VRIJE
UNIVERSITEIT
BRUSSEL

Master thesis submitted in partial fulfilment of the requirements for the degree of
Master of Science in Applied Sciences and Engineering: Computer Science

USING RESTRICTED BOLTZMANN MACHINES IN AN AGENT-BASED MODEL TO STUDY LANGUAGE CHANGE

Joni Muyshondt

January 2022

Promotor: Prof. Dr. Bart de Boer
Advisor: Peter Dekker
Science and Bio-Engineering Sciences

VRIJE
UNIVERSITEIT
BRUSSEL

Proefschrift ingediend met het oog op het behalen van de graad van Master of Science in de Ingenieurswetenschappen: Computerwetenschappen

RESTRICTED BOLTZMANN MACHINES GEBRUIKEN IN EEN AGENT-GEBASEERD MODEL OM TAALVERANDERING TE ONDERZOEKEN

Joni Muyshondt

januari 2022

Promotor: Prof. Dr. Bart de Boer
Begeleider: Peter Dekker
Wetenschappen en Bio-ingenieurswetenschappen

Acknowledgement

First and foremost I am extremely grateful to my supervisor, Peter Dekker for all the advice, time, and patience. Secondly, I would like to thank my promotor, Prof. Dr. Bart de Boer who overlooked the project from a distance and steered me in a better direction when needed. Their knowledge and experience helped me to proceed with a better view on the topics. It also lead to new ideas that could be tried and remarks to keep in mind.

In order to complete this thesis I used the knowledge gathered during my studies. Therefore, I would also like to thank all my past professors for expanding my mind.

Finally, I would like to thank my family and friends for all their support during this period. They provided the needed distractions but also motivated me by working together.

Abstract

In this thesis, the possibility to use a Restricted Boltzmann machine (RBM), to represent the production and comprehension model of an agent in an agent-based model, is studied. With these models, language change could be simulated. This is useful to get a better understanding in the way languages change and even simulate it while it happens. RBMs are used because they provide a nice feature which enables an agent to both speak and listen. The same RBM is used to produce a signal given a concept and to produce a concept given a signal. Classic neural networks only go in one direction, from input to output. In one model, Language games are played between agents using RBMs. Both a population agent-based model and an iterated learning agent-based model are used here.

First, good values for the parameters of the RBM should be found before it can be used to simulate communication well. Experiments were done varying the number of hidden units, the initial values for the weighs and biases, the learning algorithm and learning rate, and the number of loops through the training data. For each experiment the corresponding communicative success is calculated. No obvious successful parameter setting could be concluded after the experiments. Overall the communicative success was also low, only for less than half of the interactions between agents, the listener gave back the correct concept. Therefore this thesis can not yet show that an RBM could be used to investigate language change using agent-based models. However, we now have a better understanding of the behaviour of the parameters in this communication setting. Using this information, other settings could be exploited which may lead to more promising results.

Contents

1 Introduction

The following will be investigated: is it possible to use a neural model, the Boltzmann Machine, to simulate language change, more specifically morphological simplification? Insight is needed about how the neural network works and about the way languages change. Therefore, two disciplines are combined for the research in this thesis: linguistics and computer science.

First some information is given about ways in which a language can change and how this may arise. One kind of language change is discussed in more detail: morphological simplification. It happened for instance in the Alorese language, which is the case study on which the research question is partly inspired. A useful tool to investigate this language change with, is through computer models. More particularly, agent-based models are used to simulate the interactions between people so we do not need years of observing actual people. To be able to simulate morphological simplification, we need a way to make a person able to both talk and listen. This is were the neural network, the Restricted Boltzmann machine (RBM), comes in. It is chosen because it works in two ways, so it does not only go from input to output. Other reasons are that it is somewhat neurologically plausible and with the restricted variant it is usable in practice. Each person in the simulation is represented by such an RBM. The goal is to study whether it is possible to investigate language change using an RBM, and under which of its conditions it would be possible.

1.1 Language change

Languages evolve over time and it can take a while for a language change to settle, often a couple of generations (Aitchison 2001). There are many possible changes that can occur. Some examples are: words could get a different meaning, the pronunciation of words could change, or verbs could get conjugated differently. Different causes can give rise to these and other kind of changes (Labov 2011). Things could happen within one language community. For example the teaching habits of parents could change through generations, or a certain social group becomes bigger. Another possibility is that communities came in contact with each other, for example by migration. Then, habits and ideas may be interchanged among other things on language level.

Historical linguistics is the study that looks at language change over time (Bynon 1977).A part of the study involves trying to recreate the history of languages and to explain why and how changes happened. This is tried here for the Alorese language. However, the study involves much more that is not applied in this thesis, like investigating how languages are related for example. It should not be confused with evolutionary linguistics, which looks at the biological origin of languages. A big part of this is looking at how our species transitioned to use our current communication (Croft 2008). In this paper, the changes of languages are studied with respect to human interactions, not the evolution of the brain to understand a language.

Studying language change demands a lot of data and observation over a large period of time. Often, this data is not available. Computer simulations, or more specificaly, agent-based models can help understand languages, which saves time and could be beneficial when data is missing (Ke et al. 2008). An extra advantages is for example that language changes can be simulated as they are taking place.

1.2 Morphological simplification

A change that could occur in a language is morphological simplification. In linguistics, morphology is the study of words and its internal structure (Kolenchery 2015). The smallest piece of a word with a meaning is a morpheme, which can be the word or a part of it (Radford et al.

2009). Two types of morphemes exist. A free morpheme has a meaning on its own. A bound morpheme should be combined with (an)other morpheme(s) before there is a meaning. The latter are mostly affixes. In English, 'talk' is an example of a free morpheme. An example of a bound morpheme is the suffix '-s'. Together with the former one, this can be used to express 'he talks'. If interested, Radford et al. give the different kind of morphemes in more detail in part 2: Words (2009).

A morphological simplification means that some morpheme(s) disappeared or simplified over time (Kusters 2000). Many things could cause this kind of change but Atkinson, Smith & Kirby (2018) present a theory that language's morphology tends to simplify if it gets many adult learners. It goes together with a theory which states that learning on a later age tends to be more challenging.

1.3 Case study: Alorese

In the Austronesian language Alorese, spoken on the Alor and Pantar islands, morphological simplifications were described by Klamer (2011). One specific simplification was chosen as inspiration for the research in this paper.

In English, verbs are conjugated differently for different persons, and the same applies for the Alorese language. However, for the latter a prefix is added for the verb and not a suffix as with English. After some research was done, this seems to be a simplification and, prior, adding suffixes after verbs was also done (Moro 2019).

Lamaholot is the closest genealogical relative of Alorese, and here the prefixes are still used (Nishiyama and Kelen 2007). Proto-Lamaholot is the shared ancestor of Alorese and Lamaholot and a simplification took place after Alorese split from Lamaholot and came in contact with the neighbouring Papuan languages (the languages on Alor and Pantar). The latter are not genealogically related to the Alorese languages (Klamer 2012).

In Figure 1 the regional context of Alorese is given.

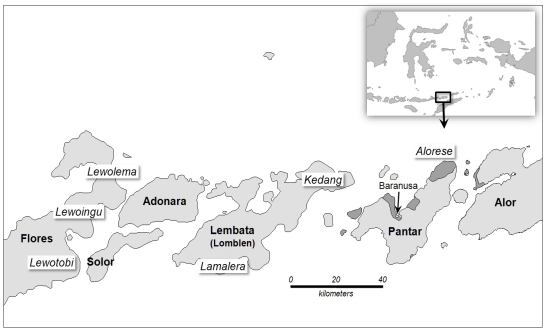


Figure 1: Alorese (dark grey areas) and Lamaholot varieties as spoken on Flores, Solor, Adonara and Lembata (Klamer 2012)

To simulate the simplification of Alorese, the closest genealogical relative is used as a starting point. The agents, representing the people, in the model are initialised with the Lamaholot language, or in practice an abstracted form of this language. From then on, due to interactions between the people, a simplification such as with Alorese is tried to be simulated. Different mechanisms are tested to find out if mechanisms leading to the current simulation of the Alorese language exist and if so to describe them.

Originally, interactions would have been simulated with the original residents only, and a combination with people speaking other languages. After a lot of interactions the resulting

language would have been investigated. The other speaking people migrated due to marriage or work and this resulted in more adult language learners (Moro 2019). A theory is that it is more challenging to learn a language when you are older (Lenneberg 1967; Krashen 1982), which could explain the simplification.

It is important to mention that the people speaking another language than Lamaholot were not represented in the model discussed in this paper. However, it was explained to clarify why this case study was used as inspiration to study morphological simplification.

The table below represents the bound morphemes that are put before or after a verb for different persons. The prefixes are used for the prefixing verbs, and the suffixes for the suffixing verbs. Which verbs are which is fixed and a bigger part of the verbs are suffixing verbs.

	Lamaholot (Lewoingu)		Alorese
	A prefix (on 20 verbs)	S Suffix	A prefix (on 8 verbs)
1SG	*k-*	*-kən*	*k-*
2SG	*m-*	*-ko, -no*	*m-*
3SG	*n-*	*-na, -nən*	*n-*
1PL.EXCL	*m-*	*-kən*	*m-*
1PL.INCL	*t-*	*-te*	*t-*
2PL	*m-*	*-ke/-ne*	*m-*
3PL	*r-*	*-ka*	*r-*

Table 1: verb affixes in Lamaholot and Alorese (Moro 2019)

Some examples are given to clarify (Nishiyama and Kelen 2007). In Lamaholot '-enun' means 'drink'. It is a prefixing verb so when looking at the table this becomes 'kenun' for the first person singular. The suffixing verb 'live' is translated as 'ia'-' and becomes 'ia'kən' for the first person singular. There is also the verb 'go' ('-a'i-'), to which both a prefix and a suffix is assigned.

This data was used as inspiration for the data used by the model discussed in this paper, but in reality an abstracted form of it was used. The verbs are not represented, but only the different persons which are connected to the prefixes/suffixes as in Table 1. Prefixes and suffixes are also viewed as one whole and phonology, which looks at speech, is not considered. Then, for example, no conclusion could be tried to be made when a prefix ends with a vowel, and the verb begins with a consonant. This would have made less sense because no specific verb is considered. The fact that a verb is transitive and/or intransitive is therefore also not included in the model. As a last abstraction, the irregular verbs present in Lamaholot are not included in the model. The same goes for the verb 'go' which uses both a prefix and a suffix to conjugate (Nishiyama and Kelen 2007).

1.4 Agent-based models

An agent-based model is used to observe the overall effect that takes place when letting individuals, or agents, interact with each other (Gilbert 2019). In this paper the agents are represented by people, and from now on we will always use the term agent when talking about an individual person. However, an agent can be any individual entity which interacts in a group: an animal, a cell in the body, an atom, etc..

There already exist applications of agent-based models in historical linguistics. A walk through of designing an agent-based model for historical linguistics can be found, using an example in Dutch about the competition between regular and irregular verb inflection (Pijpops and Beuls 2014). Another paper uses an agent-based model to predict that 'to have' grammaticalized

faster in English, and that subordinate clauses gained use more quickly in German.(Bloem et al. 2015)

Building an agent-based model includes: representing agents, providing an environment for the agents, and finding a way to let the agents communicate. All the agents in the model usually have the same goal, which in the case of language, is usually to understand each other. The way two people communicate is depicted in Figure 2, which is known as a language game (Steels 1997).

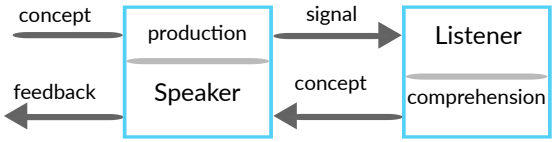


Figure 2: communication diagram between a speaker and listener

In the diagram, two agents are represented: a speaker and a listener. A speaker tries to communicate a concept in the world to the listener, and does this by producing an utterance, or a spoken word. The listener receives this utterance, through perception and tries to link it to a concept in the world. In the rest of the paper, concept and signal are used to represent the concept and utterance (Gilbert 2008).

Each person has its own internal representation of a language because it can not hold the entire language. It is not possible to exactly know the concept of another agent, this is done through the utterance and the speaker can give feedback of a correct communication when a correct reaction is given (Smith 2014).

A way to model the diagram above is through a language game. The grounded naming game is an existing language game, which simply lets the agents invent their own names for a given set of concepts (Steels and Loetzsch 2012). After some interaction they will come up with their own language, consisting of a word for each concept. This is possible through the feedback that the speaker can give. In our case study, the concept we want to represent will be the different persons. The corresponding prefix or suffix will be represented by the signal, which will be produced by a speaker and heard by a listener. One agent should be able to speak and listen to another agent. The diagram above represents 2 agents, but a population is represented by playing this little game between 2 random agents a couple of times.

A lot of agents will interact to try and simulate the morphological simplification. When comparing the data sets of languages it could be possible to get some statistical correlations between two language variables. However, this simulation could be useful to also get some insights behind the mechanisms of language change. The simulation is also time saving, it takes a lot of time to collect the needed data. Another reason why using a simulation is advantageous, is that some language hypothesis can not be tested on a real population because of practicality or they could maybe violate human rights.

Two kind of agent-based models were implemented: a population of agents and iterated learning (Smith 2014). The first one is used to simulate the interactions of the agents between one generation, and the second one between multiple generations. For the latter holds that the linguistic behavior of one generation will be the start of the internal representation of another generation (Kirby and Hurford 2002). This is also what happens when a child starts to learn a language from its parents. It is useful for looking at the learn-ability of the language.

1.5 Neural networks

There is already ongoing research on the morphological simplification of Alorese (Dekker et al. 2021), but not yet with agents represented by neural networks. However, inspiration was taken from a proposal to tackle this problem (Dekker and De Boer 2020).

The comprehension and production model of an agent in our agent-based model is represented by a neural network. It tries to achieve what happens in Figure 2. It is possible by means of the human brain and therefore neural networks are used.

The neural networks in a brain are an inspiration for the computer neural models (Gurney 2018). Neurons are represented by nodes, and the connections between them are edges which hold a number (weight). Typically, neural networks contain hidden layers, each holding some nodes and edges to other layers. The input goes through the first layer, passing the hidden layers, and an output is given by the last layer.

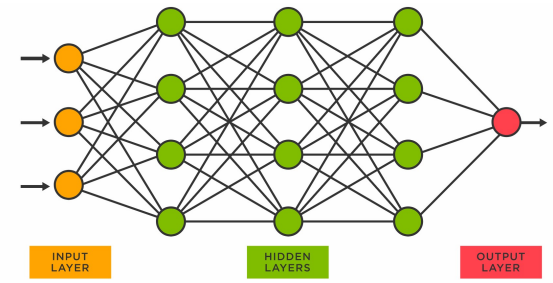


Figure 3: a neural network

The network needs to be trained, which mostly means setting the weights correctly by using back propagation. This last step is taken care off by the neural network, you just have to give it enough data to train with. Nudging the network to produce the correct output for a given input, but not dealing with how this task is learned. Gurney discusses neural networks in more detail and also presents a few applications of neural networks (2018).

Neural networks are already used a lot in different domains, also for agent-based models (Li and Bowling 2019; Cope and Schoots 2021; Ren et al. 2020; Graesser et al. 2019; Lazaridou et al. 2020; O. Tieleman et al. 2019; Chaabouni et al. 2019; Kharitonov et al. 2020; Chaabouni et al. 2021; Bait et al. 2015). These use a language game to investigate the emergent language. It is a language that a teacher starts to teach because of a new need of the student (Cope et al. 2021). A lot of research exists on emergent language and there are connections with language evolution. Here, emergent communication will also be investigated but in combination with real language change, which is still a new topic.

The goal of representing the agent as a neural network is that in the case of this thesis the network can find general rules to link the concepts that need to be communicated (persons), to the signals (prefix/suffix). Some initialisation and feedback rules should be constructed by the programmer but the internal representation takes care of how this linking is done.

A framework exists which can be used to investigate emergent language, the EGG-framework of Facebook (Kharitonov et al. 2021). It provides you with an implementation of a multi-agent communication game but also offers the freedom to use your own language, sender and receiver, loss function, and other parameter settings (Kharitonov et al. 2019). Some of the before mentioned also used this framework (Chaabouni et al. 2019; Kharitonov et al. 2020; Chaabouni et al. 2021). By trying it as a base to implement our research question, it could be concluded that it was not a right fit for this problem. While it implements a multi-agent game as we also need, it

is designed for only one speaker and one listener, each with their own architecture. Recently new code was added for making populations possible in the framework, so the problem of the single pair could have been tackled. However, for our research question it had to be possible for an agent to be both a speaker and a listener. The concept of teacher and student as with emergent language, is not completely applicable in this research. We want all the agents to communicate with each other because otherwise no simplification could be demonstrated. In the real world this also occurred after people communicated with each other.

Making it possible to let an agent speak and listen with neural networks is not straightforward because the network can not be used the other way around. The neural network has an input layer and and output layer. When an agent speaks as discussed in section 1.4, the input is a concept and it outputs a signal, and vice versa. Something exists called weight sharing (Ullrich et al. 2017; H. Pham et al. 2018). It is not yet applied to make an agent speaker and listener, but if it was it could possibly look as follows. Each agent would have to hold two networks, receiving different inputs and sending different outputs. For the information gathered by one to be transferred to the other, the weights could be shared.

Meanwhile an other solution was proposed: Boltzmann machines. The characteristic of this older neural network could be used to let an agent be a speaker and a listener.

1.6 Boltzmann Machine

This thesis uses Boltzmann machines to represent the comprehension and production model of an agent. A Boltzmann machine (BM) is a two layered neural network which has a visible layer and a hidden one (Fahlman et al. 1983; Hinton et al. 1984; Ackley et al. 1985). All nodes of both layers are connected to each other. Instead of being a deterministic model, a BM uses a stochastic approach. It tries to discover features of an unknown probability distribution using samples of this distribution, the training data (Fischer and Igel 2012). These samples are given to the network as input vectors through the visible layer. The hidden layer represents hidden features of the data which will be used to model the underlying distribution.

BMs could be useful for our research question. The concepts and their corresponding signals would be given to the visible layer. Each of them occupies a fixed number of visible nodes. Finding the underlying features of the mappings of concepts and signals, could be done with the BM. An application of BMs is that, after they are trained, they can complete partial data (Mu et al. 2015). This application suggests that it could be possible for an agent represented by a BM, to both speak and listen. The partial data would be just the concept or just the signal. If only the visible units representing the concept are turned on, the BM should be able to complete the data with the signal part, and vice versa. It is possible because a BM does not just match an output to an input but generalises over the data. In our case over the mappings between concepts and signals.

Applications of the Restricted Boltzmann machine (RBM) in linguistics already exist, also as the independently invented similar harmonium (Legendre et al. 1990; Boersma 2019). Not yet however to represent agents with RBMs to let them interact with each other, by speaking and listening.

BMss, and in particular RBMs are explained in detail in section 2. The information about the internal workings of the model was used to make certain design decisions.

1.7 Research question

The research question that will be investigated is: How can an RBM be used in an agent-based model to study language change? The goal is to use an RBM to discover features about the lan-

guage and so discover patterns. In particular, for which person to use which suffix/prefix. Here, BMs are used for another purpose than usual. Generally BMs are chosen for recommendation and recognition problems. Instead of creating a neural network from scratch, we try to use an existing one to our advantage. There is already a lot of research about using neural networks for language emergence, but less about language change.

As mentioned before, RBMs will be used to try simulating morphological simplification. The first step will be to investigate if it is possible to use RBMs to simulate communication between agents who can be speakers and listeners. This will be applied to the case study of the Alorese language, but rather using more abstracted data inspired by this language. This includes checking different circumstances of the RBM, looking for which parameters RBMs are fit to investigate language change.

Two approaches are made to investigate this. A population is made, which is done because real communication is done in a larger community and not only with 2 people. It will only be possible to simulate simplification when there is enough interaction. Secondly, iterated learning is applied because it is commonly used in evolution literature and will more show the learning point of view. It will add value because it is not often used yet with neural networks.

If this hypothesis is not rejected, more detail will be given on how an RBM could be used for the problem. The parameters of the RBM used for simulating the communication will be discussed in section 3.4. As a second step, a second null hypothesis will be introduced: It is possible to use RBMs to simulate language change after interaction between agents who can be speakers and listeners.

2 Restricted Boltzmann machine

In this section, RBMs are discussed in more detail. First, a bit of history is given to show how they evolved to computational models. The related models are also discussed in this section. From then on the focus will lay on RBMs, with a detailed explanation of its internal structure in section 2.2. It is assumed that the visible and hidden units are binary. Training an RBM and different learning algorithms for the RBM are explained in section 2.3. Lastly, for the interested reader a summary is given about Markov Random fields (MRF) and the Markov chain Monte carlo method (MCMC) because RBMs are a kind of MRFs and a part of the learning algorithms are based on the MCMC method.

2.1 History of RBM and related models

In the 1980s the Boltzmann machine (BM) was first proposed (Fahlman et al. 1983; Hinton et al. 1984; Ackley et al. 1985). A BM is a neural network with two layers: a visible layer and a hidden layer. All the visible nodes (denoted with a 'v' in Figure 4) are connected with each other, and so are the hidden ones (denoted with a 'h' in Figure 4). There is also a connection between each visible and each hidden node (Fischer and Igel 2012).

Typical neural networks, deterministic systems, are provided with an input and give back the desired output by using the back propagation algorithm (Gurney 2018). A BM does not use this algorithm, which alters the weights of the neural network when an input is given by using the feedback of the target output. There is no output used by a BM, it alters the weights when an input is given by using a loss function for the data. BMs only use input for learning and are therefore models used for unsupervised learning (Barlow 1989). The need of labeled data, as with supervised learning (Cunningham et al. 2008), is discarded and useful features of the data are tried to be found.

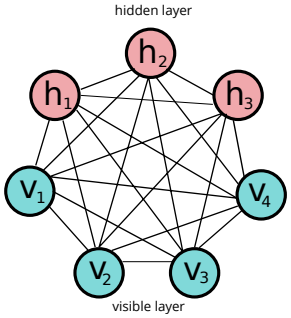


Figure 4: A Boltzmann machine with 4 visible units denoted by v and 3 hidden units denoted by h.

The data, for which the hidden features need to be discovered, is given to the visible layer. Generalizing the data into features is done in the hidden layer. Instead of trying to produce an output, the hidden layers are updated based on the input given to the visible units. Based on the hidden units the input is tried to be reconstructed in the visible units.

Something that has already been a problem many times, is the fact that a model can work theoretically but there is not enough computational power to do this in practice. For BM this was also the case, in which learning is an NP-hard problem (Bovet et al. 1994).

Finding the hidden features, means computing the maximum likelihood (Le Cam 1990) or gradient (Kelley 1962) in an undirected graphical model (Fischer and Igel 2012). The first computes the probabilities for each connection in the BM, or in other words the chance that two nodes are turned on together. This is often not possible to compute in practice and instead gradient is used to compute the highest possible likelihood. Gradient works with minimizing a loss function, which computes the difference between the distribution of the training data and the current distribution of the model you are training.Both approaches are computationally hard. By adding only one hidden unit, for each connection it forms, an extra probability needs to be computed. All the new connections also have a consequence for the number of terms that need to be computed in the loss function of the gradient.

Therefore the Restricted Boltzmann machine was proposed as a first solution (Hinton and Sejnowski 1986; Freund and Haussler 1994a). The harmonium was also proposed independently and later turned out to work the same as the RBM (Smolensky 1983; Smolensky and Riley 1984). The in-layer connections are removed as can be seen in Figure 5. This means that less probabilities needed to be computed, which is computationally better but has less expressive strength than the regular BM. However, this still turned out to be too computationally difficult until Hinton came with a more simple algorithm which made use of approximations (2012). This algorithm will be discussed in section 2.3.

Since the solutions for the computational problem were found, there were many applications of the RBM algorithm. Among others in classification (Tomczak and Zięba 2015), feature learning (Han et al. 2016), dimensionality reduction (Vrábel et al. 2020), and collaborative filtering in recommendation systems like Netflix for example (Salakhutdinov et al. 2007; Behera et al. 2021).

Until now only two-layered RBMs were considered, but there are problems for which these were not strong enough. That is why Deep belief networks (DBN) were investigated because they may generalize better and should be less prone to overfitting. Hinton introduced this DBN and drew attention to Boltzmann machines with it (2006). It also was a beginning for the current deep learning (Wang and Raj 2017). They are RBMs that are stacked and connected with each other as shown in Figure 6 (Hua et al. 2015). Every layer is connected, so the hidden nodes

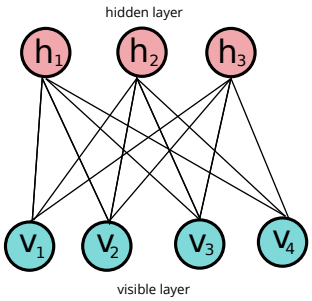


Figure 5: A Restricted Boltzmann machine with 3 hidden nodes and 4 visible nodes

become the input of the second RBM. However, there is no connection between the components of different layers. This kind of network increases the expressive power and enables the learning of features from features. It should however be taking into account that gaining expressiveness results in a computational harder problem again.

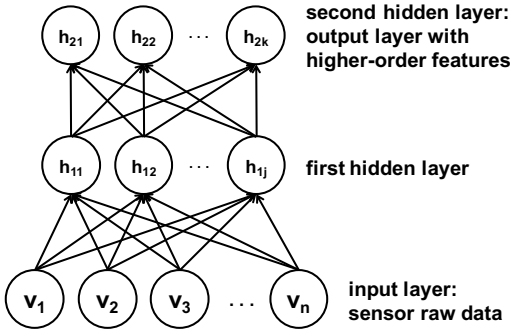


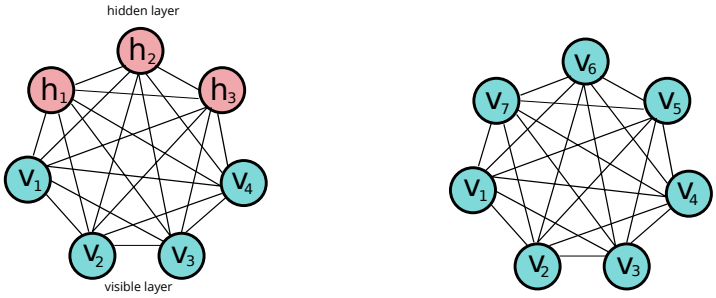
Figure 6: Deep belief network, Stacking 2 RBMs (Reimer et al. 2017)

At last, a similar network is discussed to give some insight in why BMs are used. BMs descend from Hopfield networks, and are therefore closely related (Hopfield 2007). Its nodes are binary and its connections are symmetric. In a Hopfield network, all nodes are visible as can been seen in Figure 7. A BM is kind of like a Hopfield network with a hidden layer.

The biggest reason for introducing BMs and not just use Hopfield networks was the expressiveness (Freund and Haussler 1994b). Theoretically it should be able to model any distribution with an RBM, while Hopfield networks are more limited. BMs also have hidden nodes, which makes them more powerful but also more difficult to train and to analyse, sometimes impossible in practice. For Hopfield networks, because everything is visible and the connections are symmetrical, is well understood and easier to train and analyse. Therefore Hopfield networks could outperform BMs, so some exploration is needed before deciding to use BMs.

2.2 RBMs internal structure

We will begin by looking at the RBM again, but in more detail. All the visible nodes $(v_1, v_2, ..., v_m)$ together form a vector v, which represents an input sample. The features are represented by the hidden nodes $(h_1, h_2, ..., h_n)$ which form a vector h.



(a) A Boltzmann machine with 4 visible nodes and 3 hidden nodes.

(b) A Hopfield network with 7 visible nodes

Figure 7: A Boltzmann machine VS Hopfield network

Important aspects of an RBM that were not discussed properly yet are the biases and weights. Every node (v_i and h_j) is accompanied by a bias a_i and b_j. Every connection is accompanied by a weight (w_{ij}), and W represent the matrix with these weights.

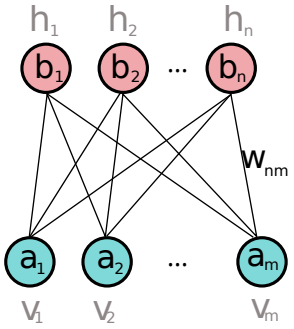


Figure 8: Biases (a_i and b_j) and weights (w_{ij}) on an RBM with m visible nodes and n hidden nodes

These weights and biases are the parameters that will be updated when training an RBM. When learned properly, these parameters tell you the importance of the nodes and connections. Before explaining how they do this exactly, energy-based models need to be explained.

BMs are energy-based models, which represent a probability distribution (LeCun et al. 2006). They come from statistical physics, in which energy is a quantitative term. The quality of a BM is evaluated by using the idea of energy, which denotes a dependency between two variables (Osogami 2006). A high energy means low compatibility between variables, and therefore low probability. The configurations of variables leading to a low energy are more probable. All the variables, or nodes, in an RBM are predefined in an energy function, which captures the global energy of the model. Energy-based models try to minimise this function because high energy is associated with low probability. In other words, the probability is proportional to the negative energy.

2.2.1 Energy function and probability distribution RBM

The energy function of an RBM looks as follow (Fischer and Igel 2012):

$$E(v,h) = -\sum_i a_i v_i - \sum_j b_j h_j - \sum_{i,j} v_i h_j w_{ij} \tag{1}$$

All the variables on which the function depends were already encountered: the visible units, hidden units, biases, and weights. This function computes the energy for a state of the RBM. Such a state consists of the binary values of the two layers, so of all the neurons. Now that the energy function of an RBM is defined, the explanation can be given of how the bias and weight parameters determine the probability of nodes and connections.

One advantage of the RBM can already be seen in the energy function. For a BM we would also have the summations for the interaction between the visible units ($-\sum_{a,b} v_a v_b w_{ab}$) and between the hidden units ($-\sum_{a,b} h_a h_b w_{ab}$). Now, only the interaction between visible and hidden units are needed ($-\sum_{i,j} v_i h_j w_{ij}$), because an RBM has no inter layer connections.

After training, the bias and weight parameters for a given data set should ensure a minimum energy, which is not necessarily the global minimum but could be a local one. If the bias a_i of a visible unit v_i is negative, $v_i = 1$ would make the first term in the energy function big, which would result in high energy. To prevent this, the term v_i is nudged to be 0 instead of 1 (Montúfar 2016). The opposite will happen when the bias of a visible unit is positive. Hidden units, based on their corresponding biases, will be nudged in a similar fashion. All weights can be found in a connection matrix W. The weight of v_i and h_j is denoted by w_{ij}, and if it is negative, v_i and h_j being one would give a higher energy. Either both or one of the units will be nudged to zero. With a positive weight, the probability of observing both units being zero is decreased.

Now that the energy function is explained intuitively, it can be shown how the distribution is defined. A probability distribution describes the probability of occurrence of all the different states of the model (Ash 2008). More specifically, the likelihoods of all the values of a random variable, within a given range. An RBM is in a certain state every time step, in which state depends on the distribution. Therefore, an RBM is probabilistic, unlike other deterministic neural networks such as classic feed-forward networks.

To go from the energy of a state with vector v and h, to a probability is intuitively just dividing it by the total energy of all possible vectors. However, because negative energies are possible, dividing by a summed energy would not result in the correct probability. For example, summing $-1, 3$, and -2 would result in a total of 0, but we cannot divide by zero. Therefore the probability of a state with vector v and h is given by:

$$p(v,h) = \frac{e^{-E(v,h)}}{Z} \tag{2}$$

with

$$Z = \sum_{v,h} e^{-E(v,h)} \tag{3}$$

It is called a joint probability distribution, the probability that two events, a state with vectors v and h, are observed together (Fischer and Igel 2012). Again, something borrowed from physics because this particular distribution is known as the Boltzmann distribution in physics (Boltzmann 1872). It gives the probability that a particle can be observed in the state, given a certain energy.

To see what 2.2.1 does intuitively, Figure 9 shows the graphic representation of the function ($y = e^{-x}$). It makes positive but relative values of the input x. It can be seen that a low energy x, will result in a high positive value y, which will result in a high probability when dividing it by the partition function. Therefore the negation, because otherwise the graph would be mirrored on the y-as. This would result in a low probability for a low energy.

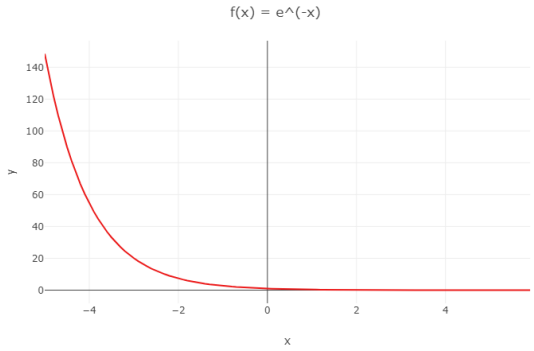


Figure 9: Graphic representation numerator softmax function

2.2.2 Sampling with conditional probabilities

First, the sigmoid activation function is introduced because a way is needed to represent the values of the model by probabilities (Basta 2020):

$$sig(x) = \frac{1}{1 + e^{-x}} \tag{4}$$

It outputs the probability that a node is one. When the function is called on the score of a node, it outputs the corresponding probability. High negative numbers will give a probability close to 0, and low negative numbers a probability close to 1. This can be seen in Figure 10. It will be

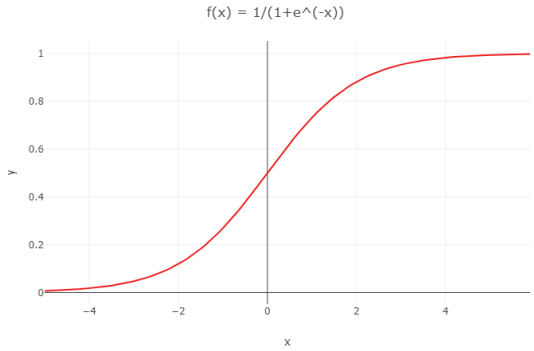


Figure 10: Graphic representation sigmoid function

used to calculate the conditional probabilities below (Fischer and Igel 2012). They are introduced because computing joint probabilities (section 2.2.1) is too difficult. The partition function Z needs to go through an exponential number of combinations of v and h. A conditional probability computes the probability of a state a given a state b, which has less possible combinations. In notation the difference can be found in the separator of the two states. Conditional probabilities are denoted by $p(a|b)$ and joint probabilities by $p(a, b)$.

The conditional probabilities will be used to obtain the values for the hidden nodes, given the values of the visible nodes, so given the input vector. Obtaining values for the visible nodes given the values for the hidden nodes is also done using conditional probabilities. These formulas will be very similar because an RBM is undirected and all nodes are connected with each other between layers, and therefore an RBM is symmetric.

First phase Binary data is used, so a visible node is turned on or not. Each visible node holds one component of a data vector. Together, all the visible nodes form one input data of the data set we want to extract features from. The hidden units are also binary, and which of these nodes turn on depends on the given input.

In the first phase, or forward pass, the score for each hidden node is calculated for a given input vector (Behera et al. 2021). The activation function above will convert the scores to probabilities, which gives the probability that a hidden node should turn on. A hidden node's score depends on the visible nodes that are turned on, the weights of their connections with the current hidden node, and the bias of the hidden node. The score, and therefore also the probability, does not depend on the other hidden nodes.

The probability of turning a hidden node j on, if a visible node i is turned on is given by (Fischer and Igel 2012):

$$p(h_j = 1|v_i = 1) = sig(w_{ij}v_i + b_j) \tag{5}$$

Following, gives the probability of turning a hidden node j on, if the input vector v is given:

$$p(h_j = 1|v) = sig(\sum_{i=1} w_{ij}v_i + b_j) \tag{6}$$

So all the visible nodes that are turned on, are multiplied with the weights of their connection with h_j. The visible nodes that are turned off result in a zero value. These multiplications are summed and this is again summed with the bias of h_j (Figure 11). This is done for every hidden node, multiplying with their own weights and biases (Figure 12).

For the probability of a certain hidden vector given a visible vector, the product of the probability of each hidden node can be taken:

$$p(h|v) = \prod_j (p(h_j|v) \tag{7}$$

When looking at this formula it can also be seen that the probability of a hidden node is independent of the probabilities of the other hidden nodes. In probability theory two events are independent if the joint probability equals the product of their probabilities (Ash 2008): $P(A \cap B) = P(A)P(B)$, and therefore $P(A|B) = P(A)$.

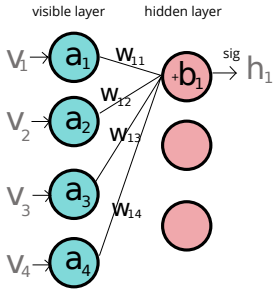


Figure 11: All visible units combined for one hidden unit.

Depending on the probability calculated for a hidden node, the node is turned on or not. Specifically, a random value is computed and if the probability is greater than this value, the node will be 1. The features represented by the hidden layer take a certain value which should be fitting to the input vector.

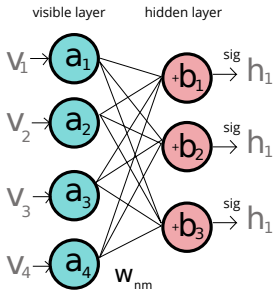


Figure 12: All visible units combined for all hidden units.

Second phase The second phase can also be called the reconstruction phase or backward pass (Behera et al. 2021). Here, the input vector given in phase one is tried to be reconstructed in the visible nodes. Therefore, the probabilities of the visible nodes are calculated based on the activation's of the hidden layer from the forward pass.

The probability of turning a visible node i on, if a hidden node j is turned on is given by (Fischer and Igel 2012):

$$p(v_i = 1|h_j = 1) = sig(w_{ij}h_j + a_i) \tag{8}$$

Following, gives the probability of turning a visible node i on, if the activation's of the hidden units are given:

$$p(v_i = 1|h) = sig(\sum_{j=1} w_{ij}h_j + a_i) \tag{9}$$

These work in a similar fashion as for the forward pass but in the other direction. Again the probability above should be calculated for each visible node (Figure 13), their binary values are based on these probabilities. For the probability of a certain visible vector given a hidden vector, the product of the probability of each visible node can be taken. As with the hidden nodes in the paragraph about the first phase, the probability of a visible node is independent from the probabilities of the other visible nodes.

$$p(v|h) = \prod_i (p(v_i|h) \tag{10}$$

Now, an approximation of the original input can be found in the visible layer.

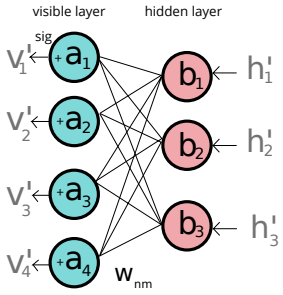


Figure 13: All hidden units combined for all visible units.

2.3 Training an RBM

Before, we assumed that the RBM had already learned the weights and biases fitting to the input data, or in other words the underlying probability distribution. This was to explain how these parameters were used, but this section will discuss how they are trained.

An RBM is trained by giving it a batch of training data (Hinton 2012). Obtaining an RBM that models this data well includes letting it assign high probability to these data vectors. One at a time such an input vector is represented by the visible nodes of the network and the biases and weights are updated accordingly. This training should continue until the distribution underlying the training data is modeled by the distribution represented by the RBM.

How the updates are done exactly depends on the training algorithm used. Further in this section (Persistent) Contrastive Divergence and Parallel Tempering are discussed. These are also the learning algorithms available in the library that is used in the implementation coupled to this thesis. First, Gradient descent and Gibbs sampling are introduced because the algorithms use these techniques to make training possible in practice.

2.3.1 Gradient descent

First, a summary is given about how Gradient descent (GD) works (Ruder 2021). It is used as a general algorithm for optimization, more specifically for optimizing the function that represents some data points. With the RBM it is also the goal to optimise its parameters so they represent the distribution underlying the data points. GD is useful because any kind of function can be optimised and it works for multiple parameters. It starts with an initial guess of the parameters and it updates them step by step. First, a loss function that suits the data should be defined. It basically represents the difference between the observed data points and the values of the current approximation of the function (residuals). Therefore, the goal is to find parameter values for which the loss function is the lowest. Than, there is a small difference between the distribution of the model and the underlying distribution. Minimizing the value of the loss function is done by taking its derivative (Spring 1985). When filling in the parameters and it results in zero, the parameters that result in a minimum are given. Therefore, for each step the current parameters are filled in, and the result gives an idea of how good they describe the underlying distribution. When we have multiple parameters of the same function, the derivative of each parameter is taken. These derivatives are called the gradient of the loss function. Now that we have the gradient, the parameters should be initialised, which for example can be done random. These values of the parameters are plugged into the gradient, or derivatives. The resulting values are not only used to evaluate the current function but they are also used to calculate the next values of the parameters. This way, the steps taken to new values are proportional to the results of the gradient, and therefore less possibilities need to be computed. Big steps are taken if the results lay far from the minimum of the loss function. Small steps are taken when they lay close to the minimum. When the step size is very close to zero or the maximum number of steps is taken, GD stops and returns the current values of the parameters. Making sure the step size is not too big, is done by using a learning rate. It is multiplied with the step size, which is in its turn subtracted from the old parameter value to get the new parameter value. In Figure 14 two graphs executing GD can be seen, each with another learning rate. GD is very sensitive to its learning rate.

Stochastic Gradient descent (SGD) is a variant of GD used when there are a lot of data points and different parameters (Ruder 2021). It takes one data point, or in practice a mini-batch of data points, at random. Which will be faster than using all of the data as in GD.

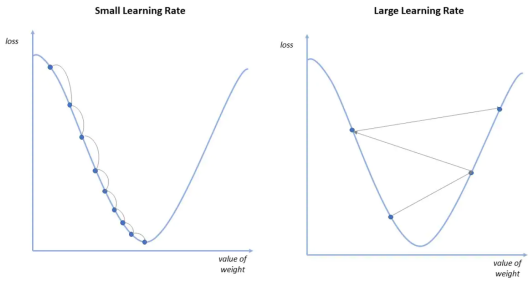


Figure 14: GD algorithms: left with a lower learning rate than right

Using GD for RBMs The loss function that can be used for RBMs is the average negative log-likelihood of the probability of an input vector (Midhun et al. 2014). This function can be found in Figure 15, on which we see that a high probability of a data sample gives a low loss. Which is what we need, the goal is thus to minimize this loss function. Sometimes the approach of maximising the log-likelihood is taken instead of minimising the negative log-likelihood, then the negative sign is dropped (Fischer and Igel 2012).

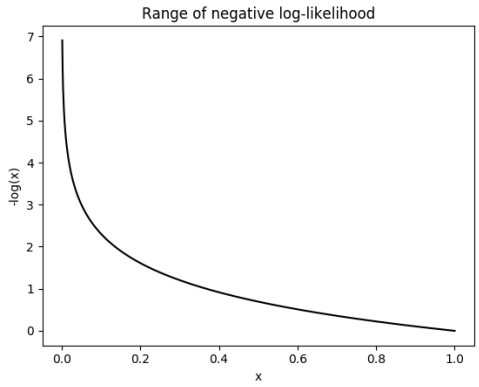


Figure 15: loss function:$L(x) = -log(x)$ (Lj 2017)

The log of the likelihood is taken because then the product can be written as a sum using the log rule log(ab) = log(a) + log(b), and this is equivalent because the log function is monotonically increasing (Jordan and Bishop 2004). Below the likelihood and negative log-likelihood are given:

$$p(x) = \prod_m p(x^m) \tag{11}$$

$$-\frac{1}{M} log(p(x)) = \sum_m -log\frac{1}{M}(p(x^m)) \tag{12}$$

The probability of v is given by $\sum_h p(v, h)$, and by filling in the formulas of section 2.2.1 and

using the log rule $\log(\frac{a}{b}) = \log(a) - \log(b)$, $\log(p(x))$ can be rewritten as:

$$\begin{aligned} log(p(x)) &= \sum_m log\left(\frac{1}{Z}\sum_h e^{-E(v,h)^m}\right) \sum_m log\left(\frac{1}{\sum_{v,h} e^{-E(v,h)^m}} \sum_h e^{-E(v,h)^m}\right) \\ &= \sum_m log\left(\sum_h e^{-E(v,h)^m}\right) - \sum_m log\left(\sum_{v,h} e^{-E(v,h)}\right) \end{aligned} \tag{13}$$

The goal is now to minimize the loss function, and therefore we need to take its derivative. The parameters that need to be adjusted are the weights W_{ij} and biases a_i and b_j, found in the energy functions. Therefore, the partial derivative of this function is calculated for each of them. The next formula give the derivation for a parameter θ (Midhun et al. 2014). Fisher further explains the derivation steps (2012).

$$\frac{\partial - log(p(v^t))}{\partial \theta} = \mathrm{E}_h\left[\frac{\partial E(v^t,h)}{\partial \theta}|v^t\right] - \mathrm{E}_{v,h}\left[\frac{\partial E(v,h)}{\partial \theta}\right] \tag{14}$$

with E, the expected value:

$$\mathrm{E}[X] = \sum_{i=1}^{k} x_i p_i \tag{15}$$

with X having $x_1, ..., x_k$ possible outcomes with their corresponding probabilities $p_1, ..., p_k$.

The derivation results in a difference of two terms: one depending on the observation and another depending only on the model. The first term of the derivation is possible to compute with the conditional probabilities of section 2.2.2. However, the two parameters for the expectation of the second term make it very hard to compute because of the exponential sum over v and h. To approximate this term, the different learning algorithms come in.

2.3.2 Gibbs sampling

Samples from the distribution in our model are obtained by using Gibbs sampling (Gelfand 2000). It is applicable when joint probabilities are too difficult to obtain and the conditional probabilities are available for each variable (each node). The conditional probabilities are obtained as discussed in section 2.2.2. This is possible because of the independency in layers. A node only depends on the nodes of the other layer.

With Gibbs sampling you start with an initialisation of all the variables $(\theta_1, ..., \theta_r)$, which are the partitions of θ (a vector). The conditional probabilities of each θ_k should be available. After one iteration the vector θ is updated by updating each of its components. this is done using the corresponding conditional probability given the variables from the previous time step. Computing $\theta^{(t)}$ is done using Algorithm 1.

Algorithm 1 Gibbs sampling for θ, partitioned in r blocks $(\theta_1, ..., \theta_r)$

1: initialize $\theta^{(0)} = (\theta_1^{(0)}, ..., \theta_r^{(0)})$
2: **for** $t = 1, 2, \ldots$ **do**
3: **for** $k = 1, 2, \ldots, r$ **do**
4: draw $\theta_k^{(t)}$ from $= P(\theta_k|\theta_1^{(t-1)}, ..., \theta_{k-1}^{(t-1)}, \theta_{k+1}^{(t-1)} ..., \theta_r^{(t-1)})$
5: **end for**
6: **end for**

A couple of variants exist for this sampling method (Oppermann 2018). Let's say you got vectors X and Y instead of one vector θ, and the conditional probabilities are $P(X|Y))$ and $P(Y|X)$. Now X^0 and Y^0 are initialised, and they are updated each iteration. First, X^t is computed with the conditional probability of X given Y^{t-1} ($P(X|Y^{t-1})$). The new Y^t is computed with the conditional probability $P(Y|X^t)$, given the just computed X^t. These two steps are repeated for a certain number of iterations.

Something similar will be done with the RBM (Fischer and Igel 2012). Now, X and Y represent the visible and hidden vector respectively, we will call them v and h accordingly. This is to make clear that the same probabilities are used as in section 2.2.2. The initialisation step is slightly different because now only v^0 is given and the value of h^0 will be sampled as in the iteration step, conditional on v^0. Initialising v^0 could be done with an input vector from the training data for example. The iteration step, sampling from the conditional distribution of v and than of h, is repeated k times. The algorithm and graphical representation can be found in Algorithm 2 and Figure 16.

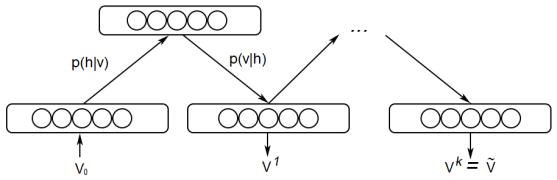


Figure 16: Gibbs sampling on an RBM (Oppermann 2018)

Algorithm 2 Gibbs sampling in an RBM with visible layer V and hidden layer H

1: initialize v^0 & $h^0 \sim P(h|v^0)$
2: **for** $t = 1, 2, \ldots$ **do**
3: $\quad v^t \sim P(v|h^{t-1})$
4: $\quad h^t \sim P(h|v^t)$
5: **end for**

2.3.3 Contrastive divergence

Contrastive divergence (CD) is the best known technique among all the learning algorithms for RBMs (Hinton et al. 2006; Hinton 2002). It will introduce a way to estimate the double expectation, which is the second term in the derivation of the loss functions (14). The term will be replaced by a point estimate of the expectation, given one observation $\widetilde{x}$. As will be discussed in section 2.4, this corresponds to doing a Monto Carlo estimate.

First, it will be discussed how $\widetilde{x}$ is obtained. In the best case, this data point is a sample from the model distribution. To imitate this, Gibbs sampling is used (section 2.3.2), which should be quite efficient for RBMs because of the independency within the layers. In CD Gibbs sampling initialises the starting v with the training sample for which an update needs to be done, and for which the gradient needs to be computed. Unlike with standard Gibbs sampling, for which a random variable is sampled. It is done for k iterations, which results in the data point $\widetilde{x}$. In practice, k does not need to be big (Bengio and Delalleau 2009).

To recapitulate, the positive term in 14 will sample h given the data point x^t. For the negative term, h will be sampled given the data point obtained from Gibbs sampling starting from the

data point x^t. This gives us this new derivative (Bengio and Delalleau 2009):

$$\frac{\partial - log(p(v^t))}{\partial \theta} \approx \frac{\partial E(v^t, h^t)}{\partial \theta} - \frac{\partial E(\widetilde{v}, \widetilde{h})}{\partial \theta} \tag{16}$$

The basic idea of the algorithm is that when an input vector is given, the energies of the model including this vector should be lowered, and all the other energies should be raised. However, due to the computation of the many possibilities, we could raise them all but it also takes too long to go through all the energies. Another technique is used which will eventually give the same result. One energy belonging to the input vector is lowered, and one random energy is raised. For the latter, even if this is an energy belonging to the input vector, one of those energies was also lowered so this has no negative impact. Otherwise, it has a positive impact so this is a good approach. This can be seen in the derivative of the loss function (14), the positive term is the energy at the training data so it should be lowered. It corresponds to increasing its probability, Which will make the gradient lower as was intended. For the negative term, the energy at the sample value of the model, the opposite is true.

In the first stages of the algorithm, the data point $\widetilde{x}$ will be noise because of the model still approximates the distribution poorly due to the random initial values for the parameters. By decreasing the probability of visible vectors which are not the training data, $\widetilde{x}$ will be looking more and more like actual training samples. When proceeding with the algorithm for a while, the sampled data from the model $\widetilde{x}$ should be getting more and more similar to the training. This implies that the gradient should be getting smaller. To sum up, each iteration of CD, a training vector x^t is selected and by using Gibbs sampling the model sample $\widetilde{x}$ is found. After calculating the hidden vectors based on these: ($h(x^t)$ and $h(\widetilde{v})$, the parameters are updated using these values. Below the actual update function of the parameters can be found, which are used after every given training vector x^t. With these update functions and α as learning rate, stochastic gradient descent is done. They should be repeated for all the visible nodes $(v_1, ..., v_m)$, hidden nodes $(h_1, ..., h_n)$, and weights between them. In other words, for every $i \in 0, ..., m$ and every $j \in 0, ..., n$.

$$w_i j \leftarrow w_i j + \alpha(v_i^t p(h_j = 1|v^t) - \widetilde{v}_i p(h_j = 1|\widetilde{v}_i)) \tag{17}$$

$$a_i \leftarrow a_i + \alpha(v_i^t - \widetilde{v}_i) \tag{18}$$

$$b_j \leftarrow b_j + \alpha(p(h_j = 1|v^t) - p(h_j = 1|\widetilde{v}_i)) \tag{19}$$

These functions were found by assigning θ to each parameter in turn and working it out further.

2.3.4 Persistent contrastive divergence

In order to improve CD, Persistent contrastive divergence (PCD) was introduced (T. Tieleman 2008). Essentially it works the same as CD, but aims to do better in drawing a sample from the model distribution. Instead of using the training vector to initialise the Gibbs sampling, it is initialised with $\widetilde{v}$ from the previous iteration. Only for the first iteration, the training vector is still used to initialise. Something small is changed, but it could result in a big difference.

2.3.5 Parallel tempering

Again, a variant of CD that is mentioned a lot is given: parallel tempering (Desjardins et al. 2010; Fischer and Igel 2012; Cho et al. 2010). It holds multiple systems, each with their own temperature. The difference between the temperatures is the distribution. Trying to prevent getting stuck in a local minimum is done by letting systems exchange information.

Formally we have M temperatures $T_1, ..., T_M$, with $T_1 = 1$ and $T1 < T_2 < ... < T_M$. They all have their own distributions ($r \in \{1, ..., M\}$):

$$p_r(v, h) = \frac{e^{-\frac{1}{T_r}E(v,h)}}{Z} \tag{20}$$

with

$$Z = \sum_{v,h} e^{-\frac{1}{T_r}E(v,h)} \tag{21}$$

The first temperature has a distribution which is exactly the model's distribution, because the fraction before the energy is factored out.

When a training example is given, this learning algorithm will compute a $\widetilde{v}$ for each temperature. It will do the Gibbs sampling for the given k, using their own distribution p_r and the state v_r as initialisation. As with persistent contrastive divergence, v_r is first initialised with the training vector, but for the following training steps it uses its previous $\widetilde{v}$. Each training step, each temperature will have their own pair of states $(\widetilde{v}_1, \widetilde{h}_1), ..., (\widetilde{v}_M, \widetilde{h}_M)$. After retrieving these, we will let the systems exchange information. It is done by letting two neighbouring temperatures exchange their pairs with a certain probability. The latter is based on the Metropolis ratio:

$$min\{1, \frac{p_r(v_{r-1}, h_{r-1})p_{r-1}(v_r, h_r)}{p_r(v_r, h_r)p_{r-1}(v_{r-1}, h_{r-1})}\} \tag{22}$$

After these swaps were done, the updates are done as before using the model distribution's sample, which is $\widetilde{v_1}$ because $T_1 = 1$. The swaps may have exchanged $\widetilde{v_1}$ and therefore explore parameter values that may not have been explored otherwise.

2.4 MRF and MCMC method

For the interested reader, this section introduces the basis on which an RBM relies. A lot of techniques described above use one or more of these concepts as a base. In a Markov Chain (MC) you go from one state to another with a certain probability (Ash 2008). It is a Markov process because it has the Markov property. Which says that the following state only depends on the current state, and not on the ones before that. All the probabilities summed together should also result in 1. One example of a Markov Chain is given in Figure 17. For example, if we are currently in state S_1, the chance we will transition to state S_2 is p_{12}.

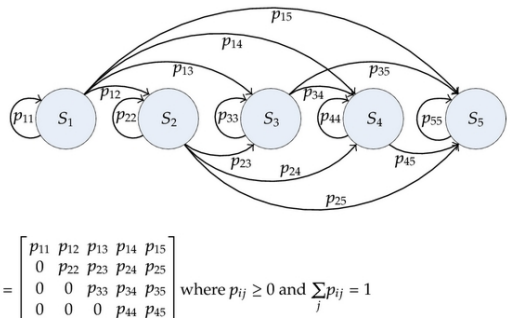


Figure 17: a markov chain with states $\{S_1, ..., S_5\}$ and probabilities $p_{i,j} i\&j \in 1, 5$ (Musumeci 2017)

It gives a directed graph in which the direction of the edge matters. However, we also got an undirected graphical model called a Markov random field (MRF) (Fischer and Igel 2012). It

is a set of random variables that have the Markov property. It could be said that MCs are 1D Markov models and MRFs are 2D Markov models. A graphical model shows the conditional probabilities between random variables on a graph (as for example in Figure 18).

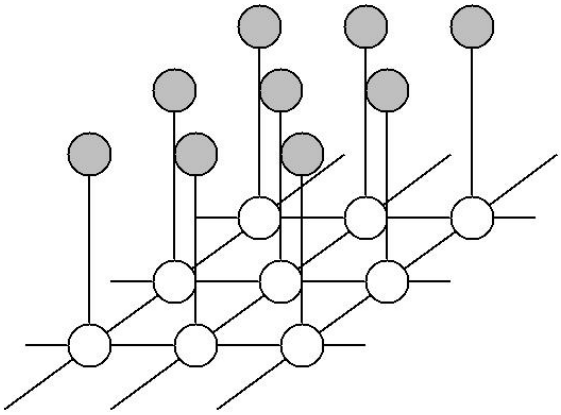


Figure 18: a markov random field

RBMs are MRFs with hidden variables as shown in Figure 8 and the learning algorithms above are based on Markov chain Monte Carlo (MCMC) methods in the form of Gibbs sampling.

MCMC combines the properties Markov chain and Monte Carlo (Van Ravenzwaaij et al. 2018). The latter is used when it is not possible to directly calculate with the equations of a distribution. Instead a large number of random samples are taken of the distribution and then examined. This way the properties of the distribution are estimated. The advantage of MC is therefore that it is easier to compute. Markov chains are combined with this property to introduce a way to take these random samples. The next sample is generated using the previous sample, but as the Markov property states, it will not be using other previous samples apart from the current sample.

3 Method

In this section an overview will be given about the experiments that were done in order to investigate the research question from section 1.7.First, the two agent-based models used to experiment with are discussed: a population of agents and iterated learning between agents. Next, the data representation of the Alorese language, used as a communication tool in these games, is introduced. In section 3.4, an overview is given about the way agents are represented by RBMs and which design decisions are involved with this. At last, based on the information discussed in the previous sections, the experiments are discussed in detail.

3.1 Agent-based models

3.1.1 Population of agents

The first model represents people of one generation, called agents, talking to each other. All of them are represented by an RBM and pretrained to learn an abstract form of the Alorese language. After this, all the agents will interact with each other.

Each time step, one speaker and one listener are randomly selected from the population and play a language game as in Figure 2. For this research, the concept will be a person corresponding

to its own conjugation. The signal represents an abstracted form of the affix depending on a conjugation from the Alorese language. When the listener receives the signal, it should retrieve and send back the concept which suits the utterance the best according to the internal structure of the agent's RBM. The feedback is positive if the concept was similar enough to the original concept, and negative otherwise. When the latter applies, the listener is trained by the speaker with the original concept. After this, two new agents are selected to interact.

The goal of letting a lot of agents interact with each other for a long time, is to see if this could result in a simplification of the language. All agents start with a similar pretrained RBM, but due to a lot of interaction a change in the language could arise.

3.1.2 Iterated learning

With iterated learning, multiple generations are considered, but a special feature is that one generation is represented by one agent. The first generation is initialised, meaning the RBM of the first agent is pretrained to learn an abstract form of the Alorese language. From then on, no pretraining is done and every agent learns the language from another agent.

Each time step two generations interact. The older one becomes the speaker, the next generation (agent) is initialised with an untrained RBM, and the speaker trains the listener with its knowledge of the language. It does this by generating the signal corresponding to a concept in its internal structure, and giving this as training data to the listener. After this is done for a lot of concepts, the listener should have its own internal representation of the language based on the knowledge of the speaker. Now, the agent that acted as a listener becomes the speaker and a new agent is created which will act as the listener.

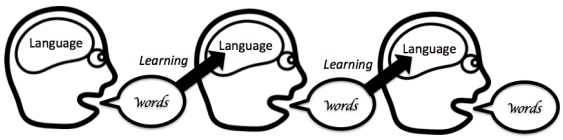


Figure 19: a representation of Iterated learning (Roberts 2013)

The reason iterated learning is used for these experiments is because it can help look at the learn-ability of a language. When the complexity of the language learned in the last generation is compared to the complexity in the first generation for example, morphological simplification could be detected.

3.2 Experiments in detail

For the experiments, the two agent-based models are run wit their agents as RBMs. One run has a specific parameter setting of these RBMs. This varies by doing a couple of runs, each time changing some value of a parameter.

First an explanation is given about how an RBM is pretrained. Then, section 3.2.2 describes how the solution to the agent being speaker an listener problem should work exactly. Another important part of the experiment is measuring the quality of an RBM parameter value. How this is done exactly is discussed in the success evaluation sections. After this, the two big experimental setups, the two agent-based models, are described in detail, and lastly the way the change in complexity of the language is measured is described.

3.2.1 Pretraining

A speaker needs a way to communicate a concept to the listener, this will be through the signal (the vector representing the suffix or prefix). Like we learn our languages, the RBM should learn to link these concepts to the signals that are commonly known. It is done by using the training data explained in detail in section 3.3, and setting the parameters from section 3.4.

3.2.2 Simulating communication

When the RBM has properly learned, we should be able to leave the signal side of the visible nodes empty, and the RBM should be able to reconstruct this. The same should be true when letting the concept side empty, in the case of the listener. There will be relied on the RBM for this communication, therefore it should be initialized and trained properly. Leaving one part of the vector empty can be done by setting these nodes all to zero, to -1 (Behera et al. 2021), or each one randomly to one or zero. The first approach is taken here, because setting the empty part of the vector is actually only turning a few bits to 'True', which is a smaller step than for the other two options. Figure 20 shows how it would look like to retrieve the signal part from the RBM, given the concept.

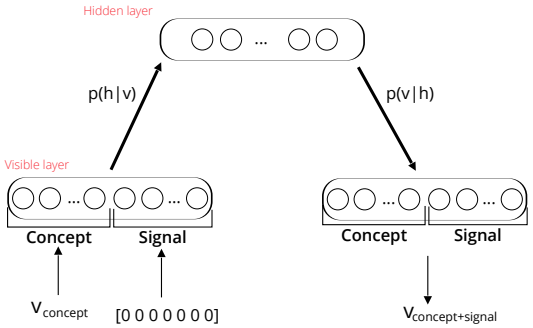


Figure 20: Giving a concept and empty signal to the RBM, which reconstruct the concept with the signal part filled in

In practice this means that the visible units are set with one part of the vector and the other part to zero. Then Gibbs sampling is used: The probabilities for the hidden nodes, given these visible nodes, are calculated and used to set the hidden nodes. From these, the probabilities of the visible nodes are calculated and used to decide which one to set, given the hidden nodes. This should result in the completed vector, and the before empty part can be retrieved. So the signal given the concept, or vice versa. The process above is used to let the speaker retrieve its signal for a given concept, and for the listener to retrieve its concept for a given signal.

3.2.3 Success evaluation

To evaluate the success of a certain parameter choice, we need a way to measure the communicative success. The communicative success is the dependent variable in that case and the parameter setting the independent variable. A communicative success takes place when the listener returns the same concept that the speaker tried to communicate through a signal. In our case this would mean that the first couple of visible nodes of the speaker, the ones representing the different persons, are the same as the visible nodes found for the listener.

The RBM receives a concept with no signal linked yet by putting the second part of the visible nodes to zero. By doing this, it may change the original concept part of the visible nodes because this output is only based on the probabilities of the hidden units. RBM are stochastic models, there is no guarantee an input will always give back the same output (reconstructed input).

A solution for this could be clamping. With this technique certain nodes are fixed and cannot be changed by the Gibbs sampling step. Unfortunately this is no feature of the library that will be used for the experiments. We will be using a different approach to try and achieve the same result. This would be the following design choice: keep the original concept as the one to compare the concept of the listener with. It would also make less sense to use the other concept, which was found after asking the RBM for the signal part. There would be no handle anymore on what concept we really try to communicate.

An alternative way of determining communicative success could have been to make sure the speaker links a person to the correct suffix/prefix. In other words, if it links the left side of Table 2 to its complementary right side, which can be found in the same row. This includes the training data, also during the interactions between agents.

In a real population, people also make mistakes but could eventually succeed in understanding what is meant. With the latter approach, the speaker is directly punished for this mistake and it is less natural. The first approach should normally keep the model closer to a real language where people still understand the meaning even though the word that was used is not completely correct.

Success evaluation redundant data The RBM is given more room for error when redundant data is used. Even when the listener did not get the concept completely correct, it is not punished as if it was completely wrong. This is where the Manhattan distance becomes useful. It outputs the vector which is the result of subtracting two vectors of the same length. When only working with ones and zeros this just gives the amount of ones that were misplaced by the RBM.

As before, the concepts are compared to represent the communicative success, so only those parts of the vectors are subtracted. However, now it will not give a success rate but a distance, which represents the number of bits that where different from the original concept the speaker wanted to communicate.

3.2.4 Population

The pseudo-code for the agent-based model of the population of agents is given in Algorithm 3. It is an overview of a whole simulation but some steps are discussed in more detail below.

Initialisation model In the first agent-based model, a population is represented using the mesa library. A model was defined which holds a certain number of agents. At the start of a run, each agent gets a new RBM assigned, and a pretraining is done with the data of section 3.3. The RBM gets this batch a couple of times: the number of loops that was predefined.

Algorithm 3 Agent-based model: population of agents

```
Initialize + pretrain RBM for the number of agents in the population
for r = 1, 2, ..., number of re-dos run do
    for i = 1, 2, ..., number of interactions do
        for a = 1, 2, ..., number of agents do
            1) pick a random agent a_r ≠ a
            2) Pick random concept from data set
            3) Get signal by giving concept to visible nodes RBM
            4) Give signal to RBM of a_r and retrieve its concept
            5) Compare concept to original concept
        end for
    end for
end for
```

After this initialisation phase, the interactions between the agents begin and this data set will not be used anymore. However, an extra data file was made with the signal part of each row put to zero. It is used to deliver a concept to the speaker, as the start of an interaction. One of these rows, and therefore the concept, is chosen randomly.

The number of people that should be in the model and the number of interactions that should be done by each agent, need to be given to initialise the model. We take a sample from the population on the Alor and Pantar Islands. Generally the minimum sample size is 100 and the maximum is 10% of the population as long as it does not exceed 1000. Due to performance, experiments will generally be done using 10 agents and the number of interactions for each step will be 100. However, one experiment will be done with 100 agents and 1000 interactions, and its results are compared with the ones using 10 agents and 100 interactions.

Interaction between 2 agents One agent (a_1) starts the interaction with another randomly chosen agent (a_2). The speaker RBM (a_1) is given a random concept, and by using the hidden unit probabilities that where set, the concept and a signal are given back. Because we do not use the predefined data anymore, it could be the case that this signal part does not comply with the original concept. By concatenating 8 zeros and the signal part, an input sample is made to give to the listener (a_2). Which, in his turn, returns a complete concept and signal, according to its hidden layer. The concept part is responded to the speaker, who evaluates this by comparing it to its original concept.

If it results in a communicative success, the personal count of the speaker is increased, otherwise it will train the listener with its original concept and the signal part that was found by the speaker's RBM.

It is the same if the count of the speaker or the count of the listener is increased, because the total count of all persons is taken and is be divide through the number of persons.

Complete run of the algorithm In the model, for each iteration an interaction is started once by each agent. This means this agent will play the speaker role. A random activation of the agents is applied, so the order in which the agents are chosen is random. For the interaction they choose another agent from the population randomly, who will play the listener role. Therefore each agent starts an interaction once per iteration, but it is not known upfront how many he will undergo. The interaction will take place as explained above, and the number of iterations is the number of interactions as defined in the initialisation phase.

The agents do not know each others' internal RBM settings, the weights/probabilities that were found after pretraining and during training. Each agent can speak to all the other agents in the model, and consequently, each agent is able to listen to any other agent.

Parameter evaluation The parameter experiments are done by using different values for the parameters and testing its communicative success. In a population, this means that all the agents, when initialised, use this value for the parameter to initialise or train the RBM with. All the parameters will be tested on how good they act in a population where a lot of interactions take place.

The communicative success is defined as the average success per person per interaction. Therefore it does not matter to link the communicative success to the speaker or the listener in section 3.2.4. At the end of the run, the sum of the communicative success of each agent is taken. To get the average, it is divided by the number of agents in the population and the number of interactions done per agent. This will give a value between 0 and 1.

For the redundant data, and therefore the Manhattan distance, the average distance per agent per interaction is calculated. It is done in the same way as above, except it will not give a value between 0 and 1. Dividing it again by the length of the vector and subtracting 1 by this, does give such a value. However, this may give a wrong interpretation on the success because there is not a high chance that all bits, or even a lot of bits, are wrong. Getting half of the bits wrong already should give a very low success rate instead of just 0,5.

A run for each parameter should be repeated to calculate the error of the result of the experiment. Generally an experiment is repeated 3 times, which ensures a two-thirds (66%) probability that the averaged results are more accurate than a single experiment. We will do five repeats.

3.2.5 Iterated learning

Now, instead of producing a population by letting different agents interact, one agent will represent a whole generation. Each generation will interact with the next one, who will in turn interact with the next, and this will go on for a couple of iterations. This is represented in the following pseudo-code (Algorithm 4) for the agent-based model of iterated learning.

Algorithm 4 Agent-based model: iterated learning

```
Initialize + pretrain one RBM a_1 and initialise one RBM a_2
for r = 1, 2, ..., number of re-dos run do
    for g = 1, 2, ..., number of generations do
        for l = 1, 2, ..., number of learning do
            1) let a_1pick random concept from data set
            2) get signal by giving concept to visible nodes RBM
            3) give original concept and retrieved signal to train RBM of a_2 with
        end for
        1) a_1 = a_2
        2) a_2 = a new initialised RBM
    end for
end for
```

Initialisation model The parameters that are needed to initialise the model are: the number of generations and the number of training iterations when two generations (two agents) interact.

For the first, 5 or 20 generations are used, and for the latter 1000 or 100000 learning iterations are done.

A first step is to initialise the newly added agent with an RBM, and pretrain only the very first agent with the data set from section 3.3. The latter will become the initial generation, and from then on this data set will not be used anymore. As in section 3.2.4, the file with concepts will still be used to obtain random concepts for the agent to express.

Interaction between 2 agents Two possible interactions can take place: an agent can pass its knowledge to another agent and an agent can evaluate another agent.

When an agent (a_1) passes his knowledge to the next agent (a_2), a concept is chosen randomly and given to the RBM of a_1, which produces the signal part. By concatenating the original concept and this signal we obtain one data sample of knowledge of this agent (a_1). This is done for a couple of random concepts, specifically for the number that would make one batch. This batch is given to agent a_2 to train its RBM with.

Another possible interaction is letting an agent (a_1) evaluate another agent a_2. It will calculate the communicate success with agent a_2. A concept is again chosen randomly and the signal part is returned by the RBM. Now, the interaction will continue as in section 3.2.4.

Complete run of algorithm After initialising the first generation, it acts as a teacher for the second generation, so it will be a_1 in the first kind of interaction. After doing this for the given number of training iterations, the second generation becomes the teacher. Instead of listening to the agent, it now becomes the speaker, and will pass its knowledge to a new third generation. This will continue until the number of generations is reached, which is initialised in the beginning. Any agent can evaluate another agent.

Again, the agents do not know each others internal RBM settings.

Parameter evaluation As for the population, each agent (generation) will use the same value of the parameter that will be tested. All the parameters will be tested on how good they act in an iteration of generations.

The communicative success between two generations is defined as the average success per interaction. There is no population, so no need to divide by the number of people, because a generation exists of only one agent. A number of evaluations will be done and the sum of the communicative success of each evaluation is taken. To get the average, it is divided by the number of iterations, which will again yield a value between 0 and 1.

For the Manhattan distance, the same approach is used. Only, it will give back an average distance per evaluation instead of a number between 0 and 1.

3.2.6 Complexity of the language

For both agent-based models, after the best parameters are selected, we will look at the complexity of the language before a run was done and after. This is done by giving the 14 concepts from the training data and looking at the signals that are returned. It is counted how many different signals are left after the run, compared to how many different ones there were before the run. In the population the complexity before the interactions happened and after they have happened are compared. For iterated learning, the complexity of the first generation and the last generation is compared.

3.3 Data representation

In order to model the communication in practice, we need a way to represent a concept and a signal. They are based on the conjugations in the Alorese language, but an abstraction of it is used. For neural networks it is not easy to pass along information that the network should not learn. It should just be directly usable as extra information. In our case this would be the verb. Given the verb and the person, the signal part should return the conjugated verb. Our implementation will not look at the phonetics, so in particular not to the letters of the verb base and the affix close to each other. Therefore, the base form of the verb does not give extra info needed to learn the affixes, and is left away as a first abstraction. The concepts only represent the persons and the signals only the affixes. Another abstraction, based on the fact that we will not be looking at the phonetics, is that an affix is viewed as a whole. Rather than every letter having their own representation, every affix has their own. It is also not taken into consideration if a verb is transitive or intransitive, if it is an irregular verb that uses other prefixes or suffixes, and if the verb is 'go' and therefore has both a suffix and a prefix.

The abstracted interpretation of the Alorese language is represented in the concepts and signals by using bit vectors. Depending on which positions are set to 'True', a certain concept or signal is expressed. First, each representation is explained in detail separately. Then, the way in which they are combined is discussed.

3.3.1 Concept

The concepts will express the different persons for which a verb needs to conjugate. In the Alorese language there are 7 persons, as can be seen in Table 1.

A one-hot vector approach is taken, for which a vector consists of a single one and the rest are zeros (D. H. Pham and Le 2018). Each concept is represented by its own one-hot vector and to make this possible, the vectors need a length of 7. A bit for every of the 7 persons: first person singular, second person singular, third person singular, first person plural exclusive, first person plural inclusive, second person plural, and third person plural. More bits are needed for this approach because each new concept adds a new bit to the vectors. The fast growing length of the vector is not a big problem because there are not a lot of concepts to express in this case. An advantage of one-hot encoding is that all concepts have an equivalent representation so no concept has a higher probability of being chosen than another one. Additionally, like this each node, from the nodes appointed to represent the concepts in the RBM, will represent one concept.

Using one-hot vectors makes our data very sparse. For a vector length l, there are 2^l possible bit patterns but only l are used to represent our data. In section 3.4 it will be discussed how this should help for training the RBMs.

In the left part of the table below you will see the bit that needs to be turned on in the vector when we want to represent a certain person. An extra bit is added at the front of the one-hot vectors to say if we are working with a suffixing verb or not. Some verbs use a prefix for the different persons and some use a suffix. This knowledge is passed along in the data because it is not the task of the model to learn this. In our data there is not a fundamental difference between them but they are used to stay closer to the Alorese language. When a more complex representation of the data should be used, it could already use this representation of them both as a base.

The vectors have length 8, one for the extra bit with the suffixing information plus the seven bits representing the different persons.

	Person								Affix						
	suffix	1SG	2SG	3SG	1PL. EXCL	1PL. INCL	2PL	3PL	suffix	k- -kðn	m- -ko -no	n- -na -nðn	t- t- -te	r- -ke -ne	-ka
1	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0
2	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0
3	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0
4	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0
5	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0
6	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0
7	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0
8	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0
9	1	0	1	0	0	0	0	0	1	0	1	0	0	0	0
10	1	0	0	1	0	0	0	0	1	0	0	1	0	0	0
11	1	0	0	0	1	0	0	0	1	1	0	0	0	0	0
12	1	0	0	0	0	1	0	0	1	0	0	0	1	0	0
13	1	0	0	0	0	0	1	0	1	0	0	0	0	1	0
14	1	0	0	0	0	0	0	1	1	0	0	0	0	0	1

Table 2: table of representation of concepts and signals from abstract Alorese

3.3.2 Signal

The signal, so what the listener will receive, is also represented by a one-hot vector. It consists of the suffixes or prefixes that are used with the different persons in the Alorese language. The different affixes can be seen in Table 1 and are shown in the top row on the right part of Table 2. In the latter, inspired by the Alorese language, the representation of the affixes are shown using one-hot vectors. Again, as with the concept a bit is added to know we are trying to communicate a suffixing or prefixing verb. Therefore the length of the vector is 7: 6 bits for the affix one-hot vectors and 1 bit for the suffixing information about the verb.

It was also possible to not use the same one-hot vector for the first suffix and the first prefix. This would have meant that 11 different one-hot vectors were needed: 5 for the different prefixes and 6 for the different suffixes. However, this would have resulted in a data vector that was 5 bits longer, while the suffix bit is suspected to include enough seperation between the two.

As an example we will take the one-hot vector [1 0 0 0 0 0], which can be found on the first row, the eight row, and on the eleventh row. If the bit before this vector is zero, it represents a prefix. More specifically the 'k-' prefix because the bit is turned on for that column. The bit before the vector can also be one, and then it represents a suffix, namely '-kvn'. In the table another abstraction can be viewed: For the suffixing verbs, for the same person, there are sometimes two possibilities to use as a suffix. These will be represented by the same one-hot vector.

3.3.3 Mapping the concept and signal

Combining the concept and signal results in one data vector of length 15. The reason why they are combined, is that the RBM can complete one, both as speaker and as listener. This is a possible solution for the problem that an agent needs to be able to be both. In the table every row depicts such a data sample, and the whole table represents the data set that will be used for our experiments. In total there are 14 of these training samples, representing the mapping of

persons with their prefixes and suffixes. Two different persons with the same affix are represented by the same one-hot vector, as can be seen in orange. As an example we will look closer to one of these samples, [0 0 1 0 0 0 0 0 0 0 1 0 0 0 0] at row 2. Its bit is turned on for the second person singular. When we look at the right side of this row, the bit is turned on for the prefix 'm-' because the suffix bit is not turned on. This mapping of the 2SG and the prefix 'm-' can be compared to what is in Table 1. Row 12, in blue, shows that a person with the suffix bit turned on, uses the suffix row ('t-' and 'te-' in this case).

	Person								Affix						
	suffix	1SG	2SG	3SG	1PL. EXCL	1PL. INCL	2PL	3PL	suffix	k- -kðn	m- -ko -no	n- -na -nðn	t- t- -te	r- -ke -ne	-ka
1	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0
2	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0
3	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0
4	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0
5	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0
6	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0
7	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0
8	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0
9	1	0	1	0	0	0	0	0	1	0	1	0	0	0	0
10	1	0	0	1	0	0	0	0	1	0	0	1	0	0	0
11	1	0	0	0	1	0	0	0	1	1	0	0	0	0	0
12	1	0	0	0	0	1	0	0	1	0	0	0	1	0	0
13	1	0	0	0	0	0	1	0	1	0	0	0	0	1	0
14	1	0	0	0	0	0	0	1	1	0	0	0	0	0	1

Table 3: table of representation of concepts and signals from abstract Alorese

3.3.4 Redundant data

Normally neural networks are optimised to remove redundancy (Jiang et al. 2018). However that is about redundant hidden units which represent the same feature. The redundancy we will be introducing is in the data. In the experiments it will be tested if this adds some reliability to the results. Each column's values from Table 2 will be copied, resulting in Table 4. For example row 1 will now give this vector: [0 0 1 1 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] The length of the data vectors will be doubled now and have a length of 30: 16 for its concept and 14 for its signal.

Now, when the signal is given and the concept is returned with one wrong bit, it could still be interpreted correctly. For the signal part above, the feedback is the best for a concept that completely aligns with the training data's concept: [0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0]. However if there is a bit wrong in the concept, and for example [0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0] is returned, it does not represent another concept but is still similar to the one in the training data. The feedback will be less positive than for the previous one but it will still be positive. Depending on the number of bits that are wrong the feedback varies. Like this, the RBM could possibly be robuster against noise because of the fact that an RBM is stochastic and not always returns the same completion for a certain signal or concept.

There should be a trade off between reliability and efficiency because adding this redundancy also doubles the length of the vector. It still has to be investigated if this realy adds reliability

for our experiment.

	Person								Affix						
	suffix	1SG	2SG	3SG	1PL. EXCL	1PL. INCL	2PL	3PL	suffix	k- -kɔ̀n	m- -ko -no	n- -na -nɔ̀n	t- t- -te	r- -ke -ne	-ka
1	0 0	1 1	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
2	0 0	0 0	1 1	0 0	0 0	0 0	0 0	0 0	0 0	0 0	1 1	0 0	0 0	0 0	0 0
3	0 0	0 0	0 0	1 1	0 0	0 0	0 0	0 0	0 0	0 0	0 0	1 1	0 0	0 0	0 0
4	0 0	0 0	0 0	0 0	1 1	0 0	0 0	0 0	0 0	0 0	1 1	0 0	0 0	0 0	0 0
5	0 0	0 0	0 0	0 0	0 0	1 1	0 0	0 0	0 0	0 0	0 0	0 0	1 1	0 0	0 0
6	0 0	0 0	0 0	0 0	0 0	0 0	1 1	0 0	0 0	0 0	1 1	0 0	0 0	0 0	0 0
7	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	1 1	0 0
8	1 1	1 1	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
9	1 1	0 0	1 1	0 0	0 0	0 0	0 0	0 0	1 1	0 0	1 1	0 0	0 0	0 0	0 0
10	1 1	0 0	0 0	1 1	0 0	0 0	0 0	0 0	1 1	0 0	0 0	1 1	0 0	0 0	0 0
11	1 1	0 0	0 0	0 0	1 1	0 0	0 0	0 0	1 1	1 1	0 0	0 0	0 0	0 0	0 0
12	1 1	0 0	0 0	0 0	0 0	1 1	0 0	0 0	1 1	0 0	0 0	0 0	1 1	0 0	0 0
13	1 1	0 0	0 0	0 0	0 0	0 0	1 1	0 0	1 1	0 0	0 0	0 0	0 0	1 1	0 0
14	1 1	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	1 1

Table 4: table of redundant representation of concepts and signals from abstract Alorese

3.4 Parameters of RBM to experiment with

The main component in the experiments will be the Restricted Boltzmann machine, which will represent an agent in the models from section 3.1. In section 2 RBMs were discussed in detail, from which could be concluded that there are a lot of things to consider when using an RBM. The task you want to carry out has to comply with the tasks for which RBMs are suited, which is the case here. Our main goal is to enable agents to be speakers and listeners while using neural networks. Because RBMs can reconstruct partial data, we will try to let the hidden units retrieve features from the training data discussed in section 3.3. This should make it possible to retrieve the missing signal when a concept is given or vice versa. The way the data is represented could therefore be a solution for the problem to make an agent both a speaker and listener. By mapping the concept and signal in one data sample, the roles of input and output could be intertwined.

To do this some design decisions need to be made about the RBM, and the ones most interesting for our experiments will be looked at. Defining the number of visible units and hidden units needs to be done correctly. Another important item is using the learning algorithm (section 2.3) that suites this task the best. Together with this algorithm goes the learning rate needed for the update rules. As we saw in section, 2.3.1, gradient descent is very sensitive to the learning rate. In order for the RBM to be decently trained, enough training examples should be presented to the learning algorithm. Just how many is interesting to explore. At last, how the weights and biases are initialised could also make a difference for the quality of and the time needed for the RBM learning.

These parameters are varied to try and discover their best setting for our task, partly by trial and error. After some technical details it is discussed how they will be varied.

Technical details The experiments were programmed in Python, and the library PyDeep was used for its implementation of RBMs (Melchior 2020). Specifically, the 'BinaryBinaryRBM' is used which has binary visible and binary hidden units. It means that a node can either be zero or one. This is also conform with the binary training data, and hidden units are typically kept binary because they were developed that way (Hinton 2012). In this library it is possible to vary all the parameters discussed above. Some extra packages were used for the details of the code. The mesa library (Kazil et al. 2020), used for agent-based modeling, is deployed for the population model. Pandas is used to make a data frame of the results of the experiment and write them to an excel file (pandas development team 2020). A data frame is like a dictionary for which the data is put in a labelled array. Seaborn (M. Waskom et al. 2017) and Matplotlib (Hunter 2007) were chosen for the representation of the results in figures. With Seaborn it is possible to display the error when repeating the same experiment. By default it shows a confidence interval of 95% (M. L. Waskom 2021).

The link of the git repository with all the code can be found here: https://github.com/Jmuyshon/RBM-in-agent-based-models.

3.4.1 Visible units

The visible units of an RBM are the nodes that can be seen and set. In section 3.3 our data was discussed, and it will be given to the RBM through these nodes. Therefore, the number of visible units is the length of the concept vector plus the length of the signal vector. The 8 first nodes always represent the concept and the next 7 nodes represent the signal. At least two nodes will always be turned on, one for the concept and one for the signal. Which one depends on the exact data sample and if it is a suffixing verb two more will be turned on. More specifically, the first and the 9^th. When the visible nodes are set, the biases and weights connected to the hidden nodes will be updated accordingly.

The number of visible nodes will not really vary, except when redundant data is used, then all the numbers discussed above need to be doubled.

3.4.2 Hidden Units

An important parameter to set is the number of hidden nodes (HN). The more HN, the more computation is needed as can be seen in the conditional probability functions of section 2. Every added hidden node adds v new weights and a new bias (v the number of visible nodes). However, not enough hidden nodes will make it impossible to extract enough meaningful features from the training data. Less hidden nodes means more generalisation. Therefore, a trade off should be made between efficiency and result.

Too many hidden nodes can also cause overfitting, which is not really a problem in our case. Overfitting is training a model completely to the training data which makes it very difficult to deal with data that was never seen before. It is not our goal to introduce other data than the training data but after extending the model, the overfitting problem should be considered more closely again.

Le Roux and Bengio state the following: *Any distribution over* $\{0,1\}^v$ *can be approximated arbitrarily well with an RBM with* $k+1$ *hidden units where* k *is the number of input vectors whose probability is not* 0.

They also provide a further proof (Le Roux and Bengio 2008). In our case v is 15 (or 30 for redundant data) and the number of input vectors whose probability is not zero, should be the 14 training examples described in section 3.3. This would result in the number of hidden nodes being 15. Each hidden unit would then represent one data sample. However, Montúfar and Ay say this could be halved (2010).

To decide what number of hidden units is best for our Boltzmann machine, we take a trial-and-error approach with the theoretical best HN as starting base. The number of hidden units will be varies from 7 to 30, comparing the quality of the resulted RBM.

3.4.3 Learning algorithm

In Pydeep an implementation is available for the 3 learning algorithms discussed in section 2.3: Contrastive Divergence(CD), Persistent Contrastive Divergence (PCD), and Parallel Tempering (PT). Apart from these, there are two more options from which we can choose: Independent Parallel Tempering (ipt), and Exact Gradient Descent (GD). The first one works as PT but the swaps in the chain are not done from sample to sample as discussed. Instead they will be done from batch to batch, so a PT instance is run for each sample in parallel which speeds up the sampling but also decreases the mixing rate. GD is the normal gradient descent, so with no approximation of the negative term, the point for which the learning algorithms were introduced. It involves calculating the partition function for each update. This can only be used for really small binary RBMs. Our RBM does not seem fit for this. It is binary but already has a lot of visible nodes, so it will not be included in our experiments.

All the other variants will be tested to ensure we find the approximation technique that matches our data. From the theory PT and IPT are expected to take longer to run than the other two. However, they may outperform the other two in quality of the learned distribution. PCD is also expected to outperform normal CD. It should take the same amount of time but ensure a better approximation of the distribution of the model.

For each of these learning algorithms, other parameters can also be varied. The learning rate, the number of loops through the data, and the initialisation of weights and biases are experimented with. Some other parameters exist, like the k from section 2.3.2. In practice $k = 1$ works well (Bengio and Delalleau 2009) and a higher k would again enlarge the computation time.

Learning rate The learning rate in a learning algorithm decides how much impact one training sample should have. Decreasing it at the end training typically helps to avoid some noise which would otherwise be there (Hinton 2012). Overall it should not be too big because the steps could stay too big and it would be hard to get the gradient close to zero. Hinton says: "Looking at a histogram of the weight updates and a histogram of the weights. The updates should be about 10^{-3} times the weights" (2012). This also needs a trial and error approach so we will try different values for the learning rate and look at the quality of the trained RBM to compare them. There will also be looked at the effect of decreasing the learning rate towards the end.

Epochs of data This parameter is used when the pretraining of an RBM is done. After the creation of the RBM, it is given the training data. For one input this means that the right visible nodes are set to true. The RBM will then set its hidden units depending on these nodes. Because we do not have a lot of different data samples, the data samples are given a couple of times. To decide the optimal number of loops (epochs) that need to be done, we take a trial-and-error approach. When we train the RBM too much, there will not be a discovery of the underlying trend, but the data will be learned literally. This could be a drawback when partial data is given and the rest should be inferred. Training the RBM too little results in a model which has not learned the mapping between concepts and signals.

3.4.4 Initialisation weights and biases

Other parameters that can be adjusted are the initial values of the weights and biases. This would steer the starting state of GD, and could possibly speed up the convergence. Initialisation can make a big change in the performance of an algorithm. Two categories of RBMs will be considered: a centered and a normal RBM. In a centered RBM the initial weights and visible/hidden biases are set to 'AUTO'. For the weights and biases this means that they are initialised randomly. An RBM is called normal in PyDeep when those initialisations are set to 0.0 instead of random.

Both variants are considered when initialising the agents with an RBM at the beginning of an experiment.

4 Results

The results below are the outcome from doing the experiments described in the methodology section 3. The quality of a parameter value is tried to be measured through communicative success and the Manhattan distance. One run represents initialising the RBMs, pretraining, and playing the game. Such a run is redone 10 times for every experiment to get a more reliable result on which the error is computed. The figures below will include this error, which is a 95% confidence interval. First the population model is discussed, both for the data described in section 3.3 and for the redundant version. Then, the iterated model is discussed for these two. To begin with an overall view, more general parameter values are combined. With the information from these results, more specific parameter values can be tried and other parameters from the model can be varied.

4.1 Population

Unless mentioned otherwise, for one run the agent-based model is initialised with 10 agents and each agent will initiate 100 interactions, each time choosing a random other agent to speak to. Only one RBM is initialised and pretrained for one run, and each agent is initialised with a copy of this RBM to use further in the interactions. Every time a run is repeated, a new RBM is initialised and pretrained to copy, which increases reliability of the results.

A more general overview of all the parameters we want to investigate is given here. A range from 1 to 30 is taken for the hidden units, so up till the double of the recommended number (15) and lower than this number because in section 3.4.2 it is discussed that this should also give a good result. The initialisation of the weights and biases is either random or zero, so a centered or normal RBM. For the number of loops through the data (number of epochs of pretraining), 3 values are investigated: 1000 epochs, 10 000 epochs, and 100 000 epochs. The 4 learning algorithms from section 3.4.3 ((P)CD/(I)PT), and 4 values for the learning rate are tried (0.01, 0.05, 0.1, and 0.5). An experiment is done for each combination of all these values. For clarity only a subsection of the results belonging to these combinations is shown here. The interested reader can find all the graphs in the 'results' map in the GitHub in section 3.4.

4.1.1 Hidden units, learning rates and number of epochs pretraining

For the figures in this section the number of hidden units is varied and can be found on the x-as. On the y-as, the communicative success or Manhattan distance can be read. The different colors represent different number of epochs that were used to pretrain, so the number of loops

through the data. For one color, the darker it is, the higher the learning rate that was used in that experiment.

The results shown in Figures 21 and 22 were produced by doing the experiments with the normal data representation (non-redundant), and using communicative success as evaluation measure. The higher the bars come, the more successful the communication is, and the better the corresponding value for the parameter is. Figure 21 uses a centered RBM and Contrastive Divergence as learning algorithm, and Figure 22 also uses a centered RBM but uses Persistent Contrastive Divergence as learning algorithm. Only these two combinations are shown because the other 6 combinations, with other learning algorithms or/and initalisation methods of the weights and biases give back a similar result. These can be found in the GitHub in section 3.4, in the directory called 'population-communicative_success'.

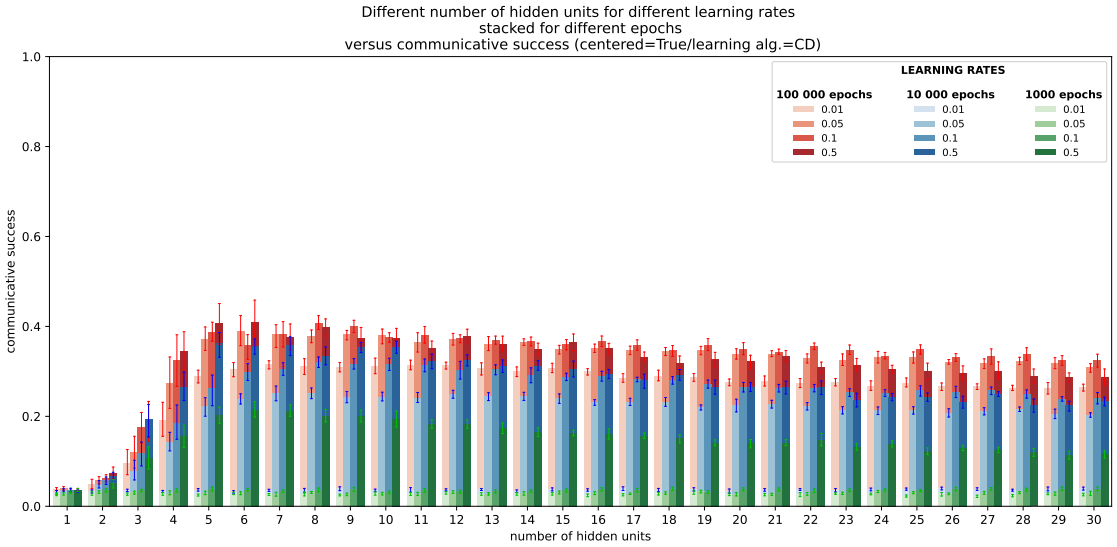


Figure 21: For different number of hidden units depicted on the x-as, the communicative success is given depicted on the y-as. A centered RBM and Contrastive Divergence is used, and for each hidden unit 4 different learning rates are tested (0.01, 0.05, 0.1, and 0.5). The darker the color, the higher the learning rate. For each of these learning rates, the gain of increasing the number of epochs pretraining by tenfold is shown by stacking them. Showing the results for using 1000 epochs (green), 10 000 epochs (blue), and 100 000 epochs (red).

For every hidden unit, four bars can be seen which represent the different learning rates with which an experiment is done. Every bar stacks the values resulting from using different number of epochs. From green to red, or from down to up, 1000, 10 000, and 100 000 epochs are represented. The blue color of one bar represent the amount of communicative success that was gained when a tenfold increase of the 1000 epochs (green) is done. Similar for the red color, which shows the gain for a tenfold increase of the blue color (10 000 epochs). Figures 23a and 23b show how the data for one hidden unit of Figure 21 would look like when it was not stacked.

Overall the higher the number of epochs, the higher the communicative success. Which also applies for the learning rate. However, it can be seen that increasing the number of epochs results in very little or very high jumps for the small learning rates. While the jumps are more evenly large for the biggest learning rate. Comparing the two graphs, PCD tends to be more error prone than CD. Looking at the graphs in the GitHUb (section 3.4), the error of IPT tends to be more error prone than that of CD, PT again more than IPT, and PCD more than PT.

A very low value for the number of hidden units performs evenly bad for all number of epochs and all learning rates. The impact of increasing the number of hidden units is big in the

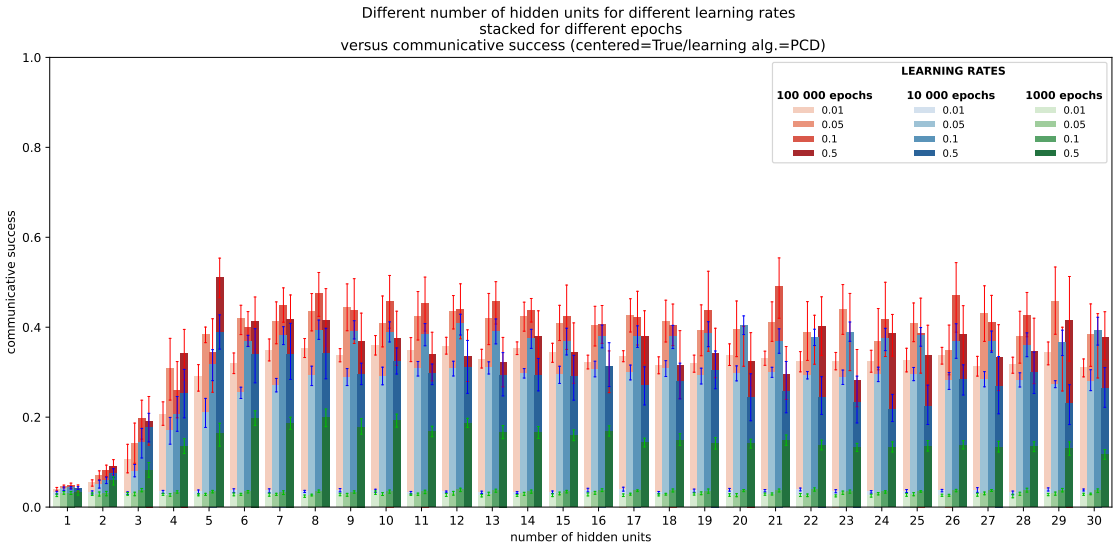


Figure 22: For different number of hidden units depicted on the x-as, the communicative success is given depicted on the y-as. A centered RBM and Persistent Contrastive Divergence is used, and for each hidden unit 4 different learning rates are tested (0.01, 0.05, 0.1, and 0.5). The darker the color, the higher the learning rate. For each of these learning rates, the gain of increasing the number of epochs pretraining by tenfold is shown by stacking them. Showing the results for using 1000 epochs (green), 10 000 epochs (blue), and 100 000 epochs (red).

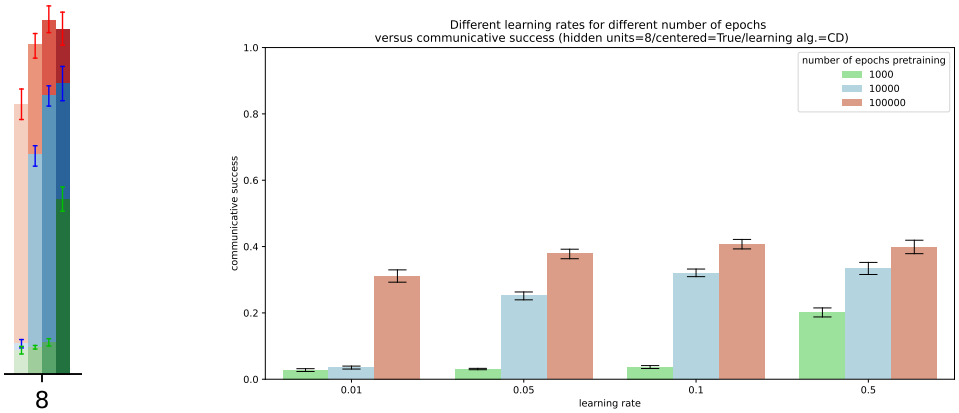


Figure 23: A close up of 8 hidden units in Figure 21 (a) and the way it would look like if the values were not stacked for different number of epochs (b)

beginning. However, from around 6 hidden units, the impact is much less when increasing it. It even begins performing worse from a certain level of hidden units. For some learning algorithms this is less visible or less rapid than for others.

In Figures 24 and 25, the experiments explained above are repeated but with the redundant data, and therefore the other evaluation measure: Manhattan distance. A point represents the average number of bits that were wrong for 1 interaction, by taking the average of the 1000 interactions (10 agents, initialising 100 interactions each). Now, the lower these points lay, the

more successful the communication is. Only these two combinations are shown, for the same reason as mentioned for the non-redundant data, and the graphs of the other combinations for redundant data can be found in the 'population-Manhattan_distance' directory in the GitHub found in section 3.4.

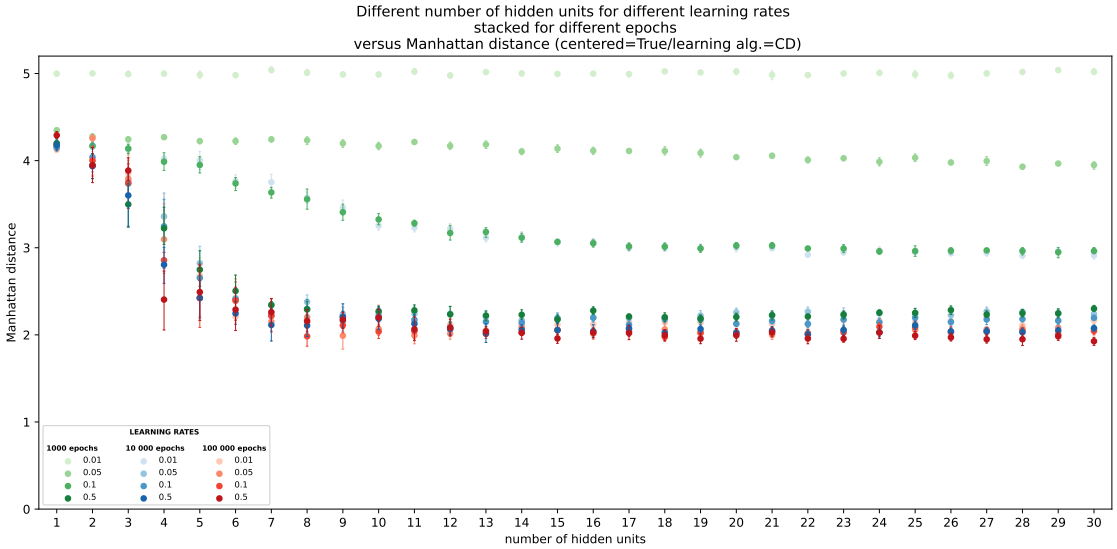


Figure 24: For different number of hidden units depicted on the x-as, the Manhattan distance is given depicted on the y-as. A centered RBM and Contrastive Divergence are used, and for each hidden unit 4 different learning rates are tested (0.01, 0.05, 0.1, and 0.5). The darker the color, the higher the learning rate. For each of these learning rates, different number of epochs are used: 1000 epochs (green), 10 000 epochs (blue), and 100 000 epochs (red).

When comparing these results with the ones for the non-redundant data in Figures 21 and 22, there are similarities. More epochs and higher learning rates tend to give a better communication, so lower Manhattan distance. However, faster than for the non-redundant data, the advantage of increasing the number of epochs and learning rate is less evident. For the non-redundant data using 1000 epochs (green), the 3 lowest learning rates give all around the same communicative success, and is very low. Here, there is a difference between those 3 epsilons, which give a lower Manhattan distance for higher learning rates. The Lowest learning rate gives around the same communicative success for 1000 epochs and 10 000 epochs in Figures 21 and 22. While the lowest rate for 10 000 epochs already gives the same Manhattan distance as the second highest learning rate of 1000 epochs. There is not a clear difference anymore for all the other learning rates and number of epochs. Summarizing, apart from the learning rates 0.01, 0.05, and 0.1 for 1000 epochs, and learning rate 0.01 for 10 000 epochs, the other combinations of learning rates and epochs give around the same Manhattan distance.

Comparing the two graphs, again PCD tends to be more error prone than CD. Looking at the graphs in the GitHub (section 3.4), the error of IPT tends to be more error prone than that of CD, PT again more than IPT, and PCD more than PT.

For the number of hidden units, something similar is visible as with the non-redundant data. A very low value for the number of hidden units results in a high Manhattan distance. In the beginning, increasing it makes a big difference in the positive sense (a lower Manhattan distance). From around a number of 6 hidden units on, increasing it does not have much of an impact.

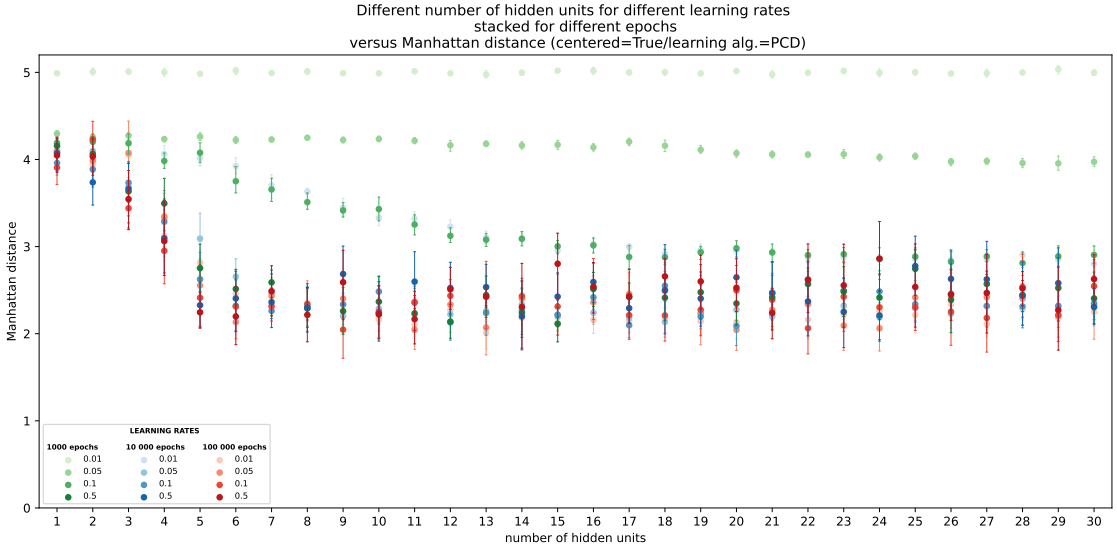


Figure 25: For different number of hidden units depicted on the x-as, the Manhattan distance is given depicted on the y-as. A centered RBM and Persistent Contrastive Divergence are used, and for each hidden unit 4 different learning rates are tested (0.01, 0.05, 0.1, and 0.5). The darker the color, the higher the learning rate. For each of these learning rates, different number of epochs are used: 1000 epochs (green), 10 000 epochs (blue), and 100 000 epochs (red).

4.1.2 Methods for initialising the weights and biases

To compare the results of using a centered RBM or not (initialise random or with zero), we set the epoch parameter to the highest value: 100 000. Also, the two are only compared for the Parallel Tempering learning algorithm as shown in Figure 26. The other combinations of epochs and learning algorithms give similar results. Lastly, a smaller range for the number of hidden units is used ($[1, 15]$), making it easier to look at it in more detail. However, the other number of hidden units give a similar result. This result is that there is no significant difference between using a centered RBM or a normal RBM. It can also be seen that the error becomes bigger when a higher learning rate is used. For the redundant data, there is also no significant difference between using a centered or normal RBM.

4.1.3 Learning algorithms

The four different learning algorithms are compared with the number of epoch set to 100 000 and the learning rate set to 0.5. The communicative success is compared for different hidden units and different learning algorithms. Both for a centered RBM (Figure 27) and a normal RBM (Figure 28). For the first, Persistent Contrastive Divergence (orange) comes out the best with 5 hidden units. The lowest value of the error is higher than the lowest value of all the other errors as depicted by the black line. Independent Parallel Tempering (red) with 9 hidden units comes out the best for the normal RBM.

In some cases, a lower learning rate performs better (directory 'population-communicative_success' in GitHub section 3.4) but generally this is not the case, and the best overall result is found with a learning rate of 0.5.

For the redundant data, the overall best performing parameter setting is for 100 000 epochs, 0.5 learning rate, a normal RBM, Contrastive Divergence and 22 hidden units. As can be seen in

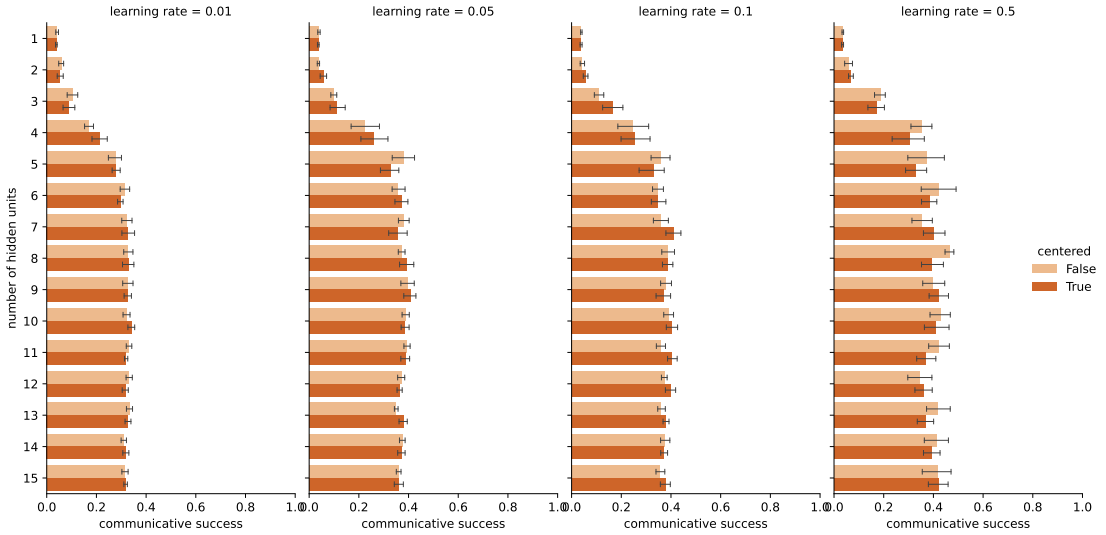


Figure 26: For different number of hidden units depicted on the y-as, the communicative success is given depicted on the x-as. The pretraining is done with 100 000 epochs and Parallel Tempering is used for training. The columns represent the 4 different learning rates (0.01, 0.05, 0.1, and 0.5). The dark color represents the communicative success when using a centered RBM and the light color that for the normal RBM.

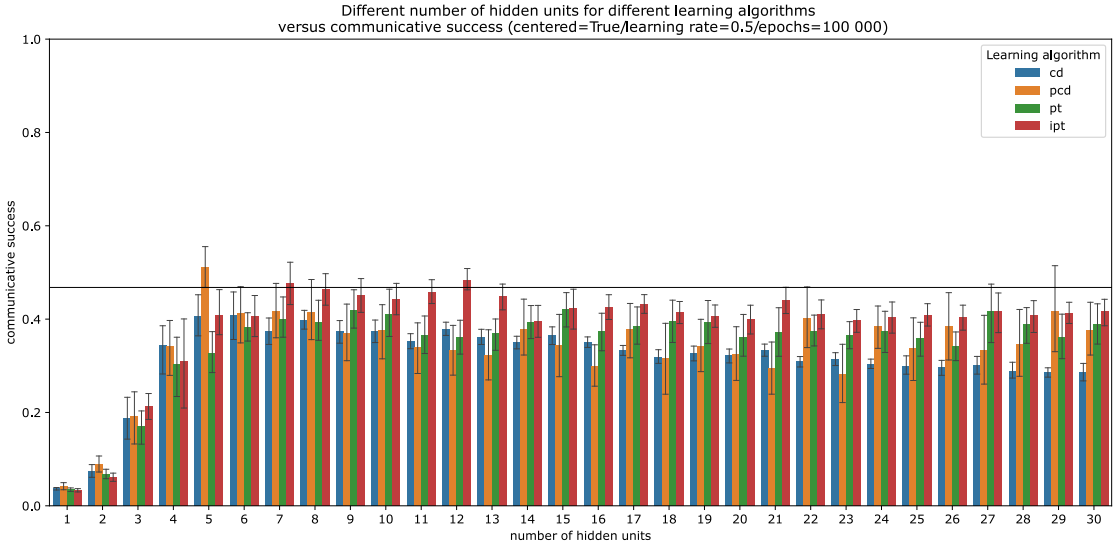


Figure 27: For different number of hidden units depicted on the x-as, the communicative success is given depicted on the y-as. A centered RBM, 100 000 epochs, and a 0.5 learning rate are used. For each hidden unit, the four learning algorithms are compared: Contrastive Divergence (blue), Persistent Contrastive Divergence (orange), Parallel Tempering (green), and Independent Parallel Tempering (red). The black line shows the lowest value of the error of the best performing parameters.

Figure 29, depicted by the black line, the highest value of the error is lower than the highest value of all the other errors. And this is also lower than that for the other combinations of learning algorithms and initialisation methods of the weights and biases (GitHub section 3.4 directory 'population-Manhattan_distance').

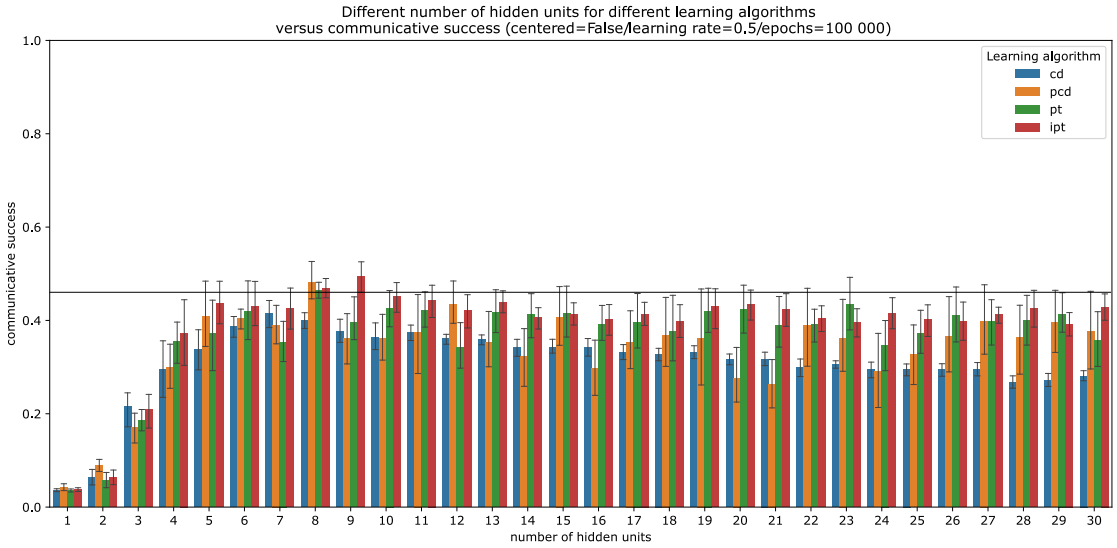


Figure 28: For different number of hidden units depicted on the x-as, the communicative success is given depicted on the y-as. A normal RBM, 100 000 epochs, and a 0.5 learning rate are used. For each hidden unit, the four learning algorithms are compared: Contrastive Divergence (blue), Persistent Contrastive Divergence (orange), Parallel Tempering (green), and Independent Parallel Tempering (red). The black line shows the lowest value of the error of the best performing parameters.

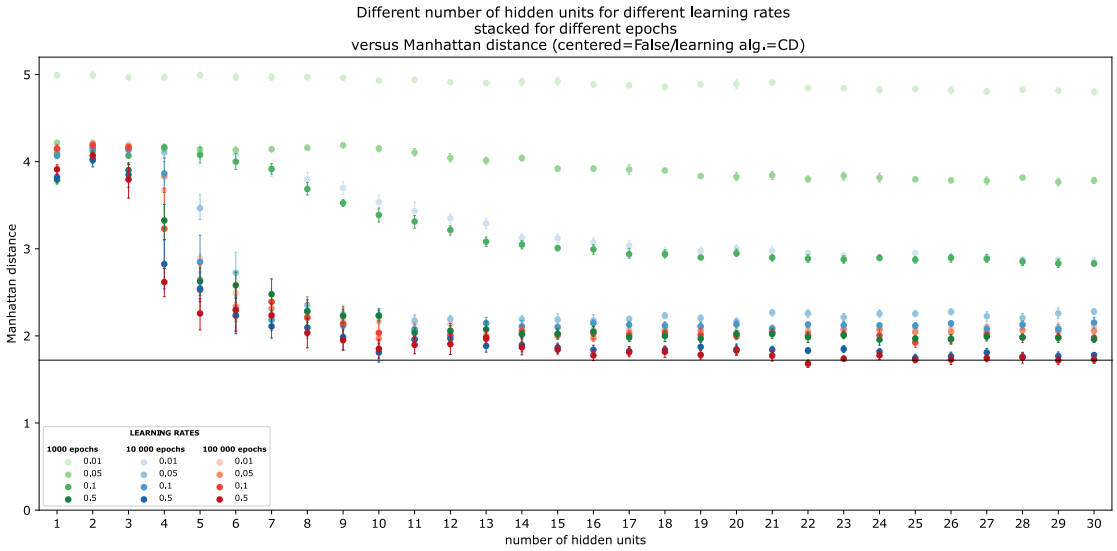


Figure 29: For different number of hidden units depicted on the x-as, the Manhattan distance is given depicted on the y-as. A normal RBM and Contrastive Divergence is used, and for each hidden unit 4 different learning rates are tested (0.01, 0.05, 0.1, and 0.5). The darker the color, the higher the learning rate. For each of these learning rates, different number of epochs are used: 1000 epochs (green), 10 000 epochs (blue), and 100 000 epochs (red). The black line shows the highest value of the error of the best performing parameters.

4.1.4 Comparing the errors of different design choices

As a design choice, one RBM was initialised and pretrained and then copied for all the agents in the population. With this method the results above were computed. However, a less error prone

solution could have been to initialise and train a new RBM for every agent, but this also gives a slower performance. In Figure 30, for the non-redundant data, the latter is depicted by the light purple, and the method used above is depicted by the dark purple. Different initialisation methods are also compared. Before, we always initialised the population with 10 agents and let each of them initiate 100 interactions. On the x-as it can be seen how the communicative success differs when both or one of them are increased by a tenfold. One setting of the parameter values is shown, but something similar happens for the other combinations of parameter values.

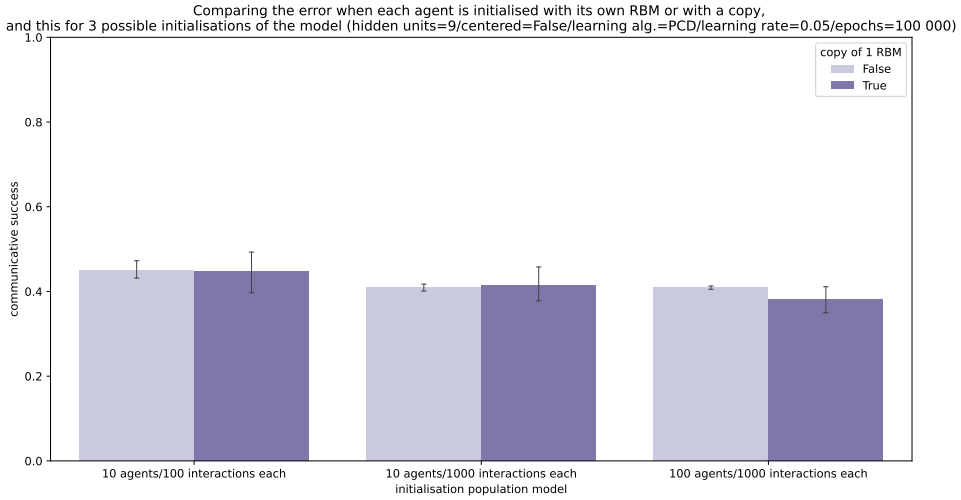


Figure 30: The communicative success (y-as) is compared for different initialisation methods (x-as). Initialising the population with 10 agents, and letting each agent initiate 100 or 1000 interactions. Another scenario is were the population is initialised with 100 people and letting each of them do 1000 interactions. A comparison is made between using 1 copy for all the agents or letting each agent have their own initialised and trained RBM before interacting. The following parameter values for the RBM and training are used: 9 hidden units, a normal RBM, PCD with a 0.05 learning rate and 100 000 epochs.

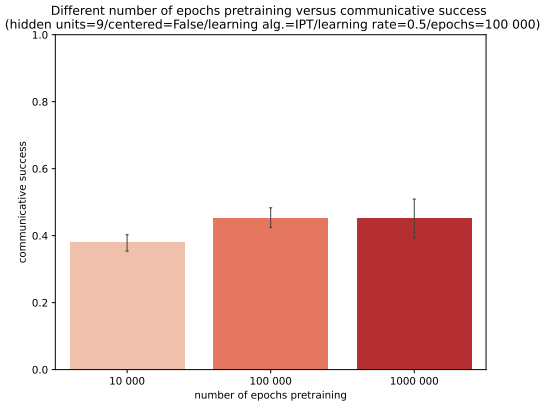
The error indeed seems to be bigger for the method used above. A difference can also be seen when changing the initialisation method. Showing a lower communicative success when the number of interactions that each agent initiates is increased. Increasing the number of people in the population does not give a big difference.

4.1.5 Comparing more values of the number of epochs pretraining

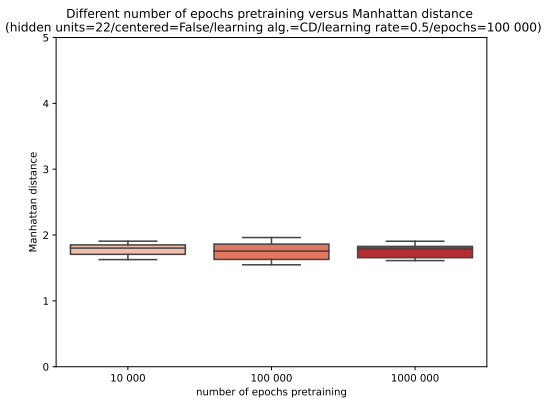
In Figure 31, an extra number of epochs pretraining (dark red) is compared against two other values from above. Both for the non-redundant and redundant data. There is no big increase of the communicative success or decrease of the Manhattan distance. Here, it can be seen that for the non-redundant data, going from 10 000 epochs to 100 000 epochs is still useful. However, for the redundant data, this is less visible because of the bigger error.

4.1.6 Comparing more values of the learning rate and different methods of using this rate during pretraining and training

Using one scenario's that came out best in the results above, the learning rate is investigated in more detail. The one with IPT (with 9 hidden units, a normal RBM, and 100 000 epochs



(a) The following parameter values for the RBM and training are used: 9 hidden units, a normal RBM, and IPT with 0.5 learning rate.



(b) The following parameter values for the RBM and training are used: 22 hidden units, a normal RBM, and CD with 0.5 learning rate.

Figure 31: Different number of epochs pretraining versus one of the evaluation measures of the communication: (a)non-redundant data and (b) redundant data

of pretraining) is used because it is less error prone than PCD. More values of the learning rate are compared because in section 2.3.1 it was mentioned that gradient descent is highly sensitive for this learning rate. Secondly, different methods are investigated for each of these learning rates. The first method is the one used above, where the given value of the learning rate is used for pretraining and for training during the interactions. Secondly, a method is introduced in which the given value for the learning rate is used for pretraining. However, for training during interactions the default learning rate from PyDeep is used: 0.01. This method is introduced because in Figure 30 a higher number of interaction initiations per person gives a lower communicative success. This could be the result of the learning rate being too high. Figure 32 shows the result for these different learning rate values, and two different methods for a model initialised with 10 agents, each initiating 100 interactions. The same is shown in Figure 33 but for each of the 10 agents initiating 1000 interactions.

The second method indeed helps for the loss of communicative success when increasing the initiated interactions per agent. In Figure 33, the communicative success of the second method stays stable (light green), unlike for the bigger learning rates of the first method (dark green). There is no big difference between the methods in Figure 32, and 0.5 still seems a good learning rate. Although other learning rates also seem to give a similar communicative success.

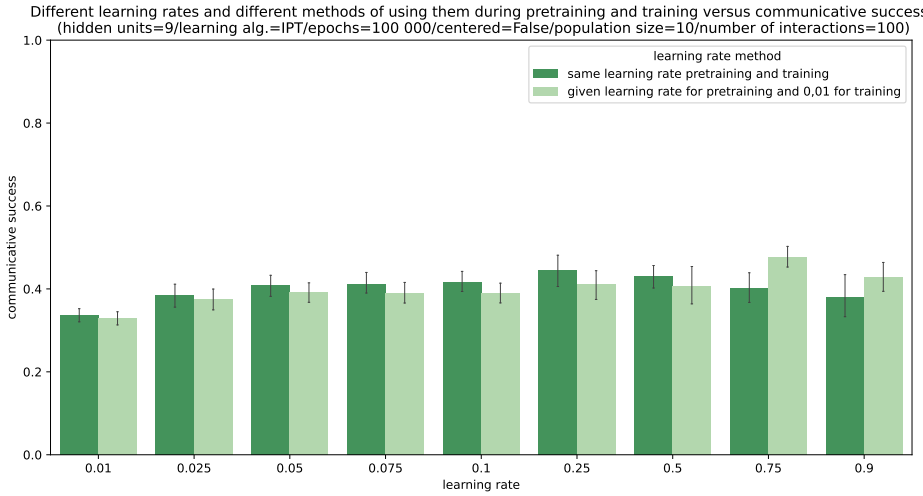


Figure 32: Different learning rates (0.01, 0.025, 0.05, 0.075, 0.1, 0.25, 0.5, 0.75, and 0.9) on the x-as and the communicative success on the y-as. For each learning rate two methods for using these learning rates are depicted. One which uses the given learning rate for pretraining and training during interaction. The other, which uses the given learning rate for pretraining but uses a 0.01 learning rate for training during interactions. The following parameter values for the RBM and training are used: 9 hidden units, a normal RBM, IPT and 100 000 epochs. The model is initialised with 10 agents, each initiating 100 interactions.

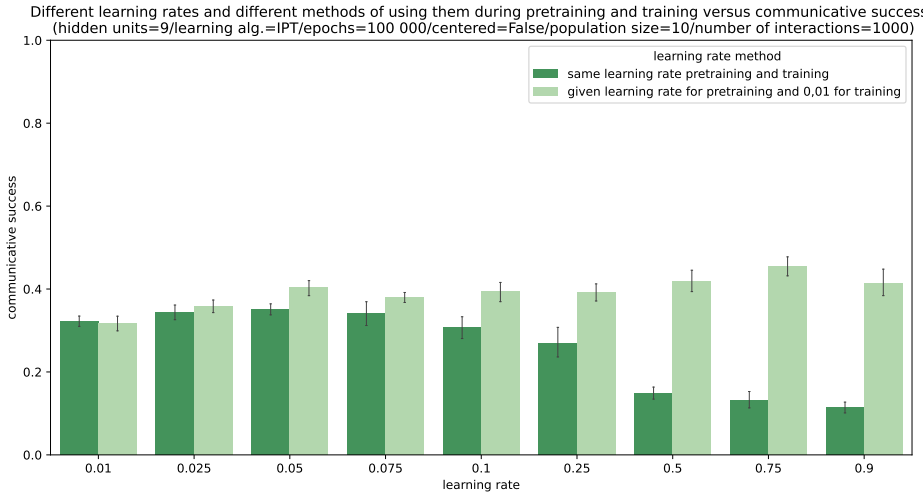


Figure 33: Different learning rates (0.01, 0.025, 0.05, 0.075, 0.1, 0.25, 0.5, 0.75, and 0.9) on the x-as and the communicative success on the y-as. For each learning rate two methods for using these learning rates are depicted. One which uses the given learning rate for pretraining and training during interaction. The other, which uses the given learning rate for pretraining but uses a 0.01 learning rate for training during interactions. The following parameter values for the RBM and training are used: 9 hidden units, a normal RBM, IPT and 100 000 epochs. The model is initialised with 10 agents, each initiating 1000 interactions.

4.1.7 Intermediate interactions in one run

To see what happens during one run, when the agents interact with each other, the intermediate communicative success for different time steps are depicted in Figure 34 and 35. The time steps

are represented by the number of interactions already done. This is done for the two methods described in section 4.1.6 and with 9 hidden units, a normal RBM, IPT with 0.5 learning rate, and 100 000 epochs of pretraining. Figure 34 shows this for a model initialised with 10 agents, each initiating 100 interactions. The same is shown in Figure 35 but for each of the 10 agents initiating 1000 interactions.

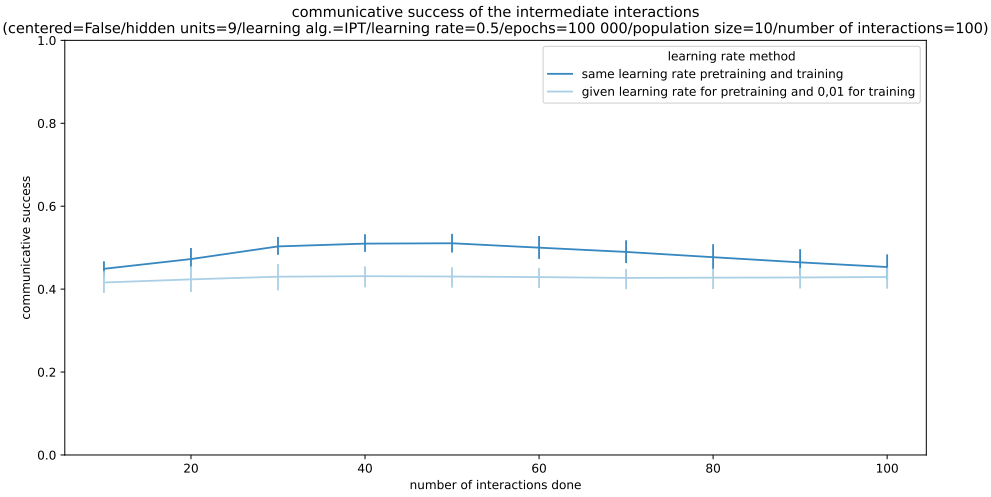


Figure 34: The different time steps, represented by the amount of interactions already done in one run, are depicted on the x-as. On the y-as the communicative success for each of the x values is shown. This is done for 9 hidden units, a normal RBM, IPT with 0.5 learning rate, 100 000 epochs of pretraining, and a model initialised with 10 agents, each initiating 100 interactions. It is repeated for 2 methods. One which uses the given learning rate for pretraining and training during interaction. The other, which uses the given learning rate for pretraining but uses a 0.01 learning rate for training during interactions.

The figures show in more detail what the results from section 4.1.6 showed. Using a relative high epsilon for training during interactions, is fine if an agent initiates a limited number of interactions (100). However, it can cause a rapid decrease in communicative success everytime more interactions are done. The method of using a 0.01 learning rate for training during interactions (light blue) has a positive effect on the communicative success for higher number of interactions initiated per agent.

Two similar figures are shown for the redundant data: Figure 36 and Figure 37. Here, letting an agent initiate 1000 interactions instead of 100 is not really worse for the outcome. The Manhattan distance grows a bit but there is no big difference between the two learning rate methods.

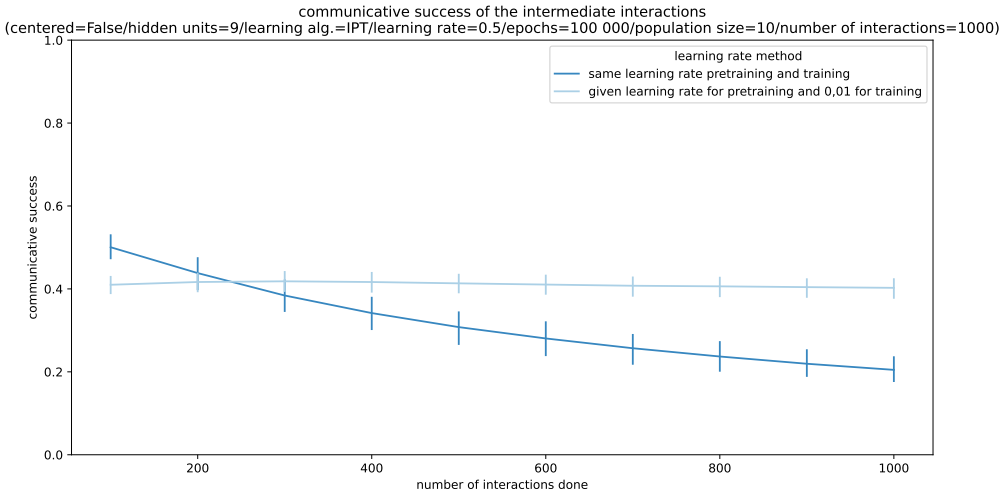


Figure 35: The different time steps, represented by the amount of interactions already done in one run, are depicted on the x-as. On the y-as the communicative success for each of the x values is shown. This is done for 9 hidden units, a normal RBM, IPT with 0.5 learning rate, 100 000 epochs of pretraining, and a model initialised with 10 agents, each initiating 1000 interactions. It is repeated for 2 methods. One which uses the given learning rate for pretraining and training during interaction. The other, which uses the given learning rate for pretraining but uses a 0.01 learning rate for training during interactions.

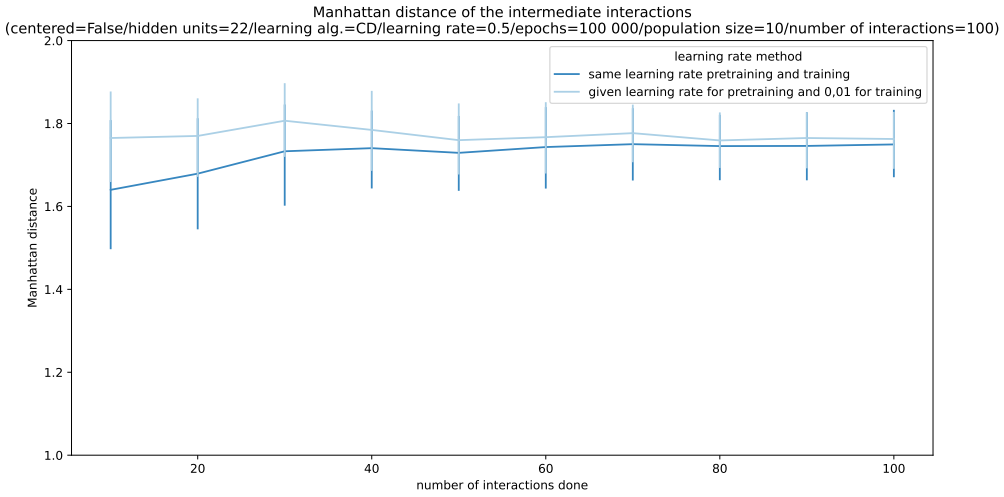


Figure 36: The different time steps, represented by the amount of interactions already done in one run, are depicted on the x-as. On the y-as the Manhattan distance for each of the x values is shown. This is done for 22 hidden units, a normal RBM, CD with 0.5 learning rate, 100 000 epochs of pretraining, and a model initialised with 10 agents, each initiating 100 interactions. It is repeated for 2 methods. One which uses the given learning rate for pretraining and training during interaction. The other, which uses the given learning rate for pretraining but uses a 0.01 learning rate for training during interactions.

.

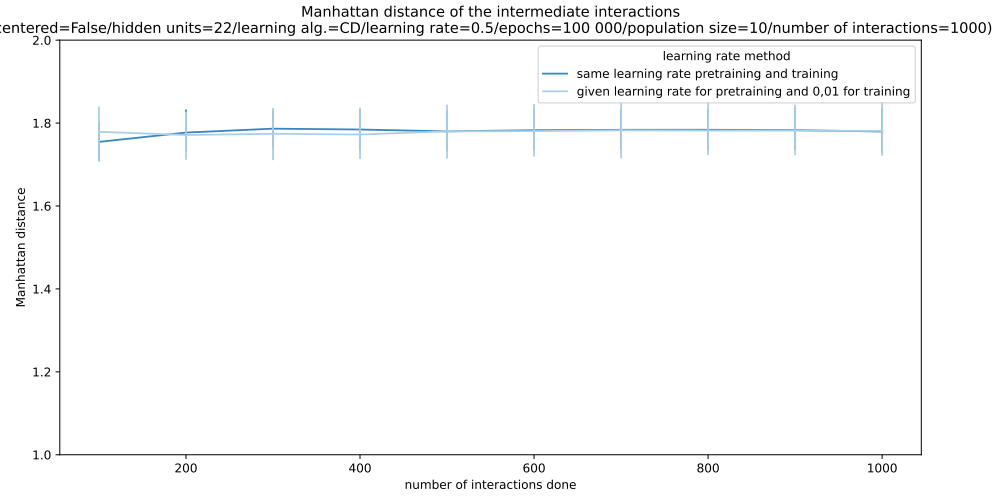


Figure 37: The different time steps, represented by the amount of interactions already done in one run, are depicted on the x-as. On the y-as the Manhattan distance for each of the x values is shown. This is done for 9 hidden units, a normal RBM, IPT with 0.5 learning rate, 100 000 epochs of pretraining, and a model initialised with 10 agents, each initiating 1000 interactions. It is repeated for 2 methods. One which uses the given learning rate for pretraining and training during interaction. The other, which uses the given learning rate for pretraining but uses a 0.01 learning rate for training during interactions.

4.1.8 Complexity

To investigate the complexity of the language, each agent is given all the concepts from our data once. The resulting signals are gathered from all the agents and for each signal from our data its appearance in these gathered signals is counted. As before, the run is repeated 10 times, and the numbers in the table are an average. The maximum number of counts that can be found is the number of agents in the population, times the number of concepts in our data (14). A signal belonging to more than one concept (for example the suffix m-) will correspond with the total count of all these different concepts.

The complexity of the language before the interactions and after them is compared. The method above is applied before the first interaction is done and another time after the last interaction is done. This comparison is done for the two learning rate methods as described in section 4.1.7 and 4.1.6, which is in its turn done for the non-redundant data and the redundant data. This can be seen in Table 5 and 6, showing the counts for an initialisation of 10 agents and accordingly 100 or 1000 initiated interactions by each agent.

In the tables, the signal count before and after the interactions is quite similar. The exception is for the non-redundant data using method 1 (using the given learning rate for pretraining and training during interaction), and this in both tables. The count after the interactions is lower, and in Table 6, the suffixes do not even exist anymore.

		communicative success				Manhattan distance			
		method 1		method 2		method 1		method 2	
signal	array	count before run	count after run	count before run	count after run	count before run	count after run	count before run	count after run
k-	[0. 1. 0. 0. 0. 0. 0.]	16,6	17,9	17,1	16,9	11	12,8	12,2	12,6
m-	[0. 0. 1. 0. 0. 0. 0.]	25,6	20,7	27,6	29,2	18,4	17,9	17	17,3
n-	[0. 0. 0. 1. 0. 0. 0.]	12,6	14,9	14,2	12,7	12,5	11,7	11,4	12,8
t-	[0. 0. 0. 0. 1. 0. 0.]	13,8	15,3	13,8	12	12,2	12,5	12,6	11,9
r-	[0. 0. 0. 0. 0. 1. 0.]	10,1	8	10,9	11,5	5	4,9	5,6	5,1
-kðn	[1. 1. 0. 0. 0. 0. 0.]	7,9	4,4	8,1	8	12,2	12,5	10,9	10,8
-ko,-no	[1. 0. 1. 0. 0. 0. 0.]	10,8	9	12,4	11,9	8	7,9	8,2	7,9
-na, -nðn	[1. 0. 0. 1. 0. 0. 0.]	7	4,5	6,4	7,7	6,5	6,9	7,4	5,7
-te	[1. 0. 0. 0. 1. 0. 0.]	6,9	5	6,1	7,5	6,6	6,7	6	6,7
-ke, -ne	[1. 0. 0. 0. 0. 1. 0.]	2,6	1,1	2,5	2,6	3,8	4,2	3,5	3,5
-ka	[1. 0. 0. 0. 0. 0. 1.]	1,8	1,3	1,3	2,6	3,3	3,3	3,4	3,9

Table 5: The average signal count when all the concepts were given once, before and after the interactions in one run. This for 2 methods: one uses the given learning rate for pretraining and training during interaction, and the other uses the given learning rate for pretraining but uses a 0.01 learning rate for training during interactions. All of this is repeated for the non-redundant and the redundant data (this table only shows the non-redundant array for clarity). The parameter values from Figure 34 and 36 are used. The models resulting in these number were initialised with 10 agents, each initiating 100 interactions.

4.2 Iterated learning

For one run the agent-based model uses 3 agents (generations) and if the agent needs to train the next agent, it will do this for the amount of epochs given as parameter. This is also used to pretrain the initial agent. Every time a run is repeated, a new RBM is initialised and pretrained for the initial agent.

The same values of the parameters are used as discussed in section 4.1.1. Except for the number of epochs however, only the value 1000 is done here. This is enough to get a general view and see if iterated learning requires different parameter setting of the RBMs than a population. Something extra was introduced for this agent-based model: the communicative success and Manhattan distance is calculated by the initial agent and by the previous agent (the second last agent), section 3.2.5 describes how they do this.

An experiment is done for each combination of all these values. For clarity only a subsection of the results belonging to these combinations is shown here. The interested reader can find all the graphs in the 'iterated_learning-communicative_success' map in the GitHub in section 3.4.

4.2.1 Hidden units and learning rates

The results in Figures 38 and 39 were gathered using the non-redundant data, and therefore the communicative success is calculated. The higher it is, the better the quality of the communication. As for the population model, only the Contrastive Divergence and Persistent Contrastive

		communicative success				Manhattan distance			
		method 1		method 2		method 1		method 2	
signal	array	count before run	count after run	count before run	count after run	count before run	count after run	count before run	count after run
k-	[0. 1. 0. 0. 0. 0. 0.]	18,4	0,4	17,7	18,6	11,5	12	12,5	12,7
m-	[0. 0. 1. 0. 0. 0. 0.]	27,9	1,6	27,2	26,9	19,1	18,8	19,1	19,4
n-	[0. 0. 0. 1. 0. 0. 0.]	13,9	4,5	14,9	14,5	12,5	13,3	12,8	12,3
t-	[0. 0. 0. 0. 1. 0. 0.]	13,7	1,2	13,2	15,4	12,6	13,6	12,3	12,2
r-	[0. 0. 0. 0. 0. 1. 0.]	9,7	0,2	7,3	8,7	5,4	5,1	6,6	5
-kðn	[1. 1. 0. 0. 0. 0. 0.]	8,2	0	7,5	5,8	10,6	11,5	11,4	11,7
-ko,-no	[1. 0. 1. 0. 0. 0. 0.]	11	0	11,2	12	8,5	8,1	7,6	7,2
-na, -nðn	[1. 0. 0. 1. 0. 0. 0.]	6,3	0	5,2	5,2	5,9	5,5	5,4	6,1
-te	[1. 0. 0. 0. 1. 0. 0.]	6,2	0	6,8	4,9	6,1	5,2	5,6	6
-ke, -ne	[1. 0. 0. 0. 0. 1. 0.]	3,9	0	3,8	3	4,2	3,4	3,1	3,8
-ka	[1. 0. 0. 0. 0. 0. 1.]	2,8	0	1,6	2,3	2,5	2,9	3,9	3,5

Table 6: The average signal count when all the concepts were given once, before and after the interactions in one run. This for 2 methods: one uses the given learning rate for pretraining and training during interaction, and the other uses the given learning rate for pretraining but uses a 0.01 learning rate for training during interactions. All of this is repeated for the non-redundant and the redundant data (this table only shows the non-redundant array for clarity). The parameter values from Figure 35 and 37 are used. The models resulting in these number were initialised with 10 agents, each initiating 1000 interactions.

Divergence using a centered RBM are depicted. When comparing the results of these figures with those of section 4.1.1 with 1000 epochs, they are similar. The Figures also show that there is not really a difference when the initial agent evaluates the last agent, or the previous agent does this.

Now the experiments from Figures 38 and 39 are also done for the redundant data, and therefore Manhattan distance is depicted on the y-as. The lower it is, the better. These results can be found in Figures 40 and 41. As for the non-redundant data experiments, these results are similar as those in section 4.1.1 for 1000 epochs. Evaluating with the initial or previous agent does not give a real difference, as was the case for communicative success. The PCD learning algorithm is again more error prone than the CD learning algorithm.

4.2.2 Intermediate time steps (each generation) in one run

To see what happens during one run, when the agents pass their knowledge on to the next agents, the intermediate communicative successes for different time steps are depicted in Figure 42 and 43. The time steps are represented by the number of generations where knowledge is already passed onto. The evaluation is done for the non-redundant data, letting both the initial agent and the previous agent evaluate. The same parameter values were used as for the population model: 9 hidden units, a normal RBM, and IPT with 0.5 learning rate. For every time step, the bars represent a different initialisation of the number of epochs pretraining for the initial agent and the number of training interactions when the knowledge of an agent is passed to the next

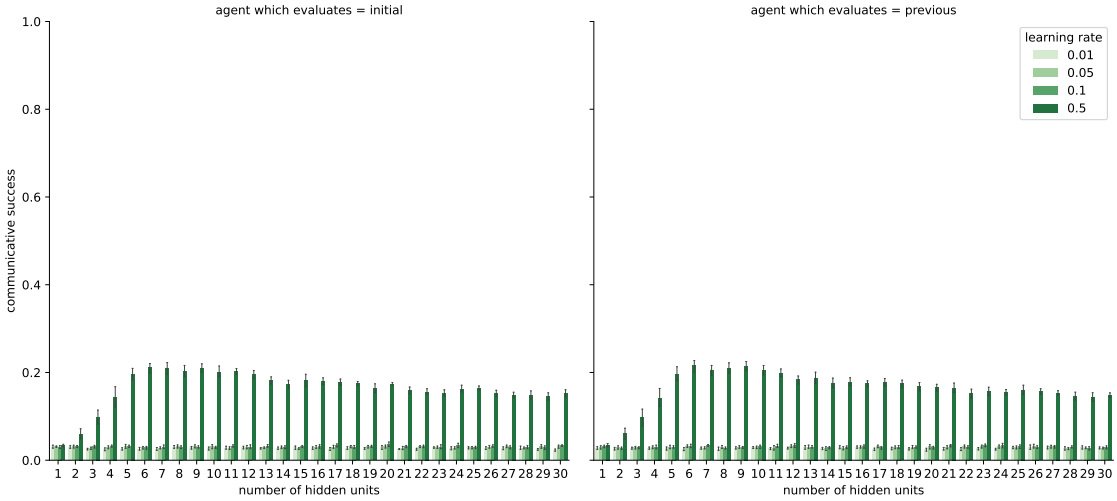


Figure 38: For different number of hidden units depicted on the x-as, the communicative success is given depicted on the y-as. A centered RBM, Contrastive Divergence, and 1000 epochs of pretraining are used. For each hidden unit 4 different learning rates are tested (0.01, 0.05, 0.1, and 0.5). The darker the color, the higher the learning rate. For the number of epochs pretraining 1000 is used.

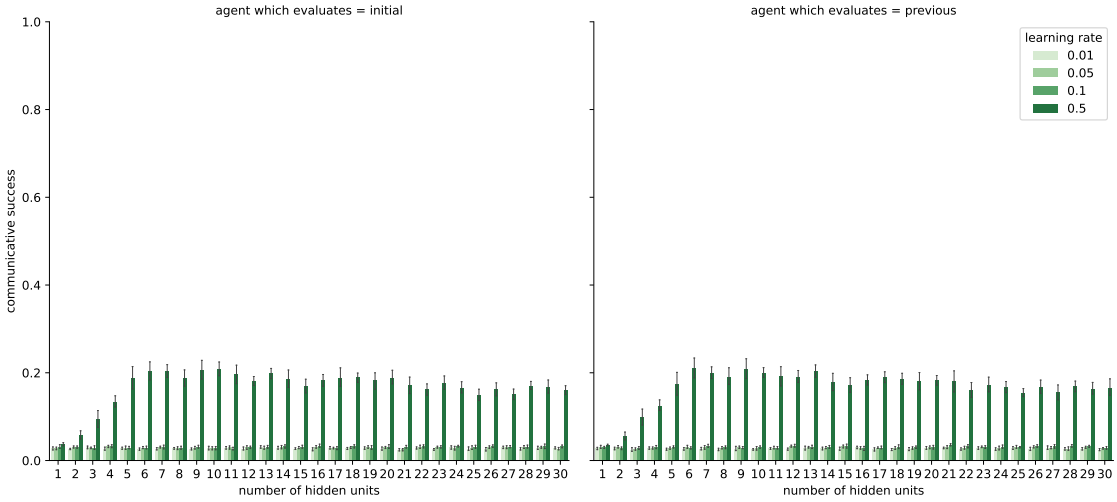


Figure 39: For different number of hidden units depicted on the x-as, the communicative success is given depicted on the y-as. A centered RBM, Persistent Contrastive Divergence and 1000 epochs of pretraining are used. For each hidden unit 4 different learning rates are tested (0.01, 0.05, 0.1, and 0.5). The darker the color, the higher the learning rate.

agent. Figure 42 shows the results using 3 generations (agents) as before, and Figure 43 shows the results using 6 generations.

Again there is not much difference between the initial agent evaluating and the previous agent. A positive effect can be seen when increasing the number of epochs pretraining for the initial agent and increasing the number of training interactions when the knowledge of an agent

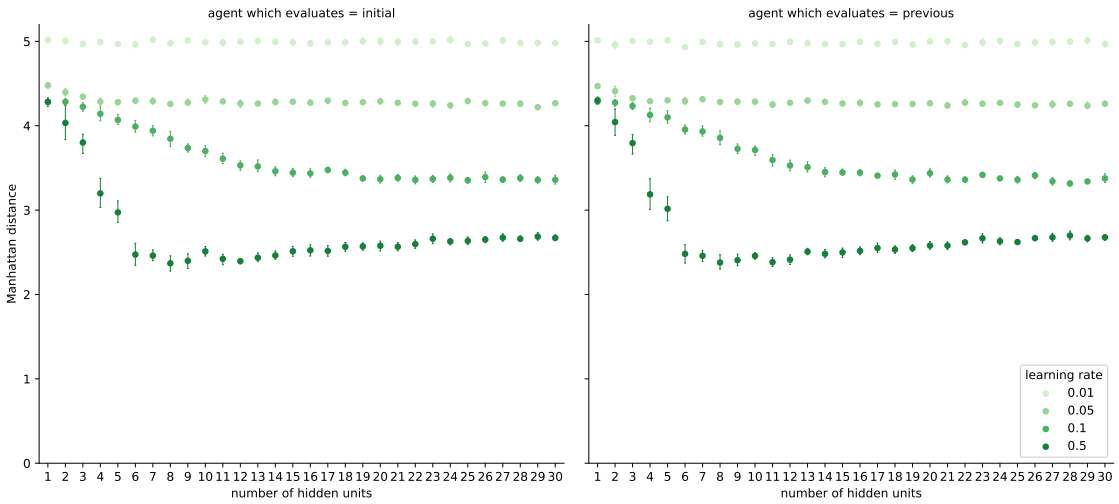


Figure 40: For different number of hidden units depicted on the x-as, the Manhattan distance is given depicted on the y-as. A centered RBM and Contrastive Divergence is used, and for each hidden unit 4 different learning rates are tested (0.01, 0.05, 0.1, and 0.5). The darker the color, the higher the learning rate. For each of these learning rates, different number of epochs are used: 1000 epochs (green), 10 000 epochs (blue), and 100 000 epochs (red). Both evaluation by the initial agent as the previous agent are depicted in the columns.

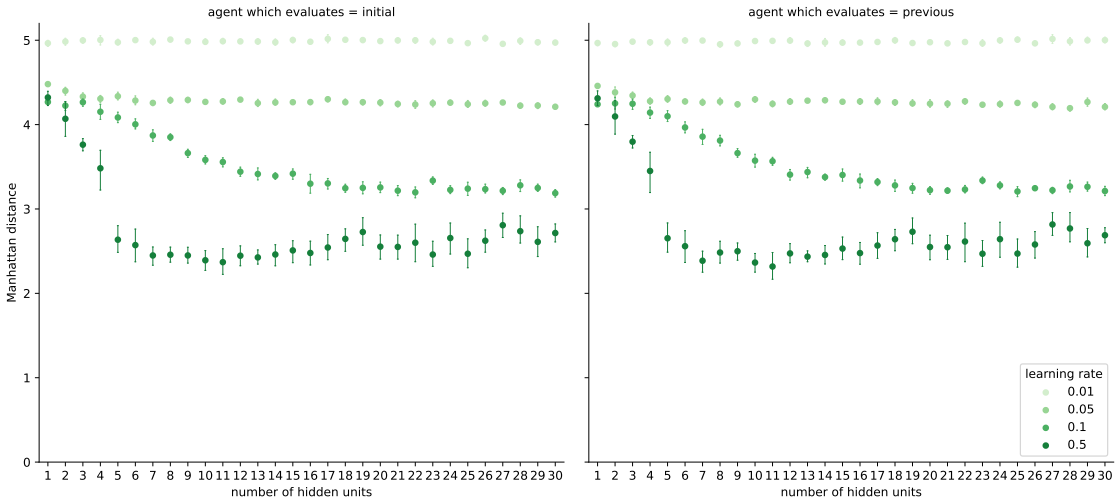


Figure 41: For different number of hidden units depicted on the x-as, the Manhattan distance is given depicted on the y-as. A centered RBM and Persistent Contrastive Divergence is used, and for each hidden unit 4 different learning rates are tested (0.01, 0.05, 0.1, and 0.5). The darker the color, the higher the learning rate. For each of these learning rates, different number of epochs are used: 1000 epochs (green), 10 000 epochs (blue), and 100 000 epochs (red). Both evaluation by the initial agent as the previous agent are depicted in the columns.

is passed to the next agent.

The same experiments are done for the redundant data, as shown in Figures 44 and 45, using the parameter values from the results on redundant data in the population model. There is no

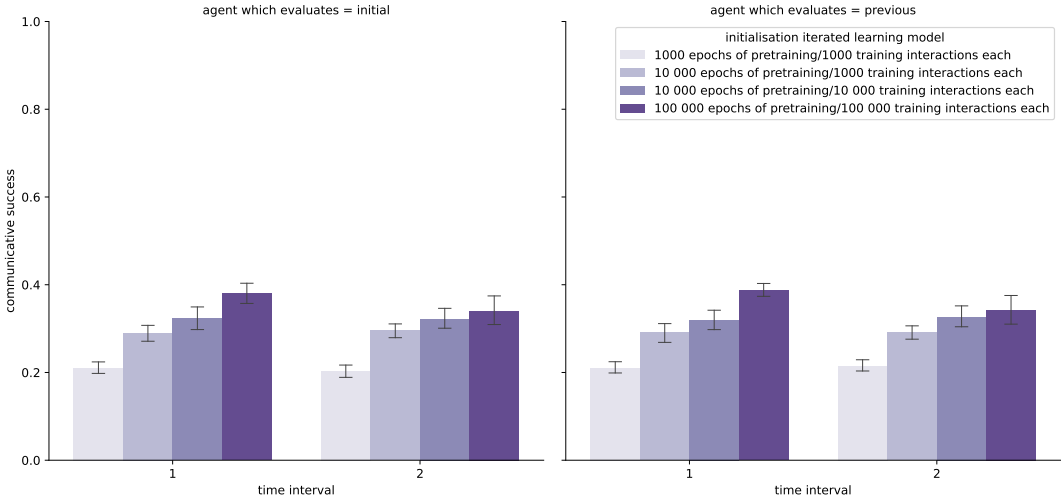


Figure 42: The time steps (the number of generations were knowledge is already passed onto) are depicted on the x-as, and the communicative success corresponding to such a step on the y-as. The following parameters are used: 9 hidden units, a normal RBM, IPT with 0.5 learning rate, and initialisation with 3 generations. Both evaluation by the initial agent as the previous agent are depicted in the columns. For each time step a different initialisation of the number of epochs pretraining for the initial agent and the number of training interactions when the knowledge of an agent is passed to the next agent is used. This is depicted by the purple bars.

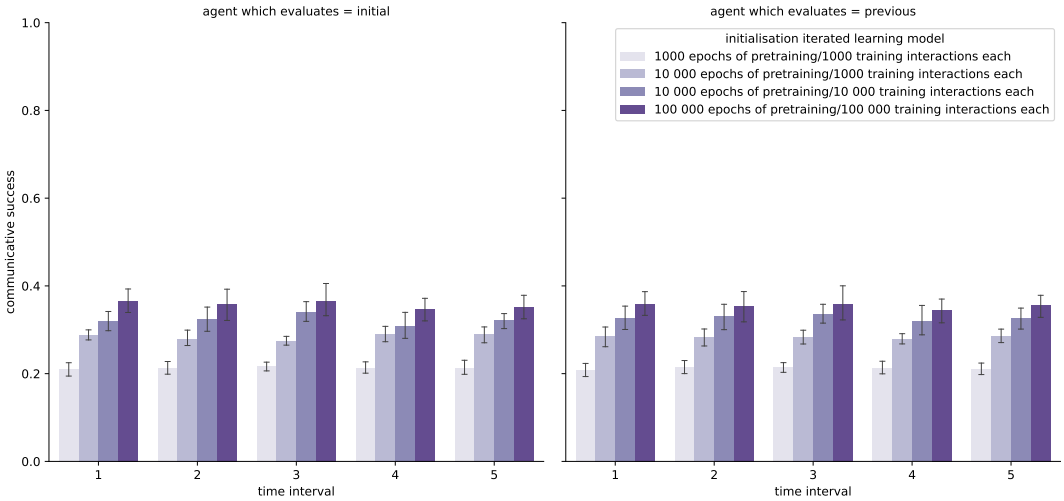


Figure 43: The time steps (the number of generations were knowledge is already passed onto) are depicted on the x-as, and the communicative success corresponding to such a step on the y-as. The following parameters are used: 9 hidden units, a normal RBM, IPT with 0.5 learning rate, and initialisation with 6 generations. Both evaluation by the initial agent as the previous agent are depicted in the columns. For each time step a different initialisation of the number of epochs pretraining for the initial agent and the number of training interactions when the knowledge of an agent is passed to the next agent is used. This is depicted by the purple bars.

clear advantage of increasing the number of epochs pretraining for the initial agent and increasing

the number of training interactions when the knowledge of an agent is passed to the next agent. It can only be concluded that 10 000 epochs of pretraining and 1000 training interactions for each agent is the best scenario. Again, no real difference can be seen between evaluation with the initial agent, and evaluation with the previous agent.

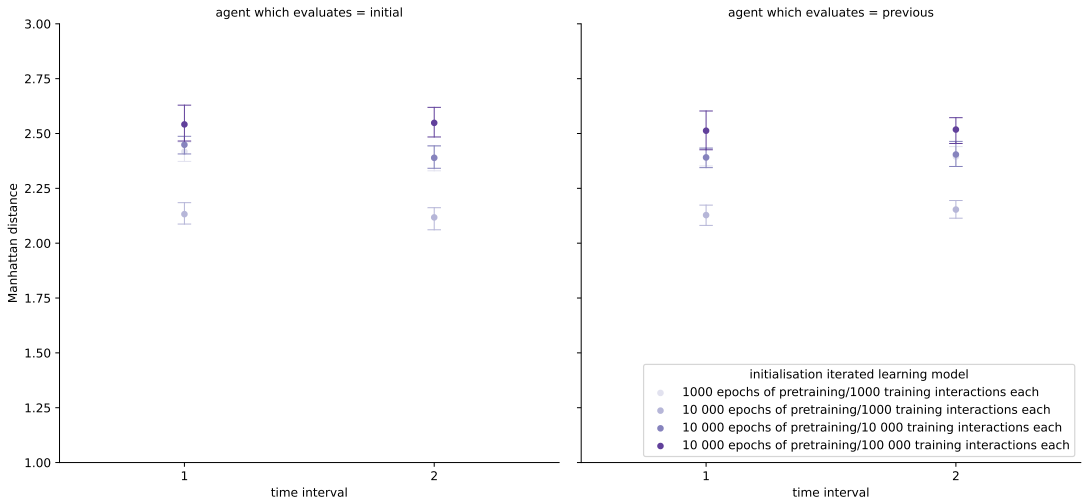


Figure 44: The time steps (the number of generations were knowledge is already passed onto) are depicted on the x-as, and the Manhattan distance corresponding to such a step on the y-as. The following parameters are used: 22 hidden units, a normal RBM, CD with 0.5 learning rate, and initialisation with 3 generations. Both evaluation by the initial agent as the previous agent are depicted in the columns. For each time step a different initialisation of the number of epochs pretraining for the initial agent and the number of training interactions when the knowledge of an agent is passed to the next agent is used. This is depicted by the purple bars.

4.2.3 Complexity

The complexity can be described as section 4.1.8, only now the maximum number of the total count can be 14, for the number of training data. Ideally each signal would have count 1, because only 1 generation gets the concept list once. Except for the signals that correspond to more concepts: idealy prefix 'm-' should have count 3 and prefix '-kən' count 2. For comparison, the first and last generation are picked and given this list of concepts. 10 000 epochs pretraining is chosen for the initial agent and 1000 training interactions when the knowledge of an agent is passed to the next agent.

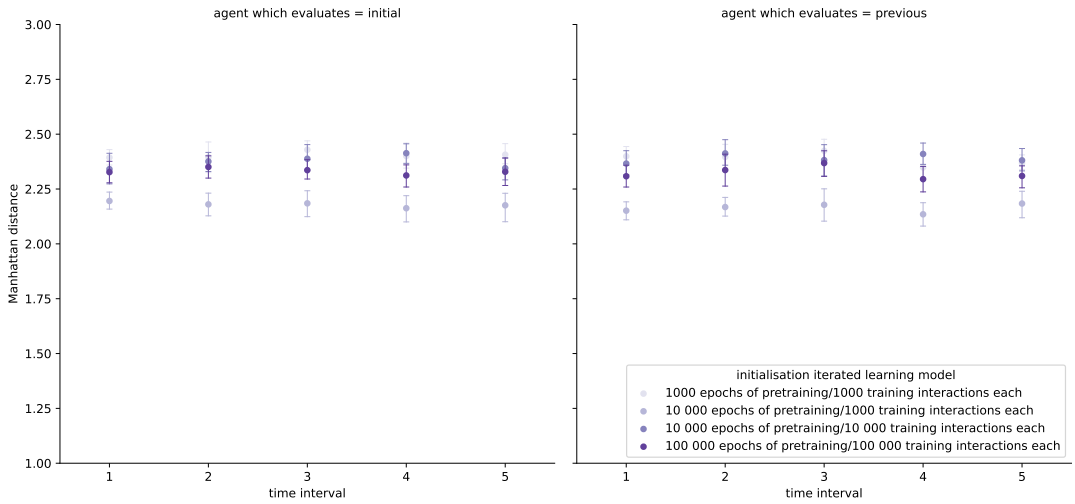


Figure 45: The time steps (the number of generations were knowledge is already passed onto) are depicted on the x-as, and the Manhattan distance corresponding to such a step on the y-as. The following parameters are used: 22 hidden units, a normal RBM, CD with 0.5 learning rate, and initialisation with 6 generations. Both evaluation by the initial agent as the previous agent are depicted in the columns. For each time step a different initialisation of the number of epochs pretraining for the initial agent and the number of training interactions when the knowledge of an agent is passed to the next agent is used. This is depicted by the purple bars.

5 Discussion

In this thesis Restricted Boltzmann machines are used to represent the production and comprehension models of the agents in agent-based models. In the long run, the goal of such a model is to study language change. RBMs have a nice feature so that they can complete partial data after they are trained properly. In our case a signal could be given back, given a concept or vice versa. First, the goal is to optimise the parameters of the RBMs so that they suit well for communication between agents. This is tackled in this thesis, and with the base this research provides, the use of RBMs to investigate language change could be investigated further.

Optimising the parameters of the RBM is done by running agent-based models in which agents play the language game, changing the values of the parameters, and looking at which ones give back the best results. These are represented by means of communicative success and Manhattan distance. Evaluating the quality of a parameter is done by using the feedback of the speaker, and comparing the original concept with the concept given back by the listener. Communicative success expects it to be the same, and increases a count for every concept that this applies to. Therefore, the higher the communicative success, the better the agents in the model communicate. For redundant data the difference between the two concepts is taken (Manhattan distance). Therefore, a small error is not punished completely, but more errors will result in a higher distance. The smaller the Manhattan distance, the better communication is between the agents.

Two agent-based models were introduced: population of agents and iterated learning. The first represents one generation with multiple agents, each with their own RBM, and it is investigated what happens when they interact with each other. In iterated learning, multiple generations are tried to be simulated, each generation represented by one agent. For this model, the goal is to see how well the data is learned from agent to agent. For both, values for the RBM

		communicative success				Manhattan distance			
		3 generations		6 generations		3 generations		6 generations	
signal	array	count before run	count after run	count before run	count after run	count before run	count after run	count before run	count after run
k-	[0. 1. 0. 0. 0. 0. 0.]	1,7	1,8	1,6	1,9	1,2	1,1	1,4	1
m-	[0. 0. 1. 0. 0. 0. 0.]	2,8	1,8	2,7	1,7	1,8	1,2	2,1	0,9
n-	[0. 0. 0. 1. 0. 0. 0.]	1,4	1,3	1,4	1,4	1	1,5	1,2	1,2
t-	[0. 0. 0. 0. 1. 0. 0.]	1,8	1,4	1,3	1,4	1,2	1,4	1,3	1,3
r-	[0. 0. 0. 0. 0. 1. 0.]	1,7	1,1	1,2	1,3	0,9	0,6	0,7	0,3
-kðn	[1. 1. 0. 0. 0. 0. 0.]	0,5	0,4	0,5	0,5	0,7	0	1	0
-ko,-no	[1. 0. 1. 0. 0. 0. 0.]	1,2	0,3	1,1	0,5	0,5	0,6	0,5	0,3
-na, -nðn	[1. 0. 0. 1. 0. 0. 0.]	0,6	0,5	0,6	0,4	0,9	0,1	0,7	0,2
-te	[1. 0. 0. 0. 1. 0. 0.]	0,3	0,3	0,5	0,1	0,7	0,1	0,6	0,2
-ke, -ne	[1. 0. 0. 0. 0. 1. 0.]	0,5	0,4	0,5	0	0,2	0	0,4	0
-ka	[1. 0. 0. 0. 0. 0. 1.]	0,2	0	0,3	0	0,3	0	0,3	0

Table 7: The average signal count when all the concepts were given once, before and after the interactions in one run. This for 3 generations and for 6. All of this is repeated for the non-redundant and the redundant data (this table only shows the non-redundant array for clarity). The parameter values from Figure 42 and 44 are used. The models resulting in these number were initialised with 10 000 epochs pretraining for the initial agent and 1000 training interactions when the knowledge of an agent is passed to the next agent.

parameters were varied and evaluated. One experiment repeats a run through the algorithm, initialising the agents and letting them interact 10 times to calculate the error. This needs to be done because an RBM is stochastic and will not always train exactly the same when given the same parameter values. For the population model an extra error may be introduced because of a design choice that was made. It entails that for one generation, one RBM is initialised and trained, and each agent starts the interactions with a copy of this RBM. It greatly improves the performance. When comparing the error of the two in the result section, although it indeed gives a greater error, the average value lays very close to the one which trains a separate RBM for every agent.

The following parameters of the RBM were investigated: the number of hidden units in the RBM (range 1 to 30), the way the weights and biases are initialised before training (zero or random), the learning algorithm ((P)CD/(I)PT) and learning rate (0.01, 0.05, 0.1, and 0.5) used while training, and the number of loops (epochs) through the data set to train with (1000, 10 000, and 100 000).

In both models, and for both redundant and non-redundant data, a very low number of hidden units gives very bad results. This way it is too hard for the RBM to learn general features of the data because those are not enough features to describe our data set. In these cases, increasing the number has a high positive impact on the communication. From around 6 hidden units for the non-redundant data, the impact is much smaller, and after increasing it even more, it has a small negative impact. When there are too many features for the data set, the RBM begins to learn useless things. The exact number of hidden units that should be used is not clear, as this

varies in our results, but it should be between 5 and 10. Higher is also possible but it does not add value and makes the performance of the RBM worse because more probabilities should be calculated during training and for sampling from the data distribution. For redundant data, this number lays a bit higher, the eventual optimal value was even 22. A possible explanation is that doubling the number of visible units, adds more information to the training data, and therefore more features are needed. However, the amount of information in the data set stays the same, as for the number of training samples in the set.

Most of the time initialisation has a big impact on how the training proceeds, as it impacts the range investigated for the weights and biases. It could also result in a global minimum instead of a local one. In our results there is not one way clearly better than the other for any model or any kind of data. Both initialising them to zero or random gives around the same communicative success or Manhattan distance.

For the non-redundant data, from the four learning algorithms (PCD/PCD/PT/IPT), there is not one clearly better than the others. Overall CD stays a bit lower but it is more predictable because it does not have many peeks and is the most error prone. All the others have more peaks, which means increasing the number of hidden units with 1 can change the communicative result drastically but not always in the positive sense. This is less reliable but after all our goal is to get a high communicative success. IPT for 9 hidden units and PCD for 5 hidden units gave back the highest results but due to the error this will not necessarily be the case if the experiments are redone with new RBMs. No fixed conclusion can be made about which learning algorithm to choose. It is a trade off between taking one with a higher error but more higher values, or one which is more error prone but maybe will not deliver as high communicative successes. Looking at the learning algorithms for the redundant data, there is also not one clearly better, but as with the non-redundant data some are more error prone. However, these results give CD back as the best algorithm, which is the algorithm with supposedly the worst approximation of the models distribution.

Overall, the higher the learning rate, the better. However, as seen in the results for the population model using the non-redundant data, this was highly depending on the number of interactions each agent in the model initiated. Increasing this number, gave worse results for higher learning rates. Therefore a method was introduced to separate the learning rate used while pretraining the RBM and the learning rate used while training during interactions. Setting the latter to the default of PyDeep 0.01, gave a more steady result when increasing the number of interactions each agent in the model initiates. This is a way of decreasing the learning rate at the end of training, where we see training during the interactions as the end of training the RBM. However, a very plausible reason why using a high number of the learning rate can yield in a worse communicative success is the fact that an RBM is stochastic. If it is not trained properly it will give back too varying results for one concept or one signal. This makes communication more difficult. For the redundant data, increasing the learning rate does not have as high an impact as for the non-redundant data. They all station around the same value quit quickly. Another difference from the non-redundant data is that a high learning rate does not have a big impact on the success when more training is done during interactions. It will even make the Manhattan distance go slower to the high Manhattan distance, which is the end result after all the interactions took place. A 0.01 learning rate for training during interactions arrives faster to this worse, higher Manhattan distance.

The last parameter is the number of epochs, or number of loops through the training data used for pretraining the RBM. For the population model with non-redundant data, a higher number of epochs gives a better result. Although, when getting to the high epochs (from 10 000) a tenfold will have less and less impact. This will already happen for a smaller number of epochs with the redundant data.

Iterated learning with the non-redundant data also has a positive effect when increasing the number of epochs of the initial agent. However, the other agents do not use the training data, so also do not use this number of epochs. They have there own number, the number of training interactions, which is similar and loops through data. Only this data does not come from a data set but from the knowledge of the previous agent. Increasing the number of loops also has a positive effect on the communicative success. For the non-redundant data this is again not really the case, increasing these number can even lead to higher Manhattan distance when looking at the results.

Looking at the results and ignoring the errors, each parameter depends on another parameter except for the number of epochs. It does not increase the communicative success evenly for each learning rate for example. However, when increasing it, it does not give a positive impact for one learning rate and a negative impact for another.

The tables with the complexity are a first step to slide into simulating the language change. For the population model, only for the non-redundant data, for a higher number (1000) of interactions initiated by each agent, and using the higher learning rate also for training during interactions, a loss of signals can be noticed. Overall it also seems to be more difficult for the RBM to interpret the extra bit that is turned on when a suffixes verb is represented. It has a lower count in the tables, both for the population of agents and iterated learning.

When the number of interactions and people in the population gets higher, the communicative success drops.

Overall it looks like the more epochs of pretraining, the higher the communicative success. Although the effect gets less big when the number of epochs gets higher. The centered RBM also does better overall than the RBM which initialises the weights and biases with zeros.

Based on the results we can conclude the best parameters to use for the RBMs used in our models. It outputs persistent contrastive divergence as the best learning algorithm with a learning rate of 0.05. Higher epochs and the centered RBM give the best results.

Iterated learning gives an overall low communicative succes, which probably depends on the amount of learning iterations with which the experiments were done. There is also not a big difference visible in the complexity of the language when comparing the first and last generation. This is probably the case because there are no other language speakers in this model.

The results only give an insight on the parameters chosen to vary, and even for them not all values and dependencies could have been investigated. It could be possible that varying other parameters gives a better result. The results also can not let us conclude if an RBM is a good fit for representing an agent who can both speak and listen. The characteristic of an RBM makes it an elegant solution but the fact that it is stochastic also makes for a lot of variation when giving the same concept (or signal) multiple times.

The models are able to complete the partial data, so fill in a signal given that this was set to zero and the concept part was filled in, or vice versa. However, it does not give the mapping from the training data often enough, but instead regularly returns a signal or concept that has no meaning in our data. The communicative success is not very high on average, not even half of the interactions in one run give a communicative success. The Manhattan distance also does not give ideal results, which would be between 0 and 1. However the best it gets for now is between 1 and 2.

Future work A couple of parameters were selected to experiment with, the ones that seemed most important according to the theory. It could however be useful to look at other parameters too. From the theory it was assumed that doing the Gibbs sampling in the contrastive divergence once should be enough. Doing more Gibbs sampling could improve the results because it should give a better approximation of the model distribution.

The most interesting thing to continue this research with, is adding a way to distinguish first language learners and second language learners. Finding a way to initialise an RBM with Lamaholot and a way to initialise it with the neighbouring languages. This is probably done, by abstracting less from the real language. An option could be too loose the binary RBM and use another type of training data with which a more complex language could be expressed. An option to stay closer to the current data is to find a way to involve an empty signal in the interactions. This would be an extra one-hot vector. It could be given while training but this is a bit unnatural. Another option is to add it as noise during interaction.

Including phonetics could help with simulating morphological simplification and trying to draw conclusions about it, as discussed in section 1.3. It could be done by using the ASCII values of the letters in the alphabet but this would take a lot of space. Then it is no longer possible to use one-hot vectors. A solution could be to only represent the affixes and the last letters of the verbs.

Although, clamping is now simulated manually, it could be interesting to add this to the implementation, or maybe new libraries will come out which do have this feature. However, it will not give a big difference because the inter layer nodes are independent.

6 Conclusion

Restricted Boltzmann machines have a very elegant feature that enables an agent to both be a speaker and listener. However, the fact that it is stochastic can be a downfall for this advantage. A lot of parameters can be varied however, so it is still possible it could improve in its communicative success. If this becomes true there is a high chance language change could be simulated with RBMs. Given that the data is represented differently, so enabling the fact that the agents can start with one of two languages: one for the mother language and another representing a second language learner. There is still a long way to go before RBMs can be used successfully for simulating morphological simplification.

References

Ackley, D. H., Hinton, G. E., & Sejnowski, T. J. (1985). A learning algorithm for boltzmann machines. *Cognitive science*, *9*(1), 147–169.

Aitchison, J. (2001). *Language change: Progress or decay?* Cambridge University Press.

Ash, R. B. (2008). *Basic probability theory*. Courier Corporation.

Atkinson, M., Smith, K., & Kirby, S. (2018). Adult learning and language simplification. *Cognitive science*, *42*(8), 2818–2854.

Bait, M., Folgieri, R., & Scarpello, O. (2015). The use of agent-based models in cognitive linguistics: An approach to chomsky's linguistics through the clarion model. *Journal of Foreign Language Teaching and Applied Linguistics*, *1*(3), 10–14706.

Barlow, H. B. (1989). Unsupervised learning. *Neural computation*, *1*(3), 295–311.

Basta, N. (2020, March). *The differences between sigmoid and softmax activation functions*. medium. https://medium.com/arteos-ai/the-differences-between-sigmoid-and-softmax-activation-function-12adee8cf322

Behera, D. K., Das, M., Swetanisha, S., & Sethy, P. K. (2021). Hybrid model for movie recommendation system using content k-nearest neighbors and restricted boltzmann machine. *Indonesian Journal of Electrical Engineering and Computer Science*, *23*(1).

Bengio, Y., & Delalleau, O. (2009). Justifying and generalizing contrastive divergence. *Neural computation*, *21*(6), 1601–1621.

Bloem, J., Versloot, A., & Weerman, F. (2015). The use of agent-based models in cognitive linguistics: An approach to chomsky's linguistics through the clarion model. *Journal of Foreign Language Teaching and Applied Linguistics*, *1*(3), 10–14706.

Boersma, P. (2019). Simulated distributional learning in deep boltzmann machines leads to the emergence of discrete categories. In *Proceedings of the 19th International Congress of Phonetic Sciences* (pp. 1520-1524).

Boltzmann, L. (1872). *Boltzmann equation* (Vol. 66). Sitzungsber Akade Wissen.

Bovet, D. P., Crescenzi, P., & Bovet, D. (1994). *Introduction to the theory of complexity* (Vol. 7). London: Prentice Hall.

Bynon, T. (1977). *Historical linguistics*. Cambridge University Press.

Chaabouni, R., Kharitonov, E., Dupoux, E., & Baroni, M. (2019). Anti-efficient encoding in emergent communication. *arXiv preprint arXiv:1905.12561*.

Chaabouni, R., Kharitonov, E., Dupoux, E., & Baroni, M. (2021). Communicating artificial neural networks develop efficient color-naming systems. *Proceedings of the National Academy of Sciences*, *118*(12).

Cho, K., Raiko, T., & Ilin, A. (2010, July). Parallel tempering is efficient for learning restricted boltzmann machines. In *The 2010 international joint conference on neural networks (ijcnn)* (pp. 1-8). IEEE.

Cope, D., & Schoots, N. (2021). Learning to communicate with strangers via channel randomisation methods. *arXiv preprint arXiv:2104.09557*.

Croft, W. (2008). Evolutionary linguistics. *Annual review of anthropology*, *37*, 219–234.

Cunningham, P., Cord, M., & Delany, S. J. (2008). Supervised learning. In *Machine learning techniques for multimedia* (pp. 21-49). Springer, Berlin, Heidelberg.

Dekker, P., & De Boer, B. (2020, December). Neural agent-based models to study language contact using linguistic data. In *4th NeurIPS Workshop on Emergent Communication: Talking to Strangers: Zero-Shot Emergent Communication*.

Dekker, P., Klamer, M., & de Boer, B. (2021). Analysing contact–induced simplification in alorese using agent–based models. In *54th Annual Meeting of the Societas Linguistica Europaea, 30 August – 3 September 2021*.

Desjardins, G., Courville, A., Bengio, Y., Vincent, P., & Delalleau, O. (2010, May). Parallel tempering for training of restricted boltzmann machines. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (pp. 145–152). Cambridge, MA: MIT Press.

Fahlman, S. E., Hinton, G. E., & Sejnowski, T. J. (1983). Massively parallel architectures for al: Netl, thistle, and boltzmann machines. In *National Conference on Artificial Intelligence, AAAI*.

Fischer, A., & Igel, C. (2012). An introduction to restricted boltzmann machines. In *Iberoamerican congress on pattern recognition* (pp. 14-36). Springer, Berlin, Heidelberg.

Freund, Y., & Haussler, D. (1994a). Unsupervised learning of distributions of binary vectors using two layer networks.

Freund, Y., & Haussler, D. (1994b). Unsupervised learning of distributions of binary vectors using two layer networks.

Gelfand, A. E. (2000). Gibbs sampling. *Journal of the American statistical Association*, *95*(452), 1300–1304.

Gilbert, N. (2008). *Agent-based models*. London: Sage.

Gilbert, N. (2019). *Agent-based models* (Vol. 153). Sage Publications.

Graesser, L., Cho, K., & Kiela, D. (2019). Emergent linguistic phenomena in multi-agent communication games. *arXiv preprint arXiv:1901.08706*.

Gurney, K. (2018). *An introduction to neural networks*. CRC press.

Han, Z., Liu, Z., Han, J., Vong, C. M., Bu, S., & Li, X. (2016). Unsupervised 3d local feature learning by circle convolutional restricted boltzmann machine. *IEEE Transactions on Image Processing*, *25*(11), 5331–5344.

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural computation*, *14*(8), 1771–1800.

Hinton, G. E. (2012). A practical guide to training restricted boltzmann machines. In *Neural networks: Tricks of the trade* (pp. 599-619). Springer, Berlin, Heidelberg.

Hinton, G. E., Osindero, S., & Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, *18*(7), 1527–1554.

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *science*, *313*(5786), 504–507.

Hinton, G. E., & Sejnowski, T. J. (1986). Learning and relearning in boltzmann machines. *Parallel distributed processing: Explorations in the microstructure of cognition*, *1*(2), 282–317.

Hinton, G. E., Sejnowski, T. J., & Ackley, D. H. (1984). *Boltzmann machines: Constraint satisfaction networks that learn*. Pittsburgh, PA: Carnegie-Mellon University, Department of Computer Science.

Hopfield, J. J. (2007). Hopfield network. *Scholarpedia*, *2*(5), 1977.

Hua, Y., Guo, J., & Zhao, H. (2015). Deep belief networks and deep learning. In *Proceedings of 2015 International Conference on Intelligent Computing and Internet of Things* (pp. 1-4). IEEE.

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

Jiang, Y., Xiao, J., Liu, X., & Hou, J. (2018, March). A removing redundancy restricted boltzmann machine. In *2018 Tenth International Conference on Advanced Computational Intelligence (ICACI)* (pp. 57-62). IEEE.

Jordan, M. I., & Bishop, C. (2004). *An introduction to graphical models.*

Kazil, J., Masad, D., & Crooks, A. (2020). Utilizing python for agent-based modeling: The mesa framework. In R. Thomson, H. Bisgin, C. Dancy, A. Hyder, & M. Hussain (Eds.), *Social, cultural, and behavioral modeling* (pp. 308–317). Springer International Publishing.

Ke, J., Gong, T., & Wang, W. S. (2008). Language change and social networks. *Communications in Computational Physics*, *3*(4), 935–949.

Kelley, H. J. (1962). Method of gradients. In *Mathematics in Science and Engineering* (Vol. 5, pp. 205-254). Elsevier.

Kharitonov, E., Chaabouni, R., Bouchacourt, D., & Baroni, M. (2019). Egg: A toolkit for research on emergence of language in games. *arXiv preprint arXiv:1907.00852*.

Kharitonov, E., Chaabouni, R., Bouchacourt, D., & Baroni, M. (2020). Entropy minimization in emergent languages. In *International Conference on Machine Learning* (pp. 5220-5230). PMLR.

Kharitonov, E., Dessì, R., Chaabouni, R., Bouchacourt, D., & Baroni, M. (2021). EGG: A toolkit for research on Emergence of lanGuage in Games.

Kirby, S., & Hurford, J. R. (2002). The emergence of linguistic structure: An overview of the iterated learning model. *Simulating the evolution of language*, 121–147.

Klamer, M. (2011). *A short grammar of alorese (austronesian).* Lincom Europa.

Klamer, M. (2012). *Papuan-austronesian language contact: Alorese from an areal perspective.* University of Hawai'i Press.

Kolenchery, G. (2015). Analytical components of morphology in linguistics. *Global English-Oriented Research Journal*, *1*(1), 161–166.

Krashen, S. D. (1982). *Child-adult differences in second language acquisition. series on issues in second language research.* Newbury House Publishers, Inc., Rowley, MA 01969.

Kusters, W. (2000). Morphological simplification: More than erosion? *Studies in Slavic and General Linguistics*, *28*, 225–230.

Labov, W. (2011). *Principles of linguistic change, volume 3: Cognitive and cultural factors* (Vol. 36). John Wiley; Sons.

Lazaridou, A., Potapenko, A., & Tieleman, O. (2020). Multi-agent communication meets natural language: Synergies between functional and structural language learning. *arXiv preprint arXiv:2005.07064*.

Le Cam, L. (1990). Maximum likelihood: An introduction. *International Statistical Review/Revue Internationale de Statistique*, 153–171.

Le Roux, N., & Bengio, Y. (2008). Representational power of restricted boltzmann machines and deep belief networks. *Neural computation*, *20*(6), 1631–1649.

LeCun, Y., Chopra, S., Hadsell, R., Ranzato, M., & Huang, F. (2006). Tutorial on energy-based learning. *Predicting structured data*, *1*(0).

Legendre, G., Miyata, Y., & Smolensky, P. (1990). Harmonic grammar—a formal multi-level connectionist theory of linguistic well-formedness: Theoretical foundations. In *Proceedings of the twelfth annual conference of the Cognitive Science Society* (pp. 884-891). Cambridge, MA: Erlbaum.

Lenneberg, E. H. (1967). The biological foundations of language. *Hospital Practice*, *2*(12), 59–67.

Li, F., & Bowling, M. (2019). Ease-of-teaching and language structure from emergent communication. *arXiv preprint arXiv:1906.02403*.

Lj, M. (2017, August). *Understanding softmax and the negative log-likelihood*. github. https://ljvmiranda921.github.io/notebook/2017/08/13/softmax-and-the-negative-log-likelihood/

Melchior, J. (2020). *Pydeep documentation*.

Midhun, M. E., Nair, S. R., Prabhakar, V. N., & Kumar, S. S. (2014, October). Deep model for classification of hyperspectral image using restricted boltzmann machine. In *Proceedings of the 2014 international conference on interdisciplinary advances in applied computing* (pp. 1-7).

Montúfar, G. (2016). Restricted boltzmann machines: Introduction and review. In *Information Geometry and Its Applications IV* (pp. 75-115). Springer, Cham.

Montúfar, G., & Ay, N. (2010). Refinements of universal approximation results for deep belief networks and restricted boltzmann machines. *arXiv preprint arXiv:1005.1593*.

Moro, F. R. (2019). Loss of morphology in alorese (austronesian): Simplification in adult language contact. *Journal of Language Contact*, *12*(2), 378–403.

Mu, Y., Ferrie, F., & Dimitrakopoulos, R. (2015). Sparse image reconstruction by two phase rbm learning: Application to mine planning. In *2015 14th IAPR International Conference on Machine Vision Applications (MVA)* (pp. 316-320). IEEE.

Musumeci, V. (2017, November). *How to handle the seo by markov chains*. vincenzomusumeci. http://www.vincenzomusumeci.com/findability-seo/how-to-handle-seo-by-markov-chains/

Nishiyama, K., & Kelen, H. (2007). *A grammar of lamaholot, eastern indonesia*. Lincom Europa.

Oppermann, A. (2018, May). *Deep learning meets physics: Restricted boltzmann machines part ii*. towardsdatascience. https://towardsdatascience.com/deep-learning-meets-physics-restricted-boltzmann-machines-part-ii-4b159dce1ffb

Osogami, T. (2006). Boltzmann machines and energy-based models. *arXiv preprint arXiv:1708.06008*.

pandas development team, T. (2020). *Pandas-dev/pandas: Pandas* (Version latest). Zenodo. https://doi.org/10.5281/zenodo.3509134

Pham, D. H., & Le, A. C. (2018). Exploiting multiple word embeddings and one-hot character vectors for aspect-based sentiment analysis. *International Journal of Approximate Reasoning*, *103*, 1–10.

Pham, H., Guan, M., Zoph, B., Le, Q., & Dean, J. (2018). Efficient neural architecture search via parameters sharing. In *International Conference on Machine Learning* (pp. 4095-4104). PMLR.

Pijpops, D., & Beuls, K. (2014). Agent-gebaseerde modellering in de historische taalkunde. *Handelingen-Koninklijke Zuid-Nederlandse Maatschappij voor Taal-en Letterkunde en Geschiedenis*, *68*, 5–23.

Radford, A., Atkinson, M., Britain, D., Clahsen, H., & Spencer, A. (2009). *Linguistics: An introduction*. Cambridge University Press.

Reimer, U., Emmenegger, S., Maier, E., Zhang, Z., & Khatami, R. (2017). Recognizing sleep stages with wearable sensors in everyday settings. In *ICT4AgeingWell* (pp. 172-179).

Ren, Y., Guo, S., Labeau, M., Cohen, S. B., & Kirby, S. (2020). Compositional languages emerge in a neural iterated learning model. *arXiv preprint arXiv:2002.01365*.

Roberts, S. (2013, April). *Iterated learning using youtube videos*. replicatedtypo. http://www.replicatedtypo.com/iterated-learning-using-youtube-videos/6093.html

Ruder, S. (2021). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Salakhutdinov, R., Mnih, A., & Hinton, G. (2007). Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning* (pp. 791-798).

Smith, A. D. (2014). Models of language evolution and change. *Wiley Interdisciplinary Reviews: Cognitive Science*, *5*(3), 281–293.

Smolensky, P. (1983). *Harmony theory: A mathematical framework for stochastic parallel processing*. CALIFORNIA UNIV SAN DIEGO LA JOLLA INST FOR COGNITIVE SCIENCE.

Smolensky, P., & Riley, M. S. (1984). *Harmony theory: Problem solving, parallel cognitive models, and thermal physics*. CALIFORNIA UNIV SAN DIEGO LA JOLLA INST FOR COGNITIVE SCIENCE.

Spring, D. (1985). On the second derivative test for constrained local extrema. *The American Mathematical Monthly*, *92*(9), 631–643.

Steels, L. (1997). Synthesising the origins of language and meaning using co-evolution, self-organisation and level formation. *Evolution of Human Language*.

Steels, L., & Loetzsch, M. (2012). The grounded naming game. *Experiments in cultural language evolution*, *3*, 41–59.

Tieleman, O., Lazaridou, A., Mourad, S., Blundell, C., & Precup, D. (2019). Shaping representations through communication: Community size effect in artificial learning systems. *arXiv preprint arXiv:1912.06208*.

Tieleman, T. (2008, July). Training restricted boltzmann machines using approximations to the likelihood gradient. In *Proceedings of the 25th international conference on Machine learning* (pp. 1064-1071).

Tomczak, J. M., & Zięba, M. (2015). Classification restricted boltzmann machine for comprehensible credit scoring model. *Expert Systems with Applications*, *42*(4), 1789–1796.

Ullrich, K., Meeds, E., & Welling, M. (2017). Soft weight-sharing for neural network compression. *arXiv preprint arXiv:1702.04008*.

Van Ravenzwaaij, D., Cassey, P., & Brown, S. D. (2018). A simple introduction to markov chain monte–carlo sampling. *Psychonomic bulletin & review*, *25*(1), 143–154.

Vrábel, J., Pořízka, P., & Kaiser, J. (2020). Restricted boltzmann machine method for dimensionality reduction of large spectroscopic data. *Spectrochimica Acta Part B: Atomic Spectroscopy*, *167*, 105849.

Wang, H., & Raj, B. (2017). On the origin of deep learning. *arXiv preprint arXiv:1702.07800*.

Waskom, M., Botvinnik, O., O'Kane, D., Hobson, P., Lukauskas, S., Gemperline, D. C., Augspurger, T., Halchenko, Y., Cole, J. B., Warmenhoven, J., de Ruiter, J., Pye, C., Hoyer, S., Vanderplas, J., Villalba, S., Kunter, G., Quintero, E., Bachant, P., Martin, M., ... Qalieh, A. (2017). *Mwaskom/seaborn: V0.8.1 (september 2017)* (Version v0.8.1). Zenodo. https://doi.org/10.5281/zenodo.883859

Waskom, M. L. (2021). Seaborn: Statistical data visualization. *Journal of Open Source Software*, *6*(60), 3021. https://doi.org/10.21105/joss.03021