Cañamero, L., Oudeyer, P.-Y., and Balkenius, C. (Eds) Proceedings of the Ninth International Conference on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems. Lund University Cognitive Studies, 146.

Applying the Schema Mechanism in Continuous Domains

Frank Guerin
Department of Computing Science
University of Aberdeen
Aberdeen, AB24 3UE, Scotland
f.guerin@abdn.ac.uk

Andrew Starkey
School of Engineering
University of Aberdeen
Aberdeen, AB24 3UE, Scotland
a.starkey@abdn.ac.uk

Abstract

We are interested in developing a computational model of Piaget's theory of sensorimotor intelligence. Existing works in this area have demonstrated mechanisms which acquire Piagetian schemas, however the sensory inputs to these systems are typically constrained to a small number of discrete values. In order to model Piagetian developments such as the acquisition of skills it will be necessary to handle continuous (or real) domains of sensor values, and to learn skills which are guided by feedback from these real valued sensors. We extend existing Piagetian work by employing a neural network function approximator, to represent a reinforcement learning value function over a real valued sensor space. Using this combination of techniques allows our system to learn skilled actions which can then be treated as Piagetian schemas, and combined with other schemas. Our experiments in a simple simulated world show that this novel combination is feasible in principle; future work will need to test the approach in more challenging domains to determine its limitations, and to improve on it.

1. Introduction

We are interested in building AI systems which can learn their own world knowledge autonomously, and exhibit ongoing development (sometimes called "ongoing emergence" (Prince et al., 2005)). This idea comes from trying to copy the biological approach to the development of world knowledge, in particular human cognitive development during infancy. Piaget's theory of constructivism gives an account of how humans build up their world knowledge through their interactions with the environment (Piaget, 1936). Piaget's theory is a grand overview of the human learning mechanism, but unfortunately it does not give the level of detail which would be necessary to inform a computer implementation. To make progress in computational models of this theory a number of AI works have carried out computational investigations on small parts of the theory (see Section 2). These works have demonstrated the possibility of a learning mechanism which acquires Piagetian schemas through trial and error, and can build on this knowledge through techniques such as chaining schemas.

Building on existing Piagetian AI work, we have set ourselves a long-term target to build a computational model of Piagetian means-end behaviours; this is the fourth of Piaget's six sensorimotor stages, and commences at about eight months of age. In order to model this fourth stage our AI system needs to be able to acquire *means* actions. These are skilled motor actions, such as grabbing a seen object, or hitting an object to make it swing, or scratching an object, etc. This gives us a short term target to be able to build a system which can acquire these skilled actions and use them within a Piagetian learning framework. We find that the existing Piagetian AI work is inadequate for skill acquisition, as existing work tends to use sensory inputs which are constrained to a small number of discrete values. For the acquisition of skills it will be necessary to handle continuous (or real) valued sensors, and to learn skills which are guided by feedback from these real valued sensors. To address this we use the technique of Neural Fitted Q Iteration (Riedmiller, 2005). This is a reinforcement learning method, which employs a neural network function approximator to represent a value function over a real valued sensor space. This technique makes use of a set of *transition experiences* (in our case these are small arm movements taken at various different positions), and generalises to find a value function over the space. A particular strength of the approach is that the same transition experiences can potentially be used in training for different goals. In order to identify the sensor variables which are relevant to the acquisition of a particular skill, we used a simple statistical analysis; this then allows the neural network to ignore irrelevant sensors, thus there are fewer inputs to the neural network.

Using this combination of techniques allows our system to learn skilled actions which can then be treated as

Piagetian schemas, and combined with other schemas. The overarching Piagetian framework we use is the Constructivist Learning Architecture (CLA) (Chaput, 2004). This architecture finds reliable schemas, and also allows higher layers of learning where composite actions can be treated as atomic in order to find further schemas.

Our experiments in a simple two-dimensional simulated world show that this novel combination is feasible in principle. Our system learnt skills to bring the hand to the mouth and to bring the hand to a seen object. Composite actions chained these to bring a seen object to the mouth.

In Section 2 we cover the background work which we are building on. Section 3 describes our simulated world. Section 4 details the learning techniques and how we have applied them. Section 5 gives results of our experimentation. Section 6 concludes with a discussion and some future directions.

2. Background

This section looks at some of the existing work in the computational modelling of Piagetian Schemas in the sensorimotor stage, and then gives the necessary background on the techniques which we will make use of.

2.1 Piagetian Schemas in AI

There exist a number of AI works which are inspired by Piagetian schemas. Drescher's "schema mechanism" was the first of these (Drescher, 1991). Drescher simulated a baby, with a hand, eye, and mouth, in a 7x7 grid world. The world also contained some objects which the baby could grab. Drescher's schemas were 3-part structures consisting of a context, action, and result. A schema is a prediction about the world: if its action is taken in the context specified, then the result is predicted. For example one schema which the program learnt is that if its current context was "HandInFrontOfMouth", and it took the action "HandBackwards", then it would expect to obtain the result "HandTouchingMouth". The learning mechanism was also capable of chaining together a number of schemas as a *composite action* in order to achieve a goal. Drescher's mechanism was criticised for its efficiency, and improved on by Chaput (2004), as described in the next subsection. Nevertheless, Drescher's work has been influential, with many subsequent works following a similar pattern. The learning mechanism incorporates many of the elements we would want in a Piagetian framework: schemas are acquired when a context/action/result triple occurs reliably, schemas are then stored in a library, and can be activated, and furthermore schemas can be chained up to make composite actions to achieve goals. The shortcoming for our purposes is the lack of a method to generalise over real valued inputs, i.e. if the context consisted of real sensor values.

In the Petitagé architecture of Stojanov (2001) the agent learns "expectancies" of the form ⟨*Sensor_state*, *action*, *Sensor_state*⟩, which are similar to Drescher's context/action/result triple. This was applied to learning the structure of a maze with walls, and the agent built a partial map of its world. The architecture is not specific about the types of sensors required and allows for the possibility of real-valued sensors, however the issue of generalising over real values of sensors has not been tackled.

A further work by Perotto and Álvares (2006) has a tripartite schema structure consisting of: context, action and expectation (just like Drescher's). The learner has the ability to generalise contexts that achieve the same result. Contexts are represented by binary strings, and the generalisation converts a '1' and '0' in the same position in two contexts to produce a '#' value, which is a wildcard and could match '1' or '0'. Although the work deals with binary strings, these strings could be used to represent real sensors (i.e. binary representation of the real number). This type of generalisation could learn tiled regions where contexts are similar, but would not be able to find regions of other shapes. We require a superior generalising ability for our purposes.

A completely different approach to schema learning appears in Hart et al. (2008) and is worth mentioning here. In this work closed-loop feedback control programs are used for basic sensorimotor actions (such as reaching and touching), and then these appear as discrete actions within a reinforcement learning framework which can learn higher level behaviours. This work handles continuous domains very well and is a feasible approach to robotic manipulation problems where there are many degrees of freedom. In such domains it might not be feasible to have a pure reinforcement learning approach exploring the whole space, so an a priori model of controller performance may be essential.

2.2 Constructivist Learning Architecture

The first three works above reach some sort of consensus on the idea of schemas being context/action/result triples. The work of Chaput (2004) goes further by incorporating this idea with a neo-piagetian learning theory, to make a new architecture for developmental learning. Chaput developed a "Constructivist Learning Architecture" (CLA) which is based on Leslie Cohen's theory of infant cognitive development Cohen (1998). This theory essentially states that infants learn to process information at increasingly higher levels of abstraction by forming higher level units out of relationships among lower level units. There is a bias to process information using the highest formed units, unless the input becomes too complex, in which case the infant drops back to a lower level and attempts to refine its abstraction so as to be

able to handle the complex information at the higher level. The CLA is very much a Piagetian learning architecture, in which schemas (similar to Drescher's) are learnt at each level. The architecture is quite generic; the higher levels can be formed by integrating multimodal information from lower layers, or integrating time delayed versions of sensory input. As one of his experiments with the architecture, Chaput recreated the achievements of Drescher, in a more efficient way.

Chaput's computational model (CLA) is based on Self Organising Maps (SOMs) which are built hierarchically (modelling the different levels of Cohen's theory). Chaput uses SOMs as way for different representations to compete for the finite available space. As in Drescher's work, the learning agent records context/action/result triples. A SOM is created for each possible action of the agent. The SOM is trained on vectors which represent the context and result of taking that action a number of times. The SOM thus finds reliable patterns of context/action/result (i.e. these come out with a strong representation in the SOM). Furthermore, the CLA learns schemas at a certain level first until it has some stable knowledge, before moving on to consider learning on the next level. Once the CLA has moved to a higher level, those schemas from the lower level are not updated anymore; learning on that level has frozen. Thus the learning resources can focus on one level at a time.

Chaput's work did not address generalisation over real sensor values however. His robot forager example had no need for this as it only had seven binary sensors. For our work we will make use of the CLA as it is quite generic, and generally suits our purpose, however we will combine it with a generalisation technique to be described next.

2.3 Riedmiller's Neural Fitted Q Iteration

In order to deal with real valued sensors we borrow a technique which uses a neural network for function approximation in reinforcement learning. Riedmiller's Neural Fitted Q Iteration (NFQ) (Riedmiller, 2005) is a method which trains on a set of transition experiences, each of which has the form $\langle s, a, s' \rangle$, state, action, resulting state; these triples are similar to Drescher's context/action/result. In addition each triple has a reward value (in practice most of the triples will typically have zero reward, and just a small proportion have a large reward). The system learns a Q value which represents the discounted expected reward to be obtained by executing a specific action in a given context. Thus the Q value function takes as input a context and action, and outputs a real value (the Q value). The Q value function is represented by a neural network. The training iterates, performing two main steps in each iteration, the first step is to do a sweep through all the transition experiences $\langle s, a, s' \rangle$, finding a new ideal value for Q, for the given context s and action a. This uses the following equation:

$$Q_{k+1}(s,a) := (1-\alpha)Q_k(s,a) + \alpha(R(s,a) + \gamma \max_b Q_k(s',b))$$

where $Q_{k+1}(s,a)$ is the new ideal value for Q, given context s and action a; α is the learning rate; $Q_k(s,a)$ is the old value for Q (from the neural network, before the update); $R(s,a)$ is the reward which was obtained for executing action a in context s; γ is the discount; $\max_b Q_k(s',b)$ is the Q value of the best action b from the resulting state s'. The second step is to use these new ideal values to update the Q value function. This update is done using the RPROP neural network training algorithm (Riedmiller and Braun, 1993). After a number of iterations we have a Q value function which can recommend to us the best action to take in any given context, in order to maximise reward. To do this we can simply query the Q value for every action from the given context; the highest value is the best action.

The generalisation performed by the neural network means that the coverage of the sensor space by the transition experiences can be relatively sparse; when the Q learning is updating a Q value by looking ahead to the expected reward to be obtained by taking a particular transition, this transition does not need to connect directly to another, as the neural network will have interpolated between the data points given. Similarly, the final Q function can be queried for a context which is not present in any transition experience. The key innovation of NFQ is the fact that it maintains a record of all transition experiences, whereas other approaches use a transition experience to train online and then discard it. This gives NFQ greater stability, as its early learning will not be damaged or undone by later learning. A further added advantage is that transition experiences are not tied to a particular reward; transition experiences merely record how an action moves the agent from one state to another. This means that the same experiences could be used in the training for multiple different goals.

3. The Simulation

Our simulated infant lives in a simple 2D world with a single rigid square block which can be grabbed and moved. A rigid body physics engine simulates the physics of the world, including friction and collisions between blocks; gravity has been disabled.

The infant (see Figure 1) has a single movable arm, consisting of two rigid rectangular blocks: an upper arm and a lower arm. The upper arm can rotate at the shoulder, and the lower arm rotates from the elbow. There is a hand at the end of the lower arm (represented by a square). When the hand overlaps with a block, a touching sensation is generated, and if the grab action is then taken, the block will then move together with the hand, until released.

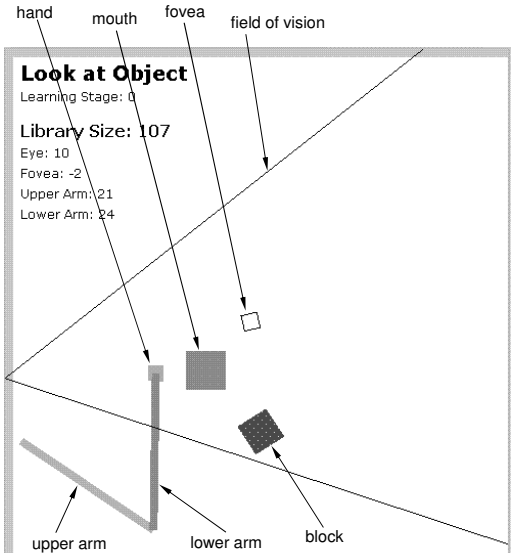

Figure 1: The simulation.

The infant's mouth is the square at the centre of the figure. Note that the mouth position had originally been at the eye, but this made the learning of hand to mouth too easy, as it was nearly always optimal to bend the elbow more, from whatever position, in order to reach the mouth. The mouth position was moved to the middle to make the learning problem more interesting.

The infant also has a field of vision, bounded by the two lines emanating from the left wall. The point of intersection of the two lines is the "eye". The field of vision can be rotated about the eye. As in Drescher's simulation, the centre of visual attention has a fovea; in our system the fovea is capable of moving along a path between the two eye lines (and equidistant from them). The fovea is shown as an outlined box in the figure.

The infant has a number of sensors as shown in table 1. The last two sensors are part of the *seen_objects* sensor, which returns a set of objects that are in view (i.e., between the lines bounding the field of vision); each object in the set is described by a pair: angular displacement from centre of visual field, distance from fovea. The "distance from fovea" is the object's distance from the arc on which the fovea would move if rotated, or alternatively, the distance from the fovea if the object were on the centre line. In the experiments we ran for this paper there were only two objects to be seen: a block, or the infant's own hand, and in fact only the hand was actually used as an input for learning. Note that there is no occlusion: objects which are behind others are still returned in the *seen_objects* list.

Type	Sensor
float	*lower_arm_angle*
float	*upper_arm_angle*
float	*eye_angle*
float	*eye_fovea_distance*
boolean	*fovea_sees*
boolean	*mouth_touched*
boolean	*mouth_touched_object*
boolean	*hand_touched*
boolean	*hand_holding_object*
float	*seen_objects_distance*
float	*seen_objects_angle*

Table 1: Infant's sensors

Table 2 shows the nine actions which the infant is capable of. As can be seen from the table, we distinguish continuous and discrete actions. Continuous here means that the action's outcome is dependent on continuous (real valued) sensors, and it affects those sensors; i.e. the action is moving something in a continuous space. Discrete means that the action's outcome depends only on Boolean sensors, and it affects only those; i.e. the action is setting some binary item in the world. There are four continuous arm actions: the upper arm and lower arm can both (independently) move up and down. There are two actions for the eye. The discrete action LOOK_AT_OBJECT is really a composite action which we have coded in innately rather than getting the system to learn it; this action moves the fovea to focus on the object, if the object is visible in the field of view (otherwise the action does nothing). The discrete action HAND_GRAB fixes the position of the object relative to the hand, if the hand is touching the object (otherwise the action does nothing). The action FIXATE fixes the position of the fovea, disabling eye movements for a time interval (this action is only effective if an object is in the fovea). Note that these last two discrete actions have been programmed in as reflexes so that they are automatically triggered when a sensor is activated: HAND_GRAB happens whenever *hand_touched* is on, and FIXATE happens whenever *fovea_sees* is on. The idea of FIXATE is that it forces the infant to spend a long time with the fovea focussed on the block, during which time other hand movement actions can be taken; this allows the program to learn how its arm movements

Type	Action
continuous	UPARM_UP
continuous	UPARM_DOWN
continuous	LOWERARM_UP
continuous	LOWERARM_DOWN
continuous	EYE_UP
continuous	EYE_DOWN
discrete	LOOK_AT_OBJECT
discrete	HAND_GRAB
discrete	FIXATE

Table 2: Infant's actions

can affect the position of the hand, relative to the block in the centre of vision (and so potentially learn how to move the hand closer to the fovea and contact the block). There is no action for the hand to release what is grabbed; a grabbed object is simply released after a random time interval.

4. The Learning Mechanism

Our learning mechanism treats discrete actions and continuous actions differently. Discrete actions are used to learn context/action/result schemas, just like Chaput's (see Section 2). These schemas reflect the most reliable context and result of the action when it was taken many times. On the other hand, when continuous actions are taken, we record transition experiences (following Riedmiller); although these are similar to the what many related works call schemas, we will reserve the term "schema" for higher level knowledge, generalised from many transition experiences. Thus from a set of transition experiences, we learn a Q value function, which means we have a policy for achieving a goal. We then treat this policy as a schema, and it becomes a new (composite) discrete action which can be taken, and context and result recorded for higher level learning.

Note that by making this distinction, and using different learning methods for discrete and continuous actions, we are effectively giving the infant some innate knowledge: i.e. the infant innately knows that the continuous actions' effects depend on the real valued sensors (but it does not know which sensors exactly).

4.1 Learning For Discrete Actions

Following Chaput (2004) we record vectors of the values of all sensors before (context) and after (result) each action is taken. For the discrete actions we discard all the real-valued sensors. This gives us a vector of ten binary digits for each time an action has been taken (the five discrete sensors before and after, corresponding to context and result). We now change the five result values to represent the change that happened as a result of the action. This is computed by subtracting the context values from the result values. Therefore, if a discrete sensor changed from 1 to 0 then it will have a change of -1 whereas if it changed from 0 to 1 the change will be 1, otherwise it will be 0. An illustrative example follows:

Initial context/result vector: [1 0 0 1 0 / 0 1 0 1 0]

After change is computed: [1 0 0 1 0 / -1 1 0 0 0]

For each discrete action we create a 10x10 Self Organising Map (SOM) and train it with all the vectors for that action (we had approximately 1000 vectors recorded for each action). We then perform thresholding on the weights of the resulting SOMs. The threshold used was 0.9. This means that, for context values, any weight greater than 0.9 becomes 1, any weight less than 0.1 becomes -1, and any value in between becomes 0; for result values, any weight greater than 0.9 becomes 1, any weight less than -0.9 becomes -1, and any value in between becomes 0. After this any weight vector which has at least one positive result becomes a schema (unless it already exists as a schema). The SOM method is averaging, so vectors which consistently have ones (or zeros) in certain positions in the context and result will be harvested; wherever there is a mix of ones and zeros in a position, an intermediate value will result, which will not make it above the threshold. It is important that this applies to both context and result, because a result which occurs regularly would be useless without a context which says when it reliably occurs. This process is known as *harvesting* schemas, and follows Chaput (2004) exactly. A schema states that if its context is matched, and the action is taken, then the result can be expected to be achieved. Note that values of 0 in the context mean "don't care" whereas 1 means "must be 1" and -1 means "must be 0".

In our experiments this procedure proved to be good at identifying results that could be reliably achieved, but it found contexts which were overly specific. For example, the following are the schemas resulting from the HAND_GRAB action.

```
-1 -1 -1  0 -1    0  0  0  0  1   218
-1 -1 -1  1 -1    0  0  0  0  1   214
-1  0 -1  0 -1    0  0  0  0  1   296
-1  1 -1  1 -1    0  0  0  0  1    77
-1  0 -1  1 -1    0  0  0  0  1   291
 0  0 -1  1 -1    0  0  0  0  1   454
 0  1 -1  1 -1    0  0  0  0  1   117
 1  1 -1  1 -1    0  0  0  0  1    40
 1  0 -1  1 -1    0  0  0  0  1   163
 1  0 -1  0 -1    0  0  0  0  1   173
 1 -1 -1  1 -1    0  0  0  0  1   123
```

Each row is a schema with five context elements and five result elements. The final column gives the number of recorded vectors which support this schema. The SOM method of harvesting does not attempt to generalise over contexts; each instance that has a sufficiently large number of supporting vectors will get its own representation in the SOM. We also tried constraining the number of schemas produced by using a smaller SOM; we found a cut-off below which the SOM produced no schema, and above which it produced too many. Ideally we would like to get one schema for the above examples. To resolve this we used the 10x10 SOM and added a generalising step which simply groups together all schemas with the same result, and replaces any context items which take multiple values with a zero. This leads to the context [0 0 -1 0 -1] in the above example.

4.2 Learning For Continuous Actions

We tried to use the same harvesting approach with our continuous actions. We explain the idea behind this with an example: one might imagine that when the mouth is

touched, the arm sensors will tend to have a constrained range of values, and that this might be represented by a cell in the SOM. However, we found that when real values are in the context, the discrete result elements (such as *mouth_touched*) take on intermediate weight values in the SOM, which do not make it past the thresholding phase. Eventually we abandoned the SOM approach for continuous actions, and used a simple statistical method. Essentially we want to find which sensors are relevant to achieving a particular result. For example, if we want to learn to touch the mouth from any position, then we should only train our neural network with the two arm angles (and ignore all other sensors).

We will now describe the statistical method for one action (the same procedure is repeated for all actions). We record approximately 1000 vectors for each continuous action. The context of our vectors includes all eleven sensors (discrete and continuous). The result part of the vector computes the change in discrete sensors, as done in the discrete action case, and contains five elements. We want to find which sensors are likely to be important in the achievement of each result. We perform the analysis for each of the five results separately. For each result we split our 1000 vectors into two sets. The first set contains the context vectors when that result was zero, and the second set contains the context vectors when the result was (positive) one. To make this concrete consider the result "mouth_touched" and one of the arm movement actions; we create a set of all context vectors where the action led to the mouth being touched (this set turned out to have 54 vectors in our example), and another (larger) set of all context vectors where the action did not lead to the mouth being touched (this set turned out to have 992 vectors in our example). For each of the eleven sensors we now compute an ANOVA, comparing the distribution of values for that sensor both when the action achieves the result, and when it does not. For our example with the result "mouth_touched", we had the following probabilities of the "null hypothesis" for the eleven sensors:

$6.7\times10^{-7}, 0, 0.37, 0.51, 0.90, 0.40, 1.0, 1.0, 1.0, 0.50, 0.56$

This very clearly shows that the first two sensors (corresponding to arm angles) are the only ones relevant to the result "mouth_touched".

In general we follow this same process for all results. We discard any result elements where the number vectors in the smallest of the two sets (for zero or one) is less than two percent of the size of the original data. This corresponds to a result which is very rarely achieved by this action. After this we are left with results which are frequently achieved by the action, and we need to find which sensors are relevant to achieving the result. We set a threshold of two percent on the probability of the "null hypothesis"; i.e. we only pick up on sensors whose probability of being affected by the result (of 1 or 0) is less than 0.02. This gives us a list of relevant sensors for each result which can be achieved by this action. This is repeated for all actions.

Now for each result we have a list of all those sensors affecting the result (if any), and the action that caused it. Any result that has a non-empty list of actions and sensors is used to create a new neural network, to represent a Q value function for achieving that result. The inputs to the network are the sensors and the actions. Continuous sensors give real valued inputs to the network, discrete sensors as well as actions give binary inputs to the network. This network is now trained by the NFQ algorithm, using all the transition experiences as training. Any transition experiences that achieve the result are given a maximal reward value; all other transition experiences have zero reward.

4.3 Learning For Composite Actions

After having learnt schemas for discrete actions, and policies for continuous actions, we form composite actions for any new result. This follows Drescher (1991) (and Chaput). Composite actions are formed by finding a chain of actions backwards from a result, which lead to the result being achieved, from a variety of different contexts. Thus the composite action has a set of contexts where it can be invoked, and a single result which it will achieve. The policies learnt above for continuous actions are also treated as composite actions in their own right.

For the next phase of learning we drop the six continuous actions of Table 2 above, and extend the set of actions with the new composite actions. This next phase of learning progresses using the method of harvesting of schemas which was used for the discrete actions. This can find correlations among the composite actions. In particular it can find new results which may be reliably achieved by the policies for continuous actions, because these composite actions are now taken frequently, and in sequences. These new results would be very unlikely to be achieved before the policies were learnt, because there was little chance that one result would be achieved after another by the random operation of individual continuous actions.

This completes our higher level of learning. As with Chaput's work, in principle there is no reason why we cannot have progressively higher layers on top of this, but we have not explored this yet.

5. Results

To gather training data we ran our simulation to gather 7000 transition experiences, where actions were randomly taken. We randomly repositioned the hand after each movement. This is simply for expedience in gathering training data; it covers most of the space in less time. A random walk would be more faithful to a real

infant's experience, but it needs to run for longer in order to cover most regions of the space. From this data we harvested schemas for discrete actions, resulting in two schemas:

```
-1  0 -1 -1 -1   1  0  0  0  0
 0  0 -1  0 -1   0  0  0  0  1
```

The first is for the LOOK_AT_OBJECT action and states that this action results in the *fovea_sees* sensor turning on. The second is for the HAND_GRAB action and states that it results in the *hand_holding_object* sensor turning on.

For the continuous actions we found actions to achieve three results. Firstly the result of *mouth_touched* was found to be caused by the four arm movement actions, with the sensors *lower_arm_angle* and *upper_arm_angle* being relevant. From this a Neural network with six inputs was created (i.e. four actions and two sensors) to represent the Q value function and learn a policy to achieve *mouth_touched*. Secondly the result of *hand_touched* was found to be caused by the four arm movement actions, with the sensors *seen_objects_distance* and *seen_objects_angle* and *mouth_touched* being relevant. The *mouth_touched* sensor is spurious here; it should not be relevant to achieving the goal, however it is probable that the fact that the object is often located close to the mouth causes the relationship to be inferred. From this a neural network was created to represent the Q value function and learn a policy to achieve *hand_touched*. Finally, the result of *fovea_sees* was found to be caused by the two eye movement actions, with the sensors *eye_angle* and *upper_arm_angle* being relevant. The *upper_arm_angle* sensor is spurious here. We did not train a network for this as it was deemed to be too simplistic, given the limited space in which the eye can rotate.

We trained the networks for the first two results using 1016 transition experiences (the same experiences were used in training for the two different results). This is much less data than had been used for the initial analysis; this reduction was simply to make the learning iterations faster; the data proved to be sufficient. We used 100 RPROP iterations for each training phase, followed by a Q-learning sweep, and this whole process was iterated approximately fifteen times to produce a reasonable result. A reasonably effective policy was learnt both for achieving *mouth_touched* and *hand_touched*. The policies were successful in achieving their result about 50% of the time, sometimes getting stuck in a back and forward loop. We suspect that if there were more than four possible actions it would likely lead to smoother and more reliable policies (although more training data and time would be required).

Given the simplicity of our scenario we were able to compile the composite actions manually. This gave us actions that could achieve *fovea_sees* by looking at an object, and which could achieve *hand_holding_object* by moving the hand (using the policy learnt above) to the block in the fovea, and performing HAND_GRAB. The final harvesting of schemas using these composite actions was able to find the schema to achieve the result *mouth_touched_object* (i.e. to grab an object and take it to the mouth). Admittedly this result is a little contrived as the simulation has been designed just to make this possible, however it does show that the techniques combined here can work in principle.

6. Conclusion

This work has brought together two techniques in an effort to model sensorimotor skill acquisition, with a view to modelling Piagetian sensorimotor developments. We will first briefly evaluate the effectiveness of these two techniques, and then the combination. Firstly, Riedmiller's NFQ works very well in our setting. It is particularly convenient that the same training experiences can be used to train for different goals. We can also illustrate its strength by comparing it with a naive reinforcement learning approach to our simulation; for example in learning to move the hand to touch an object in the fovea, a naive approach would require touching the object on each trial to propagate reward back to the actions that led there. Furthermore, the object would need to be in the fovea all the time. In contrast the NFQ approach can use any transition experience of a hand movement as part of its training data; the experiences merely describe the resulting state, and are separated from any particular goal. There are issues over biological plausibility however as this is hardly a realistic model of what an infant does; instead it is likely that memories are abstracted/generalised in some way rather than being stored precisely.

Secondly, Chaput's method for harvesting schemas works reasonably well, apart from the issue of generalising over contexts, mentioned in Section 4 above. Despite the success of this method, we suspect that a simpler statistical averaging technique may obtain the same results in a more efficient way. The clustering ability of the SOM has not been exploited by Chaput, and we suspect that the basic idea of a hierarchy of SOMS could lead to a much more powerful learning approach if used in a different way. This is an interesting area for future work.

The combination of techniques we have used works for our simple examples but needs to be tested on a wider range of example scenarios. There will be challenges to be overcome when the space is larger, for example if sensors include more inputs from vision. The training data required will become inordinately large, and it is likely that there will need to be some gradual way to build up abstractions on sensor data, so that the learning is constrained initially, either by using some innate abstractions, or the lifting of constraints (Lee et al., 2007). Fur-

thermore, the efficiency of the learner could be improved considerably, in a number of ways. One obvious example is by using online learning, where actions are taken using a greedy strategy, which would mean that less of the state space would need to be recorded. This inefficiency does not concern us for the moment because we are currently unsure how a (relatively generic) learning system could be built to learn high level knowledge from its experiences. Once this question has been answered (even with an inefficient mechanism) we can then investigate optimisations.

Finally, let us look at this as a model of Piaget's theory (which is our long-term goal), the work has achieved what it set out to do, in allowing continuous actions to be integrated with a schema learning mechanism, however, many aspects remain to be addressed to create a convincing model of Piagetian learning. To compare with Chaput's (2004) work, we note that we have not implemented synthetic items. Synthetic items seem particularly useful when there are hidden aspects of the world, for example: (1) objects which have an existence in the world, but are not always observable (Drescher's original motivation for introducing the synthetic item); (2) positions which are not accurately observable, as in Chaput's robot forager world; (3) hidden properties such as the weight of an object (Morrison et al., 2001). These "hidden aspects" do not feature in our simulation for the moment, but we expect that they will play a part in future work when we want to model later stages of development. Nevertheless, we can claim a strong similarity with these works because of the hierarchical nature of the learning; the synthetic item is used to notice a correlation between the activation and success of schemas over time, which is also a function performed by our higher layer of learning.

The next immediate step for this work would be to allow the policies learnt on continuous actions to be adjusted to achieve new goals. We have frozen the policies once they are learnt, but to model Piagetian assimilation and accommodation in stage 4 means-end behaviours, there should be the possibility to adjust a policy when a new, slightly different, goal needs to be achieved.

Acknowledgements

We are grateful to Joey Lam, Bang Wu, and Cory Lowson who worked on various parts of the software.

References

Chaput, H. H. (2004). *The constructivist learning architecture: a model of cognitive development for robust autonomous robots*. PhD thesis, AI Laboratory, The University of Texas at Austin. Supervisors: Kuipers and Miikkulainen.

Cohen, L. B. (1998). An information-processing approach to infant perception and cognition. In Simion, F. and Butterworth, G., (Eds.), *The Development of Sensory, Motor, and Cognitive Capacities in Early Infancy*, pages 277–300. East Sussex: Psychology Press.

Drescher, G. L. (1991). *Made-Up Minds, A Constructivist Approach to Artificial Intelligence*. MIT Press.

Hart, S., Sen, S., and Grupen, R. (2008). Generalization and transfer in robot control. In *Proceedings of the Eighth International Conference on Epigenetic Robotics, University of Susseex, July 30-31.*

Lee, M. H., Meng, Q., and Chao, F. (2007). Staged competence learning in developmental robotics. *Adaptive Behavior - Animals, Animats, Software Agents, Robots, Adaptive Systems*, 15(3):241–255.

Morrison, C. T., Oates, T., and King, G. (2001). Grounding the unobservable in the observable: The role and representation of hidden state in concept formation and refinement. In *In AAAI Spring Symposium on Learning Grounded Representations*, pages 45–49. AAAI Press.

Perotto, F. S. and Álvares, L. O. (2006). Learning regularities with a constructivist agent. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 807–809, New York, NY, USA. ACM.

Piaget, J. (1936). *The Origins of Intelligence in Children*. London: Routledge & Kegan Paul. (French version published in 1936, translation by Margaret Cook published 1952).

Prince, C., Helder, N., and Hollich, G. (2005). Ongoing emergence: A core concept in epigenetic robotics. In Berthouze, L., Kaplan, F., Kozima, H., Yano, H., Konczak, J., Metta, G., Nadel, J., Sandini, G., Stojanov, G., and Balkenius, C., (Eds.), *Proceedings of EpiRob05 - International Conference on Epigenetic Robotics*, pages 63–70. Lund University Cognitive Studies.

Riedmiller, M. (2005). Neural reinforcement learning to swing-up and balance a real pole. In *Int. Conference on Systems, Man and Cybernetics, 2005, Big Island, USA.*

Riedmiller, M. and Braun, H. (1993). A direct adaptive method for faster backpropagation learning: The rprop algorithm. In Ruspini, H., (Ed.), *IEEE International Conference on Neural Networks (ICNN), San Francisco*, pages 586–591.

Stojanov, G. (2001). Petitagé: A case study in developmental robotics. In Balkenius, C., Zlatev, J., Kozima, H., Dautenhahn, K., and Breazeal, C., (Eds.), *Proceedings of Epigenetic Robotics 1.*