Attention mechanisms

Alberto Mario Pirovano
albertom.pirovano@gmail.com

Outline

- Vanilla Sequence to Sequence models
- Attention based Sequence to Sequence models
- Hierarchical Attention Networks for question answering

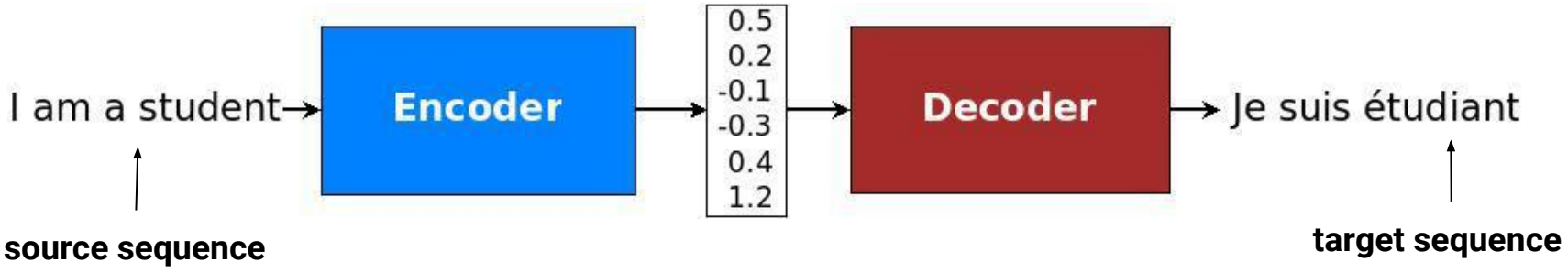
Outline

- **Vanilla Sequence to Sequence models**
- Attention based Sequence to Sequence models
- Hierarchical Attention Networks for question answering

Seq2seq models - General

Sequence-to-sequence models (Sutskever et al., 2014, Cho et al., 2014) are used in a variety of tasks such as **machine translation (NMT)**, **speech recognition**, **text summarization, question/answering, document classification, spell checking** and they can be seen as **bi-modular encoder-decoder architectures.**

A **seq2seq** model first reads the **source sequence** and, by using an **encoder**, it builds an **hidden representatio**(below [0.5, 0.2, -0.1, 0.3, 0.4, 1.2]); a **decoder**, then, processes the hidden representation to emit the **target sequence**.



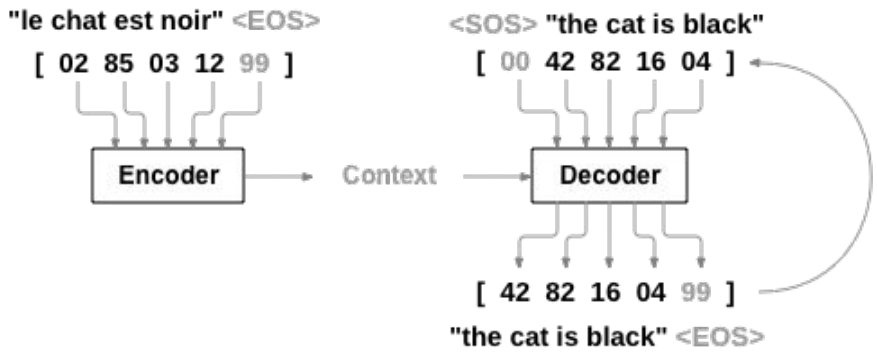
Seq2seq models - Preparation

1) Sample batch_size pairs of (**source_sequence**, **target_sequence**).

2) Prepend **<SOS>** to the target_sequence to obtain the **target_input_sequence** and append **<EOS>** to obtain the **target_output_sequence**. Append <EOS> to the source_sequence.

3) Pad them up to the **max_input_length (max_target_length)** within the batch by using the **<PAD>** token.

4) Lookup them by using their vocabularies and replace out of vocabulary (**OOV**) tokens with **<UNK>**. Compute the length of each input and target sequence in the batch.



Target vocabulary = {"<SOS>": 00,
"<EOS>": 99,
"<UNK>": 01,
"<PAD>": 03,
"the": 42,
"is": 16,
...]

Seq2seq models - Dimensionality

For each training step, the model is fed with a **batch object** containing these elements:

1) encoder_inputs: 2D tensor **[b_s, max_source_length],** 3D after em,b **[b_s, max_source_length, d_model]**

2) encoder_lengths: 1D tensor **[b_s]**

3) decoder_inputs: 2D tensor **[b_s, max_target_length],** 3D after emb **[b_s, max_source_length, d_model]**

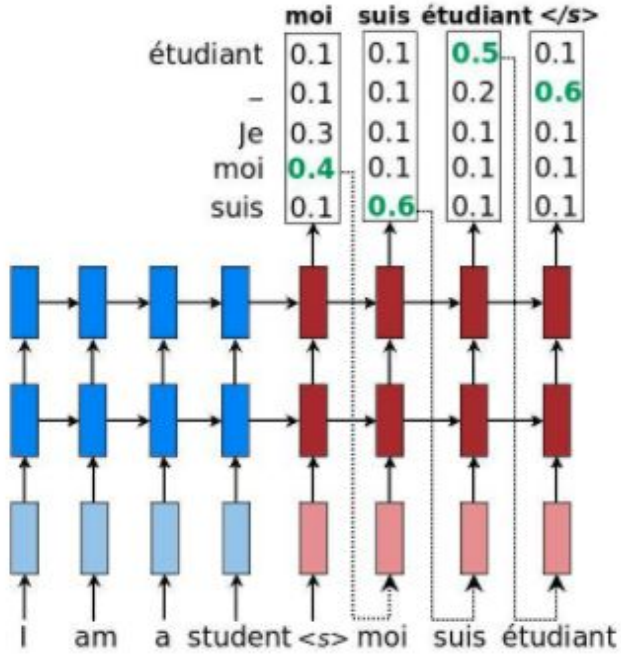
4) decoder_outputs: 2D tensor **[b_s, max_target_length],** 3D after emb **[b_s, max_source_length, d_model]**

5) decoder_lengths: 1D tensor **[b_s]**

The lengths are needed to allow the model to zero-out outputs when past a batch element's sequence length. This is done in tensorflow with **tf.nn.dynamic_rnn**. (So it's more for correctness than performance.)

Seq2seq models - Structure

- Usually, the **seq2seq** modules consist of two RNNs:
  - **encoder RNN: (BLUE on the left)** consumes the **encoder_inputs** and the **encoder_lengths** without making any prediction.
  - **decoder RNN: (RED on the right)** processes either the **decoder_inputs** and the **decoder_lengths** (*training*) or the **greedly decoded tokens** (*inference*). Its prediction is used by the **optimization module**.



The RNN models, however, differ in terms of:

- **directionality** – unidirectional (**tf.nn.dynamic_rnn**) or bidirectional (**tf.nn.bidirectional_dynamic_rnn**);
- **depth** – single or multi-layer (**tf.contrib.rnn.MultiRNNCell**);
- **type**– often either a vanilla RNN, LSTM or GRU.

Outline

- Vanilla Sequence to Sequence models
- **Attention based Sequence to Sequence models**
- Hierarchical Attention Networks for question answering

Attention - General

Considering the sequential dataset: $\{((x_1, \ldots, x_n), (y_1, \ldots, y_m))\}_{i=1}^{N}$

The decoder role is to model the generative probability: $P(y_1, \ldots, y_m | x)$

In **vanilla seq2seq** models, the decoder is conditioned by initializing its initial state with **the last source state of the encoder**.

This works well for short and medium-length sentences; however, **for long sentences**, **this single fixed-size hidden state becomes an information bottleneck**, resulting in a performance degradation.

Instead of discarding all of the **hidden states** computed by the encoder, the **attention mechanism** (Bahdanau et al., 2015, Luong et al., 2015) allows the decoder to **use them as a dynamic memory of the source information**, improving the performance for long sentences.

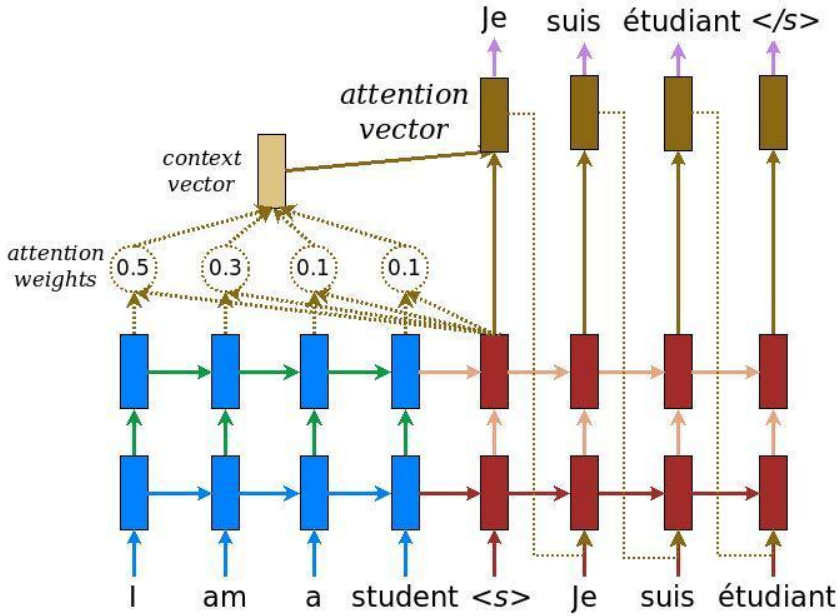
Attention - Computation

In general an **attention function** can be described as **mapping a query and a set of key-value pairs to an output**.

The output is computed as **a weighted sum of the values**, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

1) Compare the current **target hidden state h_t**, with **all the source states h_s** to derive **attention scores [source_length]**.

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



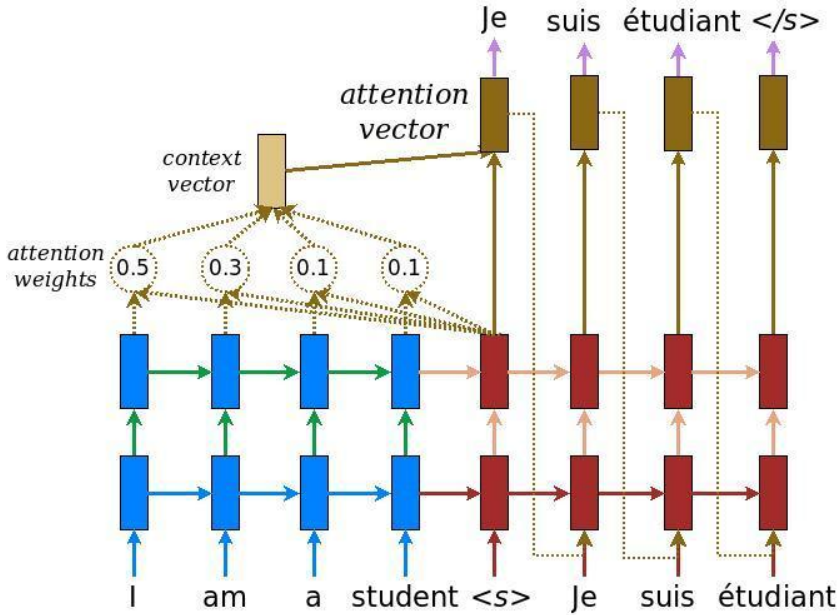
Attention - Computation

2) Apply the softmax function on the **attention scores** and compute the **attention weights**, one for each encoder token **[source_length]**:

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

3) Compute the context vector as **the weighted average** of the **source states [d_model]:**

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

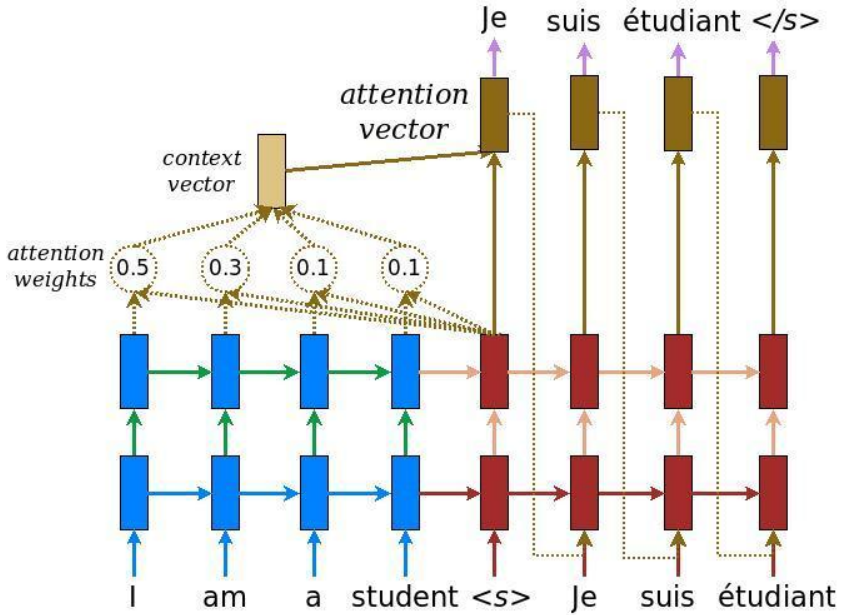


Attention - Computation

4) Combine the **context vector** with the **current target hidden state** to yield the final **attention vector [d_model].**

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

The attention vector is then projected on the target vocabulary and, if we are in **inference greedy mode**, it is then fed as an input to the next time step (**input feeding**).



Attention - Visualization

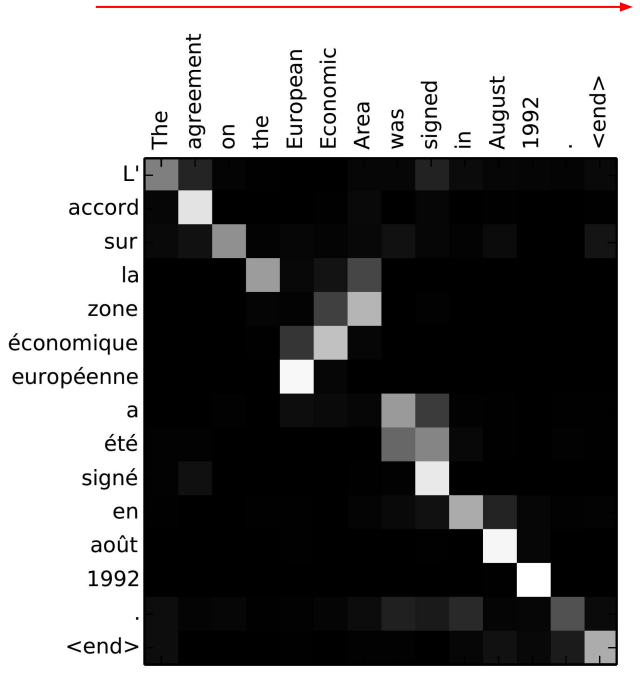
A nice **visualization** of the attention mechanism is the **alignment matrix.**

It plots the **learned attention_weights** between the source (x-axis) and target (y-axis) sentences.

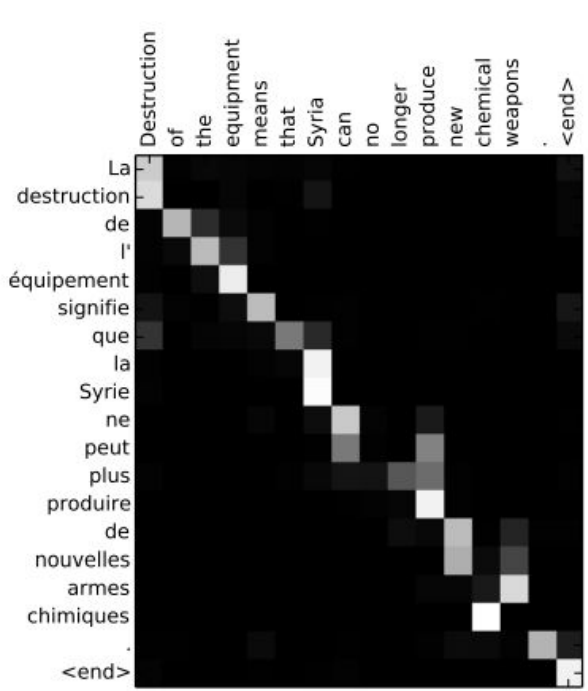
The alignments go from 0 to 1, and in the picture they are **visualized** in a **grey-scale**.

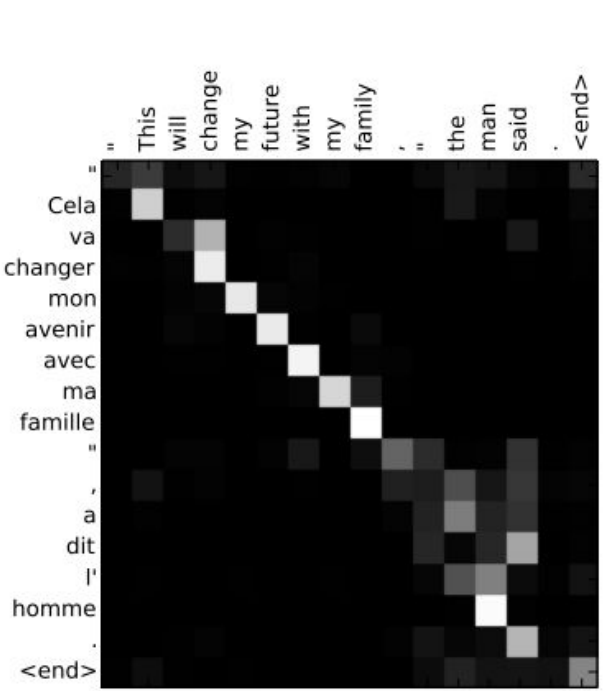
This way we can understand, for each **decoding step** and so for each **generated target token**, which are the **source tokens** that are more present in the **weighted sum** that **conditioned the decoding**.

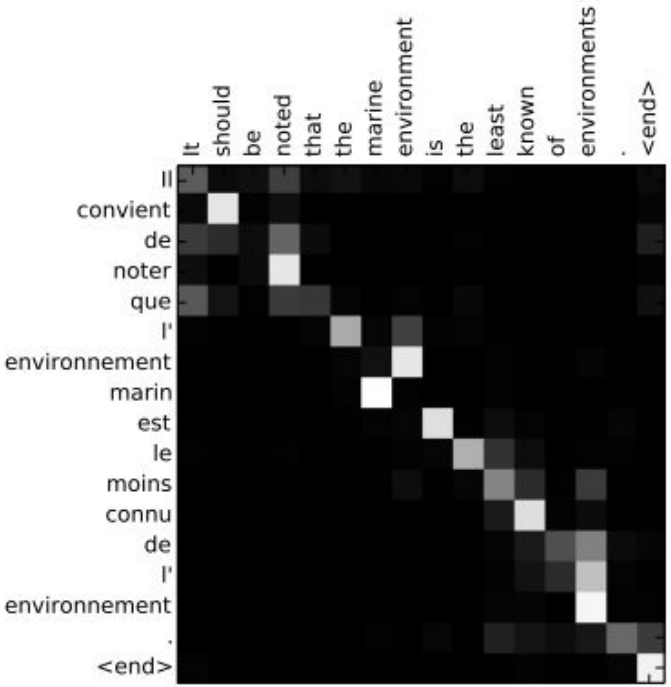
We can see **attention** as a **tool** in the network's bag that, while decoding, allows it to **pay attention** on different parts of the source sentence.



Attention - Visualization







Outline

- Vanilla Sequence to Sequence models
- Attention based Sequence to Sequence models
- **Hierarchical Attention Networks for question answering**

Response Generation - Core algorithm

Chatbots can be defined along at least two dimensions, core algorithm and context handling:

- **Core algorithm**:

  a) Generative ones encode the question into a context vector and generate the answer word by word using **learnt conditioned probability distribution over the answer's vocabulary**. This model can be for example an **encoder-decoder model**.

  b) Retrieval ones rely on a **knowledge base of question-answer pairs**. When a new question comes in, the inference phase encodes it in a context vector and by using a **similarity measure** it retrieves the top-k neighbor knowledge base items. Using a **policy** they return the answer related to one of the k retrieved question-answer pair.

Response Generation - Context

- **Context handling**:

  a) Single-turn models build the **input vector** by only considering the **incoming question**. They may lose important information about the **history of the conversation** and generate irrelevant responses.

$$\{(q_i, a_i)\}$$

  b) Multi-turn models their **input vector** is built by considering a **multi-turn conversational context**, containing also the **incoming question**. For example the input can be the **concatenation of the questions and the answers** of the last 2 conversational turns.

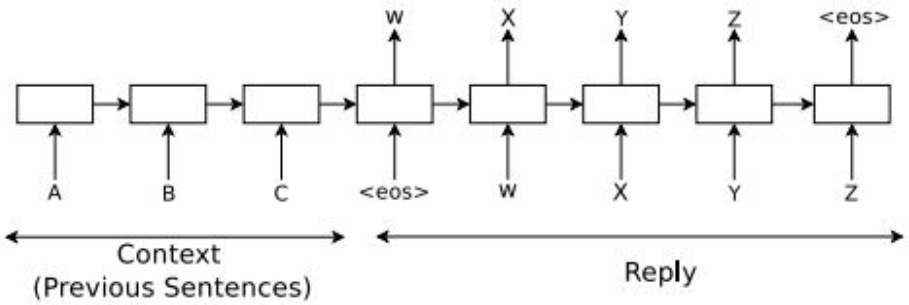
$$\{([q_{i-2}; a_{i-2}; q_{i-1}; a_{i-1}; q_i], a_i)\}$$

Generative chatbots

Considering **generative chatbots**, Vinyals and Le, 2015 and Shang et al., 2015 proposed to directly apply sequence to sequence models to the conversation between two agents.

Suppose that we observe a conversation with a single turn: the first person utters “ABC”, and the second person replies “WXYZ”.**The idea of generative chatbots is to use an RNN and train it to map** “ABC” to “WXYZ”.

The strength of this flat model lies in its **simplicity and generality**. We can borrow it from machine translation without major changes in the architecture.



Generative hierarchical chatbots

If we wanted to extend this model to consider **multi-turn conversations** we could do it by **concatenating multiple turns** into a **single long input sequence**.

This would probably result in poor performances. In fact LSTM cells often fail to catch the long term dependencies within input sequences that are longer than 100 tokens.

For this reason we should exploit a different architecture to handle the conversational context.

The solution that Xing et al., 2017 proposed to **overcome this problem** is to extend the attention mechanism for single-turn response generation with an **hierarchical attention mechanism for multi-turn response generation**.

Hierarchical attention networks

In general, **attention** can be seen as a technique to generate a dynamic **hidden representation of a sequence** **from** its contextualized word vectors.

In conversational AI and in general In text mining, we handle **text** made of **sentences**, sentences made of **words** and words made of **characters**.

The inner **hierarchical structure** that we can find into text drove the development of **Hierarchical attention networks**.

There are two really promising applications of Hierarchical attention networks:

1) **Response generation** (Xing et al., 2017) - hierarchical conversational context (*covered*)
2) **Document classification** (Yang et al.,2017) - hierarchical document context (*not covered*)

Hierarchical generative multi-turn chabots

$$\mathbf{U} = (u_1, \ldots, u_m)$$

$$u_i = (w_{i,1}, \ldots, w_{i,T_i})$$

$$\mathbf{h}_{i,k} = concat(\overrightarrow{\mathbf{h}}_{i,k}, \overleftarrow{\mathbf{h}}_{i,k}),$$

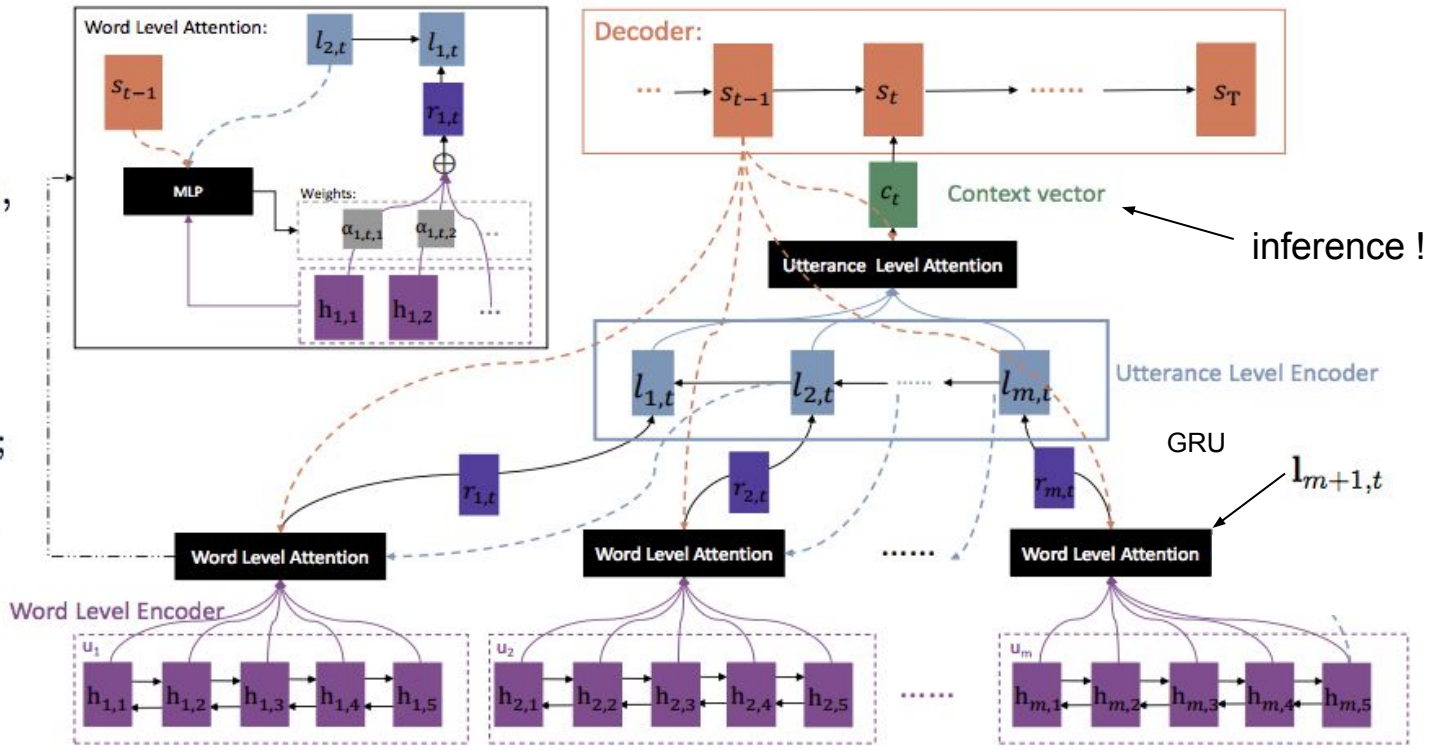
$$\mathbf{r}_{i,t} = \sum_{j=1}^{T_i} \alpha_{i,t,j} \mathbf{h}_{i,j}.$$

$$e_{i,t,j} = \eta(\mathbf{s}_{t-1}, \mathbf{l}_{i+1,t}, \mathbf{h}_{i,j});$$

$$\alpha_{i,t,j} = \frac{exp(e_{i,t,j})}{\sum_{k=1}^{T_i} exp(e_{i,t,k})},$$

$$(\mathbf{l}_{1,t}, \ldots, \mathbf{l}_{m,t})$$

$$\mathbf{c}_t = \sum_{i=1}^{m} \beta_{i,t} \mathbf{l}_{i,t}.$$



Hierarchical generative multi-turn chabots

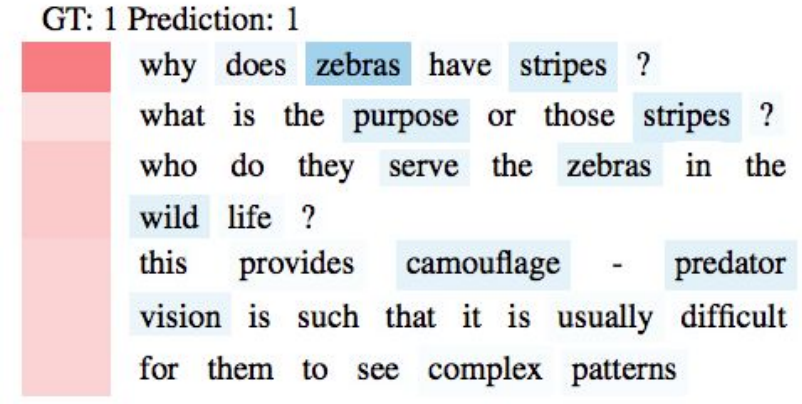
- Nicely, we can ***visualize hierarchical attention weights***. Here darker color mean more important words or utterances.
- We can see how this model is able to understand **which are** the important words and important utterances **needed** to answer a question.

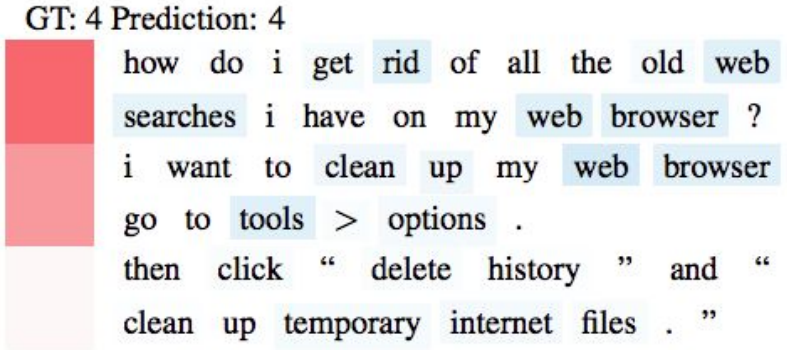
Hierarchical document classification

These are results of a hierarchical attention networks used for ***topic classification*** in the Yahoo Answer data set.

For the left document with **label 1**, which denotes **Science and Mathematics**, the model accurately localizes the words zebra, stripes, camouflage, predator and their corresponding sentences.

For the right document with **label 4**, which denotes **Computers and Internet**, the model focuses on web, searches, browsers and their corresponding sentences.





Hierarchical document classification

The model can select words carrying ***strong sentiment*** like delicious, amazing, terrible and their corresponding sentences. Sentences containing useless words like cocktails, pasta, entree are disregarded.