

Policies

- You are free to collaborate on all of the problems, subject to the collaboration policy stated in the syllabus.
- Please submit your report as a single .pdf file to Gradescope under “Homework 1” or “Homework 1 Corrections”. **In the report, include any images generated by your code along with your answers to the questions.** For instructions specifically pertaining to the Gradescope submission process, see https://www.gradescope.com/get_started#student-submission.
- Please submit your code as a .zip archive to Gradescope under “Homework 1 Code” or “Homework 1 Code Corrections”. The .zip file should contain your code files. Submit your code either as Jupyter notebook .ipynb files or .py files.

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 \mathcal{h} ?

Solution A: *The hypothesis set is the space of all possible functions that the model can fit to with its given architecture.*

Correct or equivalent solution

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

Solution B: *The hypothesis set of a linear model is the space of all possible linear functions.*

Correct or equivalent solution

Problem C [2 points]: What is overfitting?

Solution C: *Overfitting is when a model fits to a given dataset too closely, such that now it is tuned for the specific dataset and not the general behavior of the whole dataset and unseen data.*

Correct or equivalent solution

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

Solution D: *You can prevent overfitting by simplifying the model so that it has less parameters to fit to the data, or use more data so that its more difficult to fit accurately to the entire dataset. Using fewer parameters simplifies the hypothesis set so that the model has less freedom to fit to the training data.*

Correct or equivalent solution

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 that one uses to train the model and update its parameters on, while test data is the data that one uses to evaluate the model's performance on. You should never change the model based on results of the test data because then it becomes training data, and the model begins to fit to the test dataset which will always be incorrectly more accurate than to real unseen data.*

Correct or equivalent solution

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

Solution F:

1. *The data is sampled independently and identically distributed.*
2. *The data is representative of the unseen data it would be evaluating.*

Correct or equivalent solution

Problem G [2 points]: Consider the machine learning problem of classifying neutrino interaction events as in slide 14 of lecture 1. What could X , the input space, be? What could Y , the output space, be?

Solution G: *The input space X could be features of the raw data gathered from the detector and the output space Y could be a class of the interaction type.*

Correct or equivalent solution

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

Solution H: *k -fold cross-validation is when one splits a dataset into k equally sized subsets and trains on $k - 1$ of the subsets and evaluates on the remaining subset. This is repeated k times so that each subset is used as the test set once.*

Correct or equivalent solution

2 Bias-Variance Tradeoff [39 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 in the input space \mathcal{X} ,

$$\mathbb{E}_S [E_{\text{out}}(f_S)] = \mathbb{E}_x [\text{Bias}(x) + \text{Var}(x)] \quad (1)$$

given that $F(x)$ is the “average function” over all possible datasets S

$$F(x) = \mathbb{E}_S [f_S(x)] \quad (2)$$

the out-of-sample error for a particular trained model is

$$E_{\text{out}}(f_S) = \mathbb{E}_x [(f_S(x) - y(x))^2] \quad (3)$$

and we define the bias for a given data sample $\text{Bias}(x)$ by how much the average function deviates from the target function

$$\text{Bias}(x) = (F(x) - y(x))^2 \quad (4)$$

and the variance for a given data sample $\text{Var}(x)$ measures

$$\text{Var}(x) = \mathbb{E}_S [(f_S(x) - F(x))^2] \quad (5)$$

Solution A: We start from the out-of-sample squared error of a model trained on a random dataset S :

$$E_{\text{out}}(f_S) = \mathbb{E}_x [(f_S(x) - y(x))^2].$$

We want its expectation over datasets S :

$$\mathbb{E}_S [E_{\text{out}}(f_S)] = \mathbb{E}_S \mathbb{E}_x [(f_S(x) - y(x))^2]$$

Expand the squared term by adding and subtracting the average predictor $F(x) = \mathbb{E}_S [f_S(x)]$

$$(f_S(x) - y(x))^2 = (f_S(x) - F(x) + F(x) - y(x))^2$$

$$(f_S(x) - F(x) + F(x) - y(x))^2 = (f_S(x) - F(x))^2 + 2(f_S(x) - F(x))(F(x) - y(x)) + (F(x) - y(x))^2$$

By definition of $F(x)$, $\mathbb{E}_S [f_S(x) - F(x)] = 0$, so the cross term vanishes.

$$2(f_S(x) - F(x))(F(x) - y(x)) = 0$$

So now we are left with

$$(f_S(x) - y(x))^2 = (f_S(x) - F(x))^2 + (F(x) - y(x))^2$$

Taking expectation over x gives

$$\mathbb{E}_S[(f_S(x) - F(x))^2] = \mathbb{E}_S[(f_S(x) - F(x))^2] + \mathbb{E}_S[(F(x) - y(x))^2]$$

and we know that

$$\mathbb{E}_S[(f_S(x) - F(x))^2] = \text{Var}(x)$$

$$\mathbb{E}_S[(F(x) - y(x))^2] = \text{Bias}(x)$$

so we have:

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

which is the desired bias-variance decomposition.

Correct or equivalent solution

Problem B [5 points]: When there is noise in the data, the out-of-sample error is

$$E_{\text{out}}(f_S) = \mathbb{E}_{x,z}[(f_S(x) - z(x))^2] \quad (6)$$

where $z(x) = y(x) + \epsilon$. If ϵ is a Gaussian-distributed random variable with mean of zero and variance σ^2 , show that the bias-variance decomposition becomes

$$\mathbb{E}_S[E_{\text{out}}(f_S)] = \mathbb{E}_x[\text{Bias}(x) + \text{Var}(x)] + \sigma^2 \quad (7)$$

Hint: Given the mean of ϵ is zero,

$$\mathbb{E}_{x,z}[\epsilon] = \mathbb{E}_{x,\epsilon}[\epsilon] = \mathbb{E}_\epsilon[\epsilon] = 0 \quad (8)$$

Likewise,

$$\mathbb{E}_{x,z}[\epsilon^2] = \mathbb{E}_\epsilon[\epsilon^2] = \mathbb{E}_\epsilon[(\epsilon - \mathbb{E}_\epsilon[\epsilon])^2] = \sigma^2 \quad (9)$$

by the definition of variance.

Solution B: When there is noise in the data, we assume that the observed target is

$$z(x) = y(x) + \epsilon,$$

where ϵ is a Gaussian random variable with mean 0 and variance σ^2 .

The out-of-sample error of a model trained on a random dataset S is then

$$E_{\text{out}}(f_S) = \mathbb{E}_{x,z}[(f_S(x) - z(x))^2].$$

Substituting $z(x) = y(x) + \epsilon$ gives

$$E_{\text{out}}(f_S) = \mathbb{E}_{x,\epsilon}[(f_S(x) - y(x) - \epsilon)^2].$$

Taking the expectation over datasets S , we have

$$\mathbb{E}_S[E_{out}(f_S)] = \mathbb{E}_S \mathbb{E}_{x, \epsilon} [(f_S(x) - y(x) - \epsilon)^2].$$

Expanding the squared term:

$$(f_S(x) - y(x) - \epsilon)^2 = (f_S(x) - y(x))^2 - 2\epsilon(f_S(x) - y(x)) + \epsilon^2.$$

Taking the expectation with respect to ϵ , and using $\mathbb{E}_\epsilon[\epsilon] = 0$ and $\mathbb{E}_\epsilon[\epsilon^2] = \sigma^2$, we get

$$\mathbb{E}_\epsilon[(f_S(x) - y(x) - \epsilon)^2] = (f_S(x) - y(x))^2 + \sigma^2.$$

Therefore,

$$\mathbb{E}_S[E_{out}(f_S)] = \mathbb{E}_S \mathbb{E}_x [(f_S(x) - y(x))^2] + \sigma^2.$$

The first term is the same as in the noiseless case. Expanding it by adding and subtracting the average predictor $F(x) = \mathbb{E}_S[f_S(x)]$, we have

$$(f_S(x) - y(x))^2 = (f_S(x) - F(x))^2 + 2(f_S(x) - F(x))(F(x) - y(x)) + (F(x) - y(x))^2.$$

Since $\mathbb{E}_S[f_S(x) - F(x)] = 0$, the cross term vanishes, and we are left with

$$\mathbb{E}_S[(f_S(x) - y(x))^2] = \mathbb{E}_S[(f_S(x) - F(x))^2] + (F(x) - y(x))^2.$$

Recognizing these as the variance and bias terms, respectively,

$$\mathbb{E}_S[(f_S(x) - F(x))^2] = \text{Var}(x), \quad (F(x) - y(x))^2 = \text{Bias}(x),$$

we obtain

$$\mathbb{E}_S[E_{out}(f_S)] = \mathbb{E}_x[\text{Bias}(x) + \text{Var}(x)] + \sigma^2.$$

Thus, when noise is present, the bias-variance decomposition becomes

$$\boxed{\mathbb{E}_S[E_{out}(f_S)] = \mathbb{E}_x[\text{Bias}(x) + \text{Var}(x)] + \sigma^2,}$$

where σ^2 is the irreducible error introduced by the data noise.

Correct or equivalent solution

Problem C [14 points]: 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.

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 .

Solution C:

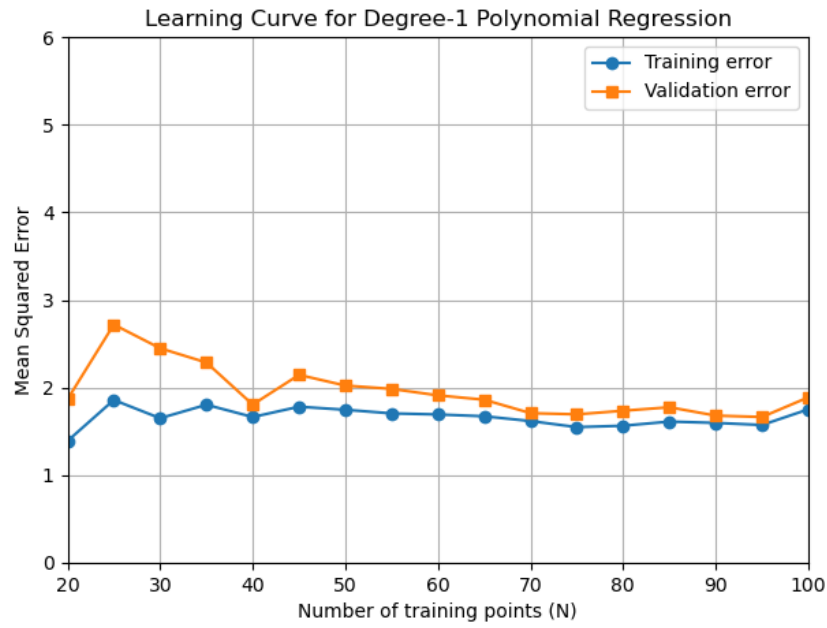


Figure 1: Learning curve for degree-2 polynomial regression

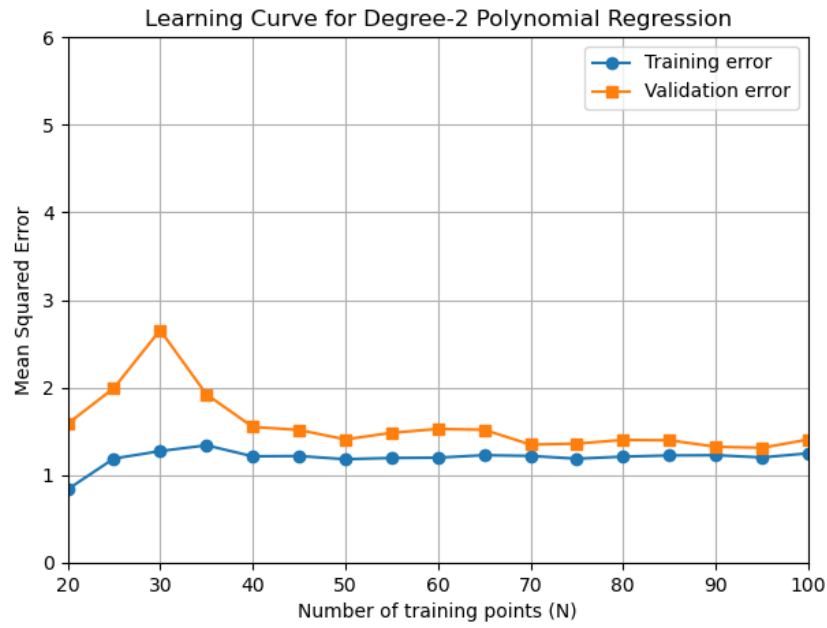


Figure 2: Learning curve for degree-2 polynomial regression

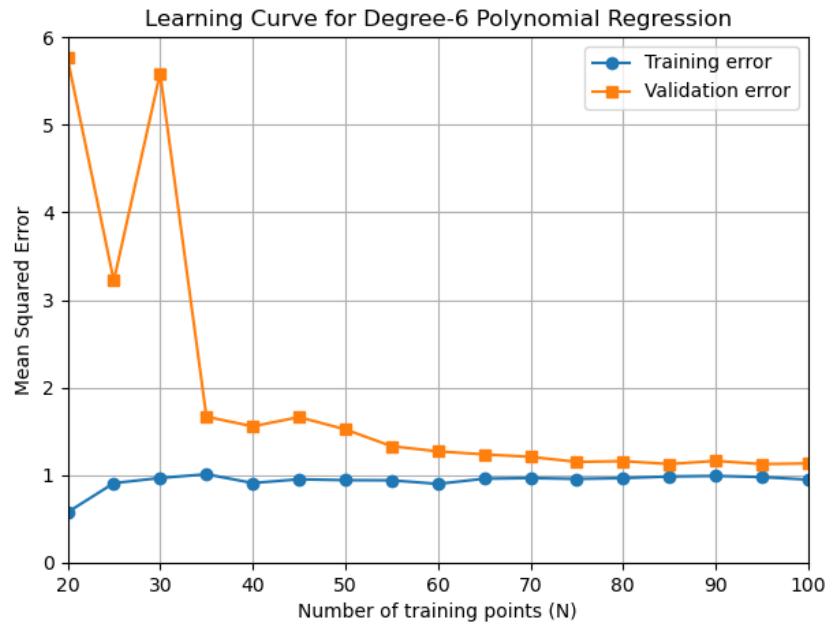


Figure 3: Learning curve for degree-6 polynomial regression

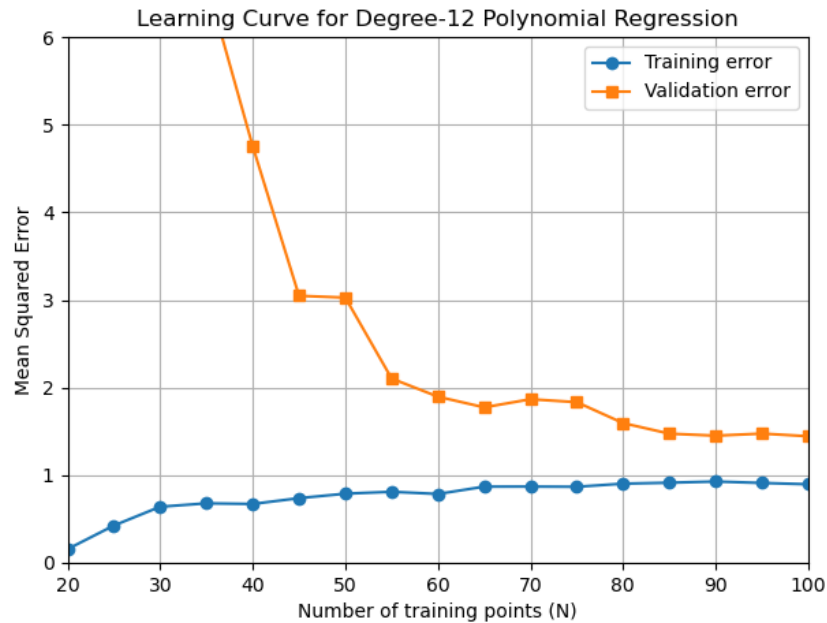


Figure 4: Learning curve for degree-12 polynomial regression

Corrections:

The following corrections were made to avoid numerical instability issues:

- Starting N at 20:** The range of training points now starts at $N=20$ instead of $N=5$. This ensures that even with 5-fold cross-validation, each training fold has at least 16 data points.
- Y-axis limits:** Added `plt.ylim(0, 6)` to clip the y-axis range, making the plots readable even when high-degree polynomials (especially degree 12) produce very large validation errors at small N values.
- Reason for changes:** With very small N values (like $N=5$ or $N=10$), 5-fold cross-validation leaves too few points in each training fold for high-degree polynomials. For example:
 - At $N=15$ with 5-fold CV: 12 training points per fold
 - Degree 12 polynomial needs 13 parameters
 - This creates an underdetermined system leading to numerical instability and massive errors (in the millions/billions)

By starting at $N=20$, we ensure sufficient training data for all polynomial degrees tested.

Problem D [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 D: *The linear model has the highest bias. We can tell because the training error is the highest and the validation error is the lowest. θ is small.*

Correct or equivalent solution

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

Solution E: *The 12th degree model has the highest variance. We can tell because the training error is very low but the validation error is very high. it is overfitting to the training data and not generalizing to the validation data.*

Correct or equivalent solution

Problem F [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 F: *The quadratic model will improve by a little bit as we add more training points but there are diminishing returns.*

Correct or equivalent solution

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

Solution G: *Because the model is training on the training data and actively adjusting to fit it.*

Correct or equivalent solution

Problem H [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 H: *The quadratic model would perform best on some unseen data drawn from the same distribution as the training data. Because it has the lowest bias and variance meaning the lowest average error. **Correct or equivalent solution***

3 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 perceptron learning algorithm (PLA) 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^{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 `3_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.

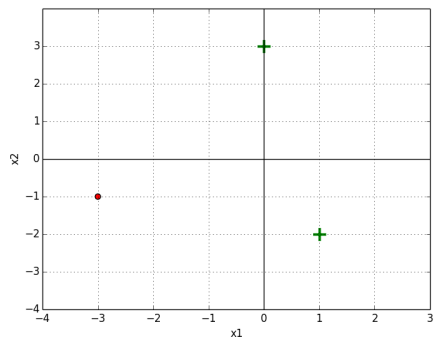


Figure 5: The green + are positive and the red \circ is negative

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 time step 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:

Python Output:

bias: 1.0, weights: [1. -1.], point: [1 -2], label: 1

bias: 2.0, weights: [1. 2.], point: [0 3], label: 1

bias: 3.0, weights: [2. 0.], point: [1 -2], label: 1

final w = [2. 0.], final b = 3.0

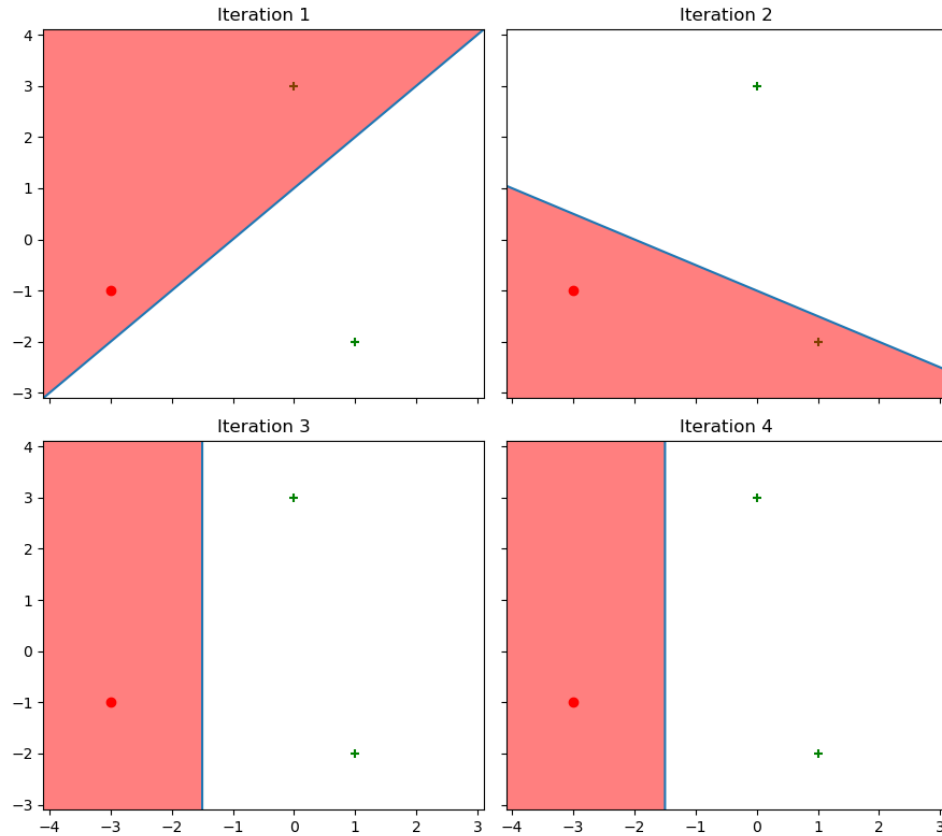


Figure 6: Perceptron plot

Correct or equivalent solution

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: *for a 2D dataset, the smallest dataset that is not linearly separable is 4 points. for a 3D dataset, the smallest dataset that is not linearly separable is 5 points. for an N -dimensional dataset, the smallest dataset that is not linearly separable is $N + 2$ points.*

For a 2D dataset, if you arrange 3 points such that they are not collinear, then you can always separate two classes with a line because they form a triangle. Therefore the smallest dataset that is not linearly separable is 4 points. This generalizes to higher dimensions.

Correct or equivalent solution

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 PLA ever converge? Why or why not?

Solution C:

The PLA will never converge because the dataset is not linearly separable.

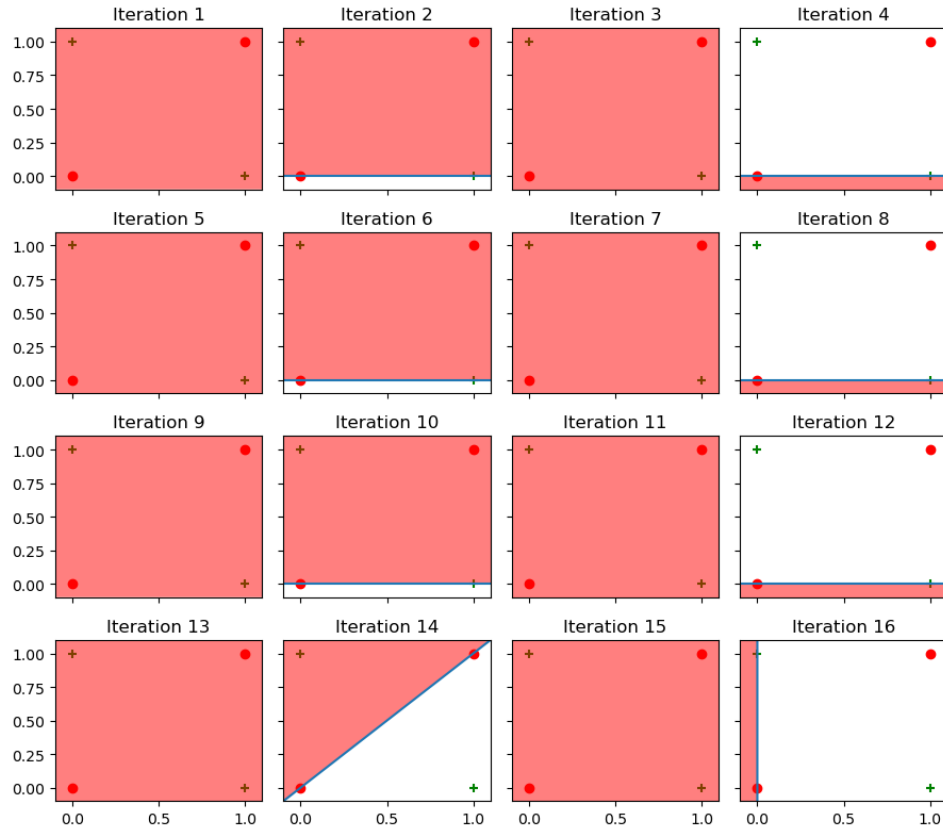


Figure 7: Perceptron plot

Correct or equivalent solution

4 TensorFlow Playground [14 Points]

The purpose of this problem is to build some intuition around linear models and neural networks.

Problem A [6 points]: Navigate your web browser to the TensorFlow Playground (<https://playground.tensorflow.org/#networkShape=&dataset=xor&discretize=true>). Select the checkerboard pattern for the data, which is known as the XOR dataset. The data is sampled from a 2D probability density distribution represented by (x_1, x_2) . The regions $x_1, x_2 > 0$ and $x_1, x_2 < 0$ have a target value $y = +1$ and are shown as blue data points, while the regions $x_1 > 0, x_2 < 0$ and $x_1 < 0, x_2 > 0$ have a target value of $y = -1$ and are shown as orange data points.

First, select a linear model with no hidden layers. For the features, select the two independent variables x_1 and x_2 . Can you fit the data with this linear model? Why or why not? What happens if you add the feature $x_1 x_2$?

Now, return the features to just (x_1, x_2) and start adding hidden layers. What's the smallest neural network (least number of layers and least number of neurons per layer) you can create that fits the training data "perfectly" (i.e. a training loss < 0.001)? What is the corresponding test loss? Detail your hyperparameter choices by providing a screenshot and the URL to your solution (the URL contains all your settings choices).

Solution A: *You cannot separate the data with a linear model because it can only separate data with a hyperplane. This is analogous to the previous question where you have a different class to the adjacent one on each corner of the square.*

*Once you add $x_1 * x_2$ as a feature, you can separate the data with a linear model. Because $x_1 * x_2$ is a non-linear feature.*

The smallest neural network that can fit the data is a 1 hidden layer network with 5 neurons.

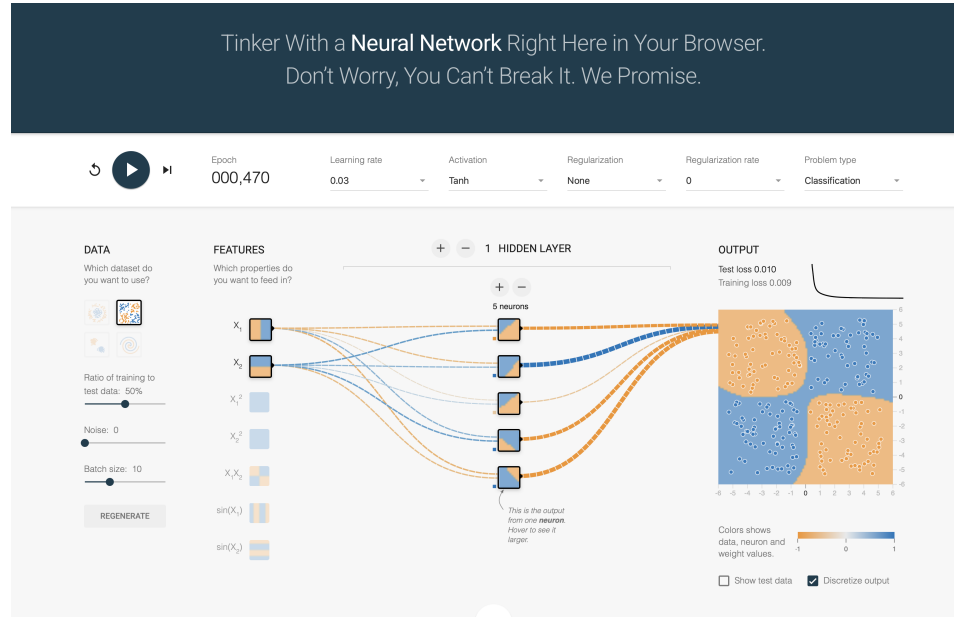


Figure 8: Linear model

Link: <https://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=xor®Dataset=reg-plane&learningRate=0.03®ularizationRate=0&noise=0&networkShape=5&seed=0.09614&showTestData=false&discretize=true&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false>

Correct or equivalent solution

Problem B [8 points]: Navigate your web browser to the TensorFlow Playground <https://playground.tensorflow.org/#dataset=spiral&discretize=true>. Select the spiral pattern for the data.

Using all the features available, find a solution that fits the training data. What training and test error do you achieve? Is this a low bias/high variance or high bias/low variance model? How do you know?

Using only (x_1, x_2) , find a solution that fits the training data. What training and test error do you achieve? Is this a low bias/high variance or high bias/low variance model? How do you know?

For both solutions, detail your hyperparameter choices by providing a screenshot and the URL to your solution (the URL contains all your settings choices).

Bonus: Can you find two engineered features that would allow you to find a solution with a linear model?

Solution B:

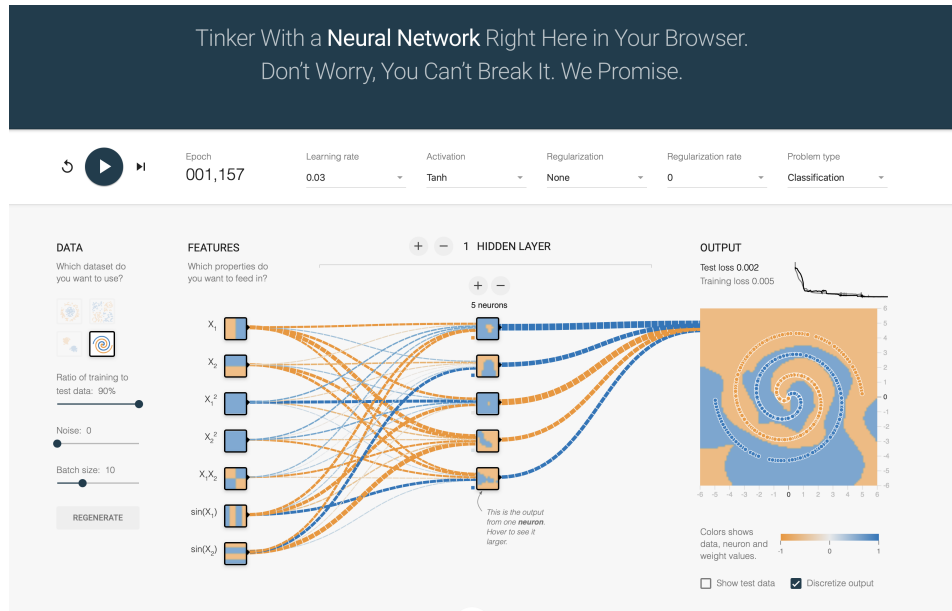


Figure 9: Spiral model

Link: <https://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=spiral®Dataset=reg-plane&learningRate=0.03®ularizationRate=0&noise=0&networkShape=5&seed=0.69944&showTestData=false&discretize=true&percTrainData=90&x=true&y=true&xTimesY=true&xSquared=true&ySquared=true&cosX=false&sinX=true&cosY=false&sinY=true&collectStats=false&problem=classification&initZero=false&hideText=false>

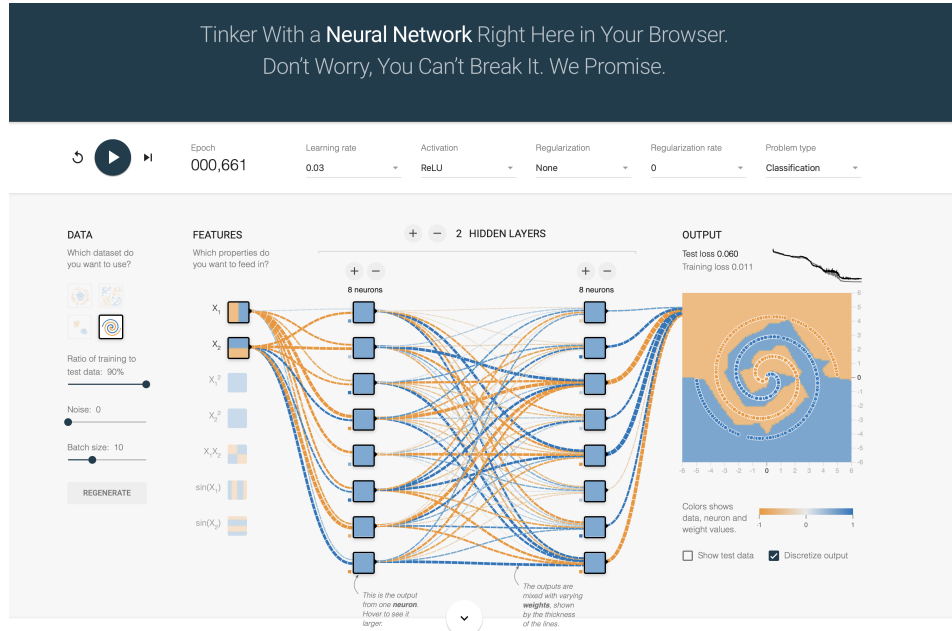


Figure 10: Spiral model

Link: <https://playground.tensorflow.org/#activation=relu&batchSize=10&dataset=spiral®Dataset=reg-plane&learningRate=0.03®ularizationRate=0&noise=0&networkShape=8,8&seed=0.69944&showTestData=false&discretize=true&percTrainData=90&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false>

This first model has higher bias and lower variance, and the second model has lower bias and higher variance. This is because the first model has fewer layers and neurons.

Correct or equivalent solution