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

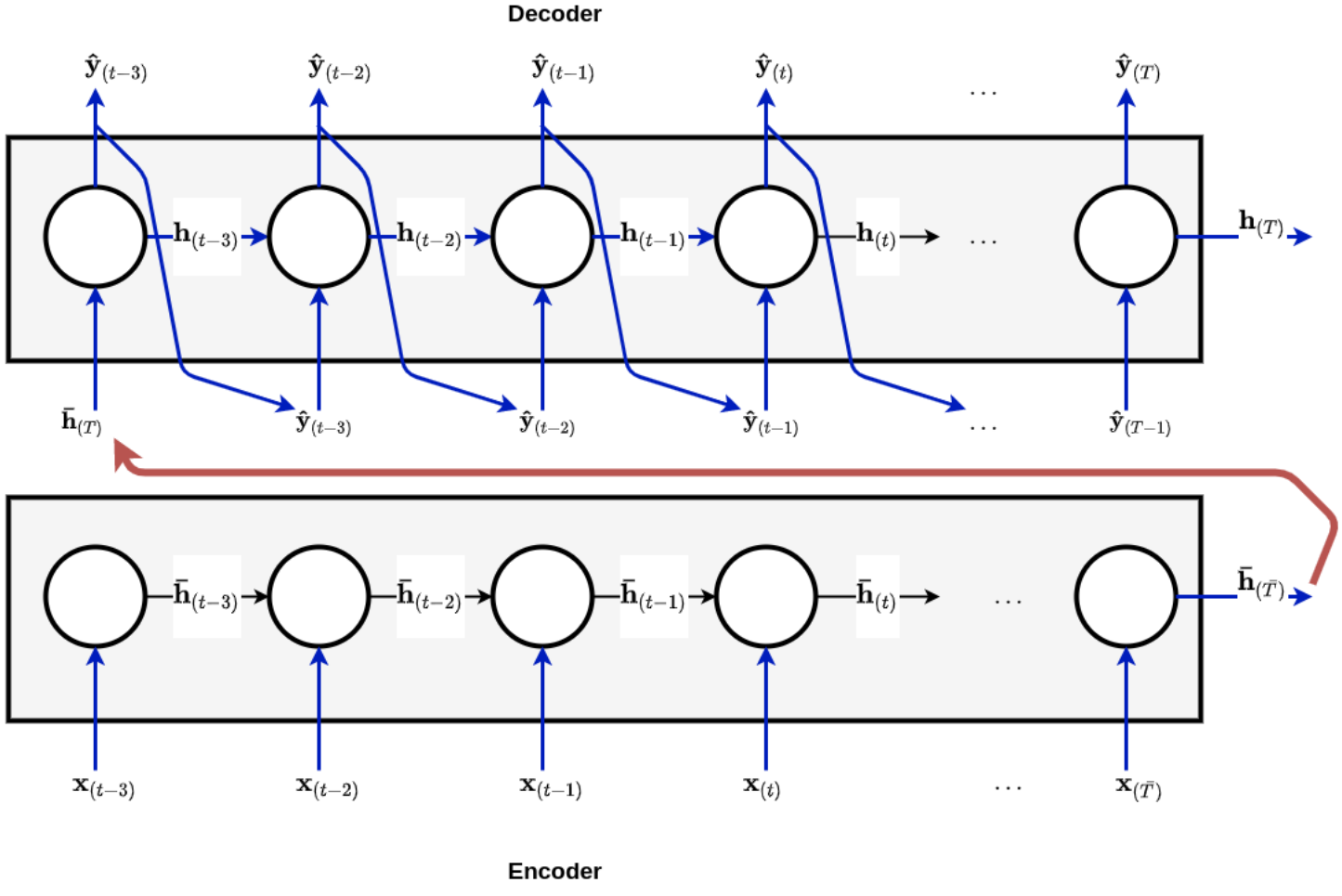
The Encoder

- 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 $(t+1)$

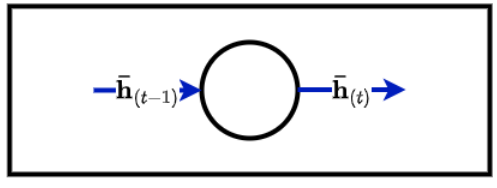
RNN Encoder/Decoder



The following diagram is a condensed depiction of the process

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

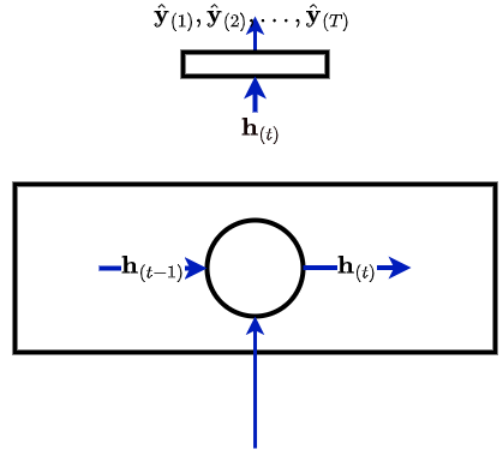
Encoder



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

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

Decoder



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

Inference

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

Training

The topic of "Attention" will focus on the part of the Decoder diagram above that transforms Decoder latent state (or short term memory) $\mathbf{h}_{(t)}$ to output $\hat{\mathbf{y}}_{(t)}$.

It is a Neural Network implementing a function

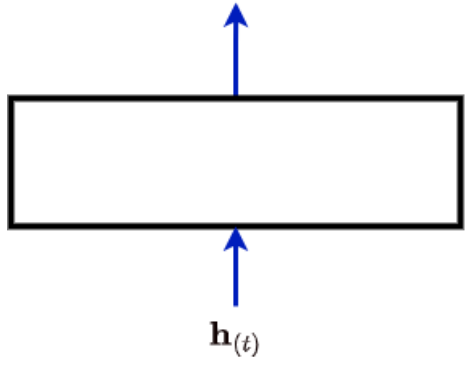
$$\hat{\mathbf{y}}_{(t)} = D(\mathbf{h}_{(t)})$$

mapping

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

Decoder

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



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

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

As an illustrative example: consider the task of Question Answering.

- a sequence to sequence task (thus, ideal for an Encoder-Decoder architecture)
- where the input sequence $\mathbf{x}_{[1:\bar{T}]}$ is the pair consisting of
    - a paragraph called the *context*
    - a question that references the context
- the target/label (i.e., desired output sequence) is $\mathbf{y}_{[1:T]}$
    - is text that "answers" the question

$$
\mathbf{x} = \left\{ \begin{array}{ll}
\text{Context:} & \text{The FRE Dept offers many Spring classes. The students are gr} \\
 & \vdots \\
 & \text{Professor Perry taught them Machine Learning. The studen} \\
 & \vdots \\
 & \text{Professor Blecherman led a class in ...} \\
 & \vdots \\
\text{Question:} & \text{What did Professor Perry do ?}
\end{array} \right.
$$

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

Suppose the Decoder has already output

$$\hat{\mathbf{y}}_{([1:3])} = \text{He taught them}$$

The remainder of the desired output sequence is

$$\hat{\mathbf{y}}_{([4:5])} = \text{Machine Learning}$$

How is possible for the Neural Network implementing D to produce

$$\hat{\mathbf{y}}_{(4)} = D(\mathbf{h}_{(4)})$$

Notice that D is conditioned on the single input $\mathbf{h}_{(t)}$.

Thus, in order for $D(\mathbf{h}_{(4)})$ to be equal to "Machine"

- this information must somehow be encoded in $\mathbf{h}_{(4)}$

But how did it get there ?

It must have been encoded in $\bar{\mathbf{h}}_{(\bar{T})}$

- the "summary" of $\mathbf{x}_{([1:\bar{T}])}$ passed by the Encoder to the Decoder

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:

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

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

In order for the concept "Machine Learning" to have been encoded in $\bar{\mathbf{h}}_{(\bar{T})}$

- it must be present in all Encoder latent states
$$\bar{\mathbf{h}}_{(\bar{t}')} \text{ for } t' \geq p$$
where p is the index of "Machine Learning" in the context.

To summarize

- because D is conditioned *only* on Decoder state $\mathbf{h}_{(t)}$
- all "facts" from the context must be transfered from Encoder to Decoder
- through final Encoder state $\bar{\mathbf{h}}_{(\bar{T})}$
- which in turn was encoded in all Encoder states $\bar{\mathbf{h}}_{(\bar{t}')}$ $\text{for } t' \geq p$

The choice of conditioning D only on $\mathbf{h}_{(t)}$ is burdensome for both the Encoder and Decoder.

some of the weights of each must be devoted to

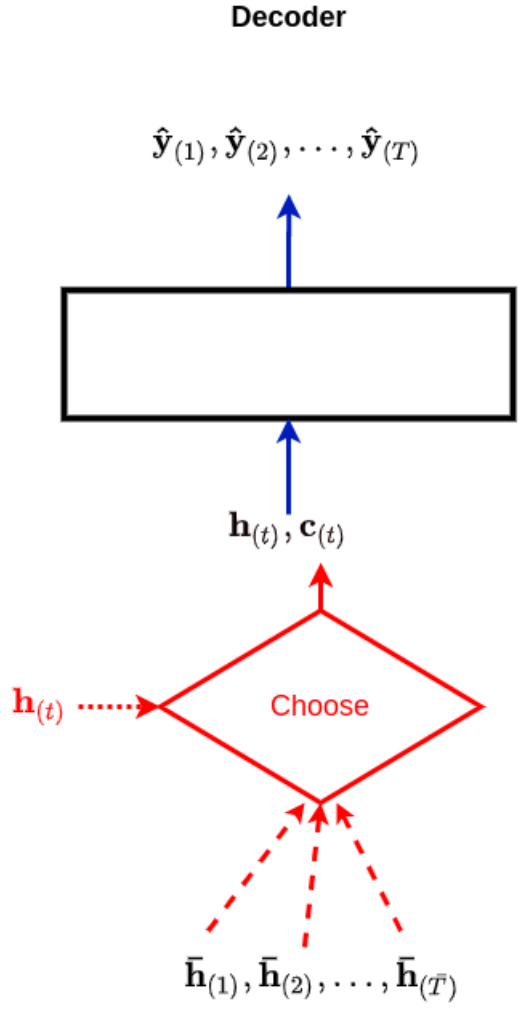
- "control"
    - how to record facts (Encoder)/produce output (Decoder) in the abstract (i.e., for any context/question)
- concrete facts of the particular context

Attend to what's important

What if we changed D so it was conditioned on both $\mathbf{h}_{(t)}$ **and** $\bar{\mathbf{h}}_{([1:\bar{T}])}$

- $\bar{\mathbf{h}}_{([1:\bar{T}])}$ enables the Decoder to refer back to input $\mathbf{x}_{([1:\bar{T}])}$ at *every output position* t
- $\mathbf{h}_{(t)}$ no longer has to encode the "facts"
    - weights can be devoted to "control"
- $\bar{\mathbf{h}}_{(\bar{T})}$ is *no longer the bottleneck* through which facts flow from Encoder to Decoder

The version of D with Attention looks something like this


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

At output position t we enable the Decoder to focus on (*attend to*)

- the position $\bar{t}$ of the input
- that is *relevant* for producing $\hat{\mathbf{y}}_{(t)}$

This seems very natural to a human

- rather than memorizing details
- we refer back to the context
- focusing of only the part that is immediately needed

The discussion of the **implementation** of Attention will be deferred to a later module Attention lookup (Attention_Lookup.ipynb).

For now, think of the "Choose" box as 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`

Visualizing Attention

We can illustrate the behavior of Neural Networks that have been augmented with Attention through diagrams.

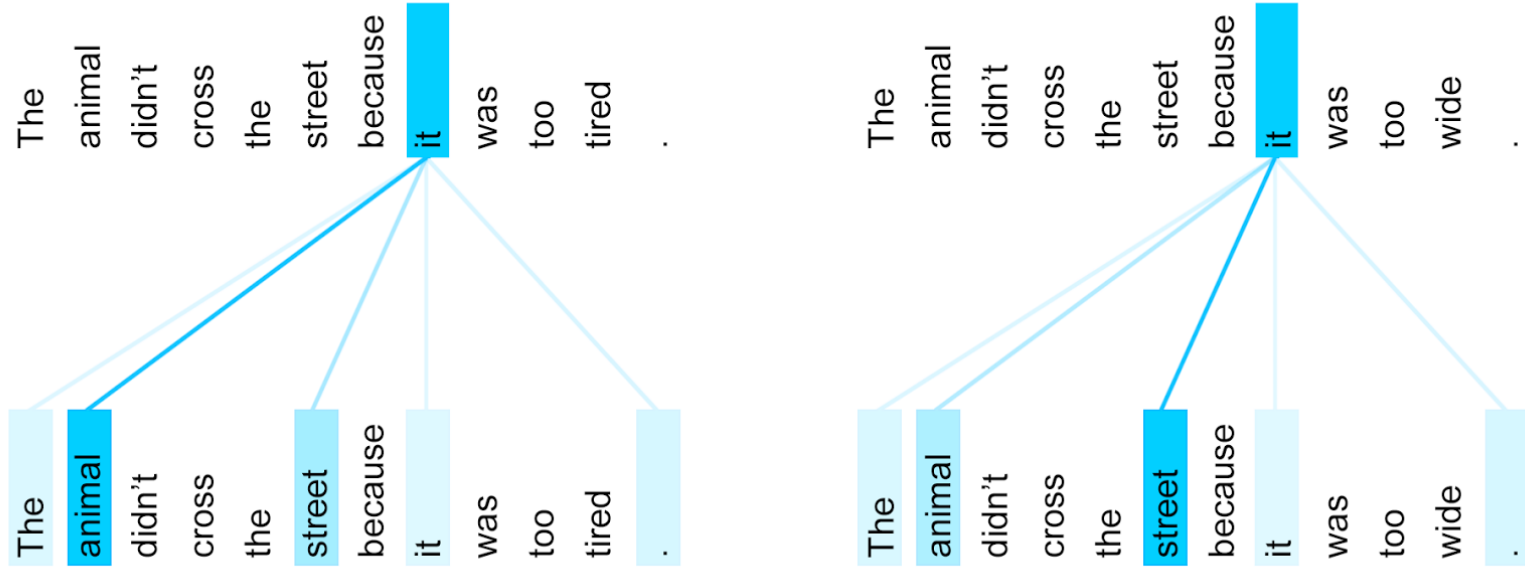
- at a particular output position t
- we can display the amount of "attention"
- that each position in the input receives

Attention can be used to create a Context Sensitive Encoding of words

- The meaning of a word may change depending on the rest of the sentence

We can illustrate this with an example: how the meaning of the word "it" changes

- 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.

We note that simple Word Embeddings

- also capture "meaning"
- but are *not* sensitive to context

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

Attention: Entailment

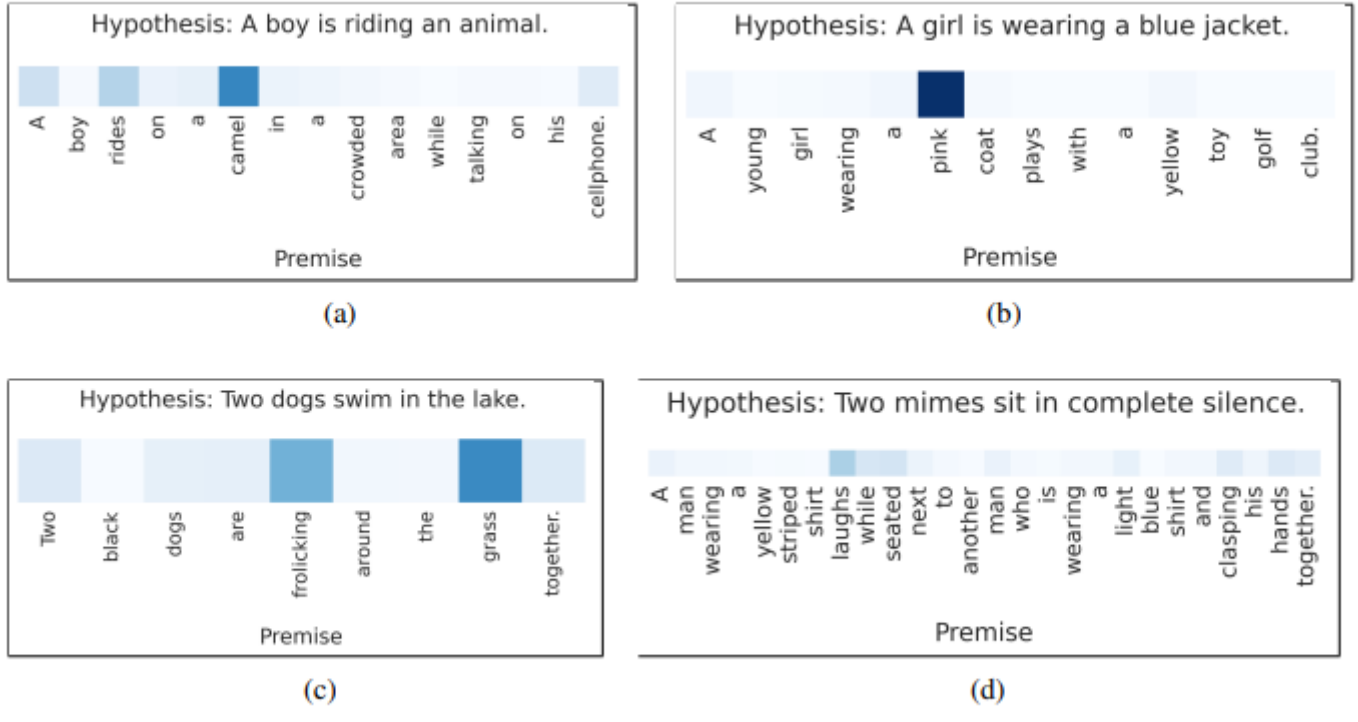


Figure 2: Attention visualizations.

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)

Benefits of Attention

Continuing with our hypothetical Question Answering task

- Attention allows the Decoder to focus on "control"
- By allowing it to access "facts" when needed

We don't **know** if the following is actually what happens, but let's imagine.

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 ?*

Output Pattern:

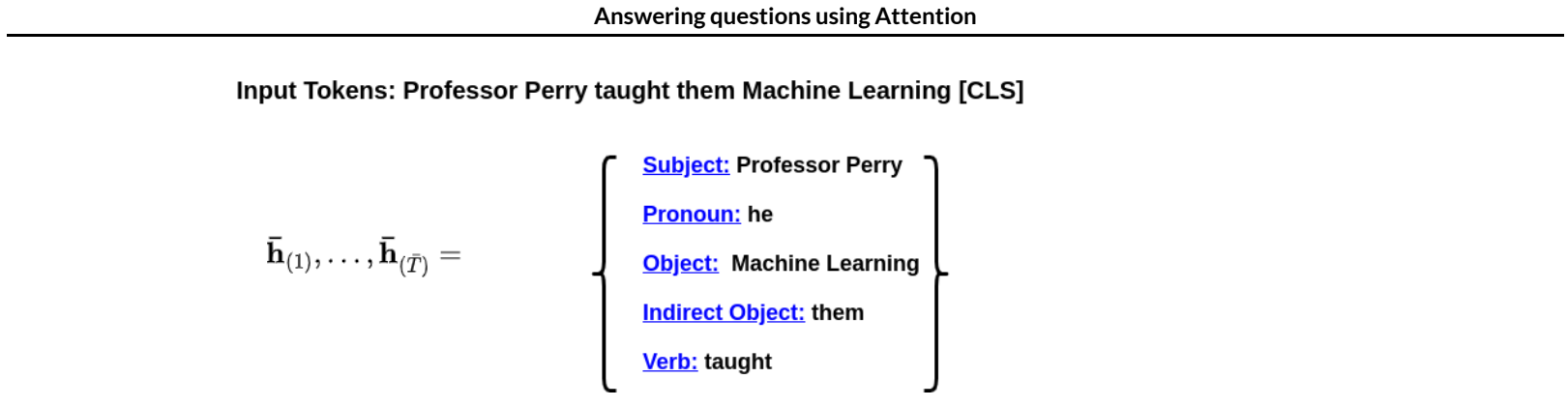
```
<PRONOUN> <VERB> <INIDRECT OBJECT> <OBJECT>
```

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

So the "control" of the Decoder

- needs to output each position of the output pattern
- binding concrete values to each place-holder

And perhaps the Encoder "learns" to create the bindings of concrete values to place-holders



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:

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

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

Answer: He taught them Machine Learning [CLS]

Have we seen this before ?

If you recall the architecture of the LSTM

- *short term* (control) memory
- was separated from *long term* memory
- elements of long term memory are moved to short term memory *as needed*

This is partly similar to the advantages of Attention.

But

- all factual information from input $\mathbf{x}$ has to flow through the bottleneck $\bar{\mathbf{h}}_{(\bar{T})}$ of the Encoder output

Self-attention

We have illustrated the benefit of enabling the Decoder to attend to the Encoder.

This form of attention is called *Cross Attention*.

But we can further simplify the Decoder control by enabling it, when generating $\hat{\mathbf{y}}_{(t)}$

- to attend to all previously generated outputs $\hat{\mathbf{y}}_{([1:t-1])}$

This form of attention if called *Self Attention*

For example, suppose the Decoder is generating a long sentence

- in many languages, there needs to be agreement between the gender/plurality of a subject and verbs
- Self attention enables the Decoder to refer back to the previously generated subject of the sentence
- when generating the verb for each subsequent output position

It is common in an Encoder-Decoder architecture to have both

- Cross Attention from Decoder to Encoder
- Self Attention from Decoder to Decoder

We will see both forms used in the Transformer.

These mechanisms are attending to different sequences (Encoder states or Decoder outputs).

We will henceforth use the term *sequence being attended to* as a general term

- instead of specifically referring to the part of the network that produced it

Masked attention

As presented, the Attention mechanism can refer to an entire sequence

- e.g., the sequence of Encoder latent states

It is sometimes desirable to *limit* what may be attended to.

For example, consider a decision at time t that may depend *only on the past*

- positions $t' < t$
- for example, a trading decision at time t may depend only on *prior* information
    - typical of sequences that are timeseries

Restricting attention to the past is called *Causal Attention*.

- the next output depends only on things that could have caused it (the past), not the future

There is a mechanism to restrict what may be attended to in a general way

- create a "mask"
- a bit vector for each position of the sequence being attended to
- such that attention is limited to positions where the mask element if True.

This is called *Masked Attention*.

It is frequently used to enable a Decoder, when predicting output $\hat{\mathbf{y}}_{(t)}$

- to attend to **previously** generate outputs $\hat{\mathbf{y}}_{[1:t-1])}$
- but not **future** outputs $\hat{\mathbf{y}}_{(t')}$ for $t' \geq t$

When used in this manner, we refer to the behavior as *Masked Self Attention*

Aside

You may wonder how it is even practically possible for a Decoder to refer to the future.

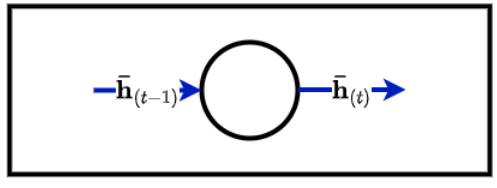
When using *Teacher Forcing* for **training**

- the Decoder does not use the *predicted* target sequence $\hat{\mathbf{y}}_{(1:T)}$
- the Decoder uses the *actual* target sequence $\mathbf{y}_{(1:T)}$
    - hence, "future" positions $t' \geq t$ are available
- this prevents a single mis-prediction at position t from cascading and ruining all future output
    - facilitates training
- at inference time: the Decoder works on the *predicted* Target sequence.

In the diagram below, we illustrate (lower right) how the Decoder input changes between Training and Test/Inference time.

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

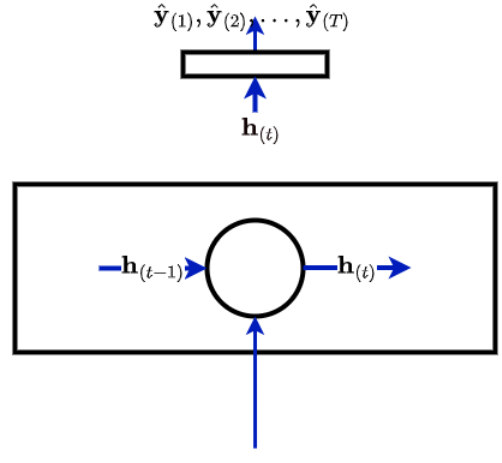
Encoder



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

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

Decoder



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

Inference

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

Training

Multi-head attention: two heads are better than one

Perhaps when generating the output for position t of the output sequence

- we need to attend to *more than one* position of the sequence being attended to
    - need to know both gender and plurality of subject
- that is: we want an Attention layer to output multiple items.

We can attend to n positions

- by creating n separate Attention mechanisms
- each one called a *head*

This behavior is referred to as *Multi-head attention*

This type of behavior is common to many layer types in a Neural Network

- a Dense layer l may produce a vector $\mathbf{y}_{(l)}$ where $n_{(l)} > 1$
- a Convolutional layer l may produce outputs (for each spatial location) for many channels

We have referred to this as layer l producing $n_{(l)}$ *features*.

It would be natural for an Attention layer to output many "features" to enable attention to many positions.

In practice, this is sometimes (always ?) not done

- Model architectures (e.g., the Transformer) are simplified when the inputs/outputs of each sub-component
- have the same length
- often denoted as d of d_{model} in the Transformer

When a Transformer needs to attend to n positions

- it uses n Attention heads
- each outputting a vector of length $\frac{d}{n}$
- which are concatenated together to produce a single output of length d

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 $\frac{d}{n}$

The picture shows n Attention heads.

Note that each head is working on vectors of length $\frac{d}{n}$ rather than original dimensions d.

- variables with superscript (j) are of fractional length

Details are deferred to the module Attention lookup (Attention_Lookup.ipynb).

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.

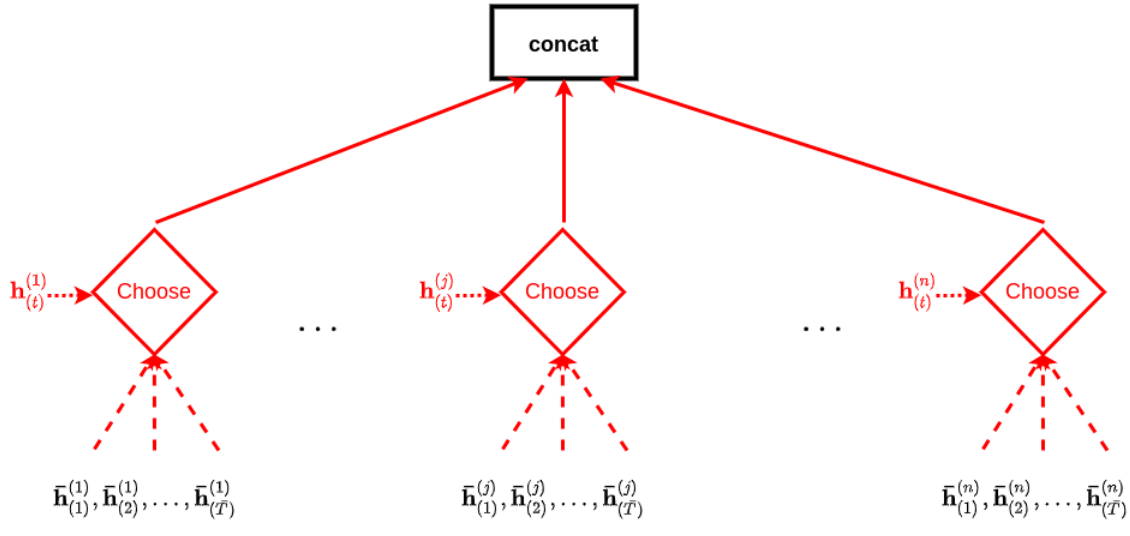
- into $\mathbf{h}_{(t)}^{(j)}$ and the key/value pairs $\bar{\mathbf{h}}_{(1)}^{(j)} \ldots \bar{\mathbf{h}}_{(\bar{T})}^{(j)}$
- Such that each head attends to a separate item

Decoder Multi-head Attention

Per-head query and value

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

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



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
```