MASTER RESEARCH INTERNSHIP

MASTER THESIS

Narrative Intelligence for Debriefing

Author:
François AUGER

Supervisors:
Anne-Gwenn BOSSER
Cédric BUCHE
Martin DIEGUEZ

13th June 2019

Contents

List of Figures

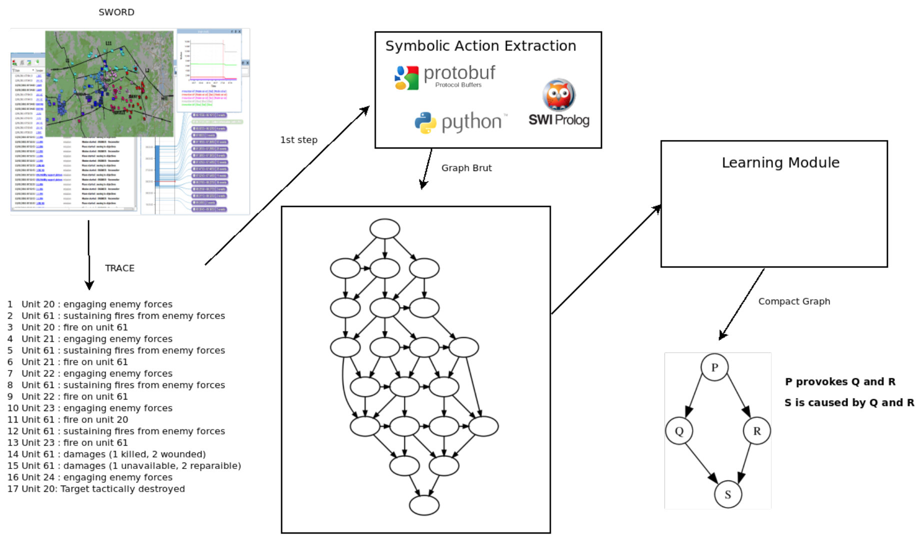
List of Tables

1 Introduction

The tool SWORD[1], is used by the French army to perform virtual training simulations which, thanks to the use of sophisticated mathematical models, are quite close to real training scenarios.

After every *virtual training*, the participants are involved in a debriefing process in which they discuss what happened, why and how the training can be improved in the subsequent simulations. For this process they take advantage of the *huge* amount of data generated by the simulator.

Information and the timing play a crucial role in the debriefing: as suggested in [1], very detailed information make the participants loose their concentration and an inadequate timing may make them forget crucial information they used for making their decisions. In order to meet such requirements, authors of [7, 8] proposed a tool for turning the data resulting from the simulation into a causal graph [23] in which all the actions triggered during the simulation and their *cause-effect* relations are represented. The left part of Figure 1 presents the architecture of that system.



```
1   Unit 20 : engaging enemy forces
2   Unit 61 : sustaining fires from enemy forces
3   Unit 20 : fire on unit 61
4   Unit 21 : engaging enemy forces
5   Unit 61 : sustaining fires from enemy forces
6   Unit 21 : fire on unit 61
7   Unit 22 : engaging enemy forces
8   Unit 61 : sustaining fires from enemy forces
9   Unit 22 : fire on unit 61
10 Unit 23 : engaging enemy forces
11 Unit 61 : fire on unit 20
12 Unit 61 : sustaining fires from enemy forces
13 Unit 23 : fire on unit 61
14 Unit 61 : damages (1 killed, 2 wounded)
15 Unit 61 : damages (1 unavailable, 2 reparaible)
16 Unit 24 : engaging enemy forces
17 Unit 20: Target tactically destroyed
```

Figure 1: Architecture of the system proposed in [8]

This system exploits the use of *Linear Logic* [13, 14] (LL) for representing both *dynamic problems* [12, 17, 10] and the notion of *causality* [9, 3, 6, 22, 23] in order to generate such causal graph associated to the SWORD's output. We will not go into the details of this algorithm since it is not the purpose of this internship, we refer the reader to [7, 8] for a more detailed presentation for this algorithm. However, the resulting causal graph may contain very detailed information that complicates its understanding. The combination of such symbolic approach with some machine learning

[1]See https://masasim.com/sword/

that simplifies the graph is a challenging problem that we will consider in this report.

In this report, I will explore how machine learning can contribute to improve the proposal of [7]. More precisely, I will describe how machine learning techniques can be used to simplify the output graph and, moreover, to detect (possible) anomalies that may have occurred during the simulation. The idea behind this contribution is to provide a method able to detect sequences of actions that are meaningful for the participants (usually military experts) of the simulation while, at the same time, find anomalies that, from a human point of view, may be unnoticed by the experts.

The reminder of this report is as follows. In Section 2 I survey several techniques of machine learning for extracting information from sequential data. Then, in Section 3 I focused on *Hidden Markov Models* [20] (HMM), the machine learning approach I will apply in this report. In Section 4 I present the main contributions of my internship and I finish this report with the conclusions where I summarise my contribution and I discuss about his limitations and future improvements.

2 State of the art on sequential supervised learning

Supervised Learning is a technique of *Machine Learning* in which the correct prediction is given during the training so the model has just to bind the input (an observable in a sequence) to the correct given output. When applied to sequential data, the resulting framework is often called *sequential supervised machine learning*.

Due to the increasing number of digital information, we often encounter many situations where it is necessary to extract information from sequential data such as sequences of words in a text or speech, frames in a video or nucleotides in the DNA. SWORD traces can be regarded as sequential data in which actions are sequentially triggered.

Along this section we will consider the following supervised machine learning algorithms for extracting information from sequential data [11]: Sliding Window, Hidden Markov Model, Maximum Entropy Markov model and Input-Output Hidden Markov Model, Conditional Random Fields and Recurrent Neural Networks. For each of them, we will focus on pros, cons and applications. Finally, based on the aforementioned aspects of these models, we will discuss the most suitable method to be applied in our context.

2.1 Sliding window

The Sliding Window is a machine learning method for sequential supervised learning. Given a *window* $w \in \mathbb{N}$, a classifier h_w is defined as a mapping from the subsequence determined by w to the predicted data. For instance, let $a_0, \cdots, a_n$ be a sequence and let w be a window, for a given i satifying that $0 \leq i \leq n$, h_w will predict a value from the elements in the interval $\left(i - \frac{w}{2}, i + \frac{w}{2}\right)$, as shown in Figure 2.

As its main advantage, authors of [11] suggest that this algorithm converts the sequential supervised learning problem into a simple supervised learning one whose input is the current observable together with its w-neighborhood. Therefore, it allows any normal classifier to be applied. The evident drawback relies on the fact that the

$$a_0, \cdots, \underbrace{a_{i-\frac{w}{2}}, \cdots,}_{\frac{w}{2}} \overset{\downarrow}{a_i}, \underbrace{\cdots a_{i+\frac{w}{2}}}_{\frac{w}{2}}, \cdots, a_n.$$

Figure 2: Selection elements in the sliding window algorithm.

sliding window does not take advantage of the correlations among the inputs and the neighborhood predictions. To solve this problem, a new technique called *recurrent sliding window* has been proposed in [4]. This new extension enhances sliding window with elements previously predicted by h_w. As for applications, the sliding window technique was successfully applied in [2] to detect change points in time series.

2.2 Hidden Markov Model

A Hidden Markov Model is a probabilistic model used to predict a sequence of hidden states after receiving an input sequence of observables. The training of the hidden Markov model is done by computing the backward-forward algorithm. The Hidden Markov Model is one of the oldest probabilistic model for sequential supervised learning and was used in many applications with a lot of variations. HMMs are often called *generative models* because of the conditional probability of having the observable X given a target Y (see Figure 3).

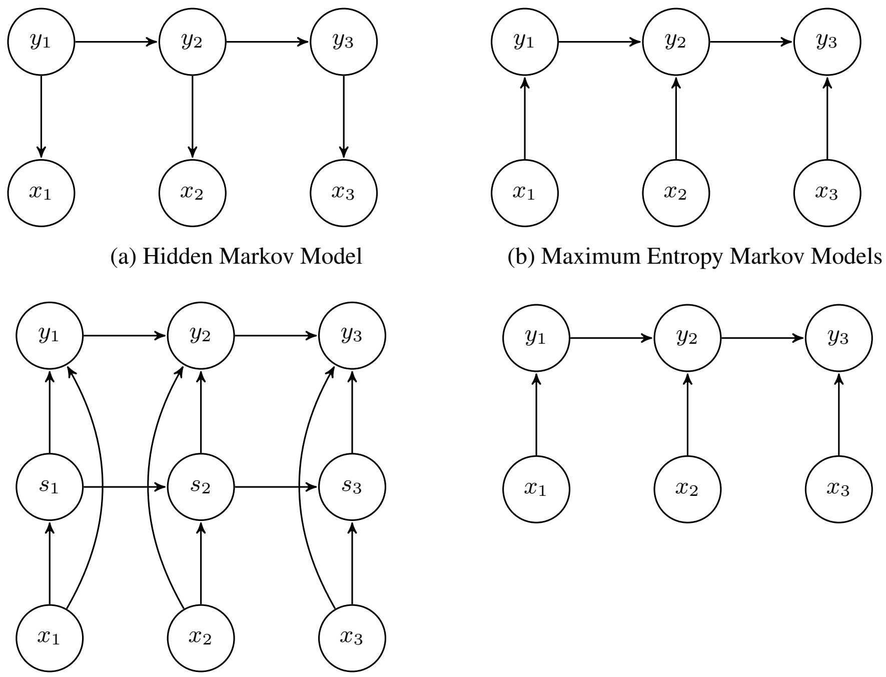
This technique is used to learn *Markovian processes*, those whose probability of being at a given state a at time t depends on being observed at a and the probability of reaching a from another state at $t-1$. This means that HMM cannot cannot predict values based on historical data[2]. This inconvenient, can be overcome by using a n-order Markov model, but the computational cost may be very high. The HMM was successfully used in [18] on protein sequences for predicting transmembrane helices.

2.3 Maximum Entropy Markov Models

A *Maximum Entropy Markov Model* (MEMM) is also a probabilistic model in which the probability $p(y_t|y_{t-1}, x_t)$ is given by an exponential equation. These kind of models are trained using maximum entropy algorithms such as *Generalized Iterative Scaling* [21] (GIS), which is applied for each pairs $< observable, state >$ at each step.
Contrary to HMMs, Maximum Entropy Markov Models are known as conditional / discriminative models because of the conditional probability of having the target Y given the observable X. The difference between HMMs and MEMMs is shown in Figure 3b. While in the HMM of Figure 3a edges go from hidden states to observables, in the HEMM of Figure 3b edges go from observables to hidden states.

Maximum Entropy Markov Models were created to overcome the problem of the hidden Markov model: HMMs do not permit to seize long distance interactions between

[2]This property is called *d-separation* and it says that, having 3 variables X,Y and Z, if knowledge about X gives you no extra information about Y once you have knowledge of Z, you have X and Y d-separated.



(a) Hidden Markov Model

(b) Maximum Entropy Markov Models

(c) Input Output Markov Model

(d) Conditional Random Field

Figure 3: Bayesian network of Hidden Markov Models and closely related probabilistic models taken from [11].

hidden states. However, this technique presents some disadvantages. For instance, the *label bias problem* [11] implies that “the total probability mass received by y_{t-1} (based on $x_1, ..., x_{t-1}$) must be transmitted to labels y_t at time t regardless of the value of x_t”. Consequently, the only role of x_t is to influence which of the labels receive more of the probability at time t. In particular, all of the probability mass must be passed on to some y_t even if x_t is completely incompatible with y_t .

As for application, in [21], MEMMs and several HMMs were used to classify lines of FAQ webpage in 4 categories: head, question, answer and tail. The result of the comparision between MEMMs and HMMs proved to be favorable for the first method with a *segmentation precision* of 0.867 against 0.413 obtained for the HMMs.

2.4 Input-Output Hidden Markov Model

Input Output Hidden Markov Model were introduced in [5]. As said in the paper: ”The architecture can be interpreted as a statistical model and can be trained by the EM or generalized EM (GEM) algorithms (Dempster et al., 1977), considering the internal state trajectories as missing data”. The model learns to map an input sequence of data to an output sequence of data. The IOHMM can be modeled as a recurrent architecture composed with a set of state networks (s in Figure 3 and a set of output networks y

Machine Learning Model	Advantages	Limitations
Sliding Window	Convert sequential supervised learning to normal sequential learning	Can't see correlation between nearby y_t
Hidden Markov Model	Quite popular model, very efficient training	Can't capture long distance relationship interaction
Input-Output Hidden Markov Model	Supports long distance interactions	Label bias problem
Maximum Entropy Hidden Markov Model	Supports long distance interactions	Label bias problem
Conditional Random Field	Overcome the label bias problem	Slow convergence
Recurrent Neural Network	Good results, very popular	Requires huge amount of data

Table 1: Comparative table of probabilistic models for sequential data

in Figure 3). The output distribution of the time $t + 1$ is estimated with the inputs and state distribution of time t. Input-Output Hidden Markov Model is able as the MEMMs to catch long range interactions. The limitation of the IOHMM is the label bias problem discussed in detail in the limitations section of MEMMs. Input Output Hidden Markov Model was successfully used in [15] for the prediction of the electricity prices in the particular case of the Spanish Spot Market. The physical explanatory variables used as inputs for the IOHMM were time series of past energy production like nuclear generation, hydro generation, thermal generation.

2.5 Conditional Random Fields

Conditional Random Fields model is a generative probabilistic model very similar to MEMMs. As said in [19], ”The critical difference between CRFs and MEMMs is that a MEMM uses per-state exponential models for the conditional probabilities of next states given the current state, while a CRF has a single exponential model for the joint probability of the entire sequence of labels given the observation sequence”. The main advantage of the Conditional Random Fields model is to solve the label bias problem. A limitation of the Conditional Random Fields is the slow convergence of the training algorithm. The expensive training is what allows CRF to solve the label bias problem. Conditional Random Fields were applied in [19] among Hidden Markov Models and Maximum Entropy Markov Models on a part-of-speech tagging problem.

2.6 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are models for sequential data. There are several types of RNNs like Deep Long Short-term Memory introduced by G. Hinton in [16], Long Short Term Memory (LSTM) and others. A standard RNNs takes as input a sequence of data, computes a hidden vector sequence and outputs a vector sequence by iterations. To avoid overfitting, we traditionally use regularisation methods like early stopping and weight noise. Overfitting is observed when a machine learning model fits too closely the training data while it will not be able to generalise to unseen data. An evident advantage of using RNNs is the number of well known frameworks offering very good API to deal with RNNs. The popularity of RNNs these last years is obviously correlated with the very good results obtained by neural networks. One main issue when dealing with RNNs is the quantity of data required by the neural network to obtained good results. Several kind of neural networks were tested on the TIMIT corpus in [16] in phoneme recognition. The training was conducted on 462 speaker set records, with a separate set of 50 speakers for early stopping. The RNNs were tested with different kind of training method (CTC, Transducer, Pretrained Transducer), different number of hidden levels (between 1 and 5), and the number of LSTM cells in each hidden layer.

2.7 Discussion

Along this section, I presented six machine learning approaches to deal with sequential data. Among them, Hidden Markov Model and Recurrent Neural Network are very popular and studied. These models are supported by efficient libraries which make their use easy. For other methods, like MEMMs, the lack of available implementations made their use less frequent.

As we presented along this section, all models have their advantages and limitations and some models are more suitable to solve a concrete type of problems rather than others. For instance, we do not need Maximum Entropy Markov Model if we are not looking for long distance interactions within the sequences. Furthermore, the Conditional Random Field approach does not have the label bias problem of the MEMM. Last but not least, we must take into account the size of the data set we work with. From this point of view, Recurrent Neural Network approaches do not perform well on small data sets.

Regarding the size of our data set, the availability of implementation and the advantages of Hidden Markov Models with respect to other approaches, we propose this method to tackle our problem.

3 Theoretical background

As we saw in Section 2, there exists several methods to learn a sequence of actions. Among them, Markov models, which are a specific type of probabilistic models, are capable of learning sequences of states by using conditional probabilities. These models have been applied on temporal data such as climatic, financial, multimedia data or text corpus. In this section we describe in detail those models.

3.1 Markov Chain

A Markov Chain is a stochastic process where the probability of moving to the next state depends on the current state. A Markov Chain has to satisfy the Markov property which says that the conditional probability distribution of the next state only depends on the current state, and not on the past ones, also called as the memoryless property of a stochastic process because everything is encoded to the present and so we don't need knowledge of the past states to predict the future. In a more formal way, let's assume the Markov chain composed of a sequence of states: $X_0, X_1, ..., X_n$. The Markov Property is given by the formula:

$$p(x_i|x_0, \cdots, x_{i-1}) = p(x_i|x_{i-1}).$$

Consequently, the probability of having the Markov chain is:

$$p(x_0, ..., x_n) = p(x_0)p(x_1|x_0)p(x_2|x_1) \cdots p(x_n|x_{n-1}).$$

Figure 4 represents the graph of a Markov chain used to describe how we dress everyday, which can be seen as a Markovian process. One day, we can dress with casual wear, semi-casual wear or winter apparel according to the weather or our wishes. We can consider this model as a 3 states Markov chain. Then, the transition probability is given by the edges between the states. For instance, the probability of dressing with casual wear after a winter apparel day is 0.05, and the probability of dressing with semi-casual after a casual wear day is 0.3. In this example, the parameters of our model (the transition probabilities) are already given, so we are ready to predict a sequence of clothing with our Markov chain. In some cases, we have to compute the parameters of our model. This is done with MLE (Maximum Likelihood Estimation).

The *Maximum Likelihood Estimation* (MLE) is used to estimate the parameters of a model, in our case, the transition matrix of our Markov chain. To do it, the MLE is computed and the parameters are updated on each sequence.

3.2 Hidden Markov Model

Hidden Markov Model is a probabilistic model that can learn sequence of hidden states thanks to a sequence of observables. An observable is whatever that can be observed by the model and gives information about the current hidden state. Because states are not directly observed by the model, the probability that the model is in a state depends on the probability that the observable is seen at the current state, and of the probability that the model is in the current state knowing the previous state of the system at the time $t-1$ (Markov property). Given the sequence of hidden states $x_1, x_2, ..x_n$, and the sequence of observables $y_1, y_2, ...y_n$, the probability to be in the hidden state x_n with the observable y_n at the time t is given by the following formula:

$$p(y_1, ..., y_n, x_1, ...x_n) = p(x_1)p(y_1|x_1).p(x_2)p(y_2|x_2)...p(x_n)p(y_n|x_n).$$

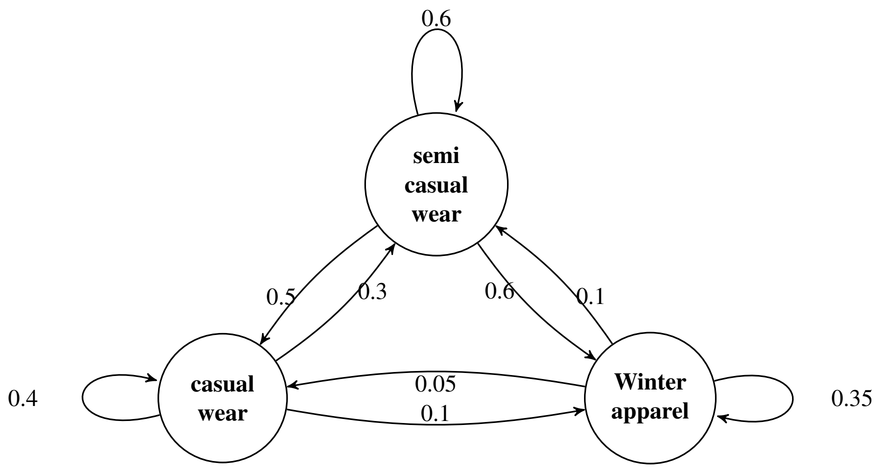


Figure 4: Markov Chain graph example

Figure 5 represents the Bayesian network of a Hidden Markov Model giving the weather from the kind of dress we wear. The hidden states of the model are given by the random variable $X(t)$ with $X(t) \in \{Hot, Mild, Cold\}$. Then the random variable $Y(t)$ represents observables of the system with

$$Y(t) \in \{Casual\ wear, Semi\ casual\ wear, Winter\ apparel\}.$$

The emission matrix B gives the probability of seeing the observable $Y(t)$ when the system is in the hidden state $X(t)$ at time t. For instance, the probability of having $Casualwear$ when we are in Hot is: $p(Casualwear|Hot) = 0.8$.

$$B = \begin{bmatrix} 0.8 & 0.2 & 0.0 \\ 0.4 & 0.5 & 0.1 \\ 0.1 & 0.2 & 0.7 \end{bmatrix}$$

As in the Markov chain, the transition matrix A contains the probability to enter in a state according to the previous state. So when I am in the hidden state Hot, the probability to enter in the hidden state $Mild$ is: $p(Mild|Hot) = 0.5$.

$$A = \begin{bmatrix} 0.3 & 0.5 & 0.2 \\ 0.3 & 0.5 & 0.2 \\ 0.2 & 0.4 & 0.4 \end{bmatrix}$$

The last parameter of the Hidden Markov Model is the initial probability distribution π of the model which gives for each state the probability to be in it at the time 0. $\pi(i) = (x_1 = i)$ with $i \in 1, 2, ..., n$.

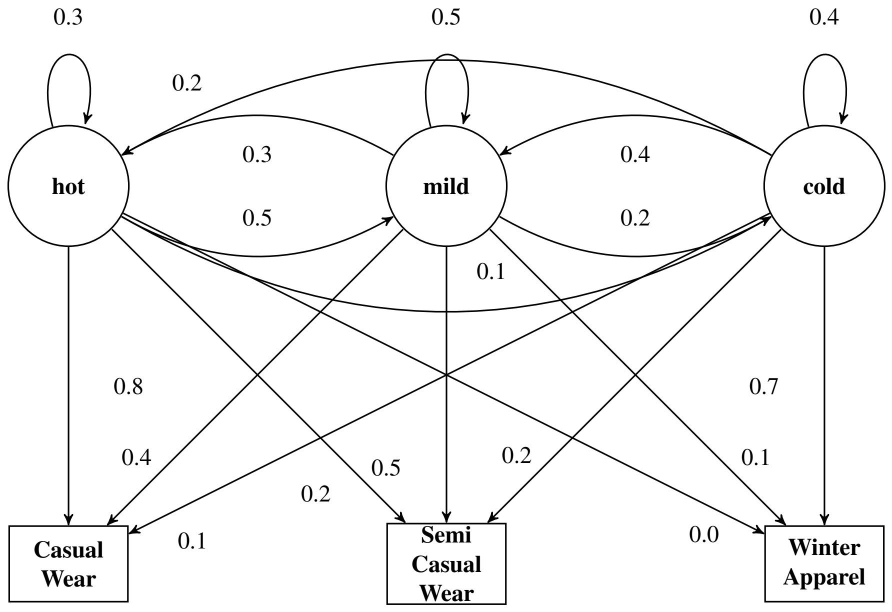


Figure 5: Bayesian network of a Hidden Markov Model

3.3 Baum-Welch algorithm

The forward-backward algorithm is used to learn the parameters of the Hidden Markov Model which are: the emission matrix, the transition matrix and some time the initial state distributions. To do it, the algorithm will use the entire data set containing a number of Markov sequences where each sequence is composed of observables.

The *forward procedure* computes the probability of being in the hidden state x_k given the sequence of previously seen observables $y_1, y_2, .., y_k$. The formula is:

$$\alpha_k(x_k) = \sum_{x_{k-1}=1}^{m} p(y_k|x_k)p(x_k|x_{k-1})\alpha_{k-1}(x_{k-1}), \tag{1}$$

for $k \in [2, n]$.

The backward procedure computes the probability of seeing the sequence $y_{k+1}, \cdots, y_n$ given the hidden state x_k. The formula is the following:

$$\beta_k(x_k) = \sum_{x_{k-1}=1}^{m} \beta_{k+1}(x_{k+1})p(y_{k+1}|x_{k+1})p(x_{k+1}|x_k) \tag{2}$$

for $k = 1, \cdots, n-1$.

4 Contribution

The initial goal was to train a Hidden Markov Model on the graph obtained by the approach of [8] and to compare the resulting Bayesian Network presented in Section 3.2 with the original graph.

Since the analysis of such Bayesian Network provides us with the information contained in the transition and emission matrix, we can learn useful information about the graph produced by the linear logic approach of [8]. For instance, things we could learn are the most common transition between the actions as well as eventualities and abnormalities produced in such sequences. Furthermore, it is also possible to find the semantics behind the obtained hidden states. This process is known as unsupervised sequential learning because the number of hidden states in the model is unknown so finding it is the main goal of the learning approach.

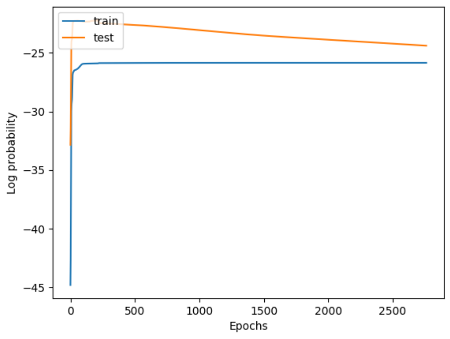
As the development of the project was in Python, it was interesting to look for a Python library implementing probabilistic models like Hidden Markov Model and Markov Chain. The library finally selected was *Pomegranate*[3] because it offers a very complete API with a detailed documentation. The library is written in C with a python wrapper. Pomegranate is a library specialized in probabilistic models such as Bayesian Networks, and Mixture Models. In our case, we used the Hidden Markov Chain class which have several methods to train the HMM such as the Baum-Welch algorithm. Callbacks are available, this allows us to keep an eye on the training step. I also modified the library in a way that permit us to follow the learning step of the HMM with the validation set. It allows us to be sure that there is no overfitting during the learning of the parameters.

4.1 Data set

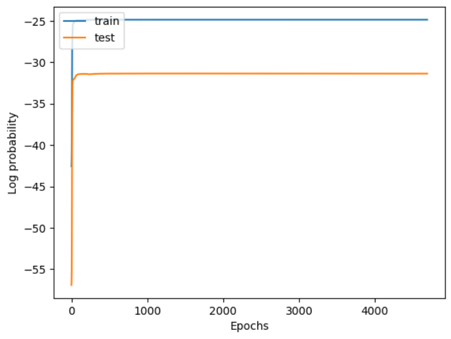
Two data sets were used for this study, the first one is from the software SWORD. And the second one is from the Protein Data Bank. Unfortunately, the linear logic approach (still under construction) used to generate the graph did not work. The main reason is that the grounding step (in Prolog) did not work cause of the huge amount of data. Finally, the data set was generated in Python with similar rules as describe in linear logic section. Thus a data set was created thanks to a workaround, it is not considered as the final data set. The SWORD's action graph is composed of more than 200 sequences, where the length of each sequence is between 1 and 350 actions with a mean length of 100 actions per sequence. An observable (action), can take 7 possible different values between 1 and 7. The Protein Data Bank (PDB) data set comes from the Data Science Challenge Website Kaggle. This data set is composed of more than 400 000 DNA sequences of different macro molecules from protein to virus. Because we didn't need the entire data set to train and test our HMM, we only took a subset of protein sequences, where each protein sequence has a length between 2 and 5037 with a mean of 266 element per sequence. The alphabet used to describe a protein sequence is composed of 23 letters. We only took 200 sequences in this set of data to test our HMM, more sequences would only increase the accuracy of the training but would also

[3]See https://pomegranate.readthedocs.io/en/latest/index.html

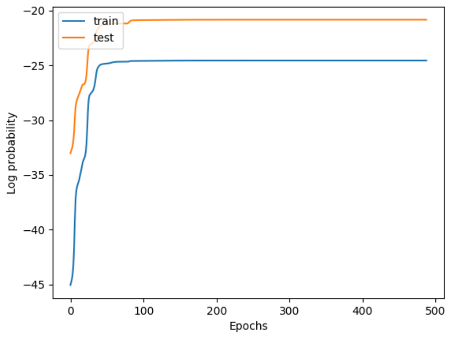
take more time for training our HMM.



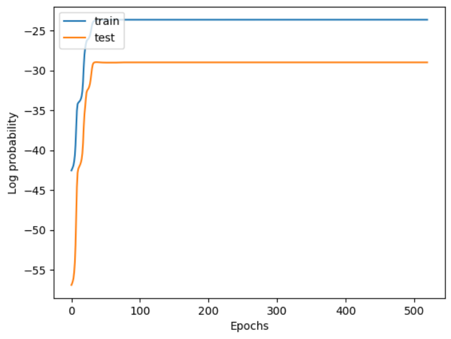
(a) 6 hidden states HMM training on k-fold 2



(b) 6 hidden states HMM training on k-fold 7



(c) 5 hidden states HMM training on k-fold 2



(d) 5 hidden states HMM training on k-fold 7

Figure 6: HMM training on SWORD's data set for several value of hidden states

4.2 Hyperparametrization

The main parameter in Hidden Markov Models is the number of hidden states. Increasing the number of hidden states will give more chance to fit the data, but will also increase the risk of having overfitting as said in [15]. A such HMM with a lot of hidden states will obviously become more complex, and so, it will be more difficult to interpret. The computational complexity of the HMM is directly dependant of the number of hidden states. The goal when trying to guess the number of hidden states is to find a good model without overfitting. The training of the HMM was conducted for different values of hidden states (between 2 and 11). And for each of these HMMs, a 10 fold cross validation was conducted. In our case, because each sequence of actions was of equal value, no weights were added to emphasise or inhibit the input. Also because it was unsupervised learning, no label were required. Other parameters like pseudo-count or inertia were ignored. The training was done using the Baum-Welch algorithm with the Pomegranate Python Library on the SWORD's data set and on the Protein Data Bank data set. On the PDB data set, the training needs many epochs (around 400 epochs depending of the cross validation set and the number of hidden states) before reaching the optimal log probability. The time required to get the best of the model was not surprising as the sequences were rather long. The training was also done with a 10-fold cross-validation and 2 to 11 hidden states HMM were trained. Figure 6 shows the training of 5 and 6 hidden states HMM on 2 different fold. The number of epochs required to converge is most of the time around 30 epochs which is not a lot in comparison with the training on the PDB data set. This could be explained by the fact that the data set is rather small. The training with 6 hidden states gives a better log probability for the 7th-fold than the 5 hidden states HMM for the same fold, we can observe an improvement from -31.5 to -29 for the 7th test fold but with a regression from to -24 to -25 for the 7th training fold between the 5 hidden states HMM and the 6 hidden states HMM. Though, there is a clear overfitting on the 2nd fold as we can see on 6a. All trainings on this fold with a number of 6 hidden states or a higher value give overfitting. Finally, the number of hidden states selected for the Hidden Markov Model was 5, it allows us to have the best of both world with a good enough log probability without suffering overfitting.

4.3 Results

The goal of the study was to analyse the HMM graph obtained after the training part and see how the HMM learned from the data. Figure 7 shows the Bayesian network of our 5 hidden states HMM trained directly from the SWORD's data set. On this graph, we can see 5 hidden states $s0$ to $s4$, and 6 actions from SWORD. The probability of entering in $s0$ for the first state is very high (0.9548) and that the action $order\_move$ is only observed by $s0$. This perfectly reflects the data set where nearly all sequences begin by the action $order\_move$. There is no surprise as a unit always receives an order to move at the beginning of a battle. Hidden states $s4$ and $s3$ have a very high probability (0.9) of staying in the same state which is the exact opposite of $s2$ and $s1$. The emission matrix probabilities shows that the probability of seeing an $order\_move$ when we are in $s0$ is 0.76, and 0.24 for $move\_completed$ which means that the HMM associated the two actions together. This is logical as we can consider that an action move (represented by $s0$) can be split in 2 actions which are an $order\_move$ (order given to the unit with a specific destination) followed by an action $complete\_move$ saying that the mission move was successfully executed. What is interesting here is that normally an action move in SWORD is decomposed as the following: an action $order\_move$,followed by a $pathfind$ and then an action $move\_completed$. But here the $pathfind$ is not at all considered as being a part of the hidden state $s0$. Instead, we have the states $s1$ and $s3$ which have a very high probability to observe a $pathfind$ (0.99 for $s1$ and 0.94 for $s3$). $s1$ has a good probability (0.83) to lead to $s2$ and 2 to lead to $s1$ (0.96), $s2$ being the main generator of the observable $partial\_complete\_move$. Here again we are not surprised as a $partial\_complete\_move$ happens often after a $pathfind$. Even if at first sight, it's rather strange having 2 hidden states generating the $pathfind$, we have in fact one ($s1$) hidden state binded to $pathfind$ which are immediately followed by $partial\_complete\_move$ and one ($s3$) hidden state which detects loops of $pathfind$ (in the sequences, there is loops of $pathfind$ which comes from the log of SWORD). The hidden state $s4$ has probabilities of seeing many actions, 0.1 for $move\_completed$, 0.13 for $pathfind$ and 0.71 for $start\_firing$. In fact, it seems that the HMM enter in the state $s4$ when it sees an action $start\_firing$ and associates all the previous and following actions to this action. Like a pattern containing the specific action $start\_firing$.

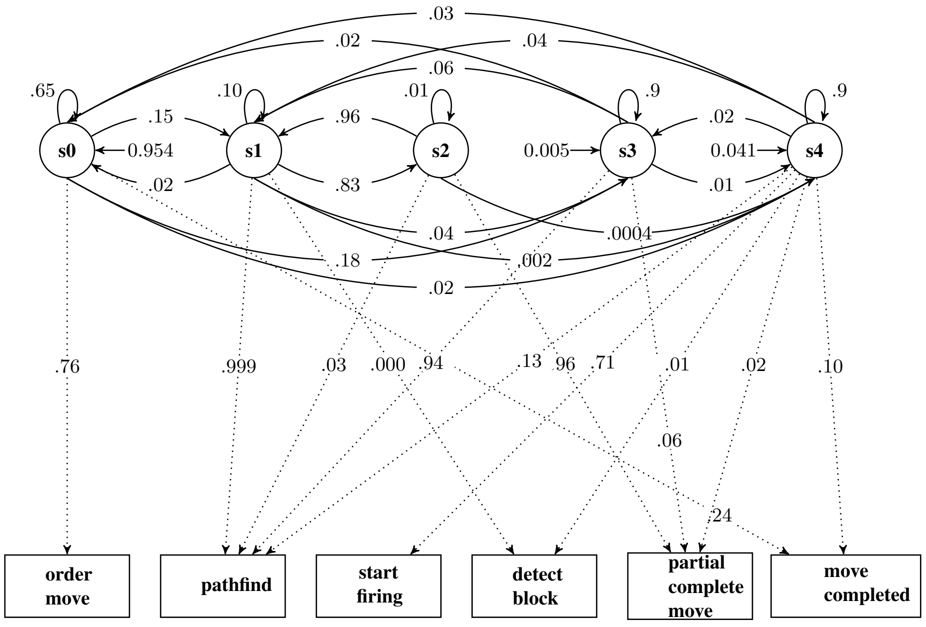


Figure 7: HMM graph

4.4 Anomaly detection

One purpose of the Markov chain is to compute the probability of having an entire sequence (of actions in our case). This can be done by computing the log probability after training the Markov Chain. For instance, let us assume that we have the following sequence of actions: Order_move, Partial_move, Explosion,... and the goal is to check if this sequence has a high probability of appearance. Because we work with log probability, the bigger is the log probability the higher is the probability of seeing the sequence.

The training of the Markov chain was done on the SWORD's data set. The parameters of the Markov Chain was found according to the Maximum Likelihood Estimation. Figure 8 represents the Bayesian network of the Markov Chain after the training with the transition probabilities written on the edges.

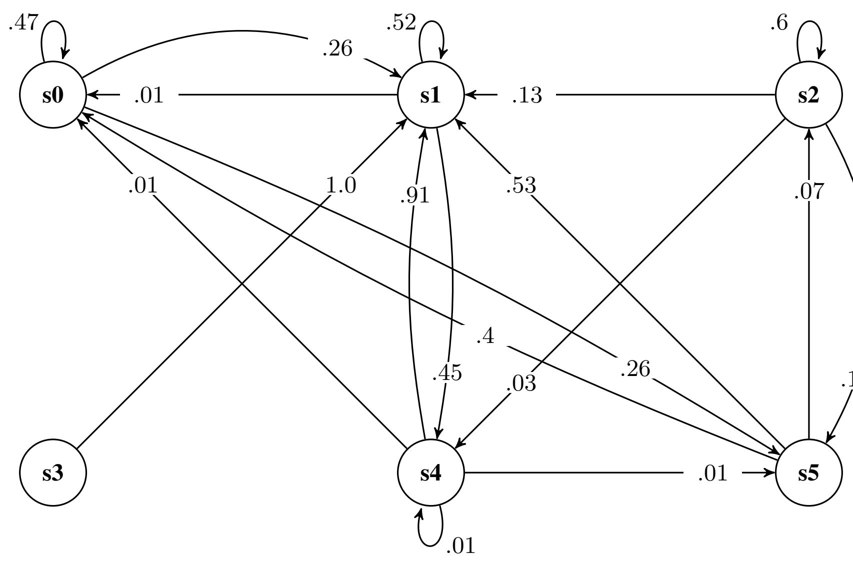


Figure 8: Markov Chain graph

After the training part, the Markov Chain was ready to compute what we call the log probability of a sequence. The log probability of a single sequence can tell if the sequence has a high probability of appearance according to the parameters of the model. Because the log probability (described by the Markov property in the previous section) is the probability of having the first action, multiplied by the probability of having the second one knowing the first one multiplied by the third etc. The log probability of a sequence decrease with its length. Furthermore, to be sure to get the right log probability independently of the length of the sequence, I divide each sequence's log probability by the length of the sequence. Then the purpose was to find the detection threshold. This was a tricky part because finding the good threshold is not obvious. Having a too low threshold will take normal sequence for not normal, and on the other hand, a too high threshold will lead to no detection at all which is not what we want. Figure 9 represents the box plot of the mean log probability of the sword's action sequences. We can notice the median at -0.76, and outlier values begin under -2.05. With a threshold at -2, the number of sequence detected as rare is 23 on the 283 sequences of the data set. This only represent 8% of the data set.

5 Conclusion

In this study, we presented a hybrid system (using linear logic and machine learning) to extract causal information from the training simulator SWORD. While the LL approach was used to generate actions from this log file, machine learning was applied to this graph in order to find anomalies and possible simplifications. The problem we faced

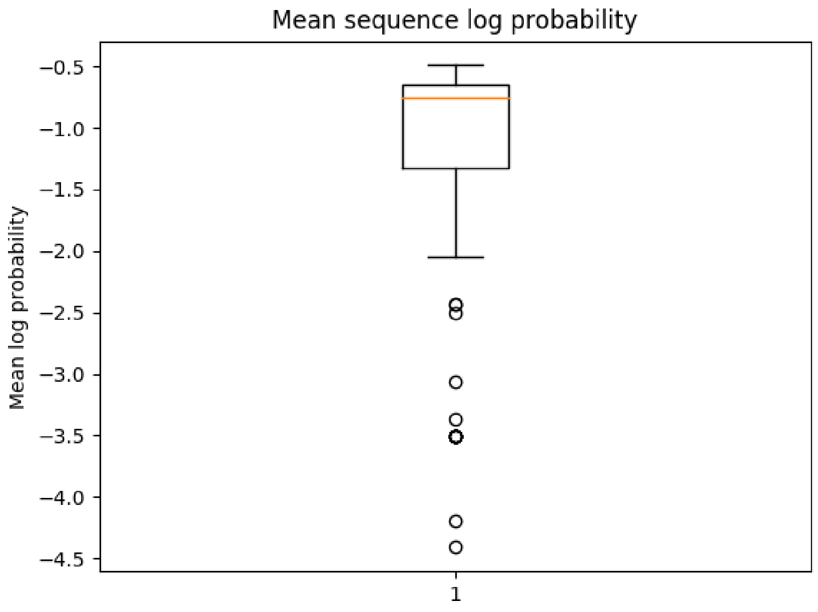


Figure 9: Markov Chain graph

concerns the huge amount of data produced by SWORD, which makes the debriefing process more complicated for the users.

In order to cope with this problem, we proposed a Hidden Markov Model, which is trained from this graph of actions in an unsupervised way. The final Hidden Markov Model will be able to accept sequences of actions with an associated probability. This approach would help the participants in a simulation in two different ways: it will help them to understand, from a statistical point of view, possible sequences that always go together and, moreover, identify sequences with low probability and associate them to "potential" anomalies.

In our examples, the HMM we considered was composed of 5 hidden states and thanks to it we could learnt many execution patterns from the data like subsequences of actions containing an action $start\_firing$ or succession of actions in the graph. A Markov Chain was also used on the sequence of actions to detect sequences with a low log probability.

Limitations one limitation of the Hidden Markov Model used in the previous section also discussed in the state of the art section is that HMMs cannot see long distance interaction between the states. In our case, we observed that our HMM got lots of regularities from the data, but we maybe missed some of them. An other limitation I noticed is the limited amount of data which constraints the model to use. Also, the accuracy of the model is correlated with the amount of data available. Then, the fact of working from the output of a previous algorithm can lead to a bias problem as the actions were elaborated by a human.

Future work several things could be done to improve the work. First we could try machine learning directly on the logs. Machine learning models like Long Short-Term Memory (LSTM) are well suited to deal with text data. The amount of text data which represents more than 300MB of text data per simulation will be far enough to apply neural networks method. We could then generate a graph from the logs without having to use the linear logic approach. Unfortunately, the SWORD's action graph generated by the linear logic couldn't be generated at the time of the realistic example. Even though a workaround was used to generated the actions from the logs, it would be interesting to accomplish it using Prolog with the linear logic approach. Using other machine learning models on the graph is also an other option and could potentially improve what we obtained with the HMM if for example, it exists long distance interactions that we didn't catch with the HMM. If more data can be generated, Recurrent Neural Network models would be a good start. Furthermore, we currently defined 6 actions from the logs, and it could be interesting to elaborate more actions. For instance, we thought about new actions like $unit\_detection$ when a unit detect an enemy unit. We should then see if the HMM could get new pattern from the data. Future work on the anomaly detection function for finding the best threshold would be interesting to do. We could use it to detect rare sequence and learn more information from the graph.

References

[1] G. Allen and R. Smith. “After action review in military training simulations”. In: *Proceedings of Winter Simulation Conference*. 1994, pp. 845–849.

[2] Samaneh Aminikhanghahi and Diane J. Cook. “A survey of methods for time series change point detection”. In: *Knowledge and Information Systems* 51.2 (2017), pp. 339–367.

[3] Andrea Asperti. “Causal Dependencies in Multiplicative Linear Logic with MIX”. In: *Mathematical Structures in Computer Science* 5.3 (1995), pp. 351–380.

[4] Ghulum Bakiri and Thomas G. Dietterich. “Achieving High-Accuracy Text-to-Speech with Machine Learning”. In: *In Data mining in speech synthesis*. Chapman and Hall, 1997.

[5] Yoshua Bengio and Paolo Frasconi. “An Input Output HMM Architecture”. In: *Proceedings of the 7th International Conference on Neural Information Processing Systems*. NIPS’94. MIT Press, 1994, pp. 427–434.

[6] Anne-Gwenn Bosser, Marc Cavazza and Ronan Champagnat. “Linear Logic for Non-Linear Storytelling”. In: *ECAI’10*. 2010, pp. 713–718.

[7] Anne-Gwenn Bosser, Ariane Ariane Bitoun, François Legras and Martín Diéguez. “Co-constructing Subjective Narratives for Understanding Interactive Simulation Sessions”. In: *INTWICED@AIIDE’18*. 2018.

[8] Anne-Gwenn Bosser, Ariane Bitoun, François Legras and Martín Diéguez. *Generating Actions from Traces in SWORD*. technical report. 2019.

[9] Simon Castellan and Nobuko Yoshida. “Causality in Linear Logic - Full Completeness and Injectivity”. In: *FOSSACS’19*. 2019, pp. 150–168.

[10] Lukas Chrpa. “Linear Logic in Planning”. In: *ICAPS’06*. Doctoral Consortium. 2019, pp. 26–29.

[11] Thomas G. Dietterich. “Machine Learning for Sequential Data: A Review”. In: *Structural, Syntactic, and Statistical Pattern Recognition*. Vol. 2396. Springer Berlin Heidelberg, 2002, pp. 15–30.

[12] Lucas Dixon, Alan Smaill and Tracy Tsang. “Plans, Actions and Dialogues Using Linear Logic”. In: *Journal of Logic, Language and Information* 18.2 (2009), pp. 251–289.

[13] Jean-Yves Girard. “Linear logic”. In: *Theoretical Computer Science* 50.1 (1987), 1–101.

[14] Jean-Yves Girard. “Linear Logic: A Survey”. In: *Logic and Algebra of Specification*. Ed. by Friedrich L. Bauer, Wilfried Brauer and Helmut Schwichtenberg. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 63–112.

[15] A. M. Gonzalez, A. M. S. Roque and J. Garcia-Gonzalez. “Modeling and forecasting electricity prices with input/output hidden Markov models”. In: *IEEE Transactions on Power Systems* 20.1 (2005), pp. 13–24.

[16] A. Graves, A. Mohamed and G. Hinton. “Speech recognition with deep recurrent neural networks”. In: *IEEE-ICASSP’13*. 2013, pp. 6645–6649.

[17] Max Kanovich and Jacqueline Vauzeilles. “Linear logic as a tool for planning under temporal uncertainty”. In: *Theoretical Computer Science* 412.20 (2011). Girard’s Festschrift, pp. 2072 –2092.

[18] Anders Krogh, Björn Larsson, Gunnar von Heijne and Erik L.L Sonnhammer. “Predicting transmembrane protein topology with a hidden markov model: application to complete genomes11Edited by F. Cohen”. In: *Journal of Molecular Biology* 305.3 (2001), pp. 567–580.

[19] John D. Lafferty, Andrew McCallum and Fernando C. N. Pereira. “Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data”. In: *ICML ’01*. Morgan Kaufmann Publishers Inc., 2001, pp. 282–289.

[20] Tshilidzi Marwala. *Handbook of Machine Learning*. World Scientific, 2018.

[21] Andrew McCallum, Dayne Freitag and Fernando C. N. Pereira. “Maximum Entropy Markov Models for Information Extraction and Segmentation”. In: *ICML’00*. Morgan Kaufmann Publishers Inc., 2000, pp. 591–598.

[22] Tim Miller. “Explanation in Artificial Intelligence: Insights from the Social Sciences”. In: *Artifical Intelligence* 267 (2019), pp. 1–38.

[23] J. Pearl. *Causality: Models, Reasoning and Inference*. 2nd. Cambridge University Press, 2009.