Noname manuscript No.
(will be inserted by the editor)

Learning to Control a Structured-Prediction Decoder for Detection of HTTP-Layer DDoS Attackers

Uwe Dick · Tobias Scheffer

the date of receipt and acceptance should be inserted later

Abstract We focus on the problem of detecting clients that attempt to exhaust server resources by flooding a service with protocol-compliant HTTP requests. Attacks are usually coordinated by an entity that controls many clients. Modeling the application as a structured-prediction problem allows the prediction model to jointly classify a multitude of clients based on their cohesion of otherwise inconspicuous features. Since the resulting output space is too vast to search exhaustively, we employ greedy search and techniques in which a parametric controller guides the search. We apply a known method that sequentially learns the controller and the structured-prediction model. We then derive an online policy-gradient method that finds the parameters of the controller and of the structured-prediction model in a joint optimization problem; we obtain a convergence guarantee for the latter method. We evaluate and compare the various methods based on a large collection of traffic data of a web-hosting service.

1 Introduction

Distributed denial-of-service (DDoS) flooding attacks [37] intend to prevent legitimate users from using a web-based service by exhausting server or network resources. DDoS attacks can target the network level or the application level. One way for attackers to target the network level is to continuously request TCP connections and leave the connection in an incomplete state, which eventually exhausts the number of connections which the server can handle; this is called SYN flooding. Adaptive SYN-received timeouts, packet-filtering policies, and an increasing network capacity are making it more difficult to mount successful network-level attacks [26,37]. By comparison, server resources such as CPU, I/O bandwidth, database and disk throughput are becoming easier targets [4,28]. Attackers turn towards HTTP-layer flooding attacks in which they flood services with protocol-compliant

U. Dick
University of Potsdam, Department of Computer Science, Potsdam, Germany
E-mail: uwedick@cs.uni-potsdam.de

T. Scheffer
University of Potsdam, Department of Computer Science, Potsdam, Germany
E-mail: tobias.scheffer@uni-potsdam.de

requests that require the execution of scripts, expensive database operations, or the transmission of large files.

HTTP-layer attacks are more difficult to detect, because the detection mechanism ultimately has to decide whether all connecting clients have a legitimate reason for requesting a service in a particular way. In protocol-compliant application-level attacks, attackers have to sign their TCP/IP packets with their real IP address, because they have to complete the TCP handshake. One can therefore defend against flooding attacks by blacklisting offending IP addresses at the network router, provided that attacking clients can be singled out.

In order to detect attacking clients, one can engineer features of individual clients, train a classifier on labeled traffic data to detect attacking clients, and blacklist detected attackers. We follow this approach and evaluate it empirically, but the following considerations already indicate that it might work less than perfectly in practice. An individual protocol-compliant request is rarely conspicuous by itself; after all, the service is there to be requested. Most individual clients only post a small number of requests to a domain after which their IP address is not seen again. This implies that classification of individual clients will be difficult, and that aggregating information over requests into longitudinal client features [28,36,22] will only provide limited additional information.

However, DDoS attacks are usually coordinated by an entity that controls the attacking clients. Their joint programming is likely to induce some behavioral coherence of all attacking clients. Features of individual clients cannot reflect this cohesion. But a joint feature function that is parametrized with all clients x_i that interact with a domain and conjectured class labels y_i for all clients can measure the behavioral variance of all clients that are labeled as attackers. Structured-prediction methods [20,32] match this situation because they are based on joint feature functions of multiple dependent inputs x_i and their output values y_i. At application time, structured-prediction models have to solve the *decoding* problem of maximizing the decision function over all combinations of class labels. If the dependencies in the feature function are sequential or tree-structured, this maximization can be carried out efficiently using, for instance, the Viterbi algorithm for sequential data. In general as well as in this particular case, however, exhaustive search of the output space is intractable. Moreover, in our application environment, the search has to terminate after a fixed but *a priori* unknown number of computational steps due to a real-time constraint.

Collective classification algorithms [23] conduct a greedy search for the highest-scoring joint labeling of the nodes of a graph. They do so by iteratively relabeling individual nodes given the conjectured labels of all neighboring nodes. We will apply this principle, and explore the resulting algorithm empirically. More generally, when exhaustive search for a structured-prediction problem is infeasible, an *undergenerating decoder* can still search a constrained part of the output space [13]. Explicit constraints that make the remaining output space exhaustively searchable may also exclude good solutions. One may instead resort to learning a search heuristic. HC search [10,11] first learns a heuristic that guides the search to the correct output for training instances, and then uses this heuristic to control the decoder during training and application of the structured-prediction model. We will apply this principle to our application, and study the resulting algorithm.

The search heuristic of the HC-search framework is optimized to guide the decoder from an initial labeling to the correct output for all training instances. It is subsequently applied to guiding the decoder to the output that maximizes the decision function of the structured-prediction model, while this model is being learned. But the decision function is an imperfect model of the input-output relationship in the training data, especially while the parameters of the decision function are still being optimized. One may argue that a heuristic that does well at guiding the search to the correct output (that is known for the training instances) may do poorly at guiding it to the output that maximizes some decision function. We will therefore derive a policy-gradient model in which the controller and the structured-prediction model that uses the controller are learned in a joint optimization problem; we will analyze convergence properties of this model.

Defense mechanisms against DDoS attacks have so far been evaluated using artificial or semi-artificial traffic data that have been generated under plausible model assumptions of benign and malicious traffic [28,36,22,8]. By contrast, we will compare all models under investigation on a large data set of network traffic that we collect in a large shared web hosting environment and classify manually. It includes unusual high-volume network traffic for more than 1,546 domains over 22,645 time intervals of 10 seconds in which we observe several million connections of more than 450,000 unique clients.

The rest of the paper is structured as follows. Section 2 derives the problem setting from our motivating application. We model the application as an anomaly-detection problem in Section 3, as the problem of independently classifying clients in Section 4, as a collective classification problem in Section 5, and as a structured-prediction problem with a parametric decoder in Section 6. Section 7 discusses how all methods can be instantiated for the attacker-identification application. We present an empirical study in Section 8; Section 9 discusses our results against the background of related work. Section 10 concludes.

2 Problem Setting, Motivating Application

This section first lays out the relevant details of the application and establishes a high-level problem setting that will be cast into various learning paradigms in the following sections.

We focus on *HTTP-layer denial-of-service flooding attacks* [37], which we define to be any malicious attempts at denying the service to its legitimate users by posting protocol-compliant HTTP requests so as to exhaust any computational resource, such as CPU, bandwidth, or database throughput. Our application environment is a shared web hosting service in which a large number of domains are hosted in a large computing center. Each domain continuously receives requests from many legitimate or attacking clients. A domain is constituted by the top-level and second-level domain in the HOST field of the HTTP header ("example.com"); a client is identified by its IP address.

The effects of an attack can be mitigated when the IP addresses of the attacking clients can be identified: IP addresses of known attackers can be temporarily blacklisted at the router. Anomalous traffic events can extend for as little as a few minutes; attacks can run for several hours. The high-level view of the system consists of three parts: the *web servers* the *blacklisting mechanism*, and the *DDoS-attacker-detection* mechanism that decides which clients should be blacklisted.

The *blacklisting mechanism* resides at the main routers. It maintains a blacklist of IP addresses, and filters incoming traffic by blocking any TCP/IP packets from clients on that list. Blacklisting client IP addresses is the only feasible mitigation mechanism in our case. If requests from attacking IP addresses were to be processed, inspected, and filtered based on the individual payload, the servers would not be relieved sufficiently under an attack.

The *attacker-detection* mechanism listens to all TCP traffic between the *web servers* and *blacklisting* entity. Since attackers usually target a specific domain, we split the overall attacker-detection problem into an independent sub-problem for each domain. This allows us to distribute the *attacker-detection* mechanism over multiple computing nodes, each of which handles a subset of domains. As long as the number of connections to a domain per unit of time, the number of clients that interact with the domain, and the estimated CPU load used by a domain lie below safe lower bounds, the *attacker-detection mechanism* can rule out the possibility of a DDoS attack to that domain and excludes its traffic from further processing. If one of the thresholds is exceeded for some domain, then the *attacker-detection mechanism* processes the traffic to that domain in batches of 10 seconds. In each 10-seconds interval, the output is a list of IP addresses that should be blacklisted. This list is forwarded to the *blacklisting* mechanism which takes the actual blacklisting action.

Hence, for each domain, we arrive at an independent learning problem that can be described abstractly by an unknown distribution $p(\mathbf{x}, \mathbf{y})$ over sets $\mathbf{x} \in \mathcal{X}$ of clients x_j that interact with the domain within a 10-seconds interval and output variables $\mathbf{y} \in \mathcal{Y}(\mathbf{x}) = \{-1, 1\}^m$ which label each individual client $x_j \in \mathbf{x}$ as legitimate ($y_j = -1$) or attacker ($y_j = +1$). The number of observed clients $x_j \in \mathbf{x}$ may be different in each time interval. In Sections 3 and 4, we will pursue approaches in which each client x_j is individually represented by a vector $\mathbf{\Phi}_{\mathbf{x}}(x_j)$ that may depend on absolute features of x_j as well as on features of x_j that are measured relatively to the set of all clients $\mathbf{x}$ that currently interact with the domain. In Sections 5 and 6, we will represent the entire set of clients $\mathbf{x}$ and a candidate labeling $\mathbf{y}$ in a single joint feature representation $\mathbf{\Phi}(\mathbf{x}, \mathbf{y})$ and thereby arrive at a structured-prediction problem.

The following example illustrates why the problem of labeling sets $\mathbf{x} \in \mathcal{X}$ of clients x_j that interact with the same domain within a time interval can be modeled as a structured-prediction problem. Consider that an attacker controls a large network of client computers distributed around the world. The attacker tries to exhaust the database capacity of a domain by posting new-user registration requests. Each individual client posts only three such requests, which is inconspicuous. It would be virtually impossible for a classifier to identify the individual requests as being malicious, because each one of them is protocol-compliant and lacks any salient or unusual property.

A structured prediction model, on the other hand, can take joint attributes $\mathbf{\Phi}(\mathbf{x}, \mathbf{y})$ of sets of clients into account. For instance, since all attacking clients post similar new-user registration requests, the inner-group standard deviation of the URL string length will be much smaller for the attacking clients than for mixed sets of attacking and legitimate clients. A structured-prediction model can assign a negative weight to a feature that measures the inner-group standard deviation of the URL string length for all clients that are labeled as attackers. It can therefore learn to label clients in such a way that groups with small inner-group standard deviation of certain traffic parameters tend to have the same class label. We will discuss the feature representation that we employ for independent classification and for structured-prediction models in Section 7.4.

The classification problem for each 10-seconds interval has to be solved within ten seconds—otherwise, a backlog of decisions could build up, especially under an attack. The number of CPU cycles that are available within these 10 seconds is not known *a priori* because it depends on the overall server load. For the structured-prediction models, we encode this *anytime constraint* by limiting the number of search steps to a random number T that is governed by some distribution. We can disregard this anytime constraint for models that treat clients as independent (Sections 3 and 4), because the resulting classifiers are sufficiently fast at calculating the predictions.

Misclassified legitimate requests can potentially result in lost business while misclassified abusive requests consume computational resources; when CPU capacity, bandwidth, or database throughput capacities are exhausted, the service becomes unavailable. The resulting costs will be reflected in the optimization criteria by cost terms of false-negative and false-positive decisions. When the true labels of the clients $\mathbf{x}$ are $\mathbf{y}$, a prediction of $\hat{\mathbf{y}}$ incurs costs $c(\mathbf{x}, \mathbf{y}, \hat{\mathbf{y}}) \geq 0$. We will detail the exact cost function in Section 7.

With the exception of the anomaly-detection models that we will discuss in Section 3, training the *attacker-detection* model requires labeled training data. Section 8.1 describes the largely manual process in which we determine which client IP addresses are in fact attackers.

3 Anomaly Detection

In our application, an abundance of network traffic can be observed. However, manually labeling clients as legitimate and attackers is an arduous effort (see Section 8.1). Therefore, our first take is to model attacker detection as an anomaly-detection problem.

3.1 Problem Setting for Anomaly Detection

In this formulation of the problem settings, the set of clients $\mathbf{x} = \{x_1, \ldots, x_m\}$ that are observed in each 10-seconds interval is decomposed into individual clients x_j. At application time, clients are labeled independently based on the value of a parametric decision function $f_\phi(\mathbf{\Phi}_\mathbf{x}(x_j))$ which is a function of feature vector $\mathbf{\Phi}_\mathbf{x}(x_j)$. We will define feature vector $\mathbf{\Phi}_\mathbf{x}(x_j)$ in Section 7.4.2; for instance, it includes the number of different resource paths that client x_j has accessed, the number of HTTP requests that have resulted in error codes, both in terms of absolute counts and in proportion to all clients that connect to the domain.

At learning time, an unlabeled sample $\mathbf{x}_1, \ldots, \mathbf{x}_n$ of sets of clients is available. Most of the clients in the training data are legitimate, but some fraction consists of attacking clients. The unlabeled training instances are pooled into a set of feature vectors

$$L^{AD} = \bigcup_{i=1}^{n} \{\mathbf{\Phi}_{\mathbf{x}_i}(x_{i,1}), \ldots, \mathbf{\Phi}_{\mathbf{x}_i}(x_{i,m_i})\}; \tag{1}$$

training results in model parameters ϕ.

3.2 Support Vector Data Description

Support-vector data description (SVDD) is an anomaly-detection method that uses *unlabeled* data to find a model for *unusual* instances. The decision function of *SVDD* is

$$f_\phi^{SVDD}(\mathbf{\Phi}_\mathbf{x}(x_j)) = ||\mathbf{\Phi}_\mathbf{x}(x_j) - \phi||^2; \tag{2}$$

that is, *SVDD* classifies a client as an attacker if the distance between feature vector $\mathbf{\Phi}_\mathbf{x}(x_j)$ and the parameter vector ϕ that describes *normal traffic* exceeds a threshold r.

$$\hat{y}_j = \begin{cases} -1 & \text{if } f_\phi^{SVDD}(\mathbf{\Phi}_\mathbf{x}(x_j)) \leq r \\ +1 & \text{else} \end{cases} \tag{3}$$

4 Independent Classification

This section models the application as a standard classification problem.

4.1 Problem Setting for Independent Classification

Clients $\mathbf{x} = \{x_1, \dots, x_m\}$ of each 10-seconds interval are treated as independent observations, described by feature vectors $\mathbf{\Phi}_{\mathbf{x}}(x_j)$. As in Section 3.1, these vector representations are classified independently, based on the value of a parametric decision function $f_{\boldsymbol{\phi}}(\mathbf{\Phi}_{\mathbf{x}}(x_j))$. However, features may be engineered to depend on properties of all clients that interact with the domain in the time interval.

In the independent classification model, misclassification costs have to decompose into a sum over individual clients: $c(\mathbf{x}, \mathbf{y}, \hat{\mathbf{y}}) = \sum_{j=1}^{m} c(x_j, y_j, \hat{y}_j)$. At learning time, a labeled sample $(\mathbf{x}_1, \mathbf{y}_1), \dots, (\mathbf{x}_n, \mathbf{y}_n)$ is available. Each pair $(\mathbf{x}_i, \mathbf{y}_i)$ contains instances $x_{i,1}, \dots, x_{i,m_i}$ and corresponding labels $y_{i,1}, \dots, y_{i,m_i}$. The training data are pooled into independent pairs of feature vectors and corresponding class labels

$$L^{IC} = \bigcup_{i=1}^{n} \{(\mathbf{\Phi}_{\mathbf{x}_i}(x_{i,1}), y_{i,1}), \dots, (\mathbf{\Phi}_{\mathbf{x}_i}(x_{i,m_i}), y_{i,m_i})\}, \tag{4}$$

and training results in model parameters $\boldsymbol{\phi}$.

4.2 Logistic Regression

Logistic regression (LR) is a linear classification model that we use to classify clients independently. The decision function $f_{\boldsymbol{\phi}}^{LR}(\mathbf{\Phi}_{\mathbf{x}}(x_j))$ of logistic regression squashes the output of a linear model into a normalized probability by using a logistic function:

$$f_{\boldsymbol{\phi}}^{LR}(\mathbf{\Phi}_{\mathbf{x}}(x_j)) = \frac{1}{1 + e^{-\boldsymbol{\phi}^\top \mathbf{\Phi}_{\mathbf{x}}(x_j)}}. \tag{5}$$

Labels are assigned according to

$$\hat{y}_j = \begin{cases} -1 & \text{if } f_{\boldsymbol{\phi}}^{LR}(\mathbf{\Phi}_{\mathbf{x}}(x_j)) \leq \frac{1}{2} \\ +1 & \text{else} \end{cases} \tag{6}$$

Logistic regression models are trained by maximizing the regularized conditional log-likelihood of the training class labels over the parameters $\boldsymbol{\phi}$. Costs are incorporated by weighting the conditional log-likelihood of each observation with the cost of misclassifying it.

5 Structured Prediction with Approximate Inference

In Section 4, the decision function has been evaluated independently for each client. This prevented the model from taking joint features of particular groups of clients based on its predicted labels into account.

5.1 Problem Setting for Structured Prediction with Approximate Inference

In the structured-prediction paradigm, a classification model infers a collective assignment $\mathbf{y}$ of labels to the entirety of clients $\mathbf{x}$ that are observed in a time interval. In our application, all clients that interact with the domain in the time interval are dependent. The model therefore has to label the nodes of a fully connected graph. This problem setting is also referred to as *collective classification* [23].

Predictions $\hat{\mathbf{y}}$ of all clients are determined as the argument $\mathbf{y}$ that maximizes a decision function $f_{\boldsymbol{\phi}}(\mathbf{x}, \mathbf{y})$ which may depend on a joint feature vector $\mathbf{\Phi}(\mathbf{x}, \mathbf{y})$ of inputs and outputs. The feature vector may reflect arbitrary dependencies between all clients $\mathbf{x}$ and all labels $\mathbf{y}$. At application time, the decoding problem

$$\hat{\mathbf{y}} \approx \operatorname*{argmax}_{\mathbf{y} \in \mathcal{Y}(\mathbf{x})} f_{\boldsymbol{\phi}}(\mathbf{x}, \mathbf{y}) \tag{7}$$

has to be solved approximately within an interval of 10 seconds. The number of processing cycles that are available for each decision depends on the overall server load. We model this by constraining the number of steps which can be spent on approximating the highest-scoring output to T plus a constant number, where $T \sim p(T|\tau)$ is governed by some distribution and its value is not known in advance. At training time, a labeled sample $L = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)\}$ is available.

5.2 Iterative Classification Algorithm

The *iterative classification algorithm (ICA)* [24] is a standard collective-classification method. We use *ICA* as a method of approximate inference for structured prediction. *ICA* uses a feature vector $\mathbf{\Phi}_{\mathbf{x},\mathbf{y}}(x_j)$ for individual nodes and internalizes labels of neighboring nodes into this feature vector. For this definition of features, decision function $f_{\boldsymbol{\phi}}(\mathbf{x}, \mathbf{y})$ is a sum over all nodes. For a binary classification problem, we can use logistic regression and the decision function simplifies to

$$f_{\boldsymbol{\phi}}(\mathbf{x}, \mathbf{y}) = \sum_j \begin{cases} f^{LR}_{\boldsymbol{\phi}'}(\mathbf{\Phi}_{\mathbf{x},\mathbf{y}}(x_j)) & \text{if } y_j = +1 \\ 1 - f^{LR}_{\boldsymbol{\phi}'}(\mathbf{\Phi}_{\mathbf{x},\mathbf{y}}(x_j)) & \text{if } y_j = -1. \end{cases} \tag{8}$$

ICA only approximately maximizes this sum by starting an initial assignment $\hat{\mathbf{y}}$ which, in our, case, is determined by *logistic regression*. It then iteratively changes labels $\hat{y}_j$ such that the summand for j is maximized, until a fixed point is reached or the maximization is terminated after T steps. When a fixed point $\hat{\mathbf{y}}$ is reached, then $\hat{\mathbf{y}}$ satisfies

$$\forall j : \hat{y}_j = \begin{cases} -1 & \text{if } f^{LR}_{\boldsymbol{\phi}}(\mathbf{\Phi}_{\mathbf{x},\hat{\mathbf{y}}}(x_j)) \leq \frac{1}{2} \\ +1 & \text{otherwise.} \end{cases} \tag{9}$$

6 Structured Prediction with a Parametric Decoder

In this section, we allow for a guided search of the label space. Since the space is vastly large, we allow the search do be guided by a parametric model that itself is optimized on the training data.

6.1 Problem Setting for Structured Prediction with Parametric Decoder

At application time, prediction $\hat{\mathbf{y}}$ is determined by solving the decoding problem of Equation 7; decision function $f_\phi(\mathbf{x}, \mathbf{y})$ depends on a feature vector $\boldsymbol{\Phi}(\mathbf{x}, \mathbf{y})$. The decoder is allowed T (plus a constant number of) evaluations of the decision function, where $T \sim p(T|\tau)$ is governed by some distribution and its value is not known in advance. The decoder has parameters ψ that control this choice of labelings.

In the available T time steps, the decoder has to create a set of candidate labelings $Y_T(\mathbf{x})$ for which the decision function is evaluated. The decoding process starts in a state $Y_0(\mathbf{x})$ that contains a constant number of labelings. In each time step $t+1$, the decoder can choose an *action* a_{t+1} from the *action space* A_{Y_t}; this space should be designed to be much smaller than the label space $\mathcal{Y}(\mathbf{x})$. Action a_{t+1} creates another labeling $\mathbf{y}_{t+1}$; this additional labeling creates successor state $Y_{t+1}(\mathbf{x}) = a_{t+1}(Y_t(\mathbf{x})) = Y_t(\mathbf{x}) \cup \{\mathbf{y}_{t+1}\}$.

In a basic definition, A_{Y_t} could consist of actions $\alpha_{\mathbf{y}j}$ (for all $\mathbf{y} \in Y_t$ and $1 \leq j \leq n_\mathbf{x}$, where $n_\mathbf{x}$ is the number of clients in $\mathbf{x}$) that take output $\mathbf{y} \in Y_t(\mathbf{x})$ and generate labeling $\bar{\mathbf{y}}$ by flipping the labeling of the j-th client; output $Y_{t+1}(\mathbf{x}) = Y_t(\mathbf{x}) \cup \{\bar{\mathbf{y}}\}$ is $Y_t(\mathbf{x})$ plus this modified output. This definition would allow the entire space $\mathcal{Y}(\mathbf{x})$ to be reached from any starting point. In our experiments, we will construct an action space that contains application-specific state transactions such as *flip the labels of the k addresses that have the most open connections*—see Section 7.3.

The choice of action a_{t+1} is based on parameters ψ of the decoder, and on a feature vector $\boldsymbol{\Psi}(\mathbf{x}, Y_t(\mathbf{x}), a_{t+1})$; for instance, actions may be chosen by following a stochastic policy $a_{t+1} \sim \pi_\psi(\mathbf{x}, Y_t(\mathbf{x}))$. We will define feature vector $\boldsymbol{\Psi}(\mathbf{x}, Y_t(\mathbf{x}), a_{t+1})$ in Section 7.4.4; for instance, it may contain the difference between the geographical distribution of clients whose label is changed by action a_{t+1} and the geographical distribution of all clients with that same label. Choosing an action a_{t+1} requires an evaluation of $\boldsymbol{\Psi}(\mathbf{x}, Y_t(\mathbf{x}), a_{t+1})$ for each possible action in $A_{Y_t(\mathbf{x})}$. Our problem setting is most useful for applications in which evaluation of $\boldsymbol{\Psi}(\mathbf{x}, Y_t(\mathbf{x}), a_{t+1})$ takes less time than evaluation of $\boldsymbol{\Phi}(\mathbf{x}, \mathbf{y}_{t+1})$—otherwise, it might be better to evaluate the decision function for a larger set of randomly drawn outputs than to spend time on selecting outputs for which the decision function should be evaluated. Feature vector $\boldsymbol{\Psi}(\mathbf{x}, Y_t(\mathbf{x}), a_{t+1})$ may contain a computationally inexpensive subset of $\boldsymbol{\Phi}(\mathbf{x}, \mathbf{y}_{t+1})$.

After T steps, the decoding process is terminated. At this point, the decision-function values f_ϕ of a set of candidate outputs $Y_T(\mathbf{x})$ have been evaluated. Prediction $\hat{\mathbf{y}}$ is the argmax of the decision function over this set:

$$\hat{\mathbf{y}} = \operatorname*{argmax}_{\mathbf{y} \in Y_T(\mathbf{x})} f_\phi(\mathbf{x}, \mathbf{y}). \tag{10}$$

At training time, a labeled sample $L = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)\}$ is available.

6.2 HC Search

HC search [9] is an approach to structured prediction that learns parameters ψ of a search heuristic, and then uses a decoder with this search heuristic to learn parameters ϕ of a structured-prediction model (the decision function f_ϕ is called the cost-function in HC-search terminology). We apply this principle to our problem setting.

At application time, the decoder produces labeling $\hat{\mathbf{y}}$ that approximately maximizes $f_{\boldsymbol{\phi}}(\mathbf{x},\mathbf{y})$ as follows. The starting point $Y_0(\mathbf{x})$ of each decoding problem contains the labeling produced by the *logistic regression* classifier (see Section 4.2). Action $a_{t+1} \in A_{Y_t}$ is chosen deterministically as the maximum of the search heuristic $f_{\psi}(\boldsymbol{\Psi}(\mathbf{x}, Y_t(\mathbf{x}), a_{t+1})) = \boldsymbol{\psi}^\top \boldsymbol{\Psi}(\mathbf{x}, Y_t(\mathbf{x}), a_{t+1})$. After T steps, the argmax $\hat{\mathbf{y}}$ of $f_{\boldsymbol{\phi}}(\mathbf{x},\mathbf{y})$ over all outputs in $Y_T(\mathbf{x}) = a_T(\ldots a_1(Y_0(\mathbf{x}))\ldots)$ (Equation 7) is returned as prediction.

At training time, *HC search* first learns a search heuristic with parameters $\boldsymbol{\psi}$ as follows. Let $L_{\boldsymbol{\psi}}$ be an initially empty set of training constraints for the heuristic. For each training instance $(\mathbf{x}_i, \mathbf{y}_i)$, starting state $Y_0(\mathbf{x}_i)$ contains the labeling produced by the *logistic regression* classifier (see Section 4.2). Time t is then iterated from 1 to an upper bound $\bar{T}$ on the number of time steps that will be available for decoding at application time. Then, iteratively, all elements a_{t+1} of the finite action space $A_{Y_t}(\mathbf{x}_i)$ and their corresponding outputs $\mathbf{y}'_{t+1}$ are enumerated and the action a^*_{t+1} that leads to the lowest-cost output $\mathbf{y}'_{t+1}$ is determined. Since the training data are labeled, the actual costs of labeling $\mathbf{x}_i$ as $\mathbf{y}'_{t+1}$ when the correct labeling would be $\mathbf{y}_i$ can be determined by evaluating the cost function. Search heuristic f_{ψ} has to assign a higher value to a^*_{t+1} than to any other a_{t+1}, and the costs $c(\mathbf{x}_i, \mathbf{y}_i, \mathbf{y}'_{t+1})$ of choosing a poor action should be included in the optimization problem. Hence, for each action $a_{t+1} \in A_{Y_t}(\mathbf{x}_i)$, constraint

$$\begin{aligned} &f_{\psi}(\boldsymbol{\Psi}(\mathbf{x}_i, Y_t(\mathbf{x}_i), a^*_{t+1})) - f_{\psi}(\boldsymbol{\Psi}(\mathbf{x}_i, Y_t(\mathbf{x}_i), a_{t+1})) \\ &\qquad > \sqrt{c(\mathbf{x}_i, \mathbf{y}_i, \mathbf{y}'_{t+1}) - c(\mathbf{x}_i, \mathbf{y}_i, \mathbf{y}^*_{t+1})} \end{aligned} \tag{11}$$

is added to $L_{\boldsymbol{\psi}}$. Model $\boldsymbol{\psi}$ should satisfy the constraints in $L_{\boldsymbol{\psi}}$. We use a soft-margin version of the constraints in $L_{\boldsymbol{\psi}}$ and squared slack-terms which results in a cost-sensitive multi-class SVM (actions a are the classes) with margin scaling [32].

After parameters $\boldsymbol{\psi}$ have been fixed, parameters $\boldsymbol{\phi}$ of structured-prediction model $f_{\boldsymbol{\phi}}(\mathbf{x},\mathbf{y}) = \boldsymbol{\phi}^\top \boldsymbol{\Phi}(\mathbf{x},\mathbf{y})$ are trained on the training data set of input-output pairs $(\mathbf{x}_i, \mathbf{y}_i)$ using SVM-struct with margin rescaling and using the search heuristic with parameters $\boldsymbol{\psi}$ as decoder. Negative pseudo-labels are generated as follows. For each $(\mathbf{x}_i, \mathbf{y}_i) \in L$, heuristic $\boldsymbol{\psi}$ is applied $\bar{T}$ times to produce a sequence of output sets $Y_0(\mathbf{x}_i), \ldots, Y_{\bar{T}}(\mathbf{x}_i)$. When $\bar{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y} \in Y_{\bar{T}}(\mathbf{x}_i)} \boldsymbol{\phi}^\top \boldsymbol{\Phi}(\mathbf{x},\mathbf{y}) \neq \mathbf{y}_i$ violates the cost-rescaled margin, then a new training constraint is added, and parameters $\boldsymbol{\phi}$ are optimized to satisfy these constraints.

6.3 Online Policy-Gradient Decoder

The decoder of *HC search* has been trained to locate the labeling $\hat{\mathbf{y}}$ that minimizes the costs $c(\mathbf{x}_i, \mathbf{y}_i, \hat{\mathbf{y}})$ for given true labels. However, it is then applied to finding candidate labelings for which $f_{\boldsymbol{\phi}}(\mathbf{x},\mathbf{y})$ is evaluated with the goal of maximizing $f_{\boldsymbol{\phi}}$. However, since the decision function $f_{\boldsymbol{\phi}}$ may be an imperfect approximation of the input-output relationship that is reflected in the training data, labelings that minimize the costs $c(\mathbf{x}_i, \mathbf{y}_i, \hat{\mathbf{y}})$ might be different from outputs that maximize the decision function. We will now derive a closed optimization problem in which decoder and structured-prediction model are jointly optimized. We will study its convergence properties theoretically.

We now demand that during the decoding process, the decoder chooses action $a_{t+1} \in A_{Y_t}$ which generates successor state $Y_{t+1}(\mathbf{x}) = a_{t+1}(Y_t(\mathbf{x}))$ according to a *stochastic* policy, $a_{t+1} \sim \pi_{\boldsymbol{\psi}}(\mathbf{x}, Y_t(\mathbf{x}))$, with parameter $\boldsymbol{\psi} \in \mathbb{R}^{m_2}$ (where m_2 is the dimensionality of the decoder feature space) and features $\boldsymbol{\Psi}(\mathbf{x}, Y_t(\mathbf{x}), a_{t+1})$. At time T, the prediction is the highest-scoring output from $Y_T(\mathbf{x})$ according to Equation 7.

The learning problem is to find parameters $\boldsymbol{\phi}$ and $\boldsymbol{\psi}$ that minimize the expected costs over all inputs, outputs, and numbers of available decoding steps:

$$\operatorname*{argmin}_{\boldsymbol{\phi},\boldsymbol{\psi}} \mathbb{E}_{(\mathbf{x},\mathbf{y}),T,Y_T(\mathbf{x})} \left[c(\mathbf{x},\mathbf{y},\operatorname*{argmax}_{\hat{\mathbf{y}}\in Y_T(\mathbf{x})} f_{\boldsymbol{\phi}}(\mathbf{x},\hat{\mathbf{y}}) \right] \tag{12}$$

$$\text{with } (\mathbf{x},\mathbf{y}) \sim p(\mathbf{x},\mathbf{y}), \quad T \sim p(T|\tau) \tag{13}$$

$$Y_T(\mathbf{x}) \sim p(Y_T(\mathbf{x})|\pi_{\boldsymbol{\psi}},\mathbf{x},T) \tag{14}$$

The costs $c(\mathbf{x},\mathbf{y},\hat{\mathbf{y}})$ of the highest-scoring element $\hat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y}'\in Y_T(\mathbf{x})} f_{\boldsymbol{\phi}}(\mathbf{x},\mathbf{y}')$ may not be differentiable in $\boldsymbol{\phi}$. Let therefore loss $\ell(\mathbf{x},\mathbf{y},Y_T(\mathbf{x});\boldsymbol{\phi})$ be a differentiable approximation of the cost that $\boldsymbol{\phi}$ induces on the set $Y_T(\mathbf{x})$. Section 7.2 instantiates the loss for the motivating problem. Distribution $p(\mathbf{x},\mathbf{y})$ is unknown. Given training data $S = \{(\mathbf{x}_1,\mathbf{y}_1),\ldots,(\mathbf{x}_m,\mathbf{y}_m)\}$, we approximate the expected costs (Equation 12) by the regularized expected empirical loss with convex regularizers $\Omega_{\boldsymbol{\phi}}$ and $\Omega_{\boldsymbol{\psi}}$:

$$\boldsymbol{\phi}^*,\boldsymbol{\psi}^* = \operatorname*{argmin}_{\boldsymbol{\phi},\boldsymbol{\psi}} \sum_{(\mathbf{x},\mathbf{y})\in S} V_{\boldsymbol{\phi},\boldsymbol{\psi},\tau}(\mathbf{x},\mathbf{y}) + \Omega_{\boldsymbol{\phi}} + \Omega_{\boldsymbol{\psi}} \tag{15}$$

$$\text{with } V_{\boldsymbol{\phi},\boldsymbol{\psi},\tau}(\mathbf{x},\mathbf{y}) = \sum_{T=1}^{\infty}\Big(p(T|\tau)\sum_{Y_T(\mathbf{x})} p(Y_T(\mathbf{x})|\pi_{\boldsymbol{\psi}},\mathbf{x},T)\ell(\mathbf{x},\mathbf{y},Y_T(\mathbf{x});\boldsymbol{\phi})\Big). \tag{16}$$

Equation 15 still cannot be solved immediately because it contains a sum over all values of T and all sets $Y_T(\mathbf{x})$. To solve Equation 15, we will liberally borrow ideas from the field of reinforcement learning. First, we will derive a formulation of the gradient $\nabla_{\boldsymbol{\psi},\boldsymbol{\phi}} V_{\boldsymbol{\psi},\boldsymbol{\phi},\tau}(\mathbf{x},\mathbf{y})$. The gradient still involves an intractable sum over all sequences of actions, but its formulation suggests that it can be approximated by sampling action sequences according to the stochastic policy. By using a baseline function—which are a common tool in reinforcement learning [16]—we can reduce the variance of this sampling process.

Let $a_{1\ldots T} = a_1,\ldots,a_T$ with $a_{t+1} \in A_{Y_t}$ be a sequence of actions that executes a transition from $Y_0(\mathbf{x})$ to $Y_T(\mathbf{x}) = a_T(\ldots(a_1(Y_0(\mathbf{x})))\ldots)$. The available computation time is finite and hence $p(T|\tau) = 0$ for all $T > \bar{T}$ for some $\bar{T}$. We can rewrite Equation 16:

$$V_{\boldsymbol{\phi},\boldsymbol{\psi},\tau}(\mathbf{x},\mathbf{y}) = \sum_{a_{1\ldots\bar{T}}}\Big(p(a_{1\ldots\bar{T}}|\boldsymbol{\psi},Y_0(\mathbf{x}))\sum_{T=1}^{\bar{T}} p(T|\tau)\ell(\mathbf{x},\mathbf{y},a_T(\ldots(a_1(Y_0(\mathbf{x})))\ldots);\boldsymbol{\phi})\Big),$$

$$\text{with } \quad p(a_{1\ldots\bar{T}}|\boldsymbol{\psi},Y_0(\mathbf{x})) = \prod_{t=1}^{\bar{T}} \pi_{\boldsymbol{\psi}}(a_t|\mathbf{x},a_{t-1}(\ldots(Y_0(\mathbf{x})\ldots)). \tag{17}$$

Equation 18 defines $D_{\ell,\tau}$ as the partial gradient $\nabla_{\boldsymbol{\phi}}$ of the expected empirical loss for an action sequence $a_1,\ldots,a_{\bar{T}}$ that has been sampled according to $p(a_{1,\ldots,\bar{T}}|\boldsymbol{\psi},Y_0(\mathbf{x}))$.

$$D_{\ell,\tau}(a_{1\ldots\bar{T}},Y_0(\mathbf{x});\boldsymbol{\phi}) = \sum\nolimits_{T=1}^{\bar{T}} p(T|\tau)\nabla_{\boldsymbol{\phi}}\ell(\mathbf{x},\mathbf{y},a_T(\ldots(a_1(Y_0(\mathbf{x})))\ldots);\boldsymbol{\phi}) \tag{18}$$

The policy gradient $\nabla_{\boldsymbol{\psi}}$ of a summand of Equation 17 is

$$\nabla_{\boldsymbol{\psi}} p(a_{1\ldots\bar{T}}|\boldsymbol{\psi},Y_0(\mathbf{x}))\sum_{T=1}^{\bar{T}} p(T|\tau)\ell(\mathbf{x},\mathbf{y},a_T(\ldots(a_1(Y_0(\mathbf{x})))\ldots);\boldsymbol{\phi})$$

$$= \Big(p(a_{1\ldots\bar{T}}|\boldsymbol{\psi}, Y_0(\mathbf{x})) \sum_{T=1}^{\bar{T}} \nabla_{\boldsymbol{\psi}} \log \pi_{\boldsymbol{\psi}}(a_T|\mathbf{x}, a_{T-1}(\ldots(a_1(Y_0(\mathbf{x}))\ldots))\Big) \tag{19}$$
$$\sum_{T=1}^{\bar{T}} p(T|\tau)\ell(\mathbf{x},\mathbf{y},a_T(\ldots(a_1(Y_0(\mathbf{x})))\ldots);\boldsymbol{\phi}).$$

Equation 19 uses the "log trick" $\nabla_\psi p = p\log\nabla_\psi p$; it sums the gradients of all actions and scales with the accumulated loss of all initial subsequences. Baseline functions [16] reflect the intuition that a_T is not responsible for losses incurred prior to T; also, relating the loss to the expected loss for all sequences that contain a_T reflects the merit of a_T better. Equation 20 defines the policy gradient for an action sequence sampled according to $p(a_{1\ldots\bar{T}}|\boldsymbol{\psi}, Y_0(\mathbf{x}))$, modified by baseline function B.

$$E_{\ell,B,\tau}(a_{1\ldots\bar{T}}, Y_0(\mathbf{x});\boldsymbol{\psi},\boldsymbol{\phi})$$
$$= \sum_{T=1}^{\bar{T}} \nabla_{\boldsymbol{\psi}} \log \pi_{\boldsymbol{\psi}}(a_T|\mathbf{x}, a_{T-1}(\ldots(a_1(Y_0(\mathbf{x})))\ldots)) \tag{20}$$
$$\left(\sum_{t=T}^{\bar{T}} p(t|\tau)\ell(\mathbf{x},\mathbf{y},a_t(\ldots(a_1(Y_0(\mathbf{x})))\ldots);\boldsymbol{\phi}) - B(a_{1\ldots T-1}, Y_0(\mathbf{x});\boldsymbol{\psi},\boldsymbol{\phi},\mathbf{x})\right)$$

Lemma 1 (General gradient) *Let $V_{\boldsymbol{\phi},\boldsymbol{\psi},\tau}(\mathbf{x},\mathbf{y})$ be defined as in Equation 17 for a differentiable loss function ℓ. Let $D_{\ell,\tau}$ and $E_{\ell,B,\tau}$ be defined in Equations 18 and 20 for any scalar baseline function $B(a_{1\ldots T}, Y_0(\mathbf{x});\boldsymbol{\psi},\boldsymbol{\phi},\mathbf{x})$. Then the gradient of $V_{\boldsymbol{\phi},\boldsymbol{\psi},\tau}(\mathbf{x},\mathbf{y})$ is*

$$\nabla_{\boldsymbol{\phi},\boldsymbol{\psi}} V_{\boldsymbol{\phi},\boldsymbol{\psi},\tau}(\mathbf{x},\mathbf{y}) = \sum_{a_{1\ldots\bar{T}}} p(a_{1\ldots\bar{T}}|\boldsymbol{\psi}, Y_0(\mathbf{x}))$$
$$\left[E_{\ell,B,\tau}(a_{1\ldots\bar{T}}, Y_0(\mathbf{x});\boldsymbol{\psi},\boldsymbol{\phi})^\top, D_{\ell,\tau}(a_{1\ldots\bar{T}}, Y_0(\mathbf{x});\boldsymbol{\phi})^\top\right]^\top \tag{21}$$

Proof The gradient stacks the partial gradients of $\boldsymbol{\psi}$ and $\boldsymbol{\phi}$ above each other. The partial gradient $\nabla_{\boldsymbol{\phi}} V_{\boldsymbol{\phi},\boldsymbol{\psi},\tau}(\mathbf{x},\mathbf{y}) = \sum_{a_{1\ldots\bar{T}}} p(a_{1\ldots\bar{T}}|\boldsymbol{\psi}, Y_0(\mathbf{x}))D_{\ell,\tau}(a_{1\ldots\bar{T}}, Y_0(\mathbf{x});\boldsymbol{\phi})$ follows from Equation 18. The partial gradient $\nabla_{\boldsymbol{\psi}} V_{\boldsymbol{\phi},\boldsymbol{\psi},\tau}(\mathbf{x},\mathbf{y}) = \sum_{a_{1\ldots\bar{T}}} p(a_{1\ldots\bar{T}}|\boldsymbol{\psi}, Y_0(\mathbf{x}))E_{\ell,B,\tau}(a_{1\ldots\bar{T}}, Y_0(\mathbf{x});\boldsymbol{\psi},\boldsymbol{\phi})$ is a direct application of the Policy Gradient Theorem [31,27] for episodic processes.

The choice of a baseline function B influences the variance of the sampling process, but not the gradient; a lower variance means faster convergence. Let $E_{\ell,B,\tau,T}$ be a summand of Equation 20 with a value of T. Variance $\mathbb{E}[(E_{\ell,B,\tau,T}(a_{1\ldots\bar{T}}, Y_0(\mathbf{x});\boldsymbol{\psi},\boldsymbol{\phi}) - \mathbb{E}[E_{\ell,B,\tau,T}(a_{1\ldots\bar{T}}, Y_0(\mathbf{x});\boldsymbol{\psi},\boldsymbol{\phi})|a_{1..T}])^2|a_{1..T}]$ is minimized by the baseline that weights the loss of all sequences starting in $a_T(\ldots(a_1(Y_0(\mathbf{x})))\ldots)$ by the squared gradient [16]:

$$B_{\mathrm{G}}(a_{1\ldots T}, Y_0(\mathbf{x});\boldsymbol{\psi},\boldsymbol{\phi},\mathbf{x}) = \frac{\sum_{a_{T+1}} G(a_{1..T+1}, Y_0)^2 Q(a_{1..T+1}, Y_0)}{\sum_{a_{T+1}} G(a_{1..T+1}, Y_0)^2} \tag{22}$$

$$\text{with } Q(a_{1..T+1}, Y_0) = \mathop{\mathbb{E}}_{a_{T+2\ldots\bar{T}}}\left[\sum_{t=T+1}^{\bar{T}} p(t|\tau)\ell(\mathbf{x},\mathbf{y},a_t(\ldots(Y_0(\mathbf{x}))\ldots);\boldsymbol{\phi})\middle| a_{1..T+1}\right] \tag{23}$$

$$\text{and } G(a_{1..T+1}, Y_0) = \nabla \log \pi_\psi(a_{T+1}|\mathbf{x}, a_T(\ldots(a_1(Y_0(\mathbf{x})))\ldots)). \tag{24}$$

This baseline function is intractable because it (intractably) averages the loss of all action sequences that start in state $Y_T(\mathbf{x}) = a_T(\dots a_1(Y_0(\mathbf{x}))\dots)$ with the squared length of the gradient of their first action a_T. Instead, the assumption that the expected loss of all sequences starting at T is half the loss of state $Y_T(\mathbf{x})$ gives the approximation:

$$B_{\mathrm{HL}}(a_{1\dots T}, Y_0(\mathbf{x}); \boldsymbol{\psi}, \boldsymbol{\phi}, \mathbf{x}) = \frac{1}{2}\sum\nolimits_{t=T+1}^{\bar{T}} p(t)\ell(\mathbf{x}, \mathbf{y}, a_T(\dots a_1(Y_0(\mathbf{x}))\dots); \boldsymbol{\phi}). \quad (25)$$

We will refer to the policy-gradient method with baseline function B_{HL} as *online policy gradient with baseline*. Note that inserting baseline function

$$B_{\mathrm{R}}(a_{1\dots T}, Y_0(\mathbf{x}); \boldsymbol{\psi}, \boldsymbol{\phi}, \mathbf{x}) = -\sum_{t=1}^{T} p(t)\ell(\mathbf{x}, \mathbf{y}, a_t(\dots(a_1(Y_0(\mathbf{x})))\dots); \boldsymbol{\phi}) \quad (26)$$

into Equation 20 resolves each summand of Equation 21 to Equation 19, the unmodified policy gradient for $a_{1\dots,\bar{T}}$. We will refer to the online policy-gradient method with baseline function B_{R} as *online policy gradient without baseline*. Algorithm 1 shows the *online policy-gradient* learning algorithm. It optimizes parameters $\boldsymbol{\psi}$ and $\boldsymbol{\phi}$ using a stochastic gradient by sampling action sequences from the intractable sum over all action sequences of Equation 21 Theorem 1 proves its convergence under a number of conditions. The step size parameters $\alpha(i)$ have to satisfy

$$\sum\nolimits_{i=0}^{\infty} \alpha(i) = \infty \ , \ \sum\nolimits_{i=0}^{\infty} \alpha(i)^2 < \infty. \quad (27)$$

Loss function ℓ is required to be bounded. This can be achieved by constructing the loss function such that for large values it smoothly approaches some arbitrarily high ceiling C. However, in our case study we could not observe cases in which the algorithm does not converge for unbounded loss functions. Baseline function B is required to be differentiable and bounded for the next theorem. However, no gradient has to be computed in the algorithm. All baseline functions that are considered in Section 7 meet this demand.

Algorithm 1 Online Stochastic Policy-Gradient Learning Algorithm

```
Input: Training data $S$, starting parameters $\boldsymbol{\psi}_0, \boldsymbol{\phi}_0$.
1: let $i = 0$.
2: **repeat**
3:     Draw $(\mathbf{x}, \mathbf{y})$ uniformly from $S$
4:     Sample action sequence $a_{1\dots\bar{T}}$ with each $a_t \sim \pi_{\psi_i}(\mathbf{x}, a_{t-1}(\dots a_1(Y_0(\mathbf{x}))\dots))$.
5:     $\boldsymbol{\psi}_{i+1} = \boldsymbol{\psi}_i - \alpha(i)(E_{\ell,B}(a_{1\dots\bar{T}}, Y_0(\mathbf{x}); \boldsymbol{\psi}_i, \boldsymbol{\phi}_i) + \nabla_{\boldsymbol{\psi}}\Omega_{\boldsymbol{\psi}_i})$
6:     $\boldsymbol{\phi}_{i+1} = \boldsymbol{\phi}_i - \alpha(i)(D_{\ell,\tau}(a_{1\dots\bar{T}}, Y_0(\mathbf{x}); \boldsymbol{\psi}_i) + \nabla_{\boldsymbol{\phi}}\Omega_{\boldsymbol{\phi}_i})$
7:     increment $i$.
8: **until** convergence
Return $\boldsymbol{\psi}_i, \boldsymbol{\phi}_i$
```

Theorem 1 (Convergence of Algorithm 1) *Let the stochastic policy $\pi_{\boldsymbol{\psi}}$ be twice differentiable, let both $\pi_{\boldsymbol{\psi}}$ and $\nabla_{\boldsymbol{\psi}}\pi_{\boldsymbol{\psi}}$ be Lipschitz continuous, and let $\nabla_{\boldsymbol{\psi}}\log\pi_{\boldsymbol{\psi}}$ be bounded. Let step size parameters $\alpha(i)$ satisfy Equation 27. Let loss function ℓ be differentiable in $\boldsymbol{\phi}$ and both ℓ and $\nabla_{\boldsymbol{\phi}}\ell$ be Lipschitz continuous. Let ℓ be bounded. Let B be differentiable and both B and $\nabla_{\boldsymbol{\phi}\boldsymbol{\psi}}B$ be bounded. Let $\Omega_{\boldsymbol{\phi}} = \gamma_1\|\boldsymbol{\phi}\|^2, \Omega_{\boldsymbol{\psi}} = \gamma_2\|\boldsymbol{\psi}\|^2$. Then, Algorithm 1 converges with probability 1.*

Proof For space limitations and in order to improve readability, throughout the proof we omit dependencies on $\mathbf{x}$ and $Y_0(\mathbf{x})$ in the notations when dependence is clear from the context. For example, we use $p(a_{1..\bar{T}}|\boldsymbol{\psi})$ instead of $p(a_{1..\bar{T}}|\boldsymbol{\psi}, Y_0(\mathbf{x}))$. We use Theorem 2 from Chapter 2 and Theorem 7 from Chapter 3 of [6] to prove convergence. We first show that the full negative gradient $-\sum_{(\mathbf{x},\mathbf{y})} \nabla_{\boldsymbol{\psi},\boldsymbol{\phi}} V_{\boldsymbol{\psi}_i,\boldsymbol{\phi}_i,\tau}(\mathbf{x},\mathbf{y}) - [\gamma_2 \boldsymbol{\psi}_i^\top, \gamma_1 \boldsymbol{\phi}_i^\top]^\top$ is Lipschitz continuous.

Let $L(a_{T..\bar{T}}, \boldsymbol{\phi}) = \sum_{t=T}^{\bar{T}} p(t)\ell(\mathbf{x}, \mathbf{y}, a_t(..Y_0(\mathbf{x})..); \boldsymbol{\phi})$. We proceed by showing that $p(a_{1..\bar{T}}|\boldsymbol{\psi}) E_{\ell,B,\tau}(a_{1..\bar{T}}; \boldsymbol{\psi}, \boldsymbol{\phi}) = \sum_{T=1}^{\bar{T}} (p(a_{1..\bar{T}}|\boldsymbol{\psi})\ \nabla_{\boldsymbol{\psi}} \log \pi_{\boldsymbol{\psi}}(a_T|\boldsymbol{\psi})(L(a_{T..\bar{T}}, \boldsymbol{\phi}) - B(a_{1..T-1}, \boldsymbol{\psi}, \boldsymbol{\phi})))$ is Lipschitz in $[\boldsymbol{\psi}^\top, \boldsymbol{\phi}^\top]^\top$. It is differentiable in $[\boldsymbol{\psi}^\top, \boldsymbol{\phi}^\top]^\top$ per definition and it suffices to show that the derivative is bounded. By the product rule,

$$
\begin{aligned}
&\nabla_{\boldsymbol{\psi},\boldsymbol{\phi}}(p(a_{1..\bar{T}}|\boldsymbol{\psi})\nabla_{\boldsymbol{\psi}} \log \pi_{\boldsymbol{\psi}}(a_T|\boldsymbol{\psi})(L(a_{T..\bar{T}}, \boldsymbol{\phi}) - B(a_{1..T-1}, \boldsymbol{\psi}, \boldsymbol{\phi}))) \\
=&\nabla_{\boldsymbol{\psi},\boldsymbol{\phi}}(p(a_{1..\bar{T}}|\boldsymbol{\psi})\nabla_{\boldsymbol{\psi}} \log \pi_{\boldsymbol{\psi}}(a_T|\boldsymbol{\psi}))(L(a_{T..\bar{T}}, \boldsymbol{\phi}) - B(a_{1..T-1}, \boldsymbol{\psi}, \boldsymbol{\phi})) \qquad (28)\\
&+ p(a_{1..\bar{T}}|\boldsymbol{\psi})\nabla_{\boldsymbol{\psi}} \log \pi_{\boldsymbol{\psi}}(a_T|\boldsymbol{\psi})\nabla_{\boldsymbol{\psi},\boldsymbol{\phi}}(L(a_{T..\bar{T}}, \boldsymbol{\phi}) - B(a_{1..T-1}, \boldsymbol{\psi}, \boldsymbol{\phi})). \qquad (29)
\end{aligned}
$$

We can see that line 29 is bounded because $p, \nabla_{\boldsymbol{\psi}} \log \pi_{\psi}, \nabla_{\boldsymbol{\phi}} L$ and $\nabla_{\boldsymbol{\phi},\boldsymbol{\psi}} B$ are bounded by definition and products of bounded functions are bounded. Regarding line 28, we state that L and B are bounded by definition. Without loss of generality, let $T = 1$.

$$
\begin{aligned}
&\nabla_{\boldsymbol{\psi}}(p(a_{1..\bar{T}}|\boldsymbol{\psi})\nabla_{\boldsymbol{\psi}} \log \pi_{\boldsymbol{\psi}}(a_1|\boldsymbol{\psi})) \\
&= \nabla_{\boldsymbol{\psi}}(\nabla_{\boldsymbol{\psi}}\pi_{\boldsymbol{\psi}}(a_1|\boldsymbol{\psi})p(a_{2..\bar{T}}|a_1\boldsymbol{\psi})) \qquad (30)\\
&= p(a_{2..\bar{T}}|a_1\boldsymbol{\psi}))\nabla_{\boldsymbol{\psi}}\nabla_{\boldsymbol{\psi}}\pi_{\boldsymbol{\psi}}(a_1|\boldsymbol{\psi}) + \nabla_{\boldsymbol{\psi}}\pi_{\boldsymbol{\psi}}(a_1|\boldsymbol{\psi})\nabla_{\boldsymbol{\psi}} p(a_{2..\bar{T}}|a_1, \boldsymbol{\psi})) \qquad (31)
\end{aligned}
$$

Equation 30 follows from $p\nabla_{\psi} \log p = \nabla \boldsymbol{\psi} p$. The left summand of Equation 31 is bounded because both p and $\nabla_{\boldsymbol{\psi}}\nabla_{\boldsymbol{\psi}}\pi_{\boldsymbol{\psi}}$ are bounded by definition. Furthermore, $\nabla_{\boldsymbol{\psi}} p(a_{2..\bar{T}}|\boldsymbol{\psi}) = \nabla_{\boldsymbol{\psi}}\pi_{\boldsymbol{\psi}}(a_2)p(a_{3..\bar{T}}|\boldsymbol{\psi}) + \pi_{\boldsymbol{\psi}}(a_2)\nabla_{\boldsymbol{\psi}} p(a_{3..\bar{T}}|\boldsymbol{\psi})$ is bounded because $\nabla_{\boldsymbol{\psi}}\pi_{\boldsymbol{\psi}}(a_t)$ and $p(a_{t..\bar{T}}|\boldsymbol{\psi})$ are bounded for all t and we can expand $\nabla_{\boldsymbol{\psi}} p(a_{3..\bar{T}}|\boldsymbol{\psi})$ recursively. From this it follows that the right summand of Equatio 31 is bounded as well. Thus we have shown the above claim.

$p(a_{1..\bar{T}}|\boldsymbol{\psi})D_{\ell,\tau}(a_{1..\bar{T}}; \boldsymbol{\phi})$ is Lipschitz because $p(a_{1..\bar{T}}|\boldsymbol{\psi})$ is Lipschitz and bounded and $D_{\ell,\tau}$ is a sum of bounded Lipschitz functions. The product of two bounded Lipschitz functions is bounded. $[\gamma_1 \boldsymbol{\psi}^\top, \gamma_2 \boldsymbol{\phi}^\top]^\top$ is obviously Lipschitz as well, which concludes the considerations regarding the full negative gradient.

Let $M_{i+1} = [E_{\ell,B,\tau}(a_{1..\bar{T}}; \boldsymbol{\psi}_i, \boldsymbol{\phi}_i)^\top, D_{\ell,\tau}(a_{1..\bar{T}}; \boldsymbol{\psi}_i)^\top]^\top - \sum_{(\mathbf{x},\mathbf{y})} \nabla_{\boldsymbol{\psi},\boldsymbol{\phi}} V_{\boldsymbol{\psi}_I,\boldsymbol{\phi}_i,\tau}(\mathbf{x},\mathbf{y}))$, where $E_{\ell,B,\tau}(a_{1..\bar{T}}; \boldsymbol{\psi}_i, \boldsymbol{\phi}_i)$ and $D_{\ell,\tau}(a_{1..\bar{T}}; \boldsymbol{\psi}_i)$ are samples as computed by Algorithm 1. We show that $\{M_i\}$ is a Martingale difference sequence with respect to the increasing family of σ-fields $\mathcal{F}_i = \sigma([\boldsymbol{\phi}_0^\top, \boldsymbol{\psi}_0^\top]^\top, M_1, ..., M_i), i \geq 0$. That is, $\forall i \in \mathbb{N}$, $\mathbb{E}[M_{i+1}|\mathcal{F}_i] = 0$ *almost surely*, and $\{M_i\}$ are square-integrable with $\mathbb{E}[\|M_{i+1}\|^2|\mathcal{F}_i] \leq K(1 + \|[\boldsymbol{\phi}_i^\top, \boldsymbol{\psi}_i^\top]^\top\|^2)$ *almost surely*, for some $K > 0$.

$\mathbb{E}[M_{i+1}|\mathcal{F}_n] = 0$ is given by the definition of M_{i+1} above. We have to show $\mathbb{E}[\|M_{i+1}\|^2|\mathcal{F}_i] \leq K(1 + \|[\boldsymbol{\phi}_i^\top, \boldsymbol{\psi}_i^\top]^\top\|^2)$ for some K. We proceed by showing that for each $(\mathbf{x}, \mathbf{y}, a_{1..\bar{T}})$ it holds that $\|[E_{\ell,B,\tau}(a_{1..\bar{T}}; \boldsymbol{\psi}_i, \boldsymbol{\phi}_i)^\top, D_{\ell,\tau}(a_{1..\bar{T}}; \boldsymbol{\psi}_i)^\top]^\top\|^2 \leq K(1 + \|[\boldsymbol{\phi}_i^\top, \boldsymbol{\psi}_i^\top]^\top\|^2)$. From that it follows that

$$
\begin{aligned}
&\|\sum_{\mathbf{x},\mathbf{y}} \sum_{a_{1..\bar{T}}} p(a_{1..\bar{T}}|\boldsymbol{\psi})[E_{\ell,B,\tau}(a_{1..\bar{T}}; \boldsymbol{\psi}, \boldsymbol{\phi})^\top, D_{\ell,\tau}(a_{1..\bar{T}}; \boldsymbol{\phi})^\top]^\top\|^2 \\
&\qquad\qquad\qquad \leq K(1 + \|[\boldsymbol{\phi}_i^\top, \boldsymbol{\psi}_i^\top]^\top\|^2)
\end{aligned}
$$

and $\|M_{i+1}\|^2 \leq 4K(1 + \|[\boldsymbol{\phi}_i^\top, \boldsymbol{\psi}_i^\top]^\top\|^2)$ which proves the claim.

Regarding $E_{\ell,B,\tau}$, we assume that $\|\nabla \log \pi_{\boldsymbol{\psi}}\|^2$ is bounded by some K'' and it follows that $\| \sum_{T=1}^{\bar{T}} \nabla_{\boldsymbol{\psi}} \log \pi_{\boldsymbol{\psi}}(a_T|\mathbf{x}, a_{T-1}(..Y_0(\mathbf{x})..)\|^2$ is also bounded by $\bar{T}^2 K''$. $\|\ell(\mathbf{x}, \mathbf{y}, a_T(..Y_0(\mathbf{x})..); \boldsymbol{\phi})\|^2 \leq K'(1 + \|\boldsymbol{\phi}\|^2)$ and B bounded per assumption and thus $\sum_{t=T}^{\bar{T}} p(t)\ell(\mathbf{x}, \mathbf{y}, a_t(..Y_0(\mathbf{x})..)(\mathbf{x}); \boldsymbol{\phi}) - B(a_{1..T-1}; \boldsymbol{\phi}, \boldsymbol{\psi}) \leq \bar{K}'(1 + \|\boldsymbol{\phi}\|^2)$ with some $\bar{K}'$. It follows that $\|E_{\ell,B,\tau}(a_{1..\bar{T}}; \boldsymbol{\psi}_i, \boldsymbol{\phi}_i)\|^2 \leq 2^{\bar{T}} K'' \bar{K}'(1 + \|\boldsymbol{\phi}\|^2)$. As $\nabla_{\boldsymbol{\phi}} \ell(\mathbf{x}, \mathbf{y}, a_T(..Y_0(\mathbf{x})..); \boldsymbol{\phi})$ is bounded per assumption, $\|D_{\ell,\tau}\|^2 \leq K'''$ for some $K''' > 0$. The claim follows: $\|[E_{\ell,B,\tau}(a_{1..\bar{T}}; \boldsymbol{\psi}_i, \boldsymbol{\phi}_i)^\top, D_{\ell,\tau}(a_{1..\bar{T}}; \boldsymbol{\psi}_i)^\top]^\top\|^2 = \|E_{\ell,B,\tau}(a_{1..\bar{T}}; \boldsymbol{\psi}_i, \boldsymbol{\phi}_i)\|^2 + \|D_{\ell,\tau}(a_{1..\bar{T}}; \boldsymbol{\psi}_i)\|^2 \leq K''' + 2^{\bar{T}} K'' \bar{K}'(1 + \|\boldsymbol{\phi}\|^2) \leq K''' + \bar{T}^2 K'' \bar{K}'(1 + \|[\boldsymbol{\psi}^\top, \boldsymbol{\phi}^\top]^\top\|^2)$.

We can now use Theorem 2 from Chapter 2 of [6] to prove convergence by identifying function $h([\boldsymbol{\phi}_i^\top, \boldsymbol{\psi}_i^\top]^\top)$ as assumed in the assumptions of that theorem with the full negative gradient $-\sum_{(\mathbf{x},\mathbf{y})} \nabla_{\boldsymbol{\psi},\boldsymbol{\phi}} V_{\boldsymbol{\psi}_i,\boldsymbol{\phi}_i,\tau}(\mathbf{x}, \mathbf{y}) - \left[\gamma_2 \boldsymbol{\psi}_i^\top, \gamma_1 \boldsymbol{\phi}_i^\top\right]^\top$. The theorem states that the algorithm converges with probability 1 if the iterates $[\boldsymbol{\phi}_{i+1}^\top, \boldsymbol{\psi}_{i+1}^\top]^\top$ stay bounded.

Now, let $h_r(\xi) = h(r\xi)/r$. Next, we show that $\lim_{r\to\infty} h_r(\xi) = h_\infty(\xi)$ exists and that the origin in $\mathbb{R}^{m_1+m_2}$ is an asymptotically stable equilibrium for the o.d.e. $\dot{\xi}(t) = h_\infty(\xi(t))$. With this, Theorem 7 from Chapter 3 of [6]—originally from [7]—states that the iterates stay bounded and Algorithm 1 converges. Next, we show that h meets (A4):

$$
\begin{aligned}
&h_r(\boldsymbol{\phi}, \boldsymbol{\psi}) \\
&= \frac{1}{r} \sum_{\mathbf{x},\mathbf{y}} \sum_{a_{1..\bar{T}}} p(a_{1..\bar{T}}|r\boldsymbol{\psi}) \left[E_{\ell,B,\tau}(a_{1..\bar{T}}; r\boldsymbol{\psi}, r\boldsymbol{\phi})^\top, D_{\ell,\tau}(a_{1..\bar{T}}; r\boldsymbol{\phi})^\top\right]^\top \\
&\quad + 1/r\left[\gamma_1 r \boldsymbol{\psi}_i^\top, \gamma_2 r \boldsymbol{\phi}_i^\top\right]^\top \\
&= \sum_{\mathbf{x},\mathbf{y}} \sum_{a_{1..\bar{T}}} p(a_{1..\bar{T}}|r\boldsymbol{\psi}) \sum_{T=1}^{\bar{T}} \left[\nabla_{\boldsymbol{\psi}} \pi_{\boldsymbol{\psi}}(a_T|r\boldsymbol{\psi})^\top (L(a_{T..\bar{T}}, r\boldsymbol{\phi}) - B(a_{1..T-1}))/r,\right. \\
&\left. p(a_{1..\bar{T}}|r\boldsymbol{\psi}) D_{\ell,\tau}(a_{1..\bar{T}}; r\boldsymbol{\phi})^\top/r\right]^\top + \left[\gamma_1 \boldsymbol{\psi}_i^\top, \gamma_2 \boldsymbol{\phi}_i^\top\right]^\top,
\end{aligned}
$$

$\nabla_{\boldsymbol{\psi}} \pi_{\boldsymbol{\psi}}, L$ and B are all bounded and it follows that $p(a_{1..\bar{T}}|r\boldsymbol{\psi}) \sum_{T=1}^{\bar{T}} \left[\nabla_{\boldsymbol{\psi}} \pi_{\boldsymbol{\psi}}(a_T|r\boldsymbol{\psi})^\top (L(a_{T..\bar{T}}, r\boldsymbol{\phi}) - B(a_{1..T-1}))/r \rightarrow 0\right.$. The same holds for the other part as $p(a_{1..\bar{T}}|r\boldsymbol{\psi})$ and $D_{\ell,\tau}(a_{1..\bar{T}}; r\boldsymbol{\phi})$ are bounded. It follows that $h_\infty([\boldsymbol{\psi}^\top, \boldsymbol{\phi}^\top]^\top) = \left[\gamma_1 \boldsymbol{\psi}_i^\top, \gamma_2 \boldsymbol{\phi}_i^\top\right]^\top$. Therefore, the ordinary differential equation $\dot{\xi}(t) = h_\infty(\xi(t))$ has an asymptotically stable equilibrium at the origin, which shows that (A4) is valid.

7 Identification of DDoS Attackers

We will now implement a DDoS-attacker detection mechanism using the techniques that we derived in the previous sections. We engineer a cost function, suitable feature representations $\boldsymbol{\Phi}$ and $\boldsymbol{\Psi}$, policy π_ψ, and loss function ℓ that meet the demands of Theorem 1.

7.1 Cost Function

False-positive decisions (legitimate clients that are mistaken for attackers) lead to the temporary blacklisting of a legitimate user. This will result in unserved requests, and potentially lost business. False-negative decisions (attackers that are not recognized as such) will result in a wasteful allocation of server resources, and possibly in a successful DDoS attack that leaves the service unavailable for legitimate users. We decompose cost function $c(\mathbf{x}, \mathbf{y}, \hat{\mathbf{y}})$ for a set of clients $\mathbf{x}$ into the following parts.

We measure two cost-inducing parameters of false-negative decisions: the *number of connections* opened by attacking clients and the *CPU use* triggered by clients' requests. According to the experience of the data-providing web hosting service, the same damage is done by attacking clients that a) collectively initiate 200 connections per 10-seconds interval t and b) collectively initiate scripts that use 10 CPUs for 10 seconds. However, those costs are not linear in their respective attributes. Instead, only limited resources are available, such as a finite number of CPUs, and the rise in costs of two scripts that use 80% or 90%, resp., of all available CPUs is different from the rise in costs of two scripts that use 20% or 30% of CPUs. We define costs incurred by connections initiated by attackers to be quadratic in the number of connections. Similarly, costs for CPU usage are also quadratic.

The hosting service assesses that blocking a legitimate client incurs the same cost as opening 200 HTTP connections to attackers in an interval or wasting 100 CPU seconds. Also, by blocking 50 connections of legitimate client the same cost is added. Based on these requirements, we define costs

$$c(\mathbf{x}, \mathbf{y}, \hat{\mathbf{y}}) = \sum_{x_i : y_i = -1, \hat{y}_i = +1} 1 + \frac{1}{50} \times \text{\#connections by } x_i + \Big(\frac{1}{200} \sum_{x_i : y_i = +1, \hat{y}_i = -1} \text{\#connections by } x_i\Big)^2 \\ + \Big(\frac{1}{100} \sum_{x_i : y_i = +1, \hat{y}_i = -1} \text{CPU seconds initiated by } x_i\Big)^2 .$$

7.2 Loss Function

In order for the *online policy-gradient* method to converge, Theorem 1 states that loss functions ℓ need to be differentiable and both ℓ and $\nabla\ell$ have to be Lipschitz continuous. We discuss loss functions in this section. As mentioned in Section 6.3, the boundedness assumption on loss functions can be enforced by smoothly transitioning the loss function to a function that approaches some arbitrarily high ceiling C. We first define the difference in costs of a prediction $\hat{\mathbf{y}}$ and an optimal label $\mathbf{y}^*$ as $\rho(\hat{\mathbf{y}}, \mathbf{y}^*) = c(\mathbf{x}, \mathbf{y}, \hat{\mathbf{y}}) - c(\mathbf{x}, \mathbf{y}, \mathbf{y}^*)$. We denote the margin as $g_{\mathbf{x},\mathbf{y}}(\mathbf{y}^*, \hat{\mathbf{y}}; \boldsymbol{\phi}) = \sqrt{\rho(\hat{\mathbf{y}}, \mathbf{y}^*)} - \boldsymbol{\phi}^\top(\boldsymbol{\Phi}(\mathbf{x}, \hat{\mathbf{y}}) - \boldsymbol{\Phi}(\mathbf{x}, \mathbf{y}^*))$. The clipped squared hinge loss is differentiable:

$$h_{\mathbf{x},\mathbf{y}}(\mathbf{y}^*, \hat{\mathbf{y}}; \boldsymbol{\phi}) = \begin{cases} 0 & \text{if } \sqrt{\rho(\hat{\mathbf{y}}, \mathbf{y}^*)} \leq \boldsymbol{\phi}^\top(\boldsymbol{\Phi}(\mathbf{x}, \hat{\mathbf{y}}) - \boldsymbol{\Phi}(\mathbf{x}, \mathbf{y}^*)) \\ g_{\mathbf{x},\mathbf{y}}(\mathbf{y}^*, \hat{\mathbf{y}}; \boldsymbol{\phi})^2 & \text{if } 0 < \boldsymbol{\phi}^\top(\boldsymbol{\Phi}(\mathbf{x}, \hat{\mathbf{y}}) - \boldsymbol{\Phi}(\mathbf{x}, \mathbf{y}^*)) < \sqrt{\rho(\hat{\mathbf{y}}, \mathbf{y}^*)} \\ 2\boldsymbol{\phi}^\top(\boldsymbol{\Phi}(\mathbf{x}, \hat{\mathbf{y}}) - \boldsymbol{\Phi}(\mathbf{x}, \mathbf{y}^*)) + 2 & \text{if } \boldsymbol{\phi}^\top(\boldsymbol{\Phi}(\mathbf{x}, \hat{\mathbf{y}}) - \boldsymbol{\Phi}(\mathbf{x}, \mathbf{y}^*)) \leq 0 \end{cases}$$

Equation 32 defines the loss that $\boldsymbol{\phi}$ induces on $Y_T(\mathbf{x})$ as the average squared hinge loss of all labels in $Y_T(\mathbf{x})$ except the one with minimal costs, offset by these minimal costs.

$$\ell_h(\mathbf{x}, \mathbf{y}, Y_T(\mathbf{x}); \boldsymbol{\phi}) = c(\mathbf{x}, \mathbf{y}, \mathbf{y}^*) + \frac{1}{|Y_T(\mathbf{x}) - 1|} \sum_{\hat{\mathbf{y}} \in Y_T(\mathbf{x}), \hat{\mathbf{y}} \neq \mathbf{y}^*} h_{\mathbf{x},\mathbf{y}}(\mathbf{y}^*, \hat{\mathbf{y}}; \boldsymbol{\phi}) \tag{32}$$
$$\text{with } \mathbf{y}^* = \operatorname*{argmin}_{\hat{\mathbf{y}} \in Y_T} c(\mathbf{x}, \mathbf{y}, \hat{\mathbf{y}})$$

In contrast to the standard squared margin-rescaling loss for structured prediction that uses the hinge loss of the output that maximally violates the margin, here we average the Huber loss over all labels in $Y_T(\mathbf{x})$; this definition of ℓ_h is differentiable and Lipschitz continuous, as required by Theorem 1. *Online policy gradient* employs loss function ℓ_h in our experimentations. We will refer to *HC search* with loss function ℓ_h as *HC search with average margin* and will also conduct experiments with *HC search with max margin* by using the standard squared margin-rescaled loss

$$\ell_m(\mathbf{x}, \mathbf{y}, Y_T(\mathbf{x}), \boldsymbol{\phi}) = \max_{\hat{\mathbf{y}} \in Y_T(\mathbf{x}), \hat{\mathbf{y}} \neq \mathbf{y}^*} \max\{g_{\mathbf{x},\mathbf{y}}(\mathbf{y}^*, \hat{\mathbf{y}}; \boldsymbol{\phi}), 0\}^2. \tag{33}$$

7.3 Action Space and Stochastic Policy

This section defines the action space $A_{Y_i}(\mathbf{x}_i)$ of *HC search* and the *online policy-gradient* method as well as the stochastic policy $\pi_{\boldsymbol{\psi}}(\mathbf{x}, Y_t(\mathbf{x}))$ of *online policy-gradient*.

The action space is based on 21 rules $r \in R$ that can be instantiated for the elements $\mathbf{y} \in Y_t(\mathbf{x})$; the action space A_{Y_t} contains all instantiations $a_{t+1} = (r, \mathbf{y})$ that add a new labeling $r(\mathbf{y})$ to the successor state: $Y_{t+1}(\mathbf{x}) = Y_t(\mathbf{x}) \cup \{r(\mathbf{y})\}$. We define the initial set Y_0 to contain labels $\{-1\}^{n_\mathbf{x}}$ and $\{+1\}^{n_\mathbf{x}}$, where $n_\mathbf{x}$ is the number of clients in $\mathbf{x}$. Some of the following rules refer to the score of a binary classifier that classifies clients independently; we use the *logistic regression* regression classifier as described in Section 4.2 in our experiments.

– Switch the labels of the 1, 2, 5, or 10 clients from -1 to $+1$ that have the highest number of connections, the highest score of the baseline classifier, or CPU consumption. All combinations of these attributes yield 12 possible rules.
– Switch the labels of the client from -1 to 1 that has the second-highest number of connections, independent classifier score, or CPU consumption (3 rules).
– Switch the label of the client from 1 to -1 that has the lowest or second-lowest number of connections, baseline classifier score, or CPU consumption (6 rules).
– Switch all clients from -1 to $+1$ whose independent classifier score exceeds -1, -0.5, 0, 0.5, or 1 (5 rules).

Theorem 1 requires that the stochastic policy be twice differentiable in $\boldsymbol{\psi}$ and that both $\pi_{\boldsymbol{\psi}}$ and $\nabla_{\boldsymbol{\psi}} \pi_{\boldsymbol{\psi}}$be Lipschitz continuous. We define $\pi_{\boldsymbol{\psi}}$ as

$$\pi_{\boldsymbol{\psi}}(a_{t+1} | \mathbf{x}, Y_t(\mathbf{x})) = \frac{\exp(\boldsymbol{\psi}^\top \boldsymbol{\Psi}(\mathbf{x}, Y_t(\mathbf{x}), a_{t+1}))}{\sum_{a \in A_{Y_t}} \exp(\boldsymbol{\psi}^\top \boldsymbol{\Psi}(\mathbf{x}, Y_t(\mathbf{x}), a))}.$$

7.4 Feature Representations

We engineer features that refer to base traffic parameters that we explain in Section 7.4.1. From these base traffic parameters, we derive feature representations for all learning approaches that we study. Figure 1 gives an overview of all features.

7.4.1 Base Traffic Parameters

In each 10-seconds interval, we calculate base traffic parameters of each client that connects to the domain. For clients that connect to the domain over a longer duration, we calculate moving averages that are reset after two minutes of inactivity. On the TCP protocol level, we extract the absolute numbers of full connections, open connections, open and resent FIN

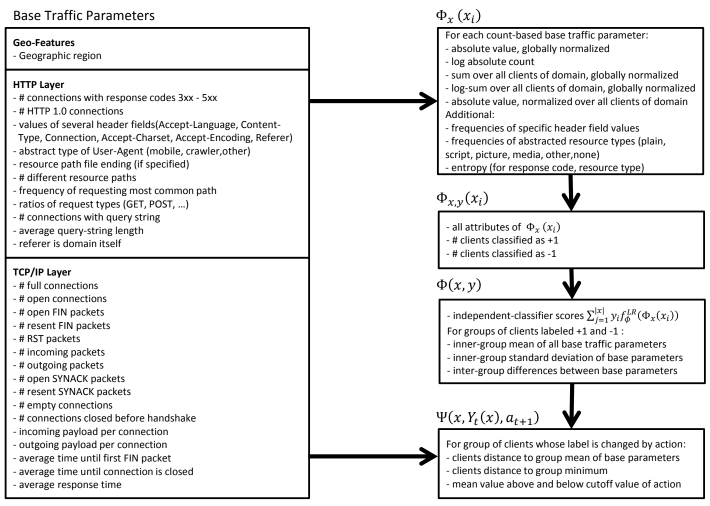
Base Traffic Parameters
Geo-Features
- Geographic region
HTTP Layer
- # connections with response codes 3xx - 5xx
- # HTTP 1.0 connections
- values of several header fields(Accept-Language, Content-Type, Connection, Accept-Charset, Accept-Encoding, Referer)
- abstract type of User-Agent (mobile, crawler,other)
- resource path file ending (if specified)
- # different resource paths
- frequency of requesting most common path
- ratios of request types (GET, POST, ...)
- # connections with query string
- average query-string length
- referer is domain itself
TCP/IP Layer
- # full connections
- # open connections
- # open FIN packets
- # resent FIN packets
- # RST packets
- # incoming packets
- # outgoing packets
- # open SYNACK packets
- # resent SYNACK packets
- # empty connections
- # connections closed before handshake
- incoming payload per connection
- outgoing payload per connection
- average time until first FIN packet
- average time until connection is closed
- average response time
$\Phi_x(x_i)$
For each count-based base traffic parameter:
- absolute value, globally normalized
- log absolute count
- sum over all clients of domain, globally normalized
- log-sum over all clients of domain, globally normalized
- absolute value, normalized over all clients of domain
Additional:
- frequencies of specific header field values
- frequencies of abstracted resource types (plain, script, picture, media, other,none)
- entropy (for response code, resource type)
$\Phi_{x,y}(x_i)$
- all attributes of $\Phi_x(x_i)$
- # clients classified as +1
- # clients classified as -1
$\Phi(x,y)$
- independent-classifier scores $\sum_{j=1}^{|x|} y_i f_\phi^{LR}(\Phi_x(x_i))$
For groups of clients labeled +1 and -1 :
- inner-group mean of all base traffic parameters
- inner-group standard deviation of base parameters
- inter-group differences between base parameters
$\Psi(x, Y_t(x), a_{t+1})$
For group of clients whose label is changed by action:
- clients distance to group mean of base parameters
- clients distance to group minimum
- mean value above and below cutoff value of action

Fig. 1 Feature representations

packets, timeouts, RST packets, incoming and outgoing packets, open and resent SYNACK packets, empty connections, connections that are closed before the handshake is completed, incoming and outgoing payload per connection. We determine the average durations until the first FIN packet is received and until the connection is closed, as well as the response time.

From the HTTP protocol layer, we extract the number of connections with HTTP response status codes 3xx, 4xx, and 5xx, the absolute counts of HTTP 1.0 connections and of the values of several HTTP header fields (Accept-Language, Content-Type, Connection, Accept-Charset, Accept-Encoding, Referer). We also extract User-Agent and define *mobile* and *crawler* which count all occurrences of a predefined set of known mobile user agents (Android and others) and crawlers (GoogleBot and others), respectively.

We count the number of different resource paths that a client accesses and also count how often each client requests the currently most common path on the domain. If a specific resource is directly accessed we extract and categorize the file ending into *plain, script, picture, download, media, other, none*, which can give a hint on the type of the requested resource. We measure the fractions of request types per connection (*GET, POST*, or *OTHER*). We extract the number of connections with a query string and the average length of each query in terms of number of fields per client. We count the number of connection in which the referrer is the domain itself. Geographic locations are encoded in terms of 21 parameters that represent a geographic region.

7.4.2 Input Features for SVDD, Logistic Regression and ICA

Independent classification uses features $\mathbf{\Phi}_{\mathbf{x}}(x_j)$ that refer to a particular client x_j and to the entirety of all clients $\mathbf{x}$ that interact with the domain. For each of the count-style base traffic parameters, $\mathbf{\Phi}_{\mathbf{x}}(x_j)$ contains the absolute value, globally normalized over all clients of all domains, a logarithmic absolute count, the globally normalized sums and log-sums over all clients that interact with the domain, and the absolute values, normalized by the values of all clients that interact with the domain. For HTTP response code, resource type header fields, we also determine the entropy and frequencies per client on for all clients on the domain. See also Fig 1.

Feature vector $\mathbf{\Phi}_{\mathbf{x},\mathbf{y}}(x_j)$ for ICA contains all features from $\mathbf{\Phi}_{\mathbf{x}}(x_j)$ plus the numbers of clients that are assigned class $+1$ and -1, respectively, in $\mathbf{x}, \mathbf{y}$.

7.4.3 Features for Structured Prediction

Feature vector $\mathbf{\Phi}(\mathbf{x}, \mathbf{y})$ contains as one feature the sum $\sum_{j=1}^{|\mathbf{x}|} y_j f_{\phi}^{LR}(\mathbf{\Phi}_{\mathbf{x}}(x_j))$ of scores of a previously trained *logistic regression* classifier over all clients $x_j \in \mathbf{x}$. In addition, we distinguish between the groups of clients that $\mathbf{y}$ labels as -1 and $+1$ and determine the inner-group means, inner-group standard deviations, inter-group differences of the base traffic parameters. This results in a total of 297 features.

7.4.4 Decoder Features

For *HC search* and *online policy gradient*, the parametric decoders depend on a joint feature representation $\mathbf{\Psi}(\mathbf{x}, Y_t(\mathbf{x}), a_{t+1})$ of input $\mathbf{x}$ and action $a_{t+1} = (r, \mathbf{y})$. It contains 92 joint features of the clients whose label a_{t+1} changes and the group (clients of positive or negative class) that a_{t+1} assigns the clients to. Features include the clients' distance to the group mean and the clients' distance to the group minimum for the base traffic parameters. For the fourth group of control actions, the feature representation includes the mean values of these same base attributes for all clients above and below the cutoff value. In order to save computation time, the mean and minimal group values before reassigning the clients are copied from $\mathbf{\Phi}(\mathbf{x}, \mathbf{y})$ which must have been calculated previously.

7.4.5 Execution-Time Constraint

We model distribution $p(T|\tau)$ that limits the number of time steps that are available for *HC search* and *online policy gradient* as a beta distribution with $\alpha = 5$ and $\beta = 3$ that is capped at a maximum value of $\bar{T} = 10$. We allow *ICA* to iterate over all instances for five times; the results do not improve after that. The execution time of *logistic regression* is negligible and therefore unconstrained.

8 Experimental Study

This section explores the practical benefit of all methods for attacker detection.

8.1 Data Collection

In order to both train and evaluate the attacker-detection models, we collect a data set of TCP/IP traffic from the application environment. We focus our data collection on high-traffic events in which a domain *might* be under attack. When the number of connections to a domain per unit of time, the number of clients that interact with the domain, and the CPU capacity used by a domain lie below safe lower bounds, we can rule out the possibility of a DDoS attack. Throughout an observation period of several days, we store all TCP/IP traffic to any domain for which a traffic threshold is exceeded starting 10 minutes before the threshold is exceeded and stopping 10 minutes after no threshold is exceeded any longer. During the 10 minutes before and after each event, around 80% of the 10-second intervals are empty.

This data collection procedure creates a sample of positive instances (attacking clients) that reflects the exact distribution which the attacker-detection system is exposed to during regular operations, because the attacker-detection model is applied when a domain exceeds the same traffic-volume and CPU thresholds. It creates a sample of negative instances (legitimate clients) that covers the operational distribution and also includes additional legitimate clients observed within 10 minutes of an unusual traffic event. Our intuition is that including additional legitimate clients that interact with the domain immediately before or after an attack in the training and evaluation data should make the model more robust against false-positive classifications.

We will refer to the entirety of traffic to a particular domain that occurs during one of these episodes as an *event*. Over our observation period, we collect 1,546 events. We record all traffic parameters described in Section 7.4. All data of one domain that are recorded within a time slot of 10 seconds are stored as a block. The same threshold-based pre-filtering is applied in the operational system, and therefore our data collection reflects the distribution which the attacker-detection system is exposed to in practice.

We then label all traffic events as *attacks* or *legitimate traffic* and all clients as *attackers* or *legitimate clients* in a largely manual process. In a joint effort with experienced administrators, we decide for each of the 1,546 unusual event whether it is in fact a flooding attack. For this, we employ several tools and information sources. We search for known vulnerabilities in the domain's scripts, analyze the domain's recent regular connection patterns, check for unusual geo-location patterns and analyze the query strings and HTTP header fields. This labeling task is inherently difficult. On one hand, repeated queries by several clients that lead to the execution of a CPU-heavy script with either identical or random parameters might very likely indicate an attack. On the other hand, when a resource is linked to by a high-traffic web site and that resource is delivered via a computationally expensive script, the resulting traffic may look very similar to traffic observed during an attack and one has to search for and check the referrer for plausibility to identify the traffic as legitimate.

After having labeled all events, we label individual clients that connect to a domain during an attack event. We use several heuristics to group clients with a nearly identical and potentially malicious behavior and label them jointly by hand. We subsequently label the remaining clients after individual inspection.

In total, 50 of the 1,546 events are actually attacks with 10,799 unique attackers. A total of 448,825 client IP addresses are labeled as legitimate. In order to reduce memory and storage usage we use a sample from all 10-second intervals that were labeled. We draw 25% of intervals per attack and 10% of intervals (but at least 5 if the event is long enough) per non-attack event. Our final data set consists of 1,096,196 labeled data points; each data point is a client that interacts with a domain within one of the 22,645 non-empty intervals of 10 seconds.

8.2 Experimental Setting

Our data includes 50 attack events; we therefore run 50-fold stratified cross validation with one attack event per fold. Since the attack durations vary, the number of test instances varies between folds. We determine the costs of all methods as the average costs over the 50 folds.

Table 1 Costs, true-positive rates, and false-positive rates of all attacker-detection models. Costs marked with “$*$” are significantly lower than the costs of *logistic regression*

Classification Method	Mean costs per fold	TPR	FPR ($\times 10^{-4}$)
No filtering	3.363 ± 1.348	0	0
SVDD	2.826 ± 1.049	0.121 ± 0.036	149.8 ± 89.5
Log. Reg. w/o domain-dependent features	1.322 ± 0.948	0.394 ± 0.056	7.0 ± 2.1
Logistic Regression	1.045 ± 0.715	0.372 ± 0.056	2.1 ± 0.6
ICA	$0.946 \pm 0.662*$	0.369 ± 0.056	3.2 ± 1.0
HC search with average margin	1.042 ± 0.715	0.406 ± 0.056	9.1 ± 4.2
HC search with max-margin	$1.040 \pm 0.714*$	0.398 ± 0.056	7.0 ± 3.3
Policy gradient with baseline function	$0.945 \pm 0.664*$	0.394 ± 0.055	3.7 ± 1.2
Policy gradient without baseline function	$0.947 \pm 0.665*$	0.394 ± 0.055	3.7 ± 1.2

In each fold, we reserve 20% of the training portion to tune the hyperparameters of all models by a grid search.

8.3 Reference Methods

All previous studies on detecting and mitigating application-layer DDoS flooding attacks are based on anomaly-detection methods [37,28,36,8,22]. A great variety of heuristic and principled approaches is used. In our study, we represent this family of approaches by *SVDD* which has been used successfully for several related computer-security problems [12,15]. Prior work generally uses smaller feature sets. Since we have not been able to improve our anomaly-detection or classification results by feature subset selection, we refrain from conducting experiments with the specific feature subsets that are used in published prior work.

Some prior work uses features or inference methods that cannot be applied in our application environment. DDosShield [28] calculates an *attack suspicion score* by measuring a client’s deviation from inter-arrival times and session workload profiles of regular traffic. Monitoring workload profiles is not possible in our case because the attacker-detection system is running on a different machine; it cannot monitor the workload profiles of the large number of host computers whose traffic it monitors. DDosShield also uses a scheduler and prioritizes requests by suspicion score. This approach is also not feasible in our application environment because it still requires all incoming requests to be processed (possibly by returning an error code). Xie and Yu [36] also follow the anomaly-detection principle. They employ a hidden Markov model whose state space is the number of individual web pages. In our application environment, both the number of clients and of hosted individual pages are huge and prohibit state inference for of each individual client.

8.4 Results

Table 1 shows the costs, true-positive rates, and false-positive rates of all methods under investigation. All methods reduce the costs that are incurred by DDoS attacks substantially at low false-positive rates. *SVDD* reduces the costs of DDoS attacks compared to not employing any attacker-detection mechanism (*no filtering*) by about 16%. *Logistic regression* reduces the costs of DDoS attacks compared (*no filtering*) by about 69%; *online policy gradient* reduces the costs by 72%. Differences between *no filtering*, *SVDD*, and *logistic regression* are highly significant. Cost values marked with an asterisk star (“$*$”)

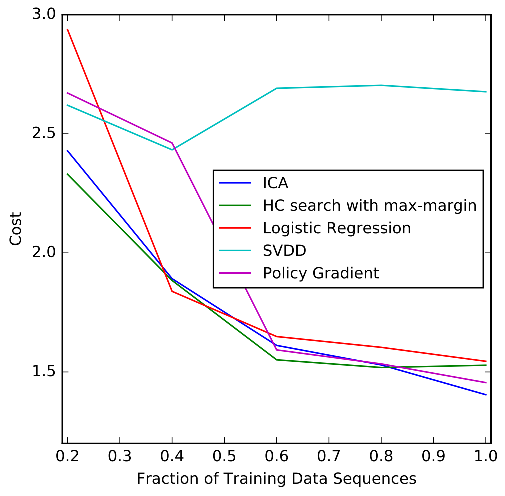

Fig. 2 Learning curves over varying fractions of training events.

are significantly lower than logistic regression in a paired t-test at $p < 0.1$. While *HC search* is only marginally (insignificantly) better than *logistic regression*, all other structured-prediction models improve upon *logistic regression*. *Policy gradient with baseline function* incurs marginally lower costs than *policy gradient without baseline function* and *ICA*, but the differences are not significant.

Logistic regression w/o domain-dependent features does not get access to features that take into account all other clients of that domain and to the entropy features. This shows that engineering context features into the feature representation of independent classification already leads to much of the benefit of structured prediction. From a practical point of view, all classification methods are useful, reduce the costs associated with DDoS attacks by around 70% while misclassifying only an acceptable proportion (below 10^{-3}) of legitimate clients. We conclude that *ICA* and *policy gradient* achieve a small additional cost reduction over independent classification of clients.

8.5 Analysis

In this section, we quantitatively explore which factors contribute to the residual costs of structured prediction models. The overall costs incurred by *policy gradient* decompose into costs that are incurred because f_ϕ fails to select the best labeling from the decoding set $Y_T(\mathbf{x})$, and costs that are incurred because decoder π_ψ approximates an exhaustive search by a very narrow and directed search that is biased by ψ.

We conduct an experiment in which decoder π_ψ is learned on training data, and a perfect decision function f^*_ϕ is passed down by way of divine inspiration. To this end, we learn π_ψ on training data, use it to construct decoding sets $Y_T(\mathbf{x}_i)$ for the test instances, and identify the elements $\hat{\mathbf{y}} = \text{argmin}_{\mathbf{y} \in Y_T(\mathbf{x}_i)} c(\mathbf{x}_i, \mathbf{y}_i, \mathbf{y})$ that have the smallest true costs; note that this is only possible because the true label $\mathbf{y}_i$ is known for the test instances. We observe costs of 0.012 ± 0.008 for the perfect decision function, compared to costs of 0.945 ± 0.664

when ϕ is learned on training data. The costs of a perfect decoder that exhaustively searches the space of all labelings, in combination with perfect decision function f^*_ϕ, would be zero. This implies that the decoder with learned parameters ψ performs almost as well as an (intractable) exhaustive search; it contributes only 1.3% of the total costs whereas 98.7% of the costs are due to the imperfection of f_ϕ. Increasing the decoding time T does not change these results.

This leaves parameter uncertainty of ϕ caused by limited labeled training data and the definition of the model space as possible sources the residual costs. We conduct a learning curve analysis to explore how decreasing parameter uncertainty decreases the costs. We determine costs for various fractions of training *events* using 10-fold cross validation in Figure 2. We use 10-fold cross validation in order to make sure that each test fold contains at least one attack event when reducing the number of events to 0.2. Since Table 1 uses 50-fold cross validation (which results in a higher number of training events), the end points of Figure 2 are not directly comparable to the values in Table 1. Figure 2 shows that the costs of all classification methods continue to decrease with an increasing number of training events. A massively larger number of training events would be required to estimate the convergence point. We conclude that parameter uncertainty of ϕ is the dominating source of costs of all classification models. Anomaly-detection method *SVDD* only requires unlabeled data that can be recorded in abundance. Interestingly, *SVDD* does not appear to benefit from a larger sample. This matches our subjective perception of the data: HTTP traffic rarely follows a "natural" distribution; anomalies are ubiquitous, but most of the time they are not caused by attacks.

8.6 Feature Relevance

For the independent classification model, leaving out features that take into account all clients that connect to the domain deteriorates the performance (see Line 3 of Table 1. We have not been able to eliminate any particular group of features by feature subset selection without deteriorating the system performance. Table 2 shows the most relevant features; that is, the features that have the highest average weights (over 50-fold cross validation) in the *logistic regression* model.

8.7 Execution Time

In our implementation, the step of extracting features $\mathbf{\Phi}$ takes on average 1 ms per domain for *logistic regression* and *ICA*. The additional calculations take about 0.03 ms for *logistic regression* and 0.04 ms for *ICA* with five iterations over the nodes which results in nearly identical total execution times of 1.03 and 1.04 ms, respectively.

HC search and *online policy gradient* start with an execution of *logistic regression*. For $T = 10$ decoding steps, repeated calculations of $\mathbf{\Phi}(\mathbf{x}, \mathbf{y})$ and $\mathbf{\Psi}(\mathbf{x}, Y_t(\mathbf{x}), a)$ lead to a total execution time of 3.1 ms per domain in a high-traffic event.

9 Discussion and Related Work

Mechanisms that merely *detect DDoS attacks* still leave it to an operator to take action. Methods for *detecting malicious HTTP requests* can potentially prevent SQL-injection and

Table 2 Most relevant features of f_ϕ

Weight	Description
3.01	Average length of query strings of client
-2.38	Number of different resource paths of client
2.34	Sum of incoming payload of all clients of domain
2.27	Fraction of connections of client that request the most frequent resource path
2.25	Sum of response times of all clients of domain
2.05	Sum of response times of client
1.64	Fraction of connections for domain that accepts any version of English (e.g., en-us) in Accept-Language
-1.46	Entropy of request type (GET/POST/OTHER)
-1.32	Sum of outgoing payload of all clients
1.27	Sum of number of open FINs of all clients at end of 10-seconds interval
1.23	Average length of query string per connection
-1.21	Fraction of connections for domain that accepts any language other than EN, DE, ES, PT, CN, RU in Accept-Language
1.19	Fraction of all connections of all clients that query most frequent path
1.17	Sum of durations of all connections of all clients of domain
1.13	Fraction of connections of client that accepts any version of English (e.g., en-us) in Accept-Language
-1.13	Fraction of combined connections of all clients that directly request a picture type
-1.11	Fraction of connections of client that specified HTTP header field Content-Type as any text variant
-1.09	Fraction of connections of client that accepts any language other than EN, DE, ES, PT, CN, RU in Accept-Language
1.08	Log-normalized combined outgoing payload of client
-1.07	Fraction of all connections of all clients that specified HTTP header field Content-Type as any text variant

cross-site scripting attacks, but their potential to mitigate DDoS flooding attacks is limited, because all incoming HTTP requests still have to be accepted and processed. Defending against network-level DDoS attacks [26,37] is a related problem; but since network-layer attacks are not protocol-compliant, better detection and mitigation mechanisms (*e.g.,* adaptive timeout thresholds, ingress/egress filtering) are available.

Since known detection mechanisms against network-level DDoS attacks are fairly effective in practice, our study focuses on application-level attacks—specifically, on HTTP-level flooding attacks. Prior work on *defending against application-level DDoS attacks* has focused on detecting anomalies in the behavior of clients over time [28,36,22,8]. Clients that deviate from a model of legitimate traffic are trusted less and less, and the rate at which their requests are processed is throttled. Trust-based and throttling approaches leave it necessary to accept incoming HTTP requests, maintain records of all connecting clients, and process the requests—possibly by returning an error code instead of the requested result. In our application environment, this would not sufficiently relieve the servers. Prior work on defending against application-level DDoS attacks have so far been evaluated using artificial or semi-artificial traffic data that have been generated under model assumptions of benign and offending traffic. This paper presents the first large-scale empirical study based on over 1,500 high-traffic events that we detected while monitoring several hundred thousand domains over several days.

Detection of *DDoS attacks* and *malicious HTTP requests* have been modeled as anomaly detection and classification problems. Anomaly detection mechanisms employ a model of legitimate network traffic [36]—and treat unlikely traffic patterns as attacks. For the detection of SQL-injection, cross-site-scripting (XSS), and PHP file-inclusion (L/RFI), traffic can be modeled based on HTTP header and query string information using HMMs [5], n-gram models [35], general kernels [12], or other models [29]. Anomaly-detection mechanisms were investigated, from centroid anomaly-detection models [18] to setting hard thresholds on the likelihood of new HTTP requests given the model, to unsupervised learning of support-vector data description (SVDD) models [12, 15].

Classification-based models require traffic data to be labeled; this gives classification methods an information advantage over anomaly-detection models. In practice, network traffic rarely follows predictable patterns. Spikes in popularity, misconfigured scripts, and crawlers create traffic patterns that resemble those of attacks; this challenges anomaly-detection approaches. Also, in shared hosting environments domains appear and disappear on a regular basis, making the definition of normal traffic even more challenging. A binary SVM trained on labeled data has been observed to consistently outperform a one-class SVM using n-gram features [35]. Similarly, augmenting SVDDs with labeled data has been observed to greatly improve detection accuracy [15]. Other work has studied SVMs [17, 21] and other classification methods [19, 25, 14].

Structured-prediction algorithms jointly predict the values of multiple dependent output variables—in this case, labels for all clients that interact with a domain—for a (structured) input [20, 32, 2]. At application time, structured-prediction models have to find the highest-scoring output during the decoding step. For sequential and tree-structured data, the highest-scoring output can be identified by dynamic programming. For fully connected graphs, exact inference of the highest-scoring output is generally intractable. Many approaches to approximate inference have been developed; for instance, for CRFs [1], structured SVMs [13], and general graphical models [3]. Several algorithmic schemes are based on iterating over the nodes and changing individual class labels locally. The iterative classification algorithm [24] for collective classification simplistically classifies individual nodes, given the conjectured labels of all neighboring nodes, and reiterates until this process reaches a fixed points.

Online policy-gradient is the first method that optimizes the parameters of the structured-prediction model and the decoder in a joint optimization problem. This allows us to prove its convergence for suitable loss functions. By contrast, *HC search* [9, 10] first learns a search heuristic that guides the search to the correct labeling for the training data, and subsequently learns the decision function of a structured-prediction model using this search heuristic as a decoder. Shi et al. [30] follow a complementary approach by first training a probabilistic structured model, and then using reinforcement learning to learn a decoder.

Wick et al. [34] sample structured outputs using a predefined, hand-crafted proposer function that samples outputs sequentially. In other work [33] a cascade of Markov models is learned that uses increasing higher-order features and prunes unlikely local outputs per cascade level. This work assumes a ordering of such cliques into levels, which is not applicable for fully connected graphs.

10 Conclusion

We have engineered mechanisms for detection of DDoS attackers based on anomaly detection, independent classification of clients, collective classification of clients, and structured-prediction with *HC search*. We have then developed the *online policy-gradient* method that learns a decision function and a stochastic policy which controls the decoding process in an integrated optimization problem. We have shown that this method is guaranteed to converge for appropriate loss functions. From our empirical study that is based on a large, manually-labeled collection of HTTP traffic with 1,546 high-traffic events we can draw three main conclusions. (a) All classification approaches outperform the anomaly-detection method *SVDD* substantially. (b) From a practical point of view, even the most basic *logistic regression* model is useful and reduces the costs by 69% at a false-positive rate of 2.1×10^{-4}. (c) *ICA* and *online policy gradient* reduce the costs just slightly further, by about 72%.

Acknowledgment

This work was supported by grant SCHE540/12-2 of the German Science Foundation DFG and by a grant from STRATO AG. We would like to thank the anonymous reviewers for their helpful comments.

References

1. Discriminative probabilistic models for relational data. In *Eighteenth Conference on Uncertainty in Artificial Intelligence*, 2002.
2. Max-margin Markov networks. In *Advances in Neural Information Processing Systems*, volume 16, 2004.
3. Approximated structured prediction for learning large scale graphical models. Arxiv 1006.2899, 2010.
4. C. Amza, E. Cecchet, A. Chanda, A. Cox, S. Elnikety, R. Gil, J. Marguerite, K. Rajamani, and W. Zwaenepoel. Bottleneck characterization of dynamic web site benchmarks. Technical report TR-02-391, Rice University, 2002.
5. Davide Ariu, Roberto Tronci, and Giorgio Giacinto. HMMPayl: An intrusion detection system based on hidden Markov models. *Computers & Security*, 30(4):221–241, 2011.
6. Vivek S. Borkar. *Stochastic approximation: A Dynamical Systems Viewpoint*. Cambridge University Press, 2008.
7. Vivek S. Borkar and Sean P. Meyn. The ODE method for convergence of stochastic approximation and reinforcement learning. *SIAM Journal on Control and Optimization*, 38(2):447–469, 2000.
8. S. Renuka Devi and P. Yogesh. Detection of application layer DDsS attacks using information theory based metrics. Department of Information Science and Technology, College of Engineering Guindy 10.5121/csit.2012.2223, 2012.
9. Janardhan Rao Doppa, Alan Fern, and Prasad Tadepalli. HC-search: Learning heuristics and cost functions for structured prediction. In *AAAI*, volume 2, page 4, 2013.
10. Janardhan Rao Doppa, Alan Fern, and Prasad Tadepalli. HC-search: A learning framework for search-based structured prediction. *Journal of Artificial Intelligence Research*, 50(1):369–407, 2014.
11. Janardhan Rao Doppa, Alan Fern, and Prasad Tadepalli. Structured prediction via output space search. *The Journal of Machine Learning Research*, 15(1):1317–1350, 2014.
12. Patrick Düssel, Christian Gehl, Pavel Laskov, and Konrad Rieck. Incorporation of application layer protocol syntax into anomaly detection. In *Information Systems Security*, pages 188–202. Springer, 2008.
13. T. Finley and T. Joachims. Training structural SVMs when exact inference is intractable. In *Proceedings of the International Conference on Machine Learning*, 2008.
14. Farnaz Gharibian and Ali A Ghorbani. Comparative study of supervised machine learning techniques for intrusion detection. In *Annual Conference on Communication Networks and Services Research*, pages 350–358. IEEE, 2007.
15. Nico Görnitz, Marius Kloft, Konrad Rieck, and Ulf Brefeld. Toward supervised anomaly detection. *Journal of Artificial Intelligence Research*, 46:235–262, 2013.
16. Evan Greensmith, Peter L. Bartlett, and Jonathan Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. *The Journal of Machine Learning Research*, 5:1471–1530, 2004.
17. Latifur Khan, Mamoun Awad, and Bhavani Thuraisingham. A new intrusion detection system using support vector machines and hierarchical clustering. *International Journal on Very Large Databases*, 16(4):507–521, 2007.
18. Marius Kloft and Pavel Laskov. Security analysis of online centroid anomaly detection. *Journal of Machine Learning Research*, 13(1):3681–3724, 2012.
19. Levent Koc, Thomas A Mazzuchi, and Shahram Sarkani. A network intrusion detection system based on a hidden naïve Bayes multiclass classifier. *Expert Systems with Applications*, 39(18):13492–13500, 2012.

20. John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: probabilistic models for segmenting and labeling sequence data. In *Proceedings of the International Conference on Machine Learning*, 2001.
21. Yinhui Li, Jingbo Xia, Silan Zhang, Jiakai Yan, Xiaochuan Ai, and Kuobin Dai. An efficient intrusion detection system based on support vector machines and gradually feature removal method. *Expert Systems with Applications*, 39(1):424–430, 2012.
22. H. Liu and K. Chang. Defending systems against tilt DDoS attacks. In *Proceedings of the International Conference on Telecommunication Systems, Services, and Applications*, 2011.
23. Luke K. McDowell, Kalyan Moy Gupta, and David W. Aha. Cautious collective classification. *The Journal of Machine Learning Research*, 10:2777–2836, 2009.
24. Jennifer Neville and David Jensen. Iterative classification in relational data. In *Proc. AAAI-2000 Workshop on Learning Statistical Models from Relational Data*, 2000.
25. Sandhya Peddabachigari, Ajith Abraham, Crina Grosan, and Johnson Thomas. Modeling intrusion detection system using hybrid intelligent systems. *Journal of Network and Computer Applications*, 30(1):114–132, 2007.
26. Tao Peng, Christopher Leckie, and Kotagiri Ramamohanarao. Survey of network-based defense mechanisms countering the DoS and DDoS problems. *ACM Computing Surveys*, 39(1):3, 2007.
27. Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697, 2008.
28. S. Ranjan, R. Swaminathan, M. Uysal, and E. Knightley. DDoS-resilient scheduling to counter application layer attacks under imperfect detection. In *Proceedings of IEEE INFOCOM*, 2006.
29. William K. Robertson and Federico Maggi. Effective anomaly detection with scarce training data. In *Network and Distributed System Security Symposium*, 2010.
30. Tianlin Shi, Jacob Steinhardt, and Percy Liang. Learning where to sample in structured prediction. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, pages 875–884, 2015.
31. Richard S. Sutton, David Mcallester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *In Advances in Neural Information Processing Systems 12*, pages 1057–1063. MIT Press, 2000.
32. I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484, 2005.
33. David Weiss and Ben Taskar. Structured prediction cascades. In *International Conference on Artificial Intelligence and Statistics*, pages 916–923, 2010.
34. Michael Wick, Khashayar Rohanimanesh, Kedar Bellare, Aron Culotta, and Andrew McCallum. Samplerank: Training factor graphs with atomic gradients. In *Proceedings of the 28th International Conference on Machine Learning*, pages 777–784, 2011.
35. Christian Wressnegger, Guido Schwenk, Daniel Arp, and Konrad Rieck. A close look on n-grams in intrusion detection: Anomaly detection vs. classification. In *Proceedings of the ACM Workshop on Artificial Intelligence and Security*, pages 67–76, 2013.
36. Y. Xie and S. Z. Yu. A large-scale hidden semi-markov model for anomaly detection on user browsing behaviors. *IEEE/ACM Transactions on Networking*, 17(1):54–65, 2009.
37. Saman Taghavi Zargar, James Joshi, and David Tipper. A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks. *IEEE Communications Surveys & Tutorials*, 15(4):2046–2069, 2013.