School of Computing and Information Systems
The University of Melbourne
COMP90042
NATURAL LANGUAGE PROCESSING (Semester 1, 2022)

Sample solutions: Week 5

Discussion

1. How does a neural network language model (feedforward or recurrent) handle a large vocabulary, and how does it deal with sparsity (i.e. unseen sequences of words)?

   - A neural language model projects words into a continuous space and represents each word as a low dimensional vector known as word embeddings. These word embeddings capture semantic and syntactic relationships between words, allowing the model to generalise better to unseen sequences of words.
   - For example, having seen the sentence *the cat is walking in the bedroom* in the training corpus, the model should understand that *a dog was running in a room* is just as likely, as (*the*, *a*), (*dog*, *cat*), (*walking*, *running*) have similar semantic/grammatical roles.
   - Count-based N-gram language model would struggle in this case, as (*the*, *a*) or (*dog*, *cat*) are distinct word types from the model's perspective.

2. Why do we say most parameters of a neural network (feedforward or recurrent) language model is in their input and output word embeddings?

   - Assume we have a vocabulary size of 10K word types, and we are using the feedforward neural network language model in the lecture (L7 page 27). If the dimension of the embeddings and the hidden representation (h) is 300, the number of parameters in our model is:
     - input word embeddings $(W_1) = 300 \times 10K = 3,000,000$
     - $W_2 = 300 \times 900 = 270,000$
     - $b_1 = 300$
     - output word embeddings $(W_3) = 10K \times 300 = 3,000,000$
   - Similarly for a simple recurrent neural language model (L8 page 14):
     - input word embeddings $W_x = 300 \times 10K = 3,000,000$
     - $W_s = 300 \times 300 = 90,000$
     - $b = 300$
     - output word embeddings $(W_y) = 10K \times 300 = 3,000,000$

3. What advantage does an RNN language model have over N-gram language model?

   - RNN language model can capture arbitrarily long contexts, as opposed to an N-gram language model that uses a fixed-width contexts.
   - RNN does so by processing a sequence one word at a time, applying a recurrence formula and using a state vector to represent words that have been previously processed.

4. What is the vanishing gradient problem in RNN, and what causes it? How do we tackle vanishing gradient for RNN?
    - An unrolled RNN is just a very deep network.
    - When we do backpropagation to compute the gradients, the gradients tend to get smaller and smaller as we move backward through the network.
    - The net effect is that neurons in the earlier layers learn very slowly, as their gradients are very small. If the unrolled RNN is deep enough we might start seeing gradients “vanish”.
    - More sophisticated RNN models such as LSTM or GRU are introduced to tackle vanishing gradients.
    - They do so by introducing “memory cells” that preserve gradients across time.

Programming

1. In the iPython notebook `07-deep-learning`:
    - Can you find other word pairs that have low/high similarity? Try to look at more nouns and verbs, and see if you can find similarity values that are counter-intuitive.
    - We can give the neural models more learning capacity if we increase the dimension of word embeddings or hidden layer. Try it out and see if it gives a better performance. One thing that we need to be careful when we increase the number of model parameters is that it has a greater tendency to “overfit”. We can tackle this by introducing dropout to the layers (`keras.layers.Dropout`), which essentially set random units to zero during training. Give this a try, and see if it helps reduce overfitting.
    - Improve the bag-of-words feed-forward model with more features, e.g. bag-of-N-grams, polarity words (based on a lexicon), occurrence of certain symbols (!).
    - Can you incorporate these additional features to a recurrent model? How?

Get ahead

- While `keras` is a great library for learning how to build basic deep learning models, it is often not as flexible as `pytorch`, due to its high level of abstraction. Follow the pytorch tutorial (`https://pytorch.org/tutorials/`) and learn how to build a word level language model in one of its examples (`https://github.com/pytorch/examples/tree/master/word_language_model`).