Loss Bounds and Time Complexity for Speed Priors

Daniel Filan **Jan Leike** **Marcus Hutter**
Research School of Computer Science, Australian National University

Abstract

This paper establishes for the first time the predictive performance of speed priors and their computational complexity. A speed prior is essentially a probability distribution that puts low probability on strings that are not efficiently computable. We propose a variant to the original speed prior (Schmidhuber, 2002), and show that our prior can predict sequences drawn from probability measures that are estimable in polynomial time. Our speed prior is computable in doubly-exponential time, but not in polynomial time. On a polynomial time computable sequence our speed prior is computable in exponential time. We show better upper complexity bounds for Schmidhuber's speed prior under the same conditions, and that it predicts deterministic sequences that are computable in polynomial time; however, we also show that it is not computable in polynomial time, and the question of its predictive properties for stochastic sequences remains open.

1 Introduction

We consider the general problem of sequence prediction, where a sequence of symbols $x_1, x_2, \dots, x_{t-1}$ is drawn from an unknown computable distribution μ, and the task is to predict the next symbol x_t. If μ belongs to some known countable class of distributions, then a Bayesian mixture over the class leads to good loss bounds: the expected loss is at most $L + O(\sqrt{L})$ where L is the loss of the informed predictor that knows μ (Hutter, 2005, Thm. 3.48). These bounds are known to be tight.

Solomonoff's theory of inductive inference handles the most general case where all we know about μ is that it is computable (Solomonoff, 1964, 1978). The Solomonoff prior M assigns to a string x the probability that a universal Turing machine prints something starting with x when fed with fair coin flips. Equivalently, the distribution M can be seen as a Bayesian mixture that weighs each distribution according to their Kolmogorov complexity (Wood et al., 2013), assigning higher a priori probability to simpler hypotheses (Hutter, 2007). However, M is incomputable (Leike and Hutter, 2015), which has thus far limited its application.

Schmidhuber has proposed a computable alternative to M which discounts strings that are not efficiently computable (Schmidhuber, 2002). This distribution is called the *speed prior* because asymptotically only the computationally fastest distributions that explain the data contribute to the mixture. However, no loss bounds for Schmidhuber's prior, which we write as S_{Fast}, are known except in the case where the data are drawn from a prior like Schmidhuber's.

We introduce a prior S_{Kt} that is related to both S_{Fast} and M, and establish in Section 3 that it is also a speed prior in Schmidhuber's sense. Our first main contribution is a bound on the loss incurred by a S_{Kt}-based predictor when predicting strings drawn from a distribution that is computable in polynomial time. This is proved in Section 4. The bounds we get are only a logarithmic factor worse than the bounds for the Solomonoff predictor. In particular, if the measure is deterministic and the loss function penalises errors, S_{Kt}-based prediction will only make a logarithmic number of errors. Therefore, S_{Kt} is able to effectively learn the generating distribution μ. Our second main contribution is a proof that the same bound holds for the loss incurred by a S_{Fast}-based predictor when computing a string deterministically generated in polynomial time, shown in the same section.

In Section 5 we discuss the time complexity of S_{Kt} and S_{Fast}. We show that S_{Fast} is computable in exponential time while S_{Kt} is computable in doubly-exponential time, but not in polynomial time, limiting its practical applicability. However, we also show that if we are predicting a sequence that is computable in polynomial time, it only takes polynomial time to compute S_{Fast}

Appearing in Proceedings of the 19^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2016, Cadiz, Spain. JMLR: W&CP volume 41. Copyright 2016 by the authors.

and exponential time to compute S_{Kt}.

Although the results of this paper are theoretical and the algorithms impractical-seeming, related ideas from the field of algorithmic information theory have been approximated and put into practice. Examples include the Universal Similarity Metric's use in clustering (Cilibrasi and Vitanyi, 2005), Solomonoff-based reinforcement learning (Veness et al., 2011), and the Levin search-inspired Optimal Ordered Problem Solver (Schmidhuber, 2004). However, using the theory to devise practical applications is a non-trivial task that we leave for future work.

2 Preliminaries

2.1 Setup and notation

Throughout this paper, we use monotone Turing machines with a binary alphabet $\mathbb{B} = \{0, 1\}$, although all results generalise to arbitrary finite alphabets. A monotone machine is one with a unidirectional read-only input tape where the head can only move one way, a unidirectional write-only output tape where the head can only move one way, and some bidirectional work tapes. We say that a monotone machine T computes string x given program p if the machine prints x after reading all of p but no more, and write $p \xrightarrow{T} x$ (Li and Vitányi, 2008, Def. 4.5.2). Some of these machines are universal Turing machines, or 'UTM's. A UTM can simulate all other machines, so that the output of U given input $I(T)p$ (where $I(T)$ is a prefix-free coding[1] of a Turing machine T) is the same as the output of T given input p. Furthermore, we may assume this simulation occurs with only polynomial time overhead. In this paper, we fix a 'reference' UTM U, and whenever a function $f(T, \dots)$ takes an argument T that is a Turing machine, we will often write $f(\dots)$, where we set T to be the reference UTM.

Our notation is fairly standard, with a few exceptions. If $p \xrightarrow{U} x$, then we simply write $p \to x$. We write $f(n) \overset{\times}{\leq} g(n)$ if $f(n) = O(g(n))$, and $f(n) \overset{\times}{=} g(n)$ if $f(n) \overset{\times}{\leq} g(n)$ and $g(n) \overset{\times}{\leq} f(n)$. Also, if x is some string, we denote the length of x by $|x|$. We write the set of finite binary strings as $\mathbb{B}^*$, the set of infinite binary sequences as $\mathbb{B}^\infty$, an element of $\mathbb{B}^\infty$ as $x_{1:\infty}$, the n^{th} symbol of a string x or $x_{1:\infty}$ as x_n, and the first n symbols of any string x or $x_{1:\infty}$ as $x_{1:n}$. $\#A$ is the cardinality of set A. Finally, we write $x \sqsubseteq y$ if string x is a prefix of string y, and $x \sqsubset y$ if x is a proper prefix of y.

2.2 $S_{\textbf{Fast}}$ and M

To define S_{Fast}, we first need to define the FAST algorithm (called SEARCH by Li and Vitányi (2008), Ch. 7.5) after which it is named. This algorithm performs PHASE i for each $i \in \mathbb{N}$, whereby $2^{i-|p|}$ instructions of all programs satisfying $|p| \leq i$ are executed as they would be on U, and the outputs are printed sequentially, separated by blanks. If string x is computed by program p in PHASE i, then we write $p \to_i x$. Then, S_{Fast} is defined as

$$S_{\text{Fast}}(x) := \sum_{i=1}^{\infty} 2^{-i} \sum_{p \to_i x} 2^{-|p|} \tag{1}$$

This algorithm is inspired by the Kt complexity of a string, defined as

$$Kt(x) = \min_p\{|p| + \log t(U, p, x)\}$$

where $t(U, p, x)$ is the time taken for program p to compute x on the UTM U, and if program p never computes x, we set $t(U, p, x) := \infty$ (Li and Vitányi, 2008, Def. 7.5.1). If we define the Kt-cost of a computation of a string x by program p as the minimand of Kt, that is,

$$Kt\text{-cost}(p, x) := |p| + \log t(p, x)$$

then we can see that program p computes string x in PHASE i of FAST iff $Kt\text{-cost}(p, x) \leq i$. As such, S_{Fast} gives low probability to strings of high Kt complexity.

Similarly to the above, the monotone Kolmogorov complexity of x is defined as

$$Km(x) = \min_p\{|p| \mid p \to x\}$$

If we define the minimand of Km as

$$Km\text{-cost}(p, x) := \begin{cases} |p| & \text{if } p \to x \\ \infty & \text{otherwise} \end{cases}$$

then the Solomonoff prior $M(x) = \sum_{p \to x} 2^{-|p|}$ can be written as $\sum_{p \to x} 2^{-Km\text{-cost}(p,x)}$. M and S_{Fast} are both semimeasures, but not measures:

Definition 1. A *semimeasure* is a function $\nu : \mathbb{B}^* \to [0, \infty)$ such that $\nu(\epsilon) \leq 1$ and $\nu(x) \geq \nu(x0) + \nu(x1)$ for all $x \in \mathbb{B}^*$. If ν satisfies these with equality, we call ν a *measure*.

Semimeasures can be used for prediction:

Definition 2. If ν is a semimeasure, the *ν-probability of x_t given $x_{<t}$* is $\nu(x_t | x_{<t}) := \nu(x_{1:t})/\nu(x_{<t})$.

3 Speed priors

By analogy to M, we can define a variant of the Solomonoff prior that penalises strings of high Kt com-

[1] A coding such that for no two different machines T and T' is $I(T)$ a prefix of $I(T')$.

plexity more directly than S_{Fast} does:

$$S_{Kt}(x) := \sum_{p \to x} 2^{-Kt\text{-cost}(p,x)} = \sum_{p \to x} \frac{2^{-|p|}}{t(p,x)} \qquad (2)$$

S_{Kt} is a semimeasure, but is not a measure.

3.1 Similar definitions for S_{Fast} and S_{Kt}

The definitions (1) of S_{Fast} and (2) of S_{Kt} have been given in different forms—the first in terms of PHASEs of FAST, and the second in terms of Kt-cost. In this subsection, we show that each can be rewritten in a form similar to the other's definition, which sheds light on the differences and similarities between the two.

Proposition 3.

$$S_{\text{Fast}}(x) \overset{\times}{=} \sum_{p \to x} \frac{2^{-2|p|}}{t(p,x)}$$

Proof. Proof contained in supplementary material. □

Proposition 4.

$$S_{Kt}(x) \overset{\times}{=} \sum_{i=1}^{\infty} 2^{-i} \sum_{p \to_i x} 1$$

Proof. Proof contained in supplementary material. □

3.2 S_{Kt} is a speed prior

Although we have defined S_{Kt}, we have not shown any results that indicate it deserves to be called a speed prior. Two key properties of S_{Fast} justify its description as a speed prior: firstly, that the cumulative prior probability measure of all x incomputable in time t is at most inversely proportional to t, and secondly, that if $x_{1:\infty} \in \mathbb{B}^\infty$, and program $p^x \in \mathbb{B}^*$ computes $x_{1:n}$ within at most $f(n)$ steps, then the contribution to $S_{\text{Fast}}(x_{1:n})$ by programs that take time much longer than $f(n)$ vanishes as $n \to \infty$ (Schmidhuber, 2002). In this subsection, we prove that both of these properties also hold for S_{Kt}. S_{Fast} and S_{Kt} are the only distributions that the authors are aware of that satisfy these two properties.

Let $\mathcal{C}_t$ denote the set of strings x that are incomputable in time t (that is, there is no program p such that $p \to x$ in t or fewer timesteps) such that for any $y \sqsubset x$, the prefix y is computable in time t. By definition, all strings that are incomputable in time t have as a prefix an element of $\mathcal{C}_t$, and $\mathcal{C}_t$ is a prefix-free set[2] (by construction). Furthermore, the probability measure of all strings incomputable in time t is simply the sum of the probabilities of all elements of $\mathcal{C}_t$.

Proposition 5.

$$\sum_{x \in \mathcal{C}_t} S_{Kt}(x) \leq \frac{1}{t}$$

Proof.

$$\sum_{x \in \mathcal{C}_t} S_{Kt}(x) = \sum_{x \in \mathcal{C}_t} \sum_{p \to x} \frac{2^{-|p|}}{t(p,x)} \leq \frac{1}{t} \sum_{x \in \mathcal{C}_t} \sum_{p \to x} 2^{-|p|} \leq \frac{1}{t}$$

by the Kraft inequality, since the fact that $\mathcal{C}_t$ is a prefix-free set guarantees that the set of programs that compute elements of $\mathcal{C}_t$ is also prefix-free, due to our use of monotone machines. □

Proposition 6. Let $x_{1:\infty} \in \mathbb{B}^\infty$ be such that there exists a program $p^x \in \mathbb{B}^*$ which outputs $x_{1:n}$ in $f(n)$ steps for all $n \in \mathbb{N}$. Let $g(n)$ grow faster than $f(n)$, i.e. $\lim_{n\to\infty} f(n)/g(n) = 0$. Then,

$$\lim_{n\to\infty} \frac{\sum_{p \xrightarrow[\geq g(n)]{} x_{1:n}} 2^{-|p|}/t(p, x_{1:n})}{\sum_{p \xrightarrow[\leq f(n)]{} x_{1:n}} 2^{-|p|}/t(p, x_{1:n})} = 0$$

where $p \xrightarrow[\leq t]{} x$ iff program p computes string x in no more than t steps.

An informal statement of this proposition is that contributions to $S_{Kt}(x_{1:n})$ by programs that take time longer than $g(n)$ steps to run are dwarfed by those by programs that take less than $f(n)$ steps to run. Therefore, asymptotically, only the fastest programs contribute to S_{Kt}.

Proof. Proof contained in supplementary material. □

4 Loss bounds

In this section, we prove a performance bound on S_{Kt}-based sequence prediction, when predicting a sequence drawn from a measure that is estimable in polynomial time. We also prove a similar bound on S_{Fast}-based sequence prediction when predicting deterministic sequences computable in polynomial time.

For the purpose of this section, we write S_{Kt} somewhat more explicitly as

$$S_{Kt}(x) = \sum_{p \xrightarrow{U} x} \frac{2^{-|p|}}{t(U,p,x)}$$

and give some auxiliary definitions. Let $\langle\cdot\rangle_{\mathbb{B}^*}$ be a prefix-free coding of the strings of finite length and $\langle\cdot\rangle_{\mathbb{N}}$ be a prefix-free coding of the integers, where both of these prefix-free codings are computable and decodable in polynomial time.

Definition 7. A function $f : \mathbb{B}^* \to \mathbb{R}$ is *finitely computable* if there is some Turing machine T_f that

[2] That is, a set such that no element is a prefix of another element.

when given input $\langle x\rangle_{\mathbb{B}^*}$ prints $\langle m\rangle_{\mathbb{N}}\langle n\rangle_{\mathbb{N}}$ and then halts, where $f(x) = m/n$. The function f is *finitely computable in polynomial time* if it takes T_f at most $p(|x|)$ timesteps to halt on input x, where p is a polynomial.

Definition 8. Let $f, g : \mathbb{B}^* \to \mathbb{R}$. g is *estimable in polynomial time by f* if f is finitely computable in polynomial time and $f(x) \overset{\times}{=} g(x)$. The function g is *estimable in polynomial time* if it is estimable in polynomial time by some function f.

First, note that this definition is reasonably weak, since we only require $f(x) \overset{\times}{=} g(x)$, rather than $f(x) = g(x)$. Also note that if f is finitely computable in polynomial time, it is estimable in polynomial time by itself. For a measure μ, estimability in polynomial time captures our intuitive notion of efficient computability: we only need to know μ up to a constant factor for prediction, and we can find this out in polynomial time.

We consider a prediction setup where a predictor outputs a prediction, and then receives some loss depending on the predicted next bit and the correct next bit. More formally, we have some loss function $\ell(x_t, y_t) \in [0, 1]$ defined for all $x_t, y_t \in \mathbb{B}$ and all $t \in \mathbb{N}$, representing the loss incurred for a prediction of y_t when the actual next bit is x_t, which the predictor observes after prediction. One example of such a loss function is the 0-1 loss, which assigns 0 to a correct prediction and 1 to an incorrect prediction, although there are many others.

We define the Λ_ρ predictor to be the predictor which minimises ρ-expected loss, outputting $y_t^{\Lambda_\rho} := \operatorname{argmin}_{y_t} \sum_{x_t} \rho(x_t|x_{1:t-1})\ell(x_t, y_t)$ at time t. If the true distribution is μ, we judge a predictor Λ by its total μ-expected loss in the first n steps:

$$L_{n\mu}^{\Lambda} := \mathbb{E}_\mu\left[\sum_{t=1}^{n} \ell(x_t, y_t^{\Lambda})\right]$$

In particular, if we are using 0-1 loss, $L_{n\mu}^{\Lambda}$ is the expected number of errors made by Λ up to time n in the environment μ.

Theorem 9 (*Bound on S_{Kt} prediction loss*)**.** If μ is a measure that is estimable in polynomial time by some semimeasure ν, and x is a sequence sampled from μ, then the expected loss incurred by the $\Lambda_{S_{Kt}}$ predictor is bounded by

$$L_{n\mu}^{\Lambda_{S_{Kt}}} - L_{n\mu}^{\Lambda_\mu} \leq 2D_n + 2\sqrt{L_{n\mu}^{\Lambda_\mu} D_n}$$

where $D_n = O(\log n)$.[3]

Since $L_{n\mu}^{\Lambda_\mu} \leq n$, this means that $\Lambda_{S_{Kt}}$ only incurs at most $O(\sqrt{n \log n})$ extra loss in expectation, although this bound will be much tighter in more structured environments where Λ_μ makes few errors, such as deterministic environments.

In order to prove this theorem, we use the following lemma:

Lemma 10. Let ν be a semimeasure that is finitely computable in polynomial time. There exists a Turing machine T_ν such that for all $x \in \mathbb{B}^*$

$$\nu(x) = \sum_{p \xrightarrow{T_\nu} x} 2^{-|p|} \tag{3}$$

and

$$2^{-Km_{T_\nu}(x)} \geq \nu(x)/4 \tag{4}$$

where $Km_{T_\nu}(x)$ is the length of the shortest program for x on T_ν.[4]

Note that a proof already exists that there is some machine T_ν such that (3) holds (Li and Vitányi, 2008, Thm. 4.5.2), but it does not prove (4), and we wish to understand the operation of T_μ in order to prove Theorem 9.

Proof of Lemma 10. The machine T_ν is essentially a decoder of an algorithmic coding scheme with respect to ν. It uses the natural correspondence between $\mathbb{B}^\infty$ and $[0, 1]$, associating a binary string $x_1x_2x_3\cdots$ with the real number $0.x_1x_2x_3\cdots$. It determines the location of the input sequence on this line, and then assigns a certain interval for each output string, such that the width of the interval for output string x is equal to $\nu(x)$. Then, if input string p lies inside the interval for the output string x, it outputs x.

T_ν first calculates $\nu(0)$ and $\nu(1)$, and sets $[0, \nu(0))$ as the output interval for 0 and $[\nu(0), \nu(0) + \nu(1))$ as the output interval for 1. It then reads the input, bit by bit. After reading input $p_{1:n}$, it constructs the input interval $[0.p_1p_2\cdots p_n, 0.p_1p_2\cdots p_n111111\cdots)$, which represents the inerval that $0.p_1p_2\cdots p_np_{n+1}\cdots$ could lie in. It then checks if this input interval is contained in one of the output intervals. If it is, then it prints output appropriate for the interval, and if not, then it reads one more bit and repeats the process.

Suppose the first output bit is a 1. Then, T_ν calculates $\nu(10)$ and $\nu(11)$, and forms the new output intervals: $[\nu(0), \nu(0) + \nu(10))$ for outputting 0, and $[\nu(0) + \nu(10), \nu(0) + \nu(10) + \nu(11))$ for outputting 1. It then reads more input bits until the input interval lies within one of these new output intervals, and then out-

[3] A similar bound that can be proved the same way is $\sqrt{L_{n\mu}^{\Lambda_{S_{Kt}}}} - \sqrt{L_{n\mu}^{\Lambda_\mu}} \leq \sqrt{2D_n}$ for the same D_n (Hutter, 2007, Eq. 8, 5).

[4] Note that this lemma would be false if we were to let ν be an arbitrary lower-semicomputable semimeasure, since if $\nu = M$, this would imply that $2^{-Km(x)} \overset{\times}{=} M(x)$, which was disproved by Gács (1983).

puts the appropriate bit. The computation proceeds in this fashion.

Equation (3) is satisfied, because $\sum_{p \xrightarrow{T_\nu} x} 2^{-|p|}$ is just the total length of all possible input intervals that fit inside the output interval for x, which by construction is $\nu(x)$.

To show that (4) is satisfied, note that $2^{-Km_{T_\nu}(x)}$ is the length of the largest input interval for x. Now, input intervals are binary intervals (that is, their start points and end points have a finite binary expansion), and for every interval I, there is some binary interval contained in I with length $\geq 1/4$ that of I. Therefore, the output interval for x contains some input interval with length at least 1/4 that of the length of the output interval. Since the length of the output interval for x is just $\nu(x)$, we can conclude that $2^{-Km_{T_\nu}(x)} \geq \nu(x)/4$. □

Proof of Theorem 9. Using Lemma 10, we show a bound on S_{Kt} that bounds its KL divergence with μ. We then apply the unit loss bound (Hutter, 2005, Thm. 3.48) (originally shown for the Solomonoff prior, but valid for any prior) to show the desired result.

First, we reason about the running time of the shortest program that prints x on the machine T_ν (defined in Lemma 10). Since we would only calculate $\nu(y0)$ and $\nu(y1)$ for $y \sqsubseteq x$, this amounts to $2|x|$ calculations. Each calculation need only take polynomial time in the length of its argument, because T_ν could just simulate the machine that takes input x and returns the numerator and denominator of x, prefix-free coded, and it only takes polynomial time to undo this prefix-free coding. Therefore, the calculations take at most $2|x|f(|x|) =: g(|x|)$, where f is a polynomial. We also, however, need to read all the bits of the input, construct the input intervals, and compare them to the output intervals. This takes time linear in the number of bits read, and for the shortest program that prints x, this number of bits is (by definition) $Km_{T_\nu}(x)$. Since $2^{-Km_{T_\nu}(x)} \overset{\times}{=} \nu(x)$, $Km_{T_\nu}(x) \leq -\log(\nu(x)) + O(1)$, and since $\nu(x) \overset{\times}{=} \mu(x)$, $-\log(\nu(x)) \leq -\log(\mu(x)) + O(1)$. Therefore, the total time taken is bounded above by $g(|x|) - O(1)\log(\mu(x))$, where we absorb the additive constants into $g(|x|)$.

This out of the way, we can calculate

$$\begin{aligned}
S_{Kt}(x) &= \sum_{p \xrightarrow{U} x} \frac{2^{-|p|}}{t(U,p,x)} \\
&= \sum_{\text{Turing machines } T} 2^{-|I(T)|} \sum_{q \xrightarrow{T} x} \frac{2^{-|q|}}{t(T,q,x)^{O(1)}} \\
&\overset{\times}{\geq} \sum_{p \xrightarrow{T_\nu} x} \frac{2^{-|p|}}{t(T_\nu,p,x)^{O(1)}} \\
&\geq \frac{2^{-Km_{T_\nu}(x)}}{(g(|x|) - O(1)\log(\mu(x)))^{O(1)}} \\
&\overset{\times}{\geq} \frac{\mu(x)}{(g(|x|) - O(1)\log(\mu(x)))^{O(1)}} \qquad (5)
\end{aligned}$$

Now, the unit loss bound tells us that

$$L_{n\mu}^{\Lambda_{S_{Kt}}} - L_{n\mu}^{\Lambda_\mu} \leq 2D_n(\mu||S_{Kt}) + 2\sqrt{L_{n\mu}^{\Lambda_\mu} D_n(\mu||S_{Kt})} \quad (6)$$

where $D_n(\mu||S_{Kt}) := \mathbb{E}_\mu\left[\ln\left(\mu(x_{1:n})/S_{Kt}(x_{1:n})\right)\right]$ is the relative entropy. We can calculate $D_n(\mu||S_{Kt})$ using equation (5):

$$\begin{aligned}
D_n(\mu||S_{Kt}) &= \mathbb{E}_\mu\left[\ln\frac{\mu(x_{1:n})}{S_{Kt}(x_{1:n})}\right] \\
&\overset{\times}{\leq} \mathbb{E}_\mu\left[\ln\left((g(n) - O(1)\log(\mu(x_{1:n})))^{O(1)}\right)\right] \\
&\overset{\times}{\leq} \mathbb{E}_\mu\left[\ln(g(n) - O(1)\log(\mu(x_{1:n})))\right] \\
&\leq \ln\mathbb{E}_\mu\left[g(n) - O(1)\log(\mu(x_{1:n}))\right] \qquad (7) \\
&= \ln\left(g(n) + O(1)H_\mu(x_{1:n})\right)
\end{aligned}$$

where $H_\mu(x_{1:n})$ denotes the binary entropy of the random variable $x_{1:n}$ with respect to μ

$$\leq \ln\left(g(n) + O(n)\right) \overset{\times}{=} \log n \qquad (8)$$

where (7) comes from Jensen's inequality. Equations (6) and (8) together prove the theorem. □

We therefore have a loss bound on the S_{Kt}-based sequence predictor in environments that are estimable in polynomial time by a semimeasure. Furthermore:

Corollary 11.

$$L_{n\mu}^{\Lambda_{S_{Kt}}} \leq 2D_n(\mu||S_{Kt}) \overset{\times}{=} \log n$$

for deterministic measures[5] μ computable in polynomial time, if correct predictions incur no loss.

We should note that this method fails to prove similar bounds for S_{Fast}, since we instead get

$$S_{\text{Fast}}(x) \overset{\times}{=} \sum_{p \xrightarrow{U} x} \frac{2^{-2|p|}}{t(U,p,x)} \overset{\times}{\geq} \frac{\mu(x)^2}{(|x|^{O(1)} - \log\mu(x))^{O(1)}} \qquad (9)$$

which gives us

$$\begin{aligned}
D_n(\mu||S_{\text{Fast}}) &= \mathbb{E}_\mu\left[\ln\frac{\mu(x_{1:n})}{S_{\text{Fast}}(x_{1:n})}\right] \\
&\leq O(\log n) + H_\mu(x_{1:n})
\end{aligned}$$

Since $H_\mu(x_{1:n})$ can grow linearly in n (for example, take μ to be $\lambda(x) = 2^{-|x|}$, the uniform measure), this can only prove a trivial linear loss bound without restrictions on the measure μ. It is also worth explicitly noting that the constants hidden in the $O(\cdot)$ notation depend on the environment μ, as will be the case for the rest of this paper.

[5] That is, measures that give probability 1 to prefixes of one particular infinite sequence.

One important application of Theorem 9 is to the 0-1 loss function. Then, it states that a predictor that outputs the most likely successor bit according to S_{Kt} only makes logarithmically many errors in a deterministic environment computable in polynomial time. In other words, S_{Kt} quickly learns the sequence it is predicting, making very few errors.

Next, we show that S_{Fast} makes only logarithmically many errors on a sequence deteriministically computed in polynomial time. This follows from a rather simple argument.

Theorem 12 (*Bound on S_{Fast} prediction loss*)**.** Let μ be a deterministic environment and $x_{1:\infty}$ be the sequence whose prefixes μ assigns probability 1 to. If $x_{1:\infty}$ is computable in polynomial time by a program p^x, then S_{Fast} only incurrs logarithmic loss, if correct predictions incur no loss.

Proof. Using the unit loss bound,

$$\begin{aligned} L_{n\mu}^{\Lambda_{S_{\text{Fast}}}} &= L_{n\mu}^{\Lambda_{S_{\text{Fast}}}} - L_{n\mu}^{\Lambda_\mu} \\ &\leq 2D_n(\mu \| S_{\text{Fast}}) \\ &= -2\ln S_{\text{Fast}}(x_{1:n}) \\ &\overset{\times}{\leq} 2|p^x| + \log t(p^x, x_{1:n}) \\ &\overset{\times}{=} \log n \end{aligned}$$

□

5 Time complexity

Although it has been proved that S_{Fast} is computable (Schmidhuber, 2002), no bounds are given for its computational complexity. Given that the major advantage of S_{Fast}-based prediction over M-based prediction is its computability, it is of interest to determine the time required to compute S_{Fast}, and whether such a computation is feasible or not. The same questions apply to S_{Kt}, to a greater extent because we have not even yet shown that S_{Kt} is computable.

In this section, we show that an arbitrarily good approximation to $S_{\text{Fast}}(x)$ is computable in time exponential in $|x|$, and an arbitrarily good approximation to $S_{Kt}(x)$ is computable in time doubly-exponential in $|x|$. We do this by explicitly constructing algorithms that perform PHASEs of FAST until enough contributions to S_{Fast} or S_{Kt} are found to constitute a sufficient proportion of the total.

We also show that no such approximation of S_{Kt} or S_{Fast} can be computed in polynomial time. We do this by contradiction: showing that if it were possible to do so, we would be able to construct an 'adversarial' sequence that was computable in polynomial time, yet could not be predicted by our approximation; a contradiction.

Finally, we investigate the time taken to compute S_{Kt} and S_{Fast} along a polynomial-time computable sequence $x_{1:\infty}$. If we wanted to predict the most likely continuation of $x_{1:n}$ according to $S \in \{S_{Kt}, S_{\text{Fast}}\}$, we would have to compute an approximation to $S(x_{1:n}0)$ and $S(x_{1:n}1)$, to see which one was greater. We show that it is possible to compute these approximations in polynomial time for S_{Fast} and in exponential time for S_{Kt}: an exponential improvement over the worst-case bounds in both cases.

5.1 Upper bounds

Theorem 13 (*S_{Fast} computable in exponential time*)**.** For any $\varepsilon > 0$, there exists an approximation $S_{\text{Fast}}^\varepsilon$ of S_{Fast} such that $|S_{\text{Fast}}^\varepsilon / S_{\text{Fast}} - 1| \leq \varepsilon$ and $S_{\text{Fast}}^\varepsilon(x)$ is computable in time exponential in $|x|$.

Proof. First, we note that in PHASE i of FAST, we try out $2^1 + \cdots + 2^i = 2^{i+1}$ program prefixes p, and each prefix p gets $2^{i-|p|}$ steps. Therefore, the total number of steps in PHASE i is $2^1 \times 2^{i-1} + 2^2 \times 2^{i-2} + \cdots + 2^i \times 2^{i-i} = i2^i$, and the total number of steps in the first k PHASEs is

$$\#\text{ steps} = \sum_{i=1}^{k} i2^i = 2^{k+1}(k-1) + 2 \tag{10}$$

Now, suppose we want to compute a sufficient approximation $S_{\text{Fast}}^\varepsilon(x)$. If we compute k phases of FAST and then add up all the contributions to $S_{\text{Fast}}(x)$ found in those phases, the remaining contributions must add up to $\leq \sum_{i=k+1}^{\infty} 2^{-i} = 2^{-k}$. In order for the contributions we have added up to contribute $\geq 1 - \varepsilon$ of the total, it suffices to use k such that

$$k = \lfloor -\log(\varepsilon S_{\text{Fast}}(x)) + 1 \rfloor \tag{11}$$

Now, since the uniform measure $\lambda(x) = 2^{-|x|}$ is finitely computable in polynomial time, it is estimable in polynomial time by itself, so we can substitute λ into equation (9) to obtain

$$S_{\text{Fast}}(x) \overset{\times}{\geq} \frac{2^{-2|x|}}{(|x|^{O(1)} + \log(2^{|x|}))^{O(1)}} = \frac{1}{|x|^{O(1)} 2^{2|x|}} \tag{12}$$

Substituting equation (12) into equation (11), we get

$$\begin{aligned} k &\leq \log\left(O(2^{2|x|}|x|^{O(1)})/\varepsilon\right) + 1 \\ &= -\log\varepsilon + 2|x| + O(\log|x|) \end{aligned} \tag{13}$$

So, substituting equation (13) into equation (10),

$$\begin{aligned} \#\text{ steps} &\leq 2^{-\log\varepsilon + 2|x| + O(\log|x|) + 1} \\ &\quad \times (-\log\varepsilon + 2|x| + O(\log|x|) - 1) + 2 \\ &= \frac{1}{\varepsilon} 2^{2|x|} |x|^{O(1)} (-\log\varepsilon + 2|x| + O(\log|x|)) \end{aligned}$$

$$\leq 2^{O(|x|)}$$

Therefore, $S^{\varepsilon}_{\text{Fast}}$ is computable in exponential time. □

Theorem 14 (*S_{Kt} computable in doubly-exponential time*)**.** For any $\varepsilon > 0$, there exists an approximation S^{ε}_{Kt} of S_{Kt} such that $|S^{\varepsilon}_{Kt}/S_{Kt} - 1| \leq \varepsilon$ and S^{ε}_{Kt} is computable in time doubly-exponential in $|x|$.

Proof. Proof contained in supplementary material. □

5.2 Lower bounds

Theorem 15 (*S_{Kt} not computable in polynomial time*)**.** For no $\varepsilon > 0$ does there exist an approximation S^{ε}_{Kt} of S_{Kt} such that $|S^{\varepsilon}_{Kt}/S_{Kt} - 1| \leq \varepsilon$ and S^{ε}_{Kt} is computable in time polynomial in $|x|$.

The proof of this theorem relies on the following lemma:

Lemma 16. If S^{ε}_{Kt} is an approximation of S_{Kt} as given in Theorem 15, then the bound of Theorem 9 applies to S^{ε}_{Kt}. That is,

$$L^{\Lambda_{S^{\varepsilon}_{Kt}}}_{n\mu} - L^{\Lambda_{\mu}}_{n\mu} \leq 2D_n + 2\sqrt{L^{\Lambda_{\mu}}_{n\mu} D_n}$$

where $D_n = O(\log n)$.

Proof of Lemma 16. From the definition of S^{ε}_{Kt}, it is clear that $S^{\varepsilon}_{Kt} \geq (1-\varepsilon) S_{Kt}$. Then,

$$\begin{aligned} D_n(\mu || S^{\varepsilon}_{Kt}) &:= \mathbb{E}_{\mu}\left[\ln \frac{\mu(x_{1:n})}{S^{\varepsilon}_{Kt}(x_{1:n})}\right] \\ &\leq \mathbb{E}_{\mu}\left[\ln \frac{\mu(x_{1:n})}{S_{Kt}(x_{1:n})}\right] - \ln(1-\varepsilon) \\ &\overset{\times}{=} \log n \end{aligned}$$

for μ estimable in polynomial time by a semimeasure, where we use Theorem 9 for the final 'equality'. Therefore, the bound of Theorem 9 applies. □

Proof of Theorem 15. Suppose by way of contradiction that S^{ε}_{Kt} were computable in polynomial time. Then, the sequence $x_{1:\infty}$ would also be computable in polynomial time, where

$$x_n = \begin{cases} 1 & \text{if } S^{\varepsilon}_{Kt}(0|x_{1:n-1}) \geq S^{\varepsilon}_{Kt}(1|x_{1:n-1}) \\ 0 & \text{if } S^{\varepsilon}_{Kt}(0|x_{1:n-1}) < S^{\varepsilon}_{Kt}(1|x_{1:n-1}) \end{cases}$$

$x_{1:\infty}$ is therefore an adversarial sequence against S^{ε}_{Kt}: it predicts whichever symbol S^{ε}_{Kt} thinks less likely, and breaks ties with 1.

Now, consider an S^{ε}_{Kt}-based predictor $\Lambda_{S^{\varepsilon}_{Kt}}$ that minimises 0-1 loss—that is, one that predicts the more likely continuation according to S^{ε}_{Kt}. Further, suppose this predictor breaks ties with 0. Since the loss bound of Theorem 9 applies independently of tie-breaking method, Lemma 16 tells us that $\Lambda_{S^{\varepsilon}_{Kt}}$ must make only logarithmically many errors when predicting $x_{1:\infty}$. However, by design, $\Lambda_{S^{\varepsilon}_{Kt}}$ errs every time when predicting $x_{1:\infty}$. This is a contradiction, showing that S^{ε}_{Kt} cannot be computable in polynomial time. □

Next, we provide a proof of the analogous theorem for Schmidhuber's speed prior S_{Fast}, using a lemma about the rate at which S_{Fast} learns polynomial-time computable deterministic sequences.

Theorem 17 (*S_{Fast} not computable in polynomial time*)**.** For no $\varepsilon > 0$ does there exist an approximation $S^{\varepsilon}_{\text{Fast}}$ of S_{Fast} such that $|S^{\varepsilon}_{\text{Fast}}/S_{\text{Fast}} - 1| \leq \varepsilon$ and $S^{\varepsilon}_{\text{Fast}}(x)$ is computable in time polynomial in $|x|$.

Lemma 18. For a sequence $x_{1:\infty}$ computed in polynomial time by some program p^x,

$$\sum_{t=1}^{n} |1 - S_{\text{Fast}}(x_t \mid x_{<t})| \overset{\times}{\leq} \log n$$

Proof of Lemma 18. We calculate

$$\begin{aligned} &\sum_{t=1}^{n} |1 - S_{\text{Fast}}(x_t \mid x_{<t})| \\ &\leq -\sum_{t=1}^{n} \ln S_{\text{Fast}}(x_t \mid x_{<t}) \\ &= -\ln \prod_{t=1}^{n} S_{\text{Fast}}(x_t \mid x_{<t}) \\ &= -\ln S_{\text{Fast}}(x_{1:n}) \\ &\overset{\times}{\leq} 2|p^x| + \log t(p^x, x_{1:n}) \\ &\overset{\times}{\leq} \log n \end{aligned}$$

□

Proof of Theorem 17. Let $S^{\varepsilon}_{\text{Fast}}$ be computable in polynomial time, and construct the adversarial sequence $x_{1:\infty}$ against $S^{\varepsilon}_{\text{Fast}}$ in the same manner as in the proof of Theorem 15. Then, $x_{1:\infty}$ would be a deterministic sequence computable in polynomial time, and so by Lemma 18,

$$\begin{aligned} \log n &\overset{\times}{\geq} \sum_{t=1}^{n} |1 - S_{\text{Fast}}(x_t \mid x_{<t})| \\ &\geq \sum_{t=1}^{n} |1 - S^{\varepsilon}_{\text{Fast}}(x_t \mid x_{<t})| - \varepsilon n \\ &\geq \left(\frac{1}{2} - \varepsilon\right) n \end{aligned}$$

a contradiction. Therefore, $S^{\varepsilon}_{\text{Fast}}$ cannot be computable in polynomial time. □

Note the similarity between the speed priors and M: all succeed at predicting sequences in a certain computability class, and therefore none are in that class.

5.3 Computability along polynomial time computable sequences

Theorem 19 (*S_{Fast} computable in polynomial time on polynomial time computable sequence*)**.** If $x_{1:\infty}$ is computable in polynomial time, then $S^{\varepsilon}_{\text{Fast}}(x_{1:n}0)$ and $S^{\varepsilon}_{\text{Fast}}(x_{1:n}1)$ are also computable in polynomial time.

Proof. Proof contained in supplementary material. □

Theorem 20 (*S_{Kt} computable in exponential time on polynomial time computable sequence*)**.** If $x_{1:\infty}$ is computable in polynomial time, then $S^{\varepsilon}_{Kt}(x_{1:n}0)$ and $S^{\varepsilon}_{Kt}(x_{1:n}1)$ are computable in time $2^{n^{O(1)}}$.

Proof. Proof contained in supplementary material. □

Note that Theorem 19 does not contradict Theorem 17, which merely states that there exists a sequence for which S_{Fast} is not computable in polynomial time, and does not assert that S_{Fast} must be computable in superpolynomial time for every sequence.

6 Discussion

In this paper, we have shown for the first time a loss bound on prediction based on a speed prior. This was proved for S_{Kt}, and we suspect that the result for stochastic sequences is not true for S_{Fast}, due to weaker bounds on its KL divergence with the true environment. However, in the special case of deterministic sequences, we show that S_{Fast} has the same performance as S_{Kt}. We have also, again for the first time, investigated the efficiency of computing speed priors. This offers both encouraging and discouraging news: S_{Kt} is good at prediction in certain environments, but is not efficiently computable, even in the restricted class of environments where it succeeds at prediction. On the other hand, S_{Fast} is efficiently computable for certain inputs, and succeeds at predicting those sequences, but we have no evidence that it succeeds at prediction in the more general case of stochastic sequences.

To illustrate the appeal of speed-prior based inference, it is useful to contrast with a similar approach introduced by Vovk (1989). This approach aims to predict certain simple measures: if α and γ are functions $\mathbb{N} \to \mathbb{N}$, then a measure ν is said to be (α, γ)-simple if there exists some 'program' $\pi^{\nu} \in \mathbb{B}^{\infty}$ such that the UTM with input x outputs $\nu(x)$ in time $\leq \gamma(|x|)$ by reading only $\alpha(|x|)$ bits of π^{ν}. Vovk proves that if α is logarithmic and γ is polynomial, and if both α and γ are computable in polynomial time, then there exists a measure $\mu_{\alpha,\gamma}$ which is computable in polynomial time that predicts sequences drawn from (α, γ)-simple measures. S_{Kt} and $\mu_{\alpha,\gamma}$ are similar in spirit, in that they predict measures that are easy to compute. However, the contrast between the two is instructive: $\mu_{\alpha,\gamma}$ requires one to fix α and γ in advance, and only succeeds on (α, γ)-simple measures. Therefore, there are many polynomials $\gamma' > \gamma$ such that $\mu_{\alpha,\gamma}$ cannot predict (α, γ')-simple measures. We are therefore required to make an arbitrary choice of parameters at the start and are limited by that choice of parameters. In contrast, S_{Kt} predicts all measures estimable in polynomial time, and does not require some polynomial to be fixed beforehand. S_{Kt}-based prediction therefore is more general than that of $\mu_{\alpha,\gamma}$.

Further questions remain to be studied. In particular, we do not know whether the loss bounds on speed-prior-based predictors can be improved. We also do not know how to tighten the gap between the lower and upper complexity bounds on the speed priors.

It would also be interesting to generalise the definition of S_{Kt}. Our performance result was due to the fact that for all measures μ estimable in polynomial time, $S_{Kt}(x) \geq \mu(x)/(f(|x|, -\log \mu(x)))$, where f was a polynomial. Now, if μ is estimable in polynomial time by ν, then the denominator of the fraction $\nu(x)$ must be small enough to be printed in polynomial time. This gives an exponential bound on $1/\nu(x)$, and therefore a polynomial bound on $-\log \mu(x)$. We therefore have that $S_{Kt}(x) \geq \mu(x)/g(|x|)$ for a polynomial g. Because g is subexponential, this guarantees that S_{Kt} converges to μ (Ryabko and Hutter, 2008).[6] This suggests a generalisation of S_{Kt} that takes a mixture over some class of measures, each measure discounted by its computation time. Loss bounds can be shown in the same manner as in this paper if the measures are computable in polynomial time, but the question of the computational complexity of this mixture remains completely open.

Acknowledgements

The authors would like to thank the reviewers for this paper, the Machine Intelligence Research Institute for funding a workshop on Schmidhuber's speed prior, which introduced the first author to the concept, and Mayank Daswani and Tom Everitt for valuable discussions of the material. This work was in part supported by ARC grant DP150104590.

[6] To see that g must be subexponential for good predictive results, note that for all measures μ, $\lambda(x) \geq \mu(x)/2^{|x|}$, but λ does not predict well.

References

Rudi Cilibrasi and Paul Vitanyi. Clustering by compression. *IEEE Transactions on Information Theory*, 51(4):1523–1545, 2005.

Péter Gács. On the relation between descriptional complexity and algorithmic probability. *Theoretical Computer Science*, 22(12):71 – 93, 1983.

Marcus Hutter. *Universal Artificial Intelligence: Sequential Decisions Based on Algorithmic Probability*. Springer Science & Business Media, 2005.

Marcus Hutter. On universal prediction and Bayesian confirmation. *Theoretical Computer Science*, 384(1): 33–48, 2007.

Jan Leike and Marcus Hutter. On the computability of Solomonoff induction and knowledge-seeking. In *Algorithmic Learning Theory*, pages 364–378. Springer, 2015.

Ming Li and Paul M.B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer Science & Business Media, 2008.

Daniil Ryabko and Marcus Hutter. Predicting nonstationary processes. *Applied Mathematics Letters*, 21(5):477–482, 2008.

Jürgen Schmidhuber. The speed prior: a new simplicity measure yielding near-optimal computable predictions. In *Computational Learning Theory*, pages 216–228. Springer, 2002.

Jürgen Schmidhuber. Optimal ordered problem solver. *Machine Learning*, 54(3):211–254, 2004.

Ray J. Solomonoff. A formal theory of inductive inference. part I. *Information and Control*, 7(1):1–22, 1964.

Ray J. Solomonoff. Complexity-based induction systems: comparisons and convergence theorems. *IEEE Transactions on Information Theory*, 24(4):422–432, 1978.

Joel Veness, Kee Siong Ng, Marcus Hutter, William Uther, and David Silver. A Monte-Carlo AIXI approximation. *Journal of Artificial Intelligence Research*, 40(1):95–142, 2011.

Vladimir G. Vovk. Prediction of stochastic sequences. *Problemy Peredachi Informatsii*, 25(4):35–49, 1989.

Ian Wood, Peter Sunehag, and Marcus Hutter. (Non-)equivalence of universal priors. In *Algorithmic Probability and Friends. Bayesian Prediction and Artificial Intelligence*, pages 417–425. Springer, 2013.