Chapter 7

How neural machine translation works

Juan Antonio Pérez-Ortiz
Universitat d’Alacant, Spain

Mikel L. Forcada
Universitat d’Alacant, Spain

Felipe Sánchez-Martínez
Universitat d’Alacant, Spain

This chapter presents the main principles behind neural machine translation systems. We introduce, one by one, key concepts used to describe these systems, so that the reader achieves a comprehensive view of their inner workings and possibilities. These concepts include: neural networks, learning algorithms, word embeddings, attention, and the encoder–decoder architecture.

1 Introduction

The first thing you should know about neural machine translation (NMT) is that it considers translation as a task involving operations on numbers performed by mathematical systems called *artificial neural networks*: these systems take a sentence and transform it into a series of numbers. They add some more numbers here (usually, thousands or millions of them), multiply by other numbers there, perform a few additional, relatively simple, mathematical operations, and eventually output a translation of the original sentence into another language.

Maybe you have always considered translation from a different perspective: as an intellectual task that involves cognitive processes which can barely be explicitly enumerated and which take place in some deep areas of the human brain.

Juan Antonio Pérez-Ortiz, Mikel L. Forcada & Felipe Sánchez-Martínez. 2022. How neural machine translation works. In Dorothy Kenny (ed.), *Machine translation for everyone: Empowering users in the age of artificial intelligence*, 141–164. Berlin: Language Science Press. DOI: 10.5281/zenodo.6760020

And you are indeed right! But the approximation currently carried out by computers follows a completely different path: millions of mathematical operations are performed in a fraction of a second to obtain a translation which may sometimes be labelled as adequate and may sometimes not. And it turns out that the percentage of times they happen to be adequate has dramatically in the last few years. But, historically, artificial neural networks were devised as a simplified model of how *natural neural networks* such as our brains work, and the cognitive processes carried out in it are also the result of distributed neural computation processes which are not that different from the mathematical operations mentioned above.

This chapter will teach you the key elements of NMT technology. We will start off by pointing out the connection between how translation could be carried out in a human brain and how an NMT system undertakes it. This will help us to introduce the basic concepts needed to get a comprehensive overview of the principles of *machine learning* and *artificial neural networks*, which constitute two of the cornerstones of NMT. After that, we will discuss the essential principles of *non-contextual word embeddings*, a computerised representation of words with many interesting properties that, when combined through a mechanism known as *attention*, will produce the so-called *contextual word embeddings*, a key factor in the realisation of NMT. All these ingredients will allow us to present an overall picture of the inner workings of the two most used NMT models, namely, the *transformer* and the *recurrent* models. The chapter wraps up by introducing a series of secondary themes that will improve your knowledge on how these systems run behind the scenes.

2 An imperfect analogy between human translation and NMT

To simplify the discussion a bit, let us make the radical approximation that translating a text is equivalent to translating each of its sentences independently of each other. Let us now assume for a minute that translating a sentence is a two-step process: the translator first determines the *interpretation* or *meaning* of the whole source sentence and then produces in one go a sentence that allows more or less the same interpretation, but is now written in the target language. But every day translators encounter sentences that they have never seen before, such as “The pencil slipped from my hand, stood up, and started talking to me”, and can still translate them: how is that possible? Linguistics has formulated the answer to this question as a principle, the *principle of semantic compositionality*:

we *build* the interpretation of each sentence by combining the individual interpretations of its component words, and the order in which they are combined is dictated by the syntactic structure of the sentence in which words form phrases, phrases form larger phrases, until one gets to the whole sentence. A translator would then analyse this interpretation and perform the inverse procedure, but in the target language. Of course, translators do not always process sentences as a whole, particularly when they are long, and they may take shortcuts to avoid building interpretations of whole sentences, but let us stick to this simplification for a while.

NMT works in a similar way. When translating a sentence, during the *encoding* phase, the system assigns a neural *representation*, or *embedding*, to each source-text word in isolation. These neural representations are then combined to produce a similar representation, but this time at sentence level. As they are combined, individual representations are also modified according to their context; one could consider this a contextualised representation of interpretation or meaning. Then, in the *decoding* phase, the sentence-level representations are unravelled step by step to predict, one by one, the words in the target sentence. The *encoder* and the *decoder* performing these two phases are artificial neural networks interconnected into a single composite neural network.

As in the case of translators, current neural architectures do not really work by considering the whole source sentence when producing each target word, but rather have learned to pay *attention* to the relevant source words and the target words already produced when they do so.

In the remaining sections of this chapter we will describe in more detail the nature of these representations, the structure of the artificial neural networks (which we may simply call ”neural networks” from now on) that build and transform them by selectively paying attention to what is important, and the ways in which these artificial neural networks can be *trained* to do this task using translation examples.

3 Artificial neural networks

To make sense of NMT, one needs to consider in more detail the artificial neural networks (Goodfellow et al. 2016) that perform it: what they are made of, how they work and how they are trained.

The name *neural* clearly invokes neurons and the way in which the nervous systems of animals, and particularly people’s brains, work. Artificial neural networks are indeed made up of thousands or millions of artificial units that resemble neurons whose *activation* (that is, how *excited* or *inhibited* they are) depends on the signals they receive from other neurons and the strength of the connections carrying these signals.

3.1 Artificial neurons

Artificial neurons are the main building blocks of artificial neural networks. These artificial neurons (we will simply call them *neurons* from now on) may be seen as operating in two steps when updating their state or activation. Let us imagine the simple situation in Figure 1 in which we study how the activation of neuron S_4 is updated in response to stimuli received from neurons S_1, S_2, and S_3.

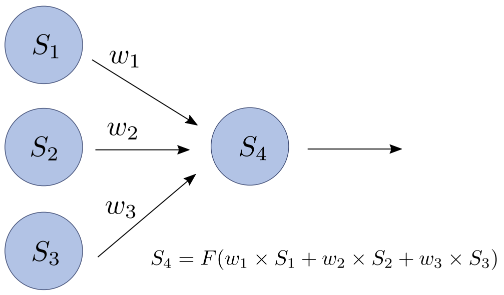

Figure 1: Updating the state S_4 of artificial neuron 4 in response to stimuli received from neurons 1, 2 and 3.

In the first step, the activations of neurons S_1, S_2 and S_3, all of them connected to neuron S_4, are added, but first each one is multiplied by a *weight* (w_1, w_2 and w_3) representing the strength of their connections; these weights determine how their activations are turned into actual stimuli for neuron S_4. Weights may be positive or negative. For instance, if weight w_2 is positive and the activation of S_2 is high, it will contribute to exciting neuron S_4 (a positive stimulus); if, however, w_2 is negative, it will contribute to inhibiting neuron S_4 (a negative stimulus). In general terms, neurons connected through positive weights tend to be simultaneously excited or inhibited, while neurons connected through negative weights tend to be in opposite states. Coming back to neuron S_4, if we add the stimuli coming from each neuron, we get a *net stimulus*:

$$x = w_1 \times S_1 + w_2 \times S_2 + w_3 \times S_3. \tag{1}$$

The net stimulus x can take any possible value, negative or positive, but it is not the activation of neuron S_4 yet. In the second step, neuron S_4 *reacts* to this stimulus. In the example, when the stimulus is intermediate, that is, not too positive or too negative, the neuron S_4 is very sensitive to it. However, when stimuli get large (no matter if positive or negative), changes in their values have a lesser impact on the output, as the neuron is respectively largely inhibited or largely excited.

In the example, neuron S_4 is such that its activation is bound between -1 and $+1$. Figure 2 represents how neuron S_4 reacts to the stimulus in equation 1. The reaction is represented with a function $F(\ldots)$, called the *activation function*, which is applied to the stimulus; the result is the activation of S_4:

$$S_4 = F(x) = F(w_1 \times S_1 + w_2 \times S_2 + w_3 \times S_3). \tag{2}$$

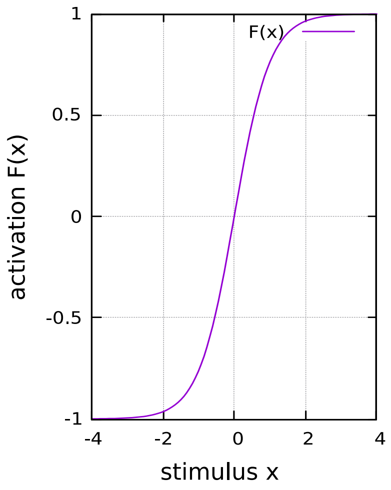

Figure 2: How a neuron reacts to the total stimulus received.

As can be seen, for values around 0 in the horizontal axis the reaction is proportional to the stimulus, but for large positive or negative stimuli, when the neuron is very inhibited or very excited, the reaction is much smaller. For this kind of neuron, the actual extreme values of -1 and $+1$ are never reached, no matter how strong the total stimulus is. As said above, neuron S_4 in our example is a specific type of neuron with an activation that varies between -1 and $+1$. There are other kinds of activation functions with different ranges, but exploring them is out of the scope of this chapter.

3.2 From neurons to networks

Neurons like the one discussed in the previous section may be connected to form an artificial neural network that performs a specific computational task, to solve

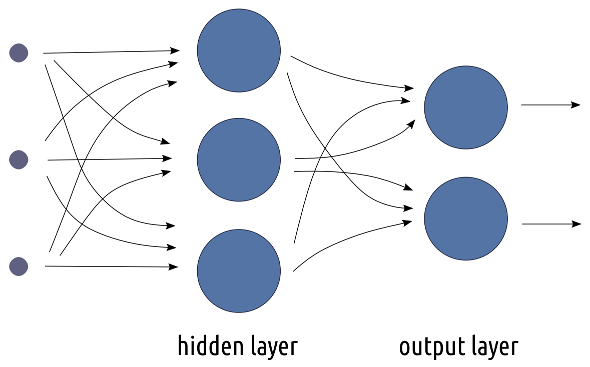

Figure 3: An artificial neural network with three three hidden neurons and two output neurons. Each connection has a weight not shown in the diagram. The three input neurons on the left are represented by smaller circles to emphasise the idea that they directly emit the values of the external input, but, unlike regular neurons, they do not compute a stimulus or react to it via an activation function.

a specific problem. In a network, some neurons receive external stimuli which act as *inputs* to the network (much as our eyes are connected to our brain and feed it with images) and represent an instance of the problem to be solved; some neurons, known as *hidden neurons*, receive stimuli only from other neurons; and finally, some neurons, known as *output neurons* represent the solution to the problem (a bit like the signals sent to the muscles of one of your hands to move it in a specific way). Figure 3 shows an example of such a neural network with five neurons; the network takes three inputs, which are fed to three hidden neurons, which in turn stimulate two output neurons.

When building a neural network to solve a specific problem, one first needs to determine its *architecture*: how many neurons it has, how they are connected, which neurons receive external inputs and which neurons are designated as output neurons; but the actual computation performed depends on the weights of all of the connections in the network. How these weights are arrived at is explained in Section 3.5. Suffice it to say here, that one nice feature of artificial neural networks is that they may be *trained* to perform a task from examples, that is, their weights may be set to specific values by observing a set of solved examples, each one made up of the values of input signals representing the problems, and the values of the desired output activations representing the solutions.

3.3 Layers of neurons

Imagine that you are an absolute beginner and want to learn some basic techniques to paint landscapes in oils. A manual might teach you a step-by-step over-simplified method with, for example, these four stages: drawing (a rough composition is sketched in), colour distribution, drawing refinement, and finish (when the final touches are made). The point here is not the number of stages or the particular characteristics of each of them, but the fact that the whole process flows in an incremental manner in such a way that the output of one step becomes the input to the next one. Each step refines the previous outcome: the outcome of the second step (colour distribution) is more of an actual landscape painting than the outcome of the first one (drawing) and, similarly, the outcome of the fourth stage (finish) can be conceptually considered as a better painting than those resulting from any of the previous steps.

It turns out that neural computation benefits from a similar step-by-step incremental process. Back in the sixties, researchers discovered that by including multiple layers of neurons more complex tasks could be tackled. Each layer in a multilayer neural network refines the output of the previous layer and takes a bigger or smaller step towards the ultimate solution. The resulting architecture would be similar to that in Figure 3 but with a number of additional hidden layers. One can clearly see this layered structure in the simple network in Figure 3: computation, performed by two layers, takes place in two steps.

A model made of neurons organised in layers is referred to as a *layered neural network*. In spite of theoretical results proving that a two-layer network has enough computational power to perform virtually any task (Hornik 1991), in the real world, the computational power of neural networks appears to be correlated with the number of layers; models with more than a few layers are often labelled as *deep neural networks* and the corresponding training algorithms are known as *deep-learning algorithms*.

As an example of the complexity that these deep models may reach, GPT-3 (Brown et al. 2020), one of the largest neural networks released in 2020 in the field of natural language generation, has 96 layers with tens of thousands of neurons each, which results in around 175,000 million weights to be learned by the training algorithm. Supercomputers were used to train the GPT-3 system, a process that can take several weeks or even months, but it has been estimated that learning the weights for such a model with a single powerful gaming desktop personal computer would have taken more than 350 years.[1]

[1]"OpenAI's GPT-3 language model: A technical overview" (2020). Retrieved from https://lambdalabs.com/blog/demystifying-gpt-3.

3.4 Neural machine translation

If we manage to represent a source sentence as a set of inputs to a neural network, and we can interpret the neural network's outputs as a target sentence, we have a *neural machine translation* (NMT) system. NMT first processes the words in the source sentence. Each time a source word is ingested by the encoder part of the neural network, the activations of sets of specific neurons in the network change. When the whole source sentence has been processed, the decoder part of the network starts its work. It has been trained to provide, step by step, a probability score for each possible target word in the translation, given the target words it has already output. This is similar to how predictive keyboards in contemporary smartphones work, but, as we will see, word predictions in NMT also depend on the source sentence, as they are meant to be a translation of it.

NMT systems are deep neural networks with architectures that will be discussed later in section 6. They have thousands of neurons and millions of weights (or many more) which have to be trained by providing examples taken from a parallel corpus containing millions of source sentences and their translations. Mathematical representations of the words in a given sentence in the source language are fed as inputs to the neural network and the words in the corresponding target-language sentence are used to represent the desired output. As you might expect, training a large network in reasonable time is computationally demanding: one needs very powerful, specialised number-crunching hardware to train the network by showing it the examples over and over again. On each iteration, small changes are made to the weights in the network to improve its prediction of target words.

3.5 Training neural networks

Training a neural network is the process of determining the weight of the connections between its neurons so that, given a *training set* of input–output examples, it produces an actual output which is as close as possible to that in the relevant example.

Training starts with a set of random weights or with weights taken from a neural network solving a similar task. During training weights are modified in such a way that the value of an error function (also known as a *loss function*), which measures how much actual outputs deviate from the desired outputs, is made as small as possible. *Training algorithms* (also called *learning algorithms*) repeatedly compute small corrections *(*updates) to weights until the error function is minimal or small enough for all examples in the training set, or a certain performance is observed in a different *development set*, which has been reserved or ”held out” for this purpose (see Section 7.2). The technical details of the training algorithm are beyond the scope of this chapter; let us just say that it is usually based on computing how much the error function varies when each weight is varied by a fixed but very small amount (the *gradient* of the error function), and then varying each weight a bit in the direction in which it reduces the error function.[2] This type of training is called *gradient descent*; it is not guaranteed to find the very best weights, but it is likely that good candidates will be found. The intensity of these weight variations is regulated by a parameter called the *learning rate*; this learning rate is usually higher in the first steps of the training algorithm, but its magnitude is made progressively smaller as the weights get closer to their final values. Note that training neural networks is quite laborious: many examples are necessary and they need to be presented many times to learn. This is often due to limitations of the training algorithms, however, rather than to the lack of capacity of a specific neural network to represent the solution to a problem.

Once the weights are determined, training stops (see Section 7.2) and the neural network can be used to obtain the outputs for new inputs which are not included among the examples used during training.

3.6 Generalisation in neural networks

Generalisation is a fundamental cognitive process for humans and animals. It allows us to use what we learned in the past in new situations which can be regarded as similar but not identical to the situation in which learning originally took place. A person does not need to relearn how to drive when entering a new street or driving a new car. Similarly, generalisation happens when an organism which already responds to a certain stimulus in a particular way responds to similar stimuli in similar ways. Generalisation is also key to language learning: young children soon learn to say sentences they have never heard before.

Neural networks may ideally generalise in the context of machine translation by producing similar outputs when fed with similar inputs, independently of whether they were included in the training set or not. One feature of neural networks is the *smoothness* of the computations, meaning that if the input values are slightly changed, the result of the formulas will not vary significantly.

In a broad sense, in order to achieve generalisation, similar sentences should get similar representations, and as sentence representations will be obtained from word representations, we may conclude that representing similar words

[2]Some of you may recognise here the mathematical concept of *derivative of a function.*

with similar numbers is a precondition for generalisation in neural language processing.

The next section will delve into how we can end up with a convenient list of neural representations for the words in a sentence that benefits from the smoothness of neural networks so that, after training, the system is able to generalise properly to sentences it has not seen before.

4 Word embeddings as vector representation of words

In the previous section we noted that neurons are usually arranged in layers in such a way that the output of the neurons of one layer becomes the input to the neurons of the following one. Interestingly, the output of the set of neurons in a given layer constitutes a representation of the information they are processing at that stage.

In the field of natural language processing, and as indicated above, the information processed by neural networks is made up of words, and their representations within the network are usually referred to as *embeddings* (Mikolov et al. 2013). What makes these embeddings really useful is that those words with similar meanings or that usually co-occur in the same contexts end up having similar embeddings. In order to better understand this, take a piece of paper and draw a square with sides of about 10 centimetres. Now, take the words in the following list and put them all on the square by following a criterion that places words which are closer in meaning nearer each other than words with less related meanings. If this concept of meaning closeness seems imprecise to you, you may place the words based on their frequency of co-occurrence in sentences or paragraphs. The words are: *restaurant*, *red*, *garden*, *fountain*, *flower*, *tomato*, *balloon*, *waiters*, *knife*, *flowers*, *menu*, *cooked*, *chromosome* and *consistently*. Do this before reading on.

The restriction imposed by means of the criterion of word meaning proximity implies that you have not been able to freely distribute the words on the square. Probably, you have decided to group words such as *restaurant*, *menu* and *waiters*, on the one hand, and words such as *garden*, *flower* and *fountain*, on the other hand. There are, however, some doubtful cases: *red* is clearly a neighbour of *tomato*, but it should be close to *flower* as well; a compromise solution would be to put it somewhere in between, a little bit closer to *tomato* than to *flower* if we acknowledge that *red* is not as essential to flowers as it is to tomatoes.

You may have noticed some clusters in your design: an island representing the semantic field of restaurants and related things, and another island around the idea of gardens and orchards. There are some outliers on the list, especially the word *consistently*, which seems in principle disconnected from the rest of words, forcing us to put it as far as possible from all of them. *Chromosome* is another isolated word, but as flowers and waiters use chromosomes to carry their genetic information, it may be put somewhere in the middle of the line between these words but at the same time not very close to *red*. See Figure 4 for a possible solution that may not match yours exactly.[3]

In order to assign mathematical codes to the words in our list, let's assign coordinates to each word to reflect its position on the square. As we are in a two-dimensional space, we need two coordinates for each word: the first coordinate is a number that represents the distance to the left vertical side of the square; the second coordinate is a number that represents the distance to the bottom horizontal side of the square. The word *restaurant* could be assigned, for example, the two numbers 0.25 and 1.1, and the word *menu* the numbers 0.6 and 1.3, close to *restaurant* as seen in Figure 4. These coordinate values can be represented using *vector notation*, which simply consists of writing the numbers as a comma-separated list of values between brackets. The vectors corresponding to *restaurant* and *menu* would therefore be $[0.25, 1.1]$ and $[0.6, 1.3]$, respectively. Each of these vectors represents a possible word embedding for these two words.

Although it may not be completely obvious, considering embeddings made up of two numbers instead of a single number boosts the possibilities of solving the problem of placing words closer or farther apart as we have more freedom to satisfy all the restrictions. In fact, moving from two dimensions to a higher number of dimensions increases these possibilities even more. A five-dimensional representation of a word could be, for example, $[2.34, 1.67, 4.81, 3.01, 5.61]$. NMT systems consider embeddings with hundreds of dimensions, and the input sentence to be translated is represented by a collection of these vast word embeddings.

Word embeddings are learned using the very same algorithm used to learn the weights of the neural network presented in Section 3.5. In fact, both the weights and the embeddings are learned at the same time. Bearing in mind that the input layer of a neural network involved in NMT usually consists of the embeddings of the words in the input sentence, there is no need to limit ourselves to fixed vectors. Instead, their values can be repeatedly updated during training in such a way that the value of the error function is minimised.

[3]We have deliberately placed Figure 4 a few pages on, so that you do not see it before you attempt the exercise.

4.1 Generalisation

As already discussed, for the network to be able to properly *generalise*, that is, to be able to learn to translate and be capable of translating sentences never seen before, similar sentences should get similar representations. As sentence representations are obtained from word embeddings, we may conclude that representing similar words with similar numbers is a precondition for generalisation in neural natural language processing. Following our example, words such as *poured*, *rained*, *pouring* or *raining* should ideally share similar embeddings as all of them are semantically similar; the codes for *pouring* and *raining* should also be closer to words such as *driving* since the three of them are gerunds and may appear in similar contexts; *poured* and *rained* should be neighbours as well because both of them are past tenses. This is why we usually need many dimensions: we want words to be close to each other in different ways or for different reasons, simultaneously.

4.2 Geometric properties of word embeddings

Word embeddings exhibit interesting properties that demonstrate that they represent semantic characteristics (or something related to semantics) of words. As already explained, the embedding of a word consists of several real numbers, usually hundreds or thousands of them, and each of these numbers seems to capture a certain aspect of the meaning of a word. For example, the word embedding for *Dublin* should capture several semantic-related aspects of it: a city, the capital of Ireland, the place for the headquarters in Europe of several multinational companies, etc.

Thanks to this specialisation of the different dimensions of the embeddings, we can perform some arithmetic operations with the embeddings and obtain meaningful results. These operations are simply additions and subtractions that are straightforward to compute. Adding (or subtracting) two embeddings simply consists of adding (or subtracting) the components of the vectors one by one; for example, $[1.24, 2.56, 5.23] + [0.12, 1.12, 0.01] = [1.36, 3.68, 5.24]$. Below are two examples of arithmetic operations with meaningful results performed on embeddings that NMT systems usually learn:

$$\begin{aligned} [\mathit{king}] - [\mathit{man}] + [\mathit{woman}] &\simeq [\mathit{queen}] \\ [\mathit{Dublin}] - [\mathit{Ireland}] + [\mathit{France}] &\simeq [\mathit{Paris}] \end{aligned}$$

where the square brackets refer to the embedding of a word, and with $\simeq$ we mean that the resulting embedding after the operation is close to the embedding of the

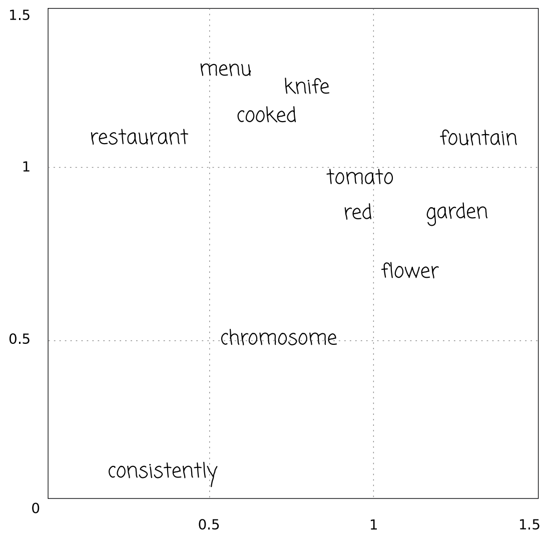

Figure 4: Placement of words in a two-dimensional area in such a way that related words are positioned close to each other, but far from words they have less in common with.

word on the right-hand side of the example. This can be interpreted as indicating that *king* is to *man* what *queen* is to *woman*, a male or female monarch; and *Dublin* is to *Ireland* what *Paris* is to *France*, the capital of a country.

5 Contextual word embeddings through attention

Words do not always have the same meaning in every sentence. The embedding of the word *letter*, for example, should not be the same when the word refers to a character of an alphabet or when it refers to a document addressed to another person. In fact, it may even be interesting for an NMT system to represent the word with different embeddings depending on whether it refers to a love letter or a complaint letter. The embeddings we introduced before are *non-contextual*: they were computed by considering words that usually co-occur in sentences but without taking into consideration the different meanings words may have.

In the NMT arena, *attention* plays an important role as it allows the neural network to compute *contextual word embeddings*, that is, vector representations of the words in a sentence computed in such a way that the representation obtained for a word is adapted to its meaning in each particular sentence. Attention is, once again, a concept which is implemented by means of mathematical operations conveniently learned by a training algorithm. In our context, attention is similar, to the situation in which we pay attention to something or someone in our everyday lives.

By conveniently using attention to concentrate on some words in the sentence, the embedding vector corresponding to the word *season*, for example, will differ between the sentences in examples 1 and 2 below:

1. The first episode will pick up right where the previous season left off.

2. Summer is the hottest season of the whole year.

In principle, it may sound as if the purpose of contextual word embeddings is that the different meanings of a word get different representations, but, while this will be usually true, the idea goes beyond this. The contextual word embeddings for *season* in the sentences "Winter is the coldest season of the year in polar and temperate zones", "Summer is the hottest season of the whole year" and even "Of the whole year, summer is the hottest season" will all be different, although presumably closer to each other than the representation of *season* in "The first episode will pick up right where the previous season left off". These divergences result from the fact that the words in the sentences or the order in which they are placed differ. Remarkably, the two instances of *the* in each of our examples will get two different contextual vectors because the context of each instance is also different.

How are contextual embeddings mathematically computed through attention? Given the sentence in example 2 above ("Summer is the hottest season of the whole year."), the procedure starts by obtaining the non-contextual word embeddings that were introduced in Section 4. As the sentence has nine words, the result is a collection of nine vectors which are the ingredients for the next step. Now, in order to compute the contextual word embedding for the word *season* in the sentence, an attention vector is mathematically produced by the neural network. This attention vector will have nine percentages representing the degree of attention that needs to be paid to each of the words in the sentence in order to obtain the representation of the word *season*. The element at a certain position in the vector corresponds to the attention to the word at that position in the sentence. For example, an attention vector $[25\%, 8\%, 10\%, 15\%, 25\%, 8\%, 2\%, 0\%, 7\%]$ would indicate that in order to compute a contextual vector representation of the word *season* in the running sentence, the word embeddings for *summer* and *season* will be equally highly relevant (together, they receive fifty percent of the total attention), which makes sense as they are semantically connected to the concept of a meteorological season. Notice that the preceding determiner gets some attention too (10%), which may be explained by the fact that it helps to label *season* as a noun. The contribution of the verb (8%) to the contextual embedding may also be described in terms of its contribution to marking the number of *season* as singular. Note that the percentages always add up to 100%.

Determining how the attention vector is used in order to obtain a new embedding that combines the original non-contextual embeddings to get a new embedding is beyond the scope of this chapter. Suffice to say that the procedure involves a specific sequence of mathematical operations and that the resulting embedding will be located somewhere in between the original embeddings.

Following our running example, nine different attention vectors will be computed for this sentence (one for each word) and then applied to the original non-contextual embeddings in order to obtain a collection of nine new embeddings, each one corresponding to a different word in the sentence. These new embeddings may be considered as contextual embeddings as they are influenced to different degrees by the rest of the words in the sentence.

5.1 Many attention layers, better than one

Previously, in Section 3.3 of this chapter, we discussed the benefits of successively refining neural computations by exploiting models with different layers. Consequently, it will come as no great surprise that in order to obtain more precise representations, the contextual embeddings just obtained may be combined with new attention vectors to obtain yet another new embedding for each word. As a real-life example, Turing Natural Language Generation (T-NLG), another of the largest language models published in 2020, has 78 attention layers that successively polish embeddings of 4,256 dimensions.[4] Recall that these representations, which are learned by applying many consecutive layers, are known as *deep* representations.

5.2 Many heads, better than one

There is no reason to restrict ourselves to a single attention vector for each word in each layer. For example, given the sentence "My grandpa baked bread in his oven daily", it could be interesting to have an embedding for *oven* which has the flavour of *grandpa* to reflect that this oven belongs to an older person, and a different embedding for *oven* with the flavour of *bread* to reflect what has been

[4]"Turing-NLG: A 17-billion-parameter language model by Microsoft", 2020. Retrieved from https://www.microsoft.com/en-us/research/blog/turing-nlg-a-17-billion-parameter-language-model-by-microsoft/

cooked in it. A single attention vector would have to mix both flavours in a single embedding containing too much heterogeneous information that could affect negatively the search for a translation for the word represented by the embedding. For this reason, some NMT systems obtain different attentions for each word in each layer and use them to compute a number of different embeddings for each word. Each of these embeddings is said to be computed by a different *head*. T-NLG has 28 attention heads in each layer. Therefore, its last layer produces 28 different 4,256-dimensional embeddings for each word.

5.3 Contextual word embeddings in natural language processing

Embeddings are the cornerstone of NMT but they have also proved to be useful in many other natural language processing applications such as sentiment analysis and automatic summarisation. As an illustration, systems that automatically classify as positive or negative the sentences in a text containing a product review may work by first computing a collection of deep contextual embeddings for each word in the sentence and then feeding these embeddings to a much simpler neural network that will compute a number between 0 and 1 indicating the degree of positiveness of the sentence (for example, 0.95 will indicate a decidedly positive sentence, 0.2 a negative sentence, and 0.51 a neutral sentence). These systems are usually trained with a corpus of sentences manually tagged by humans. The part of the model that computes the embeddings is not necessarily trained for a particular corpus as *pre-trained* models already trained with millions of sentences are freely available for many languages.

6 Neural machine translation, at last

At this point, you are hopefully in a good position to understand how NMT works, even if we describe its fundamentals in only a few sentences as we do next. We will focus on two architectures: those of so-called transformer and recurrent neural networks.

6.1 Transformer: Attention-based encoder–decoder

Put simply, a transformer NMT system is composed of a module that computes contextual word embeddings for each word in the source input sentence and a second module which successively predicts each word in the target sentence. The former module is called an *encoder* and the latter module is known as a *decoder*. For predicting the words in the target language, the decoder pays attention to

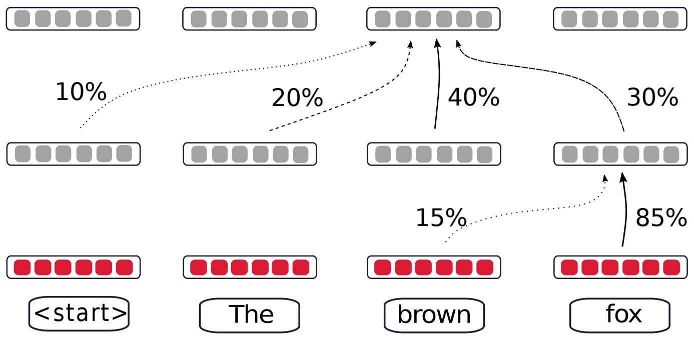

Figure 5: The encoder of a transformer-based neural machine translation system. The symbol *start* is usually prefixed to explicitly mark the beginning of the sentence. The diagram also shows that first-layer embeddings for *brown* and *fox* contribute to different degrees to obtain the embedding for *fox* in the second layer; similarly, the embedding for *brown* in the last layer integrates information from all the embeddings in the second layer using different degrees of attention.

the embeddings of all the words in the source sentence as well as to the embeddings of the target words already generated. The whole architecture is called a *transformer* (Vaswani et al. 2017). Figure 5 shows an example of a three-layered encoder and the degrees of attention considered in order to compute an embedding in the second layer and in the third one. Figure 6 depicts this encoder in an extended diagram that also includes the decoder so that it represents the whole transformer architecture.

A parallel corpus is used by the learning algorithm to obtain a set of weights, embeddings and attention vectors for the transformer such that the training data can be reproduced up to a certain degree and the system is able to generalise beyond the sentences in the training set.

For example, assume that a transformer with one single head per layer is used to translate the sentence "My grandpa baked bread in his oven daily" into Spanish. The encoder first produces a collection of eight embedding vectors. The decoder then computes an 8-dimensional attention vector such as [60%, 10%, 0%, 0%,

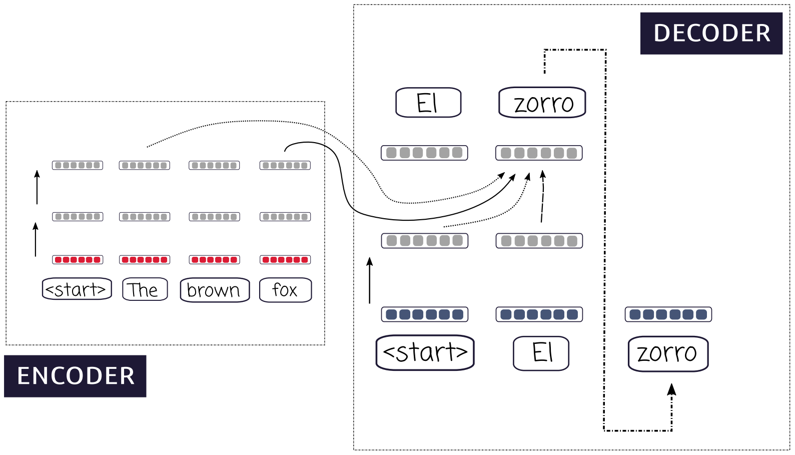

Figure 6: A complete transformer-based neural machine translation system translating a sentence. An enlarged version of the encoder can be seen in Figure 5. Note how the prediction of *zorro* is obtained by paying attention to the embeddings of the previous target words but also to the embeddings corresponding to some of the input words coming from the last layer of the encoder.

0%, 30%, 0%, 0%] and uses it to obtain a flavour of the source sentence that allows it to obtain an embedding for the first word in the target sentence. Let us assume that the system correctly generates the Spanish word *mi*. The decoder will then compute a 9-dimensional attention vector such as [50%, 10%, 0%, 0%, 0%, 20%, 0%, 0%, 20%] (the last percentage corresponds to the attention paid to the first word in the target sentence) and use it to obtain an embedding for the second word in the target sentence. The procedure will continue until the decoder generates a special token that marks the end of the sentence.

The output of the decoder at each step is not exactly an estimation of the embedding of the next word. Actually, an additional layer is added at the end of the decoder to compute a vector of probabilities or likelihoods for each word in the target-language vocabulary. Section 7.3 will discuss how these probabilities can be used in order to obtain the sequence of words that result in the target-language sentence.

6.2 Recurrent architectures

The transformer, as presented in the previous section, is the model used in most current commercial NMT systems, but alternative neural models exist. Another top model is the *recurrent* encoder–decoder model (Bahdanau et al. 2015). Similarly to transformer-based models, there is an encoder that produces a collection

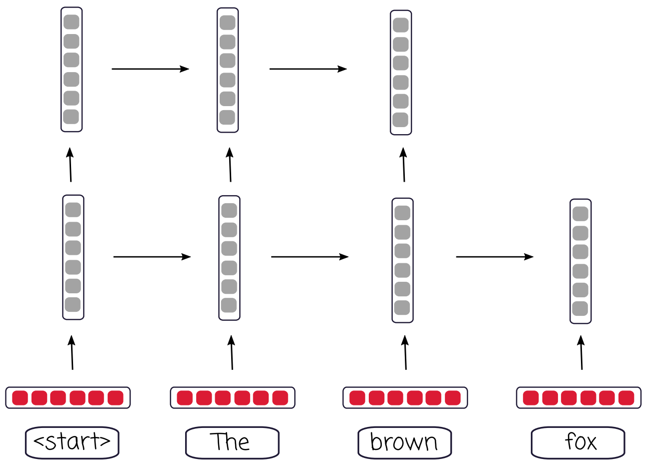

Figure 7: Left-to-right submodel of the encoder of a recurrent neural machine translation system, just after processed "`<start>` The brown" and when about to processs "fox".

of embeddings for the words in the input sentence and a decoder that uses attention to compute embeddings for each target word by integrating the information from the input words and the already generated target words. The encoder and decoder in the recurrent model, however, compute the contextual word embeddings in a local manner in such a way that the embeddings for the fifth encoded word, for example, are based on the embeddings of the four first words, on the one hand, and the embeddings of the next words, on the other hand. This is achieved by traversing the input sentence from left to right and from right to left; see Figure 7 for a diagram of this model showing only left-to-right processing.

It is worth noting that the mathematical model used imposes some restrictions on the relevance given to the words around the word for which the contextual word embeddings are computed (in our example the fifth one), resulting in a mechanism that specially focuses on the nearest words and tends to ignore the representations of distant words. Similarly to the transformer, a final layer at the end of the decoder computes a vector that gives the probability of each target-language word being the word at the corresponding position in the output sentence. Forcada (2017) describes in more detail the recurrent encoder–decoder model and also discusses the kind of outputs that NMT produces.

7 Additional settings

7.1 Words and sub-words

According to what has been presented in this chapter, independently of whether a transformer or a recurrent model is used, an embedding is obtained for each word after training. Does this mean that we end up having an embedding for every possible word in the language? Not really. Languages, specially those which are highly inflected or agglutinative, may easily have hundreds of thousands or even millions of different word forms. In order to understand why this poses a challenge for NMT systems you should know that the number of word embeddings (which is referred as the *vocabulary*) conditions the number of weights in the neural network and that large neural networks often struggle to generalise to unseen data. The size of the vocabulary could be reduced by considering only those word forms present in the training corpus but this usually still implies considering a substantial number of words and raises a new issue: when training is finished and the NMT system undertakes the translation of new sentences containing words not in the training set, these unseen words will make the model perform clumsily and lose accuracy as every unknown word is assigned a single non-contextual embedding reserved for this situation.

The solution engineers came up with is to split words into so-called *sub-word units*. Ideally, these units should make linguistic sense and carry some components of meaning; for instance, splitting *demystifying* as *de-* + *-myst-* + *-ify-* + *-ing* surely makes more linguistic sense (and is therefore likely to be more helpful when it comes to performing machine translation) than splitting it as *dem-* + *-ystif-* + *-yi-* + *-ng*. But performing a linguistically sound splitting requires the existence of a set of splitting rules and procedures for the language in question, a resource that may not be available for many languages.

A commonly-used workaround is to automatically learn splitting rules by inspecting large texts, such as one containing all the source or all the target sentences in the training set. A popular approach[5] is called *byte-pair encoding* (BPE) (Sennrich et al. 2016), and starts with letter-sized units which are joined into two-letter, three-letter, etc. units when they appear frequently in the corpus.[6] Byte-pair encoding would probably identify a frequent *-ing* suffix in many verb

[5]There are more advanced methods such as *SentencePiece* (Kudo & Richardson 2018), which treats the whole text as a sequence of characters and performs word division (*tokenization*) and sub-word division in one fell swoop.

[6]Byte-pair encoding was originally a text compression algorithm: frequent letter (*byte*) sequences would be stored once and replaced by short codes to reduce the total storage needed.

forms (*marching*, *considering*) and chop it off, even for unseen forms (such as *bartsimpsoning*); *-ing* would then be turned into a contextual embedding carrying its atomic meaning.

7.2 Stopping criteria and metrics

As mentioned in section 3.5, in addition to a large training corpus, a small *development corpus* is usually held out and not used for training. The purpose of this corpus is to monitor the performance of the NMT system while it is being trained, to decide, for instance, when training should stop. Training tries to minimise an error function (or, in NMT, actually maximise the probability of the target sentences in the training corpus). One possible problem that may occur is that training too deep on the training corpus hurts generalisation as the neural network ends up *memorising* the example translations too much. This is where the development corpus comes into play: after a certain number of iterations or steps of the training algorithm, the source sentences in the development corpus are translated with the neural network and the output is automatically compared to the desired target sentences in the corpus using simple approximate automatic evaluation metrics (see Rossi & Carré 2022 [this volume]), the most common of which is BLEU (Papineni et al. 2002). BLEU measures how many one-word, two-word, three-word and four-word sequences in the output are found in the reference, and computes a score that varies from 0 (no match) to 100% (all stretches found). If, during training, BLEU on the development set starts to signal a degradation of performance, training may be stopped, or the current set of weights may be stored and training then continued for a while to see if BLEU improves again. Of course, there are many other automatic evaluation metrics which can take the place of BLEU in this process.

7.3 Beam search

The decoder in NMT systems produces the output sentence sequentially, one target word at a time, as explained in Sections 6.1 and 6.2. At each time step, the neural network produces a probability or likelihood (a value between 0 and 100%) for every single word in the target vocabulary. One way of using this information is to pick the most likely target word and output it, ignoring other possibilities. It is worthwhile noting that, in doing so, we are completely determining the ensuing steps taken by the NMT system as the current prediction is given as input to the decoder in the next step (see, for example, the word *zorro* in Figure 6). One possible way to explore more possibilities is to consider, for instance, the three most likely words, and *clone* the system into three systems, each of which would be determined respectively by each of the three choices, and see how they fare. But one cannot do this indefinitely, as one would triplicate the number of systems translating the sentence at each step, and their number would grow exponentially. To avoid that, only a certain number of systems are allowed to *survive*, namely those obtaining the best value in an approximate calculation of the probability of the full sentence that would be produced. This is usually called *beam search* and is a common approximation in other probabilistic models of human language processing such as speech recognition.

8 Conclusions

To train an NMT system, one needs thousands or even millions of examples of source sentence–target sentence pairs. For many language pairs, many domains and many text genres, such resources do not exist, which constrains many specific applications, but for well-resourced languages, general-purpose NMT is a reality and is very widely used, not only by translators. Moreover, scientific advances in approaches such as multilingual models or unsupervised NMT have recently started to produce promising results in low-resource scenarios.[7]

This chapter has introduced – and provided technical details of – the key elements in NMT systems, and explored how they interact in the two currently most popular architectures, namely transformer-based and recurrent neural networks. Research activity in the area is so intense at the time of writing that proposals for new models arise almost every month. Transformers are currently the paradigm of choice if enough parallel corpora are available for training, because they require shorter training times and allow subtle quality improvements in comparison to recurrent neural networks, but the picture may change dramatically at any time.

[7]A *multilingual model* is a single neural network that is trained to translate between many different language pairs so that *knowledge* from well-resourced languages may be *transferred* to low-resourced ones. Interestingly, multilingual models bring the possibility of *zero-shot translation* (Ko et al. 2021) in which a system may be able to translate with reasonable quality, for example, between Spanish and Upper Sorbian using a multilingual model trained on German–Upper Sorbian and Spanish–German corpora, even when no Spanish–Upper Sorbian parallel corpus is available. *Unsupervised NMT* goes a step further by learning NMT systems from monolingual corpora only.

References

Bahdanau, Dzmitry, Kyunghyun Cho & Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio & Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015*. DOI: 10.48550/arXiv.1409.0473.

Brown, Tom B., Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Dario Amodei, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever & Dario Amodei. 2020. Language models are few-shot learners. *CoRR* abs/2005.14165. https://arxiv.org/abs/2005.14165.

Forcada, Mikel. 2017. Making sense of neural machine translation. *Translation Spaces* 6(2). 291–309.

Goodfellow, Ian, Yoshua Bengio & Aaron Courville. 2016. *Deep learning*. Cambridge, MA: MIT Press.

Hornik, Kurt. 1991. Approximation capabilities of multilayer feedforward networks. *Neural Networks* 4(2). 251–257.

Ko, Wei-Jen, Ahmed El-Kishky, Adithya Renduchintala, Vishrav Chaudhary, Naman Goyal, Francisco Guzmán, Pascale Fung, Philipp Koehn & Mona Diab. 2021. Adapting high-resource NMT models to translate low-resource related languages without parallel data. In *Proceedings of the 59th annual meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, 802–812.

Kudo, Taku & John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 66–71. Brussels, Belgium: Association for Computational Linguistics.

Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg Corrado & Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 30*, 3111–3119.

Papineni, Kishore, Salim Roukos, Todd Ward & Wei-Jing Zhu. 2002. BLEU: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 311–318.

Philadelphia, Pennsylvania, USA: Association for Computational Linguistics. DOI: 10.3115/1073083.1073135.

Rossi, Caroline & Alice Carré. 2022. How to choose a suitable neural machine translation solution: Evaluation of MT quality. In Dorothy Kenny (ed.), *Machine translation for everyone: Empowering users in the age of artificial intelligence*, 51–79. Berlin: Language Science Press. DOI: 10.5281/zenodo.6759978.

Sennrich, Rico, Barry Haddow & Alexandra Birch. 2016. Neural Machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, 1715–1725. Berlin: Association for Computational Linguistics.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser & Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, 5998–6008.