ML assignment 2:
Lovey Dovey Data

Imports:
Pandas, numpy, sklearn, tensorFlow and matplotpy.

The training of the classifier was done using python and its tensorflow toolset mainly, through an online-executable .ipynb Notebook and several packages to manage the data, process it and manipulate it.

Although many approaches were taken, many didn't result in performances significantly better than random classification (in a binary balanced dataset, that is 50% accuracy):

- sklearn native LogisticRegression, MLPClassifier, SGDClassifier classes and methods
- Jax-powered gradient descent (with neg log loss cost function) optimization algorithm
    - Following indications found in https://coderzcolumn.com/tutorials/artifical-intelligence/guide-to-create-simple-neural-networks-using-jax#2
- Sequential Keras model with different batch sizes and layer configurations:

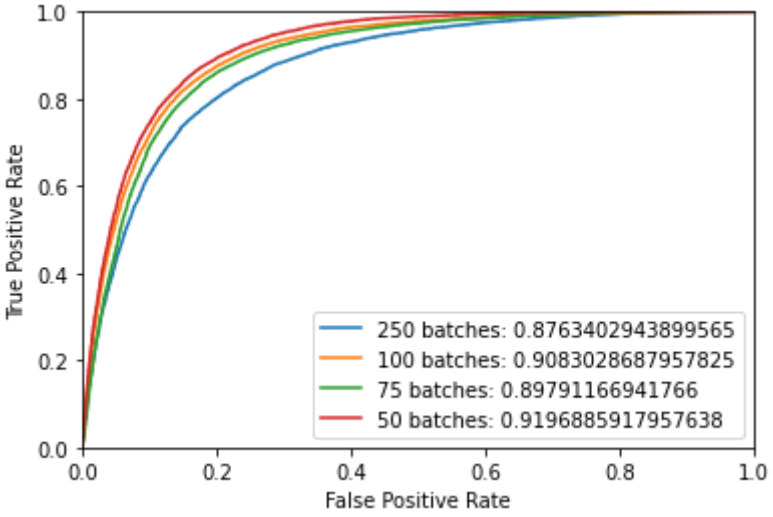
Accuracy	batch size	max epochs	layer configuration
65%	256	500	5-10-15-1
63%	128	1000	5-10-15-1
61.19%	128	750	5-5-5-1
60%	128	1000	5-10-10-15-15-1
59.90%	32	500	5-10-15-1
57.88%	32	250	5-5-5-1

No relevant results were achieved until the number of nodes per layer was augmented to a magnitude of 1000 nodes per layer and so the parameters that were being previously prodded were proved to be inconsequential indeed, in the scale they were being fine tuned.

Then, for 2 layers, the batch size and epoch number were similarly adjusted to achieve accuracy ratings of 80-100% (in the training set that is, corresponding to 80% of the original available data). At this point it became a matter of avoiding overfitting and/or reducing the time necessary for the model to be trained, ideally. Increasing the number of layers to 3 also aided somewhat in that regard but finally the chosen configuration was

256 batch size, 50 max epoch, 2 hidden layers, 1000 nodes/layer
for a 0.86% for testing data (and 0.91 for training data)

The activation function chosen for all hidden layers was relu and the output one sigmoid, as this was a binary classification task. For the same reason, the loss function was set to binary cross-entropy (the optimization method Adam with a lr = 10^{-3} was left untouched).



AUC scores for each batch size show that 50 batches' trained model seems to be the best one and indeed appears to approach the 1.0-0.0 rate between TP and FP predictions the most, being the top-most line to be shown, in red, over all the rest. An important consideration to take into account is that the model never converged so the max epoch n always stood for the actual number of periods the algorithm would run for, and therefore a really important feature to manipulate and explore the effects of.
[batch size 25 isn't shown in the plot but its AUC score was 0.874, worse than any shown]

Further efforts were made to implement a manner of Mixture of Experts algorithm as well that would combine the outputs of different models, learnt from different segments of the original data, to try and reduce the possibility of overfitting, but this attempt, at least in the low level of sophistication that was tried with, didn't show results any better than the aforementioned selected model. The gate function attributed every expert the same weight, always, and the sample was labeled as the majority of the experts predicted (y1+y2+...+yi > i/2) without ever evolving or changing the influence of each of them respect the others' over the final decision. As stated, even while achieving seemingly reasonable performances for the models individually, this style of combining them didn't have the expected effect or improved in any way the previous model, so the idea was discarded altogether at the end.