

# Classification with Costly Features using Deep Reinforcement Learning

Jaromír Janisch

Czech Technical University in Prague  
Prague, Czech Republic  
jaromir.janisch@fel.cvut.cz

Tomáš Pevný

Czech Technical University in Prague  
Prague, Czech Republic  
tomas.pevny@fel.cvut.cz

Viliam Lisý

Czech Technical University in Prague  
Prague, Czech Republic  
viliam.lisy@fel.cvut.cz

## ABSTRACT

We study a classification problem where each feature can be acquired for a cost and the goal is to optimize the trade-off between classification precision and the total feature cost. We frame the problem as a sequential decision-making problem, where we classify one sample in each episode. At each step, an agent can use values of acquired features to decide whether to purchase another one or whether to classify the sample. We use vanilla Double Deep Q-learning, a standard reinforcement learning technique, to find a classification policy. We show that this generic approach outperforms Adapt-Gbrt, currently the best-performing algorithm developed specifically for classification with costly features.

## KEYWORDS

Costly Features; Reinforcement Learning; Deep Q-learning; Adaptive Feature Acquisition; Datum-Wise Classification; Prediction on Budget

## 1 INTRODUCTION

Classification with costly features is a problem where an algorithm has to classify a sample, but can only reveal its features at a defined cost. Each sample is treated independently, and for each sample the algorithm sequentially selects features, while it can base its decision which one to acquire on values already revealed. Inherently, for different samples, a different subset of features can be selected. The algorithm has to make a trade-off between the classification precision and the total incurred cost, and the goal is to reach the highest precision and lowest total cost possible.

Classification with costly features has been studied for over 25 years [17]. It is a practical problem with many real use cases. Let's name a network security analysis, in which an expensive human operator examines network traffic, queries different analytic services and makes a decision whether the current sample was malicious or benign. Another example is a medical diagnosis, where a doctor can make a set of examinations to decide the correct treatment or a robot that can use a set of sensors to localize itself, but each measurement depletes its battery a little.

The problem is a sequential decision-making problem, and as such, it can be naturally tackled by Reinforcement Learning (RL) algorithms. We view the classification process of each sample as an episode, where an agent sequentially decides whether to acquire another feature and which, or whether it has enough information to classify the sample. At each step, the agent can base its decision on features acquired so far. For the actions requesting a feature the

agent receives a negative reward, equal to the feature cost and for the classification action, the reward is based on the true class of the sample.

We build upon prior work [2] that uses RL Q-learning for this type of problem, but it uses only a linear regression to approximate the Q function. We replace the linear regression with recent advances in deep learning and base our method on Deep Q-network [12] (DQN) with Double Q-learning [4]. Although the DQN algorithm can be extended with multiple techniques [5], we show that even basic Double DQN algorithm can outperform state-of-the-art techniques, that have been specifically tailored to this domain. Using a standard algorithm also brings all the advantages of wide community and rapid progress. Also, the algorithm can adapt to changes in non-stationary environments, commonly present in real-world problems.

We compare to the current state-of-the-art algorithm, Adapt-Gbrt [14], the latest algorithm in a sequence of problem-specific algorithms for classification with costly features [8, 15, 16, 24, 25]. We demonstrate on several datasets that our algorithm achieves the top performance.

The main contribution of this paper is that we revisit [2], replace the linear regression with DQN, a recently introduced and successful Deep RL technique, and show that it outperforms the state-of-the-art algorithms on several datasets.

## 2 METHOD

In this section, we define the problem, explain how Deep Q-learning works, describe our algorithm and analyze it theoretically.

### 2.1 Problem Definition

We view the problem as a sequential decision-making problem, where at each step, an agent selects a feature to view or makes a class prediction. We use standard reinforcement learning setting, where an agent explores its environment through actions and observes rewards and states. We represent this environment as a partially observable Markov decision process (POMDP), where each episode corresponds to a classification of one sample from a dataset. This POMDP is defined by its state space  $\tilde{S}$ , set of actions  $\mathcal{A}$ , reward function  $r(\tilde{s}, a)$  and transition function  $t(\tilde{s}, a)$ , as described below.

Let  $(x, y) \in \mathcal{D}$  be a sample drawn from a data distribution  $\mathcal{D}$ . Vector  $x \in \mathcal{X} \subseteq \mathbb{R}^n$  contains feature values, where  $x^{(i)}$  is a value of feature  $f_i \in \mathcal{F} = \{f_1, \dots, f_n\}$  and  $n$  is the number of features, and  $y \in \mathcal{Y}$  is a class. Let  $c : \mathcal{F} \rightarrow \mathbb{R}$  be a function mapping a feature  $f$  into its real-valued cost  $c(f)$ .

States  $\tilde{s} = (x, y, \tilde{\mathcal{F}}) \in \tilde{\mathcal{S}} = \mathcal{X} \times \mathcal{Y} \times \wp(\mathcal{F})$ , where  $\wp(\mathcal{F})$  is a power set<sup>1</sup> of  $\mathcal{F}$ , represents a sample  $(x, y)$  and currently selected set of features  $\tilde{\mathcal{F}}$ . The agent is given only an observed state  $s$ , which consist of only the selected parts of  $x$ , without the label. The observed state  $s \in \mathcal{S}$  is a set of pairs  $(x^{(i)}, f_i)$ , for every selected feature  $f_i$ :  $s = \{(x^{(i)}, f_i) \mid \forall f_i \in \tilde{\mathcal{F}}\}$ .

Action  $a \in \mathcal{A} = \mathcal{A}_c \cup \mathcal{A}_f$  is one of the possible classification actions  $\mathcal{A}_c = \mathcal{Y}$ , or feature selecting actions  $\mathcal{A}_f = \mathcal{F}$ . Classification actions  $\mathcal{A}_c$  terminate the episode and the agent receives a reward of 0 in case of correct classification, or  $-1$  otherwise. Feature selecting actions  $\mathcal{A}_f$  reveal a corresponding value  $x^{(i)}$  and the agent receives a negative reward of  $-\lambda c(f_i)$ . The set of currently available feature selecting actions is limited to features not yet selected.

Reward function  $r(\tilde{s}, a)$  is defined as

$$r((x, y, \tilde{\mathcal{F}}), a) = \begin{cases} -\lambda c(f_i) & \text{if } a \in \mathcal{A}_f, a = f_i \\ 0 & \text{if } a \in \mathcal{A}_c \text{ and } a = y \\ -1 & \text{if } a \in \mathcal{A}_c \text{ and } a \neq y \end{cases}$$

where  $\lambda \in \mathbf{R}^+$  is a feature cost factor. By altering its value, we can make the trade-off between precision and average cost. Higher  $\lambda$  forces the agent to prefer lower cost and shorter episodes over precision and vice versa.

The initial state does not contain any disclosed features,  $\tilde{s}_0 = (x, y, \emptyset)$ , and is drawn from the data distribution  $\mathcal{D}$ . The environment is deterministic and the transition function  $t : \tilde{\mathcal{S}} \times \mathcal{A} \rightarrow \tilde{\mathcal{S}} \cup \mathcal{T}$ , where  $\mathcal{T}$  is the terminal state, is defined as:

$$t((x, y, \tilde{\mathcal{F}}), a) = \begin{cases} \mathcal{T} & \text{if } a \in \mathcal{A}_c \\ (x, y, \tilde{\mathcal{F}} \cup a) & \text{if } a \in \mathcal{A}_f \end{cases}$$

These properties make the environment inherently episodic, with a maximal length of an episode of  $|\mathcal{F}| + 1$ .

The goal is to find a policy  $\pi$  that sequentially selects features from  $\mathcal{F}$  and eventually decides a class from  $\mathcal{Y}$ , while maximizing the expected total reward.

Because in any observed state  $s$ , some feature values from the original state  $\tilde{s}$  could be missing, multiple different states  $\tilde{s}$  can map to the same observed state  $s$ . The agent working only with observed states  $s$  therefore cannot predict which state  $\tilde{s}$  it is in. As a result, the agent experiences stochastic transitions and rewards, although the defined environment is deterministic, after the initial choice of the sample. In the following sections, we mainly use the agent's point of view and work with the observed states and expectations over their probabilities. For this reason, it is useful to overload the notation of transition function to also work with observed states. Let  $t(s, a)$  give transition probabilities for all next observed states  $s'$ , when the action  $a$  is taken in the observed state  $s$ :  $t : \mathcal{S} \times \mathcal{A} \rightarrow \Delta(\mathcal{S} \cup \mathcal{T})$ . Similarly, let the reward function  $r(s, a)$  return the *expected* reward when taking the action  $a$  from the observed state  $s$ .

## 2.2 Deep Q-learning

In this work we use Double Deep Q-learning [12, 20] with experience replay and soft-target network [10] to find the optimal policy. This section briefly describes these techniques.

Deep Q-learning is a method that estimates a state-action value function  $Q$  with a neural network. Function  $Q^\pi(s, a)$  expresses the expected discounted reward starting at observed state  $s$ , performing the action  $a$  and following policy  $\pi$  afterwards. It is often written in a recursive form:

$$Q^\pi(s, a) = r(s, a) + \mathbb{E}_{s' \sim t(s, a)} [\gamma Q^\pi(s', \pi(s'))]$$

Note that  $r(s, a)$  is already an expectation over all possible rewards. Action-state value of the terminal state is zero,  $Q(\mathcal{T}, \cdot) = 0$ .

Discount factor  $\gamma$  controls the importance of future rewards. It is useful either in non-episodic domains or when function approximation is used [3]. Although our environment is guaranteed to terminate, the discount factor can help during learning and is preserved in our algorithm as a parameter.

We are interested in the optimal function  $Q^*$ , yielded with a greedy policy  $\pi^*(s) = \operatorname{argmax}_a Q^*(s, a)$ :

$$Q^*(s, a) = r(s, a) + \mathbb{E}_{s' \sim t(s, a)} [\gamma \max_{a'} Q^*(s', a')] \quad (1)$$

In low dimensional and finite state space, the  $Q$  function can be precisely found by dynamic programming. However, the precise solution is not feasible in case of a high-dimensional or continuous state space. To overcome this issue, approximation with neural networks is often employed, leading to Deep Q-Learning [12].

Inspired with dynamic programming, a neural network  $\theta$  approximates a function  $Q^\theta$  by minimizing MSE (mean squared error) between the both sides of eq. (1), for transitions  $(s, a, r, s')$  empirically experienced by an agent following a greedy policy  $\pi^\theta(s) = \operatorname{argmax}_a Q^\theta(s, a)$ . In these transitions,  $s$  and  $a$  denote current state and taken action,  $r$  is experienced reward that in expectation should converge to  $r(s, a)$ , and  $s'$  is the following state,  $s' \sim t(s, a)$ . To put it more formally, we are looking for such parameters  $\theta$ , that iteratively minimize the loss function  $\ell_\theta$ , for a batch of transitions  $\mathcal{B}$ :

$$\ell_\theta(\mathcal{B}) = \frac{1}{|\mathcal{B}|} \sum_{(s, a, r, s') \in \mathcal{B}} (q(r, s') - Q^\theta(s, a))^2 \quad (2)$$

Where  $q$  is the target estimate of the  $Q$  function, and it is constant w.r.t. parameters  $\theta$  in the optimization step.

$$q(r, s') = r + \gamma \max_{a'} Q^\theta(s', a')$$

As the error decreases, the approximated function  $Q^\theta$  converges to  $Q^*$ . However, in practice, this method proved to be unstable [12], and several techniques exist to stabilize it [5]. In this work, we implement Experience replay, Double Q-learning and soft-target network, as they are well established in the RL community and sufficiently demonstrate that our algorithm outperforms the state-of-the-art algorithms.

**Experience replay** allows for efficient usage of acquired transitions, which are often expensive to get. Also, it overcomes issues induced by online learning, where transitions come in a highly correlated sequence. In Experience replay, a circular buffer is used to store recent transitions and each step a number of them are sampled to form a batch for optimization. These samples are less correlated, compared to online learning, and can be replayed multiple times.

<sup>1</sup>Power set  $\wp(\mathcal{S})$  is a set of all subsets of  $\mathcal{S}$ , including the empty set and  $\mathcal{S}$  itself.

**Algorithm 1** Training

---

Randomly initialize parameters  $\theta$  and  $\phi$   
Initialize environments  $\mathcal{E}$  with  $(x, y, \emptyset) \in (\mathcal{X}, \mathcal{Y}, \wp(\mathcal{F}))$   
Initialize experience replay memory  $\mathcal{M}$  with a random agent  
**while** not converged **do**  
  **for all**  $e \in \mathcal{E}$  **do**  
    Simulate one step with  $\epsilon$ -greedy policy  $\pi_\theta$ :  
       $a = \pi_\theta(s); \quad s', r = \text{STEP}(e, a)$   
    Add transition  $(s, a, r, s')$  into circular memory  $\mathcal{M}$   
  **end for**  
  Sample a random batch  $\mathcal{B}$  from  $\mathcal{M}$   
  **for all**  $(s_i, a_i, r_i, s'_i) \in \mathcal{B}$  **do**  
    Compute target  $q_i$ :  

$$q_i = \begin{cases} r_i & \text{if } s'_i = \mathcal{T} \\ r_i + \gamma Q^\phi(s'_i, \arg\max_{a'} Q^\theta(s'_i, a')) & \text{otherwise} \end{cases}$$
  
    Clip  $q_i$  into  $[-1, 0]$   
  **end for**  
  Perform one step of gradient descent on  $\ell_\theta$  w.r.t.  $\theta$ :  

$$\ell_\theta(\mathcal{B}) = \sum_{i=1}^{|\mathcal{B}|} (q_i - Q^\theta(s_i, a_i))^2$$
  
  Update parameters  $\phi := (1 - \rho)\phi + \rho\theta$   
**end while**

---

**Target network** is a technique that decouples the current network values from its update targets. If the target values are generated from the network  $\theta$  itself, it induces an unwanted feedback and causes oscillations and instability. With target network, another network  $\phi$  is used, parallel to the network  $\theta$ , and its  $Q^\phi$  values are used in the  $q$  formula:

$$q(r, s') = r + \gamma \max_{a'} Q^\phi(s', a')$$

Parameters  $\phi$  should reflect  $\theta$  with some delay. Here, two techniques are used. Either the parameters  $\phi$  are set to  $\theta$  after a number of optimization steps [12], or it slowly follows  $\theta$ , as the computation continues:  $\phi := (1 - \rho)\phi + \rho\theta$ . Number  $\rho$  is a target network factor, leading to soft-target network [10], which we use in this work.

**Double Q-learning** [4, 20] is a technique to reduce maximization bias induced by max in the formula for  $q$ . It builds upon the target network technique and combines the two estimates  $Q^\theta$  and  $Q^\phi$ . Here, the maximizing action is taken from  $Q^\theta$ , but its value from the target network  $Q^\phi$ :

$$q(r, s') = r + \gamma Q^\phi(s', \arg\max_{a'} Q^\theta(s', a')) \quad (3)$$

Further, we use  $\epsilon$ -greedy policy to enforce exploration. This policy behaves greedily most of the time, but picks a random action with probability  $\epsilon$ . The parameter  $\epsilon$  is linearly decreased over time towards zero. The exact parameters used in the implementation are summarized in Table 2 in the appendix.

**Algorithm 2** Environment simulation

---

Operator  $\odot$  marks the element-wise multiplication.

**function** STEP( $e \in \mathcal{E}, a \in \mathcal{A}$ )

**if**  $a \in \mathcal{A}_c$  **then**

$$r = \begin{cases} 0 & \text{if } a = e.y \\ -1 & \text{if } a \neq e.y \end{cases}$$

  Reset  $e$  with a new sample  $(x, y, \emptyset)$  from a dataset

  Return  $(\mathcal{T}, r)$

**else if**  $a \in \mathcal{A}_f$  **then**

    Add  $a$  to the set of selected features:  $e.\tilde{\mathcal{F}} = e.\tilde{\mathcal{F}} \cup a$

    Create mask  $m^{(i)} = 1$  if  $f_i \in \tilde{\mathcal{F}}$  and 0 otherwise

    Return  $((e.x \odot m, m), -\lambda c(a))$

**end if**

**end function**

---

**2.3 Algorithm**

We implement an environment according to the defined POMDP, where each episode corresponds to one randomly drawn sample  $(x, y)$  from a training dataset, that defines its initial state  $\tilde{s}_0$ . At any time, a state is in the form of  $\tilde{s} = (x, y, \tilde{\mathcal{F}})$ , from which an observed state  $s = \{(x^{(i)}, f_i) \mid \forall f_i \in \tilde{\mathcal{F}}\}$  is created. Neural networks accept only numeric input and so the observed state  $s$  is mapped into a tuple  $(\bar{x}, m)$ . Vector  $\bar{x} \in \mathbf{R}^n$  is a masked vector of the original  $x$ . It contains values of  $x$  which have been acquired and zeros for unknown values:

$$\bar{x}^{(i)} = \begin{cases} x^{(i)} & \text{if } f_i \in \tilde{\mathcal{F}} \\ 0 & \text{otherwise} \end{cases}$$

Mask  $m \in \{0, 1\}^n$  is a vector denoting whether a specific feature has been acquired, and it contains 1 at a position of acquired feature, or 0:

$$m^{(i)} = \begin{cases} 1 & \text{if } f_i \in \tilde{\mathcal{F}} \\ 0 & \text{otherwise} \end{cases}$$

The combination of  $\bar{x}$  and  $m$  ensures that the network can differentiate between a feature not present and observed value of zero.

We use Double DQN agent with a soft-target network to find an approximation of the optimal  $Q^*$  function and use the greedy policy to evaluate it on a test dataset.

The neural network architecture follows that of [12] – its parameters are shared for all actions, the network accepts an observed state and outputs  $Q$  values for all actions. It consists of an input layer accepting a state, three hidden fully connected layers with RELU activation and a linear output layer with one output per action.

We normalize each feature in the dataset with its mean and standard deviation. A value of zero is imputed in place of unknown feature value because it corresponds to an average in a zero-centered dataset.

Mainly for performance reasons we simulate 1000 parallel independent environments at once. Compared to [12] we have a full control of the environment, and this enables an efficient implementation. In each iteration, we simulate one step in all the environments, generating 1000 transitions  $(s, a, r, s')$  at once. From the algorithmic perspective, this approach is similar to [11], however here we do it synchronously.

Experienced transitions  $(s, a, r, s')$  are stored in a circular buffer  $\mathcal{M}$  and are sampled randomly to form a batch  $\mathcal{B}$ . We use RMSProp [18] with momentum parameter 0.95 and gradient normalization, if its norm exceeds 1.0, to optimize the loss function  $\ell_\theta(\mathcal{B})$ , eq. (2) and (3). The learning rate decays exponentially over time. Exact parameters differ for each dataset and are summarized in Table 2. After each optimization, weights of the target network are updated with  $\phi := (1 - \rho)\phi + \rho\theta$ . Overview of the algorithm can be seen in Algorithm 1 and details of the environment simulation are in Algorithm 2.

We use a value of 1.0 as the discount factor  $\gamma$ . The environment is episodic with a short average length of episodes, and as such it makes sense to consider full returns until the end. Undiscounted returns can in theory yield a better strategy to choose features. However, the discount factor can be helpful in case of a large number of features, where it can help with the stability of the function approximation, by not considering the future states as much. It could also provide a way to accelerate the learning process [3]. We do not study these effect in this paper and leave it to future work.

To promote exploration, we use  $\epsilon$ -greedy policy. Exploration rate  $\epsilon$  starts at a defined initial value and it is linearly decreased over time. Exact values of  $\epsilon$  differ based on the dataset, and they are described in Table 2 in the appendix.

Reward for a correct classification is 0 and  $-1$  for an error. Cost for feature selecting actions is dataset dependent, but normalized into  $[0, 1]$ . Feature cost factor  $\lambda$  satisfies  $0 < \lambda \leq 1$ , but in practice it is  $\lambda \ll 1$ . Rewards for feature selecting actions are  $r = -\lambda c(f)$ . In such an environment, the optimal function  $Q^*$  is also bounded:  $-1 - \lambda \leq Q^* \leq 0$ . We use this domain knowledge and clip the target  $r + \gamma Q^\phi(s', \arg\max_{a'} Q^\theta(s', a'))$  in eq. (3) to  $[-1, 0]$ . We experimentally found that this approach avoids initial explosion in predicted values and weights and as such greatly speeds up and stabilizes the learning process.

Opposed to [12], we use rather large batches in the optimization step. One batch contains  $10^5$  transitions. As we simulate 1000 environments in one step, this means that one generated transition will be replayed about 100 times on average. We found that this number works well and enables efficient use of the hardware.

## 2.4 Theoretical Analysis

We analyze the algorithm theoretically and derive the following interesting facts, that could help us to better understand the algorithm and may lead to future improvements.

**PROPOSITION 2.1.** *For any observed state  $s \in \mathcal{S}$ , let  $a^*$  be the optimal action to play in  $s$ , i.e.,  $a^* = \arg\max_a Q^*(s, a)$ , then:*

$$\forall a \in \mathcal{A}_f \quad Q^*(s, a) \geq Q^*(s, a^*) - \lambda c(a)$$

The proof is in the appendix. Since  $c(a) \leq 1$ , the immediate corollary is that for any observed state  $s$ , the maximal distance between  $Q^*$  values for any feature selecting actions is at most  $\lambda$ :

$$a_1, a_2 \in \mathcal{A}_f : |Q^*(s, a_1) - Q^*(s, a_2)| \leq \lambda$$

Since  $\lambda$  is typically very small, this means that the  $Q$  values for these actions are close to each other and their variance has to be sufficiently reduced for these actions to be distinguishable. This can be achieved either by a sufficiently small learning rate or a large batch size.

For the following proposition, we need to define a concept of samples consistent with an observed state. A sample  $(x, y)$  is consistent with an observed state  $s = \{(x_s^{(i)}, f_i) \mid \forall f_i \in \mathcal{F}\}$ , if all disclosed values  $x_s^{(i)}$  are equal to  $x^{(i)}$ . Let  $\mathcal{D}(s)$  be a subset of all samples  $(x, y)$  from a dataset  $\mathcal{D}$ , that are consistent with the observed state  $s$ .

**PROPOSITION 2.2.** *For any observed state  $s \in \mathcal{S}$  and any classifying action  $a \in \mathcal{A}_c = \mathcal{Y}$ , the following holds:*

$$Q^*(s, a) = r(s, a) = P[a \mid (x, a) \in \mathcal{D}(s)] - 1$$

The proof is in the appendix. In other words, the true  $Q$  value for any classification action is a ratio of classes corresponding to this action between all samples consistent with the current observed state  $s$ , minus one. This fact could be directly used to minimize the variance of classification actions, by directly learning this correct value instead of stochastic rewards of  $-1$  and  $0$ , that converge to the correct value. However, it would require additional computation, and we leave it to future work.

## 3 EXPERIMENTAL RESULTS

We evaluate our algorithm on three datasets – MiniBooNE particle identification [9], Forest CoverType [9] and CIFAR-10 [6]. We prepared and split the datasets into training/validation/testing sets according to [14], to use the same setting when comparing the results. Because these datasets do not have an explicit cost associated with features, uniform cost of 1.0 is used. Multiple values between 1.0 and  $1e-4$  are used as the cost factor  $\lambda$ , and different values correspond to different average feature cost. Information about the datasets is summarized in Table 1.

We directly compare to prior art algorithms Adapt-Gbrt [14] and BudgetPrune [16], which are both decision trees based algorithms. We do not report algorithms that were already shown to perform worse in [14] or [16]. We retrieved their performance from the published results, where available. One machine equipped with Intel Xeon CPU E5530 and NVIDIA TITAN X GPU was used to measure the time needed for the algorithm to converge, which we report in Table 1, and which we feel is an important factor for its practical usage. Unfortunately, the prior art did not report the learning time needed for their algorithms, therefore we cannot compare it. Note that in our algorithm, only the training needs

**Table 1: Evaluated datasets**

Parameter	MiniBooNE	Forest	CIFAR-10
Features	50	54	400
Classes	2	2 <sup>†</sup>	2 <sup>†</sup>
Train size	45359	36603	19761
Validation size	19439	15688	8468
Test size	64798	58101	10000
Network size	3x128	3x256	3x512
Training time	5 min	9 hrs	95 hrs
Transitions	10M	500M	1500M

<sup>†</sup> Classes in these datasets are binarized to be comparable to results reported in prior art.

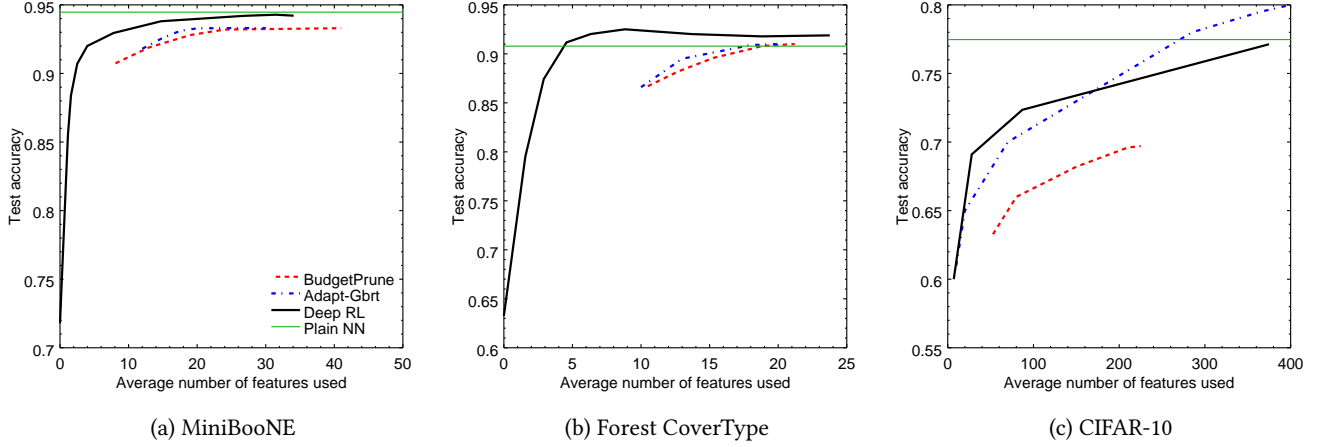


Figure 1: Algorithm performance in tested datasets.

considerable resources, the execution time of a trained model is negligible.

As our algorithm uses neural networks, we evaluate the accuracy of a plain neural network classifier and report it as a baseline. This accuracy provides an estimate of an upper bound, to which our algorithm should approach. For this baseline network, we used three layers with RELU activation, with the same size as in our algorithm for a particular dataset, and an output layer with one output per class with a softmax activation.

The comparison of the achieved trade-off between classification accuracy and feature cost is shown in Figure 1. The exact size of the neural network layers, the number of transitions<sup>2</sup> needed to converge and the training time are reported in Table 1.

**MiniBooNE** dataset is compact and well suited for fast experiments, having only two classes and 50 real-valued features. Our algorithm operates for the whole range of  $\lambda$ , giving a different trade-off for each value. At any point, it performs better compared to other algorithms, as shown in Figure 1 (a). The accuracy with about 32 features on average almost reaches the accuracy of the baseline network with all features.

**Forest CoverType** dataset originally has 581012 samples and 7 classes, however, prior art algorithms reported their results only on a reduced version with 110392 samples and 2 classes. For comparison, we report results on this reduced dataset. This dataset has 54 features – 10 real-valued and 2 categorical. The categorical features are one-hot encoded into binary vectors of length 4 and 40. These values are used as actual features in the dataset. The learning time for this dataset is 50x longer than for MiniBooNE dataset. Compared to prior art, our algorithm performs better, for any value of  $\lambda$  (see Figure 1 (b)).

Another interesting fact is that the algorithm’s best classification accuracy exceeds the accuracy of the baseline neural network. This suggests that our algorithm might work as some sort of regularization. Figure 2 (a) shows a further comparison between our

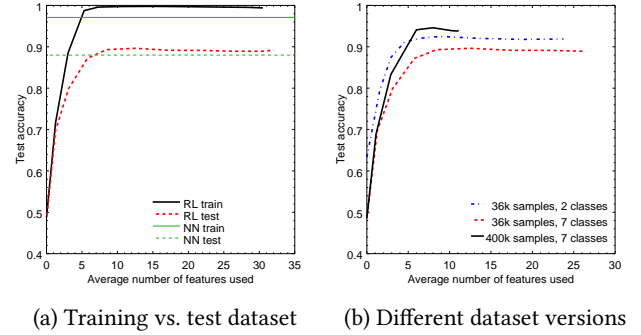


Figure 2: Experiments on Forest dataset. (a) Performance on training and test datasets (Forest with 36603 training samples and 7 classes), compared to plain neural network based classifier using all features. (b) Performance in different versions of Forest dataset. We show a difference in training dataset size (36603 or 400000 samples) and the binarized or original multiclass problem.

algorithm and the baseline classifier, and it suggests that some features can actually hurt performance. The baseline classifier using all features does not reach the performance of our algorithm on neither training nor testing dataset.

To show that our algorithm can perform multiclass classification and work with the original large dataset, we also report its performance in Figure 2 (b).

**CIFAR-10** is an image dataset, where the goal is to determine what object is in the picture. The dataset was preprocessed and classes binarized according to [14]. This dataset contains 400 real-valued features. Our algorithm performs better than prior art up to a point, where it uses 165 features on average (Figure 1 (c)). After that, the Adapt-Gbrt algorithm takes the lead. Our algorithm converges to the limit of the baseline classifier, indicating that feed-forward networks are not suitable for this problem. However, we

<sup>2</sup>Transition means one step of an agent, generating one tuple  $(s, a, r, s')$ .

could arrange the features into a grid pattern and easily extend our algorithm to use convolutional networks, which are known to work very well for this type of problems [7, 12]. We do not use this approach in this work because we maintain the same architecture for all datasets.

### 3.1 Algorithm Properties

We analyze the behavior of the algorithm with different parameters and report our observations in this section.

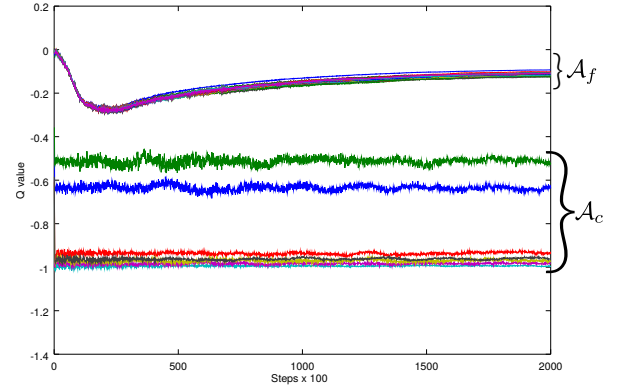
In Figure 3 (a), we observe how the Q values of the initial observed state  $s_0$  evolve during the training on the multiclass Forest dataset. We see that the algorithm quickly estimates the correct Q values for all classification actions  $a \in \mathcal{A}_c$ , since their value is dependent only on the distribution of classes in the dataset. Q values of feature selection actions  $a \in \mathcal{A}_f$  change only slowly, due to their dependence on the next state. These Q values are close to each other, as predicted by Proposition 2.1.

Figure 3 (b) shows the evolution of the average reward, episode length and accuracy on a small validation dataset during learning. The initial policy that the algorithm quickly finds corresponds to always selecting a class which has the highest ratio in the dataset. This is expectable, since this strategy achieves an optimal accuracy without any feature. From this point, the average accuracy and reward slowly increase to their supremum.

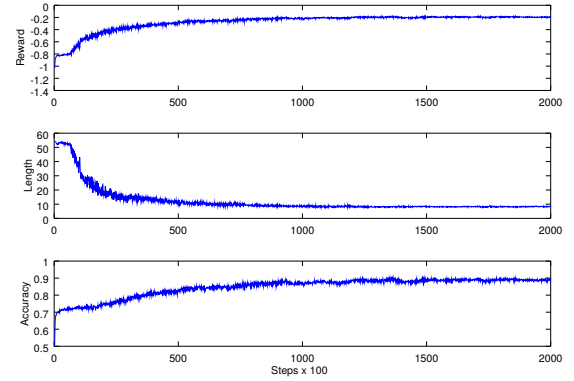
Figure 3 (b) in the middle and Figure 4 (a) shows the evolution of the average number of acquired features, which is equivalent to the average episode length, in this case. We see that the algorithm starts by selecting all the features at first and subsequently decreasing their amount. As already noted, the Q values of classification actions can be estimated very quickly, into their range between -1 and 0, since they are not dependent on any next state. However, the Q values for feature selection actions change slowly, and as a result, their value is higher than Q values of classification actions, at the beginning of the learning. This makes the algorithm to sequentially select all features before taking any classification action. However, then the algorithm finds a core subset of features that are important and subsequently add features to improve accuracy, as shown in Figure 4 (a) and (b).

In Figure 5, we show a histogram of number of used features for different samples, in the multiclass Forest dataset. It confirms that the algorithm selects a different subset of features for each sample since every episode length is present. According to the histogram, the samples can be divided into three categories – easy, average and difficult. The easy samples are classified almost immediately, as soon as they are identified. Average samples form the majority and require more features. The group of difficult samples is also of a significant size, and these samples require all the features from the dataset. Rest of the samples, not belonging to any obvious category, are distributed in between.

Next, we study the effect of  $\lambda$  and neural network size on performance. For computational efficiency, we use the MiniBooNE dataset. The trade-off between the accuracy and the average feature cost is controlled by the parameter  $\lambda$ , as shown in Figure 6 (a). Higher  $\lambda$  forces the algorithm to select fewer features on average, and as a consequence, the accuracy falls down. We also report the effect of



(a) Q values of initial state



(b) Reward, length and accuracy

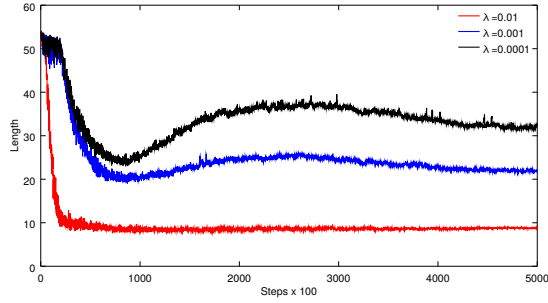
**Figure 3: Learning progress in multiclass Forest dataset with  $\lambda = 0.01$ . (a) Learned Q values for all actions,  $Q^\theta(s_0, \cdot)$ , in the initial state  $s_0$ . (b) Average reward, length and accuracy through learning on validation dataset. One point on x axis corresponds to 100 learning steps. The graph is truncated, it would end with 5000 on the x axis, for the full training.**

the neural network size on the performance, in Figure 6 (b). Here, a larger network always finds a better trade-off, for any  $\lambda$  value.

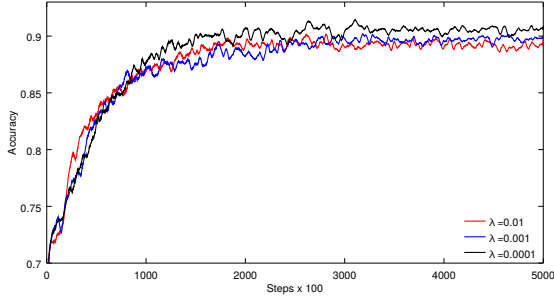
## 4 RELATED WORK

The most similar work to our approach is [2], which also used Q-learning. However, their algorithm worked with a limited linear approximation, which resulted in inferior performance. We instead use neural networks allowing us to use our algorithm in much wider range of complex problems.

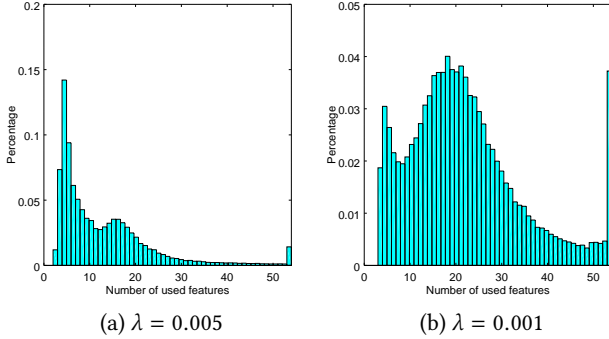
Authors of [1] use a recurrent neural network that uses attention to select blocks of features and classifies after a fixed number of steps. Compared to our work, decisions of this network are hard to explain. On the other hand, our Q-learning based algorithm produces semantically meaningful values for each action. Moreover, as we use a standard algorithm, we can directly benefit from any enhancements made by the community.



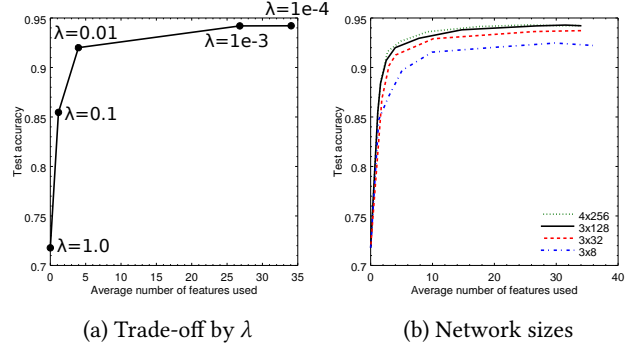
(a) Average number of used features



(b) Average accuracy

**Figure 4: Effect of different  $\lambda$  on the average number of acquired features and accuracy on multiclass Forest dataset.****Figure 5: Histogram of used features for different samples on the multiclass Forest dataset, with different  $\lambda$ . On both (a) and (b) we see three peaks, corresponding to very easy, average and difficult samples.**

There is a plethora of tree-based algorithms [8, 14–16, 24–26]. The article [14] implements Adapt-Gbdt algorithm and it represents the state-of-the-art which we directly compare to. This algorithm uses two separate models and a gating function that selects which one is more appropriate to use with a given sample. High-Prediction Cost (HPC) model classifies samples regardless of the cost and Low-Prediction Cost (LPC) model adaptively approximates HPC model in regions of input space where it achieves adequate accuracy. In

**Figure 6: Effect of  $\lambda$  and neural network size on the algorithm performance in the MiniBooNE dataset. (a) Different selected values of  $\lambda$  affect the accuracy vs. feature cost trade-off. (b) Comparison of different neural network sizes. Smaller network achieves worse overall performance.**

this concrete instantiation, Adapt-Gbdt, it uses Gradient Boosted Regression Trees (GBRT) as the LPC model and either Random Forests or RBF-SVM as the powerful HPC model. Essentially, this algorithm is not restricted to these models and could even be instantiated with a form of our algorithm.

A different set of algorithms employed Linear Programming (LP) to this domain [21, 22]. The algorithm presented in [21] uses LP to select a model with the best accuracy and lowest cost, from a set of pre-trained models, all of which use a different set of features. The algorithm also chooses a model based on the complexity of the sample, similarly to [14].

In [23], the authors propose to reduce the problem by finding different disjoint subsets of features, that are used together as macro-features. These macro-features form a graph, which is solved by a dynamic programming based algorithm. The algorithm for finding different subsets of features is complementary to our algorithm and could be possibly used jointly to improve performance.

Reference [19] uses a fixed order of features to reveal, with increasingly complex models that can use them. However, the order of features is not computed, and it is assumed that it is set manually. On the other hand, our algorithm is not restricted to a fixed order of features (for each sample it can choose a completely different subset), and it can also find their significance automatically.

The early work [17] analyzes a problem similar to our definition of classification with costly features. However, algorithms introduced there require memorization of all training examples, which is not scalable in many domains.

## 5 CONCLUSION

We formulate the problem of classification with costly features as a generic sequential decision-making problem. We solve it with vanilla Double DQN algorithm and show that even this basic RL algorithm exceeds the performance of prior-art algorithms on MiniBooNE and Forest CoverType dataset and partially exceeds the performance on the Cifar-10 dataset. However, the architecture



of the used neural network can be easily adapted to a specific domain, e.g., to use convolutions in case of Cifar-10, which is likely to further improve performance.

Our algorithm can be easily extended with any improvement in the basic DQN algorithm [5], such as Dueling DQN, Prioritized Experience Replay or Distributional RL. A great speed-up could be possibly achieved by using longer traces [13]. Combination with complementary algorithms [14, 23] could also improve performance.

## ACKNOWLEDGMENTS

The GPU used for this research was donated by the NVIDIA Corporation. Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures" (CESNET LM2015042), is greatly appreciated.

## APPENDIX

PROPOSITION 2.1. *For any observed state  $s \in \mathcal{S}$ , let  $a^*$  be the optimal action to play in  $s$ , i.e.,  $a^* = \arg\max_a Q^*(s, a)$ , then:*

$$\forall a \in \mathcal{A}_f \quad Q^*(s, a) \geq Q^*(s, a^*) - \lambda c(a)$$

Table 2: Algorithm Parameters

Parameter	Miniboone	Forest	CIFAR-10
Network size	3x128	3x256	3x512
Parallel agents	1000	1000	1000
Replay buffer size	2e6	1e7	1e6
Batch size	1e5	1e5	1e5
Training steps <sup>*</sup>	1e4	5e5	2e6
Soft-target factor $\rho$	0.01	0.01	0.01
$\epsilon_{\text{start}}$ <sup>†</sup>	0.50	0.70	1.00
$\epsilon_{\text{end}}$	0.05	0.15	0.10
$\epsilon_{\text{steps}}$	2e3	2e4	1e5
LR <sub>start</sub> <sup>‡</sup>	1e-4	1e-4	1e-6
LR <sub>end</sub>	1e-7	1e-7	3e-7
LR <sub>steps</sub>	5e3	1e4	1e4
LR <sub>factor</sub>	0.1	0.9	0.9

<sup>\*</sup> One step means simulating all 1000 agents for one step.

<sup>†</sup> Epsilon controls exploration in the  $\epsilon$ -greedy policy. The learning starts with  $\epsilon_{\text{start}}$  and linearly decreases to  $\epsilon_{\text{end}}$  within  $\epsilon_{\text{steps}}$  steps.

<sup>‡</sup> The learning rate is initialized to LR<sub>start</sub> and it is multiplied by LR<sub>factor</sub> every LR<sub>steps</sub> steps, until it reaches its minimum LR<sub>end</sub>.

PROOF. Let  $V^*(s) = \max_a Q^*(s, a)$  denote the expected value of playing the optimal policy from observed state  $s$ .

$$\begin{aligned}
Q^*(s, a) &= -\lambda c(a) + \mathbb{E}_{s' \sim t(s, a)} \left[ \max_{a'} Q^*(s', a') \right] \\
&\geq -\lambda c(a) + \mathbb{E}_{s' \sim t(s, a)} [Q^*(s', a^*)] \\
&= -\lambda c(a) - \lambda c(a^*) + \mathbb{E}_{s' \sim t(s, a)} \mathbb{E}_{s^* \sim t(s', a^*)} [V^*(s^*)] \\
&\geq -\lambda c(a) - \lambda c(a^*) + \mathbb{E}_{s^* \sim t(s, a^*)} [V^*(s^*)] \\
&= -\lambda c(a) + Q^*(s, a^*)
\end{aligned}$$

The first step follows from the maximal action being better than any other action. The second step is derived from the definition of the  $Q$  function. For the third step, we need to realize that the single expectation, as well as the pair of expectations, is over the same set of data samples. For each sample, we can follow the exact same policy from  $s^*$  as we would from  $s^*$ . This policy will have the same misclassification cost, averaged over all samples. Furthermore, if this policy asks about the feature  $a$ , we do not pay the cost for this feature. Therefore, in expectation over all samples reaching  $s$ , the average feature cost included in values  $V^*(s^*)$  will not be higher than the feature cost included in values  $V^*(s^*)$ . Hence, the overall expected value in states  $s^*$  is not higher than the expected value in states  $s^*$ .  $\square$

PROPOSITION 2.2. *For any observed state  $s \in \mathcal{S}$  and any classifying action  $a \in \mathcal{A}_c = \mathcal{Y}$ , the following holds:*

$$Q^*(s, a) = r(s, a) = P[a \mid (x, a) \in \mathcal{D}(s)] - 1$$

PROOF.

$$\begin{aligned}
Q^*(s, a) &= r(s, a) \\
&= 0 \cdot P[a \mid (x, a) \in \mathcal{D}(s)] + (-1) \cdot (1 - P[a \mid (x, a) \in \mathcal{D}(s)]) \\
&= P[a \mid (x, a) \in \mathcal{D}(s)] - 1
\end{aligned}$$

$\square$

All classification actions terminate the episode, so that the  $Q^*$  value is merely the expected reward. If  $p = P[a \mid (x, a) \in \mathcal{D}(s)]$  is the ratio of class  $a$  in the set of samples consistent with the observed state  $s$ , the correct classification happens, in expectation, with probability  $p$  and the reward is 0. Incorrect classification happens with probability  $(1 - p)$  and the reward is  $-1$ .

## REFERENCES

- [1] Gabriella Contardo, Ludovic Denoyer, and Thierry Artieres. 2016. Recurrent neural networks for adaptive feature acquisition. In *International Conference on Neural Information Processing*. Springer, 591–599.
- [2] Gabriel Dulac-Arnold, Ludovic Denoyer, Philippe Preux, and Patrick Gallinari. 2011. Datum-wise classification: a sequential approach to sparsity. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 375–390.
- [3] Vincent François-Lavet, Raphael Fonteneau, and Damien Ernst. 2015. How to discount deep reinforcement learning: Towards new dynamic strategies. *arXiv preprint arXiv:1512.02011* (2015).
- [4] Hado V Hasselt. 2010. Double Q-learning. In *Advances in Neural Information Processing Systems*. 2613–2621.
- [5] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. 2017. Rainbow: Combining Improvements in Deep Reinforcement Learning. *arXiv preprint arXiv:1710.02298* (2017).



- [6] Alex Krizhevsky and Geoffrey Hinton. 2009. *Learning multiple layers of features from tiny images*. Master's thesis. University of Toronto.
- [7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.
- [8] Matt J Kusner, Wenlin Chen, Quan Zhou, Zhixiang Eddie Xu, Kilian Q Weinberger, and Yixin Chen. 2014. Feature-Cost Sensitive Learning with Submodular Trees of Classifiers. In *AAAI*. 1939–1945.
- [9] M. Lichman. 2013. UCI Machine Learning Repository. (2013). <http://archive.ics.uci.edu/ml>
- [10] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).
- [11] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*. 1928–1937.
- [12] 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. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.
- [13] Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc Bellemare. 2016. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*. 1054–1062.
- [14] Feng Nan and Venkatesh Saligrama. 2017. Adaptive Classification for Prediction Under a Budget. *arXiv preprint arXiv:1705.10194* (2017).
- [15] Feng Nan, Joseph Wang, and Venkatesh Saligrama. 2015. Feature-Budgeted Random Forest. In *International Conference on Machine Learning*. 1983–1991.
- [16] Feng Nan, Joseph Wang, and Venkatesh Saligrama. 2016. Pruning random forests for prediction on a budget. In *Advances in Neural Information Processing Systems*. 2334–2342.
- [17] Ming Tan. 1993. Cost-sensitive learning of classification knowledge and its applications in robotics. *Machine Learning* 13, 1 (1993), 7–33.
- [18] Tijmen Tieleman and Geoffrey Hinton. 2012. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning* 4, 2 (2012), 26–31.
- [19] Kirill Trapeznikov and Venkatesh Saligrama. 2013. Supervised sequential classification under budget constraints. In *Artificial Intelligence and Statistics*. 581–589.
- [20] Hado Van Hasselt, Arthur Guez, and David Silver. 2016. Deep Reinforcement Learning with Double Q-Learning. In *AAAI*. 2094–2100.
- [21] Joseph Wang, Tolga Bolukbasi, Kirill Trapeznikov, and Venkatesh Saligrama. 2014. Model selection by linear programming. In *European Conference on Computer Vision*. Springer, 647–662.
- [22] Joseph Wang, Kirill Trapeznikov, and Venkatesh Saligrama. 2014. An lp for sequential learning under budgets. In *Artificial Intelligence and Statistics*. 987–995.
- [23] Joseph Wang, Kirill Trapeznikov, and Venkatesh Saligrama. 2015. Efficient learning by directed acyclic graph for resource constrained prediction. In *Advances in Neural Information Processing Systems*. 2152–2160.
- [24] Zhixiang Xu, Matt Kusner, Kilian Weinberger, and Minmin Chen. 2013. Cost-sensitive tree of classifiers. In *International Conference on Machine Learning*. 133–141.
- [25] Zhixiang Xu, Kilian Weinberger, and Olivier Chapelle. 2012. The greedy miser: Learning under test-time budgets. *arXiv preprint arXiv:1206.6451* (2012).
- [26] Zhixiang Eddie Xu, Matt J Kusner, Kilian Q Weinberger, Minmin Chen, and Olivier Chapelle. 2014. Classifier cascades and trees for minimizing feature evaluation cost. *Journal of Machine Learning Research* 15, 1 (2014), 2113–2144.