Neural-Network-Based Reasoning

Neural networks have proved effective primarily for learning to classify images, speech, text and other patterns, but methods are being investigated for training them to produce more complex outputs, including answers to questions requiring inference. Only very simplified versions of such tasks can be realized at present, and they require massive training data, but there is increased interest in this area

NLP and Relation Inference

Deep neural nets (DNNs) have had a lot of success in various "shallow" pattern transduction tasks. Reasoning requiring multiple steps and use of world knowledge (including QA dependent on reasoning, and natural language dialogue) are still largely outside the scope of DNN methods, but activity in this area is growing. For instance, inferring relations such as that Pablo Picasso's citizenship was Spanish, given that he was born in Spain, and given many examples of individuals born in Spain who have Spanish citizenship, is an example of a simple inference that is currently possible. Some more directly logical methods are beginning to be addressed as well.

These notes are primarily about a noteworthy 2013 paper from Stanford about relational inference (like the one about Pablo Picasso) using DNNs. However, recent (2022) work by Andrew McCallum's group at UMass excels all earlier approaches to relation QA, even though it uses DNNs only to code NL questions as knowledge-base queries, while finding answers by statistical-similarity methods applied to entities and paths in the relational database. The notes also provide a brief look at some other work aimed at NN-based reasoning.

R. Socher, et al., Reasoning with neural tensor networks ...

Reference: R. Socher, D. Chen, C.D. Manning, and A.Y. Ng, "Reasoning with neural tensor networks for knowledge base completion", NIPS 2013.
https://nlp.stanford.edu/pubs/SocherChenManningNg_NIPS2013.pdf

Authors' abstract:

> Knowledge bases are an important resource for question answering and other tasks but often suffer from incompleteness and lack of ability to reason over their discrete entities and relationships. In this paper we introduce an expressive neural tensor network suitable for reasoning over relationships between two entities. Previous work represented entities as either discrete atomic units or with a single entity vector representation. We show that performance can be improved when entities are

represented as an average of their constituting word vectors. This allows sharing of statistical strength between, for instance, facts involving the "Sumatran tiger" and "Bengal tiger." Lastly, we demonstrate that all models improve when these word vectors are initialized with vectors learned from unsupervised large corpora. We assess the model by considering the problem of predicting additional true relations between entities given a subset of the knowledge base. Our model outperforms previous models and can classify unseen relationships in WordNet and FreeBase with an accuracy of 86.2% and 90.0%, respectively.

Let's first take a look at the kinds of inference examples they are targeting. For example, they want to compute the likelihood of

- (*Pablo Picasso, nationality, Spain*), using the available Freebase triples,[1] not including this particular triple but perhaps ones like (*Pablo Picasso, place of birth, Malaga*) and (*Malaga, located in, Spain*) (where the subject and object have unique identifiers associated with them), and many instances like (*Cervantes, nationality, Spain*), (*Cervantes, place of birth, Alcala de Henares*), (*Alcala de Henares, located in, Spain*), etc. In other words, this is a kind of analogy making;
- (*German shepherd, hypernym, vertebrate*), given the Wordnet relation between German shepherd and dog, and between dog and vertebrate; it's unclear if the latter was given directly or indirectly. WordNet provides a chain from *dog* to *canine* to *carnivore* to *placental mammal* to *mammal* to *vertebrate*, so the question is whether the system was provided transitive closure information, or had to figure this out; probably the former. If so, then the training data may also have contained transitive closure information such as (*beagle, hypernym, vertebrate*), which would greatly simplify analogy-making.

To get a sense of their method, let's first of all review how the operation of a simple neural unit with d numerical inputs and one output can be written in vector notation. Suppose an entity (participating in a relation of interest) is represented as a column vector $e_1 = (e_{11}, ..., e_{1d})^T$ of d numerical features. (Since we've written down the elements as a row vector for convenience here, we get the column vector by transposing it, as indicated by superscript T.) These (so-called *embedding*) features are typically based on *co-occurrence frequencies* of the entity name with other words in large sets of sentences or in a DB like Freebase (with frequencies of high-frequency words like *the* or *have* deemphasized); d may be very large to begin with, but is typically reduced by *dimensionality reduction* methods – the authors used $d = 100$. In such a vector space, similar entities tend to be close together (in terms of the cosine between them), as a result of their tendency to occur in similar contexts (similar nearby words).

The weights applied to the components of e_1 can be written as a row vector V, also of length d. Then the weighted sum of e_1-components is $\sum_{i=1}^{d} v_i e_{1i}$, often written as

[1] There are nearly 2 billion, extracted from Wikipedia and "curated" by human judges

$$Ve_1 = (v_1, ..., v_d) \begin{bmatrix} e_{11} \\ \vdots \\ e_{1d} \end{bmatrix}.$$

We can then add a bias constant b to this weighted sum, and apply a sigmoid function or the *tanh* function f to obtain the unit's output $g(e_1)$, also throwing in a final scale factor u [normalizing the output?], i.e.,

$$g(e_1) = u.f\left((v_1, ..., v_d) \begin{bmatrix} e_{11} \\ \vdots \\ e_{1d} \end{bmatrix} + b\right).$$

This might for example be suitable for classifying the type of a given named entity (represented in terms of its word co-occurrence features) as being a person or something else.

Neural tensor networks

So now suppose we're trying to determine if two named entities e_1, e_2 are in a particular relation R (such as *nationality*, relating Picasso and Spain). The simplest way is just to apply the above method to the concatenation of e_1 and e_2, which is a column vector of length $2d$; the (row) weight vector V will now also be of length $2d$. So we have (without writing out elements of the concatenated e_1 and e_2),

$$g(e_1, R, e_2) = u.f\left((v_1, ..., v_{2d}) \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} + b\right).$$

But according to the authors, this simple NN, combining the two vectors linearly before applying nonlinear thresholding, (a) does not sufficiently allow for the relationships among their elements, and (b) does not allow for the fact that two entity names may share some substrings (Bengal tiger, Sumatran tiger), and therefore in general an entity should be represented in terms of several vectors, corresponding to separate co-occurrence behavior for their constituent words. They allow for k vectors per pair of entities, with $k = 4$ in their experiments. So this should allow separate modeling of the interaction beween the words of any 2-word entities (or between a 3- or 4-word entity and a 1-word entity).

To relate entity vectors more "intimately" than just forming the weighted sum of their concatenation, they combine them in the following fashion (which in effect allows for products of components of e_1 with components of e_2):

$e_1^T W e_2$,

where W is a d by d weight matrix. In vector/matrix algebra the order of multiplication is generally taken to be rightmost-to-leftmost. So we are applying W to column vector e_2, thus linearly transforming it and obtaining another column vector of length d; then we're applying row vector e_1^T (i.e., $(e_{11}, ..., e_{1d})$) to that result, obtaining a single number.[2] Note that if W were the identity matrix (with 1's on the diagonal and 0's elsewhere), $e_1^T W e_2$ would just be the dot product of e_1 and e_2, which makes clear that $e_1^T W e_2$ allows element-wise interaction between e_1 and e_2.

That takes care of point (a) above; to allow for (b), the authors use k weight matrices like W, where each corresponds to two words, one from each of the two entity names (as mentioned, allowing for up to 4 combinations). In accord with common practice, they term the resulting d-by-d-by-k array

$W^{[1:k]}$

a *tensor*. In ML, a tensor is just an array with more than 2 dimensions. Having a term distinct from *vector* and *matrix* is appropriate because once we have 3 or more dimensions, a variety of different kinds of products can be defined, beyond those at lower dimensions. The only products we need here, however, are the matrix products $e_1^T W^{[i]} e_2$ using each d-by-d "slice" of $W^{[1:k]}$. The single numbers yielded by each of these products are regarded as forming a column vector of length k. We write this as $e_1^T W^{[1:k]} e_2$.

This column vector, $e_1^T W^{[1:k]} e_2$, becomes the first term in the revised expression to which threshold function f is applied. The second term is like the simpler one worked out above, based on the concatenation of e_1 and e_2, but with weight vector V now extended to have k rows, intuitively intended as an appropriate weighting for each combination of a word of e_1 with a word of e_2:

$$V \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} = \begin{bmatrix} v_{11} ... v_{1,2d} \\ ... \\ v_{k1} ... v_{k,2d} \end{bmatrix} \begin{bmatrix} e_{11} \\ \vdots \\ e_{1d} \\ e_{21} \\ \vdots \\ e_{2d} \end{bmatrix}.$$

So instead of a single number, we now obtain another length-k column vector. We also double up the bias term b into a k-vector. The sum of these three k-vectors is

[2] In general, when you multiply two matrices, you take the dot product of row i of the first matrix with column j of the second matrix, to get the element of the i^{th} row, j^{th} column of the result. (The dot product is the sum of element-by-element products.) So, multiplying a matrix with m rows and k columns times a matrix with k rows and n columns, gives an m-by-n matrix. In the two multiplications at hand, we are first multiplying a d-by-d matrix times a d-by-1 matrix (a column vector), yielding another d-by-1 matrix, and then multiplying a 1-by-d matrix (a row vector) by a d-by-1 matrix (column vector), yielding a 1-by-1 "matrix", i.e., a number.

then thresholded element-by-element using f, and finally a scaling vector u of length k (instead of a single scale factor) is used to obtain the desired result – a weighted decision whether relation R holds between e_1 and e_2 (perhaps just the average of the k individual +1/-1 "decisions" based on the k constituent word combinations):

$$g(e_1, R, e_2) = u^T.f\left(e_1^T W^{[1:k]} e_2 + V \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} + b\right).$$

Of course, this particular "neural tensor network (NTN) is aimed at a single relation R, and the authors actually trained and tested separate NTNs for about a dozen relations from each of Freebase and WordNet. So in the paper, they subscript the various weight arrays with R in the above formula: $u_R^T, W_R^{[1:k]}, V_R,$, and b_R.

Comments

The authors report improvements over earlier methods, moving accuracy scores on positive and negative relations (the latter restricted to appropriate entity types, exclusive of ones like (*Pablo Picasso, nationality, Rembrandt*)) upward from about 86% to more than 88%. Getting it right for nearly 9 out of 10 "questions" is quite impressive.

The main formula for the neural tensor network above may give the impression that there's only one thresholding operation f being applied to a 2-element column vector – a 2-unit NN, where each unit has $2d$ weighted inputs. But viewed in terms of standard NN units, this is deceptive: The multiplications in the tensor term form products of the components of the e_1 and e_2 vectors, and multiplication is not what standard NN units do. So hidden in the tensor term – if we were to implement the network using standard NN units – there would be layers that do the multiplications, and for this nonlinear operation many additional units would be required.

However, the lesson learned from NN research is not so much that we should limit ourselves to basic NN units, but rather that we should use layers of linear and nonlinear operations, where the resulting outputs are differentiable with respect to the free parameters of the system. The question of how such generalized methods might be implemented in mammalian brains seems not to concern researchers in this area too much, especially given the theoretical knowledge that just about anything can be done with 2 hidden layers and enough neural units. Differentiability ensures that we can apply learning methods based on gradient descent-like optimization or backpropagation. It might be an interesting biological question how brains implement various complex functions using organic neurons, but for NN developers it would just make the learning problem harder to replace multiplication units, etc., with multiple layers of more elementary units.

The authors regard their results as demonstrating "commonsense reasoning". This is a bit of an overstatement. Keep in mind that the information in Freebase is in a very simple, regularized form. There is nothing enabling an inference like, "John got stuck on his way to work" given that his car got a flat tire on his way to work, and

knowledge such as "If a car gets a flat tire, it can no longer be driven", among other necessary items. Essentially the reasoning in the NTN, such as it is, consists of making analogies – i.e., similar entities tend to be related in similar ways. (The ConceptNet system, the main product of the Open Mind Common Sense project at MIT, has similar capabilities.) Moreover, a dozen static relations don't go very far in reasoning about our dynamic world. The Wordnet relations used were also very simple hierarchy relations, it seems (they don't specify).

A paper at the 2019 NeurIPS conference (previously NIPS, the premier machine learning conference) goes beyond the set of relations covered by Socher *et al.*, handling about 1345 of the 25,000 relation types in Freebase:

Meng Qu & Jian Tang, "Probabilistic Logic Neural Networks for Reasoning" (2019, mentioned earier). https://arxiv.org/pdf/1906.08495.pdf

By combining multiple techniques, including Markov Logic Networks, with the corresponding distribution optimized via variational EM (expectation maximization), and knowledge graph embeddings, they achieved success rates close to 90% in verifying/disconfirming omitted triples, which is close to what Socher *et al.* achieved for 13 relation types. However, note that verification is easier than actually *finding* an entity e_2 that satisfies an incomplete relation $(e_1, R, ?)$. Until the following work by Andrew McCallum's group, it was difficult to get accuracies above 70%.

Dung Thai, et al.: A Case-Based Reasoning Approach for Question Answering ...

Reference: Dung Thai, Srinivas Ravishankar, Ibrahim Abdelaziz, Mudit Chaudhary, Nandana Mihindukulasooriya, Tahira Naseem, Rajarshi Das, Pavan Kapanipathi, Achille Fokoue, and Andrew McCallum, "CBR-iKB: A case-based reasoning approach for question answering over incomplete knowledge bases", arXiv:2204.08554v1, https://arxiv.org/pdf/2204.08554v1.pdf.

The authors refer to databases of relational triples as *knowledge graphs* (KGs), as has become rather common in AI. (Traditionally, KGs were called *semantic networks* – and these generally allowed more FOL-like propositions, above and beyond binary relations over specific entities.) They tested on Freebase-derived KGs, among others, such as a movie database. In their examples, they use KG relations like

"Ken Whisenhunt" —head_coach–> "Tennessee Titans",
"Ken Whisenhunt" —head–> "Ryan Tannehill"
"Ryan Tannehill" —play_for–> "Tennessee Titans", etc.

Their system accepts NL questions such as *"What team does Ryan Tannehill play for?"*, which is translated into a KG query

("Ryan Tannehill" play_for ?).

The translation is the only aspect of the system using DNNs, namely a language model

(LM). The training data for this consist of questions and answer-entities, and the goal is to derive queries like the above in such a way that the answer is verified in the KG by a relation like

"Ryan Tannehill" —play_for–> "Tennessee Titans".

The basic idea is to answer questions (e_1 R ?) that are not directly answered by a link (e_1 R e_2) in th KG by using paths from e_1 to candidates e_2 in the KG that are *similar* to paths from e_1 to the correct answer e_2, and/or similar to paths from entities e'_1 similar to e_1 to the correct answer e'_2. Answers are correct if they are given by a direct link in the knowledge graph, or (less reliably) by relation extraction from relevant NL documents.

So this requires notions of *entity similarity* and *path similarity*. First, path similarity is just based on the sequence of relations (rather than entities) on the path. And then entity similarity is based on having similar sets of emerging paths from the entities. We won't get into the technicalities of the similarity measures. They use hierarchical clustering methods to organize the entities, so as to enable k-NN (k-nearest-neighbor) retrieval of entities similar to a given entity.

Let's call the sequence of relations (R_1, R_2, ..., R_k) a "rule" applicable to query (e1 R ?) if there is a direct link (e_1 R e_2) and a path (e_1 R_1 e' R_2 e'' ... R_k e_2) from e_1 to e_2 in the KG. (*Note:* the rule could still just be a single relation; e.g., besides the "head coach" link above, the KG might also include "Ken Whisenhunt" —coach–> "Tennessee Titans", which would be alternative evidence that the head_coach relation holds.)

They evaluate the reliability of such rules by checking all paths with the same sequence of relations, and seeing how often a direct link ($e_1 R e_2$) is present in the KG when the path leads from e_1 to e_2. It seems the rule reliability is further evaluated by also checking how often the rule gives correct results (e'_1 R e'_2) for elements e'_1 *similar* to e_1. In other words, rule reliability is evaluated on groups of similar entities. Correctness of answers supplied by rules may also be checked when there's no direct link (e'_1 R e'_2), but text-based relation retrieval suggests its correctness.

Then, when a new question is asked, and mapped to ($e_1 R$?) via the LM, and it can't be directly answered via an existing link ($e_1 R e_2$), they first of all find the rules that apply to (e'_1 R ?) for entities e'_1 similar to e_1, using the k-NN method. The rules starting at those nearest neighbors are then applied to find e'_2 candidates, and an answer is selected based on the reliability and degree of consensus of the rules employed.

They test on Freebase-derived datasets, and get substantially better accuracy (78%) than previous methods (66-71%), and way greater accuracy (70%) than others (30-48%) when half the data are omitted. The lesson here seems to be that **use of an explicit knowledge base and structural similarities within the knowledge base can outperform DNN methods whose "knowledge" is entirely implicit in the DNN parameters.**

B. Peng et al., Towards neural network-based reasoning

Reference: B. Peng, Z. Lu, H. Li, K.-F. Wong, "Towards neural network-based reasoning", Aug. 2015, Cornell Univ. Library. (This seems not to have been published in a conference venue or journal, but is frequently cited anyway.) https://arxiv.org/pdf/1508.05508v1.pdf

The authors tackle the bAbI question answering set; each has a few facts, and a couple of questions, with answers. There are thousands of (algorithmically generated) instances for various "tasks". Two examples:

1. The office is east of the hallway.
2. The kitchen is north of the office.
3. The garden is west of the bedroom.
4. The office is west of the garden.
5. The bathroom is north of the garden.

How do you go from the kitchen to the garden? south, east; relies on 2 and 4.
How do you go from the office to the bathroom? east, north; relies on 4 and 5.

1.The triangle is above the pink rectangle.
2.The blue square is to the left of the triangle.

Is the pink rectangle to the right of the blue square?
Yes; relies on 1 and 2.
Is the blue square below the pink rectangle?
No; relies on 1 and 2.

The approach here looks interesting. It makes use of many seemingly separate NNs. The words of a fact or sentence are represented as feature vectors (as in the above paper). At the base level, a given question is represented, alongside the entire sequence of known facts. The word sequence corresponding to the question and each fact is processed by a type of recurrent NN (RNN) called a " gated recurrent unit" (GRU), which like an LSTM uses Hadamard gates (component-by-component multiplication) to enable long-term memory without "vanishing or exploding gradients", and seems to require fewer training data than LSTMs. (For some details on GRUs, see e.g., https://en.wikipedia.org/wiki/Gated_recurrent_unit.) Each fact RNN provides its output, paired with the output of the question RNN, to a separate DNN, yielding an altered representation of the question paired with an altered representation of the fact. (Each has to some extent influenced the other – this is thought of as some sort of tacit inference process). Then the altered question representations (one for each fact) are pooled into a single altered question representation, using a "softmax" operation; this is again combined via a DNN with each (now altered) fact, etc. This process of combining the altered question with each altered fact, followed by question pooling, continues over several "inference" layers. The final pooled question representation is fed to a (separately trained) answering module.

Comments

They seem to do very well in "end to end" reasoning [as opposed to step-by-step supervised training? Haven't really tried to understand]. As always, the NN methods use alternating linear operations (using matrices or tensors on the vectors "passing through" the successive layers) and element-wise nonlinear operations (softmax, e.g., for a 3-vector x, y, z this is $e^x/S, e^y/S, e^z/S$, where S normalizes the vectors to sum to 1, allowing their interpretation as probabilities; or tanh – the hyperbolic tangent, which goes from -1 at $-\infty$, through 0 at 0, and to 1 at $+\infty$. These are differentiable, as required for implementing backprop.

Of course, the simple, artificial facts used in these experiments are quite restrictive, and we can't tell the system any general facts. Indeed, the idea of bAbI is that the system should automatically internalize general rules based on thousands of closely related examples. But it's unclear whether rules in any sense are actually learned, e.g., when x goes from place y to place z, then x ends up at z and is no longer at y; or that if x is to the left of y then y is to the right of x. For all we know, the system is again just making analogies between "patterns of examples" and the corresponding questions and answers.

As a note on more recent work, it's worth listing some papers from NeurIPS conferences, and an arXiv paper on finding informal inference rules. NeurIPS 2019 had over 1000 papers, very few on reasoning; I haven't checked more recent conferences several of the ones below are just posters:

Schlag & Schmidhuber, "Learning to Reason with Third Order Tensor Products" (2018). [From the abstract: "We combine Recurrent Neural Networks with Tensor Product Representations to learn combinatorial representations of sequential data. This improves symbolic interpretation and systematic generalisation." They tackle the bAbI dataset, like Peng *et al.* mentioned above, and do very well on it. The methods seem more general than Peng *et al.*] https://papers.nips.cc/paper/8203-learning-to-reason-with-third-order-tensor-products.pdf

Robert Ness, Kaushal Paneri, & Olga Vitek, "Integrating mechanistic and structural causal models enables counterfactual inference in complex systems" (2019); poster, not yet available.

Wang-Zhou Dai, Qiuling Xu, Yang Yu, & Zhi-Hua Zhou, "Bridging Machine Learning and Logical Reasoning by Abductive Learning" (2019); not yet available.

Vaishak Belle & Brendan Juba, "Implicitly learning to reason in first-order logic" (2019). A theoretical, theorem-oriented paper, not NN-based; https://delbp.github.io/papers/DeLBP-2019_Accepted-1.pdf

Drew Hudson & Christopher Manning, "Learning by Abstraction: The Neural State Machine for Visual Reasoning" (2019). [QA about images, e.g., "What is the red fruit inside the bowl to the right of the coffee maker?".] https://arxiv.org/pdf/1907.03950.pdf

Meng Qu & Jian Tang, "Probabilistic Logic Neural Networks for Reasoning" (2019, mentioned earier). https://arxiv.org/pdf/1906.08495.pdf

So the field is moving forward, attempting to capture more of human understanding and reasoning. However, the only truly competent (though generally specialized) dialogue systems and reasoning systems in AI remain those based on symbolic semantic representations and knowledge representations.

Weir and Van Durme, "'Dynamic generation of interpretable inference rules...

A recent paper on inference rule generation is

Nathaniel Weir and Benjamin Van Durme, "Dynamic generation of interpretable inference rules in a neuro-symbolic expert system", arXiv:2209.07662 [cs.CL]. https://arxiv.org/abs/2209.07662.

Starting with facts and conditionals (in English) as a KB, they train on correct conclusions (claims stated in English) requiring inference from the KB. Neural nets are used in multiple ways: sequence-to-sequence (seq-to-seq) transducers (with encoder & decoder layers) to map questions into a declarative form likely to be suitable for the proof attempt; "soft unification" of predicate-argument pairs using an NN encoder whose outputs can be compared with a cosine similarity metric; and seq-to-seq models that try to decompose a hypothesis statement into a pair of statements that might support the hypothesis, aided by templates (with gaps) that might be suitable for the supporting statements.

In trying to confirm a hypothesis h, they search for potential rules – typically stating that two KB facts f_1 and f_2 imply a conclusion h – that enable proofs of the conclusions in the training corpus. For example, they combine the facts f_1, f_2 to obtain the conclusion h:

f_1: Chlorophyll is used by plants to produce carbohydrates;
f_2: Carbohydrates are made of sugar;

h: Plants **use** chlorophyll **to** produce sugar.

Actually, in this example f_1 is itself a fact derived by such inference rules, by combining

f_3: Chlorophyll **causes** plants to absorb light; and
f_4: Plants **use** light **to** produce carbohydrates via photosynthesis;

Above, the words in bold font indicate two of several standard relations (**"use** *something* **to"** and **"causes"**) that are used in conclusions of inference rules. f_3 and f_4 are again not directly available in the KB, but rather are obtained by single-premise inferences from respective KB facts

f'_3: Chlorophyll is used for absorbing light energy by plants; and
f'_4: Photosynthesis is a source of food for the plant by converting carbon dioxide, water, and sunlight into carbohydrates.

Some of their reasoning examples (in the Appendix of the paper) are reminiscent of natural logic, if one takes generic nouns to be like universally quantified ones. (E.g., h

could be interpreted as *"All plants use some chlorophyll to produce some sugar"*, which allows inferences by substitution of a more specific term for *"plant"* and a more general terms for *"chlorophyll"* and *"sugar"*.) So this seems to be an interesting approach to extracting informal inference rules from generic facts, enabling inference of other generic facts.