Learning and retrieval of hierarchically organized information in a simple, one-layered RNN

Holk Cruse and Malte Schilling

Abstract—How is it possible for an autonomous agent to learn and retrieve hierarchically organized information? The question is particularly interesting if there is not a simple, tree-like hierarchy, but when low-level items may belong to several superordinate elements. In this article we propose a solution for this problem following the ideas of O'Connor et al. [1], which are guided by the observation that children learn superordinate concepts from implicitly given information. For the simulation, we use a simple, one-layered RNN consisting of IC units and a very simple learning rule based on teacher forcing [2], [3]. We assume the capability of figure-ground separation as being given. Furthermore, the agent is assumed to owe sensory systems. There are, however, no explicit, preformulated concepts given like ‘Red’. Further, the agent is assumed to be able to record words. Each object presented to the agent is separately and individually learned and represented in the RNN forming an episodic memory.

We show that, using this simple approach, learning and retrieval of hierarchically organized information is possible, although this information is not given explicitly and no hierarchical structure can be found in the network. Learning is very fast. The net shows top-down generalisation and bottom-up activation. Furthermore, asymmetric priming effects can be observed similar to those found in human subjects. The agent is able to chunk different sensory inputs to represent the same object in memory, but nevertheless being able to distinguish between the different stimuli if, during learning, a supervisor labels the different stimuli with the same name.

I. INTRODUCTION

How is memory organized? The classical assumptions refer to the observation that items found in the world can be ordered using a hierarchical system. For example, different individual objects with some common properties may be categorized as cows. Together with other objects, e.g. horses, they may be grouped as mammals, or, on a higher level, as animals. Such a tree-like hierarchy has been proposed by early AI to organize objects occurring in the world within a storage of an artificial system [4]. This proposal was paralleled by the suggestion that the architecture of the human memory may follow such a hierarchical structure, too. There are a number of findings in agreement with this view (for an extensive review see [1]), in turn triggering the question how this principle could be realized in terms of neuronal structure, and, of course, followed by the question concerning the mechanisms that allow such a structure to be learned by a neuronal network.

The problem is however complicated by the fact that a strict tree-like hierarchy is not sufficient to explain basic observations. Even when focusing on animals only, which, using the evolutionary tree, can definitely be ordered in such a way, humans can use and do use other structures as, for example, grouping all flying animals like birds, bats, and insects together in one category. This means that hierarchies are not strict but may be variable. To deal with these questions, O’Connor et al. [1] have proposed a fascinating solution, using a one layered RNN and a specific backpropagation algorithm. In this approach, hierarchies are not explicitly given during learning, but emerge in a self-organized way.

In this article, inspired by the ideas of O’Connor et al. [1] and Vilarroya [5], we will propose a similar solution using an even simpler RNN and a much simpler learning rule [2], [3]. This type of RNN has already been used to build up a memory for insect-like navigation [6]. This approach may be suited to further extend the memory structure of the insect-based, reactive structure Walknet [7]. The results show that the number of learning steps is unusually small. As we use a net consisting of only 27 units, the development of the weight values can easily be investigated.

Section II.A explains the basic properties of the system and the training situations, and section II.B the technical details of the network and the learning algorithm. Section III shows the results of different tasks investigated: In III.A. only four objects are treated, in III.B. four objects plus several superordinate concepts. In III.C. more superordinate categories are introduced leading to 22 input vectors. This section shows the weight distributions obtained and the behavior of the trained network. Section III.E shows the simulation of priming experiments performed with human subjects. Finally an application is presented in Section III.F showing that this network can also be used to chunk individual stimulus situations when they are given the same name.

II. THE MODEL

A. General Properties of the Model

To begin with, we describe the basic properties required by an artificial agent to cope with the questions mentioned...
its memory representing one specific object. Categories, just experiences sensory input that it will store in a human observer may call 4-legged, black-and-white. Confronted with a cow equipped with some properties which no innate categories. Imagine, for example, that the agent is following be written with capital first letters. That will be given to the system by a supervisor, will in the horse, and a table. To minimize possible confusion, words much simpler set of items compared to that used in [1].

We assume that it only stores location and that the location will not change during learning (for the case of changed locations and for introduction of time see below, Sect. III.G). Note that for each new situation experienced, a new Unit has to be chosen.

Therefore, in our example each object can be stored by a vector containing 14 components (Identifier, name, and, for each of the three properties (4-legged, surface texture, color, name) plus a further one called Identifier being explained as follows.

In principle, the agent may also be able to record the location where and the time when it has observed this object. To maintain a slot for storing this information, but nevertheless keep the network simple, we introduce a specific unit, called Identifier. However, to simplify matters we assume only one unit for each name word. Therefore, to represent an individual object, in total we need a vector able to represent four properties (4-legged, surface texture, color, name) plus a further one called Identifier pointing to it. How this by no means easy problem can be solved (To simplify the task, we perhaps feeling) different in detail (To simplify the task, we assume that a supervisor can tell the system which of the objects has a name word. These name words may, for example, be acoustically given by a supervisor. To simplify matters, we assume only one unit for each name word. Therefore, to represent an individual object, in total we need a vector able to represent four properties (4-legged, surface texture, color, name) plus a further one called Identifier being explained as follows.

To investigate the properties of our system, we use a much simpler set of items compared to that used in [1]. We have four objects only, two cows, Frieda and Emma, one horse, and a table. To minimize possible confusion, words that will be given to the system by a supervisor, will in the following be written with capital first letters.

We start with an agent that has no sensory experience and no innate categories. Imagine, for example, that the agent is confronted with a cow equipped with some properties which a human observer may call 4-legged, black-and-white colored, cow-type fur, and which by a supervisor is being named Frieda. Recall that this agent is not able to apply such categories, just experiences sensory input that it will store in its memory representing one specific object.

After having seen and learned the 14-component vector representing cow Frieda, the agent may see another cow, named Emma, according to information given by a supervisor. The sensory input of this new object may look a bit different and is also stored. For example Emma’s color may correspond to what is sometimes called red-colored, and Frieda’s black-colored. Similarly the 4-legs of Emma and Frieda may look a bit different, but still more similar than the 4-legs of the horse and of the table. Another object seen, also 4-legged, also with a fur, but brown in this case, is named ‘Horse’. The agent may also see an object being named ‘Table’, colored white, constructed of wood and containing four legs, too. Remember that, for each new object, i.e., for each new identifier we need an additional neuronal unit.

By learning these objects, the agent is able to build up an episodic memory. But can it also be able to represent some kind of hierarchy, for example by combining the two objects named Frieda and Emma as “cow”, or the two cows and the horse as “animal”? As has been stated [1], children do learn the hierarchical relation not by being explicitly taught that “Frieda is a cow” or “cows and horses are animals”, but by being confronted with an object, for example Frieda, that is, instead of being named “Frieda”, at another opportunity named “Cow” by a supervisor. Applying this suggestion to the training procedure, here, as did O’Connor et al. [1], the two objects Frieda and Emma, may, in other training sessions, be called ‘Cow’ instead of ‘Frieda’ or ‘Emma’. The two cows and the horse may correspondingly be called ‘Animals’. Furthermore, the agent may be told that the different surfaces observed in these three objects Frieda, Emma and horse are named ‘Fur’, although looking (and perhaps feeling) different in detail (To simplify the task, we assume that a supervisor can tell the system which of the sensory properties are meant by ‘Fur’, for example by pointing to it. How this by no means easy problem can be solved see Steels [8]). In addition, in the corresponding way the agent may learn that all the objects are characterized as being ‘4-Legged’.

To make the structure of the relation between the elements a bit more complex, the somewhat different colors of the two cows are given by the supervisor the category ‘Colored’, that of the (brown) horse and the (white) table the category ‘Uni’. Correspondingly, in some training sessions the items Frieda and Emma, but not horse and table, may be replaced by ‘Names’, here standing for names of individuals. Thus, altogether, we have now 7 higher-level, or subordinate categories (Cow, Animal, 4-Legged, Names, Fur, Colored, Uni).

The structure of this environment is given in Fig. 1. The figure illustrates that there is no strict tree-like hierarchy but that instead various forms of overlap can be found. The properties marked by the gray area characterize one object and may be termed basic-level concepts, the other items situated outside the gray areas are called subordinate concepts. Items that in the training situations are replaced by each other forming different training vectors are connected.
by dashed arrows.

How can this environmental structure be represented in neuronal terms, i.e. how can the information be learned and how can it be used for retrieval?

B. The Network

To be able to represent the three properties (4-legged, surface texture, color) we reserved four units for each, requiring 12 units in total. As each object further is characterized by an identifier and a name, we require eight further units for the four objects. In addition we have seven superordinate concepts, each of which is represented by one neural unit, leading to a network containing 27 units in total (see Fig. 2).

As lower level units should be able to activate higher level units, but also in turn, higher level units may activate lower level units, we require recurrent connections between all units.

Using a recurrent network, of course, raises stability problems. Therefore, a fundamental requirement for such a system being functional is that only a limited number of units should be activated after any external input has been provided. Second, if possible, the activity should not grow unlimited (this requirement is less critical because it could be solved by introduction of neurons with bounded characteristics as has been done by [1]). Earlier studies have shown that these requirements can be fulfilled by using RNN consisting of so called input compensation (IC) units [2], [3]. Here in particular we apply the version termed Suppression (Su) unit [3].

Such an IC unit has the following specific properties [2], [3]. As in traditional recurrent networks, a weighted sum of the recurrent inputs \( s_i = \sum_j w_{ij} x_j(t) \) is determined. Each unit also has an external input \( I_i(t) \). The output of the unit is given by

\[
x_i(t+1) = \begin{cases} I_i(t) & \text{if } I_i(t) > 0 \\ s_i(t) & \text{otherwise} \end{cases}
\]

Furthermore,

\[
x_i(t) = \begin{cases} x_i(t) & \text{if } x_i(t) \geq 0 \\ 0 & \text{otherwise} \end{cases}
\]

forming the nonlinear characteristic of a rectifier meaning that there is no upper bound. For learning, an error value is computed locally by \( \delta_i(t) = I_i(t) - s_i(t) \) and used to train the weights of unit \( i \) according to

\[
w_{ij}(t+1) = w_{ij}(t) + \epsilon \delta_i(t) x_j(t) \tag{1}
\]

where \( \epsilon > 0 \) is the learning rate.

Learning is performed by providing the input vector and waiting for two iterations per learning step (\( \epsilon = 0.9 \)). Makarov et al. [3] have shown that, using this type of RNN and this learning rule, the stable values do not depend on the temporal order of the input vectors given, provided each vector is given often enough.

As the weights approach their limit values asymptotically and as these values are not easy to be calculated, we used the following simple approximation providing one number describing the learning state of the complete weight matrix (or parts of it). Results have shown that the total sum of absolute weights changes by less than 0.001 after 300 training epochs (see Table I, an epoch consisting of the set of different training vectors used in the corresponding task). Therefore, as an approximate upper limit for the weights we used those values that have been obtained after 3000 epochs.

This value is used as a reference to describe the learning state of the network. The diagonal weights are fixed to a value of 3. During learning, all weights are continuously

<table>
<thead>
<tr>
<th>epoch</th>
<th>positive weights</th>
<th>negative weights</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>111.05</td>
<td>-11.34</td>
<td>122.38</td>
</tr>
<tr>
<td>3</td>
<td>121.65</td>
<td>-31.19</td>
<td>152.85</td>
</tr>
<tr>
<td>15</td>
<td>131.83</td>
<td>-62.85</td>
<td>194.69</td>
</tr>
<tr>
<td>30</td>
<td>132.26</td>
<td>-66.76</td>
<td>199.02</td>
</tr>
<tr>
<td>300</td>
<td>132.43</td>
<td>-67.45</td>
<td>199.88</td>
</tr>
<tr>
<td>3000</td>
<td>132.43</td>
<td>-67.45</td>
<td>199.88</td>
</tr>
</tbody>
</table>

![Fig. 1. Illustration of the four objects and how the training vectors are constructed. Basically the four vectors marked in gray represent the four objects. The dashed arrows connect superordinate items with those basic level items that they are replacing to form new training vectors.](image-url)
corrected by multiplication with a factor of 0.25. This expansion provides the individual unit with the property of a low-pass filter with a time constant of 2 iterations [3]. Note that in Fig. 2 and the tables weights are given without this correction by a factor 0.25.

The three properties 4-legged, surface texture and color are represented by a 4-component subvector each. Application of non-orthogonal vectors allows representing pairs of objects being more related to each other than to other objects. For example, the color of two cows Frieda and Emma may be more similar than the color of a cow and the horse. For simplicity we started with orthogonal vectors with only one component being 1, the other three being zero. However, using non-orthogonal vectors showed comparable results.

As will be explained below (Sect. III) we studied different tasks using different input vectors for training. As largest number we used 22 input vectors. After training is finished, the properties of the network are tested by stimulating a single unit using an external input for four iterations. As our network is here used as an autoassociator, the output will show the stimulated unit being active plus other units that are associated with this unit.

III. RESULTS

A. Task 1: Four objects, no categories

As a first task, we present only the four objects, i.e. four training vectors, using the properties in the gray area in Fig. 1. This means we apply the following four vectors

\[ (1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0) \]

\[ (0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0) \]

\[ (0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0) \]

\[ (0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0) \]

The meaning of the individual components is given in Fig. 2. For example, the first vector corresponds to the situation (Identifier1, 4-legged1, surf1, colored1, Frieda). Storing such vectors may be considered as storing episodic memory content. In this task, two learning steps for each object, i.e. two epochs, are sufficient to reach weight values of more than 99% of the final weights (67% after only one epoch). When, in the retrieval tests, one of the units is stimulated by an external input (to any of its five properties) for four iterations, all units representing this object, and only these, will reach a stable state (99% final activity) within three iteration after the stimulus has been switched off. The units remain active at this level, after the input is switched off. In other words, any element of the object is sufficient to activate the memory of the complete object.

B. Task 2: Four objects plus categories cow, animal

Abstract, or superordinate categories are introduced by not only providing a vector for the individual objects as has been done in task 1, but, in task 2a, also providing the “cow” vectors

\[ (1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0) \]

\[ (0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0) \]

As mentioned, this approach is inspired by the idea that children are generally not trained by explicitly telling them ‘Frieda is a cow’, and ‘Emma is a cow’, but instead because when talking with children the first object is sometimes called Frieda, and sometimes called Cow. Training of these 6 vectors required 10 epochs to reach 99% of final weight values.

In the corresponding way, in task 2b we trained the term ‘Animal’ by applying in addition the training vectors

\[ (1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0) \]

\[ (0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0) \]

\[ (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0) \]

\[ (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0) \]

We will discuss results only after a third task has been explained.

C. Task 3: More superordinate categories

In the first task, only four objects were presented. In the second task, higher-level categories (cow, animal) have been given requiring 6, or 9, input vectors. Now, in the third task, more categories are applied as illustrated in Fig. 1 leading to 22 input vectors altogether. In addition to the 9 vectors already used in task 2b, four further vectors are introduced by correspondingly replacing the “4-legged” components by a unit representing the word “Fur”:

\[ (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0) \]

\[ (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0) \]

\[ (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0) \]

Three further vectors were given by again providing the data of the three animals but with the surface texture components being replaced by the word “Fur”:

\[ (1,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0) \]

\[ (0,0,0,0,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0) \]

\[ (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0) \]

Two other vectors consist of the data of both cows, where the color subvector was replaced by the word “Colored”:

\[ (1,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0) \]

\[ (0,0,0,0,0,1,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0) \]

For another two vectors, the data of both cows were given and the names replaced by the word “Name”:

\[ (1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0) \]

\[ (0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0) \]

Finally, the data of the horse and the table were given and in

<table>
<thead>
<tr>
<th>Negative weights</th>
<th>Positive weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weight distribution after 300 epochs</td>
<td></td>
</tr>
<tr>
<td>&lt;.99</td>
<td>&lt;.9</td>
</tr>
<tr>
<td>42</td>
<td>0</td>
</tr>
</tbody>
</table>
both cases the color subvector was replaced by the word "Uni":

\[(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1)\]

In total this leads to 22 input vectors forming one epoch that have to be learned by the network consisting of 27 units.

**Weight Distribution**

The weight matrix obtained after 30 epochs is shown in Fig. 2 using a Hinton plot. Weights belonging to the unit characterized by the left hand term are arranged in the corresponding horizontal line. These weights are trained by the error value determined for that unit (see equation (1)). Correspondingly, weights arranged in a row characterize the output effects of the unit given by the term in the top line onto all other units. The size of the weight is depicted by the size of the square, white squares showing positive weights, black squares showing negative ones. Weights at the diagonal are not shown, because they were kept constant during training (see Methods). The units in this matrix are arranged in a functional order, i.e., in such a way that units, which belong to one object, are shown as being neighbored.

This has been done here for better readability. However, a re-arrangement is of course easily possible that shows an order better comparable with a modular structure as found in brains. Following this map-like structure, all ‘color’ units, all ‘fur’-units, all 4-legged-shape units etc. would be arranged as neighbors forming local modules. Superordinate units might still be separated morphologically as shown in Fig. 2 or might be part of the respective modules. Using such a map-like structure shows more clearly that weights connecting units within a module are inhibitory, whereas intermodular weights are positive. Weights connecting related superordinate and basic level units are negative.

To illustrate the general development of the weights, the sum of all positive weights, the sum of all negative weights and the total sum of absolute weights is shown in Table I for 1, 3, 15, 30, 300 and 3000 epochs. (Diagonal weights which are not learned but held fixed during training are not considered). As can be seen, after 15 epochs weights reach a total of about 97 % of the final values.

After training is finished (e.g. 300 training steps for each vector), the following results can be obtained (Table II). Many weights (134 of 702) are about zero (i.e. -0.01 < x < 0.01), but most of the weights (344) are negative (< -0.01), many (42) of them showing a value of about -1, (between -.99 and -1.04, see large black squares in Fig. 2). A large percentage (287) shows a value between -0.2 and -0.01. Altogether 224 weights have a positive sign with a value > 0.01, but all weights being smaller than 0.5. Many positive weights (198) show values between 0.1 and 0.3.

![Fig. 2](image-url) Weight matrix obtained after 30 epochs. Diagonal weights are marked by letter D. The arrangement of weights is chosen such that functional clusters can easily be recognized. Positive weights: white squares, negative weights: black
99.5 % of the final values, the following qualitative properties can be observed concerning the “morphological” distribution of the weights. Between basic-level units that belong to the same object large positive weights (0.12 to 0.45) are found. Weights which connect identifier units with the properties of their own object are positive and tend to be larger than the other weights. Small (-0.01 to -0.14) negative weights occur between basic-level units belonging to different objects. Superordinate units show positive weights (0.12 to 0.35) with units of related basic-level concepts. There are (in part strong) negative weights (0 to -7.2) between superordinate units. Perhaps counter-intuitively, these weights are the more negative the more the units are related (see extreme cases Cow – Frieda, Cow – Animal), i.e. the larger the overlap between these unit (less extreme but high, too: Colored – Name or 4-Legged – Fur).

The strongest weights are negative and are typically -1. Such weights occur if there are training vectors with overlapping components, for example vectors

\((1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)\)

and

\((1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0)\)

where component Frieda is replaced by component Cow. In these situations, the non-overlapping components, in this example unit Frieda and unit Cow, show strong mutual inhibition. This refers to those units that in Fig. 1 are connected by dashed lines, but such large negative weights can also be found between units Cow and Animal (see Sect. III, E for discussion of functional aspects). Learning of the mutually inhibitory weights between any two vectors was not possible if we had presented first one vector several times followed by the second vector being presented for several times, too. This procedure would result in establishing inhibitory weights only from the second vector to the first one, but not in the opposite direction. Therefore, mutual inhibition can only be learned if each vector happens to be preceded by its ‘opponent’ vector. In other words, a strict serially ordered training procedure would not lead to the results presented here.

Another result concerns the observation that weights can be arranged in groups that show different learning dynamics: Positive weights between objects are learned fast. They are on average very near the final value already after the first epoch (Table III). Negative weights between objects are developing slower. This means that learning to separate two objects takes more time than learning the two objects as such, although all information is given at the same time. The weights that connect superordinate units grow even slower.

**Behavior of the trained network**

In the following we show the properties of the network when one unit is stimulated externally. As a general result, we observe, after relaxation is finished, basic level units being more active than are superordinate units (between 2.2 and 4.5 or between 0.7 and 2.2, respectively).

More specifically, testing the network by external stimulation of a unit of one object (e.g. Frieda, or Identifier1) leads to a strong and stable activation of all five components characterizing this object. But there is also a spreading of activation to superordinate concept units. In this case the corresponding “local” component is less strongly activated. For example, the input

\((1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)\)

leads to the following output:

\((4.5, 4.5, 4.5, 4.5, 3.2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1.2, 0)\)

Note that activation of the three units, Frieda (3.2), Name (1.2) and Animal (0.1) sums up to 4.5, too. The superordinate units are not activated immediately, but require some iterations, during which the local unit decreases and the concept unit increases. Together with the above mentioned observation that basic level units show a higher activation, these results indicate that activation of superordinate concept representations are more difficult (in terms of time and of excitation strength) than are units of basic-level concepts. This agrees with the finding that human subjects are faster in naming basic-level concepts than superordinate concepts [9].

If the unit Cow is stimulated, in addition to this unit all components of both cows except the two name units (Frieda, Emma) are activated. Corresponding results are found for Colored, Name, 4-Legged and Uni. This result can be interpreted such that a concept is realized by activation of all related units belonging to the corresponding basic-level concept.

**D. Distributed, non-orthogonal property vectors**

For the experiments shown above, all property vectors were assumed to be orthogonal. To relax this condition, in separate investigations the vectors for surf1 and surf2 are assumed to be non orthogonal. This allows to represent that the fur of one cow shows more overlap with the fur of another cow compared to the fur of a horse, for example. In another simulation, all three properties of the two cows (4-legged, surface, color) are chosen to be non orthogonal. Qualitatively, results correspond to those obtained in the orthogonal cases.

If, after training is finished, Identifier1 is stimulated, all properties of this object are active, of course now showing a two component subvector, plus, however weaker, that of the superordinate units of 4-Legged, Fur, Colored and Name (lower line of Fig. 1). As in the orthogonal case, over time, one superordinate concept unit may receive more or all of the activations of its basic-level units. If a superordinate concept, for example the unit Cow, is stimulated, the subvectors representing the properties show a mixed values of both cows, which might be interpreted as a kind of prototype for “cow-typical color” or “cow-typical fur”.

**E. Delayed Switching off the Stimulus**

As mentioned in Sect. III.C., several units are connected via inhibitory weights of different strength. These connections have the effect of producing distributed, more or
less strong winner-take-all structures. This connectivity supports the stability of the network, but leads to the following drawback. Due to this soft WTA connectivity, unit Cow, for example, inhibits units Frieda, Animal etc., which means that if in the test unit Cow is stimulated, the other superordinate units Animal, 4-Legged, Fur, Colored and Name, although related, will not be activated. However, intuitively one would assume that a human subject, given the word Cow, would associate this input not only with the properties of all cows, but also with the related superordinate concepts like Animal, Fur, etc.. This situation could not be improved by application of the following two other training procedures. We tried to teach the hierarchical information in a direct way by offering training vectors like (Fur, surf1), (Fur, surf2), (Fur, surf3) in order to elicit a positive connection between a category and the related examples. Alternatively, we expanded the vector by using the name and the category at the same time. In both cases we did not receive weight matrices that showed stable solutions in the later tests.

The behavior of the network can however be improved by the following simple variation of the stimulation procedure applied in the retrieval test. The stimulus is not being switched off completely, but instead a small value of the input (e.g., 0.1) is provided for a short period (e.g., another 4 iterations). Using this procedure of delayed switching off the stimulus, stimulation of Cow will also lead to a transient activation of all units belonging to this item at the first iteration after the external input has been switched off. For the priming we tested two slightly different procedures. The second stimulus was either given after the first one had 7 iterations (procedure A) or 11 iterations (procedure B) for relaxation.

If either Frieda or Cow was given alone the total activation was 8.2 or 8.1, respectively. If Frieda was the primer and Cow was given after the stimulus Frieda, the total activation was 13.7 in procedure A and 14.3 in procedure B. When instead Cow was used as a primer and Frieda was given second, in case of procedure A the total activation was 12.4, in case of procedure B it was 11.9.

Thus, priming was found in all cases. Furthermore, priming by a basic-level term was more effective than priming by a superordinate term, as has been observed in the above mentioned studies with human subjects. The effect was more obvious when the first stimulus was given more time to relax (i.e., procedure B). In another test we compared the priming effect from Frieda to Emma with the effect from Frieda to Table using procedure B. There was practically no effect from Frieda to Emma (numerically from 8.2 to 8.1), whereas an inhibitory effect could be observed when Frieda was used to prime Table (from 10.7 to 9.2). So, priming was less (indeed negative) when the items were less tightly connected. We are not aware of corresponding results obtained with human subjects.

G. Chunking by naming

A crucial implicit assumption in our and most other such training studies including O’Connor et al. [1] is that the agent shows a property that at first sight appears to be obvious but is not. When the agent is experiencing a specific object, say Frieda, a second time again, it is assumed that the agent recognizes this object as being the same. This faculty is by no means trivial because the object may be seen from a different perspective, a different angle, a different illumination, and therefore the stimuli might physically be quite different. But even if it looked the same, the problem remains. How should one know that it is the same object and not a different one looking similar? More operationally, as episodic memory contains a measure of (any kind of absolute) time, at least this component is different, even for the same object. How is it possible to recognize the different stimuli as representing the same object? One solution might be that there is a supervisor who gives the same name (e.g. Frieda) to the different stimuli.

In the following we test whether this problem could be solved within our framework. To this end, we use the same network and the same learning structure. Therefore we use four objects for this task, but now instead of Frieda, Emma and Horse three versions, or aspects, of cow Frieda, and still the table. This means that the components Identifier, 4-legged, surf and color are different for the three cows seen, but they all have the same name Frieda. In other words, we use the training vectors

\[
(1,1,1,1, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0,0,0)
\]

\[
(0,0,0,1, 1,1,1,1, 0,0,0,0, 0,0,0,0, 0,0,0,0,0,0)
\]

\[
(0,0,0,0,1, 0,0,0,0, 1,1,1,1, 0,0,0,0, 0,0,0,0,0,0)
\]

and, as earlier, the object table:

\[
(0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0, 1,1,1,1,1, 0,0,0,0,0,0)
\]

Superordinate concepts as Cow, Animal, 4-Legged are applied as above, too. Note that the three different Identifiers may represent different contextual aspects including some
After training (30 epochs using 20 training vectors and 27 units) we obtain the following results. If any identifier of the three aspects of Frieda is stimulated, the vector of the four properties of all three aspects plus the name unit (Frieda) are activated, in addition with units Cow and Animal. Stimulation of any other property of the cow vector leads to the corresponding result.

If Frieda is stimulated, correspondingly all the aspects of Frieda including, of course, unit Frieda, are activated. As earlier, superordinate vectors might take over activity from basic level units. Units Cow or Animal do not show up (but become activated when using the delayed switching off the input version, Sect. III.E).

Thus, using our training procedure, different episodic situations can be combined, i.e. chunked in the memory, if a supervisor characterized them with the same name. But nevertheless, the individual episodic memory could be recalled as well.

IV. DISCUSSION

In this article, we simulate a memory system that represents memory elements of different hierarchical levels, eventually termed superordinate concepts, basic-level concepts and subordinate level elements [4], but without explicitly implementing a hierarchical structure into the network. In doing so, we follow the ideas of O'Connor et al. [1]. These authors argue that, although on the behavioral level differences between low-level concepts and higher level concepts can be found, these observations do not necessarily mean that this hierarchy has to be reflected in an explicit hierarchical structure in the corresponding neuronal representation, nor that specific learning procedures are required for basic-level concepts and superordinate concepts. In contrast, both models show that no specific different treatment is necessary for the models to show a behavior that suggests a hierarchical structure.

As a main difference to the approach of O'Connor et al. [1] we use a RNN with specific units which allow for a much simpler learning rule. Whereas O'Connor et al. [1] apply the continuous recurrent backpropagation-through-time algorithm, which is not easily to be implemented in a realistic neural network, we use a simple teacher forcing method [2] and a local version of the delta rule [11] that can be applied within the individual neuron and does not require any additional elements necessary for backpropagation of the error. To simplify the investigation, as a proof of concept we use a smaller set of concepts and features compared to their study (7 instead of 20 superordinate terms and 20 instead of 541 basic-level concepts). This simplification makes it possible to investigate the development of individual weights not studied by O'Connor et al. [1].

Different to our approach, O'Connor et al. [1] used supervised learning which requires a separation of input layer and output layer, i.e. the target vector. Instead, we use the same layer for both input and target vector. Therefore, during training there is no mechanism required telling the network which is input and which is target. A related difference is that we train static attractors representing the vector representing input and output to be used as an autoassociator, whereas O'Connor et al. [1] use the network to work as a predictor.

In our network, training is fast: If only objects as such have to be learned, two presentations of each object are sufficient. If all situations depicted in Fig. 1, i.e., 27 items with partly overlapping hierarchies (corresponding to 22 input vectors) should be learned, 15 presentations of each vector are sufficient to reach 97% of final weight values. Different groups of weights show different learning dynamics. Learning velocity in the approach of O'Connor et al. [1] is described such that after 150 epochs 95% of the units reach 80% of ideal activation level. However, these values cannot sensibly be compared with ours, because O'Connor et al. used a much larger network. Most weights are zero or negative specific connections showing the highest absolute values. In neurons, excitatory synapses tend to be located at the dendrites, whereas inhibitory synapses are concentrated at the cell body where they have stronger influence. This difference may be realized in our simulation by the fact that specific inhibitory connections show high negative values.

Testing the behavior of the trained network revealed that all individual objects and all superordinate concepts could be retrieved without cross talk, while the expected hierarchical relations are illustrated without explicit training and no hierarchical structures being preinstalled. Although treated identically, superordinate and basic-level concepts show different activations in the retrieval task both with respect to activation level and temporal dynamics. These results, as mentioned, agree with findings from human subjects.

The model allows for showing temporal dynamics for both top-down activation of basic level concepts and for bottom-up activation of superordinate concepts. Exploiting this property, it was possible to simulate bottom-up and top-down priming. When testing the hidden hierarchies, spreading of activation was always possible in top-down direction. Bottom-up activation was not possible in some cases, but application of specific dynamics of the stimulus (delayed switching off the stimulus, Sect III.E) improved the situation. Generally, activation strength of superordinate units was smaller (about half of) activation of basic-level units. According to O'Connor et al. [1, p. 298], this result may be compared with the observation that in humans superordinate concepts are less imaginable and more abstract.

O'Connor et al. [1] argue, and use as an important aspect for their training method, that higher level expressions are less often used in real world and are therefore experienced with a lower frequency than are lower-level concepts. In contrast, we presented all vectors with the same frequency, because in our learning procedure, given long enough training, results are independent of frequency [3]. Therefore,
we did not, different to O’Connor et al. [1], apply aspects of familiarity or typicality, both representing some kind of word frequency effect. Our results show that frequency is not necessary to provide hierarchical memory structures. In particular, we observed that superordinate concepts are learned slower, independent of frequency. Therefore, variation of frequency might not be relevant to explain the different learning velocities.

However, we could not find effects of typicality in contrast to O’Connor et al. [1]. Typicality means that some objects are more ‘typical’ for a given category than are other objects of that category. For example, a blackbird might be regarded a more typical bird than a penguin. Typicality is assumed to depend on frequency. Therefore, in our framework, cows Frieda and Emma should be more typical animals or more typical “fur-bearing animals” than the horse, because both cows are more similar to each other. Similarly, the cows should be more typical “4-legged objects” than is horse or table. Therefore, when stimulating one of these higher-level concepts, one would expect that the more typical object is stronger excited. This was however often not the case, neither in the orthogonal version nor in the fully distributed one, in contrast to results reported by O’Connor et al. [1]. This difference may be due to the fact that we used, to keep the network as simple as possible, units with (semi-) linear characteristics (i.e. linear ones apart from rectifier properties). If we applied the often used tanh function, for example, such frequency effects might occur, too.

O’Connor et al. [1] operate on the level of verbally described features for which appropriate detectors are assumed to be given. However, the framework can also be applied in a more general way. For the word units, we, too, apply feature detectors, which might be realized as RNN that represent the individual words (but these networks are not shown here). Beyond that, for the basic-level concepts apart from names, we instead assume a simple system that is only equipped with object recognizers (i.e. the capability of figure-ground separation) and sensors, but no specific feature detectors as, for example, a “fur-detector”. Rather we assume neural networks that monitor physical properties like color, shape, tactile stimulation etc. Thereby, the representations can be grounded and thus, in principle, be applied to an autonomous robot. In particular, the extended version of the insect-based network Walknet offers an appropriate framework [7], [12].

In our first approach (Sect. III, A - C), we assumed that the physical properties may vary from object to object, i.e. from, for example, cow Frieda to cow Emma, but are assumed invariant if the same object is seen later again. In the second approach (Sect. III, G), we however realize that the sensory input may even vary for a given object. The results have shown that also in this case chunking is indeed possible. Such chunking could already be observed in 10 month old infants [13].

In our case, the adequate information required for chunking is given by a supervisor. However, other possibilities not implemented here could be imagined. For example, the agent could be equipped with the ability to actively search for a spontaneous hypothesis on how to chunk sensory events being similar enough. These hypotheses have then to be tested whether they allow for sensible sensory events interacting with the world, for example using communicative success (as has been studied by Steels [14]). Of course, the development is faster if the concepts are provided by an already experienced supervisor as assumed here.

REFERENCES