CHAPTER

10 Machine Translation and Encoder-Decoder Models

> "I want to talk the dialect of your people. It's no use of talking unless people understand what you say."
>
> Zora Neale Hurston, *Moses, Man of the Mountain* 1939, p. 121

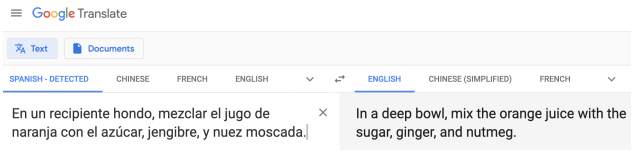
machine translation
MT

This chapter introduces **machine translation** (**MT**), the use of computers to translate from one language to another.

Of course translation, in its full generality, such as the translation of literature, or poetry, is a difficult, fascinating, and intensely human endeavor, as rich as any other area of human creativity.

information access

Machine translation in its present form therefore focuses on a number of very practical tasks. Perhaps the most common current use of machine translation is for **information access**. We might want to translate some instructions on the web, perhaps the recipe for a favorite dish, or the steps for putting together some furniture. Or we might want to read an article in a newspaper, or get information from an online resource like Wikipedia or a government webpage in a foreign language. MT for information access is probably one of the most common uses of NLP technology, and Google Translate alone (shown above) translates hundreds of billions of words a day between over 100 languages.



post-editing
CAT
localization

Another common use of machine translation is to aid human translators. MT systems are routinely used to produce a draft translation that is fixed up in a **post-editing** phase by a human translator. This task is often called **computer-aided translation** or **CAT**. CAT is commonly used as part of **localization**: the task of adapting content or a product to a particular language community.

Finally, a more recent application of MT is to in-the-moment human communication needs. This includes incremental translation, translating speech on-the-fly before the entire sentence is complete, as is commonly used in simultaneous interpretation. Image-centric translation can be used for example to use OCR of the text on a phone camera image as input to an MT system to translate menus or street signs.

encoder-decoder

The standard algorithm for MT is the **encoder-decoder** network, also called the **sequence to sequence** network, an architecture that can be implemented with RNNs or with Transformers. We've seen in prior chapters that RNN or Transformer architecture can be used to do **classification** (for example to map a sentence to a positive or negative sentiment tag for sentiment analysis), or can be used to do **sequence labeling** (for example to assign each word in an input sentence with a part-of-speech, or with a named entity tag). For part-of-speech tagging, recall that the output tag is associated directly with each input word, and so we can just model the tag as output y_t for each input word x_t.

Encoder-decoder or sequence-to-sequence models are used for a different kind of sequence modeling in which the output sequence is a complex function of the entire input sequencer; we must map from a sequence of input words or tokens to a sequence of tags that are not merely direct mappings from individual words.

Machine translation is exactly such a task: the words of the target language don't necessarily agree with the words of the source language in number or order. Consider translating the following made-up English sentence into Japanese.

(10.1) English: *He wrote a letter to a friend*
Japanese: *tomodachi ni tegami-o kaita*
friend to letter wrote

Note that the elements of the sentences are in very different places in the different languages. In English, the verb is in the middle of the sentence, while in Japanese, the verb *kaita* comes at the end. The Japanese sentence doesn't require the pronoun *he*, while English does.

Such differences between languages can be quite complex. In the following actual sentence from the United Nations, notice the many changes between the Chinese sentence (we've given in red a word-by-word gloss of the Chinese characters) and its English equivalent.

(10.2) 大会/General Assembly 在/on 1982年/1982 12月/December 10日/10 通过了/adopted 第37号/37th 决议/resolution ， 核准了/approved 第二次/second 探索/exploration 及/and 和平peaceful 利用/using 外层空间/outer space 会议/conference 的/of 各项/various 建议/suggestions 。

On 10 December 1982 , the General Assembly adopted resolution 37 in which it endorsed the recommendations of the Second United Nations Conference on the Exploration and Peaceful Uses of Outer Space .

Note the many ways the English and Chinese differ. For example the ordering differs in major ways; the Chinese order of the noun phrase is "peaceful using outer space conference of suggestions" while the English has "suggestions of the ... conference on peaceful use of outer space"). And the order differs in minor ways (the date is ordered differently). English requires *the* in many places that Chinese doesn't, and adds some details (like "in which" and "it") that aren't necessary in Chinese. Chinese doesn't grammatically mark plurality on nouns (unlike English, which has the "-s" in "recommendations"), and so the Chinese must use the modifier 各项/*various* to make it clear that there is not just one recommendation. English capitalizes some words but not others.

Encoder-decoder networks are very successful at handling these sorts of complicated cases of sequence mappings. Indeed, the encoder-decoder algorithm is not just for MT; it's the state of the art for many other tasks where complex mappings between two sequences are involved. These include **summarization** (where we map from a long text to its summary, like a title or an abstract), **dialogue** (where we map from what the user said to what our dialogue system should respond), **semantic parsing** (where we map from a string of words to a semantic representation like logic or SQL), and many others.

We'll introduce the algorithm in sections Section 10.2, and in following sections give important components of the model like **beam search decoding**, and we'll discuss how MT is **evaluated**, introducing the simple chrF metric.

But first, in the next section, we begin by summarizing the linguistic background to MT: key differences among languages that are important to consider when considering the task of translation.

10.1 Language Divergences and Typology

universal

Some aspects of human language seem to be **universal**, holding true for every language, or are statistical universals, holding true for most languages. Many universals arise from the functional role of language as a communicative system by humans. Every language, for example, seems to have words for referring to people, for talking about eating and drinking, for being polite or not. There are also structural linguistic universals; for example, every language seems to have nouns and verbs (Chapter 8), has ways to ask questions, or issue commands, linguistic mechanisms for indicating agreement or disagreement.

translation divergence

Yet languages also **differ** in many ways, and an understanding of what causes such **translation divergences** will help us build better MT models. We often distinguish the **idiosyncratic** and lexical differences that must be dealt with one by one (the word for "dog" differs wildly from language to language), from **systematic** differences that we can model in a general way (many languages put the verb before the direct object; others put the verb after the direct object). The study of these systematic cross-linguistic similarities and differences is called **linguistic typology**. This section sketches some typological facts that impact machine translation; the interested reader should also look into WALS, the World Atlas of Language Structures, which gives many typological facts about languages (Dryer and Haspelmath, 2013).

typology

10.1.1 Word Order Typology

As we hinted it in our example above comparing English and Japanese, languages differ in the basic word order of verbs, subjects, and objects in simple declarative clauses. German, French, English, and Mandarin, for example, are all **SVO** (**Subject-Verb-Object**) languages, meaning that the verb tends to come between the subject and object. Hindi and Japanese, by contrast, are **SOV** languages, meaning that the verb tends to come at the end of basic clauses, and Irish and Arabic are **VSO** languages. Two languages that share their basic word order type often have other similarities. For example, **VO** languages generally have **prepositions**, whereas **OV** languages generally have **postpositions**.

SVO

SOV

VSO

Let's look in more detail at the example we saw above. In this SVO English sentence, the verb *wrote* is followed by its object *a letter* and the prepositional phrase *to a friend*, in which the preposition *to* is followed by its argument *a friend*. Arabic, with a VSO order, also has the verb before the object and prepositions. By contrast, in the Japanese example that follows, each of these orderings is reversed; the verb is *preceded* by its arguments, and the postposition follows its argument.

(10.3) English: *He wrote a letter to a friend*

Japanese: *tomodachi ni tegami-o kaita*
friend to letter wrote

Arabic: *katabt risāla li ṡadq*
wrote letter to friend

Other kinds of ordering preferences vary idiosyncratically from language to language. In some SVO languages (like English and Mandarin) adjectives tend to appear before verbs, while in others languages like Spanish and Modern Hebrew, adjectives appear after the noun:

(10.4) **Spanish** *bruja verde* **English** *green witch*

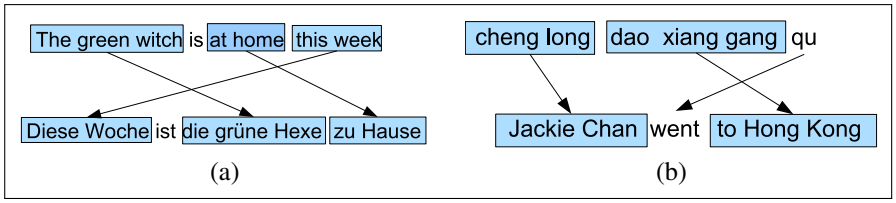


Figure 10.1 Examples of other word order differences: (a) In German, adverbs occur in initial position that in English are more natural later, and tensed verbs occur in second position. (b) In Mandarin, preposition phrases expressing goals often occur pre-verbally, unlike in English.

Fig. 10.1 shows examples of other word order differences. All of these word order differences between languages can cause problems for translation, requiring the system to do huge structural reorderings as it generates the output.

10.1.2 Lexical Divergences

Of course we also need to translate the individual words from one language to another. For any translation, the appropriate word can vary depending on the context. The English source-language word *bass*, for example, can appear in Spanish as the fish *lubina* or the musical instrument *bajo*. German uses two distinct words for what in English would be called a *wall*: *Wand* for walls inside a building, and *Mauer* for walls outside a building. Where English uses the word *brother* for any male sibling, Chinese and many other languages have distinct words for *older brother* and *younger brother* (Mandarin *gege* and *didi*, respectively). In all these cases, translating *bass*, *wall*, or *brother* from English would require a kind of specialization, disambiguating the different uses of a word. For this reason the fields of MT and Word Sense Disambiguation (Chapter 18) are closely linked.

Sometimes one language places more grammatical constraints on word choice than another. We saw above that English marks nouns for whether they are singular or plural. Mandarin doesn't. Or French and Spanish, for example, mark grammatical gender on adjectives, so an English translation into French requires specifying adjective gender.

The way that languages differ in lexically dividing up conceptual space may be more complex than this one-to-many translation problem, leading to many-to-many mappings. For example, Fig. 10.2 summarizes some of the complexities discussed by Hutchins and Somers (1992) in translating English *leg, foot*, and *paw*, to French. For example, when *leg* is used about an animal it's translated as French *jambe*; but about the leg of a journey, as French *etape*; if the leg is of a chair, we use French *pied*.

lexical gap

Further, one language may have a **lexical gap**, where no word or phrase, short of an explanatory footnote, can express the exact meaning of a word in the other language. For example, English does not have a word that corresponds neatly to Mandarin *xiào* or Japanese *oyakōkōo* (in English one has to make do with awkward phrases like *filial piety* or *loving child*, or *good son/daughter* for both).

Finally, languages differ systematically in how the conceptual properties of an event are mapped onto specific words. Talmy (1985, 1991) noted that languages can be characterized by whether direction of motion and manner of motion are marked on the verb or on the "satellites": particles, prepositional phrases, or adverbial phrases. For example, a bottle floating out of a cave would be described in English with the direction marked on the particle *out*, while in Spanish the direction

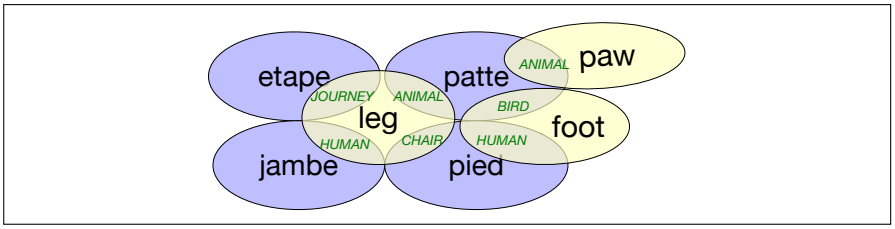


Figure 10.2 The complex overlap between English *leg*, *foot*, etc., and various French translations as discussed by Hutchins and Somers (1992).

would be marked on the verb:

(10.5) English: *The bottle floated out.*
Spanish: La botella salió flotando.
The bottle exited floating.

verb-framed

Verb-framed languages mark the direction of motion on the verb (leaving the satellites to mark the manner of motion), like Spanish *acercarse* 'approach', *alcanzar* 'reach', *entrar* 'enter', *salir* 'exit'. **Satellite-framed** languages mark the direction of motion on the satellite (leaving the verb to mark the manner of motion), like English *crawl out*, *float off*, *jump down*, *run after*. Languages like Japanese, Tamil, and the many languages in the Romance, Semitic, and Mayan languages families, are verb-framed; Chinese as well as non-Romance Indo-European languages like English, Swedish, Russian, Hindi, and Farsi are satellite framed (Talmy 1991, Slobin 1996).

satellite-framed

10.1.3 Morphological Typology

Morphologically, languages are often characterized along two dimensions of variation. The first is the number of morphemes per word, ranging from **isolating** languages like Vietnamese and Cantonese, in which each word generally has one morpheme, to **polysynthetic** languages like Siberian Yupik ("Eskimo"), in which a single word may have very many morphemes, corresponding to a whole sentence in English. The second dimension is the degree to which morphemes are segmentable, ranging from **agglutinative** languages like Turkish, in which morphemes have relatively clean boundaries, to **fusion** languages like Russian, in which a single affix may conflate multiple morphemes, like *-om* in the word *stolom* (table-SG-INSTR-DECL1), which fuses the distinct morphological categories instrumental, singular, and first declension.

isolating

polysynthetic

agglutinative

fusion

Translating between languages with rich morphology requires dealing with structure below the word level, and for this reason modern systems generally use subword models like the wordpiece or BPE models of Section 10.7.1.

10.1.4 Referential density

Finally, languages vary along a typological dimension related to the things they tend to omit. Some languages, like English, require that we use an explicit pronoun when talking about a referent that is given in the discourse. In other languages, however, we can sometimes omit pronouns altogether, as the following example from Spanish shows[1]:

[1] Here we use the ∅-notation; we'll introduce this and discuss this issue further in Chapter 22

(10.6) [El jefe]$_i$ dio con un libro. $\emptyset_i$ Mostró a un descifrador ambulante.
[The boss] came upon a book. [He] showed it to a wandering decoder.

pro-drop

Languages that can omit pronouns are called **pro-drop** languages. Even among the pro-drop languages, there are marked differences in frequencies of omission. Japanese and Chinese, for example, tend to omit far more than does Spanish. This dimension of variation across languages is called the dimension of **referential density**. We say that languages that tend to use more pronouns are more **referentially dense** than those that use more zeros. Referentially sparse languages, like Chinese or Japanese, that require the hearer to do more inferential work to recover antecedents are also called **cold** languages. Languages that are more explicit and make it easier for the hearer are called **hot** languages. The terms *hot* and *cold* are borrowed from Marshall McLuhan's 1964 distinction between hot media like movies, which fill in many details for the viewer, versus cold media like comics, which require the reader to do more inferential work to fill out the representation (Bickel, 2003).

referential density

cold language

hot language

Translating from languages with extensive pro-drop, like Chinese or Japanese, to non-pro-drop languages like English can be difficult since the model must somehow identify each zero and recover who or what is being talked about in order to insert the proper pronoun.

10.2 The Encoder-Decoder Model

encoder-decoder

Encoder-decoder networks, or **sequence-to-sequence** networks, are models capable of generating contextually appropriate, arbitrary length, output sequences. Encoder-decoder networks have been applied to a very wide range of applications including machine translation, summarization, question answering, and dialogue.

The key idea underlying these networks is the use of an **encoder** network that takes an input sequence and creates a contextualized representation of it, often called the **context**. This representation is then passed to a **decoder** which generates a task-specific output sequence. Fig. 10.3 illustrates the architecture

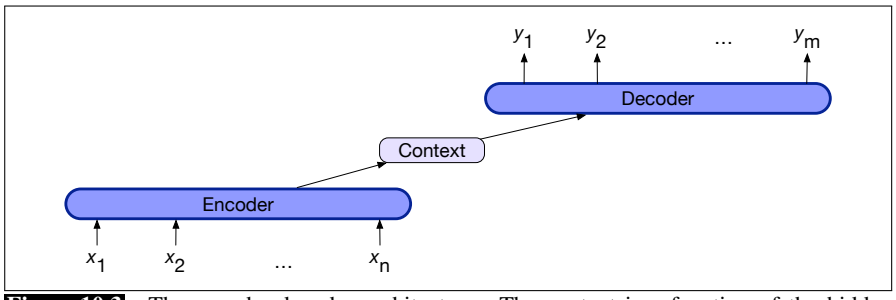


Figure 10.3 The encoder-decoder architecture. The context is a function of the hidden representations of the input, and may be used by the decoder in a variety of ways.

Encoder-decoder networks consist of three components:

1. An **encoder** that accepts an input sequence, x_1^n, and generates a corresponding sequence of contextualized representations, h_1^n. LSTMs, convolutional networks, and Transformers can all be employed as encoders.
2. A **context vector**, c, which is a function of h_1^n, and conveys the essence of the input to the decoder.

3. A **decoder**, which accepts c as input and generates an arbitrary length sequence of hidden states h_1^m, from which a corresponding sequence of output states y_1^m, can be obtained. Just as with encoders, decoders can be realized by any kind of sequence architecture.

10.3 Encoder-Decoder with RNNs

Let's begin by describing an encoder-decoder network based on a pair of RNNs.[2] Recall the conditional RNN language model from Chapter 9 for computing $p(y)$, the probability of a sequence y. Like any language model, we can break down the probability as follows:

$$p(y) = p(y_1)p(y_2|y_1)p(y_3|y_1,y_2)...P(y_m|y_1,...,y_{m-1}) \tag{10.7}$$

At a particular time t, we pass the prefix of $t-1$ tokens through the language model, using forward inference to produce a sequence of hidden states, ending with the hidden state corresponding to the last word of the prefix. We then use the final hidden state of the prefix as our starting point to generate the next token.

More formally, if g is an activation function like *tanh* or ReLU, a function of the input at time t and the hidden state at time $t-1$, and f is a softmax over the set of possible vocabulary items, then at time t the output $\mathbf{y}_t$ and hidden state $\mathbf{h}_t$ are computed as:

$$\mathbf{h}_t = g(\mathbf{h}_{t-1}, \mathbf{x}_t) \tag{10.8}$$
$$\mathbf{y}_t = f(\mathbf{h}_t) \tag{10.9}$$

source
target

We only have to make one slight change to turn this language model with autoregressive generation into a translation model that can translate from a **source** text in one language to a **target** text in a second: add a sentence separation marker at the end of the source text, and then simply concatenate the target text. We briefly introduced this idea of a sentence separator token in Chapter 9 when we considered using a Transformer language model to do summarization, by training a conditional language model.

If we call the source text x and the target text y, we are computing the probability $p(y|x)$ as follows:

$$p(y|x) = p(y_1|x)p(y_2|y_1,x)p(y_3|y_1,y_2,x)...P(y_m|y_1,...,y_{m-1},x) \tag{10.10}$$

Fig. 10.4 shows the setup for a simplified version of the encoder-decoder model (we'll see the full model, which requires **attention**, in the next section).

Fig. 10.4 shows an English source text ("the green witch arrived"), a sentence separator token (<s>, and a Spanish target text ("*llegó la bruja verde*"). To translate a source text, we run it through the network performing forward inference to generate hidden states until we get to the end of the source. Then we begin autoregressive generation, asking for a word in the context of the hidden layer from the end of the source input as well as the end-of-sentence marker. Subsequent words are conditioned on the previous hidden state and the embedding for the last word generated.

[2] Later we'll see how to use pairs of Transformers as well; it's even possible to use separate architectures for the encoder and decoder.

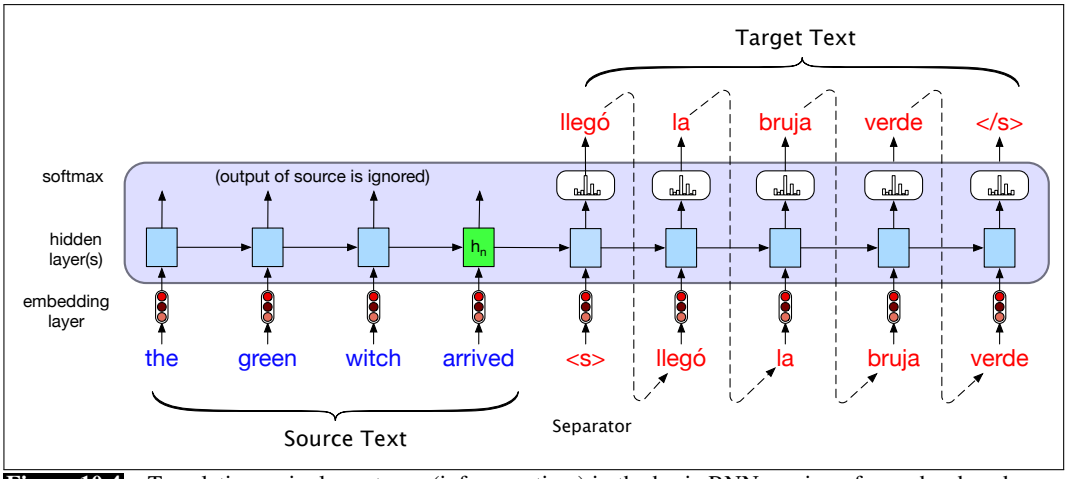


Figure 10.4 Translating a single sentence (inference time) in the basic RNN version of encoder-decoder approach to machine translation. Source and target sentences are concatenated with a separator token in between, and the decoder uses context information from the encoder's last hidden state.

Let's formalize and generalize this model a bit in Fig. 10.5. (To help keep things straight, we'll use the superscripts e and d where needed to distinguish the hidden states of the encoder and the decoder.) The elements of the network on the left process the input sequence x and comprise the **encoder**. While our simplified figure shows only a single network layer for the encoder, stacked architectures are the norm, where the output states from the top layer of the stack are taken as the final representation. A widely used encoder design makes use of stacked biLSTMs where the hidden states from top layers from the forward and backward passes are concatenated as described in Chapter 9 to provide the contextualized representations for each time step.

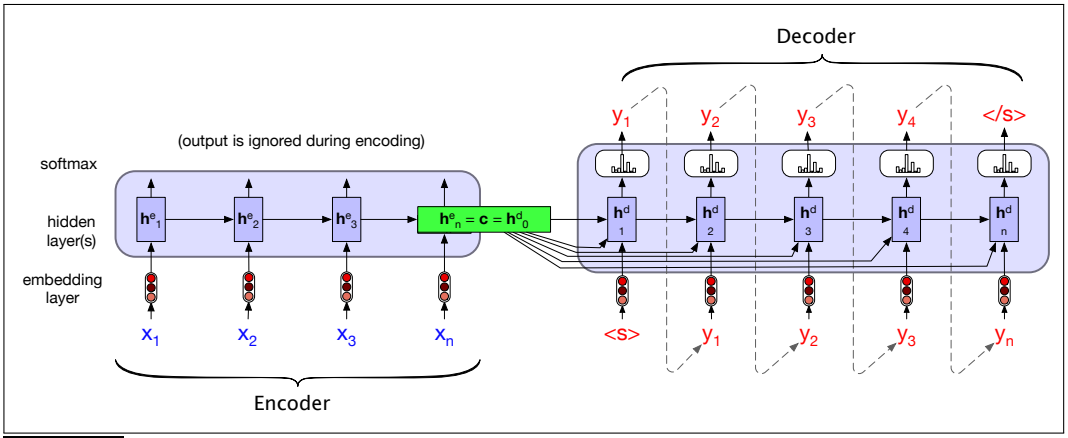


Figure 10.5 A more formal version of translating a sentence at inference time in the basic RNN-based encoder-decoder architecture. The final hidden state of the encoder RNN, h_n^e, serves as the context for the decoder in its role as h_0^d in the decoder RNN.

The entire purpose of the encoder is to generate a contextualized representation of the input. This representation is embodied in the final hidden state of the encoder, $\mathbf{h}_n^e$. This representation, also called $\mathbf{c}$ for **context**, is then passed to the decoder.

The **decoder** network on the right takes this state and uses it to initialize the first

hidden state of the decoder. That is, the first decoder RNN cell uses c as its prior hidden state $\mathbf{h}_0^d$. The decoder autoregressively generates a sequence of outputs, an element at a time, until an end-of-sequence marker is generated. Each hidden state is conditioned on the previous hidden state and the output generated in the previous state.

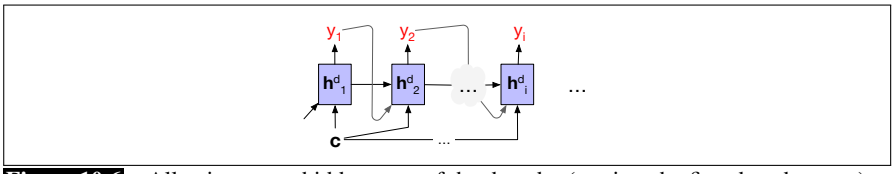


Figure 10.6 Allowing every hidden state of the decoder (not just the first decoder state) to be influenced by the context c produced by the encoder.

One weakness of this approach as described so far is that the influence of the context vector, c, will wane as the output sequence is generated. A solution is to make the context vector c available at each step in the decoding process by adding it as a parameter to the computation of the current hidden state, using the following equation (illustrated in Fig. 10.6):

$$\mathbf{h}_t^d = g(\hat{y}_{t-1}, \mathbf{h}_{t-1}^d, \mathbf{c}) \tag{10.11}$$

Now we're ready to see the full equations for this version of the decoder in the basic encoder-decoder model, with context available at each decoding timestep. Recall that g is a stand-in for some flavor of RNN and $\hat{y}_{t-1}$ is the embedding for the output sampled from the softmax at the previous step:

$$\begin{aligned}
\mathbf{c} &= \mathbf{h}_n^e \\
\mathbf{h}_0^d &= \mathbf{c} \\
\mathbf{h}_t^d &= g(\hat{y}_{t-1}, \mathbf{h}_{t-1}^d, \mathbf{c}) \\
\mathbf{z}_t &= f(\mathbf{h}_t^d) \\
y_t &= \text{softmax}(\mathbf{z}_t)
\end{aligned} \tag{10.12}$$

Finally, as shown earlier, the output y at each time step consists of a softmax computation over the set of possible outputs (the vocabulary, in the case of language modeling or MT). We compute the most likely output at each time step by taking the argmax over the softmax output:

$$\hat{y}_t = \text{argmax}_{w \in V} P(w|x, y_1 \ldots y_{t-1}) \tag{10.13}$$

10.3.1 Training the Encoder-Decoder Model

Encoder-decoder architectures are trained end-to-end, just as with the RNN language models of Chapter 9. Each training example is a tuple of paired strings, a source and a target. Concatenated with a separator token, these source-target pairs can now serve as training data.

For MT, the training data typically consists of sets of sentences and their translations. These can be drawn from standard datasets of aligned sentence pairs, as we'll discuss in Section 10.7.2. Once we have a training set, the training itself proceeds as with any RNN-based language model. The network is given the source text and then starting with the separator token is trained autoregressively to predict the next word, as shown in Fig. 10.7.

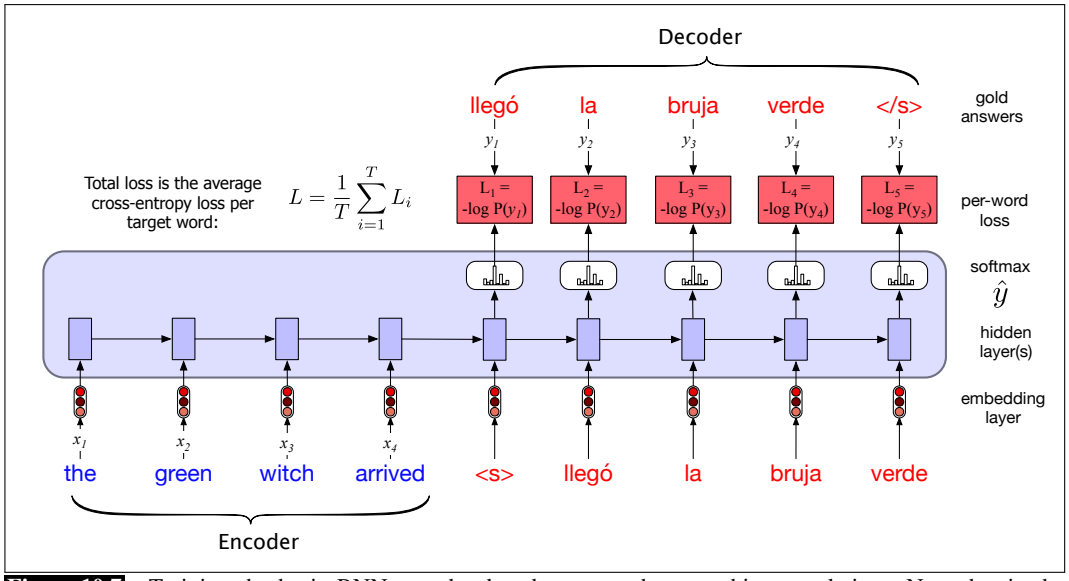


Figure 10.7 Training the basic RNN encoder-decoder approach to machine translation. Note that in the decoder we usually don't propagate the model's softmax outputs $\hat{y}_t$, but use **teacher forcing** to force each input to the correct gold value for training. We compute the softmax output distribution over $\hat{y}$ in the decoder in order to compute the loss at each token, which can then be averaged to compute a loss for the sentence.

teacher forcing

Note the differences between training (Fig. 10.7) and inference (Fig. 10.4) with respect to the outputs at each time step. The decoder during inference uses its own estimated output $\hat{y}_t$ as the input for the next time step x_{t+1}. Thus the decoder will tend to deviate more and more from the gold target sentence as it keeps generating more tokens. In training, therefore, it is more common to use **teacher forcing** in the decoder. Teacher forcing means that we force the system to use the gold target token from training as the next input x_{t+1}, rather than allowing it to rely on the (possibly erroneous) decoder output $\hat{y}_t$. This speeds up training.

10.4 Attention

The simplicity of the encoder-decoder model is its clean separation of the encoder—which builds a representation of the source text—from the decoder, which uses this context to generate a target text. In the model as we've described it so far, this context vector is h_n, the hidden state of the last (nth) time step of the source text. This final hidden state is thus acting as a **bottleneck**: it must represent absolutely everything about the meaning of the source text, since the only thing the decoder knows about the source text is what's in this context vector (Fig. 10.8). Information at the beginning of the sentence, especially for long sentences, may not be equally well represented in the context vector.

attention mechanism

The **attention mechanism** is a solution to the bottleneck problem, a way of allowing the decoder to get information from *all* the hidden states of the encoder, not just the last hidden state.

In the attention mechanism, as in the vanilla encoder-decoder model, the context vector $\mathbf{c}$ is a single vector that is a function of the hidden states of the encoder, that is, $\mathbf{c} = f(\mathbf{h}_1^e \ldots \mathbf{h}_n^e)$. Because the number of hidden states varies with the size of the

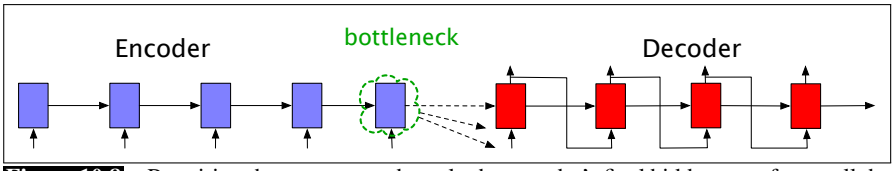


Figure 10.8 Requiring the context c to be only the encoder's final hidden state forces all the information from the entire source sentence to pass through this representational bottleneck.

input, we can't use the entire tensor of encoder hidden state vectors directly as the context for the decoder.

The idea of attention is instead to create the single fixed-length vector c by taking a weighted sum of all the encoder hidden states. The weights focus on ('attend to') a particular part of the source text that is relevant for the token the decoder is currently producing. Attention thus replaces the static context vector with one that is dynamically derived from the encoder hidden states, different for each token in decoding.

This context vector, $\mathbf{c}_i$, is generated anew with each decoding step i and takes all of the encoder hidden states into account in its derivation. We then make this context available during decoding by conditioning the computation of the current decoder hidden state on it (along with the prior hidden state and the previous output generated by the decoder), as we see in this equation (and Fig. 10.9):

$$\mathbf{h}_i^d = g(\hat{y}_{i-1}, \mathbf{h}_{i-1}^d, \mathbf{c}_i) \tag{10.14}$$

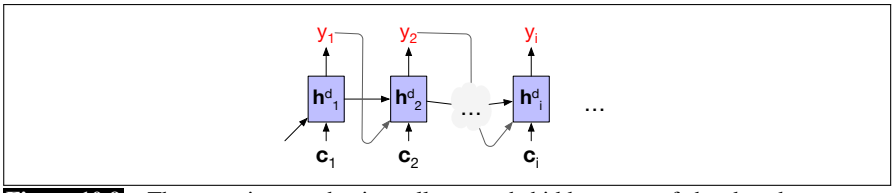


Figure 10.9 The attention mechanism allows each hidden state of the decoder to see a different, dynamic, context, which is a function of all the encoder hidden states.

The first step in computing $\mathbf{c}_i$ is to compute how much to focus on each encoder state, how *relevant* each encoder state is to the decoder state captured in $\mathbf{h}_{i-1}^d$. We capture relevance by computing— at each state i during decoding—a $score(\mathbf{h}_{i-1}^d, \mathbf{h}_j^e)$ for each encoder state j.

dot-product attention

The simplest such score, called **dot-product attention**, implements relevance as similarity: measuring how similar the decoder hidden state is to an encoder hidden state, by computing the dot product between them:

$$score(\mathbf{h}_{i-1}^d, \mathbf{h}_j^e) = \mathbf{h}_{i-1}^d \cdot \mathbf{h}_j^e \tag{10.15}$$

The score that results from this dot product is a scalar that reflects the degree of similarity between the two vectors. The vector of these scores across all the encoder hidden states gives us the relevance of each encoder state to the current step of the decoder.

To make use of these scores, we'll normalize them with a softmax to create a vector of weights, α_{ij}, that tells us the proportional relevance of each encoder hidden

state j to the prior hidden decoder state, h^d_{i-1}.

$$\begin{aligned}\alpha_{ij} &= \text{softmax}(score(\mathbf{h}^d_{i-1}, \mathbf{h}^e_j) \;\; \forall j \in e) \\ &= \frac{\exp(score(\mathbf{h}^d_{i-1}, \mathbf{h}^e_j)}{\sum_k \exp(score(\mathbf{h}^d_{i-1}, \mathbf{h}^e_k))}\end{aligned} \tag{10.16}$$

Finally, given the distribution in α, we can compute a fixed-length context vector for the current decoder state by taking a weighted average over all the encoder hidden states.

$$\mathbf{c}_i = \sum_j \alpha_{ij}\, \mathbf{h}^e_j \tag{10.17}$$

With this, we finally have a fixed-length context vector that takes into account information from the entire encoder state that is dynamically updated to reflect the needs of the decoder at each step of decoding. Fig. 10.10 illustrates an encoder-decoder network with attention, focusing on the computation of one context vector $\mathbf{c}_i$.

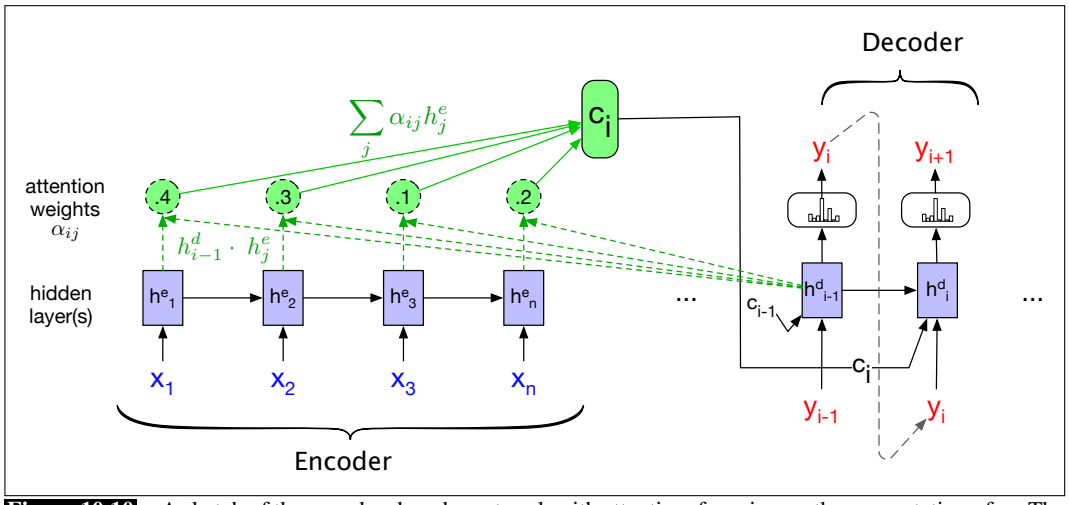


Figure 10.10 A sketch of the encoder-decoder network with attention, focusing on the computation of $\mathbf{c}_i$. The context value $\mathbf{c}_i$ is one of the inputs to the computation of $\mathbf{h}^d_i$. It is computed by taking the weighted sum of all the encoder hidden states, each weighted by their dot product with the prior decoder hidden state $\mathbf{h}^d_{i-1}$.

It's also possible to create more sophisticated scoring functions for attention models. Instead of simple dot product attention, we can get a more powerful function that computes the relevance of each encoder hidden state to the decoder hidden state by parameterizing the score with its own set of weights, $\mathbf{W}_s$.

$$score(\mathbf{h}^d_{i-1}, \mathbf{h}^e_j) = \mathbf{h}^d_{t-1} \mathbf{W}_s \mathbf{h}^e_j$$

The weights W_s, which are then trained during normal end-to-end training, give the network the ability to learn which aspects of similarity between the decoder and encoder states are important to the current application. This bilinear model also allows the encoder and decoder to use different dimensional vectors, whereas the simple dot-product attention requires that the encoder and decoder hidden states have the same dimensionality.

10.5 Beam Search

The decoding algorithm we gave above for generating translations has a problem (as does the autoregressive generation we introduced in Chapter 9 for generating from a conditional language model). Recall that algorithm: at each time step in decoding, the output y_t is chosen by computing a softmax over the set of possible outputs (the vocabulary, in the case of language modeling or MT), and then choosing the highest probability token (the argmax):

$$\hat{y}_t = \text{argmax}_{w \in V} P(w|x, y_1 \ldots y_{t-1}) \tag{10.18}$$

greedy

Choosing the single most probable token to generate at each step is called **greedy** decoding; a greedy algorithm is one that make a choice that is locally optimal, whether or not it will turn out to have been the best choice with hindsight.

Indeed, greedy search is not optimal, and may not find the highest probability translation. The problem is that the token that looks good to the decoder now might turn out later to have been the wrong choice!

search tree

Let's see this by looking at the **search tree**, a graphical representation of the choices the decoder makes in searching for the best translation, in which we view the decoding problem as a heuristic state-space search and systematically explore the space of possible outputs. In such a search tree, the branches are the actions, in this case the action of generating a token, and the nodes are the states, in this case the state of having generated a particular prefix. We are searching for the best action sequence, i.e. the target string with the highest probability. Fig. 10.11 demonstrates the problem, using a made-up example. Notice that the most probable sequence is *ok ok* </s> (with a probability of .4*.7*1.0), but a greedy search algorithm will fail to find it, because it incorrectly chooses *yes* as the first word since it has the highest local probability.

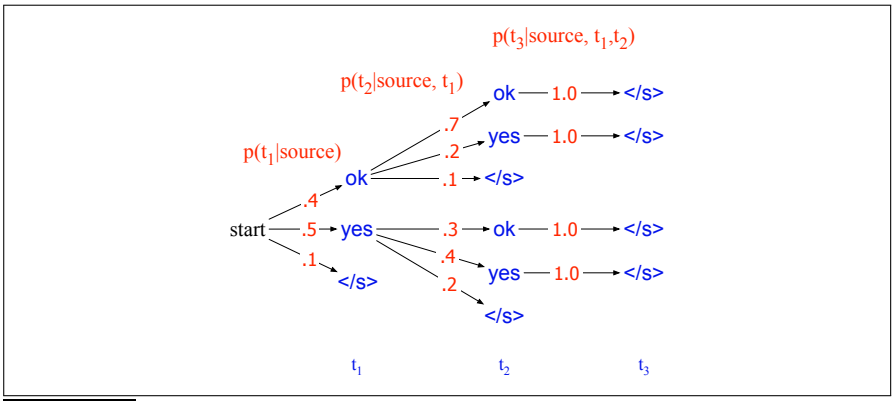


Figure 10.11 A search tree for generating the target string $T = t_1, t_2, \ldots$ from the vocabulary $V = \{\text{yes}, \text{ok}, \text{<s>}\}$, given the source string, showing the probability of generating each token from that state. Greedy search would choose *yes* at the first time step followed by *yes*, instead of the globally most probable sequence *ok ok*.

Recall from Chapter 8 that for part-of-speech tagging we used dynamic programming search (the Viterbi algorithm) to address this problem. Unfortunately, dynamic programming is not applicable to generation problems with long-distance dependencies between the output decisions. The only method guaranteed to find the

best solution is exhaustive search: computing the probability of every one of the V^T possible sentences (for some length value T) which is obviously too slow.

beam search

Instead, decoding in MT and other sequence generation problems generally uses a method called **beam search**. In beam search, instead of choosing the best token to generate at each timestep, we keep k possible tokens at each step. This fixed-size memory footprint k is called the **beam width**, on the metaphor of a flashlight beam that can be parameterized to be wider or narrower.

beam width

Thus at the first step of decoding, we compute a softmax over the entire vocabulary, assigning a probability to each word. We then select the k-best options from this softmax output. These initial k outputs are the search frontier and these k initial words are called **hypotheses**. A hypothesis is an output sequence, a translation-so-far, together with its probability.

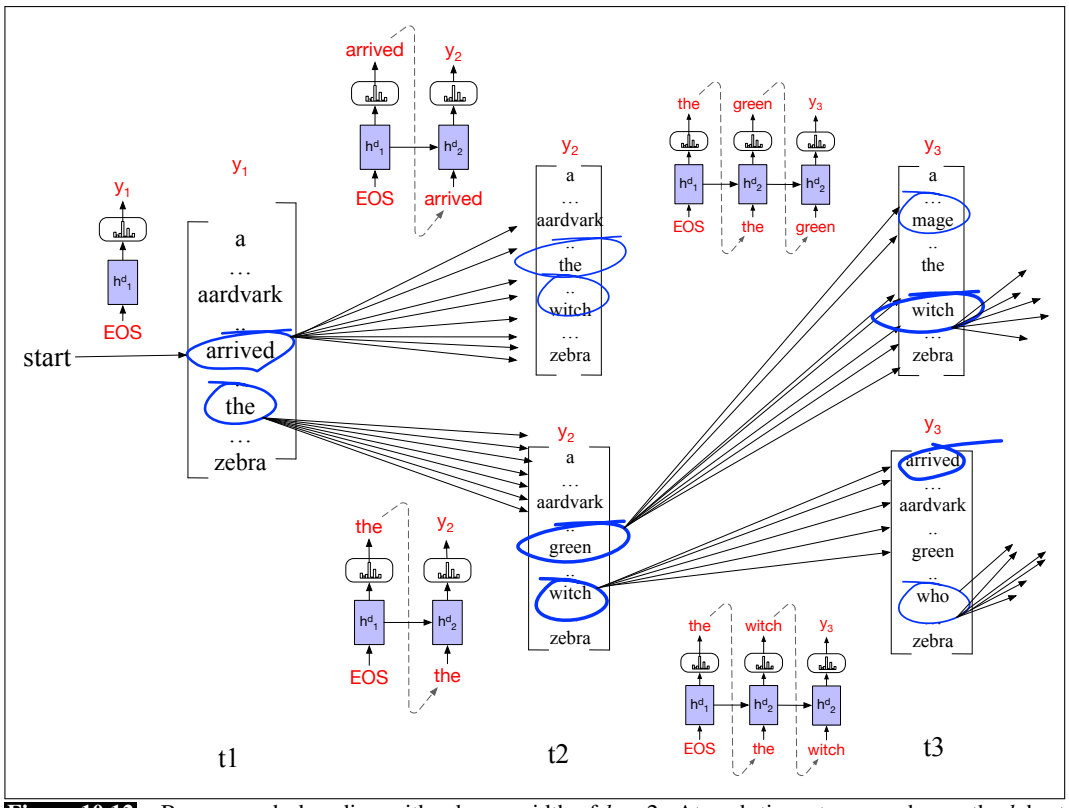


Figure 10.12 Beam search decoding with a beam width of $k = 2$. At each time step, we choose the k best hypotheses, compute the V possible extensions of each hypothesis, score the resulting $k * V$ possible hypotheses and choose the best k to continue. At time 1, the frontier is filled with the best 2 options from the initial state of the decoder: *arrived* and *the*. We then extend each of those, compute the probability of all the hypotheses so far (*arrived the*, *arrived aardvark*, *the green*, *the witch*) and compute the best 2 (in this case *the green* and *the witch*) to be the search frontier to extend on the next step. On the arcs we show the decoders that we run to score the extension words (although for simplicity we haven't shown the context value c_i that is input at each step).

At subsequent steps, each of the k best hypotheses is extended incrementally by being passed to distinct decoders, which each generate a softmax over the entire vocabulary to extend the hypothesis to every possible next token. Each of these $k * V$ hypotheses is scored by $P(y_i|x, y_{<i})$: the product of the probability of current word choice multiplied by the probability of the path that led to it. We then prune the $k * V$ hypotheses down to the k best hypotheses, so there are never more than k hypotheses

at the frontier of the search, and never more than k decoders.

Fig. 10.12 illustrates this process with a beam width of 2.

This process continues until a </s> is generated indicating that a complete candidate output has been found. At this point, the completed hypothesis is removed from the frontier and the size of the beam is reduced by one. The search continues until the beam has been reduced to 0. The result will be k hypotheses.

Let's see how the scoring works in detail, scoring each node by its log probability. Recall from Eq. 10.10 that we can use the chain rule of probability to break down $p(y|x)$ into the product of the probability of each word given its prior context, which we can turn into a sum of logs (for an output string of length t):

$$\begin{aligned} score(y) &= \log P(y|x) \\ &= \log\left(P(y_1|x)P(y_2|y_1,x)P(y_3|y_1,y_2,x)...P(y_t|y_1,...,y_{t-1},x)\right) \\ &= \sum_{i=1}^{t} \log P(y_i|y_1,...,y_{i-1},x) \end{aligned} \tag{10.19}$$

Thus at each step, to compute the probability of a partial translation, we simply add the log probability of the prefix translation so far to the log probability of generating the next token. Fig. 10.13 shows the scoring for the example sentence shown in Fig. 10.12, using some simple made-up probabilities. Log probabilities are negative or 0, and the max of two log probabilities is the one that is greater (closer to 0).

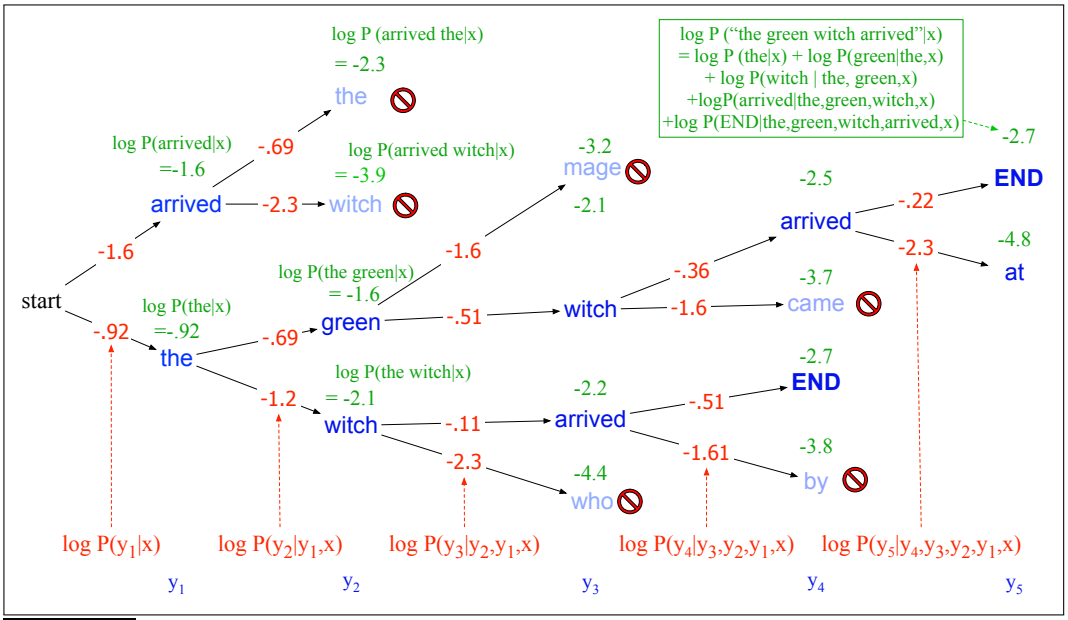


Figure 10.13 Scoring for beam search decoding with a beam width of $k = 2$. We maintain the log probability of each hypothesis in the beam by incrementally adding the logprob of generating each next token. Only the top k paths are extended to the next step.

Fig. 10.14 gives the algorithm.

One problem arises from the fact that the completed hypotheses may have different lengths. Because models generally assign lower probabilities to longer strings, a naive algorithm would also choose shorter strings for y. This was not an issue during the earlier steps of decoding; due to the breadth-first nature of beam search all the hypotheses being compared had the same length. The usual solution to this is

```
function BEAMDECODE(c, beam_width) returns best paths

  y_0, h_0 ← 0
  path ← ()
  complete_paths ← ()
  state ← (c, y_0, h_0, path)          ;initial state
  frontier ← ⟨state⟩                   ;initial frontier

  while frontier contains incomplete paths and beamwidth > 0
    extended_frontier ← ⟨⟩
    for each state ∈ frontier do
      y ← DECODE(state)
      for each word i ∈ Vocabulary do
        successor ← NEWSTATE(state, i, y_i)
        extended_frontier ← ADDTOBEAM(successor, extended_frontier,
                                      beam_width)

    for each state in extended_frontier do
      if state is complete do
        complete_paths ← APPEND(complete_paths, state)
        extended_frontier ← REMOVE(extended_frontier, state)
        beam_width ← beam_width - 1
    frontier ← extended_frontier

  return completed_paths

function NEWSTATE(state, word, word_prob) returns new state

function ADDTOBEAM(state, frontier, width) returns updated frontier

  if LENGTH(frontier) < width then
    frontier ← INSERT(state, frontier)
  else if SCORE(state) > SCORE(WORSTOF(frontier))
    frontier ← REMOVE(WORSTOF(frontier))
    frontier ← INSERT(state, frontier)
  return frontier
```

Figure 10.14 Beam search decoding.

to apply some form of length normalization to each of the hypotheses, for example simply dividing the negative log probability by the number of words:

$$score(y) = -\log P(y|x) = \frac{1}{T}\sum_{i=1}^{t} -\log P(y_i|y_1,...,y_{i-1},x) \tag{10.20}$$

Beam search is common in large production MT systems, generally with beam widths k between 5 and 10. What do we do with the resulting k hypotheses? In some cases, all we need from our MT algorithm is the single best hypothesis, so we can return that. In other cases our downstream application might want to look at all k hypotheses, so we can pass them all (or a subset) to the downstream application with their respective scores.

10.6 Encoder-Decoder with Transformers

The encoder-decoder architecture can also be implemented using transformers (rather than RNN/LSTMs) as the component modules. At a high-level, the architecture, sketched in Fig. 10.15, is quite similar to what we saw for RNNs. It consists of an encoder that takes the source language input words $\mathbf{X} = \mathbf{x}_1, ..., \mathbf{x}_T$ and maps them to an output representation $\mathbf{H}^{enc} = \mathbf{h}_1, ..., \mathbf{h}_T$; usually via $N = 6$ stacked encoder blocks. The decoder, just like the encoder-decoder RNN, is essentially a conditional language model that attends to the encoder representation and generates the target words one by one, at each timestep conditioning on the source sentence and the previously generated target language words.

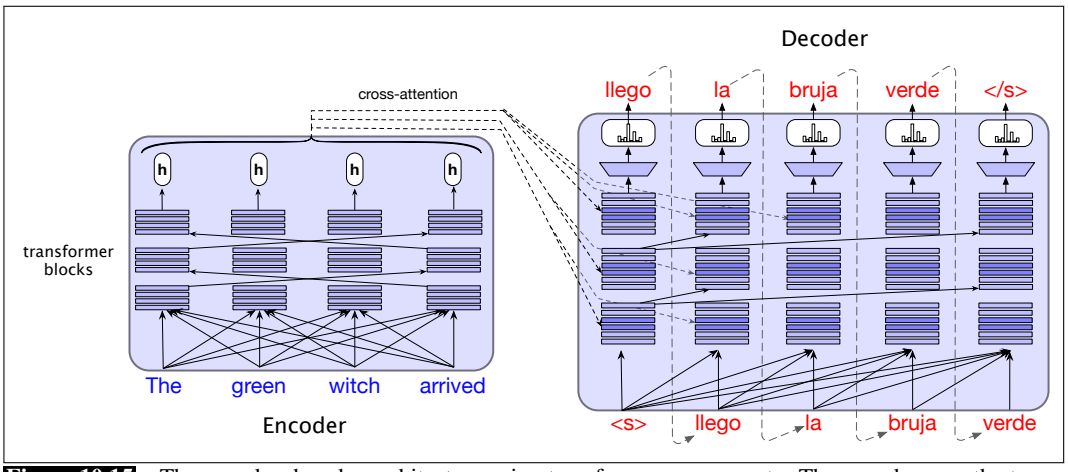


Figure 10.15 The encoder-decoder architecture using transformer components. The encoder uses the transformer blocks we saw in Chapter 9, while the decoder uses a more powerful block with an extra encoder-decoder attention layer. The final output of the encoder $\mathbf{H}^{enc} = \mathbf{h}_1, ..., \mathbf{h}_T$ is used to form the $\mathbf{K}$ and $\mathbf{V}$ inputs to the cross-attention layer in each decoder block.

But the components of the architecture differ somewhat from the RNN and also from the transformer block we've seen. First, in order to attend to the source language, the transformer blocks in the decoder has an extra **cross-attention** layer. Recall that the transformer block of Chapter 9 consists of a self-attention layer that attends to the input from the previous layer, followed by layer norm, a feed forward layer, and another layer norm. The decoder transformer block includes an extra layer with a special kind of attention, **cross-attention** (also sometimes called **encoder-decoder attention** or **source attention**). Cross-attention has the same form as the multi-headed self-attention in a normal transformer block, except that while the queries as usual come from the previous layer of the decoder, the keys and values come from the output of the *encoder*.

cross-attention

That is, the final output of the encoder $\mathbf{H}^{enc} = \mathbf{h}_1, ..., \mathbf{h}_t$ is multiplied by the cross-attention layer's key weights $\mathbf{W}^{\mathbf{K}}$ and value weights $\mathbf{W}^{\mathbf{V}}$, but the output from the prior decoder layer $\mathbf{H}^{dec[i-1]}$ is multiplied by the cross-attention layer's query weights $\mathbf{W}^{\mathbf{Q}}$:

$$\mathbf{Q} = \mathbf{W}^{\mathbf{Q}}\mathbf{H}^{dec[i-1]}; \;\; \mathbf{K} = \mathbf{W}^{\mathbf{K}}\mathbf{H}^{enc}; \;\; \mathbf{V} = \mathbf{W}^{\mathbf{V}}\mathbf{H}^{enc} \tag{10.21}$$

$$\mathit{CrossAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \;=\; \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\intercal}}{\sqrt{d_k}}\right)\mathbf{V} \tag{10.22}$$

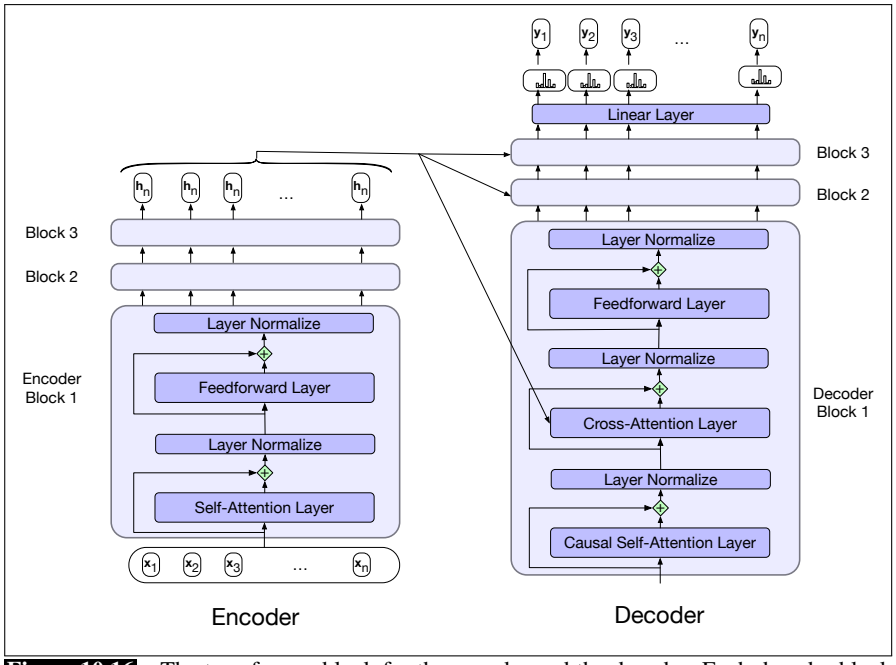


Figure 10.16 The transformer block for the encoder and the decoder. Each decoder block has an extra cross-attention layer, which uses the output of the final encoder layer $H^{enc} = h_1, ..., h_t$ to produce its key and value vectors.

The cross attention thus allows the decoder to attend to each of the source language words as projected into the the entire encoder final output representations. The other attention layer in each decoder block, the self-attention layer, is the same causal (left-to-right) self-attention that we saw in Chapter 9. The self-attention in the encoder, however, is allowed to look ahead at the entire source language text.

In training, just as for RNN encoder-decoders, we use teacher forcing, and train autoregressively, at each time step predicting the next token in the target language, using cross-entropy loss.

10.7 Some practical details on building MT systems

10.7.1 Tokenization

wordpiece

Machine translation systems generally use a fixed vocabulary, A common way to generate this vocabulary is with the **BPE** or **wordpiece** algorithms sketched in Chapter 2. Generally a shared vocabulary is used for the source and target languages, which makes it easy to copy tokens (like names) from source to target, so we build the wordpiece/BPE lexicon on a corpus that contains both source and target language data. Wordpieces use a special symbol at the beginning of each token; here's a resulting tokenization from the Google MT system (Wu et al., 2016):

words: Jet makers feud over seat width with big orders at stake
wordpieces: _J et _makers _fe ud _over _seat _width _with _big _orders _at _stake

We gave the BPE algorithm in detail in Chapter 2; here are more details on the wordpiece algorithm, which is given a training corpus and a desired vocabulary size

V, and proceeds as follows:

1. Initialize the wordpiece lexicon with characters (for example a subset of Unicode characters, collapsing all the remaining characters to a special unknown character token).
2. Repeat until there are V wordpieces:
   (a) Train an n-gram language model on the training corpus, using the current set of wordpieces.
   (b) Consider the set of possible new wordpieces made by concatenating two wordpieces from the current lexicon. Choose the one new wordpiece that most increases the language model probability of the training corpus.

A vocabulary of 8K to 32K word pieces is commonly used.

10.7.2 MT corpora

parallel corpus

Machine translation models are trained on a **parallel corpus**, sometimes called a **bitext**, a text that appears in two (or more) languages. Large numbers of parallel corpora are available. Some are governmental; the **Europarl** corpus (Koehn, 2005), extracted from the proceedings of the European Parliament, contains between 400,000 and 2 million sentences each from 21 European languages. The United Nations Parallel Corpus contains on the order of 10 million sentences in the six official languages of the United Nations (Arabic, Chinese, English, French, Russian, Spanish) Ziemski et al. (2016). Other parallel corpora have been made from movie and TV subtitles, like the **OpenSubtitles** corpus (Lison and Tiedemann, 2016), or from general web text, like the **ParaCrawl** corpus of 223 million sentence pairs between 23 EU languages and English extracted from the CommonCrawl Bañón et al. (2020).

Europarl

Sentence alignment

Standard training corpora for MT come as aligned pairs of sentences. When creating new corpora, for example for underresourced languages or new domains, these sentence alignments must be created. Fig. 10.17 gives a sample hypothetical sentence alignment.

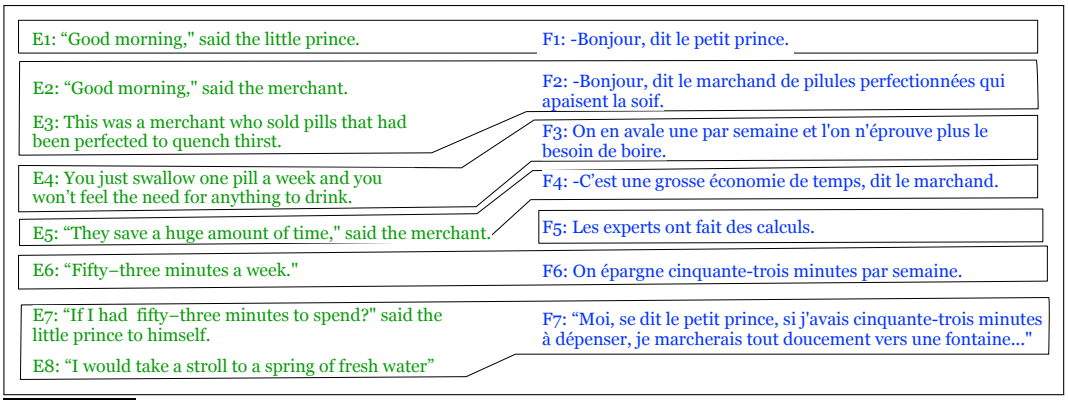


Figure 10.17 A sample alignment between sentences in English and French, with sentences extracted from Antoine de Saint-Exupery’s *Le Petit Prince* and a hypothetical translation. Sentence alignment takes sentences $e_1, ..., e_n$, and $f_1, ..., f_n$ and finds minimal sets of sentences that are translations of each other, including single sentence mappings like (e_1,f_1), (e_4,f_3), (e_5,f_4), (e_6,f_6) as well as 2-1 alignments (e_2/e_3,f_2), (e_7/e_8,f_7), and null alignments (f_5).

Given two documents that are translations of each other, we generally need two steps to produce sentence alignments:

- a cost function that takes a span of source sentences and a span of target sentences and returns a score measuring how likely these spans are to be translations.
- an alignment algorithm that takes these scores to find a good alignment between the documents.

Since it is possible to induce multilingual sentence embeddings (Artetxe and Schwenk, 2019), cosine similarity of such embeddings provides a natural scoring function (Schwenk, 2018). Thompson and Koehn (2019) give the following cost function between two sentences or spans x,y from the source and target documents respectively:

$$c(x,y) = \frac{(1-\cos(x,y))\text{nSents}(x)\ \text{nSents}(y)}{\sum_{s=1}^{S} 1-\cos(x,y_s) + \sum_{s=1}^{S} 1-\cos(x_s,y)} \tag{10.23}$$

where nSents() gives the number of sentences (this biases the metric toward many alignments of single sentences instead of aligning very large spans). The denominator helps to normalize the similarities, and so $x_1,...,x_S,y_1,...,y_S$, are randomly selected sentences sampled from the respective documents.

Usually dynamic programming is used as the alignment algorithm (Gale and Church, 1993), in a simple extension of the minimum edit distance algorithm we introduced in Chapter 2.

Finally, it's helpful to do some corpus cleanup by removing noisy sentence pairs. This can involve handwritten rules to remove low-precision pairs (for example removing sentences that are too long, too short, have different URLs, or even pairs that are too similar, suggesting that they were copies rather than translations). Or pairs can be ranked by their multilingual embedding cosine score and low-scoring pairs discarded.

10.7.3 Backtranslation

We're often short of data for training MT models, since parallel corpora may be limited for particular languages or domains. However, often we can find a large monolingual corpus, to add to the smaller parallel corpora that are available.

backtranslation

Backtranslation is a way of making use of monolingual corpora in the target language by creating synthetic bitexts. In backtranslation, we train an intermediate target-to-source MT system on the small bitext to translate the monolingual target data to the source language. Now we can add this synthetic bitext (natural target sentences, aligned with MT-produced source sentences) to our training data, and retrain our source-to-target MT model. For example suppose we want to translate from Navajo to English but only have a small Navajo-English bitext, although of course we can find lots of monolingual English data. We use the small bitext to build an MT engine going the other way (from English to Navajo). Once we translate the monolingual English text to Navajo, we can add this synthetic Navajo/English bitext to our training data.

Monte Carlo search

Backtranslation has various parameters. One is how we generate the backtranslated data; we can run the decoder in greedy inference, or use beam search. Or we can do sampling, or **Monte Carlo search**. In Monte Carlo decoding, at each timestep, instead of always generating the word with the highest softmax probability, we roll a weighted die, and use it to choose the next word according to its

softmax probability. This works just like the sampling algorithm we saw in Chapter 3 for generating random sentences from n-gram language models. Imagine there are only 4 words and the softmax probability distribution at time t is (*the*: 0.6, *green*: 0.2, *a*: 0.1, *witch*: 0.1). We roll a weighted die, with the 4 sides weighted 0.6, 0.2, 0.1, and 0.1, and chose the word based on which side comes up. Another parameter is the ratio of backtranslated data to natural bitext data; we can choose to upsample the bitext data (include multiple copies of each sentence).

In general backtranslation works surprisingly well; one estimate suggests that a system trained on backtranslated text gets about 2/3 of the gain as would training on the same amount of natural bitext (Edunov et al., 2018).

10.8 MT Evaluation

Translations are evaluated along two dimensions:

adequacy
1. **adequacy:** how well the translation captures the exact meaning of the source sentence. Sometimes called **faithfulness** or **fidelity**.

fluency
2. **fluency:** how fluent the translation is in the target language (is it grammatical, clear, readable, natural).

Using humans to evaluate is most accurate, but automatic metrics are also used for convenience.

10.8.1 Using Human Raters to Evaluate MT

The most accurate evaluations use human raters, such as online crowdworkers, to evaluate each translation along the two dimensions. For example, along the dimension of **fluency**, we can ask how intelligible, how clear, how readable, or how natural the MT output (the target text) is. We can give the raters a scale, for example, from 1 (totally unintelligible) to 5 (totally intelligible, or 1 to 100, and ask them to rate each sentence or paragraph of the MT output.

We can do the same thing to judge the second dimension, **adequacy**, using raters to assign scores on a scale. If we have bilingual raters, we can give them the source sentence and a proposed target sentence, and rate, on a 5-point or 100-point scale, how much of the information in the source was preserved in the target. If we only have monolingual raters but we have a good human translation of the source text, we can give the monolingual raters the human reference translation and a target machine translation and again rate how much information is preserved. An alternative is to

ranking
do **ranking**: give the raters a pair of candidate translations, and ask them which one they prefer.

Training of human raters (who are often online crowdworkers) is essential; raters without translation expertise find it difficult to separate fluency and adequacy, and so training includes examples carefully distinguishing these. Raters often disagree (sources sentences may be ambiguous, raters will have different world knowledge, raters may apply scales differently). It is therefore common to remove outlier raters, and (if we use a fine-grained enough scale) normalizing raters by subtracting the mean from their scores and dividing by the variance.

10.8.2 Automatic Evaluation

While humans produce the best evaluations of machine translation output, running a human evaluation can be time consuming and expensive. For this reason automatic metrics are often used. Automatic metrics are less accurate than human evaluation, but can help test potential system improvements, and even be used as an automatic loss function for training. In this section we introduce two families of such metrics, those based on character- or word-overlap and those based on embedding similarity.

Automatic Evaluation by Character Overlap: chrF

chrF

The simplest and most robust metric for MT evaluation is called **chrF**, which stands for **character F-score** (Popović, 2015). chrF (along with many other earlier related metrics like BLEU, METEOR, TER, and others) is based on a simple intuition derived from the pioneering work of Miller and Beebe-Center (1956): a good machine translation will tend to contain characters and words that occur in a human translation of the same sentence. Consider a test set from a parallel corpus, in which each source sentence has both a gold human target translation and a candidate MT translation we'd like to evaluate. The chrF metric ranks each MT target sentence by a function of the number of character n-gram overlaps with the human translation.

Given the hypothesis and the reference, chrF is given a parameter k indicating the length of character n-grams to be considered, and computes the average of the k precisions (unigram precision, bigram, and so on) and the average of the k recalls (unigram recall, bigram recall, etc.):

chrP percentage of character 1-grams, 2-grams, ..., k-grams in the hypothesis that occur in the reference, averaged.

chrR percentage of character 1-grams, 2-grams,..., k-grams in the reference that occur in the hypothesis, averaged.

The metric then computes an F-score by combining chrP and chrR using a weighting parameter β. It is common to set $\beta = 2$, thus weighing recall twice as much as precision:

$$\text{chrF}\beta = (1+\beta^2)\frac{\text{chrP}\cdot\text{chrR}}{\beta^2\cdot\text{chrP}+\text{chrR}} \tag{10.24}$$

For $\beta = 2$, that would be:

$$\text{chrF2} = \frac{5\cdot\text{chrP}\cdot\text{chrR}}{4\cdot\text{chrP}+\text{chrR}}$$

For example, consider two hypotheses that we'd like to score against the reference translation *witness for the past*. Here are the hypotheses along with chrF values computed using parameters $k = \beta = 2$ (in real examples, k would be a higher number like 6):

REF: `witness for the past,`	
HYP1: `witness of the past,`	chrF2,2 = .86
HYP2: `past witness`	chrF2,2 = .62

Let's see how we computed that chrF value for HYP1 (we'll leave the computation of the chrF value for HYP2 as an exercise for the reader). First, chrF ignores spaces, so we'll remove them from both the reference and hypothesis:

REF: `witnessforthepast,` (18 unigrams, 17 bigrams)
HYP1: `witnessofthepast,` (17 unigrams, 16 bigrams)

Next let's see how many unigrams and bigrams match between the reference and hypothesis:

unigrams that match: `w i t n e s s f o t h e p a s t ,` (17 unigrams)
bigrams that match: `wi it tn ne es ss th he ep pa as st t,` (13 bigrams)

We use that to compute the unigram and bigram precisions and recalls:

unigram P:	17/17 = 1	unigram R:	17/18 = .944
bigram P:	13/16 = .813	bigram R:	13/17 = .765

Finally we average to get chrP and chrR, and compute the F-score:

$$\begin{aligned} \text{chrP} &= (17/17 + 13/16)/2 = .906 \\ \text{chrR} &= (17/18 + 13/17)/2 = .855 \\ \text{chrF2,2} &= 5\frac{\text{chrP} * \text{chrR}}{4\text{chrP} + \text{chrR}} = .86 \end{aligned}$$

chrF is simple, robust, and correlates very well with human judgments in many languages (Kocmi et al., 2021). There are various alternative overlap metrics. For example, before the development of chrF, it was common to use a word-based overlap metric called **BLEU** (for BiLingual Evaluation Understudy), that is purely precision-based rather than combining precision and recall (Papineni et al., 2002). The BLEU score for a corpus of candidate translation sentences is a function of the **n-gram word precision** over all the sentences combined with a brevity penalty computed over the corpus as a whole. Because BLEU is a word-based metric, it is very sensitive to word tokenization, making it difficult to compare across situations, and doesn't work as well in languages with complex morphology.

Statistical Significance Testing for MT evals

Character or word overlap-based metrics like chrF (or BLEU, or etc.) are mainly used to compare two systems, with the goal of answering questions like: did the new algorithm we just invented improve our MT system? To know if the difference between the chrF scores of two MT systems is a significant difference, we use the paired bootstrap test, or the similar randomization test.

To get a confidence interval on a single chrF score using the bootstrap test, recall from Section **??** that we take our test set (or devset) and create thousands of pseudo-testsets by repeatedly sampling with replacement from the original test set. We now compute the chrF score of each of the pseudo-testsets. If we drop the top 2.5% and bottom 2.5% of the scores, the remaining scores will give us the 95% confidence interval for the chrF score of our system.

To compare two MT systems A and B, we draw the same set of pseudo-testsets, and compute the chrF scores for each of them. We then compute the percentage of pseudo-test-sets in which A has a higher chrF score than B.

chrF: Limitations

While automatic character and word-overlap metrics like chrF or BLEU are useful, they have important limitations. chrF is very local: a large phrase that is moved around might barely change the chrF score at all, and chrF can't evaluate cross-sentence properties of a document like its discourse coherence (Chapter 22). chrF and similar automatic metrics also do poorly at comparing very different kinds of systems, such as comparing human-aided translation against machine translation, or

different machine translation architectures against each other (Callison-Burch et al., 2006). Instead, automatic overlap metrics like chrF are most appropriate when evaluating changes to a single system.

10.8.3 Automatic Evaluation: Embedding-Based Methods

The chrF metric is based on measuring the exact character n-grams a human reference and candidate machine translation have in common. However, this criterion is overly strict, since a good translation may use alternate words or paraphrases. A solution first pioneered in early metrics like METEOR (Banerjee and Lavie, 2005) was to allow synonyms to match between the reference x and candidate $\tilde{x}$. More recent metrics use BERT or other embeddings to implement this intuition.

For example, in some situations we might have datasets that have human assessments of translation quality. Such datasets consists of tuples $(x, \tilde{x}, r)$, where $x = (x_1, \ldots, x_n)$ is a reference translation, $\tilde{x} = (\tilde{x}_1, \ldots, \tilde{x}_m)$ is a candidate machine translation, and $r \in \mathbb{R}$ is a human rating that expresses the quality of $\tilde{x}$ with respect to x. Given such data, algorithms like COMET (Rei et al., 2020) BLEURT (Sellam et al., 2020) train a predictor on the human-labeled datasets, for example by passing x and $\tilde{x}$ through a version of BERT (trained with extra pretraining, and then fine-tuned on the human-labeled sentences), followed by a linear layer that is trained to predict r. The output of such models correlates highly with human labels.

In other cases, however, we don't have such human-labeled datasets. In that case we can measure the similarity of x and $\tilde{x}$ by the similarity of their embeddings. The BERTSCORE algorithm (Zhang et al., 2020) shown in Fig. 10.18, for example, passes the reference x and the candidate $\tilde{x}$ through BERT, computing a BERT embedding for each token x_i and $\tilde{x}_j$. Each pair of tokens $(x_i, \tilde{x}_j)$ is scored by its cosine $\frac{x_i \cdot \tilde{x}_j}{|x_i||\tilde{x}_j|}$. Each token in x is matched to a token in $\tilde{x}$ to compute recall, and each token in $\tilde{x}$ is matched to a token in x to compute precision (with each token greedily matched to the most similar token in the corresponding sentence). BERTSCORE provides precision and recall (and hence F_1):

$$R_{\text{BERT}} = \frac{1}{|x|} \sum_{x_i \in x} \max_{\tilde{x}_j \in \tilde{x}} x_i \cdot \tilde{x}_j \qquad P_{\text{BERT}} = \frac{1}{|\tilde{x}|} \sum_{\tilde{x}_j \in \tilde{x}} \max_{x_i \in x} x_i \cdot \tilde{x}_j \tag{10.25}$$

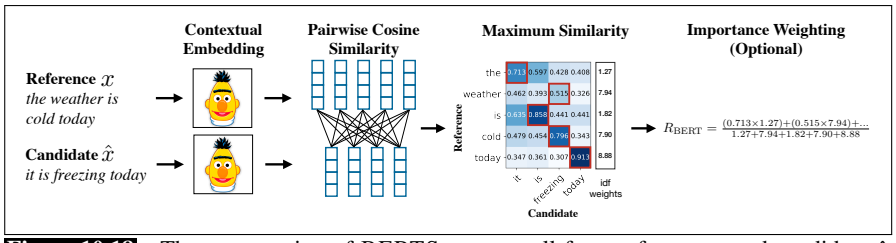


Figure 10.18 The computation of BERTSCORE recall from reference x and candidate $\hat{x}$, from Figure 1 in Zhang et al. (2020). This version shows an extended version of the metric in which tokens are also weighted by their idf values.

10.9 Bias and Ethical Issues

Machine translation raises many of the same ethical issues that we've discussed in earlier chapters. For example, consider MT systems translating from Hungarian (which has the gender neutral pronoun *ő*) or Spanish (which often drops pronouns) into English (in which pronouns are obligatory, and they have grammatical gender). When translating a reference to a person described without specified gender, MT systems often default to male gender (Schiebinger 2014, Prates et al. 2019). And MT systems often assign gender according to culture stereotypes of the sort we saw in Section **??**. Fig. 10.19 shows examples from Prates et al. (2019), in which Hungarian gender-neutral *ő is a nurse* is translated with *she*, but gender-neutral *ő is a CEO* is translated with *he*. Prates et al. (2019) find that these stereotypes can't completely be accounted for by gender bias in US labor statistics, because the biases are **amplified** by MT systems, with pronouns being mapped to male or female gender with a probability higher than if the mapping was based on actual labor employment statistics.

Hungarian (gender neutral) source	English MT output
ő egy ápoló	she is a nurse
ő egy tudós	he is a scientist
ő egy mérnök	he is an engineer
ő egy pék	he is a baker
ő egy tanár	she is a teacher
ő egy vesküvőszervező	she is a wedding organizer
ő egy vezérigazgató	he is a CEO

Figure 10.19 When translating from gender-neutral languages like Hungarian into English, current MT systems interpret people from traditionally male-dominated occupations as male, and traditionally female-dominated occupations as female (Prates et al., 2019).

Similarly, a recent challenge set, the WinoMT dataset (Stanovsky et al., 2019) shows that MT systems perform worse when they are asked to translate sentences that describe people with non-stereotypical gender roles, like "The doctor asked the nurse to help her in the operation".

Many ethical questions in MT require further research. One open problem is developing metrics for knowing what our systems don't know. This is because MT systems can be used in urgent situations where human translators may be unavailable or delayed: in medical domains, to help translate when patients and doctors don't speak the same language, or in legal domains, to help judges or lawyers communicate with witnesses or defendants. In order to 'do no harm', systems need ways to assign **confidence** values to candidate translations, so they can abstain from giving incorrect translations that may cause harm.

confidence

Another is the need for low-resource algorithms that can translate to and from all the world's languages, the vast majority of which do not have large parallel training texts available. This problem is exacerbated by the tendency of many MT approaches to focus on the case where one of the languages is English (Anastasopoulos and Neubig, 2020). ∀ et al. (2020) propose a participatory design process to encourage content creators, curators, and language technologists who speak these **low-resourced languages** to participate in developing MT algorithms. They provide online groups, mentoring, and infrastructure, and report on a case study on developing MT algorithms for low-resource African languages.

low-resourced languages

10.10 Summary

Machine translation is one of the most widely used applications of NLP, and the encoder-decoder model, first developed for MT is a key tool that has applications throughout NLP.

- Languages have **divergences**, both structural and lexical, that make translation difficult.
- The linguistic field of **typology** investigates some of these differences; languages can be classified by their position along typological dimensions like whether verbs precede their objects.
- **Encoder-decoder** networks (either for RNNs or transformers) are composed of an **encoder** network that takes an input sequence and creates a contextualized representation of it, the **context**. This context representation is then passed to a **decoder** which generates a task-specific output sequence.
- The **attention mechanism** in RNNs, and **cross-attention** in transformers, allows the decoder to view information from all the hidden states of the encoder.
- For the decoder, choosing the single most probable token to generate at each step is called **greedy** decoding.
- In **beam search**, instead of choosing the best token to generate at each timestep, we keep k possible tokens at each step. This fixed-size memory footprint k is called the **beam width**.
- Machine translation models are trained on a **parallel corpus**, sometimes called a **bitext**, a text that appears in two (or more) languages.
- **Backtranslation** is a way of making use of monolingual corpora in the target language by running a pilot MT engine backwards to create synthetic bitexts.
- MT is evaluated by measuring a translation's **adequacy** (how well it captures the meaning of the source sentence) and **fluency** (how fluent or natural it is in the target language). Human evaluation is the gold standard, but automatic evaluation metrics like **chrF**, which measure character n-gram overlap with human translations, or more recent metrics based on embedding similarity, are also commonly used.

Bibliographical and Historical Notes

MT was proposed seriously by the late 1940s, soon after the birth of the computer (Weaver, 1949/1955). In 1954, the first public demonstration of an MT system prototype (Dostert, 1955) led to great excitement in the press (Hutchins, 1997). The next decade saw a great flowering of ideas, prefiguring most subsequent developments. But this work was ahead of its time—implementations were limited by, for example, the fact that pending the development of disks there was no good way to store dictionary information.

As high-quality MT proved elusive (Bar-Hillel, 1960), there grew a consensus on the need for better evaluation and more basic research in the new fields of formal and computational linguistics. This consensus culminated in the famously critical ALPAC (Automatic Language Processing Advisory Committee) report of 1966 (Pierce et al., 1966) that led in the mid 1960s to a dramatic cut in funding for MT

in the US. As MT research lost academic respectability, the Association for Machine Translation and Computational Linguistics dropped MT from its name. Some MT developers, however, persevered, and there were early MT systems like Météo, which translated weather forecasts from English to French (Chandioux, 1976), and industrial systems like Systran.

In the early years, the space of MT architectures spanned three general models. In **direct translation**, the system proceeds word-by-word through the source-language text, translating each word incrementally. Direct translation uses a large bilingual dictionary, each of whose entries is a small program with the job of translating one word. In **transfer** approaches, we first parse the input text and then apply rules to transform the source-language parse into a target language parse. We then generate the target language sentence from the parse tree. In **interlingua** approaches, we analyze the source language text into some abstract meaning representation, called an **interlingua**. We then generate into the target language from this interlingual representation. A common way to visualize these three early approaches was the **Vauquois triangle** shown in Fig. 10.20. The triangle shows the increasing depth of analysis required (on both the analysis and generation end) as we move from the direct approach through transfer approaches to interlingual approaches. In addition, it shows the decreasing amount of transfer knowledge needed as we move up the triangle, from huge amounts of transfer at the direct level (almost all knowledge is transfer knowledge for each word) through transfer (transfer rules only for parse trees or thematic roles) through interlingua (no specific transfer knowledge). We can view the encoder-decoder network as an interlingual approach, with attention acting as an integration of direct and transfer, allowing words or their representations to be directly accessed by the decoder.

Vauquois triangle

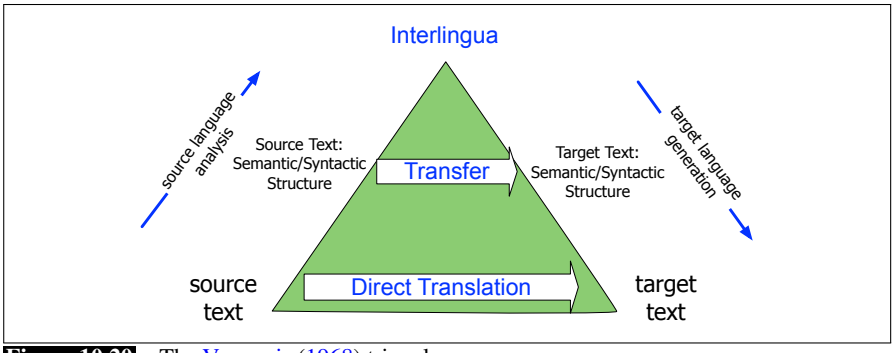


Figure 10.20 The Vauquois (1968) triangle.

Statistical methods began to be applied around 1990, enabled first by the development of large bilingual corpora like the **Hansard** corpus of the proceedings of the Canadian Parliament, which are kept in both French and English, and then by the growth of the Web. Early on, a number of researchers showed that it was possible to extract pairs of aligned sentences from bilingual corpora, using words or simple cues like sentence length (Kay and Röscheisen 1988, Gale and Church 1991, Gale and Church 1993, Kay and Röscheisen 1993).

At the same time, the IBM group, drawing directly on the **noisy channel** model for speech recognition, proposed two related paradigms for **statistical MT**. These include the generative algorithms that became known as **IBM Models 1 through 5**, implemented in the **Candide** system. The algorithms (except for the decoder) were published in full detail— encouraged by the US government who had par-

statistical MT
IBM Models
Candide

tially funded the work— which gave them a huge impact on the research community (Brown et al. 1990, Brown et al. 1993). The group also developed a discriminative approach, called MaxEnt (for maximum entropy, an alternative formulation of logistic regression), which allowed many features to be combined discriminatively rather than generatively (Berger et al., 1996), which was further developed by Och and Ney (2002).

phrase-based translation

By the turn of the century, most academic research on machine translation used statistical MT. An extended approach, called **phrase-based translation** was developed, based on inducing translations for phrase-pairs (Och 1998, Marcu and Wong 2002, Koehn et al. (2003), Och and Ney 2004, Deng and Byrne 2005, inter alia). Once automatic metrics like BLEU were developed (Papineni et al., 2002), use log linear formulation (Och and Ney, 2004) to directly optimize evaluation metrics like BLEU in a method known as **Minimum Error Rate Training**, or **MERT** (Och, 2003), also drawing from speech recognition models (Chou et al., 1993). Toolkits like GIZA (Och and Ney, 2003) and **Moses** (Koehn et al. 2006, Zens and Ney 2007) were widely used.

MERT

Moses

transduction grammars

There were also approaches around the turn of the century that were based on syntactic structure (Chapter 12). Models based on **transduction grammars** (also called **synchronous grammars** assign a parallel syntactic tree structure to a pair of sentences in different languages, with the goal of translating the sentences by applying reordering operations on the trees. From a generative perspective, we can view a transduction grammar as generating pairs of aligned sentences in two languages. Some of the most widely used models included the **inversion transduction grammar** (Wu, 1996) and synchronous context-free grammars (Chiang, 2005),

inversion transduction grammar

Neural networks had been applied at various times to various aspects of machine translation; for example Schwenk et al. (2006) showed how to use neural language models to replace n-gram language models in a Spanish-English system based on IBM Model 4. The modern neural encoder-decoder approach was pioneered by Kalchbrenner and Blunsom (2013), who used a CNN encoder and an RNN decoder. Cho et al. (2014) (who coined the name "encoder-decoder") and Sutskever et al. (2014) then showed how to use extended RNNs for both encoder and decoder. The idea that a generative decoder should take as input a soft weighting of the inputs, the central idea of attention, was first developed by Graves (2013) in the context of handwriting recognition. Bahdanau et al. (2015) extended the idea, named it "attention" and applied it to MT. The transformer encoder-decoder was proposed by Vaswani et al. (2017) (see the History section of Chapter 9).

Beam-search has an interesting relationship with human language processing; (Meister et al., 2020) show that beam search enforces the cognitive property of **uniform information density** in text. Uniform information density is the hypothesis that human language processors tend to prefer to distribute information equally across the sentence (Jaeger and Levy, 2007).

Research on evaluation of machine translation began quite early. Miller and Beebe-Center (1956) proposed a number of methods drawing on work in psycholinguistics. These included the use of cloze and Shannon tasks to measure intelligibility as well as a metric of edit distance from a human translation, the intuition that underlies all modern overlap-based automatic evaluation metrics. The ALPAC report included an early evaluation study conducted by John Carroll that was extremely influential (Pierce et al., 1966, Appendix 10). Carroll proposed distinct measures for fidelity and intelligibility, and had raters score them subjectively on 9-point scales. Much early evaluation work focuses on automatic word-overlap metrics like BLEU

(Papineni et al., 2002), **NIST** (Doddington, 2002), **TER (Translation Error Rate)** (Snover et al., 2006), **Precision and Recall** (Turian et al., 2003), and **METEOR** (Banerjee and Lavie, 2005); character n-gram overlap methods like chrF (Popović, 2015) came later. More recent evaluation work, echoing the ALPAC report, has emphasized the importance of careful statistical methodology and the use of human evaluation (Kocmi et al., 2021; Marie et al., 2021).

The early history of MT is surveyed in Hutchins 1986 and 1997; Nirenburg et al. (2002) collects early readings. See Croft (1990) or Comrie (1989) for introductions to linguistic typology.

Exercises

10.1 Compute by hand the chrF2,2 score for HYP2 on page 22 (the answer should round to .62).

Anastasopoulos, A. and G. Neubig. 2020. Should all cross-lingual embeddings speak English? *ACL*.

Artetxe, M. and H. Schwenk. 2019. Massively multilingual sentence embeddings for zero-shot cross-lingual transfer and beyond. *TACL*, 7:597–610.

Bahdanau, D., K. H. Cho, and Y. Bengio. 2015. Neural machine translation by jointly learning to align and translate. *ICLR 2015*.

Banerjee, S. and A. Lavie. 2005. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. *Proceedings of ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for MT and/or Summarization*.

Bañón, M., P. Chen, B. Haddow, K. Heafield, H. Hoang, M. Esplà-Gomis, M. L. Forcada, A. Kamran, F. Kirefu, P. Koehn, S. Ortiz Rojas, L. Pla Sempere, G. Ramírez-Sánchez, E. Sarrías, M. Strelec, B. Thompson, W. Waites, D. Wiggins, and J. Zaragoza. 2020. ParaCrawl: Web-scale acquisition of parallel corpora. *ACL*.

Bar-Hillel, Y. 1960. The present status of automatic translation of languages. In F. Alt, editor, *Advances in Computers 1*, pages 91–163. Academic Press.

Berger, A., S. A. Della Pietra, and V. J. Della Pietra. 1996. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71.

Bickel, B. 2003. Referential density in discourse and syntactic typology. *Language*, 79(2):708–736.

Brown, P. F., J. Cocke, S. A. Della Pietra, V. J. Della Pietra, F. Jelinek, J. D. Lafferty, R. L. Mercer, and P. S. Roossin. 1990. A statistical approach to machine translation. *Computational Linguistics*, 16(2):79–85.

Brown, P. F., S. A. Della Pietra, V. J. Della Pietra, and R. L. Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311.

Callison-Burch, C., M. Osborne, and P. Koehn. 2006. Re-evaluating the role of BLEU in machine translation research. *EACL*.

Chandioux, J. 1976. MÉTÉO: un système opérationnel pour la traduction automatique des bulletins météorologiques destinés au grand public. *Meta*, 21:127–133.

Chiang, D. 2005. A hierarchical phrase-based model for statistical machine translation. *ACL*.

Cho, K., B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. *EMNLP*.

Chou, W., C.-H. Lee, and B. H. Juang. 1993. Minimum error rate training based on *n*-best string models. *ICASSP*.

Comrie, B. 1989. *Language Universals and Linguistic Typology*, 2nd edition. Blackwell.

Croft, W. 1990. *Typology and Universals*. Cambridge University Press.

Deng, Y. and W. Byrne. 2005. HMM word and phrase alignment for statistical machine translation. *HLT-EMNLP*.

Doddington, G. 2002. Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. *HLT*.

Dostert, L. 1955. The Georgetown-I.B.M. experiment. In *Machine Translation of Languages: Fourteen Essays*, pages 124–135. MIT Press.

Dryer, M. S. and M. Haspelmath, editors. 2013. *The World Atlas of Language Structures Online*. Max Planck Institute for Evolutionary Anthropology, Leipzig. Available online at `http://wals.info`.

Edunov, S., M. Ott, M. Auli, and D. Grangier. 2018. Understanding back-translation at scale. *EMNLP*.

∀, W. Nekoto, V. Marivate, T. Matsila, T. Fasubaa, T. Kolawole, T. Fagbohungbe, S. O. Akinola, S. H. Muhammad, S. Kabongo, S. Osei, S. Freshia, R. A. Niyongabo, R. M. P. Ogayo, O. Ahia, M. Meressa, M. Adeyemi, M. Mokgesi-Selinga, L. Okegbemi, L. J. Martinus, K. Tajudeen, K. Degila, K. Ogueji, K. Siminyu, J. Kreutzer, J. Webster, J. T. Ali, J. A. I. Orife, I. Ezeani, I. A. Dangana, H. Kamper, H. Elsahar, G. Duru, G. Kioko, E. Murhabazi, E. van Biljon, D. Whitenack, C. Onyefuluchi, C. Emezue, B. Dossou, B. Sibanda, B. I. Bassey, A. Olabiyi, A. Ramkilowan, A. Öktem, A. Akinfaderin, and A. Bashir. 2020. Participatory research for low-resourced machine translation: A case study in African languages. *Findings of EMNLP*.

Gale, W. A. and K. W. Church. 1991. A program for aligning sentences in bilingual corpora. *ACL*.

Gale, W. A. and K. W. Church. 1993. A program for aligning sentences in bilingual corpora. *Computational Linguistics*, 19:75–102.

Graves, A. 2013. Generating sequences with recurrent neural networks. ArXiv.

Hutchins, W. J. 1986. *Machine Translation: Past, Present, Future*. Ellis Horwood, Chichester, England.

Hutchins, W. J. 1997. From first conception to first demonstration: The nascent years of machine translation, 1947–1954. A chronology. *Machine Translation*, 12:192–252.

Hutchins, W. J. and H. L. Somers. 1992. *An Introduction to Machine Translation*. Academic Press.

Jaeger, T. F. and R. P. Levy. 2007. Speakers optimize information density through syntactic reduction. *NeurIPS*.

Kalchbrenner, N. and P. Blunsom. 2013. Recurrent continuous translation models. *EMNLP*.

Kay, M. and M. Röscheisen. 1988. Text-translation alignment. Technical Report P90-00143, Xerox Palo Alto Research Center, Palo Alto, CA.

Kay, M. and M. Röscheisen. 1993. Text-translation alignment. *Computational Linguistics*, 19:121–142.

Kocmi, T., C. Federmann, R. Grundkiewicz, M. Junczys-Dowmunt, H. Matsushita, and A. Menezes. 2021. To ship or not to ship: An extensive evaluation of automatic metrics for machine translation. ArXiv.

Koehn, P. 2005. Europarl: A parallel corpus for statistical machine translation. *MT summit, vol. 5*.

Koehn, P., H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, C. Dyer, O. Bojar, A. Constantin, and E. Herbst. 2006. Moses: Open source toolkit for statistical machine translation. *ACL*.

Koehn, P., F. J. Och, and D. Marcu. 2003. Statistical phrase-based translation. *HLT-NAACL*.

Lison, P. and J. Tiedemann. 2016. Opensubtitles2016: Extracting large parallel corpora from movie and tv subtitles. *LREC*.

Marcu, D. and W. Wong. 2002. A phrase-based, joint probability model for statistical machine translation. *EMNLP*.

Marie, B., A. Fujita, and R. Rubino. 2021. Scientific credibility of machine translation research: A meta-evaluation of 769 papers. *ACL 2021*.

McLuhan, M. 1964. *Understanding Media: The Extensions of Man*. New American Library.

Meister, C., T. Vieira, and R. Cotterell. 2020. If beam search is the answer, what was the question? *EMNLP*.

Miller, G. A. and J. G. Beebe-Center. 1956. Some psychological methods for evaluating the quality of translations. *Mechanical Translation*, 3:73–80.

Nirenburg, S., H. L. Somers, and Y. Wilks, editors. 2002. *Readings in Machine Translation*. MIT Press.

Och, F. J. 1998. *Ein beispielsbasierter und statistischer Ansatz zum maschinellen Lernen von natürlichsprachlicher Übersetzung*. Ph.D. thesis, Universität Erlangen-Nürnberg, Germany. Diplomarbeit (diploma thesis).

Och, F. J. 2003. Minimum error rate training in statistical machine translation. *ACL*.

Och, F. J. and H. Ney. 2002. Discriminative training and maximum entropy models for statistical machine translation. *ACL*.

Och, F. J. and H. Ney. 2003. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51.

Och, F. J. and H. Ney. 2004. The alignment template approach to statistical machine translation. *Computational Linguistics*, 30(4):417–449.

Papineni, K., S. Roukos, T. Ward, and W.-J. Zhu. 2002. Bleu: A method for automatic evaluation of machine translation. *ACL*.

Pierce, J. R., J. B. Carroll, E. P. Hamp, D. G. Hays, C. F. Hockett, A. G. Oettinger, and A. J. Perlis. 1966. *Language and Machines: Computers in Translation and Linguistics*. ALPAC report. National Academy of Sciences, National Research Council, Washington, DC.

Popović, M. 2015. chrF: character n-gram F-score for automatic MT evaluation. *Proceedings of the Tenth Workshop on Statistical Machine Translation*.

Prates, M. O. R., P. H. Avelar, and L. C. Lamb. 2019. Assessing gender bias in machine translation: a case study with Google Translate. *Neural Computing and Applications*, 32:6363–6381.

Rei, R., C. Stewart, A. C. Farinha, and A. Lavie. 2020. COMET: A neural framework for MT evaluation. *EMNLP*.

Schiebinger, L. 2014. Scientific research must take gender into account. *Nature*, 507(7490):9.

Schwenk, H. 2018. Filtering and mining parallel data in a joint multilingual space. *ACL*.

Schwenk, H., D. Dechelotte, and J.-L. Gauvain. 2006. Continuous space language models for statistical machine translation. *COLING/ACL*.

Sellam, T., D. Das, and A. Parikh. 2020. BLEURT: Learning robust metrics for text generation. *ACL*.

Slobin, D. I. 1996. Two ways to travel. In M. Shibatani and S. A. Thompson, editors, *Grammatical Constructions: Their Form and Meaning*, pages 195–220. Clarendon Press.

Snover, M., B. Dorr, R. Schwartz, L. Micciulla, and J. Makhoul. 2006. A study of translation edit rate with targeted human annotation. *AMTA-2006*.

Stanovsky, G., N. A. Smith, and L. Zettlemoyer. 2019. Evaluating gender bias in machine translation. *ACL*.

Sutskever, I., O. Vinyals, and Q. V. Le. 2014. Sequence to sequence learning with neural networks. *NeurIPS*.

Talmy, L. 1985. Lexicalization patterns: Semantic structure in lexical forms. In T. Shopen, editor, *Language Typology and Syntactic Description, Volume 3*. Cambridge University Press. Originally appeared as UC Berkeley Cognitive Science Program Report No. 30, 1980.

Talmy, L. 1991. Path to realization: A typology of event conflation. *BLS-91*.

Thompson, B. and P. Koehn. 2019. Vecalign: Improved sentence alignment in linear time and space. *EMNLP*.

Turian, J. P., L. Shen, and I. D. Melamed. 2003. Evaluation of machine translation and its evaluation. *Proceedings of MT Summit IX*.

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. 2017. Attention is all you need. *NeurIPS*.

Vauquois, B. 1968. A survey of formal grammars and algorithms for recognition and transformation in machine translation. *IFIP Congress 1968*.

Weaver, W. 1949/1955. Translation. In W. N. Locke and A. D. Boothe, editors, *Machine Translation of Languages*, pages 15–23. MIT Press. Reprinted from a memorandum written by Weaver in 1949.

Wu, D. 1996. A polynomial-time algorithm for statistical machine translation. *ACL*.

Wu, Y., M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Ł. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. S. Corrado, M. Hughes, and J. Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. ArXiv preprint arXiv:1609.08144.

Zens, R. and H. Ney. 2007. Efficient phrase-table representation for machine translation with applications to online MT and speech translation. *NAACL-HLT*.

Zhang, T., V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi. 2020. Bertscore: Evaluating text generation with BERT. *ICLR 2020*.

Ziemski, M., M. Junczys-Dowmunt, and B. Pouliquen. 2016. The United Nations parallel corpus v1.0. *LREC*.