Important note about submitting code
We re-run your notebook in its entirety. So make sure that all your code cells run, from top to bottom. Any code cell that crashes will stop us from re-running your notebook. Also, we extract the class and function definitions and autotest them. To facilitate this extraction process, please ensure that:

- all `import` commands are in a single code cell (at the top of the notebook), and
- code cells that contain function definitions and/or class definitions should not include any lines of code that are not part of the definition.

Automatic Differentiation

Question 1: Auto-Differentiation by Hand

[4 marks]

Construct an expression graph for

$$L = w\,\sigma\left(x_1 m_1\right) + w\,\tanh\left(x_2 m_2\right) + b$$

where σ is the logistic function. Use your expression graph to compute an explicit formula for the partial derivatives of L with respect to each of its dependent variables (w, x_1, x_2, m_1, m_2, and b). **Label each operation in the graph with its derivative**, and **label each variable (including intermediate variables) with the partial derivative of L with respect to it**. You can (and should) use intermediate variables in your answer.

Question 2: Backprop using Auto-Differentiation

[8 marks]

Download the following and place in a single folder:

- `a03q2_YOU.ipynb`
- `matad.py`
- `utils.py`.

The module `matad.py` is like `ad.py`, except that it works with matrices instead of scalars; it defines the `Mat` class, and associated `MatOperation` classes.

Note that the `MatOperation` functions return `Mat` objects, but their `backward` functions deal with NumPy arrays, not `Mat` arrays.

The jupyter notebook `as04_q3.ipynb` creates and tries to train a neural network on a simple dataset. However, some critical parts of the code are incomplete. Complete the implementation by doing the following:

(a) `backward`: Complete the implementation of the `backward` method in the `Mul` class. As the documentation states, the `Mul` class implements matrix-matrix multiplication. The `backward` method takes a 2D NumPy array as input, applies its own term to the chain of derivatives, and sends those derivatives (NumPy arrays) to the `backward` function of each of its arguments. You will probably find the following theorem useful.

> **Theorem:** Consider $L(Y) \in \mathbb{R}$, for $Y \in \mathbb{R}^{D\times N}$. That is, $L : \mathbb{R}^{D\times N} \to \mathbb{R}$. Let $Y = H \cdot W$, where $H \in \mathbb{R}^{D\times M}$, $W \in \mathbb{R}^{M\times N}$, and $\cdot$ refers to the matrix product. Then $\nabla_H L = \nabla_Y L \cdot W^{\mathrm{T}}$, and $\nabla_W L = H^{\mathrm{T}} \cdot \nabla_Y L$.

(b) `__call__`: Complete the implementation of the `__call__` function in the `Connection` class. This class represents the connection weights and biases between two `Population` layers. The `__call__` function takes the activity of the layer below, multiplies it by the connection weights, and adds the bias, and returns the resulting input current (as a `Mat` array).

Hint: Take advantage of the properties of the `Mat` and `MatOperation` classes. If you do it properly, your solution to part (c) will be much easier.

(c) `learn`: Complete the `learn` function in the `Network` class. To get full marks, you must use the automatic-differentiation functionality of the `Mat` and `MatOperation` classes. Notice that the `Network` class has a method called `parameters()` that returns a list of all the `Mat` objects in the network that correspond to connection weights and biases.

There is some code at the end of the notebook that creates a network and runs `learn` on a the simple dataset. If your code works, you should see it learn to classify the dataset correctly, with accuracy better than 98%.

Be sure to make your code readable, and include informative comments.

What to submit

Question 1: You can:

- typeset your solutions using a word-processing application, such as Microsoft Word, LaTeX, Google docs, etc., or
- write your solutions using a tablet computer, or
- write your solutions on paper and take photographs.

In any case, it is **your responsibility** to make sure that your solutions are sufficiently legible.

Submit your solution to Crowdmark only. Crowdwark will accept PDFs or image files (JPEG or PNG). You may submit multiple files for each question, if needed.

Question 2: You must submit your jupyter notebook in two places: **Crowdmark**, and **D2L**.

Crowdmark: Export your jupyter notebook as a PDF, and submit the PDF to Crowdmark.

D2L: Submit your `ipynb` file to Desire2Learn in the designated dropbox. You do not need to zip the file.

Make sure you submit your solutions, and not just a copy of the supplied skeleton code. We suggest you rename the `ipynb` file by replacing "`YOU`" with your WatIAM ID (eg. `a03q2_jorchard.ipynb`). You should **not** submit `matad.py` or `utils.py`.