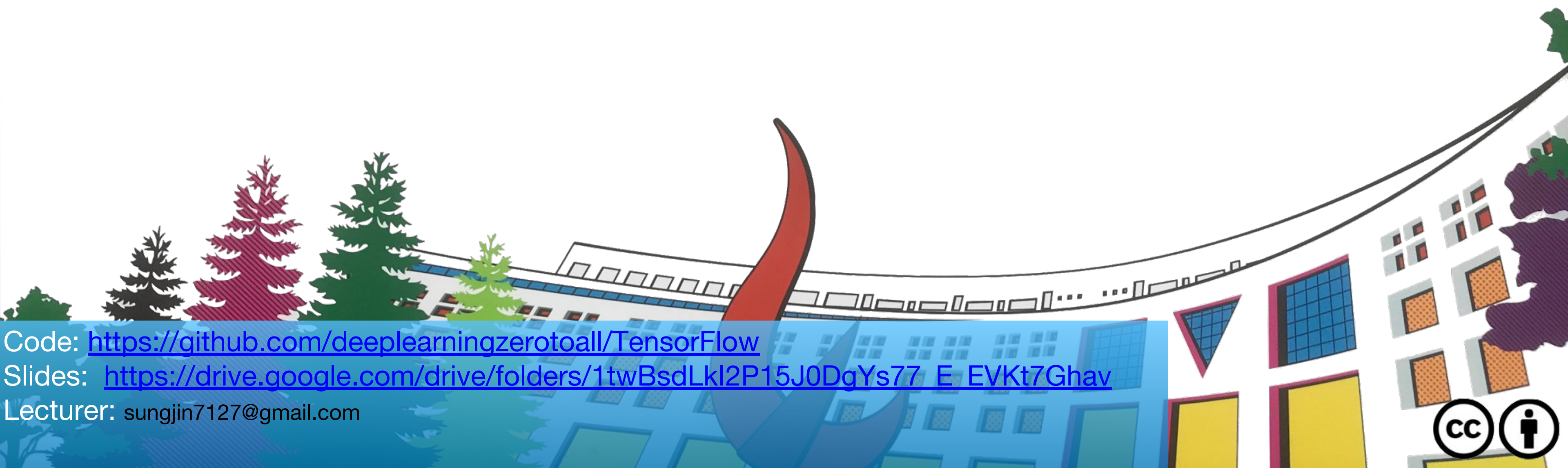
ML/DL for Everyone Season2

with TensorFlow

Lab 12-5: Seq2Seq



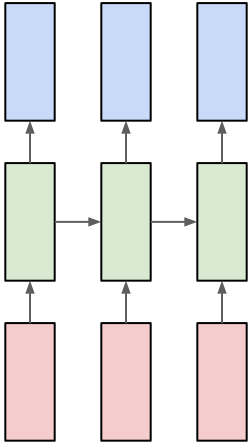
Code: https://github.com/deeplearningzerotoall/TensorFlow
Slides: https://drive.google.com/drive/folders/1twBsdLkI2P15J0DgYs77_E_EVKt7Ghav
Lecturer: sungjin7127@gmail.com

Lab12-5: Seq2Seq

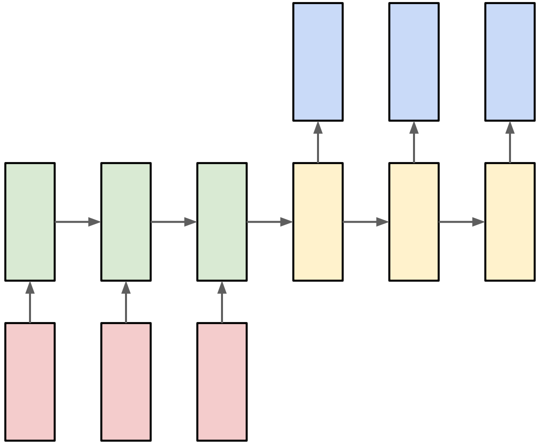
- Seq2Seq Overview
    - Chatbot Example
    - Encoder - Decoder
- Data Pipeline
- Encoder-Decoder
- Loss & Optimizer
- Train
- Prediction
- What’s Next?

Seq2Seq Overview

What is the difference between general RNN model and Seq2Seq model?

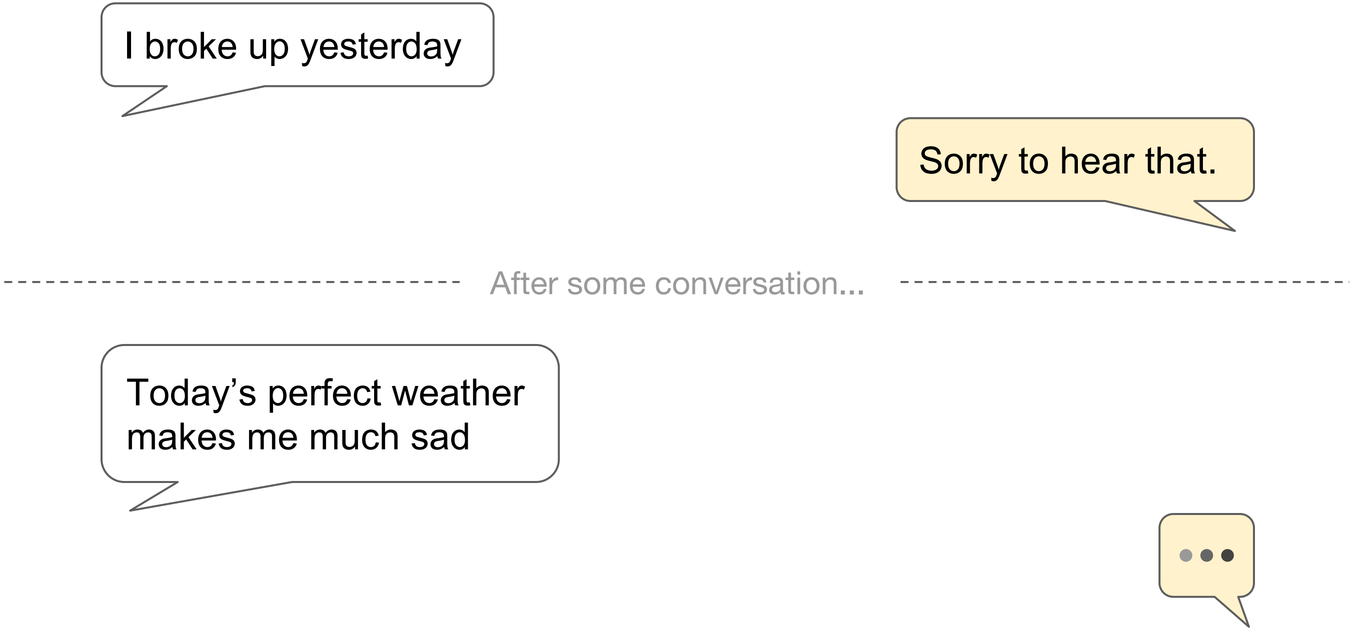


RNN



Seq2Seq

Example : Chatbot



I broke up yesterday

Sorry to hear that.

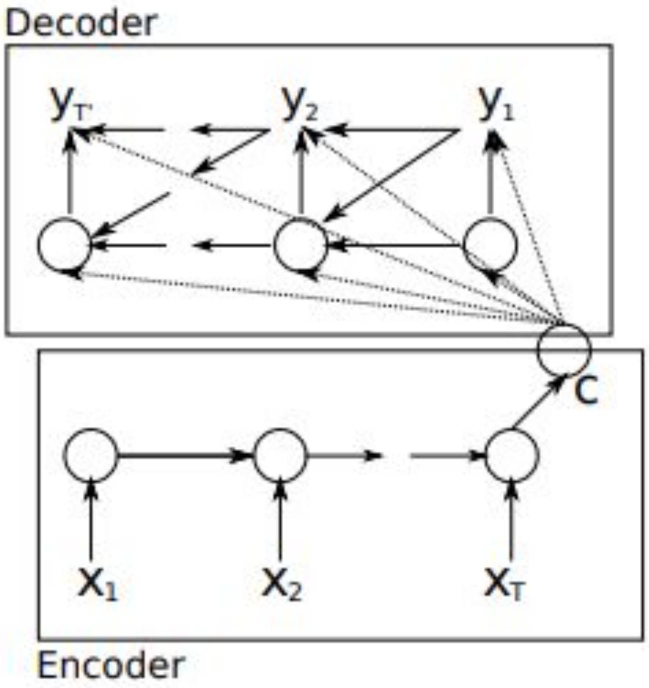
After some conversation...

Today’s perfect weather makes me much sad

●●●

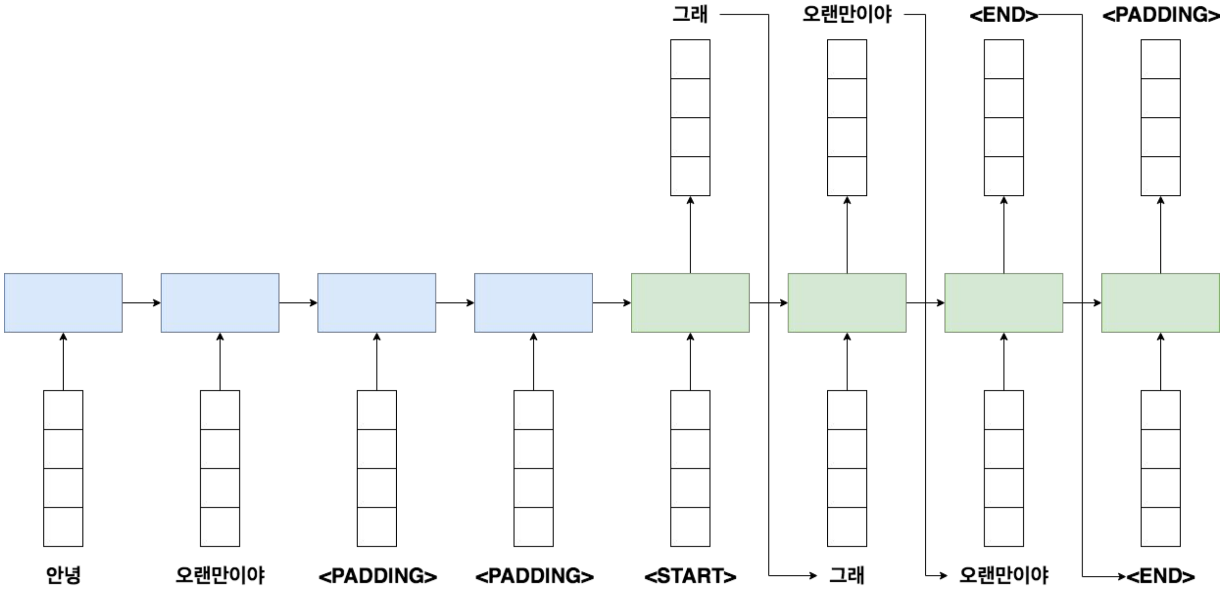
Today’s perfect weather makes me much **sad**

Encoder-Decoder



Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation
https://arxiv.org/abs/1406.1078

Encoder-Decoder



Data Pipeline: Dataset

```
sources = [['I', 'feel', 'hungry'],
     ['tensorflow', 'is', 'very', 'difficult'],
     ['tensorflow', 'is', 'a', 'framework', 'for', 'deep', 'learning'],
     ['tensorflow', 'is', 'very', 'fast', 'changing']]
targets = [['나는', '배가', '고프다'],
           ['텐서플로우는', '매우', '어렵다'],
           ['텐서플로우는', '딥러닝을', '위한', '프레임워크이다'],
           ['텐서플로우는', '매우', '빠르게', '변화한다']]
```

Data Pipeline: Vocab Dict

```python
# vocabulary for sources
s_vocab = list(set(sum(sources, [])))
s_vocab.sort()
s_vocab = ['<pad>'] + s_vocab
source2idx = {word : idx for idx, word in enumerate(s_vocab)}
idx2source = {idx : word for idx, word in enumerate(s_vocab)}

pprint(source2idx)
```

```
{'<pad>': 0,          'feel': 7,
 'I': 1,              'for': 8,
 'a': 2,              'framework': 9,
 'changing': 3,       'hungry': 10,
 'deep': 4,           'is': 11,
 'difficult': 5,      'learning': 12,
 'fast': 6,           'tensorflow': 13,
                      'very': 14}
```

```python
# vocabulary for targets
t_vocab = list(set(sum(targets, [])))
t_vocab.sort()
t_vocab = ['<pad>', '<bos>', '<eos>'] + t_vocab
target2idx = {word : idx for idx, word in enumerate(t_vocab)}
idx2target = {idx : word for idx, word in enumerate(t_vocab)}

pprint(target2idx)
```

```
{'<bos>': 1,          '매우': 6,
 '<eos>': 2,          '배가': 7,
 '<pad>': 0,          '변화한다': 8,
 '고프다': 3,          '빠르게': 9,
 '나는': 4,            '어렵다': 10,
 '딥러닝을': 5,        '위한': 11,
                      '텐서플로우는': 12,
                      '프레임워크이다': 13}
```

Data Pipeline: Preprocess (1/3)

```python
def preprocess(sequences, max_len, dic, mode = 'source'):
    assert mode in ['source', 'target'], 'source와 target 중에 선택해주세요.'

    if mode == 'source':
        # preprocessing for source (encoder)
        s_input = list(map(lambda sentence : [dic.get(token) for token in sentence], sequences))
        s_len = list(map(lambda sentence : len(sentence), s_input))
        s_input = pad_sequences(sequences = s_input, maxlen = max_len, padding = 'post',
                                 truncating = 'post')
        return s_len, s_input
```

⋮

Data Pipeline: Preprocess (2/3)

```
elif mode == 'target':
    # preprocessing for target (decoder)
    # input
    t_input = list(map(lambda sentence : ['<bos>'] + sentence + ['<eos>'], sequences))
    t_input = list(map(lambda sentence : [dic.get(token) for token in sentence], t_input))
    t_len = list(map(lambda sentence : len(sentence), t_input))
    t_input = pad_sequences(sequences = t_input, maxlen = max_len, padding = 'post',
                       truncating = 'post')

    # output
    t_output = list(map(lambda sentence : sentence + ['<eos>'], sequences))
    t_output = list(map(lambda sentence : [dic.get(token) for token in sentence], t_output))
    t_output = pad_sequences(sequences = t_output, maxlen = max_len, padding = 'post',
                             truncating = 'post')

    return t_len, t_input, t_output
```

Data Pipeline: Preprocess (3/3)

```
# preprocessing for source
s_max_len = 10
s_len, s_input = preprocess(sequences = sources,
                            max_len = s_max_len, dic = source2idx, mode = 'source')

# preprocessing for target
t_max_len = 12
t_len, t_input, t_output = preprocess(sequences = targets,
                                      max_len = t_max_len, dic = target2idx, mode = 'target')
```

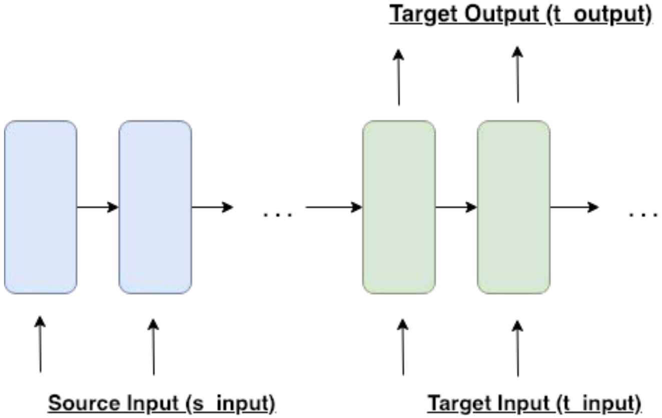
```
S_len: [3, 4, 7, 5]
S_input:
 [[ 1  7 10  0  0  0  0  0  0  0]
 [13 11 14  5  0  0  0  0  0  0]
 [13 11  2  9  8  4 12  0  0  0]
 [13 11 14  6  3  0  0  0  0  0]]
```

```
t_len: [5, 5, 6, 6]
t_input:
 [[ 1  4  7  3  2  0  0  0  0  0  0  0]
 [ 1 12  6 10  2  0  0  0  0  0  0  0]
 [ 1 12  5 11 13  2  0  0  0  0  0  0]
 [ 1 12  6  9  8  2  0  0  0  0  0  0]]
```

```
t_output
 [[ 4  7  3  2  0  0  0  0  0  0  0  0]
 [12  6 10  2  0  0  0  0  0  0  0  0]
 [12  5 11 13  2  0  0  0  0  0  0  0]
 [12  6  9  8  2  0  0  0  0  0  0  0]]
```

Data Pipeline: tf.data

```
# input
data = tf.data.Dataset.from_tensor_slices((s_len, s_input, t_len, t_input, t_output))
data = data.shuffle(buffer_size = buffer_size)
data = data.batch(batch_size = batch_size)
# s_mb_len, s_mb_input, t_mb_len, t_mb_input, t_mb_output = iterator.get_next()
```

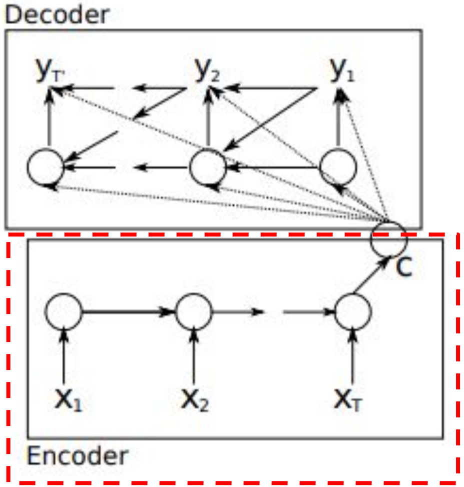


Encoder-Decoder

```
def gru(units):
  # If you have a GPU, we recommend using CuDNNGRU(provides a 3x speedup than GRU)
  # the code automatically does that.
    if tf.test.is_gpu_available():
        return tf.keras.layers.CuDNNGRU(units,
                                        return_sequences=True,
                                        return_state=True,
                                        recurrent_initializer='glorot_uniform')
    else:
        return tf.keras.layers.GRU(units,
                                   return_sequences=True,
                                   return_state=True,
                                   recurrent_activation='sigmoid',
                                   recurrent_initializer='glorot_uniform')
```

Encoder-Decoder: Encoder

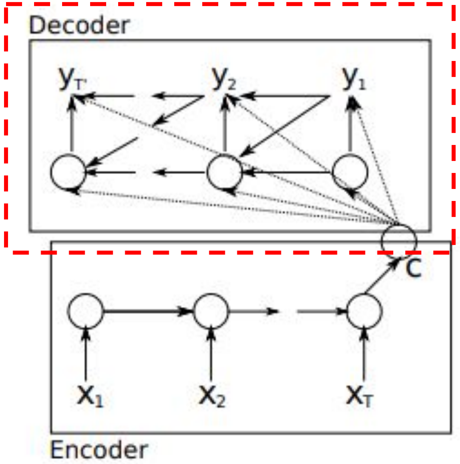
```
class Encoder(tf.keras.Model):
    def __init__(self, vocab_size, embedding_dim, enc_units, batch_sz):
        super(Encoder, self).__init__()
        self.batch_sz = batch_sz
        self.enc_units = enc_units
        self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
        self.gru = gru(self.enc_units)

    def call(self, x, hidden):
        x = self.embedding(x)
        output, state = self.gru(x, initial_state = hidden)

        return output, state

    def initialize_hidden_state(self):
        return tf.zeros((self.batch_sz, self.enc_units))
```



Encoder-Decoder: Decoder(1/2)

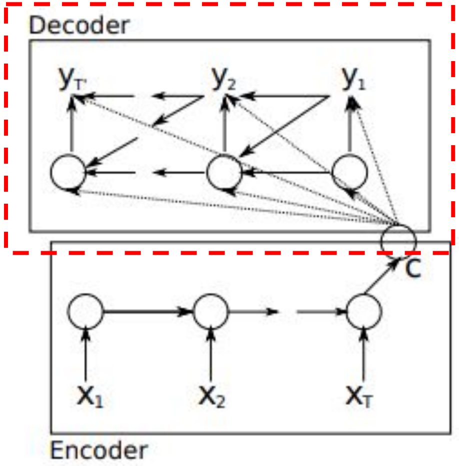
```
class Decoder(tf.keras.Model):
    def __init__(self, vocab_size, embedding_dim, dec_units, batch_sz):
        super(Decoder, self).__init__()
        self.batch_sz = batch_sz
        self.dec_units = dec_units
        self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
        self.gru = gru(self.dec_units)
        self.fc = tf.keras.layers.Dense(vocab_size)
```

⋮



Encoder-Decoder: : Decoder(2/2)

```
    def call(self, x, hidden, enc_output):

        x = self.embedding(x)
        output, state = self.gru(x, initial_state = hidden)
        # output shape == (batch_size * 1, hidden_size)
        output = tf.reshape(output, (-1, output.shape[2]))
        # output shape == (batch_size * 1, vocab)
        x = self.fc(output)

        return x, state

    def initialize_hidden_state(self):
        return tf.zeros((self.batch_sz, self.dec_units))

encoder = Encoder(len(source2idx), embedding_dim, units, batch_size
decoder = Decoder(len(target2idx), embedding_dim, units, batch_size)
```



Loss & Optimizer

```python
def loss_function(real, pred):
    mask = 1 - np.equal(real, 0)
    loss_ = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=real, logits=pred) * mask

    return tf.reduce_mean(loss_)

# creating optimizer
opt = tf.train.AdamOptimizer()

# creating check point (Object-based saving)
checkpoint_dir = './data_out/training_checkpoints'
checkpoint_prefix = os.path.join(checkpoint_dir, 'ckpt')
checkpoint = tf.train.Checkpoint(optimizer=optimizer,
                                 encoder=encoder,
                                 decoder=decoder)

# create writer for tensorboard
summary_writer = tf.contrib.summary.create_file_writer(logdir=checkpoint_dir)
```

Train (1/4)

```
EPOCHS = 100

for epoch in range(EPOCHS):
    hidden = encoder.initialize_hidden_state()
    total_loss = 0
    for i, (s_len, s_input, t_len, t_input, t_output) in enumerate(data):
        loss = 0
        with tf.GradientTape() as tape:
            enc_output, enc_hidden = encoder(s_input, hidden)
            dec_hidden = enc_hidden
            dec_input = tf.expand_dims([target2idx['<bos>']] * batch_size, 1)
            #Teacher Forcing: feeding the target as the next input
            for t in range(1, t_input.shape[1]):
                predictions, dec_hidden = decoder(dec_input, dec_hidden, enc_output)
                loss += loss_function(t_input[:, t], predictions)
                dec_input = tf.expand_dims(t_input[:, t], 1) #using teacher forcing
```

⋮

Train (2/4)

I feel hungry

W/O Teacher Forcing

	X	Y predict
1.	[bos]	a
2.	[bos], a	?

Teacher Forcing

1.	[bos]	?
2.	[bos], I	?
3.	[bos], I feel	?

Train (3/4)

```
    batch_loss = (loss / int(t_input.shape[1]))
    total_loss += batch_loss
    variables = encoder.variables + decoder.variables
    gradient = tape.gradient(loss, variables)
    optimizer.apply_gradients(zip(gradient, variables))

if epoch % 10 == 0:
    #save model every 10 epoch
    print('Epoch {} Loss {:.4f} Batch Loss {:.4f}'.format(epoch,
                                    total_loss / n_batch,
                                    batch_loss.numpy()))
    checkpoint.save(file_prefix = checkpoint_prefix)
```

Train (4/4)

```
Epoch 0 Loss 0.0307 Batch Loss 0.7687
Epoch 10 Loss 0.0297 Batch Loss 0.7414
Epoch 20 Loss 0.0267 Batch Loss 0.6676
Epoch 30 Loss 0.0237 Batch Loss 0.5925
Epoch 40 Loss 0.0159 Batch Loss 0.3964
Epoch 50 Loss 0.0131 Batch Loss 0.3271
Epoch 60 Loss 0.0100 Batch Loss 0.2498
Epoch 70 Loss 0.0075 Batch Loss 0.1874
Epoch 80 Loss 0.0051 Batch Loss 0.1283
Epoch 90 Loss 0.0031 Batch Loss 0.0763
```

⋮

Prediction (1/3)

```
def prediction(sentence, encoder, decoder, inp_lang, targ_lang, max_length_inp, max_length_targ):

    inputs = [inp_lang[i] for i in sentence.split(' ')]
    inputs = tf.keras.preprocessing.sequence.pad_sequences([inputs], maxlen=max_length_inp,
padding='post')
    inputs = tf.convert_to_tensor(inputs)

    result = ''

    hidden = [tf.zeros((1, units))]
    enc_out, enc_hidden = encoder(inputs, hidden)

    dec_hidden = enc_hidden
    dec_input = tf.expand_dims([targ_lang['<bos>']], 0)
```

⋮

Prediction (2/3)

```
for t in range(max_length_targ):
    predictions, dec_hidden = decoder(dec_input, dec_hidden, enc_out)

    predicted_id = tf.argmax(predictions[0]).numpy()

    result += idx2target[predicted_id] + ' '

    if idx2target.get(predicted_id) == '<eos>':
        return result, sentence

    # the predicted ID is fed back into the model
    dec_input = tf.expand_dims([predicted_id], 0)

return result, sentence
```

Prediction (3/3)

```
sentence = 'tensorflow is a framework for deep learning'
sentence = 'I feel hungry'

result, output_sentence = prediction(sentence, encoder, decoder, source2idx, target2idx, s_max_len,
t_max_len)
```

```
Result: tensorflow is a framework for deep learning
Output Sentence: 텐서플로우는 딥러닝을 위한 프레임워크이다 <eos>

Result: I feel hungry
OUtput Sentence: 나는 배가 고프다 <eos>
```

What’s Next?

- Seq2Seq Attention

Reference

- Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation: https://arxiv.org/abs/1406.1078
- 텐서플로우와 머신러닝으로 시작하는 자연어처리 (Wikibooks): http://wikibook.co.kr/nlp/