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 are related to rate distortion theory and involves numerical simulations in Python. It is of course possible to use Matlab and other languages, but the resources mentioned in the problem set will be Python only. We will look at the quantization of samples drawn from a normal distribution X with mean $\mu = 0$ and variance $\sigma^2 = 1$.

1d) Plot the rate-distortion function for the information source.

1e) Create different uniform quantizers, where you vary the number of quantization levels and the quantization range. You should at least use one, two and three bits per sample. Simulate the quantizers and plot the results alongside the $R(D)$ curve from the previous problem.

1f) Try and make a two-bit non-uniform quantizer that outperforms the uniform quantizer of the same rate.

1g) In this problem we will use a vector quantizer. This is a quantizer that looks at vectors from X^n and map them to a symbol from an alphabet $\mathcal{A}$. Here we will look at alphabets on the form $\mathcal{A} = \{0, 1, 2, \dots, 2^{n-1}\}$, meaning symbols represented by n bit words. You can find a vector quantizer in SciKit Learn (sklearn.cluster.KMeans).

Use the vector quantizer to create code books for rates $R = 1, 2$ and 3 bits per sample. Plot alongside the $R(D)$ function and compare with the uniform quantizer results.

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.