

## Policies

- Due 9 PM PST, January 12<sup>th</sup> on Gradescope.
- You are free to collaborate on all of the problems, subject to the collaboration policy stated in the syllabus.
- If you have trouble with this homework, it may be an indication that you should drop the class.
- In this course, we will be using Google Colab for code submissions. You will need a Google account.

## Submission Instructions

- Submit your report as a single .pdf file to Gradescope (entry code 7426YK), under "Set 1 Report".
- In the report, **include any images generated by your code** along with your answers to the questions.
- Submit your code by **sharing a link in your report** to your Google Colab notebook for each problem (see naming instructions below). Make sure to set sharing permissions to at least "Anyone with the link can view". **Links that can not be run by TAs will not be counted as turned in.** Check your links in an incognito window before submitting to be sure.
- For instructions specifically pertaining to the Gradescope submission process, see [https://www.gradescope.com/get\\_started#student-submission](https://www.gradescope.com/get_started#student-submission).

## Google Colab Instructions

For each notebook, you need to save a copy to your drive.

1. Open the github preview of the notebook, and click the icon to open the colab preview.
2. On the colab preview, go to File → Save a copy in Drive.
3. Edit your file name to "lastname\_firstname\_originaltitle", e.g. "yue-yisong\_3\_notebook\_part1.ipynb"

## 1 Basics [16 Points]

*Relevant materials: lecture 1*

Answer each of the following problems with 1-2 short sentences.

**Problem A [2 points]:** What is a hypothesis set?

**Solution A:** *A hypothesis set is the set of all hypotheses, or functions, that map the input space to the output space and that could be returned by the learning algorithm.*

**Problem B [2 points]:** What is the hypothesis set of a linear model?

**Solution B:** *The hypothesis set of a linear model is the set of all  $f(x|w) = w^T x + b$ .*

**Problem C [2 points]:** What is overfitting?

**Solution C:** *Overfitting occurs when the test error is much greater than the training error. This means that the model fits the training data too well and cannot accurately make predictions on the test data.*

**Problem D [2 points]:** What are two ways to prevent overfitting?

**Solution D:** *One way to prevent overfitting is to decrease the complexity of the model class. Another way to prevent overfitting is to use more training data.*

**Problem E [2 points]:** What are training data and test data, and how are they used differently? Why should you never change your model based on information from test data?

**Solution E:** *Training data is the data used to train a model, and test data is the data used to evaluate the performance of the trained model. Test data is also not used for training. We should never change our model based on the information from test data because this may make the model biased towards the test data.*

**Problem F [2 points]:** What are the two assumptions we make about how our dataset is sampled?

**Solution F:** *We assume that our dataset is sampled independently and identically distributed from the "true" probability distribution over all possible data.*

**Problem G [2 points]:** Consider the machine learning problem of deciding whether or not an email is spam. What could  $X$ , the input space, be? What could  $Y$ , the output space, be?

**Solution G:** *The input space could be a vector with parameters extracted using keywords or phrases, like the bag of words representation. The output space could be spam or not spam which could be represented by the binary output 1 or 0.*

**Problem H [2 points]:** What is the  $k$ -fold cross-validation procedure?

**Solution H:** *The first step of the  $k$ -fold cross-validation procedure is to split the original data set into  $k$  equal parts. Then, we train on  $k - 1$  parts and test on the remaining part. We repeat this for every choice of  $k - 1$  parts and average the validation errors.*

## 2 Bias-Variance Tradeoff [34 Points]

*Relevant materials: lecture 1*

**Problem A [5 points]:** Derive the bias-variance decomposition for the squared error loss function. That is, show that for a model  $f_S$  trained on a dataset  $S$  to predict a target  $y(x)$  for each  $x$ ,

$$\mathbb{E}_S [E_{\text{out}}(f_S)] = \mathbb{E}_x [\text{Bias}(x) + \text{Var}(x)]$$

given the following definitions:

$$\begin{aligned} F(x) &= \mathbb{E}_S [f_S(x)] \\ E_{\text{out}}(f_S) &= \mathbb{E}_x [(f_S(x) - y(x))^2] \\ \text{Bias}(x) &= (F(x) - y(x))^2 \\ \text{Var}(x) &= \mathbb{E}_S [(f_S(x) - F(x))^2] \end{aligned}$$

### Solution A:

$$\begin{aligned} \mathbb{E}_S [E_{\text{out}}(f_S)] &= \mathbb{E}_S \left[ \mathbb{E}_x [(f_S(x) - y(x))^2] \right] \\ &= \mathbb{E}_x \left[ \mathbb{E}_S [(f_S(x) - y(x))^2] \right] \\ &= \mathbb{E}_x \left[ \mathbb{E}_S [(f_S(x) - F(x) + F(x) - y(x))^2] \right] \\ &= \mathbb{E}_x \left[ \mathbb{E}_S [(f_S(x) - F(x))^2 + 2(f_S(x) - F(x))(F(x) - y(x)) + (F(x) - y(x))^2] \right] \\ &= \mathbb{E}_x \left[ \mathbb{E}_S [(f_S(x) - F(x))^2] + 2\mathbb{E}_x [\mathbb{E}_S [(f_S(x) - F(x))(F(x) - y(x))]] + \mathbb{E}_x [\mathbb{E}_S [(F(x) - y(x))^2]] \right] \\ &= \mathbb{E}_x [\text{Var}(x)] + 2\mathbb{E}_x [\mathbb{E}_S [(f_S(x) - F(x))(F(x) - y(x))]] + \mathbb{E}_x [\text{Bias}(x)] \\ &= \mathbb{E}_x [\text{Var}(x)] + 2\mathbb{E}_x [(F(x) - F(x))(F(x) - y(x))] + \mathbb{E}_x [\text{Bias}(x)] \\ &= \mathbb{E}_x [\text{Var}(x)] + 0 + \mathbb{E}_x [\text{Bias}(x)] \\ &= \mathbb{E}_x [\text{Var}(x) + \text{Bias}(x)] \\ &= \mathbb{E}_x [\text{Bias}(x) + \text{Var}(x)] \end{aligned}$$

In the following problems you will explore the bias-variance tradeoff by producing learning curves for polynomial regression models.

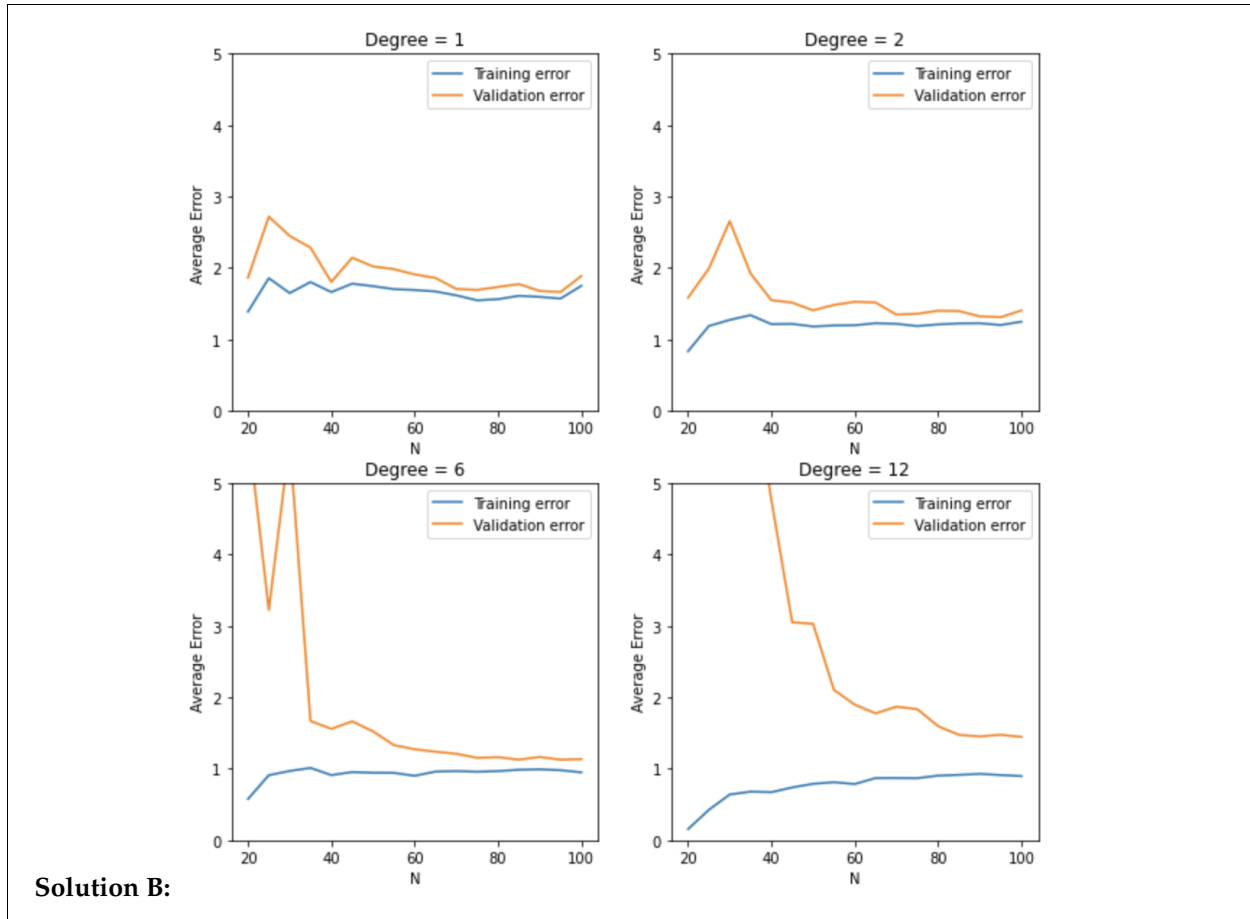
A *learning curve* for a model is a plot showing both the training error and the cross-validation error as a function of the number of points in the training set. These plots provide valuable information regarding the bias and variance of a model and can help determine whether a model is over- or under-fitting.

*Polynomial regression* is a type of regression that models the target  $y$  as a degree- $d$  polynomial function of the input  $x$ . (The modeler chooses  $d$ .) You don't need to know how it works for this problem, just know that it produces a polynomial that attempts to fit the data.

**Problem B [14 points]:** Use the provided `2_notebook.ipynb` Jupyter notebook to enter your code for this question. This notebook contains examples of using NumPy's `polyfit` and `polyval` methods, and scikit-learn's `KFold` method; you may find it helpful to read through and run this example code prior to continuing with this problem. Additionally, you may find it helpful to look at the documentation for scikit-learn's `learning_curve` method for some guidance.

The dataset `bv_data.csv` is provided and has a header denoting which columns correspond to which values. Using this dataset, plot learning curves for 1st-, 2nd-, 6th-, and 12th-degree polynomial regression (4 separate plots) by following these steps for each degree  $d \in \{1, 2, 6, 12\}$ :

1. For each  $N \in \{20, 25, 30, 35, \dots, 100\}$ :
  - i. Perform 5-fold cross-validation on the first  $N$  points in the dataset (setting aside the other points), computing the both the training and validation error for each fold.
    - Use the mean squared error loss as the error function.
    - Use NumPy's `polyfit` method to perform the degree- $d$  polynomial regression and NumPy's `polyval` method to help compute the errors. (See the example code and [NumPy documentation](#) for details.)
    - When partitioning your data into folds, although in practice you should randomize your partitions, for the purposes of this set, simply divide the data into  $K$  contiguous blocks.
  - ii. Compute the average of the training and validation errors from the 5 folds.
2. Create a learning curve by plotting both the average training and validation error as functions of  $N$ .  
*Hint: Have same y-axis scale for all degrees  $d$ .*



**Problem C [3 points]:** Based on the learning curves, which polynomial regression model (i.e. which degree polynomial) has the highest bias? How can you tell?

**Solution C:** Polynomial regression model with degree = 1 has the highest bias. The 1st degree polynomial had the greatest average training error. We know underfitting implies high bias because the model complexity is too low and is unable to fit the data well.

**Problem D [3 points]:** Which model has the highest variance? How can you tell?

**Solution D:** The 12th degree polynomial regression has the highest variance. We can see that the training error of the 12th degree polynomial is the lowest of all the models. However, at small values of N, the 12th degree polynomial has significantly higher validation error than the other models. This agrees with the fact that variance

---

*increases with model complexity.*

**Problem E [3 points]:** What does the learning curve of the quadratic model tell you about how much the model will improve if we had additional training points?

**Solution E:** *The learning curve of the quadratic model shows that the training error stays about the same after  $N = 40$ , but the validation error still decreases as  $N$  increases. Thus, if we had more training points, the quadratic model will still improve and perform better on the validation sets.*

**Problem F [3 points]:** Why is training error generally lower than validation error?

**Solution F:** *Training error is generally lower than validation error because we train the model to fit the training data. We do not use the validation data to train the model. Thus, the model is trained to best fit the training data rather than the validation data. As a result, the training error is generally lower than the validation error.*

**Problem G [3 points]:** Based on the learning curves, which model would you expect to perform best on some unseen data drawn from the same distribution as the training data, and why?

**Solution G:** *I would expect the 6th degree polynomial regression model to perform best on unseen data drawn from the same distribution as the training data. We see that at smaller values of  $N$ , the quadratic model has the lowest validation error. However, around  $N = 40$  and greater, the 6th degree polynomial has the lowest validation error. Thus, the 6th degree model generalizes the best and would perform the best on unseen data.*

### 3 Stochastic Gradient Descent [36 Points]

*Relevant materials: lecture 2*

Stochastic gradient descent (SGD) is an important optimization method in machine learning, used everywhere from logistic regression to training neural networks. In this problem, you will be asked to first implement SGD for linear regression using the squared loss function. Then, you will analyze how several parameters affect the learning process.

Linear regression learns a model of the form:

$$f(x_1, x_2, \dots, x_d) = \left( \sum_{i=1}^d w_i x_i \right) + b$$

**Problem A [2 points]:** We can make our algebra and coding simpler by writing  $f(x_1, x_2, \dots, x_d) = \mathbf{w}^T \mathbf{x}$  for vectors  $\mathbf{w}$  and  $\mathbf{x}$ . But at first glance, this formulation seems to be missing the bias term  $b$  from the equation above. How should we define  $\mathbf{x}$  and  $\mathbf{w}$  such that the model includes the bias term?

*Hint: Include an additional element in  $\mathbf{w}$  and  $\mathbf{x}$ .*

**Solution A:** We would define  $\mathbf{x} = (x_0 = 1, x_1, x_2, \dots, x_d)$ ,  $\mathbf{w} = (w_0 = b, w_1, w_2, \dots, w_d)$ .

Linear regression learns a model by minimizing the squared loss function  $L$ , which is the sum across all training data  $\{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$  of the squared difference between actual and predicted output values:

$$L(f) = \sum_{i=1}^N (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

**Problem B [2 points]:** SGD uses the gradient of the loss function to make incremental adjustments to the weight vector  $\mathbf{w}$ . Derive the gradient of the squared loss function with respect to  $\mathbf{w}$  for linear regression.

**Solution B:**

$$\begin{aligned} L(f) &= \sum_{i=1}^N (y_i - \mathbf{w}^T \mathbf{x}_i)^2 \\ \frac{\partial L(f)}{\partial w} &= \sum_{i=1}^N -2(y_i - \mathbf{w}^T \mathbf{x}_i) \mathbf{x}_i^j \\ \nabla_{\mathbf{w}} L(f) &= -2 \sum_{i=1}^N (y_i - \mathbf{w}^T \mathbf{x}_i) \mathbf{x}_i \end{aligned}$$

The following few problems ask you to work with the first of two provided Jupyter notebooks for this problem, `3_notebook_part1.ipynb`, which includes tools for gradient descent visualization. This notebook



utilizes the files `sgd_helper.py` and `multiopt.mp4`, but you should not need to modify either of these files.

For your implementation of problems C-E, **do not** consider the bias term.

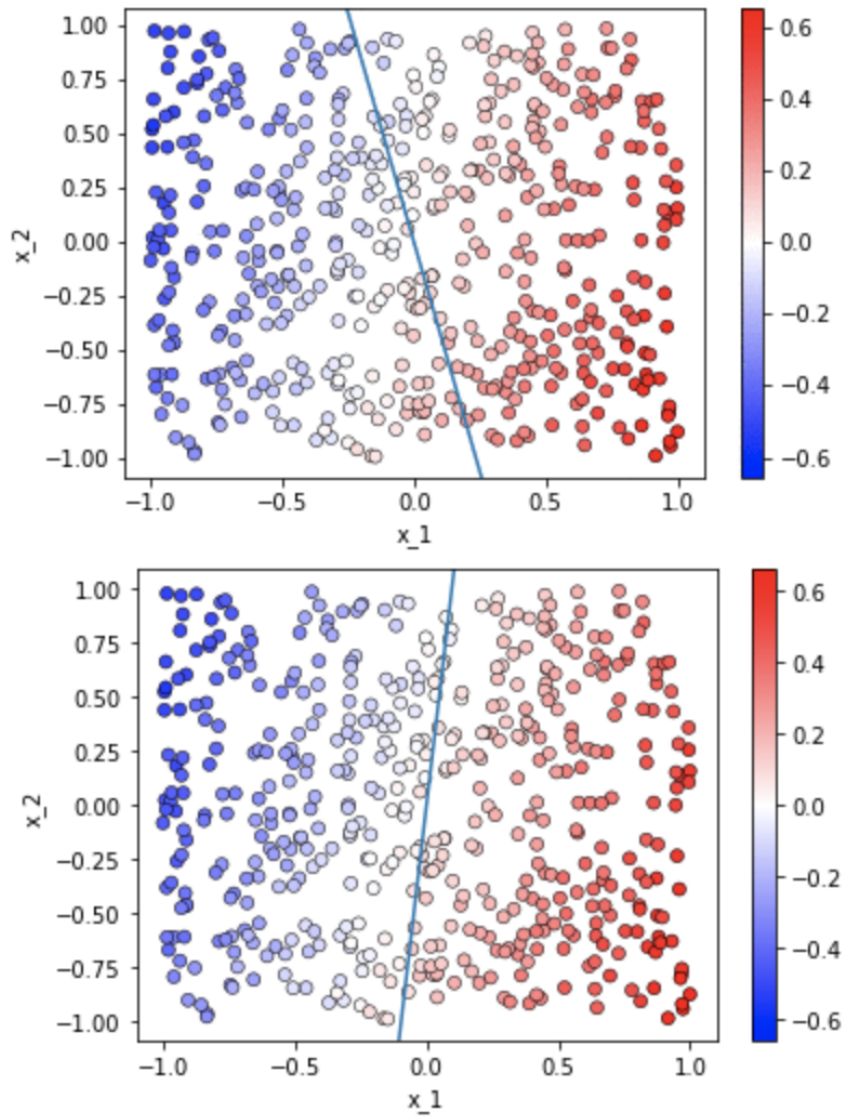
**Problem C [8 points]:** Implement the `loss`, `gradient`, and `SGD` functions, defined in the notebook, to perform SGD, using the guidelines below:

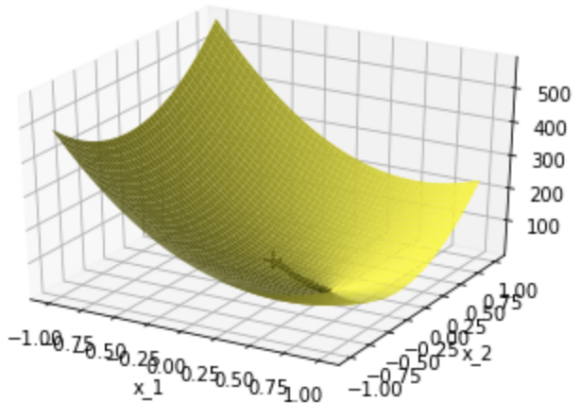
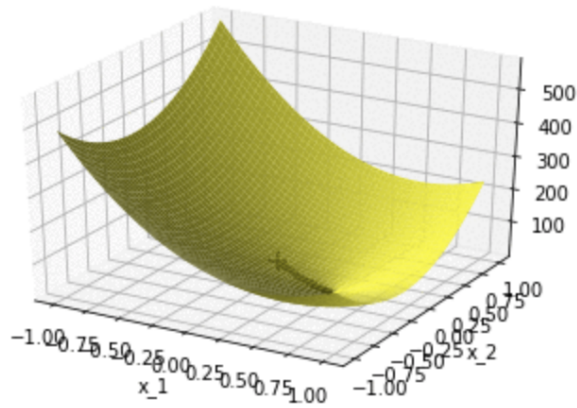
- Use a squared loss function.
- Terminate the SGD process after a specified number of epochs, where each epoch performs one SGD iteration for each point in the dataset.
- It is recommended, but not required, that you shuffle the order of the points before each epoch such that you go through the points in a random order. You can use `numpy.random.permutation`.
- Measure the loss after each epoch. Your `SGD` function should output a vector with the loss after each epoch, and a matrix of the weights after each epoch (one row per epoch). Note that the weights from all epochs are stored in order to run subsequent visualization code to illustrate SGD.

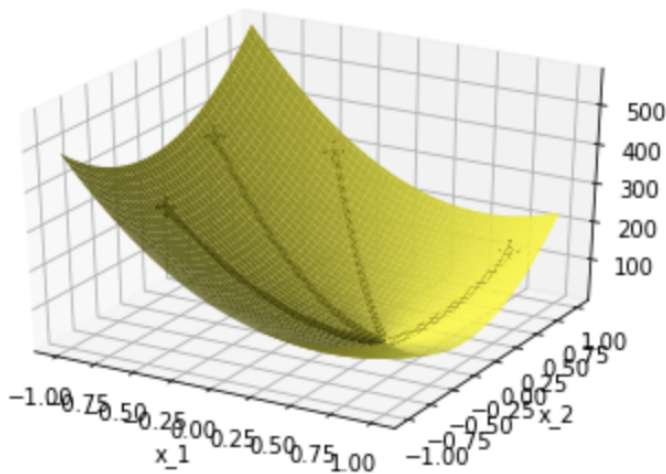
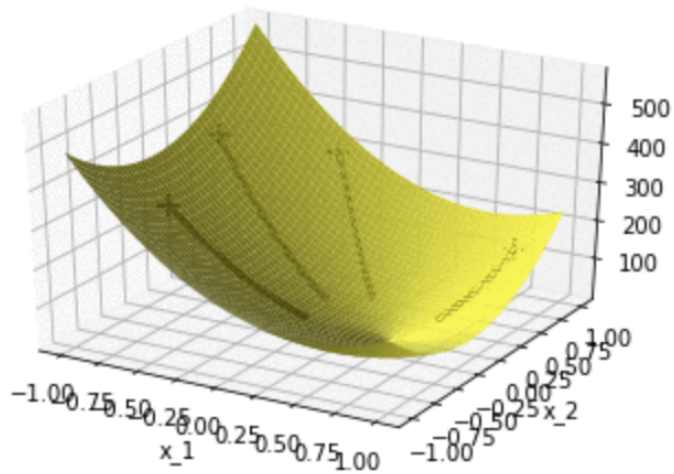
**Solution C:** *See code.*

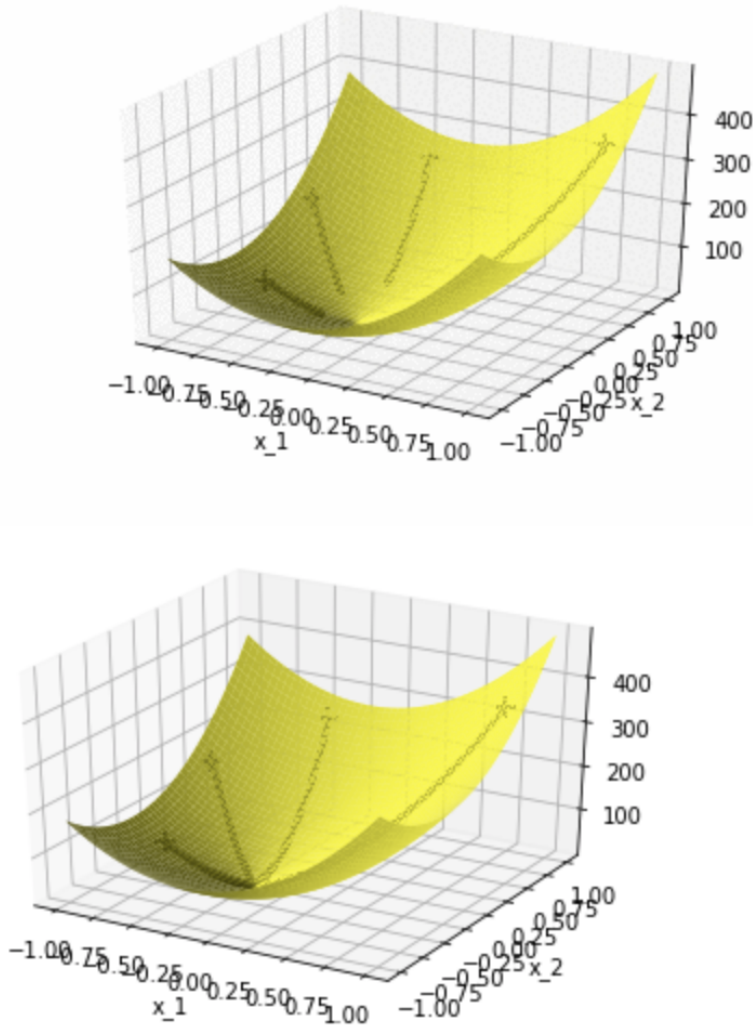
**Problem D [2 points]:** Run the visualization code in the notebook corresponding to problem D. How does the convergence behavior of SGD change as the starting point varies? How does this differ between datasets 1 and 2? Please answer in 2-3 sentences.

**Solution D:** *As the starting point varies, the SGD still converges to the same point and at the same time without any divergence. This is true for both datasets 1 and 2.*

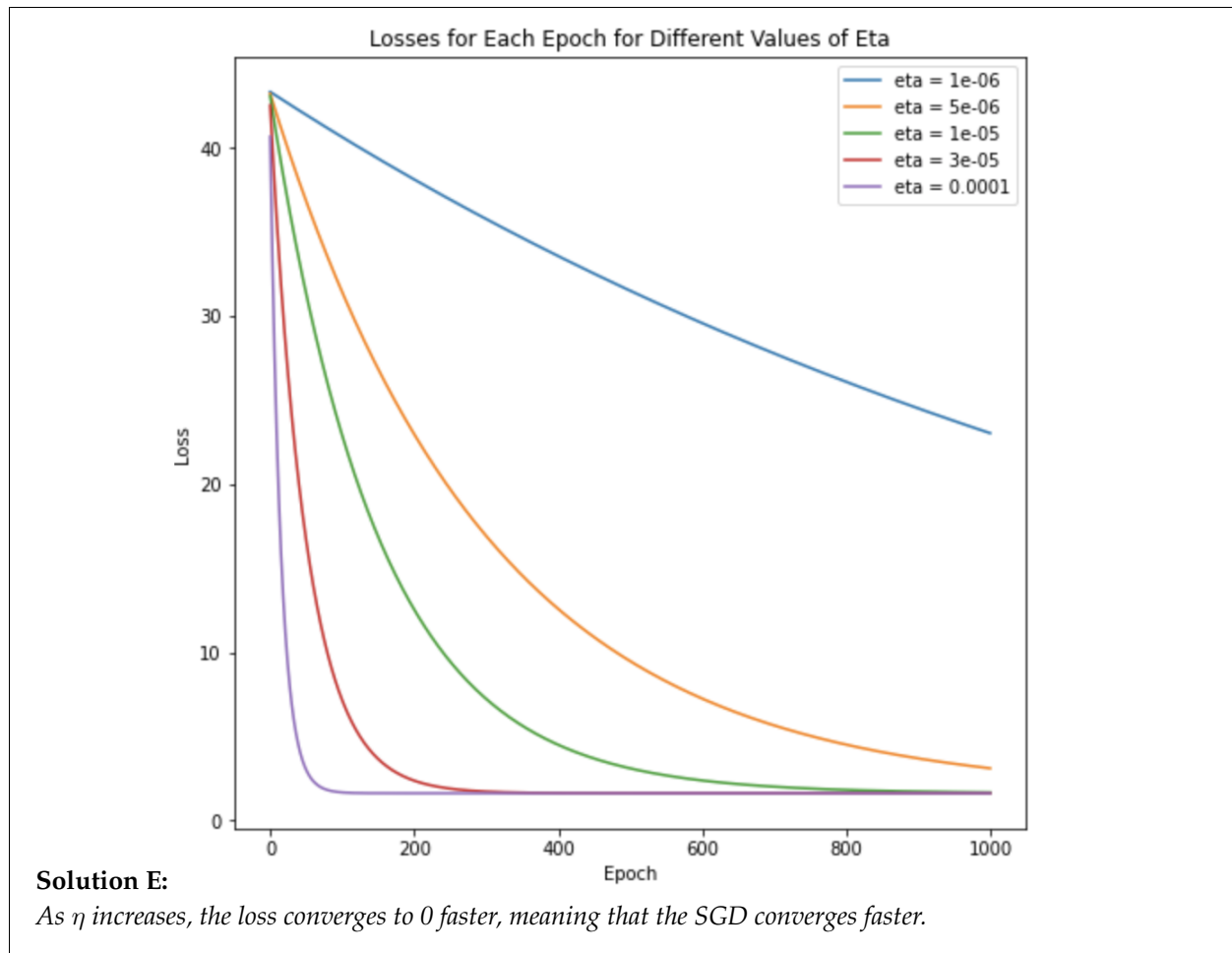


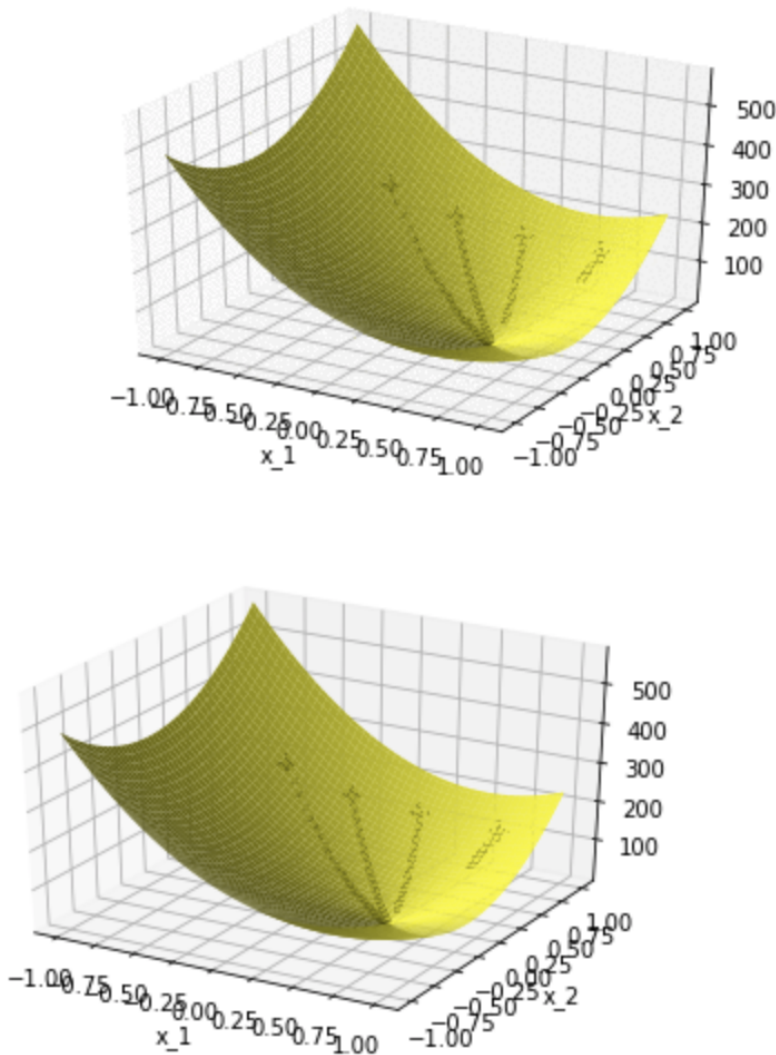






**Problem E [6 points]:** Run the visualization code in the notebook corresponding to problem E. One of the cells—titled “Plotting SGD Convergence”—must be filled in as follows. Perform SGD on dataset 1 for each of the learning rates  $\eta \in \{1e-6, 5e-6, 1e-5, 3e-5, 1e-4\}$ . On a single plot, show the training error vs. number of epochs trained for each of these values of  $\eta$ . What happens as  $\eta$  changes?





The following problems consider SGD with the larger, higher-dimensional dataset, `sgd_data.csv`. The file has a header denoting which columns correspond to which values. For these problems, use the Jupyter notebook `3_notebook_part2.ipynb`.

For your implementation of problems F-H, **do** consider the bias term using your answer to problem A.

**Problem F [6 points]:** Use your SGD code with the given dataset, and report your final weights. Follow the guidelines below for your implementation:

- Use  $\eta = e^{-15}$  as the step size.

- Use  $\mathbf{w} = [0.001, 0.001, 0.001, 0.001]$  as the initial weight vector and  $b = 0.001$  as the initial bias.
- Use at least 800 epochs.
- You should incorporate the bias term in your implementation of SGD and do so in the vector style of problem A.
- Note that for these problems, it is no longer necessary for the SGD function to store the weights after all epochs; you may change your code to only return the final weights.

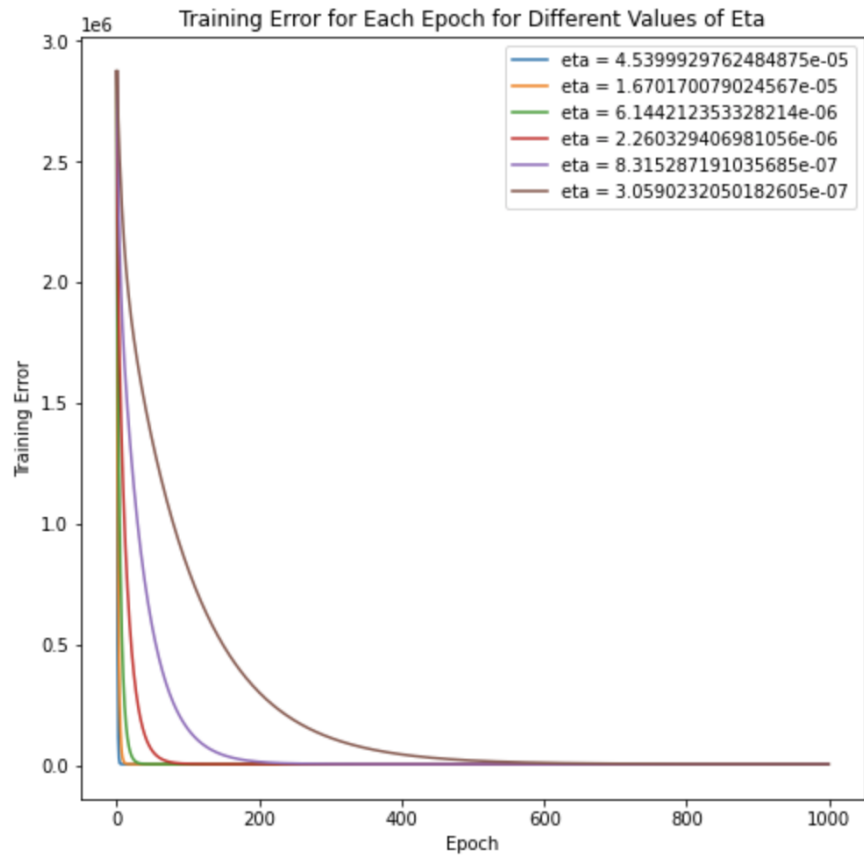
**Solution F:**  $[-0.22790879, -5.97854853, 3.98837868, -11.85702049, 8.91128034]$

**Problem G [2 points]:** Perform SGD as in the previous problem for each learning rate  $\eta$  in

$$\{e^{-10}, e^{-11}, e^{-12}, e^{-13}, e^{-14}, e^{-15}\},$$

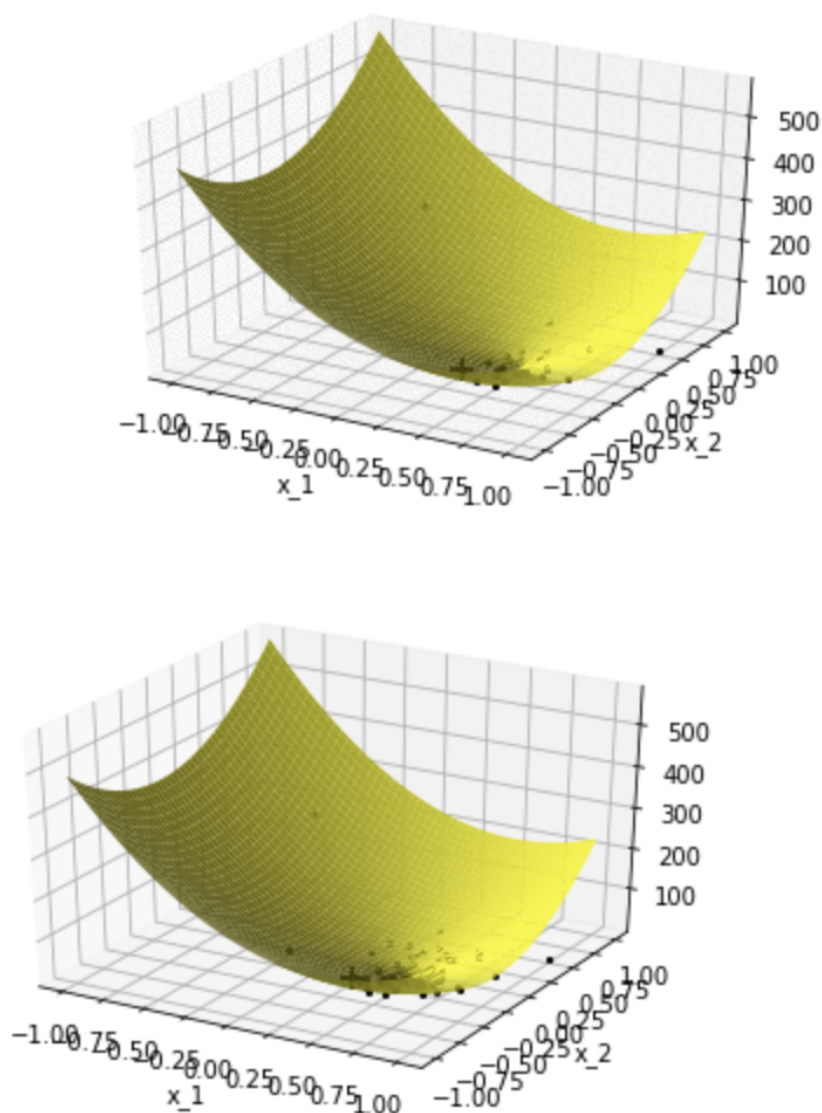
and calculate the training error at the beginning of each epoch during training. On a single plot, show training error vs. number of epochs trained for each of these values of  $\eta$ . Explain what is happening.





**Solution G:**

*As  $\eta$  increases, the training error decreases to 0 in less epochs.*



**Problem H [2 points]:** The closed form solution for linear regression with least squares is

$$\mathbf{w} = \left( \sum_{i=1}^N \mathbf{x}_i \mathbf{x}_i^T \right)^{-1} \left( \sum_{i=1}^N \mathbf{x}_i y_i \right).$$

Compute this analytical solution. Does the result match up with what you got from SGD?

**Solution H:** *Analytical solution:  $[-0.31644251, -5.99157048, 4.01509955, -11.93325972, 8.99061096]$   
This result is similar as what I got from SGD but not exactly the same.*

Answer the remaining questions in 1-2 short sentences.

**Problem I [2 points]:** Is there any reason to use SGD when a closed form solution exists?

**Solution I:** *If the dataset is very large or has high dimension, it can be computationally expensive to compute the closed form solution. We can also compute SGD with parallel computing which is fast.*

**Problem J [2 points]:** Based on the SGD convergence plots that you generated earlier, describe a stopping condition that is more sophisticated than a pre-defined number of epochs.

**Solution J:** *Instead of pre-defining a number of epochs, we can stop the SGD process when the validation/training error is close to 0, or smaller than a certain fixed value.*

**Problem K [2 points]:** How does the convergence behavior of the weight vector differ between the perceptron and SGD algorithms?

**Solution K:** *The weight vector fluctuates depending on the choice of the misclassified point as it converges. In SGD, however, the weight vector smoothly converges to a point if  $\eta$  is not too large.*

## 4 The Perceptron [14 Points]

*Relevant materials: lecture 2*

The perceptron is a simple linear model used for binary classification. For an input vector  $\mathbf{x} \in \mathbb{R}^d$ , weights  $\mathbf{w} \in \mathbb{R}^d$ , and bias  $b \in \mathbb{R}$ , a perceptron  $f : \mathbb{R}^d \rightarrow \{-1, 1\}$  takes the form

$$f(\mathbf{x}) = \text{sign} \left( \left( \sum_{i=1}^d w_i x_i \right) + b \right)$$

The weights and bias of a perceptron can be thought of as defining a hyperplane that divides  $\mathbb{R}^d$  such that each side represents an output class. For example, for a two dimensional dataset, a perceptron could be drawn as a line that separates all points of class +1 from all points of class -1.

The PLA (or the Perceptron Learning Algorithm) is a simple method of training a perceptron. First, an initial guess is made for the weight vector  $\mathbf{w}$ . Then, one misclassified point is chosen arbitrarily and the  $\mathbf{w}$  vector is updated by

$$\begin{aligned} \mathbf{w}_{t+1} &= \mathbf{w}_t + y(t)\mathbf{x}(t) \\ b_{t+1} &= b_t + y(t), \end{aligned}$$

where  $\mathbf{x}(t)$  and  $y(t)$  correspond to the misclassified point selected at the  $t^{\text{th}}$  iteration. This process continues until all points are classified correctly.

The following few problems ask you to work with the provided Jupyter notebook for this problem, titled `4_notebook.ipynb`. This notebook utilizes the file `perceptron_helper.py`, but you should not need to modify this file.

**Problem A [8 points]:** The graph below shows an example 2D dataset. The + points are in the +1 class and the  $\circ$  point is in the -1 class.

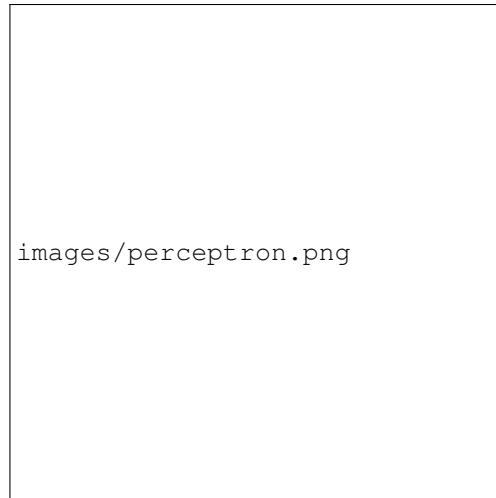


Figure 1: The green + are positive and the red o is negative

Implement the `update_perceptron` and `run_perceptron` methods in the notebook, and perform the perceptron algorithm with initial weights  $w_1 = 0, w_2 = 1, b = 0$ .

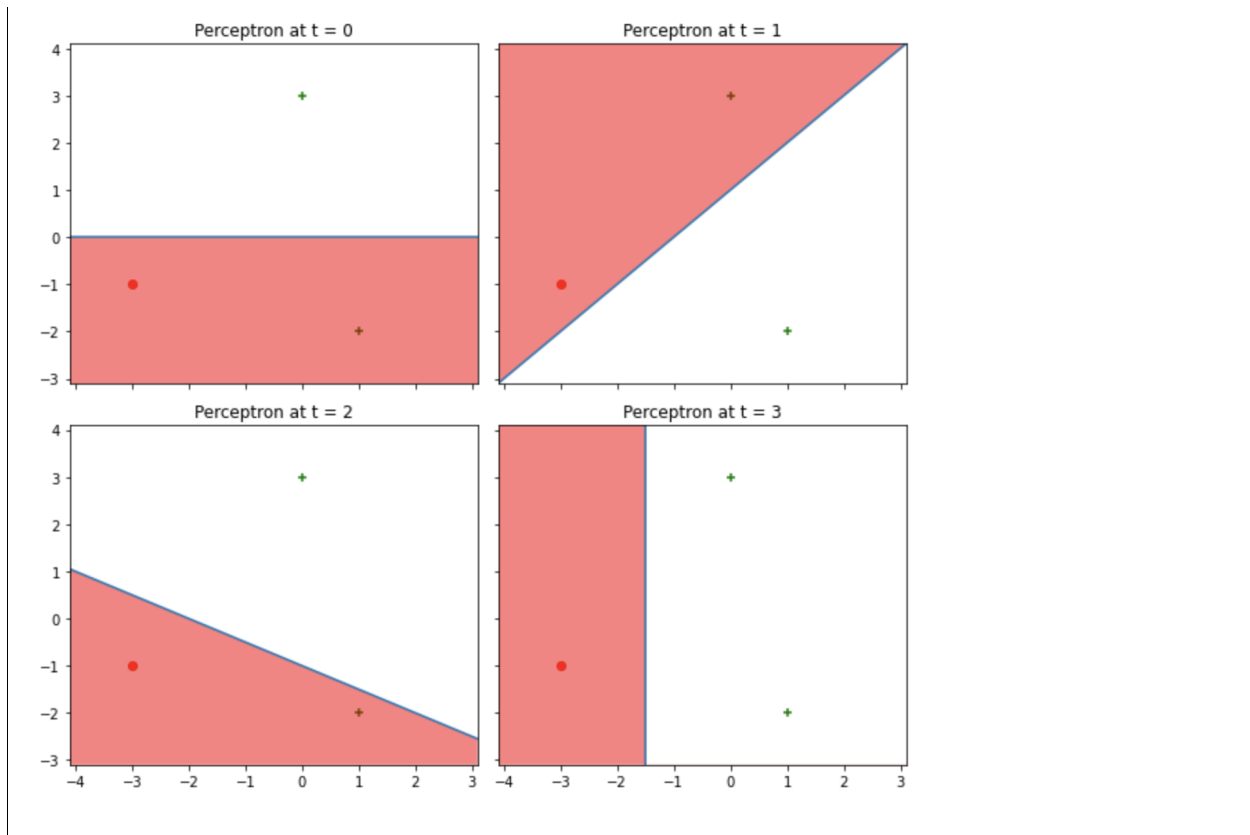
Give your solution in the form a table showing the weights and bias at each timestep and the misclassified point  $([x_1, x_2], y)$  that is chosen for the next iteration's update. You can iterate through the three points in any order. Your code should output the values in the table below; cross-check your answer with the table to confirm that your perceptron code is operating correctly.

$t$	$b$	$w_1$	$w_2$	$x_1$	$x_2$	$y$
0	0	0	1	1	-2	+1
1	1	1	-1	0	3	+1
2	2	1	2	1	-2	+1
3	3	2	0			

Include in your report both: the table that your code outputs, as well as the plots showing the perceptron's classifier at each step (see notebook for more detail).

**Solution A:**

```
t b w1 w2 x1 x2 y
0 0.0 0.0 1.0 1 -2 1
1 1.0 1.0 -1.0 0 3 1
2 2.0 1.0 2.0 1 -2 1
3 3.0 2.0 0.0
Plot below.
```



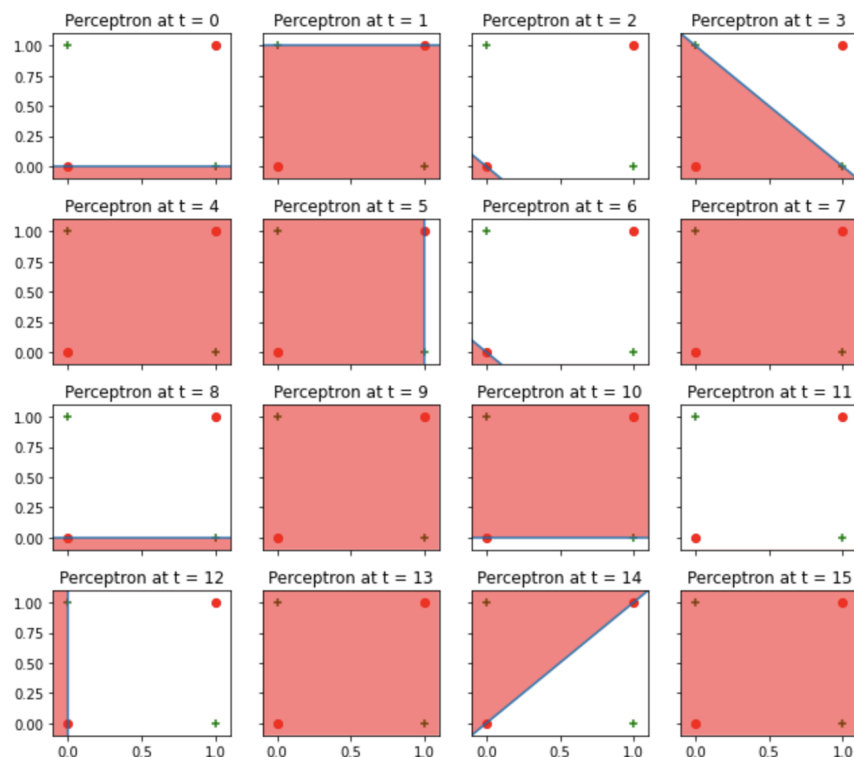
**Problem B [4 points]:** A dataset  $S = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\} \subset \mathbb{R}^d \times \mathbb{R}$  is *linearly separable* if there exists a perceptron that correctly classifies all data points in the set. In other words, there exists a hyperplane that separates positive data points and negative data points.

In a 2D dataset, how many data points are in the smallest dataset that is not linearly separable, such that no three points are collinear? How about for a 3D dataset such that no four points are coplanar? Please limit your solution to a few lines - you should justify but not prove your answer.

Finally, how does this generalize for an  $N$ -dimensional set, in which **no**  $<N$ -dimensional hyperplane contains a non-linearly-separable subset? For the  $N$ -dimensional case, you may state your answer without proof or justification.

**Solution B:** In a 2D dataset, 4 data points are in the smallest dataset that is not linearly separable in which no three points are collinear. We can choose any rectangle and place four points at the vertices, with vertices diagonal to each other having the same classification. For a 3D dataset, 5 points are in the smallest dataset that is not linearly separable in which no 4 points are coplanar. We can have 3 points on the same plane dividing 2 points that are different classifications. There would be no plane to separate these 5 points. For the  $N$ -dimensional set, a dataset of  $N+2$  points is the smallest that is not linearly separable.

**Problem C [2 points]:** Run the visualization code in the Jupyter notebook section corresponding to question C (report your plots). Assume a dataset is *not* linearly separable. Will the Perceptron Learning Algorithm ever converge? Why or why not?



**Solution C:**

*The Perceptron Learning Algorithm will never converge. The Perceptron Learning Algorithm converges when there are no more misclassified points. However, the data is not linearly separable which means that there will always be misclassified points. Thus, the algorithm will run in an infinite loop of choosing a misclassified point and updating the weight and bias.*

Code links:

[Problem 2](#)

[Problem 3 Part 1](#)

[Problem 3 Part 2](#)

[Problem 4](#)