Information Extraction Project 3

Hirtika Mirghani

May 2024

Warmup

1. Prove that the posterior distribution modeled by CTC is globally normalized

To prove that the posterior distribution modeled by CTC is globally normalized, we need to show that the probability of a sequence is computed as the score of the sequence normalized by the sum of scores over all possible sequences. The formula given is:

$$p(y_1^M|x_1^T) = \frac{p(y_1^M)e^{f(x_1^T,y_1^M)}}{E_{p(y')}\left[e^{f(x_1^T,y')}\right]} \tag{1}$$

Assumptions

- **CTC Assumptions:** CTC assumes a conditional independence between the output labels given the input sequence. This means the probability of a label sequence given an input sequence can be factorized.
- **Softmax Activation:** The scores $f(x_1^T, y_1^M)$ are log probabilities obtained from the softmax layer of the network.

Proof

1. **Define the Score Function:** Let $s(y_1^M, x_1^T) = \exp(f(x_1^T, y_1^M))$, where $f(x_1^T, y_1^M)$ is the log probability of the sequence y_1^M given the input x_1^T.

2. **Posterior Probability:** The posterior probability of the sequence y_1^M given the input x_1^T is defined as:

$$p(y_1^M|x_1^T) = \frac{s(y_1^M, x_1^T)}{\sum_{y'} s(y', x_1^T)} \tag{2}$$

This ensures that the probabilities sum to 1 over all possible sequences y':

$$\sum_{y'} p(y'|x_1^T) = 1 \tag{3}$$

3. **Using the Given Formula:** The given formula expresses this as:

$$p(y_1^M|x_1^T) = \frac{p(y_1^M)e^{f(x_1^T,y_1^M)}}{E_{p(y')}\left[e^{f(x_1^T,y')}\right]} \tag{4}$$

4. **Expected Value and Normalization:** The denominator, $E_{p(y')}\left[e^{f(x_1^T,y')}\right]$, is the expected value of $e^{f(x_1^T,y')}$ over the distribution $p(y')$. This can be seen as a normalization constant ensuring that the total probability sums to 1.

5. **Connecting the Formulas:** To connect this with the standard posterior probability formula:

$$\sum_{y'} e^{f(x_1^T,y')} = E_{p(y')}\left[e^{f(x_1^T,y')}\right] \tag{5}$$

Hence:

$$p(y_1^M|x_1^T) = \frac{e^{f(x_1^T,y_1^M)}}{\sum_{y'} e^{f(x_1^T,y')}} \tag{6}$$

This matches the form given by:

$$p(y_1^M|x_1^T) = \frac{p(y_1^M)e^{f(x_1^T,y_1^M)}}{E_{p(y')}\left[e^{f(x_1^T,y')}\right]} \tag{7}$$

Conclusion

The given expression demonstrates that the posterior distribution modeled by CTC is globally normalized by showing the probability of a sequence y_1^M given the input x_1^T is computed as the score of the sequence normalized by the sum of scores over all possible sequences. This normalization ensures that the total probability across all possible sequences sums to 1, satisfying the properties of a valid probability distribution.

2. Maximizing the CTC objective maximizes a lower bound on the mutual information

To show that maximizing the CTC objective maximizes a lower bound on the mutual information $I(X;Y)$ between input and output sequences, we need to connect the CTC objective to mutual information.

Mutual Information

The mutual information $I(X;Y)$ between input X and output Y sequences is given by:

$$I(X;Y) = H(Y) - H(Y|X) \tag{8}$$

where:

- $H(Y)$ is the entropy of the output sequences.
- $H(Y|X)$ is the conditional entropy of the output sequences given the input sequences.

CTC Objective

The CTC objective is the log probability of the correct sequence y given the input x:

$$\mathcal{L}_{CTC} = \log p(y|x) \tag{9}$$

Maximizing this log probability is equivalent to minimizing the negative log probability, which can be seen as the conditional entropy $H(Y|X)$.

Lower Bound on Mutual Information

We can express the mutual information in terms of the CTC objective as follows:

$$I(X;Y) = H(Y) - H(Y|X) \tag{10}$$

Using the CTC objective, we can rewrite the conditional entropy:

$$H(Y|X) = -E_{p(x,y)}\left[\log p(y|x)\right] \tag{11}$$

Since the CTC objective $\mathcal{L}_{CTC}$ is $\log p(y|x)$, we have:

$$H(Y|X) = -E_{p(x,y)}\left[\mathcal{L}_{CTC}\right] \tag{12}$$

Maximizing the CTC Objective

Maximizing $\mathcal{L}_{CTC}$ means minimizing $H(Y|X)$. As $H(Y|X)$ decreases, the mutual information $I(X;Y)$ increases, assuming that $H(Y)$ is constant (which is typically the case if the distribution of Y does not depend on X).

Conclusion

By maximizing the CTC objective $\mathcal{L}_{CTC}$, we are effectively minimizing the conditional entropy $H(Y|X)$, which in turn maximizes the mutual information $I(X;Y)$ between input and output sequences. This maximization of mutual information indicates a better representation and alignment between the input and output sequences.

Project Overview

This project aims to build an isolated-word speech recognizer for a vocabulary of 48 words using the Connectionist Temporal Classification (CTC) objective function. The recognizer processes speech represented via quantized spectral features and Mel-frequency cepstral coefficients (MFCCs).

Project Structure

```
.
dataset.py
model.py
main.py
README.md
requirements.txt
data/
    clsp.lblnames
    clsp.trnscr
    clsp.trnlbls
    clsp.trnwav
    clsp.endpts
    clsp.devwav
    clsp.devlbls
```

Data Description

- **clsp.lblnames**: Contains 256 two-character-long label names, one per line.
- **clsp.trnscr**: Script read by speakers, with 798 lines of data.
- **clsp.trnlbls**: Processed speech labels corresponding to the utterances in the script.
- **clsp.trnwav**: Names of the waveform files corresponding to each utterance in the script.
- **clsp.endpts**: Endpoint information indicating the start and end of each utterance.
- **clsp.devwav**: Names of the waveform files for the test set.
- **clsp.devlbls**: Labels for the test set utterances.

Implementation Details

Dataset Class (`dataset.py`)

This module handles loading the data from the provided files and preparing it for training. It supports both discrete and continuous feature representations.

- **Initialization**: The `AsrDataset` class initializes the dataset by loading label names, scripts, and features from the specified files. It also handles the mapping of letters to indices for input sequences.

- **Data Loading**: The class checks if the files exist and reads them line by line, stripping any extra whitespace and ignoring title lines if present.
- **__len__**: Returns the length of the features list.
- **__getitem__**: Retrieves the spelling of the word and the corresponding features for the given index, converting them into tensors.
- **compute_mfcc**: Computes MFCC features for the waveform files listed in the provided `wav_scp` file.

Model Class (`model.py`)

This module defines the neural network model for speech recognition using LSTM layers.

- **LSTM_ASR Class**: Defines an LSTM-based model with embedding layers for discrete features.
  - **Initialization**: Sets up the network architecture, including embedding, LSTM, and fully connected layers.
  - **Forward Pass**: Processes the input features through the LSTM and fully connected layers to generate the output.
  - **Decode**: Decodes the output of the network into predicted word sequences by taking the most likely tokens at each time step.

Training and Testing Script (`main.py`)

This module manages the training and testing process, including data loading, model initialization, and training loop. It also includes functions for decoding and computing accuracy.

- **collate_fn**: Pads sequences to the same length for batch processing.
- **train**: Trains the model using the CTC loss function. It iterates through the data loader, performs forward and backward passes, and updates the model parameters.
- **decode**: Evaluates the model on the test set and generates predictions.
- **compute_accuracy**: Calculates the accuracy of the model by comparing predictions with the ground truth labels.
- **main**: Main function to load data, initialize the model, start the training process, and evaluate the model on the test set.

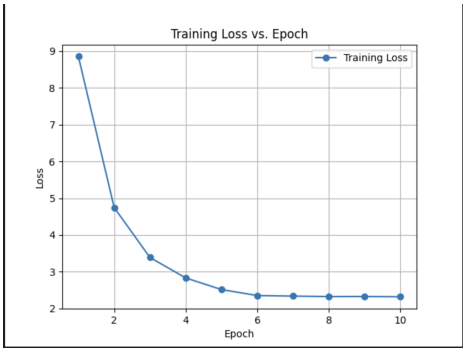

Figure 1: Training Loss vs. Epoch

Training Results

Training Loss

The training loss over 10 epochs is shown in the graph below:

Loaded Data

- 256 label names from `clsp.lblnames`
- 798 scripts from `clsp.trnscr`
- 798 features from `clsp.trnlbls`
- 393 features from `clsp.devlbls`

Training Output

```
Training Loss: 8.855269782543182
Training Loss: 4.745695620775223
Training Loss: 3.3873727416992185
Training Loss: 2.8302918243408204
Training Loss: 2.5141483449935915
Training Loss: 2.3518917465209963
Training Loss: 2.3360790157318116
Training Loss: 2.3240373134613037
Training Loss: 2.3259535789489747
Training Loss: 2.3178306865692138
```

Test Results

The test results are in the file test$_predictions.txt$.

How to Run

1. **Place Data Files**: Ensure all data files (`clsp.lblnames`, `clsp.trnscr`, `clsp.trnlbls`, `clsp.trnwav`, `clsp.endpts`, `clsp.devwav`, `clsp.devlbls`) are in the `data/` directory.
2. **Install Dependencies**:

   ```
   pip install -r requirements.txt
   ```

3. **Run the Training Script**:

   ```
   python main.py
   ```

Requirements

Here is the `requirements.txt` file for the project:

```
torch==1.10.0
torchaudio==0.10.0
librosa==0.8.1
numpy==1.21.2
matplotlib==3.4.3
```

Contarstive System Overview

This contrastive system aims to build an isolated-word speech recognizer using Mel-frequency cepstral coefficients (MFCCs) as features, in contrast to the primary system which used quantized spectral features. The recognizer is trained using the Connectionist Temporal Classification (CTC) objective function.

Project Structure

```
.
 dataset.py
 model.py
 main.py
 README.md
 requirements.txt
 data/
     clsp.lblnames
     clsp.trnscr
     clsp.trnlbls
```

```
clsp.trnwav
clsp.endpts
clsp.devwav
clsp.devlbls
```

Data Description

- **clsp.lblnames**: Contains 256 two-character-long label names, one per line. - **clsp.trnscr**: Script read by speakers, with 798 lines of data. - **clsp.trnlbls**: Processed speech labels corresponding to the utterances in the script. - **clsp.trnwav**: Names of the waveform files corresponding to each utterance in the script. - **clsp.endpts**: Endpoint information indicating the start and end of each utterance. - **clsp.devwav**: Names of the waveform files for the test set. - **clsp.devlbls**: Labels for the test set utterances.

Implementation Details

Dataset Class (`dataset`$_c$*ont.py*)

This module handles loading the data from the provided files and preparing it for training. It supports MFCC feature extraction.

- **Initialization**: The `AsrDataset` class initializes the dataset by loading scripts and waveform file names from the specified files. It also handles the mapping of letters to indices for input sequences.
- **Data Loading**: The class checks if the files exist and reads them line by line, stripping any extra whitespace and ignoring title lines if present.
- **__len__**: Returns the length of the waveform file list.
- **__getitem__**: Retrieves the spelling of the word and the corresponding MFCC features for the given index, converting them into tensors.
- **compute_mfcc**: Computes MFCC features for the waveform files listed in the provided `wav_scp` file.

Model Class (`model`$_c$*ont.py*)

This module defines the neural network model for speech recognition using LSTM layers with MFCC features.

- **LSTM_ASR Class**: Defines an LSTM-based model with linear layers for MFCC features.
  - **Initialization**: Sets up the network architecture, including linear, LSTM, and fully connected layers.

- **Forward Pass**: Processes the input features through the linear, LSTM, and fully connected layers to generate the output.
- **Decode**: Decodes the output of the network into predicted word sequences by taking the most likely tokens at each time step.

Training and Testing Script (`main_cont.py`)

This module manages the training and testing process, including data loading, model initialization, and training loop. It also includes functions for decoding and computing accuracy.

- **collate_fn**: Pads sequences to the same length for batch processing.
- **train**: Trains the model using the CTC loss function. It iterates through the data loader, performs forward and backward passes, and updates the model parameters.
- **decode**: Evaluates the model on the test set and generates predictions.
- **compute_accuracy**: Calculates the accuracy of the model by comparing predictions with the ground truth labels.
- **main**: Main function to load data, initialize the model, start the training process, and evaluate the model on the test set.

Training Results

Training Loss

The training loss over 10 epochs is shown in the graph below:

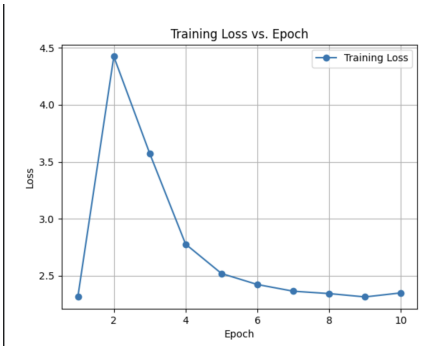

Figure 2: Training Loss vs. Epoch

Loaded Data

- 256 label names from `clsp.lblnames`
- 798 scripts from `clsp.trnscr`
- 798 features from `clsp.trnlbls`
- 393 features from `clsp.devlbls`

Training Output

```
Training Loss: 2.317989187836647
Training Loss: 4.424168062210083
Training Loss: 3.573465986251831
Training Loss: 2.777047185897827
Training Loss: 2.518873701095581
Training Loss: 2.422389259338379
Training Loss: 2.3637199783325196
Training Loss: 2.342812442779541
Training Loss: 2.3135535812377928
Training Loss: 2.3489943504333497
```

1 Test results

The test results are in test$_p redictions_c ontrast.txt$

How to Run

1. **Place Data Files**: Ensure all data files (`clsp.lblnames`, `clsp.trnscr`, `clsp.trnlbls`, `clsp.trnwav`, `clsp.endpts`, `clsp.devwav`, `clsp.devlbls`) are in the `data/` directory.
2. **Install Dependencies**:

   ```
   pip install -r requirements.txt
   ```

3. **Run the Training Script**:

   ```
   python main_cont.py
   ```

Requirements

Here is the `requirements.txt` file for the project:

```
torch==1.10.0
torchaudio==0.10.0
librosa==0.8.1
numpy==1.21.2
matplotlib==3.4.3
```

2 Additional Notes

:

Have referred to Chatgpt, StackOverflow and Github.