Natural Language Generation

Andrea Zugarini

SAILab

December 5th, 2019

Natural Language Generation

Natural Language Generation is the problem of generating text automatically.

Machine Translation, **Text Summarization** and **Paraphrasing** are all instances of NLG.

Language generation is a very challenging problem, that does not only require text **understanding**, but it also involves typical human skills, such as **creativity**.

Word representations and Recurrent Neural Networks (RNNs) are the basic tools for NLG models,usually called as **end-to-end** since they learn directly from data.

From Language Modelling to NLG

Recap: Given a sequence of words $y_1, \ldots, y_n$, a **language model** is characterized by a probability distribution:

$$P(y_1, \ldots, y_m) = P(y_m | y_1, \ldots, y_{m-1}) \ldots P(y_2 | y_1) P(y_1)$$

that can be equivalently expressed as:

$$P(y_1, \ldots, y_m) = \prod_{i=1}^{m} P(y_i | y_{<i})$$

Language Modelling is strictly related to NLG.

From Language Modelling to NLG

Many NLG problems are conditioned to some given **context**.

In **Machine Translation**, the generated text strictly depends on the input text to translate.

Hence, we can add to the equation another sequence x of size n to condition the probability distribution,

$$P(y_1, \ldots, y_m) = \prod_{i=1}^{m} P(y_i | y_{<i}, x_1, \ldots, x_m)$$

obtaining a general formulation for any **Language Generation** problem.

Natural Language Generation

A Machine-Learning model can then be used to learn $P(\cdot)$.

$$P(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}) = \prod_{i=1}^{m} P(y_i|y_{<i}, x_1, \ldots, x_n, \boldsymbol{\theta})$$

$$\max_{\boldsymbol{\theta}} P(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta})$$

where $P(\cdot)$ is the model parametrized by $\boldsymbol{\theta}$ that is trained to maximize the likelihood of $\boldsymbol{y}$ on a dataset of $(\boldsymbol{x}, \boldsymbol{y})$ sequence pairs.

Note: when $\boldsymbol{x} \in \emptyset$ we fall in **Language Modelling**.

Depending on how much x conditions P, we distinguish among two kinds of text generation:

Open-ended

- Story Generation
- Text Continuation
- **Poem Generation**
- Lyrics Generation

Non open-ended

- Machine Translation
- Text Summarization
- Text Paraphrasing
- Data-to-text generation

There is no neat separation between those kind of problems.

Once these models are trained, how do we exploit in inference to generate new tokens?

Straightforward approach: pick the sequence with maximum probability.

$$\boldsymbol{y} = \arg\max_{y_1,\ldots,y_n} \prod_{i=1}^{m} P(y_i|y_{<i}, x_1, \ldots, x_n, \boldsymbol{\theta})$$

Finding the optimal $\boldsymbol{y}$ is not tractable.

Two popular approximate methods are **greedy** and **beam** search, both successful in **non** open-ended domains.

Once these models are trained, how do we exploit in inference to generate new tokens?

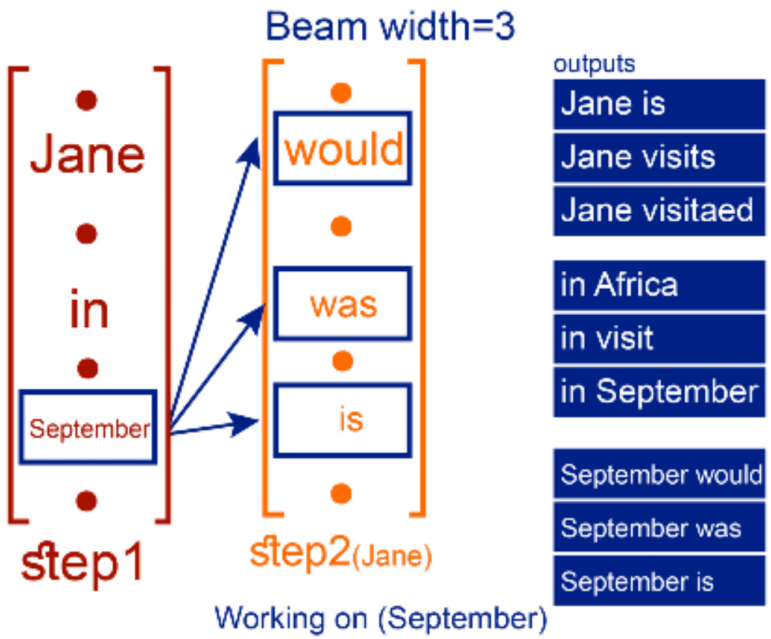
Straightforward approach: pick the sequence with maximum probability.

$$\boldsymbol{y} = \underset{y_1,\ldots,y_n}{\arg\max} \prod_{i=1}^{m} P(y_i | y_{<i}, x_1, \ldots, x_n, \boldsymbol{\theta})$$

Finding the optimal $\boldsymbol{y}$ is not tractable.

Two popular approximate methods are **greedy** and **beam** search, both successful in **non** open-ended domains.

Beam search is a search algorithm that explores k^2 nodes at each time step and keeps the best k paths.



Greedy search is a special case of beam search, where the beam width k is set to 1.

Unfortunately, likelihood maximization is only effective in **non** open-ended problems, where there is a strong correlation between input x and output y.

In open-ended domains, instead, it ends up in **repetitive**, meaningless generations.

To overcome such issue, **sampling** approaches better explore the entire learnt distribution P.

The most common sampling strategy is: **multinomial sampling**.

At each step i a token y_i is sampled from P.

$$y_i \sim P(y_i|y_{<i}, x_1, \ldots, x_n)$$

The higher is $P(y_i|y_{<i}, x_1, \ldots, x_n)$ the more y_i is likely to be sampled.

Poem Generation

Project reference

`sailab.diism.unisi.it/poem-gen/`

Poem Generation

Poem Generation is an instance of Natural Language Generation (NLG).

GOAL: Design an end-to-end **poet-based** poem generator.

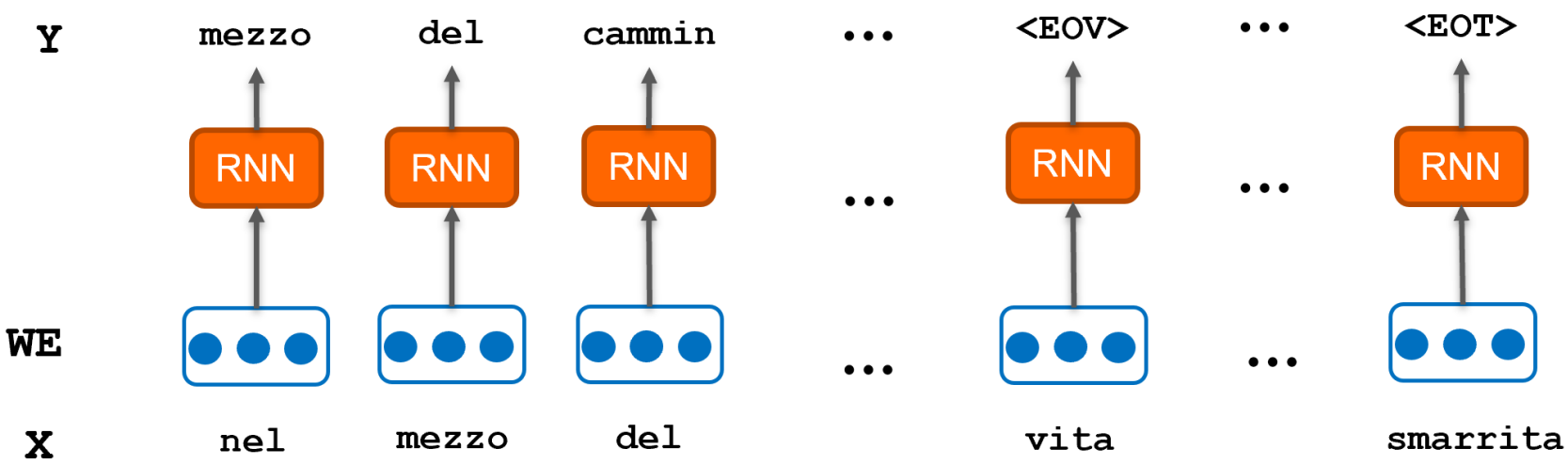
ISSUE: Poet's production is rarely enough to train a neural model.

We will describe a **general** model to learn poet-based poem generators.

We experimented it in the case of Italian poetry.

Poem Generation

The sequence of text is processed by a recurrent neural network (LSTM), that has to predict the next word at each time step.



Note: <EOV>, <EOT> are special tokens to indicate the end of a verse or a tercet.

Corpora

We considered poetries from Dante and Petrarca.

Divine Comedy

- 4811 tercets
- 108k words
- **ABA** rhyme scheme (enforced through rule-based post-processing)

Canzoniere

- 7780 tercets
- 63k words

Note: 100k words is 4 order of magnitude less data that traditional corpora!!!

Let's look at the Demo:

`www.dantepetrarca.it`