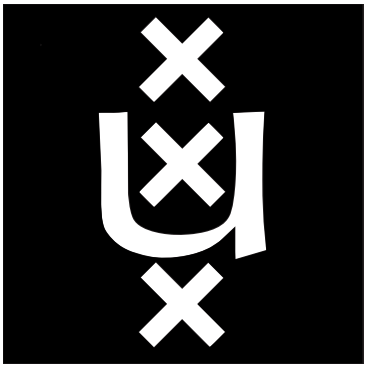
Temporality in bidirectional phonetics and phonology: the STDP neural network model

Angelica van Beemdelust



Master's Thesis
Supervisor: Paul Boersma
Second examiner: Silke Hamann
Student number: 11238720

rMA Linguistics & Communication
University of Amsterdam
the Netherlands
4 July 2022

Temporality in bidirectional phonetics and phonology: the STDP neural network model

Angelica van Beemdelust
University of Amsterdam

ABSTRACT

In this paper, a new type of neural network, the Spike-Timing Dependent Plasticity (STDP) model, is introduced within the framework of Bidirectional Phonology & Phonetics (BiPhon) in the interest of creating a more biologically accurate neural network (NN) that implements temporality through causal relations between nodes. Two STDP models were tested: the first model focused on the functionality of STDP, demonstrating the ability of the network to adjust weights accordingly; the second model was set in the BiPhon-NN framework and was tested on its ability to learn from input to produce the correct output. A simulation consisting of exclusively excitatory connections was unsuccessful in establishing balanced bidirectional weights to produce the correct output. Additionally, a second simulation including excitatory and inhibitory connections was run and elicited similar results to the first. The simplification of the model may be the key driving force behind the lack of bidirectionality of the model. Further improvements to the model can include the implementation of memory leak adjustments, a distinction of high- and low-frequency stimulation, and triplet learning.

Keywords: categories, distributional learning, neural networks, phonology

1 INTRODUCTION

The combination of artificial intelligence and linguistic research aims to describe and explain various phenomena in language using neural networks (NN) and models that replicate what occurs in natural speech. These models account for generalizations found across languages and phenomena that have been discovered through research in psycholinguistics. Focusing on speech perception and production, a bidirectional grammar model of phonetics and phonology (BiPhon) has previously been used to account for experimental and linguistic data (Boersma 2011) and has been combined with NNs (Beemdelust 2020; Boersma 2019; Boersma *et al.* 2020; Seinhorst 2021). Written language is semi-permanent - what is on paper stays on paper - while spoken language is fleeting - in that once words have been said, they are no longer physically available. Hitherto, no NN within BiPhon has realistically implemented the aspect of time, temporality, within speech. One type of NN that does include a temporal aspect and also has been considered a biologically plausible network is that of the spike-timing dependent plasticity (STDP) model (Sjöström and Gerstner 2010). In this paper, it is the aim to use an STDP-model within the framework of BiPhon-NN to create a more biologically accurate NN model. Section 2 provides the background required for understanding the BiPhon grammar model as well as the STDP-model. In section 3, the first of two STDP-models is described and analyzed with the second STDP-model being presented and tested in 4. The final two sections, section 5 and 6 contain the discussion and conclusion respectively.

2 BACKGROUND

The background is divided into three sections: section 2.1 outlines the bidirectional phonology & phonetics framework; section 2.2, contains a short description of a biological neuron to better understand the functionality of Spike-Timing Dependent Plasticity models, and in the final section, section 2.3, this model is further elaborated.

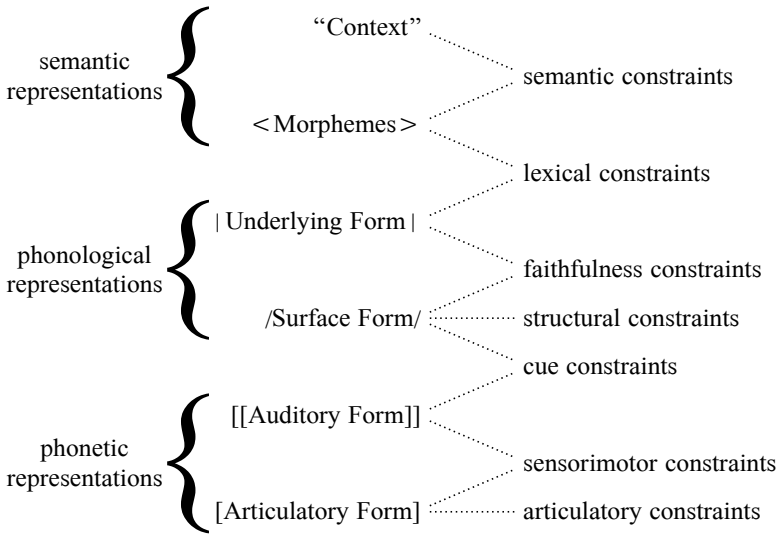


Figure 1: The BiPhon grammar model

Bidirectionality in Phonetics & Phonology 2.1

Within phonetics and phonology, one framework that is used is Bidirectional Phonology & Phonetics (BiPhon) (Boersma 2011). Previously, BiPhon was combined with Optimality Theory (Prince and Smolensky 2004; Boersma 2011) and became known as BiPhon-OT, but recently BiPhon has also gained traction using neural networks (Beemdelust 2020; Boersma 2019; Boersma *et al.* 2020; Seinhorst 2021), known as BiPhon-NN. The grammatical model for both BiPhon-OT and BiPhon-NN is the same, as seen in Figure 1.

As the name suggests, this model takes both a top-down and bottom-up approach; meaning, it is not only a grammar model but also a model of processing. Language production takes a top-down form, starting with the context moving down to the pronunciation known as the articulatory form, while comprehension is a bottom-up process which starts from the listener at the auditory form and moves up through the model to the context. Both the speaker and the listener go through intermediate representations (morphemes, underlying form, and surface form), using the same grammatical system rather than two different paths.

In BiPhon-OT, various constraints are ranked to reach the correct output (production) or input (perception). These ranked constraints

represent the knowledge and grammar of the speaker or listener. However, this framework is limited in multiple ways. Firstly, OT requires discrete categories, which are problematic in emergentist models in which category creation is gradual without hard boundaries between different categories (Boersma *et al.* 2020). Secondly, OT is biologically implausible for it contains a virtually infinite list of candidates to evaluate from.

A more biologically plausible model would be BiPhon-NN. BiPhon-NN models are able to model both directions (top-down and bottom-up) simultaneously. It has shown category formation (Guenther and Gjaja 1996; Boersma 2019) and is based on the biological neurons in a brain. While in BiPhon-OT grammar is represented in ranked constraints, in BiPhon-NN, grammatical knowledge can be found in a form of long-term memory consisting of connections, known as weights, between representations of neurons, known as nodes. The weights between different nodes are varied to be successful in the production or perception of language. Referring back to figure 1, each level of representation can be imagined as a large group of neurons and the connections between each level as the network weights. Each node in a NN is either more or less active and, depending on the configuration of the neuron activity, the connections grow stronger or weaker.

In previous models of BiPhon-NN (Beemdelust 2020; Boersma 2019; Boersma *et al.* 2020; Seinhorst 2021), neuron activity was often not represented as a binary function being either active or inactive as they are in brain cells; but instead, nodes were more active or less active to account for a temporal view in which a higher activity of a node represents a neuron firing more frequently over a period of time than those with less activity. Secondly, in order for connections between nodes in NNs to grow stronger, a method known as Hebbian learning can applied in which the weight increases when both nodes are active (Hebb 1949). However, more precisely, for biological neural connections to actually grow stronger, neuron A fires slightly before neuron B repeatedly, which results in a causal relationship between the two neurons. So, for more biologically accurate NNs, an aspect of causality needs to occur: node A must fire shortly before node B to increase the connection from A to B. Hebb (1949) described this change in biological neurons as follows (p. 335):

> *"When an axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."*

Biological neuron 2.2

To understand the rationale behind this statement by Hebb, an understanding of neuron functionality is required. A neuron consists of a cell body or soma, dendrites, and an axon. The cell body sends out a combination of an electrical and chemical signal through the axon, known as the action potential, to the synapses at the end of the axon. The synapses are connected to dendrites of another neuron, which receive the action potential pulse from the synapses, raising or lowering the voltage, also known as the membrane potential, of the receiving cell. When the action potential increases the membrane potential of the receiving cell, the connection is excitatory and vice versa, when the action potential decreases the membrane potential of the receiving cell, the connection is called inhibitory - both types of connections, inhibitory or excitatory, are possible for a single presynaptic neuron (Strata and Harvey 1999). When the membrane potential crosses a particular threshold, the cell body sends out an action potential and, after a refractory period in which no other action potential can occur, stabilizes to its resting state, known as the resting potential.

The connection Hebb (1949) describes is an excitatory connection where cell A fires, sending out action potentials repeatedly to cell B, resulting in cell B firing as well. A relationship is created between cell A firing and cell B firing, the result of which leads to a strengthened connection from cell A to cell B. In short, if cell A repeatedly causes cell B to fire, the strength of the connection from cell A to cell B increases, which then causes cell B to be more likely to fire again in the future when cell A fires upon cell B.

In order to create a more biologically accurate network, this causal relation between cell A and cell B should be implemented. It is the aim of this paper to take the next step toward a more biologically accurate bidirectional artificial neural network using what is called a spike-timing-dependent plasticity (STDP) model. In section 2.3, this STDP-model is described.

2.3 STDP-model

A model known as the spike-timing-dependent plasticity model (STDP-model) was chosen for this research for its biological plausibility (Sjöström and Gerstner 2010). This model requires a minimal temporal difference between one node firing to another node in order to strengthen or weaken a connection between the nodes as was previously described by Hebb (1949). This temporal effect has been found in various areas of the brain. One such place is the hippocampus (Bliss and Gardner-Medwin 1973; Bliss and Lømo 1973), which plays an important role in short-term and long-term memory (Caporale and Dan 1973). The strengthening of synapses as a result of this temporal shift is known as long-term potentiation (LTP). Conversely, synaptic connections may also weaken, the process of which is known as long-term depression (LTD). Combining LTP and LTD leads to a bidirectional modification of weights on the condition of the neural spikes occurring in short intervals one after another. For LTP to occur, first the presynaptic neuron spikes, followed by the postsynaptic neuron spiking within the next 20 ms (Bi and Poo 1998). For LTD to occur, the exact reverse is true: the postsynaptic neuron spikes before the presynaptic neuron (Bi and Poo 1998). In short, the plasticity between neurons is dependent on the timing of the spike, hence the name spike-timing-dependent plasticity.

STDP-models can be of varying complexity by adjusting the parameters of the model. First, the amount of modification of the weight between nodes is dependent on the temporal difference between the presynaptic and postsynaptic node firing, similar to short-term memory only being able to retain information for a short period of time. If a neuron fired shortly after another one had fired, then the LTD and LTP are stronger than if the interval between each spike is extended. To what degree this period between firing nodes plays a role is variable even within various areas of the brain (Abbott and Nelson 2000). For example, it is possible that the amount of modification of the weight is the same for both LTD and LTP, as shown in Figure 2.

Figure 2 shows LTP on the left side of the y-axis and LTD on the right side of the y-axis. The x-axis is the interval of a presynaptic node firing t^{pre} and a post-synaptic node firing t^{post} in ms, the y-axis shows

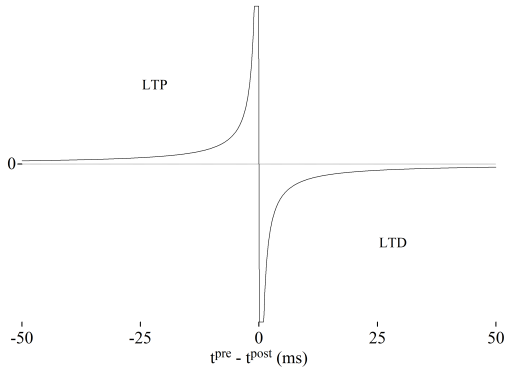


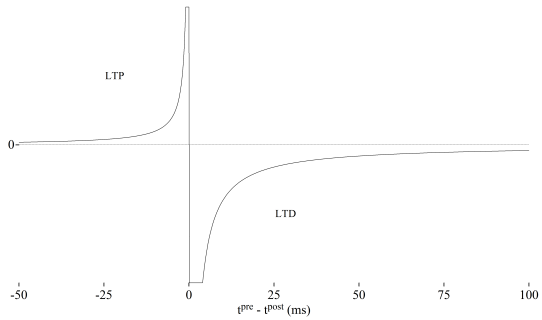
Figure 2:
The amount of modification of synaptic weights as a result of pair-based firing. Both LTP and LTD are modified an equal amount.

how much the weight strengthens or weakens (the y-axis is not quantified here for visualization purposes). Numerical labels are approximations used for scale visualization. In Figure 2, there is no difference between LTP and LTD; for example, weights receive the same LTP or LTD effect at -5 and 5 ms respectively.

However, it is possible that a weight weakens more easily than it strengthens. For example, when a postsynaptic node spikes 25 ms after a presynaptic node has spiked, the LTP may be quite weak. When the roles are reversed, however, and the postsynaptic node spikes 25 ms before the presynaptic node does, the LTD may still be relatively strong. In this situation, there is an asymmetry in LTP and LTD effectiveness in relation to the temporal offset. The visualization of this effect is presented in Figure 3, where the x-axis is the time difference between spikes in ms and the y-axis shows the modification asymmetry. More types of asymmetry than the one presented here have been found (Caporale and Dan 1973) and variations of asymmetry may lead to different results. In the model explored in this thesis, a symmetrical approach was taken.

A second possible complexity is triplet learning. Triplet learning uses three spikes (triplets) instead of two (pairs) to induce LTP or LTD (Pfister and Gerstner 2005; Gjorgjieva *et al.* 2011). In the model of Gjorgjieva *et al.* (2011), the LTP is dependent on the timing difference between the pre- and postsynaptic spikes and on the time the last postsynaptic spike occurred. Similarly to the asymmetry in weight modification mentioned above, triplets may also have an effect on the

Figure 3:
The amount of modification of synaptic weights as a result of pair-based firing. The LTP effect is smaller than the LTD effect.



network.

Finally, a third complexity is the difference in input frequencies. It has been found that LTP occurs with high-frequency stimulation (HFS) in presynaptic weights or with low-frequency stimulation (LFS) with a large change of membrane potential of the postsynaptic neuron (postsynaptic depolarization), whereas LTD may occur after LFS on its own or with a small change of membrane potential (Caporale and Dan 1973). The type of stimulation adds another level of complexity that may influence the results in an STDP model.

To simplify the creation of the neural network, the three complexities described here have been excluded from this thesis. Two STDP-models were made in Praat (Boersma and Weenink 2022) and are described in the following sections. The first STDP-model learns from internal structures but does not allow for input or output and is described further in section 3. In section 4, a second STDP-model is presented in which input and output correlating to the BiPhon grammar model is included to test bidirectionality in a time-sensitive neural network model.

3 STDP MODEL 1

The first model is an STDP-model without external input or output. It is a collection of nodes that can fire and adjust the weight between connected nodes as a result of stochastic internal activation. The network structure is described in section 3.1. The initialization phase,

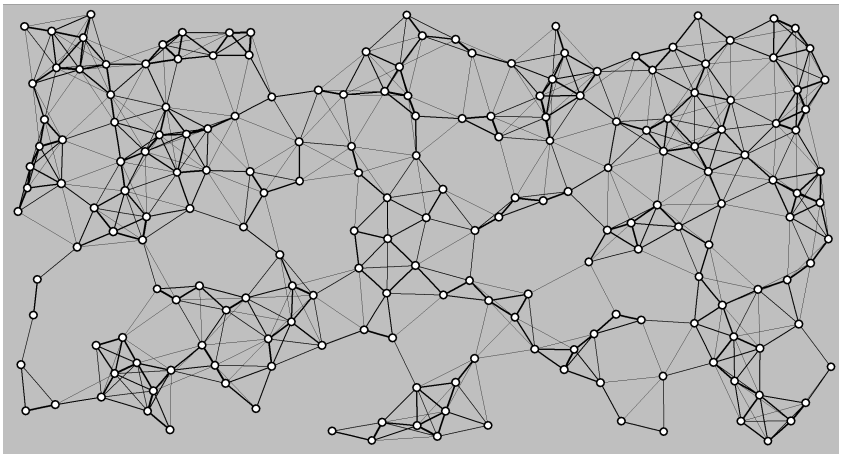


Figure 4: STDP-model 1 at time 0

dreaming phase and learning phase of this model are explained and demonstrated in subsections 3.1.1, 3.1.2, and 3.1.3, respectively. The final section, 3.1.4, contains the analysis of the simulation.

Network structure 3.1

The first model created to test the STDP-model is shown in Figure 4. This figure shows a semi-stochastic distribution of nodes. Each white circle represents a non-active node and each line between nodes is a pair of weights. In this model, there are 200 nodes in total, with initial connections being determined by the distance between the nodes.

Initialization phase 3.1.1

For the initialization phase, the placement of the nodes and the strength of the weights between nodes need to be determined. The placement of nodes is semi-stochastic in that the nodes are placed within the frame in random locations as long as no node is in the same position as, or too close to another node. The minimum distance d_{min} between each node is 7.5 mm as to avoid overlap and the weight between each node linearly correlates to the distance between the nodes. Therefore, the closer two nodes are to each other, the stronger the weights between the nodes. The maximum distance, d_{max}, between each node in which a connection remains present is 25.0 mm; anything above this distance is not connected. The weights between nodes

are expressed in Hertz, with the minimum weight, w_{min}, at maximum distance, d_{max}, set to 0.001 Hz and the maximum weight, w_{max}, at minimum distance, d_{min}, set to 60 Hz. The initial weights between each node are calculated as follows (1)[1]:

$$(1) \qquad w_{ij}(0) = \left(w_{min} + (w_{max} - w_{min}) \frac{d_{max} - d_{ij}}{d_{max} - d_{min}} \right) c_{ij}$$

where i indicates the presynaptic node that connects to postsynaptic node j, d_{ij} is the distance in millimeters between nodes i and j, and c_{ij} indicates whether there is a connection between nodes i and j. c_{ij} is 1 when a connection is present between node i and 0 when there is no connection.

For simplification of the model, the membrane potential of a neuron is not represented in microvolts, but is expressed in Hertz instead. By expressing the membrane potential in Hertz, the membrane potential can be correlated to a rate at which a neuron fires. Each node is set at a resting firing rate, f_{min}, which is set at 3 Hz, meaning that over the course of a second, the neuron will likely fire 3 times without input from its neighbors. The network has a sampling size of 1000 samples per second, meaning that the network will calculate the possible changes in the network once every millisecond, expressed as *dt*. If a node fires 3 times per second in this network, a node has a 0.3% chance of firing per sample. The stochastic chance of a node firing is calculated using the following equation (2):

$$(2) \qquad a_i(t + dt) = \mathscr{B}(f_i(t + dt)dt)$$

where a_i is the activity of each node i at time t in seconds plus the time change dt in seconds. The time change dt is equal to a single sample of 1 millisecond (ms) or 0.001 seconds. $\mathscr{B}$ denotes a Bernouilli deviate and f_i is the firing rate. When a_i is 1, the node fires and sends out an action potential to every node it is connected to. When a_i is 0, the node does not fire. For example, if f_i at time $(t + dt)$ is equal to 20 Hz, the result within the Bernouilli deviate is $20 \cdot 0.001 = 0.02$, which leads to a 2% chance of a_i being 1, and a 98% chance of a_i being 0.

[1] Equation 1 through 6 were provided by Paul Boersma after several attempts by the author

Dreaming phase 3.1.2

Now that the initialization phase is complete, the model can dream. In the dreaming phase, the network receives no external input and any activity within the network starts from possible activity a_i at time *t*. As stated previously, the resting rate of 3 Hz allows for a 0.3% activation of a node, so any activity within the network during this phase starts from the 0.3% chance of firing. When the presynaptic node fires to the postsynaptic node, the membrane potential of the postsynaptic node, p_j, increases. However, this increase in membrane potential does not necessarily mean that the postsynaptic node *j* will fire; it merely increases the odds of firing by the weight, w_{ij}.

The membrane potential exponentially decreases over time without input as if the potential leaks away. The membrane potential returns to its resting state of 0, leading to the resting firing rate of 3 Hz. The membrane potential is reduced by a factor of *e* after approximately 11 ms (Gjorgjieva *et al.* 2011), expressed as τ_{leak}. This activation spreading is presented in equation (3).

$$p_j(t+dt) = \left(p_j(t) + \sum_i a_i(t) w_{ij}(t)\right) e^{-\frac{dt}{\tau_{leak}}} \tag{3}$$

It is possible that the membrane potential increases significantly within one sample when many presynaptic nodes fire upon the same node *j*. However, a node can only fire a limited number of times within a certain time frame. Fast-spiking neurons in the human neo-cortex are estimated at a maximal firing rate of approximately 190 to 640 Hz (Wang *et al.* 2016) and in our network, the maximum firing rate, f_{max}, is set to 600 Hz. To account for the refractory period of a neuron, a period of time during which the neuron cannot fire, the firing rate and membrane potential reset to the resting rate each time the node fires. Thus, the firing rate f_j is calculated as follows (4):

$$f_j(t+dt) = f_{min} + (f_{max} - f_{min})(1 - a_j(t)) \tanh \frac{p_j(t+dt)}{f_{max}} \tag{4}$$

Due to the possibility of the membrane potential being higher than the firing rate, a soft bound is placed on the firing rate. The membrane

potential can possibly be higher than the firing rate. The hyperbolic tangent *tanh* places a soft bound on the firing rate, but allows for the membrane potential to increase past the maximum firing rate.

3.1.3 Learning within dreaming phase

The model can dream, but in order to learn, the weights between nodes will need to change as a result of LTP and LTD. To know whether a node has fired recently, so that learning may occur, a memory trace, *m*, is required. This trace decays exponentially over a period of 20 ms (Bi and Poo 1998; Song *et al.* 2000). The forgetting cause as well as the learning rate is described in Equation (5):

$$m_i(t+dt) = \begin{cases} 1, & \text{if } a_i(t) = 1 \\ m_i(t)e^{-\frac{dt}{\tau_{hist}}}, & \text{if } a_i(t) = 0 \end{cases} \tag{5}$$

Remembering the last moment of firing, the weight adjustments between nodes can now be calculated in which the memory trace is the learning rate. The equation for learning is shown here (6):

$$w_{ij}(t+dt) = \max(w_{min}, \min(w_{ij}(t) + m_i(t)a_j(t) - m_j(t)a_i(t), w_{max})) \tag{6}$$

To avoid an infinite increase or decrease of weights, a hard boundary is set. In this model, the w_{min} is set at 0.001 Hz, and w_{max} is set at 60 Hz, similarly to the minimum and maximum of the initialization phase. In short, equation (6) states that when node *i* fired before node *j*, the weight w_{ij} increases (LTP), but when node *j* fires before node *i*, weight w_{ij} decreases instead (LTD).

3.1.4 Analysis

To test the model, the network was given one second, or 1000 frames to dream. The results are shown in Figure 5. In this figure, nodes that are colored red are nodes that have an activity of 1, while nodes that are white have an activity of 0. Moreover, the size of a node is representative of its firing rate. The larger the node, the greater the firing rate.

Compared to the network at $t = 0$ shown in Figure 4, a number of weights, represented by the black lines, have changed as a result of

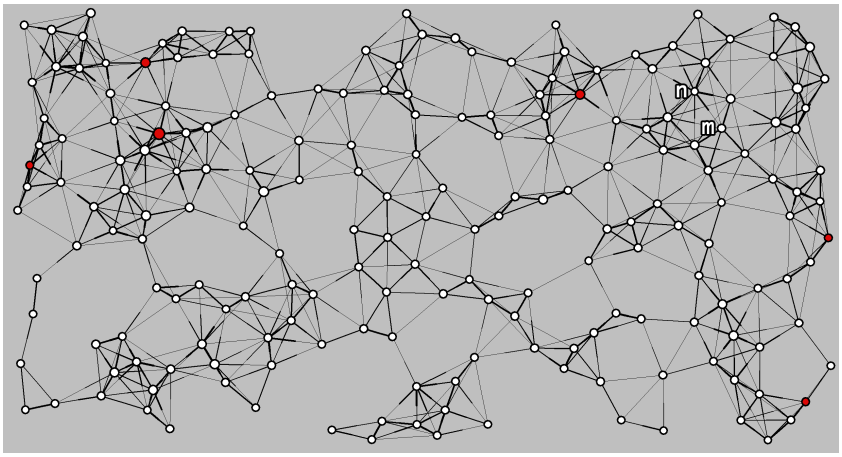


Figure 5: STDP-model 1 after one second of dreaming

the dreaming phase. Closely connected clusters of nodes produced the largest magnitude of weight changes in the network. Central nodes with many connections exhibit thicker lines, stronger weights, toward the central node, whereas the connections exhibit thinner lines, weaker weights, away from the central node. An example of this behavior can be seen in Figure 5 with the connection between node *m* and node *n*. When the presynaptic node *m* fires, the weight to the postsynaptic node *n* is strong, which means node *m* has a strong influence on the membrane potential and subsequent firing rate of node *n*. Conversely, when, what is now a presynaptic node, node *n* fires, the weight to postsynaptic node *m* is weak, which means that node *n* has a weaker influence on the membrane potential and subsequent firing rate of node *m*. In short, when node *m* fires, the membrane potential of node *n* is significantly increased, whereas if node *n* fires, the membrane potential of node *m* is only marginally increased. As a result, a causal relationship between node *n* and node *m* is created. The influence is unidirectional: node *m* likely causes node *n* to fire, but the same cannot be said for the inverse.

The key observation from this STDP network is that the weight of a connection becomes inversely proportional to the firing rate of the presynaptic node; as the firing rate of the presynaptic node increases, the influence on the postsynaptic node decreases. The more frequently a node fires, the smaller the impact of each firing is on the postsynaptic node.

While the network demonstrates its ability to learn from the activity in the dreaming state, there is no external input or output in this version of the model, which means that the network does not learn in a manner that input results in output; instead, the network strengthens or weakens connections. Whether the STDP model is capable of learning from input and producing output from said input is tested in section 4.

4 STDP MODEL 2

This section is structured similarly to section 3. First, the network structure is presented in section 4.1, which includes a description and explanation of the initialization phase (section 4.1.1), the dreaming phase (section 4.1.2) and the learning phase (section 4.1.3). In section 4.2, two different simulations with each two different trials are analyzed. The first simulation includes a network with only excitatory nodes, similarly to STDP model 1. The second simulation includes a network with not only excitatory nodes, but also inhibitory nodes. Both simulations are described in section 4.2.1 and 4.2.2, respectively.

4.1 *Network structure*

To test the bidirectionality of the network as required by the BiPhon-NN model, the neural network contains two explicit representation levels of the grammar model, shown in figure 6. Similarly to previous BiPhon-NN models (Beemdelust (2020); Boersma *et al.* (2020)), starting with the bottom-up processing, this model starts from the input at the auditory form and travels through several layers to the output at the meaning level. Vice versa, using top-down processing, the model starts the input at the semantic representation and travels down towards the output of the articulatory form. The STDP-model used in this thesis is a deep neural network, in which the neural network contains one or multiple hidden layers between the input and output layers. The intermediate steps such as morphemes, underlying form, and

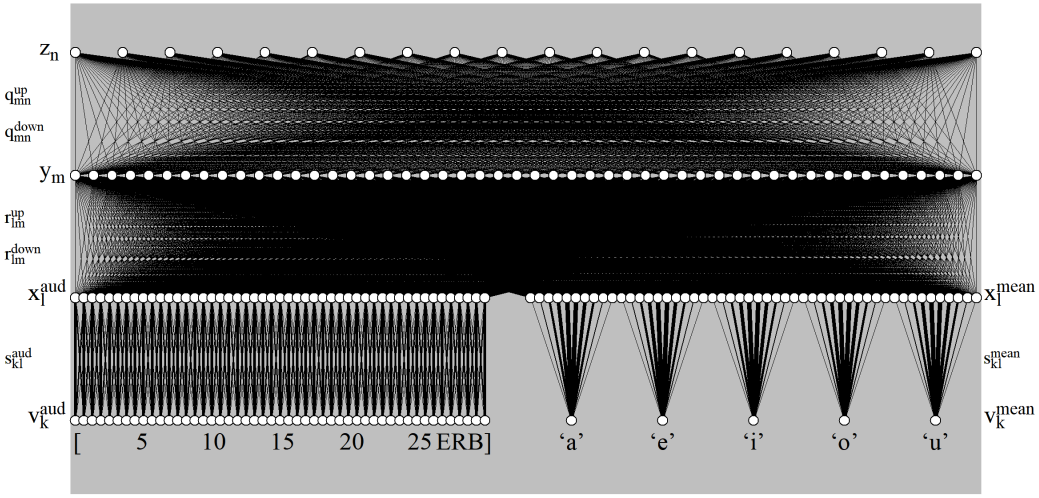


Figure 6: The structure of STDP model 2

surface form, are not explicitly represented in the model, but instead are represented in the hidden layers (see Figure 6).

This network contains 4 levels of nodes: the top level, z, with activities z_n, where n runs from 1 to $N = 20$, is a hidden layer. The second level down, level y, with activities y_m, where m runs from 1 to $M = 50$ and is another, larger hidden layer. The next two levels, level x with activities x_l, and level v, with activities v_k, are separated into two parts. The left part of the network is the auditory representation as indicated by $^{\text{aud}}$, whereas the right part of the network is the meaning representation, $^{\text{mean}}$. The third level down, level x with activities x_l, where l runs from 1 to $L = 99$, consists of 49 nodes on x^{aud} and 50 nodes on x^{mean}. The fourth and final level down, level v with activities v_k, runs from 1 to $K = 54$, where v^{aud} is equal to the number of nodes on x^{aud}, 49 nodes, and v^{mean} consists of the remaining 5 nodes.

Starting with the auditory section of the network, x^{aud} represents a basilar membrane or haircells in the inner ear, which transforms an auditory signal into an electrical one to the brain. x^{aud} is the internal input level for the network that receives external input from level v^{aud}. v^{aud} represents an external input of sound in ERB, Equivalent Rectangular Bandwidth, a measurement used for human auditory perception. v^{aud} is only an external representation and thus will never produce output. This means that without external influence on the

network, the nodes on v^{aud} will never activate through activation of the connected nodes. While the network does not possess an articulatory form, the representation of the basilar membrane x^{aud} may be a simplification of the articulatory output.

Finally, the meaning section of the network consists of levels x^{mean} and v^{mean}. x^{mean} is similar to z and y in that it does not explicitly reflect a particular level in the BiPhon grammar model and instead acts like a hidden layer. x^{mean} is an intermediate step to avoid a significant difference in number of nodes between level y and v^{mean}. As was observed from the first STDP model in section 3.1.4, a large discrepancy in number of nodes may lead to an uneven firing frequency and may undesirably influence the learning phase of the network. Lastly, each of the nodes k on v^{mean} represents the meaning of the vowel written below the node.

While this network has 4 levels of nodes, it also consists of 3 levels of connection weights connecting each level, shown in Figure 6 by the black lines connecting each level. The weights connecting level z and y are collectively called q with weight q_{mn} being the particular weight between node m and node n. As the weights in the STDP-model are asymmetrical by nature of the LTP and LTD, a distinction is made to demonstrate in which direction the node fires. For every node n on z, there is a connection to node m on y called q_{mn}^{down}. In this case, all the nodes n on level z are presynaptic nodes and all the nodes m on level y are postsynaptic nodes. Conversely, for every node m to each node n, the weight is called q_{mn}^{up} where nodes m are presynaptic nodes and nodes n are postsynaptic nodes. Therefore, all nodes are pre- and postsynaptic depending on the direction of the spike from a node. For example, weight $q_{1,3}^{down}$ is the weight from presynaptic node z_3 to postsynaptic node y_1, while $q_{1,3}^{up}$ is from presynaptic node y_1 to postsynaptic node z_3. The same weight symmetry of q_{mn}^{up} and q_{mn}^{down} exists between levels y and x called r_{lm}. r_{lm}^{down} represents all connections from level y to level x, and vice versa, r_{lm}^{up} represents all connections from levels x to y.

The weights between x^{aud} and v^{aud} as well as the weights between x^{mean} and v^{mean} are collectively called s, with s_{kl} being a particular weight between node k and l. Weights s function differently than weights q and r in that weights s do not change and remain clamped

regardless of node activation. The auditory weights s_{aud} are unidirectional from v^{aud} to x^{aud} for the reason that the external auditory input of v^{aud} will never produce output and, thus, will never activate as a result of the activity of the model. Not every node k on level v is connected to every node l on level x. Each node k is connected to a minimum of 3 and a maximum of 5 nearby nodes on x_l, the weights for which are calculated using the probability density function as a model for the distribution of weights as shown in Equation (7).

$$s_{kl} = \frac{w_{max}}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{k-l}{\sigma})^2} \tag{7}$$

where w_{max} is set to 500 Hertz in the initalization phase of this network and σ is equivalent to one node, 1. By using this function, it is possible to approximate the shape of a normal distribution in the set of weights, which represents the resonance of the basilar membrane.

For example, if node $k = 20$ on v^{aud} (v^{aud}_{20}) fires, then node $l = 20$ on x^{aud} (x^{aud}_{20}) receives that signal maximally according to the set maximum weight in the initialization phase, in this case 500 Hz, meaning an actual weight of $\frac{500}{\sqrt{2\pi}}$, or approximately 200 Hz. Node x^{aud}_{19} and node x^{aud}_{20} also receive the signal from v^{aud}_{20} but the weights between these nodes, so weights $s^{aud}_{20,19}$ and $s^{aud}_{20,21}$ are reduced by a factor of $\frac{500}{\sqrt{2\pi}} e^{-\frac{1}{2}}$, approximately 78 Hz. Node $l = 18$ and $l = 22$ on x receive the weakest input as the weights $s^{aud}_{20,18}$ and $s^{aud}_{20,22}$ are reduced by a factor of $\frac{500}{\sqrt{2\pi}} e^{-2}$, approximately 172 Hz. Node $l = 17$ and node $l = 23$ on x are too far away from node $k = 20$ and thus do not connect. For node $k = 1$ and $k = 49$, only 3 nodes on level x are connected, namely $l = 1$, 2 and 3 and $l = 47$, 48 and 49, respectively, as a result of the peripheral position. For nodes $k = 2$ and $k = 48$, the number of connections is increased by one to respectively $l = 1$, 2, 3 and 4, and $l = 46$, 47, 48 and 49.

The weights of s^{mean}_{kl}, as opposed to s^{aud}_{kl}, are not unidirectional, and signals can travel from x^{mean} to v^{mean} and vice versa, from v^{mean} to x^{mean}. Unlike weights q and r, however, the weights are identical in both directions. Thus, a descriptive difference between $s^{mean-up}_{kl}$ and $s^{mean-down}_{kl}$ is redundant and the weights are merely referred to as s^{mean}_{kl}. The weights s^{mean}_{kl} between v^{mean} and x^{mean} are calculated using the same calculation as for the auditory weights, equation 7, with a w_{max}

of 500 Hz at the peak of the bell curve and σ equivalent to two nodes. For each node k on level v^{mean}, there are 10 connections to the nearby nodes l on level x^{mean} and unlike s_{kl}^{aud}, the weights for s_{kl}^{mean} do not overlap, meaning every 10 nodes on level x^{mean} are connected to only one node k on level v^{mean}. Therefore, nodes x_l for l = 50, ..., 9 are connected to v_{50}, nodes x_l for l = 60, ..., 69 are connected to v_{51}, and so on.

4.1.1 Initialization phase

Similarly to STDP model 1, each node is set at a resting firing rate of $f_{min} = 3$ Hz, except for the external input nodes, which do not fire stochastically, similarly to if the resting firing rate would be set to $f_{min} = 0$ Hz. The same equation, equation 1, is used to calculate the probability of firing. For the four sets weights of q and r, each weight is set to 3 Hz as the starting weight, akin to the resting firing rate. As mentioned previously, for s_{kl}^{aud}, the peak is set to $w_{max} = 500$ Hz and for s_{kl}^{mean}, $w_{max} = 500$ Hz.

4.1.2 Dreaming phase

The dreaming phase of STDP model 2 is the same as in STDP model 1. All nodes, except for the external input v_k^{aud}, can activate and may lead to the network dreaming. The calculations for the dreaming phase are the same as described in equations 2 and 3 in section 3.1.2.

4.1.3 Learning phase

As in STDP 1, the second STDP network adjusts weight as a result of a memory trace, m. The same forgetting cause, equation (4), and the same learning equation, equation (5), are used in this model. The STDP model 2 uses different minimum and maximum weights set for each of the two trial simulations that are described in the analysis section of STDP 2, section 4.2. The first simulation (see 4.2.1), which uses only excitatory nodes, has a w_{min} of 0.001 Hz and w_{max} of 60 Hz akin to the hard boundaries in STDP 1. The second simulation (see 4.2.2) worked with both excitatory and inhibitory weights, setting w_{min} to -60 Hz and keeping w_{max} the same at 60 Hz. When introducing the inhibitory nodes, a limit was placed on the firing rate f_i so that the firing rate could never be lower than 0. Various other trials were run

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

Table 1: The mean ERB values per vowel

with different weight boundaries but the results were consistent across all trials.

Now that the network can dream and adjust weights, it is given an input to learn from. During the learning phase, the network hears one of five vowels, /a/, /e/, /i/, /o/ or /u/ randomly selected with an equal probability (0.2). For each vowel, the mean F1 and F2 of a male speaker in ERB are predetermined and shown in Table 1.

From the mean values, an input vowel is sampled, using a Gaussian distribution with a standard deviation of $\sigma = 1$ ERB and a mean of F1 and F2 for the corresponding vowel. The resultant sampled F1 and F2 are then rounded to select and activate a single node each on v_k^{aud} to account for the finite number of nodes as opposed to a more continuous range.

When two nodes on level v^{aud}, representing the F1 and F2 of one of the five vowels, are activated, the node on v_k^{mean} that corresponds to that particular vowel is also activated for a duration of approximately 240 samples, equal to 240 ms, the mean duration of a vowel in American English (House 1961). For example, if the vowel /a/ is activated on level v^{aud}, then 'a' on level v^{mean} is activated at the same time. After that, the network is exposed to another vowel for the same duration. This learning step is repeated 100 to 1000 times. Therefore, 240 samples multiplied by 100 to 1000 repetitions results in approximately 24,000-240,000 samples during which the network is learning, of which one fifth, roughly 4,800-48,000 samples, are for each vowel.

Analysis 4.2

This section is divided into two parts: section 4.2.1 presents a model containing only excitatory weights whereas the model in section 4.2.2 is not restricted to excitatory weights, but also includes inhibitory weights.

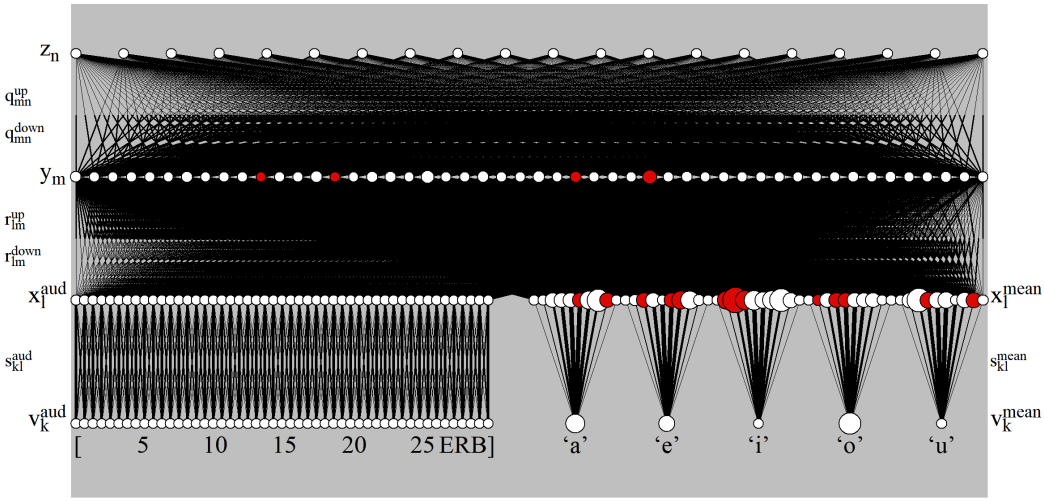


Figure 7: The STDP model with only excitatory nodes after 1000 learning steps.

4.2.1 Excitatory weights only

For this analysis, only excitatory weights were used, as the weight ranged from the minimum weight $w_{min} = 0.001$ Hz to the maximum weight $w_{max} = 60$ Hz, thus keeping the weight ranges comparable to those in STDP model 1. The model was given 1000 learning steps of 240 ms each and the resultant network is presented in Figure 7.

As in Figure 5, in Figure 7 nodes that are red are nodes with an activity of 1, while nodes that are white have an activity of 0. The size of a node demonstrates the firing rate of that node; the larger the node, the greater the firing rate.

In Figure 7, the weights are unevenly distributed from one node to another node, regardless of level. It is most obvious at weights q, as all weights q_{mn}^{down} are strong as demonstrated by the thick lines. On the other hand, all weights q_{mn}^{up} are weak, as presented by the thin lines.

It can be concluded that top level z is, for the most part, minimally influenced by the activity from level y as a result of the weak weights q_{mn}^{up}. The membrane potential and, subsequently, the firing rate of all nodes n on level z do not notably increase. Conversely, nodes m on level y are easily influenced by the nodes on level z as a result of the strong weights q_{mn}^{down}, thus the membrane potential and the firing rate of all nodes m on level y increase more drastically. Based on

input \ output	'a'	'e'	'i'	'o'	'u'
/a/	55	57	61	52	54
/e/	54	47	53	61	63
/i/	56	53	61	56	52
/o/	55	52	57	54	52
/u/	54	52	57	61	55

Table 2: Number of spikes per meaning output (column) for each auditory input (row) in trial 1.

the previous observation from STDP model 1 in section 3.1.4, it can be generalized that nodes *m* fire more frequently than nodes *n*. Such results are to be expected, as level *y* receives more input from level *x* and thus has an increased chance of firing due to the increase in membrane potential.

The weights *r* between levels *y* and *x* are more variable. Nevertheless, a pattern of anti-bidirectional weights arose. Most if not all weights r_{lm}^{down} are weak from y_m to x_l^{aud} whereas weights r_{lm}^{up} are strong from y_m to x_l^{mean}. These weights imply that all nodes *m* on *y* fire more frequently than nodes x_l for l = 1, ..., 49, but less frequently than nodes x_l for l = 50, ..., 99.

It is logical that the firing rate of nodes x_l^{mean} is higher than the firing rate of nodes *m* on level *y* as the meaning input from v_k^{mean} should increase the firing rate of nodes l^{mean} directly, whereas level *y* is not directly connected to the input and relies on x_l^{mean} for input. For x_l^{aud}, the imbalance is flipped because only a select few nodes are activated at a time and do not receive any input from v_k^{aud} unless given external input during the learning phase. This leads to nodes y_m firing more frequently than x_l^{aud}. However, this imbalance of firing rate may affect the network's ability to form more balanced connections.

To test the network, as an auditory input, each vowel was active for 240 ms, during which the spikes of the meaning nodes v_k^{mean} were counted. If the network successfully learned, then a given input /a/ on v_k^{aud} should lead to frequent activation of output node v_{50}, or node 'a' on v_k^{mean} and little activation of the other meaning nodes, 'e', 'i', 'o' or 'u'. Similarly, a given input /i/ on v_k^{aud} should lead to frequent activation of output node v_{52}, node 'i' on v_k^{mean}, and little activation of the remaining meaning nodes. The results are presented in Table 2.

No significant difference was found across any of the corresponding pairs. For example, if the network were to have successfully

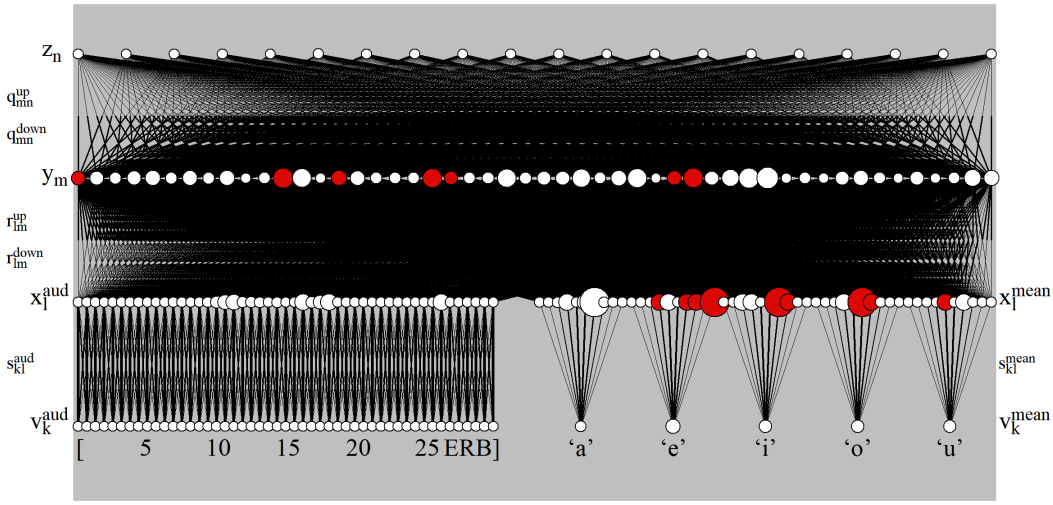


Figure 8: The STDP model with only excitatory nodes after 1000 learning steps with textitw$_{\mathbf{max}}$ of $s_{kl}^{mean} = 200$ Hz.

learned, the counted spikes for input /a/ to output 'a' should be significantly higher than for the other output nodes but Table 2 shows 55 spikes for /a/ to 'a', which is lower than both 'e' and 'i' with 57 and 61, respectively. Similarly, no significant difference was found for the rest of the input and output pairs.

As this trial did not show desired effects, the imbalance in firing rate was accounted for by reducing the maximum weight of the connection s_{kl}^{mean} from 500 Hz to 200 Hz. Increasing the weights for s_{kl}^{aud} produced little to no effect on generating a more balanced division of weights r from and to x_l^{aud} and was thus kept the same. Figure 8 presents the network with the described changes.

In the second trial, from level x_{mean} to level y approximately half of weights r_{lm}^{up} were strong and half of them were weak and the opposite was true for r_{lm}^{down}. Of nodes x_l for $l = 50, ..., 59$, exactly 5 nodes are notably affected by level y due to strong weights of r_{lm}^{down} and 5 nodes are affected little by y due to weak weights of r_{lm}^{down}. However, for nodes x_l for $l = 60, ..., 69$, 6 nodes are strongly affected by y, and 4 nodes are not. For nodes x_l for $l = 70, ..., 79$, this division is 7 to 3, x_l for $l = 80, ..., 89$ is 5 to 5, and finally x_l for $l = 90, ..., 99$ is 4 to 6. Although the division is not perfect, it is far more equal than in the

input \ output	'a'	'e'	'i'	'o'	'u'
/a/	31	40	36	32	26
/e/	30	37	41	33	32
/i/	32	36	44	31	34
/o/	31	34	39	29	29
/u/	30	41	38	34	29

Table 3: Number of spikes per meaning output (column) for each auditory input (row) in trial 2.

previous trial. The same test as in the last trial was run. The results are shown in Table 3.

As in trial 1, no significant differences across any of the corresponding pairs were found. The input /a/ does not lead to an output 'a' as intended. Nevertheless, the results in trial 2 are different from trial 1. The frequency of spikes is reduced in comparison to trial 1, which is a result of the weaker weights on s_{kl}^{mean} . The column for output 'i' shows a greater number of spikes on average (39.6) than the column for output 'u' (30). The reason for this difference in spike frequency is that for nodes $l = 70, ..., 79$, corresponding to the meaning 'i', 7 of the 10 weights r_{lm}^{up} were strong, while for nodes $l = 90, ..., 99$, corresponding to the meaning 'u', only 4 out of the 10 weights r_{lm}^{up} were strong. As a result, the activity of nodes m on level y had a stronger effect on the meaning node corresponding to 'i' than on 'u' due to the weights.

The analysis shows that excitatory weights by themselves are not enough. In the following section, section 4.2.2, inhibitory nodes were added to the network with the intention of the network learning successfully.

Inhibitory & excitatory weights 4.2.2

For this analysis, both excitatory and inhibitory weights were included. The weight ranged from the minimum weight w_{min} = -60.0 Hz and to the maximum weight w_{max} = 60 Hz. As stated previously, the firing rate was limited so that it could never be lower than 0. Various other trials were run with different weight boundaries, but the results were consistent across all trials. As in section 4.2.1, the model was given 1000 learning steps of 240 ms each. Two trials were run; the first one ran with s_{kl}^{mean} = 500 Hz, the second with s_{kl}^{mean} = 200 Hz. The network in the first trial is presented in Figure 9.

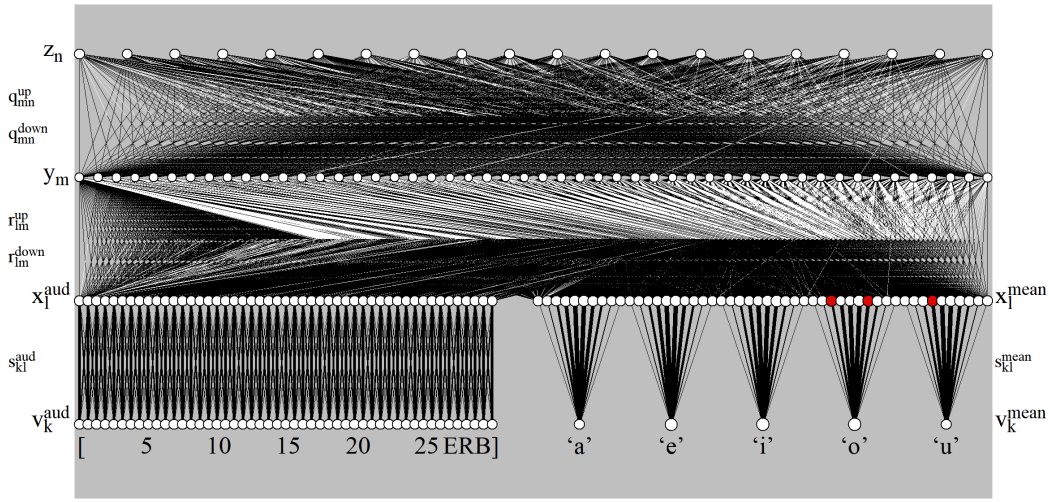


Figure 9: The STDP model with excitatory and inhibitory nodes after 1000 learning steps in trial 1.

Inhibitory weights are represented by white lines, whereas excitatory weights are drawn in black. Starting with weights q, a similar pattern is found as with the excitatory nodes only, although in this case, the weight distribution is not as homogeneous as before. Weights q_{mn}^{down} are mostly excitatory, and weights q_{mn}^{up} are both inhibitory and excitatory but consist of more inhibitory nodes than q_{mn}^{down}. There is also more variability within nodes than in the excitatory model. For example, z_1 consists of both inhibitory and excitatory weights as opposed to solely inhibitory or excitatory weights per node. It can be concluded that the nodes y_m fire more frequently than nodes z_n but the difference between the firing rate from both these levels is not nearly as great as in the excitatory model.

An explanation for this change can be found in weights r. Weights r_{lm}^{up} are mostly inhibitory, most notably from x_l for $l = 50, ..., 99$. The inhibitory weights reduce the firing rate of nodes y_m, which in turn leads to a more even firing rate distribution of nodes y_m and z_n. Weights r_{lm}^{down}, on the other hand, are mostly excitatory. When a node y_m fires, the membrane potential and firing rate of nodes x_l increases. However, it is known from the weights that nodes x_l fire more frequently than nodes y_m, and nodes y_m are subsequently suppressed by

input \ output	'a'	'e'	'i'	'o'	'u'
/a/	21	21	27	14	19
/e/	19	18	29	11	20
/i/	24	27	25	15	19
/o/	14	28	21	14	19
/u/	20	25	12	20	15

Table 4: Number of spikes per meaning output (column) for each auditory input (row) in trial 1.

the reduction in membrane potential. As a result, the connection between the various levels is obstructed. Nevertheless, the same test from the excitatory model was performed to see whether this obstruction affected the firing rate and learnability of input to output. The results are shown in Table 4.

As in the excitatory model, there is no significant connection between the input and output of the network. Input /a/ does not lead to an increased number of spikes of meaning 'a' compared to the other meaning nodes as would be expected in a model that successfully learned. The same lack of correlation is found across the other nodes. The number of spikes within 240 ms in the model of both inhibitory and excitatory nodes is notably fewer, with an average of approximately 20 spikes, than in the excitatory model, with an average of approximately 55 spikes. The explanation for this difference is that nodes y_m hardly fire and thus do not increase the firing rate of nodes x_l. Instead, the spiking activity is mostly a result of x_l^{mean} and v_k^{mean} interacting with one another.

As trial 2 for the excitatory model led to different results than trial 1, a second trial using the same method to balance the firing rate was applied. w_{max} of s_{kl}^{mean} was reduced from 500 Hz to 200 Hz. The resultant network is shown in Figure 10.

For weights q, there is little difference between trial 1 and trial 2. While weights r_{lm}^{up} are still for the majority inhibitory, there is a more even distribution with r_{lm}^{down}. r_{lm}^{down} contains more inhibitory weights than before, but there are still more excitatory weights than inhibitory ones. For the inhibitory nodes, balancing the firing rate of nodes across levels requires a different approach. Once again, an auditory input was given and the number of spikes on the output of the meaning nodes was counted to test whether the network successfully learned with the applied changes. The results are presented in Table 5.

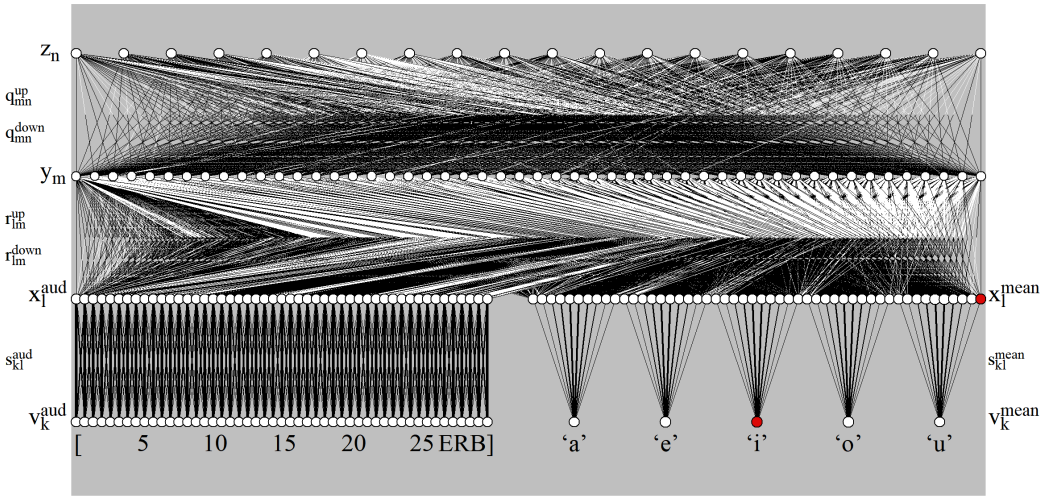


Figure 10: The STDP model with excitatory and inhibitory nodes after 1000 learning steps in trial 2.

Table 5: Number of spikes per meaning output (column) for each auditory input (row) in trial 2.

input \ output	'a'	'e'	'i'	'o'	'u'
/a/	6	11	8	8	10
/e/	6	10	4	8	8
/i/	6	3	6	2	2
/o/	7	8	5	6	2
/u/	4	3	7	5	7

Similar to the previous trials, no correlation was found between the auditory input and meaning output. In comparison to trial 1 of the inhibitory and excitatory network, the number of spikes has decreased. Trial 1 had an average number of approximately 20 spikes in a 240 ms interval, whereas for this trial, the number of spikes averaged approximately 6 per 240 ms. The difference is unsurprising on the grounds that weights s_{kl}^{mean} are weaker in trial 2 and the number of spikes reduces as the influence from one level to another diminishes. Despite the weight changes made in s_{kl}^{mean}, the network was still unable to learn as hoped.

DISCUSSION 5

Two types of STDP models were created. The first model, STDP model 1, tested the network's ability to fire and adjust weights according to the timing between these spikes of pre- and postsynaptic nodes using excitatory connections. This model was successful in that nodes that fired more frequently had a reduced influence on nodes that fired fewer times. Conversely, nodes that fired more sparingly would have a greater effect on nodes that fired more often. Nodes that fired approximately the same number of times would result in balanced weight to and fro. However, this model did not include input or output and any activity occurring in the network was induced through a dreaming state.

The second model, STDP 2, was made to include input and output so that the model could learn from the input to produce the correct output. The input and output consisted of a representation of the basilar membrane that activated nodes based on incoming vowel sounds and meaning representations of the presented vowel sounds. The first simulation using the STDP model 2 was restricted to excitatory weights and in this simulation two trials were run. The first trial showed an imbalance in firing rate as a result of the input during learning process and the given input did not result in a desired output showing successful learning. To account for this imbalance of firing, a second trial was run with reduced weights on the meaning layer. While the firing rates were now relatively equal, the model was

still incapable of learning the correct output given an input. For this reason, a second simulation of this model was performed in which not only excitatory, but also inhibitory weights existed. The first trial of the second simulation demonstrated similar results as the first trial in the first simulation; The difference in firing rate between the input levels and the hidden layers was too great for a balanced weight distribution and, once again, the network did not correlate the input with the correct output. For trial 2, the same weight adjustments were executed. In this simulation, however, the firing rates did not equalize as well as they did in simulation 1. In congruence with the previous trials, this trial, too, led to an unsuccessful network unable to provide the correct output for the input.

In previous BiPhon-NN models, inhibitory and excitatory weights were sufficient for a model to learn properly but the STDP-model is unsuccessful in doing so. This can mean that the STDP-model may require more parameters to function as desired and one such parameter may be a difference in memory leaks. In this study, the memory leak was the same for both LTP and LTD regardless of whether the weight was inhibitory or excitatory. However, a distinction between LTP and LTD, and between inhibitory and excitatory weights has been observed in various studies (Abbott and Nelson 2000; Caporale and Dan 1973; Sjöström and Gerstner 2010) and it may be beneficial to the STDP-network to implement such distinctions. A second parameter may be the type of stimulation. Studies on biological STDP observed a difference in LTP and LTD depending on high- or low-frequency stimulation (Caporale and Dan 1973) and have previously also been implemented in STDP-models (Sjöström and Gerstner 2010). Covering two or even three different types of stimulation in a model may result in a more balanced and accurate network. The final parameter provided here may be a new learning strategy: triplet learning. Triplet learning has led to models that are more accurate to observations from biological neurons than those based purely on pair-based learning (Gjorgjieva *et al.* 2011; Pfister and Gerstner 2005; Sjöström and Gerstner 2010) and may be a suitable parameter to include for future improvements.

While the STDP-model created for this thesis to test BiPhon-NN was unsuccessful in learning, it does not mean the STDP-model is not a viable network for BiPhon-NN. There is a myriad of STDP structures, which may still succeed in demonstrating bidirectionality in phonetics

and phonology and, as such, it may be interesting to further delve into more complex STDP models than the one presented in this paper.

6 CONCLUSION

It was the goal of this thesis to create a more biologically accurate BiPhon-NN model than previous BiPhon-NN models by including a temporal aspect to the network. To do so, the spike-timing dependent plasiticity (STDP) model was selected. Two separate models were made with the first model demonstrated the network's ability to adapt weights from presynaptic to postsynaptic nodes depending on the most recent activity within the network as if the model were dreaming. The second model included a source of input from which the model could learn in unsupervised conditions to produce the appropriate output. Various simulations were run, but in none of the trials was the model able to successfully learn from the input to form the correct output. The anti-bidirectional nature of the STDP-model did not allow for appropriate balancing of weights. It is a possibility that the simplified network model requires more parameters to function as desired. Such parameters may include triplet learning, distinguishing between high- and low frequency stimulation, or varying the leaking time in memory and/or the membrane potential by differentiating between excitatory and inhibitory nodes. However, if these parameters were to be implemented and the STDP-model is still unable to produce output as desired with the given input, then perhaps the STDP-model is not suitable for BiPhon-NN and another model could be implemented to include the aspect of time in BiPhon-NN structures.

REFERENCES

Larry F. Abbott and Sacha B. Nelson (2000), Synaptic plasticity: taming the beast, *Nature Neuroscience*, 3:1178–1183.

Angelica van Beemdelust (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*, Bachelor's thesis, University of Amsterdam.

Guo-qiang Bi and Mu-ming Poo (1998), Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type, *Journal of Neuroscience*, 18(24):10464–10472.

Timothy. V. P. Bliss and Anthony. R. Gardner-Medwin (1973), Long-lasting potentiation of synaptic transmission in the dentate area of the unanaesthetized rabbit following stimulation of the perforant path, *The Journal of Physiology*, 232(2):357–374.

Timothy. V. P. Bliss and Terje Lømo (1973), Long-lasting potentiation of synaptic transmission in the dentate area of the anaesthetized rabbit following stimulation of the perforant path, *The Journal of Physiology*, 232(2):331–356.

Paul Boersma (2011), A programme for bidirectional phonology and phonetics and their acquisition and evolution, in Anton Benz and Jason Mattausch, editors, *Bidirectional Optimality Theory*, pp. 33–72, John Benjamins.

Paul Boersma (2019), Simulated distributional learning in deep Boltzmann machines leads to the emergence of discrete categories, in Sasha Calhoun, Paola Escudero, Marija Tabain, and Paul Warren, editors, *Proceedings of the 19th International Congress of Phonetic Sciences*, pp. 1520–1524.

Paul Boersma, Titia Benders, and Klaas Seinhorst (2020), Neural networks for phonology and phonetics, *Journal of Language Modelling*, 8:103–177.

Paul Boersma and David Weenink (2022), Praat: doing phonetics by computer [Computer program], Version 6.2.14, retrieved 29 May 2022 from `http://www.praat.org/`.

Natalia Caporale and Yang Dan (1973), Spike timing-dependent plasticity: a Hebbian learning rule, *Annual review of neuroscience*, 31:25–46.

Julijana Gjorgjieva, Claudia Clopath, Juliette Audet, and Jean-Pascal Pfister (2011), A triplet spike-timing-dependent plasticity model generalizes the Bienenstock-Cooper-Munro rule to higher-order spatiotemporal correlations, *Proceedings of the National Academy of Sciences*, 108(48):19383–19388.

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

Donald O. Hebb (1949), *The organization of behavior: a neuropsychological theory*, John Wiley & Sons.

Arthur S. House (1961), On vowel duration in english, *Journal of the Acoustical Society of America*, 33:1174–1178.

Jean-pascal PFISTER and Wulfram GERSTNER (2005), Beyond pair-based STDP: a phenomenological rule for spike triplet and frequency effects, in Yair WEISS, Bernhard SCHÖLKOPF, and John C. PLATT, editors, *Advances in neural information processing systems*, volume 18, MIT Press.

Alan PRINCE and Paul SMOLENSKY (2004), *Optimality Theory: constraint interaction in generative grammar*, John Wiley & Sons.

Klaas SEINHORST (2021), *The learnability and complexity of phonological patterns: simulations, experiments, typology*, Ph.d. thesis, University of Amsterdam.

Jesper SJÖSTRÖM and Wulfram GERSTNER (2010), Spike-timing dependent plasticity, *Scholarpedia*, 5(2):1362, revision #184913.

Sen SONG, Kenneth D. MILLER, and Larry F. ABBOTT (2000), Competitive Hebbian learning through spike-timing-dependent synaptic plasticity, *Nature Neuroscience*, 3(9):919–926.

Piergiorgio STRATA and Robin HARVEY (1999), Dale's principle, *Brain Research Bulletin*, 50(5-6):349–350.

Bo WANG, Wei KE, Jing GUANG, Guang CHEN, Luping YIN, Suixin DENG, Quansheng HE, Yaping LIU, Ting HE, Rui ZHENG, Yanbo JIANG, Xiaoxue ZHANG, Tianfu LI, Guoming LUAN, Haidong D. LU, Mingsha ZHANG, Xiaohui ZHANG, and Yousheng SHU (2016), Firing frequency maxima of fast-spiking neurons in human, monkey, and mouse neocortex, *Frontiers in Cellular Neuroscience*, 10(239).

Angelica van Beemdelust (2022), *Temporality in bidirectional phonetics and phonology: the STDP neural network model*, Journal of Language Modelling Template, Master's thesis. 1–31

APPENDICES

Appendix 1: STDP-model 1 script

```
# Praat script STDP_model.praat
# Angelica van Beemdelust
# March-May 2022
# last update: 23-05-2022

demoWindowTitle: "STDP-model"
demoShow()
demo Times
demo Font size: 14

# set up
t = 0
dt = 0.001 ; in s
numberOfNodes = 200
radius = 3.0 ; in mm
minFiringRate = 3 ; in Hertz (So there is at least some
activity, not a dead brain)
minDist = 7.5 ; in mm
maxDist = 25.0 ; in mm
maxFiringRate = 600 ; in Hz
minWeight = 0.001 ; in Hz
maxWeight = 60 ; in Hz
excitatoryDecayTime = 0.011 ; in s
memHistDecayTime = 0.020 ; in s (Bi & Poo, 1998, p. 10464)
excitatoryLeak = exp(-dt/excitatoryDecayTime)
memHistLeak = exp(-dt/memHistDecayTime)
inhibitoryDecayTime = 0.020 ; in s

windowWidth = demo Horizontal world coordinates to mm: 1.0
windowHeight = demo Vertical world coordinates to mm: 1.0

# initialize arrays & matrices
nodeHasFiredArray# = zero# (numberOfNodes)
x_pos# = zero# (numberOfNodes)
y_pos# = zero# (numberOfNodes)
weights## = zero## (numberOfNodes, numberOfNodes)
nodePotentialArray# = zero# (numberOfNodes)
memoryFiringArray# = zero# (numberOfNodes)

@createNodeArray
@createWeightMatrix
@createFireRateArray
@drawInitialWeight

```

```
44  label start
45  demo Erase all
46  demo Select inner viewport: 0, 100, 0, 100
47  demo Axes: 0, windowWidth, 0, windowHeight
48  demo Paint rectangle: "silver", 0, windowWidth, 0,
    windowHeight
49  t += dt
50  @learnAndDrawWeight
51
52  # update firing rate + decay at beginning of cycle
53  for nn to numberOfNodes
54   if nodeHasFiredArray# [nn]
55    nodePotentialArray# [nn] = minFiringRate
56   else
57    sumOfi = 0
58    for i to numberOfNodes
59     if nn <> i
60      sumOfi += nodeHasFiredArray# [i] * weights## [i, nn]
61     endif
62    endfor
63    nodePotentialArray# [nn] = (nodePotentialArray# [nn] +
      sumOfi) * excitatoryLeak ; (formules: 15 -activation
      spreading)
64   endif
65  endfor
66
67  # check if fired and draw nodes
68  for nn to numberOfNodes
69   nodeHasFiredArray# [nn] = randomBernoulli((minFiringRate +
     (maxFiringRate - minFiringRate) * tanh(nodePotentialArray#
     [nn]/maxFiringRate) * (1-nodeHasFiredArray# [nn])) * dt) ;
     (formules: sampling 16 & 17)
70   @updateMemLeak: nn
71   memoryFiringArray# [nn] = mem_n
72   @drawNode: x_pos# [nn], y_pos#[nn], nodeHasFiredArray# [nn],
     nodePotentialArray# [nn]
73  endfor
74
75  demoPeekInput ()
76  if demoKey$ () = "q"
77   exitScript ()
78  endif
79  if demoKey$ () = "f"
80   nodePotentialArray# [25] = 10000
81  endif
82
83  if demoKey$ () = "p"
84   wait = 1
85   while wait = 1
```

```
 86     demoWaitForInput()
 87     if demoKey$ () = "i"
 88       appendInfoLine: t
 89     elsif demoKey$ () = "p"
 90       wait = 0
 91     endif
 92   endwhile
 93 endif
 94
 95 goto start
 96
 97 # procedure drawing nodes
 98 procedure drawNode (.x, .y, .active, .p)
 99   demo Paint circle (mm): if .active then "red" else "white"
      fi, .x, .y, .p/100 + 2*radius
100   demo Line width: 5.0
101   demo Draw circle (mm): .x, .y, .p/100 + 2*radius
102 endproc
103
104 # procedure drawing weights and arrows
105 procedure drawInitialWeight
106   i = 1
107   for i to numberOfNodes - 1
108     j = i + 1
109     for j to numberOfNodes
110       midx = (x_pos# [i] + x_pos# [j])/2
111       midy = (y_pos# [i] + y_pos# [j])/2
112       @getDistance: x_pos# [i], y_pos# [i], x_pos# [j], y_pos#
          [j]
113       if dist < maxDist and dist > minDist
114         demo Line width: weights## [i, j]/10
115         demo Draw line: midx, midy, x_pos# [j], y_pos# [j]
116         demo Line width: weights## [j, i]/10
117         demo Draw line: midx, midy, x_pos# [i], y_pos# [i]
118         # demo Arrow size: max (abs (weights## [i, j])/20, 1)
119         # demo Draw arrow: midx, midy, (midx + x_pos# [j]) / 2,
            (midy + y_pos# [j])/2
120       endif
121     endfor
122   endfor
123 endproc
124
125 # procedure learning and drawing weight
126 procedure learnAndDrawWeight
127   i = 1
128   for i to numberOfNodes - 1
129     j = i + 1
130     for j to numberOfNodes
131       if weights## [i,j] <> 0
```

```
132       midx = (x_pos# [i] + x_pos# [j])/2
133       midy = (y_pos# [i] + y_pos# [j])/2
134       @weightLearning: i, j
135       @weightLearning: j, i
136       demo Line width: weights## [i, j]/10
137       demo Draw line: midx, midy, x_pos# [j], y_pos# [j]
138       demo Line width: weights## [j, i]/10
139       demo Draw line: midx, midy, x_pos# [i], y_pos# [i]
140       # demo Arrow size: max (abs (weights## [i, j])/20, 1)
141       # demo Draw arrow: midx, midy, (midx+x_pos# [j])/2,
          (midy+y_pos# [j])/2
142       # demo Arrow size: max (abs (weights## [j, i])/20, 1)
143       # demo Draw arrow: midx, midy, (midx+x_pos# [i])/2,
          (midy+y_pos# [i])/2
144     endif
145    endfor
146  endfor
147 endproc
148
149 # procedure create node array
150 procedure createNodeArray
151   for node to numberOfNodes
152     repeat
153       x_pos# [node] = randomUniform (radius, windowWidth -
          radius)
154       y_pos# [node] = randomUniform (radius, windowHeight -
          radius)
155       fits = 1
156       for earlierNode to node - 1
157         @getDistance: x_pos# [node], y_pos# [node], x_pos#
          [earlierNode], y_pos# [earlierNode]
158         fits = ( fits and dist >= minDist)
159       endfor
160     until fits
161   endfor
162 endproc
163
164 # procedure creating weight matrix
165 procedure createWeightMatrix
166   i = 1
167   for i to numberOfNodes - 1
168     j = i + 1
169     for j to numberOfNodes
170       @getDistance: x_pos# [i], y_pos# [i], x_pos# [j], y_pos#
          [j]
171       if dist <= maxDist
172         weights## [i, j] = (maxWeight - minWeight) / (minDist -
          maxDist) * (dist - maxDist) + minWeight ; (formules: 10
          - connection weights)
```

```
173       weights## [j, i] = (maxWeight - minWeight) / (minDist -
          maxDist) * (dist - maxDist) + minWeight
174      endif
175    endfor
176  endfor
177 endproc
178 
179 # procedure get distance between two nodes
180 procedure getDistance (.x1, .y1, .x2, .y2)
181  dist = sqrt((.x2-.x1)^2 + (.y2 - .y1)^2)
182 endproc
183 
184 # procedure creating firing rate array
185 procedure createFireRateArray
186  for i to numberOfNodes
187    nodePotentialArray# [i] = minFiringRate
188  endfor
189 endproc
190 
191 # procedure forgetting causes (formules: 19 - forgetting
    causes)
192 procedure updateMemLeak (.nn)
193  if nodeHasFiredArray# [.nn]
194    mem_n = 1
195  else
196    mem_n = memoryFiringArray# [.nn] * memHistLeak
197  endif
198 endproc
199 
200 # procedure learning through weights
201 procedure weightLearning (.pre_nn, .post_nn)
202  weights## [.pre_nn, .post_nn] = max(minWeight, min(weights##
     [.pre_nn, .post_nn] + memoryFiringArray# [.pre_nn] *
     nodeHasFiredArray# [.post_nn] - nodeHasFiredArray#
     [.pre_nn]*memoryFiringArray# [.post_nn], maxWeight))
203 endproc
```

Appendix 2: STDP-model 2 script

```
# Praat script STDP_v2.praat
# Angelica van Beemdelust
# May-Jun 2022
# last update: 24-06-2022

demoWindowTitle: "STDP-model"
demoShow()
demo Times
demo Font size: 14

# set up
t = 0
dt = 0.001 ; in s
inputDuration = 0.24 ; in s (House, 1961)
radius = 3.0 ; in mm
step = 0
minFiringRate = 3 ; in Hertz
maxFiringRate = 600 ; in Hz
minWeight = 0.001 ; in Hz
maxWeight = 60 ; in Hz
maxExtWeight = 500 ; in Hz
maxExtMeanWeight = 500
excitatoryDecayTime = 0.011 ; in s
memHistDecayTime = 0.020 ; in s (Bi & Poo, 1998, p. 10464)
excitatoryLeak = exp(-dt/excitatoryDecayTime)
memHistLeak = exp(-dt/memHistDecayTime)
inhibitoryDecayTime = 0.020 ; in s

windowWidth = demo Horizontal world coordinates to mm: 1.0
windowHeight = demo Vertical world coordinates to mm: 1.0

@sound
soundDistribution = Create Matrix: "soundDistribution", 0.5,
sound.numberOfAuditoryNodes + 0.5,
sound.numberOfAuditoryNodes, 1.0, 1, 1, 1, 1, 1, 1, ~ 0.0

numberOfMeaningNodes = 50
numberOfExtMeaningNodes = 5
numberOfExternalInputNodes = sound.numberOfAuditoryNodes
numberOfInputNodes = sound.numberOfAuditoryNodes +
numberOfMeaningNodes ; 49
numberOfMiddleNodes = 50
numberOfTopNodes = 20
learningRate = 1

countFiredArray# = zero# (numberOfExtMeaningNodes)
```

```
44  count_active = 0
45  audNode1 = 0
46
47  # Initialization of layers
48  # External layer
49  extInputNodeStopArray# = zero# (numberOfExternalInputNodes)
50  extInputNodeFiredArray# = zero# (numberOfExternalInputNodes)
51  extNodePotentialArray# = zero# (numberOfExternalInputNodes)
52  extMeaningNodeStopArray# = zero# (numberOfExtMeaningNodes)
53  extMeaningNodeFiredArray# = zero# (numberOfExtMeaningNodes)
54  extMeaningNodePotentialArray# = zero#
    (numberOfExtMeaningNodes)
55  y_ext = 15
56  x_ext# = from_to_by# (0.5, 45.5,
    45/(numberOfExternalInputNodes - 1))
57  x_extM# = from_to_by# (55, 95, 40/(numberOfExtMeaningNodes -
    1))
58  weights1_2## = zero## (numberOfExternalInputNodes,
    sound.numberOfAuditoryNodes)
59  weights1m_2m## = zero## (numberOfExtMeaningNodes,
    numberOfMeaningNodes)
60  weights2m_1m## = zero##
    (numberOfMeaningNodes,numberOfExtMeaningNodes)
61  conn1_2## = weights1_2##
62  conn1m_2m## = weights1m_2m##
63
64  # Input layer
65  meaningNodePotentialArray# = zero# (numberOfMeaningNodes)
66  meaningNodeFiredArray# = zero# (numberOfMeaningNodes)
67  meaningNodeStopArray# = zero# (numberOfMeaningNodes)
68  intInputNodeStopArray# = zero# (numberOfInputNodes)
69  intInputNodeFiredArray# = zero# (numberOfInputNodes)
70  intNodePotentialArray# = zero# (numberOfInputNodes)
71  inputMemoryFiringArray# = zero# (numberOfInputNodes)
72  y_bottom = 40
73  x_aud# = from_to_by# (0.5, 45.5,
    45/(sound.numberOfAuditoryNodes - 1))
74  x_meaning# = from_to_by# (50.5, 99.5,
    49/(numberOfMeaningNodes - 1))
75  x_input# = zero# (numberOfInputNodes)
76  @combineAudMeaningPosition
77  weights2_3## = zero## (numberOfInputNodes,
    numberOfMiddleNodes)
78
79  # Middle layer
80  middleNodeFiredArray# = zero# (numberOfMiddleNodes)
81  middleNodePotentialArray# = zero# (numberOfMiddleNodes)
82  middleMemoryFiringArray# = zero# (numberOfMiddleNodes)
83  y_mid = 65
```

```
x_mid# = from_to_by# (0.5, 99.5, 99/(numberOfMiddleNodes - 1))
weights3_2## = zero## (numberOfMiddleNodes,
numberOfInputNodes)
weights3_4## = zero## (numberOfMiddleNodes, numberOfTopNodes)

# Initialize meaning
meaning.morpheme$ [1] = "'a'"
meaning.morpheme$ [2] = "'e'"
meaning.morpheme$ [3] = "'i'"
meaning.morpheme$ [4] = "'o'"
meaning.morpheme$ [5] = "'u'"

# Top layer
topNodeFiredArray# = zero# (numberOfTopNodes)
topNodePotentialArray# = zero# (numberOfTopNodes)
topMemoryFiringArray# = zero# (numberOfTopNodes)
y_top = 90
x_top# = from_to_by# (0.5, 99.5, 99/(numberOfTopNodes - 1))
weights4_3## = zero## (numberOfTopNodes, numberOfMiddleNodes)

@drawAndInitializeWeights
@initalizePotentialArrays

# Draw network area
label start
demo Erase all
demo Select inner viewport: 10, 90, 10, 90
demo Axes: 0, 100, 0, 100
demo Paint rectangle: "silver", 0, 100, 0, 100
t += dt
@writeLabels

@learnAndUpdateWeights
@drawAllWeights
@updatePotentialAndDecay
@fireNodesAndUpdateMem
@drawAllNodes
##@drawSoundDistribution
@writeOutsideLabels
@count

demoPeekInput()
if demoKey$ () = "q"
  exitScript ()
endif
# demo pause and wait for user interaction buttons
if demoKey$ () = "p"
  wait = 1
  while wait = 1
```

```
132     demoWaitForInput()
133     if demoInput (”AEIOU”)
134       clickedVowel = index (”AEIOU”, demoKey$ ())
135       f1_erb = randomGauss (sound.f1_erb# [clickedVowel],
          sound.ambientStdev_erb)
136       f2_erb = randomGauss (sound.f2_erb# [clickedVowel],
          sound.ambientStdev_erb)
137       f1_erb = round(f1_erb)
138       f2_erb = round(f2_erb)
139       @applySound: f1_erb, f2_erb, 1
140       extInputNodeStopArray# [audNode1] = t + inputDuration
141       extInputNodeStopArray# [audNode2] = t + inputDuration
142       extInputNodeFiredArray# [audNode1] = 1
143       extInputNodeFiredArray# [audNode2] = 1
144       count_active = 1
145       appendInfoLine: clickedVowel
146       wait = 0
147     elsif demoInput (”aeiou”)
148       clickedMeaning = index (”aeiou”, demoKey$ ())
149       extMeaningNodeStopArray# [clickedMeaning] = t +
          inputDuration
150       extMeaningNodeFiredArray# [clickedMeaning] = 1
151       wait = 0
152     elsif demoKey$ () = ”1”
153       for ministep to 1000
154         step += 1 ; to track for sound Distribution
155         t += dt
156         @learnAndUpdateWeights
157         @updatePotentialAndDecay
158         @fireNodesAndUpdateMem
159         # set input
160         vowel = randomInteger (1, sound.numberOfVowels)
161         f1_erb = randomGauss (sound.f1_erb# [vowel],
          sound.ambientStdev_erb)
162         f2_erb = randomGauss (sound.f2_erb# [vowel],
          sound.ambientStdev_erb)
163         f1_erb = round(f1_erb)
164         f2_erb = round(f2_erb)
165         @applySound: f1_erb, f2_erb, 1
166         extMeaningNodeStopArray# [vowel] = t + inputDuration
167         extMeaningNodeFiredArray# [vowel] = 1
168         # update over time in 1 learning step
169         while t < extMeaningNodeStopArray# [vowel ]
170           t += dt
171           @learnAndUpdateWeights
172           @updatePotentialAndDecay
173           @fireNodesAndUpdateMem
174         endwhile
175         # optional wait for previous input to decay
```

```
176       # for time to excitatoryDecayTime*1000
177       #  t += dt
178       #  @learnAndUpdateWeights
179       #  @updatePotentialAndDecay
180       #  @fireNodesAndUpdateMem
181       # endfor
182      endfor
183      wait = 0
184    elsif demoKey$ () = "p"
185      wait = 0
186    endif
187  endwhile
188 endif
189 
190 goto start
191 
192 procedure applySound: .f1_erb, .f2_erb,
    .recordSoundDistribution
193   audNode1 = 1 + (.f1_erb - sound.fmin_erb) / sound.erbsPerNode
194   audNode2 = 1 + (.f2_erb - sound.fmin_erb) / sound.erbsPerNode
195    if .recordSoundDistribution
196     select soundDistribution
197     Formula: ~ self + exp (-0.5 * ((col - audNode1) /
        sound.auditorySpreading_nodes) ^ 2) + exp (-0.5 * ((col -
        audNode2) / sound.auditorySpreading_nodes) ^ 2)
198    endif
199 endproc
200 
201 procedure drawSoundDistribution
202   selectObject: soundDistribution
203   demo Yellow
204   demo Select inner viewport: 10, 47, 10, 75
205   demo Line width: 3
206   demo Draw rows: 0.5, sound.numberOfAuditoryNodes + 0.5, 0,
      0, 0, step
207 endproc
208 
209 # procedure draw & initialize all weights
210 procedure drawAndInitializeWeights
211    @createInputWeightMatrix
212   @createMeaningWeightMatrix
213    @initializeWeight: numberOfInputNodes, numberOfMiddleNodes
214    weights2_3## = weightMatrix##
215    @initializeWeight: numberOfMiddleNodes, numberOfInputNodes
216    weights3_2## = weightMatrix##
217    @initializeWeight: numberOfMiddleNodes, numberOfTopNodes
218    weights3_4## = weightMatrix##
219    @initializeWeight: numberOfTopNodes, numberOfMiddleNodes
220    weights4_3## = weightMatrix##
```

```
221  endproc
222
223  # procedure draw all Weights
224  procedure drawAllWeights
225    demo Black
226      @drawInputWeight
227    @drawMeaningWeight
228      @drawWeight: weights2_3##, numberOfInputNodes,
         numberOfMiddleNodes, x_input#, y_bottom, x_mid#, y_mid
229      @drawWeight: weights3_2##, numberOfMiddleNodes,
         numberOfInputNodes, x_mid#, y_mid, x_input#, y_bottom
230      @drawWeight: weights3_4##, numberOfMiddleNodes,
         numberOfTopNodes, x_mid#, y_mid, x_top#, y_top
231      @drawWeight: weights4_3##, numberOfTopNodes,
         numberOfMiddleNodes, x_top#, y_top, x_mid#, y_mid
232
233  endproc
234
235  # procedure draw single node
236  procedure drawNode (.x, .y, .active, .p)
237    demo Black
238    demo Paint circle (mm): if .active then ”red” else ”white”
       fi, .x, .y, .p/100 + 2*radius
239    demo Line width: 2
240    demo Draw circle (mm): .x, .y, .p/100 + 2*radius
241  endproc
242
243  procedure drawAllNodes
244    for nn to numberOfExternalInputNodes
245      @drawNode: x_ext# [nn], y_ext, extInputNodeFiredArray#
       [nn], extNodePotentialArray# [nn]
246    endfor
247
248    for nn to numberOfExtMeaningNodes
249      @drawNode: x_extM# [nn], y_ext, extMeaningNodeFiredArray#
       [nn], extMeaningNodePotentialArray# [nn]
250    endfor
251    for nn to numberOfInputNodes
252      if nn <= sound.numberOfAuditoryNodes
253        @drawNode: x_input# [nn], y_bottom,
         intInputNodeFiredArray# [nn], intNodePotentialArray# [nn]
254      else
255        @drawNode: x_input# [nn], y_bottom,
         intInputNodeFiredArray# [nn], meaningNodePotentialArray#
         [nn - sound.numberOfAuditoryNodes]
256      endif
257    endfor
258
259    for nn to numberOfMiddleNodes
```

```
260     @drawNode: x_mid# [nn], y_mid, middleNodeFiredArray# [nn],
        middleNodePotentialArray# [nn]
261   endfor
262
263   for nn to numberOfTopNodes
264     @drawNode: x_top# [nn], y_top, topNodeFiredArray# [nn],
        topNodePotentialArray# [nn]
265   endfor
266 endproc
267
268 procedure createInputWeightMatrix
269   stDev = 1
270   weights1_2## ~ if abs(col - row) <= 2 then (maxExtWeight /
      (stDev * sqrt(2 * pi))) * exp(-0.5 * (abs(col - row) /
      stDev)^2) else 0 fi
271   conn1_2## ~ if abs(col - row) <= 2 then 1 else 0 fi
272 endproc
273
274 procedure createMeaningWeightMatrix
275   stDev = 2
276   weights1m_2m## ~ if abs((row - 1)* 10 + 5.5 - col) <= 5
      then (maxExtMeanWeight / (stDev * sqrt(2 * pi))) * exp(-0.5
      * (abs((row - 1) * 10 + 5.5 - col)/ stDev)^2) else 0 fi
277   weights2m_1m## ~ if abs((col - 1) * 10 + 5.5 - row) <= 5
      then (maxExtMeanWeight / (stDev * sqrt(2 * pi))) * exp(-0.5
      * (abs((col - 1) * 10 + 5.5 - row)/ stDev)^2) else 0 fi
278   conn1m_2m## ~ if abs((row - 1) * 10 + 5.5 - col) <= 5 then
      1 else 0 fi
279 endproc
280
281 # procedure calculating weight
282 procedure calculateWeight: .d
283    stDev = 1 ; one node = 1 stDev
284    calculatedWeight = (maxExtWeight / (stDev * sqrt(2 * pi)))
       * exp(-0.5 * (.d / stDev)^2)
285 endproc
286
287 # procedure drawing weights ext-aud
288 procedure drawInputWeight
289   for ext to numberOfExternalInputNodes
290     for aud to sound.numberOfAuditoryNodes
291       if conn1_2## [ext, aud] <> 0
292         demo Line width: weights1_2## [ext, aud]/35
293         demo Draw line: x_ext# [ext], y_ext, x_aud# [aud],
            y_bottom
294       endif
295     endfor
296   endfor
297 endproc
```

```
298
299  # procedure drawing weights extMeaning-meaning
300  procedure drawMeaningWeight
301    for ext to numberOfExtMeaningNodes
302      for mean to numberOfMeaningNodes
303        if conn1m_2m## [ext, mean] <> 0
304          demo Line width: weights1m_2m## [ext, mean]/10
305          demo Draw line: x_extM# [ext], y_ext, x_meaning#
             [mean], y_bottom
306        endif
307      endfor
308    endfor
309  endproc
310
311  # procedure drawing rest of weights
312  procedure drawWeight: .matrix##, .nOfi, .nOfj, .x_posi#,
     .y_posi, .x_posj#, .y_posj
313    for i to .nOfi
314      for j to .nOfj
315        weight = .matrix## [i, j]
316        midx = (.x_posi# [i] + .x_posj# [j])/2
317        midy = (.y_posi + .y_posj)/2
318        if weight > 0
319          demo Black
320          demo Line width: .matrix## [i, j]/20
321          demo Draw line: midx, midy, .x_posj# [j], .y_posj
322        elsif weight < 0
323          demo White
324          demo Line width: abs(.matrix## [i, j]/5)
325          demo Draw line: midx, midy, .x_posj# [j], .y_posj
326        endif
327      endfor
328    endfor
329  endproc
330
331  # procedure learning and update weights
332  procedure learnAndUpdateWeights
333    # Input-Mid nodes
334      weights2_3## ~ max(minWeight, min(self +
         inputMemoryFiringArray# [row] * middleNodeFiredArray#
         [col] - intInputNodeFiredArray# [row] *
         middleMemoryFiringArray# [col], maxWeight)) * learningRate
335      weights3_2## ~ max(minWeight, min(self +
         middleMemoryFiringArray# [row] * intInputNodeFiredArray#
         [col] - middleNodeFiredArray# [row] *
         inputMemoryFiringArray# [col], maxWeight)) * learningRate
336
337    # Mid-Top nodes
338      weights3_4## ~ max(minWeight, min(self +
```

```
      middleMemoryFiringArray# [row] * topNodeFiredArray# [col]
      - middleNodeFiredArray# [row] * topMemoryFiringArray#
      [col], maxWeight)) * learningRate
339   weights4_3## ~ max(minWeight, min(self +
      topMemoryFiringArray# [row] * middleNodeFiredArray# [col]
      - topNodeFiredArray# [row] * middleMemoryFiringArray#
      [col],maxWeight)) * learningRate
340 endproc
341
342 # procedure initialize rest of weights
343 procedure initializeWeight: .nOfi, .nOfj
344   weightMatrix## = zero## (.nOfi, .nOfj)
345   weightMatrix## ~ 3
346 endproc
347
348 # procedure combine auditory and meaning nodes into single
    array
349 procedure combineAudMeaningPosition
350   for i to sound.numberOfAuditoryNodes
351     x_input# [i] = x_aud# [i]
352   endfor
353   for j to numberOfMeaningNodes
354     x_input# [sound.numberOfAuditoryNodes + j] = x_meaning# [j]
355   endfor
356 endproc
357
358 # procedure creating potential rate array
359 procedure createPotentialRateArray: .numberOfNodes
360   nodePotentialArray# = zero# (.numberOfNodes)
361   nodePotentialArray# ~ minFiringRate
362 endproc
363
364 # procedure initialize potential rate array
365 procedure initalizePotentialArrays
366     @createPotentialRateArray: numberOfInputNodes
367     intNodePotentialArray# = nodePotentialArray#
368     @createPotentialRateArray: numberOfExtMeaningNodes
369     extMeaningNodePotentialArray# = nodePotentialArray#
370     @createPotentialRateArray: numberOfMiddleNodes
371     middleNodePotentialArray# = nodePotentialArray#
372     @createPotentialRateArray: numberOfTopNodes
373     topNodePotentialArray# = nodePotentialArray#
374 endproc
375
376 # update potential rate + decay at beginning of cycle
377 procedure updatePotentialAndDecay
378   # update ext meaning nodes
379   meaningNodeFiredArray# ~ intInputNodeFiredArray# [col +
      sound.numberOfAuditoryNodes]
```

```
380    sumOfBottomNode# = mul# (meaningNodeFiredArray#,
       weights2m_1m##)
381    extMeaningNodePotentialArray# ~ if extMeaningNodeFiredArray#
       [col] then minFiringRate else (self + sumOfBottomNode#
       [col]) * excitatoryLeak fi
382
383    # update input nodes
384    externalInputForBottom# = mul# (mul#
       (extInputNodeFiredArray#, weights1_2##), conn1_2##)
385    middleInputForBottom# = mul# (middleNodeFiredArray#,
       weights3_2##)
386    sumOfExtMeaning# = mul# (extMeaningNodeFiredArray#,
       weights1m_2m##)
387
388    intNodePotentialArray# ~ if col <=
       sound.numberOfAuditoryNodes
389    ... then (self + externalInputForBottom# [col] +
       middleInputForBottom# [col]) * excitatoryLeak
390    ... else (self) fi
391
392    meaningNodePotentialArray# ~ if intInputNodeFiredArray# [col
       + sound.numberOfAuditoryNodes] then minFiringRate else (self
       + middleInputForBottom# [col + sound.numberOfAuditoryNodes]
       + sumOfExtMeaning# [col]) * excitatoryLeak fi
393
394    intNodePotentialArray# ~ if col >
       sound.numberOfAuditoryNodes then meaningNodePotentialArray#
       [col - sound.numberOfAuditoryNodes] else self fi
395    intNodePotentialArray# ~ if intInputNodeFiredArray# [col]
       then minFiringRate else self fi
396
397    # update middle nodes
398    sumOfMiddleNode# = mul# (intInputNodeFiredArray#,
       weights2_3##) + mul# (topNodeFiredArray#, weights4_3##)
399
400    middleNodePotentialArray# ~ if middleNodeFiredArray# [col]
       then minFiringRate else (self + sumOfMiddleNode# [col]) *
       excitatoryLeak fi
401
402    # update top nodes
403    sumOfTopNode# = mul# (middleNodeFiredArray#, weights3_4##)
404
405    topNodePotentialArray# ~ if topNodeFiredArray# [col] then
       minFiringRate else (self + sumOfTopNode# [col]) *
       excitatoryLeak fi
406  endproc
407
408  # check if fired and update mem nodes
409  procedure fireNodesAndUpdateMem
```

```
410   extMeaningNodeFiredArray# ~ if extMeaningNodeStopArray#
      [col] < t then randomBernoulli((minFiringRate +
      (maxFiringRate - minFiringRate) *
      tanh(extMeaningNodePotentialArray# [col] / maxFiringRate) *
      (1 - self)) * dt) else self fi
411
412   # input nodes
413   extInputNodeFiredArray# ~ if extInputNodeStopArray# [col] <
      t then 0 else self fi
414
415   intInputNodeFiredArray# ~ randomBernoulli((minFiringRate +
      (maxFiringRate - minFiringRate) *
      tanh(intNodePotentialArray# [col] / maxFiringRate) * (1 -
      self)) * dt)
416
417   inputMemoryFiringArray# ~ if intInputNodeFiredArray# [col]
      then 1 else self * memHistLeak fi
418
419   # middle nodes
420   middleNodeFiredArray# ~ randomBernoulli((minFiringRate +
      (maxFiringRate - minFiringRate) *
      tanh(middleNodePotentialArray# [col]/maxFiringRate) * (1 -
      self)) * dt)
421
422   middleMemoryFiringArray# ~ if middleNodeFiredArray# [col]
      then 1 else self * memHistLeak fi
423
424   # top nodes
425   topNodeFiredArray# ~ randomBernoulli((minFiringRate +
      (maxFiringRate - minFiringRate) *
      tanh(topNodePotentialArray# [col]/maxFiringRate) * (1 -
      self)) * dt)
426
427   topMemoryFiringArray# ~ if topNodeFiredArray# [col] then 1
      else self * memHistLeak fi
428 endproc
429
430 procedure writeLabels
431   demo Black
432   demo Text special: 0, "left", 7, "bottom", "Times", 40, "0",
      "["
433   demo Text special: 0.16 * numberOfExternalInputNodes,
      "centre", 7, "bottom", "Times", 40, "0", "5"
434   demo Text special: 0.32 * numberOfExternalInputNodes,
      "centre", 7, "bottom", "Times", 40, "0", "10"
435   demo Text special: 0.475 * numberOfExternalInputNodes,
      "centre", 7, "bottom", "Times", 40, "0", "15"
436   demo Text special: 0.63 * numberOfExternalInputNodes,
      "centre", 7, "bottom", "Times", 40, "0", "20"
```

```
437   demo Text special: 0.79 * numberOfExternalInputNodes,
      "centre", 7, "bottom", "Times", 40, "0", "25 "
438   demo Text special: 0.95 * numberOfExternalInputNodes,
      "right", 7, "bottom", "Times", 40, "0", " ERB]"
439
440   demo Text special: x_extM# [1], "centre", 7, "bottom",
      "Times", 40, "0", "'a"
441   demo Text special: x_extM# [2], "centre", 7, "bottom",
      "Times", 40, "0", "'e"
442   demo Text special: x_extM# [3], "centre", 7, "bottom",
      "Times", 40, "0", "'i"
443   demo Text special: x_extM# [4], "centre", 7, "bottom",
      "Times", 40, "0", "'o"
444   demo Text special: x_extM# [5], "centre", 7, "bottom",
      "Times", 40, "0", "'u"
445 endproc
446
447 procedure writeOutsideLabels
448   demo Select inner viewport: 0, 100, 0, 100
449   demo Axes: 0, 100, 0, 100
450   demo Black
451   demo Text special: 7, "left", 80, "bottom", "Times", 40,
      "0", "z_n"
452   demo Text special: 7, "left", 60, "bottom", "Times", 40,
      "0", "y_m"
453   demo Text special: 6, "left", 40, "bottom", "Times", 40,
      "0", "x_l"
454   demo Text special: 6, "left", 40, "bottom", "Times", 40,
      "0", "x^^aud^"
455   demo Text special: 6, "left", 20, "bottom", "Times", 40,
      "0", "v_k"
456   demo Text special: 6, "left", 20, "bottom", "Times", 40,
      "0", "v^^aud^"
457   demo Text special: 90, "left", 40, "bottom", "Times", 40,
      "0", "x_l"
458   demo Text special: 90, "left", 40, "bottom", "Times", 40,
      "0", "x^^mean^"
459   demo Text special: 90, "left", 20, "bottom", "Times", 40,
      "0", "v^^mean^"
460   demo Text special: 90, "left", 20, "bottom", "Times", 40,
      "0", "v_k"
461   demo Text special: 4, "left", 73, "bottom", "Times", 30,
      "0", "q_m_n"
462   demo Text special: 4, "left", 73, "bottom", "Times", 30,
      "0", "q^^up^"
463   demo Text special: 4, "left", 67, "bottom", "Times", 30,
      "0", "q_m_n"
464   demo Text special: 4, "left", 67, "bottom", "Times", 30,
      "0", "q^^down^"
```

```
465   demo Text special: 4, "left", 53, "bottom", "Times", 30,
      "0", "r_l_m"
466   demo Text special: 4, "left", 53, "bottom", "Times", 30,
      "0", "r^^up^"
467   demo Text special: 4, "left", 47, "bottom", "Times", 30,
      "0", "r_l_m"
468   demo Text special: 4, "left", 47, "bottom", "Times", 30,
      "0", "r^^down^"
469   demo Text special: 4, "left", 30, "bottom", "Times", 30,
      "0", "s_k_l"
470   demo Text special: 4, "left", 30, "bottom", "Times", 30,
      "0", "s^^aud^"
471   demo Text special: 90, "left", 30, "bottom", "Times", 30,
      "0", "s_k_l"
472   demo Text special: 90, "left", 30, "bottom", "Times", 30,
      "0", "s^^mean^"
473 endproc
474
475 procedure count
476   if count_active = 1
477     if extInputNodeFiredArray# [audNode1]
478       countFiredArray# ~ extMeaningNodeFiredArray# [col] +
          countFiredArray# [col]
479     else
480       count_active = 0
481       appendInfoLine: countFiredArray#
482       countFiredArray# ~ 0
483     endif
484   endif
485 endproc
486
487 include demo.praatinclude
488 include sound.praatinclude
```

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
```

```
42   demo.textY -= 12
43  endproc
44
45  procedure demo.therefore .text$
46   demo Text... 10-2.5 centre demo.textY half □
47   .width = demo Text width (wc)... '.text$'
48   if .width < 85
49    demo Text... 10 left demo.textY half '.text$'
50   else
51    demo Text special... 10 left demo.textY half 'demo.font$'
      demo.fontSize*85/.width 0 '.text$'
52   endif
53   demo.textY -= 12
54  endproc
55
56  procedure demo.text .text$
57   demo.textY += 4
58   .width = demo Text width (wc)... '.text$'
59   if .width < 85
60    demo Text... 10 left demo.textY half '.text$'
61   else
62    demo Text special... 10 left demo.textY half 'demo.font$'
      demo.fontSize*85/.width 0 '.text$'
63   endif
64   demo.textY -= 12
65  endproc
66
67  procedure demo.reference .text$
68   demo.textY += 5
69   demo Text special... 98 right demo.textY half 'demo.font$'
     demo.fontSize/1.5 0 '.text$'
70   demo.textY -= 9
71  endproc
72
73  procedure demo.source .text$
74   demo Text special... 2 left 2 bottom Times demo.fontSize/1.5
     0 '.text$'
75  endproc
76
77  procedure demo.button .x1 .x2 .y .text$
78   demo Paint rounded rectangle... 'demo.buttonColour$' .x1 .x2
     .y-4 .y+4 3
79   .width = demo Text width (wc)... '.text$'
80   if .width < 0.9 * (.x2 - .x1)
81    demo Text... (.x1+.x2)/2 centre .y half '.text$'
82   else
83    demo Text special... (.x1+.x2)/2 centre .y half
      'demo.font$' demo.fontSize*0.9*(.x2-.x1)/.width 0 '.text$'
84   endif
```

```
85  endproc
86
87  procedure demo.wait .duration
88    Create Sound from formula... silence mono 0 .duration 44100 0
89    Play
90    Remove
91  endproc
```

Appendix 4: Sound script

```
# 2019-CJL/sound.praatinclude
# Paul Boersma 2020-01-04

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 = 49
  .fmin_erb = 4.0
  .fmax_erb = 28.0
  .erbsPerNode = (.fmax_erb - .fmin_erb) /
  (.numberOfAuditoryNodes - 1)
  .auditorySpreading_nodes = .auditorySpreading_erb /
  .erbsPerNode
endproc
```