EE148b Assignment 1
Implementing a Multi-Class Classifier with a Multilayer Perceptron

April 6, 2023

Due: April 20, 2023 2:00pm

Late Policy: 1% per hour (with no points after 48 hours). If you need a 1-week grace because of work/travel/health/family obligations, please make arrangements at least 3 days in advance of the deadline. If an emergency arises, contact us ASAP to make arrangements.

Regrade Policy: You may submit a regrade request if you believe there was an error in grading your assignment. Any regrade requests should be submitted through Gradescope within one week of receiving your grade. Please try to be as specific as possible with your regrade request. (The grade will be re-computed and may go up or down).

Please list any resources that you used to do this homework.

Learning Objectives

This assignment serves as a deeper dive into multilayer perceptrons (MLPs). The MLP will make a binary (0/1) prediction about the data based on its features.

Learnings:
1. Data Balances
2. Data Splits (i.e. Train-Test splits)
3. Backpropagation
4. Varying Activation Functions
5. Initializations and Hyperparameters i.e.:
   - i. Number of Training Examples
   - ii. Number of Epochs
   - iii. Number of Parameters
   - iv. Number of Layers

What and Where to Submit

You will submit the following files to Gradescope by **4/20/2023** at **2:00pm**:

- Upload a PDF file `assignment1.pdf` to the Hw1 submission, containing answers to the questions asked in the assignment along with plots asked for in the assignment (The .py files will generate each of the plots for you when given the proper input). Answers should be written clearly and succinctly; you may lose points if your answer is unclear or unnecessarily complex.

- Upload a .zip of all of the .py files. **We need to be able to run your `.py` files** through a set of automated tests (given to you as `test_functions.py`).

Getting the Code

Go to your Caltech Google Drive account. Download this zip file. This zip file contains a python file with mlp stub code, a python file that sets up the datasets, a plotting file to help you plot results, a test file to help you debug your MLP implementation, a file to set a random seed in, and a stub notebook as well as stub python files for each of the parts. You can upload the unzipped folder to Google Colab and run the notebook in Colab. However, we strongly recommend programming each of the sub-parts in an IDE (i.e. VSCode, Pycharm) with debugger access, etc.

Python Libraries

We will be implementing an MLP from scratch using **just `numpy`,**`random`**,** and other, non-deep-learning packages (i.e no `pytorch`, `jax`, `tensorflow`, etc.)

In Google Colab, these libraries should already be installed, so you will just need to import them if they are not imported already.

Dataset

We will be using 2 synthetic datasets and 1 real dataset (Iris dataset). You will be able to visualize 2D data on a plot. For data with greater dimensionality, you may visualize their principal components if you would like to gain intuition, but this is not required. Throughout the assignment, you will compare these distributions to see conditions in which the Multilayer Perceptron can and cannot distinguish classes.

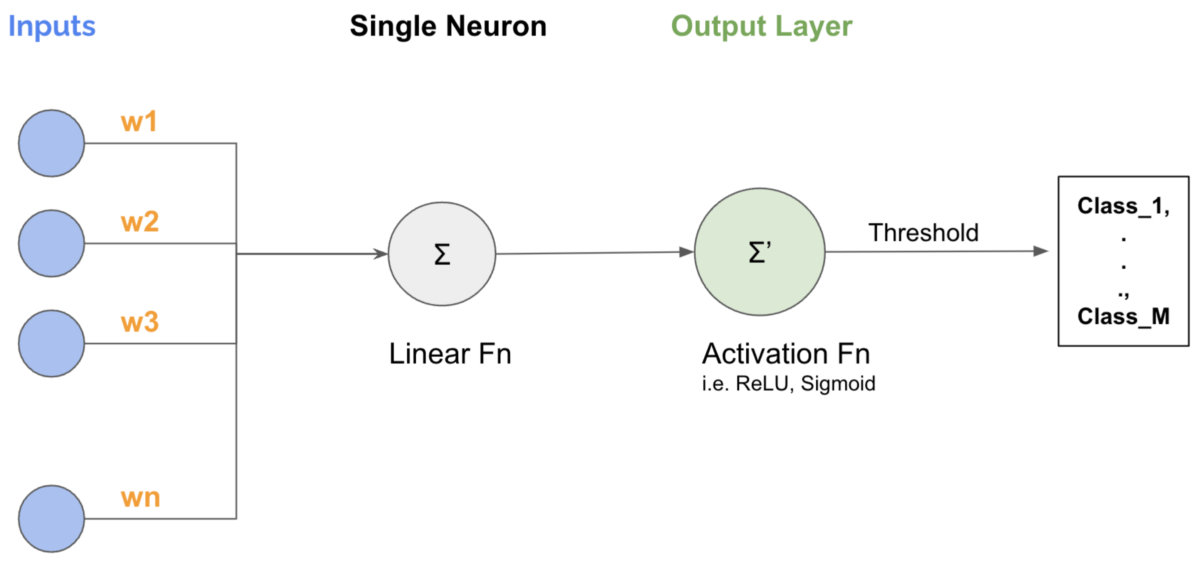
What is a Multilayer Perceptron for Multi-Class Classification?

First, what is a Perceptron?

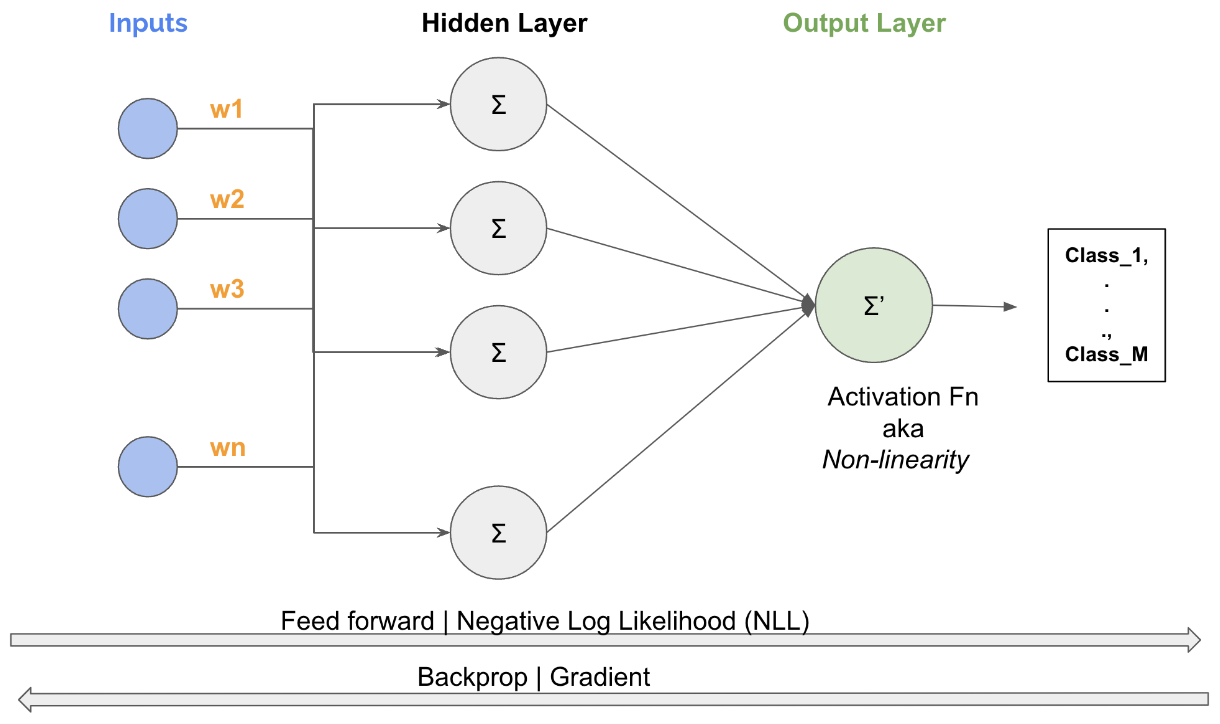
Neural Networks, as we know by now, were inspired by neuroscience and the structure of the brain. The original neuron in a deep learning paradigm was a simple linear model that generated positive and negative outputs given vectors of inputs (**x**) and weights (**w**).

$$f(x,\ w) \ = \ dot\,product(x, w)$$

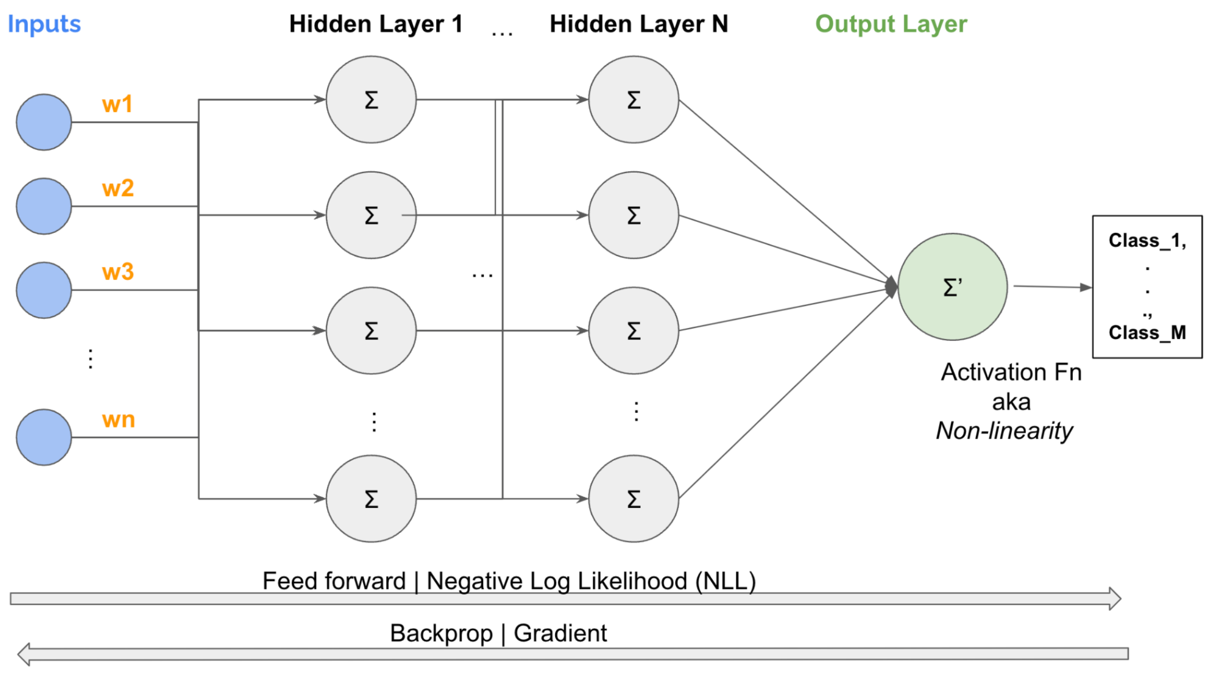
The first application of a neuron in neural networks was a logic gate (either an AND or an OR for some condition). Frank Rosenblatt, a “father” of deep learning, extended this concept of a neuron so that it would be able to *learn*. He generated one of the first Perceptron algorithms. Similar to before, neurons take combined inputs where each input has a corresponding weight.



The **Multilayer Perceptron** has input and output layers with one or more hidden layers with many neurons. While the Perceptron has an activation function that uses a threshold, the MLP can have any arbitrary activation function. A multi-perceptron *with one hidden layer* appears as follows



A multi-perceptron *with multiple hidden layers* appears as follows



Backpropagation

Backpropagation is the mechanism that allows the MLP to adaptively adjust the weights within the neural network with each iteration. The goal of the weights is to minimize the cost function or loss - we will start using the *negative log likelihood* (NLL) for this.

For the binary case, this is:

$$L(w) \;=\; -\frac{1}{N}\sum_{i=1}^{N} y_i * \log p_w(x_i) \;+\; (1 - y_i) * \log(1 \;-\; p_w(x_i))$$

For the multiclass case, this is:

$$L(w) \;=\; -\frac{1}{N}\sum_{i=1}^{N} y_i * \log p_w(x_i)$$

x is the input data, y are associated labels, $\hat{y}$ aka $p_w(x_i)$ are the predicted labels associated with each sample i, and N is the number of data points

For backpropagation to work properly, the activation function should be differentiable. This will allow the optimization functions to have bounded derivatives.

Part 1: Generate the Dataset

(6pts)

Run part1.py, which calls the `datasets.generate_nd_dataset(N, M, kGaussian, dims)` function, where `N` and `M` represent the number of data points in each class and \`dims\` represents the dimensionality of the data (including the class). Start with Gaussian-distributed data, and 100 data points in each class (2-dims) (no changes apart from uncommenting code will be needed for this).

A. Question: Describe the data. Looking at just the code alone (i.e. without generating a plot), can you tell if it is linearly separable? Why/why not? Plot the dataset using the function `show_dataset`*. Does it appear as you expected? (1pt)*

B. Question. Increase the number of dimensions to dims=3. (1pt). Is it linearly separable?

Controlling Randomness

To allow results to be reproducible, we want you to think about setting randomness in your experiments. The following C and D will affect all later parts in the rest of the assignment.

C. Question: What are some elements of randomness that could need to be controlled in a multilayer perceptron? Name 3 or more. (3pt). Note: we will only grant points to the first three you list. HINT: look at questions in later parts of the assignment to see how randomness could affect MLP-based experiments.

D. Question: Set the random seed in Part 1 D of the code. You will need to look at `random_control.py` *(1pt)*

Part 1: Total 6/6 (6/100)

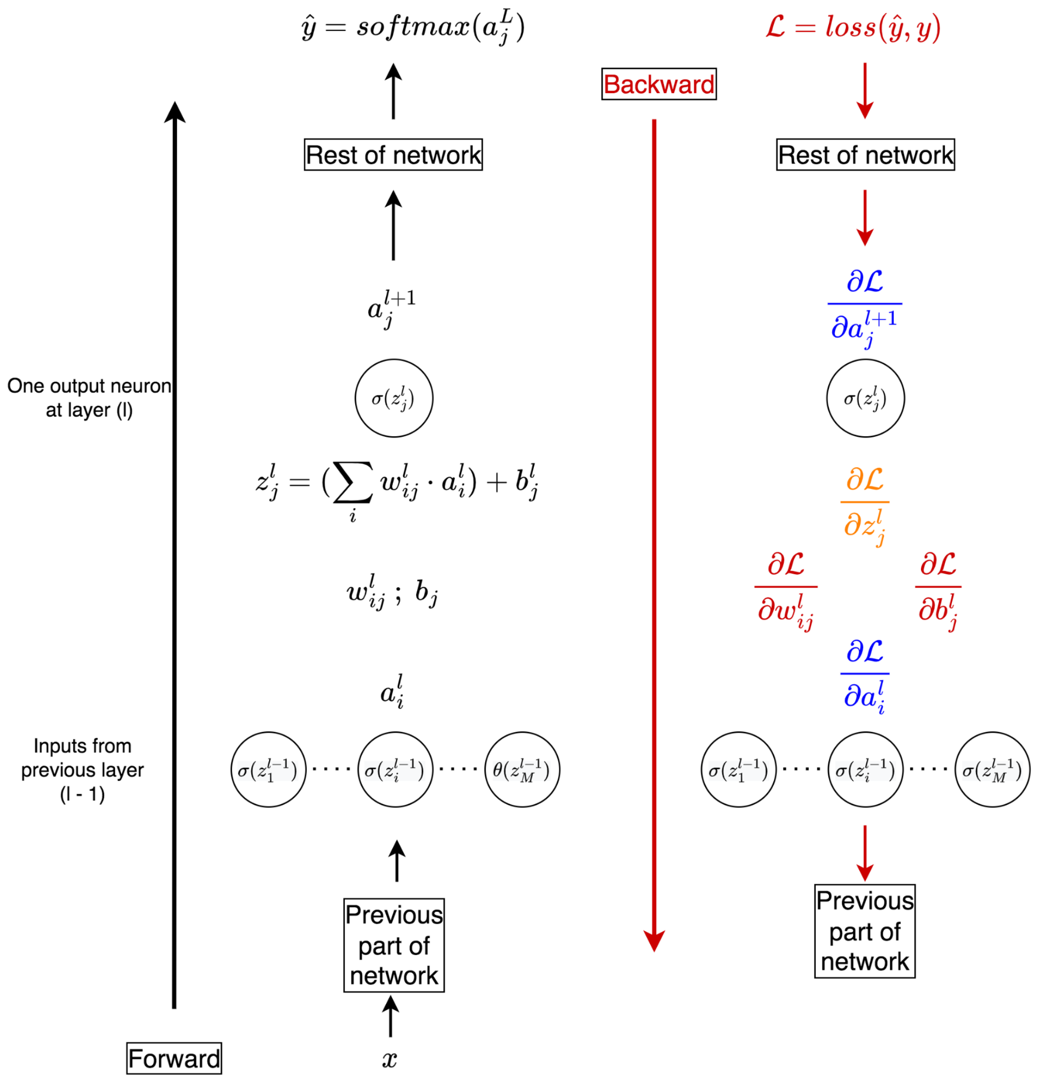
Part 2: Compute Backprop + Fill in the Code

(50pts)

To fill in the code we'll have to map some equations to code. Let's look at a diagram for a layer inspired by Lecture 10 from Professor Abu-Mostafa's Learning from Data. This formulation is not exactly the same and will use different notation.

x - input;
z - output of matrix multiplication
σ - non linearity; a - σ(z)
w_{ij}^{l} - weights at layer l for input neuron i and output neuron j
b_{j}^{l} - bias at layer l for output neuron j

$\frac{\partial \mathcal{L}}{\partial w}$ - partial derivative of loss with respect to weights



$\hat{y} = softmax(a_j^L)$
$\mathcal{L} = loss(\hat{y}, y)$
Backward
Rest of network
Rest of network
a_j^{l+1}
$\frac{\partial \mathcal{L}}{\partial a_j^{l+1}}$
One output neuron at layer (l)
$\sigma(z_j^l)$
$\sigma(z_j^l)$
$z_j^l = (\sum_i w_{ij}^l \cdot a_i^l) + b_j^l$
$\frac{\partial \mathcal{L}}{\partial z_j^l}$
$\frac{\partial \mathcal{L}}{\partial w_{ij}^l}$
$\frac{\partial \mathcal{L}}{\partial b_j^l}$
$w_{ij}^l \; ; \; b_j$
a_i^l
$\frac{\partial \mathcal{L}}{\partial a_i^l}$
Inputs from previous layer (l - 1)
$\sigma(z_1^{l-1})$ $\sigma(z_i^{l-1})$ $\theta(z_M^{l-1})$
$\sigma(z_1^{l-1})$ $\sigma(z_i^{l-1})$ $\sigma(z_M^{l-1})$
Previous part of network
Previous part of network
Forward
x

In parts A through C, do not explicitly compute the derivatives, use a general notation. See Lecture 10 from LFD for help! We will give you the derivative of the Negative Log Likelihood (NLL) loss function in the code, but you may try other loss functions for extra credit.

A. Link the partial derivative $\frac{\partial L}{\partial a_j^{l+1}}$ *to* $\frac{\partial L}{\partial a_i^{l}}$ *using the chain rule. (2pts)*

B. Show how to compute $\frac{\partial L}{\partial z_j^{l}}$ *as a byproduct of the above computation. (2pts)*

C. Show how to compute $\frac{\partial L}{\partial w_{ij}^{l}}$ *and* $\frac{\partial L}{\partial b_j^{l}}$ *from the above terms. (You will have to introduce new terms. See the next two questions to get a hint…) (2pts)*

D. Simplify your expressions for all three parts. Replace any terms that can be reduced. Use σ' *to represent the* ***derivative of the activation function****. You will need to compute this derivative for relu, tanh and sigmoid in the code. (2pts)*

E. Why do we formulate backpropagation in this manner? What property(ies) do the new terms introduced have? Feel free to use Professor Abu-Mostafa's slides for help answering this question. However, while our version is equally efficient, the backpropagation step through a layer starts and stops in a different place. If you use Professor Abu-Mostafa's slides, please describe the difference. (2pts)

We will combine all of this to implement a Multilayer Perceptron for the Gaussian dataset. The last layer should have dimensionality 2 to represent 2 classes (0 or 1). The first layer's dimensionality should be >= the dataset's dimensionality. For the Gaussian dataset where you set dims=3, the first layer should be >= 3.

Note: In the code we will be using A for the matrix of all activations a in a layer and Z for the matrix of all Z in a layer.

F. Question: Go to the mlp.py file and fill in all of the TODOs. Generate a plot with epoch on the x-axis and loss on the y-axis. Generate a plot with epoch on the x-axis and accuracy on the y-axis. The `plotting.plot_losses` *and* `plotting.plot_accuracies` *functions should help here. Hint: Intuition says loss should decrease as epochs increase. There are test functions in* `test_functions.py`. *The* `part2_tests` *function should pass after this step. (40 pts)*

Part 2: Total 50/50 (56/100)

Part 3: Comparing Dataset Distributions

(2pts)

Compare the multivariate gaussian dataset (A) with a binary split of concentric circles (B) and with the multiclass iris dataset (C). Plot the losses and accuracies of each of these datasets with a “narrow” and “short” MLP. Note: the Iris dataset has 3 classes, rather than 2 classes.

A. Question: Make a dataset of concentric circles using `kCircles`*. Conceptually, does an MLP with 1 hidden layer suffice to correctly classify all points in this 2-dimensional data set? Why or why not? (1pt)*

B. Question: Make a dataset of the Iris dataset using `kIris`*. Conceptually, does an MLP with 1 hidden layer correctly classify all points in this dataset? Why or why not? (1pt)*

Part 3: Total 2/2 (58/100)

Part 4: Plots + Hyperparameter Sweeps and their Effects

(36 pts)

Next, we explore what happens as you change various hyperparameters of the MLP algorithm. Before getting into hyperparameter variations, though, we will emphasize how we want to see these experiments' results. Throughout this class, we emphasize expressing results through plots. For each of the following experiments, plot the changing hyperparameter on the x-axis and the error rate on the y-axis. Error rate is expressed (initially) in terms of negative log likelihood.

With each performance plot, we will want to see ***error bars*** for multiple runs. For example, if you run an experiment 3x, we want to see a plot with a line indicating mean error of the 3 experiments with surrounding error bars which indicate standard deviation of those experiments. These plotting functions are provided in `plotting.py`. Please read them over to see how they are designed.

We will also want you to comment in 1-2 sentences your interpretation of what is happening given the plot.

First, regenerate your 3 datasets (gaussians, circles, stars) with a train-test split of 80% train, 20% test.

A. **Create a train-test split of all three of your datasets** (2pts)

Create a train-test split of all three of your datasets with 80% train, 20% test. You will need to look at the `datasets.py` file to create this split.

B. **Learning Rate**. *(8pts)*

Fix everything else and adjust the learning rate with successive experiments to the following values [`1e-4, 1e-3, 1e-2, 1e-1, 1e0, 1e1`]. Plot the following in a plot where the x axis is logscale and the y axis is not.

i. Visualize train and test loss against learning rate (5pts)

ii. You may have noticed that for some experiments, the number of epochs was less than the default. This is because the loss either reached nan or infinity. Why does the loss explode when the learning rate is high? (2pts)

iii. What are some disadvantages when the learning rate is low (1pt)

C. **Number of Epochs** *(8pts)*

From the previous part, examine what the best learning rate is for each of the datasets. Pass in those learning rates as fixed in these experiments in the `part4_helper.run_experiments` *function.* *Note: they may differ for each of the datasets. Fix everything else and adjust the number of epochs to the following values [*`1e1`, `1e2`, `1e3`, `1e4`*].*

i. Visualize train and test loss against number of epochs (xaxis should be logscale) (3pts)

ii. When does the loss saturate for train? For test? (2pts)

	Train	***Test***
Gaussian		
Circles		
kIris		

iii. If the loss does not converge, comment on why there is no convergence. Plot the norm of the gradients over time to visualize (1pt)

iv. Do you observe overfitting? (1pt)

v. What are some common techniques to prevent overfitting? List 3. (1pt)

D. **Batch Sizes** *(i.e. 1, 5, 10, 25, 50) (6pts)*

i. Set the batch size to [1,5,10,25,50] on all datasets. Plot the batch size vs. training error and testing error on the same plot. (3pts)

ii. What effect does a larger batch size have on the errors? (2pts)

iii. What effect does a smaller batch size have on experiment runtime? (1pt)

E. **Number of Training Samples** *(7pts)*

Regenerate the ***Gaussian-based dataset*** *and the* ***circles dataset*** *with varying numbers of training (and testing) samples. Our first splits were N=100 and M=100. Generate the datasets with [N=1000, M=100], [N=1000,M=1000], and [N=10,M=10], where dimension stays constant at dims=3 for the*

Guassian, dims = 2 for the Circles. Create an MLP for each with 3 hidden layers, where the first hidden layer has 6 nodes, the second hidden layer has 4 nodes, and the last hidden layer has 2 nodes.

HINT: `part4_helper.get_errors_simple` *will be helpful here.*

i. Visualize train and test loss against the number of samples (x-log-scale plot) (3pts)

ii. Comment on the result of the unbalanced dataset distributions [N=1000, M=100]. (2pts)

iii. Comment on how the error changes for the different datasets. Why do you suppose the error changes as it does? (2pts)

F. ***Number of Layers vs. Layer Width*** *(fat vs. skinny layers) (bonus +5)*

i. Plot training error vs. testing error for a "thin" network (i.e. MLP([6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,2]) vs. a "wide" network (i.e. MLP([20,20,20,20,2]). Set your thin network to have 15 layers and your wide network to follow an average-sized layer (as specified by some number in `part4_helper.run_layer_experiments`*) where each layer is multiplied by 5. Comment on which network does better for Circles and the Iris Dataset. Comment on any intuition behind why that is. (5pt)*

Total: 36/36 (94/100)

Part 5: Intuition for Activation Functions and Loss Functions

(11pts)

The forward-network currently uses a sigmoid activation function and the sigmoid derivative in the backpropagation step. Loss is calculated as negative log likelihood. In this module, we want you to explore what varying the gradient and varying the loss function does to the MLP's performance.

A. Question: Adjust the activation function to be a ReLU function (along with the ReLU pseudo-gradient). Comment on how/why performance adjusts. Plot the loss curves to compare using some of your plots from Part 5 to visualize this. (4pts)

B. Question: What happens when you replace the activation function to be tanh? (4pts)

C. Question: Adjust the loss function from Negative Log Likelihood to Hinge loss and to MSE. Comment on why/how performance adjusts. (3pts)

Total: 11/11 (105/100)

Assignment designed by Suzanne Stathatos and Neehar Kondapaneni