Neural-Network-Based Relation Inference

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 "shallow" natural language processing, such as speech recognition, tagging words with their parts of speech (POS), sentiment analysis, named entity recognition (NER – i.e., identifying names like *Genesee River, Lady Gaga, O'Hare International Airport, etc.*, in text), constituent and dependency parsing, machine translation, and more. Tasks requiring actual understanding, such as conducting meaningful conversations or understanding the intentions behind the actions of characters in a story, 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. Also text summarization is an active area that intuitively is dependent on some degree of understanding of the text being summarized.

These notes are about a noteworthy 2013 paper from Stanford about relational inference (like the one about Pablo Picasso); and very briefly about a paper on inferential question answering, based on brief, artificial "stories".

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 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 this is really a neural net made up of standard NN units – there must be layers of units that do the multiplications.

So how could these layers be implemented, using only standard units resembling those of mammalian brains? Well, this questions 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. As long as we're using operations, including (smoothed) threshold operations, that lead to a differentiable overall input-output behavior, 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.

They 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).

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 final note, the 2018 and 2019 NIPS conferences featured such papers as the following (2019 had over 1000 papers, very few on reasoning):

Puneet Agrawal et al., "A Deep Learning Based Conversational Social Agent" (2018)

Schlag & Schmidhuber, "Learning to Reason with Third Order Tensor Products" (2018)

Robert Ness, Kaushal Paneri, & Olga Vitek, "Integrating mechanistic and structural causal models enables counterfactual inference in complex systems" (2019)

Wang-Zhou Dai, Qiuling Xu, Yang Yu, & Zhi-Hua Zhou, "Bridging Machine Learning and Logical Reasoning by Abductive Learning" (2019)

Vaishak Belle & Brendan Juba, "Implicitly learning to reason in first-order logic" (2019).

Drew Hudson & Christopher Manning, "Learning by Abstraction: The Neural State Machine for Visual Reasoning" (2019)

Meng Qu & Jian Tang, "Probabilistic Logic Neural Networks for Reasoning" (2019)

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.