Implementing Attention: motivation

Attention is a mechanism

- Used in sequence to sequence problems
- Which maps a Source sequence to a Target sequence
- Often (but not necessarily) utilizing an Encoder-Decoder architecture

- To cause the Decoder at time step t
- To "attend to" (focus it's attention)
- On a particular prefix of the Source input sequence $\mathbf{x}$

That is

- Each output of the Target sequence
- Is dependent on a "context"
- Which is defined by the Source sequence

Decoder: Attention

We will show the basic mechanism for Attention.

Attention is all you need (https://arxiv.org/pdf/1706.03762.pdf) is the key paper on this topic

Note that current practice

- Most often uses a variant of this mechanism called *Self Attention*
- In a popular and powerful architecture called the *Transformer*
- We will provide a simplified explanation using a two part Encoder-Decoder model
- Without specifically referring to the architecture of either part

Implementing attention: mechanics

To state the problem of Attention more abstractly

- The source sequence is $\mathbf{x}_{(1)}, \ldots, \mathbf{x}_{(\bar{T})}$.
- The Encoder associates a "context" $\bar{c}_{(\bar{t})}$ with the prefix of $\mathbf{x}$ ending at $\bar{t}$, for $1 \le \bar{t} \le T$
- The Decoder associates a context $c_{(t)}$ with the output generation

The problem of Attention

- Is finding the Source context $\bar{c}_{(\bar{t})}$
- That most closely matches the desired Target context $c_{(t)}$

Getting a little philosophical:

- A "thought" is an amorphous collection of neurons in the brain: "A sunny day at the beach"
- A "sentence" is a sequence of words that describes the thought
- The "sentence" may be different in two distinct languages, but they represent the same thought
- The context is the Neural Networks representation of the thought

So we translate from Source sequence to Target sequence

- By matching the contexts of the Source (Encoder) and Target (Decoder)

The Source context $\bar{c}_{(\bar{t})}$

- Can be generated by a smaller Neural Network that is part of the Encoder

Similarly the Target context $c_{(t)}$

- Can be generated by a smaller Neural Network that is part of the Decoder

To summarize

- The Encoder creates a context for each prefix of the Source input
- The Decoder creates a context for each prefix of the Target output
- At step t, the Decoder "attends to" the Source context $\bar{c}_{(\bar{t}\,)}$ that most closely matches the Target context $c_{(t)}$
    - Using this context to generate $\hat{\mathbf{y}}_{(t)}$

The mechanism we use to match Target and Source contexts is called *Context Sensitive Memory* which we introduced in a previous module (Neural_Programming.ipynb#Soft-Lookup)

To recap:

Attention lookup: in practice

For simplicity we described attention as a soft-match of queries against keys, producing weighted values.

In practice: we transform each of the queries, keys and values

- $q \mapsto qW_Q$
- $k \mapsto qW_K$
- $v \mapsto qW_V$

by mapping through *embedding matrices* W_Q, W_K, W_V which are **learned** parameters of the model.

This generalization will find a "more useful" representation if it exists

- if there is no useful mapping then presumably we learn the identity matrix

We can perform the Context Sensitive Memory lookup *in parallel* across all elements of a sequence $\mathbf{x}$ (written as matrix $\mathbf{X}$) via matrix multiplication:

Using $\mathbf{X}$ as the queries, keys and values:

- mapping all inputs $x \in \mathbf{X}$:
$$
\begin{aligned}
Q &= \mathbf{X} W_Q \\
K &= \mathbf{X} W_K \\
V &= \mathbf{X} W_V
\end{aligned}
$$
- Computing scores $\alpha(q, k)$ of each query against each key via
$$QK^T$$
    - matching the query patterns Q against all keys K
- Returning a single composite value that is the weighted (by $\alpha(q, k)$) sum of all values

Using Context Sensitive Memory to implement Attention

Remember that our ultimate goal

- Is to generate a context
- That can be passed as the second argument $\mathbf{s}$
- Of the Decoder function responsible for generating Decoder output $\hat{\mathbf{y}}_{(t)}$

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

Context Sensitive Memory is exactly what we need to obtain a value for $\mathbf{s}$.

At time step t, the Decoder:

- Generates a query $q_{(t)}$ containing the Target context
- Matches the query against Context Sensitive Memory M
- To obtain a Source context
- That is equated to $\mathbf{s}$

We will simplify the presentation by identifying contexts with latent states (short-term memory)

$$\begin{aligned}\bar{c}_{(\bar{t})} &= \bar{\mathbf{h}}_{(\bar{t})} \\ c_{(t)} &= \mathbf{h}_{(t)}\end{aligned}$$

So matching Source and Target contexts becomes equivalent to matching Encoder and Decoder latent states.

Define Context Sensitive Memory M to be the pairs

$$\{ (\bar{\mathbf{h}}_{(\bar{t}\,)}, \bar{\mathbf{h}}_{(\bar{t}\,)}) \mid 1 \le \bar{t} \, \le \bar{T} \, \}$$

In other words:

- We make the key equal to the value
- And both are equal to the Source the context $\bar{c}_{(\bar{t}\,)}$

The Decoder then performs a Soft Lookup against Context Sensitive Memory M

- Using query $q_{(t)} = \mathbf{h}_{(t)}$
- Returning a "blend" of Encoder latent states
- As required by the "Choose" box

Extensions

It is not strictly necessary to equate contexts with latent states

- One can implement a small Neural Network to find the "best" representation for contexts

Nor is it necessary for the keys and values of the Context Sensitive Memory to be identical.

The only requirement is that the Encoder and Decoder "speak the same language" and produce values of the appropriate type.

Conclusion

We introduced Context Sensitive Memory as the vehicle with which to implement the Attention mechanism.

Context Sensitive Memory is similar to a Python dict/hash, but allowing "soft" matching.

It is easily built using the basic building blocks of Neural Networks, like Fully Connected layers.

This is another concrete example of Neural Programming.

In [2]:

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

```
Done
```