Knowledge-Augmented Language Model and Its Application to Unsupervised Named-Entity Recognition

Angli Liu
Facebook AI
anglil@cs.washington.edu

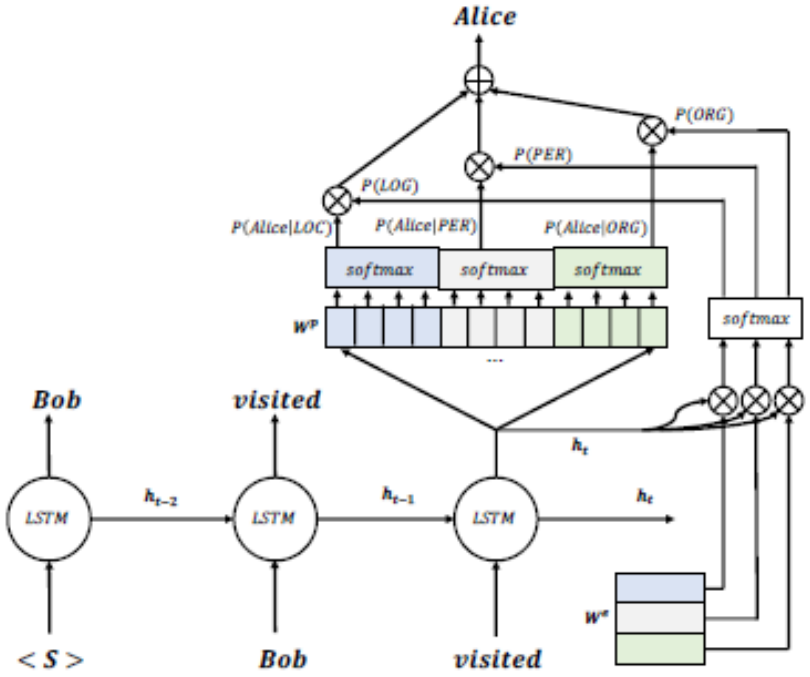
Jingfei Du
Facebook AI
jingfeidu@fb.com

Veselin Stoyanov
Facebook AI
ves@fb.com

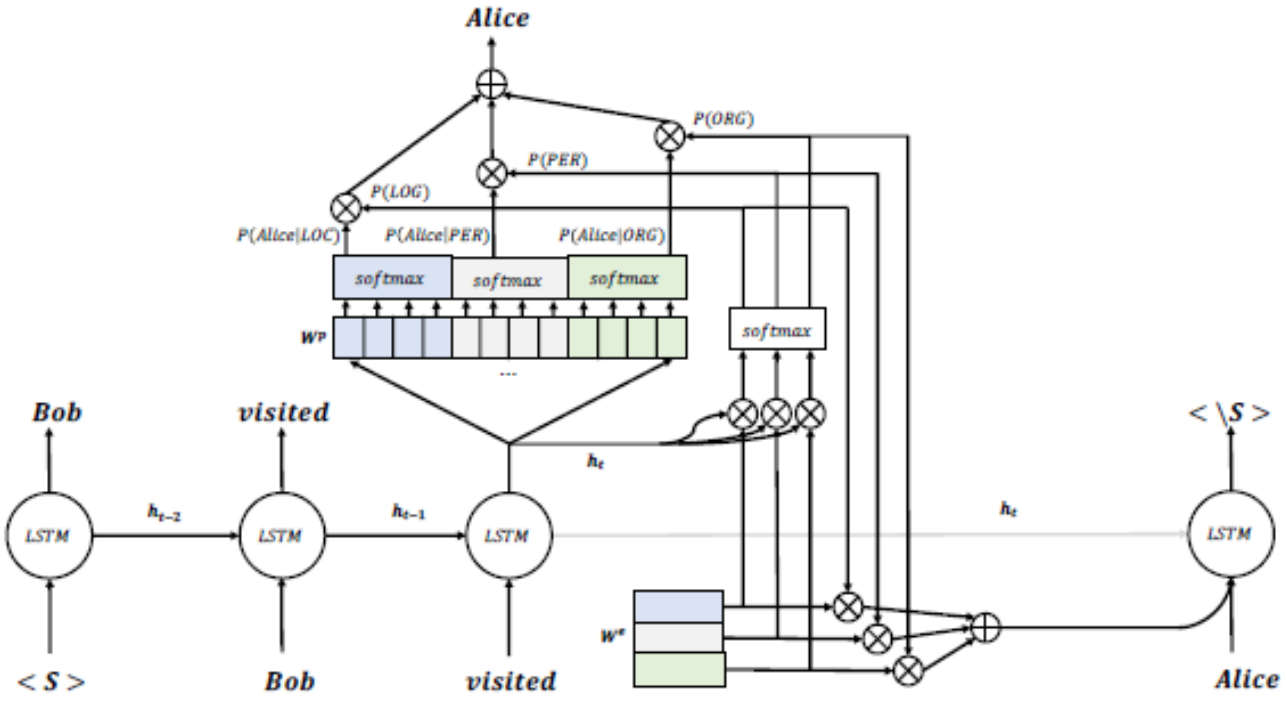
Summary

- The paper proposes Knowledge Augmented Language Model (**KALM**), a language model with access to information available in a KB.
- Learns a named entity recognizer without any explicit supervision by using only plain text.
- Learns to recognize named entities completely unsupervised by interpreting the predictions of the gating mechanism at test time.

Model



(a) Basic model architecture of KALM.



(b) Adding type representation as input to KALM.

Knowledge-Augmented Language Model

KALM extends a traditional, RNN-based neural LM. As in traditional LM, KALM predicts probabilities of words from a vocabulary V_g, but it can also generate words that are names of entities of a specific type. Each entity type has a separate vocabulary $\{V_1, ..., V_K\}$ collected from a KB. KALM learns to predict from context whether to expect an entity from a given type and generalizes over entity types.

3.1 RNN language model

$$P(y_{t+1} = i|c_t) = \frac{\exp(\boldsymbol{W}^p_{i,:} \cdot \boldsymbol{h}_t)}{\sum_{w=1}^{|V_g|} \exp(\boldsymbol{W}^p_{w,:} \cdot \boldsymbol{h}_t)} \quad (1)$$

$$\boldsymbol{h}_t, \boldsymbol{\gamma}_t = lstm(\boldsymbol{h}_{t-1}, \boldsymbol{\gamma}_{t-1}, \boldsymbol{y}_t) \quad (2)$$

where $lstm$ refers to the LSTM step function and h_i, $\boldsymbol{\gamma}_i$ and y_i are the hidden, memory and input vectors, respectively. $\boldsymbol{W}^p$ is a projection layer that converts LSTM hidden states into logits that have the size of the vocabulary $|V_g|$.

3.2 Knowledge-Augmented Language Model

KALM builds upon the LSTM LM by adding type-specific entity vocabularies $V_1, V_2, ..., V_K$ in addition to the general vocabulary V_g. Type vocabularies are extracted from the entities of specific type in a KB. For a given word, KALM computes a probability that the word represents an entity of that type by using a type-specific projection matrix $\{\boldsymbol{W}^{p,j} | j = 0, ..., K\}$. The model also computes the probability that the next word represents different entity types given the context observed so far. The overall probability of a word is given by the weighted sum of the type probabilities and the probability of the word under the give type.

More precisely, let τ_i be a latent variable denoting the type of word i. We decompose the probability in Equation 1 using the type τ_{t+1}:

$$\begin{aligned} P(y_{t+1}|c_t) &= \sum_{j=0}^{K} P(y_{t+1}, \tau_{t+1} = j|c_t) \\ &= \sum_{j=0}^{K} P(y_{t+1}|\tau_{t+1} = j, c_t) \\ &\quad \cdot P(\tau_{t+1} = j|c_t) \end{aligned} \tag{3}$$

$$P(y_{t+1} = i|\tau_{t+1} = j, c_t) = \frac{\exp(\boldsymbol{W}^{p,j}_{i,:} \cdot \boldsymbol{h}_t)}{\sum_{w=1}^{|V_j|} \exp(\boldsymbol{W}^{p,j}_{w,:} \cdot \boldsymbol{h}_t)} \tag{4}$$

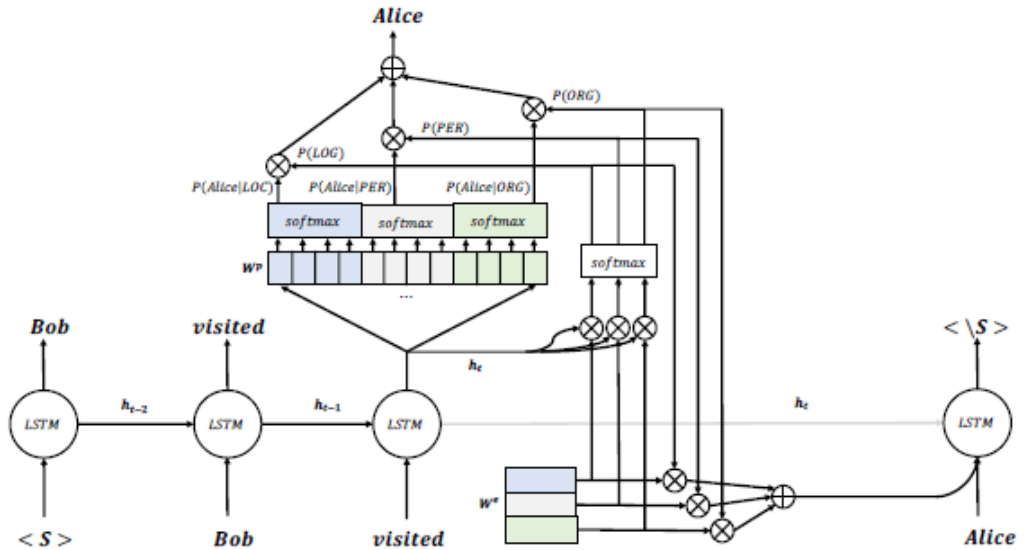
$$P(\tau_{t+1} = j|c_t) = \frac{\exp(\boldsymbol{W}^{e}_{j,:} \cdot (\boldsymbol{W}^{h} \cdot \boldsymbol{h}_t))}{\sum_{k=0}^{K} \exp(\boldsymbol{W}^{e}_{k,:} \cdot (\boldsymbol{W}^{h} \cdot \boldsymbol{h}_t))} \tag{5}$$

3.3 Type representation as input

In the base KALM model the input for word y_t consists of its embedding vector $\boldsymbol{y}_t$. We enhance the base model by adding as inputs the embedding of the type of the previous word. As type information is latent, we represent it as the weighted sum of the type embeddings weighted by the predicted probabilities:

$$\boldsymbol{\nu}_{t+1} = \sum_{j=0}^{K} P(\tau_{t+1} = j | c_t) \cdot \boldsymbol{W}^e_{j,:} \qquad (6)$$

$$\tilde{\boldsymbol{y}}_{t+1} = [\boldsymbol{y}_{t+1}; \boldsymbol{\nu}_{t+1}] \qquad (7)$$



(b) Adding type representation as input to KALM.

4 Unsupervised NER

The type distribution that KALM learns is latent, but we can output it at test time and use it to predict whether a given word refers to an entity or a general word. We compute $P(\tau_{t+1}|c_t)$ using eq. 5 and use the most likely entity type as the named entity tag for the corresponding word y_{t+1}.

This straightforward approach, however, predicts the type based solely on the left context of the tag being predicted. In the following two subsections, we discuss extensions to KALM that allow it to utilize the right context and the word being predicted itself.

$$P(\tau_{t+1} = j|c_t) = \frac{\exp(\boldsymbol{W}^e_{j,:} \cdot (\boldsymbol{W}^h \cdot \boldsymbol{h}_t))}{\sum_{k=0}^{K} \exp(\boldsymbol{W}^e_{k,:} \cdot (\boldsymbol{W}^h \cdot \boldsymbol{h}_t))} \quad (5)$$

$$\boldsymbol{h}_{l,t}, \boldsymbol{\gamma}_{l,t} = lstm_l(\boldsymbol{h}_{l,t-1}, \boldsymbol{\gamma}_{l,t-1}, [\boldsymbol{y}_{l,t}; \boldsymbol{\nu}_{l,t}]) \quad (8)$$

$$\boldsymbol{h}_{r,t}, \boldsymbol{\gamma}_{r,t} = lstm_r(\boldsymbol{h}_{r,t+1}, \boldsymbol{\gamma}_{r,t+1}, [\boldsymbol{y}_{r,t}; \boldsymbol{\nu}_{r,t}]) \quad (9)$$

$$\begin{aligned} P(\tau_t|c_l, c_r, y_t) =& \frac{P(y_t|\tau_t, c_l, c_r)}{2P(y_t|c_l, c_r)} \cdot P(\tau_t|c_l, c_r) \\ &+ \frac{P(c_l, c_r|\tau_t, y_t)}{2P(c_l, c_r|y_t)} \cdot P(\tau_t|y_t) \\ =& \alpha \cdot P(\tau_t|c_l, c_r) \\ &+ \beta \cdot P(\tau_t|y_t) \end{aligned} \quad (10)$$

4.2.2 Training with type priors

An alternative for incorporating the pre-computed $P(\tau_t|y_t)$ is to use it during training to regularize the type distribution. We use the following optimization criterion to compute the loss for each word:

$$\begin{aligned} L =& H(P(y_i|c_l, c_r), P(\hat{y}_i|c_l, c_r)) \\ &+ \lambda \cdot ||KL(P(\tau_i|c_l, c_r), P(\tau_i|y_i))||^2 \end{aligned} \quad (11)$$

where $\hat{y}_i$ is the actual word, $H(.)$ is the cross entropy function, and $KL(.)$ measures the KL divergence between two distributions. A hyper-parameter λ (tuned on validation data) controls the relative contribution of the two loss terms. The new loss forces the learned type distribution, $P(\tau_i|c_l, c_r)$, to be close to the expected distribution $P(\tau_i|y_i)$ given the information in the database. This loss is specifically tailored to help with unsupervised NER.

Datasets

- ***Recipe*** dataset is composed of 95786 recipes.
- ***CoNLL 2003*** dataset is composed of news articles.

Vocabulary We use the entity words in Table 2 to form $V_1, ..., V_K$ We extract $51,677$ general words in the recipe dataset, and $17,907$ general words in CoNLL 2003 to form V_0. Identical

type	dairy	drinks	fruits	grains
#entities	80	84	110	158
type	proteins	seasonings	sides	vegetables
#entities	316	180	140	156

type	LOC	MISC	ORG	PER
#entity words	1503	1211	3005	5404

Table 2: Statistics of recipe and CoNLL 2003 KBs

Language modeling

	model	unidirectional		bidirectional	
		validation	test	validation	test
Recipe	AWD-LSTM	3.14	2.99	1.98	2
	NE-LM	2.96	2.24	1.85	1.73
	KALM	**2.75**	**2.20**	**1.85**	**1.71**
CoNLL 2003	AWD-LSTM	5.48	5.94	4.85	5.3
	NE-LM	5.67	5.77	4.68	4.94
	KALM	**5.36**	**5.43**	**4.64**	**4.74**

Table 3: Language modeling results on the recipe and CoNLL 2003 datasets

NER

		LOC	MISC	ORG	PER	overall
unsupervised	basic^0	0.75	0.67	0.64	0.83	0.72
	+$P(\tau\|y)$ in dec^1	0.81	0.67	0.65	0.88	0.76
	+wiki-concat^2	0.83	0.83	0.76	0.95	0.84
	+wiki-concat+$P(\tau\|y)$ in dec^3	0.84	**0.84**	0.77	**0.96**	0.86
supervised	biLSTM	0.88	0.71	0.81	0.95	0.86
	CRF-biLSTM	**0.90**	0.76	**0.84**	0.95	**0.89**

Table 5: Results of KALM (above the double line in the table) and supervised NER models (under the double line). Superscript annotations: 0*: The basic bidirectional KALM with type embedding features as described in Section 4.1.* 1*: Adding* $P(\tau|y)$ *in decoding, as described in Section 4.2.1, where* α *and* β *are tuned to be* 0.4 *and* 0.6. 2*: The basic model trained on CoNLL 2003 concatenated with WikiText-2.* 3*: Adding* $(\tau|y)$ *in decoding, with the model trained on CoNLL 2003 concatenated with WikiText-2.*