Homework 2: ML 80629A (Winter 2023)

Instructions:

- Please include your name and HEC ID with submission.
- The homework is due by 11:59pm on the due date.
- The homework is worth 10% of the course's final grade.
- Assignments are to be done individually.
- All code used to arrive at answers is submitted along with answers. You can convert your jupyter notebook or colab to pdf and upload it.

1 Fundamentals (10pt)

1. (2pt) We want to automate the task of determining whether a patient has some diseases based on their symptoms. We decided to use a neural network to solve the problem with the following choices: Either to train a separate neural network for each disease, or to train a single neural network with an output neuron for each disease, but with a shared hidden shared hidden layer. Which method do you prefer? Justify your answer.
2. (2pt) You want to create a model that takes as input an image and outputs a caption. Propose an adequate model architecture and justify your answer.
3. (2pt) What are the advantages and disadvantages of both GMMs and K-means? How do they compare to each other?
4. (2pt) Both RNNs and Autoencoders can be used to produce outputs of the same size as the input. Their power can be combined through architectures such as LSTM Autoencoder. Explain the advantage of such architecture, some of its potential applications and its limitations.
5. (2pt) For each layer, calculate the number of weights, number of biases and the size of the associated feature maps of the convolutional neural network in the table. The notation follows the convention:

- CONV-K-N denotes a convolutional layer with N filters, each them of size K $\times$K, Padding and stride parameters are always 0 and 1 respectively.
- POOL-K indicates a K $\times$ K pooling layer with stride K and padding 0.
- FC-N stands for a fully-connected layer with N neurons.

Layer	Activation map dimensions	Number of weights	Number of biases
INPUT	128 x 128 x 3	0	0
CONV-9-32			
POOL-2			
CONV-5-64			
POOL-2			
FC-3			

2 CNN (30 pts)

For this exercise, you will create a convolutional neural network to classify weather data. The dataset contains images of snow, rain, rainbow, fogsmog, dew, frost, lightning, and sandstorms. Use the notebook to answer the different questions. Feel free to add cells for more code and text.

1. (2pt) Using matplotlib, create a function that visualizes an image from the training set of each class.

2. (2pt) Implement data transformations to apply to the training and validation sets. The transformations for training set should be a composition of:

   - resizing using Resize()
   - adjusting the sharpness using (RandomAdjustSharpness()) with sharpness_factor=2.
   - Adding a RandomHorizontalFlip()
   - transforming the images to tensors
   - Normalizing($[0.485, 0.456, 0.406]$, $[0.229, 0.224, 0.225]$)

   The transformations for the validation set should be a composition of the same transformations but without the random transformations.

   You can read about other possible transformations to augment data here.

3. **Building the neural network:**

   - (3pt) *Network components (init function):* The CNN has 3 convolutional layers, followed by a linear layer.
     - Each convolutional layer has a kernel size of 3, a padding of 1.

    - ReLU activation is applied on every hidden layer.
    - Max pooling layer with kernel size of 2 and stride of 2.
    - The last layer should be a linear layer.
  - (3pt) Complete the Forward function.

4. (2pt) Define the optimizer and the loss function. You will use Adam optimizer with a learning rate of $1e-3$ and cross entropy loss.

5. **Training:**

   a) (2pt) Modify the training function so that it stores the training and validation loss.

   b) (2pt) Plot the validation and training loss values.

   c) (4pt) Suggest possible changes (at least two) that can help improve the model and implement the one of your choice. For ease of evaluation, replicate the part of the code that you are changing and add comments.

6. **Evaluation:**

   a) (4pt) Use the resulting model on the test set. What is the obtained accuracy?

   b) (3pt) What is the obtained accuracy for each class? You can use a similar function to the one used in the previous question.

   c) (3pt) Which classes have lowest accuracy? Which one has the best accuracy? What might be the reason behind this? Justify your answer. For this question, you can use code or text or both.

3 RNN (60 pts)

In this section, we will train RNNs to perform a text classification task. We will be using the dataset CoNLL2000Chunking to perform chunking over text. You can find more information about the dataset and chunking text by clicking the following link. Each sequence of words is given a tag describing its part-of-speech, as well as a chunk type. The goal is to correctly classify the chunk class given the sequence of words as input. We will use RNNs to do so. In order to determine the efficiency of our model, we will report the F-score for each chunk class (definition given later).

Each word is assigned to a chunk type which describes its role in a sentence. They can belong to a noun phrase, verb phase, prepositional phrase, etc. The complete list of basic chunk types can be found in this article. The basic chunk types all come in two flavours denoted with a prepended ”B-” or ”I-”. For example, ”B-NP” implies that a word begins a noun phrase, while a a chunk type of ”I-NP” implies that it belongs to some other part of a noun phrase. The only chunk type that has one unique tag is the chunk type ”O”, or ”word outside of any chunk”

1. **Model definition:** The RNN model is defined using the equations

$$h_t = \tanh(x_t W_{ih}^T + b_{ih} + h_{t-1} W_{hh}^T + b_{hh})$$

$$y_t = h_t W_{ho}^T$$

   where x_t is the input, h_t is the hidden layer and y_t is the output at position t in the sequence. We of course start with h_0 to be a vector of zeros.

   In this question, you will need to do the following by filling out the skeleton of the code:

   a. (5pt) Define all layers in the (__init__) of the RNN class. You are required to only have one hidden layer and one output layer in the RNN cell as in the previous equation. You will need to define two embedding layers, one for words and one for the part-of-speech tag. These two embedding should have *len(words) + 1* and *len(tags) + 1* input features (see remark below) and hidden_size //2 output features

   b. (5pt) Define the *forward* method. This method should perform all the computations of the y_t and the h_t for each position t in the training sequence.

   c. (5pt) Define the methods word_embed and target_embed that compute the word embeddings and tag embeddings given a sequence of words and a sequence of tags of the same length.

      Remark: Note that *new_token* and *new_tag* correspond to a catch-all value when we have a word or a tag that is not part of the dictionaries.

Training: In this question, you will have to do the following by filling the skeleton of the code.

a. (5pt) Use *tokens* to pass it to the word embedding layer. Use *poss* to pass it to the tag embedding layer. Concatenate the two tensors on the feature dimension (*i.e.* the last dimension).

b. (5pt) Compute the loss using *loss_fn* by using the concatenated tensors as input and *chunks* as targets. You might have to cast tensors to the same type to compute it. It can also be useful to use `.view()` on tensors to reshape them before passing them to *loss_fn*. Append the average loss over the sequence to *batch_loss*.

c. (5pt) Compute the predicted class over the sequence. You will have to use `.argmax` over the output dimension (i.e. the last one). Determine the precision over the sequence by finding which elements are predicted to the right class.Append the average precision over the sequence to *batch_prec*.

d. (5pt) Compute the gradients and update the parameters. *Remember to be careful about gradients that were computed on the previous batch...*

Testing the model: You will need to compute the F-score over each class by evaluating it on the test set.

$$F_i = 2 \cdot \text{precision}_i \cdot \text{recall}_i / (\text{precision}_i + \text{recall}_i)$$

where F_i is the F-score, precision_i the precision and recall_i the recall over class i.

The value precision_i is defined as $\frac{TP_i}{TP_i+FP_i}$ where TP are true positives and FP_i are false positives for class i.

The value recall_i is defined as $\frac{TP_i}{TP_i+FN_i}$, where FN_i are false negatives for class i. In this question, you will have to do the following by filling the skeleton of the code.

a. (0pt) Use *tokens* to pass it to the word embedding layer. Use *poss* to pass it to the tag embedding layer. Concatenate the two tensors on the feature dimension (i.e. the last dimension).

b. (0pt) Compute the predicted class over the sequence. You will have to use `.argmax` over the output dimension (i.e. the last one).

c. (5pt) Once you have the predicted and true classes for each element of the sequence, update the confusion matrix. Access each position (i, j) and add 1, where i is the predicted class and j is the true class.

d. (5pt) Compute the precision and recall using the confusion matrix. Compute the mean over rows for the precision and over columns for the recall. Using the tensors corresponding to precision and recall, compute the F-score using the equation above. Return the F-score.

e. (5pt) Complete the skeleton of code to plot the training loss over each epoch and save the plot. Also, report the final F-score on the test set.

Evaluation:

a. (5pt) Looking at the F-scores over all classes, what can you observe in term of performance on the test set? Can you suggest ideas to improve the training? You can also reference the original paper for a description of the number of each chunk in the dataset.

b. (5pt) In this notebook, the hyper-parameters were given. How should we modify the training procedure if we were to search for the best hyper-parameter combination. Could a regularization method be useuful? If so, name one that could prove to be helpful.