ML/DL for Everyone Season2

with TensorFlow

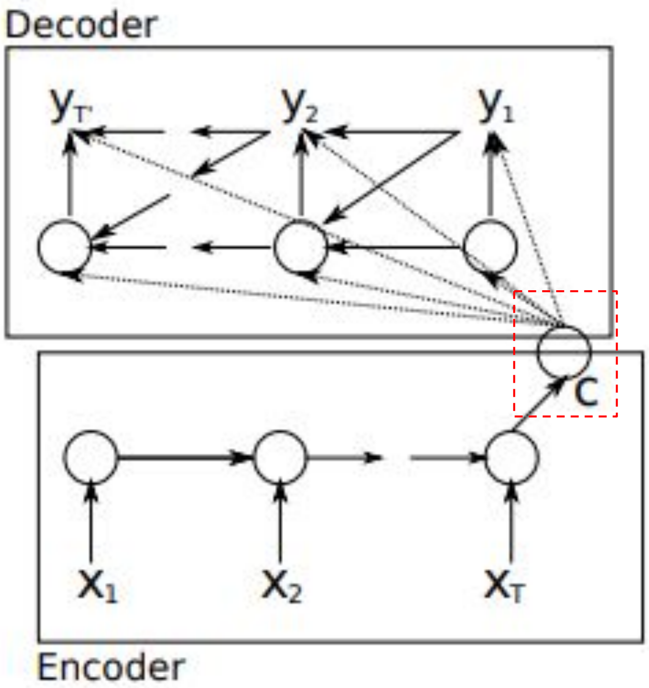
Lab 12-6: Seq2Seq with Attention

Code: https://github.com/deeplearningzerotoall/TensorFlow
Slides: http://bit.ly/2LQMKvk
Lecturer: sungjin7127@gmail.com

Lab12-6: Seq2Seq Attention

- Seq2Seq Attention Overview
- Encoder
- Decoder (Attention)
- Train
- Prediction

Problem with Seq2Seq



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

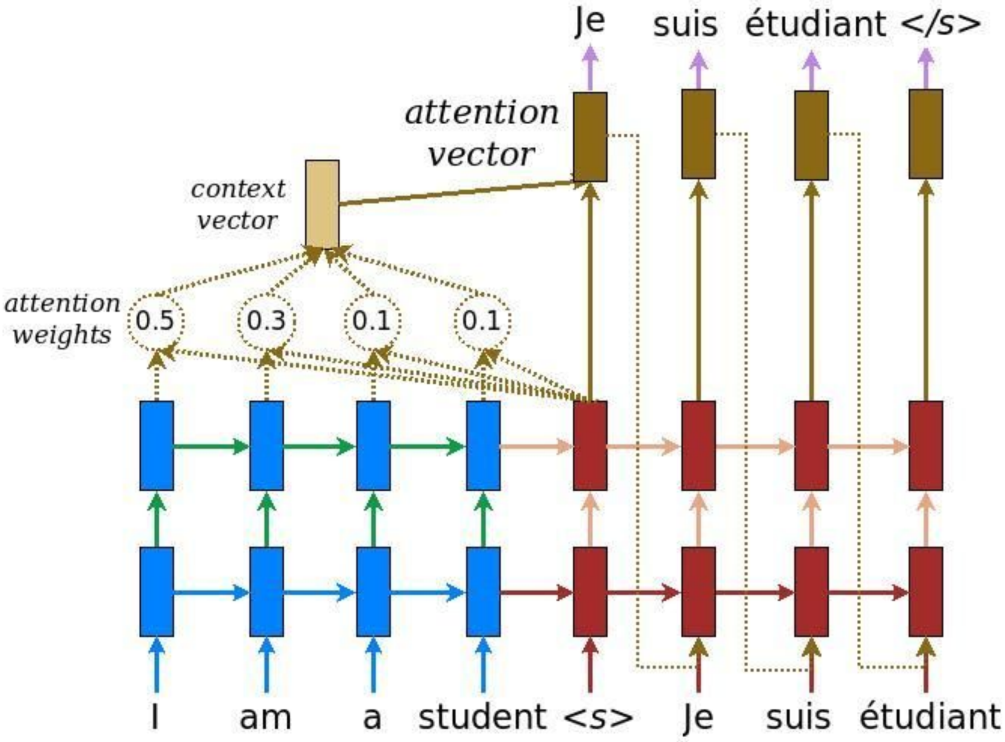
What is Attention

We want to focus more on “Important” information

텐서플로우는 딥러닝을 위한 훌륭한 프레임워크 중 하나이다.

Tensorflow is one of the great frameworks for deep learning

What is Attention?



https://www.tensorflow.org/images/seq2seq/attention_mechanism.jpg

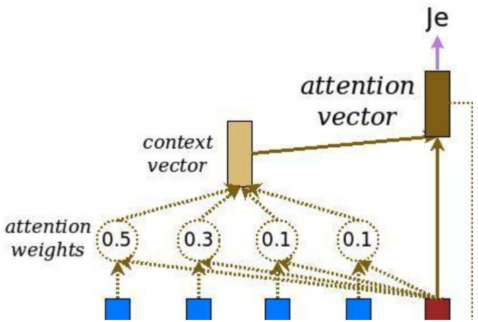
Data Pipeline: Dataset

```
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']]
```

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))
```

Decoder (W/ Attention)



$$\alpha_{ts} = \frac{\exp\left(\mathrm{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s)\right)}{\sum_{s'=1}^{S} \exp\left(\mathrm{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_{s'})\right)} \qquad \text{[Attention weights]}$$

$$\boldsymbol{c}_t = \sum_s \alpha_{ts} \bar{\boldsymbol{h}}_s \qquad \text{[Context vector]}$$

$$\boldsymbol{a}_t = f(\boldsymbol{c}_t, \boldsymbol{h}_t) = \tanh(\boldsymbol{W_c}[\boldsymbol{c}_t; \boldsymbol{h}_t]) \qquad \text{[Attention vector]}$$

$$\mathrm{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) = \begin{cases} \boldsymbol{h}_t^\top \boldsymbol{W} \bar{\boldsymbol{h}}_s & \text{[Luong's multiplicative style]} \\ \boxed{\boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W_1} \boldsymbol{h}_t + \boldsymbol{W_2} \bar{\boldsymbol{h}}_s\right)} & \text{[Bahdanau's additive style]} \end{cases}$$

Decoder (W/ Attention)

```
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)

        # used for attention
        self.W1 = tf.keras.layers.Dense(self.dec_units)
        self.W2 = tf.keras.layers.Dense(self.dec_units)
        self.V = tf.keras.layers.Dense(1)
```

⋮

Decoder (W/ Attention)

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

    hidden_with_time_axis = tf.expand_dims(hidden, 1)
    score = self.V(tf.nn.tanh(self.W1(enc_output) + self.W2(hidden_with_time_axis)))

    attention_weights = tf.nn.softmax(score, axis=1)

    context_vector = attention_weights * enc_output
    context_vector = tf.reduce_sum(context_vector, axis=1)
```

$$\mathrm{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) = \begin{cases} \boldsymbol{h}_t^\top \boldsymbol{W} \bar{\boldsymbol{h}}_s \\ \boxed{\boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W}_1 \boldsymbol{h}_t + \boldsymbol{W}_2 \bar{\boldsymbol{h}}_s\right)} \end{cases}$$

$$\alpha_{ts} = \frac{\exp\left(\mathrm{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s)\right)}{\sum_{s'=1}^{S} \exp\left(\mathrm{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_{s'})\right)}$$

$$\boldsymbol{c}_t = \sum_s \alpha_{ts} \bar{\boldsymbol{h}}_s$$

⋮

Decoder (W/ Attention)

```
    x = self.embedding(x)

    x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)

    # passing the concatenated vector to the GRU
    output, state = self.gru(x)

    # 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, attention_weights

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

$$\boldsymbol{a}_t = f(\boldsymbol{c}_t, \boldsymbol{h}_t) = \tanh(\boldsymbol{W}_c[\boldsymbol{c}_t; \boldsymbol{h}_t])$$

Train (1/3)

```
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 (2/3)

```
        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 (3/3)

Basic S2S

Epoch 0 Loss 0.0396 Batch Loss 0.9894
Epoch 10 Loss 0.0387 Batch Loss 0.9682
Epoch 20 Loss 0.0375 Batch Loss 0.9379
Epoch 30 Loss 0.0353 Batch Loss 0.8830
Epoch 40 Loss 0.0315 Batch Loss 0.7864
Epoch 50 Loss 0.0281 Batch Loss 0.7034
Epoch 60 Loss 0.0249 Batch Loss 0.6222
Epoch 70 Loss 0.0221 Batch Loss 0.5528
Epoch 80 Loss 0.0195 Batch Loss 0.4887
Epoch 90 Loss 0.0172 Batch Loss 0.4290

⋮

S2S with Attention

Epoch 0 Loss 0.0396 Batch Loss 0.9905
Epoch 10 Loss 0.0381 Batch Loss 0.9515
Epoch 20 Loss 0.0349 Batch Loss 0.8719
Epoch 30 Loss 0.0328 Batch Loss 0.8209
Epoch 40 Loss 0.0297 Batch Loss 0.7431
Epoch 50 Loss 0.0223 Batch Loss 0.5575
Epoch 60 Loss 0.0174 Batch Loss 0.4351
Epoch 70 Loss 0.0124 Batch Loss 0.3088
Epoch 80 Loss 0.0076 Batch Loss 0.1900
Epoch 90 Loss 0.0043 Batch Loss 0.1071

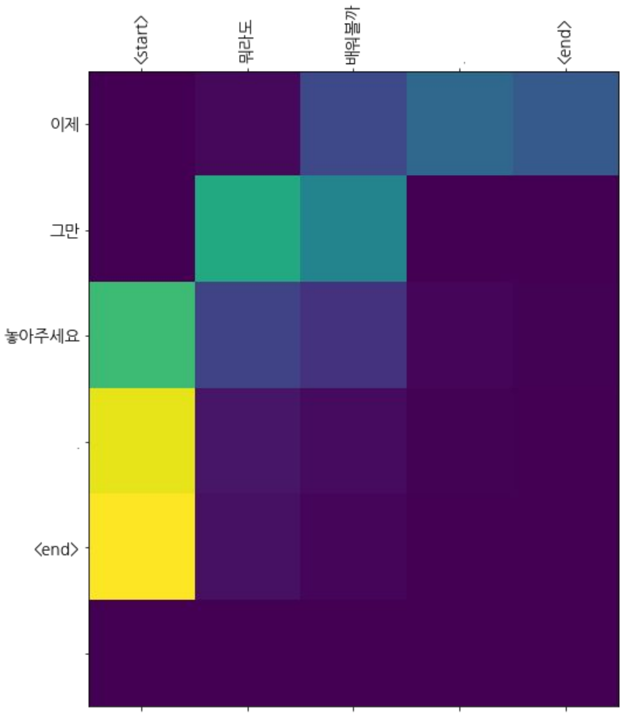
⋮

Prediction

```
sentence = 'tensorflow is a framework for deep learning'

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

sentence = 'I feel hungry'

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

Bonus: Chatbot

Lab 12-7 (bonus): Seq2Seq Attention Eng-Kor Chatbot (keras-eager version)



Korean Chatbot data from
https://github.com/songys/Chatbot_data
(MIT License)

Reference

- Sequence to Sequence Learning with Neural Networks: https://arxiv.org/abs/1409.3215
- Effective Approaches to Attention-based Neural Machine Translation: https://arxiv.org/abs/1508.04025
- Neural Machine Translation with Attention from Tensorflow: https://github.com/tensorflow/tensorflow/blob/master/tensorflow/contrib/eager/python/examples/nmt_with_attention/nmt_with_attention.ipynb
- 텐서플로우와 머신러닝으로 시작하는 자연어처리 (Wikibooks)
- **Korean Chatbot data from https://github.com/songys/Chatbot_data (MIT License)**

Decoder (W/ Attention)

We're using *Bahdanau attention*. Lets decide on notation before writing the simplified form:

- FC = Fully connected (dense) layer
- EO = Encoder output
- H = hidden state
- X = input to the decoder

And the pseudo-code:

- `score = FC(tanh(FC(EO) + FC(H)))`
- `attention weights = softmax(score, axis = 1)`. Softmax by default is applied on the last axis but here we want to apply it on the *1st axis*, since the shape of score is *(batch_size, max_length, 1)*. `Max_length` is the length of our input. Since we are trying to assign a weight to each input, softmax should be applied on that axis.
- `context vector = sum(attention weights * EO, axis = 1)`. Same reason as above for choosing axis as 1.
- `embedding output` = The input to the decoder X is passed through an embedding layer.
- `merged vector = concat(embedding output, context vector)`
- This merged vector is then given to the GRU

The shapes of all the vectors at each step have been specified in the comments in the code: