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)](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 !

In other words

- It is trained as part of the larger task

This is a common technique in Deep Learning that may, at first, appear magical

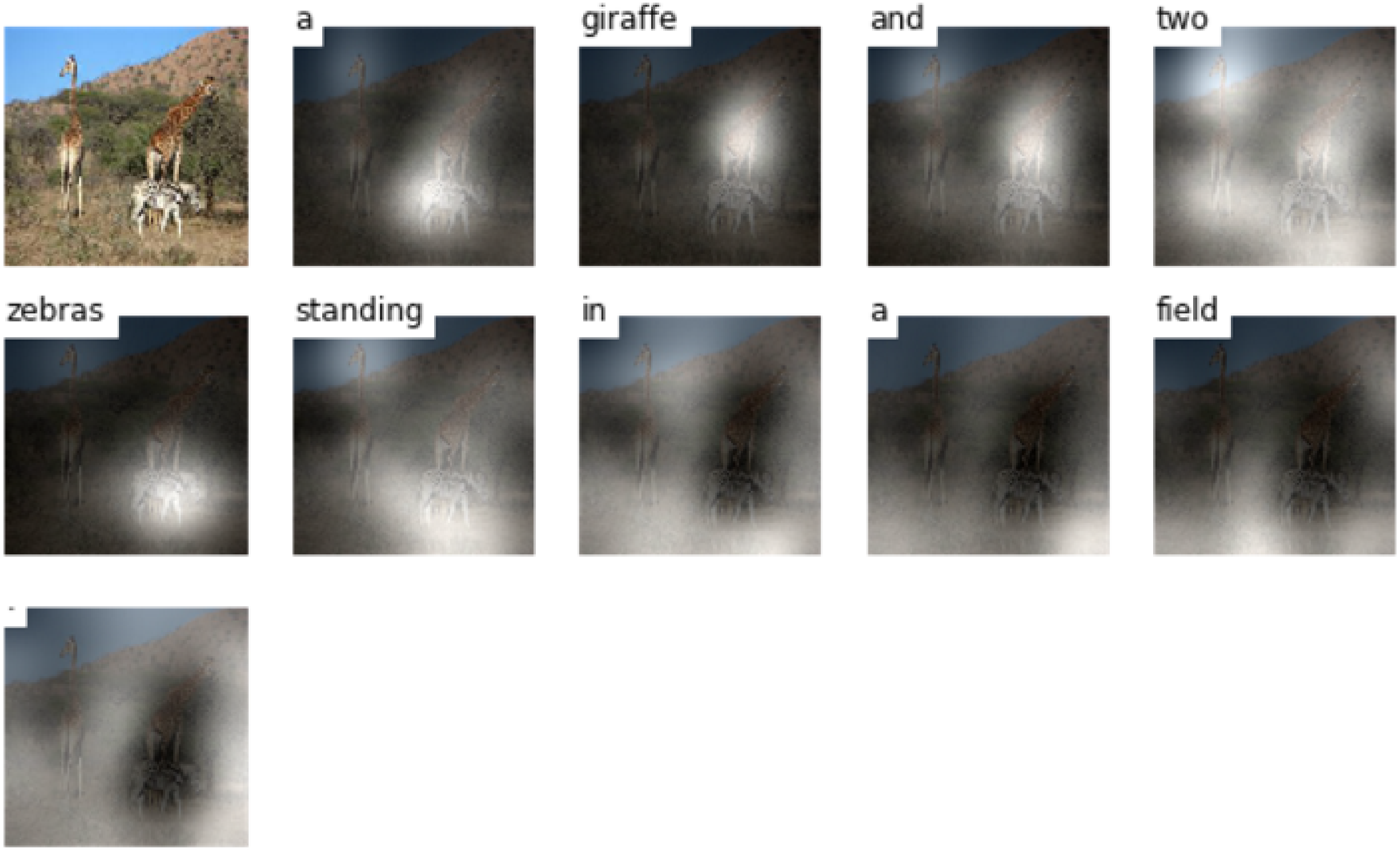
- Hypothesize the existence of a mechanism to solve your problem
- Train a Neural Network to conjure up the mechanism !

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

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
```