Dimensionality Reduction via Program Induction

Kevin Ellis and **Eyal Dechter** and **Joshua B. Tenenbaum**
Department of Brain and Cognitive Sciences
Massachusetts Institute of Technology
{ellisk,edechter,jbt}mit.edu

Abstract

How can techniques drawn from machine learning be applied to the learning of structured, compositional representations? In this work, we adopt functional programs as our representation, and cast the problem of learning symbolic representations as a symbolic analog of dimensionality reduction. By placing program synthesis within a probabilistic machine learning framework, we are able to model the learning of some English inflectional morphology and solve a set of synthetic regression problems.

Introduction

Dimensionality reduction is a core machine learning task in which the objective is to discover a latent representation of a data set. This representation should have fewer degrees of freedom than those present in the original data, revealing a lower-dimensional subspace that spans the data.

Symbolic dimensionality reduction generalizes this task by replacing the lower-dimensional subspace with a program, and by replacing the latent representation of each datum with the arguments to that program. Instead of finding a representation with fewer continuous degrees of freedom, we aim to find the program minimizing the total description length of the data.

As a motivating example, consider a problem that nearly every child faces: that of acquiring the linguistic rules of his or her native language. During the first few years of life, children learn their language's rules for forming plurals, past tense, superlatives, past participles, and other forms of inflectional morphology (O'Donnell 2015). Although forming the English plural may seem simple to a native speaker, the regular rule actually consists of three different cases that the child must learn; depending upon phonetic properties of the end of the noun, a different sound is appended to it. Additionally, the child must identify a large set of irregular patterns, such as *ox→oxen*.

When confronted with linguistic data such as in Figure 1, children can, and computers ought to, explain this data by inducing the general morphophonological rule of the English plural. This is an example of symbolic dimensionality reduction: explaining data through the induction of a symbolic, program-like rule. Throughout this paper, we will use the English plural as a running example of symbolic dimensionality reduction.

Symbolic dimensionality reduction is not limited to explaining symbolic data. For example, consider an agent tasked with a set of related regression problems. For each regression problem, the agent has a set of sampled points from an unknown relation, as illustrated in Figure 2. Upon solving some of the regression problems, the agent could more compactly represent its solutions, and make better future predictions, by extracting a common symbolic structure from the relations.

/kæts/ ("cats")	/dɔrz/ ("doors")
/bajsɪkəlz/ ("bicycles")	/tiθ/ ("teeth")
/bɒlz/ ("balls")	/ɔrəndʒəz/ ("oranges")
/ʃuz/ ("shoes")	/bʊks/ ("books")
/hɔrsəz/ ("horses")	/ajz/ ("eyes")
/dagz/ ("dogs")	/blæŋkəts/ ("blankets")

Figure 1: Plural forms of nouns commonly spoken by children at thirty months of age (Dale and Fenson 1996), written in phonetic (IPA) form

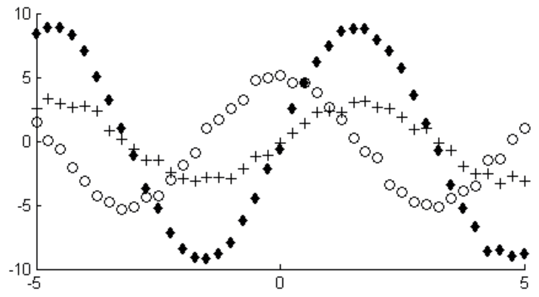


Figure 2: Three related regression problems. Each regression problem consists of a set of noisy samples from a sinusoidal function. Performing symbolic dimensionality reduction upon this data would yield a symbolic description of this class of functions; in our system, this corresponds to a program that outputs sinusoidal curves.

Representing Knowledge as Programs

Many of the representations most commonly employed in symbolic AI approaches – frames, grammars, production systems, etc – admit parsimonious representation as functional programs. Functional programs can also serve as the underlying representation of statistical machine learning models, as in probabilistic programming (Goodman et al. 2008).

In this work, we represent both morphophonological rules and real-valued functions in polymorphicly typed combinatory logic. This representation is a functional language with previous use in machine learning models of program synthesis (Dechter et al. 2013; Liang, Jordan, and Klein 2010).

Combinatory logic is a close cousin to lambda calculus, but it is simplified in some respects. In particular, the only means of combining two expressions in combinatory logic (written e_1, e_2) to produce a new one, is function application (written $(e_1\ e_2)$). Every expression in combinatory logic is also a program that outputs some value. We write $[\![e]\!]$ to indicate the value e evaluates to. We write $e : \tau$ to indicate that the expression e has the type τ. For example, the expressions in Figure 3 and Figure 4 have the types List(Phoneme) $\rightarrow$ List(Phoneme) and $\mathbb{R} \rightarrow \mathbb{R} \rightarrow \mathbb{R}$, respectively. For an overview of the typing system, see (Pierce 2002).

Figures 3 and 4 illustrate encodings of morphophonological rules and real-valued functions discovered by our system, respectively, translated from combinatory logic in to Scheme. Representations such as these are the output of symbolic dimensionality reduction.

```
(lambda (stem)
  (append stem
    (if (voiced?
          (last-one stem))
        ((cons /z/) null)
        ((cons /s/) null))))
```

Figure 3: Representation of a rule for the English plural. When given the stem of a word (such as /dag/, that is, "dog") this expression produces a plural form (such as /dagz/, that is, "dogs").

```
(lambda (a)
  (lambda (x)
    (* a (sin x))))
```

Figure 4: Representation of sinusoids with varying amplitude. When provided with a real valued coefficient a this expression produces a function that computes a sine wave with amplitude a.

Consider the example of an agent learning the English plural. This agent takes as input a set of words with the plural inflection (for example, Figure 1). If it discovers the representation in Figure 3, then the agent can write down an expression for each (regular) plural by applying the expression in Figure 3 to an expression that evaluates to the corresponding stem. This re-expression of the data (words, in the case of the plural) as a common expression applied to a small number of varying arguments (stems, in the case of the plural) is the primary operation of what we have called symbolic dimensionality reduction.

Inducing Programs from Data

Suppose our learner is given a set of N datums, $\{x_i\}_{i=1}^N$, such as words. Our objective is to find an expression d (such as those in Figures 3 and 4) allowing us to re-express each datum with a program, written $\{e_i\}_{i=1}^N$, satisfying $[\![(d\ e_i)]\!] = x_i$ (each e_i evaluating to the stem of a word in the case of the plural). Intuitively, d corresponds to the low dimensional subspace of dimensionality reduction, and each e_i indexes the location of x_i within that subspace.

Building on the work of (Liang, Jordan, and Klein 2010), we cast the problem of learning programs as a form of probabilistic inference in a hierarchical Bayesian model. Algorithm 1 specifies our model. Both d and $\{e_i\}_{i=1}^N$ are drawn according to a description length prior over expressions in typed combinatory logic as in (Dechter et al. 2013). This description length prior is specified by a stochastic grammar (G_0 in Algorithm 1) and depends upon the type of the expression, similar to the adaptor grammar approach of (Liang, Jordan, and Klein 2010).

The approach so far outlined assumes that every single datum may be expressed as the output of a single expression, d. However, in some domains, we may wish to tolerate exceptions. In the case of the English plural, these exceptions correspond to the irregular plurals, such as "teeth" (instead of "tooths"). To accomodate these exceptions, we introduce a parameter M corresponding to the probability that a given datum is expressed using d. Those not expressed using d are drawn from G_0.

In symbolic dimensionality reduction, we seek the MAP value of d. From Algorithm 1,

$$d^* = \arg\min_d \min_M \Bigg(-\ln P_{\cdot|G}(d|G_0) - \sum_i \ln \Bigg(M \sum_{\substack{e_i \\ [\![(d\ e_i)]\!]=x_i}} P_{\cdot|G}(e_i|G_0) + (1-M) \sum_{\substack{e_i \\ [\![e_i]\!]=x_i}} P_{\cdot|G}(e_i|G_0) \Bigg) \Bigg). \tag{1}$$

Approximate Inference

Exact minimization of Equation 1 is intractable, so we instead perform the sums in Equation 1 over a finite set of expressions. This finite set of expressions is enumerated from a proposal distribution learned during a pre-training phase. The proposal distribution, G, is represented by a nonparametric stochastic grammar over programs. This grammar is found by performing MAP inference within a generative model similar to that of (Liang, Jordan, and Klein 2010). We specify this model in Algorithm 2. Within Algorithm 2,

Algorithm 1 Generative model for symbolic dimensionality reduction

```
**Input:** Grammar $G_0$, $M \in [0,1]$
**Output:** Values $\{x_i\}_{i=1}^N$
$d \sim P_{\cdot|G}(\cdot|G_0)$
**for** $i = 1$ **to** $N$ **do**
    $e_i \sim P_{\cdot|G}(\cdot|G_0)$
    w.p. $M$,
        $x_i \leftarrow [\![(d\ e_i)]\!]$
    w.p. $(1-M)$,
        $x_i \leftarrow [\![e_i]\!]$
**end for**
```

Algorithm 2 A generative model similar to that used in (Liang, Jordan, and Klein 2010). We use a MAP estimate of G as a proposal distribution.

```
**Input:** Regularization parameter $\lambda$, smoothing parameter $\alpha$.
**Output:** Values $\{x_i\}_{i=1}^N$
$G \sim P_{G|\lambda,\alpha}(\cdot|\lambda,\alpha)$
**for** $i = 1$ **to** $N$ **do**
    $e_i \sim P_{\cdot|G}(\cdot|G)$
    $x_i \leftarrow [\![e_i]\!]$
**end for**
```

the parameter α corresponds to the number of pseudocounts identically to (Dechter et al. 2013). The parameter λ specifies a prior over the structure of G, namely,

$$P_{G|\lambda}(G|\lambda) \propto \exp\left(-\lambda(\text{\# productions in } G)\right). \quad (2)$$

We fit the value of G given $\{x_i\}_{i=1}^N$ using an EM algorithm (Dempster, Laird, and Rubin 1977). Given the factorization of the joint distribution induced by Algorithm 2, and treating $\{e_i\}_{i=1}^N$ as missing data, the EM updates are:

$$\begin{aligned} q_i(e_i) &\propto \mathbb{1}\Big[[\![e_i]\!] = x_i\Big] P_{\cdot|G}(e_i|G^{\text{old}}) \\ G^{\text{new}} &= \arg\min_G \Big(-\ln P_G(G) \\ &\quad - \sum_{i=1}^N \mathrm{E}_{q_i}\left[\ln P_{\cdot|G}(e_i|G)\right] \Big) \end{aligned} \quad (3)$$

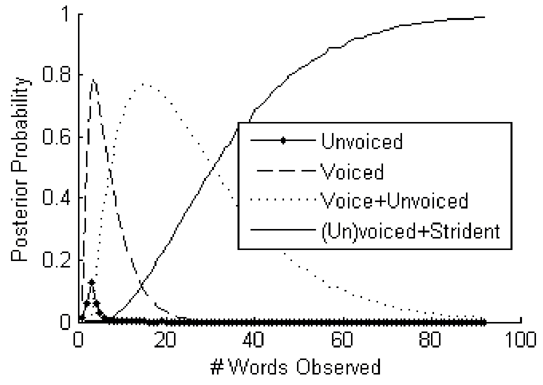
Exact computation of the normalizing constant and expectation in Equation 3 requires summing over the infinite space of expressions that G generates. To approximate this expectation, we enumerate the expressions with the highest prior likelihood under G, and take the corresponding sums only over those programs.

The expression for G^{new} in equation 3 has a natural interpretation as a maximally compressive grammar. Interpreting negative log probabilities as description lengths, this equation picks the grammar minimizing the sum of the description length of the grammar, plus the expected description length of the expressions found to evaluate to the $\{x_i\}_{i=1}^N$.

A slight generalization of the Neville-Manning algorithm (Nevill-Manning and Witten 1997) permits tractible approximate minimization in Equation 3. Concretely, an expression e is a member of G^{new} iff

$$\sum_{i=1}^N \mathrm{E}_{q_i}\left[c(e,e_i)\right] \geq \lambda \quad (4)$$

where $c(e, e_i)$ is the number of times that expression e occurs in e_i. After deciding which expressions to incorporate in to G^{new}, the production probabilities are estimated using the inside-outside algorithm (Lafferty 2000).



t]

Figure 5: Posterior probabilities of different morphophonological rules for the plural as the size of the input word list increases. (Un)voiced: Appends (un)voiced allophone. Voiced+Unvoiced: Conditional branch for voiced and unvoiced allophones. (Un)voiced+Strident: The correct linguistic rule for the regular English plural.

Experiments

English Inflectional Morphophonology

English possesses eight classes of inflectional affixes: the plural, past tense, past participle, genitive, 3rd person singular, progressive, comparative, and superlative. For each inflectional affix, we compiled a word list, drawn from (Dale and Fenson 1996), of words modified to use said affix. Each word was represented as a sequence of phonemes.

The G_0 parameter in the model included English phonemes, predicates upon phonological properties of the phones, and Lisp-style list manipulation primitives, such as `append` and `cons`. Our symbolic dimensionality reduction system was trained on each of these word lists individually. We included a separate preprocessing step that extracted stems of words and added them as primitives to the proposal distribution G.

Some inflectional rules, such as that of the superlative, modify all (regular) stems in the same manner. In these cases, our system recovered the correct regular inflection. For the superlative, this inflection consists of appending /əst/ to the stem.

Other inflectional rules, such as that of the plural, have multiple conditional branches. Our system recovered two out of the three conditional branches of the plural.

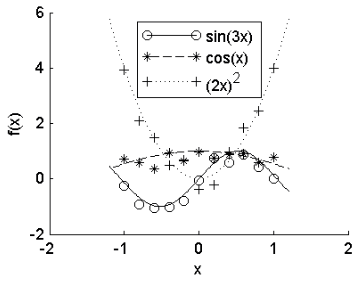
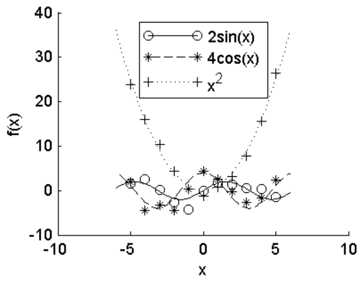
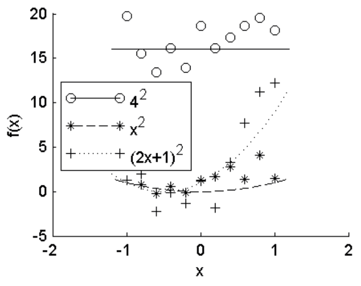
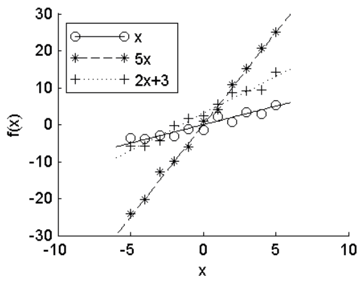
$f(x) = ...$	Example input data	Learned Expression
$g(ax)$, $a \in \{0, 1, ..., 9\}$, $g \in \{\sin, \cos, \text{square}\}$		$\lambda gax.g(a \times x)$
$a \cdot g(x)$, $a \in \{0, 1, ..., 9\}$, $g \in \{\sin, \cos, \text{square}\}$		$\lambda gax.a \times g(x)$
$(ax + b)^2$, $a \in \{0, 1, ..., 9\}$, $b \in \{0, 1, ..., 9\}$		$\lambda abx.(ax + b)^2$
$(ax + b)$, $a \in \{0, 1, ..., 9\}$, $b \in \{0, 1, ..., 9\}$		$\lambda abx.(ax + b)$

Figure 6: Learned representations in the form of expressions for different sets of functions. Expressions were translated from combinatory logic to lambda calculus. For each set of functions (left column), the system sees a set of noisy samples (circles/crosses/stars in middle column) of the function's value at predetermined points. From these noisy samples, it recovers the underlying functional forms of each curve (solid/dashed/dotted curves in middle column) and then forms a representation of this class of functions (right column).

Why does our system not recover the full regular rule for the plural inflection? The problem lies with the difficulty of the inference. When we provide our model with a correct expression for computing the plural, it achieves higher posterior probability than the expression found by our MAP search.

To demonstrate this preference of the model for the correct linguistic rule, we assembled four candidate rules for the plural inflection (d in the model Algorithm 1 specifies) of varying complexity. We varied the size of the input word list and computed (unnormalized) posterior probabilities for each of the four candidate rules. These posterior probabilities are plotted in Figure 5. The salient observation from this figure is that, as the word list grew in size, the system preferred more and more complicated morphophonological rules, converging upon the correct regular rule.

Synthetic Regression Problems

Suppose one encounters a set of regression problems all sharing some sort of structure – for example, all being polynomials of degree two or all being drawn from a Fourier basis. Application of symbolic dimensionality reduction to these regression problems should recover a program that maps the underlying degrees of freedom (eg, coefficients in the Fourier example) to the corresponding curve.

We augment the generative model in Algorithm 1 with a Gaussian noise model as follows: after evaluating the expression (eg, computing $[\![(d\ e_i)]\!]$ or $[\![e_i]\!]$) we apply said expression to a series of test points and compute the resulting real number, to which noise is added. Thus, the output of the generative model is no longer a set of expressions, but a set of vectors of real numbers.

We further augment this generative model to accommodate representations (d of Algorithm 1) of higher arity. For fixed arity A, and each datum x_i, a set of expressions $\{e_i^1, \cdots, e_i^A\}$ are drawn from G_0. Then, x_i is set to $[\![(d\ e_i^1 \cdots e_i^A)]\!]$. We consider $A = 1$ and $A = 2$.

We compiled four sets of functions, shown in Figure 6 along with their learned representations. For each set of functions, we evaluated each member of the set at 11 points and added Gaussian noise. These sampled points were provided to our system. The set of programs considered included primitives for sinusoids, addition, and multiplication. The system recovers a correct representation in these cases.

Discussion

This work introduces the problem of symbolic dimensionality reduction, which combines program synthesis with representation learning. One candidate for future research in this area is the application of modern program synthesis techniques to the problem of performing inference in our probabilistic model. Such techniques could permit the recovery of, for example, the correct regular rule for inflections with several conditionals.

Much prior work has been done on the use of programs as a representation for AI systems. Our approach draws primarily from (Dechter et al. 2013; Liang, Jordan, and Klein 2010), but also from Estimation of Distribution algorithms in the genetic programming literature (Poli, Langdon, and McPhee 2008) and from work in Cognitive Science termed logical dimensionality reduction (Katz et al. 2008).

What we have presented is not sufficient for the general purpose acquisition of morphophonological rules. Notably absent from our models is any form of semantics. The literature upon unsupervised learning of morphological and phonological rules is vast; see (O'Donnell 2015) for reviews of some of these models. But, we hope that models that use programs as a representation, such as ours, could be useful for language, nonlinguistic domains, and even continuous domains.

References

Dale, P. S., and Fenson, L. 1996. Lexical development norms for young children. *Behavioral Research Methods, Instruments, & Computers* 28:125–127.

Dechter, E.; Malmaud, J.; Adams, R. P.; and Tenenbaum, J. B. 2013. Bootstrap learning via modular concept discovery. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13, 1302–1309. AAAI Press.

Dempster, A. P.; Laird, M. N.; and Rubin, D. B. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 39:1–22.

Goodman, N. D.; Mansinghka, V. K.; Roy, D. M.; Bonawitz, K.; and Tenenbaum, J. B. 2008. Church: a language for generative models. In *UAI 2008, Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence, Helsinki, Finland, July 9-12, 2008*, 220–229.

Katz, Y.; Goodman, N. D.; Kersting, K.; Kemp, C.; and Tenenbaum, J. B. 2008. Modeling semantic cognition as logical dimensionality reduction. In *Conference of the Cognitive Science Society*, 71–76.

Lafferty, J. 2000. *A Derivation of the Inside-outside Algorithm from the EM Algorithm.* Research report. IBM T.J. Watson Research Center.

Liang, P.; Jordan, M. I.; and Klein, D. 2010. Learning programs: A hierarchical bayesian approach. In Fürnkranz, J., and Joachims, T., eds., *ICML*, 639–646. Omnipress.

Nevill-Manning, C. G., and Witten, I. H. 1997. Identifying hierarchical structure in sequences: A linear-time algorithm. *J. Artif. Int. Res.* 7(1):67–82.

O'Donnell, T. J. 2015. *Productivity and Reuse in Language: A Theory of Linguistic Computation and Storage*. The MIT Press.

Pierce, B. C. 2002. *Types and programming languages*. MIT Press.

Poli, R.; Langdon, W. B.; and McPhee, N. F. 2008. *A field guide to genetic programming*. Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk. (With contributions by J. R. Koza).