Machine Learning in Physics — UCSD PHYS 139/239

Homework 2

Draft version due: Friday, October 24, 2025, 8:00pm

Final version due: Wednesday, October 29, 2025, 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 2" or "Homework 2 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 2 Code" or "Homework 2 Code Corrections". The .zip file should contain your code files. Submit your code either as Jupyter notebook .ipynb files or .py files.

---

1 Stochastic Gradient Descent [36 Points]

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:

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

Problem A [2 points]: We can make our algebra and coding simpler by writing $f(x_1, x_2, \cdots, x_d) = \mathbf{w}^\intercal \mathbf{x}$ 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 can define a new feature* $x_0 = 1$ *and define* $\mathbf{w} = [b, w_1, w_2, \cdots, w_d]^\intercal$ *and* $\mathbf{x} = [x_0, x_1, x_2, \cdots, x_d]^\intercal$. *The 1 feature in the weight vector is the bias term* b. *correct or equivalent solution*

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

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

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: *The squared loss function is defined as:*

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

Taking the gradient with respect to $\mathbf{w}$*:*

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

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

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

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

correct or equivalent solution

The following few problems ask you to work with the first of two provided Jupyter notebooks for this problem, `1_notebook_part1.ipynb`, which includes tools for gradient descent visualization. This notebook utilizes the files `sgd_helper.py` and `sgd_multiopt_helper.py`, 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*. 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.

Solution C: *See code in* `code/1_notebook_part1.ipynb`. *correct or equivalent solution*

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:

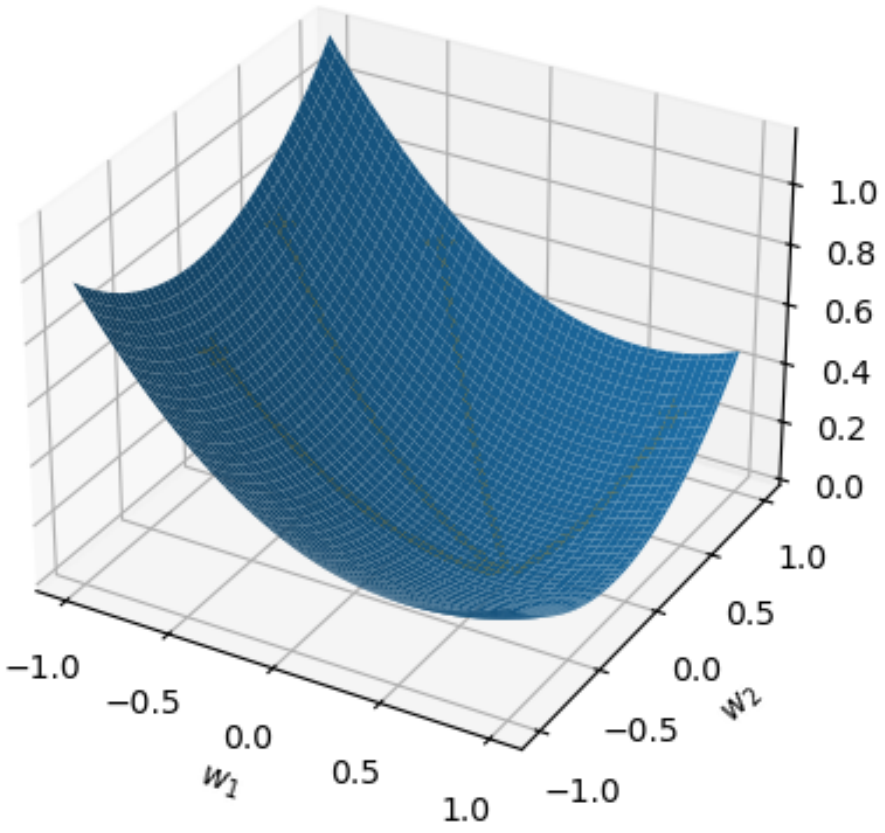

Figure 1: SGD convergence for dataset 1.

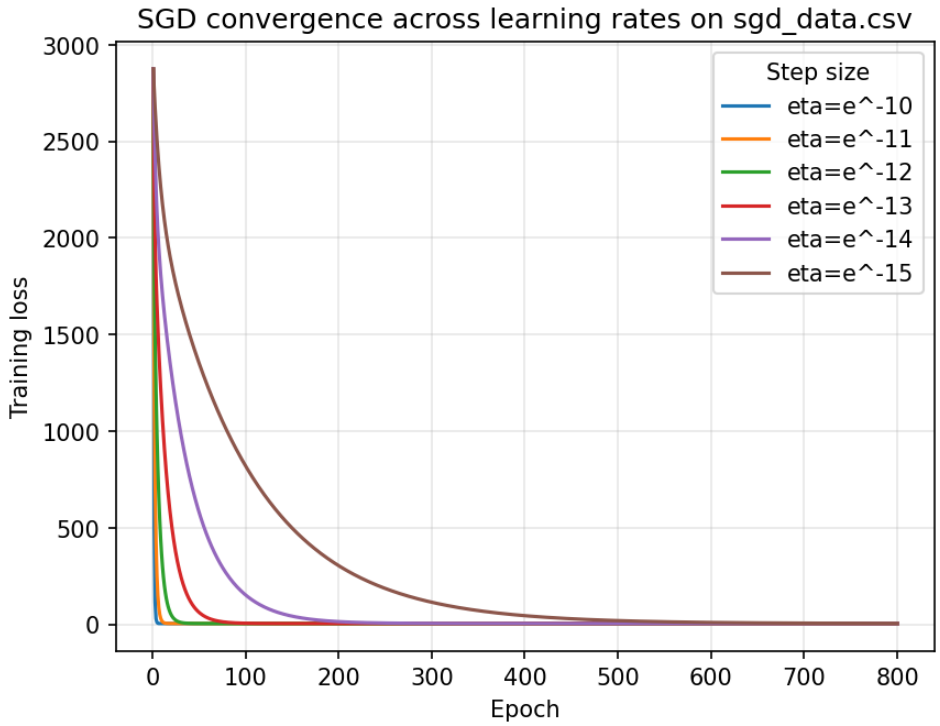

Figure 2: SGD convergence for dataset 2.

The starting point of the of the SGD affects the convergence behavior of the algorithm. If the starting point is close to the optimal point, the algorithm will converge quickly. If the starting point is far from the optimal point, the algorithm will converge slowly. The greater the loss is, the faster the algorithm will converge. They all seems to converge to the same point, but the rate of convergence is different. The behaviour of datasets 1 and 2 seems nearly identical, except the loss landscape is just shifted slightly.

correct or equivalent solution

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 $\eta \in \{10^{-6}, 5 \times 10^{-6}, 10^{-5}, 3 \times 10^{-5}, 10^{-4}\}$. On a single plot, show the training error vs. number of epochs trained for each of these values of η. What happens as η changes?

Solution E:

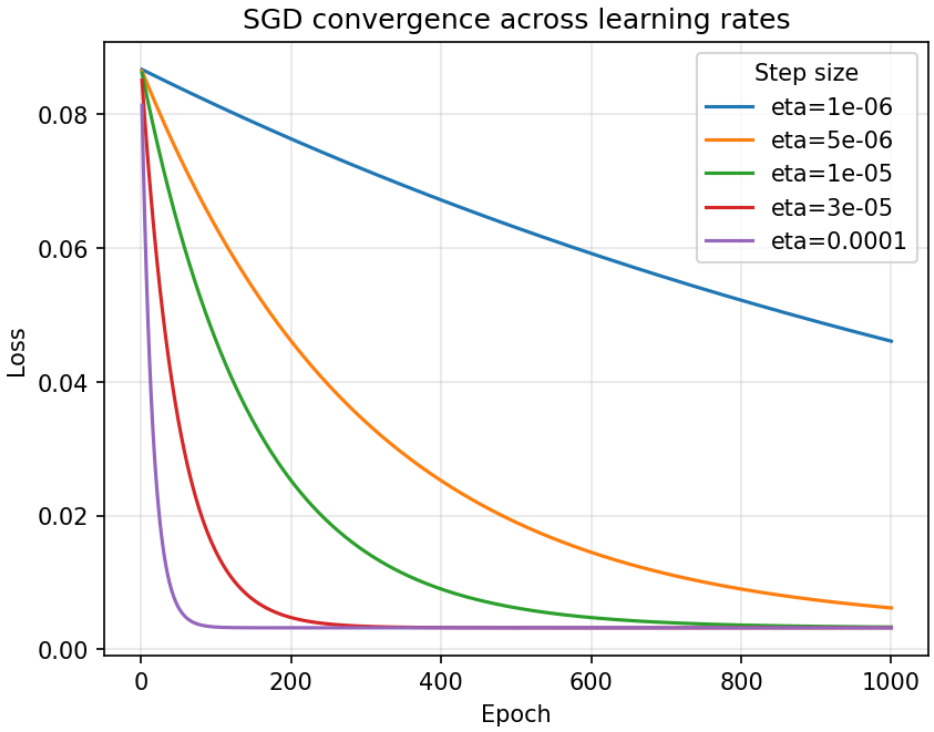

Figure 3: SGD convergence for dataset 1 across different learning rates.

The behaviour of the algorithm is different for different learning rates. The greater the learning rate, the faster the algorithm will converge. correct or equivalent solution

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 [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:

```
[ -5.94209855   3.94390726 -11.72382684   8.78568884  -0.22717454]
```

correct or equivalent solution

Problem G [2 points]: Perform SGD as in the previous problem for each learning rate η 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 η. Explain what is happening.

Solution G:

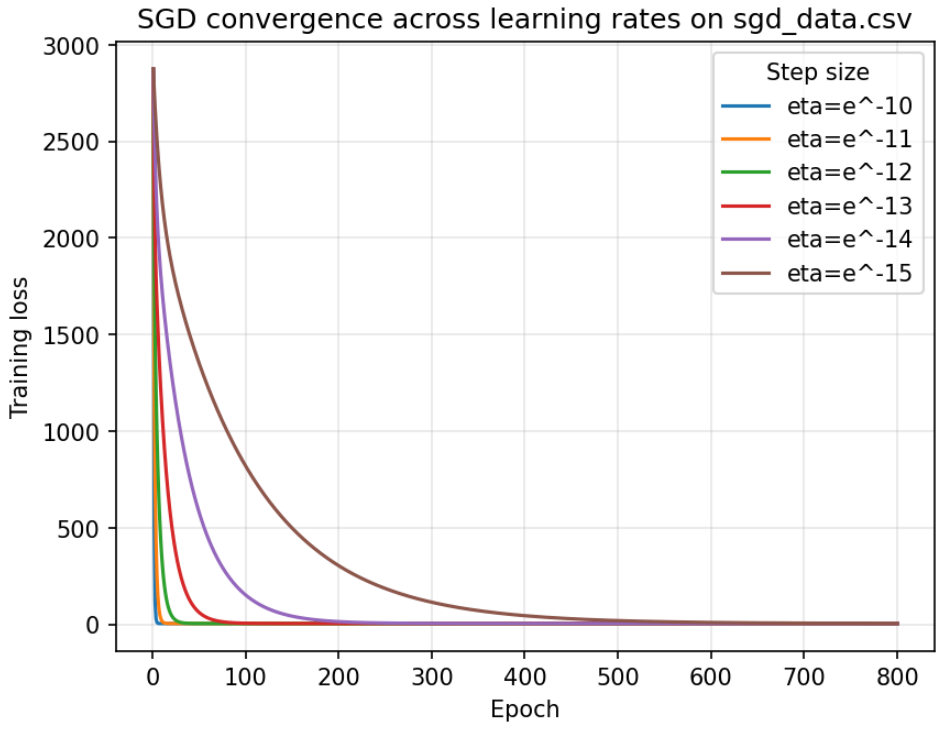

Figure 4: SGD convergence for dataset 1 across different learning rates.

The behaviour of the algorithm is different for different learning rates. The greater the learning rate, the faster the algorithm will converge. correct or equivalent solution

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

$$\mathbf{w} = \left(\sum_{i=1}^{N} \mathbf{x_i} \mathbf{x_i}^\intercal \right)^{-1} \left(\sum_{i=1}^{N} \mathbf{x_i} y_i \right).$$

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

Solution H:

```
Final SGD weights: [ -5.9421018, 3.94390759, -11.72383556,8.78568396, -0.22719006]
CF weights:[ -5.99157048, 4.01509955, -11.93325972,8.99061096, -0.31644251]
‖w_sgd - w_closed‖ = 0.31833158703929115
‖Xw_sgd - Xw_closed‖ = 27.479673911001605
```

correct or equivalent solution

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: *The closed-form solution is computationally expensive. SGD can yield similar results more efficiently correct or equivalent solution*

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: *A better stopping condition is when the rate of change of validation loss decreases below a certain threshold, meaning the model is close to converging. correct or equivalent solution*

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

Solution K: *The perceptron algorithm updates weights only for misclassified examples and converges only for linearly separable data, where SGD updates continuously. correct or equivalent solution*

2 Neural networks vs. boosted decision trees [45 Points]

In this problem, you will compare the performance of neural networks and boosted decision trees for binary classification on a tabular dataset, namely the MiniBooNE dataset: https://archive.ics.uci.edu/ml/datasets/MiniBooNE+particle+identification.

This dataset is taken from the MiniBooNE experiment and is used to distinguish electron neutrinos (signal) from muon neutrinos (background) The dataset contains 130,065 samples with 50 features and a single binary label. We will randomly split the dataset into training (80%) and testing (20%) subsets.

We will use `2_notebook_part1.ipynb` for parts A and B and `2_notebook_part1.ipynb` for parts C, D, and E.

Problem A [15 points]: Using the MiniBooNE dataset and XGBoost, train a boosted decision tree on the training dataet. Use the Scikit-learn API `xgboost.XGBClassifier`. For an initial choice of hyperparameters use 100 trees (`n_estimators`), maximum tree depth (`max_depth`) of 10, learning rate (`learning_rate`) of 0.1, `colsample_bytree` of 0.8, and `subsample` of 0.8.

Plot the receiver operating characteristic (ROC) curve using the testing dataset. What area under the curve (AUC) and accuracy do you achieve "out of the box"?

Solution A:

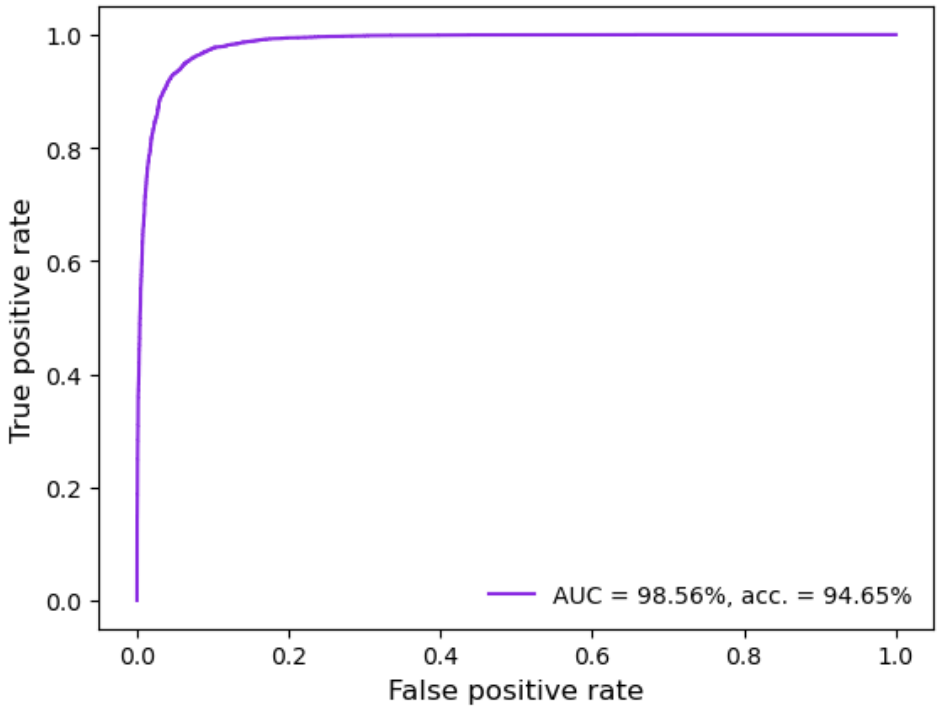

Figure 5: ROC curve for XGBoost on MiniBooNE dataset.

correct or equivalent solution

Problem B [5 points]: Plot the F-score for all the 10 "most important" features using `xgboost.plot_importance`. Which feature is the most important?

Plot this feature using the testing dataset in a 1D histogram separately for signal and background. For the histogram binning, use 100 bins from the minimum value of this feature to the maximum value of this feature in the testing dataset. What do you notice about this feature?

The F12 feature seems to be the most important feature. This feature seems to be exponentially distributed increasing exponentiallytowards 1.

Solution B:

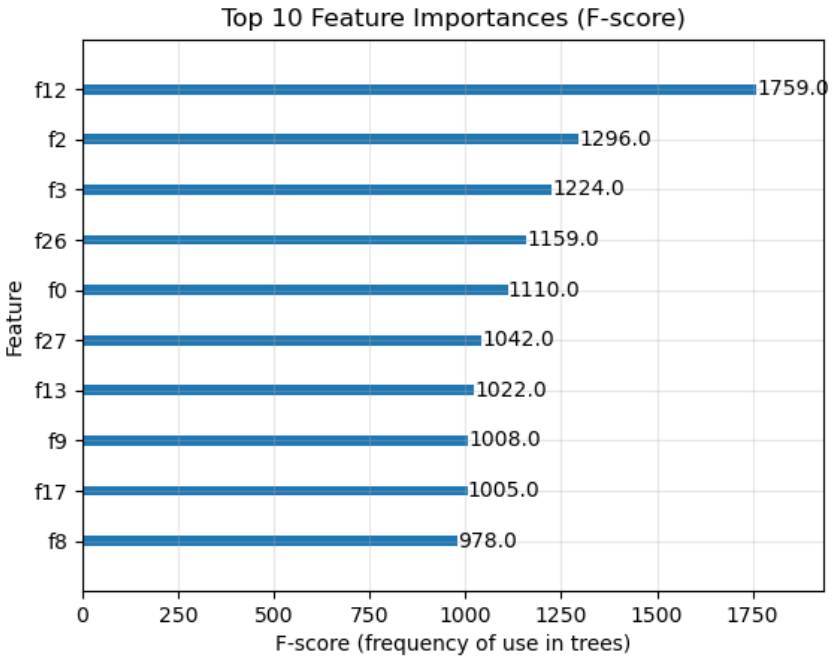

Figure 6: Top 10 feature importances for XGBoost on MiniBooNE dataset.

correct or equivalent solution

It seems that there are outlier values set to -999.0, the full plot is not very informative, because the range is too large. The zoomed plot is better, showing the distribution of the feature.

Solution B:

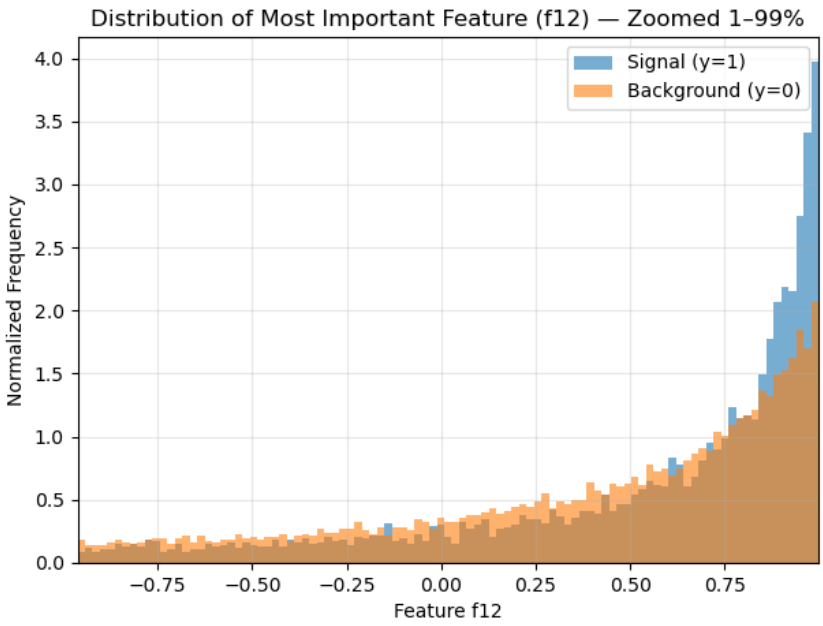

Figure 7: Feature distribution for XGBoost on MiniBooNE dataset.

correct or equivalent solution

Problem C [15 points]: Using the MiniBooNE dataset and the Keras Model API, train a neural network with 3 hidden layers each with 128 units and tanh activations. The final layer should have sigmoid activation. Use the binary crossentropy loss function, the SGD optimizer with a learning rate of 0.01 (which is the default), and a batch size of 128. Train the model for 50 epochs.

Plot the receiver operating characteristic (ROC) curve using the testing dataset. What AUC and accuracy do you achieve "out of the box"?

Solution C:

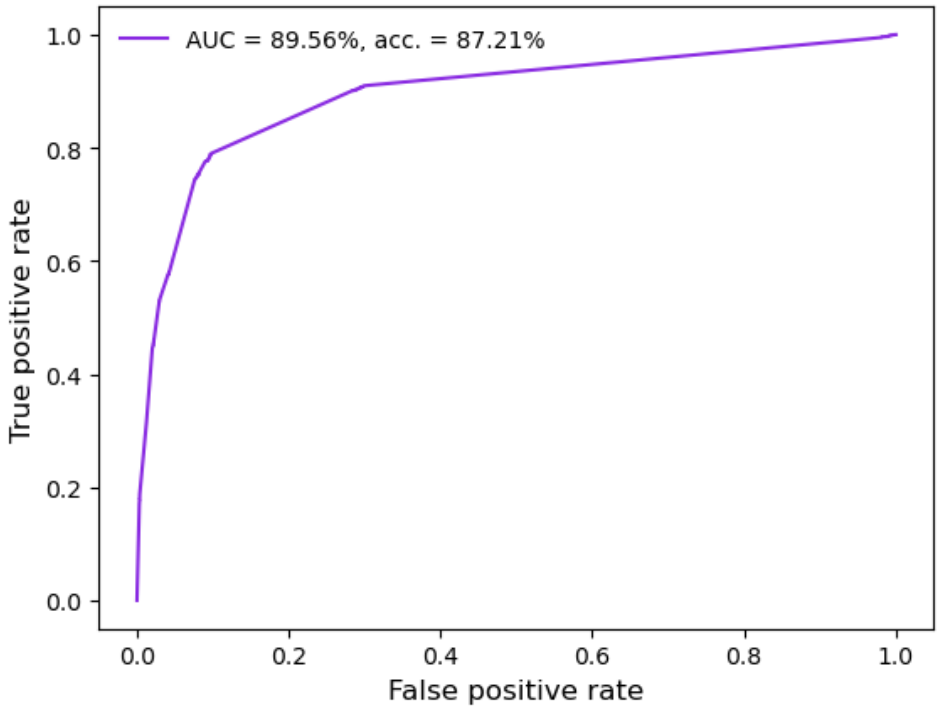

Figure 8: ROC curve for neural network on MiniBooNE dataset.

correct or equivalent solution

Problem D [5 points]: Swap out the tanh activations for ReLU activations, while keeping everything else the same. Does the network train effectively? Why or why not?

Solution D: *The network trains more effectively with ReLU activations. the ReLU activation function makes the loss landscale more convex, making it easier for a model to converge. I had to change a couple things because I kept getting NaN values likely due to the ReLU activations causing the loss to be unstable. I set the learning rate to 0.0001 and used he_normal initialization for the weights. I've also tried casting the data to float64 to avoid the NaN values.*

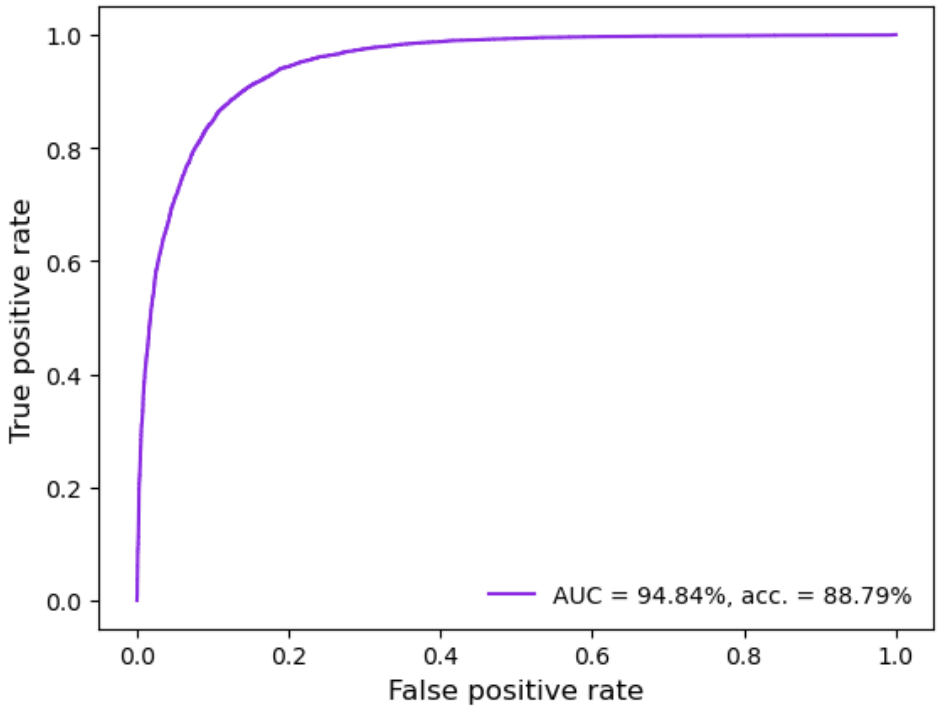

Figure 9: ROC curve for neural network on MiniBooNE dataset with ReLU activations.

correct or equivalent solution

Problem E [5 points]: Now, we will make two minor changes to the network with ReLU activations: preprocessing and the optimizer.

For the feature preprocessing use `sklearn.preprocessing.StandardScaler` to standardize the input features. Note you should fit the standard scaler to the training data *only* and apply it to both the training and testing data. For the optimizer, use Adam with a learning rate of 0.001 (which is the default) instead of SGD. Train the model for 50 epochs.

Plot the receiver operating characteristic (ROC) curve using the testing dataset. What AUC and accuracy do you achieve now? Is it comparable to the BDT?

Solution E:

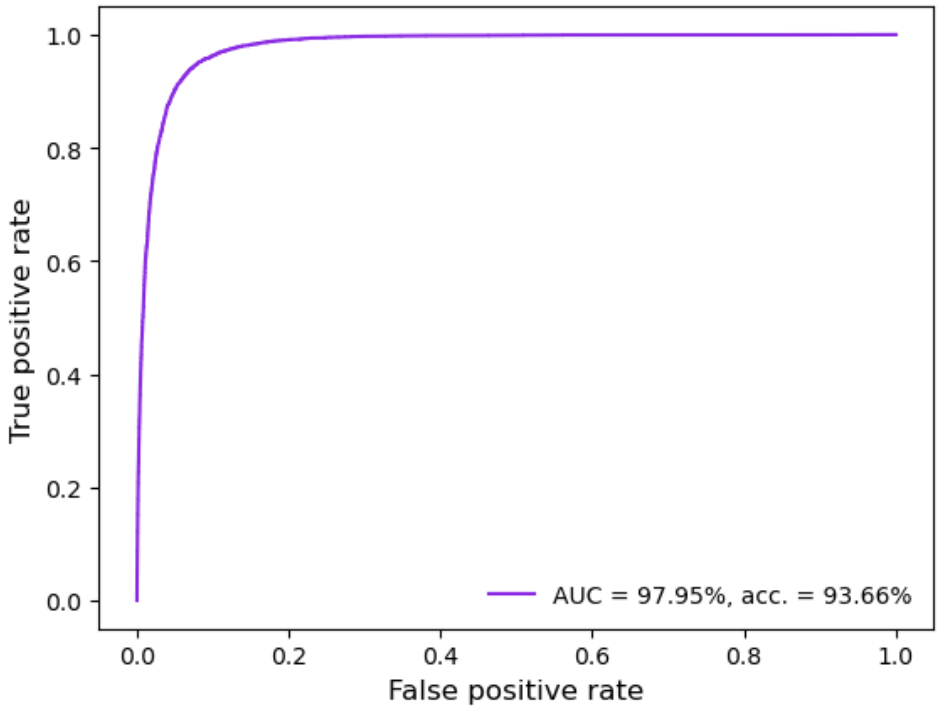

Figure 10: ROC curve for neural network on MiniBooNE dataset with ReLU activations and StandardScaler.

It is a lot worse than the BDT. The BDT is able to achieve an AUC of 98.56 and a accuracy of 94.65 while the neural network is able to achieve an AUC of 94.84 and an accuracy of 88.79. correct or equivalent solution