Using Second Order Learning to Evolve Social Representation (Theory of Mind)

Solvi Arnold Reiji Suzuki Takaya Arita
Graduate school of Information Science, Nagoya University
Furo-cho, Chikusa-ku, Nagoya, 464-8601, Japan.
+81-52-789-4716
solvi@alife.cs.is.nagoya-u.ac.jp reiji@is.nagoya-u.ac.jp arita@nagoya-u.jp

ABSTRACT
We demonstrate how basic social representation ability can be evolved in neural networks. Agents are evolved to perform a base behaviour and predict the actions of a partner performing the base behaviour. For the base behaviour agents pick actions in accordance with the environment and their mental state (a bit string). When predicting others' actions agents see the environment but not the mental state of the partner, so this task requires agents to observe the partner agent and learn. A neuromodulation mechanism is introduced to allow for evolution of the requisite learning ability. We run two sets of experiments. In the first, the mental state of the partner is constant over the duration of the interaction. This produces a first order learning task (essentially supervised learning). In the second, the mental state of the partner changes frequently, one bit at a time. This produces a second order learning task (unsolvable with supervised learning). Analysis shows that nets evolved using the second order task learn by decoding the partner's mental state from its behaviour. Connections in these nets specialize on individual bits of the partner's mental state, allowing us to read individual bits of knowledge about the other agents' minds. Such bit-level representation of mental states is not observed in nets evolved using the first order learning task. This supports the idea that evolutionary selection for second order learning is selection for representational cognition.

Categories and Subject Descriptors
I.2.6 [**Artificial Intelligence**]: Learning – *Connectionism and neural nets;* I.2.0 [**Artificial Intelligence**]: General – *cognitive simulation;* I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods – *Representations (procedural and rule-based).*

Keywords
Evolution of mind, Representation, Theory of mind, Neural networks, Social evolution, Modularity.

1. INTRODUCTION
This paper demonstrates how a simple form of social representation ability can be evolved, from scratch, in neural networks. There has been some computational work on social representation abilities in the field of artificial life, mostly under the term "theory of mind" (or ToM for short) [4][5][6][9][10]. Most of such work has focused on the evolution of recursion level. The ability to represent other minds is generally assumed (i.e. a format for knowledge about other agents is given). The project of evolving this representational ability from scratch, however, has remained largely untouched. The dearth of work on the representational fundaments of theory of mind highlights the more general problem that we do not have a clear understanding of how to evolve representation. We have previously put forward the idea that evolutionary selection for second order learning is selection for representational cognition [1]. We test this idea here for the case of social representation abilities.

2. MODEL
We let a population of neural network agents evolve to solve a social task. Agents interact in pairs. In each pair, there is a fixed role division: One agent (A0) acts at zero-order ToM, meaning it ignores the other agent and simply reacts on basis of the state of the environment and its own mental state. The other agent (A1) acts at first-order ToM, meaning it tries to predict the behaviour of A0 agent. In a richer model, the A1 agent would be tasked with acting in anticipation of A0's behaviour, but for simplicity we limit ourselves here to prediction only (generally speaking, to act in anticipation of an action, one needs at least an implicit prediction of that action).

Our model is not intended to capture any specific social interaction scenario in particular. Instead we take a more abstract approach, in which the logic that determines the fitness payoff for performing a given action in a given situation is generated randomly (within certain restrictions). The idea is that if arbitrary scenarios can be handled successfully, then the model has generality. Thus there is no concrete interaction scenario to speak of; there are merely *environmental states*, *mental states*, *actions*, and a randomly generated *base logic* that relates these elements.

Environmental state (env): An integer, in the range $[0...N_e)$, with N_e set to 6 in the experiments shown here. The environmental state is shared between interacting agents (i.e. both agents see the same state). The environmental state changes every timestep.

Mental state (mst): A bit-string of length N_m (set to 4 in the experiments discussed in this paper). In the A0 role, the agent has a private internal state, invisible to its interaction partner. Depending on the experiment, mental states remain constant over the course of the interaction of an agent pair or change gradually.

BICT 2014, December 01-03, Boston, United States

DOI 10.4108/icst.bict.2014.257889

Action: An integer, in the range $[0...N_a)$, with N_a set to 4 in the experiments discussed in this paper. At each time-step, each agent outputs an action.

Base logic: generates the optimal A0 action choice for each (environmental state, mental state) pair. The base logic abstractly represents social scenarios.

We set up two sets of experiments, with nearly identical tasks save for a small tweak. This tweak makes the difference between first and second order learning, so we can observe what the presence of absence of selection for second order learning does in otherwise identical settings. This lets us attribute any representational features that evolve exclusively in the second order learning experiments to selection for second order learning.

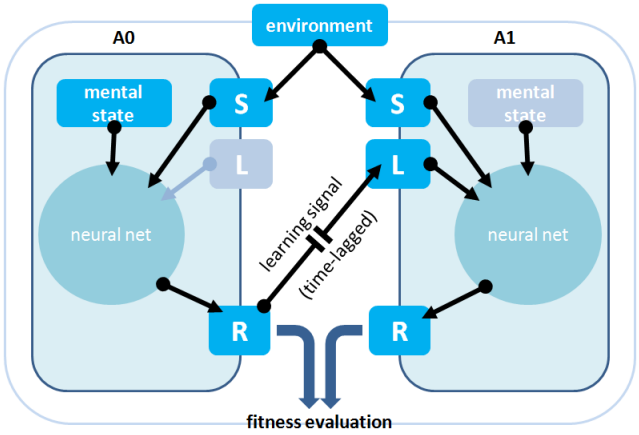


Figure 1. Schematic of agent interaction. A0 computes its response (R) from a shared environmental stimulus (S) and its private mental state. A1's response (R) is a prediction of this action. After the prediction is made, A0's actual response is revealed to A1 as data to drive its learning process (L).

2.1 Agent Interaction

Each interaction sessions spans a number of steps. Figure 1 shows the information flow within a single step, and Table 1 gives the event order. The structure of an individual step is the same regardless of the experiment type (first or second order learning).

Table 1 Event order within a single interaction step.

event	Agent A0	Agent A1
1	Sees environment and mst	Sees environment only
2	Computes own action	Computes prediction of A0's action
3	-	Observes A0's actual action and learns

The structure of an interaction session differs between the two experiment types. Before we look at the actual task setups, let us first articulate what we mean by second order learning. Very generally, (first order) learning ability can be conceptualized as a system's ability to improve its stimulus-response mapping in reaction to received information. Second order learning ability, then, is a system's ability to improve its first order learning ability in reaction to received information. More precisely, a learning event (a weight update in the case of a neural network) is a second order learning event if after the event, future observation of some information x will cause a larger improvement in the stimulus-response mapping than it would have without the event.

Table 2 gives the session structures for first and second order learning experiments. How is it that these tasks differ in learning order? The first order learning task presents what is essentially a supervised learning task: During the learning phase, A1 sees environments and A0's responses to those environments (which A0 determines on basis of its mental state). In the test phase, A1 has to recall what it has seen in the learning phase. Learning amount to linking stimuli (environments) to responses (action predictions), but there is no need for learning events to modify the way future observations modify the stimulus-response mapping, so there is no need for second order learning in this task. Tasks of this sort can be tackled very well with standard error back-propagation learning.

Table 2. Task structure. Time runs downward. Each round presents each environment (env) exactly once, in random order. The (initial) mental state (mst) is picked at random. Left: First order learning task structure (reduced example). A1's performance is measured during the test phase only (its predictions during the learning phase are inconsequential). Right: Second order leaning task structure (reduced example). Every round, one randomly picked bit of A0's mental state changes. A1's performance is measured continuously. Actual parameters: rounds per learning phase = 4, rounds per test phase = 2, rounds per main phase = 9, steps per round = N_e = 6, mental state size = N_m = 4 bits.

First order task

phase	round	step	mst	env
learning	0	0	001	0
		1		3
		2		1
		3		2
	1	4		1
		5		2
		6		0
		7		3
test	2	8		2
		9		3
		10		1
		11		0

Second order task

phase	round	step	mst	env
main	0	0	011	0
		1		3
		2		1
		3		2
	1	4	001	1
		5		2
		6		0
		7		3
	2	8	101	2
		9		3
		10		1
		11		0

Now consider what happens in the second order task: A1 observes environmental stimuli and A0's actions just like in the first order task, but A0's mental state changes at the start of each round, and each environment is seen only once per round. This means that a memory of which action A0 picked last time in a given environment is useless as a prediction of what A0 will pick the next time

that same environment comes around. At first sight it may seem impossible for A1 to predict A0's behaviour under these conditions (and for a standard error back-propagation learner it would be[1]). However, each mst produces a distinct *behaviour pattern*, so observations can be used to identify *which behaviour pattern* A0 is performing. Now note that A0's mental state changes only gradually: one bit per round. This means that given the mental state at round i, there are only N_m possibilities for the mental state at round i+1 (because there are N_m possible bit-flips), and hence just N_m possible behaviour patterns for round i+1. An agent that remembers (in some form or another) the behaviour pattern it observed in the previous round can rule out all but N_m possible behaviour patterns for the current round, which should greatly speed up its learning process in the current round. If this happens, we can say that the learning events that occurred in the previous round improved the learning process in the current round, and thus qualify as second order learning events.

Letting mental state bits change one by one is not an arbitrary choice. If we let the individual bits be pieces of knowledge, it makes sense that they would change one at a time. If we let them be affective attitudes towards a set of items (foods, persons, what-not), then too, they should usually change independently. Although very simplistic, the one-bit-at-a-time rule is an expression of the idea that for intelligence to evolve, one needs a mixture of stability and dynamicity (see e.g. Godfrey-Smith, 1996). A setup with no change (like the first order task) allows for a solution based on simple memorization, while maximal change (continuously randomizing the mental state) precludes any solution.

We hypothesize that the second order learning task should guide evolution toward representational solutions. We will see whether it does in the results section, but let us consider why it might. We saw that A1 will need to distinguish between possible behaviour patterns, while the set of possible behaviour patterns for a given round is to be inferred from the behaviour pattern observed during the preceding round. A0's behaviour patterns derive from its mental states, and the web of possible transitions between behaviour patterns derives from the distances (in bit-flips) between mental states: direct transition between two behaviour patterns is possible if and only if the distance between the underlying mental states is a single bit-flip. It would seem that clear and distinct knowledge of individual mental state bits should come in handy here. Hence we hypothesize that neural networks evolved to solve the second order task will, over the course of their interactions, learn representations of partners' mental state.

2.2 Neural Network Species

We keep the neural network architecture simple, imposing no fixed structure. Each net is composed of a list of neurons and a list of connections. Table 3 lists the attributes of neurons and connections, respectively. Propagation order is fixed and follows the neuron list order. The list is composed as follows: Two input neurons inform the net of its current role (one neuron active when running as A0, the other when running as A1). Next $2N_m$ neurons encode the current mental state, using two neurons per bit (one active when the bit is 0, the other active when the bit is 1). N_e neurons give the present environment, with the neuron corresponding to the present environment taking activation 1 and the rest 0. N_a neurons give the partner's action during learning, with the neuron corresponding to the chosen action taking activation 1 and the rest 0. Next follow N_h hidden neurons (we used $N_h = 12$ during A0 evolution and $N_h = 24$ during A1 evolution, see below). Last there are N_a output neurons.

[1] Consider how error back-propagation-based supervised learning would fare in the second order task. When it observes that the partner picks action x in state p, it will strengthen the connection between state p and action x. This is not a useful update, as the partner will change its mst before state p comes around again, and then pick some action y instead.

Table 3. Neuron and connection attributes. Every genetic attributes is defined by a gene in the neural network's genome. Dynamic attributes are updated every interaction step and do not inherit.

Neurons	**Attribute**	**Symbol**
Dynamic	Activation	A
Genetic	Activation bias	B
	Activation function	F
Connections	**Attribute**	**Symbol**
Dynamic	Weight	W
Genetic	Innate weight	W^g
	Input neuron index	I
	Output neuron index	O
	Neurotransmitter neuron index	T
	Neuromodulation neuron index	M

At the end of propagation, we determine which of the output neurons has the highest activation value, and pick the corresponding action as the net's choice of action (for A0) or prediction (for A1). Propagation amounts to iterating over the neuron list, computing the activation of each subsequent neuron. A neuron's activation is computed as follows:

$$A_i = F_i\left[\sum_{\{c|O_c=i\}}\{W_c * A_{I_c} * t(c)\}\right]$$

where

A_i = activation of neuron i
F_i = activation function of neuron i
I_c = input index of connection c
O_c = output index of connection c
W_c = current weight of connection c
t(c) = neurotransmitter value of connection c

The activation function (F_i) of a neuron is determined by a neuron-level gene. There are two options: a clipped linear function and a step function. Output neurons use a non-clipped linear function instead, to avoid output ambiguity. The neurotransmitter value t(c) is determined as follows:

$$t(c) = \begin{cases} 1 \text{ if } T_c = \text{NONE} \\ 1 \text{ if } A_{T_c} > 1 \\ 0 \text{ otherwise} \end{cases}$$

What the inclusion of the neurotransmitter does is allow an easy way of making propagation across a connection conditional (on the activation of the neuron pointed at by the transmitter index of the connection). Theoretically speaking, this should not extend the nets' computational potential (not beyond what an increase in network size would do), but we find that the inclusion facilitates

evolution, probably by simplifying the structures needed for many basic computations (for example, it allows implementation of XOR without any interneurons). In the case of A1, after propagation to determine an action prediction, A0's actual choice of action is revealed, and another propagation pass is performed, now with weight updating. In this pass, output is ignored. For connection weight updating we use the following rule:

$$\Delta W_c = 2 * A_{M_c} * A_{I_c} * t(c)$$

where

ΔW_c = change in weight of connection c
M_c = modulation index of connection c
and other values as before.

Weights are clipped to the [-1,+1] range after updating. The "learning rate" of 2 might surprise some readers, as it is common to use small (<<1) learning rates in traditional neural network learning algorithms. However, the update rule is not part of a gradient descent algorithm (in fact it is not part of a learning algorithm at all). Updating does not proceed by slowly reducing an error measure, but by large sudden changes. Nothing in the update rule guarantees that updates will be beneficial. All the update rule provides is a mechanism for weight modification. It is up to evolution to structure the networks so as to have beneficial update dynamics. This approach is borrowed from Soltoggio et al.[7][8], although our implementation differs in various ways. We want to allow that a single update can take a weight from any value within the weight range to any other, so we use the size of the weight range (2) where one would usually expect a small learning rate.

2.3 Mutation

Mutation rates are generated per individual as follows: rate = R^6, where R is a random number in the interval [0, 1]. This rate is used as the mutation probability for every individual gene. We generate the mutation rate randomly to ensure that mutation produces both heavily and lightly mutated individuals (the former as especially important during early stages of evolution, the latter during the later stages). The exponent controls mutation intensity spread. Note that it is possible to generate a rate of 1, completely randomizing all genes. Addition and removal of connections is handled separately (as these operations modify the number of genes). During mutation, there is a 0.25 probability of adding a connection (if the current number of connections is below the maximum) and a 0.26 probability of removing a connection (if there are any). After a connection is added, there is again a 0.25 probability of adding another one, and so forth (same for deletion). These probabilities are biased slightly toward deletion, so as to keep network size in check (though we have not investigated the necessity of this measure). When mutation results in a connection that runs against the propagation order, the connection is removed.

2.4 Evolution Process

We evolve the A0 and A1 behaviour separately. We first evolve five A0 behaviours, and then use each as the basis for evolving A1 (prediction ability) twice, once using the first order learning task and once using the second order learning task.

Separating A0 and A1 evolution lets us ensure that A1 has a stable target to evolve to. During A1 evolution, we apply stabilizing selection on the A0 behaviour (any deviation from the A0 behaviour the run was initialized with receives a large fitness penalty). It is possible to evolve A0 and A1 in tandem, however then changes in A0 interfere with A1 evolution (which may be interesting in itself, but for our purpose it's an unnecessary complication).

In both A0 and A1 evolution we use a population of 160 individuals, which we split into ten groups of 16. Groups evolve individually for 10 generations at a time, and are then scrambled. We use elitism of 1 individual per group. Selection is rank-based. Given the differing purpose, evaluation differs between A0 evolution and A1 evolution. We describe both evaluation schemes below.

Table 4. Example generated A0 behaviour in tabular format, showing the action generated by every mental state in every environment.

mental state \ environment	0	1	2	3	4	5
0000	3	0	1	1	2	0
1000	3	2	1	1	3	3
0100	2	0	2	1	2	1
1100	1	2	2	1	3	1
0010	3	2	0	0	2	0
1010	3	2	3	3	2	3
0110	2	2	0	0	2	1
1110	1	2	3	3	2	1
0001	0	0	1	2	0	0
1001	0	3	1	2	3	3
0101	2	0	2	2	0	2
1101	1	3	2	2	3	2
0011	0	0	0	0	0	0
1011	0	3	3	3	3	3
0111	2	0	0	0	0	2
1111	1	3	3	3	3	2

2.4.1 Evolving A0

The A0 behaviour has no particular purpose or meaning, it merely serves as a target to evolve prediction ability (A1) for. Hence for A0 evolution we use a fitness measure that assesses suitability as a prediction task. Not every possible A0 behaviour would require learning from A1. Consider an A0 that returns action 3 for each and every (mst, env) pair. Such an A0 can be predicted without learning, so from A1 this behaviour would not require learning, let alone second order learning. We want an A0 that poses a non-trivial learning task to A1. To this end we evaluate A0 as follows: First we make a tabular representation of the A0 behaviour by running the network for every (mst, env) pair. Recall that when a net plays the A0 role, it does not use learning, meaning that the behaviour is fixed (hence the table is a complete description). Then we evaluate the table using a number of measures and conditions designed to pick out sufficiently challenging A0 behaviours as follows. First we check whether any two columns or rows are identical. A large fitness penalty is incurred for any such duplicates (none of the A0 behaviours used for A1 evolution had duplicates left). Next, we check if knowledge of any single mental state bit predicts the A0 action under any environment. For example, if for environment 5 the action is 3 for every mental state with the second bit a 0. For every instance of such single-bit predictability, a fitness penalty is imposed. This ensures that we can distinguish between memories of observed actions and representations of bits of A0's m. Say this condition is violated as in the example above. A connection weight that stores the observation that A0 picked action 3 in environment 5 will be indistinguishable from a connection that stores the inference that the second bit of A0's mental state is 0. We want memories of observations to be distinct from representations of mental state bits, hence this penalty condition. If we look at the example behaviour in Table 4, we can see that if we know only that A0's first mst bit is 0, then the actions it would possibly take in environment 5 are 0, 1, and 2.

Hence we cannot infer the action from this single bit of knowledge, and the condition holds (for the case of the first bit 0 in environment 5).

While we penalize direct inference from single bits to actions, we do allow direct inference from actions to single bits. For example, it is ok for action 3 in environment 5 to reveal the value of the first mental state bit. This is indeed the case in the example behaviour: if we see action 3 in environment 5, we can infer that A0's first mental state bit is 1. These sort of situations allow for learning about mst bits, but note that a connection that simply stores the observation that A0 picked action 3 in state 5 will be distinct from a connection representing that the second mst bit is 1: it would fail to capture half the cases where the second bit is 1. In order to make complete representations of mst bits, multiple pieces of such information need to be integrated. To ensure that the necessary pieces are readily available, we include the condition that for every possible mental state, observing the actions for all environments will allow inference of every mst bit at least once. Violations of this condition receive a penalty.

We also impose a condition to avoid the existence of relations between actions and mst bits that do not involve the environment. For example, if action 2 only ever occurs when the third mst bit is 0, then it is impossible to distinguish the memory of having seen action 2 (in any environment) from a representation of the third mst bit. To ensure that no such relations exist, we demand that every action occurs in over half (we have set the threshold to 10) of all msts (i.e. for any given action, at least 10 msts produce that action in at least one environment). Since any bit will be 0 in only eight msts and 1 in only eight msts, this condition precludes existence of environment-independent relations between actions and bits. A penalty is imposed to the extent a net falls short of this threshold.

Lastly we add a small fitness factor suppressing the highest possible fitness a non-learning A1 agent could attain in interaction with the A0 under evaluation. This helps to increase the relative importance of learning. This evaluation simply amounts to checking the number of occurrences of each action in each column of the table. An optimal non-learner would for every environment pick the action that occurs most often in the column of that environment. For example, if action 0 occurs three times (i.e. is produced by three msts), action 1 four times, action 2 six times, and action 3 three times, then knowing nothing about the mst we are best off guessing action 2, which will be correct 6 out of 16 times. In a perfectly balanced A0 with 16 msts and four actions, each action occurs exactly four times per column, and the best non-learner will have a fitness of 4/16 = 0.25. No particular threshold was set, but the A0s used in our experiments all have values of at most 0.33, meaning that A1 performance will be strongly dependent on learning ability.

For A0 evolution, we set the maximum number of connections in a network to 100, and set N_h (the number of hidden neurons) to 12. A0 evolution is run for 30000 generations, after which the best network of the final generation is stored. If this network still has any penalties, we discard it. Otherwise, it is used to seed two populations for A1 evolution (one population using the first order task and one the second order task). The A0 behaviours used are shown in Figure 5.

2.4.2 Evolving A1

As noted, A0 behaviour merely serves as a target to evolve prediction ability (A1) for. Hence all analysis and discussion below focuses on A1 evolution. At the start of A1 evolution, the networks are expanded in size. The number of interneurons is doubled from 12 to 24, the input neurons for observing A0 action for learning are added in, and the maximum number of connections is increased from 100 to 200. However the nets remain functionally identical (as no actual connections are added or modified). Each net plays the A1 role against 16 partners, each starting from a different mst. As discussed above, the mst remains constant throughout the interaction in experiments using the first order task, and changes one bit per round in the second order task. The A0 and A1 agents in an interaction are generated from the same genome. The population is evolved for 300000 generations in runs using the first order task and 600000 generations in runs using the second order task (in preliminary experiments, we found that with the first order task 300000 generations suffices to reach near-optimality, after which no meaningful evolution occurs).

3. ANALYSIS

It will be easy enough to determine whether effective learning ability evolved, given that there is a clear ceiling to the performance that can be achieved without. However, assessing the degree to which a given solution is representational is less easy. We devised a method of quantifying the degree of correspondence established between individual connection weights and partner mst bits in the learning networks. It's rather crude, but gives at least a rough indication of the representationality of a solution.

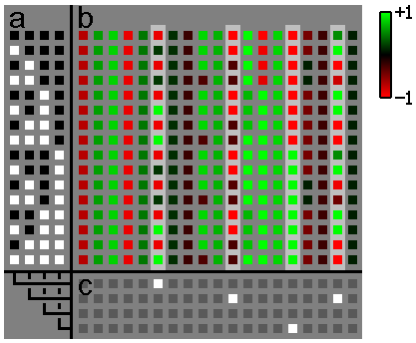


Figure 2. Mock-up example of matrix format used to quantify and visualize representationality. (a) Each row shows one mst, with black = 0 and white = 1. (b) Each row shows the connection weight vector that results when the net learns on a partner with a particular mst. Highlighted columns indicate connections that show correspondence with individual mst bits. (c) Rows indicate correspondence sets (see text). White dots indicate set membership. We see that the correspondence set of the first bit has one member connection, that of the second bit two, or the third zero, and of the fourth one.

First we look for connections whose weights show correspondence with mst bits in the partner agent. We do this as follows: We let the net under investigation learn on A0 partners as it usually would (following the task the net is evolved for), for 32 normal lifetimes. After every round, we take a snapshot of the net's connection weight vector. Then we compute the average connection weight vector per partner mst (so for example, we take all snapshots obtained by learning on a partner with mst 0101, and compute their average). The resulting set of 16 connection weight vectors can be put together into a matrix. Figure 2 shows a reduced mock-up example of such a matrix. The rows of the matrix (b) are the average connection weight vectors for every mst (the msts are shown at the left (a) of the figure). The columns now show us how an individual connection responds to the partner mst.

In the example, we see that many connections are not sensitive to the partner mst at all, i.e. many columns show uniform weight values (note that weights of connections without plasticity will always be uniform). Some connections do show sensitivity. For these connections we analyse whether they show correspondence with any of the mst bits. We do this by checking, for each bit, whether there exists a threshold value such that the connection's weight is above the threshold for one value of the bit and below the threshold for the other value of the bit. For example the first highlighted column shows a connection whose weight takes a value above 0 whenever the partner's first mst bit is 1, and a value below zero whenever the partner's first mst bit is 0. We can say that this connection *corresponds* to the first mst bit, in the sense that we can read out the value of the first mst bit from its weight. We identify all such corresponding connections for each mst bit. Now we have, for each bit, a (possibly empty) set of connections corresponding to it. We will call such a set a *correspondence set*. Note that the correspondence sets of a given net are, by mathematical necessity, non-overlapping. Hence the correspondence between the sets and bits is decidedly one-to-one.

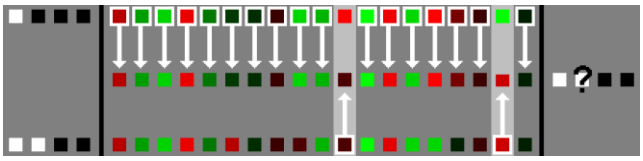


Figure 3. Recombining learned knowledge across nets (concept image). We take the weight vector of a net trained on a partner with mst m (top row, here m = 1000) and the weight vector of a net trained on a partner with mst m' (bottom row, here m' = 1100). We now make a hybrid weight vector as follows: for every connection contained in the correspondence set for the differing bit (in this case the second bit), we use the weight from the vector on the top, and for every connection not contained in the correspondence set we use the weight from the vector on the bottom. To the extent that knowledge about the second bit is localized to the correspondence set, the resulting net should behave as if it had learned on a partner with mst m (1100 in the example).

Existence of non-empty correspondence sets is interesting, but does not guarantee that a net is really using a them as representations of mst bits. To confirm that we have to also assess to what extent the correspondence sets are playing the *functional role* of representations of the bits they correspond to. We do this by "transplanting" correspondence sets between individuals. The process is illustrated in Figure 3. We take two identical nets, and let them learn on partners that differ by one mst bit, for example 1000 and 1100. Let us call the resulting individuals A_{1000} and A_{1100} for convenience. Now we copy the weights from the correspondence set for the differing bit from one net to the other. So for example, we take A_{1000}, and overwrite only the weights of connections contained in the correspondence set for the second bit with the weight values that A_{1100} has on those connections. If, by overwriting the correspondence set, we have selectively flipped the net's knowledge about the second bit of its partner mst, then the resulting net should predict as if it had learned on a 1100 partner. We measure to what extent this is the case by letting A_{1100} and the hybrid net both predict the actions of a 1100 partner. During this test we disable learning (as both nets have already completed their learning process). We ignore environments where a 1100 partner and a 1000 partner pick the same action and environments where A_{1100} fails to make the right prediction, as these are non-informative. We repeat this transplantation procedure for every bit-flip (excluding bits with empty correspondence sets) in every possible mst, and take the average performance values as a measure of the *functional importance* of the correspondence sets. For empty correspondence sets this analysis cannot meaningfully be performed, they default to a functional importance of zero. Finally, we take the average functional importance of all correspondence sets as a global measure of *representationality* of the solution. This measurement is performed once every 500 generations on the then best individual in the population.

4. RESULTS

Figure 5 shows the evolution of performance and representationality. We see that first order runs invariably produce near perfect performance, yet show only limited evolution of representationality. Non-empty correspondence sets are seen to appear but their number and functional importance remain low and bear no straightforward relation to performance. The first order task does not appear to select for representational solutions. Second order runs evolve more slowly, and show lower performance overall. This is not surprising, considering that the second order task is vastly more difficult. Here we observe a strong relation between performance and representationality: increases in one often coincide with increases in the other, and runs resulting in higher performance also result in higher representationality. Networks from the final generations of three best performing second order runs have non-empty correspondence sets for all bits, meaning that after learning, we can read out their partner's mental state bits from the weights of the connections in their correspondence sets. Also, given that knowledge about the partner mst is strongly focused in the correspondence sets in these runs (high functional importance scores), we can rewrite individual bits of knowledge by copying correspondence set weights between instances of these nets, as described in the analysis section, with little reduction in performance. These nets can be said to represent their partners' mental states, with high specificity, at the level of individual bits. This result indicates that selection pressure on second order learning led evolution towards representational solutions.

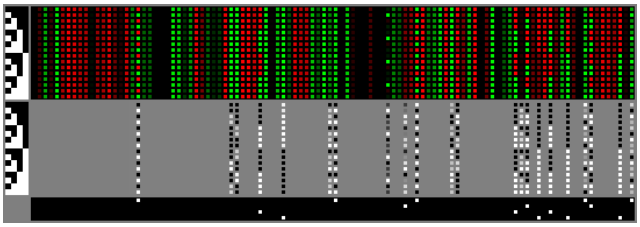
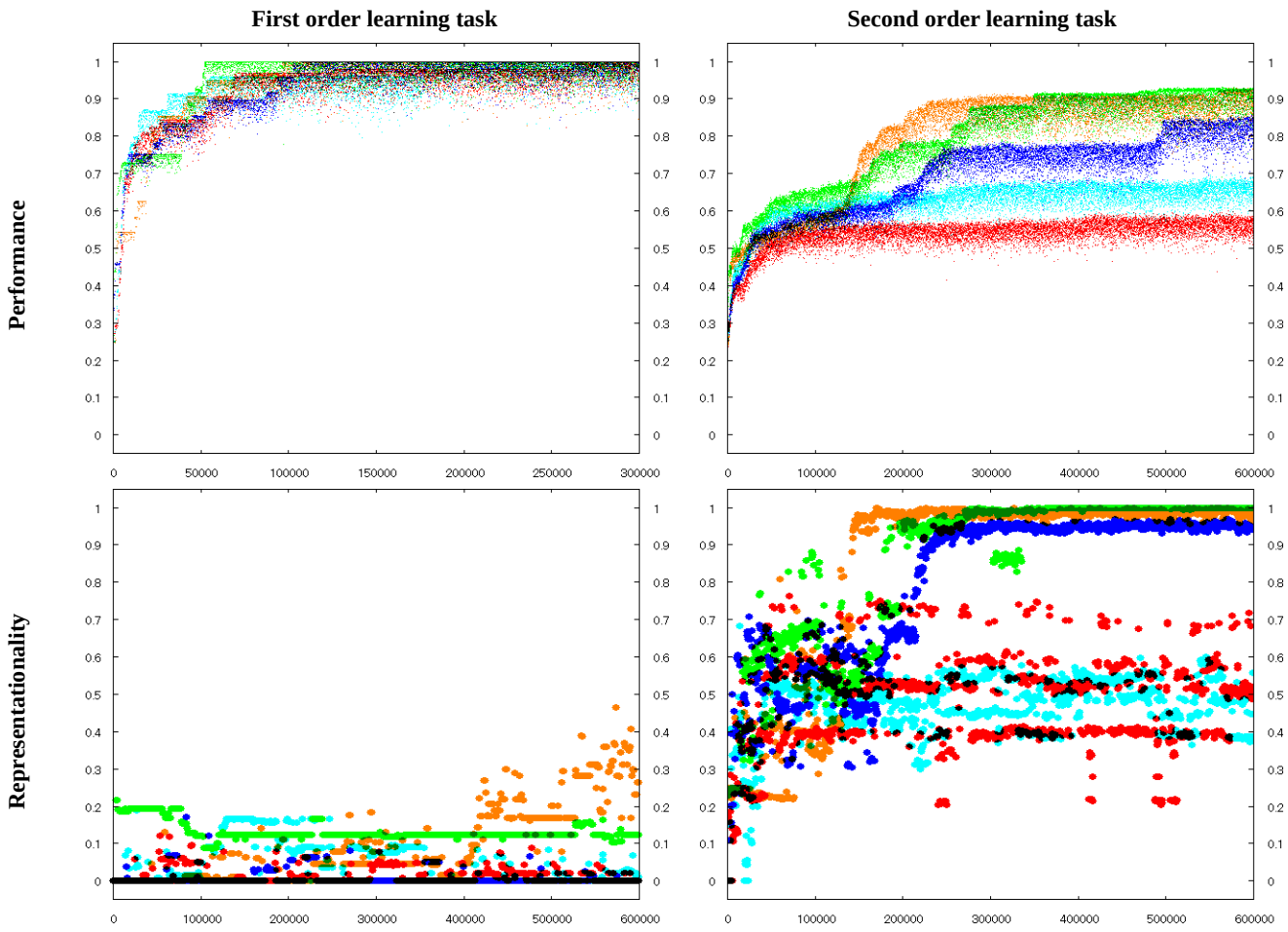


Figure 4. Post-learning weight vectors for a net evolved using the second order learning task. Format as in Figure 2. The middle area is colour-adjusted to help visualize what happens on connections with narrow weight ranges, displaying the lowest weight value in a row as black, and the highest as white (for columns with weight variability). Every mst bit can be seen to have a non-empty correspondence set.

5. DISCUSSION

Our results show that evolutionary selection for first order learning and second order learning lead to different neural architectures. First order learners showed no convincing evidence of representation of partners' mental states, while second order learners do.



m\e	012345	012345	012345	012345	012345
0000	333313	000011	300313	331333	301120
1000	233212	100301	303013	002333	321133
0100	332332	001122	211212	101123	202121
1100	132222	101101	212112	102020	122131
0010	013300	010010	300300	231332	320020
1010	003000	110300	303000	022332	323323
0110	322330	011120	211201	211121	220021
1110	322220	111100	212101	122021	123321
0001	311113	333022	030323	333303	001200
1001	201111	233331	033023	000003	031233
0101	111132	332222	121222	103113	202202
1101	101122	232231	122122	100010	132232
0011	010301	323023	030330	233302	000000
1011	000001	223333	033030	020002	033333
0111	120331	322223	121231	223212	200002
1111	120221	222233	122131	120012	133332

Figure 5. Evolution processes. A1 evolution processes for first (left panels) and second (right panels) order learning experiments. X-axis shows generation. Performance and representationality normalized to [0, 1] range. Representationality is computed once every 500 generations, from the fittest individual in the population at that time. Performance scores are computed over the unmutated offspring of the previous generation's elite. In first order runs, performance is computed over the test phase; in second order runs, over all steps that are in principle predictable for a perfect second order learner. Colours in the graphs indicate the A0 behaviour used. Bottom left panel: the five A0 behaviours used (displayed as in Table 4). Each A0 behaviour was used in one first order run and one second order run.

Second order learners' representation of partners' mental states is localized and specific, with small sets of connections allocated to the individual bits of the mental state. This is a far cry from the diffuse, distributed representation style that we have come to associate with neural networks. Distributed representation [3] is often touted as a strength of the connectionist paradigm, but in our model, we observe that under selection for second order learning, evolution steers clear of distributed representation. A full theoretical explanation of why this should be so can be found in [1], but let us briefly outline the general idea here.

The goal of first order learning is to produce some optimal stimulus-response mapping. Theoretically, any mapping can be implemented in infinitely many ways (e.g. for any given behaviour there are infinitely many neural networks that produce that behaviour). In first order learning, implementations are judged solely on basis of the mappings they produce. Any differences (in e.g. representation style) between implementations producing the same mapping are inconsequential.

In second order learning the situation is quite different. Here the focus is not on the mapping, but on the way the mapping changes.

Second order learning is modification of the implementation such that future (first order) learning events will come to produce bigger improvements in the mapping. Second order learning adapts the dynamics of the learning system to the dynamics of the task, so that when the task (in our model: the partner agent's behaviour) changes, the system can quickly change along. The dynamics of a changing task derive from (and consequently express) the underlying structure of the task (in our model, change in partner behaviour derives from and expresses change in hidden mental states). To align its dynamics with the dynamics of the task, the learning system must in some form or another assimilate the task's underlying structure into its own implementation. This assimilation of task structure can be recognized as representation.

In the model, nets have to track the changing behaviour of a partner, whose behaviour derives from individually changing bits of information (the mental state). This dynamic is most easily tracked by an implementation that employs individually modifiable bits of "knowledge" corresponding to the partner's mental state bits. By "individually modifiable" we mean that these bits of knowledge can be modified without affecting other bits of knowledge. This in turns is achieved by localizing the bits of knowledge to non-overlapping sets of connections (the correspondence sets).

Note that for this effect to occur, we need the interplay between three adaptation processes: evolution, first order learning, and second order learning. Representation emerged in the evolution of second order learning because representation helps second order learning do its job of dynamically adapting first order learning to a dynamic task.

Neural networks are typically produced using a single level of adaptation (evolution *or* learning), with as target a specific behaviour (stimulus-response mapping). Our results suggest that if we want to understand and model the emergence of representational cognition, we should consider the interplay between adaptation processes, and consider as evolutionary targets not just specific behaviours but also specific learning abilities.

6. CONCLUSIONS & FUTURE WORK

In this paper we have looked at the effect of selection for second order learning in a simple and abstract social scenario. We observed a strong tendency toward focused representation of partners' mental states, with knowledge about individual bits being localized to small, non-overlapping sets of connections. The nets observe behaviour, but learn mental states. This tendency is absent when we restrict selection to first order learning only (in an otherwise near-identical task). These results provide strong support for the idea that selection for second order learning, specifically, can drive evolution of ToM-like representation.

Future directions for this work are: 1) Improvement of the genetic algorithm. For second order learning runs, evolution time is very long and a proportion of runs get stuck in local optima. Experimentation suggests that improvements to the algorithm speed up evolution and reduce the number of runs that get stuck in local optima. This should let us tackle more substantial tasks and allow us to produce a more substantial data set. 2) Analysis of neural circuitry. We think that in order to produce second order update dynamics, networks need circuitry with at least two plasticity loci wired in series. 3) Application to concrete task. Here we have used an abstract task, but it may be instructive to try tackling a small-scale game or a relevant task from experimental psychology. 4) Extension to recursive ToM.

7. ACKNOWLEDGMENTS

This work was supported by a Grant-in-Aid from the Japanese Society for Promotion of Science (25-10032).

8. REFERENCES

[1] Arnold, S., Suzuki R. and Arita, T. 2012. Second Order Learning and the Evolution of Mental Representation. In *Proceedings of the Tenth International Conference on Artificial Life*, MIT press, Cambridge, MA, 301–308.

[2] Godfrey-Smith, P. 1996. *Complexity and the Function of Mind in Nature*. Cambridge University Press.

[3] Hinton, G. E., McClelland, J. L. and Rumelhart, D. E. 1986. Distributed Representations. In *Parallel Distributed Processing - Vol. 1*. David E. Rumelhart, Ed. MIT press, Cambridge, MA.

[4] Kanai, H., Arita, T. 2008. Effects of Recursive Estimation with a Theory of Mind in a Sequential-Move Game. Proceedings of the 35th SICE Symposium on Intelligent Systems, 91-96.

[5] Minoya, K., & Arita, T. 2011. An artificial life approach for investigating the emergence of a Theory of Mind based on a functional model of the brain. *Proceedings of the 2011 IEEE Symposium on Artificial Life*, 108-115.

[6] Noble, J., Hebbron, T., Van Der Horst, J., Mills, R., Powers, S. T. and Watson, R. 2010. Selection pressures for a theory-of-mind faculty in artificial agents. In *Proceedings of the Twelfth International Conference on the Synthesis and Simulation of Living Systems*, MIT press, Cambridge, MA, 615-615.

[7] Soltoggio, A., Durr, P., Mattiussi, C. & Floreano, D. 2007. Evolving Neuromodulatory Topologies for Reinforcement Learning-like Problems. In *Proceedings of the IEEE Congress on Evolutionary Computation*, 2471-2478. doi:10.1109/CEC.2007.4424781

[8] Soltoggio, A., Bullinaria, J. A., Mattiussi, C., Durr, P. & Floreano, D. 2008. Evolutionary Advantages of Neuromodulated Plasticity in Dynamic, Reward-based Scenarios. In *Proceedings of the Eleventh International Conference on the Simulation and Synthesis of Living Systems*, MIT press, Cambridge, MA, 569-576.

[9] Takano, M., Kato, M., Arita, T. 2005. A construcive Approach to the Evolution of the Recursion Level in the Theory of Mind. *Cognitive Studies*, 12(3):221-233.

[10] Takano, M., & Arita, T. 2006. Asymmetry between Even and Odd Levels of Recursion in a Theory of Mind. *Proceedings of the Tenth International Conference on the Simulation and Synthesis of Living Systems*, MIT press, Cambridge, MA, 405-411.