Name: Hongda Li   Class 546 SPRING 2021   HW4A

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:

$$\begin{aligned} X^TX &= (USV^T)^T(USV) \\ &= VS^TU^TUSV \\ &= VS^TSV \end{aligned} \tag{A1.b.1}$$

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

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^TX, eigen values decomposition is not unque 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 corssponds 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 algebaric multiplicity, then the rank of the matrix will be more than the number of eigenvalues it has.

A1.f

True, becaues Autoencoders with non-linear activation function can incooperate 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 incorperate the singular value decomposition for the convariance matrix. let's consider the gradient of the objective function set to zero. s

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

Using the Singular value decomposition we have:

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

We make use of the fact that, U, V are unitary matrices and the singular matrix Σ is a diagonal, containing all the singular vaues 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:

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

Taking the limit as $\lambda\to\infty$ 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. $(AB)^{-1}=B^{-1}B^{-1}$ assuming invertible A, B
5. $(AB)^T=B^TA^T$

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 $X^TX=V\Sigma^2V$ 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 $X^TX=VI_nV^T=I_n$ Becase 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:

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

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 choisce on i for a given particular x, we can make the claim that:

$$\partial[f] = \left(\inf_{i\in[m]}\{\partial[f_i]\}, \sup_{i\in[m]}\{\partial[f_i]\}\right) \tag{A2.0.1}$$

Basically, we can choose the i such to find a lower and upper bound for the sub gradient of f, a function defined via: $f(x) = \max_i\{f_i(x)\}$.

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 infinum 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 $g_i\mathbf{e}_i$:

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

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:

$$v_j \in \begin{cases} \{1\} & x_j > 0 \\ [-1, 1] & x_j = 0 \\ \{-1\} & x_j \leq 0 \end{cases} \tag{A2.b.1.3}$$

A2.b.(b)

Let λ_i be the set of coefficients for a convex combinations, meaning that $\sum_{i=1}^{n} \lambda_i = 1$ 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:

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

(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:

$$(\partial[f](x))_j = \left(\inf \{(\nabla[f_i](x))_j\}_{i=1}^{m}, \sup \{(\nabla[f_i](x))_j\}_{i=1}^{m}\right) \tag{A2.b.b.2}$$

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:

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

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

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

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

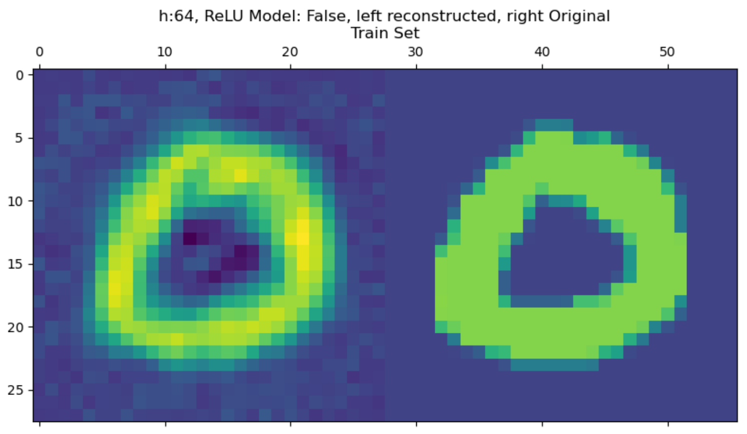
A4.a

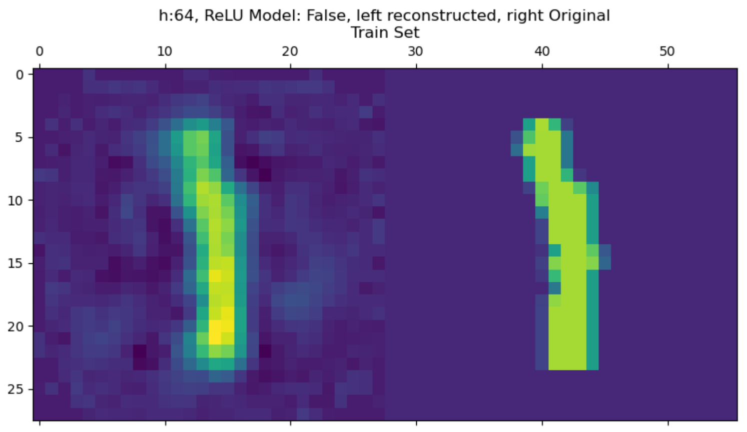
The Train Error for $h \in \{32, 64, 128\}$ for the linear models are:

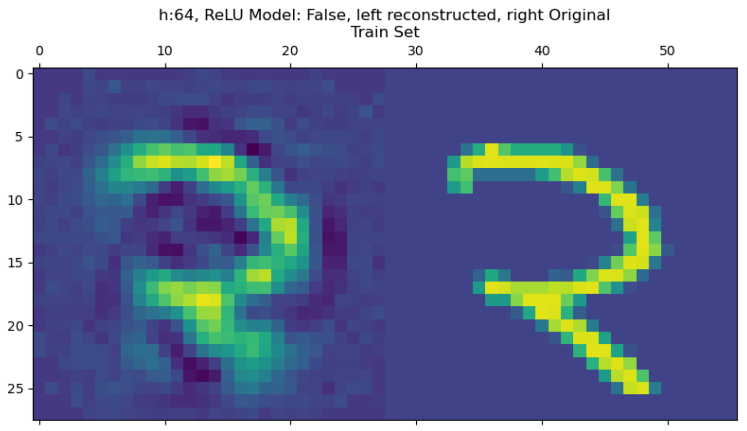
1. $h = 32$, Total Epochs: 30 Train Final MSE Loss: 0.07298633098602295
2. $h = 64$, Total Epochs: 30 Train Final MSE Loss: 0.07428810136703154
3. $h = 128$, Total Epochs: 30 Train Final MSE Loss: 0.058713519498705916

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.

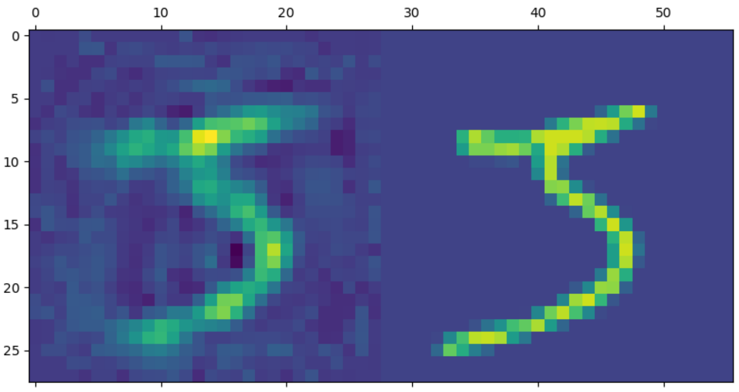
This is 10 digits reconstruction for $h = 64$ are below, and for all of the reconstruction plots for pairs of digits with $h = 32, 128$, please refer to Appendix.


h:64, ReLU Model: False, left reconstructed, right Original
Train Set

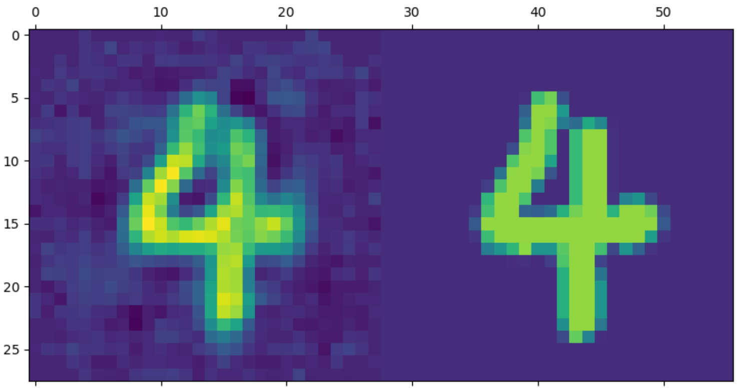

h:64, ReLU Model: False, left reconstructed, right Original
Train Set


h:64, ReLU Model: False, left reconstructed, right Original
Train Set

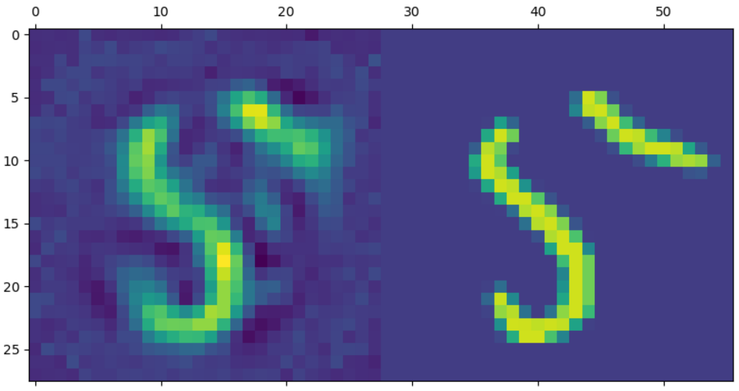
h:64, ReLU Model: False, left reconstructed, right Original
Train Set



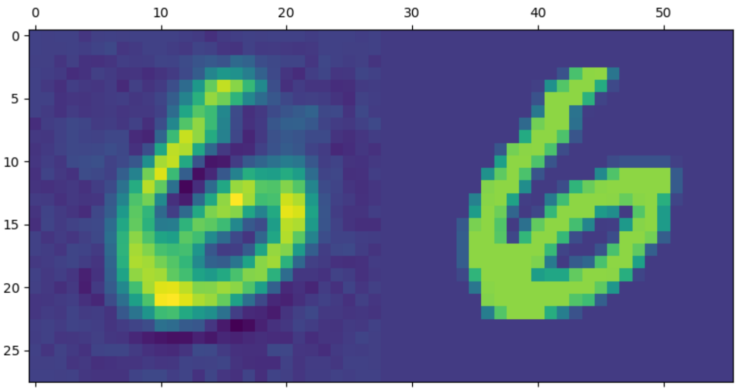
h:64, ReLU Model: False, left reconstructed, right Original
Train Set



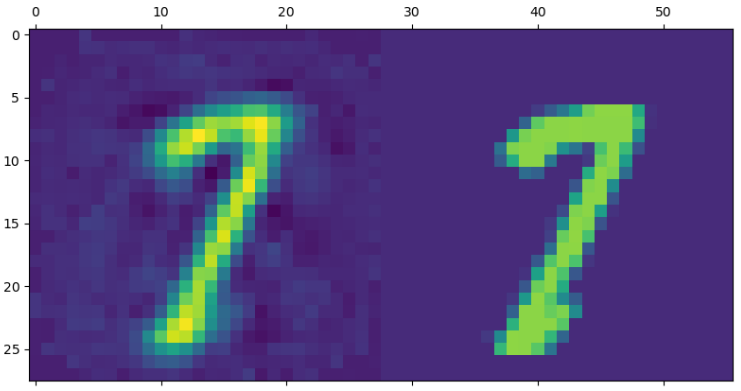
h:64, ReLU Model: False, left reconstructed, right Original
Train Set



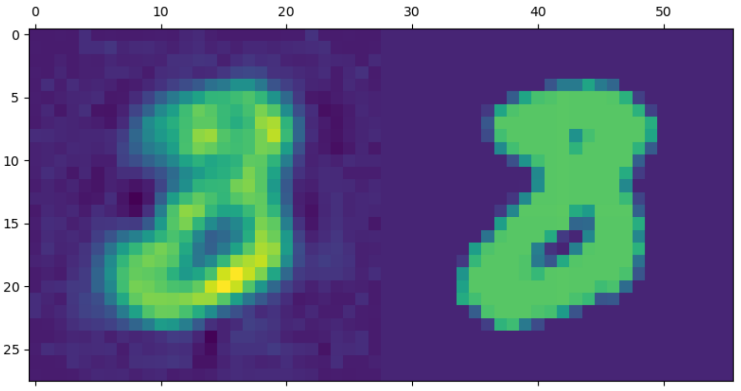
h:64, ReLU Model: False, left reconstructed, right Original
Train Set

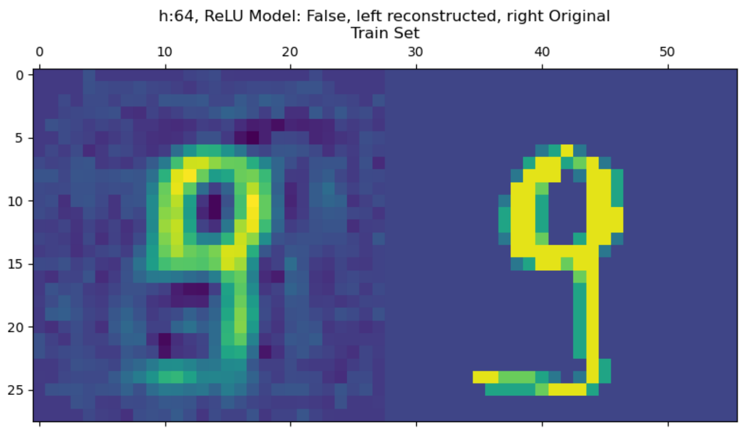


h:64, ReLU Model: False, left reconstructed, right Original
Train Set



h:64, ReLU Model: False, left reconstructed, right Original
Train Set



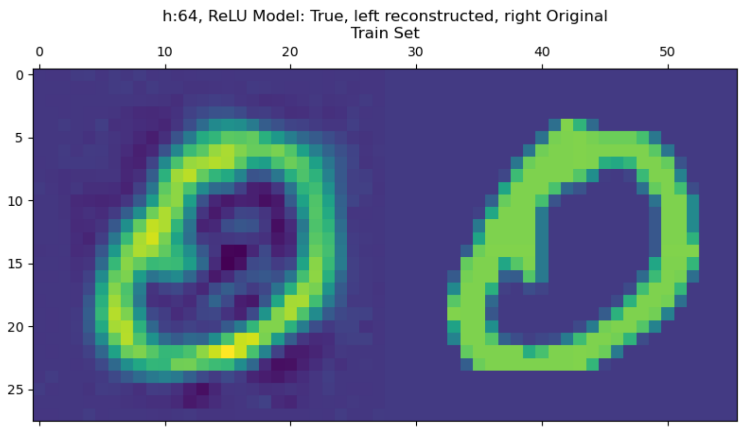


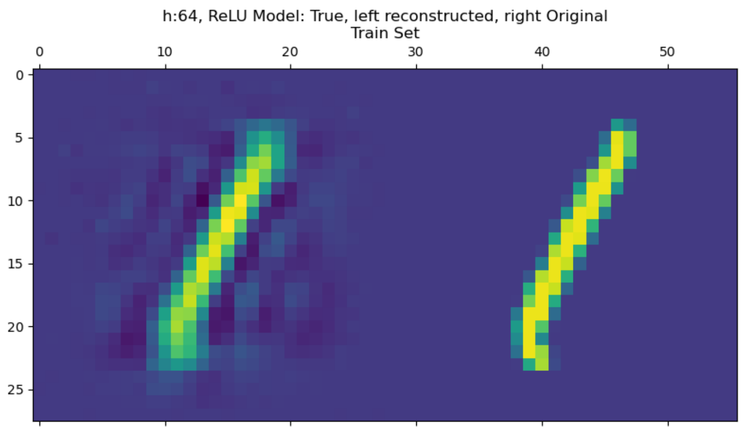
A4.b

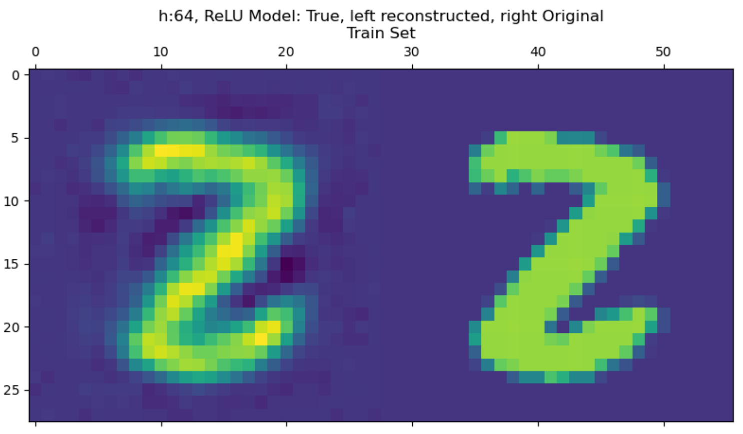
The train error for $h \in \{32, 64, 128\}$ for the non-linear model with ReLU activation is:

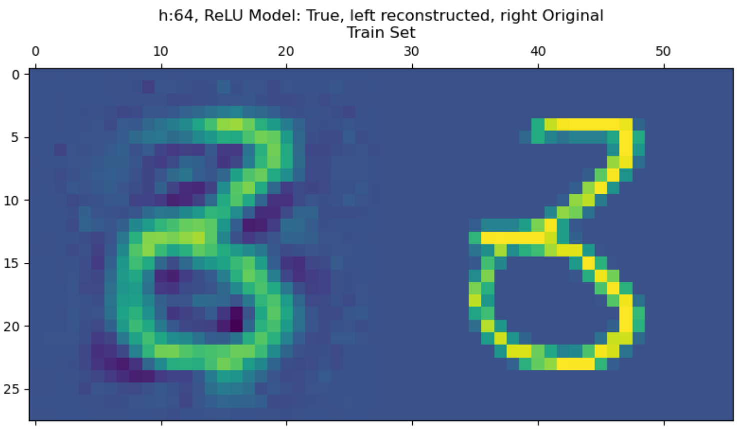
1. $h = 32$, total Epochs: 30 Train Final MSE Loss: 1.2326371114328512
2. $h = 64$, Total Epochs: 30 Train Final MSE Loss: 0.02494835205376 15
3. $h = 128$, Total Epochs: 30 Test Final MSE Loss: 0.024230041910583734

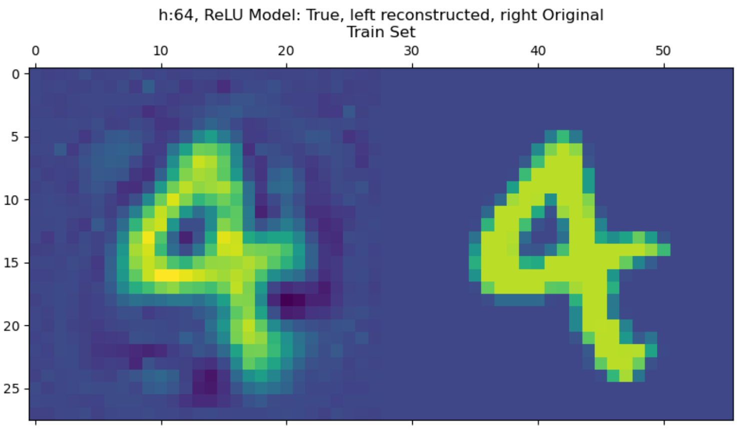
The MSE is computed the same as part A4.b. And these are some of the reconstruction images for $h = 64$:

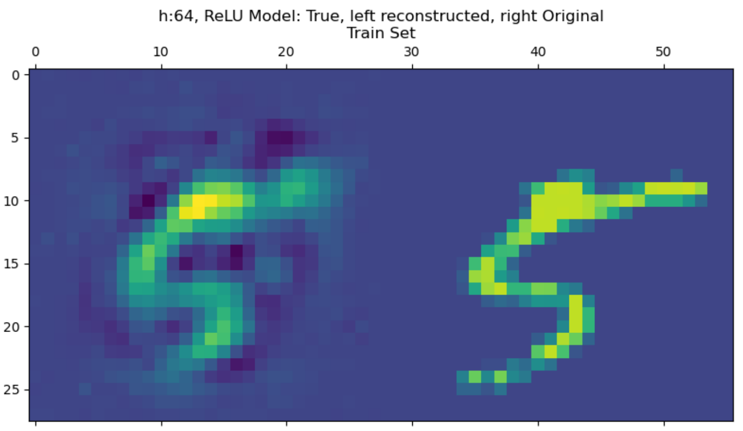


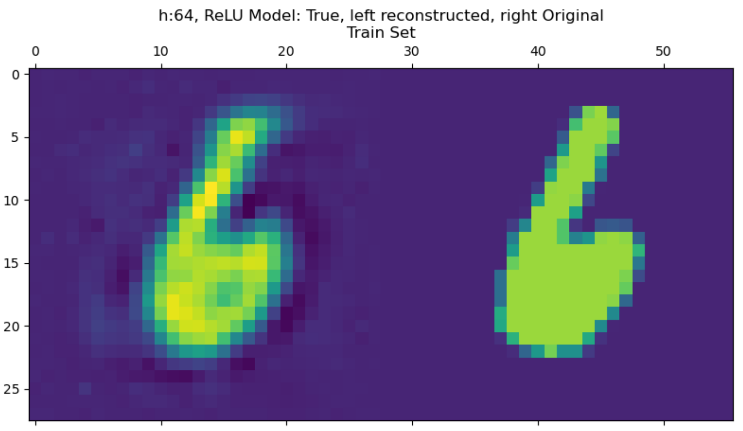


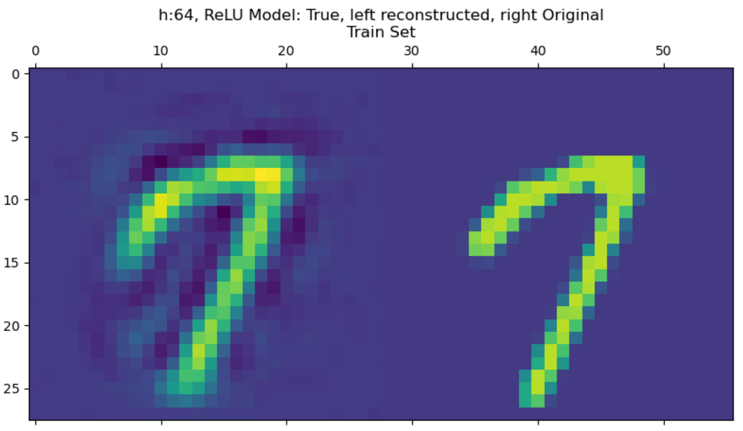

h:64, ReLU Model: True, left reconstructed, right Original
Train Set

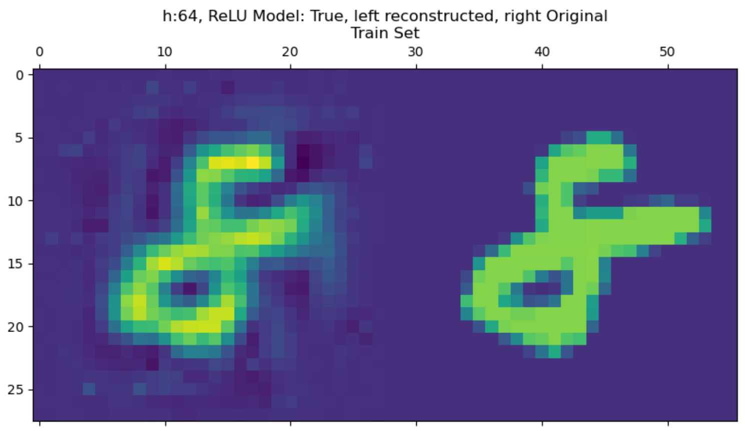

h:64, ReLU Model: True, left reconstructed, right Original
Train Set

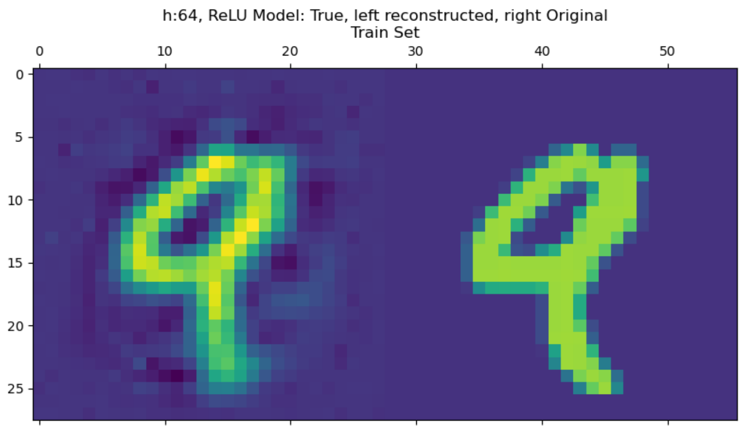

h:64, ReLU Model: True, left reconstructed, right Original
Train Set


h:64, ReLU Model: True, left reconstructed, right Original
Train Set
0
10
20
30
40
50
0
5
10
15
20
25


h:64, ReLU Model: True, left reconstructed, right Original
Train Set
0
10
20
30
40
50
0
5
10
15
20
25


h:64, ReLU Model: True, left reconstructed, right Original
Train Set
0
10
20
30
40
50
0
5
10
15
20
25





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**2
        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 GetEmbeding(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},␣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.curdir)
    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 Loyd'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):Cenroids␣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()
```

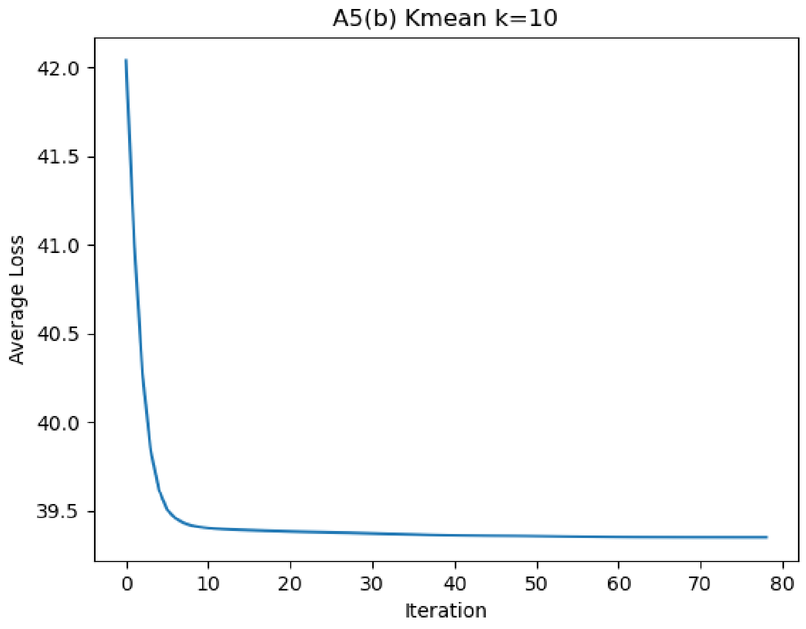
A5.b

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 $1e - 2$.

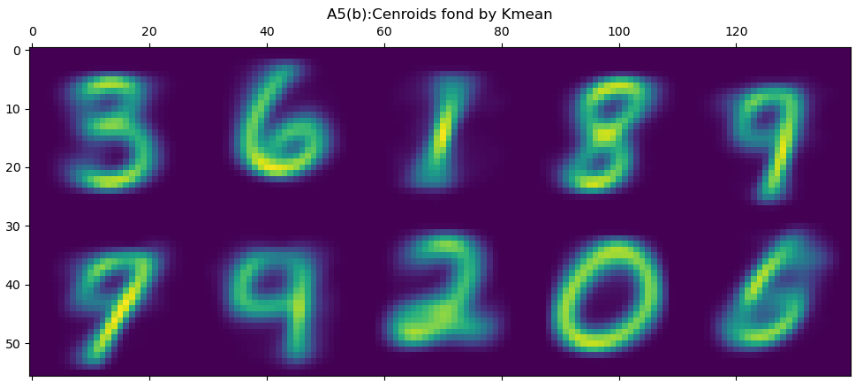
Note, here are some implementation details that might effect the average loss for each samples computed:

- : I normalized the MNIST data by dividing it by 255 so all the piexles values are in $[0, 1]$.
- : It's trained on the whole MNIST train datasset.

Code: A5.a

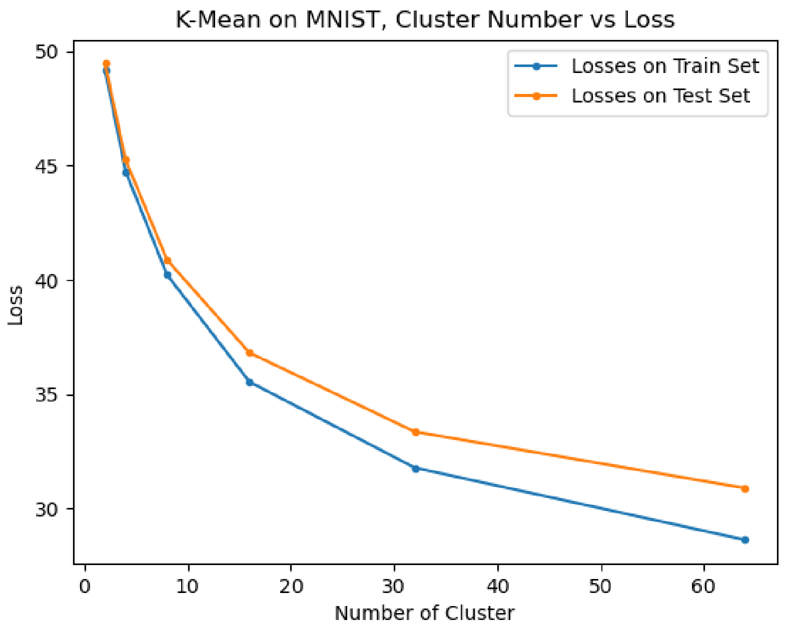


And these are 10 of the centroids identified by k-means, visualized as $28 \times 28 matrices$:



A5.c

For this part, I traied 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 plot against each value of k, and this is the graph:
Code A5.a

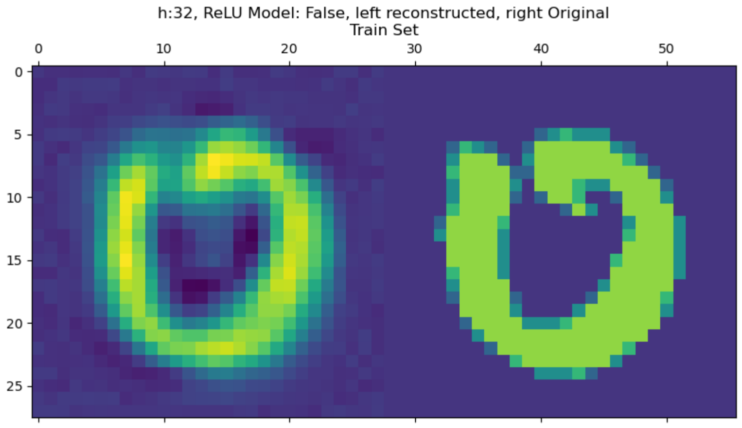


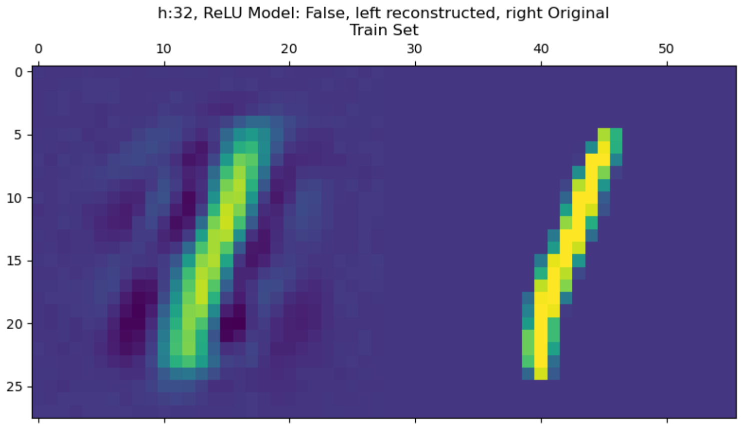
ML in the Real World

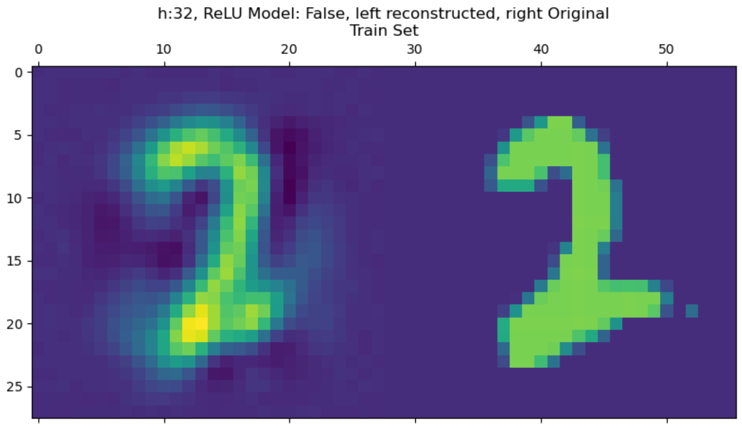
Appendix

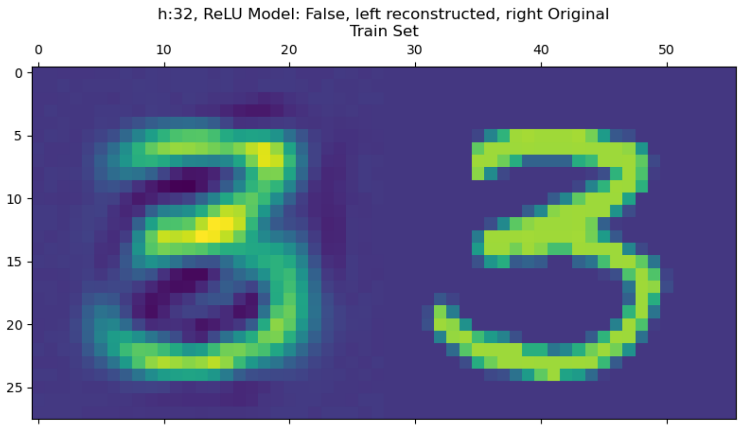
Extra A4Plots

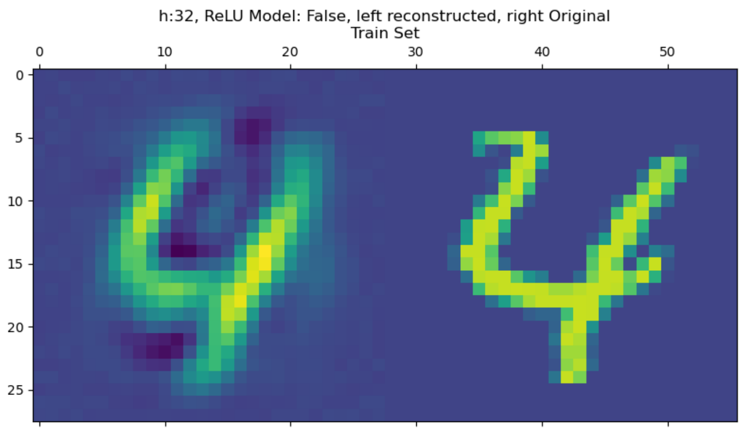
Extra Plots for Linear Model

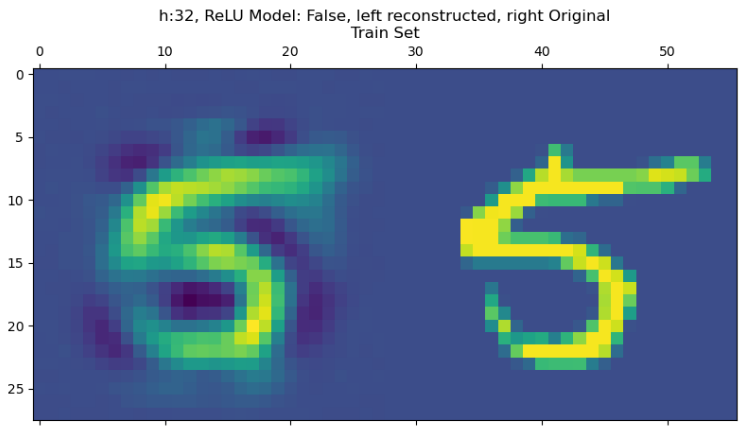


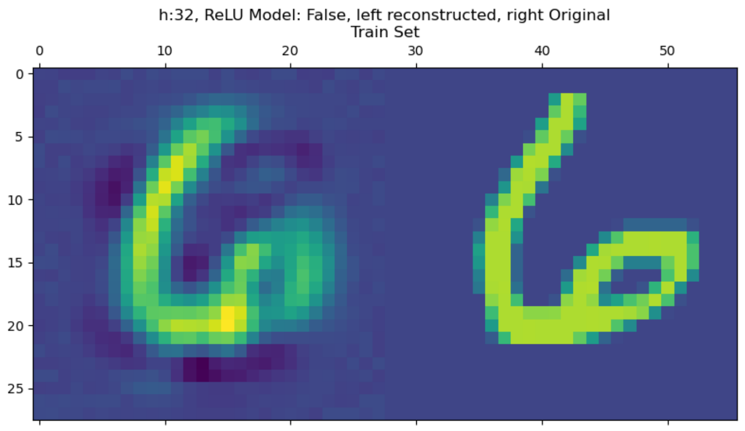

h:32, ReLU Model: False, left reconstructed, right Original
Train Set
0
10
20
30
40
50
0
5
10
15
20
25

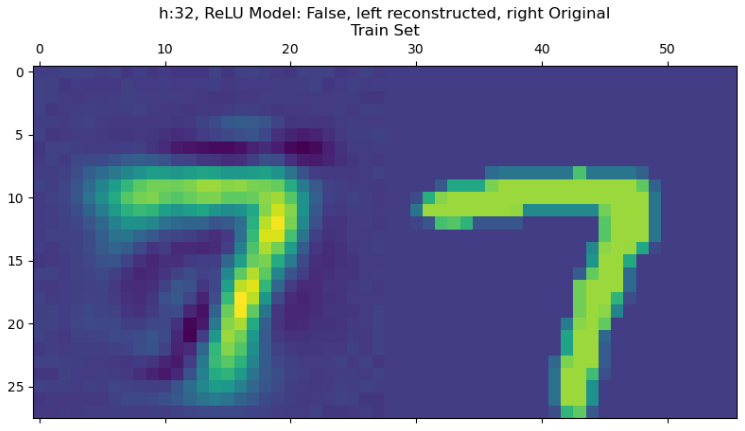

h:32, ReLU Model: False, left reconstructed, right Original
Train Set
0
10
20
30
40
50
0
5
10
15
20
25

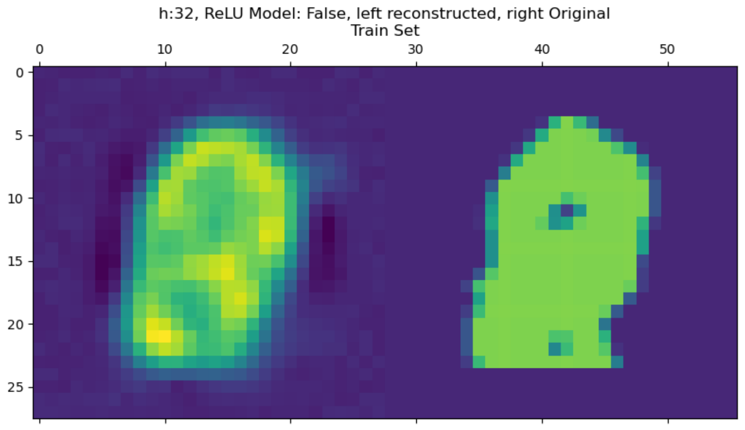

h:32, ReLU Model: False, left reconstructed, right Original
Train Set
0
10
20
30
40
50
0
5
10
15
20
25

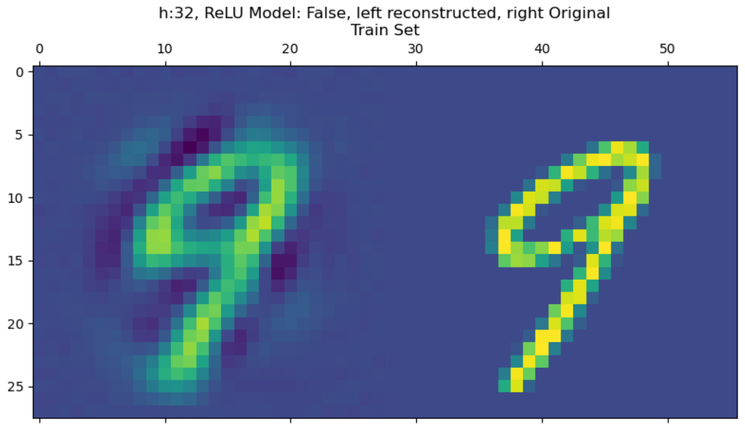

h:32, ReLU Model: False, left reconstructed, right Original
Train Set

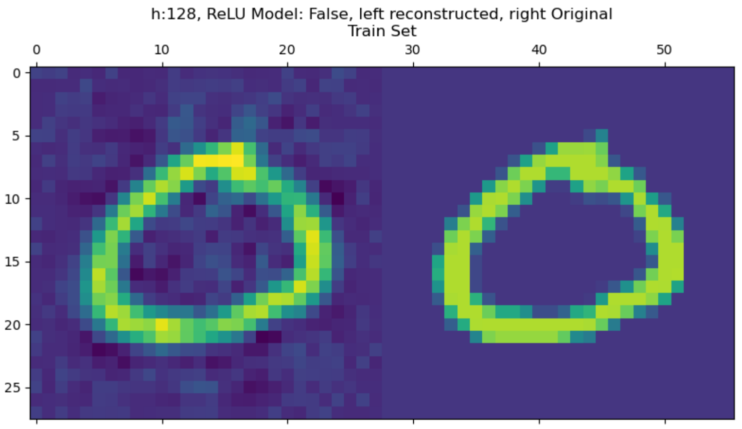

h:32, ReLU Model: False, left reconstructed, right Original
Train Set

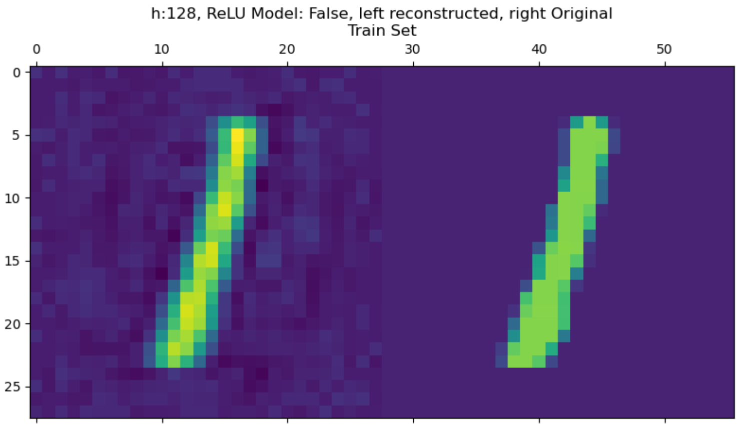

h:32, ReLU Model: False, left reconstructed, right Original
Train Set

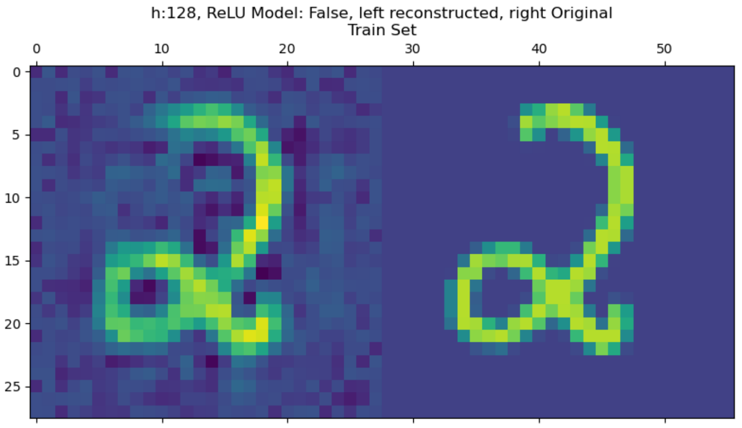

h:32, ReLU Model: False, left reconstructed, right Original
Train Set


h:32, ReLU Model: False, left reconstructed, right Original
Train Set

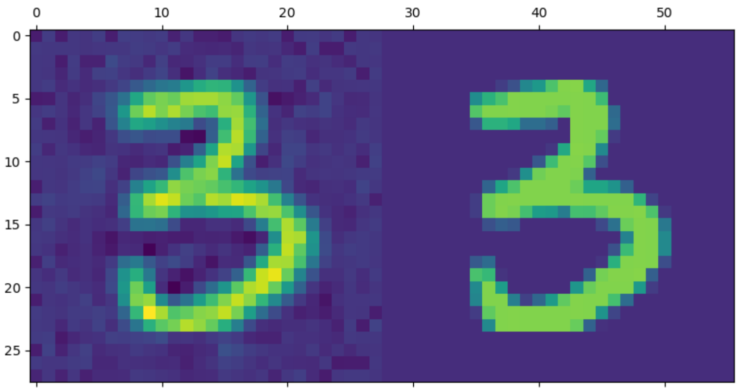

h:32, ReLU Model: False, left reconstructed, right Original
Train Set


h:128, ReLU Model: False, left reconstructed, right Original
Train Set

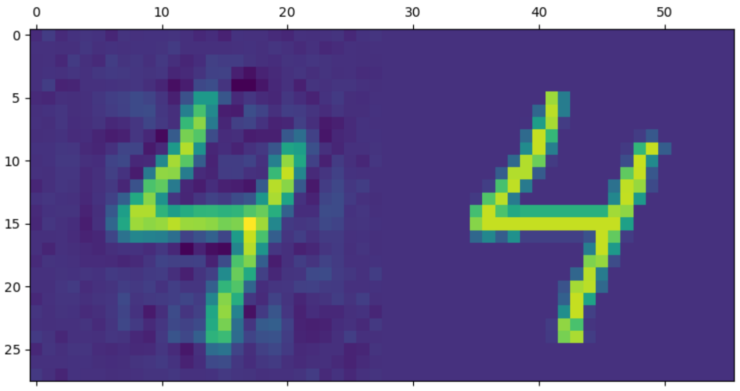

h:128, ReLU Model: False, left reconstructed, right Original
Train Set


h:128, ReLU Model: False, left reconstructed, right Original
Train Set

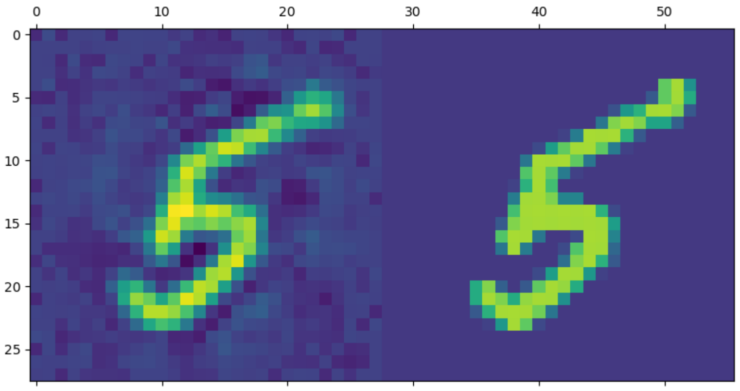
h:128, ReLU Model: False, left reconstructed, right Original
Train Set

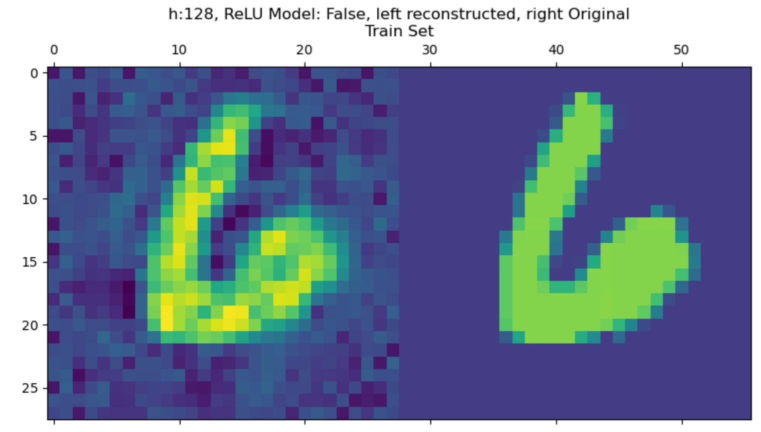


h:128, ReLU Model: False, left reconstructed, right Original
Train Set

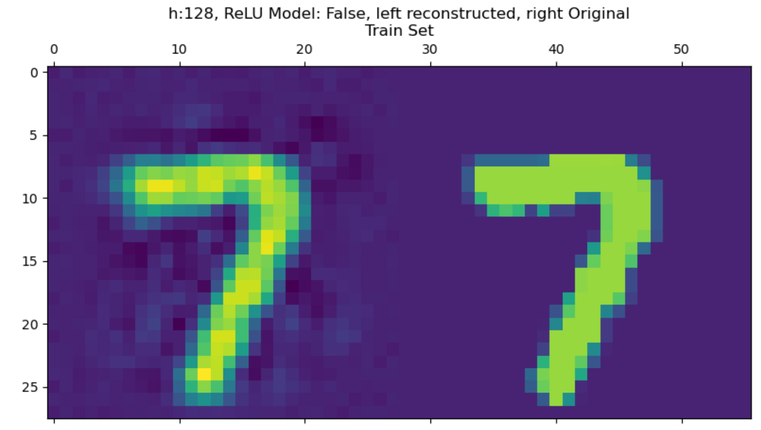


h:128, ReLU Model: False, left reconstructed, right Original
Train Set

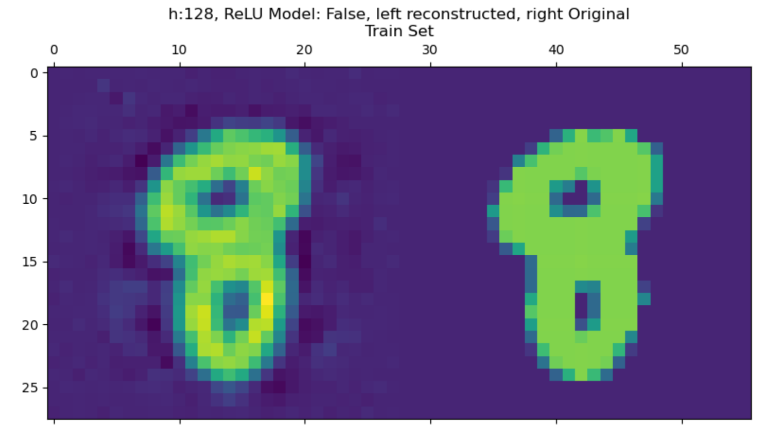




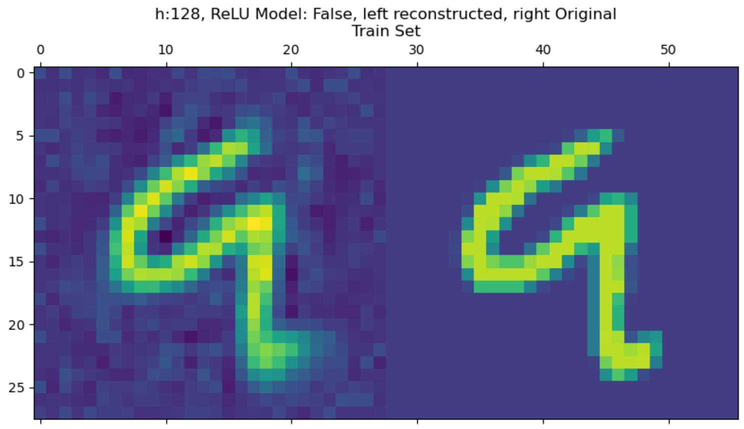
h:128, ReLU Model: False, left reconstructed, right Original
Train Set



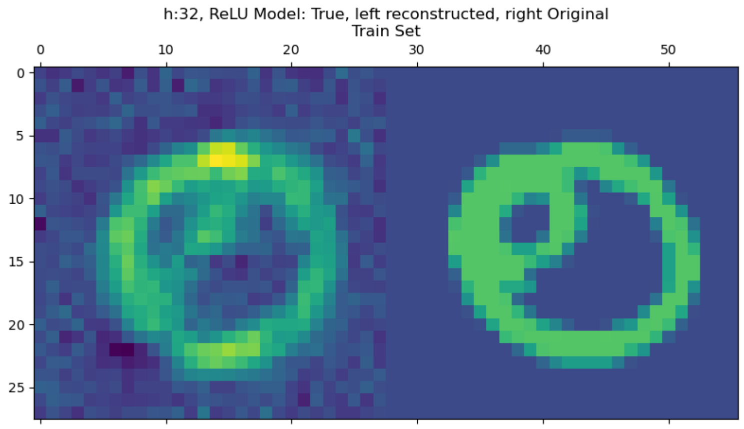
h:128, ReLU Model: False, left reconstructed, right Original
Train Set

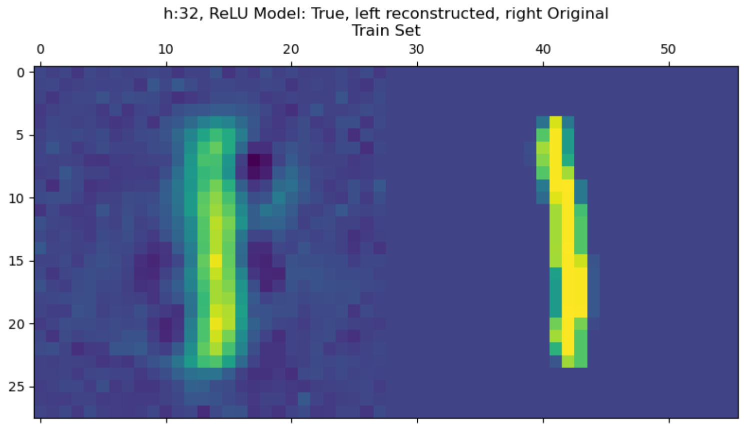


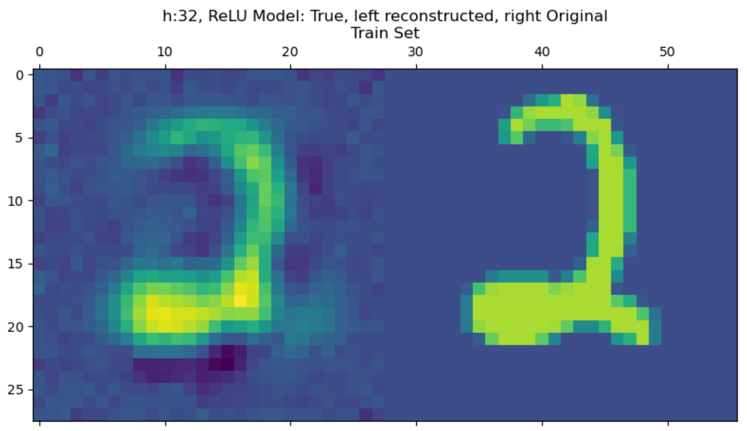
h:128, ReLU Model: False, left reconstructed, right Original
Train Set

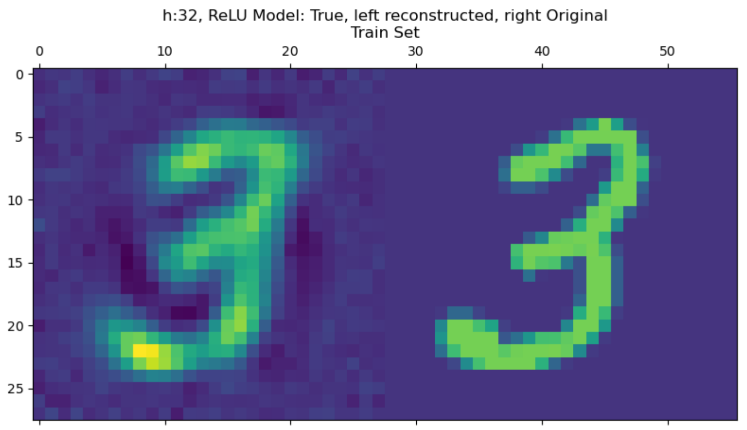


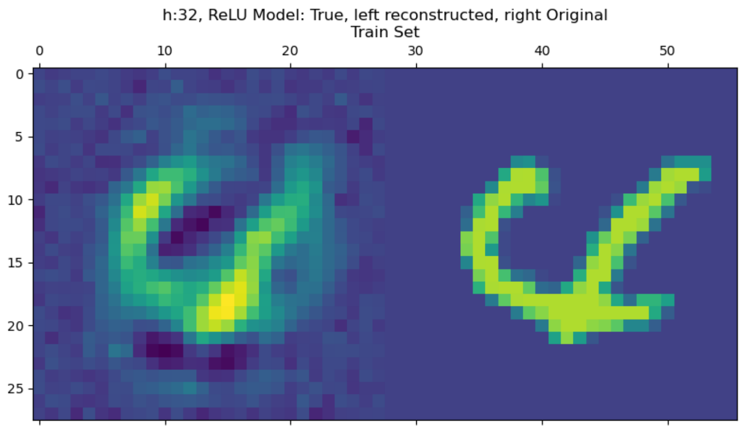
Extra Plots for Linear Model

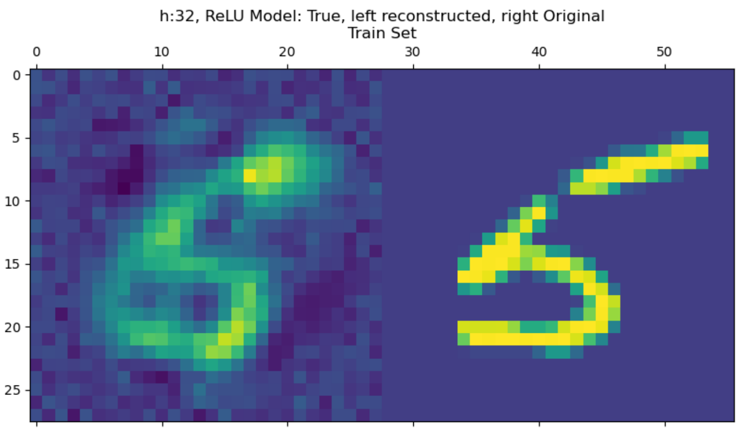


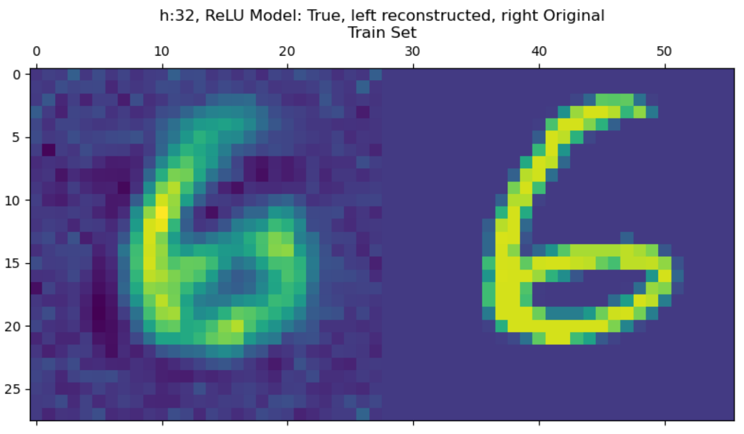


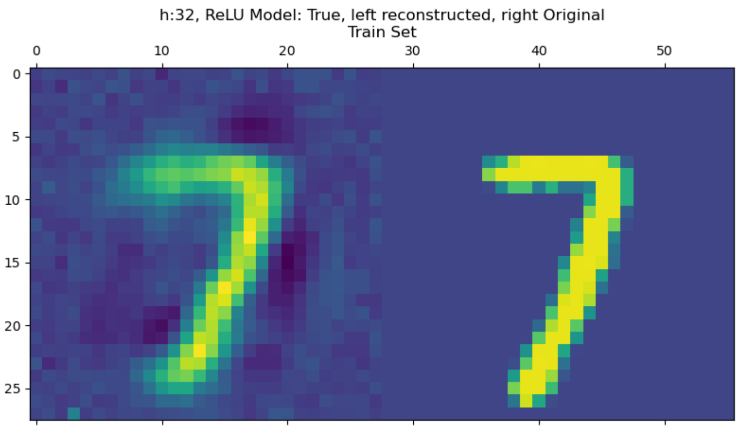

h:32, ReLU Model: True, left reconstructed, right Original
Train Set
0
10
20
30
40
50
0
5
10
15
20
25


h:32, ReLU Model: True, left reconstructed, right Original
Train Set
0
10
20
30
40
50
0
5
10
15
20
25

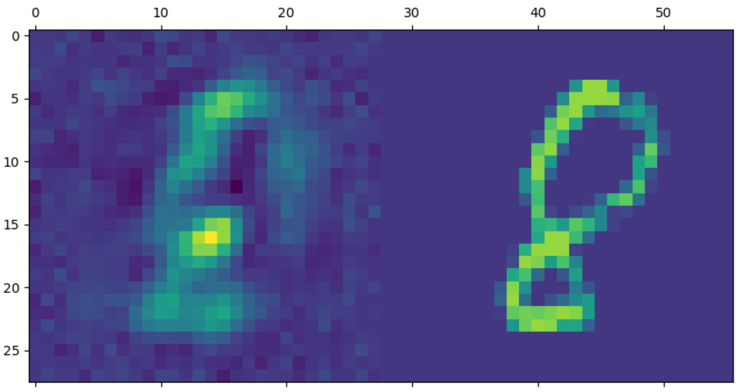

h:32, ReLU Model: True, left reconstructed, right Original
Train Set
0
10
20
30
40
50
0
5
10
15
20
25


h:32, ReLU Model: True, left reconstructed, right Original
Train Set

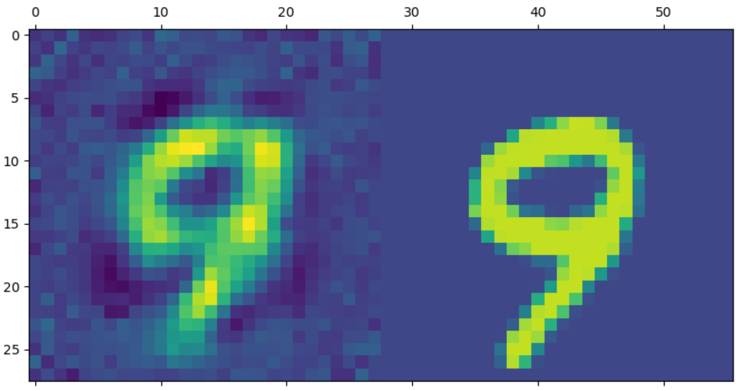

h:32, ReLU Model: True, left reconstructed, right Original
Train Set


h:32, ReLU Model: True, left reconstructed, right Original
Train Set

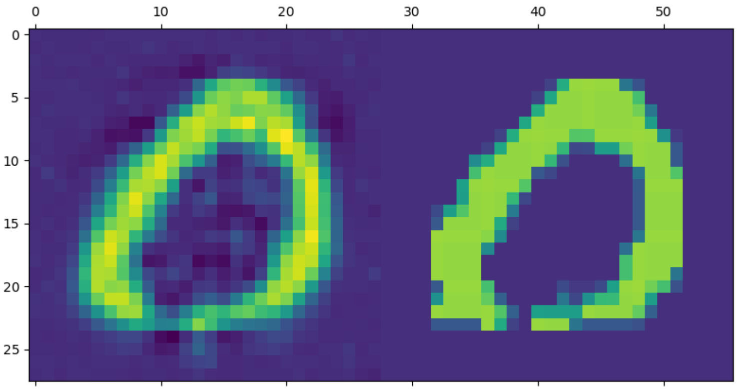
h:32, ReLU Model: True, left reconstructed, right Original
Train Set



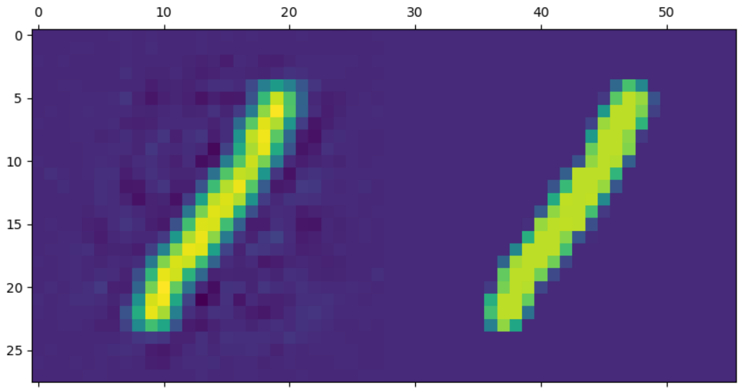
h:32, ReLU Model: True, left reconstructed, right Original
Train Set



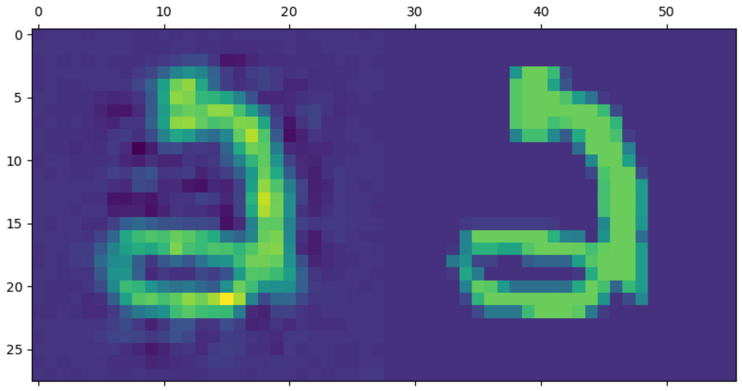
h:128, ReLU Model: True, left reconstructed, right Original
Train Set



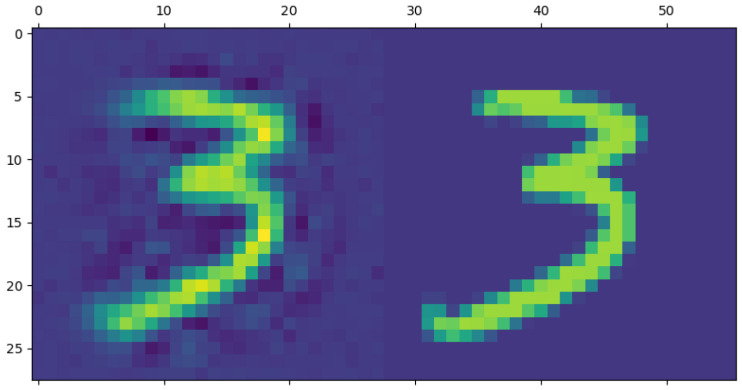
h:128, ReLU Model: True, left reconstructed, right Original
Train Set

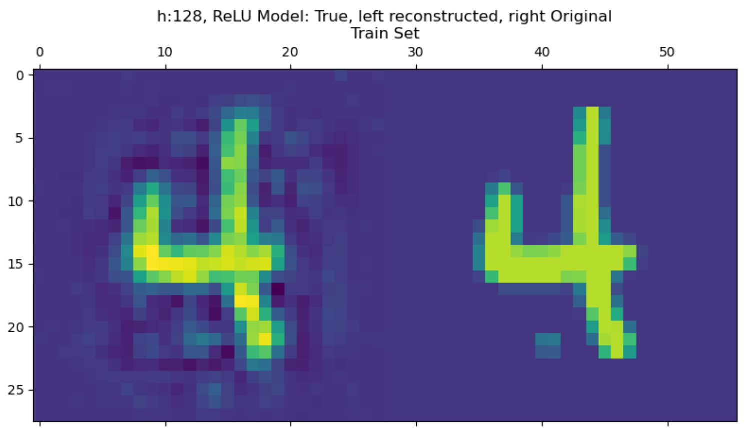


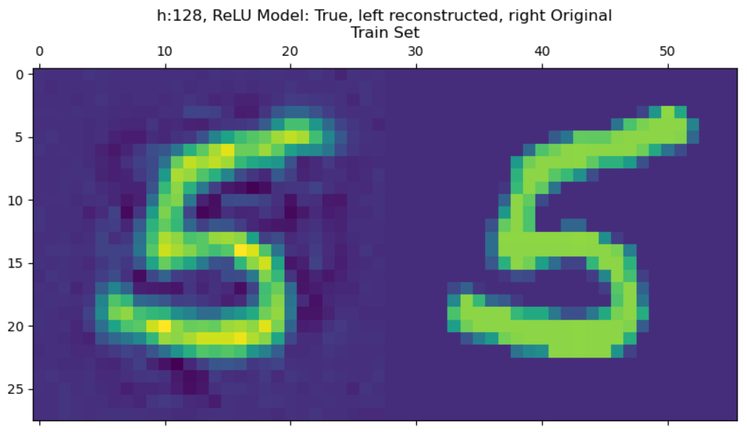
h:128, ReLU Model: True, left reconstructed, right Original
Train Set

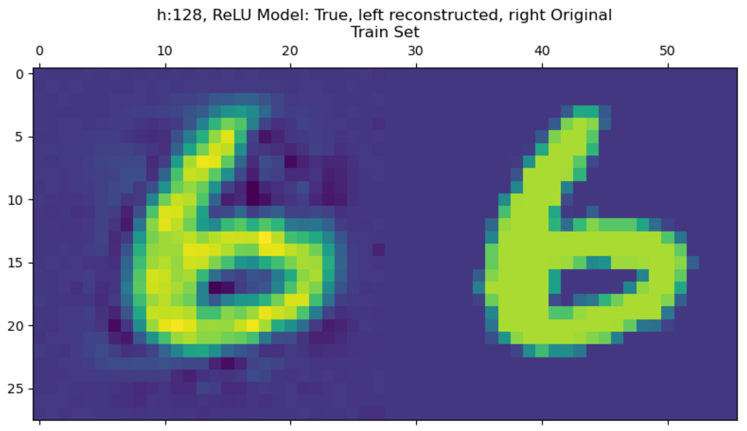


h:128, ReLU Model: True, left reconstructed, right Original
Train Set

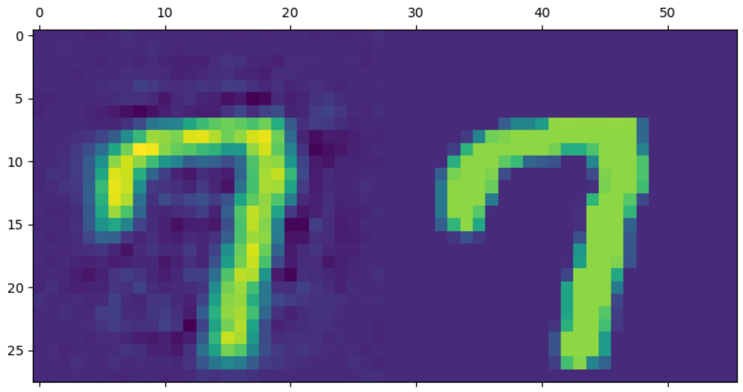



h:128, ReLU Model: True, left reconstructed, right Original
Train Set

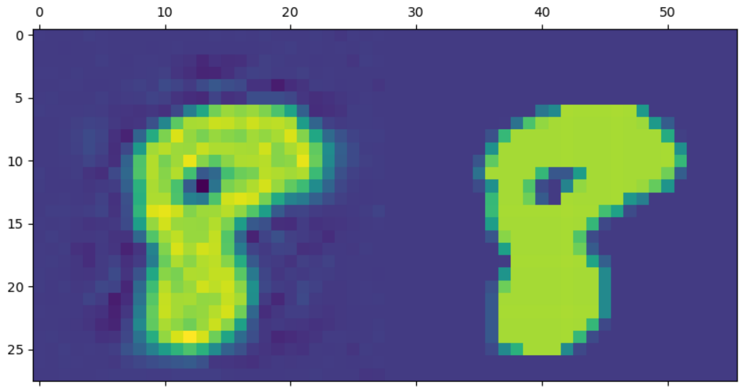

h:128, ReLU Model: True, left reconstructed, right Original
Train Set


h:128, ReLU Model: True, left reconstructed, right Original
Train Set

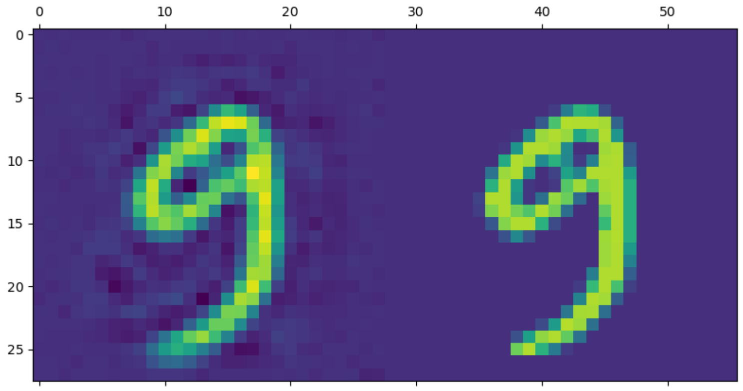
h:128, ReLU Model: True, left reconstructed, right Original
Train Set



h:128, ReLU Model: True, left reconstructed, right Original
Train Set



h:128, ReLU Model: True, left reconstructed, right Original
Train Set



0
10
20
30
40
50
0
5
10
15
20
25