Emergence of features and phonemes in a neural network with a single input continuum

L. M. H. Boekel

10268200

Research Master in Brain and Cognitive Sciences

August 2013

Supervisor: Prof. Dr. Paul Boersma

Co-assessor: Dr. Silke Hamann

Emergence of features and phonemes in a neural network with a single input continuum

The 'feature versus phoneme' debate has been the basis for research in the fields of phonetics and phonology for many years. The research described here used a neural network modelling approach to examine this issue. Several types of models were simulated to investigate whether phonemes or features emerge in a neural network with a single input continuum. It was found that under different conditions, either features or phonemes could emerge in a neural network. As the neural network modelling approach is still new to phonetics and phonology, future research using this approach may shed more light on how the brain handles phonetic and phonological processes.

1. Introduction

In the fields of phonetics and phonology, the 'features versus phoneme' debate has been the basis for research for many years. The central question in this debate is the following: Are vowels represented in the language user as phonemes or as feature bundles? Vowels can be seen as sharing some features, such as horizontal position of the tongue (the vowels /o/ and /u/ are both 'back' vowels, whereas /i/ and /e/ are 'front' vowels) and vertical position of the tongue (/i/ and /u/ are 'high' vowels, whereas /a/ is a 'low' vowel), and previous research, indeed, seems to favour the vowels-as-features side of the debate (e.g. Boersma & Chládková 2011; Recasens & Espinosa 2009).

Boersma and Chládková (2013) investigated the issue using a neural network model (NNM). Neural models have been around since the 1940's (McCulloch & Pitts 1943), and have been used to model various cognitive processes in the brain. However, researchers in linguistics have only recently begun to make use of these types of models. Guenther & Gjaja (1996), for example, showed how the perceptual magnet effect (a warping of the auditory space) arises in a two-layered neural net. More recently, Boersma, Benders and Seinhorst (2013) have shown that both phonological category creation and auditory dispersion can be modelled successfully in an artificial neural network. These types of models are more biologically plausible than, for instance, Optimality Theoretic models, and are thus preferred.

Boersma and Chládková (2013) designed a NNM to investigate whether, if learners are only presented with sound-meaning pairs, features or phonemes would emerge. Their model consisted of three layers: an auditory layer at the bottom, split into two separate formant value continua; a phonological layer in the middle, also split into two, and which is connected to the layers above and below it; and a word layer at the top. Boersma and Chládková (2013) fed the model combinations of inputs at the word level and at the auditory level, after which the activity was allowed to spread to the phonological level and the connection weights were updated using a particular learning algorithm. The researchers found that vowel features emerge at the phonological level. However, even though this model simulates the emergence of vowel features successfully, it needs two separate input continua for the first and second formant values, respectively. In reality, formant values are represented on a single input continuum: the basilar membrane. It is therefore desirable to design a neural network which employs just one input continuum on which both formant values are represented.

In this paper, I will describe research that was set out to adapt the NNM designed by Boersma and Chládková (2013) in order to make it more biologically plausible. I designed a neural network model with a single input continuum on which multiple formant values can be represented, I trained this model on sound-meaning pairs, and I investigated whether, just as in Boersma and Chládková (2013), features would emerge. In the remainder of this paper, I will describe the basic architecture of this model, parameters of the model, and the learning phase, and I will report on and discuss the results of the simulations I ran with different types of the model.

2. Methods

2.1 The architecture of the model

The basic architecture of the models described in this paper is shown in Fig.1. All models were simulated in Praat (Boersma & Weenink, 2005; scripts can be found in Appendices A through D). All models consist of three layers of nodes: a 'meaning' layer, a middle (phonological) layer, and a sound layer. Both the meaning layer and the middle layer contain 20 nodes. In the meaning layer, these nodes are grouped into five groups of four nodes, which represent the five possible words in the simplified language used in these experiments (i.e. "I", "E", "A", "O", and "U"). The nodes in the middle layer are not grouped in any way.

The sound layer contains 60 nodes. This layer represents a frequency continuum from 100 Hz to 3000 Hz. All words in our toy language correspond to a certain pattern of two values on the auditory layer: the F1 value and the F2 value (see e.g., Fig. 2). The F1 continuum ranges from 100 Hz to 1000 Hz, and the F2 continuum ranges from 500 Hz to 3000 Hz. Table 1 shows the mean values of the F1 and F2 of all possible words, in percentage of the respective continua, in frequency in Hz and in the absolute node number on the sound layer.

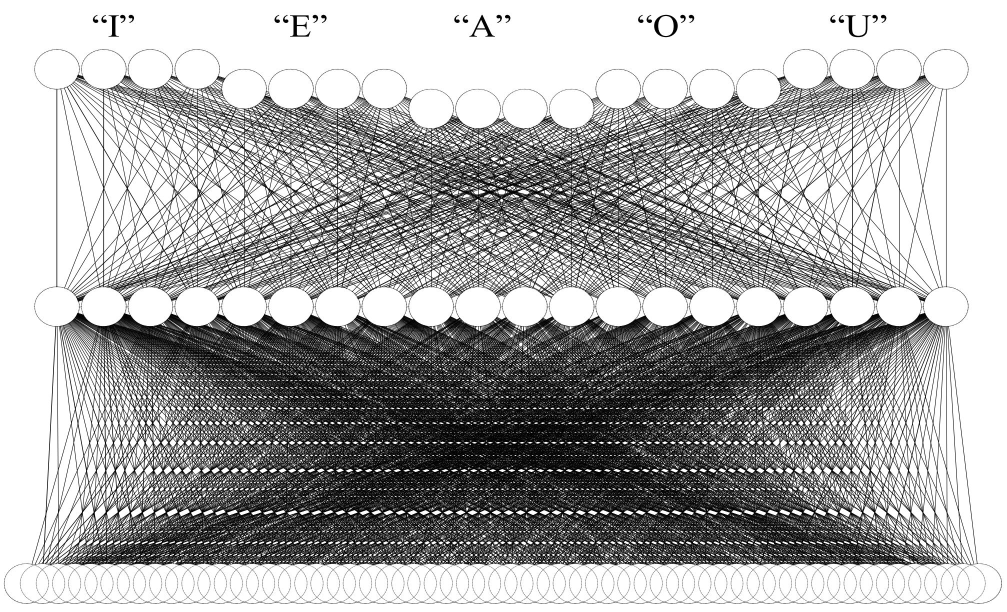


Figure 1. The architecture of the models.

All nodes in the meaning layer are connected to all nodes in the middle layer through excitatory connections. The same is true for the nodes in the sound layer and the middle layer. Initially, the weights of these connections are set to random values between 0 and 0.1. In the learning stage, these connections are allowed to learn and their weights will be updated as a result of the specific input the model receives.

In the models described in this paper, inhibitory connections exist between all nodes in the sound layer. These connections are non-learning, and their weights are set at -0.1. All nodes in the meaning layer are also connected to each other via inhibitory, non-learning connections with a weight set at -0.1. The inhibitory connections in the sound and meaning layers do not influence learning in any way, because the nodes in these layers are clamped during learning. The inhibition in these two layers makes sure that the output results of the model in production and perception are more clear-cut than they would be without inhibition. Inhibitory connections also exist in the middle layer, where they have a weight of -0.1. In this layer, inhibition does play a role in the learning phase. Varying the strength of the inhibition in the middle layer leads to different results, which I will discuss in some more detail in section 3.4 of this paper.

Table 1. *Means of F1 and F2 of each word, in % of continuum, frequency in Hz, and node number.*

Word	**Continuum (in %)**		**Frequency (in Hz)**		**Node number**	
	F1	*F2*	*F1*	*F2*	*F1*	*F2*
"I"	20	90	280	2750	5.6	55
"E"	50	75	550	2375	11	47.5
"A"	80	50	820	1750	16.4	35
"O"	50	25	550	1125	11	22.5
"U"	20	10	280	750	5.6	15

2.2 Parameters of the model

In this section, I will describe the many different parameters in the models, as well as their settings. Unless otherwise specified, parameter settings are the same as those used by Boersma and Chládková (2013), and they are kept constant across simulations.

Activity minimum and maximum. The activity minimum and activity maximum parameters determine the value of minimum and maximum activity of nodes. In the models described here, the activity minimum parameter is set to 0, and the activity maximum is set to 3. Negative activity of nodes is thus not allowed in the models.

Weight minimum and maximum. The weight minimum and weight maximum parameters determine the value of the minimum and maximum strength of connections. In the models described here, the weight minimum parameter is set to -3, and the weight maximum is set to 3. This means that both inhibitory and excitatory connections are allowed in the model.

Activity leak. The activity leak parameter, used in equation (4) in section 2.3, introduces leak into the model. It causes activity to leak away from nodes. Increasing the value of this parameter leads to more activity leaking away. This parameter is set to 1 in all models described here.

Weight leak. The weight leak parameter introduces a different type of leak into the model: it causes connection weights to decrease. Increasing the value of this parameter leads to more weight leaking away. This parameter is set to 0.5 in all models described here.

Spreading rate (η_a in (4)). The spreading rate parameter controls the speed of activity spreading through the model. Activity spreads through the model in small steps, causing the activities of the nodes to change gradually on each step. The lower the value of this parameter, the smaller the rate of activity spreading will be, and the longer it will take to complete each learning trial. This parameter is set to 0.01 in all models described here.

Number of times of activity spreading. This parameter controls the number of times of activity spreading. In all models described here, this parameter is set to 500. This means that the activity is allowed to spread 500 times (at a spreading rate of 0.01) in each learning trial before the weights are updated. The higher this value is, the longer it will take to complete each learning trial.

Learning rate (η_w in (5)). The learning rate parameter determines the change in connection weights after activity spreading is finished. This parameter dictates by how much a connection weight can change. The learning rate in all models described here is set to 0.001. This means that connection weights can only change a little in each learning trial, making a large number of learning trials necessary. If the value of this parameter is increased, learning would be faster, but also less precise. Smaller values, on the other hand, result in slower, but more precise learning.

Standard deviation of ambient. This parameter determines the variation in sounds in the environment. It is an equation, shown in (1), which includes other parameters.

(1) $$St.\ dev.\ of\, ambient = \left(\frac{number\ of\ nodes\ in\ bottom\ layer}{2} - 1\right)/peak\ sharpness/number\ of\, categories$$

For all models described here, the number of nodes in the bottom layer is 60. The *peak sharpness* and *number of categories* parameters are kept constant as well, at values of 2 and 6, respectively. The value of the standard deviation of ambient parameter thus equals 2.416. This parameter is used when the model computes which nodes in the bottom layer should be activated, which takes place in the learning phase (see also section 2.3). The larger the values of the peak sharpness and number of categories parameters get, the smaller the value of the standard deviation of ambient parameter will be. If this parameter's value is small, the variation from the mean is smaller, and activation patterns for the same words across learning trials will be more similar. A larger parameter value will ensure that there is more variance in activation patterns for the same words across learning trials.

Auditory spreading. The auditory spreading parameter determines the amount of activity on and around the F1 and F2 means of the chosen word: the higher the value of this parameter, the more concentrated the activity will be around a particular mean. If the value decreases, nodes more distant from the means will also receive activation. The auditory spreading parameter is also used in the learning phase (see section 2.3). This parameter is computed from other parameters, as shown in (2):

(2)
$$Auditory\ spreading = \frac{number\ of\ nodes\ in\ bottom\ layer - 1}{auditory\ sharpness}$$

The number of nodes in the bottom layer in all models equals 60. The *auditory sharpness* is set to 60 in all models. The value of the auditory spreading parameter thus equals 0.983. The parameter is used in computing the value of the activities on the bottom layer, which is shown in (3) in section 2.3. Increasing the value of the auditory sharpness parameter leads to a decreasing value of the auditory spreading parameter. The value of the auditory sharpness parameter in my models is increased in comparison to the model of Boersma and Chládková (2013) to ensure that activity is more concentrated around the F1 and F2 means, which leads to more clear-cut activation patterns when testing production in the models.

Instar and *outstar*. The instar and outstar parameters determine which learning algorithm is used when the connection weights are updated in the learning phase (see (5)). To use the instar learning algorithm, the instar parameter should be set to 1, and the outstar parameter to 0. To use the outstar learning algorithm, instar should be set to 0, and outstar to 1. To use the inoutstar learning algorithm, the instar and outstar parameters should both be set to 0.5 (Boersma, Benders & Seinhorst, 2013). The inoutstar learning algorithm is used in all models described in this paper, which means that here, the instar and outstar parameters are both always set to 0.5.

Total number of inputs. This parameter simply controls the number of learning trials in the learning phase. The value of this parameter in all models described here is set to 20000. The higher this value, the more combinations of activation patterns are presented to the model in the learning phase, and the longer this learning phase will take. If the value of this parameter is too low, then the model might not be presented with enough trials of the different words to be able to successfully learn the associations between activation patterns on the meaning and sound layers.

2.3 The learning phase

In the learning phase, the models are presented with specific combinations of patterns of activity in the meaning layer and in the sound layer. The weights of the connections between nodes in the different layers changes as a result of the specific inputs the model receives.

Firstly, a word is randomly selected from the five possible words in the language. All nodes in the meaning layer and in the sound layer are then 'clamped': the activities of these nodes are kept fixed during learning.

Secondly, activity is applied to all nodes in the meaning and sound layers that correspond to the randomly selected word. In the meaning layer, the activity of the four nodes corresponding to the selected word is set to one. In the sound layer, something slightly more complex happens. As Table 1 shows, the means of both F1 and F2 of all words are set at specific points on the sound layer. To account for variability, the exact same nodes are not always activated for the same words. F1 and F2 values are allowed to vary between presentations of the same word according to a Gaussian curve. The nodes in the sound layer on which the activation peak lies (the Gaussian random deviates) are computed on each learning trial with the means for F1 and F2 as shown in Table 1 and the value of the standard deviant of ambient parameter as the standard deviation. To approximate what happens when a sound wave excites part of the basilar membrane in the ear (i.e. hair cells adjacent to the ones sensitive to a specific frequency are also excited to an extent when that frequency is applied), the nodes adjacent to the mean F1 and F2 calculated for a specific word are also activated according to a Gaussian curve. Equation (3) below shows how the activation patterns in the sound layer are calculated.

(3) $$Activation\ on\ node\ number = e^{\frac{(-0{,}5(node\ number - nodeF1))^2}{auditory\ spreading^2}} + e^{\frac{(-0{,}5(node\ number - nodeF2))^2}{auditory\ spreading^2}}$$

In every learning trial, this equation is applied to every node in the sound layer (node number 1 to node number 60). In the equation, *nodeF1* and *nodeF2* refer to the F1 and F2 means of the selected word, as computed just prior to this step. The auditory spreading parameter is described in more detail in section 2.2. Adding the two sections of the equation together ensures that we end up with activation in two distinct places on the sound continuum. To summarise: for every selected word, a group of four nodes in the meaning layer is activated, and activity on the auditory layer appears as two Gaussian curves centred on a varying mean. An example of a possible input to the model when the selected word is 'I' is shown in Fig. 2.

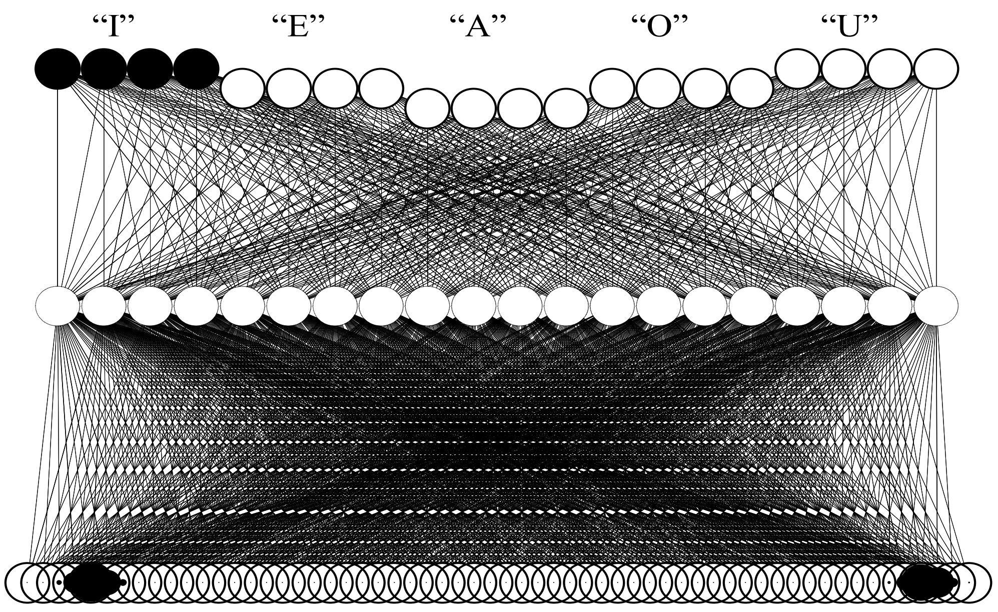


Figure 2. Example of an input presentation when the word 'I' is selected as input.

Thirdly, the activities are allowed to spread throughout the model via the connections between nodes. This spreading of activities takes place in small steps, as shown in (4) below:

(4)
$$\Delta e_j = \eta_a \left(\sum_{connected\ nodes\ i} (w_{ij} * a_i - activity\ leak * e_j\right)$$

Here, e_j is the excitation of node j, η_a is the spreading rate (see section 2.2), w_{ij} is the weight of the connection between nodes i and j and a_i is the activity of node i. The activity leak parameter is discussed in section 2.2.

Lastly, after activity spreading has finished, the connection weights are updated according to the learning rule shown in (5).

(5)
$$\Delta w_{ij} = \eta_w \left(a_i a_j - instar * a_j w_{i\,j} - outstar * a_i w_{i\,j} - weight\ leak * w_{i\,j}\right)$$

Here, w_{ij} is the weight of the connection between nodes i and j, η_w is the learning rate, and a_i and a_j are the activities of nodes i and j, respectively. The *instar* and *outstar* parameters, as well as the *weight leak* parameter, are discussed in section 2.2. This learning rule, the so-called 'inoutstar' learning rule (Boersma, Benders & Seinhorst, 2013), is a combination of the instar and outstar learning rules proposed by Grossberg (1969 and 1976). The updating of connection weights according to this rule will ensure that connections between nodes that are active together will become stronger, whereas connections between nodes that are not active together will become weaker.

After this last step, the activity of all nodes in the model is set to zero, and clamping of nodes is undone. The cycle then starts again with the random selection of a word. These steps are repeated until a certain number of input presentations, determined by the total number of inputs parameter, has been reached.

3. Results

3.1 Initial models

The first type of model I simulated was designed in the way described above. After feeding it 20000 input pairs, the model was tested to see whether phonemes or features had emerged. To test production of the model, a group of nodes belonging to a word in the meaning layer is activated, and the activity is allowed to spread to the layers below. The first thing to look for is whether the activity in the sound layer is distributed correctly. All words in the toy language are associated with a specific combination of F1 and F2 values that is represented on the sound layer, and after 20000 learning trials, the model should have learned to associate these patterns of activation with the correct words, and thus, assuming the bidirectionality of the connections, the model should be able to produce the correct sound output belonging to an activated word. Only after it is established that the model has correctly associated F1 and F2 values with words can we investigate whether features or phonemes have emerged in the middle layer of the model.

To answer the question whether words are represented in the model as features or as phonemes, we have to look at the pattern of activity on the middle layer of each word in production. If certain words share some activation patterns in the middle layer (and the activation patterns on the sound layer are still correct for all words), we may conclude that words are represented as combinations of features (e.g., vowel height and vowel backness). If, on the other hand, the activity pattern in the middle layer corresponds to the activity in the meaning layer, that is, if all different words show a different activation pattern in the middle layer, then we may conclude that words are represented in the model as phonemes.

The production success of the first types of models was very low. Only 7 out of 50 models correctly produced all words. Figure 3 shows an example of a model which correctly produces "E". Although the activation pattern on the sound layer is correct, something curious is going on in the middle layer. As we can see, activating the word "E" leads to activation on nodes 1, 8, 11 (only a tiny amount), and 13. However, there are no connections from the nodes belonging to "E" in the top layer to node 1 (or to node 11, for that matter) in the middle layer. How can these nodes be active then?

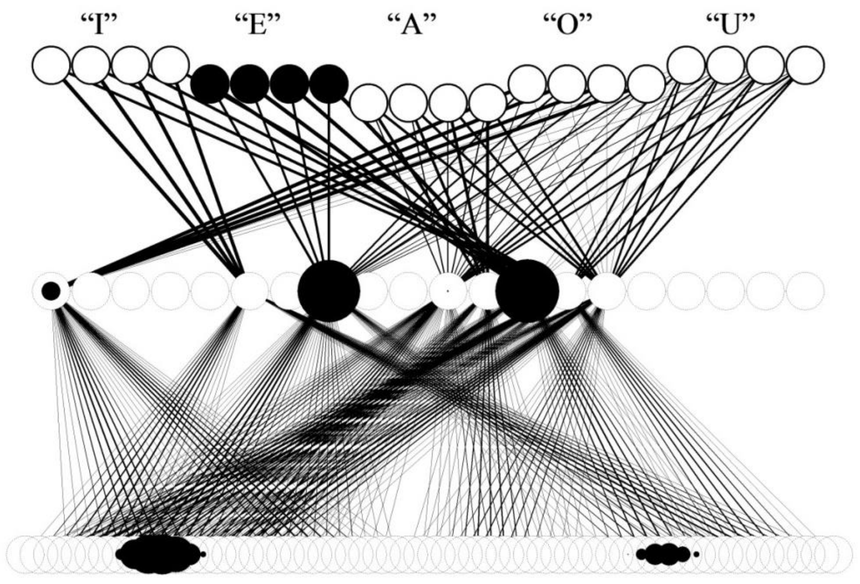


Figure 3. Example of a model producing "E".

The answer lies in the activation spreading and bidirectionality of the connections between nodes. The activation from the "E" nodes in the top layer spreads via the excitatory connections to nodes 8 and 13 on the middle layer, and from here, the activity spreads to the sound layer. This is not the end of the activation spreading, however. Activation is allowed to spread to any nodes that are unclamped, and because the connections allow activation spreading in both directions, activation can also spread from the bottom layer back up to the sound layer. The active nodes in the sound layer have connections to node 1 and 11, and, as a result, activation spreads from the sound layer to these nodes in the middle layer. Even though there are no direct connections between "E" nodes in the top layer and nodes 1 and 11 in the middle layer, these two nodes become active as activation spreads down to the sound layer and up again to the middle layer. Note that the nodes in the meaning layer are all clamped, so activation from the middle layer cannot spread back up to the meaning layer.

If we look at activation patterns in the middle layers of the models which correctly produced all words, we can conclude that we see features emerge there. Figure 4 shows an example.

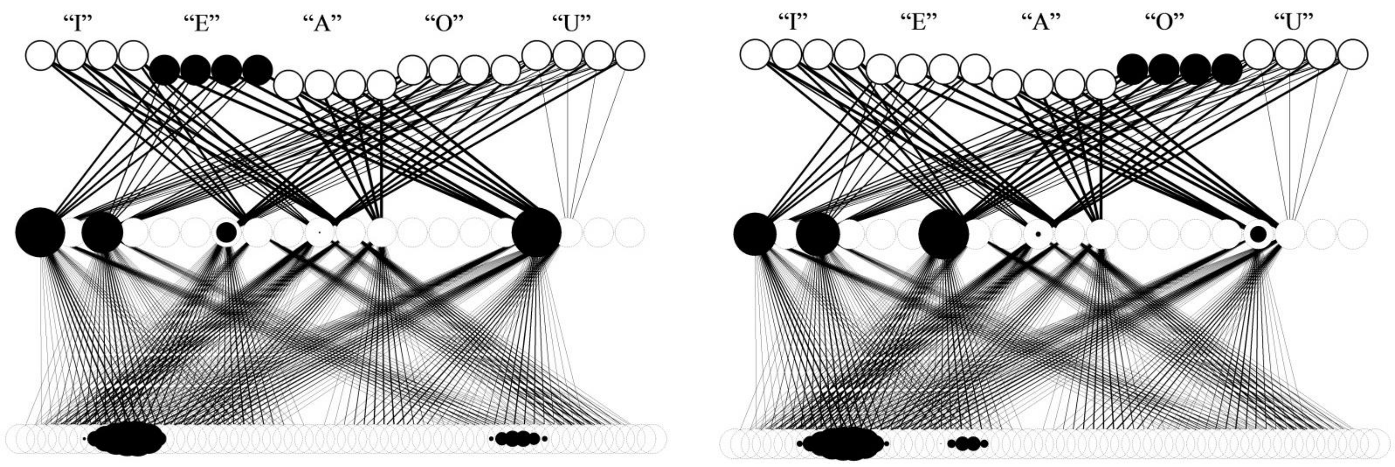


Figure 4. Emergence of features.

The words "E" and "O" are produced correctly in this model, and, moreover, they share an activation pattern on the middle layer. Both words activate nodes 1, 3, 7, 10 (a tiny amount), and 17 in the middle layer. However, some activity in the middle layer is caused by the activation spreading back upwards from the sound layer. If we look at which nodes "E" and "O" actually have connections to, that is, which nodes in the middle layer would be activated if the activity would not spread back upwards from the sound layer, a slightly different picture emerges. "E" would then activate nodes 1, 3 and 17 in the middle layer, and "O" would activate nodes 1 and 7. Node 10 would not be activated by either of these words. The activation patterns on the middle layer thus point to the emergence of features: "E" and "O" share activation on node 1 in the middle layer, and each word activates different nodes in the middle layer in addition.

3.2 Layer-by-layer activity spreading

To improve the models, activation spreading was restricted: the activity was now only allowed to spread downwards, layer by layer. To ensure this layer-by-layer spreading, the sound layer was clamped so that the activation could only spread from the meaning layer to the middle layer. Subsequently, the middle layer was clamped and the sound layer unclamped, so that the activation could spread from the middle layer downwards, but not up to the middle layer again as it could before. Throughout activation spreading, the nodes in the top layer were all clamped (as before).

As a result of this change, production success increased. Now, 28 out of 50 models correctly produced activation patterns on the bottom layer for all words. Figure 5 shows a comparison of the production of "E" in the same model, with activation spreading before (left) and after the activation spreading was restricted (right). The production of "E" has clearly improved after activation spreading is restricted.

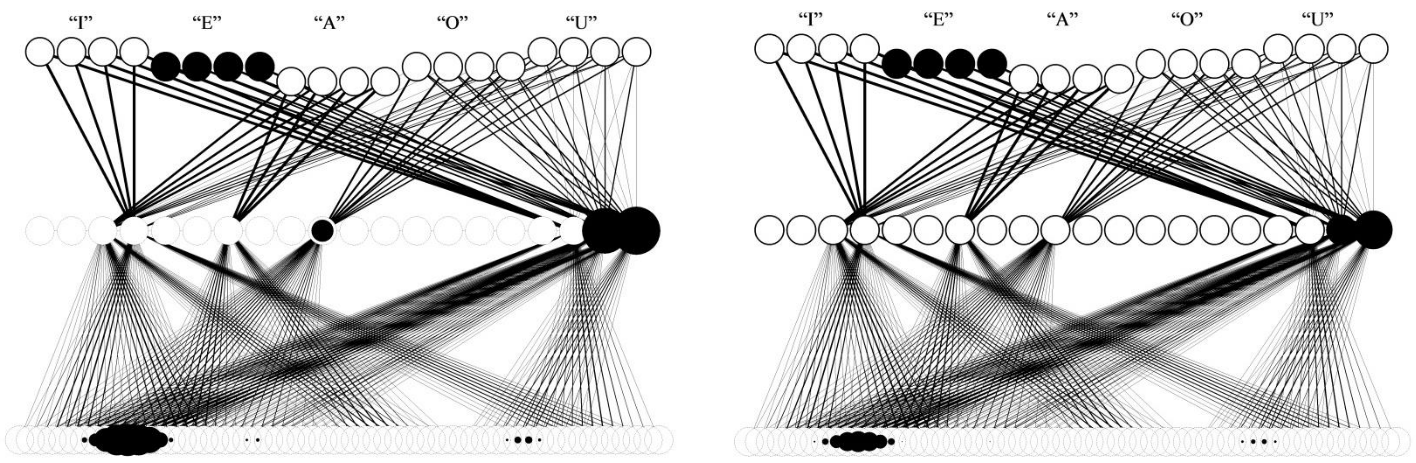


Figure 5. A model producing E when activation is allowed to spread upwards (left) and when activation is allowed to spread only downwards (right).

However, the models had still not always learned to correctly associate patterns of activity on the sound layer with patterns of activation on the meaning layer. The most common problem, which was apparent in several models, was that the activation patterns on the sound layer were the same for different words (most commonly, for "E" and "O", but also for "A" and "U" and for "O" and "U", in some cases). Figure 6 shows an example of incorrect production of "A" and "U" in a model. Varying the number of learning trials and the inhibition strength in the middle layer did not remedy the production problems. Since some models had not correctly learned the associations between activities in the sound layer and those in the meaning layer, investigating the patterns of activation on the middle layer in these models was not relevant anymore.

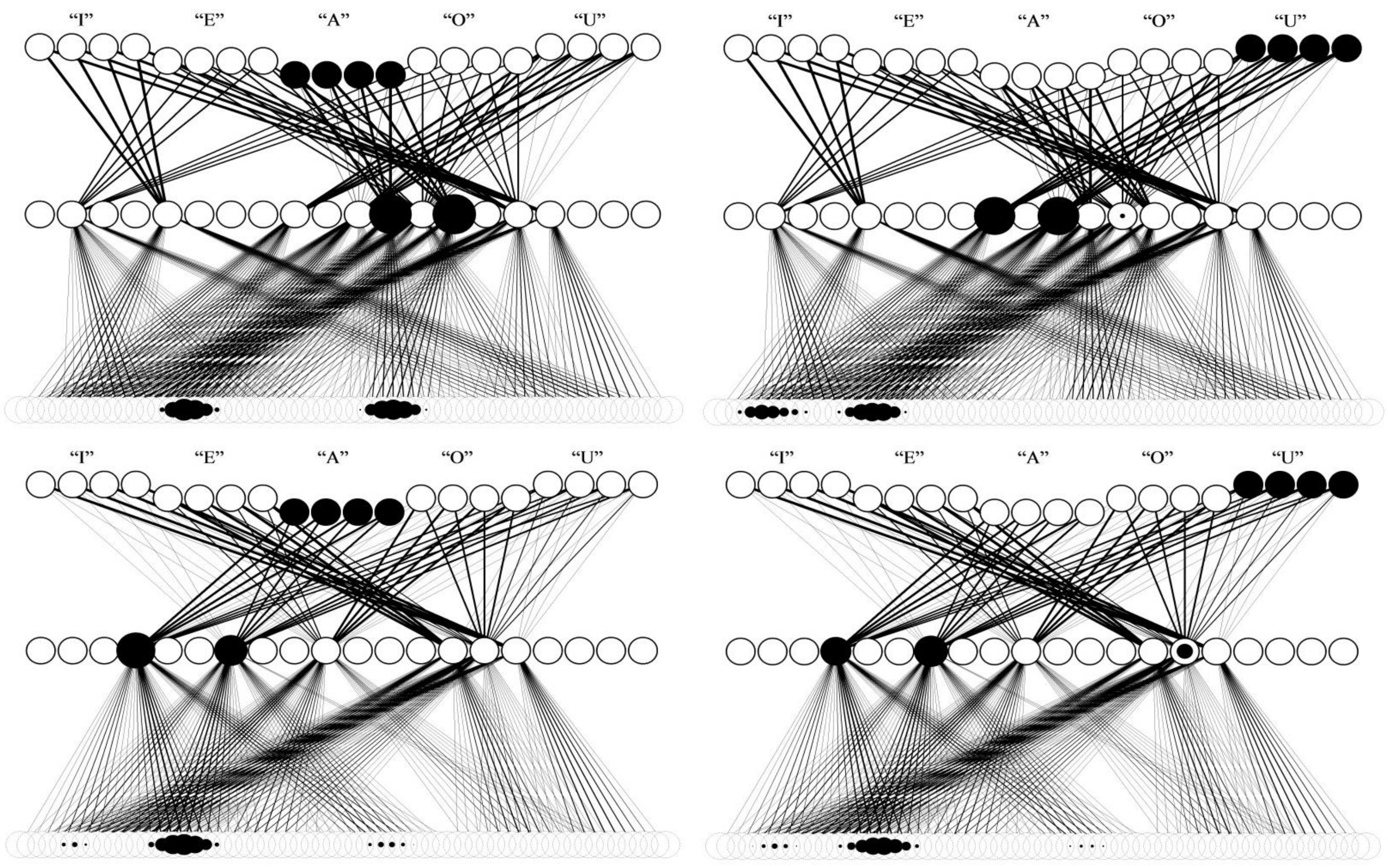


Figure 6. Testing production. The top row shows examples of a model with correct production of "A" and "U". The bottom row shows examples of a model with incorrect production of these words.

In the models which had correct production patterns for all words, the middle layer was investigated. In these models, there was an overlap in activity patterns on the middle layer of different words. Figure 7 below illustrates this.

In Figure 7, we can see that several words share activation patterns on the middle layer. The words "I" and "U", which have the same mean F1 (see Table 1), share activation on nodes 14 and 19 in the middle layer. "E" and "O" also have the same mean F1, and both activate node 6 of the middle layer in the model pictured above. Additionally, the mean F1 of "A" and the mean F2 of "U" are quite similar, and we can see that "A" and "U" both activate node 3 (although the activation of this node for "U" is not as strong as that for "A"). In other models not pictured here, there sometimes is an overlap between "O" and "U", and between "E" and "U". As shown in Table 1, the mean F1 of "O" and "E" is quite close to the mean F2 of "U", so this overlap can be expected. Overlap between other words seems unlikely, since the means of F1 and/or F2 are too far apart. The overlap in activity patterns here is a partial overlap, that is, two words share activation of certain nodes in the middle layer (e.g. "E" and "O" both activate node 6), but either may also activate other nodes ("E" activates node 20, "O" activates nodes 8 and 14).

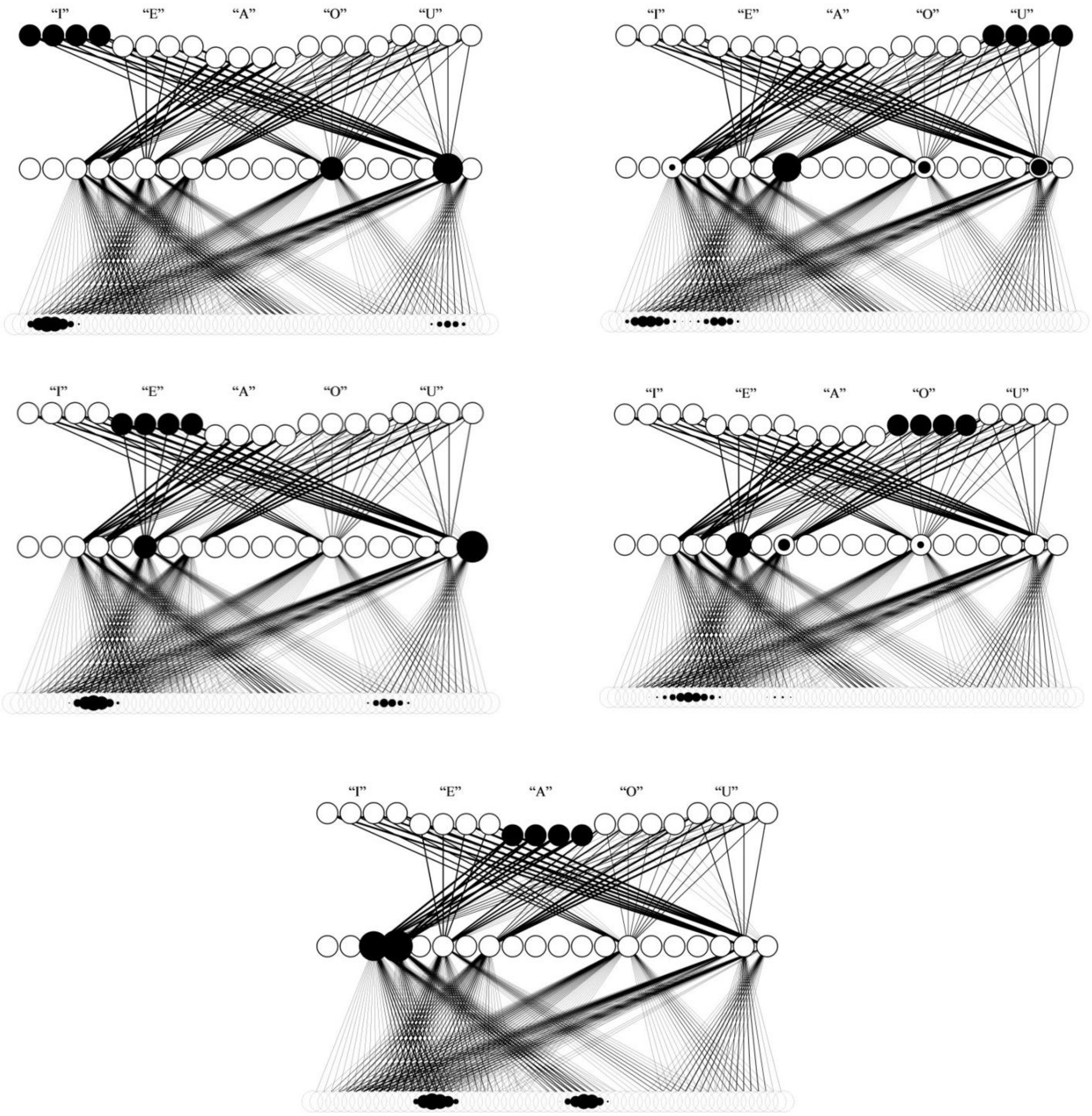


Figure 7. Example of a model with correct production, illustrating the emergence of features.

The emergence of features seen here is consistent with previous research. However, only 56% of models could correctly produce all words, which is required before we look at the activation patterns on the middle layer. Ideally, we would like to see that all models correctly produce all words, and that in all these models features emerge.

3.3 Weight normalization

In order to improve production success, a normalization of weights was built into the models (cf. Rumelhart & Zipser, 1985). This would ensure that models would correctly associate patterns of activity on the sound layer with words, and thus correctly produce all words. The additional weight normalization step changed the learning phase somewhat. After each time the weights were updated in the learning phase, they were normalized, such that for each node in the middle layer, the sum of the weights of the connections from nodes in the sound layer to that specific node in the middle layer equalled 4. Very strong connections still remained strong, and weak connections still remained weak, but the absolute values were normalized after each weight updating step. Weight normalization greatly improved the quality of the production output of the models. All models (n = 50) were now able to successfully associate patterns of activity on the sound layer to patterns of activity on the meaning layer, and they produced correct patterns of activity on the sound layer when specific words were activated in the meaning layer.

Figure 8 shows results of the production outputs typical of the majority of models with weight normalization. All models (n = 50) produced all words correctly. In 39 out of these 50 models, the activation patterns on the middle layer resembled those in Figure 8, that is, there was no overlap of active nodes in the middle layer between any of the words. From this, we can conclude that in these 78% of models, phonemes emerge.

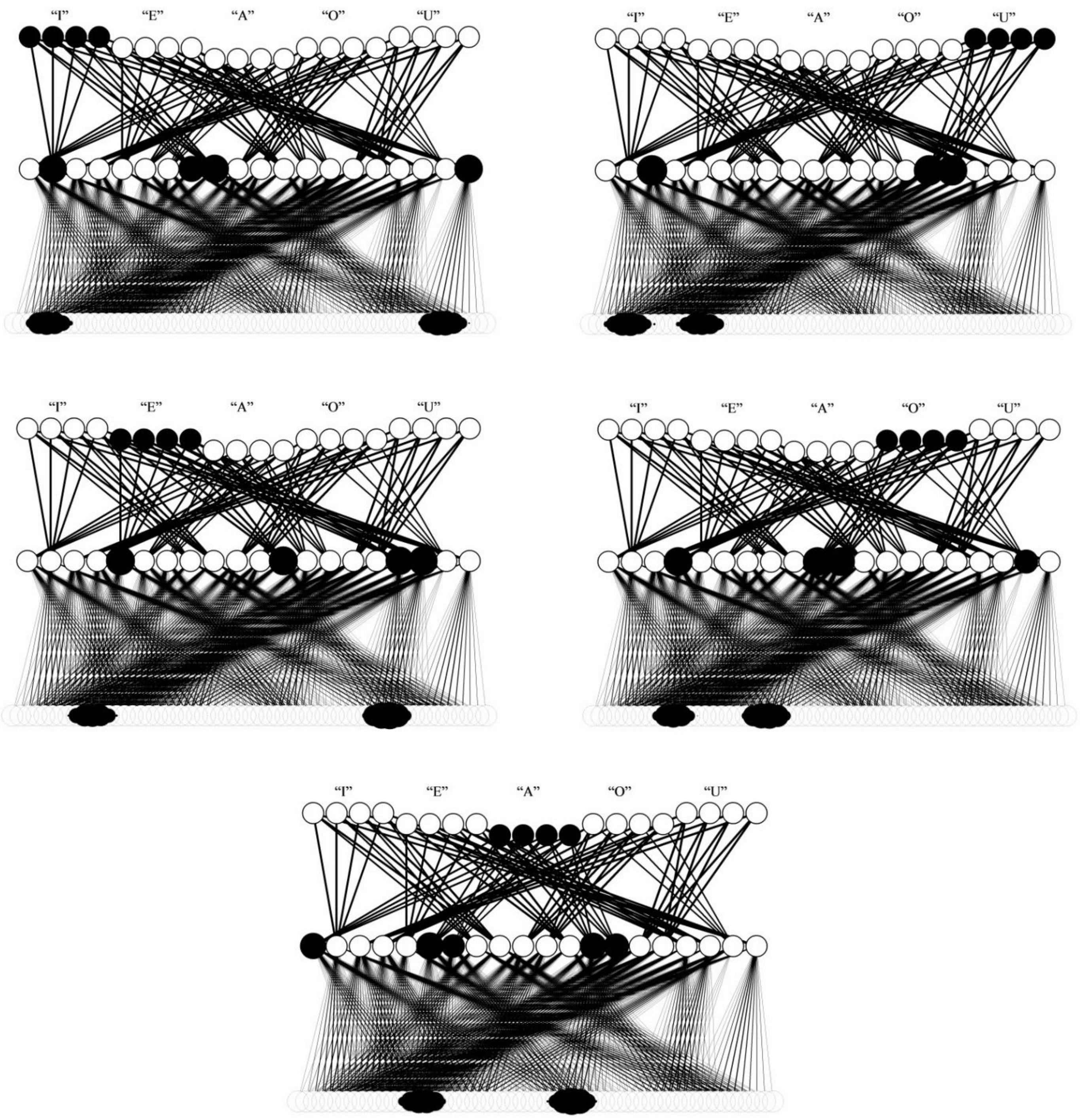


Figure 8. Production output typical of the majority of models with weight normalization; phonemes emerge.

In the remaining 11 models, there was some overlap between activity patterns in the middle layer for different words. An example is given in Figure 9. As we can see, "E" and "O" share activation on one node in the middle layer: node 3. However, the activation strength is small: smaller than 1. This is true of the majority of the models in which some words share activation in the middle layer. In most of these models, the words which share activation are "E" and "O" (54,4% of the aforementioned 11 models). In other models, the words which share activation patterns are "I" and "U" (18,2%), "O" and "U" (18,2%), or "E" and "U" (9,1%).

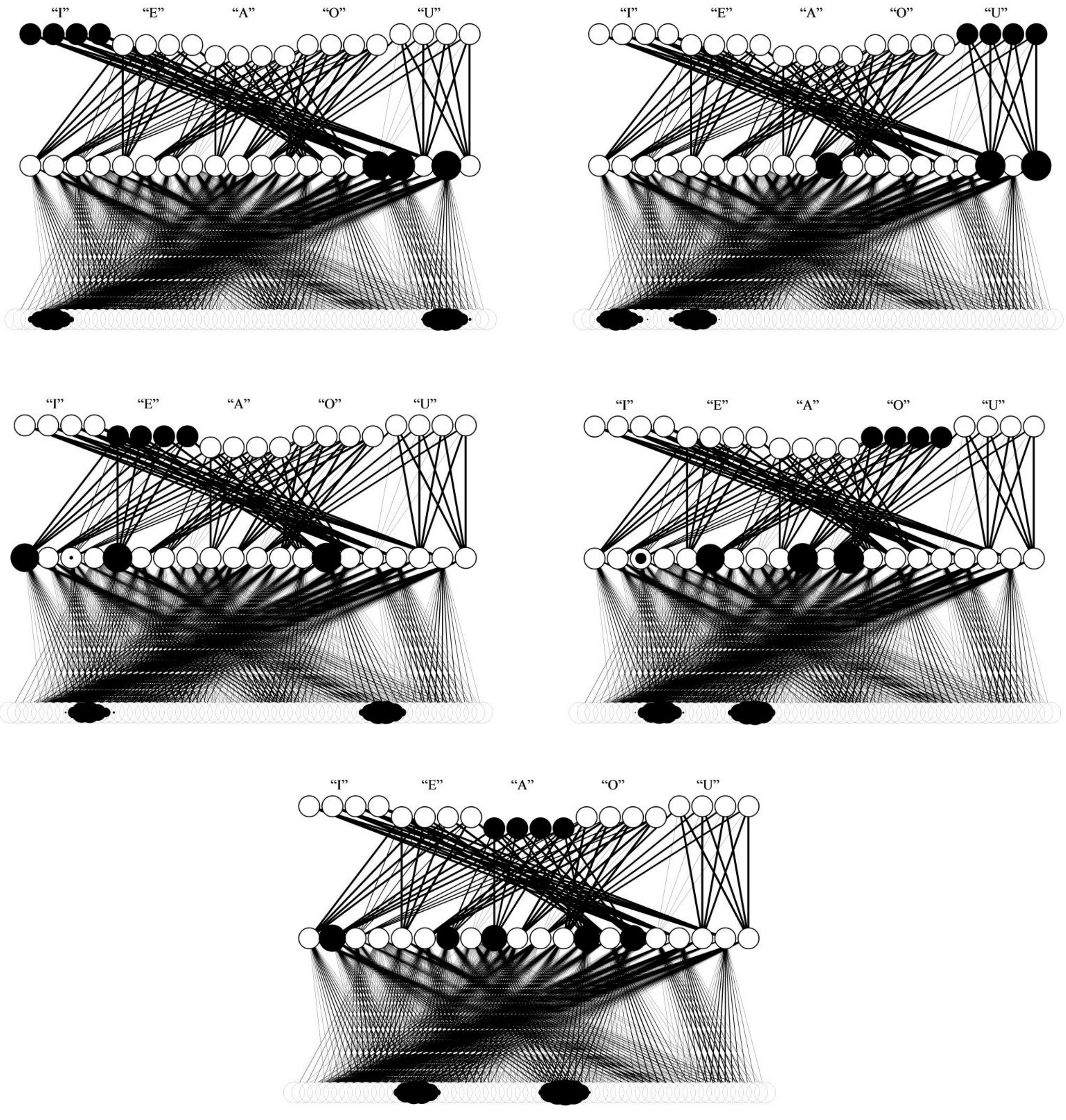


Figure 9. Production output typical of 22% of models with weight normalization.

Interestingly, in all models in which we find activity overlap in the middle layer between words, only two words display this overlap. When looking for feature emergence, we would expect that features emerge in all words, not just in two. "I" and "U" and "E" and "O" share a mean F1 value, so we would at least expect that nodes are shared between "I" and "U" *and* "E" and "O", not just between one of these word pairs.

Another similarity in the 22% of models that display overlap is that there are relatively many weak connections between nodes in the meaning layer and nodes in the middle layer. Connection strengths are translated as line thickness in the figures, where a thicker line represents a stronger connection. In the models in which we see phonemes emerge (e.g. in the model in Figure 8) the connections are very strong. Weak connections may be a sign of an 'unstable' model – a model in which the connection strengths would still change if we would take it through more learning trials. Such models have not completely finished learning the associations between patterns of activity in the different layers. It is not unthinkable that, had there been more learning trials, the connections between nodes of one word and a particular node in the middle layer would have disappeared, whereas the connections between another word and this node in the middle layer would have strengthened. In the model in Figure 9, for example, the connections from the "E" nodes to node 3 in the middle layer might have been in the process of weakening, and might have disappeared altogether if the model had been presented with additional learning trials.

Although there is some overlap in activity patterns in the middle layer in 22% of the models with weight normalization, it cannot be concluded that features emerge in these models. As explained in the previous two paragraphs, there is evidence that these models are unstable, and have simply not finished learning. In the majority of models with weight normalization, phonemes emerge, and it cannot be ruled out that phonemes would also have emerged in the other models if these models had been presented with more learning trials.

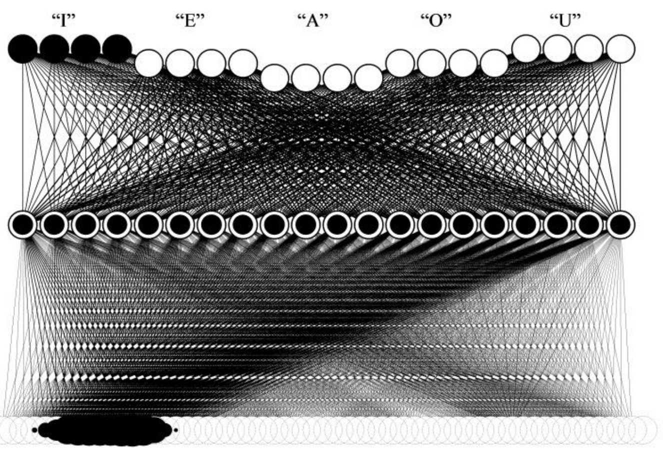
3.4 Inhibition in the middle layer

In the models described above, the strength of the inhibitory connections between all nodes in the middle layer was -0.1. To explore the effect of different inhibition strengths in the middle layer, a small number of models with inhibitory connections of varying strengths were simulated. Connection strengths were varied in increments of 0.05 (arbitrarily chosen), from 0 to -0.5. All other parameters were set to the same values as in the models described above, and their values were kept constant across models. As in the models in section 3.3, activation in these models was allowed to spread layer by layer, and weight normalization was applied. Five models were simulated for each connection strength, in the same way as described above. Although we cannot draw any definitive conclusions from such a small sample size, the simulations give an indication of the effect of the inhibition strength on the performance of the model.

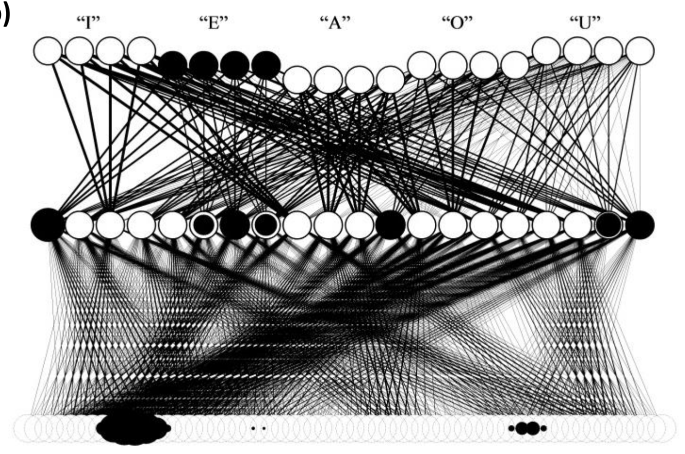
The most apparent effect of varying the inhibition strength is that this influences the number of nodes that become active in the middle layer for each word. When a node is active and there are inhibitory connections to other nodes, the activity of that node causes connected nodes to become less active. The simulations showed that when there was no inhibition at all, the different words activated all nodes in the middle layer. As the inhibition got stronger, however, words activated fewer nodes in the middle layer. The weakest inhibition (-0.05) caused words to each activate between 4 and 10 nodes in the middle layer. An inhibition strength of -0.1 caused words to activate between 3 and 6 nodes, and with an inhibition strength of -0.15 only 2 to 4 nodes in the middle layer were activated per word. The number of activated nodes per word further decreased for stronger inhibitory connections: 1-3 active nodes for inhibition strengths of -0.2 and -0.25; 1-2 active nodes for strengths of -0.3 and -0.35; and only 1 active node for inhibition strengths of -0.4 and below. Figure 10 below illustrates this.

The inhibition strength in the middle layer affects the activation patterns on the sound layer as well. In the simulated range of 0 to -0.5, there appears to be a smaller sub-range in which activation patterns on the bottom layer are always correct. Incorrect activation patterns on the bottom layer were found for inhibition strengths of 0 (i.e. no inhibition) and -0.05 (see e.g. Figure 10, a and b), and for inhibition strengths of -0.45 and stronger (e.g. Figure 10f). Activation patterns on the sound layer were always correct for inhibition strengths between -0.1 and -0.4. When inhibition is too weak, too many nodes become active in the middle layer, and there is thus too much overlap on the middle layer between different words for the model to produce the word-sound combinations correctly. When the inhibition is too strong, on the other hand, only a few nodes in the middle are allowed to become active. Words may then only activate a single node in the middle layer, and often this node is shared between different words, leading to an incorrect activation pattern on the sound layer.

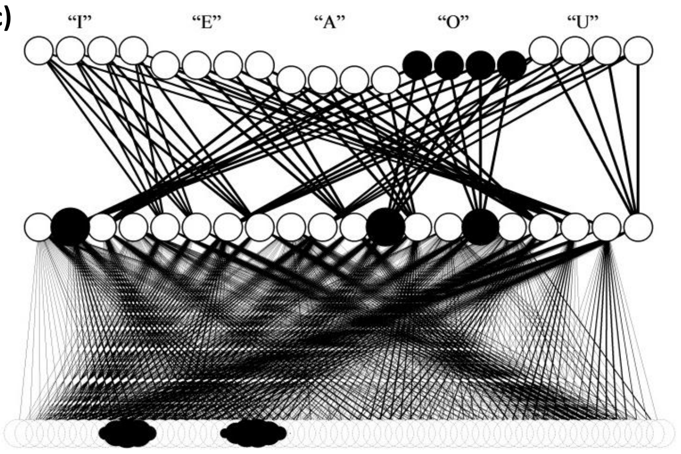
a)



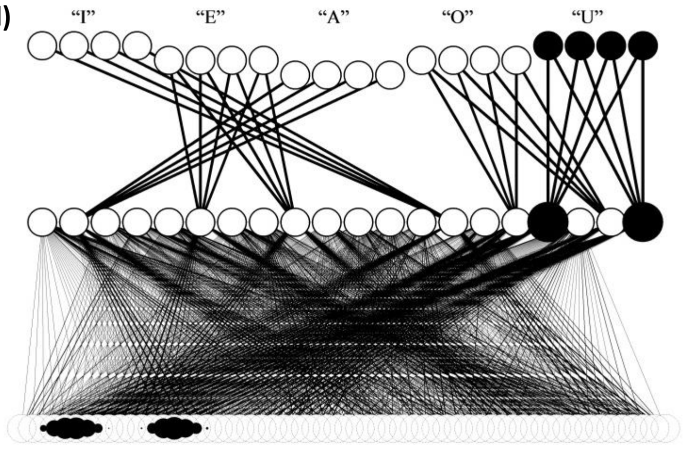
b)



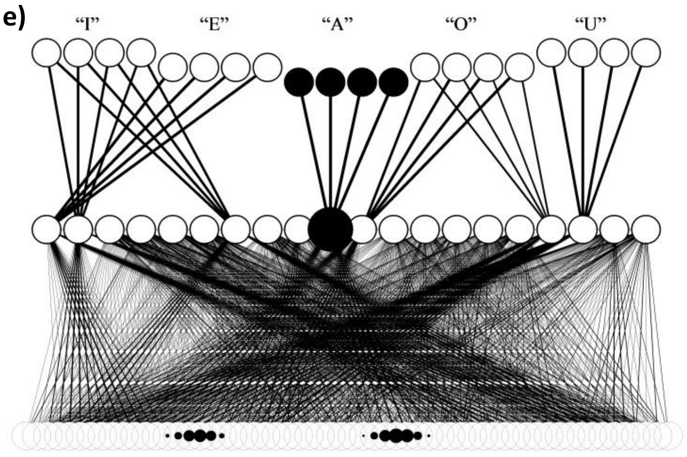
c)



d)



e)



f)

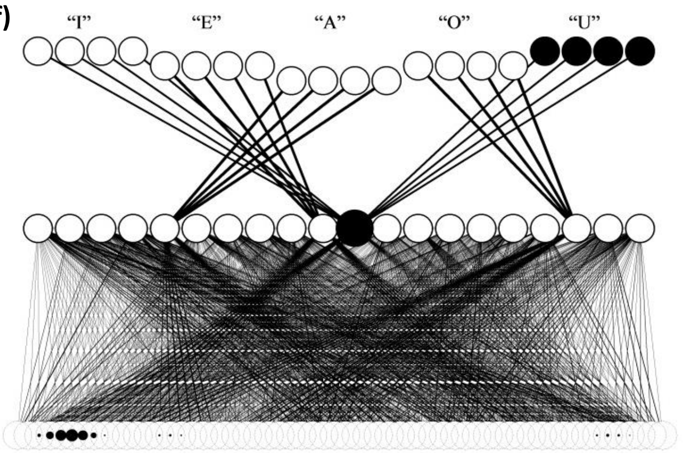


Figure 10. Production outputs of models with different inhibition strengths in the middle layer. Not all simulated inhibition strengths are shown. a) no inhibition; b) inhibition strength -0.05; c) inhibition strength -0.15; d) inhibition strength -0.25; e) inhibition strength -0.35; f) inhibition strength -0.5.

All models with correct production were examined to investigate whether phonemes or features emerged. For models in the inhibition range of -0.15 to -0.4, in which production of all words was always successful, phonemes emerged. Several models with stronger inhibition (i.e. inhibition of -0.45 and -0.5) did not produce all words correctly. When looking at the models with these inhibition strengths that did produce all words correctly (4 out of 5 models for each inhibition strength), it was found that phonemes emerged. This is not surprising, as with inhibition this strong, each word only activates one node in the middle layer, so there cannot be any overlap of activity patterns on the middle layer (unless different words share one node in the middle layer, in which case the activity pattern on the sound layer will be incorrect for these words). As described in section 3.3, in most models with inhibition of -0.1 in the middle layer, phonemes emerged. In the remaining models, it seemed like features emerged. However, for the same reasons as stated in section 3.3 above, these models may be considered unstable. The same can be said of the models with an inhibition strength of -0.05 which had correct production (3 out of 5 models) and in which features seem to emerge. Finally, in models without inhibition in the middle layer, all nodes in the middle layer were active for all different words, and none of the words were produced correctly. Therefore, looking at the emergence of features or phonemes in these models is not applicable.

Although the small number of models simulated here might not warrant a definitive conclusion, based on the results described in this section we can tentatively conclude that it is possible that an inhibition of -0.1 was too weak for all models described in section 3.3 to reach a stable state after 20000 learning trials. A stronger inhibition (between -0.1 and -0.4) might lead to better results than those described above.

3.5 Perception

In the neural network models described in this paper, activation can spread top-down, but also bottom-up. Top-down here refers to production, as words are selected and a sound (i.e. an activation pattern on the sound layer) is formed. In perception, on the other hand, models are presented with an activation pattern on the sound layer (i.e. a combination of an F1 and F2 value: a sound), and as activation spreads upwards, nodes in the top layer (i.e. words) become active. If the model has correctly learned to associate words with sound, it should give a correct word output for familiar sounds. But how does the model react to strange (i.e. unfamiliar) sounds?

Testing perception in a model is done in much the same way as testing production: activity is applied to the sound layer, and it is then allowed to spread, layer by layer, to the meaning layer. In order to test the model's reaction to unfamiliar sounds, it is presented with all possible combinations of F1 and F2 values. As it would take too long to present the model with all possible values, the F1 continuum, ranging from 100 Hz to 1000 Hz, was divided into steps of 20 Hz, and the F2 continuum into steps of 50 Hz. Simulations were run using the same parameter settings as for the models in section 3.3. The script used to test perception drew a diagram to make visualization of results easier. One of these diagrams is shown in Figure 11.

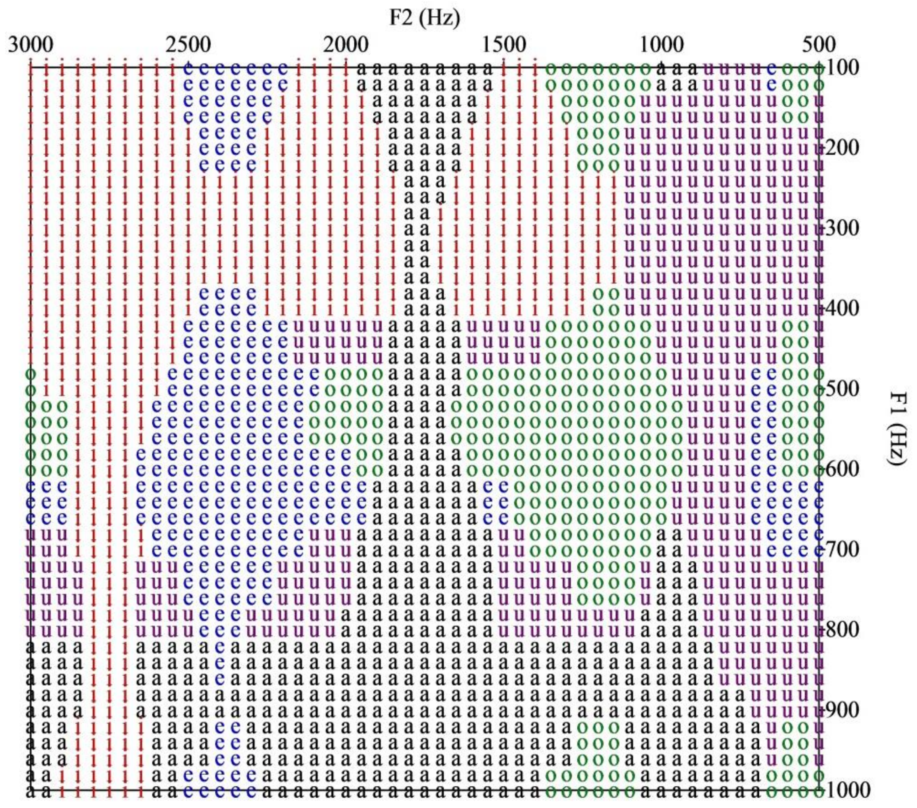


Figure 11. Perception diagram of a model with weight normalization (as described in section 3.3). The F2 continuum is drawn on the x-axis, and the F1 continuum on the y-axis. "A" is drawn as a black 'a', "E" as a blue 'e', "I" as a red 'i', "O" as a green 'o', and "U" as a purple 'u'.

Figure 11 shows a vowel diagram, in which the F2 continuum is drawn along the x-axis, and the F1 continuum is drawn along the y-axis. At every combination of F1 and F2 that is presented to the model, a symbol (a, e, i, o or u) is drawn to indicate which of the five words ("A", "E", "I", "O", or "U") 'wins'. A 'winning' word is a word of which the nodes in the meaning layer become most active for a specific sound input. Although multiple words may be activated for specific sounds, only one word is activated the most, and thus wins and is drawn in the diagram. It is important to realize that the model is being presented with an activation pattern that is not labelled as containing an F1 and an F2 value; the model simply gets an input consisting of two areas of activation on the sound value. The model is also not told that either one of these areas of activation is more important than the other for perceiving words.

As we can see in Figure 11, the main perception areas of the five words are where we expect them to be (see Table 1 for mean formant values used in learning): Sounds with a low F1 and high F2 are perceived as "I"; sounds with a low F1 and low F2 are perceived as "U"; sounds with a medium F1 and medium/high F2 as "E"; medium F1 in combination with a medium/low F2 is perceived as "O"; and a high F1 in combination with a medium F2 is perceived as "A". Additionally, the model also classifies unfamiliar sounds as one of the five words. The horizontal and vertical areas adjacent to the expected categorization areas are due to the model taking into account only one of the formant values. For instance, "E" is not only perceived when the model is presented with a medium F1 and medium/high F2, but also, for example, when it is presented with a lower F1 but similar F2 value. Similarly, "A" is not only perceived when the model is presented with a high F1 and a medium F2 value, but also when the model is presented with combinations in which one of the values is similar to either this high F1 or medium F2 value, which leads to the horizontal and vertical perception areas extending from the expected perception area for this word. The diagram in Figure 11, with its horizontal and vertical bands of perception of specific words, is quite similar to a figure in a chapter by Boersma & Escudero (2008; fig. 10). By comparing the way the model was simulated here to the Boersma & Escudero (2008) study, we might gain some insight into how to improve the current models, as I will discuss in section 4 below.

4. Discussion

The aim of the research described here was to investigate whether phonemes or features emerge in a neural network model with a single input continuum. It was expected that features would emerge, as this had been shown to happen in earlier research.

The results of this research were mixed: in some circumstances, phonemes emerged, and in others, features emerged. The production success of the initial models was very low. In the models which did correctly produce all words, features emerged (Fig. 4). When the spreading of activity was restricted (so that activity was only allowed to spread layer by layer), production success went up, and we saw that features emerged in the models which correctly produced all words (Fig. 7). However, before we draw any definitive conclusions about the emergence of features or phonemes, we would like all models to produce all words correctly, *and* see features emerge in all these models.

After a weight normalization step was added to the learning phase of models, production success went up significantly. Now, all models produced all words correctly. Despite this, not features, but phonemes emerged in most models (Fig. 8). The models that did show some overlap in activity patterns in the middle layer (Fig. 9) were argued to be unstable. As said before, we would like to see that all models correctly produce all words, and that in all these models features emerge. By adding weight normalization to the models, we succeeded in the former. However, by normalizing the weights, we changed an important aspect of the neural network model: the bidirectionality of it. The connections are still bidirectional in the sense that activity can flow through them both ways, but the weight normalization step artificially changed the weight of the connections to nodes in the middle layer. Weight normalization only works in one direction. If we are hoping to find a way to simulate models that correctly produce all words and show feature emergence, we may have to let go of weight normalization, and find another way to ensure production success in all models.

The simulations of models with varying inhibition strengths in the middle layer showed that there seems to be an optimal range of inhibition strength (Fig.10). Although the sample sizes of these simulations were small, they showed that an inhibition strength of -0.1, as used in most models in this study, might be on the border of this optimal range. Increasing the inhibition in the middle layer (i.e.

using an inhibition strength value in the optimal range of approximately -0.1 to -0.4) may improve stability in models.

When perception was tested, models were shown to correctly perceive familiar sound inputs (i.e. activation patterns on the sound layer), as well as to categorize unfamiliar sounds. The diagram in Figure 11, with its horizontal and vertical bands of perception of specific words, was said to be quite similar to a figure in a paper by Boersma & Escudero (2008; fig. 10). This paper looked at phonological perception from an Optimality-Theoretic viewpoint. In Optimality Theory, constraints determine how an incoming sound is perceived; constraints in Optimality Theory could thus be said to be analogous to connections between nodes in a neural network model. Boersma & Escudero (2008) tested several models using both negatively formulated constraints (typically used in OT modelling) and positively formulated constraints. A negatively formulated constraint could be something like: "An F1 of 280 Hz should not be perceived as /a/", whereas a positively formulated constraint could be: "An F1 of 280 Hz should be perceived as /i/". If connections in neural network models are like the constraints in OT, then excitatory connections would be analogous to positively formulated constraints, and inhibitory connections would be analogous to negatively formulated constraints. Boersma & Escudero (2008) showed that learning to perceive vowels using negatively formulated constraints results in categorization areas neatly organized around vowel centres, whereas learning to perceive vowels using positively formulated constraints results in categorization areas extending vertically or horizontally away from the vowel centres (Boersma & Escudero, 2008; fig. 10), which is similar to what we see in figure 11 above. From this, we may conclude that using inhibitory instead of excitatory connections in the neural network model may improve perception in the models. However, when the authors repeated the same simulations with Harmonic Grammar (a predecessor of OT), they found that perception success was similar for positively and negatively formulated constraints. It is therefore unclear whether the models described in this paper would benefit from having inhibitory connections instead of excitatory connections. Nevertheless, this approach may still be a productive one, which could be explored in further research.

Neural network modelling is a fairly novel approach to issues in phonetics and phonology. A lot of work still has to be done in order to better understand how the brain handles various phonetic and phonological processes, but the neural network modelling approach is certainly a promising one.

References

Boersma, P., Benders, T., & Seinhorst, K. (2013). *Neural network models for phonology and phonetics.* Manuscript in preparation.

Boersma, P. & Chládková, K. (2011). Asymmetries between speech perception and production reveal phonological structure. *Proceedings of the 17th International Congress of Phonetic Sciences, Hongkong*, 328-331.

Boersma, P & Chládková, K. (2013, January). *Emergence of vowel features in a neural network.* Talk presented at the CUNY Conference on the Feature in Phonetics and Phonology, New York.

Boersma, P. & Escudero, P. (2008). Learning to perceive a smaller L2 vowel inventory: an Optimality Theory account. In Peter Avery, Elan Dresher & Keren Rice (eds.), *Contrast in phonology: theory, perception, acquisition* (271-301). Berlin: Mouton de Gruyter.

Boersma, P., & Weenink, D. (2005). PRAAT: doing phonetics by computer (version 5.3.53). Available online: http://www.praat.org/.

Grossberg, S. (1969). Embedding Fields: A Theory of Learning with Physiological Implications. *Journal of Mathematical Psychology, 6.* 209-239.

Grossberg, S. (1976). Adaptive pattern classification and universal recoding: I. Parallel development and coding of neural feature detectors. *Biological Cybernetics, 23,* 121–134.

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

McCulloch, W., & Pitts, W. (1943). A logical calculus of the ideas imminent in neural nets. *Bulletin of Mathematical Biophysics, 5*, 115-137.

Recasens, D. & Espinosa, A. (2009). Dispersion and variability in Catalan five and six peripheral vowel systems. *Speech Communication, 51,* 240–258.

Rumelhart, D. E. & Zipser, D. (1985). Feature Discovery by Competitive Learning. *Cognitive Science, 9*, 75-112.

Appendix A: The initial model

```
#The architecture of the model, the learning phase, and testing production
#Laura Boekel, July 2013

clearinfo

for x from 1 to 1

Erase all

#parameters for network architecture
spreadingRate = 0.01
activityRangeMin = 0.0
activityRangeMax = 3.0
activityLeak = 1.0
learningRate = 0.001
weightRangeMin = -3.0
weightRangeMax = 3.0
weightLeak = 0.5

#for drawing
xMin = 0
xMax = 30
yMin = 0
yMax = 30

Create empty Network... test'x' spreadingRate linear activityRangeMin
...activityRangeMax activityLeak learningRate weightRangeMin
...weightRangeMax weightLeak xMin xMax yMin yMax

#specify number of nodes in each layer
inputNodes = 60
middleNodes = 20
outputNodes = 20

#add nodes in three layers
for i to inputNodes
        Add node... (((xMax)-(xMin+1))/inputNodes)*i yMin+1 0.0 0
endfor


for i to middleNodes
        Add node... (((xMax-1)-(xMin+1))/middleNodes)*i yMax/2 0.0 0
endfor

for i to outputNodes
        if i < 5
                Add node... (((xMax-1)-(xMin+1))/outputNodes)*i yMax-3 0.0 0
        elsif i < 9
                Add node... (((xMax-1)-(xMin+1))/outputNodes)*i yMax-4 0.0 0
        elsif i < 13
                Add node... (((xMax-1)-(xMin+1))/outputNodes)*i yMax-5 0.0 0
        elsif i < 17
                Add node... (((xMax-1)-(xMin+1))/outputNodes)*i yMax-4 0.0 0
        elsif i < 21
                Add node... (((xMax-1)-(xMin+1))/outputNodes)*i yMax-3 0.0 0
        endif
endfor

Draw... 0

#add connections between nodes in different layers
for i to inputNodes
        for j to middleNodes
                Add connection... i inputNodes+j randomUniform(0,0.1) 1.0
        endfor
```

```
endfor


for i from inputNodes+1 to inputNodes+middleNodes
            for j from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes
                        Add connection... i j randomUniform(0,0.1) 1.0
            endfor
endfor

#add inhibitory, non-learning connections between nodes in the top layer
for i from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes-1
            for j from i+1 to inputNodes+middleNodes+outputNodes
                        Add connection... i j -0.1 0.0
            endfor
endfor

#add inhibitory, non-learning connections between nodes in the middle layer
for i from inputNodes+1 to inputNodes+middleNodes-1
            for j from i+1 to inputNodes+middleNodes
                        Add connection... i j -0.1 0.0
            endfor
endfor

#add inhibitory, non-learning connections between nodes in the bottom layer
for i from 1 to inputNodes-1
            for j from i+1 to inputNodes
                        Add connection... i j -0.1 0.0
            endfor
endfor

#insert labels above output nodes and below input nodes
Text... 3.75 Right 28 Bottom "I"
Text... 9.5 Right 28 Bottom "E"
Text... 15 Right 28 Bottom "A"
Text... 20.5 Right 28 Bottom "O"
Text... 26.25 Right 28 Bottom "U"

Draw... 0
Save as Windows metafile... results/Nieuw_model/Architecture_NewModel.emf

#mean of vowel frequencies
scaleF1Min = 100
scaleF1Max = 1000

scaleF2Min = 500
scaleF2Max = 3000


meanIF1 = (20/100 * (scaleF1Max-scaleF1Min) + scaleF1Min)/(scaleF2Max/inputNodes)
meanIF2 = (90/100 * (scaleF2Max-scaleF2Min) + scaleF2Min)/(scaleF2Max/inputNodes)

meanEF1 = (50/100 * (scaleF1Max-scaleF1Min) + scaleF1Min)/(scaleF2Max/inputNodes)
meanEF2 = (75/100 * (scaleF2Max-scaleF2Min) + scaleF2Min)/(scaleF2Max/inputNodes)

meanAF1 = (80/100 * (scaleF1Max-scaleF1Min) + scaleF1Min)/(scaleF2Max/inputNodes)
meanAF2 = (50/100 * (scaleF2Max-scaleF2Min) + scaleF2Min)/(scaleF2Max/inputNodes)

meanOF1 = (50/100 * (scaleF1Max-scaleF1Min) + scaleF1Min)/(scaleF2Max/inputNodes)
meanOF2 = (25/100 * (scaleF2Max-scaleF2Min) + scaleF2Min)/(scaleF2Max/inputNodes)

meanUF1 = (20/100 * (scaleF1Max-scaleF1Min) + scaleF1Min)/(scaleF2Max/inputNodes)
meanUF2 = (10/100 * (scaleF2Max-scaleF2Min) + scaleF2Min)/(scaleF2Max/inputNodes)

#parameters for Gaussian spreading
peakSharpness = 2
numberOfCategories = 6
stdevOfAmbient = (inputNodes/2 - 1)/peakSharpness/numberOfCategories
```

```
auditorySharpness = 60
auditorySpreading = (inputNodes - 1) / auditorySharpness

#parameters for learning
Set instar... 0.5
Set outstar... 0.5
Set weight leak... 0.5
spreadActivities = 500


#create empty vowel array
for i from 1 to 5
        vowelArray[i] = 0
endfor

#learning phase

totalInputs = 20000

for i from 1 to totalInputs
        vowel = randomInteger(1,5)
        if vowel = 1
                vowelArray[1] += 1
                for j from 1 to inputNodes
                        Set clamping... j 1
                endfor
                for l from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes
                        Set clamping... l 1
                endfor
                audNode1 = randomGauss(meanIF1, stdevOfAmbient)
                audNode2 = randomGauss(meanIF2, stdevOfAmbient)
                for a from 1 to inputNodes
                        Set activity... a exp (-0.5 * (a - audNode1) ^ 2 / auditorySpreading ^ 2) + exp (-0.5 * (a -
audNode2) ^ 2 / auditorySpreading ^ 2)
                endfor
                for m from inputNodes+middleNodes+1 to inputNodes+middleNodes+4
                        Set activity... m 1
                endfor
                Spread activities... spreadActivities
                Update weights
                Zero activities... 1 0
                for j from 1 to inputNodes
                        Set clamping... j 0
                endfor
                for l from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes
                        Set clamping... l 0
                endfor
        elsif vowel = 2
                vowelArray[2] += 1
                for j from 1 to inputNodes
                        Set clamping... j 1
                endfor
                for l from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes
                        Set clamping... l 1
                endfor
                audNode1 = randomGauss(meanEF1, stdevOfAmbient)
                audNode2 = randomGauss(meanEF2, stdevOfAmbient)
                for a from 1 to inputNodes
                        Set activity... a exp (-0.5 * (a - audNode1) ^ 2 / auditorySpreading ^ 2) + exp (-0.5 * (a -
audNode2) ^ 2 / auditorySpreading ^ 2)
                endfor
                for m from inputNodes+middleNodes+5 to inputNodes+middleNodes+8
                        Set activity... m 1
                endfor
                Spread activities... spreadActivities
                Update weights
                Zero activities... 1 0
                for j from 1 to inputNodes
                        Set clamping... j 0
                endfor
                for l from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes
                        Set clamping... l 0
                endfor
        elsif vowel = 3
                vowelArray[3] += 1
                for j from 1 to inputNodes
                        Set clamping... j 1
                endfor
                for l from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes
                        Set clamping... l 1
                endfor
                audNode1 = randomGauss(meanAF1, stdevOfAmbient)
                audNode2 = randomGauss(meanAF2, stdevOfAmbient)
                for a from 1 to inputNodes
                        Set activity... a exp (-0.5 * (a - audNode1) ^ 2 / auditorySpreading ^ 2) + exp (-0.5 * (a -
audNode2) ^ 2 / auditorySpreading ^ 2)
                endfor
                for m from inputNodes+middleNodes+9 to inputNodes+middleNodes+12
                        Set activity... m 1
                endfor
                Spread activities... spreadActivities
                Update weights
                Zero activities... 1 0
                for j from 1 to inputNodes
                        Set clamping... j 0
                endfor
                for l from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes
                        Set clamping... l 0
                endfor
        elsif vowel = 4
                vowelArray[4] += 1
                for j from 1 to inputNodes
                        Set clamping... j 1
                endfor
                for l from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes
                        Set clamping... l 1
                endfor
                audNode1 = randomGauss(meanOF1, stdevOfAmbient)
                audNode2 = randomGauss(meanOF2, stdevOfAmbient)
                for a from 1 to inputNodes
                        Set activity... a exp (-0.5 * (a - audNode1) ^ 2 / auditorySpreading ^ 2) + exp (-0.5 * (a -
audNode2) ^ 2 / auditorySpreading ^ 2)
                endfor
                for m from inputNodes+middleNodes+13 to inputNodes+middleNodes+16
                        Set activity... m 1
                endfor
                Spread activities... spreadActivities
                Update weights
                Zero activities... 1 0
                for j from 1 to inputNodes
                        Set clamping... j 0
                endfor
                for l from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes
                        Set clamping... l 0
                endfor
        elsif vowel = 5
                vowelArray[5] += 1
                for j from 1 to inputNodes
                        Set clamping... j 1
                endfor
                for l from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes
                        Set clamping... l 1
                endfor
                audNode1 = randomGauss(meanUF1, stdevOfAmbient)
```

```
                    audNode2 = randomGauss(meanUF2, stdevOfAmbient)
                    for a from 1 to inputNodes
                              Set activity... a exp (-0.5 * (a - audNode1) ^ 2 / auditorySpreading ^ 2) + exp (-0.5 * (a -
audNode2) ^ 2 / auditorySpreading ^ 2)
                    endfor
                    for m from inputNodes+middleNodes+17 to inputNodes+middleNodes+outputNodes
                              Set activity... m 1
                    endfor
                    Spread activities... spreadActivities
                    Update weights
                    Zero activities... 1 0
                    for j from 1 to inputNodes
                              Set clamping... j 0
                    endfor
                    for l from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes
                              Set clamping... l 0
                    endfor
          endif
endfor

#to check the number of times each vowel has been drawn
printline The vowel I has been drawn 'vowelArray[1]' times in cycle 'x'
printline The vowel E has been drawn 'vowelArray[2]' times in cycle 'x'
printline The vowel A has been drawn 'vowelArray[3]' times in cycle 'x'
printline The vowel O has been drawn 'vowelArray[4]' times in cycle 'x'
printline The vowel U has been drawn 'vowelArray[5]' times in cycle 'x'
printline


#Testing the model (production – not in steps) and saving pictures
for i from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes
          Set clamping... i 1
endfor
for j from inputNodes+middleNodes+1 to inputNodes+middleNodes+4
          Set activity... j 1
endfor
Spread activities... spreadActivities
Erase all
#insert labels above output nodes and below input nodes
Text... 3.75 Right 28 Bottom "I"
Text... 9.5 Right 28 Bottom "E"
Text... 15 Right 28 Bottom "A"
Text... 20.5 Right 28 Bottom "O"
Text... 26.25 Right 28 Bottom "U"
Draw... 0
Save as Windows metafile... results/Nieuw_model/'x'_I_NewModel.emf
Zero activities... 1 0

for i from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes
          Set clamping... i 1
endfor
for j from inputNodes+middleNodes+5 to inputNodes+middleNodes+8
          Set activity... j 1
endfor
Spread activities... spreadActivities
Erase all
#insert labels above output nodes and below input nodes
Text... 3.75 Right 28 Bottom "I"
Text... 9.5 Right 28 Bottom "E"
Text... 15 Right 28 Bottom "A"
Text... 20.5 Right 28 Bottom "O"
Text... 26.25 Right 28 Bottom "U"
Draw... 0
Save as Windows metafile... results/Nieuw_model/'x'_E_NewModel.emf
Zero activities... 1 0

for i from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes
```

```
        Set clamping... i 1
endfor
for j from inputNodes+middleNodes+9 to inputNodes+middleNodes+12
        Set activity... j 1
endfor
Spread activities... spreadActivities
Erase all
#insert labels above output nodes and below input nodes
Text... 3.75 Right 28 Bottom "I"
Text... 9.5 Right 28 Bottom "E"
Text... 15 Right 28 Bottom "A"
Text... 20.5 Right 28 Bottom "O"
Text... 26.25 Right 28 Bottom "U"
Draw... 0
Save as Windows metafile... results/Nieuw_model/'x'_A_NewModel.emf
Zero activities... 1 0

for i from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes
        Set clamping... i 1
endfor
for j from inputNodes+middleNodes+13 to inputNodes+middleNodes+16
        Set activity... j 1
endfor
Spread activities... spreadActivities
Erase all
#insert labels above output nodes and below input nodes
Text... 3.75 Right 28 Bottom "I"
Text... 9.5 Right 28 Bottom "E"
Text... 15 Right 28 Bottom "A"
Text... 20.5 Right 28 Bottom "O"
Text... 26.25 Right 28 Bottom "U"
Draw... 0
Save as Windows metafile... results/Nieuw_model/'x'_O_NewModel.emf
Zero activities... 1 0

for i from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes
        Set clamping... i 1
endfor
for j from inputNodes+middleNodes+17 to inputNodes+middleNodes+outputNodes
        Set activity... j 1
endfor
Spread activities... spreadActivities
Erase all
#insert labels above output nodes and below input nodes
Text... 3.75 Right 28 Bottom "I"
Text... 9.5 Right 28 Bottom "E"
Text... 15 Right 28 Bottom "A"
Text... 20.5 Right 28 Bottom "O"
Text... 26.25 Right 28 Bottom "U"
Draw... 0
Save as Windows metafile... results/Nieuw_model/'x'_U_NewModel.emf
Zero activities... 1 0

endfor
```

Appendix B: Restricting the activation spreading when testing production

```
#Testing the model (production - in steps) and saving pictures
#Laura Boekel, July 2013


#To be used after the learning phase (See Appendix A for script).


#Production: I

for i from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes
        Set clamping... i 1
endfor
for i from 1 to inputNodes
        Set clamping... i 1
endfor
for i from inputNodes+middleNodes+1 to inputNodes+middleNodes+4
        Set activity... i 1
endfor
Spread activities... spreadActivities
Erase all
#insert labels above output nodes and below input nodes
Text... 3.75 Right 28 Bottom "I"
Text... 9.5 Right 28 Bottom "E"
Text... 15 Right 28 Bottom "A"
Text... 20.5 Right 28 Bottom "O"
Text... 26.25 Right 28 Bottom "U"
Draw... 0
Save as Windows metafile... results/Nieuw_model/'x'_I_NewModel_step1.emf
for i from inputNodes+1 to inputNodes+middleNodes
        Set clamping... i 1
endfor
for i from 1 to inputNodes
        Set clamping... i 0
endfor
Spread activities... spreadActivities
Erase all
#insert labels above output nodes and below input nodes
Text... 3.75 Right 28 Bottom "I"
Text... 9.5 Right 28 Bottom "E"
Text... 15 Right 28 Bottom "A"
Text... 20.5 Right 28 Bottom "O"
Text... 26.25 Right 28 Bottom "U"
Draw... 0
Save as Windows metafile... results/Nieuw_model/'x'_I_NewModel_step2.emf
Zero activities... 1 0
for i from inputNodes+1 to inputNodes+middleNodes
        Set clamping... i 0
endfor
for i from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes
        Set clamping... i 0
endfor


#Production: E

for i from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes
        Set clamping... i 1
endfor
for i from 1 to inputNodes
        Set clamping... i 1
endfor
for i from inputNodes+middleNodes+5 to inputNodes+middleNodes+8
        Set activity... i 1
endfor
Spread activities... spreadActivities
```

```
Erase all
#insert labels above output nodes and below input nodes
Text... 3.75 Right 28 Bottom "I"
Text... 9.5 Right 28 Bottom "E"
Text... 15 Right 28 Bottom "A"
Text... 20.5 Right 28 Bottom "O"
Text... 26.25 Right 28 Bottom "U"
Draw... 0
Save as Windows metafile... results/Nieuw_model/'x'_E_NewModel_step1.emf
for i from inputNodes+1 to inputNodes+middleNodes
        Set clamping... i 1
endfor
for i from 1 to inputNodes
        Set clamping... i 0
endfor
Spread activities... spreadActivities
Erase all
#insert labels above output nodes and below input nodes
Text... 3.75 Right 28 Bottom "I"
Text... 9.5 Right 28 Bottom "E"
Text... 15 Right 28 Bottom "A"
Text... 20.5 Right 28 Bottom "O"
Text... 26.25 Right 28 Bottom "U"
Draw... 0
Save as Windows metafile... results/Nieuw_model/'x'_E_NewModel_step2.emf
Zero activities... 1 0
for i from inputNodes+1 to inputNodes+middleNodes
        Set clamping... i 0
endfor
for i from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes
        Set clamping... i 0
endfor


#Prodcution: A

for i from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes
        Set clamping... i 1
endfor
for i from 1 to inputNodes
        Set clamping... i 1
endfor
for i from inputNodes+middleNodes+9 to inputNodes+middleNodes+12
        Set activity... i 1
endfor
Spread activities... spreadActivities
Erase all
#insert labels above output nodes and below input nodes
Text... 3.75 Right 28 Bottom "I"
Text... 9.5 Right 28 Bottom "E"
Text... 15 Right 28 Bottom "A"
Text... 20.5 Right 28 Bottom "O"
Text... 26.25 Right 28 Bottom "U"
Draw... 0
Save as Windows metafile... results/Nieuw_model/'x'_A_NewModel_step1.emf
for i from inputNodes+1 to inputNodes+middleNodes
        Set clamping... i 1
endfor
for i from 1 to inputNodes
        Set clamping... i 0
endfor
Spread activities... spreadActivities
Erase all
#insert labels above output nodes and below input nodes
Text... 3.75 Right 28 Bottom "I"
Text... 9.5 Right 28 Bottom "E"
Text... 15 Right 28 Bottom "A"
Text... 20.5 Right 28 Bottom "O"
Text... 26.25 Right 28 Bottom "U"
Draw... 0
Save as Windows metafile... results/Nieuw_model/'x'_A_NewModel_step2.emf
Zero activities... 1 0
for i from inputNodes+1 to inputNodes+middleNodes
        Set clamping... i 0
endfor
for i from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes
        Set clamping... i 0
endfor


#Production: O

for i from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes
        Set clamping... i 1
endfor
for i from 1 to inputNodes
        Set clamping... i 1
endfor
for i from inputNodes+middleNodes+13 to inputNodes+middleNodes+16
        Set activity... i 1
endfor
Spread activities... spreadActivities
Erase all
#insert labels above output nodes and below input nodes
Text... 3.75 Right 28 Bottom "I"
Text... 9.5 Right 28 Bottom "E"
Text... 15 Right 28 Bottom "A"
Text... 20.5 Right 28 Bottom "O"
Text... 26.25 Right 28 Bottom "U"
Draw... 0
Save as Windows metafile... results/Nieuw_model/'x'_O_NewModel_step1.emf
for i from inputNodes+1 to inputNodes+middleNodes
        Set clamping... i 1
endfor
for i from 1 to inputNodes
        Set clamping... i 0
endfor
Spread activities... spreadActivities
Erase all
#insert labels above output nodes and below input nodes
Text... 3.75 Right 28 Bottom "I"
Text... 9.5 Right 28 Bottom "E"
Text... 15 Right 28 Bottom "A"
Text... 20.5 Right 28 Bottom "O"
Text... 26.25 Right 28 Bottom "U"
Draw... 0
Save as Windows metafile... results/Nieuw_model/'x'_O_NewModel_step2.emf
Zero activities... 1 0
for i from inputNodes+1 to inputNodes+middleNodes
        Set clamping... i 0
endfor
for i from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes
        Set clamping... i 0
endfor


#Production: U

for i from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes
        Set clamping... i 1
endfor
for i from 1 to inputNodes
        Set clamping... i 1
endfor
```

```
for i from inputNodes+middleNodes+17 to inputNodes+middleNodes+20
		Set activity... i 1
endfor
Spread activities... spreadActivities
Erase all
#insert labels above output nodes and below input nodes
Text... 3.75 Right 28 Bottom "I"
Text... 9.5 Right 28 Bottom "E"
Text... 15 Right 28 Bottom "A"
Text... 20.5 Right 28 Bottom "O"
Text... 26.25 Right 28 Bottom "U"
Draw... 0
Save as Windows metafile... results/Nieuw_model/'x'_U_NewModel_step1.emf
for i from inputNodes+1 to inputNodes+middleNodes
		Set clamping... i 1
endfor
for i from 1 to inputNodes
		Set clamping... i 0
endfor
Spread activities... spreadActivities
Erase all
#insert labels above output nodes and below input nodes
Text... 3.75 Right 28 Bottom "I"
Text... 9.5 Right 28 Bottom "E"
Text... 15 Right 28 Bottom "A"
Text... 20.5 Right 28 Bottom "O"
Text... 26.25 Right 28 Bottom "U"
Draw... 0
Save as Windows metafile... results/Nieuw_model/'x'_U_NewModel_step2.emf
Zero activities... 1 0
for i from inputNodes+1 to inputNodes+middleNodes
		Set clamping... i 0
endfor
for i from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes
		Set clamping... i 0
endfor
```

Appendix C: Weight normalization

```
#Learning Phase with Weight normalization
#Laura Boekel, July 2013


#To be used instead of the #learning in Appendix A.


#learning
totalInputs = 20000

for i from 1 to totalInputs
        vowel = randomInteger(1,5)
        if vowel = 1
                vowelArray[1] += 1
                for j from 1 to inputNodes
                        Set clamping... j 1
                endfor
                for l from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes
                        Set clamping... l 1
                endfor
                audNode1 = randomGauss(meanIF1, stdevOfAmbient)
                audNode2 = randomGauss(meanIF2, stdevOfAmbient)
                for a from 1 to inputNodes
                        Set activity... a exp (-0.5 * (a - audNode1) ^ 2 / auditorySpreading ^ 2) + exp (-0.5 * (a - 
audNode2) ^ 2 / auditorySpreading ^ 2)
                endfor
                for m from inputNodes+middleNodes+1 to inputNodes+middleNodes+4
                        Set activity... m 1
                endfor
                Spread activities... spreadActivities
                Update weights
                Normalize weights... inputNodes+1 inputNodes+middleNodes 1 inputNodes 
sumNormalizeWeights
                Zero activities... 1 0
                for j from 1 to inputNodes
                        Set clamping... j 0
                endfor
                for l from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes
                        Set clamping... l 0
                endfor
        elsif vowel = 2
                vowelArray[2] += 1
                for j from 1 to inputNodes
                        Set clamping... j 1
                endfor
                for l from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes
                        Set clamping... l 1
                endfor
                audNode1 = randomGauss(meanEF1, stdevOfAmbient)
                audNode2 = randomGauss(meanEF2, stdevOfAmbient)
                for a from 1 to inputNodes
                        Set activity... a exp (-0.5 * (a - audNode1) ^ 2 / auditorySpreading ^ 2) + exp (-0.5 * (a - 
audNode2) ^ 2 / auditorySpreading ^ 2)
                endfor
                for m from inputNodes+middleNodes+5 to inputNodes+middleNodes+8
                        Set activity... m 1
                endfor
                Spread activities... spreadActivities
                Update weights
                Normalize weights... inputNodes+1 inputNodes+middleNodes 1 inputNodes 
sumNormalizeWeights
                Zero activities... 1 0
                for j from 1 to inputNodes
                        Set clamping... j 0
                endfor
                for l from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes
                        Set clamping... l 0
                endfor
        elsif vowel = 3
                vowelArray[3] += 1
                for j from 1 to inputNodes
                        Set clamping... j 1
                endfor
                for l from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes
                        Set clamping... l 1
                endfor
                audNode1 = randomGauss(meanAF1, stdevOfAmbient)
                audNode2 = randomGauss(meanAF2, stdevOfAmbient)
                for a from 1 to inputNodes
                        Set activity... a exp (-0.5 * (a - audNode1) ^ 2 / auditorySpreading ^ 2) + exp (-0.5 * (a - 
audNode2) ^ 2 / auditorySpreading ^ 2)
                endfor
                for m from inputNodes+middleNodes+9 to inputNodes+middleNodes+12
                        Set activity... m 1
                endfor
                Spread activities... spreadActivities
                Update weights
                Normalize weights... inputNodes+1 inputNodes+middleNodes 1 inputNodes 
sumNormalizeWeights
                Zero activities... 1 0
                for j from 1 to inputNodes
                        Set clamping... j 0
                endfor
                for l from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes
                        Set clamping... l 0
                endfor
        elsif vowel = 4
                vowelArray[4] += 1
                for j from 1 to inputNodes
                        Set clamping... j 1
                endfor
                for l from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes
                        Set clamping... l 1
                endfor
                audNode1 = randomGauss(meanOF1, stdevOfAmbient)
                audNode2 = randomGauss(meanOF2, stdevOfAmbient)
                for a from 1 to inputNodes
                        Set activity... a exp (-0.5 * (a - audNode1) ^ 2 / auditorySpreading ^ 2) + exp (-0.5 * (a - 
audNode2) ^ 2 / auditorySpreading ^ 2)
                endfor
                for m from inputNodes+middleNodes+13 to inputNodes+middleNodes+16
                        Set activity... m 1
                endfor
                Spread activities... spreadActivities
                Update weights
                Normalize weights... inputNodes+1 inputNodes+middleNodes 1 inputNodes 
sumNormalizeWeights
                Zero activities... 1 0
                for j from 1 to inputNodes
                        Set clamping... j 0
                endfor
                for l from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes
                        Set clamping... l 0
                endfor
        elsif vowel = 5
                vowelArray[5] += 1
                for j from 1 to inputNodes
                        Set clamping... j 1
                endfor
                for l from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes
                        Set clamping... l 1
                endfor
                audNode1 = randomGauss(meanUF1, stdevOfAmbient)
                audNode2 = randomGauss(meanUF2, stdevOfAmbient)
                for a from 1 to inputNodes
                        Set activity... a exp (-0.5 * (a - audNode1) ^ 2 / auditorySpreading ^ 2) + exp (-0.5 * (a - audNode2) ^ 2 / auditorySpreading ^ 2)
                endfor
                for m from inputNodes+middleNodes+17 to inputNodes+middleNodes+outputNodes
                        Set activity... m 1
                endfor
                Spread activities... spreadActivities
                Update weights
                Normalize weights... inputNodes+1 inputNodes+middleNodes 1 inputNodes sumNormalizeWeights
                Zero activities... 1 0
                for j from 1 to inputNodes
                        Set clamping... j 0
                endfor
                for l from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes
                        Set clamping... l 0
                endfor
        endif
endfor
```

Appendix D: Testing perception

```
#Testing perception, with layer-by-layer activity spreading.
#Laura Boekel, July 2013


#To be used after the learning phase (See appendix A for the script).

scaleHzMin = 100
scaleHzMax = 3000
scaleNodesMin = 1
scaleNodesMax = 60
scaleF1Min = 100
scaleF1Max = 1000
scaleF2Min = 500
scaleF2Max = 3000
inputNodes = 60
middleNodes = 20
outputNodes = 20

spreadActivities = 500

Select outer viewport... 0 6 0 5
Black
10
Axes... scaleF2Max scaleF2Min scaleF1Max scaleF1Min
Draw inner box
Marks right every... 1 100 yes yes no
Text top... yes F2 (Hz)
Marks top every... 1 500 yes yes no
Text right... yes F1 (Hz)


for y from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes
        activityArray[y] = 0
endfor

for i from 1 to inputNodes
        Set clamping... i 1
endfor


for i from 1 to 51
        for j from 1 to 46
                #clamp top layer
                for t from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes
                        Set clamping... t 1
                endfor
                #add activity to bottom layer
                a = i*50+450
                b = j*20+80
                c = (a - scaleHzMin) / (scaleHzMax - scaleHzMin) * (scaleNodesMax - scaleNodesMin) +
scaleNodesMin
                d = (b - scaleHzMin) / (scaleHzMax - scaleHzMin) * (scaleNodesMax - scaleNodesMin) +
scaleNodesMin
                Set activity... c 1
                Set activity... d 1
                #spread activity (to middle layer only)
                Spread activities... spreadActivities
                #unclamp top layer
                for t from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes
                        Set clamping... t 0
                endfor
                #clamp middle layer
                for m from inputNodes+1 to inputNodes+middleNodes
                        Set clamping... m 1
                endfor
                #spread activities to top layer
                Spread activities... spreadActivities
                #draw perception graph
                for y from inputNodes+middleNodes+1 to inputNodes+middleNodes+outputNodes
                        activityArray[y] = Get activity... y
                endfor
                #I
                if activityArray[inputNodes+middleNodes+1] > activityArray[inputNodes+middleNodes+5] and
activityArray[inputNodes+middleNodes+1] > activityArray[inputNodes+middleNodes+9] and
activityArray[inputNodes+middleNodes+1] > activityArray[inputNodes+middleNodes+13] and
activityArray[inputNodes+middleNodes+1] > activityArray[inputNodes+middleNodes+17]
                        Colour... red
                        Text... a centre b half i
                endif
                #E
                if activityArray[inputNodes+middleNodes+5] > activityArray[inputNodes+middleNodes+1] and
activityArray[inputNodes+middleNodes+5] > activityArray[inputNodes+middleNodes+9] and
activityArray[inputNodes+middleNodes+5] > activityArray[inputNodes+middleNodes+13] and
activityArray[inputNodes+middleNodes+5] > activityArray[inputNodes+middleNodes+17]
                        Colour... blue
                        Text... a centre b half e
                endif
                #A
                if activityArray[inputNodes+middleNodes+9] > activityArray[inputNodes+middleNodes+1] and
activityArray[inputNodes+middleNodes+9] > activityArray[inputNodes+middleNodes+5] and
activityArray[inputNodes+middleNodes+9] > activityArray[inputNodes+middleNodes+13] and
activityArray[inputNodes+middleNodes+9] > activityArray[inputNodes+middleNodes+17]
                        Colour... black
                        Text... a centre b half a
                endif
                #O
                if activityArray[inputNodes+middleNodes+13] > activityArray[inputNodes+middleNodes+1] and
activityArray[inputNodes+middleNodes+13] > activityArray[inputNodes+middleNodes+5] and
activityArray[inputNodes+middleNodes+13] > activityArray[inputNodes+middleNodes+9] and
activityArray[inputNodes+middleNodes+13] > activityArray[inputNodes+middleNodes+17]
                        Colour... green
                        Text... a centre b half o
                endif
                #U
                if activityArray[inputNodes+middleNodes+17] > activityArray[inputNodes+middleNodes+1] and
activityArray[inputNodes+middleNodes+17] > activityArray[inputNodes+middleNodes+5] and
activityArray[inputNodes+middleNodes+17] > activityArray[inputNodes+middleNodes+9] and
activityArray[inputNodes+middleNodes+17] > activityArray[inputNodes+middleNodes+13]
                        Colour... purple
                        Text... a centre b half u
                endif
                #unclamp middle layer
                for m from inputNodes+1 to inputNodes+middleNodes
                        Set clamping... m 0
                endfor
                #zero activities
                Zero activities... 1 0
        endfor
endfor

Save as Windows metafile... results/Normalize_Weights/'x'_allVowels_PerceptionGraph_Inhib_inSteps.emf
```