CPSC 340 Assignment 6

Important: Submission Format [5 points]

Please make sure to follow the submission instructions posted on the course website. We will deduct marks if the submission format is incorrect, or if you're not using LaTeX and your submission is *at all* difficult to read – at least these 5 points, more for egregious issues. Compared to assignment 1, your name and student number are no longer necessary (though it's not a bad idea to include them just in case, especially if you're doing the assignment with a partner).

1 Robust PCA for Background Subtraction

If you run `python main.py 1`, the program will load a dataset X where each row contains the pixels from a single frame of a video of a highway. The demo applies PCA to this dataset and then uses this to reconstruct the original image. It then shows the following 3 images for each frame:

1. The original frame.

2. The reconstruction based on PCA.

3. A binary image showing locations where the reconstruction error is non-trivial.

Recently, latent-factor models have been proposed as a strategy for "background subtraction": trying to separate objects from their background. In this case, the background is the highway and the objects are the cars on the highway.

Why does this make any sense? Remember that PCA tries to find principal components that reconstruct the dataset well. Here the training examples are highway images. Now, we need to reconstruct these highway images out of a small number of PC images. Since we want to reconstruct them well, we're going to create PCs that capture the *common features across the entire dataset.* In other words, the PCs will try to reconstruct the highway and slight variations of the highway, because that is always there. If a car appears in the top-left of only a couple images, we might not want a PC for that case, since that PC is useless for the majority of the training set. Thus, when we try to reconstruct the images we are left to see only the parts of the image that can be made out of the PCs, or in other words that are common across all the images. Hence why the reconstructions look like the background. We can then subtract this reconstruction (background) from the original image to get objects of interest (foreground). Cool, right?

In this demo, we see that PCA does an OK job of identifying the cars on the highway in that it does tend to identify the locations of cars. However, the results aren't great as it identifies quite a few irrelevant parts of the image as objects. Who likes good news — everyone? Well good news, everyone! We can use robust PCA to do even better.

Robust PCA is a variation on PCA where we replace the L2-norm with the L1-norm,

$$f(Z, W) = \sum_{i=1}^{n} \sum_{j=1}^{d} |\langle w^j, z_i \rangle - x_{ij}|,$$

and it has recently been proposed as a more effective model for background subtraction.

1.1 Why Robust PCA [5 points]

In a few sentences, explain why using a robust loss might help PCA perform better for this background subtraction application. Some conversation-starters: what is an outlier here — a car or the highway? What does it mean to be robust to outliers — that we're good at reconstructing them or that we're back at reconstructing them?

Answer: Robust PCA is more capable in background subtraction compared to conventional PCA because it is less sensitive to outliers. In this problem, the highway is the constant since it appears in every frame of the video while cars are outliers since they only appear in a subset of frames int the video. If we use conventional PCA, which relies on squared loss, any outliers will be amplified. When cars appear in some frames, they cause large reconstruction errors that distort the principal components, leading to blurry reconstructions in regions where cars are present. However robust PCA uses L1-regularization which computes absolute error which is more lenient on outliers. Cars in frames are treated as sparse anomalies, reducing their impact on the reconstruction. This leads to clearer background reconstructions with less blur in the areas where cars appear.

1.2   Robust PCA Approximation and Gradient [5 points]

If you run `python main.py 1.3`, the program will repeat the same procedure as in the above section, but will attempt to use the robust PCA method, whose objective functions are yet to be implemented. You will need to implement it (yay!).

We'll use a gradient-based approach to PCA and a smooth approximation to the L1-norm. In this case the log-sum-exp approximation to the absolute value may be hard to get working due to numerical issues. Perhaps the Huber loss would work. We'll use the "multi-quadric" approximation:

$$|\alpha| \approx \sqrt{\alpha^2 + \epsilon},$$

where ϵ controls the accuracy of the approximation (a typical value of ϵ is 0.0001). Note that when $\epsilon = 0$ we revert back to the absolute value, but when $\epsilon > 0$ the function becomes smooth.

Our smoothed loss is:

$$f(Z, W) = \sum_{i=1}^{n} \sum_{j=1}^{d} \sqrt{(\langle w^j, z_i \rangle - x_{ij})^2 + \epsilon}$$

The partial derivatives of this loss with respect to the elements of W are:

$$\frac{\partial}{\partial w_{cj}} f(Z, W) = \frac{\partial}{\partial w_{cj}} \sum_{i=1}^{n} \sum_{j'=1}^{d} \left((\langle w^{j'}, z_i \rangle - x_{ij'})^2 + \epsilon \right)^{\frac{1}{2}}$$

$$= \sum_{i=1}^{n} \frac{\partial}{\partial w_{cj}} \left((\langle w^j, z_i \rangle - x_{ij})^2 + \epsilon \right)^{\frac{1}{2}} \qquad \text{(since the } j' \neq j \text{ terms have no } w_{cj} \text{ in them)}$$

$$= \sum_{i=1}^{n} \frac{1}{2} \left((\langle w^j, z_i \rangle - x_{ij})^2 + \epsilon \right)^{-\frac{1}{2}} \frac{\partial}{\partial w_{cj}} \left((\langle w^j, z_i \rangle - x_{ij})^2 + \epsilon \right)$$

$$= \sum_{i=1}^{n} \frac{1}{2} \left((\langle w^j, z_i \rangle - x_{ij})^2 + \epsilon \right)^{-\frac{1}{2}} 2 \left(\langle w^j, z_i \rangle - x_{ij} \right) \frac{\partial}{\partial w_{cj}} \langle w^j, z_i \rangle$$

$$= \sum_{i=1}^{n} \left((\langle w^j, z_i \rangle - x_{ij})^2 + \epsilon \right)^{-\frac{1}{2}} \left(\langle w^j, z_i \rangle - x_{ij} \right) z_{ic}$$

The partial derivatives with respect to Z are similar:

$$\frac{\partial}{\partial z_{ic}} f(Z, W) = \frac{\partial}{\partial z_{ic}} \sum_{i'=1}^{n} \sum_{j=1}^{d} \left((\langle w^j, z_{i'} \rangle - x_{i'j})^2 + \epsilon \right)^{\frac{1}{2}}$$

$$= \sum_{j=1}^{d} \left((\langle w^j, z_i \rangle - x_{ij})^2 + \epsilon \right)^{-\frac{1}{2}} \left(\langle w^j, z_i \rangle - x_{ij} \right) \frac{\partial}{\partial z_{ic}} \langle w^j, z_i \rangle$$

$$= \sum_{j=1}^{d} \left((\langle w^j, z_i \rangle - x_{ij})^2 + \epsilon \right)^{-\frac{1}{2}} \left(\langle w^j, z_i \rangle - x_{ij} \right) w_{cj}$$

If we put this into matrix(ish) notation, we get the following:

$$\nabla_W f(Z, W) = Z^T \left[R \oslash \left(R^{\circ 2} + \epsilon \right)^{\circ \frac{1}{2}} \right]$$

where $R \equiv ZW - X$, $A \oslash B$ denotes **element-wise** division of A and B, $A + s$ for a scalar s denotes element-wise adding s to each entry of A, and $A^{\circ p}$ denotes taking A to the **element-wise** power of p.

And, similarly, the gradient with respect to Z is given by:

$$\nabla_Z f(Z, W) = \left[R \oslash \left(R^{\circ 2} + \epsilon \right)^{\circ \frac{1}{2}} \right] W^T$$

Show that the two parts of the gradient above, $\nabla_W f(Z, W)$ and $\nabla_Z f(Z, W)$, have the expected dimensions.

Answer:

For $\nabla_W f(Z, W)$:

We expect $\nabla_W f(Z, W)$ to have dimensions of W, which is k x d, because we are finding the partial derivative of $f(Z, W)$ with respect to each element within W. Z^T has dimensions k x n, R has dimensions n x d, which makes $R \oslash \left(R^{\circ 2} + \epsilon \right)^{\circ \frac{1}{2}}$ also have dimension n x d. So $\nabla_W f(Z, W)$ has dimensions k x d.

For $\nabla_Z f(Z, W)$:

We expect $\nabla_Z f(Z, W)$ to have dimensions of Z, which is n x k, because we are finding the partial derivative of $f(Z, W)$ with respect to each element within Z. R has dimensions n x d, which makes $R \oslash \left(R^{\circ 2} + \epsilon \right)^{\circ \frac{1}{2}}$ also have dimension n x d, W^T has dimensions d x k. So $\nabla_Z f(Z, W)$ has dimensions n x k.

1.3 Robust PCA Implementation [10 points]

In `fun_obj.py`, you will find classes named `RobustPCAFactorsLoss` and `RobustPCAFeaturesLoss` which should compute the gradients with respect to W and Z, respectively. Complete the `evaluate()` method for each using the smooth approximation and gradient given above. Submit (1) your code in `fun_obj.py`, and (2) one of the resulting figures.

Hint: Your code will look similar to the already implemented `PCAFactorsLoss` and `PCAFeaturesLoss` classes. Note that the arguments for `evaluate()` are carefully ordered and shaped to be compatible with our optimizers.

Note: The robust PCA is somewhat sensitive to initialization, so multiple runs might be required to get a reasonable result.

Answer:

```python
class RobustPCAFeaturesLoss(FunObj):
    def __init__(self, epsilon):
        self.epsilon = epsilon

    def evaluate(self, z, W, X):
        n, _ = X.shape
        k, _ = W.shape
        Z = z.reshape(n, k)

        R = (Z @ W) - X
        R_squared = np.sqrt(np.square(R) + self.epsilon)
        R_div_R_squared = R / R_squared
        f = np.sum(R_squared)
        g = R_div_R_squared @ W.T
        return f, g.flatten()

class RobustPCAFactorsLoss(FunObj):
    def __init__(self, epsilon):
        self.epsilon = epsilon

    def evaluate(self, w, Z, X):
        _, d = X.shape
        _, k = Z.shape
        W = w.reshape(k, d)

        R = (Z @ W) - X
        R_squared = np.sqrt(np.square(R) + self.epsilon)
        R_div_R_squared = R / R_squared
        f = np.sum(R_squared)
        g = Z.T @ R_div_R_squared
        return f, g.flatten()
```
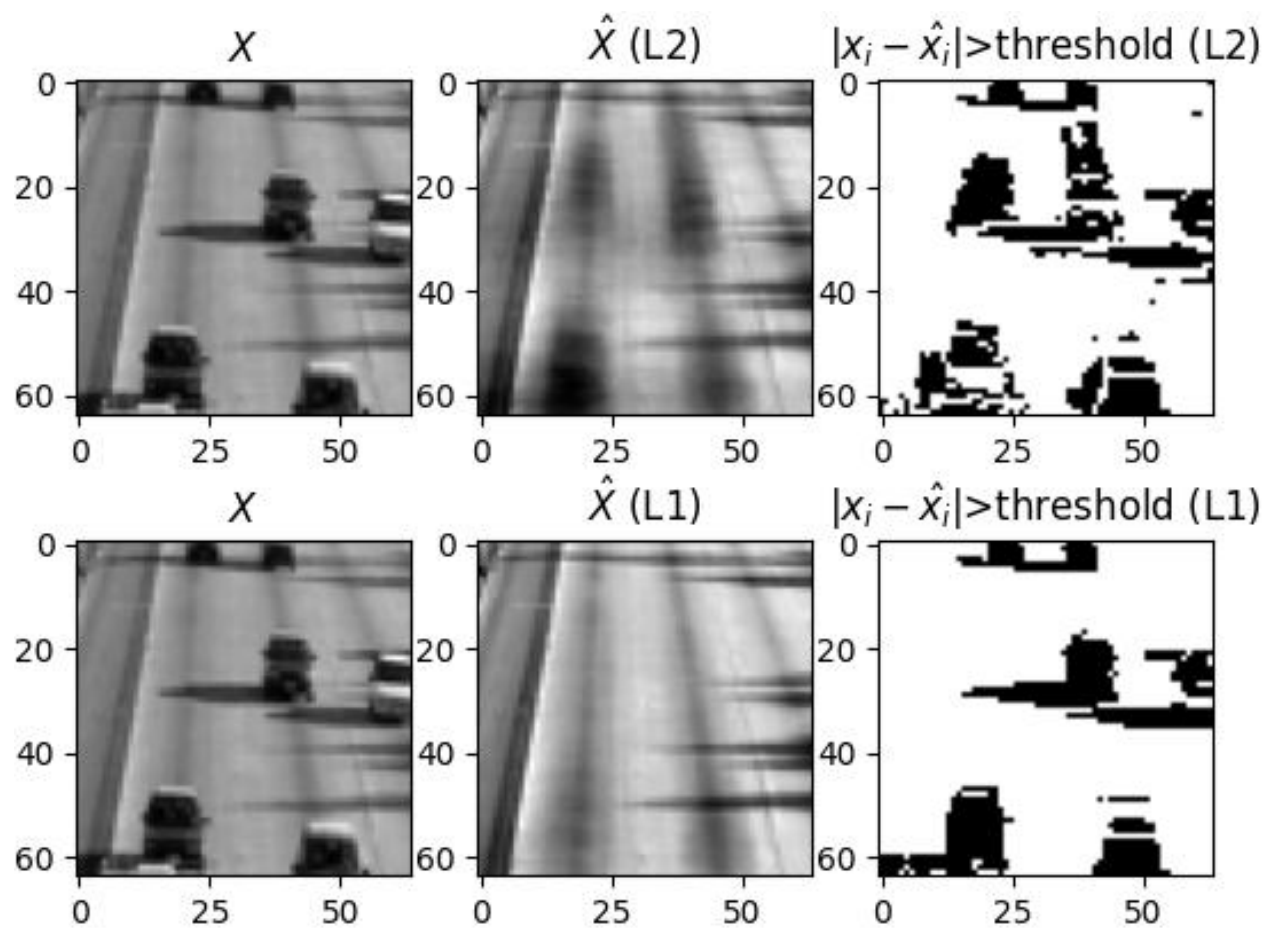
1.4 Reflection [6 points]

1. Very briefly comment on the results from the previous section — does robust PCA seem to do better than regular PCA for this task?

   Answer: Yes, robust PCA produces a more clear picture of the highway compared to conventional PCA.

2. How does the number of video frames and the size of each frame relate to n, d, and/or k?

   Answer: n is the number of frames in the video. d is the number of pixels in each picture. k is the number of principle components which is defined by the programmer.

3. What would the effect be of changing the threshold (see code) in terms of false positives (cars we identify that aren't really there) and false negatives (real cars that we fail to identify)?

   Answer: A higher threshold means we are less sensitive to outliers (cars), and will have higher false negatives, and a lower threshold will have more false positives (and noise).

2 Movie Recommendations

If you run `python main.py 2`, the program will perform the following steps:

1. Loads the small educational version of the MovieLens dataset (`https://grouplens.org/datasets/movielens/`).

2. Prints out the first 5 rows of the ratings dataframe (which we'll use) and the movies dataframe (which we won't use, but is loaded FYI).

3. Transforms the ratings table into the Y matrix described in lecture.

4. Splits Y into train (80% of the ratings) and validation (20% of the ratings) matrices.

5. Prints out some stats about the dataset, including the average rating in the training set.

2.1 Understanding Y [6 points]

Answer the following questions:

1. In lecture, we used the "?" symbol to represent missing ratings. How are these missing entries of Y implemented in the code?

   Answer: The value "NaN" is used to represent missing ratings.

2. How many (non-missing) ratings are there in `Y_train` and `Y_valid`, respectively?

   Answer:
   Number of non missing ratings are there for Y_train = 80668
   Number of non missing ratings ratings are there for Y_train = 20168

3. Does the same user-movie rating ever appear in *both* `Y_train` and `Y_valid`? Is the result as expected?

   Answer: The same user-movie rating should not appear as training and validation data sets should not contain the same examples, following the golden rule of machine learning. However, it is possible that two users, one in training and one in validation, have the same rating for the same movie. This is expected, and are not considered "identical" in the case of violating the golden rule, since they have different IDs.

2.2 Implementing Collaborative Filtering [15 points]

If you run `python main.py 2.2`, the code will fit a model that doesn't actually do anything. It uses the same alternating minimization scheme as in the previous question on Robust PCA. Fill in the `evaluate` methods of the `CollaborativeFilteringWLoss` and `CollaborativeFilteringZLoss` classes in `fun_obj.py`. Submit your code.

Hint: Since we're minimizing the regularized PCA loss, your code should be quite similar to the `evaluate` methods in `PCAFeaturesLoss` and `PCAFactorsLoss`. I suggest you copy/paste those methods as a starting point. However, there are two changes you'll need to make: (1) modify the loss and gradient to account for the regularization, and (2) accounting for the fact that the loss should only sum over the available ratings, not the entire Y matrix. The easiest way to account for this is to modify the R matrix by setting all its NaN values to 0. Since R represents the residuals (or reconstruction errors), setting these to 0 says that these entries don't contribute to the loss, which is exactly what we want.

```python
class CollaborativeFilteringZLoss(FunObj):
    def __init__(self, lammyZ=1, lammyW=1):
        self.lammyZ = lammyZ
        self.lammyW = lammyW

    def evaluate(self, z, W, Y):
        n, d = Y.shape
        k, _ = W.shape
        Z = z.reshape(n, k)

        R = Z @ W - Y
        R[np.isnan(Y)] = 0
        f = np.sum(R ** 2) + self.lammyZ * np.sum(Z**2)/2 + self.lammyW * np.sum(W**2)/2
        G = 2 * R @ W.T + self.lammyZ * Z
        return f, G.flatten()


class CollaborativeFilteringWLoss(FunObj):
    def __init__(self, lammyZ=1, lammyW=1):
        self.lammyZ = lammyZ
        self.lammyW = lammyW

    def evaluate(self, w, Z, Y):
        n, d = Y.shape
        _, k = Z.shape
        W = w.reshape(k, d)

        R = Z @ W - Y
        R[np.isnan(Y)] = 0
        f = np.sum(R ** 2) + self.lammyZ * np.sum(Z**2)/2 + self.lammyW * np.sum(W**2)/2

        G = 2 * Z.T @ R + self.lammyW * W
        return f, G.flatten()
```

2.3 Hyperparameter tuning [5 points]

If you run `python main.py 2.3`, the program will implement a baseline model that predicts the average rating (around 3.5 stars) for all missing ratings. As you'll see from the output, this baseline has a root mean squared error (RMSE) of slightly above 1 star. The hyperparameters we gave you in the code actually do worse than this (my validation RMSE is around 1.3 stars). Adjust the hyperparameters of your collaborative filtering model, namely k, λ_W, λ_Z, until you obtain a better validation error than the baseline. Submit the hyperparameters you found as well as your training and validation RMSE.

PS: there is nothing to stop you from just guessing random hyperparameters, but it would be cool if you think about the fact that the current model overfits and adjust each of the hyperparameters in the direction that reduces the complexity of the model to combat the overfitting.

Answer: After 4 hours of performing grid search, $5 <= k <= 50$ and $0 <= lambda <= 10$ for both lammys, these are the best hyper parameters we could find:

$k = 6$

$\lambda_W = 9$

$\lambda_Z = 8$

The parameters gave a training RMSE of 0.66 and a validation RMSE of 0.92.

Note: We obtain good results too at the neighbourhood of k=180, but felt that is too high of number of principal components.

2.4   Regularization with PCA [5 points]

In lecture we discussed the fact that with regularized PCA, we need to regularize both W and Z for things to make sense. Test this out empirically by setting $\lambda_Z = 0$ and $\lambda_W > 0$. Modify the code so that it prints out the Frobenius norm of W and Z after each iteration of the alternating minimization (you don't have to submit this printing code though). Describe your observations and briefly discuss.

Answer: When $\lambda_Z = 0$ and $\lambda_W > 0$, the Frobenius norm W matrix decreases with each iteration while the Frobenius norm of Z will remain constant. Alternatively, if $\lambda_Z > 0$ and $\lambda_W = 0$ then the Frobenius norm of Z matrix will decrease with each iteration while the Frobenius norm of W will remain constant. λ_Z and λ_W, controls the regularization of matrix Z and W. If they are set to zero, the matrices are not penalized for their magnitude which would result in the Frobenius norm of the matrix staying constant.

3 Neural Networks

3.1 Neural Networks by Hand [7 points]

Suppose that we train a neural network with sigmoid activations and one hidden layer and obtain the following parameters (assume that we don't use any bias variables):

$$W = \begin{bmatrix} -2 & -1 & 3 \\ 1 & 0 & -2 \end{bmatrix}, v = \begin{bmatrix} 4 \\ 2 \end{bmatrix}.$$

Assuming that we are doing regression, for a training example with features $x_i^T = \begin{bmatrix} 4 & -2 & 2 \end{bmatrix}$ what are the values in this network of z_i, $h(z_i)$, and $\hat{y}_i$?

Answer:

$$z_i = Wx_i = \begin{bmatrix} -2 & -1 & 3 \\ 1 & 0 & -2 \end{bmatrix} \begin{bmatrix} 4 \\ -2 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Since the sigmoid of 0 is 0.5,

$$h(z_i) = h\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}\right) = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

$$\hat{y}_i = v^T h(z_i) = \begin{bmatrix} 4 & 2 \end{bmatrix} \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} = 3$$

3.2 Neural Networks vs. Linear Classifier [7 points]

If you run `python main.py 3`, the program will train a multi-class logistic regression classifier on the MNIST handwritten digits data set using stochastic gradient descent with some pre-tuned hyperparameters. The scikit-learn implementation called by the code uses a minibatch size of 1 and a one-vs-rest strategy for multi-class logistic regression. It is set to train for 10 epochs. The performance of this model is already quite good, with around 8% training and validation errors. Your task is to use a neural networks classifier to outperform this baseline model.

If you run `python main.py 3.2`, the program will train a single-layer neural network (one-layer nonlinear encoder paired with a linear classifier) on the same dataset using some pre-tuned hyperparameters. Modify the code, play around with the hyperparameter values (configurations of encoder layers, batch size and learning rate of SGD, standardization, etc.) until you achieve validation error that is reliably below 5%. Report (1) the training and validation errors that you obtain with your hyperparameters, and (2) the hyperparameters that you used.

Answer: Ran `q3_2` 10 times and obtained an average training error of 0.0418 and average validation error of 0.049 was obtained when using the following parameters:

1. standardize `X_train` and `X_valid` with `standardize_cols`
2. `hidden_feature_dims` is set to 50
3. `output_dim` is set to 10
4. learning rate uses inverse square root and is set to 0.1
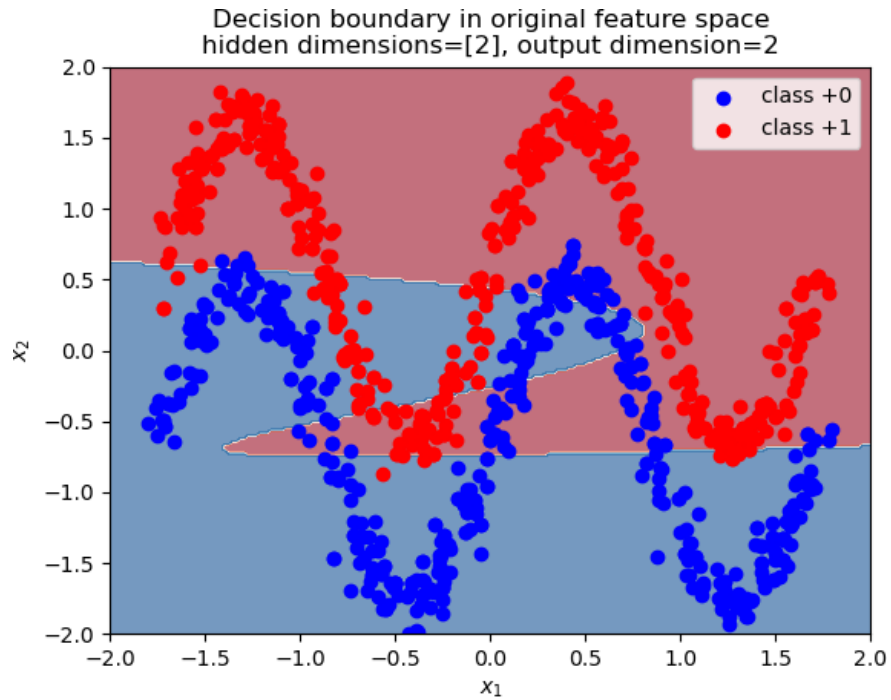5. batch size set to 700
6. max evaluations is set to 20

3.3   Neural Disentangling [10 points]

An intuitive way to think about a neural network is to think of it as a composition of an encoder (all the layers except the last one, including the final nonlinearity) and a linear predictor (the last layer). The predictor signals the encoder to learn a better feature space, in such a way that the final linear predictions are meaningful.

The following figures resulted from running `python main.py 3.3`. For this, a neural network model is used to encode the examples into a 2-dimensional feature space. Here's the original training data:


Sinusoid data, non-convex but separable.
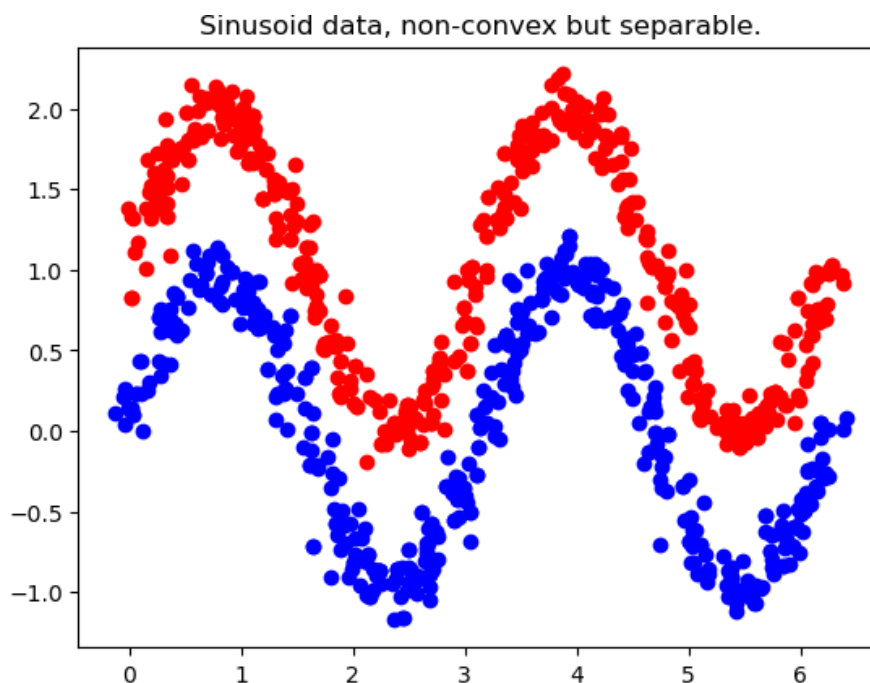
And here is the learned decision boundary of the neural network:

Decision boundary in original feature space
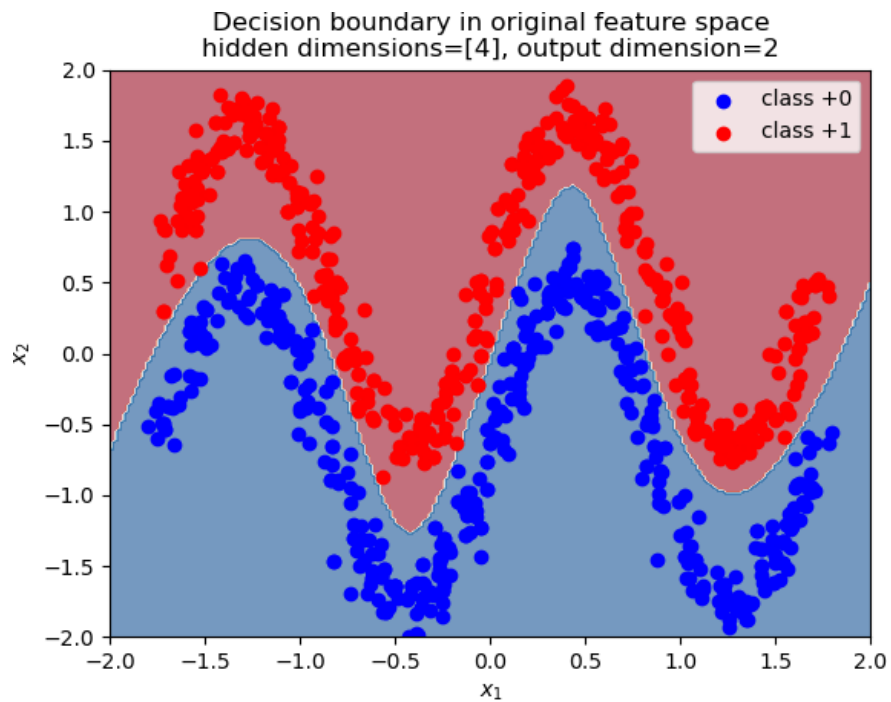hidden dimensions=[2], output dimension=2

We can look inside and see what the encoder learned. Here is the training data shown in the 2D transformed feature space learned by the encoder (first layer):
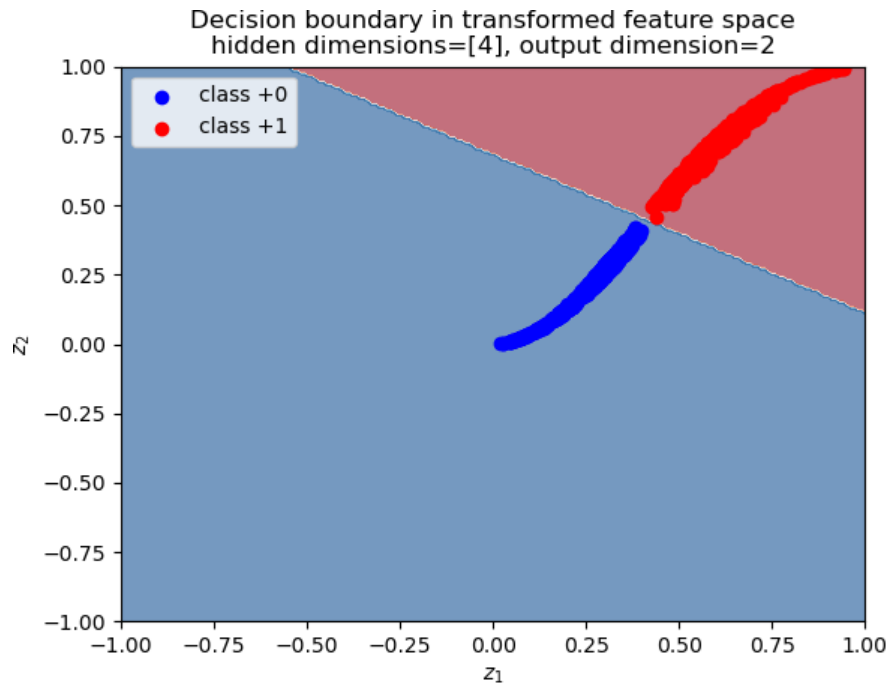

Learned features of sinusoid data
hidden dimensions=[2], output dimension=2
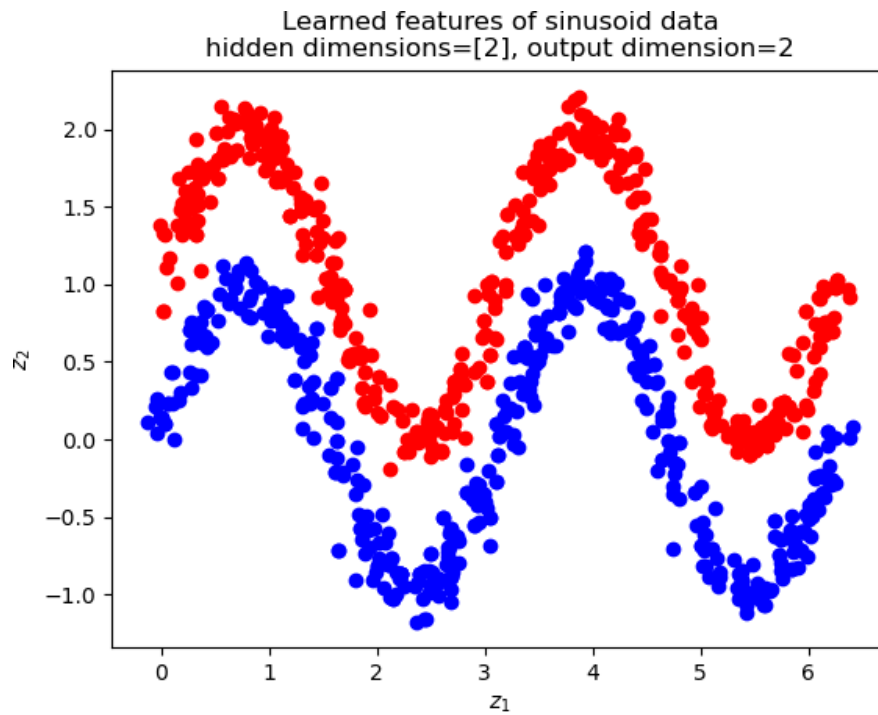
Here is the linear classifier learned by the predictor in this transformed feature space:

Decision boundary in transformed feature space
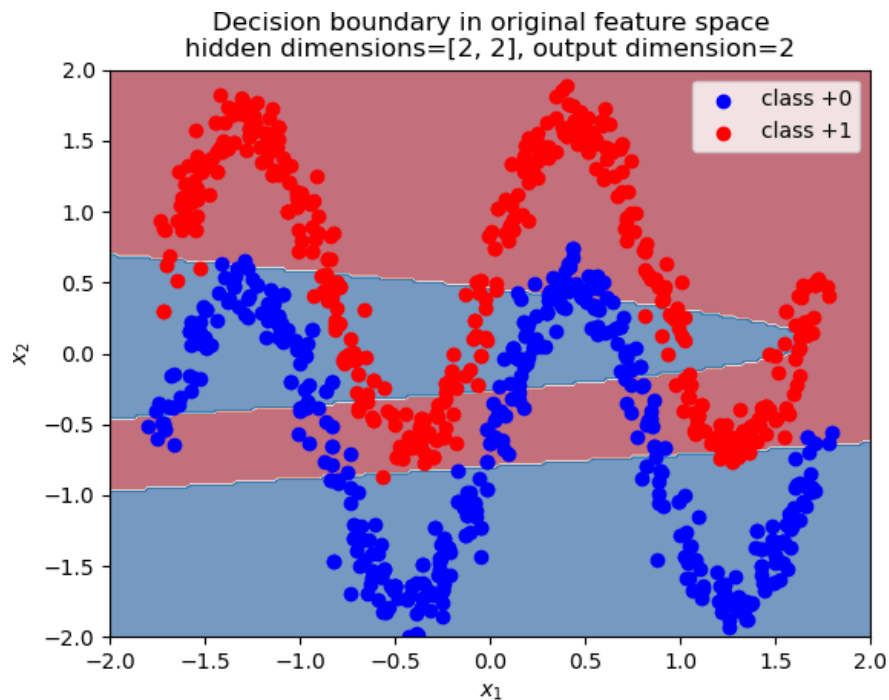hidden dimensions=[2], output dimension=2
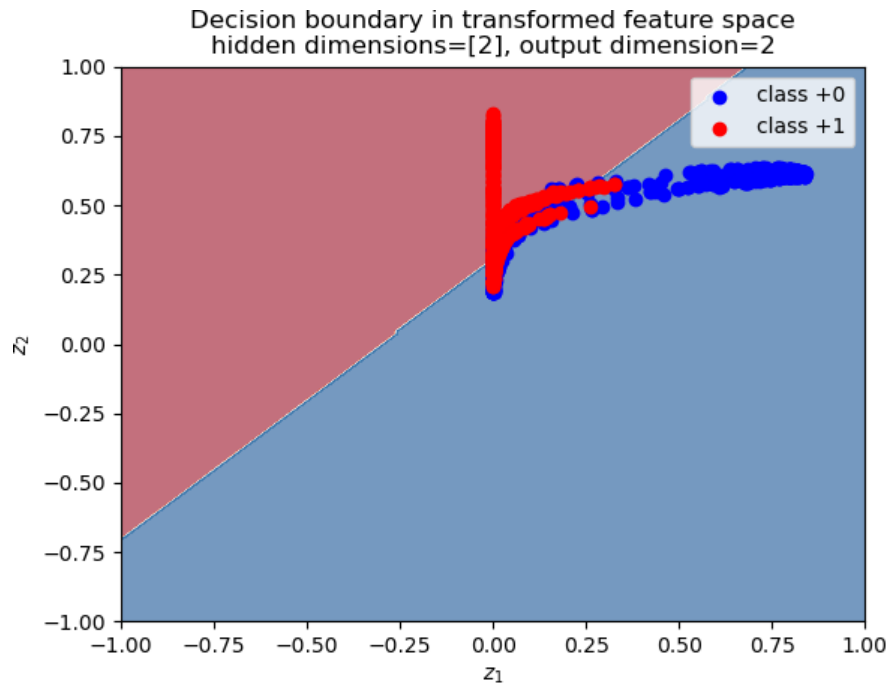
This particular neural network solution does not classify the given examples very well. However, if we use a 4-dimensional hidden features (with all else equal), we get a better-looking result:
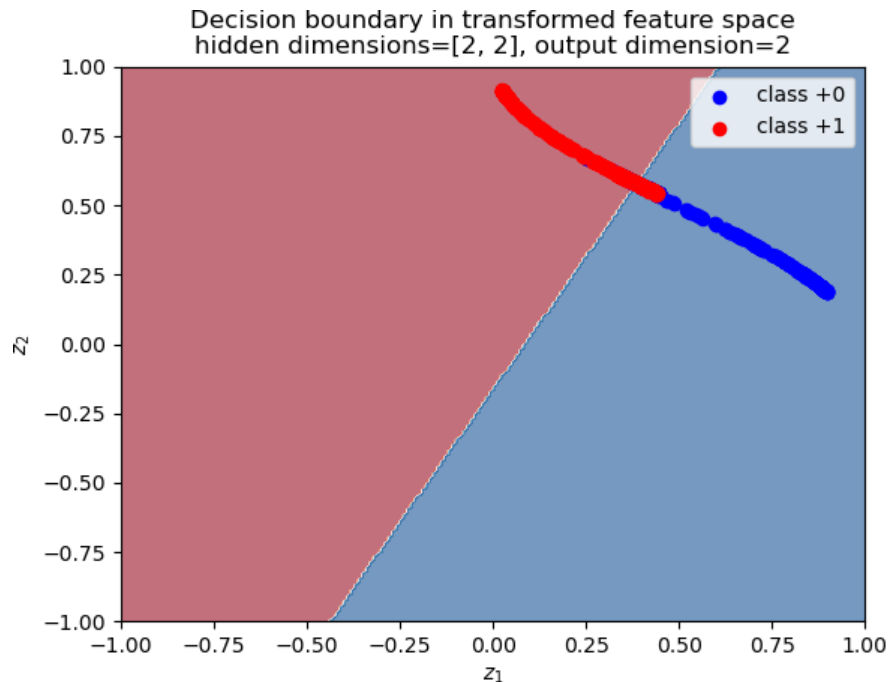

Decision boundary in original feature space
hidden dimensions=[4], output dimension=2

Decision boundary in transformed feature space
hidden dimensions=[4], output dimension=2

Note: in this case, since the hidden feature space has dimension 4, we are only able to plot 2 of the 4 dimensions. The actual linear classifiers is a 3D hyperplane dividing the 4D space. However, even with these two dimensions that we can plot, we can already see that the training data looks pretty much linearly separable, and we learn a good classifier.

Previously we changed the *hidden layer size* from 2 to 4. Another change we can make is to increase the *number of layers* from 1 to 2. Let's use 2 layers of hidden features of size 2:

Decision boundary in original feature space
hidden dimensions=[2, 2], output dimension=2

Decision boundary in transformed feature space
hidden dimensions=[2, 2], output dimension=2

Answer the following questions and provide a brief explanation for each:

1. Why are the axes (z_1 and z_2) of the learned feature space scatterplot constrained to $[0, 1]$? Note/hint: here, our definition of z is after the nonlinearity is applied, whereas in the lecture z is defined as before the nonlinearity is applied. Sorry about that!

   Answer: The activation function was a sigmoid function for applying the nonlinearity constraints the values of z_1 and z_2 to $[0,1]$.

2. Does it make sense to use neural networks on datasets whose examples are linearly separable in the original feature space?

   Answer: No, a linear classifier would be less complex and more efficient solution to use when examples are already linearly separable. However, neural networks can capture linear patterns in the data as well, and would result in suitable solutions as well, albeit with likely more complexity and time.

3. Why would increasing the dimensionality of the hidden features (which is a hyper-parameter) help improve the performance of our classifier?

   Answer: Increasing the dimensionality of the hidden features is similar to performing a transform that increases the number of feature dimensions in the data. Learning new features based on the relationships and patterns of original features help the model capture the data's complexity and learn the decision boundaries, and thus allowing the data to be separable.

4. Why would increasing the number of layers help improve the performance of our classifier?

   Answer: Increasing the number of layers in a neural networks enables the model to learn a hierarchy of features to model more complex, higher-order features from preceding layers.

5. Neural networks are known to suffer problems due to a highly non-convex objective function. What is one way we can address this problem and increase the chances of discovering a global optimum? (Hint: look at the code.)

   Answer: Initializing the model from random starting points helps in keeping track of the model that

yields the best validation error. It iterates over different random seeds, training the data, and discovering parameters that work best. This is done in the code using "warm-start".

4 Very-Short Answer Questions [14 points]

Answer each of the following questions in a sentence or two.

1. Give one reason why you might want sparse solutions with a latent-factor model.

   Answer: Spare solution reduces memory storage required for the model and computational resources for training and testing.

2. Which is better for recommending movies to a new user, collaborative filtering or content-based filtering? Briefly justify your answer.

   Answer: Content-based filtering is better as it does not require previous data from the user to make recommendations (which is unavailable for a new user). It relies on movie ratings instead of previous users' preferences.

3. Are neural networks mainly used for supervised or unsupervised learning? Are they parametric or nonparametric?

   Answer: Neural networks are mainly used for supervised learning and are parametric models.

4. Why might regularization become more important as we add layers to a neural network?

   Answer: As layers are added, a neural network can learn even more complex patterns which could lead to over-fitting. Regularization helps prevent over fitting training data with the neural network by penalizing complexity.

5. Consider using a fully-connected neural network for 5-class classification on a problem with $d = 20$. If the network has one hidden layer of size $k = 100$, how many parameters (including biases), does the network have?

   Answer: There is 20 x 100 w weights, 1 x 100 input biases, 5 x 100 v weights, and 5 output biases. In total 2605 parameters.

6. What is the "vanishing gradient" problem with neural networks based on sigmoid non-linearities?

   Answer: Gradients of the sigmoid function are near zero the further away it is from the origin, resulting in neural network weights not updating during back propagation.

7. Convolutional networks seem like a pain... why not just use regular ("fully connected") neural networks for image classification?

   Answer: CNNs provide spatial awareness to each pixel in the image when convolution is performed. Furthermore CNNs reduce the dimensions of the input and in turn reduce the number of parameters.