Encoder Decoder

a.k.a. where copying isn’t cheating (even on the test set)

Luigi Procopio (procopio@di.uniroma1.it)

SapienzaNLP Group

Sapienza University of Rome

Why

Why

- Used in several generative tasks
  - Abstractive Summarization, AMR parsing, …
  - Correction tasks such as code repair, grammar correction, …
- Elegant and neat solution to **OOV** and **task bias**
- Well, a useful tool that I can present in <45 min :)

Outline

- Quick Recap of Seq2Seq
- The OOV/UNK problem
- Token Copying
  - CopyNet
  - Pointer-Generator
- Span Copying
- Conclusions

Quick Recap on Seq2Seq

Outline

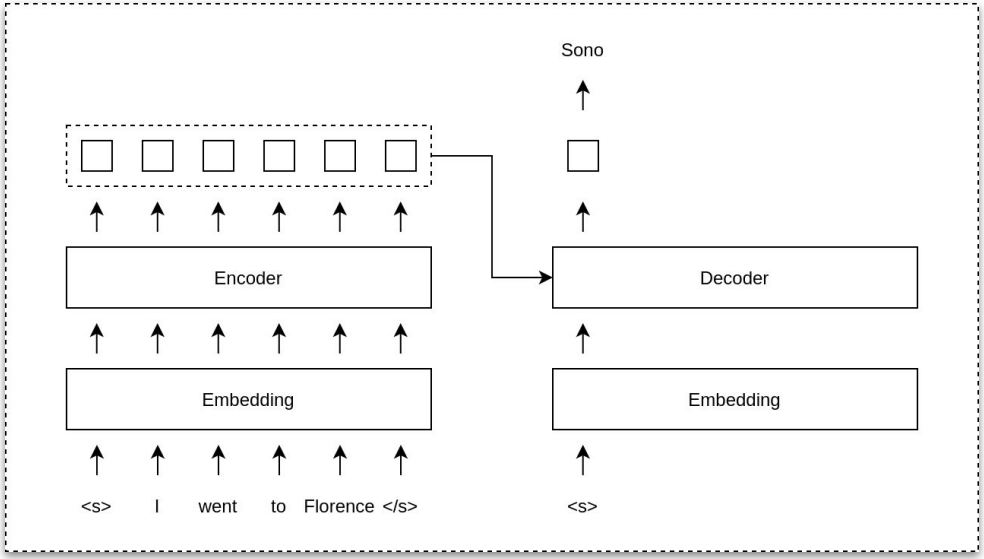
- Quick Recap of Seq2Seq
- The OOV/UNK problem
- Token Copying
  - CopyNet
  - Pointer-Generator
- Span Copying
- Conclusions

Architecture

I went to Florence ↦ Sono andato a Firenze

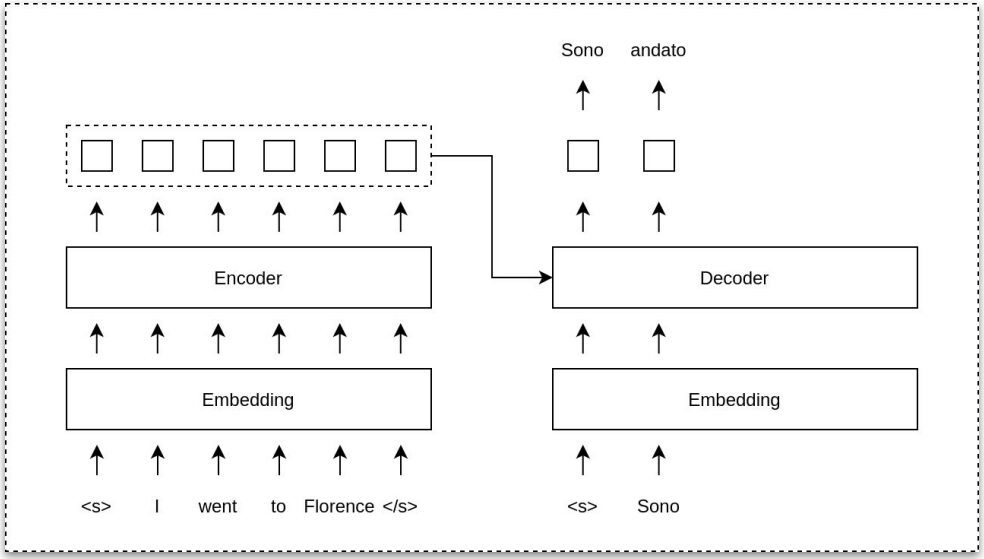
Architecture

$$p(\mathbf{y}|\,\mathbf{x}) = \prod_{j=0}^{n} p(y_j|\,\mathbf{x}, y_{[:j]})$$



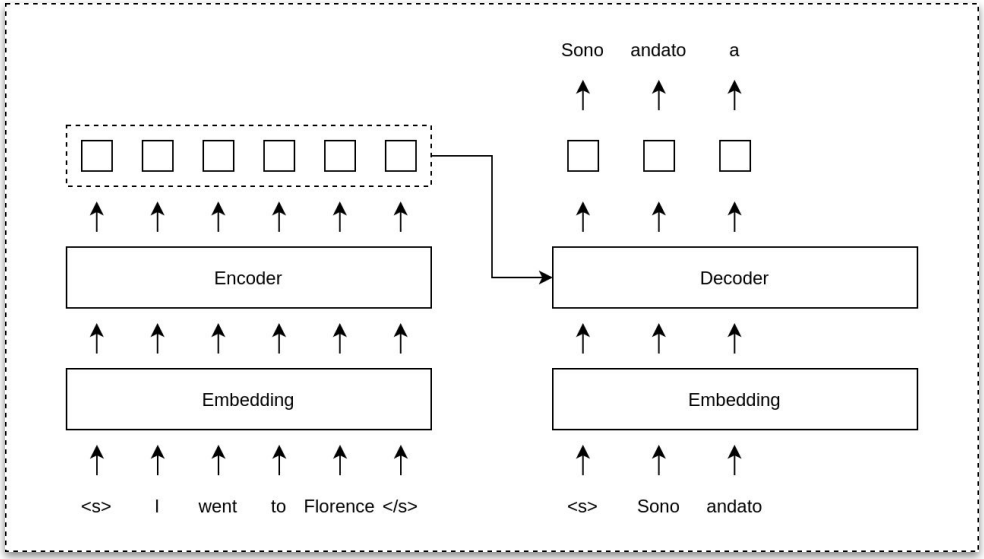
Architecture

$$p(\mathbf{y}|\,\mathbf{x}) = \prod_{j=0}^{n} p(y_j|\,\mathbf{x}, y_{[:j]})$$



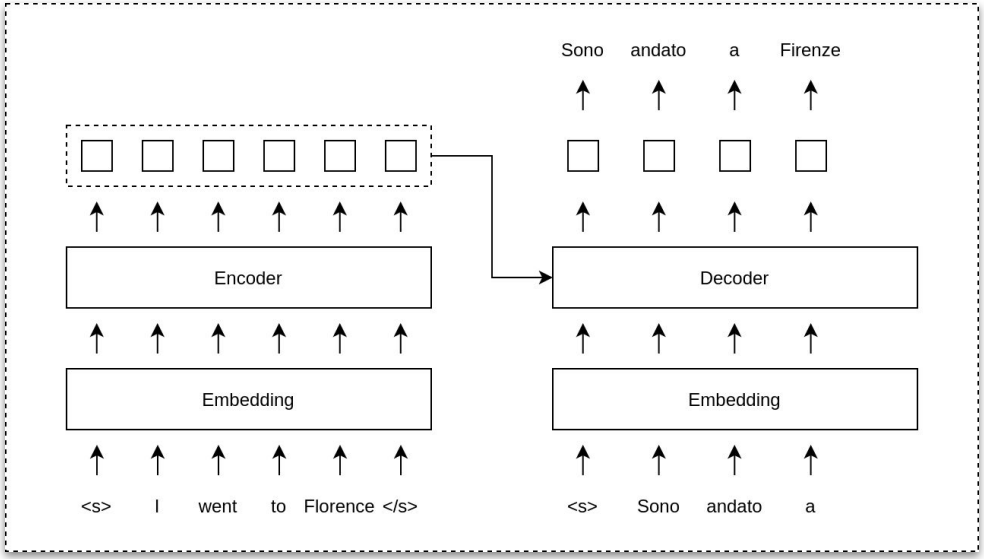
Architecture

$$p(\mathbf{y}|\,\mathbf{x}) = \prod_{j=0}^{n} p(y_j|\,\mathbf{x}, y_{[:j]})$$



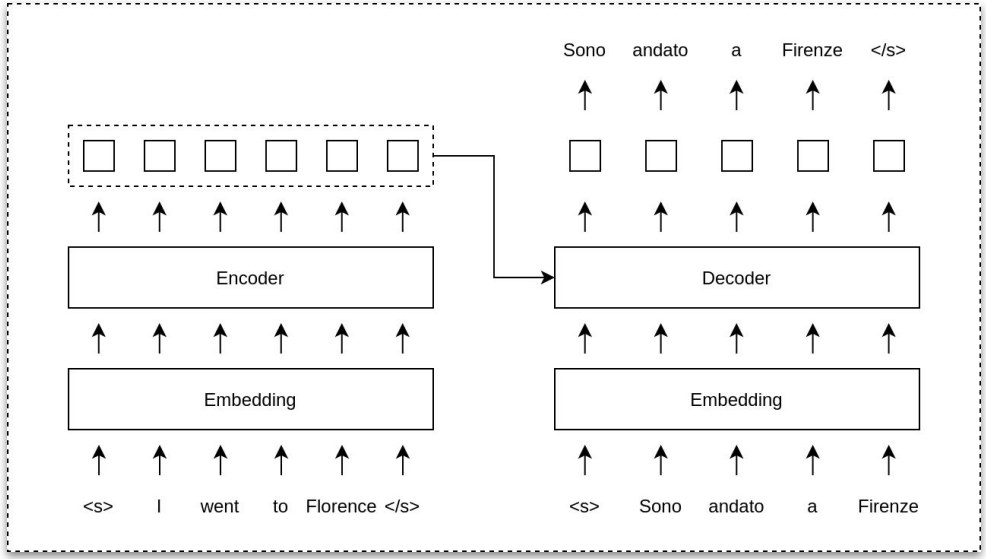
Architecture

$$p(\mathbf{y}|\,\mathbf{x}) = \prod_{j=0}^{n} p(y_j|\,\mathbf{x}, y_{[:j]})$$

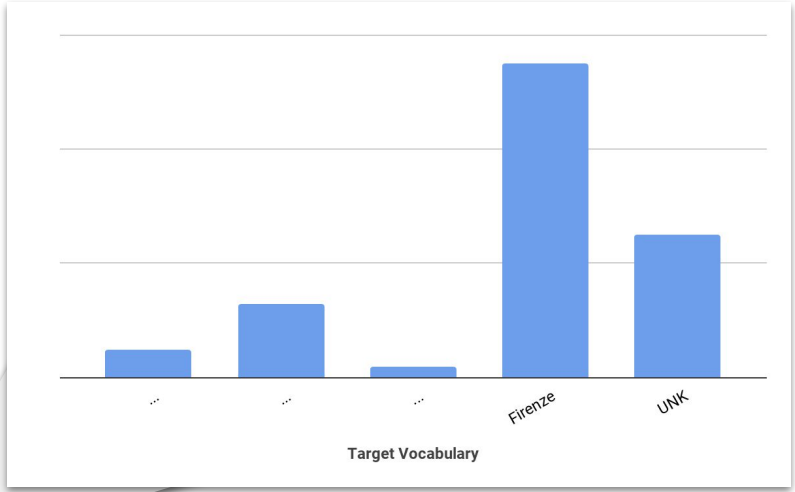


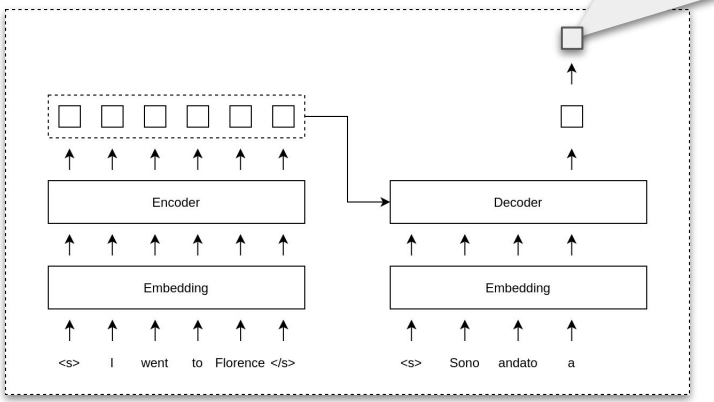
Architecture

$$p(\mathbf{y}|\,\mathbf{x}) = \prod_{j=0}^{n} p(y_j|\,\mathbf{x}, y_{[:j]})$$



Architecture





The OOV/UNK Problem

Outline

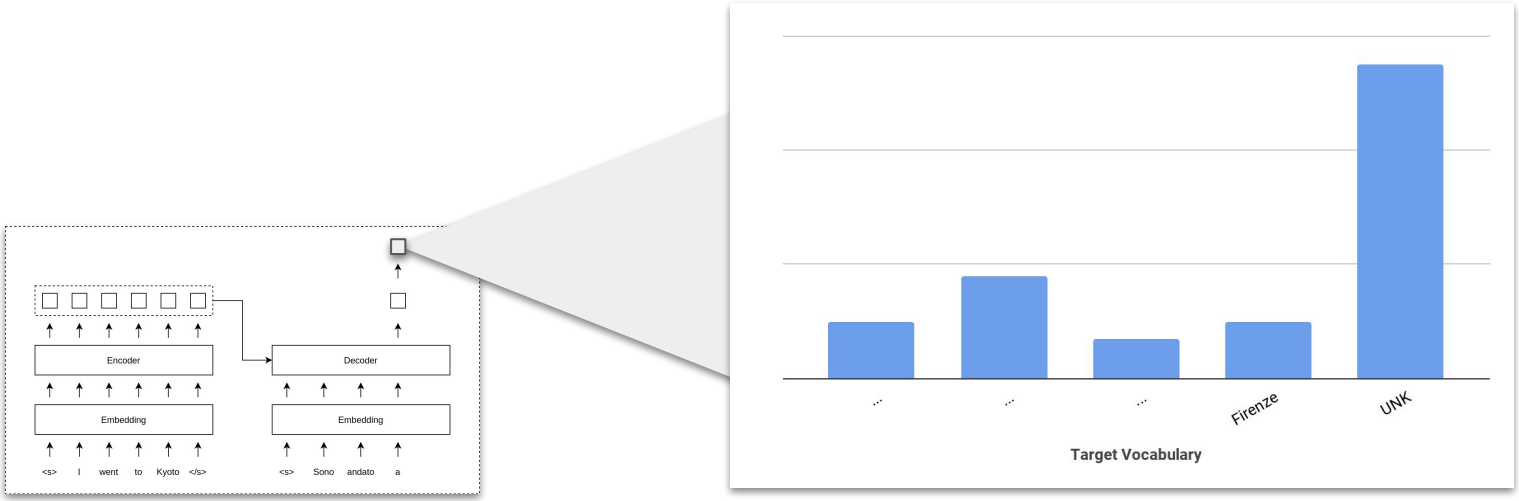
- Quick Recap of Seq2Seq
- The OOV/UNK problem
- Token Copying
  - CopyNet
  - Pointer-Generator
- Span Copying
- Conclusions

What if some token is OOV?

I went to ***Kyoto***

↦

Sono andato a **?**



SAPIENZA

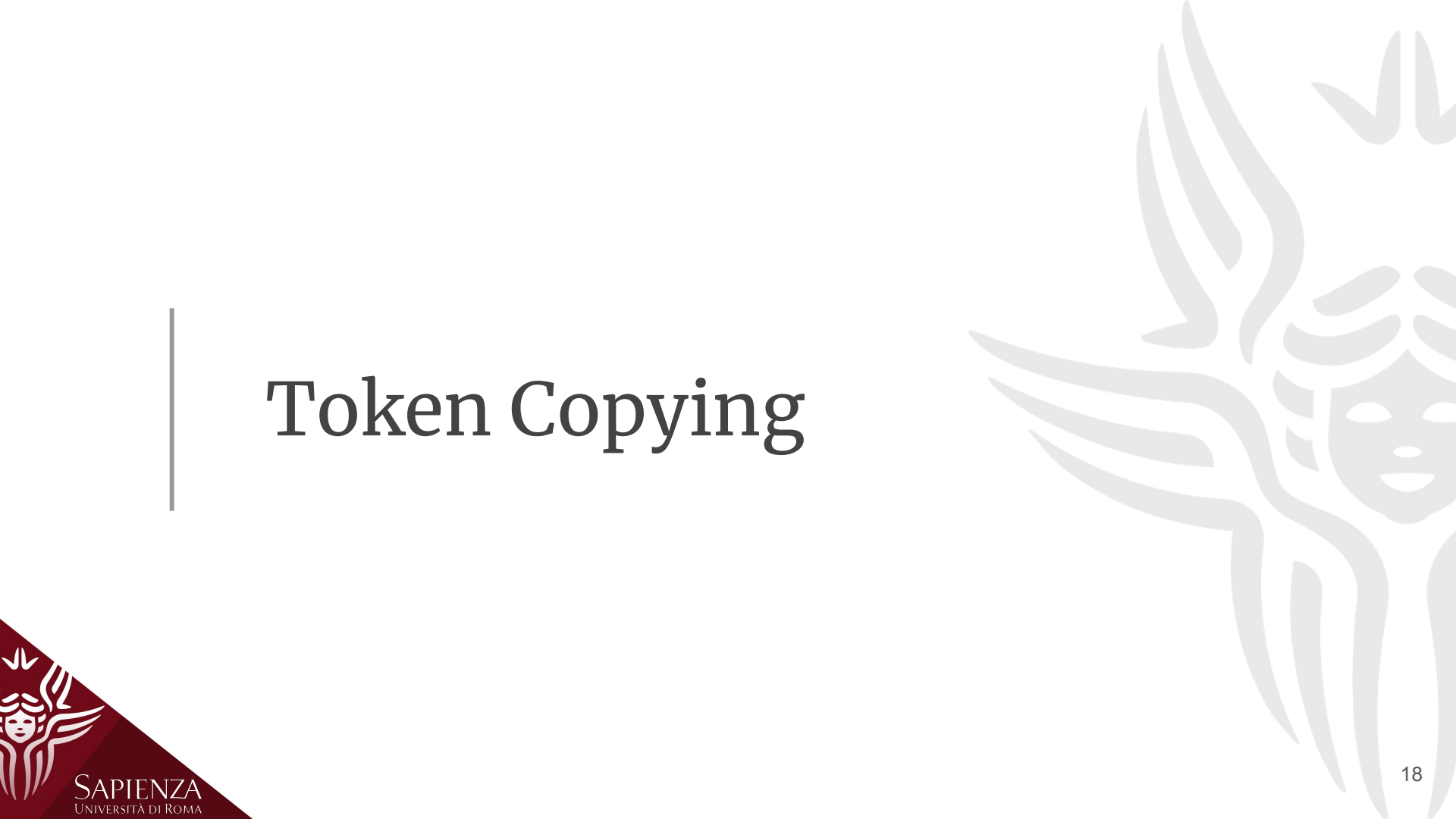
The original naive idea

Sono andato a **UNK**

- Well, we can't really output <UNK>
- Most Seq2Seq are **attentional**
  - Replace <UNK> at **post-processing** with the most attended source word
  - **Hopefully**, that will be Kyoto

Sono andato a ***Kyoto***

Token Copying



Outline

- Quick Recap of Seq2Seq
- The OOV/UNK problem
- Token Copying
  - CopyNet
  - Pointer-Generator
- Span Copying
- Conclusions

Token Copying

- The underlying attention was **not trained with this copying objective**

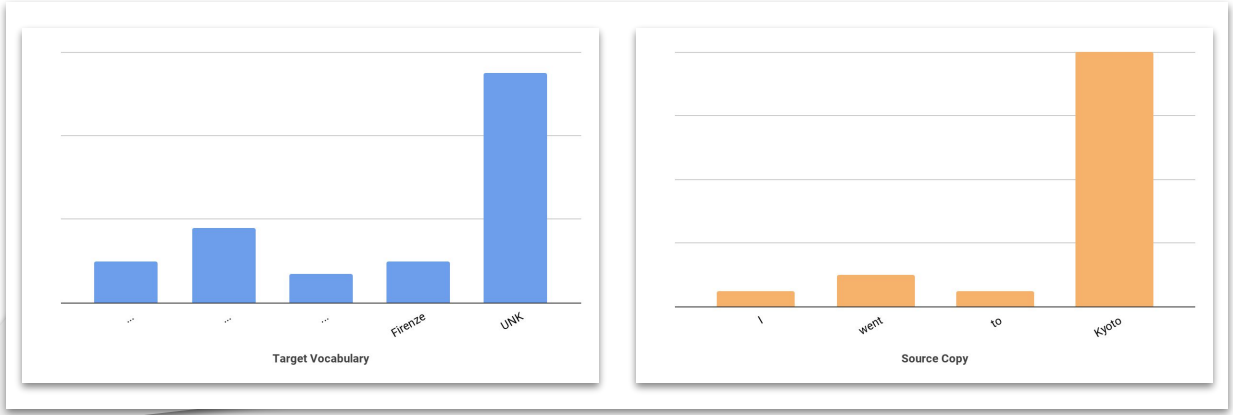
I went to ***Kyoto*** ↦ Sono andato a ***went***

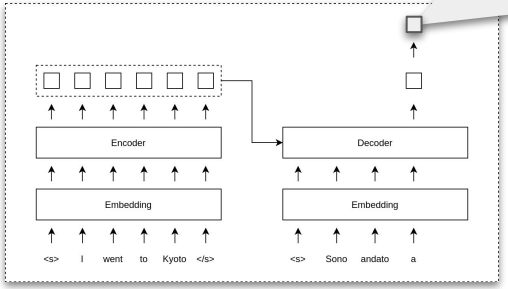
- It is better to **incorporate** it within our network
- Several different strategies
  - Copy only
  - Copy only on OOV
  - ...
  - Learn to copy or generate, **depending** on the decoder state and the considered y_j

Dynamic Vocabularies

- *Kyoto* $\notin \mathbf{V}_t$
- How do generate something that is not in our output vocabulary?
- **Dynamic vocabularies**
  - Sample $\mathbf{s}_i$, with input $\mathbf{x}_i$ and output $\mathbf{y}_i$
  - $D(\mathbf{s}_i) = \mathbf{V}_t \cup \mathbf{x}_i$
  - Stack $\mathbf{x}_i$ after $\mathbf{V}_t$

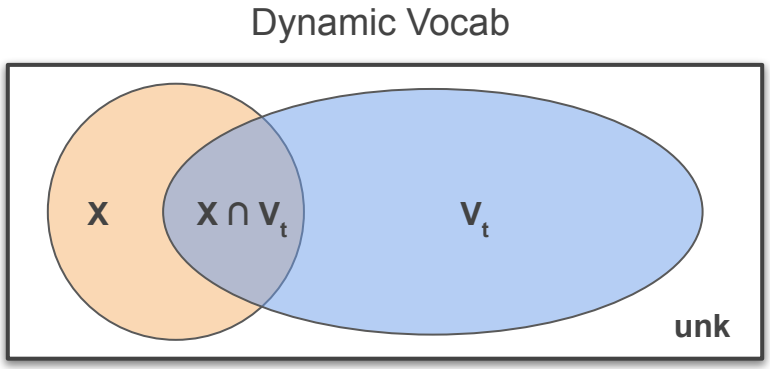
General Scheme





CopyNet

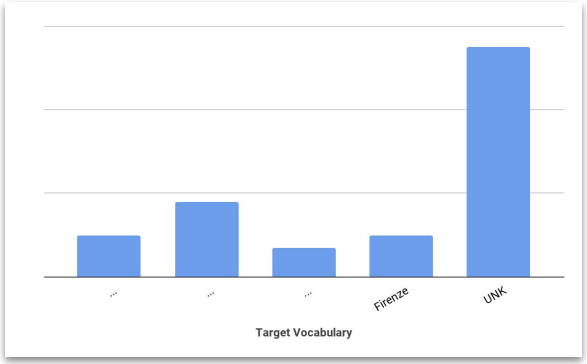
- Dynamic dictionary: output vocabulary $\mathbf{V_t}$ + set $\boldsymbol{X}$ of unique words in the input
- Use a **shared softmax**



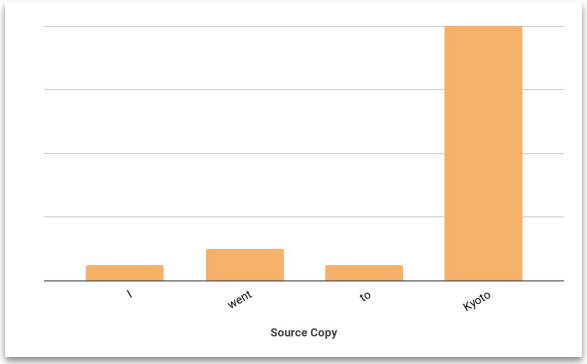
Pointer-Generator Networks

- Dynamic dictionary: output vocabulary $\mathbf{V_t}$ + words in the source sentence
- Use an explicit switching probability $\mathbf{p_c}$

$(1 - \mathbf{p_c})$



$+ \mathbf{p_c}$



Some Results

- Abstractive summarization (See et al. 2017)
  - *lead-3*: 36.57
  - *Seq2Seq*: 28.83
  - *Pointer-Generator*: **33.42**

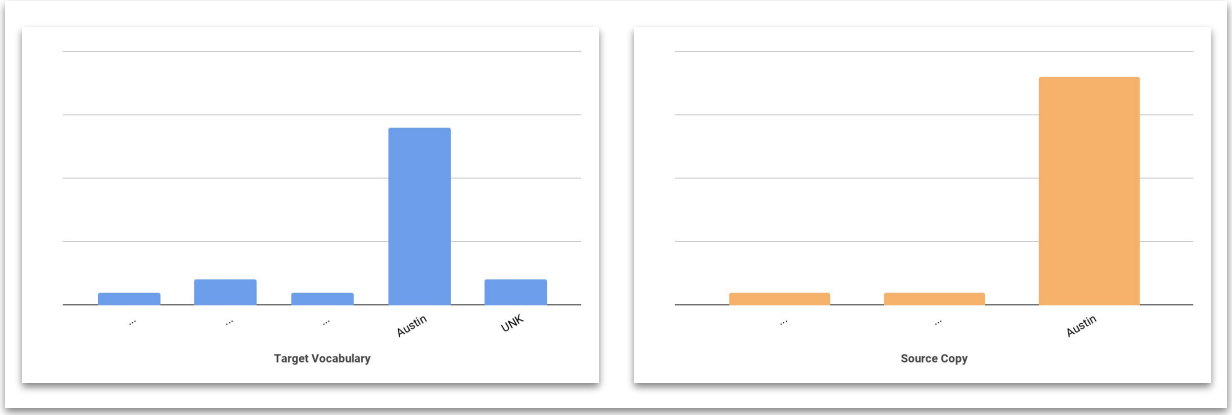
- AMR Parsing (Zhang et al. 2019)
  - *Full model*: **76.3**
  - *Without source copy*: 70.9
  - *Without target copy*: 71.6

Tricky Phenomena: *target overlap*

I went to Austin ↦ Sono andato ad **Austin**

Can be copied

Can be generated



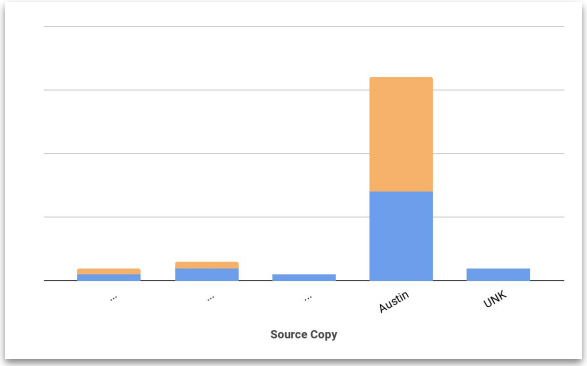
Tricky Phenomena: *target overlap*

I went to Austin ↦ Sono andato ad **Austin**

Can be copied

Can be generated

$V_s = V_t$



Easy

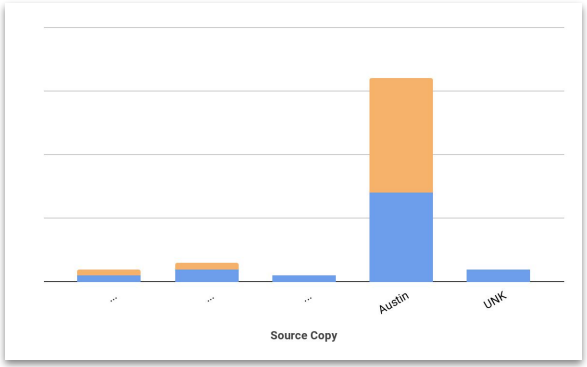
Tricky Phenomena: *target overlap*

I went to Austin ↦ Sono andato ad **Austin**

Can be copied

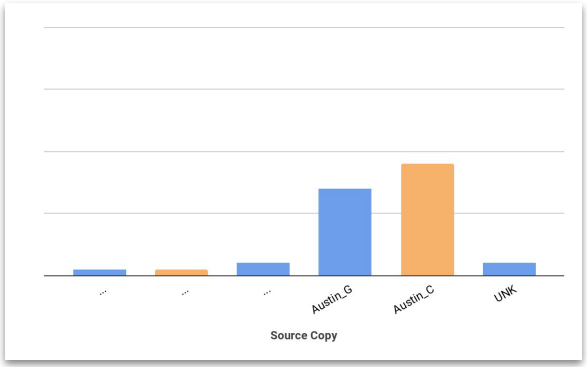
Can be generated

$V_s = V_t$



Easy

$V_s != V_t$



We need a mapping

Tricky Phenomena: *embedding(OOV)?*

I went to Austin ↦ Sono andato ad **COPY**$_{\textbf{Austin}}$

Can be copied

Can't be generated

- In order to produce y_t most Seq2Seq decoders embed y_{t-1}
- With copying, it can happen that $\mathbf{y_{t-1}}$ **is OOV** w.r.t. the target vocab and $\mathbf{y_{t-1}}$ **is not <UNK>**
- If $\mathbf{V_s}$!= $\mathbf{V_t}$, it **won't be in the embedding matrix**

Tricky Phenomena: *source vocab OOV*

I went to <UNK>$_{\text{Austin}}$ ↦ Sono andato ad **?**

- What if we have an **OOV in the source vocab**?
- We will have **UNK** in the sequence passed to the encoder (**x**)
- Both CopyNet and Pointer-Generator support this case
  a. **copy UNK**
  b. use an internal **source mapping** to replace it

Span Copying

Outline

- Quick Recap of Seq2Seq
- The OOV/UNK problem
- Token Copying
  - CopyNet
  - Pointer-Generator
- Span Copying
- Conclusions

Token Copying Shortcomings

- Several “words” span over multiple tokens (i.e. named entities)
  - New York ↦ New - York
  - SapienzaNLP Research Group ↦ SapienzaNLP - Research - Group
- In many generation tasks (i.e. text correction), **y = x** with just some minor changes
- Besides, **it’s 2020**, we have **bpe-s**
  - Los Angeles ↦ Los - Angel - es
  - Seq2Seq ↦ Seq - 2- Seq

Span Copying

I went to New York ↦ Sono andato a **New York**

- **Copy spans** rather than words

Span Copying

I went to New York ↦ Sono andato a **New York**

- **Copy spans** rather than words
- **Assumption violation:**
  - 4 actions required to generate a sequence of length 5
  - **# actions != # sequence length**

Span Copying

I went to New York ↦ Sono andato a **New York**

- **Copy spans** rather than words
- **Assumption violation:**
  - 4 actions required to generate a sequence of length 5
  - **# actions != # sequence length**
- **Regression issue:**
  - when predicting y_t, the decoder may not have seen y_{t-1}
  - For example, y_{t-2} and y_{t-1} were copied and the decoder **jumped** from y_{t-2} to y_t

Span Copying

I went to New York ↦ Sono andato a **New York**

- **Copy spans** rather than words
- **Assumption violation:**
  - 4 actions required to generate a sequence of length 5
  - **# actions != # outputs**
- **Regression issue:**
  - when predicting y_t, the decoder may not have seen y_{t-1}
  - For example, y_{t-2} and y_{t-1} were copied and the decoder **jumped** from y_{t-2} to y_t
- What should we **condition** upon, **actions or outputs**?

Sequential Copying Networks (SeqCopyNet)

- **Condition upon outputs**
- At each timestep, decide whether to copy or generate
- If it decides to generate, behave as a plain vanilla seq2seq
- If it decides to copy:
  - **select a sub-span**, i.e. predict start and stop index
  - Perform a **copy run**, i.e. run the decoder on the copied words so to update its hidden states

Some Results

- Abstractive Summarization (Zhou et al. 2018)
  - *Seq2Seq:* 33.24
  - *SeqCopyNet:* **33.35**
- Question Generation (Zhou et al. 2018)
  - *Seq2Seq:* 10.13
  - *Seq2Seq with UNK copy:* 12.18
  - *SeqCopyNet:* **13.02**

Sequential Copying Networks

$$L = -\frac{1}{n}\sum_{i=1}^{n}(\sum_{t=1}^{Ty} log\, p_g p(y_t) + \sum_{span \in C_k} log\, p_c p_{start} p_{end}))$$

- **Conceptual flaw**
- **Only full copy** and individual generation are considered correct

The prime minister of the United Kingdom is …

↦

The prime minister of the United Kingdom is … — Wrong

The prime minister of the United Kingdom is … — Correct

Copy that!

- Rejected at ICLR 2020
  - Sound approach but...
  - Too incremental for ICLR, scarce discussion of related work and not enough experiments
- **Condition upon outputs**
- Similar architecture to SeqCopyNet
- **Recursive marginalization**

$$p(\mathbf{o}_{[k:]} | \mathbf{o}_{[:k]}) = \sum_{a,l=|a| \,/\, [[a]]=\mathbf{o}_{[k:k+l]}} q(a | \mathbf{o}_{[:k]}) \cdot p(\mathbf{o}_{[k+l:]} | \mathbf{o}_{[:k+l]})$$

Some Results

- Wiki Atomic Edits (Panthaplackel et al. 2020)
  - *Token copying:* 67.8
  - *Span copying:* **78.1**
- Code Repair (Panthaplackel et al. 2020)
  - *Token Copying:* 14.8
  - *Span Copying (always longest):* 14.2
  - *Span Copying:* **17.7**

Conclusion

Conclusions

- Copying mechanisms are quite used in generation tasks
- Overall, elegant and simple tool that can boost performances (depending on the task)
  - Helps with those **OOV** that appear in the source sentence
  - Introduce some nice **task bias**
- Recent works on span copying seem promising in several generation tasks

Thank you for your attention

45

References

- Incorporating Copying Mechanism in Sequence-to-Sequence Learning
- Get To The Point: Summarization with Pointer-Generator Networks
- Sequential Copying Networks
- Copy that! Editing Sequences by Copying Spans