26 June 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

Angelica van Beemdelust 11238720
Supervisor: Paul Boersma

University of Amsterdam
BA Thesis Linguistics

Abstract

It is not controversial that the framework of neural networks can be used to replicate human cognitive abilities. In this thesis, the aspect of time is added to a neural network called a restricted deep Boltzmann machine, in order to replicate the human ability to recognize diphthongs. This is done with the use of distributional learning, a learning method used by children when acquiring the sounds of their first language. Different sequences of sound, either containing the same sounds, e.g. /au/-/ua/, or two completely different sounds, e.g. /au/-/ie/, are able to be distinguished. The network in this study makes use of a holistic model in which two representations of basilar membranes are present in the neural network. Secondly, a lexicon is added to the network to test for the machine's ability to distinguish between sounds. With this addition, the neural network is able to retain emerged phonological categories without the categories eventually disappearing. Not only is the model able to derive the correct lexical output from the sound input, the network is also able to derive the sound output from the lexical input. The sound level reveals the corresponding prototype diphthong learned from the input distribution, demonstrating its categorical behavior and the perceptual magnet effect. As the network works bidirectionally, this study further supports the theory of bidirectional phonology and phonetics, where both bottom-up and top-down processing follow the same constraints.

Table of contents

1. Introduction

Already before the age of 1, children learn to distinguish the important vowel sounds of their own language through the use of distributional learning and, as a result, stop hearing differences that are important in other languages of the world (Schure, Junge & Boersma, 2016; Dupoux, 2018; McMurray, Horst, Toscano & Samuelson, 2009; Maye, Werker & Gerken, 2002; Maye & Gerken, 2000). To further analyze how children learn and create phonological categories based on this distributional learning, we can apply artificial neural networks (Boersma, Benders & Seinhorst, 2018). These neural networks allow for bidirectional phonology and phonetic processing while also being able to model the evolution and acquisition of the phonology and phonetics (Boersma, 2011). The bidirectional model allows for both bottom-up processing as well as top-down processing. This means that from an auditory form it is possible to move up to the underlying form, as well as moving back down from the underlying form to the auditory form, demonstrating its bidirectionality across different levels of representation. Neural networks can be trained using different types of learning such as through the inoutstar algorithm (Boersma, Benders & Seinhorst, 2018) or the Hebbian algorithm, the latter of which is used in deep Boltzmann machines (Boersma, 2019). For this thesis, a restricted deep Boltzmann machine was chosen for its ability to create phonological categories (Boersma, 2019). A more thorough explanation will be provided in a later section of this thesis.

In Boersma's 2019 paper, it has been shown that the restricted deep Boltzmann machine is able to create phonological categories by using auditory distributional learning. This model demonstrates the perceptual magnet effect (Kuhl, 1991), where speech sounds are categorized, though the model fails with too much or too little training. With too little training, the network has not yet received enough input to create the different categories and considers them to be the same input. With too much training, the network imitates the actual input, which means the network no longer shows categorical behavior. Moreover, this network lacks the representation of time. While the aspect of time has been successfully added to other Recurrent Neural Networks with the use of Long Short-Term Memory (commonly known as LSTM) (Hochreiter & Schmidhuber, 1997) and is used in Automatic Speech Recognition programs, it has not yet been added to the deep Boltzmann machine models in which researchers attempt to replicate the cognitive behavior of humans. This thesis aims to add the aspect of time to a restricted deep Boltzmann machine with a lexicon included for sound–meaning pairs. Subsequently, it should be possible for the deep Boltzmann machine to recognize and differentiate between sequences of sound that use the same sounds in a different order. Moreover, it should be able to distinguish different sounds entirely while attaching the correct sequence of sounds to a word in the lexicon. Furthermore it is interesting to see how the addition of time influences the machine's ability to reflect categorical behavior. Finally, exploring the limitations of the model by removing meaning may help to better understand how phonological categories emerge. In section 2, the deep Boltzmann machine and its functionality, as used in this thesis, is explained. In the third section, the network structure is described and broken down into the different steps the model goes through. Section 4 of this thesis demonstrates the results of the different language environments that were tested within the neural network. The final sections, section 5 and 6, contains the discussion and conclusion respectively.

2. Deep Boltzmann machine

A deep Boltzmann machine is a type of stochastic recurrent neural network. Stochastic means that there are random variations in the neural network; in the case of the deep Boltzmann machine, the random variation can be found in the activation chance of the neuron. For example, if the weight between neuron A and neuron B is 0.7 and the activation of neuron A is 1, then this means there is a 70% chance that neuron A will fire to neuron B. A recurrent neural network is a type of artificial neural network that allows for bottom-up processing, as well as top-down processing. This means that the neural network can process from the input nodes up to the output nodes, but also from the output nodes, back down to the input nodes.

Restricted deep Boltzmann machines differ from deep Boltzmann machines in that they do not connect within layers, but only move up and down the different layers. In regular Boltzmann machines, neurons are able to influence each other either by activating or by inhibiting one another across levels or anywhere on the same level. The computational aspect of the regular Boltzmann machine is therefore very complex and would be required for very complex phenomena. However, in this thesis, a restricted deep Boltzmann machine will suffice and will be used for the sake of calculation simplicity.

3. The Network Structure

Diphthongs in and of themselves require the aspect of time. Without time, there is no change in sound and thus no diphthong can be created. In the neural model used in this thesis, a type of holistic model is used to represent time, in which there are two representations of basilar membranes that both interact with a representation of a lexicon simultaneously. In this case, the first basilar membrane represents the start of the diphthong, and the second the end of the diphthong. This holistic model differs from a type of sequential input model. The sequential model is a more realistic model that receives one input at a time and stores the previous one to create a chain of inputs until it can attach meaning to the sound input, similarly to the TRACE model (McClelland & Elman, 1986). Instead, when using a holistic model, the neural network uses two phoneme inputs simultaneously and recognizes them as diphthongs. This recognition means that the network can distinguish between sequences of sounds that contain the same phonemes but are in a different order. For example, the network can distinguish the difference between the sequence of the vowel /u/

followed by /i/ as the diphthong /ui/ and not as /iu/; and conversely, the sequence /i/ followed by /u/ is recognized as /iu/, not /ui/.

As mentioned in the paragraph above, this present study uses two representations of basilar membranes. By using a representation of the basilar membrane, the model represents human auditory processing more closely than without. This differs from models that use auditory distance instead, such as some self-organizing map networks (Kohonen, 1982; Guenther & Gjaja, 1996; Salminen, Tiitinen & May, 2009), or SOM-networks for short. By using a representation of the basilar membrane, the present study attempts to create a speech recognition program that more closely replicates human cognitive abilities in an attempt to better understand how sound is processed in the brain.

Activities: Parameters:

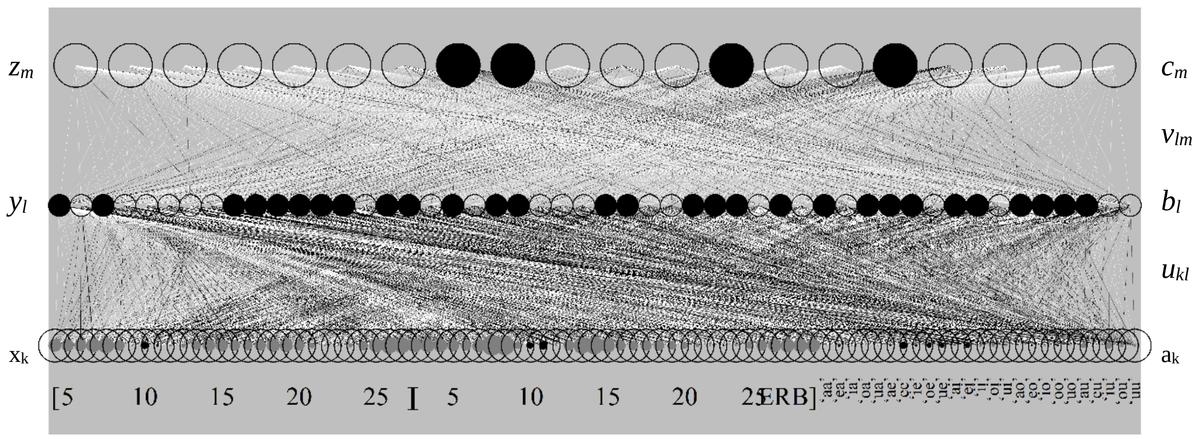

Figure 1 deep Boltzmann Machine after 10,000 inputs.

Figure 1 shows three different levels of nodes. The activity of each node is shown with black or grey circles. The black circles show positive activity whereas the grey circles show negative activity. The lowest level of nodes (*x*) represents three different types of so-called slabs. The first and second slab, running from node 1 to node 30, and from node 31 to 60 respectively, each represent the basilar membrane, and are two separate auditory-phonetic continua. As these slabs are representations of the basilar membrane, the first two slabs are marked every 5 ERB starting at 5 up to 25 ERB in figure 1, just below the bottom level. ERB stands for Equivalent Rectangular Bandwidth and is a measurement used for human hearing. The low frequencies are located on the left side of the slab, whereas the high frequencies are located on the right side of the slab. The third slab contains “words” that have meaning. In this case, diphthongs are given their own meaning node as if they were a word in the lexicon that can be recognized by the listener. Using the 5 vowels /a/, /e/, /i/, /o/ and /u/, the total number of possible words is 25 including combinations such as /aa/. Combinations of the same vowels are referred to as elongated or long vowels, as these are technically not diphthongs. With 25 total possible words, which means that the lexicon is complete, the meaning slab contains 25 nodes. The activities of the lowest level of nodes are defined as (x_k) where k runs from 1 to K where K is a variable total number of nodes between 60 and 85, depending on the number of meanings included in the network. In a network that contains no meaning at all, K = 60 as this is equal to the number of auditory nodes. In a network containing a full lexicon, K = 85, as there are 25 extra meaning nodes on top of the 60 auditory nodes.

While other models often refer to the lowest level as the "input level", it may be misleading for the model used in this thesis to call it that. When trained, the model should be able to determine the activities on the meaning slab based on input from the basilar membrane slabs. Vice versa, the model should also be able to determine the expected sound on the basilar membrane slabs based on the meaning slab. This creates the bidirectional nature of the model, as is desired. Considering that both the slabs containing the representation of the basilar membrane and the meaning slab are on the same level, we could potentially say that this level is both the input and the output level for each category. For lack of a better term, this input and output level shall be referred to as the bottom level as it is the bottom level of the visual representation of the model in figure 1.

The second level (y) has activities (y_l) with l from 1 to $L = 50$, meaning there are a total of 50 nodes on the middle level. Finally the third, top level (z) has activities (z_m) with m running from from 1 to M = 20, meaning there are a total of 20 nodes on the top level. These two upper levels are levels that contain hidden, binary nodes, meaning they contain nodes that can either be on or off. The middle and top levels represent a type of long-term memory for the network.

Each level in the model contains biases, which are a type of offset that is an extra input to neurons, with its own connection weight. Biases control when the neuron activates and are a constant to a function. For example, a bias represents b in the simple function $f(x) = ax + b$. In this thesis, we define the biases of our neural network per level. The bottom level has biases (a_k), the middle level (b_l) and the top level has biases (c_m).

Evidently, these different levels are required to interact with each other. The bottom level is connected to the middle level with weights (u_{kl}) and the middle level is connected to the top level with weights (v_{lm}). In figure 1 both black and white connections are visible between the nodes. The black nodes indicate a positive connection between two nodes. On the other hand, white connections mean a negative connection between two nodes. Thus if a node is activated, with a positive weight, the node it is connected to is more likely to activate, and vice versa with a negative weight, it is less likely to activate. A positive weight is called excitatory and a negative weight is called inhibitory.

The name "restricted deep Boltzmann machine" is somewhat deceiving in that it suggests being a single type of neural network. In reality, there are differences between networks that fall under this same name. In the neural network used for this thesis, the network will go through different consecutive phases for the training procedure. First, in the initial settling phase the network will be set up. Secondly, the network will start its learning process in the Hebbian learning phase. In the third phase, the network will go through the dreaming phase. Finally, the network will be updated with the anti-Hebbian learning phase. As this model is based on Boersma's model, the following mathematical equations are the same as in Boersma's 2019 paper.

3.1. The initial settling phase

Before even starting the initial settling phase, the network is given an input on the bottom level. In the initial settling phase, the activity of the nodes at the bottom level (x_k) are "clamped", meaning they are held at a constant at the activity with which the network was provided. The activity on the bottom level spreads up to the middle level (y_l) for all l from 1 to L, starting with the activities (z_m) at 0.

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

where σ() is monotonic and nonlinear, thus either entirely increasing or entirely decreasing. In (2) the standard logistic function is shown.

(2) $\sigma(x) \coloneqq 1/(1 + \exp(-x))$

After the activities of (y_l) have been calculated, the activities of the top level (z_m) are computed in a similar way as (y_l) is computed, where the activities are calculated for all m from 1 to M. Unlike the middle layer, the top level is not connected to the clamped bottom level, and is not directly influenced the activities.

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

Then (1) through to (3) are repeated 10 times while the actitvities (x_k) remain clamped. This resonance deterministically brings the network to a relatively stable state. Once this equilibrium state is reached, the network can move on to the Hebbian learning phase.

3.2. The Hebbian learning phase

In the Hebbian learning phase, the network increases the weight of any positive connection between two active nodes, thus strengthening their connection. This results in both nodes being active at the same time more frequently. Moreover, the node that is active receives a higher bias, making it even more likely to be active in the future. Using a learning rate of 0.001 assigned to η, we arrive at the following functions.

(4) $a_k \leftarrow a_k + \eta x_k$

(5) $b_l \leftarrow b_l + \eta y_l$

(6) $c_m \leftarrow c_m + \eta z_m$

(7) $u_{kl} \leftarrow u_{kl} + \eta x_k y_l$

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

3.3. The dreaming phase

After the Hebbian learning phase, we arrive at the dreaming phase. During this phase the model will create its own pattern, similar to dreaming. In this phase, the bottom level (x_k) can be influenced by the middle level (y_l), meaning that the bottom level is no longer clamped. From this follows function (9).

(9) $x_k \leftarrow a_k + \sum_{l=1}^{L} u_{kl} y_l$

After that, for (z_m) and (y_l) new values are computed stochastically, the top level before the middle level.

(10) $z_m \sim \mathcal{B}\left(\sigma(c_m + \sum_{l=1}^{L} y_l v_{lm})\right)$

(11) $y_l \sim \mathcal{B}\left(\sigma(b_l + \sum_{k=1}^{K} x_k u_{kl} + \sum_{m=1}^{M} v_{lm} z_m)\right)$

In these two formulas $\mathcal{B}(\ )$ represents a Bernoulli deviate. This Bernoulli distribution leads to a binary option, 0 or 1. For example, if the sigmoid function derived from formula (10) shows us a result of 0.7, then the Bernoulli function will give us an output (z_m) of 0 or 1 with a probability of 0.7 of (z_m) being 1. Then the formulas (9) through to (11), similarly to the in the settling phase, are repeated 10 times. Unlike the initial settling phase, however, formulas (10) and (11) contain a randomized algorithm. The random variation combined with the initial real inputs ensures that, ultimately, the distribution of possible activation patterns are faithfully sampled by all possible activation patterns in the network.

3.4. The anti-Hebbian learning phase

Finally, the anti-Hebbian learning phase is the same as the Hebbian learning, except rather than the weights getting strengthened and biases increasing, the weights are generally weakened and the biases are decreased, unless the weights were negative to begin with. When weights and biases are negative, then they are strengthened and increased respectively. This anti-Hebbian learning phase is required so that the weights between neurons do not infinitely grow larger.

(12) $a_k \leftarrow a_k - \eta x_k$

(13) $b_l \leftarrow b_l - \eta y_l$

(14) $c_m \leftarrow c_m - \eta z_m$

(15) $u_{kl} \leftarrow u_{kl} - \eta x_k y_l$

(16) $v_{lm} \leftarrow v_{lm} - \eta y_l z_m$

3.5 Distributional learning

As mentioned previously, the first two slabs on the bottom level running from node 1 to node 30 and from node 31 to node 60 respectively, are representations of the basilar membrane. On each slab, the input continuum ranges from the first node which corresponds to the lowest basilar frequency and to the last node which corresponds to the highest basilar frequency per slab. To create a vowel, two inputs are required per slab. The two inputs represent the F1 and F2 values of a vowel. In total, four inputs are required for both slabs in order to create the diphthong. For each vowel used in the model, in this case /a/, /e/, /i/, /o/, and /u/, there is an equal probability (0.2) of one being randomly chosen as the input vowel.

Table 1 mean ERB values per vowel

Vowels	a	e	i	o	u
mean ERB F1	13	10	7	10	7
mean ERB F2	19	22	25	16	13

From the mean F1 and F2 of the chosen vowels, the F1 and F2 values are sampled in ERB from a secondary script provided in the appendix, in which the mean ERB values per vowel are predetermined. In Table 1, the F1 and F2 in ERB are given per vowel. These F1 and F2 values in ERB are sampled with a standard deviation of σ = 0.9 ERB. This results in the following formula where *w* = 1.5 is the half-width of the Gaussian peak on the basilar membrane:

(17) $$x_k = 5\, e^{-\frac{1}{2}\left(\frac{k-F1}{w}\right)^2} + e^{-\frac{1}{2}\left(\frac{k-F2}{w}\right)^2} - 0.5$$

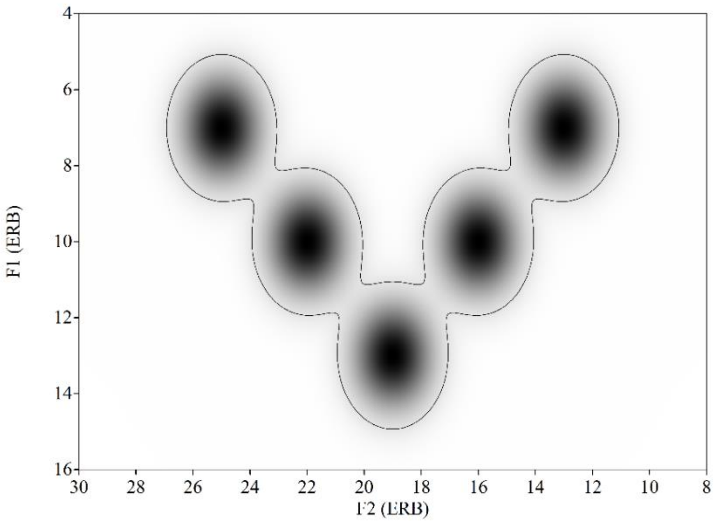

Figure 2 Input distribution of the 5 vowels /a/, /e/, /i/, /o/, and /u/

In figure 2, the input distribution of the vowels /a/, /e/, /i/, /o/, and /u/ are shown on a two-dimensional plane. The vertical axis shows the F1 in ERB, the horizontal shows the F2 in ERB. The ellipses are at a relative height of 10% on each distribution. From left to right the vowels corresponding to each point on the graph are /i/, /e/, /a/, /o/, and finally /u/. As can be seen in figure 2, there are areas in which vowels overlap.

4. Simulations of 5 different languages: language description and results

As there are different possible numbers of meaning nodes in slab 3, multiple test runs have been carried out to analyze the differences between different number of meaning nodes. In the following sections, 5 different results will be shown. The results differ from each other in that the number of meanings differ between the sections. We can compare this to different sections resembling different language environments, similarly to how English differs from French and how both of these two languages differ from Japanese.

In the first section of the results, a language environment including all possible meaning nodes will be described. While this is unrealistic for natural languages, this language environment shows all possible effects and combinations thereof without worrying about possible gaps. In the second section, all elongated vowels are removed from the meaning slab to create a gap. This means that ‘aa’, ‘ee’, ‘ii’, ‘oo’, and ‘uu’ are no longer in the list. This language environment was used as there are languages that make no distinction between short and long vowels. The third language environment contains 5 randomly chosen gaps. This environment was created to see what would happen when the gaps are not correlated to one another. Finally, the fourth language environment lacks the vowel /a/ in any position, and the fifth language environment lacks the vowel /u/ in any position. These environments have been made to compare the results in their similarity to other vowels.

4.1 All-meanings language

In the first language of the neural network, all possible “meanings” were included in the lexicon. In other words, all possible combinations of the five vowels /a/, /e/, /i/, /o/ and /u/ combined into diphthongs (e.g. /au/) or combined into elongated vowels (e.g. /aa/) had a node on the meaning slab at the bottom level. This means there are 25 nodes in total on the meaning slab. Let’s say the neural network received 30,000 inputs, each meaning on the meaning slab has been activated approximately 30,000/25 = 1,200 times and each vowel on each sound slab has been activated 30,000/5= 6000 times. For both slabs that would mean that all vowels are activated 6000*2 = 12.000 times in total. The network starts off with the initial settling phase as described in section 3.1 before moving on to the different learning stages described in section 3.2 to 3.4. The first learning phase is the Hebbian learning phase, in which any connection between two neurons is strengthened and the weight is increased. Following the Hebbian learning phase is the dreaming phase, in which the network stochastically resonates 10 times without the initial input being clamped anymore. Finally the network goes through the anti-Hebbian learning phase, in which the connection between two neurons is weakened in case of a positive connection, or strengthened in case of a negative connection.

In the first testing phase, from sound to meaning, the user chooses a sound either through a keyboard input or mouse click on each of the two slabs. For example, the user chooses the diphthong /ui/. This input is clamped on the bottom level. Then the input is resonated through the middle and top level of the network until it reaches a near equilibrium state. After reaching this near equilibrium state, the network activity on the middle level is resonated down towards the meaning slab on the bottom level. The sound slabs on the bottom level remain clamped and unchanged. Then the activity on the middle level and meaning slab is resonated until the network reaches a state of equilibrium again.

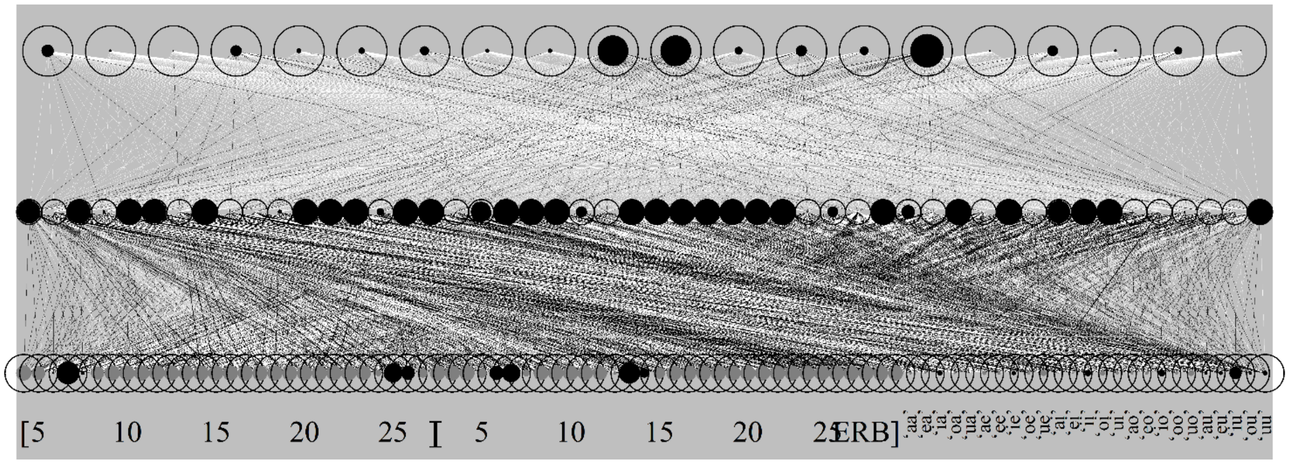

Figure 3 Neural network test from sound to meaning of diphthong /au/ after 30,000 inputs.

Figure 3 shows the neural network and its activities on the different levels. The first slab on the bottom level shows a low F1 and high F2, corresponding to the vowel /i/. The second slab shows a low F1 and low F2, corresponding to vowel /u/. Therefore, the sound slabs show the diphthong [[iu]]. On the third slab, the meaning slab, the most activated node is ‘iu’.

Table 2 activation list of all meaning nodes.

Final -a	Activation	Final -e	Activation	Final -i	Activation	Final -o	Activation	Final -u	Activation
‘aa’	-0.367	‘ae’	-0.512	‘ai’	-0.152	‘ao’	-0.365	‘au’	0.381
‘ea’	-0.204	‘ee’	-0.394	‘ei’	-0.259	‘eo’	-0.466	‘eu’	0.493
‘ia’	0.656	‘ie’	0.550	‘ii’	0.841	‘io’	0.858	**‘iu’**	**1.635**
‘oa’	-0.289	‘oe’	-0.412	‘oi’	-0.637	‘oo’	-0.383	‘ou’	0.386
‘ua’	-0.060	‘ue’	-0.756	‘ui’	-0.373	‘uo’	-0.790	‘uu’	0.623

Table 3 positive activations for meaning nodes starting with i- and meaning nodes ending with -u

Meaning node i-	Activation	Meaning node -u	Activation
‘ia’	0.656	‘au’	0.381
‘ie’	0.550	‘eu’	0.493
‘ii’	0.841	**‘iu’**	**1.635**
‘io’	0.858	‘ou’	0.386
‘iu’	**1.635**	‘uu’	0.623

As figure 3 shows, the initial input [[iu]] on the sound slabs corresponds to the meaning node ‘iu’, which is the most activated node on the meaning slab with an activation of 1.635 as can be seen in table 2 and 3. Other nodes with a positive activity are presented in table 3 for better visualization. The nodes that are activated all have a vowel in common with the actual sound input. Either the first part of the diphthong is the same, here the first vowel i-, or the second part of the diphthong is the same, here the vowel -u. All other nodes that are not related to the sound input show a negative activity as presented in table 2. This includes the node ‘ui’ with an activation of -0.373. So while the meaning node ‘iu’ was the most activated, the meaning node ‘ui’ is not activated at all, further proving the network is able to distinguish between different sequences of sounds despite approximately the same sounds being used. While only one example is demonstrated here, the same effects are found for all other diphthongs and long vowels. The network also functions when trained 3,000 times, however the differences in activations are significantly smaller with this amount of training.

The second test is from meaning to sound. This is to demonstrate the possibility of bidirectionality in the neural network. In figure 4 shown below, the meaning input given by the user is ‘eo’. Similarly to how meaning was derived from sound, sound is derived from meaning through resonance until the network reaches near equilibrium states.

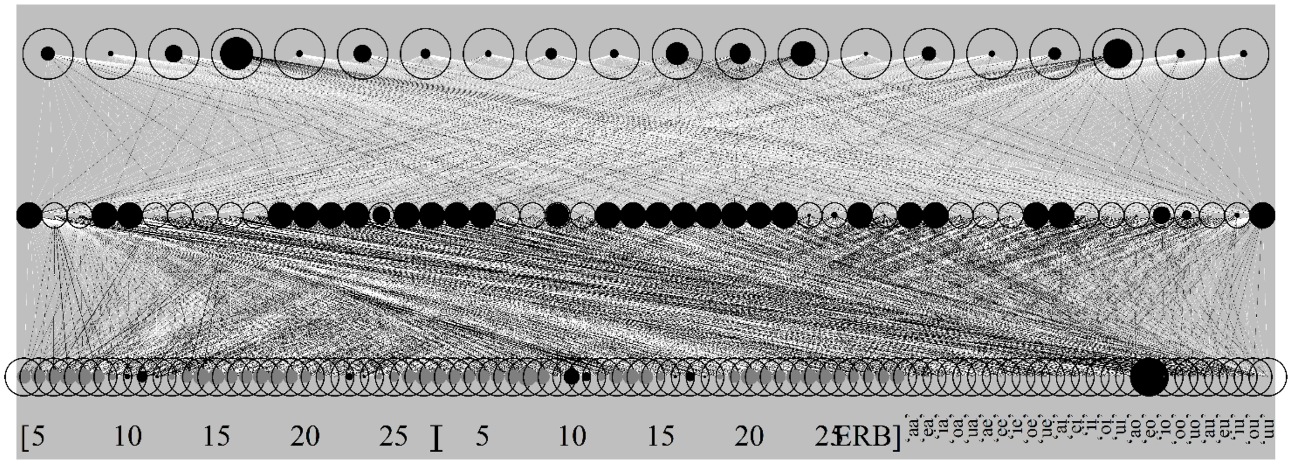

Figure 4 the neural network from meaning 'eo' to sound after 30,000 inputs.

Earlier in table 1, we saw that the mean ERB of vowel /e/ was F1 = 10 and F2 = 22, and the mean ERB of vowel /o/ was F1 = 10 and F2 = 16. In figure 4, similar results are shown. The two areas of activations on the first slab are approximately at 10 to 11 ERB for F1, and 22 to 23 ERB for F2, corresponding to the vowel /e/. The areas of activations for the second slab are approximately at 10 to 11 ERB for F1 and 16 ERB for F2, corresponding to the vowel /o/. We can conclude that the sound derived from the meaning node 'eo' does indeed correspond to the sound [[eo]], thus demonstrating the machine's ability to not only move bottom-up, but also top-down. This adheres to the bidirectional model of phonology and phonetics as suggested by Boersma (2011). It should be noted that although the diphthongs and vowels on the bottom slab were written between slashes in this paper, which corresponds to the surface form in the bidirectional model, they more closely represent the auditory form, which is written in double brackets.

Unlike Boersma's 2019 network, this neural network retains its categorical behavior even when trained 30,000 times. This is because the network does not only have sound input to rely on, but it also contains the meaning of the sounds. The meaning appears to be a placeholder for the categorical behavior of the neural network.

Using the neural model, we can compare the different diphthongs and determine how similar they are to one another by creating a matrix. This is done with cosine similarity. First we find the norm of vector a_i and b_i from for i from 1 to n, where n is the number of dimensions, in this case $n = 25$ as there are 25 meaning nodes.

(18) $$A_i = \frac{a_i}{\sqrt{\sum_{j=1}^{n} a_j^2}}$$

(19) $$B_i = \frac{b_i}{\sqrt{\sum_{j=1}^{n} b_j^2}}$$

This leaves the length of A and B at 1. The cosine similarity is equal to the inner product or dot product of vector A_i and B_i.

(20) $$Similarity = cos\,(\theta) = \sum_{i=1}^{n} A_i B_i,$$

where θ is the angle between vector A_i and B_i. When the vectors A_i and B_i are the identical, the cosine similarity is equal to 1 whereas if A_i and B_i are perpendicular to each other, the cosine similarity is equal to 0. This leaves us with a matrix full of numbers between 0 and 1. For clarity, we multiply the numbers in the matrix by 100 and round them to produce whole number values. This leaves us with a matrix with a range of values from 0 to 100 where 0 means there is no resemblance between A_i and B_i at all and a score of 100 means A_i and B_i are identical. In other words, the sounds produced are the same as one another. The following matrix is a result of a network that has received 30,000 inputs and is well-trained.

Table 4 Similarity of Sounds Matrix

	aa	ea	ia	oa	ua	ae	ee	ie	oe	ue	ai	ei	ii	oi	ui	ao	eo	io	oo	uo	au	eu	iu	ou	uu
aa	100	84	75	79	90	81	67	65	60	69	78	66	57	60	67	82	70	64	62	74	87	75	67	68	76
ea	84	100	81	88	84	67	80	65	72	64	63	78	59	69	60	64	80	64	70	65	75	87	69	76	70
ia	75	81	100	79	82	62	69	80	66	67	60	61	79	60	65	60	64	80	64	66	65	71	89	67	71
oa	79	88	79	100	79	61	72	61	81	61	63	72	60	80	61	61	70	64	82	59	73	81	70	88	68
ua	90	84	82	79	100	69	62	66	58	77	68	64	64	59	77	69	67	69	60	81	80	74	74	67	87
ae	81	67	62	61	69	100	84	82	79	87	78	65	58	60	66	90	74	70	69	79	78	64	58	59	68
ee	67	80	69	72	62	84	100	85	91	81	62	77	64	71	58	73	87	76	81	71	65	76	64	71	61
ie	65	65	80	61	66	82	85	100	79	85	65	63	78	61	68	72	72	88	69	75	61	61	76	60	66
oe	60	72	66	81	58	79	91	79	100	79	62	71	62	80	59	67	77	70	90	65	61	69	61	80	59
ue	69	64	67	61	77	87	81	85	79	100	65	61	65	60	74	76	71	75	68	87	68	61	64	61	77
ai	78	63	60	63	68	78	62	65	62	65	100	84	79	82	89	75	59	60	59	63	88	75	71	74	80
ei	66	78	61	72	64	65	77	63	71	61	84	100	81	91	82	61	75	60	68	59	78	91	72	83	77
ii	57	59	79	60	64	58	64	78	62	65	79	81	100	79	87	56	59	76	59	62	66	72	91	71	76
oi	60	69	60	80	59	60	71	61	80	60	82	91	79	100	81	56	65	59	78	54	73	82	71	92	72
ui	67	60	65	61	77	66	58	68	59	74	89	82	87	81	100	62	56	66	56	72	80	73	78	73	90
ao	82	64	60	61	69	90	73	72	67	76	75	61	56	56	62	100	83	80	77	87	74	60	55	56	65
eo	70	80	64	70	67	74	87	72	77	71	59	75	59	65	56	83	100	83	87	84	62	74	58	65	59
io	64	64	80	64	69	70	76	88	70	75	60	60	76	59	66	80	83	100	80	86	55	60	75	58	64
oo	62	70	64	82	60	69	81	69	90	68	59	68	59	78	56	77	87	80	100	75	58	67	58	77	55
uo	74	65	66	59	81	79	71	75	65	87	63	59	62	54	72	87	84	86	75	100	66	58	62	54	75
au	87	75	65	73	80	78	65	61	61	68	88	78	66	73	80	74	62	55	58	66	100	87	76	81	90
eu	75	87	71	81	74	64	76	61	69	61	75	91	72	82	73	60	74	60	67	58	87	100	82	89	83
iu	67	69	89	70	74	58	64	76	61	64	71	72	91	71	78	55	58	75	58	62	76	82	100	78	89
ou	68	76	67	88	67	59	71	60	80	61	74	83	71	92	73	56	65	58	77	54	81	89	78	100	80
uu	76	70	71	68	87	68	61	66	59	77	80	77	76	72	90	65	59	64	55	75	90	83	89	80	100

From this matrix there are a few things worth noting, starting with the elongated vowels as a base. The matrix shows that /uu/ is both similar to /ii/ and /aa/. Both of these long vowels have a score of 76 when compared to /uu/. For /ii/, the similarity is unsurprising, as both /u/ and /i/ are high vowels and, as a result, have a similar F1. However, for /aa/ and /uu/ the similarity may be more remarkable. /a/ is a front open unrounded vowel whereas /u/ is a closed back rounded vowel. Even if we were to consider /a/ a central vowel due to its height, rather than a front vowel, it does not explain the similarities between the two vowels. It is possible that the correlation is derived from the similarity of the F2 of /u/ and F1 of /a/. The similarity in elongated vowels are translated into the diphthongs as well. /uu/ is close to /au/ (90) and /ua/ (87) as well as /iu/ (89) and /ui/ (90).

Next, the matrix shows close similarity of /ee/ and /oo/ with a score of 81. This is likely because these vowels share the same vowel height at close-mid level and, similarly to /i/ and /u/, share a similar F1. The diphthongs once again reflect this similarity where /ee/ is close to /oe/ (91) and /eo/(87), and /oo/ close to /oe/(90) and /eo/ (87).

Using the matrix, it is also possible to compare different sequences of sound. /ua/-/au/ as well as /ui/-/iu/ and /eo/-/oe/ are expected to have higher similarity than for example /ei/-/ie/ or /ia/-/ai/ because of the similarity in vowel quality as discussed earlier in this section. This turns out to be the case. /au/-/ua/ have a similarity of 80, /ui/-/iu/ a similarity of 78 and /eo/-/oe/ a similarity of 77. /ei/-/ie/ have a similarity of only 63, and /ia/-/ai/ even less with 60. Comparing these last two numbers to unrelated diphthongs such as /ea/-/io/ and /ou/-/ae/ with a score of 64 and 59 respectively, there is no real difference between different sequences of sounds and completely different diphthongs. This provides further evidence that the order of sounds is considered more important by the neural network than the sounds themselves.

4.2 Short vowel language

In this language, all long vowels are removed. This test was run because there are natural languages without distinction between long and short vowels. When removing the long vowel meaning nodes ‘aa’, ‘ee’, ‘ii’, ‘oo’, and ‘uu’, we are left with 20 meaning nodes. However, it still remains possible for both slabs to produce two of the same vowels in a row given by the user. For each of the elongated vowels, the neural network then has to come up with an alternative meaning that would come closest to the perceived sound.

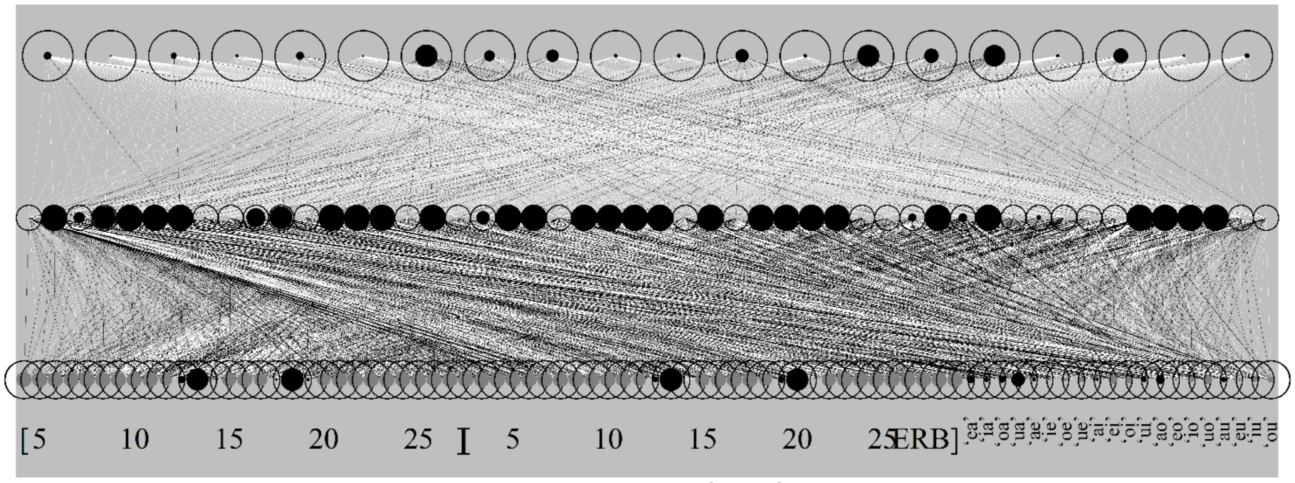
[5 10 15 20 25 I 5 10 15 20 25ERB]
‘ea’ ‘ia’ ‘oa’ ‘ua’ ‘ae’ ‘ie’ ‘oe’ ‘ue’ ‘ai’ ‘ei’ ‘oi’ ‘ui’ ‘ao’ ‘eo’ ‘io’ ‘uo’ ‘au’ ‘eu’ ‘iu’ ‘ou’

Figure 5 The network with a short vowel language with a sound input of /aa/ after 30,000 inputs.

After training with 30,000 inputs, the network attempts to pick out the meanings containing the vowel given in the input. In this case, the input is /aa/. However, as can be seen in figure 5, the most activated meaning node is ‘ua’ with an activation of 1.753. The next highest activation is by ‘ao’ with an activation of 1.153. In the previous section, we discovered that the similarities between /a/ and /u/ are quite high, thus explaining why ‘au’ is the most activated meaning node. However, when doing multiple runs, the results are not always consistent. Sometimes, other meaning nodes are activated by the network. The meaning nodes that are activated contain /a/ in either first or second position more so than meaning node ‘au’ despite the latter being the most similar to the input sound /aa/. This variation may be caused by the stochasticity of the model as well as the slight deviations from the prototype vowel, the most commonly heard vowel that is at the top of the distribution.

4.3 Random gap language

In this language of the neural network, 5 meaning nodes were removed at random. This was to test the machine’s behavior when unrelated gaps were found in the language environment. The meaning nodes that were removed were ‘ea’, ‘eo’, ‘iu’, ‘ue’, and finally ‘uo’. This leaves the neural network with 20 meaning nodes on the bottom level.

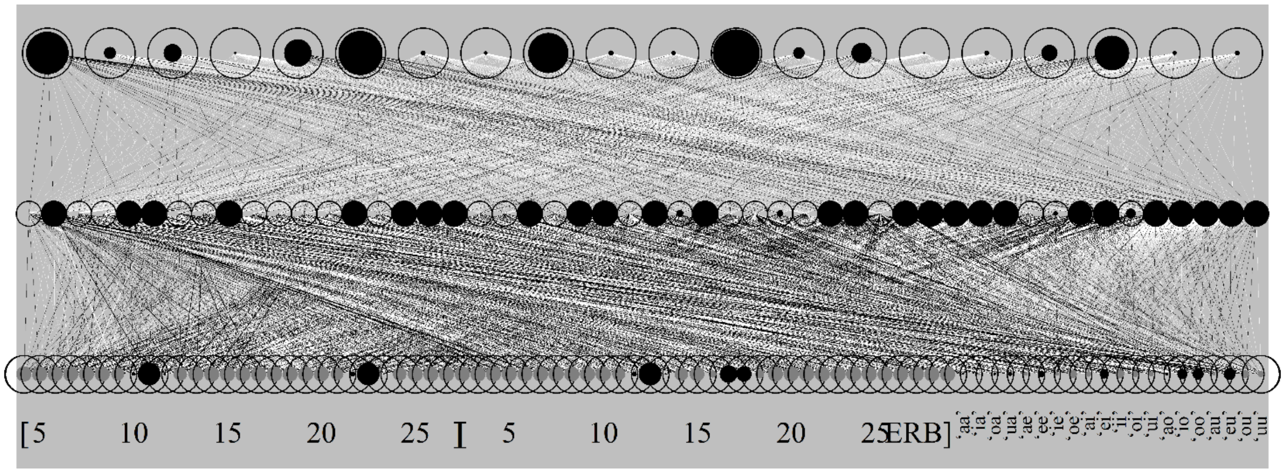

Figure 6 The network with a 5 gap language with sound input /ea/ after 30,000 inputs.

Similarly to some test runs in the short vowel language in section 4.2, when the Boltzmann machine was introduced to the missing meaning on the sound slabs, no clear winner was found in the meaning slab when the missing inputs were given on the sound slab. The nodes with the same vowel either on slab 1 or slab 2 were positively activated, but none were activated enough compared to another for there to be a clear winner. It is possible that over multiple runs, vowels that are similar to one another are more likely to be the most activated node. For example, with ‘ea’ removed, ‘eu’ may become the most frequently highest activated node. However, similarly to 4.2, this was not consistent.

4.4 a-less language

In this run of the neural network, instead of removing random meaning nodes or the elongated vowels, the vowel /a/ was completely removed from meaning. This also means that the network was not trained to perceive the vowel /a/ and the vowel was not included in the input distribution. This leaves 16 meaning nodes left on the bottom level.

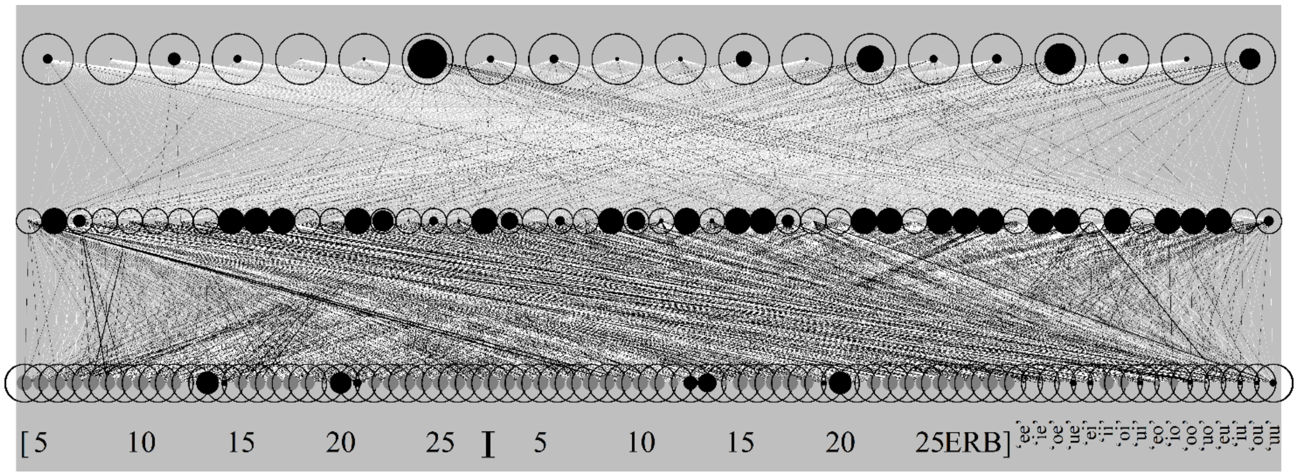

Figure 7 The network with an a-less language with sound input /aa/ after 30,000 inputs.

Earlier in section 4.1, with the similarity matrix, it was established that /u/ and /a/ showed to have some correlation to each other. While section 4.2 and 4.3 did not consistently demonstrate this correlation, when the vowel /a/ is completely removed in the meaning, their similarity does end up being visible in the network. After introducing the network to 30,000 inputs, when being confronted with /aa/ in the sound input as shown in figure 7, the meaning nodes that are more activated than others are those containing /u/, with ‘uu’ being the most activated node with an activation of 0.945. All other nodes that do not contain this vowel /u/ have a negative activation. However, this activity remains minimal, and when the sound input contains /u/ either on slab 1 or 2, the corresponding nodes are activated much more than when introduced to /a/. Moreover, even though the network often activates ‘uu’, there is still a chance of the network activating another meaning node. This is likely a result of the overlap in distributions, as was shown in Figure 2.

4.5 u-less language

Because leaving out the vowel /a/ showed interesting correlation of the similarities between /a/ and /u/, leaving out the vowel /u/ would be interesting as this vowel is related to both /a/ and /i/. Once again, the network contained 16 meaning nodes, now excluding the vowel /u/.

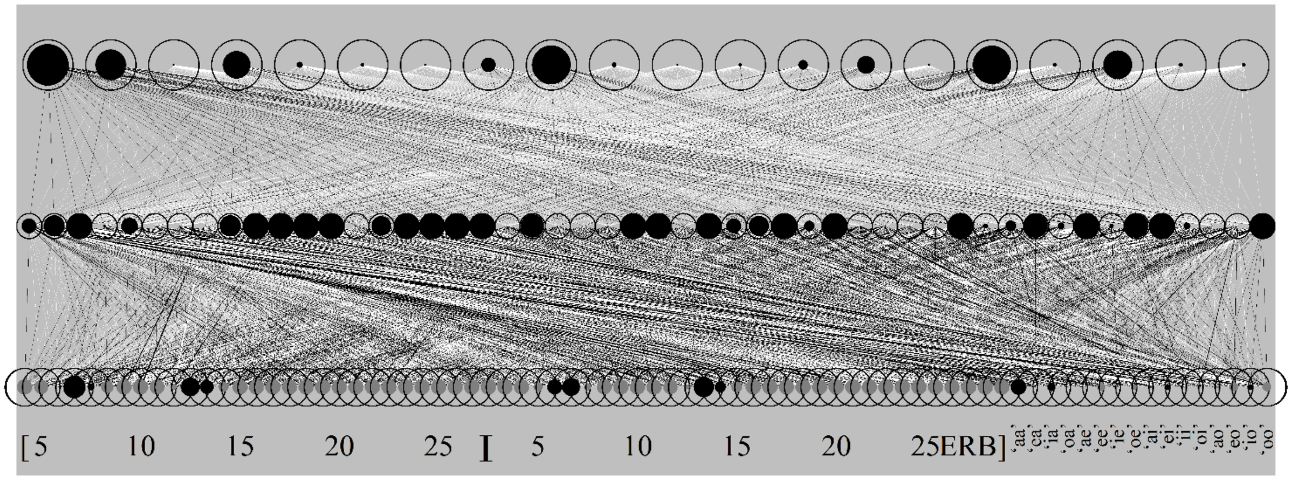
[5 10 15 20 25 I 5 10 15 20 25ERB]
‘aa’ ‘ea’ ‘ia’ ‘oa’ ‘ae’ ‘ee’ ‘ie’ ‘oe’ ‘ai’ ‘ei’ ‘ii’ ‘oi’ ‘ao’ ‘eo’ ‘io’ ‘oo’

Figure 8 The network with an u-less language with sound input /uu/ after 30,000 inputs.

Like Section 4.4, when the network was trained on 30,000 inputs and introduced to /uu/ on the sound slabs only through user input. In figure 8, the meaning node the most activated is ‘aa’. However, this was not a consistent result. Over the course of several runs, the meaning nodes that were partially activated were those that contained the vowel /a/ and those that contained the vowel /i/. Neither vowel showed more activation than the other and both vowels only showed minimal activation on the meaning nodes compared to when they were introduced to the corresponding vowels /a/ and /i/.

5. Discussion

This restricted deep Boltzmann machine model was able to distinguish between different sequences of sound with the addition of time added through a holistic model. This was tested in five different languages environments. The language containing all possible meanings allowed the machine to distinguish between all sequences of sound correctly. The following gap languages showed activation of diphthongs similar to the input, although the network was not able to choose similar inputs consistently. This is likely due to the overlap of vowel distributions. As the model contained not only sound, but also meaning, it was able to retain the categorical behavior even after 30,000 inputs, which was not the case for Boersma’s 2019 model. This is because the network is able to use meaning as a placeholder for the prototype vowel. As a result, it also modeled the perceptual magnet effect for vowels correctly regardless of being combined into diphthongs on the meaning slab. Moreover, the neural network follows the bidirectional model for phonology and phonetics in that the network is able to pick the correct meaning for a given input sound, and vice versa. However, the model does not capture the different levels proposed. The neural model only works from the top level meaning (<morphemes>), down to the second to last level, sound ([[auditory form]]), skipping the intermediate stages of the surface form and underlying form. The addition of time can be used for future research on modeling human cognitive abilities related to auditory perception. For future research, it may be worth looking into not only diphthongs but also triphthongs. Furthermore, it may be interesting to replicate not only vowels, but also consonants, consonant clusters or combinations of vowels and consonants. While the holistic model works well for the goal of this research, it fails to represent true human cognition as it uses multiple basilar membranes that supposedly capture different points in time despite the input on each slab being given by the user at the same time. Instead a more realistic model would be one in which some type of memory captures the input at different points in time on the same input slab or level to finally reach the correct output.

6. Conclusion

For this thesis, the goal was to add the aspect of time to a restricted deep Boltzmann machine. After this initial goal had been reached, the network was tested in different language environments for its ability to distinguish between sequences of sounds that either contain the same sounds or different sounds as well as its ability to create emergent phonological categories. All sequences of sounds were appropriately distinguished from one another unless the network was introduced to a sound it had not yet been trained on. In the latter case, the network would choose a sequence from the lexicon that was similar to the input sound, albeit inconsistently. From the lexicon down to the sound level, the network was able to create the prototype vowel, which can be compared to a phonological category. As the network works from both meaning to sound, and from sound to meaning, it is bidirectional, supporting bidirectional phonology and phonetics. The neural network used in this study directly builds upon the network made by Boersma's 2019 paper. Although the network does not use memory to include the aspect of time, it more accurately replicates human perception of sounds than other models that use auditory distance. It may be interesting to see how this model responds to longer sequences of sounds or sounds other than vowels.

7. References

Boersma, Paul. 2011. A programme for bidirectional phonology and phonetics and their acquisition and evolution. In Anton Benz & Jason Mattausch (eds.) *Bidirectional Optimality Theory*. 33–72. John Benjamins Publishing Company.

Boersma, Paul. 2019. Simulated distributional learning in deep Boltzmann machines leads to the emergence of discrete categories. *ICPhS*. 1520-1524.

Boersma, Paul, Titia Benders & Klaas Seinhorst. 2020. Neural network models for phonology and phonetics. University of Amsterdam. To appear in *Journal of Language Modelling.*

Dupoux, Emmanuel. 2018. Cognitive science in the era of artificial intelligence: A roadmap for reverse-engineering the infant language learner. *Cognition* 173, 43–59.

Guenther, Frank H. & Marin N. Gjaja. 1996. The perceptual magnet effect as an emergent property of neural map formation. *Journal of the Acoustical Society of America*. 100(2). 1111-1121.

Hochreiter, Sepp & Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9(8). 1735-1780.

Kohonen, Teuvo. 1982. Self-organized formation of topologically correct feature maps. Biological Cybernetics. 43. 59-69.

Kuhl, Patricia K. 1991. Human adults and human infants show a “perceptual magnet effect” for the prototypes of speech categories, monkeys do not. *Perception and Psycholinguistics* 50, 93–107.

Maye, Jessica, & LouAnn Gerken. 2000. Learning phonemes without minimal pairs. In S. Catherine Howell, Sarah A. Fish & Thea Keith-Lucas (eds.) *Proceedings of the 24th Boston University Conference on Language Development* 522–533. Somerville, MA: Cascadilla Press

Maye, Jessica, Janet. F Werker & LouAnn Gerken. 2002. Infant sensitivity to distributional information can affect phonetic discrimination. *Cognition 82*, B101–B111.

McClelland, James, Jeffrey Elman. 1986. The TRACE model of speech perception. *Cognitive Psychology* 18(1). 1-86.

McMurray, Bob, Jessica S. Horst, Joseph C. Toscano & Larissa K. Samuelson. 2009. Integrating connectionist learning and dynamical systems processing: case studies in speech and lexical development. In John Spencer (ed.). *Toward a unified theory of development: connectionism and dynamic systems theory re-considered*. 218–249. Oxford University Press.

Salminen, Nelli H., Hannu Tiitinen & Patrick J.C. May. 2009. Modeling the categorical perception of speech sounds: a step toward biological plausibility. *Cognitive, Affective, & Behavioral Neuroscience.* 9(3). 304-313.

Ter Schure, Sophie, Caroline Junge & Paul Boersma. 2016. Semantics guide infants’ vowel learning: computational and experimental evidence. *Infant Behavior and Development*. 43. 44–57.

Appendices

Appendix 1: Main script

```
# Praat script simul_dbm_diphthongs.praat
# Angelica van Beemdelust 06-06-2020
# Based on script used in Boersma, Paul. 2019. Simulated distributional learning in
deep Boltzmann machines leads to the emergence of discrete categories. ICPhS.

form Emergence of three vowels
	word Foreground_colour Yellow
	word Background_colour Maroon
	word Button_colour Olive
	word Font Times
	natural Font_size 35
	boolean Include_sound 1
	boolean Include_meaning 1
	choice Language: 1
	button All meanings language
	button Short vowel language
	button Random gap language
	button Fixed gap language
	button a-less language
	button u-less language
endform

demo.foregroundColour$ = foreground_colour$
demo.backgroundColour$ = background_colour$
demo.buttonColour$ = button_colour$
demo.font$ = font$
demo.fontSize = font_size

@sound

procedure sound
	vowels$ = "aeiou"
	numberOfVowels = length (vowels$)
	f1_erb# = { 13, 10, 7, 10, 7 }
	f2_erb# = { 19, 22, 25, 16, 13 }
	ambientStdev_erb = 1.0
	auditorySpreading_erb = 0.68
	numberOfAuditoryNodes = 60
	numberOfAuditoryNodesPerSlab = 30
	fmin_erb = 4.0
	fmax_erb = 28.0
	erbsPerNode = (fmax_erb - fmin_erb) / (numberOfAuditoryNodesPerSlab - 1)
	auditorySpreading_nodes = auditorySpreading_erb / erbsPerNode
endproc

if language$ = "Fixed gap language"
# create some sort of form within the script to let the user determine which diphthongs
to leave out.
beginPause: "Which meaning(s) do you want to remove?"
	comment: "Any pair combinations of [aeiou], space for separation e.g. 'ea eo iu
ue uo'"
	sentence: "fixed gap", "ea eo iu ue uo"
clicked = endPause: "submit", 1
writeInfoLine: "Gap(s) in this language: "
appendInfoLine: fixed_gap$
@split (" ", fixed_gap$)
numberOfRemovedMeanings = split.length
endif
```

```
# this procedure is used to split up the sentence input from the user and determine
which gaps are desired.
procedure split (sep$, str$)
        # sep$ = separator (i.e. " ")
        # str$ = string to separate
        # sepLength = separator length
        sepLength = length(sep$)
        .length = 0
        repeat
                stringLength = length(str$)
                sep = index(str$, sep$)
        if sep > 0
                part$ = left$(str$, sep-1)
                str$ = mid$(str$, sep+sepLength, stringLength)
        else
                part$ = str$
        endif
        .length = .length+1
        array$[.length] = part$
        until sep = 0
endproc

#setting up language generator
vowels1$ = "aeiou"
vowels2$ = "aeiou"
all_vowels$ = "aeiou"


#beginning language generator
procedure create_Meaning: vowels1$, vowels2$
        # k is the true index of the meaning
        k = 0
        # magic string for short vowels
        if vowels1$ = "short"
                numberOfVowels1 = length (all_vowels$)
                numberOfVowels2 = length (all_vowels$)
                for i to numberOfVowels2
                        for j to numberOfVowels1
                                if i <> j
                                        k += 1
                                        v1$ = mid$ (all_vowels$, i)
                                        v2$ = mid$ (all_vowels$, j)
                                        meaning.morpheme$ [k] = "`" + v2$ + v1$ + "’"
                                endif
                        endfor
                endfor
        # magic string for random gap language
        # l is the total number of indices of a complete language
        elif vowels1$ = "gap"
                numberOfVowels1 = length (all_vowels$)
                numberOfVowels2 = length (all_vowels$)
                l = 1
                writeInfoLine: "Gaps in this language:"
                for i to numberOfVowels2
                        for j to numberOfVowels1
                                # initializing a variable be check if meaning is to be added
                                meaningNotRemoved = 1
                                for number to size (removed#)
                                        # validity check for removal of meaning
                                        if l = removed# [number]
                                                meaningNotRemoved = meaningNotRemoved * 0
                                        else
                        meaningNotRemoved = meaningNotRemoved * 1
                    endif
                endfor
                v1$ = mid$ (all_vowels$, i)
                v2$ = mid$ (all_vowels$, j)
                if meaningNotRemoved
                    k += 1
                    meaning.morpheme$ [k] = "`" + v2$ + v1$ + "’"
                else
                    gap$ = "`" + v2$ + v1$ + "’"
                    appendInfo: gap$ + ", "
                endif
                l += 1
            endfor
        endfor
    # magic string for fixed gap based on user input language
    elif vowels1$ = "fixed gap"
        numberOfVowels1 = length (all_vowels$)
        numberOfVowels2 = length (all_vowels$)
        l = 1
        for i to numberOfVowels2
            for j to numberOfVowels1
                v1$ = mid$ (all_vowels$, i)
                v2$ = mid$ (all_vowels$, j)
                meaningNotRemoved = 1
                for removedMeaning to numberOfRemovedMeanings
                    if array$[removedMeaning] = v2$ + v1$
                        meaningNotRemoved = meaningNotRemoved * 0
                    else
                        meaningNotRemoved = meaningNotRemoved * 1
                    endif
                endfor
                    if meaningNotRemoved
                        k += 1
                        meaning.morpheme$ [k] = "`" + v2$ + v1$ + "’"
                    endif
                l += 1
            endfor
        endfor
    else
        numberOfVowels1 = length (vowels1$)
        numberOfVowels2 = length (vowels2$)
        l = 1
        for i to numberOfVowels2
            for j to numberOfVowels1
                k += 1
                v1$ = mid$ (vowels1$, i)
                v2$ = mid$ (vowels2$, j)
                meaning.morpheme$ [k] = "`" + v2$ + v1$ + "’"
            endfor
        endfor
    endif
meaning.numberOfWords = k
endproc

if language$ = "All meanings language"
# already set above.
elsif language$ = "Short vowel language"
    vowels1$ = "short"
elsif language$ = "a-less language"
    vowels1$ = "eiou"
    vowels2$ = "eiou"
elsif language$ = "u-less language"
```

```
        vowels1$ = "aeio"
        vowels2$ = "aeio"
elsif language$ = "Random gap language"
        vowels1$ = "gap"
        x = 0
        removed# = zero# (5)
        #set an array of random numbers to randomize which gaps are in the language...
        ...These are indices to be removed from a language that contains all meaning,
thus creating a new language...
        ... so here 5 random meanings are removed from all possible meanings.
        while x <= 4
                n = randomInteger(1,25)
                n_valid = 1
                #forloop guarantees different numbers in array, thus 5 removed meanings
                for number to size (removed#)
                        if n <> removed# [number]
                                n_valid = n_valid * 1
                        else
                                n_valid = n_valid * 0
                        endif
                endfor
                if n_valid
                        x += 1
                        removed# [x] = n
                endif
        endwhile
elsif language$ = "Fixed gap language"
                vowels1$ = "fixed gap"
endif
@create_Meaning: vowels1$, vowels2$
#end of language generator

numberOfInputNodes = include_sound * numberOfAuditoryNodes + include_meaning *
meaning.numberOfWords
numberOfMiddleNodes = 50
numberOfTopNodes = 20
learningRate = 0.001

semf.offsetNode = include_sound * numberOfAuditoryNodes

label NETWORK
step = 0
halfwayClickedTwoFormantsSlab1 = 0
halfwayClickedTwoFormantsSlab2 = 0
firstVowel = 1
firstMeaningVowel = 1

procedure learn: .learningRate
        @spreadUp: 1
        @hebbianLearning: .learningRate
        @resonate: 1
        @hebbianLearning: - .learningRate
endproc

procedure spreadUp: .stochastic
        activity3# = zero# (numberOfTopNodes)   ; or to random values
        .numberOfMeanFieldEchoes = 10
        for .iecho to .numberOfMeanFieldEchoes
                activity2# = sigmoid# (mul# (activity1#, weight12##) + mul# (weight23##,
activity3#) + bias2#)
                if .stochastic
                        activity2# = randomBernoulli# (activity2#)
                endif
```

```
		activity3# = sigmoid# (mul# (activity2#, weight23##) + bias3#)
		if .stochastic
			activity3# = randomBernoulli# (activity3#)
		endif
	endfor
endproc

procedure resonate: .stochastic
	.numberOfGibbsEchoes = 10
	for .iecho to .numberOfGibbsEchoes
		activity1# = mul# (weight12##, activity2#) + bias1#
		activity3# = sigmoid# (mul# (activity2#, weight23##) + bias3#)
		if .stochastic
			activity3# = randomBernoulli# (activity3#)
		endif
		activity2# = sigmoid# (mul# (activity1#, weight12##) + mul# (weight23##,
activity3#) + bias2#)
		if .stochastic
			activity2# = randomBernoulli# (activity2#)
		endif
	endfor
endproc

procedure hebbianLearning: .learningRate
	bias1# += .learningRate * activity1#
	bias2# += .learningRate * activity2#
	bias3# += .learningRate * activity3#
	weight12## += .learningRate * outer## (activity1#, activity2#)
	weight23## += .learningRate * outer## (activity2#, activity3#)
endproc

#
# Create history.
#
soundDistribution = Create Matrix: "soundDistribution",
... 0.5, numberOfAuditoryNodes + 0.5, numberOfAuditoryNodes, 1.0, 1.0, 1, 1, 1, 1, 1, ~
0.0

#
# First level.
#
activity1# = zero# (numberOfInputNodes)
bias1# = zero# (numberOfInputNodes)
x1# = linear# (0, 100, numberOfInputNodes, 1)
y1 = 42

#
# First layer.
#
weight12## = zero## (numberOfInputNodes, numberOfMiddleNodes)

#
# Second level.
#
activity2# = zero# (numberOfMiddleNodes)
bias2# = zero# (numberOfMiddleNodes)
x2# = linear# (0, 100, numberOfMiddleNodes, 1)
y2 = 66

#
# Second layer.
#
weight23## = zero## (numberOfMiddleNodes, numberOfTopNodes)
```

```
#
# Third level.
#
activity3# = zero# (numberOfTopNodes)
bias3# = zero# (numberOfTopNodes)
x3# = linear# (0, 100, numberOfTopNodes, 1)
y3 = 90

repeat
	@demo.erase
	if include_sound and include_meaning
		@demo.centredTitle: "Emergence of categories from sound-meaning pairs"
	elsif include_sound
		@demo.centredTitle: "Emergence of categories from sound alone"
	elsif include_meaning
		@demo.centredTitle: "Emergence of categories from meaning alone"
	endif
	#
	# Draw network area.
	#
	demo Select inner viewport: 20, 80, 20, 80
	demo Axes: 0, 100, 0, 100
	demo Paint rectangle: "silver", 0, 100, 0, 100
	for i to numberOfInputNodes
		for j to numberOfMiddleNodes
			weight = weight12## [i, j]
			if weight > 0
				demo Black
				demo Line width: weight
				demo Draw line: x1# [i], y1, x2# [j], y2
			elsif weight < 0
				demo White
				demo Line width: abs (weight)
				demo Draw line: x1# [i], y1, x2# [j], y2
			endif
		endfor
	endfor
	for i to numberOfMiddleNodes
		for j to numberOfTopNodes
			weight = weight23## [i, j]
			if weight > 0
				demo Black
				demo Line width: weight
				demo Draw line: x2# [i], y2, x3# [j], y3
			elsif weight < 0
				demo White
				demo Line width: abs (weight)
				demo Draw line: x2# [i], y2, x3# [j], y3
			endif
		endfor
	endfor
	demo Black
	boundaryBetweenAuditoryAndSemanticPart = semf.offsetNode / numberOfInputNodes *
100
	if include_sound
		demo Text special: 0, "left", 30, "bottom", demo.font$,
demo.fontSize/2.0, "0", "["
		demo Text special: 0.025 * boundaryBetweenAuditoryAndSemanticPart,
"centre", 30, "bottom", demo.font$, demo.fontSize/2.0, "0", "5"
		demo Text special: 0.125 * boundaryBetweenAuditoryAndSemanticPart,
"centre", 30, "bottom", demo.font$, demo.fontSize/2.0, "0", "10"
```

```
		demo Text special: 0.225 * boundaryBetweenAuditoryAndSemanticPart,
"centre", 30, "bottom", demo.font$, demo.fontSize/2.0, "0", "15"
		demo Text special: 0.325 * boundaryBetweenAuditoryAndSemanticPart,
"centre", 30, "bottom", demo.font$, demo.fontSize/2.0, "0", "20"
		demo Text special: 0.425 * boundaryBetweenAuditoryAndSemanticPart,
"centre", 30, "bottom", demo.font$, demo.fontSize/2.0, "0", "25"
		demo Text special: 0.470 * boundaryBetweenAuditoryAndSemanticPart,
"centre", 30, "bottom", demo.font$, demo.fontSize/2.0, "0", "]"
		demo Text special: 0.475 * boundaryBetweenAuditoryAndSemanticPart,
"centre", 30, "bottom", demo.font$, demo.fontSize/2.0, "0", "["
		demo Text special: 0.525 * boundaryBetweenAuditoryAndSemanticPart,
"centre", 30, "bottom", demo.font$, demo.fontSize/2.0, "0", "5"
		demo Text special: 0.625 * boundaryBetweenAuditoryAndSemanticPart,
"centre", 30, "bottom", demo.font$, demo.fontSize/2.0, "0", "10"
		demo Text special: 0.725 * boundaryBetweenAuditoryAndSemanticPart,
"centre", 30, "bottom", demo.font$, demo.fontSize/2.0, "0", "15"
		demo Text special: 0.825 * boundaryBetweenAuditoryAndSemanticPart,
"centre", 30, "bottom", demo.font$, demo.fontSize/2.0, "0", "20"
		demo Text special: 0.915 * boundaryBetweenAuditoryAndSemanticPart,
"centre", 30, "bottom", demo.font$, demo.fontSize/2.0, "0", "25"
		demo Text special: boundaryBetweenAuditoryAndSemanticPart,  "right", 30,
"bottom", demo.font$, demo.fontSize/2.0, "0", "ERB]"
	endif
	if include_meaning
		x# = linear# (boundaryBetweenAuditoryAndSemanticPart, 100,
meaning.numberOfWords, 1)
		for i to meaning.numberOfWords
			demo Text special: x# [i], "right", 37, "half", demo.font$,
demo.fontSize/3.0, "90", meaning.morpheme$ [i]
		endfor
	endif
	demo Line width: 2
	radius = 1.5
	for i to numberOfInputNodes
		input = activity1# [i] / 5
		if input <> 0
			demo Paint circle: if input > 0 then "red" else "blue" fi, x1# [i],
y1, radius * abs (input)
		endif
		demo Draw circle: x1# [i], y1, radius
	endfor
	radius = 1.0
	for i to numberOfMiddleNodes
		demo Paint circle: "red", x2# [i], y2, radius * activity2# [i] + 1e-6
		demo Draw circle: x2# [i], y2, radius
	endfor
	radius = 2.0
	for i to numberOfTopNodes
		demo Paint circle: "red", x3# [i], y3, radius * activity3# [i] + 1e-6
		demo Draw circle: x3# [i], y3, radius
	endfor
	#
	# Draw history.
	#
	selectObject: soundDistribution
	demo Yellow
	demo Line width: 3
	demo Draw rows: 0.5, numberOfInputNodes + 0.5, 0, 0, 0, step * 5
	demo Colour: demo.foregroundColour$
	#
	# Draw buttons.
	#
	demo Line width: 2
	demo Select inner viewport: 0, 100, 0, 100
	demo Axes: 0, 100, 0, 100
	demo Text: 50, "centre", 12, "half", "After " + string$ (step) + if step = 1
then " input." else " inputs." fi
	y1NDC = 20 + 60/100 * y1
	y2NDC = 20 + 60/100 * y2
	y3NDC = 20 + 60/100 * y3
	@demo.button: 14, 18, y1NDC, "1"
	@demo.button: 14, 18, y2NDC, "2"
	@demo.button: 14, 18, y3NDC, "3"
	if include_sound and include_meaning
		@demo.button: 8, 13, y1NDC, "[s]"
		@demo.button: 81, 85, y1NDC, "‘m’"
	endif
	demo Select inner viewport: 0, 100, 0, 100
	@demo.button: 88, 98, 70, "10000↑"
	@demo.button: 88, 98, 60, "1000↑"
	@demo.button: 88, 98, 50, "100↑"
	@demo.button: 88, 98, 40, "10↑"
	@demo.button: 88, 98, 30, "1↑"
	@demo.button: 88, 98, 20, "new"
	#
	# Draw manual.
	#
	y = 0
	if include_sound and include_meaning
		demo Text special: 0, "left", y, "bottom", demo.font$, demo.fontSize/3.0,
"0", "To spread nonstochastically to a level, click 1/2/3, or to spread to sound or
meaning only, click [s] or ‘m’."
		y += 2
	endif
	if include_meaning
		demo Text special: 0, "left", y, "bottom", demo.font$, demo.fontSize/3.0,
"0", "To input meaning, click on one morpheme node, or (to get the composed meaning)
type a/e/i/o/u."
		y += 2
	endif
	if include_sound
		demo Text special: 0, "left", y, "bottom", demo.font$, demo.fontSize/3.0,
"0", "To input a vowel sound, click on two formant nodes, or (to get the category
centre) type A/E/I/O/U."
	endif
	#
	# User interaction loop.
	#
	while demoWaitForInput ( )
		if demoClickedIn (88, 98, 20-4, 20+4) or demoInput ("n")   ; new
			removeObject: soundDistribution
			goto NETWORK
		elsif demoClickedIn (20, 80, 20, 50)
			demo Select inner viewport: 20, 80, 20, 80
			clickedInputNode = 0.5 + demoX ( ) / 100 * numberOfInputNodes
			clickedInAuditoryPart = ( clickedInputNode <= semf.offsetNode + 0.5
)
			if clickedInAuditoryPart
				#Determine in which slab is clicked.
				clickedInSlab1 = ( clickedInputNode <=
numberOfAuditoryNodesPerSlab + 0.5)
				if clickedInSlab1
					if halfwayClickedTwoFormantsSlab1
						clickedAuditoryNode = clickedInputNode
						activity1# ~ if col <=
numberOfAuditoryNodesPerSlab
```

```
                                        ... then self + 5 * exp (-0.5 * ((col - 
clickedAuditoryNode) / auditorySpreading_nodes) ^ 2) - 0.5
                                        ... else if col <= numberOfAuditoryNodes
                                                ... then self
                                                ... else 0
                                                ... fi
                                        ... fi
                                        clickedFormant2_erb_Slab1 = fmin_erb + 
(clickedAuditoryNode - 1) * erbsPerNode
                                        clickedFormant2_erb_Slab2 = fmin_erb + 
(clickedAuditoryNode - 1) * erbsPerNode
                                        @speak: clickedFormant1_erb_Slab1, 
clickedFormant2_erb_Slab1, clickedFormant1_erb_Slab2, clickedFormant2_erb_Slab2
                                        halfwayClickedTwoFormantsSlab1 = 0
                                else
                                        clickedAuditoryNode = clickedInputNode
                                        activity1# ~ if col <= 
numberOfAuditoryNodesPerSlab
                                        ... then 5 * exp (-0.5 * ((col - 
clickedAuditoryNode) / auditorySpreading_nodes) ^ 2) - 0.5
                                        ... else if col <= numberOfAuditoryNodes
                                                ... then self
                                                ... else 0
                                                ... fi
                                        ... fi
                                        clickedFormant1_erb_Slab1 = fmin_erb + 
(clickedAuditoryNode - 1) * erbsPerNode
                                        clickedFormant1_erb_Slab2 = fmin_erb + 
(clickedAuditoryNode - 1) * erbsPerNode
                                        halfwayClickedTwoFormantsSlab1 = 1
                                endif
                        else
                                if halfwayClickedTwoFormantsSlab2
                                        clickedAuditoryNode = clickedInputNode
                                        activity1# ~ if col <= 
numberOfAuditoryNodesPerSlab
                                        ... then self
                                        ... else if col <= numberOfAuditoryNodes
                                                ... then self + 5 * exp (-0.5 * ((col - 
clickedAuditoryNode) / auditorySpreading_nodes) ^ 2) - 0.5
                                                ... else 0
                                                ... fi
                                        ... fi
                                        clickedFormant2_erb_Slab2 = fmin_erb + 
(clickedAuditoryNode - numberOfAuditoryNodesPerSlab - 1) * erbsPerNode
                                        @speak: clickedFormant1_erb_Slab1, 
clickedFormant2_erb_Slab1, clickedFormant1_erb_Slab2, clickedFormant2_erb_Slab2
                                        halfwayClickedTwoFormantsSlab2 = 0
                                else
                                        clickedAuditoryNode = clickedInputNode
                                        activity1# ~ if col <= 
numberOfAuditoryNodesPerSlab
                                        ... then self
                                        ... else if col <= numberOfAuditoryNodes
                                                ... then 5 * exp (-0.5 * ((col - 
clickedAuditoryNode) / auditorySpreading_nodes) ^ 2) - 0.5
                                                ... else 0
                                                ... fi
                                        ... fi
                                        clickedFormant1_erb_Slab2 = fmin_erb + 
(clickedAuditoryNode - numberOfAuditoryNodesPerSlab - 1) * erbsPerNode
                                        halfwayClickedTwoFormantsSlab2 = 1
                                endif
                        endif
                else
                        clickedMeaningNode = round (clickedInputNode -
semf.offsetNode)
                        activity1# ~ if col <= semf.offsetNode
                        ... then 0
                        ... else 5 * if col - semf.offsetNode = clickedMeaningNode
then 1.0 else - 1 / (meaning.numberOfWords - 1) fi
                        ... fi
                endif
                demo Select inner viewport: 0, 100, 0, 100
                activity2# = zero# (numberOfMiddleNodes)
                activity3# = zero# (numberOfTopNodes)
                goto NETWORK_NEXT
        elsif demoClickedIn (14, 18, y1NDC-4, y1NDC+4) or demoInput ("1")
                activity1# = mul# (weight12##, activity2#) + bias1#
                goto NETWORK_NEXT
        elsif demoClickedIn (14, 18, y2NDC-4, y2NDC+4) or demoInput ("2")
                activity2# = sigmoid# (mul# (activity1#, weight12##) + mul#
(weight23##, activity3#) + bias2#)
                goto NETWORK_NEXT
        elsif demoClickedIn (14, 18, y3NDC-4, y3NDC+4) or demoInput ("3")
                activity3# = sigmoid# (mul# (activity2#, weight23##) + bias3#)
                goto NETWORK_NEXT
        elsif include_meaning and demoClickedIn (8, 13, y1NDC-4, y1NDC+4) or
demoInput ("s")
                activity1_wide# = mul# (weight12##, activity2#) + bias1#
                activity1# ~ if col <= semf.offsetNode then activity1_wide# [col]
else self fi
                goto NETWORK_NEXT
        elsif include_meaning and demoClickedIn (81, 85, y1NDC-4, y1NDC+4) or
demoInput ("m")
                activity1_wide# = mul# (weight12##, activity2#) + bias1#
                activity1# ~ if col <= semf.offsetNode then self else
activity1_wide# [col] fi
                goto NETWORK_NEXT
        elsif demoInput ("AEIOU")
                if include_sound
                        if firstVowel
                                clickedVowel1 = index ("AEIOU", demoKey$ ())
                                f1_erb_slab1 = randomGauss (f1_erb# [clickedVowel1],
ambientStdev_erb)
                                f2_erb_slab1 = randomGauss (f2_erb# [clickedVowel1],
ambientStdev_erb)
                                @cleanInput
                                firstVowel = 0
                        else
                                clickedVowel2 = index ("AEIOU", demoKey$ ())
                                f1_erb_slab2 = randomGauss (f1_erb# [clickedVowel2],
ambientStdev_erb)
                                f2_erb_slab2 = randomGauss (f2_erb# [clickedVowel2],
ambientStdev_erb)
                                @cleanInput
                                @applySound: f1_erb_slab1, f2_erb_slab1,
f1_erb_slab2, f2_erb_slab2, 0
                                @spreadUp: 0
                                ;@resonate: 0
                                @speak: f1_erb_slab1, f2_erb_slab1, f1_erb_slab2,
f2_erb_slab2
                                firstVowel = 1
                        endif
                endif
                goto NETWORK_NEXT
```

```
		elsif demoInput ("aeiou")
			if include_sound
				if firstMeaningVowel
					clickedMeaningVowel1$ = demoKey$ ()
					firstMeaningVowel = 0
				else
					clickedMeaningVowel2$ = demoKey$ ()
					firstMeaningVowel = 1
					clickedMeaning$ = "‘" + clickedMeaningVowel1$ +
clickedMeaningVowel2$ + "’"
					for i to meaning.numberOfWords
						if meaning.morpheme$ [i] = clickedMeaning$
							clickedWord = i
							@cleanInput
							@applyMeaning: clickedWord
							@spreadUp: 0
							;@resonate: 0
						endif
					endfor
				endif
			endif
			goto NETWORK_NEXT
			# Similarity between sounds
		elsif demoInput ("S")
			similarity## = zero## (meaning.numberOfWords,
meaning.numberOfWords)
			for iword to meaning.numberOfWords
				ivowel_slab1$ = mid$( meaning.morpheme$ [iword], 2)
				ivowel_slab1 = index (vowels$, ivowel_slab1$)
				ivowel_slab2$ = mid$( meaning.morpheme$ [iword], 3)
				ivowel_slab2 = index (vowels$, ivowel_slab2$)
				for jword to meaning.numberOfWords
					jvowel_slab1$ = mid$( meaning.morpheme$ [jword], 2)
					jvowel_slab1 = index (vowels$, jvowel_slab1$)
					jvowel_slab2$ = mid$( meaning.morpheme$ [jword], 3)
					jvowel_slab2 = index (vowels$, jvowel_slab2$)
					@cleanInput
					@applySound: f1_erb# [ivowel_slab1], f2_erb#
[ivowel_slab1], f1_erb# [ivowel_slab2], f2_erb# [ivowel_slab2], 0
					@spreadUp: 0
				@resonate: 0
					activity_ivowel# = activity2#
					@cleanInput
					@applySound: f1_erb# [jvowel_slab1], f2_erb#
[jvowel_slab1], f1_erb# [jvowel_slab2], f2_erb# [jvowel_slab2], 0
					@spreadUp: 0
				@resonate: 0
					activity_jvowel# = activity2#
					similarity = inner (activity_ivowel#,
activity_jvowel#) / norm (activity_ivowel#) / norm (activity_jvowel#)
					similarity## [iword, jword] = round (similarity *
100)
				endfor
			endfor
			writeInfoLine: "Similarity of sounds: ", newline$, similarity##
			# Similarity between meaning morphemes
		elsif demoInput ("M")
			similarity## = zero## (meaning.numberOfWords,
meaning.numberOfWords)
			for iword to meaning.numberOfWords
				for jword to meaning.numberOfWords
					@cleanInput
					@applyMeaning: iword
```

```
                              @spreadUp: 0
                        @resonate: 0
                              activity_ivowel# = activity2#
                              @cleanInput
                              @applyMeaning: jword
                              @spreadUp: 0
                        @resonate: 0
                              activity_jvowel# = activity2#
                              similarity = inner (activity_ivowel#,
activity_jvowel#) / norm (activity_ivowel#) / norm (activity_jvowel#)
                              similarity## [iword, jword] = round (similarity *
100)
                        endfor
                  endfor
                  writeInfoLine: "Similarity of morphemes: ", newline$, similarity##
                  # Find activity of meaning nodes
            elsif demoInput ("N")
                  meaning_activity# = zero# (meaning.numberOfWords)
                  for meaning_activity from numberOfAuditoryNodes to k
                        @cleanInput
                        @applyMeaning: meaning_activity
                        @spreadUp: 0
                  @resonate: 0
                        activity_ivowel# = activity1#
                  endfor
                  writeInfoLine: "Activity of meaning nodes", newline$
                  for activity from (numberOfAuditoryNodes + 1) to size (activity1#)
                    appendInfo: fixed$ (activity1# [activity], 3) + ", "
                  endfor
                  # Draw the network in Praat Picture
            elsif demoInput ("D")
                  Select inner viewport: 0.5, 11, 0.4, 6
                  Axes: 0, 100, 0, 100
                  Paint rectangle: "silver", 0, 100, 0, 100
                  for i to numberOfInputNodes
                        for j to numberOfMiddleNodes
                              weight = weight12## [i, j]
                              if weight > 0
                                    Black
                                    Line width: weight
                                    Draw line: x1# [i], y1, x2# [j], y2
                              elsif weight < 0
                                    White
                                    Line width: abs (weight)
                                    Draw line: x1# [i], y1, x2# [j], y2
                              endif
                        endfor
                  endfor
                  for i to numberOfMiddleNodes
                        for j to numberOfTopNodes
                              weight = weight23## [i, j]
                              if weight > 0
                                    Black
                                    Line width: weight
                                    Draw line: x2# [i], y2, x3# [j], y3
                              elsif weight < 0
                                    White
                                    Line width: abs (weight)
                                    Draw line: x2# [i], y2, x3# [j], y3
                              endif
                        endfor
                  endfor
                  Black
```

```
                    boundaryBetweenAuditoryAndSemanticPart = semf.offsetNode /
numberOfInputNodes * 100
                    if include_sound
                           Text special: 0, "left", 30, "bottom", demo.font$,
demo.fontSize/2.0, "0", "["
                           Text special: 0.025 *
boundaryBetweenAuditoryAndSemanticPart, "centre", 30, "bottom", demo.font$,
demo.fontSize/2.0, "0", "5"
                           Text special: 0.125 *
boundaryBetweenAuditoryAndSemanticPart, "centre", 30, "bottom", demo.font$,
demo.fontSize/2.0, "0", "10"
                           Text special: 0.225 *
boundaryBetweenAuditoryAndSemanticPart, "centre", 30, "bottom", demo.font$,
demo.fontSize/2.0, "0", "15"
                           Text special: 0.325 *
boundaryBetweenAuditoryAndSemanticPart, "centre", 30, "bottom", demo.font$,
demo.fontSize/2.0, "0", "20"
                           Text special: 0.425 *
boundaryBetweenAuditoryAndSemanticPart, "centre", 30, "bottom", demo.font$,
demo.fontSize/2.0, "0", "25"
                           Text special: 0.470 *
boundaryBetweenAuditoryAndSemanticPart, "centre", 30, "bottom", demo.font$,
demo.fontSize/2.0, "0", "]"
                           Text special: 0.475 *
boundaryBetweenAuditoryAndSemanticPart, "centre", 30, "bottom", demo.font$,
demo.fontSize/2.0, "0", "["
                           Text special: 0.525 *
boundaryBetweenAuditoryAndSemanticPart, "centre", 30, "bottom", demo.font$,
demo.fontSize/2.0, "0", "5"
                           Text special: 0.625 *
boundaryBetweenAuditoryAndSemanticPart, "centre", 30, "bottom", demo.font$,
demo.fontSize/2.0, "0", "10"
                           Text special: 0.725 *
boundaryBetweenAuditoryAndSemanticPart, "centre", 30, "bottom", demo.font$,
demo.fontSize/2.0, "0", "15"
                           Text special: 0.825 *
boundaryBetweenAuditoryAndSemanticPart, "centre", 30, "bottom", demo.font$,
demo.fontSize/2.0, "0", "20"
                           Text special: 0.915 *
boundaryBetweenAuditoryAndSemanticPart, "centre", 30, "bottom", demo.font$,
demo.fontSize/2.0, "0", "25"
                           Text special: boundaryBetweenAuditoryAndSemanticPart,
"right", 30, "bottom", demo.font$, demo.fontSize/2.0, "0", "ERB]"
                    endif
                    if include_meaning
                           x# = linear# (boundaryBetweenAuditoryAndSemanticPart, 100,
meaning.numberOfWords, 1)
                           for i to meaning.numberOfWords
                                  Text special: x# [i], "right", 37, "half",
demo.font$, demo.fontSize/3.0, "90", meaning.morpheme$ [i]
                           endfor
                    endif
                    Line width: 2
                    radius = 1.5
                    for i to numberOfInputNodes
                           input = activity1# [i] / 5
                           if input <> 0
                                  Paint circle: if input > 0 then "black" else "grey"
fi, x1# [i], y1, radius * abs (input)
                           endif
                           Draw circle: x1# [i], y1, radius
                    endfor
                    radius = 1.0
                    for i to numberOfMiddleNodes
                          Paint circle: "black", x2# [i], y2, radius * activity2# [i]
+ 1e-6
                          Draw circle: x2# [i], y2, radius
                    endfor
                    radius = 2.0
                    for i to numberOfTopNodes
                          Paint circle: "black", x3# [i], y3, radius * activity3# [i]
+ 1e-6
                          Draw circle: x3# [i], y3, radius
                    endfor
              elsif demoInput ("F")

              endif
              numberOfSteps =
              ... if demoClickedIn (88, 98, 30-4, 30+4) or demoInput ("↑") then 1 else
              ... if demoClickedIn (88, 98, 40-4, 40+4) then 10 else
              ... if demoClickedIn (88, 98, 50-4, 50+4) then 100 else
              ... if demoClickedIn (88, 98, 60-4, 60+4) then 1000 else
              ... if demoClickedIn (88, 98, 70-4, 70+4) then 10000 else 0 fi fi fi fi
fi
              if numberOfSteps <> 0
                    for ministep to abs (numberOfSteps)
                          step += 1
                          word = randomInteger (1, meaning.numberOfWords)
                          vowel_slab1$ = mid$( meaning.morpheme$ [word], 2)
                          vowel_slab1 = index (vowels$, vowel_slab1$)
                          vowel_slab2$ = mid$( meaning.morpheme$ [word], 3)
                          vowel_slab2 = index (vowels$, vowel_slab2$)
                          f1_slab1 = randomGauss (f1_erb# [vowel_slab1],
ambientStdev_erb)
                          f2_slab1 = randomGauss (f2_erb# [vowel_slab1],
ambientStdev_erb)
                          f1_slab2 = randomGauss (f1_erb# [vowel_slab2],
ambientStdev_erb)
                          f2_slab2 = randomGauss (f2_erb# [vowel_slab2],
ambientStdev_erb)
                          @cleanInput
                          if include_sound
                                @applySound: f1_slab1, f2_slab1, f1_slab2, f2_slab2,
1
                          endif
                          if include_meaning
                                @applyMeaning: word
                          endif
                          @learn: learningRate
                          if numberOfSteps = 1
                                @speak: f1_slab1, f2_slab1, f1_slab2, f2_slab2
                          endif
                    endfor
                    goto NETWORK_NEXT
              endif
              goto NETWORK_END demoInput ("← →")
       endwhile
       label NETWORK_NEXT
until 0
label NETWORK_END
#
# Clean up history.
#
removeObject: soundDistribution

procedure cleanInput
```

```
	activity1# ~ 0.0
endproc

procedure applySound: .f1_erb_slab1, .f2_erb_slab1, .f1_erb_slab2, .f2_erb_slab2,
.recordSoundDistribution
	.audNode1_slab1 = 1 + (.f1_erb_slab1 - fmin_erb) / erbsPerNode
	.audNode2_slab1 = 1 + (.f2_erb_slab1 - fmin_erb) / erbsPerNode
	.audNode1_slab2 = 1 + numberOfAuditoryNodesPerSlab + (.f1_erb_slab2 - fmin_erb)
/ erbsPerNode
	.audNode2_slab2 = 1 + numberOfAuditoryNodesPerSlab + (.f2_erb_slab2 - fmin_erb)
/ erbsPerNode

	activity1# ~ if col > numberOfAuditoryNodes then self else
	... 5 * exp (-0.5 * ((col - .audNode1_slab1) / auditorySpreading_nodes) ^ 2) +
	... 5 * exp (-0.5 * ((col - .audNode2_slab1) / auditorySpreading_nodes) ^ 2) - 1
+
	... 5 * exp (-0.5 * ((col - .audNode1_slab2) / auditorySpreading_nodes) ^ 2) +
	... 5 * exp (-0.5 * ((col - .audNode2_slab2) / auditorySpreading_nodes) ^ 2) - 1
	... fi
	if .recordSoundDistribution
		select soundDistribution
		Formula: ~ self + activity1# [col] + 2
	endif
endproc

procedure applyMeaning: .word
	activity1# ~ if col <= semf.offsetNode then self else -5 /
(meaning.numberOfWords - 1) fi
	activity1# [semf.offsetNode + .word] = 5
endproc

procedure speak: .f1_erb_slab1, .f2_erb_slab1, .f1_erb_slab2, .f2_erb_slab2
	.f1_Slab1 = erbToHertz (.f1_erb_slab1)
	.f2_Slab1 = erbToHertz (.f2_erb_slab1)
	.f1_Slab2 = erbToHertz (.f1_erb_slab2)
	.f2_Slab2 = erbToHertz (.f2_erb_slab2)
	runScript: "makeDiphthong.praat", .f1_Slab1, .f2_Slab1, .f2_Slab1 + 1000,
.f2_Slab1 + 1900,
	... .f1_Slab2, .f2_Slab2, .f2_Slab2 + 1000, .f2_Slab2 + 1900, 80, 160, 360, 530,
80, 160, 360, 530, 150, 0.5
	asynchronous Play
	plusObject: "KlattGrid kg"
	Remove
endproc

include demo.praatinclude
```

Appendix 2: Diphthong script

```
# Klatt Grid for basic vowel.
# Karin Wanrooij & Paul Boersma, 28 October 2010.
# Edited by Angelica van Beemdelust for basic diphthongs.

# 1. PARAMETERS

form Fill in:
       comment in Hertz:
       real F1_Slab1 400
       real F2_Slab1 1400
       real F3_Slab1 2400
       real F4_Slab1 3400
       real F1_Slab2 400
       real F2_Slab2 1400
       real F3_Slab2 2400
       real F4_Slab2 3400
#bandwidth of formants
       real B1_Slab1 80
       real B2_Slab1 160
       real B3_Slab1 360
       real B4_Slab1 530
       real B1_Slab2 80
       real B2_Slab2 160
       real B3_Slab2 360
       real B4_Slab2 530
       real startF0 150
       comment in seconds:
       real Duration 0.5
endform

startVowel = 0

# FOR BASIC KLATT GRID
       nrOfFormants = 10
       nrOfNasalFormants = 0
       nrOfNasalAntiFormants = 0
       # The frication and delta formants below are not used.
       nrOfFricationFormants = 5
       nrOfTrachealFormants = 0
       nrOfTrachealAntiFormants = 0
       nrOfDeltaFormants = 1

# FOR PHONATION
       minVoiceAmpl = 35
#      midVoiceAmpl =
       maxVoiceAmpl = 40

#random variation of the pitch
       flutter = 0.15
#glottal flow
       power1 = 3
       power2 = 4
#open phase of the glottis
       openPhase = 0.7
#collision phase, models last part of flow function with exp. decay?
```

```
	colPhase = 0.04
#not sure what this is
	spectralTilt = 10
	# doublePulsing =
	# aspirationAmpl =
	# breathinessAmpl =

# FOR VOCAL TRACT + DURATION
# See form above

# Add extra formants to get a flatter spectrum:
# NB: the f5 value is based on a male voice.
#Slab1
f5_Slab1 = f4_Slab1 + 650
f6_Slab1 = f5_Slab1 + 1000
f7_Slab1 = f6_Slab1 + 1000
f8_Slab1 = f7_Slab1 + 1000
f9_Slab1 = f8_Slab1 + 1000
f10_Slab1 = f9_Slab1 + 1000

#Slab2
f5_Slab2 = f4_Slab2 + 650
f6_Slab2 = f5_Slab2 + 1000
f7_Slab2 = f6_Slab2 + 1000
f8_Slab2 = f7_Slab2 + 1000
f9_Slab2 = f8_Slab2 + 1000
f10_Slab2 = f9_Slab2 + 1000

# The bandwidth values are based on a male voice.

upBandwidth = 8.5
for i from 5 to nrOfFormants
	b'i'_Slab1 = f'i'_Slab1/upBandwidth
	b'i'_Slab2 = f'i'_Slab2/upBandwidth
endfor

# 2. CREATE BASIC KLATT GRID.
	kg = Create KlattGrid... kg startVowel duration nrOfFormants nrOfNasalFormants
nrOfNasalAntiFormants nrOfFricationFormants
	... nrOfTrachealFormants nrOfTrachealAntiFormants  nrOfDeltaFormants

	Add oral formant frequency point: 1, duration * 0.3, f1_Slab1
	Add oral formant frequency point: 2, duration * 0.3, f2_Slab1
	Add oral formant frequency point: 1, duration * 0.7, f1_Slab2
	Add oral formant frequency point: 2, duration * 0.7, f2_Slab2

# MODIFY PHONATION.
#	The pitch declines linearly. For other pitch contours: adapt the 'Add pitch
point':
	Add pitch point... startVowel startF0
	Add pitch point... duration ((startF0)*0.75)

#	The voicing amplitude reaches a maximum at 40% of the duration. For other
amplitude contours: adapt the 'Add voicing amplitude point':
	Add voicing amplitude point... startVowel minVoiceAmpl
	Add voicing amplitude point... (duration*0.4) maxVoiceAmpl
	Add voicing amplitude point... duration minVoiceAmpl

	Add flutter point... startVowel flutter
	Add power1 point... startVowel power1
	Add power2 point... startVowel power2
	Add open phase point... startVowel openPhase
	Add collision phase point... startVowel colPhase
	;Add spectral tilt point... startVowel spectralTilt

#	Add double pulsing point... startVowel doublePulsing
#	Add aspiration amplitude point... startVowel aspirationAmpl
#	Add breathiness amplitude point... startVowel breathinessAmpl

# MODIFY VOCAL TRACT.
#	FORMANTS 1 - 10 AND BANDWIDTHS 1 - 10:
	for i to nrOfFormants
		Add oral formant frequency point... i startVowel f'i'_Slab1
		Add oral formant bandwidth point... i startVowel b'i'_Slab1
		Add oral formant frequency point... i duration f'i'_Slab2
		Add oral formant bandwidth point... i duration b'i'_Slab2
	endfor

	To Sound
	Rename... vowel
	Fade in... All 0 0.005 n
	Fade out... All duration -0.005 n
	Scale intensity... 70
```

Appendix 3: Demo script

```
# Praat include file demo.praatinclude
# Paul Boersma, 4 April 2014

procedure demo.erase
	demo 'demo.font$'
	demo Font size... demo.fontSize
	demo Select inner viewport... 0 100 0 100
	demo Axes... 0 100 0 100
	demo Erase all
	demo Paint rectangle... 'demo.backgroundColour$' 0 100 0 100
	demo Colour... 'demo.foregroundColour$'
endproc

procedure demo.title .text$
	.width = demo Text width (wc)... '.text$'
	if .width < 45
		demo Text special... 7 left 90 half 'demo.font$' 2*demo.fontSize 0
'.text$'
	else
		demo Text special... 50 centre 90 half 'demo.font$'
2*demo.fontSize*45/.width 0 '.text$'
	endif
	demo.textY = 70
endproc

procedure demo.centredTitle .text$
	.width = demo Text width (wc)... '.text$'
	if .width < 45
		demo Text special... 50 centre 90 half 'demo.font$' 2*demo.fontSize 0
'.text$'
	else
		demo Text special... 50 centre 90 half 'demo.font$'
2*demo.fontSize*45/.width 0 '.text$'
	endif
	demo.textY = 70
endproc

procedure demo.bullet .text$
	demo Text... 10-1.5 centre demo.textY-0.5 half •
	.width = demo Text width (wc)... '.text$'
	if .width < 85
		demo Text... 10 left demo.textY half '.text$'
	else
		demo Text special... 10 left demo.textY half 'demo.font$'
demo.fontSize*85/.width 0 '.text$'
	endif
	demo.textY -= 12
endproc

procedure demo.therefore .text$
	demo Text... 10-2.5 centre demo.textY half ∴
	.width = demo Text width (wc)... '.text$'
	if .width < 85
		demo Text... 10 left demo.textY half '.text$'
	else
```

```
              demo Text special... 10 left demo.textY half 'demo.font$'
demo.fontSize*85/.width 0 '.text$'
       endif
       demo.textY -= 12
endproc

procedure demo.text .text$
       demo.textY += 4
       .width = demo Text width (wc)... '.text$'
       if .width < 85
              demo Text... 10 left demo.textY half '.text$'
       else
              demo Text special... 10 left demo.textY half 'demo.font$'
demo.fontSize*85/.width 0 '.text$'
       endif
       demo.textY -= 12
endproc

procedure demo.reference .text$
       demo.textY += 5
       demo Text special... 98 right demo.textY half 'demo.font$' demo.fontSize/1.5 0
'.text$'
       demo.textY -= 9
endproc

procedure demo.source .text$
       demo Text special... 2 left 2 bottom Times demo.fontSize/1.5 0 '.text$'
endproc

procedure demo.button .x1 .x2 .y .text$
       demo Paint rounded rectangle... 'demo.buttonColour$' .x1 .x2 .y-4 .y+4 3
       .width = demo Text width (wc)... '.text$'
       if .width < 0.9 * (.x2 - .x1)
              demo Text... (.x1+.x2)/2 centre .y half '.text$'
       else
              demo Text special... (.x1+.x2)/2 centre .y half 'demo.font$'
demo.fontSize*0.9*(.x2-.x1)/.width 0 '.text$'
       endif
endproc

procedure demo.wait .duration
       Create Sound from formula... silence mono 0 .duration 44100 0
       Play
       Remove
endproc
```