entropy

MDPI

Article

Hidden Markov Neural Networks

Lorenzo Rimella [1,2,*] and Nick Whiteley [3]

[1] Dipartimento di Scienze Economico-Sociali e Matematico-Statistiche, University of Torino, 10124 Torino, Italy
[2] Statistics Initiative, Collegio Carlo Alberto, 10124 Torino, Italy
[3] School of Mathematics, University of Bristol, Fry Building, Woodland Road, Bristol BS8 1UG, UK; nick.whiteley@bristol.ac.uk
* Correspondence: lorenzo.rimella@unito.it

Abstract: We define an evolving in-time Bayesian neural network called a Hidden Markov Neural Network, which addresses the crucial challenge in time-series forecasting and continual learning: striking a balance between adapting to new data and appropriately forgetting outdated information. This is achieved by modelling the weights of a neural network as the hidden states of a Hidden Markov model, with the observed process defined by the available data. A filtering algorithm is employed to learn a variational approximation of the evolving-in-time posterior distribution over the weights. By leveraging a sequential variant of Bayes by Backprop, enriched with a stronger regularization technique called variational DropConnect, Hidden Markov Neural Networks achieve robust regularization and scalable inference. Experiments on MNIST, dynamic classification tasks, and next-frame forecasting in videos demonstrate that Hidden Markov Neural Networks provide strong predictive performance while enabling effective uncertainty quantification.

Keywords: Bayesian neural networks; Hidden Markov models; variational inference

Academic Editors: Friedhelm Schwenker, Martin Trapp and Pierre Alquier

Received: 23 December 2024
Revised: 31 January 2025
Accepted: 4 February 2025
Published: 5 February 2025

Citation: Rimella, L.; Whiteley, N. Hidden Markov Neural Networks. *Entropy* **2025**, *27*, 168. https://doi.org/10.3390/e27020168

1. Introduction

Hidden Markov models (HMMs) are an efficient statistical tool for identifying patterns in dynamic datasets, with applications ranging from speech recognition [1] to computational biology [2]. Neural networks (NNs) are currently the most popular models in machine learning and artificial intelligence, demonstrating outstanding performance across several fields. In this paper, we propose a novel hybrid model called a Hidden Markov Neural Network (HMNN), which combines Factorial Hidden Markov models [3] and neural networks.

Intuitively, we aim to perform Bayesian inference on a time-evolving NN. However, computing the posterior distribution over the weights of even a static NN is a complex and generally intractable task. The extensive literature on variational Bayes [4] and its success in Bayesian inference for NNs [5–7] has motivated us to adopt this technique in HMNNs. In particular, the resulting procedure becomes the sequential counterpart of the Bayes by the Backprop algorithm proposed by [7]. As in [7], the reparameterization trick [6] plays a pivotal role in generating unbiased and low-variance estimates of the gradient.

HMNNs are particularly suited for time-series forecasting and continual learning [8]. As noted by [9], much of the research in this area has focused on preventing forgetting [10–12]. However, sudden changes in the statistical properties of the data may be an intrinsic feature of the generating process. In such cases, preserving all prior knowledge is not desirable, and it becomes necessary to forget irrelevant information. This can be achieved through the application of a Markov transition kernel [9], which can be interpreted

as the transition kernel of an HMM. In this sense, HMNNs adopt the adaptation idea proposed by [9] and extend it by leveraging the well-established framework of HMMs, further generalizing it to a broader class of variational approximations and stochastic kernels.

An alternative to variational Bayes is particle approximation via Sequential Monte Carlo [13]. However, these algorithms are known to suffer from the curse of dimensionality [14], making them unfeasible even for simple NN architectures.

2. Hidden Markov Neural Networks

For a time horizon $T \in \mathbb{N}$, a Hidden Markov model (HMM) is a bivariate process composed of an unobserved Markov chain $(W_t)_{t=0,\dots,T}$, called the hidden process, and a collection of observations $(\mathcal{D}_t)_{t=1,\dots,T}$, called the observed process, where the single observation at t is conditionally independent of all the other variables given the hidden process at the corresponding time step. We consider the case where the latent state-space is $\mathbb{R}^V$, with V finite set, and $\mathcal{D}_t$ is valued in $\mathbb{D}$ whose form is model-specific (discrete, $\mathbb{R}^d$ with $d \in \mathbb{N}$, etc.). To describe the evolution of an HMM, three quantities are needed: the initial distribution, the transition kernel and the emission distribution. We use $\lambda_0(\cdot)$ for the probability density function of W_0 (initial distribution). We write $p(W_{t-1}, \cdot)$ for the conditional probability density function of W_t given W_{t-1} (transition kernel of the Markov chain). We call $g(W_t, \mathcal{D}_t)$ the conditional probability mass or density function of $\mathcal{D}_t$ given W_t (emission distribution). We remark that HMM is often used for finite state spaces, while state-space model is used for continuous state spaces; here, following [14], we consider the two terminologies exchangeable.

We introduce a novel HMM called a Hidden Markov Neural Network (HMNN) where the hidden process describes the evolution of the weights of a neural network. For instance, in the case of feed-forward neural networks, the finite set V collects the location of each weight and $v \in V$ can be thought of as a triplet (l, i, j), indicating that the weight W_t^v is a weight of the NN at time t, and is precisely related to the connection of the hidden unit i (or input feature i if $l - 1 = 0$) in the layer $l - 1$ with the hidden unit j in the layer l (which might be the output layer). In such a model, we also assume that the weights evolve independently from each other, meaning that the transition kernel factorizes as follows:

$$p(W_{t-1}, W_t) = \prod_{v \in V} p^v(W_{t-1}^v, W_t^v), \quad W_{t-1}, W_t \in \mathbb{R}^V. \tag{1}$$

Under this assumption, an HMNN is a well-known class of HMM called a Factorial Hidden Markov model (FHMM) [3,15,16]; see Figure 1 for a graphical representation. Note that this factorization is key to ensuring that the computational cost of performing inference does not exponentially increase, and, in some scenarios, it allows us to perform close-form calculations; see Section 2.3. As an alternative, one could use a block structure [14], which requires, however, for the user to keep track of the correlations within blocks.

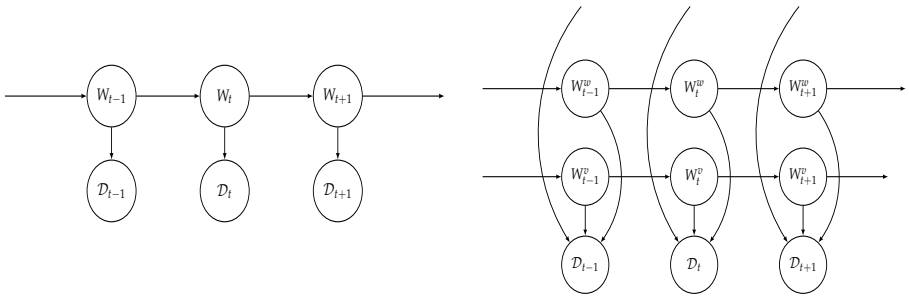

Figure 1. On the left: The conditional independence structure of an HMM. On the right: The conditional independence structure of an FHMM.

There is no restriction on the form of the neural network or the data $\mathcal{D}_t$; however, we focus on a feed-forward neural network framework and on a supervised learning scenario where the observed process is composed by an input x_t and output y_t, such that the neural network associated to the weights W_t maps the input onto a probability distribution on the space of the output, which represents the emission distribution $g(W_t, \mathcal{D}_t)$.

2.1. Filtering Algorithm for HMNN

The filtering problem aims to compute the conditional distributions of W_t given $\mathcal{D}_1, \ldots, \mathcal{D}_t$, which is called filtering distribution. In this paper, we use the operator notation from [14], which gives us a way to compactly represent the filtering algorithm. We refer to [13,14] for a thorough review.

We denote the filtering distribution with π_t and, ideally, we can compute it with a forward step through the data:

$$\pi_0 \coloneqq \lambda_0, \qquad \pi_t \coloneqq \mathsf{C}_t \mathsf{P} \pi_{t-1}, \tag{2}$$

where P and C_t are called the prediction operator and correction operator, respectively, and they are defined as follows:

$$\mathsf{P}\pi_{t-1}(A) \coloneqq \int \mathbb{I}_A(W_t) p(W_{t-1}, W_t) \pi_{t-1}(W_{t-1}) dW_{t-1} dW_t, \tag{3}$$

$$\mathsf{C}_t \pi_t(A) \coloneqq \frac{\int \mathbb{I}_A(W_t) g(W_t, \mathcal{D}_t) \pi_t(W_t) dW_t}{\int g(W_t, \mathcal{D}_t) \pi_t(W_t) dW_t}, \tag{4}$$

with π_{t-1}, π_t as the probability density functions, $\mathbb{I}.(\cdot)$ as the indicator function, and A as the set in the sigma field of $\mathbb{R}^V$. Throughout this paper, we refer to our target distribution as the filtering distribution π_t, which is indeed the time-evolving posterior distribution over the weights of our NN. Recursion (2) is intractable for any non-linear architecture of the underlying neural network. As a solution, we can apply variational inference, which allows us to approximate a posterior distribution when operations cannot be performed in closed form. The use of variational inference is then crucial to allow us to use any activation function in our NN.

Variational inference can be used to sequentially approximate the target distribution π_t with a variational approximation q_{θ_t} belonging to a pre-specified class of distributions $\mathcal{Q}$. The approximate distribution q_{θ_t} is uniquely identified inside the class of distributions by a vector of parameters θ_t, which is chosen to minimize a Kullback–Leibler (**KL**) divergence criteria:

$$q_{\theta_t} \coloneqq \arg\min_{q_\theta \in \mathcal{Q}} \mathbf{KL}(q_\theta || \pi_t) = \arg\min_{q_\theta \in \mathcal{Q}} \mathbf{KL}(q_\theta || \mathsf{C}_t \mathsf{P} \pi_{t-1}), \tag{5}$$

where the Kullback–Leibler divergence can be rewritten as

$$\mathbf{KL}(q_\theta || \mathsf{C}_t \mathsf{P} \pi_{t-1}) = const. + \mathbf{KL}(q_\theta || \mathsf{P}\pi_{t-1}) - \mathbb{E}_{q_\theta(w)}[\log(g(w, \mathcal{D}_t))], \tag{6}$$

because of the properties of the correction operator (4). From the above representation, we can also observe that minimizing the Kullback–Leibler divergence is equivalent to maximizing the evidence lower bound (**ELBO**):

$$\mathbf{ELBO}(q_\theta; \pi_{t-1}, \mathcal{D}_t) = \mathbb{E}_{q_\theta(w)}[\log(g(w, \mathcal{D}_t))] - \mathbf{KL}(q_\theta || \mathsf{P}\pi_{t-1}). \tag{7}$$

Sequential training using (5) is again intractable because it requires the filtering distribution at time $t-1$, i.e., π_{t-1}. Although under good optimization, we could consider $q_{\theta_t} \approx \pi_t$

when π_{t-1} is known, similarly, $q_{\theta_{t-1}} \approx \pi_{t-1}$ when π_{t-2} is known, and so on. Given that π_0 is our prior knowledge λ_0 on the weights before training, we can find a q_{θ_1} that approximates π_1 using (5), and then we can propagate forward our approximation by following the previous logic. In this way, an HMNN is trained sequentially on the same flavour of (5), by substituting the optimal filtering with the last variational approximation. As for the optimal procedure, we define an approximated filtering recursion, where $\tilde{\pi}_t$ stands for the sequential variational approximation of π_t:

$$\tilde{\pi}_0 := \lambda_0, \quad \tilde{\pi}_t := \mathsf{V}_{\mathcal{Q}} \mathsf{C}_t \mathsf{P} \tilde{\pi}_{t-1}, \tag{8}$$

where the operators P, C_t are as in recursion (2) and the operator $\mathsf{V}_{\mathcal{Q}}$ is defined as follows:

$$\mathsf{V}_{\mathcal{Q}} \rho := \arg\min_{q_\theta \in \mathcal{Q}} \mathbf{KL}(q_\theta || \rho), \tag{9}$$

with ρ being a probability distribution and $\mathcal{Q}$ being the class of variational distributions, e.g., in (8), we have $\rho = \mathsf{C}_t \mathsf{P} \tilde{\pi}_{t-1}$. Note that, in this final notation, we are hiding the dependence on the variational parameters θ.

Observe that this approach follows the assumed density filter paradigm [17,18], where the true posterior is projected onto a family of distributions which are easy to work with, and then propagated forward.

2.2. Sequential Reparameterization Trick

The minimization procedure exploited in recursion (8) cannot be solved in a closed form and we propose to find a suboptimal solution through gradient descent. Consider a general time step t, where we want to approximate π_t with $\tilde{\pi}_t = q_{\theta_t}$. This requires an estimate of the gradient of $\mathbf{KL}(q_\theta || \mathsf{C}_t \mathsf{P} \tilde{\pi}_{t-1})$. As explained in [7], if $W \sim q_\theta$ can be rewritten as $\epsilon \sim \nu$ through a deterministic transformation h, i.e., $W = h(\theta, \epsilon)$, then

$$\begin{aligned} \frac{\partial \mathbf{KL}(q_\theta || \mathsf{C}_t \mathsf{P} \tilde{\pi}_{t-1})}{\partial \theta} =& \mathbb{E}_{\nu(\epsilon)} \left[\frac{\partial \log(q_\theta(W))}{\partial W} \frac{\partial W}{\partial \theta} + \frac{\partial \log(q_\theta(W))}{\partial \theta} \right]_{W=h(\theta,\epsilon)} \\ & - \mathbb{E}_{\nu(\epsilon)} \left[\frac{\partial \log(\mathsf{P} \tilde{\pi}_{t-1}(W))}{\partial W} \frac{\partial W}{\partial \theta} \right]_{W=h(\theta,\epsilon)} \\ & - \mathbb{E}_{\nu(\epsilon)} \left[\frac{\partial \log(g(W, \mathcal{D}_t))}{\partial W} \frac{\partial W}{\partial \theta} \right]_{W=h(\theta,\epsilon)} \end{aligned} \tag{10}$$

where we used (6) to simplify the form of the equation (see Appendix A). Given (10), we can estimate the expectation $\mathbb{E}_{\nu(\epsilon)}$ via straightforward Monte Carlo sampling:

$$\begin{aligned} \frac{\partial \mathbf{KL}(q_\theta || \mathsf{C}_t \mathsf{P} \tilde{\pi}_{t-1})}{\partial \theta} \approx & \frac{1}{N} \sum_{i=1}^{N} \Bigg\{ \left[\frac{\partial \log(q_\theta(W))}{\partial W} \frac{\partial W}{\partial \theta} + \frac{\partial \log(q_\theta(W))}{\partial \theta} \right]_{W=h(\theta,\epsilon)} \\ & - \left[\frac{\partial \log(\mathsf{P} \tilde{\pi}_{t-1}(W))}{\partial W} \frac{\partial W}{\partial \theta} \right]_{W=h(\theta,\epsilon)} \\ & - \left[\frac{\partial \log(g(W, \mathcal{D}_t))}{\partial W} \frac{\partial W}{\partial \theta} \right]_{W=h(\theta,\epsilon)} \Bigg\}_{\epsilon=\epsilon^{(i)}} \end{aligned} \tag{11}$$

with $\epsilon^{(i)} \sim \nu$ and N the size of the Monte Carlo sample. Given the Monte Carlo estimate of the gradient, we can then update the parameters θ_t, related to the variational approximation at time t, according to any gradient descent technique. Algorithm 1 displays this procedure and, for the sake of simplicity, we write the algorithm with an update that follows a vanilla gradient descent.

Algorithm 1 Approximate filtering recursion

Set: $\tilde{\pi}_0 = \lambda_0$
for $t = 1, \dots, T$ **do**
  Initialize: θ_t
  repeat
    $\epsilon^{(i)} \sim \nu, \quad i = 1, \dots, N$
    Estimate the gradient ∇ with (11) evaluated in $\theta = \theta_t$
    Update the parameters: $\theta_t = \theta_t - l\nabla$
  until Maximum number of iterations
  Set: $\tilde{\pi}_t = q_{\theta_t}$
end for
Return: $(\theta_t)_{t=1,\dots,T}$

As suggested in the literature [5,7], the cost function in (6) is suitable for minibatch optimization. This might be useful when, at each time step, $\mathcal{D}_t$ is made of multiple data, and so, a full computation of the gradient is computationally prohibitive.

2.3. Gaussian Case

A fully Gaussian model, i.e., when both the transition kernel and the variational approximation are Gaussian distributions, is not only convenient because the form of $h(\theta, \epsilon)$ is trivial, but also because there exists a closed-form solution for $\mathsf{P}\tilde{\pi}_{t-1}$. Another appealing aspect of the Gaussian choice is that similar results hold for the scale mixture of Gaussians, which allows us to use a more complex variational approximation and a transition kernel of the same form as the prior distribution in [7]. We start by considering the variational approximation. We choose $q_\theta := \bigotimes_{v \in V} q_\theta^v$, where q_θ^v is a mixture of Gaussian with parameters $\theta^v = (m^v, s^v)$ and γ^v hyperparameter. Precisely, for a given weight W^v of the feed-forward neural network,

$$q_\theta^v(W^v) := \gamma^v \mathcal{N}\left(W^v | m^v, (s^v)^2\right) + (1 - \gamma^v)\mathcal{N}\left(W^v | 0, (s^v)^2\right), \tag{12}$$

where $\gamma^v \in (0, 1]$, $m^v \in \mathbb{R}$, $(s^v)^2 \in \mathbb{R}_+$, and $\mathcal{N}\left(\cdot | \mu, \sigma^2\right)$ is the Gaussian density with mean μ and variance σ^2. We refer to this technique as variational DropConnect because it can be interpreted as setting the weight in position v of the neural network around zero with probability $1 - \gamma^v$, and so, it plays a role of regularization similar to [19]. Under variational DropConnect, the deterministic transformation $h(\theta, \epsilon)$ is still straightforward. Indeed, given that q_θ factorizes, then $h(\theta, \epsilon) = (h^v(\theta^v, \epsilon^v))_{v \in V}$ (each W^v depends only on θ^v) and W^v is distributed as (12), which is equivalent to consider

$$W^v = \eta^v m^v + \xi^v s^v, \quad \text{with } \eta^v \sim \mathrm{Be}(\cdot | \gamma^v),\ \xi^v \sim \mathcal{N}(\cdot | 0, 1), \tag{13}$$

where $\mathrm{Be}(\cdot | p)$ is the Bernoulli density with parameter p. Observe that the distribution of (13) is (12), as γ^v comes from the Bernoulli random variable η^v, which activates or not the mean m^v, while the ξ^v represents the Gaussian term. Hence, $h^v(\theta^v, \epsilon^v) = \eta^v m^v + \xi^v s^v$, where $\theta^v = (m^v, s^v)$ and $\epsilon^v = (\eta^v, \xi^v)$ with η^v Bernoulli with parameter γ^v and ξ^v standard Gaussian, meaning that we just need to sample from a Bernoulli and a Gaussian distribution independently. The collection of hyperparameters $(\gamma^v)_{v \in V}$ represents the variational DropConnect rate per each weight in the NN and we generally choose $\gamma^v = \gamma \in (0, 1]$ per each $v \in V$, i.e., we have a global regularization parameter. Note that $(\gamma^v)_{v \in V}$ must be considered as fixed and cannot be learned during training, because from (10), we need

the distribution of ϵ to not be dependent on the learnable parameters. Consider now the transition kernel. It is chosen to be a scale mixture of Gaussians with parameters $\phi, \alpha, \sigma, c, \mu$:

$$\begin{aligned} p(W_{t-1}, W_t) := \phi \mathcal{N}\Big(W_t | \mu + \alpha(W_{t-1} - \mu), \sigma^2 \mathbf{I}_V\Big) \\ + (1-\phi)\mathcal{N}\Big(W_t | \mu + \alpha(W_{t-1} - \mu), (\sigma^2/c^2)\mathbf{I}_V\Big), \end{aligned} \tag{14}$$

where $\phi \in [0,1]$, $\mu \in \mathbb{R}^V$, $\alpha \in [0,1)$, $\sigma \in \mathbb{R}_+$, $\mathbf{I}_V$ is the identity matrix on $\mathbb{R}^{V,V}$, $c \in \mathbb{R}_+$ and $c > 1$. Intuitively, the transition kernel tells us how we are expecting the weights to be in the next time step given the states of the weights at the current time. We can interpret it, along with the previous variational approximation, as playing the role of an evolving prior distribution which constrains the new posterior distribution in regions that are determined from the previous training step. The choice of the transition kernel is crucial. A too conservative kernel would constrain too much training and the algorithm would not be able to learn patterns in new data. On the contrary, a too flexible kernel could just forget what was learned previously and adapt to the new data only.

As we are considering Gaussian distribution, we can solve $\mathsf{P}\tilde{\pi}_{t-1}$ in closed form, and, precisely, we obtain another scaled mixture of Gaussian distributions. We can then directly work on the Gaussian density $\mathsf{P}\tilde{\pi}_{t-1}(W_{t-1})$, which is a product over $v \in V$ of scaled mixture of Gaussian densities. Specifically, consider a general weight $v \in V$ and call $(\mathsf{P}\tilde{\pi}_{t-1})^v(W^v_{t-1})$ the marginal density of $\mathsf{P}\tilde{\pi}_{t-1}$ on the component v. If m^v_{t-1}, s^v_{t-1} are the estimates of m^v, s^v at time $t-1$, then

$$\begin{aligned} (\mathsf{P}\tilde{\pi}_{t-1})^v(W^v_{t-1}) =& \gamma^v \phi \mathcal{N}\Big(W^v_{t-1} | \mu^v - \alpha(\mu^v - m^v_{t-1}), \sigma^2 + \alpha^2 (s^v_{t-1})^2\Big) \\ &+ (1-\gamma^v)\phi \mathcal{N}\Big(W^v_{t-1} | \mu^v - \alpha\mu^v, \sigma^2 + \alpha^2 (s^v_{t-1})^2\Big) \\ &+ \gamma^v(1-\phi)\mathcal{N}\Big(W^v_{t-1} | \mu^v - \alpha(\mu^v - m^v_{t-1}), \sigma^2/c^2 + \alpha^2 (s^v_{t-1})^2\Big) \\ &+ (1-\gamma^v)(1-\phi)\mathcal{N}\Big(W^v_{t-1} | \mu^v - \alpha\mu^v, \sigma^2/c^2 + \alpha^2 (s^v_{t-1})^2\Big). \end{aligned} \tag{15}$$

We can observe that (15) is again a scale mixture of Gaussians, where all the variances are influenced by the variances at the previous time step according to α^2. On the one hand, the variational DropConnect rate γ^v tells how to scale the mean of the Gaussians according to the previous estimates m^v_{t-1}. On the other hand, ϕ controls the entity of the jumps by allowing the weights to stay in place with a small variance σ^2/c^2 and permitting big jumps with σ^2 if necessary. As in [7], when learning s_t, we use the transformation $\tilde{s}_t$ such that $s_t = \log(1 + \exp(\tilde{s}_t))$.

Across Section 2, we have declared multiple quantities which we summarize in Table 1.

Table 1. Notation summary.

Notation	Meaning
$W_t, \mathcal{D}_t$	hidden weights of an NN and observed data at time t
$p(W_{t-1}, W_t)$	Markov transition kernel of the weights of the NN
$g(W_t, \mathcal{D}_t)$	probability distribution of the data given the weights
v	a weight of the NN, also used to represent marginal quantities
$\pi_t, \tilde{\pi}_t$	filtering distribution and its approximation at time t
P, C_t	prediction operator and correction operator at time t
$\mathsf{V}_{\mathcal{Q}}$	operator that minimizes the KL-divergence on the class $\mathcal{Q}$
q_θ, θ	variational approximation and its parameters
$h(\theta, \cdot)$	transformation function in the reparameterization trick

Table 1. *Cont.*

Notation	Meaning
γ^v	scale mixture of Gaussian probability of weight v
m_t^v	scale mixture of Gaussian mean of weight v at time t
s_t^v	scale mixture of Gaussian standard dev. of weight v at time t
ϕ	scale mixture of Gaussian probability of transition kernel
μ	scale mixture of Gaussian stationary mean of transition kernel
α	scale mixture of Gaussian mean scaling of transition kernel
σ	scale mixture of Gaussian big-jump standard dev. of transition kernel
σ/c	scale mixture of Gaussians small-jump standard dev. of transition kernel

2.4. Performance and Uncertainty Quantification

After running Algorithm 1, we obtain a sequence of variational approximations $\tilde{\pi}_t = q_{\theta_t}$ for our filtering distributions π_t. These approximations represent the posterior distribution of W_t, meaning that we are approximating the distribution of $W_t|\mathcal{D}_1, \ldots, \mathcal{D}_t$ with a known distribution. Given a realization of W_t, we can compute $g(W_t, \cdot)$, which represents the likelihood at time t or, equivalently, the performance of the neural network (NN) with weights W_t. We generally assume the ability to compute the mean of $\tilde{\pi}_t = q_{\theta_t}$, i.e., the posterior mean, and to sample from it, i.e., obtain posterior samples. These two components allow us to assess the performance of our HMNN over time and evaluate how certain we are about that performance.

In our experiments, we typically refer to performance when evaluating our time-evolving NN using the posterior mean, $\mathbb{E}_{\tilde{\pi}_t(W_t)}[W_t]$. Conversely, when assessing uncertainty, we consider posterior samples, $W_t^{(1)}, \ldots, W_t^{(N)}$.

3. Related Work

3.1. Combining NNs and HMMs

Multiple attempts have been made in the literature to combine HMMs and NNs. In [20], an NN is trained to approximate the emission distribution of an HMM. Refs. [21,22] preprocess the data with an NN and then use the output as the observed process of a discrete HMM. Ref. [23] proposes "hidden neural networks" where NNs are used to parameterize Class HMM, an HMM with a distribution over classes assigned to each state. Other recent works include [24–26]. In neuroscience, Ref. [27] explores the idea of updating measures of uncertainty over the weights in a mathematical model of a neuronal network as part of a "Bayesian Plasticity" hypothesis of how synapses take uncertainty into account during learning. However, they did not focus on artificial neural networks and the computational challenges of using them for data analysis when network weights are statistically modelled as being time-varying. More recent works also combined Kalman Filter [28,29] and state-space model [30] with deep learning, and can be interpreted as a subclass of HMNN, as they do not consider mixtures of Gaussians.

3.2. Bayesian DropConnect and DropOut

DropConnect [19,31] and DropOut [32,33] are well-known techniques to prevent NNs from overfitting. Ref. [34] proposes variational DropOut where the authors combined fully factorized Gaussian variational approximation with the local reparameterization trick to re-interpret DropOut with continuous noise as a variational method. Ref. [35] extensively treat the connections between DropOut and Gaussian processes, and they show how to train NNs with DropOut (or DropConnect [31]) in a variational Bayes setting. Our version

of variational DropConnect has several common aspects with the cited works, but the novelty resides in the regularization being induced by the variational approximation's choice and the corresponding reformulation of the reparameterization trick.

3.3. Bayesian Filtering

There are multiple examples of NN training through Bayesian filtering [36–39]. In particular, the recent work of [40] proposed AdaBayes and AdaBayes-SS, where updates resembling the Kalman filter are employed to model the conditional posterior distribution over a weight of an NN given the states of all the other weights. However, the main difference with HMNN is the dynamical evolution of the underlined NN, and indeed, Bayesian filtering methods, in this context, do not consider any change in time. Still from a Bayesian perspective, HMNN could also be used as time-evolving prior and speed-up training of static models [41].

3.4. Continual Learning

There are significant similarities between our work and continual learning methods. Here, we perform a quick overview of the most popular ones and we refer to [8] for a complete review. Elastic Weight Consolidation (EWC) [10] uses an L2-regularization that guarantees the weights of the NN for the new task being in the proximity of the ones from the old task. Variational continual learning (VCL) [11] learns a posterior distribution over the weights of an NN by sequentially approximating the true posterior distribution through variational Bayes and by propagating forward the previous variational approximation (this is like setting our transition kernel to a Dirac delta). Online Laplace approximation [12,42,43] proposes a recursive update for the parameters of a Gaussian variational approximation which involves a Hessian of the newest negative log-likelihood. None of the cited techniques builds dynamic models, and even if this could be solved by storing the weights at each training step, there is no forgetting, meaning that EWC, VCL, and Online Laplace focus on overcoming catastrophic forgetting and they are not able to avoid outdated information. Lastly, the most similar procedure to HMNNs is the one proposed by [9], where the authors performed model adaption with Bayes forgetting through the application of a stochastic kernel. However, HMMs are not even cited and the form of the kernel and variational approximation are restricted to Gaussian distributions, not mixtures.

4. Experiments

In this section, we test the performance of HMNNs. Similar to [7], we focus our study on simple feed-forward neural networks, leaving the exploration of more complex architectures for future work. Notably, in our experiments, the computational cost per time step is comparable to that of Bayes by Backprop.

Firstly, we empirically demonstrate that variational DropConnect, i.e., using (12) as a variational approximation, yields better performance than Bayes by Backprop [7] on MNIST [44]. Secondly, we provide an experiment to show how HMNNs work and how they can quantify uncertainty in a simple time-evolving classification setting built from the "two-moons" dataset [45]. We then highlight the ability of HMNNs to retrieve the evolution of true parameters in a conceptual drift scenario [9]. Specifically, we demonstrate that employing more complex variational approximations, compared to those used in [9], does not affect the retrieval process. Following this, we explore a more complex conceptual drift framework constructed from the MNIST dataset, comparing HMNNs to continual learning baselines [9–11]. Finally, we show that HMNNs can also be applied to one-step-ahead forecasting in time-series. Specifically, we address a next-frame prediction task in

the dynamic video texture of a waving flag [46–48]. Additional experimental details are provided in Appendix B.

The experiments were run on three different clusters: BlueCrystal Phase 4 (University of Bristol), Cirrus (one of the EPSRC Tier-2 National HPC Facilities), and The Cambridge Service for Data-Driven Discovery (CSD3) (University of Cambridge).

4.1. Variational DropConnect

The experiment aims to understand if using a Gaussian mixture as a variational approximation can help improve Bayes by Backprop. We trained on the MNIST dataset with the same setup from [7]. We considered a small architecture with the vectorized image as input, 2 hidden layers with 400 rectified linear units [49,50] and a softmax layer on 10 classes as output. We considered a fully Gaussian HMNN, as in Section 2.3, with $T = 1$, $\alpha = 0$ and $\mu = \mathbf{0}$ (kernel parameter), with $\mathbf{0}$ being the zero vector. Note that such an HMNN coincides with a single Bayesian neural network, meaning that we are simply training with Bayes by Backprop with the addition of variational DropConnect; see Appendix B for more details. We trained on about 50 combinations of the parameters $(\gamma^v, \phi, \sigma, c)$ and learning rate, which were randomly extracted from pre-specified grids.

Model selection was performed according to a validation score on a held-out validation set. We cluster the 50 combinations of the parameters by their values of γ^v and we report in Figure 2 the performance of the three best models per each γ^v value on the held-out validation set.

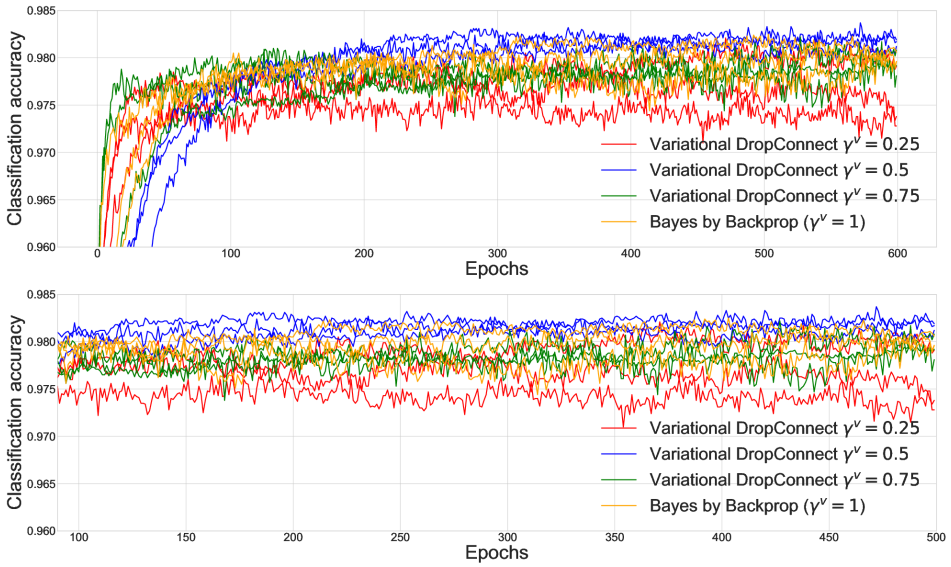

Figure 2. Performance on a validation set of a Bayes by Backprop with and without variational DropConnect. The plot on the bottom is a zoom-in of the plot on the top.

We then selected the best models per each γ^v and we computed the performance on a held-out test set; see Table 2. We found that values of $\gamma^v < 1$ led to better performance, motivating the use of variational DropConnect as a regularization technique.

Table 2. Classification accuracy (the bigger the better) of best runs for MNIST on a held-out test set (MNIST test set). $\gamma^v = 1$ refers to the case of Bayes by Backprop without variational DropConnect.

Parameter Value	Accuracy
$\gamma^v = 0.25$	0.9838
$\gamma^v = 0.5$	0.9827
$\gamma^v = 0.75$	0.9825
$\gamma^v = 1$ [7]	0.9814

4.2. Illustration: Two Moons Dataset

In this subsection, we provide an illustration of the HMNN on the "two moons" dataset from "scikit-learn" [45]. This synthetic dataset produces two half circles in the plane, which are binary-classified. To create a time dimension, we sequentially rotated these half circles. Specifically, over $t = 0, 1, 2, 3, 4$, we generated new data from the "two moons" dataset and then applied a rotation with an increasing angle by keeping the label as simulated. We also considered two scenarios: one where the "two moons" are well separated, and another where the "two moons" are overlapping; see Figure 3.

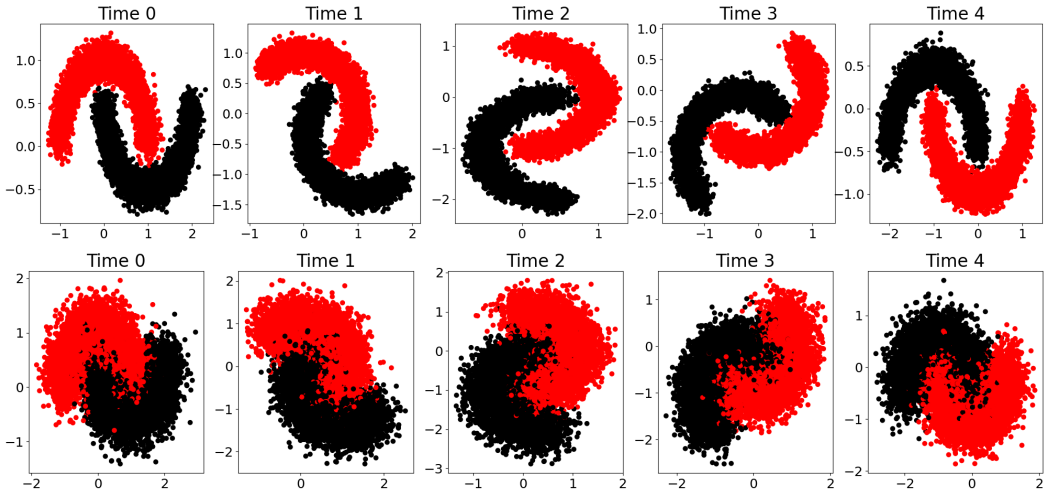

Figure 3. First row, well-separated "two moons". Second row, overlapping "two moons". Different columns are associated with different time steps. Different colours are associated with different labels.

Given the two training sets, a fully Gaussian HMNN was run for a fixed set of hyperparameters; see Appendix B. For the neural network architecture, we considered a bidimensional vector as input, i.e., (x, y) location, 2 hidden layers with 50 rectified linear units, and a softmax layer on 2 classes as output. The result was an evolving in-time Bayesian neural network that is able to associate to each location on the plane a probability of being in one class or the other, and also to quantify uncertainty on these probabilities. Note that it is enough to report one of the two probabilities as the other one is the complement.

We illustrate these results in Figure 4, where a 95% credible interval was obtained via Monte Carlo sampling on the weights of the HMNN; see Section 2.4. Here, we can observe that the scenario with overlapping moon shows a smoother transition between the two classification regions, meaning that we do not know which label to guess. In terms of uncertainty in the overlapping example, we are generally less confident about our probability estimates compared to the well-separated one. Moreover, in the well-separated example, there are some blank regions between two moons where the HMNN is completely uncertain about what to guess.

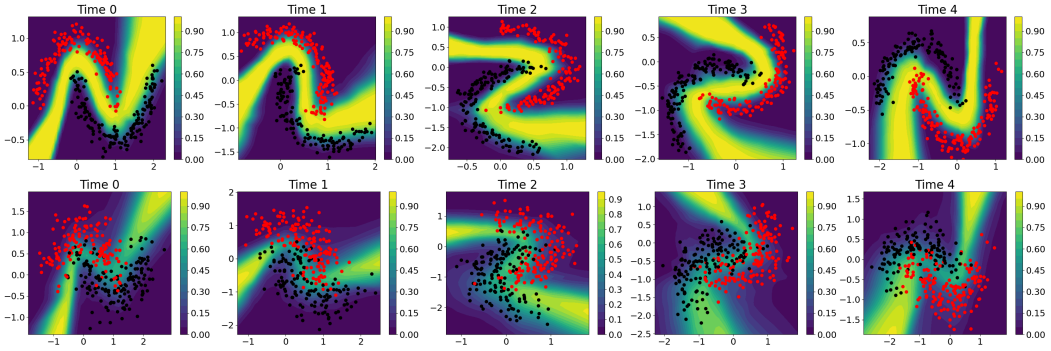

Figure 4. First row, well-separated "two moons". Second row, overlapping "two moons". Different columns are associated with different time steps. The plot shows the length of the 95% credible interval. The blue and yellow surface is the probability of prediction on the second class. Different coloured dots are associated with different labels.

This can be further checked by plotting the length of the credible interval; see Figure 5. We can observe that where the two classes overlap, the HMNN becomes less certain (bigger length of the credible interval) but it becomes completely uncertain (length close to one) in the region between the two moons and where no data are observed. Intuitively, with the yellow region in Figure 5, the HMNN expresses the following: "I do not know". We remark that this is one of the most important features of the Bayesian approach, which allows the model to be uncertain about the prediction and warn the user beforehand.

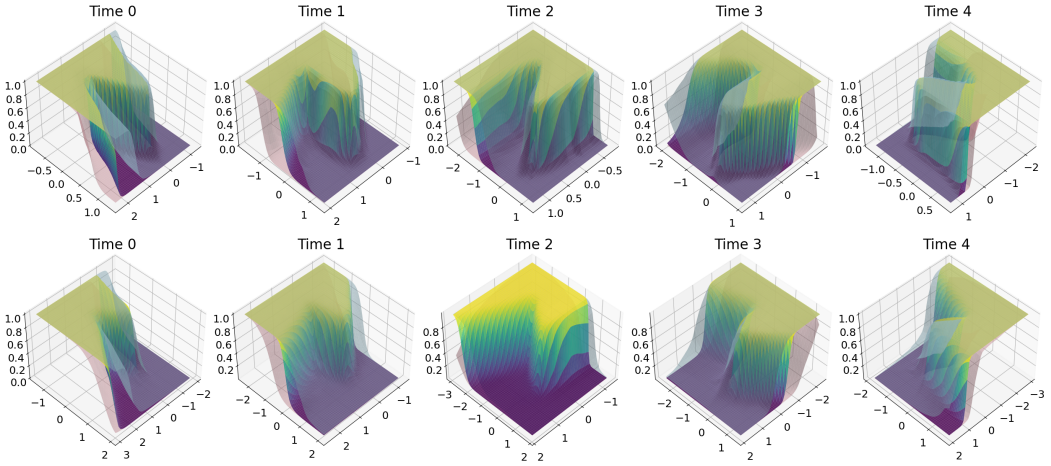

Figure 5. First row, well-separated "two moons". Second row, overlapping "two moons". Different columns are associated with different time steps. The blue and yellow surface is the probability prediction on the second class. Pink- and grey-shaded surfaces represent the 95% credible intervals.

4.3. Concept Drift: Logistic Regression

In this subsection, we compare the HMNN and the adaption with the Ornstein–Uhlenbeck process proposed by [9] on concept drift. As in [9], we consider a 2-dimensional logistic regression problem where the weights evolve in time: $w_t^{(1)} = 10\sin(at)$ and $w_t^{(2)} = 10\cos(at)$, where $a = 5$ deg/sec and $t = 1, \ldots, 700$. Precisely, per each time step, the data are generated in the following manner:

$$x_t^{(i)} \sim \mathrm{U}(\cdot|-3,3), \quad y_t^{(i)} \sim \mathrm{Be}\left(\cdot \middle| \mathbf{sigmoid}\left(x_t^{(i)}, w_t\right)\right), \tag{16}$$

where $\mathrm{U}(\cdot|a,b)$ is the uniform distribution over the interval $[a,b]$, $i = 1, \ldots, 10{,}000$ with 10,000 being the batch size per time step, $w_t = \left(w_t^{(1)}, w_t^{(2)}\right)$ and $\mathbf{sigmoid}\left(x_t^{(i)}, w_t\right)$ being the sigmoid function with weights w_t and input $x_t^{(i)}$.

Given the training set, a fully Gaussian HMNN is run under multiple combinations of hyperparameters. The experiments showed a recovery of the oscillating nature of the weights in each combination. Figure 6 reports the results of one of the considered hyperparameters' combinations and the findings from [9]. We can conclude that the HMNN is able, as for [9], to recover the sinusoidal curve of the true parameters even if the form of the posterior distribution approximation in the HMNN is chosen to be a mixture of Gaussians. Recall that the aim of this section was not classification accuracy but rather the recovery of the oscillating nature of the weights.

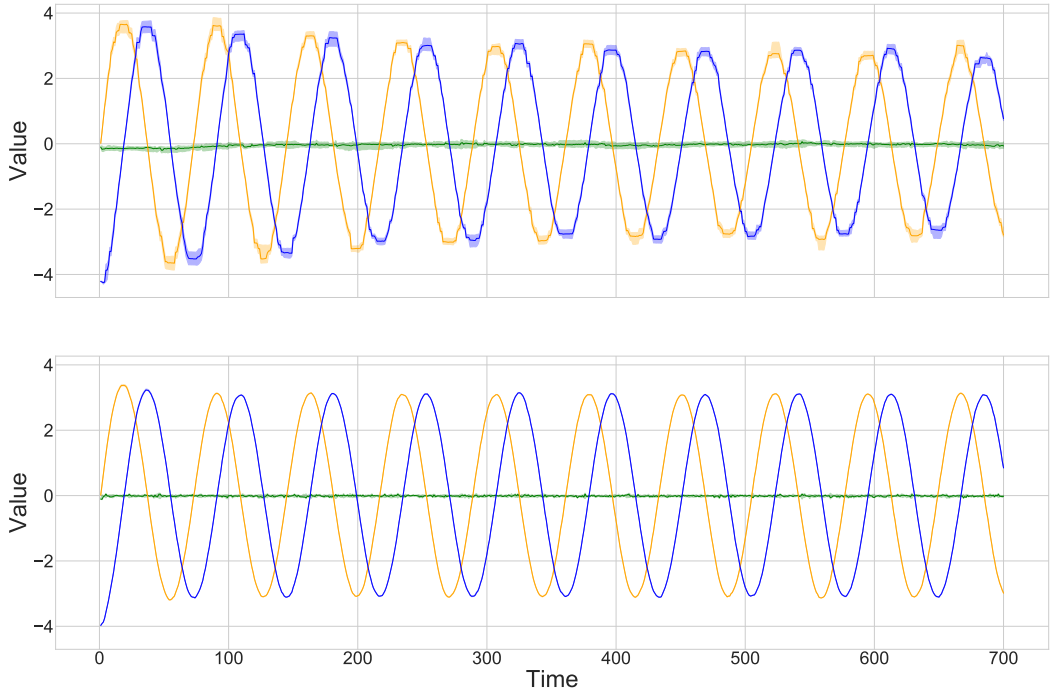

Figure 6. Mean of the approximate posterior distributions over 700 steps, where credible intervals were built from multiple runs. Orange stands for $w_t^{(1)}$, blue stands for $w_t^{(2)}$ and green is used for the bias. On the top, the HMNN. On the bottom, adaption with the Ornstein–Uhlenbeck process from [9].

4.4. Concept Drift: Evolving Classifier on MNIST

In this subsection, we compare HMNNs with continual learning baselines when the data-generating distribution is dynamic. We decide to artificially generate such a dataset from MNIST with the following procedure:

1. We define two labellers: $\mathcal{C}_1$, naming each digit with its label in MNIST; $\mathcal{C}_2$, labelling each digit with its MNIST's label shifted by one unit, i.e., 0 is classified as 1, 1 is classified as 2, . . . , 9 is classified as 0.
2. We consider 19 time steps where each time step t is associated with a probability $f_t \in [0, 1]$ and a portion of the MNIST's dataset $\mathcal{D}_t$.
3. At each time step t, we randomly label each digit in $\mathcal{D}_t$ with either $\mathcal{C}_1$ or $\mathcal{C}_2$ according to the probabilities $f_t, 1 - f_t$.

The resulting $(\mathcal{D}_t)_{t=1,\dots,19}$ is a collection of images where the labels evolve in t by switching randomly from $\mathcal{C}_1$ to $\mathcal{C}_2$ and vice versa. Validation and test sets are built similarly. In such a scenario, we would ideally want to be able to predict the correct labels by sequentially learning a classifier that is capable of inferring part of the information from the previous time step and forgetting the outdated one. Note that when $f_t = 0.5$, the best we can perform is a classification accuracy of 0.5 because $\mathcal{C}_1$ and $\mathcal{C}_2$ are indistinguishable.

We consider a fully Gaussian HMNN with $\mu = m_{t-1}$ (kernel parameter) to encourage a strong memory of the past. The evolving in-time NN is composed of the vectorized

image as input, 2 hidden layers with 100 rectified linear units and a softmax layer on 10 classes as output. The parameters α, γ^v are selected through the validation set while the other parameters are fixed before training. Along with the previous HMNN, we sequentially trained five additional models for comparison: variational continual learning (VCL), without coreset; Elastic Weight Consolidation (EWC), with tuning parameter chosen with the validation set; Bayes by Backprop trained sequentially on the dataset; Bayes by Backprop on the full dataset; and the adaption of Kurle et al.'s work with the Ornstein–Uhlenbeck process [9]. Selected graphical performances on the validation sets are displayed in Figure 7.

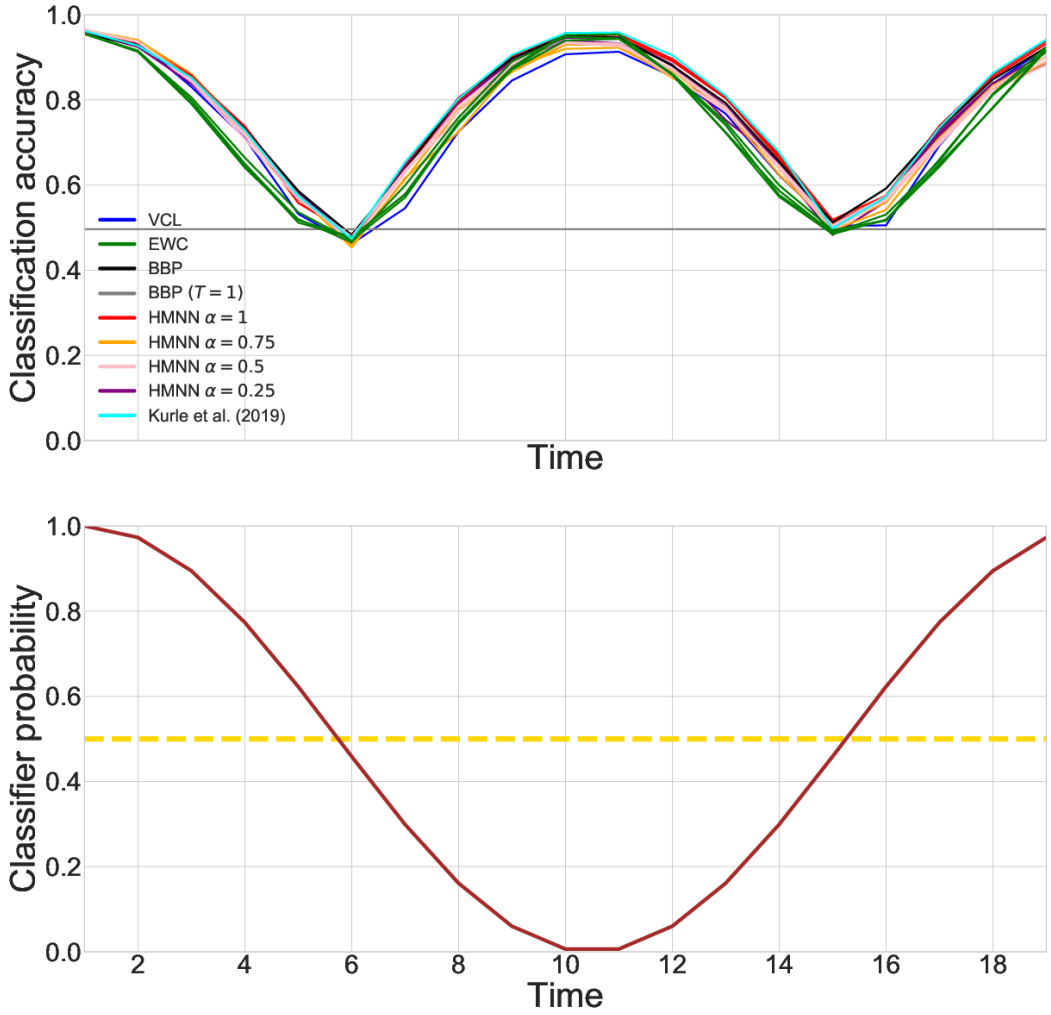

Figure 7. On the top, performances on a held-out validation set over time of the evolving classifiers obtained with different algorithms [9]. BBP refers to Bayes by Backprop trained sequentially. BBP (T = 1) refers to the training of Bayes by Backprop on the whole dataset. On the bottom, in brown, the evolution in time of the probability f_t of choosing the labeller $\mathcal{C}_1$, and in yellow, the value 0.5.

To test the method, we report the mean over time of the classification accuracy, which can be found in Table 3. For the HMNN, Kurle et al., Bayes by Backprop, EWC and VCL, we choose the parameters that perform the best on validation. We find that HMNN, the model by [9] and a sequential training of Bayes by Backprop perform the best. It is not surprising that continual learning methods fall behind. Indeed, EWC and VCL are built to preserve knowledge on the previous tasks, which might mix up $\mathcal{C}_1$ and $\mathcal{C}_2$ and confuse the network.

Table 3. Classification accuracy on a held-out test set for the evolving classifier (the bigger the better). BBP refers to Bayes by Backprop trained sequentially. BBP (T = 1) refers to the training of Bayes by Backprop on the whole dataset.

Model	Accuracy
BBP (T = 1)	0.503
VCL	0.744
EWC	0.760
BBP	0.780
Kurle et al. [9]	0.784
HMNN	0.786

4.5. One-Step-Ahead Prediction for Flag Waving

We conclude by testing HMNNs on predicting the next frame in a video. The dataset is a sequence of images extracted from a video of a waving flag [48,51]. The idea is to create an HMNN where the neural network at time t can predict the next frame, i.e., the NN maps frame t in frame $t+1$. To measure the performance, we use the metric suggested in [51], which is a standardized version of the RMSE on a chosen test trajectory:

$$\mathcal{M}(y_{1:T}, \hat{y}_{1:T}) := \sqrt{\frac{\sum_{t=1}^{T} \|y_t - \hat{y}_t\|_2^2}{\sum_{t=1}^{T} \|y_t\|_2^2}}, \tag{17}$$

where $y_{1:T}$ is the ground truth on frames $1, \dots, T$, and $\hat{y}_{1:T}$ are the predicted frames. Unless specified differently, $\hat{y}_{1:T}$ is a sequence of one-step-ahead predictions. To have a proper learning procedure, we need multiple frames per time step, otherwise, the neural network would just learn the current frame. To overcome this problem, we created a sliding window with 36 frames, meaning that at time step t, we train on predicting frames $t-35, \dots, t$ from frames $t-36, \dots, t-1$, with $t > 36$. The choice of the length for the sliding window is empirical, as we tried multiple lengths and stopped at the first one that did not overfit the data inside the window (the same procedure was repeated for all the baselines).

Per each time step, we used a simple architecture of three layers with 500, 20, 500 rectified linear units, the vectorized previous frame as input and the vectorized current frame as output (the dimension was reduced with PCA). We considered a fully Gaussian HMNN, with $\mu = m_{t-1}$ (kernel parameter) and all the other parameters selected through random grid search and validation. Figure 8 compares HMNN predictions with different baselines: Bayes by Backprop trained sequentially on the sliding windows (column named BBP), DropConnect trained sequentially on the sliding windows (column named DropConnect), LSTM trained with the same sliding window size (column named LSTM), and a trivial predictor that uses the previous frame as forecasting for the current frame (column name: frame $t-1$). We noticed that the LSTM is prone to overfitting on the sliding window and subsequently predicting frame t using the last frame seen without any uncertainty. Similar issues appear in sequential DropConnect. This unwanted behaviour is probably due to the absence of uncertainty quantification, resulting in overconfidence in the considered predictions. HMNN and sequential BBP are less certain about prediction and they create blurred regions where they expect the image to change. This phenomenon is particularly evident in the last row of Figure 8. Table 4 summarizes the performances using metric (17). Overall, the HMNN performs better than the baselines and it is directly followed by the sequential BBP.

Table 4. Metric $\mathcal{M}$ value on the test set (the smaller the better).

Model	$\mathcal{M}$
Trivial Predictor	0.2162
LSTM	0.2080
DropConnect	0.2063
BBP	0.1932
HMNN	0.1891

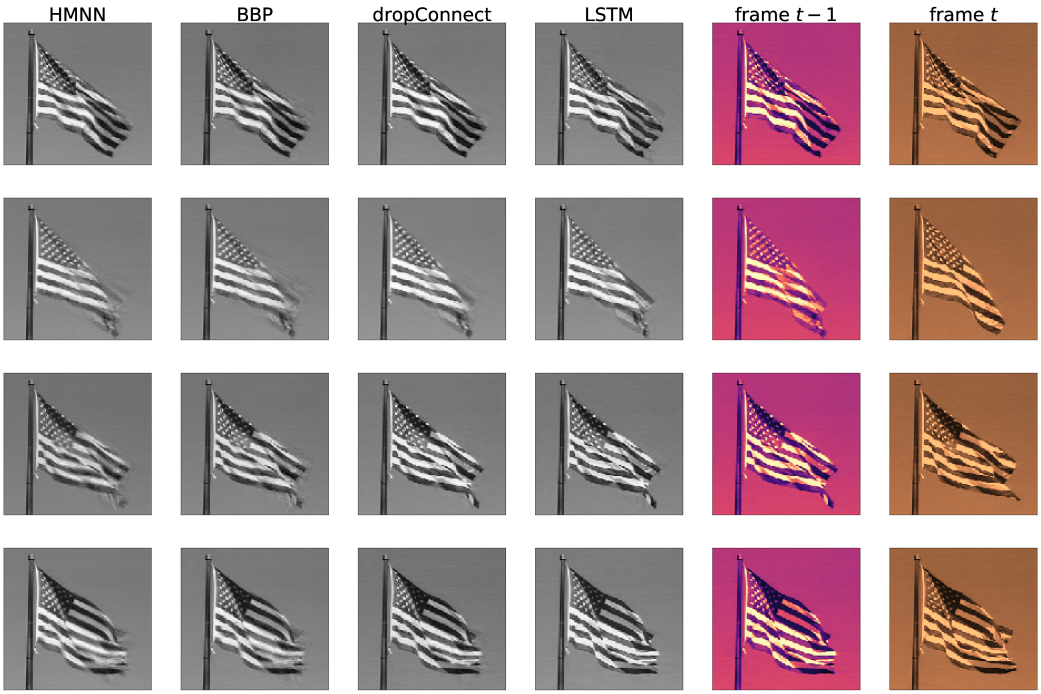

Figure 8. Columns show the prediction for different algorithms, with the last two being the last frame seen and the target frame. Rows display different time steps.

5. Discussion

We propose a hybrid model between the Bayesian neural network and FHMM called the Hidden Markov Neural Network (HMNN), where the posterior distribution over the weights is estimated sequentially through variational Bayes with an evolving prior distribution obtained by propagating forward the variational approximation from the previous time step through a stochastic transition kernel in the same vein as the assumed density filter [17,18]. We also propose a new variational approximation that induces a regularization technique called variational DropConnect, which resembles DropConnect [19] and variational DropOut [34], and we reformulate the reparameterization trick according to it. We test variational DropConnect on MNIST. We analyse the behaviour of HMNNs in a simple conceptual drift framework. We compare the HMNN with multiple baselines in a complex conceptual drift scenario and in time-series prediction. In all the experiments, our method compares favourably against the considered baselines, while allowing for uncertainty quantification. Compared to other techniques where deep learning and HMM are combined [28–30,43], HMNN uses a scale mixture of Gaussians as both variational approximations and transition kernel, allowing for a richer representation of both the dynamics of the weights and the filtering distribution.

The idea behind the HMNN is simple and it opens multiple research questions that can be answered in future works. Firstly, the quality of the approximation is not treated and we are also unsure about the rate of accumulation of the error over time. Theoretical studies

on variational approximations could be used to answer this matter [52,53]. Secondly, we do not propose any smoothing algorithm, which could improve the performance of the HMNN and could open avenues for an EM estimation of the hyperparameters. These could also be estimated via Recursive Maximum Likelihood Estimation [30] or by approximating the likelihood via sampling from the variational approximation. Thirdly, we have limited our studies to neural networks with simple architecture; it is surely possible to use similar techniques in recurrent neural networks and convolutional neural networks to tackle more complicated applications. Finally, the variational DropConnect can be applied in multiple variational Bayes scenarios to check if it leads to better performances.

Author Contributions: Conceptualization, L.R. and N.W.; Methodology, L.R. and N.W.; Software, L.R.; Validation, L.R.; Formal analysis, L.R. and N.W.; Investigation, L.R. and N.W.; Data curation, L.R.; Writing—original draft, L.R.; Writing—review & editing, L.R. and N.W.; Visualization, L.R.; Supervision, N.W.; Project administration, N.W.; Funding acquisition, N.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Data Availability Statement: A tutorial on the experiments and how to use the code is available at the GitHub repository: https://github.com/LorenzoRimella/HiddenMarkovNeuralNetwork (accessed on 20 December 2024).

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

HMM	Hidden Markov Model
HMNN	Hidden Markov Neural Network
NN	Neural Network
ELBO	Evidence Lower Bound

Appendix A. Formulae Derivation

The full derivation of the **KL**-divergence between the variational approximation and the evolving prior for a general time step t of an HMNN is

$$\begin{aligned}\mathbf{KL}(q_\theta||\mathsf{C}_t\mathsf{P}\pi_{t-1}) &:= \mathbb{E}_{q_\theta(w)}[\log(q_\theta(w)) - \log(\mathsf{C}_t\mathsf{P}\pi_{t-1}(w))] \\ &= constant + \mathbb{E}_{q_\theta(w)}[\log(q_\theta(w)) - \log(g(w,\mathcal{D}_t)) - \log(\mathsf{P}\pi_{t-1}(w))] \\ &= constant + \mathbf{KL}(q_\theta||\mathsf{P}\pi_{t-1}) - \mathbb{E}_{q_\theta(w)}[\log(g(w,\mathcal{D}_t))],\end{aligned} \tag{A1}$$

where

$$\begin{aligned}\log \mathsf{C}_t\mathsf{P}\pi_{t-1}(w) &= \log\left(\frac{g(w,\mathcal{D}_t)\mathsf{P}\pi_{t-1}(w)}{\int g(w,\mathcal{D}_t)\mathsf{P}\pi_{t-1}(w)dw}\right) \\ &= \log(g(w,\mathcal{D}_t)) + \log(\mathsf{P}\pi_{t-1}(w)) - \log\left(\int g(w,\mathcal{D}_t)\mathsf{P}\pi_{t-1}(w)dw\right).\end{aligned} \tag{A2}$$

The closed-form solution of $\mathsf{P}\tilde{\pi}_{t-1}$ for a fully Gaussian HMNN is derived from the following integral:

$$
\begin{aligned}
(\mathsf{P}\tilde{\pi}_{t-1})^v(w^v) &= \int p(\tilde{w}^v, w^v)(\tilde{\pi}_{t-1})^v(\tilde{w}^v)d\tilde{w}^v \\
&= \gamma^v\phi \int \mathcal{N}\Big(w^v|\mu^v + \alpha(\tilde{w}^v - \mu^v), \sigma^2\Big)\mathcal{N}\Big(\tilde{w}^v|m^v_{t-1}, (s^v_{t-1})^2\Big)d\tilde{w}^v \\
&\quad + (1-\gamma^v)\phi \int \mathcal{N}\Big(w^v|\mu^v + \alpha(\tilde{w}^v - \mu^v), \sigma^2\Big)\mathcal{N}\Big(\tilde{w}^v|0, (s^v_{t-1})^2\Big)d\tilde{w}^v \\
&\quad + \gamma^v(1-\phi) \int \mathcal{N}\Big(w^v|\mu^v + \alpha(\tilde{w}^v - \mu^v), \sigma^2/c^2\Big)\mathcal{N}\Big(\tilde{w}^v|m^v_{t-1}, (s^v_{t-1})^2\Big)d\tilde{w}^v \\
&\quad + (1-\gamma^v)(1-\phi) \int \mathcal{N}\Big(w^v|\mu^v + \alpha(\tilde{w}^v - \mu^v), \sigma^2/c^2\Big)\mathcal{N}\Big(\tilde{w}^v|0, (s^v_{t-1})^2\Big)d\tilde{w}^v,
\end{aligned}
\tag{A3}
$$

where the formulation in the main paper can be achieved by applying the following lemma to each element of the sum in (A3).

Lemma A1. *Consider a Gaussian random variable* $W \sim \mathcal{N}(\cdot|\mu_1, \sigma_1^2)$ *and let*

$$\tilde{W} = \mu_2 - \alpha(\mu_2 - W) + \sigma_2\xi, \tag{A4}$$

with $\xi \sim \mathcal{N}(\cdot|0,1)$. *Then, the distribution of* $\tilde{W}$ *is once again Gaussian:*

$$\tilde{W} \sim \mathcal{N}(\cdot|\mu_2 - \alpha(\mu_2 - \mu_1), \sigma_2^2 + \alpha^2\sigma_1^2). \tag{A5}$$

Proof. Note that the distribution $p(\tilde{w}|w)$ of $\tilde{W}|W$ is a Gaussian distribution, so $p(\tilde{w}) = \int p(\tilde{w}|w)p(w)dw$ is an integral of the same form of the ones in (A3). The distribution of $\tilde{W}$ can be directly computed by noting that $\tilde{W}$ is a linear combination of two independent Gaussians and a scalar, and consequently, it is Gaussian itself. The lemma is proved by computing the straightforward mean and variance from Equation (A4). □

Appendix B. Experiments

This section presents a subsection for each experiment:

- Appendix B.1 treats the experiment on variational DropConnect;
- Appendix B.3 describes the application to the evolving classifier;
- Appendix B.4 considers the video texture of a waving flag.

Unless we specify differently, we trained using the vanilla gradient descent with learning rate γ.

Appendix B.1. Variational DropConnect

We trained on the MNIST dataset consisting of 60,000 images of handwritten digits of the size 28 by 28. The images were preprocessed by dividing each pixel by 126. We used 50,000 images for training and 10,000 for validation. The test set is composed of 10,000 images. Both training and test sets can be downloaded from http://yann.lecun.com/exdb/mnist/ accessed on 1 December 2024.

As for [7], we focused on an ordinary feed-forward neural network without any convolutional layers. We considered a small architecture with the vectorized image as input, 2 hidden layers with 400 rectified linear units [49,50] and a softmax layer on 10 classes as output. We considered a cross-entropy loss and a fully Gaussian HMNN with a single time step. The variational Dropconnect technique was applied only to the internal linear layers of the network, i.e., we excluded the initial layer and the final one as in [44]. Observe that a single time step HMNN with $\gamma^v = 1, \alpha = 0$ and $\mu = \mathbf{0}$ (vector of zeros) is equivalent to Bayes by Backprop. For this reason, we could simply use our implementation of the HMNN to include Bayes by Backprop. We set $T = 1, \alpha = 0, \mu = \mathbf{0}$ and we consider

$\gamma^v \in \{0.25, 0.5, 0.75, 1\}$, $\phi \in \{0.25, 0.5, 0.75\}$, $-\log(\sigma) \in \{0, 1, 2\}$, $-\log(c) \in \{6, 7, 8\}$, learning rate $l \in \{10^{-5}, 10^{-4}, 10^{-3}\}$.

We generated more than 50 random combinations of the parameters $(p, \phi, \sigma, c, \gamma)$ for each combination; we also randomly set the number of Monte Carlo simulations to $N \in \{1, 2, 5\}$. We trained each combination for 600 epochs and we considered a minibatch size of 128. We then chose the best three models per each possible value of p and we report the performance on the validation set in the main paper.

Appendix B.2. Two Moons Dataset

We consider the following set of hyperparameters: $\alpha = 0.5$, $\gamma^v = 0.75$, $\phi = 0.5$, $-\log(\sigma) = 2$, $\log(c) = 5$, $l = 10^{-3}$.

Appendix B.3. Evolving Classifier on MNIST

The main feature of HMNN is the time dimension; hence, we need an example where the data evolve over time. We decided to build this example from the MNIST dataset by simply changing the way in which we assign the labels:

1. We preprocessed the data by dividing each pixel by 126. We then defined two labellers: $\mathcal{C}_1$, naming each digit with its label in MNIST; $\mathcal{C}_2$, labelling each digit with its MNIST's label shifted by one unit, i.e., 0 is classified as 1, 1 is classified as 2, ..., 9 is classified as 0.
2. We considered 19 time steps where each time step t is associated with a probability $f_t \in [0, 1]$ and a portion of the MNIST's dataset $\mathcal{D}_t$. The probability of choosing $\mathcal{C}_1$ evolves as follows:

$$f_t = \frac{1}{2}\sin\left(\frac{\pi}{8}\left(\frac{4t}{5} + \frac{16}{5}\right)\right) + \frac{1}{2}, \quad t = 1, \dots 19,$$

   where π is the actual Pi, i.e., $\pi \approx 3.14$.
3. At each time step t, we randomly labelled each digit in $\mathcal{D}_t$ with either $\mathcal{C}_1$ or $\mathcal{C}_2$ according to the probabilities $f_t, 1 - f_t$.

The above procedure was used for all training, validation and test sets. The sample size of each time step is 10,000 for training and 5000 for validation and testing (we resampled from the training set of MNIST to reach the desired sample size). To validate and test the models, we considered the mean classification accuracy over time:

$$\mathcal{A}(\mathcal{D}_{1:T}, \hat{\mathcal{D}}_{1:T}) := \frac{1}{T}\sum_{t=1}^{T}\frac{1}{|\mathcal{D}_t|}\sum_{x,y\in\mathcal{D}_t}\mathbb{I}_y(\hat{y}(x)), \tag{A6}$$

where $\mathcal{D}_t$ is the generated dataset of images x and labels y, $\hat{\mathcal{D}}_t$ is the collection of images x and predictions $\hat{y}(x)$ on the images x using the considered model, and $|\mathcal{D}_t|$ is the number of elements in $\mathcal{D}_t$, i.e., the total number of labels or images.

We considered a fully Gaussian HMNN with $\mu = m_{t-1}$ to encourage a strong memory on the previous posterior. The evolving in-time NN was composed of the vectorized image as input, 2 hidden layers with 100 rectified linear units and a softmax layer on 10 classes as output, and we consideed a cross-entropy loss function. The variational DropConnect technique was again applied to the internal linear layers of the network only. The parameters $\alpha \in \{0.25, 0.5, 0.75, 1\}$ and $\gamma^v \in \{0.25, 0.5, 0.75, 1\}$ were selected through the validation set (using metric (A6)) while the other parameters were as follows: $\phi = 0.5, -\log(\sigma) = 2, -\log(c) = 4, l = 10^{-3}, N = 1$. We also tried other values for ϕ, σ, c, l, N but we did not experience significant changes in terms of performance. We tried all possible combinations of α, l and we trained on the generated training set for 19 time

steps and 100 epochs per each $\mathcal{D}_t$. We remark that a single $\mathcal{D}_t$ is a collection of images and labels and it can be seen as a whole dataset itself.

We compared our method with four algorithms. The architecture of the NN is the same as that of the HMNN:

- Sequential Bayes by Backprop. At each time step t, we trained for 100 epochs on $\mathcal{D}_t$. The parameters are $\phi = 0.5, -\log(\sigma) = 2, -\log(c) = 4, l = 10^{-3}, N = 1$. The Bayesian NN at time t was initialized with the previous estimates m_{t-1}, s_{t-1}.
- Bayes by Backprop on the whole dataset. We trained for 100 epochs on the whole dataset (no time dimension; the sample size was 190,000) a Bayesian NN with Bayes by Backprop. The parameters are $\phi = 0.5, -\log(\sigma) = 2, -\log(c) = 4, l = 10^{-3}, N = 1$.
- Elastic Weight Consolidation. At each time step t, we trained for 100 epochs on $\mathcal{D}_t$. The tuning parameter was chosen from the grid $\{10, 100, 1000, 10{,}000\}$ through the validation set and the metric (A6). We found that ADAM worked better, and hence, we trained with it. Recall that this method is not Bayesian, but it is a well-known baseline for continual learning.
- Variational Continual Learning. At each time step t, we trained for 100 epochs on $\mathcal{D}_t$. We chose the learning rate from the grid $\{10^{-3}, 10^{-4}, 10^{-5}\}$ through validation and the metric (A6). The training was pursued without the use of a coreset because we were not comparing rehearsal methods.

Sequential classification accuracies on the validation set and test performances using (A6) are shown in the main paper.

Appendix B.4. One-Step-Ahead Prediction for Flag Waving

A video has an intrinsic dynamic given by the sequence of frames, which makes the HMNN suitable for a one-step-ahead prediction video's frame. Here, we consider the video of a waving flag.

The preprocessing phase is similar to MNIST: the video was converted into a sequence of frames in greyscale that were additionally divided by 126 and put in a vector form. We then reduced the dimension with PCA (130 principal components). We used 300 frames in total, validated on the frames from 100 to 150 and tested on the last 150 frames. Note that we had a single video available; hence, we needed to perform the validation and test online, meaning that validation and test sets were also part of the training, but they were not seen in advance. Precisely, during validation, we trained on the full path from 1 to 150, made predictions on the frames from 100 to 150 using the HMNN from time 99 to 149, and then computed the associated metric on the considered path. Once the validation score was available, we performed model selection and continued the training on the next frames to obtain the performance on the test set. As explained in the main paper, we trained sequentially on a sliding window that included 36 frames.

Consider a fully Gaussian HMNN with a simple architecture of 3 layers with 500, 20, 500 rectified linear units, the vectorized previous frame as input, the vectorized current frame as output, and an MSE loss. We applied variational DropConnect to all the linear layers. The parameters $\alpha \in \{0.25, 0.5, 0.75\}$, $\gamma^v \in \{0.3, 0.8, 1\}$, $\phi = \{0.25, 0.5, 0.75\}$ were chosen according to the validation set. For the other parameters, we found that setting $\log(\sigma) = 2, \log(c) = 8, l = 10^{-4}, N = 1$ performed the best. We trained for 150 epochs on each sliding window.

We compared it to four models. The architectures for sequential Bayes by Backprop and sequential DropConnect are the same as HMNN. For implementation purposes, we considered an architecture of 3 layers with 500, 500, 500 rectified linear units for the LSTM:

- Sequential Bayes by Backprop. At each time step t, we trained on the current sliding window for 150 epochs. The parameter $\phi = \{0.25, 0.5, 0.75\}$ was chosen with grid

search and $\log(\sigma) = 2, \log(c) = 8, l = 10^{-4}, N = 1$. We found that other choices of σ, c, l, N did not improve the performance. The Bayesian neural network at time t was initialized with the estimates at time $t-1$.

- Sequential DropConnect. At each time step t, we trained on the current sliding window for 150 epochs. The learning rate $l = \{10^{-3}, 10^{-4}, 10^{-5}\}$ was chosen with grid search using the validation score. The neural network at time t was initialized with the estimates at time $t-1$. This is not a Bayesian method.
- LSTM. We chose a window size of 36 and the learning rate $l = \{10^{-3}, 10^{-4}, 10^{-5}\}$ with grid search using the validation score. This is not a Bayesian method.
- Trivial predictor. We predicted frame t with frame $t-1$. We decided to include this trivial baseline because it is an indicator of overfitting on the current window.

Given that we extracted 30 frames per second, successive frames looked almost equal; this makes the comparison with the trivial predictor unfair. Testing on a subpath makes the metric $\mathcal{M}$ more sensitive to big changes and it gives us a clearer measure of the learned patterns. Hence, we tested on the subpath $150, 152, 154, \ldots 300$ (we skipped all the odd frames from 150 to 300). As already explained, this is just a subset of the full test set and no algorithm is discarding any frames during training.

References

1. Rabiner, L.R.; Juang, B.H. An introduction to hidden Markov models. *IEEE ASSP Mag.* **1986**, *3*, 4–16. [CrossRef]
2. Krogh, A.; Larsson, B.; Von Heijne, G.; Sonnhammer, E.L. Predicting transmembrane protein topology with a hidden Markov model: application to complete genomes. *J. Mol. Biol.* **2001**, *305*, 567–580. [CrossRef]
3. Ghahramani, Z.; Jordan, M.I. Factorial Hidden Markov Models. *Mach. Learn.* **1997**, *29*, 245–273. [CrossRef]
4. Blei, D.M.; Kucukelbir, A.; McAuliffe, J.D. Variational inference: A review for statisticians. *J. Am. Stat. Assoc.* **2017**, *112*, 859–877. [CrossRef]
5. Graves, A. Practical variational inference for neural networks. In Proceedings of the Advances in Neural Information Processing Systems, Granada, Spain, 12–15 December 2011; pp. 2348–2356.
6. Kingma, D.P.; Welling, M. Auto-encoding variational bayes. *arXiv* **2013**, arXiv:1312.6114.
7. Blundell, C.; Cornebise, J.; Kavukcuoglu, K.; Wierstra, D. Weight uncertainty in neural network. In Proceedings of the International Conference on Machine Learning, PMLR, Lile, France, 6–11 July 2015; pp. 1613–1622.
8. Wang, L.; Zhang, X.; Su, H.; Zhu, J. A comprehensive survey of continual learning: Theory, method and application. *IEEE Trans. Pattern Anal. Mach. Intell.* **2024**, *46*, 5362–5383. [CrossRef]
9. Kurle, R.; Cseke, B.; Klushyn, A.; van der Smagt, P.; Günnemann, S. Continual Learning with Bayesian Neural Networks for Non-Stationary Data. In Proceedings of the International Conference on Learning Representations, New Orleans, LA, USA, 6–9 May 2019.
10. Kirkpatrick, J.; Pascanu, R.; Rabinowitz, N.; Veness, J.; Desjardins, G.; Rusu, A.A.; Milan, K.; Quan, J.; Ramalho, T.; Grabska-Barwinska, A.; et al. Overcoming catastrophic forgetting in neural networks. *Proc. Natl. Acad. Sci. USA* **2017**, *114*, 3521–3526. [CrossRef]
11. Nguyen, C.V.; Li, Y.; Bui, T.D.; Turner, R.E. Variational continual learning. *arXiv* **2017**, arXiv:1710.10628.
12. Ritter, H.; Botev, A.; Barber, D. Online structured laplace approximations for overcoming catastrophic forgetting. In Proceedings of the Advances in Neural Information Processing Systems, Montreal, QC, Canada, 3–8 December 2018; pp. 3738–3748.
13. Chopin, N.; Papaspiliopoulos, O. *An Introduction to Sequential Monte Carlo*; Springer: Cham, Switzerland, 2020.
14. Rebeschini, P.; Van Handel, R. Can local particle filters beat the curse of dimensionality? *Ann. Appl. Probab.* **2015**, *25*, 2809–2866. [CrossRef]
15. Rimella, L.; Whiteley, N. Exploiting locality in high-dimensional Factorial hidden Markov models. *J. Mach. Learn. Res.* **2022**, *23*, 1–34.
16. Duffield, S.; Power, S.; Rimella, L. A state-space perspective on modelling and inference for online skill rating. *J. R. Stat. Soc. Ser. C Appl. Stat.* **2024**, *73*, 1262–1282. [CrossRef]
17. Sorenson, H.W.; Stubberud, A.R. Non-linear filtering by approximation of the a posteriori density. *Int. J. Control* **1968**, *8*, 33–51. [CrossRef]
18. Minka, T.P. A Family of Algorithms for Approximate Bayesian Inference. Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2001.

19. Wan, L.; Zeiler, M.; Zhang, S.; Le Cun, Y.; Fergus, R. Regularization of neural networks using dropconnect. In Proceedings of the International Conference on Machine Learning, Atlanta, GA, USA, 16–21 June 2013; pp. 1058–1066.
20. Franzini, M.; Lee, K.F.; Waibel, A. Connectionist Viterbi training: A new hybrid method for continuous speech recognition. In Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, Albuquerque, NM, USA, 3–6 April 1990; pp. 425–428.
21. Bengio, Y.; Cardin, R.; De Mori, R.; Normandin, Y. A hybrid coder for hidden Markov models using a recurrent neural networks. In Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, Albuquerque, NM, USA, 3–6 April 1990; pp. 537–540.
22. Bengio, Y.; De Mori, R.; Flammia, G.; Kompe, R. Global optimization of a neural network-hidden Markov model hybrid. In Proceedings of the IJCNN-91-Seattle International Joint Conference on Neural Networks, Seattle, WA, USA, 8–12 July 1991; Volume 2, pp. 789–794.
23. Krogh, A.; Riis, S.K. Hidden neural networks. *Neural Comput.* **1999**, *11*, 541–563. [CrossRef]
24. Johnson, M.J.; Duvenaud, D.; Wiltschko, A.B.; Datta, S.R.; Adams, R.P. Composing graphical models with neural networks for structured representations and fast inference. *arXiv* **2016**, arXiv:1603.06277.
25. Karl, M.; Soelch, M.; Bayer, J.; Van der Smagt, P. Deep variational bayes filters: Unsupervised learning of state space models from raw data. *arXiv* **2016**, arXiv:1605.06432.
26. Krishnan, R.; Shalit, U.; Sontag, D. Structured inference networks for nonlinear state space models. In Proceedings of the AAAI Conference on Artificial Intelligence, San Francisco, CA, USA, 4–9 February 2017; Volume 31.
27. Aitchison, L.; Pouget, A.; Latham, P.E. Probabilistic synapses. *arXiv* **2014**, arXiv:1410.1029.
28. Chang, P.G.; Durán-Martín, G.; Shestopaloff, A.Y.; Jones, M.; Murphy, K. Low-rank extended Kalman filtering for online learning of neural networks from streaming data. *arXiv* **2023**, arXiv:2305.19535.
29. Jones, M.; Scott, T.R.; Ren, M.; ElSayed, G.; Hermann, K.; Mayo, D.; Mozer, M. Learning in Temporally Structured Environments. In Proceedings of the International Conference on Learning Representations, Kigali, Rwanda, 1–5 May 2023.
30. Campbell, A.; Shi, Y.; Rainforth, T.; Doucet, A. Online variational filtering and parameter learning. *Adv. Neural Inf. Process. Syst.* **2021**, *34*, 18633–18645.
31. Mobiny, A.; Yuan, P.; Moulik, S.K.; Garg, N.; Wu, C.C.; Van Nguyen, H. Dropconnect is effective in modeling uncertainty of bayesian deep networks. *Sci. Rep.* **2021**, *11*, 5458. [CrossRef]
32. Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* **2014**, *15*, 1929–1958.
33. Salehin, I.; Kang, D.K. A review on dropout regularization approaches for deep neural networks within the scholarly domain. *Electronics* **2023**, *12*, 3106. [CrossRef]
34. Kingma, D.P.; Salimans, T.; Welling, M. Variational dropout and the local reparameterization trick. In Proceedings of the Advances in Neural Information Processing Systems, Montreal, QC, Canada, 7–12 December 2015; pp. 2575–2583.
35. Gal, Y.; Ghahramani, Z. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In Proceedings of the International Conference on Machine Learning, New York, NY, USA, 19–24 June 2016; pp. 1050–1059.
36. Puskorius, G.V.; Feldkamp, L.A. Neurocontrol of nonlinear dynamical systems with Kalman filter trained recurrent networks. *IEEE Trans. Neural Netw.* **1994**, *5*, 279–297. [CrossRef]
37. Puskorius, G.V.; Feldkamp, L.A. Parameter-based Kalman filter training: Theory and implementation. In *Kalman Filtering and Neural Networks*; Wiley: Hoboken, NJ, USA, 2001; pp. 23–67. [CrossRef]
38. Feldkamp, L.A.; Prokhorov, D.V.; Feldkamp, T.M. Simple and conditioned adaptive behavior from Kalman filter trained recurrent networks. *Neural Netw.* **2003**, *16*, 683–689. [CrossRef]
39. Ollivier, Y. Online natural gradient as a Kalman filter. *Electron. J. Stat.* **2018**, *12*, 2930–2961. [CrossRef]
40. Aitchison, L. Bayesian filtering unifies adaptive and non-adaptive neural network optimization methods. *arXiv* **2018**, arXiv:1807.07540.
41. Khan, M.E.; Swaroop, S. Knowledge-Adaptation Priors. In *Proceedings of the Advances in Neural Information Processing Systems*; Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., Vaughan, J.W., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2021; Volume 34, pp. 19757–19770.
42. Liu, L.; Jiang, X.; Zheng, F.; Chen, H.; Qi, G.J.; Huang, H.; Shao, L. A bayesian federated learning framework with online laplace approximation. *IEEE Trans. Pattern Anal. Mach. Intell.* **2023**, *46*, 1–16. [CrossRef]
43. Sliwa, J.; Schneider, F.; Bosch, N.; Kristiadi, A.; Hennig, P. Efficient Weight-Space Laplace-Gaussian Filtering and Smoothing for Sequential Deep Learning. *arXiv* **2024**, arXiv:2410.06800.
44. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
45. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.

46. Chan, A.B.; Vasconcelos, N. Classifying video with kernel dynamic textures. In Proceedings of the 2007 IEEE Conference on Computer Vision and Pattern Recognition, Minneapolis, MN, USA, 17–22 June 2007; pp. 1–6.
47. Boots, B.; Gordon, G.J.; Siddiqi, S.M. A constraint generation approach to learning stable linear dynamical systems. In Proceedings of the Advances in Neural Information Processing Systems, Vancouver, BC, Canada, 8–11 December 2008; pp. 1329–1336.
48. Basharat, A.; Shah, M. Time series prediction by chaotic modeling of nonlinear dynamical systems. In Proceedings of the 2009 IEEE 12th International Conference on Computer Vision, Kyoto, Japan, 29 September–2 October 2009; pp. 1941–1948.
49. Nair, V.; Hinton, G.E. Rectified linear units improve restricted boltzmann machines. In Proceedings of the 27th International Conference on Machine Learning (ICML-10), Haifa, Israel, 21–24 June 2010; pp. 807–814.
50. Glorot, X.; Bordes, A.; Bengio, Y. Deep sparse rectifier neural networks. In Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, Fort Lauderdale, FL, USA, 11–13 April 2011; pp. 315–323.
51. Venkatraman, A.; Hebert, M.; Bagnell, J.A. Improving multi-step prediction of learned time series models. In Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, Austin, TX, USA, 25–30 January 2015.
52. Yang, Y.; Pati, D.; Bhattacharya, A. α-Variational Inference with Statistical Guarantees. *arXiv* **2017**, arXiv:1710.03266. [CrossRef]
53. Chérief-Abdellatif, B.E. Convergence Rates of Variational Inference in Sparse Deep Learning. *arXiv* **2019**, arXiv:1908.04847.