CPSC 340 Assignment 6 (due 2021-06-18 at 11:55pm)

Answer: We are providing solutions because supervised learning is easier than unsupervised learning, and so we think having solutions available can help you learn. However, the solution file is meant for you alone and we do not give permission to share these solution files with anyone. Both distributing solution files to other people or using solution files provided to you by other people are considered academic misconduct. Please see UBC's policy on this topic if you are not familiar with it:
http://www.calendar.ubc.ca/vancouver/index.cfm?tree=3,54,111,959
http://www.calendar.ubc.ca/vancouver/index.cfm?tree=3,54,111,960

Important: Submission Format

Please make sure to follow the submission instructions posted on Piazza. **We are entitled to deduct 50% of your marks if the submission format is incorrect.**

1 Robust PCA for Background Subtraction

If you run `python main -q 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. 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.

1.1 Implementing Robust PCA

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. If you run `python main.py -q 1.1` , the program will repeat the same procedure as the above section, but will attempt to use the robust PCA method, whose objective functions are yet to be implemented.

In `fun_obj.py`, you will find classes named `FunObjRobustPCAFactors` and `FunObjRobustPCAFeatures`. Complete the `evaluate()` method for each, which uses a smooth approximation to the absolute value to implement robust PCA. Submit (1) your code in `fun_obj.py`, and (2) one of the resulting figures. Briefly comment on the results.

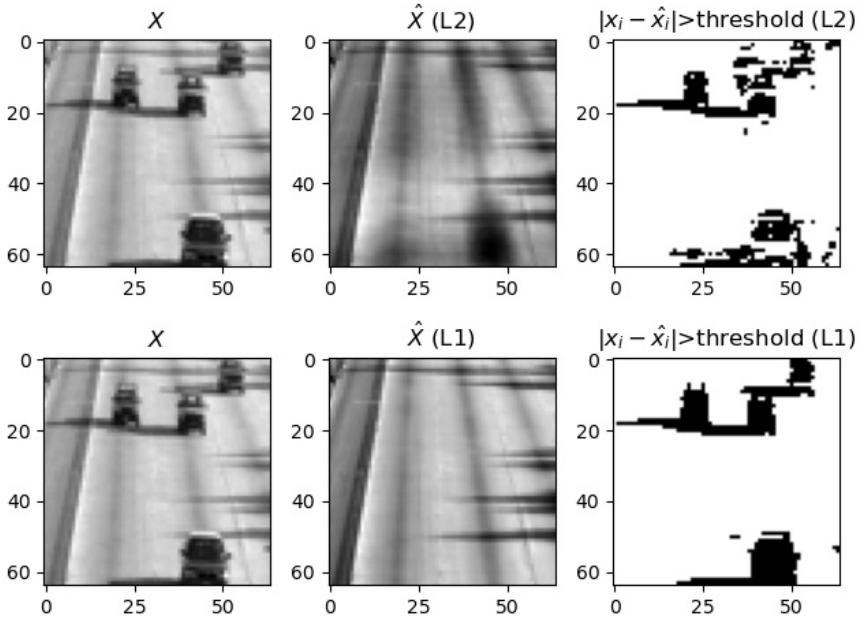
Hint: Your code will look very similar to the already implemented `FunObjPCAFactors` and `FunObjPCAFeatures` classes. Note that the arguments for `evaluate()` are carefully ordered and shaped to be compatible with

our optimizers. Gradient-based approach to PCA can be modified to use a smooth approximation of the L1-norm. Note that the log-sum-exp approximation to the absolute value may be hard to get working due to numerical issues, and a numerically-nicer approach is to 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).

Answer: See code. The robust PCA is somewhat sensitive to initialization, so multiple runs might be required to get a reasonable result. The results look slightly better in terms of isolating foreground (car) from background (road).



1.2 Reflection

1. Briefly explain why using the L1 loss might be more suitable for this task than L2.

   Answer: L1 is "robust to outliers", which means it ignores them, or, in other words, will fail to reconstruct them. This is good, as we want to detect outliers here - and our detection criterion is failure to reconstruct parts of the image.

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

   Answer: The number of frames is n, the total number of pixels per frame (width times height) is d, and k is a hyperparameter unrelated to these things, as long as it's less than d.

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 fewer false positives but more false negatives.

2 Data Visualization

If you run `python main.py -q 2`, it will load the animals dataset from Assignment 5 and create a scatterplot based on two randomly selected features.

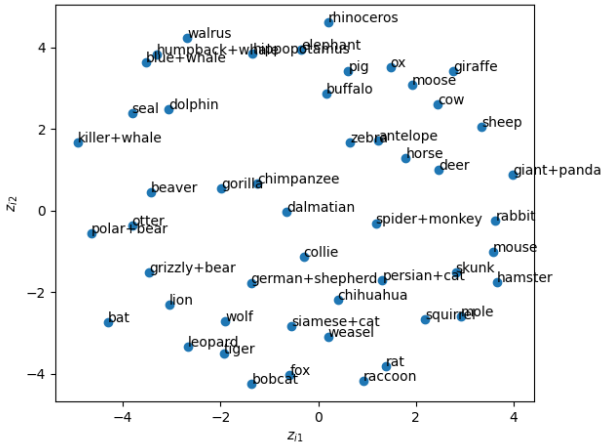
In Assignment 5, we used PCA to visualize our examples on a 2-dimensional view. Your task is to explore multi-dimensional scaling (MDS), ISOMAP, and t-SNE to visualize this dataset in alternative ways.

2.1 Multi-Dimensional Scaling

If you run `python main.py -q 2.1`, the code will load the animals dataset and then apply gradient descent to minimize the following multi-dimensional scaling (MDS) objective (starting from the PCA solution):

$$f(Z) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=i+1}^{n}(\|z_i - z_j\| - \|x_i - x_j\|)^2. \tag{1}$$

The result of applying MDS is shown below.



Although this visualization isn't perfect (with "gorilla" being placed close to the dogs and "otter" being placed close to two types of bears), this visualization does organize the animals in a mostly-logical way.

The local minimum of MDS objective function, returned by the optimizer is sensitive to initialization. In `main.py`, you will see that this MDS model is an instance of `FactorlessEncoderGradient`. By default, this class initializes Z by sampling each entry from a standard normal distribution.

Use the class `FactorlessEncoderGradientWarmStart` in conjunction with `LinearEncoderPCA` to perform MDS based on the results of PCA with 2 principal components.

Submit your code instantiating an MDS model using PCA to warm-start. Use `fun_obj.evaluate()` with the encoded feature matrix Z to obtain **MDS's objective f values** for (1) MDS solution without warm-start, (2) MDS solution with warm-start, and (3) PCA solution. Is the MDS objective f value indeed lower for the MDS solution?

Answer: The MDS objective is around 1832 without warm-start and 1777 with warm-start at the MDS solution and 4942 at the PCA solution; yes, it's lower as expected, but warm-start seems to lower the objective a tiny bit.

2.2 ISOMAP

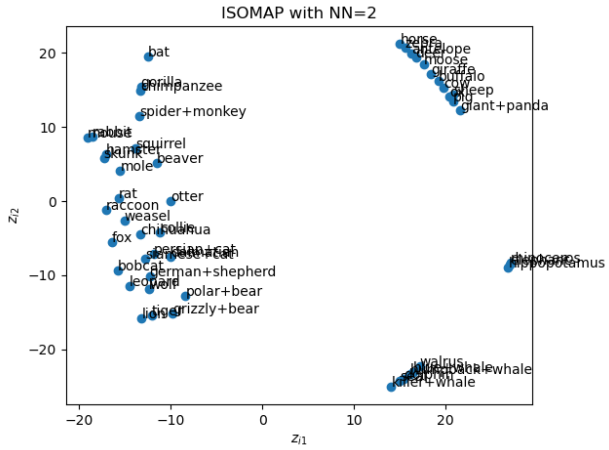
Euclidean distances between very different animals are unlikely to be particularly meaningful. However, since related animals tend to share similar traits we might expect the animals to live on a low-dimensional manifold. This suggests that ISOMAP may give a better visualization. In `fun_obj.py`, you will find a class named `FunObjMDSGeodesic`, which computes the same objective function and gradient as the Euclidean distance-based MDS but **uses approximated geodesic distances instead.** Complete the constructor of

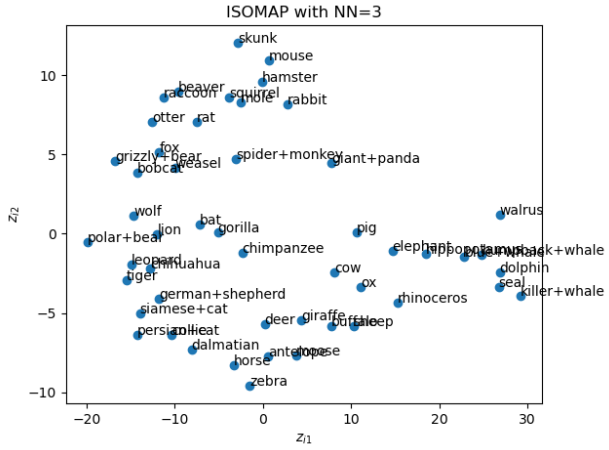
this class, such that it computes the approximate geodesic distance (shortest path through a graph where the edges are only between nodes that are k-nearest neighbours) between each pair of points. Submit your code for `FunObjMDSGeodesic`'s constructor. Use PCA with $k = 2$ to warm-start the MDS and perform ISOMAP onto the matrix X to get learned feature matrix Z. Plot the results using 2 and using 3-nearest neighbours.

Note: when we say 2 nearest neighbours, we mean the two closest neigbours excluding the point itself. This is the opposite convention from what we used in KNN at the start of the course.

The function `utils.shortest_dist` can be used to compute the shortest (weighted) distance between two points in a weighted graph. This function requires an $n \times n$ matrix giving the weights on each edge (use 0 as the weight for absent edges). Note that ISOMAP uses an undirected graph, while the k-nearest neighbour graph might be asymmetric. One of the usual heuristics to turn this into a undirected graph is to include an edge i to j if i is a KNN of j or if j is a KNN of i. (Another possibility is to include an edge only if i and j are mutually KNNs.)

Answer: We can implement ISOMAP by changing the distance function. See code. This assumes that we add an edge if i is a KNN of j or vice versa, but other distance functions in the spirit of constructing a KNN graph are also acceptable. The result of using this weighting, for 2 and 3 neighbours respectively, is:



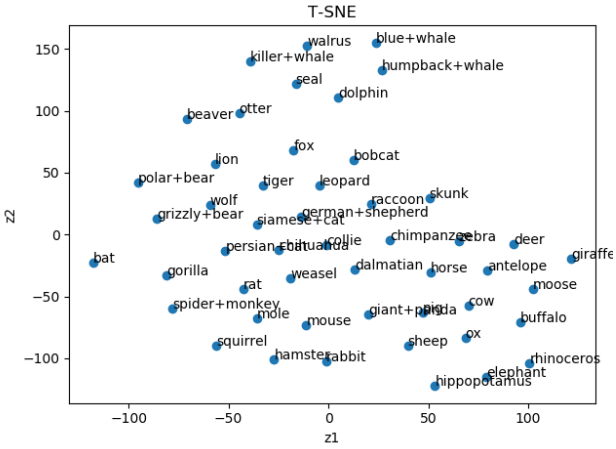


2.3 t-SNE

Try running scikit-learn's t-SNE on this dataset as well. Submit the plot from running t-SNE. Then, briefly comment on PCA vs. MDS vs. ISOMAP vs. t-SNE for dimensionality reduction on this particular data set. In your opinion, which method did the best job and why?

Note: There is no single correct answer here! Also, please do not write more than 3 sentences.

Answer: This is a matter of opinion and the goal of the question is to make sure you at least looked at the plots. I like ISOMAP with 2 neighbours because it clearly separates some sensible groups. One question though is that giant panda is away from the rest of the bears. t-SNE figure is below:



2.4 Sensitivity to Initialization

For each of the four methods (MDS without warm-start, MDS with warm-start, ISOMAP, t-SNE) tried above, which ones give different results when you re-run the code? Does this match up with what we discussed in lectures, about which methods are sensitive to initialization and which ones aren't? Briefly discuss.

Answer: t-SNE and MDS without warm start gives different results each time. PCA is not sensitive to initialization. MDS and ISOMAP are, but we're not initializing them randomly, we're using the (deterministic) PCA solution. scikit-learn is presumably using a random initialization for t-SNE.

3 Neural Networks

3.1 Neural Networks by Hand

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):

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

Assuming that we are doing regression, for a training example with features $x_i^T = \begin{bmatrix} -3 & -2 & 2 \end{bmatrix}$ what are the values in this network of z_i, $h(z_i)$, and $\hat{y}_i$?

Answer:

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

$$h(z_i) = \begin{bmatrix} \frac{1}{1+\exp(-0)} \\ \frac{1}{1+\exp(-1)} \end{bmatrix} = \begin{bmatrix} \frac{1}{2} \\ \frac{1}{1+1/e} \end{bmatrix} = \begin{bmatrix} \frac{1}{2} \\ \frac{e}{1+e} \end{bmatrix}.$$

$$\hat{y}_i = v^T h(z_i) = \begin{bmatrix} 3 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{2} \\ \frac{e}{1+e} \end{bmatrix} = 3/2 + e/(1+e).$$

3.2 Neural Networks vs. Softmax

NOTE: before starting this question you need to download the MNIST dataset from `http://deeplearning.net/data/mnist/mnist.pkl.gz` and place it in your *data* directory.

If you run `python main.py -q 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 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.

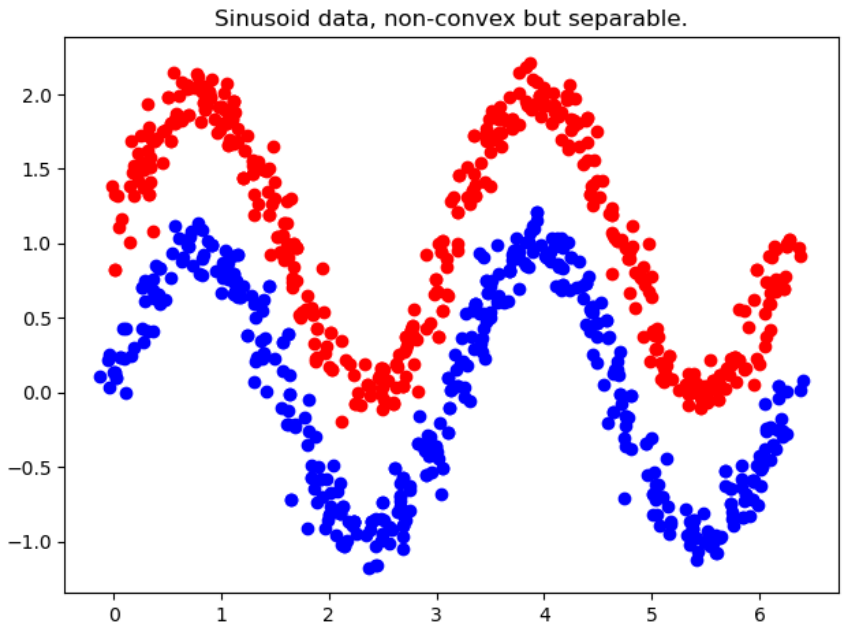
If you run `python main.py -q 3.1`, the program will train a single-layer neural network (one-layer nonlinear encoder paired with a linear classifier) on the same dataset using some pre-tuned hyperparameters. Modify the code, play around with the hyperparameter values (configurations of encoder layers, batch size and learning rate of SGD, standardization, etc.) until you achieve validation error that is reliably below 8%. Report (1) the training and validation errors that you obtain with your hyperparameters, and (2) the hyperparameters that you used.

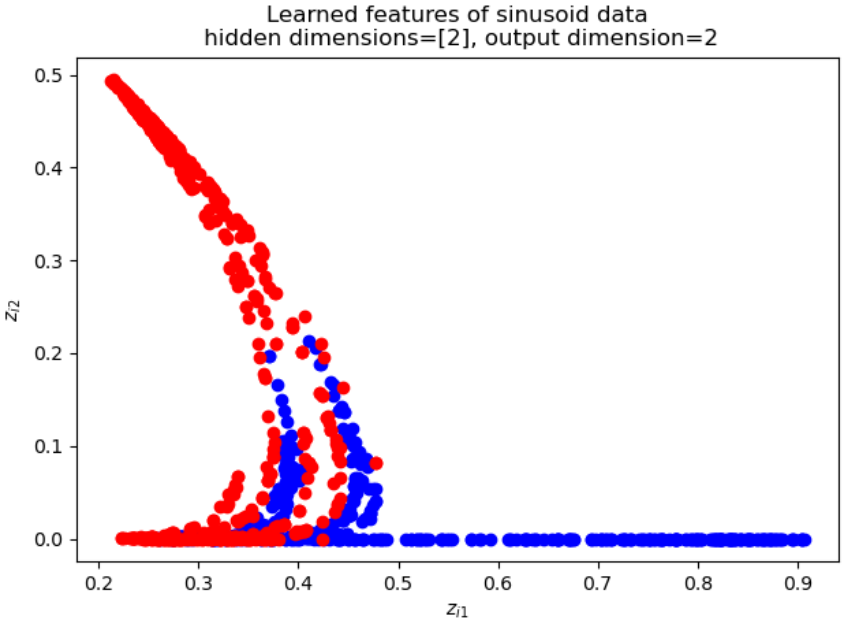
Answer: See code. Standardization alone makes a huge difference, at around 5.5% training error and around 6.3% validation error. Training for more epochs also help. Using the inverse-square-root step size strategy helps. I also decreased L2-regularization by setting $\lambda = 1e-3$, and this works surprisingly better. Contrary to the popular belief, increasing the depth of the encoder is not as helpful. Merely increasing the width of the encoder to 256 gets me down to 2.6% training error and 3.8% validation error.

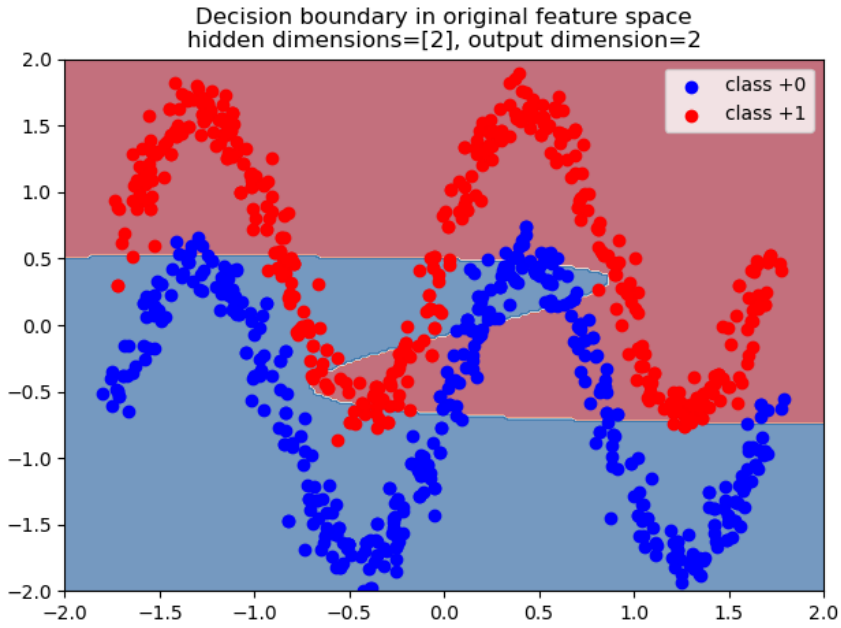
3.3 Width and Depth of Neural Networks

An intuitive way to think about a neural network is to think of it as a composition of an encoder and a predictor. The predictor signals the encoder to learn a better feature space, in such a way that the final linear predictions are meaningful.

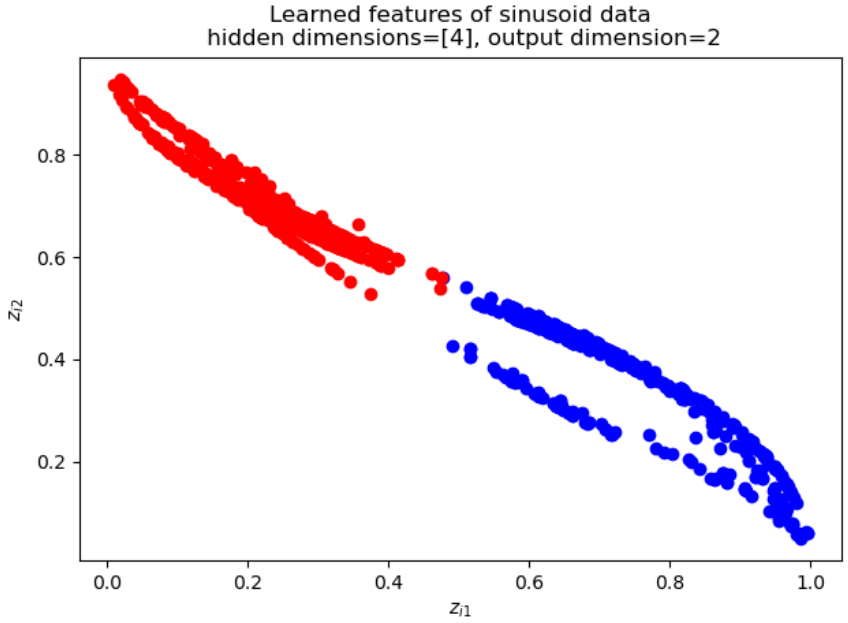
The following figures result from running `python main.py -q 3.3`. For this, a neural network model is used to encode the examples into a 2-dimensional feature space. A single hidden feature layer of 2 neurons is used:

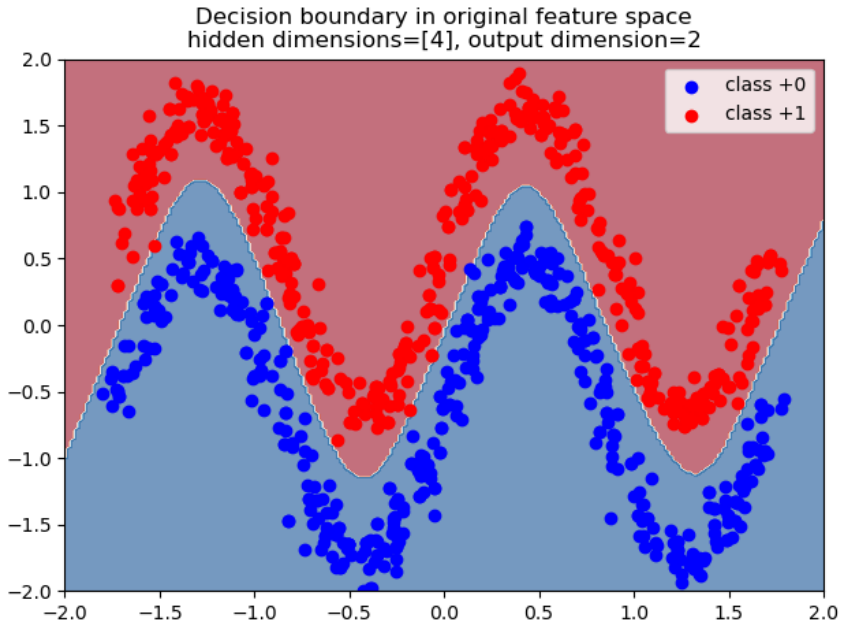

Sinusoid data, non-convex but separable.
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
0
1
2
3
4
5
6


Learned features of sinusoid data
hidden dimensions=[2], output dimension=2
0.5
0.4
0.3
0.2
0.1
0.0
z_{i2}
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
z_{i1}

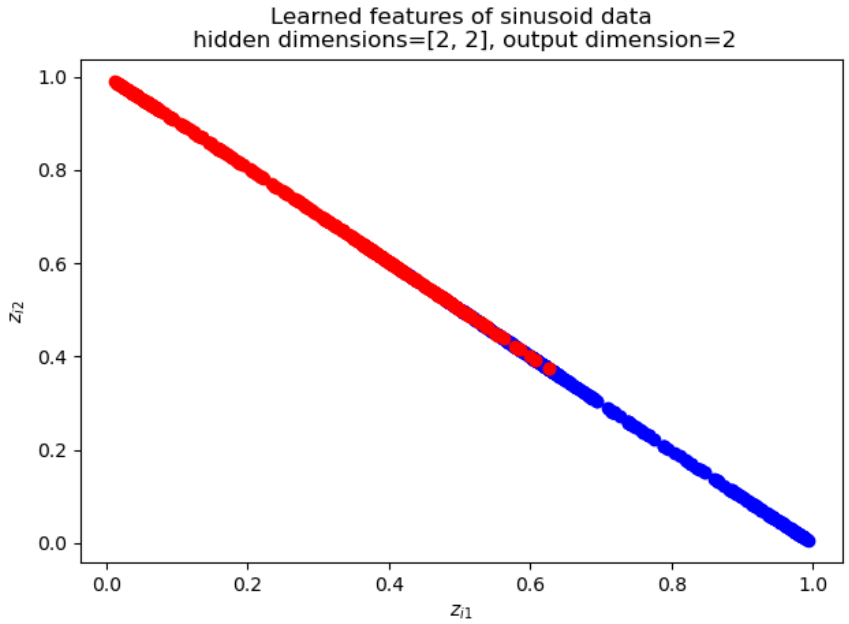


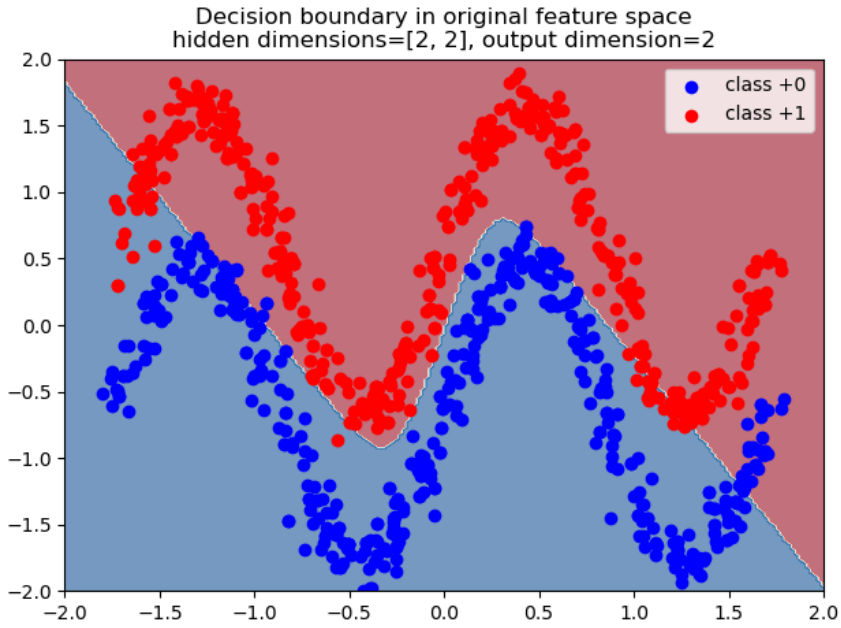
This particular neural network solution does not classify the given examples very well. However, if we use a 4-dimensional hidden features (with all else equal), we get a better-looking result:





Another change we can make is to increase the number of layers. Let's use 2 layers of hidden features of size 2:





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

1. Why are the x- and y-axes of the learned feature space scatterplot constrained to $[0, 1]$?

   Answer: The sigmoid activation makes things between 0 and 1.

2. Given the figures showing the learned features of the sinusoid data, does it make sense to use neural networks on datasets whose examples are convex and linearly separable in the original feature space?

   Answer: No, linearly separable data would have no benefit from using a complicated non-linear encoder. We only need a linear classifier in that case.

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

   Answer: "Disentangling" becomes easier once the hidden feature space is high-dimensional, as with the polynomial and exponential "change-of-basis". There are more configurations to generate linearly separable features.

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

   Answer: Although deeper networks require more sophisticated optimizers to train, deeper networks would compose non-linear transformations that can eventually approximate the sinusoidal function. (However, with vanilla gradient descent or SGD, training deep networks is very hard!)

5. Neural networks are known to suffer problems due to a highly non-convex objective function. How can one address this problem and discover the global optima? (Hint: look at the code.)

   Answer: We initialize and train the model randomly across multiple seeds, and then choose the one giving the best validation error. I used continuous warm-start to keep resetting the optimizer even after convergence to break out of the saddle points.

4 Very-Short Answer Questions

1. Why might you might want sparse solutions with a latent-factor model.

Answer: Interpretability, faster computation.

2. Is ISOMAP mainly used for supervised or unsupervised learning? Is it parametric or non-parametric?

   Answer: Unsupervised; Non-parametric.

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

   Answer: Content-based filtering. Collaborative filtering relies on ratings of that user, but there won't be any for a new user.

4. Collaborative filtering and PCA both minimizing the squared error when approximating each x_{ij} by $\langle w^j, z_i \rangle$; what is the difference between them?

   Answer: Collaborative filtering is used for recommender systems, is solving a supervised problem, and is usually done on a matrix where we don't know all the entries. PCA assumes you have all entries of the matrix, is an unsupervised method, and is used for a variety of things like visualization and factor discovery.

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

   Answer: The model is becoming more complex, which means a higher danger of overfitting. Regularization combats overfitting.

6. With stochastic gradient descent, the loss might go up or down each time the parameters are updated. However, we don't actually know which of these cases occurred. Explain why it doesn't make sense to check whether the loss went up/down after each update.

   Answer: This would require iterating over all the examples, which is exactly what we're trying to avoid with SGD.

7. Consider using a fully-connected neural network for 3-class classification on a problem with $d = 10$. If the network has one hidden layer of size $k = 100$, how many parameters (including biases), does the network have?

   Answer: Weights: $10 \times 100 + 100 \times 3 = 1300$. Biases: $100 + 3 = 103$. Total: $1300 + 103 = 1403$.

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

   Answer: Gradient might be zero because of overflow/underflow in the intermediate calculations (especially when repeated across layers)

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

   Answer: The number of parameters would be huge; we would overfit massively and run into prohibitive speed/memory issues.