CMPT 726 Machine Learning
Spring 2021

Assignment 2

Due: Monday, April 19 at 11:59 pm Pacific Time

This assignment comes with a zip archive containing essential data and starter code that are required to complete the coding component of this assignment.

This assignment is designed to be substantially more challenging than the quizzes and requires thorough understanding of the course material and extensive thought, so **start early!** If you are stuck, come to office hours. Make sure to check the discussion board often for updates, clarifications, corrections and/or hints. Unless otherwise noted, each part of the assignment is weighted equally.

Requests for extensions will not be entertained – make sure you know how to submit your assignment on Canvas and properly account for time difference if you are not in Vancouver. No late submissions will be accepted – all late submissions will receive a mark of zero.

Warning: Copying others' solutions, seeking help from others not in this course or posting questions online are considered cheating. Consequences are severe and could lead to suspension or expulsion. If you become aware of such instances, you must report them here: https://forms.gle/mKgkKbujKtQojXCf6

Submission Instructions:

Carefully follow the instructions below when submitting your assignment.

1. Submit a **separate** PDF for each problem in the assignment to Canvas. You may typeset your assignment in LaTeX or Word (submit in PDF format, **not** .doc/.docx format) or submit **neatly** handwritten and scanned solutions along with screenshots of your code. We **cannot** accept handwritten code in the PDF; screenshots of code must be included even if other parts of your assignment are handwritten. We will not be able to give credit to solutions that are not legible. If there are graphs, include those graphs in the correct sections. **Do not** put them in an appendix. We need each solution to be self-contained on pages of its own.
   - At the start of each problem, please state: (1) who you received help from on the problem, and (2) who you provided help to.
   - At the start of the first problem, please copy the following statement and sign your signature next to it. (Mac Preview and FoxIt PDF Reader, among others, have tools to let you sign a PDF file.) We want to make it *extra* clear so that no one inadvertently cheats. *"I certify that all solutions are entirely in my own words and that I have not looked at another student's solutions. I have given credit to all external sources I consulted."*
2. For all coding-related questions, we recommend working out the solution in the provided Jupyter notebook in Google Colab. Then take a screenshot of your solution and the output

and include it in your PDF. In addition, you must (1) save the Jupyter notebook with your solutions and (2) copy your code into the Python script specific to each part of the question. You need to submit both the completed Jupyter notebook and the Python scripts as a zip archive named "`CMPT726-419_A2_⟨Last Name⟩_⟨Student ID⟩.zip`", whose directory structure should be the same as that of the zip archive for the starter code. Do **NOT** include any data files we provided. Please include a short file named README listing your name and student ID. Please make sure that your code doesn't take up inordinate amounts of time or memory. If your code cannot be executed, your solution cannot be verified.

1 Python Configuration and Data Loading (Optional)

We recommend using Google Colab to complete the parts of this assignment that require coding. However, if you would like to set up your own Python environment on your computer, follow the instructions below.

Please follow the instructions below to ensure your Python environment is configured properly, and you are able to successfully load the data provided with this homework. No solution needs to be submitted for this question. For all coding questions, we recommend using Anaconda for Python 3.

(a) Either install Anaconda for Python 3, or ensure you're using Python 3. To ensure you're running Python 3, open a terminal in your operating system and execute the following command:

```
python --version
```

Do not proceed until you're running Python 3.

(b) Install the following dependencies required for this homework by executing the following command in your operating system's terminal:

```
pip install scikit-learn scipy numpy matplotlib torch torchvision
```

Please use Python 3 with the modules specified above to complete this homework.

To check whether your Python environment is configured properly for this homework, ensure the following Python script executes without error. Pay attention to errors raised when attempting to import any dependencies. Resolve such errors by manually installing the required dependency (e.g. execute `pip install numpy` for import errors relating to the numpy package).

```
import sys
if sys.version_info[0] < 3:
    raise Exception("Python 3 not detected.")

import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
from scipy import io
import torch
```

2 Autoencoder

The autoencoder is a kind of neural network that has two components: (1) an encoder function f that converts inputs $\vec{x}$ to bottleneck feature representation $\vec{z} := f(\vec{x})$, which have lower dimensions than the input and (2) a decoder function g that produces a reconstruction of the inputs as $\vec{y} = g(\vec{z})$ from the bottleneck feature representation $\vec{z}$. The overall neural network is $g \circ f$, which takes an input $\vec{x}$ and produces a reconstruction $\vec{y}$. Mathematically,

$$\vec{y} = g(f(\vec{x}))$$

In the simplest case, both f and g are multi-layer perceptrons (MLPs); in this case, the overall neural network, $g \circ f$, is an MLP as well. The primary difference between a typical MLP and an autoencoder is that an autoencoder is unsupervised, that is, it can be trained without labels. As we will see below, it can be used to generate new data by manipulating the bottleneck feature representation $\vec{z}$. The objective function we use to train the autoencoder is the same as that of multi-output nonlinear regression, namely squared ℓ_2 loss between the reconstruction $\vec{y}$ and the input $\vec{x}$.

In this question we are going to explore different autoencoder architectures. We will work on the MNIST dataset (28×28 images of digits from 0 to 9). In the starter code, we provide the routines for data loading, training loop and visualization in `autoencoder_starter.py`. We also provide a Jupter notebook `autoencoder_sample.ipynb`, where we demonstrate how to train the model and use some essential visualization functions.

(a) Implement an autoencoder where the encoder and the decoder are linear models (i.e. they consist of no hidden layer and one fully-connected / dense output layer (known as a linear layer in PyTorch [1]). The bottleneck feature representation $\vec{z}$ should be two-dimensional. The decoder's output should be transformed by a sigmoid function, so that the output lies within the range $[0, 1]$.

Implement the architecture of the encoder and decoder and print out the reconstruction losses on the train and validation sets. Also use the `scatter_plot` function in `autoencoder_starter.py` to plot the 2D bottleneck feature representation as a scatter plot, where different digit classes are represented by dots of different colours. Include a screenshot of your code that implements the architecture and the generated scatter plot in your PDF submission.

Hints:
(1) Take a look at `Autoencoder_sample.ipynb`, part (a) is partially already implemented, but you will have to find good values for several hyperparameters like the learning rate and the number of epochs to train for.
(2) The training and testing routines are already provided, and they should print out the reconstruction error.
(3) Flatten the image into a one-dimension vector before feeding it into the encoder, and reshape the output of the decoder back into an image as the last step.
(4) When properly trained, the reconstruction loss on the validation set should be around 0.73-0.75.

(b) Starting from the model from part (a), add one fully-connected / dense layer with 1024 hidden units and ReLU activations to both the encoder and the decoder, while keeping the bottleneck feature representation 2-dimensional.

Specifically, the encoder should consist of two layers:

- 1st layer: a fully-connected / dense layer with 1024 hidden units and ReLU activation function. It should map a 784-dimensional input to a 1024-dimensional vector, where

[1] PyTorch documentation: https://pytorch.org/docs/stable/generated/torch.nn.Linear.html

$784 = 28 * 28 * 1$ is the number of dimensions for images in the MNIST dataset.

- 2nd layer: a fully-connected / dense layer, known as a linear layer in PyTorch, with 2 output units and no activation function. It should map a 1024-dimensional vector to a 2-dimensional bottleneck feature vector.

The decoder should have a similar architecture as the encoder, but the layers should be in reverse order.

- 1st layer: a fully-connected / dense layer with 1024 hidden units and ReLU activation function. It should map 2-dimensional bottleneck features to a 1024-dimensional vector.
- 2nd layer: a fully-connected / dense layer with 784 output units and sigmoid activation function. It should map a 1024-dimensional to a 784-dimensional vector, which can be reshaped into $28 \times 28 \times 1$ that is interpreted as an image.

Print out the reconstruction losses on the train and validation sets and generate a scatter plot of the same form as in part (a). Describe how the plot differs from the one in part (a) and explain what this says about the architectures in this part and part (a). Why do you think the architecture in this part gave rise to the results shown in the scatter plot? Include a screenshot of the code that implements the architecture and the generated scatter plot in the PDF submission.

Hint: When properly trained, the reconstruction loss on the validation set should be around 0.65-0.67.

(c) Starting from the model from part (b), widen the bottleneck feature representation from 2 dimensions to 10 dimensions and keep everything else the same.

Print out the reconstruction losses on the train and validation sets. This time, instead of generating a scatter plot, you are required to pick the first 64 images from the validation set and visualize them and the reconstructions by the autoencoder when they are fed in as input in a row using the `display_image_in_a_row` function in `autoencoder_starter.py`. Do the same for the model from part (b) representations and compare the reconstructed images from the 10-dimensional bottleneck to those from the 2-dimensional bottleneck. Describe the difference and explain why you think it happens. Include a screenshot of your code that implements the architecture and the reconstructed images in the PDF submission.

Hint: When properly trained, the reconstruction loss on the validation set should be around 0.51-0.54.

(d) Pick two arbitrary images of different categories (i.e.: different digits) from the validation set, and linearly interpolate in the raw pixel space. More precisely, if we denote two images as $\vec{x}_1$ and $\vec{x}_2$, the linear interpolation $\widehat{\vec{x}}_{(t)}$ in raw pixel space would be

$$\widehat{\vec{x}}_{(t)} = t\vec{x}_1 + (1-t)\vec{x}_2$$

where $t \in [0, 1]$. **Show the interpolated images $\widehat{\vec{x}}_{(t)}$ for $t \in \{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$ (11 images in total) using the `display_images_in_a_row` function in `autoencoder_starter.py`.**

Include a screenshot of your code for generating linear interpolations and the interpolated images themselves in the PDF submission.

(e) Using the same pair of images $\vec{x}_1$ and $\vec{x}_2$ that you picked in part (d), and linearly interpolate between their bottleneck feature representations, $\vec{z}_1 = f(\vec{x}_1)$ and $\vec{z}_2 = f(\vec{x}_2)$, this time rather than between the raw pixels (this is known as interpolating in the bottleneck feature space). More precisely, if f and g denote the encoder and decoder respectively, the interpolation $\widehat{\vec{x}}_{(t)}$ between $\vec{x}_1$ and $\vec{x}_2$ in bottleneck feature space would be

$$\widehat{\vec{x}}_{(t)} = g(t\vec{z}_1 + (1-t)\vec{z}_2) = g(tf(\vec{x}_1) + (1-t)f(\vec{x}_2))$$

where $t \in [0, 1]$. You are required to generate interpolations using two different models, the model from part (b) with 2-dimensional bottleneck representations and the model from part (c) with 10-dimensional bottleneck representations.

For the model from part (b) and the model from part (c), show the interpolated images $\widehat{\vec{x}}_{(t)}$ for $t \in \{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$ (11 images in total) using the `display_images_in_a_row` function in `autoencoder_starter.py`. How are these interpolations in bottleneck feature space different from the interpolations in raw pixel space from part (d)? Explain why you think this happens. How are the interpolations in bottleneck feature space for the model from part (c) different from those for the model from part (b)? Include a screenshot of your code for generating interpolations and the interpolated images themselves in the PDF submission.

(f) Starting from the model from part (c), further widen the bottleneck feature representation from 10 dimensions to 64 dimensions and keep everything else the same. **Print out the reconstruction losses on the train and validation sets, and pick the first 64 images from the validation set and visualize the reconstructed images in a row using the `display_image_in_a_row` function in `autoencoder_starter.py`.** Additionally, as in part (e), show interpolations in the bottleneck feature space between the same pair of images you picked in part (d). **Specifically, show the interpolated images $\widehat{\vec{x}}_{(t)}$ for $t \in \{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$ (11 images in total) using the `display_images_in_a_row` function.** Compare the interpolated images with the ones in part (e) and **describe the differences you find and explain why you think the differences arise. Include a screenshot of your code that implements the architecture and the visualizations of the reconstructed and interpolated images in the PDF submission.**

Hint: When properly trained, the reconstruction loss on the validation set should be around 0.47-0.50.

3 SVMs for Novelty Detection

In class we have studied support vector machines (SVMs), which are binary classifiers. In this problem we will focus on a variant of an SVM designed for use when only data from a single class is available, known as a one-class SVM.

You might be wondering: what does it mean to classify a single class? Consider the following example – suppose you only like songs on Spotify and never dislike any song, and would like Spotify to suggest songs that you may like. This cannot be achieved using binary classification, because there are no data points in the negative class.

The goal of one-class SVM is to predict whether a new test data point belongs to the class or not – this problem is known as outlier detection, novelty detection or anomaly detection; outliers or anomalies are examples that do not fit in with the rest of the data. So, a one-class SVM assumes all/most of data points in the training set are normal and at test time predicts if the new test data point is normal (i.e.: it is a member of the class represented by the training data) or abnormal (outliers/anomalies).

(a) We are given a training set of data points $\{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n\}$, which all belong to one class, the normal class. One-class SVM tries to find a hyperplane that passes through the origin such that all training data points are on the positive side of it. Among all such hyperplanes, it chooses the hyperplane that maximizes the margin, that is the distance between the hyperplane and the closest data points. This hyperplane becomes the decision boundary of the one-class SVM.

Consider a training set of 2D data points shown in Fig. 1. **On the plot shown in Fig. 1, draw the decision boundary of a one-class SVM trained on this dataset as a solid line, and a hyperplane parallel to it that passes through the data points closest to the decision boundary as a dashed line, and the margin as a double-ended arrow whose length should be the same as the margin. Justify your answer.**

We recommend using PowerPoint to draw on the plot. You may find `SVM_Diagram.ppt` included in the starter code zip archive) helpful.

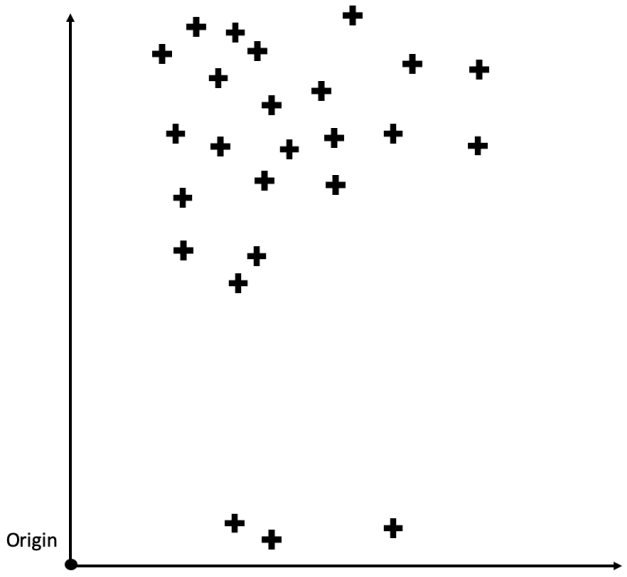

Figure 1: Training Dataset

(b) The expression of the decision boundary is

$$H := \{\vec{x} \in \mathbb{R}^d | \vec{w}^\intercal \vec{x} = 0\}, \tag{1}$$

where $\vec{w}$ is a vector perpendicular to the decision boundary.

Derive the expression of the distance from an arbitrary data point to the decision boundary from first principles, that is, without using the distance to the decision boundary formula for the classical two-class SVM. To this end, you need to consider the length of the projection of a vector onto another vector. **Then use the expression you derived to show that the margin in a one-class SVM can be expressed as:**

$$m = \min_i \frac{|\vec{w}^\intercal \vec{x}_i|}{||\vec{w}||_2} \tag{2}$$

In your derivation, show each step and justify all important steps (i.e.: steps that do not just involve simple arithmetic or algebraic manipulations).

(c) Formulate a constrained optimization problem that corresponds to the objectives of one-class SVM. In particular, your formulation should include a variable m for the margin, and achieve the following: (1) maximize the margin, (2) ensure the margin is non-negative, (3) ensure that the distance from all training data points to the decision boundary is no less than the margin, and (4) ensure all training data points lie on the positive side of the hyperplane that forms the decision boundary.

Write down the constrained optimization problem you devised, and explain the meanings of all new variables that are not given in the problem and the meanings of the objective and all constraints.

(d) It turns out that the optimization problem that one-class SVM solves can be written in the following way:

$$\begin{aligned} \widehat{\vec{w}} = \arg\min_{\vec{w}} \quad & \frac{1}{2}||\vec{w}||_2^2 \\ \text{subject to} \quad & \vec{w}^\intercal \vec{x}_i \geq 1, \quad \forall 1 \leq i \leq n \end{aligned} \tag{3}$$

From the formulation you derived in part (c), show that it is equivalent to the optimization problem above. In your derivation, show each step and briefly explain how you got from one step to the next step.

(e) There are datasets where one-class SVM cannot find a decision boundary that satisfies all constraints. **Draw an example of such a dataset on the plot in Fig. 2 and explain why there is no decision boundary that satisfies all constraints on this dataset.**

We recommend using PowerPoint to draw on the plot. You may find `SVM_Diagram.ppt` (included in the starter code zip archive) helpful.

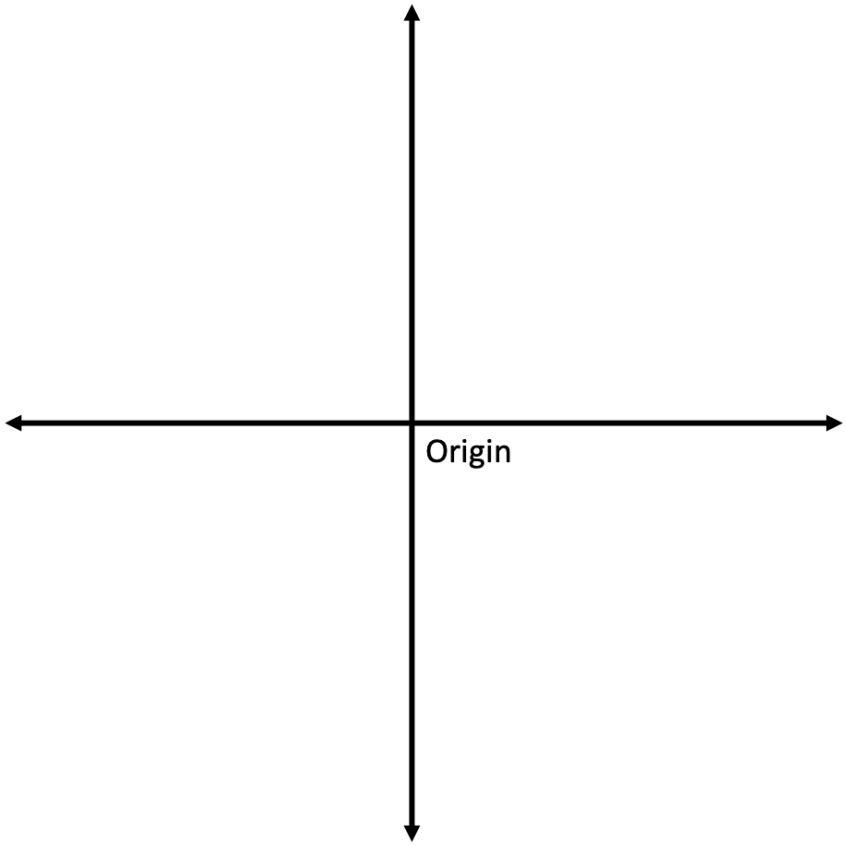

Figure 2: Blank Axes

(f) To eliminate the possibility that no decision boundary can be found, we can formulate a soft one-class SVM, which replaces each constraint in the hard one-class SVM problem (defined by Eq. (3) in part (d)) with a hinge loss term in the objective:

$$\max(0, 1 - \vec{w}^\top \vec{x}_i) \tag{4}$$

So, the optimization problem becomes unconstrained and has the following objective:

$$\widehat{\vec{w}} = \arg\min_{\vec{w}} \frac{1}{2}\|\vec{w}\|_2^2 + C\sum_{i=1}^{n} \max(0, 1 - \vec{w}^\top \vec{x}_i). \tag{5}$$

Let $\widehat{\vec{w}}^\top \vec{x}$ be the score of training data point $\vec{x}$, where $\widehat{\vec{w}}$ is the optimal solution for $\vec{w}$ found by the soft one-class SVM (5) with some particular value for C. We will classify a data point as an outlier if its score is below a threshold η. **Describe a procedure to find a threshold η so that about 7% of your training data will be classified as outliers.**

(g) The optimal $\widehat{\vec{w}}$ in the hard one-class SVM optimization problem defined by Eq. (3) in part (d) is identical to the optimal $\widehat{\vec{w}}_{\text{two-class}}$ in the traditional two-class hard-margin SVM you saw in class using the augmented training data $(\vec{x}_1, 1)$, $(\vec{x}_2, 1)$, …, $(\vec{x}_n, 1)$, $(-\vec{x}_1, -1)$, $(-\vec{x}_2, -1)$, …, $(-\vec{x}_n, -1)$.

Argue why this is true by comparing the objective functions and constraints of the two optimization problems, as well as the optimization variables.

(h) The Lagrangian dual of the soft one-class SVM optimization problem is:

$$\widehat{\vec{\alpha}} = \arg\max_{\vec{\alpha}} \quad \vec{\alpha}^\top \vec{1} - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j \vec{x}_i^\top \vec{x}_j \qquad (6)$$
$$0 \le \alpha_i \le C, \quad 1 \le i \le n.$$

Consider the data points $\vec{x}_i$ with non-zero dual variables α_i. On the dataset shown in Fig. 1 in part (a), find all data points whose dual variables could be non-zero and circle them in the plot you included in part (a). Explain why you chose these data points.

