

Chapter 19

Text generation



This chapter ...

- 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;
- **Dialogue**, in which text is generated as part of an interactive conversation with one or more human participants.

Data-to-text generation

- Input : Structured records + Unstructured data
- Structured records **example** : Description of a weather forecast
- Unstructured perceptual data **example** : Raw image or video

Data-to-text generation steps

- All data-to-text systems share some of the same challenges.
 - 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**.

Data-to-text generation

- More recent systems are unified **models** that **are trained end-to-end using backpropagation**.
- **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**.
- **Solution:** both **latent variables** and **neural attention** have been proposed as solutions

Latent data-to-text

- Consider given a **dataset of texts** and **associated records**

$$\{(w^{(i)}, y^{(i)})\}_{i=1}^N$$

- The goal is to learn a model Ψ , so that
- \mathcal{V}^* is the set of strings over a discrete vocabulary
- θ is a vector of parameters

$$\hat{w} = \operatorname{argmax}_{w \in \mathcal{V}^*} \Psi(w, y; \theta)$$

- Relationship between w and y : the data y may contain dozens of records, and w may extend to several sentences

Latent **data-to-text** alignment

$$\{(w^{(i)}, y^{(i)}, z^{(i)})\}_{i=1}^N$$

- Let's decompose the **scoring function** Ψ , into subcomponents.
- Consider if given an **alignment**, specifies **which element** of y is expressed in each part of w , specifically, let z_m indicates the record aligned to word m .

$$\Psi(w, y; \theta) = \sum_{m=1}^M \psi_{w,y}(w_m, y_{z_m}) + \psi_w(w_m, w_{m-1}) + \psi_z(z_m, z_{m-1}).$$

- Given an observed set of **alignments**, the score for a generation can be written as sum of local scores

Latent **data-to-text** alignment

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

- Given an observed set of alignments, the score for a generation can be written as sum of local scores
- ψ_w can represent a **bigram language model**
- ψ_z can be tuned to **reward coherence**, such as the use of related records in nearby words.
- The parameters of this model could be learned from labeled data:

$$\{(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}, \tilde{\mathbf{z}}^{(i)})\}_{i=1}^N$$

Neural **data-to-text** generation

- The **encoder-decoder model** and **neural attention** can be applied to **data-to-text** generation, with the data acting as the source language.
- In data-to-text generation, the **attention mechanism** can **link** each part of the **generated text** back to a **record** in the data.

Data encoders: **discrete sets**

- In some types of **structured records**, all values are drawn from **discrete sets**.

Example : The birthplace of a person is from a **discrete** set of locations

- In such cases, **vector embeddings** can be estimated for each field and possible value

Example : One **vector embedding** for the **field** *BIRTHPLACE*, and another embedding for the **value** *BERKELEY CALIFORNIA*

- The **table of such embeddings** serves as the **encoding of structured record**
- It is also possible to **compress** the entire table into a **single vector** representation, by **pooling** across the embeddings of each **field** and **value**

Data encoders : Sequences

- Some types of **structured records** have a **natural ordering**.
- We can resemble this **natural ordering** as **series of events** in a game
- Each **event** is a single record, and **can be encoded** by a **concatenation** of vector representations for the **event type**, the **event field**, and the **event values**
- This **encoding** can then act as the **input layer** for a **recurrent neural network**, yielding a **sequence of vector representations**
- This sequence-based approach can work even in cases where there is no natural ordering over the records

Data encoders : Images

- The data-to-text goal for images is the **generation of text captions for images**
- Images are represented as **tensors**: A color image of 320×240 pixels would be stored as a tensor with $320 \times 240 \times 3$ intensity values.
- **Dominant** approach to image classification is to **encode images as vectors** using a **combination** of **convolution** and **pooling**
- **Alternative** approach is we can **apply a set of convolutional networks**, yielding **vector** representations for **different parts of the image**, which can then be **combined** using **neural attention**

Data encoders : Dominant Approach

In **dominant** , we **encode** images as **vectors** using a **combination** of **convolution** and **pooling**

- The **convolution** is applied across the vertical, horizontal, and color dimensions for images.
- By **pooling** the results of successive convolutions, the image is converted to a **vector representation**
- This **vector representation**, can then be fed into the **decoder** as the initial state similar to **sequence-to-sequence** translation mode

Data encoders : **Alternative Approach**

- Given a set of **CNN embeddings** of the data $\{z_r\}_{r=1}^R$, a **decoder state** h_m , an **attention vector** over the data can be computed. When generating word m of the output, attention is computed over the records

$$\begin{aligned}\psi_\alpha(m, r) &= \beta_\alpha \cdot f(\Theta_\alpha[h_m; z_r]) \\ \alpha_m &= g([\psi_\alpha(m, 1), \psi_\alpha(m, 2), \dots, \psi_\alpha(m, R)]) \\ c_m &= \sum_{r=1}^R \alpha_{m \rightarrow r} z_r,\end{aligned}$$

- f is an elementwise nonlinearity such as \tanh or ReLU ,
- g is either **SoftMax** or **elementwise sigmoid**.
- The weighted sum c_m gets included in the recurrent update to the decoder state, or in the emission probabilities

A **CNN** gets applied to a set of image locations, and the output at each location ℓ is represented with a vector z_ℓ . Attention then can be computed over the image locations.

Decoder

- Given the encoding, the decoder can function similar to in neural machine translation
- The decoder works using the **attention-weighted encoder representation** in the decoder recurrence and/or output computation
- **Problem** : Sometimes we need to generate words **that do not appear in the training vocabulary**;
- **Solution** : Such tokens can be generated in the text by **copying** them over from the input

Generation and Copy mechanism

- Let's introduce a **variable** $s_m \in \{\text{gen}, \text{copy}\}$, indicating whether token $w_m^{(t)}$ should be **generated** or **copied**. The **decoder probability** is then:

$$p(w^{(t)} \mid \mathbf{w}_{1:m-1}^{(t)}, \mathbf{Z}, s_m) = \begin{cases} \text{SoftMax}(\beta_{w^{(t)}} \cdot \mathbf{h}_{m-1}^{(t)}), & s_m = \text{gen} \\ \sum_{r=1}^R \delta(w_r^{(s)} = w^{(t)}) \times \alpha_{m \rightarrow r}, & s_m = \text{copy}, \end{cases}$$

- $\delta(w_r^{(s)} = w^{(t)})$ is **indicator** function, taking the value 1 if the text of record $w_r^{(s)}$ is identical to the target word $w^{(t)}$
- Probability of **copying** record r from source is $\delta(s_m = \text{copy}) \times \alpha_{m \rightarrow r}$, which is the product
- of **copy probability** by **local attention**.
- The attention weights α_m are computed from the **previous decoder state** \mathbf{h}_{m-1}
- The computation graph therefore remains a feedforward network, with recurrent paths such as

$$\mathbf{h}_{m-1}^{(t)} \rightarrow \alpha_m \rightarrow w_m^{(t)} \rightarrow \mathbf{h}_m^{(t)}$$

Gen-Copy mechanism **Alteration**

- For end-to-end training, we switch the variable s_m with a **gate** π_m
- This gate gets computed from a **two-layer feedforward network**, whose input consists of the **concatenation of the decoder state** $h_{m-1}^{(t)}$
- The **attention-weighted representation** of the **data** $c_m = \sum_{r=1}^R \alpha_{m \rightarrow r} z_r$

$$\pi_m = \sigma(\Theta^{(2)} f(\Theta^{(1)} [h_{m-1}^{(t)}; c_m]))$$

- The full generative probability at token m is then

$$p(w^{(t)} \mid w_{1:m}^{(t)}, \mathbf{Z}) = \pi_m \times \underbrace{\frac{\exp \beta_{w^{(t)}} \cdot h_{m-1}^{(t)}}{\sum_{j=1}^V \exp \beta_j \cdot h_{m-1}^{(t)}}}_{\text{generate}} + (1 - \pi_m) \times \underbrace{\sum_{r=1}^R \delta(w_r^{(s)} = w^{(t)}) \times \alpha_{m \rightarrow r}}_{\text{copy}}$$

Text-to-text generation

- Text-to-text generation includes problems of **summarization** and **simplification**.
- These problems can be approached in two ways:
 - 1. Through **encoder-decoder architecture**
 - 2. By **operating** directly on the **input text**.

Text-to-text generation : summarization

- Sentence **summarization** is the task of shortening a sentence while preserving its **meaning**
- Sentence **summarization** is closely related to sentence compression, in which the summary is produced by **deleting** words or phrases from the original
- Also, a **sentence** summary can also introduce **new words**.

Neural abstractive summarization

- **Sentence summarization** can be treated as a **machine translation** problem, using the **attentional encoder-decoder** translation model
- 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 boost the encoder-decoder model with the ability to **copy words directly from the source**

Long documents, fear of repetition

- **Problem :** When summarizing longer documents, an additional concern is that the summary not be repetitive, therefore, each part of the summary should cover new ground.
- **Solution :** This can be addressed by maintaining a vector of the sum total of all attention values thus far:

$$t_m = \sum_{n=1}^m \alpha_n$$

- This total can be used as an additional input to the computation of the attention weights, which enables the model to learn to prefer parts of the source which have not been attended to yet

$$\alpha_{m \rightarrow n} \propto \exp \left(\mathbf{v}_\alpha \cdot \tanh(\Theta_\alpha[\mathbf{h}_m^{(t)}; \mathbf{h}_n^{(s)}; t_m]) \right)$$

Long documents, fear of repetition

- To further encourage diversity in the generated summary, introduce a coverage loss to the objective function:

$$\ell_m = \sum_{n=1}^{M^{(s)}} \min(\alpha_{m \rightarrow n}, t_{m \rightarrow n})$$

- This loss will be low if α_m assigns little attention to words that already have large values in 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

Dialogue

- Dialogue systems are capable of **conversing** with a **human interlocutor**, often **to perform some task**, or **just to chat**
- Commercial systems such as **Alexa** and **Siri** have recently brought this technology into widespread use
- An **example** of Dialogue can be the following conversation to order pizza.



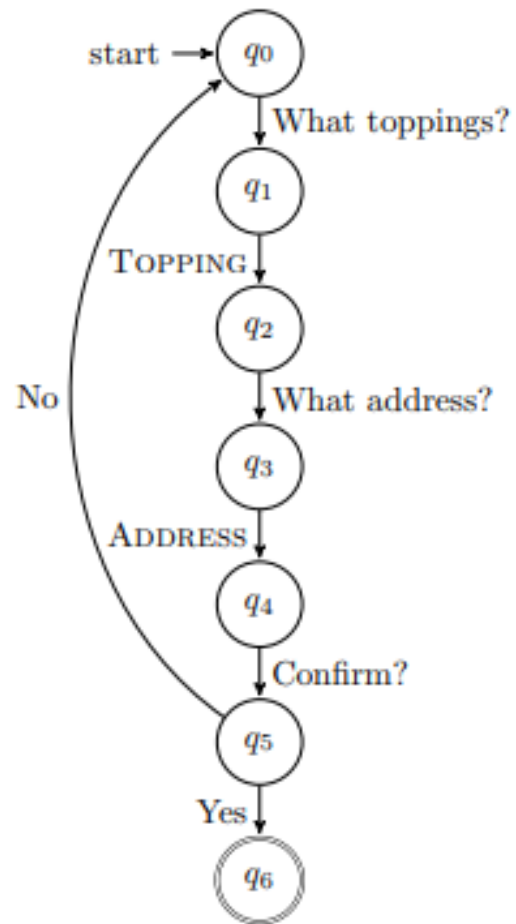
A: I want to order a pizza.
B: What toppings?
A: Anchovies.
B: Ok, what address?
A: The College of Computing building.
B: Please confirm: one pizza with artichokes, to be delivered to the College of Computing building.
A: No.
B: What toppings?

Finite-state & agenda-based dialogue systems

- **Finite-state automata** is a formal model of computation, in which string inputs and outputs are linked to transitions between a **finite number of discrete states**.
- This model fits **simple task-oriented dialogues**
- Dialogue can be represented with a **finite-state transducer**

- **Example** of **dialogue** and the associated **finite-state model**
- The TOPPING and ADDRESS are the two **slots** associated with the activity of ordering a pizza, which is called a **frame**.

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?

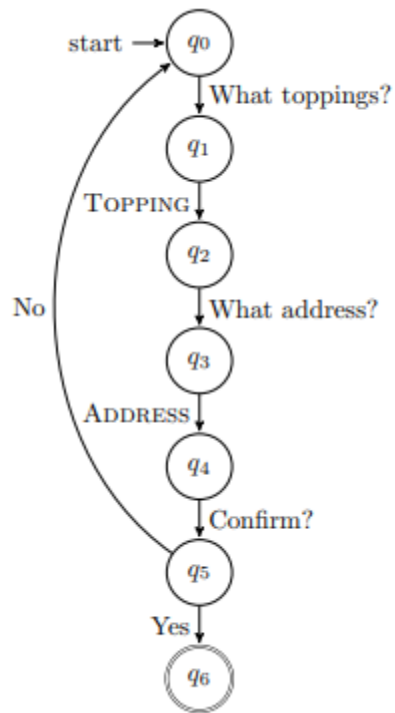


Finite-state & agenda-based dialogue systems

- **Problem:** In case the user wants to communicate more naturally using phrases like I'd, .. How should we approach?
- **Solution:** BIO-style sequence labeling. The tagger can be driven by a bi-directional RNN , similar to recurrent approaches to semantic role labeling

I'd like anchovies
O O B-TOPPING

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?



Dialogue : Neural chatbots

- **Chatbots** are systems that parry the user's input with a response that keeps the conversation going
- **Chatbots** can be built from the **Encoder-Decoder architecture**
- **Encoder** converts the user's input into a **vector**
- **Decoder** produces a sequence of words as a response

Dialogue : Neural chatbots

- **Problem** : Encoder-decoder models struggle to maintain **coherence** over longer conversations.
- **Solution** : Modeling the dialogue **context** recurrently.
- By creating **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**

Dialogue : Neural chatbots

- The **encoder-decoder architecture** can be integrated into task-oriented dialogue systems.
- Neural chatbots can be trained **end-to-end**
- The **user's turn** is analyzed by the **encoder**
- The **system output** is generated by the **decoder**.
- This architecture can be trained by **log-likelihood** using **backpropagation** or by more elaborate objectives, using **reinforcement learning**

Task-oriented **dialogue** : MDP and RNN

- The **Task-oriented** dialogue systems, typically involve set of 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**
- **RNN 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

Task-oriented **dialogue** : Element Embedding

- 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.
- The **decoder** is able with the ability to **refer to specific elements** in the knowledge base.

**Thank you
very much
for listening**