Machine Learning in Physics
Homework 1

UCSD PHYS 139/239
Due: Friday, October 10, 2025, 8:00pm
Corrections due: Wednesday, October 15, 2024, 8:00pm

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?

Solution A: *The hypothesis set $\mathcal{H}$ is the set of all hypothesis functions $g : \mathcal{X} \to \mathcal{Y}$ under consideration that map inputs to outputs.*

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 linear functions $f : \mathbb{R}^D \to \mathbb{R}$ where $f(x|w) = w^\intercal x$*

Problem C [2 points]: What is overfitting?

Solution C: *Overfitting refers to the situation when a given model has small training error, but large testing error.*

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

Solution D: *(1) Select model hypotheses with a smaller number of parameters, i.e. opting for lower variance instead of lower bias, and (2) apply regularization, such as L_1 regularization, also called a lasso regression, which adds the absolute value of the weights as a penalty term to the loss function. Additional acceptable answers include: (3) train with a larger dataset, (4) augment the dataset by generating additional data points by perturbing existing data, (5) train with k-fold cross validation, (6) apply dropout (to neural networks), (7) use early stopping to pause the model's training when training error and validation error begin to diverge, (8) aggregate many models into a large ensemble method, etc.*

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 used to train a model and test data is used to evaluate the final performance of a model (i.e. see how well it generalizes to unseen data) You should never change your model based on information from test data because then you are effectively using the test data to train your model (e.g., if you update your*

model architecture continuously) This defeats the purpose of 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 data samples are statistically independent and identically distributed (i.i.d.).*

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 could consist of 2D or 3D images with 1 channel of information corresponding to the charge deposited. This means the input space could be either $\mathbb{R}^3$ (x, y, q) or $\mathbb{R}^4$ (x, y, z, q). The output space could be $\{0, 1, 2\}$ to indicate which of the 3 pictured classes the event belongs to. Alternatively, the output space could be $\mathbb{R}^3$ (technically $[0, 1]^3$) where each dimension corresponds to the probability for the input to belong to that class. Note: any of the answers above is acceptable (as well as other reasonable variations).*

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

Solution H: *k-fold cross validation involves partitioning the data into k equally sized, non-overlapping partitions. We train on $k - 1$ partitions and evaluate on the remaining partition. The cross validation procedure is then repeated k times with each of the k partitions used exactly once as the validation data. This allows for "re-using" of validation data as training data.*

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 $\mathcal{X}$,

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

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

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

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

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

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

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

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

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

Solution A: *Taking the expected value of the out-of-sample error, we find*

$$\begin{aligned}
\mathbb{E}_S\left[E_{out}(f_S)\right] &= \mathbb{E}_S\mathbb{E}_x\left[\left(f_S(x) - y(x)\right)^2\right] &(6)\\
&= \mathbb{E}_x\mathbb{E}_S\left[\left(f_S(x) - y(x)\right)^2\right] &(7)\\
&= \mathbb{E}_x\left[\mathbb{E}_S[f_S(x)^2] - 2\mathbb{E}_S[f_S(x)]y(x) - y(x)^2\right] &(8)
\end{aligned}$$

because we can interchange the order of the expected value operators by Fubini's theorem and expand the square. Now, we can rewrite this in terms of the average function $F(x)$, and use the trick of adding and subtracting the same quantity $F(x)^2$

$$\begin{aligned}
\mathbb{E}_S\left[E_{out}(f_S)\right] &= \mathbb{E}_x\left[\mathbb{E}_S[f_S(x)^2] - 2F(x)y(x) + y(x)^2\right] &(9)\\
&= \mathbb{E}_x\big[\mathbb{E}_S[f_S(x)^2] - F(x)^2 + \underbrace{F(x)^2 - 2F(x)y(x) + y(x)^2}_{\left(F(x) - y(x)\right)^2}\big] &(10)
\end{aligned}$$

The right term is equal to $\text{Bias}(x)$*, but the left term does not quite look like the* $\text{Var}(x)$ *term we expect. We can*

use similar trick of $-F(x)^2 = -2F(x)^2 + F(x)^2$ and the definition of $F(x) = \mathbb{E}_S[f_S(x)]$

$$\begin{aligned}\mathbb{E}_S[f_S(x)^2] - F(x)^2 &= \mathbb{E}_S[f_S(x)^2] - 2F(x)^2 + F(x)^2 &(11)\\ &= \mathbb{E}_S[f_S(x)^2] - 2\mathbb{E}_S[f_S(x)]F(x) + F(x)^2 &(12)\\ &= \mathbb{E}_S[f_S(x)^2 - 2f_S(x)F(x) + F(x)^2] &(13)\\ &= \mathbb{E}_S[(f_S(x)^2 - F(x))^2] &(14)\end{aligned}$$

We also use the fact that $\mathbb{E}_S[F(x)] = F(x)$ (because the average function doesn't depend on a specif a dataset S) to freely move $F(x)$ inside the expected value operator. Thus, we arrive at:

$$\begin{aligned}\mathbb{E}_S\left[E_{out}(f_S)\right] &= \mathbb{E}_S[(f_S(x) - F(x))^2] + (F(x) - y(x))^2) &(15)\\ &= \mathrm{Var}(x) + \mathrm{Bias}(x) &(16)\end{aligned}$$

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

$$E_{\text{out}}(f_S) = \mathbb{E}_{x,z}[(f_S(x) - z(x))^2] \tag{17}$$

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

$$\mathbb{E}_S\left[E_{\text{out}}\left(f_S\right)\right] = \mathbb{E}_x[\mathrm{Bias}(x) + \mathrm{Var}(x)] + \sigma^2 \tag{18}$$

Hint: *Given the mean of ϵ is zero,*

$$\mathbb{E}_{x,z}[\epsilon] = \mathbb{E}_{x,\epsilon}[\epsilon] = \mathbb{E}_\epsilon[\epsilon] = 0 \tag{19}$$

Likewise,

$$\mathbb{E}_{x,z}[\epsilon^2] = \mathbb{E}_\epsilon[\epsilon^2] = \mathbb{E}_\epsilon[(\epsilon - \mathbb{E}_\epsilon[\epsilon])^2] = \sigma^2 \tag{20}$$

by the definition of variance.

Solution B: *We start the same way*

$$\begin{aligned}\mathbb{E}_S\left[E_{out}(f_S)\right] &= \mathbb{E}_S\mathbb{E}_{x,z}\left[\left(f_S(x) - z(x)\right)^2\right] &(21)\\ &= \mathbb{E}_{x,z}\mathbb{E}_S\left[\left(f_S(x) - z(x)\right)^2\right] &(22)\\ &= \mathbb{E}_{x,z}\left[\mathbb{E}_S[f_S(x)^2] - 2\mathbb{E}_S[f_S(x)]z(x) + z(x)^2\right] &(23)\\ &= \mathbb{E}_{x,z}\left[\mathbb{E}_S[f_S(x)^2] - 2\mathbb{E}_S[f_S(x)](y(x) + \epsilon) + (y(x) + \epsilon)^2\right] &(24)\\ &= \mathbb{E}_{x,z}\left[\mathbb{E}_S[f_S(x)^2] - 2\mathbb{E}_S[f_S(x)]y(x) - 2\mathbb{E}_S[f_S(x)]\epsilon + y(x)^2 + 2y(x)\epsilon + \epsilon^2\right] &(25)\\ &= \mathbb{E}_{x,z}\left[\mathbb{E}_S[f_S(x)^2] - 2F(x)y(x) - 2F(x)\epsilon + y(x)^2 + 2y(x)\epsilon + \epsilon^2\right] &(26)\end{aligned}$$

At this point, we add and subtract $F(x)$ as before

$$\mathbb{E}_S\left[E_{out}(f_S)\right] = \mathbb{E}_{x,z}\big[\underbrace{\mathbb{E}_S[f_S(x)^2] - F(x)^2}_{\mathrm{Var}(x)} + \underbrace{F(x)^2 - 2F(x)y(x) + y(x)^2}_{\mathrm{Bias}(x)} - 2F(x)\epsilon + 2y(x)\epsilon + \epsilon^2\big] \quad (27)$$

and identify the $\mathrm{Var}(x)$ *and* $\mathrm{Bias}(x)$ *terms from the previous problem. The remaining terms depend on* ϵ*. Following the hint,*

$$-2\mathbb{E}_{x,z}\big[(F(x) - y(x))\epsilon\big] + \mathbb{E}_{x,z}\big[\epsilon^2\big] = -2\mathbb{E}_x[F(x) - y(x)]\underbrace{\mathbb{E}_\epsilon[\epsilon]}_{0} + \underbrace{\mathbb{E}_\epsilon[\epsilon^2]}_{\sigma^2} = \sigma^2 \quad (28)$$

Putting it all together, we have the final result

$$\mathbb{E}_S\left[E_{out}(f_S)\right] = \mathbb{E}_x[\mathrm{Var}(x) + \mathrm{Bias}(x)] + \sigma^2 \quad (29)$$

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 $N \in \{20, 25, 30, 35, \ldots, 100\}$:
    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: *See the code in `2_notebook_sol.ipynb`. The output learning curves for 1st (top left), 2nd (top right), 6th (top left) and 12th (bottom right) degree polynomials are shown in the figure below.*

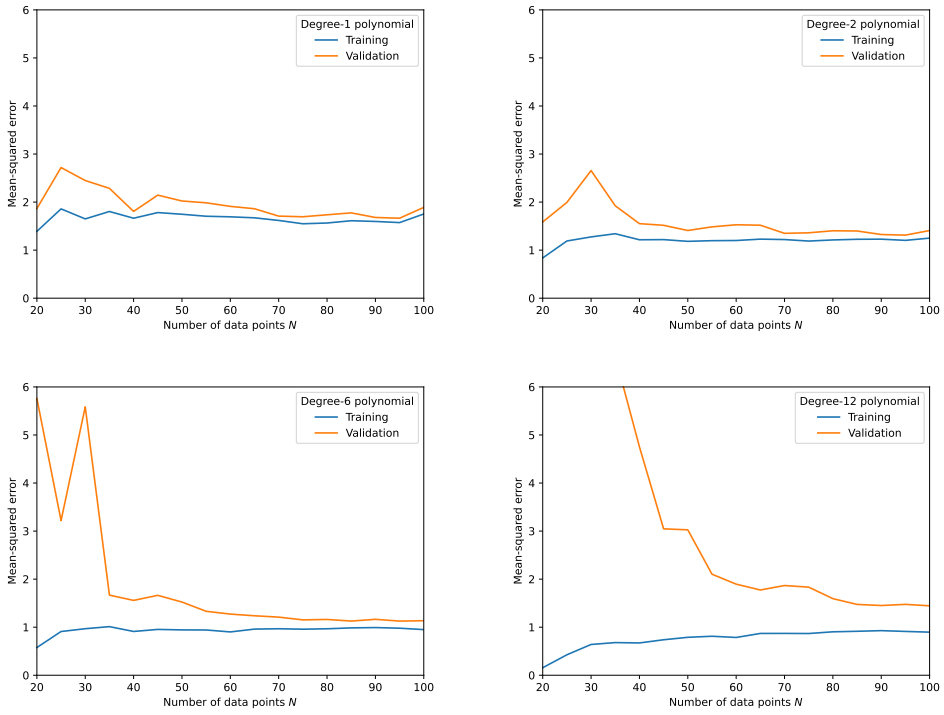

Figure 1: *Learning curves for 1st (top left), 2nd (top right), 6th (top left) and 12th (bottom right) degree polynomials.*

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 degree-1 polynomial has the highest bias, as the mean-squared error loss on the training set is consistently larger for all values of N than the corresponding values for the higher degree polynomials.*

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

Solution E: *The degree-12 polynomial has the highest variance, as the difference between the mean-squared error loss on the training and validation sets is the largest compared to other polynomials, especially at small values of* N*.*

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 training loss for the quadratic model appears to plateau at* $N \approx 40$*. This indicates that the model will not improve with additional training points.*

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

Solution G: *The training error is generally lower than the validation error by construction because the parameters chosen are the ones that minimize the loss on the training data (not the validation data).*

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: *In the large* N *limit, it appears the degree-6 polymomial model would generalize the best to unseen data drawn from the same distribution as the training data because the validation error is the lowest*

---

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 $f : \mathbb{R}^d \to \{-1, 1\}$ 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

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

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.

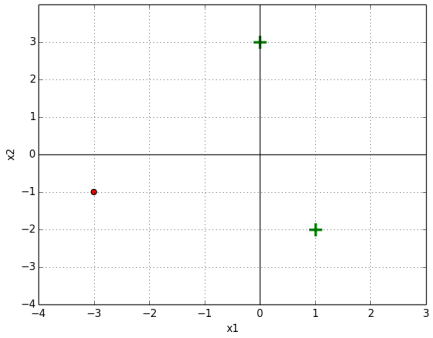

Figure 2: The green $+$ are positive and the red $\circ$ is negative

---

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: *The table our code outputs is:*

```
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
```

The iterations of the perceptron learning algorithm are shown in the figure below.

Problem B [4 points]: A dataset $S = \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)\} \subset \mathbb{R}^d \times \mathbb{R}$ 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: *With only 3 data points in a 2D dataset, you can always construct a line that separates the two classes as long as they are not collinear. Explicitly, we can apply some symmetry transformations to the dataset to construct an equivalent dataset. Suppose we have two positive examples and one negative example. Let's translate and rotate the $(x^{(1)}, x^{(2)})$ plane so that we have one of the positive examples located at the origin $x_1 = (0, 0)$ and the other positive example located along the positive $x^{(1)}$-axis (i.e., its $x^{(2)}$ coordinate is 0). Without loss of generality, we can also rescale the $x^{(1)}$ axis, so this point is located at $x_2 = (1, 0)$. Finally, we can also rescale the $x^{(2)}$ axis so that the*

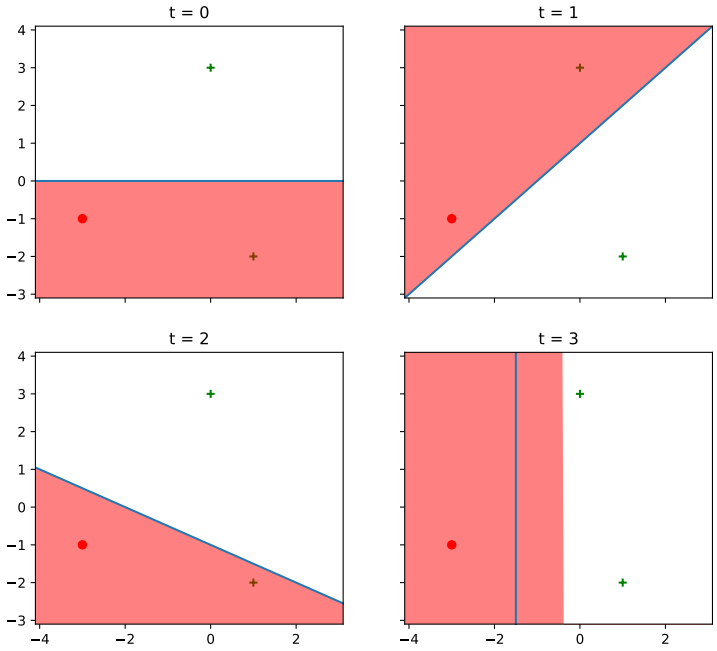

Figure 3: *Perceptron learning algorithm iterations for a linearly separable dataset.*

third data point (negative example) is at either $|x^{(2)}| = 1$*. Now, we have two possibilities for this final data point:* $x_3 = (x_3^{(1)}, \pm 1)$*. In either case, we can take the horizontal line given by* $x^{(2)} = \pm\frac{1}{2}$*, which separates the two classes.*

Thus, you need at least 4 data points in a 2D dataset that is not linearly separable. Such an example dataset is shown in the figure below. It is evident that these 4 data points are not linearly separable because the line connecting the two positive examples and the line connecting the two negative examples intersect.

For a 3D dataset, we can again apply symmetry transformations to explicitly find the plane separating the two classes if there are only 4 data points. Thus, we need at least 5 data points for a dataset that is not linearly separable.

For an N*-dimensional dataset, the corresponding minimum dataset size is* $N + 2$*.*

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 D: *No, the perceptron learning algorithm will never converge because it will cycle through the four states shown in the figure below (e.g., from* $t = 4$ *to* $t = 7$*) over and over.*

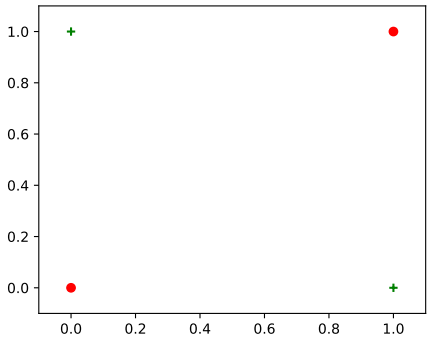

Figure 4: *2D dataset with 4 data points that is not linearly separable.*

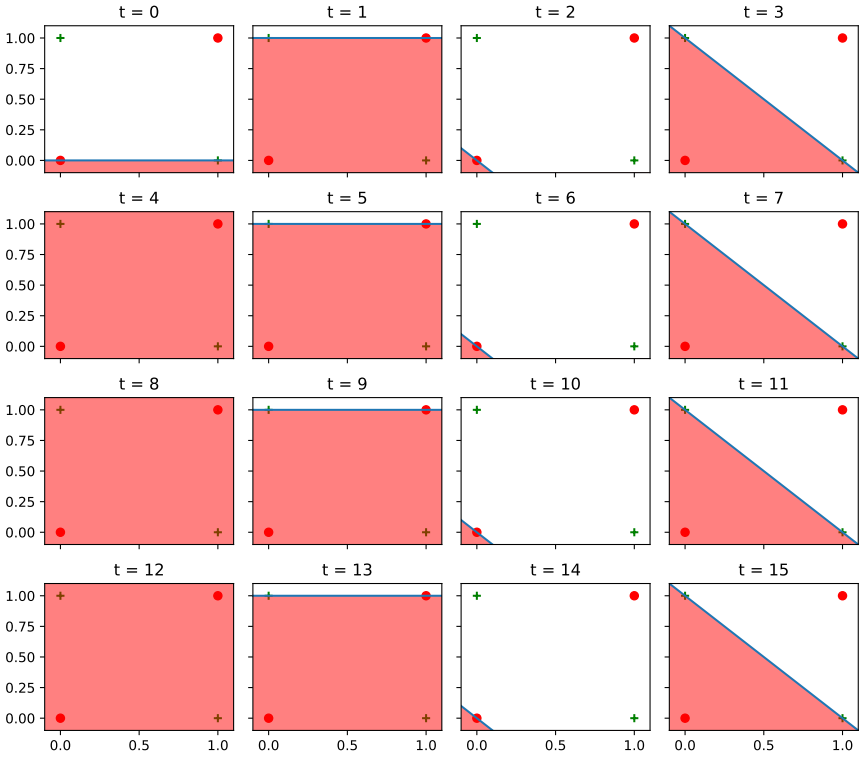

Figure 5: *Perceptron learning algorithm for not linearly separable dataset.*

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: *No, we cannot fit this dataset with a linear model because it is not linearly separable. See the URL (https://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=xor&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=&seed=0.42&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) and figure below.*

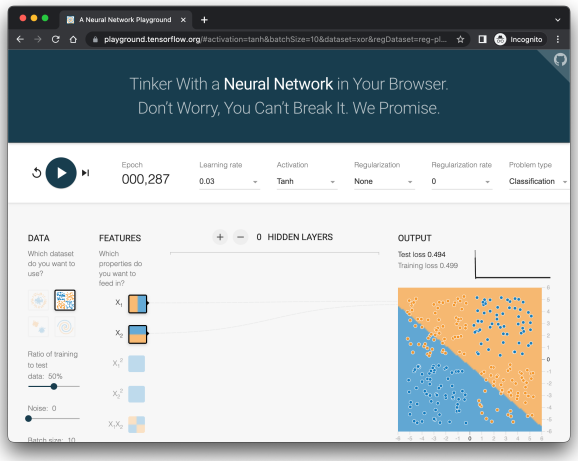

Figure 6: XOR dataset with a linear model.

If we add the feature $x_1 x_2$, we can easily fit this data with a linear model (training loss <0.001). See the URL (https://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=xor&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=&seed=0.42&showTestData=false&discretize=true&percTrainData=50&x=true&y=true&xTimesY=true&

xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false) and figure below.

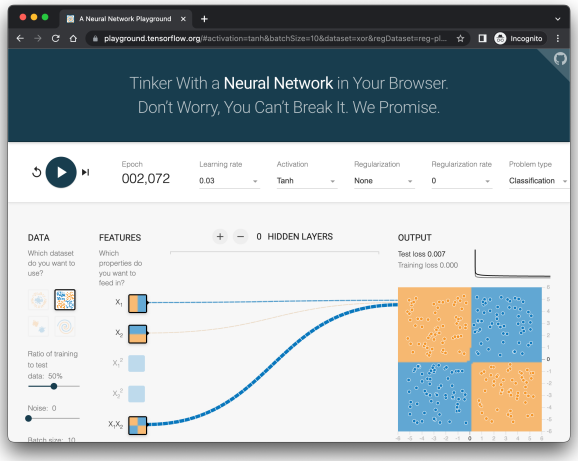

Figure 7: XOR dataset with $x_1 x_2$ as an input and a linear model.

"Perfectly" is certainly debatable, but you can get close with 1 hidden layer with 3 neurons and ReLU activation function. See the URL: https://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=xor&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=&seed=0.42&showTestData=false&discretize=true&percTrainData=50&x=true&y=true&xTimesY=true&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false and figure below.

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: *One solution for the spiral dataset using all features is a 2-hidden-layer neural network with 7 neurons in the first layer and 3 neurons in the second. See the URL (https://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=spiral&regDataset=reg-plane&learningRate=*

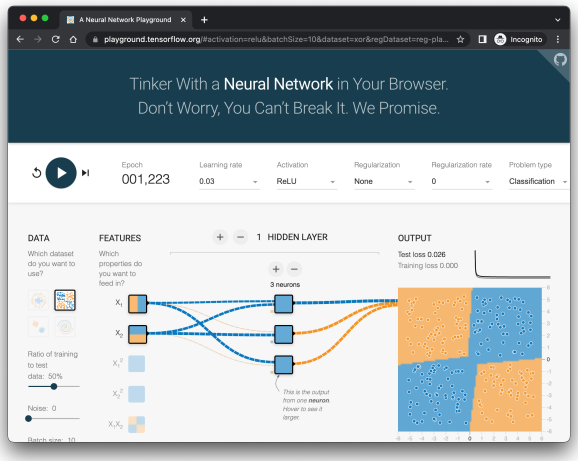

Figure 8: XOR dataset with a 1-hidden-layer neural network.

0.03&regularizationRate=0&noise=0&networkShape=7,3&seed=0.42&showTestData=false&discretize=true&percTrainData=50&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) *and the figure below. The training loss is <0.001 and the test loss is 0.008. This is a low bias/high variance model because it fits the training data well, but does not necessarily generalize well.*

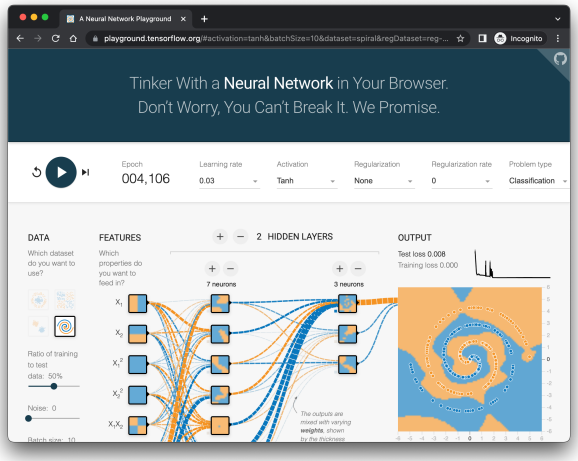

Figure 9: Spiral dataset with all features.

One solution for the spiral dataset using only x_1 and x_2 as features is a 3-hidden-layer neural network with 8 neurons in each layer. See the URL (`https://playground.tensorflow.org/#activation=relu&`

---

batchSize=10&dataset=spiral&regDataset=reg-plane&learningRate=0.003&regularizationRate=0&noise=0&networkShape=8,8,8&seed=0.42&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) and the figure below. The training loss is 0.001 and the test loss is 0.007. This is a low bias/high variance model because it fits the training data well, but does not necessarily generalize well.

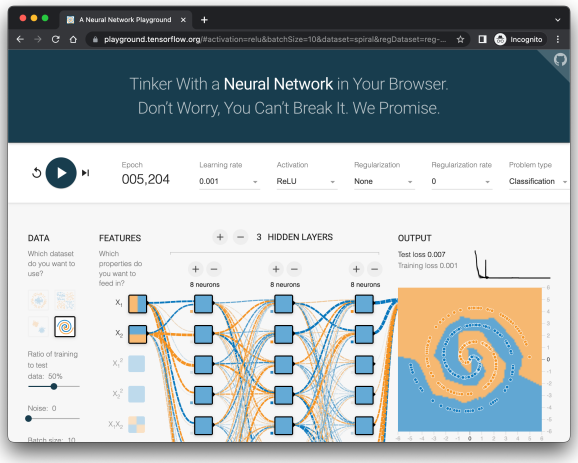

Figure 10: Spiral dataset with only x_1 and x_2 as features.

Note: This bonus problem turned out to be more difficult than I anticipated!

My initial idea for a solution of two engineered features that would allow the spiral pattern to be linearly separated was $r = \sqrt{x_1^2 + x_2^2}$ *and* $\theta = \arctan 2(x_1, x_2)$*. The four-quadrant* $\arctan 2(x_1, x_2)$ *has an output range that covers the full* $[-\pi, \pi]$ *range as defined in* https://en.wikipedia.org/wiki/Atan2*. I also had to swap* $x_1 \leftrightarrow x_2$ *from how you would usually define this because of a bug in the TensorFlow Playground code! See* https://github.com/tensorflow/playground/blob/02469bd3751764b20486015d4202b792af5362a6/src/dataset.ts#L145-L146*.*

However, if we plot these two features, we see that the resulting dataset is ***not*** *linearly separable! After staring at the source code of how the dataset is generated and some trial and error, we can come up with an engineered feature that (together with* r*) lets the dataset be linearly separated, namely* $(\theta - 2r) \mod 2\pi$ *(and there are many variations possible). See the figure below.*

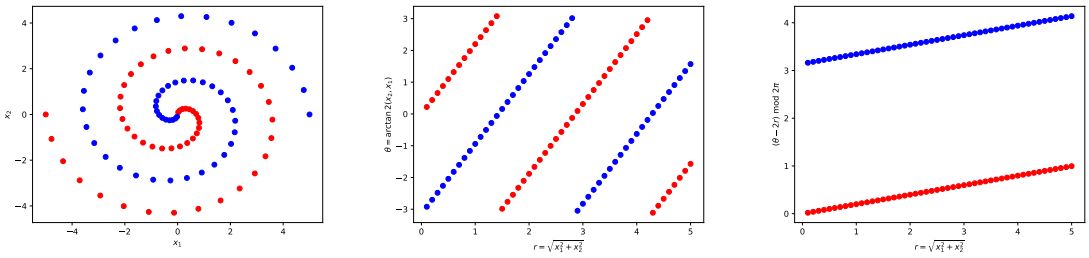

Figure 11: Spiral dataset in (x_1, x_2) (left), in (r, θ) (center), and in $(r, (\theta - 2r) \mod 2\pi)$ (right).