1 So You Want To Understand Attention?

This small write up is intended to clearly explain how Bahdanau attention (don't cate to cite) is computed for our project. According to Bahdanau, the attention energy is computed by

$$e_{ij} = a(s_{i-1}, h_j) = v_a^T \tanh\left(W_a s_{i-1} + U_a h_j\right)$$

where $W_a, U_a \in \mathbb{R}^{n \times n}$, $v_a \in \mathbb{R}^n = \mathbb{R}^{n \times 1}$, $s_{i-1} \in \mathbb{R}^n = \mathbb{R}^{n \times 1}$. Here s_{i-1} is the decoders previous hidden state and each h_j is the final hidden state layer of input j from the encoder. The yielded e_{ij} is then an integer, and computing this across all h_j yields a vector vector of shape $\mathbf{e}_i \in \mathbb{R}^l$ where l is the input sequence length.

We then compute

$$\alpha_{ij} = \text{softmax}(e_{ij})$$

meaning we can essentially describe a vector of alphas as $\bar{\alpha}_i = \text{softmax}(\mathbf{e_i})$. Given this we can compute the context vector by

$$\mathbf{c} = \bar{\alpha}_i^T \mathbf{H}$$

Where $\mathbf{H} = [h_1, h_2, ..., h_l]^T$ Because $\mathbf{H} \in \mathbb{R}^{l \times n}$ and $\bar{\alpha}_i^T \in \mathbb{R}^{1 \times l}$, this yields a vector of size $\mathbf{c} \in \mathbb{R}^{1 \times n}$, which is exactly what we want for the context vector.

2 So How Do They Use The Context Vector?

According to Bahdanau (minus their appendix on how they specifically implemented the attention), the goal is to model the conditional probability

$$\text{P}\left(y_i | y_1, ..., y_{i-1}, \mathbf{x}\right) = g(y_{i-1}, s_i, c_i)$$

Where s_i again the current decoder hidden output defined by

$$s_i = f(s_{i-1}, y_{i-1}, c_i)$$

The function f is a recurrent operation, meaning it is an operation that follows from a recurrent modelling step (RNN, LSTM, GRU). g is then any function what so ever.

3 Fine, But What Does That Mean For Us?

Alright, the linear algebra aside, what we are interested is:

1. Usually f, the recurrent function, cannot have arbitrary arguments added, and we need to model this.
2. what about g, how do we model this?

The first question is answered by a linear layer $\mathbf{L}_1$ transforming a concatenation between the decoder input $\mathbf{E}x_i$ and the context vector c_i across features, netting a feature vector of length $n2$, resulting in a linear map

$$\mathbf{L}_1 : \mathbb{R}^{2n} \mapsto \mathbb{R}^n$$

This result can then be inputted to our recurrent layer in the following way:

$$s_i = f(y_{i-1}, s_{i-1}, c_i) = RNN(k(y_{i-1}, c_i), s_{i-1})$$
$$k(y_{i-1}, c_i) = \text{ReLU}(\mathbf{L}_1\mathbf{E}(y_{i-1} \odot c_i^T))$$

The reason why we transpose c_i is that it returns to us as a row vector and we want it as a column vector. When implementing it can be beneficial to simply make the attention computation return a column vector.

The g function is, as noted in the main paper, then designed as a linear map $\mathbf{L}_2$ that maps the concatenation of s_i and c_i across features from the space $\mathbb{R}^{2n}$ to $\mathbb{R}^n$:

$$\mathbf{L}_2 : \mathbb{R}^{2n} \mapsto \mathbb{R}^n$$

the output, according to the main paper (whatever it's called) then softmaxes this:

$$o_i = g(y_{i-1}, s_i, c_i) = \text{softmax}(\mathbf{L}_2(s_i \odot c_i))$$

We then return o_i.