Stanford CS 224n Assignment 5

Hanchung Lee

February 20, 2020

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: There are only several dozens of alpha-numerical characters in writing but tens of thousands of words. Therefore character-level embeddings will have a much less dimension than word-level embeddings. More, because English is a phonetic language, the individual characters have much less if any meaning but individual words contains meanings, and the character dependencies are not as long ranged compare to words.

(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, $k = 5$, $V_{word} \approx 50,000$ 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: Char-based embedding total param is sum of the character indices embedding lookup, 1D convolution, and highway networks (X_{proj} and X_{gate}).

$$\begin{aligned}
Param_{char-embeddings} &= Param_{emb} + Param_{conv} + Param_{highway} \\
&= (V_{char} \times e_{char}) + (e_{char} \times k \times e_{word} + e_{word}) + 2 \times (e_{word} \times e_{word} + e_{word}) \\
&= (96 \times 50) + (50 \times 5 \times 256 + 256) + 2 \times (256 \times 256 + 256) \\
&= 200,640 \\
Param_{word-embeddings} &= V_{word} \times e_{word} \\
&= 50,000 \times 256 \\
&= 12,800,000
\end{aligned}$$

Therefore, the word embedding model contains almost 64x as many parameters than character embedding model.

(c) (2 points) (written) In step 3 of the character-based embedding model, instead of using a 1D convnet, we could have used a RNN instead (e.g. feed the sequence of characters into a bi-directional LSTM and combine the hidden states using max-pooling). Explain one advantage of using a convolutional architecture rather than a recurrent architecture for this purpose, making it clear how the two contrast. Below is an example answer; you should give a similar level of detail and choose a different advantage.

Answer:

i) When a 1D convnet computes features for a given window of the input, those features depend on the window only – not any other inputs to the left or right. By contrast, a RNN needs to compute the hidden states sequentially, from left to right (and also right to left, if the RNN is bidirectional). Therefore, unlike a RNN, a convnet's features can be computed in parallel, which means that convnets are generally faster, especially for long sequences.

ii) 1D convnet uses a fixed window that extracts positional information on both sides all at the same time opposed to RNN have to unroll the time steps either uni-directionally or bi-directionally. This means it will have a faster training time. In addition, it can be designed to 'zoom' into a deeper latent features using more filters.

iii) In this assignment, a 1D convnet has a $50 \times 5 \times 256 + 256 = 64,256$ parameters. An RNN with the same output dimension would have $(256 \times 256 + 256) = 65,792$ parameters trained over k time steps. However, if we use alternate RNN models such as LSTM or GRU, the number of parameters will increase by 4x for LSTM and 3x for GRU.

(d) (4 points) (written) In lectures we learned about both max-pooling and average-pooling. or 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:

	Advantages	**Disadvantages**
max pooling	Keeps only the strongest features while zeroing out the lesser features	Information loss from disregarding lessor features
average pooling	Maintains all of the features	Stronger features got diluted by lesser smaller features

(f) (4 points) (coding and written) In the empty file `highway.py`, implement the highway network as a `nn.Module` class called `Highway`. 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 4 points for this question are awarded based on your written description of the tests only.

Answer: Setup two test cases. Cannot be sure if the edge cases are tested or if the test cases are sufficient.

1 Test initialization, shape of model weights, and if biases are implemented.

2 Test numerical results by first manually calculating the expected result using Numpy. Then feed the same Numpy input matrix to the `nn.Module Highway` class. And finally compare the calculated Numpy output vs the output from `Highway` module.

(g) (4 points) (coding and written) In the empty file `cnn.py`, implement the convolutional network as a `nn.Module` class called `CNN`. As in (f), write code to thoroughly test your implementation. Once you've finished testing yourmodule, write a short description of the tests you carried out, and why you believe they are sufficient.As in (f), the 4 points are for your written description only.

Answer: Setup one test case1. Choose not to do manual hand calculation of 1D Conv as that might induce unintentional bugs. Doesn't make sense to write test cases to generate test case data. Cannot be sure if the edge cases are tested or if the test cases are sufficient. EDIT: No, this is definitely not enough. At training small data set time, even when dimensions were fine, the kernel size being too big will throw off `maxpool1d`. And Using `ceil_mode=True` to fix that problem might cause the `maxpool1d` dimension to be greater than 1. Perhaps need to write multiple test cases to test different types of inputs.

1. Test initialization, shape of model weights, and if biases are implemented.

2 Character-based LSTM decoder for NMT (26 points)

(e) (6 point) (coding) Report your test set BLEU score in your assignment write-up. Also ensure that the output fileo `outputs/test_outputs.txt` is present and unmodified – this will be included in your submission, and we'll use it to verify your self-reported BLEU score.

Answer: After 2342542 hours of traing on RTX 2080 Ti on my local machine, `BLEU score` is 234.54325324.

3 Analyzing NMT Systems (8 points)

(a) (2 points) (written) The following table shows some of the forms of the Spanish word *traducir*, which means *to translate*.

Infinitive	**traducir**	**to translate**
Present	traduzco	I translate
	traduces	you translate
	traduce	that you translate
Subjective	traduzca	that I translate
	traduzcas	that you translate

Use `vocab.json` to find (e.g. using `grep`) which of these six forms are in the word-vocabulary, which consists of the 50,000 most frequent words in the training data for English and for Spanish. Superstrings don't count (e.g. having `traducen` in the vocabulary is not a hit for `traduce`). State which of these six forms occur, and which do not. Explain in one sentence why this is a bad thing forword-based NMT from Spanish to English. Then explain in detail (approximately two sentences) how our new character-aware NMT model may overcome this problem.

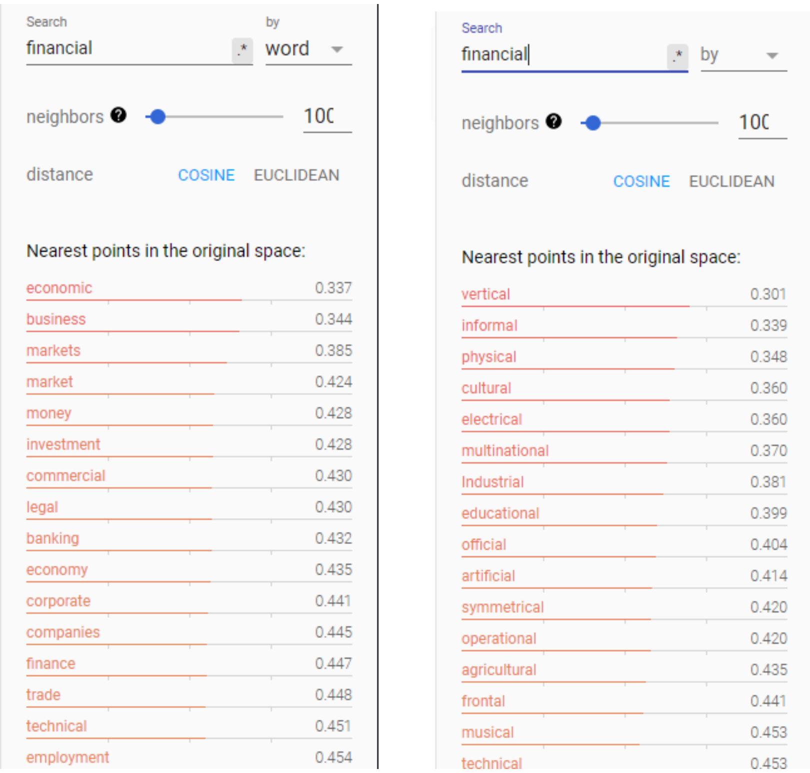
Answer: `traducir, idx:4630`, `traduce, idx:7931`, and `traduzco, idx:40991` are in the word-vocabulary while `traducuces`, `traduzca`, and `traduzcas` are not in the vocabulary. This is a problem as the variations of the word will be out-of-vocabulary, get tagged as unknown, and not get recognized by the network. With a character-aware NMT, when an unknown word was encountered it will generate the OOV word use the character level decoder path and try to generate a character sequence using greedy search in the algorithm.

(b) i. (0.5 points) (written) In Assignments 1 and 2, we investigated word embeddings created via algorithms such a Word2Vec, and found that for these embeddings, semantically similar words are close together in the embedding space. In this exercise, we'll compare this with the word embeddings constructed using the CharCNN trained in our NMT system. For each word, report the single closest neighbor. For your convenience, for each example takea screenshot of all the nearest words (so you can compare with the CharCNN embeddings).

Answer: As follows. Screenshot included in ii.

- `financial` → economic
- `neuron` → neurons
- `Francisco` → san
- `naturally` → occuring
- `expectation` → operator

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! Download the character-based word embeddings obtained from our implementation of the character-aware NMT model from this link. Navigate to https://projector.tensorflow.org/, select `Load Data`, and upload the files `character-embeddings.txt` (the embeddings themselves) and `metadata.txt` (the words associated with the embeddings). Now look at the nearest neighbors of the same words. Again, report the single closest neighbors with dataset `Word2Vec All` and take screenshots for yourself.



Search financial .* by word

neighbors 10(

distance COSINE EUCLIDEAN

Nearest points in the original space:

economic	0.337
business	0.344
markets	0.385
market	0.424
money	0.428
investment	0.428
commercial	0.430
legal	0.430
banking	0.432
economy	0.435
corporate	0.441
companies	0.445
finance	0.447
trade	0.448
technical	0.451
employment	0.454

Search financial .* by

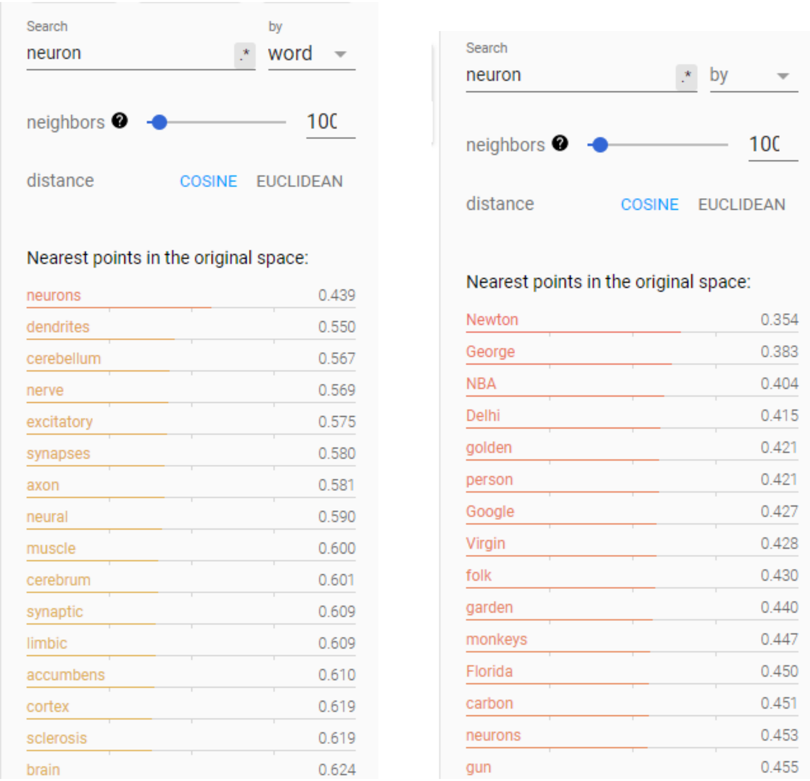
neighbors 10(

distance COSINE EUCLIDEAN

Nearest points in the original space:

vertical	0.301
informal	0.339
physical	0.348
cultural	0.360
electrical	0.360
multinational	0.370
Industrial	0.381
educational	0.399
official	0.404
artificial	0.414
symmetrical	0.420
operational	0.420
agricultural	0.435
frontal	0.441
musical	0.453
technical	0.453

Figure 1: financial. Word2Vec Left, CharCNN Right



Search neuron .* by word

neighbors 10(

distance COSINE EUCLIDEAN

Nearest points in the original space:

neurons	0.439
dendrites	0.550
cerebellum	0.567
nerve	0.569
excitatory	0.575
synapses	0.580
axon	0.581
neural	0.590
muscle	0.600
cerebrum	0.601
synaptic	0.609
limbic	0.609
accumbens	0.610
cortex	0.619
sclerosis	0.619
brain	0.624

Search neuron .* by

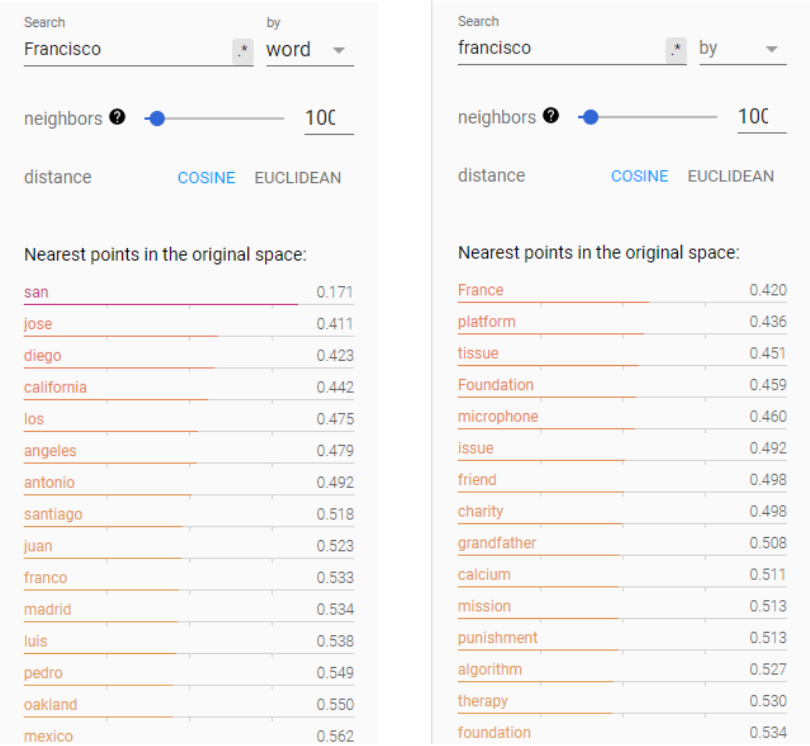
neighbors 10(

distance COSINE EUCLIDEAN

Nearest points in the original space:

Newton	0.354
George	0.383
NBA	0.404
Delhi	0.415
golden	0.421
person	0.421
Google	0.427
Virgin	0.428
folk	0.430
garden	0.440
monkeys	0.447
Florida	0.450
carbon	0.451
neurons	0.453
gun	0.455

Figure 2: neuron. Word2Vec Left, CharCNN Right



Search

Francisco .* by word

neighbors 10C

distance COSINE EUCLIDEAN

Nearest points in the original space:

san	0.171
jose	0.411
diego	0.423
california	0.442
los	0.475
angeles	0.479
antonio	0.492
santiago	0.518
juan	0.523
franco	0.533
madrid	0.534
luis	0.538
pedro	0.549
oakland	0.550
mexico	0.562

Search

francisco .* by

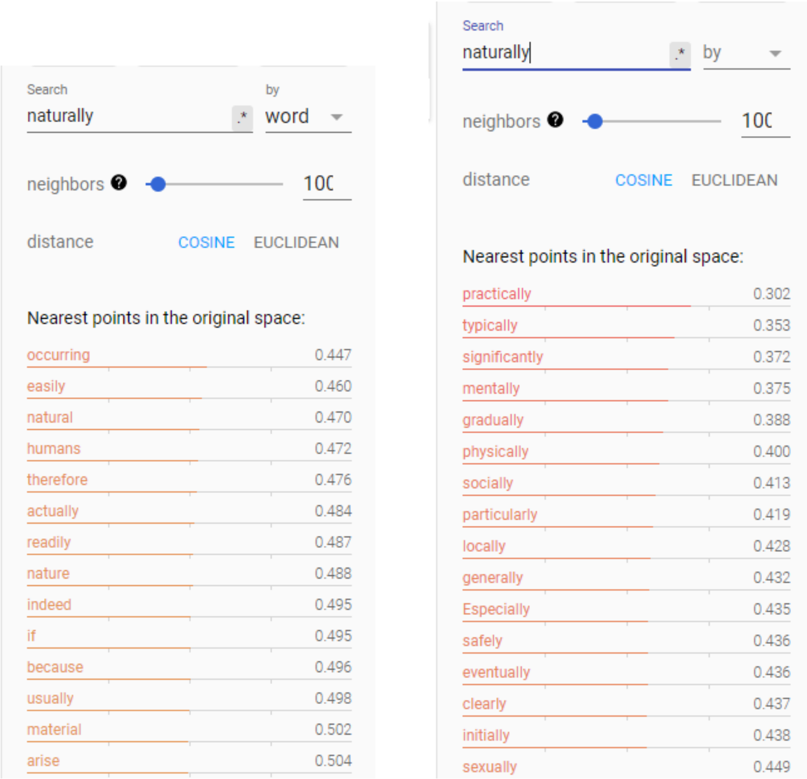
neighbors 10C

distance COSINE EUCLIDEAN

Nearest points in the original space:

France	0.420
platform	0.436
tissue	0.451
Foundation	0.459
microphone	0.460
issue	0.492
friend	0.498
charity	0.498
grandfather	0.508
calcium	0.511
mission	0.513
punishment	0.513
algorithm	0.527
therapy	0.530
foundation	0.534

Figure 3: Francisco. Word2Vec Left, CharCNN Right



Search

naturally .* by word

neighbors 10C

distance COSINE EUCLIDEAN

Nearest points in the original space:

occurring	0.447
easily	0.460
natural	0.470
humans	0.472
therefore	0.476
actually	0.484
readily	0.487
nature	0.488
indeed	0.495
if	0.495
because	0.496
usually	0.498
material	0.502
arise	0.504

Search

naturally .* by

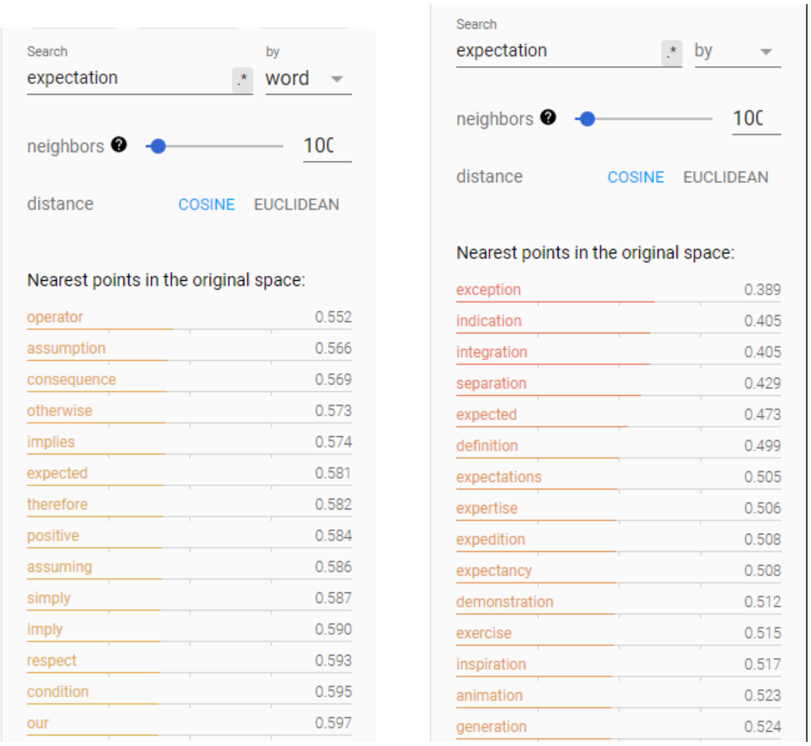
neighbors 10C

distance COSINE EUCLIDEAN

Nearest points in the original space:

practically	0.302
typically	0.353
significantly	0.372
mentally	0.375
gradually	0.388
physically	0.400
socially	0.413
particularly	0.419
locally	0.428
generally	0.432
Especially	0.435
safely	0.436
eventually	0.436
clearly	0.437
initially	0.438
sexually	0.449

Figure 4: naturally. Word2Vec Left, CharCNN Right



Search

expectation .* by word

neighbors ❓ 10C

distance COSINE EUCLIDEAN

Nearest points in the original space:

operator	0.552
assumption	0.566
consequence	0.569
otherwise	0.573
implies	0.574
expected	0.581
therefore	0.582
positive	0.584
assuming	0.586
simply	0.587
imply	0.590
respect	0.593
condition	0.595
our	0.597

Search

expectation .* by

neighbors ❓ 10C

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
expedition	0.508
expectancy	0.508
demonstration	0.512
exercise	0.515
inspiration	0.517
animation	0.523
generation	0.524

Figure 5: expectation. Word2Vec Left, CharCNN Right

iii. (3 points) (written) Compare the closest neighbors found by the two methods: briefly describe what kind of similarity is modeled by Word2Vec, and 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: Word2Vec models focuses on sematic similarity between neighboring words, i.e., word vectors is about its context. CharCNN model looks at the structural similarity of our particular dataset using a window-based feature, so words similar in word structure will be closer in feature space.

(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(e) 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).

Answer: