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

Abstract

Traditional language models are unable to efficiently model entity names observed in text. All but the most popular named entities appear infrequently in text providing insufficient context. Recent efforts have recognized that context can be generalized between entity names that share the same type (e.g., *person* or *location*) and have equipped language models with access to an external knowledge base (KB). Our Knowledge-Augmented Language Model (KALM) continues this line of work by augmenting a traditional model with a KB. Unlike previous methods, however, we train with an end-to-end predictive objective optimizing the perplexity of text. We do not require any additional information such as named entity tags. In addition to improving language modeling performance, KALM learns to recognize named entities in an entirely unsupervised way by using entity type information latent in the model. On a Named Entity Recognition (NER) task, KALM achieves performance comparable with state-of-the-art supervised models. Our work demonstrates that named entities (and possibly other types of world knowledge) can be modeled successfully using predictive learning and training on large corpora of text without any additional information.

1 Introduction

Language modeling is a form of unsupervised learning that allows language properties to be learned from large amounts of unlabeled text. As components, language models are useful for many Natural Language Processing (NLP) tasks such as generation (Parvez et al., 2018) and machine translation (Bahdanau et al., 2014). Additionally, the form of predictive learning that language modeling uses is useful to acquire text representations that can be used successfully to improve a number of downstream NLP tasks (Peters et al., 2018; Devlin et al., 2018). In fact, models pre-trained with a predictive objective have provided a new state-of-the-art by a large margin.

Current language models are unable to encode and decode factual knowledge such as the information about entities and their relations. Names of entities are an open class. While classes of named entities (e.g., *person* or *location*) occur frequently, each individual name (e.g, *Atherton* or *Zhouzhuang*) may be observed infrequently even in a very large corpus of text. As a result, language models learn to represent accurately only the most popular named entities. In the presence of external knowledge about named entities, language models should be able to learn to generalize across entity classes. For example, knowing that *Alice* is a name used to refer to a person should give ample information about the context in which the word may occur (e.g., *Bob visited <u>Alice</u>*).

In this work, we propose Knowledge Augmented Language Model (KALM), a language model with access to information available in a KB. Unlike previous work, we make no assumptions about the availability of additional components (such as Named Entity Taggers) or annotations. Instead, we enhance a traditional LM with a gating mechanism that controls whether a particular word is modeled as a general word or as a reference to an entity. We train the model end-to-end with only the traditional predictive language modeling perplexity objective. As a result, our system can model named entities in text more accurately as demonstrated by reduced perplexities compared to traditional LM baselines. In addition, KALM learns to recognize named entities completely unsupervised by interpreting the predictions of the gating mechanism at test time. In fact, KALM learns an unsupervised named entity tagger that rivals in accuracy supervised counterparts.

1142

KALM works by providing a language model with the option to generate words from a set of entities from a database. An individual word can either come from a general word dictionary as in traditional language model or be generated as a name of an entity from a database. Entities in the database are partitioned by type. The decision of whether the word is a general term or a named entity from a given type is controlled by a gating mechanism conditioned on the context observed so far. Thus, KALM learns to predict whether the context observed is indicative of a named entity of a given type and what tokens are likely to be entities of a given type.

The gating mechanism at the core of KALM is similar to attention in Neural Machine Translation (Bahdanau et al., 2014). As in translation, the gating mechanism allows the LM to represent additional latent information that is useful for the end task of modeling language. The gating mechanism (in our case entity type prediction) is latent and learned in an end-to-end manner to maximize the probability of observed text. Experiments with named entity recognition show that the latent mechanism learns the information that we expect while LM experiments show that it is beneficial for the overall language modeling task.

This paper makes the following contributions:

- Our model, KALM, achieves a new state-of-the art for Language Modeling on several benchmarks as measured by perplexity.

- We learn a named entity recognizer without any explicit supervision by using only plain text. Our unsupervised named entity recognizer achieves a performance on par with the state-of-the supervised methods.

- We demonstrate that predictive learning combined with a gating mechanism can be utilized efficiently for generative training of deep learning systems beyond representation pre-training.

2 Related Work

Our work draws inspiration from Ahn et al. (2016), who propose to predict whether the word to generate has an underlying fact or not. Their model can generate knowledge-related words by copying from the description of the predicted fact. While theoretically interesting, their model func-

tions only in a very constrained setting as it requires extra information: a shortlist of candidate entities that are mentioned in the text.

Several efforts successfully extend LMs with entities from a knowledge base and their types, but require that entity models are trained separately from supervised entity labels. Parvez et al. (2018) and Xin et al. (2018) explicitly model the type of the next word in addition to the word itself. In particular, Parvez et al. (2018) use two LSTM-based language models, an entity type model and an entity composite (entity type) model. Xin et al. (2018) use a similarly purposed entity typing module and a LM-enhancement module. Instead of entity type generation, Gu et al. (2018) propose to explicitly decompose word generation into sememe (a semantic language unit of meaning) generation and sense generation, but requires sememe labels. Yang et al. (2016) propose a pointer-network LM that can point to a 1-D or 2-D database record during inference. At each time step, the model decides whether to point to the database or the general vocabulary.

Unsupervised predictive learning has been proven effective in improving text understanding. ELMo (Peters et al., 2018) and BERT (Devlin et al., 2018) used different unsupervised objectives to pre-train text models which have advanced the state-of-the-art for many NLP tasks. Similar to these approaches KALM is trained end-to-end using a predictive objective on large corpus of text.

Most unsupervised NER models are rule-based (Collins and Singer, 1999; Etzioni et al., 2005; Nadeau et al., 2006) and require feature engineering or parallel corpora (Munro and Manning, 2012). Yang and Mitchell (2017) incorporate a KB to the CRF-biLSTM model (Lample et al., 2016) by embedding triples from a KB obtained using *TransE* (Bordes et al., 2013). Peters et al. (2017) add pre-trained language model embeddings as knowledge to the input of a CRF-biLSTM model, while still requiring labels in training.

To the best of our knowledge, KALM is the first unsupervised neural NER approach. As we discuss in Section 5.4, KALM achieves results comparable to supervised CRF-biLSTM models.

3 Knowledge-Augmented Language Model

KALM extends a traditional, RNN-based neural LM. As in traditional LM, KALM predicts prob-

abilities 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

At its core, a language model predicts a distribution for a word y_{t+1} given previously observed words $c_t := [y_1, ..., y_{t-1}, y_t]$. Models are trained by maximizing the likelihood of the observed next word. In an LSTM LM, the probability of a word, $P(y_{t+1}|c_t)$, is modeled from the hidden state of an LSTM (Hochreiter and Schmidhuber, 1997):

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

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

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}:

$$P(y_{t+1}|c_t) = \sum_{j=0}^{K} P(y_{t+1}, \tau_{t+1} = j|c_t)$$

$$= \sum_{j=0}^{K} P(y_{t+1}|\tau_{t+1} = j, c_t)$$

$$\cdot P(\tau_{t+1} = j|c_t) \quad (3)$$

Where $P(y_{t+1}|\tau_{t+1}, c_t)$ is a distribution of entity words of type τ_{t+1}. As in a general LM, it is computed by projecting the hidden state of the LSTM and normalizing through softmax (eq. 4). The type-specific projection matrix $\boldsymbol{W}^{p,j}$ is learned during training.

We maintain a type embedding matrix $\boldsymbol{W}^e$ and use it in a similar manner to compute the probability that the next word has a given type $P(\tau_{t+1}|c_t)$ (eq. 5). The only difference is that we use an extra projection matrix, $\boldsymbol{W}^h$ to project $\boldsymbol{h}_t$ into lower dimensions. Figure 1a illustrates visually the architecture of KALM.

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

$$(4)$$

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

$$(5)$$

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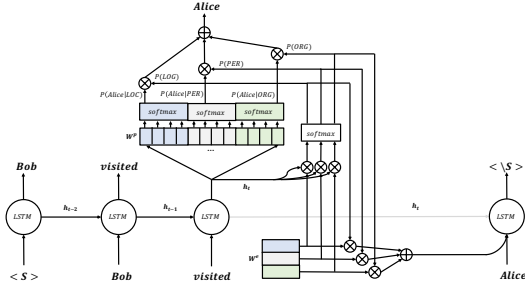
$$\boldsymbol{\nu}_{t+1} = \sum_{j=0}^{K} P(\tau_{t+1} = j|c_t) \cdot \boldsymbol{W}^e_{j,:} \quad (6)$$

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

$P(\tau_{t+1} = j|c_t)$ is computed using Equation 5 and e_j is the type embedding vector.
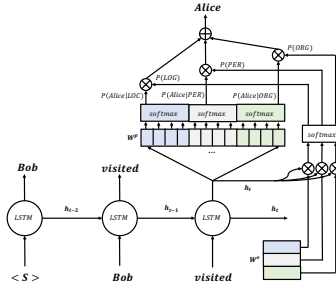
(a) Basic model architecture of KALM.

(b) Adding type representation as input to KALM.

Figure 1: KALM's architectures

Adding type information as input serves two purposes: in the forward direction, it allows KALM to model context more precisely based on predicted entity types. During back propagation, it allows us to learn latent types more accurately based on subsequent context. The model enhanced with type input is illustrated in Figure 1b.

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.

4.1   Bidirectional LM

While we cannot use a bidirectional LSTM for generation, we can use one for NER, since the entire sentence is known.

For each word, KALM generates the hidden vectors $h_{l,t}$ and $h_{r,t}$ representing context coming from left and right directions, as shown in Equations 8 and 9.

$$h_{l,t}, \gamma_{l,t} = lstm_l(h_{l,t-1}, \gamma_{l,t-1}, [y_{l,t}; \nu_{l,t}]) \quad (8)$$

$$h_{r,t}, \gamma_{r,t} = lstm_r(h_{r,t+1}, \gamma_{r,t+1}, [y_{r,t}; \nu_{r,t}]) \quad (9)$$

We concatenate the hidden vectors from the two directions to form an overall context vector h_t, and generate the final type distribution using Equation 5.

Training the bidirectional model requires that we initialize the hidden and cell states from both ends of a sentence. Suppose the length of a sentence is n. The the cross entropy loss is computed for only the $n-2$ symbols in the middle. Similarly, we compute only the types of the $n-2$ symbols in the middle during inference.

4.2   Current word information

Even bidirectional context is insufficient to predict the word type by itself. Consider the following example: *Our computer models indicate Edouard is going to* ___[1] The missing word can be either a location (e.g., *London*), or a general word (e.g., *quit*). In an NER task we observe the underlined words: *Our computer models indicate Edouard is going to London.* In order to learn predictively, we cannot base the type prediction on the current token. Instead, we can use a prior type information $P(\tau_t|y_t)$, pre-computed from entity popularity information available in many KBs. We incorporate the prior information $P(\tau_t|y_t)$ in two different ways described in the two following subsections.

4.2.1   Decoding with type prior

We incorporate the type prior $P(\tau_t|y_t)$ directly by combining it linearly with the predicted type:

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

The coefficients α and β are free parameters and are tuned on a small amount of withheld data.

---

[1] All the examples are selected from the CoNLL 2003 training set.

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:

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

where $\hat{y}_i$ is the actual word, $H(.)$ is the cross entropy function, and $KL(.)$ measures the KL divergence between two distributions. A hyperparameter 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.

5 Experiments

We evaluate KALM on two tasks: language modeling and NER. We use two datasets: *Recipe* used only for LM evaluation and *CoNLL 2003* used for both the LM and NER evaluations.

5.1 Data

Recipe The recipe dataset[2] is composed of $95,786$ recipes, We follow the same preprocessing steps as in Parvez et al. (2018) and divide the crawled dataset into training, validation and testing. A typical sentence after preprocessing looks like the following: "in a large mixing bowl combine the butter sugar and the egg yolks". The entities in the recipe KB are recipe ingredients. The 8 supported entity types are *dairy*, *drinks*, *fruits*, *grains*, *proteins*, *seasonings*, *sides*, and *vegetables*. In the sample sentence above, the entity names are *butter*, *sugar*, *egg* and *yolks*, typed as *dairy*, *seasonings*, *proteins* and *proteins*, respectively.

CoNLL 2003 Introduced in Tjong Kim Sang and De Meulder (2003), the CoNLL 2003 dataset is composed of news articles. It contains text and named entity labels in English, Spanish, German and Dutch. We experiment only with the English version. We follow the CoNLL labels and separate the KB into four entity types: *LOC* (loca-

---

[2]Crawled from http://www.ffts.com/recipes.htm

tion), *MISC* (miscellaneous), *ORG* (organization), and *PER* (person).

Statistics about the recipe and the CoNLL 2003 dataset are presented in Table 1.

	train	valid	test
#sent	61302	15326	19158
#tok	7223474	1814810	2267797
	train	valid	test
#sent	14986	3465	3683
#tok	204566	51577	46665

Table 1: Statistics of recipe and CoNLL 2003 datasets

The information about the entities in each of the KBs is shown in Table 2. The recipe KB is provided along with the recipe dataset[3] as a conglomeration of typed ingredients. The KB used by CoNLL 2003 is extracted from WikiText-2. We filtered the entities which are not belonging to the 4 types of CoNLL 2003 task.

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

5.2 Implementation details

We implement KALM by extending the AWD-LSTM[4] language model in the following ways:

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 words that fall under different entity types, such as *Washington* in *George Washington* and *Washington D.C.*, share the same input embeddings.

Model The model has an embedding layer of 400 dimensions, LSTM cell and hidden states of $1,150$ dimensions, and 3 stacked LSTM layers. We scale the final LSTM's hidden and cell states to 400 dimensions, and share weights between the projection layer $\boldsymbol{W}^p$ and the word embedding layer. Each entity type in the knowledge base is represented by a trainable 100-dimensional embedding vector. When concatenating the weighted

---

[3]The KB can be found in https://github.com/uclanlp/NamedEntityLanguageModel

[4]https://github.com/salesforce/awd-lstm-lm

average of the type embeddings to the input, we expand the input dimension of the first LSTM layer to 500. All trainable parameters are initialized uniformly randomly between -0.1 and 0.1, except for the bias terms in the decoder linear layer, which are initialized to 0.

For regularization, we adopt the techniques in AWD-LSTM, and use an LSTM weight dropout rate of 0, an LSTM first-layers locked dropout rate of 0.3, an LSTM last-layer locked dropout rate of 0.4, an embedding Bernoulli dropout rate of 0.1, and an embedding locked dropout rate of 0.65. Also, we impose L2 penalty on LSTM pre-dropout weights with coefficient 1, and L2 penalty on LSTM dropout weights with coefficient 2, both added to the cross entropy loss.

Optimization We use the same loss penalty, dropout schemes, and averaged SGD (ASGD) as in Merity et al. (2017). The initial ASGD learning rate is 10, weight decay rate is 1.2×10^{-6}, non-monotone trigger for ASGD is set to 5, and gradient clipping happens at 0.25. The models are trained until the validation set performance starts to decrease.

5.3 Language modeling

First, we test how good KALM is as a language model compared to two baselines.

5.3.1 Baselines

- **AWD-LSTM** (Merity et al., 2017) is the state-of-the-art word-level language model as measured on WikiText-2 and Penn Treebank. It uses ASGD optimization, a new dropout scheme and novel penalty terms in the loss function to improve over vanilla LSTM LMs.

- **Named-entity LM** (NE-LM) (Parvez et al., 2018) consists of a *type model* that outputs $P(\tau_{i+1}|\tau_i, \tau_{i-1}, ...)$ and an *entity composite model* that outputs $P(y_{i+1}|\{y_i, \tau_i\}, \{y_{i-1}, \tau_{i-1}\}, ...)$. The *type model* is trained on corpora with entity type labels, whereas the *entity composite model* has an input for words and another input for the corresponding types, and so needs to be trained on both the labeled corpus and the unlabeled version of the same corpus. At inference time, a joint inference heuristic aggregates type model and entity composite model predictions into a word prediction. Since both models require type labels as

input, each generation step of NE-LM requires not only the previously generated words $[y_i, y_{i-1}, ...]$, but also the type labels for these words $[\tau_i, \tau_{i-1}, ...]$.

5.3.2 Results

For language modeling we report word prediction perplexity on the recipe dataset and CoNLL 2003. Perplexity is defined as the following.

$$
\begin{aligned}
PP &= e^{-\sum_{t=1}^{N} \frac{1}{N} \log P(y_t)} \\
&= \sqrt[N]{\prod_{t=1}^{N} \frac{1}{P(y_t)}}
\end{aligned}
\tag{12}
$$

We use publicly available implementations to produce the two baseline results. We also compare the language models in the bidirectional setting, which the reference implementations do not support. In that setting, we transform both models in NE-LM to be bidirectional.

Discussion Table 3 shows that KALM outperforms the two baselines in both unidirectional and bidirectional settings on both datasets. The improvement relative to NE-LM is larger in the unidirectional setting compared to the bidirectional setting. We conjecture that this is because in that setting NE-LM trains a bidirectional NER in a supervised way. The improvement relative to NE-LM is larger on CoNLL 2003 than on the recipe dataset. We believe that the inference heuristic used by NE-LM is tuned specifically to recipes and is less suitable to the CoNLL setting.

We also find that training KALM on more unlabeled data further reduces the perplexity (see Table 4), and study how the quality of the KB affects the perplexity. We discuss both these results in Section 5.4.

5.4 NER

In this section, we evaluate KALM in NER against two supervised baselines.

5.4.1 Baselines

We train two supervised models for NER on the CoNLL 2003 dataset: a **biLSTM** and a **CRF-biLSTM**. We replicate the hyperparameters used by Lample et al. (2016), who demonstrate the state-of-the-art performance on this dataset. We use a word-level model, and 100 dimensional pretrained GloVe embeddings (Pennington et al., 2014) for initialization. We train for 50 epochs, at which point the models converge.

	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

5.4.2 Results

We evaluate the unsupervised KALM model under the following configurations:

- Basic: bidirectional model with aggregated type embeddings fed to the input at the next time step;

- With type priors: using $P(\tau_t|y_t)$ in the two ways described in Section 4.2;

- Extra data: Since KALM is unsupervised, we can train it on extra data. We use the WikiText-2 corpus in addition to the original CoNLL training data.

WikiText-2 is a standard language modeling dataset released with Merity et al. (2016). It contains Wikipedia articles from a wide range of topics. In contrast, the CoNLL 2003 corpus consists of news articles. Table 4 show statistics about the raw / characer level WikiText-2 and the CoNLL 2003 corpora. Despite the domain mismatch between the WikiText and CoNLL corpora, the WikiText coverage of the entity words that exist in the CoNLL dataset is high. Specifically, most of the person, location and organization entity words that appear in CoNLL either have a Wikipedia section, or are mentioned in a Wiki article. Therefore, we expect that the addition of WikiText can guide the unsupervised NER model to learn better entity type regularities. Indeed, the result presented in the rightmost column of Table 4 shows that when adding WikiText-2 to CoNLL 2003, the perplexity for the KALM model for the news text of CoNLL 2003 is decreased: from 4.69 down to **2.29**.

We show NER results in Table 5. The table lists the F1 score for each entity types, as well as the overall F1 score.

Discussion   Even the basic KALM model learns context well – it achieves an overall F1

score of 0.72 for NER. This illustrates that KALM has learned to model entity classes entirely from surrounding context. Adding prior information as to whether a word represents different entity types helps to bring the F1 score to 0.76.

The strength of an unsupervised model is that it can be trained on large corpora. Adding the Wikitext-2 corpus improves the NER score of KALM to 0.84.

To give a sense of how the unsupervised models compare with the supervised model with respect to training data size, we trained biLSTM and CRF-biLSTM on a randomly sampled subset of the training data of successively decreasing sizes. The resulting F1 scores are shown in Figure 2.

Our best model scores 0.86, same as a CRF-biLSTM trained on around 40% of the training data. It is less than 0.03 behind the best supervised CRF-biLSTM. The best KALM model almost always scores higher than biLSTM without the CRF loss.

Lastly, we perform an ablation experiment to gauge how sensitive KALM is to the quality of the knowledge base. Previous studies (Liu et al., 2016; Zhang et al., 2012) have shown that the amount of knowledge retrieved from KBs can impact the performance of NLP models such as relation extraction systems substantially. In this experiment, we deliberately corrupt the entity vocabularies $V_0, ..., V_{K-1}$ by moving a certain percentage of randomly selected entity words from V_i to the general vocabulary V_g. Figure 3 shows language modeling perplexities on the validation set, and NER F1 scores on the test set as a function of the corruption percentage. The language modeling performance stops reacting to KB corruption beyond a certain extent, whereas the NER performance keeps dropping as the number of entities removed from $V_1, V_2, ...$ increases. This result shows the importance of the quality of the KB

entity ratio	unique entity ratio	size ratio	LM perplexity
92.80%	82.56%	2.62	2.29 : 4.69

Table 4: Characterization of WikiText-2 relative to CoNLL 2003 training set. *Entities extracted from WikiText cover* 92.80% *of the entities in CoNLL 2003 overall, and cover* 82.56% *of the unique entities. WikiText's size is* 2.62 *times as large. And adding WikiText to CoNLL training reduces the perplexity from* 4.69 *to* 2.29.

		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.*
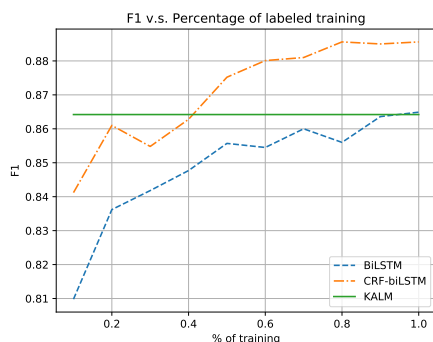

Figure 2: Unsupervised v.s. supervised NER trained on different portions of training data
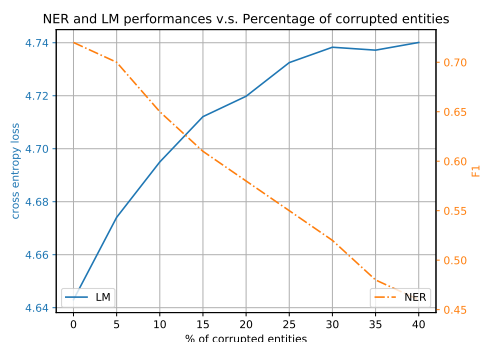

Figure 3: NER and LM performances on different portions of mislabeled entities

6 Conclusion

We propose Knowledge Augmented Language Model (KALM), which extends a traditional RNN LM with information from a Knowledge Base. We show that real-world knowledge can be used successfully for natural language understanding by using a probabilistic extension. The latent type information is trained end-to-end using a predictive objective without any supervision. We show that the latent type information that the model learns can be used for a high-accuracy NER system. We believe that this modeling paradigm opens the door for end-to-end deep learning systems that can be enhanced with latent modeling capabilities and trained in a predictive manner end-to-end. In ways this is similar to the attention mechanism in machine translation where an alignment mechanism is added and trained latently against the overall translation perplexity objective. As with our NER tags, machine translation alignments are empirically observed to be of high quality.

In future work, we look to model other types of world knowledge beyond named entities using predictive learning and training on large corpora of text without additional information, and to make KALM more robust against corrupted entities.

to KALM.

References

Sungjin Ahn, Heeyoul Choi, Tanel Pärnamaa, and Yoshua Bengio. 2016. A neural knowledge language model. *arXiv preprint arXiv:1608.00318*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*, pages 2787–2795.

Michael Collins and Yoram Singer. 1999. Unsupervised models for named entity classification. In *1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Oren Etzioni, Michael Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S Weld, and Alexander Yates. 2005. Unsupervised named-entity extraction from the web: An experimental study. *Artificial intelligence*, 165(1):91–134.

Yihong Gu, Jun Yan, Hao Zhu, Zhiyuan Liu, Ruobing Xie, Maosong Sun, Fen Lin, and Leyu Lin. 2018. Language modeling with sparse product of sememe experts. *arXiv preprint arXiv:1810.12387*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*.

Angli Liu, Stephen Soderland, Jonathan Bragg, Christopher H Lin, Xiao Ling, and Daniel S Weld. 2016. Effective crowd annotation for relation extraction. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 897–906.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2017. Regularizing and optimizing lstm language models. *arXiv preprint arXiv:1708.02182*.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*.

Robert Munro and Christopher D Manning. 2012. Accurate unsupervised joint named-entity extraction from unaligned parallel text. In *Proceedings of the 4th Named Entity Workshop*, pages 21–29. Association for Computational Linguistics.

David Nadeau, Peter D Turney, and Stan Matwin. 2006. Unsupervised named-entity recognition: Generating gazetteers and resolving ambiguity. In *Conference of the Canadian Society for Computational Studies of Intelligence*, pages 266–277. Springer.

Md Rizwan Parvez, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2018. Building language models for text with named entities. *arXiv preprint arXiv:1805.04836*.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Matthew E Peters, Waleed Ammar, Chandra Bhagavatula, and Russell Power. 2017. Semi-supervised sequence tagging with bidirectional language models. *arXiv preprint arXiv:1705.00108*.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Erik F Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 142–147. Association for Computational Linguistics.

Ji Xin, Hao Zhu, Xu Han, Zhiyuan Liu, and Maosong Sun. 2018. Put it back: Entity typing with language model enhancement. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 993–998.

Bishan Yang and Tom Mitchell. 2017. Leveraging knowledge bases in lstms for improving machine reading. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1436–1446.

Zichao Yang, Phil Blunsom, Chris Dyer, and Wang Ling. 2016. Reference-aware language models. *arXiv preprint arXiv:1611.01628*.

Ce Zhang, Feng Niu, Christopher Ré, and Jude Shavlik. 2012. Big data versus the crowd: Looking for relationships in all the right places. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 825–834. Association for Computational Linguistics.