Chapter 18

Machine translation

Machine translation (MT) is one of the "holy grail" problems in artificial intelligence, with the potential to transform society by facilitating communication between people anywhere in the world. As a result, MT has received significant attention and funding since the early 1950s. However, it has proved remarkably challenging, and while there has been substantial progress towards usable MT systems — especially for high-resource language pairs like English-French — we are still far from translation systems that match the nuance and depth of human translations.

18.1 Machine translation as a task

Machine translation can be formulated as an optimization problem:

$$\hat{\boldsymbol{w}}^{(t)} = \underset{\boldsymbol{w}^{(t)}}{\text{argmax}}\, \Psi(\boldsymbol{w}^{(s)}, \boldsymbol{w}^{(t)}), \tag{18.1}$$

where $\boldsymbol{w}^{(s)}$ is a sentence in a **source** language, $\boldsymbol{w}^{(t)}$ is a sentence in the **target language**, and Ψ is a scoring function. As usual, this formalism requires two components: a decoding algorithm for computing $\hat{\boldsymbol{w}}^{(t)}$, and a learning algorithm for estimating the parameters of the scoring function Ψ.

Decoding is difficult for machine translation because of the huge space of possible translations. We have faced large label spaces before: for example, in sequence labeling, the set of possible label sequences is exponential in the length of the input. In these cases, it was possible to search the space quickly by introducing locality assumptions: for example, that each tag depends only on its predecessor, or that each production depends only on its parent. In machine translation, no such locality assumptions seem possible: human translators reword, reorder, and rearrange words; they replace single words with multi-word phrases, and vice versa. This flexibility means that in even relatively simple

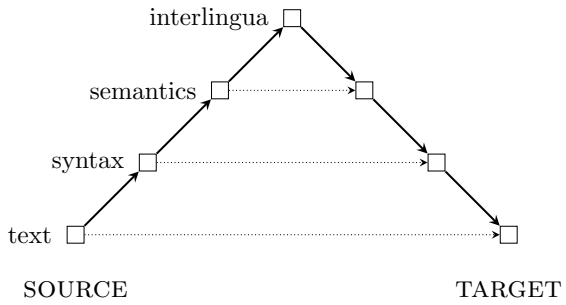

Figure 18.1: The Vauquois Pyramid

translation models, decoding is NP-hard (Knight, 1999). Approaches for dealing with this complexity are described in § 18.4.

Estimating translation models is difficult as well. Labeled translation data usually comes in the form parallel sentences, e.g.,

$$\boldsymbol{w}^{(s)} = \textit{A Vinay le gusta las manzanas.}$$
$$\boldsymbol{w}^{(t)} = \textit{Vinay likes apples.}$$

A useful feature function would note the translation pairs (*gusta, likes*), (*manzanas, apples*), and even (*Vinay, Vinay*). But this word-to-word **alignment** is not given in the data. One solution is to treat this alignment as a **latent variable**; this is the approach taken by classical **statistical machine translation** (SMT) systems, described in § 18.2. Another solution is to model the relationship between $\boldsymbol{w}^{(t)}$ and $\boldsymbol{w}^{(s)}$ through a more complex and expressive function; this is the approach taken by **neural machine translation** (NMT) systems, described in § 18.3.

The **Vauquois Pyramid** is a theory of how translation should be done. At the lowest level, the translation system operates on individual words, but the horizontal distance at this level is large, because languages express ideas differently. If we can move up the triangle to syntactic structure, the distance for translation is reduced; we then need only produce target-language text from the syntactic representation, which can be as simple as reading off a tree. Further up the triangle lies semantics; translating between semantic representations should be easier still, but mapping between semantics and surface text is a difficult, unsolved problem. At the top of the triangle is **interlingua**, a semantic representation that is so generic that it is identical across all human languages. Philosophers debate whether such a thing as interlingua is really possible (e.g., Derrida, 1985). While the first-order logic representations discussed in chapter 12 might be thought to be language independent, they are built on an inventory of predicates that are suspiciously similar to English words (Nirenburg and Wilks, 2001). Nonetheless, the idea of linking translation

	Adequate?	Fluent?
To Vinay it like Python	yes	no
Vinay debugs memory leaks	no	yes
Vinay likes Python	yes	yes

Table 18.1: Adequacy and fluency for translations of the Spanish sentence *A Vinay le gusta Python*.

and semantic understanding may still be a promising path, if the resulting translations better preserve the meaning of the original text.

18.1.1 Evaluating translations

There are two main criteria for a translation, summarized in Table 18.1.

- **Adequacy**: The translation $\boldsymbol{w}^{(t)}$ should adequately reflect the linguistic content of $\boldsymbol{w}^{(s)}$. For example, if $\boldsymbol{w}^{(s)}$ = *A Vinay le gusta Python*, the reference translation is $\boldsymbol{w}^{(t)}$ = *Vinay likes Python*. However, the **gloss**, or word-for-word translation $\boldsymbol{w}^{(t)}$ = *To Vinay it like Python* is also considered adequate because it contains all the relevant content. The output $\boldsymbol{w}^{(t)}$ = *Vinay debugs memory leaks* is not adequate.
- **Fluency**: The translation $\boldsymbol{w}^{(t)}$ should read like fluent text in the target language. By this criterion, the gloss $\boldsymbol{w}^{(t)}$ = *To Vinay it like Python* will score poorly, and $\boldsymbol{w}^{(t)}$ = *Vinay debugs memory leaks* will be preferred.

Automated evaluations of machine translations typically merge both of these criteria, by comparing the system translation with one or more **reference translations**, produced by professional human translators. The most popular quantitative metric is **BLEU** (bilingual evaluation understudy; Papineni et al., 2002), which is based on n-gram precision: what fraction of n-grams in the system translation appear in the reference? Specifically, for each n-gram length, the precision is defined as,

$$p_n = \frac{\text{number of } n\text{-grams appearing in both reference and hypothesis translations}}{\text{number of } n\text{-grams appearing in the hypothesis translation}}. \quad [18.2]$$

The n-gram precisions for three hypothesis translations are shown in Figure 18.2.

The BLEU score is then based on the average, $\exp \frac{1}{N} \sum_{n=1}^{N} \log p_n$. Two modifications of Equation 18.2 are necessary: (1) to avoid computing $\log 0$, all precisions are smoothed to ensure that they are positive; (2) each n-gram in the reference can be used at most once, so that *to to to to to to* does not achieve $p_1 = 1$ against the reference *to be or not to be*. Furthermore, precision-based metrics are biased in favor of short translations, which

Under contract with MIT Press, shared under CC-BY-NC-ND license.

	Translation	p_1	p_2	p_3	p_4	BP	BLEU
Reference	*Vinay likes programming in Python*						
Sys1	*To Vinay it like to program Python*	$\frac{2}{7}$	0	0	0	1	.21
Sys2	*Vinay likes Python*	$\frac{3}{3}$	$\frac{1}{2}$	0	0	.51	.33
Sys3	*Vinay likes programming in his pajamas*	$\frac{4}{6}$	$\frac{3}{5}$	$\frac{2}{4}$	$\frac{1}{3}$	1	.76

Figure 18.2: A reference translation and three system outputs. For each output, p_n indicates the precision at each n-gram, and BP indicates the brevity penalty.

can achieve high scores by minimizing the denominator in [18.2]. To avoid this issue, a **brevity penalty** is applied to translations that are shorter than the reference. This penalty is indicated as "BP" in Figure 18.2.

Automated metrics like BLEU have been validated by correlation with human judgments of translation quality. Nonetheless, it is not difficult to construct examples in which the BLEU score is high, yet the translation is disfluent or carries a completely different meaning from the original. To give just one example, consider the problem of translating pronouns. Because pronouns refer to specific entities, a single incorrect pronoun can obliterate the semantics of the original sentence. Existing state-of-the-art systems generally do not attempt the reasoning necessary to correctly resolve pronominal anaphora (Hardmeier, 2012). Despite the importance of pronouns for semantics, they have a marginal impact on BLEU, which may help to explain why existing systems do not make a greater effort to translate them correctly.

Fairness and bias The problem of pronoun translation intersects with issues of fairness and bias. In many languages, such as Turkish, the third person singular pronoun is gender neutral. Today's state-of-the-art systems produce the following Turkish-English translations (Caliskan et al., 2017):

(18.1) *O bir doktor.*
He is a doctor.

(18.2) *O bir hemşire.*
She is a nurse.

The same problem arises for other professions that have stereotypical genders, such as engineers, soldiers, and teachers, and for other languages that have gender-neutral pronouns. This bias was not directly programmed into the translation model; it arises from statistical tendencies in existing datasets. This highlights a general problem with data-driven approaches, which can perpetuate biases that negatively impact disadvantaged

groups. Worse, machine learning can *amplify* biases in data (Bolukbasi et al., 2016): if a dataset has even a slight tendency towards men as doctors, the resulting translation model may produce translations in which doctors are always *he*, and nurses are always *she*.

Other metrics A range of other automated metrics have been proposed for machine translation. One potential weakness of BLEU is that it only measures precision; **METEOR** is a weighted F-MEASURE, which is a combination of recall and precision (see § 4.4.1). **Translation Error Rate (TER)** computes the string **edit distance** (see § 9.1.4) between the reference and the hypothesis (Snover et al., 2006). For language pairs like English and Japanese, there are substantial differences in word order, and word order errors are not sufficiently captured by n-gram based metrics. The **RIBES** metric applies rank correlation to measure the similarity in word order between the system and reference translations (Isozaki et al., 2010).

18.1.2 Data

Data-driven approaches to machine translation rely primarily on **parallel corpora**, which are translations at the sentence level. Early work focused on government records, in which fine-grained official translations are often required. For example, the IBM translation systems were based on the proceedings of the Canadian Parliament, called **Hansards**, which are recorded in English and French (Brown et al., 1990). The growth of the European Union led to the development of the **EuroParl corpus**, which spans 21 European languages (Koehn, 2005). While these datasets helped to launch the field of machine translation, they are restricted to narrow domains and a formal speaking style, limiting their applicability to other types of text. As more resources are committed to machine translation, new translation datasets have been commissioned. This has broadened the scope of available data to news,[1] movie subtitles,[2] social media (Ling et al., 2013), dialogues (Fordyce, 2007), TED talks (Paul et al., 2010), and scientific research articles (Nakazawa et al., 2016).

Despite this growing set of resources, the main bottleneck in machine translation data is the need for parallel corpora that are aligned at the sentence level. Many languages have sizable parallel corpora with some high-resource language, but not with each other. The high-resource language can then be used as a "pivot" or "bridge" (Boitet, 1988; Utiyama and Isahara, 2007): for example, De Gispert and Marino (2006) use Spanish as a bridge for translation between Catalan and English. For most of the 6000 languages spoken today, the only source of translation data remains the Judeo-Christian Bible (Resnik et al., 1999). While relatively small, at less than a million tokens, the Bible has been translated into more than 2000 languages, far outpacing any other corpus. Some research has explored

[1] https://catalog.ldc.upenn.edu/LDC2010T10, http://www.statmt.org/wmt15/translation-task.html

[2] http://opus.nlpl.eu/

Under contract with MIT Press, shared under CC-BY-NC-ND license.

the possibility of automatically identifying parallel sentence pairs from unaligned parallel texts, such as web pages and Wikipedia articles (Kilgarriff and Grefenstette, 2003; Resnik and Smith, 2003; Adafre and De Rijke, 2006). Another approach is to create large parallel corpora through crowdsourcing (Zaidan and Callison-Burch, 2011).

18.2 Statistical machine translation

The previous section introduced adequacy and fluency as the two main criteria for machine translation. A natural modeling approach is to represent them with separate scores,

$$\Psi(\boldsymbol{w}^{(s)}, \boldsymbol{w}^{(t)}) = \Psi_A(\boldsymbol{w}^{(s)}, \boldsymbol{w}^{(t)}) + \Psi_F(\boldsymbol{w}^{(t)}). \quad [18.3]$$

The fluency score Ψ_F need not even consider the source sentence; it only judges $\boldsymbol{w}^{(t)}$ on whether it is fluent in the target language. This decomposition is advantageous because it makes it possible to estimate the two scoring functions on separate data. While the adequacy model must be estimated from aligned sentences — which are relatively expensive and rare — the fluency model can be estimated from monolingual text in the target language. Large monolingual corpora are now available in many languages, thanks to resources such as Wikipedia.

An elegant justification of the decomposition in Equation 18.3 is provided by the **noisy channel model**, in which each scoring function is a log probability:

$$\Psi_A(\boldsymbol{w}^{(s)}, \boldsymbol{w}^{(t)}) \triangleq \log \mathrm{p}_{S|T}(\boldsymbol{w}^{(s)} \mid \boldsymbol{w}^{(t)}) \quad [18.4]$$

$$\Psi_F(\boldsymbol{w}^{(t)}) \triangleq \log \mathrm{p}_T(\boldsymbol{w}^{(t)}) \quad [18.5]$$

$$\Psi(\boldsymbol{w}^{(s)}, \boldsymbol{w}^{(t)}) = \log \mathrm{p}_{S|T}(\boldsymbol{w}^{(s)} \mid \boldsymbol{w}^{(t)}) + \log \mathrm{p}_T(\boldsymbol{w}^{(t)}) = \log \mathrm{p}_{S,T}(\boldsymbol{w}^{(s)}, \boldsymbol{w}^{(t)}). \quad [18.6]$$

By setting the scoring functions equal to the logarithms of the prior and likelihood, their sum is equal to $\log \mathrm{p}_{S,T}$, which is the logarithm of the joint probability of the source and target. The sentence $\hat{\boldsymbol{w}}^{(t)}$ that maximizes this joint probability is also the maximizer of the conditional probability $\mathrm{p}_{T|S}$, making it the most likely target language sentence, conditioned on the source.

The noisy channel model can be justified by a generative story. The target text is originally generated from a probability model p_T. It is then encoded in a "noisy channel" $\mathrm{p}_{S|T}$, which converts it to a string in the source language. In decoding, we apply Bayes' rule to recover the string $\boldsymbol{w}^{(t)}$ that is maximally likely under the conditional probability $\mathrm{p}_{T|S}$. Under this interpretation, the target probability p_T is just a language model, and can be estimated using any of the techniques from chapter 6. The only remaining learning problem is to estimate the translation model $\mathrm{p}_{S|T}$.

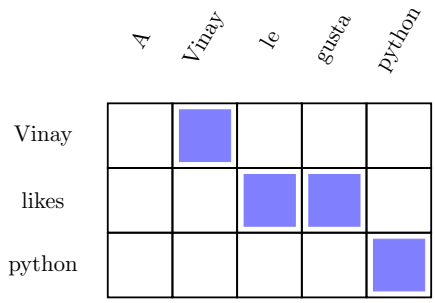

Figure 18.3: An example word-to-word alignment

18.2.1 Statistical translation modeling

The simplest decomposition of the translation model is word-to-word: each word in the source should be aligned to a word in the translation. This approach presupposes an **alignment** $\mathcal{A}(\boldsymbol{w}^{(s)}, \boldsymbol{w}^{(t)})$, which contains a list of pairs of source and target tokens. For example, given $\boldsymbol{w}^{(s)} =$ *A Vinay le gusta Python* and $\boldsymbol{w}^{(t)} =$ *Vinay likes Python*, one possible word-to-word alignment is,

$$\mathcal{A}(\boldsymbol{w}^{(s)}, \boldsymbol{w}^{(t)}) = \{(A, \varnothing), (\textit{Vinay, Vinay}), (\textit{le, likes}), (\textit{gusta, likes}), (\textit{Python,Python})\}. \quad [18.7]$$

This alignment is shown in Figure 18.3. Another, less promising, alignment is:

$$\mathcal{A}(\boldsymbol{w}^{(s)}, \boldsymbol{w}^{(t)}) = \{(\textit{A, Vinay}), (\textit{Vinay, likes}), (\textit{le, Python}), (\textit{gusta}, \varnothing), (\textit{Python}, \varnothing)\}. \quad [18.8]$$

Each alignment contains exactly one tuple for each word in the *source*, which serves to explain how the source word could be translated from the target, as required by the translation probability $\mathrm{p}_{S|T}$. If no appropriate word in the target can be identified for a source word, it is aligned to $\varnothing$ — as is the case for the Spanish function word *a* in the example, which glosses to the English word *to*. Words in the target can align with multiple words in the source, so that the target word *likes* can align to both *le* and *gusta* in the source.

The joint probability of the alignment and the translation can be defined conveniently as,

$$\mathrm{p}(\boldsymbol{w}^{(s)}, \mathcal{A} \mid \boldsymbol{w}^{(t)}) = \prod_{m=1}^{M^{(s)}} \mathrm{p}(w_m^{(s)}, a_m \mid w_{a_m}^{(t)}, m, M^{(s)}, M^{(t)}) \quad [18.9]$$

$$= \prod_{m=1}^{M^{(s)}} \mathrm{p}(a_m \mid m, M^{(s)}, M^{(t)}) \times \mathrm{p}(w_m^{(s)} \mid w_{a_m}^{(t)}). \quad [18.10]$$

This probability model makes two key assumptions:

Under contract with MIT Press, shared under CC-BY-NC-ND license.

- The alignment probability factors across tokens,

$$\text{p}(\mathcal{A} \mid \boldsymbol{w}^{(s)}, \boldsymbol{w}^{(t)}) = \prod_{m=1}^{M^{(s)}} \text{p}(a_m \mid m, M^{(s)}, M^{(t)}). \quad [18.11]$$

  This means that each alignment decision is independent of the others, and depends only on the index m, and the sentence lengths $M^{(s)}$ and $M^{(t)}$.

- The translation probability also factors across tokens,

$$\text{p}(\boldsymbol{w}^{(s)} \mid \boldsymbol{w}^{(t)}, \mathcal{A}) = \prod_{m=1}^{M^{(s)}} \text{p}(w_m^{(s)} \mid w_{a_m}^{(t)}), \quad [18.12]$$

  so that each word in $\boldsymbol{w}^{(s)}$ depends only on its aligned word in $\boldsymbol{w}^{(t)}$. This means that translation is word-to-word, ignoring context. The hope is that the target language model $\text{p}(\boldsymbol{w}^{(t)})$ will correct any disfluencies that arise from word-to-word translation.

To translate with such a model, we could sum or max over all possible alignments,

$$\begin{aligned}
\text{p}(\boldsymbol{w}^{(s)}, \boldsymbol{w}^{(t)}) =& \sum_{\mathcal{A}} \text{p}(\boldsymbol{w}^{(s)}, \boldsymbol{w}^{(t)}, \mathcal{A}) & [18.13] \\
=& \text{p}(\boldsymbol{w}^{(t)}) \sum_{\mathcal{A}} \text{p}(\mathcal{A}) \times \text{p}(\boldsymbol{w}^{(s)} \mid \boldsymbol{w}^{(t)}, \mathcal{A}) & [18.14] \\
\geq& \text{p}(\boldsymbol{w}^{(t)}) \max_{\mathcal{A}} \text{p}(\mathcal{A}) \times \text{p}(\boldsymbol{w}^{(s)} \mid \boldsymbol{w}^{(t)}, \mathcal{A}). & [18.15]
\end{aligned}$$

The term $\text{p}(\mathcal{A})$ defines the prior probability over alignments. A series of alignment models with increasingly relaxed independence assumptions was developed by researchers at IBM in the 1980s and 1990s, known as IBM Models 1-6 (Och and Ney, 2003). IBM Model 1 makes the strongest independence assumption:

$$\text{p}(a_m \mid m, M^{(s)}, M^{(t)}) = \frac{1}{M^{(t)}}. \quad [18.16]$$

In this model, every alignment is equally likely. This is almost surely wrong, but it results in a convex learning objective, yielding a good initialization for the more complex alignment models (Brown et al., 1993; Koehn, 2009).

18.2.2 Estimation

Let us define the parameter $\theta_{u \to v}$ as the probability of translating target word u to source word v. If word-to-word alignments were annotated, these probabilities could be computed from relative frequencies,

$$\hat{\theta}_{u \to v} = \frac{\text{count}(u, v)}{\text{count}(u)}, \quad [18.17]$$

where count(u, v) is the count of instances in which word v was aligned to word u in the training set, and count(u) is the total count of the target word u. The smoothing techniques mentioned in chapter 6 can help to reduce the variance of these probability estimates.

Conversely, if we had an accurate translation model, we could estimate the likelihood of each alignment decision,

$$q_m(a_m \mid \boldsymbol{w}^{(s)}, \boldsymbol{w}^{(t)}) \propto \text{p}(a_m \mid m, M^{(s)}, M^{(t)}) \times \text{p}(w_m^{(s)} \mid w_{a_m}^{(t)}), \quad [18.18]$$

where $q_m(a_m \mid \boldsymbol{w}^{(s)}, \boldsymbol{w}^{(t)})$ is a measure of our confidence in aligning source word $w_m^{(s)}$ to target word $w_{a_m}^{(t)}$. The relative frequencies could then be computed from the *expected counts*,

$$\hat{\theta}_{u \to v} = \frac{E_q\left[\text{count}(u, v)\right]}{\text{count}(u)} \quad [18.19]$$

$$E_q\left[\text{count}(u, v)\right] = \sum_m q_m(a_m \mid \boldsymbol{w}^{(s)}, \boldsymbol{w}^{(t)}) \times \delta(w_m^{(s)} = v) \times \delta(w_{a_m}^{(t)} = u). \quad [18.20]$$

The **expectation-maximization** (EM) algorithm proceeds by iteratively updating q_m and $\hat{\Theta}$. The algorithm is described in general form in chapter 5. For statistical machine translation, the steps of the algorithm are:

1. **E-step**: Update beliefs about word alignment using Equation 18.18.
2. **M-step**: Update the translation model using Equations 18.19 and 18.20.

As discussed in chapter 5, the expectation maximization algorithm is guaranteed to converge, but not to a global optimum. However, for IBM Model 1, it can be shown that EM optimizes a convex objective, and global optimality is guaranteed. For this reason, IBM Model 1 is often used as an initialization for more complex alignment models. For more detail, see Koehn (2009).

18.2.3 Phrase-based translation

Real translations are not word-to-word substitutions. One reason is that many multiword expressions are not translated literally, as shown in this example from French:

(18.3) *Nous allons prendre un verre*
We will take a glass
We'll have a drink

Under contract with MIT Press, shared under CC-BY-NC-ND license.

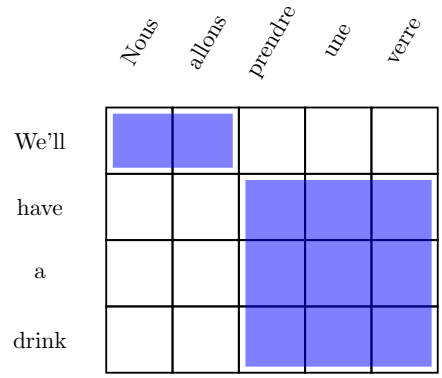

Figure 18.4: A phrase-based alignment between French and English, corresponding to example (18.3)

The line *we will take a glass* is the word-for-word gloss of the French sentence; the translation *we'll have a drink* is shown on the third line. Such examples are difficult for word-to-word translation models, since they require translating *prendre* to *have* and *verre* to *drink*. These translations are only correct in the context of these specific phrases.

Phrase-based translation generalizes on word-based models by building translation tables and alignments between multiword spans. (These "phrases" are not necessarily syntactic constituents like the noun phrases and verb phrases described in chapters 9 and 10.) The generalization from word-based translation is surprisingly straightforward: the translation tables can now condition on multi-word units, and can assign probabilities to multi-word units; alignments are mappings from spans to spans, $((i,j),(k,\ell))$, so that

$$\text{p}(\boldsymbol{w}^{(s)} \mid \boldsymbol{w}^{(t)}, \mathcal{A}) = \prod_{((i,j),(k,\ell))\in\mathcal{A}} \text{p}_{w^{(s)}|w^{(t)}}(\{w_{i+1}^{(s)}, w_{i+2}^{(s)}, \ldots, w_{j}^{(s)}\} \mid \{w_{k+1}^{(t)}, w_{k+2}^{(t)}, \ldots, w_{\ell}^{(t)}\}). \quad [18.21]$$

The phrase alignment $((i,j),(k,\ell))$ indicates that the span $\boldsymbol{w}_{i+1:j}^{(s)}$ is the translation of the span $\boldsymbol{w}_{k+1:\ell}^{(t)}$. An example phrasal alignment is shown in Figure 18.4. Note that the alignment set $\mathcal{A}$ is required to cover all of the tokens in the source, just as in word-based translation. The probability model $\text{p}_{w^{(s)}|w^{(t)}}$ must now include translations for all phrase pairs, which can be learned from expectation-maximization just as in word-based statistical machine translation.

18.2.4 *Syntax-based translation

The Vauquois Pyramid (Figure 18.1) suggests that translation might be easier if we take a higher-level view. One possibility is to incorporate the syntactic structure of the source, the target, or both. This is particularly promising for language pairs that consistent syntactic differences. For example, English adjectives almost always precede the nouns that they modify, while in Romance languages such as French and Spanish, the adjective often follows the noun: thus, *angry fish* would translate to *pez (fish) enojado (angry)* in Spanish. In word-to-word translation, these reorderings cause the alignment model to be overly permissive. It is not that the order of *any* pair of English words can be reversed when translating into Spanish, but only adjectives and nouns within a noun phrase. Similar issues arise when translating between verb-final languages such as Japanese (in which verbs usually follow the subject and object), verb-initial languages like Tagalog and classical Arabic, and verb-medial languages such as English.

An elegant solution is to link parsing and translation in a **synchronous context-free grammar** (SCFG; Chiang, 2007).[3] An SCFG is a set of productions of the form $X \rightarrow (\alpha, \beta, \sim)$, where X is a non-terminal, α and β are sequences of terminals or non-terminals, and $\sim$ is a one-to-one alignment of items in α with items in β. English-Spanish adjective-noun ordering can be handled by a set of synchronous productions, e.g.,

$$\text{NP} \rightarrow (\text{DET}_1\ \text{NN}_2\ \text{JJ}_3, \quad \text{DET}_1\ \text{JJ}_3\ \text{NN}_2), \tag{18.22}$$

with subscripts indicating the alignment between the Spanish (left) and English (right) parts of the right-hand side. Terminal productions yield translation pairs,

$$\text{JJ} \rightarrow (\textit{enojado}_1, \quad \textit{angry}_1). \tag{18.23}$$

A synchronous derivation begins with the start symbol S, and derives a pair of sequences of terminal symbols.

Given an SCFG in which each production yields at most two symbols in each language (Chomsky Normal Form; see § 9.2.1), a sentence can be parsed using only the CKY algorithm (chapter 10). The resulting derivation also includes productions in the other language, all the way down to the surface form. Therefore, SCFGs make translation very similar to parsing. In a weighted SCFG, the log probability $\log \text{p}_{S|T}$ can be computed from the sum of the log-probabilities of the productions. However, combining SCFGs with a target language model is computationally expensive, necessitating approximate search algorithms (Huang and Chiang, 2007).

Synchronous context-free grammars are an example of **tree-to-tree translation**, because they model the syntactic structure of both the target and source language. In **string-to-tree translation**, string elements are translated into constituent tree fragments, which

[3] Earlier approaches to syntactic machine translation includes syntax-driven transduction (Lewis II and Stearns, 1968) and stochastic inversion transduction grammars (Wu, 1997).

Under contract with MIT Press, shared under CC-BY-NC-ND license.

are then assembled into a translation (Yamada and Knight, 2001; Galley et al., 2004); in **tree-to-string translation**, the source side is parsed, and then transformed into a string on the target side (Liu et al., 2006). A key question for syntax-based translation is the extent to which we phrasal constituents align across translations (Fox, 2002), because this governs the extent to which we can rely on monolingual parsers and treebanks. For more on syntax-based machine translation, see the monograph by Williams et al. (2016).

18.3 Neural machine translation

Neural network models for machine translation are based on the **encoder-decoder** architecture (Cho et al., 2014). The encoder network converts the source language sentence into a vector or matrix representation; the decoder network then converts the encoding into a sentence in the target language.

$$\boldsymbol{z} = \textsc{Encode}(\boldsymbol{w}^{(s)}) \tag{18.24}$$

$$\boldsymbol{w}^{(t)} \mid \boldsymbol{w}^{(s)} \sim \textsc{Decode}(\boldsymbol{z}), \tag{18.25}$$

where the second line means that the function $\textsc{Decode}(\boldsymbol{z})$ defines the conditional probability $\text{p}(\boldsymbol{w}^{(t)} \mid \boldsymbol{w}^{(s)})$.

The decoder is typically a recurrent neural network, which generates the target language sentence one word at a time, while recurrently updating a hidden state. The encoder and decoder networks are trained end-to-end from parallel sentences. If the output layer of the decoder is a logistic function, then the entire architecture can be trained to maximize the conditional log-likelihood,

$$\log \text{p}(\boldsymbol{w}^{(t)} \mid \boldsymbol{w}^{(s)}) = \sum_{m=1}^{M^{(t)}} \text{p}(w_m^{(t)} \mid \boldsymbol{w}_{1:m-1}^{(t)}, \boldsymbol{z}) \tag{18.26}$$

$$\text{p}(w_m^{(t)} \mid \boldsymbol{w}_{1:m-1}^{(t)}, \boldsymbol{w}^{(s)}) \propto \exp\left(\boldsymbol{\beta}_{w_m^{(t)}} \cdot \boldsymbol{h}_{m-1}^{(t)}\right) \tag{18.27}$$

where the hidden state $\boldsymbol{h}_{m-1}^{(t)}$ is a recurrent function of the previously generated text $\boldsymbol{w}_{1:m-1}^{(t)}$ and the encoding $\boldsymbol{z}$. The second line is equivalent to writing,

$$w_m^{(t)} \mid \boldsymbol{w}_{1:m-1}^{(t)}, \boldsymbol{w}^{(s)} \sim \text{SoftMax}\left(\boldsymbol{\beta} \cdot \boldsymbol{h}_{m-1}^{(t)}\right), \tag{18.28}$$

where $\boldsymbol{\beta} \in \mathbb{R}^{(V^{(t)} \times K)}$ is the matrix of output word vectors for the $V^{(t)}$ words in the target language vocabulary.

The simplest encoder-decoder architecture is the **sequence-to-sequence model** (Sutskever et al., 2014). In this model, the encoder is set to the final hidden state of a **long short-term**

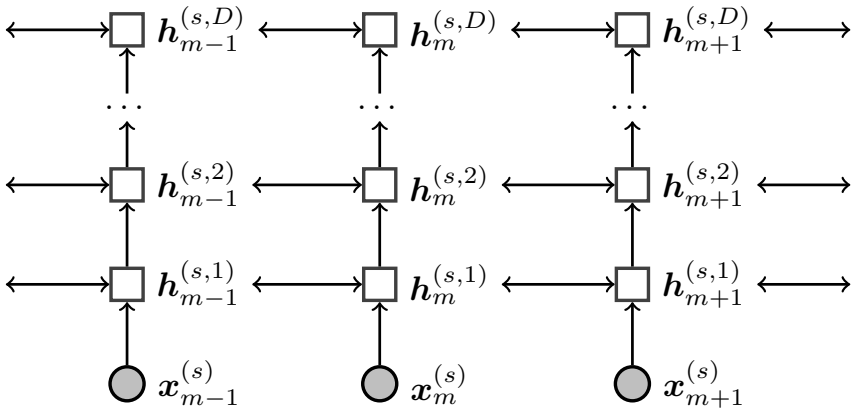

Figure 18.5: A deep bidirectional LSTM encoder

memory (LSTM) (see § 6.3.3) on the source sentence:

$$\boldsymbol{h}_m^{(s)} = \text{LSTM}(\boldsymbol{x}_m^{(s)}, \boldsymbol{h}_{m-1}^{(s)}) \quad [18.29]$$

$$\boldsymbol{z} \triangleq \boldsymbol{h}_{M^{(s)}}^{(s)}, \quad [18.30]$$

where $\boldsymbol{x}_m^{(s)}$ is the embedding of source language word $w_m^{(s)}$. The encoding then provides the initial hidden state for the decoder LSTM:

$$\boldsymbol{h}_0^{(t)} = \boldsymbol{z} \quad [18.31]$$

$$\boldsymbol{h}_m^{(t)} = \text{LSTM}(\boldsymbol{x}_m^{(t)}, \boldsymbol{h}_{m-1}^{(t)}), \quad [18.32]$$

where $\boldsymbol{x}_m^{(t)}$ is the embedding of the target language word $w_m^{(t)}$.

Sequence-to-sequence translation is nothing more than wiring together two LSTMs: one to read the source, and another to generate the target. To make the model work well, some additional tweaks are needed:

- Most notably, the model works much better if the source sentence is reversed, reading from the end of the sentence back to the beginning. In this way, the words at the beginning of the source have the greatest impact on the encoding $\boldsymbol{z}$, and therefore impact the words at the beginning of the target sentence. Later work on more advanced encoding models, such as **neural attention** (see § 18.3.1), has eliminated the need for reversing the source sentence.
- The encoder and decoder can be implemented as **deep LSTMs**, with multiple layers of hidden states. As shown in Figure 18.5, each hidden state $\boldsymbol{h}_m^{(s,i)}$ at layer i is treated

Under contract with MIT Press, shared under CC-BY-NC-ND license.

as the input to an LSTM at layer $i + 1$:

$$\boldsymbol{h}_m^{(s,1)} = \text{LSTM}(\boldsymbol{x}_m^{(s)}, \boldsymbol{h}_{m-1}^{(s)}) \quad [18.33]$$

$$\boldsymbol{h}_m^{(s,i+1)} = \text{LSTM}(\boldsymbol{h}_m^{(s,i)}, \boldsymbol{h}_{m-1}^{(s,i+1)}), \quad \forall i \geq 1. \quad [18.34]$$

The original work on sequence-to-sequence translation used four layers; in 2016, Google's commercial machine translation system used eight layers (Wu et al., 2016).[4]

- Significant improvements can be obtained by creating an **ensemble** of translation models, each trained from a different random initialization. For an ensemble of size N, the per-token decoding probability is set equal to,

$$\text{p}(w^{(t)} \mid \boldsymbol{z}, \boldsymbol{w}_{1:m-1}^{(t)}) = \frac{1}{N} \sum_{i=1}^{N} \text{p}_i(w^{(t)} \mid \boldsymbol{z}, \boldsymbol{w}_{1:m-1}^{(t)}), \quad [18.35]$$

where p_i is the decoding probability for model i. Each translation model in the ensemble includes its own encoder and decoder networks.

- The original sequence-to-sequence model used a fairly standard training setup: stochastic gradient descent with an exponentially decreasing learning rate after the first five epochs; mini-batches of 128 sentences, chosen to have similar length so that each sentence on the batch will take roughly the same amount of time to process; gradient clipping (see § 3.3.4) to ensure that the norm of the gradient never exceeds some predefined value.

18.3.1 Neural attention

The sequence-to-sequence model discussed in the previous section was a radical departure from statistical machine translation, in which each word or phrase in the target language is conditioned on a single word or phrase in the source language. Both approaches have advantages. Statistical translation leverages the idea of compositionality — translations of large units should be based on the translations of their component parts — and this seems crucial if we are to scale translation to longer units of text. But the translation of each word or phrase often depends on the larger context, and encoder-decoder models capture this context at the sentence level.

Is it possible for translation to be both contextualized and compositional? One approach is to augment neural translation with an **attention mechanism**. The idea of neural attention was described in § 17.5, but its application to translation bears further discussion. In general, attention can be thought of as using a query to select from a memory of key-value pairs. However, the query, keys, and values are all vectors, and the entire

[4] Google reports that this system took six days to train for English-French translation, using 96 NVIDIA K80 GPUs, which would have cost roughly half a million dollars at the time.

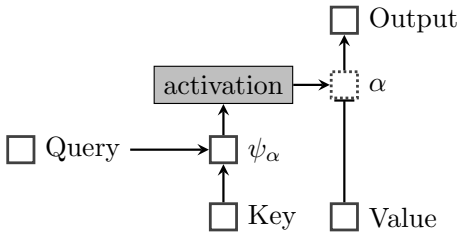

Figure 18.6: A general view of neural attention. The dotted box indicates that each $\alpha_{m\to n}$ can be viewed as a **gate** on value n.

operation is differentiable. For each key n in the memory, we compute a score $\psi_\alpha(m, n)$ with respect to the query m. That score is a function of the compatibility of the key and the query, and can be computed using a small feedforward neural network. The vector of scores is passed through an activation function, such as softmax. The output of this activation function is a vector of non-negative numbers $[\alpha_{m\to 1}, \alpha_{m\to 2}, \ldots, \alpha_{m\to N}]^\top$, with length N equal to the size of the memory. Each value in the memory $\boldsymbol{v}_n$ is multiplied by the attention $\alpha_{m\to n}$; the sum of these scaled values is the output. This process is shown in Figure 18.6. In the extreme case that $\alpha_{m\to n} = 1$ and $\alpha_{m\to n'} = 0$ for all other n', then the attention mechanism simply selects the value $\boldsymbol{v}_n$ from the memory.

Neural attention makes it possible to integrate alignment into the encoder-decoder architecture. Rather than encoding the entire source sentence into a fixed length vector $\boldsymbol{z}$, it can be encoded into a matrix $\mathbf{Z} \in \mathbb{R}^{K\times M^{(S)}}$, where K is the dimension of the hidden state, and $M^{(S)}$ is the number of tokens in the source input. Each column of $\mathbf{Z}$ represents the state of a recurrent neural network over the source sentence. These vectors are constructed from a **bidirectional LSTM** (see § 7.6), which can be a deep network as shown in Figure 18.5. These columns are both the keys and the values in the attention mechanism.

At each step m in decoding, the attentional state is computed by executing a query, which is equal to the state of the decoder, $\boldsymbol{h}_m^{(t)}$. The resulting compatibility scores are,

$$\psi_\alpha(m, n) = \boldsymbol{v}_\alpha \cdot \tanh(\Theta_\alpha[\boldsymbol{h}_m^{(t)}; \boldsymbol{h}_n^{(s)}]). \quad [18.36]$$

The function ψ is thus a two layer feedforward neural network, with weights $\boldsymbol{v}_\alpha$ on the output layer, and weights $\boldsymbol{\Theta}_\alpha$ on the input layer. To convert these scores into attention weights, we apply an activation function, which can be vector-wise softmax or an element-wise sigmoid:

Softmax attention

$$\alpha_{m\to n} = \frac{\exp \psi_\alpha(m, n)}{\sum_{n'=1}^{M^{(s)}} \exp \psi_\alpha(m, n')} \quad [18.37]$$

Under contract with MIT Press, shared under CC-BY-NC-ND license.

Sigmoid attention

$$\alpha_{m\rightarrow n} = \sigma\left(\psi_\alpha(m,n)\right) \tag{18.38}$$

The attention $\boldsymbol{\alpha}$ is then used to compute a context vector $\boldsymbol{c}_m$ by taking a weighted average over the columns of $\mathbf{Z}$,

$$\boldsymbol{c}_m = \sum_{n=1}^{M^{(s)}} \alpha_{m\rightarrow n}\boldsymbol{z}_n, \tag{18.39}$$

where $\alpha_{m\rightarrow n} \in [0,1]$ is the amount of attention from word m of the target to word n of the source. The context vector can be incorporated into the decoder's word output probability model, by adding another layer to the decoder (Luong et al., 2015):

$$\tilde{\boldsymbol{h}}_m^{(t)} = \tanh\left(\Theta_c[\boldsymbol{h}_m^{(t)}; \boldsymbol{c}_m]\right) \tag{18.40}$$

$$\text{p}(w_{m+1}^{(t)} \mid \boldsymbol{w}_{1:m}^{(t)}, \boldsymbol{w}^{(s)}) \propto \exp\left(\boldsymbol{\beta}_{w_{m+1}^{(t)}} \cdot \tilde{\boldsymbol{h}}_m^{(t)}\right). \tag{18.41}$$

Here the decoder state $\boldsymbol{h}_m^{(t)}$ is concatenated with the context vector, forming the input to compute a final output vector $\tilde{\boldsymbol{h}}_m^{(t)}$. The context vector can be incorporated into the decoder recurrence in a similar manner (Bahdanau et al., 2014).

18.3.2 *Neural machine translation without recurrence

In the encoder-decoder model, attention's "keys and values" are the hidden state representations in the encoder network, $\boldsymbol{z}$, and the "queries" are state representations in the decoder network $\boldsymbol{h}^{(t)}$. It is also possible to completely eliminate recurrence from neural translation, by applying **self-attention** (Lin et al., 2017; Kim et al., 2017) within the encoder and decoder, as in the **transformer architecture** (Vaswani et al., 2017). For level i, the basic equations of the encoder side of the transformer are:

$$\boldsymbol{z}_m^{(i)} = \sum_{n=1}^{M^{(s)}} \alpha_{m\rightarrow n}^{(i)}(\Theta_v \boldsymbol{h}_n^{(i-1)}) \tag{18.42}$$

$$\boldsymbol{h}_m^{(i)} = \Theta_2 \,\text{ReLU}\left(\Theta_1 \boldsymbol{z}_m^{(i)} + \boldsymbol{b}_1\right) + \boldsymbol{b}_2. \tag{18.43}$$

For each token m at level i, we compute self-attention over the entire source sentence. The keys, values, and queries are all projections of the vector $\boldsymbol{h}^{(i-1)}$: for example, in Equation 18.42, the value $\boldsymbol{v}_n$ is the projection $\Theta_v \boldsymbol{h}_n^{(i-1)}$. The attention scores $\alpha_{m\rightarrow n}^{(i)}$ are computed using a scaled form of softmax attention,

$$\alpha_{m\rightarrow n} \propto \exp(\psi_\alpha(m,n)/M), \tag{18.44}$$

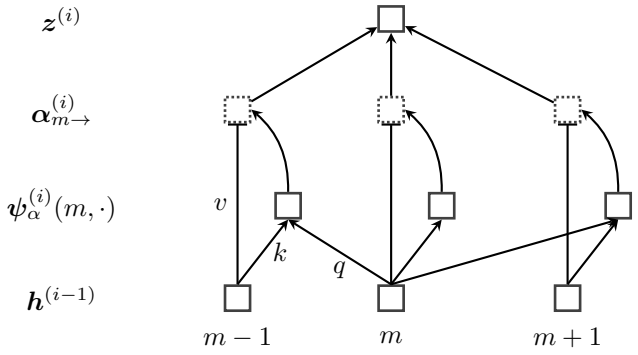

Figure 18.7: The transformer encoder's computation of $\boldsymbol{z}_m^{(i)}$ from $\boldsymbol{h}^{(i-1)}$. The **k**ey, **v**alue, and **q**uery are shown for token $m-1$. For example, $\psi_\alpha^{(i)}(m, m-1)$ is computed from the key $\Theta_k \boldsymbol{h}_{m-1}^{(i-1)}$ and the query $\Theta_q \boldsymbol{h}_m^{(i-1)}$, and the gate $\alpha_{m\rightarrow m-1}^{(i)}$ operates on the value $\Theta_v \boldsymbol{h}_{m-1}^{(i-1)}$. The figure shows a minimal version of the architecture, with a single attention head. With multiple heads, it is possible to attend to different properties of multiple words.

where M is the length of the input. This encourages the attention to be more evenly dispersed across the input. Self-attention is applied across multiple "heads", each using different projections of $\boldsymbol{h}^{(i-1)}$ to form the keys, values, and queries. This architecture is shown in Figure 18.7. The output of the self-attentional layer is the representation $\boldsymbol{z}_m^{(i)}$, which is then passed through a two-layer feed-forward network, yielding the input to the next layer, $\boldsymbol{h}^{(i)}$. This self-attentional architecture can be applied in the decoder as well, but this requires that there is zero attention to future words: $\alpha_{m\rightarrow n} = 0$ for all $n > m$.

To ensure that information about word order in the source is integrated into the model, the encoder augments the base layer of the network with **positional encodings** of the indices of each word in the source. These encodings are vectors for each position $m \in \{1, 2, \ldots, M\}$. The transformer sets these encodings equal to a set of sinusoidal functions of m,

$$e_{2i-1}(m) = \sin(m/(10000^{\frac{2i}{K_e}})) \quad [18.45]$$

$$e_{2i}(m) = \cos(m/(10000^{\frac{2i}{K_e}})), \quad \forall i \in \{1, 2, \ldots, K_e/2\} \quad [18.46]$$

where $e_{2i}(m)$ is the value at element $2i$ of the encoding for index m. As we progress through the encoding, the sinusoidal functions have progressively narrower bandwidths. This enables the model to learn to attend by relative positions of words. The positional encodings are concatenated with the word embeddings $\boldsymbol{x}_m$ at the base layer of the model.[5]

[5]The transformer architecture relies on several additional tricks, including **layer normalization** (see

Under contract with MIT Press, shared under CC-BY-NC-ND license.

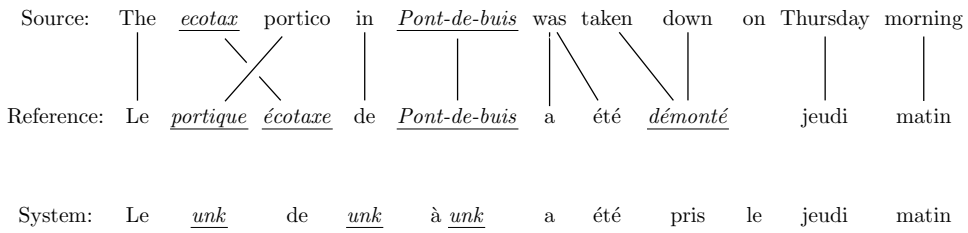

Figure 18.8: Translation with *unknown words*. The system outputs *unk* to indicate words that are outside its vocabulary. Figure adapted from Luong et al. (2015).

Convolutional neural networks (see § 3.4) have also been applied as encoders in neural machine translation (Gehring et al., 2017). For each word $w_m^{(s)}$, a convolutional network computes a representation $\boldsymbol{h}_m^{(s)}$ from the embeddings of the word and its neighbors. This procedure is applied several times, creating a deep convolutional network. The recurrent decoder then computes a set of attention weights over these convolutional representations, using the decoder's hidden state $\boldsymbol{h}^{(t)}$ as the queries. This attention vector is used to compute a weighted average over the outputs of *another* convolutional neural network of the source, yielding an averaged representation $\boldsymbol{c}_m$, which is then fed into the decoder. As with the transformer, speed is the main advantage over recurrent encoding models; another similarity is that word order information is approximated through the use of positional encodings.[6]

18.3.3 Out-of-vocabulary words

Thus far, we have treated translation as a problem at the level of words or phrases. For words that do not appear in the training data, all such models will struggle. There are two main reasons for the presence of out-of-vocabulary (OOV) words:

- New proper nouns, such as family names or organizations, are constantly arising — particularly in the news domain. The same is true, to a lesser extent, for technical terminology. This issue is shown in Figure 18.8.
- In many languages, words have complex internal structure, known as **morphology**. An example is German, which uses compounding to form nouns like *Abwasserbehandlungsanlage* (*sewage water treatment plant*; example from Sennrich et al. (2016)).

§ 3.3.4), residual connections around the nonlinear activations (see § 3.2.2), and a non-monotonic learning rate schedule.

[6] A recent evaluation found that best performance was obtained by using a recurrent network for the decoder, and a transformer for the encoder (Chen et al., 2018). The transformer was also found to significantly outperform a convolutional neural network.

While compounds could in principle be addressed by better tokenization (see § 8.4), other morphological processes involve more complex transformations of subword units.

Names and technical terms can be handled in a postprocessing step: after first identifying alignments between unknown words in the source and target, we can look up each aligned source word in a dictionary, and choose a replacement (Luong et al., 2015). If the word does not appear in the dictionary, it is likely to be a proper noun, and can be copied directly from the source to the target. This approach can also be integrated directly into the translation model, rather than applying it as a postprocessing step (Jean et al., 2015).

Words with complex internal structure can be handled by translating subword units rather than entire words. A popular technique for identifying subword units is **byte-pair encoding** (BPE; Gage, 1994; Sennrich et al., 2016). The initial vocabulary is defined as the set of characters used in the text. The most common character bigram is then merged into a new symbol, the vocabulary is updated, and the merging operation is applied again. For example, given the dictionary {*fish, fished, want, wanted, bike, biked*}, we would first form the subword unit *ed*, since this character bigram appears in three of the six words. Next, there are several bigrams that each appear in a pair of words: *fi*, *is*, *sh*, *wa*, *an*, etc. These can be merged in any order. By iterating this process, we eventually reach the segmentation, {*fish*, *fish+ed*, *want*, *want+ed*, *bik+e*, *bik+ed*}. At this point, there are no bigrams that appear more than once. In real data, merging is performed until the number of subword units reaches some predefined threshold, such as 10^4.

Each subword unit is treated as a token for translation, in both the encoder (source side) and decoder (target side). BPE can be applied jointly to the union of the source and target vocabularies, identifying subword units that appear in both languages. For languages that have different scripts, such as English and Russian, **transliteration** between the scripts should be applied first.[7]

18.4 Decoding

Given a trained translation model, the decoding task is:

$$\hat{\boldsymbol{w}}^{(t)} = \underset{\boldsymbol{w} \in \mathcal{V}^*}{\text{argmax}}\, \Psi(\boldsymbol{w}, \boldsymbol{w}^{(s)}), \tag{18.47}$$

where $\boldsymbol{w}^{(t)}$ is a sequence of tokens from the target vocabulary $\mathcal{V}$. It is not possible to efficiently obtain exact solutions to the decoding problem, for even minimally effective

[7] Transliteration is crucial for converting names and other foreign words between languages that do not share a single script, such as English and Japanese. It is typically approached using the finite-state methods discussed in chapter 9 (Knight and Graehl, 1998).

Under contract with MIT Press, shared under CC-BY-NC-ND license.

models in either statistical or neural machine translation. Today's state-of-the-art translation systems use **beam search** (see § 11.3.1), which is an incremental decoding algorithm that maintains a small constant number of competitive hypotheses. Such greedy approximations are reasonably effective in practice, and this may be in part because the decoding objective is only loosely correlated with measures of translation quality, so that exact optimization of [18.47] may not greatly improve the resulting translations.

Decoding in neural machine translation is simpler than in phrase-based statistical machine translation.[8] The scoring function Ψ is defined,

$$\Psi(\boldsymbol{w}^{(t)}, \boldsymbol{w}^{(s)}) = \sum_{m=1}^{M^{(t)}} \psi(w_m^{(t)}; \boldsymbol{w}_{1:m-1}^{(t)}, \boldsymbol{z}) \quad [18.48]$$

$$\psi(w^{(t)}; \boldsymbol{w}_{1:m-1}^{(t)}, \boldsymbol{z}) = \beta_{w_m^{(t)}} \cdot \boldsymbol{h}_m^{(t)} - \log \sum_{w \in \mathcal{V}} \exp\left(\beta_w \cdot \boldsymbol{h}_m^{(t)}\right), \quad [18.49]$$

where $\boldsymbol{z}$ is the encoding of the source sentence $\boldsymbol{w}^{(s)}$, and $\boldsymbol{h}_m^{(t)}$ is a function of the encoding $\boldsymbol{z}$ and the decoding history $\boldsymbol{w}_{1:m-1}^{(t)}$. This formulation subsumes the attentional translation model, where $\boldsymbol{z}$ is a matrix encoding of the source.

Now consider the incremental decoding algorithm,

$$\hat{w}_m^{(t)} = \operatorname*{argmax}_{w \in \mathcal{V}} \psi(w; \hat{\boldsymbol{w}}_{1:m-1}^{(t)}, \boldsymbol{z}), \quad m = 1, 2, \ldots \quad [18.50]$$

This algorithm selects the best target language word at position m, assuming that it has already generated the sequence $\hat{\boldsymbol{w}}_{1:m-1}^{(t)}$. (Termination can be handled by augmenting the vocabulary $\mathcal{V}$ with a special end-of-sequence token, ■.) The incremental algorithm is likely to produce a suboptimal solution to the optimization problem defined in Equation 18.47, because selecting the highest-scoring word at position m can set the decoder on a "garden path," in which there are no good choices at some later position $n > m$. We might hope for some dynamic programming solution, as in sequence labeling (§ 7.3). But the Viterbi algorithm and its relatives rely on a Markov decomposition of the objective function into a sum of local scores: for example, scores can consider locally adjacent tags (y_m, y_{m-1}), but not the entire tagging history $\boldsymbol{y}_{1:m}$. This decomposition is not applicable to recurrent neural networks, because the hidden state $\boldsymbol{h}_m^{(t)}$ is impacted by the entire history $\boldsymbol{w}_{1:m}^{(t)}$; this sensitivity to long-range context is precisely what makes recurrent neural networks so effective.[9] In fact, it can be shown that decoding from any recurrent neural network is NP-complete (Siegelmann and Sontag, 1995; Chen et al., 2018).

[8] For more on decoding in phrase-based statistical models, see Koehn (2009).

[9] Note that this problem does not impact RNN-based sequence labeling models (see § 7.6). This is because the tags produced by these models do not affect the recurrent state.

Beam search **Beam search** is a general technique for avoiding search errors when exhaustive search is impossible; it was first discussed in § 11.3.1. Beam search can be seen as a variant of the incremental decoding algorithm sketched in Equation 18.50, but at each step m, a set of K different hypotheses are kept on the beam. For each hypothesis $k \in \{1, 2, \ldots, K\}$, we compute both the current score $\sum_{m=1}^{M^{(t)}} \psi(w_{k,m}^{(t)}; \boldsymbol{w}_{k,1:m-1}^{(t)}, \boldsymbol{z})$ as well as the current hidden state $\boldsymbol{h}_k^{(t)}$. At each step in the beam search, the K top-scoring children of each hypothesis currently on the beam are "expanded", and the beam is updated. For a detailed description of beam search for RNN decoding, see Graves (2012).

Learning and search Conventionally, the learning algorithm is trained to predict the right token in the translation, conditioned on the translation history being correct. But if decoding must be approximate, then we might do better by modifying the learning algorithm to be robust to errors in the translation history. **Scheduled sampling** does this by training on histories that sometimes come from the ground truth, and sometimes come from the model's own output (Bengio et al., 2015).[10] As training proceeds, the training wheels come off: we increase the fraction of tokens that come from the model rather than the ground truth. Another approach is to train on an objective that relates directly to beam search performance (Wiseman et al., 2016). **Reinforcement learning** has also been applied to decoding of RNN-based translation models, making it possible to directly optimize translation metrics such as BLEU (Ranzato et al., 2016).

18.5 Training towards the evaluation metric

In likelihood-based training, the objective is the maximize the probability of a parallel corpus. However, translations are not evaluated in terms of likelihood: metrics like BLEU consider only the correctness of a single output translation, and not the range of probabilities that the model assigns. It might therefore be better to train translation models to achieve the highest BLEU score possible — to the extent that we believe BLEU measures translation quality. Unfortunately, BLEU and related metrics are not friendly for optimization: they are discontinuous, non-differentiable functions of the parameters of the translation model.

Consider an error function $\Delta(\hat{\boldsymbol{w}}^{(t)}, \boldsymbol{w}^{(t)})$, which measures the discrepancy between the system translation $\hat{\boldsymbol{w}}^{(t)}$ and the reference translation $\boldsymbol{w}^{(t)}$; this function could be based on BLEU or any other metric on translation quality. One possible criterion would be to select

[10] Scheduled sampling builds on earlier work on learning to search (Daumé III et al., 2009; Ross et al., 2011), which are also described in § 15.2.4.

Under contract with MIT Press, shared under CC-BY-NC-ND license.

the parameters $\boldsymbol{\theta}$ that minimize the error of the system's preferred translation,

$$\hat{\boldsymbol{w}}^{(t)} = \underset{\boldsymbol{w}^{(t)}}{\text{argmax}}\, \Psi(\boldsymbol{w}^{(t)}, \boldsymbol{w}^{(s)}; \boldsymbol{\theta}) \quad [18.51]$$

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\text{argmin}}\, \Delta(\hat{\boldsymbol{w}}^{(t)}, \boldsymbol{w}^{(s)}) \quad [18.52]$$

However, identifying the top-scoring translation $\hat{\boldsymbol{w}}^{(t)}$ is usually intractable, as described in the previous section. In **minimum error-rate training (MERT)**, $\hat{\boldsymbol{w}}^{(t)}$ is selected from a set of candidate translations $\mathcal{Y}(\boldsymbol{w}^{(s)})$; this is typically a strict subset of all possible translations, so that it is only possible to optimize an approximation to the true error rate (Och and Ney, 2003).

A further issue is that the objective function in Equation 18.52 is discontinuous and non-differentiable, due to the argmax over translations: an infinitesimal change in the parameters $\boldsymbol{\theta}$ could cause another translation to be selected, with a completely different error. To address this issue, we can instead minimize the **risk**, which is defined as the expected error rate,

$$R(\boldsymbol{\theta}) = E_{\hat{\boldsymbol{w}}^{(t)}|\boldsymbol{w}^{(s)};\boldsymbol{\theta}}[\Delta(\hat{\boldsymbol{w}}^{(t)}, \boldsymbol{w}^{(t)})] \quad [18.53]$$

$$= \sum_{\hat{\boldsymbol{w}}^{(t)} \in \mathcal{Y}(\boldsymbol{w}^{(s)})} \text{p}(\hat{\boldsymbol{w}}^{(t)} \mid \boldsymbol{w}^{(s)}) \times \Delta(\hat{\boldsymbol{w}}^{(t)}, \boldsymbol{w}^{(t)}). \quad [18.54]$$

Minimum risk training minimizes the sum of $R(\boldsymbol{\theta})$ across all instances in the training set.

The risk can be generalized by exponentiating the translation probabilities,

$$\tilde{p}(\boldsymbol{w}^{(t)}; \boldsymbol{\theta}, \alpha) \propto \Big(\text{p}(\boldsymbol{w}^{(t)} \mid \boldsymbol{w}^{(s)}; \boldsymbol{\theta})\Big)^{\alpha} \quad [18.55]$$

$$\tilde{R}(\boldsymbol{\theta}) = \sum_{\hat{\boldsymbol{w}}^{(t)} \in \mathcal{Y}(\boldsymbol{w}^{(s)})} \tilde{p}(\hat{\boldsymbol{w}}^{(t)} \mid \boldsymbol{w}^{(s)}; \alpha, \boldsymbol{\theta}) \times \Delta(\hat{\boldsymbol{w}}^{(t)}, \boldsymbol{w}^{(t)}) \quad [18.56]$$

where $\mathcal{Y}(\boldsymbol{w}^{(s)})$ is now the set of *all* possible translations for $\boldsymbol{w}^{(s)}$. Exponentiating the probabilities in this way is known as **annealing** (Smith and Eisner, 2006). When $\alpha = 1$, then $\tilde{R}(\boldsymbol{\theta}) = R(\boldsymbol{\theta})$; when $\alpha = \infty$, then $\tilde{R}(\boldsymbol{\theta})$ is equivalent to the sum of the errors of the maximum probability translations for each sentence in the dataset.

Clearly the set of candidate translations $\mathcal{Y}(\boldsymbol{w}^{(s)})$ is too large to explicitly sum over. Because the error function Δ generally does not decompose into smaller parts, there is no efficient dynamic programming solution to sum over this set. We can approximate the sum $\sum_{\hat{\boldsymbol{w}}^{(t)} \in \mathcal{Y}(\boldsymbol{w}^{(s)})}$ with a sum over a finite number of samples, $\{\boldsymbol{w}_1^{(t)}, \boldsymbol{w}_2^{(t)}, \ldots, \boldsymbol{w}_K^{(t)}\}$. If these samples were drawn uniformly at random, then the (annealed) risk would be

approximated as (Shen et al., 2016),

$$\tilde{R}(\boldsymbol{\theta}) \approx \frac{1}{Z} \sum_{k=1}^{K} \tilde{p}(\boldsymbol{w}_k^{(t)} \mid \boldsymbol{w}^{(s)}; \boldsymbol{\theta}, \alpha) \times \Delta(\boldsymbol{w}_k^{(t)}, \boldsymbol{w}^{(t)}) \quad [18.57]$$

$$Z = \sum_{k=1}^{K} \tilde{p}(\boldsymbol{w}_k^{(t)} \mid \boldsymbol{w}^{(s)}; \boldsymbol{\theta}, \alpha). \quad [18.58]$$

Shen et al. (2016) report that performance plateaus at $K = 100$ for minimum risk training of neural machine translation.

Uniform sampling over the set of all possible translations is undesirable, because most translations have very low probability. A solution from Monte Carlo estimation is **importance sampling**, in which we draw samples from a **proposal distribution** $q(\boldsymbol{w}^{(s)})$. This distribution can be set equal to the current translation model $\text{p}(\boldsymbol{w}^{(t)} \mid \boldsymbol{w}^{(s)}; \boldsymbol{\theta})$. Each sample is then weighted by an **importance score**, $\omega_k = \frac{\tilde{p}(\boldsymbol{w}_k^{(t)} | \boldsymbol{w}^{(s)})}{q(\boldsymbol{w}_k^{(t)}; \boldsymbol{w}^{(s)})}$. The effect of this weighting is to correct for any mismatch between the proposal distribution q and the true distribution $\tilde{p}$. The risk can then be approximated as,

$$\boldsymbol{w}_k^{(t)} \sim q(\boldsymbol{w}^{(s)}) \quad [18.59]$$

$$\omega_k = \frac{\tilde{p}(\boldsymbol{w}_k^{(t)} \mid \boldsymbol{w}^{(s)})}{q(\boldsymbol{w}_k^{(t)}; \boldsymbol{w}^{(s)})} \quad [18.60]$$

$$\tilde{R}(\boldsymbol{\theta}) \approx \frac{1}{\sum_{k=1}^{K} \omega_k} \sum_{k=1}^{K} \omega_k \times \Delta(\boldsymbol{w}_k^{(t)}, \boldsymbol{w}^{(t)}). \quad [18.61]$$

Importance sampling will generally give a more accurate approximation than uniform sampling. The only formal requirement is that the proposal assigns non-zero probability to every $\boldsymbol{w}^{(t)} \in \mathcal{Y}(\boldsymbol{w}^{(s)})$. For more on importance sampling and related methods, see Robert and Casella (2013).

Additional resources

A complete textbook on machine translation is available from Koehn (2009). While this book precedes recent work on neural translation, a more recent draft chapter on neural translation models is also available (Koehn, 2017). Neubig (2017) provides a comprehensive tutorial on neural machine translation, starting from first principles. The course notes from Cho (2015) are also useful. Several neural machine translation libraries are available: LAMTRAM is an implementation of neural machine translation in DYNET (Neubig et al., 2017); OPENNMT (Klein et al., 2017) and FAIRSEQ are available in PYTORCH;

Under contract with MIT Press, shared under CC-BY-NC-ND license.

TENSOR2TENSOR is an implementation of several of the Google translation models in TENSORFLOW (Abadi et al., 2016).

Literary translation is especially challenging, even for expert human translators. Messud (2014) describes some of these issues in her review of an English translation of *L'étranger*, the 1942 French novel by Albert Camus.[11] She compares the new translation by Sandra Smith against earlier translations by Stuart Gilbert and Matthew Ward, focusing on the difficulties presented by a single word in the first sentence:

> Then, too, Smith has reconsidered the book's famous opening. Camus's original is deceptively simple: "*Aujourd'hui, maman est morte*." Gilbert influenced generations by offering us "Mother died today"—inscribing in Meursault [the narrator] from the outset a formality that could be construed as heartlessness. But *maman*, after all, is intimate and affectionate, a child's name for his mother. Matthew Ward concluded that it was essentially untranslatable ("mom" or "mummy" being not quite apt), and left it in the original French: "Maman died today." There is a clear logic in this choice; but as Smith has explained, in an interview in *The Guardian*, *maman* "didn't really tell the reader anything about the connotation." She, instead, has translated the sentence as "My mother died today."
>
> > I chose "My mother" because I thought about how someone would tell another person that his mother had died. Meursault is speaking to the reader directly. "My mother died today" seemed to me the way it would work, and also implied the closeness of "maman" you get in the French.
>
> Elsewhere in the book, she has translated *maman* as "mama" — again, striving to come as close as possible to an actual, colloquial word that will carry the same connotations as *maman* does in French.

The passage is a reminder that while the quality of machine translation has improved dramatically in recent years, expert human translations draw on considerations that are beyond the ken of any contemporary computational approach.

Exercises

1. Using Google translate or another online service, translate the following example into two different languages of your choice:

[11] The book review is currently available online at `http://www.nybooks.com/articles/2014/06/05/camus-new-letranger/`.

(18.4) It is not down on any map; true places never are.

Then translate each result back into English. Which is closer to the original? Can you explain the differences?

2. Compute the unsmoothed n-gram precisions $p_1 \dots p_4$ for the two back-translations in the previous problem, using the original source as the reference. Your n-grams should include punctuation, and you should segment conjunctions like *it's* into two tokens.

3. You are given the following dataset of translations from "simple" to "difficult" English:

(18.5) a. *Kids like cats.*
Children adore felines.
b. *Cats hats.*
Felines fedoras.

Estimate a word-to-word statistical translation model from simple English (source) to difficult English (target), using the expectation-maximization as described in § 18.2.2. Compute two iterations of the algorithm by hand, starting from a uniform translation model, and using the simple alignment model $\text{p}(a_m \mid m, M^{(s)}, M^{(t)}) = \frac{1}{M^{(t)}}$. Hint: in the final M-step, you will want to switch from fractions to decimals.

4. Building on the previous problem, what will be the converged translation probability table? Can you state a general condition about the data, under which this translation model will fail in the way that it fails here?

5. Propose a simple alignment model that would make it possible to recover the correct translation probabilities from the toy dataset in the previous two problems.

6. Let $\ell_{m+1}^{(t)}$ represent the loss at word $m+1$ of the target, and let $\boldsymbol{h}_n^{(s)}$ represent the hidden state at word n of the source. Write the expression for the derivative $\frac{\partial \ell_{m+1}^{(t)}}{\partial \boldsymbol{h}_n^{(s)}}$ in the sequence-to-sequence translation model expressed in Equations [18.29-18.32]. You may assume that both the encoder and decoder are one-layer LSTMs. In general, how many terms are on the shortest backpropagation path from $\ell_{m+1}^{(t)}$ to $\boldsymbol{h}_n^{(s)}$?

7. Now consider the neural attentional model from § 18.3.1, with sigmoid attention. The derivative $\frac{\partial \ell_{m+1}^{(t)}}{\partial \boldsymbol{z}_n}$ is the sum of many paths through the computation graph; identify the shortest such path. You may assume that the initial state of the decoder recurrence $\boldsymbol{h}_0^{(t)}$ is *not* tied to the final state of the encoder recurrence $\boldsymbol{h}_{M^{(s)}}^{(s)}$.

Under contract with MIT Press, shared under CC-BY-NC-ND license.

8. Apply byte-pair encoding for the vocabulary *it*, *unit*, *unite*, until no bigram appears more than once.

9. This problem relates to the complexity of machine translation. Suppose you have an oracle that returns the list of words to include in the translation, so that your only task is to order the words. Furthermore, suppose that the scoring function over orderings is a sum over bigrams, $\sum_{m=1}^{M} \psi(\boldsymbol{w}_m^{(t)}, \boldsymbol{w}_{m-1}^{(t)})$. Show that the problem of finding the optimal translation is NP-complete, by reduction from a well-known problem.

10. Hand-design an attentional recurrent translation model that simply copies the input from the source to the target. You may assume an arbitrarily large hidden state, and you may assume that there is a finite maximum input length M. Specify all the weights such that the maximum probability translation of any source is the source itself. Hint: it is simplest to use the Elman recurrence $\boldsymbol{h}_m = f(\Theta \boldsymbol{h}_{m-1} + \boldsymbol{x}_m)$ rather than an LSTM.

11. Give a synchronized derivation (§ 18.2.4) for the Spanish-English translation,

    (18.6) *El pez enojado atacado.*
    The fish angry attacked.
    The angry fish attacked.

    As above, the second line shows a word-for-word gloss, and the third line shows the desired translation. Use the synchronized production rule in [18.22], and design the other production rules necessary to derive this sentence pair. You may derive (*atacado*, *attacked*) directly from VP.