Homework #4

CSE 446/546: Machine Learning
Profs. Jamie Morgenstern and Simon Du
Due: **Wednesday** December 8, 2021 11:59pm
A: 106 points, **B:** 50 points

Please review all homework guidance posted on the website before submitting to GradeScope. Reminders:

- Make sure to read the "What to Submit" section following each question and include all items.
- Please provide succinct answers and supporting reasoning for each question. Similarly, when discussing experimental results, concisely create tables and/or figures when appropriate to organize the experimental results. All explanations, tables, and figures for any particular part of a question must be grouped together.
- For every problem involving generating plots, please include the plots as part of your PDF submission.
- When submitting to Gradescope, please link each question from the homework in Gradescope to the location of its answer in your homework PDF. Failure to do so may result in deductions of up to *[5 points]* . For instructions, see `https://www.gradescope.com/get_started#student-submission`.
- Please recall that B problems, indicated in $\boxed{\text{boxed text}}$, are only graded for 546 students, and that they will be weighted at most 0.2 of your final GPA (see the course website for details). In Gradescope, there is a place to submit solutions to A and B problems separately. You are welcome to create a single PDF that contains answers to both and submit the same PDF twice, but associate the answers with the individual questions in Gradescope.
- If you collaborate on this homework with others, you must indicate who you worked with on your homework. Failure to do so may result in accusations of plagiarism.
- For every problem involving code, please submit your code to the separate assignment on Gradescope created for code. Not submitting all code files will lead to a deduction of *[1 point]* .
- Please indicate your final answer to each question by placing a box around the main result(s). To do this in LaTeX, one option is using the `boxed` command.
- You may choose **only one BLUE problem from Image Classification on CIFAR-10 and Text classification on SST-2** to complete (please do not turn in both).

Not adhering to these reminders may result in point deductions.

Changelog:

- **Date: 11/28** Updated reconstruction_error test in PCA.
- **Date: 11/28** Removed square from k-means objective function.
- **Date: 11/28** Added colab notebook for B1 problem.

Conceptual Questions

A1. The answers to these questions should be answerable without referring to external materials. Briefly justify your answers with a few words.

a. *[2 points]* True or False: Given a data matrix $X \in R^{n \times d}$ where d is much smaller than n and $k = \text{rank}(X)$, if we project our data onto a k dimensional subspace using PCA, our projection will have zero reconstruction error (in other words, we find a perfect representation of our data, with no information loss).

Solution: True. The rank of the matrix refers to the number of linearly independent rows or columns in the data matrix. If we project our data onto a k dimensional subspace using PCA, we can find a perfect representation of the data because $rank(X) = rank(X^\top X)$ and the rank of $(X^\top X)$ is the number of non-zero eigenvalues. Therefore, the covariance matrix will only have k nonzero eigenvalues. If we project our data onto a k dimensional subspace using PCA, then the sum of eigenvalues remaining =0 and all of the variance is captured by the first k eigenvalues. Therefore, we would find a perfect representation of our data.

---

b. *[2 points]* True or False: Suppose that an $n \times n$ matrix X has a singular value decomposition of $USV^\top$, where S is a diagonal $n \times n$ matrix. Then, the rows of V are equal to the eigenvectors of $X^\top X$.

Solution: False. The columns of V are equal to the eigenvectors of $X^\top X$ and not the rows of V. However, the rows of $V^\top$ will be equal to the eigenvectors of $X^\top X$.

---

c. *[2 points]* True or False: choosing k to minimize the k-means objective (see Equation (1) below) is a good way to find meaningful clusters.

Solution: False. The error decreases with k so the choosing k to minimize the k-means objective would not give us meaningful clusters. If we have a n distinct data points, then minimizing over the number of clusters k would give us $k = n$ and so each cluster has one data point. The $k-$ means objective would be zero and so it would have been minimized but the clusters are not meaningful. If we consider the MNIST dataset, we would want 10 clusters so we can assign each image of a digit to a cluster of that digit. However, minimizing over k would create clusters for each of the images which does not make much sense.

---

d. *[2 points]* True or False: The singular value decomposition of a matrix is unique.

Solution: False. Recall: the singular value decomposition of a matrix $X = U\Sigma V^\top$ where Σ is a diagonal matrix with singular values. These singular values are uniquely determined. However, U and V may not be unique. When the singular values are distinct, columns of U and V are unique upto the sign of the vectors. The magnitudes are uniquely determined but they could be positive or negative. When the singular values are not distinct then we cannot say anything about the uniqueness of U and V which means that we cannot say anything about the uniqueness of SVD.

e. *[2 points]* True or False: The rank of a square matrix equals the number of its nonzero eigenvalues.

Solution: False. Consider the counter example:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

This is a square matrix with all of its eigenvalues equal to zero. Therefore, according to the statement the rank should be equal to zero. However, the rank of this matrix is 3.

We can say that the number of non zero eigenvalues of a square matrix$\leq$ rank. This has to do with the rank nullity theorem since if the nullspace is non-empty then there exists v such that $Av = 0v$ and so 0 is an eigenvalue. The null space then must be at least one-dimensional and by the rank-nullity theorem we have the above result.

The rank of a symmetric matrix, however, equals the number of its nonzero eigenvalues.

---

f. *[2 points]* True or False: Autoencoders, where the encoder and decoder functions are both neural networks with nonlinear activations, can capture more variance of the data in its encoded representation than PCA using the same number of dimensions.

Solution: Uncertain. This will depend on how complex the underlying features are. If the features are simple and can be looked through the lens of linear maps only, PCA will capture more variance of the data using the same dimensions because it is designed to maximize the amount of variance in the reduced dimensions. If the underlying structure is linear then it is possible that the nonlinear activation functions messes up and performs worse than PCA. However, if the images are complex and there is underlying non-linearity then autoencoders will capture more of the variance in the data by the use of its nonlinear activation functions.

Think before you train

A2. **The first part of this problem (parts a, b)** explores how you would apply machine learning theory and techniques to real-world problems. There are two scenarios detailing a setting, a dataset, and a specific result we hope to achieve. Your job is to describe how you would handle each of the below scenarios with the tools we've learned in this class. Your response should include

(1) any pre-processing steps you would take (i.e., data acquisition and processing),

(2) the specific machine learning pipeline you would use (i.e., algorithms and techniques learned in this class),

(3) how your setup acknowledges the constraints and achieves the desired result.

You should also aim to leverage some of the theory we have covered in this class. Some things to consider may be: the nature of the data (i.e., *How hard is it to learn? Do we need more data? Are the data sources good?*), the effectiveness of the pipeline (i.e., *How strong is the model when properly trained and tuned?*), and the time needed to effectively perform the pipeline.

a. *[5 points]* **Scenario 1: Disease Susceptibility Predictor**

- Setting: You are tasked by a research institute to create an algorithm that learns the factors that contribute most to acquiring a specific disease.
- Dataset: A rich dataset of personal demographic information, location information, risk factors, and whether a person has the disease or not.
- Result: The company wants a system that can determine how susceptible someone is to this disease when they enter in personal information. The pipeline should take limited amount of personal data from a new user and infer more detailed metrics about the person.

Solution:

(1) We have a dataset of personal demographic information, location information, risk factors and whether a person has the disease or not. The demographic information might involve a lot of categorical data like "Race", "Gender", "Ethnicity" etc. If we want to use Neural Networks, we should use one hot encoding on these categorical variables so that they can be fed into the machine learning pipeline. If we decide to use a decision tree, then we would not require one hot encoding on these categorical variables. We should perform some sort of feature extraction. It is possible that many of our features are incomparable- so we should standardize our data. We should also account for collinearity and try to extract independent features using SVD/PCA.

(2) Given a disease, we could view this scenario as a binary classification problem: person has the disease or not. We should use a neural network with a hidden layer and non linear activation function. The activation function I would choose is ReLu because we ideally want the machine learning pipeline to spit out a probability that a person has the disease or not. We want to use non linear activation because I am not sure whether the underlying features are linear. I am doubtful about that and that is why I am not using a neural network with no hidden layers (logistic regression). While training we need to have a threshold such that if probability given by the pipeline is greater than that then the person has the disease. We can then use that to find the accuracy of our model. This threshold could be a hyperparameter which we can train using a validation set.

---

b. *[5 points]* **Scenario 2: Social Media App Facial Recognition Technology**

- Setting: You are tasked with developing a machine learning pipeline that can quickly map someone's face for the application of filters (i.e., Snapchat, Instagram).
- Dataset: A set of face images compiled from the company's employees and their families.
- Result: The company wants an algorithm that can quickly identify the key features of a person's face to apply a filter. (**Note:** Do not worry about describing the actual filter application).

Solution:

(1) The dataset consists of face images compiled from the company's employees and their families. We need to preprocess the image data. We should check for any blurry images and disregard them. We should transform the images into greyscale and we should standardize the images.

(2) The key part of this problem is to identify the key features of a person's face. We know several feacture extraction mechanims like PCA and Autoencoders. However, I would propose using Convolutional Networks.This will convolution layer, pooling layer and some fully connected layers. We would train feature extraction on the whole dataset. We obtain a dataset with a smaller dimension and then we keep the feature extractor and the last linear layer to jointly train. We will again use non-linear layers because I suspect that the underlying features display non-linearity. Once we have identified the key features, given a new input we can look for these features and identify the face in the image.

The second part of this problem (parts c, d) focuses on exploring possible shortcomings of these models, and what real-world implications might follow from ignoring these issues.

c. *[5 points]* Recall in Homework 2 we trained models to predict crime rates using various features. It is important to note that **datasets describing crime have various shortcomings in describing the entire landscape of illegal behavior in a city, and that these shortcomings often fall disproportionately on minority communities**. Some of these shortcomings include that crimes are reported at different rates in different neighborhoods, that police respond differently to the same crime reported or observed in different neighborhoods, and that police spend more time patrolling in some neighborhoods than others. What real-world implications might follow from ignoring these issues?

Solution:

The shortcomings can have profound real-world implications. If crimes are reported at different rates in different neighborhoods, then the predicted crime rates in those neighborhoods will be higher. This might have negative consequences on people on those neighborhoods. It might lead to more policing in those neighborhoods which might lead to higher levels of incarceration. This could affect the income capabilities of family and directly affect their standard of living. This can potentially lead to vicious cycles where communities with high policy presence have higher incarcerations which lead to higher perceived crime rates. This can lead to further increased police presence and further incarcerations and so on. It also needs to be mentioned that because the shortcomings fall disproportionately on minority communities who may already have poorer living conditions due to problematic policies, more funding may be given to policing due to the perceived higher crime rates. This takes away funding from other communal projects or social institutions that can improve the standard of living of people in these neighborhoods which further worsening the situation. If these issues are not addressed while training the model, the output from the model can cause these issues to become worse and its effect can be profound on already marginalized communities.

---

d. *[5 points]* Pick one of either Scenario 1 or Scenario 2 (in parts a and b). Briefly describe (1) some potential shortcomings of your training process that may result in your algorithm having different accuracy on different populations, and (2) how you may modify your procedure to address these shortcomings.

Solution:

We will pick Scenario 2 for this question. One immediate issue that comes to mind is that we are only dealing with face images. I am guessing that this means that the pictures only contains faces. We might be able to train our network fairly well on these images but in the real world pictures might have a wide range of backgrounds and this model might not generalize well. Instead, we should use a dataset that contains a wide range of images with people in them instead of just portraits. In addition, the fact that we are only using images from the company can be problematic. The sample might not be diverse and the model might overfit. It can also be the case that the company is homogenous in its demographic composition which would mean that it might not do well with other demographics. This is an issue I will expand on below.

Multiple studies have shown that facial recognition does not do well with black faces. In fact, some of the best algorithms identify black faces at rates five to ten times higher than white people. One of the reason could be the overuse of white faces in the training of the model. Now, for a filter program this might not be catastrophic but if for some reason this model was to be used in surveillance then this would be a huge problem. Innocent people might be deemed as criminal due to the model not generalizing well and the burden of this error would again fall disproportionately on people of color. We need to train our model using diverse faces. We should use face images from a random sample from the population instead of training it on face images of employees of the company, celebrities etc.

Basics of SVD

A3. Given $X \in \mathbb{R}^{m\times n}$, recall that its Singular Value Decomposition (SVD) gives us a factorization of a matrix $X = U\Sigma V^\top$ such that $U \in \mathbb{R}^{m\times m}, V^\top \in \mathbb{R}^{n\times n}$ are orthogonal matrices and $\Sigma \in \mathbb{R}^{m\times n}$ is a diagonal matrix representing the singular values of X. Show the following.

a. *[3 points]* Let $\widehat{w}$ be the solution to the regression problem $\min_w \|Xw-y\|_2^2$. Let $\widehat{w}_R$ be the solution to the ridge regression problem $\min_w \|Xw-y\|_2^2 + \lambda\|w\|_2^2$. Let $X = U\Sigma V^\top$ be a singular value decomposition of X. Using this decomposition, explain why the solution $\widehat{w}_R$ to the ridge regression problem "shrinks" as compared to the solution $\widehat{w}$ of the standard regression problem.

Solution: We will provide a mathematical explanation of the difference between regularized and non-regularized versions of linear regression. We wil consider the case of a square Σ to make the math easier.

From our lecture notes, we know that the solution $\widehat{w}$ to the standard regression problem is given by the following:

$$\begin{aligned}
\widehat{w} &= \left(X^\top X\right)^{-1} X^\top y \\
&= \left(V\Sigma U^\top U\Sigma V^\top\right)^{-1}\left(V\Sigma U^\top\right) y \qquad \text{(using the singular value decomposition of } X\text{)} \\
&= \left(V\Sigma^2 V^\top\right)^{-1}\left(V\Sigma U^\top\right) y \\
&= V^{-\top}\Sigma^{-2}V^{-1}V\Sigma U^\top y \\
&= V\Sigma^{-2}V^\top V\Sigma U^\top y \qquad \text{(since } VV^\top = I\text{)} \\
&= V(\Sigma^{-1})U^\top y
\end{aligned}$$

We know that Σ is a diagonal matrix. So,

$$\begin{aligned}
\Sigma^{-1} &= \begin{bmatrix} \frac{1}{\sigma_1} & & \\ & \ddots & \\ & & \frac{1}{\sigma_n} \end{bmatrix} \\
&= \begin{bmatrix} \frac{\sigma_1}{\sigma_1^2} & & \\ & \ddots & \\ & & \frac{\sigma_n}{\sigma_n^2} \end{bmatrix}
\end{aligned}$$

Also, recall that the solution to the ridge regression problem is given by:

$$\begin{aligned}
\widehat{w}_R &= \left(X^\top X + \lambda I\right)^{-1} X^\top y \\
&= \left(V\Sigma^2V^\top + \lambda I\right)^{-1}(V\Sigma U^\top)y \qquad \text{(using the same steps as above)} \\
&= \left(V\Sigma^2V^\top + \lambda VV^\top\right)^{-1}(V\Sigma U^\top)y \\
&= \left(V\left(\Sigma^2+\lambda I\right)V^\top\right)^{-1}\left(V\Sigma U^\top\right)\big) y \\
&= V\left(\Sigma^2+\lambda I\right)^{-1}V^\top V\Sigma U^\top y \\
&= V\left(\left(\Sigma^2+\lambda I\right)^{-1}\Sigma\right)U^\top y
\end{aligned}$$

We can see that the difference lies in the diagonal matrix in middle of the expression. For standard regression problem we had Σ^{-1} but here we have $(\Sigma^2+\lambda I)^{-1}\Sigma$. This will be given by:

$$(\Sigma^2 + \lambda I)^{-1}\Sigma = \begin{bmatrix} \frac{\sigma_1}{\sigma_1^2+\lambda} & & \\ & \ddots & \\ & & \frac{\sigma_n}{\sigma_n^2+\lambda} \end{bmatrix}$$

Therefore, it is easy to see that $\frac{\sigma_i}{\sigma_i^2+\lambda} < \frac{\sigma_i}{\sigma_i^2}$ since $\lambda > 0$ (otherwise for $\lambda = 0$ ridge regression = standard regression). Hence, $\Sigma_{ii}^2 < \left(\left(\Sigma^2 + \lambda I\right)^{-1}\Sigma\right)_{ii}$ and so $\widehat{w}_R < \widehat{w}$ and the solution $\widehat{w}_R$ to the ridge regression problem "shrinks" as compared to the solution $\widehat{w}$ of the standard regression problem.

---

b. *[3 points]* Let $U \in \mathbb{R}^{n\times n}$ be a matrix with singular values all equal to one. Show that $UU^\top = U^\top U = I$.

Solution: Consider the singular value decomposition of U:

$$\begin{aligned} U &= ASB^\top \\ &= AB^\top \qquad \text{(since the singular values are equal to one)} \end{aligned}$$

Then, $U^\top = BA^\top$. Hence:

$$\begin{aligned} UU^\top &= AB^\top BA^\top \\ &= AA^\top \qquad \text{(since } B \text{ is orthogonal)} \\ &= I \qquad \text{(since } A \text{ is orthogonal)} \end{aligned}$$

Similarly,

$$\begin{aligned} U^\top U &= BA^\top AB^\top \\ &= BB^\top \qquad \text{(since } A \text{ is orthogonal)} \\ &= I \qquad \text{(since } B \text{ is orthogonal)} \end{aligned}$$

Therefore, $UU^\top = U^\top U = I$ and so U is orthonormal.

---

c. *[3 points]* Now use the above result to show that U preserves Euclidean norms. In other words, $\|Ux\|_2 = \|x\|_2$ for any $x \in \mathbb{R}^n$.

Solution: Let $x \in \mathbb{R}^n$. Notice: $\|x\|_2^2 = x^\top x$ and similarly $\|Ux\|_2^2 = (Ux)^\top Ux$.

$$\begin{aligned}\|Ux\|_2^2 &= (Ux)^\top Ux \\ &= (x^\top U^\top)Ux \\ &= x^\top (U^\top U)x \\ &= x^T(I)x \\ &= x^T x \\ &= \|x\|_2^2.\end{aligned}$$

We had shown the above part in Homework 0. Now, this implies the following:

$$\begin{aligned}&\sqrt{\|Ux\|_2^2} = \sqrt{\|x\|_2^2} \\ \implies &\|Ux\|_2 = \|x\|_2\end{aligned}$$

Since $x \in \mathbb{R}^n$ was arbitrary we have shown that $\boxed{\|Ux\|_2 = \|x\|_2}$ for any $x \in \mathbb{R}^n$.

---

k-means clustering

A4. Given a dataset $\mathbf{x}_1, ..., \mathbf{x}_n \in \mathbb{R}^d$ and an integer $1 \le k \le n$, recall the following k-means objective function

$$\min_{\pi_1,\ldots,\pi_k} \sum_{i=1}^{k} \sum_{j\in\pi_i} \|\mathbf{x}_j - \mu_i\|_2 \ , \quad \mu_i = \frac{1}{|\pi_i|} \sum_{j\in\pi_i} \mathbf{x}_j \ . \tag{1}$$

Above, $\{\pi_i\}_{i=1}^k$ is a partition of $\{1, 2, ..., n\}$. The objective (1) is NP-hard[1] to find a global minimizer of. Nevertheless the commonly-used algorithm we discussed in lecture (Lloyd's algorithm), typically works well in practice.

a. *[5 points]* Implement Lloyd's algorithm for solving the k-means objective (1). Do not use any off-the-shelf implementations, such as those found in `scikit-learn`. Include your code in your submission.

Solution:

```python
import numpy as np

from utils import problem


@problem.tag("hw4-A")
def calculate_centers(
    data: np.ndarray, classifications: np.ndarray, num_centers: int
) -> np.ndarray:
    """
    Sub-routine of Lloyd's algorithm that calculates the centers given datapoints and
    their respective classifications/assignments.
    num_centers is additionally provided for speed-up purposes.

    Args:
        data (np.ndarray): Array of shape (n, d). Training data set.
        classifications (np.ndarray): Array of shape (n,) full of integers in range {0, 1,
  ..., num_centers - 1}.
            Data point at index i is assigned to classifications[i].
        num_centers (int): Number of centers for reference.
            Might be usefull for pre-allocating numpy array (Faster that appending to list
    ).

    Returns:
        np.ndarray: Array of shape (num_centers, d) containing new centers.
    """
    n, d =data.shape
    centers = np.zeros((num_centers, d))
    for i in range(num_centers):
        data_class = data[np.where(classifications ==i)]
        centers[i] = np.mean(data_class, axis = 0)
    return centers


@problem.tag("hw4-A")
def cluster_data(data: np.ndarray, centers: np.ndarray) -> np.ndarray:
    """
    Sub-routine of Lloyd's algorithm that clusters datapoints to centers given datapoints
    and centers.

    Args:
        data (np.ndarray): Array of shape (n, d). Training data set.
        centers (np.ndarray): Array of shape (k, d). Each row is a center to which a
    datapoint can be clustered.

    Returns:
```

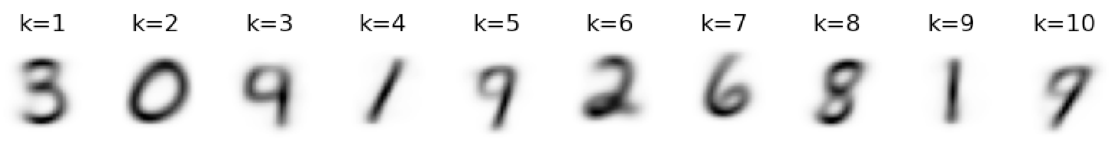
[1]To be more precise, it is both NP-hard in d when $k = 2$ and k when $d = 2$. See the references on the wikipedia page for k-means for more details.

```
        np.ndarray: Array of integers of shape (n,), with each entry being in range {0, 1,
 2, ..., k - 1}.
            Entry j at index i should mean that j^th center is the closest to data[i]
    datapoint.
    """
    n = data.shape[0]
    k = centers.shape[0]
    cluster = np.zeros(n)
    for i in range(n):
        obj = np.zeros(k)
        for j in range(k):
            val = data[i,:] - centers[j,:]
            obj[j] = np.linalg.norm(val)
        cluster[i] = np.argmin(obj)

    return cluster


@problem.tag("hw4-A")
def calculate_error(data: np.ndarray, centers: np.ndarray) -> float:
    """Calculates error/objective function on a provided dataset, with trained centers.

    Args:
        data (np.ndarray): Array of shape (n, d). Dataset to evaluate centers on.
        centers (np.ndarray): Array of shape (k, d). Each row is a center to which a
    datapoint can be clustered.
            These should be trained on training dataset.

    Returns:
        float: Single value representing mean objective function of centers on a provided
    dataset.
    """

    k = centers.shape[0]
    classifications = cluster_data(data, centers)
    err = 0
    n = data.shape[0]
    for i in range(k):
        data_class = data[np.where(classifications ==i)]
        val = data_class-centers[i]
        inner = np.power(np.sum(np.power(val, 2),1), 0.5)
        err += np.sum(inner)
    return err/n
@problem.tag("hw4-A")
def lloyd_algorithm(
    data: np.ndarray, num_centers: int, epsilon: float = 10e-3
) -> np.ndarray:
    """Main part of Lloyd's Algorithm.

    Args:
        data (np.ndarray): Array of shape (n, d). Training data set.
        num_centers (int): Number of centers to train/cluster around.
        epsilon (float, optional): Epsilon for stopping condition.
            Training should stop when max(abs(centers - previous_centers)) is smaller or
    equal to epsilon.
            Defaults to 10e-3.

    Returns:
        np.ndarray: Array of shape (num_centers, d) containing trained centers.

    Note:
        - For initializing centers please use the first `num_centers` data points.
    """
    center = data[0:num_centers]
    diff = 100
    counter = 0
    while diff > epsilon:
        counter += 1
        clusters = cluster_data(data, center)
```

```python
        new_center = calculate_centers(data,clusters,num_centers)
        inner = new_center-center
        diff = np.max(np.abs(inner))
        center = new_center
        print("counter:", counter)
        print("diff:", diff)

    return center
```

b. *[5 points]* Run the algorithm on the *training* dataset of MNIST with $k = 10$. Visualize (and include in your report) the cluster centers as a 28×28 image.

Solution:



c. *[5 points]* For $k = \{2, 4, 8, 16, 32, 64\}$ run the algorithm on the *training* dataset to obtain centers $\{\mu_i\}_{i=1}^k$. If $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ and $\{(\mathbf{x}'_i, y'_i)\}_{i=1}^m$ denote the training and test sets, respectively, plot the training error $\frac{1}{n}\sum_{i=1}^n \min_{j=1,\dots,k} \|\mu_j - \mathbf{x}_i\|_2^2$ and test error $\frac{1}{m}\sum_{i=1}^m \min_{j=1,\dots,k} \|\mu_j - \mathbf{x}'_i\|_2^2$ as a function of k on the same plot.

Solution:

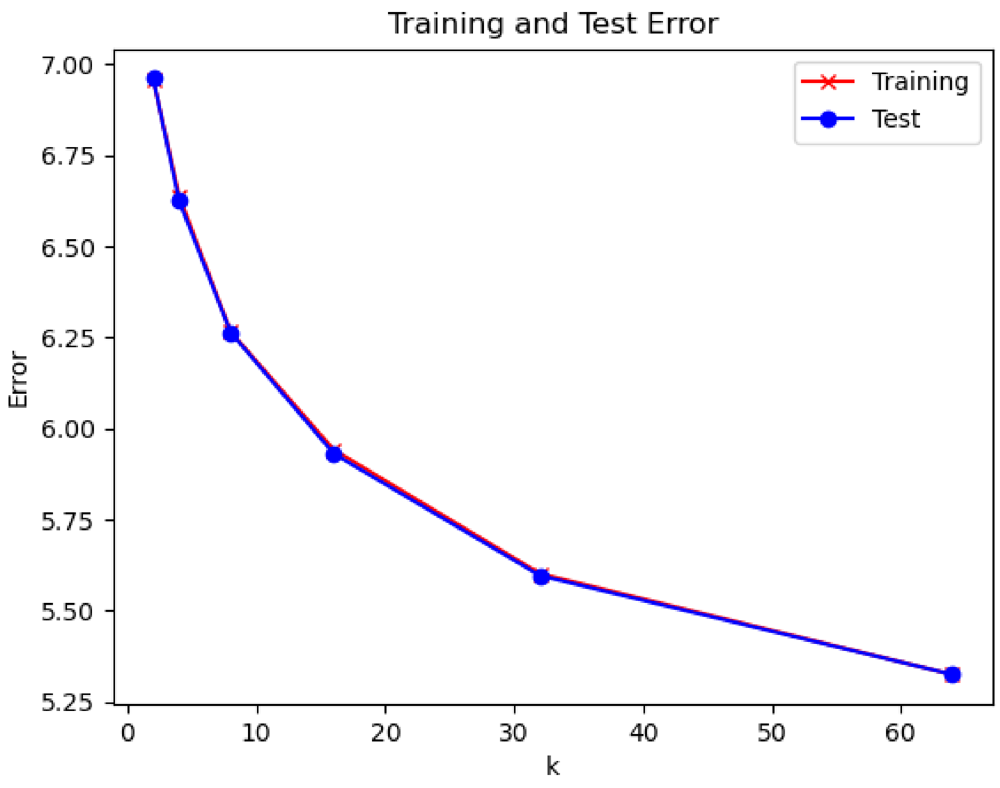


Figure: Training and Test Error as a function of k

PCA

A5. Let's do PCA on MNIST dataset and reconstruct the digits in the dimensionality-reduced PCA basis. You will actually compute your PCA basis using the training dataset only, and evaluate the quality of the basis on the test set, similar to the k-means reconstructions of above. We have $n_{train} = 50,000$ training examples of size 28×28. Begin by flattening each example to a vector to obtain $X_{train} \in \mathbb{R}^{50,000 \times d}$ and $X_{test} \in \mathbb{R}^{10,000 \times d}$ for $d := 784$.

Let $\mu \in \mathbb{R}^d$ denote the average of the training examples in X_{train}, i.e., $\mu = \frac{1}{n_{train}} X_{train}^\top \mathbf{1}^\top$. Now let $\Sigma = (X_{train} - \mathbf{1}\mu^\top)^\top (X_{train} - \mathbf{1}\mu^\top)/50000$ denote the sample covariance matrix of the training examples, and let $\Sigma = UDU^T$ denote the eigenvalue decomposition of Σ.

a. *[2 points]* If λ_i denotes the ith largest eigenvalue of Σ, what are the eigenvalues λ_1, λ_2, λ_{10}, λ_{30}, and λ_{50}? What is the sum of eigenvalues $\sum_{i=1}^{d} \lambda_i$?

Solution: The 1st eigenvalue is: $\boxed{5.11678772834209}$

The 2nd eigenvalue is: $\boxed{3.7413284788648236}$

The 10th eigenvalue is: $\boxed{1.2427293764173373}$

The 30th eigenvalue is: $\boxed{0.364255720278892}$

The 50th eigenvalue is $\boxed{0.16970842700672728}$

The sum of eigenvalues: $\boxed{52.725035495127}$

---

b. *[5 points]* Let $x \in \mathbb{R}^d$ and $k \in 1, 2, \ldots, d$. Write a formula for the rank-k PCA approximation of x.

Solution:

We want to write a formula for the rank-k PCA approximation of x. Let us define $X \in \mathbb{R}^{n \times d}$ to be the stacked data matrix. We can compute the singular value decomposition to get $X = USV^\top$. The objective function of the compressed representation is given by:

$$\min_{V_k} \|(x - \bar{x}) - V_k V_k^\top (x - \bar{x})\|^2$$

where

$$\bar{x} = \mu = \frac{1}{n} \sum_{i=1}^{n} x_i$$

So, the rank-k PCA approximation of x will be given by:

$$\begin{aligned} x - \bar{x} &\approx V_k V_k^\top (x - \bar{x}) \\ \implies \hat{x} &\approx \bar{x} + V_k V_k^\top (x - \bar{x}) \end{aligned}$$

where $V_k \in \mathbb{R}^{d \times k}$ and consists of the k eigenvectors corresponding to the k largest eigenvalues.

c. *[5 points]* Using this approximation, plot the reconstruction error from $k = 1$ to 100 (the X-axis is k and the Y-axis is the mean-squared error reconstruction error) on the training set and the test set (using the μ and the basis learned from the training set). On a separate plot, plot $1 - \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{d} \lambda_i}$ from $k = 1$ to 100.

Solution:

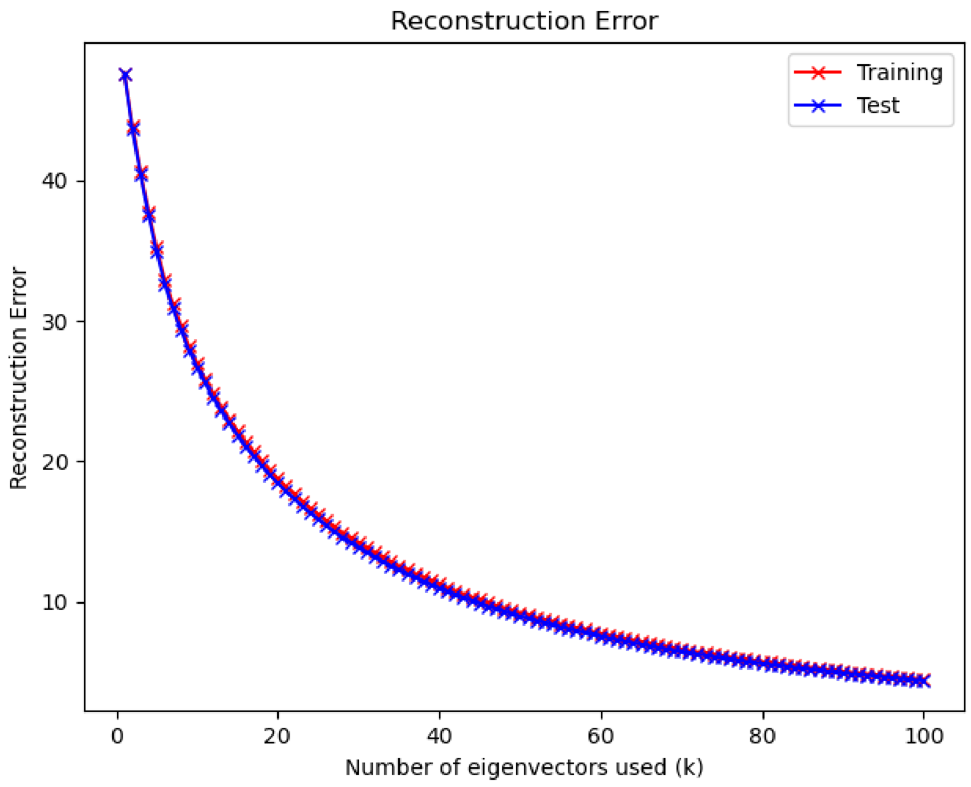


Fig 1: Reconstruction error from $k = 1$ to 100 on the training set and the test set

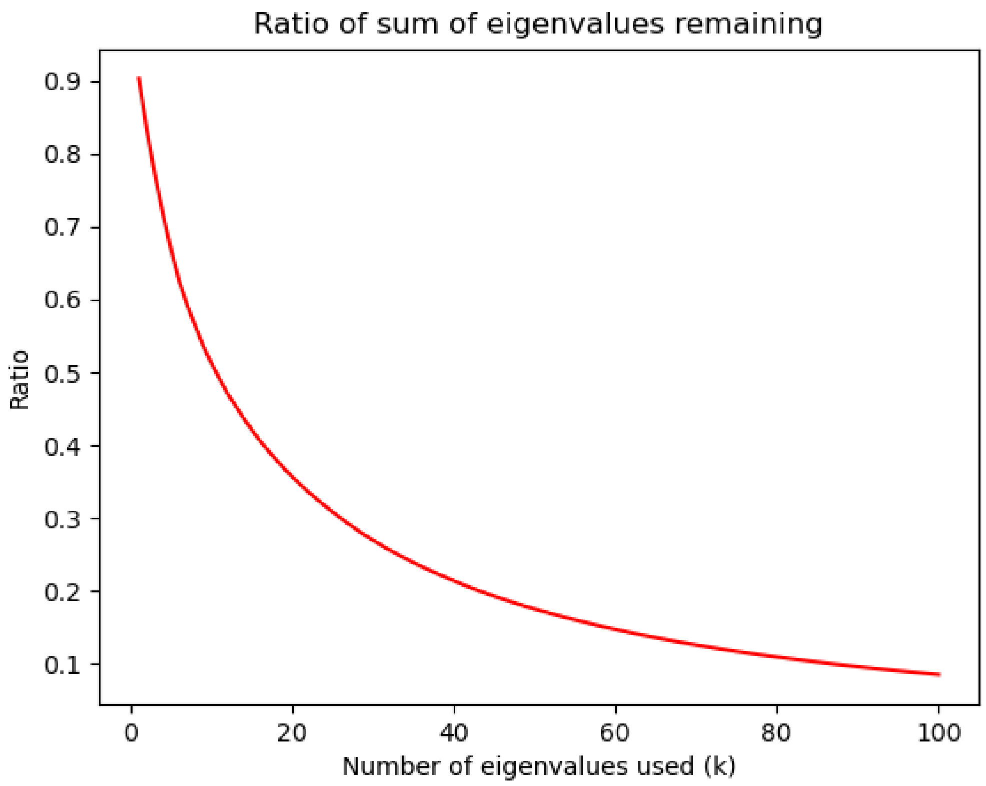


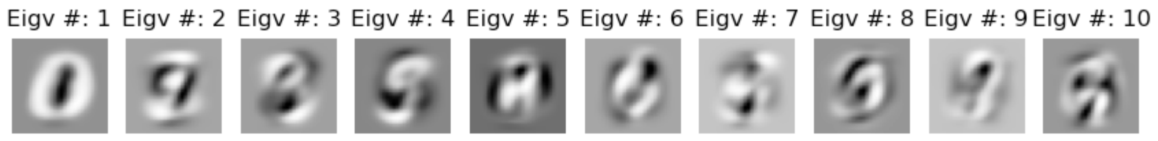
Fig 2: Plot of $1 - \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{d} \lambda_i}$ from $k = 1$ to 100.

---

d. *[3 points]* Now let us get a sense of what the top PCA directions are capturing. Display the first 10 eigenvectors as images, and provide a brief interpretation of what you think they capture.

Solution:

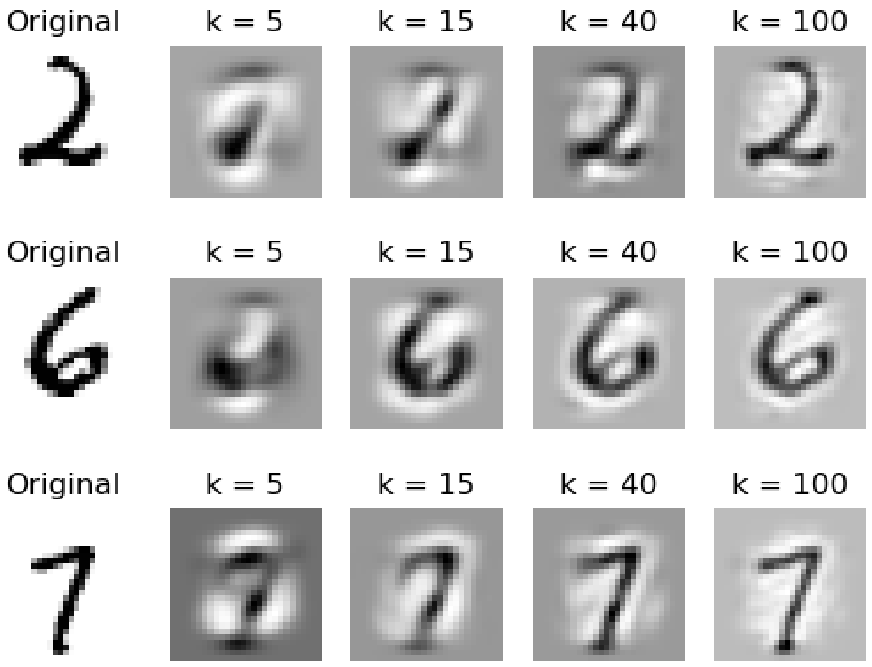
The eigenvectors are trying to identify the key characteristics of the digits. They are trying to capture important features of the digits, that is, what makes 0 a 0 and so on. The first few eigenvectors are capturing the most important features like the shapes of 9 and 3 or the circular shape of 0. We can see that the first few eigenvectors captures most of these features and the other eigenvectors only add small information about the digits. The digit zero is unique in the sense that there are no lines and that is being captured by the first eigenvector. The second eigenvector starts capturing the essence of these lines and we can see that the shapes of 2, 4, 7 and 9 are starting to form. Another unique digit is 3 and we can see that the is the third eigenvector.



e. *[3 points]* Finally, visualize a set of reconstructed digits from the training set for different values of k. In particular provide the reconstructions for digits $2, 6, 7$ with values $k = 5, 15, 40, 100$ (just choose an image from each digit arbitrarily). Show the original image side-by-side with its reconstruction. Provide a brief interpretation, in terms of your perceptions of the quality of these reconstructions and the dimensionality you used.

Solution:

We can notice that $k = 5$ does not do a great job at reconstructing the digits. We cannot even see an outline of the digit. However, with $k = 15$ we can pretty much reconstruct the original image. Higher-rank approximations are closer to the original as expected. As we increase our k, the reconstructed images get better which should make sense. We can also see that the quality of reconstructed images follows sort of a diminishing marginal returns with respect to k. The increase in quality when we move from $k = 5$ to $k = 15$ is much greater than when we move from $k = 15$ to $k = 40$. This is consistent with the plot of ratio of sum of eigenvalues remaining because we can see that the slope of that curve is decreasing. This is consistent with PCA as we except that the first few eigenvectors to capture higher variance in the data. Adding more and more eigenvectors to our approximation does not increase the amount of variance captured by the model that much. In this example, $k = 40$ might be sufficient to reconstruct the digit.



Unsupervised Learning with Autoencoders

A6. In this exercise, we will train two simple autoencoders to perform dimensionality reduction on MNIST. As discussed in lecture, autoencoders are a long-studied neural network architecture comprised of an encoder component to summarize the latent features of input data and a decoder component to try and reconstruct the original data from the latent features.

Weight Initialization and PyTorch

Last assignment, we had you refrain from using `torch.nn` modules. For this assignment, we recommend using `nn.Linear` for your linear layers. You will not need to initialize the weights yourself; the default He/Kaiming uniform initialization in PyTorch will be sufficient for this problem. *Hint: we also recommend using the* ***`nn.Sequential`*** *module to organize your network class and simplify the process of writing the forward pass. However, you may choose to organize your code however you'd like.*

Training

Use `optim.Adam` for this question. Feel free to experiment with different learning rates, though you can use $5 \cdot 10^{-5}$ as mentioned in the code. Use mean squared error (`nn.MSELoss()` or `F.mse_loss()`) for the loss function.

a. *[10 points]* Use a network with a single linear layer. Let $W_{\mathrm{e}} \in \mathbb{R}^{h\times d}$ and $W_{\mathrm{d}} \in \mathbb{R}^{d\times h}$. Given some $x \in \mathbb{R}^d$, the forward pass is formulated as

$$\mathcal{F}_1(x) = W_{\mathrm{d}} W_{\mathrm{e}} x.$$

Run experiments for $h \in \{32, 64, 128\}$. For each of the different h values, report your final training error and visualize a set of 10 reconstructed digits, side-by-side with the original image. *Note:* we omit the bias term in the formulation for notational convenience since `nn.Linear` learns bias parameters alongside weight parameters by default.

Solution:

h	Training Loss
32	0.01728956783960263
64	0.00935686488946279
128	0.004321747784937421

F1 training loss with 40 epochs

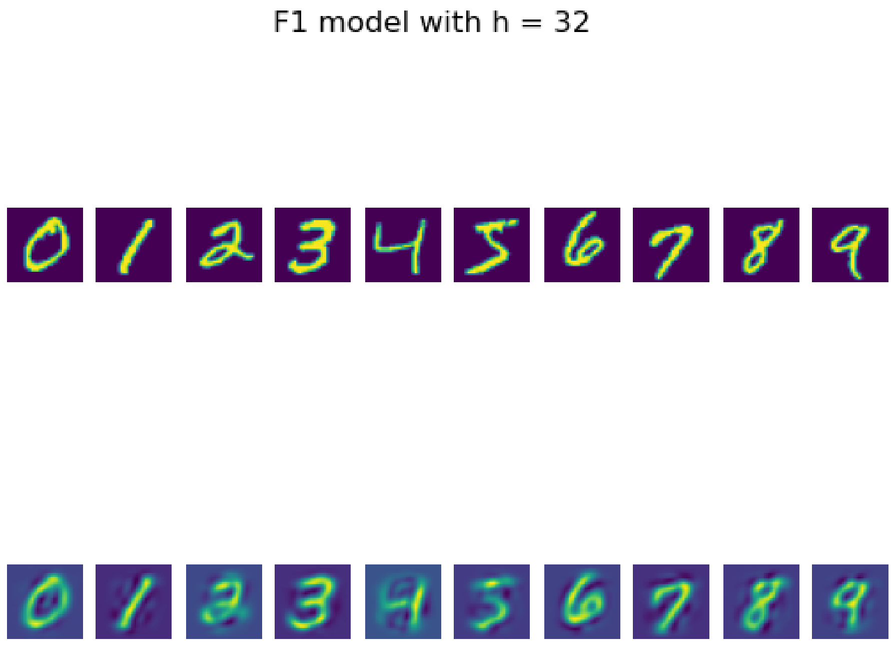


Figure: Visualizing a set of 10 reconstructed digits. The first row contains the original digits and the second row contains the reconstructed digits.

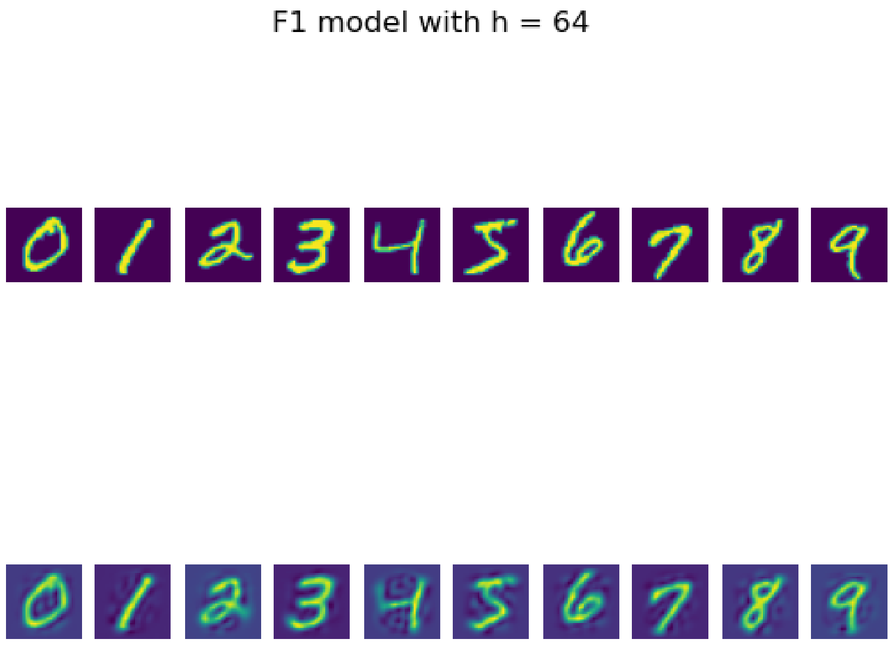


Figure: Visualizing a set of 10 reconstructed digits. The first row contains the original digits and the second row contains the reconstructed digits.

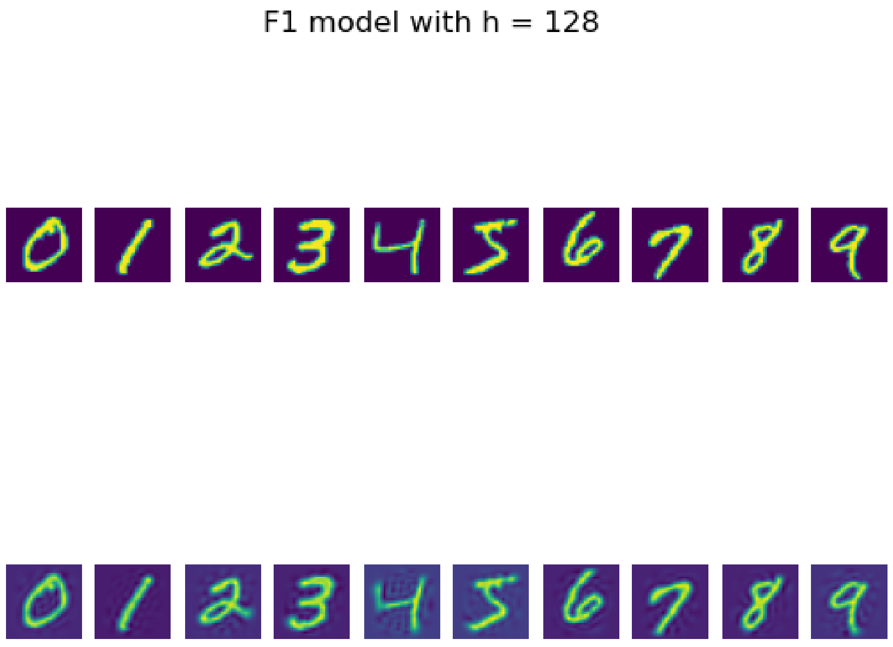


Figure: Visualizing a set of 10 reconstructed digits. The first row contains the original digits and the second row contains the reconstructed digits.

b. *[10 points]* Use a single-layer network with non-linearity. Let $W_{\text{e}} \in \mathbb{R}^{h\times d}$, $W_{\text{d}} \in \mathbb{R}^{d\times h}$, and activation $\sigma : \mathbb{R} \longmapsto \mathbb{R}$, where σ is the ReLU function. Given some $x \in \mathbb{R}^d$, the forward pass is formulated as

$$\mathcal{F}_2(x) = \sigma(W_{\text{d}}\sigma(W_{\text{e}}x)).$$

Report the same findings as asked for in part a (for $h \in \{32, 64, 128\}$).

Solution:

h	Training Error
32	0.016958676773309707
64	0.010729764411350092
128	0.007880173139522472

F2 model with 40 epochs

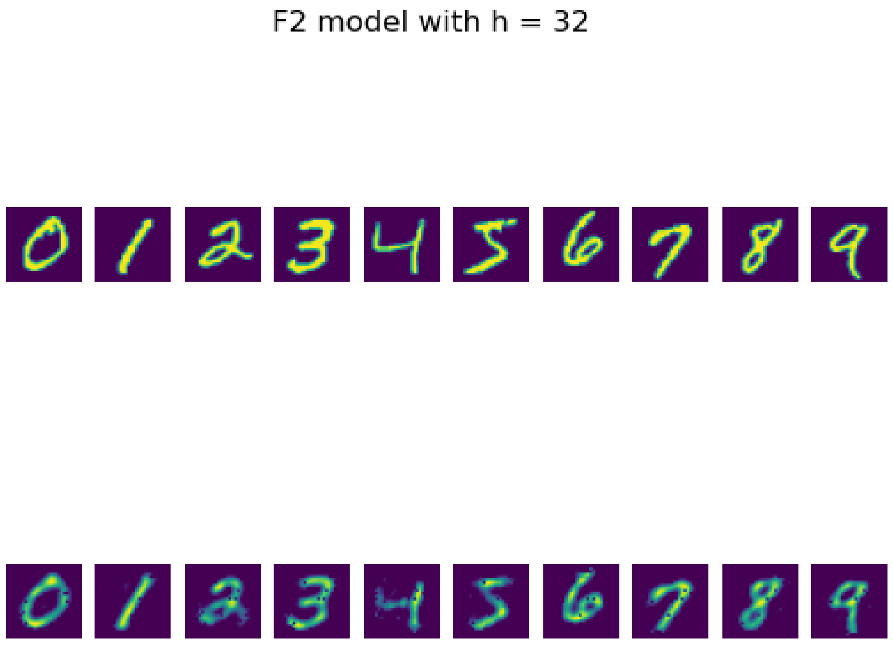


Figure: Visualizing a set of 10 reconstructed digits. The first row contains the original digits and the second row contains the reconstructed digits.

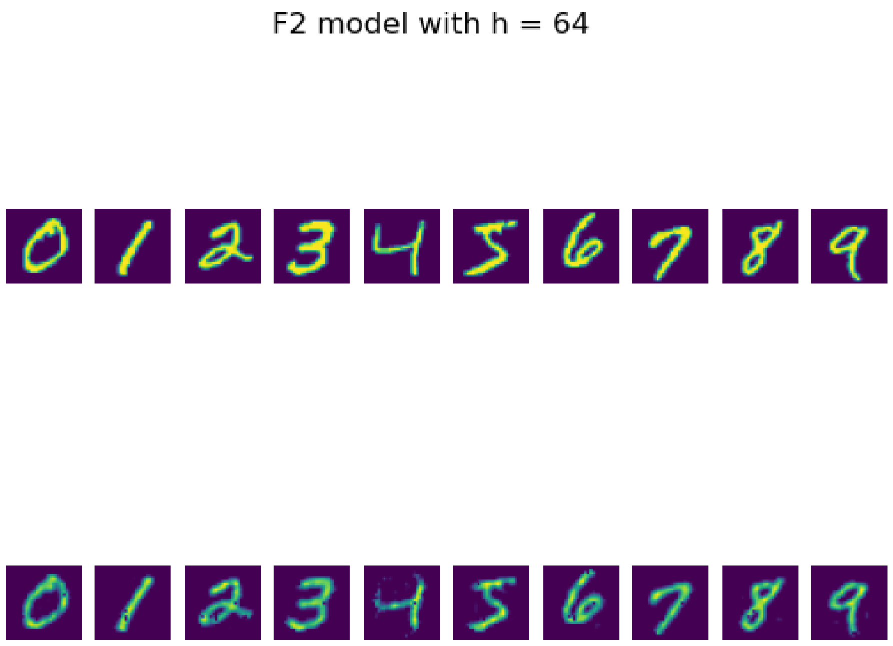


Figure: Visualizing a set of 10 reconstructed digits. The first row contains the original digits and the second row contains the reconstructed digits.

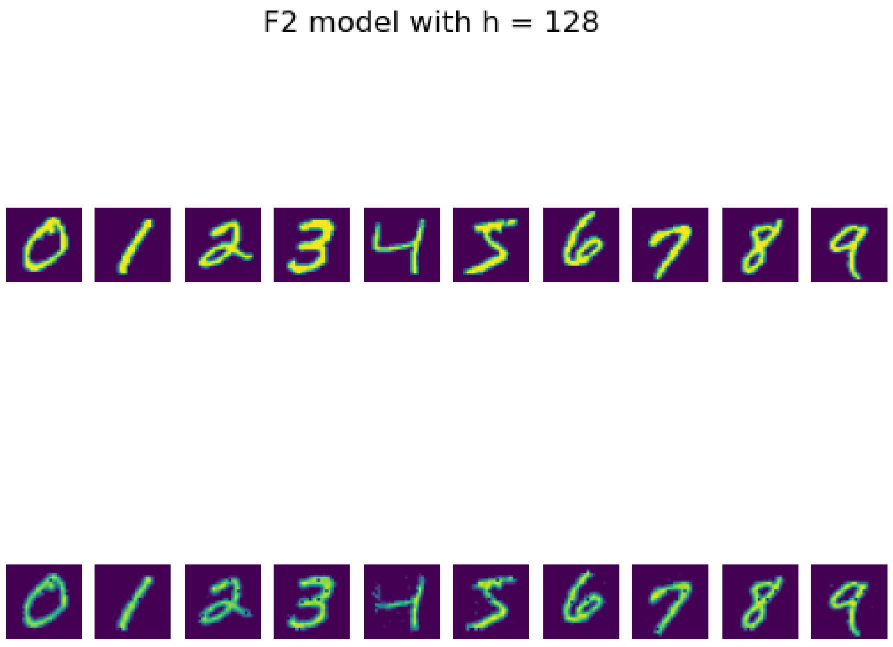


Figure: Visualizing a set of 10 reconstructed digits. The first row contains the original digits and the second row contains the reconstructed digits.

c. *[5 points]* Now, evaluate $\mathcal{F}_1(x)$ and $\mathcal{F}_2(x)$ (use $h = 128$ here) on the test set. Provide the test reconstruction errors in a table.

Solution:

Model	**Test error**
F1 (h=128)	0.004242296765793484
F2 (h=128)	0.007783581755650691

---

d. *[5 points]* In a few sentences, compare the quality of the reconstructions from these two autoencoders with those of PCA from problem A5. You may need to re-run your code for PCA using the ranks $k \in \{32, 64, 128\}$ to match the h values used above.

Solution: Autoencoders with a single layer and without Relu is almost the same as PCA because then autoencoders will be using linear activation functions. The F_2 model will be able to deal with non-linearities. We can expect them to have similar results given same dimensions. However, if we consider $k = 32$ and the digit 1 or 3 we can see that F_1 model clearly outperforms PCA. Even though the test accuracy of F_2 model is lower the reconstructed images are much better than F_1 and consequently better than PCA. This is evident when we look at the digit 4. This patter continues with $k = 64$ and 128 as well. This is probably made most clear when we consider the case of $k = 128$ and the digit 5 where F_2 has close to perfect reconstruction, F_1 has second best reconstruction and PCA is lagging behind considerably. Despite these differences, it is possible to identify the digits with PCA as well so for certian scenarios that might be sufficient and preferable given the fact that it is less computationally expensive. The digits can be multiple shapes which differ from each other and so there might be underlying non-linearities which is why F_2 does a better job at reconstructing the digits even though its test accuracy is lower than F_1.

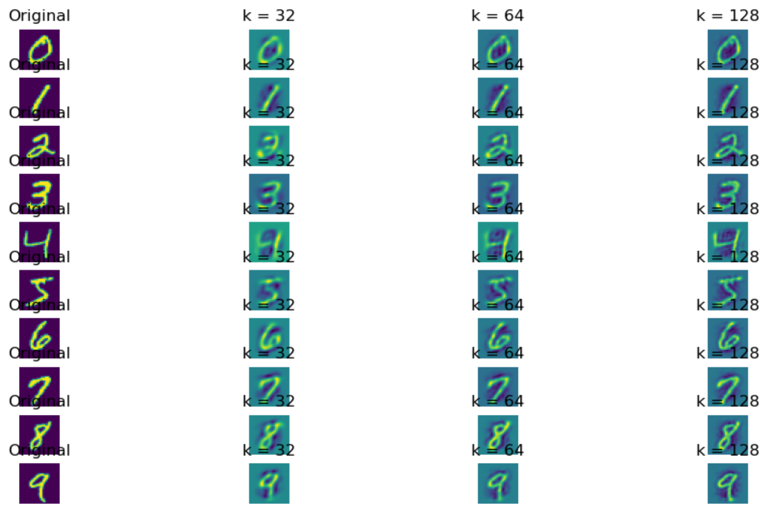


Figure: Reconstructed digits with PCA and $k = 32, 64$ and 128

Administrative

A7.

a. *[2 points]* About how many hours did you spend on this homework? There is no right or wrong answer :)

Solution: 20 hours

Image Classification on CIFAR-10

In this problem we will explore different deep learning architectures for image classification on the CIFAR-10 dataset. Make sure that you are familiar with tensors, two-dimensional convolutions (`nn.Conv2d`) and fully-connected layers (`nn.Linear`), ReLU non-linearities (`F.relu`), pooling (`nn.MaxPool2d`), and tensor reshaping (`view`).

A few preliminaries:

- Each network f maps an image $x^{\text{in}} \in \mathbb{R}^{32\times32\times3}$ (3 channels for RGB) to an output $f(x^{\text{in}}) = x^{\text{out}} \in \mathbb{R}^{10}$. The class label is predicted as $\arg\max_{i=0,1,\ldots,9} x_i^{\text{out}}$. An error occurs if the predicted label differs from the true label for a given image.
- The network is trained via multiclass cross-entropy loss.
- Create a validation dataset by appropriately partitioning the train dataset. *Hint*: look at the documentation for `torch.utils.data.random_split`. Make sure to tune hyperparameters like network architecture and step size on the validation dataset. Do **NOT** validate your hyperparameters on the test dataset.
- At the end of each epoch (one pass over the training data), compute and print the training and validation classification accuracy.
- While one would usually train a network for hundreds of epochs to reach convergence and maximize accuracy, this can be prohibitively time-consuming, so feel free to train for just a dozen or so epochs.

For parts (a)–(c), apply a hyperparameter tuning method (e.g. random search, grid search, etc.) using the validation set, report the hyperparameter configurations you evaluated and the best set of hyperparameters from this set, and plot the training and validation classification accuracy as a function of epochs. Produce a separate line or plot for each hyperparameter configuration evaluated (top 5 configurations is sufficient to keep the plots clean). Finally, evaluate your best set of hyperparameters on the test data and report the test accuracy.
Note: If you are attempting this problem and do not have access to GPU we highly recommend using Google Colab. You can copy this notebook, which will show how to enable/use GPU on that platform.

Here are the network architectures you will construct and compare.

a. *[14 points]* **Fully-connected output, 0 hidden layers (logistic regression):** this network has no hidden layers and linearly maps the input layer to the output layer. This can be written as

$$x^{\text{out}} = W(x^{\text{in}}) + b$$

where $x^{\text{out}} \in \mathbb{R}^{10}$, $x^{\text{in}} \in \mathbb{R}^{32\times32\times3}$, $W \in \mathbb{R}^{10\times3072}$, $b \in \mathbb{R}^{10}$ since $3072 = 32\cdot32\cdot3$. For a tensor $x \in \mathbb{R}^{a\times b\times c}$, we let $(x) \in \mathbb{R}^{abc}$ be the reshaped form of the tensor into a vector (in an arbitrary but consistent pattern). There is no required benchmark validation accuracy for this part.

Solution:

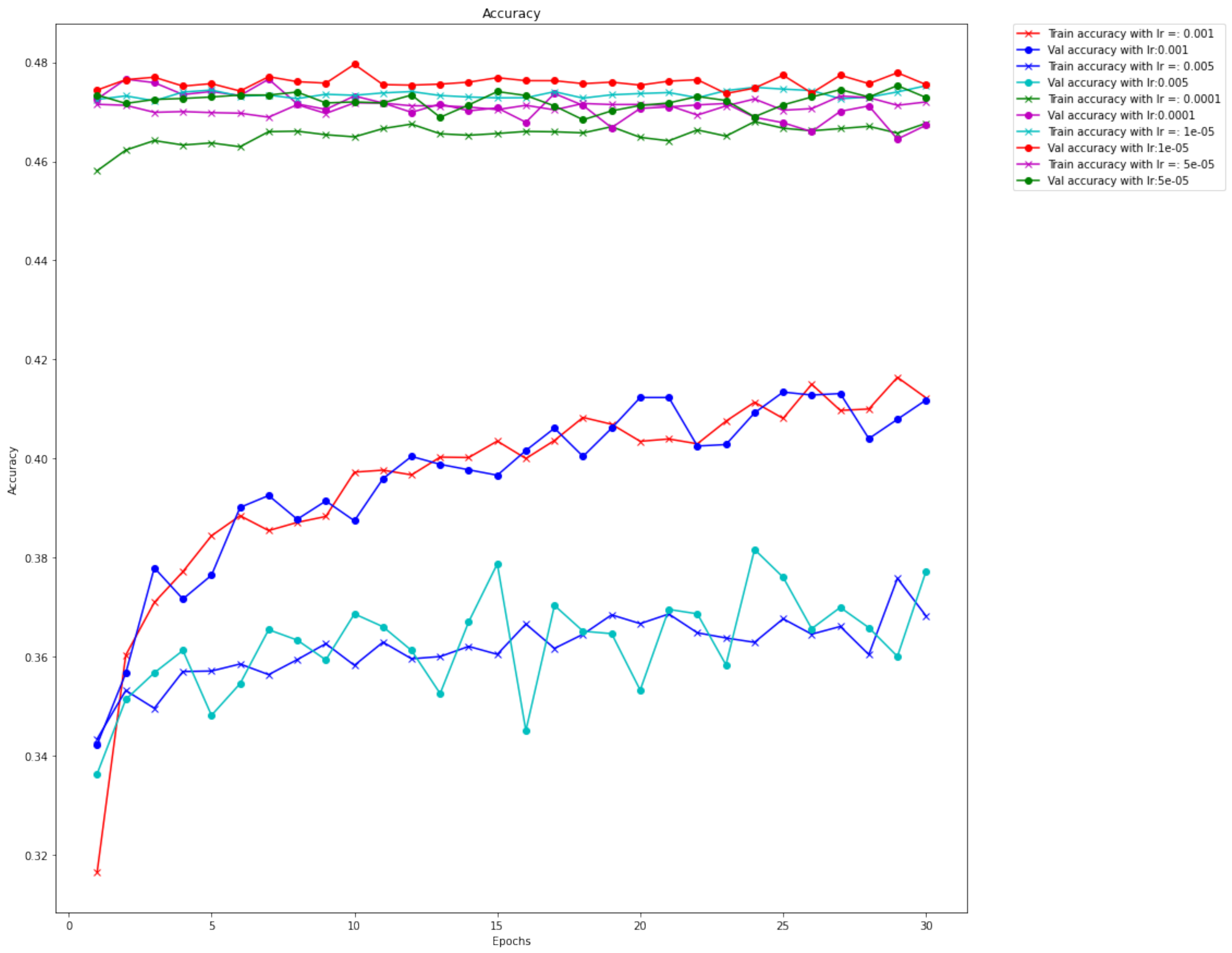


Figure: Training and validation accuracy

The hyperparameter to be tuned was the learning rate. Since, we do not have a required benchmark validation accuracy for this part I chose the best architecture to be the one that achieves the lowest validation loss at any epoch. The best performing hyperparameter was learning rate = $\boxed{1e-05.}$. The accuracy of the best model on test data was $\boxed{0.3948}$. I have also included the graph for test and validation loss below:

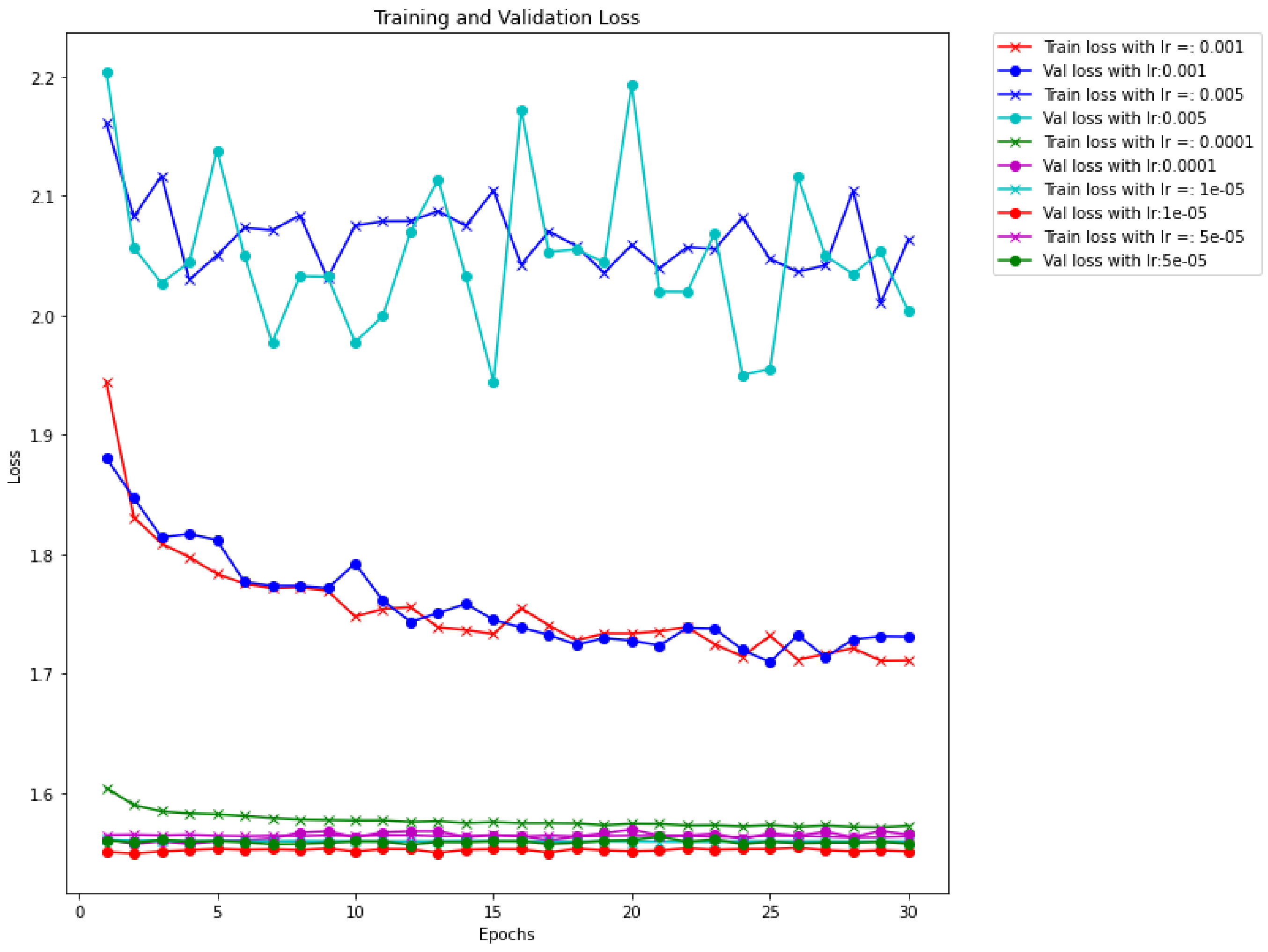


Figure: Training and validation loss

b. *[18 points]* **Fully-connected output, 1 fully-connected hidden layer:** this network has one hidden layer denoted as $x^{\text{hidden}} \in \mathbb{R}^M$ where M will be a hyperparameter you choose (M could be in the hundreds). The nonlinearity applied to the hidden layer will be the `relu` ($\text{relu}(x) = \max\{0, x\}$. This network can be written as

$$x^{out} = W_2\text{relu}(W_1(x^{in}) + b_1) + b_2$$

where $W_1 \in \mathbb{R}^{M\times 3072}$, $b_1 \in \mathbb{R}^M$, $W_2 \in \mathbb{R}^{10\times M}$, $b_2 \in \mathbb{R}^{10}$. Tune the different hyperparameters and train for a sufficient number of epochs to achieve a *validation accuracy* of at least 50%. Provide the hyperparameter configuration used to achieve this performance.

Solution: The hyperparameters to be tuned are the learning rate and the size of the hidden layer. The best hyperparamter configuration we found is **h= 256 with lr =0.001**. We found architecture with validation accuracy at least 50 percent in 16 epochs. It took only 3 hyperparameter configurations to pass performance threshold. The other hyperparamter combinations are: $(lr, h) = (0.001, 64), (0.001, 128)$

Accuracy of best model on test data = $\boxed{0.465}$.

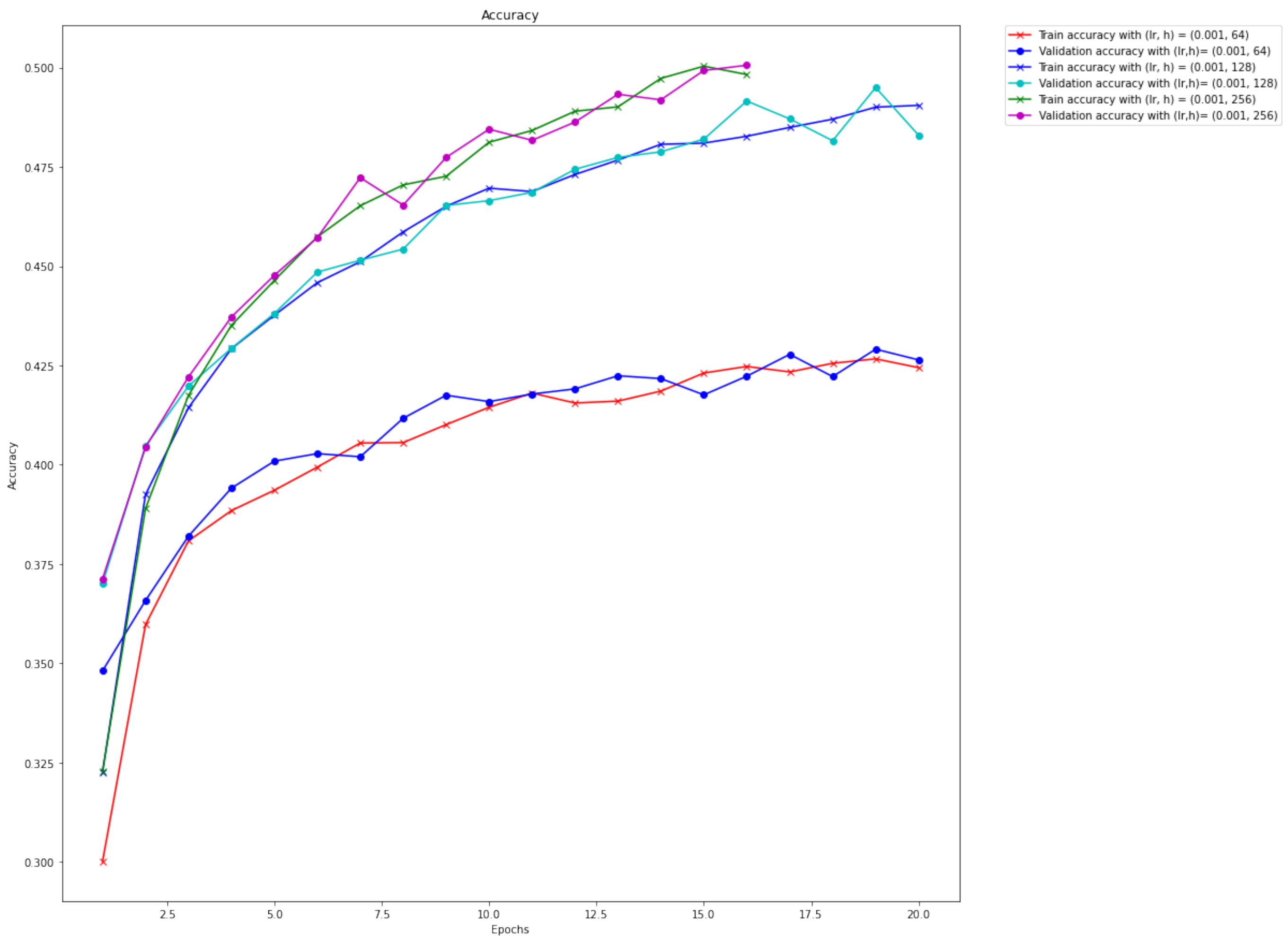


Figure: Training and validation accuracy

---

c. *[18 points]* **Convolutional layer with max-pool and fully-connected output:** for a convolutional layer W_1 with filters of size $k \times k \times 3$, and M filters (reasonable choices are $M = 100$, $k = 5$), we have that $\text{Conv2d}(x^{\text{in}}, W_1) \in \mathbb{R}^{(33-k)\times(33-k)\times M}$.

- Each convolution will have its own offset applied to each of the output pixels of the convolution; we denote this as $\text{Conv2d}(x^{\text{in}}, W) + b_1$ where b_1 is parameterized in $\mathbb{R}^M$. Apply a `relu` activation to the result of the convolutional layer.
- Next, use a max-pool of size $N \times N$ (a reasonable choice is $N = 14$ to pool to 2×2 with $k = 5$) we have that $\text{MaxPool}(\text{relu}(\text{Conv2d}(x^{\text{in}}, W_1) + b_1)) \in \mathbb{R}^{\lfloor \frac{33-k}{N} \rfloor \times \lfloor \frac{33-k}{N} \rfloor \times M}$.
- We will then apply a fully-connected layer to the output to get a final network given as

$$x^{output} = W_2(\text{MaxPool}(\text{relu}(\text{Conv2d}(x^{\text{input}}, W_1) + b_1))) + b_2$$

  where $W_2 \in \mathbb{R}^{10 \times M(\lfloor \frac{33-k}{N} \rfloor)^2}$, $b_2 \in \mathbb{R}^{10}$.

The parameters M, k, N (in addition to the step size and momentum) are all hyperparameters, but you can choose a reasonable value. Tune the different hyperparameters (number of convolutional filters, filter sizes, dimensionality of the fully-connected layers, stepsize, etc.) and train for a sufficient number of epochs to achieve a *validation accuracy* of at least 65%. Provide the hyperparameter configuration used to achieve this performance. Make sure to save this model so that you can do the next part.

The number of hyperparameters to tune, combined with the slow training times, will hopefully give you a taste of how difficult it is to construct networks with good generalization performance. State-of-the-art networks can have dozens of layers, each with their own hyperparameters to tune. Additional hyperparameters you are welcome to play with if you are so inclined, include: changing the activation function, replace max-pool with average-pool, adding more convolutional or fully connected layers, and experimenting with batch normalization or dropout.

Solution: The hyperparameters being tuned were the number of filters M, the learning rate lr and the momentum. We reached desired performance threshold with 3 hyperparemeter combinations. The best hyperparamter configuration is **number of filters = 800, lr =0.001 and momentum = 0.9**. We found this to have validation accuracy at least 65 percent in 27 epochs. The reason we need such a high number of filters is probably because that was the only aspect of the architecture we were tuning.

Accuracy of best model on test data = $\boxed{0.63983}$.

The hyperparameter combinations tried before reaching performance threshold are: [(0.001, 0.9, 200), (0.001, 0.9, 400), (0.001, 0.9, 800)]

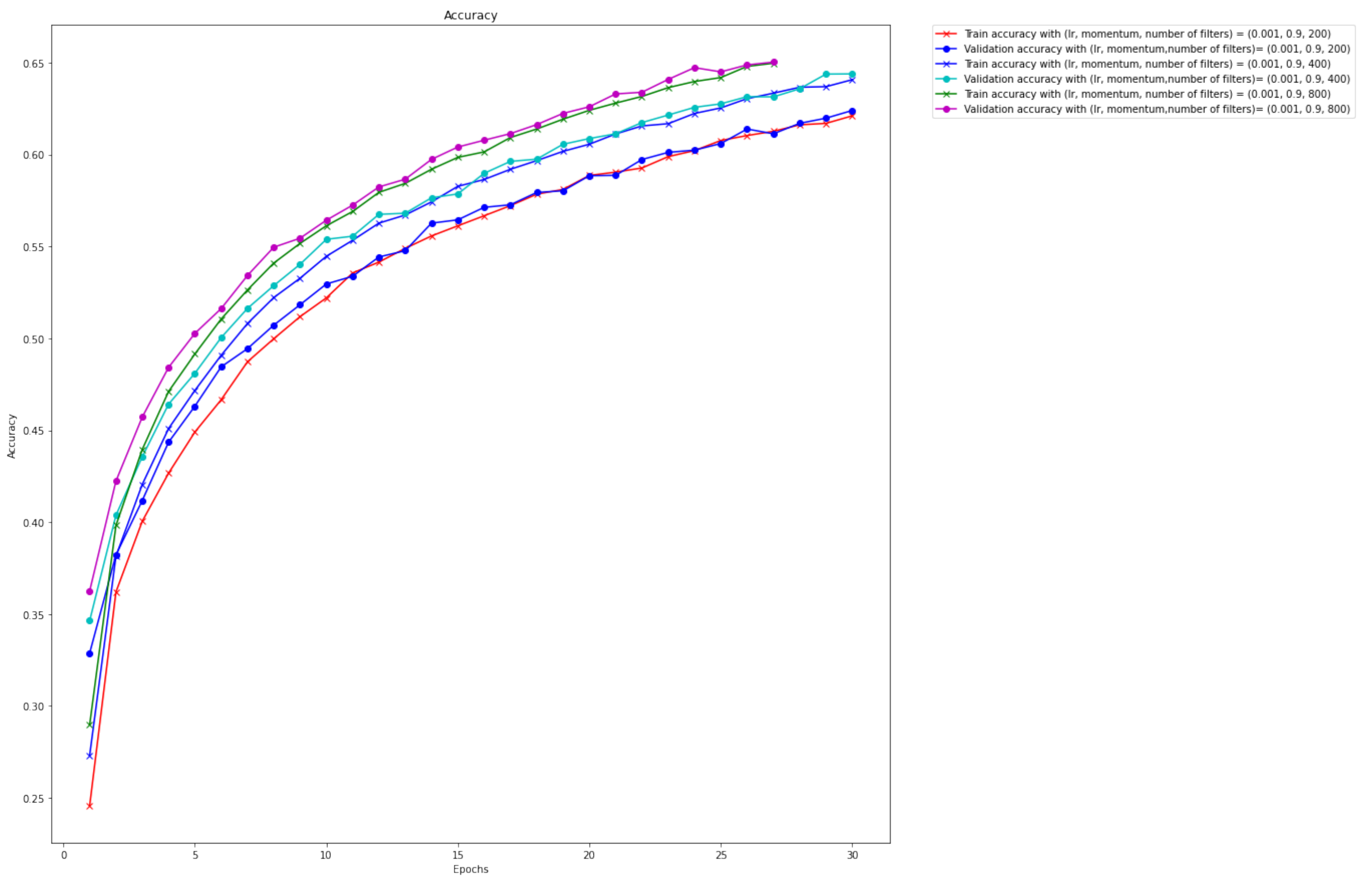


Figure: Training and validation accuracy