Part 1: Multi-Layer Perceptrons (9 points)

The file `iml/dl.py` defines a basic framework for building multi-layer models. The `forward` method of a `Module` is meant to implement a forward pass, an arbitrary operation mapping some inputs to some outputs. The `backward` method is meant to carry gradients backwards and perform parameter updates.

More specifically, if the `forward` method of a `Module` produces an output y from an input x and an internal parameter θ, then the `backward` method should receive a gradient $\partial L/\partial y$, perform some parameter update based on $\partial L/\partial \theta$, and finally return the gradient $\partial L/\partial x$. See the implementation of the `Linear` module for an example.

`Module`s admit a natural kind of composition which I have implemented in the `Sequential` class. Specifically, `forward` methods compose as functions in the usual way, and `backward` methods are composed in reverse. As long as you've defined `backward` correctly for all sub-modules, `Sequential.backward` automatically carries out the back-propagation algorithm.

The goal of this problem is to use the `Module` system to train an multi-layer perceptron (MLP) on the MNIST handwritten character dataset. Do not use any libraries besides `numpy` and `iml/` except when explicitly requested.

(a) Using the `mnist_iter()` method from `dl.py`, load and display some datapoints from MNIST. Using `LogisticRegression` from sklearn, train a baseline model on 2000 datapoints from the training set and report its accuracy on the test set. (1.5 points)

(b) Write a `ReLU` module that applies the ReLU activation. (Remember that ReLU replaces negative inputs with zero and passes positive inputs through.) Explain how you derived the backward pass. (2 points)

(c) Using `Linear` and `ReLU` modules, build some kind of multi-layer perceptron and train it on MNIST. Ensure your model has at least 95% accuracy on the test set. (3.5 points)

(d) **Bonus**: Still using only `numpy` and `iml/`, design a MLP that achieves at least 98% test accuracy and trains in a short amount of time. (Say: under 5 minutes on a median consumer CPU). You may consider using a library like optuna. (2 points)

Part 2: Linear Autoencoders (8 points)

In this exercise you'll implement a simple *autoencoder*. An autoencoder has two parts: an *encoder* f which sends an input $x \in \mathbb{R}^d$ to a code $f(x) \in \mathbb{R}^k$, and a *decoder* g which sends a code back to a vector in $\mathbb{R}^d$. The reconstruction loss of an autoencoder is an expectation like

$$L = \mathrm{E}_X \|g(f(X)) - X\|^2$$

where X is a draw from the dataset. Sometimes the code $f(X)$ is adjusted by some random operation before being sent to g, in which case reconstruction loss is an expected value over the random perturbation. In this exercise, we measure reconstruction loss as mean squared Euclidean distance between autoencoder output and our *original data* (without applying normalizations!).

As in Part 1, do not use any libraries besides numpy and `iml/dl.py`. For this part you'll need the `data/ulu.csv` file from the homework repository.

(a) Let's start by training the model `Linear(3, 3)` on `ulu.csv` with the reconstruction loss objective. Initialize the weight matrix randomly (this is the default behavior for `Linear`) and run full-batch gradient descent using the raw data as input for 50 iterations. Does there exist any learning rate for which gradient descent has a good rate of convergence? (1 point)

(b) Make an autoencoder that encodes the datapoints of `ulu.csv` as two-dimensional vectors. Minimize its reconstruction loss (in squared Euclidean distance) using gradient descent. Report the final reconstruction loss and confirm that it is close to the theoretical minimum. (2 points)

(c) Implement a `DropoutOne` module whose forward pass zeros out the first dimension of each input vector with independent probability 0.5. What is the correct way to implement a backwards pass for this module? (1.5 points)

(d) By incorporating `DropoutOne` into the forward pass, train an autoencoder in such a way that the first dimension of the code reliably distinguishes the foreground and background parts of `ulu.csv`. Ensure that your model converges approximately within 500 iterations of full-batch gradient descent and log its reconstruction loss. How can you determine the optimal expected reconstruction loss for this model? (3.5 points)

Part 3: Bouba–Kiki (7 points)

Almost everyone has an intuitive grasp for the concepts of roundness and sharpness. In fact, it's possible to talk about roundness and sharpness without reference to a pre-established language; in 1929, Wolfgang Köhler showed that subjects with different cultures and languages usually agree on how to relate the nonsense words *bouba* and *kiki* with round and sharp shapes. In this exercise, you'll investigate whether a small image model (MobileNetV3-small) has a notion of roundness and sharpness that generalizes from abstract illustrations to real objects.

You will work with a pretrained model in this exercise, and you are not expected to do any optimization yourself! However, it may come in handy to use some tools from scikit-learn.

The file `data/bouba_kiki.npz` contains a dataset of 200 100 × 100 black and white images, which can be loaded with `numpy.load`. Half have round shapes, and the other half have sharp shapes.

(a) Load mobilenet_v3_small using the `torchvision` package. Briefly describe the idea of a convolutional linear layer. Identify one convolutional layer inside MobileNetV3 and count how many parameters it has. (1 point)

(b) Without finetuning and without using the classification head, find a way to use MobileNetV3-small to separate the images in `bouba_kiki.npz` into a bouba class and a kiki class. Confirm that your technique works by showing a few examples of the classification. (I recommend upscaling the images to 224×224 with `Resize` from `torchvision.transforms` before passing them to the model.) (2.5 points)

(c) Fetch some images of frogs and airplanes from the CIFAR100 dataset. Using your method from the previous problem, determine whether frogs or airplanes are more kiki. Do you agree with the model's judgment? (1.5 points)

(d) **Bonus**: Can you identify the earliest layer of MobileNetV3 that distinguishes between the bouba and kiki examples in our dataset? Describe your approach. (2 points)