LSTM-Sentiment

April 20, 2023

1 Sentiment Classification with RNNs (LSTMs)

In this assignment you will experiment with training and evaluating sentiment classification models that use recurrent neural networks (RNNs) implemented in PyTorch, where an input document (movie review) is represented as a sequence of word embeddings.

1.1 Write Your Name Here:

2 Submission Instructions

2.1 Local computer:

1. Click the Save button at the top of the Jupyter Notebook.
2. Please make sure to have entered your name above.
3. Select Cell -> All Output -> Clear. This will clear all the outputs from all cells (but will keep the content of ll cells).
4. Select Cell -> Run All. This will run all the cells in order, and will take several minutes.
5. Once you've rerun everything, select File -> Download as -> PDF via LaTeX and download a PDF version *lstm-sentiment.pdf* showing the code and the output of all cells, and save it in the same folder that contains the notebook file *lstm-sentiment.ipynb*.
6. Look at the PDF file and make sure all your solutions are there, displayed correctly.
7. Submit **both** your PDF and notebook on Canvas. Make sure the PDF and notebook show the outputs of the training and evaluation procedures. Also upload the **output** on the test datasets.
8. Verify your Canvas submission contains the correct files by downloading them after posting them on Canvas.

2.2 Google Colab

While the code in this notebook can be run locally on a powerful machine, it is highly recommended that the notebook is run on the GPU infrastructure availabe for free through Google's Colab. To load the notebook in Colab:

1. Point your browser to https://colab.research.google.com/
2. If a pop-up window opens, click on the Upload button and select this notebook file `code/LSTM-Sentiment.ipynb`) from the homework folder.
3. Alternatively, in the notebook window menu clik File -> Open notebook and load the same notebook file.
4. You will also need to upload the `data` folder and the auxiliarry `*.py` files from the `code` folder. To do this:

- Using the menu, click ‘File’ / ‘Locate in Drive’. This will open a new browser window showing the contents of the Drive folder containing the notebook file.
- Using the manu pane on the left, clik on ‘+ New’, followed by ‘File upload’ (to upload the .py files) or ‘Folder upload’ (to upload the data folder).

2.3 Educational cluster:

1. Please make sure to have entered your name above.
2. Run the Python code on the cluster, using the instructions at: https://webpages.charlotte.edu/rbunescu/courses/itcs6156/centaurus.pdf
3. Look at the Slurm output file and make sure all your solutions are there, displayed correctly.
4. Edit the Analysis section in the notebook file, and save it as a PDF. Alternatively, you can use a text editor to edit yoru Analysis, then export it as PDF.
5. Submit the **Slurm output file**, the **Python source code** file lstm-sentiment.py, and the **analysis PDF** on Canvas. Also upload the **output** on the test datasets.
6. Verify your Canvas submission contains the correct files by downloading them after posting them on Canvas.

[ ]:
```python
from models import *
from sentiment_data import *

import random
import numpy as np
import torch
from typing import NamedTuple

class HyperParams(NamedTuple):
    lstm_size: int
    hidden_size: int
    lstm_layers: int
    drop_out: float
    num_epochs: int
    batch_size: int
    seq_max_len: int
```

3 LSTM-based training and evaluation procedures

We will use the RNNet class defined in `models.py` that uses LSTMs implemented in PyTorch. Depending on the options, this class runs one LSTM (forward) or two LSTMS (bidirectional, forward-backward) on the padded input text. The last state (or concatenated last states), or the average of the states, is used as input to a fully connected network with 3 hidden layers, with a final output sigmoid node computing the probability of the positive class.

[ ]:
```python
# Training procedure for LSTM-based models
def train_model(hp: HyperParams,
                train_exs: List[SentimentExample],
                dev_exs: List[SentimentExample],
                test_exs: List[SentimentExample],
```

```
                 word_vectors: WordEmbeddings,
                 use_average, bidirectional):
    train_size = len(train_exs)
    class_num = 1

    # Specify training on gpu: set to False to train on cpu
    # use_gpu = False
    use_gpu = torch.cuda.is_available()
    if use_gpu: # Set tensor type when using GPU
        float_type = torch.cuda.FloatTensor
    else: # Set tensor type when using CPU
        float_type = torch.FloatTensor

    # To get you started off, we'll pad the training input to 60 words to make␣
↪it a square matrix.
    train_mat = np.asarray([pad_to_length(np.array(ex.indexed_words), hp.
↪seq_max_len) for ex in train_exs])
    # Also store the actual sequence lengths.
    train_seq_lens = np.array([len(ex.indexed_words) for ex in train_exs])

    # Training input reversed, useful is using bidirectional LSTM.
    train_mat_rev = np.asarray([pad_to_length(np.array(ex.
↪get_indexed_words_reversed()), hp.seq_max_len) for ex in train_exs])

    # Extract labels.
    train_labels_arr = np.array([ex.label for ex in train_exs])
    targets = train_labels_arr

    # Extract embedding vectors.
    embed_size = word_vectors.get_embedding_length()
    embeddings_vec = np.array(word_vectors.vectors).astype(float)

    # Create RNN model.
    rnnModel = RNNet(hp.lstm_size, hp.hidden_size, hp.lstm_layers, hp.drop_out,
                     class_num, word_vectors,
                     use_average, bidirectional,
                     use_gpu =use_gpu)

    # If GPU is available, then run experiments on GPU
    if use_gpu:
        rnnModel.cuda()

    # Specify optimizer.
    optimizer = optim.Adam(filter(lambda p: p.requires_grad, rnnModel.
↪parameters()),
                           lr = 5e-3, weight_decay  =5e-3, betas = (0.9, 0.9))
```

```python
    # Define loss function: Binary Cross Entropy loss for logistic regression (binary classification).
    criterion = nn.BCELoss()


    # Get embeddings of words for forward and reverse sentence: (num_ex * seq_max_len * embedding_size)
    x = np.zeros((train_size, hp.seq_max_len, embed_size))
    x_rev = np.zeros((train_size, hp.seq_max_len, embed_size))
    for i in range(train_size):
        x[i] = embeddings_vec[train_mat[i].astype(int)]
        x_rev[i] = embeddings_vec[train_mat_rev[i].astype(int)]

    # Train the RNN model, gradient descent loop over minibatches.
    for epoch in range(hp.num_epochs):
        rnnModel.train()

        ex_idxs = [i for i in range(train_size)]
        random.shuffle(ex_idxs)

        total_loss = 0.0
        start = 0
        while start < train_size:
            end = min(start + hp.batch_size, train_size)

            # Get embeddings of words for forward and reverse sentence: (num_ex * seq_max_len * embedding_size)
            x_batch = form_input(x[ex_idxs[start:end]]).type(float_type)
            x_batch_rev = form_input(x_rev[ex_idxs[start:end]]).type(float_type)
            y_batch = form_input(targets[ex_idxs[start:end]]).type(float_type)
            seq_lens_batch = train_seq_lens[ex_idxs[start:end]]

            # Compute output probabilities over all examples in minibatch.
            probs = rnnModel(x_batch, x_batch_rev, seq_lens_batch).flatten()

            # Compute loss over all examples in minibatch.
            loss = criterion(probs, y_batch)
            total_loss += loss.data

            # Zero gradients, perform a backward pass, and update the weights.
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            start = end

        print("Loss on epoch %i: %f" % (epoch, total_loss))
```

```
        # Print accuracy on training and development data.
        if epoch % 10 == 0:
            acc = eval_model(rnnModel, train_exs, embeddings_vec, hp.
↪seq_max_len)
            print('Epoch', epoch, ': Accuracy on training set:', acc)
            acc = eval_model(rnnModel, dev_exs, embeddings_vec, hp.seq_max_len)
            print('Epoch', epoch, ': Accuracy on development set:', acc)

    # Evaluate model on the training dataset.
    acc = eval_model(rnnModel, train_exs, embeddings_vec, hp.seq_max_len)
    print('Accuracy on training set:', acc)

    # Evaluate model on the development dataset.
    acc = eval_model(rnnModel, dev_exs, embeddings_vec, hp.seq_max_len)
    print('Accuracy on develpment set:', acc)

    return rnnModel
```

Here is the testing (evaluation) procedure.

[ ]:
```
# Evaluate the trained model on test examples and return predicted labels or␣
↪accuracy.
def eval_model(model, exs, embeddings_vec, seq_max_len, pred_only = False):
    # Put model in evaluation mode.
    model.eval()

    # Extract size pf word embedding.
    embed_size = len(embeddings_vec[0])

    # Get embeddings of words for forward and reverse sentence: (num_ex *␣
↪seq_max_len * embedding_size)
    exs_mat = np.asarray([pad_to_length(np.array(ex.indexed_words),␣
↪seq_max_len) for ex in exs])
    exs_mat_rev = np.asarray([pad_to_length(np.array(ex.
↪get_indexed_words_reversed()), seq_max_len) for ex in exs])
    exs_seq_lens = np.array([len(ex.indexed_words) for ex in exs])

    # Get embeddings of words for forward and reverse sentence: (num_ex *␣
↪seq_max_len * embedding_size)
    x = np.zeros((len(exs), seq_max_len, embed_size))
    x_rev = np.zeros((len(exs), seq_max_len, embed_size))
    for i,ex in enumerate(exs):
        x[i] = embeddings_vec[exs_mat[i].astype(int)]
        x_rev[i] = embeddings_vec[exs_mat_rev[i].astype(int)]

    x = form_input(x)
```

```
    x_rev = form_input(x_rev)

    # Run the model on the test examples.
    preds = model(x, x_rev, exs_seq_lens).cpu().detach().numpy().flatten()
    preds[preds >= 0.5] = 1
    preds[preds < 0.5] = 0

    if pred_only == True:
        return preds
    else:
        targets = np.array([ex.label for ex in exs])
        return np.mean(preds == targets)
```

4 Experimental evaluations on the Rotten Tomatoes dataset.

First, code for reading the examples and the corresponding GloVe word embeddings.

[ ]:
```
random.seed(1)
np.random.seed(1)
torch.manual_seed(1)

word_vecs_path = '../data/glove.6B.300d-relativized.txt'

train_path = '../data/rt/train.txt'
dev_path = '../data/rt/dev.txt'
blind_test_path = '../data/rt/test-blind.txt'
test_output_path = 'test-blind.output.txt'

word_vectors = read_word_embeddings(word_vecs_path)
word_indexer = word_vectors.word_indexer

train_exs = read_and_index_sentiment_examples(train_path, word_indexer)
dev_exs = read_and_index_sentiment_examples(dev_path, word_indexer)
test_exs = read_and_index_sentiment_examples(blind_test_path, word_indexer)

print(repr(len(train_exs)) + " / " +
      repr(len(dev_exs)) + " / " +
      repr(len(test_exs)) + " train / dev / test examples")
```

4.1 Use only the last state from one LSTM

Evaluate One LSTM + fully connected network, use the last hidden state of LSTM. If the evaluation takes more than 1 hour on your computer, for debugging purposes you can try reducing `lstm_size`, `hidden_size`, `batch_size` and even `num_epochs`.

Our accuracy on development data is 75.98%

[ ]:
```
random.seed(1)
np.random.seed(1)
torch.manual_seed(1)

hp = HyperParams(lstm_size = 50, # hidden units in lstm
                 hidden_size = 50, # hidden size of fully-connected layer
                 lstm_layers = 1, # layers in lstm
                 drop_out = 0.5, # dropout rate
                 num_epochs = 50, # number of epochs for SGD-based procedure
                 batch_size = 1024, # examples in a minibatch
                 seq_max_len = 60) # maximum length of an example sequence
use_average = False
bidirectional = False

# Train RNN model.
model1 = train_model(hp, train_exs, dev_exs, test_exs, word_vectors,␣
 ↪use_average, bidirectional)

# Generate RNN model predictions for test set.
embeddings_vec = np.array(word_vectors.vectors).astype(float)
test_exs_predicted = eval_model(model1, test_exs, embeddings_vec, hp.
 ↪seq_max_len, pred_only = True)

# Write the test set output
for i, ex in enumerate(test_exs):
    ex.label = int(test_exs_predicted[i])
write_sentiment_examples(test_exs, test_output_path, word_indexer)

print("Prediction written to file for Rotten Tomatoes dataset.")
```

4.2 Use the average of all states from one LSTM (10 points)

Evaluate One LSTM + fully connected network, use average of all states of the LSTM.

Our accuracy on development data is 77.67%

[ ]:
```
random.seed(1)
np.random.seed(1)
torch.manual_seed(1)

## YOUR CODE HERE
```

4.3 Use a bidirectional LSTM, concatenate last states (10 points)

Evaluate Two LSTMs (bidirectional) + fully connected network, concatenate their last states.

Our accuracy on development data is 76.83%

```
random.seed(1)
np.random.seed(1)
torch.manual_seed(1)

## YOUR CODE HERE
```

4.4 Use a bidirectional LSTM, concatenate the averages of their states (10 points)

Evaluate Two LSTMs (bidirectional) + fully connected network, concatenate the averages of their states.

Our accuracy on development data is 77.39%

```
random.seed(1)
np.random.seed(1)
torch.manual_seed(1)

## YOUR CODE HERE
```

4.5 Average performance and standard deviation (20 points)

The NN performance can vary depending on the random initialization of its parameters. Train and evaluate each model 10 times, from different random initializations (10 different seeds). Average the accuracy over the 10 runs and compare the performance of the 4 models on the Rotten Tomatoes dataset. Report in your analysis the average and standard deviation for each model.

```
## YOUR CODE HERE
```

5 Experimental evaluations on the IMDB dataset (20 points)

Run the same 4 evaluations on the IMDB dataset.

```
train_path = '../data/imdb/train.txt'
dev_path = '../data/imdb/dev.txt'
test_path = '../data/imdb/test.txt'

test_output_path = 'test-imdb.output.txt'

## YOUR CODE HERE
```

5.1 Cross-domain evaluation (30 points)

Compare the performance of the Bidirectional LSTM with state averaging on the IMDB test set in two scenarios:

1. The model is trained on the IMDB training data.
2. The model is trained on the Rotten Tomatoes data.

[ ]:
```
## YOUR CODE HERE
```

5.2 Bonus points

Anything extra goes here.

5.3 Analysis (10 points)

Include an analysis of the results that you obtained in the experiments above. Also compare with the sentiment classification performance from previous assignments and explain the difference in accuracy. Show the results using table(s).

[ ]:
```

```