Name: Sai Venkata Naga Saketh Anne
USC Email: annes@usc.edu
USC ID: 3725520208
CSCI 544 – Applied Natural Language Processing
Homework 4 Report

Task 1:

Explanation of the code:

1. Data processing:
The code first defines paths to input data files to store information about the vocabulary, tag frequencies, and individual lines of data. It initializes various data structures, such as lists and dictionaries.
2. Reading Input Data:
It divides each line into words and processes each word as it comes across, reading the input data line by line from a file. It refreshes tag frequency dictionaries and vocabulary as it moves along the lines.
3. Sentence Segmentation:
This method divides the input lines into sentences by looking for blank lines, which signify the conclusion of each sentence.
4. Indexing Words and Tags:
To map words and tags to distinct indices, respectively, it generates dictionaries (w2i and t2i). The model will represent words and tags as numerical values using these indices.
5. Tensor Representation:
It uses the word-to-index and tag-to-index dictionaries that were previously constructed to transform each sentence and the tags that go with it into tensor representations.
What the BLSTM Model Is:
It defines a PyTorch model class called BLSTM for sequence tagging tasks inherited from nn.module and represents a Bidirectional LSTM network.
6. Model Training:
It defines the loss function (Cross Entropy Loss) and optimizer (Stochastic Gradient Descent), and it starts an instance of the BLSTM model with the hyperparameters set.

Detailed Model Definition:

1. BLSTM Class:
The BLSTM class is a subclass of nn.module, the base class for all neural network modules in PyTorch.

2. __init__ Method:
The __init__ method is the constructor of the class. It initializes the layers and parameters of the BLSTM model.
Parameters:
- **vocab_size:** Number of unique words in the vocabulary.
- **embedding_dim:** Dimensionality of word embeddings.
- **dropout:** Dropout rate for the LSTM layer.
- **tagToIndexD:** Dictionary mapping tags to indices.
- **hidden_dim:** Number of hidden units in the LSTM layer.

Layers Initialized:
- **word_embeddings:** Embedding layer to convert word indices into dense vectors.
- **lstm:** Bidirectional LSTM layer with hidden_dim // 2 units in each direction.
- **hidden1tag:** Linear layer for intermediate tag prediction.
- **hidden2tag:** Linear layer for final tag prediction.
- **dropout:** Dropout layer with dropout rate specified.

3. forward Method:

The forward method defines the forward pass of the BLSTM model.

Input:

- sentence: Tensor representing a sequence of word indices.

- Forward Pass:

- Embeds the input words using the word embeddings layer.
- Passes the embedded sequence through the bidirectional LSTM layer.
- Applies dropout regularization.
- Computes tag scores using linear layers and activation functions.

- Output:

- Returns the final tag scores for the input sequence.

4. Loss Function and Optimizer:

- loss_function:

CrossEntropyLoss is a commonly used loss function for classification tasks, including sequence tagging. It combines the LogSoftmax function and damaging log-likelihood loss.

- optimizer:

Stochastic Gradient Descent (SGD) is used as the optimization algorithm. It updates the model's parameters to minimize the loss with a specified learning rate (rate_of_l).

5. Training Hyperparameters:

- epochs:

Number of training epochs, i.e., the number of times the entire dataset is passed forward and backward through the model.

- rate_of_l:

Learning rate for the optimizer, determining the step size during parameter updates.

Testing the Model on the Dev Data:

1. Reading and Preprocessing Dev Data:

It reads data from a dev file, splits it into words, and organizes it into sentences and corresponding tags.

2. Converting Sentences to Tensors:

It converts the words in each sentence to their respective indices using the previously defined word-to-index dictionary (`w2i`).

3. Inference using the Model:

It performs inference using the trained BLSTM model on the dev data to predict tags for each word in the sentences.

4. Converting Predicted Tags to Labels:

It converts the predicted tag indices to their respective labels using the tag-to-index dictionary (`t2i`).

5. Writing Predictions to Output File:

It constructs lines of output containing the index, word, and actual NER tag, as well as a predicted NER tag for each word in the sentences, and writes them into an output file.

6. Output File Generation:

It generates an output file named "dev1.out" containing the predictions for each word in the dev data.

On evaluating the dev1.out file with the given eval.py script, here are the results:

```
accuracy:  94.88%; precision:  79.82%; recall:  69.64%; F1-Score:  74.38
```

Next, we evaluate the testing data similarly and generate the predictions file “test1.out”

Task 2:

Explanation of code:

Loading GloVe Word Embeddings:
The code begins by loading pre-trained GloVe word embeddings from a file (glove.6B.100d). It parses each file line to extract word vectors and stores them in a dictionary **g_E** where the key is the word, and the value is its corresponding embedding vector.

GloveBLSTM Model Definition:

1. GloveBLSTM Class:
This class defines the architecture of the new model for sequence tagging using GloVe word embeddings.

2. __init__ Method:
The constructor method is responsible for initializing the model's layers and parameters.
- **Parameters:**
  - **vocab_size:** Size of the vocabulary (number of unique words).
  - **tag_size:** Number of unique tags in the dataset.
  - **embedding_dim:** Dimensionality of word embeddings.
  - **hidden_dim:** Dimensionality of the LSTM hidden states.
  - **dropout:** Dropout rate for regularization.
  - **word_to_embedding:** Dictionary containing pre-trained word embeddings.
- **Layers Initialized:**
  - **embedding:** Embedding layer mapping word indices to GloVe word embeddings.
  - **lstm:** Bidirectional LSTM layer processing input embeddings.
  - **hidden2tag:** Linear layer mapping LSTM output to a smaller dimensionality.
  - **dropout:** Dropout layer for regularization.
  - **activation:** Activation function (ELU) applied to the output of `hidden2tag`.
  - **hidden3tag:** Final linear layer mapping to tag scores.

3. forward Method:
Defines the forward pass of the model.
- **Input:**
  - **sentence:** Tensor representing a sequence of word indices.
- **Forward Pass:**
  - Embeds the input words using the GloVe word embeddings.
  - Passes the embedded sequence through the bidirectional LSTM layer.
  - Applies dropout regularization.
  - Computes tag scores using linear layers and activation functions.
- **Output:**
  - Returns the final tag scores for the input sequence.

4. Training the Model:
The code then trains the GloveBLSTM model using the specified hyperparameters, loss function (CrossEntropyLoss), and optimizer (SGD).

5. Explanation of Hyperparameters:
- **vocab_size:** Size of the vocabulary (number of unique words).
- **tag_size:** Number of unique tags in the dataset.
- **embedding_dim:** Dimensionality of word embeddings (100 in this case, matching GloVe embeddings).
- **hidden_dim:** Dimensionality of the LSTM hidden states.
- **output_dimensions:** Dimensionality of the linear layer output.

- epochs: Number of training epochs.
- dropout: Dropout rate for regularization.
- optimizer: Adam optimizer with a learning rate of 0.1.

6. Training Loop:
The model is trained for the specified number of epochs, iterating over each sentence and its corresponding tags, computing the loss, performing backpropagation, and updating the model parameters using the optimizer.

7. Loss Calculation:
The average loss for each epoch is printed to monitor the training progress.

Dev predictions on the Task 2:

1. Processing Dev Data:
It organizes the dev data into sentences and their corresponding tags, where each sentence is represented as a list of words, and each tag is defined as a list of labels.

2. Inference using the Trained Model:
It performs inference using the trained g_model (presumably a GloveBLSTM model) on the dev data to predict tags for each word in the sentences.

3. Converting Predicted Tags to Labels:
It converts the predicted tag indices to their respective labels using the tag-to-index dictionary (`t2i`).

4. Writing Predictions to Output File:
It constructs lines of output containing the index, word, actual NER tag, and predicted NER tag for each word in the sentences and writes them to an output file named "dev2.out".

5. Output File Generation:
It generates an output file containing the predictions for each word in the dev data, with each line representing a word along with its actual NER tag and predicted NER tag.

On evaluating the dev2.out file with the given eval.py script, here are the results:

```
accuracy:  97.34%; precision:  88.29%; recall:  86.40%; FB1:  87.34
```

Next, we generate the predictions on the test data and create a **test2.out** file.