SOMETHING SOMETHING SOMETHING...

Lukas Nabergall

April 15, 2017

1 Idea Overview

The idea is to use state-of-the-art language models for a natural language (e.g. English) and a programming language (e.g. C++ or Python) to augment a sequence-to-sequence model with the goal of learning to generate source code from natural language descriptions of programming problems.

Formally, let $\mathcal{X}$ and $\mathcal{Y}$ be the space of natural language descriptions and source code modules on alphabets $\Sigma_{\mathcal{X}}$ and $\Sigma_{\mathcal{Y}}$, respectively. The goal is to find a function $f : \mathcal{X} \to \mathcal{Y}'$ that minimizes $C(f(X), Y)$ for all $X = (x_1, \ldots, x_n) \in \mathcal{X}$ and $Y = (y_1, \ldots, y_m) \in \mathcal{Y}$, where $C : \mathcal{Y}' \times \mathcal{Y} \to \mathbb{R}$ is some cost function and $\mathcal{Y}'$ is the space of "artificial" source code modules.

We use two deep "branching" character-level long short-term memory networks N_1 and N_2 trained with hierarchical transfer learning to learn language models for natural languages and a programming languages. At each time step, N_1 processes an input character sequence $(c_{i_1}, \ldots, c_{i_k})$ for a word w_i using a character-level convolutional neural network and outputs $p(w_i|w_1, \ldots, w_{i-1})$ using a softmax output layer. Each "branch" at the i-th layer in the network takes as input the hidden state from a designated branch in the $(i-1)$-th layer. The i-th layer processes input text at the i-th level of language diversity: all text (e.g. natural languages of a given alphabet) is processed by the single branch in the first layer, then text in each language (e.g. English, French, Spanish, German, Swedish, Dutch, etc.) is processed by a distinct branch ins the second layer, text in each language style or domain[1] (e.g. journalism, third-person fiction, academia, blogs, Tweets, law, etc.) is processed by a distinct branch in the third layer, and finally text written by each author (Shakespeare, Trump, John Green, James Madison, etc.) is processed by a distinct branch in the fourth layer. Similarly, the network N_2 acts as a language model for programming languages.

Word vectors for a given natural language and programming language are derived from the columns of a weight matrix found in the first layer of N_1 and N_2.

2 Language Modeling via Hierarchical Transfer Learning: Overview

The basic idea is to use a "massively hierarchical" transfer learning approach to learn better (general) language models with faster training during adaptation to new domains. We use a deep neural network (of some type) as the model. Each layer (or adjacent group of layers)

1. This may be further divided—should all academic writing be processed in a single layer, or should different disciplines by processed in different branches?

of the network corresponds to a different level in the language hierarchy: the first layer (or group of layers) is learned on a very large dataset containing a large number of samples from every available language type, of every available language, of every available linguistic style, etc.; the second layer is specific to a given language type (and trained on the corresponding restricted data set); the third layer is specific to a given language; the fourth layer is specific to a given linguistic style; etc. The learning task is (something like) to predict the next character (and/or word) in a sequence.

Three reasons to think transfering from such different languages and, in particular, from such different language types, will be (at least a little) beneficial:

- At a high level, even seemingly vastly different language types, e.g. natural languages and programming languages, do have some common features—separation into "words", similarly structured whitespace and punctuation, some intersections in vocabulary, etc. In particular, a programming language and a natural language are more similar than a programming/natural language and a randomly sampled sequence of characters.
- We can typically describe a given program using natural language. Although this often assumes a significant amount of background knowledge, it already points to large overlaps in the expressible meanings between the two language types, indicating the existence of some shared features.
- It is likely (although currently unverified) that when a human reads and writes, say, natural language and programming language, they are using some common parts of the brain. Although the intersection between the parts of the brain used for each of the tasks may be small, this also strongly suggests the existence of common features between the two language types (in particular, that a neural network should likely be able to learn high-level representations useful to modeling/processing both language types).

3 Problem Ideas

Some Rough Ideas:

1. Recognizing/generating "comprehensible" natural language
2. Recognizing/generating/classifying any "structured" language (generally, English, Chinese, HTML, C, LaTeX, Morse, etc.)
3. Converting between linguistic styles (e.g. academic $\leftrightarrow$ Trump $\leftrightarrow$ Shakespeare $\leftrightarrow$ Twitter)
4. Correcting informal/sloppy/incorrect natural language (or, generally, correcting any "structured" language) — more specifically, domain-specific natural language correction (e.g. correcting text from/for Twitter, Wikipedia, a Trump speech, a Senate bill; more generally, HTML, C, LaTeX, Morse, Chinese, etc.)

4 Related Work

1. Plank, What to do about non-standard (or non-canonical) language in NLP, (2016).
2. Jozefowicz, Vinyals, Schuster, Shazeer, Wu, Exploring the Limits of Language Modeling, (2016).
3. Zhang, Liu, Wang, Zhu, Neural Personalized Response Generation as Domain Adaptation, (2017).
4. Wang and Zheng, Domain Adaptation of Recurrent Neural Networks for Natural Language Understanding, (2015).
5. Wang and Zheng, Transfer Learning for Speech and Language Processing, (2015).
6. Yoon, Yun, Kim, Park, Jung, Efficient Transfer Learning Schemes for Personalized Language Modeling using Recurrent Neural Network, (2017).
7. Yosinski, Clune, Bengio, Lipson, How transferable are features in deep neural networks, (2014).
8. Shazeer, Mirgoseini, Maziarz, Davis, Le, Hinton, Dean, Outrageously large neural networks: The Sparsely-Gated Mixture-of-Experts Layer, (2017).
9. Dam, Tran, Pham, A deep language model for software code, (2016).
10. Ruder, Ghaffari, Breslin, Character-level and Multi-channel Convolutional Neural Networks for Large-scale Authorship Attribution, (2016).
11. Kim, Jernite, Sontag, Rush, Character-aware Neural Language Models, (2015).
12. Krause, Lu, Murray, Renals, Multiplicative LSTM for sequence modelling, (2016).
13. Wu et al., Google's Neural Machine Translation System - Bridging the Gap between Human and Machine Translation, (2016).
14. Jozefowicz, Zaremba, Sutskever, An Empirical Exploration of Recurrent Network Architectures, (2015).
15. Olah and Carter, Attention and Augmented Recurrent Neural Networks, (2016)

5 Long short-term memory

6 Model

Task

- Develop a single character-level language model for many different "specific" languages[2] with simultaneous training and the ability to quickly transfer learn other specific languages.
- Formally, given a sequence of words $w_1, \ldots, w_n$ with character representations $w_i = (c_{i_1}, \ldots, c_{i_k})$ and a 4-tuple (L_1, L_2, L_3, L_4) (specifying, respectively, the language type, language, language style, and author of the word sequence), the language model estimates joint probabilities using the chain rule:

$$p(w_1, \ldots, w_n) = p(w_1) \prod_{i=2}^{n} p(w_i \mid w_1, \ldots, w_{i-1}).$$

Base Model

- A "branching" 5-layer long short-term memory network which, at each time step, processes an input character sequence $(c_{i_1}, \ldots, c_{i_k})$ for a word w_i using a character-level convolutional neural network and outputs $p(w_i \mid w_1, \ldots, w_{i-1})$ using a softmax output layer.
- Each "branch" at the i-th layer in the network takes as input the hidden state from the $(i-1)$-th layer. The i-th layer processes input text at the i-th level of language diversity: all text is processed by the single branch in the first layer, then text in each language type is processed by a distinct branch in the second layer, text in each language is processed by a distinct branch in the third layer, text in each language style is processed by a distinct branch in the fourth layer, and finally text written by each author is processed by a distinct branch in the fifth layer.

Possible Model Modifications/Improvements

- Connections between additional previous time steps to increase the network's ability to learn very long-term dependencies—instead of only passing in the cell state and hidden state from the previous time step to the current time step, we pass in the cell state and/or hidden state of a subset of all the previous timesteps (e.g. the previous n time steps). We can view this as a kind of attention mechanism.
- Residual connections between layers to improve training of the deep LSTM—instead of passing in the hidden state to the next layer, the hidden state plus this layer's input is passed to the next layer (e.g. [13]). Related to the above, we may also implement residual connection across time (e.g. [**??**]).
- Initialize the bias of the forget gate to $\geqslant 1$ to help prevent the vanishing gradient problem [14].
-
-

2. Essentially defined as a subspace of a language (which itself is the space of "all possible" texts in that language) with a given style and written by a single author.

Data

- There are many different language spaces[3], types of languages, languages, language styles, and authors—in particular, there are thousands (or more) in the last three categories. For tractability, we restrict to a small number of the most common from each category. On the other hand, to maximize the benefits of simultaneous "transfer" learning, we will likely want to use at least 3 elements from each category at each language diversity level. The most important considerations when choosing language types, languages, etc. are diversity and "popularity", both of which should be maximized.
- For the language space, we choose only one, and there are two possibilities: (1) Roman alphabet languages[4], (2) all languages expressible in unicode. For language type, we will work with the 3 most relevant and commonly used:[5] natural language, programming language, and markup language. For languages, there are in general many possible choices per language type—perhaps: English, French, and German (natural language);[6] C, Python, and Lisp (programming language);[7] HTML, LaTeX, YAML/Markdown/ASN.1/RTF (markup language).
-
-

Training

-
-
-
-

Testing/Evaluation

-
-
-
-

3. That is, for a given alphabet.

4. We may restrict this to be languages that use the alphabet used by Western European and American languages, or consider more broadly all Roman alphabet languages (which would include many more letters and languages).

5. Probably.

6. Other possibilities: Spanish, Portuguese, Danish, Swedish, Norwegian, etc.

7. Other possibilities: Java, C++, C#, Fortran, JavaScript, Pascal, Perl, Ruby, Visual Basic, Go, etc.