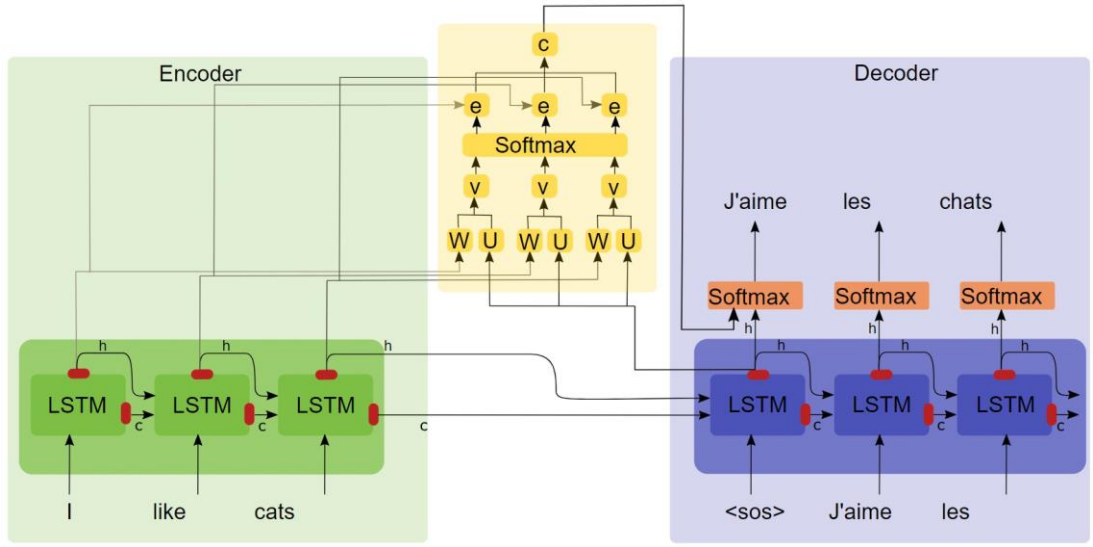
Bài Tập Attention

Người thực hiện 1: Nguyễn Đức Toàn - 520H0690

Chương 1

Giải Thích Attention

1.1 Kiến trúc của mô hình



1.2 Encode

Sau khi trải qua Encode thì ta được các encode state: $H = [h_1, h_3, h_3, ..., h_n]$

1.3 Decode

Tại decode, để dự đoán ra $word_j$, ta cần tính context vector CV_j tương ứng. Attention dùng khái niệm query, key, value:

- Keys $= [k_1, k_2, ..., k_n]$, values $= [v_1, v_2, ..., v_n]$. Thông thường, keys = values = $[h_1, h_2, ...h_n]$ (là các encode states)
- Queries $= [q_1, q_2, .., q_m] = [h_n, d_1, d_2, ...d_{m-1}]$
- Context vector (CV_j) là tổng hợp có trọng số các phần tử thuộc values. $CV_j = \Sigma_{i=0}^{n} (w_i^j * v_i)$ trong đó công thức tính trọng số là $w^j = [w_1^j, w_2^j, w_3^j, ..., w_n^j] =$ Softmax($[s_1^j, s_2^j, ..., s_n^j]$) và $s_0^j = \text{Dense}(\text{Tanh}(\text{Dense}(q^j) + \text{Dense}(k_i))$
- Kết hợp CV_j và từ liền trước(step1 thì từ liền trước là <START>) thì sẽ thu được concat vector, đưa concat vector vào GRU/LSTM thu được từ mới output. Tiếp tục lần lượt cho đến khi output = <END>

1.4 Code

- Ở đây chúng mình sẽ ưu tiên giải thích các tính toán liên quan đến decoding và attention. Còn những câu lệnh khác mình sẽ bỏ qua và giải thích kĩ hơn ở phần demo code. Đầu tiên là lớp Attention:

```
class Attention(tf.keras.Model):
    def __init__(self, hidden_units):
        super(Attention, self).__init__()
        self.W_out_encode = tf.keras.layers.Dense(hidden_unit)
        self.W_state = tf.keras.layers.Dense(hidden_unit)
        self.V = tf.keras.layers.Dense(1)

    def call(self, encode_outs, pre_state):
        pre_state = tf.expand_dims(pre_state, axis=1)
        pre_state = self.W_state(pre_state)
        encode_outs = self.W_out_encode(encode_outs)
        score = self.V(
            tf.nn.tanh(
                pre_state + encode_outs)
        )
        score = tf.nn.softmax(score, axis=1)
        context_vector = score*encode_outs
        context_vector = tf.reduce_sum(context_vector, axis=1)
        return context_vector, score
```

- $W_{outEncode}$: sẽ nhận đầu vào là dense của encode outputs (tức là các giá trị đầu ra từ encode, là các sequences). Sau khi qua lớp dense thì W out encode giống như các Hidden states
- W_{state}: sẽ nhận đầu vào là dense của pre state(là các trạng thái trước đó đã được tạo bởi model)
- V: sẽ tính Dense(Tanh($W_{outEncode}$ + W_{state}))
- trọng số score = Softmax(V)
- ContextVector(CV) = score * encode outs

Tiếp theo là lớp Decode:

```
class Decode(tf.keras.Model):
    def __init__(self, vocab_size, embedding_size, hidden_units):
        super(Decode, self).__init__()
        self.hidden_units = hidden_units
        self.Embedding = tf.keras.layers.Embedding(vocab_size,embedding_size)
        self.Attention = Attention(hidden_units)
        self.GRU = tf.keras.layers.GRU(
            hidden_units,
            return_sequences=True,
            return_state=True,
            recurrent_initializer='glorot_uniform'
        )
        self.Fc = tf.keras.layers.Dense(vocab_size)

    def call(self, x, encode_outs, pre_state):
        x = tf.expand_dims(x, axis=1)
        try:
            x = self.Embedding(x)
        except:
            print(x, print(tar_lang.vocab_size))
        context_vector, attention_weight = self.Attention(encode_outs, pre_state)
        context_vector = tf.expand_dims(context_vector, axis=1)
        gru_inp = tf.concat([x, context_vector], axis=-1)
        out_gru, state = self.GRU(gru_inp)
        out_gru = tf.reshape(out_gru, (-1, out_gru.shape[2]))
        return self.Fc(out_gru), state


decode = Decode(tar_lang.vocab_size, embedding_size, hidden_unit)
print(last_state.shape, tmp_outputs.shape, tmp_y[:, 0].shape)
decode_out, state = decode(tmp_y[:, 0], tmp_outputs, last_state)
```

Sau khi tính được Context vector từ Attention(encode outs, pre state) thì nối với x. X ở đây là từ liền trước của output. Vậy là chúng ta sẽ có concat vector. Sau đó đưa concat vector vào GRU và ta được output và state của bước hiện tại.

Cuối cùng là phần nối giữa Encode và Decode để huấn luyện mô hình:

```
for epoch in range(EPOCHS):
    total_loss = 0
    for batch_id, (x, y) in enumerate(dataset.take(N_BATCH)):
        loss = 0
        with tf.GradientTape() as tape:
            first_state = encoder.init_hidden_state(batch_size=BATCH_SIZE)
            encode_outs, last_state = encoder(x, first_state)
            decode_state = last_state
            decode_input = [tar_lang.word2id["<start>"]]*BATCH_SIZE

            for i in range(1, y.shape[1]):
                decode_out, decode_state = decoder(
                        decode_input, encode_outs, decode_state
                )
                loss += loss_function(y[:, i], decode_out)
                decode_input = y[:, i]

            train_vars = encoder.trainable_variables + decoder.trainable_variables
            grads = tape.gradient(loss, train_vars)
            optimizer.apply_gradients(zip(grads, train_vars))
        total_loss += loss
    print(total_loss.numpy())
```

- Đầu tiên trong phần này là tạo một state rỗng (Ma trận 0) là first state
- Sau đó ta tính được encode outs(có chứa tất cả các sequences của từng bước) và last state (có chứa tất cả các hidden state). Khi đó state đầu tiên của decode là last state(state cuối cùng của Encode) - Sau đó theo từng bước thì ra sẽ trả về decode state để làm pre state cho bước tiếp theo. cứ thế ta lặp đến khi kết thúc

1.5 Code khi không sử dụng attention

Đương nhiên việc đầu tiên là bỏ đi lớp Attention. Ở đây mình sẽ vô hiệu hóa đoạn code của lớp Attention. Mình sẽ biến đoạn code đó thành comment để sau này có thể tái sử dụng

```
[ ]  # class Attention(tf.keras.Model):
     #     def __init__(self, hidden_units):
     #         super(Attention, self).__init__()
     #         self.W_out_encode = tf.keras.layers.Dense(hidden_unit)
     #         self.W_state = tf.keras.layers.Dense(hidden_unit)
     #         self.V = tf.keras.layers.Dense(1)

     #     def call(self, encode_outs, pre_state):
     #         pre_state = tf.expand_dims(pre_state, axis=1)
     #         pre_state = self.W_state(pre_state)
     #         encode_outs = self.W_out_encode(encode_outs)
     #         score = self.V(
     #             tf.nn.tanh(
     #                 pre_state + encode_outs)
     #         )
     #         score = tf.nn.softmax(score, axis=1)
     #         context_vector = score*encode_outs
     #         context_vector = tf.reduce_sum(context_vector, axis=1)
     #         return context_vector, score
```

```
[ ]  # attention = Attention(hidden_unit)
     # context_vector, attention_weight = attention(tmp_outputs, last_state)
     # print(context_vector.shape, attention_weight.shape)
```

Tiếp theo là mình sẽ sửa lại đoạn code của Encode. Mình sửa lại đoạn return sequences = False. Như vậy thì GRU sẽ không trả về toàn bộ trạng thái của từng bước mà chỉ trả về trạng thái cuối cùng của Encode.

```
class Encode(tf.keras.Model):
    def __init__(self, embedding_size, vocab_size, hidden_units):
        super(Encode, self).__init__()
        self.Embedding = tf.keras.layers.Embedding(vocab_size,embedding_size)
        self.GRU = tf.keras.layers.GRU(
            hidden_units,
            return_sequences=False,
            return_state=True,
            recurrent_initializer='glorot_uniform')
        self.hidden_units = hidden_units

    def call(self, x, hidden_state):
        x = self.Embedding(x)
        outputs, last_state = self.GRU(x, hidden_state)
        return outputs, last_state

    def init_hidden_state(self, batch_size):
        return tf.zeros([batch_size, self.hidden_units])
```

Cuối cùng thì mình sẽ sửa lại đoạn code của Decode. Mình sẽ vô hiệu hóa thuộc tính Attention của Decode. Sau đó mình sẽ vô hiệu hóa phần code tính Context Vector của Attention và phần nối concat vector. Thay vì đưa con catvector vào GRU thì mình sẽ đưa x và pre state vào GRU.

```
class Decode(tf.keras.Model):
    def __init__(self, vocab_size, embedding_size, hidden_units):
        super(Decode, self).__init__()
        self.hidden_units = hidden_units
        self.Embedding = tf.keras.layers.Embedding(vocab_size,embedding_size)
        #self.Attention = Attention(hidden_units)
        self.GRU = tf.keras.layers.GRU(
            hidden_units,
            return_sequences=True,
            return_state=True,
            recurrent_initializer='glorot_uniform'
        )
        self.Fc = tf.keras.layers.Dense(vocab_size)

    def call(self, x, encode_outs, pre_state):
        x = tf.expand_dims(x, axis=1)
        x = self.Embedding(x)
        # context_vector, attention_weight = self.Attention(encode_outs, pre_state)
        # context_vector = tf.expand_dims(context_vector, axis=1)
        # gru_inp = tf.concat([x, context_vector], axis=-1)
        out_gru, state = self.GRU(x, pre_state)
        out_gru = tf.reshape(out_gru, (-1, out_gru.shape[2]))
        return self.Fc(out_gru), state
decode = Decode(tar_lang.vocab_size, embedding_size, hidden_unit)
print(last_state.shape, tmp_outputs.shape, tmp_y[:, 0].shape)
decode_out, state = decode(tmp_y[:, 0], tmp_outputs, last_state)
```

1.6 So sánh 2 code:

Đây là code dùng Attention: (điểm khác biệt so với không dùng attention. Không phải toàn bộ code)

- Lớp Attention:

```
class Attention(tf.keras.Model):
    def __init__(self, hidden_units):
        super(Attention, self).__init__()
        self.W_out_encode = tf.keras.layers.Dense(hidden_unit)
        self.W_state = tf.keras.layers.Dense(hidden_unit)
        self.V = tf.keras.layers.Dense(1)

    def call(self, encode_outs, pre_state):
        pre_state = tf.expand_dims(pre_state, axis=1)
        pre_state = self.W_state(pre_state)
        encode_outs = self.W_out_encode(encode_outs)
        score = self.V(
            tf.nn.tanh(
                pre_state + encode_outs)
        )
        score = tf.nn.softmax(score, axis=1)
        context_vector = score*encode_outs
        context_vector = tf.reduce_sum(context_vector, axis=1)
        return context_vector, score
```

- Lớp Encode:

```
class Encode(tf.keras.Model):
    def __init__(self, embedding_size, vocab_size, hidden_units):
        super(Encode, self).__init__()
        self.Embedding = tf.keras.layers.Embedding(vocab_size,embedding_size)
        self.GRU = tf.keras.layers.GRU(
            hidden_units,
            return_sequences=True,
            return_state=True,
            recurrent_initializer='glorot_uniform')
        self.hidden_units = hidden_units

    def call(self, x, hidden_state):
        x = self.Embedding(x)
        outputs, last_state = self.GRU(x, hidden_state)
        return outputs, last_state

    def init_hidden_state(self, batch_size):
        return tf.zeros([batch_size, self.hidden_units])
```

- Lớp Decode:

```
class Decode(tf.keras.Model):
    def __init__(self, vocab_size, embedding_size, hidden_units):
        super(Decode, self).__init__()
        self.hidden_units = hidden_units
        self.Embedding = tf.keras.layers.Embedding(vocab_size,embedding_size)
        self.Attention = Attention(hidden_units)
        self.GRU = tf.keras.layers.GRU(
            hidden_units,
            return_sequences=True,
            return_state=True,
            recurrent_initializer='glorot_uniform'
        )
        self.Fc = tf.keras.layers.Dense(vocab_size)

    def call(self, x, encode_outs, pre_state):
        x = tf.expand_dims(x, axis=1)
        try:
            x = self.Embedding(x)
        except:
            print(x, print(tar_lang.vocab_size))
        context_vector, attention_weight = self.Attention(encode_outs, pre_state)
        context_vector = tf.expand_dims(context_vector, axis=1)
        gru_inp = tf.concat([x, context_vector], axis=-1)
        out_gru, state = self.GRU(gru_inp)
        out_gru = tf.reshape(out_gru, (-1, out_gru.shape[2]))
        return self.Fc(out_gru), state


decode = Decode(tar_lang.vocab_size, embedding_size, hidden_unit)
print(last_state.shape, tmp_outputs.shape, tmp_y[:, 0].shape)
decode_out, state = decode(tmp_y[:, 0], tmp_outputs, last_state)
```

Đây là code không dùng Attention:
- Lớp Attention: Không có
- Lớp Encode:

```
class Encode(tf.keras.Model):
    def __init__(self, embedding_size, vocab_size, hidden_units):
        super(Encode, self).__init__()
        self.Embedding = tf.keras.layers.Embedding(vocab_size,embedding_size)
        self.GRU = tf.keras.layers.GRU(
            hidden_units,
            return_sequences=False,
            return_state=True,
            recurrent_initializer='glorot_uniform')
        self.hidden_units = hidden_units

    def call(self, x, hidden_state):
        x = self.Embedding(x)
        outputs, last_state = self.GRU(x, hidden_state)
        return outputs, last_state

    def init_hidden_state(self, batch_size):
        return tf.zeros([batch_size, self.hidden_units])
```

- Lớp Decode:

```
class Decode(tf.keras.Model):
    def __init__(self, vocab_size, embedding_size, hidden_units):
        super(Decode, self).__init__()
        self.hidden_units = hidden_units
        self.Embedding = tf.keras.layers.Embedding(vocab_size,embedding_size)
        #self.Attention = Attention(hidden_units)
        self.GRU = tf.keras.layers.GRU(
            hidden_units,
            return_sequences=True,
            return_state=True,
            recurrent_initializer='glorot_uniform'
        )
        self.Fc = tf.keras.layers.Dense(vocab_size)

    def call(self, x, encode_outs, pre_state):
        x = tf.expand_dims(x, axis=1)
        x = self.Embedding(x)
        # context_vector, attention_weight = self.Attention(encode_outs, pre_state)
        # context_vector = tf.expand_dims(context_vector, axis=1)
        # gru_inp = tf.concat([x, context_vector], axis=-1)
        out_gru, state = self.GRU(x, pre_state)
        out_gru = tf.reshape(out_gru, (-1, out_gru.shape[2]))
        return self.Fc(out_gru), state
decode = Decode(tar_lang.vocab_size, embedding_size, hidden_unit)
print(last_state.shape, tmp_outputs.shape, tmp_y[:, 0].shape)
decode_out, state = decode(tmp_y[:, 0], tmp_outputs, last_state)
```