Optimization of a Sequential Decision Making Problem for a Rare Disease Diagnostic Application

Rémi Besson[1], Erwan Le Pennec[1,4], Emmanuel Spaggiari[2], Antoine Neuraz[2], Julien Stirnemann[2] and Stéphanie Allassonnière [3]

[1]*CMAP, École Polytechnique, Route de Saclay, 91128 Palaiseau, France*

[2]*Necker-Enfants Malades Hospital, Paris-Descartes University, 149 Rue De Sèvres, 75015 Paris, France*

[3]*School of Medicine, Paris-Descartes University, 15 Rue de l'École de Médecine, 75006 Paris, France*

[4]*XPop, Inria Saclay, 91120 Palaiseau, France*

Keywords: Symptom Checker, Stochastic Shortest Path, Decision Tree Optimization, Planning in High-dimension.

Abstract: In this work, we propose a new optimization formulation for a sequential decision making problem for a rare disease diagnostic application. We aim to minimize the number of medical tests necessary to achieve a state where the uncertainty regarding the patient's disease is less than a predetermined threshold. In doing so, we take into account the need in many medical applications, to avoid as much as possible, any misdiagnosis. To solve this optimization task, we investigate several reinforcement learning algorithms and make them operable in our high-dimensional setting: the strategies learned are much more efficient than classical greedy strategies.

1 INTRODUCTION

Motivation. The development of a decision support tool for medical diagnosis has long been an objective. In the 1980s, researchers were already trying to develop an algorithm that would allow them to find the patient's disease by sequentially asking questions about the relevant symptoms.

Recent works aim to apply the breakthrough of reinforcement learning (RL) to learn good diagnostic strategy, see for example (Chen et al., 2019). We present in this work our contributions to this objective of adapting the new progress of machine learning for the optimization of symptom checkers.

We are particularly interested in the field of rare diseases where such a tool is particularly desirable. Indeed, no doctor has the encyclopedic knowledge to take into account all diagnoses, because rare diseases are numerous and each of them will be observed only a very small number of times or never in the doctor's career. As a result, rare diseases are often underdiagnosed or diagnosed late after medical wandering.

A symptom checker for the prenatal diagnosis of rare diseases by fetal ultrasound would be even more useful. Indeed, in this particular case, the patient can not describe the symptoms himself and the physician has to check several anatomical regions looking for abnormalities that can be hard to detect.

We aim to help the practitioner make the diagnosis with a high probability while minimizing the average number of symptoms to be checked. To do this, we design an algorithm that proposes the most promising symptoms to be checked at each stage of the medical examination and provides the probability of each possible disease. Eventually, our algorithm must be operable and interpretable online at the bedside.

Note that this article only presents the optimization task associated with recommending the next symptoms to be checked. Several other modules are needed to build the final decision support tool. We detail some of them in (Besson, 2019), including the one that builds the environment model and another that deals with the granularity of symptoms.

Available Data and Some Dimensions. The data is structured as a list of diseases with their estimated prevalence in the whole population and a list of symptoms for each disease. We will call them associated or typical symptoms. We also have an estimation of the probability of the symptom given the disease. We denote B_i the binary random variable (r.v) associated to the presence/absence of symptom of type i.

All this information, that we will refer to as expert data, has been provided by physicians of Necker hos-

Besson, R., Pennec, E., Spaggiari, E., Neuraz, A., Stirnemann, J. and Allassonnière, S.
Optimization of a Sequential Decision Making Problem for a Rare Disease Diagnostic Application.
DOI: 10.5220/0008938804750482
In *Proceedings of the 12th International Conference on Agents and Artificial Intelligence (ICAART 2020) - Volume 2*, pages 475-482
ISBN: 978-989-758-395-7; ISSN: 2184-433X

pital based on the available literature. We map all the symptoms found in the literature to the Human Phenotype Ontology (HPO) (Köhler and al., 2017). HPO is a recent work which provides a standardized vocabulary of phenotypic abnormalities encountered in human disease. We use it to harmonize the terminology. We could then combine our list of symptoms per disease and map it to OrphaData [1]. OrphaData is useful to fill the missing data on prevalence of symptoms in the diseases. We restrict our analyses to the subset of symptoms that can be detected using fetal ultrasound.

We assume, in this work, that we know the joint distribution of the typical symptoms given the disease. This issue is addressed in (Besson et al., 2019). We proposed a method to mix expert data (the marginals and some additional constraints as for example censored combinations) with clinical data which is collected as the algorithm is used. We therefore focus in this work on the planning task since we assume the environment model to be known and fixed.

Currently, our database references 81 diseases and 220 symptoms. We make the assumption that a patient presents only one disease at a time which is a reasonable hypothesis in the rare disease framework.

Main Contributions. The main contributions of this work are the following: we propose a novel notion of what should be a good symptom checker, taking into account the need in medicine to have a high level of confidence in the diagnosis made. This led us to formulate the task of optimizing symptom checkers as a stochastic shortest path problem. We also detail in section 4.4 a new algorithm that we call DQN-MC-Bootstrap inspired by DQN from (Mnih et al., 2013). We have drastically reduced the computing power needed to solve our problem by partitioning the state space and appropriately using the connections between the subtasks with DQN-MC-Bootstrap.

2 RELATED WORKS

Shortest Path Algorithms. $A^\star$ is a classic algorithm which finds the minimum cost path between a start node and a final node through a deterministic graph (Hart et al., 1968). (Zubek and Dietterich, 2005) proposed an algorithm inspired of a variant of $A^\star$ for the optimization of the medical diagnostic procedure. Nevertheless such an approach is not tractable in a high-dimensional graph since its assume the possibility to store the whole tree of the paths investigated. Some improvements using less memory have been proposed as IDA* (Korf, 1985). However, this algorithm does not exactly match our problem. We are not looking for a single shortest path in a deterministic graph, but rather to find for each node of the graph the shortest path to the objective states. Indeed, since we allow the doctor to answer a different question than the one we propose, we want to have a good solution even in a part of the tree that is not the optimal path. In this sense, we cannot avoid using some form of policy parameterization in the hope that a good solution on the optimized part of the tree will have learned a good enough representation of the state to be good on the unvisited part of the tree.

Disease Diagnostic Task using RL Algorithms. Recent works (Chen et al., 2019), (Peng et al., 2018), (Tang et al., 2016) focus on this problem of optimizing symptom checkers using RL algorithms. Nevertheless our approach is fairly different to these previous works. They formulated their optimization problem as a trade-off between asking less questions and making the right diagnosis while we formulate it as the task of reaching as quick as possible, on average, a pre-determined high degree of certainty about the patient disease. In our case, the parameter to be tuned is the degree of certainty we want at the end of the examination: we should stop when the entropy of the disease falls below this threshold ε. The smaller the ε the more symptoms our algorithm will need before considering that the game ends.

(Tang et al., 2016) makes use of a discounted factor $\gamma \in [0,1[$ in their reward signal design. The reward associated to each question is zero until possessing a diagnosis (which is an additional possible action) where the reward is equal to γ^q (if the guess was correct, 0 otherwise), q being the number of questions that have been inquired before possessing the diagnosis. In this context γ makes the compromise between asking fewer question and making the right diagnosis. The smaller γ, the more likely the algorithm is to make a wrong diagnosis by trying to ask fewer questions.

3 A MARKOV DECISION PROCESS FRAMEWORK

3.1 The Optimization Problem

Modelization. We formulate our sequential decision making problem using the Markov Decision Process (MDP) framework. For the state space $\mathbb{S}$, we use

[1]Orphanet. INSERM 1997. An online rare disease and orphan drug data base. http://www.orpha.net. Accessed [02/10/2018]

the ternary base encoding 1 if the considered symptom is present, 0 if it is absent, 2 if non observed yet:

$$\mathbb{S} = \{(2,\dots,2),(1,2,\dots,2),\dots,(0,\dots,0)\}.$$

An element $s \in \mathbb{S}$ is a vector of length 220 (the number of possible symptoms), it sums up our state of knowledge about the patient's condition: the i-th element of s encode information about the symptom whose identifier is i.

Concerning the action space $\mathbb{A}$, we write $\mathbb{A} = \{a^1,...,a^{220}\}$. An action is a symptom that we suggest to the obstetrician to look for, more specifically a^j is the action to suggest to check symptom j.

Our environment dynamic is by construction Markovian in the sense that $\mathbb{P}[s_{t+1} \mid a_t, s_t, a_{t-1}, s_{t-1}, ...a_0, s_0] = \mathbb{P}[s_{t+1} \mid a_t, s_t]$ where a_t is the action taken at time t.

Diagnostic Policy. We aim to learn a diagnostic policy that associates each state of knowledge (list of presence/absence of symptoms) with an action to take (a symptom to check), $\pi : \mathbb{S} \to \mathbb{A}$. What should be a good diagnostic policy? Many medical applications consider a trade-off between the cost of performing more medical tests (measuring it in time or money) and the cost of a misdiagnosis (Tang et al., 2016), (Zubek and Dietterich, 2005).

However in our case the cost of performing more medical tests (i.e to check more possible symptoms) is negligible against the potential cost of a misdiagnosis. In theory, the obstetrician have to check all possible symptoms to ensure the fetus does not present any disease. Therefore we do not take the risk of a misdiagnosis by trying to ask fewer questions. However if the physician observes a sufficient amount of symptoms he can stop the ultrasound examination and perform additional tests, like an amniocentesis, to confirm his hypotheses.

This is why we can label some states as terminal: they satisfy the condition that the entropy of the r.v disease is so low that we have no doubt on the diagnosis. Our goal is to minimize the average number of inquiries before reaching a terminal state:

$$\pi^\star = \underset{\pi}{\arg\min}\ \mathbb{E}_{\mathcal{P}}\big[I \big| s_0, \pi\big], \tag{1}$$

where $s_0 = (2,...,2)$ is the initial state, $\mathcal{P}$ the law of the environment currently used, π the diagnostic policy, and I is the random number of inquiries before reaching a terminal states, i.e:

$$I = \inf\{t \mid H(D \mid S_t) \le \varepsilon\} \tag{2}$$

where $H(D \mid S_t) = \sum_{s_t} \mathbb{P}[S_t = s_t] H(D \mid S_t = s_t)$ is the entropy of the r.v disease D given what we know at time t: S_t. We should think s_t as a realization of S_t.

Note that we are not ensured that for all t we had $H(D \mid s_{t+1}) \le H(D \mid s_t)$. Nevertheless this inequality holds when taking the average $H(D \mid S_{t+1}) \le H(D \mid S_t)$, see theorem 2.6.5 of (Cover and Thomas, 2006), "information can't hurt". Then, when we consider that entropy is sufficiently low and that we can stop and propose a diagnosis, we know that on average, the uncertainty about the patient's disease would not have increased if we had continued checking symptoms.

Setting a reward function as follow, $\forall s_t, a_t$:

$$r_{t+1} := r(s_t, a_t) = -1, \tag{3}$$

we can write (1) as a classical episodic reinforcement learning problem (Sutton and Barto, 2018):

$$\pi^\star = \underset{\pi}{\arg\max}\ \mathbb{E}_{\mathcal{P}}\left[\sum_{t=1}^{I} r_t \middle| s_0, \pi\right]. \tag{4}$$

In RL such a reward design is called action-penalty representation, since the agent is penalized for every action executed. It is notably a classic way to model shortest path problems.

The Use of the Environment Model. Theoretically the environment model that we have at our disposal, the joint distribution of the combination of symptoms given the disease, provides us with a transition model.

Nevertheless, in practice, we do not use directly the model. It is indeed not possible to store the transition matrix for dimensional reasons. That is why our only alternative to solve the optimization problem (4) is to simulate games by recalculating the transition probabilities on the fly. Moreover, the incremental nature of the states implies that the cost for simulating a game from the start s_0 to a certain state s_t is approximately the same than the cost to do an unique transition from state s_{t-1} to s_t. That is why our environment model should be looked in practice as a simulator of games starting from state s_0 to s_I.

Finally, note that a subtlety of our stopping criterion (2) is that it requires us to compute the probabilities of the diseases given the symptoms combination of the current state at each step of the medical examination. Indeed for dimensional reasons we can not store the set of goal states and have to check at each step if we can stop and possess a diagnostic or not.

Note that (4) is more a planning problem than a RL one since we assume to know the environment law $\mathcal{P}$ and aim to solve the MDP associated. Nevertheless, since the dimension of our problem is high, we solve the MDP by sampling trajectories and then treating the environment model as a simulator which blurs the boundaries between RL and planning.

3.2 Decomposition of the State Space

Our full model is of very high dimension: we have 220 different symptoms and then theoretically $3^{220} \approx 10^{104}$ different states. Thus a tabular dynamic programming approach (section 4.1) is impossible. According to our experiments, a classical Deep-Q learning is also not numerically tractable (section 4.2).

In order to break the dimension, we capitalize first on the fact that the physicians use our algorithm mainly after seeing a first symptom. In such case, we make the assumption that this initial symptom is typical. It might be possible to have a disease which also presents a non-typical symptom but this happens with a low probability. Anyway, in this case, we would end up with a high entropy and no disease identification. This leads to switch to another strategy. With such an assumption the dimension drops significantly since we now only consider diseases for which this initial symptom is typical, the only relevant symptoms are the one which are typical of these remaining diseases.

Therefore we created 220 tasks $\mathcal{T}_i$ to solve. We denote $\mathbb{B}_i = (B_{i_1}, ..., B_{i_k})$ the set of symptoms related with the symptom i, i.e this is the set of symptoms which are still relevant to check after observing the presence of symptom i. For all i, we start from state $s_{(i)}$ which is the state of length $|\mathbb{B}_i|$ with the presence of symptom B_i and no other information on the others symptoms, and we aim to solve:

$$\pi^\star_{(i)} = \arg\max_\pi \mathbb{E}_{\mathcal{P}} \left[\sum_{t=1}^{I} r_t \mid s_{(i)}, \pi \right]. \qquad (\mathcal{T}_i)$$

In RL, there exist several ways to solve a problem like $(\mathcal{T}_i)$. If the dimension is small enough it is possible to find the optimal solution explicitly using a dynamic programming algorithm (see section 4.1). If the number of states is too high we have to parameterize the policy, section 4.5, or to parameterize the Q-values, section 4.2. We have investigated both approaches to solve our problem.

4 A VALUE-BASED APPROACH

4.1 Look-up Table Algorithm

We recall that the Q-values are defined as $Q_\pi(s,a) = \mathbb{E}\left[\sum_{t'=t}^{I} r_{t'} \mid s_t = s, a_t = a, \pi\right]$. This is the expecting amount of reward when starting from state s, taking action a and then following the policy π. The optimal Q-values, are defined as $Q^\star(s,a) = \max_\pi Q_\pi(s,a)$ and satisfy the following Bellman optimality equation: $Q^\star(s,a) = r(s,a) + \sum_{s'} \mathbb{P}[s' \mid s,a] \max_{a' \in \mathbb{A}} Q^\star(s',a')$

The optimal policy $\pi^\star$, is directly derived from $Q^\star$: $\pi^\star(s) = \arg\max_a Q^\star(s,a)$. Therefore we "only" need to evaluate $Q^\star(s,a)$, $\forall s,a$. This can be done by a value-iteration algorithm which uses the Bellman equation as an iterative update:

$$Q^\star_{k+1}(s,a) \leftarrow r(s,a) + \sum_{s'} \mathbb{P}[s' \mid s,a] \max_{a' \in \mathbb{A}} Q^\star_k(s',a').$$

It is known, see (Sutton and Barto, 2018), that $Q_k \to Q^\star$ when $k \to \infty$. The main limitation of this algorithm is to require the storage of all Q-values which is of course not possible when the size of the state space is large.

4.2 Q-learning with Function Approximation

To cope with the high dimensionality of the state space, the idea is to parameterize the Q-values by a neural network: $Q(s,a) \approx Q_w(s,a)$ called Q-network. It takes as input the state and output the different Q-values. The heuristic for training such a Q-network is to simulate transitions (s_t, a_t, r_t, s_{t+1}) with an ε-greedy version of the current Q-values (to enforce exploration) and, at each iteration i, to minimize the following loss function:

$$L(w_i) = \mathbb{E}\left[\big(\underbrace{r_t + \max_{a'} Q_{w_i}(s_{t+1}, a')}_{\text{target}} - Q_{w_i}(s_t, a_t) \big)^2 \right].$$

To successfully combine deep learning with RL, (Mnih et al., 2013) proposed to use experience replay to break correlation between data: build a batch of experiences (transitions s, a, r, s') from which one samples afterwards. Another trick is to freeze the target network during some iterations to overcome the learning instability. The update then becomes where Q_{w^-} is the frozen network:

$$w_{t+1} \leftarrow w_t - \alpha\Big(r_t + \max_{a'} Q_{w^-}(s_{t+1}, a') - Q_w(s_t, a_t)\Big) \frac{\partial Q_w(s_t, a_t)}{\partial w}$$

4.3 The Update Target: TD vs MC

There are different alternatives for the definition of the update target, one can use the Monte Carlo return or bootstrap with an existing Q-function. We recall (Sutton and Barto, 2018) that an algorithm is a bootstraping method if it bases its update in part on an existing estimate. This is the case of the Temporal-Difference (TD) algorithm defined as:

$$Q_{k+1}(s,a) \leftarrow \underbrace{Q_k(s,a)}_{\text{old estimate}} + \underbrace{\alpha \left(r(s,a,s') + \max_{a'} Q_k(s',a') - Q_k(s,a) \right)}_{\textit{update}}$$

where s, a, s' is sampled using the current policy in a ε-greedy way. Q_k is the estimate at iteration k, α the learning rate. On the contrary a Monte-Carlo method does not bootstrap:

$$Q_{k+1}(s,a) \leftarrow Q_k(s,a) + \alpha(G - Q_k(s,a))$$

where G is the reward from a simulated game.

Usually TD method is seen as a better alternative than MC method which is often discarded because of the high variance of the return.

Nevertheless our case study is specific: we face a finite-horizon task with a final reward: the reward signal is not very informative before reaching a terminal state while it is well known that TD is slow to propagate rewards through the state space. In addition, for the subproblems of intermediate dimensions, we are ensured that games do not last too much time and then that there is a small variance in the return of the Monte-Carlo episodes. It should also be noted that using the return of a simulated game G as a target produces a true stochastic gradient with all the classical guarantees of convergence.

We show, see section 5, that DQN-MC performs well on small and intermediate sub-tasks of our problem while DQN-TD appears very sensitive to the chosen learning rate and can diverge. This is why we chose to use DQN-MC instead of DQN-TD. It is, indeed, a well-known issue sometimes referred as "deadly triad" (Sutton and Barto, 2018) that combining function approximation, off-policy learning and bootstrap to compute the target (what the DQN-TD algorithm does) is not safe. These observations are consistent with some recent works as (Amiranashvili et al., 2018) which show that MC approaches can be a viable alternative to TD in the modern RL era.

The higher dimensional tasks are harder to solve because the games are expected to last longer which is a challenge both in term of computing time that in terms of learning stability (higher variance of the return). To scale up on such problems, we break down the state space into a partition and leverage already solved sub-tasks as bootstrapping methods resulting in a kind of n-step bootstraping method.

4.4 Bootstrapping with Solved Sub-tasks

Recall $\mathbb{B}_i = (B_{i_1}, \ldots, B_{i_k})$ is the set of symptoms related with the symptom i. When $|\mathbb{B}_i|$ is small enough (say $|\mathbb{B}_i| < 12$), we can learn the optimal policy $\pi^\star$ by a simple Q-learning lookup table algorithm.

Considering intermediate dimension problems (say $11 < |\mathbb{B}_i| < 31$) we can use the DQN-MC algorithm which performs pretty well on these problems. For high-dimensional problems ($|\mathbb{B}_i| > 30$) using directly the DQN algorithm would be time-consuming.

Algorithm 1: DQN-MC-Bootstrap.

```
Start with low dimensional tasks.
for i such that the task $\mathcal{T}_i$ has not been yet optimized do
  if $|\mathbb{B}_i| \leq 30$ then
    while the budget is not exhausted do
      Play 100 games ($\varepsilon$-greedy) from the start $s_{(i)}$ to
      a terminal state.
      Integrate all the obtained transitions to the
      Replay-Memory
      Throws part of the Replay-Memory away (the
      oldest transitions of the replay)
      Sample $1/20$ of the Replay-Memory
      Perform a gradient ascent step on the sample
    end while
  end if
end for

Continue with higher-dimension tasks.
while there are still tasks to be optimized do
  Choose the easiest task to optimize: the one with
  the highest weighted proportion of subtasks already
  solved
  while the budget is not exhausted do
    Play 100 games ($\varepsilon$-greedy) from the start $s_{(i)}$ to a
    terminal state (condition $\star$)
    or to a state that was yet encountered in an already
    solved task (condition $\star\star$)
    if we stopped a game because of condition $\star\star$ then
      Bootstrap i.e use the network of the sub-task to
      predict the average number of question to reach
      a terminal state
    end if
    Integrate the transitions to the Replay-Memory
    Throws part of the Replay-Memory away
    Sample $1/20$ of the Replay-Memory
    Perform a gradient ascent step
  end while
end while
```

An easy way to accelerate the learning phase is to make use of the related sub-tasks previously solved. Indeed if B_i is a symptom for which $|\mathbb{B}_i|$ is high, there must have some $B_j \in \mathbb{B}_i$ such as $|\mathbb{B}_j|$ is small enough and therefore such as the Q-values of $\pi^\star_{(j)}$ have been yet computed or at least approached. Then, when we try to learn the optimal Q-network of a given problem, we yet know, for some inputs, the Q-values that should output a quasi-optimal Q-network.

Then the idea is to play games starting from the initial state $s^{(i)}$ and bootstrap when reaching a state that belongs to a state set of the partition where there yet exist an optimized network. Each time we receive a positive answer we checks if there already exists a network optimized for such a starting symptom. If this is indeed the case, the current game is stopped and the corresponding optimized network is called to predict the average number of question to ask to reach a terminal state. The main lines of the whole procedure

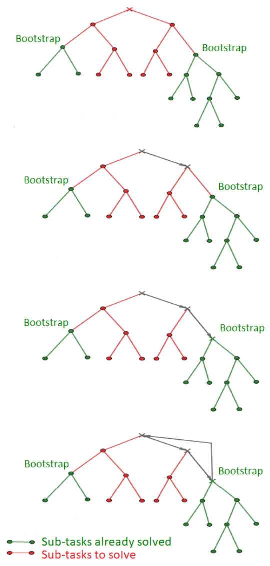

Figure 1: Example of the iterations of DQN-MC-Bootstrap.

are summarized in Algorithm 1 and Figure 1 provides a visual interpretation.

In a certain way our new algorithm DQN-MC-Bootstrap is a n-step method with a random n which corresponds to the random time where we jump from one element to another of the state space partition. Note that in doing so, we do not optimize the network for the entire task and then do not need a more complex architecture for the higher dimension task.

4.5 Baseline Algorithms

Breiman Algorithm. We will call Breiman policy, the classic greedy algorithm which consists in choosing at each node the features which minimize the average entropy of the target (Breiman et al., 1984). More precisely $\forall s \in \mathbb{S}$: $\pi^{\text{Breiman}}(s) = \arg\max_{a\in\mathbb{A}} H(D \mid s) - \mathbb{E}[H(D \mid s,a)]$.

The quantity maximized is the *information gain*, it measures the average loss of uncertainty about the target outcome that produces action a.

Our Baseline: An Energy-based Policy with Hand--crafted Features. As long as the environment $\mathcal{P}$ is well known and fixed, the optimal policy $\pi^\star$ is deterministic. Nevertheless in our application it can be interesting to propose several symptoms to check at the user instead of a single one. Indeed the physicians might be reluctant to use a decision support tool which do not let them a part of freedom in their choice. This is why we consider for our baseline an energy-based formulation, a popular choice as in (Heess et al., 2013): $\pi_\theta(s,a) \propto e^{\theta^T\phi(s,a)}$ where $\pi_\theta(s,a)$ is the probability to take action a in state s, $\phi(s,a)$ is a feature vector: a set of measures linked with the interest of taking action a when we are in state s. To be more precise $\phi(s,a)$ is a three-dimensional vector which includes the information gain provided by action a in state s, but also the probability of presence of the symptom suggested by action a when being in state s and finally a binary function indicating whether the suggested symptom is typical of the currently most plausible disease.

The proposed π_θ is equivalent to a neural network without hidden layer designed with hand-crafted features. When properly optimized this policy outperforms by construction the Breiman policy. A REINFORCE algorithm (Williams, 1992) with a baseline to reduce the variance is perfectly suitable in our case and exhibits good results, see section 5.

5 NUMERICAL RESULTS

For all the experiments, we used the same architecture for the neural networks (see Table 1) and the ε parameter of our stopping criterion is set to 10^{-6}.

Table 1: Neural network architecture for task $\mathcal{T}_i$. $|\mathbb{B}_i|$ the number of remaining relevant symptoms to check.

Name	Type	Input Size	Output Size
L1	Embedding Layer	$\lvert\mathbb{B}_i\rvert$	$3\times\lvert\mathbb{B}_i\rvert$
L2	ReLu	$3\times\lvert\mathbb{B}_i\rvert$	$2\times\lvert\mathbb{B}_i\rvert$
L3	ReLu	$2\times\lvert\mathbb{B}_i\rvert$	$\lvert\mathbb{B}_i\rvert$
L4	Linear	$\lvert\mathbb{B}_i\rvert$	$\lvert\mathbb{B}_i\rvert$

Our Baseline has Quasi-optimal Performances on Small Subproblems. Figure 2 compares on some of our subtasks the performance of the two policies proposed in section 4.5: the energy-based policy and the classic Breiman policy. We added the true optimal policies when it was possible to compute the latter, i.e. when the dimension was small enough.

Our energy-based policy appears to clearly outperforms a classic Breiman algorithm and all the more so as the dimension increases: the average number of questions to ask may be divided by two in some cases. On small subproblems where we have been able to compute the optimal policy, our energy-based policy appears to be very close to the optimal policy.

DQN-TD is Much More Instable than DQN-MC. We also implemented the classical DQN algorithm with TD method: DQN-TD. We kept the main features of DQN-MC in order to facilitate their compari-

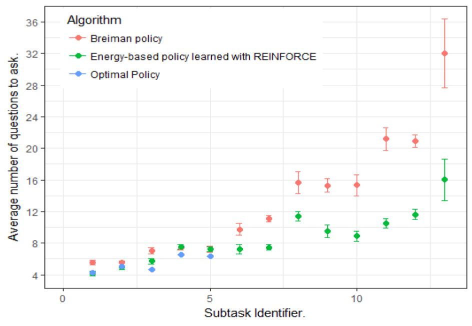

Figure 2: Comparison of Breiman and our energy based-policy on several subtasks.

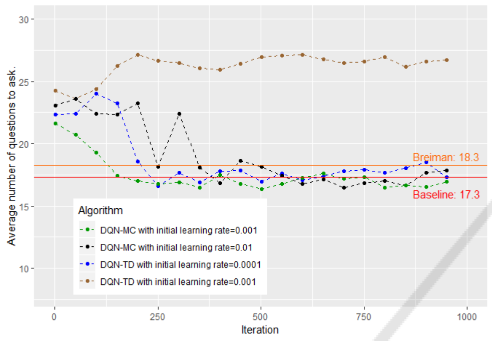

Figure 3: DQN-TD vs DQN-MC. Task dimension: 29.

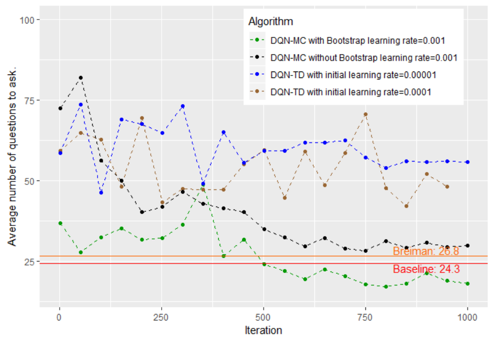

Figure 4: Comparison of DQN-TD, DQN-MC and DQN-MC-Bootstrap. Task dimension: 104.

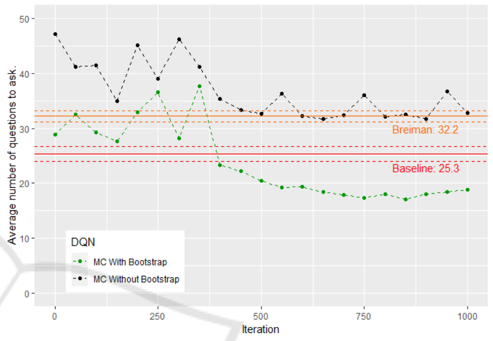

Figure 5: Evolution of the performance of the neural network during the training phase. Task dimension: 70.

son. The learning rate is initialized with a lower value than in the DQN-MC algorithm but it is decreased in exactly the same way in both cases: divided by two each 300 iterations. Another difference is the frozen network we use as target in DQN-TD which is not needed in DQN-MC. We update the frozen network each 2 iterations (we have also tried to update it less frequently but have not observed any major differences with the results presented here).

We compared these two algorithms, DQN-MC and DQN-TD, on severals of our sub-tasks (see Figures 3 and 4). We did not observed much difference on small and intermediate sub-problems: both algorithms converge at the same speed towards solutions of the same quality. Nevertheless DQN-TD appears much more sensitive to the learning rate. Indeed as it can be seen in Figure 3, DQN-TD converge on this problem, where it remains 29 relevant symptoms to check and 8 possible diseases, when the learning rate is initialized at 0.0001. Nevertheless if the learning rate is chosen a little bit higher, at 0.001, DQN-TD diverge. On the contrary, DQN-MC converge when the learning rate is initialized to 0.001 and also when initialized to 0.01 even if the returns of the algorithm are less stable in this latter case. These observations have to be combined with the one of Figure 4 where it remains 104 relevant symptoms to check and 18 possible diseases. In this case DQN-TD with an initial learning rate of 0.0001 diverge. Reducing the learning rate to 0.00001 does not change this fact. On the contrary we do not need to reduce the initial learning rate of DQN-MC (we take 0.001) to make it converge to a good solution. Since we have to train as many neural networks as the number of sub-tasks, we need a robust algorithm able to deal with different task complexity without changing all the hyper-parameters.

Bootstraping on Already Solved Sub-tasks Helps a Lot. In these experiments, we compare the performance of a simple DQN-MC algorithm against DQN-MC-Bootstrap on some tasks. The two algorithms use exactly the same hyper-parameters, the only difference being the bootstrap trick of DQN-MC-Bootstrap.

Figures 4 and 5 show the benefits of using the solved sub-tasks as bootstraping methods. In both cases a simple DQN-MC is unable to find a good solution while a DQN-MC-Bootstrap outperforms quickly our baseline. Note that the neural network trained with DQN-MC-Bootstrap starts with a policy that is not that bad. It is appreciable as it reduces, since the beginning of the training phase, the length of the episodes and then the computing cost associated.

For the experiment of Figure 5 it remains 70 relevant symptoms to check, 9 possible diseases including the disease ”other”, and 20 sub-tasks have been already solved. For Figure 4 it remains 104 relevant symptoms to check, 18 possible diseases including the disease ”other”, and 103 sub-tasks have been already solved.

Finally we have been able to learn a good policy for the main task (1) where it remains 220 relevant symptoms, 82 possible diseases including the disease ”other” and all the possible sub-tasks have been already solved. Our DQN-MC-Bootstrap starts with a good policy which needs 45 questions on average to reach a terminal state and only 40 after some training iterations. On the contrary, the strategy learned by DQN-MC trying to solve this task from scratch must ask on average 117 questions to reach a terminal state and does not improve significantly during the 1000 iterations. The Breiman policy on the global task needs 89 questions on average to reach a terminal states (with a variance of 10 questions).

6 CONCLUSION

In this work, we formulated as a stochastic shortest path problem the sequential decision making task associated with the objective of building a symptom checker for the diagnosis of rare diseases.

We have studied several RL algorithms and made them operational in our very high dimensional environment. To do so, we divided the initial task into several subtasks and learned a strategy for each one. We have proven that appropriate use of intersections between subtasks can significantly accelerate the learning process. The strategies learned have proven to be much better than classic greedy strategies.

Finally, a first preliminary study was carried out internally at Necker Hospital to check the diagnostic performance of our decision support system. This experiment was conducted on a set of 40 rare disease patients from a fetopathology dataset, which has no connection to the data used to train our algorithms. We get good results; indeed more than 80% of the scenarios led to a good diagnosis. Note that theoretically our definition of the stopping rule of equation (2) makes impossible any mis-diagnosis as long as ε is chosen sufficiently low. In practice, of course, mis-diagnosis are possible because of the inevitable shortcomings of the environmental model (synonyms, omissions,...).

Research is currently underway to improve and enrich the environmental model by adding new rare diseases and symptoms. Finally, we are studying several avenues to make our decision support tool more robust in the face of the unavoidable defects of the environmental model. A larger scale study is underway but faces difficulties in obtaining clinical data.

REFERENCES

Amiranashvili, A., Dosovitskiy, A., Koltun, V., and Brox, T. (2018). Td or not td: Analyzing the role of temporal differencing in deep reinforcement learning. *arXiv*.

Besson, R. (2019). *Decision making strategy for antenatal echographic screening of foetal abnormalities using statistical learning*. Theses, Université Paris-Saclay.

Besson, R., Pennec, E. L., and Allassonnière, S. (2019). Learning from both experts and data. *Entropy*, 21, 1208.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA.

Chen, Y.-E., Tang, K.-F., Peng, Y.-S., and Chang, E. Y. (2019). Effective medical test suggestions using deep reinforcement learning. *ArXiv*, abs/1905.12916.

Cover, T. M. and Thomas, J. A. (2006). *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, New York, NY, USA.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107.

Heess, N., Silver, D., and Teh, Y. W. (2013). Actor-critic reinforcement learning with energy-based policies. In *Proceedings of the Tenth European Workshop on Reinforcement Learning*, volume 24, pages 45–58.

Köhler, S. and al. (2017). The human phenotype ontology in 2017. In *Nucleic Acids Research*.

Korf, R. E. (1985). Depth-first iterative-deepening: An optimal admissible tree search. *Artif. Intell.*, 27(1):97–109.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. A. (2013). Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602.

Peng, Y.-S., Tang, K.-F., Lin, H.-T., and Chang, E. (2018). Refuel: Exploring sparse features in deep reinforcement learning for fast disease diagnosis. In *NIPS*.

Sutton, R. S. and Barto, A. G. (2018). *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 2nd edition.

Tang, K.-F., Kao, H.-C., Chou, C.-N., and Chang, E. Y. (2016). Inquire and diagnose : Neural symptom checking ensemble using deep reinforcement learning. In *NIPS*.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256.

Zubek, V. B. and Dietterich, T. G. (2005). Integrating learning from examples into the search for diagnostic policies. *CoRR*, abs/1109.2127.