Artificial Intelligence Final Report Assignment 問題 2 (Problem 2)
レポート解答用紙（Report Answer Sheet）

Group Leader

学生証番号（Student ID): 19522238

名前(Name): Nguyễn Lê Thanh

Group Members

学生証番号（Student ID): 19522145

名前(Name): Đinh Thị Diễm Sương

学生証番号（Student ID): 19522310

名前(Name): Phạm Hoàng Thư

問題 2　(Problem 2)のレポート

## Program

★ Link Colab (Problem 2): Problem2_Labwork5.ipynb

★ Link GitHub: Artificial-Intelligence-IE229.M21.CNCL

## Execution Results


## Explanation

First, we are building the model RNN + LSTM, with hyperparameters random_seed = 123, vocabulary_size = 20000, learning_rate = 1e-4, batch_size = 128, num_epochs = 15, embedding_dim = 128, hidden_dim = 256, output_dim = 1, tokenize = 'spacy' and divide the IMDB dataset into train, test, valid (num_train = 20000, num_valid = 5000, num_test = 25000), using binary accuracy, optimzer = adam.

★ The first training, build a model that has an Embedding layer, an LSTM layer, and a Linear layer

| | |
|---|---|
| *Model* | |

```python
import torch.nn as nn

class RNN(nn.Module):
    def __init__(self, input_dim, embedding_dim, hidden_dim, output_dim):

        super().__init__()

        self.embedding = nn.Embedding(input_dim, embedding_dim)
        self.rnn = nn.LSTM(embedding_dim, hidden_dim)
        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, text, text_length):

        #[sentence len, batch size] => [sentence len, batch size, embedding size]
        embedded = self.embedding(text)

        packed = torch.nn.utils.rnn.pack_padded_sequence(embedded, text_length.to('cpu'))

        #[sentence len, batch size, embedding size] =>
        #  output: [sentence len, batch size, hidden size]
        #  hidden: [1, batch size, hidden size]
        packed_output, (hidden, cell) = self.rnn(packed)

        return self.fc(hidden.squeeze(0)).view(-1)
```

| | |
|---|---|
| *Output* | |

```
training accuracy: 90.47%
valid accuracy: 84.68%
Time elapsed: 2.78 min
Total Training Time: 2.78 min
Test accuracy: 84.50%
```

★ This is training of the RNN uses pre-trained word vectors (here: GloVe [5]), which are readily available in PyTorch via the build_vocab method of a tokenized data field.

| | |
|---|---|
| *GloVe Word Vectors* | |

```
[ ] TEXT.build_vocab(train_data, max_size=VOCABULARY_SIZE, vectors='glove.6B.100d', unk_init=torch.Tensor.normal_)
    LABEL.build_vocab(train_data)

    print(f'Vocabulary size: {len(TEXT.vocab)}')
    print(f'Number of classes: {len(LABEL.vocab)}')
```

| | |
|---|---|
| *Model* | |

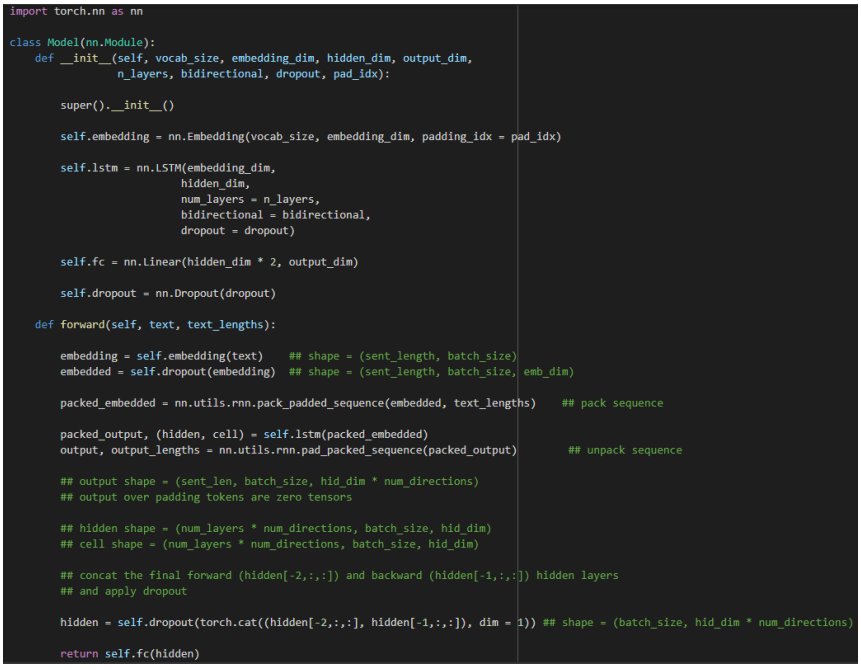
```python
import torch.nn as nn

class RNN(nn.Module):
    def __init__(self, input_dim, embedding_dim, hidden_dim, output_dim):

        super().__init__()

        self.embedding = nn.Embedding(input_dim, embedding_dim)
        self.rnn = nn.LSTM(embedding_dim, hidden_dim)
        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, text, text_length):

        #[sentence len, batch size] => [sentence len, batch size, embedding size]
        embedded = self.embedding(text)

        packed = torch.nn.utils.rnn.pack_padded_sequence(embedded, text_length.to('cpu'))

        #[sentence len, batch size, embedding size] =>
        #  output: [sentence len, batch size, hidden size]
        #  hidden: [1, batch size, hidden size]
        packed_output, (hidden, cell) = self.rnn(packed)

        return self.fc(hidden.squeeze(0)).view(-1)
```

| | |
|---|---|
| *Output* | training accuracy: 90.47%<br>valid accuracy: 84.68%<br>Time elapsed: 2.69 min<br>Total Training Time: 2.69 min<br>Test accuracy: 84.50% |

★ The third training, we use the Multilayer bidirectional RNN + LSTM (num_layers = 2, hidden_dim = 128)

| | |
|---|---|
| *Model* | `import torch.nn as nn`<br><br>`class RNN(nn.Module):`<br>`  def __init__(self, input_dim, embedding_dim, hidden_dim, output_dim):`<br><br>`  super().__init__()`<br><br>`  self.embedding = nn.Embedding(input_dim, embedding_dim)`<br>`  self.rnn = nn.LSTM(embedding_dim,`<br>`  hidden_dim,`<br>`  num_layers=NUM_LAYERS,`<br>`  bidirectional=BIDIRECTIONAL)`<br>`  self.fc = nn.Linear(hidden_dim*2, output_dim)`<br><br>`  def forward(self, text, text_length):`<br><br>`  #[sentence len, batch size] => [sentence len, batch size, embedding size]`<br>`  embedded = self.embedding(text)`<br><br>`  packed = torch.nn.utils.rnn.pack_padded_sequence(embedded, text_length.to('cpu'))`<br><br>`  packed_output, (hidden, cell) = self.rnn(packed)`<br><br>`  # combine both directions`<br>`  combined = torch.cat((hidden[-2,:,:], hidden[-1,:,:]), dim=1)`<br><br>`  return self.fc(combined.squeeze(0)).view(-1)` |
| *Output* | training accuracy: 94.01%<br>valid accuracy: 86.66%<br>Time elapsed: 4.52 min<br>Total Training Time: 4.52 min<br>Test accuracy: 85.96% |

★ The last training, we use RNN + LSTM + Glob + dropout (num_layers = 2, hidden_dim = 128, dropout = 0.4), replace initial embedding with pretrained embedding, replace and with zeros (they were initialized with the normal distribution).

| | |
|---|---|
| *Replace initial embedding with pretrained embedding* | `pretrained_embeddings = TEXT.vocab.vectors`<br>`model.embedding.weight.data.copy_(pretrained_embeddings)` |
| *Replace and with zeros (they were initialized with the normal distribution)* | `UNK_IDX = TEXT.vocab.stoi[TEXT.unk_token]`<br><br>`model.embedding.weight.data[UNK_IDX] = torch.zeros(EMBEDDING_DIM)`<br>`model.embedding.weight.data[PAD_IDX] = torch.zeros(EMBEDDING_DIM)`<br><br>`print(model.embedding.weight.data)` |
| *Model* |  |
| *Output* | `Test Loss: 0.306 \| Test Acc: 88.74%` |

```
import torch.nn as nn

class Model(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim, output_dim,
                 n_layers, bidirectional, dropout, pad_idx):

        super().__init__()

        self.embedding = nn.Embedding(vocab_size, embedding_dim, padding_idx = pad_idx)

        self.lstm = nn.LSTM(embedding_dim,
                            hidden_dim,
                            num_layers = n_layers,
                            bidirectional = bidirectional,
                            dropout = dropout)

        self.fc = nn.Linear(hidden_dim * 2, output_dim)

        self.dropout = nn.Dropout(dropout)

    def forward(self, text, text_lengths):

        embedding = self.embedding(text)    ## shape = (sent_length, batch_size)
        embedded = self.dropout(embedding)  ## shape = (sent_length, batch_size, emb_dim)

        packed_embedded = nn.utils.rnn.pack_padded_sequence(embedded, text_lengths)    ## pack sequence

        packed_output, (hidden, cell) = self.lstm(packed_embedded)
        output, output_lengths = nn.utils.rnn.pad_packed_sequence(packed_output)       ## unpack sequence

        ## output shape = (sent_len, batch_size, hid_dim * num_directions)
        ## output over padding tokens are zero tensors

        ## hidden shape = (num_layers * num_directions, batch_size, hid_dim)
        ## cell shape = (num_layers * num_directions, batch_size, hid_dim)

        ## concat the final forward (hidden[-2,:,:]) and backward (hidden[-1,:,:]) hidden layers
        ## and apply dropout

        hidden = self.dropout(torch.cat((hidden[-2,:,:], hidden[-1,:,:]), dim = 1)) ## shape = (batch_size, hid_dim * num_directions)

        return self.fc(hidden)
```

Overview result table

<table>
<tr><td>RNN + LSTM</td><td>RNN + LSTM + GloVe</td><td>Multilayer bidirectional RNN + LSTM</td><td>RNN + LSTM + Glob + dropout</td></tr>
<tr><td></td><td></td><td></td><td>Replace initial embedding with pretrained embedding</td></tr>
<tr><td></td><td></td><td></td><td>Replace and with zeros</td></tr>
<tr><td></td><td></td><td></td><td>dropout = 0.4</td></tr>
<tr><td></td><td>vectors='glove.6B.100d'</td><td colspan="2">num_layers = 2,</td></tr>
<tr><td colspan="2">hidden_dim = 256</td><td colspan="2">hidden_dim = 128</td></tr>
<tr><td colspan="4">random_seed = 123</td></tr>
<tr><td colspan="4">vocabulary_size = 20000</td></tr>
<tr><td colspan="4">learning_rate = 1e-4</td></tr>
<tr><td colspan="4">batch_size = 128</td></tr>
<tr><td colspan="4">num_epochs = 15</td></tr>
<tr><td colspan="4">embedding_dim = 128</td></tr>
<tr><td colspan="4">output_dim = 1</td></tr>
</table>

## Conclusion

- ★ We have built RNN with LSTM. From basic networking to using pre-train word vectors (GloVe Word Vectors), and Multilayer bidirectional RNN + LSTM to the final model adding a dropout layer and data processing replace initial embedding with pretrained embedding and replace and with zeros.
- ★ Improved test accuracy from 68.35% to 88.74%.

## Future work

- ★ Data processing and adjustment of hyperparameters.
- ★ Can use models such as: CNN + LSTM, Alexnet model,…

## References


1. Sequence models and long short-term memory networks (Last visited: 07/07/2022)
2. Deeplearning-models (Last visited: 07/07/2022)
3. IMDb_reviews_classification (Last visited: 07/07/2022)
4. Sentiment Analysis on IMDb (Last visited: 07/07/2022)
5. Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP) (pp. 1532-1543).