1. **Generative Models for Text**

   (a) In this problem, we are trying to build a generative model to mimic the writing style of prominent British Mathematician, Philosopher, prolific writer, and political activist, Bertrand Russell.

   (b) Download the following books from Project Gutenberg `http://www.gutenberg.org/ebooks/author/355` in text format:

      i. The Problems of Philosophy
      ii. The Analysis of Mind
      iii. Mysticism and Logic and Other Essays
      iv. Our Knowledge of the External World as a Field for Scientific Method in Philosophy

   Project Gutenberg adds a standard header and footer to each book and this is not part of the original text. Open the file in a text editor and delete the header and footer.

   The header is obvious and ends with the text:
   *** START OF THIS PROJECT GUTENBERG EBOOK AN INQUIRY INTO MEANING AND TRUTH ***

   The footer is all of the text after the line of text that says:
   THE END

   To have a better model, it is strongly recommended that you download the following books from The Library of Congress `https://archive.org` and convert them to text files:

      i. The History of Western Philosophy
      `https://archive.org/details/westernphilosophy4`
      ii. The Analysis of Matter
      `https://archive.org/details/in.ernet.dli.2015.221533`
      iii. An Inquiry into Meaning and Truth
      `https://archive.org/details/BertrandRussell-AnInquaryIntoMeaningAndTruth`

   Try to only use the text of the books and throw away unwanted text before and after the text, although in a large corpus, these are considered as noise and should not make big problems.[1]

   (c) **LSTM:** Train an LSTM to mimic Russell's style and thoughts:

      i. Concatenate your text files to create a corpus of Russell's writings.
      ii. Use a character-level representation for this model by using extended ASCII that has $N = 256$ characters. Each character will be encoded into a an integer using its ASCII code. Rescale the integers to the range $[0, 1]$, because LSTM

[1] If this is a large corpus for your computer's power and it makes training LSTM hard, use as many of the books as possible.

uses a sigmoid activation function. LSTM will receive the rescaled integers as its input.[2]

iii. Choose a window size, e.g., $W = 100$.

iv. Inputs to the network will be the first $W-1 = 99$ characters of each sequence, and the output of the network will be the W^{th} character of the sequence. Basically, we are training the network to predict each character using the 99 characters that precede it. Slide the window in strides of $S = 1$ on the text. For example, if $W = 5$ and $S = 1$ and we want to train the network with the sequence ABRACADABRA, The first input to the network will be ABRA and the corresponding output will be C. The second input will be BRAC and the second output will be A, etc.

v. Note that the output has to be encoded using a one-hot encoding scheme with $N = 256$ (or less) elements. This means that the network reads integers, but outputs a vector of $N = 256$ (or less) elements.

vi. Use a single hidden layer for the LSTM with $N = 256$ (or less) memory units.

vii. Use a Softmax output layer to yield a probability prediction for each of the characters between 0 and 1. This is actually a character classification problem with N classes. Choose log loss (cross entropy) as the objective function for the network (research what it means).[3]

viii. We do not use a test dataset. We are using the whole training dataset to learn the probability of each character in a sequence. We are not seeking for a very accurate model. Instead we are interested in a generalization of the dataset that can mimic the gist of the text.

ix. Choose a reasonable number of epochs[4] for training, considering your computational power (e.g., 30, although the network will need more epochs to yield a better model).

x. Use model checkpointing to keep the network weights to determine each time an improvement in loss is observed at the end of the epoch. Find the best set of weights in terms of loss.

xi. Use the network with the best weights to generate 1000 characters, using the following text as initialization of the network:

> There are those who take mental phenomena naively, just as they would physical phenomena. This school of psychologists tends not to emphasize the object.

[2] A smarter way is to parse the whole corpus to figure out how many distinct characters you have in the corpus (the number may be less than 256, e.g., 53). One can also disregard lowercase and uppercase letters or even remove punctuation characters such as !.

[3] In Keras, you can use the ADAM optimization algorithm for speed.

[4] **one epoch** = one forward pass and one backward pass of all the training examples.

batch size = the number of training examples in one forward/backward pass. The higher the batch size, the more memory space you'll need.

number of iterations = number of passes, each pass using [batch size] number of examples. To be clear, one pass = one forward pass + one backward pass (we do not count the forward pass and backward pass as two different passes).

See `https://stats.stackexchange.com/questions/153531/what-is-batch-size-in-neural-network`

xii. Extra Practice: Use one-hot encoding for the input sequence. Use a large number of epochs, e.g., 150. Add dropout to the network, and use a deeper LSTM (e.g. with 3 or more layers). Generate 3000 characters using the above initialization and report if you get more meaningful text.

xiii. Extra Practice- **HMM**: Train a Hidden Markov Model with V hidden states and V possible outputs using Baum-Welch Algorithm (or any other modern algorithm that is available) using the Russell corpus, where V is the number of distinct words in the corpus. Note that for HMM, you NOT use character level encoding, because it may yield totally meaningless results, although the transition matrices associated with it will be way smaller (you are welcome to try it). Generate 200 words using the model and comment on its meaningfulness. Extra extra practice: can you train a higher order HMM (i.e. an HMM that assumes dependency on more than one previous state) to get a better model?

2. (Deep) CNNs for Image Colorization

(a) This assignment uses a convolutional neural network for image colorization which turns a grayscale image to a colored image.[5] By converting an image to grayscale, we loose color information, so converting a grayscale image back to a colored version is not an easy job. We will use the CIFAR-10 dataset. Downolad the dataset from `http://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz`.

(b) From the train and test dataset, extract the class birds. We will focus on this class, which has 6000 members.

(c) Those 6000 images have $6000 \times 32 \times 32$ pixels. Choose at least 10% of the pixels randomly. It is strongly recommended that you choose a large number or all of the pixels. You will have between $P = 614400$ and $P = 6144000$ pixels. Each pixel is an RGB vector with three elements.

(d) Run k-means clustering on the P vectors using $k = 4$. The centers of the clusters will be your main colors. Convert the colored images to k-color images by converting each pixel's value to the closest main color in terms of Euclidean distance. These are the outputs of your network, whose each pixel falls in one of those k classes.[6]

(e) Use any tool (e.g., openCV or scikit-learn) to obtain grayscale $32 \times 32 \times 1$ images from the original $32 \times 32 \times 3$ images. The grayscale images are inputs of your network.

[5] MATLAB seems to have an easy to use CNN library. `https://www.mathworks.com/help/nnet/examples/train-a-convolutional-neural-network-for-regression.html`

[6] Centers of clusters have been reported too close previously, so the resultant tetra-chrome images will be very close to grayscale. In case you would like to see colorful images, repeat the exercise with colors you select from `https://sashat.me/2017/01/11/list-of-20-simple-distinct-colors/` or `https://www.rapidtables.com/web/color/RGB_Color.html`. A suggestion would be Navy = (0,0,128), Red =(230, 25, 75), Mint = (170, 255, 195), and White = (255, 255, 255).

(f) Set up a deep convolutional neural network with two convolution layers (or more) and two (or more) MLP layers. Use 5×5 filters and a softmax output layer. Determine the number of filters, strides, and whether or not to use padding yourself. Use a minimum of one max pooling layer. Use a classification scheme, which means your output must determine one of the $k = 4$ color classes for each pixel in your grayscale image. Your input is a grayscale version of an image ($32 \times 32 \times 1$) and the output is $32 \times 32 \times 4$. The output assigns one of the $k = 4$ colors to each of the 32×32 pixels; therefore, each of the pixels is classified into one of the classes [1 0 0 0], [0 1 0 0], [0 0 1 0], [0 0 0 1]. After each pixel is classified into one of the main colors, the RGB code of that color can be assigned to the pixel. For example, if the third *main color*[7] is [255 255 255] and pixel (32,32) of an image has the one-hot encoded class [0 0 1 0], i.e it was classified as the third color, the (32,32) place in the output can be associated with [255 255 255]. The size of the output of the convolutional part, $c_1 \times c_2$ depends on the size of the convolutional layers you choose and is a feature map, which is a matrix. That matrix must be *flattened* or *reshaped*, i.e. must be turned into a vector of size $c_1 c_2 \times 1$, before it is fed to the MLP part. Choose the number of neurons in the first layer of the MLP (and any other hidden layers, if you are willing to have more than one hidden layer) yourself, but the last layer must have $32 \times 32 \times 4 = 4096$ neurons, each of which represents a pixel being in one of the $k = 4$ classes. Add a softmax layer[8] which will choose the highest value out of its $k = 4$ inputs for each of the 1024 pixels; therefore, the output of the MLP has to be reshaped into a $32 \times 32 \times 4$ matrix, and to get the colored image, the RGB vector of each of the $k = 4$ classes has to be converted to the RGB vector, so an output image will be $32 \times 32 \times 3$. Train at least for 5 epochs (30 epochs is strongly recommended). Plot training, (validation), and test errors in each epoch. Report the train and test errors and visually compare the artificially colored versions of the first 10 images in the test set with the original images.[9]

(g) Extra Practice: Repeat the whole exercise with $k = 16, 24, 32$ colors if your computer can handle the computations.

[7] Do not use the original CIFAR-10 images as the output. You must use the tetrachrome images you created as your output.

[8] Compile the network with `loss = cross_entropy` .

[9] If you are using matplotlib, you may get a floating point error because to print an image, matplotlib either expects ints in range 0-255 or floats in range 0-1. You might be having, for example 153.0 representation of 153 in your array and this is what makes matplotlib think that you are sending floats.

Wrap your array into `np.uint8()`. It will convert 153.0 into 153. NOTE that you cannot use `np.round` or `np.int` etc because matplotlib's requirement is unsigned int of 8 bit (think 0-255).