Problem 1 Information theory and data compression

This problem set addresses information theory and lossless data compression. The first set of problems are theoretical only,

1a) Find the entropy for the following distributions:

- $P(X = x) = 1/2$ for $x \in \{0, 1\}$
- $P(X = 0) = 0$, $P(X = x) = 1/2$ for $x \in \{1, 2\}$
- $P(X = 0, Y = 0) = 1/2,\ P(X = 0, Y = 1) = 1/4,\ P(X = 1, Y = 0) = 1/8,\ P(X = 1, Y = 1) = 1/8$
- $P(X = n) = 2^{-n},\ n \in \{1, 2, 3, \dots, \infty\}$

1b) Let $X, Y \in \{0, 1\}$. Let $P(Y = 0) = 1/4, P(X = 0|Y = 0) = 1/2$ and $P(X = 1|Y = 1) = 1/2$. Compute $H(X),\ H(Y),\ H(X, Y),\ H(X|Y),\ H(Y|X)$ and $I(X; Y)$.

1c) (Hard) A *Markov chain* is a sequence $X_1, X_2, X_3, \dots$ of random variables that has the following property:

$$P(X_n|X_{n-1}, X_{n-2}, X_{n-3}, \dots) = P(X_n|X_{n-1})$$

for all n. We can describe a Markov chain completely by its initial probabilities $P(X_1)$ and transition probabilities $P(X_n|X_{n-1})$. Let $X \in \{0, 1\}$ and

$$\begin{aligned} P(X = 0) &= 1/2 \\ P(X_n = 0|X_{n-1} = 0) &= 7/8 \\ P(X_n = 1|X_{n-1} = 1) &= 7/8 \end{aligned}$$

Find the *steady state* distribution for X, which is the probability $P(X_n = x)$ for any n. Then compute the *entropy rate* of the sequence $X_1, X_2, X_3, \dots, X_n$ as n goes to infinity.

The next problems involve numerical exercises using Python. There are three data files named `data[123].txt` that shall be used for the problems below. They are all text files with a single 0 or 1 per line. You can easily load the files with `numpy.loadtxt` (see NumPy manual).

1d) Estimate the entropy for each of the three files.

1e) Estimate the entropy for each of the three files when considering two, three and four bits at the time.

1f) Attempt to estimate the entropy rate for each of the three files. Note that you will need to approximate the rate by some finite symbols length n which is up to you to determine (Note: The choice of n is important as increasing n arbitrarily will lead to a entropy rate equal to zero. Why?).

1g) Try and model each of the three data files as a Markov chain. Estimate the transition and steady state parameters and compute the entropy. Compare with the entropy rate from the previous problem. Which problem is likely to come from a binary Markov chain?

A *run length code* can be defined as follows. Given a bit sequence

$$S = 1110011111001111110000$$

we can code it as

$$C = 0325254$$

which can be interpreted as "zero 0s, three 1s, 2 0s, 5 1s, ..." etc. We have assumed that we always start counting the zeros (here there are zero of them). We also need to determine what the maximum length we can represent is, and how we will represent sequences longer than this. Here we use the convention that if a sequence of eg. zeros are larger than some maximum N, we write the code as $N0M$ where M is the remainder of the symbols.

1h) Implement a run-length code in Python and code the file `data3.txt`, using

$$N \in \{7, 15, 31, 63, 127\}$$

What is the coding gain for those three cases? How does it compare with the entropy rate?

1i) The file `data4.bin` contains a version of the binary data from `data3.txt` where the bits have been packed into bytes and stored as a binary file. Use any compression program of you choice (Zip, GZip, BZip2, 7Zip etc.) to compress the file and compare the results with the entropy rate.

Problem 2 Deep neural networks

This problem is designed to be very open and is aiming to be similar to the process of training a deep neural network from scratch.

You will find the file `problem2_template.py` attached. This file contains a bare bones script for training a very simple classifier on the CIFAR 10 datasets (`https://www.cs.toronto.edu/~kriz/cifar.html`). This is a classification task that has ten classes - 'plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship' and 'truck'.

We want to you create a DNN classifier and use it to do image retrieval on the test set. For example, if you decide to retrieve images of dogs, you should measure the accuracy, precision and recall for this task. We also want you to use the AUPRC (area under the PR curve) to rank the different systems you make.

In your task you are free to add new layers, increase the number of nodes per layer, add dropout and change the optimization algorithm if so be. The dataset is small, but the training will take some time, so you will need to make sure you pick the lowest hanging fruits first.

The answer to this problem should of course be the precision-recall curve for the retrival problem, but also a description of how you improved the original neural network with the relevant results in tables and graphs.

As a starting point you should to the following:

1. Run the script and make sure it works. You will need the two Python modules `torch` and `torchvision`. Make a note of the time it takes to finish the training, and adjust the number of epochs to make your turn-around between experiments quick enough.

2. Divide the training set into a training and validation set. You can do this manually or using tools from `scikit-learn`.

3. Add layers to the neural network and observe the performance on the validation set.

4. Use `scikit-learn` to compute PR curves for the retrieval problem and compare the results.

We are aware that training the model may be slower for some groups than others depending on the beefiness of their hardware (gamers with NVidia cards will have an edge), but the point of this problem is not as much to get the absolutely best system as to work the problem with the resources you have. This is true for any machine learning problem.