LECTURE 27: INTRO TO LARGE LANGUAGE MODELS

Mehmet Can Yavuz, PhD.

Adapted from Julia Hockenmaier, NLP S2023 - course material
https://courses.grainger.illinois.edu/cs447/sp2023/

TODAY'S CLASS

01
Recap: Using RNNs for various NLP tasks

02
From static to contextual embeddings: ELMO

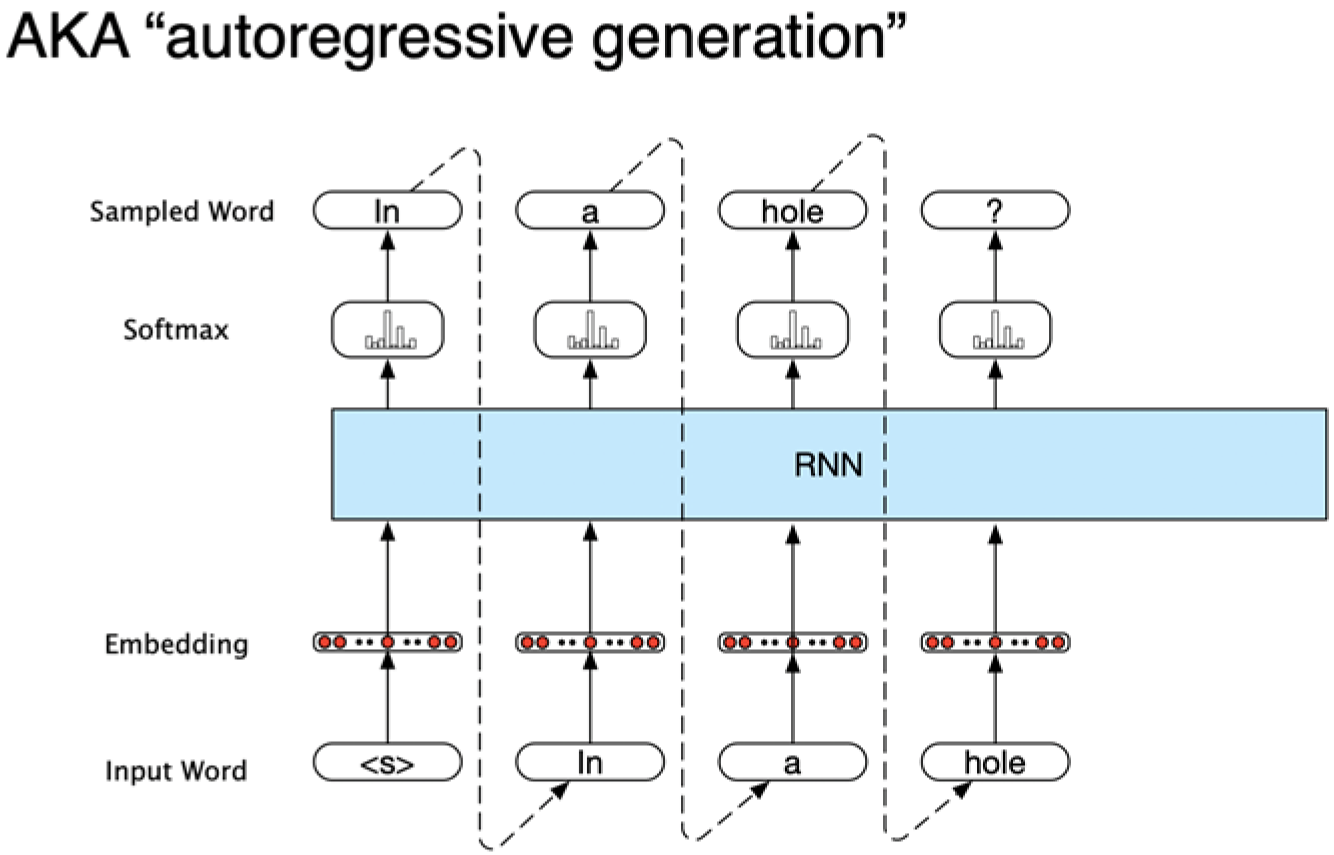
03
Recap: Transformers

04
Subword tokenizations

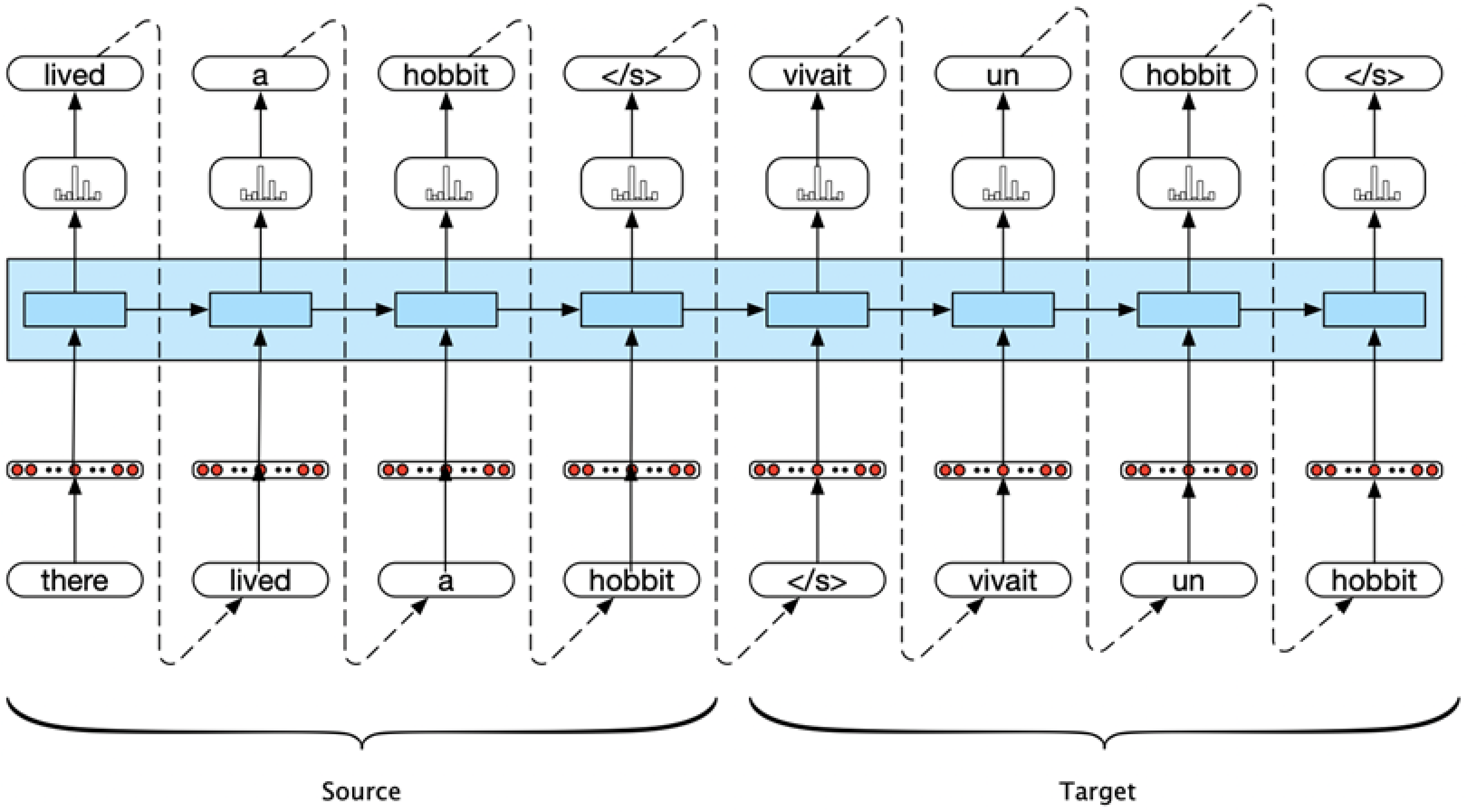
05
Early Large Language Models (GPT, BERT)

RECAP: USING RNNs FOR DIFFERENT NLP TASKS

RNNS FOR LANGUAGE GENERATION

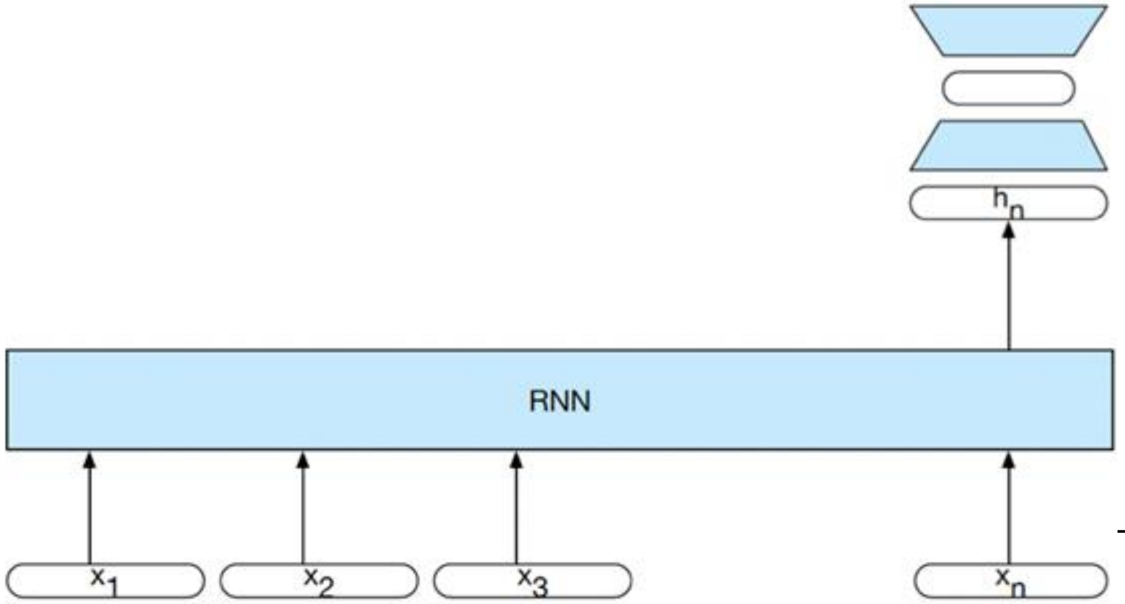


AN RNN FOR MACHINE TRANSLATION



RNNS FOR SEQUENCE CLASSIFICATION

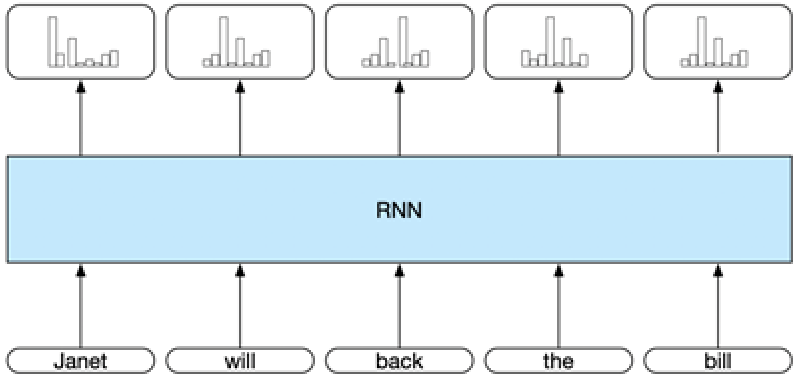
- If we just want to assign **one label** to the entire sequence, we don't need to produce output at each time step, so we can use a simpler architecture.
- We can use the hidden state of the last word in the sequence as input to a feedforward net:



BASIC RNNS FOR SEQUENCE LABELING

Sequence labeling (e.g. POS tagging):
Assign **one label to each element** in the sequence.

RNN Architecture:
Each time step has a distribution over output classes



Extension: add a CRF layer to capture dependencies among labels of adjacent tokens.

ELMo

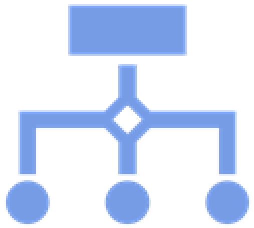
EMBEDDINGS FROM LANGUAGE MODELS

Replace static embeddings (lexicon lookup) with **context-dependent embeddings** (produced by a **neural language model**)

- => **Each token**'s representation is a **function of the entire input sentence**, computed by a deep (multi-layer) bidirectional language model
- => Return for each token a **(task-dependent) linear combination of its representation across layers**.
- => Different layers capture different information

Peters et al., NAACL 2018

ELMo

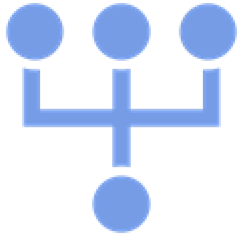


Pre-training:

— Train a **multi-layer bidirectional language model** with character convolutions on **raw text**

— **Each layer** of this language model network computes **a vector** representation **for each token.**

— **Freeze the language model** parameters.



Fine-tuning (for each task)

Train **task-dependent softmax** weights to **combine the layer-wise representations** into **a single vector for each token** *jointly* with **a task-specific model that uses those vectors**

ELMO'S INPUT TOKEN REPRESENTATIONS

The input token representations are purely **character- based:** a character CNN, followed by linear projection to reduce dimensionality

"2048 character n-gram convolutional filters with two highway layers, followed by a linear projection to 512 dimensions"

Advantage over using fixed embeddings: no UNK tokens, any word can be represented

ELMo's BIDIRECTIONAL LANGUAGE MODELS

Forward LM: a deep LSTM that goes over the sequence from start to end to predict token t_k based on the prefix $t_1 \ldots t_{k-1}$:

$$p(t_k \mid t_1, \ldots, t_{k-1}; \Theta_x, \overrightarrow{\Theta}_{LSTM}, \Theta_s)$$

Parameters: token embeddings Θ_x LSTM $\overrightarrow{\Theta}_{LSTM}$, softmax Θ_s

Backward LM: a deep LSTM that goes over the sequence from end to start to predict token t_k based on the suffix $t_{k+1} \ldots t_N$:

$$p(t_k \mid t_{k+1}, \ldots, t_N; \Theta_x, \overleftarrow{\Theta}_{LSTM}, \Theta_s)$$

Train these LMs jointly, with the same parameters for the token representations and the softmax layer (but not for the LSTMs)

$$\sum_{k=1}^{N} \left(\log p(t_k \mid t_1, \ldots, t_{k-1}; \Theta_x, \overrightarrow{\Theta}_{LSTM}, \Theta_s) + \log p(t_k \mid t_{k+1}, \ldots, t_N; \Theta_x, \overleftarrow{\Theta}_{LSTM}, \Theta_s) \right)$$

ELMO'S OUTPUT TOKEN REPRESENTATIONS

Given **an input token representation $\mathbf{x}_k$,**
each layer j of the LSTM language models computes
a vector representation $\mathbf{h}_{k,j}$ for every token k.

With L layers, ELMo represents each token as L vectors $\mathbf{h}_{k,l}^{LM}$

$$
\begin{aligned}
R_k &= \{\mathbf{x}_k^{LM}, \overrightarrow{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM} \mid j = 1, \ldots, L\} \\
&= \{\mathbf{h}_{k,j}^{LM} \mid j = 0, \ldots, L\},
\end{aligned}
$$

where $\mathbf{h}_{k,j}^{LM} = [\overrightarrow{\mathbf{h}}_{k,j}^{LM}; \overleftarrow{\mathbf{h}}_{k,j}^{LM}]$ and $\mathbf{h}_{k,0}^{LM} = \mathbf{x}_k$

ELMo learns **softmax weights** s_j^{task} and a **task-specific scalar** γ^{task}
to collapse these L vectors into **a single task-specific token vector:**

$$
\mathbf{ELMo}_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \mathbf{h}_{k,j}^{LM}.
$$

RESULTS

ELMo gave improvements on a variety of tasks:

— question answering (SQuAD)

— entailment/natural language inference (SNLI)

— semantic role labeling (SRL)

— coreference resolution (Coref)

— named entity recognition (NER)

— sentiment analysis (SST-5)

TASK	PREVIOUS SOTA		OUR BASELINE	ELMO + BASELINE	INCREASE (ABSOLUTE/ RELATIVE)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

ELMo:

ELMo showed that **contextual embeddings** are very useful: it outperformed other models on many tasks

- ELMo embeddings could also be concatenated with other token-specific features, depending on the task

ELMo requires training a task-specific softmax and scalar to predict how best to combine each layer

- Not all layers were equally useful for each task