Commonsense Knowledge Base Completion

Xiang Li[*‡] **Aynaz Taheri**[†] **Lifu Tu**[‡] **Kevin Gimpel**[‡]

[*]University of Chicago, Chicago, IL, 60637, USA
[†]University of Illinois at Chicago, Chicago, IL, 60607, USA
[‡]Toyota Technological Institute at Chicago, Chicago, IL, 60637, USA

lix1@uchicago.edu, ataher2@uic.edu, {lifu,kgimpel}@ttic.edu

Abstract

We enrich a curated resource of commonsense knowledge by formulating the problem as one of knowledge base completion (KBC). Most work in KBC focuses on knowledge bases like Freebase that relate entities drawn from a fixed set. However, the tuples in ConceptNet (Speer and Havasi, 2012) define relations between an unbounded set of *phrases*. We develop neural network models for scoring tuples on arbitrary phrases and evaluate them by their ability to distinguish true held-out tuples from false ones. We find strong performance from a bilinear model using a simple additive architecture to model phrases. We manually evaluate our trained model's ability to assign quality scores to novel tuples, finding that it can propose tuples at the same quality level as medium-confidence tuples from ConceptNet.

1 Introduction

Many ambiguities in natural language processing (NLP) can be resolved by using knowledge of various forms. Our focus is on the type of knowledge that is often referred to as "commonsense" or "background" knowledge. This knowledge is rarely expressed explicitly in textual corpora (Gordon and Van Durme, 2013). Some researchers have developed techniques for inferring this knowledge from patterns in raw text (Gordon, 2014; Angeli and Manning, 2014), while others have developed curated resources of commonsense knowledge via manual annotation (Lenat and Guha, 1989; Speer and Havasi, 2012) or games with a purpose (von Ahn et al., 2006).

Curated resources typically have high precision but suffer from a lack of coverage. For certain resources, researchers have developed methods to automatically increase coverage by inferring missing entries. These methods are commonly categorized under the heading of knowledge base completion (KBC). KBC is widely-studied for knowledge bases like Freebase (Bollacker et al., 2008) which contain large sets of entities and relations among them (Mintz et al., 2009; Nickel et al., 2011; Riedel et al., 2013; West et al., 2014), including recent work using neural networks (Socher et al., 2013; Yang et al., 2014).

relation	right term	conf.
MOTIVATEDBYGOAL	relax	3.3
USEDFOR	relaxation	2.6
MOTIVATEDBYGOAL	your muscle be sore	2.3
HASPREREQUISITE	go to spa	2.0
CAUSES	get pruny skin	1.6
HASPREREQUISITE	change into swim suit	1.6

Table 1: ConceptNet tuples with left term "soak in hotspring"; final column is confidence score.

We improve the coverage of commonsense resources by formulating the problem as one of knowledge base completion. We focus on a particular curated commonsense resource called ConceptNet (Speer and Havasi, 2012). ConceptNet contains tuples consisting of a left term, a relation, and a right term. The relations come from a fixed set. While terms in Freebase tuples are entities, ConceptNet terms can be arbitrary *phrases*. Some examples are shown in Table 1. An NLP application may wish to query ConceptNet for information about soaking in a hotspring, but may use different words from those contained in the ConceptNet tuples. Our goal is to do on-the-fly knowledge base completion so that queries can be answered robustly without requiring the precise linguistic forms contained in ConceptNet.

To do this, we develop neural network models to embed terms and provide scores to arbi-

Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, pages 1445–1455,
Berlin, Germany, August 7-12, 2016.

trary tuples. We train them on ConceptNet tuples and evaluate them by their ability to distinguish true and false held-out tuples. We consider several functional architectures, comparing two composition functions for embedding terms and two functions for converting term embeddings into tuple scores. We find that all architectures are able to outperform several baselines and reach similar performance on classifying held-out tuples.

We also experiment with several training objectives for KBC, finding that a simple cross entropy objective with randomly-generated negative examples performs best while also being fastest. We manually evaluate our trained model's ability to assign quality scores to novel tuples, finding that it can propose tuples at the same quality level as medium-confidence tuples from ConceptNet. We release all of our resources, including our ConceptNet KBC task data, large sets of randomly-generated tuples scored with our model, training code, and pretrained models with code for calculating the confidence of novel tuples.[1]

2 Related Work

Our methods are similar to past work on KBC (Mintz et al., 2009; Nickel et al., 2011; Lao et al., 2011; Nickel et al., 2012; Riedel et al., 2013; Gardner et al., 2014; West et al., 2014), particularly methods based on distributed representations and neural networks (Socher et al., 2013; Bordes et al., 2013; Bordes et al., 2014a; Bordes et al., 2014b; Yang et al., 2014; Neelakantan et al., 2015; Gu et al., 2015; Toutanova et al., 2015). Most prior work predicts new relational links between terms drawn from a *fixed* set. In a notable exception, Neelakantan and Chang (2015) add new entities to KBs using external resources along with properties of the KB itself. Relatedly, Yao et al. (2013) induce an unbounded set of entity categories and associate them with entities in KBs.

Several researchers have developed techniques for discovering commonsense knowledge from text (Gordon et al., 2010; Gordon and Schubert, 2012; Gordon, 2014; Angeli and Manning, 2014). Open information extraction systems like REVERB (Fader et al., 2011) and NELL (Carlson et al., 2010) find tuples with arbitrary terms and relations from raw text. In contrast, we start with a set of commonsense facts to use for training, though our methods could be applied to the output of these or other extraction systems.

Our goals are similar to those of the AnalogySpace method (Speer et al., 2008), which uses matrix factorization to improve coverage of ConceptNet. However, AnalogySpace can only return a confidence score for a pair of terms drawn from the training set. Our models can assign scores to tuples that contain novel terms (as long as they consist of words in our vocabulary).

Though we use ConceptNet, similar techniques can be applied to other curated resources like WordNet (Miller, 1995) and FrameNet (Baker et al., 1998). For WordNet, tuples can contain lexical entries that are linked via synset relations (e.g., "hypernym"). WordNet contains many multiword entries (e.g., "cold sweat"), which can be modeled compositionally by our term models; alternatively, entire glosses could be used as terms. To expand frame relationships in FrameNet, tuples can draw relations from the frame relation types (e.g., "is causative of") and terms can be frame lexical units or their definitions.

Several researchers have used commonsense knowledge to improve language technologies, including sentiment analysis (Cambria et al., 2012; Agarwal et al., 2015), semantic similarity (Caro et al., 2015), and speech recognition (Lieberman et al., 2005). Our hope is that our models can enable many other NLP applications to benefit from commonsense knowledge.

Our work is most similar to that of Angeli and Manning (2013). They also developed methods to assess the plausibility of new facts based on a training set of facts, considering commonsense data from ConceptNet in one of their settings. Like us, they can handle an unbounded set of terms by using (simple) composition functions for novel terms, which is rare among work in KBC. One key difference is that their best method requires iterating over the KB at test time, which can be computationally expensive with large KBs. Our models do not require iterating over the training set. We compare to several baselines inspired by their work, and we additionally evaluate our model's ability to score novel tuples derived from both ConceptNet and Wikipedia.

3 Models

Our goal is to represent commonsense knowledge such that it can be used for NLP tasks. We as-

[1] Available at `http://ttic.uchicago.edu/~kgimpel/commonsense.html`.

sume this knowledge is given in the form of tuples $\langle t_1, R, t_2 \rangle$, where t_1 is the **left term**, t_2 is the **right term**, and R is a (directed) **relation** that exists between the terms. Examples are shown in Table 1.[2]

Given a set of tuples, our goal is to develop a parametric model that can provide a confidence score for new, unseen tuples. That is, we want to design and train models that define a function $\text{score}(t_1, R, t_2)$ that provides a quality score for an arbitrary tuple $\langle t_1, R, t_2 \rangle$. These models will be evaluated by their ability to distinguish true held-out tuples from false ones.

We describe two model families for scoring tuples. We assume that we have embeddings for words and define models that use these word embeddings to score tuples. So our models are limited to tuples in which terms consist of words in the word embedding vocabulary, though future work could consider character-based architectures for open-vocabulary modeling (Huang et al., 2013; Ling et al., 2015).

3.1 Bilinear Models

We first consider bilinear models, since they have been found useful for KBC in past work (Nickel et al., 2011; Jenatton et al., 2012; García-Durán et al., 2014; Yang et al., 2014). A bilinear model has the following form for a tuple $\langle t_1, R, t_2 \rangle$:

$$v_1^\top M_R \, v_2$$

where $v_1 \in \mathbb{R}^r$ is the (column) vector representing t_1, $v_2 \in \mathbb{R}^r$ is the vector for t_2, and $M_R \in \mathbb{R}^{r \times r}$ is the parameter matrix for relation R.

To convert terms t_1 and t_2 into term vectors v_1 and v_2, we consider two possibilities: word averaging and a bidirectional long short-term memory (LSTM) recurrent neural network (Hochreiter and Schmidhuber, 1997). This provides us with two models: **Bilinear AVG** and **Bilinear LSTM**.

One downside of this architecture is that as the length of the term vectors grows, the size of the relation matrices grows quadratically. This can slow down training while requiring more data to learn the large numbers of parameters in the matrices. To address this, we include an additional nonlinear transformation of each term:

$$u_i = a(W^{(B)} v_i + b^{(B)})$$

where a is a nonlinear activation function (tuned among ReLU, $\tanh$, and logistic sigmoid) and where we have introduced additional parameters $W^{(B)}$ and $b^{(B)}$. This gives us the following model:

$$\text{score}_{\text{bilinear}}(t_1, R, t_2) = u_1^\top M_R \, u_2$$

When using the LSTM, we tune the decision about how to produce the final term vectors to pass to the bilinear model, including possibly using the final vectors from each direction and the output of max or average pooling. We use the same LSTM parameters for each term.

3.2 Deep Neural Network Models

Our second family of models is based on deep neural networks (DNNs). While bilinear models have been shown to work well for KBC, their functional form makes restrictions about how terms can interact. DNNs make no such restrictions.

As above, we define two models, one based on using word averaging for the term model (**DNN AVG**) and one based on LSTMs (**DNN LSTM**). For the DNN AVG model, we obtain the term vectors v_1 and v_2 by averaging word vectors in the respective terms. We then concatenate v_1, v_2, and a relation vector v_R to form the input of the DNN, denoted v_{in}. The DNN uses a single hidden layer:

$$\begin{aligned} u &= a(W^{(D1)} v_{in} + b^{(D1)}) \\ \text{score}_{\text{DNN}}(t_1, R, t_2) &= W^{(D2)} u + b^{(D2)} \end{aligned} \quad (1)$$

where a is again a (tuned) nonlinear activation function. The size of the hidden vector u is tuned, but the output dimensionality (the numbers of rows in $W^{(D2)}$ and $b^{(D2)}$) is fixed to 1. We do not use a nonlinear activation for the final layer since our goal is to output a scalar score.

For the DNN LSTM model, we first create a single vector for the two terms using an LSTM. That is, we concatenate t_1, a delimiter token, and t_2 to create a single word sequence. We use a bidirectional LSTM to convert this word sequence to a vector, again possibly using pooling (the decision is tuned; details below). We concatenate the output of this bidirectional LSTM with the relation vector v_r to create the DNN input vector v_{in}, then use Eq. 1 to obtain a score. We found this to work better than separately using an LSTM on each term. We can not try this for the Bilinear LSTM model since its functional form separates the two term vectors.

[2] These examples are from the Open Mind Common Sense (OMCS) part of ConceptNet version 5 (Speer and Havasi, 2012). In our experiments below, we only use OMCS tuples.

The relation vectors v_R are learned in addition to the DNN parameters $W^{(D1)}$, $W^{(D2)}$, $b^{(D1)}$, $b^{(D2)}$, and the LSTM parameters (in the case of the DNN LSTM model). Also, word embedding parameters are updated in all settings.

4 Training

Given a tuple training set T, we train our models using two different loss functions: hinge loss and a binary cross entropy function. Both rely on ways of generating negative examples (Section 4.3). Both also use regularization (Section 4.4).

4.1 Hinge Loss

Given a training tuple $\tau = \langle t_1, R, t_2 \rangle$, the hinge loss seeks to make the score of τ larger than the score of negative examples by a margin of at least γ. This corresponds to minimizing the following loss, summed over all examples $\tau \in T$:

$$\begin{aligned}&\text{loss}_{\text{hinge}}(\tau) = \\ &\quad \max\{0, \gamma - \text{score}(\tau) + \text{score}(\tau_{neg(t_1)})\} \\ &\quad + \max\{0, \gamma - \text{score}(\tau) + \text{score}(\tau_{neg(R)})\} \\ &\quad + \max\{0, \gamma - \text{score}(\tau) + \text{score}(\tau_{neg(t_2)})\}\end{aligned}$$

where $\tau_{neg(t_1)}$ is the negative example obtained by replacing t_1 in τ with some other t_1, and $\tau_{neg(R)}$ and $\tau_{neg(t_2)}$ are defined analogously for the relation and right term. We describe how we generate these negative examples in Section 4.3 below.

4.2 Binary Cross Entropy

Though we only have true tuples in our training set, we can create a binary classification problem by assigning a label of 1 to training tuples and a label of 0 to negative examples. Then we can minimize cross entropy (CE) as is common when using neural networks for classification. To generate negative examples, we consider the methods described in Section 4.3 below. We also need to convert our models' scores into probabilities, which we do by using a logistic sigmoid σ on score. We denote the label as ℓ, where the label is 1 if the tuple is from the training set and 0 if it is a negative example. Then the loss is defined:

$$\begin{aligned}&\text{loss}_{\text{CE}}(\tau, \ell) = \\ &-\ell \log \sigma(\text{score}(\tau)) - (1-\ell) \log(1 - \sigma(\text{score}(\tau)))\end{aligned}$$

When using this loss, we generate three negative examples for each positive example (one for swapping each component of the tuple, as in the hinge loss). For a mini-batch of size β, there are β positive examples and 3β negative examples used for training. The loss is summed over these 4β examples yielded by each mini-batch.

4.3 Negative Examples

For the loss functions above, we need ways of automatically generating negative examples. For efficiency, we consider using the current mini-batch only, as our models are trained using optimization on mini-batches. We consider the following three strategies to construct negative examples. Each strategy constructs three negative examples for each positive example τ: one by replacing t_1, one by replacing R, and one by replacing t_2.

Random sampling. We create the three negative examples for τ by replacing each component with its counterpart in a randomly-chosen tuple in the same mini-batch.

Max sampling. We create the three negative examples for τ by replacing each component with its counterpart in some other tuple in the mini-batch, choosing the substitution to maximize the score of the resulting negative example. For example, when swapping out t_1 in $\tau = \langle t_1, R, t_2 \rangle$, we choose the substitution t_1' as follows:

$$t_1' = \underset{t:\langle t, R', t_2' \rangle \in \mu \setminus \tau}{\text{argmax}} \text{score}(t, R, t_2)$$

where μ is the current mini-batch of tuples. We perform the analogous procedure for R and t_2.

Mix sampling. This is a mixture of the above, using random sampling 50% of the time and max sampling the remaining 50% of the time.

4.4 Regularization

We use L_2 regularization. For the DNN models, we add the penalty term $\lambda \|\theta\|^2$ to the losses, where λ is the regularization coefficient and θ contains all other parameters. However, for the bilinear models we regularize the relation matrices M_R toward the identity matrix instead of all zeroes, adding the following to the loss:

$$\lambda_1 \|\theta\|^2 + \lambda_2 \sum_R \|M_R - I_r\|_2^2$$

where I_r is the $r \times r$ identity matrix, the summation is performed over all relations R, and θ represents all other parameters.

5 Experimental Setup

We now evaluate our tuple models. We measure whether our models can distinguish true and false tuples by training a model on a large set of tuples and testing on a held-out set.

5.1 Task Design

The tuples are obtained from the Open Mind Common Sense (OMCS) entries in the ConceptNet 5 dataset (Speer and Havasi, 2012). They are sorted by a confidence score. The most confident 1200 tuples were reserved for creating our test set (TEST). The next most confident 600 tuples (i.e., those numbered 1201–1800) were used to build a development set (DEV1) and the next most confident 600 (those numbered 1801–2400) were used to build a second development set (DEV2).

For each set S ($S \in \{\text{DEV1}, \text{DEV2}, \text{TEST}\}$), for each tuple $\tau \in S$, we created a negative example and added it to S. So each set doubled in size. To create a negative example from $\tau \in S$, we randomly swapped one of the components of τ with another tuple $\tau' \in S$. One third of the time we swapped t_1 in τ for t_1 in τ', one third of the time we swapped their R's, and the remaining third of the time we swapped their t_2's. Thus, distinguishing positive and negative examples in this task is similar to the objectives optimized during training.

Each of DEV1 and DEV2 has 1200 tuples (600 positive examples and 600 negative examples), while TEST has 2400 tuples (1200 positive and 1200 negative). For training data, we selected 100,000 tuples from the remaining tuples (numbered 2401 and beyond).

The task is to separate the true and false tuples in our test set. That is, the labels are 1 for true tuples and 0 for false tuples. Given a model for scoring tuples, we select a threshold by maximizing accuracy on DEV1 and report accuracies on DEV2. This is akin to learning the bias feature weight (using DEV1) of a linear classifier that uses our model's score as its only feature. We tuned several choices—including word embeddings, hyperparameter values, and training objectives—on DEV2 and report final performance on TEST. One annotator (a native English speaker) attempted the same classification task on a sample of 100 tuples from DEV2 and achieved an accuracy of 95%. We release these datasets to the community so that others can work on this same task.

5.2 Word Embeddings

Our tuple models rely on initial word embeddings. To help our models better capture the commonsense knowledge in ConceptNet, we generated word embedding training data using the OMCS sentences underlying our training tuples (we excluded the top 2400 tuples which were used for creating DEV1, DEV2, and TEST). We created training data by merging the information in the tuples and their OMCS sentences. Our goal was to combine the grammatical context of the OMCS sentences with the words in the actual terms, so as to ensure that we learn embeddings for the words in the terms. We also insert the relations into the OMCS sentences so that we can learn embeddings for the relations themselves.

We describe the procedure by example and also release our generated data for ease of replication. The tuple ⟨soak in a hotspring, CAUSES, get pruny skin⟩ was automatically extracted/normalized (by the ConceptNet developers) from the OMCS sentence "The effect of [soaking in a hotspring] is [getting pruny skin]" where brackets surround terms. We replace the bracketed portions with their corresponding terms and insert the relation between them: "The effect of soak in a hotspring CAUSES get pruny skin". We do this for all training tuples.[3]

We used the `word2vec` (Mikolov et al., 2013) toolkit to train skip-gram word embeddings on this data. We trained for 20 iterations, using a dimensionality of 200 and a window size of 5. We refer to these as "CN-trained" embeddings for the remainder of this paper. Similar approaches have been used to learn embeddings for particular downstream tasks, e.g., dependency parsing (Bansal et al., 2014). We use our CN-trained embeddings within baseline methods and also provide the initial word embeddings of our models. For all of our models, we update the initial word embeddings during learning.

In the baseline methods described below, we compare our CN-trained embeddings to pretrained word embeddings. We use the GloVe (Pennington et al., 2014) embeddings trained on 840 billion tokens of Common Crawl web text and the PARAGRAM-SimLex embeddings of Wieting et al. (2015), which were tuned to have strong performance on the SimLex-999 task (Hill et al., 2015).

[3]For reversed relations, indicated by an asterisk in the OMCS sentences, we swap t_1 and t_2 in the tuple.

5.3 Baselines

We consider three baselines inspired by those of Angeli and Manning (2013):

- **Similar Fact Count (Count)**: For each tuple $\tau = \langle t_1, R, t_2 \rangle$ in the evaluation set, we count the number of similar tuples in the training set. A training tuple $\tau' = \langle t_1', R', t_2' \rangle$ is considered "similar" to τ if $R = R'$, one of the terms matches exactly, and the other term has the same head word. That is, $(R = R') \wedge (t_1 = t_1') \wedge (head(t_2) = head(t_2'))$, or $(R = R') \wedge (t_2 = t_2') \wedge (head(t_1) = head(t_1'))$. The head word for a term was obtained by running the Stanford Parser (Klein and Manning, 2003) on the term. This baseline does not use word embeddings.
- **Argument Similarity (ArgSim)**: This baseline computes the cosine similarity of the vectors for t_1 and t_2, ignoring the relation. Vectors for t_1 and t_2 are obtained by word averaging.
- **Max Similarity (MaxSim)**: For tuple τ in an evaluation set, this baseline outputs the maximum similarity between τ and any tuple in the training set. The similarity is computed by concatenating the vectors for t_1, R, and t_2, then computing cosine similarity. As in ArgSim, we obtain vectors for terms by averaging their words. We only consider R when using our CN-trained embeddings since they contain embeddings for the relations. When using GloVe and PARAGRAM embeddings for this baseline, we simply use the two term vectors (still constructed via averaging the words in each term).

We chose these baselines because they can all handle unbounded term sets but differ in their other requirements. ArgSim and MaxSim use word embeddings while Count does not. Count and MaxSim require iterating over the training set during inference while ArgSim does not.

For each baseline, we tuned a threshold on DEV1 to maximize classification accuracy then tested on DEV2 and TEST.

5.4 Training and Tuning

We used AdaGrad (Duchi et al., 2011) for optimization, training for 20 epochs through the training tuples. We separately tuned hyperparameters for each model and training objective. We tuned the following hyperparameters: the relation matrix size r for the bilinear models (also the length of the transformed term vectors, denoted u_1 and u_2 above), the activation a, the hidden layer size g for the DNN models, the relation vector length d for the DNN models, the LSTM hidden vector size h for models with LSTMs, the mini-batch size β, the regularization parameters λ, λ_1, and λ_2, and the AdaGrad learning rate α.

All tuning used early stopping: periodically during training, we used the current model to find the optimal threshold on DEV1 and evaluated on DEV2. Due to computational limitations, we were unable to perform thorough grid searches for all hyperparameters. We combined limited grid searches with greedy hyperparameter tuning based on regions of values that were the most promising.

For the Bilinear LSTM and DNN LSTM, we did hyperparameter tuning by training on the full training set of 100,000 tuples for 20 epochs, computing DEV2 accuracy once per epoch. For the averaging models, we tuned by training on a subset of 1000 tuples with $\beta = 200$ for 20 epochs; the averaging models showed more stable results and did not require the full training set for tuning. Below are the tuned hyperparameter values:

- **Bilinear AVG**: for CE: $r = 150$, $a = \tanh$, $\beta = 200$, $\alpha = 0.01$, $\lambda_1 = \lambda_2 = 0.001$. Hinge loss: same values as above except $\alpha = 0.005$.
- **Bilinear LSTM**: for CE: $r = 50$, $a =$ ReLU, $h = 200$, $\beta = 800$, $\alpha = 0.02$, and $\lambda_1 = \lambda_2 = 0.00001$. To obtain vectors from the term bidirectional LSTMs, we used max pooling. For hinge loss: $r = 50$, $a = \tanh$, $h = 200$, $\beta = 400$, $\alpha = 0.007$, $\lambda_1 = 0.00001$, and $\lambda_2 = 0.01$. To obtain vectors from the term bidirectional LSTMs, we used the concatenation of average pooling and final hidden vectors in each direction. For each sampling method and loss function, α was tuned by grid search with the others fixed to the above values.
- **DNN AVG**: for both losses: $a =$ ReLU, $d = 200$, $g = 1000$, $\beta = 600$, $\alpha = 0.01$, $\lambda = 0.001$.
- **DNN LSTM**: for both losses: $a =$ ReLU, $d = 200$, bidirectional LSTM hidden layer size $h = 200$, hidden layer dimension $g = 800$, $\beta = 400$, $\alpha = 0.005$, and $\lambda = 0.00005$. To get vectors from the term LSTMs, we used max pooling.

6 Results

Word Embedding Comparison. We first evaluate the quality of our word embeddings trained on the ConceptNet training tuples. Table 2 compares

	GloVe	PARAGRAM	CN-trained
ArgSim	68	69	**73**
MaxSim	73	70	**82**

Table 2: Accuracies (%) on DEV2 of two baselines using three different sets of word embeddings. Our ConceptNet-trained embeddings outperform GloVe and PARAGRAM embeddings.

	DNN AVG		DNN LSTM	
	CE	hinge	CE	hinge
random	124	230	710	783
mix	20755	21045	25928	26380
max	39338	41867	49583	49427

Table 4: Loss function runtime comparison (seconds per epoch) of the DNN models.

accuracies on DEV2 for the two baselines that use embeddings: ArgSim and MaxSim. We find that pretrained GloVe and PARAGRAM embeddings perform comparably, but both are outperformed by our ConceptNet-trained embeddings. We use the latter for the remaining experiments in this paper.

Training Comparison. Table 3 shows the results of our models with the two loss functions and three sampling strategies. We find that the binary cross entropy loss with random sampling performs best across models. We note that our conclusion differs from some prior work that found max or mix sampling to be better than random (Wieting et al., 2016). We suspect that this difference may stem from characteristics of the ConceptNet training data. It may often be the case that the max-scoring negative example in the mini-batch is actually a true fact, due to the generic nature of the facts expressed.

Table 4 shows a runtime comparison of the losses and sampling strategies.[4] We find random sampling to be orders of magnitude faster than the others while also performing the best.

Final Results. Our final results are shown in Table 5. We show the DEV2 and TEST accuracies for our baselines and for the best configuration (tuned on DEV2) for each model. All models outperform all baselines handily. Our models perform similarly, with the Bilinear models and the DNN AVG model all exceeding 90% on both DEV2 and TEST.

We note that the AVG models performed strongly compared to those that used LSTMs for

[4]These experiments were performed using 2 threads on a 3.40-GHz Intel Core i7-3770 CPU with 8 cores.

	DEV2	TEST
Count	75.4	79.0
ArgSim	72.9	74.2
MaxSim	81.9	83.5
Bilinear AVG	90.3	91.7
Bilinear LSTM	90.8	90.7
DNN AVG	91.3	92.0
DNN LSTM	88.1	89.2
Bilinear AVG + data	91.8	92.5
human	∼95.0	—

Table 5: Accuracies (%) of baselines and final model configurations on DEV2 and TEST. "+ data" uses enlarged training set of size 300,000, and then doubles this training set by including tuples with conjugated forms; see text for details. Human performance on DEV2 was estimated from a sample of size 100.

modeling terms. We suggest two reasons for this. The first is that most terms are short, with an average term length of 2.3 words in our training tuples. An LSTM may not be needed to capture long-distance properties. The second reason may be due to hyperparameter tuning. Recall that we used a greedy search for optimal hyperparameter values; we found that models with LSTMs take more time per epoch, more epochs to converge, and exhibit more hyperparameter sensitivity compared to models based on averaging. This may have contributed to inferior hyperparameter values for the LSTM models.

We also trained the Bilinear AVG model on a larger training set (row labeled "Bilinear AVG + data"). We note that the ConceptNet tuples typically contain unconjugated forms; we sought to use both conjugated and unconjugated words. We began with a larger training set of 300,000 tuples from ConceptNet, then augmented them to include conjugated word forms as in the following example. For the tuple ⟨soak in a hotspring, CAUSES, get pruny skin⟩ obtained from the OMCS sentence "The effect of [soaking in a hotspring] is [getting pruny skin]", we generated an additional tuple ⟨soaking in a hotspring, CAUSES, getting pruny skin⟩. We thus created twice as many training tuples. The results with this larger training set improved from 91.7 to 92.5 on the test set. We release this final model to the research community.

We note that the strongest baseline, MaxSim, is a nonparametric model that requires iterating over all training tuples to provide a score to new tuples. This is a serious bottleneck for use in NLP applications that may need to issue large numbers of

	Bilinear AVG		Bilinear LSTM		DNN AVG		DNN LSTM	
	CE	hinge	CE	hinge	CE	hinge	CE	hinge
random	**90**	84	**91**	83	**91**	87	**88**	57
mix	**90**	83	90	87	90	78	82	63
max	86	75	65	66	61	52	56	52

Table 3: Accuracies (%) on DEV2 of models trained with two loss functions (cross entropy (CE) and hinge) and three sampling strategies (random, mix, and max). The best accuracy for each model is shown in bold. Cross entropy with random sampling is best across models and is also fastest (see Table 4).

queries. Our models are parametric models that can compress a large training set into a fixed number of parameters. This makes them extremely fast for answering queries, particularly the AVG models, enabling use in downstream NLP applications.

7 Generating and Scoring Novel Tuples

We now measure our model's ability to score novel tuples generated automatically from ConceptNet and Wikipedia. We first describe simple procedures to generate candidate tuples from these two datasets. We then score the tuples using our MaxSim baseline and the trained Bilinear AVG model.[5] We evaluate the highly-scoring tuples using a small-scale manual evaluation.

The DNN AVG and Bilinear AVG models reached the highest TEST accuracies in our evaluation, though in preliminary experiments we found that the Bilinear AVG model appeared to perform better when scoring the novel tuples described below. We suspect this is because the DNN function class has more flexibility than the bilinear one. When scoring novel tuples, many of which may be highly noisy, it appears that the constrained structure of the Bilinear AVG model makes it more robust to the noise.

7.1 Generating Tuples From ConceptNet

In order to get new tuples, we automatically modify existing ConceptNet tuples. We take an existing tuple and randomly change one of the three fields (t_1, t_2, or R), ensuring that the result is not a tuple existing in ConceptNet. We then score these tuples using MaxSim and the Bilinear AVG model and analyze the results.

7.2 Generating Tuples from Wikipedia

We also propose a simple method to extract candidate tuples from raw text. We first run the Stanford part-of-speech (POS) tagger (Toutanova et al., 2003) on the terms in our ConceptNet training tuples. We enumerate the 50 most frequent term pair tag sequences for each relation. We do limited manual filtering of the frequent tag sequences, namely removing the sequences "DT NN NN" and "DT JJ NN" for the ISA relation. We do this in order to reduce noise in the extracted tuples. To focus on finding nontrivial tuples, for each relation we retain the top 15 POS tag sequences in which t_1 or t_2 has at least two words.

We then run the tagger on sentences from English Wikipedia. We extract word sequence pairs corresponding to the relation POS tag sequence pairs, requiring that there be a gap of at least one word between the two terms. We then remove word sequence pairs in which one term is solely one of the following words: be, the, always, there, has, due, however. We also remove tuples containing words that are not in the vocabulary of our ConceptNet-trained embeddings. We require that one term does not include the other term. We create tuples consisting of the two terms and all possible relations that occur with the POS sequences of those two terms. Finally, we remove tuples that exactly match our ConceptNet training tuples.

We use our trained Bilinear AVG model to score these tuples. We extract term pairs that occur within the same sentence because we hope that these will have higher precision than if we were to pair together arbitrary pairs. Some example tuples for $t_1 =$ *bus* are shown in Table 6.

t_1, R, t_2	score
bus, ISA, *public transportation*	0.95
bus, ISA, *public transit*	0.90
bus, ISA, *mass transit*	0.79
bus, ATLOCATION, *downtown area*	0.98
bus, ATLOCATION, *subway station*	0.98
bus, ATLOCATION, *city center*	0.94
bus, CAPABLEOF, *low cost*	0.72
bus, CAPABLEOF, *local service*	0.65
bus, CAPABLEOF, *train service*	0.63

Table 6: Top Wikipedia tuples for 3 relations with $t_1 =$ *bus*, scored by Bilinear AVG model.

[5] For the results in this section, we used the Bilinear AVG model that achieved 91.7 on TEST rather than the one augmented with additional data.

7.3 Manual Analysis of Novel Tuples

To evaluate our models on newly generated tuples, we rank them using different models and manually score the high-ranking tuples for quality. We first randomly sampled 3000 tuples from each set of novel tuples. We do so due to the time requirements of the MaxSim baseline, which requires iterating through the entire training set for each candidate tuple. We score these sampled tuples using MaxSim and the Bilinear AVG model and rank them by their scores. The top 100 tuples under each ranking were given to an annotator who is a native English speaker. The annotator assigned a quality score to each tuple, using the same 0-4 annotation scheme as Speer et al. (2010): 0 ("Doesn't make sense"), 1 ("Not true"), 2 ("Opinion/Don't know"), 3 ("Sometimes true"), and 4 ("Generally true"). We report the average quality score across each set of 100 tuples.

The results are shown in Table 7. To calibrate the scores, we also gave two samples of ConceptNet (CN) tuples to the annotator: a sample of 100 high-confidence tuples (first row) and a sample of 100 medium-confidence tuples (second row). We find the high-confidence tuples to be of high quality, recording an average of 3.68, though the medium-confidence tuples drop to 3.14.

The next two rows show the quality scores of the MaxSim baseline and the Bilinear AVG model. The latter outperforms the baseline and matches the quality of the medium-confidence ConceptNet tuples. Since our novel tuples are not contained in ConceptNet, this result suggests that our model can be used to add medium-confidence tuples to ConceptNet.

The novel Wikipedia tuples (top 100 tuples ranked by Bilinear AVG model) had a lower quality score (2.78), but this is to be expected due to the difference in domain. Since Wikipedia contains a wide variety of text, we found the novel tuples to be noisier than those from ConceptNet. Still, we are encouraged that on average the tuples are judged to be close to "sometimes true."

7.4 Text Analysis with Commonsense Tuples

We note that our method of tuple extraction and scoring could be used as an aid in applications that require sentence understanding. Two example sentences are shown in Table 8, along with the top tuples extracted and scored using our method. The tuples capture general knowledge about phrases contained in the sentence, rather than necessarily indicating what the sentence means. This procedure could provide relevant commonsense knowledge for a downstream application that seeks to understand the sentence. We leave further investigation of this idea to future work.

tuples	quality
high-confidence CN tuples	3.68
medium-confidence CN tuples	3.14
novel CN tuples, ranked by MaxSim	2.74
novel CN tuples, ranked by Bilinear AVG	3.20
novel Wiki tuples, ranked by Bilinear AVG	2.78

Table 7: Average quality scores from manual evaluation of novel tuples. Each row corresponds to a different set of tuples. See text for details.

After nine years of primary school, students can go to the high school or to an educational institution.	
t_1, R, t_2	score
school, HASPROPERTY, *educational*	0.89
school, ISA, *educational institution*	0.80
school, ISA, *institution*	0.78
school, HASPROPERTY, *high*	0.77
high school, ISA, *institution*	0.71
On March 14, 1964, Ruby was convicted of murder with malice, for which he received a death sentence.	
t_1, R, t_2	score
murder, CAUSES, *death**	1.00
murder, CAUSES, *death sentence*	0.86
murder, HASSUBEVENT, *death*	0.84
murder, CAPABLEOF, *death*	0.51

Table 8: Top ranked tuples extracted from two example sentences and scored by Bilinear AVG model. $*$ = contained in ConceptNet.

8 Conclusion

We proposed methods to augment curated commonsense resources using techniques from knowledge base completion. By scoring novel tuples, we showed how we can increase the applicability of the knowledge contained in ConceptNet. In future work, we will explore how to use our model to improve downstream NLP tasks, and consider applying our methods to other knowledge bases. We have released all of our resources—code, data, and trained models—to the research community.[6]

Acknowledgments

We thank the anonymous reviewers, John Wieting, and Luke Zettlemoyer.

[6]Available at `http://ttic.uchicago.edu/~kgimpel/commonsense.html`.

References

Basant Agarwal, Namita Mittal, Pooja Bansal, and Sonal Garg. 2015. Sentiment analysis using common-sense and context information. *Comp. Int. and Neurosc.*

Gabor Angeli and Christopher Manning. 2013. Philosophers are mortal: Inferring the truth of unseen facts. In *Proc. of CoNLL*.

Gabor Angeli and D. Christopher Manning. 2014. NaturalLI: Natural logic inference for common sense reasoning. In *Proc. of EMNLP*.

Collin F Baker, Charles J Fillmore, and John B Lowe. 1998. The Berkeley FrameNet project. In *Proc. of COLING*.

Mohit Bansal, Kevin Gimpel, and Karen Livescu. 2014. Tailoring continuous word representations for dependency parsing. In *Proc. of ACL*.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proc. of the ACM SIGMOD International Conference on Management of Data*.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *Advances in NIPS*.

Antoine Bordes, Sumit Chopra, and Jason Weston. 2014a. Question answering with subgraph embeddings. In *Proc. of EMNLP*.

Antoine Bordes, Xavier Glorot, Jason Weston, and Yoshua Bengio. 2014b. A semantic matching energy function for learning with multi-relational data. *Machine Learning*, 94(2).

Erik Cambria, Daniel Olsher, and Kenneth Kwok. 2012. Sentic activation: A two-level affective common sense reasoning framework. In *Proc. of AAAI*.

Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R Hruschka Jr, and Tom M Mitchell. 2010. Toward an architecture for never-ending language learning. In *Proc. of AAAI*.

Luigi Di Caro, Alice Ruggeri, Loredana Cupi, and Guido Boella. 2015. Common-sense knowledge for natural language understanding: Experiments in unsupervised and supervised settings. In *Proc. of AI*IA*.

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*.

Anthony Fader, Stephen Soderland, and Oren Etzioni. 2011. Identifying relations for open information extraction. In *Proc. of EMNLP*.

Alberto García-Durán, Antoine Bordes, and Nicolas Usunier. 2014. Effective blending of two and three-way interactions for modeling multi-relational data. In *Machine Learning and Knowledge Discovery in Databases*.

Matt Gardner, Partha Pratim Talukdar, Jayant Krishnamurthy, and Tom Mitchell. 2014. Incorporating vector space similarity in random walk inference over knowledge bases. In *Proc. of EMNLP*.

Jonathan Gordon and Lenhart K Schubert. 2012. Using textual patterns to learn expected event frequencies. In *Proc. of Joint Workshop on Automatic Knowledge Base Construction and Web-scale Knowledge Extraction*.

Jonathan Gordon and Benjamin Van Durme. 2013. Reporting bias and knowledge acquisition. In *Proc. of Workshop on Automated Knowledge Base Construction*.

Jonathan Gordon, Benjamin Van Durme, and Lenhart K Schubert. 2010. Learning from the web: Extracting general world knowledge from noisy text. In *Collaboratively-Built Knowledge Sources and AI*.

Jonathan Gordon. 2014. *Inferential Commonsense Knowledge from Text*. Ph.D. thesis, University of Rochester.

Kelvin Gu, John Miller, and Percy Liang. 2015. Traversing knowledge graphs in vector space. In *Proc. of EMNLP*.

Felix Hill, Roi Reichart, and Anna Korhonen. 2015. SimLex-999: Evaluating semantic models with (genuine) similarity estimation. *Computational Linguistics*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*.

Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *Proc. of CIKM*.

Rodolphe Jenatton, Nicolas L. Roux, Antoine Bordes, and Guillaume R Obozinski. 2012. A latent factor model for highly multi-relational data. In *Advances in NIPS*.

Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *Proc. of ACL*.

Ni Lao, Tom Mitchell, and William W Cohen. 2011. Random walk inference and learning in a large scale knowledge base. In *Proc. of EMNLP*.

Douglas B Lenat and Ramanathan V Guha. 1989. *Building large knowledge-based systems: representation and inference in the Cyc project*. Addison-Wesley Longman Publishing Co., Inc.

Henry Lieberman, Alexander Faaborg, Waseem Daher, and José Espinosa. 2005. How to wreck a nice beach you sing calm incense. In *Proc. of IUI*.

Wang Ling, Chris Dyer, Alan W Black, Isabel Trancoso, Ramon Fermandez, Silvio Amir, Luis Marujo, and Tiago Luis. 2015. Finding function in form: Compositional character models for open vocabulary word representation. In *Proc. of EMNLP*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in NIPS*.

George A Miller. 1995. WordNet: a lexical database for English. *Communications of the ACM*, 38(11).

Mike Mintz, Steven Bills, Rion Snow, and Daniel Jurafsky. 2009. Distant supervision for relation extraction without labeled data. In *Proc of ACL*.

Arvind Neelakantan and Ming-Wei Chang. 2015. Inferring missing entity type instances for knowledge base completion: New dataset and methods. In *Proc. of NAACL*.

Arvind Neelakantan, Benjamin Roth, and Andrew McCallum. 2015. Compositional vector space models for knowledge base completion. In *Proc. of ACL*.

Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. 2011. A three-way model for collective learning on multi-relational data. In *Proc. of ICML*.

Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. 2012. Factorizing YAGO: Scalable machine learning for linked data. In *Proc. of WWW*.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global vectors for word representation. In *Proc. of EMNLP*.

Sebastian Riedel, Limin Yao, Andrew McCallum, and M. Benjamin Marlin. 2013. Relation extraction with matrix factorization and universal schemas. In *Proc. of NAACL-HLT*.

Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. 2013. Reasoning with neural tensor networks for knowledge base completion. In *Advances in NIPS*.

Robert Speer and Catherine Havasi. 2012. Representing general relational knowledge in ConceptNet 5. In *Proc. of LREC*.

Robert Speer, Catherine Havasi, and Henry Lieberman. 2008. AnalogySpace: Reducing the dimensionality of common sense knowledge. In *Proc. of AAAI*.

Robert Speer, Catherine Havasi, and Harshit Surana. 2010. Using verbosity: Common sense data from games with a purpose. In *Proc. of FLAIRS*.

Kristina Toutanova, Dan Klein, Christopher D Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proc. of NAACL-HLT*.

Kristina Toutanova, Danqi Chen, Patrick Pantel, Hoifung Poon, Pallavi Choudhury, and Michael Gamon. 2015. Representing text for joint embedding of text and knowledge bases. In *Proc. of EMNLP*.

Luis von Ahn, Mihir Kedia, and Manuel Blum. 2006. Verbosity: a game for collecting common-sense facts. In *Proc. of CHI*.

Robert West, Evgeniy Gabrilovich, Kevin Murphy, Shaohua Sun, Rahul Gupta, and Dekang Lin. 2014. Knowledge base completion via search-based question answering. In *Proc. of WWW*.

John Wieting, Mohit Bansal, Kevin Gimpel, Karen Livescu, and Dan Roth. 2015. From paraphrase database to compositional paraphrase model and back. *Transactions of the ACL*.

John Wieting, Mohit Bansal, Kevin Gimpel, and Karen Livescu. 2016. Towards universal paraphrastic sentence embeddings. In *Proc. of ICLR*.

Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2014. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575*.

Limin Yao, Sebastian Riedel, and Andrew McCallum. 2013. Universal schema for entity type prediction. In *Proc. of Workshop on Automated Knowledge Base Construction*.