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 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 (CwCF) 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 at the lowest total cost.

CwCF has been studied for over 25 years [16]. It is a practical problem with many real use cases. Let's name a network intrusion detection, in which an analyst examines network traffic, queries different expensive data sources, and makes a decision whether the current sample is malicious or benign. Another example is a medical diagnosis, where a doctor can make a sequence of expensive 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.

CwCF has been addressed by linear programming [21], trees of classifiers [7, 23], or recurrent neural networks with specific structure [1]. Most recent line of work relays on random forests with pruning [14, 15], and gradient boosted trees [13].

In this paper, we show that even the most recent of these problem specific methods can be outperformed by generic Reinforcement Learning (RL) algorithms. We view classification of each sample as an episode of a reinforcement learning problem, where an agent sequentially decides whether to acquire another feature and which, or whether to already classify the sample. At each step, the agent can base its decision on the values of the features acquired so far. For the actions requesting a feature the agent receives a negative reward, equal to the feature cost. For the classification actions, the reward is based on the true class of the sample.

We build upon prior work [2] that uses RL algorithm Q-learning for CwCF. It uses linear regression to approximate the Q function, which does not allow to reach the state-of-the-art performance. We replace the linear regression with deep learning and base our method on Deep Q-network [11] (DQN) with Double Q-learning [3]. Although the DQN algorithm can be extended with multiple techniques [4], we show that even basic Double DQN algorithm can outperform the state-of-the-art techniques, that have been specifically tailored to this domain. This brings the immediate benefit of improved performance in the task of classification with costly features, but also allows any future advancements of Deep Reinforcement Learning to be readily applied also in CwCF. Furthermore, using RL for CwCF enables continual adaptation to changes in non-stationary environments, commonly present in real-world problems.

The main contribution of this paper is the empirical evaluation showing that solving the problem of classification with costly features using deep reinforcement learning outperforms the current state-of-the-art developed specifically for this problem. On three data sets, DQN [11] achieves a substantially higher precision than Adapt-Gbrt [13] and BudgetPrune [15] with the same average cost for acquired features. An additional contribution is showing that DQN performs well in problems, which are substantially different from those where DQN was developed and evaluated (primarily on Atari games). Specifically, CwCF is stochastic, with a large number of actions (hundreds to thousands) and without an apparent structure that could be leveraged with convolutions.

2 METHOD

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 the 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 $\tilde{\mathcal{S}}$, set of actions $\mathcal{A}$, reward function r and transition function t.

Let $(x, y) \in \mathcal{D}$ be a sample drawn from a data distribution $\mathcal{D}$. Vector $x \in \mathcal{X} \subseteq \mathbf{R}^n$ contains feature values, where $x^{(i)}$ is a value

Adaptive Learning Agents Workshop, 2018
2018. https://doi.org/

of feature $f_i \in \mathcal{F} = \{f_1, ..., f_n\}$, n is the number of features, and $y \in \mathcal{Y}$ is a class. Let $c : \mathcal{F} \to \mathbf{R}$ be a function mapping a feature f into its real-valued cost $c(f)$.

States $\tilde{s} = (x, y, \bar{\mathcal{F}}) \in \tilde{\mathcal{S}} = \mathcal{X} \times \mathcal{Y} \times \wp(\mathcal{F})$, where $\wp(\mathcal{F})$ is a power set of $\mathcal{F}$, represents a sample (x, y) and currently selected set of features $\bar{\mathcal{F}}$. 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: $s = \{(x^{(i)}, f_i) \mid \forall f_i \in \bar{\mathcal{F}}\}$.

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 $r : \tilde{S} \times \mathcal{A} \to \mathbf{R}$ is defined as

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

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 λ 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 $t : \tilde{\mathcal{S}} \times \mathcal{A} \to \tilde{\mathcal{S}} \cup \mathcal{T}$, where $\mathcal{T}$ is the terminal state, is:

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

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 π 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: $t : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S} \cup \mathcal{T})$. Similarly, let the reward function $r(s, a)$ return the *expected* reward when taking the action a from the observed state s.

2.1 Deep Q-learning

In this work we use Double Deep Q-learning [11, 19] with experience replay and soft-target network [9] 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 π afterwards. It is often written in a recursive form:

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

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 γ controls the importance of future rewards. It is useful either in non-episodic domains or when function approximation is used [?]. 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 by a greedy policy $\pi^*(s) = \operatorname{argmax}_a Q^*(s, a)$:

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

In a 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 [11].

Inspired with dynamic programming, a neural network θ approximates a function Q^θ 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 parameters θ by iteratively minimizing the loss function ℓ_θ, for a batch of transitions $\mathcal{B}$:

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

where q is the target estimate of the Q function, and it is constant w.r.t. parameters θ in the optimization step.

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

As the error decreases, the approximated function Q^θ converges to Q^*. However, in practice, this method proved to be unstable [11], and several techniques exist to stabilize it [4]. 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 in 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.

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

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

Parameters ϕ should reflect θ with some delay. Either the parameters ϕ are set to θ after a number of optimization steps [11], or it slowly follows θ, as the computation continues: $\phi := (1 - \rho)\phi + \rho\theta$. Number ρ is a target network factor, leading to soft-target network [9], which we use in this work.

Double Q-learning [3, 19] is a technique to reduce maximization bias induced by maximum in the formula for q. It builds upon the target network technique and combines the two estimates Q^{θ} and Q^{ϕ}. Here, the maximizing action is taken from Q^{θ}, but its value from the target network Q^{ϕ}:

$$q(r, s') = r + \gamma Q^{\phi}(s', \operatorname*{argmax}_{a'} Q^{\theta}(s', a')) \tag{3}$$

Further, we use ϵ-greedy policy to enforce exploration. This policy behaves greedily most of the time, but picks a random action with probability ϵ. The parameter ϵ is linearly decreased over time. The exact parameters used in the implementation are summarized in Table A.1 in the appendix.

2.2 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 it's initial state $\tilde{s}_0$. At any time, a state is in the form of $\tilde{s} = (x, y, \bar{\mathcal{F}})$, from which an observed state $s = \{(x^{(i)}, f_i) \mid \forall f_i \in \bar{\mathcal{F}}\}$ 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:

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

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:

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

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 [11] – 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.

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 replay buffer $\mathcal{M}$ 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 buffer $\mathcal{M}$
    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 $q_i$:
            $q_i = \begin{cases} r_i & \text{if } s'_i = \mathcal{T} \\ r_i + \gamma Q^{\phi}(s'_i, \operatorname*{argmax}_{a'} Q^{\theta}(s'_i, a')) & \text{otherwise} \end{cases}$
        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
```

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 set of selected features: $e.\bar{\mathcal{F}} = e.\bar{\mathcal{F}} \cup a$
        Create mask $m^{(i)} = 1$ if $f_i \in \bar{\mathcal{F}}$ and 0 otherwise
        Return $((e.x \odot m, m), -\lambda c(a))$
    end if
end function
```

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 [11] we have 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 [10], 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

[17] 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 decay exponentially over time. Exact parameters differ for each dataset and are summarized in Table A.1. After each optimization step, the 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 γ. The environment is episodic with a short average length of episodes, and as such it makes sense to consider full returns. Undiscounted returns can in theory yield a better strategy to choose features.

To promote exploration, we use ϵ-greedy policy. Exploration rate ϵ starts at a defined initial value and it is linearly decreased over time. Exact values of ϵ differ based on the dataset, and they are described in Table A.1 in the appendix.

Reward for a correct classification is 0 and -1 for an error. This leads to an optimistic initialization, since the outputs of an untrained network are around zero. Cost for feature selecting actions is dataset dependent, but normalized into $[0, 1]$. Feature cost factor λ satisfies $0 < \lambda \leq 1$, but in pracice 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 $r + \gamma Q^\phi(s', \operatorname{argmax}_{a'} Q^\theta(s', a'))$ 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 [11], 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.

2.3 Theoretical Analysis

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

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

The intuition behind the proof is that after taking a suboptimal action a, the agent can still take the original optimal action a^*, entering a state that has at least the same amount of information, but the cost for the extra action is $\lambda c(a)$. The exact proof is in the appendix. Since $c(a) \leq 1$, the corollary is that for any observed state s, the distance between Q^* values for any feature selecting actions is at most λ:

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

Since λ 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 $s = \{(x_s^{(i)}, f_i) \mid \forall f_i \in \bar{\mathcal{F}}\}$, 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:*

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

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. The exact proof is in the appendix. It is derived from the fact that correct classification happens with probability $P\left[a \mid (x, a) \in \mathcal{D}(s)\right]$ and the possible rewards are 0 and -1. The expression allows to verify the Q value of the starting state, and in discrete domains it could be used to minimize the variance of classification actions, by directly learning the correct value instead of stochastic rewards.

3 EXPERIMENTAL RESULTS

We evaluate our algorithm on three datasets – MiniBooNE particle identification [8], Forest CoverType [8] and CIFAR-10 [5]. We prepared and split the datasets into training/validation/testing sets according to [13], 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 λ, and different values correspond to different average feature cost. Datasets information is summarized in Table 1.

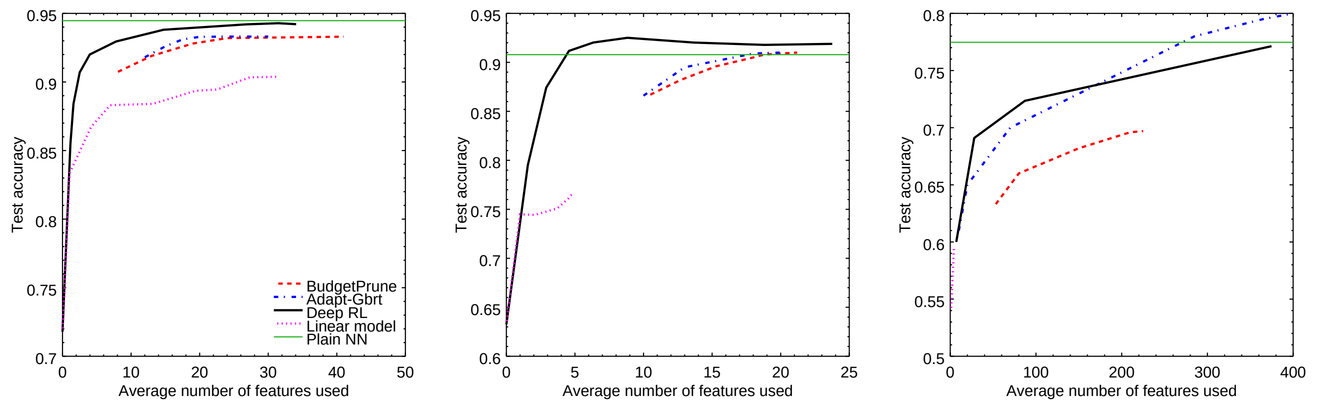
We directly compare to prior art algorithms Adapt-Gbrt [13] and BudgetPrune [15], which are both decision trees based algorithms, and also to the linear model used in [2]. We do not report other algorithms that were already shown to perform worse in [13] or [15]. We retrieved their performance from the 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 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

Table 1: Evaluated datasets

Parameter	MiniBooNE	Forest	CIFAR-10
Features	50	54	400
Classes	2	$2^\dagger$	$2^\dagger$
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

† Classes in these datasets are binarized to be comparable to results reported in prior art.



(a) MiniBooNE (b) Forest CoverType (c) CIFAR-10

Figure 1: Algorithm performance in tested datasets.

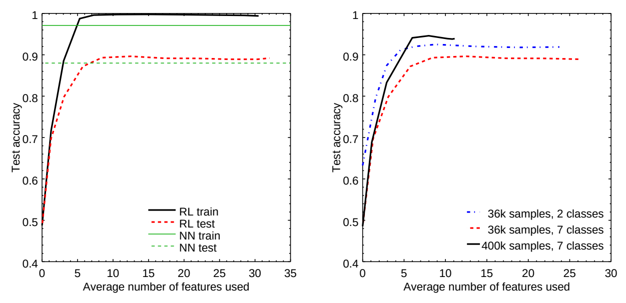
accuracy provides an estimate of an upper bound, to which our algorithm should approach. For this baseline network, we used three layers with RELU activation, with the same size as in our algorithm for a particular dataset, and an output layer with one output per class with a softmax activation.

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[1] 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. For different values of λ, our algorithm gives a distinct trade-off point between the average feature cost and classification accuracy. At any point, it performs better compared to other algorithms, as shown in Figure 1 (a). The linear model performs substantially worse than the deep model. The accuracy of our algorithm 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. It has 54 features – 10 real-valued and 2 categorical, encoded into binary vectors of length 4 and 40. 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 λ (see Figure 1 (b)). The linear model achieves its top performance with about 5 features and stops at this point, not able to utilize more.

An interesting fact is that the algorithm's best classification accuracy exceeds the accuracy of the baseline network. This suggests that our algorithm might work as some sort of regularization. Figure 2 (a) shows a further comparison between our algorithm and the baseline classifier, and suggests that some features can actually hurt performance. The baseline classifier using all features does



(a) Training vs. test dataset (b) Different dataset versions

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.

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 classification dataset. It was preprocessed and classes were binarized according to [13]. 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. The linear model is unable to acquire more then 5 features on average and fails to make different trade-offs. Our algorithm converges to the limit of the baseline classifier. In this dataset we could possibly boost the performance by arranging the features into a grid pattern and use convolutional networks, which are known to work well for this

[1]Transition means one step of an agent, generating one tuple (s, a, r, s').

type of problems [6, 11]. We do not use this approach in this work because we want to maintain the same architecture for all datasets.

3.1 Algorithm Properties

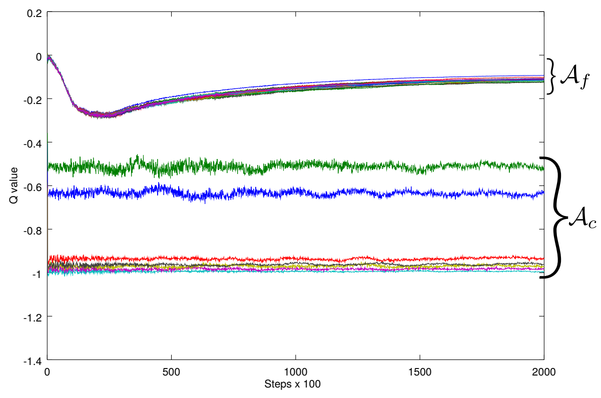
In Figure 3 (a), we observe how the Q values of the initial observed state s_0 evolve during the training on the multiclass Forest dataset. The algorithm quickly estimates the correct Q values for all classification actions $a \in \mathcal{A}_c$, since their value depends only on the distribution of classes. 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.

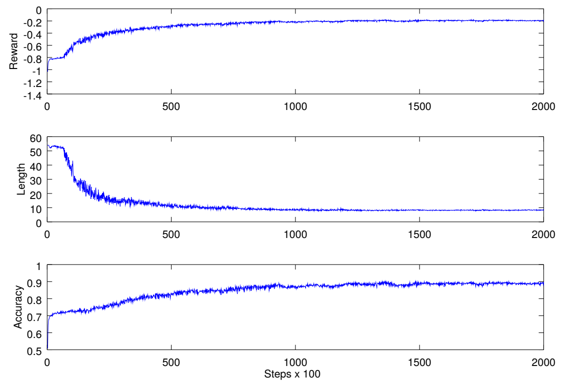
Figure 3 (b) in the middle and Figure 4 (a) shows the evolution of the average number of acquired features, which is equivalent to the average episode length, in this case. The algorithm starts by selecting all the features at first and then decreasing their amount. This is a result of optimistic initialization. The Q values of classification actions are estimated quickly into their range between -1 and 0. On the contrary, the Q values for feature selection actions change slowly and are around zero at the beginning of the training. At some point, the algorithm finds a core subset of essential features and then add to it to improve accuracy, as shown in Figure 4.

Figure 5 shows 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. The histogram additionally suggests that there are three modalities, corresponding to easy, average and difficult samples.

Figure 6 (a) shows how the parameter λ affects the trade-off between accuracy and average feature cost. Higher λ forces the algorithm to select fewer features on average, and as a consequence, the accuracy falls down. In Figure 6 (b) we report the effect of the neural network size on the performance. A larger network performs better for any λ value.



(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.

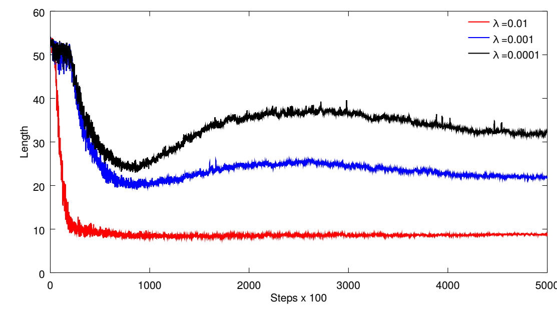
4 RELATED WORK

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

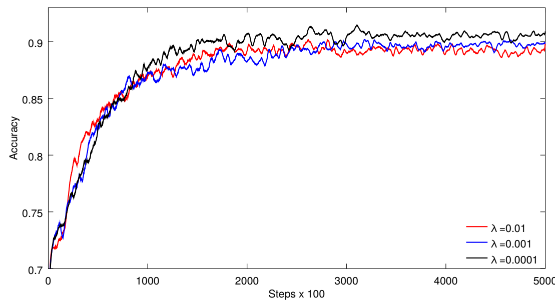
Authors of [1] use a recurrent neural network that uses attention to select blocks of features and classifies after a fixed number of steps. Compared to our work, decisions of this network are hard to explain. On the other hand, our Q-learning based algorithm produces semantically meaningful values for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

There is a plethora of tree-based algorithms [7, 13–15, 23–25]. The article [13] implements Adapt-Gbrt 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 this concrete instantiation, Adapt-Gbrt, 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 be even used together with our algorithm.

A different set of algorithms employed Linear Programming (LP) to this domain [20, 21]. The algorithm presented in [20] 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.

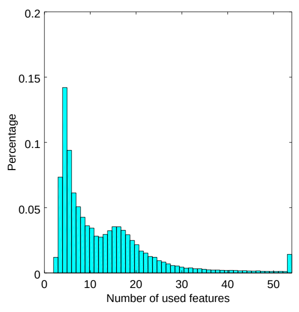


(a) Average number of used features

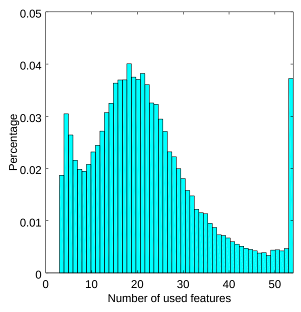


(b) Average accuracy

Figure 4: Effect of different λ on the average number of acquired features and accuracy on multiclass Forest dataset.



(a) $\lambda = 0.005$

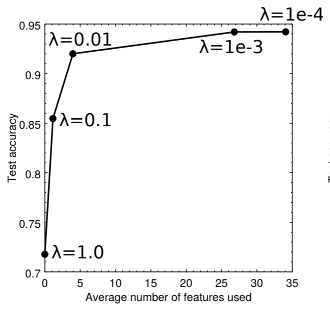


(b) $\lambda = 0.001$

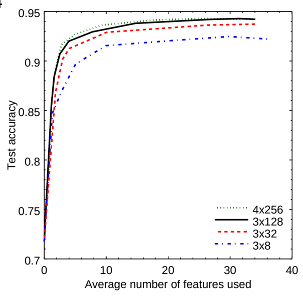
Figure 5: Histogram of used features on the multiclass Forest dataset, with different λ. On both (a) and (b) we see three peaks, corresponding to very easy, average and difficult samples.

The algorithm also chooses a model based on the complexity of the sample, similarly to [13].

In [22], 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.



(a) Trade-off by λ



(b) Network sizes

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

Reference [18] 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 [16] analyzes a problem similar to our definition of CwCF. 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 use Q-learning algorithm implemented with neural networks and show that even this basic reinforcement learning algorithm exceeds the performance of prior-art algorithms on several datasets. Moreover, the architecture of the used neural network can be easily adapted to a specific domain, e.g., to use convolutions in case of images, which is likely to further improve performance.

Our algorithm can be easily extended with any improvement in the deep learning or reinforcement learning. Performance can be improved by implementing extensions to the underlying DQN algorithm [4], or possibly by using longer traces [12]. Combination with complementary algorithms [13, 22] could also improve performance. As a future work, the algorithm could be extended to active learning domain, where not all labels are available.

ACKNOWLEDGMENTS

This research was supported by The Czech Science Foundation (grants no. 18-21409S and 18-27483Y). 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., $a^* = \operatorname{argmax}_a Q^*(s, a)$, then:*

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

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

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

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'^*. □

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:*

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

Proof. Let $p(a) = P[a \mid (x,a) \in \mathcal{D}(s)]$ denote the ratio of class a in the set of samples consistent with the observed state s.

$$\begin{aligned} Q^*(s,a) &= r(s,a) \\ &= 0 \cdot p(a) + (-1) \cdot (1 - p(a)) \\ &= p(a) - 1 \end{aligned}$$

All classification actions terminate the episode, so that the Q^* value is merely the expected reward. The correct classification happens, in expectation, with probability $p(a)$ and the reward is 0. Incorrect classification happens with probability $(1 - p(a))$ and the reward is -1. □

Table A.1: Algorithm Parameters

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

* One step means simulating all 1000 agents for one step.

† Epsilon controls exploration in the ϵ-greedy policy. The learning starts with ϵ_{start} and linearly decreases to ϵ_{end} within ϵ_{steps} steps.

‡ The learning rate is initialized to LR_{start} and it is multiplied by $\text{LR}_{\text{factor}}$ every LR_{steps} steps, until it reaches its minimum LR_{end}.

REFERENCES

[1] Contardo, Gabriella ; Denoyer, Ludovic ; Artieres, Thierry: Recurrent neural networks for adaptive feature acquisition. In: *International Conference on Neural Information Processing* Springer (Veranst.), 2016, S. 591–599

[2] Dulac-Arnold, Gabriel ; Denoyer, Ludovic ; Preux, Philippe ; Gallinari, Patrick: Datum-wise classification: a sequential approach to sparsity. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases* Springer (Veranst.), 2011, S. 375–390

[3] Hasselt, Hado V.: Double Q-learning. In: *Advances in Neural Information Processing Systems*, 2010, S. 2613–2621

[4] Hessel, Matteo ; Modayil, Joseph ; Hasselt, Hado van ; Schaul, Tom ; Ostrovski, Georg ; Dabney, Will ; Horgan, Dan ; Piot, Bilal ; Azar, Mohammad ; Silver, David: Rainbow: Combining Improvements in Deep Reinforcement Learning. In: *arXiv preprint arXiv:1710.02298* (2017)

[5] Krizhevsky, Alex ; Hinton, Geoffrey: *Learning multiple layers of features from tiny images*, University of Toronto, Diplomarbeit, 2009

[6] Krizhevsky, Alex ; Sutskever, Ilya ; Hinton, Geoffrey E.: Imagenet classification with deep convolutional neural networks. In: *Advances in neural information processing systems*, 2012, S. 1097–1105

[7] Kusner, Matt J. ; Chen, Wenlin ; Zhou, Quan ; Xu, Zhixiang E. ; Weinberger, Kilian Q. ; Chen, Yixin: Feature-Cost Sensitive Learning with Submodular Trees of Classifiers. In: *AAAI*, 2014, S. 1939–1945

[8] Lichman, M.: *UCI Machine Learning Repository*. 2013. – URL http://archive.ics.uci.edu/ml

[9] Lillicrap, Timothy P. ; Hunt, Jonathan J. ; Pritzel, Alexander ; Heess, Nicolas ; Erez, Tom ; Tassa, Yuval ; Silver, David ; Wierstra, Daan: Continuous control with deep reinforcement learning. In: *arXiv preprint arXiv:1509.02971* (2015)

[10] Mnih, Volodymyr ; Badia, Adria P. ; Mirza, Mehdi ; Graves, Alex ; Lillicrap, Timothy ; Harley, Tim ; Silver, David ; Kavukcuoglu, Koray: Asynchronous methods for deep reinforcement learning. In: *International Conference on Machine Learning*, 2016, S. 1928–1937

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

[12] Munos, Rémi ; Stepleton, Tom ; Harutyunyan, Anna ; Bellemare, Marc: Safe and efficient off-policy reinforcement learning. In: *Advances in Neural Information Processing Systems*, 2016, S. 1054–1062

[13] Nan, Feng ; Saligrama, Venkatesh: Adaptive Classification for Prediction Under a Budget. In: *Advances in Neural Information Processing Systems*, 2017, S. 4730–4740

[14] Nan, Feng ; Wang, Joseph ; Saligrama, Venkatesh: Feature-Budgeted Random Forest. In: *International Conference on Machine Learning*, 2015, S. 1983–1991

[15] Nan, Feng ; Wang, Joseph ; Saligrama, Venkatesh: Pruning random forests for prediction on a budget. In: *Advances in Neural Information Processing Systems*, 2016, S. 2334–2342

[16] Tan, Ming: Cost-sensitive learning of classification knowledge and its applications in robotics. In: *Machine Learning* 13 (1993), Nr. 1, S. 7–33

[17] Tieleman, Tijmen ; Hinton, Geoffrey: Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. In: *COURSERA: Neural networks for machine learning* 4 (2012), Nr. 2, S. 26–31

[18] Trapeznikov, Kirill ; Saligrama, Venkatesh: Supervised sequential classification under budget constraints. In: *Artificial Intelligence and Statistics*, 2013, S. 581–589

[19] Van Hasselt, Hado ; Guez, Arthur ; Silver, David: Deep Reinforcement Learning with Double Q-Learning. In: *AAAI*, 2016, S. 2094–2100

[20] Wang, Joseph ; Bolukbasi, Tolga ; Trapeznikov, Kirill ; Saligrama, Venkatesh: Model selection by linear programming. In: *European Conference on Computer Vision* Springer (Veranst.), 2014, S. 647–662

[21] Wang, Joseph ; Trapeznikov, Kirill ; Saligrama, Venkatesh: An lp for sequential learning under budgets. In: *Artificial Intelligence and Statistics*, 2014, S. 987–995

[22] Wang, Joseph ; Trapeznikov, Kirill ; Saligrama, Venkatesh: Efficient learning by directed acyclic graph for resource constrained prediction. In: *Advances in Neural Information Processing Systems*, 2015, S. 2152–2160

[23] Xu, Zhixiang ; Kusner, Matt ; Weinberger, Kilian ; Chen, Minmin: Cost-sensitive tree of classifiers. In: *International Conference on Machine Learning*, 2013, S. 133–141

[24] Xu, Zhixiang ; Weinberger, Kilian ; Chapelle, Olivier: The greedy miser: Learning under test-time budgets. In: *arXiv preprint arXiv:1206.6451* (2012)

[25] Xu, Zhixiang E. ; Kusner, Matt J. ; Weinberger, Kilian Q. ; Chen, Minmin ; Chapelle, Olivier: Classifier cascades and trees for minimizing feature evaluation cost. In: *Journal of Machine Learning Research* 15 (2014), Nr. 1, S. 2113–2144