DEEP LEARNING FOR NLP CS 224D

LECTURE #16

JUNE 1, 2015

LECTURER: RICHARD SOCHER SCRIBE: KIRAN VODRAHALLI

Contents

1 Introduction

The last lecture is on Dynamic Memory Networks. We will look at an overview, and then introduce a new model developed at MetaMind called Dynamic Memory Networks.

2 Course Summary

In this course, we studied the following NLP tasks.

1. Sequence labeling - NER (named entity recognition), POS (part of speech tagging).
2. Classification - sentiment: any label you can assign to a piece of text.
3. structure prediction, syntactic parsing - understanding syntactic structure of a sentence.
4. machine translation - hard task, much larger and complex
5. question answering - memory networks

Then we had a bunch of techniques that we could combine and use in order to try to solve these tasks.

1. Word vectors
2. word vector averages
3. window based models
4. conv nets
5. tree based: Recursive NN
6. memory networks
7. sequence based: RNN (gated, LSTM)

If you payed close attention, there were hand designed features in the memory networks (Weston et. al.) from Facebook.

3 A New Paradigm

All tasks can be reduced to question answering. Every task is essentially a question answering task. It generally covers semantic questions (where was Obama's wife born?). Machine translation: "What is the translation into french". Sequence modeling: "what are the named entities in the sentence?".

Classification problems: "what is the sentiment?"

Even multi-sentence joint cliassficaiton problems like coreference resolution: who does "they" refer to?

All these tasks are essentially question-answering problems. So NLP -¿ QA. Interesting but useless? At first.

Yes, we could reduce these things, but does it help you?

4 Dynamic Memory Network

CAn you train any QA task with raw text and question-answer pairs? This is what the dynamic memory model tends to do.

We are producing a current paper: "Ask Me Anything: Dynamic Memory Networks for NLP".

These tasks kind of span the convex hull of NLP. So what is the DMN?

You have a semantic memory that holds word vectors and knowledge basis which feeds into an episodic memory. There is an input text sequence wich feeds into the semantic memory and back into the input text. The input text feeds into episodic memory, which feeds to an answer. The question also influences the arrows from input text sequence to episodic memory, and question feeds into answer and episodic memory directly as well.

Episodic and semantic memories need to be incorporated. For a lot of these different pieces, we have the more basic lego pieces to put them together. The sequence will be an input of texts, which will learn general facts about concepts. From time to time you have a question about concepts, and we will have gates which will trigger specific combinations of memories. From the episodic memory, you will produce an answer (which has to reason over a bunch of different facts).

If you never show it a specific type of question, it will not figure it out necessarily.

5 Module 1: Input

You have this to read input corpus: responsible for computing representations of audio, visual, or textual input. We only make one assumption: temporal sequence where you index each timestep such that they can be retrieved when needed later. Assume a temporal sequence indexable by a time stamp. For written language we have a sequence of words $(v_1, ..v_{T_w})$. Word vectors are considered unsupervised, which is the context for each word. This is important since the unsupervised part which gives us word vectors, which is then stored in semantic memory. We also need supervised learning: Ideally, the input module is independent. But in the end, you need to take derivatives and do backprop and send error messages in the input module. We also need context-independent (word vectors are what they are) and context-dependent (complex facts) hidden states.

We compute word vectors with Glove and store them in a semantic memory module, which is essentially a GRU (gated recurrent unit in recurrent networks). You can keep around the last memory, or update it.

$h_t = GRU(x_t, h_{t-1})$. Same equations but different weights dependeing on what sequence youre in (input sequence, question sequence, answer sequence).

$\tilde{h}_t = \tanh(Wx_t + r_t \circ Uh_{t-1} + b^{(h)})$ $h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t$.

How does this model generalize to different inputs. At every time step, you just need to have a network at that time step. Convolutional nets give you a representation too at each time step. You could break up an image into blocks, and assume that each type has its own vector representation.

This is end-to-end trainable, which is necessary to be able to apply backprop.

6 Semantic Memory

Word vectors are where we get the representations from. We also use knowledge bases! How do we translate a bunch of facts in a database into a vector space? See Socher et al 2013: Reasoning with Neural Tensor Networks for Knowledge Base Completion. We can basically take inputs (as word vectors) - for every relationship you have, you have a specific type of neural network. And for every word you have, you have a word vector. When you try to push a specific fact into a word vector, you give word vector as input into neural network at a specific location. Basically you train a neural network for EACH RELATION. You use max margin loss to train this. Essentially you want to maximize the score for triplets. It's very easy to add additional facts.

Another way of doing it: each relationship is a vector - you could pipe them for a very deep neural network. Each relation is a function- it is interesting that you can actually memorize a database by projecting into a vector space.

Is the knowledgebase fixed? Aspirationally, no. But in the current implementation, you stick in a bunch of city names. It helps you a lot here. It doesn't always help you, but it can. In the future, you would like to dynamically update this based on the input sequence. Feedback and feedforward in both directions.

The triplets are an entity, a relation, and another entity. You train one model for all the types questions at once. How does the model scale as more facts come?- it increases the size of the dimensions based on more entity vectors. This is a fresh new model - over time, I think this can be extended a lot, and say "as I read more, I increased model capacity over time".

LOOK UP ALL INSTANCES WHERE MAX MARGIN LOSS IS USED AND REPLACE WITH CONVEX OBJECTIVE THAT I DEFINED.

lists of things for training supervised manner seems somehow brittle - how much can you actually do to win new facts that are relevant? Ask after.

Questions require you to output full sequence. In this setting also, relationships change over episodes - your question is different depending on which facts are relevant.

7 Question module

A simple GRU over question word vectors. $q_t = GRU(v_t, q_{t-1})$. These weights are shared with the input sequence. We have a vector v_t from input, and q_t from question.

8 Episodic Memory

Most novel, most complex. It combines the previous three modules outputs in order to then reason over them and give the resulting knowledge to the answer module.

On a high level, it dynamically retrieves necessary information over the words or sentences. You might want to look at every word, or you might want to look at sentences over the entire corpus. You need more facts and you have to go over your input again. You have to iterate over inputs multiple times. For transitive inference, this part is in hippocampus, which is the seat of episodic memory in humans, and is active during this kind of inference and distruption of the hippocampus.

How do we compute sepecific gaets and the inmportance of specific facts.

Sentence vector s and a current memory vector m, and it is initialized to question vector. We have a bunch of comparisons between question and sentence vectors.

$z(s, m, q) = [s \circ q, s \circ m, |s - q|, |s - m|, s, m, q, s^T W^{(b)} q, s^T W^{(b)} m]$

Then we have a gating function which puts this all together and gives a score:

$G(s, m, q) = \sigma\left(W^{(2)} \tanh\left(z(s, m, q) + b^{(1)}\right) + b^{(2)}\right)$.

Then you summarize important facts in an episode vector: $e^1 = \sum_{t=1}^{T} \text{softmax}(g_t^1) s_t$. The g_t are the gates. The set of sentences in an episode summarizes the facts from the sentences that are important. When we ask another question, we compute a second set of gates - this second question is the new episode. The episode vector from the previous episode is an input for the second gate - a memory vector that is the summary of the first memory state and the second memory state.

We use another GRU to compute new gates:

$g_t^2 = GRU(s_t, m^1, q)$. Then we have a GRU over the memories. You can think of this as a very deep hierarchical GRU.

$m^1 = GRU(e^1, m^0)$. You update every time. And recall that m^0 is initialized to q. The reason you do this is because the memory updates: you remember what the question was, then you have to remember new facts to update the question.

To remember: each unit is a sum over weighted sentences. The sentences themselves come from the input vector, which was a GRU.

You can build hierarchical GRUs over any subset of entities that you want. Assign a GRU to a word, and then the representation of a paragraph is the final state after you have iterated your GRU over all the words in the paragraph. Then, you could do this for each paragraph and get a bunch of paragraph vectors trained by GRUs. These could be a new sequence of inputs, each which gets its own GRU. You could iterate and combine over these as well.

GRUs gave similar performance to LSTM, but they are easier to implement.

How do you know when to stop? Sometimes it figures it out, sometimes there are a certain number of facts. So you can supervise this. You can wait until the answer is good, and then stop it. And use that as supervision, to train a classifier in advance to know when to stop. It depends on whether or not you have supervision on the gates.

How do you handle things like ”he was in the water. is he wet?” - you could train based on cause and effect triplets. The main problem is you do not have enough data to train it on. You need the number of examples. Is there any way of choosing your training data appropriately. Once you have enough training data - things become easy.

Episodic memory module gives you a final output memory.

9 Answer Module

Also a GRU, predicts an output at every step using softmax, this output becomes the next input again. That vector allows you to predict an EOS token and stop.

Answering module has two abilities: answer at end for semantic questions, or allow an answer to be output at every element of the memory?

10 Training

We use cross-entropy error and backpropagation. You start deltas in the answer and input text sequence, and they backpropagate to word vectors.

Memories are interesting. They outperform the memory networks. Memory is a necessary but not sufficient condition for doing question answering. Once you can do it on a simple question, it does not mean you can answer more questions.

Sequence to Sequence modeling seems to be a special case of this model. Deep LSTM on encoder and decoder.

So hopefully QA-DMN can become CNN of NLP. It is much trickier in NLP who do not talk as much to each other. There are a lot of different subcommunities. To show the potential of deep learning, we wanted to create one model that can solve all kinds of tasks.

You have to add new word vectors and update all the time. You take as large a corpus as you can find, and train word vectors. Then you take smaller supervised sets, and propagate on top of the word vectors.

You could still do a lot of stuff regarding semantic memory for instance. Maybe you want to circumvent input sequence and go directly to knowledge base. Maybe you want to speed things up with hierarchical softmax.

11 Question and Answer Session

How do you do sentiment analysis?

You could use episodic memory to turn on gates that have stronger sentiment. Model could learn to focus on words that matter more for specific type of question.

How do you come up with z?

lots of trial and error, some intuition. Memory should change over time - leads to difference. So it matters how similar is to input.

What is next?

Improve this model, try it on other tasks that it hasn't been tried on.