Learning to Organize Knowledge with N-Gram Machines

Fan Yang **William W. Cohen** **Jiazhong Nie** **Ni Lao**
{fanyang1,wcohen}@cs.cmu.edu {niejiazhong,nlao}@google.com
Carnegie Mellon University, Pittsburgh, PA Google Inc., Mountain View, CA

1 Introduction

Although there is a great deal of recent research on extracting structured knowledge from text [5, 16] and answering questions from structured knowledge stores [4, 8, 10], much less progress has been made on the problem of unifying these approaches in an end-to-end model, and removing the bottleneck of having human experts to design the schema and annotate examples. This paper considers the problem of treating the schema and the content of a structured storage as discrete hidden variables in an QA system, and induce these structures automatically from only weak supervisions such as QA pairs. The structured storage we consider is simply a set of "n-grams", which can represent a wide range of semantics, and can be indexed for efficient computations at scale. We present an end-to-end trainable system which combines an text auto-encoding component for encoding the knowledge, and a memory enhanced sequence to sequence component for answering questions from the storage.

1.1 Question Answering As A Testbed for Text Understanding

There is a wide range of text understanding tasks such as machine translation, summarization, dialogue, and question answering (QA). While it is difficult to quantitatively evaluate performances for text generation tasks, it is straightforward to evaluate responses to QA tasks [27]. The reminder of this section analyzes why existing technologies are insufficient to meet its challenges, and outlines our proposed solution. Before that we define a question answering task more formally as producing the answer a given sentences $\mathbf{s} = (s_1, \ldots, s_{|\mathbf{s}|})$ and question q. Each sentence s_i is represented as a sequence of words, i.e. $s_i = (w_1, \ldots, w_n)$. And the question q is also represented as a sequence of words, i.e. $q = (w_1, \ldots, w_m)$. We focus on extractive question answering, where the answer a is always a word in one of the sentences.

1.2 Text Understanding: Current Practice

In recent years, several large-scale knowledge bases (KBs) have been constructed, such as YAGO [23], Freebase [2], NELL [16], Google Knowledge Graph [21], Microsoft Satori [19], and others. However, all of them suffer from a completeness problem [5] – namely to convert *arbitrary* text to graph structures that can be used to answer questions. There are two core issues in this difficulty:

The schema (or representation) problem Traditional text understanding approaches need some fixed and finite target schema [20]. However, there is infinite granularities of semantics that can be associated with a text expression, but are really hard to predefine manually at scale. It is prohibitively expensive to have experts to clearly define all these differences as a universal schema, and annotate enough utterances from which they are expressed. However, given a particular context, the grounding of these expressions might not even need all these granular supervision. A desirable solution should induce schema automatically from the corpus such that the groundings can help downstream tasks.

The end-to-end training (or inference) problem The state-of-the-art approaches break down QA solutions into independent components, such as schema definition and manually annotating examples [16, 2], relation extraction [15], entity resolution [22], and semantic parsing of questions [1, 10]. However, these components are supposed to work together and depend on each other. Existing systems [5, 1] leverage human annotation and supervised training for each component separately, which create consistency issues, and is not directly optimizing the performance of the QA task. A desirable solution should use less human annotations of intermediate steps, and rely on end-to-end training to directly optimize the QA quality.

1.3 Text Understanding: Deep Neural Nets

More recently there has been a lot of progress in applying deep neural networks (DNNs) to text understanding [13, 26, 18, 6, 14]. The key ingredient to these solutions is embedding text expressions into a latent *continuous* space. It removes the need of manually deciding schema and greatly simplifies the design of QA systems, enabling end-to-end training that directly optimizes the QA quality. However, a key issue that prevent these models to be applied to many applications is **scalability**. After receiving a question, all text in a corpus need to be analyzed by the model. Therefore it leads to at least $O(n)$ complexity, where n is the text size. Approaches which rely on a search subroutine (e.g., DrQA [3]) lose the benefit of end-to-end training, and are limited by the quality of the retrieval stage, which itself is as hard as the QA problem.

1.4 N-Gram Machines: A Scalable End-to-End Approach

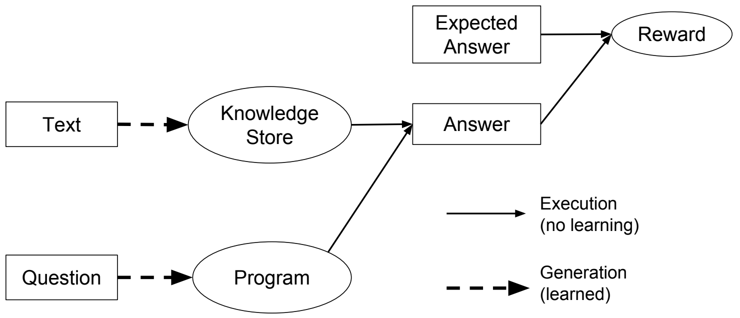

Figure 1: End-to-end QA system with a symbolic knowledge store.

We propose to solve the scalability issue of DNN text understanding models by learning to represent the meaning of text as a *symbolic* knowledge storage. Because the storage can be indexed before being used for question answering, the inference step can be done very efficiently with complexity that is independent of the original text size. More specifically the structured storage we consider is simply a set of "n-grams", which we show can represent complex semantics presented in bAbI tasks [27] and can be indexed for efficient computations at scale. Each n-gram consists of a sequence of tokens, and each token can be a word, or any predefined special symbol. The whole system (Figure 1) consists of learnable components which convert text into symbolic knowledge storage and questions into programs (details in Section 2.1). The whole system is trained end-to-end with no human annotation other than the expected answers to a set of question-text pairs.

1.5 Related Work

The auto-encoding part of our model (Figure 2) is similar to the text summarization model proposed by Miao and Blunsom [12]. Our approach differs from Miao and Blunsom [12] in dealing with the large search space when generating hidden sequences. In particular, we 1) use a less restricted hidden space (through a CopyNet [7]) by allowing both copied tokens and generated tokens; 2) stabilize the decoder by forcing it (through experience replay) to train from randomly generated hidden sequences; and 3) use the log-likelihood of the pre-trained decoder to guide the training of the encoder.

The question answering part of our model (Figure 2) is similar to the Neural Symbolic Machine (NSM) [10], which is a memory enhanced sequence-to-sequence model that translates questions into programs in λ-calculus [11]. The programs, when executed on a knowledge graph, can produce answers to the questions. Our work extends NSM by removing the assumption of a given knowledge bases or schema, and instead learns to generate storage by end-to-end training to answer questions.

2 N-Gram Machines

In this section we first describe the N-Gram Machine (NGM) model structure, which contains three sequence to sequence modules, and an executor that executes programs against knowledge storage. Then we describe how this model can be trained end-to-end with reinforcement learning.

2.1 Model Structure

Knowledge storage Given a sequence of sentences $\mathbf{s} = \{s_1, \ldots, s_T\}$, our knowledge storage is a collection of knowledge tuples $\mathbf{\Gamma} = \{\Gamma_1, \ldots, \Gamma_T\}$. The tuple Γ_i has two parts: a time stamp i and a sequence of symbols $(\gamma_1, \ldots, \gamma_N)$, where each symbol γ_j is either a word from the sentence s_i or a special symbol from a pre-defined set. The time stamps in the tuple are useful for reasoning about time and are just sentence indices for the bAbI task. The knowledge storage is probabilistic – each tuple Γ_i also has a probability, and the probability of the knowledge storage is the product of the probabilities of all its tuples (Equation 1). An example of a knowledge storage is shown in Table 1.

Table 1: Example of probabilistic knowledge storage. Each sentence may be converted to a distribution over multiple tuples, but only the one with the highest probability is shown here.

Sentences	Knowledge tuples		
	Time stamp	Symbols	Probability
Mary went to the kitchen.	1	`mary to kitchen`	0.9
Mary picked up the milk.	2	`mary the milk`	0.4
John went to the bedroom.	3	`john to bedroom`	0.7
Mary journeyed to the garden.	4	`mary to garden`	0.8

Programs Programs in the N-Gram Machine are similar to those introduced in Neural Symbolic Machine [10], except that our functions operate on n-grams (i.e. knowledge tuples)[1] instead of Freebase triples. In general, functions specify how symbols can be retrieved from a knowledge storage. Specifically, a function in NGM use a prefix (or suffix) to retrieve symbols from tuples – i.e. if a prefix "matches" a tuple, the immediate next symbol in the tuple is returned. For the bAbI tasks, we define four functions, which are illustrated in Table 2: Function `Hop` and `HopFR` return symbols from all the matched tuples while function `Argmax` and `ArgmaxFR` return symbols from the latest matches (i.e. the tuples with the latest the time stamp in all the matches)

Table 2: Functions in N-Gram Machines. The knowledge storage on which the programs can execute is $\mathbf{\Gamma}$, and a knowledge tuple Γ_i is represented as $(i, (\gamma_1, \ldots, \gamma_N))$. "`FR`" means *from right*.

Name	Inputs	Return
`Hop`	$v_1 \ldots v_L$	$\{\gamma_{L+1} \mid \text{if } (\gamma_1 \ldots \gamma_L) == (v_1, \ldots, v_L), \forall \Gamma \in \mathbf{\Gamma}\}$
`HopFR`	$v_1 \ldots v_L$	$\{\gamma_{N-L} \mid \text{if } (\gamma_{N-L+1} \ldots \gamma_N) == (v_L, \ldots, v_1), \forall \Gamma \in \mathbf{\Gamma}\}$
`Argmax`	$v_1 \ldots v_L$	$\text{argmax}_i\{(\gamma_{L+1}, i) \mid \text{if } (\gamma_1 \ldots \gamma_L) == (v_1, \ldots, v_L), \forall \Gamma_i \in \mathbf{\Gamma}\}$
`ArgmaxFR`	$v_1 \ldots v_L$	$\text{argmax}_i\{(\gamma_{N-L}, i) \mid \text{if } (\gamma_{N-L+1} \ldots \gamma_N) == (v_L, \ldots, v_1), \forall \Gamma_i \in \mathbf{\Gamma}\}$

Seq2Seq components Our N-Gram Machine uses three sequence-to-sequence [25] neural network models to define probability distributions over knowledge tuples and programs. As illustrated in Figure 2, these models are:

- A *knowledge encoder* that converts sentences to knowledge tuples and defines a distribution $P(\Gamma_i|s_i, s_{i-1}; \theta_{\text{enc}})$. It is conditioned on the previous sentence s_{i-1} to handle cross sentence linguistic phenomenons such as co-references[2]. The probability of a knowledge storage

[1]We will use "n-gram" and "knowledge tuple" interchangeably.

[2]Ideally it should condition on the partially constructed $\mathbf{\Gamma}$ at time $t-1$, but that makes it hard to do batch training of the DNN models, and is beyond the scope of this work.

$\mathbf{\Gamma} = \{\Gamma_1 \dots \Gamma_n\}$ is defined as the product of its knowledge tuples' probabilities:

$$P(\mathbf{\Gamma}|\mathbf{s};\theta_{\text{enc}}) = \Pi_{\Gamma_i \in \mathbf{\Gamma}} P(\Gamma_i|s_i, s_{i-1};\theta_{\text{enc}}) \tag{1}$$

- A *knowledge decoder* that converts tuples back to sentences and defines a distribution $P(s_i|\Gamma_i, s_{i-1};\theta_{\text{dec}})$. It enables auto-encoding training, which is crucial for finding good knowledge representations.
- A *programmer* that converts questions to programs and defines a distribution $P(C|q,\mathbf{\Gamma};\theta_{\text{prog}})$. It is conditioned on the knowledge storage $\mathbf{\Gamma}$ for code assistance [10] – before generating each token the programmer can query $\mathbf{\Gamma}$ about the valid next tokens given a tuple prefix, and therefore avoid writing invalid programs.

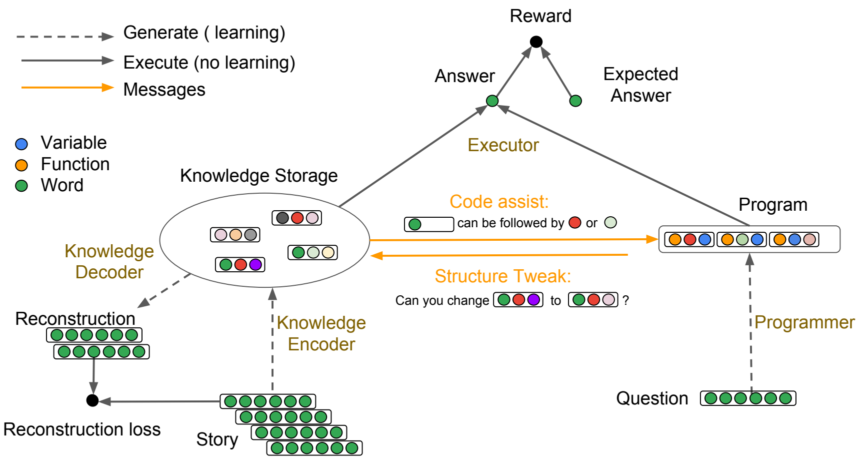

Figure 2: N-Gram Machine. The model contains two discrete hidden structures, the knowledge storage and the program, which are generated from the story and the question respectively. The executor executes programs against the knowledge storage to produce answers. The three learnable components, knowledge encoder, knowledge decoder, and programmer, are trained to maximize the answer accuracy as well as minimize the reconstruction loss of the story. Code assist and structure tweak help the knowledge encoder and programmer to communicate and cooperate with each other.

2.2 Inference and Optimization

Given an example $(\mathbf{s}, q, a)$ from our training set, we would like to maximize the expected reward

$$O^{QA}(\theta_{\text{enc}}, \theta_{\text{prog}}) = \sum_{\mathbf{\Gamma}} \sum_{C} P(\mathbf{\Gamma}|\mathbf{s};\theta_{\text{enc}}) P(C|q,\mathbf{\Gamma};\theta_{\text{prog}}) R(\mathbf{\Gamma}, C, a), \tag{2}$$

where the reward function $R(\cdot)$ returns 1 if executing C on $\mathbf{\Gamma}$ produces a, and 0 otherwise. We approximate the expectation with beam searches – the summation over all programs is replaced by summing over programs found by beam search based on the programmer model $P(C|q,\mathbf{\Gamma};\theta_{\text{prog}})$. For the summation over knowledge storages $\mathbf{\Gamma}$, we first run beam search for each sentence based on the knowledge encoder model $P(\Gamma_i|s_i, s_{i-1};\theta_{\text{enc}})$, and then sample a set of knowledge storages by independently sampling from the knowledge tuples of each sentence. However, since the beam searches explore exponentially large spaces, it is very challenging to optimize O^{QA}. We introduce two special techniques to iteratively reduce and improve the search space:

Stabilized Auto-Encoding (AE) We add an auto-encoding objective to our framework, similar to the text summarization model proposed by Miao and Blunsom [12]. The training of this objective can be done by variational inference [9, 17]:

$$O^{\text{VAE}}(\theta_{\text{enc}}, \theta_{\text{dec}}) = \mathbb{E}_{p(z|x;\theta_{\text{enc}})}[\log p(x|z;\theta_{\text{dec}}) + \log p(z) - \log p(z|x;\theta_{\text{enc}})], \tag{3}$$

where x is text, and z is the hidden discrete structure. However, it suffers from instability due to the strong coupling between encoder and decoder – the training of the decoder θ_{dec} relies solely on a distribution parameterized by the encoder θ_{enc}, which changes throughout the course of training. To improve the auto-encoding training stability, we propose to augment the decoder training with a more stable objective – predict the data x back from noisy partial observations of x, which are independent of θ_{enc}. More specifically, for NGM we force the knowledge decoder to decode from a fixed set of hidden sequences $z \in \mathbf{Z}^N(x)$, which includes all knowledge tuples of length N that consist of only words from the text x:

$$O^{\text{AE}}(\theta_{\text{enc}}, \theta_{\text{dec}}) = \mathbb{E}_{p(z|x;\theta_{\text{enc}})}[\log p(x|z;\theta_{\text{dec}})] + \sum_{z \in \mathbf{Z}^N(x)} \log p(x|z;\theta_{\text{dec}}), \tag{4}$$

The knowledge decoder θ_{dec} converts knowledge tuples back to sentences and the reconstruction log-likelihoods approximate how informative the tuples are, which can be used as reward for the knowledge encoder. We also drop the KL divergence (last two terms in Equation 3) between language model $p(z)$ and the encoder, since the z's are produced for NGM computations instead of human reading, and do not need to be in fluent natural language.

Structure Tweaking (ST) Even with AE training, the knowledge encoder is encoding tuples without the understanding of how they are going to be used, and may encode them inconsistently across sentences. At the later QA stage, such inconsistency can lead to no reward when the programmer tries to reason with multiple knowledge tuples. To retrospectively correct the inconsistency in tuples, we apply *structure tweak*, a procedure which is similar to code assist [3], but works in an opposite direction. While code assist uses the knowledge storage to inform the programmer, structure tweak adjusts the knowledge encoder to cooperate with an uninformed programmer. Together they allow the decisions in one part of the model to be influence by the decisions from other parts – similar in spirit to the Markov chain Monte Carlo (MCMC) methods.

Because the knowledge storage and the program are non-differentiable discrete structures, we optimize our objective by a coordinate ascent approach – optimizing the three components in alternation with REINFORCE [28].

3 Results

bAbI The bAbI dataset contains twenty tasks in total. We consider the subset of them that are extractive question answering tasks. Each task is learned separately. For all tasks, we set the knowledge tuple length to three. In Table 3, we report results on the test sets. NGM outperforms MemN2N [24] on all tasks listed. The results show that auto-encoding is essential to bootstrapping the learning. Without auto-encoding, the expected rewards are near zero; but auto-encoding alone is not sufficient to achieve high rewards (See Section 2.2). Since multiple discrete latent structures (i.e. knowledge tuples and programs) need to agree with each other over the choice of their representations for QA to succeed, the search becomes combinatorially hard. Structure tweaking is an effective way to refine the search space – improving the performance of more than half of the tasks.

Table 3: Test accuracy on bAbI tasks with auto-encoding (AE) and structure tweak (ST)

	Task 1	Task 2	Task 11	Task 15	Task 16
MemN2N	1.000	0.830	0.840	1.000	0.440
QA	0.007	0.027	0.000	0.000	0.098
QA + AE	0.709	0.551	**1.000**	0.246	**1.000**
QA + AE + ST	**1.000**	**0.853**	**1.000**	**1.000**	**1.000**

Table 4 lists sampled knowledge storages learned with different objectives and procedures. Knowledge storages learned with auto-encoding are much more informative compared to the ones without. After structure tweaking, the knowledge tuples converge to use more consistent symbols – e.g., using `went` instead of `back` or `travelled`. Our experiment results show the tweaking procedure can help NGM to deal with various linguistic phenomenons such as singular/plural ("cats" vs "cat") and synonyms ("grabbed" vs "got").

Life-long bAbI To demonstrate the scalability advantage of the N-Gram Machine, we conduct experiments on question answering data where the number of sentences may increase up to 10 million.

Table 4: Sampled knowledge storage with question answering (QA) objective, auto-encoding (AE) objective, and structure tweak (ST) procedure. Using AE alone produces similar tuples to QA+AE. The differences between the second and the third column are underlined.

QA	QA + AE	QA + AE + ST
went went went	daniel went office	daniel went office
mary mary mary	mary back garden	mary went garden
john john john	john back kitchen	john went kitchen
mary mary mary	mary grabbed football	mary got football
there there there	sandra got apple	sandra got apple
cats cats cats	cats afraid wolves	cat afraid wolves
mice mice mice	mice afraid wolves	mouse afraid wolves
is is cat	gertrude is cat	gertrude is cat

More specifically we generated longer bAbI stories using the open-source script from Facebook[3]. We measure the answering time and answer quality of MemN2N [24][4] and NGM at different scales. The answering time is measured by the amount of time used to produce an answer when a question is given. For MemN2N, this is the neural network inference time. For NGM, because the knowledge storage can be built and indexed in advance, this is the programmer decoding time.

Figure 3 compares MemN2N and NGM. In terms of answering time, MemN2N scales poorly – the inference time increases linearly as the story length increases. While for NGM, the answering time is not affected by story length. The crossover of the two lines is when the story length is around 1000, which is due to the difference in neural network architectures – NGM uses recurrent networks while MemN2N uses feed-forward networks. To compare the answer quality at scale, we apply MemN2N and NGM to solve three life-long bAbI tasks (Task 1, 2, and 11). For each life-long task, MemN2N is run for 10 trials and the test accuracy of the trial with the best validation accuracy is used. For NGM, we use the same models trained on regular bAbI tasks. We compute the average and standard deviation of test accuracy from these three tasks. MemN2N performance is competitive with NGM when story length is no greater than 400, but decreases drastically when story length further increases. On the other hand, NGM answering quality is the same for all story lengths. These scalability advantages of NGM are due to its "machine" nature – the symbolic knowledge storage can be computed and indexed in advance, and the program execution is robust on stories of various lengths.

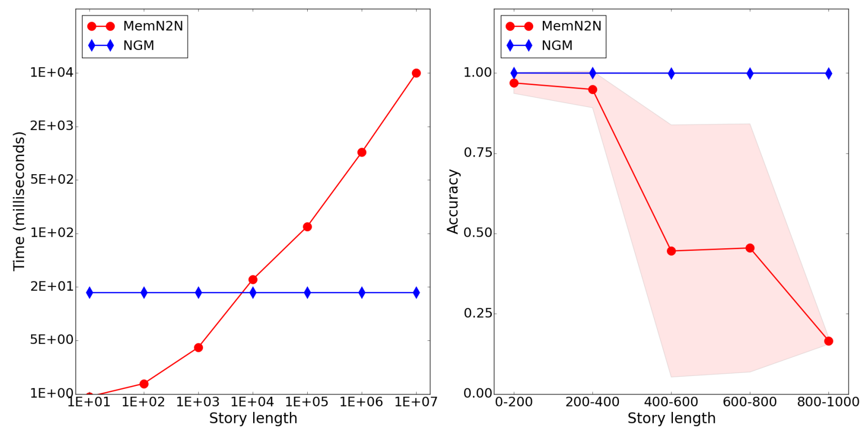

Figure 3: Scalability comparison of MemN2N and NGM. Left: Answering time. Right: Answer quality. Story length is the number of sentences in each QA pair.

[3]https://github.com/facebook/bAbI-tasks

[4]https://github.com/domluna/memn2n

References

[1] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on freebase from question-answer pairs. In *EMNLP*, volume 2, page 6, 2013.

[2] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250. ACM, 2008.

[3] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading Wikipedia to answer open-domain questions. In *Association for Computational Linguistics (ACL)*, 2017.

[4] Li Dong and Mirella Lapata. Language to logical form with neural attention. In *Association for Computational Linguistics (ACL)*, 2016.

[5] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 601–610, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2956-9.

[6] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 2016.

[7] Jiatao Gu, Zhengdong Lu, Hang Li, and Victor OK Li. Incorporating copying mechanism in sequence-to-sequence learning. *arXiv preprint arXiv:1603.06393*, 2016.

[8] Robin Jia and Percy Liang. Data recombination for neural semantic parsing. In *Association for Computational Linguistics (ACL)*, 2016.

[9] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *ICLR*, 2014.

[10] Chen Liang, Jonathan Berant, Quoc Le, Kenneth D. Forbus, and Ni Lao. Neural symbolic machines: Learning semantic parsers on freebase with weak supervision. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 23–33, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[11] P. Liang, M. I. Jordan, and D. Klein. Learning dependency-based compositional semantics. In *Association for Computational Linguistics (ACL)*, pages 590–599, 2011.

[12] Yishu Miao and Phil Blunsom. Language as a latent variable: Discrete generative models for sentence compression. *the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2016.

[13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013.

[14] Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. Key-value memory networks for directly reading documents. *arXiv preprint arXiv:1606.03126*, 2016.

[15] Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2 - Volume 2*, ACL '09, pages 1003–1011, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics. ISBN 978-1-932432-46-6.

[16] Tom M. Mitchell, William W. Cohen, Estevam R. Hruschka Jr., Partha Pratim Talukdar, Justin Betteridge, Andrew Carlson, Bhavana Dalvi Mishra, Matthew Gardner, Bryan Kisiel, Jayant Krishnamurthy, Ni Lao, Kathryn Mazaitis, Thahir Mohamed, Ndapandula Nakashole, Emmanouil Antonios Platanios, Alan Ritter, Mehdi Samadi, Burr Settles, Richard C. Wang, Derry Tanti Wijaya, Abhinav Gupta, Xinlei Chen, Abulhair Saparov, Malcolm Greaves, and Joel Welling. Never-ending learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 2302–2310, 2015.

[17] Andriy Mnih and Karol Gregor. Neural variational inference and learning in belief networks. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages II–1791–II–1799. JMLR.org, 2014.

[18] Arvind Neelakantan, Quoc V Le, and Ilya Sutskever. Neural programmer: Inducing latent programs with gradient descent. *arXiv preprint arXiv:1511.04834*, 2015.

[19] Richard Qian. Understand your world with bing. *Bing Blog*, 2013.

[20] Sebastian Riedel, Limin Yao, Andrew McCallum, and Benjamin M. Marlin. Relation extraction with matrix factorization and universal schemas. In *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 9-14, 2013, Westin Peachtree Plaza Hotel, Atlanta, Georgia, USA*, pages 74–84, 2013.

[21] Amit Singhal. Introducing the knowledge graph: things, not strings. *Official Google Blog*, 2012.

[22] Parag Singla and Pedro Domingos. Entity resolution with markov logic. In *Data Mining, 2006. ICDM'06. Sixth International Conference on*, pages 572–582. IEEE, 2006.

[23] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: A core of semantic knowledge. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 697–706, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-654-7.

[24] Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. In *Advances in neural information processing systems*, pages 2440–2448, 2015.

[25] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[26] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.

[27] Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M Rush, Bart van Merriënboer, Armand Joulin, and Tomas Mikolov. Towards ai-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*, 2015.

[28] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.