The Amazing World of Neural Language Generation

Part II: Neural Network Modeling for Text Generation

Yangfeng Ji

November 20, 2020

Department of Computer Science
University of Virginia

UNIVERSITY of VIRGINIA | ENGINEERING

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**:

$$\text{Input } \boldsymbol{x} \longrightarrow \boxed{\text{Encoder}} \longrightarrow \boxed{\text{Decoder}} \longrightarrow \text{Text } \boldsymbol{y}$$

where

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

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**:

$$\text{Input } \boldsymbol{x} \longrightarrow \boxed{\text{Encoder}} \longrightarrow \boxed{\text{Decoder}} \longrightarrow \text{Text } \boldsymbol{y}$$

where

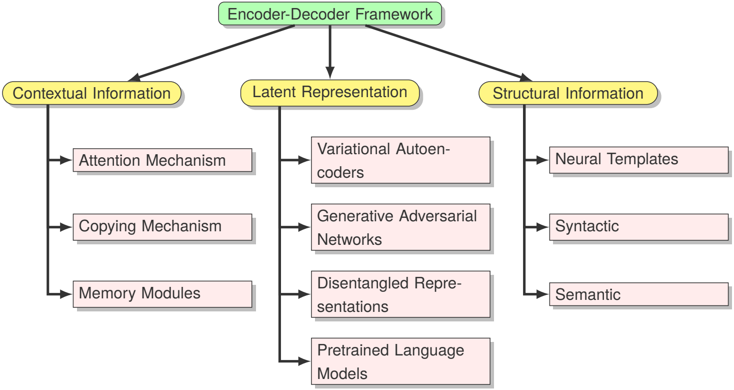
- Input: A sequence of words $\boldsymbol{x} = (x_1, \cdots, x_m)$, m words
- Output: A sequence of words $\boldsymbol{y} = (y_1, \cdots, y_n)$, n words

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?

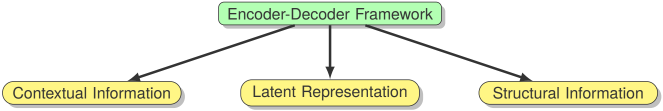
Roadmap

In this part, we will cover the three major neural network modeling strategies for text generation

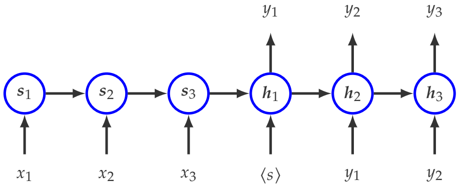


Roadmap

In this part, we will cover the three major neural network modeling strategies for text generation



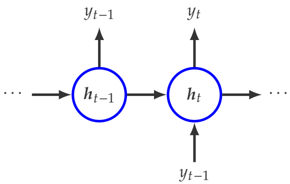
The simple implementation of the encoder-decoder framework is to realize each component with a recurrent neural network as illustrated in the following



where each $\boldsymbol{s}_\cdot/\boldsymbol{h}_\cdot$ is a hidden state of the encoder/decoder recurrent neural network

In general, an decoder can be implemented as an auto-regressive model, with the hidden state computed as

$$\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}, y_{t-1}) \tag{1}$$



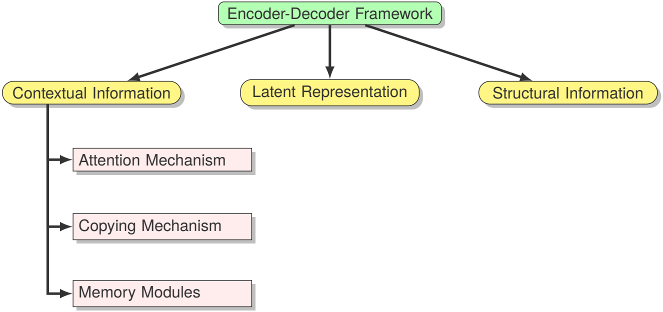
For generation, the probability of the word y_t is computed as

$$p(y) = \text{softmax}(\boldsymbol{W}_o \boldsymbol{h}_t + \boldsymbol{b}_o) \tag{2}$$

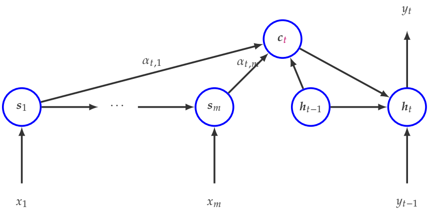
where $\boldsymbol{W}_o \in \mathbb{R}^{H \times V}$ is a learnable weight matrix for the output layer and $\boldsymbol{b}_o \in \mathbb{R}^V$ is the bias item.

Roadmap

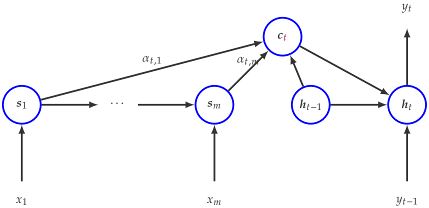
In this part, we will cover the three major neural network modeling strategies for text generation



Attention mechanism (Bahdanau et al., 2015) provides a way of actively encoding context information from preceding text $\boldsymbol{x} = (x_1, \cdots, x_m)$



Attention mechanism (Bahdanau et al., 2015) provides a way of actively encoding context information from preceding text $\boldsymbol{x} = (x_1, \cdots, x_m)$



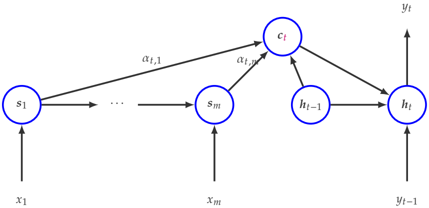
Attention weights

$$\alpha_{t,i} = g(\boldsymbol{s}_i, \boldsymbol{h}_{t-1}) \in (0,1)$$

$$\text{s.t.} \sum_{i=1}^{m} \alpha_{t,i} = 1;$$

$$\boldsymbol{c}_t = \sum_{i=1}^{m} \alpha_{t,i} \cdot \boldsymbol{s}_i$$

Attention mechanism (Bahdanau et al., 2015) provides a way of actively encoding context information from preceding text $\boldsymbol{x} = (x_1, \cdots, x_m)$



Attention weights

$$\alpha_{t,i} = g(\boldsymbol{s}_i, \boldsymbol{h}_{t-1}) \in (0,1)$$
$$\text{s.t.} \sum_{i=1}^{m} \alpha_{t,i} = 1;$$
$$\boldsymbol{c}_t = \sum_{i=1}^{m} \alpha_{t,i} \cdot \boldsymbol{s}_i$$

Additional comments

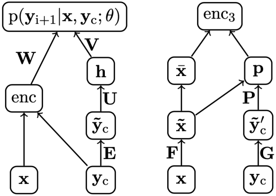
- $\boldsymbol{c}_t$ is a function of $\boldsymbol{h}_{t-1}$, which means it dynamically changes at every time step of decoding
- $\boldsymbol{c}_t$ enables the decoder to be more selective on using contextual information

The basic idea is proposed in Bahdanau et al. (2015) with a specific implementation of computing attentions

- Initially designed for machine translation
- Widely used in any generative tasks
  - Story generation
  - Response generation
  - Document summarization
- The idea of attention is fundamental in encoding contextual information for text generation

Attention Mechanism in FFNs

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



(a) (b)

Attention Mechanism in FFNs

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

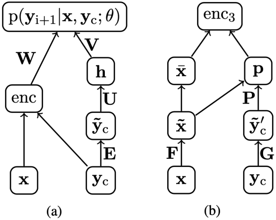


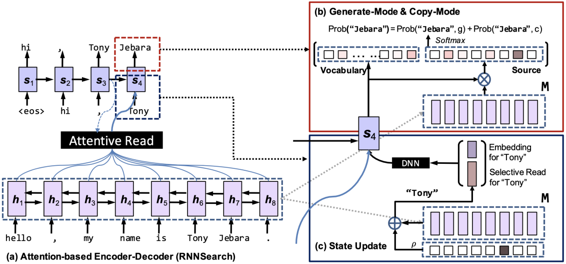
Diagram (b) represents the attention-based encoder, which use the input x and a fixed context window $y_c = y_{i-c+1:i}$ to compute the attention weights

$$\alpha \propto \exp(\tilde{x} P y'_c) \quad \bar{x}_i = \sum_{q=i-Q}^{i+Q} \tilde{x}_i / Q \quad \mathrm{enc}(x, y_c) = \alpha^\top \bar{x} \tag{3}$$

Gu et al. (2016) propose a model called CopyNet to directly copy a phrases from input $\boldsymbol{x}$ to output $\boldsymbol{y}$,

$$p(y_t \mid \boldsymbol{h}_t) = p_g(y_t \mid \boldsymbol{h}_t) + p_c(y_t \mid \boldsymbol{h}_t) \tag{4}$$

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



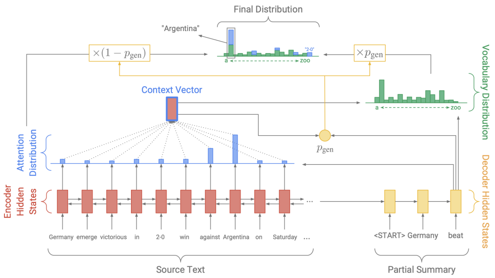
In other words, it defines a mixture model with equal mixture coefficients.

Copying Mechanism: Pointer-Generator Networks

See et al. (2017) propose a similar idea to copy words from input $\boldsymbol{x}$ to output $\boldsymbol{y}$ in text summarization. The probability of $w \in \mathcal{Y}$ being the next word is

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

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

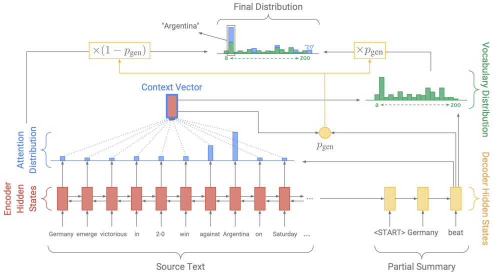


Copying Mechanism: Pointer-Generator Networks

See et al. (2017) propose a similar idea to copy words from input $\boldsymbol{x}$ to output $\boldsymbol{y}$ in text summarization. The probability of $w \in \mathcal{Y}$ being the next word is

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

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



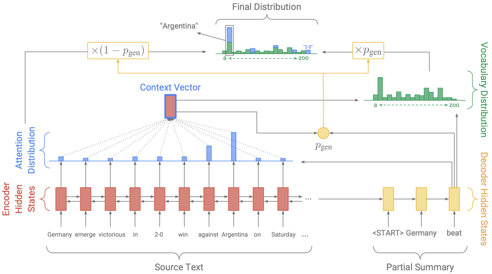
- a soft switch $\beta \in (0, 1)$ to decide the probability of generation instead of copying

Copying Mechanism: Pointer-Generator Networks

See et al. (2017) propose a similar idea to copy words from input $\boldsymbol{x}$ to output $\boldsymbol{y}$ in text summarization. The probability of $w \in \mathcal{Y}$ being the next word is

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

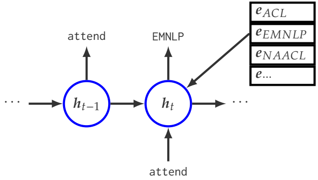
Among the many implementation differences with (Gu et al., 2016), the work uses



- a soft switch $\beta \in (0, 1)$ to decide the probability of generation instead of copying
- the probability of copy a word w is defined by the attention weights associated with it

Memory Modules

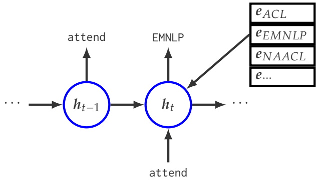
The idea of using memory modules is to have a set of individual memory cells (distributed representations) to memorize some particular information from context. One example is proposed in (Clark et al., 2018) for entity-driven text (story) generation, where each memory cell is associated with a particular entity, to encode entity related information from context



where $e_{\cdot}$ is a distributed representation of an entity

Memory Modules

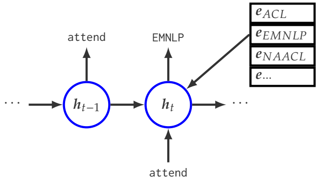
The idea of using memory modules is to have a set of individual memory cells (distributed representations) to memorize some particular information from context. One example is proposed in (Clark et al., 2018) for entity-driven text (story) generation, where each memory cell is associated with a particular entity, to encode entity related information from context



where $\boldsymbol{e}_{\cdot}$ is a distributed representation of an entity

- Entity prediction $p(e = \texttt{EMNLP}) \propto \exp(\boldsymbol{h}_{t-1}^{\top} W_e \boldsymbol{e}_{\texttt{EMNLP}} + \boldsymbol{w}_f \boldsymbol{f}(e))$, where $\boldsymbol{f}$ is the surface feature related to this entity (Ji et al., 2017).

The idea of using memory modules is to have a set of individual memory cells (distributed representations) to memorize some particular information from context. 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

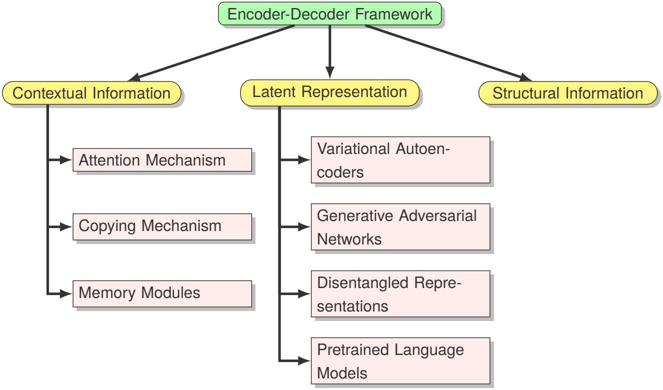


where $\boldsymbol{e}_{\cdot}$ is a distributed representation of an entity

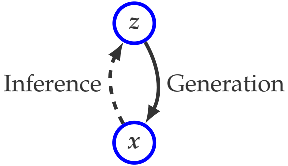
- Entity prediction $p(e = \text{EMNLP}) \propto \exp(\boldsymbol{h}_{t-1}^{\top} \boldsymbol{W}_e \boldsymbol{e}_{\text{EMNLP}} + \boldsymbol{w}_f \boldsymbol{f}(e))$, where $\boldsymbol{f}$ is the surface feature related to this entity (Ji et al., 2017).
- Dynamic entity updating $\boldsymbol{e}_{\text{EMNLP}}^{(\text{new})} \propto \delta_t \boldsymbol{e}_{\text{EMNLP}}^{(\text{new})} + (1 - \delta_t)\boldsymbol{h}_t$, where $\delta_t \in (0, 1)$ determines how much information should be encoded from $\boldsymbol{e}_{\text{EMNLP}}^{(\text{new})}$.

Roadmap

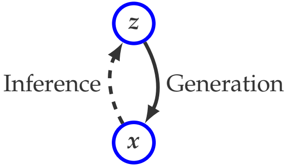
In this part, we will cover the three major neural network modeling strategies for text generation



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



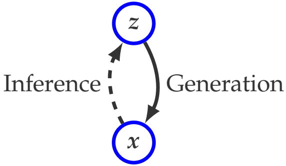
One-page summary of variational autoencoder (Kingma and Welling, 2014)



A formulation of variational autoencoders for text generation

$$
\begin{aligned}
\boldsymbol{h} &= \text{Encoder}(\boldsymbol{x}) && (6)\\
\boldsymbol{z} &= \boldsymbol{h} + \boldsymbol{\epsilon} \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \text{diag}(\boldsymbol{\sigma}^2)) && (7)\\
\tilde{\boldsymbol{x}} &= \text{Decoder}(\boldsymbol{z}) && (8)
\end{aligned}
$$

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

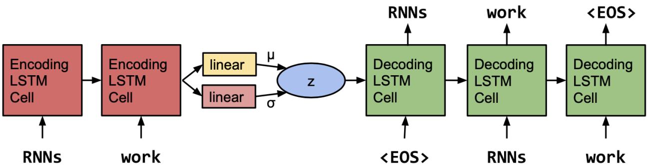


A formulation of variational autoencoders for text generation

$$
\begin{aligned}
\boldsymbol{h} &= \text{Encoder}(\boldsymbol{x}) \qquad (6)\\
\boldsymbol{z} &= \boldsymbol{h} + \boldsymbol{\epsilon} \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \text{diag}(\boldsymbol{\sigma}^2)) \qquad (7)\\
\tilde{\boldsymbol{x}} &= \text{Decoder}(\boldsymbol{z}) \qquad (8)
\end{aligned}
$$

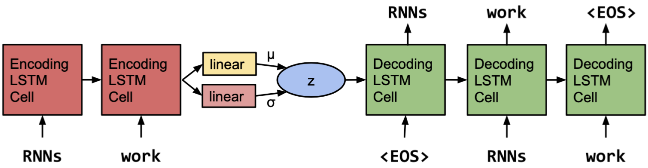
An impact of using VAE is that it (1) produces a robust encoder for input $\boldsymbol{x}$ and (2) enriches the hidden space $\mathcal{H}$.

An example application of variational autoencoder in language generation



(Bowman et al., 2016)

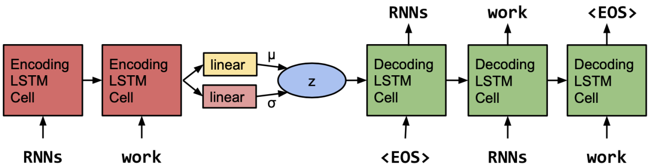
An example application of variational autoencoder in language generation



- 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)

An example application of variational autoencoder in language generation

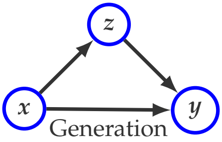


- The mean and variance of latent variable z is computed by the linear transformations of the last hidden states from the RNN encoder s_m
- Other influential ideas from this work are *KL cost annealing* and *adversarial evaluation*

(Bowman et al., 2016)

Conditional VAE

A simple formulation of conditional VAE is proposed by Sohn et al. (2015), which initially was used in computer vision. Unlike the VAE model, the input and output in this case are different

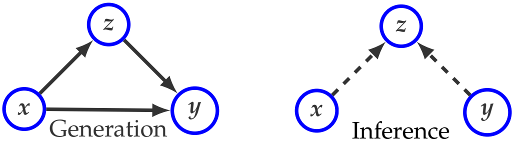


In text generation, consider $\boldsymbol{x}$, $\boldsymbol{y}$ and $\boldsymbol{z}$ are input texts, output texts, and latent representations of input texts

- Generation network: $p_\theta(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{z})$, where $\boldsymbol{z} \sim p_\theta(\boldsymbol{z} \mid \boldsymbol{x})$

Conditional VAE

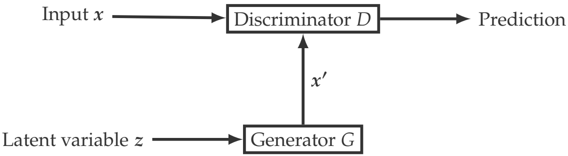
A simple formulation of conditional VAE is proposed by Sohn et al. (2015), which initially was used in computer vision. Unlike the VAE model, the input and output in this case are different



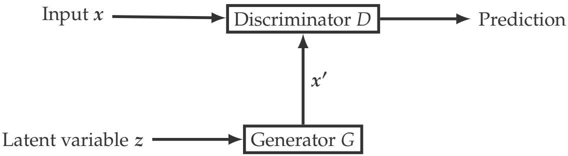
In text generation, consider $\boldsymbol{x}$, $\boldsymbol{y}$ and $\boldsymbol{z}$ are input texts, output texts, and latent representations of input texts

- Generation network: $p_\theta(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{z})$, where $\boldsymbol{z} \sim p_\theta(\boldsymbol{z} \mid \boldsymbol{x})$
- Inference network: $q_\phi(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{y})$

The basic pipeline of GAN is described in the following pipeline



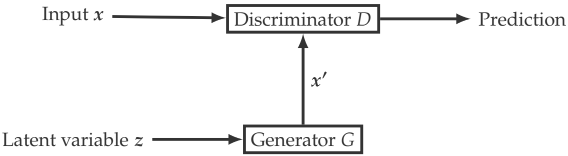
The basic pipeline of GAN is described in the following pipeline



During learning, the discriminator D will learn (with $\max_D$)to distinguish the real samples and the samples from the generator G, while the generator G will learn to "fool" the discriminator D (with $\min_G$).

- One goal is to learn a generator G that can generate text $\boldsymbol{x}'$ with the same quality as $\boldsymbol{x}$ (in other words, to "fool" the discriminator)

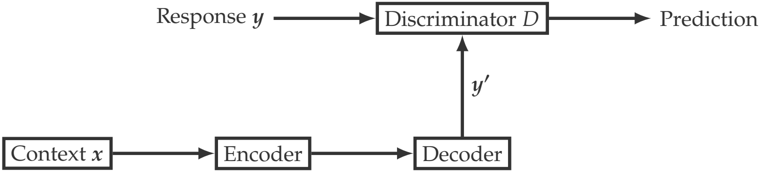
The basic pipeline of GAN is described in the following pipeline



During learning, the discriminator D will learn (with $\max_D$)to distinguish the real samples and the samples from the generator G, while the generator G will learn to "fool" the discriminator D (with $\min_G$).

- One goal is to learn a generator G that can generate text $\boldsymbol{x}'$ with the same quality as $\boldsymbol{x}$ (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)



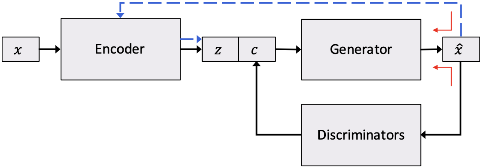
- The decoder is to generate a response y' based on input context x
- The discriminator is to predict whether a response is generated by humans or the decoder

One way to utilize a discriminator is to learn disentangled representations and make sure latent representations encoding expected attributes for generation.

One way to utilize a discriminator is to learn disentangled representations and make sure latent representations encoding 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 $c = \text{Dis}(\hat{x})$

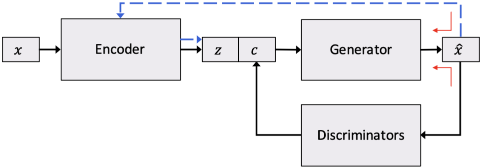


where c encodes the attributes of a text (e.g., sentiment categories, formality).

One way to utilize a discriminator is to learn disentangled representations and make sure latent representations encoding expected attributes for generation.

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

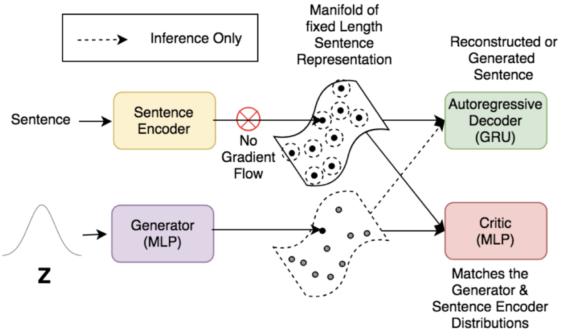
- Encoder ($\boldsymbol{z}$) = Encoder($\boldsymbol{x}$)
- Decoder
  $\hat{\boldsymbol{x}} = \text{Decoder}(\boldsymbol{z}, \boldsymbol{c})$
- Discriminator $\boldsymbol{c} = \text{Dis}(\hat{\boldsymbol{x}})$



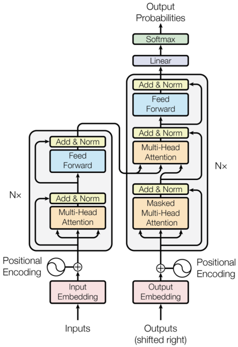
where $\boldsymbol{c}$ encodes the attributes of a text (e.g., sentiment categories, formality).

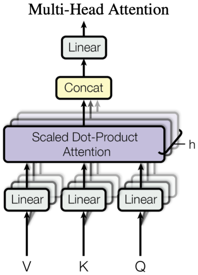
During generation, we can rewrite the code $\boldsymbol{c}$ to generate a text with expected attributes.

Another idea of using GAN is to leverage pre-trained sentence representation models and train a better model for text generation. The example along this line is proposed by Subramanian et al. (2018).



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

Generative Pre-trained Transformers (GPT)

GPT (Radford et al., 2018) is trained simply by predicting the next words with

$$\begin{aligned} \boldsymbol{h}_0 &= \boldsymbol{W}_e \boldsymbol{x}_{t-k:t-1} + \boldsymbol{W}_p \\ \boldsymbol{h}_l &= \text{transformer\_block}(\boldsymbol{h}_{l-1})\ \forall l \in [1, 12] \\ p(y_t \mid \boldsymbol{h}_n) &= \text{softmax}\left(\boldsymbol{W}_o \boldsymbol{h}_n\right) \end{aligned} \tag{9}$$

where $\boldsymbol{W}_e$ and $\boldsymbol{W}_p$ are the word and position embeddings.

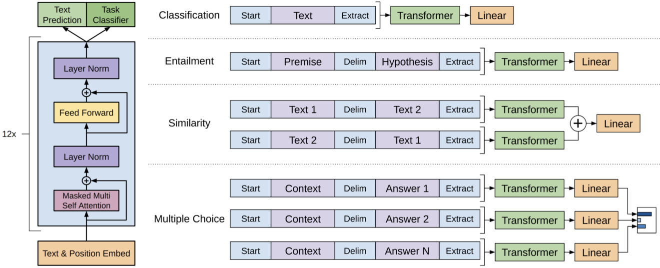
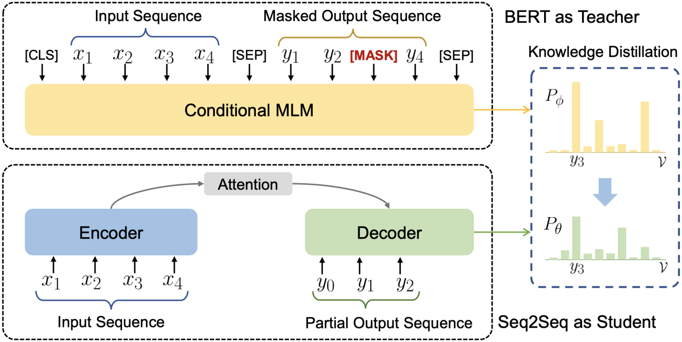


Figure 1: **(left)** Transformer architecture and training objectives used in this work. **(right)** Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

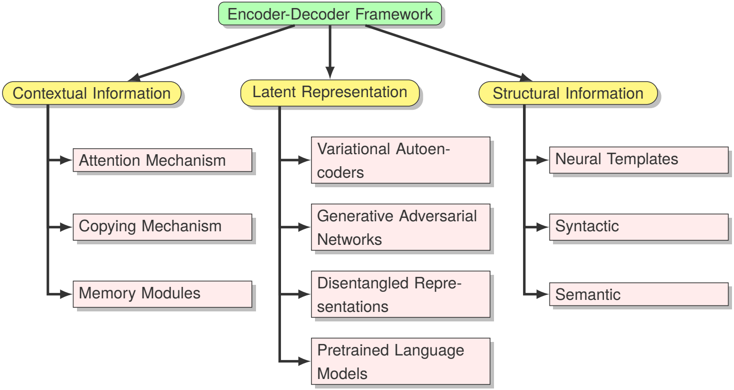
Pre-trained Models as Teachers

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



Roadmap

In this part, we will cover the three major neural network modeling strategies for text generation



Generation with Neural Templates

An extension of variational antoencoders is to incorporate sequential information in latent variables. For example, Wiseman et al. (2018) propose a semi-Markov model on the latent variable sequence $\boldsymbol{z} = (z_1, \ldots, z_n)$ 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:
<u>Cotto</u> is a <u>coffee shop</u> serving <u>English</u> food in the <u>moderate</u> price range. It is located near <u>The Portland Arms</u>. Its customer rating is <u>3 out of 5</u>.

Neural Template:

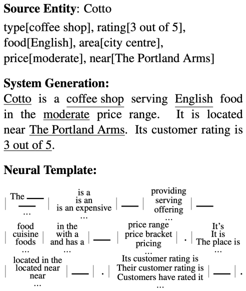
| The ___ / ___ / … | is a / is an / is an expensive / … | ___ | providing / serving / offering / … | ___ |

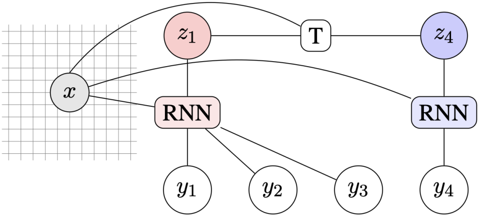
| food / cuisine / foods / … | in the / with a / and has a / … | ___ | price range / price bracket / pricing / … | . | It's / It is / The place is / … |

| located in the / located near / near / … | ___ | . | Its customer rating is / Their customer rating is / Customers have rated it / … | ___ | .

Generation with Neural Templates

An extension of variational antoencoders is to incorporate sequential information in latent variables. For example, Wiseman et al. (2018) propose a semi-Markov model on the latent variable sequence $\boldsymbol{z} = (z_1, \ldots, z_n)$ to capture dependency among adjacent words for text-to-data generation.



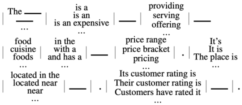


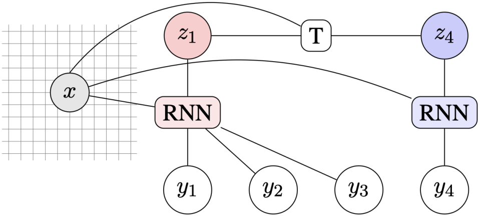
An extension of variational antoencoders is to incorporate sequential information in latent variables. For example, Wiseman et al. (2018) propose a semi-Markov model on the latent variable sequence $\boldsymbol{z} = (z_1, \ldots, z_n)$ to capture dependency among adjacent words for text-to-data generation.

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:



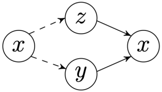


A neural hidden semi-Markov model decoder

- Transition distribution $p(z_{t+1} \mid z_t, \boldsymbol{x})$
- Length distribution $p(l_{t+1} \mid z_{t+1})$
- Emission distribution $p(\boldsymbol{y}_{t-l_t+1:t} \mid z_t = k, l_t = l, \boldsymbol{x})$

Generation with Syntactic Exemplar

As another example, an extension of latent variable models for text generation, Chen et al. (2019) introduce a structural constraint on latent variables for paraphrase generation

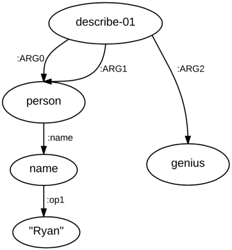


where z and c are two latent variable, one for syntax and the other for semantics.

$$p(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{c}) = p(\boldsymbol{c})p(\boldsymbol{z}) \prod_{t=1}^{T} p(x_t \mid x_{t-1}, \boldsymbol{z}, \boldsymbol{c}) \tag{10}$$

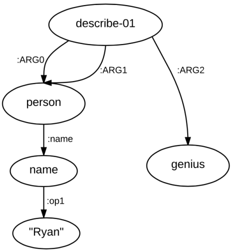
Ideally, $\boldsymbol{y}$ encodes semantic information from $\boldsymbol{x}$ and $\boldsymbol{z}$ encodes syntactic information.

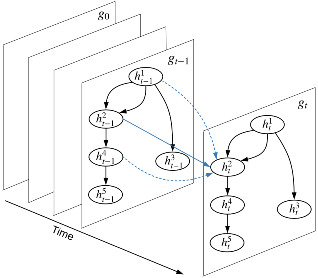
An example of inptus is an AMR graph, where the task is to generate a text with the same meaning as the input graph.



Graph-to-Sequence Generation

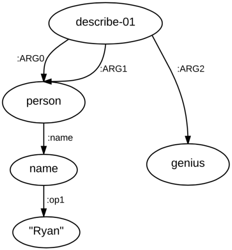
An example of inptus is an AMR graph, where the task is to generate a text with the same meaning as the input graph.

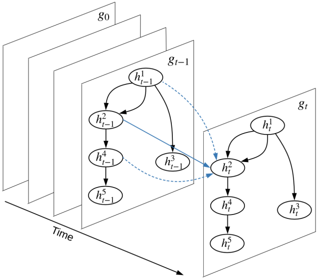




Graph-to-Sequence Generation

An example of inptus is an AMR graph, where the task is to generate a text with the same meaning as the input graph.

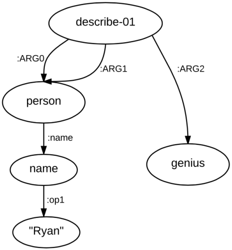


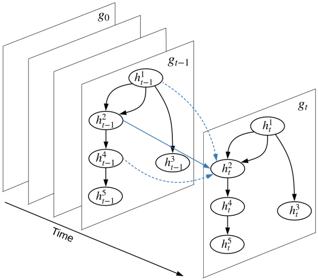


- the graph structure is fed to a recurrent graph encoder, which is based on LSTM with the following state updating equations, for node j at layer t

$$\boldsymbol{h}_j^{\text{in}} = \sum_{(i,j,l)\in\text{Incoming-Edges}(j)} \boldsymbol{h}_t^i \quad \boldsymbol{h} \tag{11}$$

An example of inptus is an AMR graph, where the task is to generate a text with the same meaning as the input graph.





- the graph structure is fed to a recurrent graph encoder, which is based on LSTM with the following state updating equations, for node j at layer t

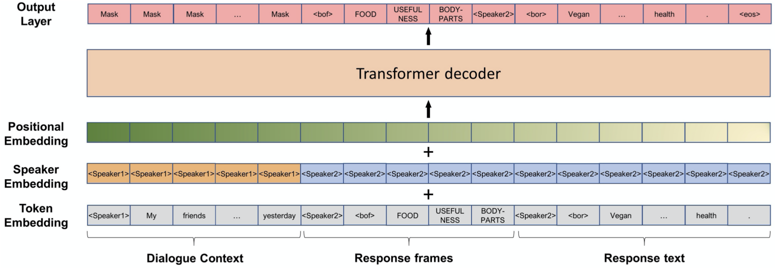
$$\boldsymbol{h}_j^{\text{in}} = \sum_{(i,j,l)\in\text{Incoming-Edges}(j)} \boldsymbol{h}_t^i \quad \boldsymbol{h} \tag{11}$$

- in general, we can also graph neural networks (GNNs) to encode graph structures e.g., (Chen et al., 2020a)

Dialogue Generation with Semantic Exemplars

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 [1]

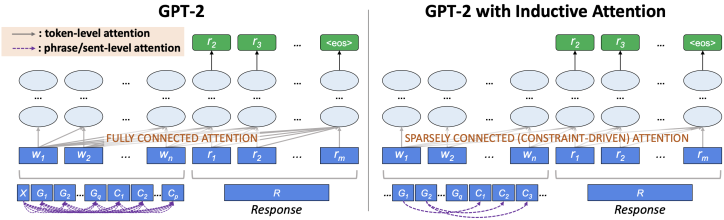
$$\begin{aligned} x &= (\text{Dialogue context}, \text{Response frames}, \text{Response text}) \\ y &= (\langle\text{MASK}\rangle, \text{Response frames}, \text{Response text}) \end{aligned} \tag{12}$$



[1] A semantic frame example PERCEPTION: `hear, say, see, smell, feel`

Controllable Grounded Response Generation

The input sequence contains the original context x, grounding G, and constraints C



This work proposes inductive (sparse and ordinal) attentions to help focus on constraints.

(Wu et al., 2020)

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural Machine Translation by Jointly Learning to Align and Translate. In *International Conference on Learning Representations*. arXiv: 1409.0473.

Bowman, S. R., Vilnis, L., Vinyals, O., Dai, A., Jozefowicz, R., and Bengio, S. (2016). Generating Sentences from a Continuous Space. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 10–21, Berlin, Germany. Association for Computational Linguistics.

Chen, M., Tang, Q., Wiseman, S., and Gimpel, K. (2019). Controllable Paraphrase Generation with a Syntactic Exemplar. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5972–5984, Florence, Italy. Association for Computational Linguistics.

Chen, Y., Wu, L., and Zaki, M. J. (2020a). Reinforcement Learning Based Graph-to-Sequence Model for Natural Question Generation. *arXiv:1908.04942 [cs]*. arXiv: 1908.04942.

Chen, Y.-C., Gan, Z., Cheng, Y., Liu, J., and Liu, J. (2020b). Distilling Knowledge Learned in BERT for Text Generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7893–7905, Online. Association for Computational Linguistics.

Clark, E., Ji, Y., and Smith, N. A. (2018). Neural Text Generation in Stories Using Entity Representations as Context. 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 2250–2260, New Orleans, Louisiana. Association for Computational Linguistics.

Gu, J., Lu, Z., Li, H., and Li, V. O. (2016). Incorporating Copying Mechanism in Sequence-to-Sequence Learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1631–1640, Berlin, Germany. Association for Computational Linguistics.

Gupta, P., Bigham, J. P., Tsvetkov, Y., and Pavel, A. (2020). Controlling Dialogue Generation with Semantic Exemplars. *arXiv:2008.09075 [cs]*. arXiv: 2008.09075.

Hu, Z., Yang, Z., Liang, X., Salakhutdinov, R., and Xing, E. P. (2017). Toward Controlled Generation of Text. volume 70 of *Proceedings of Machine Learning Research*, pages 1587–1596, International Convention Centre, Sydney, Australia. PMLR.

Ji, Y., Tan, C., Martschat, S., Choi, Y., and Smith, N. A. (2017). Dynamic Entity Representations in Neural Language Models. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1830–1839, Copenhagen, Denmark. Association for Computational Linguistics.

Kingma, D. P. and Welling, M. (2014). Auto-Encoding Variational Bayes. In *ICLR*. arXiv: 1312.6114.

Li, J., Monroe, W., Shi, T., Jean, S., Ritter, A., and Jurafsky, D. (2017). Adversarial Learning for Neural Dialogue Generation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2157–2169, Copenhagen, Denmark. Association for Computational Linguistics.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). *Improving language understanding by generative pre-training*.

Rush, A. M., Chopra, S., and Weston, J. (2015). A Neural Attention Model for Abstractive Sentence Summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 379–389, Lisbon, Portugal. Association for Computational Linguistics.

See, A., Liu, P. J., and Manning, C. D. (2017). 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, Vancouver, Canada. Association for Computational Linguistics.

Sohn, K., Lee, H., and Yan, X. (2015). Learning Structured Output Representation using Deep Conditional Generative Models. In *Advances in neural information processing systems*, pages 3483–3491.

Subramanian, S., Mudumba, S. R., Sordoni, A., Trischler, A., Courville, A. C., and Pal, C. (2018). Towards Text Generation with Adversarially Learned Neural Outlines. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, pages 7551–7563. Curran Associates, Inc.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is All you Need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Wiseman, S., Shieber, S., and Rush, A. (2018). Learning Neural Templates for Text Generation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3174–3187, Brussels, Belgium. Association for Computational Linguistics.

Wu, Z., Galley, M., Brockett, C., Zhang, Y., Gao, X., Quirk, C., Koncel-Kedziorski, R., Gao, J., Hajishirzi, H., Ostendorf, M., and Dolan, B. (2020). A Controllable Model of Grounded Response Generation. *arXiv:2005.00613 [cs]*. arXiv: 2005.00613.