The phonological inventory emerges substance-freely from the phonetics and the morphology

Jeppe Mul, student no. 13626701

Supervised by dr. Paul Boersma

University of Amsterdam

BA Thesis Linguistics

23-6-2025 Anno Domini

Abstract

In this thesis, a deep restricted Boltzmann machine is trained to learn a toy language based on Moravian Czech for the purpose of gaining insight into the nature of the distribution of the phonological inventory. For the first time, monophthongs are modelled on a time axis with 17 timesteps. After learning, the distribution of the excitation pattern between different vowel inputs is compared. In line with previous work, a bidirectional substance-free network is shown to be able to acquire a phonological inventory. In the distribution of the excitation patterns after learning, the model displays features of vowel height, backness and length. This supports the hypothesis that the phonological inventory is substance-free. It took the model significantly longer to learn to discriminate in excitation pattern between long and short vowels than it took for the model to learn to discriminate between vowels on the basis of their formants. Once acquired, vowel length appears to have a lower degree of influence on the distribution of the excitation pattern than the other features as the algorithm learns more, whereas height and backness have a higher degree of influence on the results as the number of iterations increases. In combination with a theoretical argument that the kind of model used in this thesis is significantly cognitively unrealistic, this thesis casts doubts upon the generalizability to human cognition not just for the results in this thesis, but on any model which uses a similar method to capture the time axis.

Table of Contents

1. Introduction

Nativist phonologists have long struggled with a linking problem. If phonological features are linguistic universals "available to the child learning a language as an a priori, innate endowment" (Chomsky & Halle, 1968. p.4), there is no known learning algorithm for mapping these features onto acoustic hyperspace language-specifically (Boersma, 2012). Such positions therefore fail to explain the well-documented phenomenon that the same acoustic event may be mapped onto different phonetic categories and therefore mapped onto different phonemes crosslinguistically (Hamann, Boersma & Ćavar 2010).

This problem disappears if phonetic categories are not assumed to be innate, but instead learned from the acoustic events that serve as the input for the first language learner (Hamann 2007). From the interactions between the phonetics-phonology and phonology-morphology interfaces, features emerge on the phonological level that do not contain information, but are merely linked to the substantive information present at the phonetic and morphological levels (Boersma, Chládková & Benders, 2022).

One model of phonetics and phonology that is consistent with the learned, not innate position is BiPhon-NN, which distinguishes itself as the only known minimal and comprehensive model of phonetics and phonology that comes with a programme for its acquisition. Crucially, it shows explanatory power on the same level as other models (Boersma 2011). Whilst the original optimal-theoretic version of the framework, BiPhon-OT suffered from the same linking problem, as it assumes constraints are innate and only the ranking between them differs, a version that has been translated to neural networks has shown promising results without relying on innate constraints (Boersma, Benders & Seinhorst, 2020).

1.1 Neural networks in phonetics and phonology

Boersma, Benders & Seinhorst (2020) argue that the ultimate model of phonetics and phonology, and by extension the learning rule for this model must be a framework on the basis of neural networks, as they are the only type of framework that is both cognitively realistic and able to adequately explain the phonological and phonetic phenomena in naturalistic and experimental contexts. In that paper, a bidirectional learning algorithm was developed, which enabled a neural network to not just learn a toy language, but display featural behaviour from continuous input.

Although that algorithm will not be used in this paper, it does suit a pattern of similar research done in recent years, such as Boersma, Chládková & Benders (2022), as well as Bachelor theses such as van Beemdelust (2020) and Shchupak (2023). However, all of these papers suffer from a similar weakness, namely that the neural network only ever performs an analysis on the basis of at most two samples of acoustic data in the form of nodes that represent basilar membrane activations. This can be seen as relatively cognitively unrealistic, as the acoustic data does not arrive all at the same time for moments of analysis, but instead comes in over a timespan nearly continuously.

This is especially problematic when it comes to the distinction between long and short vowels. These vowel pairs are distinguished in some languages purely on the basis of duration, such as in the Moravian variety of Czech (Šimáčková, Podlipský & Chládková, 2012. p.229). Since duration is the primary cue for differentiation, a neural network in which the architecture reflects the fact that the acoustic data comes in the

form of a time series might serve as a better reflection of what a learner of Czech experiences.

A naive but computationally simple way to approach this problem is by scaling the number of input nodes. One or two samples across a vowel are not enough to capture its acoustic length, but by sampling a consistent amount of times no matter whether the vowel is long or not, the values in the later samples will be zero for short vowels, as they have already ended, whereas long vowels will continue to provide non-zero values of excitation in the later samples. A network may then be able to differentiate the vowels on the basis of those nodes which are sometimes but not always zero depending on the vowel's length.

This thesis therefore analyses the ability of a neural network with such an architecture to learn a toy language that is based on the Moravian variety of Czech, and if successfully learns to do so, analyses whether it can recreate the emergence of categories. In the event that these categories do emerge, an analysis is performed to test whether the short-long vowel pairs with the same quality show greater similarity between themselves as opposed to the similarity between different quality vowels with the same length. This could provide insight into a possible organization of the representations of vowels in the phonemic inventory.

2. The toy language

Šimáčková, Podlipský & Chládková (2012. p.229) state that the Moravian variety of Czech contains 5 qualities for its monophthongs, namely /i/, /ɛ/, /a/, /o/ and /u/. Each short monophthong has a counterpart with the exact same quality. Moravian Czech short vowels are roughly equally long, whereas long vowels are 1.7 times longer than their short counterparts. The exception to this is /iː/, which is only 1.3 times longer than /i/. A table of the vowels and their acoustic qualities is given below.

Table 1: an overview of the acoustic qualities of vowels in Moravian Czech

Quality	F1	F2	Duration
/i/	300	2500	1.3
/ɛ/	650	1800	1.7
/a/	800	1500	1.7
/o/	500	900	1.7
/u/	300	500	1.7

These properties make for an excellent dataset to generate data from for the purposes of this study, as the only cue for differentiating long and short vowels is the difference in duration, and there are no gaps in the inventory such that certain samples

can be classified in length on the basis of their quality alone, forcing the network to differentiate on length for each quality.

Moravian Czech, as all natural languages, is semantically and morphologically far too complex to accurately capture in a manner simple enough for a model like the one used in this paper. As such, even though the acoustic elements of the toy language are very closely related to Moravian Czech, the semantic properties of the language used in the model cannot be translated in a similarly close way.

An imperfect, but useful remedy is to create a toy language in terms of semantics, in which meaning is captured only by differences in vowel quality and differences in vowel length. An utterance is then considered to exist only of a single vowel, as every feature in the vowel now corresponds to a given atomic meaning. These meanings have been assigned to each feature arbitrarily. Note that /a/ and /aː/ happen to pattern with the back vowels, this decision was made entirely arbitrarily. For an overview of the meanings by utterance, see table 2 below.

Table 2: an overview of meanings by utterance

Utterance	Meaning
/i/	[EGG], [SG], [NOM]
/iː/	[EGG], [SG], [ACC]
/ɛ/	[CHEESE], [SG], [NOM]
/ɛː/	[CHEESE], [SG], [ACC]
/a/	[BUTTER], [PL], [NOM]
/aː/	[BUTTER], [PL], [ACC]
/o/	[CHEESE], [PL], [NOM]
/oː/	[CHEESE], [PL], [ACC]
/u/	[EGG], [PL], [NOM]
/uː/	[EGG], [PL], [ACC]

A human linguist would no doubt come to the conclusion that the binary features length and backness denote case and number respectively in this language. They should also conclude that the lexical meanings are denoted by vowel height, which could be split into multiple binary features such as [high] and [low]. A neural network cannot construct a phonological inventory like this. The hope is that in the similarity in excitations between different standard vowels as input, categories emerge.

3. The neural network

The network trained in this thesis is a deep restricted Boltzmann machine, which is a variant of Boltzmann machines. The distinction comes in the inability of a restricted

Boltzmann machine to have nodes on the same layer inhibit each other. This makes restricted Boltzmann machines significantly less complex computationally, but they have been shown previously to suffice. A “deep Boltzmann machine” refers to any such machine with multiple layers of non-input nodes.

The network contains 3 different types of nodes, namely auditory input nodes, semantic input nodes and deep nodes. The term input node is a little misleading, as they only serve as the input at the very start of a training iteration. This is then spread throughout the deeper layers of the network, populated by deep nodes, only to be echoed back to the input layer and back up again. The network is bidirectional in the sense that the weights that are used between different layers of the network are used to determine the activation in both directions. This is to keep the model in line with BiPhon, of which one of the core tenets is its bidirectionality. The details of the learning procedure can be found in chapter 3.4.

The network includes some amount of randomness in the dreaming step, see details in chapter 3.4.3. To preserve a fair testing environment, all testing and results have been done using the same random seed using Python’s in-built random library. This guarantees constant model behaviour during the iterations that different models share, e.g. the first 3,000 iterations for all models in this thesis, even though it cannot guarantee equally favourable variation past that point.

3.1 Auditory input nodes

Auditory input nodes have an inherent ERB (Equivalent Rectangular Bandwith) value associated with them, as they represent parts of the basilar membrane that vibrate when sound with that frequency enters the ear. The network does not have access to these locations during the learning procedure, they are only used to determine the extent to which a given acoustic input activates each auditory input node.

Auditory input nodes come in groups of 49, namely with ERB values ranging from 4 to 28 inclusive, spaced 0.5 ERB apart. These values are identical to Boersma, Chládková & Benders (2022). One such group of 49 nodes represents a sample taken from the sound at a point in time, and will from this point on be referred to as a “slab”. This procedure and naming convention is analogous to van Beemdelust (2020). For every vowel, 17 slabs are excitated, each representing a consecutive point in time. This number represents the biggest number of the smallest set of integers at a 1:1.3:1.7 ratio, the ratios between long and short vowels in Moravian Czech. Long vowels other than /iː/ excitate all 17 slabs with non-zero values, /iː/ excitates only excitate slabs 1-13 with non-zero values and all short vowels only excitate slabs 1-10 with non-zero values. In unreported testing, it appears as though the model is robust to adding slabs that always have nothing but zero excitations.

The activation of every auditory input node is determined by Gaussian bumps. These bumps have a maximum height of 1 and a standard deviation of 0.68 ERB. Crucially, the half-width of these bumps is great enough such that the auditory nodes that are spaced 0.5 ERB apart cannot undersample the bumps. The means of the bumps correlate with the F1 and F2 of the sampled utterance. The exact procedure for calculating these points is given in chapter 3.4. After these bumps have been generated, every node calculates the distance between itself and these means, using the information about its location on the basilar membrane given by its inherent associated ERB value, after which it sets its excitation to the sum of the bumps. This excitation is then multiplied by 5 and has 1 subtracted from it to provide the activation for an auditory node prior to the learning procedure. This is analogous to Boersma, Chládková & Benders (2022).

Note that although the formant frequencies are determined by random sampling, this sampling only happens once to determine the excitation at every timestep simultaneously. In practice this means that auditory input nodes with the same inherent ERB values across different groups will have the exact same excitation as each other, the difference between the groups is only ever whether the group is 0 altogether or not. Note also that although these nodes have bias terms, these terms are not used to determine the clamped activation prior to the learning procedure.

3.2 Semantic input nodes

As opposed to the 17 slabs of auditory input nodes, each utterance activates only 1 group of semantic input nodes. Every semantic input node stands for a single atomic meaning, leading to a total of 7 semantic input nodes. This stems from the fact that the toy language contains 3 lexical meanings, [BUTTER], [CHEESE] and [EGG], 2 numbers [SG] and [PL], as well as 2 cases [NOM] and [ACC]. Notably, the network does not have access to these labels, nor does it have access to the knowledge that the first 3 represent a lexical meaning and the last 4 represent a morphological meaning. It also does not know which of the nodes are necessarily never on together as they represent semantically incompatible meanings.

Semantic nodes are activated by assigning an activation of -2 to a node with a meaning that the utterance does not have, and 3.5 for a node with a meaning that the utterance does have. This is analogous to the semantic nodes that concerned utterances with multiple atomic meanings in Boersma, Chládková & Benders (2022). Similarly to auditory nodes, these nodes do have bias terms, but they are also not used to calculate the activation prior to the learning procedure.

3.3 Deep nodes

The final kind of node, deep nodes, is the type of nodes in the middle and top layers of the network. For this thesis, the middle layer consists of 50 nodes, and the top layer

consists of 20. These numbers are chosen arbitrarily on the basis of Boersma, Chládková & Benders (2022), although they do need to be large enough to allow for distributional differences to appear.

Deep nodes in the middle layer have weights between themselves and the top layer, as well as weights between themselves and each of the auditory input nodes and each of the semantic input nodes. Nodes in the top layer only have weights connecting them to each of the nodes in the middle layer. Deep nodes on both levels also have bias terms, which are always added to the sum of excitations.

3.4 The learning algorithm

Before the start of learning, every weight and every bias in the network is set to 0. Before every learning step, the activations of every node in the network is also set to 0. To start a learning step, a random sample is drawn, after which the clamped excitation is set for every input node, as described in paragraph 3.1. Samples will be randomly chosen to be one of the 10 utterances with equal probability.

Once the utterance has been determined, some variation is introduced to determine the exact formant frequencies for that sample. This is done by converting the corresponding formant frequencies in table 1 to an ERB value, then taking a sample from a Gaussian with that value as the mean and a standard deviation of 1 ERB. F1 and F2 vary independently in this way. The formula for conversion from Hertz to ERB is given in equation 1 below and has been taken directly from Praat (Boersma & Weenink 2025). The rest of the algorithm is taken from Boersma, Chládková & Benders (2012), including every other equation shown in this chapter.

1) $ERB(Hz) = 11.17 * \ln(\frac{Hz+312}{Hz+1460} + 43)$

The learning procedure starts with the initial settling. Next, the network goes through a single phase of Hebbian learning. The network then repeats a dreaming step, to finish off with a round of Hebbian unlearning. The process of drawing a sample and learning from it is repeated some predetermined number of times, ranging from 1.000 to 30.000 iterations in this thesis.

3.4.1 Initial settling

Throughout the process of initial settling, the clamped excitation is spread throughout the network. Throughout this thesis, y_l stands for the excitation of deep node l on the middle layer, K stands for the total number of input nodes, adding both semantic input nodes and auditory input nodes, x_k is the excitation of input node k, where k can be either an auditory or a semantic input node, where u_{kl} stands for the weight of the connection between input node k and middle layer node l. Conversely, M stands for the

number of deep nodes on the top level, z_m stands for the excitation of node z on the top layer, and v_{lm} stands for the weight of the connection between middle layer deep node l and top layer node m. The function σ() stands for the logistic function, given in equation 2.

2) $\sigma(x) := \frac{1}{1+e^{-x}}$

The first step in initial settling is to spread the initial excitation to the middle nodes, shown in equation 3. This is done by adding the sums of the products of the activations on the other two layers with their respective weights, then adding a bias term. This sum of excitation is then put through the logistic function. The purpose of the activation function is to keep the activation of nodes in deep layers between 0 and 1, which prevents the nodes on the top layers from being dominated by just a few nodes on the middle layer that happen to be assigned a high activation early during training.

3) $y_l \leftarrow \sigma(b_l + \sum_{k=1}^{K} x_k u_{kl} + \sum_{m=1}^{M} z_m v_{lm})$

After excitation has been spread to the middle nodes, it is spread to the top nodes in similar fashion, given in equation 4. In this thesis, z_m stands for the excitation of deep node m on the top layer, c_m stands for the bias term of node m, L stands for the number of deep nodes on the middle layer, y_l stands for the excitation of node l on the middle layer and $y_l v_{lm}$ stands for the weight of the connection between nodes l and m. Equations 3 and 4 combine to create the initial settling step, which is repeated 10 times before moving onto the Hebbian learning step.

4) $z_m \leftarrow \sigma(c_m + \sum_{l=1}^{L} y_l v_{lm})$

3.4.2 Hebbian learning

This step can be understood as implementing the principle of neurons that fire together wiring together, strengthening the connections such that the network learns to associate certain regions of acoustic input and certain semantic nodes with some deep nodes. In this step, the biases for every node and the weights between the layers are increased using a learning rate η, in this thesis 0.001, modelled after Boersma, Chládková & Benders (2022). In the case of the biases, the learning rate is multiplied by the excitation of the node for which the bias term is to be increased. For the weights, the learning rate is instead multiplied by the product of the excitation of the nodes that the weight connects. The equations for the biases are shown in equations 5-6, the equations for the weights in equations 7-9. Equations 5 and 8 cover both auditory input nodes and semantic input nodes.

5) $a_k \leftarrow a_k + \eta x_k$
6) $b_l \leftarrow b_l + \eta y_l$
7) $c_m \leftarrow c_m + \eta z_m$
8) $u_{kl} \leftarrow u_{kl} + \eta x_k y_l$

9) $v_{lm} \leftarrow v_{lm} + \eta y_l z_m$

3.4.3 Dreaming

After 1 such step of Hebbian learning, the model performs dreaming steps, which adds random variation into the model by way of Bernoulli distribution. The Bernoulli distribution is a function where if z ~ B(p), where B() stands for the Bernoulli distribution and p is a probability from 0 to 1, z will be 1 with a chance of p and 0 with a chance of 1-p. Firstly, the input layer becomes unclamped, meaning that the model re-calculates the excitation at the input layer, using equation 10 instead of the procedure described in chapters 3.1 and 3.2. This is done for both the auditory and semantic input nodes. Then, using equations 11 and 12, new activities are computed at the top and middle levels.

10) $x_k \leftarrow a_k + \sum_{l=1}^{L} u_{kl} y_l$
11) $z_m \sim B(\sigma(c_m + \sum_{l=1}^{L} y_l v_{lm}))$
12) $y_l \sim B(\sigma(b_l + \sum_{k=1}^{K} x_k u_{kl} + \sum_{m=1}^{M} z_m v_{lm}))$

Adding random variation in some form ensures that the model cannot get stuck in a very similar distribution as the one it randomly forms near the beginning of training, and instead samples a large part of the distribution space throughout the training. These 3 steps are repeated 10 times before moving on to the final step of the procedure, the Antihebbian learning step.

3.4.4 Antihebbian learning

In this phase, the exact same procedure is done as in the Hebbian learning, except instead of adding the excitation to the biases and adding the product of the excitations to the weight, these numbers are subtracted from the biases and weights instead. This can be seen in equations 13-17. This is necessary to stop all weights and biases from exploding, which would result in convergent cosine similarities for all vowels as any input would fully excitate every node in the network given enough iterations have passed for this to occur. These steps are each performed once, similarly to the Hebbian learning step. This ensures that on average the amount of knowledge gained by the model in a learning step is similar to the amount of knowledge lost, preventing the explosion.

13) $a_k \leftarrow a_k - \eta x_k$
14) $b_l \leftarrow b_l - \eta y_l$
15) $c_m \leftarrow c_m - \eta z_m$
16) $u_{kl} \leftarrow u_{kl} - \eta x_k y_l$
17) $v_{lm} \leftarrow v_{lm} - \eta y_l z_m$

4. Results

After the learning procedure, the cosine similarity is calculated between the excitation of the output nodes for each of the standard utterances. A standard utterance is an utterance where the formants are exactly the average formants for each of the 10 possible utterances, without any random variation. The semantic input is unchanged from the training samples for that utterance. The cosine similarity is then given by equation 18, where subscript (a) and (b) represent different standard utterances.

$$18)\ cos(\theta) = \frac{\sum_{l=1}^{L} y_{l(a)} y_{l(b)}}{\sqrt{\sum_{l=1}^{L} y_{l(a)}^2} \cdot \sqrt{\sum_{l=1}^{L} y_{l(b)}^2}}$$

In practice, the cosine similarity between two lists converges to 1 as the lists become more similar, and converges to 0 as the lists become more dissimilar. This is because the input for these lists is the activation of a deep layer, which has had a sigmoid applied to it, making all the elements of both vectors non-negative. When there is no correlation between the lists, a value around 0.5 is expected, as there will be some random covariation.

The behaviour of the model changes with the number of iterations that the learning algorithm is ran. During the first iterations, the model is still very close to its initial state, and reporting on the outcomes would be moot, as no excitation pattern has been able to form yet, giving cosine similarities between all vowels of nearly 1. For this reason, this thesis only reports on the cosine similarities for the patterns after 3,000, 10,000 and 30,000 iterations. These numbers are chosen as models after 3,000 and 10,000 learning steps represented learning and mature models respectively in Boersma, Chládková & Benders (2022), whereas the model in van Beemdelust (2020) was robust up to 30,000 inputs. This variation in the amount of training the model received was the only differing variable between the three tests.

All results in this chapter have been obtained using the same Python script, which can be found in appendix A. It is callable from the command line and prints the results directly back to the command line, but the number of iterations cannot be specified from the command line. The script depends on SciPy, an open-source library that implements a number of mathematical and scientific functions. (Virtanen et al., 2020).

4.1 Results after 3,000 iterations

After 3,000 iterations, the model is able to produce results that seem to indicate the model has already been able to learn to differentiate between utterances of different qualities. It seems as though it is not yet fully able to differentiate between vowels of the same quality and different lengths however, as can be seen in table 3. The tables in this chapter will all be flipped along the diagonal axis, as the cosine similarity between two lists does not depend on which of the lists is considered the primary list.

Table 3: cosine similarities between the excitation patterns after 3,000 iterations

	/aː/	/a/	/ɛː/	/ɛ/	/iː/	/i/	/oː/	/o/	/uː/	/u/
/aː/	1	0.985607365	0.576138794	0.55423457	0.631286095	0.603526669	0.844014088	0.818298521	0.893521547	0.890514662
/a/	0.985607365	1	0.567419293	0.571747488	0.607928231	0.606699575	0.827754034	0.828643597	0.860263211	0.884244347
/ɛː/	0.576138794	0.567419293	1	0.987417374	0.801052785	0.801848801	0.804556865	0.800914845	0.595545788	0.593324238
/ɛ/	0.55423457	0.571747488	0.987417374	1	0.773207553	0.798592149	0.783513037	0.805118048	0.554917185	0.578584239
/iː/	0.631286095	0.607928231	0.801052785	0.773207553	1	0.986425243	0.612557914	0.580447751	0.78589359	0.778914542
/i/	0.603526669	0.606699575	0.801848801	0.798592149	0.986425243	1	0.594610302	0.589065839	0.751350775	0.770883577
/oː/	0.844014088	0.827754034	0.804556865	0.783513037	0.612557914	0.594610302	1	0.986033846	0.804969847	0.801909154
/o/	0.818298521	0.828643597	0.800914845	0.805118048	0.580447751	0.589065839	0.986033846	1	0.760643365	0.784284795
/uː/	0.893521547	0.860263211	0.595545788	0.554917185	0.78589359	0.751350775	0.804969847	0.760643365	1	0.985161362
/u/	0.890514662	0.884244347	0.593324238	0.578584239	0.778914542	0.770883577	0.801909154	0.784284795	0.985161362	1

As can be seen in table 3, the cosine similarity between vowels of the same quality ranges between 98.5% and 98.7%. These numbers indicate near-complete similarity in excitation pattern, which suggests that the model has not yet learned to display a phonological feature on the basis of length, as it implies that the model cannot really distinguish long vowels from short ones. There does appear to be some indication that the model is starting to figure out that length matters, as long utterances typically display slightly greater similarity to other long utterances as opposed to their short counterparts. For example, /aː/ displays a similarity of ~84.4% to /oː/, but a slightly smaller ~81.8% to /o/.

This stands in stark contrast to height and backness, which already emerge quite strongly after 3,000 samples. As can be seen in table 1, /ɛ/ has relatively similar formant frequencies to /a/. However, as seen in table 2, they don't actually share a vowel height or a backness feature, as /a/ patterns with the back vowels in this toy language. If the model was going off mere acoustics, one would predict /ɛ/ and /a/ to have a relatively high cosine similarity, but the opposite pattern emerges already. Moreover, the similarity between /ɛ/ and /i/ is very similar to the similarity between /ɛ/ and /o/, namely 79.9% to 80.5% respectively, which can be explained by the fact they each share 1 non-length feature, namely a lack of backness with /i/ and a vowel height that might best be explained in featural terms as both non-low and non-high with /o/.

4.2 Results after 10,000 iterations

After 10,000 samples, the model has learned to discriminate long vowels from their short counterparts, but only translates to some extent into apparent phonologization. This can be seen in table 4.

Table 4: cosine similarities between the excitation patterns after 10,000 iterations

	/aː/	/a/	/ɛː/	/ɛ/	/iː/	/i/	/oː/	/o/	/uː/	/u/
/aː/	1	0.836937425	0.600308348	0.466637363	0.676947201	0.521881088	0.809015151	0.66096994	0.886206953	0.722917867
/a/	0.836937425	1	0.44686154	0.598562055	0.516168801	0.677949242	0.648581533	0.803021446	0.732467473	0.893564681
/ɛː/	0.600308348	0.44686154	1	0.867558692	0.827409556	0.685757044	0.803581819	0.663034933	0.641209791	0.471044967
/ɛ/	0.466637363	0.598562055	0.867558692	1	0.68473256	0.829692615	0.67661332	0.809849665	0.504503542	0.635701145
/iː/	0.676947201	0.516168801	0.827409556	0.68473256	1	0.834283846	0.617124045	0.457373154	0.794933304	0.608922537
/i/	0.521881088	0.677949242	0.685757044	0.829692615	0.834283846	1	0.477218858	0.627576856	0.64935485	0.796532913
/oː/	0.809015151	0.648581533	0.803581819	0.67661332	0.617124045	0.477218858	1	0.849472832	0.797759436	0.650612806
/o/	0.66096994	0.803021446	0.663034933	0.809849665	0.457373154	0.627576856	0.849472832	1	0.642134737	0.810912078
/uː/	0.886206953	0.732467473	0.641209791	0.504503542	0.794933304	0.64935485	0.797759436	0.642134737	1	0.819083155
/u/	0.722917867	0.893564681	0.471044967	0.635701145	0.608922537	0.796532913	0.650612806	0.810912078	0.819083155	1

From table 4, we can observe that the difference between a long and short vowel of the same quality is no longer nearly 1, but instead ranges from ~81.9% for /u/ and /uː/ to 86.7% for /ɛː/ and /ɛ/. This appears to be roughly similar to the similarities between utterances that share two other features, such as the ~81% similarity between /oː/ and /uː/, which share backness and length.

Whereas in the results with 3,000 iterations /a/ was roughly equally similar to both /ɛː/ and /ɛ/, there is now a ~15% gap between the two with the also short /ɛ/ being the more similar one. The similarity is however still not equally large as the similarity between /a/ and /uː/, with which it also shares 1 feature, namely backness. This is in spite of the previously noted acoustic similarities between /a/ and /ɛ/. This pattern can be seen in every utterance, there is always a significant difference between the similarities of any utterance with respect to the short and long utterances of another quality.

Contrary to length, backness and height appear to be roughly similarly present in these results when compared to the results at 3,000 iterations. The lowest cosine similarities are once again the utterances with the smallest amount of common features of height and backness between them, regardless of acoustic similarities, and vice versa the strongest similarities those which share multiple features.

4.3 Results after 30,000 iterations

Similarly to van Beemdelust (2020), and in opposition to Boersma (2019), the model was able to keep producing meaningful results after 30,000 iterations. In contrast to van Beemdelust (2020) however, the model in this thesis continues to change this deep into the learning process. Results for this model are shown in table 5.

Table 5: cosine similarities between the excitation patterns after 30,000 iterations

	/aː/	/a/	/ɛː/	/ɛ/	/iː/	/i/	/oː/	/o/	/uː/	/u/
/aː/	1	0.783320326	0.607767932	0.445736967	0.744351571	0.556091528	0.766177438	0.578239684	0.876929163	0.680332534
/a/	0.783320326	1	0.425440146	0.685389484	0.522610069	0.786347504	0.585299994	0.82282215	0.673430742	0.910969909
/ɛː/	0.607767932	0.425440146	1	0.781002571	0.815149468	0.620886556	0.847240009	0.612862407	0.688283339	0.471730711
/ɛ/	0.445736967	0.685389484	0.781002571	1	0.589293176	0.840604107	0.660405003	0.86179797	0.492951155	0.720313043
/iː/	0.744351571	0.522610069	0.815149468	0.589293176	1	0.755479132	0.654340119	0.414179763	0.847652248	0.598881732
/i/	0.556091528	0.786347504	0.620886556	0.840604107	0.755479132	1	0.495710865	0.709481189	0.661227325	0.870697118
/oː/	0.766177438	0.585299994	0.847240009	0.660405003	0.654340119	0.495710865	1	0.776475098	0.775947748	0.604847783
/o/	0.578239684	0.82282215	0.612862407	0.86179797	0.414179763	0.709481189	0.776475098	1	0.560483006	0.828337464
/uː/	0.876929163	0.673430742	0.688283339	0.492951155	0.847652248	0.661227325	0.775947748	0.560483006	1	0.760587759
/u/	0.680332534	0.910969909	0.471730711	0.720313043	0.598881732	0.870697118	0.604847783	0.828337464	0.760587759	1

Contrary to the results after 10,000 timesteps, for every vowel pair with the same formant quality but different lengths, the similarity between themselves is smaller than that with other vowels that share the same amount of features between them. Whereas after 10.000 timesteps, the similarities between these vowels ranged between ~81.9% and ~86.7%, they now range between only ~75.4% and ~78.3%. This is in opposition to other utterance pairs that share two qualities, such as /i/ and /u/, which actually increased in similarity from ~79.7% to ~87%.

Although long and short vowels with same quality are still more similar than comparisons to vowels with a lesser amount of shared features, which suggests that length is a feature in this phonological inventory, the weakened similarity as opposed to the results after 10.000 iterations indicates that it does not have the same level of salience as backness and vowel height after 30.000 iterations, even though backness is just one binary feature just like length.

In this matrix, we can clearly observe the other features emerge in a more similar pattern to the previous behaviour. As before, despite the acoustic similarities between /a/ and /ɛ/ in terms of formant frequencies being greater than between /a/ and /o/, the fact they share the plural number in this language leads to a greater similarity between the excitation patterns of /a/ and /o/. Just like after 3,000 & 10,000 iterations, for every utterance the smallest similarity is with the utterance they share the least amount of features with.

These results serve as an indication that although length can be trained to play a role in the phonological inventory, in these results there exists a significant difference between it, a feature primarily cued by duration, and features primarily cued by formant frequencies such as backness and height. It took longer for length as a feature to appear, it behaved differently at 10.000 timesteps and weakened considerably after 30.000 iterations.

5. Discussion

The present research is not without its flaws, the most notable of which is the inaccurate representation of the time dimension in the model. Although each of the 17 samples represents a particular point in time, the model has no access to this information. As such, for every location on the basilar membrane, there are 17 connections to each deep node on the middle layer, one for each timestep.

This is clearly cognitively unrealistic, as it is essentially claiming that neurons at the basilar membrane are connected to neurons in the brain through 17 different direct pathways, each with their own different connection strengths. Neurons are only ever connected to each other by 1 connection and are not able to distinguish between which connection they should be using based on an arbitrary timestamp they cannot access.

Not only is it cognitively unrealistic, it is also not improbable that it explains why in this thesis, vowel length did not manifest in the same manner as the other features. Whilst the model could use 17 groupings to infer the other features, length could only be inferred by at most 7 and in the case of /i/ only 3 groupings of auditory nodes. It is then certainly possible that over time the sheer number of nodes that determine acoustic similarity dominate the few that can also determine length differences to the extent that the model is not able to phonologize length to the same degree as quickly or as robustly as backness or height.

One possible next step is therefore to devise a network that can adequately address this weakness. Architectures that are able to model a basilar membrane with multiple timestamped inputs while only restricting themselves to 1 weight between each auditory input node and a deep node include architectures such as long short-term memory models. The reason these models were not chosen is that these architectures were originally intended to work under supervised learning conditions (Hochreiter & Schmidhuber, 1997). They are also not designed to work bidirectionally. Although learning algorithms for unsupervised learning in such models have been proposed, such as the BINGO algorithm (Schraudolph & Sejnowski, 1992) used in Klapper-Rybicka, Schraudolph & Schmidhuber (2001), it proved infeasible to incorporate them into this thesis.

Another significant weakness is the extent to which the toy language maps onto natural language. For this thesis, the acoustic mapping to natural language is very strong, but the mapping of meaning to any natural language is completely absent. Moreover, the relationship between feature and meaning is completely transparent in the toy language. This calls into question the extent to which the findings can be extrapolated to human natural language learning, in which the link between meaning and features is far less clear.

Similarly, although the toy language and the model parameters were constructed carefully to the best ability following previous research, the model is not tested in this thesis with different numbers of deep nodes, or different configurations of the toy language. Consequently, the generalizability of the results is not just low with respect to human phonological language learning, but possibly limited with respect to other neural networks too.

6. Conclusion

This thesis set out to determine whether a restricted deep Boltzmann machine is able to learn a toy language even when the number of auditory input nodes is 17 times larger than previously known to be possible, implicitly adding an axis of time to the model. It has clearly achieved this goal, as the results show the ability of the model to

discriminate not just between the vowels of different acoustic qualities, but also vowels with the same quality but different lengths.

The thesis then asked the question whether each of the features present in the toy language, namely vowel height, backness and length emerged from an analysis of the distribution of the excitation patterns produced by each standard utterance. Results varied depending on the amount of iterations the model ran for. It took significantly longer for a feature on the basis of length to appear. The more learning steps were performed, even when the model was finally able to discriminate long vowels from short vowels, the greater the influence of vowel height and backness on the model as compared to length. The finding that all features could be elicited was therefore found to be somewhat robust, but the relative strength between the features was not.

The thesis also originally set out to generalize these findings to a human phoneme inventory. In this, it has not been able to achieve its goal. In some part, this can be attributed to the lack of robustness in the findings regarding the different features, but it also stems from the theoretical limitation of the model. Whilst the methodology is in line with previous research, it is shown theoretically that this type of neural network has a significant flaw in its capacity to represent time in a manner that resembles human cognition. In combination with the lack of testing for robustness, the findings in this thesis should therefore not be construed as to argue for a particular view on the structure of the phonological inventory in a human brain, nor on the feasibility of a human language learner to acquire a phonological length feature substance-freely.

7. Bibliography

van Beemdelust, A. (2020) The addition of time to a restricted deep Boltzmann machine using a holistic model, and the machine's ability to distinguish between different sequences of sounds. [BA Thesis, University of Amsterdam]

Boersma, P. (2011). A programme for bidirectional phonology and phonetics and their acquisition and evolution. In *Bidirectional Optimality Theory* (pp. 33–72). essay, John Benjamins.

Boersma, P. (2012). Modeling phonological category learning. In *The Oxford handbook of laboratory phonology* (pp. 207–218). Oxford University Press.

Boersma, P., Benders, T., & Seinhorst, K. (2020). Neural network models for phonology and Phonetics. *Journal of Language Modelling*, *8*(1).

Boersma, P., Chládková, K., & Benders, T. (2022). Phonological features emerge substance-freely from the phonetics and the morphology. *Canadian Journal of Linguistics/Revue Canadienne de Linguistique*, *67*(4), 611–669.

Chomsky, N., & Halle, M. (1968). *The sound pattern of English*. Harper and Row.

Hamann, S. (2007). *Constructing features: the example of Dutch and German labiodentals*. Phonetic Sciences Amsterdam. https://www.fon.hum.uva.nl/silke/handouts/features2007.pdf

Hamann, S., Boersma, P., Cavar, M. (2010). Language-specific differences in the weighting of perceptual cues for labiodentals. In *Proceedings of the 6th International Symposium on the Acquisition of Second Language Speech, New Sounds* (pp. 167–172).

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Klapper-Rybicka, M., Schraudolph, N., & Schmidhuber, J. (2001). Unsupervised learning in LSTM recurrent neural networks. *Lecture Notes in Computer Science*, 684–691.

Schraudolph, N., & Sejnowski, T. (1992). Unsupervised discrimination of clustered data via optimization of binary information gain. *Advances in Neural Information Processing Systems, 5*.

Shchupak, A. (2023) Akanje in a Deep Boltzmann Machine. [BA Thesis, University of Amsterdam]

Šimáčková, Š., Podlipský, V. J., & Chládková, K. (2012). Czech spoken in Bohemia and Moravia. *Journal of the International Phonetic Association*, *42*(2), 225–232. https://doi.org/10.1017/s0025100312000102

Virtanen, P., Gommers, R., Oliphant, T., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., Walt, S., Brett, M., Wilson, J., Millman, K., Mayorov, N., Nelson, A., Jones, E., Kern, R., Larson, E., Carey, C., Polat, ., Feng, Y., Moore, E., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E., Harris, C., Archibald, A., Ribeiro, A., Pedregosa, F., van Mulbregt, P., & SciPy 1.0 Contributors (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods, 17, 261–272*.

8. Appendix A: Script

```
import math
from scipy.stats import norm
import random

# ensure the same random seed for cross-model testing
random.seed(0)

def hz_to_erb (formant:int):
  """
  This definition is the same one as used in Praat.
  """

  return 11.17 * math.log(((formant+312)/(formant+14680))) + 43

def gaussian_activation(input_ERB: float, membrane_location: float):
  """
  This implements what is meant by a Gaussian bump, namely that
  the activation of an input node is the height of a Gaussian with peak 1
  and SD 0.68 (after Boersma, Chládková & Benders 2022) at the value that the
  node takes on the basilar membrane. The first part ensures that the peak has height 1.
  """
  max_height = 1
  standard_dev = 0.68

  return  max_height * math.sqrt(2*math.pi) * standard_dev * norm.pdf(membrane_location, input_ERB, standard_dev)

def logistic_function(x:float):
  """
  This is the logistic function.
  """
  if x <= 0:
    return 1- (1 / (1 + math.exp(x)))

  return 1 / (1 + math.exp(-x))
def cosine_similarity(a:list, b:list):
  """
  This function computes the cosine similarity between 2 vectors.
```

```
    The cosine similarity is given by summing the products of the i'th element of A and B,
    then dividing this by the product of the square roots of the summations of the squares of each element of A and B respectively.

    This function assumes the lists are equally long.
    """
    # calculate numerator
    numerator = 0
    for i in range(len(a)):
        numerator += a[i] * b[i]

    # calculate root of sum of squares of A
    a_sum = 0
    for a_number in a:
        a_sum += a_number**2

    a_sum = math.sqrt(a_sum)

    # calculate root of sum of squares of B
    b_sum = 0
    for b_number in b:
        b_sum += b_number**2

    b_sum = math.sqrt(b_sum)

    # calculate denominator:
    denominator = a_sum * b_sum

    # calculate cosine similarity
    return numerator / denominator

class semanticNode():
    def __init__(self):
        self.bias = 0
        self.excitation = 0

class inputNode():
    def __init__ (self, erb:int):
        self.erb = erb
```

```
        self.bias = 0
        self.excitation = 0

    def set_clamped_excitation(self, formant1:int, formant2:int):

        self.excitation = gaussian_activation(self.erb, formant1) + gaussian_activation(self.erb, formant2)
        self.excitation = self.excitation*5 -1

class middleNode:
    def __init__(self):
        self.bias = 0
        self.excitation = 0
        self.weights = [[0 for x in range(49)] for x in range(17)]
        self.semantic_weights = [0 for x in range(7)]

class topNode:
    def __init__(self):
        self.bias = 0
        self.excitation = 0
        self.weights = [0 for x in range(50)]

class model:
    def __init__(self):
        self.input_layer = [[inputNode(x/2) for x in range(8,57)] for x in range(17)]

        self.semantic_layer = [semanticNode() for x in range(7)]

        self.middle_layer = [middleNode() for x in range(50)]

        self.top_layer = [topNode() for x in range(20)]

    def run(self, vowels:int):
        """
        This method is the main way to call on this class.
        It implements the learning algorithm described in
        Boersma, Chládková & Benders (2022)
        """

        for datum in range(vowels):
```

```
        # this just keeps track of where we are so I can see if my laptop got stuck
        print(datum)

        self.determine_vowel()

        for i in range(10):
            self.initial_settling()

        self.hebbian_learning()

        for i in range(10):
            self.dreaming()

        self.antihebbian_learning()

        # this is a debugging tool to keep track of what the output is
        #if datum % 100 == 1:
            #self.obtain_results()



def obtain_results(self):
    """
    This method first creates lists of the excitations of the top layer of the model for each standard utterance.
    It then calculates the cosine similarity between each of them and prints it.
    """
    formant_list = [["A", 800,1500],["E",650,1800],["I",300,2500],["O",500,900],["U",300,500]]

    output_dict= {}

    for vowel in formant_list:
        output_list = []
        self.set_input_manually(vowel[0], vowel[1], vowel[2], True)

        for i in range(10):
            self.initial_settling()

        for output_node in self.middle_layer:
            output_list.append(output_node.excitation)
```

```
            output_dict[vowel[0]] = output_list

            output_list = []
            self.set_input_manually(vowel[0], vowel[1], vowel[2], False)

            for i in range(10):
                self.initial_settling()

            for output_node in self.middle_layer:
                output_list.append(output_node.excitation)

            output_dict[vowel[0].lower()] = output_list



        for list1 in output_dict.values():
            for list2 in output_dict.values():
                print(cosine_similarity(list1, list2))
            print("")

        # this is a debugging tool
        #for list in output_dict.values():
            #print(list)

    def determine_vowel(self):
        """
        This method determines the vowel to use as input.
        I'm using averages from Šimáčková, Podlipský & Chládková (2012)
        """
        formant_list = [["A", 800,1500],["E",650,1800],["I",300,2500],["O",500,900],["U",300,500]]
        vowel = random.choice(formant_list)

        self.quality = vowel[0]
        f1 = vowel[1]
        f2 = vowel[2]

        self.f1_erb = hz_to_erb(f1)
        self.f2_erb = hz_to_erb(f2)
```

```
# this function is implemented in the random module as random(mean, std.dev)
self.f1_erb = random.gauss(self.f1_erb, 1)
self.f2_erb = random.gauss(self.f2_erb, 1)

self.long_vowel = False

# >= because 0 is included in random.random, but 1 is not
if random.random() >= 0.5: self.long_vowel = True

# this just happens to be a fact about Czech, /iː/ is only 1.3 times as long
if self.long_vowel == True:
  if not self.quality == "I":
    length = 17
  else:
    length = 13
else:
  length = 10

for layer in self.input_layer[0:length]:
  for node in layer:
    node.set_clamped_excitation(self.f1_erb, self.f2_erb)

for layer in self.input_layer[length:]:
  for node in layer:
    node.excitation = 0


# set the semantic layer
if not self.long_vowel:
  self.quality = self.quality.lower()

# first 2 stand for number, next 2 stand for case, last 3 stand for semantic meaning
semantic_dict = {
  "A":[-2,3.5,-2,3.5,-2,-2,3.5],
  "a":[-2,3.5,3.5,-2,-2,-2,3.5],
  "E":[3.5,-2,-2,3.5,3.5,-2,-2],
  "e":[3.5,-2,3.5,-2,3.5,-2,-2],
  "I":[3.5,-2,-2,3.5,-2,3.5,-2],
  "i":[3.5,-2,3.5,-2,-2,3.5,-2],
```

```
    "O":[-2,3.5,-2,3.5,3.5,-2,-2],
    "o":[-2,3.5,3.5,-2,3.5,-2,-2],
    "U":[-2,3.5,-2,3.5,-2,3.5,-2],
    "u":[-2,3.5,3.5,-2,-2,3.5,-2]}

  for semantic_node in range(len(self.semantic_layer)):
    self.semantic_layer[semantic_node].excitation = semantic_dict[self.quality][semantic_node]



def set_input_manually(self, quality:str, f1:int, f2:int, input_length:bool):
  """
  This method sets a standard utterance for testing purposes.

  Quality should be one of "A", "E", "I", "O" and "U".
  F1 and F2 are the formants. input_length = True means a long vowel, False means a short vowel
  """

  self.quality = quality
  self.f1_erb = hz_to_erb(f1)
  self.f2_erb = hz_to_erb(f2)
  self.long_vowel = input_length

  # this just happens to be a fact about Czech, /iː/ is only 1.3 times as long
  if self.long_vowel == True:
    if not self.quality == "I":
      length = 17
    else:
      length = 13
  else:
    length = 10

  for layer in self.input_layer[0:length]:
    for node in layer:
      node.set_clamped_excitation(self.f1_erb, self.f2_erb)

  for layer in self.input_layer[:length]:
    for node in layer:
      node.excitation = 0
```

```
    # set the semantic layer
    if not self.long_vowel:
        self.quality = self.quality.lower()

    # first 2 stand for number, next 2 stand for case, last 3 stand for semantic meaning
    semantic_dict = {
        "A":[-2,3.5,-2,3.5,-2,-2,3.5],
        "a":[-2,3.5,3.5,-2,-2,-2,3.5],
        "E":[3.5,-2,-2,3.5,3.5,-2,-2],
        "e":[3.5,-2,3.5,-2,3.5,-2,-2],
        "I":[3.5,-2,-2,3.5,-2,3.5,-2],
        "i":[3.5,-2,3.5,-2,-2,3.5,-2],
        "O":[-2,3.5,-2,3.5,3.5,-2,-2],
        "o":[-2,3.5,3.5,-2,3.5,-2,-2],
        "U":[-2,3.5,-2,3.5,-2,3.5,-2],
        "u":[-2,3.5,3.5,-2,-2,3.5,-2]}

    for semantic_node in range(len(self.semantic_layer)):
        self.semantic_layer[semantic_node].excitation = semantic_dict[self.quality][semantic_node]

def initial_settling(self, test:bool = False):
    """
    This method implements the initial settling phase
    from Boersma, Chládková & Benders (2022)

    First, activity is spread from both outer layers to the middle layer.
    Then, activity spreads up from the middle layer to the top layer.
    Activity spreading is done by taking the sum of the incoming weights * activations
    and adding a bias. This sum is then put through the logistic function to determine
    the new activity level of the deep layer.

    the test variable is only for debugging
    """
    for node in self.middle_layer:
        node.excitation = 0

    for node in self.top_layer:
        node.excitation = 0
```

```
    # update the middle layer first, spreading from both directions
    for node in range(len(self.middle_layer)):
      excitation_sum = self.middle_layer[node].bias

      # add from below
      for layer in range(len(self.middle_layer[node].weights)):
        for weight in range(len(self.middle_layer[node].weights[layer])):
          excitation_sum += self.middle_layer[node].weights[layer][weight] * self.input_layer[layer][weight].excitation

      for semantic_node in range(len(self.middle_layer[node].semantic_weights)):
        excitation_sum += self.middle_layer[node].semantic_weights[semantic_node] *
self.semantic_layer[semantic_node].excitation

      # add from above
      for upper_node in self.top_layer:
        excitation_sum += upper_node.excitation * upper_node.weights[node]

      # apply logistic function
      self.middle_layer[node].excitation = logistic_function(excitation_sum)

      if math.isinf(excitation_sum) or math.isnan(excitation_sum):

        for layer in range(len(self.middle_layer[node].weights)):
          for weight in range(len(self.middle_layer[node].weights[layer])):
            print(layer, weight, self.middle_layer[node].weights[layer][weight],  self.input_layer[layer][weight].excitation)

        for semantic_node in range(len(self.middle_layer[node].semantic_weights)):
          print(semantic_node, self.middle_layer[node].semantic_weights[semantic_node],
self.semantic_layer[semantic_node].excitation)

        # add from above
        for upper_node in self.top_layer:
          print(upper_node, upper_node.excitation, upper_node.weights[node])
        raise Exception(f"It went wrong at the middle layer {node}")


    # update the top layer ...
    for upper_node in self.top_layer:
      excitation_sum = upper_node.bias
```

```
        # by spreading from the middle
        for node in range(len(self.middle_layer)):
            excitation_sum += self.middle_layer[node].excitation * upper_node.weights[node]

        # apply logistic function
        upper_node.excitation = logistic_function(excitation_sum)

        if math.isinf(excitation_sum) or math.isnan(excitation_sum):
            raise Exception("It went wrong at the upper layer")

    def hebbian_learning(self, test:bool = False):
        """
        Update each weight and each bias by adding the relevant excitation * eta,
        which is defined as 0.001 following Boersma, Chládková & Benders (2022

        the test variable is only for debugging
        """
        eta = 0.001

        # change the bias at the input layer
        for input_layer in self.input_layer:
            for input_node in input_layer:
                input_node.bias += eta * input_node.excitation

        for n in self.semantic_layer:
            n.bias += eta * n.excitation

        # change the bias at the middle layer
        for node in self.middle_layer:
            node.bias += eta * node.excitation

            # change the weights between input and middle layer
            for layer in range(len(node.weights)):
                for weight in range(len(node.weights[layer])):
                    node.weights[layer][weight] += eta * node.excitation * self.input_layer[layer][weight].excitation

            for semantic_weight in range(len(node.semantic_weights)):
                node.semantic_weights[semantic_weight] += eta * node.excitation * self.semantic_layer[semantic_weight].excitation
```

```
    for upper_node in self.top_layer:
      # change the bias at the top layer
      upper_node.bias += eta * upper_node.excitation

      # change the weights between middle and top layer
      for weight in range(len(upper_node.weights)):
        upper_node.weights[weight] += eta * upper_node.excitation * self.middle_layer[weight].excitation

  def antihebbian_learning(self):
    """
    The exact same procedure as described above, except it's minus eta instead of plus.
    """
    eta = 0.001

    # change the bias at the input layer
    for input_layer in self.input_layer:
      for input_node in input_layer:
        input_node.bias -= eta * input_node.excitation

    for n in self.semantic_layer:
      n.bias -= eta * n.excitation

    for node in self.middle_layer:
      # change the bias at the middle layer
      node.bias -= eta * node.excitation

      # change the weights between input and middle layer
      for layer in range(len(node.weights)):
        for weight in range(len(node.weights[layer])):
          node.weights[layer][weight] -= eta * node.excitation * self.input_layer[layer][weight].excitation



      for semantic_weight in range(len(node.semantic_weights)):
        node.semantic_weights[semantic_weight] -= eta * node.excitation * self.semantic_layer[semantic_weight].excitation

    for upper_node in self.top_layer:
      # change the bias at the top layer
      upper_node.bias -= eta * upper_node.excitation
```

```
        # change the weights between middle and top layer
        for weight in range(len(upper_node.weights)):
            upper_node.weights[weight] -= eta * upper_node.excitation * self.middle_layer[weight].excitation

    def dreaming(self):
        """
        Implements the dreaming procedure as described in
        Boersma, Chládková & Benders (2022). First, activity is spread towards the bottom nodes.
        Then, using a Bernoulli distribution, some randomness is introduced into the network.
        """
        # spread activity to the input layer
        for input_layer in range(len(self.input_layer)):
            for input_node in range(len(self.input_layer[input_layer])):
                excitation_sum = self.input_layer[input_layer][input_node].bias
                for node in range(len(self.middle_layer)):
                    excitation_sum += self.middle_layer[node].weights[input_layer][input_node] * self.middle_layer[node].excitation

                self.input_layer[input_layer][input_node].excitation = excitation_sum

        # spread to the semantic nodes too?
        for semantic_node in range(len(self.semantic_layer)):
            excitation_sum = self.semantic_layer[semantic_node].bias
            for node in range(len(self.middle_layer)):
                excitation_sum += self.middle_layer[node].semantic_weights[semantic_node] * self.middle_layer[node].excitation

            self.semantic_layer[semantic_node].excitation = excitation_sum

        # use the bernoulli distribution to recalculate excitation at the top layer
        for upper_node in self.top_layer:
            excitation_sum = upper_node.bias

            # by spreading from the middle
            for node in range(len(self.middle_layer)):
                excitation_sum += self.middle_layer[node].excitation * upper_node.weights[node]

            # apply logistic function
            bernoulli_chance = logistic_function(excitation_sum)
```

```
        # apply Bernoulli distribution
        upper_node.excitation = 0 if random.random() >= bernoulli_chance else 1


    # use the bernoulli distribution to recalculate excitation at the top layer from both directions
    for node in range(len(self.middle_layer)):
        excitation_sum = self.middle_layer[node].bias

        # add from below
        for layer in range(len(self.middle_layer[node].weights)):
            for weight in range(len(self.middle_layer[node].weights[layer])):
                excitation_sum += self.middle_layer[node].weights[layer][weight] * self.input_layer[layer][weight].excitation

            for semantic_node in range(len(self.middle_layer[node].semantic_weights)):
                excitation_sum += self.middle_layer[node].semantic_weights[semantic_node] *
self.semantic_layer[semantic_node].excitation

        # add from above
        for upper_node in self.top_layer:
            excitation_sum += upper_node.excitation * upper_node.weights[node]

        # apply logistic function
        bernoulli_chance = logistic_function(excitation_sum)
        self.middle_layer[node].excitation = 0 if random.random() >= bernoulli_chance else 1


model = model()
model.run(3000)
model.obtain_results()
```