FROM AMORTISED TO MEMOISED INFERENCE: COMBINING WAKE-SLEEP AND VARIATIONAL-BAYES FOR UNSUPERVISED FEW-SHOT PROGRAM LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Given a large database of concepts but only one or a few examples of each, can we learn models for each concept that are not only generalisable, but interpretable? In this work, we aim to tackle this problem through hierarchical Bayesian program induction. We present a novel learning algorithm which can infer concepts as short, generative, stochastic programs, while learning a global prior over programs to improve generalisation and a recognition network for efficient inference. Our algorithm, *Wake-Sleep-Remember* (WSR), combines gradient learning for continuous parameters with neurally-guided search over programs. We show that WSR learns compelling latent programs in two tough symbolic domains: cellular automata and Gaussian process kernels. We also collect and evaluate on a new dataset, *Text-Concepts*, for discovering structured patterns in natural text data.

1 INTRODUCTION

A grand challenge for building more flexible AI is developing learning algorithms which quickly pick up a concept from just one or a few examples, yet still generalise well to new instances of that concept. In order to instill algorithms with the correct inductive biases, research in few-shot learning usually falls on a continuum between *model-driven* and *data-driven* approaches.

Model-driven approaches place explicit domain-knowledge directly into the learner, often as a stochastic *program* describing how concepts and their instances are produced. For example, we can model handwritten characters with a motor program that composes distinct pen strokes (Lake et al., 2015), or spoken words as sequences of phonemes which obey particular phonotactic constraints. Such representationally explicit models are highly interpretable and natural to compose together into larger systems, although it may be difficult to completely pre-specify the required inductive biases.

By contrast, *data-driven* approaches start with only minimal assumptions about a domain, and instead acquire the inductive biases themselves from a large background dataset. This is typified by recent work in deep meta-learning, such as the Neural Statistian (Edwards & Storkey (2016); see also Hewitt et al. (2018)), MAML (Finn et al. (2017); see also Reed et al. (2017)) and Prototypical Networks (Snell et al., 2017). Crucially, these models rely on *stochastic gradient descent* (SGD) for the meta-learning phase, as it is a highly scalable algorithm that applies easily to datasets with thousands of classes.

Ideally these approaches would not be exclusive – for many domains of AI we have access to large volumes of data and also rich domain knowledge, so we would like to utilise both. In practice, however, different algorithms are suited to each end of the continuum: SGD requires objectives to be differentiable, but explicit domain knowledge often introduces discrete latent variables, or *programs*. Thus, meta-learning from large datasets is often challenging in more explicit models.

In this work, we aim to bridge these two extremes: we learn concepts represented explicitly as *stochastic programs*, while meta-learning generative parameters and an inductive bias over programs from a large unlabelled dataset. We introduce a simple learning algorithm, *Wake-Sleep-Remember* (WSR), which combines SGD over continuous parameters with neurally-guided search over latent programs to maximize a variational objective, the evidence lower bound (*ELBo*).

In evaluating our algorithm, we also release a new dataset for few-shot concept learning in a highly-structured natural domain of short text patterns (see Table 1). This dataset contains 1500 concepts such as phone numbers, dates, email addresses and serial numbers, crawled from public GitHub repositories. Such concepts are easy for humans to learn using only a few examples, and are well described as short programs which compose discrete, interpretable parts. Thus, we see this as an excellent challenge domain for structured meta-learning and explainable AI.

TN	[4, 4]	Der.	21.8%	B,J,U	img25.png	$4,050	Day 6 pH 7.9	(715) 292-6400	Jul 10, 2012
NV	[1, 2]	Yoo.	7.5%	B,D,E	img19.png	$4,000	Day 2 pH 7.9	(262) 949-7411	Nov 16, 2004
AZ	[3, 4]	Ade.	-10.1%	A	img42.jpg	$340	Day 10 pH 7.0	(715) 416-2942	Nov 1, 2000
ID	[2, 3]	Mik.	22.4%	A,B,J,U	img18.png	$102	Day 4 pH 7.4	(715) 983-2293	Jun 9, 2016
WY	[1, 1]	Won.	-4.0%	B,E	img43.jpg	$1,200	Day 10 pH 7.4	(920) 848-8885	Nov 1, 1999
...	...	...	...	...	...	...	...	...	...

Table 1: Examples from 10 of the 1500 classes contained in the *Text-Concepts* dataset. The full dataset can be found at [redacted for anonymous review]

2 Background: Helmholtz Machines and Variational Bayes

Suppose we wish to learn generative models of spoken words unsupervised, using a large set of audio recordings. We may aim to include domain knowledge that words are built up from different short phonemes, without defining in advance exactly what the kinds of phoneme are, or exactly which phonemes occur in each recording. This means that, in order to learn a good model of words in general, we must also infer the particular latent phoneme sequence that generated each recording.

This latent sequence must be re-estimated whenever the global model is updated, which itself can be a hard computational problem. To avoid a costly learning 'inner-loop', a longstanding idea in machine learning is to train two distinct models simultaneously: a *generative model* which describes the joint distribution of latent phonemes and sounds, and a *recognition model* which allows phonemes to be inferred quickly from data. These two models together are often called a *Helmholtz Machine* (Dayan et al., 1995).

Formally, algorithms for training a Helmholtz Machine are typically motivated by Variational Bayes. Suppose we wish to learn a generative model $p(z, x)$, which is a joint distribution over latent variables z and observations x, alongside a recognition model $q(z; x)$, which is a distribution over latent variables conditional on observations. It can be shown that the marginal likelihood of each observation is bounded below by

$$\log p(x) \geq \log p(x) - \mathrm{D}_{\mathrm{KL}}[q(z;x)||p(z|x)] \tag{1}$$

$$= \mathop{\mathbb{E}}_{z\sim q(z;x)} \log p(x|z) - \mathrm{D}_{\mathrm{KL}}[q(z;x)||p(z)] \tag{2}$$

where $\mathrm{D}_{\mathrm{KL}}[q(z;x)||p(z|x)]$ is the KL divergence from the true posterior $p(z|x)$ to the recognition model's approximate posterior $q(z; x)$. Learning a Helmholtz machine is then framed as maximisation of this *evidence lower bound* (or *ELBo*), which provides the shared basis for two historically distinct approaches to learning.

2.1 The Wake-Sleep Algorithm

The first method, proposed by Hinton et al., is an *alternating optimisation* algorithm: alternate between updates to the generative model p and recognition model q. The update for $p(x|z)$, called the 'wake' phase, can be derived simply from Eq. 2 as:

Wake phase: Maximise $\mathbb{E}[\log p(x|z)]$ of observed data x using inferred latent variables $z \sim q(z; x)$

Unfortunately, the exact update for $q(z; x)$, which is minimisation of $\mathrm{D}_{\mathrm{KL}}[q(z;x)||p(z|x)]$, cannot be computed since it requires knowledge of the true posterior $p(z|x)$ on observed data. Instead the update for q is approximated using the 'other' KL divergence, $\mathrm{D}_{\mathrm{KL}}[p(z|x)||q(z;x)]$, by 'dreaming' training data from the generative model:

Sleep phase: Maximise $\mathbb{E}[\log q(z; x)]$ using latents $z \sim p(z)$ and hallucinations $x \sim p(x|z)$

This approximation will be accurate if the recognition model is able to very well match the true posterior (and so both KL divergences approach 0). However, in practice the difference between

$\mathrm{D_{KL}}[q||p]$ and $\mathrm{D_{KL}}[p||q]$ can be very large. Thus, other than in special cases, the Wake-Sleep algorithm cannot be well-understood as as optimisation of a single training objective, but rather alternation between two training objectives. Thus, for most models of interest it is difficult to know if the algorithm will converge on a good model of the data (or whether it converges at all).

2.2 Variational Autoencoders

More recently Kingma & Welling (2013) proposed the Variational Autoencoder (VAE). This offers an alternative solution to the problem of training q without relying on the above KL-divergence approximation. Instead, the authors note that it is possible to construct an unbiased approximation to the ELBo (Eq. 2) using only a single sample from q.

Under specific assumptions about the form of q – specifically, that it is a *continuous* distribution which can be reparametrised by transforming a fixed auxiliary distribution – they use this to construct a low variance estimate for the *gradient* of the ELBo. As it is unbiased, this gradient estimate can used in SGD to train both q and p, typically neural networks, simultaneously towards the ELBo

VAE Update: Sample $z \sim q(z;x)$ and take a gradient step on $\log p(x|z) - \mathrm{D_{KL}}[q(z;x)||p(z)]$.

When z are discrete, VAEs cannot be trained through the use of reparameterisation but instead rely on the *policy gradient* (otherwise called REINFORCE, Williams (1992)) estimator from reinforcement learning. This estimator is notoriously high variance, in many cases rendering SGD ineffectual. This difficulty has motivated a wide literature on variance reduction techniques, (Greensmith et al., 2004; Jang et al., 2016; Tucker et al., 2017; Grathwohl et al., 2017), yet training VAEs with discrete latent variables remains a challenging open research problem.

2.3 Comparison

The above description highlights a bias-variance tension between these two approaches (Table 2). The wake-sleep algorithm applies well to a wide variety of models, including structured models with discrete latent variables, but relies on an approximate update for q which may be heavily biased. By contrast, VAEs are proven to converge to a local optimum of the evidence lower bound (and so are often seen as more 'principled') but require much stronger assumptions on the form of the model in order for learning to be practical.

Additionally, both VAEs and Wake-sleep rely on the ability of the recognition model, $q(z;x)$, to learn to carry out posterior inference *accurately*; any departure from this changes the optimal p. This strong constraint is often unrealistic and unnecessary: on hard problems, a recognition model may still be useful if only one in a hundred samples are of high quality. Recent work aims to address this in both VAEs (Burda et al., 2015) and Wake-sleep (Bornschein & Bengio, 2014) by using importance weighting over many samples from q. This solution is well suited when fully amortised inference is just out of reach of the recognition model, but is bottlenecked by how many samples it is practical to evaluate per gradient step.

The next section describes our alternative approach, motivated by the idea that good explanations needn't be forgotten. Simply put, we mitigate the difficulties of discrete inference by introducing a separate 'memory' into the Helmholtz Machine, explicitly keeping track of the best discovered latent explanations, or *programs* z_i, for each observation x_i.

	Inference loss q	Unbiased ELBo estimate	Discrete z	Memory
Wake-Sleep	$D_{\mathrm{KL}}[p(z\|x)\|\|q(z;x)]$	X (unconvergent)	✓	X
VAEs	$D_{\mathrm{KL}}[q(z;x)\|\|p(z\|x)]$	✓	X (REINFORCE)	X
WSR	$D_{\mathrm{KL}}[q(z)\|\|p(z\|x)]$ $+D_{\mathrm{KL}}[p(z\|x)\|\|r(z;x)]$	✓	✓	✓

Table 2: Comparison of VAE and Wake-Sleep algorithms for training Helmholtz machines. Wake-sleep uses an approximation to the correct update for q, which may be heavily biased. VAE updates are unbiased, but for discrete variables they are often too high variance for learning to succeed.

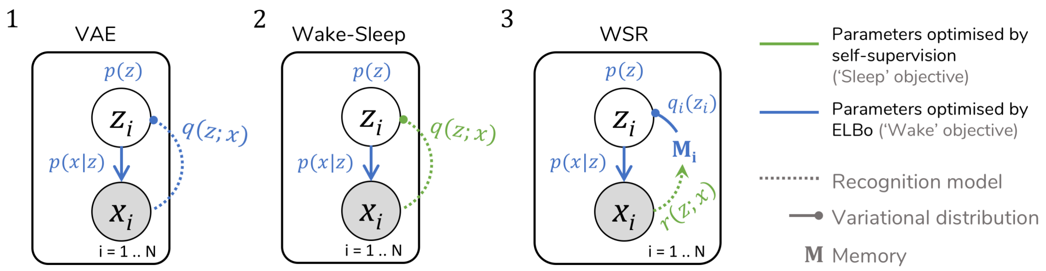

Figure 1: For VAEs and Wake-sleep, the recognition model q also serves as the variational distribution that trains p. WSR distinguishes these, learning a recognition model r and a categorical variational posterior q which is separate from r. This means that like VAEs, WSR jointly trains p and q using an unbiased estimate of the variational objective (blue). Like wake-sleep, the recognition model can train self-supervised (green), allowing WSR to handle discrete latent variables. To optimise the finite support of q, WSR incorporates a memory module M that remembers the best values of z_i found by $r(z_i; x_i)$ across iterations.

3 This Work

In this work we start from a different set of modelling assumptions to those typical of VAE-family models. Rather than describe each observation with a latent vector z which lacks explicit structure, we assume each observation is generated by an explicit latent *program*, and wish to learn:

1. A posterior distribution over the latent program $q_i(z_i)$ for each instance x_i
2. A prior $p(z)$ that captures a global inductive bias over programs.
3. Any continuous parameters of the decoder $p(x|z)$
4. An approximate recognition network $r(z; x)$ which helps infer programs for novel data.

Using programs as a latent representation makes this setup challenging for two reasons. First, as seen in Table 2, training discrete Helmholtz machines usually requires accepting either high bias or high variance in the learning objective. Second, by assumption, inferring programs from data is a hard problem, so performing highly accurate amortised inference may be overly ambitious. We therefore desire a learning algorithm for which weaker recognition models may reduce the speed of learning but will not change the set of stable solutions to the learning problem.

To achieve this, we depart from the usual Helmholtz machines formulation by separating the recognition model from the variational distribution (Figure 1). As in Wake-sleep, we train the recognition model $r(z; x)$ self supervised using samples from the prior - an effective strategy when z is discrete.

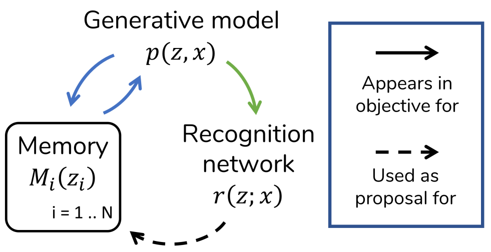

Figure 2: ***What trains what?*** r is a recognition network trained self-supervised on samples from the p. The memory M_i for each task i is a set of the 'best' z values ever sampled by r, selected according their joint probability $p(z, x_i)$. p is then trained using samples from M.

However, unlike Wake-sleep we do not use samples from r directly to train the generative model p. Instead, they are routed through a separate *Memory* module, M, which maintains a set of the best values of z found for each x, across training iterations, weighted by their joint probability $p(z, x)$. Then, we simply *resample* from M in order to train p.

By weighting each z proportional to its join probability, we guarantee that every update to M decreases the KL divergence between M_i and the true posterior $p(z_i|x_i)$. Thus, we may view each M_i as a truncated variational posterior over z_i, which is optimised towards the ELBo using samples from r as proposals. Our full training procedure is detailed in Algorithm 1.

Algorithm 1: Basic WSR training procedure (batching omitted for notational simplicity). In practice, we avoid evaluation of p_θ in the each wake phase by maintainging a cache of $p_\theta(z_M, x)$ in the sleep phase. We re-calculate each $p_\theta(z, x)$ only as a final correctness check before modifying $\mathbf{M}_i$.

```
initialize $\theta$            *Parameters for prior $p_\theta(z)$, ‘decoder’ $p_\theta(x|z)$, recognition network $r_\theta(z;x)$*
initialize $M_i$ for $i = 1, \dots, |X|$            *For each instance i, $M_i$ is a size-k set of programs*
**repeat**
  draw instance $(i, x_i)$ from dataset $X$
  *Wake*        $z_r \sim r_\theta(z; x_i)$                                          *1. Update memory with sample*
                **if** $\log p_\theta(z_r, x) > \arg\min_{z \in \mathbf{M}_i} \log p_\theta(z, x)$ :     *from recognition network*
                    add $z_r$ to $\mathbf{M}_i$ (replacing min element)
  *Remember*    $z_M \sim \mathbf{M}_i$, with probabilies $\pi_z \propto \log p_\theta(z, x)$       *2. Train generative model with*
                $s_{\text{wake}} = \log p_\theta(z_M, x)$                                  *sample from memory*
  *Sleep*       $z_p \sim p_\theta(z)$                                               *3. Train recognition network with*
                $x_p \sim p_\theta(x|z)$                                             *sample from generative model*
                $s_{\text{sleep}} = \log r_\theta(\hat{z}_p; \hat{x}_p)$
  *Hyperprior*  $z_{p'} \sim p'(z)$                                                *4. Train prior with sample from*
  *(optional)*  $s_{\text{hyper}} = \alpha \log p_\theta(z_{p'})/|X|$                     *reference distribution*
  $\mathbf{g} = \nabla_\theta \Big[ s_{\text{sleep}} + s_{\text{wake}} \ (+s_{\text{hyper}}) \Big]$
  $\theta = \theta + \lambda \mathbf{g}$                                             *Gradient step (e.g. SGD, Adam)*
**until** convergence
```

3.1 Extension to hierarchical Bayes

In the above problem setup, we assumed that the prior $p(z)$ either was fixed, or was learned to maximise the ELBo training objective. However, for many modelling problems neither is adequate: we often have some idea about the global distribution over of latent programs, but deciding on the exact $p(z)$ in advance would be too strong a commitment. In these cases we would rather provide a reference distribution $p'(z)$ as a first approximation to the global distribution, while but still allow the model to update its prior $p(z)$ to move away from p' as it learns from data. In this situation we may place a hyperprior over $p(z)$, defined with respect to the reference distribution as:

$$\mathbb{P}(p) \propto \exp\Big(- \alpha \mathrm{D}_{\mathrm{KL}}[p' || p] \Big)$$

where α is a concentration parameter controlling the level of confidence in the reference distribution p'. This form of hyperprior can be integrated into the training objective simply by addition of an extra term: $\mathbb{E}_{z \sim p'} \, \alpha \log p(z)$, estimated by sampling from p'. Algorithm 1 includes this as an optional variant, which corresponds to maximum a posteriori estimation over p.

4 Experiments

4.1 Learning cellular automata

We first test our algorithm at learning the rules for noisy 1-dimensional cellular automata, from the images they generate. We create 64×64 binary images generated row by row, sampling each pixel using a rule that depends only on its ‘neighbours’ in the row above. Specifically, given a ‘neighbourhood size’ D and ‘corruption probability’ ϵ, we generate images by the following procedure:

- Choose a binary vector $z \in \{0,1\}^{2^D}$ to represent the update rule for the cellular automaton.
- Sample the first generation uniformly at random, as $g_1 \in \{0,1\}^{64}$
- For each subsequent row $i = 2, \dots, 64$ and each cell (i, j):
    1. read the neighbouring D cells from the previous row: $s_{ij} = g_{i-1, j-\frac{D}{2}:j+\frac{D}{2}}$
    2. sample g_{ij} according to: $p(g_{ij} = z_{s_{ij}}) = 1 - \epsilon$

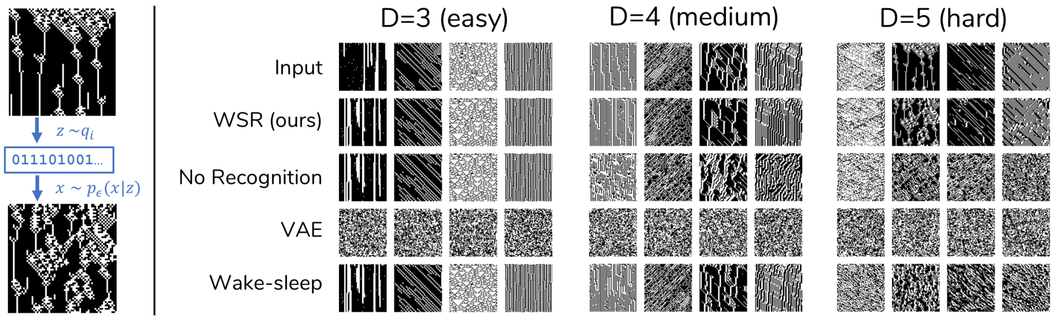

Figure 3: One-shot generalisations produced by each algorithm on each the cellular automata datasets. For each input image we sample a program z from the variational distribution q, then synthesize a new image in the same style from $p_\epsilon(z|x)$ using the learned ϵ.

We create *easy*, *medium* and *hard* datasets corresponding to increasingly complex cellular automaton rules, with neighbourhood sizes of $D = 3, 4, 5$ respectively (*easy* corresponds to the full set of 256 elementary automata studied by Wolfram in Wolfram (2002). For *medium* and *hard*, we sample 10,000 of the 65,000 and 4 billion available rules). All datasets share a noise parameter $\epsilon = 0.01$.

Our goal is discover a latent rule, or *program*, z_i corresponding to each image in the dataset, while also learning the global noise ϵ. Thus, we learn a $p_\epsilon(z, x)$ with same structure as the true generative process, and use a CNN with independent Bernoulli outputs as the recognition network $r(z; x)$. Fixing this architecture, we train WSR using $k = 5$ as the memory size, and compare performance of for the against three baseline algorithms:

- **VAE.** We use *policy-gradient* (REINFORCE) for discete choices, and additionally reduce variance by subtracting a learned baseline for each task.
- **Wake-Sleep.** We perform gradient descent on the recognition model q and generative model p together, using samples from the p to train q, and samples from q to train p.
- **No Recognition.** We evaluate a lesioned Algorithm 1 in which no recognition model is learned. We instead propose updates to M_i using samples from the prior $z_p \sim p(z)$.

Our results highlight clear differences between these approaches. Despite our efforts at variance reduction, a VAE reliably struggles to get off the ground on any dataset, and instead learns quickly to model all instances as noise (Figure 3 and 4 bottom). Wake-sleep is able to learn accurate rules for images from the easiest datasets, but on the most challenging dataset but its performance appears to asymptote prematurely. By contrast, WSR reliably learns accurate programs that can be used to classify unseen images 100-way with $> 99\%$ accuracy, even on the *hard* dataset.

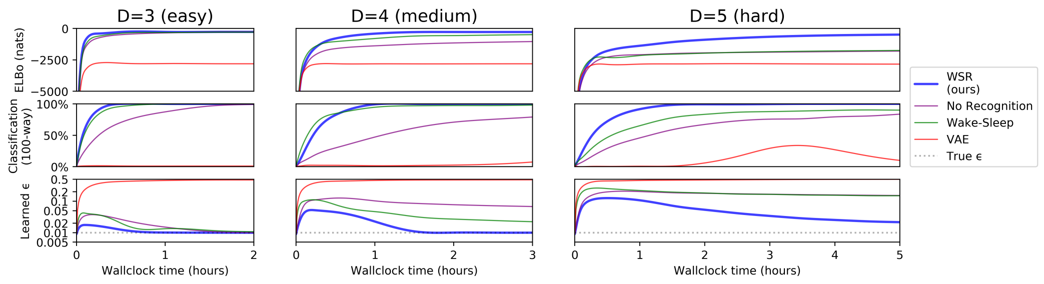

Figure 4: Quantitative results on all variants of the cellular automata dataset. In all cases WSR learns programs which generalise to unseen images of the same concepts, achieving $> 99\%$ accuracy on a 100-way classification task (second row). WSR also best recovers the true noise parameter $\epsilon = 0.01$ (third row). *Note:* x-axis is wallclock time on a single Titan-X GPU to allow a fair comparison, as WSR requires several times more computation per iteration.

4.2 Composing Gaussian process kernels

Next, we evaluate our algorithm on the the task of finding explainable models for time-series data. We draw inspiration from Duvenaud et al. (2013), who frame this problem as Gaussian process (GP) kernel learning. They describe a grammar for building kernels compositionally, and demonstrate that inference in this grammar can produce highly interpretable and generalisable descriptions of the structure in a time series. Inference is achieved on a single time-series through a custom greedy search algorithm, requiring a costly inner loop that approximately marginalises over kernel parameters.

Here, we follow a similar approach embedded within a larger goal: we use a dataset of many different time-series, and learn a *hierarchical model* over time-series. That is, we learn a separate GP kernel for each series in the dataset while also learning an inductive bias over kernels themselves.

We start with time series data provided by the UCR Time Series Classification Archive. This dataset contains 1-dimensional times series data from a variety of sources (such as household electricity usage, apparent brightness of stars, and seismometer readings). In this work, we use 1000 time series randomly drawn from this archive, and normalise each to zero mean and unit variance.

For our model, we define the following simple grammar over kernels:

$$K \to K + K \mid K * K \mid \text{WN} \mid \text{SE} \mid \text{Per} \mid \text{C}, \quad \text{where} \tag{3}$$

- WN is the *White Noise* kernel, $K(x_1, x_2) = \sigma^2 \mathbb{I}_{x_1 = x_2}$
- SE is the *Squared Exponential* kernel, $K(x_1, x_2) = \exp(-(x_1 - x_2)^2/2l^2)$
- Per is a *Periodic* kernel, $K(x_1, x_2) = \exp(-2\sin^2(\pi|x_1 - x_2|/p)/l^2)$
- C is a *Constant*, c

We wish to learn a prior distribution over both the symbolic structure of a kernel and its continuous variables (σ, l, etc.). Rather than describe a prior over kernel structures directly, we define the latent program to z to be a symbolic kernel 'expression': a string over the characters

$$\{(,), +, *, \text{WN}, \text{SE}, \text{Per}, \text{C}\}$$

We define an LSTM prior $p_\theta(z)$ over these kernel expressions, alongside parametric prior distributions over continuous latent variables ($p_{\theta_\sigma}(\sigma), p_{\theta_l}(l), \ldots$). As in previous work, exact evaluation of the marginal likelihood $p(x|z)$ of a kernel expression z is intractable and so requires an approximation. For this we use a simple variational inference scheme which cycles through coordinate updates to each continuous latent variable (up to 100 steps), and estimates a lowerbound on $p(x|z)$ using 10 samples from the variational distribution. Finally, following section 3.1, we place a hyperprior on the distribution over kernel expressions, using the grammar above (Eq. 3) as a reference distribution.

Examples of latent programs discovered by our model are displayed in Figure 5. These programs describe meaningful compositional structure in the time series data, and can also be used to make highly plausible extrapolations.

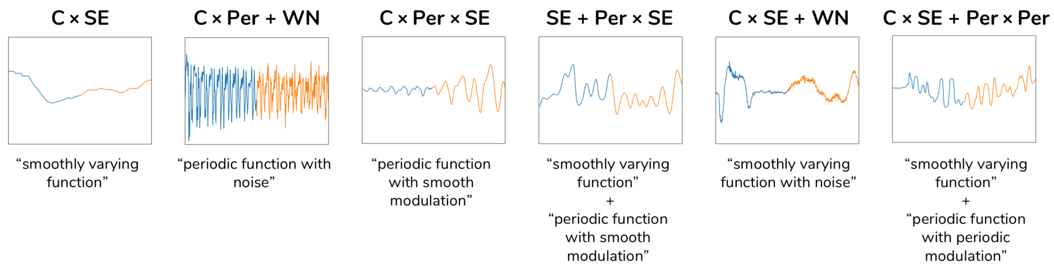

Figure 5: Kernels inferred by the WSR for various real time series in the UCR dataset. Blue (left) is a 256-timepoint observation, and orange (right) is a sampled extrapolation using the inferred kernel (top, simplified where possible). The explicit compositional structure of this latent representation allows each discovered concept to be easily translated into natural language.

4.3 Discovering structured text concepts

Finally, we test our model on the task of learning short text concepts, such as 'phone number' or 'email address', from a few examples. For this task we created a new dataset, *Text-Concepts* comprising 1500 of such concepts with 10 examples of each (Figure 1).

To collect the data, we first crawled several thousand randomly chosen spreadsheets from online public repositories on GitHub. We then automatically selected a subset of 1500 the columns from this set, filtered to remove columns that contain only numbers, English words longer than 5 characters, or common first names and surnames. To promote diversity in the dataset, we also filtered so that no two columns originated from the same spreadsheet, and no more than 3 columns share the same column header. This us to capture a wide variety of concept types (e.g. 'date', 'time') while maintaining variation that exists within each type.

Common to most patterns in the *Text-Concepts* dataset is that they can be well described by concatenative structure, as they usually involve the composition of discrete parts. With this in mind, we aim to model of this dataset using the language of *Regular Expressions* (regex).

We first define a grammar over regular expressions as follows, borrowing the standard syntax that is common to many programming languages:

$$\begin{aligned} E &\rightarrow \epsilon \mid EE \mid E{*} \mid E{+} \mid (E|E) \mid (R?) \mid \textit{Character} \mid \textit{CharacterClass} \\ \textit{Character} &\rightarrow a \mid b \mid \ldots \\ \textit{CharacterClass} &\rightarrow \texttt{.} \mid \texttt{w} \mid \texttt{d} \mid \texttt{u} \mid \texttt{l} \mid \texttt{s} \end{aligned} \tag{4}$$

where *Character* can produce any printable ASCII character, and ϵ is the empty string.

We assume that each class x_i in the Text-Conceptsdataset can be described by a latent regular expression z_i from this grammar. However, for our purposes, we endow each regex z with probabilistic, *generative* semantics. We define a likelihood (decoder) $p_\theta(x|z)$ by placing probability distributions over every random choice involved in generating a string from regex, as given in Table 3.

To evaluate the probability of a regex z generating a set of strings i, we use dynamic programming to efficiently calculate the exact probability of the most probable parse for each string in the set, and multiply these to serve as our likelihood $p_\theta(x|z)$. As in the Gaussian Process example, our $p_\theta(z)$ is parametrised as a simple LSTM, and we define a hyperprior over this by using the above grammar (Eq. 4) the reference grammar distribution.

For the recognition model we require a network which is able to generate a sequence of tokens (the regex) taking a *set* of strings as an input. We achieve this using a variant of the RobustFill architecture, introduced in Devlin et al. (2017). We pass each string in the set individually through an LSTM, and then attend to these while decoding a regular expression character by character.

Given this problem setup, our goal is to learn a regex z corresponding to each set of strings, x in the dataset, while also learning a global distribution $p(z)$ and a recognition model $r(z; x)$ to guide inference on novel sets.

For any regex expression **e**:
e* evaluates to **e+** with probability θ_+, and ϵ with otherwise.
e+ evaluates to **ee***.
e|e$_2$ evaluates to **e** with probability $\theta_|$, and **e**$_2$ otherwise.
e? evaluates to **e** with probability $\theta_?$, and ϵ otherwise.
. evaluates to any character, with probabilities $\boldsymbol{\theta}_.$
w evaluates to any alphanumeric character, with probabilities $\boldsymbol{\theta}_w$
d evaluates to any digit, with probabilities $\boldsymbol{\theta}_d$
u evaluates to any uppercase character, with probabilities $\boldsymbol{\theta}_u$
l evaluates to any lowercase character, with probabilities $\boldsymbol{\theta}_l$
s evaluates to any whitespace character, with probabilities $\boldsymbol{\theta}_s$
where θ are parameters to be learned

Table 3: The probabilistic regular expression 'decoder' $p_\theta(x|z)$, used in our model of text concepts

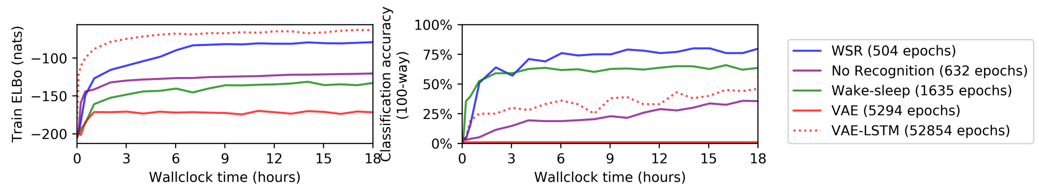

Figure 6: Quantitative comparison of models trained on the *Text-Concepts* dataset.

Quantitative results from training the above model using the WSR algorithm (with $k = 5$), are shown in Figure 6. From five examples of each concept, WSR learns a regular expression that generalises well to new examples, achieving over 75% accuracy in a challenging 100-way classification task. Comparing to Wake-Sleep and No-Recognition baselines, we find that WSR crucially utilises on both its recognition model and its memory in order to achieve this result - neither are sufficient alone.

The VAE algorithm was unable to learn effectively in our regex model, even when using control variates to reduce variance. For a more fair comparison, we also provide results from training a VAE using a different model architecture to which is better suited: for *VAE-LSTM* we use a 32-dimensional vector for the latent representation, with a fixed Gaussian prior $p(z)$, and LSTM networks for both $p(x|z)$ and $q(z|x)$. While this model is able to optimise its training objective effectively, it instead suffers from the lack of domain knowledge built into its structure. The latent representations it infers for concepts are not only less explicit but also generalise less effectively to new examples of a given concept. Table 4.3 provides examples of concepts learned from our *Text-Concepts* dataset by each of the best three models.

	Posterior sample $z \sim q_i$	**Conditional generation** $x \sim p_\theta(x\|z)$
Input:	{*\$2.80, \$3.40, \$3.70, \$5.40, \$5.70*}	
WSR:	`$d.d0`	\$9.80, \$6.60, \$9.30, …
Wake-Sleep:	`$d.dd`	\$6.03, \$1.00, \$6.15, …
VAE-LSTM:	[-1.01, 0.78, ..., -1.09]	\$66.89, \$40,722, \$4,072,08, …
Input:	{*SRX894622, SRX897016, SRX897025, SRX897027, SRX897032*}	
WSR:	`SRX89dddd`	SRX890154, SRX891233, SRX892503, …
Wake-Sleep:	`uuuddddddd`	SDE677741, IFR104042, ASO235252, …
VAE-LSTM:	[0.99, 1.35, ..., -0.10]	S19E7U0003, CPF009999, S19000143, …
Input:	{*ecous, indus, midwous, midwus, totus*}	
WSR:	`ll+us`	bpgbeus, hcus, beus, …
Wake-Sleep:	`l+`	fl, s, ab, …
VAE-LSTM:	[1.44, 1.16, ..., -0.36]	sdajm, -0.79489, al_scip-inf-1, …
Input:	{*bp_60_6, dt_62_1, gk_55_4, kt_14_5, tp_8_8*}	
WSR:	`ll_dd?_d`	ra_8_2, lc_32_4, su_6_0, …
Wake-Sleep:	`kl_dd_dd?`	km_13_95, kb_50_26, kb_83_95, …
VAE-LSTM:	[-0.28, -0.16, ..., -1.54]	h052346, ipr_pr160, v24260, …
Input:	{*(650) 323-4532, (650) 573-2385, (650) 599-1423, (650) 599-7479, (650) 877-5773*}	
WSR:	`(650)sddd-dddd`	(650) 430-0484, (650) 890-2662, (650) 060-8473, …
Wake-Sleep:	`(ddd)sddd-dddd`	(965) 179-5812, (921) 992-1623, (135) 584-3078, …
VAE-LSTM:	[0.34, 0.01, ..., -0.13]	(429) 893-1700, (425) 859-23830, (715) 236-1297, …
Input:	{*Feb 07, 2014, Jul 07, 2014, Mar 11, 2005, Nov 21, 2007, Oct 04, 2010*}	
WSR:	`ullsdd,s20dd`	Bvk 33, 2097, Ylb 67, 2016, Teb 59, 2069, …
Wake-Sleep:	`u?ol dd|d|d, dddd`	oo 25, 8421, Sor 13, 1017, oa 89, 1130, …
VAE-LSTM:	[1.26, 1.22, ..., -1.17]	M_6235451-9029, Nov 11, 2003, Dec 55,24D, …
Input:	{*L - 10.0 lbs., L - 25.2 lbs., L - 31.0 lbs., L - ??, S - 8.6 lbs.*}	
WSR:	`us-sdd?.dslbs.`	A - 75.0 lbs., M - 99.6 lbs., S - 50.4 lbs., …
Wake-Sleep:	`o*|s|(L -) (- .sl)+*`	oo - 0 s- 0 t- 1 f- 3 d, o , - 6 j- X o, …
VAE-LSTM:	[1.72, -0.03, ..., 0.21]	C214, 2-942q18, YE-2M2, …

Table 4: Learned latent representations and posterior predictive samples from models trained on the *Text-Concepts* dataset (from five examples per class).

Prior $z \sim p_\theta$	**Conditional generation** $x \sim p_\theta(x\|z)$
`csd.d+`	c 0.6, c 4.4, c 6.0, ...
`wdddd-dd`	56144-73, 60140-63, 21646-60, ...
`lld`	hc8, ft5, vs9, ...
`u0dddddd`	B0522234, M0142810, A0994226, ...
`uuud.sd0%`	TAP0. 70%, THR6. 50%, FPS9. 20%, ...
`u+.`	EA., SD., CSB., ...

Table 5: Novel concepts hallucinated by the WSR model. For each row we first sample a from the learned prior, and then generate examples of the concept by sampling from this program.

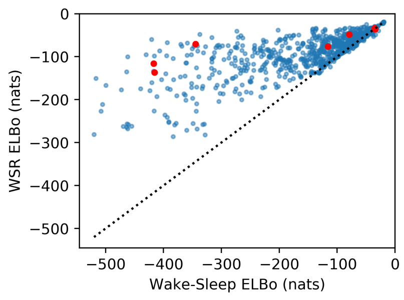

Figure 7: Comparison of ELBo between WSR and Wake-Sleep models for every example in the dataset. Examples in red are shown in Table 4.3 ordered by WSR ELBo (descending).

Finally, investigate whether WSR learns a realistic inductive bias over concepts, by sampling new concepts from the learned prior $p_\theta(z)$ and then for each of these sampling a set of instances from $p_\theta(x|z)$. In Table 4.3, we see that our model generalises meaningfully from the training data, learning higher level part structure that is common in the dataset (e.g. strings of uppercase characters) and then composing these parts in new ways.

5 Discussion

In this paper, we consider learning interpretable concepts from one or a few examples: a difficult task which gives rise to both inductive and computational challenges. Inductively, we aim to achieve strong generalisation by starting with rich domain knowledge and then 'filling in the gaps', using a large amount of background data. Computationally, we aim to tackle the challenge of finding high-probability programs by using a neural recognition model to guide search.

Putting these pieces together we propose the *Wake-Sleep-Remember* algorithm, in which a Helmholtz machine is augmented with an persistent memory of discovered latent programs - optimised as a finite variational posterior. We demonstrate on several domains that our algorithm can learn generalisable concepts, and comparison with baseline models shows that WSR (a) utilises both its recognition model and its memory in order to search for programs effectively, and (b) utilises both domain knowledge and extensive background data in order to make strong generalisations.

REFERENCES

Jörg Bornschein and Yoshua Bengio. Reweighted wake-sleep. *arXiv preprint arXiv:1406.2751*, 2014.

Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*, 2015.

Peter Dayan, Geoffrey E Hinton, Radford M Neal, and Richard S Zemel. The helmholtz machine. *Neural computation*, 7(5):889–904, 1995.

Jacob Devlin, Jonathan Uesato, Surya Bhupatiraju, Rishabh Singh, Abdel-rahman Mohamed, and Pushmeet Kohli. Robustfill: Neural program learning under noisy i/o. *arXiv preprint arXiv:1703.07469*, 2017.

David Duvenaud, James Robert Lloyd, Roger Grosse, Joshua B Tenenbaum, and Zoubin Ghahramani. Structure discovery in nonparametric regression through compositional kernel search. *arXiv preprint arXiv:1302.4922*, 2013.

Harrison Edwards and Amos Storkey. Towards a neural statistician. *arXiv preprint arXiv:1606.02185*, 2016.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017.

Will Grathwohl, Dami Choi, Yuhuai Wu, Geoff Roeder, and David Duvenaud. Backpropagation through the void: Optimizing control variates for black-box gradient estimation. *arXiv preprint arXiv:1711.00123*, 2017.

Evan Greensmith, Peter L Bartlett, and Jonathan Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5(Nov):1471–1530, 2004.

Luke B Hewitt, Maxwell I Nye, Andreea Gane, Tommi Jaakkola, and Joshua B Tenenbaum. The variational homoencoder: Learning to learn high capacity generative models from few examples. *UAI*, 2018.

Geoffrey E Hinton, Peter Dayan, Brendan J Frey, and Radford M Neal. The" wake-sleep" algorithm for unsupervised neural networks. *Science*.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

Scott Reed, Yutian Chen, Thomas Paine, Aäron van den Oord, SM Eslami, Danilo Rezende, Oriol Vinyals, and Nando de Freitas. Few-shot autoregressive density estimation: Towards learning to learn distributions. *arXiv preprint arXiv:1710.10304*, 2017.

Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, pp. 4077–4087, 2017.

George Tucker, Andriy Mnih, Chris J Maddison, John Lawson, and Jascha Sohl-Dickstein. Rebar: Low-variance, unbiased gradient estimates for discrete latent variable models. In *Advances in Neural Information Processing Systems*, pp. 2627–2636, 2017.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Stephen Wolfram. *A new kind of science*, volume 5. Wolfram media Champaign, IL, 2002.