1. **Problem Description**: The following write-up was prepared by the TA of the course, Victor Ardulov. The notations might be different from what you are used to, but you do not need to follow the notation. I slightly edited some parts of the text.

   Artificial Neural Networks (ANNs) are a Machine Learning model used to approximate a function. Biologically inspired by the interconnectedness of synapses and neurons in animal brains, an ANN is can be interpreted as a connected graph of smaller functions as shown in Figure 1.

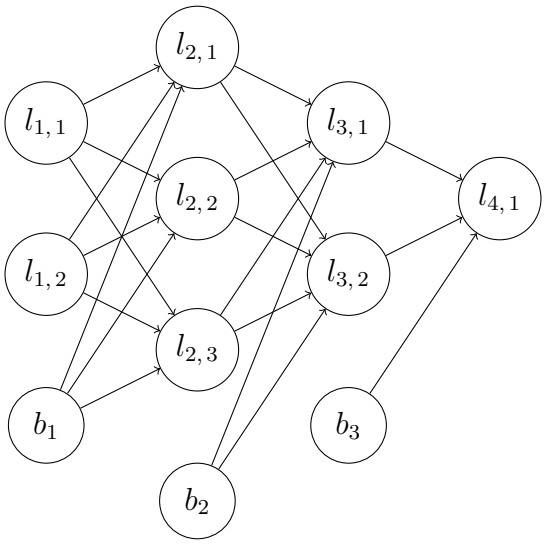

Figure 1: A Graph representing an ANN

Each node in the graph represents a "neuron". Neurons are organized into layers the first and last are known as the input and output respectively, while the remaining intermediate are known as the hidden layers. Additionally there is a "bias" neuron associated with each layer except the which serves as a constant offset in Algorithm 1. Each neuron has an "activation" function, which for the purposes of this assignment will be the Rectified Linear Unit or "relu"

$$\text{relu}(x) = \max(0, x) \tag{1}$$

Furthermore as seen in Figure 1 given a neuron in layer L, it takes as input the weighted sum of all outputs from neurons in the previous layer and the bias, then applies the activation function to these inputs.

By adjusting the weights in between layers, you can tune a neural network to represent different functions. This is particularly useful when you have data for which you wish to describe the relationship, but lack a fundamentally motivated system of equations.

The question then becomes, how can we use data to "learn" the appropriate weights to apply in the neural network. This question remained a challenge which confronted neural network imp limitation for decades until the **backpropagation** algorithm was demonstrated to robustly optimize the weights.

In this lab you will be asked to implement the backpropagation (BP) algorithm along with some auxiliary functions that will be necessary to successfully implement it.

BP is a method for calculating the gradients of an objective function J, which compares the "distance" between the predicted output of the neural network with the desired outputs. Then by computing the derivative of J with respect to the parameters in the network, we are able to use this objective function (often referred to as the loss) to iteratively update the ANN's weights. The key to BP is "propagating" the loss back through the layers of the network so that the weights of hidden layers can be appropriately updated.

Let us define an ANNs output using the following algorithm with L layers:

Algorithm 1 $f_{W,b}(x)$

$\hat{y} \leftarrow \text{relu}(x)$
for $l \in [0, L]$ **do**
  $z_l = w_l \cdot \hat{y} + b_l$
  $\hat{y} = \text{relu}(z_l)$
end for
return $\hat{y}$

The BP algorithm roughly follows:

Algorithm 2 `Backpropogation`($f_{(\theta,b)}$, X, Y, n, α)

$m \leftarrow |X|$
for $i \in [0, 1, \ldots, n]$ **do**
  $\hat{Y} \leftarrow f_{\theta,b}(X)$
  $\partial A_l \leftarrow \frac{\partial C}{\partial \hat{Y}}$
  for $l \in [0, |\theta|]$ **do**
    $A_{l-1} \leftarrow \text{relu}(z_{l-1})$
    $\partial z_l \leftarrow \partial A_l \cdot \partial \text{relu}(z_l)$ ▷ chain rule
    $\partial w_l \leftarrow \partial z_l \cdot A_{l-1}^T$
    $\partial b \leftarrow \frac{1}{m} \sum \partial z_{l,i}$
    $\partial A_l \leftarrow w_l^T \cdot \partial z_l$

    $w_l \leftarrow w_l - \alpha(\partial w_l)$
    $b_l \leftarrow b_l - \alpha(\partial b)$
  end for
end for

For this lab your specific tasks are to implement the following 3 functions in `lab5.py`

(a) For this lab the loss function we will be using is the mean squared error (MSE) which is defined as:

$$J(Y, \hat{Y}) = \frac{1}{2} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{2}$$

Variable	Definition
W	collection of weights for a particular neural network organized into layers $[w_1, w_2, \ldots, w_L]$
w_l	collection of weights associated with layer l
b	collection of biases associated with a neural network
b_l	bias associated with a particular layer
z_l	pre-activation output for a particular layer

Table 1: Definition of certain variables pertaining to the BP algorithm

(You have seen $\hat{Y}$ as the output vector of the neural network **a**.)

The implementation of which can be found in `lab5_utils.py` under the `log_loss` function, you are asked to implement the partial derivative (gradient) function $\partial J/\partial \hat{Y}$:

$$\frac{\partial}{\partial \hat{y}_i} J(Y, \hat{Y}) = -[(y_i - \hat{y}_i) \,|\, \forall y_i \in Y] \tag{3}$$

under the function `d_mse` in `lab5.py`

(b) First derive and then implement the function $\partial \text{relu}/\partial x$ in the `d_relu` function in `lab5.py`.

(c) Implement the BP algorithm in the function `train` which accepts an ANN in the form of an `ArtificialNeuralNetwork` which is found in `lab5_utils.py`, `training_inputs` which is a 2-D numpy array where each row represents a feature and each column represents the sample, `training_labels` contains the subsequent positive real values, `n_epochs` which defines the number of passes over the training data which will be done and `learning_rate` which is the rate at which the weights and biases will be updated (α in the above pseudo-code)

Besides the implemented code, please be sure to commit the images that will be produced by the script as well as a small write up txt or PDF where you discuss how you believe the number of iterations, architecture, data, and learning rate each impact the model training.

As usual only code in `lab5.py` will be graded a test file is provided to you with some but not all test/grading scripts. Your assessment will be 60pts for soundness and 40pts for code correctness.

Extra Credit 1: (20 pts) Implement the function `extra_credit` where you train multiple models modulating the architecture (number and size of the layers), the learning-rate, and the number of iterations (at least 5 times each). Return the losses as a list of lists, commit the generated plots, and in your write up add the combination that you found had the highest test set accuracy.

Here is how your code will be graded:

- General soundness of code: 60 pts.
- Passing multiple test cases: 40 pts. The test cases will be based on different splits of the data into training and testing.

2. **Extra Credit 2** (20 pts):

   The goal here is to use a famous Machine Learning package in Python called sklearn. You are recommended to use Jupyter Notebooks and run your code and submit the notebook with the results.

   (a) Download the concrete compressive strength dataset from UCI Machine Learning Repository:

   `http://archive.ics.uci.edu/ml/datasets/Concrete+Compressive+Strength`

   (b) Select the first 730 data points as the training set and the last 300 points as the test set.

   (c) Use sklearn's neural network implementation to train a neural network that predicts compressive strength of the network. Use a single layer. You are responsible to determine other architectural parameters of the network, including the number of neurons in the hidden and output layers, method of optimization, type of activation functions, and the L2"regularization" parameter etc. Research what this means. You should determine the design parameters via trial and error, by testing your trained network on the test set and choosing the architecture that yields the smallest test error. For this part, set early-stopping=False.

   (d) Use the design parameters that you chose in the first part and train a neural network, but this time set early-stopping=True. Research what early stopping is, and compare the performance of your network on the test set with the previous network. You can leave the validation-fraction as the default (0.1) or change it to see whether you can obtain a better model.

   Note: there are a lot of design parameters in a neural network. If you are not sure how they work, just set them as the default of sklearn, but if you use them masterfully, you can have better models.

Note: You must submit Extra Credit along with your main homework. If you submit Extra Credit Late, your whole homework will be late.