Eavesdropping Opponent Agent Communication Using Deep Learning

Thomas Gabel, Alaa Tharwat, and Eicke Godehardt

Faculty of Computer Science and Engineering
Frankfurt University of Applied Sciences
60318 Frankfurt am Main, Germany
{tgabel|aothman|godehardt}@fb2.fra-uas.de

Abstract. We present a method for learning to interpret and understand foreign agent communication. Our approach is based on casting the contents of intercepted opponent agent communication to a bit-level representation and on training and employing deep convolutional neural networks for decoding the meaning of received messages. We empirically evaluate our method on real-world data acquired from the multi-agent domain of robotic soccer simulation, demonstrating the effectiveness and robustness of the learned decoding models.

1 Introduction

Communication plays a central role in many multi-agent systems. The ability to communicate with one another can, among others, enable agents to coordinate their behavior or to overcome problems arising from partial state observability by, e.g., sharing local observations with one another. While these usages of communication adhere to cooperative multi-agents systems, it may be of importance in an adversarial setting as well. More particularly, in an adversarial multi-agent system it is very tempting to intercept the messages opponent agents exchange and understanding them can bring about significant benefits for the eavesdropping agent.

In this paper, we focus on settings in which the communication within a team of adversarial agents can be heard easily, but where the "language" used by those opponent agents is unknown. Thus, the challenge for the listening agent is to learn a model of the opponent agents' communication which effectively allows for interpreting the contents of messages received and to possibly act upon that information. The basic idea of our approach is to leverage state of the art deep learning techniques for recognizing the meaning of intercepted communication. To achieve this we formalize the problem as a supervised learning problem and develop a deep, bit-level convolutional neural network model that is trained on real, non-encrypted, and non-compressed communication data. The application area we are targeting is the domain of robotic soccer simulation where communication across agents, like in real soccer, plays a crucial role. Besides this multi-agent application, we are convinced that the basic ideas of our approach

might be utilized also for other message-based tasks such as reverse engineering or decoding data transfer in bus systems (like controller area networks).

We start this paper with some technical and algorithmic foundations and a summary of relevant related work. In Section 3 we present in detail our problem modeling and learning approach, followed by an empirical evaluation of our learned deep models (Section 4), applying them in the context of the competitive multi-agent world of robotic soccer simulation. Finally, Section 5 concludes. Since this paper makes use of quite some mathematical notation, the Appendix provides an overview of the most important symbols we use and a brief explanation of their meaning.

2 Foundations

We start off by providing some basics of our multi-agent application domain, simulated robotic soccer, before we continue to elaborate on foundations required for our deep communication learning approach. Also, this section is meant to provide references to related work.

2.1 Robotic Soccer Simulation

RoboCup [22] is an established international research initiative that aims at fostering research in AI, intelligent robotics, and multi-agent systems. Annually, there are championship tournaments in several leagues. Among these, the most tactically advanced and richest in terms of behavioral complexity is the 2D Soccer Simulation League, where two teams of simulated soccer-playing agents compete against one another using the Soccer Server [15]. This software implements a real-time soccer simulation system which puts into practice all aspects that are of relevance to multi-agent systems research.

The Soccer Server allows autonomous software agents to play soccer in a client/server-based style: The server simulates the playing field, communication, the environment and its dynamics, while the clients are permitted to send their actions once per simulation cycle to the server (each cycle lasts 100ms). Then, the server takes all agents' actions into account, computes the subsequent world state and provides all agents with (partial) information about their environment.

Robotic Soccer represents an excellent testbed for machine learning and, particularly, for multi-agent tasks. Several research groups have dealt with the task of learning parts of a soccer-playing agent's behavior autonomously (e.g. [12, 5, 16]) and even more have defined [20, 9] and investigated [11, 19, 2] various multi-agent related (sub-)tasks in robotic soccer. However, to the best of the authors' knowledge, the work we are presenting here is the first one that aims at learning and understanding the contents of opponent communication.

2.2 Communication in Simulated Soccer

As outlined, decision making must be performed in real-time or, more precisely, in discrete time steps: Every 100ms the agents can execute a low-level action

(like to *dash*, to *kick* the ball, *turn* their body or neck, or to *point* with their arm into some direction) and the world-state will change based on the individual actions of all players. Beyond these actions which influence the physical behavior of the agent, each agent is additionally allowed to communicate. In many existing multi-agent systems with communicating agents, the agents are allowed to use a reliable, high-bandwidth, and low-cost communication [23]. By contrast, in simulated soccer communication is unreliable, is restricted to low bandwidth, and uses a single shared communication channel for all 22 agents, thus mimicking the way spoken messages are transmitted and heard by humans in real soccer.

2.2.1 Technical Details

Direct agent-to-agent communication is strictly forbidden (more exactly, it would be considered a fraud attempt during competitions). Instead, each agent is allowed to broadcast a string of up to 10 characters in each time step. The broadcast is received by the Soccer Server and will be conveyed to selected players in the next simulation cycle, implying that any form of communication in soccer simulation is inherently delayed. Moreover, the communication channel is shared which imposes the restriction that each player can hear only one message from a teammate plus one from an opponent in each simulation step. The selection of the teammate and/or opponent to listen to can also be influenced by each agent by focusing its listening attention to exactly one specific player (otherwise random messages will be received). As a consequence, communication in soccer simulation is inherently unreliable since no agent can ever be sure whether some teammate has heard its say message or not. If all agents speak all the time, on average two of every eleven messages will be heard.

2.2.2 Communication Examples

Given these restrictions, at RoboCup competitions some teams make more intensive use of communication while others do less. Considering the amount of information sent to the Soccer Server during an average game, the share of communication data relative to the overall amount of information that needs to be transmitted in order to control all simulated parts of the agent's body is as high as

- 54% and 36% for former champion teams WrightEagle and Brainstormers / FRA-UNIted, implying that each agent sends a message in each time step,
- 17% and 13% for last year's finalists Gliders and Helios, meaning that each agent sends a say message in one of every six time steps on average.

Having a look at an excerpt of the text log file (created by the Soccer Server, storing all actions and commands issued by all players) of the 2016 final match, we can easily detect communicated say messages among other data transmitted by the agents (Figure 1). For example, we see that in time step 7 Helios' agent #4 dashed with 60% power, while #6 kicked the ball. We also observe, that four out of the six agents listed made use of communication by sending a 10-char *say* message, though we have a hard time understanding its meaning.

```
7,0 Recv Gliders2016_6: (dash 100)(turn_neck 41)(say "-EsQdKhGQI")
7,0 Recv HELIOS2016_4: (dash 60)(turn_neck -47)
7,0 Recv HELIOS2016_6: (kick 89.676 3.5062)(turn_neck 84)(attentionto off)(say "paw07w4-v ")
7,0 Recv Gliders2016_9: (dash 100)(turn_neck -5)(say "fbkHkNRaZP")
7,0 Recv Gliders2016_1: (dash 100)(turn_neck 0)(attentionto our 6)
7,0 Recv HELIOS2016_9: (dash 100)(turn_neck -55)(say "RZvS?uDYKw")
```

Fig. 1. Excerpt of the text log file of RoboCup 2016's final match.

2.3 Feed-Forward and Convolutional Neural Networks

Artificial neural networks are known for their excellent performance in different areas of machine learning. They are frequently applied for classification, regression, prognosis, as well as in reinforcement learning tasks. Specifically, multi-layer perceptron neural nets were shown to be universal function approximators [8].

In recent years, deep learning methods have witnessed significant improvements and brought about a number of breakthroughs in classic AI tasks that at times outperform human-level performance [14], including traditional multi-agent scenarios with imperfect information [4]. Additionally, deep (convolutional) neural networks as critic or actor in a reinforcement learning setting were, in combination with more classic tree search algorithms, able to beat human champions in challenging games like Go [18].

2.3.1 Multi-Layer Perceptron Neural Networks

A multi-layer perceptron is a neural network whose units (neurons) are connected in an acyclic graph. When calculating the activation of a neuron, we first calculate the net input to this neuron which is a weighted sum derived from all predecessor neurons' activations and the corresponding connection weights. In a second step, the net input to this neuron is passed through a differentiable monotonous activation function (e.g. the $tanh$, the logistic sigmoid function, or the rectified linear activation unit $ReLu$ [7]), thus yielding the neuron's output.

In a multi-layer perceptron neural network all of its neurons are arranged in layers that are disjoint from one another in that there are no connections among units within the same layer. Data is propagated through the network (forward propagation) by providing inputs to the network's first layer (input layer) and, subsequently, calculating the activations of all neurons in all successive layers (hidden layers) till the final, so-called output layer. For a given training set

$$\mathbb{P} = \{(x^p, t^p) | p \in \{1, \ldots, |\mathbb{P}|\}\} \tag{1}$$

of training patterns (x^p, t^p) with input vectors $x^p = (x_1^p, \ldots, x_m^p) \in \mathbb{R}^m$ and target values $t^p = (t_1^p, \ldots, t_n^p) \in \mathbb{R}^n$, a multi-layer neural net can be trained using one of the many variants of the back-propagation algorithm which essentially performs a gradient descent-based adaptation of the net's connection weights such that the summed squared error between the net's outputs under input of pattern x^p and the corresponding target values t^p is minimized. For more basics on neural networks the reader is referred to [17, 6].

2.3.2 Convolutional Neural Networks

Convolutional neural networks (CNNs, [13]) are a specialized type of neural networks for processing data with a grid-like topological layout, like image data where pixels form a 2D grid. CNNs differ from standard feed-forward neural nets in that they use convolution as a special linear operation in some of their layers. That operation is not limited to two-dimensional data, but can be applied to one-dimensional problems as well, e.g. for time series data.

Discrete convolution over an input $I \in \mathbb{R}^{m \times n}$ using a convolution kernel $K \in \mathbb{R}^{k \times k}$ with kernel size k produces a matrix $S \in \mathbb{R}^{m \times n}$ whose entries $S[i, j]$ are defined as

$$S[i, j] = (K * I)[i, j] = \sum_{p=1}^{m} \sum_{q=1}^{n} I[i - p, j - q] \cdot K[p, q]$$

This definition is a well-known operator in computer vision and it trivially extends to one-dimensional data, if $m = 1$. In machine learning, the input is usually an array of data (training examples) and the kernel's entries are the parameters that are adapted by the learning algorithm similar to what has been described in Section 2.3.1. The linear output of convolution is then run through a non-linear activation function such as *tanh* or *ReLu*. The output of the non-linear activation is often followed by maximum or average pooling layers that are meant to downsample the overall input, representing the output calculated so far by some summary of it. Besides, it should be stressed that each convolutional layer does not just employ a single, but a larger number of convolutional kernels, also called filters, each of which represents its own representation for the next layer of the network. Lastly, deep feed-forward neural networks do usually employ a number of stacked convolutional layers as described before, followed by one or more fully connected layer(s) to form the net's final output(s). A comprehensive and more detailed introduction and overview of CNNs can be found in [6].

3 Problem Modeling

As we saw in the previous section, intercepting opponent communication can be done easily in robotic soccer simulation. The agent can simply put its listening attention to a desired opponent and will receive its next say message in the subsequent simulation cycle given that the wire-tapped opponent has said something. In this section, we describe our approach to interpreting and understanding the contents of the received opponent message.

3.1 General Approach

Our goal is to pose the communication understanding task as a supervised learning problem. Generally, the to-be-predicted contents of a 10 char opponent say message could contain anything, from an agent's internal stamina state to high-level results of reasoning about our team's playing strategy. However, it is natural

to assume that in a competitive application domain like robotic soccer any agent is likely to frequently communicate such information that is most beneficial to its teammates with respect to the real-time properties and the partial observability of the simulation. Among such highly beneficial pieces of information are

1. the ball location and its velocity,
2. locations of teammates and opponent players,
3. pass announcements or requests,
4. locker room agreements and corresponding team strategy information (e.g. formation, role assignments, player markings etc. [21]).

In the remainder of this paper, we are going to specifically address points 1. and 3. In order to tackle these tasks with supervised learning we require a training data set $\mathbb{P}$ (cf. Equation 1) containing a substantial amount of training patterns (x^p, t^p).

Inputs x^p are communicated messages $\mathcal{C} = (c_s, \ldots, c_0)$ with maximal message size S and $s < S$ as well as with literals c_i from an alphabet A which in our case is the printable subset of ASCII-128 characters (cf. Figure 1). Since the length of each message is restricted to a length of S, the discrete set of possible communication messages is

$$\mathcal{A} = \cup_{i=0}^{S} A^i.$$

In our targeted application domain it holds $|\mathcal{A}| \approx 5.4 \cdot 10^{19}$ since $S = 10$ and A represents the set of 94 printable 7-bit ASCII characters (without quote sign).

Input Vector: A naive approach to learning a model for recognizing the contents of opponent say messages would be to feed a numeric representation of each letter c_i into the first layer of the network. This approach could be expected to bring about acceptable results already, if we could guarantee that the payload is never spread across multiples letters. Given the limited communication bandwidth in the type of multi-agent system we are considering, however, such an assumption is unrealistic as it would, for example, be wasteful to use one full letter (7 bit) for transmitting the unique number of some other agent for whose encoding 4 bit are sufficient.

By contrast, we suggest to define and utilize a bit-level representation for any received say message $\mathcal{C} = (c_s, \ldots, c_0)$ for the following reasons:

- Such a more fine-grained representation allows for capturing payload data that is spread across multiple characters c_i. Moreover, it allows for covering chunks of information that start at arbitrary bit positions within $\mathcal{C}$.
- Assuming that the communicating agents do not apply some kind sophisticated encryption/decryption techniques, the bit representation is likely to contain patterns that hint to the type of information contained (e.g. an identifier indicating it is a pass announcement) as well as bit patterns that contain the details or parameters (e.g. the starting point or velocity of a pass).

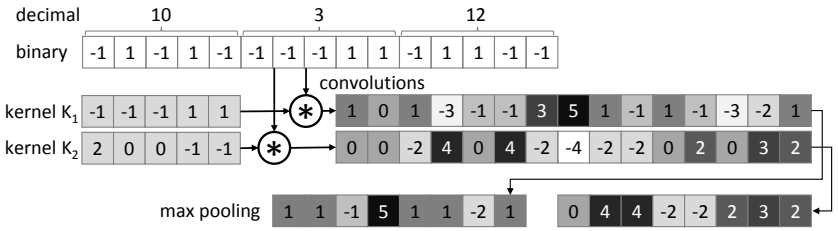

Fig. 2. One-dimensional Bit-Level Convolution

– The bit-level representation enables the usage of convolutional neural networks to detect features (i.e. patterns) within the bit sequence that makes it possible to classify a message's type, its contents or for doing regression on the encoded parameters it contains.

Given a new say message $\mathcal{C} \in \mathcal{A}$ we use a function

$$b : \mathcal{A} \rightarrow \{0,1\}^B \tag{2}$$

which maps $\mathcal{C}$ to a bit sequence $b(\mathcal{C})$ of length $B = S\lfloor \log_2 |A| \rfloor$ where A is the underlying alphabet (in our application it holds $B = 10\lfloor \log_2 94 \rfloor = 70$. In Section 3.3.2, we will discuss possible concrete implementations of b.

Figure 2 visualizes our approach for an exemplary bit vector of length 15 containing some "identifiers", including one represented by the decimal number 3 encoded as a 5-bit binary number. The figure also shows two 1D convolutional kernels that might be understood as the result of learning. Apparently, K_1 is able to "detect" a possible location of the searched for identifier "3" in the bit sequence, whereas K_2 is obviously meant to recognize other bit features.

Target Values: In what follows, we distinguish between two different types of learning problems that we want to tackle.

(a) Classification Problems: Here, we aim at the recognition of the type of information that is contained in a say message $\mathcal{C}$. Since each message can contain multiple chunks of information (and these in an arbitrary order, too), this amounts to a multi-label binary classification problem where $t^p \in \{true, false\}^l$ with l denoting the number of information chunks considered. If we, for example, aim at recognizing communicated ball information (one class), pass announcements (one class) as well as player information (one class for each player on the field) simultaneously, we arrive at $l = 24$. In the experiments reported below, we focus on ball and pass classification with $l = 1$ in each setting.

(b) Regression Problems: Assuming that our trained model in (a) states that a certain piece $\mathcal{P}$ of information is contained in $\mathcal{C}$ (e.g. data about the ball's location), the next logical step is to also extract the details or parameters of $\mathcal{P}$. In the example of a ball location such details are its x and y position on

the field, in the case of a pass announcement it might be the unique number of the receiving agent or the pass velocity. As a consequence, in this setting the challenge for the trained model is to extract numerical data from $\mathcal{C}$ which corresponds to a classical regression problem. Thus, $t^p \in \mathbb{R}^l$ with l taking a value that depends on the kind of information $\mathcal{P}$ contains.

As far as the determination of exact target values t^p is concerned, we have to extract the relevant information from real simulated soccer matches. For example, if we observe a pass being played by an opponent agent at time t, and if this is accompanied by a say message sent by that player at t or shortly before t, we have a high chance that this message contains a pass announcement and that it, hopefully, carries more information about the intended pass.

3.2 Model Architecture

We use a deep neural network architecture that is identical for all settings and learning tasks we are considering. As pointed out, input to the network is a bit-level representation $x^p = b(\mathcal{C})$ of an opponent agent's received say message $\mathcal{C}$ containing B bits. The positive and negative level of each bit a mapped to -1 and 1, respectively, which prevents us from simply ignoring all zero bits in the context of the convolutional operation (cf. Section 2.3.2).

The input is fed into a first convolutional layer of 128 one-dimensional convolutional filters of size 13, followed each by rectified linear activation units, whose output is then handed on to a maximum pooling layer with a stride of two. The second convolutional layer uses exactly the same layout as the first one with the exception of reduced kernel sizes of seven, and is also followed by *ReLu* activations and max pooling. The output of this layer is fed into a standard fully connected layer with 512 hidden units whose outputs are linearly combined to form the single (or multiple) output(s) of the net. The number of neurons in the output layer depends on the specific learning task as described in Section 3.1. To enforce regularization during learning we apply drop-out in the fully connected layer [3] with a drop-out rate of 0.5.

3.3 Communication En-/Decoding

In order to define a suitable bit-level representation function $b : \mathcal{A} \rightarrow \{0,1\}^B$ (cf. Equation 2) that maps an opponent say message $\mathcal{C} \in \mathcal{A}$ to a bit vector, we need to reflect about common ways for encoding some payload data efficiently in a message of limited size S over a given alphabet A. This means, for a moment (Section 3.3.1), we adopt the perspective of an opponent agent opp_1 and compare different approaches for defining a function $enc^{opp} : \mathcal{D} \rightarrow \mathcal{A}$ that encodes a given tuple $V = (v_1, \ldots v_d) \in \mathcal{D}$ of payload data, i.e. of numeric values v_i, to a maximally S-letter message $\mathcal{C}$ over alphabet A that can be decoded easily by one of its teammates opp_2 that receives $\mathcal{C}$.

3.3.1 Encoding Payload Data as Say Message

As a general note, we make the fundamental assumption that *all* agents do *not* apply any form of error detection or recovery since, on the one hand, the communication channel (simulated by the Soccer Server) does not introduce transmission errors and, on the other hand, agents are expected to maximize the amount of information transmitted (making check bits superfluous). Furthermore, we proceed on the assumption that no agent performs any kind of sophisticated data encryption which would render our entire approach useless.

(a) One Letter per Value: The most intuitive approach is to discretize the domain $D_i = [d_i^{min}, d_i^{max}]$ of v_i to $|A|$ values and map v_i to its nearest representative according to

$$enc_i^{opp} : D_i \to A \text{ with } v_i \mapsto A\left[\left\lfloor |A| \frac{v_i - d_i^{min}}{d_i^{max} - d_i^{min}} \right\rfloor\right]$$

where $A[j]$ provides access to the jth element of the alphabet. While easy to implement, this approach is suboptimal as it allocates one letter to each v_i and disregards the requirements on the resolution. For example, using one character for encoding a unique player number $v_i \in \{1, \ldots, 11\}$ is wasteful, whereas for conveying an x position on the field[1] this approach would yield a rather coarse discretization of steps with size $\frac{105}{|A|}$ which is $1.11m$ for printable 7-bit ASCII characters. It is unlikely that any of the existing robotic soccer teams is adopting such a basic encoding approach.

(b) Information-Oriented Bit Allocation: A clearly more efficient approach is to specify the required number of bits β_i for each piece of information v_i and to then create a block code (without error correction) from it. Hence,

$$enc_i^{opp} : D_i \to \{0,1\}^{\beta_i} \text{ with } v_i \mapsto bin\left(\left\lfloor 2^{\beta_i} \frac{v_i - d_i^{min}}{d_i^{max} - d_i^{min}} \right\rfloor\right)$$

where *bin* takes a natural number, converts it to a binary number, and represents it as a bit string of fixed length β_i. From this, a trivial block code can be created by simple string concatenation *bs* of the bit strings returned from each enc_i^{opp}, i.e. $bs^{opp}(V) = \oplus_i enc_i^{opp}$.

Given, for instance, the data $V = (v_1, v_2, v_3)$ where v_1 stands for one out of a set of 32 identifiers and $\binom{v_2}{v_3}$ represents the ball's location. Then, five bit are sufficient for encoding v_1 whereas one might spend 1024 possible discrete values (i.e. ten bit) for encoding the ball's x location on the pitch and 512 discrete values (nine bit) for its y location. This yields a sufficiently accurate resolution of $\frac{105}{1024} = 10.3cm$ and $\frac{68}{512} = 13.3cm$ in x and y direction, respectively. Concatenating the three binary numbers yields a 24-bit string as shown in Figure 3 for a payload extended by two more data fields.

[1] Pitch size in 2D simulated soccer is 105×68m.

(c) Fixed-Length Encoding: A straightforward approach for turning the block-code bit string $bs^{opp}(V)$ into a message $\mathcal{C} = (c_{S-1}, \ldots, d_0)$ is to split it into pieces of identical size σ, interpret each such piece as a natural number n, and to map it to a character c_i using some bijective index function $idx_A : A \to \{0, \ldots, |A|-1\}$ (the "code") such that $n = idx_A(c_i)$.

Given the size $|A|$ of the alphabet, it is natural to set $\sigma = \lfloor \log_2 |A| \rfloor$ and to use only a 2^σ-element subset of A. Knowing that $|A| = 94$ in our application, an agent using this approach encodes each $\sigma = 6$ bit from $bs^{opp}(V)$ by one character from A (see Figure 3, left-hand side).

(d) Base-$|A|$ Encoding: The fixed-length encoding in (c) is still wasteful in the sense that it does not fully exploit the available alphabet. The bit string $bs^{opp}(V)$ formed in (b), however, corresponds to a (possibly large) base-2 number N_2. Therefore, it is standing to reason to convert that number into a base-$|A|$ number and to simply employ that as the encoded message to be sent. Thus,

$$enc^{opp} : D \to \mathcal{C} \text{ with } V \mapsto base_{|A|}(bs^{opp}(V))$$

where $base_{|A|} : \{0,1\}^{\sum_i \beta_i} \to \mathcal{C}$. A binary number N_2 can be written in base $|A|$ as $N_2 = N_{|A|} = \sum_{i=0}^{S-1} idx_A(c_i)|A|^i$. So, the function $base_{|A|}$ yields a string $base_{|A|}(N_2) = (c_{S-1}, \ldots, c_0)$ over alphabet A with S denoting the maximal size of a message. Here, again, $idx_A : A \to \{0, \ldots, |A|-1\}$ refers to the index assigned to each letter from A (thus, depicting the agent's code).

3.3.2 Bit-Level Representation of Say Messages

We now return to our agent's point of view and, hence, to the question on how to interpret intercepted communication.

Naive Bit-Level Representation: Given an opponent's say message $\mathcal{C} = (c_s, \ldots, c_0)$, we can naively represent it as a bit vector using the individual characters' 7-bit ASCII code representation. Thus,

$$b_{asc} : \mathcal{A} \to \{0,1\}^B \text{ with } (c_s, \ldots, c_0) \mapsto \oplus_{i=0}^{s} asc(c_i)$$

where $asc(c_i)$ returns the index of c_i in the ASCII-128 table as a bit string of length 7. This way, we might easily retain the structure contained in a say message encoded using the fixed-length regime from above.

Base-$|A|$ Bit-Level Representation: Instead of the naive approach, throughout the rest of this paper, we are going to use the following more sophisticated one in which we take account of the base-$|A|$ encoding possibly applied by the communicating agent. Here, we define

$$b_{|A|} : \mathcal{A} \to \{0,1\}^B \text{ with } (c_s, \ldots, c_0) \mapsto bin\left(\sum_{i=0}^{S-1} idx_A(c_i)|A|^i\right)$$

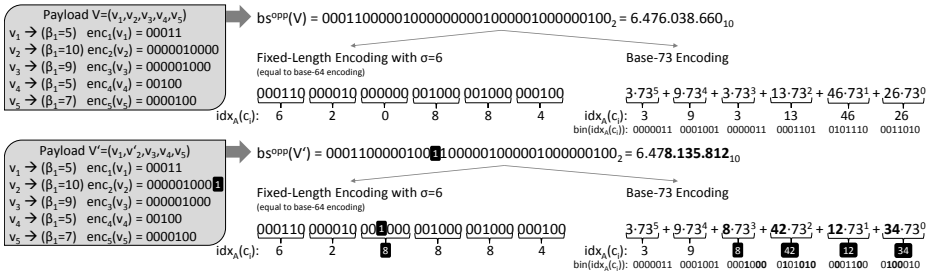

Fig. 3. Influence of a single bit swapped under fixed-length and base-73 encoding: Although only the least significant bit in only one of the five values v_i has changed from V to V', the resulting encoded message looks totally different under base-73 while the Hamming distance is only one under fixed-length encoding.

where *bin* returns a bit string as defined in Section 3.3.1 (b) and idx_A provides the index of a letter in the code as introduced above.

For this approach to be applicable, we need to know the full contents of the alphabet A used by the opponent agent. While we can infer the exact value of $|A|$ after having intercepted a sufficient number of messages, we have no chance of getting to know how the opponent agent's code idx_A is defined. In order to get along with this we assume some arbitrarily chosen function idx_A (e.g. with character indices sorted identically to the *asc* function from above) and rely on the learned deep representation of our approach to do the actual decoding work.

3.3.3 Encoding Remarks and Example

It is important to emphasize that the fixed-length encoding in Section 3.3.1 (c) is a special case of the base-$|A|$ encoding in (d) for $|A| = 2^\sigma$. Accordingly, our bit-level representation $b_{|A|}$ in Section 3.3.2 is capable of handling both of them.

We have, however, presented them separately in order to highlight that:

- For a message size of $S = 10$ over an alphabet A with $|A| = 94$ we find that in (c) a payload of 60 bits can be transmitted per message. By contrast, in (d) the payload is effectively as high as $\log_2(94^{10} - 1) = 65.5$ bit.
- In general, a base-κ encoding can be employed by the opponent agents with any value $\kappa \leq |A|$.
- If, however, κ is *not* a power of two, then this has severe implications for our bit pattern recognition approach based on convolutional neural networks.

To understand the last of these three points, consider the example shown in Figure 3. Here, we have a tuple of payload data V that is altered by just the least significant bit of v_2 forming V'. Under an opponent-side fixed-length encoding (i.e. with κ a power of two) the encoded message changes by only one bit, too, whereas for values of κ that are not a power of two (in the figure: $\kappa = 73$) the encoded message $\mathcal{C}$ is mutated heavily. As a consequence, payload data with very similar contents is mapped to strongly differing say messages which

makes it difficult, if not impossible, for a convolutional neural network to learn patterns within the data that do generalize across communicated messages. For these reasons, it is essential to our approach to apply the base-$|A|$ bit-level representation $b_{|A|}$, if the opponent agent has encoded its payload data using a base-$|A|$ encoding function enc^{opp} as outlined above.

4 Empirical Evaluation

In the previous section, we have elaborated on different ways how opponent agents might encode their payload data $V \in \mathcal{D}$ to a fixed-size message. Each of the encoding functions is bijective and could be inverted easily by our eavesdropping agent, if it had access to the relevant parameters, i.e. to the alphabet subset size $|A|$ employed by the opponent, the values β_i and σ, the composition of the elements within V as well as the mapping between elements of A and their numeric representation (i.e. the assumed ordering of characters within A belonging to idx_A). Since, however, we do not have access to the internals of the opponent agent, it is a challenging task to learn a deep representation of them which allows us to decode opponent agent messages.

4.1 Experiment Overview

As a proof of concept, we start with the task of eavesdropping and correctly understanding the contents of out own team's (FRA-UNIted) communication. This is indeed a particularly straightforward task since we know the basic principles of our communication and how they are implemented. After that, we will put our focus on the communication of a selection of current top teams.

While we consider communicated ball information as part of our proof of concept, the main focus of this evaluation is on eavesdropping information about passes. In any case this includes both, classifying whether some message contains a certain piece of information as well as, if it does, extracting the numeric details of that information.

Our model has been implemented in Python using TensorFlow 1.0 [1] and was trained on a single Intel i7 machine with two GeForce GTX1080 GPUs using the Adam optimizer [10] with learning rate 10^{-4} minimizing the l_2 loss using stochastic gradient descent with a batch size of 64. A single epoch of stochastic gradient descent requires approximately 1.5ms on this hardware which is a speed-up by a factor of ten compared to training on a CPU, only.

When applying a learned model on a standard PC with an i7 CPU (as it is likely to be the case during competitions, for example), processing of a single say message (forward pass through the deep convolutional neural network) takes circa 0.5ms on average using TensorFlow's Python API. Given the time frame of 100ms per time step imposed by the Soccer Server, the computational requirements of our approach in terms of evaluating a learned model do, therefore, not represent an obstacle for a practical application under real-time requirements.

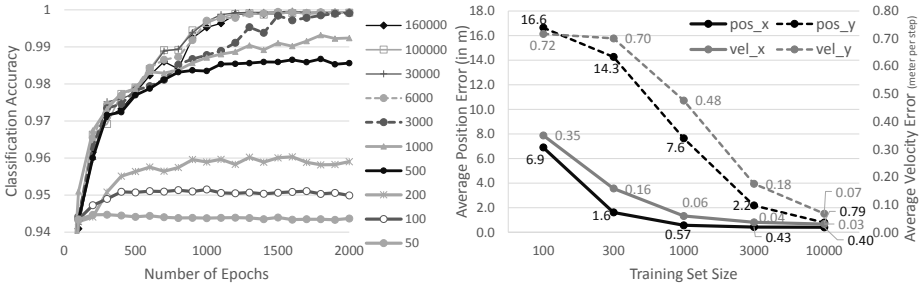

Fig. 4. Uncovering Ball Information: Classification accuracy achieved when trying to predict whether some message contains ball data for different training set sizes subject to neural network training epochs (left) and correctness of predicting ball position and velocity information from the messages' contents subject to increasing training set sizes (right).

4.2 Proof of Concept: Ball Information

The communication system of our soccer-playing agents currently supports 8 different payload types including ball and ball holder information, pass announcements and requests, information on individual players as well as free-form messages. Our implementation employs the information-oriented bit allocation approach (cf. part (b) of Section 3.3.1) and can, thus, embed multiple payload types in a single message. The agents employ a fixed-length encoding with $\sigma = 6$ (Section 3.3.1 (c)) using a 64-element subset of 7-bit ASCII characters as code.

In this section, we focus on communicated ball information which is a data chunk of 36 bits that can be placed at an arbitrary position within a say message (e.g. preceded by a pass request or followed by information about some other player). It contains a 5-bit payload type identifier, the encoded ball position and its velocity (each made up of an x and a y component) plus two 3-bit values indicating the age of the position and velocity values.

Our data set $\mathbb{P}$ consists of 200k say messages recorded from a single player in the course of more than thirty matches. 40k samples are left as an independent test set. We investigate the effectiveness of learning for different training set sizes sampled randomly from the remaining examples. Note that only a small subset (5.8%) of these messages contains a ball information, i.e. $\mathbb{P}$ contains about 16 times as many negative examples as positive ones.

Figure 5 (left) shows that 500 samples are sufficient for obtaining an average quality classifier, whereas about 3000 samples (i.e. the data from approximately one half-time of a match) yield a classification of the type of the say message with nearly zero error. As the right part of that figure shows, in order to also accurately decode the contents of the ball information, most prominently its position and velocity on the field, again the data collected within one half-time is sufficient. For assessing the usefulness of the accuracy of the decoded data it is worth noting that the domain of pos_x is $[-52.5, 52.5]$ (in meters), of pos_y is $[-34, 34]$ (in meters), and of $vel_{x/y}$ is $[-3, 3]$ (in meters per time step).

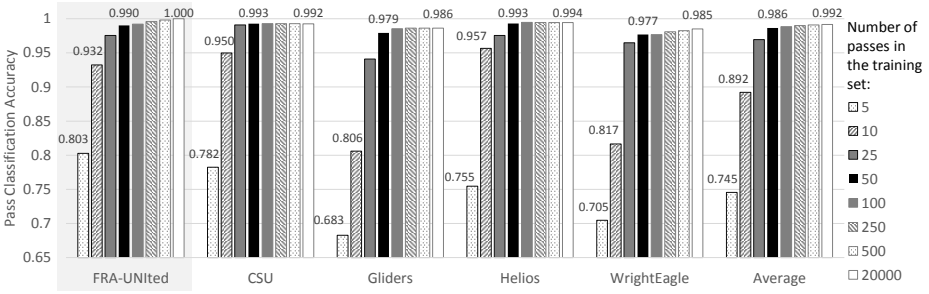

Fig. 5. Pass Classification: Accuracy of detecting the membership of pass announcing information in a say message subject to different opponent agents and to different amounts of observed passes with accompanying messages.

4.3 Pass Classification

Next, we aimed at classifying whether messages contain pass announcements. This time, however, we focused not just on our own team, for which we can create accurate class labels easily, but on a selection of current top-level teams[2]. We made these teams play a series of matches, recording both, their passes played as well as their entire communication. From this data we built a set $\mathbb{P}^+$ of 20k positive training examples by associating any actual pass played with the say message sent concurrently or shortly before the pass by the passing player.

Accordingly, the built training data set may contain wrong labels if, for example, some player played a pass, but did not announce it and, instead, broadcast some other information concurrently. Without knowing the internals of other agents, it is difficult to quantify that level of noise, though we expect it to be low since playing and announcing passes is a fundamental soccer skill that is likely to be mastered nearly perfectly by any of the opponent agents considered. Henceforth, all accuracy values reported refer to the classification performance on independent training examples that originate from the same data distribution including the same potential level of noisy labels.

In this series of experiments, we are also interested in assessing how many passes need to be observed until a reliable classifier can be trained. Therefore, in each run we selected a specific number p of samples from $\mathbb{P}^+$ and joined it with p negative examples (for which we used randomly selected say messages received during non-pass situations), thus forming $\mathbb{P}$. Again, 40k samples are left out as an independent test data set on which all results reported are evaluated.

Figure 5 summarizes the results obtained after applying stochastic gradient descent optimization after 5k epochs. Note that a random classifier, e.g. an untrained net, would yield a classification accuracy of 0.5 due to our data set compilation. Also, note that during an average match approximately 50 passes (sometimes more) are played by a team. Thus, for any of the teams considered we are able, for instance, to learn a pass announcement detector with an accuracy of

[2] Binaries of all contemporary teams are available at `chaosscripting.net`.

above $\approx 93\%$ using the intercepted communication data from a single half-time. The average accuracy using pass and communication data from a single game (black series) is as high as $\approx 98\%$ despite the fact that labels are possibly noisy.

4.4 Pass Regression: White and Black Box Experiments

The final stage of our experiments aims at decoding the payload data, i.e. the exact numeric details, of messages that were previously classified as containing pass information. Such details can contain the starting position $p = \binom{p_x}{p_y}$ of the pass, its velocity $v = \binom{v_x}{v_y}$ or, eventually, the unique number of the intended pass-receiving player.

To this end, we emphasize that the Soccer Server adds small random Gaussian noise to all actions performed by the agents (e.g. to kicks that are intended to play a pass) as well as, in each time step, to all movement vectors of objects on the field (e.g. the velocity vector of the ball). As a consequence, each pass actually played will most likely deviate slightly (in terms of its direction and its speed) from the intended pass whose properties were communicated by the pass-playing agent. Therefore, our training data set $\mathbb{P}$ contains inherently noisy target values, since we build the data set from actually observed passes.

White Box Agents: Since it is straightforward to generate pass announcements for agents of our team (as we do have access to their source code), we started by building a dataset of $200k$ such pass-announcing messages that we could label perfectly. We trained our decoder on 80% of the data, leaving $40k$ pass messages as an independent test set. After 10^6 epochs of neural network training, i.e. after about half an hour wall clock time, the error on the test set had dropped on average to $\binom{0.34}{0.50}$ for the pass start position and $\binom{0.03}{0.04}$ for the velocity (units: meters and meters per time step, respectively). Interestingly, even for much smaller data sets (e.g. $|\mathbb{P}| = 5k$, cf. Table 1) a decoding accuracy can be achieved that would be sufficient for taking appropriate counter measures during a game. Figure 6 visualizes the quality of the decoded information for FRA-UNIted as well as for the (black box) agents from team Helios.

Black Box Agents: To create the necessary data sets for other opponents agents (black box agents to whose source code we have no access), we made each agent team considered play a series of 1000 simulated matches of soccer (each one lasting 10 minutes wall clock time) during which we recorded their communication. Again, our deep decoding model was trained using the same parameterization, employing up to $20k$ examples in the training set and an independent test set covering $40k$ passes.

As can be read from Table 1, outstanding results could be obtained for all opponents considered with the exception of WrightEagle (no learning progress w.r.t. p and v at all). Here, we found, however, that a pass announcement by an agent from that team contains different pieces of data, namely the number of the targeted pass-receiving teammate as well as the absolute value of the pass

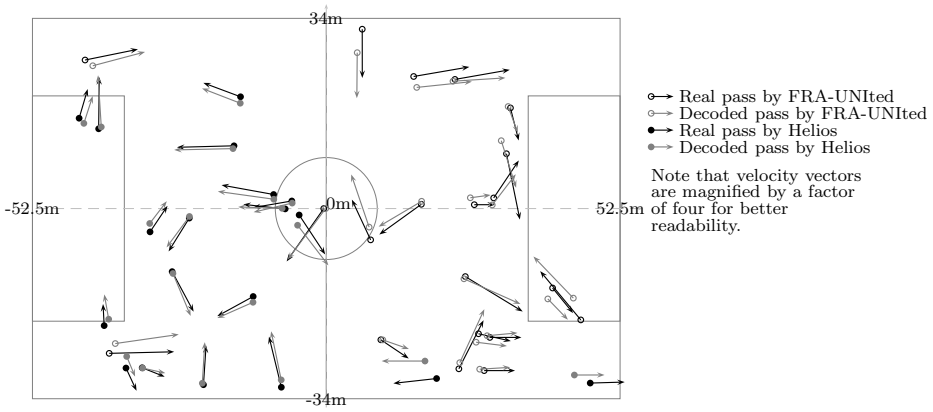

Fig. 6. For two of the teams considered, 20 randomly chosen (not cherry-picked) passes are visualized, opposing the real passes played and the information extracted from pass announcing say messages that were sent by pass-playing agents and decoded using our learned models.

	FRA-UNIted				CSU_Yunlu			
$\lvert\mathbb{P}\rvert$	p_x	p_y	v_x	v_y	p_x	p_y	v_x	v_y
500	1.18 ± 1.70	13.1 ± 16.7	$.13 \pm .22$	1.37 ± 1.65	2.78 ± 5.16	4.20 ± 5.77	$.28 \pm .44$	$.14 \pm .28$
5000	0.54 ± 0.85	4.15 ± 6.48	$.06 \pm .07$	0.37 ± 0.56	0.86 ± 2.28	1.47 ± 3.34	$.14 \pm .25$	$.10 \pm .20$
20000	0.48 ± 0.55	1.43 ± 2.21	$.04 \pm .04$	0.09 ± 0.15	0.44 ± 1.62	1.04 ± 2.86	$.13 \pm .21$	$.10 \pm .18$

Gliders				Helios				WrightEagle	
p_x	p_y	v_x	v_y	p_x	p_y	v_x	v_y	$\lvert v\rvert$	r
3.01 ± 5.97	4.50 ± 6.48	$.30 \pm .46$	$.16 \pm .37$	2.31 ± 4.16	3.61 ± 4.91	$.28 \pm .36$	$.13 \pm .26$	$.41 \pm .52$	.672
1.50 ± 4.18	1.86 ± 4.39	$.14 \pm .31$	$.12 \pm .32$	0.69 ± 2.21	1.22 ± 2.24	$.10 \pm .19$	$.09 \pm .20$	$.29 \pm .40$	.332
0.79 ± 2.85	1.23 ± 3.71	$.11 \pm .21$	$.10 \pm .23$	0.45 ± 1.34	0.74 ± 1.85	$.09 \pm .16$	$.09 \pm .18$	$.19 \pm .28$	.251

Table 1. Average errors (in meters and meters per time step, respectively) and their standard deviations for the decoded pass data (pass start position p, pass velocity v, unique number r of pass receiver) intercepted from different opponent agent teams.

velocity. Then, when modifying the labels of $\mathbb{P}$ accordingly, for a (extended) training set of $50k$ passes we are able to achieve an error of the pass velocity of 0.04 ± 0.05 meters per time step and an error in classifying the pass receiver (one out of 11 classification) of 21.6%.

5 Conclusion

In this paper, we have presented an approach that aims at understanding foreign agent communication using deep learning. Our learner intercepts fixed-size messages from opponent agents, casts their contents to a bit-level representation, and uses deep convolutional neural networks to interpret the contents of the message. Our work has been embedded into the multi-agent domain of robotic soccer simulation. In our experiments, we have shown that our approach is capable of correctly classifying the type of the payload data contained in a message

as well as inferring numeric values therein. While we have shown that our approach works well for recognizing ball and pass information, in future work we aim at recognizing further categories of data communicated by opponent agents. Our very next step, however, is to do the engineering work for integrating the approach presented into our competition team in such a manner that it adheres to all real-time constraints of the soccer simulation domain and such that it can be utilized during competitions.

Appendix

Notation and mathematical symbols used and their meaning in order of their appearance.

$\mathbb{P}$	training data	d_i^{max}	upper bound of D_i
x^p	input of pth training pattern	β_i	number of bits used to encode v_i
t^p	target value of pth training pattern	bin	mapping from natural number to binary
$\mathcal{C}$	communicated message	bs	mapping from payload tuple to bit string
c_i	ith character in message	idx_A	mapping from character to its index in the alphabet
S	maximal message size	σ	number of bits mapped to one character (fixed-length encoding)
A	alphabet	$base_{\|A\|}$	mapping from binary number to a base-$\|A\|$ number
$A[j]$	character at index j in alphabet	asc	mapping from character to its index in the ASCII table
$\mathcal{A}$	set of all formable messages	b_{asc}	mapping from message to bit string using ASCII indices
b	mapping from message to bit string	$b_{\|A\|}$	mapping from message to bit string using indices in alphabet
B	length of bit string		
V	tuple of payload data		
v_i	ith numeric value in payload data		
$\mathcal{D}$	domain of payload data		
enc^{opp}	encoding of payload data to message		
$\oplus$	concatenation of bit strings		
D_i	domain of ith value in payload data		
d_i^{min}	lower bound of D_i		

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems (2015), http://tensorflow.org/, software available from tensorflow.org
2. Almeida, F., Abreu, P., Lau, N., Reis, L.: An Automatic Approach to Extract Goal Plans from Soccer Simulated Matches. Soft Computing 17(5), 835–848 (2013)
3. Ba, J., Frey, B.: Adaptive Dropout for Training Deep Neural Networks. In: Advances in Neural Information Processing Systems (NIPS). pp. 3084–3092 (2013)
4. Brown, N., Sandholm, T.: Safe and Nested Endgame Solving for Imperfect-Information Games. In: Proceedings of the AAAI workshop on Computer Poker and Imperfect Information Games (2017)

5. Gabel, T., Riedmiller, M.: Learning a Partial Behavior for a Competitive Robotic Soccer Agent. KI Zeitschrift 20(2), 18–23 (2006)
6. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2017)
7. Hahnloser, R., Sarpeshkar, R., Mahowald, M., Douglas, R., Seung, H.: Digital Selection and Analogue Amplification Coesist in a Cortex-Inspired Silicon Circuit. Nature 405(6789), 947–951 (2000)
8. Hornick, K., Stinchcombe, M., White, H.: Multilayer Feedforward Networks Are Universal Approximators. Neural Networks 2, 359–366 (1989)
9. Kalyanakrishnan, S., Liu, Y., Stone, P.: Half Field Offense in RoboCup Soccer: A Multiagent Reinforcement Learning Case Study. In: RoboCup-2006: Robot Soccer World Cup X. pp. 72–85. Springer, Berlin, Germany (2007)
10. Kingma, D., Ba, J.: Adam: A Method for Stochastic Optimization. In: Proceedings of the 3rd International Conference on Learning Representations (2015)
11. Kok, J., Spaan, M., Vlassis, N.: Non-Communicative Multi-Robot Coordination in Dynamic Environments. Robotics & Autonomous Systems 50(2-3), 99–114 (2005)
12. Kuhlmann, G., Stone, P.: Progress in Learning 3 vs. 2 Keepaway. In: RoboCup-2003: Robot Soccer World Cup VII. Springer Verlag, Berlin (2004)
13. LeCun, Y.: Generalization and Network. Design Strategies. Technical Report CRG-TR-89-4, University of Toronto (1989)
14. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A., Veness, J., Bellemare, M., Graves, A., Riedmiller, M., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-Level Control Through Deep Reinforcement Learning. Nature 518(7540), 529–533 (2015)
15. Noda, I., Matsubara, H., Hiraki, K., Frank, I.: Soccer Server: A Tool for Research on Multi-Agent Systems. Applied Artificial Intelligence 12(2-3), 233–250 (1998)
16. Riedmiller, M., Gabel, T., Hafner, R., Lange, S.: Reinforcement Learning for Robot Soccer. Autonomous Robots 27(1), 55–74 (2009)
17. Rumelhart, D., Hinton, G.: Learning Representations by Back-Propagating Errors. Nature 323, 533–536 (1986)
18. Silver, D., Huang, A., Maddison, C., Guez, A., Sifre, L., Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D.: Mastering the Game of Go with Deep Neural Networks and Tree Search. Nature 529(7587), 484–489 (2016)
19. Stolzenburg, F., Murray, J., Sturm, K.: Multiagent Matching Algorithms with and without Coach. Journal of Decision Systems 15(2-3), 215–240 (2006)
20. Stone, P., Kuhlmann, G., Taylor, M., Liu, Y.: Keepaway Soccer: From Machine Learning Testbed to Benchmark. In: RoboCup-2005: Robot Soccer World Cup IX. pp. 93–105. Springer, Berlin, Germany (2006)
21. Stone, P., Veloso, M.: Task Decomposition, Dynamic Role Assignment, and Low-Bandwidth Communication for Real-Time Strategic Teamwork. Artificial Intelligence 110(2), 241–273 (June 1999)
22. Veloso, M., Balch, T., Stone, P.: RoboCup 2001: The Fifth Robotic Soccer World Championships. AI Magazine 1(23), 55–68 (2002)
23. Woolridge, M.: Reasoning about Rational Agents. MIT Press (2003)