CE7455: Deep Learning for Natural Language Processing

Assignment1

Name	Matric Number	email
ZHANG ZIHAO	G1903381A	ZZHANG049@e.ntu.edu.sg

Question 1

We get the loss function as :

$loss = -\sum_{k=1}^{K} y_k * log(p_k), which\ p_k = y_k^{'} = \frac{exp(w_k^T z)}{\sum_{k^{'}=1}^{K} exp(w_{k^{'}}^T z)}$

base on the chain rule, we have :

$$\frac{\partial loss}{\partial w_k} = \frac{\partial loss}{\partial (w_k^T z)} * \frac{\partial (w_k^T z)}{\partial w_k} = z * \frac{\partial loss}{\partial (w_k^T z)}$$

We assume $c_k = w_k^T z$, so we have :

$$\frac{\partial loss}{\partial w_k} = z * \frac{\partial loss}{\partial c_k}, while\ loss = -\sum_{k=1}^{K} y_k * log(p_k), p_k = \frac{exp(c_k)}{\sum_{k^{'}=1}^{K} exp(c_k^{'})}$$

$$\frac{\partial loss}{\partial c_k} = \frac{\partial loss}{\partial p_j} * \frac{\partial p_j}{\partial c_k},\ p_j\ could\ be\ each\ p$$

We use ∂p_j not ∂p_k , this is due to the denominator of softmax. The denominator includes all neurons' output, so for p_j while j!=k, it also includes c_k, so we should consider each p_j .

for $\frac{\partial loss}{\partial p_j}$, it's easy to get $\frac{\partial loss}{\partial p_j} = -\sum_j y_j * \frac{1}{p_j}$.

but for $\frac{\partial p_j}{\partial c_k}$, the situation becomes complicated.

We discuss 2 situations, while k=j :

$$While\ k = j,\ \frac{\partial p_j}{\partial c_k} = \frac{\partial p_k}{\partial c_k} = \frac{\partial \frac{exp(c_k)}{\sum_{k^{'}=1}^{K} exp(c_k^{'})}}{\partial c_k}$$

$$= \frac{exp(c_k) * \sum_{k^{'}=1}^{K} exp(c_k^{'}) - exp(c_k) * exp(c_k)}{(\sum_{k^{'}=1}^{K} exp(c_k^{'}))^2}$$

$$= \frac{exp(c_k)}{\sum_{k^{'}=1}^{K} exp(c_k^{'})} - (\frac{exp(c_k)}{\sum_{k^{'}=1}^{K} exp(c_k^{'})})^2$$

$$= p_k - p_k^2$$

and while k!=j :

$$While\ k!=j,\ \frac{\partial p_j}{\partial c_k}=\frac{\partial \frac{exp(c_j)}{\sum_{k'=1}^{K}exp(c_k^{'})}}{\partial c_k}$$
$$=-exp(c_j)*\frac{exp(c_k)}{(\sum_{k'=1}^{K}exp(c_k^{'}))^2}=-p_j*p_k$$

and then, we combine these 2 situations :

$$\frac{\partial loss}{\partial c_k}=-\sum_j y_j*\frac{1}{p_j}*\frac{\partial p_j}{\partial c_k}=-(\frac{y_k}{p_k}*(p_k-p_k^2)+\sum_{j\ but\ j!=k}\frac{y_j}{p_j}*(-p_j*p_k))$$
$$=-y_k+y_kp_k+\sum_{j\ but\ j!=k}y_jp_k=-y_k+p_k\sum_j y_j$$

only one of y_j is 1, and the rest are 0, so we can get $\sum_j y_j=1$

In summary :

$$\frac{\partial loss}{\partial w_k}=z*\frac{\partial loss}{\partial c_k}=z*(p_k-y_k)=z*(y_k^{'}-y_k)$$

Question 2

- **The implement of FNN model:**

  if we want to write a FNN model base on `torch.nn.Module` , we just need to rewrite `__init__` function and `forward` . FNN model consists of linear transformation, so `nn.Linear` is enough to finish our network. Code as follows:

```
class FNNModel(nn.Module):
    def __init__(self,ntoken, ninp, nhid, nlayers, dropout):
        super(FNNModel, self).__init__()

        self.drop = nn.Dropout(dropout)
        self.encoder = nn.Embedding(ntoken, ninp)
        self.decoder = nn.Linear(nhid, ntoken)
        self.hidden = nn.Linear(ninp,nhid);

        self.nhid = nhid
        self.nlayers = nlayers

    def forward(self, input):
        L1_output=self.encoder(input) # ntoken*ninp
        hidden_input=self.hidden(L1_output) #ntoken*nhid
        hidden_output=torch.tanh(hidden_input)
        final_input=self.decoder(hidden_output)
        return final_input
```

  The output layer of FNN model is a Softmax layer but we can't add a Softmax transformation in forward function. This is due to CrossEntropyLoss, which is used in the sample code. CrossEntropyLoss has transformed the output by Softmax, it combines `nn.LogSoftmax()` and `nn.NLLLoss()` in one single class, so we don't need to do that again. If we want to add Softmax in the forward function, we could use NLLLoss when we are training our model.

  I used Adam algorithm to train FNN model. It's very easy to implement Adam algorithm by pytorch, code as follows:

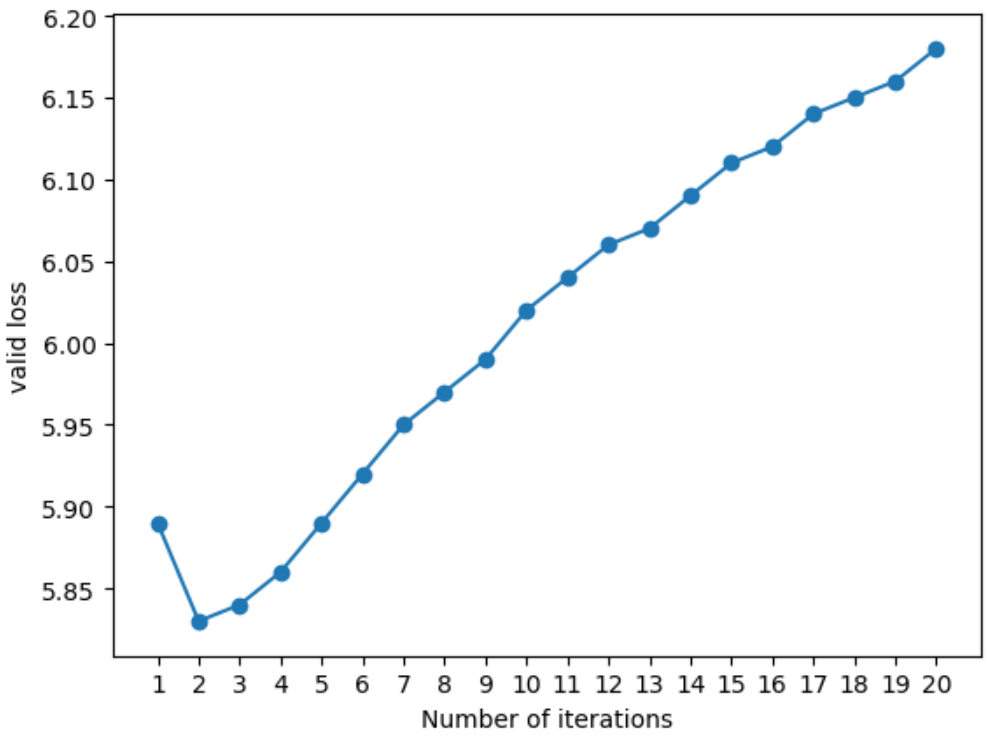
```
lr = 0.001
optimizer = torch.optim.Adam(model.parameters(),lr)

#train process
model.zero_grad()
output = model(data)
loss = criterion(output.view(-1, ntokens), targets)
loss.backward()
optimizer.step()
```

We make 20 iterations. After training, we evaluate our model on validation set and test set. The result as follows:

```
-----------------------------------------------------------------------------
| end of epoch  20 | time: 54.09s | valid loss  6.18 | valid ppl
481.02
-----------------------------------------------------------------------------
=============================================================================
| End of training | test loss  5.74 | test ppl   310.21
=============================================================================
```

The following figure shows valid loss's changing trend:



Using Adam algorithm, we get the best model on the validation set after the second iteration. Compared to BGD or SGD, these two methods can't get the best model after the second iteration, it usually happens after 10 iterations. This proves Adam algorithm's effectiveness, but after 20 trainings, my model is overfitting, so I add dropout in FNN model, but the situation has not improved.

- **Do steps (iv)-(v) again, but now with sharing the input (look-up matrix) and output layer**
  embeddings

We share the weight when we initialize the model:

```
if flag_share:
    self.encoder.weight=self.decoder.weight
```

It reduces the number of parameters of the neural network, so the model could run faster. Besides that, the model performs better in validation set and test set, but not obvious.

```
-----------------------------------------------------------------------------
| end of epoch  20 | time: 47.25s | valid loss  6.05 | valid ppl
425.42
-----------------------------------------------------------------------------
=============================================================================
| End of training | test loss  5.64 | test ppl   280.90
=============================================================================
```

- **Generate texts by using my language model**

  After training our model, we run generate.py and then we can get generated texts. I attach the generated.txt in `model.rar` .

- **Which computation/operation is the most expensive one in inference or forward pass?**

  The forward pass consists of follow parts:

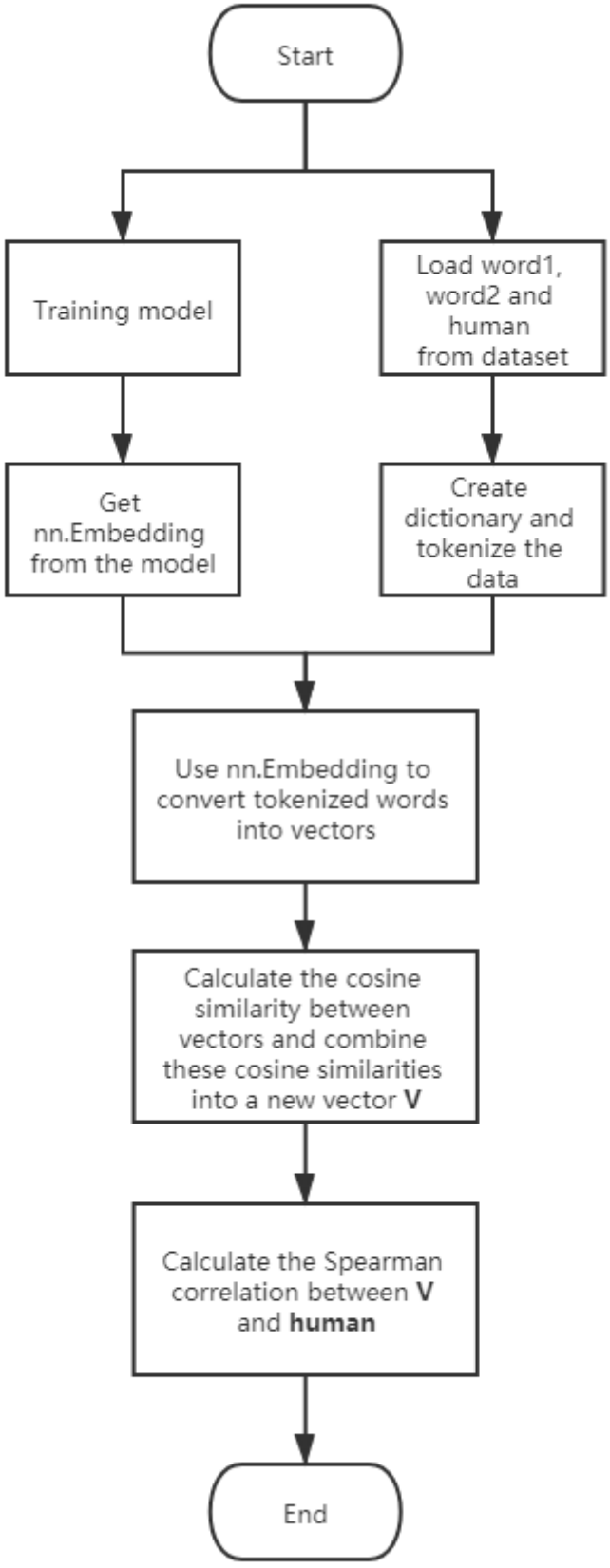
  1. do matrix multiplication with output from last layer **x** and the weight in current layer **w** , **w*x**
  2. add bias **b** , **w*x+b** ·
  3. activate and output it , **f(w*x+b)**

  The most expensive computation in forward pass is matrix multiplication——**w*x** in neurons.

  A feasible method to improve this is reduce the size of the network (layers and neurons) . But the size affects the performance of the neural network, so we need to find a appropriate size of the neural network between performance and cost time.

- **Report the Spearman correlation base on another dataset**

  Actually, I spend a lot of time at this part. First, I give a flowchart of the solution to this problem as follows. If there is anything wrong about this flowchart, I really hope you could tell me. Thank you in advance.



Well...I think this task is not hard, so I write follow codes:

```
import spearman as sp

def get_spearman():
    data, human = sp.Load_data('./data/wikitext-2/combined.csv')
    corpus = sp.Corpus(data,human)
    encoder = model.get_embedding()
    word_data = batchify(corpus.train, 1)
    vector = encoder(word_data) #nwords*100, included repeated word
    cosine = []
    for i in range(0,vector.size(0)-1,2):
        cosine.append(torch.cosine_similarity(vector[i][0],vector[i+1][0],dim=-1))
    spearman=stats.spearmanr(cosine, human)
    print(spearman)
```

And then I get a very confused result.:

`SpearmanrResult(correlation=0.009046541332703507, pvalue=0.8655058631407702)`

By looking at the dataset, I think the similarity between word1 and word2 should be related to human value. So I tried some ways to correct this error. I turned **cosine** into float data from tensor and calculated the Spearman correlation again, the situation has changed, but no much use.

`SpearmanrResult(correlation=0.024885354666622915, pvalue=0.6412389859490015)`

Actually I don't think there should be a change here, but I don't know the details about `scipy.stats.spearmanr` . Maybe there is something different when the input is a tensor.

When I printed out the calculated cosine similarities, I found that they did not measure the similarity between word1 and word2 very well. For example, the cosine similarity between computer and keyboard is just -0.0224, but the cosine similarity between love and sex is 0.2100. I thinks there is something wrong in my `nn.Embedding` which is used to convert tokenized words to vectors. I changed the data set and used combined.csv to train FNN model, but this had no change to the result. I found that almost all words in combined.csv are included in wikitext-2, so it means this attempt is useless.

After these attempts, I have no ideas to this problem, I attached the code with this report, maybe professor you could help me if you are free. Thanks again in advance.

Question 3

I spent 2 hours on question 1, this is because I neglected the denominator of softmax contains all outputs, so I got a wrong answer as follows:

Question 1:

We get the loss function as :

$loss = -\sum_{k=1}^{K} y_k * log(p_k), which\ p_k = y_k^{'} = \frac{exp(w_k^T z)}{\sum_{k'=1}^{K} exp(w_{k'}^T z)}$

for each sample, we have

公式 </>

$$\frac{\partial loss}{\partial w_k} = \frac{\partial\ -y_k * log\frac{exp(w_k^T z)}{\sum_{k'=1}^{K} exp(w_{k'}^T z)}}{\partial w_k}$$

$$= -y_k * \frac{\sum_{k'=1}^{K} exp(w_{k'}^T z)}{exp(w_k^T z)} * \frac{z * exp(w_k^T z) * \sum_{k'=1}^{K} exp(w_{k'}^T z) - exp(w_k^T z) * z * exp(w_k^T z)}{(\sum_{k'=1}^{K} exp(w_{k'}^T z))^2}$$

$$= -y_k * \frac{z * \sum_{k'=1}^{K} exp(w_{k'}^T z) - z * exp(w_k^T z)}{\sum_{k'=1}^{K} exp(w_{k'}^T z)}$$

$$= -z * y_k * (1 - \frac{exp(w_k^T z)}{\sum_{k'=1}^{K} exp(w_{k'}^T z)})$$

$$= (y_k^{'} - 1) * y_k * z$$

when I realize my mistake, I corrected my mistake and did it as previous part.

When it comes to question 2 , I spent much more time on it. I have never used python or pytorch before, it took me a few days to know well python and pytorch. I have studied FNN model during my undergraduate study, so it didn't take too much time to implement FNN model base on pytorch. But talking about Spearman correlation, as I said before, it confused me for almost a week.