Attention: Motivation

Let's revisit the Encoder-Decoder architecture

The Encoder

- Acts on input sequence $[\mathbf{x}_{(1)} \ldots \mathbf{x}_{(\bar{T})}]$
- Producing a sequence of latent states $[\bar{\mathbf{h}}_{(1)}, \ldots, \bar{\mathbf{h}}_{(\bar{T})}]$

The Decoder

- Acts on the *final* Encoder latent state $\bar{\mathbf{h}}_{(\bar{T})}$
- Producing a sequence of outputs $[\hat{\mathbf{y}}_{(1)}, \ldots, \hat{\mathbf{y}}_{(T)}]$
- Often feeding step t output $\hat{\mathbf{y}}_{(t)}$ as Encoder input at step $(t+1)$

RNN Encoder/Decoder

The following diagram is a condensed depiction of the process

Sequence to Sequence: training (teacher forcing) + inference: No attention

Recall that $\bar{\mathbf{h}}_{(\bar{t})}$ is a fixed length encoding of the input prefix $\mathbf{x}_{(1)}, \ldots, \mathbf{x}_{(\bar{t})}$.

So $\bar{\mathbf{h}}_{(\bar{T})}$, which initializes the Decoder, is a summary of entire input sequence $\mathbf{x}$.

This fact enables us to decouple the Encoder from the Decoder

- The consumption of input $\mathbf{x}$ and production of output $\hat{\mathbf{y}}$ do not have to be synchronized
- Allowing for the possibility that $T \neq \bar{T}$
- For example
    - There is no one to one mapping between languages (nor does ordering of words get preserved)

Let's focus on the part of the Decoder

- That transforms latent state (or short term memory) $\mathbf{h}_{(t)}$ to output $\hat{\mathbf{y}}_{(t)}$

Decoder: No attention

We can generalize this transformation as

$$\hat{\mathbf{y}}_{(t)} = D(\mathbf{h}_{(t)}; \mathbf{s})$$

In the vanilla RNN, this was governed by the equation

$$\hat{\mathbf{y}}_{(t)} = D(\mathbf{h}_{(t)}; \mathbf{s}) = \mathbf{W}_{hy}\mathbf{h}_{(t)} + \mathbf{b}_y$$

Additional parameter $\mathbf{s}$

- Was unused in this example (our illustration used $\bar{\mathbf{h}}_{(\bar{T})}$ as a place-holder)
- But may be used in other cases

This simple mapping of $\mathbf{h}_{(t)}$ to $\hat{\mathbf{y}}_{(t)}$ can be extremely burdensome

It is often the case that $\hat{\mathbf{y}}_{(t)}$

- Depends mostly on a **specific element** $\mathbf{x}_{(\bar{t})}$ of the input
- Or on a **specific prefix** of the input: $\mathbf{x}_{(1)}, \ldots, \mathbf{x}_{(\bar{t})}$

Consider the example of language translation

- When predicting word $\hat{\mathbf{y}}_{(t)}$ in the Target language
- Some "context" provided by the Source language may greatly influence the prediction
    - For example: gender/plurality of the subject

This context is usually much smaller than the entire sequence $\mathbf{x}$ of length $\bar{T}$.

By not allowing $D(\mathbf{h}_{(t)}; \mathbf{s})$ *direct* access to the required context, we force the Decoder

- To encode the context of the Source
- Along with the specific information of the Target
- Into $\mathbf{h}_{(t)}$

This makes $\mathbf{h}_{(t)}$ unnecessarily complex and perhaps difficult to learn well.

We will introduce a mechanism called *Attention* to alleviate this burden.

To give you a better feel for context, here are some examples

Image captioning example

- Source: Image
- Target: Caption: "A woman is throwing a **frisbee** in a park."
- Attending over *pixels* **not** sequence

Visual attention

A woman is throwing a **frisbee** in a park.

Attribution: [https://arxiv.org/pdf/1502.03044.pdf (https://arxiv.org/pdf/1502.03044.pdf)](https://arxiv.org/pdf/1502.03044.pdf)

Image captioning example

- Source: Image
- Target: Caption: "A giraffe standing in a forest with **trees** in the background."
- Attending over *pixels* **not** sequence

Visual attention

A giraffe standing in a forest with **trees** in the background.

Attribution: https://arxiv.org/pdf/1502.03044.pdf (https://arxiv.org/pdf/1502.03044.pdf)

Date normalization example

- Source: Dates in free-form: "Saturday 09 May 2018"
- Target: Dates in normalized form: "2018-05-09"

[link (https://github.com/datalogue/keras-attention#example-visualizations)](https://github.com/datalogue/keras-attention#example-visualizations)

Attend to what's important

The solution to over-loading $\mathbf{h}_{(t)}$ with Source context is conceptually straight forward.

In the Decoder expression $D(\mathbf{h}_{(t)}; \mathbf{s})$, let

$$\mathbf{s} = \mathbf{c}_{(t)}$$

where $\mathbf{c}_{(t)}$ is a variable

- That supplies the appropriate context for output $\hat{\mathbf{y}}_{(t)}$
- Conditional on $\mathbf{h}_{(t)}$

Because $\bar{\mathbf{h}}_{(\bar{t})}$

- Is a fixed length encoding of the input prefix $\mathbf{x}_{(1)}, \ldots, \mathbf{x}_{(\bar{t})}$
- It can be assigned to $\mathbf{c}_{(t)}$ as the context for the prefix of $\mathbf{x}$ of length $\bar{t}$

$$\mathbf{c}_{(t)} \in \{\bar{\mathbf{h}}_{(1)}, \ldots, \bar{\mathbf{h}}_{(\bar{T})}\}$$

We say

- The Decoder "attends to" (pays attention) $\bar{\mathbf{h}}_{(\bar{t})}$
- When generating output $\hat{\mathbf{y}}_{(t)}$

That is: it focuses its attention on a specific part of the input $\mathbf{x}$

Decoder: Attention

The dotted line from $\mathbf{h}_{(t)}$ on the left of the Choose box

- Indicates that the Choice is conditional on Decoder state $\mathbf{h}_{(t)}$

Here is a diagram summarizing the Attention mechanism

Sequence to Sequence: attention

How is the choice of $\mathbf{c}_{(t)}$ from the set $\{\bar{\mathbf{h}}_{(1)}, \ldots, \bar{\mathbf{h}}_{(\bar{T})}\}$ accomplished ?

The "Choose" box

- Is a Neural Network
- With it's own weights
- That learn to make the best choice for the Target task !
    - It is trained as part of the larger task

The "Choose" box is implementing Attention and is called an Attention *head*

Why is Attention so important ?

Let's illustrate with a hypothetical example from Natural Language Processing: Question Answering.

A training example is encoded as

- Features: context + question
- Target: Answer

$$\mathbf{x} = \begin{cases} \text{Context:} & \text{The FRE Dept offers many Spring classes. The students ar} \\ & \vdots \\ & \text{Professor Perry taught them Machine Learning. The stu} \\ & \vdots \\ & \text{Professor Blecherman led a class in ...} \\ & \vdots \\ \text{Question:} & \text{What did Professor Perry do ?} \end{cases}$$

$$\mathbf{y} = \text{Answer:} \quad \text{He taught them Machine Learning}$$

Perhaps, after seeing many such examples, the NN "learns" a pattern for answering questions of the type

What did Professor <PROPER NOUN> teach in the Spring ?

Pattern:

```
<PRONOUN> <VERB> <INIDRECT OBJECT> <OBJECT>
```

The question can then be answered by extracting the values of the place holders in the pattern.

Answering questions using Attention

Notice that, by using Attention

- The latent state of the NN **does not** need to include the details of the context
- It can instead learn a pattern (useful for **many** similar questions)
- And use Attention Lookup to fill in values for the place holders

The weights of the model thus generalize higher level concepts, rather than low level details.

- Allows bigger contexts

Furthermore, this facilitates achieving long range dependencies

- No computational/space overhead in back propagation
- No vanishing gradients

Attention is one of the main contributors powering recent advances in Deep Learning

- particularly Natural Language Processing

Multi-head attention: two heads are better than one

Remember:

- The elements of the output sequences are *vectors*: have multiple features

We may need to attend to a different Encoder latent state for different output features

- May even need to attend to multiple Encoder latent states for a single output feature

Rather than having a single "head" attending to the latent states, we can have many.

A "head" is similar to the channel dimension of a CNN

- Each head (resp., channel) implements the same computation
- Using per-head (resp., per channel) weights
- Each computing a separate feature

The picture shows n Attention heads.

Each head j uniquely transforms the query $\mathbf{h}_{(t)}$ and the key/value pairs $\bar{\mathbf{h}}_{(1)} \ldots \bar{\mathbf{h}}_{(\bar{T})}$ being queried.

- into $\mathbf{h}_{(t)}^{(j)}$ and the key/value pairs $\bar{\mathbf{h}}_{(1)}^{(j)} \ldots \bar{\mathbf{h}}_{(\bar{T})}^{(j)}$
- Such that each head attends to a separate item

Head j

- uses query $\mathbf{h}^{(j)} = \mathbf{h} * \mathbf{W}_{\text{query}}^{(j)}$
- against keys/values $\bar{\mathbf{h}}^{(j)} = \bar{\mathbf{h}} * \mathbf{W}_{\text{value}}^{(j)}$

The weights matrices $\mathbf{W}_{\text{query}}^{(j)}, \mathbf{W}_{\text{value}}^{(j)}$ that transform queries and key/value pairs are *learned* during training

In practice: $\mathbf{W}_{\text{query}}^{(j)}, \mathbf{W}_{\text{value}}^{(j)}$

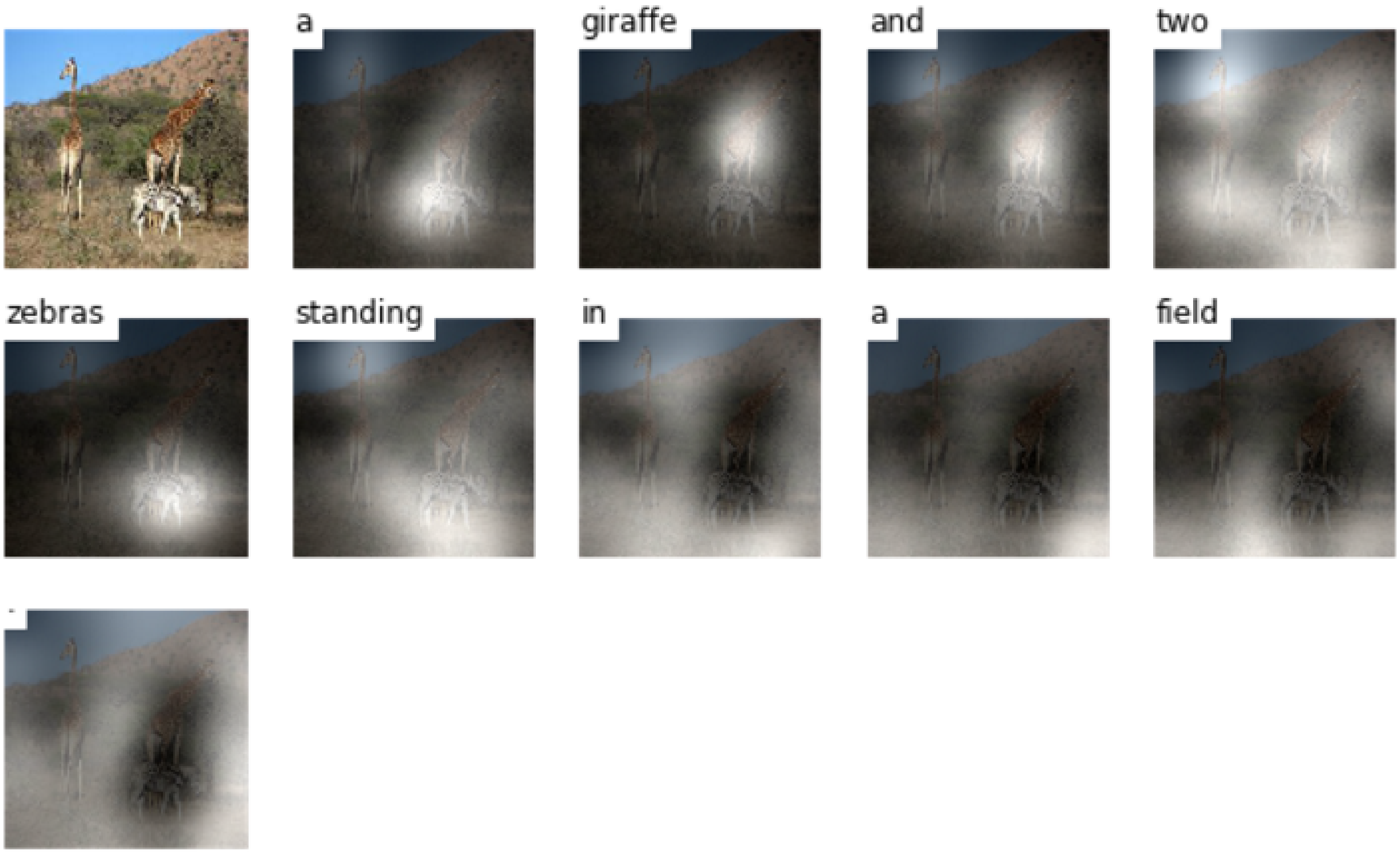
- reduce the length of $\mathbf{h}, \bar{\mathbf{h}}$
- such that the concatenated length is the same as the length of $\mathbf{h}, \bar{\mathbf{h}}$

Just for fun: Attention in action

Here are some examples of Sequence to Sequence problems using Attention.

Visual Attention example

- Source: Image
- Target: Caption: "A giraffe and two zebras standing in a field."
- Attending over *pixels* **not** sequence



Language Translation example

- Source: Spanish
- Target: English
- Colab notebook ! [Translation example (https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tut

Self-attention

We have illustrated Attention in the context of the Decoder attending to an Encoder.

But Attention may be used to relate one element of the *input* sequence to all other elements of the input sequence.

This is called *self-attention*

To illustrate, suppose we want to generate an embedding of words that is context sensitive.

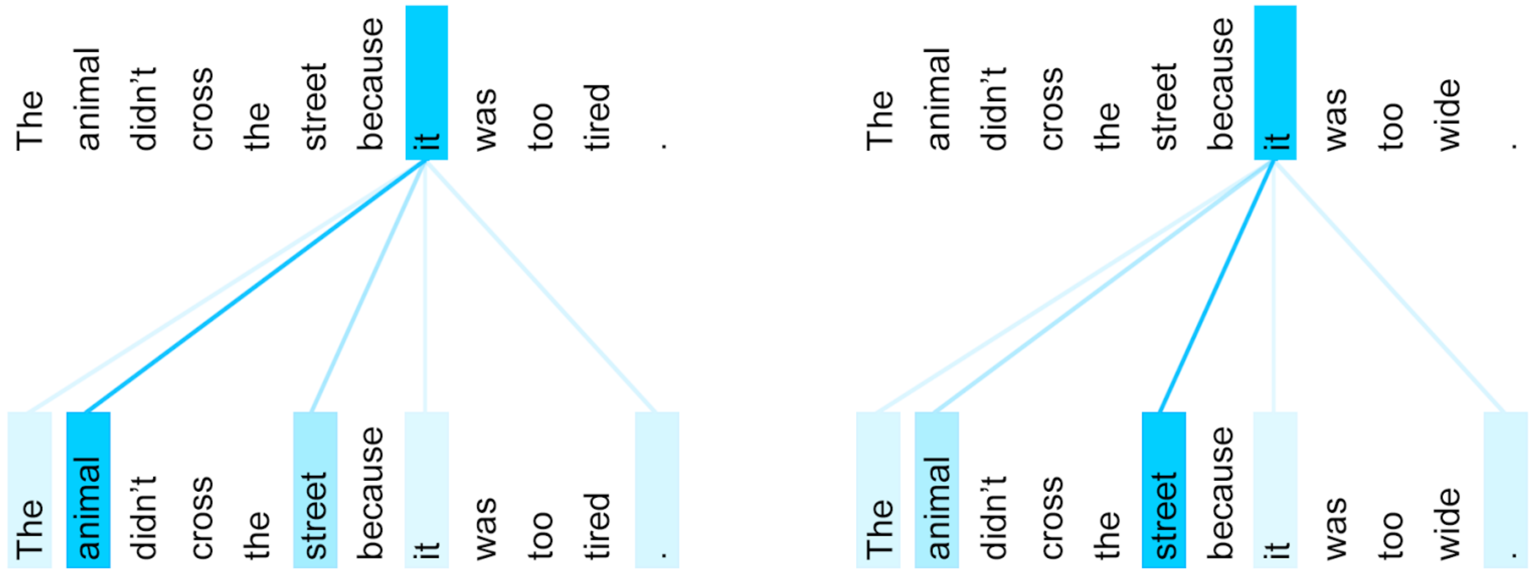
Consider

- "The animal didn't cross the street because **it** was too *tired*"
- "The animal didn't cross the street because **it** was too *wide*"

The meaning of the word "it" in each sentence depends on the context.

By using a model for word embeddings that uses self-attention we can differentiate between the two.

The thickness of the blue line indicates the attention weight that is given in processing the word "it".



Much of the recent advances in NLP may be attributed to these improved, context sensitive embeddings.

Masked self-attention

Self attention is applied to the *entire* input sequence to determine on which elements to focus.

It is almost as if the sequence $\mathbf{x}$ is treated as an *unordered* set.

Sometimes order is important.

For example, consider a generative model where

$$\mathbf{x}_{(t+1)} = \mathbf{y}_{(t)}$$

- That is: input element $(t+1)$ is the t^{th} output
- Can't attend to something that hasn't been generated yet !
- Causal ordering is important

Other times, the fact that $\mathbf{x}_{(t)}$ precedes $\mathbf{x}_{(t+1)}$ is important.

The solution to both problems is to pair $\mathbf{x}_{(t)}$ with a *positional encoding* (of t)

To implement causal ordering for output t

- mask out all $\mathbf{x}_{(t')}$ where $t' > t$

This is called *masked self-attention*

The positional encoding can also be used in problem domains where relative order is important.

- The encoding is *non-trivial*

Transformers

There is a new model (the Transformer) that processes sequences much faster than RNN's.

It is an Encoder/Decoder architecture that uses multiple forms of Attention

- Self Attention in the Encoder
    - to tell the Encoder the relevant parts of the input sequence $\mathbf{x}$ to attend to
- Decoder/Encoder attention
    - to tell the Decoder which Encoder state $\bar{\mathbf{h}}_{(t')}$ to attend to when outputting $\mathbf{y}_{(t)}$
- Masked Self-Attention in the Decoder
    - to prevent the Decoder from looking ahead into inputs that have not yet been generated

Conclusion

We recognized that the Decoder function responsible for generating Decoder output $\hat{\mathbf{y}}_{(t)}$

$$\hat{\mathbf{y}}_{(t)} = D(\mathbf{h}_{(t)}; \mathbf{s})$$

was quite rigid when it ignored argument $\mathbf{s}$.

This rigidity forced Decoder latent state $\mathbf{h}_{(t)}$ to assume the additional responsibility of including Encoder context.

Attention was presented as a way to obtain Encoder context through argument $\mathbf{s}$.

In [2]:

```
print("Done")
```

Done