Chapter 19

Text generation

In many of the most interesting problems in natural language processing, language is the output. The previous chapter described the specific case of machine translation, but there are many other applications, from summarization of research articles, to automated journalism, to dialogue systems. This chapter emphasizes three main scenarios: data-to-text, in which text is generated to explain or describe a structured record or unstructured perceptual input; text-to-text, which typically involves fusing information from multiple linguistic sources into a single coherent summary; and dialogue, in which text is generated as part of an interactive conversation with one or more human participants.

19.1 Data-to-text generation

In data-to-text generation, the input ranges from structured records, such as the description of an weather forecast (as shown in Figure 19.1), to unstructured perceptual data, such as a raw image or video; the output may be a single sentence, such as an image caption, or a multi-paragraph argument. Despite this diversity of conditions, all data-to-text systems share some of the same challenges (Reiter and Dale, 2000):

- determining what parts of the data to describe;
- planning a presentation of this information;
- **lexicalizing** the data into words and phrases;
- organizing words and phrases into well-formed sentences and paragraphs.

The earlier stages of this process are sometimes called **content selection** and **text planning**; the later stages are often called **surface realization**.

Early systems for data-to-text generation were modular, with separate software components for each task. Artificial intelligence **planning** algorithms can be applied to both

Temperature

time	*min*	*mean*	*max*
06:00-21:00	9	15	21

Cloud sky cover

time	*percent (%)*
06:00-09:00	25-50
09:00-12:00	50-75

Wind speed

time	*min*	*mean*	*max*
06:00-21:00	15	20	30

Wind direction

time	*mode*
06:00-21:00	S

Cloudy, with temperatures between 10 and 20 degrees. South wind around 20 mph.

Figure 19.1: An example input-output pair for the task of generating text descriptions of weather forecasts (adapted from Konstas and Lapata, 2013).

the high-level information structure and the organization of individual sentences, ensuring that communicative goals are met (McKeown, 1992; Moore and Paris, 1993). Surface realization can be performed by grammars or templates, which link specific types of data to candidate words and phrases. A simple example template is offered by Wiseman et al. (2017), for generating descriptions of basketball games:

(19.1) The `<team1>` (`<wins1>-losses1`) defeated the `<team2>` (`<wins2>-<losses2>`), `<pts1>-<pts2>`.
The New York Knicks (45-5) defeated the Boston Celtics (11-38), 115-79.

For more complex cases, it may be necessary to apply morphological inflections such as pluralization and tense marking — even in the simple example above, languages such as Russian would require case marking suffixes for the team names. Such inflections can be applied as a postprocessing step. Another difficult challenge for surface realization is the generation of varied **referring expressions** (e.g., *The Knicks*, *New York*, *they*), which is critical to avoid repetition. As discussed in § 16.2.1, the form of referring expressions is constrained by the discourse and information structure.

An example at the intersection of rule-based and statistical techniques is the NITROGEN system (Langkilde and Knight, 1998). The input to NITROGEN is an abstract meaning representation (AMR; see § 13.3) of semantic content to be expressed in a single sentence. In data-to-text scenarios, the abstract meaning representation is the output of a higher-level text planning stage. A set of rules then converts the abstract meaning representation into various sentence plans, which may differ in both the high-level structure (e.g., active versus passive voice) as well as the low-level details (e.g., word and phrase choice). Some examples are shown in Figure 19.2. To control the combinatorial explosion in the number of possible realizations for any given meaning, the sentence plans are unified into a single finite-state acceptor, in which word tokens are represented by arcs (see § 9.1.1). A bigram

```
(a / admire-01
  :ARG0 (v / visitor
        :ARG1-of (c / arrive-01
                   :ARG4 (j / Japan)))
  :ARG1 (m / "Mount Fuji"))
```

- Visitors who came to Japan admire Mount Fuji.
- Visitors who came in Japan admire Mount Fuji.
- Mount Fuji is admired by the visitor who came in Japan.

Figure 19.2: Abstract meaning representation and candidate surface realizations from the NITROGEN system. Example adapted from Langkilde and Knight (1998).

language model is then used to compute weights on the arcs, so that the shortest path is also the surface realization with the highest bigram language model probability.

More recent systems are unified models that are trained end-to-end using backpropagation. Data-to-text generation shares many properties with machine translation, including a problem of **alignment**: labeled examples provide the data and the text, but they do not specify which parts of the text correspond to which parts of the data. For example, to learn from Figure 19.1, the system must align the word *cloudy* to records in CLOUD SKY COVER, the phrases *10* and *20 degrees* to the MIN and MAX fields in TEMPERATURE, and so on. As in machine translation, both latent variables and neural attention have been proposed as solutions.

19.1.1 Latent data-to-text alignment

Given a dataset of texts and associated records $\{(\boldsymbol{w}^{(i)}, \boldsymbol{y}^{(i)})\}_{i=1}^{N}$, our goal is to learn a model Ψ, so that

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w} \in \mathcal{V}^*}{\operatorname{argmax}} \Psi(\boldsymbol{w}, \boldsymbol{y}; \boldsymbol{\theta}), \qquad [19.1]$$

where $\mathcal{V}^*$ is the set of strings over a discrete vocabulary, and $\boldsymbol{\theta}$ is a vector of parameters. The relationship between $\boldsymbol{w}$ and $\boldsymbol{y}$ is complex: the data $\boldsymbol{y}$ may contain dozens of records, and $\boldsymbol{w}$ may extend to several sentences. To facilitate learning and inference, it would be helpful to decompose the scoring function Ψ into subcomponents. This would be possible if given an **alignment**, specifying which element of $\boldsymbol{y}$ is expressed in each part of $\boldsymbol{w}$. Specifically, let z_m indicates the record aligned to word m. For example, in Figure 19.1, z_1 might specify that the word *cloudy* is aligned to the record `cloud-sky-cover:percent`. The score for this alignment would then be given by the weight on features such as

$$(\textit{cloudy}, \texttt{cloud-sky-cover:percent}). \qquad [19.2]$$

In general, given an observed set of alignments, the score for a generation can be written as sum of local scores (Angeli et al., 2010):

$$\Psi(\boldsymbol{w}, \boldsymbol{y}; \boldsymbol{\theta}) = \sum_{m=1}^{M} \psi_{w,y}(\boldsymbol{w}_m, \boldsymbol{y}_{z_m}) + \psi_w(w_m, w_{m-1}) + \psi_z(z_m, z_{m-1}), \qquad [19.3]$$

where ψ_w can represent a bigram language model, and ψ_z can be tuned to reward coherence, such as the use of related records in nearby words. [1] The parameters of this model could be learned from labeled data $\{(\boldsymbol{w}^{(i)}, \boldsymbol{y}^{(i)}, \boldsymbol{z}^{(i)})\}_{i=1}^{N}$. However, while several datasets include structured records and natural language text (Barzilay and McKeown, 2005; Chen and Mooney, 2008; Liang and Klein, 2009), the alignments between text and records are usually not available.[2] One solution is to model the problem probabilistically, treating the alignment as a latent variable (Liang et al., 2009; Konstas and Lapata, 2013). The model can then be estimated using expectation maximization or sampling (see chapter 5).

19.1.2 Neural data-to-text generation

The **encoder-decoder model** and **neural attention** were introduced in § 18.3 as methods for neural machine translation. They can also be applied to data-to-text generation, with the data acting as the source language (Mei et al., 2016). In neural machine translation, the attention mechanism linked words in the source to words in the target; in data-to-text generation, the attention mechanism can link each part of the generated text back to a record in the data. The biggest departure from translation is in the encoder, which depends on the form of the data.

Data encoders

In some types of structured records, all values are drawn from discrete sets. For example, the birthplace of an individual is drawn from a discrete set of possible locations; the diagnosis and treatment of a patient are drawn from an exhaustive list of clinical codes (Johnson et al., 2016). In such cases, vector embeddings can be estimated for each field and possible value: for example, a vector embedding for the field BIRTHPLACE, and another embedding for the value BERKELEY_CALIFORNIA (Bordes et al., 2011). The table of such embeddings serves as the encoding of a structured record (He et al., 2017). It is also possible to compress the entire table into a single vector representation, by **pooling** across the embeddings of each field and value (Lebret et al., 2016).

[1] More expressive decompositions of Ψ are possible. For example, Wong and Mooney (2007) use a synchronous context-free grammar (see § 18.2.4) to "translate" between a meaning representation and natural language text.

[2] An exception is a dataset of records and summaries from American football games, containing annotations of alignments between sentences and records (Snyder and Barzilay, 2007).

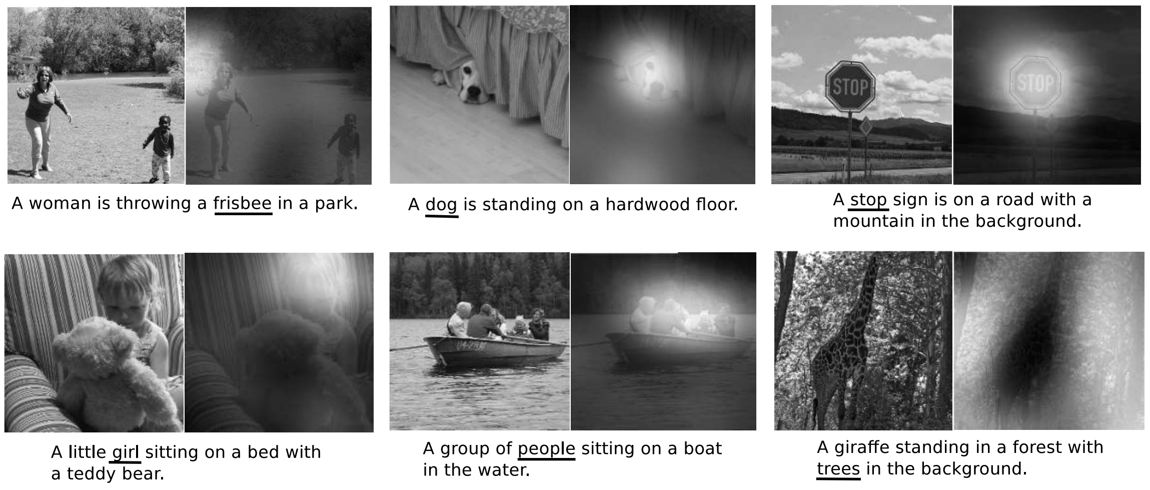
A woman is throwing a frisbee in a park.
A dog is standing on a hardwood floor.
A stop sign is on a road with a mountain in the background.
A little girl sitting on a bed with a teddy bear.
A group of people sitting on a boat in the water.
A giraffe standing in a forest with trees in the background.

Figure 19.3: Examples of the image captioning task, with attention masks shown for each of the underlined words (Xu et al., 2015).

Sequences Some types of structured records have a natural ordering, such as events in a game (Chen and Mooney, 2008) and steps in a recipe (Tutin and Kittredge, 1992). For example, the following records describe a sequence of events in a robot soccer match (Mei et al., 2016):

$$\begin{aligned}&\textsc{pass}(\text{arg1} = \textsc{purple6}, \text{arg2} = \textsc{purple3})\\&\textsc{kick}(\text{arg1} = \textsc{purple3})\\&\textsc{badpass}(\text{arg1} = \textsc{purple3}, \text{arg2} = \textsc{pink9}).\end{aligned}$$

Each event is a single record, and can be encoded by a concatenation of vector representations for the event type (e.g., PASS), the field (e.g., arg1), and the values (e.g., PURPLE3), e.g.,

$$\mathbf{X} = \left[\boldsymbol{u}_{\textsc{pass}}, \boldsymbol{u}_{\text{arg1}}, \boldsymbol{u}_{\textsc{purple6}}, \boldsymbol{u}_{\text{arg2}}, \boldsymbol{u}_{\textsc{purple3}}\right]. \qquad [19.4]$$

This encoding can then act as the input layer for a recurrent neural network, yielding a sequence of vector representations $\{\boldsymbol{z}_r\}_{r=1}^{R}$, where r indexes over records. Interestingly, this sequence-based approach can work even in cases where there is no natural ordering over the records, such as the weather data in Figure 19.1 (Mei et al., 2016).

Images Another flavor of data-to-text generation is the generation of text captions for images. Examples from this task are shown in Figure 19.3. Images are naturally represented as tensors: a color image of 320×240 pixels would be stored as a tensor with $320 \times 240 \times 3$ intensity values. The dominant approach to image classification is to encode images as vectors using a combination of convolution and pooling (Krizhevsky et al.,

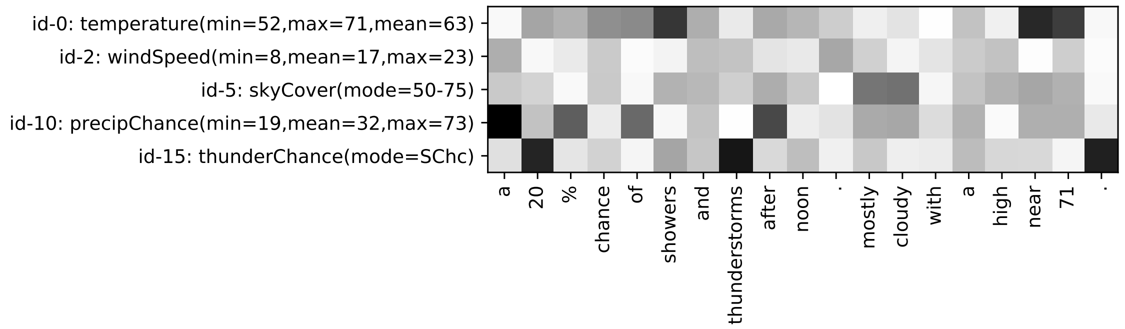

Figure 19.4: Neural attention in text generation. Figure adapted from Mei et al. (2016).

2012). Chapter 3 explains how to use convolutional networks for text; for images, convolution is applied across the vertical, horizontal, and color dimensions. By pooling the results of successive convolutions, the image is converted to a vector representation, which can then be fed directly into the decoder as the initial state (Vinyals et al., 2015), just as in the sequence-to-sequence translation model (see § 18.3). Alternatively, one can apply a set of convolutional networks, yielding vector representations for different parts of the image, which can then be combined using neural attention (Xu et al., 2015).

Attention

Given a set of embeddings of the data $\{\boldsymbol{z}_r\}_{r=1}^{R}$ and a decoder state $\boldsymbol{h}_m$, an attention vector over the data can be computed using the same techniques as in machine translation (see § 18.3.1). When generating word m of the output, attention is computed over the records,

$$\psi_\alpha(m, r) = \boldsymbol{\beta}_\alpha \cdot f(\Theta_\alpha[\boldsymbol{h}_m; \boldsymbol{z}_r]) \quad [19.5]$$

$$\boldsymbol{\alpha}_m = g\left([\psi_\alpha(m,1), \psi_\alpha(m,2), \ldots, \psi_\alpha(m,R)]\right) \quad [19.6]$$

$$\boldsymbol{c}_m = \sum_{r=1}^{R} \alpha_{m \to r} \boldsymbol{z}_r, \quad [19.7]$$

where f is an elementwise nonlinearity such as $\tanh$ or ReLU, and g is a either softmax or elementwise sigmoid. The weighted sum $\boldsymbol{c}_m$ can then be included in the recurrent update to the decoder state, or in the emission probabilities, as described in § 18.3.1. Figure 19.4 shows the attention to components of a weather record, while generating the text shown on the x-axis.

Adapting this architecture to image captioning is straightforward. A convolutional neural networks is applied to a set of image locations, and the output at each location ℓ is represented with a vector $\boldsymbol{z}_\ell$. Attention can then be computed over the image locations, as shown in the right panels of each pair of images in Figure 19.3.

Various modifications to this basic mechanism have been proposed. In **coarse-to-fine attention** (Mei et al., 2016), each record receives a global attention $a_r \in [0, 1]$, which is independent of the decoder state. This global attention, which represents the overall importance of the record, is multiplied with the decoder-based attention scores, before computing the final normalized attentions. In **structured attention**, the attention vector $\boldsymbol{\alpha}_{m\to\cdot}$ can include structural biases, which can favor assigning higher attention values to contiguous segments or to dependency subtrees (Kim et al., 2017). Structured attention vectors can be computed by running the forward-backward algorithm to obtain marginal attention probabilities (see § 7.5.3). Because each step in the forward-backward algorithm is differentiable, it can be encoded in a computation graph, and end-to-end learning can be performed by backpropagation.

Decoder

Given the encoding, the decoder can function just as in neural machine translation (see § 18.3.1), using the attention-weighted encoder representation in the decoder recurrence and/or output computation. As in machine translation, beam search can help to avoid search errors (Lebret et al., 2016).

Many applications require generating words that do not appear in the training vocabulary. For example, a weather record may contain a previously unseen city name; a sports record may contain a previously unseen player name. Such tokens can be generated in the text by copying them over from the input (e.g., Gulcehre et al., 2016).[3] First introduce an additional variable $s_m \in \{\textsf{gen}, \textsf{copy}\}$, indicating whether token $w_m^{(t)}$ should be generated or copied. The decoder probability is then,

$$\mathrm{p}(w^{(t)} \mid \boldsymbol{w}_{1:m-1}^{(t)}, \mathbf{Z}, s_m) = \begin{cases} \text{SoftMax}(\boldsymbol{\beta}_{w^{(t)}} \cdot \boldsymbol{h}_{m-1}^{(t)}), & s_m = \textsf{gen} \\ \sum_{r=1}^{R} \delta\left(w_r^{(s)} = w^{(t)}\right) \times \alpha_{m\to r}, & s_m = \textsf{copy}, \end{cases} \quad [19.8]$$

where $\delta(w_r^{(s)} = w^{(t)})$ is an indicator function, taking the value 1 iff the text of the record $w_r^{(s)}$ is identical to the target word $w^{(t)}$. The probability of copying record r from the source is $\delta\left(s_m = \textsf{copy}\right) \times \alpha_{m\to r}$, the product of the copy probability by the local attention. Note that in this model, the attention weights $\boldsymbol{\alpha}_m$ are computed from the *previous* decoder state $\boldsymbol{h}_{m-1}$. The computation graph therefore remains a feedforward network, with recurrent paths such as $\boldsymbol{h}_{m-1}^{(t)} \to \boldsymbol{\alpha}_m \to w_m^{(t)} \to \boldsymbol{h}_m^{(t)}$.

To facilitate end-to-end training, the switching variable s_m can be represented by a gate π_m, which is computed from a two-layer feedforward network, whose input consists of the concatenation of the decoder state $\boldsymbol{h}_{m-1}^{(t)}$ and the attention-weighted representation

[3]A number of variants of this strategy have been proposed (e.g., Gu et al., 2016; Merity et al., 2017). See Wiseman et al. (2017) for an overview.

of the data, $\boldsymbol{c}_m = \sum_{r=1}^{R} \alpha_{m\to r}\boldsymbol{z}_r$,

$$\pi_m = \sigma(\Theta^{(2)} f(\Theta^{(1)}[\boldsymbol{h}^{(t)}_{m-1}; \boldsymbol{c}_m])). \qquad [19.9]$$

The full generative probability at token m is then,

$$\mathrm{p}(w^{(t)} \mid \boldsymbol{w}^{(t)}_{1:m}, \mathbf{Z}) = \pi_m \times \underbrace{\frac{\exp \boldsymbol{\beta}_{w^{(t)}} \cdot \boldsymbol{h}^{(t)}_{m-1}}{\sum_{j=1}^{V} \exp \boldsymbol{\beta}_j \cdot \boldsymbol{h}^{(t)}_{m-1}}}_{\text{generate}} + (1-\pi_m) \times \underbrace{\sum_{r=1}^{R} \delta(w^{(s)}_r = w^{(t)}) \times \alpha_{m\to r}}_{\text{copy}}. \qquad [19.10]$$

19.2 Text-to-text generation

Text-to-text generation includes problems of summarization and simplification:

- reading a novel and outputting a paragraph-long summary of the plot;[4]
- reading a set of blog posts about politics, and outputting a bullet list of the various issues and perspectives;
- reading a technical research article about the long-term health consequences of drinking kombucha, and outputting a summary of the article in language that non-experts can understand.

These problems can be approached in two ways: through the encoder-decoder architecture discussed in the previous section, or by operating directly on the input text.

19.2.1 Neural abstractive summarization

Sentence summarization is the task of shortening a sentence while preserving its meaning, as in the following examples (Knight and Marcu, 2000; Rush et al., 2015):

(19.2) a. The documentation is typical of Epson quality: excellent.
Documentation is excellent.

b. Russian defense minister Ivanov called sunday for the creation of a joint front for combating global terrorism.
Russia calls for joint front against terrorism.

[4] In § 16.3.4, we encountered a special case of single-document summarization, which involved extracting the most important sentences or discourse units. We now consider the more challenging problem of **abstractive summarization**, in which the summary can include words that do not appear in the original text.

Sentence summarization is closely related to **sentence compression**, in which the summary is produced by deleting words or phrases from the original (Clarke and Lapata, 2008). But as shown in (19.2b), a sentence summary can also introduce new words, such as *against*, which replaces the phrase *for combatting*.

Sentence summarization can be treated as a machine translation problem, using the attentional encoder-decoder translation model discussed in § 18.3.1 (Rush et al., 2015). The longer sentence is encoded into a sequence of vectors, one for each token. The decoder then computes attention over these vectors when updating its own recurrent state. As with data-to-text generation, it can be useful to augment the encoder-decoder model with the ability to copy words directly from the source. Rush et al. (2015) train this model by building four million sentence pairs from news articles. In each pair, the longer sentence is the first sentence of the article, and the summary is the article headline. Sentence summarization can also be trained in a semi-supervised fashion, using a probabilistic formulation of the encoder-decoder model called a **variational autoencoder** (Miao and Blunsom, 2016, also see § 14.8.2).

When summarizing longer documents, an additional concern is that the summary not be repetitive: each part of the summary should cover new ground. This can be addressed by maintaining a vector of the sum total of all attention values thus far, $\boldsymbol{t}_m = \sum_{n=1}^{m} \boldsymbol{\alpha}_n$. This total can be used as an additional input to the computation of the attention weights,

$$\alpha_{m\to n} \propto \exp\left(\boldsymbol{v}_\alpha \cdot \tanh(\Theta_\alpha[\boldsymbol{h}_m^{(t)}; \boldsymbol{h}_n^{(s)}; \boldsymbol{t}_m])\right), \quad [19.11]$$

which enables the model to learn to prefer parts of the source which have not been attended to yet (Tu et al., 2016). To further encourage diversity in the generated summary, See et al. (2017) introduce a **coverage loss** to the objective function,

$$\ell_m = \sum_{n=1}^{M^{(s)}} \min(\alpha_{m\to n}, t_{m\to n}). \quad [19.12]$$

This loss will be low if $\boldsymbol{\alpha}_m$assigns little attention to words that already have large values in $\boldsymbol{t}_m$.Coverage loss is similar to the concept of **marginal relevance**, in which the reward for adding new content is proportional to the extent to which it increases the overall amount of information conveyed by the summary (Carbonell and Goldstein, 1998).

19.2.2 Sentence fusion for multi-document summarization

In **multi-document summarization**, the goal is to produce a summary that covers the content of several documents (McKeown et al., 2002). One approach to this challenging problem is to identify sentences across multiple documents that relate to a single theme, and then to fuse them into a single sentence (Barzilay and McKeown, 2005). As an example, consider the following two sentences (McKeown et al., 2010):

(19.3) a. Palin actually turned against the bridge project only after it became a national symbol of wasteful spending.

b. Ms. Palin supported the bridge project while running for governor, and abandoned it after it became a national scandal.

An *intersection* preserves only the content that is present in both sentences:

(19.4) Palin turned against the bridge project after it became a national scandal.

A *union* includes information from both sentences:

(19.5) Ms. Palin supported the bridge project while running for governor, but turned against it when it became a national scandal and a symbol of wasteful spending.

Dependency parsing is often used as a technique for sentence fusion. After parsing each sentence, the resulting dependency trees can be aggregated into a lattice (Barzilay and McKeown, 2005) or a graph structure (Filippova and Strube, 2008), in which identical or closely related words (e.g., *Palin, bridge, national*) are fused into a single node. The resulting graph can then be pruned back to a tree by solving an **integer linear program** (see § 13.2.2),

$$\max_{\boldsymbol{y}} \quad \sum_{i,j,r} \psi(i \xrightarrow{r} j, \boldsymbol{w}; \boldsymbol{\theta}) \times y_{i,j,r} \tag{19.13}$$

$$\text{s.t.} \quad \boldsymbol{y} \in \mathcal{C}, \tag{19.14}$$

where the variable $y_{i,j,r} \in \{0, 1\}$ indicates whether there is an edge from i to j of type r, the score of this edge is $\psi(i \xrightarrow{r} j, \boldsymbol{w}; \boldsymbol{\theta})$, and $\mathcal{C}$ is a set of constraints, which ensures that $\boldsymbol{y}$ forms a valid dependency graph. As usual, $\boldsymbol{w}$ is the list of words in the graph, and $\boldsymbol{\theta}$ is a vector of parameters. The score $\psi(i \xrightarrow{r} j, \boldsymbol{w}; \boldsymbol{\theta})$ reflects the "importance" of the modifier j to the overall meaning: in intersective fusion, this score indicates the extent to which the content in this edge is expressed in all sentences; in union fusion, the score indicates whether the content in the edge is expressed in any sentence. The constraint set $\mathcal{C}$ can impose additional linguistic constraints: for example, ensuring that coordinated nouns are sufficiently similar. The resulting tree must then be **linearized** into a sentence. Linearization is like the inverse of dependency parsing: instead of parsing from a sequence of tokens into a tree, we must convert the tree back into a sequence of tokens. This is typically done by generating a set of candidate linearizations, and choosing the one with the highest score under a language model (Langkilde and Knight, 1998; Song et al., 2016).

19.3 Dialogue

Dialogue systems are capable of conversing with a human interlocutor, often to perform some task (Grosz, 1979), but sometimes just to chat (Weizenbaum, 1966). While re-

(19.6) A: I want to order a pizza.
B: What toppings?
A: Anchovies.
B: Ok, what address?
A: The College of Computing building.
B: Please confirm: one pizza with artichokes, to be delivered to the College of Computing building.
A: No.
B: What toppings?
. . .

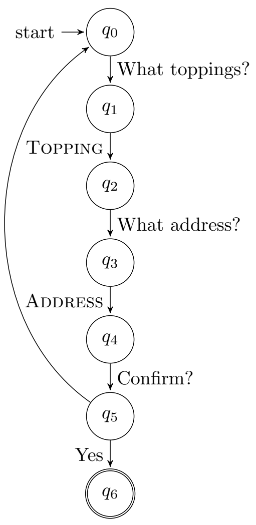

Figure 19.5: An example dialogue and the associated finite-state model. In the finite-state model, SMALL CAPS indicates that the user must provide information of this type in their answer.

search on dialogue systems goes back several decades (Carbonell, 1970; Winograd, 1972), commercial systems such as Alexa and Siri have recently brought this technology into widespread use. Nonetheless, there is a significant gap between research and practice: many practical dialogue systems remain scripted and inflexible, while research systems emphasize abstractive text generation, "on-the-fly" decision making, and probabilistic reasoning about the user's intentions.

19.3.1 Finite-state and agenda-based dialogue systems

Finite-state automata were introduced in chapter 9 as a formal model of computation, in which string inputs and outputs are linked to transitions between a finite number of discrete states. This model naturally fits simple task-oriented dialogues, such as the one shown in the left panel of Figure 19.5. This (somewhat frustrating) dialogue can be represented with a finite-state transducer, as shown in the right panel of the figure. The accepting state is reached only when the two needed pieces of information are provided, and the human user confirms that the order is correct. In this simple scenario, the TOPPING and ADDRESS are the two **slots** associated with the activity of ordering a pizza, which is called a **frame**. Frame representations can be hierarchical: for example, an ADDRESS could have slots of its own, such as STREET and CITY.

In the example dialogue in Figure 19.5, the user provides the precise inputs that are needed in each turn (e.g., *anchovies*; *the College of Computing building*). Some users may prefer to communicate more naturally, with phrases like *I'd, uh, like some anchovies please*. One approach to handling such utterances is to design a custom grammar, with non-terminals for slots such as TOPPING and LOCATION. However, context-free parsing of unconstrained speech input is challenging. A more lightweight alternative is BIO-style sequence labeling (see § 8.3), e.g.:

(19.7)

I'd	*like*	*anchovies*	*,*	*and*	*please*	*bring*	*it*	*to*	*the*	*College*	*of*	*Computing*	*Building*	*.*
O	O	B-TOPPING	O	O	O	O	O	O	B-ADDR	I-ADDR	I-ADDR	I-ADDR	I-ADDR	O

The tagger can be driven by a bi-directional recurrent neural network, similar to recurrent approaches to semantic role labeling described in § 13.2.3.

The input in (19.7) could not be handled by the finite-state system from Figure 19.5, which forces the user to provide the topping first, and then the location. In this sense, the "initiative" is driven completely by the system. **Agenda-based dialogue systems** extend finite-state architectures by attempting to recognize all slots that are filled by the user's reply, thereby handling these more complex examples. Agenda-based systems dynamically pose additional questions until the frame is complete (Bobrow et al., 1977; Allen et al., 1995; Rudnicky and Xu, 1999). Such systems are said to be **mixed-initiative**, because both the user and the system can drive the direction of the dialogue.

19.3.2 Markov decision processes

The task of dynamically selecting the next move in a conversation is known as **dialogue management**. This problem can be framed as a **Markov decision process**, which is a theoretical model that includes a discrete set of states, a discrete set of actions, a function that computes the probability of transitions between states, and a function that computes the cost or reward of action-state pairs. Let's see how each of these elements pertains to the pizza ordering dialogue system.

- Each state is a tuple of information about whether the topping and address are known, and whether the order has been confirmed. For example,

$$(\text{KNOWN TOPPING}, \text{UNKNOWN ADDRESS}, \text{NOT CONFIRMED}) \quad [19.15]$$

  is a possible state. Any state in which the pizza order is confirmed is a terminal state, and the Markov decision process stops after entering such a state.

- The set of actions includes querying for the topping, querying for the address, and requesting confirmation. Each action induces a probability distribution over states, $p(s_t \mid a_t, s_{t-1})$. For example, requesting confirmation of the order is not likely to result in a transition to the terminal state if the topping is not yet known. This probability distribution over state transitions may be learned from data, or it may be specified in advance.

- Each state-action-state tuple earns a reward, $r_a(s_t, s_{t+1})$. In the context of the pizza ordering system, a simple reward function would be,

$$r_a(s_t, s_{t-1}) = \begin{cases} 0, & a = \text{CONFIRM}, s_t = (\text{*, *, CONFIRMED}) \\ -10, & a = \text{CONFIRM}, s_t = (\text{*, *, NOT CONFIRMED}) \\ -1, & a \neq \text{CONFIRM} \end{cases} \quad [19.16]$$

  This function assigns zero reward for successful transitions to the terminal state, a large negative reward to a rejected request for confirmation, and a small negative reward for every other type of action. The system is therefore rewarded for reaching the terminal state in few steps, and penalized for prematurely requesting confirmation.

In a Markov decision process, a **policy** is a function $\pi : \mathcal{S} \to \mathcal{A}$ that maps from states to actions (see § 15.2.4). The value of a policy is the expected sum of discounted rewards, $E_\pi[\sum_{t=1}^{T} \gamma^t r_{a_t}(s_t, s_{t+1})]$, where γ is the discount factor, $\gamma \in [0, 1)$. Discounting has the effect of emphasizing rewards that can be obtained immediately over less certain rewards in the distant future.

An optimal policy can be obtained by dynamic programming, by iteratively updating the **value function** $V(s)$, which is the expectation of the cumulative reward from s under the optimal action a,

$$V(s) \leftarrow \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \text{p}(s' \mid s, a)[r_a(s, s') + \gamma V(s')]. \quad [19.17]$$

The value function $V(s)$ is computed in terms of $V(s')$ for all states $s' \in \mathcal{S}$. A series of iterative updates to the value function will eventually converge to a stationary point. This algorithm is known as **value iteration**. Given the converged value function $V(s)$, the optimal action at each state is the argmax,

$$\pi(s) = \operatorname*{argmax}_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \text{p}(s' \mid s, a)[r_a(s, s') + \gamma V(s')]. \quad [19.18]$$

Value iteration and related algorithms are described in detail by Sutton and Barto (1998). For applications to dialogue systems, see Levin et al. (1998) and Walker (2000).

The Markov decision process framework assumes that the current state of the dialogue is known. In reality, the system may misinterpret the user's statements — for example, believing that a specification of the delivery location (PEACHTREE) is in fact a specification of the topping (PEACHES). In a **partially observable Markov decision process (POMDP)**, the system receives an *observation* o, which is probabilistically conditioned on the state, $p(o \mid s)$. It must therefore maintain a distribution of beliefs about which state it is in, with $q_t(s)$ indicating the degree of belief that the dialogue is in state s at time t. The POMDP formulation can help to make dialogue systems more robust to errors, particularly in the context of spoken language dialogues, where the speech itself may be misrecognized (Roy et al., 2000; Williams and Young, 2007). However, finding the optimal policy in a POMDP is computationally intractable, requiring additional approximations.

19.3.3 Neural chatbots

It's easier to talk when you don't need to get anything done. **Chatbots** are systems that parry the user's input with a response that keeps the conversation going. They can be built from the encoder-decoder architecture discussed in § 18.3 and § 19.1.2: the encoder converts the user's input into a vector, and the decoder produces a sequence of words as a response. For example, Shang et al. (2015) apply the attentional encoder-decoder translation model, training on a dataset of posts and responses from the Chinese microblogging platform Sina Weibo.[5] This approach is capable of generating replies that relate thematically to the input, as shown in the following examples (translated from Chinese by Shang et al., 2015).

(19.8) a. A: High fever attacks me every New Year's day.
B: Get well soon and stay healthy!

b. A: I gain one more year. Grateful to my group, so happy.
B: Getting old now. Time has no mercy.

While encoder-decoder models can generate responses that make sense in the context of the immediately preceding turn, they struggle to maintain coherence over longer conversations. One solution is to model the dialogue context recurrently. This creates a **hierarchical recurrent network**, including both word-level and turn-level recurrences. The turn-level hidden state is then used as additional context in the decoder (Serban et al., 2016).

An open question is how to integrate the encoder-decoder architecture into task-oriented dialogue systems. Neural chatbots can be trained end-to-end: the user's turn is analyzed by the encoder, and the system output is generated by the decoder. This architecture can be trained by log-likelihood using backpropagation (e.g., Sordoni et al., 2015; Serban et al., 2016), or by more elaborate objectives, using reinforcement learning (Li et al., 2016). In contrast, the task-oriented dialogue systems described in § 19.3.1 typically involve a

[5] Twitter is also frequently used for construction of dialogue datasets (Ritter et al., 2011; Sordoni et al., 2015). Another source is technical support chat logs from the Ubuntu linux distribution (Uthus and Aha, 2013; Lowe et al., 2015).

set of specialized modules: one for recognizing the user input, another for deciding what action to take, and a third for arranging the text of the system output.

Recurrent neural network decoders can be integrated into Markov Decision Process dialogue systems, by conditioning the decoder on a representation of the information that is to be expressed in each turn (Wen et al., 2015). Specifically, the long short-term memory (LSTM; § 6.3) architecture is augmented so that the memory cell at turn m takes an additional input $\boldsymbol{d}_m$, which is a representation of the slots and values to be expressed in the next turn. However, this approach still relies on additional modules to recognize the user's utterance and to plan the overall arc of the dialogue.

Another promising direction is to create embeddings for the elements in the domain: for example, the slots in a record and the entities that can fill them. The encoder then encodes not only the words of the user's input, but the embeddings of the elements that the user mentions. Similarly, the decoder is endowed with the ability to refer to specific elements in the knowledge base. He et al. (2017) show that such a method can learn to play a collaborative dialogue game, in which both players are given a list of entities and their properties, and the goal is to find an entity that is on both players' lists.

Additional resources

Gatt and Krahmer (2018) provide a comprehensive recent survey on text generation. For a book-length treatment of earlier work, see Reiter and Dale (2000). For a survey on image captioning, see Bernardi et al. (2016); for a survey of pre-neural approaches to dialogue systems, see Rieser and Lemon (2011). **Dialogue acts** were introduced in § 8.6 as a labeling scheme for human-human dialogues; they also play a critical in task-based dialogue systems (e.g., Allen et al., 1996). The incorporation of theoretical models of dialogue into computational systems is reviewed by Jurafsky and Martin (2009, chapter 24).

While this chapter has focused on the informative dimension of text generation, another line of research aims to generate text with configurable stylistic properties (Walker et al., 1997; Mairesse and Walker, 2011; Ficler and Goldberg, 2017; Hu et al., 2017). This chapter also does not address the generation of creative text such as narratives (Riedl and Young, 2010), jokes (Ritchie, 2001), poems (Colton et al., 2012), and song lyrics (Gonçalo Oliveira et al., 2007).

Exercises

1. Find an article about a professional basketball game, with an associated "box score" of statistics. Which are the first three elements in the box score that are expressed in the article? Can you identify template-based patterns that express these elements of the record? Now find a second article about a different basketball game. Does it

mention the same first three elements of the box score? Do your templates capture how these elements are expressed in the text?

2. This exercise is to be done by a pair of students. One student should choose an article from the news or from Wikipedia, and manually perform semantic role labeling (SRL) on three short sentences or clauses. (See chapter 13 for a review of SRL.) Identify the main the semantic relation and its arguments and adjuncts. Pass this structured record — but not the original sentence — to the other student, whose job is to generate a sentence expressing the semantics. Then reverse roles, and try to regenerate three sentences from another article, based on the predicate-argument semantics.

3. Compute the BLEU scores (see § 18.1.1) for the generated sentences in the previous problem, using the original article text as the reference.

4. Align each token in the text of Figure 19.1 to a specific single record in the database, or to the null record $\varnothing$. For example, the tokens *south wind* would align to the record `wind direction: 06:00-21:00: mode=S`. How often is each token aligned to the same record as the previous token? How many transitions are there? How might a system learn to output *10 degrees* for the record `min=9`?

5. In sentence compression and fusion, we may wish to preserve contiguous sequences of tokens (n-grams) and/or dependency edges. Find five short news articles with headlines. For each headline, compute the fraction of bigrams that appear in the main text of the article. Then do a manual depenency parse of the headline. For each dependency edge, count how often it appears as a dependency edge in the main text. You may use an automatic dependency parser to assist with this exercise, but check the output, and focus on UD 2.0 dependency grammar, as described in chapter 11.

6. § 19.2.2 presents the idea of generating text from dependency trees, which requires **linearization**. Sometimes there are multiple ways that a dependency tree can be linearized. For example:

   (19.9) a. The sick kids stayed at home in bed.

   b. The sick kids stayed in bed at home.

   Both sentences have an identical dependency parse: both *home* and *bed* are (oblique) dependents of *stayed*.

   Identify two more English dependency trees that can each be linearized in more than one way, and try to use a different pattern of variation in each tree. As usual, specify your trees in the Universal Dependencies 2 style, which is described in chapter 11.

7. In § 19.3.2, we considered a pizza delivery service. Let's simplify the problem to take-out, where it is only necessary to determine the topping and confirm the order. The state is a tuple in which the first element is T if the topping is specified and ? otherwise, and the second element is either YES or NO, depending on whether the order has been confirmed. The actions are TOPPING? (request information about the topping) and CONFIRM? (request confirmation). The state transition function is:

$$p(s_t \mid s_{t-1} = (?, \text{NO}), a = \text{TOPPING?}) = \begin{cases} 0.9, & s_t = (\text{T}, \text{NO}) \\ 0.1, & s_t = (?, \text{NO}). \end{cases} \quad [19.19]$$

$$p(s_t \mid s_{t-1} = (?, \text{NO}), a = \text{CONFIRM?}) = \begin{cases} 1, & s_t = (?, \text{NO}). \end{cases} \quad [19.20]$$

$$p(s_t \mid s_{t-1} = (\text{T}, \text{NO}), a = \text{TOPPING?}) = \begin{cases} 1, & s_t = (\text{T}, \text{NO}). \end{cases} \quad [19.21]$$

$$p(s_t \mid s_{t-1} = (\text{T}, \text{NO}), a = \text{CONFIRM?}) = \begin{cases} 0.9, & s_t = (\text{T}, \text{YES}) \\ 0.1, & s_t = (\text{T}, \text{NO}). \end{cases} \quad [19.22]$$

Using the reward function defined in Equation 19.16, the discount $\gamma = 0.9$, and the initialization $V(s) = 0$, execute three iterations of Equation 19.17. After these three iterations, compute the optimal action in each state. You can assume that for the terminal states, $V(*, \text{YES}) = 0$, so you only need to compute the values for non-terminal states, $V(?, \text{NO})$ and $V(\text{T}, \text{NO})$.

8. There are several toolkits that allow you to train encoder-decoder translation models "out of the box", such as FAIRSEQ (Gehring et al., 2017), XNMT (Neubig et al., 2018), TENSOR2TENSOR (Vaswani et al., 2018), and OPENNMT (Klein et al., 2017).[6] Use one of these toolkits to train a chatbot dialogue system, using either the NPS dialogue corpus that comes with NLTK (Forsyth and Martell, 2007), or, if you are feeling more ambitious, the Ubuntu dialogue corpus (Lowe et al., 2015).

[6] `https://github.com/facebookresearch/fairseq`; `https://github.com/neulab/xnmt`; `https://github.com/tensorflow/tensor2tensor`; `http://opennmt.net/`