Intro to Speech and Audio Processing and Recognition - Exercise 3

Due Date: July 28th

Yossi Adi

July 21, 2024

In the following exercise you will implement the CTC algorithm (part 1), and train your first NN with the CTC loss (part 2).
The exercise should be done **in pairs** and is to be submitted via moodle by the deadline appearing under the submission box.
See submission guidelines for further instructions.

1 Connectionist Temporal Classification

In the first part of this exercise you will implement the CTC loss in Python. CTC calculates the probability of a specific labeling given the model's output distribution over phonemes. Formally, CTC calculates $P(\boldsymbol{p}|\mathbf{x})$ where $\mathbf{x} = [x_1, x_2, \ldots, x_T]$ is an input sequence of acoustic features, $\boldsymbol{p} = [p_1, p_2, \ldots, p_{|\boldsymbol{p}|}]$ is a sequence of transcription phonemes.

Recall, to calculate the aforementioned probability, we first set,

$$\boldsymbol{z} = [\epsilon, p_1, \epsilon, p_2, \epsilon, \ldots, p_{|\boldsymbol{p}|}, \epsilon], \tag{1}$$

where ϵ is the `blank` token. Then, we define $\alpha_{s,t}$ to be the probability of the subsequence $z_{1:s}$ after t time steps. We can calculate α using the following initialization:

$$\begin{aligned} \alpha_{1,1} &= y^1_{\epsilon} \\ \alpha_{2,1} &= y^1_{z_1} \\ \alpha_{s,1} &= 0, \quad \forall s > 2, \end{aligned} \tag{2}$$

and the following dynamic programming:

$$\alpha_{s,t} = \begin{cases} (\alpha_{s-1,t-1} + \alpha_{s,t-1}) \cdot y^t_{z_s} & z_s = \epsilon or z_s = z_{s-2} \\ (\alpha_{s-2,t-1} + \alpha_{s-1,t-1} + \alpha_{s,t-1}) \cdot y^t_{z_s} & \text{else.} \end{cases} \tag{3}$$

Instructions. Assume you are given a sequence of phonemes $\boldsymbol{p}$ and the network's output $\boldsymbol{y}$, that is, y^t_k can be interpreted as the probability of observing label k at time t. In words, $\boldsymbol{y}$ is a matrix with the shape of $T \times K$ where T is the number of time steps, and K is the amount of phonemes. Each row i of $\boldsymbol{y}$ is a distribution over K phonemes at time i.

Your goal is to implement the CTC function to calculate $P(\boldsymbol{p}|\boldsymbol{y})$ using the above equations. Your code should get 3 arguments:

1. A path to a 2D numpy matrix of network outputs ($\boldsymbol{y}$). This should be loaded using `numpy.load`.
2. A string of the labeling you wish to calculate the probability for (e.g., "aaabb" means we want the probability of aaabb).
3. A string specifying the possible output tokens (e.g., for an alphabet of [a,b,c] the string should be "abc").

Overall, your code should run with the following command: "`python ex3.py /some/path/to/mat.npy b ab`". For your convenience, we also attach an example of inputs.
Expected output: A single printed line containing only $P(\boldsymbol{p}|\boldsymbol{y})$ rounded up to 3 decimal points, you can use the following function to print your prediction:

```
def print_p(p: float):
    print("%.3f" % p)
```

It is crucial that you make sure that you only print a single line! we will be using 'diff' / auto-testing to evaluate your performance hence failing to follow the instructions could decrease your grade.

2 Neural Network Optimization using CTC

In the second part of this exercise, you will train your first NN with the CTC loss using PyTorch. For that, you will use the CTC loss from PyTorch as follows:

```
ctc_loss = nn.CTCLoss()
loss = ctc_loss(prods, trans, input_lengths, target_lengths)
```

where `probs` is the network output, `trans` is the target transcription, and `input_lengths` and `target_lengths` are the lengths of the `probs` and `trans` respectively. We additionally attached a code snippet example. You goal is the train a 2-layers neural network, on top of MFCC features, to classify digits between 0-9. Notice, you should optimize your network over a sequence of characters using the CTC loss, and not as a multi-class classification problem of digits. You are provided with both train, test and validation sets to evaluate your performance.
Your goal is to build the best classifier considering both train / test / and validation accuracies. Some ideas that you could try:

- You can explore different layer configurations: e.g., Fully connected, Conv, RNN, etc.
- You can explore different optimization algorithms: e.g., SGD, Adam, AdamW, etc.
- You can concatenate adjacent features.
- Etc.

3 What to submit?

You should submit the following files:

- A **one-page** report with your names and IDs, describing what you have found in the second part of the exercise. What network architecture worked the best for you? what feature did you use? what hyper-parameters worked the best for you? etc.
- The text files from the first part of the exercise with your output for the test Matrix.
- Python 3.9 code with your implementation that supports that running command via CLI mentioned above.