DLIP - Assignment 1: Training

Bradley Spronk - s2504758

February 11, 2026

1 Python data science refresher.

Link of github repository to past python project demonstrating I don't need the refresher.

2 Tensorflow playground.

Problem 4A: First, select a linear model with no hidden layers. For the features, select the two independent variables $x1$ and $x2$. Can you fit the data with this linear model? Why or why not?

It is not possible to fit the data with a linear model with only the features $x1$ and $x2$, because the data is not linearly separable. Or in other words, we cannot draw a straight line that separates the two classes of points in the feature space defined by $x1$ and $x2$.

What happens if you add the feature $x1x2$?

However, if we add the feature $x1x2$, we can fit the data perfectly, because it allows us to capture the interaction between $x1$ and $x2$, which is necessary to separate the classes.

Now, return the features to just $(x1, x2)$ 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)?

I suppose we have to use a non-linear activation function in the hidden layer, otherwise we would just be fitting a linear model again. With ReLU, a neural network with 1 hidden layer with 4 neurons is the smallest neural network that fits the training data perfectly (with a training loss of $<= 0.001$). With tanh, a neural network with 1 hidden layer with 5 neurons is the smallest neural network that fits the training data perfectly.

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).

Homework 1: Problem 4B

Navigate your web browser to the TensorFlow Playground `https://playground.tensorflow.org/#dataset=spiral&discretize=true`. Select the spiral pattern for the data.

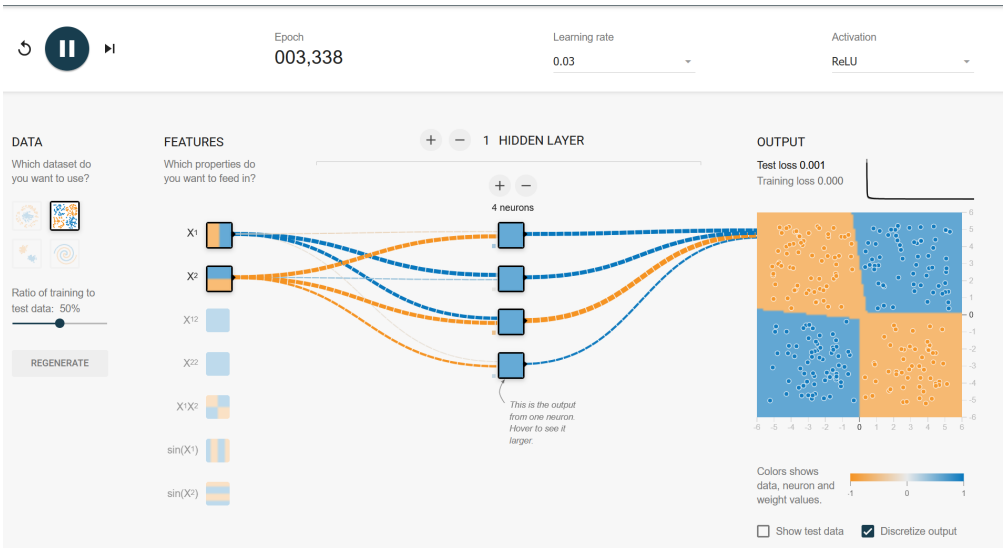

Figure 1: Screenshot of the smallest neural network that fits the training data perfectly.

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).

3 Training curves for polynomial regression.

Problem 2C

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 $N \in \{20, 25, 30, 35, \ldots, 100\}$:

   (a) 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.

   (b) 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.

Problem 2D

Based on the learning curves, which polynomial regression model (i.e. which degree polynomial) has the highest bias? How can you tell?

Problem 2E

Which model has the highest variance? How can you tell?

3.1 Problem 2F

What does the learning curve of the quadratic model tell you about how much the model will improve if we had additional training points?

3.2 Problem 2G

Why is training error generally lower than validation error?

3.3 Problem 2H

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?

4 Stochastic gradient descent for linear regression.

Homework 2: Problem 1 A-I

Problem A: We can make our algebra and coding simpler by writing $f(x1, x2, \ldots, xd) = w^{\mathrm{T}}x$ for vectors w and x. But at first glance, this formulation seems to be missing the bias term b from the equation above. How should we define x and w such that the model includes the bias term? Hint: Include an additional element in w and x.

Problem B: SGD uses the gradient of the loss function to make incremental adjustments to the weight vector w. Derive the gradient of the squared loss function with respect to w for linear regression

Problem C: 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. Each epoch corresponds to one full pass over the entire dataset. One SGD iteration (weight update) is performed for each point in the dataset. So one epoch is equivalent to N gradient updates, where N is the size of 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

Problem D: 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.

Problem E: 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 η set $10^{-6}, 5 \times 10^{-6}, 10^{-5}, 3 \times 10^{-5}, 10^{-4}$. 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 1_notebook_part2.ipynb. For your implementation of problems F-H, do consider the bias term using your answer to problem A.

Problem F: 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 $w = [0.001, 0.001, 0.001, 0.001]$ 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.

Problem G: 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

Problem H: The closed-form solution for linear regression with least squares is:
$w = \left(\sum_{i=1}^{N} x_i x_i^{\mathrm{T}}\right)^{-1} \left(\sum_{i=1}^{N} x_i y_i\right)$.
Compute this analytical solution. Does the result match up with what you got from SGD?

Problem I: Is there any reason to use SGD when a closed-form solution exists?