1. Encoder-decoder Architecture

Suppose that we want to build an ML architecture for *machine translation*.

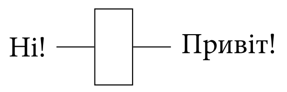

Figure 1: Translating English to Ukrainian

For concreteness, say we want to translate sentence by sentence. The rule of thumb in designing any kind of algorithm is to capture the structure of the problem that we want to solve. A sentence is naturally a sequence of words, and since we want to perform mathematical operations including matrix multiplication on the input, it would be adequate to represent words by vectors.

A common way of representing words as vectors is *one-hot encoding*. That is, suppose we have a dictionary of n words. Then using one-hot encoding, we represent the words by the standard basis vectors of $\mathbb{R}^n$:

$$\text{(a word)} \mapsto \mathrm{e}_k, \tag{1.1}$$

where $\mathrm{e}_k \in \mathbb{R}^n$ has 1 in its kth entry and the other entries are 0.

Although this way of representing words by vectors is straightforward, one major drawback is that it does not capture the structure of words in a language. Some words in a language are synonyms and some are antonyms. Some have totally unrelated meaning. However, in our current representation, this is not reflected; the one-hot encodings are *orthonormal*. Therefore, to capture the structure behind the vectors, we build a transformation called *encoder*.

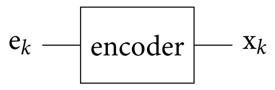

Figure 2: An Encoder

In this setting, an encoder is a map $f : \{\mathrm{e}_k\}_k \to \mathbb{R}^m$. Now, let's think about the relation between n, m. Clearly, the words in a language are related to each other. To represent such relationships, we expect the encoded vectors x_k to be linearly dependent of each other, so it is natural to take $m \leq n$ (in fact, $m \ll n$). Another reason for this would be, by reducing the dimensionality of vector representations of a word, we are *compressing* the data, therefore requiring less resources during the training.

Of course, if we are using encoders to compress our input to the model, we should be also able to *decompress* to obtain one-hot representations back. This is the job of a *decoder*. Therefore, we have an *encoder-decoder architecture*:

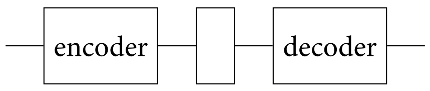

Figure 3: An encoder-decoder architecture

The crucial question is,

$$\textit{what should be the network that goes between the encoder and decoder?} \tag{1.2}$$

2. Recurrent Neural Network (RNN)

As mentioned earlier, we should try to capture the structure of languages, and sentences in a language has a natural sequencial structure. To see this closely, let us consider a sentence represented as $X = \left(\mathrm{x}^1, \ldots, \mathrm{x}^k\right)$, where x^i are the encoded representations of the words in the sentence. Clearly, the words in a sentence are highly-correlated. For instance, if we see the word *the*, we are quite certain that the following word would not be a verb. In other words, if we know the *prior* words $\mathrm{x}^1, \ldots, \mathrm{x}^{i-1}$ in the sentence, then given a candidate ξ for x^i, we expect the probabilities $p\left(\mathrm{x}^i = \xi\right)$ and $p\left(\mathrm{x}^i = \xi \middle| \mathrm{x}^1 \cdots \mathrm{x}^{i-1}\right)$[1] to be quite different. This is at the core of a *recurrent neural network* (RNN). To discuss RNN efficiently, let us first consider a diagrammatical representation of it, at least on the input side.

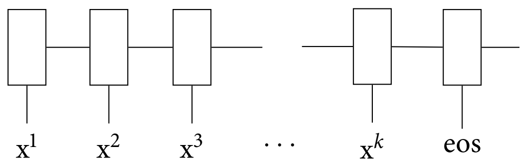

Figure 4: An input part of an RNN

An important part in the above diagram is that *all boxes represent the same network*. This is an important feature of an RNN and is where the word *recurrent* comes up. Note that, starting from the second network, the ith box accepts not only x^i but also the output of the previous box (i.e. $(i-1)$th box). But $(i-1)$th box also accepts an output from the $(i-2)$th box and so on. Therefore, we can see that the output of ith box depends on not only x^i but $\mathrm{x}^1, \ldots, \mathrm{x}^{i-1}$ as well. If we denote y^i to be the output of the ith box, then we see that $\left(\mathrm{y}^i\right)_i$ is a stochastic process where the y^i depends on $\mathrm{y}^1, \ldots, \mathrm{y}^{i-1}$, which is a form of a (generalized) *auto-regressive* process.

Problems of an RNN

A crucial caveat of an RNN comes from this way of modelling a sequence, however.

(a) In a sentence, the ith word x^i correlates with all other words x^j in the sentence, not only for $j < i$. RNN fails to capture this property.

(b) Say we have a very long sentence. When an RNN considers the final word x^k, notice how the input from the previous words x^i $(i < k)$ become progressively weaker as i decreases. To see why, note that, at ith network, we are combining $\mathrm{y}^{i-1}, \mathrm{x}^i$ to output y^i. But y^{i-1} in turn depends on the outpus of the previous networks, for instance y^1. Since we are considering an additional input x^i to get y^i, effectively we expect y^1 to be less correlated with y^i than y^{i-1}. So as the *time-step* i increases, the information from the earlier networks get weaker. So virtually, we won't be able to access information from x^1 at the ith time-step when i is very large. Another way of saying this is that, accessing information at distance i from the current time-step requires an operation whose time complexity is nonconstant in i (it is $O(\log(i))$ at its best, at least to the author's current knowledge).

To tackle this problem, an *attention mechanism* was proposed.

3. Attention Mechanism

In short, an *attention mechanism* is a mechanism (unsurprisingly) that allows one to access all other x^j in an equal footing when considering x^i. Originally, it was proposed to boost the performance of RNNs, but in the paper *Attention Is All You Need*, it was proposed that it can be solely used to build, say, a translator that outperforms RNN based ones. Such a model is called a *transformer*. So why is the attention mechanism so powerful?

[1]This is quite an abuse of notation, since we are implicitly treating x^i to be a random variable. So we should really specify realizations $\xi^1 \cdots \xi^{i-1}$ for $\mathrm{x}^1 \cdots \mathrm{x}^{i-1}$ and write $p\left(\mathrm{x}^i = \xi \middle| \mathrm{x}^1 \cdots \mathrm{x}^{i-1} = \xi^1 \cdots \xi^{i-1}\right)$.