UDACITY

PROJECT

Translation From One Language to Another Language

A part of the Deep Learning Nanodegree Foundation Program

PROJECT REVIEW

CODE REVIEW

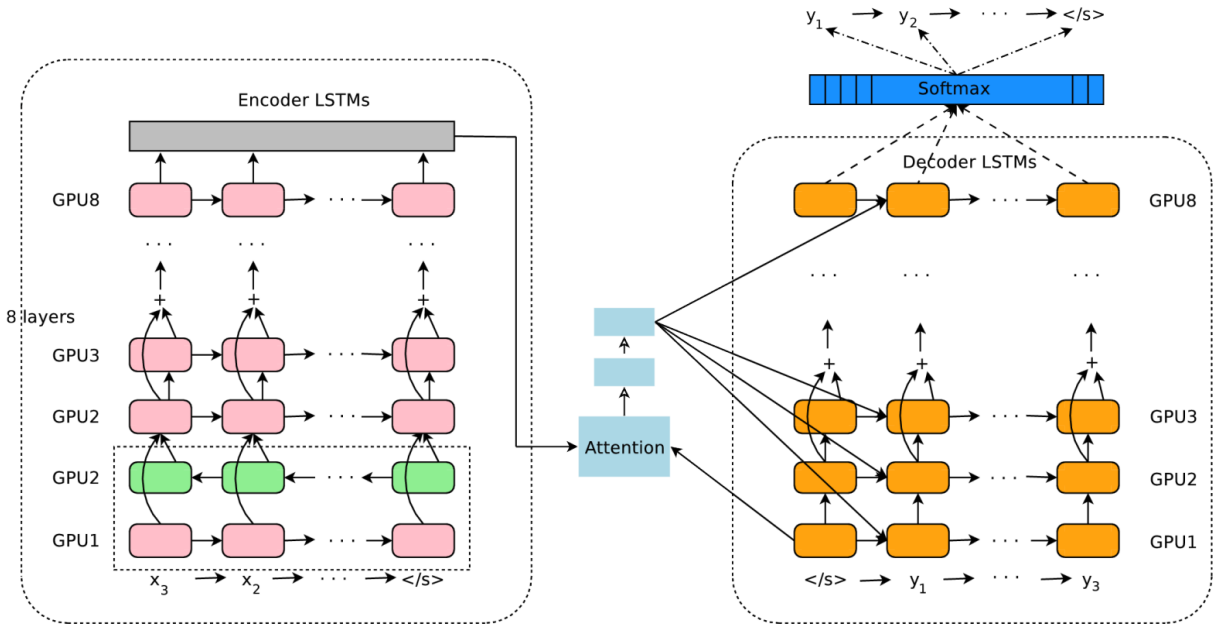
NOTES

SHARE YOUR ACCOMPLISHMENT!

Meets Specifications

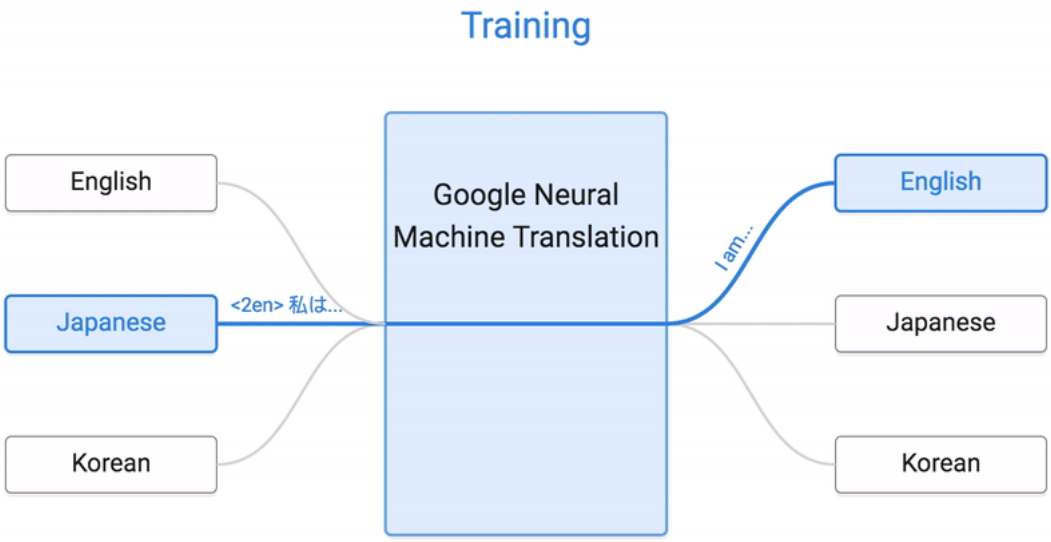
Congratulations You passed this Project. Best of Luck for next project.

Have you ever thought how google translator works?(please read following details)
Answer Very good question(appreciated). I would be giving answer based upon what google currently is using for their translator.



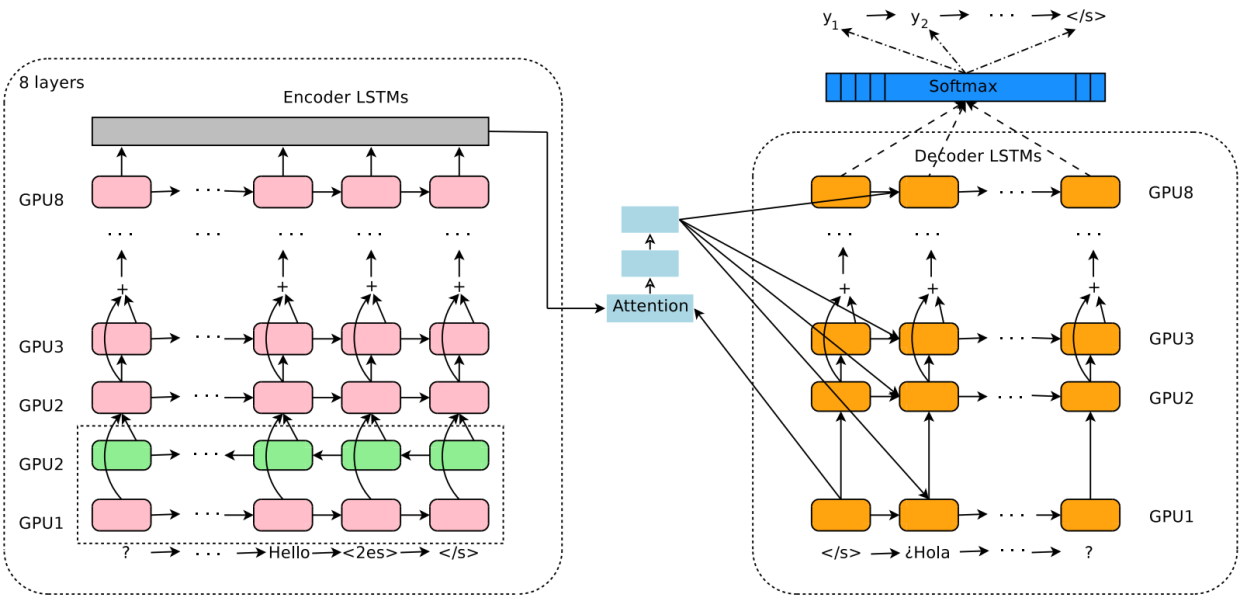
Lets first talk about GNMT model. The model architecture of GNMT, Google's Neural Machine Translation system. On the left is the encoder network, on the right is the decoder network, in the middle is the attention module. The
bottom encoder layer is bi-directional: the pink nodes gather information from left to right while the green nodes gather information from right to left. The other layers of the encoder are uni-directional. Residual connections start from the layer third from the bottom in the encoder and decoder. The model is partitioned into multiple GPUs to speed up training. In our setup, we have 8 encoder LSTM layers (1 bi-directional layer and 7 uni-directional layers), and 8 decoder layers.
Reference

But now to answer your question, google done a very smart thing. Google addressed this challenge by extending their previous GNMT system, allowing for a single system to translate between multiple languages. Their proposed architecture requires no change in the base GNMT system, but instead uses an additional "token" at the beginning of the input sentence to specify the required target language to translate to. In addition to improving translation quality, their method also enables "Zero-Shot Translation" — translation between language pairs never seen explicitly by the system.



Here's how it works. Let's say we train a multilingual system with Japanese⇄English and Korean⇄English examples, shown by the solid blue lines in the animation. Our multilingual system, with the same size as a single GNMT system, shares its parameters to translate between these four different language pairs. This sharing enables the system to transfer the "translation knowledge" from one language pair to the others. This transfer learning and the need to translate between multiple languages forces the system to better use its modeling power.

Now, Can we translate between a language pair which the system has never seen before? An example of this would be translations between Korean and Japanese where Korean⇄Japanese examples were not shown to the system. Impressively, the answer is yes — it can generate reasonable Korean⇄Japanese translations, even though it has never been taught to do so. We call this "zero-shot" translation, shown by the yellow dotted lines in the animation. To the best of our knowledge, this is the first time this type of transfer learning has worked in Machine Translation.
Reference



Now look at the change. The model architecture of the Multilingual GNMT system. In addition to what is described in GMNT, our input has an artificial token to indicate the required target language. In this example, the token "<2es>"
indicates that the target sentence is in Spanish, and the source sentence is reversed as a processing step.

Important Links for this Project:
Basic

1. http://sebastianruder.com/word-embeddings-1/
2. http://monik.in/a-noobs-guide-to-implementing-rnn-lstm-using-tensorflow/
3. http://suriyadeepan.github.io/2016-12-31-practical-seq2seq/
4. http://r2rt.com/recurrent-neural-networks-in-tensorflow-i.html
5. Understanding Truncation in RNN
6. https://indico.io/blog/sequence-modeling-neuralnets-part1/

Intermidiate

1. https://chunml.github.io/ChunML.github.io/project/Sequence-To-Sequence/
2. https://indico.io/blog/sequence-modeling-neural-networks-part2-attention-models/

Highly Involved

1. A good course: Deep Learning for Natural Language Processing!

Further reading material(After you understood all that is happening in this project.)

1. Use of attention for better translation. (http://stanford.edu/~lmthang/data/papers/emnlp15_attn.pdf)
2. If we want to implement more complex mechanism:
   Now, what if we want to implement more complex mechanic like when we want decoder to receive previously generated tokens as input at every timestamp (instead of lagged target sequence)? Or when we want to implement soft attention, where at every timestep we add additional fixed-len representation, derived from query produced by previous step's hidden state? tf.nn.raw_rnn is a way to solve this problem.
3. http://selfdrivingcars.mit.edu/
4. https://in.udacity.com/course/self-driving-car-engineer-nanodegree--nd013/

I am including this section so as to cover Topic in good depth. (I hope you will appericiate it, if not I will remove it after getting some feedback from you all)
FAQ (Frequently asked questions) (I am including answer to some questions, so as to cover this project in good detail and keeping in mind my previous experience as reviewer)

What the difference between an LSTM memory cell and an LSTM layer?
Answer: Link!

What a tf.nn.dynamic_rnn requires?
Answer: Remember that standard `tf.nn.dynamic_rnn` requires all inputs (t, ..., t+n) be passed in advance as a single tensor. "Dynamic" part of its name refers to the fact that n can change from batch to batch.

Difference between RNN and LSTM. Why to prefer LSTM?
Answer: All RNNs have feedback loops in the recurrent layer. This lets them maintain information in 'memory' over time. But, it can be difficult to train standard RNNs to solve problems that require learning long-term temporal dependencies. This is because the gradient of the loss function decays exponentially with time (called the vanishing gradient problem). LSTM networks are a type of RNN that uses special units in addition to standard units. LSTM units include a 'memory cell' that can maintain information in memory for long periods of time. A set of gates is used to control when information enters the memory, when it's output, and when it's forgotten. This architecture lets them learn longer-term dependencies. (Reference Link!)

READING MATERIAL

1. http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Why do we need Clipping of gradients?
Answer: As we know that LSTM solves our problem by learning long term dependencies according to activation function used can also create another problem. It is easy to imagine that, depending on our activation functions and network parameters, we could get exploding gradients instead of vanishing gradients if the values of the Jacobian matrix are large. Indeed, that's called the exploding gradient problem. The reason that vanishing gradients have received more attention than exploding gradients is two-fold. For one, exploding gradients are obvious. Your gradients will become NaN (not a number) and your program will crash. Secondly, clipping the gradients at a pre-defined threshold (as discussed in this paper!) is a very simple and effective solution to exploding gradients.

READING MATERIAL

1. https://cs224d.stanford.edu/lecture_notes/LectureNotes4.pdf
2. https://arxiv.org/pdf/1211.5063v2.pdf

Why do we have to do the mapping anyway?
Answer: Because it's better to input numeric training data into the Networks (as well as other learning algorithms). And we also need a different dictionary to convert the numbers back to the original characters. That's why we created the two dictionaries in previous project.

What is Word Embedding?
Answer: Word Embedding is a technique for learning dense representation of words in a low dimensional vector space. Each word can be seen as a point in this space, represented by a fixed length vector. Semantic relations between words are captured by this technique. The word vectors have some interesting properties.
Word Embedding is typically done in the first layer of the network : Embedding layer, that maps a word (index to word in vocabulary) from vocabulary to a dense vector of given size. In the seq2seq model, the weights of the embedding layer are jointly trained with the other parameters of the model. Follow this tutorial! by Sebastian Ruder to learn about different models used for word embedding and its importance in NLP.

Applications for seq2seq2 Model:

1. Language Translation (We did it in this project)
2. Making Chatbot

Required Files and Tests

The project submission contains the project notebook, called "dlnd_language_translation.ipynb".

All the unit tests in project have passed.

Great work. Unit testing is very good practice to ensure that your code is free from all bugs without getting confused and prevent you from wasting a lot of time while debugging minor things. Unit test also help in improving our code standards. For more details read this and I really hope that you will continue to use unit testing in every module that you write to keep it clean and speed up your development and for quality development. Unit testing is highly motivated in industries.

It is not always that if you passed unit test your code is okay, there can be some errors.

Preprocessing

The function `text_to_ids` is implemented correctly.

Good Work!! Correctly implemented.

Neural Network

The function `model_inputs` is implemented correctly.

Good Work!!

(Following abstract is from Tensorflow documentation)
TensorFlow programs use a tensor data structure to represent all data -- only tensors are passed between operations in the computation graph. You can think of a TensorFlow tensor as an n-dimensional array or list. A tensor has a static type, a rank, and a shape.

In the TensorFlow system, tensors are described by a unit of dimensionality known as rank. Tensor rank is not the same as matrix rank. Tensor rank (sometimes referred to as order or degree or n-dimension) is the number of dimensions of the tensor. For example, the following tensor (defined as a Python list) has a rank of 2:

```
t = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

A rank two tensor is what we typically think of as a matrix, a rank one tensor is a vector. For a rank two tensor you can access any element with the syntax t[i, j]. For a rank three tensor you would need to address an element with t[i, j, k].

Link! that might help you with better understanding of rank in Tensor.

The function `process_decoding_input` is implemented correctly.

Good Work!!
Removed the last word id from each batch in target_data and concatenated the GO ID to the beginning of each batch.

The function `encoding_layer` is implemented correctly.

Good Work!! Used dropout at correct place.

What we understand by encoding? I read this definition from internet by Adam Geitgey and found very easy to understand.
When you are trying to tell two faces apart with a computer, you collect different measurements from each face and use those measurements to compare faces. For example, we might measure the size of each ear or the spacing between the eyes and compare those measurements from two pictures to see if they are the same person. The idea of turning a face into a list of measurements is an example of an encoding. Similary we make same encodding for our sentences and our first network helps us to do so.

We discard encoder_outputs because we are not interested in them within seq2seq framework. What we actually want is encoder_final_state — state of LSTM's hidden cells at the last moment of the Encoder rollout.
encoder_final_state is also called "thought vector". We will use it as initial state for the Decoder. In seq2seq without attention this is the only point where Encoder passes information to Decoder. We hope that backpropagation through time (BPTT) algorithm will tune the model to pass enough information throught the thought vector for correct sequence output decoding.

AutoEncoders Link: https://www.youtube.com/watch?v=FzS3tMl4Nsc

The function `decoding_layer_train` is implemented correctly.

Good Work!! Correctly implemented.

The function `decoding_layer_infer` is implemented correctly.

Good Work!! Correctly implemented.

The function `decoding_layer` is implemented correctly.

Good Work!! Correctly implemented

How decoder works, Intuitively?
Answer In the decoder step, a language model is trained on both the output sequence (such as the translated sentence) as well as the fixed representation from the encoder. Since the decoder model sees an encoded representation of the input sequence as well as the translation sequence, it can make more intelligent predictions about future words based on the current word. For example, in a standard language model, we might see the word “crane” and not be sure if the next word should be about the bird or heavy machinery. However, if we also pass an encoder context, the decoder might realize that the input sequence was about construction, not flying animals. Given the context, the decoder can choose the appropriate next word and provide more accurate translations.

Reference

The function `seq2seq_model` is implemented correctly.

Good Work!!
Correctly stacked all the lego blocks.

Neural Network Training

The parameters are set to reasonable numbers.

Good choice of Hyperparameters but as you know there is always scope of improvement.

Some Suggestion

- Chosen `batch_size` is good. Try to observe what happens if you lower it or increase it. We know that larger batch sizes might speed up the training but can degrade the quality of the model at the same time. This link might help you - http://stats.stackexchange.com/questions/164876/tradeoff-batch-size-vs-number-of-iterations-to-train-a-neural-network
- Chosen `rnn_size` is small, it would be hard for network to learn something good. First try increasing and see if your model converges better and has a lower training loss or not. Consider changing according to your system configuration. (Please read tips also)
- The embedding size chosen by you is on lower side. Please try to increase them and see how your model responds.

Some Tips on how to choose Hyper Parameters

- **rnn_size** : For each LSTM cell that we initialise, we need to supply a value for the hidden dimension(`rnn_size`), or as some people like to call it, the number of units in the LSTM cell. The value of it is it up to you, too high a value may lead to overfitting or a very low value may yield extremely poor results.
- **Learning Rate** : Learning rate is the most important hyperparameter, therefore
  it is very important to understand how to set it correctly in order to achieve good performance. A related but unsurprising observation is that there is a sweet-spot for the learning rate at the high end of the basin. In this region, the performance is good and the training time is small. So while searching for a good learning rate for the LSTM, it is sufficient to do a coarse search by starting with a high value (e.g. 1.0) and dividing it by ten until performance stops increasing.
- **Hidden Layers** : As expected, larger networks perform better, and the required
  training time increases with the network size.
- **Embedding size** : I've actually found smaller vectors to work better (64 and 128 - note the binary sizes, I think this helps performance by assisting theano with copying chunks of data to and from GPU), likely as that's fewer parameters to learn, and too many can make learning hard for a neural network. Cross validation would help you determine a good size, try varying in magnitude (32,64,128,256) rather than more linear scales, as the relationship between these items and performance tend to be more of an exponential than a linear nature.

Reference Link!

Link that would help in fine tune model.
http://neupy.com/2016/12/17/hyperparameter_optimization_for_neural_networks.html

The project should end with a validation and test accuracy that is at least 90.00%

Language Translation

The function `sentence_to_seq` is implemented correctly.

Good Work!! You took care of converting sentence into lower case.

The project gets majority of the translation correctly. The translation doesn't have to be perfect.

Good Work!!

DOWNLOAD PROJECT

RETURN TO PATH

Student FAQ