Language Translation

Get the Data

Since translating the whole language of English to French will take lots of time to train, we have provided you with a small portion of the English corpus.

In [1]:

```
import helper
import problem_unittests as tests

source_path = 'data/small_vocab_en'
target_path = 'data/small_vocab_fr'
source_text = helper.load_data(source_path)
target_text = helper.load_data(target_path)
```

```
C:\Users\santhu\Anaconda3\lib\site-packages\h5py\__init__.py:36: FutureWarni
ng: Conversion of the second argument of issubdtype from `float` to `np.floa
ting` is deprecated. In future, it will be treated as `np.float64 == np.dtyp
e(float).type`.
  from ._conv import register_converters as _register_converters
```

Explore the Data

Play around with view_sentence_range to view different parts of the data.

In [2]:

```
view_sentence_range = (10, 20)

"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
import numpy as np

print('Dataset Stats')
print('Roughly the number of unique words: {}'.format(len({word: None for word in source_te

sentences = source_text.split('\n')
word_counts = [len(sentence.split()) for sentence in sentences]
print('Number of sentences: {}'.format(len(sentences)))
print('Average number of words in a sentence: {}'.format(np.average(word_counts)))

print()
print('English sentences {} to {}:'.format(*view_sentence_range))
print('\n'.join(source_text.split('\n')[view_sentence_range[0]:view_sentence_range[1]]))
print()
print('French sentences {} to {}:'.format(*view_sentence_range))
print('\n'.join(target_text.split('\n')[view_sentence_range[0]:view_sentence_range[1]]))
```

```
Dataset Stats
Roughly the number of unique words: 227
Number of sentences: 137861
Average number of words in a sentence: 13.225277634719028

English sentences 10 to 20:
the lime is her least liked fruit , but the banana is my least liked .
he saw a old yellow truck .
india is rainy during june , and it is sometimes warm in november .
that cat was my most loved animal .
he dislikes grapefruit , limes , and lemons .
her least liked fruit is the lemon , but his least liked is the grapefruit .
california is never cold during february , but it is sometimes freezing in j
une .
china is usually pleasant during autumn , and it is usually quiet in october
.
paris is never freezing during november , but it is wonderful in october .
the united states is never rainy during january , but it is sometimes mild i
n october .

French sentences 10 to 20:
la chaux est son moins aimé des fruits , mais la banane est mon moins aimé.
il a vu un vieux camion jaune .
inde est pluvieux en juin , et il est parfois chaud en novembre .
ce chat était mon animal le plus aimé .
il n'aime pamplemousse , citrons verts et les citrons .
son fruit est moins aimé le citron , mais son moins aimé est le pamplemousse
.
californie ne fait jamais froid en février , mais il est parfois le gel en j
uin .
chine est généralement agréable en automne , et il est généralement calme en
octobre .
paris est jamais le gel en novembre , mais il est merveilleux en octobre .
les états-unis est jamais pluvieux en janvier , mais il est parfois doux en
octobre .
```

Implement Preprocessing Function

Text to Word Ids

As you did with other RNNs, you must turn the text into a number so the computer can understand it. In the function `text_to_ids()`, you'll turn `source_text` and `target_text` from words to ids. However, you need to add the `<EOS>` word id at the end of `target_text`. This will help the neural network predict when the sentence should end.

You can get the `<EOS>` word id by doing:

```
target_vocab_to_int['<EOS>']
```

You can get other word ids using `source_vocab_to_int` and `target_vocab_to_int`.

In [3]:

```
def text_to_ids(source_text, target_text, source_vocab_to_int, target_vocab_to_int):
    """
    Convert source and target text to proper word ids
    :param source_text: String that contains all the source text.
    :param target_text: String that contains all the target text.
    :param source_vocab_to_int: Dictionary to go from the source words to an id
    :param target_vocab_to_int: Dictionary to go from the target words to an id
    :return: A tuple of lists (source_id_text, target_id_text)
    """
    # Just go through the text and transform it.
    source_id_text = []
    for idx, line in enumerate(source_text.split('\n')):
        source_id_text.append([])
        for word in line.split():
            source_id_text[idx].append(source_vocab_to_int[word])

    target_id_text = []
    for idx, line in enumerate(target_text.split('\n')):
        target_id_text.append([])
        for word in line.split():
            target_id_text[idx].append(target_vocab_to_int[word])
        target_id_text[idx].append(target_vocab_to_int['<EOS>'])

    return (source_id_text, target_id_text)

tests.test_text_to_ids(text_to_ids)
```

```
Tests Passed
```

Preprocess all the data and save it

Running the code cell below will preprocess all the data and save it to file.

In [4]:

```
helper.preprocess_and_save_data(source_path, target_path, text_to_ids)
```

Check Point

This is your first checkpoint. If you ever decide to come back to this notebook or have to restart the notebook, you can start from here. The preprocessed data has been saved to disk.

In [5]:

```
import numpy as np
import helper
import problem_unittests as tests

(source_int_text, target_int_text), (source_vocab_to_int, target_vocab_to_int), _ = helper.
```

Check the Version of TensorFlow and Access to GPU

This will check to make sure you have the correct version of TensorFlow and access to a GPU

In [6]:

```
from distutils.version import LooseVersion
import warnings
import tensorflow as tf
from tensorflow.python.layers.core import Dense

# Check TensorFlow Version
assert LooseVersion(tf.__version__) >= LooseVersion('1.1'), 'Please use TensorFlow version
print('TensorFlow Version: {}'.format(tf.__version__))

# Check for a GPU
if not tf.test.gpu_device_name():
    warnings.warn('No GPU found. Please use a GPU to train your neural network.')
else:
    print('Default GPU Device: {}'.format(tf.test.gpu_device_name()))
```

```
TensorFlow Version: 1.12.0

C:\Users\santhu\Anaconda3\lib\site-packages\ipykernel_launcher.py:15: UserWa
rning: No GPU found. Please use a GPU to train your neural network.
  from ipykernel import kernelapp as app
```

Build the Neural Network

You'll build the components necessary to build a Sequence-to-Sequence model by implementing the following functions below:

- `model_inputs`
- `process_decoder_input`
- `encoding_layer`
- `decoding_layer_train`
- `decoding_layer_infer`
- `decoding_layer`
- `seq2seq_model`

Input

Implement the `model_inputs()` function to create TF Placeholders for the Neural Network. It should create the following placeholders:

- Input text placeholder named "input" using the TF Placeholder name parameter with rank 2.
- Targets placeholder with rank 2.
- Learning rate placeholder with rank 0.
- Keep probability placeholder named "keep_prob" using the TF Placeholder name parameter with rank 0.

- Target sequence length placeholder named "target_sequence_length" with rank 1
- Max target sequence length tensor named "max_target_len" getting its value from applying tf.reduce_max on the target_sequence_length placeholder. Rank 0.
- Source sequence length placeholder named "source_sequence_length" with rank 1

Return the placeholders in the following the tuple (input, targets, learning rate, keep probability, target sequence length, max target sequence length, source sequence length)

In [7]:

```
def model_inputs():
    """
    Create TF Placeholders for input, targets, learning rate, and lengths of source and tar
    :return: Tuple (input, targets, learning rate, keep probability, target sequence length
    max target sequence length, source sequence length)
    """
    # TODO: Implement Function
    inputs = tf.placeholder(tf.int32, shape=[None,None], name= "input")
    targets = tf.placeholder(tf.int32, shape=[None,None], name= "targets")
    lrate = tf.placeholder(tf.float32, name= "learning_rate")
    keep_prob = tf.placeholder(tf.float32, name= "keep_prob")
    target_seq_lenth = tf.placeholder(tf.int32, shape=[None], name= "target_sequence_length
    max_target_len = tf.reduce_max(target_seq_lenth, name= 'max_target_len')
    source_seq_length = tf.placeholder(tf.int32, shape=[None], name= "source_sequence_lengt
    return (inputs, targets, lrate, keep_prob, target_seq_lenth, max_target_len, source_sec


tests.test_model_inputs(model_inputs)
```

Tests Passed

Process Decoder Input

Implement `process_decoder_input` by removing the last word id from each batch in `target_data` and concat the GO ID to the begining of each batch.

In [8]:

```
def process_decoder_input(target_data, target_vocab_to_int, batch_size):
    """
    Preprocess target data for encoding
    :param target_data: Target Placehoder
    :param target_vocab_to_int: Dictionary to go from the target words to an id
    :param batch_size: Batch Size
    :return: Preprocessed target data
    """
    # Create a constant tensor with the 'go id'.
    go_id = tf.constant(target_vocab_to_int['<GO>'], shape=(batch_size,1), dtype=tf.int32)
    # Concatenate the vector without the last word id with the go ids vector
    processed_input = tf.concat([go_id,target_data[:,:-1]],1)
    return processed_input

tests.test_process_encoding_input(process_decoder_input)
```

Tests Passed

Encoding

Implement `encoding_layer()` to create a Encoder RNN layer:
 * Embed the encoder input using [`tf.contrib.layers.embed_sequence`](https://www.tensorflow.org/api_docs/python/tf/contrib/layers/embed_sequence)

```
 * Construct a [stacked]
(https://github.com/tensorflow/tensorflow/blob/6947f65a374ebf29e74bb71e36fd82760056d82c/te
nsorflow/docs_src/tutorials/recurrent.md#stacking-multiple-lstms)
[`tf.contrib.rnn.LSTMCell`]
(https://www.tensorflow.org/api_docs/python/tf/contrib/rnn/LSTMCell) wrapped in a
[`tf.contrib.rnn.DropoutWrapper`]
(https://www.tensorflow.org/api_docs/python/tf/contrib/rnn/DropoutWrapper)
 * Pass cell and embedded input to [`tf.nn.dynamic_rnn()`]
(https://www.tensorflow.org/api_docs/python/tf/nn/dynamic_rnn)
```

In [9]:

```
from imp import reload
reload(tests)

def encoding_layer(rnn_inputs, rnn_size, num_layers, keep_prob,
                   source_sequence_length, source_vocab_size,
                   encoding_embedding_size):
    """
    Create encoding layer
    :param rnn_inputs: Inputs for the RNN
    :param rnn_size: RNN Size
    :param num_layers: Number of layers
    :param keep_prob: Dropout keep probability
    :param source_sequence_length: a list of the lengths of each sequence in the batch
    :param source_vocab_size: vocabulary size of source data
    :param encoding_embedding_size: embedding size of source data
    :return: tuple (RNN output, RNN state)
    """
    # Build the lstm cells wrapped in dropout
    def build_cell(rnn_size, keep_prob):
        lstm = tf.contrib.rnn.LSTMCell(rnn_size)
        lstm_drop = tf.contrib.rnn.DropoutWrapper(lstm, output_keep_prob=keep_prob)
        return lstm_drop
    # Stack them all
    stacked_lstm = tf.contrib.rnn.MultiRNNCell([build_cell(rnn_size, keep_prob) for _ in ra
    # Creathe embedding layer.
    embed_encoder = tf.contrib.layers.embed_sequence(rnn_inputs, vocab_size = source_vocab_
    # If we don't have an initial zero state, provide a dtype.
    output, state = tf.nn.dynamic_rnn(stacked_lstm, embed_encoder, source_sequence_length,
    return (output, state)

tests.test_encoding_layer(encoding_layer)
```

```
ERROR:tensorflow:==================================
Object was never used (type <class 'tensorflow.python.framework.ops.Operatio
n'>):
<tf.Operation 'assert_rank_2/Assert/Assert' type=Assert>
If you want to mark it as used call its "mark_used()" method.
It was originally created here:
  File "C:\Users\santhu\Anaconda3\lib\runpy.py", line 193, in _run_module_as
_main
    "__main__", mod_spec)  File "C:\Users\santhu\Anaconda3\lib\runpy.py", li
ne 85, in _run_code
    exec(code, run_globals)  File "C:\Users\santhu\Anaconda3\lib\site-packag
es\ipykernel_launcher.py", line 16, in <module>
    app.launch_new_instance()  File "C:\Users\santhu\Anaconda3\lib\site-pack
ages\traitlets\config\application.py", line 658, in launch_instance
    app.start()  File "C:\Users\santhu\Anaconda3\lib\site-packages\ipykernel
\kernelapp.py", line 486, in start
    self.io_loop.start()  File "C:\Users\santhu\Anaconda3\lib\site-packages
\tornado\platform\asyncio.py", line 127, in start
    self.asyncio_loop.run_forever()  File "C:\Users\santhu\Anaconda3\lib\asy
ncio\base_events.py", line 422, in run_forever
    self._run_once()  File "C:\Users\santhu\Anaconda3\lib\asyncio\base_event
s.py", line 1433, in _run_once
    handle = None  # Needed to break cycles when an exception occurs.  File
"C:\Users\santhu\Anaconda3\lib\asyncio\events.py", line 157, in _run
    self = None  # Needed to break cycles when an exception occurs.  File
"C:\Users\santhu\Anaconda3\lib\site-packages\tornado\ioloop.py", line 776, i
```

```
n _run_callback
    self.handle_callback_exception(callback)  File "C:\Users\santhu\Anaconda
3\lib\site-packages\tornado\stack_context.py", line 278, in null_wrapper
    _state.contexts = current_state  File "C:\Users\santhu\Anaconda3\lib\sit
e-packages\zmq\eventloop\zmqstream.py", line 536, in <lambda>
    self.io_loop.add_callback(lambda : self._handle_events(self.socket, 0))
File "C:\Users\santhu\Anaconda3\lib\site-packages\zmq\eventloop\zmqstream.p
y", line 463, in _handle_events
    raise  File "C:\Users\santhu\Anaconda3\lib\site-packages\zmq\eventloop\z
mqstream.py", line 480, in _handle_recv
    self._run_callback(callback, msg)  File "C:\Users\santhu\Anaconda3\lib\s
ite-packages\zmq\eventloop\zmqstream.py", line 438, in _run_callback
    raise  File "C:\Users\santhu\Anaconda3\lib\site-packages\tornado\stack_c
ontext.py", line 278, in null_wrapper
    _state.contexts = current_state  File "C:\Users\santhu\Anaconda3\lib\sit
e-packages\ipykernel\kernelbase.py", line 283, in dispatcher
    return self.dispatch_shell(stream, msg)  File "C:\Users\santhu\Anaconda3
\lib\site-packages\ipykernel\kernelbase.py", line 241, in dispatch_shell
    self._publish_status(u'idle')  File "C:\Users\santhu\Anaconda3\lib\site-
packages\ipykernel\kernelbase.py", line 421, in execute_request
    self._abort_queues()  File "C:\Users\santhu\Anaconda3\lib\site-packages
\ipykernel\ipkernel.py", line 258, in do_execute
    return reply_content  File "C:\Users\santhu\Anaconda3\lib\site-packages
\ipykernel\zmqshell.py", line 537, in run_cell
    return super(ZMQInteractiveShell, self).run_cell(*args, **kwargs)  File
"C:\Users\santhu\Anaconda3\lib\site-packages\IPython\core\interactiveshell.p
y", line 2667, in run_cell
    return result  File "C:\Users\santhu\Anaconda3\lib\site-packages\IPython
\core\interactiveshell.py", line 2801, in _run_cell
    return result  File "C:\Users\santhu\Anaconda3\lib\site-packages\IPython
\core\interactiveshell.py", line 2931, in run_ast_nodes
    return False  File "C:\Users\santhu\Anaconda3\lib\site-packages\IPython
\core\interactiveshell.py", line 2983, in run_code
    return outflag  File "<ipython-input-7-8e62e961e9c8>", line 21, in <modu
le>
    tests.test_model_inputs(model_inputs)  File "C:\Users\santhu\Desktop\New
folder\problem_unittests.py", line 112, in test_model_inputs
    _print_success_message()  File "C:\Users\santhu\Anaconda3\lib\site-packa
ges\tensorflow\python\ops\check_ops.py", line 827, in assert_rank
    return assert_op  File "C:\Users\santhu\Anaconda3\lib\site-packages\tens
orflow\python\ops\check_ops.py", line 765, in _assert_rank_condition
    return control_flow_ops.Assert(condition, data, summarize=summarize)  Fi
le "C:\Users\santhu\Anaconda3\lib\site-packages\tensorflow\python\util\tf_sh
ould_use.py", line 189, in wrapped
    return _add_should_use_warning(fn(*args, **kwargs))
================================
ERROR:tensorflow:================================
Object was never used (type <class 'tensorflow.python.framework.ops.Operatio
n'>):
<tf.Operation 'assert_rank_3/Assert/Assert' type=Assert>
If you want to mark it as used call its "mark_used()" method.
It was originally created here:
  File "C:\Users\santhu\Anaconda3\lib\runpy.py", line 193, in _run_module_as
_main
    "__main__", mod_spec)  File "C:\Users\santhu\Anaconda3\lib\runpy.py", li
ne 85, in _run_code
    exec(code, run_globals)  File "C:\Users\santhu\Anaconda3\lib\site-packag
es\ipykernel_launcher.py", line 16, in <module>
    app.launch_new_instance()  File "C:\Users\santhu\Anaconda3\lib\site-pack
ages\traitlets\config\application.py", line 658, in launch_instance
    app.start()  File "C:\Users\santhu\Anaconda3\lib\site-packages\ipykernel
```

```
\kernelapp.py", line 486, in start
    self.io_loop.start()  File "C:\Users\santhu\Anaconda3\lib\site-packages
\tornado\platform\asyncio.py", line 127, in start
    self.asyncio_loop.run_forever()  File "C:\Users\santhu\Anaconda3\lib\asy
ncio\base_events.py", line 422, in run_forever
    self._run_once()  File "C:\Users\santhu\Anaconda3\lib\asyncio\base_event
s.py", line 1433, in _run_once
    handle = None  # Needed to break cycles when an exception occurs.  File
"C:\Users\santhu\Anaconda3\lib\asyncio\events.py", line 157, in _run
    self = None  # Needed to break cycles when an exception occurs.  File
"C:\Users\santhu\Anaconda3\lib\site-packages\tornado\ioloop.py", line 776, i
n _run_callback
    self.handle_callback_exception(callback)  File "C:\Users\santhu\Anaconda
3\lib\site-packages\tornado\stack_context.py", line 278, in null_wrapper
    _state.contexts = current_state  File "C:\Users\santhu\Anaconda3\lib\sit
e-packages\zmq\eventloop\zmqstream.py", line 536, in <lambda>
    self.io_loop.add_callback(lambda : self._handle_events(self.socket, 0))
File "C:\Users\santhu\Anaconda3\lib\site-packages\zmq\eventloop\zmqstream.p
y", line 463, in _handle_events
    raise  File "C:\Users\santhu\Anaconda3\lib\site-packages\zmq\eventloop\z
mqstream.py", line 480, in _handle_recv
    self._run_callback(callback, msg)  File "C:\Users\santhu\Anaconda3\lib\s
ite-packages\zmq\eventloop\zmqstream.py", line 438, in _run_callback
    raise  File "C:\Users\santhu\Anaconda3\lib\site-packages\tornado\stack_c
ontext.py", line 278, in null_wrapper
    _state.contexts = current_state  File "C:\Users\santhu\Anaconda3\lib\sit
e-packages\ipykernel\kernelbase.py", line 283, in dispatcher
    return self.dispatch_shell(stream, msg)  File "C:\Users\santhu\Anaconda3
\lib\site-packages\ipykernel\kernelbase.py", line 241, in dispatch_shell
    self._publish_status(u'idle')  File "C:\Users\santhu\Anaconda3\lib\site-
packages\ipykernel\kernelbase.py", line 421, in execute_request
    self._abort_queues()  File "C:\Users\santhu\Anaconda3\lib\site-packages
\ipykernel\ipkernel.py", line 258, in do_execute
    return reply_content  File "C:\Users\santhu\Anaconda3\lib\site-packages
\ipykernel\zmqshell.py", line 537, in run_cell
    return super(ZMQInteractiveShell, self).run_cell(*args, **kwargs)  File
"C:\Users\santhu\Anaconda3\lib\site-packages\IPython\core\interactiveshell.p
y", line 2667, in run_cell
    return result  File "C:\Users\santhu\Anaconda3\lib\site-packages\IPython
\core\interactiveshell.py", line 2801, in _run_cell
    return result  File "C:\Users\santhu\Anaconda3\lib\site-packages\IPython
\core\interactiveshell.py", line 2931, in run_ast_nodes
    return False  File "C:\Users\santhu\Anaconda3\lib\site-packages\IPython
\core\interactiveshell.py", line 2983, in run_code
    return outflag  File "<ipython-input-7-8e62e961e9c8>", line 21, in <modu
le>
    tests.test_model_inputs(model_inputs)  File "C:\Users\santhu\Desktop\New
folder\problem_unittests.py", line 112, in test_model_inputs
    _print_success_message()  File "C:\Users\santhu\Anaconda3\lib\site-packa
ges\tensorflow\python\ops\check_ops.py", line 827, in assert_rank
    return assert_op  File "C:\Users\santhu\Anaconda3\lib\site-packages\tens
orflow\python\ops\check_ops.py", line 765, in _assert_rank_condition
    return control_flow_ops.Assert(condition, data, summarize=summarize)  Fi
le "C:\Users\santhu\Anaconda3\lib\site-packages\tensorflow\python\util\tf_sh
ould_use.py", line 189, in wrapped
    return _add_should_use_warning(fn(*args, **kwargs))
==================================

Tests Passed
```

Decoding - Training

Create a training decoding layer:
* Create a [`tf.contrib.seq2seq.TrainingHelper`]
(https://www.tensorflow.org/api_docs/python/tf/contrib/seq2seq/TrainingHelper)
* Create a [`tf.contrib.seq2seq.BasicDecoder`]
(https://www.tensorflow.org/api_docs/python/tf/contrib/seq2seq/BasicDecoder)
* Obtain the decoder outputs from [`tf.contrib.seq2seq.dynamic_decode`]
(https://www.tensorflow.org/api_docs/python/tf/contrib/seq2seq/dynamic_decode)

In [10]:

```
def decoding_layer_train(encoder_state, dec_cell, dec_embed_input,
                         target_sequence_length, max_summary_length,
                         output_layer, keep_prob):
    """
    Create a decoding layer for training
    :param encoder_state: Encoder State
    :param dec_cell: Decoder RNN Cell
    :param dec_embed_input: Decoder embedded input
    :param target_sequence_length: The lengths of each sequence in the target batch
    :param max_summary_length: The length of the longest sequence in the batch
    :param output_layer: Function to apply the output layer
    :param keep_prob: Dropout keep probability
    :return: BasicDecoderOutput containing training logits and sample_id
    """
    # TODO: Implement Function
    trainig_helper = tf.contrib.seq2seq.TrainingHelper(dec_embed_input, target_sequence_ler
    basic_decoder = tf.contrib.seq2seq.BasicDecoder(dec_cell, trainig_helper, encoder_state
    f_output, _, _ = tf.contrib.seq2seq.dynamic_decode(basic_decoder,maximum_iterations=max
    return f_output



tests.test_decoding_layer_train(decoding_layer_train)
```

Tests Passed

Decoding - Inference

Create inference decoder:
* Create a [`tf.contrib.seq2seq.GreedyEmbeddingHelper`]
(https://www.tensorflow.org/api_docs/python/tf/contrib/seq2seq/GreedyEmbeddingHelper)
* Create a [`tf.contrib.seq2seq.BasicDecoder`]
(https://www.tensorflow.org/api_docs/python/tf/contrib/seq2seq/BasicDecoder)
* Obtain the decoder outputs from [`tf.contrib.seq2seq.dynamic_decode`]
(https://www.tensorflow.org/api_docs/python/tf/contrib/seq2seq/dynamic_decode)

In [11]:

```
def decoding_layer_infer(encoder_state, dec_cell, dec_embeddings, start_of_sequence_id,
                         end_of_sequence_id, max_target_sequence_length,
                         vocab_size, output_layer, batch_size, keep_prob):
    """
    Create a decoding layer for inference
    :param encoder_state: Encoder state
    :param dec_cell: Decoder RNN Cell
    :param dec_embeddings: Decoder embeddings
    :param start_of_sequence_id: GO ID
    :param end_of_sequence_id: EOS Id
    :param max_target_sequence_length: Maximum length of target sequences
    :param vocab_size: Size of decoder/target vocabulary
    :param decoding_scope: TenorFlow Variable Scope for decoding
    :param output_layer: Function to apply the output layer
    :param batch_size: Batch size
    :param keep_prob: Dropout keep probability
    :return: BasicDecoderOutput containing inference logits and sample_id
    """
    # Convert the start_ids to be a vector with batch size (the go id repeated batch size t
    start_ids = tf.tile([start_of_sequence_id], [batch_size])
    # Create the embedding helper.
    embedding_helper = tf.contrib.seq2seq.GreedyEmbeddingHelper(
        dec_embeddings, start_ids, end_of_sequence_id)
    basic_decoder = tf.contrib.seq2seq.BasicDecoder(
        dec_cell, embedding_helper, encoder_state, output_layer)
    f_output, _, _ = tf.contrib.seq2seq.dynamic_decode(
        basic_decoder,maximum_iterations=max_target_sequence_length)
    return f_output



tests.test_decoding_layer_infer(decoding_layer_infer)
```

```
Tests Passed
```

Build the Decoding Layer

Implement `decoding_layer()` to create a Decoder RNN layer.

* Embed the target sequences
* Construct the decoder LSTM cell (just like you constructed the encoder cell above)
* Create an output layer to map the outputs of the decoder to the elements of our vocabulary
* Use the your `decoding_layer_train(encoder_state, dec_cell, dec_embed_input, target_sequence_length, max_target_sequence_length, output_layer, keep_prob)` function to get the training logits.
* Use your `decoding_layer_infer(encoder_state, dec_cell, dec_embeddings, start_of_sequence_id, end_of_sequence_id, max_target_sequence_length, vocab_size, output_layer, batch_size, keep_prob)` function to get the inference logits.

Note: You'll need to use [tf.variable_scope](https://www.tensorflow.org/api_docs/python/tf/variable_scope) to share variables between training and inference.

In [12]:

```
def decoding_layer(dec_input, encoder_state,
                   target_sequence_length, max_target_sequence_length,
                   rnn_size,
                   num_layers, target_vocab_to_int, target_vocab_size,
                   batch_size, keep_prob, decoding_embedding_size):
    """
    Create decoding layer
    :param dec_input: Decoder input
    :param encoder_state: Encoder state
    :param target_sequence_length: The lengths of each sequence in the target batch
    :param max_target_sequence_length: Maximum length of target sequences
    :param rnn_size: RNN Size
    :param num_layers: Number of layers
    :param target_vocab_to_int: Dictionary to go from the target words to an id
    :param target_vocab_size: Size of target vocabulary
    :param batch_size: The size of the batch
    :param keep_prob: Dropout keep probability
    :param decoding_embedding_size: Decoding embedding size
    :return: Tuple of (Training BasicDecoderOutput, Inference BasicDecoderOutput)
    """
    # Use the same proess as in the encoding layer.
    def build_cell(rnn_size, keep_prob):
        lstm = tf.contrib.rnn.LSTMCell(rnn_size)
        lstm_drop = tf.contrib.rnn.DropoutWrapper(lstm, output_keep_prob=keep_prob)
        return lstm_drop
    # Stack them all
    stacked_lstm = tf.contrib.rnn.MultiRNNCell([build_cell(rnn_size, keep_prob) for _ in ra

    dec_embeddings = tf.Variable(tf.random_uniform([target_vocab_size, decoding_embedding_s
    dec_embed_input = tf.nn.embedding_lookup(dec_embeddings, dec_input)

    dense_layer = Dense(target_vocab_size,
                        kernel_initializer = tf.truncated_normal_initializer(mean = 0.0, s

    with tf.variable_scope("decode") as scope:
        tr_decoder_output = decoding_layer_train(
            encoder_state, stacked_lstm, dec_embed_input,
            target_sequence_length, max_target_sequence_length,
            dense_layer, keep_prob)
        scope.reuse_variables()
        inf_decoder_output = decoding_layer_infer(
            encoder_state, stacked_lstm, dec_embeddings,
            target_vocab_to_int['<GO>'], target_vocab_to_int['<EOS>'],
            max_target_sequence_length, target_vocab_size,
            dense_layer, batch_size, keep_prob)

    return tr_decoder_output, inf_decoder_output

tests.test_decoding_layer(decoding_layer)
```

```
Tests Passed
```

Build the Neural Network

Apply the functions you implemented above to:

- Encode the input using your `encoding_layer(rnn_inputs, rnn_size, num_layers, keep_prob, source_sequence_length, source_vocab_size, encoding_embedding_size)`.

- Process target data using your `process_decoder_input(target_data, target_vocab_to_int, batch_size)` function.
- Decode the encoded input using your `decoding_layer(dec_input, enc_state, target_sequence_length, max_target_sentence_length, rnn_size, num_layers, target_vocab_to_int, target_vocab_size, batch_size, keep_prob, dec_embedding_size)` function.

In [13]:

```
def seq2seq_model(input_data, target_data, keep_prob, batch_size,
                  source_sequence_length, target_sequence_length,
                  max_target_sentence_length,
                  source_vocab_size, target_vocab_size,
                  enc_embedding_size, dec_embedding_size,
                  rnn_size, num_layers, target_vocab_to_int):
    """
    Build the Sequence-to-Sequence part of the neural network
    :param input_data: Input placeholder
    :param target_data: Target placeholder
    :param keep_prob: Dropout keep probability placeholder
    :param batch_size: Batch Size
    :param source_sequence_length: Sequence Lengths of source sequences in the batch
    :param target_sequence_length: Sequence Lengths of target sequences in the batch
    :param source_vocab_size: Source vocabulary size
    :param target_vocab_size: Target vocabulary size
    :param enc_embedding_size: Decoder embedding size
    :param dec_embedding_size: Encoder embedding size
    :param rnn_size: RNN Size
    :param num_layers: Number of layers
    :param target_vocab_to_int: Dictionary to go from the target words to an id
    :return: Tuple of (Training BasicDecoderOutput, Inference BasicDecoderOutput)
    """
    output, state = encoding_layer(input_data, rnn_size, num_layers, keep_prob,
                   source_sequence_length, source_vocab_size,
                   enc_embedding_size)

    processed_input = process_decoder_input(target_data, target_vocab_to_int, batch_size)

    tr_decoder_output, inf_decoder_output = decoding_layer(processed_input, state,
                   target_sequence_length, max_target_sentence_length,
                   rnn_size, num_layers, target_vocab_to_int, target_vocab_size,
                   batch_size, keep_prob, dec_embedding_size)

    return tr_decoder_output, inf_decoder_output


tests.test_seq2seq_model(seq2seq_model)
```

```
Tests Passed
```

Neural Network Training
Hyperparameters

Tune the following parameters:

- Set `epochs` to the number of epochs.
- Set `batch_size` to the batch size.
- Set `rnn_size` to the size of the RNNs.
- Set `num_layers` to the number of layers.
- Set `encoding_embedding_size` to the size of the embedding for the encoder.
- Set `decoding_embedding_size` to the size of the embedding for the decoder.
- Set `learning_rate` to the learning rate.

- Set `keep_probability` to the Dropout keep probability
- Set `display_step` to state how many steps between each debug output statement

In [14]:

```
# Number of Epochs
epochs = 10
# Batch Size
batch_size = 512
# RNN Size
rnn_size = 128
# Number of Layers
num_layers = 2
# Embedding Size
encoding_embedding_size = 128
decoding_embedding_size = 128
# Learning Rate
learning_rate = 0.001
# Dropout Keep Probability
keep_probability = 0.55
display_step = True
```

Build the Graph

Build the graph using the neural network you implemented.

In [15]:

```
save_path = 'checkpoints/dev'
(source_int_text, target_int_text), (source_vocab_to_int, target_vocab_to_int), _ = helper.
max_target_sentence_length = max([len(sentence) for sentence in source_int_text])

train_graph = tf.Graph()
with train_graph.as_default():
    input_data, targets, lr, keep_prob, target_sequence_length, max_target_sequence_length,

    #sequence_length = tf.placeholder_with_default(max_target_sentence_length, None, name='
    input_shape = tf.shape(input_data)

    train_logits, inference_logits = seq2seq_model(tf.reverse(input_data, [-1]),
                                                   targets,
                                                   keep_prob,
                                                   batch_size,
                                                   source_sequence_length,
                                                   target_sequence_length,
                                                   max_target_sequence_length,
                                                   len(source_vocab_to_int),
                                                   len(target_vocab_to_int),
                                                   encoding_embedding_size,
                                                   decoding_embedding_size,
                                                   rnn_size,
                                                   num_layers,
                                                   target_vocab_to_int)


    training_logits = tf.identity(train_logits.rnn_output, name='logits')
    inference_logits = tf.identity(inference_logits.sample_id, name='predictions')

    masks = tf.sequence_mask(target_sequence_length, max_target_sequence_length, dtype=tf.f

    with tf.name_scope("optimization"):
        # Loss function
        cost = tf.contrib.seq2seq.sequence_loss(
            training_logits,
            targets,
            masks)

        # Optimizer
        optimizer = tf.train.AdamOptimizer(lr)

        # Gradient Clipping
        gradients = optimizer.compute_gradients(cost)
        capped_gradients = [(tf.clip_by_value(grad, -1., 1.), var) for grad, var in gradier
        train_op = optimizer.apply_gradients(capped_gradients)
```

Batch and pad the source and target sequences

In [16]:

```
def pad_sentence_batch(sentence_batch, pad_int):
    """Pad sentences with <PAD> so that each sentence of a batch has the same length"""
    max_sentence = max([len(sentence) for sentence in sentence_batch])
    return [sentence + [pad_int] * (max_sentence - len(sentence)) for sentence in sentence_


def get_batches(sources, targets, batch_size, source_pad_int, target_pad_int):
    """Batch targets, sources, and the lengths of their sentences together"""
    for batch_i in range(0, len(sources)//batch_size):
        start_i = batch_i * batch_size

        # Slice the right amount for the batch
        sources_batch = sources[start_i:start_i + batch_size]
        targets_batch = targets[start_i:start_i + batch_size]

        # Pad
        pad_sources_batch = np.array(pad_sentence_batch(sources_batch, source_pad_int))
        pad_targets_batch = np.array(pad_sentence_batch(targets_batch, target_pad_int))

        # Need the lengths for the _lengths parameters
        pad_targets_lengths = []
        for target in pad_targets_batch:
            pad_targets_lengths.append(len(target))

        pad_source_lengths = []
        for source in pad_sources_batch:
            pad_source_lengths.append(len(source))

        yield pad_sources_batch, pad_targets_batch, pad_source_lengths, pad_targets_lengths
```

Train

Train the neural network on the preprocessed data. If you have a hard time getting a good loss, check the forms to see if anyone is having the same problem.

In [17]:

```
def get_accuracy(target, logits):
    """
    Calculate accuracy
    """
    max_seq = max(target.shape[1], logits.shape[1])
    if max_seq - target.shape[1]:
        target = np.pad(
            target,
            [(0,0),(0,max_seq - target.shape[1])],
            'constant')
    if max_seq - logits.shape[1]:
        logits = np.pad(
            logits,
            [(0,0),(0,max_seq - logits.shape[1])],
            'constant')

    return np.mean(np.equal(target, logits))

# Split data to training and validation sets
train_source = source_int_text[batch_size:]
train_target = target_int_text[batch_size:]
valid_source = source_int_text[:batch_size]
valid_target = target_int_text[:batch_size]
(valid_sources_batch, valid_targets_batch, valid_sources_lengths, valid_targets_lengths ) =




with tf.Session(graph=train_graph) as sess:
    sess.run(tf.global_variables_initializer())

    for epoch_i in range(epochs):
        for batch_i, (source_batch, target_batch, sources_lengths, targets_lengths) in enum
                get_batches(train_source, train_target, batch_size,
                            source_vocab_to_int['<PAD>'],
                            target_vocab_to_int['<PAD>'])):

            _, loss = sess.run(
                [train_op, cost],
                {input_data: source_batch,
                 targets: target_batch,
                 lr: learning_rate,
                 target_sequence_length: targets_lengths,
                 source_sequence_length: sources_lengths,
                 keep_prob: keep_probability})


            if batch_i % display_step == 0 and batch_i > 0:


                batch_train_logits = sess.run(
                    inference_logits,
                    {input_data: source_batch,
                     source_sequence_length: sources_lengths,
                     target_sequence_length: targets_lengths,
                     keep_prob: 1.0})


                batch_valid_logits = sess.run(
```

```
                    inference_logits,
                    {input_data: valid_sources_batch,
                     source_sequence_length: valid_sources_lengths,
                     target_sequence_length: valid_targets_lengths,
                     keep_prob: 1.0})

                train_acc = get_accuracy(target_batch, batch_train_logits)

                valid_acc = get_accuracy(valid_targets_batch, batch_valid_logits)

                print('Epoch {:>3} Batch {:>4}/{} - Train Accuracy: {:>6.4f}, Validation Ac
                      .format(epoch_i, batch_i, len(source_int_text) // batch_size, train_a

    # Save Model
    saver = tf.train.Saver()
    saver.save(sess, save_path)
    print('Model Trained and Saved')
```

```
Epoch   9 Batch   64/269 - Train Accuracy: 0.8877, Validation Accuracy: 0.
8826, Loss: 0.1718
Epoch   9 Batch   65/269 - Train Accuracy: 0.8774, Validation Accuracy: 0.
8847, Loss: 0.1837
Epoch   9 Batch   66/269 - Train Accuracy: 0.8814, Validation Accuracy: 0.
8847, Loss: 0.1838
Epoch   9 Batch   67/269 - Train Accuracy: 0.8915, Validation Accuracy: 0.
8826, Loss: 0.1990
Epoch   9 Batch   68/269 - Train Accuracy: 0.8813, Validation Accuracy: 0.
8852, Loss: 0.1952
Epoch   9 Batch   69/269 - Train Accuracy: 0.8783, Validation Accuracy: 0.
8803, Loss: 0.2099
Epoch   9 Batch   70/269 - Train Accuracy: 0.8888, Validation Accuracy: 0.
8847, Loss: 0.1919
Epoch   9 Batch   71/269 - Train Accuracy: 0.8831, Validation Accuracy: 0.
8813, Loss: 0.2026
Epoch   9 Batch   72/269 - Train Accuracy: 0.8816, Validation Accuracy: 0.
8795, Loss: 0.1895
Epoch   9 Batch   73/269 - Train Accuracy: 0.8757, Validation Accuracy: 0.
8807, Loss: 0.2002
```

Save Parameters

Save the `batch_size` and `save_path` parameters for inference.

In [18]:

```
# Save parameters for checkpoint
helper.save_params(save_path)
```

Checkpoint

In [19]:

```
import tensorflow as tf
import numpy as np
import helper
import problem_unittests as tests

_, (source_vocab_to_int, target_vocab_to_int), (source_int_to_vocab, target_int_to_vocab) =
load_path = helper.load_params()
```

Sentence to Sequence

To feed a sentence into the model for translation, you first need to preprocess it. Implement the function `sentence_to_seq()` to preprocess new sentences.

- Convert the sentence to lowercase
- Convert words into ids using `vocab_to_int`
 - Convert words not in the vocabulary, to the `<UNK>` word id.

In [20]:

```
def sentence_to_seq(sentence, vocab_to_int):
    """
    Convert a sentence to a sequence of ids
    :param sentence: String
    :param vocab_to_int: Dictionary to go from the words to an id
    :return: List of word ids
    """
    word_ids = []
    for word in sentence.lower().split():
        if word in vocab_to_int:
            word_ids.append(vocab_to_int[word])
        else:
            word_ids.append(vocab_to_int['<UNK>'])
    return word_ids


tests.test_sentence_to_seq(sentence_to_seq)
```

```
Tests Passed
```

Translate

This will translate `translate_sentence` from English to French.

In [21]:

```
translate_sentence = 'he saw a old yellow truck .'
translate_sentence = sentence_to_seq(translate_sentence, source_vocab_to_int)

loaded_graph = tf.Graph()
with tf.Session(graph=loaded_graph) as sess:
    # Load saved model
    loader = tf.train.import_meta_graph(load_path + '.meta')
    loader.restore(sess, load_path)

    input_data = loaded_graph.get_tensor_by_name('input:0')
    logits = loaded_graph.get_tensor_by_name('predictions:0')
    target_sequence_length = loaded_graph.get_tensor_by_name('target_sequence_length:0')
    source_sequence_length = loaded_graph.get_tensor_by_name('source_sequence_length:0')
    keep_prob = loaded_graph.get_tensor_by_name('keep_prob:0')

    translate_logits = sess.run(logits, {input_data: [translate_sentence]*batch_size,
                                         target_sequence_length: [len(translate_sentence)*2
                                         source_sequence_length: [len(translate_sentence)]*
                                         keep_prob: 1.0})[0]

print('Input')
print('  Word Ids:      {}'.format([i for i in translate_sentence]))
print('  English Words: {}'.format([source_int_to_vocab[i] for i in translate_sentence]))

print('\nPrediction')
print('  Word Ids:      {}'.format([i for i in translate_logits]))
print('  French Words: {}'.format(" ".join([target_int_to_vocab[i] for i in translate_logit
```

```
INFO:tensorflow:Restoring parameters from checkpoints/dev
Input
  Word Ids:      [71, 38, 215, 132, 156, 89, 42]
  English Words: ['he', 'saw', 'a', 'old', 'yellow', 'truck', '.']

Prediction
  Word Ids:      [99, 280, 170, 143, 221, 175, 348, 87, 270, 302, 1]
  French Words: il a vu le camion , et les pêches . <EOS>
```

In [ ]: