Self-learning improvement by means of cloud computing

Crina Narcisa DEAC¹, Gicu Călin DEAC¹, Costel Emil Cotet¹, Mihalache GHINEA¹
¹University POLITEHNICA of Bucharest, Splaiul Independentei nr. 313, Sector 6, Bucharest, Tel. 021.402.9302, 021.402.9520, Fax: 021.310.7753:

Abstract: This paper describes some results of authors' research in machine reading at scale as a support for self-learning, which combines the challenges of document retrieval (finding the relevant articles) with that of machine comprehension of text (identifying the answer spans from those articles). Our approach combines a search component based on bigram hashing and TF-IDF (term frequency–inverse document frequency) matching with a multi-layer recurrent neural network model trained to detect answers in Wikipedia paragraphs.

Keywords: self-learning, NLP, machine learning

JEL classification: I29

Introduction

Education is the product of two distinct types of work: teaching and learning. To teach effectively, educators need to expand and enrich their learning experiences to make them accessible and memorable. To learn, students need the ability to access and understand information about their own passions and interests, using them as personally as possible. Educators can now embrace the possibilities introduced by the cognitive age to customize learning experiences and cultivate knowledge, to support learners as individuals despite the large number and diverse needs of students.

We must change the paradigm from "I need to learn" to "I want to know!"

The skills of learning must extend beyond traditional school days, to a lifetime of constant change and self-reinvention when workplace studies predict working adults will change jobs many times.

Cost barriers of devices, storage, content and data transmission have fallen. The accessibility of cloud and mobile devices has brought down infrastructure barriers. The
adoption barriers are disappearing as more educators and administrators catch up with student trends of mobile and social usage.

The proliferation of technology has created more data and at the same time has made it more accessible; educators just need the tools to put it working to shape more personalized learning. Provide students with the tools needed to take ownership of their learning journeys and set their own goals. Leadbeater (2008) [1] argues that the successful reinvention of educational systems worldwide depends on transforming pedagogy and redesigning learning tasks. Promoting learner autonomy and creativity is part of the solution. Technologies can be used to support efforts to transform pedagogy, but it is essential to recognize that twenty-first century learning experiences must incorporate more than just technology. Leadbeater also emphasizes that learning strategies for this century will not be limited to school, but will also encompass learning through peers, inter-generational partnerships and community relationships. Learning may take place outside of school in libraries, museums, community centers, local businesses or nearby farms, among others. Both Robinson (2006)[2] and Leadbeater (2008)[1] maintain that, ultimately, the idea of school as the sole provider of learning needs to be radically transformed.

Twenty-first century education will require more personalized learning with an emphasis on supporting rather than stifling creativity. Redecker et al. (2011)[3] stress that “personalization has implications for what, how and where we teach”. Personalization occurs through collaboration, provides for more rapid sharing of innovation and good practice, and quickly captures information about learners’ aptitudes and progress. Personalized learning is not an ‘add-on’ but a different way to undertake educational endeavors and includes peer-to-peer self-organized learning (Leadbeater, 2008).

1. Cloud application as a support for self-learning

To support self-learning, a cloud application based on Questions and Answers model could be an essential tool. The student can find a lot of useful information in a simple way of asking, without the need of complex search on the internet, filtering lot of results to find the best matching. But such an application it is a true challenge considering the large amount of data that must be indexed, the speed of querying, and the relevance of automatic filtering which must be based on NLP.

Natural Language Processing (NLP) is ability of machines to understand and interpret human language the way it is written or spoken. The objective of NLP is to make computer/machines as intelligent as human beings in understanding language [8]. The goal of NLP is to the fill the gap how the humans communicate (natural language) and what the computer understands (machine language).

There are three different levels of linguistic analysis done before performing NLP:

- Syntax - What part of given text is grammatically true.
- Semantics - What is the meaning of given text?
- Pragmatics - What is the purpose of the text?
NLP deal with different aspects of language such as Phonology - systematic organization of sounds in language, Morphology - study of words formation and their relationship with each other.

For understanding semantic analysis NLP use different approaches:

- **Distributional** - It employs large-scale statistical tactics of Machine Learning and Deep Learning.
- **Frame-Based** - The sentences which are syntactically different but semantically same are represented inside data structure (frame) for the stereotyped situation.
- **Theoretical** - This approach is based on the idea that sentences refer to the real word (the sky is blue) and parts of the sentence can be combined to represent whole meaning.
- **Interactive Learning** - It involves pragmatic approach and user is responsible for teaching the computer to learn the language step by step in an interactive learning environment.

**There are few Deep Learning Algorithms used in NLP:**

**Neural Network - NN (feed)**
- Part-of-speech Tagging
- Tokenization
- Named Entity Recognition
- Intent Extraction

**Recurrent Neural Networks - RNN**
- Machine Translation
- Question Answering System
- Image Captioning

**Recursive Neural Networks**
- Parsing sentences
- Sentiment Analysis
- Paraphrase detection
- Relation Classification
- Object detection

**Convolutional Neural Network - CNN**
- Sentence/ Text classification
- Relation extraction and classification
- Spam detection
- Categorization of search queries
- Semantic relation extraction
There are a number of highly developed full pipeline QA approaches using either the Web, as does QuASE, or Wikipedia as a resource, as do Microsoft’s AskMSR, IBM’s DeepQA and YodaQA that relies on data redundancy rather than sophisticated linguistic analyses of either questions or candidate answers, and it does not focus on machine comprehension, as we do.

We propose a cloud assistant application which can answer to user questions. The application consists in a large database of articles from a multitude of domains, a NLP (natural language processing) module and a document reader and retriever module. This task of machine reading at scale combines the challenges of document retrieval (finding the relevant articles) with that of machine comprehension of text (identifying the answer spans from those articles).

Figure 1 shows the architecture of the application. The user begins by asking the question in natural language from web interface (fig 2.). The application then parses the question using a natural language parser which constructs a tree of the question's phrasal structure.

The parse tree is given to the classifier which determines the type of answer to expect. Next, the query formulator uses the parse tree to translate the question into a series of search engine queries. These queries are issued in parallel to the search engine, which fetches the Wiki data base for each query.

Our approach combines a search component based on bigram hashing and TF-IDF (term frequency–inverse document frequency) matching with a multi-layer recurrent neural network model trained to detect answers in Wikipedia paragraphs (fig.1).
The setup of this system is to design a feature vector \( FP,Q(e) \) for each candidate entity \( e \), and to learn a weight vector \( \theta \) such that the correct answer \( (a) \) is expected to rank higher than all other candidate entities (similar to Wang et al., 2015 [9]):

\[
\theta^T FP,Q(a) > \theta^T FP,Q(e), \forall e \in E \cap \mathcal{P} \setminus \{a\}
\]

We use the following feature templates:
1. Whether entity \( e \) occurs in the passage.
2. Whether entity \( e \) occurs in the question.
3. The frequency of entity \( e \) in the passage.
4. The first position of occurrence of entity \( e \) in the passage.
5. n-gram exact match: whether there is an exact match between the text surrounding the place holder and the text surrounding entity \( e \). (left-right matching)
6. Word distance: we align the placeholder with each occurrence of entity \( e \), and compute the average minimum distance of each non-stop question word from the entity in the passage.
7. Sentence co-occurrence: whether entity \( e \) co-occurs with another entity or verb that appears in the question, in some sentence of the passage.
8. Dependency parse match: we dependency parse both the question and all the sentences in the passage, and extract an indicator feature.

The answer extraction module extracts relevant snippets called summaries from the articles (based on bigram hashing and TF-IDF), and generates a list of possible candidate answers from these snippets. These candidates are given to the answer selector for scoring and ranking (recursive neural network); the final answer (based on maximum score) is displayed to the user, presented in the context of their respective summary.

The Document Reader component is implemented using a 3-layer bidirectional LSTMs (Long Short-Term Memory) with \( h=128 \) hidden units for both paragraph and question encoding applying the tokenizers (Stanford CoreNLP and Spacy), and generating lemma, part of speech, and entity tags. First all the words are mapped to \( d \)-dimensional vectors via an embedding matrix \( E \in \mathbb{R}^{d \times |V|} \);

therefore we have \( p: p_1, \ldots, p_m \in \mathbb{R}^d \) and \( q : q_1, \ldots, q_l \in \mathbb{R}^d \). Next, we use a shallow bi-directional recurrent neural network (RNN) with hidden size \( h=128 \) to encode contextual embeddings \( p_i \) of each word in the passage,

\[
\begin{align*}
h_{i} &= \text{RNN}(h_{i-1}, p_i), i = 1, \ldots, m \\
h_{i} &= \text{RNN}(h_{i+1}, p_i), i = m, \ldots, 1
\end{align*}
\]

and \( p_i = \text{concat}(h_{i}, h_{i}) \in \mathbb{R}^h \), where \( h = 128 \)

We use another bi-directional RNN to map the question \( q_1, \ldots, q_l \), \( q \in \mathbb{R}^h \).

The goal in this step is to compare the question embedding and all the contextual embeddings, and select the pieces of information that are relevant to the question.
We compute a probability distribution $\alpha$ depending on the degree of relevance between word $p_i$ (in its context) and the question and then produce an output vector $o$ which is a weighted combination of all contextual embeddings

$$\alpha_i = \text{softmax}_i q^T W s p_i a o = \sum_i \alpha_i p_i$$

Prediction: Using the output vector $o$, the system outputs the most likely answer using: $a = \arg\max_{a \in p \cap E W a} o$ Finally, the system adds a softmax function on top of $W o$ and adopts a negative log likelihood objective for training. To make scores compatible across paragraphs in one or several retrieved documents, we use the unnormalized exponential and take argmax over all considered paragraphs pans for the final prediction.

We use the following process for each question / answer pair to build our training set.

run Document Retriever on the question to retrieve the top 5 Wikipedia articles.

All paragraphs from those articles without an exact match of the known answer are directly discarded.

All paragraphs shorter than 25 or longer than 1500 characters are also filtered out.

If any named entities are detected in the question, we remove any paragraph that does not contain them at all.

For every remaining paragraph in each retrieved page, we score all positions that match an answer using unigram and bigram overlap between the question and a 20-token window, keeping up to the top 5 paragraphs with the highest overlaps.

If there is no paragraph with non-zero overlap, the example is discarded; otherwise we add each found pair to our DS training dataset.

The application is written in python, using PyTorch with CUDA support. We use a dump of English Wikipedia, processed with WikiExtractor and filtered for internal disambiguation, index, list and outline pages and stored in an SQLite database. We retain over 5 million articles consisting of 9 million unique uncased token types. As NLP tokenizer we can use Stanford CoreNLP[5], spaCy [6], RegexpTokenizer or SimpleTokenizer [7].

As a future work we want to integrate voice recognition and voice synthesizers to create a better user experience.
Figure 2. Example of answers provided by the application (in web interface terminal mode):

Conclusion

Using Pytorch with CUDA support we achieve a great speed of data processing on a very large database (over 5 million Wikipedia articles), using parallel computing. The article database can contain any collection of documents from different knowledge domains and the answers are returned almost in real time.

This kind of application could be a real support for self-learning, and can be extended when will be deployed to the cloud with an historian for questions and it can provide based on this historic data, recommendations and statistics.
References


