An Integrated Neural Symbolic Cognitive Agent Architecture for Training and Assessment in Simulators

H.L.H. (Leo) de Penning1, A.S. d’Avila Garcez2, Luís C. Lamb3, John-Jules C. Meyer4

1TNO Defense, Security and Safety, Kampweg 5, Soesterberg, The Netherlands, leo.depenning@tno.nl
2Department of Computing, City University, London, UK, aag@soi.city.ac.uk
3Instituto de Informática, UFRGS, Porto Alegre, Brazil, lamb@inf.ufrgs.br
4Department of Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands, jj@cs.uu.nl

Abstract
Training and assessment of complex tasks has always been a complex task in itself. Training simulators can be used for training and assessment of low-order skills. High-order skills (e.g. safe driving, leadership, tactical manoeuvring, etc.) are generally trained and assessed by human experts, due to its complex nature (i.e. many temporal relations, biased behaviour and poorly documented). This paper proposes a new cognitive agent architecture that is able to model this complex behaviour and use it for the assessment and training of both low- and high-order skills. Therefore the agent integrates learning from observation, temporal logic and probabilistic reasoning in a unified architecture that is based on Neural-Symbolic Learning and Reasoning. This so-called Neural Symbolic Cognitive Agent (NSCA) architecture combines encoding temporal logic based expert knowledge and learning new knowledge by observing experts and trainees during task execution in a simulator. The learned knowledge can be extracted in temporal logic rules for validation. Learning and reasoning is done using a Recurrent Temporal Restricted Boltzmann Machine (RTRBM). For training organizations, this provides a quicker, cost-saving and more objective evaluation of the trainee in simulation-based training. A prototype NSCA has been developed and tested as part of a three-year research project on assessment in driving simulators for training and certification, and will be tested in various other domains, such as jetfighter pilot training and strategic command and control training.

Introduction
Training and assessment of complex tasks has always been a complex task in itself. Using training simulators for this purpose simplifies the data collection on task execution and can be used for the assessment and training of low-order skills. But when it comes to high-order skills (e.g., safe driving, leadership, tactical manoeuvring, etc.) training and assessment is generally done by human experts (Van den Bosch & Riemersma, 2004). The reason is that expert behaviour on this matter is too complex to model in an automated system. There can be many temporal relations between low- and high-order aspects of a training task, behaviour is often non-deterministic and subjective (i.e. biased by personal experience and other factors like stress or fatigue) and what is known is often vaguely described and limited to explicit (i.e. “explainable”) behaviour. Still many attempts have been done with the most promising results using cognitive agents, but we are still a long way from real-world applications (Sandercock, 2004; Heuvelink, 2009).

This paper proposes a new cognitive agent architecture that is able to model this complex behaviour and use it for the assessment and training of both low- and high-order skills. Therefore the architecture integrates several modelling approaches; temporal logic to encode existing expert knowledge (Lamb et al., 2007), machine learning to learn from observation of experts and trainees during task execution (Fernlund et al., 2006), and probabilistic reasoning to deal with biased behaviour (Heuvelink, 2009).

Neural Symbolic Cognitive Agent
To model complex expert behaviour for training and assessment in simulators, the agent must be able to encode existing expert knowledge and learn new knowledge from observation. Neural-Symbolic Learning and Reasoning Systems are massively parallel computational models based on artificial neural networks that integrate inductive learning and deductive reasoning (Bader et al., 2005; d’Avila Garcez et al., 2002, 2009). In such systems, a translation algorithm maps a logical theory $T$ into a neural network $N$ such that $N$ computes the logical consequences of $T$. This provides also a learning system in the network that can be trained by examples using $T$ as background knowledge. In neural computation, induction is typically seen as the process of changing the weights of a network in ways that reflect the statistical properties of a dataset (set of examples), allowing for useful generalisations over unseen examples. In the same setting, deduction is the network computation of output values as a response to input values.
(stimuli) given a particular set of weights. Such network computations have been shown equivalent to a range of logical formalisms (Lamb et al., 2007). Based on this approach we have created a Neural Symbolic Cognitive Agent (NSCA) architecture that uses temporal logic as theory $T$ and a Restricted Boltzmann Machine (RBM) (Hinton, 2002) as neural network $N$.

**Architecture**

The NSCA architecture, depicted in figure 1, is provided with information about measured simulation data (e.g. training task, learner profile, simulation data, assessed objectives, etc.) and symbolic rules that describe expert knowledge about this data. The measured data is encoded in so-called beliefs, similar to the beliefs used in BDI agents (Bratman, 1999; Rao et al., 1995). These beliefs are described in XML elements that contain a real value and an upper and lower boundary. The value of a belief can represent the probability on the occurrence of some event or state of the world (e.g. $\text{Raining}=\text{true}$ or $\text{Raining}=\text{false}$), or a real value (e.g. $\text{AirplaneX}=45000$). Especially the latter is required for simulators to encode for example positions, orientations and other attributes of objects in the simulation. During execution of the training task the beliefs will be updated by the NSCA according to the data received from the simulator.

The symbolic rules are used by the NSCA to encode relations between the beliefs. These rules are encoded in the rules layer and can be related to previously applied rules, allowing the NSCA to model temporal relations present in a training task (e.g. procedures and behaviour). We will show that new symbolic rules can be extracted by the NSCA based on relations between beliefs learned from observation of the training task. These rules are also encoded in XML and available for validation, other agents, external systems and long-term storage (e.g. XML-file).

**Learning and Reasoning**

The neural network used in the NSCA is based on a Restricted Boltzmann Machine (RBM). A RBM is a partially connected neural network with symmetric weighted connections that implements an auto-associative memory for patterns presented at its input layer (called visible layer). It does so by training a number of hidden units to encode a probability distribution for these patterns (or parts of it), making each hidden unit an expert on part of the pattern space. Because each expert is conditionally independent from any other expert, these experts form together a high-dimensional probability distribution of the whole pattern space, which is the product of the probability distributions encoded in each individual expert. Such a network, called “Products of Experts (PoE)”, can be effectively trained by minimizing Contrastive Divergence (Hinton, 2002), which is an optimized form of Gibbs sampling implementing a Hidden Markov Chain.

![Figure 1. Global architecture of the automated performance assessment module.](image-url)
Temporal Relations
To encode temporal relations in a neural network we use a specialized form of a RBM called Recurrent Temporal Restricted Boltzmann Machine (RTRBM; Sutskever et al., 2009). The RTRBM is able to encode temporal aspects present in a pattern space as in (Lamb et al., 2007), by implementing recurrent connections to the previous state of the hidden units. These connections are trained using a combination of Contrastive Divergence and Back-propagation through time. This allows the RTRBM to relate the current state of its hidden units to all previous states. Using this recurrence in the NSCA, we can encode, learn and reason with rules that are related to the current state of the beliefs and previously applied rules.

Continuous Beliefs
As stated before a belief can represent a probability (binary value) or the value of a continuous variable. To encode both binary and continuous valued variables in the RTRBM, we use the activation function, described by Chen et al. (2003), for the visible units. This function implements a sigmoid function that can model binary as well as continuous stochastic functions. Therefore it applies a ‘noise-control’ parameter that controls the steepness of the sigmoid function. This parameter can be trained and is directly related to the data distribution of a belief. A small parameter results in a deterministic almost linear function and a large parameter results in a binary stochastic function.

The main benefit of this activation function, besides being able to handle binary as well as continuous data, is that it largely improves the RTRBM its ability to model asymmetric data. This is very useful since measured data coming from a simulator is often asymmetric (e.g. training tasks typically take place in a small region of the simulated world).

Due to the stochastic nature of the sigmoid functions used in our model, the beliefs can be regarded as fuzzy sets with a Gaussian membership function. This allows us to represent fuzzy concepts, like good and bad or fast and slow or approximations of learned values, which is especially useful when reasoning with implicit and subjective knowledge. In fact, our model can be regarded as a neural-fuzzy system similar to the fuzzy systems described in (Kosko, 1992) and (Sun, 1994).

Symbolic Rules
To express symbolic rules in terms of beliefs and previously applied rules we use the temporal propositional logic described in (Lamb et al., 2007). This logic contains several modal operators that extend classical modal logic with a notion of past and future. All these operators can be translated to a form that relates only to the immediate previous timepoint (denoted by the temporal operator ◊). This allows us to encode any rule from this language in the RTRBM as a combination of visible units (i.e. beliefs) and hidden units that represent the applied rules in the previous timepoint. For example the proposition α◊β denotes that a proposition α has been true since the occurrence of proposition β. This can be translated to the following rules: β → α◊β and α ∧ ◊(α◊β) → α◊β, where α and β are modelled by visible units, α◊β by a hidden unit and ◊(α◊β) is modelled by a recurrent connection to the same hidden unit.

To express beliefs on continuous variables we extend this logic with the use of equality and inequality formulas (e.g. A=r, A<x, etc). Note that the beliefs on binary variables can also be represented in equality formulas, A=true or A=false, which allows us to handle the outcome of these atoms in the same way as with the continuous atoms. But for readability we will use the classical notation A and ¬A.

As an example let’s take the training task depicted in Figure 2. In this task, a trainee drives on an urban road and approaches an intersection. In this scenario the trainee has to apply the yield-to-the-right-rule, which can be regarded as a training objective. Using our extended temporal logic, we can describe rules about the conditions, scenario and performance assessment related to this task. Below, ◊ denotes the modal operator “possibly true”. In temporal logic, ◊A can be read “A is true sometime in the future”. We also use ∈ to denote the modal operator “necessarily true”, or in temporal logic “always true in the future”.

Figure 2. Example training task for driving simulation.

Conditions:

\[
\begin{align*}
\text{(Area} = \text{urban}) & \quad \text{meaning: the area is urban} \\
\text{(Weather} \geq \text{good}) & \quad \text{meaning: the weather is at least good}
\end{align*}
\]

Scenario:

\[
\begin{align*}
\text{ApproachingIntersection} & \land \Diamond (\text{ApproachingTraffic} = \text{right}) & \quad \text{meaning: the car is approaching an intersection and it is possible that there is traffic approaching from the right} \\
(Speed \geq 0) & \land \text{HeadingIntersection} & \quad \text{(4.4)}
\end{align*}
\]
Assessment:

ApproachingIntersection \land (DistanceIntersection = 0) \land (ApproachingTraffic = \text{right}) \land (Speed = 0) \\
\rightarrow (Evaluation = \text{good})

meaning: if the car is approaching an intersection and arrives on the intersection when traffic is coming from the right and stops, the driver gets a good evaluation.

The temporal operator $S$, used in rule 4.4, denotes that $ApproachingIntersection$ is true when the driver has been driving towards an intersection since a certain distance $x$ to an intersection was passed. This is a typical example of a fuzzy notion that is highly subjective (i.e. the distance at which a person is regarded as approaching an intersection is dependent on the situation and personal experience). When this rule is encoded in a RTRBM, it becomes possible to learn a more objective value for $x$ based on the observed behaviour of different people in various scenarios.

Rule Encoding and Extraction

As mentioned above, we take a neural-symbolic approach whereby temporal logic rules are encoded in a RTRBM that can be trained by examples, and revised knowledge can be extracted from the trained network. To encode and extract symbolic rules in symmetric connectionist networks, like the Boltzmann Machines, Pinkas (1995) describes a generic method that directly maps these rules to the energy function of such networks. He does so, by adding a penalty to the symbolic rules creating a so-called Penalty Logic Well Formed Formula (PLOFF). This penalty can be regarded as a strength or certainty of the rule and is directly related to the weights of the connections between the rule and beliefs.

A similar approach is used in the extraction and encoding algorithms for the NSCA. Because the NSCA uses a restricted version of the Boltzmann Machines, in which hidden units are conditionally independent of each other, we can optimize these algorithms, making them much simpler. Due to this conditional independence we can treat each hidden unit as a separate rule (as an expert on some features in the belief space). This means our encoding and extraction algorithms can be applied on each rule (i.e. hidden unit) separately.

Extraction algorithm

Step 1. Determine the expertise of experts. First we need to rewrite the energy function of the RTRBM as a sum-of-products form to isolate the hidden units in terms of the states in the visible units. We do so by finding the states of the visible units ($V$) and recurrently connected hidden units ($H_{r,j}$) that lower the total energy in the energy function. These states form the local minima (stable states) in the energy function and depend on the weights encoded in the RTRBM, which define the conditional probability distribution for each hidden unit. Therefore, we can find the states by simply switching each hidden unit on ($H_{r,j} = 1$) and all others off ($H_{r,j} = 0$) and obtaining the conditional probability of each visible unit ($V^{(t)}$) and recurrent hidden unit ($H_{r,j}^{(t)}$) from the RTRBM. More formally, for each hidden unit $H_{r,j}$ we create a rule $R_i$ and for each visible unit $V^{(t)}$ a belief $B_j$, for which the conditional probability is defined by:

$$P(B_j \mid R_i) = P(V^{(t)} \mid H_{r,j}^{(t)} = 1, H_{r}^{(t)} = 0)$$

$$P(R_{r,j}^{(t)} \mid R_i) = P(H_{r,j}^{(t)} \mid H_{r}^{(t)} = 1, H_{r}^{(t)} = 0)$$

The probabilities in 5.1 are calculated using the activation function described in (Chen et al. 2003), and the probabilities in 5.2 are calculated using the binary stochastic variant of this function (by setting $\sigma=0$ and $\alpha=1$).

Now for each rule $R_i$, the value of each belief in the belief base, denoted as a vector $b_i$, is set to $P(B \mid R_j)$ (denoted by operation $\leftarrow$) and the value of each rule being applied in the previous timepoint, denoted as $r_{t-1}$ is sampled from $P(R_{r,j}^{(t-1)} \mid R_j)$ (denoted by operation $~$). Sampling means $r_{t-1}$ is set to 1 with probability $P(R_{r,j}^{(t-1)} \mid R_j)$ and 0 with probability $1 - P(R_{r,j}^{(t-1)} \mid R_j)$.

$$b_i \leftarrow P(B \mid R_j)$$

$$r_{t-1} \sim P(R_{r,j}^{(t-1)} \mid R_j)$$

Step 2. Construct a PLOFF. When we have calculated the belief and rule states that minimize the energy function of the RTRBM for each rule, we can construct a PLOFF $\Psi$ with temporal modalities using the following equations

$$\psi = \left\{ e_i, R, \leftarrow \prod_{i=1}^{k} \varphi_{i}^{(t)} \prod_{i=1}^{m} \rho_{i}^{(t)} \right\} R_i \in R$$

$$\varphi_{i}^{(t)} = \left\{ \begin{array}{ll} b_i & i \leq b_i(j) \land w_i < 0 \\ \emptyset & \land w_i = 0 \end{array} \right.$$  

$$\rho_{i}^{(t)} = \left\{ \begin{array}{ll} \cdot R_{r}^{(t)} & r_{t-1}^{(t)} = 1 \\ \cdot R_{r}^{(t)} & r_{t-1}^{(t)} = 0 \end{array} \right.$$  

Using the transformation algorithms described in (Lamb et al., 2007) we can simplify the PLOFF rules further with other temporal modalities.

The penalty $r_{t}$ of each rule is calculated similarly to the algorithm described in section 3.4 of (Pinkas, 1995) and represents the conditional probability of the rule, given a certain state of beliefs $b_i$ and previous applied rules $r_{t-1}$ (see 5.8).

Since the beliefs of the NSCA represent continuous variables, propositions on beliefs, denoted by $\varphi_{i}^{(t)}$, are
represented by inequality formulas on their values. A proposition $\phi_{it}^{(0)}$ depends on the weight $w_{it}$ of the connection between the hidden unit $H^{(i)}$ that represents rule $R_i$ and visible unit $V^{(j)}$ that represents belief $B_j$. From the activation function that calculates the probability of rule $R_i$, we can see that a negative weight will increase the probability when we decrease the value of belief $B_i$. So all values for belief $B_i$ less or equal to $b_{ij}(j)$ will increase the probability of rule $R_i$. The inverse applies to a positive weight. When the weight is exactly 0 a belief has no influence on the rule probability and can therefore be omitted from the rule.

**Encoding algorithm**

**Step 1. Create a PLOFF.** Create a PLOFF from a temporal logic theory:
1. Rewrite each rule in the logical theory to its conjunctive normal form using a reduced set of connectives {\land, \neg}.
2. Rewrite each temporal proposition in the rules to a form that only uses the temporal operator $\bullet$ according to the transformation algorithms in (Lamb et al., 2007).
3. Construct for each rule an equivalent PLOFF rule as defined in equation 5.5.
4. For each rule, rewrite the temporal propositions to an equality formula (i.e. $\bullet R \rightarrow R^{t-1} = 1, \bullet \neg R \rightarrow R^{t-1} = 0$).

**Step 2. Define the Experts.** Now for each rule $R_i$ in the PLOFF define the hidden and visible units in the RTRBM:
1. Add a hidden unit $H^{(i)}$ to represent rule $R_i$.
2. Add a visible unit $V^{(j)}$ for each atomic proposition related to the current timepoint to represent the beliefs in the rule’s proposition.
3. Randomize the weights connecting $V^{(j)}$ with $H^{(i)}$.

**Step 3. Train the Experts.** Now for each rule $R_i$ in the PLOFF calculate the weights that maximize the likelihood of the beliefs and previously applied rules in the rule’s proposition:
1. Calculate for all hidden units $H^{(i)}$ the probability using the values for the beliefs $B_j$ and previously applied rules $R^{t-1}$ defined in rule $R_i$ using equation 5.8.
2. Set the value of the hidden unit $H^{(i)}$ that represents rule $R_i$ to 1.
3. Using the new state of the hidden units, calculate the values of the beliefs ($=b_i$) and previous applied rules ($=r^{t-1}$) using equations 5.3 and 5.4.
4. Minimize the difference (i.e. contrastive divergence) between the calculated values for $b_i$ and $r^{t-1}$, and the values of beliefs $B_j$ and previous applied rules $R^{t-1}_j$ defined in rule $R_i$ using the learning algorithm described in (Chen et al. 2003).

**Initial Results**

A prototype NSCA has been developed as part of a three year research project on assessment in driving simulators, carried out by TNO in cooperation with the Dutch licensing authority (CBR), Research Centre for Examination and Certification (RCEC), Rozendom Technologies, and ANWB driving schools. The prototype is implemented using a Virtual Instruction platform, called SimSCORM (Penning et al., 2008).

The prototype is currently being tested in our lab and initial results show that the prototype is able to learn relations from real-time observation and extract valid symbolic rules in XML format. For example, rules have been learned that relate the RPM of a car to the selected gear and the mission success of a F16 to the tanks it destroyed.

More experiments have been planned for this year on the driving simulators used in the drivers training curriculum of the ANWB (see Figure 3). This allows the NSCA to be validated in an operational setting with many scenarios, a large trainee population and multiple driving simulators. If successful, the NSCA will be used to support the Dutch driver training and examination program. In parallel, the module will also be tested in other simulation domains, like jetfighter pilot training and for strategic command and control training.

**Conclusions and Future Work**

The Neural Symbolic Cognitive Agent (NSCA) architecture presented in this paper offers an approach that integrates symbolic reasoning and neural learning in a unified model. This approach allows the agent to learn complex expert behaviour for training and assessment from observation of human experts and trainees during task execution in a simulator. Learned behaviour can be extracted in symbolic rules for validation, reporting and feedback. Furthermore the approach allows existing knowledge to be encoded in the model.

A prototype NSCA has been built and is currently being tested at TNO. Initial results look promising and the feasibility of the presented approach will be validated further by the use of real-world training simulators and scenarios in multiple domains.

**Figure 3. ANWB Driving Simulators**
Abstract Concepts

In the NSCA, assessment of high-order skills can now be related to the assessment of low-order skills and simulation data by training the connections of the RTRBM based on observed data from the simulator. This is possible due to the auto-associative nature of the RTRBM. But sometimes it is not known which high-level concepts exist in a training scenario. One may want to use the RTRBM to find these abstract higher-level concepts. This can be done with so-called Deep Boltzmann Machines (Salakhutdinov et al., 2009), which are essentially a hierarchy of Boltzmann machines. Similar deep networks can be implemented using the RTRBM in a hierarchy. It remains to be investigated how such high-level concepts can be learned and extracted.

Adaptive Training

Since the NSCA is capable of learning the relations between trainee competences, constructs being measured in the simulation and assessments and instructions given during the training, the NSCA can be used for adaptive training as well. This means that the results of the NSCA can be used to adapt the training scenario and/or instructions to make the training task easier or more challenging. This is made possible by the auto-associative nature of the RTRBM used by the agent, which allows the learned rules to be applied backwards, in generative mode, for example, by calculating the desired state of the visible and recurrent hidden units according to the desired competences for the trainee. This desired state then becomes an intention for the agent to fulfill in the simulator.

References


