Memory-Based Meta-Learning on Non-Stationary Distributions

Tim Genewein [*1] **Grégoire Delétang** [*1] **Anian Ruoss** [*1] **Li Kevin Wenliang** [1] **Elliot Catt** [1] **Vincent Dutordoir** [1 2]
Jordi Grau-Moya [1] **Laurent Orseau** [1] **Marcus Hutter** [1] **Joel Veness** [1]

Abstract

Memory-based meta-learning is a technique for approximating Bayes-optimal predictors. Under fairly general conditions, minimizing sequential prediction error, measured by the log loss, leads to implicit meta-learning. The goal of this work is to investigate how far this interpretation can be realized by current sequence prediction models and training regimes. The focus is on piecewise stationary sources with unobserved switching-points, which arguably capture an important characteristic of natural language and action-observation sequences in partially observable environments. We show that various types of memory-based neural models, including Transformers, LSTMs, and RNNs can learn to accurately approximate known Bayes-optimal algorithms and behave as if performing Bayesian inference over the latent switching-points and the latent parameters governing the data distribution within each segment.

1. Introduction

Memory-based meta-learning (MBML) has recently risen to prominence due to breakthroughs in sequence modeling and the proliferation of data-rich multi-task domains. Previous work (Ortega et al., 2019; Mikulik et al., 2020) showed how, in principle, MBML can lead to Bayes-optimal predictors by learning a fixed-parametric model that performs amortized inference via its activations. This interpretation of MBML can provide theoretical understanding for counter-intuitive phenomena such as in-context learning that emerge in large language models with frozen weights (Xie et al., 2022).

In this work, we investigate the potential of MBML to learn parametric models that implicitly perform Bayesian inference with respect to more elaborate distributions than the ones investigated in Mikulik et al. (2020). We focus on *piecewise stationary* Bernoulli distributions, which produce sequences that consist of Bernoulli *segments* (see Figure 1). The predictor only observes a stream of samples (0s and 1s), with abrupt changes to local statistics at the unobserved switching-points between segments. The focus on piecewise stationary sources is inspired by natural language, where documents often switch topic without explicit indication (Xie et al., 2022), and observation-action streams in environments with discrete latent variables, e.g., multi-task RL without task-indicators. In both domains, neural models that minimize sequential prediction error demonstrate hallmarks of sequential Bayesian prediction: strong context sensitivity or "in-context learning" (Reed et al., 2022), and rapid adaptation or "few-shot learning" (Brown et al., 2020).

To solve the sequential prediction problem, Bayes-optimal (BO) predictors simultaneously consider a number of hypotheses over switching-points and use prior knowledge over switching-points and segment-statistics. Tractable exact BO predictors require non-trivial algorithmic derivations, and are only known for certain switching-point distributions. The main question of this paper is whether neural predictors with memory, trained by minimizing sequential prediction error (log loss), can learn to mimic Bayes-optimal solutions and match their prediction performance.

Our contributions are:

- Review of the theoretical connection between minimizing sequential prediction error, meta-learning, and its implied Bayesian objective (Section 3).
- Theoretical argument for the necessity of memory to minimize the former (Bayesian) objective (Section 4).
- Empirical demonstration that meta-learned neural predictors can match prediction performance of two general non-parametric Bayesian predictors (Section 7).
- Comparison of off-distribution generalization of learned solutions and Bayesian algorithms (Section 7).
- Source code available at: `https://github.com/deepmind/nonstationary_mbml`.

[*]Equal contribution [1]DeepMind [2]University of Cambridge. Correspondence to: Tim Genewein <timgen@deepmind.com>, Grégoire Delétang <gdelt@deepmind.com>, Anian Ruoss <anianr@deepmind.com>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

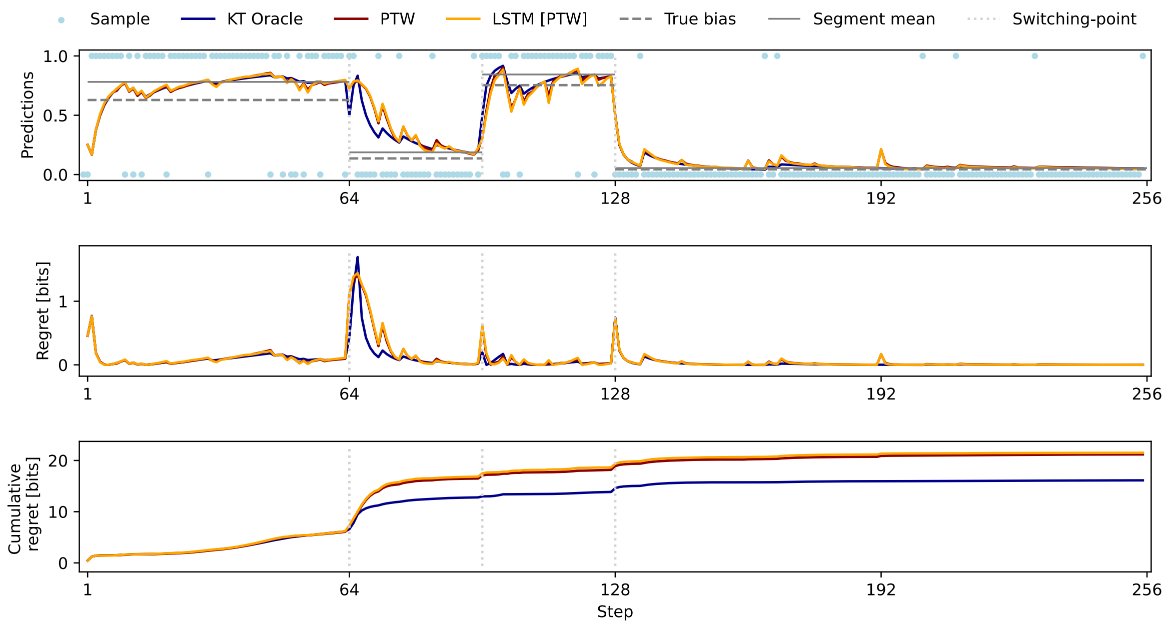


Figure 1. A single sequence from a piecewise Bernoulli source with three switching-points drawn from the PTW prior (see Section 6). Top: The predictors observe streams of binary samples x_t and, at each step, predict the probability of the next observation. The solid lines show predictions $p(x_t|x_{<t})$ by the Bayes-optimal PTW, the KT Oracle that observes switching-points, and the trained LSTM (trained on data from PTW prior, indicated in the square bracket). Both the LSTM and PTW rapidly adapt after switching-points, enabled via the inductive bias of the PTW prior and acquired by the LSTM via meta-learning on data following the PTW prior. Middle: Per-time-step regret (see Section 6) measures the prediction error by quantifying the excess log-loss compared to a predictor that always knows the ground-truth bias. Bottom: Cumulative regret; the value at the final time-step is the basis for our main performance metric (see Equation (4)).

2. Background

We begin with some terminology for sequential, probabilistic data generating sources. An alphabet is a finite, non-empty set of symbols, which we denote by $\mathcal{X}$. A string $x_1x_2\ldots x_n \in \mathcal{X}^n$ of length n is denoted by $x_{1:n}$. The prefix $x_{1:j}$ of $x_{1:n}$, $j \leq n$, is denoted by $x_{\leq j}$ or $x_{<j+1}$. The empty string is denoted by ϵ. Our notation also generalizes to out of bounds indices; that is, given a string $x_{1:n}$ and an integer $m > n$, we define $x_{1:m} := x_{1:n}$ and $x_{n:m} := \epsilon$. The concatenation of two strings s and r is denoted by sr.

Probabilistic Data Generating Sources A probabilistic data generating source ρ is defined by a sequence of probability mass functions $\rho_n : \mathcal{X}^n \to [0,1]$, for all $n \in \mathbb{N}$, satisfying the compatibility constraint that $\rho_n(x_{1:n}) = \sum_{y\in\mathcal{X}} \rho_{n+1}(x_{1:n}y)$ for all $x_{1:n} \in \mathcal{X}^n$, with base case $\rho_0(\epsilon) = 1$. From here onward, whenever the meaning is clear from the argument to ρ, the subscripts on ρ will be dropped. Under this definition, the conditional probability of a symbol x_n given previous data $x_{<n}$ is defined as $\rho(x_n|x_{<n}) := \rho(x_{1:n})/\rho(x_{<n})$ provided $\rho(x_{<n}) > 0$, with the familiar chain rules $\rho(x_{1:n}) = \prod_{i=1}^{n} \rho(x_i|x_{<i})$ and $\rho(x_{i:j}|x_{<i}) = \prod_{k=i}^{j} \rho(x_k|x_{<k})$ now following.

Temporal Partitions A sub-sequence is described via a segment, which is a tuple of time-indices $(a,b) \in \mathbb{N}\times\mathbb{N}$ with $a \leq b$. A segment (a,b) is said to overlap with another segment (c,d) if there exists an $i \in \mathbb{N}$ such that $a \leq i \leq b$ and $c \leq i \leq d$. Let $S = \{1,2,\ldots n\}$ denote a set of time-indices for some $n \in \mathbb{N}$. A temporal partition $\mathcal{P}$ of S is a set of non-overlapping segments such that each $i \in S$ is covered by exactly one segment $(a,b) \in \mathcal{P}$ with $a \leq i \leq b$. We also use the overloaded notation $\mathcal{P}(a,b) := \{(c,d) \in \mathcal{P} : a \leq c \leq d \leq b\}$. Finally, $\mathcal{T}_n$ will be used to denote the set of all possible temporal partitions of $\{1,2,\ldots,n\}$.

Piecewise Stationary Sources We now define a piecewise stationary data generating source μ in terms of a partition $\mathcal{P} = \{(a_1,b_1),(a_2,b_2),\ldots\}$ and a set of probabilistic data generating sources $\{\mu^1,\mu^2,\ldots\}$, such that for all $n \in \mathbb{N}$, for all $x_{1:n} \in \mathcal{X}^n$,

$$\mu(x_{1:n}) := \prod_{(a,b)\in\mathcal{P}_n} \mu^{f(a)}(x_{a:b}), \tag{1}$$

where $\mathcal{P}_n := \{(a_i,b_i) \in \mathcal{P} : a_i \leq n\}$ and $f(i)$ returns the index of the time segment containing i; that is, it gives a value $k \in \mathbb{N}$ such that both $(a_k,b_k) \in \mathcal{P}$ and $a_k \leq i \leq b_k$. In other words: a piecewise stationary data generating source consists of a number of non-overlapping segments (covering the entire range without gaps), with one stationary data generating distribution per segment. An example-draw from such a source is shown in Figure 1, where the distribution per segment is a Bernoulli process.

3. Memory-Based Meta-Learning

Given a parametric, memory-dependent probabilistic model $\rho_\theta(x_{1:n})$, a standard MBML setup works by repeating the following steps:

1. Sample a task τ from a task distribution ψ;
2. Generate data $x_{1:n} \sim \tau$;
3. Perform one or more steps of optimization of the model parameters θ using the loss $-\log \rho_\theta(x_{1:n}) = -\sum_{i=1}^{n} \rho_\theta(x_i|x_{<i})$.

In our piecewise stationary Bernoulli setup, a task corresponds to prediction on a particular binary sequence (meaning τ is an instance of switching-points and Bernoulli biases for each segment), and the distribution over tasks is exactly the piecewise stationary distribution. In the case where the task distribution is defined over a finite number of tasks, the marginal probability of the MBML data generating source is simply:

$$\xi(x_{1:n}) = \sum_{\tau} \psi(\tau)\, \tau(x_{1:n}). \tag{2}$$

In other words: in meta-learning, the training data is *implicitly* generated by a Bayesian mixture whose properties are determined from the particular details of the meta-training setup. Note that this marginal form of a Bayesian mixture still captures the usual notion of posterior updating implicitly; see Appendix F for more background.

Optimality of Bayesian Predictor for MBML Consider the expected excess log loss of using any sequential predictor ρ on data $x_{1:n} \sim \xi$. Notice that for all $n \in \mathbb{N}$, we have that

$$\mathbb{E}_\xi \left[-\log \rho(x_{1:n}) + \log \xi(x_{1:n})\right] = \mathbb{E}_\xi \left[\log \frac{\xi(x_{1:n})}{\rho(x_{1:n})}\right] = D_{KL}(\xi \,||\, \rho) \geq 0, \tag{3}$$

with equality holding if and only if $\rho = \xi$ by the Gibbs inequality.

In the context of our the generic MBML setup, Equation (3) implies that the Bayesian mixture $\rho = \xi$ (as given by Equation (2)) is the unique optimal predictor in expectation. The set of all hypotheses/tasks in the mixture is called the model class $\mathcal{M}$. Neural networks trained to minimize log loss should thus converge towards the Bayes-optimal solution (see Ortega et al. (2019) for a detailed theoretical analysis). Two conditions need to be fulfilled for trained meta-learners to behave Bayes-optimally:

1. Realizability: the amortized Bayes-optimal solution needs to be representable by the model with the right set of parameters.
2. Convergence: training needs to converge to this set of parameters.

The hope is that by using sufficiently powerful function approximation techniques such as modern neural network architectures in an MBML setup, we can circumvent the need for explicit Bayesian inference and instead get the computational advantages associated with the Bayes-optimal predictor from a learned model with fixed weights. But what properties of a model are needed for it to be sufficiently powerful? The next section formally shows the necessity of using models with memory to achieve the Bayesian ideal. After establishing theoretically that Bayes-optimal predictors require memory, it is far from clear that memory-based neural network architectures achieve realizability (i.e., have a set of parameters that represents the Bayes-optimal predictor) and convergence (via mini-batch based SGD). We investigate these questions empirically in Section 7.

4. The Essential Role of Memory

It is important to emphasize that a fixed-parametric *memoryless* model cannot, in general, learn the Bayesian mixture predictor ξ (with model class $\mathcal{M}$). The intuition is that the Bayesian mixture requires computation of posterior mixture weights, which, in general, depend on the history observations (the sufficient statistics) and thus necessitate some form of memory. We now state this formally.

Definition 4.1. A model ν is defined to be memoryless if ν can be written in the form $\nu_\Theta(x_{1:n}) := \prod_{i=1}^{n} \nu_{\theta_i}(x_i)$, where $\Theta = (\theta_i)_{i=1}^{n}$ for all $x_{1:n}$.

In other words, ν_θ is a product measure. Next we present a negative result which explicitly quantifies the limitations of memoryless models to approximate general Bayesian inference.

Theorem 4.2. *Assume there exist $\mu_1, \mu_2 \in \mathcal{M}$ such that $\exists a_{1:\infty} : |\mathbb{E}_{\mu_1}[\mu_1(a_t|x_{<t})] - \mathbb{E}_{\mu_2}[\mu_2(a_t|x_{<t})]| \not\to 0$. Then there does not exist a $\Theta = (\theta_t)_{t=1}^{\infty}$ for a memoryless model ν_Θ such that for all $\mu \in \mathcal{M}$ we have $\mathbb{E}_\mu|\nu_\Theta(a_t|x_{<t}) - \xi(a_t|x_{<t})| \to 0$ as $t \to \infty$.*

For instance, for μ_i = Bernoulli(ϑ_i), which are in most classes $\mathcal{M}$, we have $|\mathbb{E}_{\mu_1}[\mu_1(a_t|x_{<t})] - \mathbb{E}_{\mu_2}[\mu_2(a_t|x_{<t})]| = |\vartheta_1 - \vartheta_2| \neq 0$ for any choice of $\vartheta_1 \neq \vartheta_2$.

The main intuition is that a discrete Bayesian mixture cannot always be represented as a product measure, as $\xi(x_n|x_{<n}) = \sum_{\rho\in\mathcal{M}} w^{\rho}_{n-1}\rho(x_n|x_{<n})$, where the posterior weight $w^{\rho}_{n-1} := w^{\rho}_0\,\rho(x_{<n})/\xi(x_{<n})$ for $n > 1$; in other words, w^{ρ}_{n-1} can depend upon the whole history. A complete proof is given in Appendix D.

Importantly, this argument is independent of the representation capacity of ν_θ, and for example still holds even if ν_θ is a universal function approximator, or if ν_θ can represent each possible $\rho \in \mathcal{M}$ given data *only* from ρ. The same argument extends to any k-Markov stationary model for finite k, though one would expect much better approximations to be possible in practice with larger k.

5. Priors and Exact Inference Baselines

This section describes our baseline Bayesian algorithms for exact Bayesian inference on piecewise stationary Bernoulli data. The algorithms make different assumptions regarding the statistical structure of switching-points. If the data generating source satisfies these assumptions, then the baselines are theoretically known to perform optimally in terms of expected cumulative regret. This allows us to assess the quality of the meta-learned solutions against known optimal predictors. Note that while exact Bayesian inference is often computationally intractable, the cases we consider here are noteworthy in the sense that they can be computed efficiently, and in some cases with quite elaborate algorithms involving combinations of dynamic programming (see Koolen & de Rooij (2008) for a comprehensive overview) and the generalized distributive law (Aji & McEliece, 2000).

In order to ensure that the data generating source matches the statistical prior assumptions made by the different baselines, we use their underlying priors as data generating distributions in our experiments (see Appendix E for details on the algorithms that sample from the priors).

KT Estimator The KT estimator is a simple Beta-Binomial model which efficiently implements a Bayesian predictor for Bernoulli(θ) sources with unknown θ by maintaining sufficient statistics in the form of counts. By using a Beta($\frac{1}{2}, \frac{1}{2}$) prior over θ, we obtain the KT-estimator (Krichevsky & Trofimov, 1981), which has optimal worst case regret guarantees with respect to data generated from an unknown Bernoulli source. Conveniently, the predictive probability has a closed form

$$\textsc{kt}(x_{n+1} = 1|x_{1:n}) = \frac{c(x_{1:n}) + \frac{1}{2}}{n+1},$$

where $c(x_{1:n})$ returns the number of ones in $x_{1:n}$, and $\textsc{kt}(x_{n+1} = 0|x_{1:n}) = 1 - \textsc{kt}(x_{n+1} = 1|x_{1:n})$. This can be implemented efficiently online by maintaining two counters, and the associated marginal probability can be obtained via the chain rule $\textsc{kt}(x_{1:n}) = \prod_{i=1}^{n} \textsc{kt}(x_i|x_{<i})$. The KT estimator cannot handle (piecewise) non-stationary distributions; to allow for this we next make a simple extension, and later more complex extensions.

KT Oracle Our first baseline extends the KT estimator to deal with piecewise stationarity: KT Oracle is provided with knowledge of when switching-points occur. This allows using a KT estimator and simply resetting its counters at each switching-point. The KT Oracle serves as a lower bound to show achievable regret in case switching-points could be instantaneously predicted with perfect accuracy. The prior underlying the KT Oracle is never used to generate data in our experiments, since the KT Oracle does not specify a distribution over switching-points.

PTW**: Partition Tree Weighting** Our second baseline is Partition Tree Weighting (Veness et al., 2013). In contrast to the KT Oracle, PTW does not need to observe switching-points. Instead, it performs Bayesian model averaging over a carefully chosen subset $\mathcal{C}_d \subset \mathcal{T}_n$ of temporal partitions by computing

$$\textsc{ptw}_d(x_{1:n}) = \sum_{\mathcal{P}\in\mathcal{C}_d} 2^{-\Gamma_d(\mathcal{P})} \prod_{(a,b)\in\mathcal{P}} \rho(x_{a:b}),$$

where ρ is a base-predictor for a single segment (in our case the KT-estimator), and d is the depth of the partition tree which needs to be at least $\log n$. In other words, the technique gives a way to extend a given base predictor ρ to a piecewise setting, with known worst case regret guarantees that follow from the use of model averaging over a tree structured prior. Although the number of partitions $|\mathcal{C}_d|$ grows $O(2^{2^d}) = O(2^n)$, this technique adds only a $O(\log n)$ time/space overhead compared with computing $\rho(x_{1:n})$, and can be computed online in a recursive/incremental fashion. In this work we restrict our attention to the case where the base model is the KT-estimator, $\rho = \textsc{kt}$, to obtain a low-complexity universal algorithm for piecewise Bernoulli sources. Informally, PTW assumes that a trajectory has a switching-point at half its length with probability $1/2$, and both resulting sub-trajectories also have a switching-point at their respective halves with probability $1/2$, and so on (recursively) for all subsequent sub-trajectories. This assumption allows for efficient implementation and leads to a characteristic inductive bias. In our experiments we investigate whether neural models can meta-learn this structured inductive bias and match prediction performance of PTW on data that follows these assumptions.

LIN**: Exact Model Averaging Over All Temp. Partitions** Our final baseline, LIN, is the linear complexity method introduced by Willems (1996). It performs Bayesian model averaging over all temporal partitions (whereas PTW only considers a subset), and all possible Bernoulli models within each segment, and has the marginal form

$$\text{LIN}(x_{1:n}) = \sum_{\mathcal{P} \in \mathcal{T}_n} w(\mathcal{P}) \prod_{(a,b) \in \mathcal{P}} \text{KT}(x_{a:b}),$$

where $w(\mathcal{P})$ is a prior over the linear-transition diagram representation of $\mathcal{P}$, the details of which are not important for this work, but they introduce a different assumption over the distribution and location of switching-points compared to PTW. To process a sequence of n symbols, this algorithm runs in time $O(n^2)$ and has space complexity of $O(n)$. In our experimental section we also investigate whether neural models can meta-learn to match the inductive bias of LIN.

6. Methodology

The general approach for our experiments is to train various memory-based neural models according to the MBML training setup described in Section 3. We explore multiple neural architectures to get a better sense as to how architectural features influence the quality of the meta-learned Bayesian approximation. After training, we evaluate models either on data drawn from the same meta-distribution as during training (on-distribution experiments) or from a different distribution (off-distribution experiments). We quantify prediction performance by the expected cumulative regret (called redundancy in information theory) with respect to the ground-truth piecewise data generating source μ, quantifying the expected excess log loss of the neural predictor. More formally, we define the expected instantaneous regret of model π at time t with respect to the piecewise source μ as

$$R_{\pi\mu}(t) := \mathbb{E}_{x_t \sim \mu^{f(t)}} \left[\log \mu^{f(t)}(x_t) - \log \pi(x_t) \right],$$

compare Equation (3), and the cumulative expected regret as

$$R^T_{\pi\mu} := \sum_{t=1}^{T} R_{\pi\mu}(t). \tag{4}$$

An illustration of both metrics is shown in Figure 1. Note that a cumulative expected regret of zero corresponds to the performance of an oracle which knows both the location of the switching-points, as well as the parameter of each Bernoulli process governing a segment.

We now introduce the different types of data generating sources used in our experiments, before describing the different types of memory-based neural models that we evaluated.

Data-Generation We consider data sources that are piecewise stationary in the form given by Equation (1). Within a stationary segment i, μ^i is a Bernoulli distribution with bias sampled from a Beta prior $\mu^i \sim \text{Beta}(\alpha, \beta)$; see Figure 1 for a concrete example. In our experiments, we always use $\alpha = \beta = 0.5$, which is consistent with the prior used by the KT-estimator.

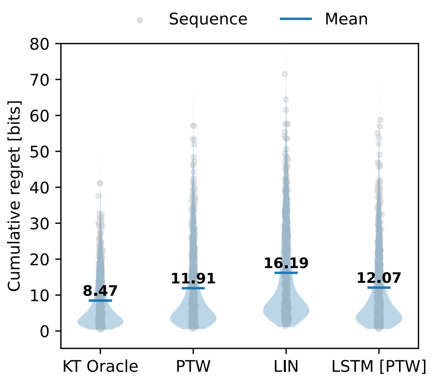
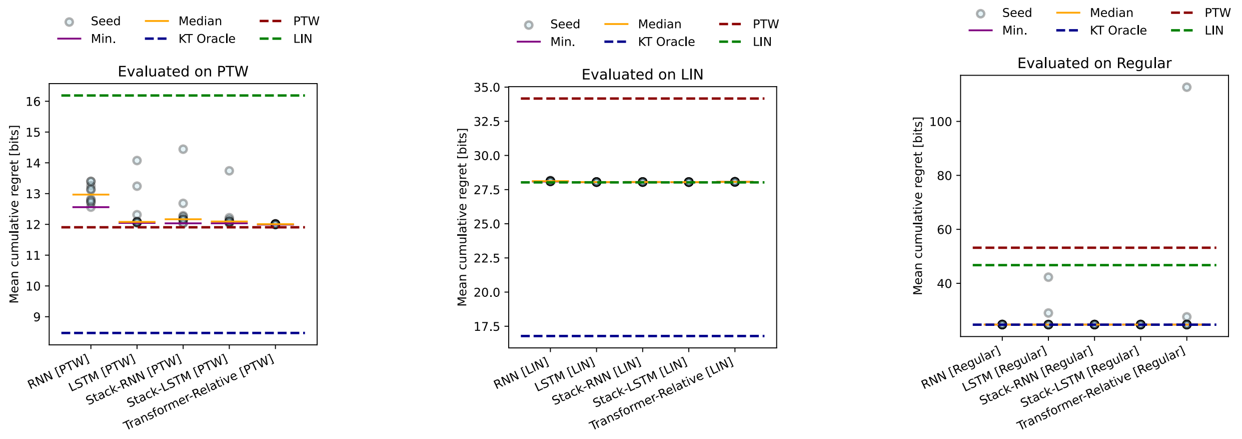


Figure 2. Mean cumulative regret across 10k sequences of length 256 drawn from PTW prior (same setting as Figure 1). The LSTM trained on data from the PTW prior matches prediction performance of the optimal PTW predictor. We also compare against LIN, a strong but suboptimal predictor for this distribution.

Across our experiments, we consider four different distributions over switching-points, two of which coincide with the statistical assumptions of our exact inference baselines (PTW and LIN):

- **Regular Periodic:** All segments have fixed length l, meaning that switching-points occur deterministically at the same locations across all sampled trajectories. Neural predictors can, during meta-learning, pick up on l and thus learn to predict switching-points with perfect accuracy.
- **Random Uniform:** Segment-lengths are repeatedly drawn from a Uniform$(1, n)$ distribution until the combined summed segment length matches or exceeds the desired sequence length n.
- PTW **prior:** Switching-points are sampled from the PTW prior. More specifically, a temporal partition can be sampled from the PTW$_d$ prior using Algorithm 1 with an expected running time of $O(d)$, where d is the depth of the partition-tree; see Appendix E for more detail. Unless otherwise indicated, PTW in our experiments refers to using the minimally necessary depth for the given sequence length, e.g., PTW$_8$ for length 256 and PTW$_9$ for length 512.
- LIN **prior:** Switching-points are sampled from the LIN prior. Algorithm 2 in Appendix E provides a method for sampling temporal partitions from the LIN prior, whose worst-case time and space complexity grows linearly with the sequence length n.



(a) PTW$_8$. (b) LIN. (c) Regular shifts with period 20.

Figure 3. On-distribution performance (models trained and evaluated on same distribution, denoted below panels). Evaluation on 10k sequences of length 256. Columns in each plot show individual trained models (circles), and minimum and median results across random initializations. Square-brackets denote the training distribution for models. Dashed lines show the three exact Bayesian inference algorithms as dashed lines—of course PTW and LIN are only optimal for their respective data regimes, but serve as a strong baseline predictor in the other regimes.

Example draws and visualizations of the switching-point statistics of all prior distributions are shown in Appendix B.

Neural Predictors Our neural models sequentially observe binary samples from the data generating source and output probabilities over the next observation. $\pi_\theta(\cdot|x_{<t})$ given their parameters θ and the data seen so far up to time t. We use the logarithmic loss for training; for a sequence up to time T, we have $\ell_\theta(x_{1:T}) := -\frac{1}{T}\sum_{t=1}^{T} \log \pi_\theta(x_t|x_{<t})$. During training, parameters are updated via mini-batch stochastic gradient descent using ADAM.

We evaluate the following network architectures:

- **RNN:** One layer of vanilla RNN neurons, followed by a two-layer fully connected read-out.
- **LSTM:** One layer of LSTM (Hochreiter & Schmidhuber, 1997) memory cells, followed by a two-layer fully connected read-out.
- **Stack-RNN/LSTM:** We also augment the LSTM and RNN predictors with a stack, similar to the Stack-RNN of Joulin & Mikolov (2015). The stack has three operations, PUSH, POP, and NO-OP, which are implemented in a "soft" fashion for differentiability, i.e., stack updates are computed via a linear combination of each stack-action probability. At each time-step the RNN/LSTM reads the top of the stack as an additional input. A push writes a lower-dimensional projection of the RNN/LSTM cell states to the top of the stack. We treat the dimensionality of the projection and the maximum depth of the stack as hyperparameters.
- **Transformer:** We use a Transformer encoder with incremental causal masking to implement sequential online prediction. The context of the transformer thus acts as a (verbose) memory, storing all observations seen so far. In our ablations we also simulate having a smaller context length (via masking), but the best results are achieved with the full context. We evaluate three different positional encodings (see Appendix A): standard sin/cos (Vaswani et al., 2017), ALiBi (Press et al., 2022), and the relative positional encodings from TransformerXL (Dai et al., 2019). For our experiments in Section 7, we use the relative encoding, as it performed best in the ablations.

For all our network architectures, we conducted an initial ablation study to determine architecture hyperparameters (see Appendix A). The experimental results shown in Section 7 use the hyperparameter-set that led to the lowest expected cumulative redundancy in the ablations (we provide the exact values in Appendix A).

We provide an open-source implementation of our models, tasks, and training and evaluation suite at `https://github.com/deepmind/nonstationary_mbml`.

7. Results

To clarify how our main results are computed, an example sequence from a PTW source, and corresponding model predictions, as well as our performance metric, are shown in Figure 1; example draws from the other sources are in Appendix B. To compare models' performance we empirically compute the mean cumulative regret across 10k sequences, see Figure 2. Finally, we perform the same evaluation over 10 different random initializations for each model.

On-Distribution Evaluation We first evaluate the performance of neural models when trained and evaluated on the same data generating distribution—results shown in Figure 3. Generally, we find that neural models match prediction performance of the Bayes-optimal predictors very well on their respective data regimes. Picking the best random initialization (Min in the figure), all neural predictors achieve near-optimal performance, except the RNN which has a slightly larger error on the PTW data. Median results (across random initializations) reveal some differences in training stability. It is quite remarkable that all neural models across all random seeds, when trained on LIN data, manage to match LIN performance almost exactly. Somewhat less surprising, for regular periodic shifts all neural models quite reliably learn to predict switching-points with perfect accuracy, allowing them to reach KT Oracle performance levels. Figure 23 in the Appendix shows on-distribution evaluation results for the Random Uniform distribution.

Off-Distribution Evaluation The experiments in this section serve to illustrate that models pick up precise inductive biases during meta-learning. Biases, that match the statistical structure of the data distribution during training. If the data distribution at test time violates this statistical structure, optimal prediction performance can no longer be guaranteed. Figure 4 shows how models trained on data from the PTW and LIN prior perform when evaluated with data drawn from a random uniform changepoint distribution. Overall, neural networks trained on PTW are slightly more robust against this change compared to PTW—the better neural models fit PTW in Figure 3 (a), the less robust they seem to be against this distributional shift. Off-distribution generalization for the models trained on LIN is very uniform across models and closely aligned with the exact inference implementation in terms of prediction performance. We show more off-distribution evaluations in Appendix C.2.

Sequence-Length Generalization Figure 5 shows length-generalization behavior of the neural models. All models shown are trained on sequences of length 256 but evaluated on much longer sequences. As expected the models' performance degrades with longer sequences, but remains reasonably good, indicating that, e.g., internal dynamics of the recurrent networks do not break down catastrophically. See Figure 27 for an example trajectory for the LSTM evaluated on a sequence of length 512, showing that predictions overall remain quite close to the optimum.

Note that the most likely switching-points under the PTW prior depend on the sequence length, and thus our sequence-length generalization experiment also induces a slight distributional shift (models trained on length 256 have a different prior expectation over switching point locations than the PTW prior assigns for shorter or longer sequence lengths). To quantify this effect Figure 6 shows results of a sequence-length ablation that compares two types of models: one, models trained on length 32 and evaluated on shorter and longer lengths (suffering from the implicit distributional shift that arises from PTW priors of different depth), and two, models evaluated on the length that they were trained on (for a range of different lengths).

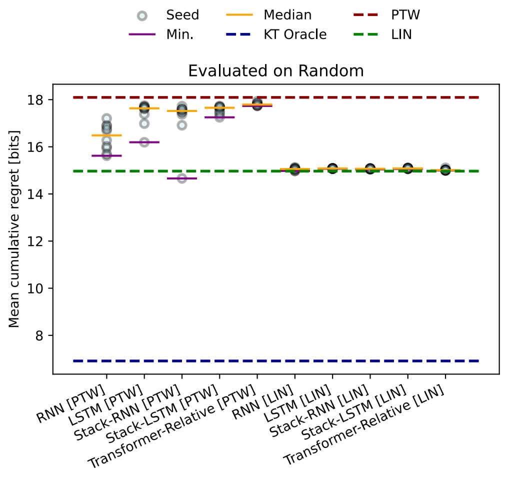


Figure 4. Off-distribution evaluation (10k sequences, length 256). Models' training distribution indicated in the square brackets. All models are evaluated with a random-uniform distribution over segment lengths (Uniform$(1, 256)$). Red dashed line shows PTW_8.

8. Related Work and Discussion

Meta-learning is a technique for producing data-efficient learners at test time through the acquisition of inductive biases from training data (Bengio et al., 1991; Schmidhuber et al., 1996; Thrun & Pratt, 1998). Recently, Ortega et al. (2019) showed theoretically how (memory-based) meta-learning leads to predictors that perform amortized Bayesian inference, i.e., meta-learners are trained to minimize prediction error (log loss) over a task distribution which requires (implicit) inference of the task at hand. Minimal error is achieved by taking into account a priori regularities in the data in a Bayesian fashion and, in decision-making tasks, implies automatically trading-off exploration and exploitation (Zintgraf et al., 2020). Memory-based meta-learners pick up on a priori statistical regularities simply by training over the distribution of tasks without directly observing task indicators. This leads to parametric functions that implement amortized Bayesian inference (Gershman & Goodman, 2014; Ritchie et al., 2016), where a parametric model π_θ behaves as if performing Bayesian

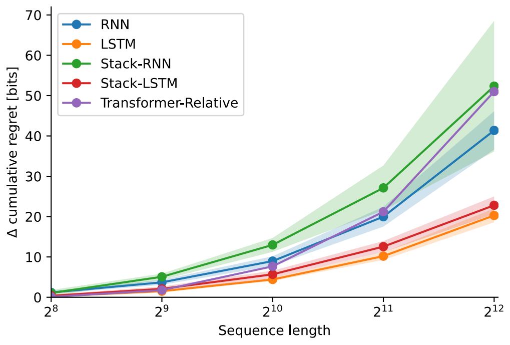


Figure 5. Evaluation of models on longer sequences. Models are trained on length 256 with switching-points drawn from PTW$_8$ (same as Figure 3 (a)) and evaluated on sequences up to length 4096 (depth of PTW is $\log_2$(sequence length)). The plot shows the difference between the models' cumulative regret and PTW over 1k sequences. Lines show the mean and shaded areas the standard deviation over 10 random seeds. The LSTM and Stack-LSTM generalize best, but for all models performance degrades as the sequence length increases beyond the training length, which is a signature of learned amortized inference.

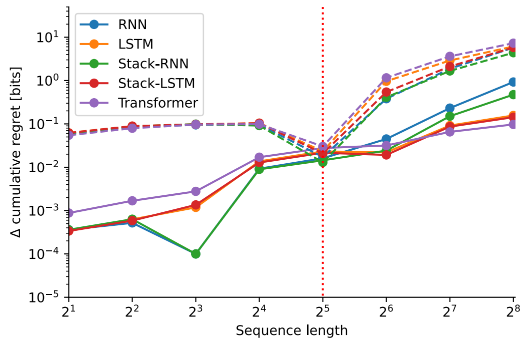


Figure 6. Evaluation of models on sequences of different lengths. The plot shows the difference between the models' expected cumulative regret and PTW over 1k sequences (depth of PTW is $\log_2$(sequence length)). Results are averaged over 10 random seeds. Solid lines correspond to models evaluated on the length they were trained on. Dashed lines correspond to models trained on length 32 (dotted vertical line) and evaluated on other lengths. As expected, models trained on 32 generalize worse to other lengths ('U' shape curve), which is explained by the implicit distributional shift induced by the PTW prior with different depth.

inference "under the hood": $\pi_\theta(x_{<t}) \approx p(x_t|x_{<t}) = \sum_\tau p(x_t|\tau, x_{<t})p(\tau|x_{<t})$. The r.h.s. requires posterior inference over the task-parameters $p(\tau|x_{<t}) \propto p(x_{<t}|\tau)p(\tau)$, which is often analytically intractable. The result is a model with fixed parameters that implements an adaptive algorithm via its activations, and at its core is the collection of sufficient statistics for rapid online task inference. The argument can be extended to Bayes-optimal decision-making (Ortega et al., 2019; Mikulik et al., 2020); recently, Adaptive Agent Team et al. (2023) reported a large-scale demonstration of the principle, where models are trained over 25 billion distinct tasks in simulated 3D environments. Trained models are able to adapt to novel tasks on human time-scale (i.e., with tens or a few hundreds of seconds of interaction) purely via in-context learning (conditioning). Kirsch et al. (2022) also conducted an exploration of memory-based meta-learning over a vast set of tasks to produce in-context and few-shot learning abilities, with up to 2^{24} tasks created by randomly projecting inputs and randomly permuting labels on MNIST. They find that having both, a large enough model and a rich enough training distribution is required for an in-context learning algorithm that generalizes.

While Bayes-optimality in sequential prediction and decision-making is theoretically well understood, cf. Hutter (2005), an important question is whether neural networks, when meta-trained appropriately, can approach the Bayesian solution at all (realizability and convergence, see Section 3), or whether they operate primarily in a suboptimal regime that is not well described by Bayesian theory. Mikulik et al. (2020) conducted a first targeted empirical comparison of meta-learned neural predictors with Bayes-optimal algorithms, focusing on simple prediction- and decision-making tasks where episodes had a fixed number of steps, and changepoints were observed (internal memory states were explicitly reset at episode boundaries). This setting is similar to our regular periodic switching-point distribution, but, crucially, switching-points are always unobserved in our experiments. That is, the emphasis of our study is on non-stationary data sources with abrupt changes in local statistics. While piecewise stationary sources are conceptually simple, the switching-points make accurate prediction challenging, particularly under a cumulative error metric. Furthermore, piecewise Bernoulli data makes switching-point detection difficult, which is, counter-intuitively, often easier on more complex distributions when different segments exhibit strongly characteristic statistics. In Reed et al. (2022), observations are, for instance, frames from Atari games, where a single frame often suffices to determine the task accurately.

We also aim at furthering the understanding of inductive biases and reasoning principles acquired by sequential predictors such as large language models. Recently observed in-context learning abilities in large language models (Brown et al., 2020) have rekindled interest in black-box parametric models capable of learning-to-learn purely in-context, that is, via activations, with frozen parameters (Hochreiter et al., 2001; Duan et al., 2016; Santoro et al., 2016; Wang et al., 2017). While the capabilities to learn in-context have been heavily explored empirically, the connections to Bayesian theory are still somewhat sparse (Ortega et al., 2019; Mikulik et al., 2020; Müller et al., 2022; Xie et al., 2022). From an AI safety viewpoint it is desirable to understand the mechanisms that enable few-shot and in-context learning; which are plausibly the same mechanisms that create susceptibility to prompt injections and context poisoning attacks. These characteristics are expected from a model that performs implicit Bayesian inference over piecewise stationary data. For instance, Xie et al. (2022) argued that in-context learning in large language models can be explained by (implicit) Bayesian inference over a latent variable, but does not draw a connection to the theory of meta-learning (which explains why amortized Bayesian inference arises from minimizing log loss) and does not compare against a known Bayes-optimal algorithm to establish optimality of the neural predictor. Our meta-learning interpretation is in line with the arguments in Xie et al. (2022) but is more general. Our interpretation also does not rely on special delimiter characters that signal a topic switch and needing to have a posterior over the latent variable that is highly concentrated on a single value. We believe it could be interesting in the future to contrast the meta-learning interpretation with the model by Xie et al. (2022) and extend our experimental suite to incorporate their hidden Markov model as a more complex piecewise stationary source.

Limitations Our results show the potential of memory-based meta-learning to accurately approximate Bayes-optimal solutions. However, our findings are currently limited to Bernoulli statistics per segment, and four types of switching-point distributions. For known Bayes-optimal algorithms the complexity of dealing with different switching-point distributions seems to dominate over increasing the complexity of the base distributions per segment. This makes us optimistic that our findings would generalize to more complex per-segment distributions when training neural predictors—but at the current stage this remains speculative. The main challenge with more complex data generating sources, such as real-world datasets, is the lack of a (computationally or analytically) tractable Bayes-optimal solution against which we could compare. The main point of this paper is to demonstrate that neural networks can learn to predict Bayes-optimally and not simply that they can learn to predict well (which has already been demonstrated extensively in the literature). Another limitation of our study is that many known Bayes-optimal algorithms come with performance guarantees and robustness bounds, and while our generalization experiments attempt to shed some light on robustness and out-of-distribution behavior of meta-learned neural models, no formal guarantees can be provided.

9. Conclusion

In this paper we investigated whether neural networks, trained to minimize sequential prediction error (log loss) over statistically structured but highly non-stationary data sources, can learn to match the prediction performance of Bayes-optimal algorithms. We found this to be the case, despite non-trivial algorithmic requirements for optimal prediction in these settings. Our results empirically confirm the theoretical Bayesian interpretation of memory-based meta-learning (Ortega et al., 2019), which states that log-loss minimization on a meta-distribution over data sources with a memory-based parametric model leads to approximately Bayes-optimal solutions. By focusing on piecewise stationary data sources, we study a highly relevant regime that holds the promise to shed light onto recently observed capabilities of large sequential prediction models. We believe that few-shot and in-context learning abilities of these models, as well as their susceptibility to context-corruption and prompt-injection attacks at test time, can be better understood from the viewpoint of inferring changes in local statistics under a non-stationary distribution. A more concrete, and near-term take-away from our study is to highlight the potential of using memory-based meta-learning to *learn* (near-) Bayes-optimal predictors in settings where closed-form solutions are not obtainable or algorithmically intractable. The ingredients to succeed with this are highly expressive parametric models (for realizability of the Bayes-optimal predictor) and strong optimizers (to ensure convergence)—our current study shows that modern neural networks in a standard meta-learning setup with mini-batch based SGD can fit this bill.

Acknowledgements

We thank Jane Wang, Christopher Mattern, and Shane Legg for their helpful feedback and insightful conversations.

References

Adaptive Agent Team, Bauer, J., Baumli, K., Baveja, S., Behbahani, F. M. P., Bhoopchand, A., Bradley-Schmieg, N., Chang, M., Clay, N., Collister, A., Dasagi, V., Gonzalez, L., Gregor, K., Hughes, E., Kashem, S., Loks-Thompson, M., Openshaw, H., Parker-Holder, J., Pathak, S., Nieves, N. P., Rakicevic, N., Rocktäschel, T., Schroecker, Y., Sygnowski, J., Tuyls, K., York, S., Zacherl, A., and Zhang, L. Human-timescale adaptation in an open-ended task space. *CoRR*, abs/2301.07608, 2023.

Aji, S. M. and McEliece, R. J. The generalized distributive law. *IEEE Trans. Inf. Theory*, 46(2):325–343, 2000.

Bengio, Y., Bengio, S., and Cloutier, J. Learning a synaptic learning rule. In *IJCNN-91-Seattle International Joint Conference on Neural Networks*, 1991.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners. In *NeurIPS*, 2020.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J. G., Le, Q. V., and Salakhutdinov, R. Transformer-xl: Attentive language models beyond a fixed-length context. In *ACL (1)*, pp. 2978–2988. Association for Computational Linguistics, 2019.

Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., and Abbeel, P. Rl^2: Fast reinforcement learning via slow reinforcement learning. *CoRR*, abs/1611.02779, 2016.

Gershman, S. and Goodman, N. D. Amortized inference in probabilistic reasoning. In *CogSci*. cognitivesciencesociety.org, 2014.

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

Hochreiter, S., Younger, A. S., and Conwell, P. R. Learning to learn using gradient descent. In *ICANN*, volume 2130 of *Lecture Notes in Computer Science*, pp. 87–94. Springer, 2001.

Hutter, M. *Universal Artificial Intelligence: Sequential Decisions Based on Algorithmic Probability*. Springer, 2005.

Joulin, A. and Mikolov, T. Inferring algorithmic patterns with stack-augmented recurrent nets. In *NIPS*, pp. 190–198, 2015.

Kirsch, L., Harrison, J., Sohl-Dickstein, J., and Metz, L. General-purpose in-context learning by meta-learning transformers. *CoRR*, abs/2212.04458, 2022.

Koolen, W. M. and de Rooij, S. Combining expert advice efficiently. In *COLT*, pp. 275–286. Omnipress, 2008.

Krichevsky, R. E. and Trofimov, V. K. The performance of universal encoding. *IEEE Trans. Inf. Theory*, 27(2): 199–206, 1981.

Mikulik, V., Delétang, G., McGrath, T., Genewein, T., Martic, M., Legg, S., and Ortega, P. A. Meta-trained agents implement bayes-optimal agents. In *NeurIPS*, 2020.

Müller, S., Hollmann, N., Pineda-Arango, S., Grabocka, J., and Hutter, F. Transformers can do bayesian inference. In *ICLR*. OpenReview.net, 2022.

Ortega, P. A., Wang, J. X., Rowland, M., Genewein, T., Kurth-Nelson, Z., Pascanu, R., Heess, N., Veness, J., Pritzel, A., Sprechmann, P., Jayakumar, S. M., McGrath, T., Miller, K. J., Azar, M. G., Osband, I., Rabinowitz, N. C., György, A., Chiappa, S., Osindero, S., Teh, Y. W., van Hasselt, H., de Freitas, N., Botvinick, M. M., and Legg, S. Meta-learning of sequential strategies. *CoRR*, abs/1905.03030, 2019.

Press, O., Smith, N. A., and Lewis, M. Train short, test long: Attention with linear biases enables input length extrapolation. In *ICLR*. OpenReview.net, 2022.

Reed, S. E., Zolna, K., Parisotto, E., Colmenarejo, S. G., Novikov, A., Barth-Maron, G., Gimenez, M., Sulsky, Y., Kay, J., Springenberg, J. T., Eccles, T., Bruce, J., Razavi, A., Edwards, A., Heess, N., Chen, Y., Hadsell, R., Vinyals, O., Bordbar, M., and de Freitas, N. A generalist agent. *CoRR*, abs/2205.06175, 2022.

Ritchie, D., Horsfall, P., and Goodman, N. D. Deep amortized inference for probabilistic programs. *CoRR*, abs/1610.05735, 2016.

Santoro, A., Bartunov, S., Botvinick, M. M., Wierstra, D., and Lillicrap, T. P. Meta-learning with memory-augmented neural networks. In *ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, pp. 1842–1850. JMLR.org, 2016.

Schmidhuber, J., Zhao, J., and Wiering, M. Simple principles of metalearning. Technical report, IDSIA, 1996.

Thrun, S. and Pratt, L. Y. Learning to learn: Introduction and overview. In *Learning to Learn*, pp. 3–17. Springer, 1998.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *NIPS*, pp. 5998–6008, 2017.

Veness, J., White, M., Bowling, M., and György, A. Partition tree weighting. In *DCC*, pp. 321–330. IEEE, 2013.

Wang, J., Kurth-Nelson, Z., Soyer, H., Leibo, J. Z., Tirumala, D., Munos, R., Blundell, C., Kumaran, D., and Botvinick, M. M. Learning to reinforcement learn. In *CogSci*. cognitivesciencesociety.org, 2017.

Willems, F. M. J. Coding for a binary independent piecewise-identically-distributed source. *IEEE Trans. Inf. Theory*, 42(6):2210–2217, 1996.

Xie, S. M., Raghunathan, A., Liang, P., and Ma, T. An explanation of in-context learning as implicit bayesian inference. In *ICLR*. OpenReview.net, 2022.

Zintgraf, L. M., Shiarlis, K., Igl, M., Schulze, S., Gal, Y., Hofmann, K., and Whiteson, S. Varibad: A very good method for bayes-adaptive deep RL via meta-learning. In *ICLR*. OpenReview.net, 2020.

A. Architecture Ablation Study

We conducted an ablation study on the neural models we trained. This was used both to select the best parameters for the main experiments, and better understand the impact of number of parameters and memory size on the models' capabilities. We trained all the networks on sequences of length 256, sampled from the PTW prior. We trained 5 different seeds for each set of hyperparameters. We ran each distribution-architecture-hyperparameter triplet on a single GPU on our internal cluster.

Vanilla RNNs and LSTMs For these networks, we swept over three hidden sizes: 64, 128 and 256. We also swept over the number of dense layers to be appended after the recurrent core: 0, 1, or 2 layers. These layers all contain 128 neurons. The best performing models for both architectures were the largest ones, i.e., 256 neurons in the recurrent core and 2 extra dense layers of 128 neurons after the core. These are the hyperparameters we picked for the main study. In Figure 7, we plot the performance over the number of parameters of the model. The performance is the averaged cumulative regret over 10k trajectories sampled from the PTW prior. The figure generally reveals a downward trend: the more parameters, the lower the prediction error.

Stack-RNNs/LSTMs For these networks, we performed the same sweep as for simple RNNs and LSTMs above. In addition, we swept over the stack size (1, 8 or total sequence length, e.g., 256) and the stack cell width (1, 2 and 8 dimensions). The best performing models for both architectures were again the largest ones, i.e., 256 neurons in the core and 2 extra dense layers of 128 neurons after the core. These are the hyperparameters we picked for the main study. Figure 7 also shows the same trend as for standard RNNs and LSTMs. Furthermore, the best performing models for both architectures use a stack size of 8 and a stack cell size of 8 too. This means that the networks cannot store the whole history of observations in the stack (at least not straightfowardly), but this size seems sufficient and smaller stacks might make training easier.

Transformers For these networks, we used an embedding size of 64 and 8 heads. We swept over three positional encodings: classical sin/cos from the original Transformer paper, ALiBi which work well for short span dependencies, and the relative positional encodings from the TransformerXL paper. We also swept over the number of layers: 2, 4, 8 and 16. We first observe that all networks, regardless their size or positional encodings, train very well: The loss curves are smooth (not shown here) and the variance over seeds is small. The best performing models are the ones using the relative positional encodings and the largest ones, i.e., with 16 layers. These are the hyperparameters we picked for the main study. In Figure 8, we report the performance over the capacity of the model, measured in number of parameters, for the different positional encodings.

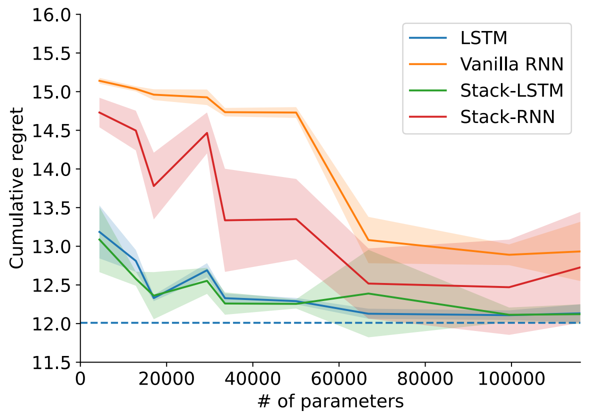


Figure 7. Cumulative regret (in bits) of the different RNNs, over their number of parameters. Dashed line shows PTW_8.

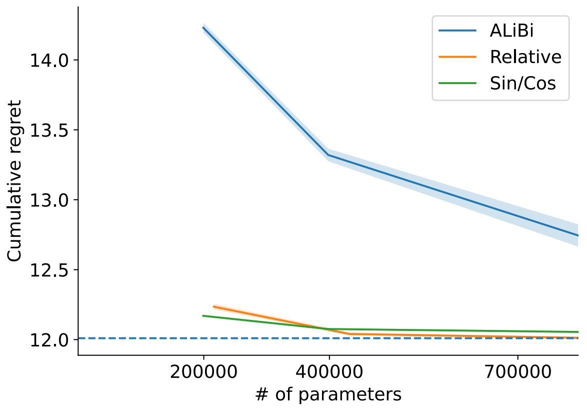


Figure 8. Cumulative regret (in bits) of the different Transformers, over their number of parameters. Dashed line shows PTW_8.

B. Illustration of Data Generating Sources

B.1. PTW Switching-Point Statistics

Positions of the Switching-Points To give a better intuition on where the PTW switching-points occur in the sequence, we plot their distribution in Figure 9. They are mostly present at half the sequence (probability 1/2), then at all the quarters of the sequence (probability 1/4), and so on, dividing by two the intervals recursively (and dividing the probability by 2).

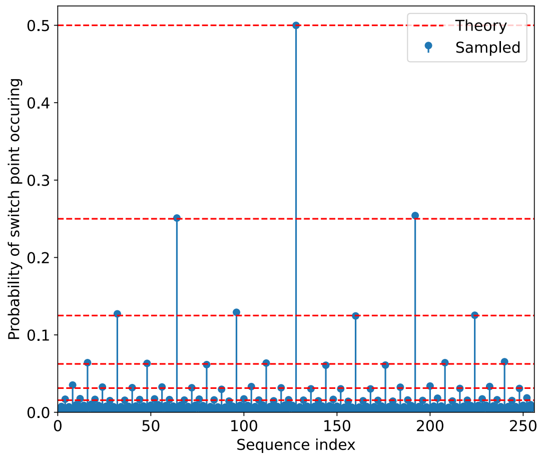


Figure 9. Distribution of PTW switching-points, over sequence indexes, computed over 10000 sequences. The length of the sequences is fixed to 256.

Number of switching-points We are also interested in the prior distribution of the number of switching-points, which is implicitly given by the PTW prior. We can get recursive and explicit formulas as follows: The recursive definition of the PTW distribution (Veness et al., 2013)

$$\text{PTW}_d(x_{1:n}) = \tfrac{1}{2}\rho(x_{1:N}) + \tfrac{1}{2}\text{PTW}_{d-1}\rho(x_{1:2^{d-1}})\text{PTW}_{d-1}\rho(x_{2^{d-1}+1:n}) \tag{5}$$

leads to the following recursion for the probability of k switching-points for $d \geq 1$

$$P_d[k] \;=\; \frac{1}{2}\delta_{k,0} + \sum_{l=1}^{k-1} P_{d-1}[k-l]\cdot P_{d-1}[l-1], \quad \text{and} \quad P_0[k] = \delta_{k,0} \tag{6}$$

(the number of switching-points is the number in the left half plus the number in the right half plus 1). From this we can compute $P_d[k]$ in time $O(d \cdot k_{\max})$. We plot the curves for $d = 0, \ldots, 9, \infty$ in Figure 10. We also plot the same curves in Figure 11, but from empirically sampling from our PTW data source and counting the number of switching-points. We sample 10 batches of 1000 sequences and report the mean and standard deviations of number of switching-points. The match is very good, with a very little statistical error.

The empirically observed kink at $k = d$ is indeed real for small d and gets washed out for larger d. It is easy to see from the recursion and from the plot that $P_d[k]$ is the same for all $d > k$. We can hence compute the limit for $d \to \infty$: A sequence with k switches corresponds to a full binary tree with k inner-switch nodes and $k+1$ leaves-segments. PTW assigns a probability $1/2$ to each decision of whether to switch or not. Therefore for such a partition $\mathcal{P}$ we have $2^{-\Gamma_d(\mathcal{P})} = (\frac{1}{2})^{k+(k+1)}$. There are $C(k)$ such trees, where $C(k) = \frac{(2k)!}{k!(k+1)!} = [1, 1, 2, 5, 14, 42, 132, \ldots]$ are the Catalan numbers. Therefore

$$P_\infty[k] \;=\; C(k)\cdot 2^{-\Gamma_d(\mathcal{P})} \;=\; \tfrac{(2k)!}{k!(k+1)!}2^{-2k-1} \;=\; P_d[k] \quad \text{for} \quad d > k \tag{7}$$

This expression can also be verified by inserting it into (6), using binomial identities. For large k, Stirling approximation gives $P_\infty[k] \approx k^{-3/2}/2\sqrt{\pi}$, which is quite accurate even for k as low as 1. This is good news: The prior distribution of switches is as close to non-dogmatic as possible: $1/k$ would not sum, $1/k^2$ is quite good, $1/k^{1.5}$ is even better, while $1/2^k$ would be very dogmatic and therefore bad. This good behavior is not a priori obvious. Indeed, if in PTW we would choose

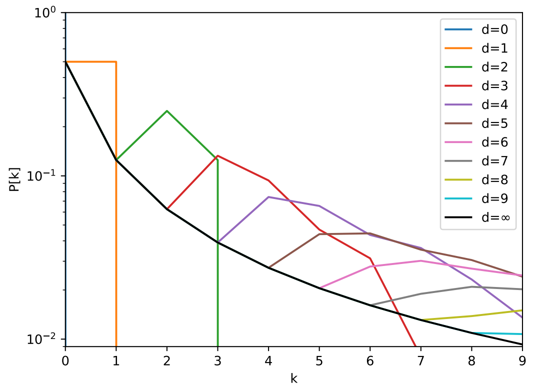


Figure 10. Theoretical PTW distribution of number of switches. For $k \geq d = 0, \ldots, 9$ (colored curves). For $k < d$, $P_d[k] = P_\infty[k]$ (black curve).

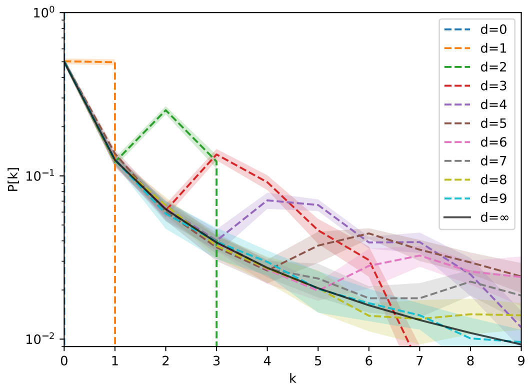


Figure 11. PTW empirical distribution of number of switches over 10 batches of 1000 sequences each (colored curves). We also added the theoretical case $P_\infty[k]$ (black curve).

the switch probability p anything but $1/2$ (larger or smaller!), $P_{\infty,p}[k] = P_\infty[k] \cdot 2(1-p) \cdot [4p(1-p)]^k$ which decreases exponentially in k for $p \neq 1/2$. From (5), we can also derive the expected number of switching-points

$$\mathbb{E}_d[k] \;=\; \tfrac{1}{2} \cdot 0 + \tfrac{1}{2}(1 + \mathbb{E}_{d-1}[k] + \mathbb{E}_{d-1}[k]) \;=\; \tfrac{1}{2} + \mathbb{E}_{d-1}[k] \;=\ldots=\; d/2 \tag{8}$$

which grows linearly with d (as expected) due to the tail of $P_d[k]$ being dragged out for $d \to \infty$. Similarly for $p \neq 1/2$ we have

$$\begin{aligned}\mathbb{E}_d[k] \;&=\; (1-p) \cdot 0 + p(1 + \mathbb{E}_{d-1}[k] + \mathbb{E}_{d-1}[k]) \;=\; p + 2p \cdot \mathbb{E}_{d-1}[k] \;=\ldots \\ \ldots =\; p \cdot [1 + 2p + (2p)^2 + \ldots + (2p)^{d-1}] \;&=\; p\frac{1-(2p)^d}{1-2p} \;\overset{d\to\infty}{\longrightarrow}\; \begin{cases} \frac{p}{1-2p} \text{ for } p < \frac{1}{2} \\ \frac{p}{2p-1}(2p)^d \text{ for } p > \frac{1}{2} \end{cases}\end{aligned}$$

That is, for $p < \frac{1}{2}$ this implies a prior believe of k (strongly) peaked around $\frac{p}{1-2p}$, not growing with d, while for $p > \frac{1}{2}$, it increases exponentially in d: $k \propto (2p)^d = n^\alpha$ with $0 < \alpha := \log_2(2p) < 1$.

B.2. Switching-Point Statistics for Other Priors

An example draw from the LIN prior is shown in Figure 12. Empirical switching-point statistics are in Figure 15 and Figure 16.

An example draw from the Random Uniform prior is shown in Figure 13. Empirical switching-point statistics are in Figure 17 and Figure 18.

An example draw from the Random Periodic prior is shown in Figure 14. Empirical switching-point statistics are in Figure 19 and Figure 20.

B.3. Models' Regret Along the Sequences

In Figure 21 we plot the average regret of the different models for all sequence indexes on 10000 sequences of length 256, drawn from the PTW prior. The models have also been trained on this prior. The match is almost perfect. We also plot the difference between the models' regret and PTW's regret in Figure 22, to emphasize the models' relative performance. Note that in theory, the models can do better than PTW on some indexes, but not when summing over all of them.

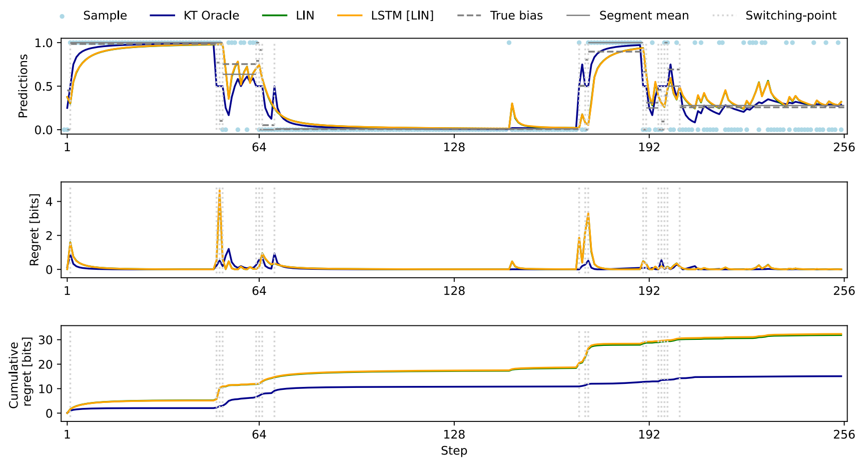


Figure 12. Example draw from LIN prior and model predictions.

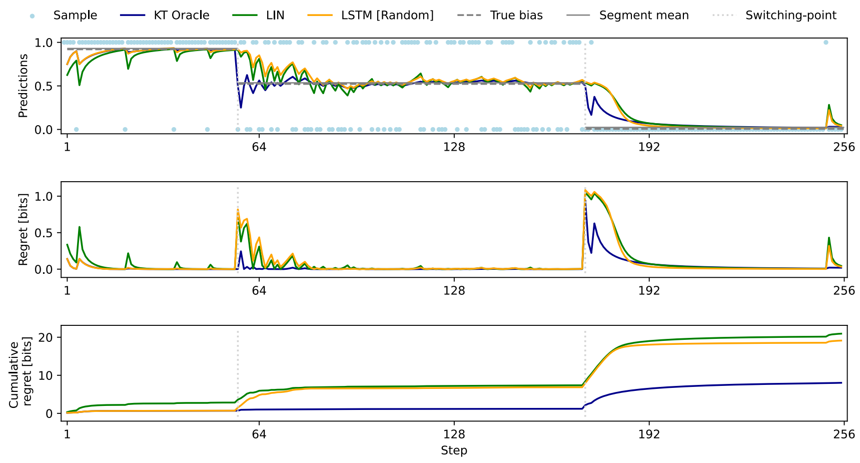


Figure 13. Example draw from Random Uniform prior (Uniform$(1, 256)$) and model predictions.

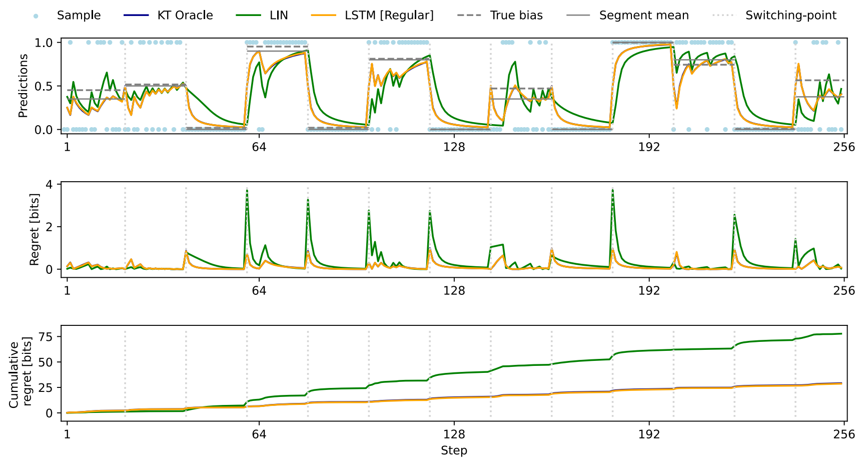


Figure 14. Example draw from Random periodic prior (period=20 steps) and model predictions.

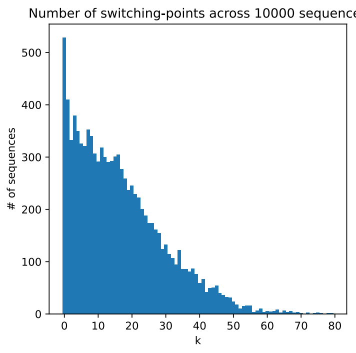


Figure 15. No. of switching-points per sequence (LIN prior).

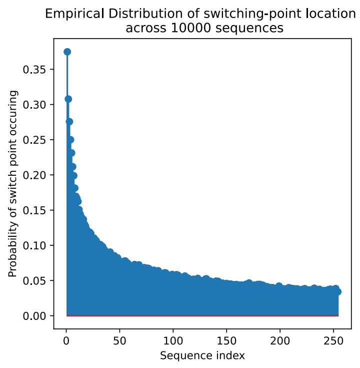


Figure 16. Switching-point locations (LIN prior).

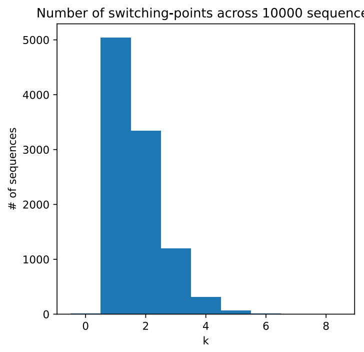


Figure 17. No. of switching-points per sequence (Random Uniform prior, Uniform$(1, 256)$).

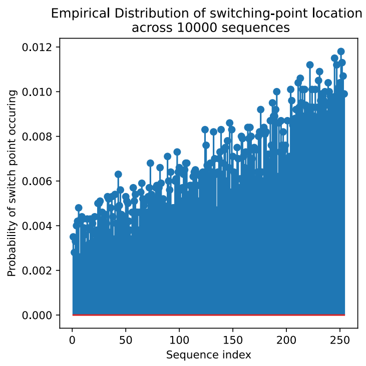


Figure 18. Switching-point locations (Random Uniform prior, Uniform$(1, 256)$).

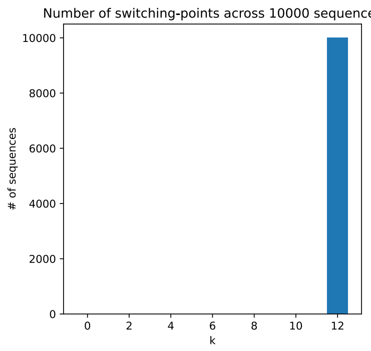


Figure 19. No. of switching-points per sequence (Regular Periodic prior, period=20 steps).

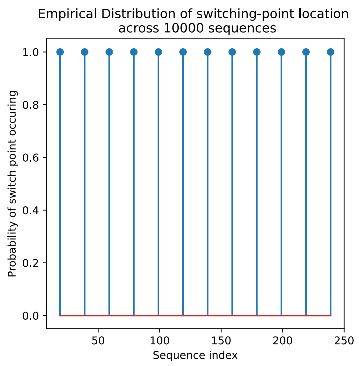


Figure 20. Switching-point locations (Regular Periodic prior, period=20 steps).

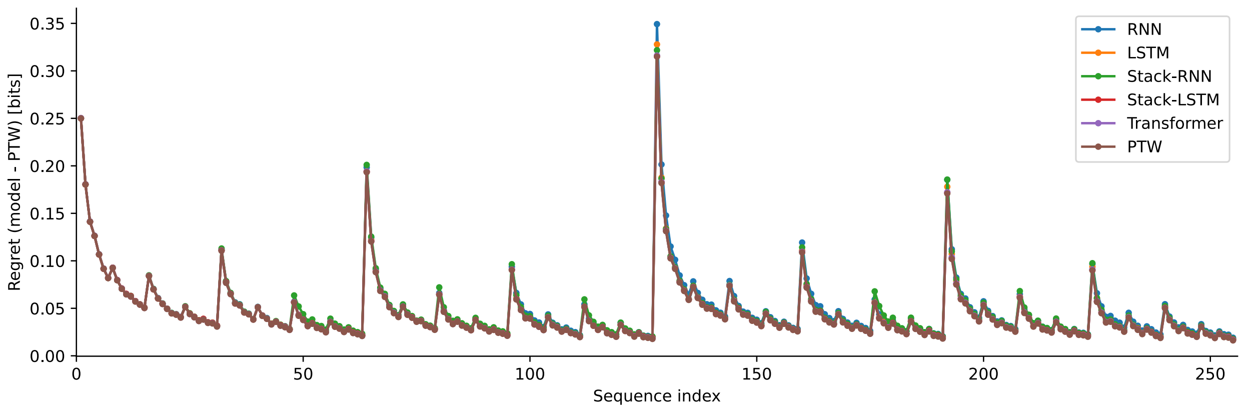


Figure 21. Average regret per sequence index, over 10000 sequences of length 256, drawn from the PTW prior.

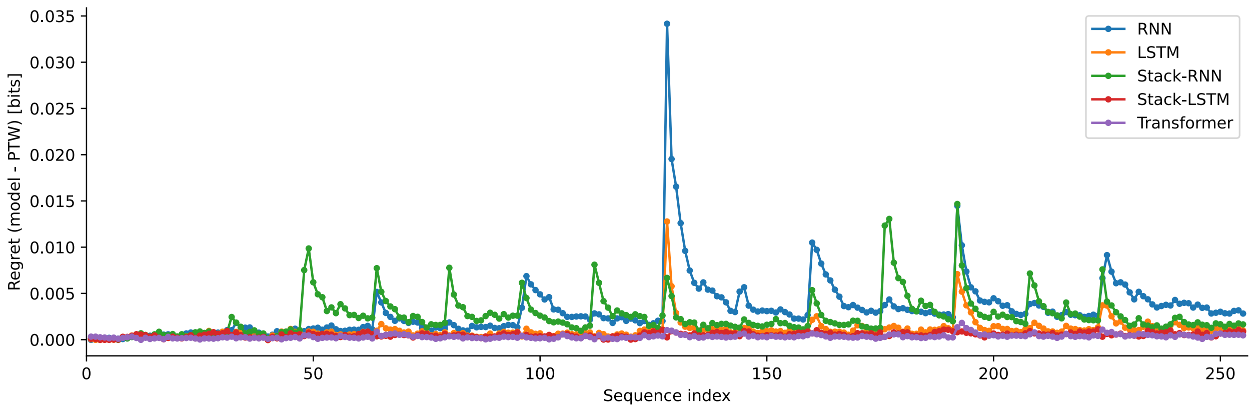


Figure 22. Difference of the average regret per sequence index, over 10000 sequences of length 256, drawn from the PTW prior.

C. Additional Experiments

C.1. On-Distribution Performance

Figure 23 shows the models' performance for training and evaluating on data with segment lengths drawn from a Random Uniform prior.

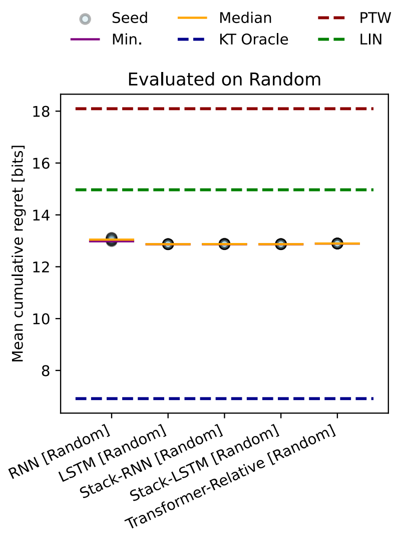


Figure 23. On-distribution evaluation (10k sequences, length 256). Models were trained and evaluated on data from the Random Uniform distribution (Uniform$(1, 256)$) over segment lengths. Note that we have no known exact Bayesian inference baseline in this case, though LIN comes with certain robustness guarantees that ensure good prediction performance in this setting. Neural networks trained precisely on this data distribution manage to outperform LIN though.

C.2. Off-Distribution Evaluation

Figure 24 shows how models trained on data from the PTW and LIN priors generalize to evaluating on data that follows Regular Periodic shifts. Figure 25 and Figure 26 show how models trained on Random Uniform segment lengths behave when evaluated on data from the PTW and LIN priors, respectively.

C.3. Evaluation on Longer Sequence Lengths at Test Time

See Figure 27, Figure 28, Figure 29, Figure 30, and Figure 31 for example sequences of length generalization of the different models. For a large-scale quantitative evaluation see Figure 5 in the main text. Finally, Figure 32 gives some insight into generalization behavior of the different models. In the figure, models were trained on sequences of length 256 drawn from PTW$_8$, but evaluated on sequences of length 512 drawn from PTW$_9$. In that case, the most likely change point occurs at 256, but since models were trained on trajectories of length 256 all models, except the transformer predict better than PTW$_9$ if no change point occurs (for all trajectories with 0 switching-points, roughly the upper half of each panel, there is a dark red band at 256). If the most likely change point actually occurs (trajectories with 1 or more switching-points), neural models predict the change at 256 with lower probability than PTW$_9$, leading to a white/blue band in the lower half of each panel. Similar trends are also seen for other highly likely switching-points such as 128 or 384, with the Stack-RNN showing the strongest white bands (consistent with having the worst performance in Figure 5).

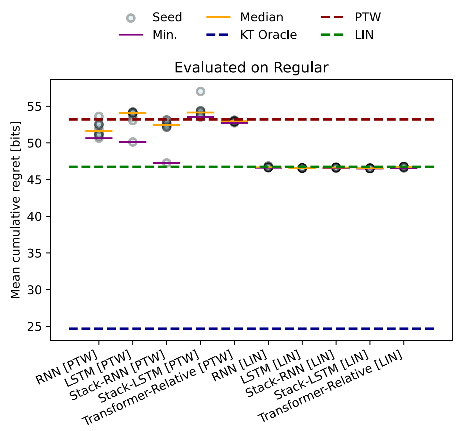


Figure 24. Off-distribution evaluation (10k sequences, length 256). The models' training distribution indicated in the square brackets. All models are evaluated with regular periodic segment lengths of period 20. Red dashed line shows PTW$_8$.

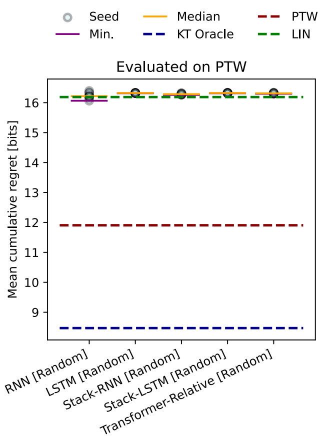


Figure 25. Off-distribution evaluation (10k sequences, length 256). Models were trained on data from Random Uniform segment lengths (Uniform$(1, 256)$) and evaluated on data from PTW$_8$.

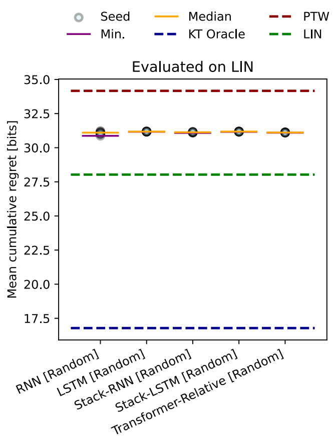


Figure 26. Off-distribution evaluation (10k sequences, length 256). Models were trained on data from Random Uniform segment lengths (Uniform$(1, 256)$) and evaluated on data from LIN.

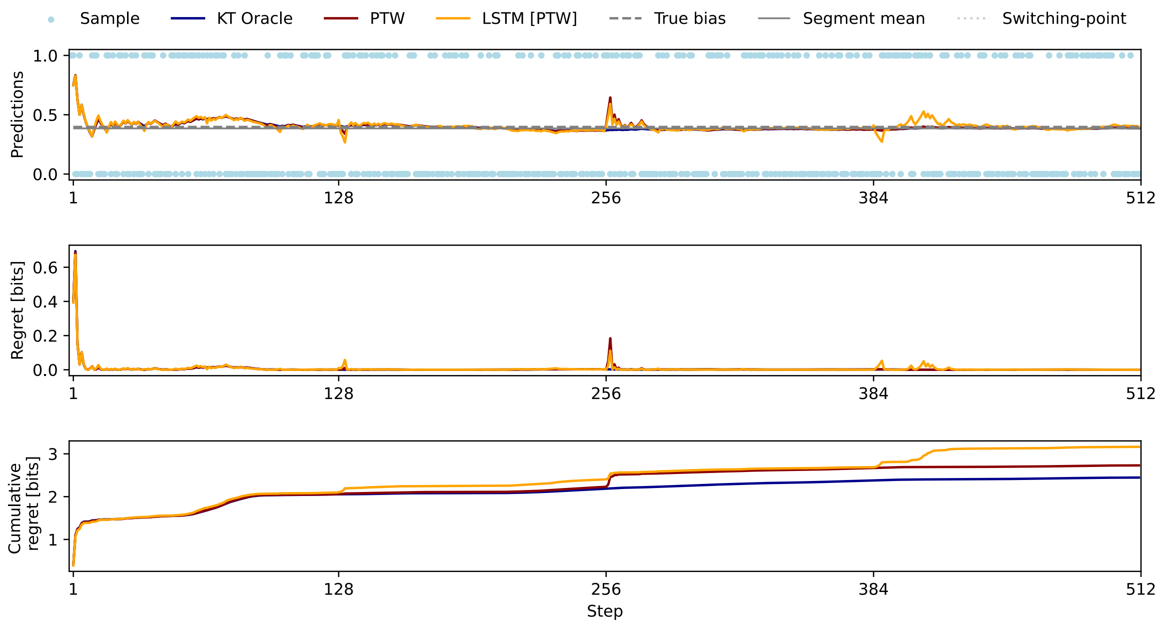


Figure 27. Sequence-length generalization: single sequence of length 512 without switching points (which is quite likely under PTW$_9$ prior). The LSTM predictions shown are taken from a model trained on sequences of length 256 (from PTW$_8$ prior). The LSTM generalizes well to sequences of longer length, taking the main hit in terms of cumulative regret (compared to PTW) around step 128, which is the most likely switching-point on the data that the model was trained on, and step 384 (which is a multiple of 128). Otherwise, predictions remain stable despite the sequence being twice as long as any sequence the model has ever experienced during training (which is an indicator that internal dynamics remain stable too).

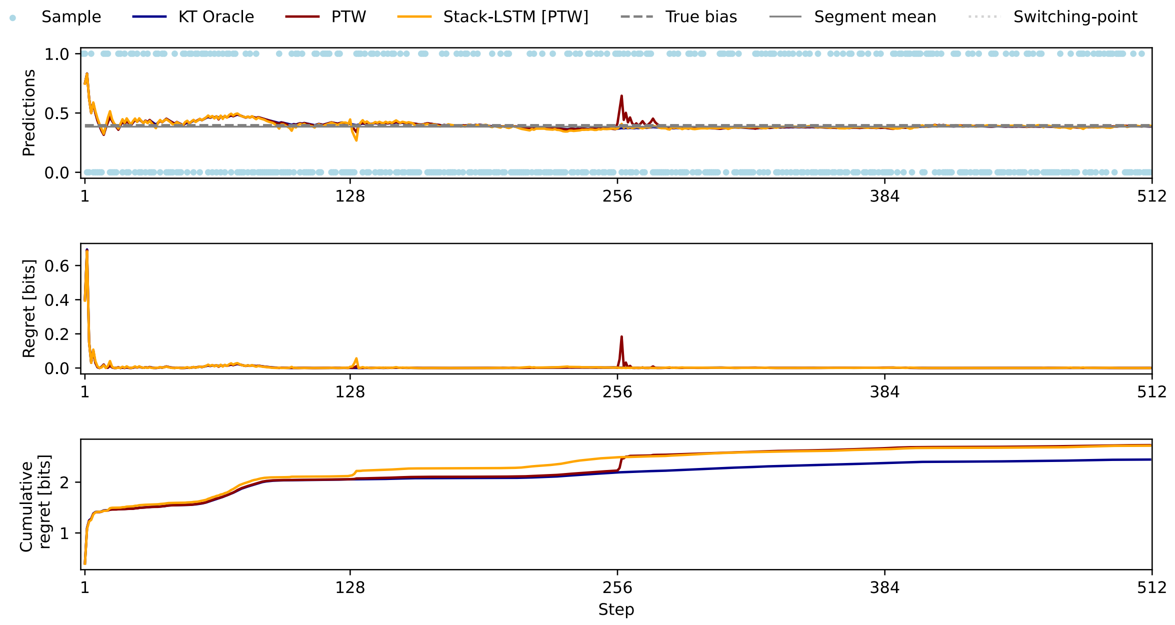


Figure 28. Same as Figure 27 but model shown here is Stack-LSTM.Compared to the plain LSTM, the Stack-LSTM seems to predict a change point at step 384 with lower probability.

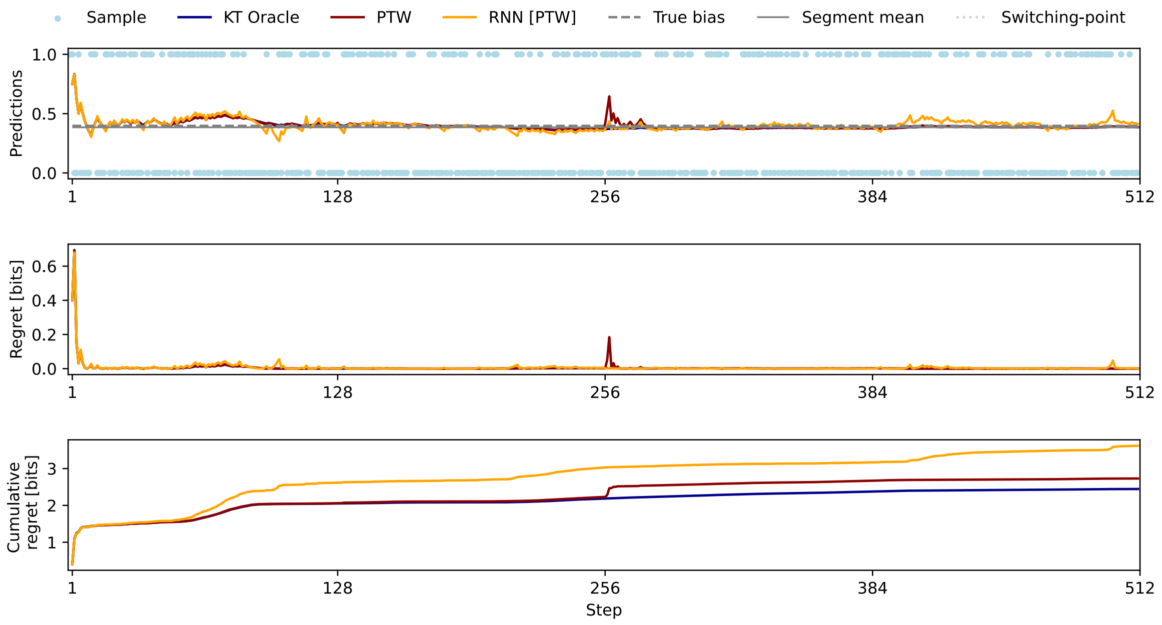


Figure 29. Same as Figure 27 but model shown here is RNN. Compared to the LSTM, the RNN predictions are a bit worse on this sequence, but internal dynamics seem to remain very stable far beyond the training range of 256 steps.

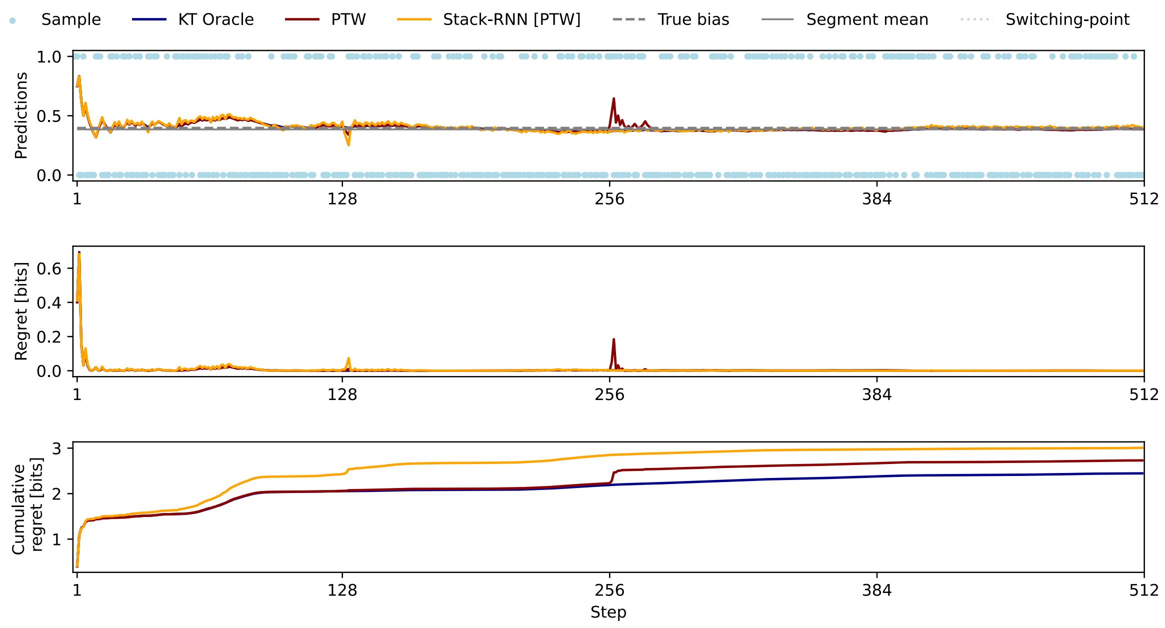


Figure 30. Same as Figure 29 but model shown here is Stack-RNN. It is hard to identify a qualitative difference to the plain RNN; the Stack-RNN performs better / more stable in the second half of the trajectory, which is in line with the trend seen for the Stack-LSTM in Figure 28 compared to the plain LSTM.

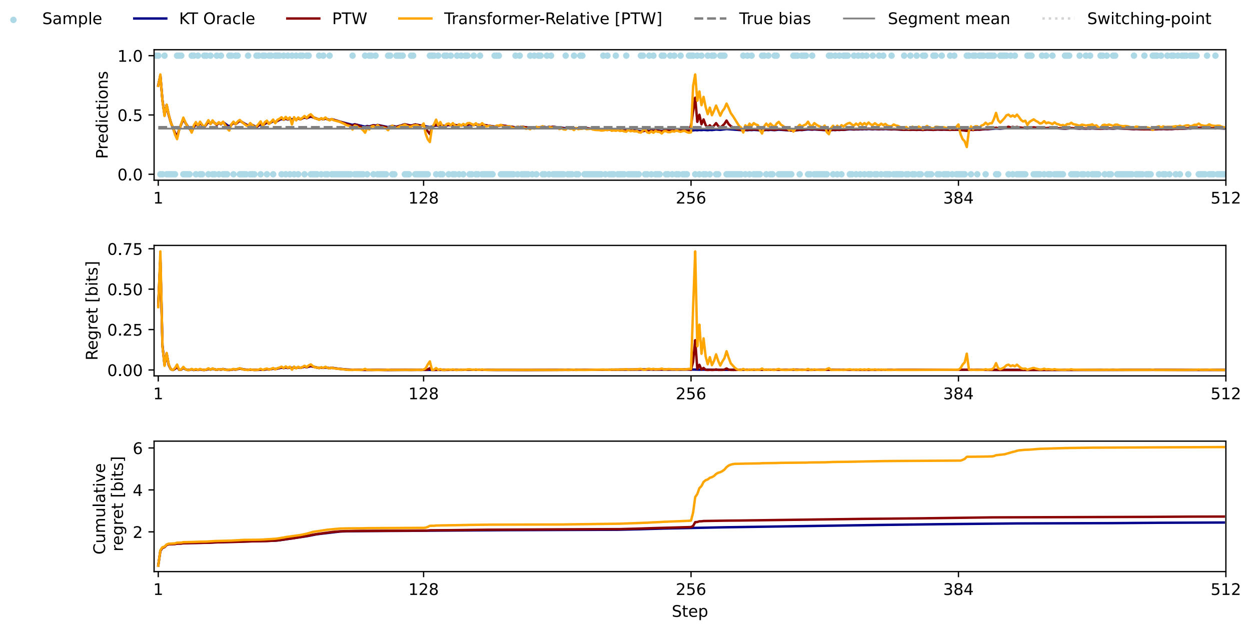


Figure 31. Same as Figure 27 but model shown here is Transformer-Relative. Compared to all other neural models, the transformer seems to struggle with predicting well from step 256 onward (note that the model was trained with sequences of length 256).

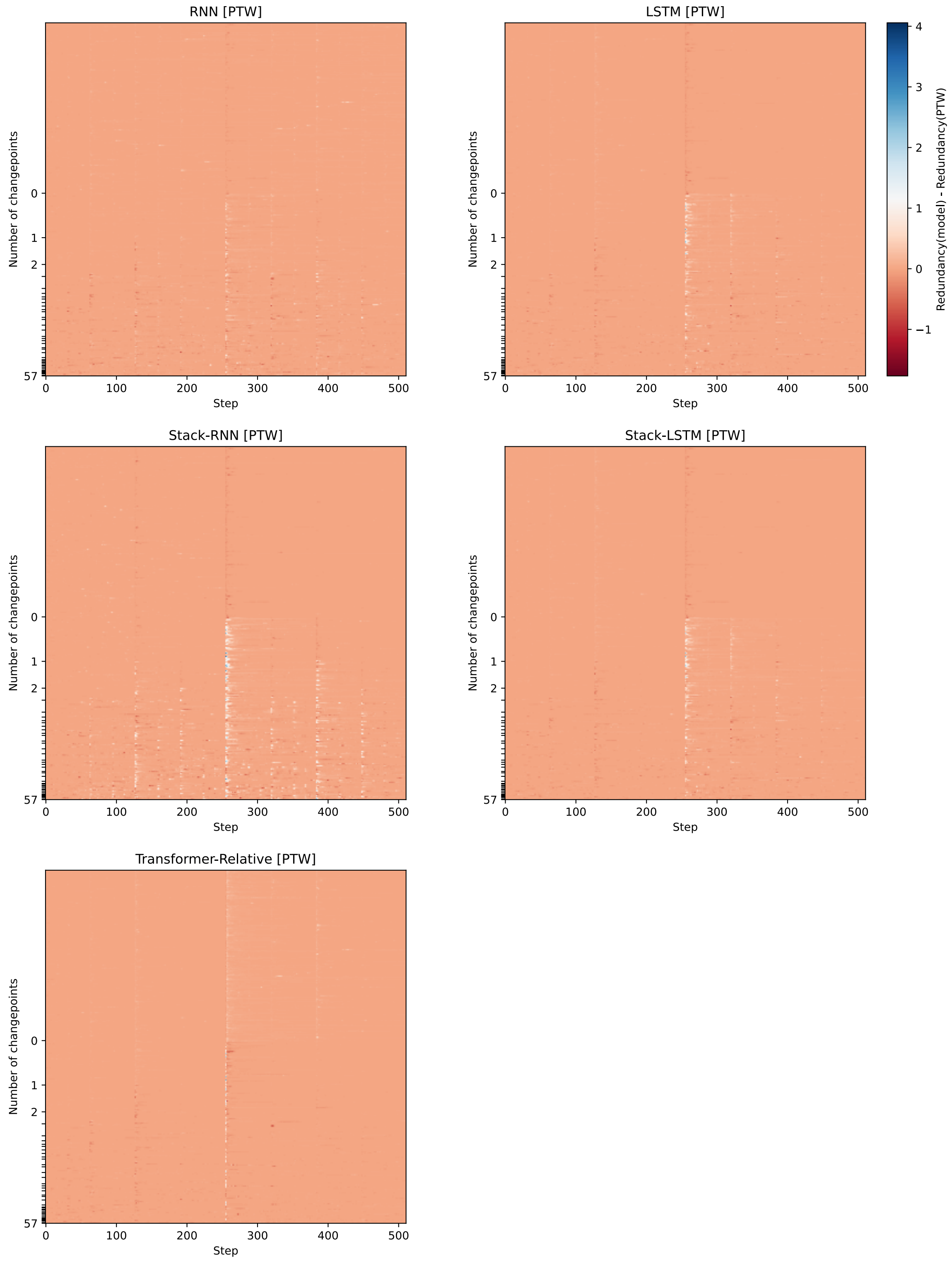


Figure 32. Models evaluated on 500 trajectories of length 512 drawn from PTW_9 prior. Models trained on sequences of length 256 drawn from PTW_8. In each panel: each row is a single trajectory, and the color encodes the difference in redundancy between the model minus PTW_9. Trajectories are ordered by the number of switching-points (y-axis). See main text for a discussion of the figure.

D. Proof of Theorem 4.2

Proof. By way of contradiction, assume $\mathbb{E}_\mu|\nu_\Theta(a_t|x_{<t}) - \xi(a_t|x_{<t})| \to 0 \, \forall \mu \in \mathcal{M}$. In particular this implies

$$\mathbb{E}_{\mu_i}|\nu_{\theta_t}(a_t) - \xi(a_t|x_{<t})| \ \longrightarrow \ 0$$

where we have used $\nu_\Theta(a_t|x_{<t}) = \nu_{\theta_t}(a_t)$. Combining this with Solomonoff's theorem (Hutter, 2005) (Thm.3.19iii)

$$\mathbb{E}_{\mu_i}|\xi(a_t|x_{<t}) - \mu_i(a_t|x_{<t})| \ \longrightarrow \ 0$$

we get

$$|\nu_{\theta_t}(a_t) - \mathbb{E}_{\mu_i}\mu_i(a_t|x_{<t})| \ \leq \ \mathbb{E}_{\mu_i}|\nu_{\theta_t}(a_t) - \mu_i(a_t|x_{<t})| \ \longrightarrow \ 0$$

The inequality exploits that ν_Θ is memoryless. Finally combining these convergences for $i = 1$ and $i = 2$ we get

$$|\mathbb{E}_{\mu_1}\mu_1(a_t|x_{<t}) - \mathbb{E}_{\mu_2}\mu_2(a_t|x_{<t})| \ \longrightarrow \ 0$$

which contradicts the theorem's assumption on μ_i. □

E. Prior Sampling Algorithms

This section provides more detail on how the temporal partitions are sampled under the PTW and LIN priors which are defined in Section 5. Both priors are hierarchical in the sense that they first define a prior on the latent switching-point structure, and then assign a $Beta(0.5, 0.5)$ prior to the Bernoulli process governing each segment. Here we focus just on the non-trivial first stage of each hierarchical process.

E.1. Sampling From the PTW Prior

Given a fixed d, Algorithm 1 samples a binary temporal partition from $\mathcal{C}_d$ distributed according to the PTW prior when invoked with an offset $o = 0$. The algorithm works by first flipping a fair coin which determines whether or not to continue splitting the current segment in half; in the case of a split, the process continues recursively on the two half segments. The base case is handled by $d = 0$ which corresponds to a segment consisting of a single time point, which obviously cannot be split further. The expected running time is proportional to the expected number of switches, which we show in Appendix B.1, Equation (8) to be equal to $\frac{d}{2} = O(\log n)$.

Algorithm 1 TPS$_d(o)$

```
Require: An offset $o \in \mathbb{N}$
  if $d = 0$ then
    return $\{(o+1, o+1)\}$
  end if
  Sample $r \sim Bernoulli(0.5)$
  if $r = 0$ then
    return $\{(o+1, o+2^d)\}$
  else
    return TPS$_{d-1}(o) \cup$ TPS$_{d-1}(o + 2^{d-1})$
  end if
```

E.2. Sampling From the LIN Prior

Algorithm 2 samples a temporal partition from $\mathcal{P}_n$ distributed according to the LIN prior. The algorithm starts in state $(1, 1)$, with the left component representing the current time, and the right component representing the time index of the current segment. The current state (t, t_c) is adapted n times, where $\frac{1/2}{t - t_c + 1}$ gives the probability of a change-point occurring at time t. The worst-case runtime complexity of this algorithm is clearly linear in n.

Algorithm 2 LIN-PRIOR-SAMPLE(n)

```
Require: Sequence length $n \in \mathbb{N}$
  $t \leftarrow 1, t_c \leftarrow 1, \mathcal{T} \leftarrow \{\}$
  while $t < n$ do
    Sample $r \sim \text{BERNOULLI}\left(\frac{1/2}{t-t_c+1}\right)$
    if $r = 1$ then
      $\mathcal{T} \leftarrow \mathcal{T} \cup \{(t_c, t)\}$
      $t_c = t + 1$
    end if
    $t \leftarrow t + 1$
  end while
  $\mathcal{T} \leftarrow \mathcal{T} \cup \{(t_c, t)\}$
  return $\mathcal{T}$
```

F. Discrete Bayesian Mixtures

A fundamental technique for constructing algorithms that work well under the logarithmic loss is Bayesian model averaging. Given a non-empty discrete set of probabilistic data generating sources $\mathcal{M} := \{\rho_1, \rho_2, \dots\}$ and a prior weight $w_0^\rho > 0$ for each $\rho \in \mathcal{M}$ such that $\sum_{\rho\in\mathcal{M}} w_0^\rho = 1$, the Bayesian mixture predictor is defined in terms of its marginal by $\xi(x_{1:n}) := \sum_{\rho\in\mathcal{M}} w_0^\rho \, \rho(x_{1:n})$. The predictive probability is thus given by the ratio of the marginals $\xi(x_n|x_{<n}) = \xi(x_{1:n})/\xi(x_{<n})$. The predictive probability can also be expressed in terms of a convex combination of conditional model predictions, with each model weighted by its posterior probability. More explicitly,

$$\begin{aligned}\xi(x_n|x_{<n}) &= \frac{\sum_{\rho\in\mathcal{M}} w_0^\rho \, \rho(x_{1:n})}{\sum_{\rho\in\mathcal{M}} w_0^\rho \, \rho(x_{<n})} \\ &= \sum_{\rho\in\mathcal{M}} w_{n-1}^\rho \, \rho(x_n|x_{<n}) \\ \text{where} \quad w_{n-1}^\rho &:= \frac{w_0^\rho \, \rho(x_{<n})}{\sum_{\nu\in\mathcal{M}} w_0^\nu \, \nu(x_{<n})}.\end{aligned}$$

A fundamental property of Bayesian mixtures is that if there exists a model $\rho^* \in \mathcal{M}$ that predicts well, then ξ will predict well since the cumulative loss satisfies

$$\begin{aligned}-\log \xi(x_{1:n}) &= -\log \sum_{\rho\in\mathcal{M}} w_0^\rho \, \rho(x_{1:n}) \\ &\le -\log w_0^{\rho^*} \rho^*(x_{1:n}) \\ &= \log\left(\frac{1}{w_0^{\rho^*}}\right) - \log \rho^*(x_{1:n}). \end{aligned} \tag{9}$$

Equation (9) implies that a constant regret bounded by $\log(1/w_0^{\rho^*})$ is suffered when using ξ in place of the best (in hindsight) model $\rho^* \in \mathcal{M}$.