GENERATIVE TEMPORAL MODELS WITH MEMORY

Mevlana Gemici*, **Chia-Chun Hung***, **Adam Santoro***, **Greg Wayne***
Shakir Mohamed, Danilo J. Rezende, David Amos, Timothy Lillicrap

DeepMind, London

ABSTRACT

We consider the general problem of modeling temporal data with long-range dependencies, wherein new observations are fully or partially predictable based on temporally-distant, past observations. A sufficiently powerful temporal model should separate predictable elements of the sequence from unpredictable elements, express uncertainty about those unpredictable elements, and rapidly identify novel elements that may help to predict the future. To create such models, we introduce *Generative Temporal Models* augmented with external memory systems. They are developed within the variational inference framework, which provides both a practical training methodology and methods to gain insight into the models' operation. We show, on a range of problems with sparse, long-term temporal dependencies, that these models store information from early in a sequence, and reuse this stored information efficiently. This allows them to perform substantially better than existing models based on well-known recurrent neural networks, like LSTMs.

1 INTRODUCTION

Many of the data sets we use in machine learning applications are sequential, whether these be natural language and speech processing data, streams of high-definition video, longitudinal time-series from medical diagnostics, or spatio-temporal data in climate forecasting. Generative Temporal Models (GTMs) are a core requirement for these applications. Generative Temporal Models are also important components of intelligent agents, as they permit counterfactual reasoning, physical predictions, robot localisation, and simulation-based planning among other capacities (Sutton, 1991; Deisenroth and Rasmussen, 2011; Watter et al., 2015; Levine and Abbeel, 2014; Assael et al., 2015). These tasks require models of high-dimensional observation sequences and contain complex, long temporal dependencies—requirements that most available GTMs are unable to fulfil. Developing such GTMs is the aim of this paper.

Many GTMs—whether they are linear or nonlinear, deterministic or stochastic—assume that the underlying temporal dynamics is governed by low-order Markov transitions and use fixed-dimensional sufficient statistics. Examples of such models include Hidden Markov Models (Rabiner, 1989), and linear dynamical systems such as Kalman filters and their non-linear extensions (Kalman, 1960; Ghahramani and Hinton, 1996; Krishnan et al., 2015). The fixed-order Markov assumption used in these models is insufficient for characterising many systems of practical relevance. Bialek et al. (2001) quantitatively show that Markov assumptions fail to describe physical systems with long-range correlations, and fail to approximate the long-distance dependencies in written literature. Models that instead maintain information in large, variable-order histories, e.g., recurrent neural networks (Pearlmutter, 1995), can have significant advantages over ones constrained by fixed-order Markov assumptions.

Most recently proposed GTMs, like variational recurrent neural networks (VRNNs) (Chung et al., 2015b) and Deep Kalman Filters (Krishnan et al., 2015), are built upon well-known recurrent neural networks, like Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) and

*Equal Contributions.

Gated Recurrent Units (GRUs) (Chung et al., 2015a). In principle, these recurrent networks can solve variable-order Markovian problems, as the additive dynamics are designed to store and protect information over long intervals. In practice, they scale poorly when higher capacity storage is required. These RNNs are typically densely connected, so the parametric complexity of the model can grow quadratically with the memory capacity. Furthermore, their recurrent dynamics must serve two competing roles: they must preserve information in a stable state for later retrieval, and they must perform relevant computations to distill information for immediate use. These limitations point to the need for RNNs that separate memory storage from computation.

Recurrent networks that successfully separate memory storage from computation have been developed for several settings such as algorithm learning (Graves et al., 2014; Grefenstette et al., 2015; Joulin and Mikolov, 2015; Reed and de Freitas, 2015; Riedel et al., 2016; Vinyals et al., 2015), symbolic reasoning (Weston et al., 2014; Sukhbaatar et al., 2015), and natural language processing (Bahdanau et al., 2014; Kumar et al., 2015; Hermann et al., 2015; Kadlec et al., 2016). These recurrent networks store information in a memory buffer and use differentiable addressing mechanisms (often called "differentiable attention") to efficiently optimise reading from and writing to memory. The particular details of a system's memory access mechanisms play a critical role in determining its data efficiency.

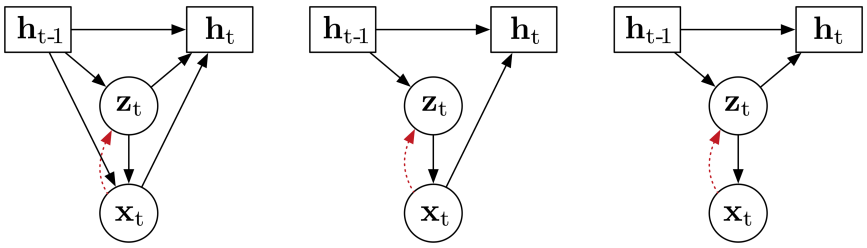
We demonstrate that generative temporal models *with memory* (GTMMs) exhibit a significantly enhanced capacity to solve tasks involving complex, long-term temporal dependencies. We develop a common architecture for generative temporal models and study four instantiations that each use a different type of memory system. These four models allow us to show how different memory systems are adapted to different types of sequential structure and the resulting impact on modelling success, data-efficiency, and generation quality. Our models are distinct from the one presented by Li et al. (2016), who developed a deep generative model for images posessing an attentional lookup mechanism. For Li et al. (2016), the memory contains a table of parameters that is not updated within a sequence. Instead, it is a table of biases that is jointly optimised for end-to-end performance. In contrast, our systems dynamically update the memory within each sequence.

We structure our discussion by first describing the general approach for designing generative temporal models and performing variational inference (Section 2). We then compare GTMMs with VRNNs (Chung et al., 2015b) on a set of visual sequence tasks designed to stress different problems that arise when modelling information with long time dependencies. Finally, we make strides toward scaling the models to richer perceptual modelling in a three-dimensional environment. In the process, we make the following technical contributions:

- We develop a general architecture for generative models with memory. This architecture allows us to develop GTMMs based on four memory systems: a new positional memory architecture referred to as an Introspection Network, the Neural Turing Machine (NTM) (Graves et al., 2014), the Least-Recently Used access mechanism (LRU) (Santoro et al., 2016), and the Differentiable Neural Computer (DNC) (Graves et al., 2016).
- We show that variational inference makes it easy to train scalable models capable of handling high-dimensional input streams leading to new state-of-the-art temporal VAEs.
- We show that our new models outperform the current state-of-the-art for GTMs based on several tasks that range from generative variants of the copy task to one-shot recall across long time delays.
- We show that our GTMMs can model realistic 3D environments and demonstrate that these models capture important aspects of physical and temporal consistency, such as coherently generating first-person views under loop closure.

2 GENERATIVE TEMPORAL MODELS

Generative temporal models (GTMs), such as Kalman filters, non-linear dynamical systems, hidden Markov models, switching state-space models, and change-point models (Särkkä, 2013) are a popular choice for modeling temporal and sequential data using latent variables. These models explain a set of observations $\mathbf{x}_{\leq T} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T\}$ with a set of corresponding latent variables



(a) Variational RNNs (b) Autoreg. dynamics (c) Latent-only

Figure 1: Variants of generative temporal models. Circled variables are stochastic; boxed variables are deterministic. Solid lines show dependencies in the generative model; dashed lines show additional dependencies for the inference model.

$\mathbf{z}_{\leq T} = \{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_T\}$ and specify the joint distribution

$$p_\theta(\mathbf{x}_{\leq T}, \mathbf{z}_{\leq T}) = \prod_{t=1}^{T} p_\theta(\mathbf{x}_t | f_x(\mathbf{z}_{\leq t}, \mathbf{x}_{<t})) p_\theta(\mathbf{z}_t | f_z(\mathbf{z}_{<t}, \mathbf{x}_{<t})), \tag{1}$$

where θ are model parameters. This formulation supports a wide range of models, some variants of which are shown in Fig. 1. Particular examples include non-linear state space models (Tornio et al., 2007), Deep Kalman Filters (Krishnan et al., 2015), and stochastic recurrent neural networks (Fraccaro et al., 2016; Bayer and Osendorfer, 2014). A particular model can be specified in Eq. (1) by fixing the distributions and the functional dependencies on the conditioned variables.

- **Distributions.** We typically assume that the prior distribution $p_\theta(\mathbf{z}_t | f_z(\mathbf{z}_{<t}, \mathbf{x}_{<t}))$ is a Gaussian and the likelihood function $p_\theta(\mathbf{x}_t | f_x(\mathbf{z}_{\leq t}, \mathbf{x}_{<t}))$ is any distribution appropriate for the observed data, such as a Gaussian for continuous observations or a Bernoulli for binary data.
- **Conditional dependencies.** Our models introduce a deterministic hidden-state variable h_t that is modified at every time point using a *transition map* $\mathbf{h}_t = f_h(\mathbf{h}_{t-1}, \mathbf{x}_t, \mathbf{z}_t)$. The function $f_z(\mathbf{h}_{t-1})$ is a *prior map* that describes the non-linear dependence on past observations and latent variables, using the hidden state, and provides the parameters of the latent variable distribution. The non-linear function $f_x(\mathbf{z}_t, \mathbf{h}_{t-1})$ is an *observation map* that provides the parameters of the likelihood function, and depends on the latent variables and state. These functions are specified using deep neural networks, which can by fully-connected, convolutional, or recurrent networks.

The most general model retains all possible dependencies between latent variables and deterministic state variables in its maps: the transition map $\mathbf{h}_t = f_h(\mathbf{h}_{t-1}, \mathbf{x}_t, \mathbf{z}_t)$ is parameterised by an LSTM network and depends on the history variable $\mathbf{h}_{t-1}$, the current observation $\mathbf{x}_t$, and the current latent variable $\mathbf{z}_t$; the observation map $f_x(\mathbf{h}_{t-1}, \mathbf{z}_t)$ depends on the past history and the current latent variable. This is the structure used by Chung et al. (2015b) in variational RNNs (VRNN) in figure 1a. VRNNs forms the baseline in our comparisons since it retains all possible dependencies within the model and provides one of the best existing models. Other dependency structures can also be considered, although they are not used in this paper: GTMs with autoregressive dynamics (figure 1b) have a transition map that depends only on visible variables, i.e. $\mathbf{h}_t = f_h(\mathbf{h}_{t-1}, \mathbf{x}_t)$, and other state-space models use observation maps that depends only on latent variables $p_\theta(\mathbf{x}_t | \mathbf{x}_{<t}, \mathbf{z}_{\leq t}) = f_x(\mathbf{z}_t)$.

2.1 Variational Inference for GTMs

Having specified a model (1), our task is to infer the posterior distribution of the latent variables and learn the model parameters. Variational inference is currently one of the most widely-used approaches, since it is well suited to problems with high-dimensional observations and high-dimensional parameter spaces. Variational inference also allows for the design of fast and scalable algorithms, is easily composed with other gradient-based learning systems, and provides tools for principled model evaluation and comparison. To compute the marginal probability of the observed data $p(\mathbf{x}_{\leq T})$, we must integrate out any latent variables $\mathbf{z}_{\leq T}$. This integration is often intractable and variational methods compute marginal probabilities by transforming this intractable integration

problem into a tractable optimization problem. We construct a variational bound on the log-marginal likelihood as follows:

$$\log p(\mathbf{x}_{\leq T}) = \log \int p_\theta(\mathbf{x}_{\leq T}, \mathbf{z}_{\leq T}) d\mathbf{z}_{\leq T} = \log \mathbb{E}_{q_\phi(\mathbf{z}_{\leq T}|\mathbf{x}_{\leq T})} \left[\frac{p_\theta(\mathbf{x}_{\leq T}, \mathbf{z}_{\leq T})}{q_\phi(\mathbf{z}_{\leq T}|\mathbf{x}_{\leq T})} \right] \tag{2}$$

$$\geq \mathbb{E}_{q_\phi(\mathbf{z}_{\leq T}|\mathbf{x}_{\leq T})} \left[\log p_\theta(\mathbf{x}_{\leq T}|\mathbf{z}_{\leq T}) \right] - \mathrm{KL} \left[q_\phi(\mathbf{z}_{\leq T}|\mathbf{x}_{\leq T}) \| p_\theta(\mathbf{z}_{\leq T}) \right] = \mathcal{F}(q; \boldsymbol{\theta}). \tag{3}$$

In Eq. (2), we rewrote the expectation in terms of a distribution $q_\phi(\mathbf{z}_{\leq T}|\mathbf{x}_{\leq T})$ with variational parameters $\boldsymbol{\phi}$. In equation (3), by application of Jensen's inequality, we obtained a lower bound on the marginal likelihood; $KL[q\|p]$ is the Kullback-Leibler divergence between distributions q and p. This lower bound (3), known as the negative free energy, has two terms that trade off reconstruction accuracy (the expected log-likelihood term) against the complexity of the posterior approximation (the KL-divergence term), and provides a tractable objective function for optimization. In this form, the distribution $q_\phi(\mathbf{z}_{\leq T}|\mathbf{x}_{\leq T})$ is an approximation to the true posterior distribution over the latent variables $p_\theta(\mathbf{z}_{\leq T}|\mathbf{x}_{\leq T})$.

We further choose an auto-regressive form for this distribution.

$$q_\phi(\mathbf{z}_{\leq T}|\mathbf{x}_{\leq T}) = \textstyle\prod_{t=1}^{T} q_\phi(\mathbf{z}_t|\mathbf{z}_{<t}, \mathbf{x}_{\leq t}); \quad q_\phi(\mathbf{z}_{<t}|\mathbf{x}_{<t}) = \prod_{\tau=1}^{t-1} q_\phi(\mathbf{z}_\tau|\mathbf{z}_{<\tau}, \mathbf{x}_{\leq \tau}). \tag{4}$$

This choice of approximate posterior distribution allows us to rewrite the total free energy $\mathcal{F}$ as the sum of per-step free energies $\mathcal{F}_t$:

$$\mathcal{F}(q; \boldsymbol{\theta}) = \textstyle\sum_{t=1}^{T} \mathbb{E}_{q_\phi(\mathbf{z}_{<t}|\mathbf{x}_{<t})} \left[\mathcal{F}_t(\, q; \boldsymbol{\theta}) \right] \tag{5}$$

$$\mathcal{F}_t = \mathbb{E}_{q_\phi(\mathbf{z}_t|\mathbf{z}_{<t}, \mathbf{x}_{\leq t})} \left[\log p_\theta(\mathbf{x}_t|\mathbf{z}_{\leq t}, \mathbf{x}_{<t}) \right] - \mathrm{KL} \left[q_\phi(\mathbf{z}_t|\mathbf{z}_{<t}, \mathbf{x}_{\leq t}) \| p_\theta(\mathbf{z}_t|\mathbf{z}_{<t}, \mathbf{x}_{<t}) \right] \tag{6}$$

A detailed derivation appears in Appendix A.

Recent approaches for variational inference use two additional tools to optimize the free energy. First, since the expectations in (5) and (6) are typically not known in closed form, the gradient of (6) is computed using a Monte Carlo estimator. For continuous latent variables, the pathwise derivative (reparameterisation trick) can be used (Fu, 2005; Rezende et al., 2014; Kingma and Welling, 2014). Second, the approximate posterior distribution q is represented by an inference (or recognition) model whose outputs are the parameters of the posterior distribution. Inference networks amortise the cost of inference across all posterior computations and make joint optimisation of the model and the variational parameters possible. The inference model $q_\phi(\mathbf{z}_t|f_q(\mathbf{x}_{\leq t}, \mathbf{z}_{<t}))$ uses a *posterior map* f_q specified by a deep network that provides the parameters of the q-distribution as a function of the current observation, and the past history of latent variables and observations. Latent variable models trained using amortised variational inference and Monte Carlo gradient estimation are referred to as variational auto-encoders (VAEs) (Kingma and Welling, 2014). For generative temporal models, they are referred to as temporal VAEs.

3 Generative Temporal Models with External Memory

In existing models, temporal structure is captured by LSTM networks with state variables $\mathbf{h}_t$ (Fig. 1). As we summarised in the introduction and will exhibit in the experiments, LSTMs are powerful sequence models but suffer from the limitation that they strongly couple memory capacity with recurrent processing and the number of trainable parameters. This limitation can result in slow learning or demand large models to achieve high capacity memory. To overcome this issue, we now develop generative temporal models with memory (GTMMs), i.e. ones that are augmented with external memory systems.

We modify our temporal VAEs to rely on the output of an external memory system, which at every point in time is queried to produce a memory context $\boldsymbol{\Psi}_t$. The prior and the posterior used become:

$$\text{Prior} \qquad p_\theta(\mathbf{z}_t|\mathbf{z}_{<t}, \mathbf{x}_{<t}) = \mathcal{N}(\mathbf{z}_t|f_z^\mu(\boldsymbol{\Psi}_{t-1}), f_z^\sigma(\boldsymbol{\Psi}_{t-1})) \tag{7}$$

$$\text{Posterior} \quad q_\phi(\mathbf{z}_t|\mathbf{z}_{<t}, \mathbf{x}_{\leq t}) = \mathcal{N}(\mathbf{z}_t|f_q^\mu(\boldsymbol{\Psi}_{t-1}, \mathbf{x}_t), f_q^\sigma(\boldsymbol{\Psi}_{t-1}, \mathbf{x}_t)) \tag{8}$$

where we use a prior that is a diagonal Gaussian that depends on the memory context through the prior map f_z, and use a diagonal Gaussian approximate posterior that depends on the observation $\mathbf{x}_t$ and the memory context $\boldsymbol{\Psi}_{t-1}$ through a posterior map f_q. We show a stochastic computational

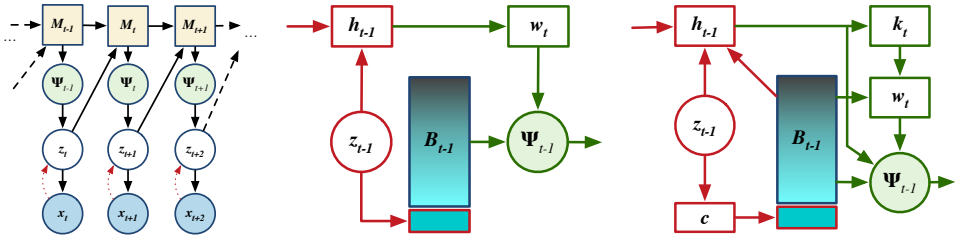


Figure 2: Components of a generative temporal model with external memory. (Left) High-level structure of the model showing the memory system M and how it connects to the generative model. Red and green lines indicate writing and reading operations, respectively. At update t, the controller state from time $t-1$ is combined with the latent variable from time $t-1$ to produce the attention weight. This produces a memory context that is only a function of the data that were in memory before time t, so we denote it $\mathbf{\Psi}_{t-1}$. (Middle) Schematic of the introspective memory system. (Right) Schematic of memory systems like NTM and DNC.

graph for the modified generative process in Fig. 2. This structure is generic and flexible and allows any type of memory system to be used, allowing the remainder of the system to be unchanged since all dependencies are through the memory context $\mathbf{\Psi}_t$.

External memory systems comprise two components: an *external memory* $\mathbf{M}_t$, which stores latent variables $\mathbf{z}_t$ (or transformations of them), and a *controller*, which implements the addressing scheme that informs memory storage and retrieval. Two types of addressing schemes are possible: *content-based addressing* accesses memories based on their similarity to a given cue, while *position-based addressing* accesses memories based on their position within the memory-store. We now expand on four types of memory systems that have different characteristics, describing the specific memory and controllers used, and how the final memory context $\mathbf{\Psi}_t$ is computed.

3.1 Introspective-GTMMs

We now develop an external memory system that uses a position-based addressing scheme for fast learning of temporal dependencies, related to the Pointer Networks of Vinyals et al. (2015). It is able to effectively handle sequences with temporally-extended dependency structures, trains quickly, and can be robustly applied to a wide variety of tasks (and we verify this in the experimental section). We refer to this memory system as an introspection network.

Memory. The memory $\mathbf{M}$ is a first-in-first-out buffer with at most L storage locations into which latent variables $\mathbf{z}_t$ are written as they are generated at each time step. It is natural to directly store the latent variables, since they are compressed representations of the data at each time point. This type of memory does not require the model to learn how to *write* to memory, but only how to read from it. This feature is what enables fast learning.

Controller. The controller is responsible for memory retrieval. At every time step t, the controller first updates the hidden states using an LSTM network f_{rnn} (Eq. 9), which fuses information from the previous hidden state $\mathbf{h}_{t-1}$ and the previously generated latent variable $\mathbf{z}_{t-1}$. If additional context information $\mathbf{c}_t$ is available, then this is also included as an input. To access a memory, soft-attention weights are computed using an attention network f_{att} based on the output of the controller $\mathbf{h}_t$ (Eq. 10). A set of R attention weights (or read heads) is used to retrieve multiple memories at the same time. Each attention weight $\mathbf{w}_t^r$ is used to compute a weighted average over the rows of the memory matrix to produce a retrieved memory vector $\boldsymbol{\phi}_t^r$.

$$\text{State update} \qquad \mathbf{h}_t = f_{rnn}(\mathbf{h}_{t-1}, \mathbf{z}_{t-1}, \mathbf{c}_t) \tag{9}$$

$$\text{Attention} \qquad \mathbf{w}_t^r = f_{att}(\mathbf{h}_t); \quad \|\mathbf{w}_t^r\| = 1, w_t^r[i] > 0 \tag{10}$$

$$\text{Retrieved memory} \qquad \boldsymbol{\phi}_t^r = \mathbf{w}_t^r \cdot \mathbf{M}_{t-1} \tag{11}$$

A number of attention functions f_{att} can be used, including softmax and Gaussian. We make use of normalised linear attention with softplus outputs $\mathbf{w}_t = k(\mathbf{h}_t)/\sum_u k(\mathbf{h}_u)$, where the function k is a deep feed-forward network. This attention system proved easy to use and did not require special initialisation. In this model, we found that softmax attention had slower convergence.

Gating mechanism. The ability to retrieve multiple memories $\boldsymbol{\phi}_t^r$ makes it possible for the latent variables $\mathbf{z}_t$ to depend on a variable number of past latent variables. We allow the network to adjust the importance of each of the retrieved memories $\boldsymbol{\phi}_t^r$ by learning correction biases $\mathbf{g}_t^r$. The corrections are passed through a sigmoid function $\sigma(\cdot)$ and element-wise multiplied with the context vector.

$$\text{Memory context} \quad \boldsymbol{\Psi}_t^r = \boldsymbol{\phi}_t^r \odot \sigma(\mathbf{g}_{t-1}^r) \tag{12}$$

The final memory context $\boldsymbol{\Psi}_t$ that is the output of the memory system is the concatenation of the memory-contexts for each read-head, $\boldsymbol{\Psi}_t = [\boldsymbol{\Psi}_t^1, \boldsymbol{\Psi}_t^2, \ldots, \boldsymbol{\Psi}_t^R]$, and forms the memory context that is passed to the generative model. The complete flow of information in the Introspection Network is shown in the stochastic computational graph in Fig. 2.

3.2 Models with Content-Based Addressing

Introspective GTMMs can learn fast, but their simple memory structure limits the range of applications to which they can be applied. We now develop GTMMs with three alternative types of memory architectures: the Neural Turing Machine (NTM) (Graves et al., 2014), which combines both content-based and positional addressing; Least-Recently Used (LRU) access, which exclusively employs content-based addressing; and the Differentiable Neural Computer (DNC) (Graves et al., 2016), which uses content-based addressing and a mechanism of positional addressing that links positions in memory based on temporal adjacency of writing. We call these models the NTM-GTMM, LRU-GTMM, and DNC-GTMM, respectively. We describe high-level aspects of the memory and controllers used, but defer detailed discussion of the properties and alternative parameterisations to Graves et al. (2014), Santoro et al. (2016) and Graves et al. (2016).

Memory. Unlike the first-in-first-out buffer used previously, the memory for NTMs and DNCs are a generic storage that allows information to be written to, and read from any location.

Controller. The controller uses an LSTM network f_{rnn} (Eq. 13) that updates the state-history $\mathbf{h}_t$ and the external memory $\mathbf{M}_t$ using the latent variables from the previous time step and any additional, context information $\mathbf{c}_t$ on which the generative model is conditioned:

$$\text{State update} \quad (\mathbf{h}_t, \mathbf{M}_t) = f_{rnn}(\mathbf{h}_{t-1}, \mathbf{M}_{t-1}, \mathbf{z}_{t-1}, \mathbf{c}_t) \tag{13}$$

To perform a content-based read of R items from the memory $\mathbf{M}_t$, the controller generates a set of keys $\mathbf{k}_t^r$ (14), and compares them to each row of the memory $\mathbf{M}_{t-1}$ using a cosine similarity measure to yield a set of soft attention weights (15). The retrieved memory $\boldsymbol{\phi}_t^r$ is then obtained by a weighted sum of the attention weights and the memory $\mathbf{M}_{t-1}$ (16).

$$\text{Keys} \qquad \mathbf{k}_t^r = f_{key}^r(\mathbf{h}_t); \quad r \in \{1, \ldots, R\} \tag{14}$$

$$\text{Attention} \quad \mathbf{w}_t^r = f_{att}(\mathbf{M}_{t-1}, \mathbf{k}_t); \quad \mathbf{w}_t^r[i] \geq 0; \|\mathbf{w}_t^r\| = 1 \tag{15}$$

$$\text{Retrieved memory} \qquad \boldsymbol{\phi}_t^r = \mathbf{w}_t^r \cdot \mathbf{M}_{t-1} \tag{16}$$

The memory context $\boldsymbol{\Psi}_t$ that is passed to the generative model is the concatenation of the retrieved memories for each read-head and the controller state, $\boldsymbol{\Psi}_t = [\boldsymbol{\phi}_t^1, \ldots, \boldsymbol{\phi}_t^R, \mathbf{h}_t]$.

4 Evaluation Methodology

We evaluated our models both qualitatively and quantitatively. Qualitative assessment involved the visual inspection of generated sequences; for example, if the task were to copy a particular portion of an observed sequence after some set number of steps, then this copy procedure should be evident in sequences generated by the models. Qualitative assessments revealed significant differences between models even when differences in their variational lower bounds were minimal. We also

Table 1: Number of parameters used in models.

Model	Digits and characters	3D environments
VRNN	1,884,177	5,912,706
Introspective-GTMM	1,863,107	5,972,806
NTM-GTMM	1,869,381	5,986,692
LRU-GTMM	1,866,282	5,979,115
DNC-GTMM	1,859,336	5,980,327

used three quantitative metrics to gather a more complete picture of the model behaviour: first, the variational lower bound objective function, tracked across training; second, the KL-divergence at a particular time-point for every training sequence (the last time-point per episode) tracked across training steps; and third, the per time-point KL-divergences averaged over a batch of sequences after training was completed.

Per time-step KL-divergences, $\mathrm{KL}\left[q_\phi(\mathbf{z}_t|\mathbf{z}_{<t}, \mathbf{x}_{\leq t})\|p_\theta(\mathbf{z}_t|\mathbf{z}_{<t}, \mathbf{x}_{<t})\right]$, measure the number of bits of additional information needed to represent the posterior distribution relative to the prior distribution over the latent variable being used to explain the current observation. They indicate the amount of prior knowledge the model contains. If a KL-divergence is close to zero, then the current observation is fully-predictable from previous information. For the tasks considered here, which were defined by random sequences at every episode, this would imply that the model stores information in memory from the beginning of the episode to construct predictive priors for the rest of the episode.

Calculating this quantity across training sequences for the last time point in each sequence (the second metric) demonstrates how quickly the memory system becomes useful for prediction across training (the last time point in our problems was always the most predictable). Viewing the average per time-step KL, averaged over a batch of sequences after training (the third metric), indicates how much information a trained model gathers throughout a sequence to make predictions.

4.1 Training Details

Our posterior and observation maps used convolutional and deconvolutional networks, in some cases with residual, skip connections (He et al., 2015). Refer to Appendix B for explicit details. All models were trained by stochastic gradient descent on the variational lower bound (Eq. 6) using the Adam optimizer (Kingma and Ba, 2014) with a learning rate of 10^{-3} (except for sequences with > 100 steps, where we used 10^{-4}). Mini-batches of 10 training sequences were used for computing gradients in all tasks. For tasks involving digits and characters, we used latent variables of size 32; for the 3D environment, we used latent variables of size 256.

We used five read heads in all tasks. The number of memory slots used for the GTMMs was taken to be the same as the number of steps in the training sequences in each task, except for the LRU-GTMM, which used a number of slots that was five times the number of time steps (note: the LRU ties the number of write heads to the number of read heads, and with many write heads it fills up a small memory quickly). The hidden state size of the LSTM in all models was chosen to keep the total number of parameters within $\sim 5\%$ of one another (see Table 1).

For each model, we ran 20 replicas with the same hyperparameters. When we performed quantifications for figures, we first averaged over all 10 example sequences in a mini-batch, then we computed means and standard errors across the replicas for the model type.

5 Experimental Results

We tested our models on seven tasks that probed their capacity to learn and make predictions about temporal data with complex dependencies. Tasks involved image-sequence modelling, and offered tests of deduction, spatial reasoning, and one-shot generalisation. Example training sequences are provided for each task described below. In Appendix C, we present generated samples for most of the tasks. For tasks with artificial data sets, pseudo-code to generate the training sequences is given in Appendix D.

Figure 3: Example sequence for the *perfect recall* task using a pre-recall interval $l = 20$ followed by a recall interval $k = 5$.

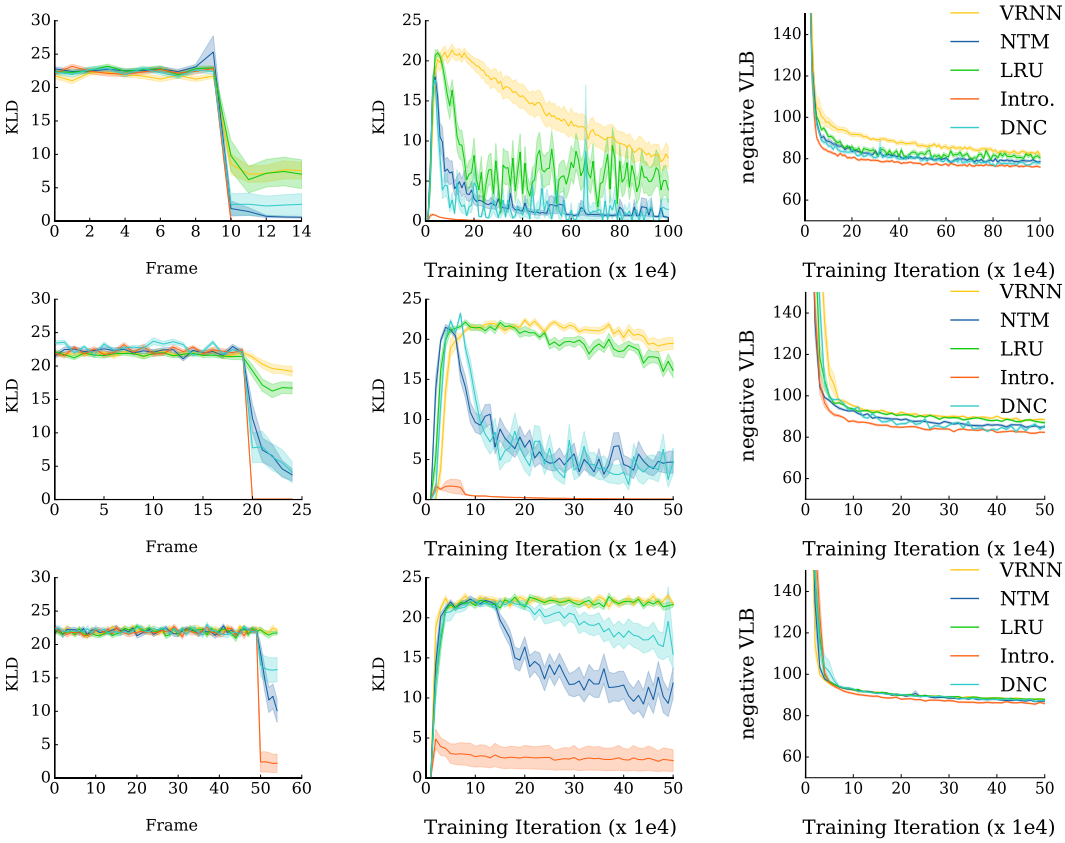


Figure 4: *Top Row*: the perfect recall task with $l = 20$ and $k = 5$. *Left:* The average Kullback-Leibler divergence (KLD) per frame, serving as a measure of prediction error between the prior and posterior. Each of the models learned that there is repetition at frame 10, but the Introspective GTMM exhibited the lowest error. *Middle:* The last frame KLD was also lowest for the Introspective GTMM at convergence. *Right:* The Introspective GTMM convergenced fastest and to the lowest level, but we see that the negative variational lower bound was close for all models. *Middle Row*: the perfect recall task with $l = 20$ and $k = 5$. The results were roughly similar. *Bottom Row*: the perfect recall task with $l = 50$ and $k = 5$. Over substantially larger time intervals, the models were able to detect regularity in the data sequences.

5.1 Perfect Recall

Training sequences consisted of k randomly sampled MNIST digits to be remembered during a pre-recall interval, which extended for l time steps. Thus, $l - k$ digits were distractor stimuli. A recall interval occurred after l steps, during which the first k images were presented again. We constructed variants of the task with $k = 5$ and $l \in \{20, 25, 55\}$. Fig. 3 shows a typical training sequence for this task. To succeed at this task, the models had to encode and store the first k images, protect them during the distractor interval, and retrieve them during the recall interval. Successful use of memory would elicit a drop in the KL-divergence during the recall interval, since information stored in memory – and not information extracted from the current observation – would be used for image reconstruction.

Figure 5: In the *parity recall* task, the recall interval consisted of images indicating the evenness or oddness of each of the initial k images.

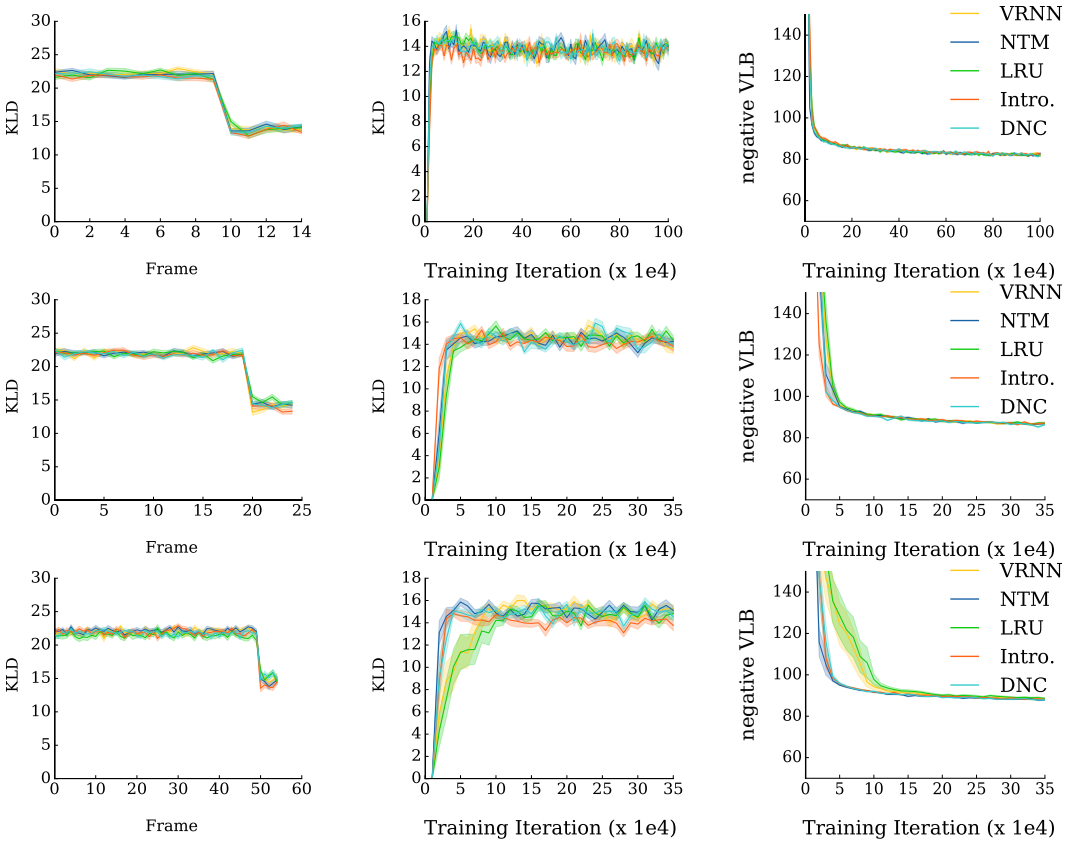


Figure 6: *Top Row*: the parity recall task with $l = 30$ and $k = 5$. The difference between models was minimal here, likely because the memory storage required to solve the task is only $k = 5$ bits. *Middle Row*: the task with parameters $l = 25$ and $k = 5$. *Bottom Row*: with $l = 50$ and $k = 5$. Again, all models succeeded equally despite the longer delay.

The performance for this task is reported in Fig. 4. All models showed the effect that the KL-divergence between the prior and posterior becomes reduced at the beginning of the recall phase. However, for the GTMMs and especially the Introspective GTMM, the reduction was most significant. This implies that the models were predicting the arriving frames based on memory. The effect was more pronounced the larger the sequence length.

5.2 Parity Recall

In contrast to the previous experiment in which exact recall of the images was demanded, in the parity recall task we asked the model to identify and report on a latent property of the data. During the recall interval, the model must generate a sequence of k 0-s and 1-s, matched to the parity of the first k images. That is, the first recalled digit should be a zero if the first digit in the initial sequence was odd, and one if it was even. Fig. 5 shows a typical training sequence. Successful models, then, need to implicitly classify input digits. Although the *computation* required for this task is more complicated than for perfect recall, the information content that is to be stored is actually smaller – i.e., a single bit per image.

Figure 7: For the *one-shot recall* task, sequences at test time were created from a set of characters that were not used during training. Even so, perfect recall should still be possible.

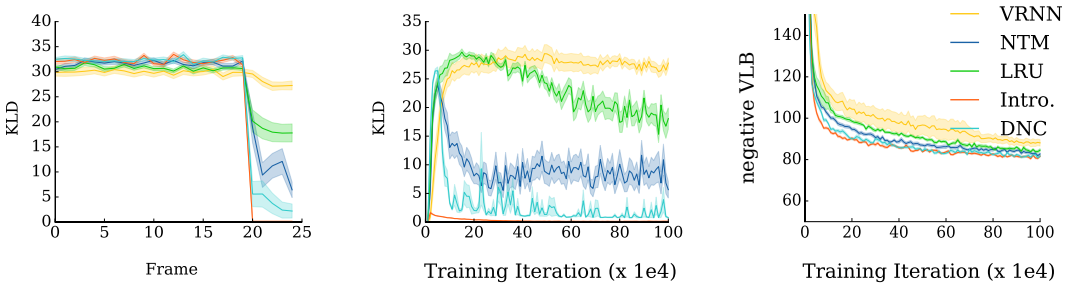


Figure 8: The one-shot recall task with experimental parameters $l = 20$ and $k = 5$. Again, because the task was very similar to perfect recall in structure, the models performed in a comparable rank order, with the Introspective GTMM showing significant KL reductions when the sequence was predictable from memory.

Less memory is required for parity recall than for perfect recall as the details of each image need not be retained; instead, only the parity of each image should be tracked, requiring 5 bits total. All models performed satisfactorily here, as we see in Fig. 6, exhibiting KL reductions when the number of possible digit classes drops from 10 to only 1 (the remembered parity digit images). The models were able to contend with long delays equally, suggesting that the primary advantage for GTMMs over VRNNs is in tasks that require the storage of a large number of bits.

5.3 One-Shot Recall

We also examined the abilities of the GTMMs to memorise novel information by testing on sequences of data on which they had not been directly trained. A typical training sequence was shown in Fig. 7, where the images at every point in time are drawn from the Omniglot data set (Lake et al., 2015). The training data consisted of all 50 alphabets with three characters excluded from each alphabet. The unseen characters were used to form new, unseen sequences at test time. The task was otherwise the same as the perfect recall task, but the demands on the generative model and memory were more substantial.

The GTMMs all outperformed the VRNN with the Introspective GTMM showing significant reductions in KL divergence at the beginning of the recall phase (Fig. 8). This was notable because the memorised images are entirely novel hold-outs from the training set. Thus, the GTMMs, and in particular the Introspective GTMM, were able to construct useful latent variable representations for novel stimuli, store them in memory, and use them to predict future events during the recall phase.

5.4 Learning Dynamic Dependencies

The preceding tasks have all demanded ordered recall of the sequence. Here, we tested whether recall in a more complicated order is possible. A typical training sequence is shown in Figure 9. We began with a sequence of l digits as before. The next k digits were generated by an "index-and-recall" game. In the figure example, the final digit in the pre-recall interval is an 8. The numerical value of the digit indicates from which position in the sequence the next digit is copied. Here, position 8 contains a 3, which is the first digit in the recall interval. Position 3 contains a 0, and so on. Successful models therefore had to learn to classify digits and to use the class labels to find images based on their temporal order of presentation.

Figure 9: Training sequence for the *dynamic dependency* task following an index-and-recall game in which each image digit provided a positional reference to the next digit in the sequence order.

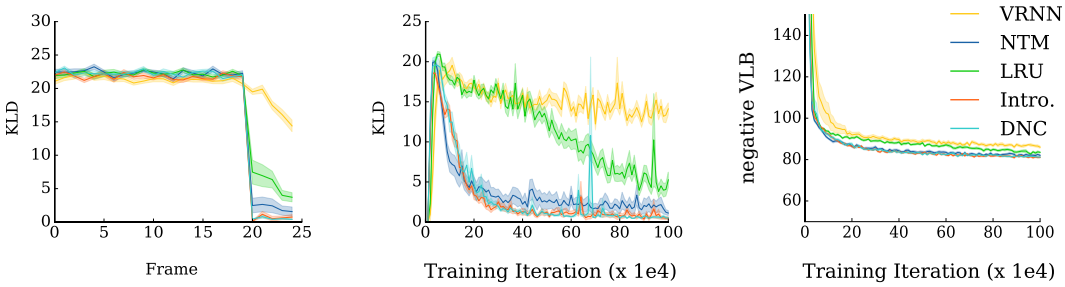


Figure 10: Dynamic dependency task with $l = 20$ and $k = 5$: The DNC-GTMM and the Introspective GTMM were the best at this complex addressing task.

This task requires the models to learn an algorithmic addressing procedure in which the current image indicates the time point the next image was stored, allowing the memory address storing the latent variables from that time point to be looked up. All of the GTMMs perform considerably better than the VRNN on the task, with the most substantial improvements achieved by the Introspective GTMM and the DNC-GTMM (Fig. 10).

5.5 Similarity-Cued Recall

The last task probed positional indexing, but here we construct a task that demands content-based addressing. In each training sequence, we first present a *random sequence* of digits for l time steps. The k digits in the *recall interval* are a randomly chosen, contiguous sub-sequence of length k from the pre-recall interval (Fig. 11). To solve this task, a model must be able to use the first image in the recall interval as a cue, find the most similar image to the cue that it has seen previously, and produce the temporal sequence that followed it.

Similarity-cued recall played more strongly to the advantages of memory systems with content-based addressing. The task required using a cue image to find the images in sequence that followed the cue during the pre-recall exposure phase. To perform this operation, the memory systems with content-based addressing, the NTM, LRU, and DNC, could encode the cue and look up similar feature encodings in memory. As long as there was a mechanism to iterate through the subsequent latent variables, the task can then be easily solved. DNC-GTMM could use its temporal transition links for this task, as described in (Graves et al., 2016). Because it lacks content-based addressing, the Introspective-GTMM did not perform better than the VRNN (Fig. 12).

5.6 Navigation in an MNIST Map

An important motivation for developing GTMMs was the desire to improve the capacity of agents to understand the spatial structure of their environments. As an example problem, we created a 2D environment represented by a 4×4 grid, where each grid cell contained a random MNIST digit (Fig. 13). In this case, instead of an agent, we took a random walk on the grid using actions up, down, left, and right for 25 steps to move between neighbouring locations, receiving an observation corresponding to the current grid cell. The actions were always treated as context variables and were not predicted by the GTMMs. We expected that the GTMMs conditioned on random walk action sequences would be able to generate the same observation in case a grid cell is revisited in an action sequence; that is, the GTMM should have maintained a coherent map of an environment.

Figure 11: Example training sequence for the *similarity-cued recall* Task.

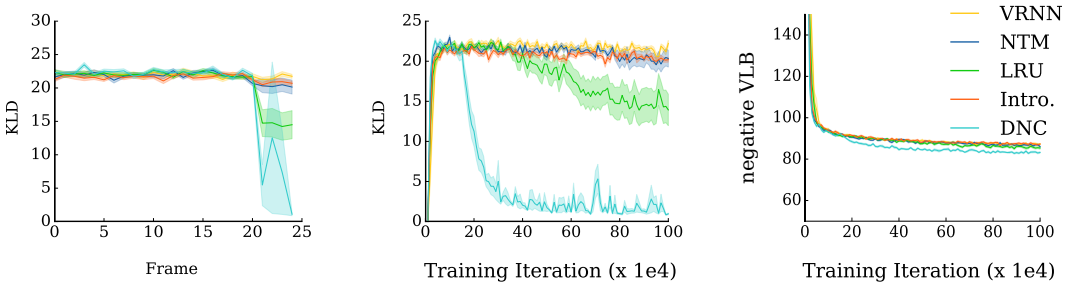


Figure 12: Similarity-cued recall with temporal dependencies of length $l = 20$ and $k = 5$. The memory models with content-based addressing, i.e., the NTM-, LRU-, and DNC-GTMM, showed the most significant KL reductions and lowest task losses.

Since the actions are generated by a random walk process, the structure of this problem is poorly captured by memory addressing mechanisms that are based on time or positional order in a sequence. Instead, models that used content-based addressing could encode the sequence of actions alongside the latent variable representations of the images. When it was necessary to predict what is present at an already visited location, content-based addressing could be used to look up the latent variables based on the action sequence. Since any grid location could be reached via multiple routes, the models had also to capture the invariance that action sequences should be converted into displacements from the origin. The DNC- and LRU-GTMM performed best at this task (Figure 14), exhibiting significant KL reductions as more of the environment was explored. In Fig. 15, we also show generation of a sequences in the maze by each model. Only LRU-GTMM and DNC-GTMM consistently generated the same digits when returning to the same positions.

Although the NTM-backed GTMM has content-based addressing, its ability to allocate free locations in memory is generally inferior to the abilities of models using LRUs and DNCs. The LRU- and DNC-based mechanisms could easily store new memories but also could collocate the new memories with context information registering the computed position on the grid each image was located.

5.7 Towards Coherent Generation in Realistic 3D Environments

Ultimately, we wish to design agents that operate in realistic environments and can learn from sequential information, using memory to form predictions. These agents should possess spatio-temporally coherent memories of environments, understanding that walls typically do not shift and that undisturbing movements change camera angles but not scene arrangements. Our first study of this problem was to test whether GTMMs can maintain consistent predictions when provided with frames from an in-place rotation of a camera for two full turns. These experiments used a procedurally-generated 3D maze environment with random wall configurations, textures, and object positions.

The rotational dynamics of the environment included acceleration, so each frame did not represent a view from an angle that is equally distributed around the unit circle. The models had to cope with this structure. Additionally, because the frames were captured at discrete moments, the models had to learn to interpolate past views after the full turn, instead of merely copying frames from the first rotation. The rotational period was $t = 15$ steps, and a full episode took place over $t = 30$ steps. An example training sequence is shown in Figure 16. A successful generative model of this data had to create random environment panoramas and generate views consistent with the panorama on the second full turn.

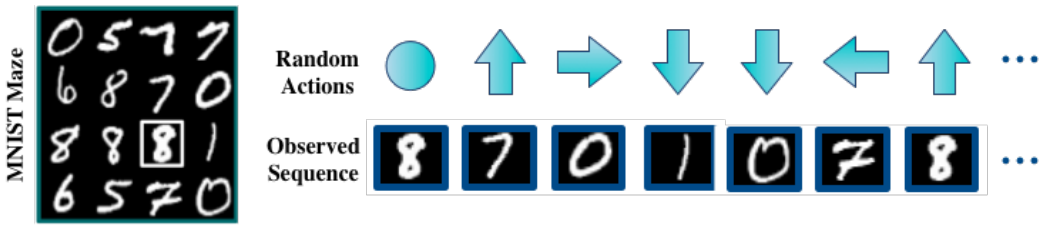


Figure 13: A random walk action sequence was provided to the model along with images found at each location. At boundaries, the action sequence was restricted to stay in bounds. Each action was provided as a conditioning variable for generation that was not modeled itself.

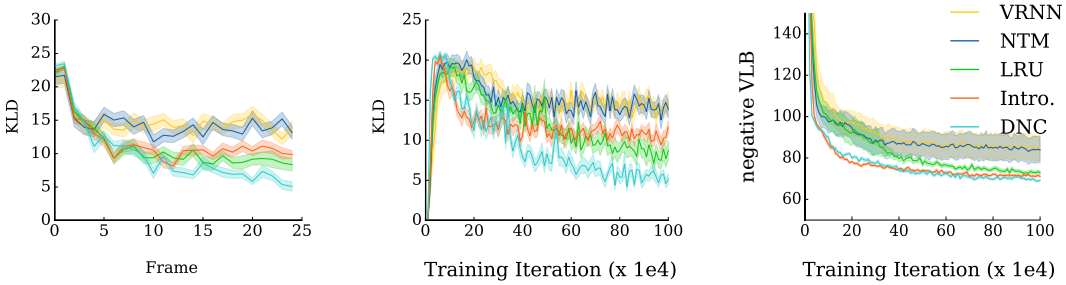


Figure 14: MNIST Maze task. The GTMMs with content-based addressing acquired information about the environment and could coherently model it.

In Fig. 17, we see that the VRNN had the lowest variational lower bound. However, in the generative samples, it is clear that the VRNN was also incoherent across time, as it forgot information about paintings on the walls and buildings on the skyline (Fig. 28). We argue that the more representative quantification, of more significance than the training loss, is the Kullback-Leibler divergence of the last frame, as well as the KL reduction at the time of the turn. These indicators showed that the DNC-GTMM and Introspective-GTMM models were able to predict the redundant frames from memory.

6 Discussion and Conclusions

The aim of this paper has been to open a research direction. We have seen that, on a range of tasks, standard generative temporal models are limited by their memory capacity and memory access mechanisms. This has motivated the design of new generative temporal models with external memory systems whose performance is in some cases qualitatively better.

We have tried to provide proofs of concept without extraneous complication: for example, our variational distributions are simple, diagonal Gaussians; we could consider more complex posterior distributions, like those used in DRAW (Gregor et al., 2015), normalising flows (Rezende and Mohamed, 2015), auxiliary variables (Ranganath et al., 2016), or models with discrete variables (Eslami et al., 2016). We have also aimed to explore the advantages and disadvantages of a variety of external memory mechanisms for generative temporal modelling. Our results suggest that none of the architectures we report on is uniformly dominant across tasks. However, an interesting direction of further research is strongly suggested by our results. Namely, we imagine that a new model, combining the direct storage of latent variables, as in the Introspective-GTMM, with content-based addressing, as in the NTM-, LRU-, and DNC-GTMM, could indeed prove to have performance that is uniformly dominant across all tasks. Furthermore, many of these sparse memory access models can be more efficiently implemented by using fast K-nearest neighbour lookup, though we did not explore such savings here. We leave the development of these more sophisticated models to future work.

The storage of latent variables or transformed latent variables in memory additionally suggests several intriguing extensions of our framework. Currently, the models are based on the mathematics

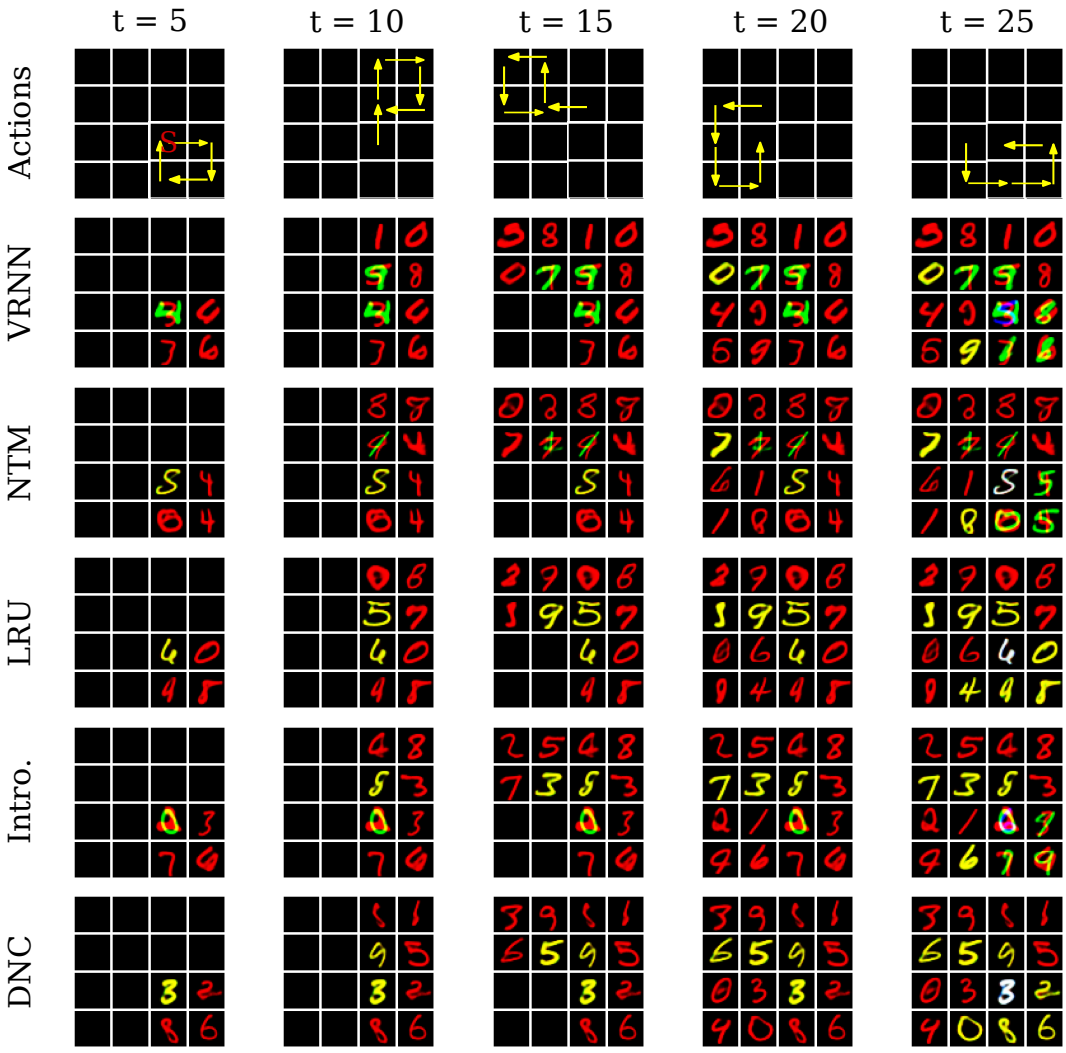


Figure 15: Generation examples in the MNIST Maze. Actions are shown in the top row. In each column of that row, the last five actions at each time point ($t = 5, 10, 15, \dots$) are indicated by yellow arrows. The best four models of each type are shown in each row below. The first time a grid cell was visited, the red pixel channel was turned on; the second time, a new image was generated and superimposed, with the red and green channels turned on (so overlaid image colour is red+green=yellow); the third time, the blue channel was turned on, so the overlaid, generated image was white. We see that the VRNN had inconsistent generations, as the subsequent visits to grid cells produced overlays of different colours that did not share the same shape. The NTM and Introspection models were by comparison better at coherent generation, and the LRU- and DNC-based GTMMs were the best here, so that, for example, the white and yellow digits masked the underlying red digit.

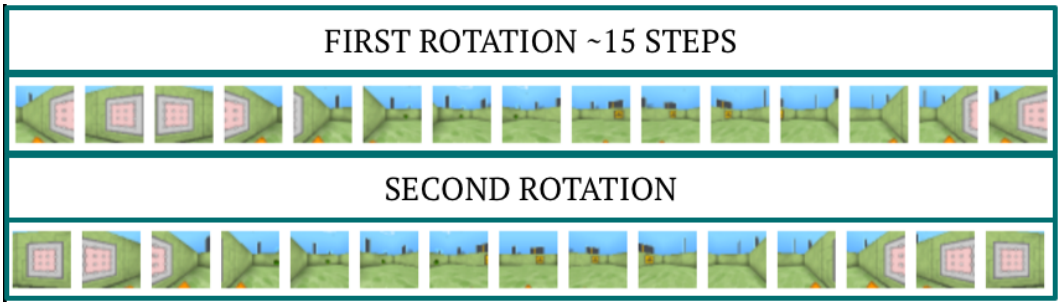


Figure 16: A training sequence for a 30 timestep in-place rotation, where the period until repetition was approximately 15 steps.

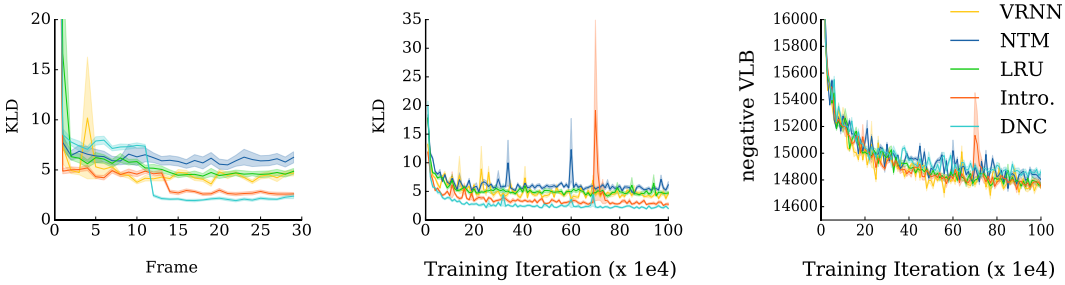


Figure 17: Realistic environments task. The VRNN had the lowest variational lower bound, but this measure tells very little of the overall story. Instead, the DNC-GTMM and Introspective-GTMM showed significant KL reductions when the sequence is predictable from memory, and this measure translates into good behaviour during generation of samples.

of optimal filtering: they produce a sample at every time step that is drawn from the filtering posterior $q_\phi(\mathbf{z}_t|\mathbf{z}_{<t}, \mathbf{x}_{\leq t})$. Once latent variables are stored in memory, the formal distinction between filtering and smoothing (with a non-temporally causal posterior $q_\phi(\mathbf{z}_t|\mathbf{x}_{\leq T})$) begins to break down. We can imagine further mechanisms that modify previously written latent variables after the fact by overwriting previously written locations. The explicit storage of latent variables in memory supports this, whereas it would be difficult to precisely modify the component of the latent state that encodes a historical latent variable in a more conventional, densely connected RNN.

REFERENCES

J.-A. M. Assael, N. Wahlström, T. B. Schön, and M. P. Deisenroth. Data-efficient learning of feedback policies from image pixels using deep dynamical models. *arXiv preprint arXiv:1510.02173*, 2015.

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

J. Bayer and C. Osendorfer. Learning stochastic recurrent networks. *arXiv preprint arXiv:1411.7610*, 2014.

W. Bialek, I. Nemenman, and N. Tishby. Predictability, complexity, and learning. *Neural computation*, 13(11):2409–2463, 2001.

J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Gated feedback recurrent neural networks. *arXiv preprint arXiv:1502.02367*, 2015a.

J. Chung, K. Kastner, L. Dinh, K. Goel, A. C. Courville, and Y. Bengio. A recurrent latent variable model for sequential data. In *Advances in neural information processing systems*, pages 2962–2970, 2015b.

M. Deisenroth and C. E. Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.

S. A. Eslami, N. Heess, T. Weber, Y. Tassa, D. Szepesvari, G. E. Hinton, et al. Attend, infer, repeat: Fast scene understanding with generative models. In *Advances In Neural Information Processing Systems*, pages 3225–3233, 2016.

M. Fraccaro, S. K. Sønderby, U. Paquet, and O. Winther. Sequential neural models with stochastic layers. In *Advances in Neural Information Processing Systems*, 2016.

M. C. Fu. Stochastic gradient estimation. Technical report, DTIC Document, 2005.

Z. Ghahramani and G. E. Hinton. Parameter estimation for linear dynamical systems. Technical report, Technical Report CRG-TR-96-2, University of Totronto, Dept. of Computer Science, 1996.

A. Graves, G. Wayne, and I. Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.

E. Grefenstette, K. M. Hermann, M. Suleyman, and P. Blunsom. Learning to transduce with unbounded memory. In *Advances in Neural Information Processing Systems*, pages 1828–1836, 2015.

K. Gregor, I. Danihelka, A. Graves, D. Jimenez Rezende, and D. Wierstra. Draw: A recurrent neural network for image generation. In *ICML*, 2015.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

K. M. Hermann, T. Kocisky, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and P. Blunsom. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, pages 1693–1701, 2015.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

A. Joulin and T. Mikolov. Inferring algorithmic patterns with stack-augmented recurrent nets. In *Advances in Neural Information Processing Systems*, pages 190–198, 2015.

R. Kadlec, M. Schmid, O. Bajgar, and J. Kleindienst. Text understanding with the attention sum reader network. *arXiv preprint arXiv:1603.01547*, 2016.

R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, 2014.

D. P. Kingma and M. Welling. Auto-encoding variational Bayes. In *ICLR*, 2014.

R. G. Krishnan, U. Shalit, and D. Sontag. Deep kalman filters. *arXiv preprint arXiv:1511.05121*, 2015.

A. Kumar, O. Irsoy, J. Su, J. Bradbury, R. English, B. Pierce, P. Ondruska, I. Gulrajani, and R. Socher. Ask me anything: Dynamic memory networks for natural language processing. *arXiv preprint arXiv:1506.07285*, 2015.

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

S. Levine and P. Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems*, pages 1071–1079, 2014.

C. Li, J. Zhu, and B. Zhang. Learning to generate with memory. *arXiv preprint arXiv:1602.07416*, 2016.

B. A. Pearlmutter. Gradient calculations for dynamic recurrent neural networks: A survey. *IEEE Transactions on Neural networks*, 6(5):1212–1228, 1995.

L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

R. Ranganath, D. Tran, and D. M. Blei. Hierarchical variational models. In *International Conference on Machine Learning*, 2016.

S. Reed and N. de Freitas. Neural programmer-interpreters. *arXiv preprint arXiv:1511.06279*, 2015.

D. J. Rezende and S. Mohamed. Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770*, 2015.

D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *ICML*, 2014.

S. Riedel, M. Bošnjak, and T. Rocktäschel. Programming with a differentiable forth interpreter. *arXiv preprint arXiv:1605.06640*, 2016.

A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap. One-shot learning with memory-augmented neural networks. *arXiv preprint arXiv:1605.06065*, 2016.

S. Särkkä. *Bayesian filtering and smoothing*, volume 3. Cambridge University Press, 2013.

S. Sukhbaatar, J. Weston, R. Fergus, et al. End-to-end memory networks. In *Advances in neural information processing systems*, pages 2440–2448, 2015.

R. S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin*, 2(4):160–163, 1991.

M. Tornio, A. Honkela, and J. Karhunen. Time series prediction with variational bayesian nonlinear state-space models. In *Proc. European Symp. on Time Series Prediction (ESTSP07)*, pages 11–19, 2007.

O. Vinyals, M. Fortunato, and N. Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700, 2015.

M. Watter, J. Springenberg, J. Boedecker, and M. Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in Neural Information Processing Systems*, pages 2746–2754, 2015.

J. Weston, S. Chopra, and A. Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.

A PER STEP VARIATIONAL LOWER BOUND

The variational bound gives

$$\log p(\mathbf{x}) = \log \int d\mathbf{z}\, p(\mathbf{x}, \mathbf{z}) \tag{17}$$

$$= \log \int d\mathbf{z}\, p(\mathbf{x}, \mathbf{z}) \frac{q(\mathbf{z}|\mathbf{x})}{q(\mathbf{z}|\mathbf{x})} \tag{18}$$

$$= \log \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} \frac{p(\mathbf{x}, \mathbf{z})}{q(\mathbf{z}|\mathbf{x})} \tag{19}$$

$$\geq \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} \log \frac{p(\mathbf{x}, \mathbf{z})}{q(\mathbf{z}|\mathbf{x})} \quad \text{(by Jensen's ineq.)}. \tag{20}$$

For sequences of random variables, the corresponding inequality is

$$\log p(\mathbf{x}_{\leq T}) \geq \mathbb{E}_{q(\mathbf{z}_{\leq T}|\mathbf{x}_{\leq T})} \log \frac{p(\mathbf{x}_{\leq T}, \mathbf{z}_{\leq T})}{q(\mathbf{z}_{\leq T}|\mathbf{x}_{\leq T})}. \tag{21}$$

In our formulation, we assume the factorisations

$$p(\mathbf{x}_{\leq T}, \mathbf{z}_{\leq T}) = \prod_{t=1}^{T} p(\mathbf{x}_t|\mathbf{z}_{\leq T}, \mathbf{x}_{<t}) p(\mathbf{z}_t|\mathbf{z}_{<t}, \mathbf{x}_{<t})$$

and

$$q(\mathbf{z}_{\leq T}|\mathbf{x}_{\leq T}) = \prod_{t=1}^{T} q(\mathbf{z}_t|\mathbf{z}_{<t}, \mathbf{x}_{\leq t}).$$

These allow us to convert equation 21 into a per time-step bound. We have:

$$\log p(\mathbf{x}_{\leq T}) \geq \mathbb{E}_{\prod_{t=1}^{T} q(\mathbf{z}_t|\mathbf{z}_{<t}, \mathbf{x}_{\leq t})} \left[\sum_{t=1}^{T} \log p(\mathbf{x}_t|\mathbf{z}_{\leq t}, \mathbf{x}_{<t}) + \log p(\mathbf{z}_t|\mathbf{z}_{<t}, \mathbf{x}_{<t}) - \log q(\mathbf{z}_t|\mathbf{z}_{<t}, \mathbf{x}_{\leq t}) \right] \tag{22}$$

$$= \mathbb{E}_{\prod_{t=1}^{T} q(\mathbf{z}_t|\mathbf{z}_{<t}, \mathbf{x}_{\leq t})} \left[\sum_{t=1}^{T} \log p(\mathbf{x}_t|\mathbf{z}_{\leq t}, \mathbf{x}_{<t}) + \log \frac{p(\mathbf{z}_t|\mathbf{z}_{<t}, \mathbf{x}_{<t})}{q(\mathbf{z}_t|\mathbf{z}_{<t}, \mathbf{x}_{\leq t})} \right]. \tag{23}$$

Let $C_t \equiv \log p(\mathbf{x}_t|\mathbf{z}_{\leq t}, \mathbf{x}_{<t}) + \log \frac{p(\mathbf{z}_t|\mathbf{z}_{<t}, \mathbf{x}_{<t})}{q(\mathbf{z}_t|\mathbf{z}_{<t}, \mathbf{x}_{\leq t})}$. Then:

$$\log p(\mathbf{x}_{\leq T}) \geq \mathbb{E}_{\prod_{t=1}^{T} q(\mathbf{z}_t|\mathbf{z}_{<t}, \mathbf{x}_{\leq t})} \left[\sum_{t=1}^{T} C_t \right] \tag{24}$$

$$= \int_{z_1} q(\mathbf{z}_1|\mathbf{x}_1) \int_{z_2} q(\mathbf{z}_2|\mathbf{z}_1, \mathbf{x}_{\leq 2}) \cdots \int_{z_T} q(\mathbf{z}_T|\mathbf{z}_{<T}, \mathbf{x}_{\leq T}) \sum_{t=1}^{T} C_t. \tag{25}$$

Each C_t is not a function of the elements of the set $\{z_{t+1}, z_{t+2}, \ldots, z_T\}$. Therefore, we can move each C_t out from the integrals involving only those terms:

$$\begin{aligned}
&= \int_{z_1} q(\mathbf{z}_1|\mathbf{x}_1) C_1 \int_{z_2} q(\mathbf{z}_2|\mathbf{z}_1, \mathbf{x}_{\leq 2}) \cdots \int_{z_T} q(\mathbf{z}_T|\mathbf{z}_{<T}, \mathbf{x}_{\leq T}) \\
&+ \int_{z_1} q(\mathbf{z}_1|\mathbf{x}_1) \int_{z_2} q(\mathbf{z}_2|\mathbf{z}_1, \mathbf{x}_{\leq 2}) C_2 \cdots \int_{z_T} q(\mathbf{z}_T|\mathbf{z}_{<T}, \mathbf{x}_{\leq T}) \\
&+ \ldots \\
&+ \int_{z_1} q(\mathbf{z}_1|\mathbf{x}_1) \int_{z_2} q(\mathbf{z}_2|\mathbf{z}_1, \mathbf{x}_{\leq 2}) \cdots \int_{z_T} q(\mathbf{z}_T|\mathbf{z}_{<T}, \mathbf{x}_{\leq T}) C_T
\end{aligned} \tag{26}$$

The interior integrals all sum to 1 since the q-s are distributions. Thus, we have:

$$
\begin{aligned}
&= \int_{z_1} q(\mathbf{z}_1|\mathbf{x}_1)C_1 \\
&+ \int_{z_1} q(\mathbf{z}_1|\mathbf{x}_1) \int_{z_2} q(\mathbf{z}_2|\mathbf{z}_1, \mathbf{x}_{\leq 2})C_2 \\
&+ \dots \\
&+ \int_{z_1} q(\mathbf{z}_1|\mathbf{x}_1) \int_{z_2} q(\mathbf{z}_2|\mathbf{z}_1, \mathbf{x}_{\leq 2}) \cdots \int_{z_T} q(\mathbf{z}_T|\mathbf{z}_{<T}, \mathbf{x}_{\leq T})C_T. \qquad (27)
\end{aligned}
$$

We can write this more simply as

$$
\mathcal{F} = \sum_{t=1}^{T} \mathbb{E}_{\prod_{\tau=1}^{t} q(\mathbf{z}_\tau|\mathbf{z}_{<\tau}, \mathbf{x}_{\leq \tau})} C_t. \qquad (28)
$$

Finally, we bring this expression into a more conventional form by writing

$$
\begin{aligned}
\mathcal{F} &= \sum_{t=1}^{T} \mathbb{E}_{\prod_{\tau=1}^{t} q(\mathbf{z}_\tau|\mathbf{z}_{<\tau}, \mathbf{x}_{\leq \tau})} \left[\log p(\mathbf{x}_t|\mathbf{z}_{\leq t}, \mathbf{x}_{<t}) + \log \frac{p(\mathbf{z}_t|\mathbf{z}_{<t}, \mathbf{x}_{<t})}{q(\mathbf{z}_t|\mathbf{z}_{<t}, \mathbf{x}_{\leq t})} \right] \\
&= \sum_{t=1}^{T} \mathbb{E}_{\prod_{\tau=1}^{t-1} q(\mathbf{z}_\tau|\mathbf{z}_{<\tau}, \mathbf{x}_{\leq \tau})} \left[\mathbb{E}_{q(\mathbf{z}_t|\mathbf{z}_{<t}, \mathbf{x}_{\leq t})} \left(\log p(\mathbf{x}_t|\mathbf{z}_{\leq t}, \mathbf{x}_{<t}) + \log \frac{p(\mathbf{z}_t|\mathbf{z}_{<t}, \mathbf{x}_{<t})}{q(\mathbf{z}_t|\mathbf{z}_{<t}, \mathbf{x}_{\leq t})} \right) \right] \\
&= \sum_{t=1}^{T} \mathbb{E}_{\prod_{\tau=1}^{t-1} q(\mathbf{z}_\tau|\mathbf{z}_{<\tau}, \mathbf{x}_{\leq \tau})} \Big[\mathbb{E}_{q(\mathbf{z}_t|\mathbf{z}_{<t}, \mathbf{x}_{\leq t})} \log p(\mathbf{x}_t|\mathbf{z}_{\leq t}, \mathbf{x}_{<t}) \\
&\qquad\qquad - \mathrm{KL}[q(\mathbf{z}_t|\mathbf{z}_{<t}, \mathbf{x}_{\leq t})||p(\mathbf{z}_t|\mathbf{z}_{<t}, \mathbf{x}_{<t})] \Big]. \qquad (29)
\end{aligned}
$$

Bringing back the distributional parameters p_θ and q_ϕ yields an equation equivalent to the main text.

B Visual Architectures

B.1 Tasks with Digits and Characters

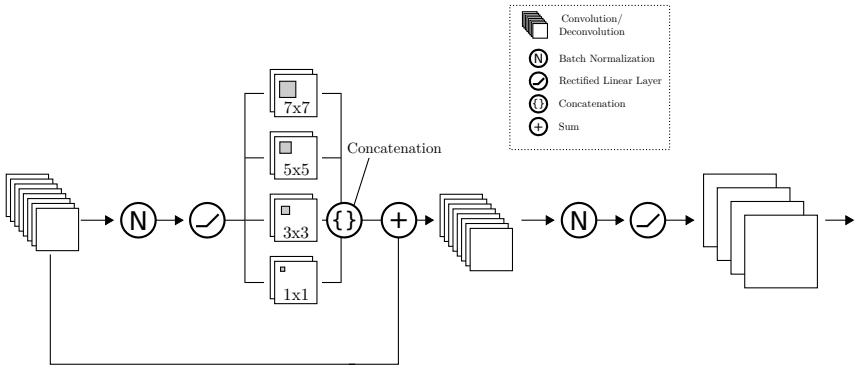


Figure 18: The first visual block.

The posterior map – that is, the mapping from inputs x_t to latents z_t – was implemented as a convolutional neural network (CNN). The CNN consisted of two "blocks," arranged in series (the first block is shown in Figure 18). The first block was fed a $1 \times 28 \times 28$ grey-scale image as input (rescaling the input images if necessary), where the saturation was mapped into the range $(-1, 1)$. This input was passed to four, parallel, dimension-preserving convolutional streams, which each

convolved the input using 8 kernels of size 1×1, 3×3, 5×5, and 7×7, respectively, padding as necessary for dimension-preservation. The outputs from the parallel streams were passed through a batch-normalization layer and a rectified-linear layer, before being concatenated to a total of 32 feature-maps. These 32 feature-maps served as the input to a dimension-halving convolution using a kernel of size 3×3, which was followed by a batch normalization layer and a rectified linear layer. Thus, this first processing block functioned to take in a 32×32 image as input and return 32 feature-maps of size 16×16. A second, identical block followed from the first, except for two differences: there was no batch normalization and rectified layer after the final convolution, and the final number of kernels was 64. To produce a sample of a latent variable, we then mapped the kernels through a linear layer to a vector of length 64. 32 of these dimensions were used to construct the mean μ and 32 were used to construct the vector $\log \sigma$ for the parameters of a Gaussian distribution. Together with a sample from a standard normal ϵ, the latent was generated as $z_t = \mu_t + \sigma_t \epsilon_t$, thus completing the posterior map.

The observation map – the mapping from latents and any recurrent deterministic variables to reconstructions $\hat{x}_t$ – was identical to the posterior map, except convolution operations were replaced with deconvolution operations.

B.2 Tasks Involving Frames of 3D Environments

For the three-dimensional environment visual model, our inputs included colour channels comprising $3 \times 32 \times 32$ values. We used two separate pathways, one roughly to encode global information across an image, and another roughly to encode local textures. The global information pathway convolved the image using 128 5×5 kernels with stride 3, no padding. These were then passed through 256 4×4 kernels with stride 2 without padding. Then the feature-maps were convolved with 256 4×4 kernels with stride 1 without padding. This gave 256 1×1 super-pixels.

The local texture pathway had a convolutional layer with 16 kernels of size 4×4, stride 2, padding of 1. The feature-maps were passed through another convolutional layer with 8 kernels of size 4×4, stride 2, and padding of 1. This yielded a block of size $8 \times 8 \times 8$, which was linearised to 512 vector elements and concatenated with the 256 features from the global information pathway.

These were passed through a linear layer to produce a 250-dimensional μ and 250-dimensional $\log \sigma$ for the latent distribution.

The observation map was the transpose of this model as before.

C Generated Samples for Select Tasks

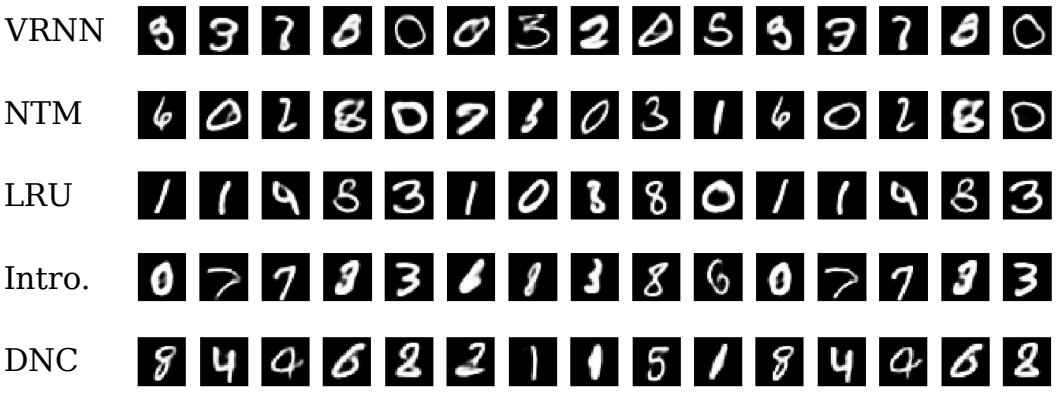


Figure 19: Perfect recall with $l = 10$, $k = 5$. The last $k = 5$ frames should match the first 5.

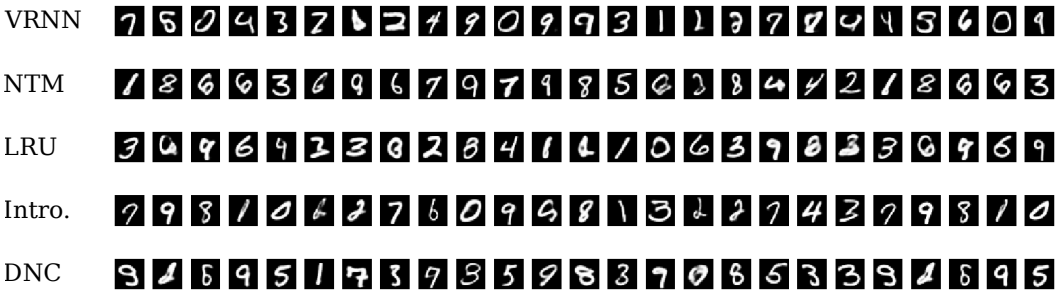


Figure 20: Perfect recall with $l = 20$, $k = 5$. The last $k = 5$ frames should match the first 5.

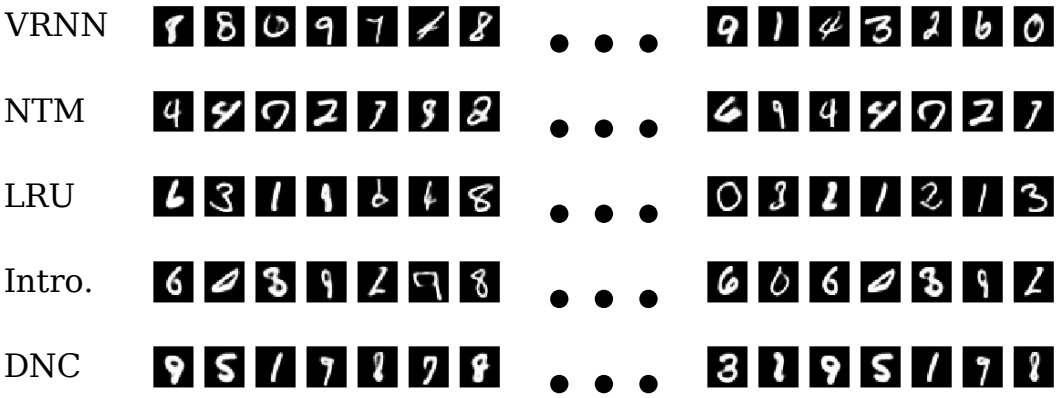


Figure 21: Perfect recall with $l = 50$, $k = 5$. Intermediate frames are omitted. The last $k = 5$ frames should match the first 5.

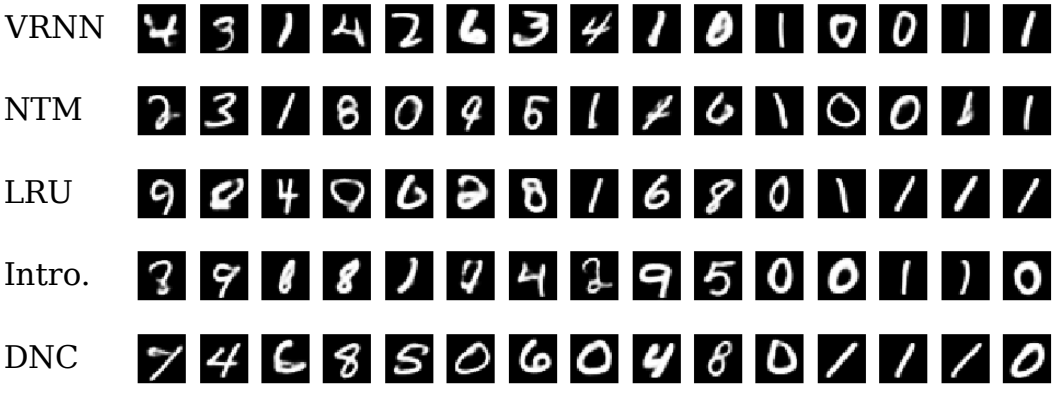


Figure 22: Parity recall with $l = 10$, $k = 5$. Over the last 5 frames, a generated 0 indicates should correspond to an even digit in the first five frames, and a generated 1 should correspond to an odd digit.

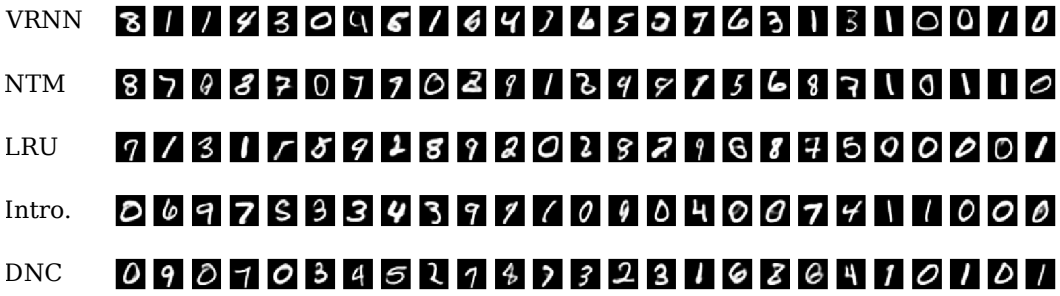


Figure 23: Parity recall with $l = 20$, $k = 5$. Same as above.

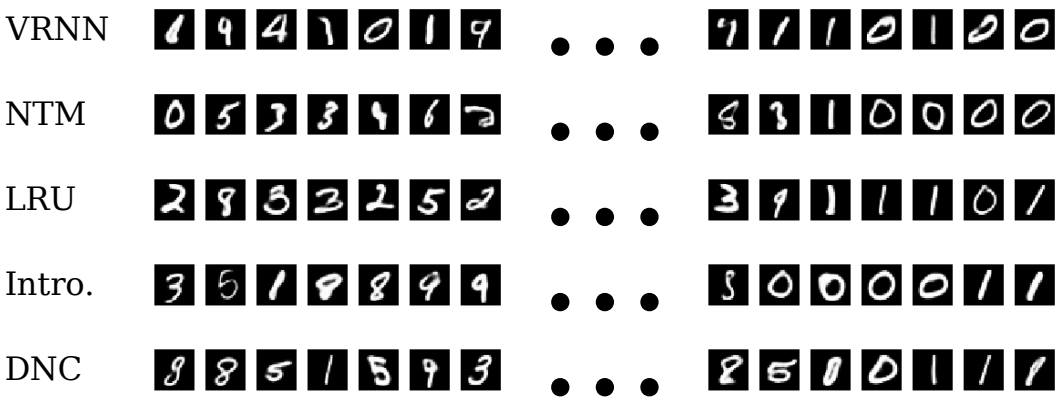


Figure 24: Parity recall with $l = 50$, $k = 5$.

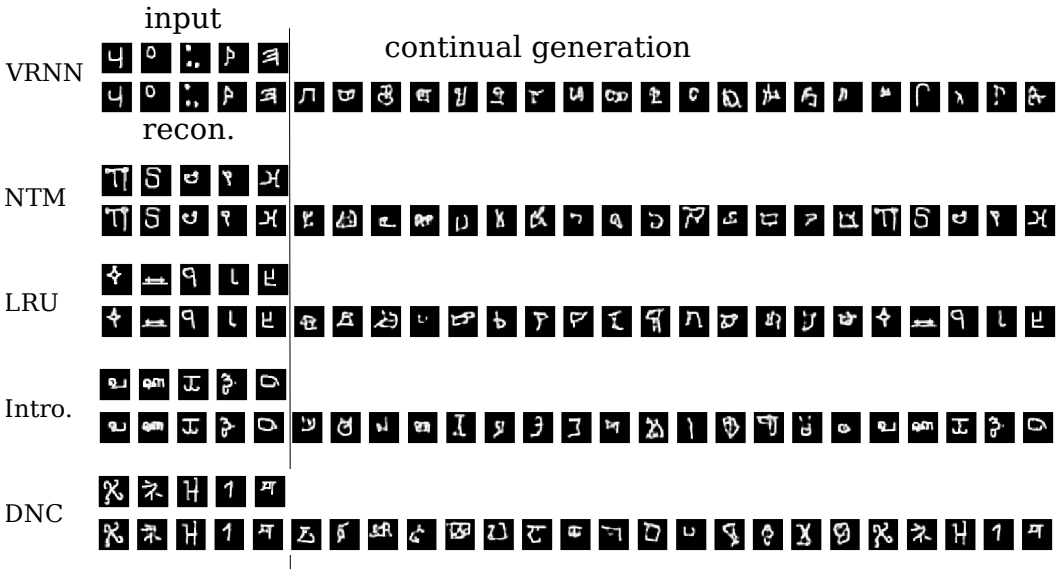


Figure 25: One-shot recall with $l = 20$, $k = 5$. The last 5 generated images should match the first 5, even though these images were held out from the training set.

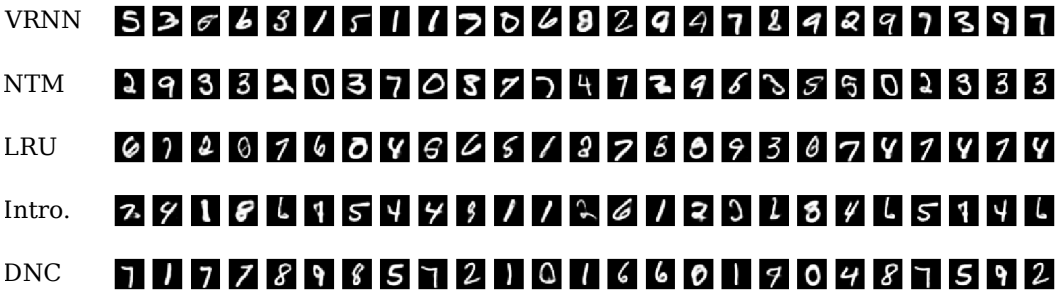


Figure 26: Dynamic dependency task with $l = 20$, $k = 5$. Starting from the fifth-to-last frame, the numeric value of each digit should indicate the temporal position of the digit to retrieve next.

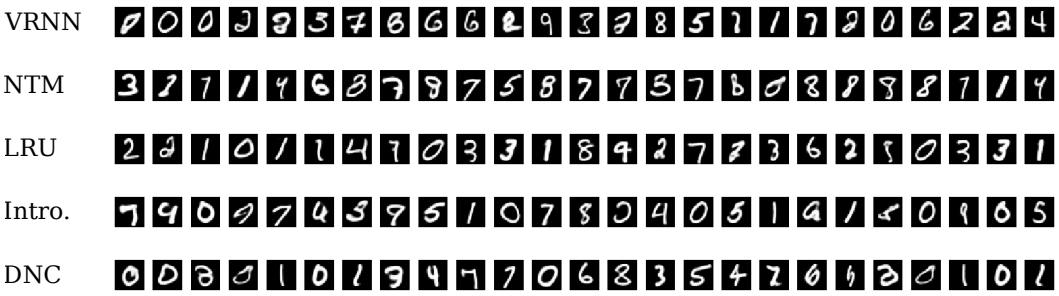


Figure 27: Content-based recall with $l = 20$, $k = 5$. The fifth-to-last frame is a repeat of a frame in the pre-recall phase. A working model should have continued to generate the same sequence from there.

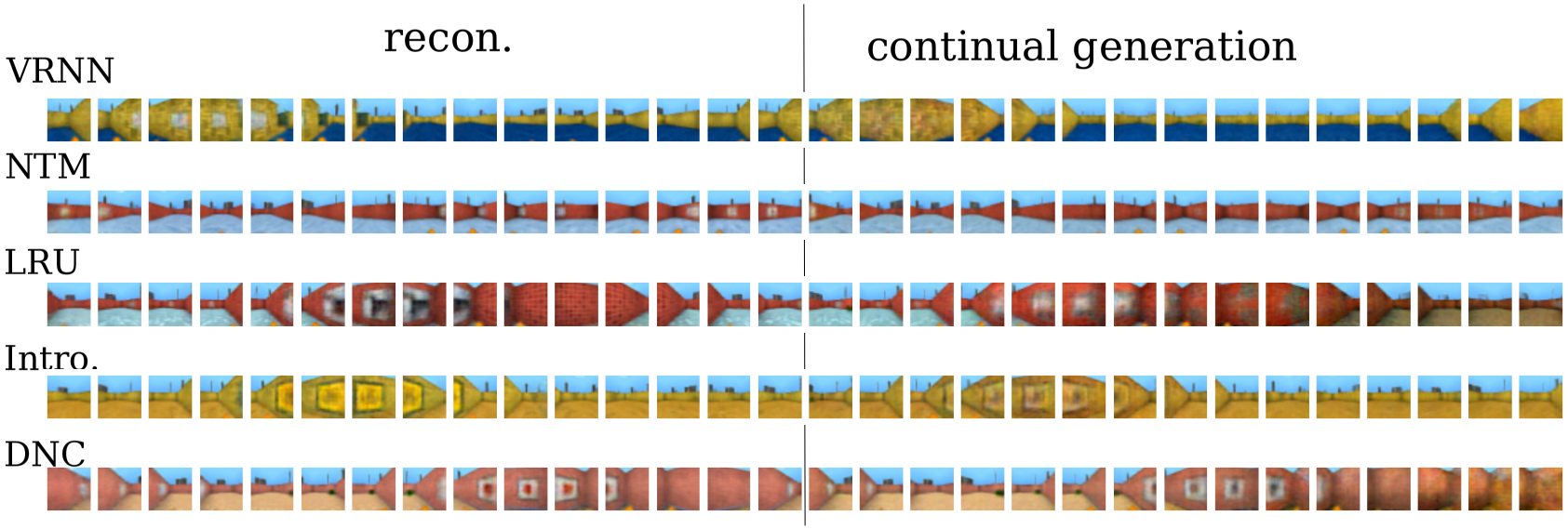


Figure 28: Rotation in a complex environment. The models observed 15 frames from the environment during one rotation, then generated corresponding frames during a second rotation. The VRNN model clearly forgot information from the first rotation, but the other models clearly demonstrated memory of paintings on the walls, floor colour, and buildings on the skyline.

D Pseudocode for Generating Artificial Sequences with Characters and Digits

Algorithm 1 Training Sequence Generation for Perfect Recall/One-shot Recall Tasks

procedure PERFECTRECALL/ONESHOTRECALL
  generator ← instance generator from dataset (MNIST/OmniGlot)
  l ← length of random sequence
  k ← length of recall interval
  sequence ← {}
  ▷ Pick a random sequence of instances from the dataset
  for $i = 1, l$ **do**
    $sequence[i] \leftarrow s \sim generator()$
  ▷ Repeat instances from the beginning of the sequence
  for $j = 1, k$ **do**
    append(*sequence*, $sequence[j]$)
  return *sequence* ▷ Return the full sequence

Algorithm 2 Training Sequence Generation for Parity Recall Task

procedure PARITYRECALL
  generator ← instance generator from dataset (MNIST)
  l ← length of random sequence
  k ← length of recall interval
  sequence, *labels* ← {}, {}
  ▷ Pick a random sequence of instances from
  ▷ the dataset and their labels
  for $i = 1, l$ **do**
    $sequence[i], labels[i] \leftarrow s, l_s \sim generator()$
  ▷ Append random 0 or 1 instances based on the parity of the
  ▷ labels of instances from beginning of the sequence
  for $j = 1, k$ **do**
    append(*sequence*, $s \sim generator(label = \text{parity}(labels[j]))$)
  return *sequence* ▷ Return full sequence

Algorithm 3 Training Sequence Generation for Dynamic Dependency Task

procedure DYNAMICDEPENDENCY
  generator ← instance generator from dataset (MNIST)
  l ← length of random sequence
  k ← length of recall interval
  sequence, *labels* ← {}, {}
  ▷ Pick a random sequence of instances from
  ▷ the dataset and their labels
  for $i = 1, l$ **do**
    $sequence[i], labels[i] \leftarrow s, l_s \sim generator()$
  ▷ Use the label of the previous instance as an 0-based *address* and
  ▷ append the instance located in that *address* in the sequence
  for $j = 1, k$ **do**
    append(*sequence*, *sequence*[*labels*[*l*+*j*-*1*]])
  return *sequence* ▷ Return full sequence

Algorithm 4 Training Sequence Generation for Similarity-Cued Recall Tasks

procedure SIMILARITYBASEDDEPENDENCY
  generator ← instance generator from dataset (MNIST)
  l ← length of random sequence
  k ← length of recall interval
  sequence ← {}
  ▷ Pick a random sequence of instances from the dataset
  for $i = 1, l$ **do**
    $sequence[i] \leftarrow s \sim generator()$
  ▷ Uniformly choose a random *sub*-sequence and
  ▷ append it to the end of the sequence
  $r \sim \text{Uniform}[1, l - k]$
  sub-sequence ← $sequence[r : r + k]$
  append(*sequence*, *sub-sequence*)
  return *sequence* ▷ Return full sequence