Language Translation

In this project, you’re going to take a peek into the realm of neural network machine translation. You’ll be training a sequence to sequence model on a dataset of English and French sentences that can translate new sentences from English to French.

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 [16]:

```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
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)
```

Explore the Data

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

In [17]:

```
view_sentence_range = (0, 10)

"""
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_text.split()})))

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 0 to 10:
new jersey is sometimes quiet during autumn , and it is snowy in april .
the united states is usually chilly during july , and it is usually freezing in november .
california is usually quiet during march , and it is usually hot in june .
the united states is sometimes mild during june , and it is cold in september .
your least liked fruit is the grape , but my least liked is the apple .
his favorite fruit is the orange , but my favorite is the grape .
paris is relaxing during december , but it is usually chilly in july .
new jersey is busy during spring , and it is never hot in march .
our least liked fruit is the lemon , but my least liked is the grape .
the united states is sometimes busy during january , and it is sometimes warm in november .

French sentences 0 to 10:
new jersey est parfois calme pendant l' automne , et il est neigeux en avril .
les états-unis est généralement froid en juillet , et il gèle habituellement en novembre .
```

```
california est généralement calme en mars , et il est généralement chaud en juin .
les états-unis est parfois légère en juin , et il fait froid en septembre .
votre moins aimé fruit est le raisin , mais mon moins aimé est la pomme .
son fruit préféré est l'orange , mais mon préféré est le raisin .
paris est relaxant en décembre , mais il est généralement froid en juillet .
new jersey est occupé au printemps , et il est jamais chaude en mars .
notre fruit est moins aimé le citron , mais mon moins aimé est le raisin .
les états-unis est parfois occupé en janvier , et il est parfois chaud en novembre .
```

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 each sentence from `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` .

Preprocess all the data and save it

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

In [18]:

```
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)
    """
    # TODO: Implement Function

    target_text = target_text.replace(".", ". <EOS>")

    source_id_text = []
    target_id_text = []

    for source_sentence in source_text.split("\n"):
        source_id_sentence = [ source_vocab_to_int[word] for word in source_sentence.split() ]
        source_id_text.append(source_id_sentence)

    for target_sentence in target_text.split("\n"):
        target_id_sentence = [ target_vocab_to_int[word] for word in target_sentence.split() ]
        target_id_text.append(target_id_sentence)

    return source_id_text, target_id_text

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_text_to_ids(text_to_ids)
```

```
Tests Passed
```

In [19]:

```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
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 [20]:

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

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

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 [21]:

```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
from distutils.version import LooseVersion
import warnings
import tensorflow as tf

# Check TensorFlow Version
assert LooseVersion(tf.__version__) in [LooseVersion('1.0.0'), LooseVersion('1.0.1')], 'This project requires Te
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: 1.0.0
Default GPU Device: /gpu:0
```

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_decoding_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.

Return the placeholders in the following the tuple (Input, Targets, Learing Rate, Keep Probability)

In [22]:

```
def model_inputs():
    """
    Create TF Placeholders for input, targets, and learning rate.
    :return: Tuple (input, targets, learning rate, keep probability)
    """
    # TODO: Implement Function

    inputs = tf.placeholder(tf.int32, [None, None], "input")
    targets = tf.placeholder(tf.int32, [None, None], "target")
    learning_rate = tf.placeholder(tf.float32, name="learning_rate")
    keep_probability = tf.placeholder(tf.float32, name="keep_prob")

    return inputs, targets, learning_rate, keep_probability

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_model_inputs(model_inputs)
```

```
Tests Passed
```

Process Decoding Input

Implement `process_decoding_input` using TensorFlow to remove the last word id from each batch in `target_data` and concat the GO ID to the beginning of each batch.

In [23]:

```
def process_decoding_input(target_data, target_vocab_to_int, batch_size):
    """
    Preprocess target data for dencoding
    :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
    """
    # TODO: Implement Function

    ending = tf.strided_slice(target_data, [0, 0], [batch_size, -1], [1, 1])
    decoded = tf.concat([tf.fill([batch_size, 1], source_vocab_to_int['<GO>']), ending], 1)

    return decoded

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_process_decoding_input(process_decoding_input)
```

```
Tests Passed
```

Encoding

Implement `encoding_layer()` to create a Encoder RNN layer using [`tf.nn.dynamic_rnn()` (https://www.tensorflow.org/api_docs/python/tf/nn/dynamic_rnn)](https://www.tensorflow.org/api_docs/python/tf/nn/dynamic_rnn).

In [24]:

```
def encoding_layer(rnn_inputs, rnn_size, num_layers, keep_prob):
    """
    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
    :return: RNN state
    """
    # TODO: Implement Function

    lstm = tf.contrib.rnn.BasicLSTMCell(rnn_size)

    dropout = tf.contrib.rnn.DropoutWrapper(lstm, output_keep_prob = keep_prob)

    cell = tf.contrib.rnn.MultiRNNCell([dropout] * num_layers)

    outputs, final_state = tf.nn.dynamic_rnn(cell, rnn_inputs, dtype=tf.float32)

    return final_state

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_encoding_layer(encoding_layer)
```

```
Tests Passed
```

Decoding - Training

Create training logits using `tf.contrib.seq2seq.simple_decoder_fn_train()` (https://www.tensorflow.org/versions/r1.0/api_docs/python/tf/contrib/seq2seq/simple_decoder_fn_train) and `tf.contrib.seq2seq.dynamic_rnn_decoder()` (https://www.tensorflow.org/versions/r1.0/api_docs/python/tf/contrib/seq2seq/dynamic_rnn_decoder). Apply the `output_fn` to the `tf.contrib.seq2seq.dynamic_rnn_decoder()` (https://www.tensorflow.org/versions/r1.0/api_docs/python/tf/contrib/seq2seq/dynamic_rnn_decoder) outputs.

In [25]:

```
def decoding_layer_train(encoder_state, dec_cell, dec_embed_input, sequence_length, decoding_scope,
                         output_fn, 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 sequence_length: Sequence Length
    :param decoding_scope: TenorFlow Variable Scope for decoding
    :param output_fn: Function to apply the output layer
    :param keep_prob: Dropout keep probability
    :return: Train Logits
    """
    # TODO: Implement Function

    dropout = tf.contrib.rnn.DropoutWrapper(dec_cell, output_keep_prob = keep_prob)

    train_decoder = tf.contrib.seq2seq.simple_decoder_fn_train(encoder_state)

    outputs, final_state, final_context_state = tf.contrib.seq2seq.dynamic_rnn_decoder(
        dropout,
        train_decoder,
        dec_embed_input,
        sequence_length,
        scope = decoding_scope
    )

    return output_fn(outputs)


"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_decoding_layer_train(decoding_layer_train)
```

```
Tests Passed
```

Decoding - Inference

Create inference logits using `tf.contrib.seq2seq.simple_decoder_fn_inference()` (https://www.tensorflow.org/versions/r1.0/api_docs/python/tf/contrib/seq2seq/simple_decoder_fn_inference) and `tf.contrib.seq2seq.dynamic_rnn_decoder()` (https://www.tensorflow.org/versions/r1.0/api_docs/python/tf/contrib/seq2seq/dynamic_rnn_decoder).

In [26]:

```
def decoding_layer_infer(encoder_state, dec_cell, dec_embeddings, start_of_sequence_id, end_of_sequence_id,
                         maximum_length, vocab_size, decoding_scope, output_fn, 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 maximum_length: The maximum allowed time steps to decode
    :param vocab_size: Size of vocabulary
    :param decoding_scope: TensorFlow Variable Scope for decoding
    :param output_fn: Function to apply the output layer
    :param keep_prob: Dropout keep probability
    :return: Inference Logits
    """
    # TODO: Implement Function

    inference_decoder_fn = tf.contrib.seq2seq.simple_decoder_fn_inference(
        output_fn,
        encoder_state,
        dec_embeddings,
        start_of_sequence_id,
        end_of_sequence_id,
        maximum_length,
        vocab_size
    )

    outputs, final_state, final_context_state = tf.contrib.seq2seq.dynamic_rnn_decoder(
        dec_cell,
        inference_decoder_fn,
        scope = decoding_scope
    )

    return outputs


"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_decoding_layer_infer(decoding_layer_infer)
```

```
Tests Passed
```

Build the Decoding Layer

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

- Create RNN cell for decoding using `rnn_size` and `num_layers` .
- Create the output fuction using [`lambda` (https://docs.python.org/3/tutorial/controlflow.html#lambda-expressions)](https://docs.python.org/3/tutorial/controlflow.html#lambda-expressions) to transform it's input, logits, to class logits.
- Use the your `decoding_layer_train(encoder_state, dec_cell, dec_embed_input, sequence_length, decoding_scope, output_fn, 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, maximum_length, vocab_size, decoding_scope, output_fn, 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)](https://www.tensorflow.org/api_docs/python/tf/variable_scope) to share variables between training and inference.

In [27]:

```
def decoding_layer(dec_embed_input, dec_embeddings, encoder_state, vocab_size, sequence_length, rnn_size,
                   num_layers, target_vocab_to_int, keep_prob):
    """
    Create decoding layer
    :param dec_embed_input: Decoder embedded input
    :param dec_embeddings: Decoder embeddings
    :param encoder_state: The encoded state
    :param vocab_size: Size of vocabulary
    :param sequence_length: Sequence Length
    :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 keep_prob: Dropout keep probability
    :return: Tuple of (Training Logits, Inference Logits)
    """
    # TODO: Implement Function

    lstm = tf.contrib.rnn.BasicLSTMCell(rnn_size)

    dropout = tf.contrib.rnn.DropoutWrapper(lstm, output_keep_prob = keep_prob)

    cell = tf.contrib.rnn.MultiRNNCell([dropout] * num_layers)

    with tf.variable_scope('decoding') as decoding_scope:

        output_fn = lambda x: tf.contrib.layers.fully_connected(
            x,
            vocab_size,
            None,
            scope = decoding_scope
        )

        train_logits = decoding_layer_train(
            encoder_state,
            cell,
            dec_embed_input,
            sequence_length,
            decoding_scope,
            output_fn,
            keep_prob
        )
        decoding_scope.reuse_variables()

        inference_logits = decoding_layer_infer(
            encoder_state,
            cell,
            dec_embeddings,
            target_vocab_to_int['<GO>'],
            target_vocab_to_int['<EOS>'],
            sequence_length,
            vocab_size,
            decoding_scope,
            output_fn,
            keep_prob
        )


    return train_logits, inference_logits


"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_decoding_layer(decoding_layer)
```

```
Tests Passed
```

Build the Neural Network

Apply the functions you implemented above to:

- Apply embedding to the input data for the encoder.
- Encode the input using your `encoding_layer(rnn_inputs, rnn_size, num_layers, keep_prob)` .
- Process target data using your `process_decoding_input(target_data, target_vocab_to_int, batch_size)` function.
- Apply embedding to the target data for the decoder.
- Decode the encoded input using your `decoding_layer(dec_embed_input, dec_embeddings, encoder_state, vocab_size, sequence_length, rnn_size, num_layers, target_vocab_to_int, keep_prob)` .

In [28]:
```
def seq2seq_model(input_data, target_data, keep_prob, batch_size, sequence_length, source_vocab_size, target_voc
                  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 sequence_length: Sequence Length
    :param source_vocab_size: Source vocabulary size
    :param target_vocab_size: Target vocabulary size
    :param enc_embedding_size: Encoder embedding size
    :param dec_embedding_size: Decoder 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 Logits, Inference Logits)
    """
    # TODO: Implement Function

    encoder_embed_input = tf.contrib.layers.embed_sequence(
        input_data,
        source_vocab_size,
        enc_embedding_size)

    encoder_state = encoding_layer(
        encoder_embed_input,
        rnn_size,
        num_layers,
        keep_prob=keep_prob)

    target_data = process_decoding_input(
        target_data,
        target_vocab_to_int,
        batch_size)

    decoder_embed = tf.Variable(tf.random_uniform([target_vocab_size, dec_embedding_size]))

    decoder_embed_input = tf.nn.embedding_lookup(decoder_embed, target_data)

    decoder_layer = decoding_layer(
        decoder_embed_input,
        decoder_embed,
        encoder_state,
        target_vocab_size,
        sequence_length,
        rnn_size,
        num_layers,
        target_vocab_to_int,
        keep_prob)

    return decoder_layer


"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
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

In [37]:

```
# Number of Epochs
epochs = 8
# Batch Size
batch_size = 512
# RNN Size
rnn_size = 256
# Number of Layers
num_layers = 2
# Embedding Size
encoding_embedding_size = 256
decoding_embedding_size = 256
# Learning Rate
learning_rate = 0.001
# Dropout Keep Probability
keep_probability = 0.5
```

Build the Graph

Build the graph using the neural network you implemented.

In [38]:

```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
save_path = 'checkpoints/dev'
(source_int_text, target_int_text), (source_vocab_to_int, target_vocab_to_int), _ = helper.load_preprocess()
max_source_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 = model_inputs()
    sequence_length = tf.placeholder_with_default(max_source_sentence_length, None, name='sequence_length')
    input_shape = tf.shape(input_data)

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

    tf.identity(inference_logits, 'logits')
    with tf.name_scope("optimization"):
        # Loss function
        cost = tf.contrib.seq2seq.sequence_loss(
            train_logits,
            targets,
            tf.ones([input_shape[0], sequence_length]))

        # 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 gradients if grad is not Nor
        train_op = optimizer.apply_gradients(capped_gradients)
```

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 [39]:

```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
import time

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]), (0,0)],
            'constant')

    return np.mean(np.equal(target, np.argmax(logits, 2)))

train_source = source_int_text[batch_size:]
train_target = target_int_text[batch_size:]

valid_source = helper.pad_sentence_batch(source_int_text[:batch_size])
valid_target = helper.pad_sentence_batch(target_int_text[:batch_size])

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) in enumerate(
                helper.batch_data(train_source, train_target, batch_size)):
            start_time = time.time()

            _, loss = sess.run(
                [train_op, cost],
                {input_data: source_batch,
                 targets: target_batch,
                 lr: learning_rate,
                 sequence_length: target_batch.shape[1],
                 keep_prob: keep_probability})

            batch_train_logits = sess.run(
                inference_logits,
                {input_data: source_batch, keep_prob: 1.0})
            batch_valid_logits = sess.run(
                inference_logits,
                {input_data: valid_source, keep_prob: 1.0})

            train_acc = get_accuracy(target_batch, batch_train_logits)
            valid_acc = get_accuracy(np.array(valid_target), batch_valid_logits)
            end_time = time.time()
            print('Epoch {:>3} Batch {:>4}/{} - Train Accuracy: {:>6.3f}, Validation Accuracy: {:>6.3f}, Loss: {
                  .format(epoch_i, batch_i, len(source_int_text) // batch_size, train_acc, valid_acc, loss))

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

```
Epoch   0 Batch    0/269 - Train Accuracy:  0.242, Validation Accuracy:  0.310, Loss:  5.868
Epoch   0 Batch    1/269 - Train Accuracy:  0.233, Validation Accuracy:  0.310, Loss:  5.494
Epoch   0 Batch    2/269 - Train Accuracy:  0.266, Validation Accuracy:  0.310, Loss:  5.034
Epoch   0 Batch    3/269 - Train Accuracy:  0.245, Validation Accuracy:  0.310, Loss:  4.795
Epoch   0 Batch    4/269 - Train Accuracy:  0.244, Validation Accuracy:  0.322, Loss:  4.728
Epoch   0 Batch    5/269 - Train Accuracy:  0.265, Validation Accuracy:  0.339, Loss:  4.594
Epoch   0 Batch    6/269 - Train Accuracy:  0.313, Validation Accuracy:  0.342, Loss:  4.186
Epoch   0 Batch    7/269 - Train Accuracy:  0.320, Validation Accuracy:  0.351, Loss:  4.101
Epoch   0 Batch    8/269 - Train Accuracy:  0.289, Validation Accuracy:  0.351, Loss:  4.159
Epoch   0 Batch    9/269 - Train Accuracy:  0.314, Validation Accuracy:  0.354, Loss:  3.921
Epoch   0 Batch   10/269 - Train Accuracy:  0.283, Validation Accuracy:  0.354, Loss:  3.971
Epoch   0 Batch   11/269 - Train Accuracy:  0.333, Validation Accuracy:  0.367, Loss:  3.760
Epoch   0 Batch   12/269 - Train Accuracy:  0.311, Validation Accuracy:  0.372, Loss:  3.824
Epoch   0 Batch   13/269 - Train Accuracy:  0.375, Validation Accuracy:  0.373, Loss:  3.464
Epoch   0 Batch   14/269 - Train Accuracy:  0.342, Validation Accuracy:  0.376, Loss:  3.574
Epoch   0 Batch   15/269 - Train Accuracy:  0.331, Validation Accuracy:  0.374, Loss:  3.551
Epoch   0 Batch   16/269 - Train Accuracy:  0.353, Validation Accuracy:  0.382, Loss:  3.443
Epoch   0 Batch   17/269 - Train Accuracy:  0.343, Validation Accuracy:  0.380, Loss:  3.401
Epoch   0 Batch   18/269 - Train Accuracy:  0.320, Validation Accuracy:  0.386, Loss:  3.484
Epoch   0 Batch   19/269 - Train Accuracy:  0.381, Validation Accuracy:  0.385, Loss:  3.172
```

Save Parameters

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

In [40]:

```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
# Save parameters for checkpoint
helper.save_params(save_path)
```

Checkpoint

In [41]:

```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
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) = helper.load_preproce
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 [42]:

```
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
    """
    # TODO: Implement Function

    unk = vocab_to_int['<UNK>']
    sentence = sentence.lower()
    ids = [vocab_to_int.get(w, unk) for w in sentence.split()]

    return ids


"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_sentence_to_seq(sentence_to_seq)
```

```
Tests Passed
```

Translate

This will translate `translate_sentence` from English to French.

In [44]:

```
translate_sentence = 'he saw a old yellow truck .'


"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
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('logits:0')
    keep_prob = loaded_graph.get_tensor_by_name('keep_prob:0')

    translate_logits = sess.run(logits, {input_data: [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 np.argmax(translate_logits, 1)]))
print('  French Words: {}'.format([target_int_to_vocab[i] for i in np.argmax(translate_logits, 1)]))
```

```
Input
  Word Ids:      [114, 76, 50, 60, 192, 129, 179]
  English Words: ['he', 'saw', 'a', 'old', 'yellow', 'truck', '.']

Prediction
  Word Ids:      [212, 297, 318, 145, 112, 106, 292, 166, 1]
  French Words: ['il', 'a', 'vu', 'un', 'camion', 'jaune', 'brillant', '.', '<EOS>']
```

In [ ]: