Language Translation- English to French.

Seq2Seq is a method of encoder-decoder based machine translation and language processing that maps an input of sequence to an output of sequence with a tag and attention value.

In this file, we'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.

The Data

Since translating the whole language of English to French will take lots of time to train, I'm making use of a small portion of the English corpus.

In [1]:

```
import helper
import problem_unittests as tests

#getting the data
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)
```

Exploring the Data:-

In [2]:

```
view_sentence_range = (2000, 2010)

import numpy as np

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

#splitting

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 2000 to 2010:
she saw the shiny red automobile .
new jersey is pleasant during july , and it is sometimes freezing in april .
the peach is your most loved fruit , but the lemon is our most loved .
he likes grapes , strawberries , and oranges.
new jersey is usually cold during november , but it is never warm in october .
```

```
the peach is her least favorite fruit , but the grapefruit is my least favorite .
china is never pleasant during spring , and it is usually dry in july .
california is never dry during fall , but it is never wonderful in autumn .
how was your visit to china last autumn ?
china is sometimes chilly during may , but it is sometimes cold in spring .

French sentences 2000 to 2010:
elle a vu la brillante voiture rouge .
new jersey est agréable en juillet , et il est parfois le gel en avril .
la pêche est votre fruit le plus aimé , mais le citron est notre plus aimé .
il aime les raisins , les fraises et les oranges .
new jersey est généralement froid en novembre , mais il est jamais chaud en octobre .
la pêche est moins son fruit préféré , mais le pamplemousse est mon préféré moins .
chine est jamais agréable au printemps , et il est généralement sec en juillet .
californie est jamais à sec pendant l' automne , mais il est jamais merveilleux à l' auto
mne .
comment était votre visite en chine l' automne dernier ?
la chine est parfois frisquet en mai , mais il est parfois froid au printemps .
```

Preprocessing Function:-

Text to Word Ids

We'll now turn the text into a number to make our computer understand it. In the function `text_to_ids()`, we will 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 particular thing will help the neural network predict when the sentence should end.

Creating a function named "text_to_ids" which will fulfil our purpose.

In [3]:

```
def text_to_ids(source_text, target_text, source_vocab_to_int, target_vocab_to_int):

#converting source and target text to proper word ids

    source_sentences = source_text.split('\n')
    target_sentences = target_text.split('\n')
    source_id_text = []
    target_id_text = []

    for sentence in source_sentences:
        source_sentence_id_text = []
        for word in sentence.split():
            source_sentence_id_text.append(source_vocab_to_int[word])

        source_id_text.append(source_sentence_id_text)

    for sentence in target_sentences:
        target_sentence_id_text = []
        for word in sentence.split():
            target_sentence_id_text.append(target_vocab_to_int[word])

        target_sentence_id_text.append(target_vocab_to_int['<EOS>'])
        target_id_text.append(target_sentence_id_text)


    return source_id_text, target_id_text


tests.test_text_to_ids(text_to_ids)
```

```
Tests Passed
```

Saving the data after preprocessing it.

In [4]:

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

Building the Neural Network- The following are necessary to build a Sequence-to-Sequence model:-

process_decoding_input , encoding_layer , decoding_layer_train , decoding_layer_infer , decoding_layer , seq2seq_model

Function 'model_inputs' will return a tuple holding input, targets, learning rate, keep probability.

In [7]:

```
def model_inputs():

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

    return inputs, targets, learning_rate, keep_prob

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 begining of each batch.

In [8]:

```
def process_decoding_input(target_data, target_vocab_to_int, batch_size):

    ending = tf.strided_slice(target_data, [0, 0], [batch_size, -1], [1, 1])
    decoded_input = tf.concat([tf.fill([batch_size, 1], target_vocab_to_int['<GO>']), en
ding], 1)

    return decoded_input

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()`]

**Here, param rnn_inputs= Inputs for the RNN ,
param rnn_size= RNN Size,
param num_layers= Number of layers,
param keep_prob= Dropout keep probability**

In [9]:

```
def encoding_layer(rnn_inputs, rnn_size, num_layers, keep_prob):


    LSTM_cell = tf.contrib.rnn.BasicLSTMCell(rnn_size)
    LSTM_cell = tf.contrib.rnn.DropoutWrapper(LSTM_cell, keep_prob)
    encoded_cell = tf.contrib.rnn.MultiRNNCell([LSTM_cell] * num_layers)

    _, RNN_state = tf.nn.dynamic_rnn(encoded_cell, rnn_inputs, dtype=tf.float32)
```

```
    return RNN_state


tests.test_encoding_layer(encoding_layer)
```

```
Tests Passed
```

Decoding - Training

Create training logits using [`tf.contrib.seq2seq.simple_decoder_fn_train()`] and [`tf.contrib.seq2seq.dynamic_rnn_decoder()`]. Apply the `output_fn` to the [`tf.contrib.seq2seq.dynamic_rnn_decoder()`] outputs.

In [10]:

```
def decoding_layer_train(encoder_state, dec_cell, dec_embed_input, sequence_length, decod
ing_scope,
                         output_fn, keep_prob):

    train_decoder_fn = tf.contrib.seq2seq.simple_decoder_fn_train(encoder_state)
    train_pred, _, _ = tf.contrib.seq2seq.dynamic_rnn_decoder(
        dec_cell, train_decoder_fn, inputs=dec_embed_input, sequence_length=sequence_len
gth, scope=decoding_scope)

    train_logits = output_fn(train_pred)

    return train_logits


tests.test_decoding_layer_train(decoding_layer_train)
```

```
Tests Passed
```

Decoding - Inference

Create inference logits using [`tf.contrib.seq2seq.simple_decoder_fn_inference()`] and [`tf.contrib.seq2seq.dynamic_rnn_decoder()`].

In [11]:

```
def decoding_layer_infer(encoder_state, dec_cell, dec_embeddings, start_of_sequence_id, e
nd_of_sequence_id,
                         maximum_length, vocab_size, decoding_scope, output_fn, keep_pro
b):


    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)

    inference_logits, _, _ = tf.contrib.seq2seq.dynamic_rnn_decoder(
        dec_cell, inference_decoder_fn, scope=decoding_scope)


    return inference_logits

tests.test_decoding_layer_infer(decoding_layer_infer)
```

```
Tests Passed
```

Build the Decoding Layer

Implementing the function `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`] to transform it's input, logits, to class logits.**

- Create the output fuction using [`lambda`] to transform it's input, logits, to class logits.

In [12]:

```
def decoding_layer(dec_embed_input, dec_embeddings, encoder_state, vocab_size, sequence_l
ength, rnn_size,
                   num_layers, target_vocab_to_int, keep_prob):

    # Creating RNN cell for decoding using rnn_size and num_layers
    dec_cell = tf.contrib.rnn.MultiRNNCell([tf.contrib.rnn.BasicLSTMCell(rnn_size)] * nu
m_layers)

    # Create output function using lambda to transform inputs, logits, to class logits
    with tf.variable_scope("decoding") as decoding_scope:
        output_fn = lambda x: tf.contrib.layers.fully_connected(x, vocab_size, None, sco
pe=decoding_scope)

    # Use decoding_layer_train() function to get training logits
    with tf.variable_scope("decoding") as decoding_scope:
        training_logits = decoding_layer_train(
            encoder_state, dec_cell, dec_embed_input, sequence_length, decoding_scope, o
utput_fn, keep_prob)

    # Use decoding_layer_infer() function to get inference logits
    with tf.variable_scope("decoding", reuse=True) as decoding_scope:
        inference_logits = decoding_layer_infer(
            encoder_state, dec_cell, dec_embeddings, target_vocab_to_int['<GO>'], target
_vocab_to_int['<EOS>'],
            sequence_length - 1, vocab_size, decoding_scope, output_fn, keep_prob)

    return training_logits, inference_logits


tests.test_decoding_layer(decoding_layer)
```

```
Tests Passed
```

Build the Neural Network

Now we will be applying the functions implemented above to and combine the model:-

In [13]:

```
def seq2seq_model(input_data, target_data, keep_prob, batch_size, sequence_length, source
_vocab_size, target_vocab_size,
                  enc_embedding_size, dec_embedding_size, rnn_size, num_layers, target_v
ocab_to_int):

    # Apply embedding to input data
    enc_embed_input = tf.contrib.layers.embed_sequence(input_data, source_vocab_size, en
c_embedding_size)

    # Encode the input using the encoding_layer() function.
    encoder_state = encoding_layer(enc_embed_input, rnn_size, num_layers, keep_prob)

    # Process target data using process_decoding_input() function.
    decoded_input = process_decoding_input(target_data, target_vocab_to_int, batch_size)

    # Apply embedding to target data for the decoder.
    dec_embeddings = tf.Variable(tf.random_uniform([target_vocab_size, dec_embedding_siz
e]))
    dec_embed_input = tf.nn.embedding_lookup(dec_embeddings, decoded_input)

    # Decode the encoded input using decoding_layer() function.
    training_logits, inference_logits = decoding_layer(
        dec_embed_input, dec_embeddings, encoder_state, target_vocab_size, sequence_leng
th,
        rnn_size, num_layers, target_vocab_to_int, keep_prob)

    return training_logits, inference_logits
```

```
tests.test_seq2seq_model(seq2seq_model)
```

```
Tests Passed
```

Neural Network Training

Defining all the hyperparameters:-

In [40]:

```
epochs = 30

batch_size = 256

rnn_size = 100

num_layers = 2

# Embedding Size
encoding_embedding_size = 100
decoding_embedding_size = 100

learning_rate = 0.001
# Dropout Keep Probability
keep_probability = 0.9
```

Build the Graph

Build the graph using the neural network you implemented.

In [41]:

```
save_path = 'checkpoints/dev'
(source_int_text, target_int_text), (source_vocab_to_int, target_vocab_to_int), _ = help
er.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, l
en(source_vocab_to_int), len(target_vocab_to_int),
        encoding_embedding_size, decoding_embedding_size, rnn_size, num_layers, target_vo
cab_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 grad
ients if grad is not None]
        train_op = optimizer.apply_gradients(capped_gradients)
```

Training the NN on the preprocessed data

In [42]:

```
import time

def get_accuracy(target, logits):

    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 Accurac
y: {:>6.3f}, Loss: {:>6.3f}'
              .format(epoch_i, batch_i, len(source_int_text) // batch_size, train_acc, v
alid_acc, loss))

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

```
Epoch   0 Batch  536/538 - Train Accuracy:  0.529, Validation Accuracy:  0.553, Loss:  1.
020
Epoch   1 Batch  536/538 - Train Accuracy:  0.626, Validation Accuracy:  0.639, Loss:  0.
605
Epoch   2 Batch  536/538 - Train Accuracy:  0.772, Validation Accuracy:  0.741, Loss:  0.
380
```

```
Epoch   3 Batch  536/538 - Train Accuracy:  0.863, Validation Accuracy:  0.840, Loss:  0.
224
Epoch   4 Batch  536/538 - Train Accuracy:  0.901, Validation Accuracy:  0.881, Loss:  0.
127
Epoch   5 Batch  536/538 - Train Accuracy:  0.921, Validation Accuracy:  0.906, Loss:  0.
086
Epoch   6 Batch  536/538 - Train Accuracy:  0.931, Validation Accuracy:  0.929, Loss:  0.
066
Epoch   7 Batch  536/538 - Train Accuracy:  0.943, Validation Accuracy:  0.929, Loss:  0.
054
Epoch   8 Batch  536/538 - Train Accuracy:  0.949, Validation Accuracy:  0.941, Loss:  0.
043
Epoch   9 Batch  536/538 - Train Accuracy:  0.956, Validation Accuracy:  0.947, Loss:  0.
037
Epoch  10 Batch  536/538 - Train Accuracy:  0.959, Validation Accuracy:  0.953, Loss:  0.
032
Epoch  11 Batch  536/538 - Train Accuracy:  0.957, Validation Accuracy:  0.946, Loss:  0.
027
Epoch  12 Batch  536/538 - Train Accuracy:  0.962, Validation Accuracy:  0.946, Loss:  0.
024
Epoch  13 Batch  536/538 - Train Accuracy:  0.964, Validation Accuracy:  0.948, Loss:  0.
022
Epoch  14 Batch  536/538 - Train Accuracy:  0.965, Validation Accuracy:  0.955, Loss:  0.
020
Epoch  15 Batch  536/538 - Train Accuracy:  0.971, Validation Accuracy:  0.953, Loss:  0.
017
Epoch  16 Batch  536/538 - Train Accuracy:  0.971, Validation Accuracy:  0.947, Loss:  0.
016
Epoch  17 Batch  536/538 - Train Accuracy:  0.974, Validation Accuracy:  0.958, Loss:  0.
016
Epoch  18 Batch  536/538 - Train Accuracy:  0.969, Validation Accuracy:  0.959, Loss:  0.
014
Epoch  19 Batch  536/538 - Train Accuracy:  0.967, Validation Accuracy:  0.961, Loss:  0.
013
Epoch  20 Batch  536/538 - Train Accuracy:  0.974, Validation Accuracy:  0.965, Loss:  0.
012
Epoch  21 Batch  536/538 - Train Accuracy:  0.975, Validation Accuracy:  0.966, Loss:  0.
011
Epoch  22 Batch  536/538 - Train Accuracy:  0.977, Validation Accuracy:  0.966, Loss:  0.
010
Epoch  23 Batch  536/538 - Train Accuracy:  0.975, Validation Accuracy:  0.965, Loss:  0.
010
Epoch  24 Batch  536/538 - Train Accuracy:  0.975, Validation Accuracy:  0.963, Loss:  0.
011
Epoch  25 Batch  536/538 - Train Accuracy:  0.972, Validation Accuracy:  0.973, Loss:  0.
010
Epoch  26 Batch  536/538 - Train Accuracy:  0.977, Validation Accuracy:  0.966, Loss:  0.
008
Epoch  27 Batch  536/538 - Train Accuracy:  0.976, Validation Accuracy:  0.968, Loss:  0.
009
Epoch  28 Batch  536/538 - Train Accuracy:  0.978, Validation Accuracy:  0.959, Loss:  0.
009
Epoch  29 Batch  536/538 - Train Accuracy:  0.975, Validation Accuracy:  0.965, Loss:  0.
009
Model Trained and Saved
```

Save Parameters

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

In [43]:

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

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.

`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 [45]:

```
def sentence_to_seq(sentence, vocab_to_int):


    # Convert sentence to lowercase.
    sentence_l = sentence.lower()

    # Convert words to ids using vocab_to_int.
    word_ids = []
    for word in sentence_l.split():
        if word in vocab_to_int.keys():
            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 [51]:

```
translate_sentence = 'france is lovely in the spring .'

late_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_l
ogits, 1)]))
```

```
Input
  Word Ids:      [175, 134, 2, 174, 146, 116, 56]
  English Words: ['france', 'is', '<UNK>', 'in', 'the', 'spring', '.']

Prediction
  Word Ids:      [333, 157, 121, 335, 297, 141, 179, 110, 1]
  French Words: ['la', 'france', 'est', 'jamais', 'sec', 'au', 'printemps', '.', '<EOS>']
```