

A1: Conceptual Questions

A1.a

True. This is true because SVD is looking for a orthogonal matrix (PCA uses SVD) U , such that $U\Sigma V^T$ minimizes that reconstruction error. In this case the rank of matrix U can have the same rank as the subspace span by the columns of the data matrix X giving us zero reconstruction errors.

A1.b

True Because:

$$\begin{aligned} X^T X &= (USV^T)^T (USV) \\ &= VS^T U^T USV \\ &= VS^T SV \end{aligned} \tag{A1.b.1}$$

Take notice that $S^T S$ is diagonal and V is orthogonal, and $X^T X$ is Symmetric. By properties of Hermitian Adjoint Matrices, it has orthogonal Eigen Decomposition with unique real eigenvectors. And $VS^T SV$ matches it, therefore V is the eigen vectors of matrix $X^T X$.

A1.c

False. The objective should not be choosing k to minimize the Loss because if $k = n$ is always the global minimum in that case and it doesn't provide any useful interpretations on the data.

A1.d

False. Singular values Decomposition has U, V that are the eigenvectors for XX^T and $X^T X$, eigen values decomposition is not unique because you can multiply eigenvector by negative one (or even worse by the complex unit $\exp(i\theta)$) to get another normalized eigen vector that still works. In the case of SVD remember to flip the u, v vector corresponds to the same singular value together to get different decomposition for the same matrix.

A1.e

False when the matrix is degenerate. In this case a eigenvalue can have a geometric multiplicity higher than it's algebraic multiplicity, then the rank of the matrix will be more than the number of eigenvalues it has.

A1.f

True, because Autoencoders with non-linear activation function can incorporate non-linear representation of data in the lower dimension.

A2: Basics of SVD and Subgradients

A2.a

A2.a.(a)

I will just show you the math and explain some keys step on why this is true. This one easy to show because there is a closed form solution that we can incorporate the singular value decomposition for the covariance

matrix. let's consider the gradient of the objective function set to zero. s

$$\begin{aligned}
\nabla [\|Xw - y\|_2^2 + \lambda\|w\|_2^2] &= \mathbf{0} \\
2X^T(Xw - y) + 2\lambda w &= \mathbf{0} \\
X^T(Xw - y) + \lambda w &= \mathbf{0} \\
X^T Xw - X^T y + \lambda w &= \mathbf{0} \\
(X^T X + \lambda I)w &= X^T y
\end{aligned} \tag{A2.a.a.1}$$

Using the Singular value decomposition we have:

$$\begin{aligned}
X^T X &= (U\Sigma V^T)^T (U\Sigma V^T) \\
X^T X &= (V\Sigma U^T)(U\Sigma V^T) \\
X^T X &= V\Sigma^2 V^T
\end{aligned} \tag{A2.a.a.2}$$

We make use of the fact that, U, V are unitary matrices and the singular matrix Σ is a diagonal, containing all the singular values ranked in order of magnitude, and padded with zeros. Here we consider the case of Economic Singular Value Decomposition. Substituting the previous expression to the previous previous expression we have:

$$\begin{aligned}
\hat{w}_R &= (X^T X + \lambda I)^{-1} X^T y \\
\hat{w}_R &= ((V\Sigma^2 V) + V(\lambda I)V^T)^{-1} X^T y \\
\hat{w}_R &= (V(\Sigma^2 + \lambda I)V^T)^{-1} X^T y \\
\hat{w}_R &= V^T(\Sigma^2 + \lambda I)^{-1} V X^T y \\
\|\hat{w}_R\|_2 &= \|V^T\|_2 \|(\Sigma^2 + \lambda I)^{-1}\|_2 \|V\|_2 \|X^T\|_2 \|y\|_2 \\
\|\hat{w}_R\|_2 &= \left(\sum_{i=1}^{\min(m,n)} \frac{1}{\sigma_i^2 + \lambda} \right) \|X^T\|_2 \|y\|_2
\end{aligned} \tag{A2.a.a.3}$$

Taking the limit as $\lambda \rightarrow \infty$ will yield zero for the norm of \hat{w}_R , and in this case, here are the facts we used

1. we used the fact that the induced 2 norm of a unitary matrix is one, which will be proven in the next part.
2. And the induced 2 norm for a diagonal matrix is just the sum of all it's diagonal elements.
3. The inverse of a unitary matrix is it's Transpose, assuming it's real, and in this case, U, V are unitary matrices.
4. $(AB)^{-1} = B^{-1}A^{-1}$ assuming invertible A, B
5. $(AB)^T = B^T A^T$

Note: The bound on the summation is implicitly making the assumption that X is a $m \times n$ or $n \times m$ matrix.

A2.a.(b)

A2.a.b) From the previous part, [A2.a\(a\)](#), I have shown that $X^T X = V\Sigma^2 V$ where $X = U\Sigma V$. And in this question, we just had U instead of X , let's use X , cause U is already involved in the SVD. Let $\Sigma = I_n$ which sets the singular values of X to be all ones, then $X^T X = V I_n V^T = I_n$ Because V is a unitary matrix and its

inverse it's its transpose, similarly for XX^T

$$\begin{aligned}
XX^T &= (U\Sigma V^T)(U\Sigma V^T)^T \\
XX^T &= (U\Sigma V^T)(V\Sigma U^T) \\
XX^T &= (U\Sigma\Sigma U^T) \\
XX^T &= U \underbrace{\Sigma^2}_{I_n} U^T \\
XX^T &= UU^T = I_n
\end{aligned} \tag{A2.a.b.1}$$

Now, using the definition of the Norm we have:

$$\begin{aligned}
\|Ux\|_2^2 &= (Ux)^T(Ux) \\
x^T U U x &= x^T x \\
&= \|x\|_2^2
\end{aligned} \tag{A2.a.b.2}$$

A2.b

A2.0: Preliminaries

Let's denote the set of $\{g : f(y) \geq f(x) + g^T(y-x)\}$ to be $\partial[f]$, which is really a compact set in the Euclidean space. In the case of vector is going to be a cone.

From the problem statement we gather: $\exists i \in [m] : f(x) = f_i(x) \implies \partial[f_i] \subseteq \partial[f]$. Now, if we make the deliberate choice on i for a given particular x , we can make the claim that:

$$\partial[f] = (\inf\{\partial[f_i(x)] : f_i(x) = f(x)\}, \sup\{\partial[f_i(x)] : f_i(x) = f(x)\}) \tag{A2.0.1}$$

Basically, we can choose the i such to find a lower and upper bound for the sub gradient of f if $f_i(x) = f(x)$.

And when we consider the sum of a lot of convex functions $\sum_{i=1}^n f_i(x)$, then:

$$\begin{aligned}
f_i(y) &\geq f_i(x) + v^T(x-y) \quad \forall i \in [m], v \in \partial[f_i](x) \\
\sum_{i=1}^m f_i(x) &\geq v^T(x-y) \quad \forall v \in \left(\sum_{i=1}^m \inf\{\partial[f_i]\}, \sum_{i=1}^m \sup\{\partial[f_i]\} \right)
\end{aligned} \tag{A2.0.2}$$

And this is how summation for sub gradient works if we want to sum up several functions, we just need to sup up the supremum and infimum to get the range for the new subgradient, which is still going to be a compact set, or a cone.

A2.b.(a)

$$\begin{aligned}
&\partial \left[\sum_{i=1}^n |x_i| \right] \\
&= \sum_{i=1}^n \partial[|x_i|] \\
&= \sum_{i=1}^n g_i \mathbf{e}_i
\end{aligned} \tag{a2.b.a.1}$$

g_i is essentially:

$$g_i \in \partial[|x_i|] = \begin{cases} \{1\} & x_i \geq 1 \\ [-1, 1] & x_i = 0 \\ \{-1\} & x_i \leq 0 \end{cases} \tag{a2.b.a.2}$$

And using the hint from the next part, the sub gradient of $\|x\|_1$ is the convex combinations of all $g_i \mathbf{e}_i$:

$$\sum_{i=1}^n \lambda_i g_i \mathbf{e}_i \in \partial[\|x\|_1] \quad \sum_{i=1}^n \lambda_i \leq 1 \wedge \lambda_i \geq 0 \quad (\text{a2.b.a.3})$$

And the span of all sub gradient for each $|x_i|$ will make up the set of sub-gradient for the original function, and hence, let v_j be the j th element of the sub gradient of $\|x\|_1$, the closed form will be:

$$v_j \in \begin{cases} \{1\} & x_j > 0 \\ [-1, 1] & x_j = 0 \\ \{-1\} & x_j < 0 \end{cases} \quad (\text{A2.b.1.3})$$

A2.b.(b)

Let λ_i be the set of coefficients for a convex combinations, meaning that $\sum_{i=1}^m \lambda_i = 1$ and $\lambda_i \geq 0$, implying that $\lambda_i \in (0, 1)$. Using this fact and the definition of $f(x) := \max\{f_i(x)\}_i^m$, consider the following:

$$\begin{aligned} f(y) &\geq f_i(y) \quad \forall i & (\text{A2.b.b.1}) \\ \lambda_i f(y) &\geq \lambda_i f_i(y) \quad \forall i \\ \sum_{i=1}^m \lambda_i f(y) &\geq \sum_{i=1}^m \lambda_i f_i(y) \\ \stackrel{(1)}{\implies} f(y) &\geq \sum_{i=1}^m \lambda_i f_i(y) \\ f(y) &\geq \underbrace{\left(\sum_{i=1}^m \lambda_i f_i(x) \right)}_{\leq f(x)} + \lambda_i \nabla[f_i](x)^T (y - x) \\ \stackrel{(2)}{\implies} f(y) &\geq f(x) + \lambda_i \nabla[f_i](x)^T (y - x) \quad \forall i \end{aligned}$$

(1) : True because the convex combinations coefficients $\sum_{i=1}^m \lambda_i = 1$ and $f(y)$ is independent of the summation.

(2) : True because the $\sum_{i=1}^m \lambda_i f_i(x) \leq f(x)$ is already proven in (1).

Now, we are free to choose λ_i to find the bound of the all the convex combinations of the sub gradient on f_i at x . Therefore, the sub-gradient is the set defined as the following:

$$(\partial[f](x))_j = (\inf \{(\nabla[f_i](x))_j : f_i(x) = f(x)\}, \sup \{(\nabla[f_i](x))_j : f_i(x) = f(x)\}) \quad (\text{A2.b.b.2})$$

Note: The notation of $(\bullet)_j$ is denoting the j th element of a vector, in this case, we are saying that the j th element of the sub gradient vector for f is bounded by the sup and inf of the j th element of the gradient of the smooth function f_i .

A2.c

In this case $f_i(x) = |x_i - (1 + \eta/i)|$ hence we can say v_i is a subgradient of f_i if:

$$\begin{aligned} v_i \in \partial[|x_i - (1 + \eta/i)|] &= \begin{cases} \{1\} & x_i > 1 + \frac{\eta}{i} \\ [-1, 1] & x_i = 1 + \frac{\eta}{i} \\ \{-1\} & x_i < 1 + \frac{\eta}{i} \end{cases} & (\text{A2.c.1}) \\ \implies \forall x \in \text{dom}(f), i \in [n] : & -1 \leq v_i \leq 1 \\ & \implies \|v_i \mathbf{e}_i\|_\infty \leq 1 \end{aligned}$$

Therefore, we know that the convex combinations will be bounded too and it's like:

$$\forall \lambda_i \geq 0 \wedge \sum_{i=1}^n \lambda_i \leq 1 : \left\| \underbrace{\sum_{i=1}^n \lambda_i v_i \mathbf{e}_i}_{\in \partial[f]} \right\| \in [-1, 1] \quad (\text{A2.c.2})$$

Therefore, the infinity norm of the sub gradient of the function f is in the set interval $[-1, 1]$.

A3: PCA

A3.a

A3.b

A3.c

A3.d

A3.e

A4: Unsupervised Learning with Autoencoders

A4.a

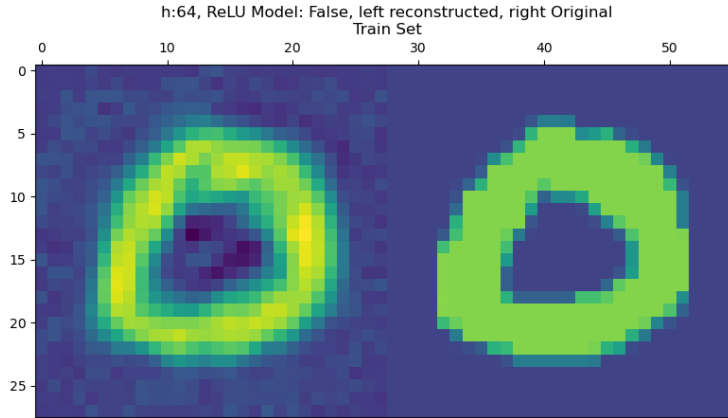
The Train Error for $h \in \{32, 64, 128\}$ for the linear models are:

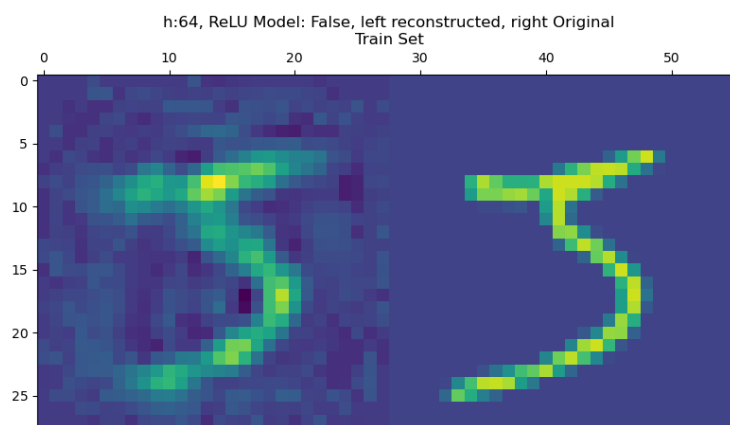
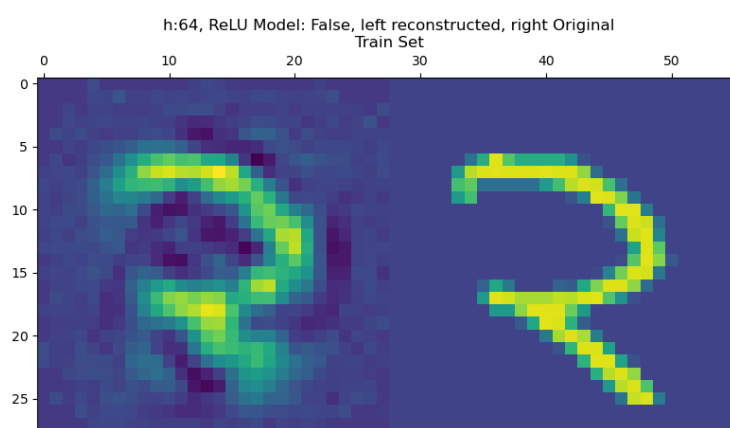
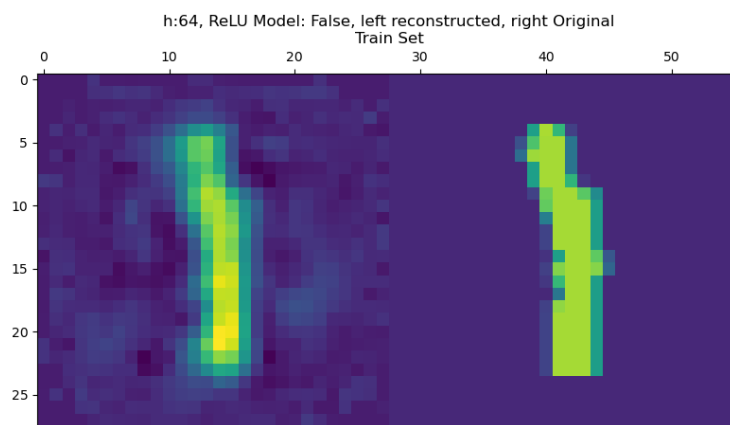
1. $h = 32$, Total Epochs: 30 Train Final MSE Loss: 0.07298633098602295
2. $h = 64$, Total Epochs: 30 Train Final MSE Loss: 0.07428810136703154
3. $h = 128$, Total Epochs: 30 Train Final MSE Loss: 0.058713519498705916

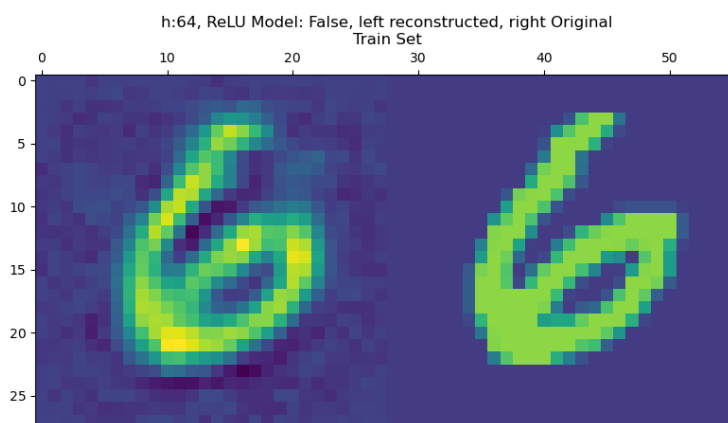
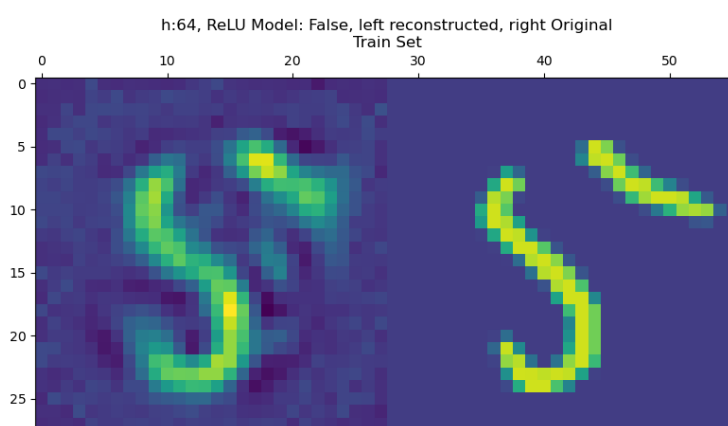
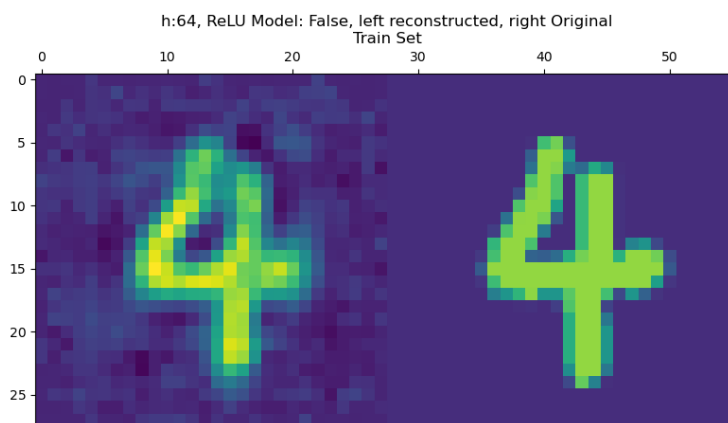
The MSE loss is computed via $\frac{1}{N} \sum_{i=1}^N \|f(g(x_i)) - x_i\|_2^2$, Where f, g are the encoder and decoder. The losses are divided by the total number of batches from the data loader, hence the error in the end is the squared loss on a persample basis. For code implementation, refers [A4.Code](#).

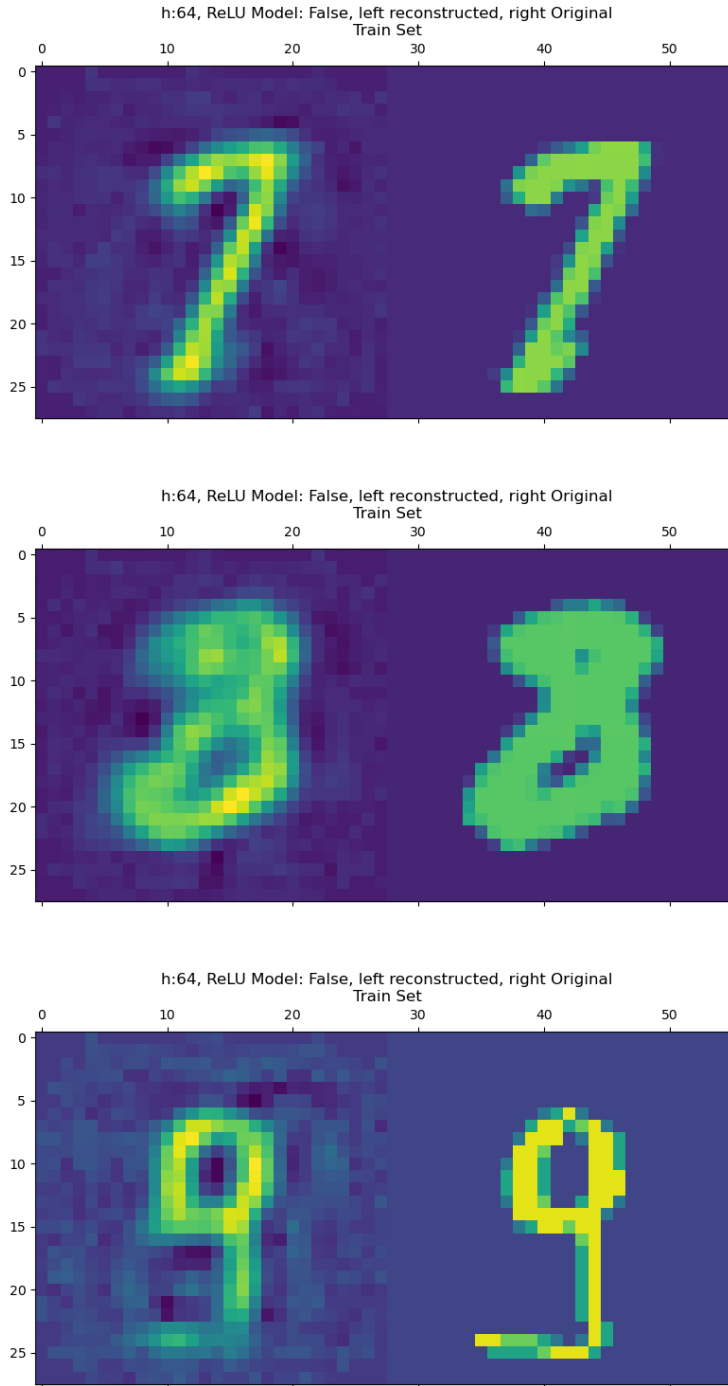
The Autoencoder is trained on the whole training data set of MNIST.

This is 10 digits reconstruction for $h = 64$ are below, and for all of the reconstruction plots for pairs of digits with $h = 32, 128$, please refer to [Appendix](#).







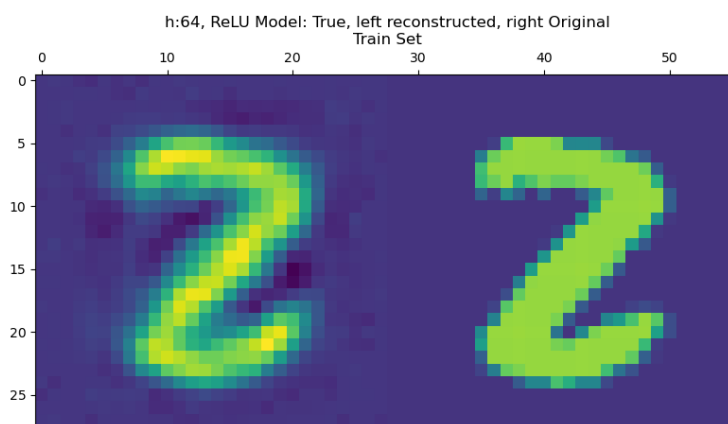
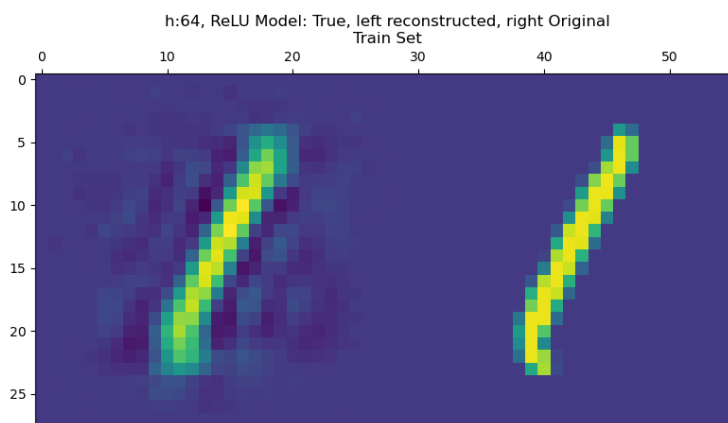
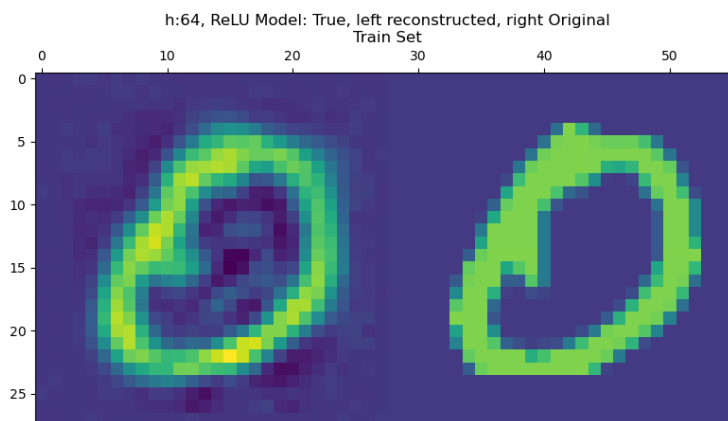


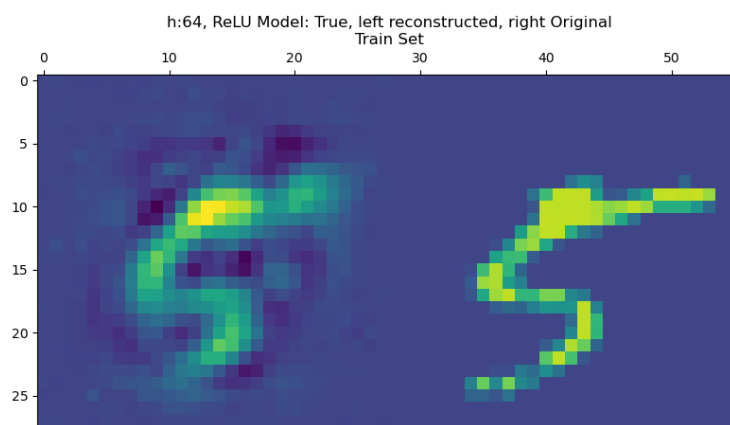
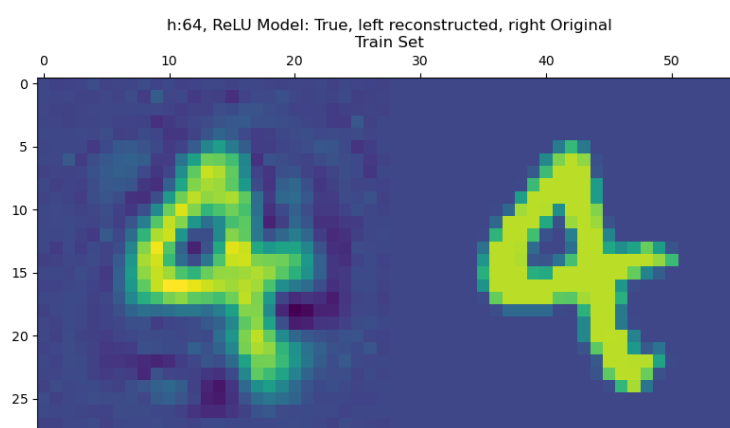
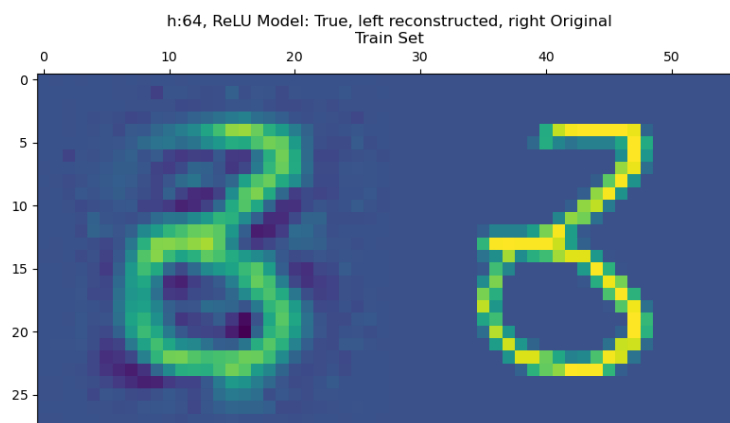
A4.b

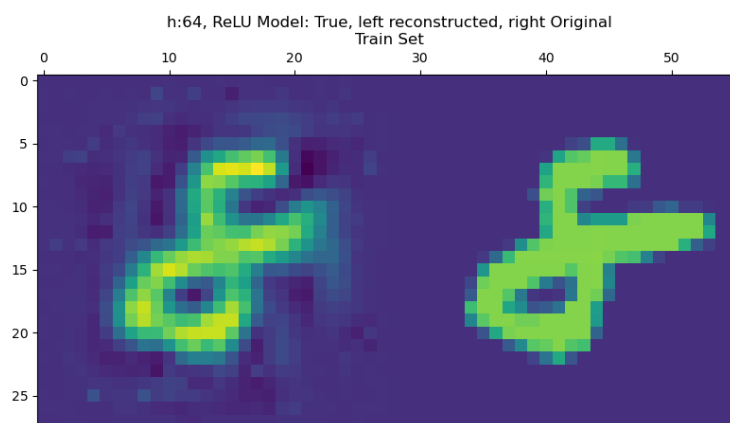
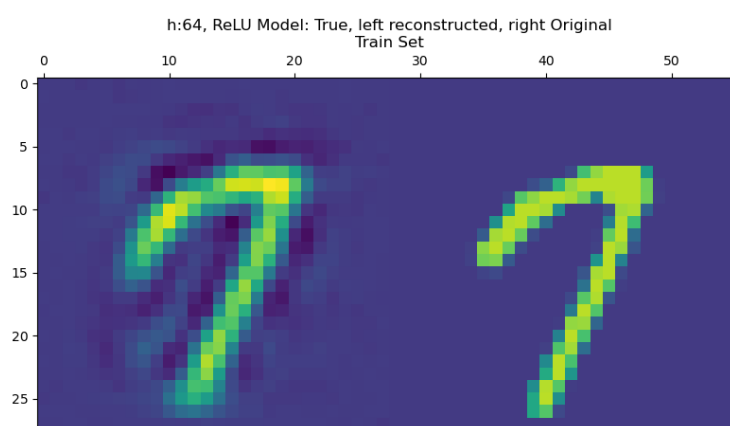
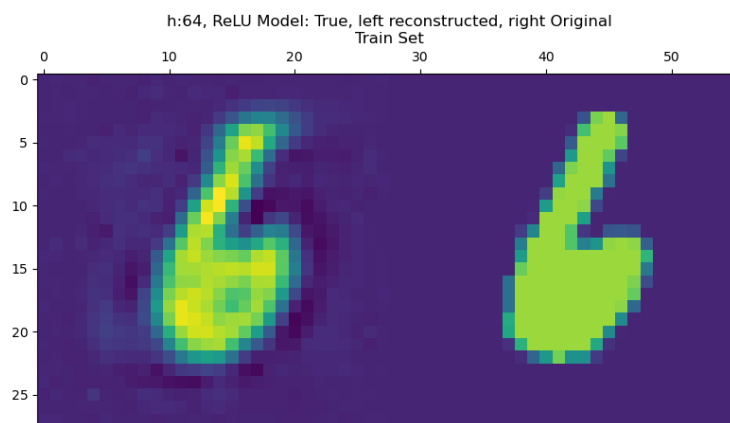
The train error for $h \in \{32, 64, 128\}$ for the non-linear model with ReLU activation is:

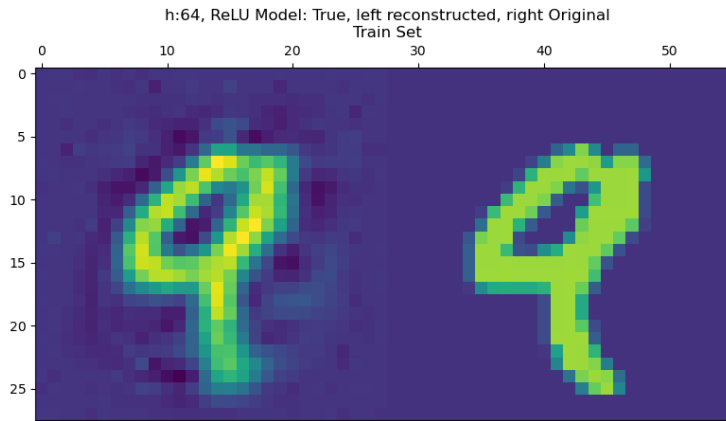
1. $h = 32$, total Epochs: 30 Train Final MSE Loss: 1.2326371114328512
2. $h = 64$, Total Epochs: 30 Train Final MSE Loss: 0.0249483520537615
3. $h = 128$, Total Epochs: 30 Test Final MSE Loss: 0.024230041910583734

The MSE is computed the same as part A4.b. And these are some of the reconstruction images for $h = 64$:









A4.c

A4.d

A4.Code

This is the code I used for the assignment:

Filename: "mnist.autoencoders"

```
### CLASS: CSE 546 SPRING 2021 HW4, A3
### Name: Hongda Li
### My code has my style in it don't copy.
```

```
import numpy as np
import torch
import torchvision
import matplotlib.pyplot as plt
from time import time
from torchvision import datasets, transforms
from torch import nn, optim
from tqdm import tqdm
import random as sysrandom

TRANSFORM = transforms.Compose([transforms.ToTensor(),
                                transforms.Normalize((0.5,),(0.5,)),
                                ])

TRAIN_SET = \
    datasets.MNIST('./data', download=True, train=True, transform=TRANSFORM)
TEST_SET = \
    datasets.MNIST('./data', download=False, train=True, transform=TRANSFORM)

TRAIN_SET, TEST_SET = \
    torch.utils.data.Subset(TRAIN_SET, range(0, 1000)), \
    torch.utils.data.Subset(TEST_SET, range(0, 1000))

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

class A4Model(nn.Module):

    def __init__(this, h:int, non_linear:bool=False):
        """
        :param kargs:
            d: The dimension of the data.
            h: The width of the hidden layer.
            activation: TRUE, FALSE
```

```

        Whether to use ReLU activation function on hidden and output layer.
    """
    super().__init__()
    d = 28*28
    this.L1 = nn.Linear(d, h)
    this.L2 = nn.Linear(h, d)
    this.NonLin = non_linear
    this.MSE = nn.MSELoss()

def forward(this, X):
    """
        Feed Forward without Loss function capped onto the output layer.
    :param X:
        The data matrix, row data matrix.
    :return:

    """
    x = this.L1(X)
    if this.NonLin:
        x = nn.ReLU(x)
    x = this.L2(x)
    if this.NonLin:
        x = nn.ReLU(x)
    return x

def GetEmbedding(this, X):

    pass

def FeedForward(this, X):
    return this.MSE(this(X), X)

def MNISTTenUniqueDigitsLoader(train=True):
    data = \
        datasets.MNIST('./data', download=True, train=train, transform=TRANSFORM)
    Indices = []
    for II in range(10):
        Idx = torch.where(data.targets == II)[0]
        Indices.append(Idx[sysrandom.randint(0, len(Idx))])
    Subset = torch.utils.data.Subset(data, Indices)
    return torch.utils.data.DataLoader(Subset, batch_size=10, shuffle=False)

def BatchThisModel(theModel:A4Model,
                    dataLoader:torch.utils.data.DataLoader,
                    optimizer:optim.Adam=None,
                    transform:callable=None):
    """
        Batch this model for one epoch, give me the model optimizer and some extra thing, then it will collect the average loss of one epoch.
    Note:
        This one is for Regression Model, it assumes MSE loss, loss of each batch if divided by the total number of batches from the data loader.
    :param theModel:
    :param dataLoader:
    :param transform:
    :return:
    """
    AvgLoss = 0; theModel.to(DEVICE)
    L = len(dataLoader)
    for II, (X, _) in enumerate(tqdm(dataLoader)):
        if transform is not None: X = transform(X)
        X = X.to(DEVICE)
        if optimizer is None:
            with torch.no_grad():
                AvgLoss += float(theModel.FeedForward(X))/L

```

```

        else:
            optimizer.zero_grad()
            Loss = theModel.FeedForward(X)
            AvgLoss += Loss.item() / L
            Loss.backward()
            optimizer.step()
    with torch.no_grad():
        Loss = theModel.FeedForward(X)
        AvgLoss += Loss.item() / L
    return AvgLoss

def GetTrainTestLoaders(bs=100):
    TrainLoader = \
        torch.utils.data.DataLoader(TRAIN_SET, batch_size=bs, shuffle=True)
    TestLoader = \
        torch.utils.data.DataLoader(TEST_SET, batch_size=bs, shuffle=True)
    return TrainLoader, TestLoader

def Ts():
    from time import gmtime, strftime
    TimeStamp = strftime("%H-%M-%S", gmtime())
    return TimeStamp

def mkdir(dir):
    from pathlib import Path
    Path(dir).mkdir(parents=True, exist_ok=True)

def log(fname:str, content:str, dir):
    mkdir(dir)
    TimeStamp = Ts()
    with open(f"{dir}{TimeStamp}-{fname}.txt", "w+") as f:
        f.write(content)

def main():
    def A4Run(NonLin, h):
        Bs, Epochs, lr = 100, 30, 0.01
        Losses = []
        Model = A4Model(h=h, non_linear=False)
        Optimizer = optim.Adam(Model.parameters(), lr=lr)
        TrainLoader, TestsLoader = GetTrainTestLoaders(bs=Bs)
        for Epoch in range(Epochs):
            Loss = BatchThisModel(theModel=Model,
                                   dataLoader=TrainLoader,
                                   optimizer=Optimizer,
                                   transform=lambda x: x.reshape(Bs, -1))
            Losses.append(Loss)
        log(fname=f"A4-final-train-Loss-h={h}-{ 'Non' if NonLin else '' }lin-model",
            content=f"Total_Epochs:{Epochs}_Train_Final_MSE_Loss:{Losses[-1]}",
            dir="./A4logs/")
    def Visualize(Train=True):
        mkdir("./A4plots")
        DigitsLoader = MNISTTenUniqueDigitsLoader(Train)
        for X, _ in DigitsLoader:
            X = X.reshape(X.shape[0], -1)
            Reconstructed = Model(X.to(DEVICE))
        for II, (Row1, Row2) in enumerate(zip(Reconstructed.data, X.data)):
            Together = torch.zeros(28, 28*2)
            Together[:, :28] = Row1.reshape(28, 28)
            Together[:, 28:] = Row2.reshape(28, 28)
            plt.matshow(Together)
            plt.title(f"h:{h}, _ReLU_Model:{ 'NonLin' if NonLin else '' }, _left_reconstructed, _right_Original\n"
                    f"{ 'Train_Set' if Train else 'Test_Set' }")

```

```

plt.savefig(f"./A4plots/{Ts()}-h-{h}-{ 'non' if NonLin else '' }lin-digit-{II}.png"
)
plt.show()

Visualize()
if h == 128:
    Loss = BatchThisModel(theModel=Model,
                           dataLoader=TestsLoader,
                           transform=lambda x: x.reshape(Bs, -1))
    log(fname=f"A4-test-loss-h={h}-{ 'Non' if NonLin else '' }lin-model",
        content=f"Total_Epochs:_{Epochs}_Test_Final_MSE_Loss:_{Loss}",
        dir="./A4logs/")

    return Model, Losses

A4Run(False, h=32)
A4Run(False, h=64)
A4Run(False, h=128)
A4Run(True, h=32)
A4Run(True, h=64)
A4Run(True, h=128)

if __name__ == "__main__":
    import os
    print(os.getcwd())
    print(os.getcwd())
    main()

```

A5: K-Mean Clustering

A5.a

This is the code I used for the whole A5, it contains my implementation of the Lloyd's Kmeans algorithm using numpy.

filename: "k_mean.py"

```

### CLASS CSE 564 SPRING 2021 HW4 A4
### Name: Hongda Li
### My code has my style in it don't copy.

import numpy as np
import matplotlib.pyplot as plt
from torchvision import datasets
from mnist import MNIST
from tqdm import tqdm
zeros = np.zeros
randint = np.random.randint
randn = np.random.randn

if "MNIST_DATA" not in dir(): # running on interactive console will be faster
    datasets.MNIST('./data', download=True, train=True)
    MNIST_DATA = MNIST("./data/MNIST/raw/")
    TRAIN_X, _ = MNIST_DATA.load_training()
    TEST_X, _ = MNIST_DATA.load_testing()
    TRAIN_X = np.array(TRAIN_X, dtype=np.float)/255
    TEST_X = np.array(TEST_X, dtype=np.float)/255
    print("Mnist_Dataset_is_ready.")

```

```

# ===== Helper Functions =====

def Ts():
    from datetime import datetime
    SysTime = datetime.now()
    TimeStamp = SysTime.strftime("%H-%M-%S")
    return TimeStamp

def mkdir(dir):
    from pathlib import Path
    Path(dir).mkdir(parents=True, exist_ok=True)

def log(fname:str, content:str, dir):
    mkdir(dir)
    TimeStamp = Ts()
    with open(f"{dir}{TimeStamp}-{fname}.txt", "w+") as f:
        f.write(content)

# =====

class KMean:

    def __init__(this, k:int, X:np.ndarray):
        """
        :param k: Number of cluster
        :param X: Row data matrix in np array type
        """
        assert k < X.shape[0] and k > 1
        assert X.ndim == 2
        n, d = X.shape[0], X.shape[1]
        this._X = X
        # this._AugX = X[:, :, np.newaxis]
        this.Assignment = {}
        this._C = np.transpose(this._X[randint(0, n, k), :][..., np.newaxis],
                               (2, 1, 0))
        this._ComputeAssignment()

    @property
    def Centroids(this):
        return np.transpose(this._C, (2, 1, 0))[..., 0].copy()
    @property
    def X(this):
        return this._X.copy()
    @property
    def AugX(this):
        return this._AugX.copy()
    @property
    def C(this):
        return this._C.copy()

    def TransferLearningFrom(this, other):
        this._C = other.C
        this._ComputeAssignment()

    def _ComputeCentroid(this):
        """
        Compute centroid using the current assignment.

        :return:
        """
        for Centroid, Idx in this.Assignment.items():
            this._C[..., Centroid] = \
                np.mean(this._X[Idx], axis=0, keepdims=True)

```



```

def _ComputeAssignment(this, XTest=None):
    """
    Given current centroids make an assignment.
    :return:
    """

    X = this._X if XTest is None else XTest
    Distances = zeros((X.shape[0], 1, this._C.shape[2]))
    for CIdx in range(this._C.shape[2]):
        Centroid = this._C[:, :, CIdx]
        Distances[:, :, CIdx] = np.sum((X - Centroid)**2,
                                         axis=1,
                                         keepdims=True)

    AssignmentVec = np.argmin(Distances, axis=2).reshape(-1)
    NewAssignment = {}
    for Idx, Class in enumerate(AssignmentVec):
        IdxArr = NewAssignment.get(Class, [])
        IdxArr.append(Idx)
        NewAssignment[Class] = IdxArr
    if XTest is None:
        this.Assignment = NewAssignment
    del Distances
    return NewAssignment.copy()

def Update(this):
    this._ComputeCentroid()
    this._ComputeAssignment()

def Loss(this, Xtest=None):
    X = this._X if Xtest is None else Xtest
    TestAssignment = this._ComputeAssignment(Xtest)
    Centroids = this.Centroids
    Loss = 0
    for CentroidIdx, Idx in TestAssignment.items():
        Loss += np.sum((X[Idx] - Centroids[CentroidIdx, :])**2)
    return Loss/X.shape[0]

def main():

    def BasicTest():
        Points1 = randn(1000, 2)
        Points2 = np.array([[3, 3]]) + randn(1000, 2)
        PointsAll = np.concatenate((Points1, Points2), axis=0)
        Km = KMean(X=PointsAll, k=2)
        Losses = []
        for II in range(10):
            Km.Update()
            Losses.append(Km.Loss())
        plt.plot(Losses)
        plt.show()
        return Km

    def Learn(Km:KMean, n=None):
        Losses = []
        if n is not None:
            for _ in tqdm(range(n)):
                Km.Update()
                Losses.append(Km.Loss())
        else:
            C = Km.Centroids
            while True:
                Km.Update()
                Losses.append(Km.Loss())
                Delta = np.linalg.norm(C - Km.Centroids, np.inf)
                print(f"Delta:_{Delta}")
                if Delta < 1e-1:
                    break

```

```

        C = Km.Centroids
    return Km, Losses

def ClusterMnist(k=10,X=None):
    if X is None: X = TRAIN_X
    Km, Losses = Learn(KMean(X=X,k=k))
    return Km, Losses

def A5b():
    Km, Losses = Learn(KMean(X=TRAIN_X,k=10))
    plt.plot(Losses)
    plt.title("A5(b)_Kmean_k=10")
    plt.xlabel("Iteration")
    plt.ylabel("Average_Loss")
    mkdir("./A5bplots")
    plt.savefig(f"./A5bplots/{Ts()}-A5b-k=10-losses.png")
    plt.show()
    AllCentroid = zeros((28*2, 28*5))
    for Idx, Centroid in enumerate(Km.Centroids):
        Image = Centroid.reshape((28, 28))
        VerticalOffset, HorizontalOffset = (Idx//5)*28, (Idx%5)*28
        AllCentroid[VerticalOffset:VerticalOffset+28,
                    HorizontalOffset:HorizontalOffset+28] = Image
    plt.matshow(AllCentroid)
    plt.title("A5(b):Centroids_fond_by_Kmean")
    plt.savefig(f"./A5bplots/{Ts()}-A5b-k=10-centroids.png")
    plt.show()

def A5c():
    NumberOfCluster = list(map(lambda x: 2**x, range(1, 7)))
    TrainLosses, TestLosses = [], []
    for K in NumberOfCluster:
        Km, Losses = ClusterMnist(k=K, X=TRAIN_X[:5000])
        TrainLosses.append(Losses[-1])
        TestLosses.append(Km.Loss(TEST_X))
    plt.plot(NumberOfCluster, TrainLosses, "-")
    plt.plot(NumberOfCluster, TestLosses, "-")
    plt.legend(["Losses_on_Train_Set", "Losses_on_Test_Set"])
    plt.title("K-Mean_on_MNIST,_Cluster_Number_vs_Loss")
    plt.xlabel("Number_of_Cluster")
    plt.ylabel("Loss")
    plt.savefig(f"./A5bplots/{Ts()}-A5b-k-vs-loss.png")
    plt.show()

# A5b()
A5c()

if __name__ == "__main__":
    import os
    print(f"{os.getcwd()}")
    print(f"{os.curdir}")
    main()

```

A5.b

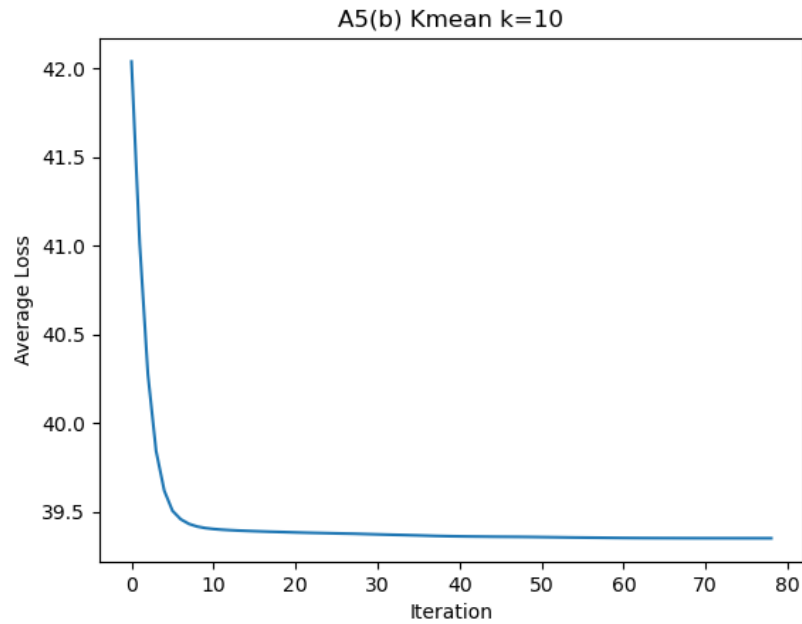
This is the error for the k mean algorithm with $k = 10$. The algorithm iterates until the maximal centroid's position doesn't change by more than $1e - 2$.

Note, here are some implementation details that might effect the average loss for each samples computed:

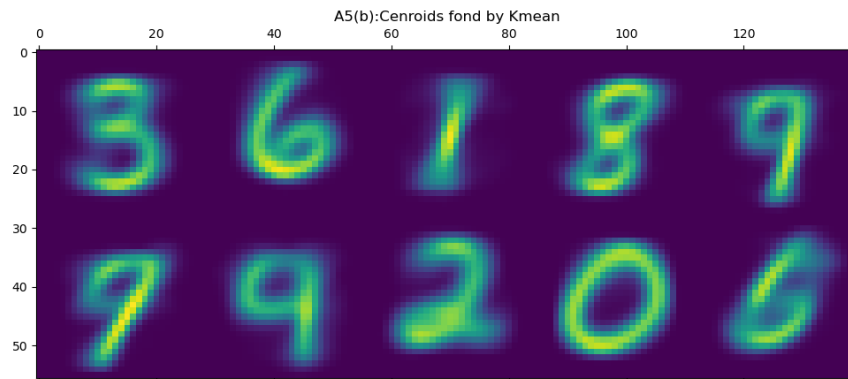
- : I normalized the MNIST data by dividing it by 255 so all the pixels values are in $[0, 1]$.

- : It's trained on the whole MNIST train dataset.

Code: [A5.a](#)



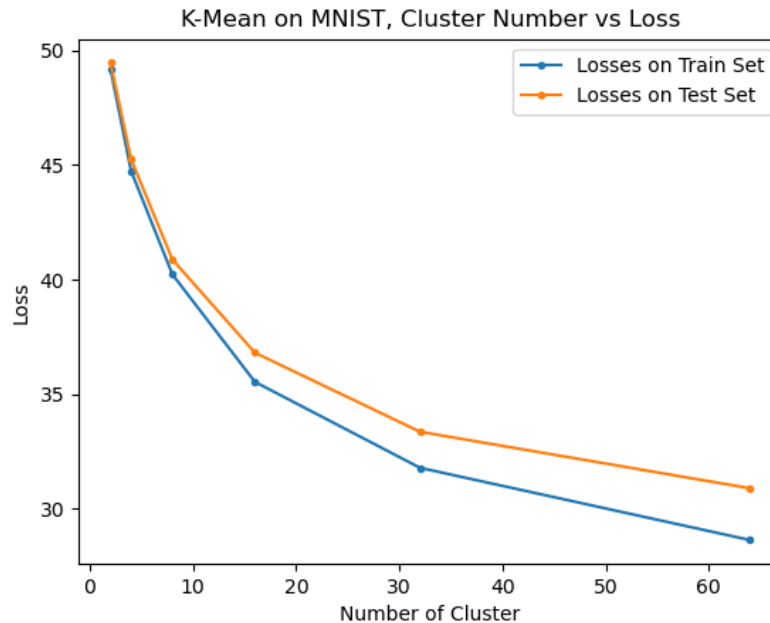
And these are 10 of the centroids identified by k-means, visualized as 28×28 matrices:



A5.c

For this part, I trained the model with values of $k \in \{2, 4, 8, 16, 32, 64\}$ on the whole normalized training MNIST dataset, and the value of the loss function after the algorithm converged are plotted against each value of k , and this is the graph:

Code [A5.a](#)



A6: ML in the Real World

A6.a: Disease Susceptibility Predictor

Assuming all the data are collected correctly and filled in and “None”, “Nan” are not in the dataset.

We are going to use the lasso regression to look for the best predictors for the system. More specifically we are look for the shrinkage of difference factors to determine the most relavent factors that cause the disease.

During this stage of development, ask experts about the identified factors, and then improve models like, group lasso, or lasso ridge method etc to get more alternatives the most important group of identifiers.

After the most relavent factors are identified, we will only use those predictor and then Logistic Regression, or Neural Networks, or whatever binary classifier models that get's the best test accuracy.

A6.b: Social Media App Facial Recognition Technology

I will look for a pretrained YOLO V3 network, this model can frame the item in the image with a label. And it's fast. A homebrew Neural CNN can do the job, but it's really slow, and it might not work as well since it's just a classifier and can't really identify the objects with its position in the photo (The bounding box with labels). If we were going to make our own neural network, then we will have to make the output of the classifier corresponds to different region of the image, and each region should output a label. And the loss function will have to be tailored for this usage, and we will be consider using the Residual Network for dimensionality reduction in this case.

Since that data is only comming from employees and their families, we might need to augment the dataset with a monochrome filters, this is an attempt to avoid biases created by the limited dataset.

Let's hope on of the ways can work out.

A6.c: Malware Detection

For binary executable file, I will consider the usage of deep Recurrent Neural Network. However, it's possible that we have executable source code (because the question says: **"Including its contents"**), such as javascript or python script, so it's preferable to train a model for the file content for each of these above cases.

For binary file, we would need to decode it to assembly, and then train the data on Deep Recurrent Neural net, with classification loss function. The model is non-parametric so it's easy to scale. It has the challenge of vanishing and exploding gradient.

If, the above RNN doesn't work well, we are considering using BERT model as a classifier as the alternative, training on the Assembly code of course.

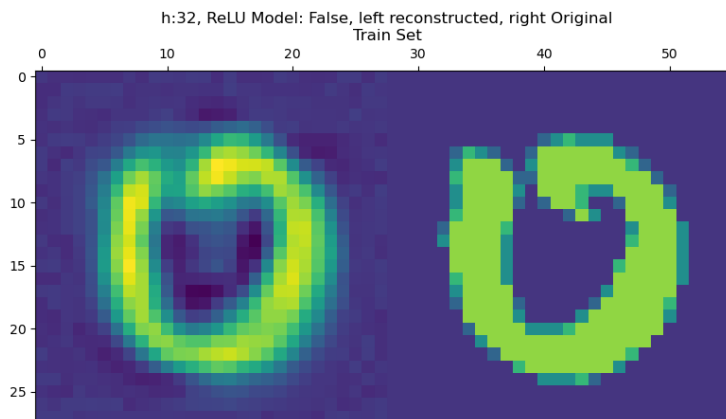
For source code, we will need to parse it into graph that keep track of all the variable flows (Decide on all the branches the code and take and how it changes the dictionary of variables.), which should get pass obfuscated code. And then, we extract out all the API calls for the JS scripting languages, like system call, file IOs, network connection, etc. Then, we represent those elements as canonical vectors, they are like the keywords we are interested in. Then, one vector represents a computer program. And then we are using these vector to train a Neural Net classifier.

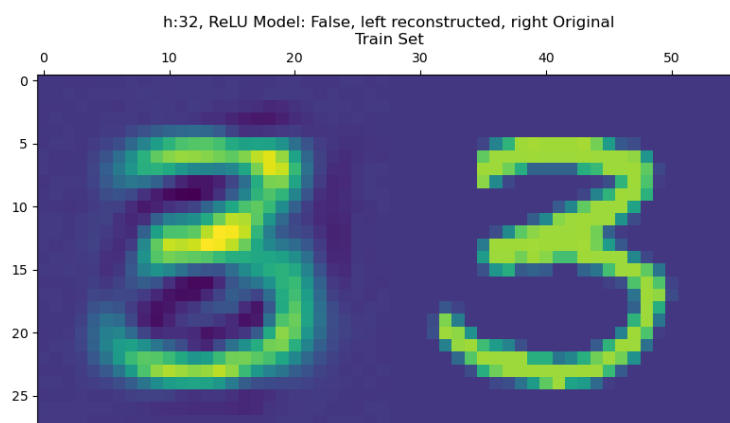
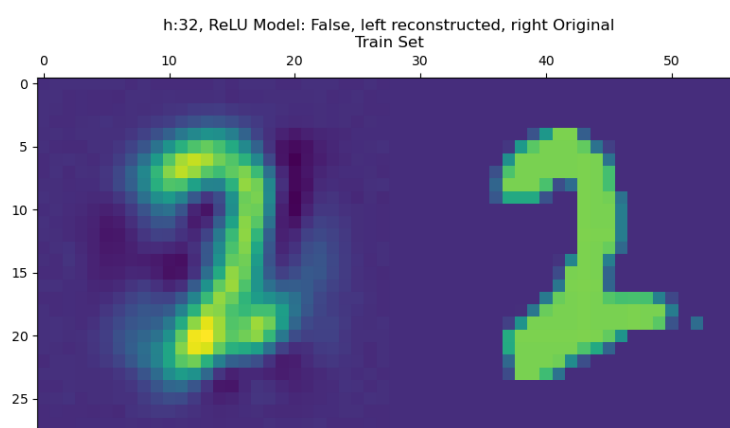
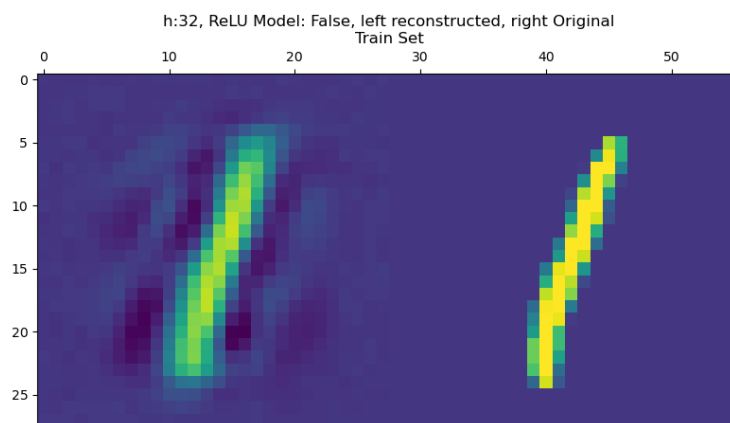
For meta data, we just include it together as an attribute for the computer program. If it's the binary content, then we encode that as part of the Assembly sequence. If it's source code, then we add those in as a vector as well. Both are non-parametric and accurate predictor and it should scale well.

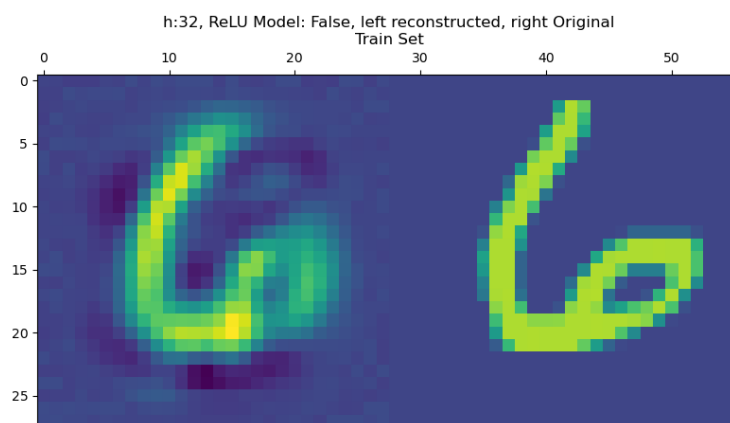
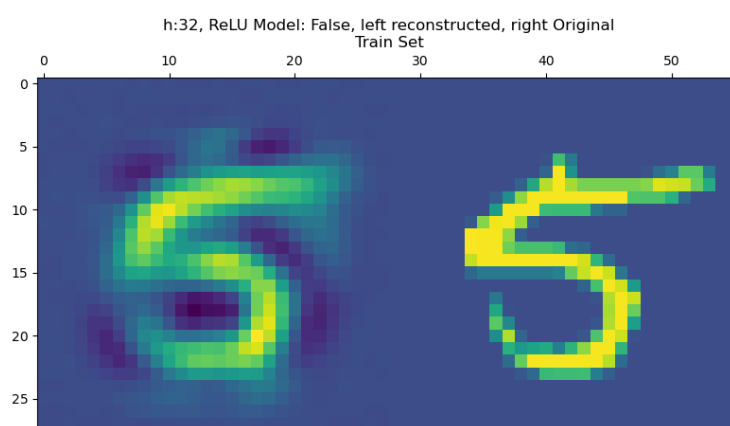
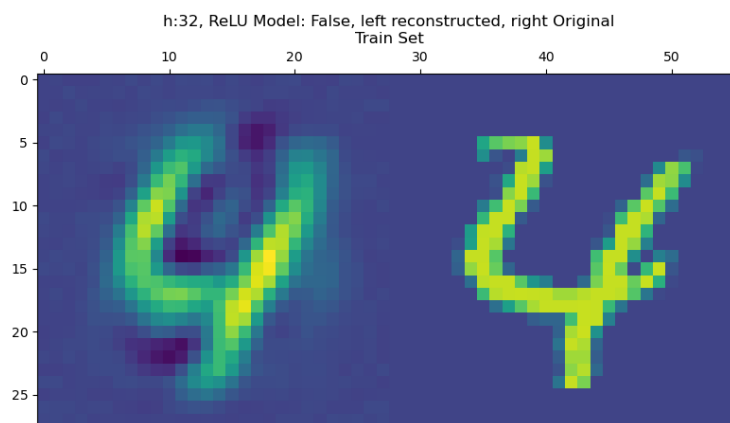
Appendix

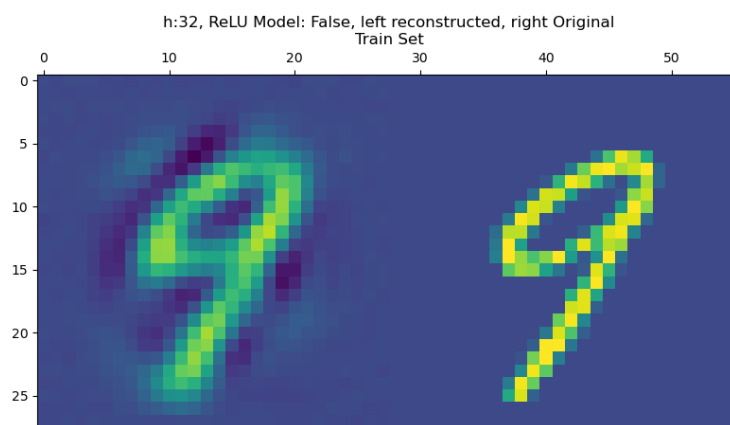
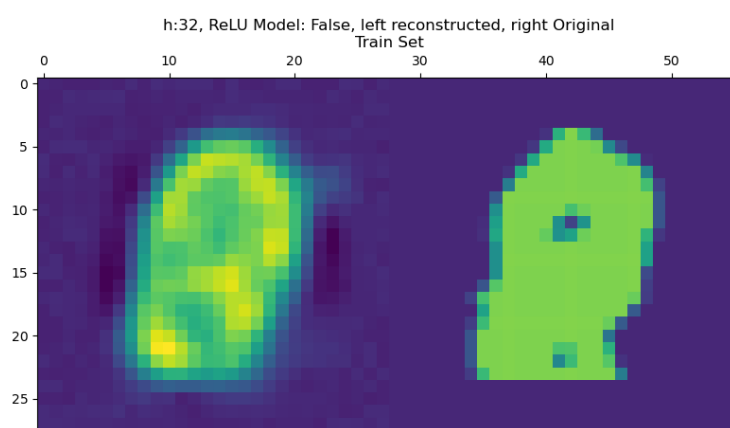
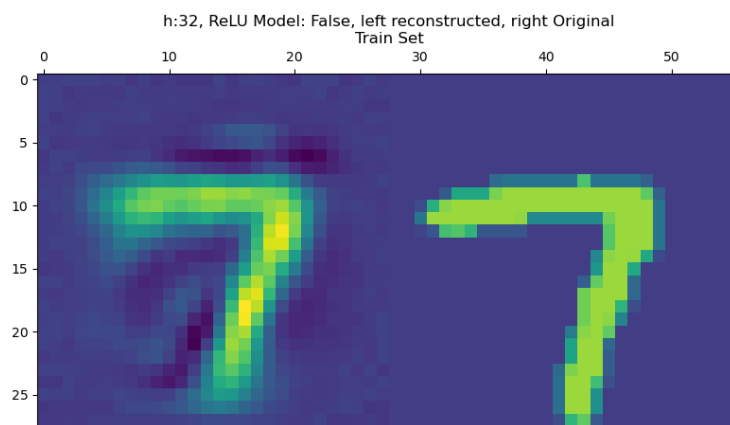
Extra A4Plots

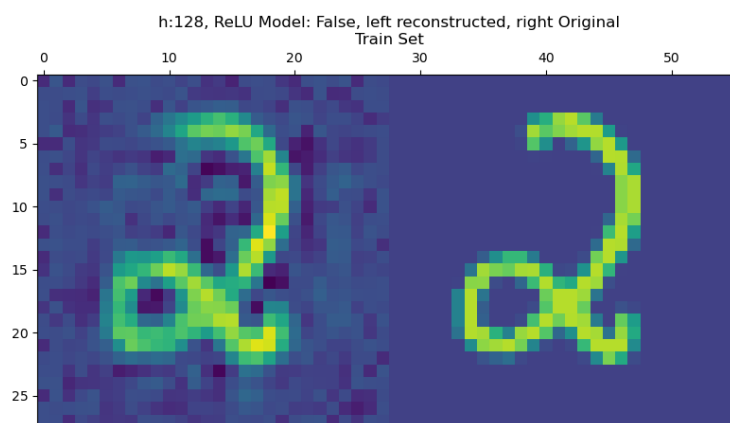
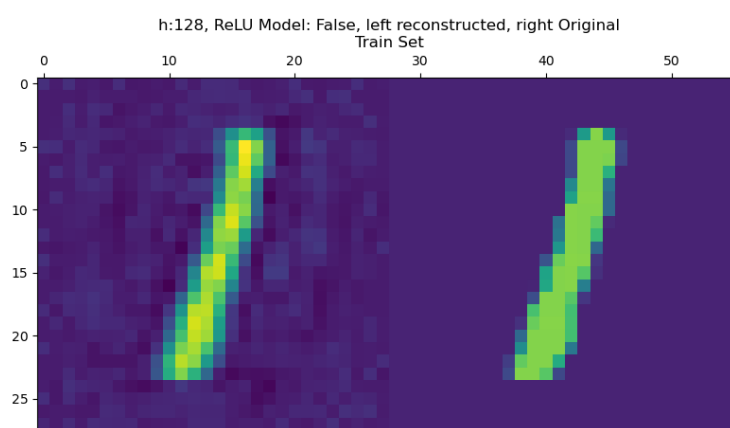
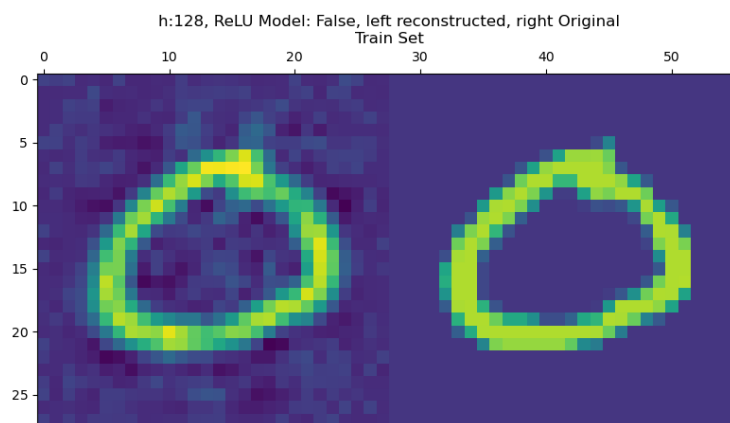
Extra Plots for Linear Model

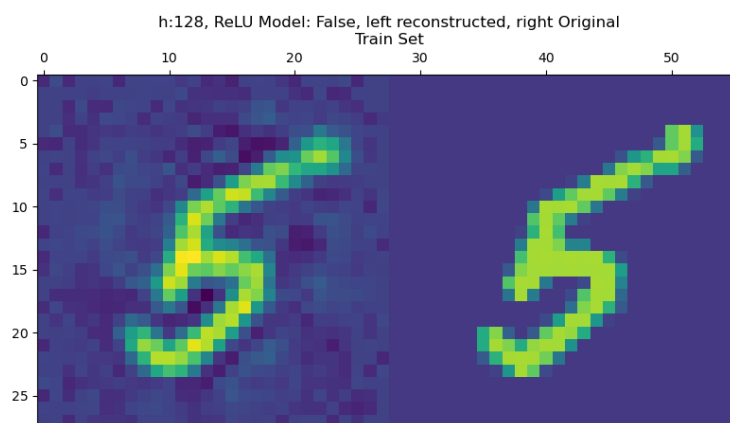
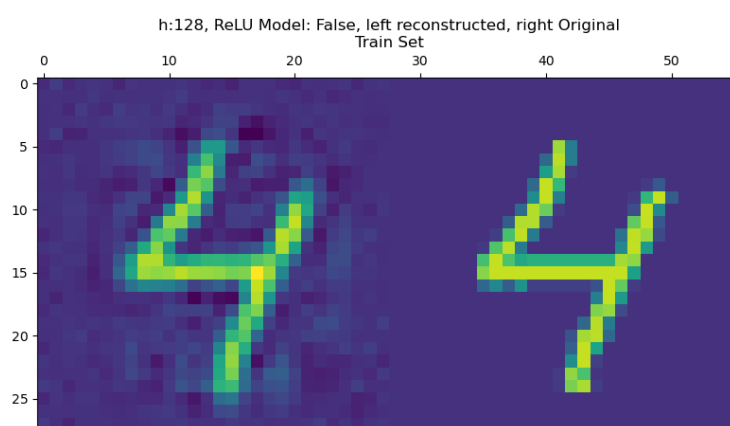
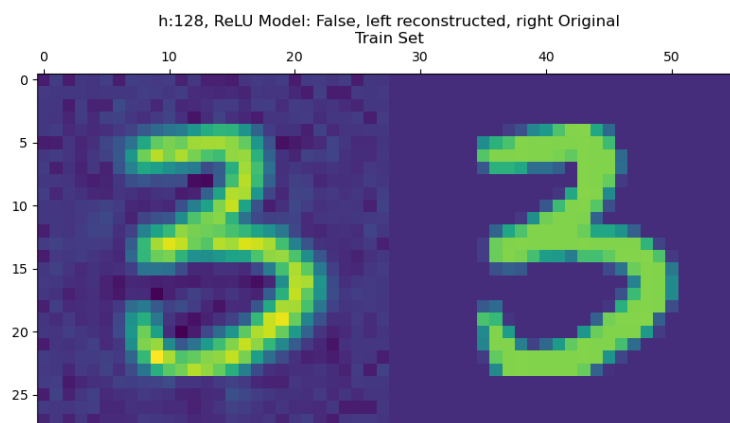


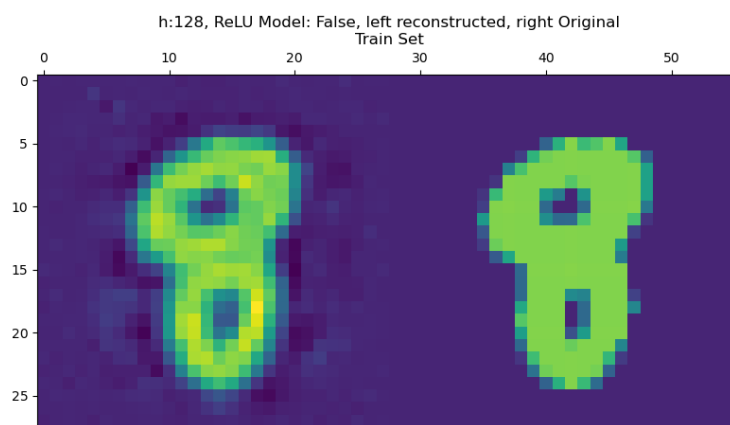
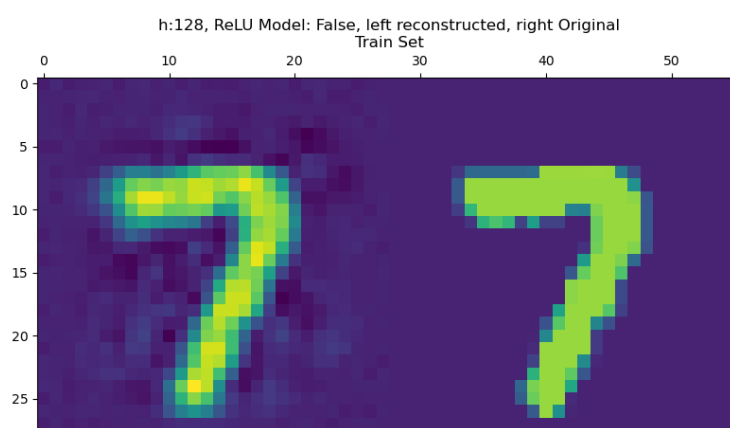
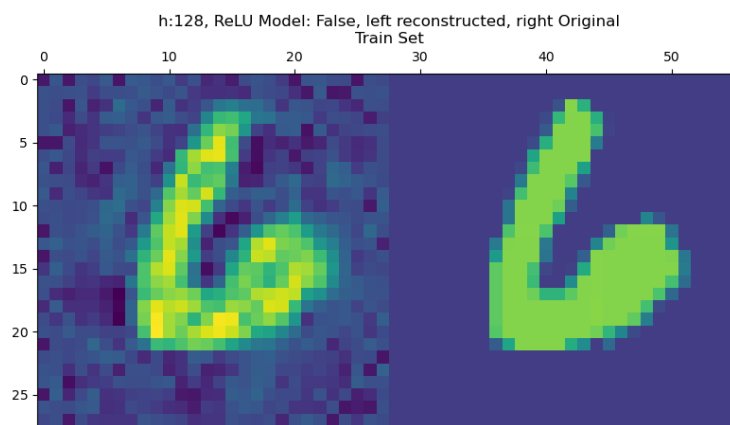


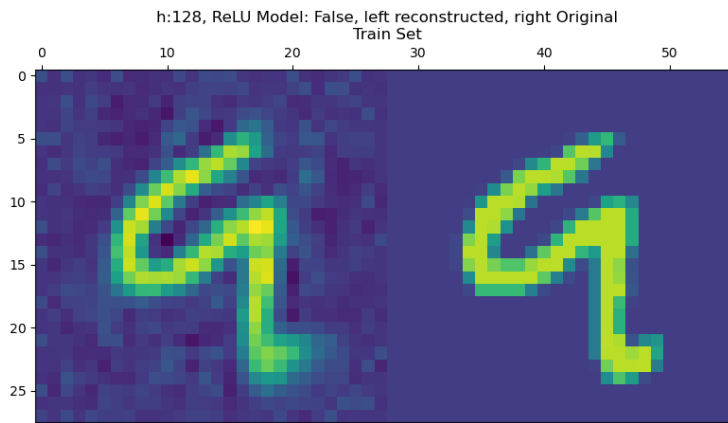












Extra Plots for Linear Model

