Solving probability questions with neural networks

Simon Šuster

June 4, 2019

1 Data

We split the annotated data into train (80) / dev (10) / test (10%) parts. Some statistics are reported in Table 1.

n train	1720
n dev	217
n test	214
n toks in dataset (problems)	100205
vocab size (lower)	4686
avg sen len	47

Table 1: Data statistics.

2 End-to-end models

Here, we are interested in reading/encoding the text given by the math problem with a sequence model, and then based on the encoded vector for the entire text, predict the answering probability directly, without taking into account in any way the (intermediate) formal statements. This should serve as a sanity check, as we would not expect to obtain particularly high scores with end-to-end learning given the difficulty of the task and the small size of the dataset.

Broadly speaking, we can approach end-to-end modeling in two ways, by treating the task as a regression problem with clearly defined bounds $[0, 1]$; or as a multi-class classification problem, where we first transform the continuous output space of possible answers into discrete labels, e.g. $f : [0, 1] \rightarrow \{0, 1, 2, 3, 4\}$, where the labels in the set represent 5 different bins. Next, I will first present some experiments and results using discretization, and then move on to regression. The reason I am also exploring discretized labels is that learning is much easier than in the regression case where it is thought to be more brittle (due to difficulty optimizing with the L2 squared norm loss).

2.1 A discretized model

Our encoder is a 50-dimensional 1-layer LSTM, with a multi-class predictor (softmax) on the decoder side, which approximates the true probabilities given as answers in the dataset. For example, using discretization into 2 bins, the network learns to predict whether the answering probability is less or more than roughly 0.5[1] We run experiments with different number of bins ("resolution"). In training, we use the cross-entropy loss which is standard in multi-class problems.

An alternative way would be to train a binary classifier *for each bin* (with a single sigmoidal unit at output), and accept as answer the prediction of that classifier which is the most confident.

2.2 A continuous-output (regression) model

We have a single node at the network output instead of a softmax, and train with the mean squared error. We tried both sigmoid on top of the output node and without any activation function, but found no real difference in results.

[1]The exact value depends on the discretization technique used, for example the width-based or the quantile-based techniques.

2.3 Other possible models

We also implement a baseline where the answer at test time is the answer from that training instance whose similarity to the test instance is the greatest. The training and test instances are vectorized with pretrained 50-dimensional embeddings. For a test instance i, its nearest neighbor in the training set is found:

$$\textit{near-neigh(i)} = \underset{n \in N}{\operatorname{argmax}} \cos\left(\frac{1}{|C_n|} \sum_{j \in C_n} c_j, \frac{1}{|Q|} \sum_{k \in Q} q_k\right), \tag{1}$$

where $c, q \in R^d$, the multiset C_n contains all words in the training instance n, and Q contains all words of the current test instance.

Try a BertForTokenClassification model, where we fine-tune the pretrained general-domain BERT (contextual language model) on our data.

A character-based sequential decoder could be used as many problem answers are stated in its "long" form (e.g. 1/6, which might be easier to relate to the parts in the text). In our experiments at this stage, all long forms are evaluated into a float.

3 Text-to-statement models

lstm-enc-pointer-dec [Provide a description of the architecture.]

We extend our LSTM-encoding approach from above by adding a decoder on top that can point to a position in the passage containing the answer (Vinyals et al., 2015b). We train and evaluate for each of the predicates separately. For example, for the *group()* predicate the goal is to predict the position of the predicate argument. Since *group()* only accepts a single argument, we constrain the length of the sequence to be decoded to 1. We adapt the decoding length depending on the predicate in question. Note that this approach only works for arguments which point (or can be easily resolved to point) to a position in the text.

We include three baselines. The first simply selects a random index from the passage as prediction (**random baseline**). The second looks only at the first sentence (where most *group* predicates are introduced), and chooses an index of a random noun (based on the PoS tag assigned by CoreNLP). We call it **PoS random baseline**. Thirdly (**sampling baseline**), we obtain a pointer using a sampling procedure. We first create a probability distribution over words indicated by the pointer in the training set. Then, given a non-uniform random sample of the size of the training set, we iteratively try to find a match for the word from the sample in the test instance. When we have a hit, we use that word's index as the predicted pointer.

Ideas for future work:

- add PoS tag feature to LSTM, e.g. by appending the one-hot encoded representation to word embeddings (Chen et al., 2017)
- treat as labeling problem with seq2seq (like in relation extraction)
- baseline: syntactic trees as input to the network. Cf. tree encoder (Tai et al., 2015)
- an LSTM with character or token-based decoding (Vinyals et al., 2015c; Wang et al., 2017)
- number and entity mapping to variables for reducing sparsity
- coarse-to-fine decoding similar to Dong and Lapata (2018) where simplified statements are predicted first, and the slots filled after, perhaps using a pointer network
- output constraints (Wang et al., 2017)

3.1 Data augmentation, common sense knowledge

See the dataset released by Saxton et al. (2019), which contains automatically generated questions, including probability questions. The problems in their case are much more impoverished, but perhaps there are some interesting ideas there.

Try to separate the common-sense knowledge added by the annotators from the problem (use the original problem where available). This could allow us to study adding common sense in an initial experiment. Judging on a few examples, dice sides and numbers are often added, as well as card ranks. But things like "even number" are not explained.

4 Results

The reported results in Table 2 were obtained by training the models for 30 epochs with early stopping based on the performance on the development set. We train each model 10 times and report the average performance. The general outcome is that all models consistently outperform the simple baselines, but they are by itself not very good. For example, when classifying into 20 bins, a random baseline gets 1 example correct out of 20, whereas our model gets 3 correct.

We also try out initializing the word embeddings with the embeddings pretrained on news texts, but this shows little effect.

Model	accuracy
random baseline	
2 bins	0.5
5 bins	0.2
10 bins	0.1
20 bins	0.05
lstm-enc-discrete-dec	
2 bins	0.659
5 bins	0.375
10 bins	0.215
20 bins	0.152
2 bins +pretrained	0.667
5 bins +pretrained	0.369
10 bins +pretrained	0.232
20 bins +pretrained	0.160

Model	MAE
random baseline	0.349
nearest-neigh	0.307
avg-prob	0.247
lstm-enc-regression-dec	0.206
+sigmoid	0.206
+2 layers	0.206
+2 layers, 0.5 dropout	0.204

Table 2: Direct prediction of answer (without solver). Results on the test set. For the discrete task, accuracy is reported, and for the continuous task, mean absolute error (smaller is better).

Model	accuracy
random baseline for group(y)	0.038
PoS random baseline for group(y)	0.260
sampling baseline for group(y)	0.452
lstm-enc-pointer-dec + embs	
group(y)	0.650
+ PoS-d10	0.669
take	
(y1, ·, ·)	0.720
(y1, y2, ·)	0.607
(y1, y2, n)	0.557
+ PoS-d10	0.598
(·, ·, n)	0.845
+ PoS-d10	0.829
take_wr	
(y1, ·, ·)	0.598
(y1, y2, ·)	0.436
take(y1, ·, ·) & take_wr(y1, ·, ·)	0.648
take(y1, y2, n) & take_wr(y1, y2, n)	0.584
+ PoS-d10	0.626

Table 3: Prediction of statements. Results on the test set, averaged over 5 runs.

Evaluate some of the methods on probability questions from Saxton et al. (2019).

4.1 Evaluation methods

Per-predicate accuracy See https://github.com/PietroTotis/nlp4plp/blob/master/evaluate/eval.py

Execution accuracy

4.2 Error analysis with examples

Some examples we get right in the discrete setup (20 bins) are listed below. We currently don't have a way of knowing *why* we get those right, or which we get right only by chance. (We could try to find those examples which we get right in different bin setups and at the same time not far away from the true answer in the regression setup.)

bin range: (0.188, 0.231)
id: h56
there are 14 marbles in a bag : 3 red , 5 blue and 6 white . if one marble is picked at random ; what is the probability to pick a red marble ?

bin range: (0.5, 0.533)
id: l411
a 6-sided die , labeled 1 , 2 , 3 , 4 , 5 and 6 , is rolled , find the probability that an even number is obtained . bin: (0.33987590900000003, 0.4)

bin range: (0.083, 0.115)
id: m917
a box contains 22 red bows , 19 pink bows and 11 saffron bows . what is the probability of drawing a red bow first and then a saffron bow with replacement ?

5 Other related work

- Generalization of seq2seq models to sets as either inputs or outputs. Show that better to impose some order (e.g. alphabetical) on elements which should be permutation invariant. Show a simple technique for choosing an optimal ordering at output by maximizing the LL over different ordering via sampling etc. (Vinyals et al., 2015a).
- At output, we seek an unordered set of constraints yet with dependence between them, cf. the nl2sql work by Xu et al. (2017).

References

D. Chen, A. Fisch, J. Weston, and A. Bordes. Reading wikipedia to answer open-domain questions. *arXiv preprint arXiv:1704.00051*, 2017.

L. Dong and M. Lapata. Coarse-to-fine decoding for neural semantic parsing. *arXiv preprint arXiv:1805.04793*, 2018.

D. Saxton, E. Grefenstette, F. Hill, and P. Kohli. Analysing mathematical reasoning abilities of neural models. In *ICLR*, 2019.

K. S. Tai, R. Socher, and C. D. Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

O. Vinyals, S. Bengio, and M. Kudlur. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*, 2015a.

O. Vinyals, M. Fortunato, and N. Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700, 2015b.

O. Vinyals, Ł. Kaiser, T. Koo, S. Petrov, I. Sutskever, and G. Hinton. Grammar as a foreign language. In *Advances in neural information processing systems*, pages 2773–2781, 2015c.

Y. Wang, X. Liu, and S. Shi. Deep neural solver for math word problems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 845–854, Copenhagen, Denmark, Sept. 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1088. URL `http://www.aclweb.org/anthology/D17-1088`.

X. Xu, C. Liu, and D. Song. Sqlnet: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436*, 2017.