Tong Xiao

Jingbo Zhu

Natural Language Processing

Neural Networks and Large Language Models

Natural Language Processing Lab
Northeastern University

&

NiuTrans Research

https://github.com/NiuTrans/NLPBook
https://niutrans.github.io/NLPBook

NATURAL LANGUAGE PROCESSING LAB, NORTHEASTERN UNIVERSITY
&
NIUTRANS RESEARCH

https://github.com/NiuTrans/NLPBook
https://niutrans.github.io/NLPBook

June 12, 2025

Tong Xiao and Jingbo Zhu
June, 2025

Chapter 5

Sequence-to-Sequence Models

天下万物之理，无独必有对。

According to the Principle of Heaven and Earth and all things, nothing exists in isolation but everything necessarily has its opposite.

– 《近思录》

Reflections on things at hand

朱熹/Xi Zhu (AD 1130-1200)
吕祖谦/Zuqian Lv (AD 1137-1181)
translated by Chang [1967]

In the language world, things often come in pairs. If there is a question, there would be an answer; if there is a Chinese text, there would be an English translation; if there is a sentence, there would be a parse of it according to some syntax. Many NLP systems are designed to model the correspondence between these pairs, i.e., one of the two is taken as input and the other is taken as output. These problems can be expressed in a form that we have encountered several times, like this

$$\hat{y} = \arg\max_{y} \Pr(y|x) \tag{5.1}$$

where x is an input variable, y is an output variable, and $\Pr(y|x)$ is a model that estimates how likely y would be the true output given x.

This chapter is more interested in a particular family of problems where both x and y are sequences of words, called **sequence-to-sequence** (or **seq2seq**) problems. Unlike classification problems where the output $\hat{y}$ is selected from a fixed set of classes, sequence-to-sequence problems require producing an output from an exponentially larger set of sequences. Obtaining $\hat{y}$ in this case turns out to be a much more complex problem than the case of classification, because we need more powerful models to describe $\Pr(y|x)$ and more efficient search algorithms to solve Eq. (5.1).

This chapter will discuss the well-known **encoder-decoder architecture** for sequence-to-sequence modeling. Also, this chapter will discuss the attention mechanism which is an improvement on this architecture. Both of these models lay the foundation of discussions of several state-of-the-art models in the following chapters. Furthermore, this chapter will discuss the search problem which plays an important role in sequence generation and related problems.

5.1 Sequence-to-Sequence Problems

We choose machine translation as an illustrative example throughout this chapter, because it is now one of the most popular sequence-to-sequence tasks. We use $\mathbf{x} = x_1...x_m$ to denote a sequence of words in one language (call it a **source-side sequence** or **source sequence**), and use $\mathbf{y} = y_1...y_n$ to denote a sequence of words in another language (call it a **target-side sequence** or **target sequence**). We can write Eq. (5.1) using the new notation, as follows

$$\begin{aligned} \hat{\mathbf{y}} &= \arg\max_{\mathbf{y}} \Pr(\mathbf{y}|\mathbf{x}) \\ &= \arg\max_{y_1...y_n} \Pr(y_1...y_n|x_1...x_m) \end{aligned} \tag{5.2}$$

As discussed in Chapter 1 and in [Brown et al., 1993], this formulation implies three fundamental issues.

- **Modeling**. First, we need to define the form of $\Pr(\mathbf{y}|\mathbf{x})$. In this chapter we show that $\Pr(\mathbf{y}|\mathbf{x})$ can be computed using a single neural network based on the encoder-decoder architecture and the attention mechanism. Note that sometimes we just need a model for discriminating “good” from “bad” target sequences. In this case, it is not necessary to require the model to make probability sense, and we can take a discriminant function instead.
- **Training**. Then, we need to learn parameters of the model $\Pr(\mathbf{y}|\mathbf{x})$ given some training data. As $\Pr(\mathbf{y}|\mathbf{x})$ is expressed as a neural network, we can train it in a regular way: we optimize some loss by gradient descent. See Chapter 3 for common approaches to training neural networks. We will also discuss techniques that are tailored for specific tasks in this and the following chapters.
- **Search** (or **Decoding**). Once we have learned a model, we will obtain $\hat{\mathbf{y}}$ by searching for the target sequence that maximizes $\Pr(\mathbf{y}|\mathbf{x})$. This is a computational challenge because the number of candidate sequences grows with the maximum length of the sequences and the size of the vocabulary. In Section 5.4, we will discuss efficient and effective search methods for sequence-to-sequence problems, particularly for machine translation.

Many NLP problems that fit the form of Eq. (5.2) can fall into sequence-to-sequence problems, and the research on these problems is largely motivated by discussions of the above issues. Table 5.1 shows common examples of sequence-to-sequence problems taken from the literature. When the target-side is a text, the problems can broadly be categorized as the **text generation** problems, although a general text generation system does not require the

Task	Source	Target
Machine Translation	Text	Translation
	in One Language	in Another Language
Question Answering	Question	Answer
Dialogue Systems	Text/Speech for Conversation	Response
Summarization	Long Text	Summaries of the Text
Text Simplification	Text	Simpler Text
Text Style Transfer	Text	Same Content
	in One Style	in Another Style
Grammar Correction	Text with Errors	Corrected Text
Speech Recognition	Speech	Transcription
Speech Synthesis	Text	Speech
Speech Translation	Speech	Translation
	in One Language	in Another Language

Table 5.1: Examples of sequence-to-sequence problems.

source-side to be sequential. In addition to language and speech processing, sequence-to-sequence problems can be generalized to cases where the input and/or output of a system are not naturally sequential. For example, **image-to-text generation** (or **image captioning**) and **text-to-image generation** systems both involve dealing with images that are typically represented as 2D data. By representing images as sequences in some way (such as sequences of patches), sequence-to-sequence models are directly applicable to these tasks.

Historically, most systems in these tasks were developed somewhat independently, resulting in different architectures, features, and training methods for different tasks. However, as shown in this chapter, when we represent these models as neural networks and train them in an end-to-end fashion, there appears to be a "universal" paradigm for all these problems. This is a big change for the AI community because many research fields come together and systems can be shared across them. We can gain some insight into the common nature of a broad variety of problems, though there are many task-specific considerations in practice. In the following sections, we will discuss some of the common threads among sequence-to-sequence models.

5.2 The Encoder-Decoder Architecture

In this section we discuss the encoder-decoder architecture and a simple neural machine translation model based on this architecture.

5.2.1 Encoding and Decoding

From a supervised learning viewpoint, we would ideally like to learn a model from a number of sequence pairs such that any source-side sequence can be mapped to the corresponding target-side sequence. However, learning the mapping between sequences of discrete variables is typically a problem of learning from high-dimensional data. It inevitably suffers from the curse of dimensionality, making the modeling and training difficult.

One approach to learning such a mapping is to divide the problem into "simpler" sub-problems. We assume that there is a low-dimensional representation shared by $\mathbf{x}$ and $\mathbf{y}$, denoted by $\mathbf{H}$. Then, the mapping $\mathbf{x} \rightarrow \mathbf{y}$ can be achieved by mapping $\mathbf{x}$ to $\mathbf{H}$ and then to $\mathbf{y}$. Formally, given a source-side sequence $\mathbf{x}$, we map it to the representation $\mathbf{H}$ by using an **encoding system** (call it an encoder)

$$\mathbf{H} = \mathrm{Encode}(\mathbf{x}) \tag{5.3}$$

Then, we map $\mathbf{H}$ to the target-side sequence $\mathbf{y}$ by using a **decoding system** (call it a decoder)[1]

$$\mathbf{y} = \mathrm{Decode}(\mathbf{H}) \tag{5.4}$$

This architecture, also known as the encoder-decoder architecture, is widely used in recent sequence-to-sequence systems (see Figure 5.1 for an illustration). It is easy to see that the form of Eq. (5.3) is the same as those of the sequence models mentioned in Chapter 4, and so there are many encoding models to choose from, such as bi-directional LSTM. The goal of the decoder is to produce a "best" target-side sequence given the representation of the source-side sequence. Like classification models, the prediction is made by first producing a distribution over all possible sequences, and then selecting the one with the maximum probability. As such, we can re-define $\mathrm{Decode}(\cdot)$ as a probability function

$$\begin{aligned} \Pr(\cdot|\mathbf{H}) &= \mathrm{Decode}(\mathbf{H}) \\ &= \mathrm{Decode}(\mathrm{Encode}(\mathbf{x})) \end{aligned} \tag{5.5}$$

In other words, given a target-side sequence $\mathbf{y}$, the decoder assigns it a probability

$$\Pr(\mathrm{y}|\mathbf{x}) = \Pr(\mathrm{y}|\mathbf{H}) \tag{5.6}$$

Then, the optimal sequence $\hat{\mathbf{y}}$ is obtained by performing $\arg\max_{\mathbf{y}} \Pr(\mathbf{y}|\mathbf{x})$ as in Eq. (5.2). In many systems based on the encoder-decoder architecture, both $\mathrm{Encode}(\cdot)$ and $\mathrm{Decode}(\cdot)$ are models constructed from neural networks. Thus, we can treat the sequence-to-sequence model

[1] It is important to distinguish between the concept of *decoding* (or *decoder*) used in conventional sequence-to-sequence systems and that used in the encoder-decoder architecture. The two are often confused, though they are different somehow. In many machine translation or speech recognition systems, *decoding* has the same meaning as *translation* or *transcription*, that is, we recover the optimal $\mathbf{y}$ from $\mathbf{x}$. As pointed out in Eq. (5.2), this process involves a search over all candidate $\mathbf{y}$. Therefore, the conventional use of decoding in these systems is to refer to a search process (i.e., the $\arg\max$ operation in Eq. (5.2)) [Koehn, 2010]. By contrast, in the encoder-decoder architecture *decoding* means a process of recovering the target-side sequence $\mathbf{y}$ from the intermediate representation $\mathbf{H}$. It is all about modeling rather than searching. It is also worth noting that, while the term *decoding* (or *decoder*) is used in different ways, it can be thought of as a process of mapping an encoded message back to the original message in a communication system as defined in information theory [Shannon, 1948]. In this sense, the *decoding* processes in these systems do the same thing as the word sounds like: convert something to its original form.

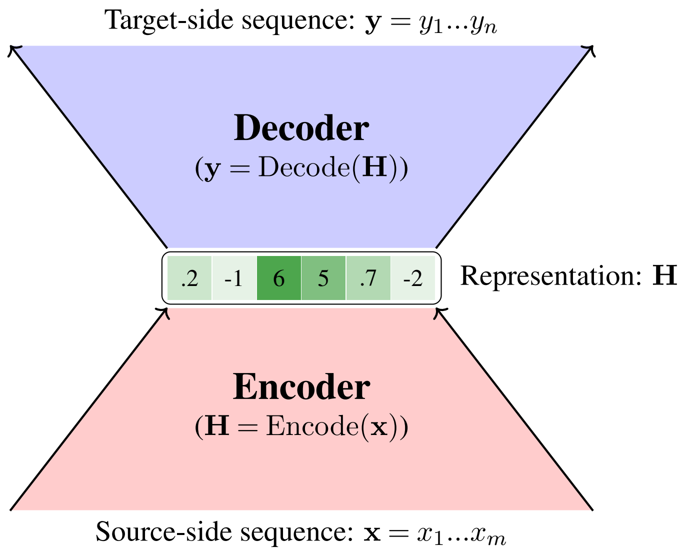


Figure 5.1: The encoder-decoder architecture. In the case of sequence-to-sequence problems, it transforms a source-side sequence $\mathbf{x} = x_1...x_m$ to a target-side sequence $\mathbf{y} = y_1...y_n$. This procedure involves two steps: $\mathbf{x}$ is first encoded as a representation $\mathbf{H}$, and this representation is then decoded to $\mathbf{y}$.

as a single neural network and train it as usual, provided the entire model is some combination of $\text{Encode}(\cdot)$ and $\text{Decode}(\cdot)$.

To apply the encoder-decoder architecture to a real-world task, we need to make a number of design choices, such as the forms of $\mathbf{H}$, $\text{Encode}(\cdot)$ and $\text{Decode}(\cdot)$. As a very simple example, consider the task of regenerating an input word. We can define $\text{Encode}(\cdot)$ as a feed-forward neural network that takes a word (in one-hot representation) and outputs a word vector. In this way, $\mathbf{H}$ is a distributed representation of the word. Then, we define $\text{Decode}(\cdot)$ as another feed-forward neural network that takes the word vector and generates a distribution over the vocabulary. For training, we wish to learn a system that assigns the largest probability to the input word. As discussed in Chapter 2, we can call this an auto-encoder which is a special instance of the encoder-decoder architecture.

5.2.2 Example: Neural Machine Translation

Next we illustrate the application of the encoder-decoder architecture using a working example — **neural machine translation** (**NMT**). We consider a well-known NMT model which uses RNN or its variants for building both the encoder and decoder [Cho et al., 2014; Sutskever et al., 2014]. The encoder of the NMT model is a standard RNN-based encoder. As the RNN-based sequence model has been discussed in detail in Chapter 4, we just give a brief review of this model here. Suppose that the source-side vocabulary is V_{x} and each source-side word x_j is represented as a one-hot vector in $\mathbb{R}^{|V_{\text{x}}|}$. Then, x_j is transformed into a h_s-dimensional vector

(or word embedding)

$$\mathbf{x}_j^{\mathrm{e}} = \mathrm{Embed}_{\mathrm{s}}(x_j) \tag{5.7}$$

where $\mathrm{Embed}_{\mathrm{s}}(\cdot)$ is the word embedding function. More details about word embedding models can be found in Chapter 3.

The RNN model takes the sequence of the word vectors $\mathbf{x}_1^{\mathrm{e}}...\mathbf{x}_m^{\mathrm{e}}$ and produces a sequence of RNN state vectors $\mathbf{h}_1...\mathbf{h}_m$. An RNN state vector $\mathbf{h}_j \in \mathbb{R}^{d_h}$ is defined to be

$$\mathbf{h}_j = \mathrm{RNN}(\mathbf{h}_{j-1}, \mathbf{x}_j^{\mathrm{e}}) \tag{5.8}$$

Here $\mathrm{RNN}(\cdot)$ is an RNN unit that summarises the information up to position j by combining the previous state $\mathbf{h}_{j-1}$ and the current input $\mathbf{x}_j^{\mathrm{e}}$ in some way. Then, the last state $\mathbf{h}_m$ can be treated as a representation of the input sequence $x_1...x_m$, and we can use $\mathbf{h}_m$ as the output of the encoder, written as

$$\mathbf{h}_m = \mathrm{Encode}(x_1...x_m) \tag{5.9}$$

Figure 5.2 (a-b) shows an illustration of the encoding process. Note that the model described above just involves a single-layer RNN. In practical systems, this framework can be easily extended to include multiple layers and more powerful recurrent units (such as LSTM units).

The decoder of the NMT model is a standard RNN-based language model, that is, we predict the next word y_{i+1} given all previous words $y_1...y_i$. To incorporate the source-side information into translation, a simple and straightforward method is to treat $\mathbf{h}_m$ as the initial state of the target-side RNN. Let $\mathbf{y}_0^{\mathrm{e}} \in \mathbb{R}^{d_s}$ be the word vector of the start symbol $\langle \mathrm{SOS} \rangle$ (denoted by y_0). The corresponding RNN state is given by

$$\mathbf{s}_0 = \mathrm{RNN}(\mathbf{h}_m, \mathbf{y}_0^{\mathrm{e}}) \tag{5.10}$$

Here RNN$(\cdot)$ has the same form as the recurrent unit used in the encoder, but with different parameters.

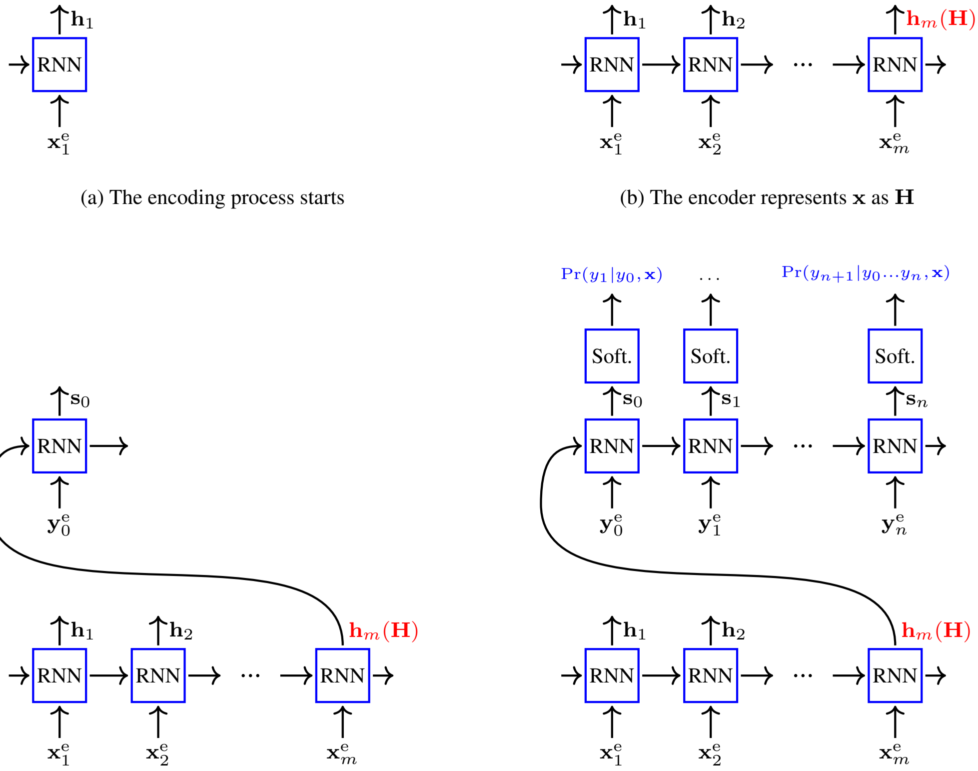
For $i > 0$, the state vector $\mathbf{s}_i \in \mathbb{R}^{d_s}$ is given in the form

$$\mathbf{s}_i = \mathrm{RNN}(\mathbf{s}_{i-1}, \mathbf{y}_i^{\mathrm{e}}) \tag{5.11}$$

Then, $\mathbf{s}_i$ is fed into a Softmax layer to produce a distribution over the target-side vocabulary V_{y}. The output of the Softmax layer is given by

$$\begin{aligned} \mathrm{Pr}(\cdot|y_1...y_i, x_1...x_m) &= \mathrm{Pr}(\cdot|\mathbf{s}_i) \\ &= \mathrm{Softmax}(\mathbf{s}_i\mathbf{U}_{\mathrm{y}} + \mathbf{b}_{\mathrm{y}}) \end{aligned} \tag{5.12}$$

where $\mathbf{U}_{\mathrm{y}} \in \mathbb{R}^{d_s \times |V_{\mathrm{y}}|}$ and $\mathbf{b}_{\mathrm{y}} \in \mathbb{R}^{|V_{\mathrm{y}}|}$ are the parameters of the Softmax layer. $\mathrm{Pr}(y_{i+1}|y_1...y_i, x_1...x_m)$ can be seen as the probability of predicting word y_{i+1} by conditioning on both the translated



(a) The encoding process starts

(b) The encoder represents $\mathbf{x}$ as $\mathbf{H}$

(c) The decoder takes the representation of $\mathbf{x}$

(d) The decoder predicts the target-side words

Figure 5.2: The encoding and decoding steps for an RNN-based NMT system. The encoder is a standard RNN. The encoding process starts with the first source-side word and ends up with the last source-side word. The last state of the RNN is taken to be the representation of the entire source-side sequence (i.e., $\mathbf{H} = \mathbf{h}_m$). The decoder is another RNN. At the first step, it takes $\mathbf{H}$ from the encoder. After representing $(\mathbf{y}_0^e...\mathbf{h}_i^e, \mathbf{H})$ as $\mathbf{s}_i$ at position i, a softmax layer is built to predict the next word y_{i+1}.

words $y_1...y_i$ and the source-side sequence $x_1...x_m$. See Figure 5.2 (c-d) for an illustration of the word predictions of a decoder.

Armed with this model of word prediction, we turn to a form that is frequently used in papers on NMT, like this

$$
\begin{aligned}
\Pr(\mathbf{y}|\mathbf{x}) &= \Pr(y_0\mathbf{y}|\mathbf{x}) \\
&= \Pr(y_0 y_1...y_n|x_1...x_m) \\
&= \Pr(y_0|x_1...x_m)\Pr(y_1...y_n|y_0, x_1...x_m) \\
&= \prod_{i=0}^{n-1}\Pr(y_{i+1}|y_0...y_i, x_1...x_m)
\end{aligned}
\tag{5.13}
$$

Sometimes, this equation is also written in an equivalent form

$$\Pr(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{n} \Pr(y_i|y_0...y_{i-1}, x_1...x_m) \tag{5.14}$$

Here we assume that $\mathbf{y}$ always starts with y_0 (i.e., $\langle\text{SOS}\rangle$) and so $\Pr(y_0|x_1...x_m) = 1$. In many practical systems, it is also common to assume that $\mathbf{y}$ ends with a special symbol $\langle\text{EOS}\rangle$. Therefore, we can modify this equation to involve $\langle\text{SOS}\rangle$ and $\langle\text{EOS}\rangle$ on both the source and target-sides, as follows

$$\begin{aligned} \Pr(y_0\mathbf{y}y_{n+1}|x_0\mathbf{x}x_{m+1}) &= \Pr(y_0y_1...y_ny_{n+1}|x_0x_1...x_mx_{m+1}) \\ &= \Pr(y_0|x_0...x_{m+1}) \cdot \\ &\quad\ \Pr(y_1...y_ny_{n+1}|y_0, x_0...x_{m+1}) \\ &= \prod_{i=0}^{n} \Pr(y_{i+1}|y_0...y_i, x_0...x_{m+1}) \end{aligned} \tag{5.15}$$

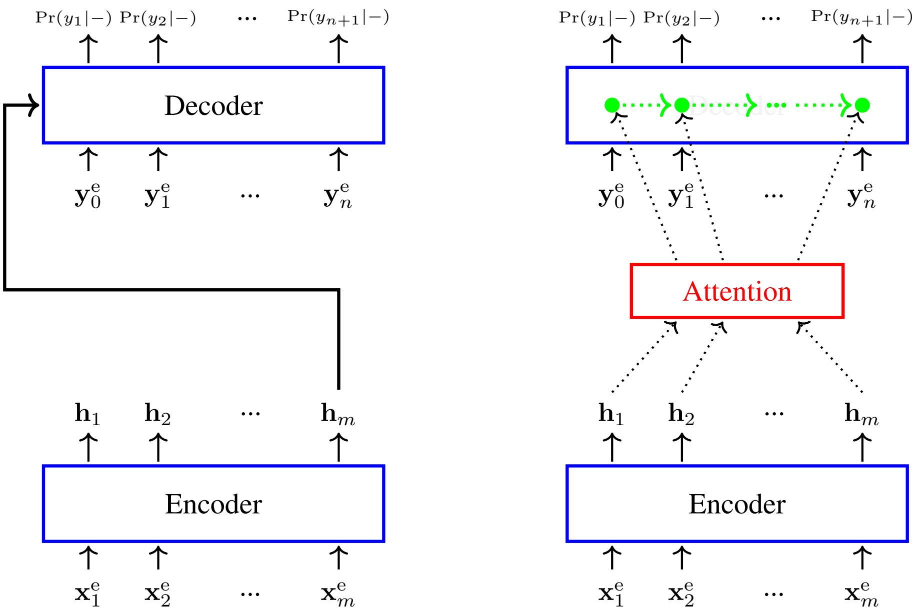
where $x_0 = y_0 = \langle\text{SOS}\rangle$, $x_{m+1} = y_{n+1} = \langle\text{EOS}\rangle$, and $\Pr(y_0|x_0x_1...x_mx_{m+1}) = 1$.

Since $\Pr(\mathbf{y}|\mathbf{x})$ can be expressed as a neural network, training this model is straightforward. As described in Chapter 4, RNN-based language models are trained by using the cross-entropy loss and gradient descent. NMT can use this same method for training model parameters. Once we have obtained the optimized model, we can then use it to translate new sentences. Finding the best translation for any given source-side sentence is a standard search problem. We will discuss it in Section 5.4.

5.3 The Attention Mechanism

The NMT model discussed in the previous section was based on a fixed-length representation of the source-side sequence. While this model is easy to implement, in many practical applications it is unsatisfactory because a fixed-length vector might not be sufficient for representing a variable-length sequence, especially when the sequence is long. This system will therefore need some mechanism to couple the encoder and the decoder in a fine-grained manner. In this section we discuss the attention mechanism by which a system can learn, for each word of the target-side sequence, an adaptive representation that focuses more on important parts of the source-side sequence.

In fact, the discussion here is related to the attention models in psychology because translation is itself a cognitive process [Sternberg, 1996; Neisser, 2014]. The key idea behind this type of model is natural: attention is generally concentrated on specific parts of the data when we process something. This forms the basis of many state-of-the-art sequence-to-sequence models, and the attention mechanism has been the de facto standard for the development of these systems.



(a) An NMT system without attention (b) An NMT system with attention

Figure 5.3: NMT architectures without (left) and with (right) the attention model. When the attention model is not involved, a fixed-length representation is considered for generating the entire target-side sequence. By contrast, when the attention model is involved, a new representation is computed specifically for each target-side state so that the decoder can learn to concentrate on different parts of the source-side sequence for predicting a target-side word.

5.3.1 A Basic Model

Recall that in the NMT model of the previous section, the encoder represents a source-side word sequence as $\mathbf{h}_1 ... \mathbf{h}_m$, and the decoder represents a target-side word sequence as $\mathbf{s}_1 ... \mathbf{s}_n$. The attention mechanism addresses the question of how a representation can be learned from $\mathbf{h}_1 ... \mathbf{h}_m$ so that this representation can explain the source-side sequence well for a given target state $\mathbf{s}_i$[2]. From an information processing perspective, so long as we ignore the meanings of $\mathbf{h}_1 ... \mathbf{h}_m$ and $\mathbf{s}_i$ in NMT, attention can be thought of as a generic process of processing the input information $\mathbf{h}_1 ... \mathbf{h}_m$ by considering how each $\mathbf{h}_j$ is related to the interest $\mathbf{s}_i$. Figure 5.3 compares NMT architectures with and without the attention mechanism.

More formally, an attention model produces a linear combination of $\{\mathbf{h}_1, ..., \mathbf{h}_m\}$ in the form

$$\mathbf{c}_i = \sum_{j=1}^{m} \alpha_{i,j} \cdot \mathbf{h}_j \tag{5.16}$$

where $\alpha_{i,j}$ is the **attention weight** that describes how much the model should rely on $\mathbf{h}_j$ when

[2]Following the convention in machine translation [Brown et al., 1993], we use j to represent a position in the source-side sequence, and use i to represent a position in the target-side sequence.

computing $\mathbf{c}_i$ for $\mathbf{s}_i$. Sometimes $\mathbf{c}_i$ is also called a **context vector**.

A common approach to computing attention weights is to normalize **alignment scores** in the following form

$$\begin{aligned}\alpha_{i,j} &= \mathrm{Softmax}(a(\mathbf{s}_i, \mathbf{h}_j)) \\ &= \frac{\exp(a(\mathbf{s}_i, \mathbf{h}_j))}{\sum_{j'=1}^{m} \exp(a(\mathbf{s}_i, \mathbf{h}_{j'}))}\end{aligned} \tag{5.17}$$

Here the alignment score $a(\mathbf{s}_i, \mathbf{h}_j)$ measures how strong $\mathbf{h}_j$ is related to $\mathbf{s}_i$. In general, $a(\mathbf{s}_i, \mathbf{h}_j)$ can be defined in several different ways [Graves et al., 2014; Bahdanau et al., 2014; Luong et al., 2015]. A comprehensive list of these functions can be found in survey papers on this subject [Chaudhari et al., 2021]. Here we introduce some of the common ones.

- **Dot-product Attention**. One of the simplest methods is to measure the similarity between $\mathbf{h}_j$ and $\mathbf{s}_i$. Thus, we can calculate the dot-product of the two vectors, as follows

$$\begin{aligned}a(\mathbf{s}_i, \mathbf{h}_j) &= \mathbf{s}_i \mathbf{h}_j^{\mathrm{T}} \\ &= \sum_{k=1}^{d_h} s_i(k) \cdot h_j(k)\end{aligned} \tag{5.18}$$

  A variant of this model, called **scaled dot-product attention**, adds a scalar factor $\frac{1}{\beta}$ to the right-hand side of Eq. (5.18), as follows

$$a(\mathbf{s}_i, \mathbf{h}_j) = \frac{\mathbf{s}_i \mathbf{h}_j^{\mathrm{T}}}{\beta} \tag{5.19}$$

  We will see an example of this model later in this section.

- **Cosine Attention**. Another commonly used similarity measure in vector algebra is the cosine of the angle between two vectors, given by

$$\begin{aligned}a(\mathbf{s}_i, \mathbf{h}_j) &= \cos(\mathbf{s}_i, \mathbf{h}_j) \\ &= \frac{\mathbf{s}_i \mathbf{h}_j^{\mathrm{T}}}{\|\mathbf{s}_i\|_2 \cdot \|\mathbf{h}_j\|_2}\end{aligned} \tag{5.20}$$

  where $\|\mathbf{a}\|_2 = (\mathbf{a} \cdot \mathbf{a})^{\frac{1}{2}}$ is the Euclidean norm of the vector $\mathbf{a}$.

- **Weighted Dot-product Attention**. This attention model involves a linear mapping of the input vectors before performing the dot-product operation, given by

$$a(\mathbf{s}_i, \mathbf{h}_j) = \mathbf{s}_i \mathbf{W}_{\mathrm{a}} \mathbf{h}_j^{\mathrm{T}} \tag{5.21}$$

  where $\mathbf{W}_{\mathrm{a}} \in \mathbb{R}^{d_h \times d_h}$ is the parameter matrix of the linear mapping. Both this approach and the dot-product attention approach are also called **multiplicative attention** [Ruder, 2017].

- **Additive Attention**. In additive attention, the entries of the two vectors are summed in some way. A widely-used form is given by Bahdanau et al. [2014]

$$a(\mathbf{s}_i, \mathbf{h}_j) = \mathbf{v}_{\mathrm{a}}^{\mathrm{T}} \mathrm{TanH}(\mathbf{s}_i \mathbf{W}_{\mathrm{s}} + \mathbf{h}_j \mathbf{W}_{\mathrm{h}}) \tag{5.22}$$

where $\mathbf{W}_{\mathrm{h}}, \mathbf{W}_{\mathrm{s}} \in \mathbb{R}^{d_h \times d_a}$ and $\mathbf{v}_{\mathrm{a}} \in \mathbb{R}^{d_a}$ are parameters. $\mathrm{TanH}(\mathbf{s}_i \mathbf{W}_{\mathrm{s}} + \mathbf{h}_j \mathbf{W}_{\mathrm{h}})$ produces a d_a-dimensional vector where each entry is a transformed weighted sum of the entries of $\mathbf{h}_j$ and $\mathbf{s}_i$. It is followed by a dot-product with another weight vector $\mathbf{v}_{\mathrm{a}}$.

Now let us return to Eqs. (5.16-5.17) and rethink the role of attention weights. Eq. (5.17) informally defines a "distribution" over $\mathbf{h}_1 ... \mathbf{h}_m$, written as

$$\Pr(\mathbf{h}_j | \mathbf{s}_i) = \alpha_{i,j} \tag{5.23}$$

If we consider $\mathbf{h}$ a random variable that takes a value from $\{\mathbf{h}_1, ..., \mathbf{h}_m\}$, then $\alpha_{i,j}$ can be thought of as the probability of $\mathbf{h} = \mathbf{h}_j$, conditioned on $\mathbf{s}_i$, and Eq. (5.16) can be rewritten as

$$\begin{aligned} \mathbf{c}_i &= \sum_{j=1}^{m} \Pr(\mathbf{h}_j | \mathbf{s}_i) \cdot \mathbf{h}_j \\ &= \mathbb{E}_{\mathbf{h} \sim \Pr(\mathbf{h}|\mathbf{s}_i)}(\mathbf{h}) \end{aligned} \tag{5.24}$$

In other words, $\mathbf{c}_i$ can be viewed as an **expected representation** of the source-side sequence given the target-side state $\mathbf{s}_i$, that is, the expectation of $\{\mathbf{h}_1, ..., \mathbf{h}_m\}$ under the distribution $\Pr(\mathbf{h}_j | \mathbf{s}_i)$. This provides a general framework for describing the way the decoder receives the information from the encoder: the decoder is a **receiver** that determines how much information is accepted from each **sender**. For example, in the NMT model of the previous section, there is only one sender $\mathbf{h}_m$, and so the receiver receives all the information the sender sends. By contrast, in the NMT model armed with the attention mechanism, there are m senders $\{\mathbf{h}_1, ..., \mathbf{h}_m\}$ and the receiver receives information according to a distribution of preferences for the senders.

It is straightforward to introduce the attention model into the process of word prediction. We modify our treatment of $\mathbf{s}_i$ so as to make use of both the source-side and target-side information at each decoding step. We slightly modify the definition of $\mathbf{s}_i$ to include the context vector corresponding to the previous state $\mathbf{s}_{i-1}$, as follows

$$\mathbf{s}_i = \mathrm{RNN}(\mathbf{s}_{i-1}, \mathbf{c}_{i-1}, \mathbf{y}_i^{\mathrm{e}}) \tag{5.25}$$

Compared with the model of Eq. (5.11), the model of Eq. (5.25) takes $\mathbf{c}_{i-1}$ as an additional input. Therefore, this model considers both the representation of the target-side words $y_1 ... y_{i-1}$ (as encoded in $\mathbf{s}_{i-1}$ and $\mathbf{y}_i^{\mathrm{e}}$) and the representation of the entire source-side sequence $x_1 ... x_m$ (as encoded in $\mathbf{c}_{i-1}$). Then, the distribution of target words at position i can be conditioned on $\mathbf{s}_i$ as usual

$$\Pr(\cdot | y_1 ... y_i, x_1 ... x_m) = \Pr(\cdot | \mathbf{s}_i) \tag{5.26}$$

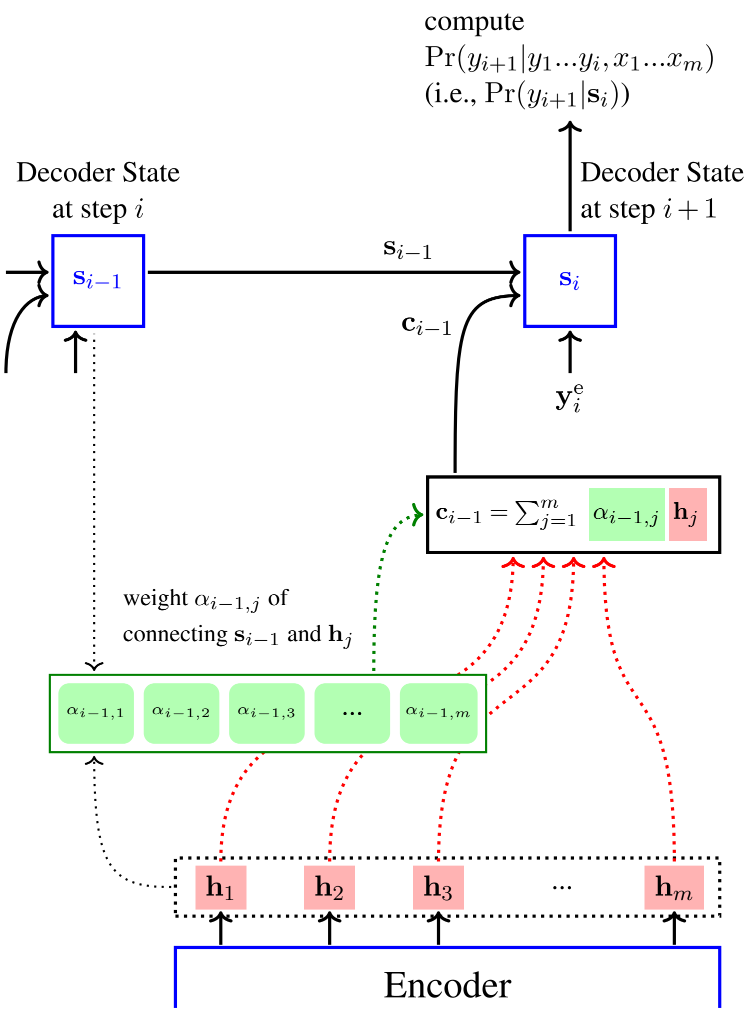


Figure 5.4: An attention model for NMT. Suppose we have obtained the representations $\{\mathbf{h}_1,...,\mathbf{h}_m\}$ and the decoder state $\mathbf{s}_{i-1}$ up to this point. We wish to obtain the decoder state at the next step. To this end, we first compute attention weights by normalizing some attention scores between $\mathbf{s}_{i-1}$ and $\{\mathbf{h}_1,...,\mathbf{h}_m\}$, and then compute a context vector $\mathbf{c}_{i-1}$ by summing over $\{\mathbf{h}_1,...,\mathbf{h}_m\}$ with the attention weights. A new decoder state $\mathbf{s}_i$ is created by taking the context vector $\mathbf{c}_{i-1}$, the previous state $\mathbf{s}_{i-1}$, and the word representation $\mathbf{y}^{\mathrm{e}}_i$. $\mathbf{s}_i$ will be used as a condition for predicting a distribution of words at step $i+1$.

where $\Pr(\cdot|\mathbf{s}_i)$ is generally a Softmax layer. This process is illustrated in Figure 5.4.

We now have a model for computing $\Pr(y_{i+1}|y_1...y_i, x_1...x_m)$. A brief outline of the key steps of this model is given by

1. Encode the source-side sequence as $\mathbf{h}_1...\mathbf{h}_m$ where $\mathbf{h}_j = \mathrm{RNN}(\mathbf{h}_{j-1}, \mathbf{x}^{\mathrm{e}}_j)$.
2. Repeat the following procedure from $i = 1$ to $n-1$.

a. Compute the alignment score $a(\mathbf{s}_{i-1}, \mathbf{h}_j)$ for each j.

b. Compute the attention weights $\{\alpha_{i-1,1}, ..., \alpha_{i-1,m}\}$ where $\alpha_{i-1,j} = \frac{\exp(a(\mathbf{s}_{i-1}, \mathbf{h}_j))}{\sum_{j'=1}^{m} \exp(a(\mathbf{s}_{i-1}, \mathbf{h}_{j'}))}$.

c. Compute the context vector $\mathbf{c}_{i-1} = \sum_{j=1}^{m} \alpha_{i-1,j} \cdot \mathbf{h}_j$.

d. Compute the target-side state $\mathbf{s}_i = \mathrm{RNN}(\mathbf{s}_{i-1}, \mathbf{c}_{i-1}, \mathbf{y}_i^{\mathrm{e}})$.

e. Compute the distribution of target-side words $\Pr(\cdot|\mathbf{s}_i)$.

f. Compute $\Pr(y_{i+1}|y_1...y_i, x_1...x_m) = \Pr(y_{i+1}|\mathbf{s}_i)$ for a given word y_{i+1} (as in training), or select the most likely word $\hat{y}_{i+1} = \arg\max_{y_{i+1}} \Pr(y_{i+1}|y_1...y_i, x_1...x_m)$ (as in testing).

In real-world systems, this basic model can be modified to better predict the target-side words. For example, we can introduce fusion layers to combine $\mathbf{s}_i$, $\mathbf{c}_{i-1}$, and $\mathbf{y}_i^{\mathrm{e}}$ before the Softmax layer so that we have a deeper model for prediction [Bahdanau et al., 2014]. Another commonly used approach is to stack multiple RNN layers on the target-side. In this case, one can perform attention in either each layer of the stack [Wu et al., 2016] or the top-most layer of the stack [Luong et al., 2015]. See Section 5.3.5 for more information about multi-layer approaches to attention.

5.3.2 The QKV Attention

Because the attention mechanism is such a powerful approach, many variants have been developed. Perhaps the most widely used approach is to reframe the attention problem as one of matching a query in a set of key-value pairs. It lays the foundation for the well-known sequence model — Transformer [Vaswani et al., 2017].

Here we assume that there are a number of key-value pairs $\{(\mathbf{k}_1, \mathbf{v}_1), ..., (\mathbf{k}_m, \mathbf{v}_m)\}$ and a query $\mathbf{q}$. The goal of the **query-key-value attention** (or **QKV attention**) model is to obtain a value by considering the correspondence between the query and the keys. This is a standard searching problem in database systems in which information is returned in its original form or a new form when it matches the query. In the QKV attention, the result of searching is not a single value in $\{\mathbf{v}_1, ..., \mathbf{v}_m\}$ but instead a combination of these values. This is the key difference of this attention model compared with the conventional models of searching.

Formally, the result of the QKV attention is defined to be

$$\mathbf{c} = \sum_{j=1}^{m} \alpha_j \mathbf{v}_j \tag{5.27}$$

where

$$\alpha_j = \mathrm{Softmax}(\frac{\mathbf{q}\mathbf{k}_j^{\mathrm{T}}}{\beta}) \tag{5.28}$$

is the attention weight. It turns out that the above model has precisely the same general form as the model described in the previous subsection, and $\mathbf{c}$ can be simply viewed as a context vector.

While the basic form of the QKV attention is not something "new", it can handle a variety of problems by giving $\mathbf{q}$, $\mathbf{k}_j$ and $\mathbf{v}_j$ appropriate meanings. Here we consider a more general case where there are n queries $\{\mathbf{q}_1, ..., \mathbf{q}_n\}$ and n output vectors $\{\mathbf{c}_1, ..., \mathbf{c}_n\}$. To simplify notation, we use $\mathbf{Q}$ to denote a matrix where the i-th row vector is $\mathbf{q}_i$, like this

$$\mathbf{Q} = \begin{bmatrix} \mathbf{q}_1 \\ \vdots \\ \mathbf{q}_n \end{bmatrix} \tag{5.29}$$

Likewise, we can define $\mathbf{K} = \begin{bmatrix} \mathbf{k}_1 \\ \vdots \\ \mathbf{k}_m \end{bmatrix}$, $\mathbf{V} = \begin{bmatrix} \mathbf{v}_1 \\ \vdots \\ \mathbf{v}_m \end{bmatrix}$, and $\mathbf{C} = \begin{bmatrix} \mathbf{c}_1 \\ \vdots \\ \mathbf{c}_n \end{bmatrix}$. Then, the attention model can be formulated as

$$\mathbf{C} = \mathrm{Softmax}(\frac{\mathbf{Q}\mathbf{K}^{\mathrm{T}}}{\beta})\mathbf{V} \tag{5.30}$$

Figure 5.5 shows an illustration of this equation. Note that $\mathrm{Softmax}(\frac{\mathbf{Q}\mathbf{K}^{\mathrm{T}}}{\beta})$ computes a matrix of attention weights

$$\mathrm{Softmax}(\frac{\mathbf{Q}\mathbf{K}^{\mathrm{T}}}{\beta}) = \begin{bmatrix} \alpha_{1,1} & ... & \alpha_{1,m} \\ \vdots & & \vdots \\ \alpha_{n,1} & ... & \alpha_{n,m} \end{bmatrix} \tag{5.31}$$

where a row vector $\begin{bmatrix} \alpha_{i,1} & ... & \alpha_{i,m} \end{bmatrix}$ represents a distribution over $\{\mathbf{v}_1, ..., \mathbf{v}_m\}$. We can then expand Eq. (5.30) for easy understanding of the model

$$\begin{aligned} \mathbf{C} &= \begin{bmatrix} \mathbf{c}_1 \\ \vdots \\ \mathbf{c}_n \end{bmatrix} \\ &= \begin{bmatrix} \sum_{j=1}^{m} \alpha_{1,j}\mathbf{v}_j \\ \vdots \\ \sum_{j=1}^{m} \alpha_{n,j}\mathbf{v}_j \end{bmatrix} \\ &= \begin{bmatrix} \alpha_{1,1} & ... & \alpha_{1,m} \\ \vdots & & \vdots \\ \alpha_{n,1} & ... & \alpha_{n,m} \end{bmatrix} \begin{bmatrix} \mathbf{v}_1 \\ \vdots \\ \mathbf{v}_m \end{bmatrix} \end{aligned} \tag{5.32}$$

In sequence-to-sequence modeling, $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$ can be defined in several different ways. To describe the correspondence between the source-side and target-side sequences, one

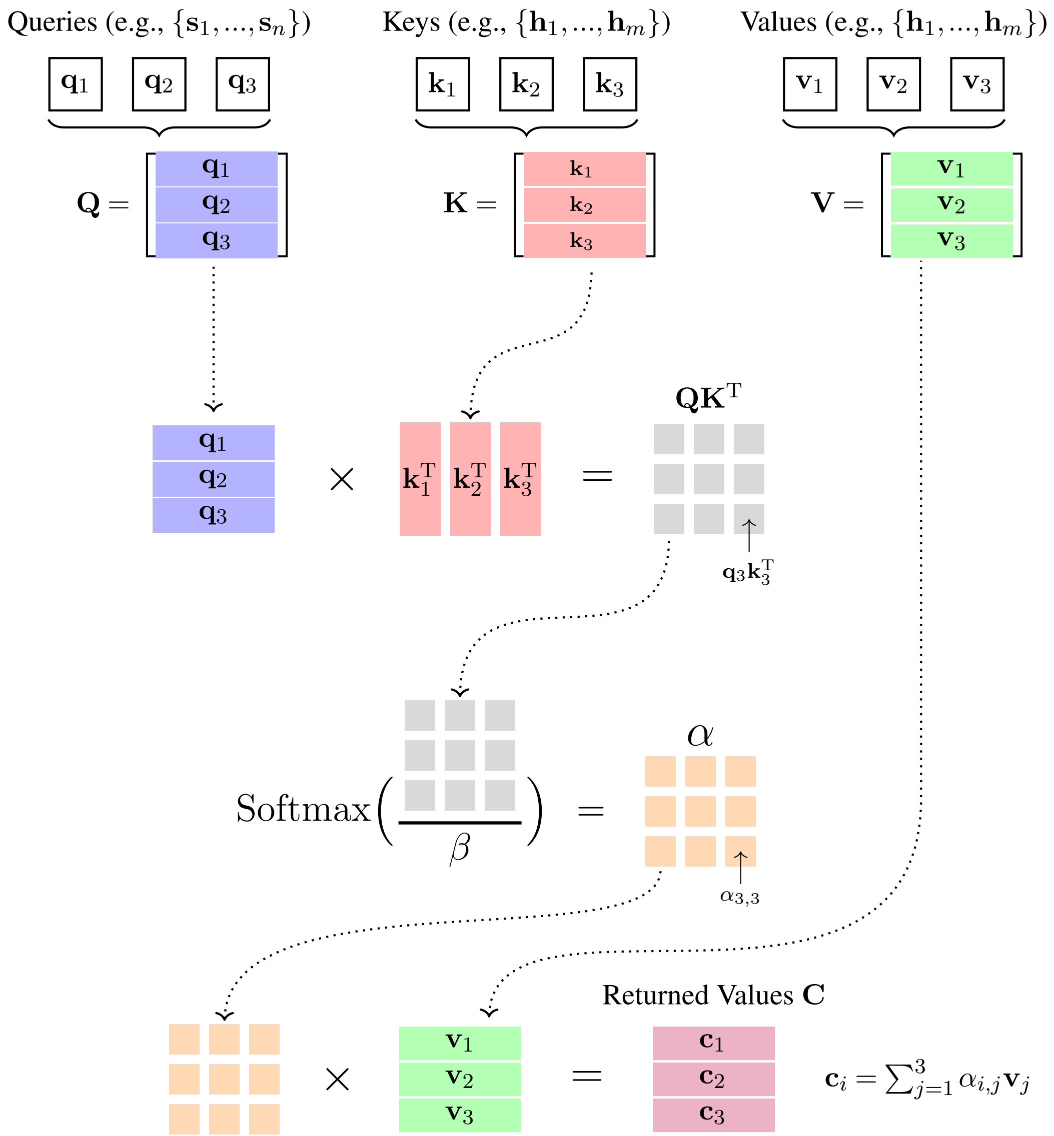


Figure 5.5: The QKV attention model for batches of queries ($\mathbf{Q}$), keys ($\mathbf{K}$), and values ($\mathbf{V}$). The figure shows a direct implementation of the formula $\mathbf{C} = \mathrm{Softmax}(\frac{\mathbf{QK}^{\mathrm{T}}}{\beta})\mathbf{V}$. $\mathrm{Softmax}(\frac{\mathbf{QK}^{\mathrm{T}}}{\beta})$ computes the attention weights by normalizing a scaled dot-product of $\mathbf{Q}$ and $\mathbf{K}^{\mathrm{T}}$. This results in a matrix α in which a row vector describes weights of different values. By multiplying α with $\mathbf{V}$, we obtain a sequence of new values, each expressing a weighted sum of the original values.

approach, called **encoder-decoder attention**, is to simply assume that

$$\mathbf{Q} = \begin{bmatrix} \mathbf{s}_1 \\ \vdots \\ \mathbf{s}_n \end{bmatrix} \tag{5.33}$$

and

$$\mathbf{K} = \mathbf{V} = \begin{bmatrix} \mathbf{h}_1 \\ \vdots \\ \mathbf{h}_m \end{bmatrix} \tag{5.34}$$

In this case, $\mathbf{C}$ is a sequence of new representations of the source-side sequence given the representations of the target-side sequence. As with the model described in the previous subsection, each $\mathbf{c}_i \in \mathbf{C}$ can be used to predict the word y_{i+1}.

In addition to applying the model to sequence-to-sequence problems, another type of approach is to regard it as a sequence model, that is, we use the QKV attention to represent a sequence in one language. In this case, the QKV attention is also called **self-attention** which forms the basis of the well-known Transformer model [Vaswani et al., 2017]. Consider, for example, the sequence of states $\mathbf{h}_1 ... \mathbf{h}_m$. The self-attention model assumes that

$$\mathbf{Q} = \mathbf{K} = \mathbf{V} = \begin{bmatrix} \mathbf{h}_1 \\ \vdots \\ \mathbf{h}_m \end{bmatrix} \tag{5.35}$$

Then, the output of the model is a sequence of representations $\mathbf{c}_1 ... \mathbf{c}_m$. $\mathbf{c}_j$ is a representation which considers the correlations between $\mathbf{h}_j$ and any other element of the input sequence. We will see a more detailed discussion on this model in Chapter 6.

5.3.3 Multi-head Attention

Multi-head attention is an interesting extension to the above models. The key idea is to perform attention in different sub-spaces of representations simultaneously rather than in a single space of representations. To illustrate, consider a standard attention model that takes sequences of source-side and target-side states and outputs a sequence of new states, written as

$$\mathbf{c}_1 ... \mathbf{c}_n = \mathrm{Att}(\mathbf{h}_1 ... \mathbf{h}_m, \mathbf{s}_1 ... \mathbf{s}_n) \tag{5.36}$$

where $\mathbf{h}_j, \mathbf{s}_i, \mathbf{c}_i \in \mathbb{R}^{d_h}$, and $\mathrm{Att}(\cdot)$ is the attention function. We can map $\mathbf{h}_j$ into τ vectors $\{\mathbf{h}_j^{[1]}, ..., \mathbf{h}_j^{[\tau]}\}$ via the following linear transformations

$$\mathbf{h}_j^{[1]} = \mathbf{h}_j \mathbf{W}_h^{[1]} \tag{5.37}$$

$$\vdots$$

$$\mathbf{h}_j^{[\tau]} = \mathbf{h}_j \mathbf{W}_h^{[\tau]} \tag{5.38}$$

where $\mathbf{h}_j^{[1]}, ..., \mathbf{h}_j^{[\tau]} \in \mathbb{R}^{\frac{d_h}{\tau}}$, and $\mathbf{W}_h^{[1]}, ..., \mathbf{W}_h^{[\tau]} \in \mathbb{R}^{d_h \times \frac{d_h}{\tau}}$.

Similarly, we can map $\mathbf{s}_i$ into τ vectors $\{\mathbf{s}_i^{[1]}, ..., \mathbf{s}_i^{[\tau]}\}$. We then define τ **feature sub-spaces** in which the attention function is performed independently. For the k-th feature sub-space, we have

$$\mathbf{c}_1^{[k]}...\mathbf{c}_n^{[k]} = \mathrm{Att}(\mathbf{h}_1^{[k]}...\mathbf{h}_m^{[k]}, \mathbf{s}_1^{[k]}...\mathbf{s}_n^{[k]}) \tag{5.39}$$

The output of the model is a sequence of d_h-dimensional vectors, each of which is obtained by concatenating the vectors that are produced in all these feature sub-spaces, followed by a linear transformation. This procedure is given by

$$\mathbf{c}_1 = [\mathbf{c}_1^{[1]}, ..., \mathbf{c}_1^{[\tau]}]\mathbf{W}_c \tag{5.40}$$
$$...$$
$$\mathbf{c}_n = [\mathbf{c}_n^{[1]}, ..., \mathbf{c}_n^{[\tau]}]\mathbf{W}_c \tag{5.41}$$

where $\mathbf{W}_c \in \mathbb{R}^{d_h \times d_h}$.

Following the notation used in the previous subsection, we can express a sequence of vectors as a matrix, say, $\mathbf{H} = \begin{bmatrix} \mathbf{h}_1 \\ \vdots \\ \mathbf{h}_m \end{bmatrix} \in \mathbb{R}^{m \times d_h}$, $\mathbf{S} = \begin{bmatrix} \mathbf{s}_1 \\ \vdots \\ \mathbf{s}_n \end{bmatrix} \in \mathbb{R}^{n \times d_h}$, and $\mathbf{C} = \begin{bmatrix} \mathbf{c}_1 \\ \vdots \\ \mathbf{c}_n \end{bmatrix} \in \mathbb{R}^{n \times d_h}$. Using this notation, we rewrite Eq. (5.36) as

$$\mathbf{C} = \mathrm{Att}(\mathbf{H}, \mathbf{S}) \tag{5.42}$$

To give a formal definition of multi-head attention, we first introduce the split and merge functions. The split function divides each row vector of a matrix into a number of sub-vectors, resulting in a 3D tensor. For example, splitting a $m \times d_h$ matrix $\mathbf{A}$ with τ produces a $\tau \times m \times \frac{d_h}{\tau}$ tensor[3]

$$\mathbf{A}_{\text{heads}} = \mathrm{Split}(\mathbf{A}, \tau) \tag{5.43}$$

The merge function has a reverse form of the split function. Given a $\tau \times n \times \frac{d_h}{\tau}$ tensor (say $\mathbf{A}_{\text{heads}}$), it merges each group of τ $\frac{d_h}{\tau}$-dimensional sub-arrays in the form

$$\mathbf{A}_{\text{merge}} = \mathrm{Merge}(\mathbf{A}_{\text{heads}}, \tau) \tag{5.44}$$

Thus the form of multi-head attention is given by

$$\begin{aligned} \mathbf{C} &= \mathbf{C}_{\text{merge}}\mathbf{W}_c \\ &= \mathrm{Merge}(\mathbf{C}_{\text{heads}}, \tau)\mathbf{W}_c \\ &= \mathrm{Merge}(\mathrm{Att}(\mathbf{H}_{\text{heads}}, \mathbf{S}_{\text{heads}}), \tau)\mathbf{W}_c \end{aligned} \tag{5.45}$$
$$\mathbf{H}_{\text{heads}} = \mathrm{Split}(\mathbf{H}\mathbf{W}_h, \tau) \tag{5.46}$$
$$\mathbf{S}_{\text{heads}} = \mathrm{Split}(\mathbf{S}\mathbf{W}_s, \tau) \tag{5.47}$$

[3] A $a \times b \times c$ tensor can be treated as an array of a matrices whose shapes are $b \times c$.

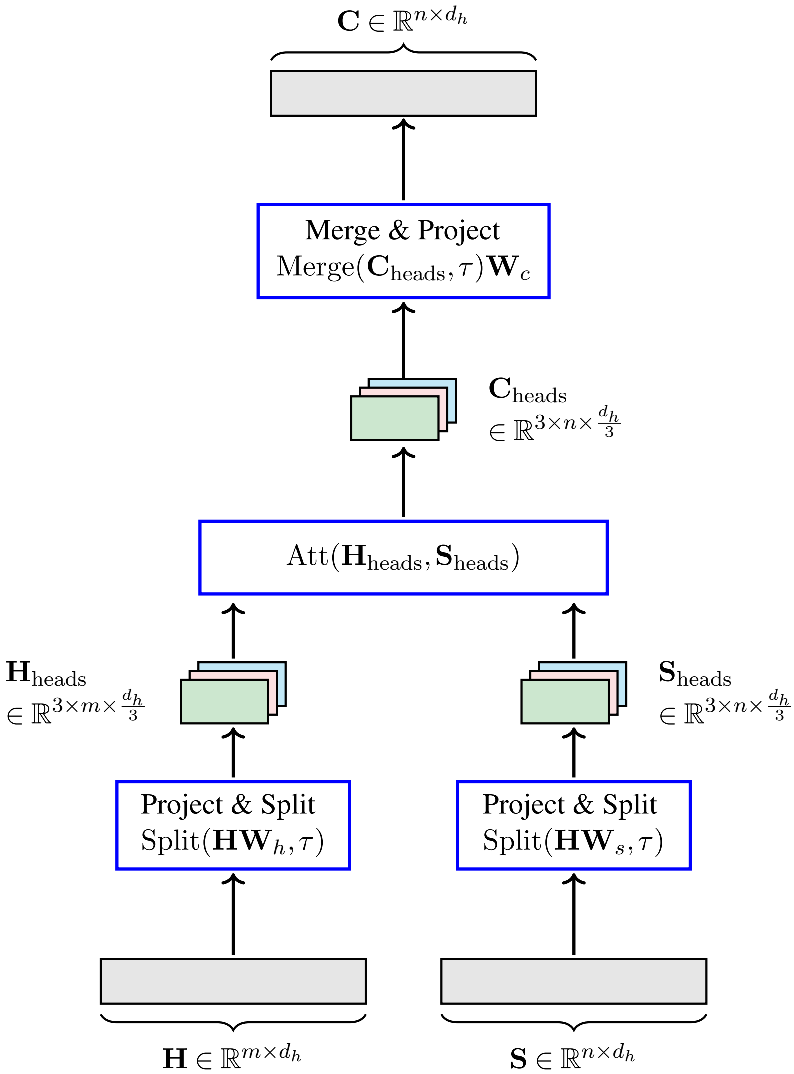


Figure 5.6: An attention model with $\tau = 3$ heads. First, we transform the input matrices into multi-head representations, i.e., 3D tensors $\mathbf{H}_{\mathrm{heads}} \in \mathbb{R}^{3 \times m \times \frac{d_h}{3}}$ and $\mathbf{S}_{\mathrm{heads}} \in \mathbb{R}^{3 \times n \times \frac{d_h}{3}}$. These tensors are then taken by an attention model. The output of this model is a tensor $\mathbf{C}_{\mathrm{heads}} \in \mathbb{R}^{3 \times n \times \frac{d_h}{3}}$. We then merge the heads of $\mathbf{C}_{\mathrm{heads}}$, followed by a linear transformation. Finally, we obtain n vectors of size d_h, represented by an $n \times d_h$ matrix.

where $\mathbf{W}_h, \mathbf{W}_s \in \mathbb{R}^{d_h \times d_h}$ are the parameters. $\mathrm{Split}(\mathbf{HW}_h, \tau)$ implements the projections of Eqs. (5.37-5.38) for all $\mathbf{h}_j$. Likewise, we can have the meaning of $\mathrm{Split}(\mathbf{HW}_h, \tau)$. Note that here $\mathrm{Att}(\cdot)$ is extended to deal with multi-head inputs. See Figure 5.6 for an illustration of this model.

Multi-head attention is a very general approach that can be extended to many models. As a simple example of this extension, consider the QKV attention model discussed in the previous subsection. Let $\mathrm{Att}_{\mathrm{QKV}}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$ be the attention function, and $\mathbf{Q} \in \mathbb{R}^{d_k}, \mathbf{K} \in \mathbb{R}^{d_k}, \mathbf{V} \in \mathbb{R}^{d_v}$

be the inputs. The multi-head QKV attention model is given by

$$\mathbf{C} = \mathrm{Merge}(\mathrm{Att}_{\mathrm{QKV}}(\mathbf{Q}_{\mathrm{heads}}, \mathbf{K}_{\mathrm{heads}}, \mathbf{V}_{\mathrm{heads}}))\mathbf{W}_c \tag{5.48}$$

$$\mathbf{Q}_{\mathrm{heads}} = \mathrm{Split}(\mathbf{Q}\mathbf{W}_q, \tau) \tag{5.49}$$

$$\mathbf{K}_{\mathrm{heads}} = \mathrm{Split}(\mathbf{K}\mathbf{W}_k, \tau) \tag{5.50}$$

$$\mathbf{V}_{\mathrm{heads}} = \mathrm{Split}(\mathbf{V}\mathbf{W}_v, \tau) \tag{5.51}$$

where $\mathbf{W}_q \in \mathbb{R}^{d_k \times d_k}, \mathbf{W}_k \in \mathbb{R}^{d_k \times d_k}, \mathbf{W}_v \in \mathbb{R}^{d_v \times d_v}, \mathbf{W}_c \in \mathbb{R}^{d_v \times d_v}$ are the model parameters.

One advantage of multi-head attention is that the feature sub-spaces will each describe a different perspective of attention (call it an **attention head** or **head** for short). Therefore, the concatenation of the outputs over these heads represents an ensemble of attention models that deal with different parts of the data. This is similar to learning a group of models independently and combining them to form a stronger model. This type of machine learning approach has been proven to be useful in many problems [Opitz and Maclin, 1999; Zhou, 2012]. Note that the multi-head attention models discussed here are parameterized by the linear projections on the input and output spaces. The use of these linear projections is generally helpful as the models become deeper and can describe more complex problems.

From an architecture design perspective, multi-head attention falls into a broad class of neural networks — those involving a number of branches of layer stacks for dealing with the same input (call them **multi-branch neural networks**). However, unlike conventional approaches, which require different model architectures for different branches, the multi-head attention approach is based on a single model for all the heads. As a result, such systems are very efficient in practice because the attention procedure can run in parallel over these heads.

5.3.4 Hierarchical Attention

In many cases the underlying structure of an NLP problem is hierarchical. For example, documents may have a multi-level structure: a document is made up of sentences, a sentence is made up of words, and a word is made up of characters. It is therefore desirable to modify the attention models to take into account the hierarchical nature of this data [Yang et al., 2016].

To illustrate, we consider a simple problem where the source-side has a 2-level tree structure. Suppose the source-side sequence is a concatenation of a number of sub-sequences $\{\bar{\mathbf{u}}_1, ..., \bar{\mathbf{u}}_T\}$. Each $\bar{\mathbf{u}}_t$ yields a sequence of words

$$\bar{\mathbf{u}}_t = x_{p(t,1)} ... x_{p(t,|\bar{\mathbf{u}}_t|)} \tag{5.52}$$

where $p(t,i)$ is the position of the i-th word of $\bar{\mathbf{u}}_t$ in the entire source-side sequence $x_1...x_m$. Then, the sequence $x_1...x_m$ can be written as a composition of T sub-sequences:

$$x_1...x_m = \underbrace{x_{p(1,1)} ... x_{p(1,|\bar{\mathbf{u}}_1|)}}_{\bar{\mathbf{u}}_1} \ \underbrace{x_{p(2,1)} ... x_{p(2,|\bar{\mathbf{u}}_2|)}}_{\bar{\mathbf{u}}_2} \ \cdots \ \underbrace{x_{p(T,1)} ... x_{p(T,|\bar{\mathbf{u}}_T|)}}_{\bar{\mathbf{u}}_T} \tag{5.53}$$

Similarly, the encoder output $\mathbf{h}_1...\mathbf{h}_m$ can be written as

$$\mathbf{h}_1...\mathbf{h}_m = \mathbf{h}_{p(1,1)}...\mathbf{h}_{p(1,|\bar{\mathbf{u}}_1|)}\ \mathbf{h}_{p(2,1)}...\mathbf{h}_{p(2,|\bar{\mathbf{u}}_2|)}\ \cdots\ \mathbf{h}_{p(T,1)}...\mathbf{h}_{p(T,|\bar{\mathbf{u}}_T|)} \tag{5.54}$$

On the target-side, we assume that there are two sequences of state vectors: one for placing the standard representations of the target-side sequence (i.e., $\mathbf{s}_1...\mathbf{s}_n$) and one for placing higher-level representations of $\mathbf{s}_1...\mathbf{s}_n$. Let $\phi(i)$ denote the position in the higher-level sequence of $\mathbf{s}_i$, and $\bar{s}_{\phi(i)}$ denote the corresponding state vector. For each i, we thus have a pair of state vectors $\mathbf{s}_i$ and $\bar{\mathbf{s}}_{\phi(i)}$. In general, the relationship between $\mathbf{s}_i$ and $\bar{\mathbf{s}}_{\phi(i)}$ comes from the hierarchical structure of the problem. For example, $\mathbf{s}_i$ is the representation of a word, and $\bar{\mathbf{s}}_{\phi(i)}$ is the representation of the sentence the word belongs to[4].

As before, our goal is to obtain a context vector $\mathbf{c}_i$ for each target-side position i. Here we still take $\mathbf{c}_i$ to be a weighted sum of $\{\mathbf{h}_1, ..., \mathbf{h}_m\}$, as in Eq. (5.16). All that remains is to specify the attention weight for each $\mathbf{h}_j$. As a first step we attend $\mathbf{s}_i$ to each $\mathbf{u}_t$. This is a standard procedure. We just need to run the attention model on $\mathbf{h}_{p(t,1)}...\mathbf{h}_{p(t,|\bar{\mathbf{u}}_t|)}$ instead of $\mathbf{h}_1...\mathbf{h}_m$, given by

$$\begin{aligned}
\bar{\mathbf{h}}_t &= \mathrm{Att}(\mathbf{h}_{p(t,1)}...\mathbf{h}_{p(t,|\bar{\mathbf{u}}_t|)}, \mathbf{s}_i) \\
&= \sum_{k=1}^{|\bar{\mathbf{u}}_t|} \pi_{i,k,t}\mathbf{h}_{p(t,k)}
\end{aligned} \tag{5.55}$$

where $\pi_{i,k,t}$ is the attention weight restricted to $\mathbf{u}_t$. $\bar{\mathbf{h}}_t$ is a representation of $\mathbf{u}_t$, and so we have a new sequence of representations $\bar{\mathbf{h}}_1...\bar{\mathbf{h}}_T$.

Then, we run the attention model on $\bar{\mathbf{h}}_1...\bar{\mathbf{h}}_T$ to perform a second round of attention. This is done by attending $\mathbf{s}_{\phi(i)}$ to $\bar{\mathbf{h}}_1...\bar{\mathbf{h}}_T$. The output is a context vector for the hierarchical attention model, given by

$$\begin{aligned}
\mathbf{c}_i &= \mathrm{Att}(\bar{\mathbf{h}}_1...\bar{\mathbf{h}}_T, \mathbf{s}_{\phi(i)}) \\
&= \sum_{t=1}^{T} \gamma_{i,t}\bar{\mathbf{h}}_t
\end{aligned} \tag{5.56}$$

where $\gamma_{i,t}$ is the weight of attending $\mathbf{s}_{\phi(i)}$ to $\bar{\mathbf{h}}_t$. Substituting Eq. (5.55) into Eq. (5.56), we can write $\mathbf{c}_i$ as

$$\begin{aligned}
\mathbf{c}_i &= \sum_{t=1}^{T}\sum_{k=1}^{|\bar{\mathbf{u}}_t|} \gamma_{i,t}\pi_{i,k,t}\mathbf{h}_{p(t,k)} \\
&= \sum_{j=1}^{m} \alpha_{i,j}\mathbf{h}_j
\end{aligned} \tag{5.57}$$

While the notation in this subsection is a bit complicated, the form of the resulting model

[4] If the a-th sentence covers words from position b to c, then $\phi(b) = \phi(b+1) = ... = \phi(c) = a$.

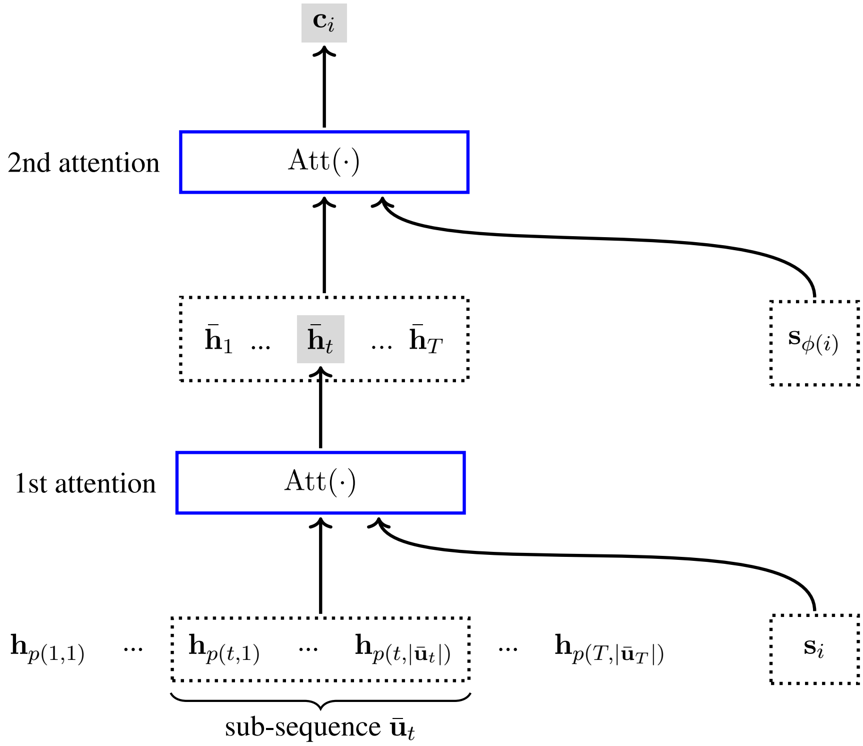


Figure 5.7: A 2-level hierarchical attention model. The input sequence $\mathbf{h}_1...\mathbf{h}_m$ is made up of T sub-sequences. For each sub-sequence $\bar{\mathbf{u}}_t$, an attention model is used to produce a context vector $\bar{\mathbf{h}}_t$ by considering the target-side state (i.e., $\mathbf{s}_i$) and the representations of the sub-sequence (i.e., $\mathbf{h}_{p(t,1)}...\mathbf{h}_{p(t,|\bar{\mathbf{u}}_t|)}$). The result of running this procedure on the T sub-sequences is T level-1 representations $\bar{\mathbf{h}}_1...\bar{\mathbf{h}}_T$. They are then taken by a second attention model to consider the attention between these representations and a higher-level target-side state $\mathbf{s}_{\phi(i)}$. This results in the context vector $\mathbf{c}_i$ which describes the attention between the target-side state $\mathbf{s}_i$ and the entire source-side sequence $\mathbf{h}_1...\mathbf{h}_m$.

is simple. We still combine $\{\mathbf{h}_1, ..., \mathbf{h}_m\}$ in a linear manner but with new weights [Maruf et al., 2019]. Computing $\alpha_{i,j}$ describes a generative process in which we first determine the weight of each sub-sequence and then determine the weight of each word in a sub-sequence, as illustrated in Figure 5.7. See below for an alignment among different types of attention weight.

sequence	$\mathbf{h}_1$	...	$\mathbf{h}_{\lvert\mathbf{u}_1\rvert}$	$\mathbf{h}_{\lvert\mathbf{u}_1\rvert+1}$	...	$\mathbf{h}_{\lvert\mathbf{u}_1\rvert+\lvert\mathbf{u}_2\rvert}$	......	$\mathbf{h}_{\sum_{t=1}^{T-1}\lvert\mathbf{u}_t\rvert+1}$	...	$\mathbf{h}_m$
weight (α)	$\alpha_{i,1}$	...	$\alpha_{i,\lvert\mathbf{u}_1\rvert}$	$\alpha_{i,\lvert\mathbf{u}_1\rvert+1}$	...	$\alpha_{i,\lvert\mathbf{u}_1\rvert+\lvert\mathbf{u}_2\rvert}$	......	$\alpha_{i,\sum_{t=1}^{T-1}\lvert\mathbf{u}_t\rvert+1}$	...	$\alpha_{i,m}$
sequence	$\mathbf{h}_{p(1,1)}$	...	$\mathbf{h}_{p(1,\lvert\bar{\mathbf{u}}_1\rvert)}$	$\mathbf{h}_{p(2,1)}$	...	$\mathbf{h}_{p(2,\lvert\bar{\mathbf{u}}_2\rvert)}$	......	$\mathbf{h}_{p(T,1)}$	...	$\mathbf{h}_{p(T,\lvert\bar{\mathbf{u}}_T\rvert)}$
weight (γ)	$\gamma_{i,1}$	...	$\gamma_{i,1}$	$\gamma_{i,2}$	...	$\gamma_{i,2}$	......	$\gamma_{i,T}$	...	$\gamma_{i,T}$
weight (π)	$\pi_{i,1,1}$	...	$\pi_{i,\lvert\bar{\mathbf{u}}_1\rvert,1}$	$\pi_{i,1,2}$	...	$\pi_{i,\lvert\bar{\mathbf{u}}_2\rvert,2}$	......	$\pi_{i,1,T}$	...	$\pi_{i,\lvert\bar{\mathbf{u}}_T\rvert,T}$

5.3.5 Multi-layer Attention

So far we have considered the case of **single-layer attention** — the output of the attention models is written as a linear combination of the source-side representations. Now we extend it in a natural way to **multi-layer attention** in which the single-layer attention procedure runs a number of times for forming a "deeper" attention model.

To do this, a multi-layer neural network is created on the target-side. The model architecture is regular. We stack a number of attention layers, each interacting with the source-side sequence and feeding its output to the next layer. In an attention layer, we perform attention as usual. For the l-th layer in the stack, this step takes the source-side sequence (denoted by $\mathbf{h}_1...\mathbf{h}_m$) as well as the output of the previous layer (denoted by $\mathbf{s}_1^{l-1}...\mathbf{s}_n^{l-1}$), and produces a sequence of vectors by

$$\mathbf{c}_1^l...\mathbf{c}_n^l = \mathrm{Att}(\mathbf{h}_1...\mathbf{h}_m, \mathbf{s}_1^{l-1}...\mathbf{s}_n^{l-1}) \tag{5.58}$$

where $\mathrm{Att}(\cdot)$ could be any attention function described in this chapter.

Then, we create another neural network $f(\cdot)$ to give more modeling power to the model. The output of the attention layer is thus defined to be

$$\mathbf{s}_1^l...\mathbf{s}_n^l = f(\mathbf{c}_1^l...\mathbf{c}_n^l, \mathbf{s}_1^{l-1}...\mathbf{s}_n^{l-1}) \tag{5.59}$$

$f(\cdot)$ can be designed in many ways [Sukhbaatar et al., 2015; Wu et al., 2016; Vaswani et al., 2017]. A popular choice is to define $f(\cdot)$ as a feed-forward neural network with a residual connection, given by

$$f(\mathbf{c}_1^l...\mathbf{c}_n^l, \mathbf{s}_1^{l-1}...\mathbf{s}_n^{l-1}) = \mathrm{FFN}(\mathbf{c}_1^l...\mathbf{c}_n^l) + \mathbf{s}_1^{l-1}...\mathbf{s}_n^{l-1} \tag{5.60}$$

Substituting for the vectors $\mathbf{c}_1^l...\mathbf{c}_n^l$, using Eq. (5.58), the output of layer i is written in the form

$$\mathbf{s}_1^l...\mathbf{s}_n^l = \mathrm{FFN}(\mathrm{Att}(\mathbf{h}_1...\mathbf{h}_m, \mathbf{s}_1^{l-1}...\mathbf{s}_n^{l-1})) + \mathbf{s}_1^{l-1}...\mathbf{s}_n^{l-1} \tag{5.61}$$

As with the models in the previous subsections, it is convenient to use a more compact notation by expressing a sequence of vectors as a matrix. Thus this model can be given in another form

$$\mathbf{S}^l = \mathrm{FFN}(\mathrm{Att}(\mathbf{H}, \mathbf{S}^{l-1})) + \mathbf{S}^{l-1} \tag{5.62}$$

Here $\mathrm{FFN}(\cdot)$ is generally a multi-layer neural network with non-linear activation functions. The identity mapping (i.e., $+\mathbf{S}^{l-1}$) creates a direct path from the input to the output of the layer, making it easier to train a deep neural network.

Figure 5.8 shows the architecture of this model. The attention model starts with the initial

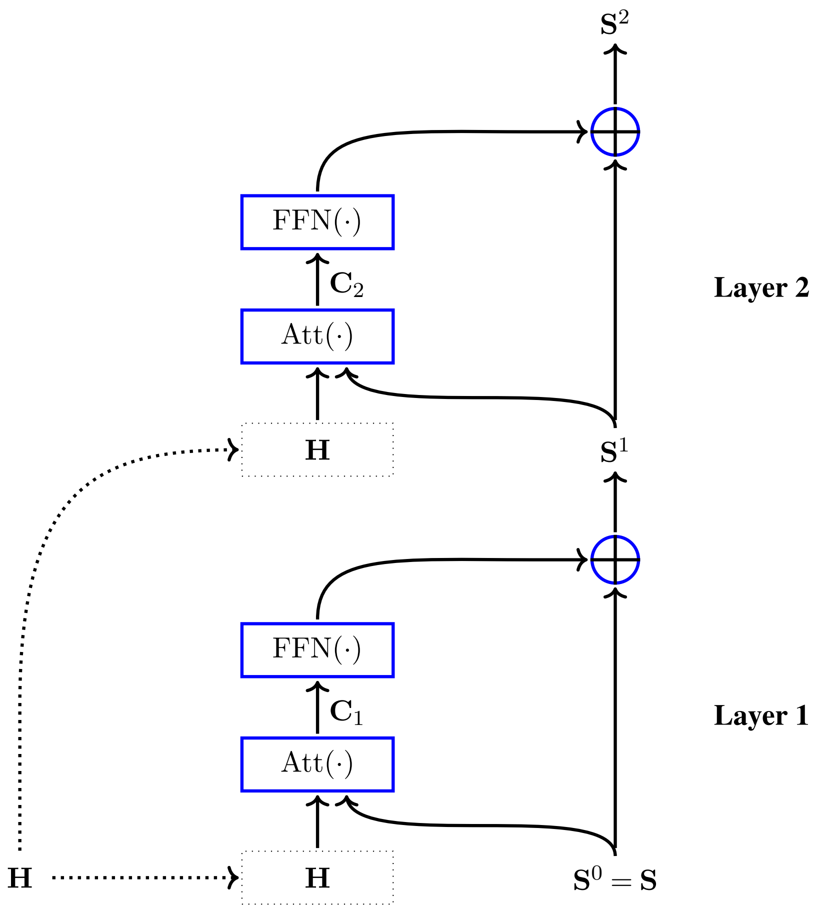


Figure 5.8: A 2-layer attention model. These layers take the same "key-value" pairs (i.e., $\mathbf{H}$) but each takes a different "query" (i.e., $\mathbf{S}^l$). The attention model is standard: context vectors $\mathbf{C}^l$ are generated by taking both $\mathbf{H}$ and $\mathbf{S}^l$. A feed-forward neural network is built to transform $\mathbf{C}^l$, followed by an addition of $\mathbf{S}^l$. So this model can be formulated as $\mathbf{S}^l = \mathrm{FFN}(\mathrm{Att}(\mathbf{H}, \mathbf{S}^{l-1})) + \mathbf{S}^{l-1}$. $\mathbf{S}^l$ is then used in the next layer as the query, that is, layer $l+1$ takes $\mathbf{H}$ and $\mathbf{S}^l$, and repeats the above process. The output of the last layer can be viewed as a deeper representation of $\mathbf{H}$ for $\mathbf{S}$.

representation of the target-side sequence, that is, $\mathbf{S}^0 = \mathbf{S} = \begin{bmatrix} \mathbf{s}_1 \\ \vdots \\ \mathbf{s}_n \end{bmatrix}$. If there are L attention layers, then the final output will be $\mathbf{S}^L$.

5.3.6 Remarks

Above we considered a basic attention model and a series of refinements. The literature on attention and related topics contains a wide range of methods for modeling how a system concentrates on different parts of the input, as well as a wide range of applications of such models. This subsection provides discussions on some of the interesting issues.

1. Alignment vs Attention

Attention is related to a long line of research on alignment approaches to modeling the correspondence between two groups of language units. In NLP, alignment is a very general concept that is used to refer to several problems. For example, most statistical machine translation systems are trained on bilingual texts which are annotated with word-to-word alignment [Koehn et al., 2003; Chiang, 2005]. **Word alignment** models are thus developed to generate links between words in two sentences [Vogel et al., 1996; Och and Ney, 2003; Taskar et al., 2005; Dyer et al., 2013]. While the outputs of these systems are discrete variables, the underlying models are mostly probabilistic and continuous. Therefore, the correspondence between word alignment and the attention models discussed here is apparent because they are both learned to assign a weight to each pair of words.

Note that despite the similarity between alignment and attention problems, their goals are different. In most cases word alignment models are used as individual systems to produce alignment results for downstream systems, whereas attention models are typically treated as components of bigger systems and do not work alone (see Figure 5.9 for a comparison of these models). This makes them fit different types of sequence-to-sequence systems in practice: word alignment is one step of a pipelined system, and attention is some intermediate states of a neural network.

Nevertheless, word alignment and attention are two related problems, and can help each other in some cases. For example, one way to see how an attention model behaves is to induce word alignments from it and measure the quality of these word alignments [Tu et al., 2016; Li et al., 2019; Garg et al., 2019]. Also, systems equipped with the attention mechanism can be guided by external word alignment resources [Mi et al., 2016b; Liu et al., 2016b].

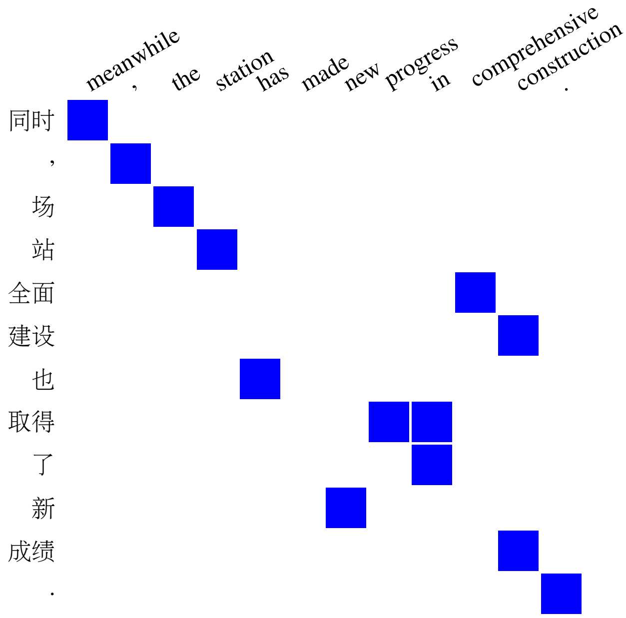
2. Introducing Priors

As discussed in Section 5.3.1, the attention models implicitly define an attention distribution over $\{\mathbf{h}_1, ..., \mathbf{h}_m\}$ by which we can compute a weighted sum of these representations. This distribution is expressed in terms of the alignment weights and is learned as part of a model. In addition to learning the attention distribution in an end-to-end fashion, we can also define it based on our knowledge about how we concentrate on different parts of a sequence when reading it.

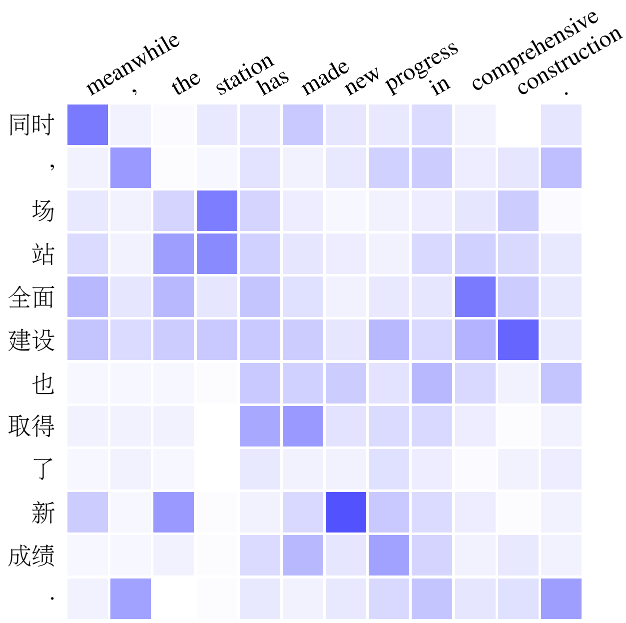
One approach is to directly impose some structure on the distribution. A simple example is that we consider only a span of the sequence for attention and discard the rest. Let $[\rho_i - D, \rho_i + D]$ be a $2D+1$ word window centered at position ρ_i of the source-side sequence. We can connect $\mathbf{s}_i$ only to $\mathbf{h}_{\rho_i - D} ... \mathbf{h}_{\rho_i + D}$ and obtain a local context vector in the following form

$$\mathbf{c}_i = \mathrm{Att}(\mathbf{h}_{\rho_i - D} ... \mathbf{h}_{\rho_i + D}, \mathbf{s}_i) \tag{5.63}$$

This approach is also called **local attention**. The problem of determining ρ_i is similar to the **reordering problem** in machine translation. For translation between languages with



(a) A heat map of word alignments



(b) A heat map of attention weights

Figure 5.9: Heat maps of a word alignment model and an attention model for a pair of Chinese and English sentences. The heat maps are represented as shaded grids in which each cell describes the correspondence of a pair of Chinese and English words. This correspondence is shown on a color scale ranging from white denoting a weight of 0 to pure blue denoting a weight of 1.

similar word orders, it is common to assume that the translation is monotonic and ρ_i is linear with respect to i [Koehn, 2004], e.g., $\rho_i = \lfloor m\frac{i}{n} \rfloor$ or $\lceil m\frac{i}{n} \rceil$. An alternative method is to use a neural network to predict a relative position in the source-side sequence (denoted by r_i) [Luong et al., 2015]. ρ_i can then be defined to be $\lfloor mr_i \rfloor$ or $\lceil mr_i \rceil$.

In another thread of research, a new distribution is derived by combining the original attention distribution and some prior distribution. The simplest such distribution takes the form of linear interpolation

$$\widetilde{\Pr}(\mathbf{h}_j|\mathbf{s}_i) = \eta \cdot \Pr(\mathbf{h}_j|\mathbf{s}_i) + (1-\eta) \cdot \mathrm{Prior} \tag{5.64}$$

where Prior is the prior, and η is the interpolation coefficient. When $\eta = 1$, it is a standard attention model. By contrast, when $\eta = 0$, the attention is completely dependent on the prior [You et al., 2020].

The prior can be chosen in many ways. A simple choice is to assume Prior to be a Gaussian distribution $\mathrm{Gaussian}(\mu, \sigma^2)$. This makes the model well explained: the attention is concentrated on some point of the sequence and decreases its strength as we spread the attention from this point. To determine the mean and variance of the Gaussian distribution, we can use the same technique described above, say, we develop two neural networks to compute them respectively.

The interpolation can also be considered an intermediate step of computing the attention distribution. For example, consider the QKV attention discussed in Section 5.3.2. The interpolation can be placed on the query-key dot-product [Yang et al., 2018a; Guo et al., 2019]. To this end, we can modify Eq.(5.28) in the following form

$$\begin{aligned}\alpha_j &= \mathrm{Softmax}(\frac{\mathbf{q}\mathbf{k}_j^{\mathrm{T}}}{\beta} + \eta \mathrm{Prior}) \\ &= \mathrm{Softmax}(\frac{\mathbf{s}_i\mathbf{h}_j^{\mathrm{T}}}{\beta} + \eta \mathrm{Prior})\end{aligned} \tag{5.65}$$

As $\frac{\mathbf{q}\mathbf{k}_j^{\mathrm{T}}}{\beta}$ (or $\frac{\mathbf{s}_i\mathbf{h}_j^{\mathrm{T}}}{\beta}$) is not constrained in $[0,1]$, Prior is re-scaled by a hyper-parameter η.

Sometimes, priors arise in the context where the knowledge of attention comes from external sources. As discussed above, incorporating word alignments into attention models is one of the simplest ways to do this. The idea can be extended to make use of more structural information, such as fertility and coverage [Cohn et al., 2016; Feng et al., 2016; Tu et al., 2016], or more task-specific constraints, such as monotonic alignments between input and output sequences [Raffel et al., 2017; Chiu and Raffel, 2018]. Also, as with syntactic machine translation systems, parse trees can be used to bias the process of attention as an auxiliary input. For example, dependency trees are a widely used source of information in modeling word correspondence for either sequence-to-sequence [Chen et al., 2018] or sequence modeling problems [Zhang et al., 2020b; Nguyen et al., 2020; Xu et al., 2021].

Since attention models can be computationally expensive in large-scale applications, researchers have also developed efficient attention models by introducing more inductive biases into model design [Tay et al., 2020]. This line of research can broadly be categorized into efficient methods for NLP. In Chapter 6 we will present a discussion.

3. Attention in Memory Networks

As well as being of great interest in sequence-to-sequence systems, the attention mechanism is extensively used in memory-based neural models [Sukhbaatar et al., 2015; Graves et al., 2014; Kumar et al., 2016]. As discussed in Chapter 4, a memory system maintains a collection of data items and allows users to retain and retrieve information. Given a query, it computes, in some way, the match between the query and the key of each data item. This procedure is also called **addressing** [Graves et al., 2014]. Such addressing is typically implemented by first representing the query and the data item as real-valued vectors, and then calculating a weight by considering some similarity between the two vectors. The result of the retrieval is a weighted sum of all the data items. This formalism fits perfectly with the model of the QKV attention discussed in Section 5.3.2.

Provided the attention mechanism and the memory mechanism are correlated, though not from a psychology perspective, we can view attention as a process of retrieving information in a memory (i.e., $\{\mathbf{h}_1, ..., \mathbf{h}_m\}$) for a given query (i.e., $\mathbf{s}_i$). Thus we can interpret a sequence-to-sequence system with the attention mechanism as follows. On the source-side, we store information in a memory represented as a sequence of vectors $\mathbf{h}_1...\mathbf{h}_m$. Then, we retrieve from this memory to recover step by step the source-side information on the target-side.

4. Beyond Sequence-to-Sequence Problems

While we restrict our discussion to the problem of transformation from one sequence to another sequence in this section, the general approach of attention can be used to deal with other problems. As mentioned in Section 5.3.2, and will be discussed in Chapter 6, a well-known variant of this approach is self-attention. In self-attention, the QKV attention model is used as a sequence model, and we have only one sequence of variables as input. As a result, the outputs of this attention model can be treated as new representations of the input sequence. Self-attention provides a general approach to modeling the interactions and dependencies between input variables, and so can be applied to a variety of problems. For example, we can concatenate $\mathbf{h}_1...\mathbf{h}_m$ and $\mathbf{s}_1...\mathbf{s}_n$ as a new sequence $\mathbf{h}_1...\mathbf{h}_m\mathbf{s}_1...\mathbf{s}_n$, and run this model on the sequence. In this way, self-attention is easily extended to a sequence-to-sequence model [Lample and Conneau, 2019; Raffel et al., 2020]. Such an approach even works when $\mathbf{h}_1...\mathbf{h}_m$ and $\mathbf{s}_1...\mathbf{s}_n$ represent different types of data. For example, we can use $\mathbf{h}_1...\mathbf{h}_m$ to represent a text and use $\mathbf{s}_1...\mathbf{s}_n$ to represent an image. Then, we have a multi-modal model that fuses textual and visual representations by performing self-attention on them [Chen et al., 2020].

Another approach to joint representation learning of sequences is to build multiple attention models so that each sequence can learn from other sequences. An example of such models is **co-attention**, which has been widely used in multi-modal language processing [Lu et al., 2016]. For example, for the purposes of **visual question answering** (**VQA**), we wish to figure out which parts of the image are related to a word of the question and to figure out which words of the question are related to a given part of the image. To do this we will build two attention models: one for image-to-text attention, and one for text-to-image attention. The outputs of both models can be thought of as joint representations for the image and text, and thus can be used as features for answer prediction.

The attention models discussed in this section are order-independent for input. This is an issue for dealing with sequential data, and can be addressed by encoding order information in the inputs themselves (see Chapters 4 and 6). On the other hand, the simplicity of this formulation makes these models general: the input data of the models needs not to be sequential. As a result, the attention models can be directly applied to more complex data, such as graphs [Veličković et al., 2018; Lee et al., 2019].

5.4 Search

Search is a fundamental issue in artificial intelligence, and plays an important role in many NLP systems. The search problem is a computational challenge here because the number of hypotheses in the search space increases exponentially with the length of the sequence and the size of the vocabulary on the target-side. Exhaustive search in this case is simply slow. Therefore, real-world systems often involve search algorithms or heuristics to ensure that optimal or sub-optimal solutions can be found in an acceptable time.

For many practical sequence-to-sequence applications, the search problem, also called the inference problem sometimes, can be formulated as the following equation

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y}\in\Omega} \mathrm{Score}(\mathbf{x},\mathbf{y}) \tag{5.66}$$

where $\mathrm{Score}(\mathbf{x},\mathbf{y})$ is a model that measures the goodness of $\mathbf{y}$ given $\mathbf{x}$.

This equation takes a slightly different form from that of Eq. (5.2). First, we use $\mathrm{Score}(\mathbf{x},\mathbf{y})$ instead of $\Pr(\mathbf{y}|\mathbf{x})$ as the goodness function. While a typical approach to training sequence-to-sequence models is to maximize $\Pr(\mathbf{y}|\mathbf{x})$ (or $\Pr(\mathbf{x},\mathbf{y})$), we often need to consider task-specific problems when performing inference on test data, for example the problem of length bias. It is therefore common to involve other terms, as well as $\Pr(\mathbf{y}|\mathbf{x})$, to define the objective function for search (see Section 5.4.1). A second difference between Eq. (5.66) and Eq. (5.2) arises from the form of the search space which is constrained to Ω. In general, Ω is a pruned search space and contains a relatively small number of hypotheses. A common way to achieve this is through the use of pruning techniques and advanced search algorithms (see Section 5.4.2). In this section we consider solutions to these problems and some of the refinements. These methods are largely motivated by the development of machine translation, but the discussions here are general and can be considered in most text generation problems.

5.4.1 The Length Problem

Recall from Section 5.2.2 that the probability of the target-side sequence $\mathbf{y}$ given the source-side sequence $\mathbf{x}$ can be written as a product of probabilities of each word y_i given both the generated words $y_0...y_{i-1}$ and $\mathbf{x}$. Here we re-express Eq. (5.14) using simpler notation, as follows

$$\Pr(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{n} \Pr(y_i|\mathbf{y}_{<i}, \mathbf{x}) \tag{5.67}$$

where $\mathbf{y}_{<i}$ denotes the sequence $y_0...y_{i-1}$. This can be equivalently expressed in terms of log probabilities

$$\log \Pr(\mathbf{y}|\mathbf{x}) = \sum_{i=1}^{n} \log \Pr(y_i|\mathbf{y}_{<i}, \mathbf{x}) \tag{5.68}$$

Such a simple form of modeling has clear advantages as practical models for NLP, but unfortunately, this leads to a preference for shorter target-side sequences over longer target-side sequences. So it seems implausible to simply take $\mathrm{Score}(\mathbf{x}, \mathbf{y}) = \Pr(\mathbf{y}|\mathbf{x})$ or $\log \Pr(\mathbf{y}|\mathbf{x})$ in search because the result is very probably a short sequence, say, a sequence of one or two words. This problem is a direct consequence of the inductive bias of the above model. From a supervised learning perspective, another reason for this is that **teacher forcing** is used to train the model: only a ground-truth target-side sequence is considered in training, and the model is forced to output this ground-truth. By contrast, when applying this model to test data, we need to explore a big set of $\mathbf{y}$ of different lengths, and to compare different $\mathbf{y}$ in terms of a function that is different from the one learned on the training data.

This problem can be addressed through a technique called **length reward**, which gives bonuses to longer sequences by adding a term to $\mathrm{Score}(\mathbf{x}, \mathbf{y})$ [He et al., 2016]. As discussed in Chapter 3, a commonly used form of length reward is given by

$$\mathrm{Score}(\mathbf{x}, \mathbf{y}) = \log \Pr(\mathbf{y}|\mathbf{x}) + \lambda \cdot n \tag{5.69}$$

Here the length reward term is the length of $\mathbf{y}$ (i.e., $n = |\mathbf{y}|$), weighted by the parameter $\lambda > 0$. Improvements on this approach involve replacing n with an estimated sequence length by using a length prediction model. For example, we can bound the reward in the following form [Huang et al., 2017; Yang et al., 2018b]

$$\mathrm{Score}(\mathbf{x}, \mathbf{y}) = \log \Pr(\mathbf{y}|\mathbf{x}) + \lambda \cdot \max(l_p, n) \tag{5.70}$$

where l_p is a predicted length, and is generally defined to be a scaled length of $\mathbf{x}$, that is, $l_p = \mathrm{scalar}_p \cdot m$.

If we substitute the log probability $\log \Pr(\mathbf{y}|\mathbf{x})$ given by Eq. (5.68) into Eq. (5.69), we obtain

$$\begin{aligned}\mathrm{Score}(\mathbf{x}, \mathbf{y}) &= \sum_{i=1}^{n} \log \Pr(y_i|\mathbf{y}_{<i}, \mathbf{x}) + \lambda \cdot n \\ &= \sum_{i=1}^{n} [\log \Pr(y_i|\mathbf{y}_{<i}, \mathbf{x}) + \lambda] \end{aligned} \tag{5.71}$$

Thus, we can interpret the length reward term as a reward to each word y_i. Such a method has been widely used in **statistical machine translation** (**SMT**) systems in which the rewards are treated as features of a log-linear model [Koehn et al., 2003; Chiang, 2007]. To find an appropriate value of λ, we can either use minimum error rate training [Och, 2003], following the convention in SMT, or use gradient-based methods as in neural network-based systems [Murray and Chiang, 2018].

A second approach to biasing search towards longer sequences, called **length normalization**, is to divide $\log\Pr(\mathbf{y}|\mathbf{x})$ by a length correction term, written in the following form

$$\mathrm{Score}(\mathbf{x},\mathbf{y}) = \frac{\log\Pr(\mathbf{y}|\mathbf{x})}{n_{\mathrm{correct}}} \tag{5.72}$$

A simple example of this model is to define the length correction term as the sequence length [Jean et al., 2015], like this

$$\begin{aligned} n_{\mathrm{correct}} &= n \\ &= |\mathbf{y}| \end{aligned} \tag{5.73}$$

In this case, $\frac{\log\Pr(\mathbf{y}|\mathbf{x})}{n} = \frac{\sum_{i=1}^{n}\log\Pr(y_i|\mathbf{y}_{<i},\mathbf{x})}{n}$ can be viewed as the log-scale geometric mean of the probabilities $\{\Pr(y_i|\mathbf{y}_{<i},\mathbf{x})\}$[5].

Another example is the one used in the GNMT system [Wu et al., 2016]. It takes the exponential of the shifted, re-scaled n, as follows

$$n_{\mathrm{correct}} = \frac{(5+n)^{\alpha}}{(5+1)^{\alpha}} \tag{5.76}$$

where the power α is a hyper-parameter and can be determined empirically on a tuning set. To compare different methods, Table 5.2 shows a list of scoring functions for length reward and length normalization.

In machine translation, the length problem is also closely related to the **coverage** problem which has been discussed extensively in SMT. When translating a source-side sequence, we wish to know how many times each word is translated. Then, we will say that **over-translation** occurs (i.e., a longer translation) if some words are translated too many times, and that **under-translation** occurs (i.e., a shorter translation) if some words are not sufficiently translated. Traditionally, the coverage of a source-side sequence is described in terms of an m-dimensional

[5] Suppose $\{a_1,...,a_n\}$ are n variables. Since

$$\exp\left(\frac{\sum_{i=1}^{n}\log a_i}{n}\right) = \left(\prod_{i=1}^{n} a_i\right)^{\frac{1}{n}} \tag{5.74}$$

we have

$$\frac{\sum_{i=1}^{n}\log a_i}{n} = \log\left(\prod_{i=1}^{n} a_i\right)^{\frac{1}{n}} \tag{5.75}$$

Method	Form of $\mathrm{Score}(\mathbf{x},\mathbf{y})$
No Reward/Normalization	$\mathrm{Score}(\mathbf{x},\mathbf{y}) = \log\Pr(\mathbf{y}\vert\mathbf{x})$
Length Reward	$\mathrm{Score}(\mathbf{x},\mathbf{y}) = \log\Pr(\mathbf{y}\vert\mathbf{x}) + \lambda \cdot n$
Bounded Length Reward	$\mathrm{Score}(\mathbf{x},\mathbf{y}) = \log\Pr(\mathbf{y}\vert\mathbf{x}) + \lambda \cdot \max(l_p, n)$
Length Normalization (Basic)	$\mathrm{Score}(\mathbf{x},\mathbf{y}) = \frac{\log\Pr(\mathbf{y}\vert\mathbf{x})}{n}$
Length Normalization (GNMT)	$\mathrm{Score}(\mathbf{x},\mathbf{y}) = \frac{\log\Pr(\mathbf{y}\vert\mathbf{x})}{(5+n)^{\alpha}/(5+1)^{\alpha}}$

Table 5.2: Scoring functions for length reward and length normalization. $m = |\mathbf{x}|$, $n = |\mathbf{y}|$, and $l_p = \mathrm{scalar}_p \cdot m$. λ and α are parameters.

vector $\begin{bmatrix} v_1 & ... & v_m \end{bmatrix}$, called the **coverage vector**. v_j describes to what extent the source-side word x_j is translated. In SMT systems v_j is a binary variable: 0 denotes untranslated, and 1 denotes translated. However, NMT systems have no such symbolic coverage mechanism. Instead, they have models that compute the attention weights between x_j and all the target-side words. Therefore, one way to define what we mean by the coverage of a word is to consider how strong x_j connects to the target-side words. To do this, we extend v_j to be a continuous variable, given by

$$v_j \quad = \quad \sum_{i=1}^{n} \alpha_{i,j} \tag{5.77}$$

v_j can thus be viewed as the "number of times" x_j is translated, say, $v_j = 0$ means that x_j is not translated at all, and $v_j = 1$ means that x_j is counted only once in translation. Consider the example in Figure 5.9. For the source-side word 建设, the corresponding attention weights are shown below.

	meanwhile	,	the	station	has	made	new	progress	in	comprehensive	construction	.
建设 ($j=5$)	0.23	0.14	0.20	0.22	0.22	0.21	0.10	0.29	0.15	0.30	0.60	0.09

$$v_5 = \sum_{i=1}^{12} \alpha_{i,5} = 2.75$$

We will say that 建设is translated 2.75 times. It is possible to make use of $\{v_1, ..., v_m\}$ to define how much the source-side sequence is covered in translation. A simple way to do this is to develop a coverage score $\mathrm{cp}(\mathbf{x},\mathbf{y})$ by combining $\{v_1, ..., v_m\}$. For example, the GNMT system defines $\mathrm{cp}(\mathbf{x},\mathbf{y})$ in the following form

$$\mathrm{cp}(\mathbf{x},\mathbf{y}) \quad = \quad \beta \sum_{j=1}^{m} \log(\min(v_j, 1)) \tag{5.78}$$

where β is a weight for the coverage model. The underlying idea is that when $\upsilon_j \geq 1$ the word x_j is assumed to be adequately translated; when $\upsilon_j < 1$ the word x_j is assumed to be lack of translation. Thus $\mathrm{cp}(\mathbf{x},\mathbf{y})$ penalizes hypotheses in which some of the source-side words miss parts of the translations. An improvement to this form is given by Li et al. [2018]

$$\mathrm{cp}(\mathbf{x},\mathbf{y}) = \beta \sum_{j=1}^{m} \log(\max(\upsilon_j,\gamma)) \tag{5.79}$$

where γ is the hyper-parameter for truncation, giving a tolerance for under-translation. A similar form was proposed in [Chorowski and Jaitly, 2017]

$$\mathrm{cp}(\mathbf{x},\mathbf{y}) = \beta \sum_{j=1}^{m} \mathbb{1}(\upsilon_j > \gamma) \tag{5.80}$$

It just counts the number of times υ_j is greater than γ.

$\mathrm{cp}(\mathbf{x},\mathbf{y})$ can be easily introduced into search by adding it to $\mathrm{Score}(\mathbf{x},\mathbf{y})$. For example, the GHKM-style scoring function is defined to be

$$\mathrm{Score}(\mathbf{x},\mathbf{y}) = \frac{\log \Pr(\mathbf{y}|\mathbf{x})}{(5+n)^{\alpha}/(5+1)^{\alpha}} + \mathrm{cp}(\mathbf{x},\mathbf{y}) \tag{5.81}$$

In practice, modifying $\mathrm{Score}(\mathbf{x},\mathbf{y})$ is not the only way to address the length problem in search. An alternative approach is to have architecture changes for modeling the problem [Tu et al., 2016; Mi et al., 2016a; Sankaran et al., 2016; See et al., 2017; Malaviya et al., 2018]. Note that, sometimes the length of the target-side sequence has been specified or predicted in some way. In these cases, we can either develop models not dependent on the auto-regressive assumption [Gu et al., 2018], or develop length-controllable text generation systems for interesting applications [Rush et al., 2015; Kikuchi et al., 2016].

5.4.2 Pruning and Beam Search

There are many ways to define a search space. As a general concept in computer science, a search space is often referred to as the domain of the problem that is searched. For sequence-to-sequence problems, we can think of a hypothesis as a mapping from a source-side sequence $\mathbf{x}$ to a target-side sequence $\mathbf{y}$, and can think of a search space as a collection of such hypotheses[6].

We can implement a search program by organizing hypotheses in an understandable way so that we can look at the search space for the problem. Recall that in Eqs. (5.67-5.68) we assign a probability of $\mathbf{y}$ given $\mathbf{x}$ by using a left-to-right factorization. A typical search system maintains a set of hypotheses (or partial hypotheses) and builds up these hypotheses from left to right[7]. The search procedure begins with an initial hypothesis set Z_0 containing

[6] Here we use $(\mathbf{x},\mathbf{y})$ to denote a hypothesis. When there are multiple mappings from $\mathbf{x}$ to $\mathbf{y}$, a hypothesis can be represented as $(\mathbf{x},\mathbf{y},d)$ where d denotes the mapping. For example, if we transform $\mathbf{x}$ to $\mathbf{y}$ with a synchronous grammar, there might be multiple derivations of grammar rules to do this.

[7] A hypothesis is called partial when the corresponding target-side sequence does not end with $\langle \mathrm{EOS} \rangle$, i.e., an incomplete target-side sequence. In this section we use the terms *hypothesis* and *partial hypothesis* interchangeably

only one hypothesis z_0 whose target-side is y_0 by considering $y_0 = \langle\text{SOS}\rangle$ is the start symbol for all target-side sequences. Then, we extend this hypothesis set over a number of search steps. Suppose we have a sequence of hypothesis sets $Z_0 ... Z_{n_{\max}}$ where $n_{\max}$ is the maximum number of search steps. At step i, we wish to extend each hypothesis by adding a new word v_k drawn from the vocabulary V_{y}. Let $z.src$ be the source-side of z and $z.tgt$ be the target-side of z. Clearly, we have $z.src = \mathbf{x}$ for any z. Given a hypothesis $z_{\text{cur}} \in Z_{i-1}$, we can extend it to $|V_{\mathrm{y}}|$ hypotheses $\{z^1_{\text{next}}, ..., z^{|V_{\mathrm{y}}|}_{\text{next}}\}$, given by

$$\begin{aligned} \{z^1_{\text{next}}, ..., z^{|V_{\mathrm{y}}|}_{\text{next}}\} &= \text{Extend}(z_{\text{cur}}, V_{\mathrm{y}}) \\ &= \bigcup_{v_k \in V_{\mathrm{y}}} \text{Extend}(z_{\text{cur}}, v_k) \end{aligned} \tag{5.82}$$

Here $\text{Extend}(z_{\text{cur}}, v_k)$ is a function that extends the input hypothesis z_{cur} with a word $v_k \in V_{\mathrm{y}}$. The target-side of a resulting hypothesis is the concatenation of $z_{\text{cur}}.tgt$ and v_k, written as[8],

$$z^k_{\text{next}}.tgt = z_{\text{cur}}.tgt \circ v_k \tag{5.83}$$

These new hypotheses $\{z^1_{\text{next}}, ..., z^{|V_{\mathrm{y}}|}_{\text{next}}\}$ are then added to Z_i. Figure 5.10 illustrates a few steps in this hypothesis extension process. We see that all the hypotheses can easily be represented as a tree structure. Here Z_i corresponds to a set of the nodes at level i of the search tree, and we simply have

$$|Z_i| = |V| \cdot |Z_{i-1}| \tag{5.84}$$

In other words, the size of Z_i grows exponentially with the number of steps, say, $|Z_i| = |V|^i$.

Each hypothesis z is associated with a log probability $\log \Pr(z.tgt|z.src)$. $\log \Pr(z.tgt|z.src)$ simply takes the form of Eq. (5.68), and can be defined in a recursive fashion

$$\begin{aligned} \log \Pr(z^k_{\text{next}}.tgt|z^k_{\text{next}}.src) &= \log \Pr(z_{\text{cur}}.tgt|z_{\text{cur}}.src) + \\ &\quad \log \Pr(v_k|z_{\text{cur}}.tgt, z_{\text{cur}}.src) \end{aligned} \tag{5.85}$$

As an example, suppose $z^k_{\text{next}}.tgt = y_0 ... y_{i+1}$. The form of Eq. (5.85) becomes clear from the following rewriting

$$\begin{aligned} \underbrace{\log \Pr(y_0 ... y_{i+1}|\mathbf{x})}_{\log \Pr(z^k_{\text{next}}.tgt|z^k_{\text{next}}.src)} &= \underbrace{\log \Pr(y_0 ... y_i|\mathbf{x})}_{\log \Pr(z_{\text{cur}}.tgt|z_{\text{cur}}.src)} + \underbrace{\log \Pr(y_{i+1}|y_0 ... y_i, \mathbf{x})}_{\log \Pr(v_k|z_{\text{cur}}.tgt, z_{\text{cur}}.src)} \\ &= \sum_{k=1}^{i} \log \Pr(y_k|\mathbf{y}_{<k}, \mathbf{x}) + \log \Pr(y_{i+1}|y_0 ... y_i, \mathbf{x}) \\ &= \sum_{k=1}^{i+1} \log \Pr(y_k|\mathbf{y}_{<k}, \mathbf{x}) \end{aligned} \tag{5.86}$$

because their forms are the same.

[8] We use $a \circ b$ to denote the concatenation of two strings a and b.

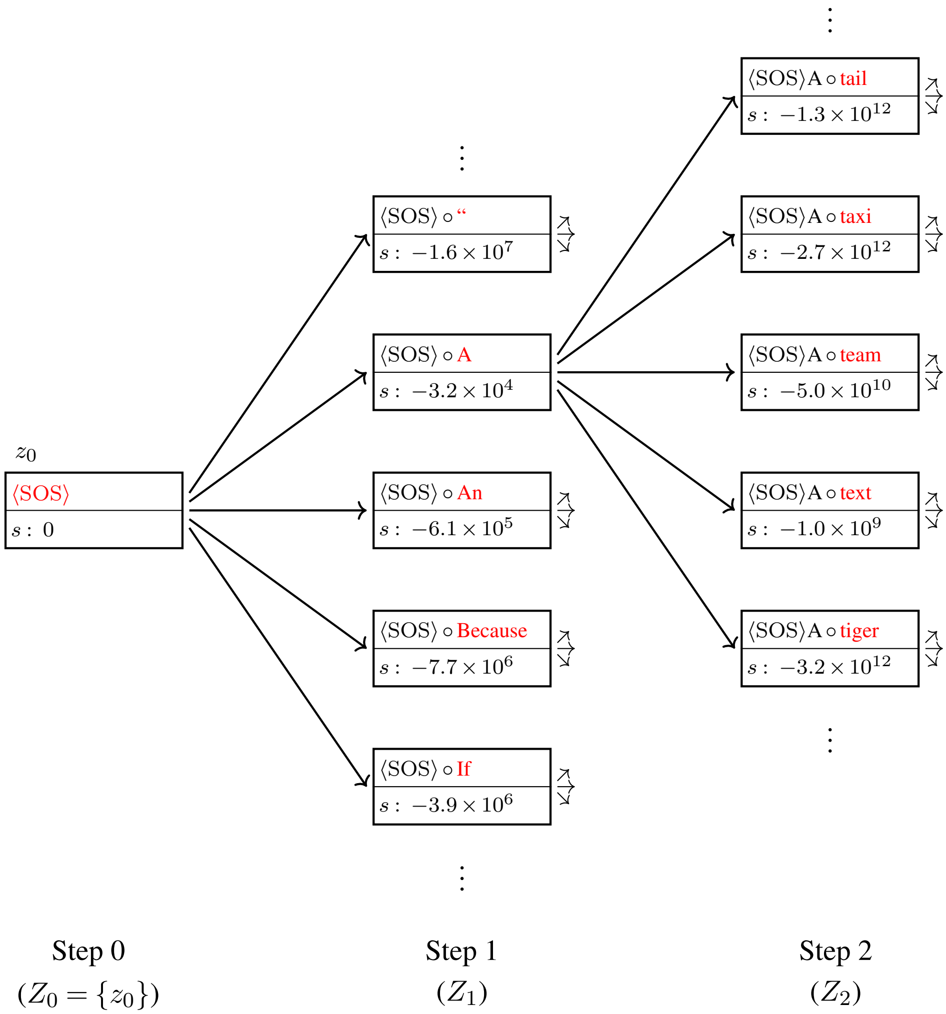


Figure 5.10: Illustration of hypothesis extension in first 3 steps. Each (partial) hypothesis is represented as a box in which we show the corresponding target-side sequence and model score. Each search step is associated with a hypothesis set Z_i. We start with a hypothesis $z_0 \in Z_0$ denoting the start symbol ⟨SOS⟩. In step i, we extend every hypothesis in Z_{i-1} by trying to append every word from a vocabulary V (see words in red). This operation will result in $|V| \cdot |Z_{i-1}|$ hypotheses, forming the hypothesis set Z_i. The hypothesis extension procedure represents a breadth-first search algorithm: we create all the nodes (or search states) at depth $i-1$ before moving to depth i. A tree structure is created along with this procedure, and a leaf node of the tree can trace the search path back to the root node.

Given this probability, we can then compute $z.score = \text{Score}(z.src, z.tgt)$, as in Section 5.4.1. This enables us to compare different hypotheses in terms of $z.score$. If a hypothesis ends with the symbol ⟨EOS⟩, it is called complete and is not extended anymore. Once a hypothesis is complete, it is added to a max-heap[9]. We can dump the hypotheses with maximum model scores from the heap. In general, the search procedure will stop if we find a certain number of complete hypotheses. For example, we can stop searching when the heap is full (see more discussions later in this subsection). The resulting search algorithm is described below.

Algorithm: A Simple Breadth-first Search Algorithm
SimpleSearch($\mathbf{x}$)
// Search for the best hypothesis given the source-side sequence $\mathbf{x}$
1. Create a Heap with $size_{\text{heap}}$ elements
2. $Z_0 = \{z_0\}$ where $z_0.src = \mathbf{x}$ and $z_0.tgt = y_0$
3. For each step $i = 1$ to $n_{\max}$
4.   For each hypothesis $z_{\text{cur}} \in Z_{i-1}$
5.   For each word $v_k \in V_{\text{y}}$
6. $z_{\text{next}} =$ Extend$(z_{\text{cur}}, v_k, \mathbf{x})$
7.   If $z_{\text{next}}.tgt$ ends with $\langle \text{EOS} \rangle$, then
8.   Add z_{next} to Heap
9.   Else
10.   Add z_{next} to Z_i
11.   If Heap is full and/or other stopping criteria are met, then
12.   Break all the loops
13. return Heap.Pop()

Extend$(z_{\text{cur}}, v_k, src)$
// Create a new hypothesis by appending a new word v_k to the target-side of z_{cur}
1. Create a new hypothesis z_{next}
2. $z_{\text{next}}.src = src$
3. $z_{\text{next}}.tgt = z_{\text{cur}}.tgt \circ v_k$
4. $z_{\text{next}}.prob = z_{\text{cur}}.prob + \log \Pr(v_k | z_{\text{cur}}.tgt, z_{\text{cur}}.src)$ // see Eq. (5.85)
5. $z_{\text{next}}.score = \text{score}(z_{\text{next}}.src, z_{\text{next}}.tgt)$ // see Section 5.4.1
6. Return z_{next}

If the hypothesis heap has an infinite capacity ($size_{\text{heap}} = \infty$), this algorithm will perform an exhaustive search over a space of all hypotheses whose target-side lengths are up to $n_{\max}$, resulting in at most $1 + |V_{\text{y}}| + |V_{\text{y}}|^2 + \cdots + |V_{\text{y}}|^{n_{\max}} = \frac{|V_{\text{y}}|^{n_{\max}+1} - 1}{|V_{\text{y}}| - 1}$ complete hypotheses. This is an extremely huge search space which is computationally intractable in practice[10]. Therefore, in practical systems it is common to prune the search space in order to make the search tractable. In later parts of this subsection we will introduce two popular search algorithms, both adopting pruning for efficient search.

[9] Given a max-heap a, we use a.Pop() to denote a function popping the top-1 item of a, and use a.PopAll() to denote a function popping all the items of a.

[10] Consider, for example, a vocabulary size of $20,000$ ($|V_{\text{y}}| = 20,000$) and a length limit of 20 ($n_{\max} = 20$). $\frac{|V_{\text{y}}|^{n_{\max}+1} - 1}{|V_{\text{y}}| - 1}$ would be 1.05×10^{86}.

1. Greedy Search

The **greedy strategy** is one of the most common concepts that one learns in textbooks on algorithms. It is based on a heuristic that the globally optimal solution can be approximated by making locally optimal decisions. Although such an approximation can only obtain a locally optimal solution, this is sufficient for many practical applications and its low computational complexity is a clear advantage.

Applying the greedy strategy to the search problem here is straightforward. In each extension given step i, we only consider the best hypothesis up to i. To be more precise, for any Z_i, we only keep the hypothesis with the highest model score and discard the rest. The output of each step of the greedy search is given by

$$z_{\text{best}} = \underset{z_{\text{next}} \in \text{Extend}(Z_{i-1}, V_{\text{y}})}{\arg\max} z_{\text{next}}.score \tag{5.87}$$

Here the function $\text{Extend}(Z_{i-1}, V_{\text{y}})$ has the same meaning as that in Eq. (5.82), but operates on a set of hypotheses, that is,

$$\text{Extend}(Z_{i-1}, V_{\text{y}}) = \bigcup_{z \in Z_{i-1}} \text{Extend}(z, V_{\text{y}}) \tag{5.88}$$

Then, Z_i is defined to be

$$Z_i = \{z_{\text{best}}\} \tag{5.89}$$

A greedy search algorithm for sequence-to-sequence problems is described below.

Algorithm: A Greedy Search Algorithm
GreedySearch($\mathbf{x}$)
// Search for the "best" hypothesis in a greedy manner
1. Create a hypothesis z_{best}
2. $Z_0 = \{z_0\}$ where $z_0.src = \mathbf{x}$ and $z_0.tgt = y_0$
3. For each step $i = 1$ to $n_{\max}$
4. $\quad z_{\text{best}}.score = -\infty$
5. $\quad$ For each hypothesis $z_{\text{cur}} \in Z_{i-1}$
6. $\quad\quad$ For each word $v_k \in V_{\text{y}}$
7. $\quad\quad\quad z_{\text{next}} = \text{Extend}(z_{\text{cur}}, v_k, \mathbf{x})$
8. $\quad\quad\quad$ If $z_{\text{best}}.score < z_{\text{next}}.score$, then
9. $\quad\quad\quad\quad z_{\text{best}} = z_{\text{next}}$
10. $\quad$ If $z_{\text{best}}.tgt$ ends with $\langle\text{EOS}\rangle$ and/or other stopping criteria are met, then
11. $\quad\quad$ Break the loop
12. $\quad Z_i = \{z_{\text{best}}\}$
13. Return z_{best}

In each step of search, we have only one active hypothesis to extend (i.e., $|Z_{i-1}| = 1$) and therefore need $|V|$ extensions from which we select the best one for the next step of search. The total number of times $\mathrm{Extend}(z_{\mathrm{cur}}, v_k)$ is called is $|V| \cdot n_{\max}$. Provided $\mathrm{Extend}(z_{\mathrm{cur}}, v_k)$ is a fixed-cost function, the time complexity of the algorithm is linear with respect to $|V|$ and $n_{\max}$.

2. Beam Search

Beam search is a natural extension of the above 1-best greedy search algorithm. It is based on the greedy heuristics as well, and is thus a type of greedy algorithm. The idea of beam search is to keep at each step a number of the most promising hypotheses rather than the 1-best hypothesis. A beam is a data structure that stores the best hypotheses we have generated so far. The number of hypotheses in a beam is a predetermined parameter, called **beam width** or **beam size**. Here we can simply view Z_i as a beam, written as

$$Z_i = \{z_{\mathrm{best}}^1, ..., z_{\mathrm{best}}^{size_{\mathrm{beam}}}\} \tag{5.90}$$

where $size_{\mathrm{beam}}$ is the beam size. z_{best}^1 is the best hypothesis in the extension $\mathrm{Extend}(Z_{i-1}, V_{\mathrm{y}})$ (see Eq. (5.87)), z_{best}^2 is the 2nd best hypothesis in $\mathrm{Extend}(Z_{i-1}, V_{\mathrm{y}})$, and so on.

The following pseudo-code describes a beam search algorithm for sequence-to-sequence problems.

Algorithm: A Beam Search Algorithm
$\mathrm{BeamSearch}(\mathbf{x})$
// Search for the "best" hypothesis by considering a number of best candidates
// in each step
1. Create a Heap with $size_{\mathrm{heap}}$ elements
2. $Z_0 = \{z_0\}$ where $z_0.src = \mathbf{x}$ and $z_0.tgt = y_0$
3. For each step $i = 1$ to $n_{\max}$
4.   Create a heap Beam with $size_{\mathrm{beam}}$ elements
5.   For each hypothesis $z_{\mathrm{cur}} \in Z_{i-1}$
6.   For each word $v_k \in V_{\mathrm{y}}$
7. $z_{\mathrm{next}} = \mathrm{Extend}(z_{\mathrm{cur}}, v_k, \mathbf{x})$
8.   If $z_{\mathrm{next}}.tgt$ ends with $\langle\mathrm{EOS}\rangle$, then
9.   Add z_{next} to Heap
10.   Else
11. $\mathrm{UpdateBeam}(\mathrm{Beam}, z_{\mathrm{next}})$
12.   If Heap is full and/or other stopping criteria are met, then
13.   Break all the loops
14. $Z_i = \mathrm{Beam.PopAll}()$
15. Return $\mathrm{Heap.Pop}()$

$\mathrm{UpdateBeam}(\mathrm{Beam}, z_{\mathrm{next}})$
// Update Beam with a newly-generated hypothesis z_{next}

1. Add z_{next} to Beam [a]

---

[a] Beam is a max-heap with $size_{\text{beam}}$ elements. So, if $z_{\text{next}}.score$ is lower than all the elements in the heap, the heap will be left unchanged. In other words, Beam only stores top-$size_{\text{beam}}$ best hypotheses and ignores the rest.

The function $\text{UpdateBeam}(\text{Beam}, z_{\text{next}})$ is a direct implementation of **histogram pruning**. Note that this general-purpose framework provides a simple way to implement other pruning methods, and one can modify $\text{UpdateBeam}(\text{Beam}, z_{\text{next}})$ as needed. For example, an alternative method, called **threshold pruning**, retains the hypotheses whose differences in model scores with the best hypothesis in Beam are below a threshold θ_{beam}, say, we discard z_{next} in $\text{UpdateBeam}(\text{Beam}, z_{\text{next}})$ if

$$z_{\text{next}}.score \quad < \quad z_{\text{best}}.score - \theta_{\text{beam}} \tag{5.91}$$

where z_{best} is the best hypothesis in Beam. Alternatively, we can consider a relative threshold method [Freitag and Al-Onaizan, 2017], given by

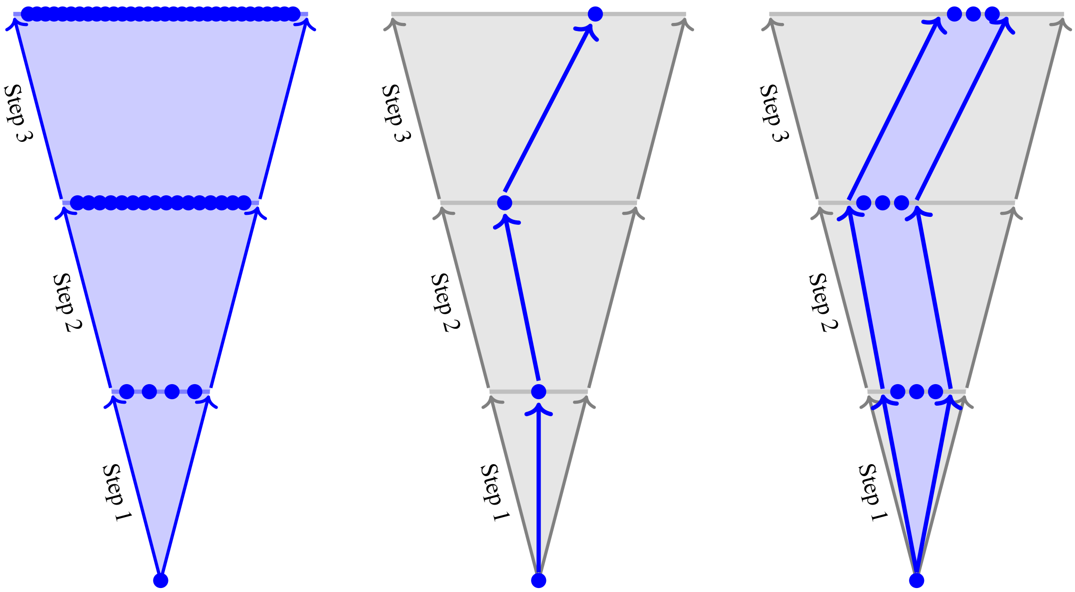
$$z_{\text{next}}.score \quad < \quad z_{\text{best}}.score \cdot \theta_{\text{beam}} \tag{5.92}$$

Figure 5.11 shows a comparison of exhaustive search, (1-best) greedy search and beam search. At one extreme, the optimal solution is guaranteed, but an exponentially large number of search states are visited. At the other extreme, only the minimum number of search states are visited, but the solution is sub-optimal. By contrast, beam search makes a trade-off between the two methods. A larger beam size means more search effort and a higher possibility of finding the optimum, while a smaller beam size means faster search and a higher risk of missing the optimum. It is also possible to use a variable beam size to make a better trade-off during search [Buckman et al., 2016; Post and Vilar, 2018; Kulikov et al., 2019].

An important problem related to these search algorithms is the problem of **search errors**. In general, search errors can be defined in several different ways. Here we say that a search error occurs if the search result is not the same as that of exhaustive search. Common sense tells us that fewer search errors are helpful for finding "better" results. Thus, we often wish to have a more desirable target-side sequence by enlarging the beam size. However, this is not the case for some sequence-to-sequence systems. For example, a search with a larger beam size may lead to a lower translation quality for neural machine translation systems [Koehn and Knowles, 2017]. This inspires very interesting studies on the deterioration issue of large beam search [Ott et al., 2018b; Yang et al., 2018b; Stahlberg and Byrne, 2019].

3. Stopping Criteria

Although the time complexities of the above algorithms are bounded by the maximum number of search steps (i.e., n_{max}), it is important to have more efficient algorithms to stop searching as early as possible, especially for latency-sensitive applications. This typically requires heuristics to design additional criteria for stopping the search procedure at the appropriate point. Some of these stopping criteria are:



(a) Exhausted Search (b) (1-best) Greedy Search (c) Beam Search

Figure 5.11: A comparison of exhaustive search, (1-best) greedy search and beam search. Balls represent search states or partial hypotheses. Exhausted search explores all search states in the search space. By contrast, greedy search keeps only the 1-best path of search states and prunes away the rest. Beam search is a trade-off between them and keeps the most promising search states in a beam in each step.

- If a given number of complete hypotheses are created, then we stop searching. For example, in the beam search algorithm described in this subsection, the search program terminates when we have $size_{\text{heap}}$ complete hypotheses. Another way to implement this idea is to shrink the beam as the number of complete hypotheses increases. In Bahdanau et al. [2014]'s system, once a new complete hypothesis is created, the beam size decreases by 1. Therefore, the search program will terminate if the beam size is reduced to 0.
- If every hypothesis at step i has a score lower than that of the best complete hypothesis in Heap by some margin, then we stop searching. Suppose $z_{\text{bestinall}}$ is the best hypothesis we have generated so far (i.e., $z_{\text{bestinall}} = \text{Heap.Pop}()$). If every hypothesis z_{next} at step i satisfies

$$z_{\text{bestinall}}.score - z_{\text{next}}.score \quad \geq \quad \theta_{\text{all}} \tag{5.93}$$

then we will finish the search process at this step. Here θ_{all} is a parameter. One can specify it with an appropriate value through multiple tries. A simple choice is $\theta_{\text{all}} = 0$, which is employed in some of the popular sequence-to-sequence systems [Ott et al., 2019]. Under some circumstances, such an **early-stop** strategy can guarantee the optimality of search [Huang et al., 2017; Yang et al., 2018b].

- If every hypothesis at step i has a score lower than that of the last complete hypothesis in Heap by some margin, then we stop searching. This is a weak condition for early-stop.
- If the top ranked hypotheses at step i are all complete hypotheses, then we stop searching. This is a more aggressive version of early-stop. For example, in Klein et al. [2017]'s system, the search program terminates at step i if the top-1 hypothesis is a complete hypothesis.
- If the search program consumes a certain amount of computing resources, such as a certain number of floating-point instructions and a certain amount of wall clock time, then we stop searching. In applications where computer performance is limited and latency plays an important role, we will often be interested in this kind of stopping criterion.

Sometimes, the search algorithm will not find any complete hypothesis until hitting the length limit $n_{\max}$. As a practical matter it might be easy in this case to force the best partial hypothesis to be complete by adding $\langle\mathrm{EOS}\rangle$ to its end.

Note that choosing appropriate stopping criteria reflects a trade-off between fast computation and accurate prediction at inference time (call it the **speed-accuracy trade-off**). While it is not always the case that more time a search program takes could result in better results for a sequence-to-sequence system, we would always want to know how close we can get to a better solution to the problem by searching through a larger region of the search space. A discussion of accurate search algorithms can be found in Section 5.4.4.

5.4.3 Online Search

So far in our general discussion of sequence-to-sequence problems, we have assumed that all the source-side words come together as a whole and can be accessed in the entire search process. However, in some practical applications, the inputs are received in order, and we wish to make predictions conditioned on some of the observed inputs. An example of this is online automatic speech recognition in which the system continually takes new acoustic signals and at the same time outputs the corresponding transcription units.

Intuitively, we might think of the generation of the i-th target-side word as a problem of mapping a prefix of the source-side sequence to the target-side vocabulary. We can formulate this by introducing a function $g(i)$ which denotes the maximum length of the prefix of $\mathbf{x}$ we use in generating y_i. Thus, the probability of y_i given the entire sequence $\mathbf{x}$ and the previously generated words $\mathbf{y}_{<i}$ can be approximated by

$$\Pr(y_i|\mathbf{y}_{<i},\mathbf{x}) \quad \approx \quad \Pr(y_i|\mathbf{y}_{<i},\mathbf{x}_{\leq g(i)}) \tag{5.94}$$

where $\mathbf{x}_{\leq g(i)}$ denotes the sub-sequence $x_1...x_{g(i)}$. Then, the log probability of the target-side sequence $\mathbf{y}$ given the source-side sequence $\mathbf{x}$ is written as

$$\begin{aligned}\log \Pr(\mathbf{y}|\mathbf{x}) &= \sum_{i=1}^{n} \log \Pr(y_i|\mathbf{y}_{<i}, \mathbf{x}) \\ &\approx \sum_{i=1}^{n} \log \Pr(y_i|\mathbf{y}_{<i}, \mathbf{x}_{\leq g(i)}) \end{aligned} \tag{5.95}$$

This equation frames a sequence-to-sequence problem as a prefix-to-prefix problem, that is, the prefix $\mathbf{y}_{\leq i}$ is only dependent on the prefix $\mathbf{x}_{\leq g(i)}$. Inference for this model is simple. For each i, the search system waits until all $g(i)$ source-side words are received, and then extends the hypotheses as usual. This can be done by reusing the algorithms described in the previous subsection. For example, we can modify the beam search algorithm and obtain the following online search algorithm.

Algorithm: An Online Beam Search Algorithm
$\mathrm{OnlineBeamSearch}(\mathbf{x}, g(\cdot))$
// Online search in which the search is operated once an adequate number of input
// words are received. In each search step, a number of the most promising candidates
// are considered.
Create a Heap with $size_{\mathrm{heap}}$ elements
$Z_0 = \{z_0\}$ where $z_0.tgt = y_0$
$j = 0$
$i = 1$
$input = \phi$
While $i \leq n_{\max}$ do
If $j < g(i)$, then // read a word from the input stream
$input = input \circ x_j$
Else // make a prediction at step i
// when $g(i)$ input words are observed (stored in $input$)
Create a heap Beam with $size_{\mathrm{beam}}$ elements
For each hypothesis $z_{\mathrm{cur}} \in Z_{i-1}$
For each word $v_k \in V_{\mathrm{y}}$
$z_{\mathrm{next}} = \mathrm{Extend}(z_{\mathrm{cur}}, v_k, input)$
If $input$ equals $\mathbf{x}$ and $z_{\mathrm{next}}.tgt$ ends with $\langle \mathrm{EOS} \rangle$, then
Add z_{next} to Heap
Else
$\mathrm{UpdateBeam}(\mathrm{Beam}, z_{\mathrm{next}})$
If Heap is full and/or other stopping criteria are met, then
Break all the loops
$\mathrm{OutputPartial}(\mathrm{Beam})$
$Z_i = \mathrm{Beam.PopAll}()$

```
23.         i++
24.  Return Heap.Pop()
OutputPartial(Beam)
// Output a partial result
1.   Display the best hypothesis in Beam
```

An advantage of this system is that the output at step i is immediate once we have seen $\mathbf{x}_{\leq g(i)}$. This results in an **online sequence-to-sequence system** in which input words arrive in a continuous stream and predictions can be made just after a "sufficient" number of input words are seen.

While the search problem here seems simple, much remains to be done to define $g(i)$. Clearly, $g(i)$ is a monotonically non-decreasing function. As a simple example, we can define $g(i) = m$ for any i. This will make the above algorithm precisely the same as the standard beam search algorithm that works with a complete input sequence. By contrast, in online sequence-to-sequence tasks, we want $g(i)$ to be as small as possible, and so we can start computation as early as possible in inference. The simplest case of these is that the input and output sequences are synchronous in some way. For example, an automatic speech recognition system assigns each spectral frame a transcription unit. In this case, we have a simple correspondence between inputs and outputs: $m = n$ (i.e., $|\mathbf{x}| = |\mathbf{y}|$), and x_i corresponds to y_i. Then, we can simply define $g(i) = i$, in other words, each time a new input arrives, we make a prediction.

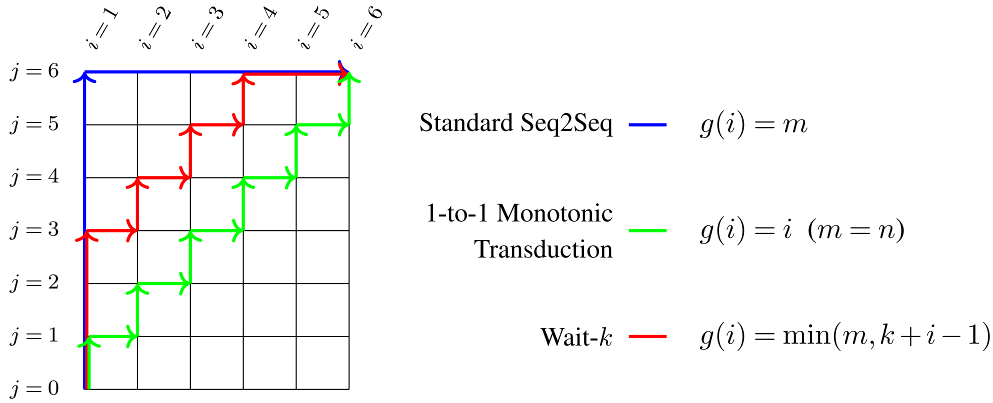
A more complicated case is online sequence-to-sequence problems with reordering, such as **simultaneous translation**, in which a target-side word may depend on source-side words with long-range dependencies. A simple way to address this is to delay the predictions for a number of steps. For example, the wait-k method forces each prediction to lag behind the inputs by k words [Ma et al., 2019]. More formally, the wait-k version of the function $g(i)$ is defined to be

$$g(i) \quad = \quad \min(m, k+i-1) \tag{5.96}$$

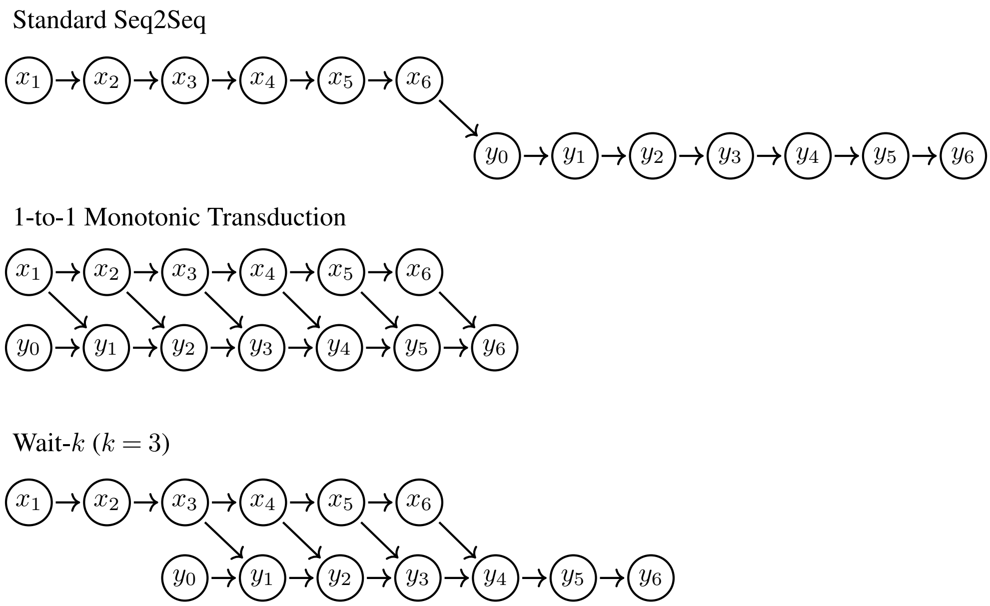
Here k is a hyper-parameter that controls how large a source-side context is considered in predicting target-side words. When $k = \infty$, it is the same as the standard search methods for sequence-to-sequence inference. In simultaneous translation and related tasks, results are in general satisfactory by using a small value of k. A comparison of different $g(i)$ is shown in Figure 5.12.

In some applications of online sequence-to-sequence problems, we may know when to perform search and when to read inputs. For example, in **interactive machine translation** [Casacuberta et al., 2009], the translation of a partial input sequence is triggered by some behaviors of users (such as the action of pressing buttons). In this case, we do not need to define $g(i)$, but view it as an input variable of the model.

Note that while one can directly employ pre-trained sequence-level models for online inference, developing such systems often requires additional training effort. A more principled approach to online sequence-to-sequence modeling is to model the transformation from $\mathbf{x}$ to $\mathbf{y}$ as a sequence of actions [Grissom II et al., 2014; Cho and Esipova, 2016; Gu et al., 2017;



(a) Visualization of $g(i)$.



(b) Action sequences.

Figure 5.12: Visualization (top) and action sequences (bottom) of different $g(i)$ for a pair of sequences ($\mathbf{x} = x_1...x_6, \mathbf{y} = y_1...y_6$). In an action sequence, a circled x_j stands for the action of reading a source-side word (x_j), and a circled y_i stands for the action of predicting the probability of y_i given $\mathbf{x}_{\leq g(i)}$ and $\mathbf{y}_{<i}$. Arrows here stand for dependencies between words. Because y_0 denotes the start symbol $\langle\text{SOS}\rangle$, it could be generated without dependencies on any words.

Zheng et al., 2019]. For example, an action can be either a predict operation that performs search at the current step, or a read operation that accepts a new input word. Then, we can frame the task of designing the function $g(i)$ as learning a policy to determine which action is taken given a source-side prefix $\mathbf{x}_{\leq j}$ and a target-side prefix $\mathbf{y}_{<i}$. And sequence-to-sequence models can be trained on the states of these action sequences so that they can make better predictions conditioned on part of the input. However, a discussion of training online sequence-to-sequence models lies outside the scope of this section. We refer the reader to the above papers for more details on these methods.

5.4.4 Exact Search

From a formal point of view, we would ideally like to develop a system with no search errors. Although approximate search algorithms have been used successfully in many applications, it is important to study model errors of these systems, and thus to focus on the problem in principle, not just in practice. So developing exact search algorithms for sequence-to-sequence models has long been an interesting topic in NLP research. However, the search problem for a simple word-based machine translation system with n-gram language models has been found to be an NP-hard problem [Knight, 1999]. Much of earlier research formulated the search problem as classical **combinatorial optimization problems**, such as the linear programming problem and the traveling salesman problem, and employed the corresponding solvers [Germann et al., 2004; Zaslavskiy et al., 2009]. Additional research efforts explored exact search algorithms for statistical machine translation systems by using the Lagrangian relaxation technique [Chang and Collins, 2011; Rush and Collins, 2012] and finite-state automata [de Gispert et al., 2010; Allauzen et al., 2014].

Unlike these methods, which are more or less dependent on the integration of n-gram language models into sequence-to-sequence models, the models described in this chapter take a simpler form. We begin with a basic model in which the scoring function $\mathrm{score}(\mathbf{x},\mathbf{y})$ is the log probability $\log\Pr(\mathbf{y}|\mathbf{x})$. Eq. (5.68) tells us that $\log\Pr(\mathbf{y}|\mathbf{x})$ can be written as a sum of word-level log probabilities, and $\log\Pr(\mathbf{y}|\mathbf{x})$ becomes smaller as more target-side words are generated (i.e., a larger n)[11]. In other words, $\log\Pr(\mathbf{y}|\mathbf{x})$ is a monotonic decreasing function with respect to the target-side length n: for any i, we have

$$\begin{aligned} \log\Pr(\mathbf{y}_{\leq i}|\mathbf{x}) &= \log\Pr(\mathbf{y}_{<i}|\mathbf{x}) + \log\Pr(y_i|\mathbf{y}_{<i},\mathbf{x}) \\ &\leq \log\Pr(\mathbf{y}_{<i}|\mathbf{x}) \end{aligned} \tag{5.97}$$

This is also called the **monotonicity** of the scoring function.

Then, by making use of the monotonic nature of model scores, we can develop a heuristic to rule out hypotheses that would never be the best. Let $z_{\mathrm{bestinall}}$ be the global best complete hypothesis we have found. If a new hypothesis has a model score lower than $z_{\mathrm{bestinall}}.score$, then we will not need to extend it. Thus we can explore a region that is significantly smaller than the original search space, without loss of optimality. Note that $z_{\mathrm{bestinall}}.score$ continues to become larger in search. It will be more difficult to find a better hypothesis and more hypotheses will be pruned away as the search process proceeds. See the pseudo-code below for an exact search algorithm of the sequence-to-sequence model of Eq. (5.68).

[11]Consider $\log\Pr(\mathbf{y}|\mathbf{x}) = \sum_{i=1}^{n}\log\Pr(y_i|\mathbf{y}_{<i},\mathbf{x})$. Since $\log\Pr(y_i|\mathbf{y}_{<i},\mathbf{x})$ has a non-positive value, $\log\Pr(\mathbf{y}|\mathbf{x})$ will be smaller or unchanged if n grows.

Algorithm: An Exact Search Algorithm

```
ExactSearch(x)
// Search for the "best" hypothesis by making use of the monotonicity of the
// scoring function (score(x, y) = log Pr(y|x)).
Create a priority queue (max-heap) Queue
Create a hypothesis z_best with z_best.score = -∞
While Queue is not empty do
    z_cur = Queue.Pop()
    If |z_cur.tgt| > n_max, then
        skip z_cur and continue the loop
    For each word v_k ∈ V_y
        z_next = Extend(z_cur, v_k, x)
        bound = z_best.score          // a lower bound on model scores
        If bound < z_next.score, then // admissible pruning
            If z_next.tgt ends with ⟨EOS⟩, then
                z_best = z_next
                bound = z_next.score
            Else
                Add z_next to Queue
Return z_best
```

This is a general algorithm for exact search, and its search efficiency is greatly influenced by the design of the priority queue [Meister et al., 2020]. For example, we can view $\mathrm{score}(\mathbf{x}, \mathbf{y})$ as the priority of each hypothesis in the priority queue, as in a max-heap[12]. Then, the resulting algorithm performs a procedure of breadth-first-like search, since a hypothesis with a shorter target-side sequence is more likely to have a higher model score and to be a top-ranked item in the priority queue. For efficient search, however, we wish to find complete hypotheses as early as possible, such that more unpromising hypotheses can be thrown away in the early stage of search. To do this, we can bias the priority of a hypothesis towards a longer target-side sequence. This provides a **depth-first search** algorithm which is more likely to find complete hypotheses in a shorter time [Stahlberg and Byrne, 2019].

While the exact search algorithm becomes apparent by considering the monotonicity of $\Pr(\mathbf{y}|\mathbf{x})$, in practical systems, as discussed in Section 5.4.1, $\mathrm{score}(\mathbf{x}, \mathbf{y})$ often has a more complex form involving length reward or normalization, and so the monotonic property does not hold. Fortunately, the assumption of monotonicity can be dropped at the expense of slightly relaxing the lower bound on model scores for pruning. Here we define *bound* to be the lowest model score that a hypothesis should have so that it can at best be extended to an equally good hypothesis with z_{best}. For example, consider a simple word reward model described in Eq. (5.69): $\mathrm{Score}(\mathbf{x}, \mathbf{y}) = \log \Pr(\mathbf{y}|\mathbf{x}) + \lambda \cdot n$. For a hypothesis z_{next}, there are at most

[12] We can implement a priority queue using a max-heap.

$n_{\max} - |z_{\text{next}}.tgt|$ words we can predict to obtain a complete hypothesis. Suppose all these $n_{\max} - |z_{\text{next}}.tgt|$ words are predicted with a probability of 1. Then, the model score of the resulting hypothesis (denoted by z_{new}) will be given by

$$\begin{aligned} z_{\text{new}}.score &= z_{\text{next}}.score + \sum_{i=|z_{\text{next}}.tgt|+1}^{n_{\max}} (\log 1 + \lambda) \\ &= z_{\text{next}}.score + \lambda \cdot (n_{\max} - |z_{\text{next}}.tgt|) \end{aligned} \tag{5.98}$$

Using this result, we can define *bound* as

$$bound = z_{\text{best}}.score - \lambda \cdot (n_{\max} - |z_{\text{next}}.tgt|) \tag{5.99}$$

An alternative way to derive the lower bound is to simply consider $n_{\max}$ times of word reward, given by

$$bound = z_{\text{best}}.score - \lambda \cdot n_{\max} \tag{5.100}$$

This is a loose lower bound and leads to less pruning.

In the case of length normalization, we can do this in a similar way. For example, consider the length normalization model $\text{Score}(\mathbf{x}, \mathbf{y}) = \frac{\log \Pr(\mathbf{y}|\mathbf{x})}{n}$, as in Eqs. (5.72-5.73). A lower bound on admissible model scores is given by

$$bound = \frac{\Pr(z_{\text{next}}.tgt|\mathbf{x})}{n_{\max}} \tag{5.101}$$

In practice, such a lower bound can be defined in several different ways to guarantee the optimality of search, depending on which model and search strategy are used in the sequence-to-sequence systems [Huang et al., 2017; Stahlberg and Byrne, 2019].

We can easily apply these lower bounds to the above exact search algorithm by replacing line 9 with Eq. (5.99) or (5.101). As a side effect, the search will explore more hypotheses and thus be much slower.

5.4.5 Differentiable Search

We have addressed the search problem through the introduction of heuristic search algorithms in which we try to minimize the scoring function on a set of sequences of discrete variables. An alternative possibility is to relax these discrete variables to continuous variables and to formulate the problem using the framework of **continuous optimization** [Hoang et al., 2017; Kumar et al., 2021]. While we try to use a consistent notation throughout this book, it is convenient here to introduce some new notation that is slightly different from that adopted in the previous chapters. We will use a vector $\mathbf{y}_i^{\text{w}} \in \{0,1\}^{|V_{\text{y}}|}$ to denote the one-hot representation of y_i. Suppose the output at step i is a distribution over V_{y}, denoted by $\Pr(\cdot|\mathbf{y}_{<i}, \mathbf{x})$. Then, we

can write the log probability of y_i at step i as a dot product of two vectors, like this

$$\begin{aligned}\log\Pr(y_i|\mathbf{y}_{<i},\mathbf{x}) &= \mathbf{y}_i^{\mathrm{w}}\cdot\log\Pr(\cdot|\mathbf{y}_{<i},\mathbf{x}) \\ &= \mathbf{y}_i^{\mathrm{w}}\cdot\log\Pr(\cdot|\mathbf{y}_0^{\mathrm{w}}...\mathbf{y}_{i-1}^{\mathrm{w}},\mathbf{x}) \qquad (5.102)\end{aligned}$$

where $\mathbf{y}_{<i} = y_0...y_{i-1}$ is represented as a sequence of one-hot vectors $\mathbf{y}_0^{\mathrm{w}}...\mathbf{y}_{i-1}^{\mathrm{w}}$. As discussed in Chapter 3, the right-hand side of the above equation means the selection of the entry y_i of the vector $\log\Pr(\cdot|\mathbf{y}_{<i},\mathbf{x})$ (or $\log\Pr(\cdot|\mathbf{y}_0^{\mathrm{w}}...\mathbf{y}_{i-1}^{\mathrm{w}},\mathbf{x})$).

Using this notation, we can write $\log\Pr(\mathbf{y}|\mathbf{x})$ as

$$\begin{aligned}\log\Pr(\mathbf{y}|\mathbf{x}) &= \sum_{i=1}^{n}\log\Pr(y_i|\mathbf{y}_{<i},\mathbf{x}) \\ &= \sum_{i=1}^{n}\mathbf{y}_i^{\mathrm{w}}\cdot\log\Pr(\cdot|\mathbf{y}_0^{\mathrm{w}}...\mathbf{y}_{i-1}^{\mathrm{w}},\mathbf{x}) \qquad (5.103)\end{aligned}$$

Provided we use $\log\Pr(\mathbf{y}|\mathbf{x})$ as the objective function (i.e., $\mathrm{score}(\mathbf{x},\mathbf{y}) = \log\Pr(\mathbf{y}|\mathbf{x})$), the search problem can be formulated as

$$\hat{\mathbf{y}}_0^{\mathrm{w}}...\hat{\mathbf{y}}_n^{\mathrm{w}} = \underset{\mathbf{y}_1^{\mathrm{w}}...\mathbf{y}_n^{\mathrm{w}}}{\arg\max}\sum_{i=1}^{n}\mathbf{y}_i^{\mathrm{w}}\cdot\log\Pr(\cdot|\mathbf{y}_0^{\mathrm{w}}...\mathbf{y}_{i-1}^{\mathrm{w}},\mathbf{x}) \qquad (5.104)$$

This is equivalent to the standard form for inference of sequence-to-sequence models, given by

$$\begin{aligned}\hat{\mathbf{y}} &= \hat{y}_0...\hat{y}_n \\ &= \underset{y_0...y_n}{\arg\max}\Pr(y_0...y_n|\mathbf{x}) \qquad (5.105)\end{aligned}$$

Given Eq. (5.104), we can now relax each one-hot vector to a real-valued vector with a constraint that the sum of all its entries is equal to 1, that is,

$$\mathbf{y}_i^{\mathrm{w}} \in +\mathbb{R}^{|V_{\mathrm{y}}|} \qquad (5.106)$$

$$\text{s.t. } ||\mathbf{y}_i^{\mathrm{w}}||_1 = 1 \qquad (5.107)$$

In this way, $\mathbf{y}_i^{\mathrm{w}}$ can be informally treated as a $|V_{\mathrm{y}}|$-dimensional embedding of y_i, though it has much more dimensions than the usual embeddings used in NLP. Now $\mathbf{y}_i^{\mathrm{w}}$ does not correspond to a specific word in the vocabulary, but describes a distribution over the vocabulary. In Hoang et al. [2017]'s work, $\mathbf{y}_i^{\mathrm{w}}\cdot\log\Pr(\cdot|\mathbf{y}_0^{\mathrm{w}}...\mathbf{y}_{i-1}^{\mathrm{w}},\mathbf{x})$ is called the **expected embedding** under the distribution $\log\Pr(\cdot|\mathbf{y}_0^{\mathrm{w}}...\mathbf{y}_{i-1}^{\mathrm{w}},\mathbf{x})$. What is interesting about this formulation is that Eq. (5.104) in fact defines a "new" task in which we try to maximize a sum of continuous variables (i.e., a sum of n expected embeddings).

We can solve Eq. (5.104) by using the off-the-shelf toolkits in optimization. Since we have a constraint that $\mathbf{y}_i^{\mathrm{w}}$ is a variable in a **simplex**[13], it is straightforward to apply general

[13] Simplex is a term used in geometry. In a Euclidean space, a k-simplex is a k-dimensional polytope described

constrained optimization algorithms to this problem. An alternative way is to use algorithms that are designed to solve the optimization problem with simplex constraints. The details of these algorithms can be found in many books on optimization.

A third choice of solving Eq. (5.104) is to formulate the constraints in the objective function explicitly and to use gradient descent methods to optimize this function. For example, Hoang et al. [2017] modify Eq. (5.104) and obtain a new form for optimization

$$\hat{\mathbf{y}}_0^{\mathrm{w}}...\hat{\mathbf{y}}_n^{\mathrm{w}} = \underset{\mathbf{y}_1^{\mathrm{w}}...\mathbf{y}_n^{\mathrm{w}}}{\arg\max} \sum_{i=1}^{n} \mathrm{Softmax}(\mathbf{y}_i^{\mathrm{w}}) \cdot \log \Pr(\cdot|\mathbf{y}_0^{\mathrm{w}}...\mathbf{y}_{i-1}^{\mathrm{w}}, \mathbf{x}) \tag{5.111}$$

Here we remove the simplex constraint from $\mathbf{y}_i^{\mathrm{w}}$, and impose it on a new output that is produced by a $\mathrm{Softmax}$ function.

Once we have obtained the optimal sequence $\hat{\mathbf{y}}_0^{\mathrm{w}}...\hat{\mathbf{y}}_n^{\mathrm{w}}$, we need to map each $\mathbf{y}_i^{\mathrm{w}}$ to a unique word. A simple method is to take the word corresponding to the entry of $\mathbf{y}_i^{\mathrm{w}}$ with the largest value. However, this may break the optimality of the solution because the condition $\mathbf{y}_0^{\mathrm{w}}...\mathbf{y}_{i-1}^{\mathrm{w}}$ is changed when these variables are discretized. A more practical method is to perform optimization to predict the next word given a prefix, say, we fix $\mathbf{y}_0^{\mathrm{w}}...\mathbf{y}_{i-1}^{\mathrm{w}}$ to the one-hot representations of the optimal prefix, and maximize $\sum_{k=i}^{n} \mathbf{y}_k^{\mathrm{w}} \cdot \log \Pr(\cdot|\mathbf{y}_0^{\mathrm{w}}...\mathbf{y}_{k-1}^{\mathrm{w}}, \mathbf{x})$. Then, we select the best word at position i and move on to the next position.

So far we have assumed that the search objective is derived from the log probability $\log \Pr(\mathbf{y}|\mathbf{x})$ and the length of the output is given in advance. To have a search over sequences with different lengths, we can repeat the above optimization procedure for every $n \in [1, n_{\max}]$, and select the sequence with the maximum score. This also makes it easy to introduce length normalization and reward into search. We can ignore the length bias issue in each search with a fixed n, and add the length models after optimization, that is, we leave the search objective unchanged, but, in the final step, we select the best sequence in a set of candidates with different n in terms of $\mathrm{score}(\mathbf{x}, \mathbf{y})$.

5.4.6 Hypothesis Diversity

Multiple outputs are often required when one wants to rescore these outputs and/or interact with the system. One of the most widely used methods is to use beam search to generate a number of top-ranked hypotheses. For example, we can simply view the elements of Heap as the k-best hypotheses in beam search (see Section 5.4.2). However, this approach suffers from the problem that there is often little difference among the hypotheses in the beam, and

by a set of $k+1$ independent points $\{\mathbf{p}_0, \mathbf{p}_1, ..., \mathbf{p}_k\}$. This polytope is defined as a set of points

$$P_{k\text{-simplex}} = \{a_0 \cdot \mathbf{p}_0 + a_1 \cdot \mathbf{p}_1 + ... + a_k \cdot \mathbf{p}_k\} \tag{5.108}$$

where

$$\sum_{i=0}^{k} a_i = 1 \tag{5.109}$$

$$a_i \geq 0 \quad \text{for any } i \in [0, k] \tag{5.110}$$

Rank	Output
1	Manuela Arbelaez accidentally revealed the correct answer to a guessing game for a new Hyundai Sonata. Host Drew Carey couldn't stop laughing. It's been a busy week for "The Price Is Right" when Bob Barker, 91, showed up to run his old show.
2	Manuela Arbelaez accidentally revealed the correct answer to a guessing game for a new Hyundai Sonata. Host Drew Carey couldn't stop laughing. It's been a busy week for "The Price Is Right" when Bob Barker showed up to run his old show.
3	Manuela Arbelaez accidentally revealed the correct answer to a guessing game for a new Hyundai Sonata. Host Drew Carey couldn't stop laughing. It's been a busy week for "The Price Is Right" when Bob Barker, 91, showed up to run the show.
4	Manuela Arbelaez accidentally revealed the correct answer to a guessing game for a new Hyundai Sonata. Host Drew Carey couldn't stop laughing. It's been a busy week for "The Price Is Right" when Bob Barker, 91, showed up to run his show.

Table 5.3: 4-best outputs of a text summarization system on a sample in the CNN/Daily Mail dataset (beam size $= 4$). We see that these texts differ only by a few words.

it is difficult to figure out which one is better though more options are available to users. Table 5.3 shows the 4-best outputs of a text summarization system. We see that these texts are fairly similar to each other. One reason for this phenomenon is that diverse hypotheses, though probably with high model scores when completed, will be pruned away if they are low-ranked in some stages of beam search. From a modeling perspective, we can interpret this as a problem with the locally normalized models that we use here: every prediction is made on an intermediate step of search, and there is no way for the following steps to escape if the prefix is fixed [Murray and Chiang, 2018].

One approach to improving the hypothesis diversity is to give penalties to cases where the hypotheses in the beam are less diverse [Li and Jurafsky, 2016; Vijayakumar et al., 2018]. A simple example of such objective functions is given by

$$\mathrm{score}_d(\mathbf{x},\mathbf{y}) \quad = \quad \mathrm{score}(\mathbf{x},\mathbf{y}) - \lambda \cdot dp \tag{5.112}$$

It combines the original model score $\mathrm{score}(\mathbf{x},\mathbf{y})$ and a diversity penalty dp. dp can be defined in a few different ways. An idea is to penalize hypotheses that are close in the search tree. For example, one can define dp as the rank of a hypothesis in the set of its siblings that are extended from the same parent hypothesis, and so the beam can spread its members over a larger region of the space of hypotheses [Li and Jurafsky, 2016]. Another way to introduce diversity measures is to consider the differences between the target-side sequences of the hypotheses in the beam. For example, we can define dp as the average string similarity between a given hypothesis and other hypotheses in the beam [Xiao et al., 2013].

The above idea can also be expressed as constraints imposed on the search procedure. For example, we can constrain the beam to include only the hypotheses that are rooted at different parents in the last step [Boulanger-Lewandowski et al., 2013]. More precisely, for each hypothesis $z_{\mathrm{cur}} \in Z_{i-1}$, we seek the best next-step hypothesis by

$$\hat{z}_{\mathrm{next}} = \underset{z_{\mathrm{next}} \in \mathrm{Extend}(z_{\mathrm{cur}}, V_{\mathrm{y}})}{\arg\max} \Pr(z_{\mathrm{next}}.tgt|\mathbf{x}) \tag{5.113}$$

The hypothesis $\hat{z}_{\mathrm{next}}$ is then added to Z_i. Note that this is essentially a **sub-space method** that divides a space of hypotheses into sub-spaces of hypotheses, and collects results over these sub-spaces. An intuition behind this method is that different sub-spaces can describe different aspects of the problem, and so we can have diverse solutions.

Another approach to addressing the diversity issue is to perturb beam search by introducing randomly generated hypotheses into the beam [Holtzman et al., 2020; Wiher et al., 2022]. One common way to do this is to choose some random words for extending a hypothesis, and to add the extended hypotheses to the beam. In general, these words can be sampled from the distribution $\Pr(\cdot|\mathbf{y}_{<i}, \mathbf{x})$ over the entire vocabulary or its subset. Randomness can also be added to the inputs of a system at test time. For example, one can express an input word as a linear combination of its original embedding and the embedding of a word of a random sequence drawn from the training data [Li et al., 2021]. In problems having many local minima, adding random "noise" to search procedures is generally helpful, as we can explore more diverse hypotheses and prevent the systems from getting stuck in certain regions of the search space.

Instead of performing search using a single system, we can use multiple systems to obtain diverse hypotheses. These systems can be built on either different architectures or different hidden structures/configurations [He et al., 2018; Shen et al., 2019; Wu et al., 2020; Sun et al., 2020]. Although methods of this type do not fall under the search framework that we have been discussing, combining the results from multiple systems is generally helpful. The following section will present a discussion on this issue.

5.4.7 Combining Multiple Models

From a machine learning point of view, **ensembling** are methods for addressing modeling issues, not search issues. In this subsection, we discuss these methods because their implementations typically require modifications to the search modules, and we can gain some insight into the resulting system by viewing it from the search perspective.

In machine learning, ensemble methods aim to make better predictions by combining predictions of a number of **constituent systems** or **component systems**. The problem of combining multiple systems has been discussed extensively in times when statistical models emerged in NLP, and is sometimes called **system combination methods** for emphasizing its practical use. For sequence-to-sequence models discussed here, a widely used form of system combination is an average of predictions [Sutskever et al., 2014]. Suppose we have K sequence-to-sequence models that have been trained. The log probability of the target-side word y_i given its left context $\mathbf{y}_{<i}$ and the source-side sequence $\mathbf{x}$ can be defined by using the

geometric average

$$\log \Pr(y_i|\mathbf{y}_{<i}, \mathbf{x}) = \frac{1}{K}\sum_{k=1}^{K} \log \Pr_k(y_i|\mathbf{y}_{<i}, \mathbf{x}) \tag{5.114}$$

or alternatively by using the arithmetic average

$$\log \Pr(y_i|\mathbf{y}_{<i}, \mathbf{x}) = \log \frac{1}{K}\sum_{k=1}^{K} \Pr_k(y_i|\mathbf{y}_{<i}, \mathbf{x}) \tag{5.115}$$

where $\Pr_k(y_i|\mathbf{y}_{<i}, \mathbf{x})$ is the output of the k-th component system. These forms are so simple that one can implement them for any sequence-to-sequence models without significant modifications to existing systems, and they have been used as the basis of many successful systems in various evaluation tasks [Barrault et al., 2020; Akhbardeh et al., 2021].

A problem with prediction averaging is that all the component systems are required to follow the same basic form of modeling (see Eq. (5.68)) and we need to have access to the probabilities $\{\Pr_k(y_i|\mathbf{y}_{<i}, \mathbf{x})\}$. When we have only a set of black-box systems in hand, we need to perform sequence ensembling. A common idea is to vote from the ensemble of the sequences produced by the component systems. For example, one of the simplest ways to do this is **hypothesis selection** [Hildebrand and Vogel, 2008], in which we simply select the "best" sequence from the ensemble using some criterion. An alternative way of sequence ensembling is to regenerate a new sequence differing from any of the original sequences [Matusov et al., 2006; Rosti et al., 2007]. This typically requires a model that represents the sequences into a compact representation (such as a lattice), as well as an additional search pass by which we can find the best output in this new representation of hypotheses (such as lattice search and rescoring) [Deoras et al., 2011; Stahlberg et al., 2016; Khayrallah et al., 2017].

Note that the ensembling of sequence-to-sequence models is related to the diversity issue discussed in the previous subsection. It is often thought that component systems need to be diverse for a better ensembling result, and so we need to build these systems in some way that we can make them different [Sutskever et al., 2014; Zhou et al., 2017]. One of the most popular methods is **checkpoint ensembling**. It takes a number of copies of a model at different checkpoints during training, and combines these model copies via prediction averaging. This method can be useful for alleviating the overfitting problem in practice. Also, different models can be created from a base model under different settings. For example, we can build models with different numbers of parameters on the basis of a backbone model. A more general approach is to take models based on different architectures, although this is at the expense of more development effort.

Another way to view sequence ensembling is that it provides a two-pass search scheme. In the first pass of search, multiple systems are used to perform inference individually. Each of these systems has its own bias for modeling and search, and explores different regions of the search space. A hypothesis explored by one system might not be seen and evaluated by another system. The result of this pass is a diverse ensemble of hypotheses that are "optimal" from some perspectives. In the second pass of search, we use this ensemble to define a new space of hypotheses, and use a fine-grained model to search for the final result.

5.4.8 More Search Objectives

In this subsection, we consider more objective functions that can be applied to the search problem.

1. Search with Future Scores

Most of the algorithms described in this subsection can be viewed as some optimizations of best-first search algorithms [Meister et al., 2020]. As another example of best-first search, **A* search** is widely considered to be a good solution to the general search problem. Vanilla A* search requires that all states of search are sorted in every search step, which is intractable in our problems. We therefore still consider beam search and greedy search for our discussion, but use an A* search-like objective function instead. Specifically, given a search state $(\mathbf{x}, \mathbf{y}_{\leq i})$, the A* search-like objective function can be defined as

$$\mathrm{score}_{\mathrm{A}*}(\mathbf{x}, \mathbf{y}_{\leq i}) = g(\mathbf{x}, \mathbf{y}_{\leq i}) + h(\mathbf{x}, \mathbf{y}_{\leq i}) \tag{5.116}$$

Here $g(\mathbf{x}, \mathbf{y}_{\leq i})$ is the reward of the path from the start state to $(\mathbf{x}, \mathbf{y}_{\leq i})$, and $h(\mathbf{x}, \mathbf{y}_{\leq i})$ is the estimated reward of the "optimal" path from $(\mathbf{x}, \mathbf{y}_{\leq i})$ to the final goal. Because $g(\mathbf{x}, \mathbf{y}_{\leq i})$ and $h(\mathbf{x}, \mathbf{y}_{\leq i})$ can have arbitrary forms, this framework is very general. For example, if we define

$$g(\mathbf{x}, \mathbf{y}_{\leq i}) = \mathrm{score}(\mathbf{x}, \mathbf{y}_{\leq i}) \tag{5.117}$$

$$h(\mathbf{x}, \mathbf{y}_{\leq i}) = 0 \tag{5.118}$$

then $\mathrm{score}_{\mathrm{A}*}(\mathbf{x}, \mathbf{y})$ is exactly the same as the objective functions discussed previously.

To make full use of this formulation, it seems natural to seek a function of future reward or future cost. Ideally, we would like $h(\mathbf{x}, \mathbf{y}_{\leq i})$ to be able to compute how much additional reward we can obtain if we extend $(\mathbf{x}, \mathbf{y}_{\leq i})$ to the best complete hypothesis. This is, however, intractable because we need to explore all the hypotheses extended from $(\mathbf{x}, \mathbf{y}_{\leq i})$ and find the best one. It is common practice to use a computationally cheaper model analogous to the real future reward model. Conventional approaches rely on heuristics to define $h(\mathbf{x}, \mathbf{y}_{\leq i})$ [Koehn et al., 2007], such as estimating the weights of the words that could be further generated. These heuristics can be generalized to the knowledge of the model design of sequence-to-sequence systems [He et al., 2017; Zheng et al., 2018]. A more general approach is to use a value-based treatment of the problem [Ren et al., 2017; Li et al., 2017; Leblond et al., 2021]. We can develop a policy that learns to predict the distribution of y_i given $\mathbf{x}$ and $\mathbf{y}_{<i}$, and a **value function** for this policy that learns to predict future rewards. Eq. (5.116) can therefore be interpreted as a linear combination of the policy score of $(\mathbf{x}, \mathbf{y}_{\leq i})$ and the corresponding value. Such a treatment of search objectives falls into the framework of **value-based search**, and has been successfully employed in reinforcement learning [Silver et al., 2017].

2. Search with Language Models

For a long time, language models played an important role in text generation tasks. For example, statistical machine translation systems and automatic speech recognition systems typically rely on large n-gram language models to produce fluent texts. While modern sequence-to-sequence models are not required to have separate language models, applying them to sequence-to-sequence search still makes intuitive sense for machine translation and related problems.

Following the convention that a language model can be treated as a feature of a log-linear (or linear) model [Och and Ney, 2002], the language model-augmented objective can be defined as

$$\mathrm{score}_{\mathrm{lm}}(\mathbf{x},\mathbf{y}) \quad = \quad \log \Pr(\mathbf{y}|\mathbf{x}) + \lambda \cdot \log \Pr(\mathbf{y}) \tag{5.119}$$

This formulation does not involve length reward and normalization terms, but either of them can be easily used as an additional feature of the model. In general, the language model $\Pr(\mathbf{y})$ is trained solely on target-side sequences, enabling the use of large-scale monolingual data in sequence-to-sequence models [Gulcehre et al., 2017]. Interestingly, it has been found that current sequence-to-sequence models are strong language models themselves if they are trained sufficiently, and a better way to make use of target-side data might be to use it to create synthetic data, called **data augmentation**. An example of this is **back translation** in which we use a backward translation system to translate target-side sentences to source-side sentences, and then use this synthetic bilingual data as additional data for training a forward translation system [Sennrich et al., 2016; Edunov et al., 2018]. In many tasks, such a simple method can achieve significant improvements in translation quality, but this result questions the necessity of using additional language models in neural machine translation.

Note that the model of Eq. (5.119) depends on our choice for the coefficient λ. For machine translation, we are usually interested in a positive value of λ so that our system can produce more fluent texts. By contrast, a negative value of λ means that we want some output that is less frequent. For example, if $\lambda = -1$, then Eq. (5.119) can be written as the point-wise mutual information of $\mathbf{x}$ and $\mathbf{y}$

$$\begin{aligned} \mathrm{score}_{\mathrm{lm}}(\mathbf{x},\mathbf{y}) &= \log \Pr(\mathbf{y}|\mathbf{x}) - \log \Pr(\mathbf{y}) \\ &= \log \frac{\Pr(\mathbf{x},\mathbf{y})}{\Pr(\mathbf{x}) \cdot \Pr(\mathbf{y})} \end{aligned} \tag{5.120}$$

This scoring function has been shown to be useful for generating more diverse outputs for neural conversation systems [Li et al., 2016].

3. Minimum Bayes Risk Search

So far, our discussion of search objectives has focused on the use of the decision rule of choosing the highest score hypothesis, called **maximum a posteriori** (**MAP**) search[14]. An

[14] In statistics, MAP is a method for inference of the parameters of a statistical model. Suppose we have a model that describes the distribution of a variable x and the model is parameterized by θ. MAP seeks the optimal value of

assumption behind this method is that the posterior probability $\Pr(\mathbf{y}|\mathbf{x})$ (or the model score $\mathrm{score}(\mathbf{x},\mathbf{y})$) correlates with the true quality of outputs. In practice, this assumption leads to several useful properties, e.g., the search system is easy to implement, and the objective of search is consistent with that of training. However, there are some shortcomings with MAP search, which causes researchers to consider more powerful methods. One problem with MAP search is that the objective does not reflect the way one evaluates the system. The metrics used in end-to-end evaluation of a system may have very different forms from $\Pr(\mathbf{y}|\mathbf{x})$. A second problem is that MAP is just a special case of the Bayesian treatment of determining posterior probabilities. It provides a point estimate of θ with no uncertainty measure, and is sometimes overconfident. In some applications, sequence-to-sequence models spread too much probability mass across many different hypotheses [Ott et al., 2018a], and MAP may not describe the major portion of the distribution.

Here we consider **minimum Bayes risk** (**MBR**) search that provides ways to introduce evaluation measures into search, as well as ways to make use of the distributions over hypotheses. The MBR method assumes a risk function on a pair of sequences, denoted by $R(\mathbf{y},\mathbf{y}_r)$. It computes the cost of replacing $\mathbf{y}_r$ with $\mathbf{y}$ in terms of some evaluation metric. For example, we can define the risk score to be $1-\mathrm{BLEU}$ for machine translation. Then, the risk for $\mathbf{y}$ on a set of sequences Ω is given by the expectation of $R(\mathbf{y},\mathbf{y}_r)$ with respect to the distribution $\Pr(\mathbf{y}_r|\mathbf{x})$

$$\begin{aligned} \mathrm{Risk}(\mathbf{y}) &= \mathbb{E}_{\mathbf{y}_r\sim\Pr(\mathbf{y}_r|\mathbf{x})} R(\mathbf{y},\mathbf{y}_r) \\ &= \sum_{\mathbf{y}_r\in\Omega} R(\mathbf{y},\mathbf{y}_r)\cdot\Pr(\mathbf{y}_r|\mathbf{x}) \end{aligned} \tag{5.124}$$

However, the summation over all possible target-side sequences is computationally infeasible. We therefore define Ω to be the k-best outputs or sampled outputs of a system [Eikema and Aziz, 2020], denoted by Ω_{system}. Then, we take $\mathrm{score}(\mathbf{x},\mathbf{y}) = -\mathrm{Risk}(\mathbf{y})$ and obtain the

θ by maximizing the probability of θ given x, written as

$$\hat{\theta}_{\mathrm{MAP}} = \arg\max_{\theta} \Pr(\theta|x) \tag{5.121}$$

$\hat{\theta}_{\mathrm{MAP}}$ is also called the **mode** of the posterior distribution of θ. For the MAP search problem here, we simply denote θ by $\mathbf{y}$ and seek the mode of $\Pr(\mathbf{y}|\mathbf{x})$.

As a Bayesian method, we can re-express the above equation using the Bayes' rule

$$\begin{aligned} \hat{\theta}_{\mathrm{MAP}} &= \arg\max_{\theta} \frac{\Pr(x|\theta)\cdot\Pr(\theta)}{\Pr(x)} \\ &= \arg\max_{\theta} \Pr(x|\theta)\cdot\Pr(\theta) \end{aligned} \tag{5.122}$$

where θ is treated as a variable having a prior distribution $\Pr(\theta)$.

By contrast, MLE directly maximizes the likelihood function $\Pr(x|\theta)$

$$\hat{\theta}_{\mathrm{MLE}} = \arg\max_{\theta} \Pr(x|\theta) \tag{5.123}$$

Thus, the MAP result can be viewed as an estimation of θ that considers both MLE of x given θ and the prior of θ. Note that MAP and MLE will be equivalent if $\Pr(\theta)$ is a uniform distribution.

following objective for MBR search

$$\begin{aligned}\hat{\mathbf{y}} &= \arg\max_{\mathbf{y}} -\mathrm{Risk}(\mathbf{y}) \\ &= \arg\min_{\mathbf{y}} \sum_{\mathbf{y}_r \in \Omega_{\text{system}}} R(\mathbf{y}, \mathbf{y}_r) \cdot \Pr(\mathbf{y}_r|\mathbf{x}) \end{aligned} \tag{5.125}$$

This model is very general and applies to a wide range of NLP problems in which one needs to search for an optimal hypothesis in a large set of candidates [Goodman, 1996; Goel and Byrne, 2000; Kumar and Byrne, 2004]. It allows for flexible forms of risk functions, for instance having various factors considered in evaluating hypotheses. MBR search has recently been of interest to NLP researchers as they are found to be effective in eliminating the biases caused by MAP search [Müller and Sennrich, 2021; Freitag et al., 2022]. In addition to providing a formulation of search objectives, MBR methods can be used for training sequence-to-sequence models, and are thought to be solutions to the discrepancy issue between objectives of training and evaluation [Shen et al., 2016].

5.5 Summary

In this chapter, we attempted to provide an overview of sequence-to-sequence modeling which can serve as the basis for many NLP systems. Sequence-to-sequence modeling is a very rich area of research, and has been widely discussed in different disciplines, even beyond NLP. This chapter is not a review of all the literature on this subject (this would be a big project), but focuses on some of the core methods and ideas. We started with an introduction of sequence-to-sequence problems, as well as the encoder-decoder architecture which lays the foundations for most of the state-of-the-art sequence-to-sequence systems. As an illustration of the application of this architecture, we considered the problem of neural machine translation, and built a simple neural machine translation model using the basic knowledge we have learned so far.

We also presented the attention mechanism and a series of refinements. If we look back to the past few years, we will find that exploring attention models is the next natural step in developing sequence-to-sequence models. While these models are well known for their application and impressive performance in machine translation, they have dominated the NLP community. There is also great interest in attention models in some other sub-fields of AI, such as computer vision [Borji and Itti, 2012; Xu et al., 2015; Jaderberg et al., 2015] and speech processing [Chorowski et al., 2015; Chan et al., 2016; Bahdanau et al., 2016]. The result is that the past few years were an exciting time for people in these areas.

Sequence-to-sequence models are so successful that we try to put everything in the same pocket. Not only have we developed powerful sequence-to-sequence models to deal with very general problems, but current research is forced to be unifying. An example is that Transformer, a self-attention-based sequence-to-sequence model, has become one of the fundamental models for many tasks ranging over different types of data, from textual to visual and acoustic data. It can even be extended to deal with multimodal problems which are sometimes more challenging.

This makes things more interesting and exciting: an improvement to one model can be used to improve systems in a variety of tasks. And we are seeing a significant change in our research paradigm in which the NLP and machine learning fields are marrying and results in NLP research are becoming more influential. However, on the other side of the coin is that we are making much room for some of the problems but leaving less room for the others. In recent NLP conferences, we can see many, many papers talking about how to train big sequence-to-sequence models and apply them to different text generation tasks, but there are a relatively small number of papers on parsing. There have always been debates on this over the past few decades, for example, what and how much prior knowledge do we need to build an NLP system? [Church, 2011; See, 2018] Getting involved in such debates is simply beyond the discussions in this chapter. Fortunately, NLP research promises to continue to be diverse and active, and we can always hear and learn from both sides of the debates. For example, there are interesting findings that the neural sequence models can learn some linguistic properties from data, and linguistic structures can help system design. In Chapter 6, we will see a few examples.

The "bias" of research focus also exists on the machine learning side of problem-solving. For example, for sequence-to-sequence problems discussed here, recent years have witnessed a drastic increase of interest in model design and training methods, but only a relatively small group of people discuss the search problem. While search is a classical problem in AI and plays an important role in practical systems [Russell and Norvig, 2010], it is even not discussed in recent tutorials and surveys in NLP. This motivates us to write a section on this subject so that we can have a more complete picture of the problem. However, our general discussion does not cover all aspects of the search problem. A topic we left out is efficiency [Birch et al., 2018; Heafield et al., 2021]. While this chapter includes some discussions on the efficiency issue, such as stopping criteria of search algorithms, efficient methods are a wide-ranging topic and are generally dependent on model architectures. A more detailed discussion of them can be found in Chapter 6. Another topic that one may be interested in is **constrained search** in which constraints are imposed on the search process [Hokamp and Liu, 2017; Anderson et al., 2017]. In general, these constraints come from our prior knowledge or interactions with users. For example, constrained search has been used to enforce term translation constraints on machine translation [Hasler et al., 2018; Post and Vilar, 2018].

One last note on limitations of this chapter. The formulation of the general sequence-to-sequence problem described here is based on the left-to-right factorization of $\Pr(\mathbf{y}|\mathbf{x})$, resulting in an autoregressive model. One limitation of this formulation is that each prediction at some step depends only on the preceding words, and so the model cannot access the right context. To make use of the right context of a word, a simple approach is to build another model that performs right-to-left generation. The left-to-right and right-to-left models can then be combined to generate a better output sequence [Liu et al., 2016a; Hoang et al., 2017; Zhang et al., 2018; 2020a]. An alternative approach is given by **non-autoregressive generation** or **non-autoregressive decoding** in which the constraint of autoregressive generation is removed and each word prediction is conditioned on the global context [Gu et al., 2018; Ghazvininejad et al., 2019; Lee et al., 2020]. A nice property of non-autoregressive generation is the possibility of system speed-up, since all the words in a sequence can be generated in parallel and we can do this efficiently using GPUs.

Bibliography

[Akhbardeh et al., 2021] Farhad Akhbardeh, Arkady Arkhangorodsky, Magdalena Biesialska, Ondřej Bojar, Rajen Chatterjee, Vishrav Chaudhary, Marta R. Costa-jussa, Cristina España-Bonet, Angela Fan, Christian Federmann, Markus Freitag, Yvette Graham, Roman Grundkiewicz, Barry Haddow, Leonie Harter, Kenneth Heafield, Christopher Homan, Matthias Huck, Kwabena Amponsah-Kaakyire, Jungo Kasai, Daniel Khashabi, Kevin Knight, Tom Kocmi, Philipp Koehn, Nicholas Lourie, Christof Monz, Makoto Morishita, Masaaki Nagata, Ajay Nagesh, Toshiaki Nakazawa, Matteo Negri, Santanu Pal, Allahsera Auguste Tapo, Marco Turchi, Valentin Vydrin, and Marcos Zampieri. Findings of the 2021 conference on machine translation (WMT21). In *Proceedings of the Sixth Conference on Machine Translation*, pages 1–88, 2021.

[Allauzen et al., 2014] Cyril Allauzen, Bill Byrne, Adrià de Gispert, Gonzalo Iglesias, and Michael Riley. Pushdown automata in statistical machine translation. *Computational Linguistics*, 40(3): 687–723, 2014.

[Anderson et al., 2017] Peter Anderson, Basura Fernando, Mark Johnson, and Stephen Gould. Guided open vocabulary image captioning with constrained beam search. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 936–945, 2017.

[Bahdanau et al., 2014] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[Bahdanau et al., 2016] Dzmitry Bahdanau, Jan Chorowski, Dmitriy Serdyuk, Philemon Brakel, and Yoshua Bengio. End-to-end attention-based large vocabulary speech recognition. In *2016 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 4945–4949. IEEE, 2016.

[Barrault et al., 2020] Loïc Barrault, Magdalena Biesialska, Ondřej Bojar, Marta R. Costa-jussà, Christian Federmann, Yvette Graham, Roman Grundkiewicz, Barry Haddow, Matthias Huck, Eric Joanis, Tom Kocmi, Philipp Koehn, Chi-kiu Lo, Nikola Ljubešić, Christof Monz, Makoto Morishita, Masaaki Nagata, Toshiaki Nakazawa, Santanu Pal, Matt Post, and Marcos Zampieri. Findings of the 2020 conference on machine translation (WMT20). In *Proceedings of the Fifth Conference on Machine Translation*, pages 1–55, 2020.

[Birch et al., 2018] Alexandra Birch, Andrew Finch, Minh-Thang Luong, Graham Neubig, and Yusuke Oda. Findings of the second workshop on neural machine translation and generation. In *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*, pages 1–10, 2018.

[Borji and Itti, 2012] Ali Borji and Laurent Itti. State-of-the-art in visual attention modeling. *IEEE transactions on pattern analysis and machine intelligence*, 35(1):185–207, 2012.

[Boulanger-Lewandowski et al., 2013] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Audio chord recognition with recurrent neural networks. In *Proceedings of 14th*

International Society for Music Information Retrieval Conference, 2013.

[Brown et al., 1993] Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311, 1993.

[Buckman et al., 2016] Jacob Buckman, Miguel Ballesteros, and Chris Dyer. Transition-based dependency parsing with heuristic backtracking. In *Proceedings of the 2016 Conference on empirical methods in natural language processing*, pages 2313–2318, 2016.

[Casacuberta et al., 2009] Francisco Casacuberta, Jorge Civera, Elsa Cubel, Antonio L Lagarda, Guy Lapalme, Elliott Macklovitch, and Enrique Vidal. Human interaction for high-quality machine translation. *Communications of the ACM*, 52(10):135–138, 2009.

[Chan et al., 2016] William Chan, Navdeep Jaitly, Quoc Le, and Oriol Vinyals. Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *2016 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 4960–4964. IEEE, 2016.

[Chang, 1967] Wing-Tsit Chang. *Reflections on things at hand*. Columbia University Press, 1967.

[Chang and Collins, 2011] Yin-Wen Chang and Michael Collins. Exact decoding of phrase-based translation models through lagrangian relaxation. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 26–37, 2011.

[Chaudhari et al., 2021] Sneha Chaudhari, Varun Mithal, Gungor Polatkan, and Rohan Ramanath. An attentive survey of attention models. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 12(5):1–32, 2021.

[Chen et al., 2018] Kehai Chen, Rui Wang, Masao Utiyama, Eiichiro Sumita, and Tiejun Zhao. Syntax-directed attention for neural machine translation. In *Proceedings of the AAAI conference on artificial intelligence*, 2018.

[Chen et al., 2020] Yen-Chun Chen, Linjie Li, Licheng Yu, Ahmed El Kholy, Faisal Ahmed, Zhe Gan, Yu Cheng, and Jingjing Liu. Uniter: Universal image-text representation learning. In *Proceedings of European conference on computer vision*, pages 104–120, 2020.

[Chiang, 2005] David Chiang. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43rd annual meeting of the association for computational linguistics (acl'05)*, pages 263–270, 2005.

[Chiang, 2007] David Chiang. Hierarchical phrase-based translation. *computational linguistics*, 33(2): 201–228, 2007.

[Chiu and Raffel, 2018] Chung-Cheng Chiu and Colin Raffel. Monotonic chunkwise attention. In *Proceedings of the 8th International Conference on Learning Representations ICLR*, 2018.

[Cho and Esipova, 2016] Kyunghyun Cho and Masha Esipova. Can neural machine translation do simultaneous translation? *arXiv preprint arXiv:1606.02012*, 2016.

[Cho et al., 2014] Kyunghyun Cho, Bart van Merriënboer, Çağlar Guùlçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, 2014.

[Chorowski and Jaitly, 2017] Jan Chorowski and Navdeep Jaitly. Towards better decoding and language model integration in sequence to sequence models. *Proc. Interspeech 2017*, pages 523–527, 2017.

[Chorowski et al., 2015] Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. *Advances in neural information processing systems*, 28, 2015.

[Church, 2011] Kenneth Church. A pendulum swung too far. *Linguistic Issues in Language Technology*, 6, 2011.

[Cohn et al., 2016] Trevor Cohn, Cong Duy Vu Hoang, Ekaterina Vymolova, Kaisheng Yao, Chris Dyer, and Gholamreza Haffari. Incorporating structural alignment biases into an attentional neural translation model. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 876–885, 2016.

[de Gispert et al., 2010] Adrià de Gispert, Gonzalo Iglesias, Graeme Blackwood, Eduardo R. Banga, and William Byrne. Hierarchical phrase-based translation with weighted finite-state transducers and shallow-n grammars. *Computational linguistics*, 36(3):505–533, 2010.

[Deoras et al., 2011] Anoop Deoras, Tomáš Mikolov, and Kenneth Church. A fast re-scoring strategy to capture long-distance dependencies. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1116–1127, 2011.

[Dyer et al., 2013] Chris Dyer, Victor Chahuneau, and Noah A Smith. A simple, fast, and effective reparameterization of ibm model 2. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 644–648, 2013.

[Edunov et al., 2018] Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. Understanding back-translation at scale. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 489–500, 2018.

[Eikema and Aziz, 2020] Bryan Eikema and Wilker Aziz. Is map decoding all you need? the inadequacy of the mode in neural machine translation. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 4506–4520, 2020.

[Feng et al., 2016] Shi Feng, Shujie Liu, Nan Yang, Mu Li, Ming Zhou, and Kenny Zhu. Improving attention modeling with implicit distortion and fertility for machine translation. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 3082–3092, 2016.

[Freitag and Al-Onaizan, 2017] Markus Freitag and Yaser Al-Onaizan. Beam search strategies for neural machine translation. In *Proceedings of the First Workshop on Neural Machine Translation*, pages 56–60, 2017.

[Freitag et al., 2022] Markus Freitag, David Grangier, Qijun Tan, and Bowen Liang. High quality rather than high model probability: Minimum bayes risk decoding with neural metrics. *Transactions of the Association for Computational Linguistics*, 10:811–825, 2022.

[Garg et al., 2019] Sarthak Garg, Stephan Peitz, Udhyakumar Nallasamy, and Matthias Paulik. Jointly learning to align and translate with transformer models. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4453–4462, 2019.

[Germann et al., 2004] Ulrich Germann, Michael Jahr, Kevin Knight, Daniel Marcu, and Kenji Yamada. Fast and optimal decoding for machine translation. *Artificial Intelligence*, 154(1-2):127–143, 2004.

[Ghazvininejad et al., 2019] Marjan Ghazvininejad, Omer Levy, Yinhan Liu, and Luke Zettlemoyer.

Mask-predict: Parallel decoding of conditional masked language models. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6112–6121, 2019.

[Goel and Byrne, 2000] Vaibhava Goel and William J Byrne. Minimum bayes-risk automatic speech recognition. *Computer Speech & Language*, 14(2):115–135, 2000.

[Goodman, 1996] Joshua Goodman. Parsing algorithms and metrics. In *34th Annual Meeting of the Association for Computational Linguistics*, pages 177–183, 1996.

[Graves et al., 2014] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[Grissom II et al., 2014] Alvin Grissom II, He He, Jordan Boyd-Graber, John Morgan, and Hal Daumé III. Don't until the final verb wait: Reinforcement learning for simultaneous machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1342–1352, 2014.

[Gu et al., 2017] Jiatao Gu, Graham Neubig, Kyunghyun Cho, and Victor O.K. Li. Learning to translate in real-time with neural machine translation. In *Proceedings of the European Chapter of the Association for Computational Linguistics (EACL) Conference, 2017*, 2017.

[Gu et al., 2018] Jiatao Gu, James Bradbury, Caiming Xiong, Victor O.K. Li, and Richard Socher. Non-autoregressive neural machine translation. In *Proceedings of International Conference on Learning Representations*, 2018.

[Gulcehre et al., 2017] Caglar Gulcehre, Orhan Firat, Kelvin Xu, Kyunghyun Cho, and Yoshua Bengio. On integrating a language model into neural machine translation. *Computer Speech & Language*, 45: 137–148, 2017.

[Guo et al., 2019] Maosheng Guo, Yu Zhang, and Ting Liu. Gaussian transformer: a lightweight approach for natural language inference. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 6489–6496, 2019.

[Hasler et al., 2018] Eva Hasler, Adrià de Gispert, Gonzalo Iglesias, and Bill Byrne. Neural machine translation decoding with terminology constraints. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 506–512, 2018.

[He et al., 2017] Di He, Hanqing Lu, Yingce Xia, Tao Qin, Liwei Wang, and Tie-Yan Liu. Decoding with value networks for neural machine translation. *Advances in Neural Information Processing Systems*, 30, 2017.

[He et al., 2016] Wei He, Zhongjun He, Hua Wu, and Haifeng Wang. Improved neural machine translation with smt features. In *Proceedings of the Thirtieth AAAI conference on artificial intelligence*, 2016.

[He et al., 2018] Xuanli He, Gholamreza Haffari, and Mohammad Norouzi. Sequence to sequence mixture model for diverse machine translation. In *Proceedings of the 22nd Conference on Computational Natural Language Learning*, pages 583–592, 2018.

[Heafield et al., 2021] Kenneth Heafield, Qianqian Zhu, and Roman Grundkiewicz. Findings of the WMT 2021 shared task on efficient translation. In *Proceedings of the Sixth Conference on Machine Translation*, pages 639–651, 2021.

[Hildebrand and Vogel, 2008] Almut Silja Hildebrand and Stephan Vogel. Combination of machine translation systems via hypothesis selection from combined n-best lists. In *Proceedings of the 8th Conference of the Association for Machine Translation in the Americas: Student Research Workshop*, pages 254–261, 2008.

[Hoang et al., 2017] Cong Duy Vu Hoang, Gholamreza Haffari, and Trevor Cohn. Towards decoding as continuous optimisation in neural machine translation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 146–156, 2017.

[Hokamp and Liu, 2017] Chris Hokamp and Qun Liu. Lexically constrained decoding for sequence generation using grid beam search. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1535–1546, 2017.

[Holtzman et al., 2020] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *Proceedings of the 6th International Conference on Learning Representations ICLR*, 2020.

[Huang et al., 2017] Liang Huang, Kai Zhao, and Mingbo Ma. When to finish? optimal beam search for neural text generation (modulo beam size). In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2134–2139, 2017.

[Jaderberg et al., 2015] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and koray kavukcuoglu. Spatial transformer networks. *Advances in neural information processing systems*, 28, 2015.

[Jean et al., 2015] Sébastien Jean, Orhan Firat, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. Montreal neural machine translation systems for wmt'15. In *Proceedings of the tenth workshop on statistical machine translation*, pages 134–140, 2015.

[Khayrallah et al., 2017] Huda Khayrallah, Gaurav Kumar, Kevin Duh, Matt Post, and Philipp Koehn. Neural lattice search for domain adaptation in machine translation. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 20–25, 2017.

[Kikuchi et al., 2016] Yuta Kikuchi, Graham Neubig, Ryohei Sasano, Hiroya Takamura, and Manabu Okumura. Controlling output length in neural encoder-decoders. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1328–1338, 2016.

[Klein et al., 2017] Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M Rush. Opennmt: Open-source toolkit for neural machine translation. In *Proceedings of ACL 2017, System Demonstrations*, pages 67–72, 2017.

[Knight, 1999] Kevin Knight. Decoding complexity in word-replacement translation models. *Computational linguistics*, 25(4):607–615, 1999.

[Koehn, 2004] Philipp Koehn. Pharaoh: a beam search decoder for phrase-based statistical machine translation models. In *Conference of the Association for Machine Translation in the Americas*, pages 115–124. Springer, 2004.

[Koehn, 2010] Philipp Koehn. *Statistical Machine Translation*. Cambridge University Press, 2010.

[Koehn and Knowles, 2017] Philipp Koehn and Rebecca Knowles. Six challenges for neural machine translation. In *Proceedings of the First Workshop on Neural Machine Translation*, pages 28–39, 2017.

[Koehn et al., 2003] Philipp Koehn, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 127–133, 2003.

[Koehn et al., 2007] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 177–180, 2007.

[Kulikov et al., 2019] Ilia Kulikov, Alexander Miller, Kyunghyun Cho, and Jason Weston. Importance of search and evaluation strategies in neural dialogue modeling. In *Proceedings of the 12th International Conference on Natural Language Generation*, pages 76–87, 2019.

[Kumar et al., 2016] Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. Ask me anything: Dynamic memory networks for natural language processing. In *International conference on machine learning*, pages 1378–1387, 2016.

[Kumar et al., 2021] Sachin Kumar, Eric Malmi, Aliaksei Severyn, and Yulia Tsvetkov. Controlled text generation as continuous optimization with multiple constraints. *Advances in Neural Information Processing Systems*, 34:14542–14554, 2021.

[Kumar and Byrne, 2004] Shankar Kumar and William Byrne. Minimum bayes-risk decoding for statistical machine translation. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004*, pages 169–176, 2004.

[Lample and Conneau, 2019] Guillaume Lample and Alexis Conneau. Cross-lingual language model pretraining. *arXiv preprint arXiv:1901.07291*, 2019.

[Leblond et al., 2021] Rémi Leblond, Jean-Baptiste Alayrac, Laurent Sifre, Miruna Pislar, Lespiau Jean-Baptiste, Ioannis Antonoglou, Karen Simonyan, and Oriol Vinyals. Machine translation decoding beyond beam search. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8410–8434, 2021.

[Lee et al., 2020] Jason Lee, Elman Mansimov, and Kyunghyun Cho. Deterministic non-autoregressive neural sequence modeling by iterative refinement. In *2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018*, pages 1173–1182, 2020.

[Lee et al., 2019] John Boaz Lee, Ryan A Rossi, Sungchul Kim, Nesreen K Ahmed, and Eunyee Koh. Attention models in graphs: A survey. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 13(6):1–25, 2019.

[Li et al., 2021] Jicheng Li, Pengzhi Gao, Xuanfu Wu, Yang Feng, Zhongjun He, Hua Wu, and Haifeng Wang. Mixup decoding for diverse machine translation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 312–320, 2021.

[Li and Jurafsky, 2016] Jiwei Li and Dan Jurafsky. Mutual information and diverse decoding improve neural machine translation. *arXiv preprint arXiv:1601.00372*, 2016.

[Li et al., 2016] Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and William B Dolan. A diversity-promoting objective function for neural conversation models. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 110–119, 2016.

[Li et al., 2017] Jiwei Li, Will Monroe, and Dan Jurafsky. Learning to decode for future success. *arXiv preprint arXiv:1701.06549*, 2017.

[Li et al., 2019] Xintong Li, Guanlin Li, Lemao Liu, Max Meng, and Shuming Shi. On the word alignment from neural machine translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1293–1303, 2019.

[Li et al., 2018] Yanyang Li, Tong Xiao, Yinqiao Li, Qiang Wang, Changming Xu, and Jingbo Zhu. A simple and effective approach to coverage-aware neural machine translation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 292–297, 2018.

[Liu et al., 2016] Lemao Liu, Masao Utiyama, Andrew Finch, and Eiichiro Sumita. Agreement on target-bidirectional neural machine translation. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 411–416, 2016a.

[Liu et al., 2016] Lemao Liu, Masao Utiyama, Andrew Finch, and Eiichiro Sumita. Neural machine translation with supervised attention. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 3093–3102, 2016b.

[Lu et al., 2016] Jiasen Lu, Jianwei Yang, Dhruv Batra, and Devi Parikh. Hierarchical question-image co-attention for visual question answering. *Advances in neural information processing systems*, 29, 2016.

[Luong et al., 2015] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, 2015.

[Ma et al., 2019] Mingbo Ma, Liang Huang, Hao Xiong, Renjie Zheng, Kaibo Liu, Baigong Zheng, Chuanqiang Zhang, Zhongjun He, Hairong Liu, Xing Li, Hua Wu, and Haifeng Wang. Stacl: Simultaneous translation with implicit anticipation and controllable latency using prefix-to-prefix framework. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3025–3036, 2019.

[Malaviya et al., 2018] Chaitanya Malaviya, Pedro Ferreira, and André FT Martins. Sparse and constrained attention for neural machine translation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 370–376, 2018.

[Maruf et al., 2019] Sameen Maruf, André FT Martins, and Gholamreza Haffari. Selective attention for context-aware neural machine translation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3092–3102, 2019.

[Matusov et al., 2006] Evgeny Matusov, Nicola Ueffing, and Hermann Ney. Computing consensus translation for multiple machine translation systems using enhanced hypothesis alignment. In *11th Conference of the European Chapter of the Association for Computational Linguistics*, pages 33–40, 2006.

[Meister et al., 2020] Clara Meister, Tim Vieira, and Ryan Cotterell. Best-first beam search. *Transactions of the Association for Computational Linguistics*, 8:795–809, 2020.

[Mi et al., 2016] Haitao Mi, Baskaran Sankaran, Zhiguo Wang, and Abe Ittycheriah. Coverage embedding models for neural machine translation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 955–960, 2016a.

[Mi et al., 2016] Haitao Mi, Zhiguo Wang, and Abe Ittycheriah. Supervised attentions for neural machine translation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural*

Language Processing, pages 2283–2288, 2016b.

[Müller and Sennrich, 2021] Mathias Müller and Rico Sennrich. Understanding the properties of minimum bayes risk decoding in neural machine translation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 259–272, 2021.

[Murray and Chiang, 2018] Kenton Murray and David Chiang. Correcting length bias in neural machine translation. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 212–223, 2018.

[Neisser, 2014] Ulric Neisser. *Cognitive Psychology: Classic Edition*. Psychology Press, 2014.

[Nguyen et al., 2020] Xuan-Phi Nguyen, Shafiq Joty, Steven Hoi, and Richard Socher. Tree-structured attention with hierarchical accumulation. In *Proceedings of the 8th International Conference on Learning Representations ICLR*, 2020.

[Och, 2003] Franz Josef Och. Minimum error rate training in statistical machine translation. In *Proceedings of the 41st annual meeting of the Association for Computational Linguistics*, pages 160–167, 2003.

[Och and Ney, 2002] Franz Josef Och and Hermann Ney. Discriminative training and maximum entropy models for statistical machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 295–302, 2002.

[Och and Ney, 2003] Franz Josef Och and Hermann Ney. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51, 2003.

[Opitz and Maclin, 1999] David Opitz and Richard Maclin. Popular ensemble methods: An empirical study. *Journal of artificial intelligence research*, 11:169–198, 1999.

[Ott et al., 2018] Myle Ott, Michael Auli, David Grangier, and Marc'Aurelio Ranzato. Analyzing uncertainty in neural machine translation. In *International Conference on Machine Learning*, pages 3956–3965. PMLR, 2018a.

[Ott et al., 2018] Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. Scaling neural machine translation. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 1–9, October 2018b.

[Ott et al., 2019] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 48–53, 2019.

[Post and Vilar, 2018] Matt Post and David Vilar. Fast lexically constrained decoding with dynamic beam allocation for neural machine translation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1314–1324, 2018.

[Raffel et al., 2017] Colin Raffel, Minh-Thang Luong, Peter J Liu, Ron J Weiss, and Douglas Eck. Online and linear-time attention by enforcing monotonic alignments. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2837–2846, 2017.

[Raffel et al., 2020] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

[Ren et al., 2017] Zhou Ren, Xiaoyu Wang, Ning Zhang, Xutao Lv, and Li-Jia Li. Deep reinforcement learning-based image captioning with embedding reward. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 290–298, 2017.

[Rosti et al., 2007] Antti-Veikko Rosti, Spyros Matsoukas, and Richard Schwartz. Improved word-level system combination for machine translation. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 312–319, 2007.

[Ruder, 2017] Sebastian Ruder. Deep learning for nlp best practices. https://ruder.io/deep-learning-nlp-best-practices/index.html, 2017.

[Rush and Collins, 2012] Alexander M Rush and MJ Collins. A tutorial on dual decomposition and lagrangian relaxation for inference in natural language processing. *Journal of Artificial Intelligence Research*, 45:305–362, 2012.

[Rush et al., 2015] Alexander M Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 379–389, 2015.

[Russell and Norvig, 2010] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (3nd ed.)*. Prentice Hall, 2010.

[Sankaran et al., 2016] Baskaran Sankaran, Haitao Mi, Yaser Al-Onaizan, and Abe Ittycheriah. Temporal attention model for neural machine translation. *arXiv preprint arXiv:1608.02927*, 2016.

[See, 2018] Abigail See. Deep learning, structure and innate priors: A discussion between yann lecun and christopher manning, 02 2018. URL http://www.abigailsee.com/2018/02/21/deep-learning-structure-and-innate-priors.html.

[See et al., 2017] Abigail See, Peter J Liu, and Christopher D Manning. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, 2017.

[Sennrich et al., 2016] Rico Sennrich, Barry Haddow, and Alexandra Birch. Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 86–96, 2016.

[Shannon, 1948] Claude E. Shannon. A mathematical theory of communication. Report, Bell Labs, 1948.

[Shen et al., 2016] Shiqi Shen, Yong Cheng, Zhongjun He, Wei He, Hua Wu, Maosong Sun, and Yang Liu. Minimum risk training for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1683–1692, 2016.

[Shen et al., 2019] Tianxiao Shen, Myle Ott, Michael Auli, and Marc'Aurelio Ranzato. Mixture models for diverse machine translation: Tricks of the trade. In *International conference on machine learning*, pages 5719–5728, 2019.

[Silver et al., 2017] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[Stahlberg and Byrne, 2019] Felix Stahlberg and Bill Byrne. On nmt search errors and model errors: Cat got your tongue? In *Proceedings of the 2019 Conference on Empirical Methods in Natural*

Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pages 3356–3362, 2019.

[Stahlberg et al., 2016] Felix Stahlberg, Eva Hasler, Aurelien Waite, and Bill Byrne. Syntactically guided neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 299–305, 2016.

[Sternberg, 1996] Robert J Sternberg. *Cognitive psychology*. Harcourt Brace College Publishers, 1996.

[Sukhbaatar et al., 2015] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. End-to-end memory networks. *Advances in neural information processing systems*, 28, 2015.

[Sun et al., 2020] Zewei Sun, Shujian Huang, Hao-Ran Wei, Xin-yu Dai, and Jiajun Chen. Generating diverse translation by manipulating multi-head attention. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8976–8983, 2020.

[Sutskever et al., 2014] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.

[Taskar et al., 2005] Ben Taskar, Simon Lacoste-Julien, and Dan Klein. A discriminative matching approach to word alignment. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 73–80, 2005.

[Tay et al., 2020] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *CoRR*, abs/2009.06732, 2020.

[Tu et al., 2016] Zhaopeng Tu, Zhengdong Lu, Yang Liu, Xiaohua Liu, and Hang Li. Modeling coverage for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 76–85, 2016.

[Vaswani et al., 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of Advances in Neural Information Processing Systems*, volume 30, 2017.

[Veličković et al., 2018] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

[Vijayakumar et al., 2018] Ashwin Vijayakumar, Michael Cogswell, Ramprasaath Selvaraju, Qing Sun, Stefan Lee, David Crandall, and Dhruv Batra. Diverse beam search for improved description of complex scenes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[Vogel et al., 1996] Stephan Vogel, Hermann Ney, and Christoph Tillmann. Hmm-based word alignment in statistical translation. In *COLING 1996 Volume 2: The 16th International Conference on Computational Linguistics*, 1996.

[Wiher et al., 2022] Gian Wiher, Clara Meister, and Ryan Cotterell. On decoding strategies for neural text generators. *Transactions of the Association for Computational Linguistics*, 10:997–1012, 2022.

[Wu et al., 2020] Xuanfu Wu, Yang Feng, and Chenze Shao. Generating diverse translation from model distribution with dropout. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1088–1097, 2020.

[Wu et al., 2016] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason

Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[Xiao et al., 2013] Tong Xiao, Jingbo Zhu, and Tongran Liu. Bagging and boosting statistical machine translation systems. *Artificial Intelligence*, 195:496–527, 2013.

[Xu et al., 2015] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057. PMLR, 2015.

[Xu et al., 2021] Zenan Xu, Daya Guo, Duyu Tang, Qinliang Su, Linjun Shou, Ming Gong, Wanjun Zhong, Xiaojun Quan, Daxin Jiang, and Nan Duan. Syntax-enhanced pre-trained model. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 5412–5422, 2021.

[Yang et al., 2018] Baosong Yang, Zhaopeng Tu, Derek F Wong, Fandong Meng, Lidia S Chao, and Tong Zhang. Modeling localness for self-attention networks. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4449–4458, 2018a.

[Yang et al., 2018] Yilin Yang, Liang Huang, and Mingbo Ma. Breaking the beam search curse: A study of (re-) scoring methods and stopping criteria for neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3054–3059, 2018b.

[Yang et al., 2016] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, pages 1480–1489, 2016.

[You et al., 2020] Weiqiu You, Simeng Sun, and Mohit Iyyer. Hard-coded gaussian attention for neural machine translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7689–7700, 2020.

[Zaslavskiy et al., 2009] Mikhail Zaslavskiy, Marc Dymetman, and Nicola Cancedda. Phrase-based statistical machine translation as a traveling salesman problem. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 333–341, 2009.

[Zhang et al., 2020] Jiajun Zhang, Long Zhou, Yang Zhao, and Chengqing Zong. Synchronous bidirectional inference for neural sequence generation. *Artificial Intelligence*, 281:103234, 2020a.

[Zhang et al., 2018] Xiangwen Zhang, Jinsong Su, Yue Qin, Yang Liu, Rongrong Ji, and Hongji Wang. Asynchronous bidirectional decoding for neural machine translation. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[Zhang et al., 2020] Zhuosheng Zhang, Yuwei Wu, Junru Zhou, Sufeng Duan, Hai Zhao, and Rui Wang. Sg-net: Syntax-guided machine reading comprehension. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 9636–9643, 2020b.

[Zheng et al., 2019] Baigong Zheng, Renjie Zheng, Mingbo Ma, and Liang Huang. Simpler and faster learning of adaptive policies for simultaneous translation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on*

Natural Language Processing (EMNLP-IJCNLP), pages 1349–1354, 2019.

[Zheng et al., 2018] Zaixiang Zheng, Hao Zhou, Shujian Huang, Lili Mou, Xinyu Dai, Jiajun Chen, and Zhaopeng Tu. Modeling past and future for neural machine translation. *Transactions of the Association for Computational Linguistics*, 6:145–157, 2018.

[Zhou et al., 2017] Long Zhou, Wenpeng Hu, Jiajun Zhang, and Chengqing Zong. Neural system combination for machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 378–384, 2017.

[Zhou, 2012] Zhi-Hua Zhou. *Ensemble methods: foundations and algorithms*. CRC press, 2012.