Achieving Coverage and Confidence for Commonsense Knowledge Base Inference

Anonymous authors

Abstract

In the world of knowledge inference, word embeddings and human-curated knowledge bases (KB) are some of the most popular, but flawed sources. Word embeddings (e.g. FastText) can provide a wide coverage of concepts, however their reliance on words appearing in similar contexts makes the knowledge relatively weak evidence for inference. On the other hand, human-curated knowledge bases (e.g. WordNet and ConceptNet) provide stronger explicit evidence of relationships between concepts, however they have relatively lower coverage of concepts. Retrofitting is used to address these shortcomings by moving word vectors closer or further in their space to reflect their relationships in a knowledge base. However, retrofitting only works on concepts that are present in that knowledge base. This paper introduces two systems to address these issues: RetroGAN and Deep Relationship Discovery. RetroGAN uses the popular Cycle Generative Adversarial Network architecture to learn a retrofitting mapping. RetroGAN thus makes it possible to retrofit out-of-knowledge-base concepts in a manner similar to how some natural language systems handle out-of-vocabulary entries. We demonstrate the knowledge generalization capabilities of RetroGAN in another system called Deep Relationship Discovery which performs analogy-like inference for out-of-knowledge assertions given a pair of retrofitted embeddings. We evaluate the system in a commonsense knowledge base completion task and demonstrate that it is capable of inferring unseen assertions. By training Deep Relationship Discovery on RetroGAN produced embeddings and ConceptNet assertions, we are able to get a basic commonsense understanding of a set of concepts through generation of a commonsense-based knowledge base.

1. Introduction

Our understanding of the world is dependent on our ability to grasp the relationships between concepts. Without this, we struggle to interpret meaning and develop a commonsense understanding. If we were given a list of concepts without any context such as a hammer, wood, and nails, how could we get a basic understanding of them? Humans inherit innate ("commonsense") knowledge that expands as we develop. Commonsense knowledge is regarded as a broad, but shallow source of knowledge—while we know about many concepts,
we may not know the detailed intricacies of them. ConceptNet [Speer et al., 2017] is a commonsense knowledge base that tries to capture this. By using commonsense we could possibly know that hammers are used for building, that some houses are made of wood, and typically nails are found in construction sites. Through commonsense we could get a simple understanding that we are talking about building something. However, we were only able to get this understanding by building a knowledge base filled with the concepts and their relations that lie in commonsense. If we wanted machines to get this kind of understanding, we would need them to be able to infer the relationships between concepts in the case that there was no explicit collected assertion. Going one step further, we would also want to be able to infer relationships between concepts that are new.

To determine the relationship between two concepts, we can take multiple approaches. Here we define concepts as a superset of entities because we include actions and things. One approach is to utilize word embeddings: one can find the concept’s embeddings and determine the similarity between them. As the embeddings are trained on a large corpora, this approach offers far-reaching coverage of concepts. The advantage to this approach is that since the embeddings are trained on large corpora, the coverage of concepts is far reaching. The disadvantage is, however, that one cannot determine a concepts relationship that attributes to the similarity of the vectors [Camacho-Collados et al., 2019][Alsuhaibani et al., 2018]. Alternatively, we can query a knowledge base (KB), such as ConceptNet, to see whether (and if so, how) two concepts are connected. A curated KB has the advantage that it has quality content that is explicit, and one could look up or trace a path through a graph representation to determine how concepts are related. The disadvantage here is that coverage of assertions is sparse, and the coverage of concepts in the KB is also considerably smaller than the one in word embeddings trained on large corpora. This brings us to the question: Can we get the best of both to determine relationships amongst concepts?

Retrofitting word embeddings with KBs [Faruqui et al., 2014][Mrksic et al., 2017] is one such middle ground. It takes the vector space of the original word embedding, and finds a transformation that moves word vectors closer together (or further apart) to a position that would better reflect their relationships in a more explicit KB. However, the retrofitting process can only work on concepts that are actually present in that KB. To remedy this problem, we develop a system called RetroGAN. We build upon the approach presented as AuxGAN [Ponti et al., 2018] by extending it to have a CycleGAN [Zhu et al., 2017] architecture rather than a regular Generative Adversarial Network (GAN [Goodfellow et al., 2014]) architecture. We chose the CycleGAN architecture because through it we take into account the cyclic conversion process and through that we preserve the information in both domains rather than favoring one or the other. In addition, by doing this, we bound more the outputs for unseen data in both our domains. The use of this adversarial technique permits us to learn a mapping to be able to retrofit any input word embedding (from the same type of word embedding family). Intuitively, the adversarial technique expands the KB that is used to retrofit because it tries to make similar embeddings have similar retrofitted counterparts, even if these are not in the KB. A concrete example of this could be with the concepts “dog” and “doggo”. “Doggo” is internet slang for dog. In a KB it may be the case that we do not see “doggo” since it is slang. However, if we do have the concept “dog” in our KB, and through word embeddings we can generate a representation
for “doggo” and “dog”, then through RetroGAN we would be able to retrofit “doggo” with the knowledge (or some extent of it) that we have for “dog”. This can be seen in Table 1.

<table>
<thead>
<tr>
<th>Word</th>
<th>Distributional Neighbor</th>
<th>Retrofitted Neighbor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dog</td>
<td>dog, dogs, puppy, pup , canine, pet, doggie, beagle, dachshund, cat</td>
<td>dog, beagle, pooch, dachshund, puppy, mutt, poodle, Rottweiler, canine, labrador</td>
</tr>
<tr>
<td>Doggo</td>
<td>pooch, doggies, bae, chihuahua, rad, pug, kitty, dane, furbabies, ♢</td>
<td>doggies, pooch, dachshund, four-legged, Yorkie, corgi, whippet, amigos, Weimaraner, Dog</td>
</tr>
</tbody>
</table>

Table 1: Results of the 10 most similar embeddings for “dog” and “doggo” for FastText. The distributional neighbors are the closest embeddings in the original distributional space. The retrofitted neighbors are the closest embeddings in the RetroGAN generated FastText-based space. We can see that “doggo” was in use with slang (“bae”, ”furbabies”) and after being retrofitted it now is closer to what we would expect with “dog”.

Even if we do manage to achieve the same KB concept coverage as the embeddings we are retrofitting to, we still run into the problem of being able to get the knowledge back in an explicit manner. By retrofitting, some of the information in the knowledge base is being fused with the one found in the embeddings, but it is stored as a vector for the concept rather than as a graph. We need to be able to deconstruct this knowledge into a form that we could utilize to make inferences. To this end, we developed a multi-task learning system called Deep Relationship Discovery (DRD) which takes pairs of the RetroGAN fitted embeddings and a list of assertions from a knowledge base, in our case ConceptNet, and proceeds to learn to predict relationship assertion validity. Intuitively, DRD infers assertions in a manner similar to analogies. One example of how this analogical reasoning works is trying to infer whether a squirrel desires a brownie. We could make an analogy like, ”A squirrel likes a brownie” because ”A bear likes honey”. We can make the inference because both concepts (squirrel and bear) are animals, and animals like food. We train our Deep Relationship Discovery system on data for a Commonsense KB completion task and demonstrate that is capable of inferring unseen assertions.

In the following sections we give some work that is being done in this area of KB inference and retrofitting. Following that, we describe in more detail training and evaluation for RetroGAN and Deep Relationship Discovery.

2. Related Work

Within the field of retrofitting, much work has been done in exploring the ways in which to retrofit. The original work by [Faruqui et al., 2014] only used synonymy relationships, this did not give counter examples. The attract-repel work by [Mrkšić et al., 2017] looked to incorporate antonymy relationships to be able to address this. More recently the work done by [Vulić and Mrkšić, 2017] looks to incorporate the asymmetric lexical entailment relationship. We do not utilize this retrofitting procedure in this work because according
to the authors did not improve/degrade the performance of attract-repel. Building on this work, there have been neural models that have been utilized to learn retrofitting mappings such as [Glavaš and Vulić, 2018] and [Ponti et al., 2018] which use a Deep Feed Forward Network and a Generative Adversarial Network respectively. The important part of this work is that one is able to, if given a static word embedding, generate a retrofitted embedding on the fly. As it stands, many systems right now are focused on using contextual embeddings such as BERT [Devlin et al., 2018] on downstream tasks. Only recently have there been some efforts in trying to incorporate external, knowledge base assertions into these pre-trained systems. Some notable efforts are KnowBERT [Peters et al., 2019], Align-mask-select [Ye et al., 2019], and [Lauscher et al., 2019]. Additionally there has been research on trying to understand the extent to which these pre-trained models understand commonsense [Wang et al., 2020]. All of these efforts give some performance improvements, but there is still more work to be done in integrating external knowledge into contextual representations.

Commonsense Knowledge Base completion and inference is another field that has had extensive work done on it. There have been works on utilizing dimensionality reduction techniques to be able to fuse similar assertions and be able to make inferences with the fused knowledge. One of these examples is AnalogySpace [Speer et al., 2008]. More recently, work has been shifting to open domain commonsense question answering [Talmor et al., 2018] given that we can generate contextual embeddings. This line of work only highlights the need to be able to incorporate external knowledge to boost a system’s commonsense understanding. There have also been the approaches of COMET [Bosselut et al., 2019], [Saito et al., 2018] and [Li et al., 2016]. COMET uses a transformer model trained on a commonsense dataset to be able to generate commonsense assertions. [Saito et al., 2018] propose a joint learning method for an attention-based encoder-decoder that incorporates both commonsense KB completion and commonsense KB generation. [Li et al., 2016] use an LSTM method combined with a hidden layer to be able to score assertions.

3. RetroGAN

Retrofitting is a process in which word embeddings are post processed and optimized to favor a certain optimization criteria. In one of the most recent work on retrofitting [Mrkšić et al., 2017], the criteria enforces that concepts which are connected in a knowledge graph through an attract (synonymy) relationship should have similar embeddings and concepts that are connected through a repel (antonymy) relationship should have dissimilar (further apart) embeddings. Retrofitting greatly improves the performance of static embeddings in word similarity and the performance of downstream tasks such as lexical simplification. However, the retrofitting operation can only be performed on embeddings whose concepts are present in a KB. This leads to typically having high performance on a small subset of the entire vocabulary of a word embedding corpus [put percentages]. The line of work by [Ponti et al., 2018, Glavaš and Vulić, 2018] tries to remedy this problem by making neural architectures that learn a mapping for retrofitting. RetroGAN builds on [Ponti et al., 2018] by utilizing a CycleGAN-like architecture. The benefits to this architecture are that since the domain transformation is constrained to be cyclic, we preserve the information from both domains as they are transformed one to the other, and the generations that we make will be constrained to create more realistic transformations. We focus particularly on static
embeddings (FastText[Bojanowski et al., 2017]/Glove[Pennington et al., 2014]) rather than contextual embeddings, (BERT[Devlin et al., 2018] and other transformer-based systems) because there is still no effective way of being able to inject the external KB constraints to enforce explicit semantics into these systems. We note that there are efforts in this area that seem promising with regards to injecting constraints [Wang et al., 2020, Peters et al., 2019, Lauscher et al., 2019], and we also note that there is some evidence of having a some commonsense knowledge present in these approaches but it is not as strong as explicit knowledge[Kwon et al., 2019].

3.1 Architecture

The high level architecture that we use for the RetroGAN system can be seen in Figure 1.

![Figure 1: RetroGAN System Architecture](image)

Within our architecture we have 2 GAN networks that interplay to balance a cyclic loss to be able to transform a word embedding from its original domain over to its retrofitted domain and backwards. The GANs that we employ have the following structure. The generator component consists of an input layer followed by 2 hidden dense layers with 2048 neurons and each with a dropout layer (with a percentage of 0.2 for the dropouts) at their output, and a final output layer with a linear activation function with the same dimensionality as the input layer since we are transforming an embedding to another kind of embedding. The hidden layers employ the ReLU [Nair and Hinton, 2010] activation function. The discriminator component has a similar structure (an input layer, 2 hidden layers with dropout but percentage of 0.3), however, the second hidden layer is followed by a batch normalization layer and the output layer is a single neuron with a sigmoid activation. The reason for the batch normalization layer was to stabilize the training, and the sigmoid output is to be able to classify whether the generated embeddings are valid or not.
3.2 Training and Evaluation

In RetroGAN we have 4 optimization objectives: the regular adversarial loss for both GANs, the cyclic loss for the joint generators, the identity loss for both of the generators, and we add the max margin loss similar to [Weston et al., 2011][Ponti et al., 2018] to the generators. The combined objective has the following form: $L(G, F, D_X, D_Y) = L_{GAN}(G, D_Y, X, Y) + L_{GAN}(F, D_X, Y, X) + \lambda L_{cyc}(G, F) + L_{identity}(G, F) + L_{max-margin}(G, F)$. Where $L_{GAN}$ is the adversarial loss as proposed by [Goodfellow et al., 2014], $L_{cyc}$ is the cycle consistency loss and $L_{identity}$ is the identity loss used by [Zhu et al., 2017] and $L_{max-margin}$ is the max margin loss with confounders as used by [Ponti et al., 2018]. For training the systems we utilize the popular ADAM [Kingma and Ba, 2014] optimizer with a learning rate of 0.0001 for the generators and 0.001 for the discriminators. Our reasoning for this is that we also run 2 steps of optimization for the discriminator to keep it at a minimum, but want the discriminator to catch up with a higher learning rate. Additionally, we set the margin in the max margin loss to 1 and run 25 iterations of random confounders. We note that there was no tuning for these parameters. We train for 150 epochs with a batch size of 32.

We set the lambda constant for the Cycle loss to be 1 and the weight for the identity loss to be 0.01 since we do not want it to influence too much on the rest of the learning, just as a check of whether the embedding is already in the correct domain or not. Additionally in epochs 100 and 125 we drop the learning rate by a factor of 10 on each of the optimizers because in our observations around epoch 100 the system has mostly converged in some cases.

![Figure 2: SimLex and SimVerb evaluation on word embeddings](image)

We now describe the embeddings that we use in our tests. We utilized the Glove trained on the common crawl with 840 billion tokens and dimensionality of 300 as what we describe as the Glove-CC model. For FastText we utilized 2 types: wiki-news trained with subword information (FastText-wiki) and the english common FastText with subword information (FastText-cc). To evaluate the system we used the generated embeddings to run the word similarity benchmarks: SimLex (SL)[Hill et al., 2015] and SimVerb (SV)[Gerz et al., 2016].
The results for these are listed in Figure 2. Similar to how the authors of [Ponti et al., 2018] perform their evaluation we utilize the Disjoint and Full settings for SimLex and SimVerb. The disjoint setting consists of performing attract-repel along with the constraints from [Ponti et al., 2018] without having the words from SL and SV present in the constraints and without having them in the training data. In this setting the words are completely unseen for the model. The full setting in contrast the words are included in the constraints and in the training data with the idea of seeing how much more they are modified and if performance increases or not. We note that our results are very similar to the AuxGAN system. In the case of the Full setting, we note the same observations that were noted in AuxGAN: there are some inconsistent gains and losses in the full setting, which may be due to including the SL/SV words in the constraints and the system to a certain extent is trying to match the data that it knows and is not trying to explore the generation space too much.

We note that it seems that AuxGAN system was evaluated on an attract-repel retrofitting with a subset of Glove and FastText. We found that utilizing the complete vocabulary to perform attract-repel gave different results in which the performance of Glove and FastText was inverted. We utilized the publicly available AuxGAN model, with the default settings, to train on this extra set of training data and note that our model performs significantly better in some cases. We believe this may be due to the configuration of training for the AuxGAN model. We note that we achieved comparable performance to the AuxGAN model as early as in 50 epochs of 1000 iterations, which is over an order of magnitude faster than AuxGAN (10 epochs of 1000000 iterations in their default configuration). In some cases such as Full FastText our model underperformed, and we believe it is due to the training not having reached the optimum before beginning to slow down. Overall, our system achieved a higher performance on the Disjoint setting which demonstrates that it has a better generalization capability for unseen words.

Additionally, similar to [Ponti et al., 2018] we also evaluated on the Light-LS[Glavaš and Štajner, 2015] with the default dataset [cite this] and saw that with the full versions of the word embeddings we achieved better performance than [Ponti et al., 2018]. We note that the tool utilized to generate the accuracy measurements in [Ponti et al., 2018] and [Glavaš and Vulić, 2018] is no longer available. Instead we use the publicly available Github 1 one with the arguments: “-tc 0.05 -st 0 -cd 0”. We evaluate the accuracy of the substitutions, this being defined as the amount of words that are substituted correctly (found in the Mechanical Turk Dataset options) out of the total words that have to be substituted (which in this case is 500). The results for this benchmark can be seen in Table 3.2. The purpose of this benchmark is to demonstrate the usefulness of being able to generalize retrofitting to the rest of the word vector corpus in downstream tasks. We note that RetroGAN’s performance is higher within the Full setting which utilized the FastText-CC data. We also note that there are inconsistencies with the performance gains in particular for Glove. We believe this could be undertraining seeing as how the performance in the Full setting was considerably less than what the attract-repel was able to accomplish.

---

1. https://github.com/codogogo/lightls
<table>
<thead>
<tr>
<th>Models</th>
<th>Full FastText-CC</th>
<th>Full Glove-CC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributional</td>
<td>38.2</td>
<td>30.0</td>
</tr>
<tr>
<td>Specialized (AR)</td>
<td>36.4</td>
<td>44.2</td>
</tr>
<tr>
<td>AUXGAN</td>
<td>38.0</td>
<td>35.4</td>
</tr>
<tr>
<td>RetroGAN</td>
<td>47.0</td>
<td>42.6</td>
</tr>
</tbody>
</table>

Table 2: Light-ls accuracy measurements with the publicly available Python tool.

4. Deep Relationship Discovery

Given that we can generate word embeddings that have generalized/expanded semantic information from a knowledge graph inside of them, we now need a way to extract and utilize that knowledge. We attack this problem by trying to predict which assertions are possible from pairs of retrofitted word vectors. We build a Multi-Task Learning (MTL) system whose inputs are a pair of retrofitted word embeddings, and whose outputs are the strength of the relationships present in an existing knowledge base. In our specific case, it would be that the inputs to our system are going to be pairs of the vectors that are generated from the RetroGAN system, and the outputs are going to the strength of the set of relationships found in ConceptNet. Intuitively our system should learn to associate assertions to semantically similar input concepts which could be seen as a way of analogical reasoning.

4.1 Architecture

The Deep Relationship Discovery system can be broadly viewed in three sections: an input section, a common body section, and the output section.

![Figure 3: DRD System Architecture](image-url)

The input section consists of 2 tracks which are for each of the word vectors. Each track consists of a densely connected layer of 1024 neurons followed by a batch normalization layer followed by a dropout layer (with the percentage being 0.1). This is followed by another densely connected layer of 256 neurons and another batch normalization layer. The purpose
of each of these tracks is to abstract the input word vectors. This is done for both of the input vectors individually with the idea being that the relationships we are trying to learn are not necessarily symmetric. The output of these blocks is then concatenated and passed to the common body section. The common body section is the area that is common to all of the "tasks" that we want to predict. In our case these tasks are assertions whose weights are the strength of the relationships. The common body consists of 3 densely connected layers of 512 neurons followed by a batch normalization layer and a dropout layer (with the percentage being 0.1). The intuition behind these layers is that the pertinent information in both of the inputs is fused and that fused representation is even further abstracted. Intuitively in this area, as the system is trained, we learn the information about the knowledge base, how it is arranged and how things relate to produce certain assertions. The output section contains a task "tail" for each of the relationships that we want to predict the validity of. Each tail section is composed of a densely connected layer of 512 neurons followed by a batch normalization layer and a dropout layer, which are followed by another densely connected layer with a batch normalization layer and a final output layer with a sigmoid activation function. The rest of the layers have the ReLU activation function. The intuition for this section is that each tail takes the common body as an input and extracts the information that it needs to be able to determine the strength of the assertion that the tail represents.

4.2 Training and Evaluation

We train DRD on the training ("train600k") set provided by [Li et al., 2016]. We intended to train for 100 epochs, unfortunately given time constraints we were only able to train for 20 epochs. We start with an initial batch size of 32 which grows to 96 through a generation of 64 counter examples by selecting a random concept from the batch to be the first concept in a new assertion and by selecting another random concept to be the second concept in another new assertion (this idea was implemented in [Li et al., 2016] and amounts to an extra 2 assertions per true assertion). In every epoch, we cycle training through every output task, in a random order per epoch. We consider an epoch to be in which each iteration we train until the exhaustion of data of 63% of the relations (tasks) that we are training. The reason for this is that the remaining relationships produce a data imbalance (there is much more data for the remaining ones than there is for the other 63%). We test our model on the dev set for the corpus to select the threshold for the testing to determine whether it is a 1 or a 0 during testing. The results of our system can be seen in table 4.2. We believe that given more epochs and possibly alternating the epochs to have the same amount of assertions per relation could improve our performance considerably. We are currently finding out more ways to evaluate being able to infer relationships from out of knowledge (OOK) concepts, which we believe is a unique feature of this entire system given that we can generate a FastText embedding for the OOK concepts and be able to RetroGAN-retrofit them and generate a commonsense based KB out of them.

5. Conclusion

This work presents an expansion on work done to generalize retrofitting mappings through the use of a CycleGAN-like system called RetroGAN. We show that this system is capable
Table 3: Test set accuracy for a commonsense knowledgebase completion task.

<table>
<thead>
<tr>
<th>System</th>
<th>Test Set Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commonsense KB Completion [Li et al., 2016]</td>
<td>0.925</td>
</tr>
<tr>
<td>Commonsense KB Completion and Generation [Saito et al., 2018]</td>
<td>0.954</td>
</tr>
<tr>
<td>COMET [Bosselut et al., 2019]</td>
<td>0.925</td>
</tr>
<tr>
<td>DRD</td>
<td>0.744</td>
</tr>
</tbody>
</table>

of retrofitting effectively out of knowledge embeddings. Additionally, we develop a novel way to discover common sense based assertions between concepts, by training a multi-task learning system on a subset of the assertions present in ConceptNet. We explored the combination of the RetroGAN system with the Deep Relationship Discovery one to be able to infer assertions from concepts that may or may not be in the vocabulary, and that may or may not be in the knowledge base. By training Deep Relationship Discovery on RetroGAN produced embeddings and ConceptNet assertions, if given a set of concepts, we are able to get a basic commonsense understanding of a set of concepts through the generation of a knowledge base of their embeddings.

Acknowledgments

Omitted for blind review.

References


