Supplementary Materials for "Structured Generative Models of Natural Source Code"

EM Learning for Latent Traversal Variable LTTs

Here we describe EM learning of LTTs with latent traversal variables. Consider probability of α with deterministic traversal variables h_i^d and latent traversal variables h_i^l (where h_i represents the union of $\{h_i^l\}$ and h_i^d):

$$\sum_{h_{0:N}} \mathrm{p}(n_1, h_0) \prod_{i=1}^{N} \mathrm{p}(C_i \,|\, n_i, h_i)\, \mathrm{p}(h_i^l \,|\, h_{i-1}^l) \times \mathrm{p}(h_i^d \,|\, h_{0:i-1}, n_{1:i}, , \alpha_{1:t}) \tag{4}$$

Firstly, the $\mathrm{p}(h_i^d \,|\, \cdot)$ terms drop off because as above we can use the compiler to compute the AST from α then use the AST to deterministically fill in the only legal values for the h_i^d variables, which makes these terms always equal to 1. It then becomes clear that the sum can be computed using the forward-backward algorithm. For learning, we follow the standard EM formulation and lower bound the data log probability with a free energy of the following form (which for brevity drops the prior and entropy terms):

$$\sum_{i=2}^{N} \sum_{h_i^l, h_{i-1}^l} Q_{i,i-1}(h_i^l, h_{i-1}^l) \log P(h_i^l \,|\, h_{i-1}^l) + \sum_{i=1}^{N} \sum_{h_i^l} Q_i(h_i^l) \log \mathrm{p}(C_i \,|\, n_i, h_i) \tag{5}$$

In the E step, the Q's are updated optimally given the current parameters using the forward backward algorithm. In the M step, given Q's, the learning decomposes across productions. We represent the transition probabilities using a simple tabular representation and use stochastic gradient updates. For the emission terms, it is again straightforward to use standard log-bilinear model training. The only difference from the previous case is that there are now K training examples for each i, one for each possible value of h_i^l, which are weighted by their corresponding $Q_i(h_i^l)$. A simple way of handling this so that log-bilinear training methods can be used unmodified is to sample h_i^l values from the corresponding $Q_i(\cdot)$ distribution, then to add unweighted examples to the training set with h_i^l values being given their sampled value. This can then be seen as a stochastic incremental M step.

More Experimental Protocol Details

For all hyperparameters that were not validated over (such as minibatch size, scale of the random initializations, and learning rate), we chose a subsample of the training set and manually chose a setting that did best at optimizing the training log probabilities. For EM learning, we divided the data into databatches, which contained 10 full programs, ran forward-backward on the databatch, then created a set of minibatches on which to do an incremental M step using AdaGrad. All parameters were then held fixed throughout the experiments, with the exception that we re-optimized the parameters for the learning that required EM, and we scaled the learning rate when the latent dimension changed. Our code used properly vectorized Python for the gradient updates and a C++ implementation of the forward-backward algorithm but was otherwise not particularly optimized. Run times (on a single core) ranged from a few hours to a couple days.

Smoothed Model

In order to avoid assigning zero probability to the test set, we assumed knowledge of the set of all possible tokens, as well as all possible internal node types – information available in the Roslyn API. Nonetheless, because we specify distributions over tuples of children there are tuples in the test set with no support. Therefore we smooth every $\mathrm{p}(C_i \,|\, h_i, n_i)$ by mixing it with a default distribution $\mathrm{p}_{def}(C_i \,|\, h_i, n_i)$ over children that gives broad support.

$$\mathrm{p}_\pi(C_i \,|\, h_i, n_i) = \pi\, \mathrm{p}(C_i \,|\, h_i, n_i) + (1-\pi)\, \mathrm{p}_{def}(C_i \,|\, h_i, n_i) \tag{6}$$

For distributions whose children are all 1-tuples of tokens, the default model is an additively smoothed model of the empirical distribution of tokens in the train set. For other distributions we model the number of children in the tuple as a Poisson distribution, then model the identity of the children independently (smoothed additively).

This smoothing introduces trees other than the Roslyn AST with positive support. This opens up the possibility that there are multiple trees consistent with a given token sequence and we can no longer compute $\log \mathrm{p}(\alpha)$ in the manner discussed in Section 5. Still we report the log-probability of the AST, which is now a lower bound on $\log \mathrm{p}(\alpha)$.

Parent Kind	% Log prob	Count
(IdentifierToken, global)	30.1	17518
(IdentifierToken, local)	10.9	27600
Block	10.6	3556
NumericLiteralToken	4.3	8070
Argument	3.6	10004
PredefinedType	3.0	7890
IfStatement	2.9	2204
AssignExpression	2.4	2747
ExpressionStatement	2.1	4141
EqualsValueClause	2.0	3937
StringLiteralToken	1.9	680
AddExpression	1.9	1882
ForStatement	1.6	1759

Figure 9. Percent of log probability contributions coming from top parent kinds for LTT-HiSeq-Scope (50) model on test set.

```
for ( int i = 0 ; i < words . Length ; ++ i ) i = i . Replace ( "X" , i ) ;

for ( int j = 0 ; j < words . X ; j ++ ) {
 if ( j [ j ] == - 1 ) continue ;
 if ( words [ j ] != words [ j ] ) j += thisMincost ( 1 ) ;
 else {
   j = ( j + 1 ) % 2 ;
   words [ j + 1 ] += words [ 0 ] ;
 }
}

for ( int j = words ; j < words . Pair ; ++ j )
 for ( int i = 0 ; i < words . Length ; ++ i ) {
   isUpper ( i , i ) ;
 }

for ( int i = 0 ; i < words . Length ; ++ i ) {
 words [ i , i ] = words . Replace ( "*" , i * 3 ) ;
}

for ( int j = 360 ; j < j ; ) {
 if ( ! words . ContainsKey ( j ) ) {
   if ( words . at + " " + j == j ) return ume ( j , j ) ;
 } else {
   j = 100 ;
 }
}

for ( int c = 0 ; c < c ; ++ c )
 for ( int i = 0 ; i < c ; i ++ ) {
   if ( ! words [ i ] ) i = i ;
 }

for ( int i = 0 ; i < words . Length ; i ++ ) {
 i . Parse ( i ) ;
}

for ( int i = words ; i < 360 ; ++ i ) {
 words [ i ] = words [ i ] ;
 i = 4 ;
}
```

Figure 10. More example `for` loops generated by LTT-HiSeq-Scope (50). Whitespace edited to improve readability.

```
using System ;
using System . Collections . Text ;
using System . Text . Text ;
using System . Text . Specialized ;
using kp . Specialized ;
using System . Specialized . Specialized ;

public class MaxFlow
{
    public string maximalCost(int[] a, int b)
    {
        int xs = 0;
        int board = 0;
        int x = a;
        double tot = 100;
        for (int i = 0; i < xs; i++) {
            x = Math.mask(x);
        }
        for (int j = 0; j < x; j++) {
            int res = 0;
            if (res > 0) ++res;
            if (res == x) {
                return -1;
            }
        }
        for (int i = 0; i < a.Length; i++) {
            if (a[i - 2].Substring(board, b, xs, b, i)[i] == 'B') x = "NO";
            else if (i == 3) {
                if (i > 1) {
                    x = x.Abs();
                }
                else if (a[i] == 'Y') return "NO";
            }
            for (int j = 0; j < board; j++) if (j > 0) board[i] = j.Parse(3);
        }
        long[] x = new int[a.Count];
        int[] dp = board;
        for (int k = 0; k < 37; k++) {
            if (x.Contains < x.Length) {
                dp[b] = 1000000;
                tot += 1;
            }
            if (x[k, k] < k + 2) {
                dp = tot;
            }
        }
        return "GREEN";
    }
}
```

Figure 11. Example CompilationUnit generated by LTT-HiSeq-Scope (50). Whitespace edited to improve readability.

```
using System ;
using System . sampling . Specialized ;

public class AlternatingLane
{
 public int count = 0 ;
 int dc = { 0 , 0 , 0 , 0 } ;
 double suma = count . Reverse ( count ) ;

 public int go ( int ID , int To , int grape , string [ ] next )
 {
  if ( suma == 1000 || next . StartsWith . To ( ID ) [ To ] == next [ To ] ) return ;
  if ( next [ next ] != - 1 ) {
   next [ To , next ] = 1010 ;
   Console . Add ( ) ;
  }
  for ( int i = 0 ; i < ( 1 << 10 ) ; i ++ ) {
   if ( i == dc ) NextPerm ( i ) ;
   else {
    count ++ ;
   }
  }
  return div ( next ) + 1 ;
 }

 string solve ( string [ ] board ) {
  if ( board [ count ] == '1' ) {
   return 10 ;
  }
  return suma ;
 }
}
```

Figure 12. Example CompilationUnit generated by LTT-HiSeq-Scope (50). Whitespace edited to improve readability.

```
using System ;
using System . Collections . Collections ;
using System . Collections . Text ;
using System . Text . Text ;

public class TheBlackJackDivTwo
{
 int dp = 2510 ;
 int vx = new int [ ] { - 1 , 1 , 1 , 2 } ;
 int [ ] i ;
 int cs ;
 double xy ;
 int pack2 ;

 long getlen ( char tree ) {
  return new bool [ 2 ] ;
 }

 int getdist ( int a , int splitCost )
 {
  if ( ( ( 1 << ( a + vx ) ) + splitCost . Length + vx ) != 0 )
    i = splitCost * 20 + splitCost + ( splitCost * ( splitCost [ a ] - i [ splitCost ] ) ) ;
  int total = i [ a ] ;
  for ( int i = 0 ; i < pack2 ; i ++ ) {
   if ( can ( 0 , i , a , i , i ) ) {
    total [ a ] = true ;
    break ;
   }
   i [ i ] -= i [ a ] ;
   total = Math . CompareTo ( ) ;
   saiki1 ( a , vx , a , i ) ;
  }
  return total + 1 < a && splitCost < a ;
 }
}
```

Figure 13. Example CompilationUnit generated by LTT-HiSeq-Scope (50). Whitespace edited to improve readability.