Typo 1:

2.2 Write code to inplement global attention with dot product/multiplicative attention function

rubric={mechanics:1}

Intead of using the **additive attention** to get attenntion score a_t:

$$\alpha_i = softmax(v_a \tanh(W_a[S_{i-1}; H]^T))$$

We will use the *global attention with dot product alignment function* (Dot-Product/Multiplicative based attention function, that is, $\boxed{\alpha_i = softmax([s_i^T h_1, \cdots, s_i^T h_t])}$ **to calculate attention score α_i and follow the subsequent steps to generate translation token $\hat{y_{t+1}}$:**

1. initialize the `outputs` tensor is created to hold all predictions, $\hat{Y} = \{\hat{y_1} \dots \hat{y_t}\}$ where t is the maximal length of target language;
2. the source sequence, $X = \{x_1, \dots, x_t\}$, is fed into the encoder to receive last hidden state, h_t, and last cell state $c_t^{Encoder}$;
3. the initial decoder hidden state is set to be the h_t, and the initial decoder cell state is set to be the c_t. (i.e., $s_0 = h_t$; $c_0^{Decoder} = c_t^{Encoder}$);
4. we use a batch of `<sos>` tokens as the first `input` (i.e., y_0);
5. we then decode within a loop:

for i in range(1,t): t is the maximal length of target language

A. inserting the input token y_i, previous hidden state, s_{i-1}, and previous cell state, $c_{i-1}^{Decoder}$, into the Decoder we get new states, s_i and $c_i^{Decoder}$;
B. use **`attention_function()`** to calculate attention vector based on $h_1, \dots, h_t$ (all encoder hidden states stacked up is H) and s_i;
C. use this attention vector to create a weighted context vector, c_i, denoted by `weighted`, which is a weighted sum of the encoder hidden states, H, using α_i as the weights (i.e., $c_i = \alpha_i^T H$);
D. concatenate the current hidden state s_i with weighted context vector c_i, then give this concatanation to a linear layer, f_{mid}, to get a new hidden state s_i' that shape is `[batch, decoder hidden dimension]`;
E. pass s_i' through the linear layer, f_{output}, to make a prediction of the next word in the target sentence, $\hat{y}_{t+1}$.
F. decide if use **teacher forcing** or not, setting the next input as appropriate.

where **`attention_function()`** is based on **dot product attention**: $\alpha_i = softmax([s_i^T h_1, \cdots, s_i^T h_t])$

The pseudo code for computing attention vector:

```
class Decoder(nn.Module):
    INPUT:
    current Decoder hidden state (s_{i}), decoder output: [batch size, dec hid dim]
    all hidden state of last layer of Encoder (H), encoder_outputs: [src len, batch size, enc hid dim]
    OUTPUT:
    attention_vector (a_t), attention_vector: (batch_size, src_len)
    ---------------------------------------------------------------------------------
    # For-loop version:
    attention_vector = Variable(torch.zeros(batch_size, ecoder_src_len))

    # For every batch, every time step of encoder's hidden state, calculate attention score.
    for b in range(batch_size):
        for t in range(max_src_len):
            # Luong et al. (2015) equation(8) -- dot form content-based attention:
            attention_vector[b,t] = decoder_output[b] **dot product** encoder_outputs[t,b]

    ---------------------------------------------------------------------------------
```

```
#trg = [trg len, batch size]

batch_size = trg.shape[1]

# create a placeholder for traget language with shape of [max_trg_len, batch_size] where all the eleme
trg_placeholder = torch.Tensor(max_trg_len, batch_size)
trg_placeholder.fill_(TRG_PAD_IDX)
trg_placeholder = trg_placeholder.long().to(device)
if attention == True:
    output,_ = model(src, trg_placeholder, 0) #turn off teacher forcing
else:
    #original
    #output,_ = model(src, trg_placeholder, 0) #turn off teacher forcing

    # update:
    output = model(src, trg_placeholder, 0) #turn off teacher forcing
# get translation results, we ignor first token <sos> in both translation and target sentences.
# output_translate = [(trg len - 1), batch, output dim] output dim is size of target vocabulary.
output_translate = output[1:]
# store gold target sentences to a list
all_trg.append(trg[1:].cpu())

# Choose top 1 word from decoder's output, we get the probability and index of the word
prob, token_id = output_translate.data.topk(1)
translation_token_id = token_id.squeeze(2).cpu()

# store gold target sentences to a list
all_translated_trg.append(translation_token_id)
```