COMP 6721 Applied Artificial Intelligence (Summer 2023)

Assignment #12: Deep Learning for NLP

Solutions

Question 1 Consider the following sentence:

"the cat drinks the milk"

We will use this sentence to train a CBOW Word2Vec model. Assume that:

- you want to produce word embeddings of dimension 2,
- you use a context window of size 2 (1 word before and 1 word after the target word), and
- your vocabulary only contains the words in the sentence above

(a) Using only the sentence above, how many instances will be generated as training set?

3 instances

Instance	Context Word -1	Context Word +1	To Predict
1	the	drinks	cat
2	cat	the	drinks
3	drinks	milk	the

(b) List the one-hot vectors that correspond to each word in the vocabulary. (Assume alphabetical ordering, no stop-word filtering)

Word	Hot Vector			
cat	1	0	0	0
drinks	0	1	0	0
milk	0	0	1	0
the	0	0	0	1

(c) List the one-hot vectors that correspond to each training instance in the input layer.

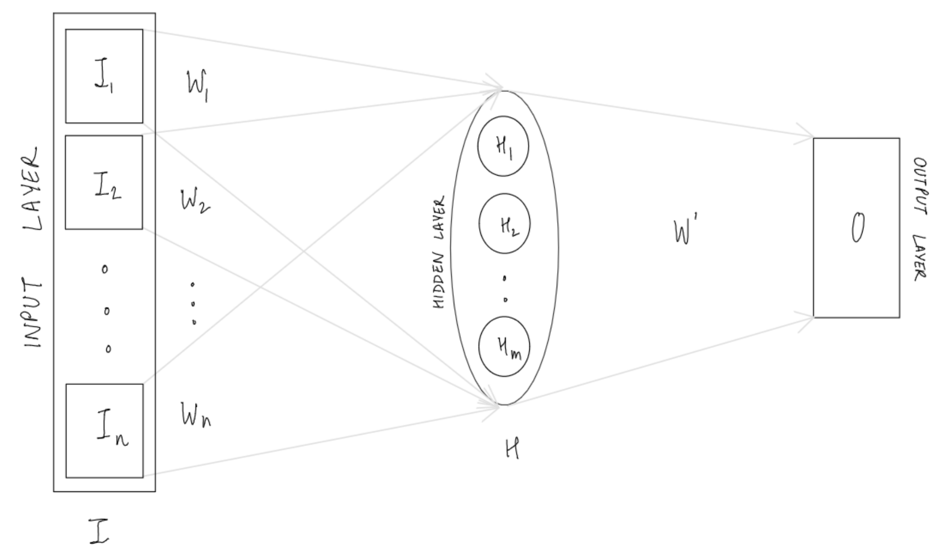
Instance	Context	Word	Hot Vector			
1	Context Word -1	the	0	0	0	1
	Context Word +1	drinks	0	1	0	0
2	Context Word -1	cat	1	0	0	0
	Context Word +1	the	0	0	0	1
3	Context Word -1	drinks	0	1	0	0
	Context Word +1	milk	0	0	1	0

(d) How many nodes will the hidden layer contain?
Number of nodes in hidden layer = 2

(e) What is the target hot vector for each training instance?

Instance	To Predict	Hot Vector			
1	cat	1	0	0	0
2	drinks	0	1	0	0
3	the	0	0	0	1

(f) Assume that the Word2Vec model is trained with the standard network depicted below:



i. What will be the values of n and m?
$n = 2$, $m = 2$

ii. What will be the sizes of I, W_i (for each $1 \leq i \leq n$), W' and O?
Size of $I_1 = I_2 = 1 \times 4$, $I = 2 \times 4$
Size of $W_1 = W_2 = 4 \times 2$
Size of $W' = 2 \times 4$

Size of $O = 1 \times 4$

(g) Assume that we have these weight vectors:

$$W = \begin{bmatrix} 2 & 6 \\ 4 & 3 \\ 1 & 4 \\ 5 & 2 \end{bmatrix}$$

$$W' = \begin{bmatrix} 6 & 2 & 8 & 3 \\ 4 & 5 & 9 & 7 \end{bmatrix}$$

To compute the final probabilities at the output layer, we use the softmax function as shown in class. Recall that for a given vector of size k, the softmax function is defined as:

$$p_i = \frac{e^{x_i}}{\sum_{i=1}^{k} e^{x_i}}, \text{ where } 1 \leq i \leq k$$

i. Trace the first feed forward pass in the network and show the values propagated all the way to the output layer.

Instance	Context	Word	Hot Vector				To Predict	Target			
1	Context Word -1	the	0	0	0	1	cat	1	0	0	0
	Context Word +1	drinks	0	1	0	0					

Calculate the output of each hidden node for each context word

$$H = I \times W = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 2 & 6 \\ 4 & 3 \\ 1 & 4 \\ 5 & 2 \end{bmatrix} = \begin{bmatrix} 5 & 2 \\ 4 & 3 \end{bmatrix}$$

Take the average

$$H_{AVG} = \begin{bmatrix} 4.5 & 2.5 \end{bmatrix}$$

Calculate output

$$O = H_{AVG} \times W' = \begin{bmatrix} 4.5 & 2.5 \end{bmatrix} \times \begin{bmatrix} 6 & 2 & 8 & 3 \\ 4 & 5 & 9 & 7 \end{bmatrix} = \begin{bmatrix} 37 & 21.5 & 58.5 & 31 \end{bmatrix}$$

Calculate softmax probabilities for the output

$$\text{softmax}(O) = \text{softmax}(\begin{bmatrix} 37 & 21.5 & 58.5 & 31 \end{bmatrix})$$

$$= \begin{bmatrix} 4.6 \times 10^{-10} & 8.53 \times 10^{-17} & 0.99 & 1.14 \times 10^{-12} \end{bmatrix}$$

ii. What is the error after the first pass?

Calculate error

$$E = O - T = \begin{bmatrix} 4.6 \times 10^{-10} & 8.53 \times 10^{-17} & 0.99 & 1.14 \times 10^{-12} \end{bmatrix} - \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$$
$$= \begin{bmatrix} \approx -1 & 8.53 \times 10^{-17} & 0.99 & 1.14 \times 10^{-12} \end{bmatrix}$$

Question 2 In this question, you will implement a network model to compute *word embeddings*. You will see how to train the network over the provided data, compute the loss function on the training example, and update the parameters with backpropagation. Finally, you can see how the loss decreases by iterating over the training data.

Use the following Python imports:

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
```

Assume that you want to produce word embeddings of dimension 2 and use a context window of size 2 (two words before the target word).

```python
CONTEXT_SIZE = 2
EMBEDDING_DIM = 10
```

Your vocabulary only contains the words in Shakespeare's Sonnet 2. Index each word in the vocabulary.

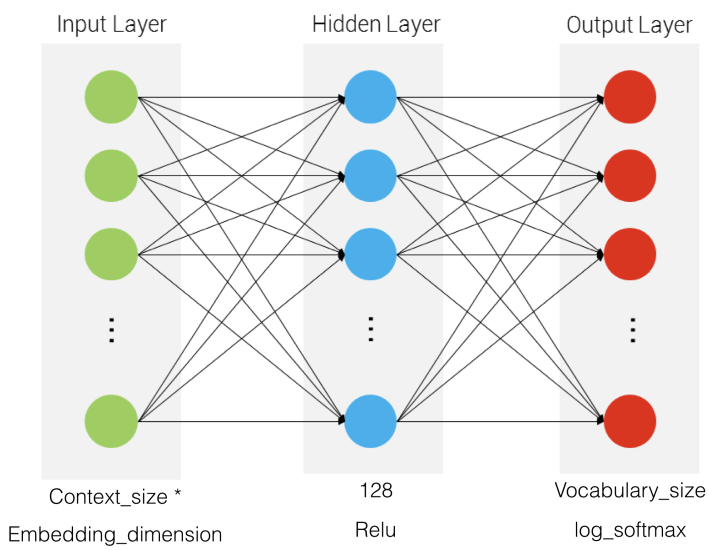
```python
test_sentence = """When forty winters shall besiege thy brow,
And dig deep trenches in thy beauty's field,
Thy youth's proud livery so gazed on now,
Will be a totter'd weed of small worth held:
Then being asked, where all thy beauty lies,
Where all the treasure of thy lusty days;
To say, within thine own deep sunken eyes,
Were an all-eating shame, and thriftless praise.
How much more praise deserv'd thy beauty's use,
If thou couldst answer 'This fair child of mine
Shall sum my count, and make my old excuse,'
Proving his beauty by succession thine!
This were to be new made when thou art old,
And see thy blood warm when thou feel'st it cold.""".split()

vocab = set(test_sentence)
word_to_ix = {word: i for i, word in enumerate(vocab)}
```

Build a list of tuples. Each tuple is ($[\text{word}_{i-2}, \text{word}_{i-1}]$, target word). Print the first 3 elements of the tuples list, so you can see what they look like.

```python
trigrams = [([test_sentence[i], test_sentence[i + 1]], test_sentence[i + 2])
    for i in range(len(test_sentence) - 2)]

print(trigrams[:3])
```



(a) Now we will define an `NGramLanguageModeler`. The variable `embeddings` is a simple lookup table that stores embeddings of a fixed dictionary and size. Consider the network shown below to complete the class definition. Use `nn.Linear`[1] to apply linear transformation and `nn.LogSoftmax`[2] to apply $\log(\text{softmax}(x))$.

```
class NGramLanguageModeler(nn.Module):

  def __init__(self, vocab_size, embedding_dim, context_size):
    super(NGramLanguageModeler, self).__init__()
    self.embeddings = nn.Embedding(vocab_size, embedding_dim)
    # *** Your Code Here (2 lines) ***


  def forward(self, inputs):
    embeds = self.embeddings(inputs).view((1, -1))
    # *** Your Code Here (4 lines) ***
```

The completed code should look like this:

```
class NGramLanguageModeler(nn.Module):

  def __init__(self, vocab_size, embedding_dim, context_size):
    super(NGramLanguageModeler, self).__init__()
    self.embeddings = nn.Embedding(vocab_size, embedding_dim)
    self.linear1 = nn.Linear(context_size * embedding_dim, 128)
    self.linear2 = nn.Linear(128, vocab_size)

  def forward(self, inputs):
```

[1] https://pytorch.org/docs/stable/generated/torch.nn.Linear.html
[2] https://pytorch.org/docs/stable/generated/torch.nn.LogSoftmax.html

```python
        embeds = self.embeddings(inputs).view((1, -1))
        out = F.relu(self.linear1(embeds))
        out = self.linear2(out)
        log_probs = F.log_softmax(out, dim=1)
        return log_probs
```

(b) Define the loss function, create an instance of the class you defined earlier, then use an SGD optimizer like below:

```python
losses = []
loss_function = nn.NLLLoss()
model = NGramLanguageModeler(len(vocab), EMBEDDING_DIM, CONTEXT_SIZE)
optimizer = optim.SGD(model.parameters(), lr=0.001)
```

Now it's time to train the model. Iterate through the `trigrams` for `max_epoch` steps (you can start with 1000).

- Use `nn.tensor`[3] to prepare the inputs (wrap them in tensors) to be passed to the model.
- Recall that Torch accumulates gradients. Before passing in a new instance, you need to zero out the gradients from the old instance.
- Run the forward pass, getting log probabilities over the next words.
- Compute your loss function. Again, Torch wants the target word wrapped in a tensor.
- Do the backward pass and update the gradient.

```python
for epoch in range(1000):
   total_loss = 0
   for context, target in trigrams:

      context_idxs = torch.tensor([word_to_ix[w] for w in context],
                                  dtype=torch.long)

      model.zero_grad()

      log_probs = model(context_idxs)

      loss = loss_function(log_probs, torch.tensor([word_to_ix[target]],
                                  dtype=torch.long))

      loss.backward()
      optimizer.step()

      total_loss += loss.item()
   losses.append(total_loss)
```

[3] https://pytorch.org/docs/stable/tensors.html

(c) Plot the loss to see how it decreased in every iteration.

```
import matplotlib.pyplot as plt

plt.plot(losses)
plt.show()
```

Question 3 OpenNMT[4] is an open-source machine translation framework. In this exercise, we are going to set up and train the PyTorch version of OpenNMT for a simple English to French translation dataset and understand how it works.

- Setup

  To set up the project, first, we need to activate the Conda environment and install the OpenNMT-py using pip:

  ```
  pip install OpenNMT-py
  ```

- Data

  Now we need to prepare the data. The data consists of parallel source (src) and target (tgt) data containing one sentence per line with tokens separated by a space. To get started, first create a dataset folder named `mini_en_fr`. Inside the dataset folder put the data files and their contents as shown below:

  `src-train.txt`
  the green box
  the red circle
  the cat likes box
  the boy eats orange
  the girl eats apple
  the boy plays tennis
  the woman likes tennis
  the girl likes apple
  the green apple
  the red apple
  the boy likes cat
  the girl likes paris

  `tgt-train.txt`
  la boîte verte
  le cercle rouge
  le chat aime la boîte
  le garçon mange de l'orange
  la fille mange la pomme
  le garçon joue au tennis
  la femme aime le tennis
  la fille aime la pomme
  la pomme verte
  la pomme rouge
  le garçon aime le chat
  la fille aime paris

[4] https://github.com/OpenNMT/OpenNMT-py

src-val.txt
the green circle
the woman likes cat
the red cat

tgt-val.txt
le cercle vert
la femme aime le chat
le chat rouge

src-test.txt
the woman likes paris
the red box
the cat plays in box

tgt-val.txt
la femme aime paris
la boîte bleue
le chat joue en boite

- Configuration

  Now we need to build a YAML configuration file to specify the data that will be used:

  mini_en_fr.yaml

```yaml
## Where the samples will be written
save_data: mini_fr_en/run/example

## Where the vocab(s) will be written
src_vocab: mini_fr_en/run/example.vocab.src
tgt_vocab: mini_fr_en/run/example.vocab.tgt

# Prevent overwriting existing files in the folder
overwrite: False

# Corpus opts:
data:
   corpus_1:
      path_src: mini_fr_en/src-train.txt
      path_tgt: mini_fr_en/tgt-train.txt
   valid:
      path_src: mini_fr_en/src-val.txt
      path_tgt: mini_fr_en/tgt-val.txt
```

```
# Vocabulary files that were just created
src_vocab: mini_fr_en/run/example.vocab.src
tgt_vocab: mini_fr_en/run/example.vocab.tgt

# Train on CPU
world_size: 1
#gpu_ranks: [-1]

# Where to save the checkpoints
save_model: mini_fr_en/run/model
save_checkpoint_steps: 100
train_steps: 200
valid_steps: 100
```

From this configuration, we can build the vocab(s) that will be necessary to train the model:

```
onmt_build_vocab -config mini_en_fr.yaml -n_sample 100
```

- Train

  To train the model you can simply run:

  ```
  onmt_train -config toy_en_de.yaml
  ```

  This configuration will run the default model, which consists of a 2-layer LSTM with 500 hidden units on both the encoder and decoder.

- Translate

  Now you have a model which you can use to predict on new data. This will output predictions into `mini_en_fr/pred_100.txt`. Run the command below and check the translation result.

  ```
  onmt_translate -model mini_en_fr/run/model_step_200.pt -src
     mini_en_fr/src-test.txt -output mini_en_fr/pred_1000.txt
     -gpu -1 -verbose
  ```

  The predictions might be quite terrible, as the demo dataset is very small. Try running it on some larger datasets!

Question 4 For a real-world translation system, we will use the pre-trained OpenNMT models available from *Argus Translate*.[5] A nice user interface for these models, similar to Google Translate or DeepL, is *LibreTranslate*,[6] an open source self-hosted machine translation library. You can run your own API server in just a few lines of setup!

First, we need to make sure you have Python 3.8 or higher installed, so lets create a specific Conda environment for `libretranslate`:

```
conda --name libretranslate python=3.8
conda activate libretranslate
```

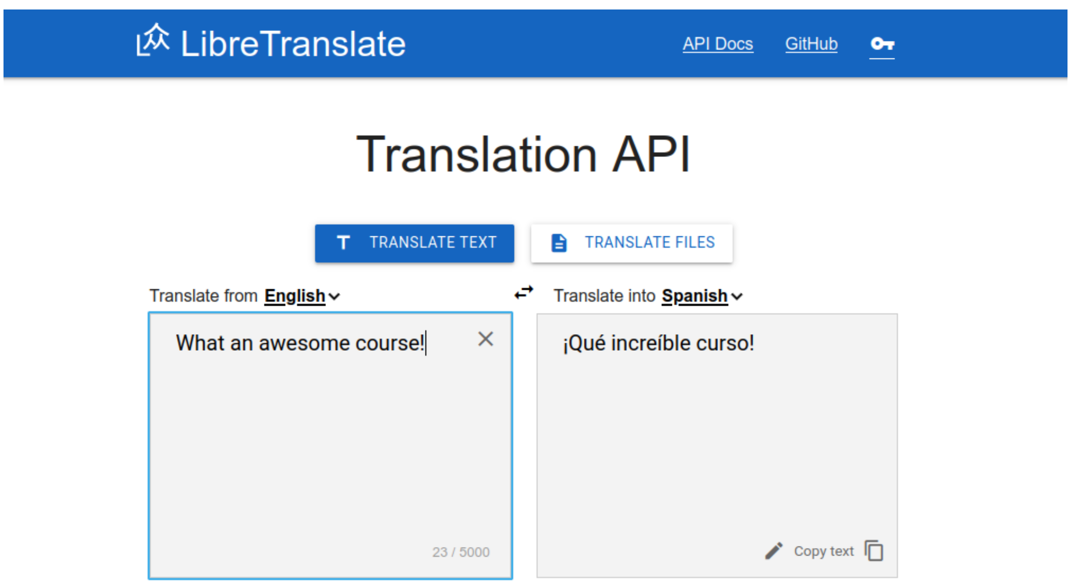
Then we need to install `libretranslate` using `pip`:

```
pip install libretranslate
```

Now you can run the command `libretranslate` on the command line. It loads a number of different language models and creates a web interface:

```
libretranslate
```

You can now access your own translation server at `http://localhost:5000`.



[5] See `https://www.argosopentech.com/`
[6] See `https://github.com/uav4geo/LibreTranslate`