CS 224n: Assignment #5

1. Character-based convolutional encoder for NMT (36 points)

(a) (1 point) (written) In Assignment 4 we used 256-dimensional word embeddings (e word = 256), while in this assignment, it turns out that a character embedding size of 50 suffices (e char = 50).
In 1-2 sentences, explain one reason why the embedding size used for character-level embeddings is typically lower than that used for word embeddings.

Answer:

Most character itself doesn`t have meaning , it`s just part of word, but word itself have meanings. . And , Characters construct words, words construct sentence, So the information contained in the sentence is more than word, So we need larger dimensions for words than characters to store more information.

(b) (1 point) (written) Write down the total number of parameters in the character-based embedding model (Figure 2), then do the same for the word-based lookup embedding model (Figure 1). Write each answer as a single expression (though you may show working) in terms of e char , k, e word , V word (the size of the word-vocabulary in the lookup embedding model) and V char (the size of the character-vocabulary in the character-based embedding model).
Given that in our code, k = 5, V word ≈ 50, 000 and V char = 96, state which model has more parameters, and by what factor (e.g. twice as many? a thousand times as many?).

Answer:

Word based embedding model learnable parameter :

$$N_{word} = e_{word} \ast V_{word}$$

Character based embedding model learnable parameter:

$$N_{char} = e_{char} \ast V_{char}$$

Given that in our code, k = 5, V word ≈ 50, 000 and V char = 96, state which model has more parameters, and by what factor (e.g. twice as many? a thousand times as many?).

Word based emebedding has more parameters. About 63.7 times of char embedding.

In our case:
For CNN / CharEmbedding model:

$$N_{char} = n_{cnn} + n_{highway} + n_{¿} = (5 \ast 256 \ast 50 + 256) + (256 \ast 256 \ast 2 + 256 \ast 2) + (96 \ast 50) = 200640$$
$$N_{word} = 256 \ast 50000 = 12800000$$
$$N_{word} / N_{char} = 12800000 / 200640$$ **= 63.7**

(c) (2 points) (written) In step 3 of the character-based embedding model, instead of using a 1D con-vnet, we could have used a RNN instead (e.g. feed the sequence of characters into a bidirectional LSTM and combine the hidden states using max-pooling). Explain one advantage of using a con-volutional architecture rather than a recurrent architecture for this purpose, making it clear how the two contrast.

Answer:

Compare to conv1d, RNN has much more parameters to learn than conv1d. For conv1d, the learning parameter size is `in_channel * out_channel * kernel_size + bias_size` In our case it`s 256 * 50 * 5 + 256 = 64256. For RNN model, the paramater size is (input_hidden + hidden + bias_input + bias) * k , for example the decoder RNN (similar to Embedding RNN) in out case, the parameter number will be : (256 * 256 +256) * k, even we take k =1, it still about 200 times more parameters than conv1d. So Conv1d will be much cheaper to train.

(d) (4 points) (written) In lectures we learned about both max-pooling and average-pooling. For each pooling method, please explain one advantage in comparison to the other pooling method. For each advantage, make it clear how the two contrast, and write to a similar level of detail as in the example given in the previous question.

Answer:

Average pooling brings all into count and flows it to next layer which means all values actually are used for feature mapping and creating output - which is a very generalized computation and keep more information, but lack of emphasizing of important data. Max pooling extracts the most important features (good features) with "zeroing out" less important feature, so the important feature got emphasized, so model can learn more towards important features.

(h) (3 points) (coding and written)
Once you've finished testing your module, write a short description of the tests you carried out, and why you believe they are sufficient. The 3 points for this question are awarded based on your written description of the tests only.

Test 1:
Class initialization test:

Initialization a Highway module with a random parameter, e.g . 5. Then check all two linear layers both in and out size. It should match. E.g :

```
from highway import Highway

size = 5
hw = Highway(size)
assert hw.gate_layer.in_features == size
assert hw.proj_layer.out_features == size
```

Test 2:
Shapes and Types

Initialize a tensor with same size that we will be feed in to Highway module, check the module output (forward) shapes are expected sizes.

```
from highway import Highway
size = 5
hw = Highway(size)
t = torch.randn((5,5))
out = hw(t)
assert t.size() == out.size()
```

Test3 :
In and Out Test

Use torch.randn, or numpy to generate a certain size of input ,for example 5x5, then manually calculate the data follow the same data pipe line algorithms get the expected value, then initialize the linear weights to be all ones, (weights and bias), feed the same input data to Highway function. Compare the data and result. Make sure all intermediate values are printed out and compared with the manual calculate value.

Test 4:
Make sure Highway module only initialize once. Or we will fail to train the module, since the weights are all reinitialized.

To test this, add weight and bias initialize code for all 2 linear layser`s wegiht and bias in __init__() of the highway module.

add prnit out code for those 2 linear layers`s bias and weights. Start training, and log outputs. Make sure the weights and bias are getting updated no tre-initialized.

Reason:

1. Test1 will be make sure the module parameter are correct, or Test 2, Test3 will fail, and the model will fail.
2. Test2 will make sure the output shape are correct, so that we don`t have shape errors downstream
3. Test3 Will verify the correct layer / functions are used.
4. Test 4 will catch a hard to find issue cause model fail.

Above 4 cases covers initialize, input data, output data, so it looks like sufficient.

(i) (3 points) (coding and written) In the empty file cnn.py, implement the convolutional network Once you've finished testing your module, write a short description of the tests you carried out

Answer:

Test 1:
Class initialization test:

Initialization a CNN module with a . in channel =5 ,out channel= 4. kernel = 2 Then check conv1d layer both in and out channel value. It should match :

```
from highway import Highway

from cnn import CNN

in_channel= 5
out_channel = 4
k = 2
cnn = CNN(in_channel, out_channel)

assert cnn.cov1d_layer.in_channels == in_channel
assert cnn.cov1d_layer.out_channels == out_channel
assert cnn.cov1d_layer.kernel_size == k
```

Test 2:
Shapes and Types

Initialize a tensor with same size that we will be feed in to CNN module, check the module output (forward) shapes are expected sizes.

```
from highway import Highway
in_channel= 5
out_channel = 4
k = 2
cnn= CNN(size)
t = torch.randn((1,5,1))
out = cnn(t)
assert t.size() == out.size()
```

Test3 :
In and Out Test

Use torch.randn, or numpy to generate a certain size of input ,for example 1x5x1, then manually calculate the data follow the same data pipe line algorithms get the expected value, feed the same input data to CNN module. Compare the data and result. Make sure all intermediate values are printed out and compared with the manual calculate value.

Test 4:
Make sure CNN module only initialize once. Or we will fail to train the module, since the weights are all reinitialized.

To test this, add weight code for con1d layer

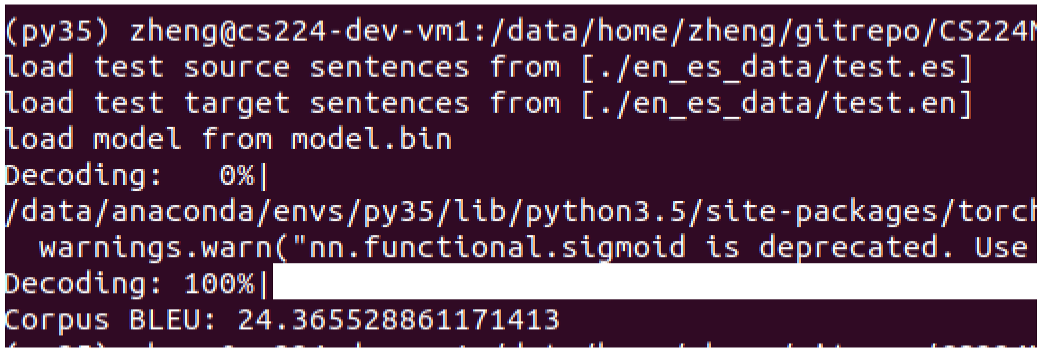
add print out code for those 2 linear layers`s bias and weights. Start training, and log outputs. Make sure the weights are getting updated not re-initialized.

Reason:

1. Test1 will be make sure the module parameter are correct, or Test 2, Test3 will fail, and the model will fail.
2. Test2 will make sure the output shape are correct, so that we don`t have shape errors downstream
3. Test3 Will verify the correct layer / functions are used.
4. Test 4 will catch a hard to find issue cause model fail.

Above 4 cases covers initialize, input data, output data, so it looks like sufficient.

(f)
BLEU = 24.365



```
(py35) zheng@cs224-dev-vm1:/data/home/zheng/gitrepo/CS224
load test source sentences from [./en_es_data/test.es]
load test target sentences from [./en_es_data/test.en]
load model from model.bin
Decoding:   0%|
/data/anaconda/envs/py35/lib/python3.5/site-packages/torc
  warnings.warn("nn.functional.sigmoid is deprecated. Use
Decoding: 100%|
Corpus BLEU: 24.365528861171413
```

3. Analyzing NMT Systems (8 points)

(a)
Answer:

“traducir” an “traduce” are in vocabulary .

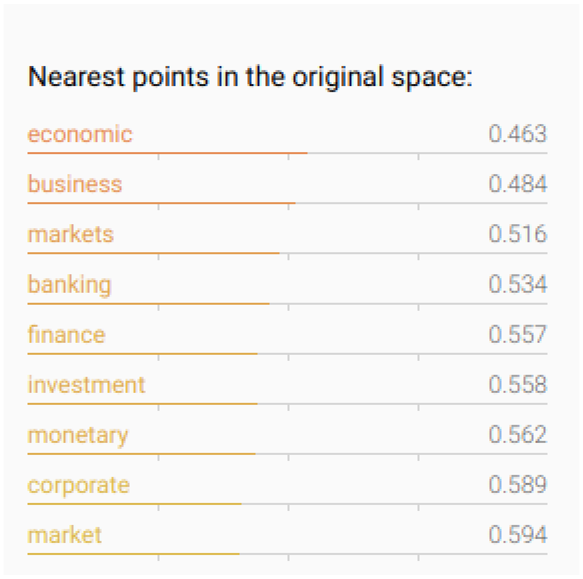
Traducir is the most frequent words in spanish training data.
I translate is most frequent words in english training data
traduzco is not in vocabulary, but in spanish training data
traduces is not in both vocabulary and training data
traduzca is not in vocabulary but in trainingdata
traduzcas is not in both vocabulary and training data.

If word doesn`t exists in vocabulary, it will generate <unk>, if the word never exists in training data, this system will not know this word.

New character-aware NMT model doesn`t lookup words, it generates words. So it doesn`t need lookup into the large predefined vocabulary. Only lookup the fixed char vocabulary, which is what we all need for all words.

(b)
i.

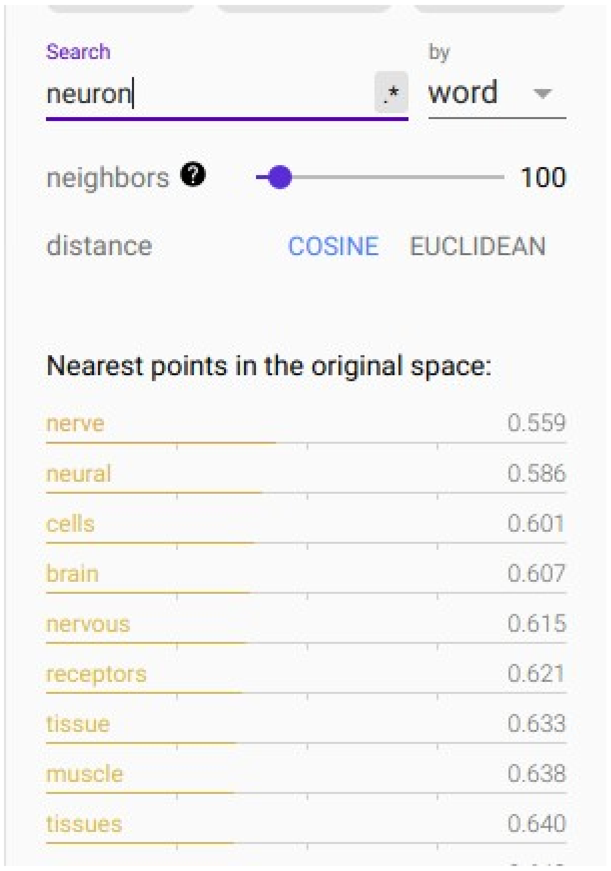
Answer:
• financial
economic is closest neighbor.



Nearest points in the original space:

economic	0.463
business	0.484
markets	0.516
banking	0.534
finance	0.557
investment	0.558
monetary	0.562
corporate	0.589
market	0.594

• neuron



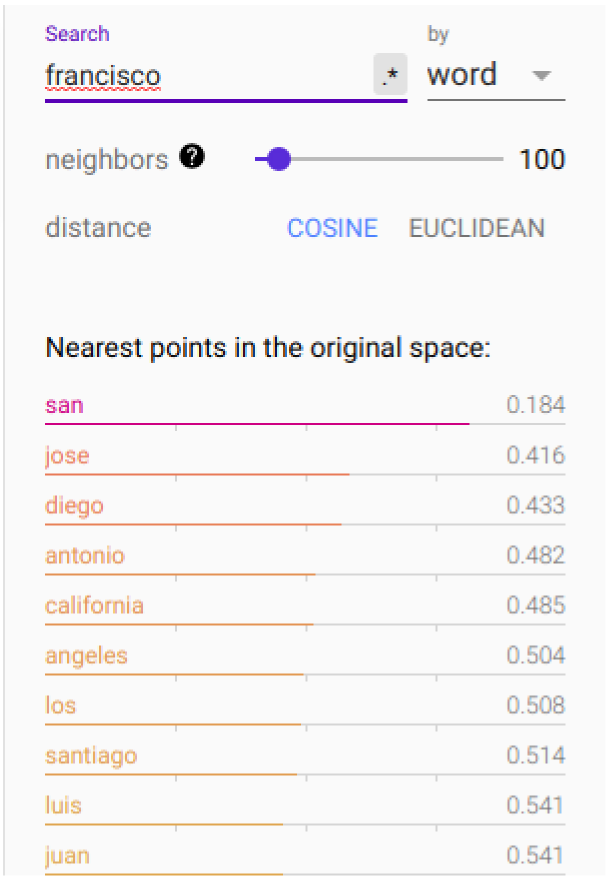
Search
neuron
by
word
neighbors 100
distance COSINE EUCLIDEAN

Nearest points in the original space:

nerve	0.559
neural	0.586
cells	0.601
brain	0.607
nervous	0.615
receptors	0.621
tissue	0.633
muscle	0.638
tissues	0.640

nerve is the closet

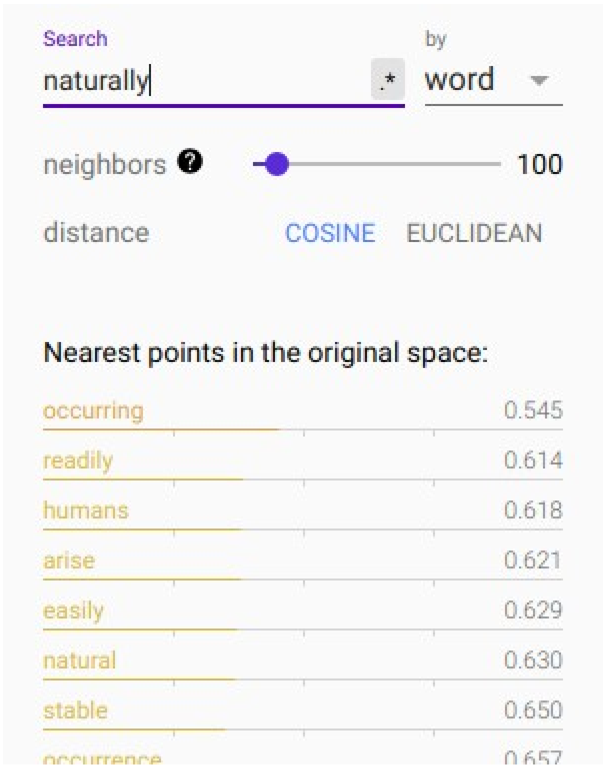
• Francisco
san is closest


Search: francisco
by: word
neighbors: 100
distance: COSINE EUCLIDEAN

Nearest points in the original space:

san	0.184
jose	0.416
diego	0.433
antonio	0.482
california	0.485
angeles	0.504
los	0.508
santiago	0.514
luis	0.541
juan	0.541

• naturally

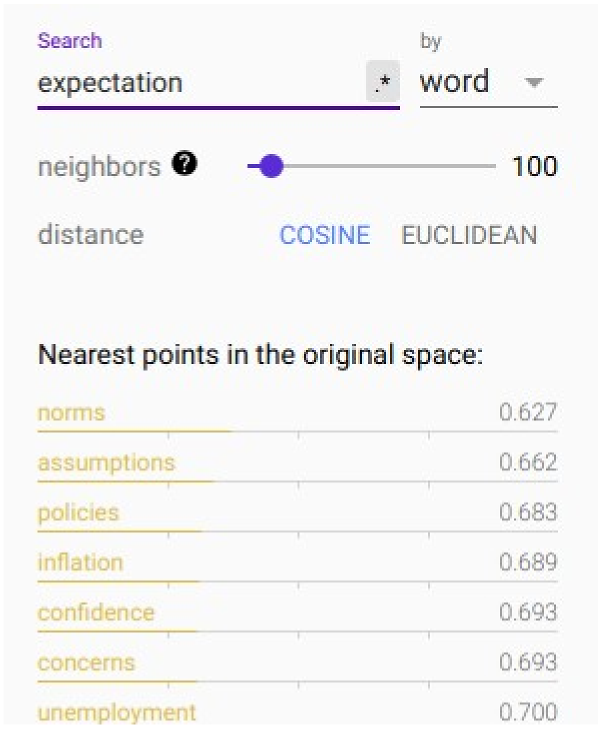

Search: naturally
by: word
neighbors: 100
distance: COSINE EUCLIDEAN

Nearest points in the original space:

occurring	0.545
readily	0.614
humans	0.618
arise	0.621
easily	0.629
natural	0.630
stable	0.650
occurrence	0.657

occurring is the closest

• expectation
norms is the closest



Search: expectation by word

neighbors 100

distance COSINE

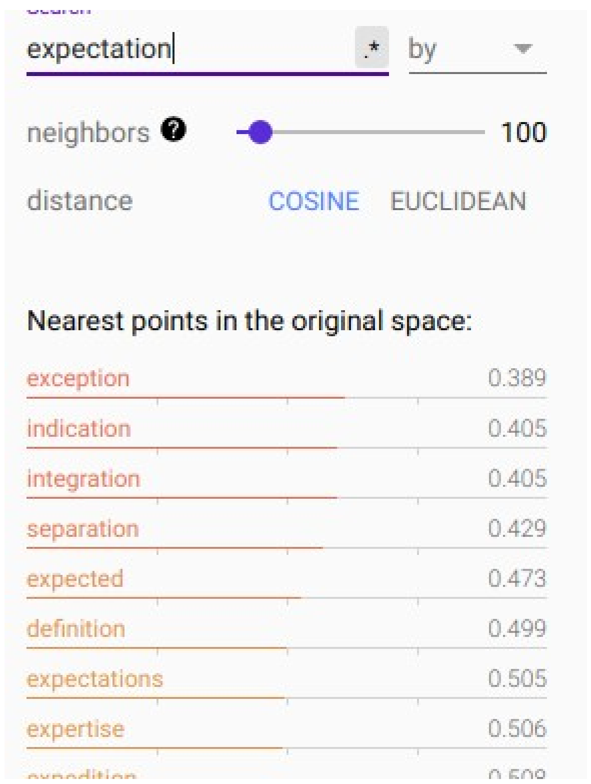
Nearest points in the original space:

norms	0.627
assumptions	0.662
policies	0.683
inflation	0.689
confidence	0.693
concerns	0.693
unemployment	0.700

ii. (0.5 points) (written) The TensorFlow embedding projector also allows you to upload your own data – you may find this useful in your projects!

Answer:

Choose word: expectation
closest : exception


expectation

neighbors 100

distance COSINE EUCLIDEAN

Nearest points in the original space:

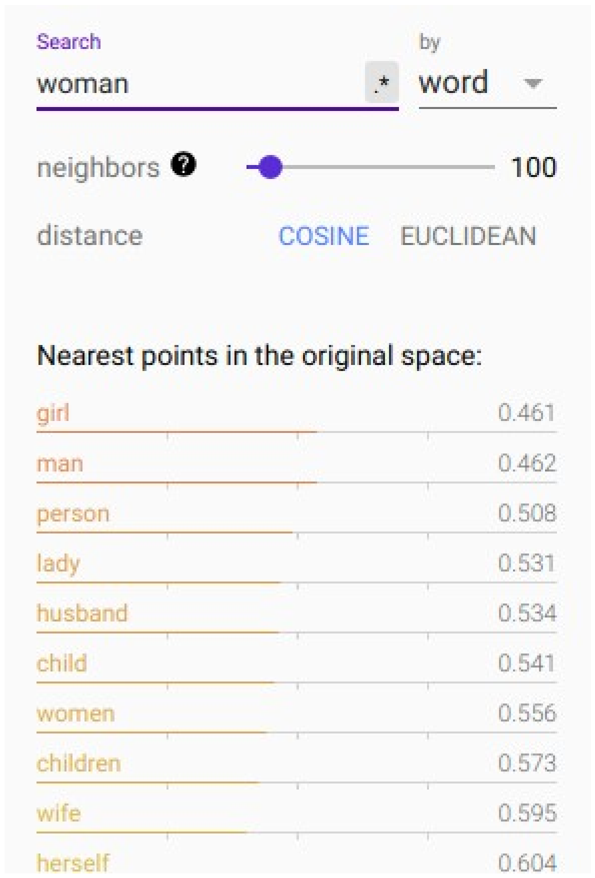
exception	0.389
indication	0.405
integration	0.405
separation	0.429
expected	0.473
definition	0.499
expectations	0.505
expertise	0.506

iii. (3 points) (written) Compare the closest neighbors found by the two methods. Briefly describe what kind of similarity is modeled by Word2Vec. Briefly describe what kind of similarity is modeled by the CharCNN. Explain in detail (2-3 sentences) how the differences in the methodology of Word2Vec and a CharCNN explain the differences you have found.

Answer

For Word: happy

word2vec:



Search: woman by word

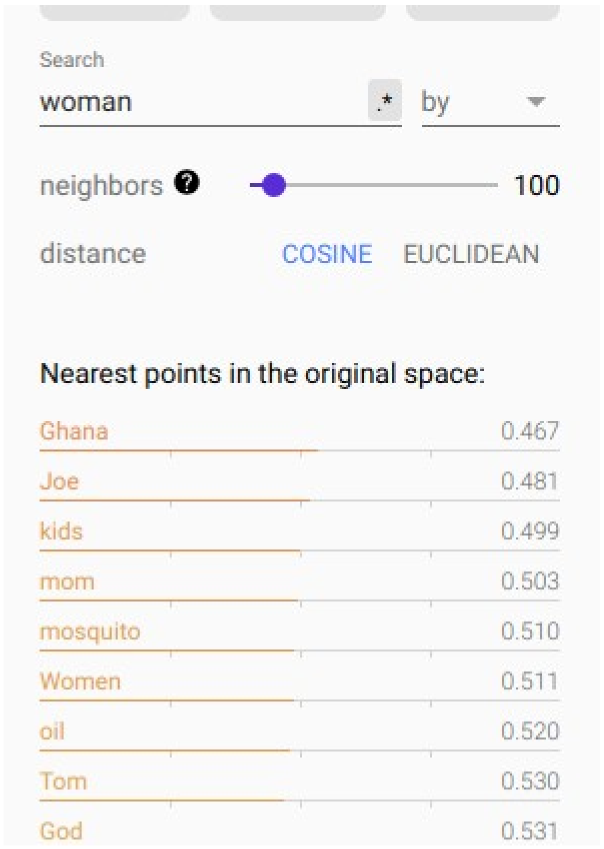
neighbors 100

distance COSINE

Nearest points in the original space:

girl	0.461
man	0.462
person	0.508
lady	0.531
husband	0.534
child	0.541
women	0.556
children	0.573
wife	0.595
herself	0.604

charCNN:



Search

woman

neighbors 100

distance COSINE EUCLIDEAN

Nearest points in the original space:

Ghana	0.467
Joe	0.481
kids	0.499
mom	0.503
mosquito	0.510
Women	0.511
oil	0.520
Tom	0.530
God	0.531

Word2Vec modeled has more similarity on meanings and some common atrributes and group together. Like wormam closet is gril, they all indicate female. CharCNN modeled is the similarity of property of words itself. The word of property may contain this meaning. Like kids could be girl (woman), could be boy(man).

CharCNN modeled similarity is sparse, cross different domains, it most likely similarly for the occurrence in the data set. And it exists bias.Word2Vec modeled similarity is related more integrity , similar attributes words grouped together.

(c) (2 points) (written) As in Assignment 4, we'll take a look at the outputs of the model that you have trained! The test set translations your model generated in 2(f) should be located in the outputs directory at: outputs/test outputs.txt. We also provided translations from a word-based model from our Assignment 4 model in the file outputs/test outputs a4.txt.
Find places where the word-based model produced <UNK>, and compare to what the character-based decoder did. Find one example where the character-based decoder produced an acceptable translation in place of <UNK>, and one example where the character-based decoder produced an incorrect translation in place of <UNK>. As in Assignment 4, 'acceptable' and 'incorrect' doesn't just mean 'matches or doesn't match the reference translation' – use your own judgment (and Google Translate, if necessary). For each of the two examples, you should:
1. Write the source sentence in Spanish. The source sentences are in en es data/test.es.

2. Write the reference English translation of the sentence. The reference translations are in en es data/test.en.
3. Write the English translation generated by the model from Assignment 4. These translations are in outputs/test outputs a4.txt. Underline the <UNK> you are talking about.
4. Write your character-based model's English translation. These translations are in outputs/test outputs.txt. Underline the CharDecoder-generated word you are talking about.
5. Indicate whether this is an acceptable or incorrect example. Give a brief possible explanation (one sentence) for why the character-based model performed this way.

Answer:

Example 1:

Spanish: Puedo vestirme como agricultor, o con ropa de cuero, y nunca nadie ha elegido un agricultor.

Reference: You can have me as a farmer, or in leathers, and no one has ever chose farmer.

A4 Translation: I can <unk> like farmer, or with leather <unk> and no one has chosen a farmer.

CharModel Translation: I can look like a farmer, or with leather clothes, and nobody has chosen a farmer.

This s acceptable example. CharModel is generating words based on the context words.

Example 2:

Spanish: Un amigo mo hizo eso -- Richard Bollingbroke.

Reference: A friend of mine did that -- Richard Bollingbroke.

A4 Translation: A friend of mine did that -- Richard <unk>

CharModelTranslation: A friend of mine did that -- Richard Bollywood.

This is incorrect example. CharModel will always generated some words based on the context words. So for names it just check the context and generated closest words. It could not exactly match for a name.