## Semparse Bakeoff

Kelsey Josund / CS224U / Spring 2018



A few people got 100% (denotation accuracy 1.0) Certain initializations work better. Start making "guesses" about what each symbol was by eliminating alternatives from lexical rules and keeping the guess if the accuracies remained high.

For example, after adding 'if w == 'glarc' and i != 0: continue' to eliminate adding lexical rules other than glarc == 1, oracle accuracy only decrease by ~1% and training and dev accuracy consistently stayed above 20%. After doing this a few times, train accuracy maxed out ~58%, but oracle accuracy dropped to ~90%.

Added some new unary functions lambda x: x and lambda x: abs(x). These improved the oracle accuracy back up to 100%, and the training model favored abs(x). With these added, dev accuracy jumped up to ~90%.

There were issues with cases like 3-4-5 being interpreted as 3-(4-5), so selecting for how "left-leaning" a tree is to prioritize earlier operations first when all else is equal helps. This was computed recursively by counting nodes and adding 2x priority to left nodes. I also added a count of total number of nodes. I also added the operator precedence feature.

Train 96%, dev 98%. Just by locking in the rest of the feature rules (scincs = 4, sherle = 5, fribbs = 2, volms = 3, kugns = 1 and sniese=+, sklofg=-,thouch=~,scwokt=abs), I could get train 100%, dev 100% (it could learn precedence on its own given complete lexical information, but not with partial).

Added and modified operator\_precedence\_features to traverse the tree and add features (op1, op2), (op1), (op2) whenever op1 appears lower in the tree than op2, using the default sgd with hinge loss for optimization.

The missing operator was the unary 'abs' (absolute value) and that '\*' (multiplication) did not occur in the dataset.

I started with ArithmeticDomain.operator\_precedence\_features.

Looking at misclassified samples I noticed that many of the mistakes made by the model was predicting the wrong "depth" of application for unary operators, where applying the unary operator at a shallower depth would have resulted in the correct denotation.

A second feature function looked at the maximum depth of any child of an operator and counted the number of times an operator appears at a certain depth.

This boosted accuracy from 0.88 to 0.96

Optimization settings: T=20, loss='log', l2\_penalty=0.0001, eta=0.005

For a feature function, we used the provided featurizer along with the suggested operator precedence featurizer.

We used the provided optimization hyperparameters except we used a learning rate of 0.3 rather than 0.5.

We also recognized that there were only two binary and two unary ops in the training set, and through ablative analysis we were able to identify addition, subtraction, negation, and absolute value as the most likely candidates for these four ops.

We removed all other ops, allowing us to increase the maximum cell size while still allowing us to train our model in a reasonable amount of time.

Final denotation accuracy was 0.8.

Commonalities:
traversing the tree
studying errors very closely