SUBMODULAR OPTIMZATION VIA REINFORCEMENT LEARNING

Rob Cornish & Philip H. S. Torr
Department of Engineering Science
University of Oxford

ABSTRACT

We propose a method for learning how to solve classes of submodular maximisation problems. We give a precise formulation of this task in terms of maximising the expected performance of our optimiser over the domain interest, and then show how we can treat this as a problem in reinforcement learning. Our method has shown some preliminary empirical success when applied to small synthetic sensor placement problems.

1 INTRODUCTION

Recent years have seen machine learning systems surpass human-level performance in a broad array of tasks ranging from image recognition to game playing. This progress is especially attributable to the advent of novel deep architectures, which have been proven able to identify complex and far-reaching patterns within extremely large datasets. It is natural to consider other applications that may benefit from this improved capability.

In this paper, we focus on the task of maximising a monotone submodular function subject to a cardinality constraint, which has been of significant interest in the literature since Nemhauser et al. (1978). A set function $g : 2^V \to \mathbb{R}$ is *submodular* if it satisfies the following *diminishing returns* property: whenever $A \subseteq B \subseteq V$, we have

$$g(A \cup \{x\}) - g(A) \geq g(B \cup \{x\}) - g(B)$$

for all $x \in V \backslash B$. We moreover say that g is *monotone* if

$$g(A) \leq g(B)$$

whenever $A \subseteq B$. Within this context, our goal is to find a set $A^* \subseteq V$ with at most ℓ elements that maximizes g; that is, we want

$$A^* = \underset{A \subseteq V : |A| \leq \ell}{\operatorname{argmax}} \; g(A). \tag{1}$$

This problem arises naturally in a wide variety of contexts ranging from image segmentation (Boykov & Jolly, 2001) to document summarization (Lin & Bilmes, 2011).

It is NP-hard to compute a solution to (1) in general (Nemhauser & Wolsey, 1978), and so we must resort to an approximation. One such technique is given by the *greedy algorithm*, which starts with an empty set $A_0 = \emptyset$ and at the ith iteration adds to A_i an element $x_i \in V \backslash A_i$ that maximises the *marginal gain* $g(A_i \cup \{x\}) - g(A_i)$. It has been shown that the greedy algorithm produces a solution whose objective value is bounded below by $(1 - 1/e)g(A^*)$, and moreover that this is the optimal worst-case bound unless $P = NP$. (Nemhauser & Wolsey, 1978). And while the greedy algorithm is somewhat limited by its $O(\ell\,|V|)$ complexity, techniques such as Minoux (1978) and Mirzasoleiman et al. (2014) have been proposed that significantly reduce runtime in practice.

This is a somewhat intimidating state of affairs if we wish to improve the state of the art for submodular maximisation. However, we perceive a direction forward in the fact that the problem formulation given above hides domain-specific information for the sake of genericity. For most applications, g is not a black box, and V is not an haphazard array of points; instead, the two often interact in a complex but highly structured fashion that may be useful to exploit when we to compute (1). Moreover, it is likely not the case that all specific instances of (1) are of equal importance to us. Instead,

we expect to see a *distribution* of submodular problems of interest, and it would be desirable for an optimisation algorithm to be designed with this in mind. We might be perfectly willing to reduce our worst-case performance if this means an improvement on the most likely problems we will observe in practice, for instance.

To expand on these points, we now give a very simple example that we will refer back to throughout this paper. A canonical submodular optimization problem involves optimal *sensor placement*; we consider a *1-D sensor placement problem* here. Specifically, we take $V = \{x_1, \ldots, x_n\} \subseteq \mathbb{R}$ to represent positions at which we may choose to place a (unique) sensor. We have ℓ sensors in total, each of which has a fixed coverage radius $\delta > 0$. Our goal is to place our sensors in such a way as to maximise their total coverage, which is given by

$$g(\{x_1, \ldots, x_k\}) := \lambda \left(\bigcup_{i=1}^{k} [x_i - \delta, x_i + \delta] \right) \tag{2}$$

for $k \leq \ell$, where λ the 1-D Lebesgue measure and by $B_\delta(x_i)$ a ball of radius δ around x_i. It is easy to check that g is submodular and monotone, so that this problem exactly corresponds to (1) above.

Consider 1-D sensor placement when $\ell = 2$. It is fairly easy to see that if we ensure $x_1 \leq \cdots \leq x_n$ then $\{x_1, x_n\}$ maximises g. To see why, note that, for this problem, (1) is equivalent to choosing $x_i < x_j$ to minimise their pairwise overlap

$$\mu(x_i, x_j) := \min(x_i - x_j + 2\delta, 0).$$

But now since $x_1 \geq x_i$ and $x_j \leq x_n$, we will have

$$\mu(x_i, x_j) \geq \mu(x_1, x_j) \geq \mu(x_1, x_n),$$

giving $\{x_1, x_n\}$ as our minimiser as desired. Moreover, if V is such that $x_n - x_1 < 2\delta$, which guarantees that all sensor configurations would involve some overlap, then $\{x_1, x_n\}$ is strictly optimal. However, the greedy algorithm cannot know this: for it, all the x_i's are equally appealing as its first choice, since each generates the same marginal gain of 2δ. It is only by enriching our algorithm to consider domain-specific structure – such as the total ordering of V here, for example – that we may hope to avoid this sort of problem. [1]

Identifying useful structure is difficult and time-consuming; as such, we propose a method for *learning* to optimise new classes of problems automatically. We begin by formalising precisely what we mean by this, and then show how our task exactly corresponds to a reinforcement learning problem. We then approach this problem using DQN (Mnih et al., 2013), showing empirically that our algorithm learns to solve 1-D sensor placement problems more accurately and faster than the greedy algorithm for small problem instances.

2 Related work

There has been some recent interest in learning how to optimise functions defined on a continuous domain, with efforts mostly focussed on learning a good update rule for a gradient descent algorithm. Andrychowicz et al. (2016) obtain this update rule by learning a recurrent neural network which takes as input the history of objective values and gradients observed on the optimisation trajectory thus far. Li & Malik (2016) formulate the problem as a Markov decision process, policies for which correspond directly to update rules in the original problem.

Much of the recent effort in submodular maximization specifically has focused on approximating the greedy algorithm in order to avoid its $O(\ell|V|)$ complexity. Mirzasoleiman et al. (2014), for example, propose an algorithm that computes only a subsample of the possible marginal gains at each step of the greedy algorithm, which they show is still able to achieve a $(1 - 1/e - \epsilon)$ guarantee in expectation to the optimal objective value.

inline]Need to expand on

[1] This example is too trivial for the actual paper - certainly we should be more informal here if we use this exact example. However, I believe it would be possible to give an exact algorithm for maximisation in the general case that $\ell \geq 2$; possibly this will have some form such as "greedy on the outermost empty sensor position", in which case this formalism might be a little useful. I think it is potentially compelling to have an explicit example of how we may use problem structure to improve performance.

3 METHOD

Our plan is to learn how to exploit domain-specific structure to improve performance over particular classes of submodular problems. We choose to delineate these classes by considering (1) to be parameterised by the ground set V, with g viewed as a fixed operator applied to specific problem instances. This works fairly naturally for applications: in 1-D sensor placement, for example, the definition of g in (2) has the same form despite any specific numerical values of $x_1, \ldots, x_n$. We now formulate this precisely, and then show how we can approach our problem using methods from reinforcement learning.

3.1 FORMULATION

We will denote by $\mathfrak{F}(A)$ the set of all finite subsets of A. Our problem then has the following components:

1. A fixed set Σ of all possible data, which may be infinite
2. An objective function $g : \mathfrak{F}(\Sigma) \to \mathbb{R}$ that is monotone and submodular when its domain is restricted to any member of $\mathfrak{F}(\Sigma)$
3. A random ground set V chosen according to some distribution $\mathbb{P}$ on $\mathfrak{F}(\Sigma)$
4. A fixed $\ell \in \mathbb{N}$ corresponding to the maximum number of elements of V that we are allowed to select

Note that since g is monotone and since our goal is maximisation, we may assume for convenience that $g(\emptyset) = 0$ without any loss of generality.

In 1-D sensor placement, we would take $\Sigma = \mathbb{R}$ and g as defined by (2) for arbitrary $n \in \mathbb{N}$. For fixed n, we could also define $\mathbb{P}$ by sampling $x_1, \ldots, x_n$ independently from some real distribution, for example.

In this context, we define an *optimiser* to be a mapping $\omega : \mathfrak{F}(\Sigma) \to \mathfrak{F}(\Sigma)$ satisfying

$$\omega(V) \in \mathfrak{F}(V) \text{ and } |\omega(V)| < \ell$$

for all $V \in \mathfrak{F}(\Sigma)$. Intuitively, an ω may be thought of as a black box which produces an estimate

$$\omega(V) \approx \operatorname*{argmax}_{A \subseteq V : |A| \leq \ell} g(A)$$

for arbitrary $V \in \mathfrak{F}(\Sigma)$. We may then specify our problem as that of finding the best possible ω that we are able. Ideally we would take ω^* defined by

$$\omega^*(V) := \operatorname*{argmax}_{A \subseteq V : |A| \leq \ell} g(A),$$

but in general we do not expect this to be achievable. Consequently, we must aim for a *good* ω rather than the best one, which this requires us to define some performance metric for comparing optimisers. Similarly to Andrychowicz et al. (2016), we posit that a natural such metric is given by

$$\Theta(\omega) := \mathop{\mathbb{E}}_{V \sim \mathbb{P}} \left[g(\omega(V)) \right]. \tag{3}$$

Some justification for this is given by the fact that if $g(\omega_1(V)) \leq g(\omega_2(V))$ for all $V \in \mathfrak{F}(\Sigma)$ then $\Theta(\omega_1) \leq \Theta(\omega_2)$, so in particular

$$\omega^* = \operatorname*{argmax}_{\omega} \Theta(\omega).$$

3.2 CHARACTERISATION AS A MARKOV DECISION PROCESS

We approach the task of optimizing (3) via reinforcement learning. In order to do this, we now characterise the problem as a Markov decision process (MDP). We consider an MDP defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \mathcal{I})$, where

- $\mathcal{S}$ is our *state space*

- $\mathcal{A}$ maps a state $s \in \mathcal{S}$ to a set $\mathcal{A}(s)$ of possible *actions* when in s
- $\mathcal{R}$ maps an $s \in \mathcal{S}$ and an $a \in \mathcal{A}(s)$ to the *reward* $\mathcal{R}(s) \in \mathbb{R}$ obtained by taking action a when in state s
- $\mathcal{T}$ maps a state $s \in \mathcal{S}$ and an action $a \in \mathcal{A}(s)$ to some $\mathcal{T}(s, a) \in \mathcal{S}$, characterising the *transitions* made by our MDP
- $\mathcal{I}$ is a distribution on our *initial state* $s_0 \in \mathcal{S}$

A *policy* π is a map taking $s \in \mathcal{S}$ to $\mathcal{A}(s)$. Given a particular π, our MDP evolves by first sampling an initial state s_0 according to $\mathcal{I}$, and then transitions according to

$$s_{t+1} := \mathcal{T}(s_t, \pi(s_t))$$

at each timestep $t \geq 1$. We consider only deterministic $\mathcal{T}$ and π which means that each s_t is a function of π and s_0, allowing us to write

$$s_t \equiv s_t(\pi, s_0).$$

Furthermore, for reasons that will be made clear below, we impose a *finite horizon* of ℓ steps on our MDP, so that we do not consider states beyond s_ℓ.

We will denote by $\mathcal{G}(\pi, s_0)$ the total reward received by an agent following a policy π after observing an initial state s_0:

$$\mathcal{G}(\pi, s_0) := \sum_{t=0}^{\ell-1} \mathcal{R}(s_t(\pi, s_0)). \tag{4}$$

The goal in reinforcement learning is then to find an optimal policy π^*, defined by

$$\pi^* = \operatorname*{argmax}_{\pi} \mathop{\mathbb{E}}_{s_0 \sim \mathcal{I}} \mathcal{G}(\pi, s_0). \tag{5}$$

Various approaches have been proposed for solving this problem; we use a deep Q-network (DQN) as described in Mnih et al. (2013) that is trained to predict action-values by minimising a sequence of loss functions that changes at each iteration.

We now show how our problem stated in Section 3.1 can be formulated within this framework. First, we choose our state space

$$\mathcal{S} := \{(V, A) \mid V \in \mathfrak{F}(\Sigma), A \in \mathfrak{F}(V), |A| < \ell\}.$$

Intuitively, V gives the ground set corresponding to the submodular maximization problem that we are attempting to solve, and A gives the current subset of V that we have selected as a candidate for our maximizer so far. For notational convenience, we denote the ground set and chosen set of a given state s by $V(s)$ and $A(s)$ respectively.

Next, we define our action space by

$$\mathcal{A}(s) := V(s)$$

for each $s \in \mathcal{S}$; our transition function by

$$\mathcal{T}(s, a) := (V(s), A(s) \cup \{a\});$$

and the corresponding reward by

$$\mathcal{R}(s, a) := g(A(s) \cup \{a\}) - g(A(s)).$$

Otherwise stated, taking an action adds an element to our chosen set A and yields a reward of the corresponding marginal gain. Finally, we obtain our initial state distribution $\mathcal{I}$ by setting

$$s_0 := (V, \emptyset)$$

with $V \sim \mathbb{P}$. Note that s_0 is essentially just V, so for convenience we will write $s_0 \equiv V$ and $\mathcal{I} \equiv \mathbb{P}$.

This construction has several nice implications for our task. Every policy π corresponds to an optimiser ω_π defined as

$$\omega_\pi(V) := A(s_\ell(\pi, V). \tag{6}$$

Moreover, every optimiser ω corresponds to a policy π_ω obtained by, for each V, fixing an ordering

$$\omega(V) = \{x_1(V), \ldots, x_\ell(V)\}$$

(where we allow $x_i(V) = x_j(V)$ with $i \neq j$ to account for $|\omega(V)| < \ell$), and then defining

$$\pi_\omega(V, \{x_1(V), \ldots, x_k(V)\}) := x_{k+1}(V) \tag{7}$$

for $k < \ell$. These considerations entail the following:

Proposition 3.1. *If*

$$\mathop{\mathbb{E}}_{V\sim\mathbb{P}} \mathcal{G}(\pi_1, V) \leq \mathop{\mathbb{E}}_{V\sim\mathbb{P}} \mathcal{G}(\pi_2, V),$$

then $\Theta(\omega_{\pi_1}) \leq \Theta(\omega_{\pi_2})$. *Moreover,* $\Theta(\omega_{\pi^*}) = \Theta(\omega^*)$.

Proof. Our choice of $\mathcal{R}$ means the summation in (4) telescopes to

$$\mathcal{G}(\pi, V) = g(A(s_\ell(\pi, V))) - g(\emptyset) = g(A(s_\ell(\pi, V))),$$

since we assume $g(\emptyset) = 0$. This means that

$$\mathcal{G}(\pi, V) = g(\omega_\pi(V)),$$

from which we obtain the first part of the proposition by taking the expectation of both sides with respect to $V \sim \mathbb{P}$. For the second part, we also observe that

$$g(\omega(V)) = \mathcal{G}(\pi_\omega, V)$$

for any optimiser ω. But then

$$\Theta(\omega^*) = \mathop{\mathbb{E}}_{V\sim\mathbb{P}} \mathcal{G}(\pi_{\omega^*}, V) \leq \mathop{\mathbb{E}}_{V\sim\mathbb{P}} \mathcal{G}(\pi^*, V) = \Theta(\omega_{\pi^*}) \leq \Theta(\omega^*)$$

by the optimality of π^* and ω^*. □

This result shows that it makes sense to optimise (3) by optimising (5): better policies π correspond directly to better optimisers ω, and we do not preclude the possibility of achieving the optimal performance $\Theta(\omega^*)$.

3.3 Implementation

At present, we assume that $\Sigma \subseteq \mathbb{R}^M$ and V has a fixed size N with probability 1. This enables us to represent a state s by a fixed-dimensional vector

$$(x_1, \ldots, x_N, y_1, \ldots, y_N) \in \mathbb{R}^{NM+N},$$

where $V(s) = \{x_1, \ldots, x_N\}$, and each $y_i = 1$ if $x_i \in A(s)$ and -1 otherwise.

As in Mnih et al. (2013), our Q-network takes as input a state representation and outputs a Q-value for each possible action, where we understand that component i of the output corresponds to the action $x_i \in \mathcal{A}(s)$. The network consists of three fully-connected hidden layers each with 128 hidden units and ReLU activations, and a fully-connected output layer with linear activation.

We have also found a significant performance boost results if we ensure that

$$x_1 < \cdots < x_N,$$

which entails that each state representation used by our system is unique. We believe this occurs because our Q-network must otherwise learn to map all $N!$ equivalent state representations of s to the same Q-values, which quickly becomes a very tall order as N grows large.

Our remaining implementation details are as follows: we learn our Q-network using Adam (Kingma & Ba, 2014) with a learning rate of 10^{-5}; we anneal ϵ linearly 1 to 10^{-4} over the course of 10^5 iterations; and our experience replay has a memory size of 3000 timesteps.

4 Experimental results

We have so far applied our method to small instances of the 1-D sensor placement problem, although more extensive and larger-scale testing is currently in progress. We obtain $V = \{x_1, \ldots, x_N\}$ by sampling each x_i from an independent Gaussian distribution with mean 0 and standard deviation 1, for N fixed. To evaluate our performance, every 100 iterations we sample a new V and compute $g(\omega_{\pi_t}(V))$, where π_t denote the greedy policy (and *not* the ϵ-greedy policy) corresponding to our Q-network at the current step t. We then compare this value against $g(\omega_g(V))$, where ω_g denotes the output of the greedy algorithm.

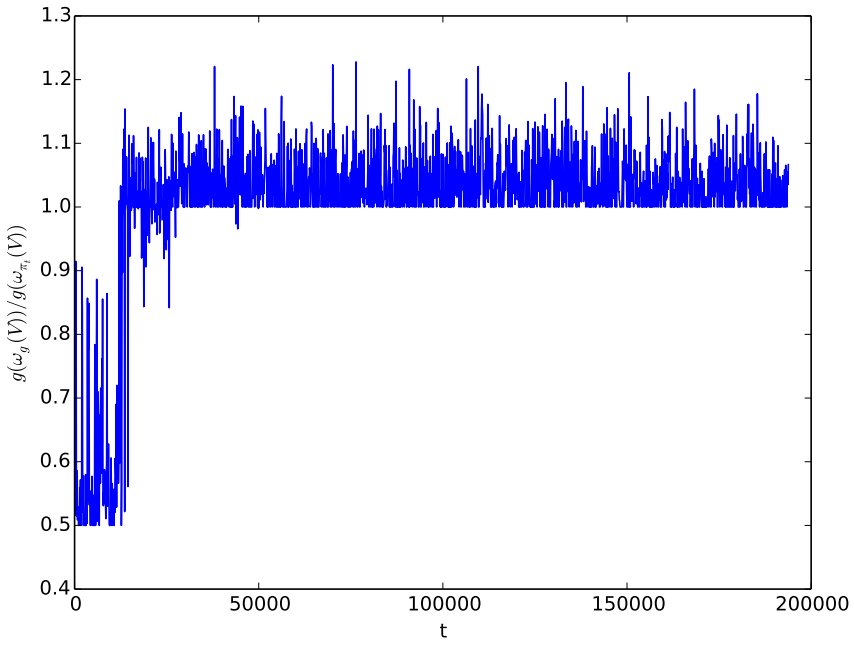

Figure 1: Relative performance of learned optimiser compared with greedy algorithm for $\ell = 2$ and $N = 40$

Figure 1 shows the result of a typical run obtained when $\ell = 2$ and $N = 40$. We observe that our performance stabilises after approximately $30,000$ iterations, after which point we do consistently better than the greedy algorithm. This occurs due to the low variance of the elements $x_i \in V$, which means that the greedy algorithm falls into the sort of trap described in Section 1. We also see in Figure 2 how our algorithm eventually learns to predict the true optimiser of g, which we compute manually for this case.

Likewise, Figure 3 shows the result of a typical run obtained when $\ell = 20$ and $N = 80$. In this case, since ℓ is large and the standard deviation of each x_i is still small, the greedy algorithm has a very high probability of producing a true optimiser, so that we cannot expect surpass its performance here. However, we do converge to match its performance after approximately $200,000$ iterations, albeit with some more instability than in the previous experiment. One possible remedy for this behaviour is to use a separate target network for performing Q-learning updates as described in Mnih et al. (2015); our implementation does not currently include this feature.

5 Conclusion and Future Work

We have presented a method for learning how to optimise submodular functions, which we believe may hold some promise based on our preliminary results. Our main focus now is on scaling our implementation to work on larger problems. We particularly aim to surpass the performance of the stochastic greedy algorithm in Mirzasoleiman et al. (2014) on the "80 million tiny images" dataset (Torralba et al., 2008), which we believe is possible given the abundance of training data available in this case, and the effectiveness of neural networks at processing image data in general.

Finally, we see an interesting direction in the fact that our formulation in Sections 3.1 and 3.2 never actually makes use of the fact that g is submodular. Consequently, it should be possible to apply these same techniques to solve (1) in the case that g is simply a real-valued set function, which may open the door to further application domains of interest.

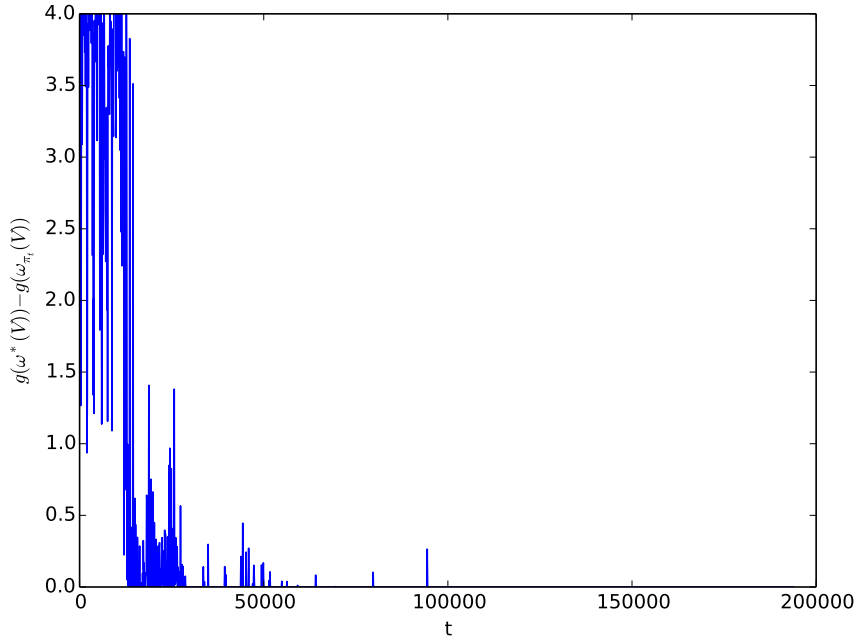

Figure 2: Error $g(\omega^*(V)) - g(\omega_{\pi_t}(V))$ in result of learned optimiser compared with true optimiser for $\ell = 2$ and $N = 40$

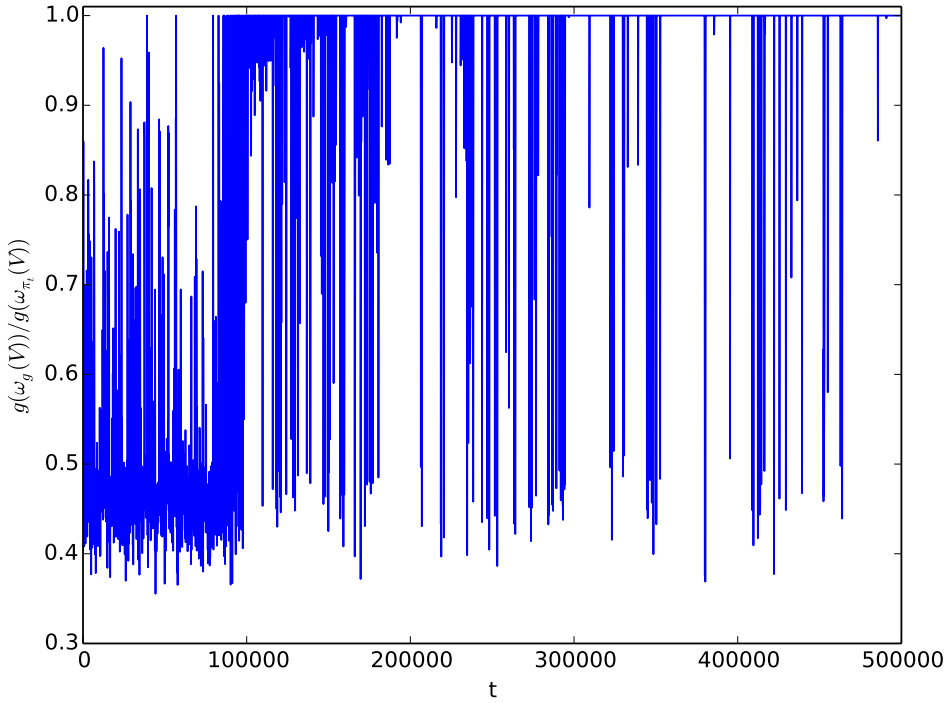

Figure 3: Relative performance of learned optimiser compared with greedy algorithm for $\ell = 20$ and $N = 80$

REFERENCES

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. *arXiv preprint arXiv:1606.04474*, 2016.

Yuri Y Boykov and M-P Jolly. Interactive graph cuts for optimal boundary & region segmentation of objects in nd images. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 1, pp. 105–112. IEEE, 2001.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Ke Li and Jitendra Malik. Learning to optimize. *arXiv preprint arXiv:1606.01885*, 2016.

Hui Lin and Jeff Bilmes. A class of submodular functions for document summarization. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pp. 510–520. Association for Computational Linguistics, 2011.

Michel Minoux. Accelerated greedy algorithms for maximizing submodular set functions. In *Optimization Techniques*, pp. 234–243. Springer, 1978.

Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, Amin Karbasi, Jan Vondrak, and Andreas Krause. Lazier than lazy greedy. *arXiv preprint arXiv:1409.7938*, 2014.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

George L Nemhauser and Leonard A Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Mathematics of operations research*, 3(3):177–188, 1978.

George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functionsi. *Mathematical Programming*, 14(1):265–294, 1978.

Antonio Torralba, Rob Fergus, and William T Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE transactions on pattern analysis and machine intelligence*, 30(11):1958–1970, 2008.