solve::words

applying(machine_learning) to:
automaticallySolve(math_word_problems)

Rationale

For a long time, a goal for researchers has been to automatically solve mathematical word problems (Upadhyay & Chang, 2017). Contained within the text of most mathematical word problems is all the information needed to solve it, meaning they can easily be solved without understanding the full implications of what words mean. Still, they combine both a task that is easy for humans, natural language parsing and understanding as well as advanced mathematics, and act as a measure of artificial intelligence (Clark & Etzioni, 2016).

A current standard for automatic word problem solvers advocates solving word problems by deriving templates and solving them (Upadhyay & Chang, 2017). Template equations are limited in scope, and can only solve equations similar to those it's already seen - it's not smart enough to do multi-step problems. Another published method, called ARRIS, uses verb categorization (Hosseini, Hajishirzi, Etzioni & Kushman 2014). Verb categorization methods fall prey to errors related to parsing errors, set completion, irrelevant information, and entailment, reaching accuaracy of only 70% in perfect scenarios (Hosseini et. al 2014).

New research shows that it's possible to parse sentences with 92.1% accuracy (Kong, Alberti, Andor, Bogatyy, & Weiss, 2017) by using Sequence to Sequence models (Suskever, Vinyals & Le, 2017). I propose an algorithm that uses a sequence to sequence model, based on these recent advances in machine learning technology, to extract equations from word problems.

The results of this research could be generalized to enhance personal assistants like Apple's Siri or the Google Assistant.

Materials

The model was built with Keras (Chollet F., 2015) and using the Tensorflow backend (Abadi et al., 2015).

The training and validation dataset is a mix of problems from MAWPS (Koncel-Kedziorski et al., 2016) and Dolphin 18k (Huang, Shi, Lin, Yin, & Ma, 2016) and consists of about 5,000 problems when cleaned. The testing dataset is from Upadhyay S. & Change, 2016. All datasets are cleaned to only contain problems with correct equations that use numbers from the text.

Due to the small size of publically available word problem corpuses and the high cost of generating and labeling new ones, the model takes spaCy (Honnibal & Montani, 2017) word vectors as input to lessen training data requirements.

The model was trained on Google's ML Engine - a public cloud platform for training neural networks built with TensorFlow.

Sympy is used to solve the generated equations, an open-source Python computer algebra system (Meurer et al., 2017).

The source of the algorithm is written in Python 2.7

All graphs and visuals created by the researcher using TensorBoard and Matplotlib (Hunter, 2007)

References

Chollet, F., & others. (2015). Keras. GitHub. Retrieved from https://github.com/keras-team/keras

Clark, P., Etzioni, O., Khot, T., Sabharwal, A., Tafjord, O., Turney, P. D., & Khashabi, D. (2016). Combining Retrieval, Statistics, and Inference to Answer Elementary Science Questions. In AAAI (pp. 2580–2586).

Honnibal, M., & Montani, I. (2017). spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To Appear.

Hosseini, M. J., Hajishirzi, H., Etzioni, O., & Kushman, N. (2014). Learning to solve arithmetic word problems with verb categorization. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP) (pp. 523–533).

Huang, D., Shi, S., Lin, C.-Y., Yin, J., & Ma, W.-Y. (2016). How well do computers solve math word problems? Large-scale Dataset construction and evaluation. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers) (Vol. 1, pp. 887–896).

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. Computing In Science & Engineering, 9(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

Koncel-Kedziorski, R., Roy, S., Amini, A., Kushman, N., & Hajishirzi, H. (2016). MAWPS: A math word problem repository. In Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (pp. 1152–1157).

Kushman, N., Artzi, Y., Zettlemoyer, L., & Barzilay, R. (2014). Learning to automatically solve algebra word problems. In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers) (Vol. 1, pp. 271–281).

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, … Xiaoqiang Zheng. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Retrieved from https://www.tensorflow.org/

Meurer, A., Smith, C. P., Paprocki, M., Čertík, O., Kirpichev, S. B., Rocklin, M., … Scopatz, A. (2017). SymPy: symbolic computing in Python. PeerJ Computer Science, 3, e103. https://doi.org/10.7717/peerj-cs.103

Upadhyay, S., & Chang, M.-W. (2016). Annotating Derivations: A New Evaluation Strategy and Dataset for Algebra Word Problems. CoRR, abs/1609.07197. Retrieved from http://arxiv.org/abs/1609.07197

Design Process & Explanation

The initial design for Solve Words was a rule-based process. It involved parsing a sentence using a natural language parser, classifying each token to create a tree ("classification"), then converting that tree through a series of defined steps to "reify" the tree into an equation ready to use with a computer algebra system ("reification"). This proved problematic: English is not static, even with nicely written word problems. Attempting to take a static route from tree to an equation is almost impossible to do well.

To address this difficulty the design shifted to combine the classification and reification stages using a neural network. Sequence to sequence models work well for machine translation (Wu et al., 2016), thus the design of the network is based on a sequence to sequence model.

The model consists of three sections: the encoder, decoder, and attention. The encoder section creates a tensor containing a representation of each word's use in the equation. The attention section generates a score for each word's importance, which is used to compute a weighted average of the encoded representation for each output. The decoder decodes the attention and outputs a set of values, each corresponding to either an operation, a variable, or a number from the input sentence.

Instead of writing equations in standard notation, a machine-friendly prefix notation is used instead. It reduces errors by assigning numerical indexes to variables in equations and can be represented with high information density.

Because of limitations on available data, the model is only able to generate equations with a length of fewer than 70 characters, using single-token numbers contained in the first 71 words of the problem. Pre-trained word vectors from SpaCy are used as inputs, cutting data requirements.

Results

The best model scored 94.09% per-character accuracy on its testing dataset of problems. This translates to generating correct equations for 25.74% of the problems in the model's training dataset and 12.34% of problems outside of the model's training dataset.

Figure 1: A demonstration of attention

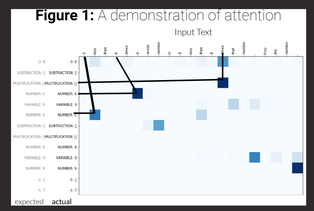
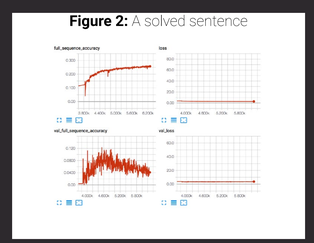


Figure 2: A solved sentence



Conclusion

Computers struggle with understanding word problems. When compared to other models on real-world problems, SolveWords achieved 12.35% accuracy. In Huang et al. (2016), the highest accuracy solver scored about 15.9%, using a similarity-based model that may fail outside the scope of problems on which it has been trained. Since SolveWords processes the entire sentence, it should be more accurate tested against problems unlike those it has seen before compared to other models, but a lack of data makes it hard to verify this.

SolveWords achieved testing scores of 25.74% (and could obtain more with training time), an indicator that the model fits word problems well, but "overfits" (failed to generalize) on training data because of a lack of samples. In the future, it should be possible to achieve higher accuracy with this model by simply gathering more data to train with, ideally at least 30,000 problems (which was impossible due to budget constraints). Still, using limited publically available datasets, SolveWords is a state-of-the-art model for solving word problems.

Model Design

II Attention

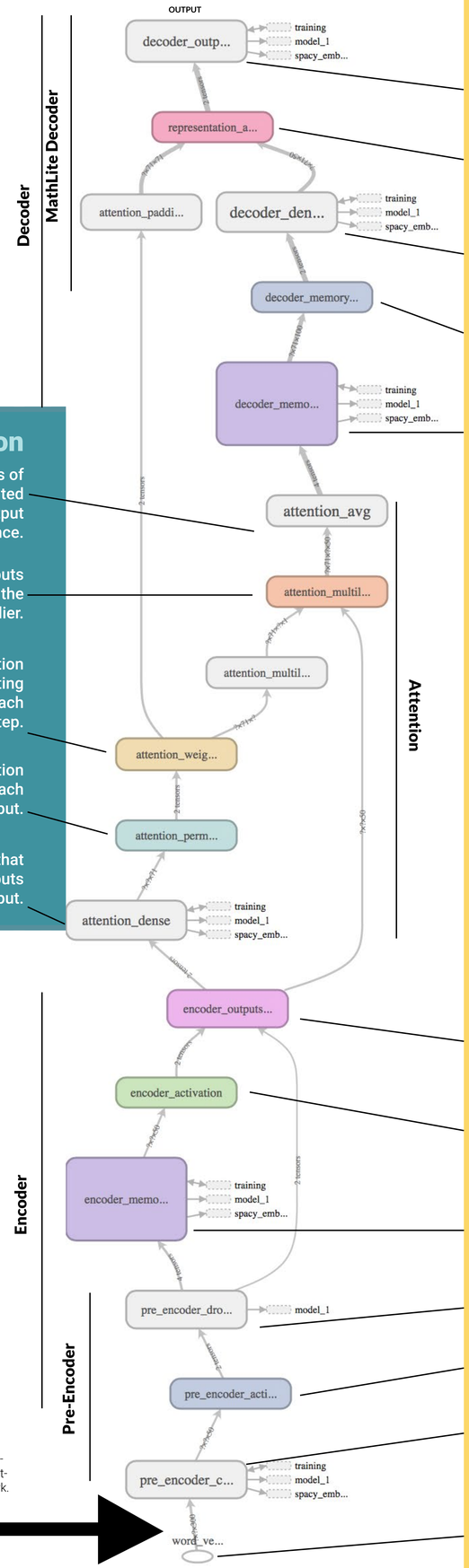
12) Averages the input axis of the encoder outputs weighted by attention creating an output sequence.

11) Multiplies the encoder outputs (a sequence over time) by the attention weights from earlier.

10) Applies a softmax function to the attention tensor, creating weights that sum to one for each output timestep.

9) Permutes ("pivots") the attention tensor to show how much each output relies on each input.

8) A densely connected layer that predicts how important the inputs will be for each output.



III Decoder

17) A final, magical layer that predicts each part of the equation using a softmax activation function.

16) Concatenates the attention and decoded representation of the equation together to help the final layer reference important numbers in the problem.

15) A densely-connected layer, probably used to create a representation of the equation in MathLite.

14) A dropout layer used during training to prevent memorization by randomly dropping parts of the tensor.

13) An LSTM that receives the attention as input for each timestep, and probably creates a universal representation of the equation.

I Encoder

7) Used during training to prevent memorization by randomly dropping parts of the encoder's outputs.

6) A tanh activation function, which turns the encoder's output into a positive gentle curve.

5) An LSTM memory layer that allows the model to understand the problem as a whole, probably allowing the model to learn to reference variables together.

4) Used during training to prevent memorization by randomly dropping parts of the vectors.

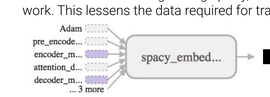
3) Applies ReLu activation to the model, a non-linear function inspired by biology.

2) Applies a 3-word long filter over the vectors, allowing it to recognize simple phrases and entities like numbers and variables.

1)Takes in a sequence of word vectors, basically sequences of "meaning" that represent each word in the problem.

INPUTS

Words are first converted into vectors - numerical representations of word meaning - using spaCy, then fed into the network. This lessens the data required for training the network.



Layer Types

Dense

A layer consisting of neurons connected to all of the neurons in the layer before it.

LSTM

A layer similar to a dense layer that can remember state across timesteps.

Convolutional

A layer consisting of filters applied to limited groups of nuerons in the previous layer. This allows for pattern matching.