Learning structured generative models with memoised wake-sleep

Abstract

Learning models with discrete and interpretable latent representations is an enduring goal of AI, and typically requires concurrent inference of these latents. 'Helmholtz machine' algorithms such as *Reweighted Wake-Sleep* (RWS) aim to facilitate model-learning by training a separate recognition network to perform such inferences quickly. However, while RWS is successful for learning neural generative models, we find that it struggles on more structured models which cannot easily adapt their latent space to simplify recognition. We propose an alternative framework, *Memoised Wake-Sleep* (MWS), in which a fast recognition network guides a slower inference process taking place in a persistent memory across training iterations. On three structured learning domains (stroke-based character models, regular expressions, and cellular automata) we show that MWS utilises this slow inference to learn more accurate generative models.

1 Introduction

From the phonemes that make up a word to the nested goals and subgoals that make up a plan, many of our models of the world rely on symbolic structures such as categories, objects, and the composition of parts. Models with this kind of rich and explicit representation are desirable for machine learning not only for their interpretability, but also because they are often the most flexible and robust. For example, Lake et al. (2015) developed a part-based model of handwritten characters which remains state-of-the-art at both novel character classification and generation, despite competition from a wide variety of neural models (Lake et al., 2019).

However, *learning* such a model that understands and exploits these rich structured representations remains a substantial challenge. A major barrier is the requirement for structured inference. To learn a part-based model of characters we must not only discover the building blocks themselves, but simultaneously infer which building blocks come together to make which characters.

To minimise the cost of inference, a longstanding technique in machine learning involves training two models together: a *generative model* which describes the joint distribution of latent structures and observations $p(z, x)$, and a *recognition network* $q(z|x)$ which allows fast inference of these latents. These two models, which jointly train each other, are together called a *Helmholtz Machine* (Dayan et al., 1995).

It is perhaps surprising that a fast recognition network can perform inference reliably enough to support learning. As noted by Hinton et al. (1995): *"The reason it is not a fatal flaw is that ... the algorithm adapts the generative weights so as to make $p(-|x)$ close to $q(-|x)$."*. For example, p may avoid creating difficult "explaining away" effects by finding a representation where latent variables are typically independent in the posterior.

This suggests that Helmholtz machines, and other recent variants such as VAEs (Kingma & Welling, 2013), are well-suited to learning 'unstructured' generative models such as neural networks which make minimal assumptions about the latent representation space. However, when learning models with more explicit structure, we cannot necessarily rely on finding a latent representation for which inference is sufficiently easy.

In this paper we propose a new variant of the Wake-Sleep algorithm, called *Memoised Wake-Sleep* (MWS), which is better suited to learning in these more richly structured domains. Our work rests on the insight that, despite the computational challenge of structured inference, the problems in domains such as vision and language are often also highly constrained.

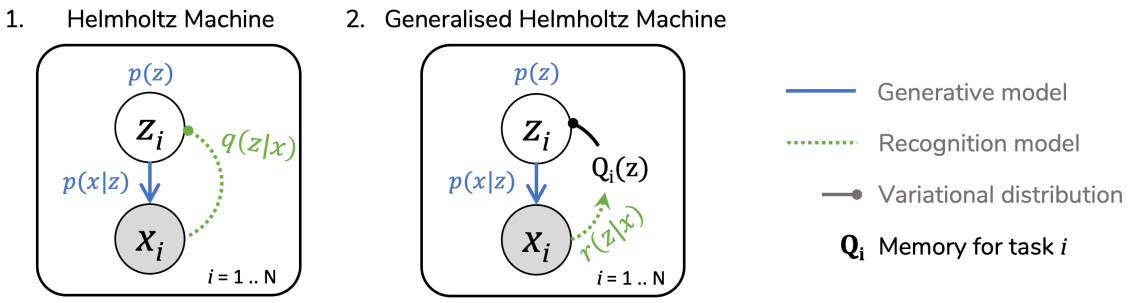

Figure 1: In a traditional Helmholtz machine, the recognition network $q(z|x)$ itself defines the variational distribution for each z_i. A generalised Helmholtz machine distinguishes these, learning both a recognition network $r(z|x)$ and a separate variational posterior Q_i for each observation. In MWS, we take Q_i to be a weighted finite set containing the top K values of z_i discovered by the recognition model in previous training iterations. todo: explain memory in pic

Rather than require that inference of each latent variable can be achieved accurately by a recognition network (as in usual Helmholtz machines) we make a different assumption: that each observation yields only a small set of likely explanations (i.e. *posteriors are narrow*).

MWS maintains a finite approximate posterior Q_i for each observation, and alternates discrete updates to these with optimisation of the model's continuous parameters, as in the generalised EM algorithm (Dempster et al., 1977). Unlike EM, discrete updates are guided by a fast recognition model to predict latents from observations.

Our work relates to the Reweighted Wake-Sleep (RWS) algorithm of Bornschein & Bengio (2014). MWS also takes a number of samples from the recognition model and weights them optimally using the generative model. However, rather than discard these samples after each iteration, the best discovered samples are remembered and integrated into the approximate posterior Q_i for use in future learning. This reuse allows MWS to learn effectively with often 5-10x less computation than RWS.

Furthermore, MWS guarantees convergence by optimising a single objective (ELBO), defined using the finite posteriors Q_i. The optima of the generative model therefore do not depend on the capacity of the recognition network. By contrast, as we show in several examples, a weaker recognition network in RWS often bottlenecks the quality of the learned generative model.

In the remainder of this paper, we first describe the *Memoised Wake-Sleep* algorithm in full, and then evaluate it on several learning domains. MWS learns models with *continuous* parameters and *discrete* latent variables, but is best-suited to those with rich latent structure. We use MWS to discover latent structure ranging from parts, to categories, to *programs* which compose functions and arguments.

2 Background

The Helmholtz Machine (Dayan et al., 1995) is a framework for learning *generative models* that explain observed data through latent variables. To aid learning, a Helmholtz machines trains an auxiliary *recognition network* which allows fast inference of these latent variables from observed data.

Formally, algorithms for training a Helmholtz Machine are motivated by Variational Bayes. Suppose we wish to learn a generative model $p(z, x)$, which is a joint distribution over latents z and observations x, alongside a recognition network $q(z|x)$, which describes 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*').

Variational Autoencoders and their variants (Kingma & Welling, 2013; Burda et al., 2015) are recent descendants of the Helmholtz Machine which have proven highly successful at learning deep generative models with continuous latent variables. However, when latents are discrete, these algorithms are often less effective due to high variance in the REINFORCE gradient estimator. A promising alternative stems from the Wake-Sleep algorithm.

2.1 The Wake-Sleep Algorithm

Proposed by Hinton et al. (1995), the Wake-Sleep algorithm alternates 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 $\log p(x|z)$ 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 $\mathrm{D_{KL}}[q(z|x)||p(z|x)]$, 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 reversed KL divergence, $\mathrm{D_{KL}}[p(z|x)||q(z|x)]$, by 'dreaming' training data from the generative model:

Sleep phase: Maximise $\log q(z|x)$ using hallucinated data: $z, x \sim p(z, x)$.

The justification for this approximation is that $\mathrm{D_{KL}}[q||p]$ and $\mathrm{D_{KL}}[p||q]$ will coincide on 0 so long as the recognition network can achieve sufficient accuracy. However, this is often too much to expect in practice, and posterior inference must be refined using Monte Carlo estimation. This is the basis for *Reweighted Wake-Sleep*.

2.2 Reweighted Wake-Sleep

The Wake-Sleep algorithm assumes that the recognition model $q(z|x)$ is able to accurately match the true posterior $p(z|x)$. However even in deep generative models, which have much flexibility in adapting the parameterisation of p to make this inference easier), fully amortised inference is still often too difficult for q to achieve.

An alternative approach views the recognition network as merely specifying a guide distribution, which can be refined to generate samples from the true posterior $p(z|x)$ by importance sampling. The *Reweighted Wake-Sleep* algorithm (Bornschein & Bengio, 2014) takes this approach, training the generative model p in the wake phase by using a set of K samples $z_k \sim q(z|x)$ and importance weighting them by $\tilde{\omega}_k = \frac{p(z_k,x)/q(z_k|x)}{\sum_k p(z_k,x)/q(z_k|x)}$

Wake phase p-update:
Update p using a gradient step of: $\sum_k \tilde{\omega}_k \nabla p(z, x)$

The recognition network may again be trained through a sleep phase, exactly as in the Wake-Sleep algorithm. An attractive alternative choice is to update q on observed rather than hallucinated data, by again using a importance sampler during the wake phase.

Wake phase q-update:
Update q using a gradient step of: $\sum_k \tilde{\omega} \nabla q(z_k|x)$

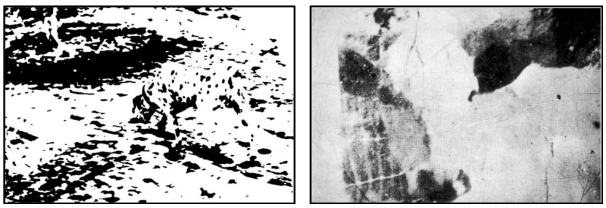

Figure 2: **Memory in perceptual recognition.** What is depicted above? Inferring the structure behind an observation can be a hard search problem even if the solution is well-defined. Most viewers are slow to identify what these images depict, yet after initial recognition, this structure is immediately remembered on future viewings. *(Answer in footnote.)*

3 The Memoised Wake-Sleep algorithm

Importance sampling in the RWS algorithm can go a long way to remedying inadequacies in the recognition network. Indeed, Bornschein & Bengio (2014) suggest that just $K = 5$ importance samples is enough for accurate inference in the neural models they study. However, for other problems this strategy may be insufficient.

Real-world structured inference problems are often similar to those depicted in Figure 2: an observation x results from a complex interaction of factors which is difficult to discover but easy to verify. x may provide strong Bayesian evidence about a latent z, with just one or two points dominating the posterior $p(z|x)$ – yet finding these explanations is still a hard search problem.

Solving this problem often requires very many samples from a recognition model before the correct explanation is found. In RWS this costly computation is repeated every iteration, estimating the posterior afresh with each gradient step and then discarding that estimate. This is wasteful if the same explanations dominate the posterior across iterations (even while their weights vary). Just as the visual system never forgets its hard-won explanations for the visual inference problems in Figure 2, we would like a Helmholtz Machine to amortise the cost of inference by explicitly remembering the solutions it finds to these hard search problems.

In the Memoised Wake-Sleep algorithm, we do not discard the result of inference after each training iteration. Instead, for each observation x_i we introduce a memory Q_i to contain a set of at most K historical samples from the recognition model. Formally, Q_i will form a variational distribution over z_i which has some finite support $\{\tilde{z}_{i1}, \ldots, \tilde{z}_{iK}\}$, and some weighting $Q_i(\tilde{z}_{ik}) = \omega_{ik}$ with $\sum_{k=1}^{K} \omega_{ik} = 1$.

Algorithm 1: Basic MWS training procedure (batching omitted for notational simplicity). In practice, we avoid evaluation of p_θ in the each wake phase by maintaining a cache of $\omega_{ik} = p_\theta(z_{ik}, xi)$, but re-calculate each $p_\theta(z, x)$ as a correctness check before modifying $\mathbf{Q}_i$.

initialize θ — *Parameters for prior $p_\theta(z)$, 'decoder' $p_\theta(x|z)$, recognition network $r_\theta(z|x)$*

initialize Q_i for $i = 1, \ldots, |X|$ — *For each instance i, Q_i is a weighted set of K programs*

repeat

draw instance (i, x_i) from dataset X

Wake

$z_r \sim r_\theta(z|x_i)$

if $\log p_\theta(z_r, x) > \arg\min_{z \in Q_i} \log p_\theta(z, x)$:

add z_r to $\mathbf{Q}_i$ (replacing min element)

1. Update memory with sample from recognition network

Remember

$z_Q \sim \mathbf{Q}_i$, with probabilies $\omega_z \propto \log p_\theta(z, x_i)$

$s_p = \log p_\theta(z_Q, x)$

$s_r = \log r_\theta(z_Q|x)$ *(optional)*

2. Train generative (and recognition) models with sample from memory

Sleep (optional)

$z_p, x_p \sim p_\theta(z, x)$

$s_r = \log r_\theta(\hat{z}_p; \hat{x}_p)$

3. Train recognition model with sample from generative model

$\mathbf{g} = \nabla_\theta \left[s_p + s_r \right]$

$\theta = \theta + \lambda \mathbf{g}$ — *Gradient step (e.g. SGD, Adam)*

until convergence

We wish to optimise Q_i to match the true posterior as closely as possible, minimising $\mathrm{D}_{\mathrm{KL}}[Q_i(z_i)||p(z_i|x_i)]$ and therefore maximising the ELBO objective (Eq 1). For a fixed set of $\tilde{z}_{ik}$s, this is achieved exactly by:

$$\omega_{ik} = p(\tilde{z}_{ik}, x_i) / \sum_{k=1}^{K} p(\tilde{z}_{ik}, x_i)$$

Given these optimal weights, it can be shown that $\mathrm{D}_{\mathrm{KL}}[Q_i(z_i)||p(z_i|x_i)]$ will decrease whenever we replace one element z_{ik} with another z'_{ik} such that $p(z'_{ik}, x_i) > p(z_{ik}, x_i)$.

This suggests a simple algorithm for optimising Q_i. Every iteration, we draw a sample z' from the recognition model, and compare it against the support set of Q_i. If there is any z_{ik} in such that $p(z', x_i) > p(z_{ik}, x_i)$, then we can replace the minimal such value and update Q_i. Otherwise Q_i is unaffected.

Memoised Wake-Sleep optimises the continuous parameters of p by the same objective as the continuous and discrete parmeters of Q_i, minimising the ELBO (Eq 2). However, the way p is updated is in a separate *remember* phase, taking a gradient step using a sample from Q_i.

To train the recognition network (now referred to as r rather than q to avoid collision) we would presumably like to supervise on pairs z, x which are close to the true posterior $p(z|x)$. As is familiar from the RWS algorithm, this is possible in two ways. One option, **Sleep-r**, is to train r in a *sleep* phase during which we hallucinate pairs $z, x \sim p$ from the generative model. This has the advantage that it matches the true posterior exactly, although the marginal $p(x)$ may be poorly matched to the true data-generating distribution. The other option, **Mem-r**, is to train r on real observations x_i where z_i is simply from the memory Q_i. This can be performed quickly alongside the *remember* phase for the p update. These two variants are described above in Algorithm 1, and summarised in Figure 3. In the experiments that follow, we evaluate MWS on three strutured learning domains, testing both **Mem-r** and **Sleep-r** variants.

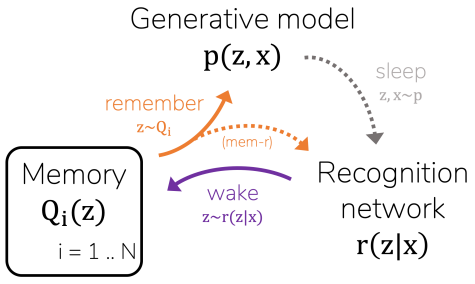

Figure 3: ***Phases of MWS*** Each phase of the algorithm corresponds to using a sample from one model to update another. During *wake*, the recognition network samples a z for x_i; if z scores sufficiently highly on $p(z, x_i)$ it is placed in memory Q_i for future iterations. During *remember*, a 'memory' z_i drawn from Q_i is used to train p and, optionally, r. Alternatively, r may be trained by sampling from p in a separate *sleep* phase.

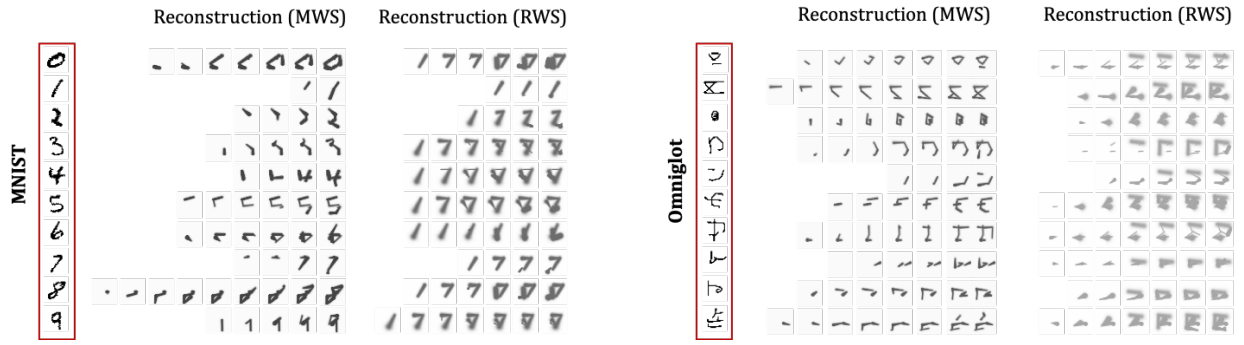

Figure 4: MWS learns an explicit latent representation for each character, in which strokes chosen from a fixed bank are sequenced together. When this same model is trained by RWS, posterior inaccuracy causes the generative model to compensate by producing more blurry strokes. Both models use K=5 with wake/mem recognition updates.

4 Experiments

4.1 Handwritten characters

Bornschein & Bengio (2014) demonstrate that the RWS algorithm (with $K = 5$) is able to learn a good model of handwritten digits from MNIST, using only binary latent variables. For the renderer $p(x|z)$ they use an autoregressive neural network (NADE Larochelle & Murray (2011)). This means that, although the latent representations are discrete, they remain uninterpretable.

For our first experiment, we investigate whether RWS and MWS can learn a more explicit generative model of the same data, utilising this discreteness for categorization and composition. We develop a simple part-based model of characters in which each image is built from a sequence pen strokes (Figure 5). Strokes can vary in darkness, thickness, and each begins where the previous stroke finished. At each stroke we allow the pen to be ON or OFF, to enable moving of the cursor without drawing.

This generative process is similar to that of Ganin et al. (2018), who aim to synthesize programs from images. We share this goal but further aim to learn a generative model, including a finite bank of possible strokes and a NADE prior $p(z)$ over characters. We train models with this same architecture on Omniglot and MNIST, with further training details provided in supplement.

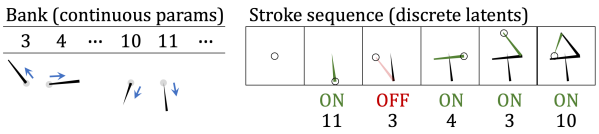

Figure 5: Model schematic. Each character is formed by chaining together a strokes chosen from a learned bank. At each stroke, the pen may be ON or OFF.

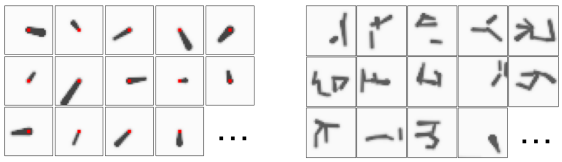

Figure 6: Left: Strokes bank learned by MWS. Right: Characters sampled from prior of Omniglot model.

Reconstructions from the learned models are shown in Figure 4. We find that MWS is able to accurately reconstruct images in both MNIST and Omniglot using a natural sequencing of strokes. Additionally, the learned prior generates realistic samples (Figure 6) by composing strokes together in novel ways. We therefore see this simple model as a good step in the open challenge to learn structured models from Omniglot data.

The training likelihood shown in Tables 1 and 2 verifies that, even with $K = 1$, MWS learns a significantly more accurate model than RWS, ammounting to a 5-10x reduction in computation.

	$K=1$	$K=3$	$K=5$	$K=10$
RWS *(Sleep-r)*	187.5	212.0	204.4	171.6
RWS *(Wake-r)*	-	168.4	166.4	152.9
MWS *(Sleep-r)*	165.5	155.1	153.7	159.7
MWS *(Mem-r)*	**156.3**	**149.1**	**144.7**	**148.8**

Table 1: Train NLL on MNIST data (Nats)

	$K=1$	$K=3$	$K=5$	$K=10$
RWS *(Sleep-r)*	161.9	156.8	155.8	151.6
RWS *(Wake-r)*	-	152.9	147.6	139.8
MWS *(Sleep-r)*	180.6	181.4	174.4	171.3
MWS *(Mem-r)*	**147.0**	**142.1**	**141.8**	**129.0**

Table 2: Train NLL on Omniglot data (Nats)

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 3: Examples from 10 of the 1500 classes contained in the *Text-Concepts* dataset.

4.2 Discovering structured text concepts

Next, we test our model on the task of learning short text concepts, such as 'date' or 'email address', from a few examples. This task is interesting for two reasons: first, because such concepts often have a richly compositional and interpretable structure. Second, because each concept is itself a fairly rich generative model, which can be flexibly applied to generate or classify new instances. Our goal is to learn a collection such concepts as latent probabilistic programs - expressed as *Regular Expressions* (Regexes). At the same time as inferring these latent programs, we wish to optimise continuous global parameters - including a distribution $p(z)$ over regular expressions themselves, and weights over character productions in the regex interpreter.

For this task we created a new dataset, *Text-Concepts* comprising 1500 of such concepts with 10 examples of each (Figure 3). 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.

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

$$\begin{aligned} E \rightarrow & \epsilon \mid EE \mid E* \mid E+ \mid (E|E) \mid (E?) \\ & \mid \textit{Character} \mid \textit{CharacterClass} \\ \textit{Character} \rightarrow & a \mid b \mid \dots \\ \textit{CharacterClass} \rightarrow & \texttt{.} \mid \texttt{w} \mid \texttt{d} \mid \texttt{u} \mid \texttt{l} \mid \texttt{s} \end{aligned} \tag{3}$$

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.

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

We define a likelihood (decoder) $p_\theta(x|z)$ by placing probability distributions over every choice involved in generating a string from regex, as given in Table 4. 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.

As in the handwritten characters example, we learn $p_\theta(z)$ as an autoregressive neural network. In paticular, we use an LSTM which can generate regular expressions as a sequence of tokens, and we define a hyperprior over this (Eq 7.1) in which the above grammar (Eq. 3) is used as a reference distribution.

	$K=1$	$K=3$	$K=5$	$K=10$
RWS *(Sleep-r)*	151.4	105.0	100.1	95.6
RWS *(Wake-r)*	-	94.7	92.7	90.9
MWS *(Sleep-r)*	**85.8**	**85.1**	87.1	87.9
MWS *(Mem-r)*	88.5	86.1	**85.4**	**85.1**

Table 5: NLL bound for Text-Concepts dataset (nats)

	$K=1$	$K=3$	$K=5$	$K=10$
RWS *(Sleep-r)*	61.5%	35.4%	31.2%	27.7%
RWS *(Wake-r)*	-	28.4%	25.3%	25.5%
MWS *(Sleep-r)*	**22.8%**	**22.9%**	24.0%	23.3%
MWS *(Mem-r)*	**22.9%**	**22.9%**	**22.8%**	**22.3%**

Table 6: Five-shot classification error (100-way)

Input x		Inferred regex $z \sim q$	Predictive samples $x' \sim p_\theta(x'\|z)$	Input x		Inferred regex $z \sim q$	Predictive samples $x' \sim p_\theta(x'\|z)$
16.5 18 18.5 19 19.5	MWS	1d(.5)?	17.5 16 17.5	[2d,Ac,6s,7h,Js] [4d,2h,7d,3h,9s] [4s,6d,8c,7s,Ks] [5s,3c,Qc,Kc,Tc] [5s,6d,2s,9d,Jh]	MWS	[dl,wl,wl,wl,wl]	[2o,6f,Py,Xv,zz] [1y,vm,Sz,ks,qi] [0x,Dk,Cb,kg,Kz]
	RWS	1d.5	17.5 16.5 18.5		RWS	[dl,d.,ww,ww,.l]	[0z,2S,x0,Fc,ao] [5f,1Q,Ur,c5,ix] [3y,0},CU,QW,5k]
A A,B,J,U B,D,E B,E B,J,U	MWS	u(,u)*	L Y,J,C P,W,Y	Jul 12 2010 Jun 1 2001 Nov 1 2004 Oct 23 2013 Sep 17 2012	MWS	ull d+ 20dd	Vna 3 2020 Wnc 06 2029 Pst 1 2057
	RWS	u,u,.*	T,C,ne Z,M, A,R,		RWS	ull d..d0d+	Loz 8MZ5056 Znd 6!m109 Kvj 10d2024402
f1f343 f1f4 f3 f3f1 f3f4f1	MWS	(fd)+d*	f4 f7 f4f9f7	$1,435,000 $118,200,000 $475,000 $5,590,000 $962,710,000	MWS	$d+,ddd,000	$0,016,000 $638,396,000 $601,968,000
	RWS	fdfd+	f7f8 f7f0 f1f42804		RWS	$d.d.d.0.0+	$1A6M7\0#0000 $9&9d2u0q0 $7[3c8O0@000
burra imburana jubranca juca jupreta	MWS	lll+a	homa ansna npga	V21 - F4~7 V34 - F8~10 V36 - F5~7 V60 - F9~11 V76 - F6~8	MWS	Vdd - Fd~dd?	V37 - F7~02 V34 - F1~09 V17 - F0~1
	RWS	llll+	occzig roomw cwlf		RWS	Vdd - Fd~d+	V48 - F2~4919 V43 - F0~62 V70 - F6~77

Figure 7: Inferred regexes and posterior predictive samples from models trained on the *Text-Concepts* dataset. For both MWS and RWS we use $K = 10$, with no sleep phase. Posterior samples are taken from Q in MWS, and from $q(z|x)$ with $K = 10$ importance sampling in RWS.

For the recognition network we require a model 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. In the case where the network produces an invalid regex string, p defaults to a catch-all regex: `.*`

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

Quantitative results from training the above model using the MWS algorithm, are shown in Table 6. From five examples of each concept, MWS learns a regular expression that generalises well to new examples, achieving over high accuracy in a challenging 100-way classification task. Figure 7 provides examples of specific concepts learned from our *Text-Concepts* dataset, either RWS or MWS algorithms. While RWS is picks up on many patterns in the text, we find that it is not able to discover concepts with more rich compositional structure, such as a list `u(,u)*`, despite using $K = 10$ samples.

We also we investigate whether MWS 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 4.2, we see that our model generalises meaningfully from the training data, learning higher level structures that are common in the dataset (e.g. strings of digits, or expressions containing `%` as the final character) and then composing these in new ways.

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, …
$d00	$600, $300, $000, …
lld	hc8, ft5, vs9, …
#dduu	#57EP, #11UW, #26KR, …
u0dddddd	B0522234, M0142810, A0994226, …
uuud.sd0%	TAP0. 70%, THR6. 50%, FPS9. 20%, …
R0<d+	R0<3, R0<9, R0<80, …
u+.	EA., SD., CSB., …

Table 7: Novel concepts hallucinated by the MWS model. For each row we first sample a from the learned prior, and then generate examples of the concept by sampling from this program.

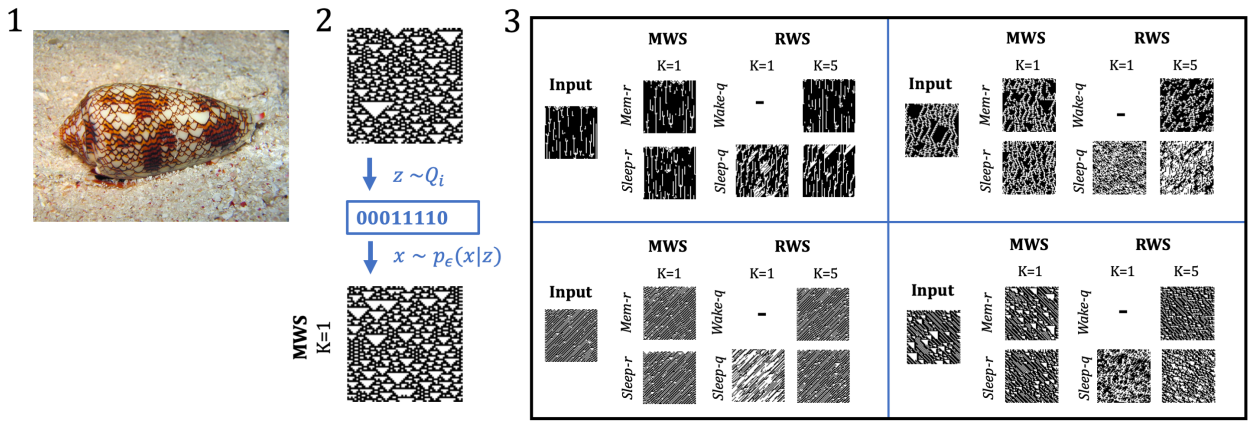

Figure 8: 1. Conus textile shell. 2. Given an image, we infer the cellular automaton rule that generated it. We then visualise this by synthesizing a new image with this rule and the model's learned noise parameter. 3. Inference and resynthesis for each algorithm. MWS learns an accurate model using only $K = 1$, producing visually similar images, whereas Wake-Sleep (RWS, $K = 1$), compensates for recognition inaccuracy by increasing the global noise ϵ.

4.3 Learning cellular automata

Finally, we highlight the use case of MWS for parameter estimation. It is common that we understand a particular process as having a clear generative structure (for example, in scientific modelling) and wish to estimate some parameter of this process. As an illustration of this, we consider the domain of noisy 1D cellular automata, studied by Stephen Wolfram (2002) and cited, among other things, as a model of seashell pigmentation.

For this experiment we create a synthetic dataset of 64×64 images generated by noisy cellular automata, and attempt to recover the true noise parameter ϵ from which they were generated. Each image is generated in rows, sampling each pixel by a rule that depends on its $D = 3$ 'neighbours' in the row above. We therefore 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 row at random: $g_1 \in \{0, 1\}^{64}$
- For each subsequent row $i = 2, \ldots, 64$ and each cell (i, j):
  1. read the neighbouring cells from the previous row: $s_{ij} = g_{i-1, j-\frac{D}{2}:j+\frac{D}{2}}$
  2. sample g_{ij} according to: $p(g_{ij} = z_{s_{ij}}) = 1 - \epsilon$

All images in the dataset were created using a noise parameter of $\epsilon = 2\%$. We aim to recover this true value of ϵ from the dataset, which requires inference of the latent rule z_i that generated each image. We thus train a model $p_\epsilon(z, x)$ with same structure as the true generative process. For the recognition network, we choose a CNN with independent Bernoulli outputs.

	$K = 1$	$K = 3$	$K = 5$	$K = 10$
RWS *(Sleep-r)*	4.30%	3.21%	2.24%	2.25%
RWS *(Wake-r)*	-	1.69%	0.28%	0.27%
MWS *(Sleep-r)*	**0.01%**	**0.01%**	**0.01%**	**0.01%**
MWS *(Mem-r)*	0.12%	0.16%	0.10%	0.18%

Table 8: Absolute error on learned ϵ when using a neighbourhood of size $D = 3$ (256 possible rules)

Figure 8 vizualises the latent rules inferred by each algorithm, by resynthesizing new images using these rules. It is visually apparent that, while MWS creates accurate resyntheses of the texture with just $K = 1$, images synthesized by Wake-Sleep (RWS, $K = 1$)are more noisy. As can be seen Table 8, this is because Wake-Sleep substantially overestimates the global noise parameter ϵ in order to compensate for inaccurate inferences by the recognition network. When K is increased to 10, RWS is still unable to achieve the accuracy that MWS reaches with $K = 1$.

Lastly, it is interesting to observe from Table 8 that, unlike our previous experiments and Le et al. (2018), MWS performs significantly better when the recognition network is trained by the *Sleep-r* objective. We hypothesize that sleep training is superior in this case because, due to the strong domain knowledge we place into p, the initial distribution of samples $x \sim p$ already matches the distribution of training data fairly well. This is in contrast to previous experiments, for which a poor initial generative model may cause r never to be trained on realistic data.

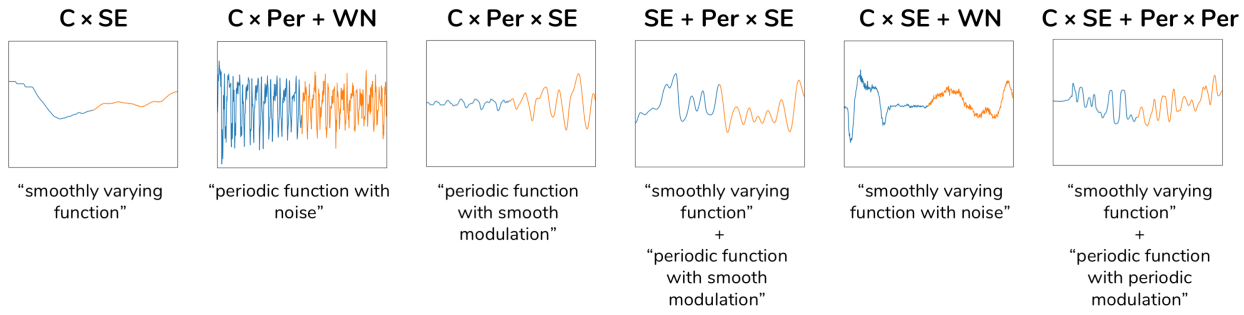

Figure 9: Learning to model the UCR time series dataset with Gaussian Processes, by inferring a latent kernel for each observed timeseries. Blue (left) is a 256-timepoint observation, and orange (right) is a sampled extrapolation using the inferred kernel (symbolic representation above, natural language representation below). During learning, we use Q_i as a memory for the discrete structure of kernels, but use a Variational Bayes inner loop to marginalise out a kernel's continuous variables when evaluation of $p(z, x)$ is required. Details in supplement.

5 Discussion

Our work builds on a history of tools for joint learning and inference. Memoised Wake-Sleep is perhaps most related to the Reweighted Wake-Sleep algorithm (Bornschein & Bengio, 2014) and to the generalised EM algorithm (Dempster et al., 1977); indeed, it may be seen as finding a bridge between the two for the specific case where all latent variables are discrete.

This view suggests that we may benefit from other inference strategies often used alongside these algorithms. For example, although MWS currently updates Q_i by sampling from the recognition network, it is a simple to include more traditional 'local search' proposals by perterbing or recombining the values of z_i already in Q_i. This strategy is common in MCMC or evolutionary search algorithms, and is often highly successful when domain knowledge can be incorporated in the design of proposals (such as split/merge proposals).

MWS also relates closely to the EC^2 algorithm (Ellis et al., 2018), in which a fast recognition model is used as a tool to guide discrete search, rather than in defining the objective for the generative model. However, they focus on learning the discrete structure of a generative models rather than its continuous parameters, and so they update to p (paralleling MWS's *remember* step) by discrete search rather than gradient optimisation. Our approach is complementary, and we see promise in the combined application of these approaches towards learning models with both with discrete and continuous parameters.

What poses a greater challenge is the inclusion of continuous *latent variables* alongside discrete ones. Our work rests on defining a variational distribution for each z_i which has only finite support. This is clearly not suited to the case where z_i involves continuous parts.

One approach to overcoming this involves using a memory of only discrete latent variables in Q_i, while marginalising out the continuous latent variables at each iteration of training (e.g. by importance sampling from a recognition model, as in RWS). In preliminary work, we have found this approach to be fruitful when dealing with the continuous latent variables that arise in a Gaussian Process model of time series data (Figure 9). By approximately marginalising out all continuous latent variables, WSR is able infer a discrete and interpretable kernel structure to describe observed timeseries in this dataset, while learning a global prior $p(z_i)$ over these kernels.

Unfortunately, as with RWS, using an approximate inference 'inner-loop' has the limitation that convergence of MWS is no longer guaranteed. MWS relies on *exact* evaluation of $p(z, x)$ in to ensure that discrete updates to the Q_i will strictly improve the ELBO objective. Adapting the algorithm so that Q_i may be updated using stochastic estimates of $p(z, x)$, without forcing these updates to be small gradient steps, is an open challenge.

6 Acknowledgments

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis id sapien lorem.

[0]Answers to '**Memory in perceptual recognition**':

Left: A dalmatian sniffs the ground, Gregory (1970)

Right: A cow looks towards the camera, KMD (1951)

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.

Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

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.

Kevin Ellis, Lucas Morales, Mathias Sablé-Meyer, Armando Solar-Lezama, and Josh Tenenbaum. Learning libraries of subroutines for neurally–guided bayesian program induction. In *Advances in Neural Information Processing Systems*, pp. 7816–7826, 2018.

Yaroslav Ganin, Tejas Kulkarni, Igor Babuschkin, SM Eslami, and Oriol Vinyals. Synthesizing programs for images using reinforced adversarial learning. *arXiv preprint arXiv:1804.01118*, 2018.

Richard Langton Gregory. The intelligent eye. 1970.

Geoffrey E Hinton, Peter Dayan, Brendan J Frey, and Radford M Neal. The" wake-sleep" algorithm for unsupervised neural networks. *Science*, 268(5214): 1158–1161, 1995.

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

KMD. A puzzle-picture with a new principle of concealment. *The American journal of psychology*, pp. 431–433, 1951.

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

Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. The omniglot challenge: A 3-year progress report. *arXiv preprint arXiv:1902.03477*, 2019.

Hugo Larochelle and Iain Murray. The neural autoregressive distribution estimator. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 29–37, 2011.

Tuan Anh Le, Adam R Kosiorek, N Siddharth, Yee Whye Teh, and Frank Wood. Revisiting reweighted wake-sleep. *arXiv preprint arXiv:1805.10469*, 2018.

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

7 Supplement

7.1 RWS-MAP

In Algorithm 1, we assumed that $p(z)$ either was either fixed, or else its parameters were learned to maximise by maximum likelihood. However, for many modelling problems neither is adequate: we may have some idea about the global distribution over latents, 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, yet allow the model to update p to move away from p' as it learns from data. In this situation we may place a 'hyperprior' over p, defined with respect to the reference distribution as:

$$\mathbb{P}(p) \propto \exp\Big(- \alpha \mathrm{D}_{\mathrm{KL}}[p' || p]\Big) \tag{4}$$

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 2 includes this as an optional variant, which corresponds to maximum a posteriori estimation over p.

7.2 Experiments

All experiments were completed on a single Titan-x GPU, with training times ranging from several hours to a day.

7.3 Timeseries data

We evaluate our algorithm on 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:

$$K \to K + K \mid K * K \mid \mathrm{WN} \mid \mathrm{SE} \mid \mathrm{Per} \mid \mathrm{C}, \quad \text{where} \tag{5}$$

- 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

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

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 7.1, we place a hyperprior on the distribution over kernel expressions, using the grammar above (Eq. 5) as a reference distribution.

Examples of latent programs discovered by our model are displayed in Figure 9. These programs describe meaningful compositional structure in the time series data, and can also be used to make highly plausible extrapolations.

7.4 Handwritten characters

In order to remain as consistent as possible with Bornschein & Bengio (2014), we use NADE for both the prior $p(z)$ and the recognition model $r(z|x)$. However, to avoid overfitting, we reduce the size of the hidden state to 16.

Algorithm 2: Basic MWS training procedure (batching omitted for notational simplicity). In practice, we avoid evaluation of p_θ in the each wake phase by maintaining a cache of $\omega_{ik} = p_\theta(z_{ik}, xi)$, but re-calculate each $p_\theta(z, x)$ only as a correctness check before modifying $\mathbf{Q}_i$.

initialize θ — *Parameters for prior $p_\theta(z)$, 'decoder' $p_\theta(x|z)$, recognition network $r_\theta(z|x)$*

initialize Q_i for $i = 1, \ldots, |X|$ — *For each instance i, Q_i is a weighted set of K programs*

repeat

draw instance (i, x_i) from dataset X

Wake
$z_r \sim r_\theta(z|x_i)$
if $\log p_\theta(z_r, x) > \arg\min_{z \in Q_i} \log p_\theta(z, x)$:
add z_r to $\mathbf{Q}_i$ (replacing min element)
— *1. Update memory with sample from recognition network*

Remember
$z_Q \sim \mathbf{Q}_i$, with probabilies $\omega_z \propto \log p_\theta(z, x_i)$
$s_p = \log p_\theta(z_Q, x)$
$s_r = \log r_\theta(z_Q|x)$ *(optional)*
— *2. Train generative (and recognition) models with sample from memory*

Sleep (optional)
$z_p, x_p \sim p_\theta(z, x)$
$s_r = \log r_\theta(\hat{z}_p; \hat{x}_p)$
— *3. Train recognition model with sample from generative model*

Hyperprior
$z_{p'} \sim p'(z)$
$s_{p'} = \alpha \log p_\theta(z_{p'})/|X|$
— *4. Train $p(z)$ with sample from reference distribution $p'(z)$*

$\mathbf{g} = \nabla_\theta \left[s_p + s_r + s_{p'} \right]$

$\theta = \theta + \lambda \mathbf{g}$ — *Gradient step (e.g. SGD, Adam)*

until convergence

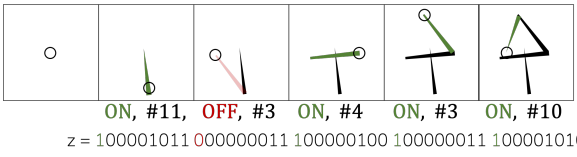

Figure 10: How the sequence of strokes in our character model is represented as a binary string for use with NADE.

NADE requires that our latent representation can be written as a binary string. We implement this by taking z to be a length-90 binary string representing 10 strokes. For each stroke, 1 bit is used to encode whether the pen is ON or OFF, and 8 are used to encode the stroke identity (see Figure 10). For the likelihood $p(x|z)$ we use a differentiable renderer to draw lines on a canvas, and then take the output to be the logits of a bernoulli likelihood. We use this same generative model structure for learning both MNIST and Omniglot.

Because the generative model is highly structured, we require significantly less data. We therefore speed up training by a smaller subset containing only $n = 500$ characters. We train using ADAM for 400,000 iterations, with a batch size of 50 and a learning rate of 0.001.

7.5 Text concepts

Our observations are sets of 5 strings, and the recognition network r that reads these strings is a RobustFill network Devlin et al. (2017) with hidden size of 128. The prior p is a highly constrained LSTM with a hidden size of 3.

We train our model for 40,000 iterations with a batch size of 100 and a learning rate of 0.002.

7.6 Cellular automata

We generated a dataset of 500 automata images, where the rules were sampled from a multivariate bernoulli, with each probability drawn uniformly at random. The noise level ϵ was fixed of 0.02.

The prior distribution $p(z)$ over (binary string) rules is a multivariate bernoulli, and the recognition model is simple 4-layer CNN (with BatchNorm). We train our model for 10,000 iterations with a batch size of 25.

As a much harder task, we also consider learning cellular automata with a neighbourhood size of $D = 5$ (yielding a total of $2^{(2^5)} = 4$ billion possible rules z. For this task we find that neither MWS nor RWS are able to accurately recover the true value of epsilon using only $K = 5$ samples. Yet, for all values of K, but that that MWS remains several times more efficient.

	$K=1$	$K=3$	$K=5$
RWS *(Sleep-r)*	17.96%	14.52%	12.51%
MWS *(Sleep-r)*	2.27%	1.99%	1.52%
RWS *(Wake-r)*	-	5.10%	3.09%
MWS *(Mem-r)*	1.85%	2.44%	2.23%

Table 9: Absolute error on learned ϵ when using $D = 5$