logs(model, verbose=True))

for j in range(num_epochs):
```

```
    # step decay learning rate schedule
    num_epoch = j + 1
    learning_rate = initial_learning_rate / ( math.pow(2, math.floor(num_epoch/
 ↪learning_rate_threshold)) )

    model = minibatch_sgd_one_pass(model, X_train, learning_rate,␣
 ↪batch_size=batch_size, verbose=False)
    print(f'Iteration {num_epoch}, LR: {learning_rate}', end=', ')
    logs.append(compute_logs(model, verbose=True))

with open('./models/logs.pkl', 'wb') as f:
    pickle.dump(logs, f)

# save the model parms
torch.save(model.state_dict(), f'./models/parms.pt')
```

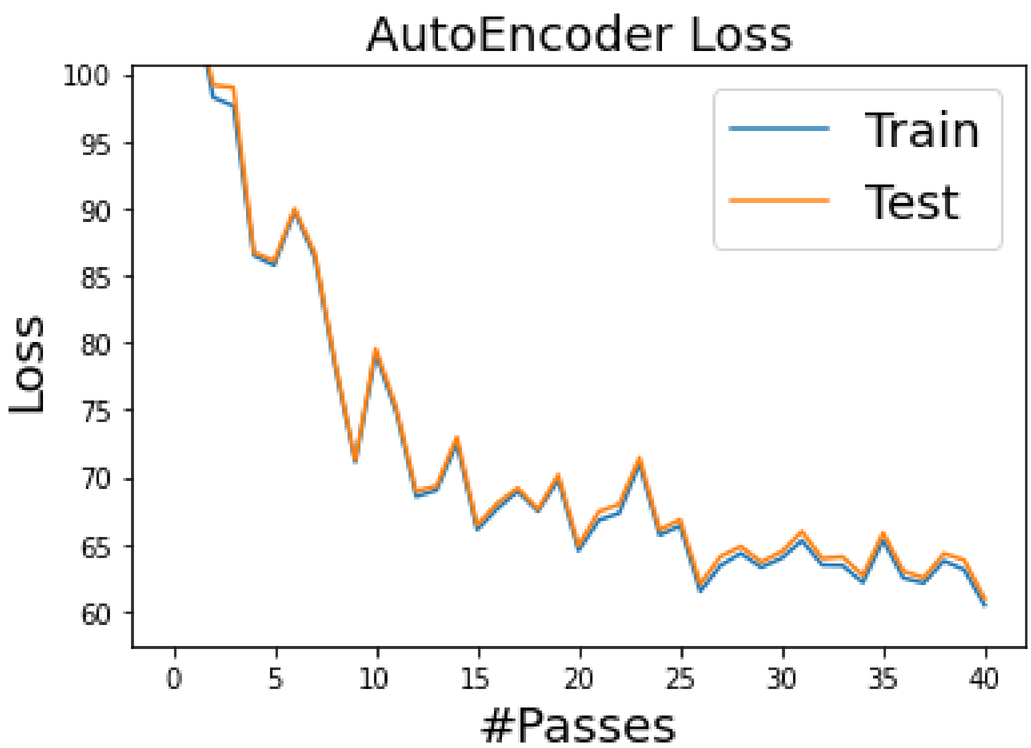
```
Iteration 0, LR: 0.00025, Train Loss = 403.148, Test Loss = 409.312
Iteration 1, LR: 0.00025, Train Loss = 105.786, Test Loss = 105.933
Iteration 2, LR: 0.00025, Train Loss = 98.256, Test Loss = 99.154
Iteration 3, LR: 0.00025, Train Loss = 97.623, Test Loss = 98.991
Iteration 4, LR: 0.00025, Train Loss = 86.478, Test Loss = 86.705
Iteration 5, LR: 0.00025, Train Loss = 85.741, Test Loss = 86.091
Iteration 6, LR: 0.00025, Train Loss = 89.762, Test Loss = 89.973
Iteration 7, LR: 0.00025, Train Loss = 86.330, Test Loss = 86.676
Iteration 8, LR: 0.00025, Train Loss = 78.222, Test Loss = 78.539
Iteration 9, LR: 0.00025, Train Loss = 71.088, Test Loss = 71.248
Iteration 10, LR: 0.000125, Train Loss = 79.126, Test Loss = 79.552
Iteration 11, LR: 0.000125, Train Loss = 74.888, Test Loss = 75.221
Iteration 12, LR: 0.000125, Train Loss = 68.541, Test Loss = 68.927
Iteration 13, LR: 0.000125, Train Loss = 69.009, Test Loss = 69.327
Iteration 14, LR: 0.000125, Train Loss = 72.473, Test Loss = 72.941
Iteration 15, LR: 0.000125, Train Loss = 66.059, Test Loss = 66.402
Iteration 16, LR: 0.000125, Train Loss = 67.637, Test Loss = 68.053
Iteration 17, LR: 0.000125, Train Loss = 68.926, Test Loss = 69.170
Iteration 18, LR: 0.000125, Train Loss = 67.436, Test Loss = 67.592
Iteration 19, LR: 0.000125, Train Loss = 69.775, Test Loss = 70.145
Iteration 20, LR: 6.25e-05, Train Loss = 64.511, Test Loss = 64.899
Iteration 21, LR: 6.25e-05, Train Loss = 66.767, Test Loss = 67.420
Iteration 22, LR: 6.25e-05, Train Loss = 67.294, Test Loss = 67.952
Iteration 23, LR: 6.25e-05, Train Loss = 70.986, Test Loss = 71.420
Iteration 24, LR: 6.25e-05, Train Loss = 65.656, Test Loss = 66.042
Iteration 25, LR: 6.25e-05, Train Loss = 66.357, Test Loss = 66.819
Iteration 26, LR: 6.25e-05, Train Loss = 61.469, Test Loss = 62.023
Iteration 27, LR: 6.25e-05, Train Loss = 63.393, Test Loss = 64.070
Iteration 28, LR: 6.25e-05, Train Loss = 64.305, Test Loss = 64.820
Iteration 29, LR: 6.25e-05, Train Loss = 63.250, Test Loss = 63.656
Iteration 30, LR: 3.125e-05, Train Loss = 63.937, Test Loss = 64.424
```

```
Iteration 31, LR: 3.125e-05, Train Loss = 65.244, Test Loss = 65.932
Iteration 32, LR: 3.125e-05, Train Loss = 63.428, Test Loss = 63.922
Iteration 33, LR: 3.125e-05, Train Loss = 63.397, Test Loss = 64.042
Iteration 34, LR: 3.125e-05, Train Loss = 62.125, Test Loss = 62.669
Iteration 35, LR: 3.125e-05, Train Loss = 65.279, Test Loss = 65.835
Iteration 36, LR: 3.125e-05, Train Loss = 62.458, Test Loss = 62.966
Iteration 37, LR: 3.125e-05, Train Loss = 62.087, Test Loss = 62.481
Iteration 38, LR: 3.125e-05, Train Loss = 63.737, Test Loss = 64.289
Iteration 39, LR: 3.125e-05, Train Loss = 63.072, Test Loss = 63.793
Iteration 40, LR: 1.5625e-05, Train Loss = 60.442, Test Loss = 60.926
```

View the loss

[12]:
```
logs_arr = np.asarray(logs)

plt.plot(logs_arr[:, 0], label="Train")
plt.plot(logs_arr[:, 1], label="Test")
plt.title('AutoEncoder Loss', fontsize=18)
plt.ylabel('Loss', fontsize=18)
plt.xlabel('#Passes', fontsize=18)
plt.legend(fontsize=18)

# Try to set clever bounds
plt.ylim((logs_arr[1:].reshape(-1).min() * 0.95, logs_arr[1:].reshape(-1).max()
 ↪* 0.95))
```

[12]:
```
(57.420057106018064, 100.63650093078613)
```

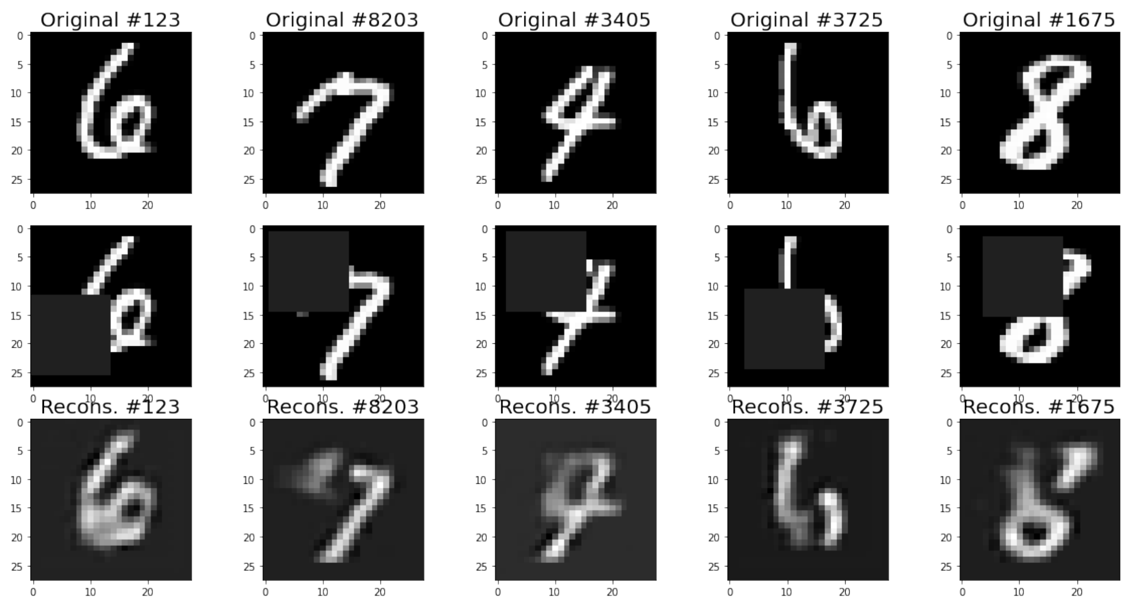


2.0.6 Model Output

Show some examples of the denoising process from the test set.

[13]:
```
f, ax = plt.subplots(3, 5, figsize=(20, 10))

idxs = np.random.choice(X_test.shape[0], 5)
images = X_test[idxs].unsqueeze(1)
images_corrupted = corrupt_image_batch(images)

for i, idx in enumerate(idxs):
    ax[0, i].imshow(images[i].squeeze() * std + mean, cmap='gray')  # Note:␣
 ↪Undo mean and std normalization before viewing image
    ax[0, i].set_title(f'Original #{idx}', fontsize=20)

    ax[1, i].imshow(images_corrupted[i].squeeze() * std + mean, cmap='gray')  #␣
 ↪Note: Undo mean and std normalization before viewing image
    ax[2, i].set_title(f'Corrupted #{idx}', fontsize=20)

    # add batch and channel dimensions before passing through the model and␣
 ↪squeeze them out later
    xr = model(images_corrupted[i].view(1, 1, 28, 28)).detach().squeeze()
```

```
    ax[2, i].imshow(xr * std + mean, cmap='gray')  # Note: Undo mean and std normalization before viewing image
    ax[2, i].set_title(f'Recons. #{idx}', fontsize=20)
```



3 2. (Bonus) AutoEncoders as a non-linear PCA

In this exercise, we will compare autoencoders versus PCA for dimensionality reduction. We will note their usefulness on the end goal of training a linear model using the extracted low-dimensional features.

In the first few labs, we used images as 784 dimensional vectors. Here, you will use either autoencoders or PCA on the training dataset to project the data onto a lower dimension d. You will then train a multinomial logistic regression model with scikit-learn and keep track of the test accuracy.

[14]:
```
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
```

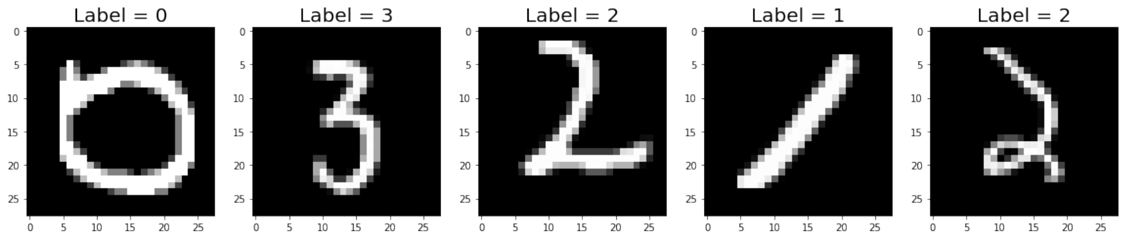
[15]:
```
num_components_list = [10, 25, 50, 100]
```

3.0.1 Download and Process MINST dataset

Perform the same preprocessing as in this week's lab.

[16]:
```
f, ax = plt.subplots(1, 5, figsize=(20, 4))
for i, idx in enumerate(np.random.choice(X_train.shape[0], 5)):
    ax[i].imshow(X_train[idx].squeeze() * std + mean, cmap='gray', vmin=0, vmax=255)
```

```
    ax[i].set_title(f'Label = {y_train[idx]}', fontsize=20)
```

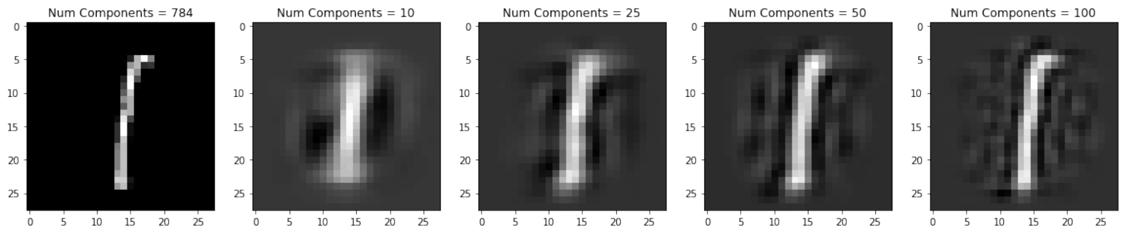


3.0.2 PCA Projections

Given a lower dimension d, use PCA to reduce the dimensionality of the training set to d dimensions. Transform the test set by projecting on to the same space. You may use scikit-learn's PCA implementation.

[17]:
```
X_train_2d = X_train.view((X_train.shape[0], -1))
X_test_2d = X_test.view((X_test.shape[0], -1))
```

[18]:
```
f, ax = plt.subplots(1, len(num_components_list)+1, figsize=(20, 10))
idx = np.random.choice(X_train_2d.shape[0], 1)

ax[0].imshow(X_train[idx].squeeze() * std + mean, cmap='gray')  # Note: Undo mean and std normalization before viewing image
ax[0].set_title(f'Num Components = {28*28}', fontsize=12)

for i, num_components in enumerate(num_components_list):
    pca = PCA(n_components = num_components)
    principalComponents = pca.fit(X_train_2d)

    # Apply transform to both the training set and the test set.
    train_img = pca.transform(X_train_2d)
    test_img = pca.transform(X_test_2d)

    approximation = pca.inverse_transform(train_img)

    ax[i+1].set_title(f'Num Components = {num_components}', fontsize=12)
    ax[i+1].imshow(torch.tensor(approximation[idx].reshape(28,28).squeeze()) * std + mean, cmap='gray')  # Note: Undo mean and std normalization before viewing image
```



3.0.3 Multinomial logistic Regression

[19]:
```python
def get_accuracy(y_true, y_pred):
    return np.mean(y_pred == y_true)
```

[20]:
```python
log_reg_model = LogisticRegression(max_iter=200)
log_reg_model.fit(train_img, y_train)
```

```
/home/apoorvsharma/anaconda3/envs/data598/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[20]:
```
LogisticRegression(max_iter=200)
```

[21]:
```python
y_pred = log_reg_model.predict(test_img)
print(get_accuracy(y_pred, y_test.numpy()) * 100)
```

```
89.68
```

3.0.4 Define AutoEncoder and Training Functions

These need to be re-written/re-defined since they were changed for #1.

The same

[22]:
```python
def loss_function(true_images, reconstructed_images):   # square loss
    residual = (true_images - reconstructed_images).view(-1)   # flatten into a␣
↪vector
    # return the average over examples
    return 0.5 * torch.norm(residual) ** 2 / (true_images.shape[0])
```

```
def compute_objective(model, images):
    # reshape images from (B, 28, 28) -> (B, 1, 28, 28) as required by the model
    images = images.unsqueeze(1)  # Add channel dimension
    reconstructed_images = model(images)
    return loss_function(images, reconstructed_images)
```

[23]:
```
@torch.no_grad()
def compute_logs(model, verbose=False): # Only report loss
    train_loss = compute_objective(model, X_train)
    test_loss = compute_objective(model, X_test)
    if verbose:
        print('Train Loss = {:.3f}, Test Loss = {:.3f}'.format(
                train_loss.item(), test_loss.item(),
    ))
    return (train_loss, test_loss)

def minibatch_sgd_one_pass(model, X, learning_rate, batch_size, verbose=False):
    num_examples = X.shape[0]
    average_loss = 0.0
    num_updates = int(round(num_examples / batch_size))
    for i in range(num_updates):
        idxs = np.random.choice(num_examples, size=(batch_size,))
        # compute the objective.
        objective = compute_objective(model, X[idxs])
        average_loss = 0.99 * average_loss + 0.01 * objective.item()
        if verbose and (i+1) % 100 == 0:
            print("{:.3f}".format(average_loss))

        # Exercise:
        # compute the gradient using automatic differentiation
        gradients = torch.autograd.grad(outputs=objective, inputs=model.
 ↪parameters())

        # Perform the SGD update
        with torch.no_grad():
            for (w, g) in zip(model.parameters(), gradients):
                w -= learning_rate * g

    return model
```

3.0.5 Dimension Reduction Analysis

Here we perform the following steps, for the following values of d `[10, 25, 50, 100]`: 1. Given a lower dimension d, use PCA to reduce the dimensionality of the training set to d dimensions. Transform the test set by projecting on to the same space. 2. Train an autoencoder using the same settings as the lab, but with a hidden dimension as d. Train it for 40 epochs. Use the encoder to obtain d-dimensional representations for all training and test images. 3. Train a multinomial logistic regression model with scikit-learn using each of the representations you have obtained.

[28]:
```
initial_learning_rate = 2.5e-4
learning_rate_threshold = 10
batch_size = 1
num_epochs = 40
```

[29]:
```
pca_accuracy = []
ae_accuracy = []
```

[30]:
```
for i, num_components in enumerate(num_components_list):

    print(f'Starting dimension reduction to {num_components} components')

    # STEP 1: Dimension Reduction using PCA
    print(f'\tStarting PCA')
    pca = PCA(n_components = num_components)
    principalComponents = pca.fit(X_train_2d)

    # Apply transform to both the training set and the test set.
    train_img_pca = pca.transform(X_train_2d)
    test_img_pca = pca.transform(X_test_2d)
    print(f'\tFinished PCA')

    # STEP 2: Dimension Reduction using AutoEncoder
    print(f'\tStarting AutoEncoder Training')
    logs = []
    model = AutoEncoder(num_components)
    print(f'\t\tIteration 0, LR: {initial_learning_rate}', end=', ')
    logs.append(compute_logs(model, verbose=True))

    for j in range(num_epochs):
        # step decay learning rate schedule
        num_epoch = j + 1
        learning_rate = initial_learning_rate / ( math.pow(2, math.
↪floor(num_epoch/learning_rate_threshold)) )

        model = minibatch_sgd_one_pass(model, X_train, learning_rate,␣
↪batch_size=batch_size, verbose=False)
        print(f'\t\tIteration {num_epoch}, LR: {learning_rate}', end=', ')
        logs.append(compute_logs(model, verbose=True))

    with open('./models/q2_logs_nc_{num_components}.pkl', 'wb') as f:
        pickle.dump(logs, f)

    # save the model parms
    torch.save(model.state_dict(), f'./models/q2_parms_nc_{num_components}.pt')
```

```python
    # Obtain encoded traina and test images
    train_img_ae = model.encode_images(X_train.unsqueeze(1))
    test_img_ae = model.encode_images(X_test.unsqueeze(1))

    print(f'\tDone AutoEncoder Training')

    # STEP 3a Create Logistic Regression Models for each reduction method
    log_reg_model_pca = LogisticRegression(max_iter=200)
    log_reg_model_pca.fit(train_img_pca, y_train)

    log_reg_model_ae = LogisticRegression(max_iter=200)
    log_reg_model_ae.fit(train_img_ae.detach().numpy(), y_train)

    # STEP 3b: Predict and get accuracy from each model
    # Accuracy for PCA model
    y_pred = log_reg_model_pca.predict(test_img_pca)
    pca_accuracy.append(get_accuracy(y_pred, y_test.numpy()) * 100)

    # Accuracy for AutoEncoder model
    y_pred = log_reg_model_ae.predict(test_img_ae.detach().numpy())
    ae_accuracy.append(get_accuracy(y_pred, y_test.numpy()) * 100)

    print(f'AE Acc: {ae_accuracy[-1]:.2f}, PCA Acc: {pca_accuracy[-1]:.2f}')
```

```
Starting dimension reduction to 10 components
        Starting PCA
        Finished PCA
        Starting AutoEncoder Training
                Iteration 0, LR: 0.00025, Train Loss = 398.408, Test Loss =
406.882
                Iteration 1, LR: 0.00025, Train Loss = 388.082, Test Loss =
395.253
                Iteration 2, LR: 0.00025, Train Loss = 125.771, Test Loss =
126.142
                Iteration 3, LR: 0.00025, Train Loss = 118.647, Test Loss =
118.829
                Iteration 4, LR: 0.00025, Train Loss = 121.273, Test Loss =
121.887
                Iteration 5, LR: 0.00025, Train Loss = 114.497, Test Loss =
114.543
                Iteration 6, LR: 0.00025, Train Loss = 112.021, Test Loss =
112.409
                Iteration 7, LR: 0.00025, Train Loss = 110.072, Test Loss =
110.112
                Iteration 8, LR: 0.00025, Train Loss = 108.094, Test Loss =
108.636
```

Iteration 9, LR: 0.00025, Train Loss = 110.723, Test Loss = 111.919

Iteration 10, LR: 0.000125, Train Loss = 103.235, Test Loss = 104.026

Iteration 11, LR: 0.000125, Train Loss = 101.062, Test Loss = 101.586

Iteration 12, LR: 0.000125, Train Loss = 101.835, Test Loss = 102.618

Iteration 13, LR: 0.000125, Train Loss = 101.978, Test Loss = 102.940

Iteration 14, LR: 0.000125, Train Loss = 100.311, Test Loss = 101.439

Iteration 15, LR: 0.000125, Train Loss = 100.948, Test Loss = 101.921

Iteration 16, LR: 0.000125, Train Loss = 100.228, Test Loss = 101.313

Iteration 17, LR: 0.000125, Train Loss = 100.060, Test Loss = 101.000

Iteration 18, LR: 0.000125, Train Loss = 99.171, Test Loss = 99.822

Iteration 19, LR: 0.000125, Train Loss = 98.942, Test Loss = 99.748

Iteration 20, LR: 6.25e-05, Train Loss = 96.578, Test Loss = 97.640

Iteration 21, LR: 6.25e-05, Train Loss = 96.684, Test Loss = 97.898

Iteration 22, LR: 6.25e-05, Train Loss = 96.339, Test Loss = 97.371

Iteration 23, LR: 6.25e-05, Train Loss = 96.418, Test Loss = 97.486

Iteration 24, LR: 6.25e-05, Train Loss = 95.683, Test Loss = 96.701

Iteration 25, LR: 6.25e-05, Train Loss = 95.896, Test Loss = 97.046

Iteration 26, LR: 6.25e-05, Train Loss = 95.799, Test Loss = 96.880

Iteration 27, LR: 6.25e-05, Train Loss = 95.466, Test Loss = 96.521

Iteration 28, LR: 6.25e-05, Train Loss = 96.302, Test Loss = 97.360

Iteration 29, LR: 6.25e-05, Train Loss = 95.937, Test Loss = 96.938

Iteration 30, LR: 3.125e-05, Train Loss = 94.265, Test Loss = 95.533

Iteration 31, LR: 3.125e-05, Train Loss = 94.139, Test Loss = 95.275

Iteration 32, LR: 3.125e-05, Train Loss = 94.069, Test Loss = 95.372

```
                Iteration 33, LR: 3.125e-05, Train Loss = 94.028, Test Loss =
95.211
                Iteration 34, LR: 3.125e-05, Train Loss = 94.090, Test Loss =
95.221
                Iteration 35, LR: 3.125e-05, Train Loss = 93.887, Test Loss =
95.229
                Iteration 36, LR: 3.125e-05, Train Loss = 93.858, Test Loss =
94.989
                Iteration 37, LR: 3.125e-05, Train Loss = 93.658, Test Loss =
94.879
                Iteration 38, LR: 3.125e-05, Train Loss = 93.673, Test Loss =
94.732
                Iteration 39, LR: 3.125e-05, Train Loss = 93.820, Test Loss =
95.022
                Iteration 40, LR: 1.5625e-05, Train Loss = 93.179, Test Loss =
94.494
        Done AutoEncoder Training

/home/apoorvsharma/anaconda3/envs/data598/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/home/apoorvsharma/anaconda3/envs/data598/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(

AE Acc: 88.13, PCA Acc: 80.59
Starting dimension reduction to 25 components
        Starting PCA
        Finished PCA
        Starting AutoEncoder Training
                Iteration 0, LR: 0.00025, Train Loss = 392.124, Test Loss =
399.276
                Iteration 1, LR: 0.00025, Train Loss = 110.442, Test Loss =
```

109.876
Iteration 2, LR: 0.00025, Train Loss = 77.722, Test Loss =
77.165
Iteration 3, LR: 0.00025, Train Loss = 73.069, Test Loss =
72.832
Iteration 4, LR: 0.00025, Train Loss = 72.282, Test Loss =
71.602
Iteration 5, LR: 0.00025, Train Loss = 70.758, Test Loss =
70.191
Iteration 6, LR: 0.00025, Train Loss = 65.048, Test Loss =
64.995
Iteration 7, LR: 0.00025, Train Loss = 62.681, Test Loss =
62.431
Iteration 8, LR: 0.00025, Train Loss = 61.486, Test Loss =
61.192
Iteration 9, LR: 0.00025, Train Loss = 61.099, Test Loss =
61.214
Iteration 10, LR: 0.000125, Train Loss = 57.536, Test Loss =
57.585
Iteration 11, LR: 0.000125, Train Loss = 56.843, Test Loss =
56.884
Iteration 12, LR: 0.000125, Train Loss = 56.359, Test Loss =
56.382
Iteration 13, LR: 0.000125, Train Loss = 56.539, Test Loss =
56.739
Iteration 14, LR: 0.000125, Train Loss = 56.082, Test Loss =
56.090
Iteration 15, LR: 0.000125, Train Loss = 55.765, Test Loss =
55.853
Iteration 16, LR: 0.000125, Train Loss = 55.421, Test Loss =
55.634
Iteration 17, LR: 0.000125, Train Loss = 55.756, Test Loss =
55.947
Iteration 18, LR: 0.000125, Train Loss = 55.017, Test Loss =
55.174
Iteration 19, LR: 0.000125, Train Loss = 55.691, Test Loss =
55.674
Iteration 20, LR: 6.25e-05, Train Loss = 53.242, Test Loss =
53.456
Iteration 21, LR: 6.25e-05, Train Loss = 53.222, Test Loss =
53.410
Iteration 22, LR: 6.25e-05, Train Loss = 53.244, Test Loss =
53.376
Iteration 23, LR: 6.25e-05, Train Loss = 53.491, Test Loss =
53.672
Iteration 24, LR: 6.25e-05, Train Loss = 52.973, Test Loss =
53.179
Iteration 25, LR: 6.25e-05, Train Loss = 53.386, Test Loss =

```
53.583
                Iteration 26, LR: 6.25e-05, Train Loss = 52.634, Test Loss =
52.852
                Iteration 27, LR: 6.25e-05, Train Loss = 52.682, Test Loss =
52.903
                Iteration 28, LR: 6.25e-05, Train Loss = 53.113, Test Loss =
53.432
                Iteration 29, LR: 6.25e-05, Train Loss = 52.535, Test Loss =
52.723
                Iteration 30, LR: 3.125e-05, Train Loss = 51.927, Test Loss =
52.209
                Iteration 31, LR: 3.125e-05, Train Loss = 51.827, Test Loss =
52.125
                Iteration 32, LR: 3.125e-05, Train Loss = 51.801, Test Loss =
52.100
                Iteration 33, LR: 3.125e-05, Train Loss = 51.868, Test Loss =
52.159
                Iteration 34, LR: 3.125e-05, Train Loss = 51.747, Test Loss =
52.049
                Iteration 35, LR: 3.125e-05, Train Loss = 51.760, Test Loss =
52.059
                Iteration 36, LR: 3.125e-05, Train Loss = 51.637, Test Loss =
51.883
                Iteration 37, LR: 3.125e-05, Train Loss = 51.615, Test Loss =
51.946
                Iteration 38, LR: 3.125e-05, Train Loss = 51.505, Test Loss =
51.866
                Iteration 39, LR: 3.125e-05, Train Loss = 51.488, Test Loss =
51.840
                Iteration 40, LR: 1.5625e-05, Train Loss = 51.186, Test Loss =
51.570
        Done AutoEncoder Training
```

```
/home/apoorvsharma/anaconda3/envs/data598/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/home/apoorvsharma/anaconda3/envs/data598/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
AE Acc: 90.73, PCA Acc: 88.52
Starting dimension reduction to 50 components
        Starting PCA
        Finished PCA
        Starting AutoEncoder Training
                Iteration 0, LR: 0.00025, Train Loss = 393.328, Test Loss =
400.636
                Iteration 1, LR: 0.00025, Train Loss = 76.671, Test Loss =
76.322
                Iteration 2, LR: 0.00025, Train Loss = 62.838, Test Loss =
62.395
                Iteration 3, LR: 0.00025, Train Loss = 57.246, Test Loss =
56.715
                Iteration 4, LR: 0.00025, Train Loss = 54.207, Test Loss =
54.016
                Iteration 5, LR: 0.00025, Train Loss = 75.574, Test Loss =
75.448
                Iteration 6, LR: 0.00025, Train Loss = 50.080, Test Loss =
49.592
                Iteration 7, LR: 0.00025, Train Loss = 53.834, Test Loss =
54.251
                Iteration 8, LR: 0.00025, Train Loss = 46.501, Test Loss =
46.436
                Iteration 9, LR: 0.00025, Train Loss = 45.986, Test Loss =
46.062
                Iteration 10, LR: 0.000125, Train Loss = 39.784, Test Loss =
39.671
                Iteration 11, LR: 0.000125, Train Loss = 39.110, Test Loss =
38.953
                Iteration 12, LR: 0.000125, Train Loss = 38.691, Test Loss =
38.656
                Iteration 13, LR: 0.000125, Train Loss = 38.134, Test Loss =
38.016
                Iteration 14, LR: 0.000125, Train Loss = 38.228, Test Loss =
38.085
                Iteration 15, LR: 0.000125, Train Loss = 37.777, Test Loss =
37.737
                Iteration 16, LR: 0.000125, Train Loss = 37.115, Test Loss =
37.096
                Iteration 17, LR: 0.000125, Train Loss = 36.782, Test Loss =
```

```
36.758
                Iteration 18, LR: 0.000125, Train Loss = 36.945, Test Loss =
36.935
                Iteration 19, LR: 0.000125, Train Loss = 36.630, Test Loss =
36.673
                Iteration 20, LR: 6.25e-05, Train Loss = 35.518, Test Loss =
35.549
                Iteration 21, LR: 6.25e-05, Train Loss = 35.370, Test Loss =
35.382
                Iteration 22, LR: 6.25e-05, Train Loss = 35.105, Test Loss =
35.116
                Iteration 23, LR: 6.25e-05, Train Loss = 35.148, Test Loss =
35.179
                Iteration 24, LR: 6.25e-05, Train Loss = 34.953, Test Loss =
34.986
                Iteration 25, LR: 6.25e-05, Train Loss = 34.871, Test Loss =
34.940
                Iteration 26, LR: 6.25e-05, Train Loss = 34.837, Test Loss =
34.886
                Iteration 27, LR: 6.25e-05, Train Loss = 34.756, Test Loss =
34.792
                Iteration 28, LR: 6.25e-05, Train Loss = 34.764, Test Loss =
34.795
                Iteration 29, LR: 6.25e-05, Train Loss = 34.699, Test Loss =
34.778
                Iteration 30, LR: 3.125e-05, Train Loss = 34.240, Test Loss =
34.283
                Iteration 31, LR: 3.125e-05, Train Loss = 34.187, Test Loss =
34.240
                Iteration 32, LR: 3.125e-05, Train Loss = 34.183, Test Loss =
34.273
                Iteration 33, LR: 3.125e-05, Train Loss = 34.084, Test Loss =
34.143
                Iteration 34, LR: 3.125e-05, Train Loss = 34.048, Test Loss =
34.126
                Iteration 35, LR: 3.125e-05, Train Loss = 34.067, Test Loss =
34.140
                Iteration 36, LR: 3.125e-05, Train Loss = 33.938, Test Loss =
33.999
                Iteration 37, LR: 3.125e-05, Train Loss = 33.972, Test Loss =
34.066
                Iteration 38, LR: 3.125e-05, Train Loss = 33.906, Test Loss =
33.977
                Iteration 39, LR: 3.125e-05, Train Loss = 33.930, Test Loss =
34.031
                Iteration 40, LR: 1.5625e-05, Train Loss = 33.737, Test Loss =
33.846
    Done AutoEncoder Training
```

```
/home/apoorvsharma/anaconda3/envs/data598/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/home/apoorvsharma/anaconda3/envs/data598/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
AE Acc: 90.40, PCA Acc: 89.98
Starting dimension reduction to 100 components
        Starting PCA
        Finished PCA
        Starting AutoEncoder Training
                Iteration 0, LR: 0.00025, Train Loss = 400.061, Test Loss =
411.124
                Iteration 1, LR: 0.00025, Train Loss = 67.329, Test Loss =
66.640
                Iteration 2, LR: 0.00025, Train Loss = 56.515, Test Loss =
56.282
                Iteration 3, LR: 0.00025, Train Loss = 45.893, Test Loss =
45.705
                Iteration 4, LR: 0.00025, Train Loss = 76.234, Test Loss =
76.851
                Iteration 5, LR: 0.00025, Train Loss = 40.744, Test Loss =
40.700
                Iteration 6, LR: 0.00025, Train Loss = 43.252, Test Loss =
43.148
                Iteration 7, LR: 0.00025, Train Loss = 38.617, Test Loss =
38.480
                Iteration 8, LR: 0.00025, Train Loss = 39.971, Test Loss =
40.016
                Iteration 9, LR: 0.00025, Train Loss = 37.326, Test Loss =
37.274
```

Iteration 10, LR: 0.000125, Train Loss = 32.580, Test Loss = 32.562
Iteration 11, LR: 0.000125, Train Loss = 32.723, Test Loss = 32.687
Iteration 12, LR: 0.000125, Train Loss = 31.059, Test Loss = 31.100
Iteration 13, LR: 0.000125, Train Loss = 29.890, Test Loss = 29.885
Iteration 14, LR: 0.000125, Train Loss = 33.256, Test Loss = 33.379
Iteration 15, LR: 0.000125, Train Loss = 28.503, Test Loss = 28.517
Iteration 16, LR: 0.000125, Train Loss = 28.047, Test Loss = 28.110
Iteration 17, LR: 0.000125, Train Loss = 27.495, Test Loss = 27.527
Iteration 18, LR: 0.000125, Train Loss = 27.164, Test Loss = 27.194
Iteration 19, LR: 0.000125, Train Loss = 27.145, Test Loss = 27.188
Iteration 20, LR: 6.25e-05, Train Loss = 25.625, Test Loss = 25.715
Iteration 21, LR: 6.25e-05, Train Loss = 25.671, Test Loss = 25.715
Iteration 22, LR: 6.25e-05, Train Loss = 25.602, Test Loss = 25.677
Iteration 23, LR: 6.25e-05, Train Loss = 25.323, Test Loss = 25.429
Iteration 24, LR: 6.25e-05, Train Loss = 25.313, Test Loss = 25.385
Iteration 25, LR: 6.25e-05, Train Loss = 25.209, Test Loss = 25.301
Iteration 26, LR: 6.25e-05, Train Loss = 25.052, Test Loss = 25.142
Iteration 27, LR: 6.25e-05, Train Loss = 24.830, Test Loss = 24.948
Iteration 28, LR: 6.25e-05, Train Loss = 24.860, Test Loss = 24.994
Iteration 29, LR: 6.25e-05, Train Loss = 24.748, Test Loss = 24.886
Iteration 30, LR: 3.125e-05, Train Loss = 24.256, Test Loss = 24.390
Iteration 31, LR: 3.125e-05, Train Loss = 24.209, Test Loss = 24.339
Iteration 32, LR: 3.125e-05, Train Loss = 24.059, Test Loss = 24.193
Iteration 33, LR: 3.125e-05, Train Loss = 24.122, Test Loss = 24.247

```
                Iteration 34, LR: 3.125e-05, Train Loss = 23.989, Test Loss =
24.119
                Iteration 35, LR: 3.125e-05, Train Loss = 23.989, Test Loss =
24.117
                Iteration 36, LR: 3.125e-05, Train Loss = 23.926, Test Loss =
24.066
                Iteration 37, LR: 3.125e-05, Train Loss = 23.941, Test Loss =
24.077
                Iteration 38, LR: 3.125e-05, Train Loss = 23.771, Test Loss =
23.893
                Iteration 39, LR: 3.125e-05, Train Loss = 23.837, Test Loss =
23.979
                Iteration 40, LR: 1.5625e-05, Train Loss = 23.562, Test Loss =
23.699
        Done AutoEncoder Training
```

```
/home/apoorvsharma/anaconda3/envs/data598/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
AE Acc: 90.31, PCA Acc: 89.70
```

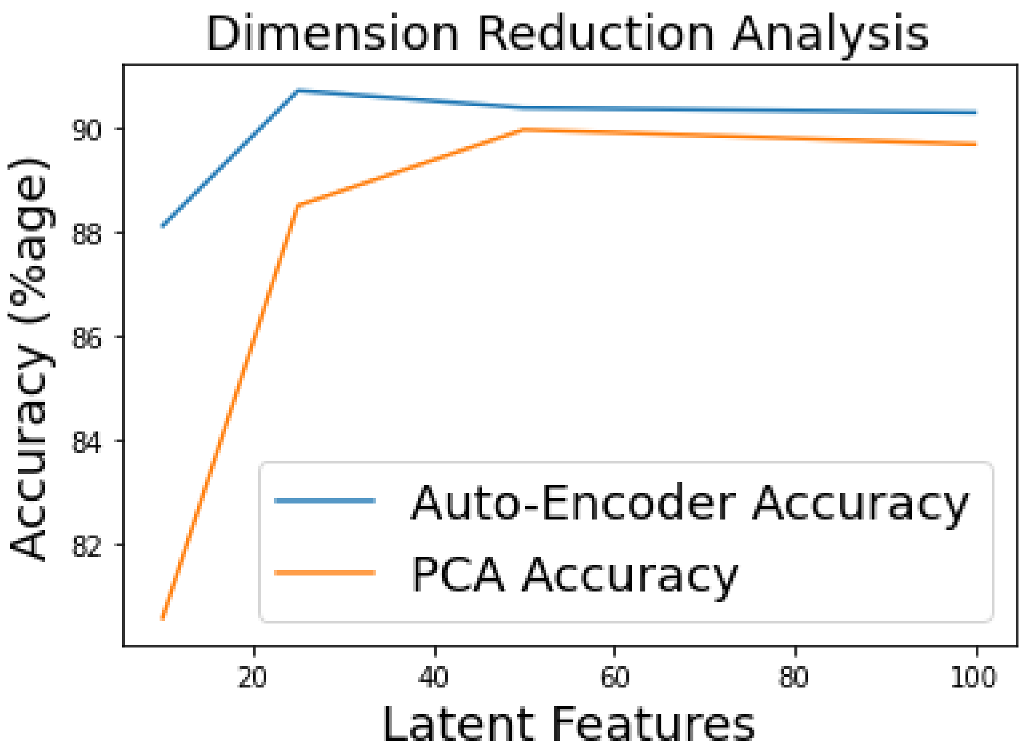
```
/home/apoorvsharma/anaconda3/envs/data598/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

3.0.6 Output Analysis

Make a plot with d on the x-axis and the best test accuracy of the logistic regression model on the y-axis with the d-dimensional representations. The plot should have two lines, corresponding to PCA and autoencoders.

[31]:
```
plt.plot(num_components_list, ae_accuracy, label="Auto-Encoder Accuracy")
plt.plot(num_components_list, pca_accuracy, label="PCA Accuracy")
plt.title('Dimension Reduction Analysis', fontsize=18)
plt.ylabel('Accuracy (%age)', fontsize=18)
plt.xlabel('Latent Features', fontsize=18)
plt.legend(fontsize=18)
```

[31]: <matplotlib.legend.Legend at 0x7f358027efa0>



Based on these observations, could you speculate why one of the two might be better or worse than the other?

PCA is essentially a linear transformation. However, AutoEncoders can map complex non-linear functions. As a result, for image data, auto-encoders perform better at reconstruction.