Skip Context Tree Switching

Marc G. Bellemare BELLEMARE@GOOGLE.COM
Joel Veness VENESS@GOOGLE.COM
Google DeepMind

Erik Talvitie ERIK.TALVITIE@FANDM.EDU
Franklin and Marshall College

Abstract

Context Tree Weighting is a powerful probabilistic sequence prediction technique that efficiently performs Bayesian model averaging over the class of all prediction suffix trees of bounded depth. In this paper we show how to generalize this technique to the class of K-skip prediction suffix trees. Contrary to regular prediction suffix trees, K-skip prediction suffix trees are permitted to ignore up to K contiguous portions of the context. This allows for significant improvements in predictive accuracy when irrelevant variables are present, a case which often occurs within record-aligned data and images. We provide a regret-based analysis of our approach, and empirically evaluate it on the Calgary corpus and a set of Atari 2600 screen prediction tasks.

1. Introduction

The sequential prediction setting, in which an unknown environment generates a stream of observations which an algorithm must probabilistically predict, is highly relevant to a number of machine learning problems such as statistical language modelling, data compression, and model-based reinforcement learning. A powerful algorithm for this setting is Context Tree Weighting (CTW, Willems et al., 1995), which efficiently performs Bayesian model averaging over a class of prediction suffix trees (Ron et al., 1996). In a compression setting, Context Tree Weighting is known to be an asymptotically optimal coding distribution for D-Markov sources.

A significant practical limitation of CTW stems from the fact that model averaging is only performed over prediction suffix trees whose ordering of context variables is fixed in advance. As we discuss in Section 3, reordering these variables can lead to significant performance improvements given limited data. This idea was leveraged by the class III algorithm of Willems et al. (1996), which performs Bayesian model averaging over the collection of prediction suffix trees defined over all possible fixed variable orderings. Unfortunately, the $O(2^D)$ computational requirements of the class III algorithm prohibit its use in most practical applications.

Our main contribution is the Skip Context Tree Switching (SkipCTS) algorithm, a polynomial-time compromise between the linear-time CTW and the exponential-time class III algorithm. We introduce a family of nested model classes, the Skip Context Tree classes, which form the basis of our approach. The K^{th} order member of this family corresponds to prediction suffix trees which may skip up to K runs of contiguous variables. The usual model class associated with CTW is a special case, and corresponds to $K = 0$. In many cases of interest, SkipCTS's $O(D^{2K+1})$ running time is practical and provides significant performance gains compared to Context Tree Weighting.

SkipCTS is best suited to sequential prediction problems where a good fixed variable ordering is unknown *a priori*. As a simple example, consider the record aligned data depicted by Figure 1. SkipCTS with $K = 1$ can improve on the CTW ordering by skipping the five most recent symbols and directly learning the lexicographical relation.

While Context Tree Weighting has traditionally been used as a data compression algorithm, it has proven useful in a diverse range of sequential prediction settings. For example, Veness et al. (2011) proposed an extension (FAC-CTW) for Bayesian, model-based reinforcement learning in structured, partially observable domains. Bellemare et al. (2013b) used FAC-CTW as a base model in their Quad-Tree Factorization algorithm, which they applied to the problem of predicting high-dimensional video game screen images. Our empirical results on the same video game domains (Section 4.2) suggest that SkipCTS is par-

Proceedings of the 31^{st} International Conference on Machine Learning, Beijing, China, 2014. JMLR: W&CP volume 32.

Figure 1. A sequence of lexicographically sorted fixed-length strings, which is particularly well-modelled by SkipCTS.

ticularly beneficial in this more complex prediction setting.

2. Background

We consider the problem of probabilistically predicting the output of an unknown sequential data generating source. Given a finite alphabet $\mathcal{X}$, we write $x_{1:n} := x_1 x_2 \dots x_n \in \mathcal{X}^n$ to denote a string of length n, xy to denote the concatenation of two strings x and y, and x^i to denote the concatenation of i copies of x. We further denote $x_{<n} := x_{1:n-1}$ and the empty string by ϵ. Given an arbitrary finite length string y, we denote its length by $|y|$. The space of probability distributions over a finite alphabet $\mathcal{X}$ is denoted by $\mathscr{P}(\mathcal{X})$. A sequential probabilistic model ρ is defined by a sequence of probability mass functions $\{\rho_i \in \mathscr{P}(\mathcal{X}^i)\}_{i\in\mathbb{N}}$ that satisfy, for any $n \in \mathbb{N}$, for any string $x_{1:n} \in \mathcal{X}^n$, the constraint $\sum_{x_n\in\mathcal{X}} \rho_n(x_{1:n}) = \rho_{n-1}(x_{<n})$. Since the subscript to ρ_n is always clear from its argument, we henceforth write $\rho(x_{1:n})$ for the probability assigned to $x_{1:n}$ by ρ. We use $\rho(x_n \,|\, x_{<n})$ to denote the probability of x_n conditional on $x_{<n}$, defined as $\rho(x_n \,|\, x_{<n}) := \rho(x_{1:n})/\rho(x_{<n})$ provided $\rho(x_{<n}) > 0$, from which the chain rule $\rho(x_{1:n}) = \prod_{i=1}^n \rho(x_i \,|\, x_{<i})$ follows.

We assess the quality of a model's predictions through its cumulative, instantaneous logarithmic loss $\sum_{i=1}^n -\log \rho(x_i \,|\, x_{<i}) = -\log \rho(x_{1:n})$. Given a set of models $\mathcal{M}$, we define the regret of ρ with respect to $\mathcal{M}$ as

$$\mathcal{R}_n(\rho, \mathcal{M}) := -\log \rho(x_{1:n}) - \min_{\nu\in\mathcal{M}} -\log \nu(x_{1:n}).$$

Our notion of regret corresponds to the excess total loss suffered from using ρ in place of the best model in $\mathcal{M}$. In our later analysis, we will show that the regret of our technique grows sublinearly and therefore that $\lim_{n\to\infty} \mathcal{R}_n(\rho, \mathcal{M})/n = 0$. In other words, the average instantaneous excess loss of our technique with respect to the best model in $\mathcal{M}$ asymptotically vanishes.

2.1. Bayesian Mixture Models

One way to construct a model with guaranteed low regret with respect to some model class $\mathcal{M}$ is to use a Bayesian mixture model

$$\xi_{\text{MIX}}(x_{1:n}) := \sum_{\rho\in\mathcal{M}} w_\rho \, \rho(x_{1:n}),$$

where $w_\rho > 0$ are prior weights satisfying $\sum_{\rho\in\mathcal{M}} w_\rho = 1$. It can readily be shown that, for any $\rho \in \mathcal{M}$, we have

$$\mathcal{R}_n(\xi_{\text{MIX}}, \{\rho\}) \le -\log w_\rho,$$

which implies that the regret of $\xi_{\text{MIX}}(x_{1:n})$ with respect to $\mathcal{M}$ is bounded uniformly by a constant that depends only on the prior weight assigned to the best model in $\mathcal{M}$. For example, the Context Tree Weighting approach of Willems et al. (1995) applies this principle recursively to efficiently construct a mixture model over a doubly-exponential class of tree models.

A more refined nonparametric Bayesian approach to mixing is also possible. Given a model class $\mathcal{M}$, the *switching* method (Koolen & de Rooij, 2013) efficiently maintains a mixture model ξ_{SWITCH} over all *sequences* of models in $\mathcal{M}$. We review here a restricted application of this technique based on the work of Veness et al. (2012) and Herbster & Warmuth (1998). More formally, given an indexed set of models $\{\rho_1, \rho_2, \dots, \rho_k\}$ and an index sequence $i_{1:n} \in \{1, 2, \dots, k\}^n$ let

$$\rho_{i_{1:n}}(x_{1:n}) := \prod_{t=1}^n \rho_{i_t}(x_t \,|\, x_{<t})$$

be a model which predicts at each time step t according to the model with index i_t. The switching technique implicitly computes a Bayesian mixture over the exponentially many possible index sequences. This mixture is efficiently computed in $O(k)$ per step by using

$$\xi_{\text{SWITCH}}(x_{1:n}) = \sum_{\rho\in\mathcal{M}} w_{\rho,n-1}\rho(x_n \,|\, x_{<n})$$

where, for $t \in 1 \dots n$, we have that

$$w_{\rho,t} := \frac{t}{t+1} w_{\rho,t-1}\rho(x_i \,|\, x_{<t}) + \frac{1}{t+1} \sum_{\nu\in\mathcal{M}\setminus\{\rho\}} w_{\nu,t-1}\,\nu(x_i \,|\, x_{<t}) \quad (1)$$

and in the base case $w_{\rho,0} := 1/k$ for each $\rho \in \mathcal{M}$. It can be shown (Veness et al., 2012) that for any $\rho_{i_{1:n}}$ we have

$$\mathcal{R}_n(\xi_{\text{SWITCH}}, \{\rho_{i_{1:n}}\}) \le [m(i_{1:n}) + 1]\,(\log k + \log n)$$

where $m(i_{1:n}) := \sum_{t=2}^n [\![i_{t-1} \neq i_t]\!]$ counts the number of times the index sequence switches models. In particular, if

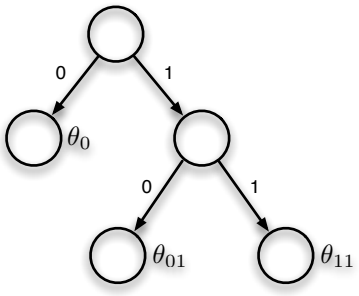

Figure 2. A prediction suffix tree.

a single model performs best throughout, ξ_{SWITCH} only incurs an additional $\log n$ cost compared to a Bayesian mixture model using a uniform prior. The switching method is a key component of the Context Tree Switching algorithm, which we review in Section 2.3, as well as our new SkipCTS algorithm. In practice, especially when the models in $\mathcal{M}$ are both adaptive and of varying complexity, the ability to rapidly switch between models often leads to an empirical performance improvement (Erven et al., 2007). A more comprehensive overview of switching strategies can be found in the work of Koolen & de Rooij (2013).

2.2. Prediction Suffix Trees

A Prediction Suffix Tree (Ron et al., 1996; Figure 2) is a type of variable order Markov model. Informally, a prediction suffix tree combines a collection of models using a data-partitioning tree, whose purpose is to select which particular model to use at each time step.

Given finite strings $c := c_1 \dots c_m \in \mathcal{X}^m$ and $x_{1:n} := x_1 \dots x_n \in \mathcal{X}^n$, we say that c is a *suffix* of $x_{1:n}$ if $x_{n-i} = c_{m-i}$ for all $i \in \{0, \dots, m-1\}$, and that c is the *context* of $x_{1:n}$ if it is a suffix of $x_{<n}$. We also write $\mathcal{T}_c(x_{1:n}) := \{i \in \mathbb{N} : c \text{ is a suffix of } x_{<i}, 1 \leq i \leq n\}$ to denote the set of time indices occurring in context c, and denote by $x^c_{1:n} := \langle x_i : i \in \mathcal{T}_c(x_{1:n}) \rangle$ the subsequence of $x_{1:n}$ that matches context c. Furthermore, given an alphabet $\mathcal{X}$ and an upper bound on the maximum Markov order $D \in \mathbb{N}$, let $\bar{\mathcal{X}} := \bigcup_{i=0}^{D} \mathcal{X}^i$, with $\mathcal{X}^0 := \{\epsilon\}$. A set $\mathcal{S} \subseteq \bar{\mathcal{X}}$ is called a *proper suffix set over* $\bar{\mathcal{X}}$ if for any finite string x there is exactly one $c \in \mathcal{S}$ such that c is a suffix of x. We also write $\mathcal{S}(x)$ to denote the matching context $c \in \mathcal{S}$ of a string x. The key property of $\mathcal{S}(\cdot)$ is that given $x_{1:n}$, it defines a partition of the time indices $\{1, \dots, n\}$ in the sense that the collection of sets $\{\mathcal{T}_c(x_{1:n})\}_{c \in \mathcal{S}}$ is mutually exclusive and exhaustive.

A prediction suffix tree is a tuple $(\mathcal{S}, \Theta)$ where $\mathcal{S}$ is a proper suffix set over $\bar{\mathcal{X}}$ and $\Theta := \{\theta_c\}_{c \in \mathcal{S}}$ is a set of sequential probabilistic models, with each model being associated with a particular context in $\mathcal{S}$. If $c_t = \mathcal{S}(x_{<t})$ is the context in $\mathcal{S}$ corresponding to $x_{<t}$, then the t^{th} symbol is predicted as $\theta_{c_t}(x_t \mid x^{c_t}_{<t})$. Since $\mathcal{S}$ is a proper suffix set, this gives the sequential probabilistic model

$$\psi_{\mathcal{S},\Theta}(x_{1:n}) := \prod_{c \in \mathcal{S}} \prod_{t \in \mathcal{T}_c(x_{1:n})} \theta_c(x_t \mid x^c_{<t}) = \prod_{c \in \mathcal{S}} \theta_c(x^c_{1:n}).$$

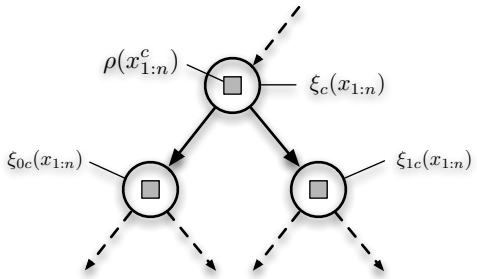

Figure 3. The Context Tree Switching recursive operation. For every context c we construct a model which switches between a base estimator ρ and a recursively defined split estimator.

2.3. Context Tree Switching

Context Tree Switching (CTS, Veness et al., 2012) is a recent extension of Context Tree Weighting (CTW, Willems et al., 1995) that retains the strong theoretical properties of CTW but performs better in practice. Let $\mathcal{X} := \{0, 1\}$ be the binary alphabet, $D \in \mathbb{N}$ be arbitrary but fixed, and let $\mathcal{C}_D$ be the collection of all binary prediction suffix trees $(\mathcal{S}, \Theta)$ of depth less than or equal to D. Let $\xi_{\text{CTS}}(x_{1:n})$ denote the probability assigned to $x_{1:n}$ by CTS. The regret $\mathcal{R}_n(\xi_{\text{CTS}}, \mathcal{C}_D)$ of CTS with respect to $\mathcal{C}_D$ is upper bounded by

$$\Gamma_D(\mathcal{S}) + (\Delta(\mathcal{S}) + 1) \log n + |\mathcal{S}| \gamma\left(\tfrac{n}{|\mathcal{S}|}\right), \qquad (2)$$

where $\Gamma_D(\mathcal{S}) := |\mathcal{S}| - 1 + |\{c : c \in \mathcal{S}, |c| \neq D\}|$ is the description length of $\mathcal{S}$, $\Delta(\mathcal{S}) := \max_{c \in \mathcal{S}} |c|$, and

$$\gamma(z) := \begin{cases} z & \text{if } 0 \leq z < 1 \\ \frac{1}{2} \log_2(z) + 1 & \text{if } z \geq 1. \end{cases}$$

Notice that the bound makes explicit an Occam bias towards smaller prediction suffix trees. The last summand in Equation 2 arises from having to learn the parameters of $|\mathcal{S}|$ unknown Bernoulli distributions. This, combined with the fact that $\Gamma_D(\mathcal{S}) = O(|\mathcal{S}|)$, causes CTS to perform well whenever a small prediction suffix tree is sufficient to describe the data.

Algorithm. CTS is best understood as a recursive application of the switching technique described in Section 2.1. As depicted in Figure 3, for each context $c \in \bar{\mathcal{X}}$ we define a switching model between two components: a *base estimator* ρ which predicts the substring $x^c_{1:n}$ and a *split estimator* which subdivides c into $0c$ and $1c$ and predicts $x^c_{1:n}$ by querying the corresponding switching models for the further partitioned data. The latter operation is well-defined

by the partitioning property of proper suffix sets: x_n belongs to either $x_{1:n}^{0c}$ or $x_{1:n}^{1c}$ but not both. The algorithm then assigns a probability to $x_{1:n} \in \mathcal{X}^n$ using its top-level switching model, i.e. $\xi_{\text{CTS}}(x_{1:n}) := \xi_\epsilon(x_{1:n})$.

The algorithmic core of CTS is a *context tree* data structure: a perfect binary tree of depth D whose nodes correspond to all possible strings in $\bar{\mathcal{X}}$. Each node $c \in \bar{\mathcal{X}}$ stores a base estimator ρ_c as well as the data-dependent quantities ξ_c, α_c, and β_c. Informally, $\alpha_c(x_{<t})$ and $\beta_c(x_{<t})$ correspond to $w_{\rho,t-1}$ in Equation 1, while ξ_c corresponds to $\xi_{\text{SWITCH}}(x_{1:t})$. CTS incrementally maintains these quantities as follows. Given a new symbol x_t and its associated history $x_{<t}$, CTS updates the $D+1$ nodes along the path $\epsilon, x_{t-1}, x_{t-2:t-1}, \ldots, x_{t-D:t-1}$; all other nodes are left unchanged. CTS performs a post-order traversal along this path, first updating each node's base estimator and the other quantities as follows. At the leaf $c = x_{t-D:t-1}$, CTS sets $\alpha_c(x_{1:t}) \leftarrow \alpha_c(x_{<t})\rho_c(x_t \mid x_{<t}^c)$ and then $\xi_c(x_{1:t}) \leftarrow \alpha_c(x_{1:t})$. At the internal nodes, the following updates occur:

$$\begin{aligned}
\xi_c(x_{1:t}) &\leftarrow \alpha_c(x_{<t})\rho_c(x_t \mid x_{<t}^c) + \beta_c(x_{<t})z_{c,t} \\
\alpha_c(x_{1:t}) &\leftarrow \frac{1}{t+1}\xi_c(x_{1:t}) + \frac{t-1}{t+1}\alpha_c(x_{<t})\rho_c(x_t \mid x_{<t}^c) \\
\beta_c(x_{1:t}) &\leftarrow \frac{1}{t+1}\xi_c(x_{1:t}) + \frac{t-1}{t+1}\beta_c(x_{<t})z_{c,t}
\end{aligned}$$

where $z_{c,t} := \xi_{x'c}(x_{1:t})/\xi_{x'c}(x_{<t})$ is the probability assigned to x_t by the recursively-defined split estimator, with $x' := x_{t-|c|-1}$. Every node c is initialized with $\alpha_c(\epsilon) = \beta_c(\epsilon) = \frac{1}{2}$, except for leaf nodes where we set $\alpha_c(\epsilon) = 1$ and $\beta_c(\epsilon) = 0$ to reflect the fact that no further splitting occurs at depth D.

2.4. Choice of Base Estimator

If the alphabet is binary, a natural choice of base model is the KT estimator (Krichevsky & Trofimov, 1981). This estimator probabilistically predicts each symbol according to a Beta-Binomial model, using a Beta$(\frac{1}{2}, \frac{1}{2})$ prior over the unknown parameter. The regret of the KT estimator with respect to any Bernoulli process is known to be at most $\frac{1}{2}\log n + 1$. Non-binary alphabets can be handled in a number of ways. The most direct approach is to use a Dirichlet-Multinomial model with a Dirichlet$(\frac{1}{2})$ prior over $\mathcal{X}$, leading to the multi-alphabet KT estimator, whose regret is $O(|\mathcal{X}|\log n)$ (Tjalkens et al., 1993). When $|\mathcal{X}|$ is large and only a small fraction of the possible symbols are observed, this approach is inefficient (e.g. Volf, 2002). A recently developed solution to this problem is the Sparse Adaptive Dirichlet (SAD) estimator (Hutter, 2013). The SAD approach enjoys regret guarantees close to those of the multi-alphabet KT restricted to the subalphabet $\mathcal{A} \subseteq \mathcal{X}$ of symbols which occur in $x_{1:n}$. In Section 4 we describe a large experiment whose performance is much improved by the use of a SAD-like estimator.

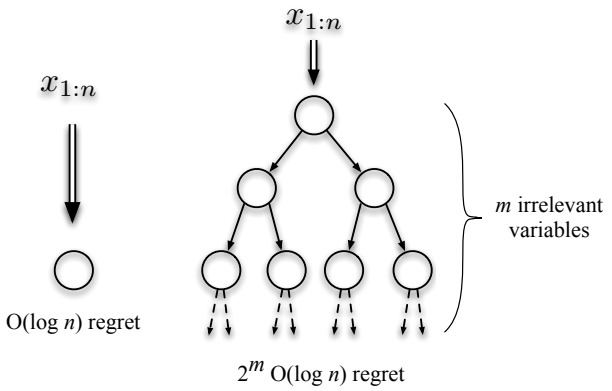

Figure 4. Worst-case regret when using CTS instead of SkipCTS.

3. Skip Context Tree Switching

In this section we generalize CTS to partial context matches to produce the Skip Context Tree Switching (SkipCTS) algorithm. We begin with some notation describing partial context matches, then describe how the SkipCTS algorithm incorporates these. Finally we provide a bound on the regret of SkipCTS with respect to the set of all bounded K-skip prediction suffix trees.

To gain some intuition as to why ignoring irrelevant variables matters, consider what happens internally in the context tree in the presence of irrelevant variables. As the right-hand side of Figure 4 shows, in the worst case, the data used to train the base models can be dispersed uniformly across 2^m bins. On the other hand, SkipCTS can ignore the intervening variables and directly aggregate all the available data into a single bin (Figure 4, left). In the particular case where the base estimator is the KT estimator (with regret $O(\log n)$ for memoryless sources), we see that SkipCTS can enjoy an exponential reduction in regret compared to CTS.

3.1. Partial Context Matches

We begin with some notation. Let $\star$ denote a wildcard symbol and let $\mathcal{Y} := \mathcal{X} \cup \{\star\}$ be the wildcard extension of $\mathcal{X}$. We say that a string $a \in \mathcal{X}^m$ *matches* $b \in \mathcal{Y}^m$ if for all $1 \leq i \leq m$ such that $b_i \neq \star$, we have $a_i = b_i$. For example, if $\mathcal{X}$ is the set of uppercase letters then BAY, BOY, BUY, etc. all match B$\star$Y. Given a finite string x, we say that $c \in \mathcal{Y}^m$ is a suffix of x iff $x = yc'$ for some $c' \in \mathcal{X}^m$ such that c' matches c. We call a string c *k-skip contiguous* if it contains at most k contiguous runs of $\star$ symbols. For example, $0\star\star1\star0$ is 2-skip contiguous. Finally, we denote the set of all k-skip contiguous strings of length i by $\mathcal{Y}_k^i$ and the union of all such sets for $i = 0 \ldots D$ as $\bar{\mathcal{Y}}_k$.

3.2. Algorithm

The SkipCTS algorithm is parametrized by a maximum depth $D \in \mathbb{N}$ and a number of allowable skips $K \in \mathbb{N}$. The key idea is to maintain a context tree whose nodes correspond to all possible contexts $c \in \bar{\mathcal{Y}}_K$. To do so, we generalize the CTS update equations described in Section 2.3, leading to a recursive switching model which chooses between not only a base estimator and a split estimator, but also between a variable number of recursively defined skip estimators. Effectively, these additional estimators correspond to ignoring one or more symbols in the context. As we will show in Section 3.3, the addition of these new models allows us to obtain a competitive regret bound with respect to the set of all bounded K-skip prediction suffix trees.

In designing SkipCTS, one additional subtle issue arises: some of the contexts in $\bar{\mathcal{Y}}_K$ are redundant for the purpose of sequential prediction. To see this, consider two contexts from $\bar{\mathcal{Y}}_K$, $c = 010$ and $c' = \star 010$. Given any string $x_{1:n} \in \mathcal{X}^n$, both contexts induce the same substring $x^c_{1:n} = x^{c'}_{1:n}$, so that only one of them needs to be considered by our algorithm. More generally, the contexts $c \in \bar{\mathcal{Y}}_K$ for which $c = \star^l c'$ with $l \in \mathbb{N}$ are equivalent. We refine the algorithm by considering only the set of *representative contexts*, i.e. the contexts c such that $c = xc'$ where $x \in \mathcal{X}$. In other words, representative contexts are those contexts which do not contain trailing wildcard symbols. This results in a more efficient algorithm than if we were to consider all possible contexts in $\bar{\mathcal{Y}}_K$. To avoid notational clutter, from here onwards we will use the notation $\bar{\mathcal{Y}}_K$ to denote the set of representative contexts. For a given string x, we call a representative context c which is a suffix of x a *representative suffix*.

For every $c \in \bar{\mathcal{Y}}_K$ we incrementally maintain a base estimator ρ_c and the quantities $\xi_c, \alpha_c, \beta_{c,0}, \beta_{c,1}, \ldots$. The number of $\beta_{c,\cdot}$ quantities depends on c as follows. Define $\kappa(c)$ as the number of contiguous runs of $\star$ symbols in c. For $c \in \bar{\mathcal{Y}}$, define $r(c) := 1$ if either $|c| = D$ or $\kappa(c) = K$, and $D - |c|$ otherwise. The $\beta_{c,0}$ term then corresponds to the split estimator, while the remaining $\beta_{c,\cdot}$ terms correspond to $r(c) - 1$ skip estimators; in particular, when $r(c) = 1$ no skip estimators are updated for c.

Upon observing a new symbol x_t, SkipCTS first updates $\alpha_c(x_{1:t}) \leftarrow \alpha_c(x_{<t})\rho_c(x_t \,|\, x^c_{<t})$ and then $\xi_c(x_{1:t}) \leftarrow \alpha_c(x_{1:t})$ for all $c \in \bar{\mathcal{Y}}_K$ representative suffixes of $x_{<t}$ with $|c| = D$. The remaining representative suffixes of $x_{<t}$ are updated recursively as

$$\xi_c(x_{1:t}) \leftarrow \alpha_c(x_{<t}) b_{c,t} + \sum_{l=0}^{r(c)-1} \beta_{c,l}(x_{<t}) z_{c,l,t} \tag{3}$$

$$\alpha_c(x_{1:t}) \leftarrow \eta_t \xi_c(x_{1:t}) + \left(\tfrac{t}{t+1} - \eta_t\right) \alpha_c(x_{<t}) b_{c,t} \tag{4}$$

$$\beta_{c,l}(x_{1:t}) \leftarrow \eta_t \xi_c(x_{1:t}) + \left(\tfrac{t}{t+1} - \eta_t\right) \beta_{c,l}(x_{<t}) z_{c,l,t} \tag{5}$$

where $\eta_t = 1/\left[r(c)(t+1)\right]$, $b_{c,t} := \rho_c(x_t \,|\, x^c_{<t})$, and $z_{c,l,t}$ is the probability assigned to x_t by the l^{th} skip estimator, which is defined as

$$z_{c,l,t} := \xi_{x'c'}(x_{1:t}) \,/\, \xi_{x'c'}(x_{<t}), \tag{6}$$

where $c' := \star^l c$ and $x' := x_{t-|c'|-1}$. As with CTS, any node not corresponding to a representative suffix of $x_{<t}$ is left unchanged. The probability $\xi_{\text{SkipCTS}}(x_{1:n})$ output by SkipCTS is the probability assigned by the root switching model $\xi_\epsilon(x_{1:n})$.

The particular update structure of SkipCTS, i.e. the contexts c for which $\xi_c(x_{1:n})$ are updated, depends on both K and D. As with CTS, we set $\alpha_c(\epsilon) = 1$ whenever $|c| = D$. For $|c| < D$, we set $\alpha_c(\epsilon) = \frac{1}{2}$ and

1. $\beta_{c,0}(\epsilon) = \frac{1}{2} \quad \text{if} \quad \kappa(c) = K$;
2. otherwise
$$\beta_{c,l}(\epsilon) = \begin{cases} \frac{1}{4} & \text{if } l = 0; \\ \frac{1}{4r(c)} & \text{if } 1 \le l < r(c); \\ 0 & \text{otherwise.} \end{cases}$$

It can be verified that for any $c \in \bar{\mathcal{Y}}_K$ we have $\alpha_c(\epsilon) + \sum_{l=0}^{D-1} \beta_{c,l}(\epsilon) = 1$. In general, we may freely initialize the non-zero α_c and β_c terms, provided they are nonnegative and sum to one. Because $\kappa(c)$ in Equation 3 ranges from 0 to K and $|c|$ from 0 to D, performing one update requires $O(D^{2K+1})$ operations – effectively the number of representative suffixes $c \in \bar{\mathcal{Y}}_K$ which match $x_{<t}$. In particular, note that when $K = 0$ we recover the original Context Tree Switching algorithm of Veness et al. (2012).

3.3. Regret Analysis

In this section we provide a bound on the regret of Skip Context Tree Switching with respect to any bounded K-skip prediction suffix tree $(\mathcal{S}, \Theta)$, where K is fixed and $\mathcal{S}$ is a proper suffix set over $\bar{\mathcal{Y}}$. At a high level, Lemma 1 bounds the regret induced by the base estimators at the leaves of $(\mathcal{S}, \Theta)$. Lemma 2 bounds the regret contributed from a single level of internal nodes in the tree, and Lemma 3 applies a recursive argument to combine Lemmas 1 and 2. Theorem 1 finally uses Lemma 3 to bound the regret of SkipCTS with respect to an arbitrary K-skip prediction suffix tree.

Lemma 1. *For any proper suffix set $\mathcal{S}$ over $\bar{\mathcal{Y}}_K$ and for any $x_{1:n} \in \mathcal{X}^n$, we have*

$$\prod_{c \in \mathcal{S}} \alpha_c(x_{1:n}) \ge \frac{1}{n+1} \prod_{c \in \mathcal{S}} \alpha_c(\epsilon) \rho_c(x^c_{1:n}).$$

Proof. Let $c_t := \mathcal{S}(x_{<t})$ be the context in $\mathcal{S}$ which is a suffix of $x_{<t}$, which is guaranteed to be unique as $\mathcal{S}$ is a

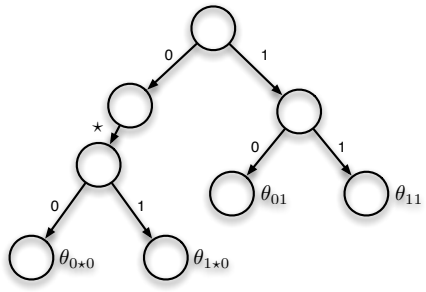

Figure 5. A 1-skip prediction suffix tree.

proper suffix set. By combining Equations 3 and 4 we have $\alpha_{c_t}(x_{1:t}) \geq \frac{t}{t+1}\alpha_{c_t}(x_{<t})\rho_{c_t}(x_t \mid x^{c_t}_{<t})$. By definition, for all other $c \in \mathcal{S}$ we have $\alpha_c(x_{1:t}) = \alpha_c(x_{<t})$. Recalling that $\mathcal{T}_c(x_{1:n}) := \{t \in \mathbb{N} : c \text{ is a suffix of } x_{<t}\}$, we expand Equation 4 as

$$\begin{aligned}\alpha_c(x_{1:n}) &\geq \alpha_c(\epsilon) \prod_{t\in\mathcal{T}_c(x_{1:n})} \tfrac{t}{t+1}\rho_c(x_t \mid x^c_{<t}) \\ &= \alpha_c(\epsilon)\rho_c(x^c_{1:n}) \prod_{t\in\mathcal{T}_c(x_{1:n})} \tfrac{t}{t+1},\end{aligned}$$

hence

$$\prod_{c\in\mathcal{S}} \alpha_c(x_{1:n}) \geq \prod_{c\in\mathcal{S}} \alpha_c(\epsilon)\rho_c(x^c_{1:n}) \prod_{t\in\mathcal{T}_c(x_{1:n})} \tfrac{t}{t+1},$$

and the desired result follows by recalling that $\{\mathcal{T}_c(x_{1:n})\}_{c\in\mathcal{S}}$ is a partition of $\{1,\dots,n\}$. □

We now define the various kinds of decision points (e.g. split only; split or skip) within the context tree.

Definition 1. *Let $\mathcal{S}$ be a proper suffix set over $\bar{\mathcal{Y}}$. A string $c \in \bar{\mathcal{Y}}$ is a* choice point in $\mathcal{S}$ *whenever c is a strict suffix of some $c' \in \mathcal{S}$ and $c = xc'$, where $x \in \mathcal{X}$.*

Definition 2. *Let $\mathcal{S}$ be a proper suffix set over $\bar{\mathcal{Y}}$, and for $c \in \bar{\mathcal{Y}}$ let $\mathcal{E}_{\mathcal{S}}(c) := \{l \in \mathbb{N} : \star^l c \text{ is a strict suffix of some } c' \in \mathcal{S}\}$. We call $\tilde{\mathcal{S}} := \{(c,l) : c \text{ is a choice point in } \mathcal{S}, l \in \mathcal{E}_{\mathcal{S}}(c)\}$ the set of* choice nodes *of $\mathcal{S}$.*

Figure 5, for example, is described by $\mathcal{S} = \{0\star 0, 1\star 0, 01, 11\}$, of which 0 and 1 are choice points. The set of choice nodes corresponding to $\mathcal{S}$ is $\tilde{\mathcal{S}} := \{(0,1),(1,0)\}$. Intuitively, these choice nodes correspond exactly to the nodes with more than one child.

Definition 3. *Let $c := c_1 c_2 \dots c_m \in \mathcal{Y}^m$. We define the* effective length *of c as $\ell(c) := |\{i \in \{1,\dots m\} : c_i \neq \star\}|$ and for a set $\mathcal{V}$ of such strings define $\ell(\mathcal{V}) := \max_{c\in\mathcal{V}} \ell(c)$.*

Recall that our aim is to bound $-\log \xi_\epsilon(x_{1:n})$. Having bounded the product of terms at the leaves, $\prod_{c\in\mathcal{S}} \alpha_c(x_{1:n})$, we now derive a similar bound for the internal nodes.

Lemma 2. *Let $\mathcal{S}$ be a proper suffix set over $\bar{\mathcal{Y}}_K$, and let $\tilde{\mathcal{S}}_d := \{(c,l) \in \tilde{\mathcal{S}} : \ell(c) = d\}$. For any $d \in \{0,\dots,D-1\}$ and any $x_{1:n} \in \mathcal{X}^n$, we have that*

$$\prod_{(c,l)\in\tilde{\mathcal{S}}_d} \beta_{c,l}(x_{1:n}) \geq \frac{1}{n+1} \prod_{(c,l)\in\tilde{\mathcal{S}}_d} \beta_{c,l}(\epsilon) \prod_{x\in\mathcal{X}} \xi_{x\star^l c}(x_{1:n}).$$

Proof. For any $(c,l) \in \tilde{\mathcal{S}}$, and any $t \in \{1,\dots,n\}$, let $c' = \star^l c$ and $x' = x_{t-|c'|-1}$. Now for any $x \in \mathcal{X}\setminus\{x'\}$, we have $\xi_{xc'}(x_{1:t}) = \xi_{xc'}(x_{<t})$, which enables us to rewrite Equation 6 as

$$z_{c,l,t} = \prod_{x\in\mathcal{X}} \xi_{xc'}(x_{1:t})/\xi_{xc'}(x_{<t}),$$

which, following a similar approach to the proof of Lemma 1 leads to the desired result. □

Lemma 3. *For every proper suffix set $\mathcal{S}$ over $\bar{\mathcal{Y}}_K$, with $\tilde{\mathcal{S}}$ denoting the set of choice nodes of $\mathcal{S}$, we have*

$$\xi_\epsilon(x_{1:n}) \geq n^{-\ell(\mathcal{S})} \prod_{(c,l)\in\tilde{\mathcal{S}}} \beta_{c,l}(\epsilon) \prod_{c\in\mathcal{S}} \alpha_c(x_{<n})\rho_c(x_n \mid x^c_{<n}).$$

Proof. For any $c \in \bar{\mathcal{Y}}_K$ and $l \leq r(c)$, $\xi_c(x_{1:n}) \geq \alpha_c(x_{<n})\rho_c(x_n \mid x^c_{<n})$ and $\xi_c(x_{1:n}) \geq \beta_c(x_{<n})z_{c,l,n}$. As $\mathcal{S}$ is a proper suffix set, at any time step t at most one $c \in \tilde{\mathcal{S}} \cup \mathcal{S}$ of effective length $d \leq \ell(\mathcal{S})$ matches $x_{<t}$. By recursively applying Lemma 2 to the right hand side of Equation 3 for every $c \in \tilde{\mathcal{S}}$, and keeping the left hand side whenever $c \in \mathcal{S}$, we obtain

$$\xi_\epsilon(x_{1:n}) \geq \prod_{d=0}^{\ell(\mathcal{S})-1} \prod_{(c,l)\in\tilde{\mathcal{S}}_d} \tfrac{1}{n}\beta_{c,l}(\epsilon) \prod_{c\in\mathcal{S}} \alpha_c(x_{<n})\rho_c(x_n \mid x^c_{<n}),$$

The result then follows since $\tilde{\mathcal{S}} = \cup_{d=0}^{\ell(\mathcal{S})-1} \tilde{\mathcal{S}}_d$. □

We are now in a position to bound the regret of Skip Context Tree Switching with respect to any skipping prediction suffix tree structure that can be obtained by pruning the SkipCTS context tree.

Theorem 1. *Let $\psi_{\mathcal{S}}$ denote a $D \in \mathbb{N}$ bounded k-skip prediction suffix tree $(\mathcal{S},\Theta)$, where $\mathcal{S}$ is a proper suffix set over $\bar{\mathcal{Y}}_k$ and $\Theta := \{\rho_c\}_{c\in\mathcal{S}}$ is a set of sequential probabilistic models inside the SkipCTS context tree. For any $x_{1:n} \in \mathcal{X}^n$, the regret of SkipCTS run with parameters D and $K \geq k$ with respect to $\psi_{\mathcal{S}}$ is bounded as*

$$\mathcal{R}_n(\xi_{\text{SkipCTS}}, \{\psi_{\mathcal{S}}\}) \leq [\ell(\mathcal{S})+1]\log n + \Gamma^K_D(\mathcal{S}),$$

where $\Gamma^K_D(\mathcal{S}) := -\sum_{(c,l)\in\tilde{\mathcal{S}}} \log \beta_{c,l}(\epsilon) - \sum_{c\in\mathcal{S}} \log \alpha_c(\epsilon)$.

Proof. (Sketch) Beginning with $\mathcal{R}_n(\xi_{\text{SkipCTS}}, \{\psi_{\mathcal{S}}\})$ we apply Lemma 3, then Lemma 1, and finally simplify using $\psi_{\mathcal{S}}(x_{1:n}) := \prod_{c\in\mathcal{S}} \rho_c(x^c_{1:n})$. □

If the data is generated by some unknown prediction suffix tree and the base estimators are KT estimators, the above regret bound leads to a result that is similar to the regret bound for CTS given by Veness et al. (2012), save for two main differences. First, recall that $\mathcal{C}_D$, the collection of models considered by CTS, is a subset of $\mathcal{C}_{D,K}$, the collection of models considered by SkipCTS (with equality only when $K = 0$). Our bound thus covers a broader collection of models. Second, for a proper suffix set $\mathcal{S}$ defined over $\mathcal{X}$, i.e. a no skip prediction suffix tree, the description length $\Gamma_D^K(\mathcal{S})$ under SkipCTS with $K > 0$ is necessarily larger than its CTS counterpart. While these differences negatively affect our regret bound, we have seen in Section 3 that we should expect significant savings whenever the data can be well-modelled by a small K-skip prediction suffix tree. We explore these issues further in the next section.

4. Experiments

We tested the Skip Context Tree Switching on a series of prediction problems. The first set of experiments uses a popular data compression benchmark, while the second set of experiments investigates performance on a diverse set of structured image prediction problems taken from an open source Reinforcement Learning test framework. A reference implementation of SkipCTS is provided at: `http://github.com/mgbellemare/SkipCTS`.

4.1. The Calgary Corpus

Our first experiment evaluated SkipCTS in a pure compression setting. Recall that any algorithm which sequentially assigns probabilities to symbols can be used for compression by means of arithmetic coding (Witten et al., 1987). In particular, given a model ξ assigning a probability $\xi(x_{1:n})$ to $x_{1:n} \in \mathcal{X}^n$, arithmetic coding is guaranteed to produce a compressed file size of essentially $-\log_2 \xi(x_{1:n})$.

We ran SkipCTS (with $D = 48$, $K = 1$) and CTS (with $D = 48$) on the Calgary Corpus (Bell et al., 1989), an established compression benchmark composed of 14 different files. The results, provided in Table 1, show that SkipCTS performs significantly better than CTS on certain files, and never suffers by more than a negligible amount. Of interest, the files best improved by SkipCTS are those which contain highly-structured binary data: GEO, OBJ1, and OBJ2. For reference, we also included some CTW experiments, indicated by the CTW and and SkipCTW rows, that measured the performance of skipping using the original recursive CTW weighting scheme; here we see that the addition of skipping also helps. Table 1 also provides results for CTW*, an enhanced version of CTW for byte-based data (Willems, 2013). Here both CTS and SkipCTS outperform CTW, with SkipCTS providing the best results overall. Finally, it is worth noting that averaged over the Calgary Corpus, the bits per byte performance of SkipCTS is superior (2.10 vs 2.12) to DEPLUMP (Gasthaus et al., 2010), a state-of-the-art n-gram model. While SkipCTS is consistently slightly worse for text data, it is significantly better on binary data. It is also worth pointing out that no regret guarantees are yet known for DEPLUMP.

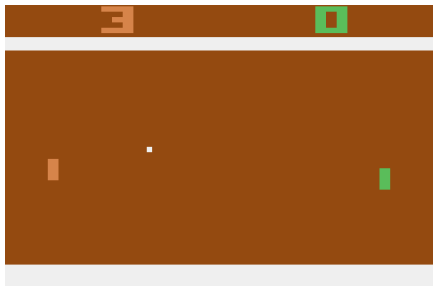

Figure 6. The game PONG, in which the player controls the vertical position of a *paddle* in order to return a ball and score points.

4.2. Atari 2600 Frame Prediction

We also tested our algorithm on the task of video game screen prediction. We used the Arcade Learning Environment (Bellemare et al., 2013a), an interface that allows agents to interact with Atari 2600 games. Figure 6 depicts the well-known PONG, one of the Atari 2600's flagship games. In the Atari 2600 prediction setting, the alphabet $\mathcal{X}$ is the set of all possible Atari 2600 screens. Because each screen contains 160×210 7-bit pixels, it is both impractical and undesirable to learn a model which predicts each $x_t \in \mathcal{X}$ atomically. Instead, we take a similar approach to that of Bellemare et al. (2013b): we divide the screen into 16×16 blocks and predict each block atomically using SkipCTS or CTS combined with the SAD estimator.

Each block prediction is made using a context composed of the symbol value of neighbouring blocks at previous timesteps, as well as the last action taken, for a total of 11 variables. In this setting, skipping irrelevant variables is particularly important because of the high branching factor at each level. For example, when predicting the motion of the opponent's paddle in PONG, SkipCTS can disregard horizontally neighbouring blocks and the player's action.

We trained SkipCTS with $K = 0$ and 1 on 54 Atari 2600 games. Each experiment consisted of 10 trials, each lasting 100,000 time steps, where one time step corresponds to 4 emulated frames. Each trial was assigned a specific random seed which was used for all values for K. We report the average log-loss per frame over the last 4500 time steps, corresponding to 5 minutes of real-time Atari 2600 play. Throughout our trials actions were selected uniformly at random from each game's set of legal actions.

The full table of results is provided as supplementary mate-

Table 1. Compression results on the Calgary Corpus, in average bits needed to encode each byte. Highlights indicate improvements greater than 3% from CTW to SkipCTW and from CTS to SkipCTS, respectively. CTW* results are taken from Willems (2013).

File	bib	book1	book2	geo	news	obj1	obj2	paper1	paper2	pic	progc	progl	progp	trans
CTW*	1.83	2.18	1.89	4.53	2.35	3.72	2.40	2.29	2.23	0.80	2.33	1.65	1.68	1.44
CTW	2.25	2.31	2.12	5.01	2.78	4.63	3.19	2.84	2.59	0.90	3.00	2.11	2.24	2.09
SkipCTW	2.15	2.32	2.10	3.91	2.77	4.57	2.96	2.75	2.54	0.90	2.91	2.00	2.08	1.83
Diff.	**4.4%**	-0.4%	0.9%	**22.0%**	0.4%	1.3%	**7.2%**	**3.2%**	1.9%	0.0%	3.0%	**5.2%**	**7.1%**	**12.4%**
CTS	1.81	2.20	1.90	4.18	2.34	3.66	2.36	2.28	2.23	0.79	2.33	1.61	1.64	1.39
SkipCTS	1.75	2.20	1.89	3.60	2.34	3.40	2.19	2.26	2.22	0.76	2.30	1.59	1.61	1.35
Diff.	**3.3%**	0.0%	0.5%	**13.9%**	0.0%	**7.1%**	**7.2%**	0.9%	0.4%	**3.8%**	1.3%	1.2%	1.8%	2.9%

rial. For each game we computed the improvement in log-loss per frame and determined whether the difference in loss was statistically significant using the Wilcoxon signed rank test. As a whole, SkipCTS achieved lower log-loss than CTS in 54 out of 55 games; all these differences are significant. While SkipCTS performed slightly worse in Elevator Action, the difference was not statistically significant. The average overall log-loss improvement was 9.0% and the median, 8.25%; improvements ranged from -2% (Elevator Action) to 36% (Freeway). SkipCTS with $K = 1$ processed on average 34 time steps (136 frames) per second, corresponding to just over twice the real-time speed of the Atari 2600. We further ran our algorithm with $K = 2$ and observed an additional, significant increase in predictive performance on 18 games (up to 21.7% over $K = 1$ for Time Pilot). On games where $K = 2$ is unnecessary, however, the performance of SkipCTS degraded somewhat. As discussed above, this behaviour is an expected consequence of the larger $\Gamma_D^K(\mathcal{S})$.

5. Discussion

We have seen that by allowing context trees to skip over variables, SkipCTS can achieve substantially better performance over CTS in problems where a good variable ordering may not be known *a priori*. Theoretically we have seen that SkipCTS can, in the extreme case, have exponentially lower regret. Empirically we observe substantial benefits in practice over state of the art lossless compression algorithms in problems involving highly structured data (e.g. the GEO problem in the Calgary Corpus). The dramatic and consistent improvement seen across over 50 Atari prediction problems indicate that SkipCTS is especially important in multi-dimensional prediction problems where issues of variable ordering are naturally exacerbated.

The main drawback of SkipCTS is the increased computational complexity of inference as a result of the more expressive model class. However, our experiments have demonstrated that small values of K can make a substantial difference. Furthermore, the computational and memory costs of SkipCTS can be alleviated in practice. The tree structure induced by the recursive SkipCTS update (Equations 3–5) can naturally be parallelized, while the SkipCTS memory requirements can easily be bounded through hashing. Finally note that *sampling* from the model remains a $O(D)$ operation, so, for instance, planning with a SkipCTS-based reinforcement learning model is nearly as efficient as planning with a CTS-based model.

Tree-based models have a long history in sequence prediction, and the persistent issue of variable ordering has been confronted in many ways. The main strengths of SkipCTS are inherited from CTW – efficient, incremental, and exact Bayesian inference, and strong theoretical guarantees on asymptotic regret. Other approaches with more representational flexibility lack these strengths. In the model based reinforcement learning setting, some methods (e.g. McCallum, 1996; Holmes & Isbell, 2006; Talvitie, 2012) extend the traditional predictive suffix tree by allowing variables from different time steps to be added in any order, or by allowing the tree to excise portions of history, but these methods are not incremental and do not provide regret guarantees. Bayesian decision tree learning methods (e.g. Chipman et al., 1998; Lakshminarayanan et al., 2013) could in principle be applied in the sequential prediction setting. These typically allow arbitrary variable ordering, but require approximate inference to remain tractable.

6. Conclusion

In this paper we presented Skip Context Tree Switching, a polynomial-time algorithm which efficiently mixes over sequences of prediction suffix trees that may skip over K contiguous runs of variables. Our results show that SkipCTS is practical for small K and can produce significant empirical improvements compared to members of the Context Tree Weighting family (even with $K = 1$) in problems where irrelevant variables are naturally present.

Acknowledgments

The authors would like to thank Alex Graves, Andriy Mnih and Michael Bowling for some helpful discussions.

References

Bell, Timothy, Witten, Ian H., and Cleary, John G. Modeling for text compression. *ACM Computing Surveys (CSUR)*, 21(4):557–591, 1989.

Bellemare, Marc G., Naddaf, Yavar, Veness, Joel, and Bowling, Michael. The Arcade Learning Environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47, June 2013a.

Bellemare, Marc G., Veness, Joel, and Bowling, Michael. Bayesian learning of recursively factored environments. In *Proceedings of the Thirtieth International Conference on Machine Learning*, 2013b.

Chipman, Hugh A., George, Edward I., and McCulloch, Robert E. Bayesian CART model search. *Journal of the American Statistical Association*, 93(443):935–948, 1998.

Erven, Tim Van, Grunwald, Peter, and de Rooij, Steven. Catching up faster in Bayesian model selection and model averaging. In *NIPS*, 2007.

Gasthaus, Jan, Wood, Frank, and Teh, Yee Whye. Lossless compression based on the sequence memoizer. In *Data Compression Conference (DCC)*, 2010.

Herbster, Mark and Warmuth, Manfred K. Tracking the best expert. *Machine Learning*, 32(2):151–178, 1998.

Holmes, Michael P. and Isbell, Jr, Charles Lee. Looping suffix tree-based inference of partially observable hidden state. In *Proceedings of the 23rd International Conference on Machine Learning*, pp. 409–416, 2006.

Hutter, Marcus. Sparse adaptive Dirichlet-multinomial-like processes. In *Proceedings of the Conference on Learning Theory (COLT)*, 2013.

Koolen, Wouter M. and de Rooij, Steven. Universal codes from switching strategies. *IEEE Transactions on Information Theory*, 59(11):7168–7185, 2013.

Krichevsky, R. and Trofimov, V. The performance of universal encoding. *IEEE Transactions on Information Theory*, 27(2):199–207, 1981.

Lakshminarayanan, Balaji, Roy, Daniel M., and Teh, Yee Whye. Top-down particle filtering for Bayesian decision trees. In *Proceedings of the 30th International Conference on Machine Learning*, 2013.

McCallum, Andrew K. *Reinforcement learning with selective perception and hidden state*. PhD thesis, University of Rochester, 1996.

Ron, Dana, Singer, Yoram, and Tishby, Naftali. The power of amnesia: Learning probabilistic automata with variable memory length. *Machine learning*, 25(2):117–149, 1996.

Talvitie, Erik. Learning partially observable models using temporally abstract decision trees. In *Advances in Neural Information Processing Systems (25)*, 2012.

Tjalkens, Tj. J, Shtarkov, Y.M., and Willems, F.M.J. Context tree weighting: Multi-alphabet sources. In *14th Symposium on Information Theory in the Benelux*, pp. 128–135, 1993.

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

Veness, Joel, Ng, Kee Siong, Hutter, Marcus, and Bowling, Michael H. Context tree switching. In *Data Compression Conference (DCC)*, pp. 327–336, 2012.

Volf, P. *Weighting techniques in data compression: Theory and algorithms*. PhD thesis, Eindhoven University of Technology, 2002.

Willems, Frans M., Shtarkov, Yuri M., and Tjalkens, Tjalling J. The context tree weighting method: Basic properties. *IEEE Transactions on Information Theory*, 41:653–664, 1995.

Willems, Frans M., Shtarkov, Yuri M., and Tjalkens, Tjalling J. Context weighting for general finite-context sources. *IEEE Transactions on Information Theory*, 42 (5):1514–1520, 1996.

Willems, Frans M. J. CTW website. http://www.ele.tue.nl/ctw/, 2013.

Witten, Ian H., Neal, Radford M., and Cleary, John G. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, 1987.

Skip Context Tree Switching
Supplementary Material

1 Proof of Theorem 1

The probability SkipCTS assigns to $x_{1:n} \in \mathcal{X}^n$ is defined as $\xi_{\text{SkipCTS}}(x_{1:n}) := \xi_\epsilon(x_{1:n})$. We thus expand the loss $-\log \xi_{\text{SkipCTS}}(x_{1:n})$ using Lemma 3:

$$\begin{aligned} -\log \xi_{\text{SkipCTS}}(x_{1:n}) &\leq -\log \left[n^{-\ell(\mathcal{S})} \prod_{(c,l)\in\tilde{\mathcal{S}}} \beta_{c,l}(\epsilon) \prod_{c\in\mathcal{S}} \alpha_c(x_{<n}) \rho_c(x_n \mid x^c_{<n}) \right] \\ &= \ell(\mathcal{S}) \log n - \log \prod_{(c,l)\in\tilde{\mathcal{S}}} \beta_{c,l}(\epsilon) - \log \left[\prod_{c\in\mathcal{S}} \alpha_c(x_{<n}) \rho_c(x_n \mid x^c_{<n}) \right]. \end{aligned} \tag{1}$$

We now expand the rightmost term of this equation using Lemma 1:

$$\begin{aligned} -\log \left[\prod_{c\in\mathcal{S}} \alpha_c(x_{<n}) \rho_c(x_n \mid x^c_{<n}) \right] &\leq -\log \left[\frac{1}{n} \prod_{c\in\mathcal{S}} \alpha_c(\epsilon) \rho_c(x^c_{<n}) \rho_c(x_n \mid x^c_{<n}) \right] \\ &= \log n - \log \prod_{c\in\mathcal{S}} \alpha_c(\epsilon) - \log \prod_{c\in\mathcal{S}} \rho_c(x^c_{1:n}) \\ &= \log n - \log \prod_{c\in\mathcal{S}} \alpha_c(\epsilon) - \log \psi_{\mathcal{S}}(x_{1:n}), \end{aligned}$$

where the last equality follows from the definition of $\psi_{\mathcal{S}}(x_{1:n})$. Incorporating this result into Equation 1, we obtain

$$-\log \xi_{\text{SkipCTS}}(x_{1:n}) \leq [\ell(\mathcal{S}) + 1] \log n - \log \prod_{(c,l)\in\tilde{\mathcal{S}}} \beta_{c,l}(\epsilon) - \log \prod_{c\in\mathcal{S}} \alpha_c(\epsilon) - \log \psi_{\mathcal{S}}(x_{1:n})$$

and hence

$$\begin{aligned} \mathcal{R}_n(\xi_{\text{SkipCTS}}, \{\psi_{\mathcal{S}}\}) &\leq [\ell(\mathcal{S}) + 1] \log n - \log \prod_{(c,l)\in\tilde{\mathcal{S}}} \beta_{c,l}(\epsilon) - \log \prod_{c\in\mathcal{S}} \alpha_c(\epsilon) \\ &= [\ell(\mathcal{S}) + 1] \log n + \Gamma^K_D(\mathcal{S}) \end{aligned}$$

as desired.

	K = 0		K = 1			K = 2		
	Loss	Speed	Loss	Speed	Sig.	Loss	Speed	Sig.
ASTERIX	55.79	346.61	48.39	34.41	✓	38.30	7.35	✓
BEAM RIDER	94.98	296.17	86.11	34.40	✓	101.63	7.01	✓
PONG	7.78	346.79	6.83	37.96	✓	7.90	19.98	✓
Q*BERT	7.04	330.02	6.46	39.92	✓	7.18	9.71	✓
SEAQUEST	99.31	316.30	89.73	33.49	✓	78.07	6.85	✓
ALIEN	59.13	276.87	53.92	35.23	✓	62.35	6.09	✓
AMIDAR	11.84	317.90	10.75	38.60	✓	10.84	9.00	
ASSAULT	39.26	352.55	34.89	35.75	✓	34.84	10.42	
ASTEROIDS	24.25	352.44	23.99	35.47	✓	16.19	21.42	✓
ATLANTIS	21.96	273.05	19.80	30.67	✓	19.98	12.69	
BANK HEIST	78.71	282.97	77.99	30.29	✓	72.97	6.58	✓
BATTLE ZONE	193.35	240.58	179.92	28.37	✓	169.89	6.12	
BERZERK	54.76	278.57	50.80	29.41	✓	52.26	7.12	✓
BOWLING	1.46	314.82	1.43	35.90	✓	1.58	17.73	
BOXING	194.76	275.75	192.38	39.84	✓	183.68	6.66	✓
BREAKOUT	5.46	321.19	4.21	35.41	✓	5.56	18.81	✓
CARNIVAL	31.38	301.71	24.67	32.76	✓	24.17	9.78	
CENTIPEDE	72.05	291.38	66.94	29.57	✓	66.97	5.61	
CHOPPER COMMAND	181.75	281.84	179.02	26.64	✓	155.57	5.65	✓
CRAZY CLIMBER	42.61	357.73	35.18	34.54	✓	39.00	7.34	✓
DEMON ATTACK	154.61	346.35	144.81	28.64	✓	143.56	8.26	
DOUBLE DUNK	192.16	297.47	190.52	27.85	✓	205.78	5.71	✓
ELEVATOR ACTION	67.04	322.66	68.31	29.85		66.38	5.59	
ENDURO	276.54	274.30	247.83	26.53	✓	313.12	4.37	✓
FISHING DERBY	111.75	254.84	101.51	29.34	✓	129.73	5.16	✓
FREEWAY	6.71	260.11	4.32	37.14	✓	5.22	9.82	✓
FROSTBITE	55.56	295.62	52.25	33.79	✓	60.35	6.09	✓
GOPHER	23.18	358.41	19.14	36.11	✓	15.49	10.51	✓
GRAVITAR	61.14	343.60	57.16	33.48	✓	53.98	8.18	✓
H.E.R.O.	20.97	280.24	18.05	37.72	✓	19.75	7.96	✓
ICE HOCKEY	98.01	296.70	97.22	29.65	✓	80.33	5.87	✓
JAMES BOND	160.23	306.32	147.03	28.67	✓	162.47	5.28	✓
JOURNEY ESCAPE	1104.44	182.08	1085.74	17.03	✓	1106.69	1.88	✓
KANGAROO	17.11	319.69	16.52	37.63	✓	14.58	9.38	✓
KRULL	143.16	261.17	129.05	29.41	✓	154.31	4.18	✓
KUNG-FU MASTER	27.78	300.10	23.55	35.28	✓	25.38	6.70	✓
MONTEZUMA'S REVENGE	12.42	316.31	11.54	36.08	✓	12.86	7.35	✓
MS. PACMAN	33.92	321.21	31.58	34.23	✓	34.09	5.30	✓
NAME THIS GAME	54.61	301.56	45.73	32.97	✓	50.06	5.08	✓
POOYAN	21.49	303.85	19.67	35.70	✓	21.26	8.66	✓
PRIVATE EYE	95.03	290.91	83.69	31.46	✓	85.28	4.88	
RIVER RAID	83.65	283.98	74.08	32.50	✓	70.63	4.87	✓
ROAD RUNNER	101.87	295.63	98.46	32.56	✓	104.66	6.05	✓
ROBOTANK	206.08	262.14	179.70	28.84	✓	149.26	4.18	✓
SKIING	73.33	273.06	71.77	32.64	✓	57.63	6.21	✓
SPACE INVADERS	47.14	328.05	44.07	40.59	✓	45.02	5.81	
STAR GUNNER	133.17	363.04	111.79	36.55	✓	78.34	7.02	✓
TENNIS	58.36	311.96	54.09	38.42	✓	49.38	5.93	✓
TIME PILOT	173.83	334.52	163.52	30.20	✓	131.81	4.76	✓
TUTANKHAM	79.16	330.27	68.17	44.46	✓	63.74	6.97	✓
UP AND DOWN	204.05	231.64	196.29	37.32	✓	190.42	4.09	
VENTURE	24.13	349.33	22.38	51.87	✓	19.17	6.72	
VIDEO PINBALL	33.55	282.75	29.27	47.92	✓	40.83	6.48	✓
WIZARD OF WOR	24.69	357.22	23.42	51.06	✓	23.66	7.52	
YAR'S REVENGE	112.39	324.94	104.83	36.36	✓	82.30	8.14	✓

Table 1: Prediction results for 55 Atari games. *Loss* corresponds to the per frame negative $\log_2$ probability, averaged over the last 4500 frames of data of 10 trials. *Speed* corresponds to the average number of frames processed per second. *Sig.* indicates a statistically significant difference between $K \in \{0, 1\}$ and $K \in \{1, 2\}$, respectively.