Learning When to Stop Thinking and Do Something!

Barnabás Póczos POCZOS@CS.UALBETA.CA
Yasin Abbasi-Yadkori YASIN.ABBASI@GMAIL.COM
Csaba Szepesvári SZEPESVA@CS.UALBERTA.CA
Russell Greiner GREINER@CS.UALBERTA.CA
Nathan Sturtevant NATHANST@CS.UALBERTA.CA
Department of Computing Science, University of Alberta, Edmonton, AB, T6G 2E8 CANADA

Abstract

An anytime algorithm is capable of returning a response to the given task at essentially any time; typically the quality of the response improves as the time increases. Here, we consider the challenge of learning when we should terminate such algorithms on each of a sequence of iid tasks, to optimize the expected average reward per unit time. We provide a system for addressing this challenge, which combines the global optimizer Cross-Entropy method with local gradient ascent. This paper theoretically investigates how far the estimated gradient is from the true gradient, then empirically demonstrates that this system is effective by applying it to a toy problem, as well as on a real-world face detection task.

1. Introduction and Motivation

A mail sorter will scan each envelope as it passes by, trying to find and read the zip code. If it misreads the zip code, that mail item may be misdelivered, which incurs a penalty. It is also important for the sorter to be efficient, as there are many remaining mail items queued up — and spending too long on one envelope will delay the others, and possibly prevent some from being delivered. . . which is also problematic (Russell & Subramanian, 1995).

Here, it is important to be both accurate and efficient. This is true of many other situations — finding and classifying parts in a factory assembly line, finding and identifying car license plates as cars drive in a freeway, or detecting possible terrorists in airports.

In all of these situations, we assume an anytime system has already been designed for the task (*e.g.*, sorting the mail), which basically runs a series of subroutines $\langle A_1(\cdot),\ A_2(\cdot),\ \ldots\rangle$ on each instance (*e.g.*, each envelope); we assume that later subroutines generally return successively higher quality responses. This paper addresses the challenge of learning a "stopping policy": That is, after the k^{th} subroutine $A_k(X_t)$ has terminated on instance X_t, after spending time $\tau_{A_k}^{X_t} = \tau_{tk}$, the stopping policy has to decide whether to accept that algorithm's decision about X_t (that is, decide to route X_t to $A_k(X_t)$) and then proceed to process instance X_{t+1}, or instead to consider running the next subroutine A_{k+1} on this instance X_t (which means it will return the value of $A_{k+1}(X_t)$, or perhaps $A_{k+2}(X_t)\ldots$). To help make this decision, $A_k(X_t)$ returns some information that can help the stopping policy. Such information might indicate whether the envelope was handwritten or typed, information about the layout of the address field or any other information that became available when the subroutine processed the envelope.

For each instance X_t, once the system has accepted the decision of a subroutine, say, at the $L_t{}^{\text{th}}$-stage of the thinking process, the overall system (and hence the stopping policy) receives a (real-valued) *reward*, $r_t = r(\,X_t,\, A_{L_t}\,) \in \Re$, based on whether the letter X_t was routed correctly. This is after the system has spent some time "thinking" about this instance, which is the sum of the times required by the subroutines $A_1, \ldots, A_{L_t}$ on this instance, $\sum_{k=1}^{L_t} \tau_{tk}$. The time spent with the decision to stop at the k^{th}-stage is included in τ_{tk}. Also, the reward function $r = r(x, a)$ is not available for the decision maker (otherwise, the optimal action would be easy to compute, at least when the action space is not large). Our ultimate goal is to maximize the *average reward*, which is the accumu-

Appearing in *Proceedings of the 26th International Conference on Machine Learning*, Montreal, Canada, 2009. Copyright 2009 by the author(s)/owner(s).

lated reward on the envelope sequence divided by the time we spent "thinking" about the envelopes. Therefore, we have to consider both the reward we can accumulate, as well as the thinking time required.

Obviously, our problem can be viewed as finding an optimal policy in an average-reward partially observable Markovian Decision Problem (MDP) (Sutton & Barto, 1998), where the actions have different (possibly random) durations: The hidden state (in one formalization of the problem) is the optimal action $\arg\max_a r(X_t, a)$ (i.e., the zip-code), the observations are the information that is gathered while processing the instance, including the decision of the subroutines; the actions are 'stop thinking' or 'continue'; and the reward is based on whether the letter was correctly routed. (A formal description appears in Section 2). Given that the full scanned image of an envelope is available in the computer, one may as well model the problem as a *fully observable* MDP, where the state is X_t. This is promising since powerful value-function based approaches tend to work poorly in partially observable environments when state aliasing is a problem. However, by the very nature of our problem it seems that practical value-function based methods would themselves need to introduce significant state aliasing: Since we want to keep the processing times small, we cannot allow arbitrary complex procedures when computing the values of actions for some input X_t. Indeed, all standard value function computations would extract some finite (small) representation of X_t and obtain the values based on this representation. Given the complicated nature of X_t, significant state aliasing seems then unavoidable.

Therefore, in this paper we consider *policy search* approaches. More precisely, we investigate both generic policy search based on the Cross-Entropy (CE) method (Rubinstein & Kroese, 2004) and policy gradient methods (Williams, 1992; Baxter & Bartlett, 2001), and their combination, given some fixed parametric class of policies. The main problem with policy gradient methods is that the variance of the gradient estimate is hard to control. Due to the aforementioned difficulties related to the use of value-functions, we do not attempt to employ value-function based approaches to reduce the variance, which usually is essential to achieve good performance (Greensmith et al., 2004). Thanks to the special structure of our problem, we will show how to obtain reasonable estimates of the gradient. This observation is one of the main contributions of the paper. We prove a bound on the quality of the estimate of the gradient as a function of the number of instances processed and propose a stopping rule for deciding if the gradient estimate is sufficiently good. We also demonstrate that these approaches are effective using empirical studies on a simple artificial example, as well as on a real-world face detection application.

After Section 2 formally defines our problem, Section 3 provides two efficient learning algorithms, and summarizes our empirical studies to evaluate these algorithms. We close this section with a brief overview of related work.

Related work: As mentioned above, as an *anytime algorithm* (Zilberstein, 1996) is working on a problem, it can be interrupted; if so, it will then return its current best answer. Note this model assumes the *external world* will terminate the process (*e.g.*, when the current envelope is gone and a new one appears in its place); by contrast, our task is to design a policy that decides itself when to terminate. Russell and Subramanian (1995) consider "bounded rational agents" in this anytime framework. Their work focuses on the challenge of *building* this sequence of algorithms, given the relevant information about the world. Our paper, by contrast, describes a way to *learn* how to make decisions based on the available information.

Turney (2000) summarizes many different types of "cost" in the context of learning. Here, we are considering only the cost of the learned system, in terms of time required to produce an accurate response; we are not explicitly considering the cost of the learning process.

Many others have considered this challenge of learning a system that must be both efficient and accurate: Greiner et al. (2002) defines an "active classifier" as a process that can sequentially acquire information about an instance before rendering a decision (*e.g.*, performing some sequence of tests on a patient before declaring that patient to be healthy or sick); an optimal such active classifier will minimize the expected cost of collecting this information plus the expected penalty for misclassification. They observe an active classifier corresponds to an "action policy" (given the results of the previous tests, take some specific action; here this corresponds to a decision tree), and then consider (PAC)learning the optimal such policy in various settings. By contrast, our system is only *thinking* about an instance, and is not making decisions about what extra information to collect (information collection happens independently); moreover, we are not learning the optimal action policy, but instead considering the optimal "stopping policy", wrt a fixed action policy. (We note that our approach naturally extends to this case and also to the case when information collection has a price.) Moreover, their model requires knowing a model of the actions in advance; our approach does not. Finally, while their approach is basically a "batch learner", our system can continuously improve based on observations.

Optimal stopping has been extensively investigated in the statistics and economics literature (Shiryaev, 2007). Using their terminology, we are exploring a Bayesian optimal stopping problem, where the prior distribution is over the instances (the distribution of envelopes). However, the focus in this literature is the existence of optimal stopping rules (often by reducing the problem to a MDP), or obtaining closed form solutions in specific problems. By contrast, here we are trying to use a set of instances to learn a good (not necessarily optimal) stopping policy.

Finally, we note that Viola and Jones (2001) were also interested in producing an efficient face detection system by applying a sequence of successively more expensive classifiers. That system used only a single simple type of base classifier (linear separator), and had a one-sided stopping critera (stop as soon as any base-classifier says "no"); by contrast, we consider general classifiers, and allow termination on either "yes" or "no" decisions.

2. Formal Definition of the Problem

Let $X_1, X_2, \dots$ be an independent and identically distributed (iid) sequence — *e.g.*, corresponding to the series of envelopes mentioned in the previous section. On the t^{th} step, when first dealing with the instance X_t, we start a "thinking process" that can involve at most K stages. In each stage k ($1 \le k \le K$), we compute information about X_t, denoted $Y_{tk} \in \mathcal{Y}_k$, which is the result of the thinking at stage k with respect to X_t. We let τ_{tk} denote the time used in thinking at stage k. We make the Markov assumption, that the new knowledge Y_{tk} is independent of $Y_{t,1}, \dots, Y_{t,k-2}$ given $Y_{t,k-1}$ and X_t. This is not a restrictive assumption since the domain of $Y_{t,k}$ and $Y_{t,j}$ could be different for $k \neq j$. In particular, $Y_{t,k}$ may simply include $Y_{t,k-1}, \dots, Y_{t,1}$.

Let q_k be the stochastic policy that determines whether we terminate the thinking process at stage k. More precisely, at state k, after seeing Y_{tk}, q continues thinking at X_t with probability $q_k(0|Y_{tk})$, or quits with probability $q_k(1|Y_{tk}) = 1 - q_k(0|Y_{tk})$. By convention, q will terminate by the K^{th} stage if we have not terminated earlier; *i.e.*, $q_K(1|Y_{tK}) = 1$ for all $Y_{tK} \in \mathcal{Y}_K$. Let $L_t \in \{1, \dots, K\}$ denote when we quit for instance X_t, and let $T_t = \sum_{k=1}^{L_t} \tau_{tk}$ be the total thinking time required to deal with X_t.

Let $\mu_k : \mathcal{Y}_k \to \mathcal{A}$ be a fixed mapping from the result of the k^{th} information stage to the action space, determining what action we would take if we quit at stage k; hence $A_t = \mu_{L_t}(Y_{t,L_t})$ is the action taken on instance X_t. (This corresponds to sending the envelope to the destination determined by A_t.) Figure 1 summarizes the whole process.

The performance criterion for policy $q = (q_1, \dots, q_K)$ is defined as the expected average reward per time step:

$$\rho^q \quad = \quad \mathbb{E}\left[\liminf_{t\to\infty} \frac{\sum_{s=1}^t r(X_s, A_s)}{\sum_{s=1}^t T_s}\right], \qquad (1)$$

where $r : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$ is the reward function, which is assumed to be uniformly bounded. Here, ρ^q's dependence on q is defined through A_s and T_s, whose distributions are each determined by q. We also assume that T_s is bounded away from zero with probability one and is bounded from above with probability one — *i.e.*, there is a minimal and maximal thinking time.

Our challenge is to determine the q that maximizes ρ^q. We consider a learning scenario where all quantities involved (including the distribution of X_t and the transition kernels p_k and reward r) are unknown, but where we can sample trajectories and observe the rewards.

3. Learning Stopping Policies

In this section we derive two policy-gradient-based algorithms. Since $\{X_t\}$ is iid, so are $\{R_s = r(X_s, A_s)\}$ and $\{T_s\}$. As $\frac{\sum_{s=1}^t r(X_s, A_s)}{\sum_{s=1}^t T_s} = \frac{\frac{1}{t}\sum_{s=1}^t r(X_s, A_s)}{\frac{1}{t}\sum_{s=1}^t T_s}$, by the law of large numbers the numerator and the denominator converge (respectively) to $\mathbb{E}[r(X_1, A_1)]$ and $\mathbb{E}[T_1]$. This allows us to rewrite the performance criterion as

$$\rho^q \quad = \quad \frac{\mathbb{E}[r(X_1, A_1)]}{\mathbb{E}[T_1]}, \qquad (2)$$

which is much easier to optimize than the original average cost criterion (Equation 1). In particular, this is the "trick" that allows us to obtain gradient estimates relatively quickly without resorting to value functions or other variance reduction techniques. In the next sections we propose two estimates of the gradient of our criterion based on (2) assuming that the parameters of q are collected in the vector θ — *i.e.*, $q = q_\theta$. That is, our goal now is to calculate the gradient of the performance criterion ρ^{q_θ} with respect to θ. Clearly, using the notation $\mathbb{E}[r_1] = \mathbb{E}[r(X_1, A_1)]$, $\Delta\mathbb{E}[r_1] = \frac{\partial}{\partial\theta}\mathbb{E}[r(X_1, A_1)]$, $\Delta\mathbb{E}[T_1] = \frac{\partial}{\partial\theta}\mathbb{E}[T_1]$,

$$\frac{\partial}{\partial\theta}\rho^q \quad = \quad \frac{\partial}{\partial\theta}\frac{\mathbb{E}[r_1]}{\mathbb{E}[T_1]} \quad = \quad \frac{\mathbb{E}[T_1]\Delta\mathbb{E}[r_1] - \mathbb{E}[r_1]\Delta\mathbb{E}[T_1]}{\mathbb{E}[T_1]^2}.$$

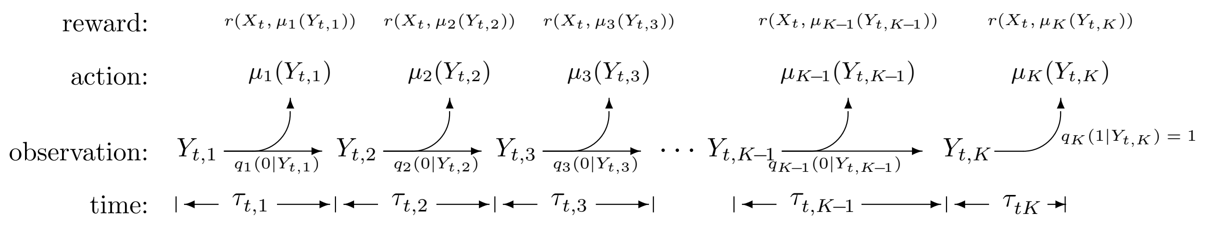

Figure 1. Summary of the Overall Process

Our gradient estimates will be based on this formula: we estimate each quantity in this formula individually and then combine the estimates.

3.1. Direct Gradient Ascent

We can estimate the expectation of both T_1 and $r(X_1, A_1)$ by using the sample means based on n instances:

$$\mathbb{E}[T_1] \approx \frac{1}{n}\sum_{t=1}^{n}\sum_{k=1}^{L_t}\tau_k \quad \mathbb{E}[r(X_1,A_1)] \approx \frac{1}{n}\sum_{t=1}^{n} R_t.$$

Now to develop an estimate to the gradient of $\mathbb{E}[r(X_1,A_1)]$ and $\mathbb{E}[T_1]$, observe that $\mathbb{E}[r(X_1,A_1)]$ is just the reward obtained in an episodic problem (the reward is received at the end of the episode). Hence, one can use the method of likelihood ratios, also known as the REINFORCE algorithm in the reinforcement learning literature, to calculate the derivative (Williams, 1992). To deal with $\mathbb{E}[T_1]$, recall that stage k of the thinking process costs τ_k. This is just the expected value of the total cost in this finite horizon, episodic problem, and hence we can use REINFORCE again to estimate the derivative.

In detail, let $(Q_{tk})_{k=1,\ldots,L_t}$ be the decision at stage k on instance t. For each $t \in \{1,\ldots,n\}$, we define the trajectories $\tilde{Y}_t = [Y_{t0}, Y_{t1}, \hat{Q}_{t1}, Y_{t2}, \hat{Q}_{t2}, \ldots, Y_{tK}, \hat{Q}_{tK}]$, where $Y_{t0} = X_t$, $\hat{Q}_{tk} = Q_{tk}$ if $k \le L_t$; otherwise, by definition, $\hat{Q}_{tk} = 1$. According to our notation, the final reward at the end of the tth trajectory is simply $R_t = r(X_t, A_t)$, which (as a slight abuse of notation) we will denote as $r(\tilde{Y}_t)$. Note that $r(\tilde{Y}_t)$ depends only on the first L_t stages of the thinking process.

Letting $f_\theta(\tilde{y})$ denote the density function of trajectory $\tilde{y}$, and $r(\tilde{y})$ the corresponding reward, we have $\mathbb{E}[r(X_t,A_t)] = \int r(\tilde{y}) f_\theta(\tilde{y})d\tilde{y}$. (When the space of trajectories is countable, $f_\theta(\tilde{y})$ should be understood as the probability of seeing $\tilde{y}$ under q_θ.)

Under mild regularity conditions,

$$\frac{\partial}{\partial\theta}\mathbb{E}[r(X_t,A_t)] = \frac{\partial}{\partial\theta}\int r(\tilde{y}) f_\theta(\tilde{y})d\tilde{y}$$
$$= \int r(\tilde{y})\frac{\partial}{\partial\theta}\left(\ln f_\theta(\tilde{y})\right) f_\theta(\tilde{y})d\tilde{y} = \mathbb{E}[r(\tilde{Y}_t)\frac{\partial}{\partial\theta}\ln f_\theta(\tilde{Y}_t)].$$

We can use the Markov property along the trajectory $\tilde{y} = (y_0, y_1, \hat{q}_1, \ldots, y_{k-1}, \hat{q}_{k-1}, y_k)$ to derive

$$\begin{aligned}\ln f_\theta(\tilde{y}) \;&=\; \ln f_0(y_0) + \sum_{k=1}^{\ell(\tilde{y})}\ln p_k(y_k|y_{k-1}) + \\ &\quad + \sum_{k=1}^{\ell(\tilde{y})-1}\ln q^\theta(0|y_k) + \ln q^\theta(1|y_{\ell(\tilde{y})}),\end{aligned}$$

where f_0 is the density underlying $Y_{t0} = X_t$, p_k is the Markov kernel underlying the kth transition and $\ell(\tilde{y}) = \min\{\, k > 0 : \hat{q}_k = 1\,\}$ is the index of the stage in $\tilde{y}$ when q_θ quits. Thus, $\frac{\partial}{\partial\theta}\ln f_\theta(\tilde{Y}_t) = \sum_{k=1}^{L_t-1}\frac{\partial}{\partial\theta}\ln q^\theta(0|Y_{tk}) + \frac{\partial}{\partial\theta}\ln q^\theta(1|Y_{tL_t})$, leading to the unbiased estimate of $\frac{\partial}{\partial\theta}\mathbb{E}[r(X_1,A_1)]$:

$$\hat{\Delta}_r = \frac{1}{n}\sum_{t=1}^{n} r(\tilde{Y}_t)s(\tilde{Y}_t),$$

where $s(\tilde{Y}_t) = \sum_{k=1}^{L_t-1}\frac{\partial}{\partial\theta}\ln q^\theta(0|Y_{tk}) + \frac{\partial}{\partial\theta}\ln q^\theta(1|Y_{tL_t})$. Similarly $\hat{\Delta}_T = \frac{1}{n}\sum_{t=1}^{n} T_t s(\tilde{Y}_t)$ is an unbiased estimate of $\frac{\partial}{\partial\theta}\mathbb{E}[T_1]$. Putting the pieces together, we see that the gradient of the performance criterion can be estimated with

$$\begin{aligned}\hat{G}_n \;&=\; \frac{1}{n}\sum_{t=1}^{n}\left(\frac{r(\tilde{Y}_t)}{\hat{T}} - \frac{\hat{r}T_t}{\hat{T}^2}\right)\times \\ &\quad\times\left(\sum_{k=1}^{L_t-1}\frac{\partial}{\partial\theta}\ln q^\theta(0|Y_{tk}) + \frac{\partial}{\partial\theta}\ln q^\theta(1|Y_{tL_t})\right),\end{aligned}$$

where $\hat{T} = \frac{1}{n}\sum_{t=1}^{n}\sum_{k=0}^{L_t}\tau_k$ is the empirical average thinking time, and $\hat{r} = \frac{1}{n}\sum_{t=1}^{n} r(\tilde{Y}_t)$ is the empirical average reward. We will refer to this algorithm as *Direct Gradient Ascent* (DGA).

This algorithm proceeds as follows: given a parameter θ, it first simulates the policy q^θ on a batch of n samples, then it calculates the estimated gradient of the parameters and updates them using $\theta_{\text{new}} := \theta + \lambda\hat{G}_n$, for some learning rate $\lambda > 0$ (the learning rate may change over time). It then collects another batch of n samples and iterates.

3.2. The Quality of the Estimated Gradient

The previous section derived the DGA estimation of the gradient. This section estimates how far this estimated gradient is from the true gradient. Fix θ and let G denote the true gradient. Observe that

$$\hat{G}_n = \frac{\hat{\Delta}_r}{\hat{T}} - \frac{\hat{r}}{\hat{T}}\frac{\hat{\Delta}_T}{\hat{T}}, \quad G = \frac{\Delta\mathbb{E}[r_1]}{\mathbb{E}[T_1]} - \frac{\mathbb{E}[r_1]}{\mathbb{E}[T_1]}\frac{\Delta\mathbb{E}[T_1]}{\mathbb{E}[T_1]}.$$

Let $\|\cdot\|$ be an arbitrary vector norm. We can use Hoeffding's Inequality (1963) to bound the chance that our empirical estimates will be far from their true means: for every $\delta \in [0,1]$ and every n, there exist ε_T, ε_r, $\varepsilon_{\Delta T}$, $\varepsilon_{\Delta r}$ such that $\mathbb{P}(|\mathbb{E}[T_1] - \hat{T}| \leq \varepsilon_T) \geq 1-\delta$, $\mathbb{P}(|\mathbb{E}[r_1] - \hat{r}| \leq \varepsilon_r) \geq 1-\delta$, $\mathbb{P}(\|\Delta\mathbb{E}[T_1] - \hat{\Delta}_T\| \leq \varepsilon_{\Delta_T}) \geq 1-\delta$ and $\mathbb{P}(\|\Delta\mathbb{E}[r_1] - \hat{\Delta}_r\| \leq \varepsilon_{\Delta_r}) \geq 1-\delta$, where $\varepsilon_T, \varepsilon_r, \varepsilon_{\Delta_T}, \varepsilon_{\Delta_r} = O(\sqrt{\log(1/\delta)/n})$. Using these quantities and applying the triangle inequality, we can bound the deviation between the true gradient G and its empirical estimation $\hat{G}_n$:

$$\|G - \hat{G}_n\| \leq \left\|\left(\frac{\mathbb{E}[r_1]}{\mathbb{E}[T_1]} - \frac{\hat{r}}{\hat{T}}\right)\frac{\Delta\mathbb{E}[T_1]}{\mathbb{E}[T_1]}\right\| + \\ + \left\|\frac{\hat{\Delta}_r}{\hat{T}} - \frac{\Delta\mathbb{E}[r_1]}{\mathbb{E}[T_1]}\right\| + \left\|\frac{\hat{r}}{\hat{T}}\left(\frac{\hat{\Delta}_T}{\hat{T}} - \frac{\Delta\mathbb{E}[T_1]}{\mathbb{E}[T_1]}\right)\right\|.$$

With some more tedious calculations (using the triangle inequality several times and $(1-x)^{-1} \leq 1+2x$ for $x \leq 1/2$), we arrive at the following result:

Proposition 1. *Assume that $n \geq 2\log(1/\delta)/\tau_0^2$, where τ_0 is an almost sure lower bound on T_1. Then with probability $1-\delta$,*

$$\|G - \hat{G}_n\| \leq c_1\sqrt{\frac{\log(4/\delta)}{n}} + c_2\frac{\log(4/\delta)}{n},$$

where c_1, c_2 are constants that depend only on the range of the rewards, thinking times and their gradients.

3.3. A Stopping Rule for Preventing Slow Convergence Near Optima

The previous section proved that, with probability $\geq 1-\delta$, $\|G - \hat{G}_n\| \leq c(\delta, n)$, where $c(\cdot,\cdot)$ can be calculated exactly, knowing the range of the rewards and the thinking times, independent of θ. We can use this to calculate a bound on n that ensures that G and $\hat{G}_n$ are sufficiently close to each other. However, knowing this is not enough for the purposes of gradient ascent: If we fix any $\varepsilon > 0$, we may have $\|G\|$ is $O(\varepsilon)$; here knowing that $\|G - \hat{G}_n\| < \varepsilon$ does not guarantee that following $\hat{G}_n$ moves the parameters in the direction of the gradient. The idea here is to let ε change depending on the length of G. In particular, if $\|G - \hat{G}_n\| \leq \frac{1}{2}\|G\|$ then the angle between G and $\hat{G}_n$ must be smaller than 90^o — *i.e.*, $\hat{G}_n^T G > 0$.

Since we do not know $\|G\|$, we must estimate it from the sample. With probability $\geq 1-\delta$, both $\max(0, \|\hat{G}_n\| - c(\delta,n)) \leq \|G\|$ and $\|G - \hat{G}_n\| \leq c(\delta,n)$ hold. Hence, when

$$c(\delta,n) \leq \frac{1}{2}\max(0, \|\hat{G}_n\| - c(\delta,n)) , \quad (3)$$

we also have $\|G - \hat{G}_n\| \leq \frac{1}{2}\|G\|$. This proves the following result:

Theorem 2. *Fix $0 < \delta < 1$ and let $n = n(\delta)$ be the first (random) time when Equation (3) holds. Then $\hat{G}_n^T G > 0$ with probability $\geq 1-\delta$.*

Further, it is possible to the bound the expected number of samples required to achieve this bound. Another possible improvement is to use an empirical Bernstein bound (taking into account the variances) instead of relying on Hoeffding's inequality, as in Mnih et al. (2008). Although these modifications might improve the efficiency, they are outside of the scope of the present work and are not explored here any further.

3.4. A Toy Problem

Our experiments are based on the situation mentioned above, where we want to sort a sequence of envelopes based on their respective zipcodes. For each envelope X_t, we will apply a fixed sequence of (up to $K = 4$) subroutines $\langle A_1, \ldots, A_K\rangle$, where each A_k requires a fixed amount of time τ_k, and returns a response Y_{tk}, *i.e.*, , an interpretation of the zip code digits and some estimate of how certain this decision is. At each stage k of processing each envelope X_t, our stopping policy q decides whether to continue onto the next stage (running A_{k+1} on this X_t), or to "quit", which means it will follow the action $A_t = \mu_k(Y_{t,k})$, which provides reward $r(X_t, A_t)$, then go on to consider the next envelope X_{t+1}.

We simulate this problem as follows: we characterize the difficulty of an envelope by $h \sim \text{Beta}(1,1)$, where $h = 1$ means very easy and $h = 0$ means very difficult. At stage k, let p_k be the probability that $y_{t,k}$ is the correct zip code; here we initialize $p_0 = h$, which improves as $p_{k+1} = \min\{p_k + 0.1,\ 1\}$ (Hence, at stage k, $p_k = \min\{h + 0.1 \times k,\ 1\}$). Reward $r(\,X_t, \mu_k(Y_{t,k})\,)$ equals to 1 with probability $f(p_k)$ where $f(\cdot)$ is an unknown function; we use $f(p) = \sqrt{p}$ in our experiments.

In each stage Y_{tk} by assumption contains p_k. The decision to quit or not will then depend on p_k, as well as the index k. We discretize the 2D space $[0,1] \times \{1,2,3,4\}$ into M grid cells and introduce the corresponding

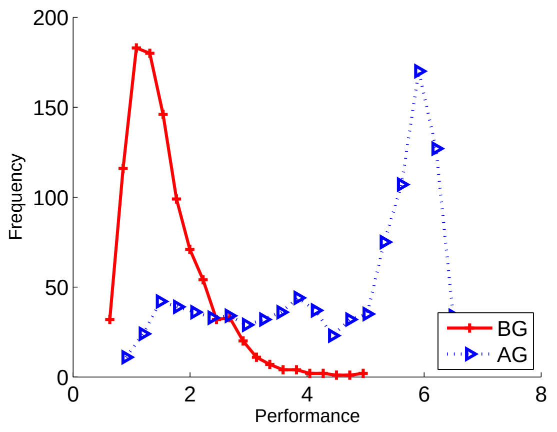

Figure 2. The performance histogram of a set of parameters before (BG) and after (AG) applying the DGA method.

$\phi(\cdot,\cdot) \in \{0,1\}^M$ feature vector, where $\phi_i(p_k,k) = 1$ if and only if (p_k,k) belongs to the i^{th} grid cell, and otherwise $\phi_j(p_k,k) = 0$, $\forall j \neq i$. In our experiments we used $M = 25$ grid cells. The stopping policy is $q^\theta(0|p_k,k) = g(\theta^T\phi(p_k,k))$, where $g(x) = 1/(1+e^{-x})$ is the sigmoid function. The gradient of this function has a simple form: $\frac{\partial}{\partial\theta}\ln q^\theta(\chi|p_k,k) = (1-\chi-q^\theta(0|p_k,k))\ \phi(p_k,k)$ for $\chi \in \{0,1\}$.

We have to wait for $\tau_0 = 0.1$ time unit before observing each X_t. The thinking time for each subsequent stage k is $0.002 + 0.2495 \times k$; note that the thinking time at stage $k = 4$ is 1.

First, we generated 1,000,000 random parameter vectors. Each vector represents a policy. We ran each policy on 100 samples (*e.g.*, envelopes) and computed the average payoff of the policy over these samples. The highest, lowest and average performances of these policies were 6.85, 0.34 and 1.60, respectively. Next, we generated 1000 initial parameters randomly and evaluated them. The performance histogram of these parameters is denoted by BG (Before Gradient) in Figure 2. For each of these parameter values, we ran our simulator for 1,000 timesteps. In each timestep, the DGA algorithm uses n =10 trajectories to approximate its gradient. (We decrease the learning rate according to $\lambda_t = 5/t$ for each run). Finally, after applying 1,000 gradient steps, we evaluated the parameters. The performance histogram of the resulting parameters is denoted by AG (After Gradient) in Figure 2. This figure shows that DGA improves the policies considerably.

3.5. Face Detection

In this section we demonstrate how our approach can be used in face detection applications. We ran our experiments on the face database (Viola & Jones, 2001), which contains 4916 pieces of facial and 7872 pieces of non-facial gray scale images of size 24×24. We also used the 22-stage hierarchical face classifier of Lienhart et al. (2003), which is an implementation of the AdaBoost algorithm (Viola & Jones, 2001), downloaded from Bradski and Kaehler (2008); we call this classification algorithm VJ. Here, each stage can classify images, and while the higher level classifiers perform better, they have higher complexity and are more costly (this is because the higher level stages require more feature evaluations). In the original version of VJ, this 22-stage classifier contained 22 parameters ($\alpha_k \in \mathbb{R}$, $k = 1,\ldots,22$). If α_k is smaller than the response of stage k (which is the weighted sum of the features on stage k), then the algorithm proceeds to stage $k+1$, where it runs a more complex classifier; otherwise it classifies the picture as non-face. If the algorithm reaches stage 22, and the response of this stage $> \alpha_{22}$, then the picture is classified as a face.

Each stage gets its input from the previous stage, and the parameter α_k is set to the value where the actual classifier recognizes 99.5% of faces, and rejects about 30% of the non-faces.

The problem with this approach is that the algorithm is not able to classify an image as a face before reaching stage 22, not allowing the process to quit earlier with a positive result. Also, the fixed 0.995 true positive rate (TPR) on each stage seems to be ad-hoc, too, and as we will see, we can optimize these parameters to achieve better results.

To handle these problems we made the following modifications. On each stage, we introduced 2 parameters $\alpha_k < \beta_k$ to help define the stopping and classification policy. Let Y_{tk} denote the response of stage k on the t^{th} instance. If $Y_{tk} < \alpha_k$ on stage k, then we quit thinking, and decide that the object is not a face. If $\alpha_k < Y_{tk} < \beta_k$, then we keep thinking and proceed on stage $k+1$. Finally, if $\beta_k < Y_{tk}$, then we quit thinking, and classify the object as a face. In practice, however, we used the soft versions of these hard decision functions: here using $q_k(0|Y_{tk}) = 1/(1+\exp(-c(Y_{tk}-\alpha_k))) \times 1/(1+\exp(-c(\beta_k - Y_{tk})))$ transition functions with $c = 50$, and a similar function for computing μ. The algorithm was forced to quit on stage 22; here the classification was based on parameter α_{22}. While our method is similar to Wald's sequential probability ratio test (1948), note that we tune the parameters α_k and β_k, we do not know the exact likelihood functions, and we apply soft decisions only.

The reward after each decision was determined by an $\mathbf{R} \in \mathbb{R}^{2\times 2}$ reward matrix. In the following experiment, we set $R_{11} = R_{22} = 100$, $R_{12} = R_{21} = 0$, corresponding respectively to true positive, true negative, false positive and false negative results.

To reduce the chance of being trapped in a local minima, we combined the gradient descent approach with the Cross-Entropy (CE) method (Rubinstein & Kroese, 2004): We generated 100 random initial parameters from some $\mathcal{N}(\theta_0, \Theta_0)$ normal distribution. We evaluated the performances of these parameters and selected the best 10 results — *i.e.*, the "elite" samples. From these elite samples we move into the direction of the respective gradients with a fixed step-size. After calculating the empirical mean and variance of these new improved samples (denoted $\bar{\theta}_0$, $\bar{\Theta}_0$) we update the internal parameters of the CE-method — $\theta_1 = \lambda\theta_0 + (1-\lambda)\bar{\theta}_0$, $\Theta_1 = \lambda\Theta_0 + (1-\lambda)\bar{\Theta}_0$ — then repeat the whole CE process from the beginning: again generating 100 new samples from distribution $\mathcal{N}(\theta_1, \Theta_1)$ and so on. Here we used 25 CE iterations, and set λ to 0.9. As this method combines CE with local search in the directions of the gradients, we call it CE-DGA.

We assumed a fixed τ_0 minimal thinking time for each instance. We tested our method using several $\mathbf{R}$ reward matrix and τ_0 minimal thinking time, and found empirically that our method always performed better than the purely randomly selected parameters. However, when τ_0 is set to a large value (40,000 in our experiments), then we found that the actual thinking time is not important, as our algorithm has to maximize only its expected reward, which is TPR+(1-FPR), where FPR is the false positive rate. Here we can compare our performance with VJ. The following results show that our proposed algorithm actually outperforms VJ here, even when the thinking time is not crucial.

For the experiments, we selected the first $1,000$ facial and $1,000$ non-facial images from the VJ database. Table 1 shows the empirical average reward, the average number of stages needed for classification, as well as the TPR and FPR values for the VJ algorithm, and for the parameters optimized by the proposed CE-DGA algorithm. It also shows the average performance values of 100 randomly selected parameters. It is interesting that CE-DGA could achieve higher expected reward, higher TPR, and smaller FPR than the VJ parameters, while using many fewer classification stages (6.1 instead of 13.17 on average)! Figure 3 shows the performances of several randomly selected parameters, as well as the performances of the VJ and the CE-DGA methods, on the "ROC domain" — *i.e.*, the True Positive Rate × False Positive Rate coordinate system.

Table 1. Expected reward, Expected stage numbers, True Positive Rate (TPR), False Positive Rate (FPR) for the Viola-Jones parameters (VJ), random parameters, and our method, respectively.

	VJ	Random	CE-DGA
$\mathbb{E}[R]$	96.95	76.80	97.70
$\mathbb{E}[\text{STAGE}]$	13.17	1.25	6.1
TPR	96.70%	99.20%	97.00%
FPR	2.80%	45.60%	1.60%

One could use CE only, *i.e.*, without local search in every iteration. Nonetheless, in Figure 4 we demonstrate that using the gradients, too, we can achieve better performance in the first few CE iterations. Later, as the iterations proceed, however, the objective becomes flat, and the gradient can not help anymore. To have a fair comparison between CE and CE-DGA we allowed the CE method to sample in each iteration as many samples as the CE-DGA evaluated in that iteration.

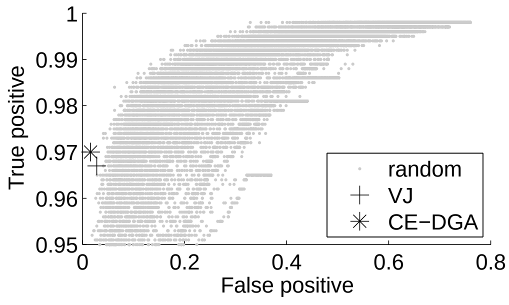

Figure 3. The ROC domain. The performances of randomly drawn parameters, Viola-Jones parameters (VJ), and parameters optimized by the CE-DGA method, respectively, shown in the True Positive Rate × False Positive Rate coordinate system.

4. Discussion and Conclusion

This article considered only the trade-off between the accuracy of a performance system versus its computational *time* cost. It is straightforward to generalize the approach when other costs are also incurred (*e.g.*, material costs in a physical process) while processing the instances. We considered learning the best parameters θ^* only for the stopping policy $q^\theta = (q_1^\theta, \ldots, q_K^\theta)$, which governs the time allowed to process the input instance. This was only to simplify the presentation of the task: the techniques developed here can also be

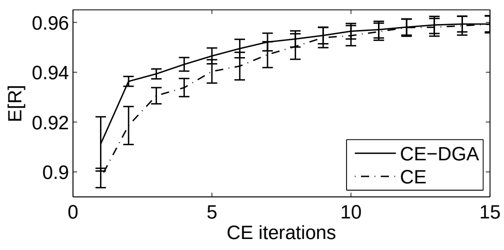

Figure 4. CE vs CE-DGA. With the combination of gradient ascent and CE (CE-DGA), in the first few iterations we can achieve higher performance than using CE only. The curves are averaged for 10 experiments. The error bars indicate the standard deviations.

used to optimize the "action policy" — to determine both the final decision policy $\mu^\theta = \langle \mu_1, \ldots, \mu_k \rangle$ and the parameters of the actual sequence of "subroutines" $\langle A_i \rangle$ being followed.

Contributions: This article investigates when to stop thinking about specific tasks, when we have to consider both the quality of our solution to the task and also the cost of thinking. We proposed an algorithm for this problem. To reduce the chance of being caught in local minima, we extended the gradient ascent method with Cross-Entropy optimization. We theoretically examined how far the estimated gradient can be from the true gradient, and provided experiments on a real-world face detection task. We found that in this task our proposed CE-DGA method could achieve higher true positive rate and smaller false positive rate than the VJ with its standard weights, while using many fewer classification stages.

We also note that our gradient estimates can be biased and hence can indeed point into the "wrong" direction. We therefore provide a stopping rule (Theorem 2) that guarantees (with high probability) that the estimated gradient does not point into the wrong direction.

Acknowledgements

The motivation for this paper comes from a discussion with Matt Ginsberg. We gratefully acknowledge support from the Alberta Ingenuity Centre for Machine Learning, the Alberta Ingenuity Fund, iCORE and NSERC. Csaba Szepesvári is on leave from MTA SZTAKI, Bp. Hungary.

References

Baxter, J., & Bartlett, P. (2001). Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, *15*, 319–350.

Bradski, G., & Kaehler, A. (2008). *Learning OpenCV: Computer vision with the OpenCV library.* O'Reilly Press.

Greensmith, E., Bartlett, P., & Baxter, J. (2004). Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, *5*, 1471–1530.

Greiner, R., Grove, A., & Roth, D. (2002). Learning cost-sensitive active classifiers. *Artificial Intelligence*, *139*, 137–174.

Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, *58*, 13–30.

Lienhart, R., Kuranov, A., & Pisarevsky, V. (2003). Empirical analysis of detection cascades of boosted classifiers for rapid object detection. *DAGM'03, 25th Pattern Recognition Symposium* (pp. 297–304).

Mnih, V., Szepesvari, C., & Audibert, J.-Y. (2008). Empirical Bernstein stopping. *International Conference on Machine learning* (pp. 672–679).

Rubinstein, R. Y., & Kroese, D. P. (2004). *The cross-entropy method.* Information Science and Statistics. Springer.

Russell, S., & Subramanian, D. (1995). Provably bounded-optimal agents. *Journal of Artificial Intelligence Research*, *2*, 575–609.

Shiryaev, A. (2007). *Optimal stopping rules.* Springer.

Sutton, R., & Barto, A. (1998). *Reinforcement learning: An introduction.* MIT Press.

Turney, P. (2000). Types of cost in inductive concept learning. *Workshop on Cost-Sensitive Learning (International Conference on Machine Learning).*

Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. *Computer Vision and Pattern Recognition* (pp. 511–518).

Wald, A., & Wolfowitz, J. (1948). Optimum character of the sequential probability ratio test. *Annals of Mathematical Statistics*, *19*, 326–339.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, *8*, 229–256.

Zilberstein, S. (1996). Using anytime algorithms in intelligent systems. *AI magazine*, *17*, 73–83.