CSCI-561 - Fall 2022 - Foundations of Artificial Intelligence

Homework 3

Due Monday November 28, 2022, 23:59:59 PST

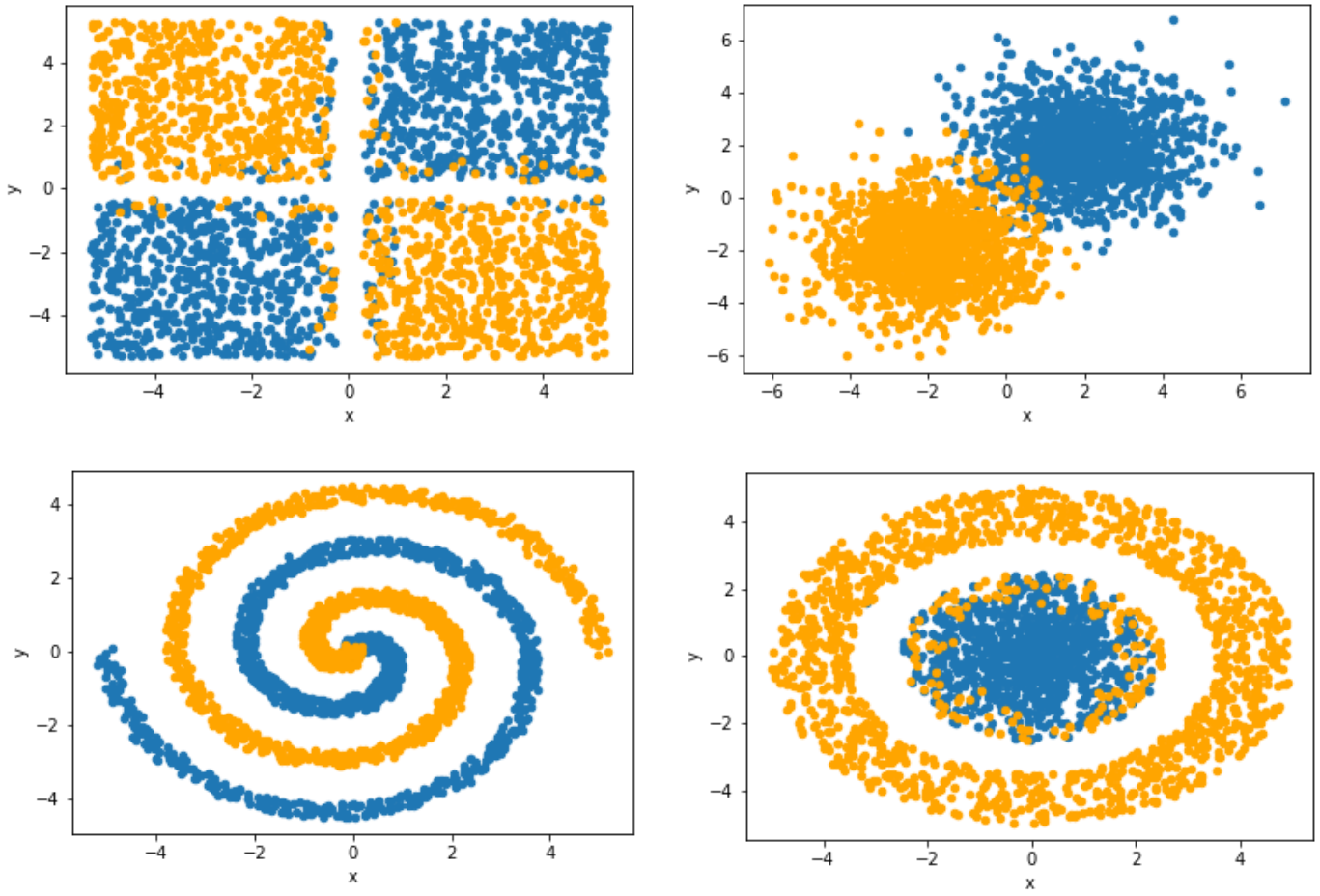


Figure 1: Plots depicting the four datasets you will classify in this assignment.

1. Assignment Overview

In this homework assignment, you will implement a multi-layer perceptron (MLP) neural network and use it to classify data from four different datasets, shown in Figure 1. Your implementation will be made from scratch, using no external libraries other than Numpy; machine learning libraries are **NOT** allowed (e.g. Scipy, TensorFlow, Caffe, PyTorch, Torch, mxnet, etc.).

2. Data Description

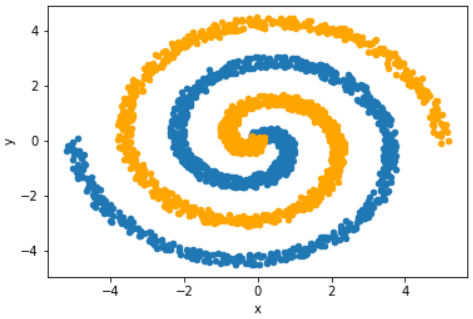
You will train and test your neural network implementation on four datasets inspired by the TensorFlow Neural Network Playground (https://playground.tensorflow.org). We encourage you to visit this site and experiment with various model settings and datasets.

There are 4 files associated with each dataset. The files have the following naming scheme:

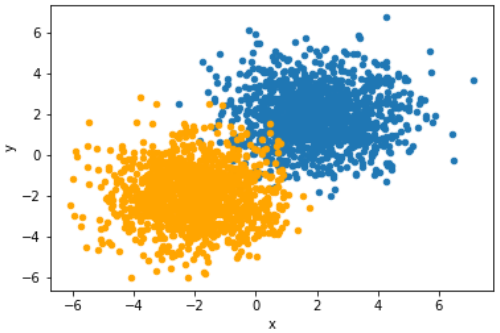
1. `<name>_train_data.csv` : training samples, each $x \in R^2$
2. `<name>_train_label.csv` : training labels, each $y \in \{0, 1\}$
3. `<name>_test_data.csv` : test samples, each $x \in R^2$
4. `<name>_test_label.csv` : test labels, each $y \in \{0, 1\}$

where `<name>` is one of the 4 dataset names: **spiral**, **circle**, **xor**, or **gaussian**. As a result, there are a total of **16** data files, all of which can be found in `HW3->resource->asnlib->public`.
Below is a visual representation of each dataset along with a brief description.

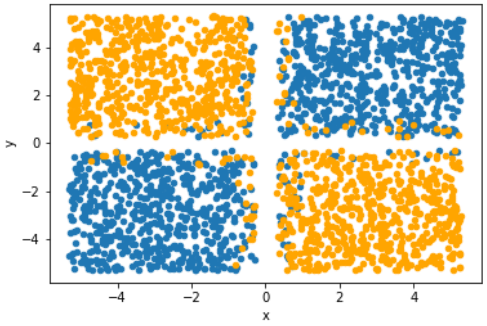
Spiral
Both classes are interwoven in a spiral pattern. Data points are subject to noise along their respective spiral and thus may overlap.



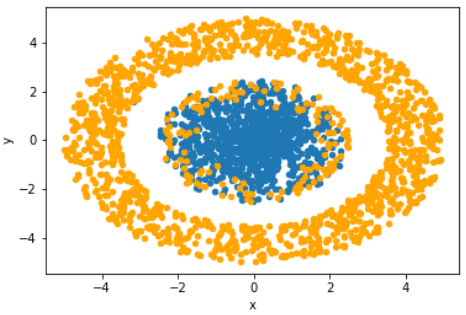
Gaussian
Data points are generated and classified according to two Gaussian distributions. The distributions have different means, but samples may overlap as pictured.



XOR
Data points classified according to the XOR function. Noise may push data classes over XOR “boundaries” as seen in the figure below.



Circle
Data points are generated and classified according to two annuli (rings) sharing a common center. Although the annuli are not overlapping, noise may push data points across the gap



The train and test files for each dataset represent an **80/20 train/test split**. You are welcome to aggregate the data from each set and re-split to your liking. All datasets have 2-dimensional data points (the x,y coordinates of the point in R^2), along with binary labels (either 0 or 1).

3. Task description

Your task is to implement a multi-hidden-layer neural network learner (see model description part for additional details), that will do the following. For a given dataset,

1. Construct and train a neural network classifier using provided labeled training data,
2. Use the learned classifier to classify the unlabeled test data,
3. Output the predictions of your classifier on the test data into a file in the ***same*** directory**,**
4. ***Finish in 2 minutes (for both training your model and making predictions)***.

Your program will take three input files (provided as paths through command line arguments) and produce one output file as follows:

```
run your_program train_data.csv train_label.csv test_data.csv
⇒ test_predictions.csv
```

For example,

```
python3 NeuralNetwork3.py train_data.csv train_label.csv test_data.csv
⇒ test_predictions.csv
```

In other words, your algorithm file `NeuralNetwork.*` will take training data, training labels, and testing data as inputs, and output classification predictions on the testing data. Note that your neural network implementation should not depend on which of the four datasets are provided during a given execution; your script will only receive the training data/labels and test data for a single dataset type at a time.

As mentioned in the overview, ***NumPy is the only external library you can use in your implementation (or equivalent numerical computing-only library in non-Python languages).*** By *external* we mean *outside the standard library* (e.g. in Python, `random`, `os`, etc are fine to use). No component of the neural network implementation can leverage a call to an external ML library; you must implement the algorithm yourself, from scratch. (You will receive no credit for this assignment if this rule is not adhered to).

The format of `*_data.csv` looks like:

$x_1^1, x_2^1,$
$x_1^2, x_2^2,$
...

Where $x_1^{(n)}, x_2^{(n)},$ are the coordinates of the n^{th} data point. The `*_label.csv` and ***your*** output `test_predictions.csv` will look like

y^1
y^2
...

where $y^{(n)}$ is either 0 or 1 corresponding to the label for data point $x^{(n)}$ (where is the n^{th} data point, $[x_1^{(n)}, x_2^{(n)}]$). Thus, there is a single column indicating the predicted class label for each unlabeled sample in the input test file.

The format of your `test_predictions.csv` file is crucial. Your output file must have this ***name and format*** so that it can be parsed correctly to compare with true labels by the auto-grading scripts. This file should be written to your working path.

When we grade your submission, we will use ***hidden*** training data and ***hidden*** testing data for each dataset instead of the public data you are provided. That is, for each of the four datasets (spiral, circle, xor, or gaussian), your NN submission will be trained from scratch on hidden training data and evaluated on hidden test data. The handling of arguments in your program, along with the name/format of your output prediction file must match the above specifications to ensure your submission is auto-graded correctly.

The maximum running time to train and test a model is 2 minutes *for each dataset*. This means training/testing across all datasets can take at most 8 minutes, where a 2 minute limit is applied *per dataset* (i.e. time does not bleed over if a dataset is "finished" prior to the 2 minute mark).

4. Model description

The model you will implement is a vanilla feed-forward neural network, possibly with many hidden layers (see Figure 2 for a generic depiction). Your network should have 2 input nodes and output a single value. Beyond this, there are no constraints on your model's structure; it is up to you to decide what activation function, number of hidden layers, number of nodes per hidden layer, etc your model should use. It's worth noting you should be using **cross-entropy** as your loss function (each dataset presents a binary classification task). Depending on your implementation, you may also need to employ the **softmax function** on your last-layer outputs and select a single value for your final output.

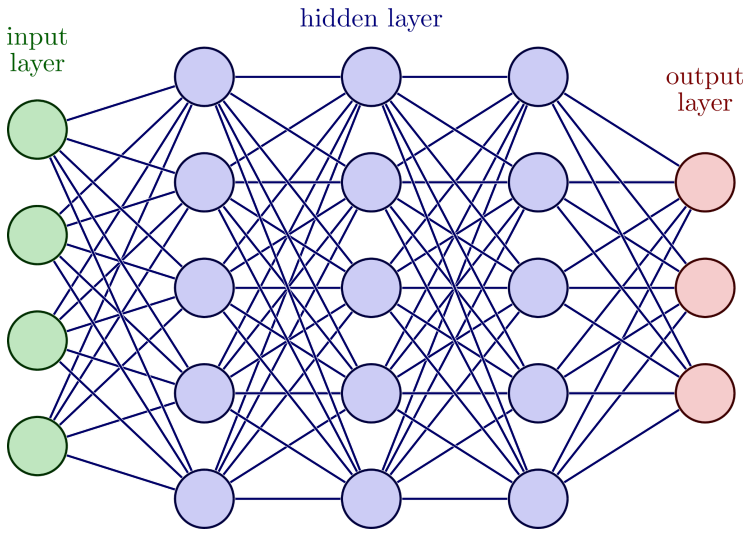


Figure 2: Diagram of an example neural network with 3 hidden layers.

There are many hyperparameters you will likely need to tune to get better performance. These can be hard-coded by you in your program (possibly after structured exploration of your hyperparameter space), or selected through a cross validation process dynamically (in the latter case, be wary of runtime limits). A few example hyperparameters are as follows:

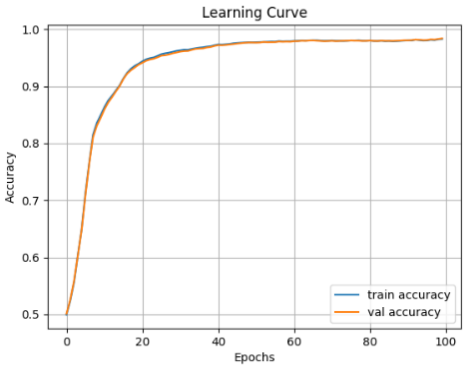
- **Learning rate**: step size for update weights (*e.g. weights = weights - learning * grads*), different optimizers have different ways to use learning rate.

- **Mini-batch size**: number of samples processed each time before the model is updated. The mini-batch size is some value smaller than the size of the dataset that effectively splits it into smaller chunks during training. Using batches to train your network is highly recommended.
- **Number of the epochs**: the number of complete passes through the training dataset (*e.g. you have 1000 samples, 20 epochs mean you loop through these 1000 samples 20 times).*
- **Number of hidden layers & number of units in each hidden layer**: these decide the overall structure of your model. Unnecessarily deep or wide networks may negatively impact your model's performance given the time constraints. Here you need to find a proper tradeoff between feasible time to convergence and model expressivity.

5. Implementation Guidance

Here are a few suggestions you might want to consider during your implementation:

1. **Train your model using mini-batches**: there are many good reasons to use mini-batches to train your model (instead of individual points or the entire dataset at once), including benefits to performance and convergence.
2. **Initialize weights and biases**: employ a proper random initialization scheme for your weights and biases. This can have a large impact on your final model.
3. **Loss function**: as mentioned, you need to use cross-entropy as your loss function.
4. **Use backpropagation:** hardly needs mentioning, but you should be using backpropagation along with a gradient descent-based optimization algorithm to update your network's weights during training.
5. **Vectorize your implementation:** vectorizing your implementation can have a large impact on performance. Use vector/matrix operations when possible instead of explicit programmatic loops.
6. **Regularize your model**: leverage regularization techniques to ensure your model doesn't overfit the training and keeps model complexity in check. This can be especially important in settings with noisy data (which you will face on both the public and hidden grading datasets).
7. **Plot your learning curve:** plotting your train/test accuracy after each epoch is a quick and helpful way to see how your network is performing during training. Here you **are allowed** to use external plotting libraries, but worth noting that you should likely remove them prior to submission for performance reasons. The figure on the right shows a generic example of such a plot; your plot(s) may look different.



8. **Putting it all together**: see Figure 3 on the next page for a basic depiction of an example training pipeline. Note that this diagram lacks detail and is only meant to provide a rough outline for how your training loop might look.

While recommended, the use of these suggestions in your implementation is not explicitly required. Your grade will be determined entirely by your model's performance as described in Section 6.

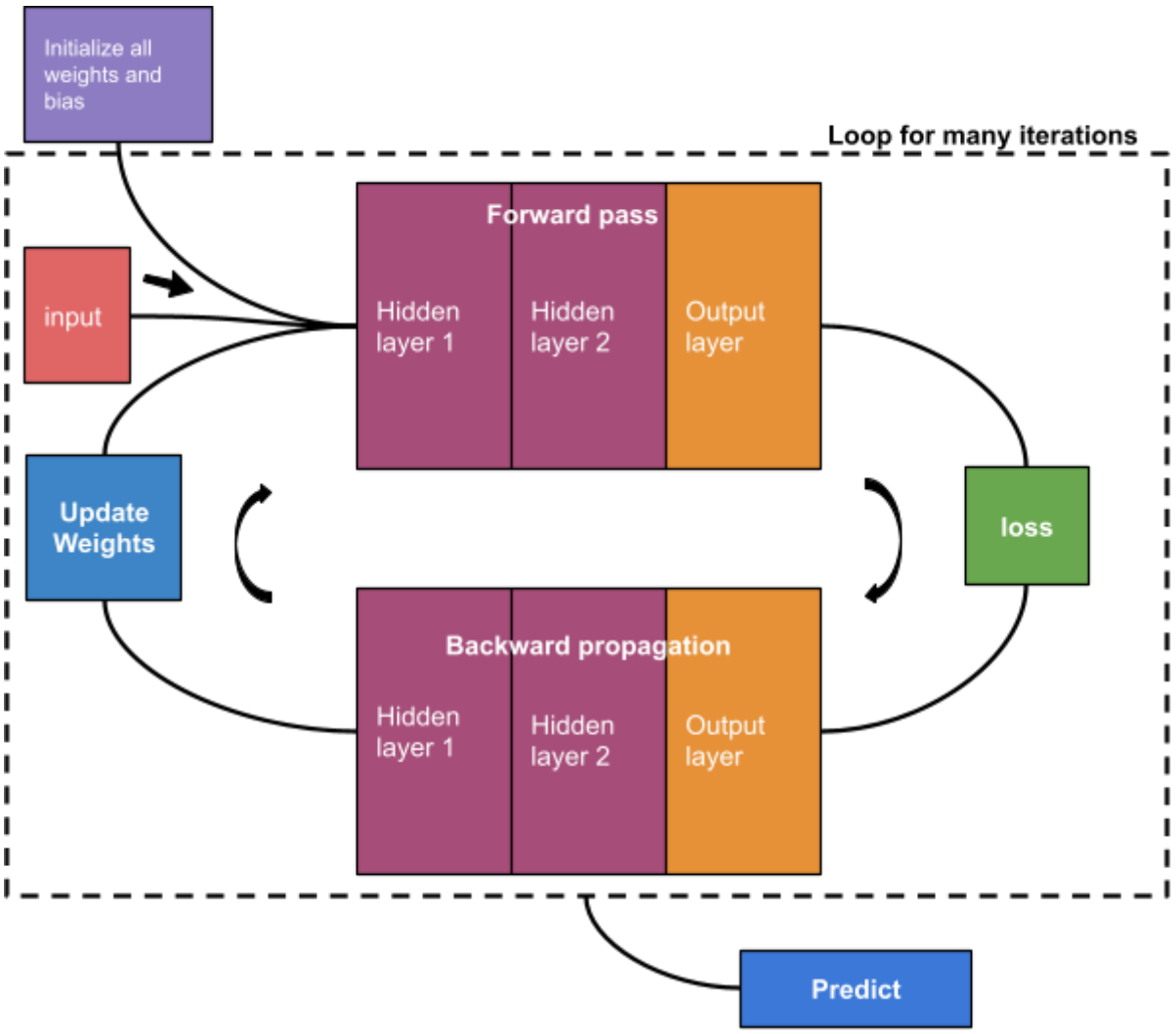


Figure 3: Diagram depicting the basic components of the training process.

6. Submission and Grading

Submission

- **Program name**: name your program **NeuralNetwork.*** where '*' is the extension for the programming language you choose ("py" for python, "cpp" for C++, and "java" for Java). If you are using C++11, then the name of your file should be "NeuralNetwork11.cpp" and if you are using python3 then the name of your file should be "NeuralNetwork3.py". Please use only the programming languages mentioned above for this homework. Please note the highest version of Python that is offered is Python 3.7.5, hence the walrus operator and other features of more recent Python releases are not supported.
- **Program arguments**: as described previously, we will pass 3 input file paths as *command-line arguments* to your program. Each call to your program will look as follows:

```
run your_program <path_to_training_data> <path_to_training_label> <path_to_test_data>
```

Your program must be able to handle these three CLI arguments and read data from the files at the provided locations. Note that this means you shouldn't be hard-coding any input filenames in your program; otherwise there will likely be issues during runtime in Vocareum.

- **Output file**: Your program should output a file containing your model's predictions on the test set named `test_predictions.csv`. The format of this file must adhere to the specifications discussed in Section 3 (Task Description).

Grading

Your implementation will be graded on its test classification accuracy on a **hidden** version of each of the 4 datasets described above. For each dataset type, your model will be trained on a hidden training set with *1000 samples*, and evaluated on an associated hidden test set with *250 samples*.

Note: The score you receive from submissions on Vocareum *prior* to the deadline is computed only from your implementation's performance on the provided public datasets. This means your final score (which will be computed *after* the deadline on the *hidden* data) may be different from the scores you see during development. For implementations that are sufficiently robust (e.g. using appropriate regularization, cross validation, etc), this change in score will likely be minimal. The hidden sets are all sampled from the same distributions that yielded the available public data.

Your grade is then determined by your model's prediction accuracy ***on all 4 hidden test sets, using the following scheme***:

XOR test acc-to-score mapping:
[90, 100] → 100
[86, 90) → 85
[80, 86) → 70
[75, 80) → 50
[0, 75) → 0

Circle test acc-to-score mapping:
[86, 100] → 100
[81, 86) → 85
[75, 81) → 70
[70, 75) → 50
[0, 70) → 0

Spiral test acc-to-score mapping:
[95, 100] → 100
[91, 95) → 85
[85, 91) → 70
[75, 85) → 50
[0, 75) → 0

Gaussian test acc-to-score mapping:
[96, 100] → 100
[92, 96) → 85
[85, 92) → 70
[75, 85) → 50
[0, 75) → 0

Final Grade = 0.25 * [Score(XOR set) + Score(Spiral set) + Score(Circle set) + Score(Gaussian set)]

Note[1]: [A, B) means A <= x < B.
Note[2]: the grading rubrics vary by dataset due to differences in relative complexity across datasets.
Note[3]: Directly loading pre-trained weights of the neural network is prohibited.

7. Academic Honesty and Integrity

All homework material is checked vigorously for dishonesty using several methods. All detected violations of academic honesty are forwarded to the Office of Student Judicial Affairs. To be safe, you are urged to err on the side of caution. Do not copy work from another student or off the web. Keep in mind that sanctions for dishonesty are reflected in *your permanent record* and can negatively impact your future success. As a general guide:

- **Do not copy** code or written material from another student. Even single lines of code should not be copied.
- **Do not collaborate** on this assignment. The assignment is to be solved individually.
- **Do not copy** code off the web. This is easier to detect than you may think.

- **Do not share** any custom test cases you may create to check your program's behavior in more complex scenarios than the simplistic ones that are given.
- **Do not copy** code from past students. We keep copies of past work to check for this. Even though this project differs from those of previous years, do not try to copy from the homework of previous years.
- **Do not ask Piazza about** how to implement some function for this homework, or how to calculate something needed for this homework.
- **Do not post your code on Piazza** asking whether or not it is correct. This is a violation of academic integrity because it biases other students who may read your post.
- **Do not post test cases on Piazza** asking for what the correct solution should be.
- **Do** ask the professor or TAs if you are unsure about whether certain actions constitute dishonesty. It is better to be safe than sorry.
- **DO NOT USE ANY** existing machine learning library such as Tensorflow, Pytorch, Scikit-Learn, etc. Violation will cause a penalty to your credit.