THE UNIVERSITY OF SYDNEY

Sample Take-home Exam Paper

Semester 1, 2022

COMP5046 Natural Language Processing

*This sample exam paper contains one exam question for each weekly topic, and it is for sharing the structure and style of the final exam. You can see which types of questions can be 4 mark short questions and 18 mark essay questions.

*Please do not write your answer by hand. (For only drawing, you can do it either by hand or computer.)

*Your answer (including drawings and illustration MUST be written by you.) For the illustration, you MUST NOT copy from other resources.

*The final exam will be an open-book, unsupervised exam.

Week 1 & 2. Word Representation

Q. Explain the difference between FastText and Word2Vec with examples. (4 marks)

Solution

Word2vec treats each word in the corpus like an atomic entity and generates a vector for each word. It treats words as the smallest unit to train on. Word2Vec learns vectors only for complete words found in the training corpus so shows Out-of-Vocabulary (OOV) cases for unseen words.

FastText, an extension of the word2vec model, treats each word as composed of character n-grams. So the vector for a word is made of the sum of this character n-grams. For example, the word vector "aquarium" is a sum of the vectors of the n-grams: "<aq/aqu/qua/uar/ari/riu/ium/um>". Note that "<" and ">" means Start of word and End of word.
As Word Embedder encounters the word "Aquarius", it might not recognize it, but it can guess by the sharing part in "aquarium" and "Aquarius", to embed Aquarius near the aquarium.

Hence, FastText learns vectors for the n-grams that are found within each word, as well as each complete word. The N-gram feature is the most significant improvement in FastText, it's designed to solve OOV issues.

Week 3 and 4. Word Classification with Machine Learning

Q. In class, we learned that the family of recurrent neural networks have many important advantages and can be used in a variety of NLP tasks. For each of the following tasks and inputs, state how you would run an RNN to do that task. **(4 marks)**

1. how many outputs i.e. the number of times the softmax $\widehat{y}(t)$is called from your RNN. If the number of outputs is not fixed, state it as arbitrary
2. what each $\widehat{y}(t)$is a probability distribution over
3. which inputs are fed at each time step to produce each output

Task A: Named-Entity Recognition: For each word in a sentence, classify that word as either a person, organization, location, or none. (Inputs: A sentence containing n-words)

Task B: Sentiment Analysis: Classify the sentiment of a sentence ranging from negative to positive (integer values from 0 to 4). (Inputs: A sentence containing n-words.)

Solution
Task A: Named Entity Recognition
1. Number of Outputs: n outputs
2. Each $\widehat{y}(t)$is a probability distribution over 4 NER categories.
3. Each word in the sentence is fed into the RNN and one output is produced at every time step corresponding to the predicted tag/category for each word.

Task B: Sentiment Analysis
1. Number of Outputs: 1 output. (n outputs is also acceptable if it takes the average of all outputs)
2. Each $\widehat{y}(t)$is a probability distribution over 5 sentiment values.
3. Each word in the sentence is fed into the RNN and one output is produced from the hidden states (by either taking only the final, max or mean across all states) corresponding to the sentiment value of the sentence.

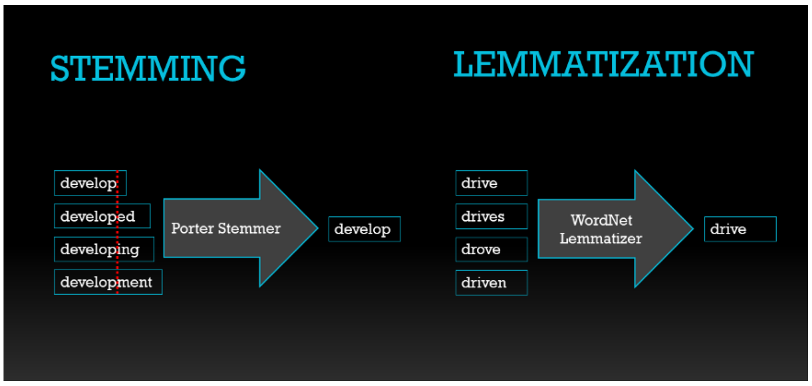
Week 5. Language Fundamental

Q. Describe the difference between lemmatization and stemming. Give application examples and illustrations to support your argument (4 marks)

Solution

Stemming is a procedure to reduce all words with the same stem to a common form whereas lemmatization removes inflectional endings and returns the base or dictionary form of a word. For example, the words “trouble”, “troubling” and “troubled” may be stemmed to be “troubl” (not a valid English word) but will be lemmatized to be “trouble” for comparison. Also, another good example of lemmatization would be the words “was”, “is” to be mapped to “be”.

The following illustration shows the difference between lemmatization and stemming as described above.



(This is just a sample example - YOU MUST NOT copy from other resources. It MUST be your own drawing)

Week 6. Part of Speech Tagging

Q. Given the sentence "I promise to back the bill.", **show how you would compute the probability of** "back" **as a verb versus the probability of** "back" **as a noun using the probabilities in Tables a and b using the Viterbi algorithm. You are given the values for the third column of the Viterbi table which corresponds to observation 3 or** "to"**. They are VB: 0, TO: .00000018, NN: 0, PRP: 0. Thus, you will show two computations both of which will use these values. You do not need to do the arithmetic; just show the formula that would be computed. (4 marks)**

*(*assume all verb tags as VB)*

Table a. *Observation Likelihoods*

	I	***promise***	***to***	***back***
VB	***0***	***.0093***	***0***	***.00008***
TO	***0***	***0***	***.99***	***0***
NN	***0***	***.0085***	***0***	***.00068***
PRP	***.37***	***0***	***0***	***0***

Table b. *Tag transition probabilities.*

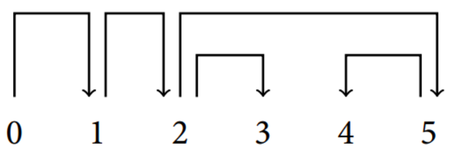
	VB	*TO*	*NN*	*PRP*
<s>	***.019***	***.0043***	***.041***	***.067***
VB	***.0038***	***.035***	***.047***	***.0070***
TO	***.83***	***0***	***.00047***	***0***
NN	***.0040***	***.016***	***.087***	***.0045***
PRP	***.23***	***.00079***	***.0012***	***.00014***

Solution

- back as a verb:
  .00000018 * Prob(VB| TO) * Prob (back |VB) = .00000018 *.83 * .00008
- back as a noun:
  .00000018* Prob (NN | TO) * Prob (back |NN) = .00000018 * .00047 * .00068

Week 7. Dependency Parsing

Q. State a sequence of transitions that make a transition-based dependency parser produce the following dependency tree. Explain how to get the sequence of transitions. (4 marks)



Solution

Suppose SH = Shift, RA = Right Arc, LA = Left Arc.
SH SH SH SH RA SH SH LA RA RA RA

In order to get this dependency tree using the arc-standard algorithm, we need to do the following steps based on the three possible transactions (SH, RA, LA):
Step 1. SH the ROOT 0 to the stack while all the other words from 1 to 5 will be in the buffer as our initial state.
Step 2. SH the 1 from buffer to the stack
Step 3. SH the 2 from buffer to the stack
Step 4. SH the 3 from buffer to the stack
Step 5. RA from 2 to 3 and remove 3 out of stack
Step 6. SH 4 from buffer to the stack
Step 7. SH 5 from the buffer to the stack
Step 8. LA from 5 to 4 and remove 4 out of stack
Step 9. RA from 2 to 5 and remove 5 out of stack
Step 10. RA from 1 to 2 and remove 2 out of stack
Step 11. RA from 0 to 1 and remove 1 out of stack

Head and modifier refer to the two words in a dependency relation where the head is the one that is governor, parent and the modifier is the one that is dependen, daughter. Using the arrow for the dependencies, it will point from the head to the modifier. For example, considering the dependency of words ‘red hat’, the red will be the modifier while the hat will be the head. And the arrow will point from “hat” to “red” in this case.

And, please put a detailed explanation on how to get the dependency tree using SH, LA, RA definition and what the head and modifier are.

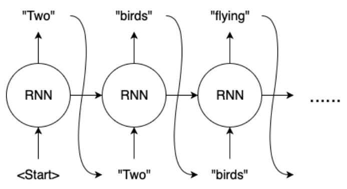
Week 8. Language Model and Natural Language Generation

Q. During training a neural language model, we normally apply teacher forcing. Describe what the teacher forcing technique is. Give application examples and drawings to support your argument. (4 marks)

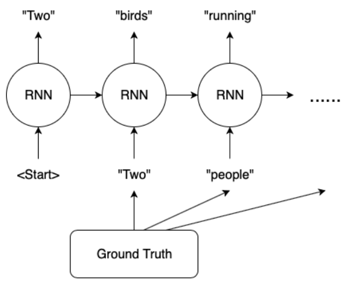
Solution

Teacher forcing is the technique where the target word is passed as the next input to the decoder. Let us assume we want to train a sentence generation model, and the ground truth caption for the above image is "Two people reading a book". Our model makes a mistake in predicting the 2nd word and we have "Two" and "birds" for the 1st and 2nd prediction respectively. If we use Teacher Forcing, we would feed "people" to our RNN for the 3rd prediction, after computing and recording the loss for the 2nd prediction.

Without Teacher Forcing, we would feed "birds" back to our RNN to predict the 3rd word. Let's say the 3rd prediction is "flying". Even though it makes sense for our model to predict "flying" given the input is "birds", it is different from the ground truth.



Without Teacher Forcing



With Teacher Forcing

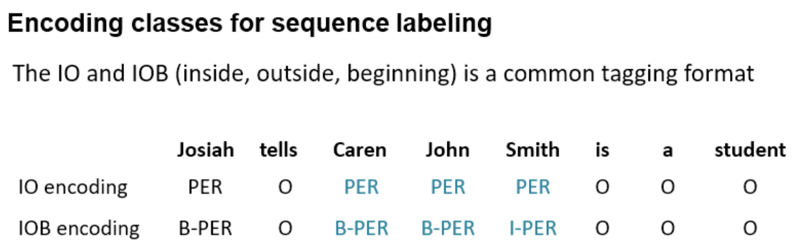
(This is just a sample example - YOU MUST NOT copy from other resources. It MUST be your own drawing)

Week 9. Named Entity Recognition and Coreference Resolution

Q. The IOB format categorizes tagged tokens as I, O and B. Why are three tags necessary? What problem would be caused if we used I and O tags exclusively? Give application examples and illustrations to support your argument. (4 marks)

Solution

The IOB format (short for inside, outside, beginning) is a common tagging format for tagging tokens in a chunking task. If two chunks follow each other, it would not be possible to make clear that they are two chunks instead of one chunk consisting of two words and also not where the first ends and the second begins. For example, considering the NER tags using only IO format for the sentence 'Josiah/I-PER, tells/O, Caren/I-PER, John/I-PER Smith/I-PER is/O a/O student/O', the two words Caren and John cannot be distinguished as separate two chunks as expected. However, this can be solved by using the IOB format as 'Josiah/B-PER, tells/O, Caren/B-PER, John/B-PER Smith/I-PER is/O a/O student/O'.



Encoding classes for sequence labeling

The IO and IOB (inside, outside, beginning) is a common tagging format

	Josiah	**tells**	**Caren**	**John**	**Smith**	**is**	**a**	**student**
IO encoding	PER	O	PER	PER	PER	O	O	O
IOB encoding	B-PER	O	B-PER	B-PER	I-PER	O	O	O

Week 10. Attention and Reading Comprehension

Q. Describe the main intuition behind attention in a neural network model and describe how to calculate attention (You can use any attention mechanism - e.g. Encoder-decoder attention, self-attention, or any attention score calculation - e.g. dot product) with an illustration. (4 marks)

Solution

Using the encoder-decoder architecture, An encoder processes the input sequence and compresses the information into a context vector (last hidden state) of a fixed length. A critical and apparent disadvantage of this fixed-length context vector design is the incapability of remembering long sentences. Often it has forgotten the first part once it completes processing the whole input. Rather than building a single context vector out of the encoder's last hidden state, the secret sauce invented by attention is to create shortcuts between the context vector and the entire source input. The weights of these shortcut connections are customizable for each output element.

The following illustration shows the calculation of attention in seq2seq models.
(NOTE: you need to draw and explain the process like we did in the lecture 10, page 30 to 35)