Assignment 5
Machine Learning

Please justify all answers. Simply giving the correct answer is not enough.

Problem 1 (10 points)**.**
`Problem 1.ipynb` shows how to use RNNs for sentiment analysis. The dataset consists of movie reviews from IMDB. Each review is a piece of text and is labelled with a 0 or 1 indicating whether the sentiment in the review is negative or positive. The original dataset can be found *here*. However, the data needs some preprocessing, in particular, every movie needs to be converted to a sequence of numbers so that it can be processed by a neural network. In general, this can be done using a *text vectorization layer*. However we don't need to do so in this case since the function `imdb.load_data` returns the data after this kind of preprocessing has already been done.

(a) What does the embedding layer do? What is the shape of the output of the embedding layer? *Hint: use Google search.*

(b) Why don't we need to flatten the output of the embedding layer before feeding it to the SimpleRNN layer?

(c) Replace the SimpleRNN layer by an LSTM layer or a GRU layer with the same number of units and retrain the model. How much improvement do you get? Which of the three SimpleRNN, LSTM or GRU is the best in this case?

Note. *This dataset is not ideal for testing RNNs, LSTMs or GRUs since there are no long term dependencies. So, don't expect too much from this exercise apart from experience in using these in Keras.*

Problem 2 (10 points)**.**
In this exercise we will compare feedforward neural networks with convolutional neural networks on the cifar10 dataset. You can find the description of the dataset at `https://www.cs.toronto.edu/~kriz/cifar.html`. Start with the code in `Problem 1.ipynb`.

- Train a feedforward neural network and evaluate its performance on the training and test sets. You can experiment with the number and size of the hidden layers to see what works best.

- Train a convolutional neural network and evaluate its performance on the training and test sets. Remember to include dense layers towards the end. Experiment with different combinations of 2D convolutions, max-pooling and dropout.

Note that the last layer in either case should look like this:

```
model.add(Dense(units=10))
model.add(Activation('softmax'))
```

We suggest using Google Colab with a GPU runtime to speed up the training. About 50 epochs of training should suffice. Use `categorical_crossentropy` as the loss function in `model.compile`. You can plot the training and validation loss by using code like this:

```
import matplotlib.pyplot as plt
h = model.fit(...)
plt.plot(h.history['loss'])
plt.plot(h.history['val_loss'])
plt.show()
```

You can use `model.evaluate` to evaluate the models.

Problem 3 (20 points)**.**
`Problem 3.ipynb` demonstrates how to learn the output of a sequence of numbers. In this case, the sequences are of length 2 and have the form $[x, y]$ where $x, y \in \{0, \cdots, p-1\}$ and the output is $(x - y) \bmod p$ for some integer p. Note that we do one-hot encoding of the outputs.

1. Use this idea to train a network which given a sequence of integers $[x_0, \cdots, x_{k-1}]$ where $x_i \in \{0, 1, \cdots, p-1\}$ outputs $(x_0 - x_1 + x_2 - \cdots + (-1)^{k-1} x_{k-1}) \bmod p$. You can assume that the input sequences have length at most 5 and any shorter sequence can be padded with 0's so that it has length 5.

2. Any text can thought of as a long string of characters and if we map each character to an integer we can think of the text as a sequence of characters. In particular, if a text T of length n has m distinct characters then we can map it to a sequence of integers $x_0, \cdots, x_{n-1}$ where $x_i \in \{0, \cdots, m-1\}$. Once we have such a sequence, we can define a sequence prediction problem where given any k consecutive elements $[x_i, x_{i+1}, \cdots, x_{i+k-1}]$, the goal is to predict the next one x_{i+k}.

   – Use the above idea to train a network on a large text (e.g. *this text.*).
   – Use the trained model to generate text as follows: set $[x_0, \cdots, x_{k-1}]$ as you wish and the repeatedly use the model to predict the next element from the previous k. Note that the model actually predicts a probability distribution on the next element - you should pick the next element randomly according to this distribution.

Experiment with different architectures and parameters. As in the last problem, we suggest using Google Colab with GPU runtime.

Problem 4 (10 points)**.**
Suppose that the characters 'A' to 'Z' and ' ' (space) are encoded into 4-bit strings in some unknown way. You are given 10000 strings with 1000 characters (i.e. 4000 bits) each along with a label indicating whether the string contains as a substring the three letter name of one of the following agencies: ICA, GKB, SMS, SII, ARW. How would you efficiently learn a classifier based on neural networks for this problem?