UE 803 - Data Science for NLP

Lecture 16: Generating Text with RNNs

Claire Gardent - CNRS / LORIA

Outline

- Language Modeling
    - What is it ?
    - What are LM useful for ?
    - Evaluating a LM
- Training LMs and generating with them
    - Pre-neural Language Models
    - RNN-based LMs
- Conditional Generation
    - the encoder-decoder framework
    - the attention mechanism

Language Modeling

Language Modeling

How probable is that text for a given language?

A language model assigns a probability to a text

$$P(W) = P(w_1, w_2, w_3, \ldots, w_n)$$

$$\Leftrightarrow$$

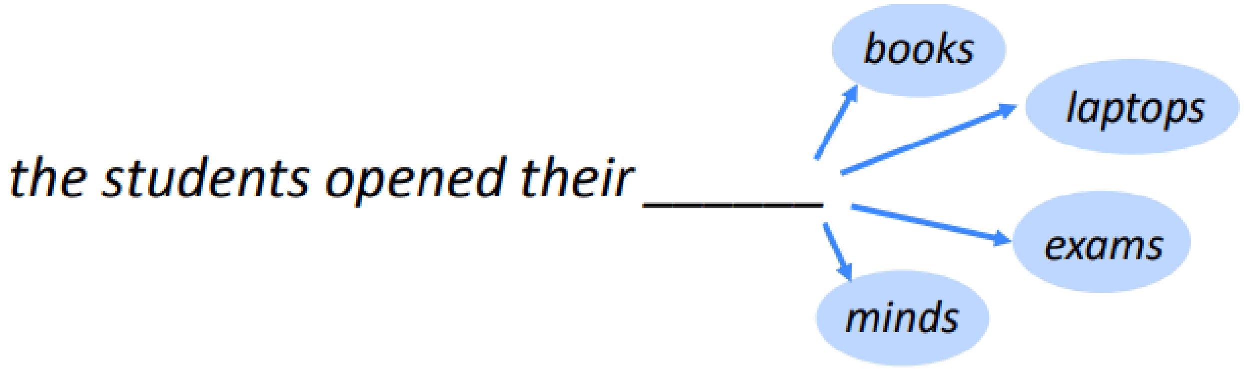
$$P(w_1, \ldots, w_n) = P(w_1) \times P(w_2 \mid w_1) \times ... \times P(w_n \mid w_1 ... w_{n-1})$$

$$\Leftrightarrow$$

$$P(w_1, \ldots, w_n) = \prod_i P(w_i | w_1, w_2, \ldots, i_{i-1})$$

Language Modeling

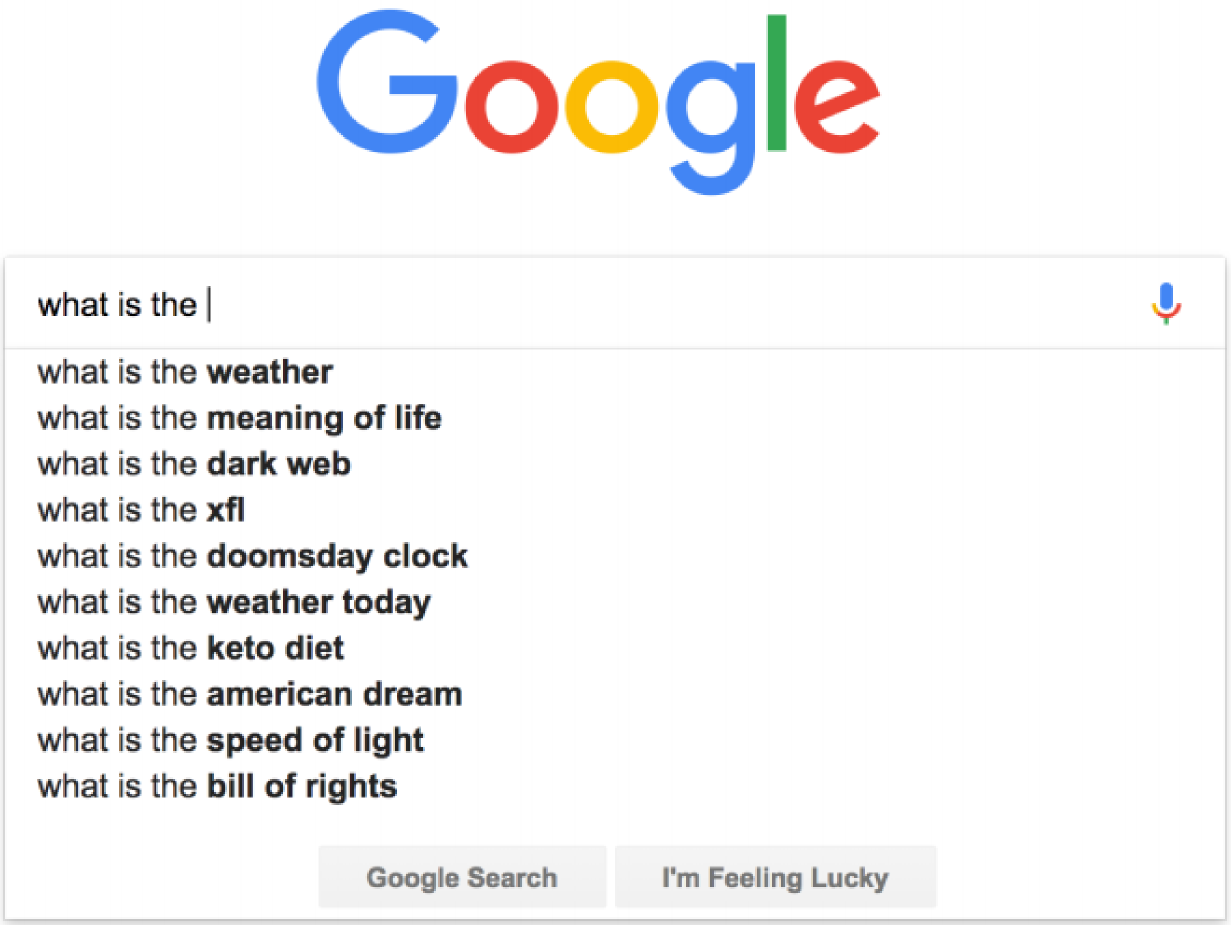
Language Modeling can predict what word comes next.



Given a sequence of words $x_1 \ldots x_n$, a Language Model (LM) can compute the probability distribution of the next word x_{n+1}

$$P(x_{n+1} \mid x_1 \ldots x_n)$$

Language Models are everywhere



What is LM useful for ?

Language Modeling is a subcomponent of many NLP tasks, especially those involving generating text or estimating the probability of text:

- Predictive typing
- Speech recognition
- Handwriting recognition
- Spelling/grammar correction
- Authorship identification
- Machine translation
- Summarization
- Dialogue
- etc.

Estimating word sequence probabilities

To create a language model we need to estimate the ***conditional probability*** of each word in all possible contexts

$$P(w_1, \ldots, w_n) = P(w_1) \times P(w_2 \mid w_1) \times ... \times P(w_n \mid w_1 ... w_{n-1})$$

The conditional probability of a word can be estimated on a large corpus as follows

$$P(w_n \mid w_1, \ldots, w_{n-1}) = \frac{count(w_1, w_2, w_3 ... w_n)}{count(w_1, w_2, w_3 ... w_{n-1})}$$

But this is not doable because there are too many possible sequences in natural language.
So we simplify and ***approximate conditional probabilities*** by reducing the context (***Markow Assumption***)

Approximating the joint probability of a sequence

$$P(w1, \ldots, w_n) = \prod_i P(w_i | w1, \ldots, i_{i-1})$$

is approximated to

$$P(w1, \ldots, w_n) = \prod_i P(w_i | w_{i-k}, \ldots, i_{i-1})$$

That is, we approximate each factor as

$$P(w_n \mid w1, \ldots, w_{n-1}) \approx P(w_n \mid w_{i-k}, \ldots, w_{n-1})$$

Bigram Language Model

A Bigram Language Model computes conditional probabilities of sequences of two words.

Example

$$P(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

$P(\text{I} \mid \text{<s>}) = \frac{2}{3} = .67$ $P(\text{Sam} \mid \text{</s>}) = \frac{1}{3} = .33$ $P(\text{am} \mid \text{I}) = \frac{2}{3} = .67$

$P(\text{<s>} \mid \text{Sam}) = \frac{1}{2} = .5$ $P(\text{Sam} \mid \text{am}) = \frac{1}{2} = .5$ $P(\text{do} \mid \text{I}) = \frac{1}{3} = .33$

Evaluating a LM, the Shanon game:

How well can we predict the next word?

When I eat pizza, I wipe off the ...

Many children are allergic to ...

I saw a ...

$$\left\{ \begin{array}{ll} \textit{mushrooms} & 0.1 \\ \textit{pepperoni} & 0.1 \\ \textit{anchovies} & 0.01 \\ \ldots & \\ \textit{and} & 1e-100 \end{array} \right.$$

A better language model is one which assigns a higher probability to the word that actually occurs

Evaluating a LM, Perplexity

- We train (estime word sequence probabilities) the LM on a large corpus
- We test it on an unseen test set
- The best language model is one that best predicts an unseen test set, that assigns a high probability to the sentences in the test corpus.

Perplexity is the inverse probability of the test set, normalized by the number of words in the test set

$$PP(W) = \sqrt[N]{\frac{1}{P(w1, w2, \ldots, wN)}}$$

Minimizing perplexity is the same as maximizing the probability over the test set

Perplexity and Cross-Entropy Loss

The standard evaluation metric for Language Models is perplexity.

$$\text{perplexity} = \underbrace{\prod_{t=1}^{T} \left(\frac{1}{P_{\text{LM}}(\boldsymbol{x}^{(t+1)} | \, \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})} \right)}_{\text{Inverse probability of corpus, according to Language Model}}^{1/T}$$

Normalized by number of words

This is equal to the exponential of the cross-entropy loss

$$= \prod_{t=1}^{T} \left(\frac{1}{\hat{\boldsymbol{y}}^{(t)}_{\boldsymbol{x}_{t+1}}} \right)^{1/T} = \exp\left(\frac{1}{T} \sum_{t=1}^{T} -\log \hat{\boldsymbol{y}}^{(t)}_{\boldsymbol{x}_{t+1}} \right) = \exp(J(\theta))$$

Pre-Neural Language Models

Pre-Neural LM Learning

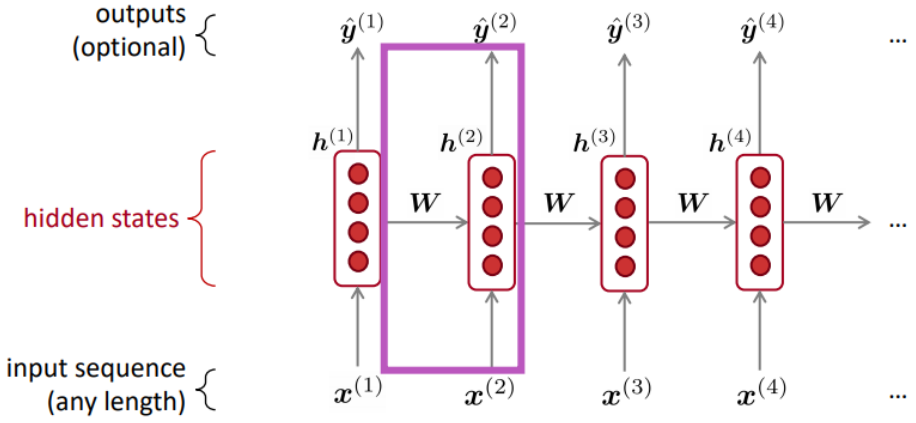
- An n-gram is a sequence of n consecutive words.
    - unigrams: “the”, “students”, “opened”, ”their”
    - bigrams: “the students”, “students opened”, “opened their”
    - trigrams: “the students opened”, “students opened their”
    - 4-grams: “the students opened their”
- Collect statistics about the frequency of different n-grams
- Use these to compute conditional probabilities

Neural Language Models

Recurrent Network for Language Modeling

- $\hat{y_t}$: distribution over vocabulary, specify a probability for each word in the vocabulary

- can be used to learn the conditional probability of words



Output distribution

$$\hat{y_t} = softmax(W_{hy}^{\top} h_t)$$

A Simple RNN Language Model

output distribution

$$\hat{\boldsymbol{y}}^{(t)} = \mathrm{softmax}\left(\boldsymbol{U}\boldsymbol{h}^{(t)} + \boldsymbol{b}_2\right) \in \mathbb{R}^{|V|}$$

hidden states

$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h\boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e\boldsymbol{e}^{(t)} + \boldsymbol{b}_1\right)$$

$\boldsymbol{h}^{(0)}$ is the initial hidden state

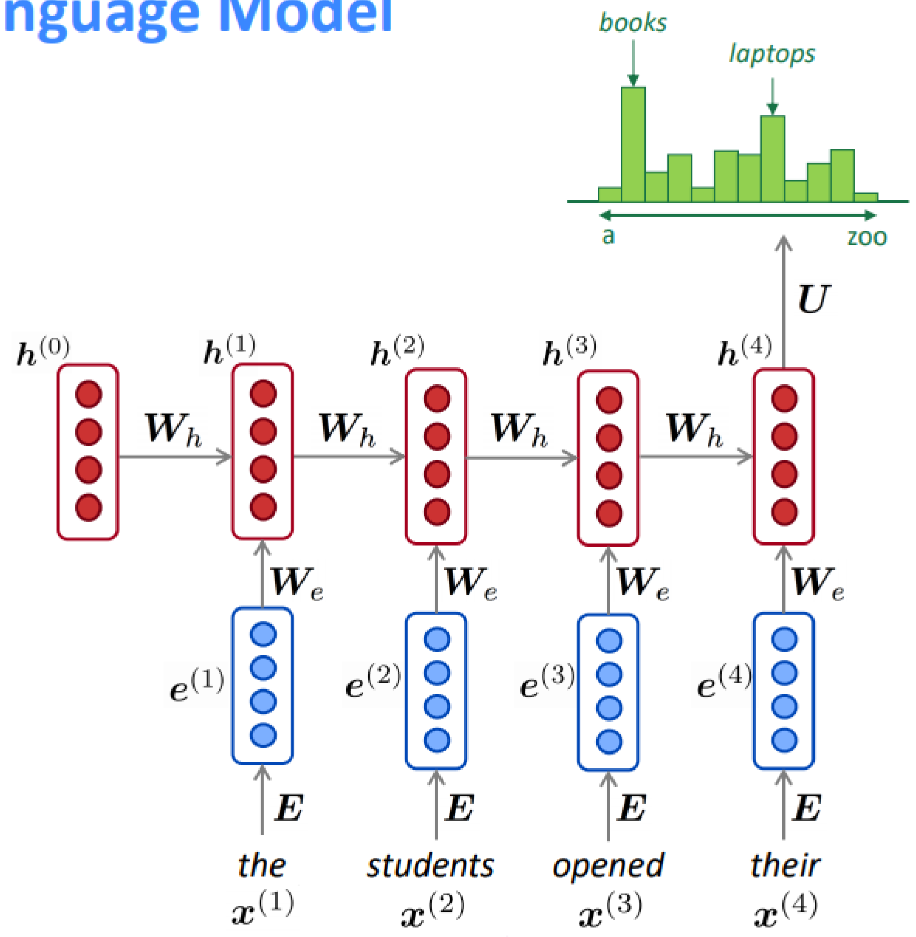
word embeddings

$$\boldsymbol{e}^{(t)} = \boldsymbol{E}\boldsymbol{x}^{(t)}$$

words / one-hot vectors

$$\boldsymbol{x}^{(t)} \in \mathbb{R}^{|V|}$$

$$\hat{\boldsymbol{y}}^{(4)} = P(\boldsymbol{x}^{(5)}|\text{the students opened their})$$



Training

A neural LM is learned by running an RNN over large quantities of text

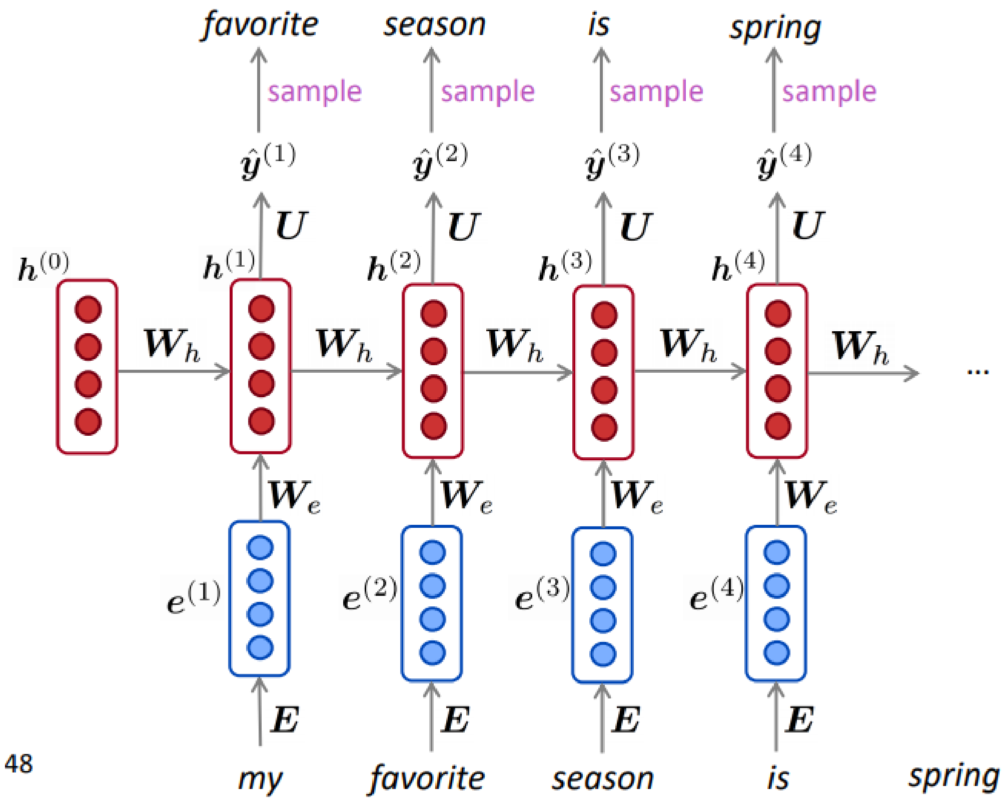
At each time step

- the RNN outputs a ***probability distribution over the corpus vocabulary***
  i.e., it tells us which words are most likely given the preceding context
- The prediction is compared over the expected word (the word that actually occurs at that time step in the text)
- The difference between expectation and prediction is computed using Cross Entropy loss (difference between two distributions)
- The RNN weights are adjusted accordingly (using Stochastic Gradient Descent)

Generating with RNN

Generating text with a RNN Language Model

- An RNN Language Model can be used to generate text by repeated sampling.
- The sampled output is next step's input.



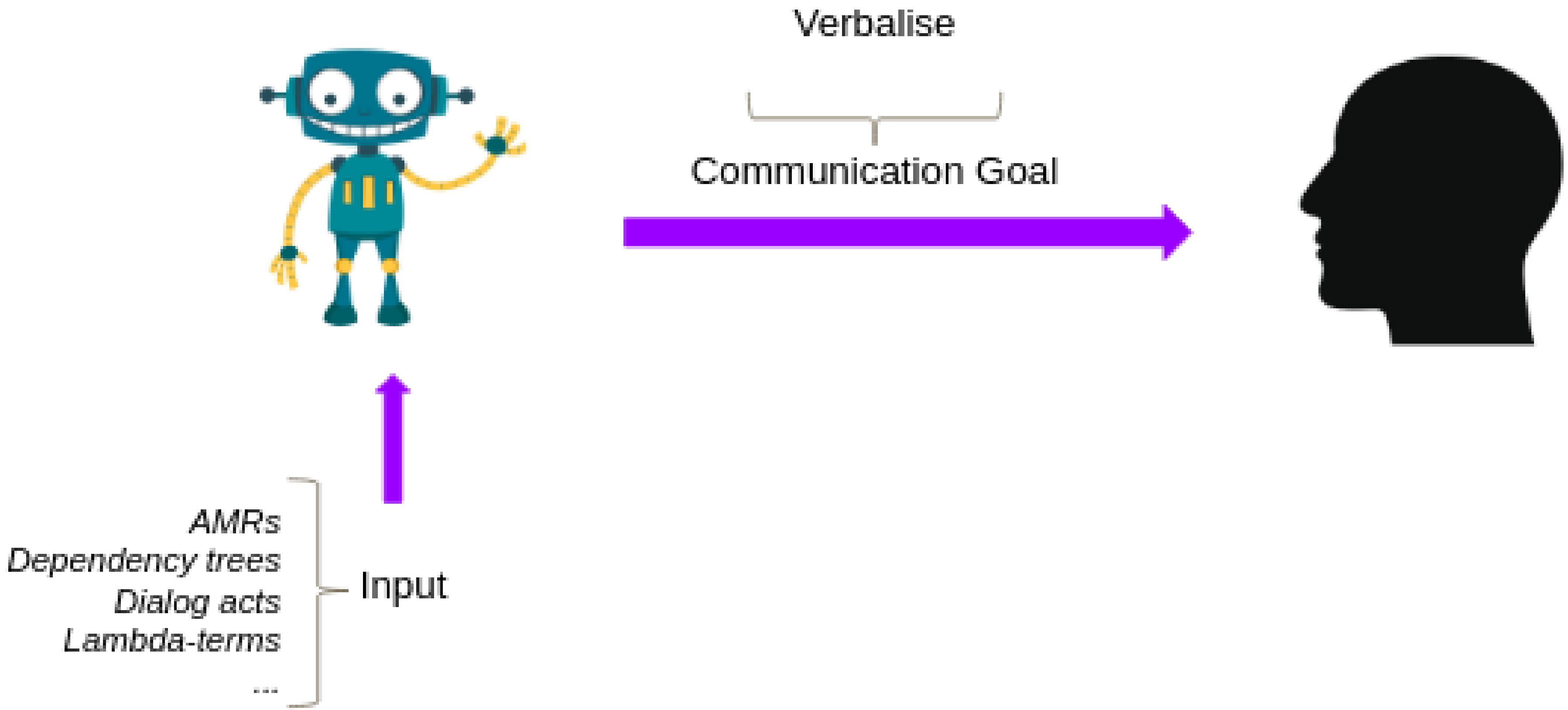
Input-Constrained Text Generation

The Encoder-Decoder Framework

Generating text from an input

- Language Models generate text independently of any input
- Text can also be generated from some input
- This is the research domain of ***Natural Language Generation*** (NLG)
  - ***Data-to-Text NLG***: Generating text from KB, DB, numerical data etc
  - ***MR-to-Text***: Generating text from Meaning Representations
  - ***Text-to-Text***: Generating text from Text (summarisation, simplification, paraphrasing)

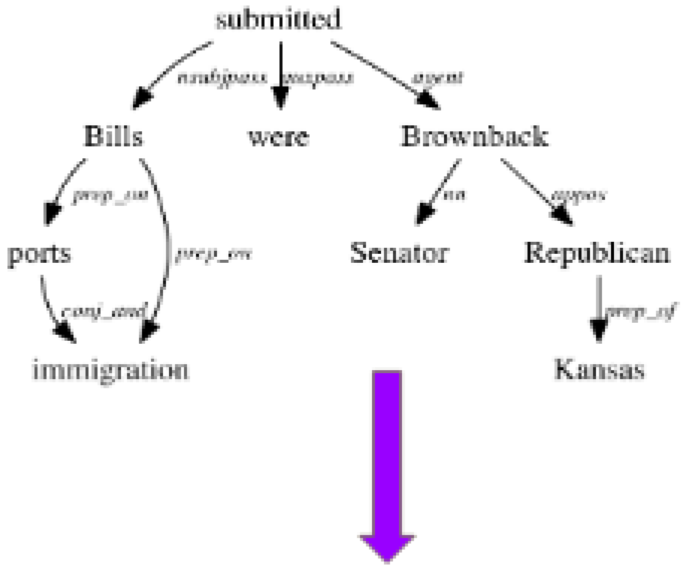
Generating text from Meaning Representations



Generating text from Dependency Trees

Surface Realization Challenge 2011 and 2018

- Shallow and deep approaches
- Universal dependency trees

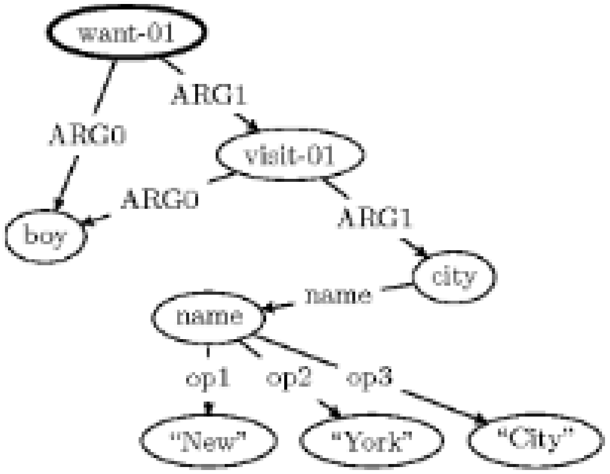


Bills on immigration were submitted by Senator Brownback, a Republican of Kansas.

Generating text from Abstract Meaning Representations

SemEval Shared Task 2017: AMR Generation and Parsing

A boy wants to visit New York City.
A boy wanted to visit New York City.

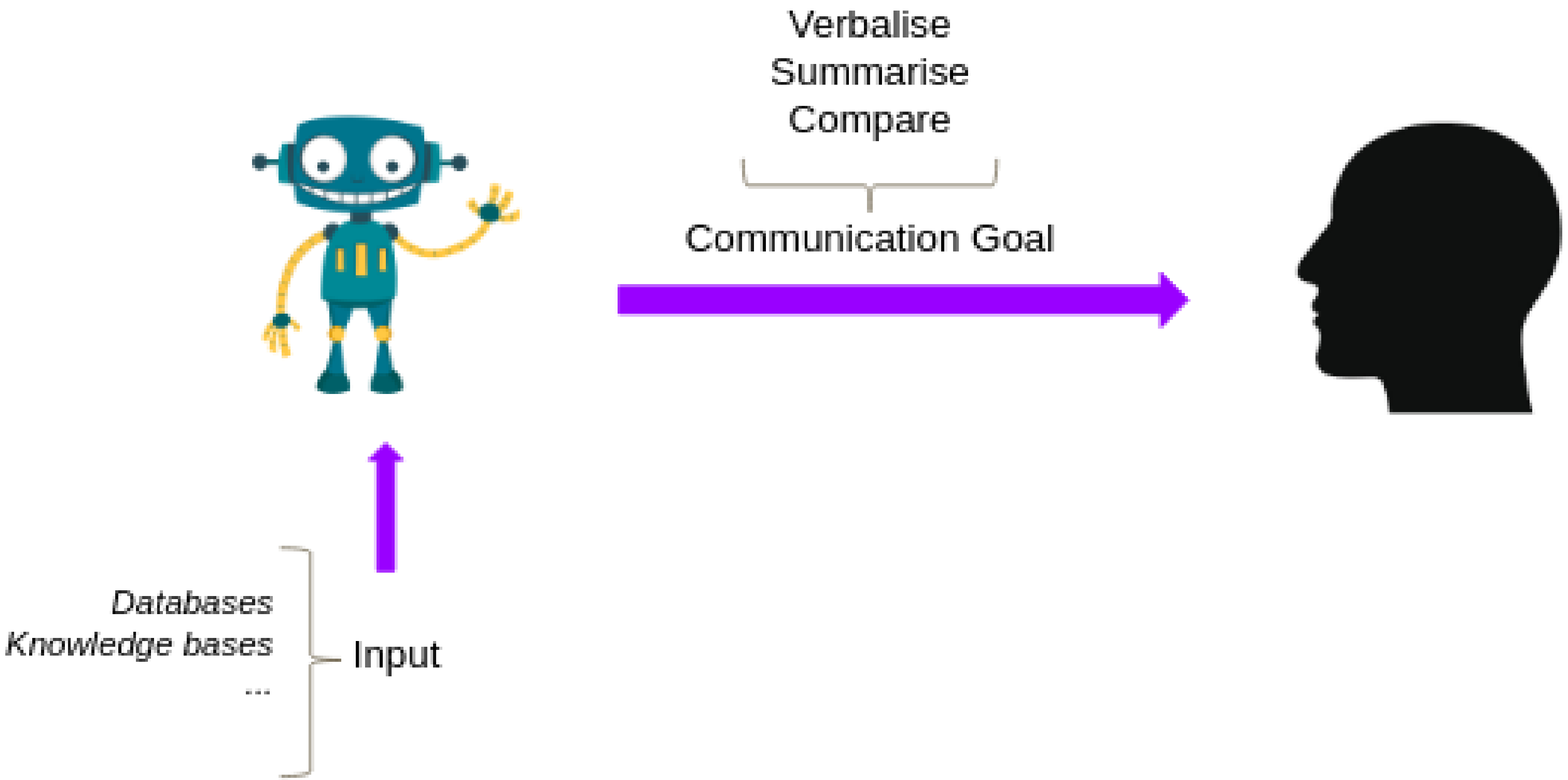


(a) Graph.

```
(w / want-01
  :ARG0 (b / boy)
  :ARG1 (g / visit-01
          :ARG0 b
          :ARG1 (c / city
                  :name (n / name
                          :op1 "New"
                          :op2 "York"
                          :op3 "City"))))
```

(b) AMR annotation.

Generating text from Data



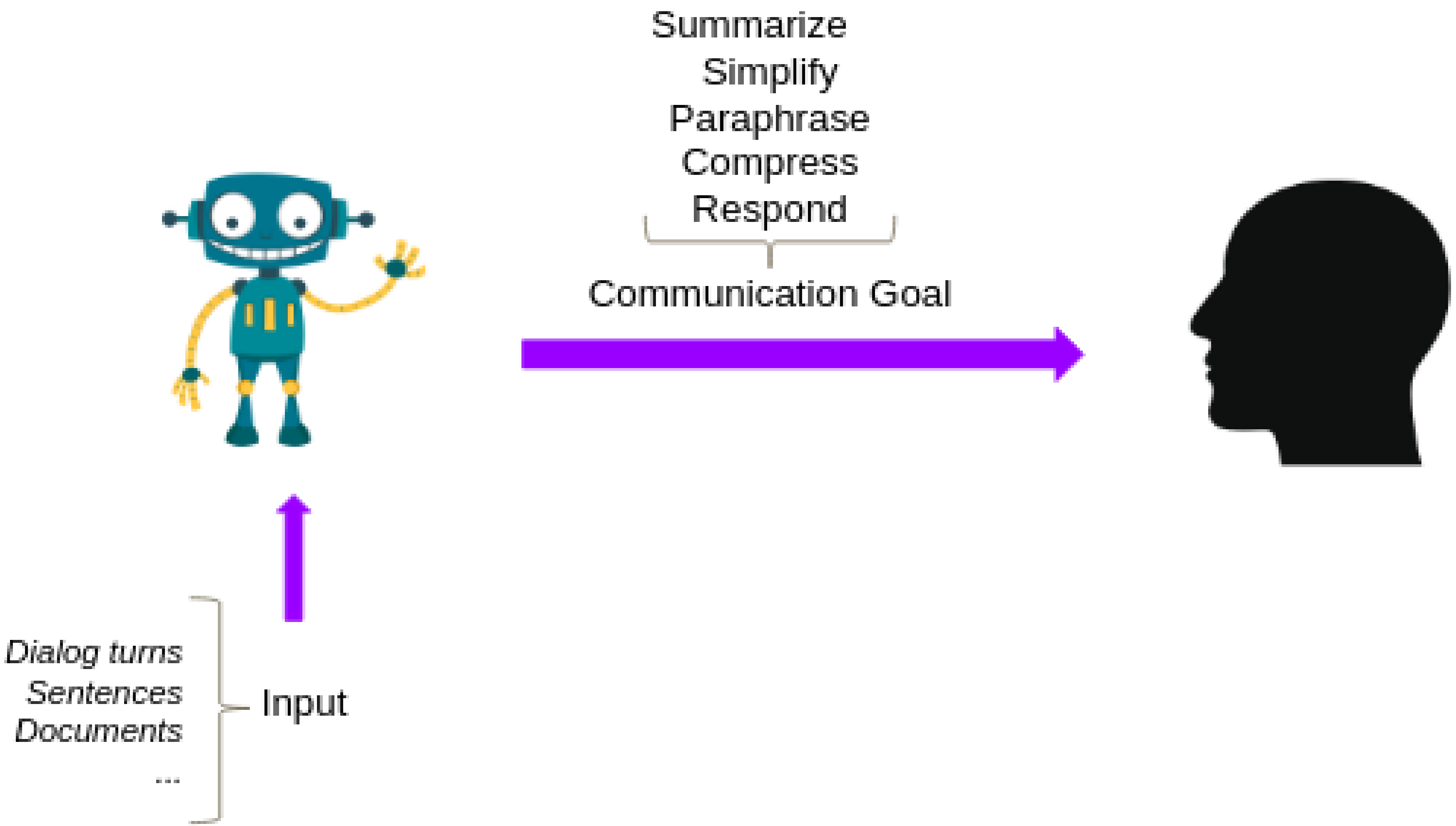
Generating text from Knowledge Bases

The WebNLG Challenge 2017

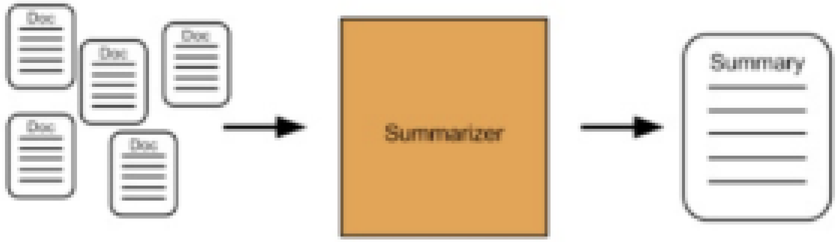
(John_E_Blaha *birthDate* 1942_08_26)
(John_E_Blaha *birthPlace* San_Antonio)
(John_E_Blaha *occupation* Fighter_pilot)

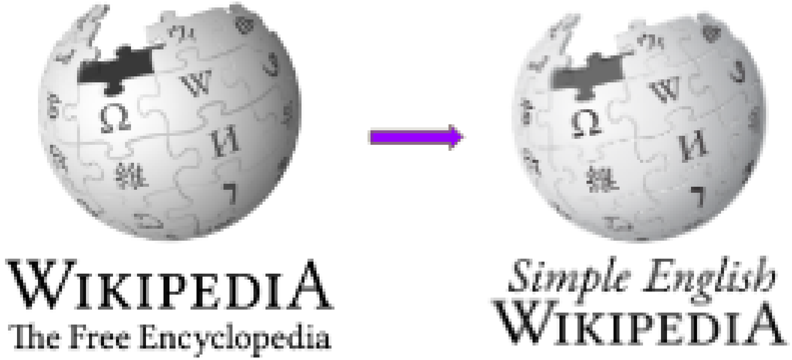
"John E Blaha, born in San Antonio on 1942-08-26, worked as a fighter pilot."

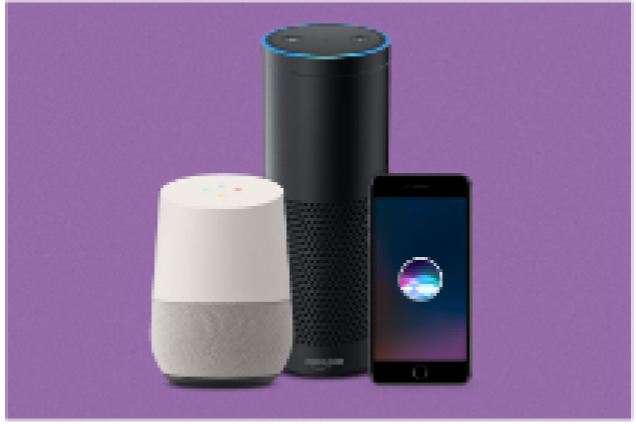
Generating text from text



Generating text from Text

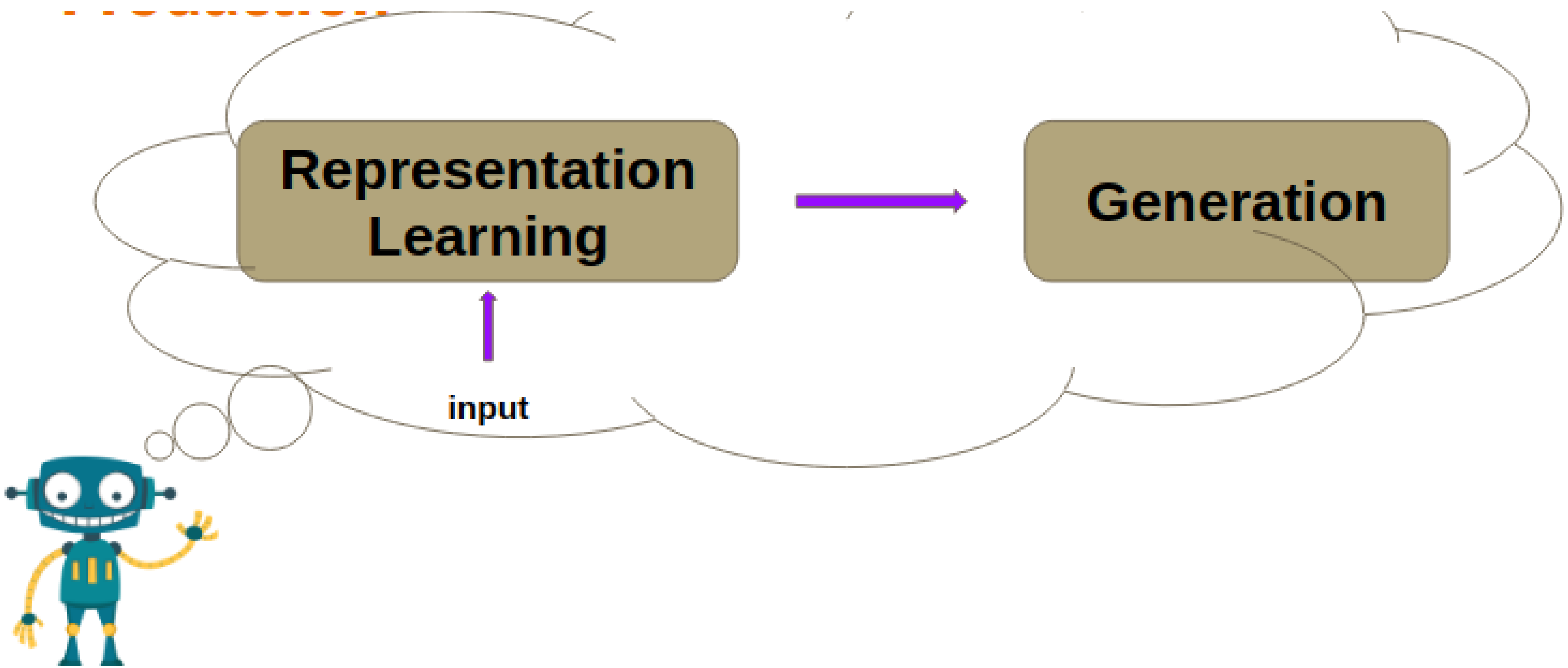






Neural Natural Language Generation

Neural approaches to NLG use the so-called ***encoder-decoder*** framework



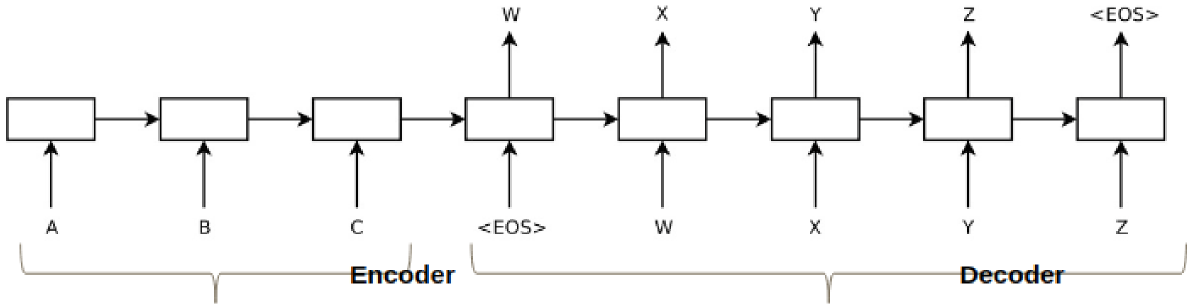
Encoder

- Builds a ***representation*** for the input
- Converts the input to a real valued vector
- Commonly used encoders:
    - Recurrent: RNN, LSTM, GRU
    - Convolutional
    - Graph
    - Tranformer

Decoder

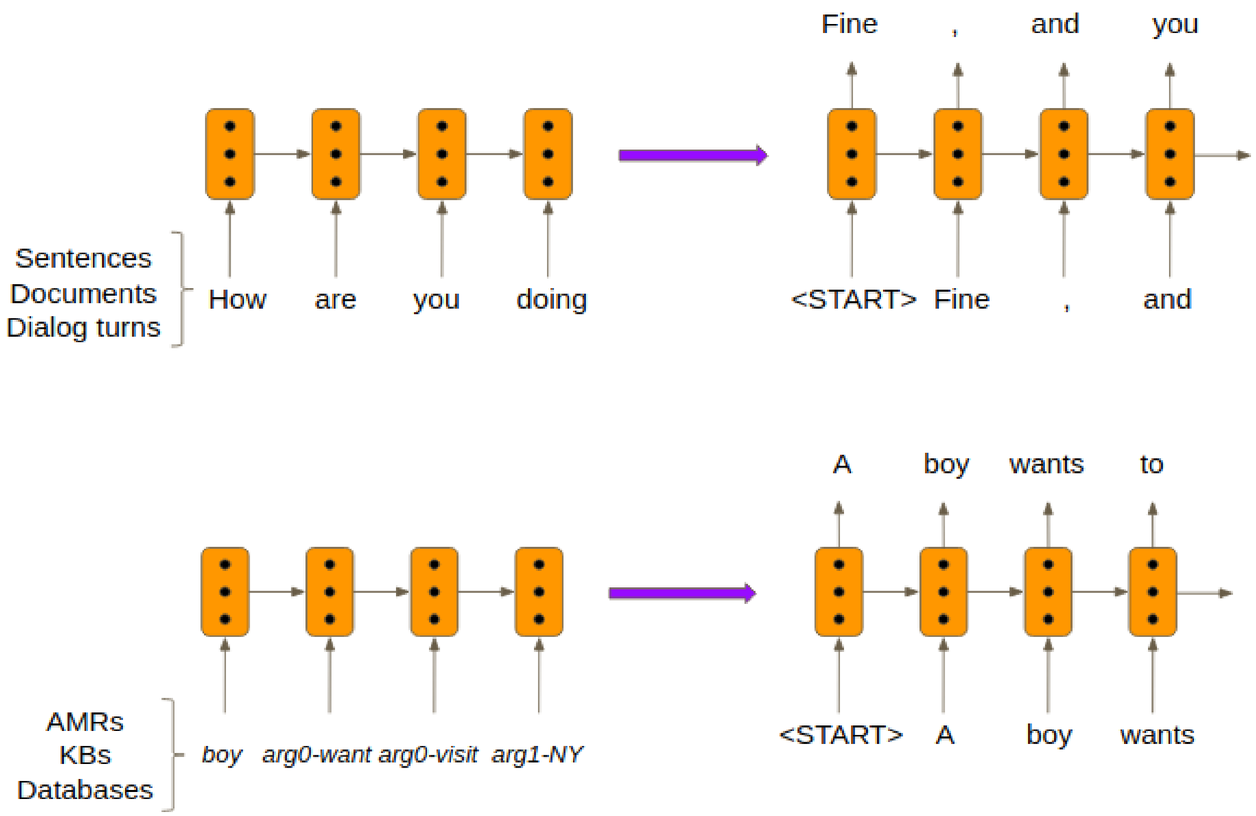
- A Recurrent network
- ***Generates text*** one word at a time
- Conditioned on input

Encoder-Decoder Model using a Recurrent Encoder



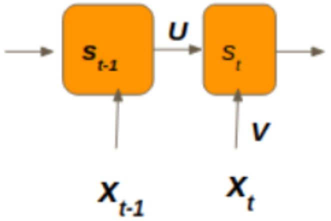
- The encoder processes each input token ***sequentially*** (one after the other)
- The input representation is generally taken to be the ***vector resulting from processing the last token in the input***
- This input representation is a ***real-valued vector*** "representing" the whole input

The different types of input to NLG (text, data, MRs) can be encoded using a recurrent network



Data or meaning representations need to be linearised first

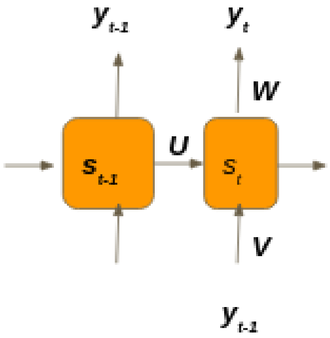
Encoding the Input using an RNN



$$s_t = tanh(U * s_{t-1} + V * x_t)$$

- x_i are vectors representing the input tokens (words, data or MR tokens)
- At each step, the encoder produces a new vector s_t (***state***) which represents the content of the preceding string of tokens
- The last state represents the meaning of the whole input
- U and V are the ***parameters*** learned during training
- tanh is a non linear function

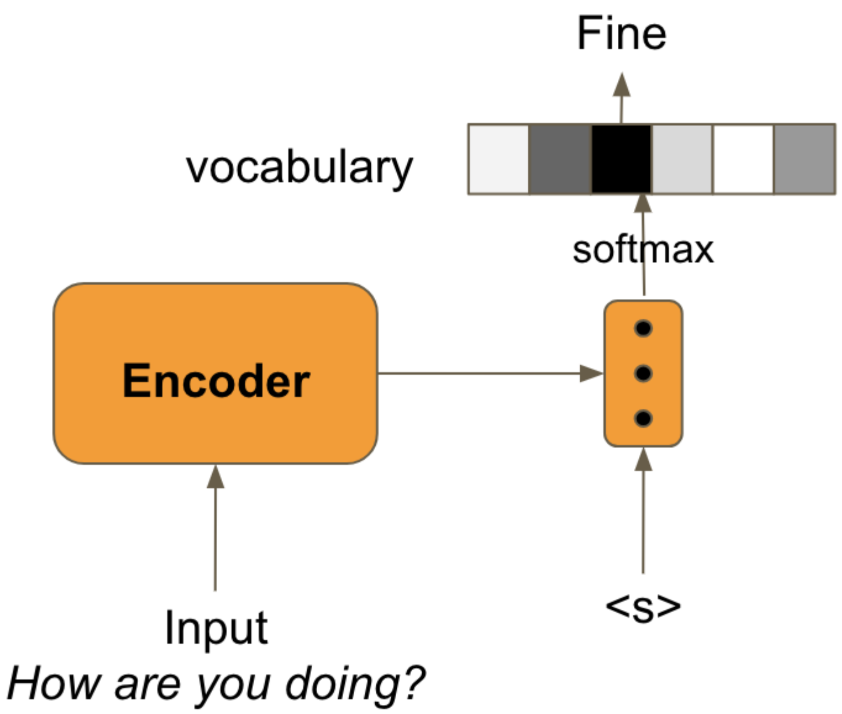
Decoding Words using an RNN



$$
\begin{aligned}
s_t &= tanh(U * s_{t-1} + V * y_{t-1}) \\
y_t &= softmax(W * s_t)
\end{aligned}
$$

- y_t is the word predicted at time t
- s_t is the network state at time t
- Each new state is computed taking into account the previous state s_{t-1} and the last predicted word y_{t-1}.
- The softmax function turns a vectors of scores into a probability distribution
- At each time step t, the output/predicted token y_t is sampled from this probability distribution

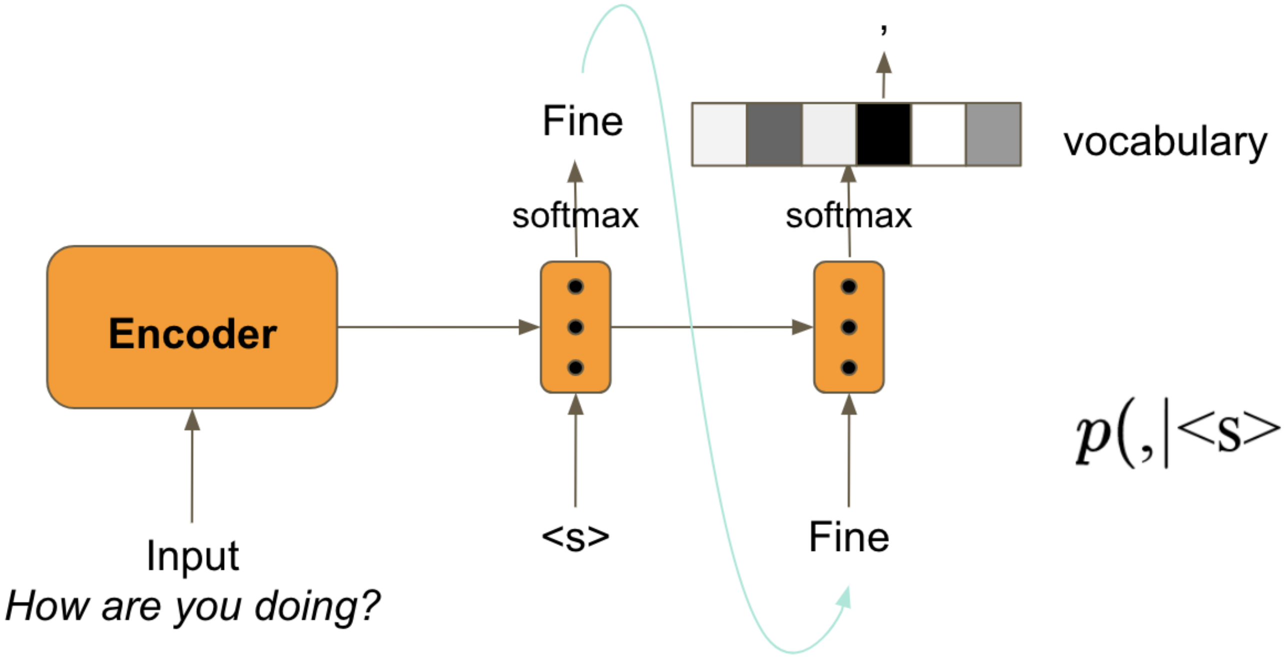
Generating Text using an RNN



$$p(\text{Fine}|\text{<s>}, \text{How are you doing?})$$

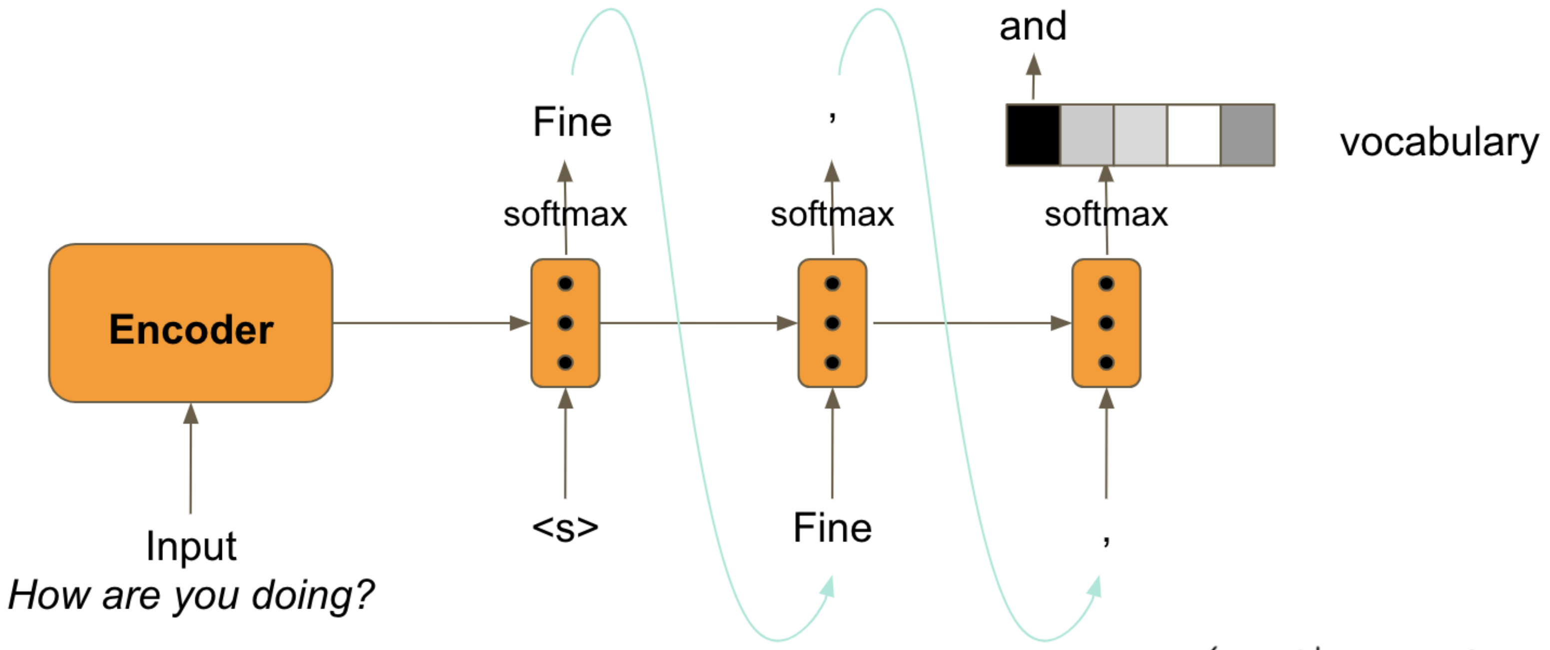
Conditional Generation

Generating Text using an RNN



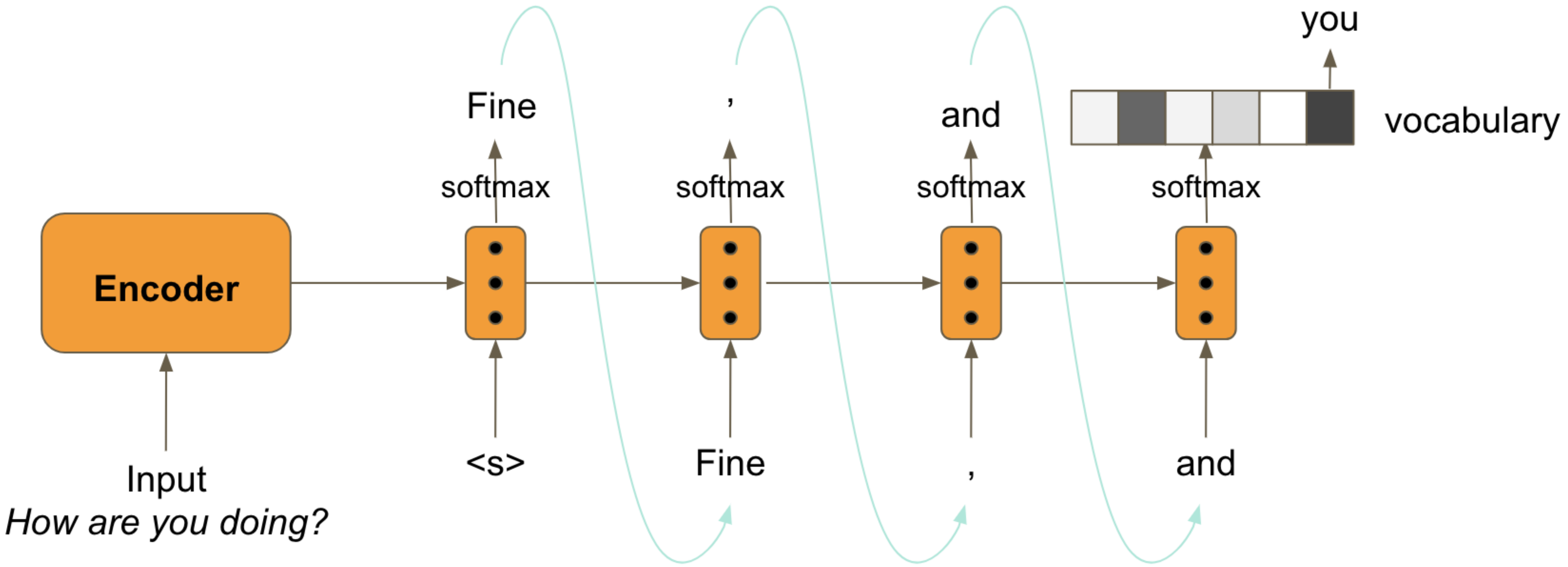
$$p(,|\langle s\rangle\ \text{Fine, How are you doing?})$$

Generating Text using an RNN

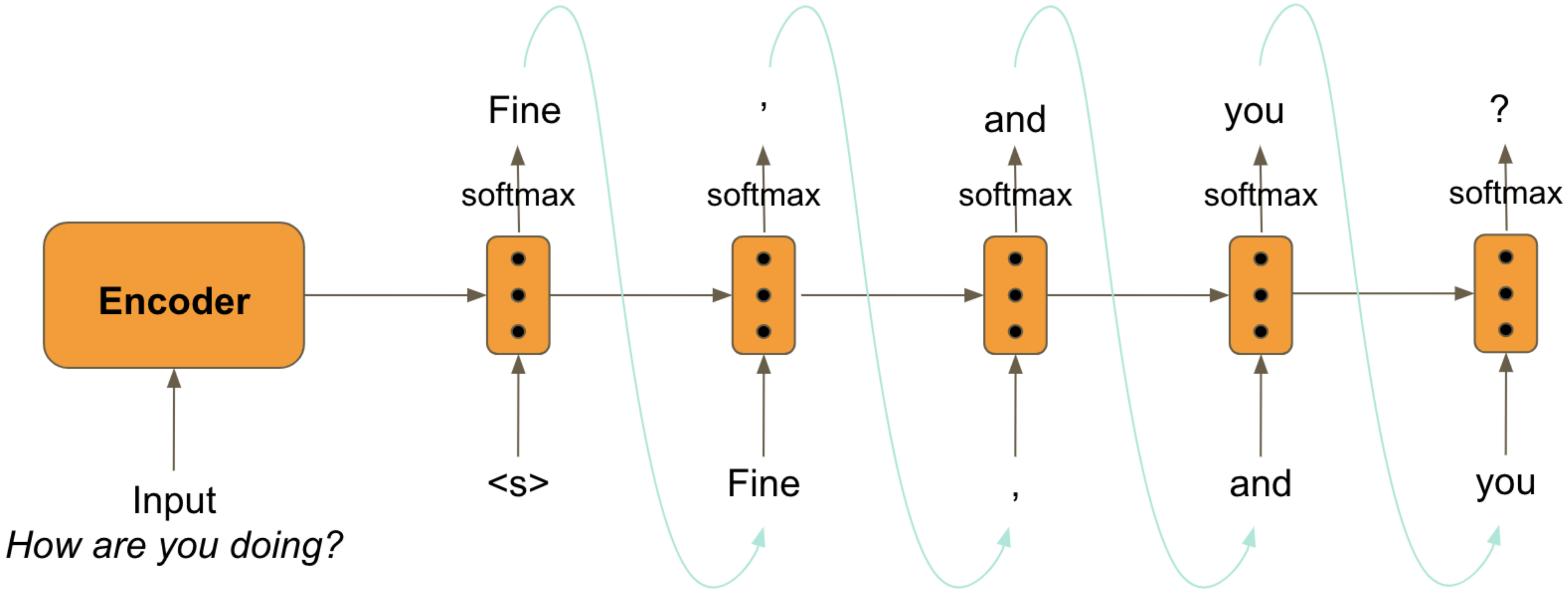


$$p(\text{and}|\text{<s> Fine,; How are you doing?})$$

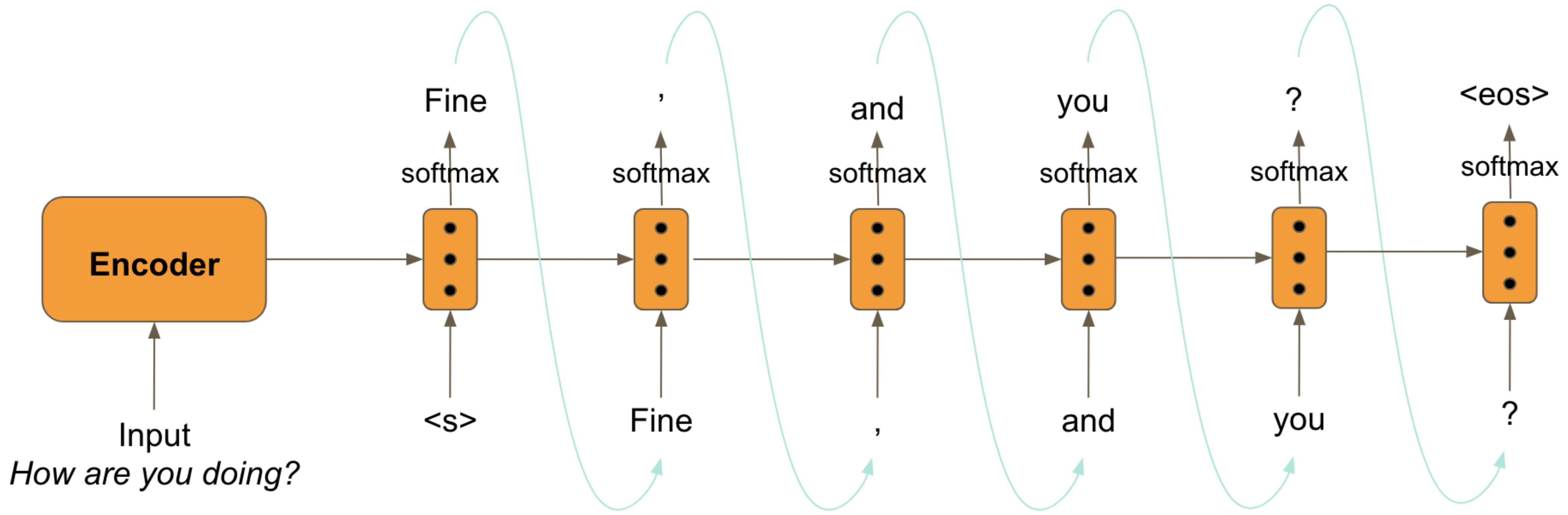
Generating Text using an RNN



Generating Text using an RNN

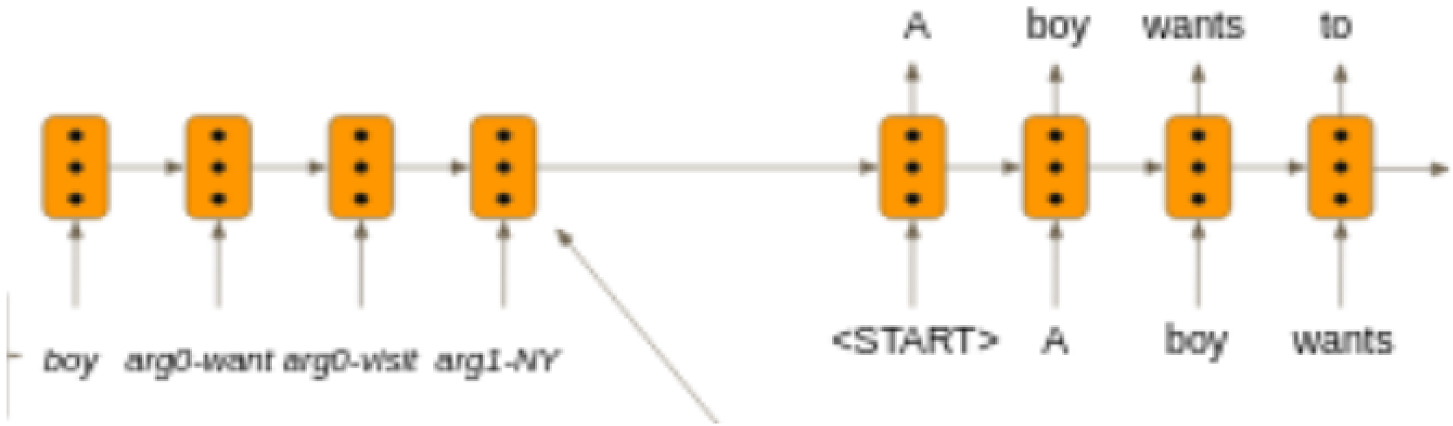


Generating Text using an RNN



Attention

Standard RNN Decoding



- The input is compressed into a ***fixed-length vector***
- Performance decreases with the length of the input [Sutskever et al. 2014]]

Decoding with Attention

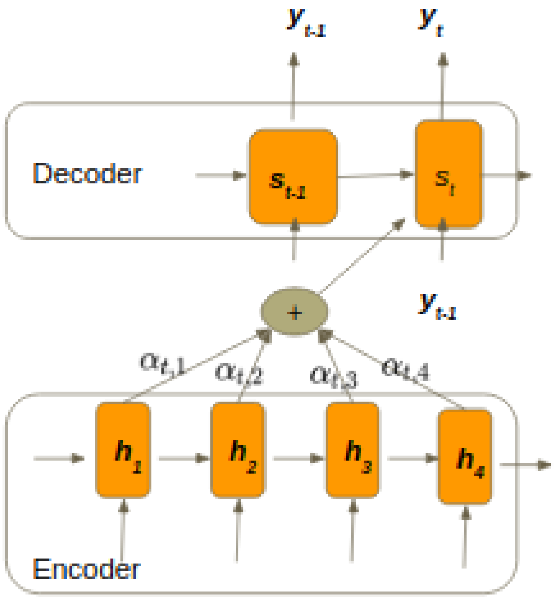
Input

- the previous state s_{t-1}
- the previously generated token y_{t-1} and
- a context vector c_t

Context vector

- depends on the previous state and therefore ***changes at each step***
- indicates ***which part of the input is most relevant*** to the decoding step

RNN with Attention



α can be viewed as a probability distribution over the source words

The next predicted token is sampled from the new ***target vocabulary distribution*** $softmax(Ws_t)$

A ***score*** is computed between each encoder hidden state and the current decoder state

$$\alpha_{t,j} = v^\top tanh(W_h h_j + W_s s_t + b)$$

Context Vector, the weighted sum of the encoder states

$$c_t = \sum \alpha_{t,j} . h_j$$

The ***new state*** is computed taking into account this context vector.

$$s_t = f(s_{t-1}, y_{t-1}, c_t)$$

Lab Session

Generate a film title

- Modify the sequence tagging RNN from last session so that it can be used to generate character sequences
- Write a function which given the `<start>` symbol and some initial hidden state, generates a film title one character at a time

Useful Links

- Lecture Slides from Stanford NLP Course: Language Modeling. Source for many of the slides in this lecture (for the language modeling part)
    - The video of the class
- A practical guide to Neural NLG. Up to date notes on neural NLG.
- Lecture Slides from Stanford NLP Course: Natural Language Generation