### The Problem

How can techniques drawn from machine learning be applied to the learning of structured, compositional representations, such as programs? The space of possible programs is enormous: how can we search through, or sample from, this space? I approach this problem in a multitask setting, and show that learning compressive representations in the form of programs suggests two complementary approaches to dimensionality reduction.

Concretely, the problem is to learn programs that evaluate to a given constant string. By learning compressive representations of these strings, one hopes that patterns within those strings will be extracted.

### The Algorithm

The algorithm maintains a set of observed programs $\Omega_n$ s.t. if $e_n \in \Omega_n$ then $e_n$ evaluates to $t_n$. It alternates between:

1. Improving an estimate of the MAP $G$ by approximately maximizing a lower bound on the log posterior, which is

$$
\log P(G|\{t_n\}) \geq \log P(G) + \sum_{n=1}^N \log \sum_{e \in \Omega_n} P(t_n|e)P(e|G)
$$

2. Finding more programs $e$ to put in $\Omega_n$ for which $P(e|G)$ is high. Programs are found either by sampling from $G$ using the Metropolis-Hastings, or performing a heuristic search over programs.

### Finding Programs

Programs are represented in Combinatory Logic. Semantics are given as rewrite rules, such as

- $S f g x \rightarrow (f x) (g x)$
- $K x y \rightarrow x$
- $I x \rightarrow x$

Writing the rewrite rules backwards gives rules for turning strings in to programs that produce that string. These “backward rules” can serve either as moves for a search procedure or a Metropolis-Hastings sampler. I tried a heuristic beam search as well as a Metropolis-Hastings sampler. Empirically the heuristic search gave better results quicker than the MCMC sampler.

### Model Fitting

Model fitting yields a grammar $G$ and distributions $P(e_n|t_n, G)$. Expected production counts under $P(e_n|t_n, G)$ give features for $t_n$. The dimensionality of the representation grows with the data, because $G$ is non-parametric. I trained the model on words containing the affixes “anti-” and “-tion.” Running PCA on the resulting feature vectors separated the affixes. Features could be used for data visualization or as input to a discriminative classifier, but they lack a clear generative interpretation.

### Symbolic Dimensionality Reduction

Symbolic dimensionality reduction extracts a compressive symbolic representation from a dataset, ideally one which is interpretable by humans. I used the algorithm to learn a single representative program, $d^*$, from a dataset of strings. $d^*$ is given by

$$
d^* = \arg \max_d \sum_{n=1}^N \log \sum_{a \in \Omega_n} P(a|G_0) + \log P(G_0)
$$

where $G_0$ is the base grammar. This objective function finds the $d^*$ that maximally compresses the data, which corresponds to MAP inference in the model given below.

Unsupervised pretraining is used to find programs, $\{\Omega_n\}_{n=1}^N$, that are likely to be mutually compressive. When given words starting with “anti-” and ending in a single character, this algorithm learned the program:

$$
(\text{LAMBDA} (C) (\text{APPEND} “\text{ANTI}” (\text{LIST} C)))
$$

(Scheme translation of original Combinatory Logic program.)

### References


### Future Work

These two approaches to dimensionality reduction also immediately suggest a clustering algorithm, wherein strings belong to the same cluster if they are sufficiently mutually compressive.

Other forms of structured data could be modeled. Protein primary sequences and DNA sequences might be interesting avenues of research.

Inductive synthesis of programs from input/output examples, as in Programming by Demonstration, could also be explored in this framework, similarly to [2].