The Amazing World of Neural Language Generation

Part II: Neural Network Modeling for Generation

Yangfeng Ji

November 20, 2020

Department of Computer Science
University of Virginia
Basic Architecture of Neural NLG Models

From a (very) high-level viewpoint, neural NLG model can be formulated as with two fundamental components: **encoder** and **decoder**:

\[
\text{Input } x \xrightarrow{\text{Encoder}} \text{Decoder} \xrightarrow{\text{Text}} y
\]

where

- **Input**: A sequence of words \( x = (x_1, \cdots, x_m) \), \( m \) words
- **Output**: A sequence of words \( y = (y_1, \cdots, y_n) \), \( n \) words
From a (very) high-level viewpoint, neural NLG model can be formulated as with two fundamental components: encoder and decoder:

\[
\begin{array}{c}
\text{Input } x \\
\text{Encoder} \\
\text{Decoder} \\
\text{Text } y
\end{array}
\]

where

- **Input**: A sequence of words \( x = (x_1, \cdots, x_m) \), \( m \) words
- **Output**: A sequence of words \( y = (y_1, \cdots, y_n) \), \( n \) words

We will try to answer the following three questions with neural network modeling strategies:

- How to select *contextual information*?
- How to build better *latent representations*?
- How to incorporate *structural information*?
This part of the tutorial will cover the three major neural network modeling strategies for text generation.
This part of the tutorial will cover the three major neural network modeling strategies for text generation.
The simple implementation of the encoder-decoder framework is to realize each component with a recurrent neural network as illustrated in the following

where each $s./h.$ is a hidden state of the encoder/decoder recurrent neural network.
The simple implementation of the encoder-decoder framework is to realize each component with a recurrent neural network as illustrated in the following

where each $s./h.$ is a hidden state of the encoder/decoder recurrent neural network.

Simple extensions on the encoder side

- Bi-directional RNN
- Stacked LSTM
In general, an decoder can be implemented as an auto-regressive model, with the hidden state computed as

$$h_t = f(h_{t-1}, y_{t-1})$$ (1)

For generation, the probability of the word $h_C$ is computed as

$$\hat{y} = \text{softmax}(\text{output} > h_C)$$

where $\text{output}$ is a learnable weight matrix for the output layer.
In general, an decoder can be implemented as an auto-regressive model, with the hidden state computed as

\[ h_t = f(h_{t-1}, y_{t-1}) \]  

For generation, the probability of the word \( y_t \) is computed as

\[ p(y) = \text{softmax}(W_0 h_t) \]

where \( W_0 \in \mathbb{R}^{H \times V} \) is a learnable weight matrix for the output layer.
In general, an decoder can be implemented as an auto-regressive model, with the hidden state computed as

$$h_t = f(h_{t-1}, y_{t-1})$$  \hspace{1cm} (1)

For generation, the probability of the word $y_t$ is computed as

$$p(y) = \text{softmax}(W_oh_t)$$  \hspace{1cm} (2)

where $W_o \in \mathbb{R}^{H \times V}$ is a learnable weight matrix for the output layer.

Decoding algorithms will be discussed in the next part of the tutorial.
Contextual Information
Roadmap

- Encoder-Decoder Framework
  - Contextual Information
    - Attention Mechanism
  - Latent Representation
  - Structural Information
    - Copying Mechanism
    - Memory Modules
Attention mechanism (Bahdanau et al., 2015) provides a way of actively encoding context information from preceding text \( x = (x_1, \cdots, x_m) \) via hidden states \((s_1, \ldots, s_m)\)
Attention mechanism (Bahdanau et al., 2015) provides a way of actively encoding context information from preceding text \( x = (x_1, \cdots, x_m) \) via hidden states \((s_1, \ldots, s_m)\).

Attention weights
\[
\alpha_{t,i} = g(s_i, h_{t-1}) \in (0, 1)
\]
\[
c_t = \sum_{i=1}^{m} \alpha_{t,i} \cdot s_i
\]
Attention mechanism (Bahdanau et al., 2015) provides a way of actively encoding context information from preceding text $x = (x_1, \cdots, x_m)$ via hidden states $(s_1, \ldots, s_m)$.

- $c_t$ is a function of $h_{t-1}$, which means it dynamically changes at every time step of decoding.
- $c_t$ enables the decoder to be more selective on using contextual information.
- Widely used in many text generation tasks: response generation, story generation, and document summarization.

Attention weights

$$\alpha_{t,i} = g(s_i, h_{t-1}) \in (0, 1)$$

$$c_t = \sum_{i=1}^{m} \alpha_{t,i} \cdot s_i$$
Rush et al. (2015) use the attention mechanism in a feed-forward neural network for abstractive sentence summarization, where the neural network architecture is constructed as the following:
Rush et al. (2015) use the attention mechanism in a feed-forward neural network for abstractive sentence summarization, where the neural network architecture is constructed as the following

Diagram (b) represents the attention-based encoder, which use the input $x$ and a fixed context window $y_c = y_{i-c+1:i}$ to compute the attention weights

$$\alpha \propto \exp(\tilde{x}P y'_c) \quad \tilde{x}_i = \sum_{q=i-Q}^{i+Q} \tilde{x}_i/Q \quad \text{enc}(x, y_c) = \alpha^T \tilde{x} \quad (3)$$
Gu et al. (2016) propose a model called CopyNet to directly copy a phrases from input $x$ to output $y$,

$$p(y_t | h_t) = p_g(y_t | h_t) + p_c(y_t | h_t)$$

(4)

where $p_g(y_t | h_t)$ is a probability distribution defined on $\mathcal{V}$ and $p_c(y_t | h_t)$ is a probability distribution defined only on the input $x$.

In other words, it defines a mixture model with equal mixture coefficients.
See et al. (2017) propose a similar idea to copy words from input $x$ to output $y$ in text summarization. The probability of $w \in Y$ being the next word is

$$p(y_t = w \mid \cdot) = \beta p_g(y_t = w \mid \cdot) + (1 - \beta) \sum_{x_i \in x} \delta(w = x_i)\alpha_{t,i}$$  \hspace{1cm} (5)$$

Among the many implementation differences with (Gu et al., 2016), the work uses

A soft gate $\in (0, 1)$ to decide the probability of generation instead of copying. The probability of copying a word $F$ is defined by the attention weights associated with it.
See et al. (2017) propose a similar idea to copy words from input $x$ to output $y$ in text summarization. The probability of $w \in Y$ being the next word is

$$p(y_t = w \mid \cdot) = \beta p_g(y_t = w \mid \cdot) + (1 - \beta) \sum_{x_i \in x} \delta(w = x_i) \alpha_{t,i} \quad (5)$$

Among the many implementation differences with (Gu et al., 2016), the work uses

- a soft gate $\beta \in (0, 1)$ to decide the probability of generation instead of copying
See et al. (2017) propose a similar idea to copy words from input $x$ to output $y$ in text summarization. The probability of $w \in Y$ being the next word is

$$p(y_t = w \mid \cdot) = \beta p_g(y_t = w \mid \cdot) + (1 - \beta) \sum_{x_i \in x} \delta(w = x_i)\alpha_{t,i}$$

Among the many implementation differences with (Gu et al., 2016), the work uses

- a soft gate $\beta \in (0, 1)$ to decide the probability of generation instead of copying
- the probability of copying a word $w$ is defined by the attention weights associated with it
One example is proposed in (Clark et al., 2018) for entity-driven text (story) generation, where each memory cell is associated with a particular entity, to encode entity related information from context.

where $e.$ is a distributed representation of an entity
One example is proposed in (Clark et al., 2018) for entity-driven text (story) generation, where each memory cell is associated with a particular entity, to encode entity related information from context

where $e_i$ is a distributed representation of an entity

- Entity prediction $p(e = \text{EMNLP}) \propto \exp(h_{t-1}^T W_e e_{\text{EMNLP}} + w_ff(e))$, where $f$ is the surface feature related to this entity (Ji et al., 2017).
One example is proposed in (Clark et al., 2018) for entity-driven text (story) generation, where each memory cell is associated with a particular entity, to encode entity related information from context

where $e_e$ is a distributed representation of an entity

- **Entity prediction** $p(e = \text{EMNLP}) \propto \exp(h_{t-1}^T W_e e_{\text{EMNLP}} + w_f f(e))$, where $f$ is the surface feature related to this entity (Ji et al., 2017).
- **Dynamic entity updating** $e_{\text{EMNLP}}^{(\text{new})} \propto \delta_t e_{\text{EMNLP}}^{(\text{new})} + (1 - \delta_t) h_t$, where $\delta_t \in (0, 1)$ determines how much information should be encoded from $e_{\text{EMNLP}}^{(\text{new})}$. 
Latent Representation
Roadmap

Encoder-Decoder Framework

- Contextual Information
  - Attention Mechanism
  - Copying Mechanism
  - Memory Modules

- Latent Representation
  - Variational Autoencoders
  - Generative Adversarial Networks
  - Disentangled Representations
  - Pretrained Language Models

- Structural Information
One-page summary of variational autoencoder (Kingma and Welling, 2014)

\[ h = \text{Encoder}(x) \]
\[ z = h + \epsilon \sim \mathcal{N}(0, \text{diag}(2^2)) \]
\[ \tilde{x} = \text{Decoder}(z) \]

An impact of using VAE is that it produces a robust encoder for input \( x \) and enriches the hidden space \( H \).
One-page summary of variational autoencoder (Kingma and Welling, 2014)

A formulation of variational autoencoders for text generation

\[
\begin{align*}
  h &= \text{Encoder}(x) \\
  z &= h + \epsilon \quad \epsilon \sim \mathcal{N}(0, \text{diag}(\sigma^2)) \\
  \tilde{x} &= \text{Decoder}(z)
\end{align*}
\]
One-page summary of variational autoencoder (Kingma and Welling, 2014)

A formulation of variational autoencoders for text generation

\[
\begin{align*}
    h &= \text{Encoder}(x) \\
    z &= h + \epsilon \quad \epsilon \sim \mathcal{N}(0, \text{diag}(\sigma^2)) \\
    \tilde{x} &= \text{Decoder}(z)
\end{align*}
\]

An impact of using VAE is that it (1) produces a robust encoder for input \( x \) and (2) enriches the hidden space \( \mathcal{H} \).
An example application of variational autoencoder in language generation

(Bowman et al., 2016)
Variational Seq2seq Models

An example application of variational autoencoder in language generation

The mean and variance of latent variable $z$ is computed by the linear transformations of the last hidden states from the RNN encoder $s_m$.

(Bowman et al., 2016)
Variational Seq2seq Models

An example application of variational autoencoder in language generation

- The mean and variance of latent variable $z$ is computed by the linear transformations of the last hidden states from the RNN encoder $s_m$

- Other influential ideas from this work are KL cost annealing and adversarial evaluation

(Bowman et al., 2016)
A simple formulation of conditional VAE is proposed by Sohn et al. (2015), which initially was used in computer vision.

In text generation, consider $x$, $y$ and $z$ are input texts, output texts, and latent representations of input texts.

- **Generation network:** $p_\theta(y \mid x, z)$, where $z \sim p_\theta(z \mid x)$
- **Inference network:** $q_\phi(z \mid x, y)$
- **Example applications:** in response generation, latent variable $z$ indicates whether the next utterance is a generic response (Shen et al., 2017)
The basic pipeline of GAN is described in the following pipeline:

- Input $x$
- Latent variable $z$
- Generator $G$
- Discriminator $D$
- Prediction $x'$

- One goal is to learn a generator $G$ that can generate text $x'$ with the same quality as $x$ (in other words, to “fool” the discriminator).
- Potential applications are to adopt the framework as one component in other task-specific generation tasks (e.g., style transfer).
As a straightforward application of adversarial learning is to replace the generator with a sequence-to-sequence model as we discussed before, as proposed by Li et al. (2017)

- The decoder is to generate a response $y'$ based on input context $x$
- The discriminator is to predict whether a response is generated by humans or the decoder
One way to utilize a discriminator is to learn disentangled representations and make sure latent representations encoding the *expected attributes* for generation.
One way to utilize a discriminator is to learn disentangled representations and make sure latent representations encoding the *expected attributes* for generation.

An example of learning disentangled representations is proposed by Hu et al. (2017), with

- **Encoder**
  \[(z) = \text{Encoder}(x)\]

- **Decoder**
  \[\hat{x} = \text{Decoder}(z, c)\]

- **Discriminator**
  \[c = \text{Dis}(\hat{x})\]

where \(c\) encodes the attributes of a text (e.g., sentiment categories, formality). Other work on learning disentangled representations include (Bao et al., 2019; Cheng et al., 2020)
Vaswani et al. (2017): “Attention mechanism . . . , allowing modeling of dependencies without regard to their distance in the input or output sequences.”

Recent applications of transformers simply use them as basic building blocks, just like the way of using LSTM
GPT (Radford et al., 2018) is trained simply by predicting the next words with

\[ h_0 = W_e x_{t-k:t-1} + W_p \]

\[ h_l = \text{transformer\_block}(h_{l-1}) \quad \forall l \in [1, 12] \]

\[ p(y_t \mid h_n) = \text{softmax} (W_0 h_n) \]

where \( W_e \) and \( W_p \) are the word and position embeddings.

Figure 1: (left) Transformer architecture and training objectives used in this work. (right) Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.
Besides the generative models like GPT-2 and its variants, we can also use BERT for text generation. One example from (Chen et al., 2020b) is that BERT can be used as a teacher model to help train a sequence-to-sequence models for better output probability.
Structural Information
Roadmap

Encoder-Decoder Framework

Contextual Information
- Attention Mechanism
- Copying Mechanism
- Memory Modules

Latent Representation
- Variational Autoencoders
- Generative Adversarial Networks
- Disentangled Representations
- Pretrained Language Models

Structural Information
- Neural Templates
- Syntactic
- Semantic
An extension of variational autoencoders is to incorporate sequential information in latent variables. For example, Wiseman et al. (2018) propose a semi-Markov model on the latent variable sequence \( z = (z_1, \ldots, z_n) \) to capture dependency among adjacent words for text-to-data generation.

**Source Entity:** Cotto

type[coffee shop], rating[3 out of 5], food[English], area[city centre], price[moderate], near[The Portland Arms]

**System Generation:**
Cotto is a coffee shop serving English food in the moderate price range. It is located near The Portland Arms. Its customer rating is 3 out of 5.

**Neural Template:**

| The ... | is an expensive ... | providing serving offering ... |
| food ... | in the ... | price range ... |
| cuisine foods ... | and has a ... | price bracket pricing ... |
| located in the ... | located near ... | ... |
| Its customer rating is ... | Their customer rating is ... |

A neural hidden semi-Markov model decoder

- Transition distribution: \( p(z_{t+1} \mid z_t, x) \)
- Length distribution: \( p(l_{t+1} \mid z_{t+1}) \)
- Emission distribution: \( p(y_{t-l_t+1:t} \mid z_t = k, l_t = l, x) \)
An example of inputs is an AMR graph, where the task is to generate a text with the same meaning as the input graph.
An example of inputs is an AMR graph, where the task is to generate a text with the same meaning as the input graph.

- The whole model is a layer-wise LSTM, within each layer LSTM runs on the linearized graph, and between layers the hidden states are updated with neighbor nodes from the previous layer.
An example of inputs is an AMR graph, where the task is to generate a text with the same meaning as the input graph.

- The whole model is a layer-wise LSTM, within each layer LSTM runs on the linearized graph, and between layers the hidden states are updated with neighbor nodes from the previous layer.
- The idea of linearizing an AMR graph (using depth-first search) is proposed by Konstas et al. (2017)
An example of inputs is an AMR graph, where the task is to generate a text with the same meaning as the input graph.

- The whole model is a layer-wise LSTM, within each layer LSTM runs on the linearized graph, and between layers the hidden states are updated with neighbor nodes from the previous layer.
- The idea of linearizing an AMR graph (using depth-first search) is proposed by Konstas et al. (2017).
- Other ideas of incorporating AMR graphs: Graph neural network (Chen et al., 2020a)
A straightforward application of GPT is the response generation proposed by Gupta et al. (2020). The prediction is still done by word-by-word prediction, where the input sequence and output sequence are

\[ x = (\text{Dialogue context, Response frames, Response text}) \]

\[ y = (\langle \text{MASK}\rangle, \text{Response frames, Response text}) \]

\[ (10) \]

\[ 1 \text{A semantic frame example \textit{Perception}: hear, say, see, smell, feel} \]
Summary

Encoder-Decoder Framework

Contextual Information
- Attention Mechanism
- Copying Mechanism
- Memory Modules

Latent Representation
- Variational Autoencoders
- Generative Adversarial Networks
- Disentangled Representations
- Pretrained Language Models

Structural Information
- Neural Templates
- Syntactic
- Semantic

Disentangled Representations
Generative Adversarial Networks
Variational Autoencoders
Summary

Encoder-Decoder Framework

- Contextual Information
- Latent Representation
- Structural Information


