Attention: Motivation

The models that we studied for processing input sequences differed from models for non-sequence inputs

- **memory** (latent state) required for processing sequences
    - because sequence length is unbounded
    - finite representation of unbounded length input sequence
    - output at step t fed as input to step $t+1$

The use of latent state/memory evolved over the models we studied

- RNN
    - latent state encodes
        - input representation
        - "control" state
            - guiding how the model processes the data: state transitions
- LSTM
    - latent state partitioned into
        - Short Term memory: control state
        - Long Term memory

Both these models processed the input sequence **once**

- so input-specific representation needs to be part of memory

We will introduce a mechanism called *Attention*

- that allows the input sequence to be *re-visited* at each time step
- cleaner separation between control memory and input memory

Let's revisit the Encoder-Decoder architecture

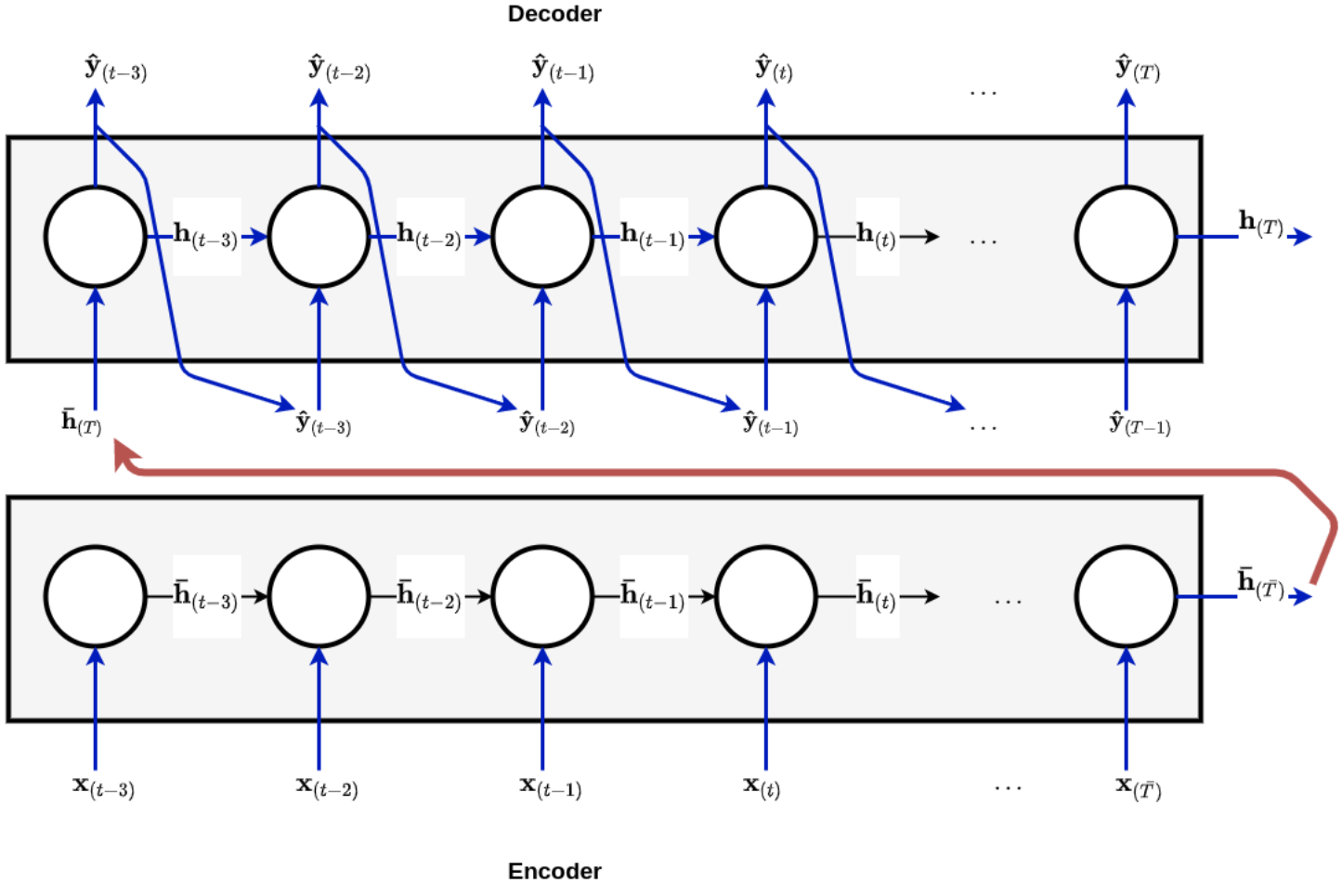
The Encoder

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

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 $(t+1)$

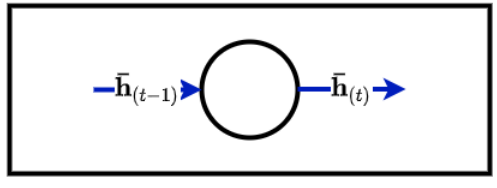
RNN Encoder/Decoder



The following diagram is a condensed depiction of the process

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

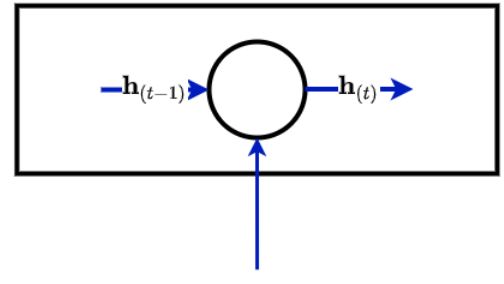
Encoder



$\mathbf{x}_{(1)}\mathbf{x}_{(2)} \quad \dots \quad \mathbf{x}_{(\bar{T})}$

$$\mathbf{h}_{(0)} = \bar{\mathbf{h}}_{(\bar{T})}$$
$$\mathbf{s} = \bar{\mathbf{h}}_{(\bar{T})}$$

Decoder



$\langle \text{START} \rangle, \hat{\mathbf{y}}_{(1)}, \hat{\mathbf{y}}_{(2)}, \dots, \hat{\mathbf{y}}_{(T)}$

Inference

$\langle \text{START} \rangle, \mathbf{y}_{(1)}, \mathbf{y}_{(2)}, \dots, \mathbf{y}_{(\bar{T})}$

Training

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

For example:

$$\mathbf{x}_0, \ldots, \mathbf{x}_{\bar{T}} = \text{Machine, learning, is, easy, not, hard}$$

$$\begin{aligned}
\bar{\mathbf{h}}_{(0)} &= \text{summary}([\text{Machine}]) \\
\bar{\mathbf{h}}_{(1)} &= \text{summary}([\text{Machine, Learning}]) \\
&\vdots \\
\bar{\mathbf{h}}_{\bar{t}} &= \text{summary}([\mathbf{x}_{(0)}, \ldots \mathbf{x}_{(\bar{t})}]) \\
&\vdots \\
\bar{\mathbf{h}}_{(5)} &= \text{summary}([\text{Machine, Learning, is, easy, not, hard}])
\end{aligned}$$

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

Allowing the Encoder to complete its task before the Decoder starts enables us to decouple the two

- 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)

But all Decoder outputs other than $\bar{\mathbf{h}}_{(\bar{T})}$ are discarded.

Let's focus on the part of the Decoder that transforms Decoder latent state (or short term memory) $\mathbf{h}_{(t)}$ to output $\hat{\mathbf{y}}_{(t)}$.

The box in the diagram below is a Neural Network implementing a function

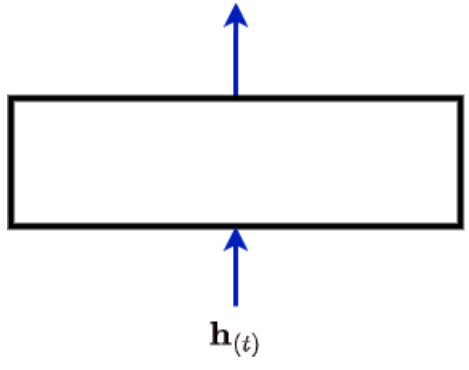
$$D(\mathbf{h}_{(t)})$$

mapping

- the Decoder short term memory $\mathbf{h}_{(t)}$
- to next output $\hat{\mathbf{y}}_{(t)}$.

Decoder

$\hat{\mathbf{y}}_{(1)}, \hat{\mathbf{y}}_{(2)}, \cdots, \hat{\mathbf{y}}_{(T)}$



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

$\bar{\mathbf{h}}_{(1)}, \bar{\mathbf{h}}_{(2)}, \ldots, \bar{\mathbf{h}}_{(\bar{T})}$

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

- the full semantics of the input sequence $\mathbf{x}_{(1)}, \ldots, \mathbf{x}_{(\bar{t})}$
- is only available to the Decoder via the final Encoder representation $\bar{\mathbf{h}}_{(\bar{T})}$
- which must be captured in Decoder latent state $\mathbf{h}_{(t)}$
- since $\bar{\mathbf{h}}_{(\bar{T})}$ is only available to the Decoder on the **first** step of the Decoder

It is often the case that the output $\hat{\mathbf{y}}_{(t)}$ at position 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 $\bar{T}$.

But the Neural Network for D has no mechanism for referring back to specific input positions

- The only part of the Encoder output sequence retained is $\bar{\mathbf{h}}_{(\bar{T})}$
- And that was "absorbed" into Decoder latent state on the first Decoder time step

Attention is a mechanism

- that allows the Decoder to "focus on" ("attend to")
- the *part* of the input sequence $\mathbf{x}$
- that is *most relevant* to the generation of $\hat{\mathbf{y}}_{(t)}$

This is done by conditioning the output Neural Network $D(\mathbf{h}_{(t)}; \mathbf{s})$ on a variable $\mathbf{s}$

- where $\mathbf{s} \in$
$\{\bar{\mathbf{h}}_{(t')}$
$\mid 1$
$\leq t'$
$\leq \bar{T}$
$\}$

That is, Attention allows the Neural Network creating the output $\hat{\mathbf{y}}_{(t)}$ at position t to

- focus ("**attend to**") on
- position t' of the input
- through representation $\bar{\mathbf{h}}_{(t')}$, which summarizes input $\mathbf{x}_{(1)}, \ldots \mathbf{x}_{(t')}$
- that is **most relevant** part of the input for creating output position t.

This potentially greatly simplifies the Decoder latent state $\mathbf{h}_{(t)}$.

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

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

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

Pattern:

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

where <PRONOUN>, <VERB> , etc. are *pattern place-holders*

And perhaps the Encoder "learns" to bind concrete values to place-holders

Answering questions using Attention

Input Tokens: Professor Perry taught them Machine Learning [CLS]

$$\bar{\mathbf{h}}_{(1)}, \ldots, \bar{\mathbf{h}}_{(\bar{T})} = \left\{ \begin{array}{l} \text{Subject: Professor Perry} \\ \text{Pronoun: he} \\ \text{Object: Machine Learning} \\ \text{Indirect Object: them} \\ \text{Verb: taught} \end{array} \right\}$$

Then the control state $\mathbf{h}_{(t)}$ for the Decoder could use Attention to **attend to** the binding for the next place-holder in the output pattern.

- Following the pattern it learned
- Issuing a "query" to lookup the concrete value bound to a place-holder (the "key")
    - for each element of the pattern

Answering questions using Attention

Question: What did Professor Perry do ?

Answer template: <PRONOUN> <VERB> <INDIRECT OBJECT> <OBJECT> [CLS]

Queries:

$\mathbf{h}_{(\mathbf{Pronoun})} = \{$ **Pronoun: ?** $\}$

$\mathbf{h}_{(\mathbf{Verb})} = \{$ **Verb: ?** $\}$

Answer: He taught them Machine Learning [CLS]

Without Attention, the Decoder's control state $\mathbf{h}_{(t)}$

- would have to store the key/value (place-holder/concrete value) associations

With Attention

- the finite number of Decoder weights could be utilized for other purposes.

Visualizing Attention

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

- particularly Natural Language Processing

To give you a better feel for how it's used, here are some visualizations of Attention.

Entailment: Does the "hypothesis" logically follow from the "premise"

Attention: Entailment

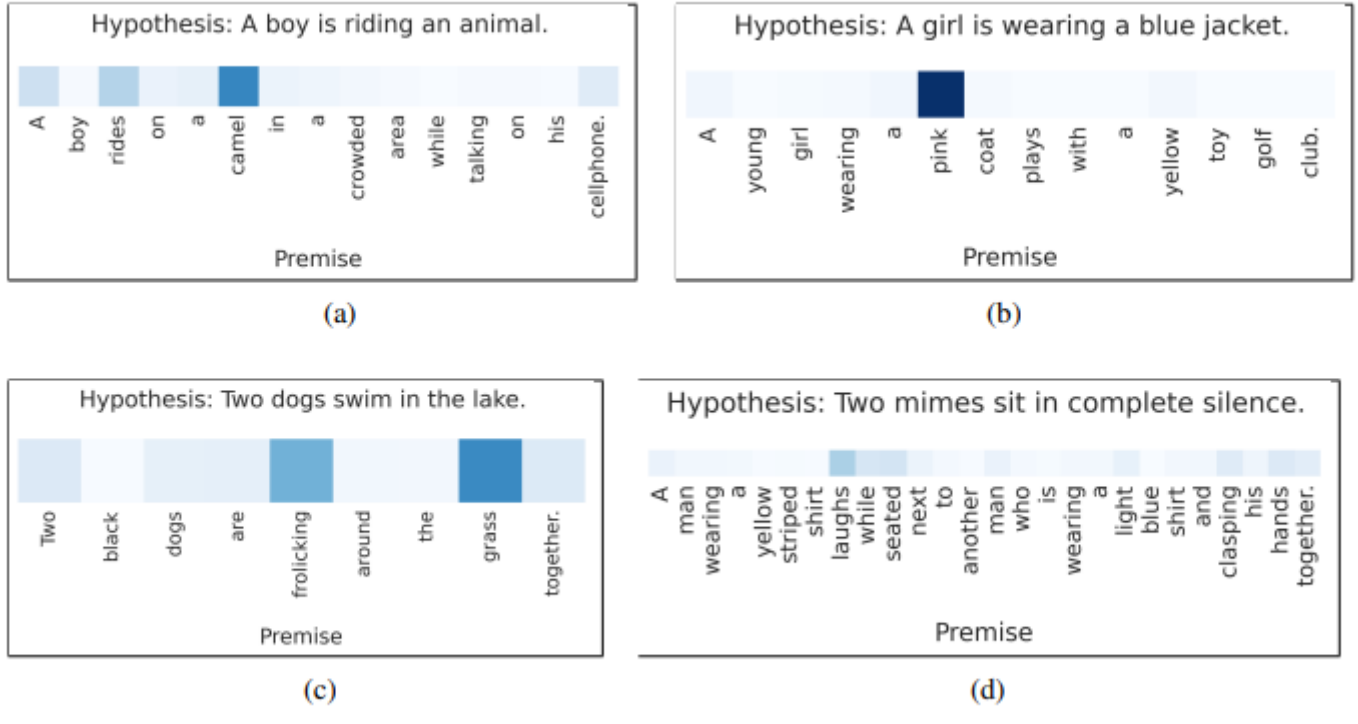


Figure 2: Attention visualizations.

Does the Premise logically entail the Hypothesis.

Attribution: https://arxiv.org/pdf/1509.06664.pdf#page=6 (https://arxiv.org/pdf/1509.06664.pdf#page=6)"

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)

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)

Have we seen this before ?

Similarities to the LSTM

- LSTM separated *short term* (control) memory from *long term* memory
- Created a somewhat complicated mechanism to
    - update/change/forget long term memory
    - move parts of long term memory to the short-term control memory

Differences

- LSTM: attend to $\mathbf{c}$ (long-term memory)
- Attention: attend to *input*
    - not latent state
- Stacked Attention blocks
    - attend to input of *layer*, not raw input of Layer 0

Attend to what's important

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

We condition the Neural Network D on a *context* $\mathbf{s}$

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

and compute the value of the necessary context at each step t

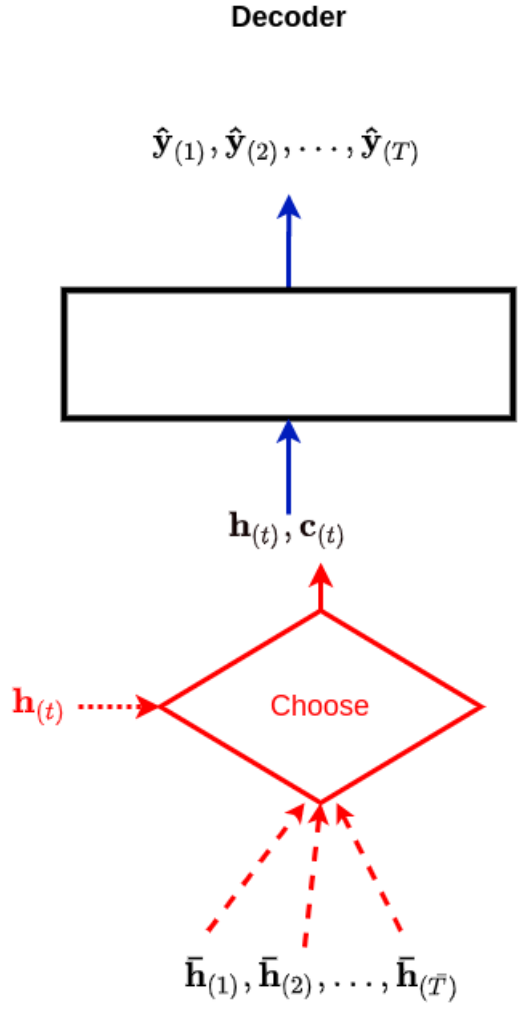
$$\mathbf{s} = \mathbf{c}_{(t)}$$

The context at step t is limited to one of the representations created by the Encoder

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

and is chosen based on the Decoder state $\mathbf{h}_{(t)}$.

Here is the diagram

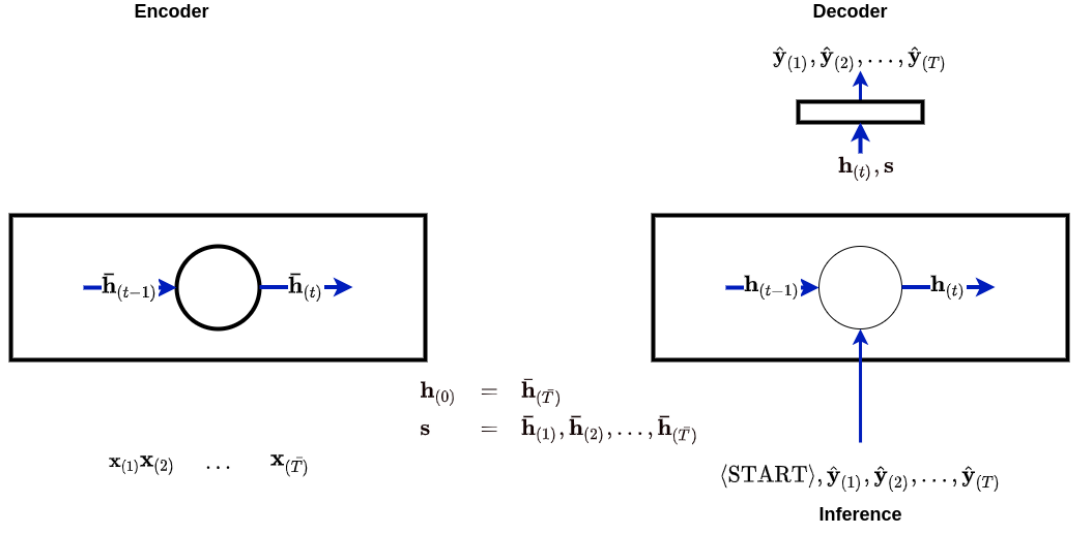

Decoder
$\hat{\mathbf{y}}_{(1)}, \hat{\mathbf{y}}_{(2)}, \cdots, \hat{\mathbf{y}}_{(T)}$
$\mathbf{h}_{(t)}, \mathbf{c}_{(t)}$
$\mathbf{h}_{(t)}$
Choose
$\bar{\mathbf{h}}_{(1)}, \bar{\mathbf{h}}_{(2)}, \ldots, \bar{\mathbf{h}}_{(\bar{T})}$

The "Choose" box is a Context Sensitive Memory (as described in the module on Neural Programming (Neural_Programming.ipynb#Soft-Lookup))

- Like a Python `dict`
    - Collection of key/value pairs: $\langle \bar{\mathbf{h}}_{(\bar{t})}, \bar{\mathbf{h}}_{(\bar{t})} \rangle$
    - Key is equal to value; they are latent states of the Encoder
- But with *soft* lookup
    - The current Decoder state $\mathbf{h}_{(t)}$ is presented to the CSM
        - Called the *query*
        - Is matched across each key of the dict (i.e., a latent state $\bar{\mathbf{h}}_{(\bar{t})}$)
    - The CSM returns an approximate match of the query to a *key* of the `dict`
        - The distance between the query and each key in the CSM is computed
        - The Soft Lookup returns a *weighted* (by inverse distance) sum of the *values* in the CSM `dict`

Here is a diagram summarizing the Attention mechanism

Sequence to Sequence: attention



$\langle \text{START} \rangle, \mathbf{y}_{(1)}, \mathbf{y}_{(2)}, \ldots, \mathbf{y}_{(\bar{T})}$

Training

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

Let d denote the length of each Encoder output (and hence, the latent state sizes too)

- $||\bar{\mathbf{h}}_{(t)}|| = ||\mathbf{h}_{(t)}|| = d$

Since the Encoder outputs are used as the keys and values in the CSM

- d is also the length of keys, values and queries

When we have n heads

- Rather than having one Attention head operating on vectors of length d
    - producing an output of length d (weighted sum of values in the CSM)
- We create n Attention heads operating on vectors (keys, values, queries) of length $\frac{d}{n}$.
    - Output of these smaller heads are values, and hence also of length $\frac{d}{n}$
- The final output concatenates these n outputs into a single output of length d
    - identical in length to the single head
- we project each of these length d vector into vectors of length $\frac{d}{n}$

How do we create the shorter length $\frac{d}{n}$ vectors ?

We use projection matrices of size $\left(d \times \frac{d}{n}\right)$ **for each head** j

- multiplying each key by matrix $\mathbf{W}_{\text{key}}^{(j)}$
- multiplying each value by matrix $\mathbf{W}_{\text{value}}^{(j)}$
- multiplying the original length d query by matrix $\mathbf{W}_{\text{query}}^{(j)}$

How do we know how to reduce the length d vectors to length $\frac{d}{n}$ for head j ?

We learn project matrices $\mathbf{W}_{\text{key}}^{(j)}$, $\mathbf{W}_{\text{value}}^{(j)}$, $\mathbf{W}_{\text{query}}^{(j)}$ **in training**, for each j

The "Choose" box

- Is a Neural Network
- With it's own weights
- That learns 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**

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.

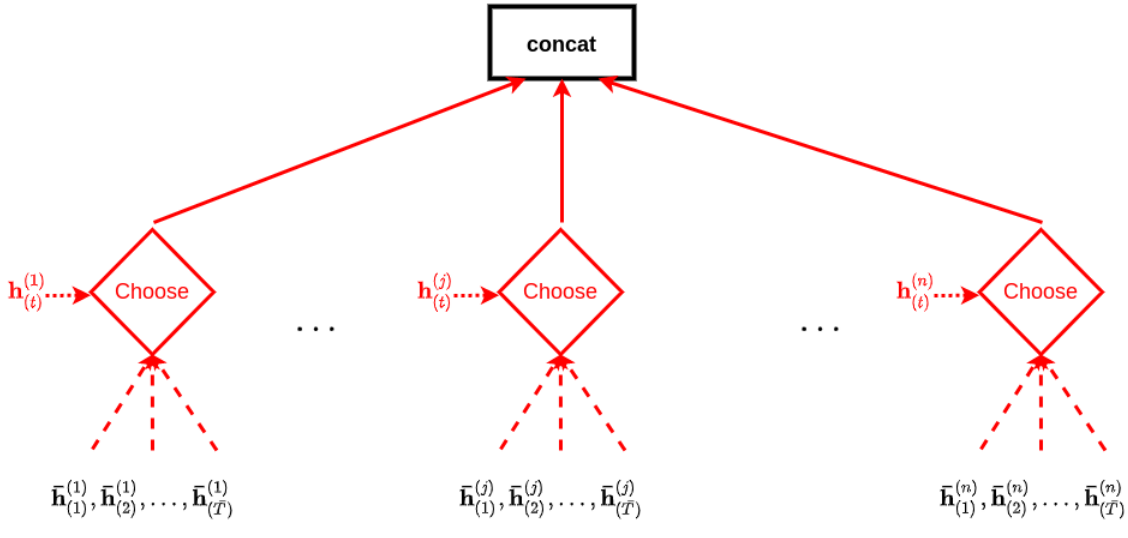
- into $\mathbf{h}_{(t)}^{(j)}$ and the key/value pairs $\bar{\mathbf{h}}_{(1)}^{(j)} \ldots \bar{\mathbf{h}}_{(\bar{T})}^{(j)}$
- Such that each head attends to a separate item

Decoder Multi-head Attention

Per-head query and value

$$\mathbf{h}_{(t)}^{(j)} = \mathbf{W}_{\text{query}}^{(j)} \mathbf{h}_{(t)}$$

$$\bar{\mathbf{h}}_{(t)}^{(j)} = \mathbf{W}_{\text{value}}^{(j)} \bar{\mathbf{h}}_{(t)}$$



Head j

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

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.

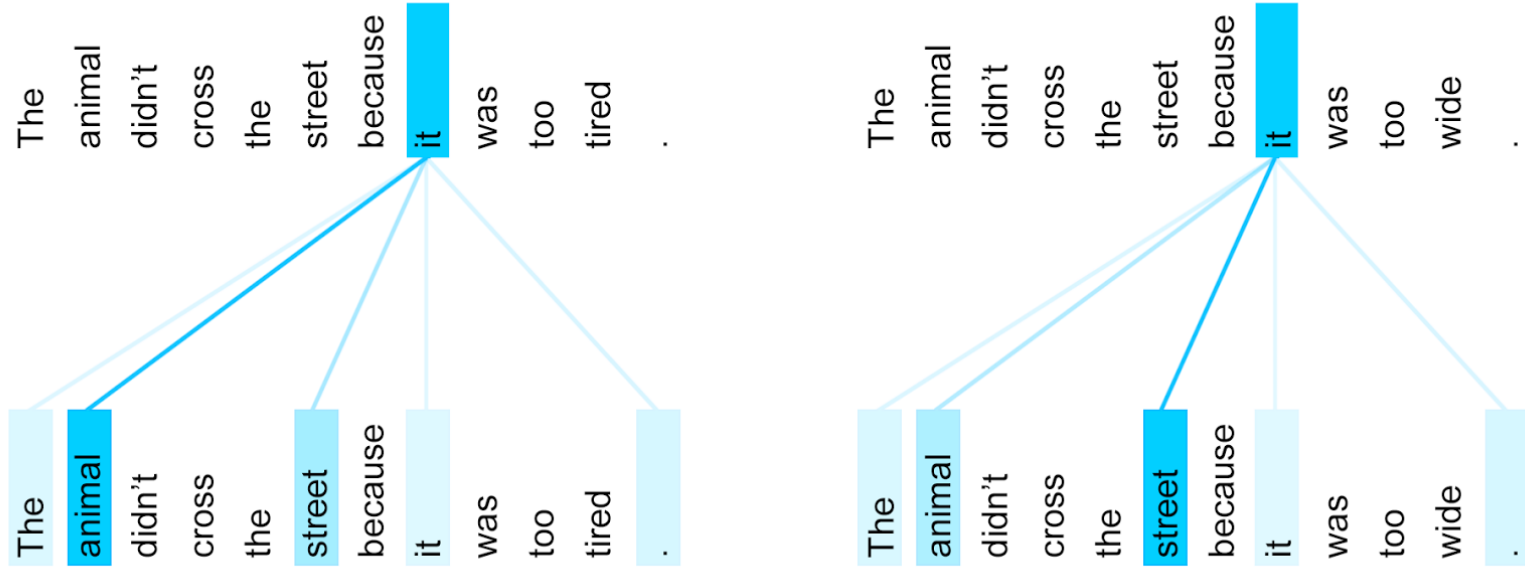
Consider

- "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
```